PIANO DI LAVORO DI EDUCAZIONE CIVICA CLASSE 5 ^ A INF

Classe 5°

UNITÀ DI APPRENDIMENTO	
Denominazione	**La pace, la guerra e i rapporti internazionali**
Compito - prodotto	Produzione di un glossario inerente al campo semantico Produzioni di testi descrittivi e/o narrativi. Relazioni ed elaborati Rappresentazioni grafiche e schemi
Obiettivi e competenze mirate	• Conoscere l'organizzazione costituzionale ed amministrativa del nostro Paese per rispondere ai propri doveri di cittadino ed esercitare con consapevolezza i propri diritti politici a livello territoriale e nazionale. • Conoscere i valori che ispirano gli ordinamenti comunitari e internazionali, nonché i loro compiti e funzioni essenziali • Essere consapevoli del valore e delle regole della vita democratica anche attraverso l'approfondimento degli elementi fondamentali del diritto che la regolano, con particolare riferimento al diritto del lavoro. • Esercitare correttamente le modalità di rappresentanza, di delega, di rispetto degli impegni assunti e fatti propri all'interno di diversi ambiti istituzionali e sociali. • Partecipare al dibattito culturale. • Cogliere la complessità dei problemi esistenziali, morali, politici, sociali, economici e scientifici e formulare risposte personali argomentate. • Prendere coscienza delle situazioni e delle forme del disagio giovanile ed adulto nella società contemporanea e comportarsi in modo da promuovere il benessere fisico, psicologico, morale e sociale. • Rispettare l'ambiente, curarlo, conservarlo, migliorarlo, assumendo il principio di responsabilità. • Adottare i comportamenti più adeguati per la tutela della sicurezza propria, degli altri e dell'ambiente in cui si vive, in condizioni ordinarie o straordinarie di pericolo, curando l'acquisizione di elementi formativi di base in materia di primo intervento e protezione civile. • Perseguire con ogni mezzo e in ogni contesto il principio di legalità e di solidarietà dell'azione individuale e sociale, promuovendo principi, valori e abiti di contrasto alla criminalità organizzata e alle mafie. • Esercitare i principi della cittadinanza digitale, con competenza e coerenza rispetto al sistema integrato di valori che regolano la vita democratica. • Compiere le scelte di partecipazione alla vita pubblica e di cittadinanza coerentemente agli obiettivi di sostenibilità sanciti a livello comunitario attraverso l'Agenda 2030 per lo sviluppo sostenibile. • Operare a favore dello sviluppo eco-sostenibile e della tutela delle identità e delle eccellenze produttive del Paese. • Rispettare e valorizzare il patrimonio culturale e dei beni pubblici comuni. • Sicurezza nei laboratori Le **competenze di cittadinanza** • Imparare ad imparare: ogni giovane deve acquisire un proprio metodo di studio e di lavoro. • Progettare: ogni giovane deve essere capace di utilizzare le conoscenze apprese per darsi obiettivi significativi e realistici. Questo richiede la

	capacità di individuare priorità, valutare i vincoli e le possibilità esistenti, definire strategie di azione, fare progetti e verificarne i risultati. • Comunicare: ogni giovane deve poter comprendere messaggi di genere e complessità diversi nelle varie forme comunicative e deve poter comunicare in modo efficace utilizzando i diversi linguaggi. • Collaborare e partecipare: ogni giovane deve saper interagire con gli altri comprendendone i diversi punti di vista. • Agire in modo autonomo e responsabile: ogni giovane deve saper riconoscere il valore delle regole e della responsabilità personale. • Risolvere problemi: ogni giovane deve saper affrontare situazioni problematiche e saper contribuire a risolverle. • Individuare collegamenti e relazioni: ogni giovane deve possedere strumenti che gli permettano di affrontare la complessità del vivere nella società globale del nostro tempo. • Acquisire ed interpretare l'informazione: ogni giovane deve poter acquisire ed interpretare criticamente l'informazione ricevuta valutandone l'attendibilità e l'utilità, distinguendo fatti e opinioni.
Utenti destinatari	Alunni della classe quinta dell'Istituto
Prerequisiti	Competenze linguistiche e lessicali di base Competenze informatiche di base
Tempi	Ottobre-Aprile
Materie coinvolte	Italiano, Storia, Inglese, Matematica, Educazione fisica + MATERIE DI INDIRIZZO
Metodologie	Brainstorming: dalle idee, alle parole, ai segni grafici sulla lavagna Apprendimento collaborativo Lavori di gruppo Lezione frontale Flipped classroom Approfondimenti personali Apprendimento esperienziale Debriefing
Risorse	Collaborazioni fra i docenti Testimonianze degli stessi studenti
Strumenti	LIM Libro di testo Film Quotidiani Materiali multimediali
Valutazione	Vedasi griglia di valutazione

PIANO DI LAVORO DI EDUCAZIONE CIVICA A.S. 2020/2021

CLASSE V AInf

1^ QUADRIMESTRE

MATERIA	ARGOMENTO	ORE
Italiano	Verga/Pirandello: il lavoro minorile, lavoro dei minorati psicofisici; articoli Costituzione connessi al lavoro;	5
Informatica	Lezioni inerenti la sicurezza nei laboratori	3
Inglese	Agenda 2030 and the 17 goals	2
Sistemi e reti	I diritti e i doveri connessi al lavoro subordinato.	2
Tepsi	La quinta dimensione della conflittualità. La rilevanza strategica del cyberspace e i rischi di guerra cibernetica. L'avvento delle reti di quinta generazione comporta nuove sfide sul fronte della cybersicurezza.	2
Ed. fisica	Il servizio di leva: ieri, oggi, domani. Protezione civile. L'arma come possibilità di lavoro e di carriera.	2

2^ QUADRIMESTRE

MATERIA	ARGOMENTO	ORE
Storia	l'Unione Europea come spazio di pace: a. la storia dell'integrazione europea b. i trattati europei dal 1947 ad oggi c. le istituzioni europee d. il sistema economico e monetario europeo e. principali tradizioni culturali in Europa; *Articoli della Costituzione*	4
Informatica	Come si apre un conto corrente; a cosa serve; cosa è un iban; come si scrive un assegno; come si effettuano pagamenti sicuri.	2
Inglese	Agenda 2030 and the 17 goals	2
Sistemi	Il valore del titolo di studio in Italia ed Europa.	2
Gestione	I diversi tipi di società e le diverse responsabilità dei soci;	2
Matematica	Distribuzione gaussiana di probabilità	3
Ed. fisica	Cenni di 1° soccorso, massaggio cardiaco, uso del defibrillatore.	2

TOTALE GENERALE **33 ORE**

GRIGLIA DI VALUTAZIONE

INDICATORI	DESCRITTORI
10	Conoscenza approfondita, organica ed interdisciplinare degli argomenti. Esposizione scorrevole, chiara ed autonoma con lessico appropriato e usando fonti soggettive. Interesse spiccato e partecipazione attiva. Capacità di rielaborazione con apporti personali. Metodo di lavoro produttivo.
9	Conoscenza approfondita ed organica dei contenuti. Esposizione personale e sicura con utilizzo appropriato del lessico specifico e degli strumenti. Attenzione e partecipazione attiva. Capacità di rielaborare le conoscenze e di cogliere i collegamenti disciplinari. Metodo di studio proficuo.
8	Conoscenza completa ed organica dei contenuti. Esposizione sicura con buon uso del lessico e dei linguaggi specifici. Impegno e partecipazione positivi. Usa con autonomia le conoscenze e le informazioni. Metodo di studio efficace.
7	Complessiva conoscenza dei contenuti. Esposizione ed uso adeguati del lessico e degli strumenti. Nella rielaborazione evidenzia i concetti e gli elementi importanti. Metodo di lavoro e di studio abbastanza efficaci.
6	Conoscenze parziali dei contenuti. Comprensione elementare dei concetti. Esposizione abbastanza corretta ma con poca padronanza del lessico specifico e degli strumenti. Metodo di lavoro poco efficace.
5	Carenze di base. Difficoltà a riconoscere gli elementi fondamentali degli argomenti trattati. Esposizione imprecisa e confusa. Difficoltà a memorizzare, ad eseguire procedure e a applicare le informazioni. Metodo ed uso degli strumenti poco adeguati.
≥ 4	Scarse conoscenze e gravi lacune di base. Lavori e verifiche parziali o non eseguite.

N 1

Testo breve

Questo racconto è lo studio sincero e spassionato del come probabilmente devono nascere e svilupparsi nelle più umili condizioni, le prime irrequietudini pel benessere; e quale perturbazione debba arrecare in una famigliuola vissuta fino allora relativamente felice, la vaga bramosia dell'ignoto, l'accorgersi che non si sta bene, o che si potrebbe star meglio. Il movente dell'attività umana che produce la fiumana del progresso è preso qui alle sue sorgenti, nelle proporzioni più modeste e materiali. Il meccanismo delle passioni che la determinano in quelle basse sfere è meno complicato, e potrà quindi osservarsi con maggior precisione. Basta lasciare al quadro le sue tinte schiette e tranquille, e il suo disegno semplice. Man mano che cotesta ricerca del meglio di cui l'uomo è travagliato cresce e si dilata, tende anche ad elevarsi, e segue il suo moto ascendente nelle classi sociali. Nei Malavoglia non è ancora che la lotta pei bisogni materiali. Soddisfatti questi, la ricerca diviene avidità di ricchezze, e si incarnerà in un tipo borghese, Mastro-don Gesualdo, incorniciato nel quadro ancora ristretto di una piccola città di provincia, ma del quale i colori cominceranno ad essere più vivaci, e il disegno a farsi più ampio e variato. Poi diventerà vanità aristocratica nella Duchessa di Leyra; e ambizione nell'Onorevole Scipioni, per arrivare all'Uomo di lusso, il quale riunisce tutte coteste bramosie, tutte coteste vanità, tutte coteste ambizioni, per comprenderle e soffrirne, se le sente nel sangue, e ne è consunto.

Da : Prefazione al romanzo *I Malavoglia* di G. Verga

N 2

Testo breve

Un tempo i Malavoglia erano stati numerosi come i sassi della strada vecchia di Trezza; ce n'erano persino ad Ognina, e ad Aci Castello, tutti buona e brava gente di mare, proprio all'opposto di quel che sembrava dal nomignolo, come dev'essere. Veramente nel libro della parrocchia si chiamavano Toscano, ma questo non voleva dir nulla, poiché da che il mondo era mondo, all'Ognina, a Trezza e ad Aci Castello, li avevano sempre conosciuti per Malavoglia, di padre in figlio, che avevano sempre avuto delle barche sull'acqua, e delle tegole al sole. Adesso a Trezza non rimanevano che i Malavoglia di padron 'Ntoni, quelli della casa del nespolo, e della Provvidenza ch'era ammarrata sul greto, sotto il lavatoio, accanto alla Concetta dello zio Cola, e alla paranza di padron Fortunato Cipolla. Le burrasche che avevano disperso di qua e di là gli altri Malavoglia, erano passate senza far gran danno sulla casa del Nespolo e sulla barca ammarrata sotto il lavatoio; e padron 'Ntoni, per spiegare il miracolo, soleva dire, mostrando il pugno chiuso – un pugno che sembrava fatto di legno di noce – Per menare il remo bisogna che le cinque dita s'aiutino l'un l'altro. Diceva pure: – Gli uomini son fatti come le dita della mano: il dito grosso deve far da dito grosso, e il dito piccolo deve far da dito piccolo.

Da : Cap1 , *I Malavoglia* di G. Verga

N 3

Testo breve

Così stette un gran pezzo pensando a tante cose, guardando il paese nero, e ascoltando il mare che gli brontolava lì sotto. E ci stette fin quando cominciarono ad udirsi certi rumori ch'ei conosceva, e delle voci che si chiamavano dietro gli usci, e sbatter d'imposte, e dei passi per le strade buie. Sulla riva, in fondo alla piazza, cominciavano a formicolare dei lumi. Egli levò il capo a guardare i Tre Re che luccicavano, e la Puddara che annunziava l'alba, come l'aveva vista tante volte. Allora tornò a chinare il capo sul petto, e a pensare a tutta la sua storia. A poco a poco il mare cominciò a farsi bianco, e i Tre Re ad impallidire, e le case spuntavano ad una ad una nelle vie scure, cogli usci chiusi, che si conoscevano tutte, e solo davanti alla bottega di Pizzuto c'era il lumicino, e Rocco Spatu colle mani nelle tasche che tossiva e sputacchiava. – Fra poco lo zio Santoro aprirà la porta, pensò 'Ntoni, e si accoccolerà sull'uscio a cominciare la sua giornata anche lui. – Tornò a guardare il mare, che s'era fatto amaranto, tutto seminato di barche che avevano cominciato la loro giornata anche loro, riprese la sua sporta e disse: – Ora è tempo d'andarmene, perché fra poco comincerà a passar gente. Ma il primo di tutti a cominciar la sua giornata è stato Rocco Spatu.

Da : Ultima pagina de , *I Malavoglia* di G. Verga

N 4

Testo breve

Di una cosa sola gli doleva, che cominciasse a farsi vecchio, e la terra doveva lasciarla là dov'era. Questa è una ingiustizia di Dio, che dopo di essersi logorata la vita ad acquistare della roba, quando arrivate ad averla, che ne vorreste ancora, dovete lasciarla! E stava delle ore seduto sul corbello, col mento nelle mani, a guardare le sue vigne che gli verdeggiavano sotto gli occhi, e i campi che ondeggiavano di spighe come un mare, e gli oliveti che velavano la montagna come una nebbia, e se un ragazzo seminudo gli passava dinanzi, curvo sotto il peso come un asino stanco, gli lanciava il suo bastone fra le gambe, per invidia, e borbottava: - Guardate chi ha i giorni lunghi! costui che non ha niente! -

Sicché quando gli dissero che era tempo di lasciare la sua roba, per pensare all'anima, uscì nel cortile come un pazzo, barcollando, e andava ammazzando a colpi di bastone le sue anitre e i suoi tacchini, e strillava: - Roba mia, vientene con me! –

Da : Ultima pagina novella *Mazzarò* di G. Verga

N 5

Testo poetico

X Agosto

San Lorenzo, io lo so perchè tanto
di stelle per l'aria tranquilla
arde e cade, perchè sì gran pianto
nel concavo cielo sfavilla.

Ritornava una rondine al tetto:
l'uccisero: cadde tra spini:
ella aveva nel becco un insetto:
la cena de' suoi rondinini.

Ora è là, come in croce, che tende
quel verme a quel cielo lontano;
e il suo nido è nell'ombra, che attende,
che pigola sempre più piano.

Anche un uomo tornava al suo nido:
l'uccisero: disse: Perdono;
e restò negli aperti occhi un grido:
portava due bambole, in dono...
Ora là, nella casa romita,
lo aspettano, aspettano, in vano:
egli immobile, attonito, addita
le bambole al cielo lontano.

E tu, Cielo, dall'alto dei mondi
sereni, infinito, immortale,
oh! d'un pianto di stelle lo inondi
quest'atomo opaco del Male!

Da Myricae, sez "Elegie" Pascoli

N 6

Testo breve

È dentro noi un fanciullino. Quando la nostra età è tuttavia tenera, egli confonde la sua voce con la nostra, e dei due fanciulli che ruzzano e contendono tra loro, e, insieme sempre, temono sperano godono piangono, si sente un palpito solo, uno strillare e un guaire solo. Ma quindi noi cresciamo, ed egli resta piccolo; noi accendiamo negli occhi un nuovo desiderare, ed egli vi tiene fissa la sua antica serena maraviglia; noi ingrossiamo e arrugginiamo la voce, ed egli fa sentire tuttavia e sempre il suo tinnulo squillo come di campanello. Il quale tintinnio segreto noi non udiamo distinto nell'età giovanile forse così come nella più matura, perché in quella occupati a litigare e perorare la causa della nostra vita, meno badiamo a quell'angolo d'anima d'onde esso risuona. E anche, egli, l'invisibile fanciullo, si perita vicino al giovane più che accanto all'uomo fatto e al vecchio, ché più dissimile a sé vede quello che questi.

Tratto da Pascoli, *Il Fanciullino,*

N 7

Temporale

Un bubbolìo lontano. . .

Rosseggia l'orizzonte,
come affocato, a mare:
nero di pece, a monte,
stracci di nubi chiare:
tra il nero un casolare:
un'ala di gabbiano.

Il lampo

E cielo e terra si mostrò qual era:

la terra ansante, livida, in sussulto;
il cielo ingombro, tragico, disfatto:
bianca bianca nel tacito tumulto
una casa apparì sparì d'un tratto,
come un occhio, che, largo, esterrefatto,
s'aprì si chiuse, nella notte nera.

Il tuono

E nella notte nera come il nulla,
a un tratto, col fragor d'arduo dirupo
che frana, il tuono rimbombò di schianto:
rimbombò, rimbalzò, rotolò cupo,
e tacque, e poi rimareggiò rinfranto,
e poi vanì. Soave allora un canto
s'udì di madre, e il moto di una culla.

Da Myricae, sez "Elegie" Pascoli

N 8

Testo breve

Il padre gli aveva dato, tra le altre, questa massima fondamentale: «Bisogna fare la propria vita, come si fa un'opera d'arte. Bisogna che la vita d'un uomo d'intelletto sia opera di lui. La superiorità vera è tutta qui.» Anche, il padre ammoniva: «Bisogna conservare ad ogni costo intiera la libertà, fin nell'ebrezza. La regola dell'uomo d'intelletto, eccola: – Habere, non haberi.» Anche, diceva: «Il rimpianto è il vano pascolo d'uno spirito disoccupato. Bisogna sopra tutto evitare il rimpianto occupando sempre lo spirito con nuove sensazioni e con nuove imaginazioni.» Ma queste massime volontarie, che per l'ambiguità loro potevano anche essere interpretate come alti criterii morali, cadevano appunto in una natura involontaria, in un uomo, cioè, la cui potenza volitiva era debolissima.

Da D'Annunzio, *Il Piacere,* libro 1 , cap 2

N 9

Testo poetico

Piove su le tue ciglia nere
sìche par tu pianga
ma di piacere; non bianca
ma quasi fatta virente,
par da scorza tu esca.
E tutta la vita è in noi fresca
aulente,
il cuor nel petto è come pesca
intatta,
tra le pàlpebre gli occhi
son come polle tra l'erbe,
i denti negli alvèoli
con come mandorle acerbe.

E andiam di fratta in fratta,
or congiunti or disciolti
(e il verde vigor rude
ci allaccia i mallèoli
c'intrica i ginocchi)
chi sa dove, chi sa dove!
E piove su i nostri vólti
silvani,
piove su le nostre mani
ignude,
su i nostri vestimenti
leggieri,
su i freschi pensieri
che l'anima schiude
novella,
su la favola bella
che ieri
m'illuse, che oggi t'illude,
o Ermione.

Da : D'Annunzio , *Libro delle laudi , seconda sezione, La pioggia nel Pineto.*

N 10

Testo breve

Prendete un giornale.
Prendete delle forbici.
Scegliete nel giornale un articolo della lunghezza che desiderate per la vostra poesia.

Ritagliate l'articolo.
Ritagliate poi con cura ognuna delle parole che compongono l'articolo e mettete le parole in un sacchetto.

Agitate dolcemente.
Estraete le parole una dopo l'altra, disponendole nell'ordine in cui sono uscite dal sacchetto.

Copiate scrupolosamente.
La poesia vi somiglierà.
Ed eccovi diventato uno scrittore infinitamente originale e di incantevole sensibilità, benché incompresa dal volgo.

Da : Manifesto dal DADAISMO

N 11

Testo breve

Io sono il dottore di cui in questa novella si parla talvolta con parole poco lusinghiere. Chi di psico-analisi s'intende, sa dove piazzare l'antipatia che il paziente mi dedica2 . Di psico-analisi non parlerò perché qui entro se ne parla già a sufficienza. Debbo scusarmi di aver indotto il mio paziente a scrivere la sua autobiografia; gli studiosi di psico-analisi arricceranno il naso a tanta novità3 . Ma egli era vecchio ed io sperai che in tale rievocazione il suo passato si rinverdisse, che l'autobiografia fosse un buon preludio alla psico-analisi. Oggi ancora la mia idea mi pare buona perché mi ha dato dei risultati insperati, che sarebbero stati maggiori se il malato sul più bello non si fosse sottratto alla cura truffandomi del frutto della mia lunga paziente analisi di queste memorie. Le pubblico per vendetta e spero gli dispiaccia. Sappia però ch'io sono pronto di dividere con lui i lauti onorarii che ricaverò da questa pubblicazione a patto egli riprenda la cura. Sembrava tanto curioso di se stesso! Se sapesse quante sorprese potrebbero risultargli dal commento delle tante verità e bugie ch'egli ha qui accumulate!... Dottor S.

Da: *La Coscienza di Zeno , Prefazione*

N 12

Testo breve

Ma l'occhialuto uomo, invece, inventa gli ordigni fuori del suo corpo e se c'è stata salute e nobiltà in chi li inventò, quasi sempre manca in chi li usa. Gli ordigni si comperano, si vendono e si rubano e l'uomo diventa sempre più furbo e più debole. Anzi si capisce che la sua furbizia cresce in proporzione della sua debolezza. I primi suoi ordigni parevano prolungazioni del suo braccio e non potevano essere efficaci che per la forza dello stesso, ma, oramai, l'ordigno non ha più alcuna relazione con l'arto. Ed è l'ordigno che crea la malattia con l'abbandono della legge che fu su tutta la terra la creatrice. La legge del più forte sparì e perdemmo la selezione salutare. Altro che psico-analisi ci vorrebbe: sotto la legge del possessore del maggior numero di ordigni prospereranno malattie e ammalati. Forse traverso una catastrofe inaudita prodotta dagli ordigni ritorneremo alla salute. Quando i gas velenosi non basteranno più, un uomo fatto come tutti gli altri, nel segreto di una stanza di questo mondo, inventerà un esplosivo incomparabile, in confronto al quale gli esplosivi attualmente esistenti saranno considerati quasi innocui giocattoli. Ed un altro uomo fatto anche lui come tutti gli altri, ma degli altri un po' più ammalato, ruberà tale esplosivo e s'arrampicherà al centro della terra per porlo nel punto ove il suo effetto potrà essere il massimo. Ci sarà un'esplosione enorme che nessuno udrà e la terra ritornata alla forma di nebulosa errerà nei cieli priva di parassiti e di malattie.

Da: *La Coscienza di Zeno , Una catastrofe inaudita , cap 8, "Psico-Analisi"*

N 13

Testo breve

Vedo una vecchia signora, coi capelli ritinti, tutti unti non si sa di quale orribile manteca, e poi tutta goffamente imbellettata e parata d'abiti giovanili. Mi metto a ridere. Avverto che quella vecchia signora è il contrario di ciò che una vecchia rispettabile signora dovrebbe essere. Posso così, a prima giunta e superficialmente, arrestarmi a questa impressione comica. Il comico è appunto un avvertimento del contrario. Ma se ora interviene in me la riflessione, e mi suggerisce che quella vecchia signora non prova forse nessun piacere a pararsi così come un pappagallo, ma che forse ne soffre e lo fa soltanto perché pietosamente s'inganna che, parata così, nascondendo così le rughe e la canizie, riesca a trattenere a sé l'amore del marito molto più giovane di lei, ecco che io non posso più riderne come prima, perché appunto la riflessione, lavorando in me, mi ha fatto andar oltre a quel primo avvertimento, o piuttosto, più addentro: da quel primo avvertimento del contrario mi ha fatto passare a questo sentimento del contrario. Ed è tutta qui la differenza tra il comico e l'umoristico.

Da Pirandello: *Saggio sull'Umorismo , parte seconda*

N 14

Testo breve

– E poi? Me lo metto come titolo nei biglietti da visita. Signor giudice, mi hanno assassinato. Lavoravo. Mi hanno fatto cacciar via dal banco dov'ero scritturale, con la scusa che, essendoci io, nessuno più veniva a far debiti e pegni; mi hanno buttato in mezzo a una strada, con la moglie paralitica da tre anni e due ragazze nubili, di cui nessuno vorrà più sapere, perché sono figlie mie; viviamo del soccorso che ci manda da Napoli un mio figliuolo, il quale ha famiglia anche lui, quattro bambini, e non può fare a lungo questo sacrifizio per noi. Signor giudice, non mi resta altro che di mettermi a fare la professione dello jettatore! Mi sono parato così, con questi occhiali, con quest'abito; mi sono lasciato crescere la barba; e ora aspetto la patente per entrare in campo! Lei mi domanda come? Me lo domanda perché, le ripeto, lei è un mio nemico! – Io? – Sissignore. Perché mostra di non credere alla mia potenza! Ma per fortuna ci credono gli altri, sa? Tutti, tutti ci credono! E ci son tante case da giuoco in questo paese! Basterà che io mi presenti; non ci sarà bisogno di dir nulla. Mi pagheranno per farmi andar via! Mi metterò a ronzare attorno a tutte le fabbriche; mi pianterò innanzi a tutte le botteghe; e tutti, tutti mi pagheranno la tassa, lei dice dell'ignoranza? io dico la tassa della salute! Perché, signor giudice, ho accumulato tanta bile e tanto odio, io, contro tutta questa schifosa umanità, che veramente credo d'avere ormai in questi occhi la potenza di far crollare dalle fondamenta una intera città!

Da Pirandello: *Novello per un anno, La patente.*

N 15

Chi venne a riferirmele insieme con la notizia dell'improvvisa alienazione mentale rimase però sconcertato, non notando in me, non che meraviglia, ma neppur una lieve sorpresa. Difatti io accolsi in silenzio la notizia. Il mio silenzio era pieno di dolore. Tentennai il capo, con gli angoli della bocca contratti in giù, amaramente, e dissi: – Belluca, signori, non è impazzito. State sicuri che non è impazzito. Qualche cosa dev'essergli accaduta; ma naturalissima. Nessuno se la può spiegare, perché nessuno sa bene come quest'uomo ha vissuto finora. Io che lo so, son sicuro che mi spiegherò tutto naturalissimamente, appena l'avrò veduto e avrò parlato con lui. Cammin facendo verso l'ospizio ove il poverino era stato ricoverato, seguitai a riflettere per conto mio: «A un uomo che viva come Belluca finora ha vissuto, cioè una vita "impossibile", la cosa più ovvia, l'incidente più comune, un qualunque lievissimo inciampo impreveduto, che so io, d'un ciottolo per via, possono produrre effetti straordinarii, di cui nessuno si può dar la spiegazione, se non pensa appunto che la vita di quell'uomo è "impossibile". Bisogna condurre la spiegazione là, riattaccandola a quelle condizioni di vita impossibili, ed essa apparirà allora semplice e chiara. Ero suo vicino di casa, e non io soltanto, ma tutti gli altri inquilini della casa si domandavano con me come mai quell'uomo potesse resistere in quelle condizioni di vita. Aveva con sé tre cieche, la moglie, la suocera e la sorella della suocera: queste due, vecchissime, per cataratta; l'altra, la moglie, senza cataratta, cieca fissa; palpebre murate. Tutt'e tre volevano esser servite. Strillavano dalla mattina alla sera perché nessuno le serviva. Le due figliuole vedove, raccolte in casa dopo la morte dei mariti, l'una con quattro, l'altra con tre figliuoli, non avevano mai né tempo né voglia da badare ad esse; se mai, porgevano qualche ajuto alla madre soltanto. Con lo scarso proventodel suo impieguccio di computista poteva Belluca dar da mangiare a tutte quelle bocche? Si procurava altro lavoro per la sera, in casa: carte da ricopiare. E ricopiava tra gli strilli indiavolati di quelle cinque donne e di quei sette ragazzi finché essi, tutt'e dodici, non trovavano posto nei tre soli letti della casa. Letti ampii, matrimoniali; ma tre.
Ebbene, signori: a Belluca, in queste condizioni, era accaduto un fatto naturalissimo. Quando andai a trovarlo all'ospizio, me lo raccontò lui stesso, per filo e per segno. Era, sì, ancora esaltato un po', ma naturalissimamente, per ciò che gli era accaduto. Rideva dei medici e degli infermieri e di tutti i suoi colleghi, che lo credevano impazzito. Signori, Belluca s'era dimenticato da tanti e tanti anni – ma proprio dimenticato – che il mondo esisteva.

Da Pirandello: *Novello per un anno, Il treno ha fischiato .*

N 16

Testo poetico

IN MEMORIA.
Locovizza il 30 settembre 1916.

Si chiamava
Moammed Sceab

Discendente
di emiri di nomadi
suicida
perché non aveva più
Patria
Amò la Francia
e mutò nome

Fu Marcel
ma non era Francese
e non sapeva più
vivere
nella tenda dei suoi
dove si ascolta la cantilena
del Corano
gustando un caffè

E non sapeva
sciogliere
il canto
del suo abbandono

L'ho accompagnato
insieme alla padrona dell'albergo
dove abitavamo
a Parigi
dal numero 5 della rue des Carmes
appassito vicolo in discesa.

Riposa
nel camposanto d'Ivry
sobborgo che pare
sempre
in una giornata
di una
decomposta fiera

E forse io solo
so ancora
che visse

Da Ungaretti : *L'Allegria*

N 17

Un’intera nottata
Buttato vicino
A un compagno
Massacrato
Con la bocca
Digrignata
Volta al plenilunio
Con la congestione
Delle sue mani
Penetrata
Nel mio silenzio
Ho scritto
Lettere piene d’amore

Non sono mai stato
Tanto
Attaccato alla vita.

Da Ungaretti : *Veglia*

N 18

Testo poetico

ALLE FRONDE DEI SALICI.

E come potevano noi cantare
Con il piede straniero sopra il cuore,
fra i morti abbandonati nelle piazze
sull'erba dura di ghiaccio, al lamento
d'agnello dei fanciulli, all'urlo nero
della madre che andava incontro al figlio
crocifisso sul palo del telegrafo?
Alle fronde dei salici, per voto,
anche le nostre cetre erano appese,
oscillavano lievi al triste vento.

Da Quasimodo . *Giorno dopo giorno .*

N 19

UOMO DEL MIO TEMPO

Sei ancora quello della pietra e della fionda,
uomo del mio tempo. Eri nella carlinga,
con le ali maligne, le meridiane di morte,
t'ho visto – dentro il carro di fuoco, alle forche,
alle ruote di tortura. T'ho visto: eri tu,
con la tua scienza esatta persuasa allo sterminio,
senza amore, senza Cristo. Hai ucciso ancora,
come sempre, come uccisero i padri, come uccisero
gli animali che ti videro per la prima volta.
E questo sangue odora come nel giorno
Quando il fratello disse all'altro fratello:
«Andiamo ai campi». E quell'eco fredda, tenace,
è giunta fino a te, dentro la tua giornata.
Dimenticate, o figli, le nuvole di sangue
Salite dalla terra, dimenticate i padri:
le loro tombe affondano nella cenere,
gli uccelli neri, il vento, coprono il loro cuore.

Da Quasimodo . *Giorno dopo giorno .*

N 20

Testo poetico

Meriggiare pallido e assorto
presso un rovente muro d'orto,
ascoltare tra i pruni e gli sterpi
schiocchi di merli, frusci di serpi.

Nelle crepe dei suolo o su la veccia
spiar le file di rosse formiche
ch'ora si rompono ed ora s'intrecciano
a sommo di minuscole biche.

Osservare tra frondi il palpitare
lontano di scaglie di mare
mentre si levano tremuli scricchi
di cicale dai calvi picchi.

E andando nel sole che abbaglia
sentire con triste meraviglia
com'è tutta la vita e il suo travaglio
in questo seguitare una muraglia
che ha in cima cocci aguzzi di bottiglia.

Da. Montale: *Ossi di Seppia*

N 21

Testo poetico

Ho sceso, dandoti il braccio, almeno un milione di scale
e ora che non ci sei è il vuoto ad ogni gradino.
Anche così è stato breve il nostro lungo viaggio.
Il mio dura tuttora, nè più mi occorrono
le coincidenze, le prenotazioni,
le trappole, gli scorni di chi crede
che la realtà sia quella che si vede.

Ho sceso milioni di scale dandoti il braccio
non già perché con quattr'occhi forse si vede di più.
Con te le ho scese perché sapevo che di noi due
le sole vere pupille, sebbene tanto offuscate,
erano le tue.

Da Montale : *Satura, sezione "Xenia,II"*

ASP.NET Core MVC

Introduzione alle applicazioni web MVC su ASP.NET Core

Anno 2020/2021

Indice generale

1 Introduzione

Il tutorial introduce i fondamenti sullo sviluppo di applicazioni web mediante il framework ASP.NET Core e utilizzando il *design pattern* **M**odel **V**iew **C**ontroller.

Prima di procedere a esaminare l'architettura il funzionamento delle applicazioni di ASP.NET Core, intendo fare un riepilogo sul funzionamento delle applicazioni web in generale.

1.1 Applicazioni web

Il termine *applicazione web* designa un'applicazione *distribuita*, nella quale un processo, in esecuzione su un *server*, offre dei servizi a dei *client*, processi in esecuzione sulle macchine degli utenti finali.[1] I client comunicano con il processo server utilizzando il protocollo HTTP.

La parte server dell'applicazione è "ospitata" all'interno di un processo chiamato *web server*[2], il quale fornisce i servizi fondamentali – connettività, *storage*, etc – ed è in grado di ospitare più applicazioni nello stesso momento. Tra i web server più famosi vi sono **Apache** e **Internet Information Services**.

Di seguito, e per tutto il tutorial, mi occuperò soltanto della parte server.

1.2 Siti web

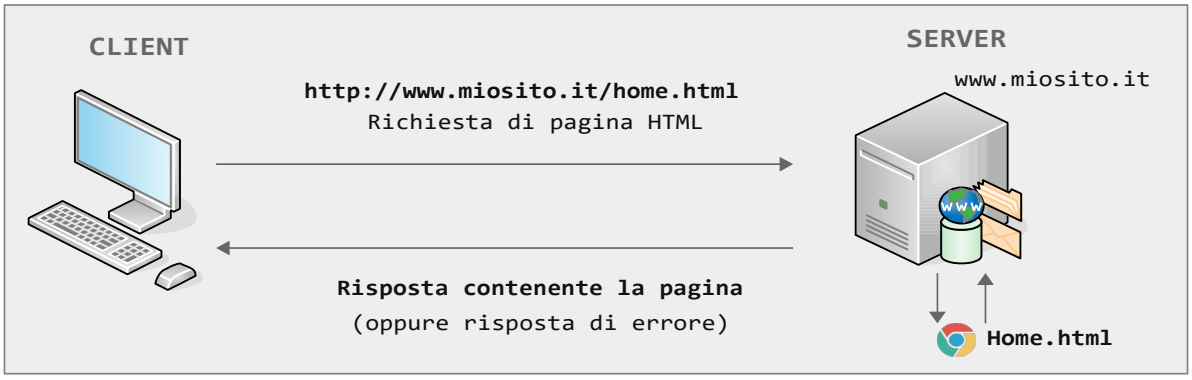
Un *sito web* è un tipo di applicazione web nel quale i client sono dei *browser* – Chrome, Opera, Safari, Edge, etc. Questi consentono all'utente di fruire dei contenuti e dei servizi forniti dal server, di solito sotto forma di pagine HTML.

1.2.1 Siti web statici

Un *sito web statico* fornisce un servizio di natura "documentale". Il client – il browser – richiede un documento specificandone il percorso e il nome - l'**URL** -, il server lo invia al client. Nella maggior parte dei casi i documenti richiesti sono pagine HTML, le quali rendono possibile la navigazione tra i contenuti del sito mediante gli *hyperlink*.

A pagina successiva è mostrata una tipica interazione client-server. Il client esegue una richiesta HTTP che, nell'URL, specifica la pagina **home.html**. Il server cerca la pagina nel proprio spazio e ne invia il contenuto al client mediante una risposta HTTP. Se la pagina non viene trovata, il server invia una risposta di errore.

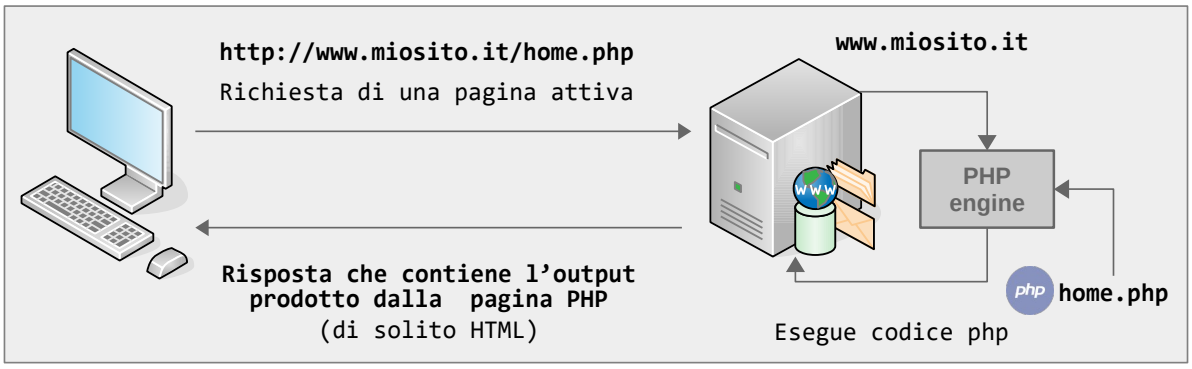
1 In realtà un client è qualunque processo richieda i servizi dell'applicazione web. Può accadere che un processo server sia a sua volta un client di un altro processo server.
2 Spesso si usa il termine *web server* anche per indicare la macchina fisica che ospita l'applicazione.



1.2.2 Siti web dinamici

Nei siti web dinamici i contenuti inviati al client sono in parte o in tutto prodotti mediante l'esecuzione di codice. Esistono varie tecnologie, ma la più comune è l'uso delle cosiddette *active server pages*: file che integrano al proprio interno sia codice HTML che codice esecutivo, scritto in un linguaggio che il server può processare.

Nello schema seguente, il client richiede una pagina PHP. Il server, dopo averla riconosciuta come una *pagina attiva*, non si limita a caricarla e a inviarla al client, ma la processa, eseguendo il codice PHP. Il risultato finale, tipicamente HTML, viene inviato al client.

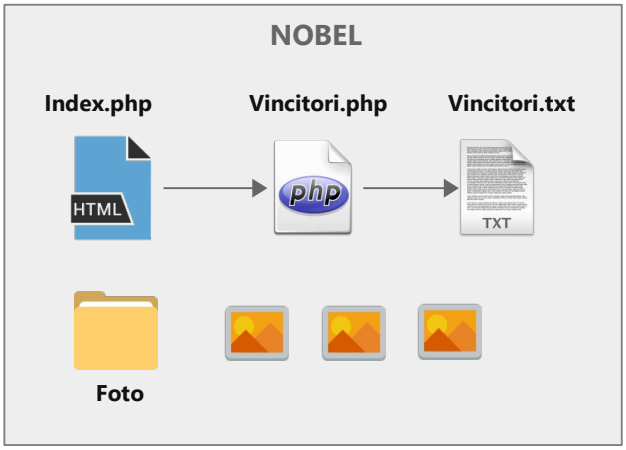


È fondamentale sottolineare che il processo di esecuzione della pagina – **home.php**, nell'esempio – avviene interamente nel server. Il client riceve soltanto il risultato prodotto dalla sua esecuzione.

1.3 Un semplice sito web di esempio

Di seguito propongo un semplicissimo sito composto da due pagine, una HTML, l'altra PHP. Il mio obiettivo è mostrare un sito con contenuti statici e dinamici e usarlo come punto di riferimento quando introdurrò il funzionamento di ASP.NET Core.

La home page del sito, **Index.html**, contiene un *hyperlink* alla pagina PHP, **Vincitori.php**, la cui funzione è visualizzare un elenco di vincitori di premio Nobel. Le informazioni sui vincitori sono memorizzate nel file, in formato CSV, **Vincitori.txt**. (Esiste anche una cartella, **Foto**, contenente le le foto dei vincitori.)



La pagina **Index.html** è una risorsa statica: il server ne invia il contenuto al client senza eseguire alcuna elaborazione

```
<html>
<head>
...
</head>
<body>
    <h1>NOBEL PRIZE</h1>
    <hr />
    <p><a href='Vincitori.php'>Elenco vincitori</a></p>
</body>
</html>
```

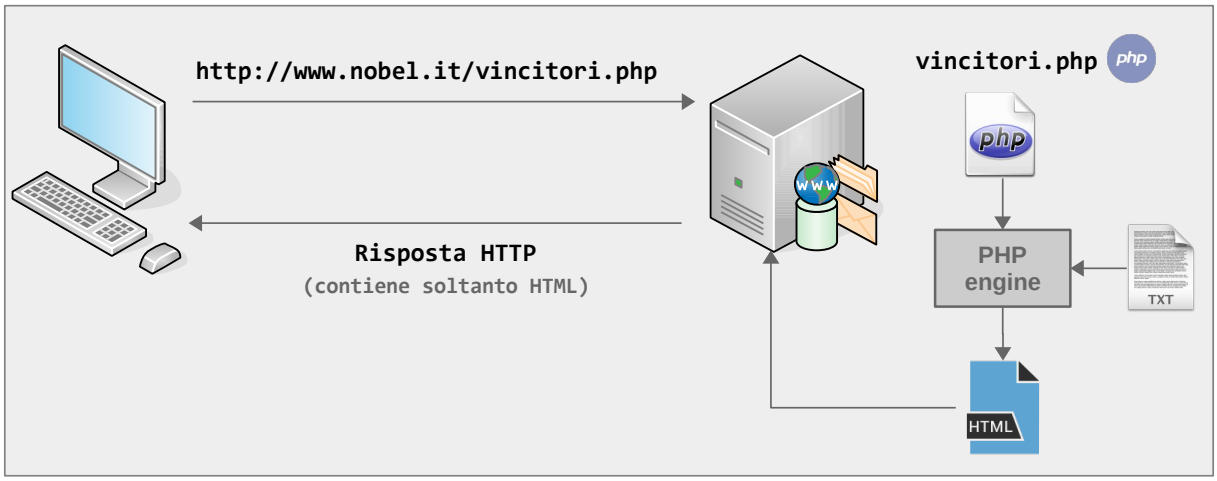
Quando l'utente clicca il link, il client richiede al server la pagina **Vincitori.php**:

```
<html>
<head>
...
</head>
<body>
    <?php

    $righe = file("Vincitori.txt");
    foreach ($righe as $value)
    {
        $items = str_getcsv($value, ";");
        echo '<p>' . $items[0] . ', ' . $items[1] . '</p>';
    }
    ?>
</body>
</html>
```

Questa è una *pagina attiva*: il server la processa ed esegue il codice PHP, il cui output viene integrato con l'HTML della pagina. L'output finale, puro HTML, viene inviato al client:

```
<html>
<head>
    ...
</head>
<body>
    <p>Einstein,  Albert </p>
    <p>Fermi,  Enrico </p>
    <p>Feynman,  Richard </p>
</body>
</html>
```

Lo schema riassume il processo che conduce, dalla richiesta della pagina **Vincitori.php**, alla risposta HTTP contenente l'HTML inviato al browser:



1.4 Conclusioni

L'uso di *pagine attive* consente di fornire al client dei contenuti esterni alle pagine, prelevati da un file, un database, etc, nonché di personalizzare tali contenuti in base alle azioni e/o all'identità dell'utente.

