Server Web e
Pattern MVC per il Web

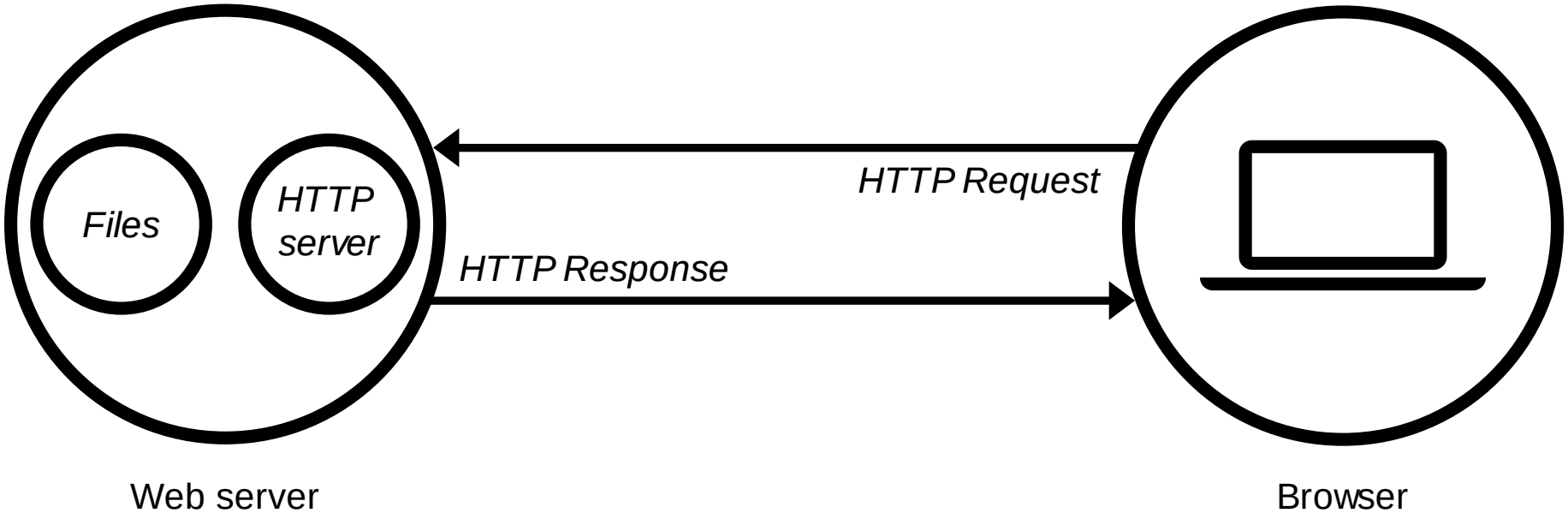
Server Web

Server Web

Quando parliamo di Server Web possiamo riferirci all'hardware oppure al software, oppure ancora ad entrambi (considerando il loro lavoro comune)

- **Hardware**: un server web è un computer che ospita il software server web e i file che compongono un sito Web (documenti HTML, immagini, Fogli di stile CSS, script Javascript, ecc). E' collegato alla rete Internet e supporta l'interscambio di dati con altri device collegati al Web
- **Software**: un server web include diverse parti/componenti che controllano come l'utente accede ai file del sito web, che si occupano della comunicazione HTTP (server HTTP). Un server HTTP è in grado di comprendere gli URL e di risolvere i path corrispondenti ed è in grado di comprendere il protocollo di comunicazione HTTP

Server Web


Files
HTTP server
HTTP Request
HTTP Response
Web server
Browser

Server Web: principali funzionalità

- Un server web deve quindi ospitare, processare, effettuare il delivery delle pagine Web ai browser
- Le risorse sono documenti HTML, che possono includere immagini, fogli di stile, script

- Server HTTP: riceve le richieste HTTP da parte del client e manda le risposte HTTP al client, inviandogli le risorse richieste, in risposta alle richieste ricevute

Server Web

- Ogni volta che un browser richiede una risorsa ad un Web server, invia una Richiesta HTTP
- La richiesta arriva al web server (hardware)
- Il server HTTP (software) del web server accetta la richiesta
- Il server Web (software) si occupa di cercare la risorsa
- Se trova la risorsa richiesta allora invia una risposta HTTP al client, mandando la risorsa richiesta
- Altrimenti un messaggio errore ***404 Not found*** viene inviato e visualizzato dal browser

Server Web: statico/dinamico

- **Server Web Statico**: dotato di HTTP server, ospita le risorse del sito Web, che invia al client senza alcuna elaborazione.
- **Server Web Dinamico**: si tratta di un server web statico, a cui si aggiungono altre parti/componenti che consentono la computazione lato server. Queste componenti aggiuntive solitamente sono application server oppure database. Si chiama «dinamico» perché l'application server aggiorna/elabora i file prima di inviarli al client tramite il server HTTP (ad esempio, le pagine HTML potrebbero essere riempite di contenuti presi dal database tramite query, oppure potrebbe essere eseguito codice PHP)

Server Web: gli step principali

- Considerando le componenti e le funzionalità di base del server web, possiamo quindi individuare gli step principali:
  - Hosting dei file
  - Comunicazione con protocollo HTTP (server HTTP)
  - Traduzione dei path
  - Gestione dei contenuti statici e dinamici