D'altra parte, il codice esecutivo che produce i contenuti di una pagina è integrato all'interno del codice HTML che stabilisce la loro presentazione. In sostanza, *non c'è alcuna separazione tra la logica che produce i contenuti e quella che li deve presentare*.

ASP.NET Core consente di superare brillantemente questo problema.

2 Introduzione ad ASP.NET Core

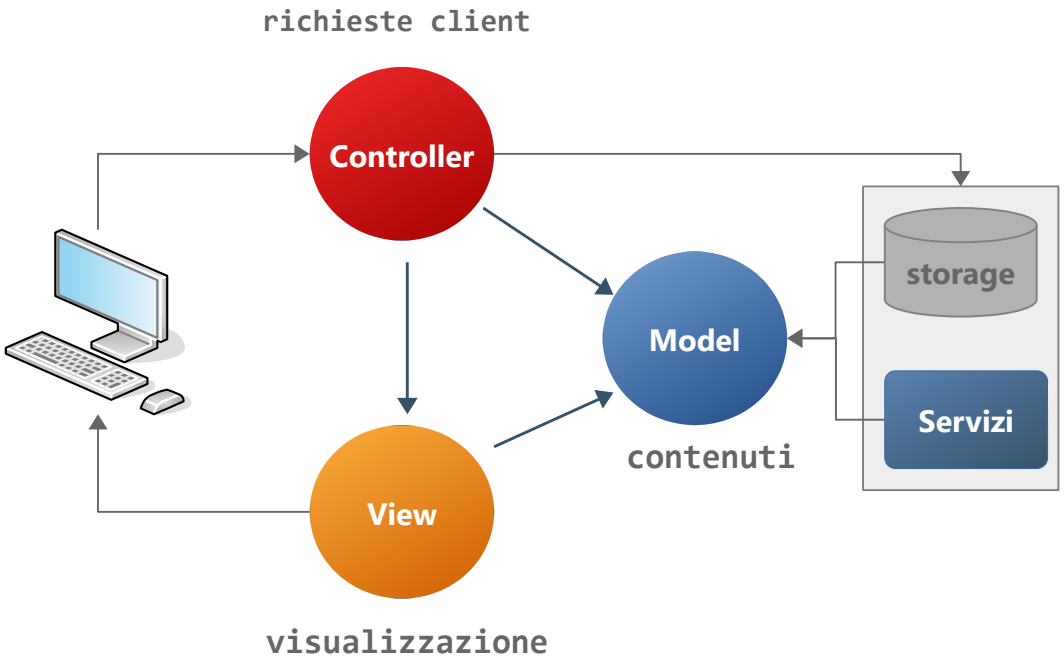
ASP.NET Core (***A**ctive **S**erver **P**ages over .**NET Core***) rappresenta un *framework cross-platform* per la realizzazione di applicazioni web. Il *framework* è basato su .NET Core, la versione multi piattaforma del .NET Framework.

ASP.NET consente di realizzare vari tipi di applicazione e di integrare varie tecnologie, alcune delle quali agiscono specificatamente sul client (e delle quali, dunque, non ci interesseremo). In questo tutorial mi limiterò a trattare la realizzazione di siti web mediante l'uso del *pattern* architetturale ***Model View Controller***.

2.1 Model View Controller (MVC)

Il *design pattern* MVC viene incontro all'esigenza, soprattutto nelle applicazioni distribuite, di isolare la logica che fornisce i contenuti da quella che li presenta. A questo scopo il cuore dell'applicazione è suddiviso in tre tipi di componenti:

- *model*: definiscono i contenuti. Considerando il programma **Library**, fanno parte del *model* le *entity class* `Book`, `Author`, `Genre`, etc.
- *view*: presentano i contenuti; rappresentano dunque l'interfaccia utente dell'applicazione.
- *controller*: processano le richieste dell'utente, ottengono il *model* che soddisfa tali richiese e lo passano alle *view*, che lo presenterà all'utente.



Di norma, nelle applicazioni realistiche, esistono altri componenti: *business logic*, servizi, accesso dati, configurazione, etc.

2.1.1 Relazione tra i componenti Model, View e Controller

Le frecce blu nello schema evidenziano le relazioni che intercorrono tra *model*, *view* e *controller*, e che caratterizzano questo *design pattern*.

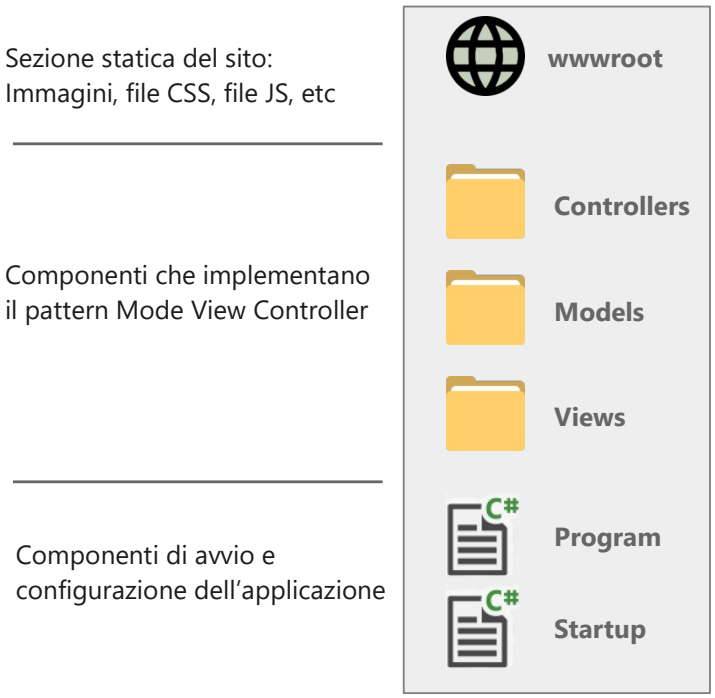
Il *model* è indipendente da *view* e *controller*: non dipende dunque dal tipo applicazione, dalla sua architettura o interfaccia utente. In generale, lo stesso *model* dovrebbe poter essere utilizzato in applicazioni desktop, *mobile*, web, etc.

Il *view* dipende dal *model*, poiché ha la funzione di presentarlo, ma non dipende dal *controller*. Semplicemente: il *view* riceve dei dati e li presenta all'utente.

Infine, il *controller* usa sia *view* che *model*: in risposta alle richieste dell'utente, ottiene il *model* da classi *data access*, servizi, etc, e lo passa al *view*.

2.2 Architettura di un'applicazione ASP.NET Core MVC

Lo schema sottostante riassume la struttura generale di un'applicazione ASP.NET Core MVC[3]:



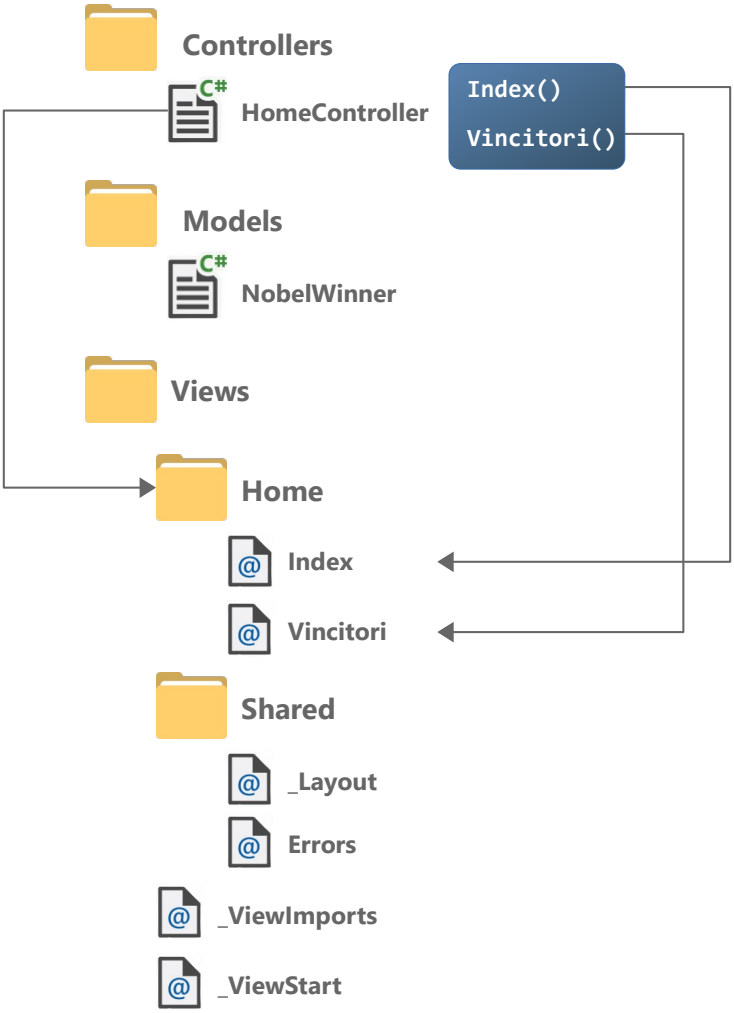
Gli elementi importanti, che caratterizzano questo tipo di applicazione, si trovano nelle tre cartelle centrali. Per quanto riguarda gli altri:

- la cartella **wwwroot** definisce la parte statica del sito, nella quale saranno collocate le immagini, i file CSS, le librerie javascript.
- **Program** produce l'avvio dell'applicazione. **Startup** consente di configurarla e di stabilire i servizi e componenti utilizzati, permettendo di aggiungerne di nuovi, come ad esempio Entity Framework.

3 Il progetto contiene anche altri file, che svolgono varie funzioni, tra le quali la configurazione del protocollo impiegato – HTTPS vs HTTP – e il numero di porta al quale il server starà in ascolto delle richieste del client.

2.2.1 Architettura del pattern MVC

Lo schema sottostante riassume la struttura dei componenti che, nell'applicazione, implementano il pattern MVC. Nota bene: la separazione di componenti non è soltanto logica, ma fisica, poiché risiedono in file separati; addirittura in cartelle separate.[4]



Nello schema ho supposto l'esistenza, come *model*, della classe `NobelWinner`.

Di default, l'applicazione definisce un solo *controller*, `HomeController`. Il *controller* definisce un metodo – *action* – per ogni *view* che deve visualizzare. (`Index()` e `Vincitori()`, nell'esempio)

Le *view*, file con estensione **cshtml**, sono collocate in sotto cartelle. Ad ogni *controller* corrisponde una cartella contenente le *view* che dovrà eseguire (nell'esempio, la cartella **Home** contiene le *view* **Index** e **Vincitori**).

La cartella **Shared** contiene le *view* condivise da tutta l'applicazione, tra queste **_Layout**, che stabilisce la struttura HTML comune a tutte le pagine del sito.

Infine, **_ViewImports** e **_ViewStarts** non hanno la funzione di presentare contenuti ma, come vedremo, di semplificare la realizzazione delle altre *view*.

4 In realtà la strutturazione nelle cartelle **Models**, **Controllers** e **Views** è l'impostazione di default e può essere modificata mediante un'opportuna configurazione.

2.3 Funzionamento di un'applicazione ASP.NET Core MVC

L'elaborazione di una richiesta da parte di un'applicazione ASP.NET Core MVC segue un percorso diverso da quello schematizzato nel paragrafo 1.3.

Le differenza comincia fin dalla codifica degli URL, che stabiliscono le risorse richieste dal client.

2.3.1 Codifica degli URL: route

Nei siti "normali" il client specifica nell'URL il nome della risorsa richiesta (file HTML, PHP, o altro), come mostra il frammento della pagina **Index.php**:

```
...
    <p><a href='Vincitori.php'>Elenco vincitori</a></p>
...
```

Il caricamento della pagina **Index.html** implica una richiesta analoga, con la differenza che, essendo la pagina di default, può essere omessa. Dunque:

`http://www.nobel.it`

equivale a:

`http://www.nobel.it/index.html`

In ASP.NET Core gli URL hanno una struttura diversa, definita *route*: indicano il *controller* e l'*action* che devono rispondere alla richiesta. In sintesi, dunque, *una route specifica un metodo da eseguire*.

Una *route* rispecchia la seguente struttura:

***/controller*=Home / *action*=Index / [id]**

dove **home** e **index** sono i valori predefiniti nel caso non vengano specificato *controller* e/o *action*.

Nel caso del sito Nobel, le due *route* per ottenere la home page e i vincitori sarebbero:

`/home/index` e `/home/vincitori`

Nel primo caso, la *route* può ridursi al carattere `/`, dato che *home* e *index* sono il *controller* e l'*action* predefiniti.

2.3.2 Route con parametro

Una *route* può specificare un parametro facoltativo (campo **id** nella struttura della *route*). Tratterò le *route* con parametri più avanti.

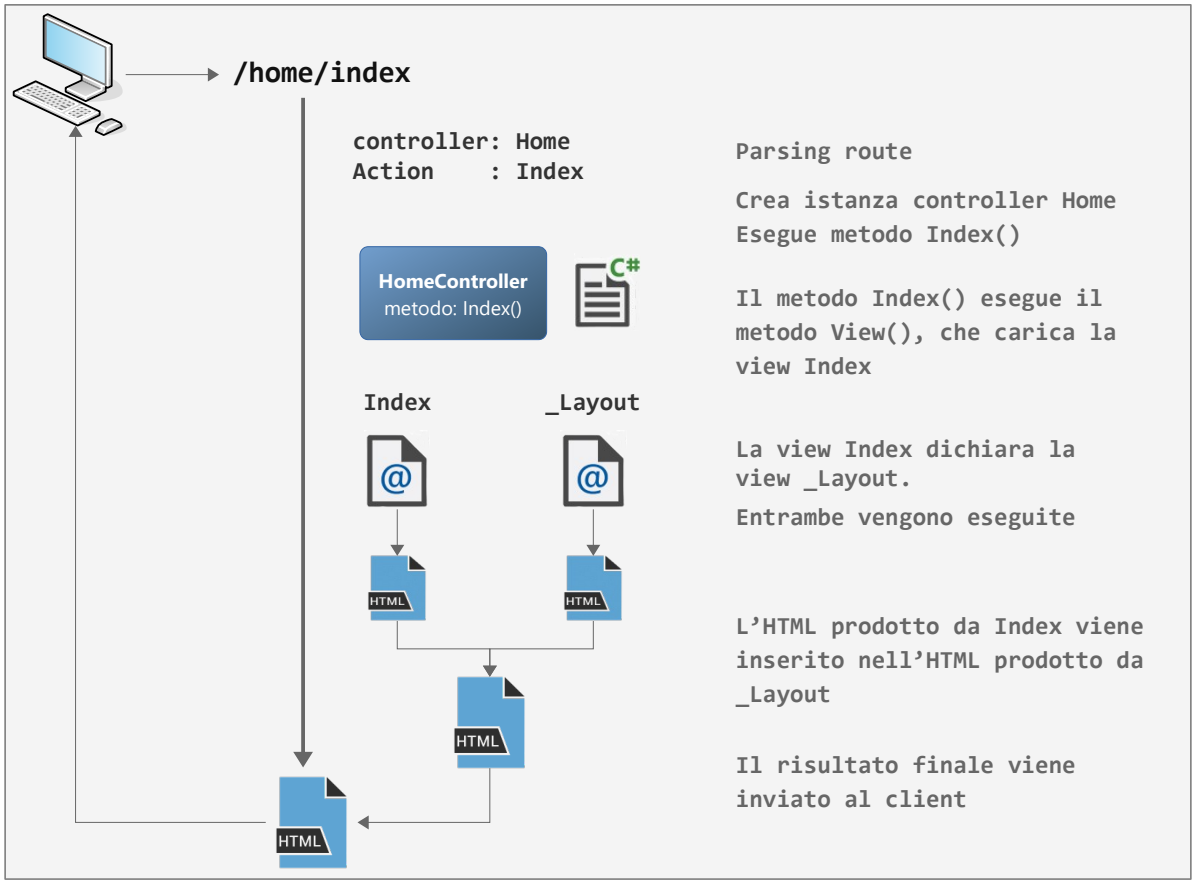
2.3.3 Elaborazione della richiesta

Una richiesta del client viene sempre processata da un metodo definito in un *controller*. La *route* della richiesta serve appunto a stabilire il metodo da eseguire.

È opportuno ribadirlo: il client non chiede una pagina o qualunque altro tipo di risorsa, statica o dinamica (come una pagina PHP); alla richiesta del client corrisponde sempre l'esecuzione di un metodo di un *controller*.

Naturalmente, il client "ignora" completamente i concetti di *controller* e *action*; a seguito della sua richiesta otterrà una pagina HTML o altro tipo di contenuto.

Lo schema seguente mostra a grandi linee come avvenga tutto ciò. Supponi che il client richieda la home page del sito, specificando dunque l'URL **http://www.nobel.it/home/index**:



Lo schema nasconde molti dettagli, ma mette in evidenza la questione principale: è il metodo `Index()` a stabilire il contenuto da restituire al client. Lo schema, quindi, mostra il comportamento predefinito:

1. Il metodo `Index()`, mediante l'esecuzione del metodo `View()`, carica la *view* **Index**.
2. La *view* **Index**, che definisce i contenuti da visualizzare, dichiara la *layout view* **_Layout**, la quale definisce la struttura generale delle pagine del sito.
3. Entrambe le *view* sono eseguite (allo stesso modo di una pagina PHP).
4. Il risultato prodotto dalla *view* **Index** viene inserito nell'HTML prodotto dalla *layout view* **_Layout**.
5. Il risultato finale, una pagina HTML, viene inviato in risposta al client.

2.4 Implementazione delle *view: C# + HTML = razor*

Le *view* sono le equivalenti delle *pagine attive*, possono contene codice HTML e codice esecutivo. ASP.NET Core fornisce una tecnologia di parsing, chiamata *razor*, che consente di integrare in molto semplice naturale codice C# e HTML.

Mediante il carattere `@` è possibile inserire un costrutto C# in qualsiasi punto della *view*. Seguono alcuni esempi (per un approfondimento vedi: Guida completa Razor [5])

2.4.1 Blocchi di codice

Ad esempio:

```
@{
    ViewData["Title"] = "Home";
    string[] elencoCittà = {"Roma", "Milano", "Firenze" };
   // da qui in poi "elencoCittà" è utilizzabile ovunque
   // (non può essere dichiarata un'altra variabile con lo stesso nome)
}
```

Nota bene: le variabili dichiarate in blocco sono accessibili ovunque nella *view*, a partire dalla linea successiva al blocco. (Si comportano come le variabili locali di un metodo)

2.4.2 Costrutti di controllo

`if`, `for`, `foreach`, `while`, etc. Il costrutto non deve terminare con il carattere `;`. Ad esempio:

```
@for (int i = 0; i < 10; i++)
{
    <p>HELLO!</p>
}
```

2.4.3 Espressioni implicite

Costrutti che producono un valore, comprese le chiamate a metodi. Ad esempio:

```
@DateTime.Now

@foreach (var città in elencoCittà)
{
    <p>@città.ToLower()</p>

    <a href="home/dettagli/@città">@città</a>
}
```

Nota bene: le *espressioni implicite* non devono contenere spazi, a meno che l'istruzione non abbia una terminazione non ambigua. (Ad esempio, la lista degli argomenti passata a un metodo può avere degli spazi, dato che la parentesi finale termina esplicitamente l'espressione.)

2.4.4 Espressioni esplicite

Sono rappresentate da una coppia di `(` `)` contenenti l'espressione. Ad esempio:

```
<p>L'altra settimana: @( DateTime.Now - TimeSpan.FromDays(7) )</p>
```

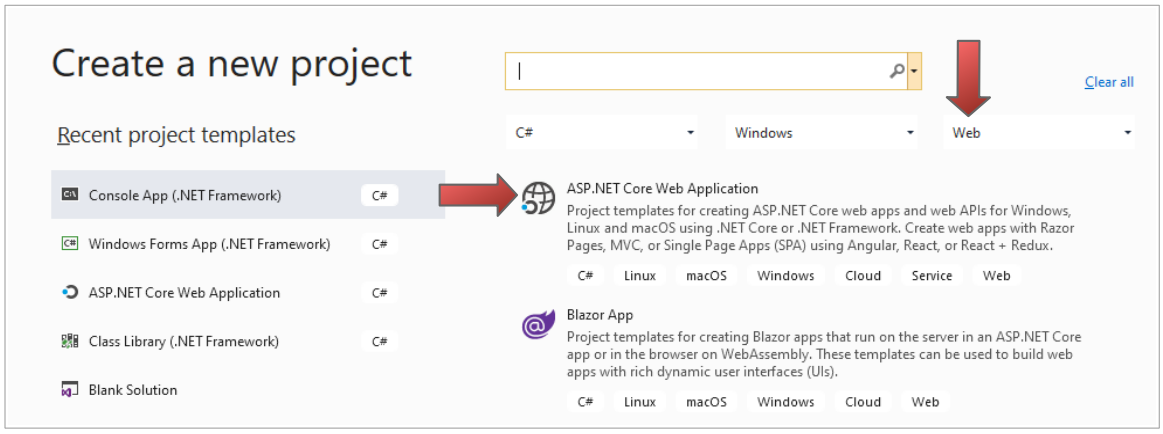
5 https://docs.microsoft.com/it-it/aspnet/core/mvc/views/razor?view=aspnetcore-3.1

3 Creare applicazioni ASP.NET Core MVC

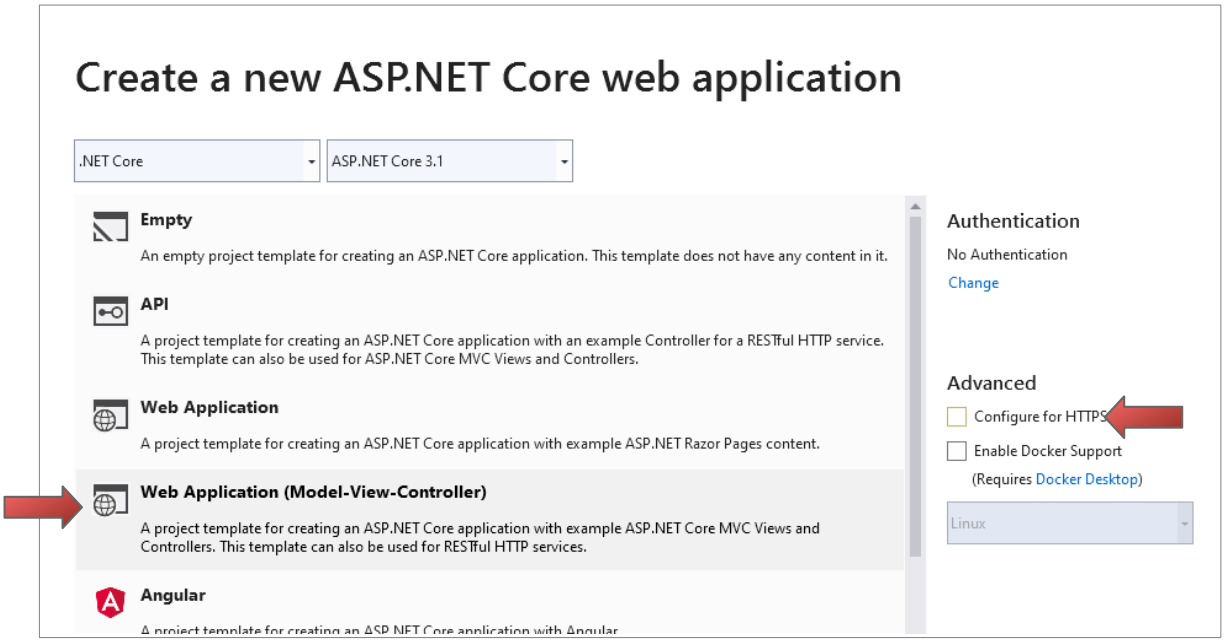
Di seguito mostrerò come realizzare un sito ASP.NET Core MVC che emuli il comportamento di quello realizzato in PHP. Nel mentre aggiungerò alcuni dettagli sul funzionamento di ASP.NET Core sui quali ho finora sorvolato.

3.1 Creazione di un progetto

Si crea un progetto selezionando il tipo di applicazione **ASP.NET Core Web Application**, nella la categoria **web**[6]:



Successivamente si stabilisce il nome e la collocazione della *solution*. Infine si sceglie il *template* da utilizzare per la creazione del progetto (**Model-View-Controller**):



Per semplificare lo sviluppo è opportuno disabilitare la voce **Configure for HTTPS**.

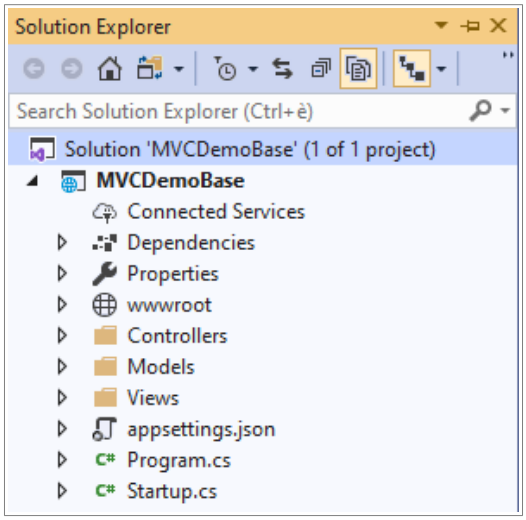
6 Nell'ultima versione di Visual Studio il processo di creazione de un'applicazione web presenta delle schermate diverse. Le opzioni disponibili, comunque, sono le stesse.

3.2 Struttura generale del progetto

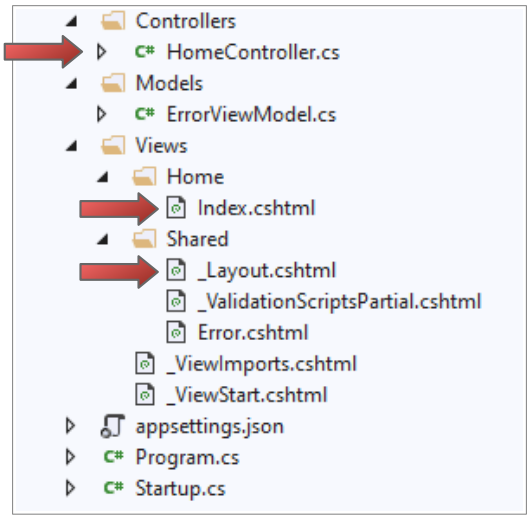
ASP.NET Core crea un'applicazione già funzionante, dotata di due *view* e una *layout view* che forniscono dei contenuti dimostrativi. Poiché intendo partire praticamente da zero, elimino quasi tutto il codice generato, lasciando le *view* **Index** e **_Layout** ridotte al minimo.

Dopo queste modifiche, *Solution Explorer* mostra la seguente struttura, della quale interessano soprattutto i tre elementi evidenziati:

Struttura generale



Struttura pattern MVC



3.3 Contenuto predefinito delle *view*

Dopo il mio intervento, le *view* **_Layout** e **Index** hanno il seguente contenuto:

_Layout

(ho omesso parte del tag **head**)

```
<!DOCTYPE html>
<html lang="en">
<head>
    ...
    <title>@ViewData["Title"]</title>
</head>
<body>
    <h2>NOBEL PRIZE</h2>
    <hr />
    @RenderBody()
</body>
</html>
```

Index

```
@{
    ViewData["Title"] = "Home";
}
<div>
   <a href='/home/vincitori'>Vincitori</a>
</div>
```

Le due linee evidenziano il legame tra **Index** e **_Layout**: il risultato prodotto da **Index** viene inserito nel risultato prodotto da **_Layout**, in corrispondenza del metodo `RenderBody()`.

Esistono altre due *view*, **_ViewStart** e **_ViewImports**, che non hanno la funzione di presentare contenuti, ma di definire una volta per tutte il codice da eseguire al caricamento di ogni *view*, evitando che sia necessario specificarlo ogni volta:

- **_ViewStart** stabilisce la *layout view* che presenta la struttura generale delle pagine del sito (nei siti reali possono esistere più *layout view*). Di default è **_Layout**.
- **_ViewImports** importa i *namespace* necessari ad ogni *view*.

_ViewStart

```
@{
    Layout = "_Layout";
}
```

_ViewImports

```
@using MVCDemoBase
@using MVCDemoBase.Models
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
```

3.4 Controller

Di default viene generato lo scheletro dell'`HomeController` (anche in questo caso ho semplificato il codice):

```
public class HomeController : Controller
{
    public HomeController()
    {
    }

    public IActionResult Index()
    {
        return View();
    }
}
```

Il metodo `View()` restituisce la *view* corrispondente al nome del metodo `Index()`. Infatti, internamente il metodo `View()` esegue un codice equivalente a quello mostrato di seguito:

```
public class HomeController : Controller
{
    public IActionResult Index()
    {
        return new ViewResult() { ViewName = "Index" };
        return View();
    }
    ...
}
```

Una volta stabilito il nome della *view*, ASP.NET Core la localizza all'interno della struttura del sito e la esegue (usando il pattern *controller* **home** → cartella **Home**; action **Index** → *view* **Index**).

3.5 Esecuzione dell'applicazione

L'applicazione è configurata per essere eseguita sul web server **IIExpress**, all'indirizzo **localhost**. Il numero di porta è generato casualmente, ma può essere modificato aprendo le proprietà del progetto, selezionando la scheda **Debug** e modificando l'**App URL**. (Ho impostato la porta **5000**).

L'esecuzione del progetto implica:

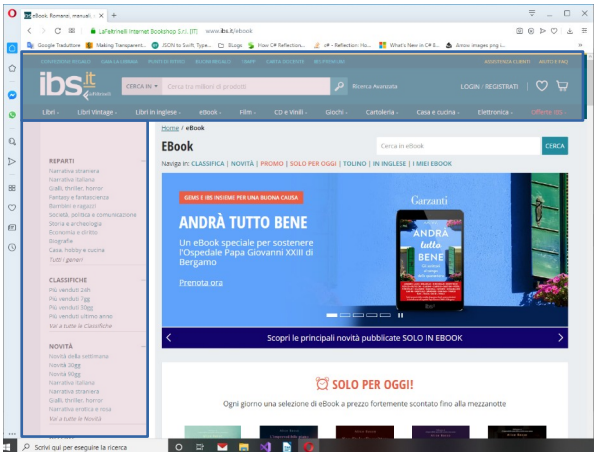
1. L'avvio del *web server* IIExpress (nell'area di notifica appare l'icona corrispondente), se non è già in esecuzione.
2. L'avvio dell'applicazione.
3. L'apertura del browser e la richiesta all'URL: **localhost:5000**, che corrisponde alla *route*: **localhost:5000/home/index**

3.6 *View*, *layout view* e pagine web

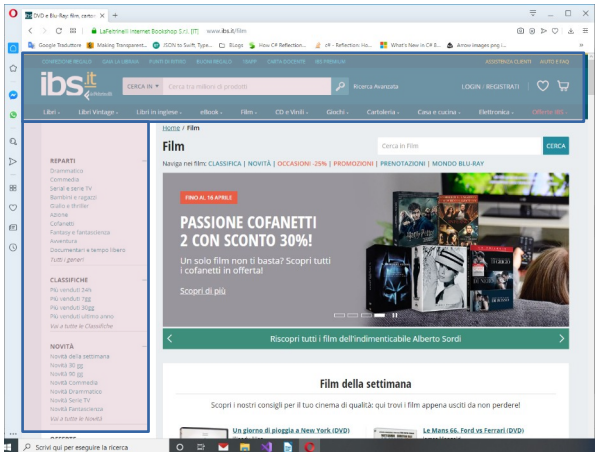
Il modello di presentazione dei contenuti adottato da ASP.NET Core viene incontro a un problema tipico nella realizzazione di siti web: evitare la duplicazione del codice comune nelle pagine del sito.

Ad esempio, lo *screen shot* sottostante mostra due pagine del sito IBS; le sezioni in alto e a sinistra sono identiche, cambia il contenuto mostrato nella sezione centrale:

IBS – Contenuto EBook



IBS – Contenuto Film



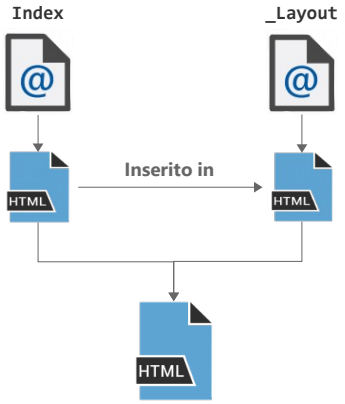
Nella maggior parte dei siti, infatti, le pagine condividono una struttura comune, dei contenuti comuni e, infine, l'uso di risorse comuni, come fogli di stile e librerie javascript.

A questo scopo, tutti i *framework* per lo sviluppo di siti web implementano il concetto di *master page*, o *layout page*, e cioè l'idea di collocare la struttura comune in un componente speciale, in modo che le pagine del sito possano limitarsi a presentare i contenuti specifici (Ebook e Film, nello *screen shot*).

In ASP.NET Core questa soluzione è implementata mediante l'uso delle *layout view*, e cioè *view*

che definiscono struttura, contenuti e risorse comuni alle pagine del sito.

Durante l'elaborazione di una richiesta, l'HTML della *view* che presenta i contenuti, caricata dal metodo *action* del *controller*, viene inserito nell'HTML prodotto dalla *layout view*.



Nell'esempio considerato, **Index** e **_Layout** producono il seguente risultato; a sinistra, l'HTML in grigio chiaro è prodotto da **_Layout**, quello in grigio scuro da **Index**:

HTML inviato al browser

```
<!DOCTYPE html>
<html>
<body>
    <h2>NOBEL PRIZE</h2>
    <hr />
    <div>
     <a href='/home/vincitori'>vincitori</a>
    </div>
</body>
</html>
```

Output prodotto dal browser

3.7 Impostare il titolo della pagina: uso di ViewData

Il titolo delle pagine web viene visualizzato sulla barra del titolo del browser e, in HTML, viene impostato mediante il tag **title**, contenuto a sua volta nel tag **head**.

In ASP.NET Core questo è un problema, poiché i contenuti inviati al client sono definiti nelle *view*, ma il tag **head** è specificato una volta per tutte nella *layout view*. Ergo: il titolo della pagina non può essere definito nelle *view*, come invece dovrebbe.

La soluzione è quella di memorizzare il titolo nel dizionario `ViewData` all'interno delle singole *view*. **_Layout** utilizzerà il valore nel tag **title** della pagina:

Index

```
@{ViewData["Title"] = "Home";}
<div>
```

_Layout

```
<html>
    <head>
```

```
    HOME
</div>
```

```
        <title>@ViewData["Title"]</title>
    </head>
...
```

Come vedremo, `ViewData` può essere impiegato anche per altri scopi.

3.8 Visualizzazione dei vincitori: passare il *model* alla *view*

Nella versione PHP del sito, la pagina **Vincitori.php** carica i nomi dei vincitori dal disco e li presenta all'utente. In ASP.NET Core è il *controller* che risponde alle richieste del client, dunque è il *controller* che deve ottenere i dati e passarli alla *view* che dovrà presentarli.

3.8.1 Caricare i nomi dei vincitori

Dopo aver aggiunto una cartella **Data** al progetto e avervi collocato il file **Vincitori.txt**, aggiungo alla cartella **Models** la classe che rappresenta il singolo vincitore:

```
public class NobelWinner
{
    public string LastName { get; set; }
    public string FirstName { get; set; }
    public string FullName => LastName + ", " + FirstName;
}
```

In `HomeController` colloco il metodo che carica i dati (si tratta di una soluzione temporanea):

```
public class HomeController : Controller
{

   ...
   List<NobelWinner> LoadNobelWinners()
   {
      var winners = new List<NobelWinner>();
      foreach (var riga in System.IO.File.ReadAllLines("Data/Vincitori.txt"))
      {
          if (riga.StartsWith("#"))
              continue;

          var items = riga.Split(';');

          var nw = new NobelWinner
          {
              LastName = items[0].Trim(),
              FirstName = items[1].Trim(),
          };
          winners.Add(nw);
      }
      return winners;
  }
}
```

3.8.2 Passaggio dei contenuti alla view Vincitori: uso di ViewData

Occorre un metodo *action* gestisca la richiesta **home/vincitori**. Il metodo ha responsabilità di caricare la *view* omonima, passandogli i contenuti da visualizzare. A questo scopo esistono due modalità, la prima delle quali prevede di collocare i contenuti nel dizionario `ViewData`:

```
public class HomeController : Controller
{
    ...
    public IActionResult Index()
    {
        return View();
    }

    public IActionResult Vincitori()
    {
        ViewData["vincitori"] = LoadNobelWinners();
        return View();
    }
    ...
}
```

3.8.3 Visualizzazione dei vincitori: creazione view Vincitori

Per creare una *view*, Visual Studio fornisce la voce di menù necessaria, ma è altrettanto valido eseguire un semplice copia incolla di un'altra *view* (**Index** in questo caso).

Nella *view* **Vincitori**, prima si accede ai dati memorizzati in `ViewData`, poi si scorrono in modo del tutto simile a quanto faremmo in PHP.

```
@{
    ViewData["Title"] = "Vincitori";
    var winners = ViewData["vincitori"] as List<NobelWinner>;
}

<div>
    @foreach (var wi in winners)
    {
        <p>@wi.FullName</p>
    }
</div>
```

3.9 Usare il *model*: passare alla *view* dati *tipizzati*

L'uso di `ViewData` per passare i contenuti alla *view* è funzionale, ma non ottimale. ASP.NET Core fornisce un modo migliore, nel quale la *view* definisce il *model* come se dichiarasse una variabile. Si usa la direttiva `@model`:

```
@model <tipo variabile>
```

Nella *view* **Vincitori** il *model* è rappresentato da una lista di `NobelWinner`:

```
@{
```

```
    ViewData["Title"] = "Vincitori";
}
@model List<NobelWinner>  // variabile implicita di nome Model

<div>

    @foreach (var nw in Model)
    {
        <p>@nw.FullName</p>
    }
</div>
```

Nota bene: è come se fosse dichiarata una variabile di nome `Model` di tipo `List<NobelWinner>`.

3.9.1 Passare il model

Il passaggio dei dati dal *controller* alla *view* è estremamente semplice. Il metodo `View()` ha una versione che accetta come argomento il *model* da passare alla *view*:

```
public IActionResult Vincitori()
{
    var winners = LoadNobelWinners();
    return View(winners);

}
```

3.10 Passare dati in una richiesta: databinding

Negli esempi considerati finora, sia in PHP che in ASP.NET Core, le richieste del client non contenevano informazioni nell'URL se non la risorsa da visualizzare. In molti casi, però, l'URL definisce uno o più valori necessari a qualificare la richiesta.

Ad esempio, un sito di prodotti elettronici potrebbe visualizzare nella home un elenco di articoli in offerta; cliccando su uno degli articoli si apre una pagina che ne mostra i dettagli. In questo caso la richiesta dovrà specificare sia la pagina che visualizza i dettagli, sia il codice dell'articolo da visualizzare.

In ASP.NET Core si possono usare tre tecniche:

1. Passare l'informazione nella *route;* modalità che consente di passare un solo parametro.
2. Usare una *query string*, che consente di passare più parametri.
3. Usare entrambe le modalità nello stesso URL.

Qualunque sia la modalità utilizzata, ASP.NET Core applica un meccanismo molto potente e versatile, il ***databinding***, che associa automaticamente il valore/i presenti nell'URL al parametro/i del metodo che dovrà elaborare la richiesta.

3.11 Passare un parametro nella *route*

La struttura di una *route* prevede la possibilità di specificare, oltre a *controller* e *action*, un valore aggiuntivo.

/controller=Home / *action*=Index / **[id]**

Nel sito Nobel è possibile utilizzare questa possibilità per ottenere la visualizzazione "dettagli" di un premo Nobel selezionato dall'elenco dei vincitori. Per farlo occorre innanzitutto modificare la *view* **Vincitori**, in modo che visualizzi l'elenco mediante degli *hyperlink*:

```
@{
    ViewData["Title"] = "Vincitori";
}
@model List<NobelWinner>

<div>
    @foreach (var nw in Model)
    {
        <p><a href="/home/vincitore/@nw.FullName">@nw.FullName</a> </p>
    }
</div>
```

Ad esempio, se l'utente clicca sul primo link, il client invia la richiesta:

/home/vincitore/`Einstein, Albert`

dove quello evidenziato è il valore associato al campo **id** della route.

3.11.1 Gestire una richiesta con parametro nella route

Il metodo *action* che gestisce la richiesta deve semplicemente specificare un parametro stringa di nome `id`:

```
public IActionResult Vincitore(string id)  //id -> FullName del vincitore scelto
{
    ...
}
```

Nota bene: essendo il valore specificato nella *route*, il parametro deve chiamarsi `id`. (Non esistono distinzioni tra maiuscole/minuscole nel nome del parametro.)

3.12 Usare una *query string*

Una *query string* è una stringa che contiene un elenco di coppie *chiave=valore*, separate dal carattere `&`. La stringa è preceduta dal carattere `?` e segue l'URL della richiesta.

La seguente *query string* specifica un solo valore, di chiave *fullname*:

`http://www.Nobel.it/home/vincitore?fullname=Einstein, Albert`

Ecco la sua applicazione nella *view* **Vincitori**:

```
...
<div>
    @foreach (var nw in Model)
    {
        <p><a href="/home/vincitore?fullname=@nw.FullName">@nw.FullName</a> </p>
    }
<div>
```

3.12.1 Gestire una richiesta con query string

Il metodo *action* che gestisce la richiesta deve specificare un parametro stringa con lo stesso nome della chiave usata nella *query string*:

```
public IActionResult Vincitore(string fullName)
{
    ...
}
```

Il processo di *databinding* si occuperà di trovare la corrispondenza tra la chiave della *query string* e il parametro del metodo.

(Ancora una volta: non esiste distinzione tra maiuscole/minuscole.)

3.12.2 Gestire due parametri nella query string

L'uso di una *query string* è utile soprattutto quando occorre passare due o più parametri:

```
<div>
    @foreach (var nw in Model)
    {
        <p><a href="/home/vincitore?firstname=@nw.FirstName&lastname=@nw.LastName">
           @nw.FullName</a> </p>
    }
<div>
```

Il metodo *action* dovrà specificare due parametri con lo stesso nome delle chiavi:

```
public IActionResult Vincitore(string firstName, string lastName)
{
    ...
}
```

Nota bene: l'ordine di dichiarazione dei parametri non ha importanza, il processo di *databinding* basa la propria corrispondenza sui nomi.

3.13 Definire gli *hyperlink* mediante i *tag helper*

I *tag helper* sono attributi *server-side*, poiché vengono processati dal server; sono utili per personalizzare la definizione dei tag HTML. Sono usati in molti scenari; qui li utilizzo per definire l'*hyperlink* della *view* **Vincitori** e semplificare il passaggio dei parametri *firstname* e *lastname*.

Ricordo che l'URL usato è il seguente:

/home/vincitore?firstname=Albert&lastname=Einstein

il quale specifica: *controller*, *action*, *firstname* e *lastname*. Ebbene, è possibile definire questi valori separatamente e lasciare ad ASP.NET Core il compito di costruire l'URL:

```
<div>
   @foreach (var nw in Model)
   {
       <p><a asp-controller ="home"
             asp-action="vincitore"
             asp-route-firstname="@nw.FirstName"
             asp-route-lastname="@nw.LastName">@nw.FullName</a></p>
   }
</div>
```

Nota bene: mediante i *tag helper* `asp-route-chiave` è possibile stabilire il nome della chiave da associare al valore.

Questa tecnica è valida anche se vogliamo passare un parametro nella *route*. Dunque, l'*hyperlink*:

```
<a href="/home/vincitore/@nw.FullName">@nw.FullName</a>
```

può essere scritto:

```
<a asp-controller ="home" asp-action="vincitore"
   asp-route-id="@nw.FullName">="@nw.FullName</a>
```

Sarà ASP.NET Core, nell'eseguire la *view*, a tradurre la seconda forma nella prima.

4 Un esempio completo di applicazione: MotoGP

Di seguito svilupperò una nuova applicazione allo scopo di mostrare altre funzioni delle applicazioni web e la loro implementazione mediante di ASP.NET Core.

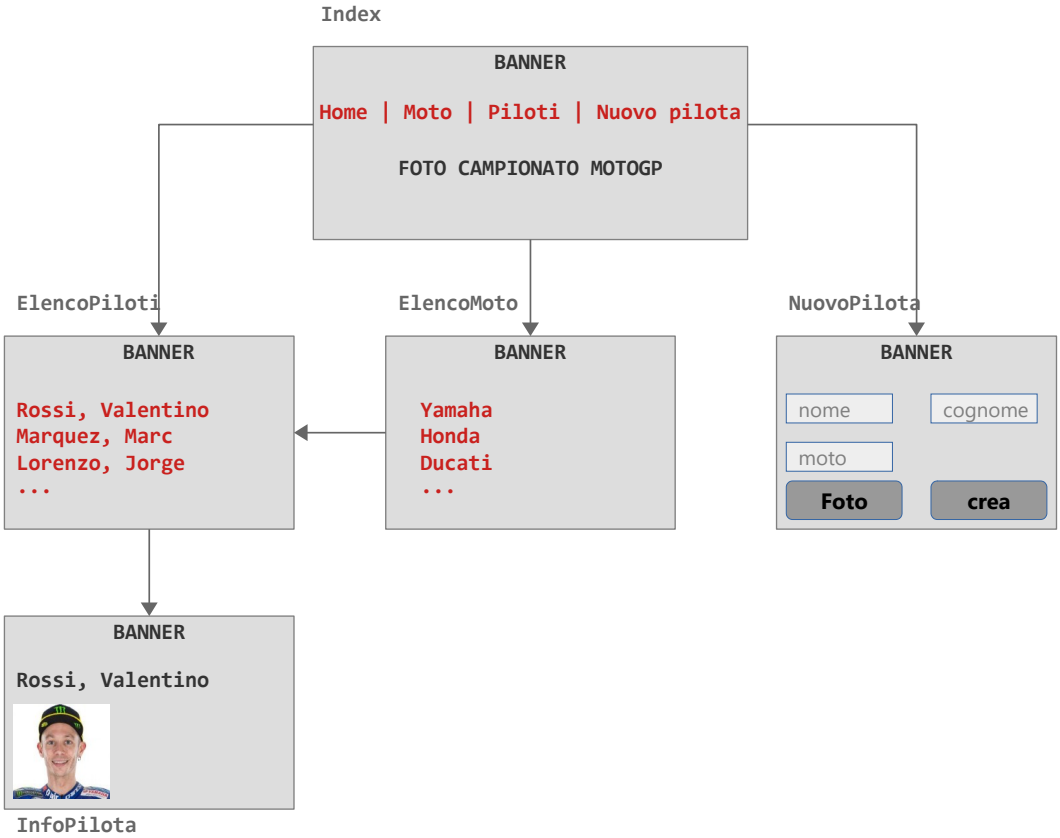
4.1 Descrizione generale dell'applicazione

L'obiettivo è realizzare un sito web per la gestione del campionato di Moto GP. Il sito dovrà consentire di:

- Visualizzare l'elenco dei piloti e delle moto.
- Visualizzare i piloti che corrono con una determinata moto.
- Visualizzare tutte le informazioni relative a un singolo pilota, foto compresa.
- Inserire un nuovo pilota.

Segue lo schema delle pagine del sito, ad ognuna delle quali corrisponde una *view*. In tutte le pagina comparirà un menù per accedere alle funzioni principali.

Le frecce nello schema indicano la navigazione tra le pagine. Ad esempio, selezionando una moto in **ElencoMoto**, il sito dovrà visualizzare i piloti che corrono con quella moto.



4.1.1 *View e controller*

Riepilogando, intendo definire le seguenti *view:*

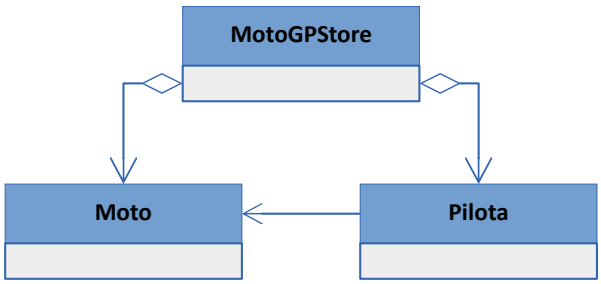
- **Index**: è la home page e mostra semplicemente una foto e definisce il menù.
- **ElencoMoto**: visualizza l'elenco delle marche iscritte al mondiale; consente di cliccare su una moto e ottenere l'elenco dei piloti che corrono con essa. (*view* **ElencoPiloti**)
- **ElencoPiloti**: visualizza l'elenco dei piloti; tutti, o soltanto i piloti di una determinata moto. Consente di cliccare sul nominativo di un pilota e ottenere le informazioni disponibili (*view* **InfoPilota**)
- **InfoPilota**: visualizza la informazioni sul singolo pilota.
- **NuovoPilota**: consente l'inserimento di un nuovo pilota.

Tutte le richieste saranno gestire dall'`HomeController`.

4.2 Model e *data access*

Rispetto alla soluzione adottata per il sito Nobel, intendo usare un approccio "più professionale" nella gestione dei contenuti.

Il *model* è rappresentato da due classi, `Pilota` e `Moto`, collocate entrambe nella cartella **Models**. L'accesso ai dati è fornito dalla classe `MotoGPStore`. Questa genera in memoria le liste dei piloti e delle moto; sono entrambe liste statiche, dunque il loro ciclo di vita coincide con quello dell'applicazione.



Segue il codice delle classi:

```
public class Pilota
{
    public int PilotaId { get; set; }
    public string Nome { get; set; }
    public string Cognome { get; set; }
    public string Nominativo { get {...}  }
    public int MotoId { get; set; }
    public Moto Moto { get; set; }
    public int Punti { get; set; }
    public int Vittorie { get; set; }
    public string FileFoto { get; set; }
}
```

```
public class Moto
{
    public int MotoId { get; set; }
    public string Nome { get; set; }
}
```

```
public class MotoGPStore
{
    private static List<Pilota> piloti = new List<Pilota>();
    private static List<Moto> moto = new List<Moto>();
    static int pilotId = 0;
    static int motoId = 0;
    static MotoGPRepository()
    {
        CreaMoto(new Moto { Nome = "Yamaha" });
        ...

        CreaPilota(new Pilota {...});
        CreaPilota(new Pilota {...});
        ...
    }
}
```

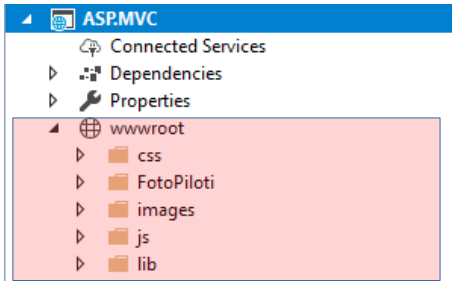
Per semplicità, anche `MotoGPStore` è collocata nella cartella **Models**.

4.3 Layout view

La *layout view* **_Layout** definisce una foto che funge banner e il menù, poiché entrambi gli elemento devono comparire in tutte le pagine:

```
<!DOCTYPE html>
<html>
    <head>
        <title>@ViewData["Title"]</title>
        <link rel="stylesheet" href="~/css/site.css" />
    </head>
<body>
    <header>
        <img src="~/images/Banner.png"/>
    </header>
    <menu>
        <ul>
            <li><a asp-controller="home" asp-action="Index">Home</a></li>
            <li><a asp-controller="home" asp-action="ElencoMoto">Moto</a></li>
            <li><a asp-controller="home" asp-action="ElencoPiloti">Piloti</a></li>
            <li><a asp-controller="home" asp-action="NuovoPilota">Nuovo pilota</a></li>
        </ul>
    </menu>
    <div>
        @RenderBody()
    </div>
</body>
</html>
```

La pagina referenzia due elementi statici, il foglio di stile e l'immagine del banner. Entrambi sono memorizzati nella cartella **wwwroot**, all'interno della quale sono presenti anche le foto dei piloti.



4.4 Home page

Sia la *view* **Index** che il metodo `Index()` sono minimali, poiché non devono eseguire alcuna elaborazione:

Index

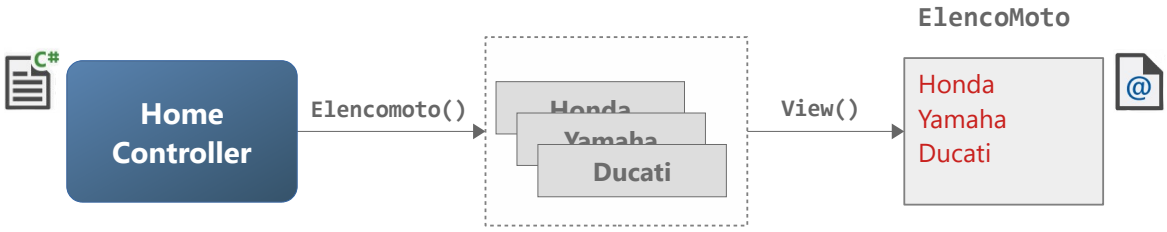
```
@{ViewData["Title"] = "Home";}

<div class="textcenter">
        <img src="~/images/Homefoto.png"/>
</div>
```

Index()

```
public IActionResult Index()
{
    return View();
}
```

4.5 Visualizzare l'elenco delle moto - ElencoMoto

Schematizziamo questa funzione secondo il *pattern* MVC:



- La *view* **ElencoMoto** ha la funzione di visualizzare le moto iscritte al campionato mediante un elenco di *hyperlink*.
- Il *model* utilizzato dalla *view* è pertanto rappresentato da una lista di `Moto`.
- Il *controller* ottiene questo elenco da `MotoGPStore` e lo passa alla *view*.

4.5.1 Implementazione della view

Prima di implementare la *view*, intendo mostrare l'HTML che dovrà essere prodotto:

```
<p class="moto"> <a href="/Home/ElencoPiloti/1">Yamaha</a></p>
<p class="moto"> <a href="/Home/ElencoPiloti/2">Honda</a></p>
<p class="moto"> <a href="/Home/ElencoPiloti/3">Ducati</a></p>
<p class="moto"> <a href="/Home/ElencoPiloti/4">Aprilia</a></p>
```

Ogni link referenzia l'*action* `ElencoPilotiMoto()` e specifica nella *route* l'id della moto visualizzata.

Segue l'implementazione:

```
@{ViewData["Title"] = "Elenco moto";}

@model IEnumerable<Moto>


<div class="textcenter">
    @foreach (var m in Model)  // Model è di tipo IEnumerable<Moto>; "m" è di tipo Moto
    {
        <p class="moto">
           <a asp-controller="home" asp-action="ElencoPiloti"
              asp-route-id="@m.MotoId">@m.Nome</a>
        </p>
    }
</div>
```

Nota bene: alternativamente avrei potuto scrivere:

```
<a href="/Home/ElencoPiloti/@m.MotoId">@m.Nome</a>
```

4.5.2 Controller: ottenere il model e passarlo alla view

Il compito del *controller* è quello di usare il *repository* per ottenere i dati richiesti e quindi passarli alla *view*:

```
public class HomeController : Controller
{
    MotoGPRepository repo = new MotoGPRepository();
    ...
    public IActionResult ElencoMoto()
    {
        return View(repo.GetMoto());  // passa l'elenco delle moto alla view
    }
}
```

4.6 Visualizzazione dell'elenco dei piloti - ElencoPiloti

La visualizzazione dell'elenco dei piloti rappresenta un esempio di come il pattern MVC faciliti la separazione tra elaborazione delle richieste e presentazione dei dati. L'elenco dei piloti può essere visualizzato in due circostanze, che producono elenchi diversi:

- Cliccando sulla voce **Piloti** del menù: elenco completo dei piloti.
- Nella *view* **ElencoMoto** cliccando sul nome di una moto: elenco dei piloti che corrono con la moto selezionata.

Ma dal punto di vista della *view* **ElencoPiloti**, ciò non ha importanza, poiché si limita a ricevere un elenco e a visualizzarlo.

```
@{ViewData["Title"] = "Elenco piloti";}

@model IEnumerable<Pilota>
<div>

    <table class="center grid">
        <thead>
            <tr>
                <th>Nominativo</th>
                <th>Moto</th>
                <th>N°</th>
                <th class="textright">Vittorie</th>
                <th class="textright">Punti</th>
            </tr>
        </thead>
        @foreach (var p in Model) //Model è di tipo IEnumerable<Pilota>,
        {                         "p" è di tipo Pilota
            <tr>
                <td><a asp-controller="home" asp-action="InfoPilota"
                       asp-route-id="@p.PilotaId">@p.Nominativo</a></td>
                <td>@p.Moto.Nome</td>
                <td>@p.Numero</td>
                <td class="textright">@p.Vittorie</td>
                <td class="textright">@p.Punti</td>
            </tr>
        }
    </table>
</div>
```

Nota bene: il nome del pilota viene visualizzato mediante un *hyperlink*, in modo che l'utente possa selezionarlo per accedere alla pagina di informazioni sul pilota.

4.7 Caricamento dei piloti: *databinding* con parametro facoltativo

Poiché esistono due richieste distinte che devono produrre un elenco di piloti:

`home/ElencoPiloti>` (tutti i piloti)

e

`home/ElencoPiloti/<id>` (piloti della moto <id>)

occorre stabilire come gestirle sul *controller*. La prima idea potrebbe essere quella di definire due metodi con lo stesso nome, uno dei quali dichiara un parametro di nome `id`:

```
public class HomeController : Controller
{
    ...
    //Home/ElencoPiloti
    public IActionResult ElencoPiloti()
    {
        ...
    }
```

```
    //Home/ElencoPiloti/<id>
    public IActionResult ElencoPiloti(int id)
    {
        ...
    }
}
```

Purtroppo questa modalità non funziona: essendo il parametro **id** della *route* facoltativo, il *databinding* non fa distinzione tra i due metodi, i quali sarebbero entrambi eleggibili a rispondere ad ognuna delle due richieste.

4.7.1 Definire un metodo action con parametro id nullable

La soluzione è quella di definire un solo metodo, che dichiari un parametro `id` nullabile.

```
public class HomeController : Controller
{
    MotoGPRepository repo = new MotoGPRepository();
    ...
    public IActionResult ElencoPiloti(int? id)
    {
        if (id == null)
            return View(repo.GetPiloti());       //Home/ElencoPiloti/
        return View(repo.PilotiMoto(id.Value));  //Home/ElencoPiloti/<id>
    }
}
```

Nel processare la richiesta **home/ElencoPiloti** il meccanismo *databinding* assegnerà `null` al parametro id (non essendo specificato alcun valore nella richiesta).

4.8 Informazioni sul pilota - InfoPilota

La pagina contenente le informazioni sul pilota viene caricata in risposta al click sul pilota nella *view* **ElencoPiloti**:

```
...
     <a asp-controller="home" asp-action="InfoPilota"
        asp-route-id="@p.PilotaId">@p.Nominativo</a>
...
```

Il metodo *action* `InfoPilota()` riceve l'id, carica il pilota corrispondente e lo passa alla *view* omonima:

```
public class HomeController : Controller
{
    MotoGPRepository repo = new MotoGPRepository();
    ...
    public IActionResult InfoPilota(int id)
    {
        return View(repo.GetPilota(id));
    }
}
```

La *view* dichiara il tipo `Pilota` come *model* e ne visualizza le proprietà:

```
@{ViewData["Title"] = "Pilota";}
@model Pilota
@{
    string statistiche = string.Format("Vittorie: {0} --- Punti: {1}", Model.Vittorie,
                                                                        Model.Punti);
    string moto = string.Format("{0} {1}", Model.Moto.Nome, Model.Numero);
}
<table class="content">

    <tr>
        <th colspan="2"><h1>@Model.Nominativo</h1></th>
    </tr>
    <tr>
        <th colspan="2" class="textcenter">
            <img src="~/FotoPiloti/@Model.FileFoto" />
        </th>
    </tr>
    <tr>
        <td class="textcenter"><h3>@moto</h3></td>
    </tr>
    <tr>
        <td class="textcenter">@statistiche</td>
    </tr>
</table>
```

Nota bene: in questa *view* imposto innanzitutto due variabili stringa contenenti le statistiche sul pilota, che utilizzerò successivamente nel codice HTML. .

5 Inserimento dei dati

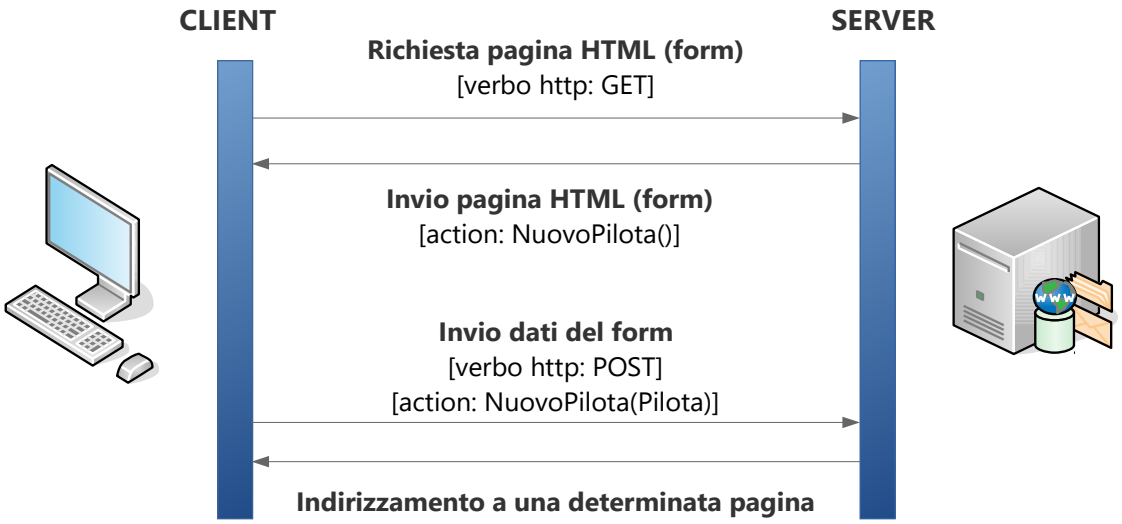
La creazione di un nuovo pilota richiede di implementare:

- Un form HTML per l'inserimento dei dati (Nome, Cognome, etc).
- L'uso di un *combobox* per selezionare la moto del pilota. Questo deve essere popolato con l'elenco delle moto.
- La possibilità di "uploadare" la foto del pilota.

5.1 Gestione di un form HTML

IL funzionamento di un form HTML si suddivide in due fasi:

- Il browser chiede la pagina contenente il form. Il server risponde con un form vuoto.
- Il browser invia al server i dati inseriti dall'utente (invio del form). Il server processa i dati, li verifica e, dopo l'inserimento, reindirizza il browser a una nuova pagina.



Il form è gestito mediante due metodi *action*. Il primo metodo carica la *view* contenente il form; il secondo riceve i dati inseriti nel form e procede all'inserimento. Segue il primo dei due metodi:

```
public class HomeController : Controller
{
    MotoGPRepository repo = new MotoGPRepository();

    [HttpGet]
    public IActionResult NuovoPilota()
    {
        return View();
    }
}
```

L'attributo `[HttpGet]`, benché non obbligatorio, qualifica il metodo come un'*action* che risponde a una richiesta del form da parte del browser. Corrisponde al verbo HTTP **GET**.

5.1.1 Implementazione del form HTML in ASP.NET

Segue la *view* **NuovoPilota**, che, nell'attuale versione, non considera l'input della moto e del file immagine. Nota bene: l'attributo **method** del form specifica l'impiego del verbo **POST** per l'invio dei dati.

```
@{ViewData["Title"] = "Nuovo pilota";}
@model Pilota
<div>
    <form asp-controller="Home" asp-action="NuovoPilota" method="post">
        <table class="center">
            <tr>
                <td><label asp-for="Nome"></label></td>
                <td><input asp-for="Nome"/></td>
            </tr>
            <tr>
                <td><label asp-for="Cognome"></label></td>
                <td><input asp-for="Cognome"/></td>
            </tr>
            <tr>
                <td><label asp-for="Numero"></label> </td>
                <td><input asp-for="Numero" /></td>
            </tr>
            <tr>
                <td><label asp-for="Punti"></label> </td>
                <td><input asp-for="Punti" value="0"/></td>
            </tr>
            <tr>
                <td><label asp-for="Vittorie"></label></td>
                <td><input asp-for="Vittorie" value="0"/></td>
            </tr>
            <tr>
                <td colspan="2" class="textcenter">
                    <input type="submit" value="Crea" />
                </td>
            </tr>
        </table>
    </form>
</div>
```

Nota bene: anche questa *view* dichiara il tipo del *model;* ciò, unito all'uso dei *tag helper*, consente di generare automaticamente il tipo appropriato dei tag di **input**, di inserire i dati nelle proprietà specificate e di impostare automaticamente le **label**.

5.1.2 Processare i dati inseriti

Nel form, il *tag helper* **asp-action** specifica il metodo `NuovoPilota()` come il destinatario dei dati inseriti; ovviamente non si tratta dello stesso metodo che carica la *view*:

```
[HttpPost]
public IActionResult NuovoPilota(Pilota pilota)
{
    repo.NuovoPilota(pilota);
    return RedirectToAction("ElencoPiloti");
}
```

Vi sono tre elementi degni di nota:

- Il metodo è decorato con l'attributo `[HttpPost]`; questo lo identifica come un metodo che riceve i dati inseriti nel form.
- ASP.NET è in grado di "bindare" i dati ricevuti, assegnandoli alle corrispondenti proprietà del parametro `pilota`.
- Dopo aver inserito il pilota nel *repository*, il metodo reindirizza l'utente alla *view* **ElencoPiloti** utilizzando `RedirectToAction()`.

5.2 Selezionare la moto da un elenco

Il modo corretto per l'inserimento della moto del pilota è quello di consentire all'utente di selezionarla da un elenco, implementato mediante un tag **select**. Qui sorge un problema, perché il *model* utilizzato nella *view* è di tipo `Pilota`, e il *record* non definisce l'elenco delle moto. Una soluzione è quella di passare l'elenco alla *view* mediante `ViewData`:

```
public class HomeController : Controller
{
    MotoGPRepository repo = new MotoGPRepository();

    [HttpGet]
    public IActionResult NuovoPilota()
    {
        ViewData["listaMoto"] = repo.GetElencoMoto();
        return View();
    }

    [HttpPost]
    public IActionResult NuovoPilota(Pilota pilota)
    {
        ... // resta invariato
    }
}
```

Nella *view* si memorizza innanzitutto l'elenco in una variabile; successivamente si usa il *tag helper* **asp-items** per generare il tag **select**:

```
@{
    ViewData["Title"] = "Nuovo pilota";
    var listaMoto = ViewData["listaMoto"] as IEnumerable<Moto>;
}
@model Pilota
<div>
    <form asp-controller="Home" ...>
        <table class="center">
            ...
            <tr>
                <td><label asp-for="MotoId">Moto</label></td>
                <td>
                    <select asp-for="MotoId"
                            asp-items="@(new SelectList(listaMoto,"MotoId","Nome"))">
                    </select>
                </td>
            </tr>
            ...
        </table>
    </form>
</div>
```

Nota bene, l'uso di **asp-items** permettere di generare dinamicamente il seguente codice HTML:

```
<select id="MotoId" name="MotoId">
    <option value="1">Yamaha</option>
    <option value="2">Honda</option>
    <option value="3">Ducati</option>
    <option value="4">Aprilia</option>
</select>
```

5.3 Upload del file con la foto del pilota

Per implementare la funzionalità di *upload* della foto del pilota occorre:

- Utilizzare un tag **input** di tipo "file".
- Nel metodo `NuovoPilota()` accedere al file caricato.
- Ottenere il percorso della cartella **FotoPiloti**, collocata in **wwwroot** (la cartella radice per risorse statiche del sito), e copiare il file.

Alla *view* **NuovoPilota** occorre aggiungere il tag **input** per la selezione del file; inoltre, perché sia possibile l'upload, occorre aggiungere l'attributo **enctype** al form:

```
@{
    ViewData["Title"] = "Nuovo pilota";
    var listaMoto = ViewData["listaMoto"] as IEnumerable<Moto>;
}
@model Pilota
<div>
```

```
    <form asp-controller="Home" enctype="multipart/form-data" ...>
        <table class="center">
             ...
            <tr>
                <td><label asp-for="FileFoto">File foto</label></td>
                <td><input type="file" name="file" /></td>
            </tr>
            ...
         </table>
    </form>
</div>
```

Nota bene: al tag **input** deve essere dato un nome ben definito, poiché dovrà essere lo stesso utilizzato nel secondo parametro del metodo `NuovoPilota()`:

```
 using Microsoft.AspNetCore.Http;   // definisce il tipo IFormFile

public class HomeController : Controller
{
    MotoGPRepository repo = new MotoGPRepository();

    [HttpGet]
    public IActionResult NuovoPilota()
    {
        ViewData["listaMoto"] = repo.GetElencoMoto();
        return View();
    }

    [HttpPost]
    public IActionResult NuovoPilota(Pilota p, IFormFile file)
    {
        //... crea pilota e salva file
    }
}
```

Il tipo `IFormFile` memorizza le informazioni relative al file caricato e consente di salvarlo su disco.

5.3.1 Salvare il file su disco

Si suppone di voler salvare il file in una sotto cartella del sito. Occorre innanzitutto conoscere il percorso della cartella **wwwroot**; un modo è quello di accedere all'oggetto *host environment*, che memorizza le informazioni sull'ambiente di esecuzione. L'oggetto viene creato automaticamente da ASP.NET e viene passato al *controller*, purché questo dichiari il costruttore appropriato:

```
 using Microsoft.AspNetCore.Hosting;  // definisce il tipo IWebHostEnvironment

public class HomeController : Controller
{
    MotoGPRepository repo = new MotoGPRepository();
    IWebHostEnvironment hostEnv;
```

```
    public HomeController(IWebHostEnvironment hostEnv)
    {
        this.hostEnv = hostEnv;
    }
    ...
    [HttpPost]
    public IActionResult NuovoPilota(Pilota pilota, IFormFile file)
    {
        if (file != null) // è stato selezionato un file?
        {
            var ext = Path.GetExtension(file.FileName);
            var filePath = string.Format("{0}/FotoPiloti/{1}{2}.{3}", hostEnv.WebRootPath,
                                         pilota.Cognome, pilota.Nome, ext);

            var fs = new FileStream(filePath, FileMode.Create))
            file.CopyTo(fs);  // salva il file sul filestream e dunque su disco
            fs.Close();

            pilota.FileFoto = Path.GetFileName(filePath);
        }

        pilota.Moto = repo.GetMoto(pilota.MotoId);
        repo.NuovoPilota(pilota);
        return RedirectToAction("ElencoPiloti");
    }
}
```

Il codice del metodo `NuovoPilota()`:

- Verifica se è stato caricato un file. In caso positivo:
  - Ottiene l'estensione del file.
  - Genera il percorso di destinazione, utilizzando la proprietà `WebRootPath` per accedere al percorso di **wwwroot**.
  - Crea un `FileStream` e lo usa per salvare il file.
  - Imposta il nome del file nell'oggetto `pilota`.
- Usando la proprietà `MotoId`, ottiene un *reference* alla moto corrispondente.
- Inserisce il pilota nel *repository*.

6 Validare i dati

I controller dovrebbero sempre validare i dati prima di elaborarli. In caso di errore, dovrebbe essere riproposto il form di inserimento, in modo che l'utente possa correggere l'input. A questo proposito, ASP.NET fornisce un meccanismo di validazione che consente:

- Di validare il contenuto dei singoli campi di input, in accordo a determinati criteri.
- Di validare l'input nel suo insieme, verificando che i dati inseriti siano coerenti tra loro e con lo stato dell'applicazione.

In entrambi i casi è possibile stabilire il contenuto e la modalità di visualizzazione dei messaggi di errore.

6.1 Validazione dei singoli campi del pilota

È possibile utilizzare il meccanismo di validazione automatica specificando nel *model* i criteri che i dati devono soddisfare:

```
using System.ComponentModel.DataAnnotations;

public class Pilota
{
    public int PilotaId { get; set; }

    [Required]
    public string Nome { get; set; }

    [Required]
    public string Cognome { get; set; }

    public string Nominativo { get { return Cognome + ", " + Nome; } }
    public int MotoId { get; set; }
    public Moto Moto { get; set; }

    [Range(1, 99)]
    public int Numero { get; set; }

    [Range(0, 450)]
    public int Punti { get; set; }

    [Range(0, 18)]
    public int Vittorie { get; set; }

    public string FileFoto { get; set; }
}
```

Gli attributi stabiliscono i criteri utilizzati per stabilire la validità dei dati inseriti. (Esistono altri tipi di attributi, che consentono un elevato livello di personalizzazione nella validazione dei campi.)

Nel metodo *action* che elabora il form, prima di procedere all'elaborazione dell'input, occorre

verificare che il *model* sia valido:

```
[HttpPost]
public IActionResult NuovoPilota(Pilota pilota, IFormFile file)
{
    if (!ModelState.IsValid)  // se non è valido, carica nuovamente il form
    {
        ViewData["listaMoto"] = repo.GetElencoMoto();
        return View(pilota);
    }

    // ... procede all'inserimento

    return RedirectToAction("ElencoPiloti");
}
```

6.2 Visualizzazione degli errori: uso dei tag helper

In risposta a un input errato, è possibile visualizzare un messaggio di errore riepilogativo e/o dei messaggi corrispondenti ai campi non validi. ASP.NET definisce dei *tag helper* per la visualizzazione degli errori; ne esistono di due tipi:

- **asp-validation-for** è utilizzato per visualizzare errori di validazione dei singoli campi.
- **asp-validation-summary** è utilizzato per visualizzare un riepilogo degli errori e/o dei messaggi personalizzati.

Entrambi i *tag helper*, insieme agli attributi e all'oggetto `ModelState`, consentono un elevato livello di personalizzazione del processo di validazione. L'approccio standard è quello di utilizzare dei tag **span**, adiacenti ai campi di input, per mostrare i singoli errori, e un tag **div** che riepiloghi gli errori e/o visualizzi messaggi sulla validità del modello in generale.

```
@{
    ViewData["Title"] = "Nuovo pilota";
    var listaMoto = ViewData["listaMoto"] as IEnumerable<Moto>;
}
@model Pilota
<div>
    <form asp-controller="Home" enctype="multipart/form-data" ...>
        <table class="center">
            <tr>
                <td colspan="2">
                    <div asp-validation-summary="ModelOnly" class="error-text"></div>
                </td>
            </tr>
            <tr>
                <td><label asp-for="Nome"></label></td>
                <td><input asp-for="Nome"/>
                    <span asp-validation-for="Nome" class="error-text"></span>
                </td>
```

```
            </tr>
            <tr>
                <td><label asp-for="Cognome"></label></td>
                <td><input asp-for="Cognome"/>
                    <span asp-validation-for="Cognome" class="error-text"></span>
                </td>
            </tr>
            <tr>
                <td><label asp-for="Numero"></label> </td>
                <td><input asp-for="Numero" />
                    <span asp-validation-for="Numero" class="error-text"></span>
                </td>
            </tr>
            ...
        </table>
    </form>
</div>
```

Il **div** posto all'inizio ha la funzione di riepilogo. Il valore **ModelOnly** dell'attributo indica che non saranno visualizzati i singoli errori relativi ai campi. Questi vengono visualizzati attraverso dei tag **span**, posizionati accanto ai tag di **input**. Alternativamente, si può decidere di usare soltanto il **div** di riepilogo, specificando il valore **All** per l'attributo, in modo da visualizzare automaticamente tutti gli errori.

6.2.1 Messaggi di errore riepilogativi: aggiungere errori al modello

Se impostato al valore **ModelOnly**, il *tag helper* **asp-validation-summary** non produce alcun output, a meno che non siano aggiunti uno o più errori nel *controller*.

```
[HttpPost]
public IActionResult NuovoPilota(Pilota pilota, IFormFile file)
{
    if (!ModelState.IsValid)  // se non è valido, carica nuovamente il form
    {
        ViewData["listaMoto"] = repo.GetElencoMoto();
        ModelState.AddModelError("", "Uno o più campi contengono dati corretti");
        return View(pilota);
    }
    ...
}
```

Nota bene: il primo parametro del metodo `AddModelError()` è vuoto; ciò distingue un errore di riepilogo rispetto a un errore che riguarda uno specifico campo.

6.2.2 Personalizzare i messaggi di errore

I messaggi di errori prodotti dagli attributi che decorano il modello sono stabiliti da ASP.NET e sono in inglese. È possibile personalizzarli, oppure semplicemente rimpiazzarli con un simbolo, quando la natura dell'errore è evidente.

```
public class Pilota
{
```

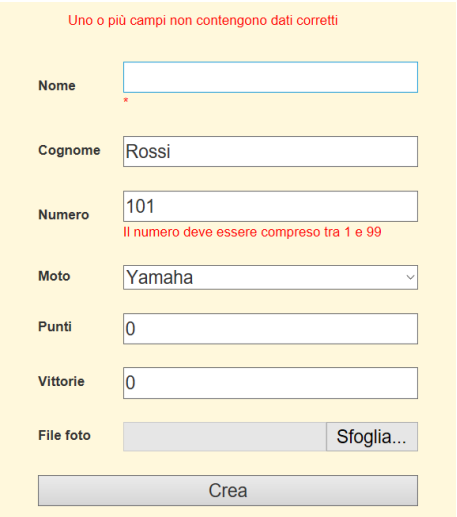
```
    ...
    [Required(ErrorMessage="*")]
    public string Nome { get; set; }

    [Required(ErrorMessage="*")]
    public string Cognome { get; set; }

    [Range(1, 99, ErrorMessage ="Il numero deve essere compreso tra 1 e 99")]
    public int Numero { get; set; }
    ...
}
```

6.3 Validazione generale del modello

Può accadere che i singoli campi del modello siano validi, ma che non lo sia il modello nel suo insieme. È compito del metodo *action* verificare questa eventualità ed eventualmente aggiungere un errore all'oggetto `ModelState`:

```
[HttpPost]
public IActionResult NuovoPilota(Pilota pilota, IFormFile file)
{
    if (!ModelState.IsValid)
    {
        ViewData["listaMoto"] = repo.GetElencoMoto();
        ModelState.AddModelError("", "Uno o più campi non contengono dati corretti");
        return View(pilota);
    }

    if (pilota.Vittorie > 0 && pilota.Punti == 0)
    {
        ViewData["listaMoto"] = repo.GetElencoMoto();
        ModelState.AddModelError("", "Valori incoerenti tra 'Punti' e 'Vittorie'");
        return View(pilota);
    }
    ...
}
```

Segue uno *screen shot* che mostra il risultato del processo di validazione. Il form è stato inviato senza aver inserito il nome, e con un numero della moto non valido:



7 Usare Entity Framework

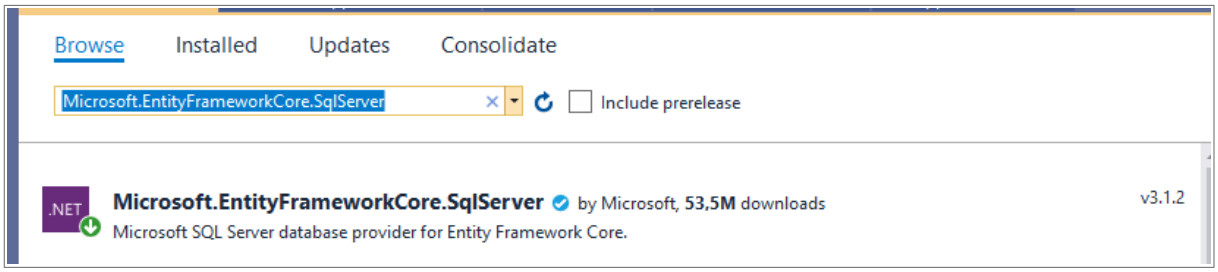
Diversamente dagli altri tipi di applicazione, ASP.NET Core MVC non è compatibile con Entity Framework versione 6; occorre usare **Entity Framework Core**. Questo presenta alcune differenze rispetto a EF 6:

- Implementa diversamente alcune funzionalità, come ad esempio il *lazy loading*.
- Implementa un sistema di configurazione dell'*entity model* leggermente diverso.
- Implementa un diverso meccanismo per gestire la stringa di connessione.
- Ha un'architettura modulare, basata sul concetto di *provider*, che consente selezionare il modulo necessario per dialogare con un determinato DBMS.

Detto questo, i concetti di base appresi su EF 6 restano validi anche per EF Core.

7.1 Aggiungere EF a un progetto ASP.NET Core

Dalla versione ASP.NET Core 3.0, il pacchetto deve essere aggiunto all'applicazione (come è stato fatto con EF 6.0). Si esegue il Nuget Package Manager e, nella scheda **Browse**, si seleziona il provider corrispondente al tipo di DBMS. Nello *screen shot* seguente mostro il pacchetto da installare per interfacciarsi con un database SQL Server:



7.2 Configurare e utilizzare l'oggetto context

La classe `MotoGPContext` fornisce l'accesso al database **PilotiMotoGP**, che memorizza le tabelle **Piloti** e **Moto**.

Innanzitutto occorre istruire il *context* sulla stringa di connessione da usare; un modo è stabilire la stringa di connessione nel metodo `OnConfiguring()`:

```
using Microsoft.EntityFrameworkCore;
...

public class MotoGPContext: DbContext
{
    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        optionsBuilder.UseSqlServer("Server=(localdb)\\mssqllocaldb...");
    }

    public DbSet<Pilota> Piloti { get; set; }
```

```
    public DbSet<Moto> Moto { get; set; }
}
```

7.2.1 Configurazione del model

La configurazione del model richiede la mappatura delle *entità* mediante attributi, poiché la convenzione sui nomi, anglosassone, le assocerebbe alle tabelle **Motos** e **Pilotis**.

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
...
[Table("Moto")]
public class Moto
{
    public int MotoId { get; set; }
    public string Nome { get; set; }
}

[Table("Piloti")]
public class Pilota
{
    public int PilotaId { get; set; }
    ...
    public byte[] Foto { get; set; }

}
```

7.2.2 Uso del context

Consideriamo il metodo *action* `ElencoPiloti()`; l'uso di EF ricalca il pattern già conosciuto:

```
public class HomeController : Controller
{
    IHostingEnvironment hostEnv;
    MotoGPRepository repo = new MotoGPRepository();
    MotoGPContext db = new MotoGPContext();
    ...
    public IActionResult ElencoPiloti()
    {
        return View(db.Piloti.Include(p => p.Moto));
        return View(repo.GetPiloti());
    }
    ...
}
```

Due cose degne di nota:

- il *context* viene dichiarato e creato globalmente, così può essere utilizzato in tutti i metodi del *controller*.
- Mediante il metodo `Include()`, per ogni pilota viene inclusa la moto corrispondente. Si

tratta della tecnica di *eager loading* (usata anche in EF 6), ed è necessaria, poiché, nell'ambito delle applicazioni web, il *lazy loading* è normalmente inutilizzabile.

7.3 Definire un ViewModel

La nuova versione di `Pilota` incorpora l'immagine corrispondente, adesso memorizzata nel database e non più in un file separato[7]. Ciò diminuisce notevolmente le prestazioni della *view* **ElencoPiloti**, poiché ogni pilota memorizza anche l'immagine, nonostante la *view* non la visualizzi.

In questo caso non è conveniente passare direttamente il *model* alla view; è opportuno definire una nuova classe che definisca soltanto le informazioni rilevanti. Si parla in questo caso di *viewmodel*, poiché si tratta di un tipo che definisce sì i dati del *model*, ma è progettato allo scopo di favorire l'implementazione della *view*.

```
public class PilotaInfo
{
    public int PilotaId { get; set; }
    public string Nominativo { get; set; }
    public string NomeMoto { get; set; }
    public int Numero { get; set; }
    public int Punti { get; set; }
    public int Vittorie { get; set; }
}
```

Naturalmente, occorre modificare il *controller*, il quale dovrà restituire un elenco di `PilotiInfo`:

```
// usato dai metodi action ElencoPiloti() e ElencoPilotiMoto()
private IEnumerable<PilotaInfo> GetPilotiInfo(int id = 0) // id->0: tutti i piloti
{
    var piloti = id == 0 ? db.Piloti.Include(p => p.Moto)
        : db.Piloti.Include(p => p.Moto).Where(p => p.MotoId == id);

    return piloti.Select(p => new PilotaInfo
    {
        PilotaId = p.PilotaId,
        Nominativo = p.Nominativo,
        NomeMoto = p.Moto.Nome,
        Numero = p.Numero,
        Vittorie = p.Vittorie,
        Punti = p.Punti
    });
}

public IActionResult ElencoPilotiMoto(int id)
{
    return View("ElencoPiloti", GetPilotiInfo(id));
}
```

7 Qui non discuto sull'opportunità di memorizzare l'immagine nel database. In realtà si tratta di una scelta errata, che diminuisce le performance.

```
public IActionResult ElencoPiloti()
{
    return View(GetPilotiInfo());
}
```

Infine, occorre modificare la *view* **ElencoPiloti**:

```
@{ViewData["Title"] = "Elenco piloti";}

@model IEnumerable<PilotaInfo>

<div>

    <table class="center grid">
        ...
        @foreach (var p in Model)
        {
            ...
                <td>@p.NomeMoto</td>
            ...
        }
    </table>
</div>
```

7.4 Visualizzare le immagini memorizzate nel database

La *view* **InfoPilota** visualizza la foto del pilota mediante un tag **img** che referenzia il file contenente l'immagine, collocato nella cartella **FotoPiloti**; il nome del file è memorizzato nella proprietà `FileFoto` di `Pilota`. Ma se le immagini sono memorizzate nel database e accessibili mediante la proprietà `Foto`, occorre adottare una tecnica diversa per visualizzarle.

Vi sono due possibilità, la prima delle quali, molto semplice, sfrutta una caratteristica del tag **img**: visualizzare un'immagine "incorporata" nella pagina e codificata in **base64string**.

Dunque, è sufficiente modificare il tag **img** della *view* **InfoPilota**:

```
@{ViewData["Title"] = "Pilota";}
...
<table class="content">

    <tr>
        <th colspan="2" class="textcenter"><h1>@Model.Nominativo</h1></th>
    </tr>
    <tr>
        <th colspan="2" class="textcenter">
            <img src="~/FotoPiloti/@Model.FileFoto" />
            <img src="data:image; base64,@System.Convert.ToBase64String(Model.Foto)" />
        </th>
    ...
</table>
```

I byte dell'immagine vengono trasferiti insieme alla pagina e codificati in base64. Questa tecnica, di per sé, non gestisce il caso in cui la proprietà `Foto` sia `null`. Una soluzione consiste nel verificare questa condizione:

```
@{ViewData["Title"] = "Pilota";}
...
<table class="content">

    <tr>
        <th colspan="2" class="textcenter"><h1>@Model.Nominativo</h1></th>
    </tr>
    <tr>
        <th colspan="2" class="textcenter">
            @if (Model.Foto != null)
            {
                <img src="data:image; base64,@System.Convert.ToBase64String(Model.Foto)" />
            }
            else
            {
                <img src="#" />
            }
        </th>
    ...
</table>
```

7.4.1 Implementare un metodo action che restituisce l'immagine

Un'alternativa è implementare un metodo *action* che restituisca l'immagine da visualizzare.