Hosting dei file

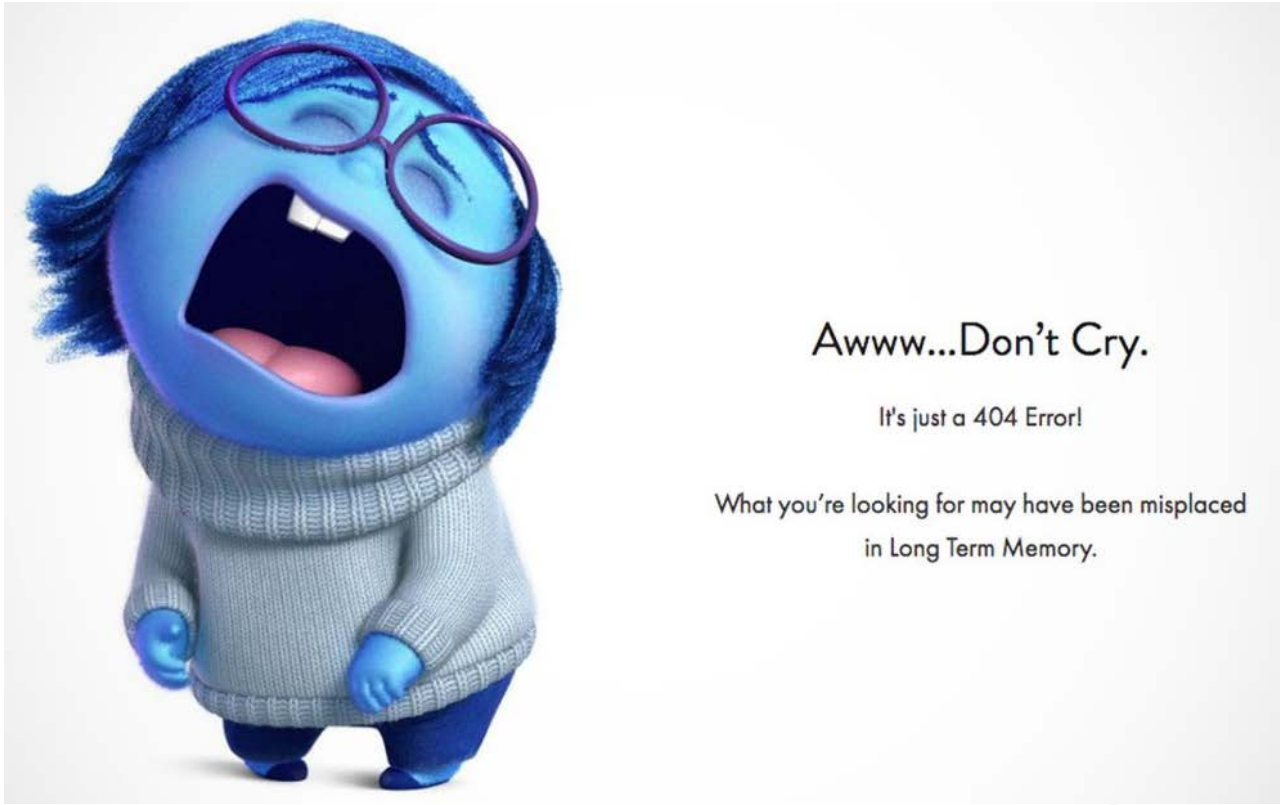
- Il server web deve ospitare i file del sito web (documenti HTML, immagini, fogli di stile CSS, script Javascript, font, video, ecc)
- Tecnicamente è possibile ospitarli in un personal computer, ma ovviamente non è efficace ed efficiente, è necessario quindi utilizzare un server web dedicato con le seguenti caratteristiche
  - Sia sempre acceso e funzionante
  - Sia sempre collegato ad Internet
  - Abbia sempre lo stesso IP address
  - In alcuni casi potrebbe essere utile la manutenzione da parte di un provider di terze parti (un buon hosting provider può rappresentare un elemento chiave)

Server HTTP

- Un server Web deve contenere un server HTTP
- Questo significa che è in grado di comunicare con i client condividendo il protocollo HTTP (HyperText Transfer Protocol)
- HTTP è:
  - ***Testuale***: tutti i comandi e messaggi sono plain-text e leggibili (da un utente umano)
  - ***Stateless***: server e client non hanno memoria delle eventuali precedenti comunicazioni occorse tra di loro. Infatti, basandosi sul solo HTTP un server non è in grado di ricordarsi della password immessa dall'utente oppure dal punto di una transazione in cui si trova l'utente. Per fare ciò è necessario un application server che sia in grado di occuparsi di questi task oppure di altri meccanismi specifici (ad esempio lato client)

Server HTTP

- Quando un client HTTP invia una richiesta al server HTTP, specifica l'URL della risorsa desiderata
- Il server HTTP ***deve*** rispondere ad ogni richiesta HTTP, inviando la risorsa richiesta oppure inviando un messaggio di errore (es. 404, nel caso in cui non si tratti di risorsa non trovata)



Server HTTP

- Il server HTTP è quindi incaricato di ricevere le richieste HTTP, di processarle e di inviare le risposte HTTP al client

1. Per ogni richiesta ricevuta, il server HTTP controlla che l'URL richiesto corrisponda effettivamente ad un file
2. Se il server HTTP trova il file, il server web invia la risorsa richiesta al server oppure un application server costruisce/elabora il file (creando/generando la risorsa)
3. Se il processo di creazione/elaborazione non è possibile oppure se la risorsa non esiste, allora il server web ritorna al browser un messaggio di errore

Path translation

- Per poter trovare la risorsa richiesta dal client HTTP è fondamentale mappare il percorso di un URL in
  - Una risorsa di un file system locale (per richieste di risorse statiche)
  - Un nome di un programma interno o esterno (per richieste di risorse dinamiche)
- Per una richiesta statica, il path URL specificato dal client è relativo alla root directory del server web