```
public FileContentResult GetFotoPilota(int id)
{
    var p = db.Piloti.Find(id);
    return File(p.Foto, "image/jpg");
}
```

Nota bene: il metodo `File()` restituisce un file incorporato in un oggetto `FileContentResult`.

Il metodo *action* viene richiamato direttamente dal tag **img**, specificando l'URL opportuno e passando l'id del pilota:

```
@{ViewData["Title"] = "Pilota";}
...
<table class="content">

    <tr>
        <th colspan="2" class="textcenter">
            <h1>@Model.Nominativo</h1>
        </th>
    </tr>
    <tr>
        <th colspan="2" class="textcenter">
            <img src="/Home/GetFotoPilota/@Model.PilotaId" />

        </th>
```

```
    ...
</table>
```

7.5 Usare un database in memoria

La modularità di EF Core consente di utilizzare diversi *provider*, e dunque diversi DBMS, nella stessa applicazione. Particolarmente utile è la possibilità di gestire un database completamente in memoria, allo scopo di semplificare e velocizzare le fasi di sviluppo e test dell'applicazione.

A questo scopo è innanzitutto necessario installare il giusto provider, mediante il Nuget Packege Manager, oppure la **Package Manager Console**, mediante il comando:

```
PM>Install-Package Microsoft.EntityFrameworkCore.InMemory
```

> **InMemory database e modello relazionale**
>
> Il provider **InMemory** non implementa un database relazionale e dunque non può imporre il rispetto dei vincoli di integrità referenziale. Se si desidera gestire un database in memoria senza rinunciare al modello relazionale, occorre utilizzare il provider SQLite: **Microsoft.EntityFrameworkCore.Sqlite**.

7.6 Configurare il context in modo che usi l'InMemory provider

Nel metodo `ConfigureServices()` della classe `Startup` scrivere:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<MotoGPContext>(options =>options.UseInMemoryDatabase("MotoGP"));

    services.AddMvc();
}
```

(Il nome fornito al database, `"MotoGP"`, non è significativo.)

7.7 Inserire i dati nel database

Poiché il database è gestito in memoria, quando parte l'applicazione le tabelle sono vuote; per simulare l'esistenza dei dati, questi devono essere inseriti automaticamente all'avvio. A questo scopo di può implementare un metodo che inserisca i dati; questo sarà eseguito in `Configure()` della classe `Startup`:

```
public void Configure(IApplicationBuilder app, ...)
{
    ...
    var db = app.ApplicationServices.GetService<MotoGPContext>();
    GeneraDatabase(db);
}
```

L'istruzione evidenziata ottiene un oggetto *context*, sulla base della configurazione effettuata nel metodo `ConfigureServices()`:

```
private static void GeneraDatabase(MotoGPContext db)
{
    if (db.Moto.Any()) //se ci sono già i dati, termina metodo
        return;

    //... inserisce moto e piloti
    db.SaveChanges();
}
```

8 Configurare l'applicazione

ASP.NET Core implementa un meccanismo di configurazione modulare che consente di stabilire i servizi utilizzati dall'applicazione. Di seguito ne introduco la struttura generale e fornisco un esempio di configurazione dell'oggetto *context* usato per accedere al database.

8.1 Classe Startup

Il codice di avvio di una applicazione ASP.NET è collocato nel metodo `Main()` dei file **Program.cs**: questo costruisce ed esegue l'oggetto che rappresenta il server web (e/o che dialoga con esso, se viene utilizzato un server web esterno, come IIS o Apache.)

```
public class Program
{
    public static void Main(string[] args)
    {
        CreateHostBuilder(args).Build().Run();
    }

    public static IHostBuilder CreateHostBuilder(string[] args) =>
        Host.CreateDefaultBuilder(args)
            .ConfigureWebHostDefaults(webBuilder =>
            {
                webBuilder.UseStartup<Startup>();
            });
}
```

Viene anche stabilita la classe che conterrà il codice di configurazione: `Startup`:

```
public class Startup
{
    public Startup(IConfiguration configuration)
    {
        Configuration = configuration;
    }

    public IConfiguration Configuration { get; }

    public void ConfigureServices(IServiceCollection services)
    {
        // ... definisce i servizi usati dall'applicazione
    }

    public void Configure(IApplicationBuilder app, IHostingEnvironment env)
    {
        // ... configura i servizi usati dall'applicazione
    }
}
```

8.2 Impostare i servizi utilizzati

La modularità di ASP.NET Core è dimostrata dal fatto che, di default, l'applicazione non fornisce alcun servizio[8]; questi devono essere definiti nel metodo `ConfigureServices()`:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddControllersWithViews();
}
```

8.3 Configurare i servizi

Il metodo `Configure()` configura i servizi specificati nel metodo precedente.

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    if (env.IsDevelopment())  // stabilisce la pagina di errore da mostrare in base al
    {                         // fatto che l'applicazione nello stato "sviluppo" o meno.
        app.UseDeveloperExceptionPage();
    }
    else
    {
        app.UseExceptionHandler("/Home/Error");
    }

    app.UseStaticFiles(); // configura MVC perché possa restituire i file statici (pagine
                          // HTML, css, immagini, etc.

    app.UseRouting();
    app.UseAuthorization();

    app.UseEndpoints(endpoints =>   // stabilisce la route predefinita utilizzata da MVC
    {
        endpoints.MapControllerRoute(
            name: "default",
            pattern: "{controller=Home}/{action=Index}/{id?}");
    });

}
```

L'ultima istruzione definisce la *route* predefinita utilizzata da MVC per stabilire i *controller* e i metodi *action* da eseguire in risposta alle richieste dell'utente. (Errore: sorgente del riferimento non trovata)

8 Questo vale per il progetto "empty". Il progetto "web application" prevede appunto di aggiungere il servizio che implementa il pattern MVC.

8.4 Configurare il *context* in Startup

Intervenendo sulla classe `Startup` è possibile istruire ASP.NET Core a creare automaticamente il *context* e a passarlo ai *controller* che ne richiedono l'uso. Per farlo occorre aggiungere un nuovo servizio all'applicazione, nel metodo `ConfigureServices()`.

Nell'esempio seguente configuro il context, facendo in modo che usi una determinata stringa di connessione:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<MotoGPContext>(
        options =>options.UseSqlServer("Server = (localdb)\\mssqllocaldb;..."));

    services.AddMvc();
}
```

Perché il *context* possa utilizzare questa modalità di creazione, è necessario che definisca un costruttore appropriato, in grado di ricevere dall'esterno le opzioni di configurazione:

```
public class MotoGPContext: DbContext
{
    public MotoGPContext(DbContextOptions options):base(options) {}
    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        ...
    }
    public DbSet<Pilota> Piloti { get; set; }
    public DbSet<Moto> Moto { get; set; }
}
```

Infine, nei *controller* è necessario aggiungere un parametro al costruttore: sarà ASP.NET, quando crea il *controller*, a costruire il *context* e a passarlo come argomento:

```
public class HomeController : Controller
{
    IHostEnvironment hostEnv;
    MotoGPContext db = new MotoGPContext();
    MotoGPContext db;

    public HomeController(IHostEnvironment hostEnv, MotoGPContext db)
    {
        this.db = db;
        this.hostEnv = hostEnv;
    }
    ...
}
```

8.4.1 Memorizzare la stringa di connessione in appsettings.json

Un secondo vantaggio della creazione del *context* in `Startup` è quello di poter ottenere la stringa di connessione da una qualsiasi delle sorgenti utilizzate per la configurazione dell'applicazione. Di

norma la stringa viene memorizzata nel file **appsettings.json**:

```
{
  "ConnectionStrings": {
    "MotoGPConnection": "Server=(localdb)\\mssqllocaldb;..."
  },

  "Logging": {
    "IncludeScopes": false,
    "LogLevel": {
      "Default": "Debug",
      "System": "Information",
      "Microsoft": "Information"
    }
  }
}
```

Per accedervi è necessario eseguire il metodo `GetConnectionString()` dell'oggetto `Configuration`, passando come argomento la chiave associata alla stringa:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<MotoGPContext>(
        opt => opt.UseSqlServer(Configuration.GetConnectionString("MotoGPConnection")));

    services.AddMvc();
}
```

8.4.2 Conclusioni

Questo approccio ha il vantaggio di centralizzare il codice di creazione e configurazione del *context*. Semplicemente modificando `Startup`, e senza intervenire nella classe *context*, è possibile cambiare la configurazione utilizzata, compresa l'origine del database.

9 Autenticazione

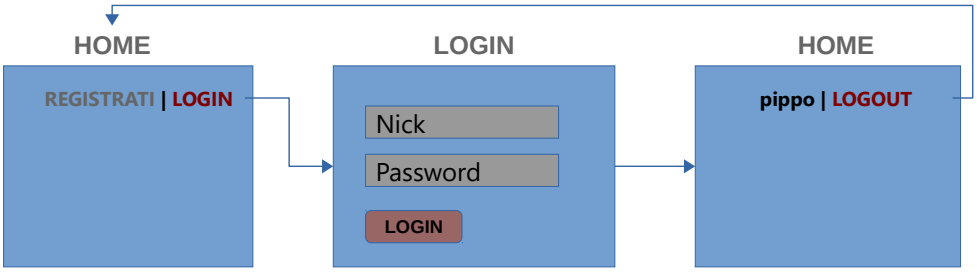
Il termine *autenticazione* disegna il processo con il quale un'applicazione stabilisce l'identità dell'utente, allo scopo di fornire servizi specifici e/o autorizzare (o negare) l'accesso a determinate risorse. Tale processo si basa su alcune premesse:

- L'utente deve essere stato precedentemente registrato.[9]
- L'utente, inizialmente anonimo, deve fornire le proprie credenziali per essere autenticato (*login*). È possibile usare e credenziali di un servizio esterno (Google, Facebook, Microsoft, etc)
- L'applicazione deve memorizzare lo stato dell'utente per tutta la durata della sessione.
- L'applicazione dovrebbe fornire la possibilità all'utente di ritornare anonimo (*logout*)

ASP.NET Core fornisce tutti i servizi necessari al processo di autenticazione; di seguito introduco i fondamenti per implementare i processi di *login*, *logout*, e per conoscere lo stato dell'utente: anonimo/autenticato.

9.1 Implementazione dei processi di login/logout

In generale, i processi di *login* e *logout* possono essere schematizzati nel seguente modo:



In sintesi: l'applicazione consente agli utenti anonimi di autenticarsi e a quelli già autenticati di eseguire il *logout*. *login* è un form HTML che chiede le credenziali dell'utente. L'invio del form produce l'esecuzione del metodo `Login()` dell'*home controller*.

9.1.1 Login

Come ogni form, anche quello di *login* è gestito mediante due metodi; il primo che carica il form, il secondo che ne elabora i dati:

```
using Microsoft.AspNetCore.Authentication.Cookies;
using System.Security.Claims;
using Microsoft.AspNetCore.Http;
...

public class HomeController : Controller
{
```

9 ASP.NET Core fornisce anche un'infrastruttura per la registrazione e la memorizzazione del profilo utente.

```
    public IActionResult Login()
    {
        return View();
    }

    [HttpPost]
    public IActionResult Login(User user)
    {...}
}
```

Il metodo `Login(User)` ha la funzione di:

1. Validare i dati inseriti.
2. Verificare se le credenziali corrispondono a un utente registrato.
3. Eseguire il ***sign-in***: l'utente viene riconosciuto come autenticato.

Di seguito mostro il codice che esegue l'ultima fase:

```
[HttpPost]
public IActionResult Login(User user)
{
    //... valida i dati
    //... verifica esistenza utente
    SignIn(user);
    return RedirectToAction("Index");
}

public void SignIn(User user)
{
  1 string scheme = CookieAuthenticationDefaults.AuthenticationScheme;

    var identity = new ClaimsIdentity(scheme);
  2 identity.AddClaim(new Claim(ClaimTypes.Name, user.FullName));
    var principal = new ClaimsPrincipal(identity);

  3 HttpContext.SignInAsync(scheme, principal).Wait();
}
```

Il metodo `SignIn()`:

1 Stabilisce il tipo di autenticazione: basata su *cookie*.

2 Crea un'*identità* per l'utente da autenticare. A questa identità associa il nome completo dell'utente, memorizzato nel record `user`.

3 Esegue il *sign-in*: ogni successiva richiesta dell'utente (nella stessa sessione) viene associata all'*identità* suddetta.

(Nota bene: il metodo `SignInAsync()` è asincrono e restituisce un *task*; la chiamata al metodo `Wait()` sospende l'esecuzione fintantoché il task non è completato. Non è l'approccio corretto per eseguire un metodo asincrono; qui lo uso soltanto per semplicità.)

9.1.2 Logout

Il *logout* si riduce all'invocazione di un unico metodo:

```
public class HomeController : Controller
{
    ...

    public IActionResult Logout() // chiamato dal link Logout della home page
    {
       string scheme = CookieAuthenticationDefaults.AuthenticationScheme;
       HttpContext.SignOutAsync(scheme).Wait();
       return RedirectToAction("Index");
   }
}
```

Dopo l'esecuzione di `SignOutAsync()`, le successive richieste dell'utente non sono più associate all'*identità* precedentemente creata: l'utente è ritornato ad essere anonimo.

> **HttpContext**
>
> `HttpContext` è una proprietà dell'*home controller* che fornisce l'accesso a tutte le informazioni e servizi relativi al *contesto* della richiesta in corso e, in generale, della sessione dell'utente. Come vedremo, questo oggetto è accessibile (sotto altro nome) anche nelle *view*.

9.2 Visualizzazione dello stato dell'utente

Nell'esempio schematizzato, le informazioni visualizzate in *home* dipendono dal fatto che l'utente sia autenticato oppure no. A questo scopo occorre ottenere l'identità associata all'utente e quindi verificare se è autenticata o anonima.

```
@using Microsoft.AspNetCore.Http

@{
    var identity = Context.User.Identity;
}

<div>
    @if (identity.IsAuthenticated)
    {
        <span>@identity.Name | </span>
        <a asp-controller="Home" asp-action="Logout">Logout</a>
    }
    else
    {
        <a asp-controller="Home" asp-action="Login">Login</a>
        <a asp-controller="Home" asp-action="Register">Register</a>
    }
</div>
```

Alcune considerazioni:

- `Contex`, pur con nome diverso, referenzia lo stesso oggetto della proprietà `HttpContext` usata nell'*home controller*.
- `Name` (proprietà di `User.Identity`) memorizza il nome dell'utente, impostato nel metodo `SignIn()` dall'istruzione:

```
dentity.AddClaim(new Claim(ClaimTypes.Name, user.FullName));
```

9.3 Configurazione del servizio di autenticazione

Il servizio di autenticazione, come qualsiasi altro servizio di ASP.NET Core, deve essere configurato nella classe `Startup`.

```
public class Startup
{
    ...
    public void ConfigureServices(IServiceCollection services)
    {
        string scheme = CookieAuthenticationDefaults.AuthenticationScheme;
        services.AddAuthentication(scheme).AddCookie();
        ...
        services.AddMvc();
    }

    public void Configure(IApplicationBuilder app, IHostingEnvironment env)
    {
        ...
        app.UseStaticFiles();

        // questa istruzione deve precedere UseEndPoints()
        app.UseAuthentication();

        app.UseEndpoints(endpoints =>
        {
            endpoints.MapControllerRoute(
                name: "default",
                pattern: "{controller=Home}/{action=Index}/{id?}");
        });
}
```

Nota bene: il metodo `AddCookie()` esiste in più versioni, e consente di personalizzare il servizio di autenticazione.

10 Autorizzazione

Il termine *autorizzazione* designa il processo che stabilisce le risorse rese accessibili, o negate, a un determinato utente.

Ad esempio, in un sito che implementa una biblioteca virtuale, gli utenti potranno consultare il catalogo dei libri ed eventualmente prenotare dei prestiti. Il personale amministrativo, d'altra parte, avrà anche la facoltà di accedere alle funzioni del sito dedicate all'inserimento e modifica dei contenuti.

Pur essendo una funzione ortogonale all'autenticazione (la seconda può esistere senza la prima), l'autorizzazione richiede che sia implementato il meccanismo di autenticazione, in modo da poter identificare gli utenti, per distinguerli tra di loro e distinguerli da quelli anonimi.

La funzione, come tutte le funzioni di ASP.NET Core, deve essere esplicitamente attivata in `Startup`, nel metodo che configura l'applicazione:

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    ...
    app.UseAuthentication();
    app.UseAuthorization();
    ...
}
```

10.1 Autorizzazione semplice

L'infrastruttura di autorizzazione di ASP.NET Core permettere di utilizzare dei criteri per consentire o proibire l'accesso a determinate risorse. Questi possono basarsi sul *ruolo* assegnato a un utente ("amministratore", "visitatore", etc), oppure su delle ***attestazioni***, (*claim*) informazioni sull'utente - nome, data di nascita, e-mail, telefono - che potranno essere usate per regolare l'accesso alle risorse.

La forma di autorizzazione base, comunque, è quella che distingue semplicemente tra utenti autenticati e utenti anonimi.

10.2 Attributo *Authorize*: autorizzare l'esecuzione di un *action method*

In ASP.NET Core le risorse sono prodotte dagli *action method*; si può negare l'esecuzione di un metodo a un utente anonimo semplicemente decorandolo con l'attributo `Authorize`.

```
public class HomeController : Controller
{
    ...
    [Authorize]
    public IActionResult ModificaCarrello()
    {
        ...
   }
}
```

Se un utente anonimo tenta di eseguire il metodo (cliccando su un link, oppure digitando direttamente l'URL nella barra degli indirizzi), sarà rediretto automaticamente al processo di *login*.

Se il meccanismo di autenticazione non è stato implementato, o non è configurato correttamente, sarà prodotto un errore.

10.3 Configurazione delle *route* di *login* e *logout*

Di default, ASP.NET presuppone che il processo di autenticazione sia implementato dal metodo `Login()` dell'`AccountController`. Se la *route* o il nome del metodo di autenticazione non rispecchiano le impostazioni predefinite, occorre configurare ASP.NET con le nuove impostazioni:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddAuthentication(CookieAuthenticationDefaults.AuthenticationScheme)
            .AddCookie();

    services.AddAuthentication(CookieAuthenticationDefaults.AuthenticationScheme)
        .AddCookie(options =>
        {
            options.LoginPath = "/home/login";
            options.LogoutPath = "/home/logout";
        });
    ...
}
```

10.4 Ritornare automaticamente alla risorsa richiesta

Di norma, quando l'utente naviga a una risorsa che richiede autenticazione (e dunque deve soddisfare il processo di autorizzazione semplice), dopo essersi autenticato nella pagina di *login* dovrebbe essere rediretto alla risorsa richiesta e non alla home o a qualsiasi altra pagina del sito.

A questo scopo è possibile sfruttare una funzione offerta da ASP.NET Core quando l'utente viene diretto al metodo `Login()` in risposta alla richiesta di una risorsa da autorizzare. Infatti, il link generato contiene una *querystring* che specifica l'URL della risorsa che era stata richiesta:

```
<URL metodo login>?ReturnUrl=<URL risorsa richiesta>
```

Ad esempio, posto il codice scritto finora, se l'utente naviga a: */Home/ModificaCliente*, ASP.NET Core lo indirizza a:

```
/home/login?ReturnUrl=/home/ModificaCliente
```

Nella gestione dell'indirizzo a cui redirigere l'utente devono partecipano sia i due metodi di `Login()` che il form di *login*.

(Alternativamente si potrebbe memorizzare l'indirizzo nella sessione dell'utente, per poi utilizzarlo successivamente.)

10.4.1 Salvare l'indirizzo di ritorno

Poiché il `Login()` che carica il form di *login* potrebbe ricevere l'indirizzo di ritorno, deve dichiarare un parametro appropriato:

```
public IActionResult Login(string returnUrl) //returnURL può essere null!
{
    ViewData["ReturnUrl"] = returnUrl;
    return View();
}
```

Nota bene: `returnUrl` sarà `null` nel caso in cui `Login()` sia richiesto direttamente dall'utente (in quel caso l'utente sarà rediretto alla home).

Il metodo lo salva in `ViewData` perché possa essere utilizzato successivamente.

10.4.2 Form login: reinviare l'indirizzo di ritorno

Al metodo di `Login()` che elabora le credenziali inserite dall'utente occorre che sia inviato anche l'indirizzo a cui redirigere l'utente. Pertanto è necessario modificare il form di *login*, in modo che specifichi l'indirizzo come ulteriore dato da inviare al server:

```
<form asp-controller="Home" asp-action="Login" method="post"
        asp-route-returnUrl="@ViewData["ReturnUrl"]">
    ...

    <input type="submit" value="Invia" />
</form>
```

Riepilogando: `ViewData` contiene l'indirizzo di ritorno (o `null`), il form di *login* lo invia al server quando viene confermato. Infatti, il codice precedente viene tradotto in:

```
<form method="post" action="/Home/Login?returnUrl=/Home/ModificaCliente">
    ...
</form>
```

10.4.3 Uso di returnURL nel metodo di autenticazione

Il metodo di autenticazione usare `returnURL` per redirigere l'utente, mediante il metodo `Redirect()`, ma soltanto se contiene un valore; in caso contrario significa che l'utente aveva richiesto direttamente di autenticarsi, pertanto sarà rediretto alla home.

```
[HttpPost]
public IActionResult Login(LoginViewModel model, string returnUrl)
{
    ...
    if (returnUrl == null)
        return RedirectToAction("Index");
    return Redirect(returnUrl);
}
```

IISS “E.Fermi” Bibbiena
A.S. 2020/2021

Schemi Costituzionali di Educazione Civica

FORMA DI STATO

Rapporto tra chi detiene il potere e coloro che ne rimangono assoggettati, e quindi il vario modo di realizzarsi della correlazione tra autorità e libertà (MORTATI)

Il rapporto che corre tra le autorità dotate di potestà di imperio e la società civile (BIN-PITRUZZELLA)

FORMA DI GOVERNO

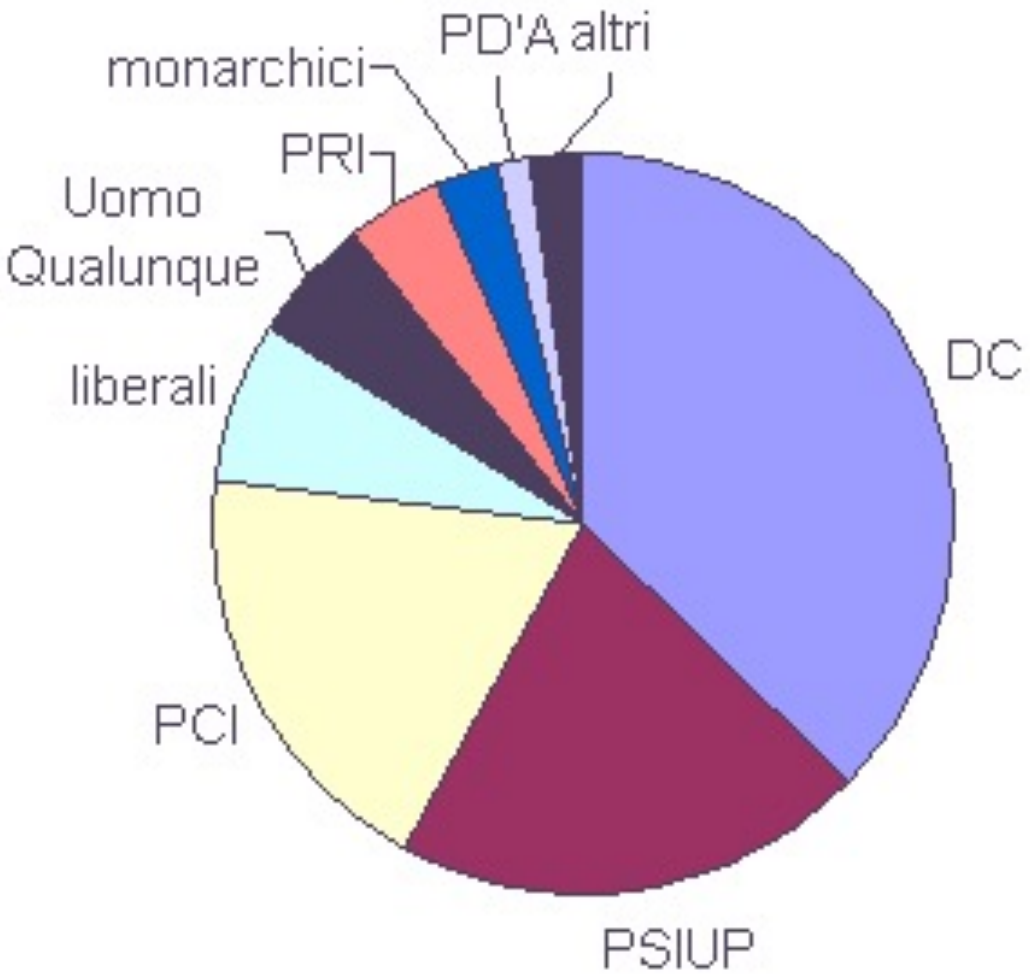
Modo in cui le funzioni dello Stato sono distribuite fra i vari organi costituzionali, con particolare riguardo all'attività di indirizzo politico e ai modi del suo svolgimento (criterio *formale*: si guarda al modo di formazione e ai poteri di indirizzo politico)

Esame del reale atteggiarsi dei rapporti tra gli organi costituzionali, della loro capacità di incidere sull'indirizzo politico e del loro condizionamento da parte delle forze politico sociali (criterio *sostanziale*)

Dall'Assemblea costituente alla Costituzione

Il fascismo crolla il 25 luglio 1943, quando il Gran consiglio destituisce Mussolini attribuendo il comando del Paese al re.

Il maresciallo Badoglio viene nominato capo del Governo e vengono soppresse tutte le istituzioni introdotte durante il periodo fascista (partito, Gran consiglio, camera dei fasci e delle corporazioni).



Composizione dell'Assemblea costituente

La Costituzione è suddivisa in tre parti

- una premessa che contiene i ***Principi fondamentali*** su cui si basa il nostro sistema politico e sociale (art. 1-12);
- una prima parte che riguardai ***Diritti e i doveri dei cittadini*** nell'ambito dei rapporti civili (art. 13-28), dei rapporti etico-sociali (art. 29-34), dei rapporti economici (art. 35-47) e dei rapporti politici (art. 48-54);
- Una seconda parte dedicata all'***Ordinamento della Repubblica*** (art. 55-139), cioè agli organi istituzionali: Parlamento, Presidente della Repubblica, Governo, Magistratura, Regioni, Province, Comuni, Corte Costituzionale

STRUTTURA DELLA COSTITUZIONE ITALIANA

139 ARTICOLI

PRINCIPI FONDAMENTALI

1. PRINCIPIO DEMOCRATICO
2. INVIOLABILITÀ DEI DIRITTI FONDAMENTALI
3. PRINCIPIO DI UGAUGLIANZA
4. DIRITTO-DOVERE AL LAVORO
5. PRINCIPIO DI DECENTRAMENTO
6. TUTELA DELLE MINORANZE LINGUISTICHE
7. RAPPORTI TRA STATO E CHIESA CATTOLICA
8. LIBERTÀ RELIGIOSA
9. TUTELA DELLA CULTURA, DELLA RICERCA E DEL PATRIMONIO AMBIENTALE
10. TUTELA DEGLI STRANIERI
11. TUTELA DELLA PACE
12. LA BANDIERA

PARTE PRIMA	**DIRITTI E DOVERI DEI CITTADINI**	**ART. 13 – 54**
PARTE SECONDA	**ORDINAMENTO DELLA REPUBBLICA**	**ART. 55 – 139**

Costituzione legge suprema

La Costituzione è gerarchicamente superiore a tutte le altre leggi, è il patto fondamentale su cui è stata edificata la società italiana.

Dalla lettura della Costituzione nel suo insieme emerge soprattutto un elemento:

- Si tratta di una Costituzione compromesso:

si fonda cioè sull'accordo fra i diversi partiti del Comitato di liberazione nazionale, primi fra tutti quello democristiano, comunista e socialista che hanno accettato un compromesso per creare una repubblica veramente democratica senza guardare agli interessi di parte

LA DIVISIONE DEI POTERI NELL'ORDINAMENTO ITALIANO

La Costituzione stabilisce che in italia i tre poteri più importanti dello Stato (esecutivo, legislativo e giudiziario) siano dati ad organismi diversi, per evitare la concentrazione dei poteri che aveva caratterizzato la dittatura fascista.

POTERE LEGISLATIVO	POTERE ESECUTIVO	POTERE GIUDIZIARIO
Parlamento	Governo	Magistratura
- Camera dei Deputati - Senato della Repubblica	- Consigli dei ministri - Presidente del consiglio	- Giudice - Tribunale

Il Governo

Il Governo è costituito da una pluralità di organi, previsti sia dalla Costituzione sia da leggi ordinarie.

Art. 92.- Il Governo della Repubblica è composto dal Presidente del Consiglio e dai ministri, che costituiscono insieme il Consiglio dei ministri.

Il Presidente della Repubblica nomina il Presidente del Consiglio dei ministri e, su proposta di questo, i ministri.

Il potere esecutivo

Il Presidente del Consiglio (Capo del Governo) e il Governo, formato da vari Ministri, hanno il potere di governare.

Il Governo applica le leggi decise dal Parlamento.

Il Presidente del Consiglio guida la politica generale del governo e organizza il lavoro dei Ministri che sono a capo dei ministeri (difesa, interni, esteri, ecc …).

Come nasce un governo?

Dopo le elezioni il Presidente della Repubblica avvia le consultazioni, ossia ascolta tutti i partiti, poi dice di formare il nuovo governo ad una persona del gruppo politico che ha vinto le elezioni. Questa persona, che diventerà il Capo del Governo, prepara una lista dei ministri che vengono poi nominati dal Presidente della Repubblica. Entro 10 giorni il nuovo governo deve presentarsi davanti alle due camere (deputati e senatori) per avere la fiducia (deve cioè ottenere la maggioranza dei voti dei deputati e dei senatori in entrambe le Camere).

Il Parlamento

La Costituzione pone al centro della vita politica del Paese il Parlamento, l'unico organo eletto direttamente dal popolo a livello nazionale e, perciò, espressione piena della sovranità popolare.

Il potere legislativo

Il Parlamento fa le leggi. Il Parlamento è composto da due camere: la camera dei Deputati (630 deputati) e il Senato della Repubblica (315 senatori). Le due camere svolgono in modo separato gli stessi compiti, questo sistema si chiama bicameralismo perfetto, vuol dire che una legge deve essere discussa e accettata da tutti e due le camere del Parlamento.

Il Parlamento controlla il lavoro del Governo. I deputati e i senatori sono cittadine e cittadini italiani eletti con suffragio universale (voto di tutti gli uomini e le donne maggiori di 18 anni, 25 per Senato). Durano in carica 5 anni.

Come nasce una legge (procedura ordinaria):

I tappa: presentazione di un progetto al Parlamento che deve decidere se portarlo avanti. Il progetto di legge può essere presentato dal Governo, dalle Regioni, dal Consiglio nazionale dell'economia e del lavoro, dai singoli componenti del Parlamento e dal popolo (almeno 50000 elettori).

II tappa: presentazione del progetto a una delle due Camere.

III tappa: si dà la proposta di legge alla Commissione di competenza, viene esaminato, eventualmente modificato e passato in aula.

IV tappa: è di nuovo esaminato, eventualmente modificato e trasmesso all'altra Camera;

V tappa: viene dato alla Commissione di competenza, esaminato, eventualmente modificato e trasmesso in aula;

VI tappa: è di nuovo esaminato, se è approvato nel testo già approvato dalla Camera che lo ha esaminato per prima, la legge viene inviata al Presidente della Repubblica;

VII tappa: la legge è promulgata dal Presidente della repubblica e pubblicata sulla Gazzetta ufficiale.

La Magistratura

Art. 101.- La giustizia è amministrata in nome del popolo.

I giudici sono soggetti soltanto alla legge.

Art. 104.- La magistratura costituisce un ordine autonomo e indipendente da ogni altro potere.

Il Consiglio superiore della magistratura è presieduto dal Presidente della Repubblica.

[…]

Il potere giudiziario

. Questo potere è dato ai giudici (magistrati) che formano la Magistratura. In uno Stato di diritto la legge è uguale per tutti. Il giudice deve usare le leggi e non giudicare secondo l'opinione personale. La Magistratura è un organo indipendente che non dipende dal ministero della Giustizia, ma si autogoverna attraverso il Consiglio Superiore della Magistratura (CSM), che è presieduto dal Presidente della Repubblica.

Si diventa magistrati per concorso pubblico.

NB

Tutti hanno diritto alla difesa. Nessuno può essere considerato colpevole finché non è pronunciata la sentenza finale. Chi è condannato può normalmente ricorrere in appello e poi in Cassazione (la sentenza di quest'ultima è definitiva).

In alternativa una sentenza di grado inferiore diventa definitiva quando è passata in giudicato ,sono cioè decorsi i termini per ricorrere ai gradi superiori

Il Presidente della Repubblica

Art. 83.- Il Presidente della Repubblica è eletto dal Parlamento in seduta comune dei suoi membri. […]

Il Presidente della Repubblica non esercita nessuno dei tre poteri fondamentali, egli rappresenta l'unità della nazione ed ha il compito di vigilare perché i principi democratici della Costituzione vengano sempre rispettati.

Il Presidente della Repubblica

il capo dello Stato e rappresenta la Nazione. Dura in carica sette anni. Il presidente non ha il potere di fare le leggi né di governare, ma ha comunque un fondamentale potere di indirizzo (indirizza l'attività delle camere e del governo).

Egli:

-scioglie il Parlamento e ordina nuove elezioni prima della fine naturale di una legislatura o in caso di crisi di governo;

-pubblica tutte le leggi decise o decide di rinviarle di nuovo al Parlamento per una loro nuova valutazione quando siano reputate contrastanti con i principi fondanti della nostra Costituzione;

-invia messaggi alle Camere nei momenti di crisi per cercare in modo imparziale di indicare possibili strade da seguire nel rispetto della Costituzione.

Principi fondamentali

……la sovranità popolare

Art. 1.

- L’Italia è una Repubblica democratica fondata sul lavoro.

La sovranità appartiene al popolo, che la esercita nelle forme e nei limiti della Costituzione.

…la dignità della persona, i diritti inviolabili dell’uomo, l’eguaglianza

Art. 2.

- La repubblica riconosce e garantisce i diritti inviolabili dell’uomo, sia come singolo sia nelle formazioni sociali ove si svolge la sua personalità, e richiede l’adempimento dei doveri inderogabili di solidarietà politica, economica, sociale.

L’uguaglianza

Articolo 3
-Tutti i cittadini hanno pari dignità sociale e sono eguali davanti alla legge, senza distinzione di sesso, di razza, di lingua, di religione, di opinioni politiche, di condizioni personali e sociali.
(uguaglianza formale)
-E` compito della Repubblica rimuovere gli ostacoli di ordine economico e sociale, che, limitando di fatto la libertà e l'eguaglianza dei cittadini, impediscono il pieno sviluppo della persona umana e l'effettiva partecipazione di tutti i lavoratori all'organizzazione politica, economica e sociale del Paese.
(uguaglianza sostanziale)

….la pace

art. 11.

- L’Italia ripudia la guerra come strumento di offesa alla libertà degli altri popoli e come mezzo di risoluzione delle controversie internazionali; consente, in condizioni di parità con gli altri Stati, alle limitazioni di sovranità necessarie ad un ordinamento che assicuri la pace e la giustizia fra le Nazioni; promuove e favorisce le organizzazioni internazionali rivolte a tale scopo.

….la forma repubblicana

Art. 1.

- L’Italia è una Repubblica democratica fondata sul lavoro.

La sovranità appartiene al popolo, che la esercita nelle forme e nei limiti della Costituzione.

Art. 139.

- La forma repubblicana non può essere oggetto di revisione costituzionale

… il lavoro

Art. 1.- L'Italia è una Repubblica democratica fondata sul lavoro.[…]

Art. 4.- La Repubblica riconosce a tutti i cittadini il diritto al lavoro e promuove le condizioni che rendano effettivo questo diritto.

Ogni cittadino ha il dovere di svolgere, secondo le proprie possibilità e la propria scelta, un'attività o una funzione che concorra al progresso materiale o spirituale della società.

….. la libertà religiosa

Art. 8.

- Tutte le confessioni religiose sono egualmente libere davanti alla legge.

Le confessioni religiose diverse dalla cattolica hanno diritto ad organizzarsi secondo i propri statuti, in quanto non contrastino con l'ordinamento giuridico italiano.

I loro rapporti con lo Stato sono regolati per legge sulla base di intese con le relative rappresentanze.

….lo sviluppo culturale, scientifico e tecnologico

Art. 9.

- La Repubblica promuove lo sviluppo della cultura e la ricerca scientifica e tecnica.

Tutela il paesaggio e il patrimonio storico artistico della Nazione.

….le norme del diritto internazionale

Art. 10.

- L’ordinamento giuridico italiano si conforma alle norme del diritto internazionale generalmente riconosciute.

  La condizione giuridica dello straniero è regolata dalla legge in conformità delle norme e dei trattati internazionali.

  Lo straniero, al quale sia impedito nel suo paese l’effettivo esercizio delle libertà democratiche garantite dalla Costituzione italiana, ha diritto d’ asilo nel territorio della Repubblica, secondo le condizioni stabilite dalla legge.

  Non è ammessa l’estradizione dello straniero per reati politici.

….. riconoscimento delle autonomie locali

Art. 5.- La Repubblica, una e indivisibile, riconosce e promuove le autonomie locali; attua nei servizi che dipendono dallo Stato il più ampio decentramento amministrativo; adegua i principi ed i metodi della sua legislazione alle esigenze dell’autonomia e del decentramento.

… rapporti civili (art. 13→28)

In questa prima parte, la Costituzione garantisce l'inviolabilità delle libertà personali. Nessun cittadino può essere arrestato o perquisito arbitrariamente; il domicilio è inviolabile. Tutti i cittadini possono manifestare e sostenere pubblicamente le proprie opinioni non solo con la parola, ma anche a mezzo stampa o tramite un qualunque altro mezzo di comunicazione di massa, come la radio o la televisione.

…… rapporti etico-sociali (art. 29→34)

La Costituzione afferma il fondamentale ruolo della famiglia e stabilisce la sostanziale uguaglianza dei coniugi nel matrimonio; definisce la salute un diritto fondamentale dell'individuo e si impegna a garantire cure gratuite ai poveri.

… la scuola

Art. 33.- L’arte e la scienza sono libere e libero ne è l’insegnamento.

La Repubblica detta le norme generali sull’istruzione ed istituisce scuole statali per tutti gli ordini e gradi.

Enti e privati hanno il diritto di istituire scuole ed istituti di educazione, senza oneri per lo Stato. […]

…. rapporti economici (art. 35→47)

Questa parte della Costituzione tutela i diritti dei lavoratori, uomini e donne; riconosce la proprietà privata; pone però vincoli e limiti alla proprietà terriera privata, per evitare il fenomeno del latifondismo.

…. rapporti politici (art. 48→54)

Il legame esistente tra l'individuo e lo Stato comporta il diritto e il dovere di ogni cittadino di partecipare alla vita e allo sviluppo del Paese.

Questa partecipazione si realizza attraverso il diritto di voto, di candidarsi alle elezioni, di associarsi in partiti e, quindi, di esprimere le proprie idee e valutazioni sulla politica nazionale.

Ordinamento della Repubblica (art. 55→139)

Questa seconda parte della Costituzione è la più ampia e tratta temi della massima importanza per garantire il buon funzionamento della macchina dello stato e il corretto rapporto fra i poteri statali.

La Corte Costituzionale

Tutti i Paesi che adottano Costituzioni rigide sono dotati di strumenti di garanzia costituzionale, volti a controllare la corretta applicazione dei principi sanciti dalla legge fondamentale e a verificare che i supremi organi dello Stato operino in conformità ad essi.

In Italia tali compiti di controllo sono affidati alla Corte costituzionale.

Competenze della Corte costituzionale

art. 134 Cost + L cost. 1/53

Controllo di costituzionalità delle leggi

- **In via incidentale**

(la quaestio sorge come incidente nel corso di un processo, il giudice di merito sospende il giudizio e formula l'ordinanza di rimessione alla Corte)

oppure

- **In via principale**

(impugnazione diretta: di L statale da parte della Regione oppure di L regionale da parte dello Stato)

Conflitti di attribuzione

- **fra poteri dello Stato**

(interorganici)

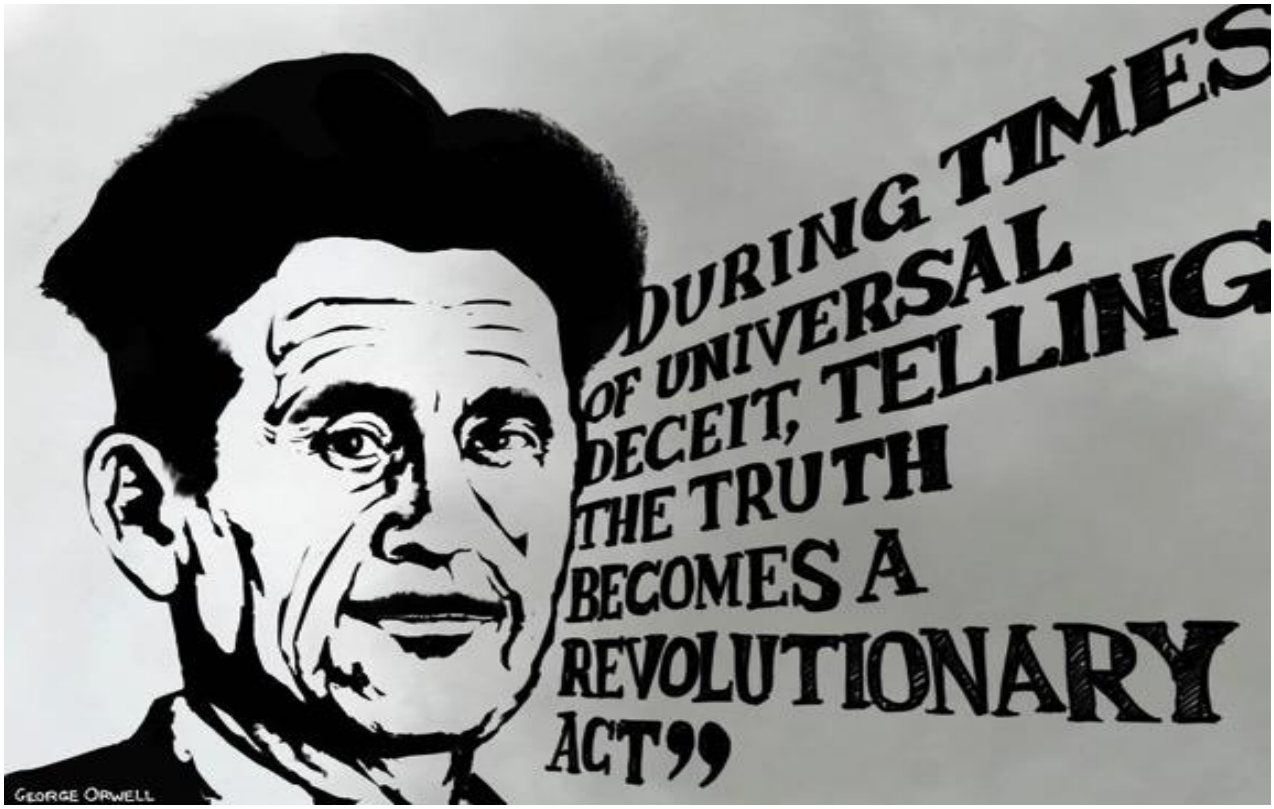
oppure

- **fra Stato e Regioni**

(intersoggettivi)

Giudizi d'accusa

Giudizio ammissibilità referendum abrogativo


"DURING TIMES
OF UNIVERSAL
DECEIT, TELLING
THE TRUTH
BECOMES A
REVOLUTIONARY
ACT"
GEORGE ORWELL

Dulce et Decorum Est

BY WILFRED OWEN

Bent double, like old beggars under sacks,
Knock-kneed, coughing like hags, we cursed through sludge,
Till on the haunting flares we turned our backs,
And towards our distant rest began to trudge.
Men marched asleep. Many had lost their boots,
But limped on, blood-shod. All went lame; all blind;
Drunk with fatigue; deaf even to the hoots
Of gas-shells dropping softly behind.

Gas! GAS! Quick, boys!—An ecstasy of fumbling
Fitting the clumsy helmets just in time,
But someone still was yelling out and stumbling
And flound'ring like a man in fire or lime.—
Dim through the misty panes and thick green light,
As under a green sea, I saw him drowning.

In all my dreams before my helpless sight,
He plunges at me, guttering, choking, drowning.

If in some smothering dreams, you too could pace
Behind the wagon that we flung him in,
And watch the white eyes writhing in his face,
His hanging face, like a devil's sick of sin;
If you could hear, at every jolt, the blood
Come gargling from the froth-corrupted lungs,
Obscene as cancer, bitter as the cud
Of vile, incurable sores on innocent tongues,—
My friend, you would not tell with such high zest
To children ardent for some desperate glory,
The old Lie: *Dulce et decorum est*
Pro patria mori.

The Social Dilemma: a wake-up call for a world drunk on dopamine?
John Naughton

(*The Guardian*, 19 Sept, 2020)

Where the movie fails is in its inability to accurately explain the engine driving this industry that harnesses applied psychology to exploit human weaknesses and vulnerabilities. A few times it wheels on Prof Shoshana Zuboff, the scholar who gave this activity a name – "surveillance capitalism", a mutant form of our economic system that mines human experience (as logged in our data trails) in order to produce marketable predictions about what we will do/read/buy/believe next. Most people seem to have twigged the "surveillance" part of the term, but overlooked the second word. Which is a pity because the business model of social media is not really a mutant version of capitalism: it's just capitalism doing its thing – finding and exploiting resources from which profit can be extracted. Having looted, plundered and denuded the natural world, it has now turned to extracting and exploiting what's inside our heads. And the great mystery is why we continue to allow it to do so.

God of heaven there's nothing like nature
the wild mountains and then the sea,
the waves rushing
and as for them saying that there's no God
well I wouldn't give a snap of me two fingers for all their learning.
who was the first person in the universe
before there was anybody that made it all
who? that they don't know neither do I
so there you are
my little friends try to stop the Sun from rising tomorrow.
The Sun shines for you he said
today we were lying among the rhododendrons on Howth head
in the grey tweed suit and in straw hat
the day I got him to propose to me yes
first I gave him the bit of seedcake out of my mouth
and it was leapyear like now yes
16 years ago,
my God, after that long kiss I near lost my breath yes
he said I was a flower of the mountain yes
and that was one true thing he said in his life
and the Sun shines for you today yes
and that was why I liked him
because I saw he understood or that what a woman is

excerpt from Molly’s monologue

James Joyce, *Ulysses*


IGNORANCE
IS
STRENGTH
WAR
IS
PEACE
FREEDOM
IS
SLAVERY

“ Freedom is the freedom to say that two plus two make four. If that is granted, all else follows. ”

George Orwell

(1984)

Introduction by Guy Berger,[1]

Head of the School of Journalism and Media Studies,
Rhodes University, South Africa

Media freedom is a precondition

To make empowerment into a reality, several conditions are necessary. A legal and regulatory environment must exist that allows for an open and pluralistic mediasector to emerge. Political will to support the sector, and rule of law to protect, it must also exist. There should also be law that ensures practical access to information, especially information in the public domain. Finally, news consumers need the necessary skills to produce and circulate information and engage with the media, and also to critically analyze and synthesize the information they receive.

Ensuring freedom for the media around the world is a priority. Independent, free and pluralistic media are central to good governance in democracies that are young and old. Free media can ensure transparency, accountability and the rule of law; they promote participation in public and political discourse, and contribute to the fight against poverty. An independent media sector draws its power from the community itserves and in return empowers that community to be full a partner in the democratic process


1 NO POVERTY
2 ZERO HUNGER
3 GOOD HEALTH AND WELL-BEING
4 QUALITY EDUCATION
5 GENDER EQUALITY
6 CLEAN WATER AND SANITATION
7 AFFORDABLE AND CLEAN ENERGY
8 DECENT WORK AND ECONOMIC GROWTH
9 INDUSTRY, INNOVATION AND INFRASTRUCTURE
10 REDUCED INEQUALITIES
11 SUSTAINABLE CITIES AND COMMUNITIES
12 RESPONSIBLE CONSUMPTION AND PRODUCTION
13 CLIMATE ACTION
14 LIFE BELOW WATER
15 LIFE ON LAND
16 PEACE, JUSTICE AND STRONG INSTITUTIONS
17 PARTNERSHIPS FOR THE GOALS
SUSTAINABLE DEVELOPMENT GOALS

Tecnologie e Progettazione di Sistemi Informatici e di Telecomunicazioni

Livello di trasporto

Classe V A INF

ISIS “E.Fermi”

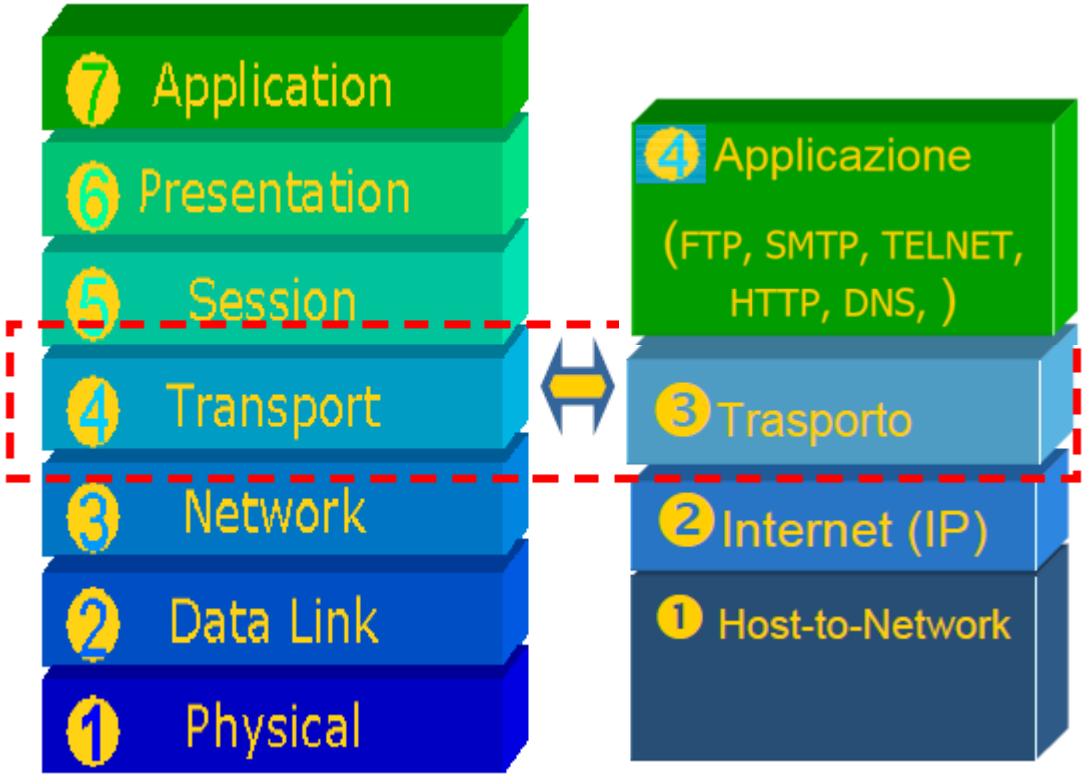
Prof. Federico Santolini

Livello di trasporto e Protocolli TCP/UDP

(a) Funzionalità del livello di trasporto (1/15)

E' utile ricordare che …

- Il compito del livello trasporto è di fornire un trasporto efficace dall'host di origine a quello di destinazione, indipendentemente dalla rete utilizzata



Livello di trasporto e Protocolli TCP/UDP

(a) Funzionalità del livello di trasporto (2/15)

… alcune generalità …

- Il livello di trasporto si occupa di fornire servizi al soprastante livello (di sessione nel ISO/OSI e di Applicazione nel TCP/IP)

- per raggiungere tale scopo raccoglie i dati offerti dal sottostante livello di rete

- lo scopo del livello di trasporto e' anche quello di fornire un canale logico e affidabile di comunicazione end-to-end per "pacchetti"

- il nome trasporto per tale livello puo' quindi trarre in inganno in quanto, non implementa alcun meccanismo di trasferimento logico e fisico dei dati direttamente sul canale

- si occupa di supplire alle mancanze delle funzionalità del trasferimento in termini di affidabilita', implementando alcune funzioni come garanzie sul trasporto stesso

Livello di trasporto e Protocolli TCP/UDP

(a) Funzionalità del livello di trasporto (3/15)

- Il livello di trasporto assolve al compito di garantire un trasferimento dati grazie ad alcune funzionalità (o servizi) , quali:
  1. Controllo di flusso
  2. Controllo delle connessioni (Servizio orientato alla connessione)
  3. Corretto ordine di consegna (Sequenzializzazione)
  4. Trasferimento affidabile
  5. Multiplexing sulle applicazioni (Multiplazione)
  6. Controllo della congestione
  7. Orientamento al Byte

- Nelle slides successive queste funzionalità vengono analizzate e descritte una ad una per facilitarne la comprensione, soprattutto in relazione alla trattazione dettagliata dei protocolli di trasporto TCP e UDP

Livello di trasporto e Protocolli TCP/UDP

(a) Funzionalità del livello di trasporto (4/15)

1. **Controllo di flusso (I)**

➢ Nell'ambito delle reti di TLC il **controllo di flusso**, (oltre al controllo della congestione), è un tipo di controllo di trasmissione effettuato dagli agenti di una comunicazione (mittente e destinatario) sui pacchetti inviati e ricevuti attraverso alcuni protocolli di comunicazione (vedi TCP)

➢Se gli host coinvolti nella comunicazione hanno prestazioni molto differenti , può capitare che un PC più veloce "*inondi*" di dati uno più lento portando alla perdita di pacchetti

➢ Mediante il controllo di flusso, un host in "*difficoltà*" può chiedere di abbassare il tasso di trasmissione in modo da poter gestire le informazioni in ingresso

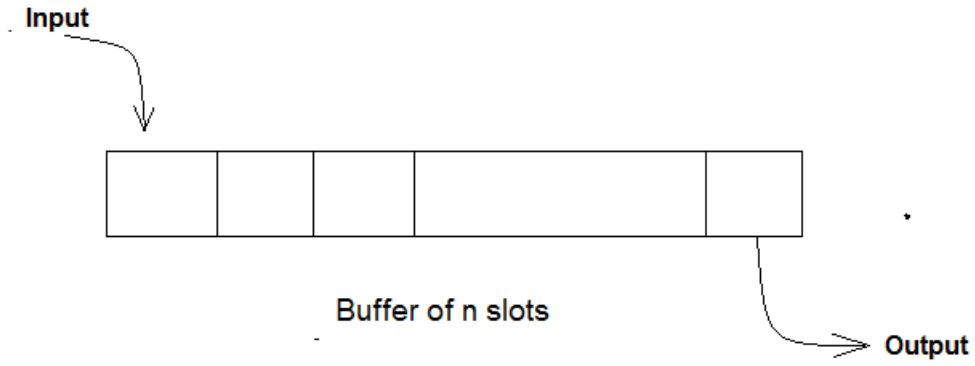
Livello di trasporto e Protocolli TCP/UDP

(a) Funzionalità del livello di trasporto (5/15)

1. Controllo di flusso (II)

- L'obiettivo di tale controllo è evitare che il mittente invii una quantità eccessiva di dati che potrebbero, in alcune situazioni, mandare in *overflow* il *buffer* di ricezione del destinatario generando una perdita di pacchetti e la necessità di ritrasmissione con perdita in efficienza (*Goodput*) a causa delle ritrasmissioni dei pacchetti persi

- Il controllo di flusso risulta dunque particolarmente utile per il mantenimento delle prestazioni della connessione



Livello di trasporto e Protocolli TCP/UDP

(a) Funzionalità del livello di trasporto (6/15)

2. Controllo delle connessioni (o servizio orientato alla connessione)

- E' una modalità di comunicazione dati in cui i dispositivi terminali usano un protocollo di comunicazione per stabilire una <u>connessione logica o fisica</u> (**NON trasporto**) *end-to-end* tra gli agenti della comunicazione prima della trasmissione di qualsiasi tipo di dato

- Si contrappone invece ad una comunicazione senza connessione

- Il livello di trasporto si incarica di realizzare una connessione persistente del tipo necessario al livello di sessione per ogni connessione richiesta, che viene poi chiusa quando non è piu' necessaria

Livello di trasporto e Protocolli TCP/UDP

(a) Funzionalità del livello di trasporto (7/15)

3. Corretto ordine di consegna (I)

- Il messaggio che l'host sorgente deve trasmettere generalmente viene scomposto in pacchetti numerati progressivi
- questi pacchetti vengono immessi in sequenza sulla rete
- a causa della dinamicita' del traffico, non e' detto che tutti percorrano lo stesso canale e quindi arrivino a destinazione nello stesso ordine con cui sono partiti
- E' compito del livello di trasporto effettuare la ricostruzione esatta dei dati rimuovendo possibili errori
- Il livello di trasporto verifica che i pacchetti vengano riordinati nella giusta sequenza in ricezione prima di passarli al livello superiore (sessione)

Livello di trasporto e Protocolli TCP/UDP

(a) Funzionalità del livello di trasporto (8/15)

4. **Trasferimento affidabile**

- Il protocollo si occupa di garantire che tutti i dati inviati vengano ricevuti

- nel caso il servizio di rete utilizzato perda pacchetti, il protocollo di trasporto si occupa dei protocolli per la ritrasmissione dei pacchetti corrotti

Livello di trasporto e Protocolli TCP/UDP

(a) Funzionalità del livello di trasporto (9/15)

5. Multiplazione (I)

- Il protocollo permette di stabilire diverse connessioni contemporanee tra gli stessi due host, tipicamente utilizzando l'astrazione delle porte

- Nell'uso comune diversi servizi utilizzano porte logiche di comunicazione diverse

- La **multiplazione** (***multiplexing***), nei settori delle TLC, elettronica e reti di computer, è il meccanismo o tecnica di trasmissione per cui più canali trasmissivi in ingresso condividono la stessa capacità trasmissiva disponibile in uscita ovvero combinando più segnali analogici o flussi di dati digitali (detti segnali *tributari*) in un solo segnale (detto *multiplato*) trasmesso in uscita su uno stesso collegamento fisico

Livello di trasporto e Protocolli TCP/UDP

(a) Funzionalità del livello di trasporto (10/15)

5. Multiplazione (II)

- In una comunicazione dati la multiplazione permette di risparmiare sul cablaggio (riducendo il numero di linee di segnale) e sul numero di componenti

- in elettronica il multiplexing permette a diversi segnali analogici di essere elaborati da un unico convertitore analogico-digitale (ADC) e in telecomunicazioni a chiamate differenti di essere trasmesse usando un solo cavo

- Non è possibile infatti implementare una rete di telecomunicazioni su grande scala completamente *magliata* (vedi immagine slide 12), in cui ogni coppia di utenti è collegata in modo diretto, dato l'enorme numero di collegamenti *point to point*,

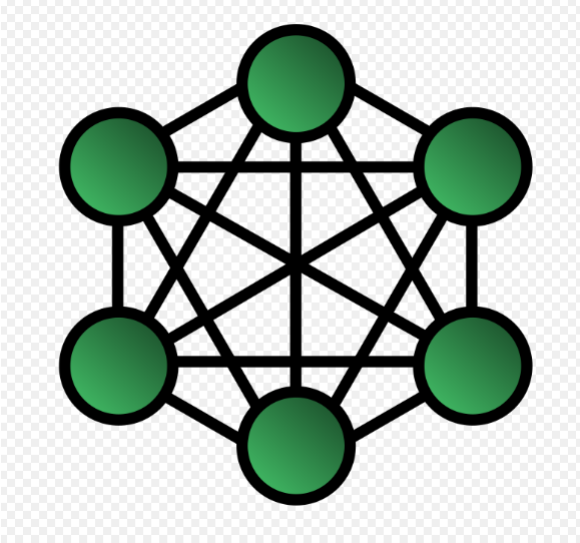
Livello di trasporto e Protocolli TCP/UDP

(a) Funzionalità del livello di trasporto (11/15)

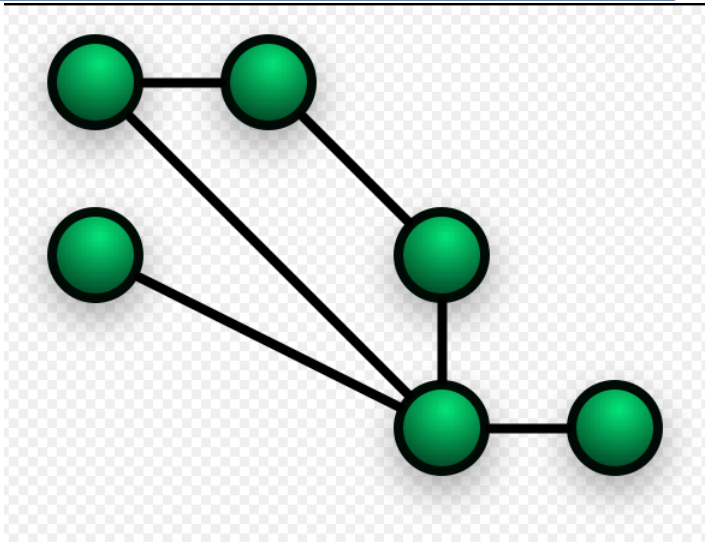
5. Multiplazione (III)

- è quindi necessario pensare a dei meccanismi per far convivere su uno stesso cavo di collegamento (o mezzo trasmissivo) più segnali portanti informativi

- Il dispositivo elettronico preposto alla multiplazione è detto multiplexer



Topologia completamente magliata



Topologia parzialmente magliata

Livello di trasporto e Protocolli TCP/UDP

(a) Funzionalità del livello di trasporto (12/15)

6. Controllo della congestione (I)

- Il protocollo riconosce uno stato di congestione della rete e adatta di conseguenza la velocita' di trasmissione

- Nell'ambito delle reti di telecomunicazioni il **controllo della congestione** è una funzionalità delle reti a commutazione di pacchetto

- lo scopo è prevenire e limitare i fenomeni di congestione che possono verificarsi nei nodi interni di commutazione della rete

- Nelle reti a commutazione di pacchetto, i pacchetti attraversano una grande quantità di dispositivi diversi (router, switch, bridge, ecc…)

Livello di trasporto e Protocolli TCP/UDP

(a) Funzionalità del livello di trasporto (13/15)

6. Controllo della congestione (II)

- Questi dispositivi, e i collegamenti che li interconnettono, hanno capacità di elaborazione e di trasmissione finite

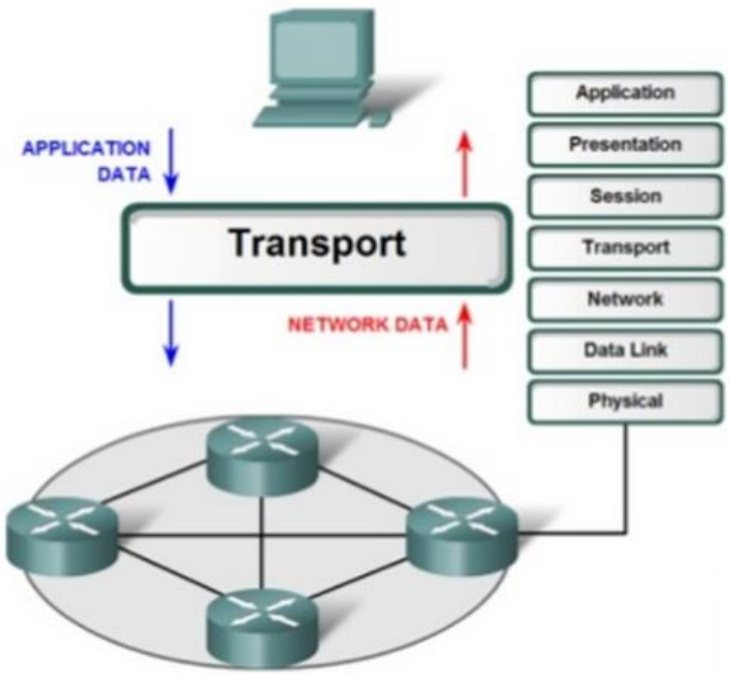
- queste *"finitezze"* possono portare a situazioni di congestione, nelle quali i dispositivi suddetti non sono in grado di smistare tutto il traffico offerto in ingresso da varie connessioni tra utenti causando perdita di pacchetti e/o eccessivi ritardi

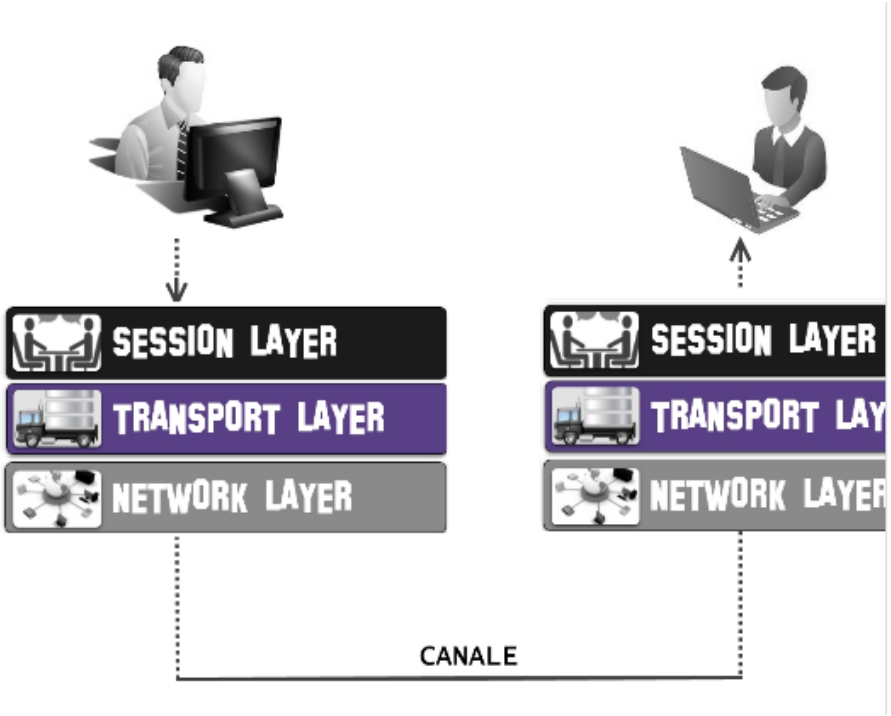
Livello di trasporto e Protocolli TCP/UDP

(a) Funzionalità del livello di trasporto (14/15)

7. Orientamento al Byte

➢ Invece che gestire i dati in base ai pacchetti, viene fornita la possibilità di vedere la comunicazione come uno stream di byte, in modo da semplificarne l'utilizzo





Livello di trasporto e Protocolli TCP/UDP
(a) Funzionalità del livello di trasporto (15/15)

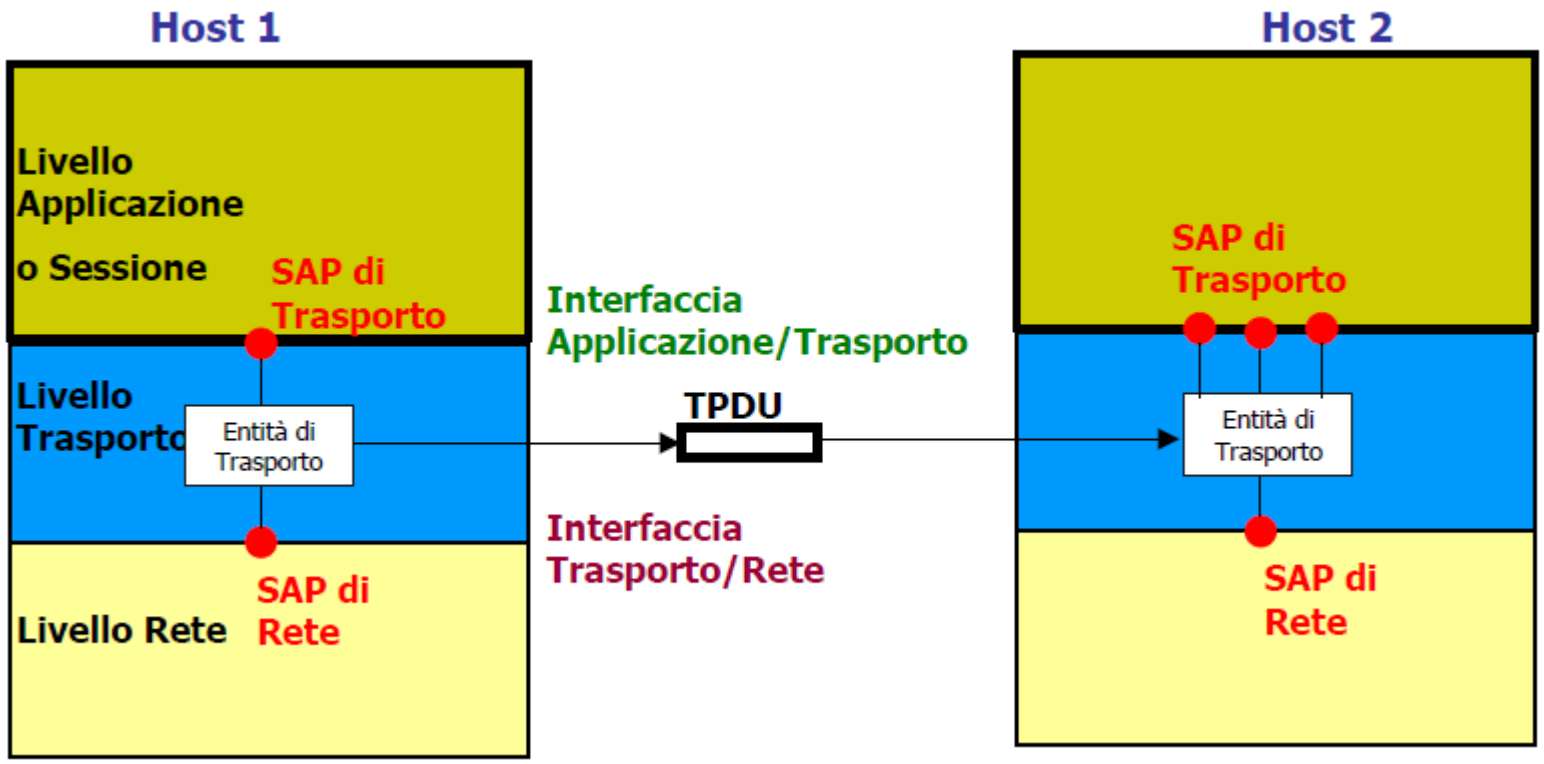
Conclusioni

- I servizi (o funzionalità) precedentemente descritti non sono obbligatori
- Di conseguenza, per ciascuna applicazione è possibile scegliere il protocollo più adatto allo scopo
- In altre parole esistono un'infinità di protocolli relativi al livello di trasporto, noti appunto come *Protocolli di Trasporto*
- Ciascun protocollo presenterà caratteristiche (servizi) comuni ad altri protocolli e peculiarità (servizi implementati in modalità personalizzata '*custom*') sviluppate *ad hoc* per risolvere particolari situazioni tipiche dell'applicazione in analisi

Livello di trasporto e Protocolli TCP/UDP

(b) Servizio di trasporto (1/6)

➢ Nel sottostante schema di principio relativo ad un generico servizio di trasporto emergono alcune "figure" chiave del livello di trasporto,ovvero: Entità di trasporto, SAP e TPDU



- Schema di principio del generico servizio di trasporto -

Livello di trasporto e Protocolli TCP/UDP

(b) Servizio di trasporto (2/6)

In dettaglio …

1) l'***Entità di trasporto*** (o *Unità di trasporto*): è il software o l'hardware che fornisce i servizio di trasporto

2) SAP (*Service Access Point*): è il punto di accesso ad un servizio che un livello OSI offre al suo livello superiore (in un'architettura a strati come OSI ciascun livello offre una serie di servizi a quello gerarchicamente superiore ed usufruisce dei servizi offerti da quello sottostante).

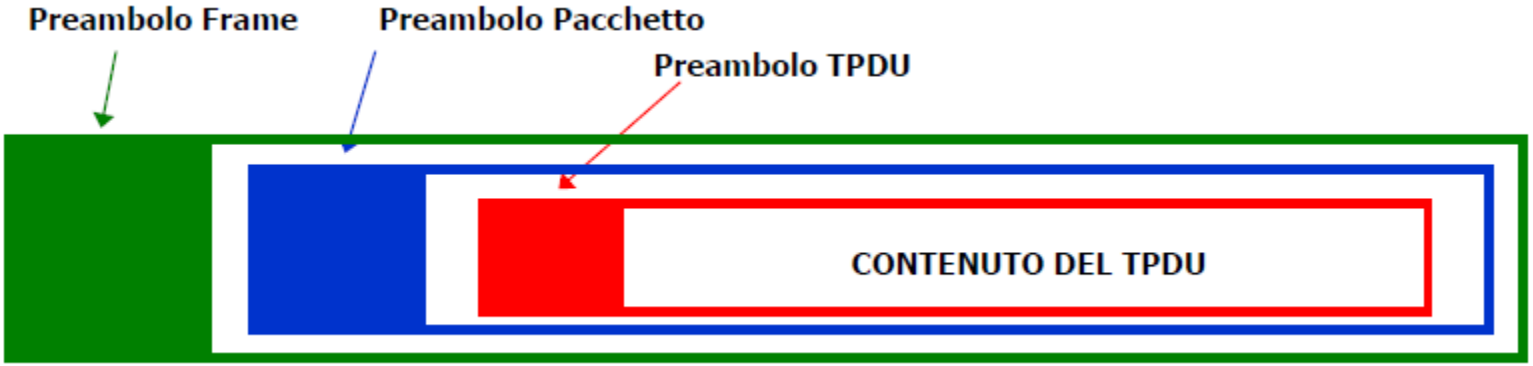
Generica interfaccia logica tra due entità una di livello N-1 e l'altra di livello N.

Nell'architettura TCP/IP qualsiasi protocollo di livello applicazione può accedere al servizio di trasporto offerto dal protocollo TCP attraverso un ***socket*** che rappresenta il punto di accesso al servizio di trasporto del protocollo TCP

Livello di trasporto e Protocolli TCP/UDP
(b) Servizio di trasporto (3/6)

3) ***TPDU*** (*Transfer Protocol Data Unit*): è l'unità dei dati scambiati dal protocollo di trasporto.

✓ La comunicazione tra applicazioni avviene con scambi di messaggi che vengono segmentati e trasformati in TPDU.



Struttura dati relativa al TPDU

Livello di trasporto e Protocolli TCP/UDP
(b) Servizio di trasporto (4/6)

… ancora su TPDU …

✓ i campi dati della TPDU sono “incapsulati” nelle strutture dati relative ai livelli sottostanti (*Frame* per il livello di collegamento e *Pacchetto* per quello di rete)

✓ l’*incapsulamento* è dovuto al fatto che ogni livello genera dati e li aggiunge (percorrendo la pila dall’alto verso il basso) o li sottrae (percorrendo la pila dal basso verso l’alto) in relazione allo stato della comunicazione in cui si trova l’host (trasmissione o ricezione)

Livello di trasporto e Protocolli TCP/UDP
(b) Servizio di trasporto (5/6)

Origine della TPDU (I)

- **Protocol Data Unit** (PDU) $\stackrel{\text{def}}{=}$ è l'unità d'informazione o pacchetto scambiata tra due *peer entities* in un protocollo di comunicazione di un'architettura di rete a strati
- La PDU è composta da:
  - *Protocol Control Information* (PCI), ovvero le informazioni di controllo come gli indirizzi, i numeri di sequenza e i flag. La PCI è generalmente posta in testa alla PDU (in tal caso è detta *header*) o in coda (*trailer* o *footer*)
  - *Service Data Unit* (SDU), ovvero i dati da trasmettere. La SDU costituisce il *payload* della PDU ed è generalmente ottenuta a partire dalle PDU degli strati più in alto nella pila protocollare

Livello di trasporto e Protocolli TCP/UDP
(b) Servizio di trasporto (6/6)

Origine della TPDU (II)

- Nel modello ISO/OSI sono definiti diversi tipi di PDU ,una per ogni strato del modello.

Elencandoli in relazione ai livelli della pila si ha:

- ✓ APDU, Application, a livello di applicazione (*messaggio*)
- ✓ PPDU, Presentation, a livello di presentazione
- ✓SPDU, Session, a livello di sessione
- ✓ **TPDU**, Transport, a livello di trasporto (*segmento*)
- ✓ NPDU, Network, a livello di rete (*pacchetto* o *datagramma*)
- ➢ LPDU, Link, a livello di collegamento (*trama* o *frame*)
- ✓ bit o simboli, a livello fisico

- Ogni PDU ha un formato caratteristico che implementa le specifiche del relativo protocollo

Livello di trasporto e Protocolli TCP/UDP
(c) Primitive del servizio di trasporto (1/3)

Premessa

- E' utile ribadire che il livello di trasporto è caratterizzato da:
  - Servizi che possono essere:
    - ✓ *affidabili* se eseguono le operazioni nel perfetto ordine
    - ✓ *non affidabili* se garantiscono solo l'*indirizzamento*
  - Protocolli che si distinguono in:
    - ✓ *UDP* (User Datagram Protocol): protocollo asincrono che non richiede la trasmissione di avvenuta ricezione
    - ✓ *TCP* (Transfer Control Protocol): protocollo sincrono dove è richiesto il messaggio di accettazione dei dati

Livello di trasporto e Protocolli TCP/UDP
(c) Primitive del servizio di trasporto (2/3)

- La premessa consente di affermare che i protocolli di trasporto, oltre che ad essere implementati nei più diffusi sistemi operativi, forniscono ai programmatori le *funzioni base* o più semplicemente ***primitive***

- Alcuni esempi di primitive elementari sono:
  - **LISTEN** : l'host si mette in attesa di richiesta di connessione
  - **SEND DATA** : utile a trasmettere un contenuto
  - **RECEIVE DATA** : utile per ricevere un contenuto
  - **T-CONNECT** : per aprire una connessione
  - **T-DISCONNECT** : per chiudere una connessione

Livello di trasporto e Protocolli TCP/UDP

(c) Primitive del servizio di trasporto (3/3)

- Per ogni primitiva ci sono i seguenti metodi:

Metodo primitiva	Descrizione
request()	Si chiede al servizio di compiere un'azione
indication()	Il servizio segnala un evento
response()	Si chiede al servizio di rispondere all'evento
confirm()	Il servizio segnala l'arrivo di una conferma

Livello di trasporto e Protocolli TCP/UDP

(d) Tipologie di connessione del servizio di trasporto (1/5)

- Le primitive servono per favorire e scandire le varie fasi (o sessioni) di una comunicazione (o scambio dati) tra due host connessi in rete

- Esistono due tipi di connessione in cui vengono utilizzate suddette primitive:
  - **a.** *connessione asincorna* (o *connection-less*)
  - **b.** *connessione sincrona* (o *connection-oriented*)

NB. Parlare di comunicazione anziché di connessione è equivalente

Connessione ≡ Comunicazione

Livello di trasporto e Protocolli TCP/UDP

(d) Tipologie di connessione del servizio di trasporto (2/5)

… in dettaglio …

a. Connessione asincrona (I)

- la **comunicazione senza connessione** (*connectionless*) in reti a commutazione di pacchetto è un metodo di trasmissione dati
- in questa modalità ogni pacchetto dati trasporta informazioni nell'intestazione (*header*)
- nella *header* è anche contenuto l'indirizzo di destinazione
- suddetto indirizzo è sufficiente per permettere una spedizione indipendente del pacchetto attraverso la rete
- un pacchetto trasmesso in una modalità senza connessione è tipicamente chiamato *datagramma*

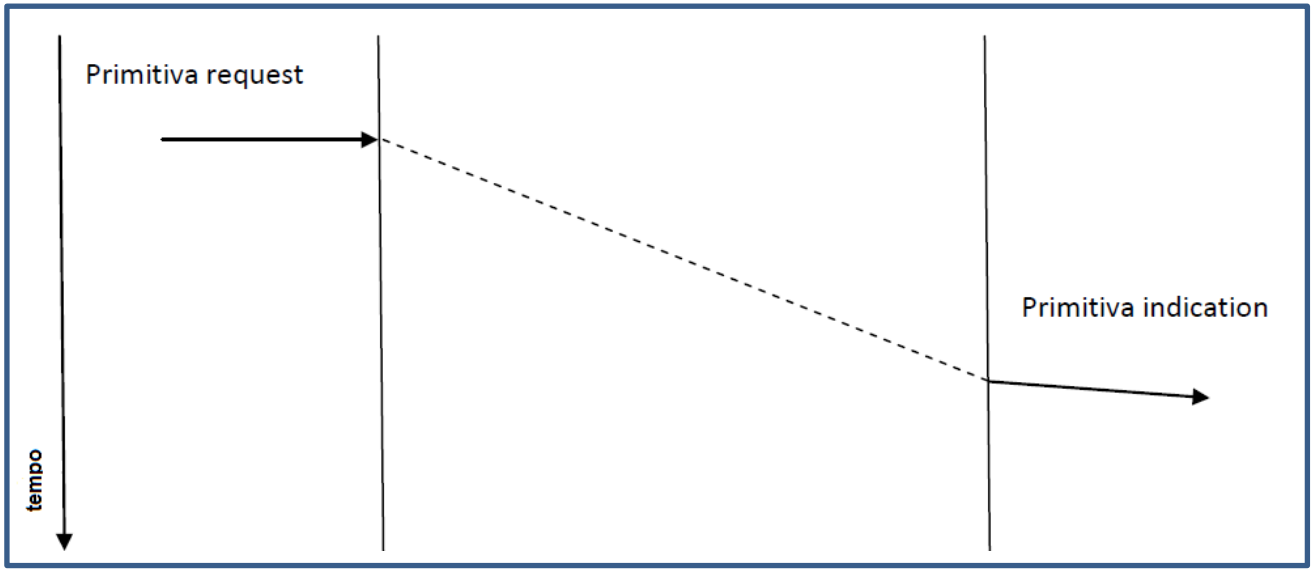
Livello di trasporto e Protocolli TCP/UDP

(d) Tipologie di connessione del servizio di trasporto (3/5)

… in dettaglio …

a. Connessione asincrona (II)

- Questa tipologia di comunicazione ha il vantaggio, rispetto a una *comunicazione orientata alla connessione*, di un basso sovraccarico di lavoro
- Consente inoltre operazioni di broadcast e multicast (in modo da salvare più di una risorsa di rete quando c'è bisogno di trasmettere gli stessi dati a diversi destinatari)



Esempio di Connessione Asincrona

Livello di trasporto e Protocolli TCP/UDP
(d) Tipologie di connessione del servizio di trasporto (4/5)

… in dettaglio …

b. Connessione sincrona (I)

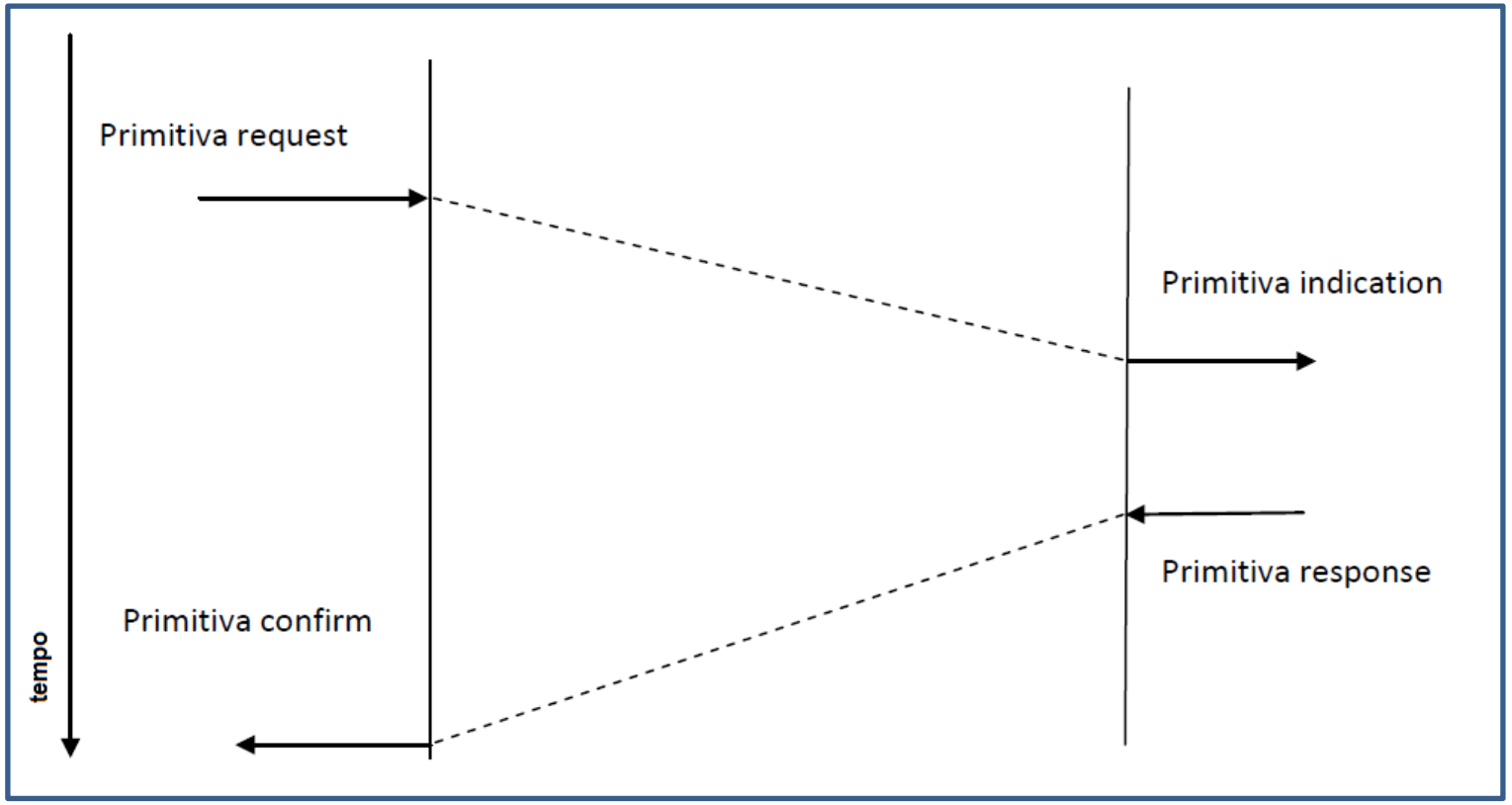
- La **comunicazione orientata alla connessione** (*Connection oriented*) è una modalità di comunicazione dati tramite la quale i dispositivi terminali usano un protocollo di comunicazione per stabilire una connessione logica o fisica *end-to-end* tra gli agenti della comunicazione prima della trasmissione di qualsiasi tipo di dato
- Si contrappone ad una comunicazione senza connessione

- I protocolli di trasporto orientati alla connessione (in senso logico) offrono spesso servizi affidabili, che richiedono una conferma o riscontro del successo nella trasmissione/ricezione dei pacchetti dati e funzionalità automatiche di richiesta di ritrasmissione in caso di pacchetti mancanti/persi o corrotti da errori

Livello di trasporto e Protocolli TCP/UDP
(d) Tipologie di connessione del servizio di trasporto (5/5)

… in dettaglio …

b. Connessione sincrona (II)



Esempio di Connessione Sincrona

autore
SANDRA SCAZZINO

materia
Diritto commerciale

Il mondo del P2P

Che cos'è il P2P

Il *peer to peer*, o P2P, è un'abbreviazione che sta per "condivisione di risorse tra pari". È un modo rivoluzionario di concepire la rete che fino a poco tempo fa utilizzava quasi esclusivamente un'architettura di tipo *client/server*, dove cioè si prevedeva la presenza di un elaboratore centrale (*server*) che metteva a disposizione degli elaboratori a esso collegati (***client***) programmi e informazioni.