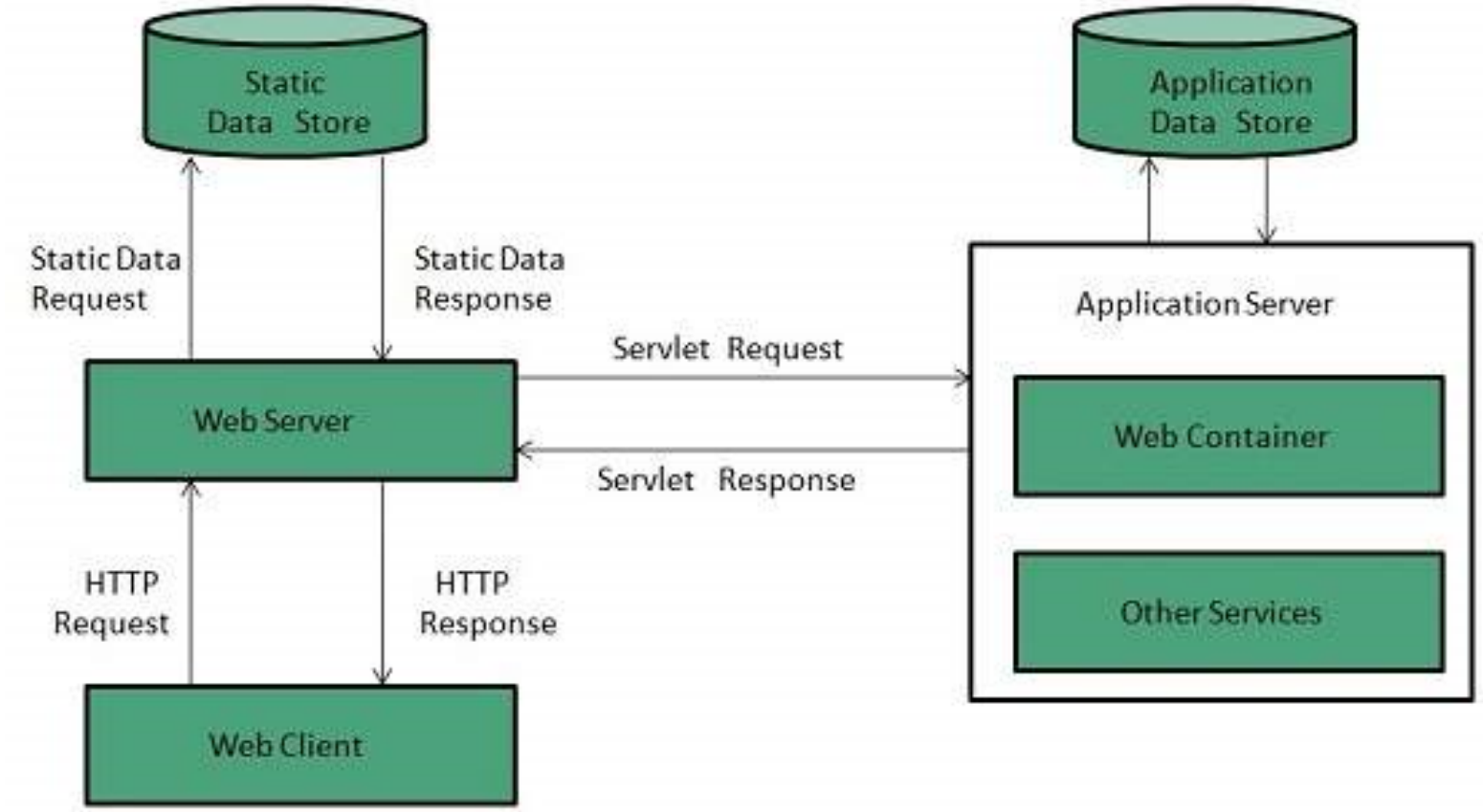
Path translation

- Esempio: http://www.example.com/path/file.html
  Il client traduce questo URL in una connessione a www.example.com con una richiesta GET della risorsa **`/path/file.html`**
- Il server web ospitato da www.example.com appenderà il path ricevuto con la richiesta alla sua root directory
  - su un server Apache solitamente è **`/home/www`**, e il risultato sarà: **`/home/www/path/file.html`**
  - Su una macchina Unix potrebbe essere **`/var/www/htdocs`** e il risultato sarà: **`/var/www/htdocs/path/file.html`**
- Il server web legge il file, se esiste, e manda questa risorsa al browser. La risposta descrive il contenuto del file e contiene il file stesso (oppure viene inviato un messaggio di errore che comunica che il file non esiste oppure che è non raggiungibile)

Contenuto statico e contenuto dinamico

- Un server Web potrebbe dover gestire sia contenuto statico sia contenuto dinamico
- Nel caso di contenuto statico, il server Web si occupa semplicemente di recuperare le risorse e di inviarle al client così come sono
- Nel caso di contenuto dinamico, il server Web (application server) processa il contenuto della risorsa oppure genera la risorsa “on-the-fly” ottenendo dati da un database. Questa soluzione è più flessibile, ma sicuramente più complessa
- Ci sono molti application server
- Alcuni offrono supporto e funzionalità a specifiche categorie di siti web, come ad esempio blog, wiki, e-shop. Altri Content Management System offrono funzionalità e supporti più generici

Architettura: un esempio



Architettura

- L'Architettura di un server web può seguire due approcci:
  - Approccio **concorrente**
  - Approccio Single-Process-Event-Driven
- L'approccio **concorrente** permette al server web di gestire richieste multiple da parte dei client allo stesso tempo, con uno dei seguenti metodi:
  - Multi-processing
  - Multi-threaded
  - Metodo ibrido

Architettura

- ***Multi-processing***: un processo padre inizia diversi processi figli e distribuisce le richieste in arrivo a questi processi figli. Ogni processo figlio è responsabile della gestione di una singola richiesta. Il processo padre è il responsabile del monitoraggio del carico e decide se i processi devono essere killed oppure forked
- ***Multi-threaded***: vengono creati più processi single-threaded
- ***Ibrido***: è una combinazione dei due metodi precedenti. In questo metodo, sono creati processi multipli e ogni processo inizia thread multipli. Ogni thread gestisce una connessione. Usando multi thread in single process si ottiene meno carico sulle risorse del sistema

Altri server

- ***Application Server***: offrono un ambiente per l'esecuzione delle applicazioni. Spesso agiscono come interfaccia ai database server, gestendo il flusso dei dati, così da inviarli all'utente
- ***Database Server***: gestiscono l'archiviazione e il recupero dei dati, fornendo i dati richiesti dagli altri server, sulla base delle informazioni che richiedono

Application server

- Gli application server si occupano di fornire la computazione relativa alla business logic delle applicazioni, così da generare il contenuto dinamico
- I client di un application server sono a loro volta applicazioni, e potrebbero essere dei server Web oppure altri application server
- La comunicazione tra un application server e i suoi client potrebbe avvenire via messaggi HTTP, ma non è mandatorio, altri protocolli di comunicazione potrebbero essere utilizzati (sulla base del tipo di application server)

Application server e server Web

- In molti casi application server e server Web sono implementati insieme
- Non esistono standard o recommendation che definiscono le proprietà dei server Web e degli application server
- Ad esempio, un reverse proxy potrebbe fornire anche le funzionalità di un load balancer, ma a loro volta la maggior parte dei load balancer sono anche reverse proxy

Altre funzionalità

- ***Logging***: solitamente i server web hanno anche la capacità di gestire log file con le informazioni relative alle richieste dei client e alla risposte del server. Questo permette di raccogliere dati statistici e di fare analisi dei log
- ***Authentication***: accesso tramite nome utente e password, proteggendo l'accesso ad alcuni tipi di risorse ospitate sul server
- ***Supporto a HTTPS (usando SSL o TLS)***: permette la connessione sicura (criptata) al server su porta 443 invece che sulla porta 80
- ***Compressione dei contenuti***: permette di ridurre la dimensione delle risposte (ad esempio usando **gzip**), limitando la larghezza di banda limitata dedicata alla comunicazione
- ***Virtual hosting***: per poter gestire più siti Web usando un solo IP address
- ***Supporto a file di grandi dimensioni***: ad esempio per gestire file con dimensione maggiore di 2 GB su sistemi operativi a 32 bit
- ***Bandwidth throttling***: per limitare la velocità delle risposte, così da non saturare la rete e da poter gestire le richieste di più client

Limiti di carico

- I server Web hanno limiti di carico definiti, dato che possono gestire un numero limitato di connessioni client concorrenti per IP address (e porta IP) e possono servire solo un certo numero massimo di richieste per secondo, sulla base di:
  - Setting del server
  - Tipo di richieste HTTP
  - Tipo del contenuto (statico o dinamico)
  - Presenza e utilizzo di una cache per il contenuto
  - Limiti di hardware, software e Sistema operativo su cui il server sta girando
- Se il server web si avvicina o raggiunge i suoi limiti di carico, allora diventa sovraccarico e quindi non risponde in modo opportuno

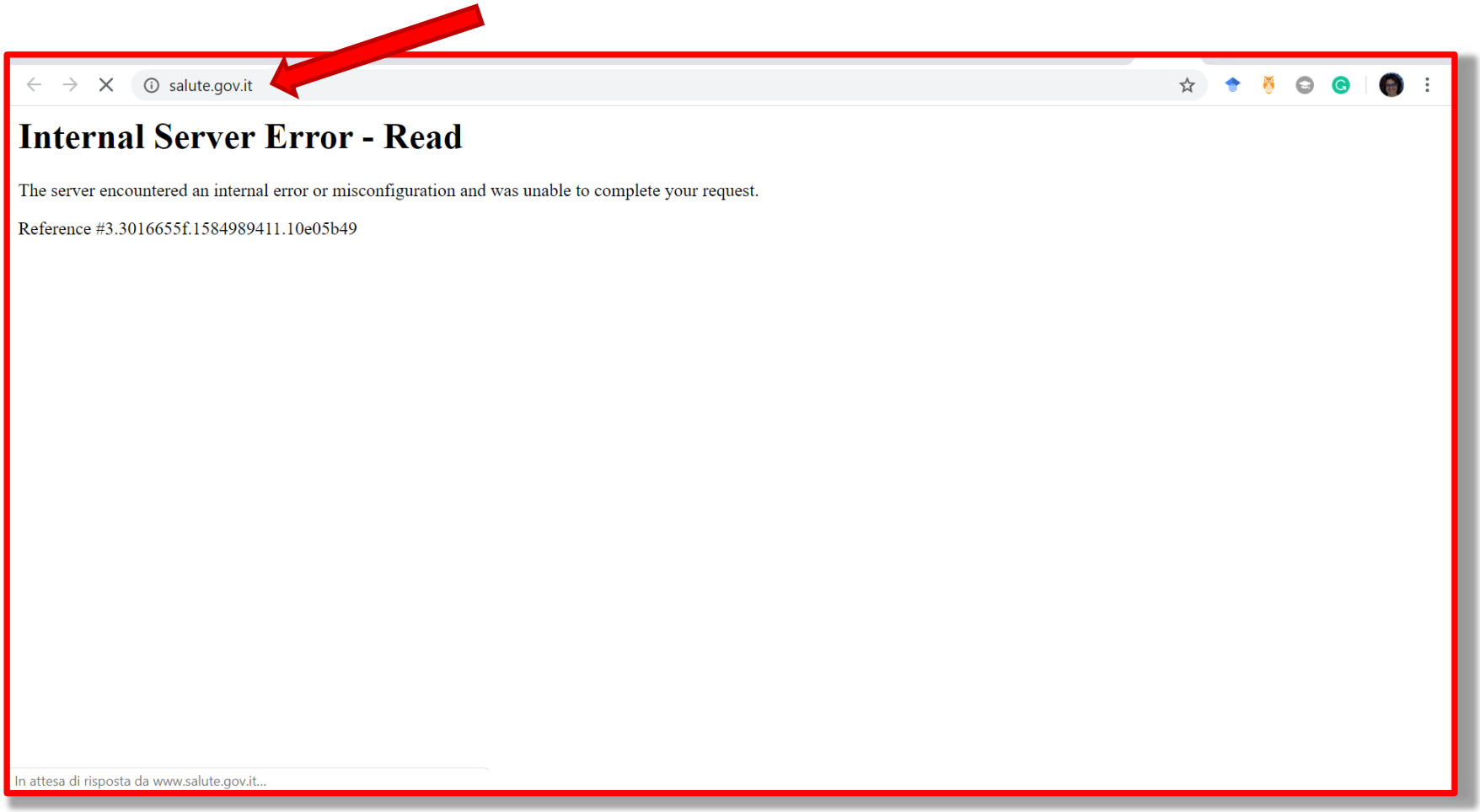
Cause dell'overload

- **Eccessivo traffico legittimo** (migliaia o anche milioni di client richiedono pagine del sito in un intervallo di tempo molto breve)
- **Attacchi DDoS** (Distributed Denial of Server)
- **Worm** che possono causare un traffico anormale, a causa di numerosi computer infetti (senza alcun coordinamento tra loro)
- **Virus XSS** (dovuti a controlli dell'input nei form insufficienti o assenti) che possono causare un traffico elevato a causa di numerosi browser e/o server web infetti
- **Traffico generato da robot** non filtrati o limitati su siti web con poche risorse (ad esempio larghezza di banda)
- **Rallentamenti nella rete Internet**, che causano ritardi o lentezza nella gestione delle richieste dei client. In questo modo il numero di connessioni aumenta a tal punto che si raggiungono i limiti del server
- **Non disponibilità** (parziale o totale) del server web, dovuta a manutenzione o aggiornamento, failure dell'hardware o del software, failure del backend (ad esempio del database); in questi casi i server web restanti hanno troppo traffico e diventano sovraccarichi