Il P2P è una rete di computer che non possiede client o server fissi, ma un numero di nodi equivalenti, i "peer" appunto, che svolgono la funzione sia di client sia di server verso altri nodi della rete. Lo scopo essenziale di questa tipologia di rete è scambiare e condividere gratuitamente file (*file sharing*). Ogni nodo della rete può avere una differente configurazione locale, così come diverse possono essere le caratteristiche hardware, ma è comunque in grado di effettuare una transazione tra nodi che appartengono al suo stesso livello gerarchico.

Grazie al P2P è possibile scaricare dalla rete file musicali, film e i **software** più diffusi. Migliore è la connessione alla rete, più veloce è lo scambio tra utenti: infatti scaricare attraverso una linea ADSL (Asymmetric Digital Subscriber Line) è molto più rapido che con la normale connessione telefonica di tipo analogico.

Le opinioni su questo modello sono diametralmente opposte: i produttori e gli autori di musica e film sostengono che uno scambio selvaggio di musica in rete rischia di penalizzare il mercato; i sostenitori del P2P ribattono affermando che la diffusione di musica grazie alla rete amplia il mercato, tesi peraltro sostenuta anche da studi recenti.

Come si entra a far parte della rete P2P

Per entrare nel mondo del P2P bastano poche semplici operazioni: occorre innanzitutto collegarsi alla rete e scegliere uno dei programmi gratuiti che permettono di porre in condivisione il proprio materiale e di scaricare quello altrui. La scelta va fatta in base ai propri interessi (musica, cinema, software) e alla propria dimestichezza informatica. Effettuata la scelta, è sufficiente scaricare il programma sul proprio computer e avviarlo: penserà da solo all'installazione e alla configurazione!

A questo punto cercare i file di proprio interesse diventa semplicissimo. Basta inserire nel motore di ricerca del programma il nome del cantante o il

Glossario minimo

Client: con questo termine si indica una componente che accede ai servizi o alle risorse di un'altra componente, il server.

Software: è un programma o un insieme di programmi in grado di funzionare su un elaboratore.

Spyware: è un'ampia gamma di software maligni.

Ip: detto anche indirizzo ip, è il numero che identifica univocamente nell'ambito di una singola rete i dispositivi collegati con una rete informatica che utilizza lo standard IP (Internet Protocol).

obiettivi

- conoscere il significato del *peer to peer*
- individuare i soggetti coinvolti e l'oggetto di tale tecnologia
- collocare il fenomeno del *peer to peer* nella realtà giuridico-economica attuale
- individuare le relazioni con la normativa inerente il diritto d'autore

proposte didattiche

- quali ragioni spiegano il crescente successo del *peer to peer* tra i fruitori della rete?
- come valuti la normativa italiana relativa al diritto d'autore, con particolare riferimento al fenomeno del *peer to peer*?

titolo della canzone (o del film) per avere in tempi rapidissimi una lunga lista di risultati.

L'ultimo passo da compiere è quello di salvare il file scaricato sul proprio hard disk o su una memoria di massa esterna.

P2P ibrido o puro?

L'esempio più classico di P2P è la rete per la condivisione di file (*file sharing*). Queste reti (Gnutella, Fast Tracce e l'ormai oscurato Napster) forniscono lo scambio libero (e qualche volta **anonimo**) di file tra i computer connessi a Internet.

Napster fu il primo sistema di P2P di massa, disponibile già a partire dal 1999. In verità non era un P2P puro, in quanto utilizzava un sistema di server centrali che mantenevano la lista dei nodi connessi e dei file condivisi, mentre le transazioni vere e proprie avvenivano direttamente tra i vari utenti. Nel luglio 2001 un giudice ordinò ai server Napster di chiudere l'attività a causa della ripetuta violazione di *copyright*.

I sostenitori di Napster pensavano che il *file sharing* fosse la caratteristica fondamentale di Internet e consideravano Napster essenzialmente un motore di ricerca. Molti erano convinti che la soluzione di chiudere Napster avrebbe spinto gli utenti a utilizzare altri mezzi per lo scambio di file su Internet.

Il file sharing anonimo

Rappresenta l'ultima frontiera del P2P: "**Mute**" è un programma P2P per il file sharing che garantisce un elevatissimo grado di sicurezza proteggendo efficacemente la privacy dei suoi utenti. Il programma creato da Jason Roher impedisce che i due estremi dello scambio di file (chi dà e chi riceve) si parlino e comunichino direttamente. Non solo: cripta con un algoritmo militare i file scambiati.
Le strategie adottate da Roher a difesa totale della privacy sono quindi due: l'assegnazione di un **ip** virtuale a ogni utente connesso e la criptazione dei dati scambiati. Secondo l'autore è impossibile intercettare chi sta scambiando materiale e risalire dall'ip virtuale a quello reale dell'utente. Alcuni però sostengono che proprio questo algoritmo di funzionamento, quando sarà usato da milioni di utenti, ne causerà il collasso.

Dopo la chiusura di Napster si diffusero diversi programmi **P2P ibridi** di condivisione file, tra i quali possiamo citare Morpheus e Kazaa, anch'essi però colpiti dalla reazione dell'industria discografica. Più fortuna hanno avuto i sistemi **P2P puri**, basati sul protocollo Gnutella (eMule), che hanno dimostrato l'inattaccabilità da parte delle *major* a causa della loro natura di server decentralizzati. Ogni nodo della rete è client e server nello stesso tempo, rendendo non necessaria la presenza di un server centrale detentore di informazioni; non è quindi possibile un potenziale controllo sulla globalità degli utenti e dei file condivisi.

File sharing: solo vantaggi?

Per la filosofia stessa del P2P il computer diventa un nodo di una vasta rete. È quindi necessariamente soggetto a tutti i rischi derivanti dall'attività di scaricamento di file e altresì dall'attività di accesso da parte di altri utenti alla porzione di disco in cui è messo a disposizione il materiale da prelevare. È importante quindi comprendere e prevedere i rischi inerenti alla condivisione di file P2P prima di procedere con il download dei file.

Un primo rischio consiste nell'**installare sul computer un software indesiderato**. Infatti, come per la maggior parte del materiale scaricato da Internet, anche i file condivisi possono presentare rischi per la sicurezza, in quanto si possono scaricare inavvertitamente file contenenti virus, ***spyware*** e altri software indesiderati. Un file apparentemente regolare potrebbe essere in realtà un virus. Accade spesso che utenti inconsapevoli che condividono abitualmente file, scarichino un software utile ma contenente spyware nascosti. È possibile ridurre i pericoli associati a questa procedura installando software antivirus e antispyware e facendo estrema attenzione affinché questi strumenti siano sempre attivi e aggiornati.

Il secondo rischio consiste nel **violare le leggi sul copyright**. Infatti, anche se è sorta una controversia sull'uso della condivisione P2P per trasmettere o "piratare" illegalmente materiale protetto da copyright, in particolare file audio e video, l'uso del software P2P è tuttora legale. Se si utilizza il software P2P, è però importante distinguere tra materiale di dominio pubblico e materiale protetto da copyright, in modo che la condivisione avvenga in maniera responsabile.

La legge sul copyright vale per il **software proprietario**, quello che viene concesso con licenza spesso non esclusiva, a tempo e per unica copia, ma non vale per il *freeware* (software che può essere

copiato e utilizzato gratuitamente) e per lo *shareware* (che è in prova per un determinato e limitato periodo di tempo e che può essere copiato e utilizzato entro i limiti indicati nella licenza). Ancor meno vale per il software considerato di *public domain*, per il quale l'autore si sia completamente spogliato di ogni diritto riconosciutogli dalle norme in tema di proprietà intellettuale. In quest'ultimo caso chiunque può copiare e utilizzare il programma, assemblandolo ad altri o modificandolo (i programmi di dominio pubblico sono spesso accompagnati dall'indicazione "no copyright"). Men che meno si pongono limiti per la distribuzione del software libero.

I tipi di file maggiormente condivisi in rete sono gli **mp3** (file di musica) e i **DivX** (file contenenti i film). Questo ha portato le compagnie discografiche e i media ad affermare che queste reti potrebbero costituire una minaccia per i loro interessi. Di conseguenza, il P2P è diventato il bersaglio legale delle organizzazioni che riuniscono queste aziende, che hanno determinato la chiusura del servizio di Napster. La manifestazione più estrema di questi sforzi risale al gennaio 2003, quando viene introdotto negli Stati Uniti un disegno di legge nel quale si garantiscono, al detentore del *copyright*, i diritti legali per fermare i computer che distribuiscono materiale tutelato dai diritti d'autore. Risale invece al 2004 la Legge Urbani (**legge n. 128/2004**) nella quale si è regolamentata la distribuzione di opere coperte dal diritto d'autore, anche attraverso il cosiddetto P2P e viene sancita la possibilità di incorrere in sanzioni penali anche per chi fa un uso esclusivamente personale dei file protetti.

Fino all'approvazione di tale legge, non erano previste sanzioni per la condivisione di opere tutelate dal diritto d'autore qualora non vi fosse scopo di lucro. Ma la sostituzione della locuzione "*a fini di lucro*" con "*per trarne profitto*", operato da questa legge, inserisce la possibilità di incorrere in gravi sanzioni penali anche per chi fa esclusivamente un uso personale di opere protette dal diritto d'autore ottenute attraverso questa pratica. Pertanto lo scambio di opere protette attraverso sistemi di *file sharing* sarebbe ricaduto nelle sanzioni penali, poiché i sistemi di condivisione di file più diffusi utilizzano reti P2P, nelle quali come abbiamo visto ciascun nodo (utente) è sia client (*downloader*, e quindi scarica) sia server (*uploader*, e quindi condivide), ossia i file scaricati sono automaticamente condivisi, anche durante la fase di scaricamento.

Nel 2007, la Corte di Cassazione ha accolto il ricorso presentato da due studenti torinesi, condannati in appello a una pena detentiva, sostituita da un'ammenda, per avere «duplicato abusivamente e distribuito» programmi illecitamente duplicati. I reati contestati erano quelli previsti dalla legge sul diritto d'autore, che prevede «la punibilità da sei mesi a tre anni, di chiunque abusivamente duplica, per trarne profitto, programmi per elaboratore o ai medesimi fini importa, distribuisce, vende, detiene a scopo commerciale o imprenditoriale o concede in locazione programmi contenuti in supporti non contrassegnati dalla Siae» e punisce con la reclusione da uno a quattro anni chi «riproduce, duplica, trasmette o diffonde abusivamente, vende o pone altrimenti in commercio, cede a qualsiasi titolo o importa abusivamente oltre cinquanta copie o esemplari di opere tutelate dal diritto d'autore e da diritti connessi».

Per la Cassazione era da escludere per i due studenti la configurabilità del reato di duplicazione abusiva, attribuibile non a chi in origine aveva effettuato il download, ma a chi semmai aveva salvato il programma dal server sul proprio computer per poi farne delle copie. Ma soprattutto doveva essere escluso che la condotta degli autori della violazione fosse stata determinata da fini di lucro, emergendo dall'accertamento di merito che gli imputati non avevano tratto alcun vantaggio economico dalla predisposizione del server Ftp. Per "fine

di lucro" infatti, deve intendersi un fine di guadagno economicamente apprezzabile o di incremento patrimoniale da parte dell'autore del fatto.

Negli ultimi anni le reti P2P hanno avuto un'enorme espansione, diventando tecnologicamente più difficili da smantellare. Questo, unitamente allo scarso interesse delle major per i tafferugli legali, che non sembrano effettivamente giovare alla causa del copyright, ha determinato uno spostamento dell'attenzione da parte degli operatori del settore nei confronti dell'*utente*, contro cui è iniziata una vera e propria serie di operazioni ostili.

Tra queste ricordiamo innanzitutto il *decoying*, un sistema basato sulla distribuzione di file fasulli che contengono brevi trailer promozionali o che addirittura sono vuoti, nonostante l'aspetto apparentemente completo per il ricco corredo informativo (come i nomi di regista, protagonisti, anno di produzione, aggiunti al titolo), che fa balzare il corrispondente file in vetta ai risultati di ricerca, normalmente ordinati in base a un criterio qualitativo.

Lo *spoofing* determina invece il dirottamento di chi vuole scaricare materiale verso file inesistenti o, peggio, contenenti virus informatici.

Se poi il *searcher* si dimostra particolarmente tenace, si può ricorrere ad altri sistemi, per esempio all'*interdiction*, che blocca ogni contatto tra il collezionista di materiale illecito e i server P2P, oppure allo *swarming*, tecnica che sfrutta il protocollo Bit Torrent (un protocollo che prevede il frazionamento in parti uguali del contenuto dei file che devono essere trasmessi). Durante l'assemblaggio delle varie parti che componevano il file originale, lo *swarming* prevede l'inserimento di una piccola tranche di bit atta a determinare malfunzionamenti e rallentamenti durante lo scaricamento del file stesso.

Prospettive future del P2P

È probabile che le reti P2P verranno sempre più spesso utilizzate per la trasmissione di grandi flussi di dati, come programmi televisivi o film, sfruttando la banda di trasmissione di cui dispongono i singoli utenti che a loro volta trasmetteranno agli altri fruitori il flusso dati ricevuto.

Un simile metodo di diffusione permette la trasmissione in tempo reale di contenuti video, ma richiede che i singoli utenti siano dotati di connessioni a elevata banda sia in ricezione sia in trasmissione. Questo limita la diffusione di tale tecnologia in Italia, dove sono molto diffuse linee asimmetriche ADSL, che forniscono una banda elevata in ricezione, ma scarsa in trasmissione.

Utilizzando tale tecnologia grandi società stanno sperimentando la possibilità di fornire contenuti a pagamento tramite l'opportunità del P2P. Questo scelta è motivata dal fatto che la tecnologia P2P non richiede server di grandi dimensioni per gestire molti utenti, dato che se la rete è ben bilanciata si autosostiene, dunque è indipendente dal numero di utenti. Vanno ancora risolti però problemi di copyright e di sicurezza.

Progettazione di protocolli di comunicazione

(Linee guida generali)

Appunti integrativi per il corso di

Tecnologie e Progettazione di Sistemi Informatici e di Telecomunicazioni

Classe 5A Informatica

Anno scolastico 2020/2021

Istituto Isis “E.Fermi”, Bibbiena (Ar)

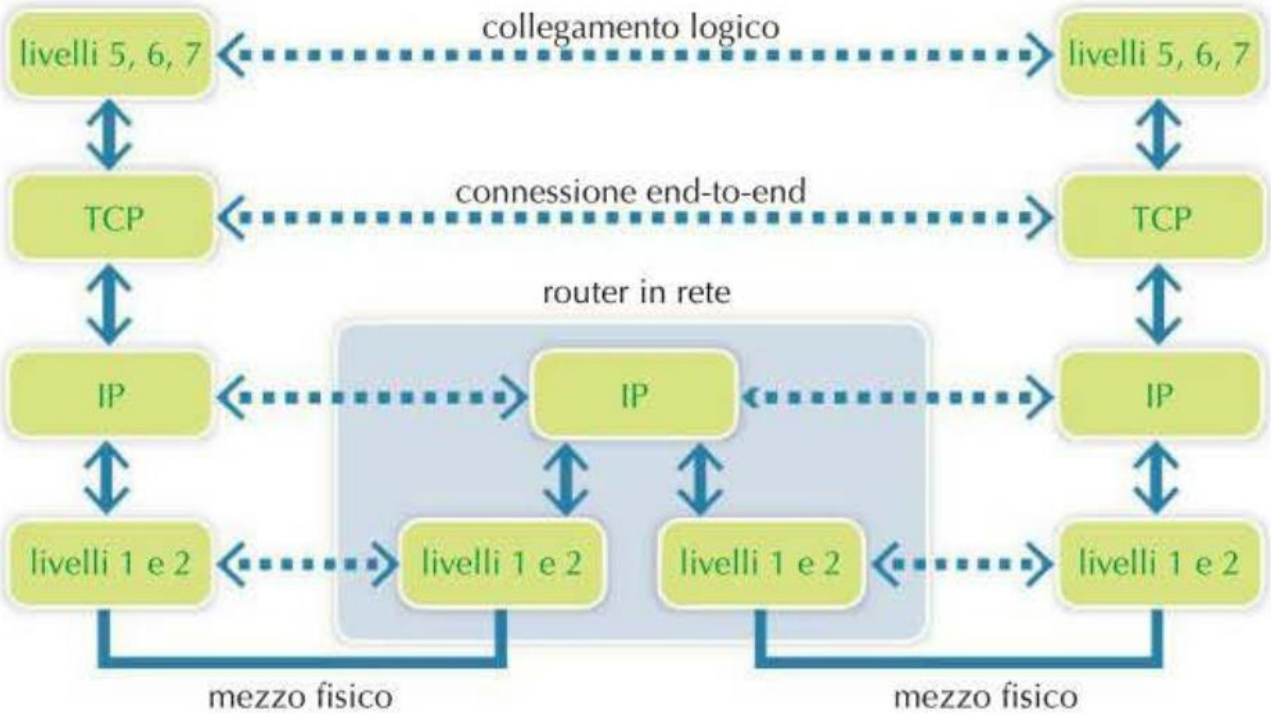
Prof. Federico Santolini

Indice degli argomenti

Protocolli di rete

Un **protocollo è l'insieme di regole utilizzate da due entità di pari livello per scambiarsi informazioni**, **specificando cosa** deve essere comunicato, **in che modo e quando**. In pratica, due entità remote potranno colloquiare fra di loro tramite i diversi protocolli usati nei vari livelli della pila protocollare.

Degli esempi di protocolli sono quelli utilizzati nell'architettura di rete a strati TCP/IP, che a sua volta fornisce i servizi associati ad ogni protocollo del livello applicativo, come ad es. SMTP, FTP, HTTP, ecc.

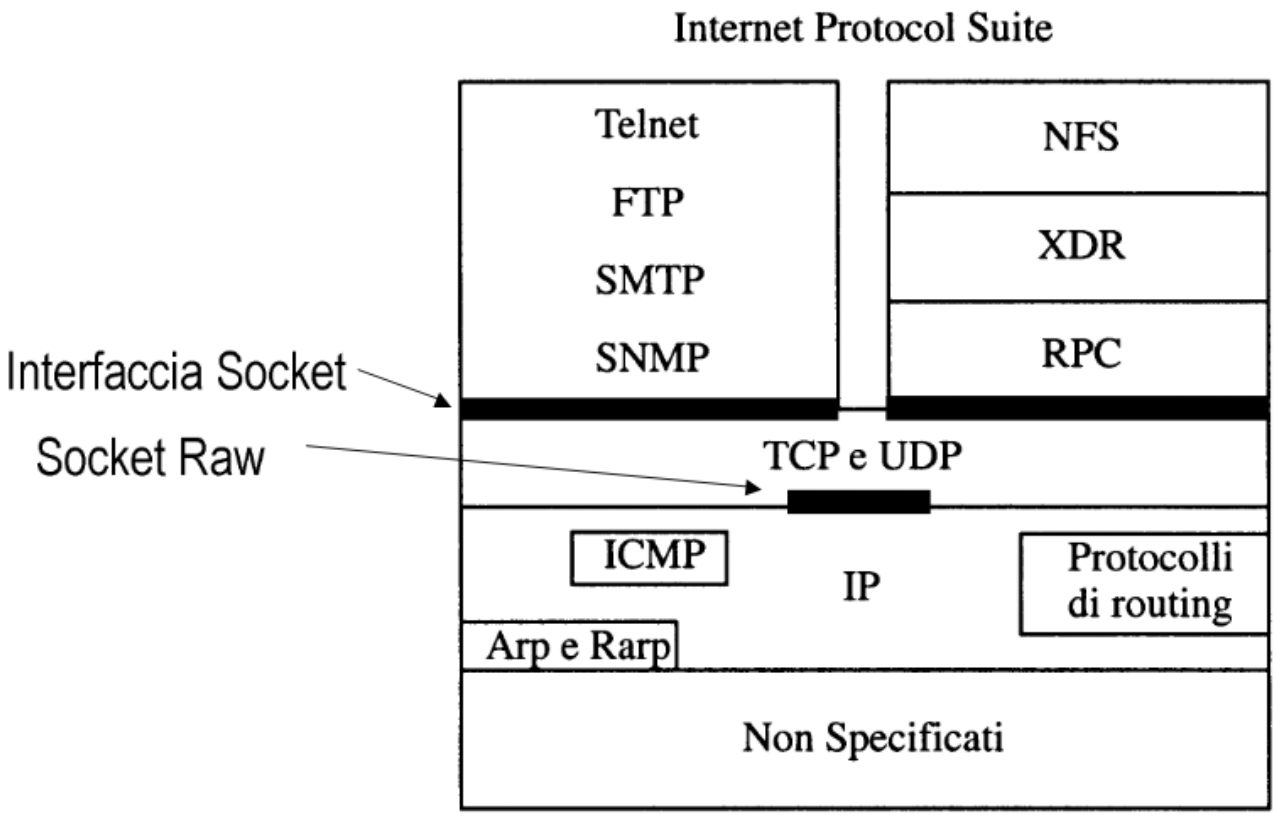


I protocolli del livello applicativo supportano diversi programmi applicativi offrendo determinati servizi, che variano a seconda dell'architettura scelta. Ad esempio, se il servizio è basato su un'architettura di tipo client-server, il protocollo dovrà sempre prevedere la gestione di queste due diverse entità e delle relative comunicazioni. Ciò vuol dire che i processi dovranno essere almeno due, il client e il server.

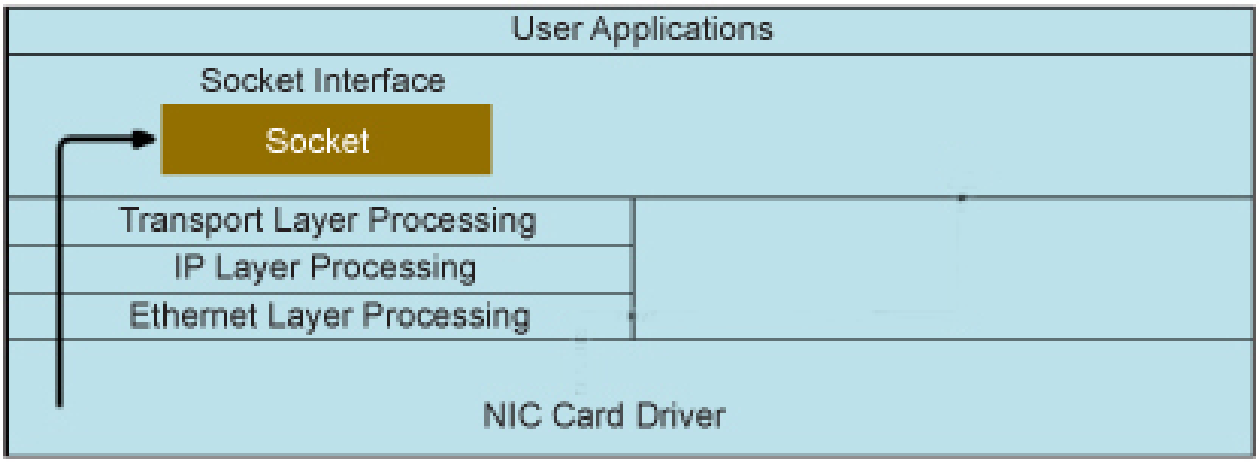
Interfacce e servizi

Affinché due protocolli remoti di pari livello possano comunicare devono necessariamente usare i servizi offerti dal livello inferiore. Nella pila protocollare **TCP/IP** la comunicazione tra il livello applicativo e il livello di trasporto avviene attraverso una interfaccia che usa i **socket** e le **API** offerte dal sistema operativo per la programmazione. Il socket permette di identificare univocamente il processo all'interno di un'entità remota.

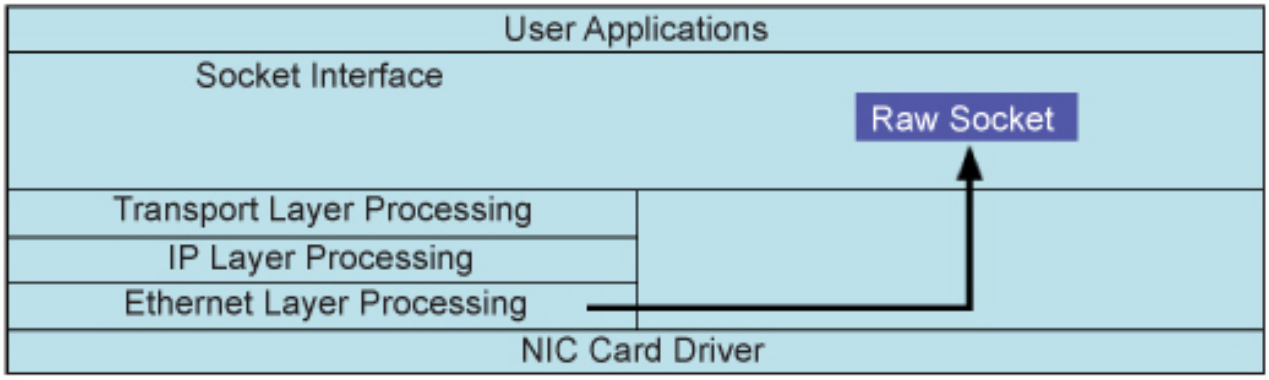
Torna all'indice



I socket usati normalmente, come lo **stream socket** (TCP) o il **datagram socket** (UDP), ricevono i dati dal livello di trasporto privati degli header dei livelli sottostanti, quindi ricevono solo il **payload** costituito dai dati finali. Questo vuol dire che non sarà fornito direttamente alcuna informazione relativamente all'indirizzo MAC e IP della sorgente; se la comunicazione avviene tra macchine diverse, viene effettuato solo uno scambio di dati.

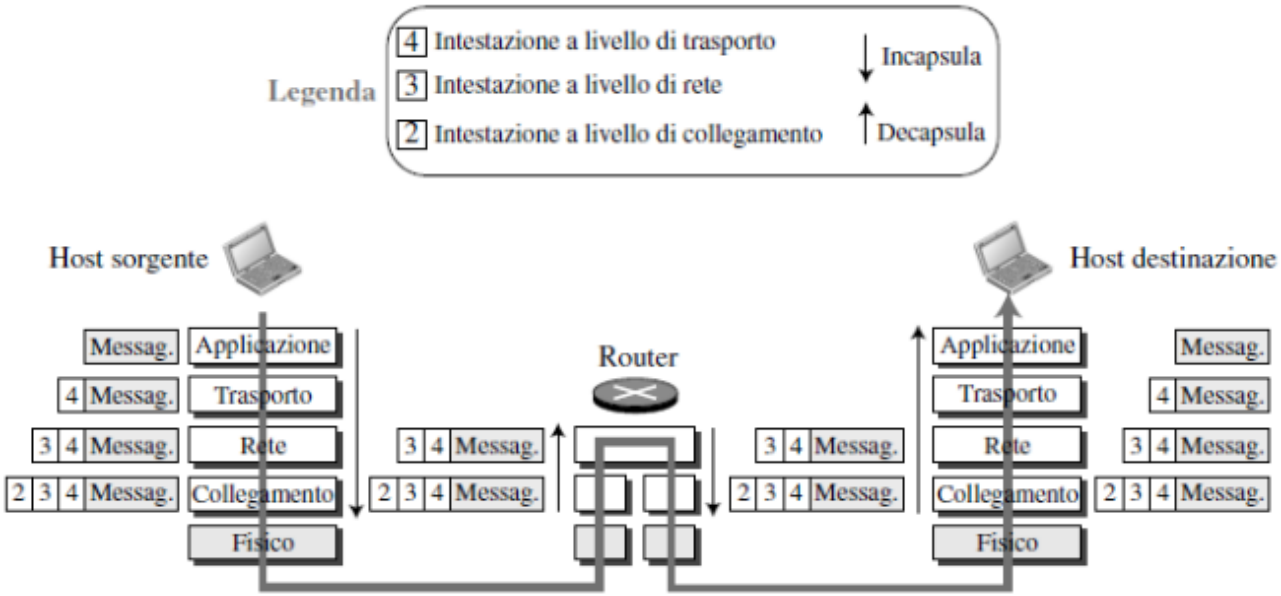


Lo scopo dei **raw socket** è completamente diverso. Un **raw socket** permette all'applicazione ricevente di accedere direttamente ai protocolli di basso livello (ad esempio direttamente al livello di rete), permettendo quindi di ricevere un pacchetto non decapsulato, con tutti gli header dei diversi livelli protocollari.



Implementazione del servizio

I dati inviati dal livello applicativo saranno **incapsulati** dai diversi livelli protocollari per **aggiungere** negli **header** tutti i dati necessari per permettere la comunicazione tra i protocolli di pari livello. Una volta giunto a destinazione avverrà la fase di **decapsulamento** per passare il **payload** al livello superiore.



A seconda dei protocolli e della capacità della rete da utilizzare, i pacchetti possono essere **frammentati** in modo da generare dei pacchetti di dimensioni adeguatamente piccole per essere inviati nella rete, per poi essere **riassemblati** dal protocollo di livello paritario a destinazione.

Il pacchetto derivante dal processo di **incapsulamento**, conterrà:

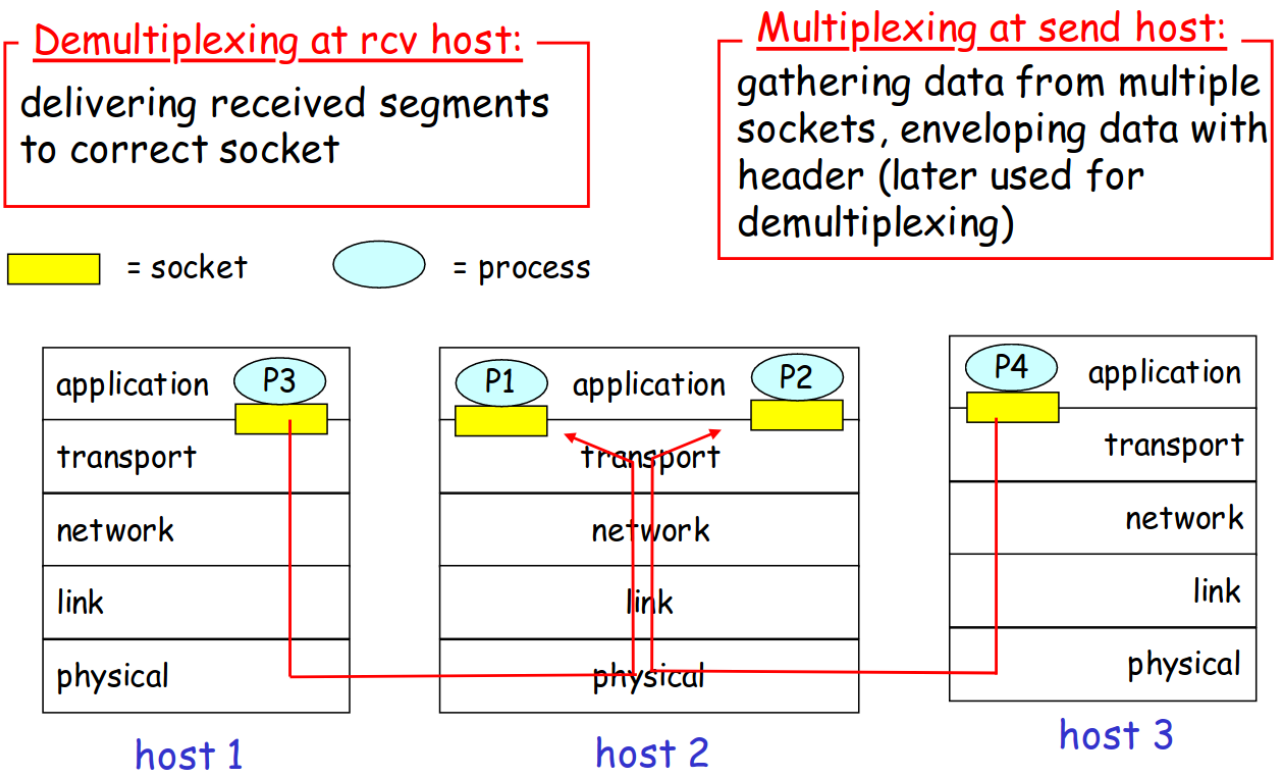
- una intestazione o **header** contenente dati di controllo aggiunti dal protocollo;
- il corpo o **paylod** costituito dagli eventuali frammenti;
- una eventuale coda o **tail** (anche **trailer** o **footer**) contenente anch'essa dati di controllo aggiunti dal protocollo.

La **header** e la **tail** sono usate dal protocollo per controllare la comunicazione, ad esempio, la header contiene informazioni utilizzate dall'entità del protocollo come:

- indirizzo del sistema mittente e destinatario;

- numero di sequenza per poter ordinare i pacchetti;
- codice per la individuazione di errori di trasmissione;
- codice identificativo di servizi particolari (es. priorità).

L'**indirizzamento** è una delle informazioni più importanti **presente** praticamente **in qualsiasi layer** della pila protocollare TCP/IP, perché **permette** di effettuare le operazioni di **multiplexing** e **demultiplexing**. I campi ***Ethertype***, l'***IP address*** e il ***Port Number*** sono i diversi indirizzi usati rispettivamente dal protocollo *Ethernet*, *IP* e *TCP/UDP*. L'**indirizzamento** può però anche essere **implementato a livello applicativo**, ad esempio specificando nel pacchetto del livello applicativo il nome di un file da manipolare, che identifica la risorsa finale della comunicazione.



I servizi offerti dai vari livelli della pila protocollare possono inoltre essere di due tipi:

- **Orientati alla connessione**: il protocollo prevede delle istruzioni che permettano di creare una connessione fra gli estremi della comunicazione; i pacchetti sono consegnati in ordine agli strati protocollari di più alto livello, e se persi possono essere ritrasmessi (**affidabilità**). Di solito è previsto anche un riscontro della ricezione in modo da prevedere un metodo di ritrasmissione dei dati nel caso venissero persi (**conferma**, **acknowledgment**).

- **Senza connessione**: i dati non vengono consegnati in ordine agli strati protocollari di più alto livello, e **può o meno esserci il riscontro della ricezione**. Di conseguenza, nel caso non ci sia il riscontro della ricezione, i dati persi non saranno gestiti in alcun modo.

Il concetto di **conferma della ricezione** è comunque una caratteristica indipendente dalla presenza o meno di una connessione; spesso vengono abbinate, ma la conferma della ricezione può rimane una caratteristica a sé stante che può essere presente anche

in un protocollo non connesso, come nel caso di una delle possibili implementazioni del protocollo LLC Layer (*Logical Link Control Layer* - *Standard IEEE 802.2*).

Un altro servizio offerto da diversi protocolli della pila protocollare TCP/IP è il **controllo del flusso**, con il quale è possibile stabilire la quantità di pacchetti che le entità possono scambiarsi senza sovraccaricare le risorse che localmente hanno a disposizione. Una delle **tecniche di controllo del flusso** più semplici è l'implementazione della tecnica stop-and-wait, mentre i protocolli sliding window usati dal protocollo Data Link Layer e dal protocollo TCP, seppur complessi, risultano molto più efficienti soprattutto in presenza di molto rumore che può disturbare il segnale.

Spesso i protocolli sono anche progettati per effettuare un **controllo della presenza di errori**, come nel caso dell'uso del FCS (*Frame Check Sequence*) del frame Ethernet, o verificare che non siano stati persi pacchetti come nel caso del protocollo di conferma, numerazione dei segmenti e timer del protocollo TCP, o che non siano errati i comandi protocollari, come nel caso del Checksum usato dallo header TCP per rilevare l'eventuale presenza di errori nello header del segmento. Non necessariamente queste caratteristiche sono implementate dai protocolli della pila protocollare, in quanto **il controllo di errori introduce un overhead** che potrebbe essere ritenuto non accettabile; la loro implementazione dipende dal servizio che il protocollo deve offrire.

Decisamente più complessa, ma in costante aumento è infine l'implementazione dei **servizi di trasmissione**, come la **prioritizzazione dei pacchetti**, la **garanzia di un certo livello di qualità del servizio** e la **gestione della sicurezza**. Questi ultimi aspetti stanno di fatto assumendo un'importanza sempre maggiore, e poco per volta le nuove revisioni dei protocolli che compongono la suite TCP/IP, così come i moderni protocolli del livello applicativo, si stanno sempre più orientando ad una loro gestione.

Analisi di un protocollo applicativo: TFTP

TFTP (*Trivial File Transfer Protocol*) è un semplice protocollo di trasferimento dei file descritto nello RFC 1350., ed è stato implementato al di sopra di UDP. Il protocollo è stato progettato per essere piccolo e semplice da implementare, di fatti non presenta tutte le caratteristiche di FTP, può solo leggere e scrivere file da e verso un server remoto; non ha alcuna forma di autenticazione e non permette di ottenere la lista di file e directory, come invece fa FTP. I dati scambiati sono quantificati in byte e prevede tre diversi formati di trasferimento dei file, di fatto implementando un livello di presentazione.

Il protocollo applicativo implementa una serie di caratteristiche che sono deducibili anche solo osservando il formato dei sui messaggi.

TFTP è un **protocollo connesso**, infatti la connessione viene richiesta dal client inviando il seguente messaggio nel quale i primi due byte costituiscono lo header e indicano se il cliente vuole effettuare un'operazione di lettura (01) o di scrittura (02). Il campo successivo, di lunghezza variabile (in byte), indica il nome del file su cui vuole agire il client; per indicare la fine del nome viene usato un byte posto a 0 (stringa zero terminata). Il nome del file è di fatto una forma di **indirizzamento** a livello applicativo in quanto individua la risorsa su cui si vuole agire. Dopo il nome del file viene indicata

la modalità di trasferimento e, anche in questo caso la lunghezza del campo è variabile e quindi la terminazione della stringa viene effettuata tramite un byte posto a zero.

```
Type   Op #     Format without header
       2 bytes   string   1 byte     string   1 byte
       ------------------------------------------------
RRQ/  | 01/02 | Filename |  0  |   Mode   |  0  |
WRQ    ------------------------------------------------
```

Il server se accetta la richiesta di connessione si comporterà nel modo seguente:

- se il client ha richiesto una connessione per leggere un file il server invierà il primo blocco di dati con un messaggio che ha il seguente formato:

```
        2 bytes    2 bytes      n bytes
        ---------------------------------
       DATA | 03   | Block # |   Data   |
        ---------------------------------
```

  L'operazione di invio dei dati è identificata nello header dal codice 03, salvato in due byte, i blocchi dei dati sono numerati nello header con numeri progressivi partendo dal valore 01 ed utilizzando due byte, mentre i dati seguono occupando esattamente 512 byte;

- se invece il client ha richiesto una connessione per scrivere un file il server invierà un messaggio di conferma cha ha il seguente formato:

```
        2 bytes    2 bytes
        -------------------
       ACK  | 04  | Block # |
        -------------------
```

  Ogni **invio dei dati è confermato** e la conferma è individuata dal codice 04 nello header del messaggio (due byte), seguito dal numero di blocco che viene confermato; nel caso di conferma per l'apertura di connessione in scrittura il numero del blocco assume il valore 00.

La **chiusura della connessione** avviene nel momento in cui viene spedito un messaggio dati contenente un payload di lunghezza inferiore a 512 byte.

Qualunque sia l'operazione richiesta dal client, il trasferimento dei dati, da e verso il server, avviene sempre usando il messaggio dati e la conferma indicati in precedenza. È importante però sottolineare che il protocollo prevede la trasmissione di messaggi con un payload non superiori a 512 byte, che implica di fatto la **frammentazione e il riassemblaggio** del file.

Inoltre prima di poter effettuare la trasmissione di un nuovo messaggio deve essere stata ricevuta la conferma di ricezione del messaggio precedente, quindi il protocollo **esegue un controllo di flusso** di tipo stop-and-wait.

Il protocollo effettua anche un **controllo errori** inviando un messaggio con il seguente formato.

```
        2 bytes  2 bytes       string    1 byte
        ----------------------------------------
ERROR | 05    | ErrorCode |   ErrMsg   |   0  |
        ----------------------------------------
```

Un messaggio di errore non è mai confermato, quindi di fatto è una pura cortesia in quanto potrebbe andare perso durante la trasmissione. Nello header viene posto il codice 05 nei primi due byte, seguono due byte che riportano uno dei codici di errore elencati di seguito, e infine una stringa di messaggio di errore di lunghezza variabile e terminata da un byte posto a 0; il messaggio di errore è destinato ad un utente umano, quindi dovrà essere comprensibile.