Sintomi e conseguenze dell'overload

- Le richieste sono gestite con ritardi (che possono variare tra 1 secondo fino a qualche centinaio di secondi)
- Errori 500 (Internal Server Error), 502 (Bad Gateway), 503 (Service Unavailable), 504 (Gateway Timeout) vengono restituiti al client. In alcuni casi potrebbero essere restituiti anche errori non correlati, come ad esempio 404 (Not found) o 408 (Request Timeout)
- Le connessioni TCP sono rifiutate o interrotte prima che il contenuto sia inviato ai client
- In alcuni (rari) casi, solo parte del contenuto potrebbe essere inviata al client (potrebbe dipendere da risorse del sistema non disponibili)

Overload: un esempio



Data: lunedì 23 Marzo 2020, ore 19:55

Tecniche anti-overload

- Gestire il traffico di rete, usando
  - Firewall per bloccare traffico non desiderato
  - Manager di traffico HTTP
  - Manager di larghezza di banda e shaping del traffico, per limitare i picchi nell'uso della rete
- Usare Domain name diversi per gestire differenti contenuti attraverso diversi server web (es: http://images.example.com e http://www.example.com)
- Usare Domain name diversi o diversi computer per separare i file di grandi dimensioni da quelli di piccolo/medie dimensioni; l'idea è quella di usare una cache per i file di piccolo e medie dimensioni e per servire in modo più efficiente i file di grandi dimensioni (10 – 1000 MB), usando setting differenti

Tecniche anti-overload

- Applicare tecniche di web caching
- Usare molti server web (software) per computer, ognuno legato al proprio IP address
- Usare molti server web (hardware) raggruppati insieme, così che possano agire o essere visti come un solo server web
- Aggiungere più risorse hardware al server web (es: RAM, memoria su disco)
- Impostare i parametri del sistema operativo
- Utilizzare in modo più efficiente i programmi per server Web
- Usare workaround, in particolare se il server web si occupa di generare contenuto dinamico

Q&A

- Ci sono domande?

Pattern MVC nel Web

MVC

Model

- Gestisce il comportamento e i dati relativi al dominio
- Risponde alla ***view*** su interrogazioni sul proprio stato
- Risponde a richieste di cambiamento dello stato (dal ***controller***)

View

- Visualizza il ***modello*** in maniera utile per l'interazione utente
- Possono esistere ***view*** multiple per lo stesso modello

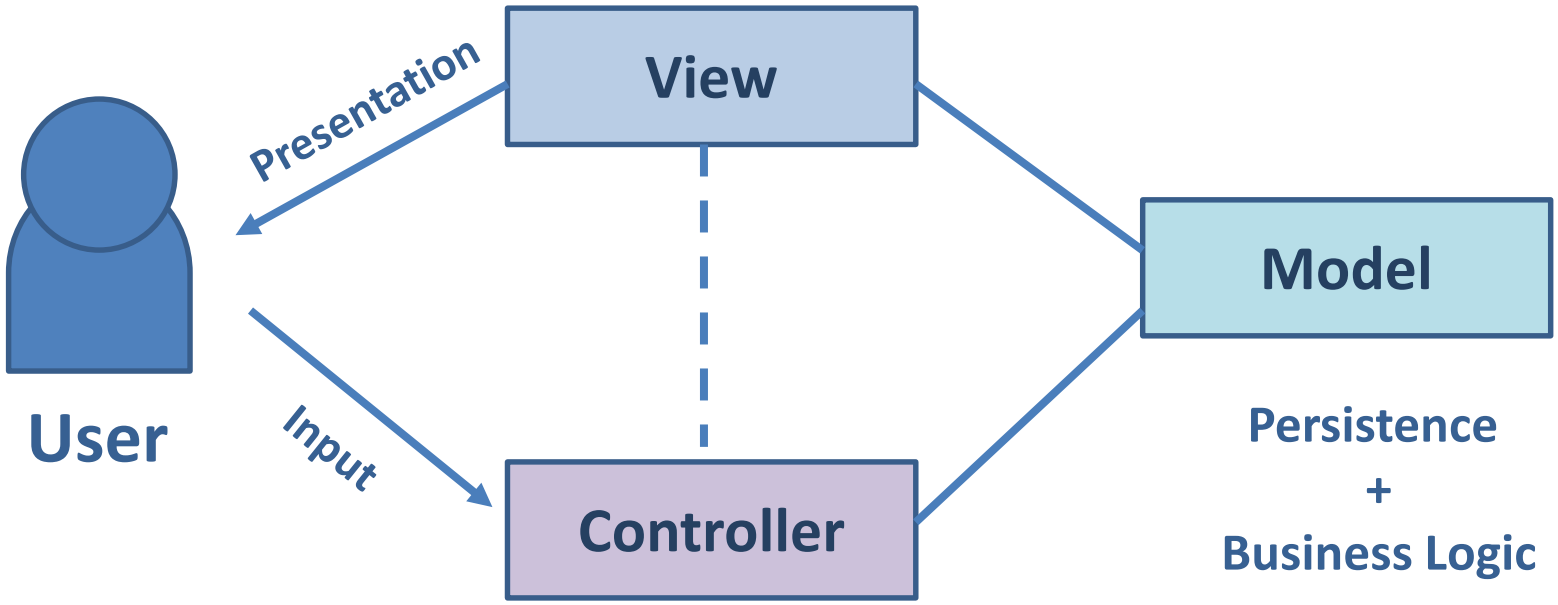
Controller

- Definisce il comportamento
- Mappa le azioni degli utenti agli aggiornamenti del ***model***
- Seleziona le ***view*** per l'interazione con l'utente