Error Codes

Value	Meaning
0	Not defined, see error message (if any).
1	File not found.
2	Access violation.
3	Disk full or allocation exceeded.
4	Illegal TFTP operation.
5	Unknown transfer ID.
6	File already exists.
7	No such user.

Da quanto visto finora **il protocollo effettua l'incapsulamento** dei messaggi in quanto in ogni tipologia di messaggio **è presente uno header** che indica tramite un codice lo scopo del messaggio stesso. Questo codice è ovviamente comprensibile solo alle due entità che implementano il protocollo TFTP, e quindi solo alle due entità a livello applicativo.

Il protocollo non prevede **servizi di trasmissione**, come la **prioritizzazione dei pacchetti**, la **garanzia di un certo livello di qualità del servizio** e la **gestione della sicurezza**.

Progettazione di un protocollo

Per **progettare un nuovo protocollo applicativo** si deve prima di tutto stabilire il **tipo di servizio che è necessario usare al livello di trasporto**, decidendo se il nuovo protocollo di comunicazione a livello applicativo dovrà appoggiarsi ad un servizio connection-oriented o connectionless; nel primo caso si userà TCP, nel secondo UDP.

Questa scelta dipende da molti fattori in quanto optare per un servizio orientato alla connessione permette di demandare al livello di trasporto tutto ciò che è relativo all'instaurazione di una connessione affidabile e, se non viene richiesta una ulteriore gestione di connessione, la progettazione del protocollo risulterà più semplice.

Chiaramente un servizio connection-oriented necessiterà di un maggiore overhead a livello di trasporto, proprio per l'operazione di instaurazione e gestione della

connessione stessa, ma se il protocollo che si sta progettando non necessita di un servizio complesso come quello offerto da TCP, si potrà optare per un servizio non orientato alla connessione e non affidabile, come quello offerto da UDP. In questo caso sarà compito del nuovo protocollo a livello applicativo la gestione di una eventuale connessione, così come di un sistema di conferma di ricezione se fosse necessario anche evitare la perdita di pacchetti, implicando una progettazione molto più complessa.

Si osservi che **la scelta del servizio a livello di trasporto non determina le caratteristiche del nuovo protocollo di comunicazione a livello applicativo**. Ad esempio, se si sceglie a livello di trasporto il protocollo TCP, **non consegue** che il nuovo protocollo sia necessariamente connection-oriented; il nuovo protocollo potrà essere orientato alla connessione o meno a seconda degli aspetti che saranno implementati. Usare un certo servizio a livello di trasporto determina solo i servizi che il nuovo protocollo userà.

Quindi, a seconda dell'obiettivo del nuovo protocollo di comunicazione, la progettazione del protocollo dovrà prevedere la definizione di tutti o parte dei seguenti aspetti, analizzati in dettaglio nel seguito della trattazione:

1. indirizzamento;
2. frammentazione e riassemblaggio;
3. incapsulamento;
4. controllo della connessione;
5. servizio confermato o non confermato;
6. controllo degli errori;
7. controllo del flusso;
8. multiplexing e demultiplexing;
9. servizi di trasmissione.

Indirizzamento

L'**indirizzamento** permette di **identificare univocamente una entità nella rete**. Esistono diversi livelli di indirizzamento e ogni livello ha una propria visibilità:

- l'indirizzo fisico, come ad esempio l'indirizzo MAC, ha una visibilità locale ad una rete;
- l'indirizzo di rete, come ad esempio l'indirizzo IP, ha una visibilità globale su più reti;
- l'indirizzo del processo, come ad esempio il numero di porta in TCP/IP o il Service Access Point in OSI, ha una visibilità locale al sistema.

Nella progettazione di un protocollo di comunicazione che si appoggia allo stack TCP/IP, si dovrà **decidere** almeno il **numero di porta che identifica il processo** che implementa il protocollo stesso. Inoltre, se il protocollo è di tipo client-server, il numero di porta del server dovrà essere stabilito a livello di progettazione del protocollo, mentre la scelta per il client sarà lasciata come al solito al Sistema Operativo. Ma questo aspetto riguarda l'indirizzamento a livello di trasporto; a livello applicativo il protocollo che si andrà a progettare potrebbe richiedere una sua propria forma di indirizzamento per

identificare la risorsa finale oggetto della comunicazione, come ad esempio il nome di un file.

Frammentazione e riassemblaggio

Se il protocollo progettato per il livello applicativo lo prevede, i **dati** dovranno essere **frammentati in blocchi più piccoli**, se necessario di dimensione fissata, e ogni blocco sarà successivamente passato al livello di trasporto. Se nel nuovo protocollo viene prevista questa fase, vista la tecnica di commutazione usata in Internet, di tipo packet switching, sarà probabilmente opportuno determinare una modalità per **numerare i frammenti**, per consentire di **riassemblarli** nel giusto ordine una volta raggiunta la destinazione (il protocollo applicativo paritario).

La **frammentazione** e il **riassemblaggio** sono utili in quanto permettono di:

- effettuare il controllo degli errori su ogni blocco;
- utilizzare in modo equo il mezzo trasmissivo;
- ritrasmettere un solo blocco in caso di errori, invece dell'intero messaggio originale;
- subire minori ritardi;
- avere un buffering di dimensione ridotte,

ma al contempo prevedono anche alcuni svantaggi, fra cui:

- una maggiore elaborazione in quanto i frammenti devono essere riassemblati in ordine;
- una minore quantità di dati trasmessi nell'unità di tempo vista la maggiore elaborazione richiesta (minor throughput).

Incapsulamento

L'**incapsulamento** definisce quali **informazioni di controllo** devono essere aggiunte ai dati e come vengono organizzate nel pacchetto, affinché due entità remote di pari livello possano gestire correttamente il payload.

L'incapsulamento prevede ad esempio l'aggiunta di:

- indirizzi;
- codici per il controllo degli errori;
- comandi di controllo per la gestione del dato;
- ecc.,

organizzandoli in un header ed eventualmente un tail.

Controllo della connessione

La progettazione di un nuovo protocollo comunicativo a livello applicativo servirà al perseguimento di uno specifico scopo e, in base a questo si dovrà decidere se il nuovo protocollo stesso dovrà essere orientato alla connessione o non orientato alla connessione, indipendentemente dal servizio scelto al livello di trasporto.

Se il nuovo protocollo è **orientato alla connessione** (*connection-oriented*) deve prevedere le seguenti fasi:

1. creazione della connessione;
2. trasmissione dei dati;
3. chiusura della connessione.

Se invece il protocollo **non è orientato alla connessione** (*connectionless*) queste fasi non sono previste.

Fasi di una connessione

Nella fase di **creazione della connessione** le entità si accordano su come scambiarsi i dati:

a. nel **caso** più **semplice** il mittente fa una richiesta di connessione e il ricevente la accetta o la rifiuta;
b. nel **caso** più **complesso** le due entità dovranno prevedere l'instaurazione di una connessione, oltre alla eventuale fase di negoziazione per garantire un determinato livello del servizio di trasmissione (*priorità*, *qualità del servizio*, *sicurezza*).

Nella fase di **trasferimento dei dati** si dovranno prevedere tutti o solo parte dei seguenti controlli:

a. **corretta sequenza dei blocchi di dati** prevedendo, ad esempio, nello header un codice che permetta di ricostruire il dato originale, nel caso sia stato frammentato, indipendentemente dall'ordine di arrivo dei pacchetti;
b. **controllo degli errori**, ad esempio utilizzando un CRC (*Cyclic Redundancy Check*);
c. **controllo del flusso**, ad esempio prevedendo un limite massimo di pacchetti che possono essere trasmessi per motivi di bufferizzazione.

Nella fase di **chiusura della connessione** le due entità concordano in qualche modo di terminare la comunicazione.

Servizio affidabile

A seconda dello scopo per cui viene creato il nuovo protocollo, si può prevedere che esso sia **affidabile** progettando una tecnica di **conferma** dei pacchetto ricevuti, oppure può essere previsto che **non** sia **affidabile** evitando tale progettazione.

Le modalità di conferma possono essere molto semplici come la conferma di ogni singolo pacchetto appena questo viene ricevuto, oppure prevedere tecniche più raffinate come la conferma cumulativa simile a quella adottata dal protocollo TCP.

Controllo degli errori

Se il protocollo che si sta progettando richiede la rilevazione degli errori, sarà necessario sviluppare o adottare una tecnica di **controllo degli errori** che permetta di individuare eventuali alterazioni dei dati e/o delle informazioni di controllo.

Le modalità adottabili possono essere le più varie, ad esempio si possono prevedere due fasi dove il destinatario individua gli errori e quindi richiede esplicitamente la ritrasmissione del dato, oppure il destinatario rileva l'errore scartando il pacchetto e demandando al mittente la ritrasmissione del dato che eventualmente non risulta confermato, come avviene in TCP, avvalendosi dell'ausilio di timer appositi.

Inoltre esistono diversi algoritmi che permettono di generare codici in grado solo di individuare l'errore, come i codici di parità, la check-sum o il CRC (*Cyclic Redundancy Check*), o addirittura di correggerli, come i codici di parità trasversale e longitudinale o il codice di Hamming[1].

Controllo del flusso

Il **controllo del flusso** è utilizzato dal ricevente per limitare o aumentare la velocità con cui la sorgente invia i dati; la sorgente non deve inviare più dati di quanti il ricevente ne possa processare, ma allo stesso tempo, se il ricevente fosse in grado di reggere la velocità di invio del mittente, sarebbe opportuno aumentare la velocità di spedizione.

La progettazione di un nuovo protocollo di comunicazione deve prevedere una modalità di controllo del flusso, in quanto da questa dipenderà anche la gestione delle risorse necessarie per la bufferizzazione dei dati ricevuti. Se questa gestione non viene progettata accuratamente si rischierà di perdere dei dati.

La modalità più semplice di gestione del flusso prevede che ad ogni invio di un dato il mittente debba attendere la conferma di ricezione da parte del destinatario. Chiaramente questa gestione è inefficiente, ma raggiunge perfettamente lo scopo in modo molto semplice.

Tecniche più avanzate possono prevedere la bufferizzazione di più dati e la conferma cumulativa dei segmenti, come avviene per TCP, ma ovviamente la sua implementazione sarà decisamente più complessa rispetto alla precedente.

Multiplexing e demultiplexing

Nelle reti di computer il **multiplexing** è una tecnica che permette a più connessioni di un determinato livello protocollare di confluire in una sola connessione a livello inferiore. Il **multiplexing** consente quindi di aggregare i dati e di ottimizzare l'efficienza della trasmissione.

Il ricevente dovrà necessariamente implementare un meccanismo di disaggregazione (**demultiplexing**) dei dati per consegnare i dati ricevuti al corretto destinatario.

---

[1] Per un approfondimento dell'argomento si veda Roberta Gerboni: *Codici rivelatori e correttori di errori*.

Nell'ambito delle reti le tecniche di multiplexing e demultiplexing sono ampiamente usate:

- a livello di trasporto i protocolli TCP e UDP usano i numeri di porta per distinguere tra i diversi protocolli del livello applicativo;
- a livello di rete il protocollo IP prevede il campo *Protocollo* nel suo header per capire a quale protocollo di trasporto consegnare il pacchetto;
- a livello data link il frame Ethernet prevede il campo *EtherType* nel suo header per consegnare i dati al giusto protocollo del livello di rete.

Progettando un nuovo protocollo di comunicazione a livello applicativo, lo sviluppo di tecniche di **multiplexing** e di **demultiplexing** implicano la necessità di aggregare i dati provenienti da livelli superiori, e di disaggregarli una volta ricevuti, e quindi il protocollo oggetto di progettazione sarà utilizzato almeno come trasporto da un altro protocollo, come avviene ad esempio nei Web Service SOAP (*Simple Object Access Protocol*).

Servizi di trasmissione

Nel caso il nuovo protocollo progettato richiedesse determinate caratteristiche di **servizi di trasmissione**, tali **servizi** dovranno essere **negoziati**, e quindi generalmente il nuovo protocollo sarà connection-oriented in quanto la negoziazione avviene di solito nella fase di creazione della connessione.

I **servizi di trasmissione** possono riguardare:

- la **priorità** dei messaggi, ad esempio rendendo alcuni messaggi di controllo prioritari sui dati, o ammettendo la possibilità di utenti con diversi livelli di priorità in base al tipo di contratto sottoscritto;
- la **qualità del servizio** che può riguardare il minimo ritardo o il minimo throughput accettabile;
- la **sicurezza** imponendo delle restrizioni di accesso al messaggio o all'intera comunicazione.

Ad esempio nello header IPv4 è presente il campo *Type of Service* che potrebbe permettere di impostare dei servizi di trasmissione a livello di rete, così come nello header IPv6 possono essere usati i due campi *Traffic Class* e *Flow Label*.

La gestione dei servizi di trasmissione è comunque **particolarmente complessa**, anche se di fatto la tendenza è quella di diversificare sempre più le comunicazione in base a questo aspetto.

Esempi di protocolli applicativi di rete

I protocolli applicativi di rete consentono la comunicazione tra processi utente su sistemi diversi e forniscono diversi servizi di comunicazione

Protocolli applicativi di rete esistenti

Di seguito sono elencati alcuni protocolli applicativi di rete comunemente usati su Internet:

- TELNET: terminale remoto;
- FTP: trasferimento di file;
- SMTP: invio della posta elettronica;
- HTTP: trasferimento di dati ipertestuali;
- BGP: protocollo di gestione del routing fra autonomous system diversi;
- SNMP: protocollo che consente la configurazione, la gestione e la supervisione di apparati collegati in una rete (dispositivi di rete o terminali utente), riguardo a tutti gli aspetti che richiedono azioni di tipo amministrativo;
- DNS: protocollo per la risoluzione dei nomi di dominio utilizzando un database distribuito;
- DHCP: protocollo che permette ai dispositivi di una rete locale di ricevere automaticamente la configurazione di rete necessaria per stabilire una connessione nella LAN;
- SSH: protocollo che utilizza tecniche crittografiche per effettuare operazioni di rete in modo sicuro su una rete non sicura;
- NTP: protocollo usato per la sincronizzazione degli orologi tra sistemi di computer;
- POP: protocollo per il recupero da parte di un client delle e-mail depositate in un server remoto;
- IMAP: protocollo per il recupero da parte di un client delle e-mail depositate in un server remoto;
- SFTP: protocollo per il trasferimento non sicuro di file, con un livello di complessità intermedio tra TFTP e FTP;
- TFTP: protocollo di trasferimento file molto semplice, che veniva utilizzato ad es. per l'avvio di computer che non hanno dispositivi di memoria di massa, come i router;
- ECHO: protocollo originariamente proposto per testare e misurare il round-trip time nelle reti IP.
- DAYTIME: protocollo di spedizione della data e dell'ora fornita da un server;
- TIME: protocollo di spedizione della data e dell'ora fornita da un server come numero di secondi a partire dal 1 gennaio 1900;
- DISCARD: protocollo che butta via qualsiasi dato riceva;
- CHARACTER GENERATOR: protocollo che genera e invia dati indipendentemente dall'input;
- QOTD: protocollo che invia un breve messaggio (*Quote Of The Day*);
- MESSAGE SEND PROTOCOL: protocollo per l'invio di brevi messaggi fra nodi di una rete;
- WHOIS: protocollo per il recupero delle informazioni di utenti Internet.

Un protocollo di spedizione all'Esame di Stato

L'Esame di Stato del 2018 per gli Istituti Tecnici Industriali di Informatica e Telecomunicazioni - Articolazione "Informatica" ha previsto una prova che richiedeva

l'implementazione di un protocollo per la gestione delle spedizioni di pacchi, dal mittente al destinatario. Anche se il protocollo richiesto non era di tipo informatico, le competenze richieste sono fortemente sviluppate durante il corso di TPSI che nel quinto anno è completamente incentrata allo sviluppo di protocolli di reti.

Di seguito sono proposte alcune soluzioni dell'Esame per consentirne un'analisi critica da parte dello studente:

- Soluzione Sistemi e Reti Esame di Stato 2018, Prof. Mauro De Brnardis;
- Prova scritta esame di stato 2018 - Informatica ITIA, Prof.ssa Giselda De Vita;
- Esame di Stato 2018 ITIA - Soluzione, Zanichelli online per la scuola e simple.bo;

Introduzione (rif. 1-2)

Il cloud sta assumendo un ruolo sempre più centrale nella nostra quotidianità. Smartphone e tablet hanno infatti rivoluzionato **il concetto di fruizione dei dati.** C'è sempre più l'esigenza di consultare fotografie, documenti, filmati e progetti in mobilità. I metodi classici di archiviazione (le memorie fisiche) non riescono più a sopperire a queste necessità e non a caso, negli ultimi anni, le grandi aziende hi-tech si sono lanciate nel magico mondo del cloud. Il risultato è che oggi è possibile sottoscrivere decine di servizi basati su questa tecnologia, che ormai ha invaso i settori più disparati. **Fare riferimento al semplice concetto di archiviazione è diventato riduttivo.**

Dietro la semplice parola "Cloud" si nascondono varie tecnologie molto complesse le quali, se ben comprese, possono davvero rivoluzionare il modo in cui si lavora da remoto nella stragrande maggioranza delle aziende attualmente in attività: dalla gestione della contabilità all'elaborazione di fatture e ordini, l'impiego del Cloud ha un margine d'azione pressoché smisurato.

Il sistema "Cloud computing", nella fattispecie, identifica **dei servizi per cui viene eseguita la distribuzione del calcolo su Internet**, evitando il più possibile l'uso di risorse locali dedicate. Se l'attività possiede dei server, vari database, software di analisi o altre tecnologie locali (siano esse hardware o software), è possibile prendere in esame l'idea di spostarle completamente sul cloud, così da ottimizzare le risorse a disposizione dell'azienda.

Che cos'è il Cloud? (rif. 1-2)

Il cloud, o più precisamente cloud computing (letteralmente "nuvola informatica"), è un servizio offerto da un insieme di computer (di server, per la precisione) che possono anche essere disseminati nel mondo. Un qualcosa che, in passato, veniva messo a disposizione da macchine localizzate nel medesimo posto fisico. Questa evoluzione è stata possibile grazie all'incredibile miglioramento dell'infrastruttura di rete, che oggi può contare sulla fibra ottica e su tecnologia inimmaginabili fino a un decennio fa (vedi le Reti 4G e le ancor più recenti 5G).

La "nuvola informatica" nasce dunque per sfruttare le potenzialità della nuova infrastruttura di rete in abbinamento alla capacità dei server di lavorare in contemporanea, mettendo a disposizione **un'enorme capacità di calcolo.** Ecco dunque la creazione dei servizi più disparati basati su questa tecnologia, in grado di

superare i limiti di una singola macchina, specialmente in relazione alla portata di elaborazione dei dati e, ancora di più, allo spazio fisico. Per tale ragione, quando si parla di cloud, è bene far riferimento a tre tipologie: archiviazione, elaborazione e trasmissione di dati.

La tipologia di cloud più diffusa è quella dell'archiviazione dati. È proprio in quest'ambito infatti che le grandi aziende hi-tech hanno deciso di espandersi, proponendo alcuni servizi divenuto popolari a livello mondiale. In tal senso, basti pensare come i guadagni di una realtà come Microsoft siano legati oggi, in buona parte, a questa tecnologia per comprendere quanto sia ormai diffusa e non solo in ambito consumer ma anche e soprattutto nel B2B (Business To Business).

Come funziona l'archiviazione cloud e dove si trovano i dati (rif. 1)

Sono tanti ad utilizzare questa tecnologia inconsapevolmente, senza saperlo. Un esempio classico sono i possessori di smartphone Android che necessariamente sono titolari anche di un account Google: come fa la rubrica dei contatti a comparire magicamente su ogni nuovo telefono? Semplice, merito del cloud storage! In realtà il funzionamento dell'archiviazione cloud è legata al profondo rapporto intercorrente tra i dispositivi con SO Android e i servizi dell'azienda di Mountain View, cioè la sede della Google LLC. Il progetto Google nasce come una cordata commerciale per agevolare la diffusione di specifici prodotti hardware e software.

Salvare un file nella "nuvola" **è un'operazione molto simile al salvataggio su un classico hard disk.** Per l'utente finale è esattamente identica: infatti si spostano fotografie, documenti, progetti e filmati in una cartella. Dietro a tutto questo ci sta un servizio cloud: i file anziché essere salvati fisicamente nel proprio computer (o smartphone o tablet) finiscono sui server del fornitore del servizio, sparsi fisicamente in giro per il mondo, sfruttando la connessione offerta da Internet.

A questo punto è importante soffermarsi su una precisazione: **i server sono ovviamente di proprietà del fornitore del servizio** (ad esempio Apple nel caso si faccia riferimento ad iCloud) **ma i file rimangono di proprietà dell'utente**. Una condizione esplicitata in maniera chiara nella documentazione fornita prima della sottoscrizione di un qualsivoglia abbonamento. Un passaggio spesso sottovalutato ma che è opportuno affrontare dedicandoci tutto il tempo necessario è quello di **leggere sempre la policy**, ossia il regolamento di utilizzo del servizio prima di affidare i propri dati personali a un'azienda terza.

I vantaggi del cloud (rif. 1-2)

Il Cloud Computing presenta numerosi vantaggi, a prescindere dalla categoria dell'azienda, dell'organizzazione o, in generale, dell'attività svolta in ufficio. Di seguito vengono elencati i principali e più conosciuti di questi vantaggi.

- **Risparmio sui costi**: il primo vantaggio e spesso quello che spinge le aziende ad adottare soluzioni cloud. Con il Cloud Computing infatti si risparmia notevolmente sui costi di gestione della piattaforma informatica (o di parte della stessa), visto che l'hardware viene gestito dalla società di cloud a cui ci si rivolge (inclusa anche la manodopera e i costi di gestione). Con il Cloud Computing l'investimento iniziale ritornerà in fretta e dopo qualche tempo si avrà un grande risparmio alla voce "infrastruttura informatica". Eliminare le elevate spese iniziali, come gli investimenti in server e data center interni, vuol dire moltissimo. Invece di acquistare hardware e il software, è possibile noleggiarli dal fornitore di servizi cloud. Ciò significa che il fornitore sarà quello che si farà carico dei costi di installazione, esecuzione, gestione, protezione e aggiornamento costante della tecnologia.

- **Pay per Use:** i servizi cloud vengono spesso erogati a fronte di pagamenti legati all'effettivo utilizzo. Si tratta di un approccio estremamente diffuso soprattutto nelle tipologie di elaborazione e trasmissione dei dati, con particolare riferimento ai server dei siti web. Nell'ambito dell'archiviazione è molto frequente imbattersi in servizi in abbonamento, che prevedono dunque un corrispettivo mensile (o annuale) a fronte di un certo quantitativo di spazio per "accatastare" i propri file.

- **Prestazioni**: le società che forniscono i servizi Cloud Computing, denominate generalmente **Cloud Provider**, offrono dei server in parallelo molto potenti, difficilmente replicabili in locale senza l'impiego di una grande quantità di risorse e denaro. Puntando sul Cloud Computing si avrà quindi una velocità d'esecuzione elevata, qualsiasi possa essere il programma, il sistema operativo o il database che sarà necessario caricare sul cloud.

- **Gestione IT veloce**: appaltare esternamente la manutenzione, avere accesso a un'assistenza tecnica specializzata disponibile 24 ore su 24, non dover necessariamente possedere complesse infrastrutture. Sono solo alcuni dei vantaggi offerti dal cloud in ambito aziendale, che inevitabilmente semplificano la gestione IT di una qualsiasi società. I sistemi di information technology possono rappresentare un vero e proprio collo di bottiglia nel business e la "nuvola" è in grado di far bypassare questa potenziale criticità.

- **Scalabilità**: altro vantaggio non indifferente è la scalabilità dei servizi cloud, ossia la possibilità di riadattare "al volo" la potenza di calcolo necessaria, in base al numero di utenti da connettere e/o dell'attività svolta in azienda. Se inizialmente le necessità aziendali richiedono soluzioni con minore capacità di calcolo e con un numero massimo di utenti connessi, con la crescita dell'azienda sarà possibile espandere le potenzialità della piattaforma cloud in pochi passi, aumentando la potenza di calcolo man mano che l'azienda raggiunge traguardi importanti.

- **Affidabilità**: la maggior parte dei provider Cloud Computing offre sistemi di backup integrati e automatici, che rendono davvero molto semplice ripristinare il lavoro svolto o i dati conservati in caso d'attacco/disastro informatico o di un'infezione virale proveniente dall'interno. I dati saranno sempre disponibili e, in caso di problemi, sarà sempre possibile ripristinare uno dei tanti backup presenti nel sistema.
  Il sistema di server su cui si basa il cloud la rende una tecnologia meno soggetta a problematiche rispetto a soluzioni basate su macchine in locale. Lo dicono i numeri ma lo suggerisce anche la logica. Infatti, prima dell'avvento dei vari Microsoft Azure, Google Cloud Platform o Amazon Web Services, l'interruzione di servizi poi passati sul web era all'ordine del giorno. Se anche un solo server dovesse non funzionare adeguatamente, tutti gli altri possono essere in grado di sopperire alla sua mancanza.

- **Sicurezza**: visto che le piattaforme Cloud Computing sono spesso progettate per gestire attività e dati importanti e che richiedono massima riservatezza, la sicurezza è ai massimi livelli: il provider Cloud scelto si occuperà di garantire la sicurezza dei dati che transitano sui server, oltre a garantire un controllo

antivirale su ogni link o su ogni tipo di file immagazzinato. Difficilmente è possibile raggiungere il livello di sicurezza offerto dal Cloud Computing con una piattaforma o infrastruttura di tipo locale.

Questi vantaggi hanno spinto molte aziende ad adottare il Cloud Computing in pianta stabile, aumentandone notevolmente la produttività.

Tipi di cloud computing (Modelli di distribuzione) (rif. 1-2-3)

Essendo il cloud un servizio a tutti gli effetti, ci sono inevitabilmente differenti metodi di erogazione. È possibile riassumerli in quattro tipologie che poi spesso si intersecano tra di loro, creando sinergie che possono risultare cruciali per la sua corretta implementazione.

- **Cloud pubblico**
  È un servizio che fornisce servizi IT pubblicamente su Internet. Per fare questo i fornitori gestiscono gruppi di server interconnessi, le così dette **server farm**. Gli utenti accedono allo spazio di archiviazione attraverso un browser. La caratteristica peculiare del servizio è che si paga solo l'ammontare di risorse che si consuma. Si risparmiano dunque i costi di acquisto del relativo hardware, avendo comunque costantemente accesso on-demand a un pool di strumenti informatici, scalabili in base alle esigenze.
  Su un cloud pubblico l'hardware, il software e l'infrastruttura di supporto appartengono al provider scelto, che li gestisce e li ottimizza automaticamente. Di fatto si accede a risorse già pronte all'uso, pronte per lavorarci sopra senza dover configurare praticamente nulla (o quasi). Un service provider pubblico mette a disposizione risorse come applicazioni e storage con la logica del Pay per Use.
  In sintesi nel cloud pubblico i vantaggi sono la flessibilità, l'economicità ed i minori costi di manutenzione. Di contrappunto è impossibile avere un diretto controllo dell'infrastruttura da parte dell'utente finale e della sicurezza (**quindi nel cloud pubblico né infrastruttura né sicurezza**)
- **Cloud privato**
  È un servizio che fornisce servizi di cloud computing tramite Internet o una rete interna agli utenti autorizzati. Essendo fruibile solo da uno specifico gruppo di utenti definito in maniera chiara, il cloud privato viene anche definito come cloud aziendale o cloud interno. Gli utenti che hanno accesso a questa tipologia di servizio beneficiano di vantaggi come la scalabilità e l'elasticità, ma

anche di ulteriori opzioni di controllo e adattamento. Di fatto i fornitori mettono a disposizione risorse dedicate, in grado di fare la differenza in determinati ambiti.

Su un cloud privato le risorse sono gestite all'azienda o dall'organizzazione ed è possibile accedervi solo dalla rete locale o con connessioni remote dedicate (VPN). Ogni aspetto del cloud è gestito dal personale IT dedicato, così come ogni miglioria od ottimizzazione.

Un service provider privato mette a disposizione risorse come applicazioni e storage con la logica del Pay per Use, come nel cloud pubblico, soltanto con vincoli più stringenti e maggior personalizzazione dei servizi. Tutto questo però all'interno della rete aziendale e non in quella di dominio pubblico (Internet).

I principali vantaggi di questo secondo modello sono quello di avere un ambiente completamente autogestito, unitamente ad un pieno controllo della privacy e della sicurezza. Gli svantaggi sono quelli di avere una ridotta scalabilità rispetto al cloud pubblico e maggiori costi di gestione.

- **Cloud ibrido**

  È un servizio, appunto, ibrido e per questo estremamente potente. I servizi cloud ibridi offrono infatti alle aziende un maggiore controllo sui propri dati privati. Un'organizzazione può infatti decidere di archiviare dati sensibili su un cloud privato o un datacenter locale (con tutti i vantaggi a esso connessi) e, contemporaneamente, sfruttare quelle che sono le tipiche risorse di computing avanzato di un classico cloud pubblico gestito.

  Il cloud ibrido rappresenta un buon compromesso tra cloud privato e cloud pubblico: un provider di servizi può installare un cloud privato in azienda ma offrire anche una o più componenti tipiche del cloud pubblico, così da poter separare le risorse tra i vari dipartimenti presenti in azienda. In sintesi il cloud ibrido cerca di unire i vantaggi di quello pubblico ai vantaggi di quello privato.

  Un sistema di cloud ibrido è molto più efficiente, riesce a garantire una maggior privacy e sicurezza dei servizi. Inoltre riesce a gestire al meglio i picchi di carico, cioè un elevato ed improvviso numero di utenti vuole utilizzare quel particolare servizio. In ultima analisi il principale svantaggio del cloud ibrido è quello di dover integrare tra di loro infrastrutture eterogenee, dunque profondamente diverse.

- **Multi cloud**
  Si intende la presenza del deployment di più cloud dello stesso tipo (pubblico o privato) offerti da diversi fornitori. Un approccio multi cloud può quindi prevedere due ambienti di cloud pubblico o due ambienti di cloud privato. La presenza di deployment multi cloud, sia pubblici che privati, è molto frequente nelle aziende il cui obiettivo è migliorare sicurezza e le prestazioni attraverso l'utilizzo di più ambienti.

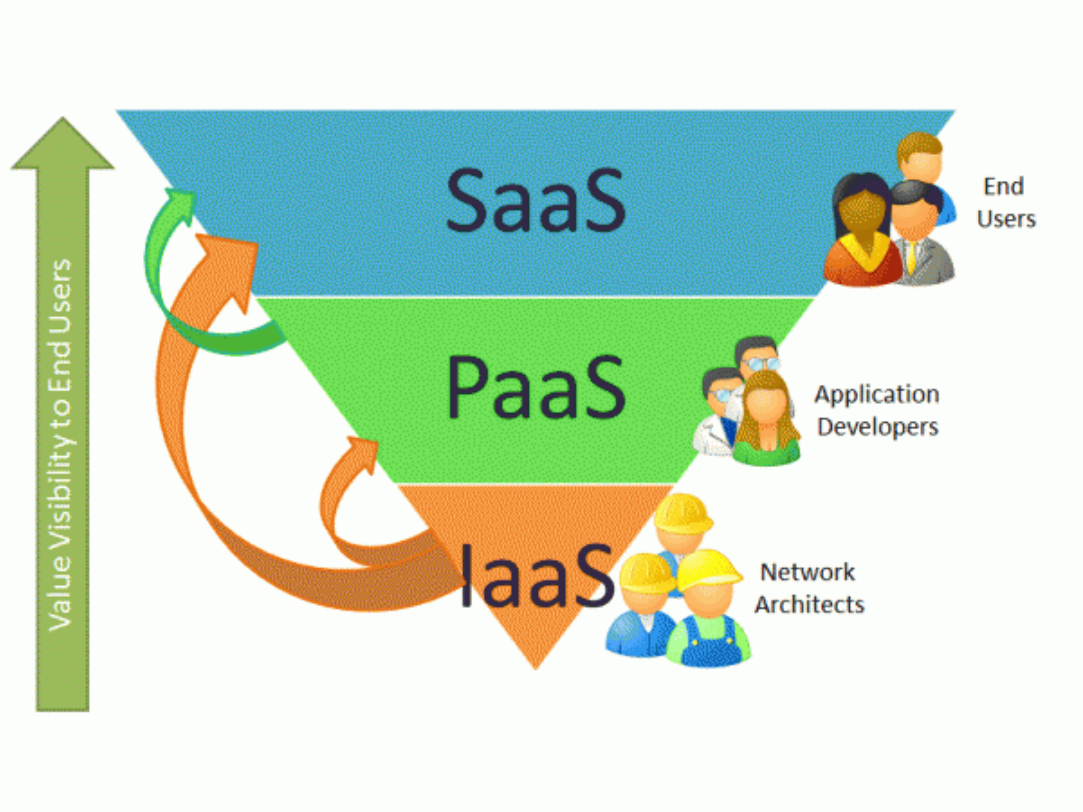
I servizi di un'infrastuttura cloud (rif. 2-3)

All'interno di un'infrastruttura cloud ci sono tra attori principali:

- il fornitore: l'azienda o l'entità che eroga il servizio;
- l'amministratore: l'azienda terza o l'entità che amministra questo servizio e che si interfaccia con l'utente finale;
- il cliente finale (utente): azienda/privato cittadino che sfrutta il servizio.

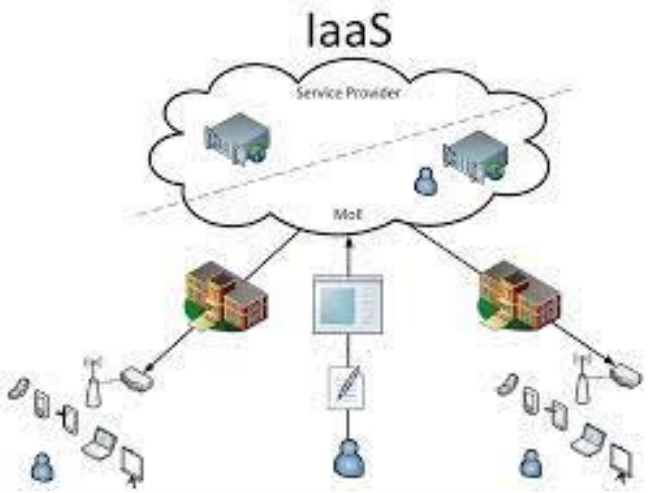
I tre sistemi tipici della tecnologia cloud sono:

a) **SaaS**: **S**oftware **a**s **a** **S**ervice
b) **PaaS**: **P**latform **a**s **a** **S**ervice
c) **IaaS**: **I**nfrastructure **a**s **a** **S**ervice



In dettaglio:

a) Con **IaaS** si identifica il cloud basato sull'infrastruttura distribuita come servizio: il provider mette a disposizione un'intera infrastruttura remota, composta da hardware e risorse dedicate, liberamente configurabile. Con esso si avrà a disposizione ampio margine di scelta sulla tipologia di sistema IT (server o macchine virtuali), sulle risorse per l'archiviazione e su altri elementi utili come sistemi operativi e banda. Per accedere a questo tipo di servizio cloud è sufficiente pagare una quota, generalmente variabile in base al consumo effettivo di risorse, incrementando (anche in maniera automatica) le risorse impiegate.

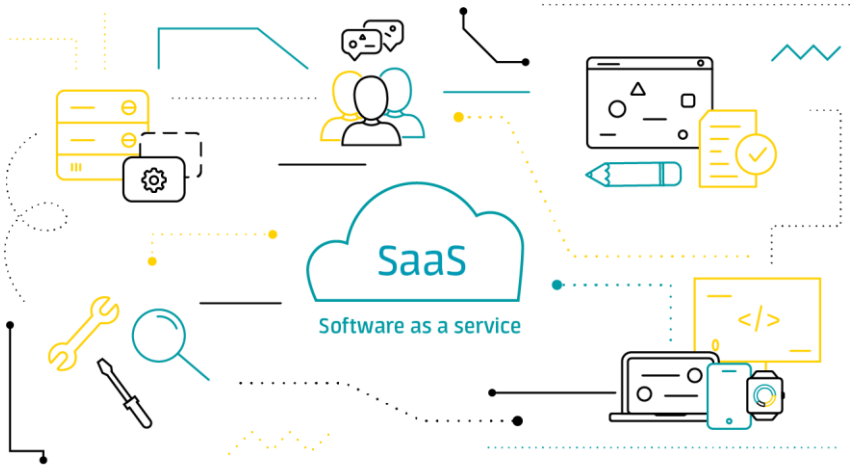


IaaS rappresenta il livello di astrazione più basso in cui il fornitore mette a disposizione soltanto la potenza di calcolo e lo spazio di archiviazione. L'utente, una volta installato il software, gestisce l'hardware ed il suo monitoraggio. Le risorse hardware vengono allocate soltanto quando necessarie.

b) Con **PaaS** si identifica invece il cloud basato sulla piattaforma distribuita come servizio. Con esso sarà possibile utilizzare piattaforme già pronte all'uso e ottimizzate dal provider per l'implementazione di soluzioni personali di sviluppo, testing ed erogazione di applicazioni aziendali. Queste sono le soluzioni di partenza più utilizzate dalle aziende che si avvicinano per la prima volta al Cloud Computing, visto che la parte più difficile da gestire – proprio l'infrastruttura IT – viene lasciata al controllo da parte del provider. La quota mensile varia, anche questa volta, in base alla categoria e alla quantità di risorse richieste.

PaaS è definito *Middleware*. In sostanza si tratta di una piattaforma erogata sotto forma di servizio attraverso delle specifiche API in modo da interfacciare diversi sistemi. Si tratta di una scelta vantaggiosa perché consente agli sviluppatori di incrementare la produttività in azienda. Questi tipo di servizio favorisce la collaborazione tra utenti su piattaforme differenti o tra software che condividono dati utili all'integrazione.

c) Con **SaaS** si identifica il cloud basato sul software come servizio. Con esso la parte software (programmi e sistemi operativi) e hardware (infrastruttura) è già pronta all'uso e accessibile tramite Internet. Con questo servizio si avrà, di fatto, un pacchetto "tutto incluso", con cui usare il programma scelto senza dover pensare a nulla, visto che viene lasciato tutto nelle mani del provider.

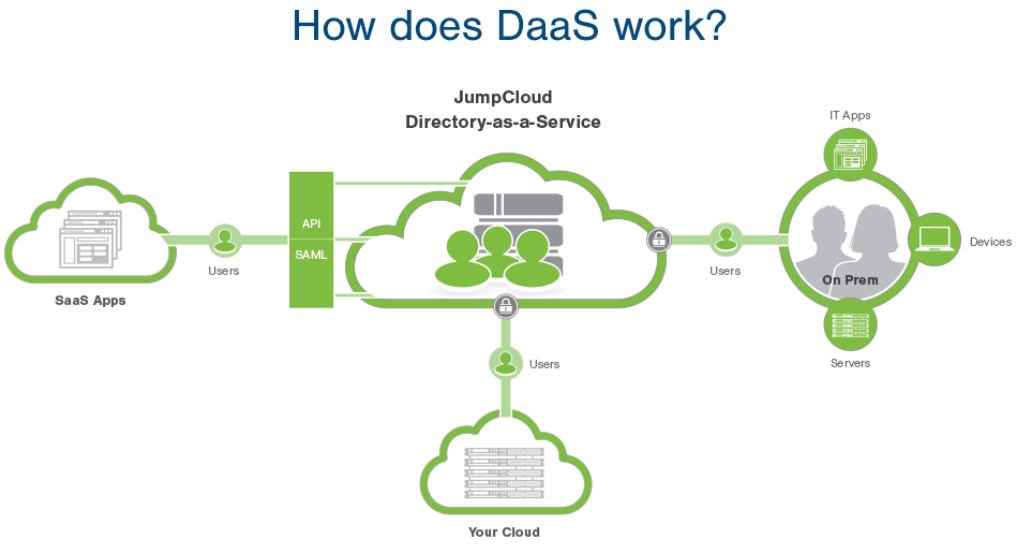


Si tratta, dunque, di un modello di distribuzione del software on demand ed è il principale vantaggio è quello che l'utente fruisce sempre della versione più aggiornata del software senza necessità di installazione, di configurazione del software e di aggiornamento. E' il sistema di cloud più diffuso tra i tre principali finora visti. Lo logica è sempre quella del Pay per Use. Non necessita di uno specifico hardware e c'è un'elevata scalabilità.

Esistono altre tipologie di sistemi che supportano la tecnologia cloud. I due sistemi più noti che si vanno ad aggiungere ai tre precedenti sono: DaaS e HaaS.

Rispettivamente:

d) **DaaS** significa **D**irectory **a**s **a** **S**ervice ed identifica il cloud basato sui dati come un servizio. Derivato del SaaS, questo approccio consente di gestire una grande mole di dati (spesso in formato database) e di rispondere alle richieste dei singoli utenti in pochi secondi, senza occupare troppe risorse di sistema locale. Questo tipo di cloud è l'ideale per chi deve lavorare su più database o, in generale, grandi quantità di file, ma desidera comunque grande velocità quando effettua una richiesta (che verrà evasa impiegando esclusivamente le risorse del provider).



e) **HaaS** sta per **H**ardware **a**s **a** **S**ervice ovvero il cloud basato sull'hardware come un servizio. Anch'esso derivato dal SaaS, permette di "affittare" una determinata quantità di risorse e di lasciare al provider anche la gestione della componente software (come per esempio l'aggiornamento del sistema operativo o delle varie applicazioni). In questo modo è possibile, per esempio, creare uno o più computer virtuali su cui lavorare da remoto, senza dover ogni volta cambiare le componenti interne e senza dover pensare a formattazioni o eventuali operazioni di manutenzione.

Utilizzi concreti del cloud computing (rif. 2)

Viste tutte le tipologie di Cloud Computing e di come vengono erogate dai vari provider è utile vedere quali potrebbero essere gli utilizzi concreti di questa tecnologia. Poiché un'azienda deve investire del denaro, deve comprendere bene se il Cloud Computing è adatto alle proprie necessità o se sia preferibile tenere ancora una soluzione server-client locale.

Ecco alcuni esempi validi di come il Cloud Computing può essere impiegato in azienda:

- **Smart Working**: con il Cloud Computing è possibile lavorare da casa avendo a disposizione le stesse risorse (hardware e software) disponibili in un ufficio tradizionale, senza doversi preoccupare della gestione dell'infrastruttura (se scegliete di operare in modalità PaaS o SaaS).

- **Conservazione dei dati**: con il Cloud Computing si può ricreare uno spazio virtuale cloud nel quale conservare tutti i documenti o i file generati dall'azienda, separandoli anche in base al dipartimento o dal ramo da cui provengono. Anche in caso di blackout improvviso o di altri incidenti in sede, i dati saranno sempre accessibili online.

- **Lavoro multiutente condiviso**: con il Cloud Computing è possibile far lavorare più dipendenti sullo stesso progetto o sullo stesso file, così da poter velocizzare tutti i processi di produzione e introdurre anche servizi aggiuntivi per la revisione del lavoro svolto.