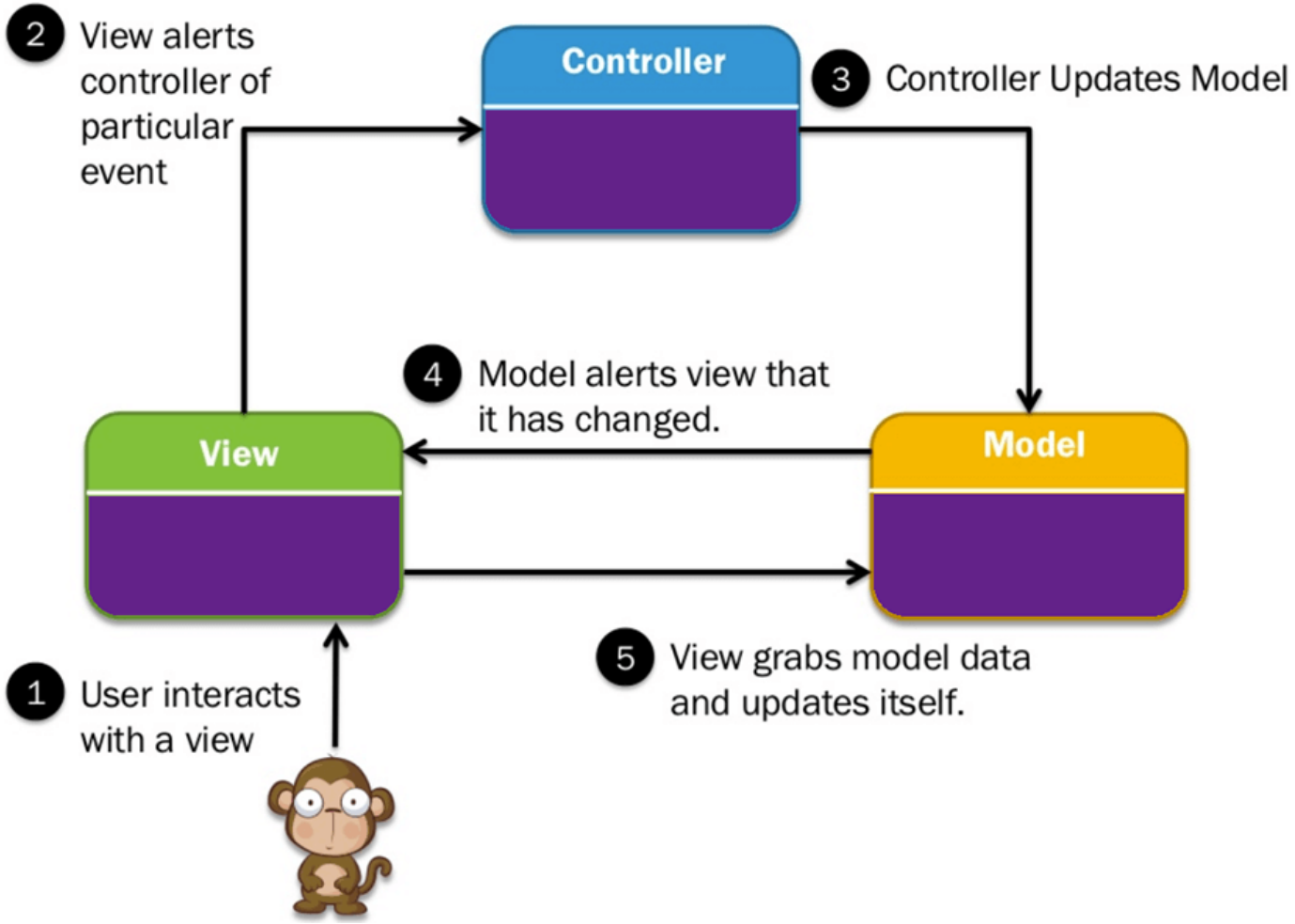
MVC: l'origine

- Il termine "Model-View-Controller" viene coniato nel 1978, da Trygve Reenskaugh e Adele Goldberg
- La loro idea era basata sul fatto che MVC e le sue varianti potessero costituire un Pattern Language, ovvero un linguaggio condiviso per la discussione, definizione, formalizzazione di problemi e delle loro soluzioni
- Il concetto di Pattern Language ha influenzato anche i concetti alla base del libro "Design Patterns"