- **Testare e compilare le applicazioni**: si possono creare e testare nuove applicazioni sul cloud, così da accedere rapidamente a tutti gli strumenti di sviluppo e le risorse necessarie alla compilazione, senza dover attrezzare dei server o delle macchine di test locali.

- **Trasmettere in streaming**: con il cloud è possibile svolgere il lavoro in sede e mostrarlo a tutti i dipendenti in smart working, così come realizzare conferenze con un numero elevato di dipendenti separando anche le dirette in base al dipartimento con cui comunicare.

- **Fornire software on demand**: con le piattaforme di tipo SaaS è possibile rendere un'applicazione disponibile per tutti i dipendenti e gli utenti senza dover installare nulla sui computer: si avvia il browser, si digitano le credenziali

d'accesso e si è subito operativi, con la massima efficacia e con la massima velocità possibile.

Se l'azienda utilizza delle soluzioni locali o poco performanti per una qualsiasi delle finalità mostrate poc'anzi, potrebbe essere una buona idea prendere in considerazione l'approccio Cloud Computing, visto che i vantaggi sarebbero davvero notevoli e gli eventuali costi iniziali subito ammortizzati dopo pochissimi mesi.

Cloud ibridi ed ambienti multicloud (rif. 4)

Tra i modelli di distribuzione cloud analizzati i due più diffusi sono quello ibrido e ancor di più quello multi. E' utile conoscere questi due modelli in maniera più approfondita, soprattutto nell'ottica un giorno di doverne scegliere uno come soluzione a problemi legati all'azienda per la quale lavoriamo.

1) **I migliori servizi di cloud ibrido**
   Il mercato mette a disposizione un'enormità di servizi: il cloud possiede un potenziale economico tale da aver attirato l'attenzione di buona parte delle aziende big dell'hi-tech: da Amazon a Google, da Apple a Microsoft.
   Di conseguenza tutte le aziende hi-tech offrono soluzioni basate sul cloud computing, ottenendo enormi guadagni e successi economici. Nel caso specifico del cloud ibrido, di seguito vengono proposte a titolo di esempio alcune tra le migliori alternative presenti su piazza.

   - **Cloud Platform:** offerto da Google, è una suite di servizi di cloud computing che gira sulla stessa infrastruttura utilizzata internamente per i suoi prodotti per gli utenti finali, come Ricerca Google, Gmail, archiviazione di file e YouTube. In relazione al cloud ibrido, la soluzione dell'azienda di Mountain View mette a disposizione la tecnologia denominata "Dedicated Interconnect". Questa fornisce connessioni fisiche dirette tra la rete locale del cliente e quella del gigante californiano. In questo modo è possibile trasferire grandi quantità di dati tra le reti, il che può essere più conveniente rispetto all'acquisto di larghezza di banda aggiuntiva su Internet pubblico.

- **Amazon Web Services Inc.** è un'azienda di proprietà del gruppo Amazon, che fornisce servizi di cloud computing su un'omonima piattaforma on demand. Questi sono operativi in 20 regioni geografiche in cui l'azienda di Seattle ha virtualmente suddiviso il globo. Tra i servizi spicca certamente "Direct Connect" che semplifica la creazione di connessioni di rete dedicate dalle imprese ad AWS. AWS Direct Connect consente di stabilire una connettività privata tra AWS e un data center, un ufficio o un ambiente condiviso. In molti casi, questa caratteristica può contribuire a ridurre i costi di rete, aumentare il throughput della larghezza di banda e fornire un'esperienza di rete più uniforme rispetto alle connessioni basate su Internet.
- **Microsoft Azure** è la piattaforma cloud pubblica di Microsoft, che offre servizi di cloud computing. Tra questi, in relazione al cloud ibrido, è importante citare "ExpressRoute", che serve a stabilire connessioni private tra i data center di Azure e l'infrastruttura presente in locale o in un ambiente con più sedi. Le connessioni ExpressRoute non usano la rete Internet pubblica e offrono maggiore affidabilità, velocità superiore e minore latenza rispetto alle connessioni Internet tradizionali. In alcuni casi, l'uso di connessioni ExpressRoute per trasferire dati tra sistemi locali e Azure può offrire vantaggi significativi in termini di costi.

- **OpenStack** è una piattaforma cloud estremamente diffusa soprattutto negli Stati Uniti, in grado però di ritagliarsi un importante spazio anche in Europa. In particolare, il programma OpenStack Global Passport rappresenta una collaborazione tra i provider di cloud pubblico OpenStack al fine di per consentire una maggiore libertà, migliori prestazioni e interoperabilità dell'infrastruttura open source. È possibile accedere rapidamente e facilmente all'infrastruttura OpenStack tramite programmi di prova dei provider di cloud pubblico OpenStack partecipanti in tutto il mondo.

2) Sicurezza del cloud ibrido

Considerando la natura intrinseca del cloud ibrido, spesso questa tecnologia è oggetto di veri e propri pregiudizi in merito alla propria sicurezza. Questi sistemi spesso prevedono contemporaneamente l'utilizzo di un'infrastruttura proprietaria abbinata a quella del provider del servizio. Una situazione

intermedia che fa appunto nascere dubbi. Un cloud ibrido progettato, integrato e gestito in modo appropriato è sicuro tanto quanto un ambiente IT on-premise tradizionale. Sebbene esistano problematiche di sicurezza uniche del cloud ibrido (come la migrazione dei dati, una maggiore complessità e una più ampia superficie vulnerabile agli attacchi), la presenza di più ambienti può essere una delle migliori difese contro i rischi relativi alla sicurezza. Tali ambienti interconnessi consentono alle aziende di scegliere dove collocare i dati sensibili in base alle proprie esigenze. I team della sicurezza, di contro, possono standardizzare gli storage cloud ridondanti per diminuire le attività di disaster recovery.

3) Differenza tra multicloud e cloud ibrido

Un approccio multicloud può prevedere due ambienti di cloud pubblico o due ambienti di cloud privato. Un cloud ibrido può prevedere un ambiente cloud pubblico e un ambiente cloud privato con un'infrastruttura (agevolata da API o container) che facilita la portabilità del carico di lavoro. Il primo si differenzia dunque dal secondo in quanto si riferisce a più servizi cloud piuttosto che a più modalità di distribuzione (pubblica, privata, legacy). Inoltre, in un ambiente multicloud, la sincronizzazione tra diversi fornitori non è essenziale per completare un processo di calcolo, a differenza del calcolo parallelo o degli ambienti di calcolo distribuito.

E' possibile dunque concludere, dopo aver analizzato in maniera approfondita le due tecnologie, **che questi approcci cloud si escludono a vicenda:** non possono coesistere allo stesso tempo perché i cloud saranno interconnessi (cloud ibrido) oppure no (multicloud). La presenza di deployment multicloud, sia pubblici che privati, è molto frequente nelle aziende il cui obiettivo è migliorare sicurezza e le prestazioni attraverso l'utilizzo di più ambienti.

4) Perché scegliere una soluzione multicloud

La tendenza nel mondo cloud che si sta evidenziando con maggiore frequenza, stando ai dati rilevati dalle maggiori società di analisi di mercato a livello mondiale, è proprio quella dell'introduzione nelle aziende di strategie multicloud. Le ragioni sono molte. Una è sicuramente la propensione, in un'epoca in cui il business diventa sempre più digitale, e quindi dipendente dalla disponibilità (availability) delle applicazioni e dei dati più vicina possibile

al 100%, a non tenere i propri software e le proprie informazioni mission-critical e business-critical in singole infrastrutture il cui costo di manutenzione e aggiornamento sarebbe difficile da sostenere per la singola impresa.

Un'altra, molto rilevante, è la propensione a voler essere liberi nella scelta del provider che risulti più competitivo rispetto a determinati tipi di esigenza. Per questa ragione, sempre secondo le ricerche degli analisti, vi sono aziende che utilizzando tre o quattro public cloud ed altri che arrivano anche a dieci o dodici. E le esigenze, come è noto, si suddividono in servizi di tipo infrastrutturale (Infrastructure as a Service, IaaS), applicativo (Software as a Service, SaaS) e come piattaforme di development e testing (Platform as a Service, PaaS).

Ci sono però quattro aspetti del multicloud che stanno facendo la differenza al momento della scelta da parte delle aziende.

- **Shadow IT:** è ormai una realtà che contribuisce al multicloud. L'hardware o il software distribuito in modo indipendente dal team IT centrale può acquisire dimensioni tali da richiedere una maggiore supervisione. In tal caso, la migrazione dell'infrastruttura e dei dati verso un sistema preferito (ad esempio i cloud pubblici) potrebbe essere un approccio inadeguato. Il deployment dello shadow IT si aggiunge ai cloud in uso dall'azienda, con la conseguente creazione di un multicloud.
- **Flessibilità:** È facile trovare la soluzione cloud perfetta per una singola esigenza: un cloud proprietario adatto per l'hosting di un'app proprietaria, un cloud per l'archiviazione di dati pubblici, o un cloud con elevata scalabilità idoneo a ospitare sistemi con frequenza d'utilizzo altamente variabile. Tuttavia, un solo cloud non potrà soddisfare tutte le esigenze al meglio.
- **Prossimità:** Per migliorare i tempi di risposta per gli utenti che si trovano a migliaia di chilometri di distanza dalla sede centrale di un'azienda, alcuni carichi di lavoro potrebbero essere ospitati da provider di servizi cloud che operano in prossimità degli utenti. Questa soluzione consente all'azienda di mantenere un'elevata disponibilità e di rispettare le leggi sulla sovranità dei dati, ovvero protocolli che regolamentano i dati in base alle normative del paese in cui essi sono ubicati.
- **Failover:** Gli ambienti multicloud aiutano a garantire l'operatività dei sistemi aziendali. Come soluzione di failover, un ambiente multicloud

offre alle aziende un backup disponibile e altamente scalabile per dati, flussi di lavoro e sistemi, in caso di problemi al cloud primario.

Riferimenti:

1) www.punto-informatico.it/guida-cloud-tutto-quello-che-ce-da-sapere/#h158977-1
2) www.punto-informatico.it/cloud-computing-cosa-e-come-funziona-esempi/
3) www.youtube.com/watch?v=D6K2np9kngM
4) www.punto-informatico.it/cloud-ibridi-ambienti-multicloud/

La connessione tramite socket

1) Generalità

I socket sono stati introdotti come Release 4.2 di BSD (Berkley Software Distribution) nel 1983 e, ad oggi, rappresentano lo "standard de facto" per la programmazione di rete.

L'idea di socket è stata sviluppata come estensione del *paradigma* UNIX di I/O su file. In informatica, un *paradigma di programmazione* è uno stile fondamentale di programmazione, ovvero un insieme di strumenti concettuali forniti da un linguaggio di programmazione per la stesura del codice sorgente di un programma, definendo dunque il modo in cui il programmatore concepisce e percepisce il programma stesso. Diversi paradigmi si differenziano per i concetti e le astrazioni usate per rappresentare gli elementi di un programma (come ad esempio le funzioni, gli oggetti, le variabili, vincoli, ecc.) e per i procedimenti usati per l'esecuzione delle procedure di elaborazione dei dati (assegnazione, calcolo, iterazione, data flow, ecc).

I socket sono disponibili su tutte le piattaforme UNIX e Mac OS e, sebbene con sintassi di programmazione lievemente differenti, su altri tipi di Sistemi Operativi (SO) come Microsoft Windows (WinSock).

Il loro successo è dovuto al fatto che costituiscono un'interfaccia estremamente potente e flessibile, poiché:

- consentono la comunicazione interprocesso sia *localmente*, ossia nell'ambito di uno stesso sistema, sia *in remoto*, ovvero tra sistemi differenti interconnessi tramite rete;

- sebbene il loro impiego più diffuso sia con la famiglia di protocolli di Internet (TCP/IP), possono essere utilizzati con molte altre famiglie di protocolli di rete (Appletalk, X.25, …)

Il concetto di Socket si basa sulla sequenza di istruzioni *open-read-write-close*.

Le istruzioni corrispondono a ben precisi e chiari comandi da eseguire sui file, ossia:

- ✓ *open* : questa istruzione consente di accedere ad un file;
- ✓ *read/write* : queste due istruzioni permettono di accedere direttamente ai contenuti di un file (rispettivamente sono istruzioni di lettura e di scrittura);
- ✓ *close* : questa istruzione termina l'utilizzo di un file.

L'uso dei socket in pratica viene eseguito con la stessa modalità, ossia sequenzialmente, ma aggiungendo alla precedente struttura l'insieme di parametri necessari a realizzare la connessione tra macchine remote.

I suddetti parametri sono:

- ✓ gli indirizzi;
- ✓ il protocollo ed il numero di porta;
- ✓ il tipo del protocollo**.**

Ciascun sistema operativo mette a disposizione nelle API i meccanismi e le procedure necessarie per attuare l'interfacciamento tra diversi protocolli. In particolare le *socket API* sono delle specifiche *API* di protocollo. Queste particolari *API* traggono origine da Berkley BSD UNIX ed oggi sono disponibili sui più diffusi sistemi operativi, quali Windows, Solaris e Linux.

La connessione tramite socket si avvale di tutta una serie di funzioni scritte principalmente in *C* ed in *Java*. Di seguito vengono elencate e descritte le più comuni, nonché quelle utilizzate ai fini del nostro corso.

Le principali istruzioni per la comunicazione tramite socket sono:

- ✓ *socket()/server socket()* : serve per la creazione di un nuovo socket;
- ✓ *close()* : pone fine all'utilizzo di un socket;
- ✓ *bind()* : collega un indirizzo di rete (IP address) ad un socket;
- ✓ *listen()* : aspetta messaggi in ingresso;
- ✓ *accept()* : inizia ad utilizzare una connessione in ingresso;
- ✓ *connect()* : crea una connessione con un host remoto;
- ✓ *send()/write()* : trasmette dati su una connessione attiva;
- ✓ *rcv()/read()* : riceve dati da una connessione attiva.

2) Famiglie e tipologie di socket

La logica alla base della programmazione con i socket è abbastanza semplice: per realizzare i vari tipi di comunicazione viene utilizzato sempre lo stesso insieme di *API* generiche, precisandone la semantica (ossia il funzionamento) attraverso opportuni argomenti.

I socket permettono di specificare il tipo di comunicazione attraverso le nozioni di *dominio* e di *stile*. Si tratta di due nozioni equivalenti rispettivamente a quelle di *famiglie* e di *tipi* di socket.

<u>Famiglie di socket</u>

Il *dominio* di un socket equivale alla scelta di una famiglia di protocolli.

Ogni dominio è individuato univocamente da un numero intero non negativo, a cui corrispondono una o più costanti simboliche del tipo *PF_nomefamiglia*.

In altri termini si può affermare che esistono varie famiglie di socket, dette appunto *domini*; ogni famiglia riunisce i socket che utilizzano gli stessi protocolli sottostanti, detti *Protocol Family (PF)*.

Tra i domini più importanti si ricordano:

- ✓ *PF_LOCAL* (o *PF_UNIX* o *PF_FILE*), per le comunicazioni in locale tramite file system (reale o virtuale);
- ✓ *PF_INET*, la famiglia TCP/IP con Ipv4 (Internet Protocol Version 4);
- ✓ *PF_INET6*, la famiglia TCP/IP con Ipv6 (Internet Protocol Version 6)**;**
- ✓ *PF_IPX*, famiglia di Protocolli per reti Novell;
- ✓ etc…

A ciascun dominio possono corrispondere in teoria uno o più schemi di indirizzamento, ossia tipi di indirizzi. Per questo motivo i socket prevedono la nozione di *famiglia di indirizzi (o AF≡ Address Family)*, ciascuna univocamente individuata attraverso un numero non negativo,cui corrisponde una costante simbolica del tipo *AF_nomefamiglia*.

I domini finora introdotti dispongono di un unico schema di indirizzi, per cui le attuali implementazioni prevedono l'equivalenza tra nomi di dominio e nomi di indirizzi.

Tra tutte le famiglie di socket citate precedentemente (in relazione all'equivalenza tra nomi di dominio e di indirizzi) le più importanti sono:

- *Internet socket* (*AF_INET*): è quella che verrà utilizzata durante il corso; si tratta della famiglia che consente il trasferimento dati tra processi posti su macchine remote e connesse tramite una LAN o più semplicemente Internet.

- *Unix Domain socket* (*AF_UNIX*): questa consente il trasferimento dati tra processi sulla stessa macchina UNIX.

Un socket nei domini *AF_UNIX* e *AF_INET* presenta un'unica sostanziale differenza: l'indirizzo. In dettaglio:

- ✓ *AF_UNIX* corrisponde al *pathname* valido nel file system della macchina;

[Un *pathname*, in informatica, indica la posizione specifica di un elemento (file o cartella) all'interno di un archivio dati, con un certo file system. Il percorso consiste dunque in una stringa di caratteri che elenca, in modo ordinato, i diversi nodi del file system da visitare per raggiungere l'elemento in questione **]**

- ✓ *AF_INET* è specificato da due valori:
  - ✓ *indirizzo IP* (32 bit o 4 byte), che individua un unico host Internet;
  - ✓ *numero di porta* (16 bit o 2 byte), che specifica una particolare porta dell'host

- Esempio di Socket -

E' l'ambito di competenza della famiglia *AF_INET* in linguaggio *C*

```
struct sockaddr_in  {
        short int sin_family;          /* famiglia              */
        struct in_addr sin_addr;     /* indirizzo internet  */
        insigne short int sin_port;   /* numero di porta   */
}
```

La relativa istanza è

```
sin_family:   AF_INET;
sin_addr:   127.0.0.1;
sin_port:  6916;
```

NOTA BENE:

Nel linguaggio Java non serve specificare la famiglia di appartenenza del socket, poiché di default viene utilizzato il dominio *AF_INET*. Si tratta della famiglia che utilizza indirizzi composti da 4 ottetti (32 bit).

Tipi di socket

Come già è stato detto i socket permettono di specificare il tipo di comunicazione attraverso le nozioni di *dominio* e di *stile*.

Lo *stile* di comunicazione definisce il tipo di canale logico instaurato per la comunicazione. In altre parole lo *stile* fissa le caratteristiche della trasmissione.

Le trasmissioni possono ad esempio avvenire a flusso o pacchetti, essere affidabili o non affidabili, richiedere o meno la negoziazione di una connessione, etc …

Lo *stile* di comunicazione è individuato da costanti simboliche del tipo *SOCK_nomestile*.

Gli stili individuano i *tipi* di socket. I *tipi* principali sono tre:

- ✓ *SOCK_STREAM* o *Stream Socket* , corrispondente ad un canale di trasmissione a flusso bidirezionale, con connessione sequenziale ed affidabile;
- ✓ *SOCK_DGRAM* o *Datagram Socket* , che consiste in trasmissione a pacchetti (datagram) di lunghezza massima prefissata, senza connessione e non affidabile;
- ✓ *SOCK_RAW* o *Raw Socket* , per l'accesso a basso livello ai protocolli di rete e alle varie interfacce.

Ai fini del corso verranno trattati solo i primi due tipi di socket (Stream e Datagram).

Assegnato un dominio, la scelta dello stile di comunicazione corrisponde in pratica ad individuare uno specifico protocollo tra quelli appartenenti al dominio.

Stream socket

Attraverso questo primo tipo di socket si attua una connessione sequenziale, quasi sempre asimmetrica, affidabile e full-duplex (quindi "*collision free*") basata su stream (flusso) di byte di lunghezza variabile.

Operativamente ogni processo crea il proprio *endpoint* attraverso la chiamata della primitiva *socket()* in *C* oppure creando l'oggetto *socket* in *Java*.

> **[** Un *endpoint* è un tipo di nodo per la comunicazione in rete. È un'interfaccia esposta tramite una parte comunicativa o tramite una canale di comunicazione. Un esempio dell'ultimo tipo di una comunicazione endpoint è un argomento *Publish/Subscribe*. In informatica, l'espressione *Publish/Subscribe* si riferisce a un stile architetturale utilizzato per la comunicazione asincrona fra diversi processi, oggetti o altri agenti. Per la sua natura di integrazione tra diverse sorgenti software, il pattern publish/subscribe può essere considerato un middleware **]**

Una volta creato l'*endpoint* accade che:

- il *server* si mette in ascolto della rete, in attesa di una richiesta di connessione. Una volta che la richiesta è arrivata, il server la esaudisce utilizzando la primitiva *accept()*, creando un nuovo *socket* dedicato alla conessione;
- il *client* si mette in coda sul *socket* precedentemente generato dal *server* in attesa che la sua richiesta di connessione sia processata dal *server*. Una

volta "accettato" dal *server*, il *client* crea implicitamente il *binding* con la porta locale.

[La primitiva *bind*, si ricorda, è utilizzata per assegnare un indirizzo locale al *socket* individuato nel *socket descriptor* (*sd*) generato dalla precedente chiamata a *socket***]**

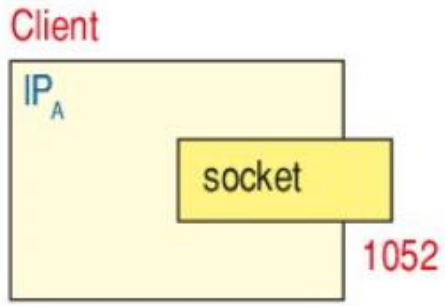
NOTA BENE:

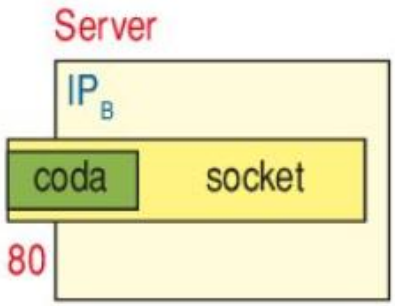
Nella comunicazione tipo *server-client* instaurata attraverso i *socket* è il *server* ad avere il maggior controllo, dato che è corrisponde allo stesso processo iniziale che ha generato il *socket*. Più *client* possono comunicare attraverso lo stesso *socket*, ma soltanto un solo *server* può essere associato in modo specifico ad un ben preciso *socket*.

Un'altra sostanziale differenza tra *client* e *server* risiede nel fatto che il *client* deve necessariamente conoscere l'indirizzo del *server*, mentre quest'ultimo acquisisce le informazioni del *client* soltanto dopo che la connessione è stata stabilita.

Di seguito viene illustrata graficamente la sequenza di passaggi necessari alla generazione di una connessione tra *server* e *client* .

- inizializzazione dei processi -



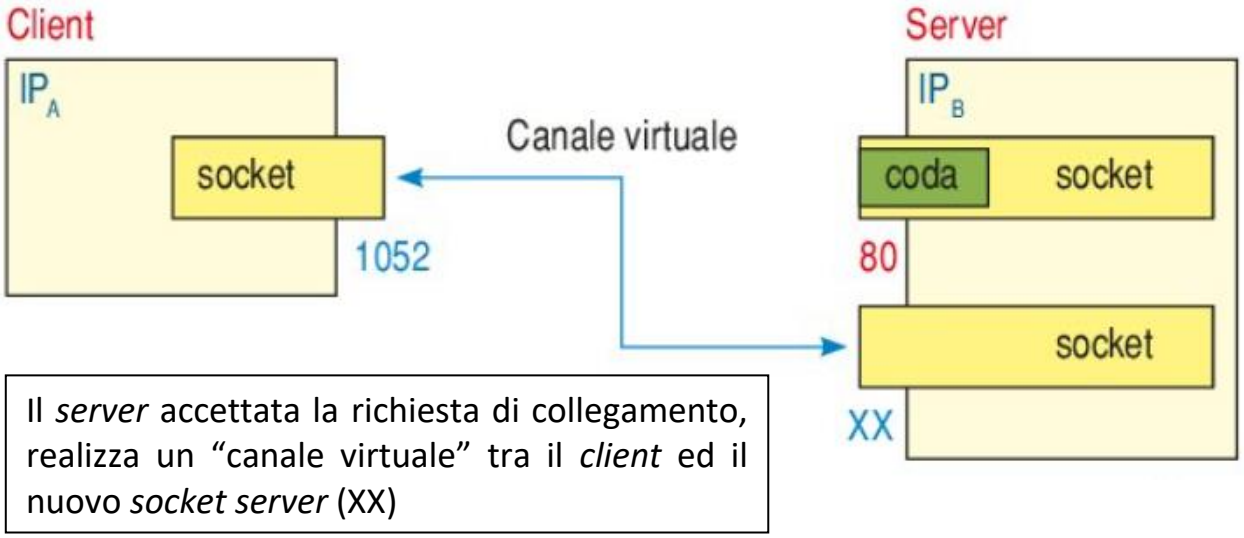


Il *client* crea un *socket*

Il *server* crea un *socket*, collegandolo alla porta 80 e ponendosi in ascolto

- il *client* richiede il collegamento -

- il *server* accetta la richiesta di collegamento -



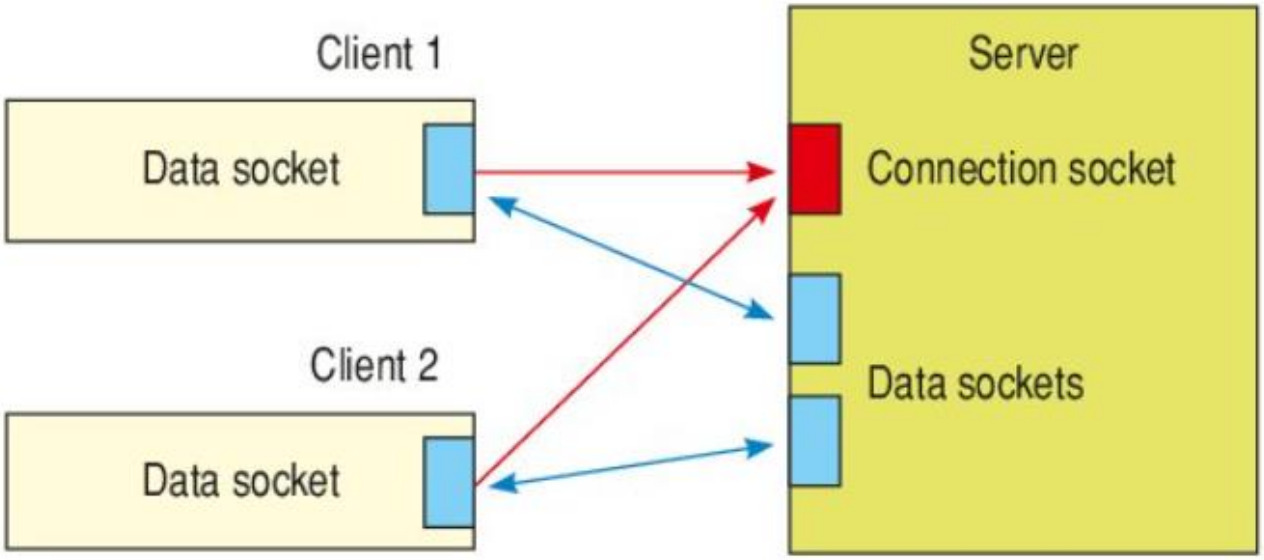
Nel momento in cui il *server* stabilisce il canale virtuale tra il *client* ed il nuovo *socket*, libera il *socket* originario (quello con la coda) in modo da poter accogliere e processare altre richieste di connessione. Tali richieste possono provenire dallo stesso *client* oppure da *client* diversi.

Stabilito il collegamento i due processi (*client* e *server*) iniziano a scambiarsi dati attraverso le funzioni *read()* e *write()*. Lo scambio dati termina con la chiusura del canale. La chiusura ha luogo quando entrambi gli attori della comunicazione effettuano la chiamata della primitiva *close()*.

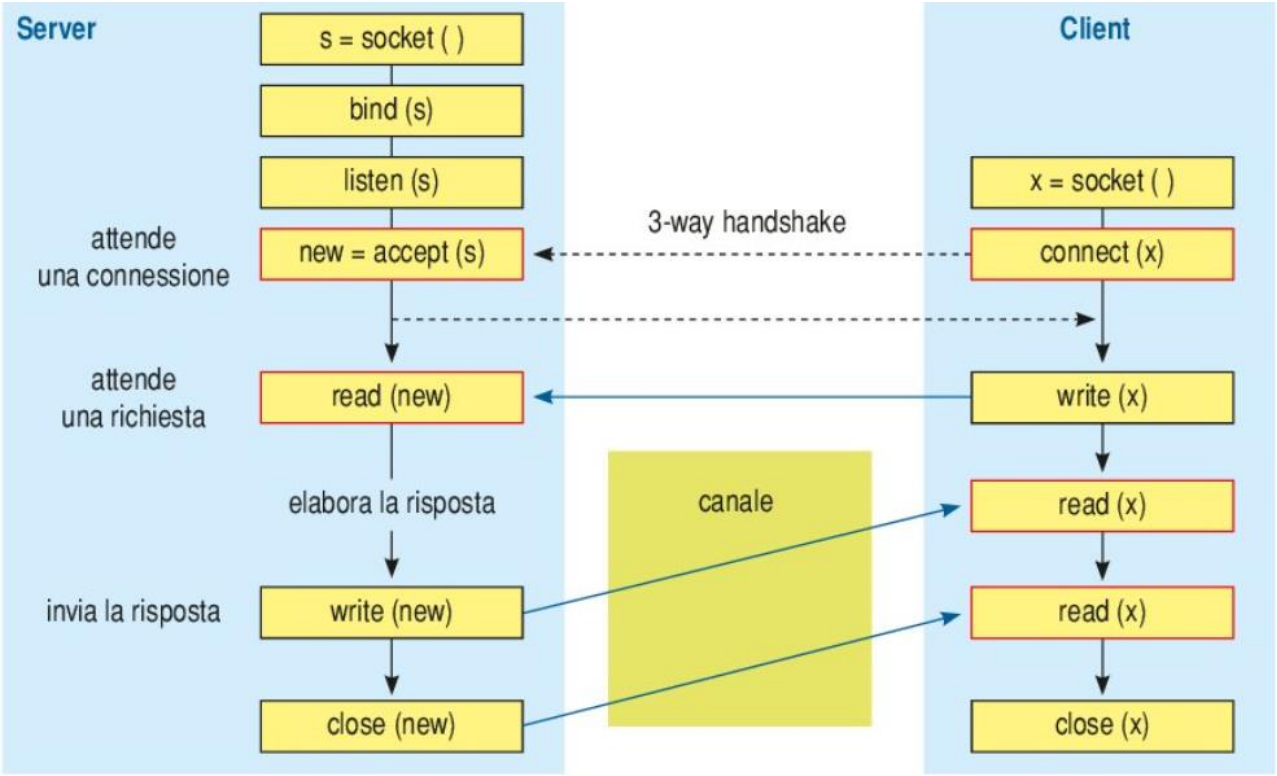
E' importante sottolineare e ricordare che il protocollo ***TCP*** si basa su questo tipo di *socket*.

Un *server TCP* possiede due tipi di *socket* :

- ✓ un primo tipo condiviso da più *client*, generato per accettare le richieste di connessione, detto *connection socket*. Questo particolare *socket* presenta un buffer per l'accodamento delle richieste di connessione da parte dei vari *client*.
- ✓ un secondo tipo dedicato ad un solo *client*, detto *data socket*. Questi serve per le operazioni *send()* e *receive()*. Si tratta di un tipo di *socket* non condiviso. Ne viene generato uno per ciascuna richiesta di connessione "accettata" dal *server*.



Di seguito lo schema logico completo della comunicazione *TCP client-server* attraverso *socket* nel linguaggio *C* :



NOTA BENE:

Il linguaggio *Java* genera delle *API* che differenziano i due precedenti tipi di socket TCP. Conseguentemente (ai fini del nostro corso) vengono utilizzate due *entry* diverse:

- ServerSocket(*int* port): utile per creare una *conection socket* relazionata alla porta specificata per accettare connessioni lato server;
- Socket(*InetAddress* address, *int* port): utile per creare una *data socket*. Questa seconda *entry* chiede come parametro una *server socket*, ovvero una connessione esistente (individuata tramite indirizzo IP e nimero di porta logica)

Datagram socket

Attraverso il secondo tipo di *socket*, ovvero i *datagram socket* (*SOCK_DGRAM*) si realizza una comunicazione capace di scambiare dati senza connessione. I messaggi contengono gli indirizzi di destinazione e di provenienza. I datagrammi trasferiscono messaggi di dimensione variabile, senza garanzia alcuna su ordine o perfino arrivo (ovvero consegna al destinatario) dei pacchetti, poiché si tratta di una comunicazione inaffidabile.

Le caratteristiche basilari precedentemente descritte permettono di inviare da un *socket* a più destinazioni e ricevere su un *socket* da più sorgenti.

Tipicamente i *datagram socket* realizzano il modello di elaborazione parallela "da molti a molti" e sono supportate nel dominio Internet dal protocollo *UDP*.

A livello operativo, ciascun processo genera il rispettivo *endpoint* richiamando la primitiva che crea il *socket* . A questa fase iniziale seguono due ulteriori step:

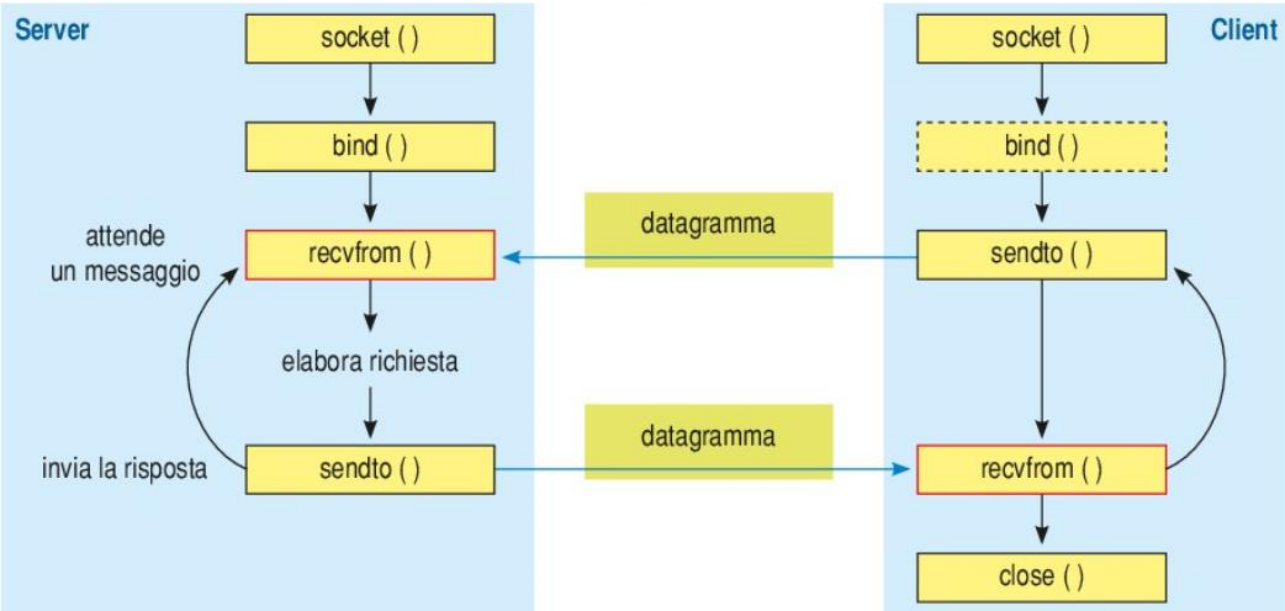
1. il *server* si mette in attesa di ricevere dati utilizzando la primitiva *receive()* in *Java* e *rcvfrom()* in *C*. Una volta ricevuti i dati il *server* genera una risposta con la primitiva *send()* in *Java* e *sendto()* in *C*.
2. il *client* invia il pacchetto dati al *server* con *send()* (*Java*) oppure *sendto()* (*C*). Si mette poi in attesa di una risposta utilizzando le stesse primitive viste per il *server* (*receive()* e *rcvfrom()*).

Una volta che la comunicazione ha termine il *socket* viene chiuso.

Il principale vantaggio è quello di trasferire i dati molto velocemente. Inoltre non ci sono differenze tra le chiamate effettuate dai vari processi coinvolti nella comunicazione. Quest'ultimo aspetto sottolinea come nel caso di *datagram socket* "decade" il concetto di *client* e *server*: entrambi utilizzano le stesse primitive e

soltanto il loro ordine di chiamata stabilisce in seno all'applicazione quale processo svolga il ruolo di *server* e quale quello di *client*.

Di seguito viene riportato lo schema logico completo di una comunicazione *UDP* in linguaggio *C* e con *socket* :



3) Trasmissione multi cast

Con il termine *multicast*, nelle reti di calcolatori, si indica la distribuzione simultanea di informazione verso un gruppo di destinatari, cioè la possibilità di trasmettere la medesima informazione a più dispositivi finali, senza dover indirizzare questi ultimi singolarmente e senza avere, quindi, la necessità di duplicare per ciascuno di essi l'informazione da diffondere.

In alternativa alla modalità di trasmissione *multicast* si hanno le modalità:

- *Broadcast*: invio a tutti gli host di una rete; c'è un'associazione uno-a-molti tra indirizzi di rete e ricevitori: ogni indirizzo di rete identifica un insieme di ricevitori, ai quali tutte le informazioni sono riportate.

- *Anycast*: invio ad un host qualunque di un gruppo, normalmente il più vicino. Un indirizzo di tipo *anycast* è un indirizzo IP che può corrispondere a più di un host sulla rete. Quando un pacchetto viene inviato ad un indirizzo *anycast*, la rete lo consegnerà ad *uno qualsiasi* tra quelli associati all'indirizzo, tipicamente al più "vicino" o al "migliore" (ossia quello con situazione del traffico migliore). In *anycast*, c'è anche un'associazione uno-a-molti tra indirizzi di rete e ricevitori: ogni indirizzo destinatario identifica un insieme di ricevitori, ma soltanto uno tra questi ricevitori è scelto per ricevere i dati da una qualsiasi delle sorgenti.

- *Unicast*: invio ad un solo e determinato host; c'è un'associazione uno-a-uno tra gli indirizzi di rete e ricevitori: ogni indirizzo destinazione identifica univocamente un ricevitore.

Tipiche applicazioni *multi cast* sono ad esempio:

- *usenet news*: pubblicazione e diffusione di nuove notizie in tempo reale;
- *videoconferenze*: ciascun host genera un flusso audio-video che viene ricevuto dagli host associati ai partecipanti alla videoconferenza;
- *massive multiplayer games*: giochi di ruolo e Internet games dove un elevato numero di giocatori interagiscono in un mondo virtuale;
- *DNS (Domain Name System)*: aggiornamenti delle tabelle di naming inviati a gruppi di DNS;
- altre applicazioni come chat, instant messaging, ecc …

[In informatica (e tlc) il ***sistema dei nomi di dominio*** (o ***Domain Name System, DNS***), è un sistema utilizzato per la risoluzione di nomi dei nodi della rete (gli *host*) in indirizzi IP. Il servizio è realizzato tramite un database distribuito, costituito dai server DNS. Il DNS ha una struttura gerarchica ed è diviso in domini (com, org, it, ecc...). Ad ogni dominio o nodo corrisponde un *nameserver*, che conserva un database con le informazioni di alcuni domini di cui è responsabile e si rivolge ai nodi successivi quando deve trovare informazioni che appartengono ad altri domini. **]**

L'implementazione di un sistema *multicast* richiede la definizione di uno schema di indirizzamento dei gruppi e di un supporto che memorizzi la corrispondenza tra un gruppo ed i partecipanti.

L'implementazione necessita in ultima istanza anche di un meccanismo di ottimizzazione dell'utilizzo della rete nel caso di invio di pacchetti ad un gruppo: si tratta dell'utilizzo dei cosiddetti *M-router* (*Multicast Router*).

Gli *M-Router* sono utilizzati in reti di notevoli dimensioni come le *WAN* (*WIDE AREA NETWORK* : è una tipologia di rete di computer che si contraddistingue per avere un'estensione territoriale pari a una o più regioni geografiche).

In una WAN l'host sorgente invia una sola copia dell'informazione (indipendentemente dal numero di destinatari), ma qui particolari protocolli di routing devono individuare dove sono localizzati tutti gli host che vogliono ricevere quella trasmissione e trasmettere il flusso solo sulle reti interessate. Saranno gli M-Router (Multicast Router) che, utilizzando protocolli multicast, moltiplicheranno l'informazione quando necessario sulle n reti interessate.

Il protocollo più utilizzato per il *multicast* su Internet è l'*IGMP* (*Internet Group Management Protocol*). Tale protocollo garantisce la trasmissione tra host e *M-Router* ad essi collegati direttamente. La trasmissione riguarda messaggi relativi alla costituzione dei gruppi interessati al *multicasting* : *IGMP* viene usato dai computer per richiedere di essere iscritti ad un gruppo multi cast. *IGMP* , ad esempio, fornisce ad un host i mezzi per informare l' M-router più vicino che un'applicazione vorrebbe unirsi ad un ben preciso gruppo *multicast* utilizzando normali datagrammi IP.

Le *API multicast* , in virtù delle citate caratteristiche del *multicasting*, devono quindi contenere primitive per:

- ✓ *unirsi* ad un gruppo di *multicast* (*join*): ciascun gruppo va identificato ed indirizzato univocamente;
- ✓ *lasciare* un gruppo di *multicast* (*leave*);
- ✓ *spedire* messaggi ad un gruppo, ossia a tutti i processi che ad un determinato momento fanno parte di quel gruppo;
- ✓ *ricevere* messaggi indirizzati ad un gruppo di cui l'host fa parte.

I gruppi di *multicasting* sono gestiti in modo dinamico poiché:

- un host può abbandonare o unirsi ad un gruppo in qualsiasi momento o appartenre contemporaneamente a più gruppi;
- per poter inviare messaggi ad un gruppo non è richiesto appartenervi;
- i membri di un gruppo possono appartenere alla stessa rete oppure a reti fisiche differenti.

Per l'indirizzamento viene utilizzato un indirizzo *IP* di classe *D*, del tipo:

1110 XXXX	.....

con intervallo tra 224.0.0.0 e 239.255.255.255 .

Gli indirizzi possono essere:

- *permanenti* : l'indirizzo di *multicast* viene assegnato da IANA (Internet Assigned Numbers Authority) ad un bem preciso gruppo al quale rimane associato anche se non ci fossero più partecipanti connessi (sono i *Well-Known Addresses*);

- *temporanei* : questo secondo tipo di indirizzi necessita di definire un opportuno protocollo onde evitare conflitti nell'attribuzione degli indirizzi di quei gruppi che esistono fino alla condizione minima di esistenza di un solo partecipante.

Una proprietà basilare del *multicast* è quella che gli *M-router* mantengono la corrispondenza tra l'indirizzo di *multicast* e gli indirizzi IP dei singoli host che partecipano al gruppo di *multicast*.

La comunicazione *multicast* utilizza il paradigma *connectionless*, poiché più connessioni dovrebbero essere gestite contemporaneamente a fronte di un carico di lavoro (in termini di controllo) troppo elevato se la rete fosse connection-oriented. La comunicazione non orientata alla connessione riduce notevolmente questo carico di lavoro. Per la natura del servizio di rete *multicast*, risulta molto difficile usare protocolli di trasporto orientati alla connessione come TCP, per cui si usano protocolli senza connessione come UDP.