MVC


View
Presentation
Model
User
Input
Controller
Persistence
+
Business Logic

MVC

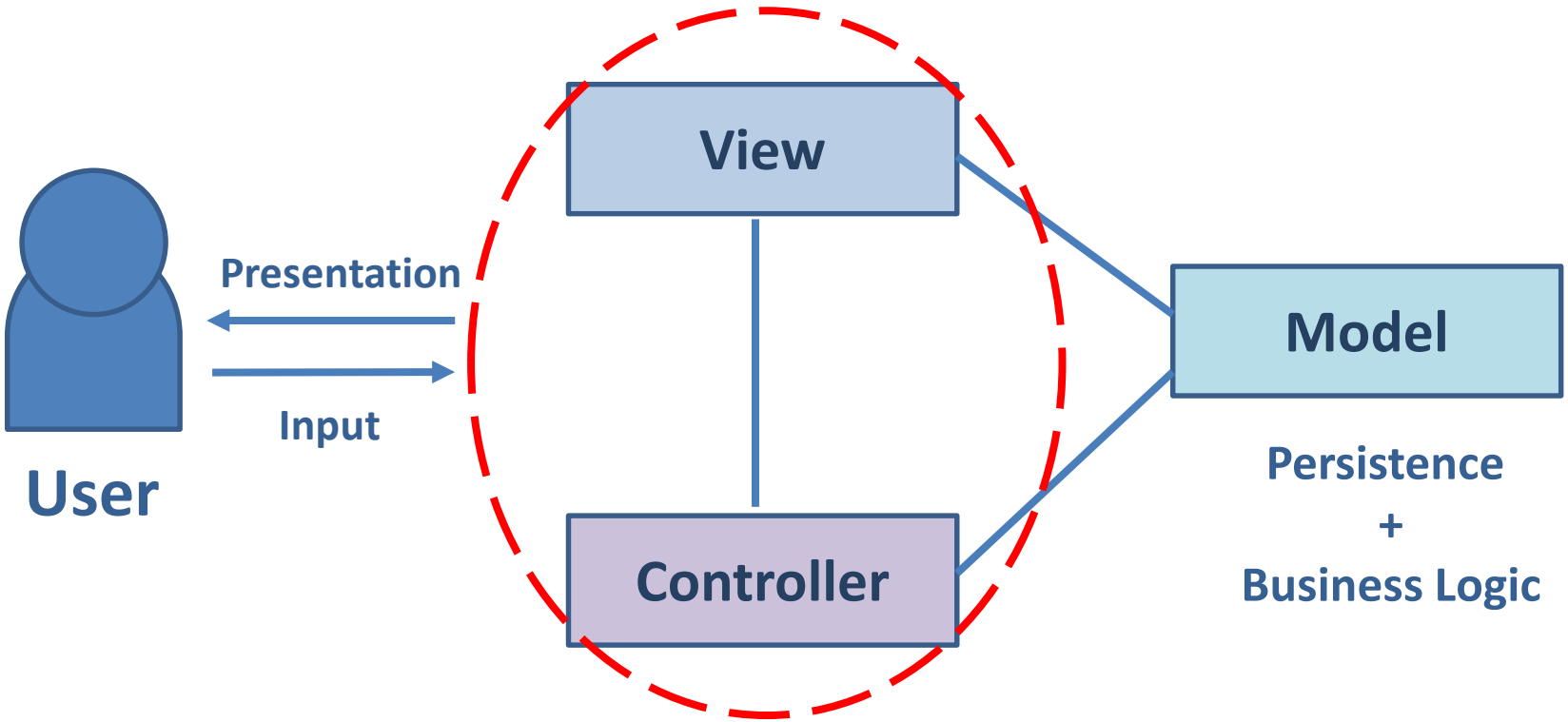


https://www.guru99.com/mvc-tutorial.html

MVC

- Il **model** è l'informazione che viene manipolata dall'applicazione, o anche solo dal widget
- La **view** è la visualizzazione del modello sullo schermo. Possono esistere view molteplici per ogni modello
- Il **controller** riceve gli input dall'utente e decide cosa significhino e cosa si debba fare
- Vanno quindi implementate separatamente le classi/elementi che implementano il **modello** funzionale dell'oggetto, da quelle che implementano la **view**. Il **controller** può essere associato in molti casi alla **view**

MVC: una variante

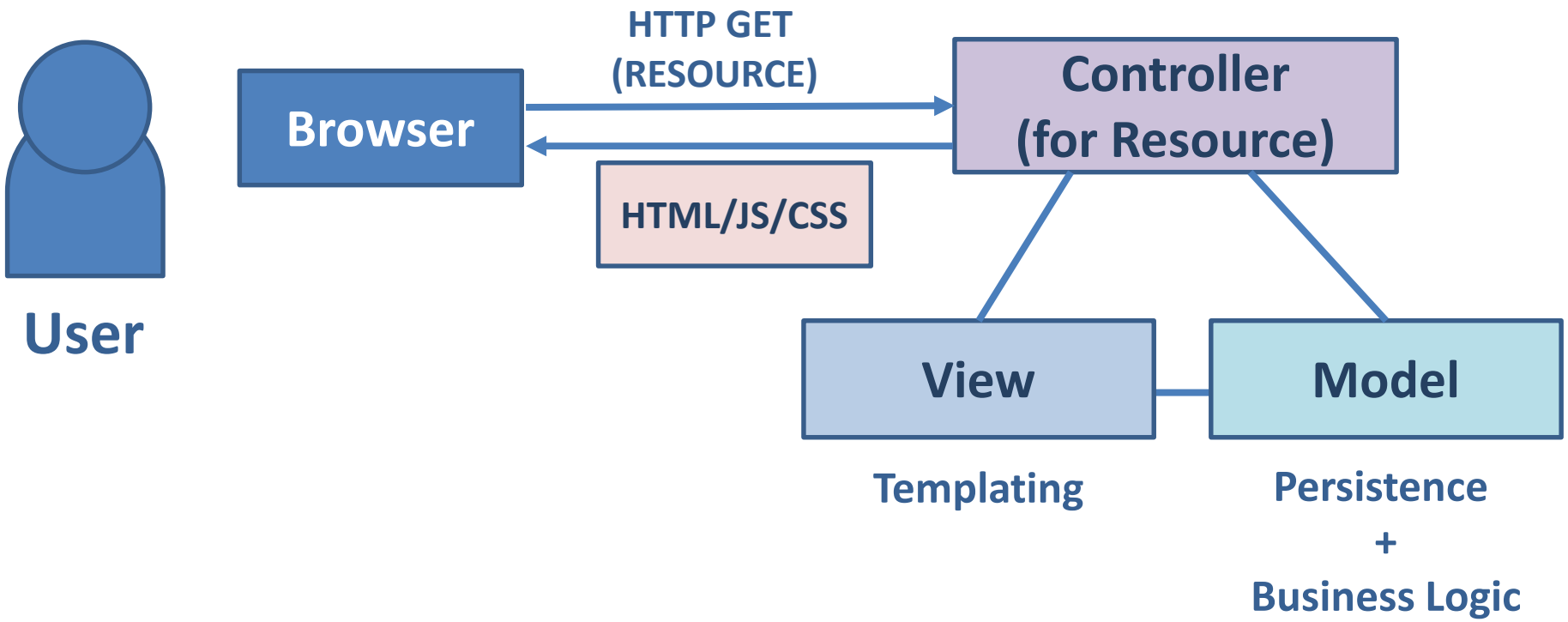


- I ruoli di View e Controller possono essere concentrati in un unico oggetto

MVC nel Web

- All'inizio degli anni 2000, diversi framework web hanno iniziato ad adottare MVC
- Ad esempio: Spring, Ruby on Rails, PHP, ASP.net, ecc
- In questi framework, il Controller si fa carico della gestione della Richiesta HTTP iniziale

MVC nel Web


HTTP GET
(RESOURCE)
Browser
Controller
(for Resource)
HTML/JS/CSS
User
View
Model
Templating
Persistence
+
Business Logic

MVC nel Web

- Il **Controller** funge da entrypoint dell'applicazione (al posto del **View**) e si occupa della comunicazione HTTP
- La **View** si occupa di assemblare bundle HTML, JS e CSS nella risposta che deve essere inviata al browser (gestisce il template)
- Il **Model** gestisce la business logic e lo storage
- HTML e JS contengono alcuni elementi della logica di interazione dell'applicazione, come ad esempio l'handler del click di un bottone che invia una azione al **Controller** tramite una **`XMLHttpRequest`**
- Il browser è esterno a questo approccio, non è coinvolto

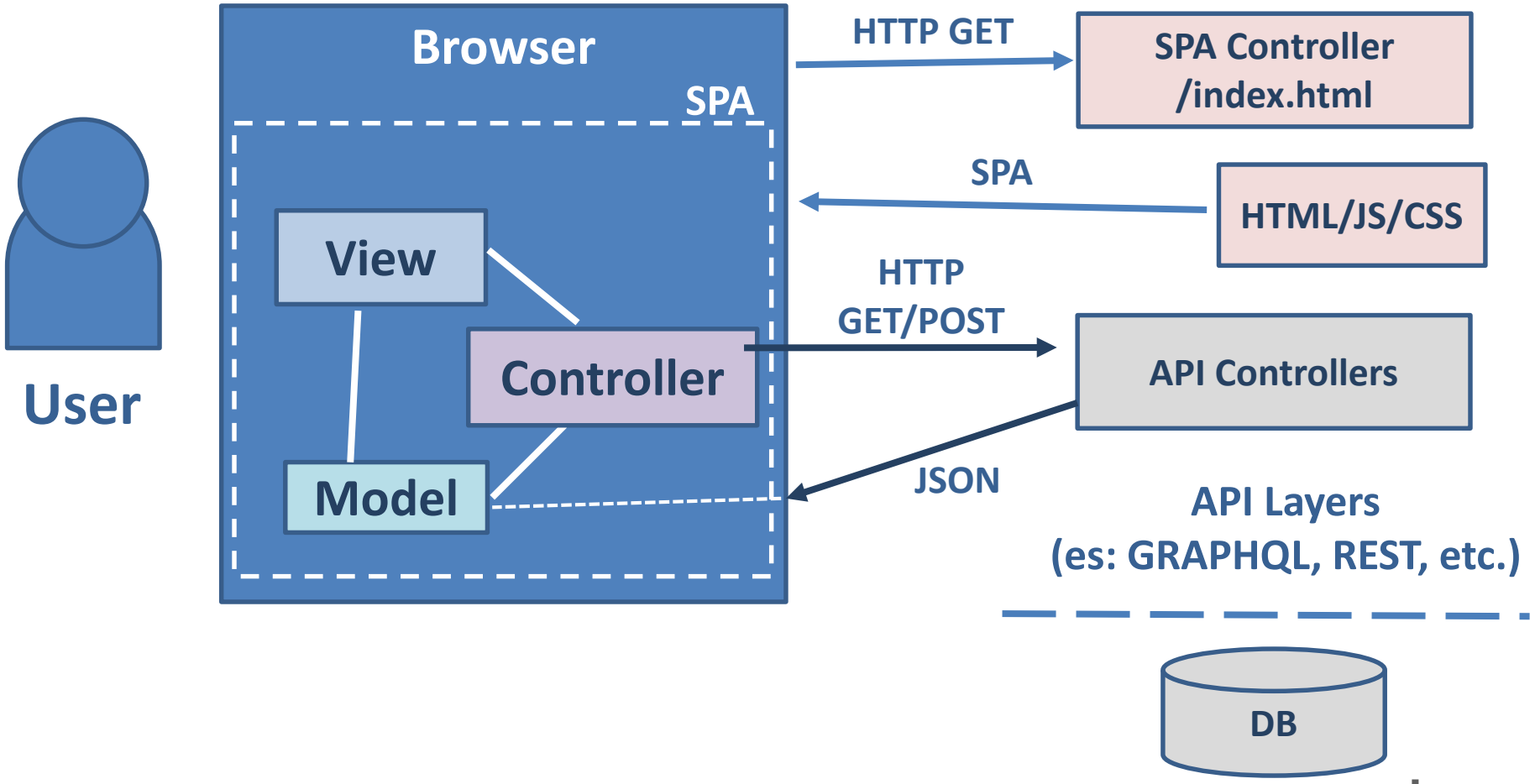
MVC nei moderni Web framework

- Con l'avvento dei browser più moderni, l'avvento di alcune innovazioni (come **`XMLHttpRequest`**, DOM API stabili, ECMAScript 6) ha portato a flessibilità e migliori performance
- Definizione e diffusione di web app (incluse Single Page Applications - SPA), con una maggiore computazione ed esecuzione lato client
- I web framework moderni supportano questa maggiore complessità lato client, mantenendo lo sviluppo delle applicazioni produttivo e predicibile

MVC nei moderni Web framework

- Questi framework introducono la possibilità di creare dei bundle di HTML, JS e CSS statici
- Queste Single Page Application includono la logica (in JS) per effettuare delle richieste HTTP API per un insieme di risorse servite da Controller API, che solitamente rispondono con del JSON

MVC nei moderni Web framework



MVC nei moderni Web framework

- Questo ci ha portato alla situazione attuale, in cui React, Vue e Angular sono i Web framework più utilizzati
- Tutti questi framework hanno un qualche tipo di **View**
- Vue ed Angular applicano un pattern **Model–view–view-model** (**MVVM**)
- **React** si basa su **Flux** (supporta il concetto del data flow unidirezionale, può essere considerata una alternativa al pattern MVC; le ***action*** sono inviate ad uno ***store*** attraverso un ***dispatcher***, le modifiche allo *store* sono propagate fino alla ***view***)

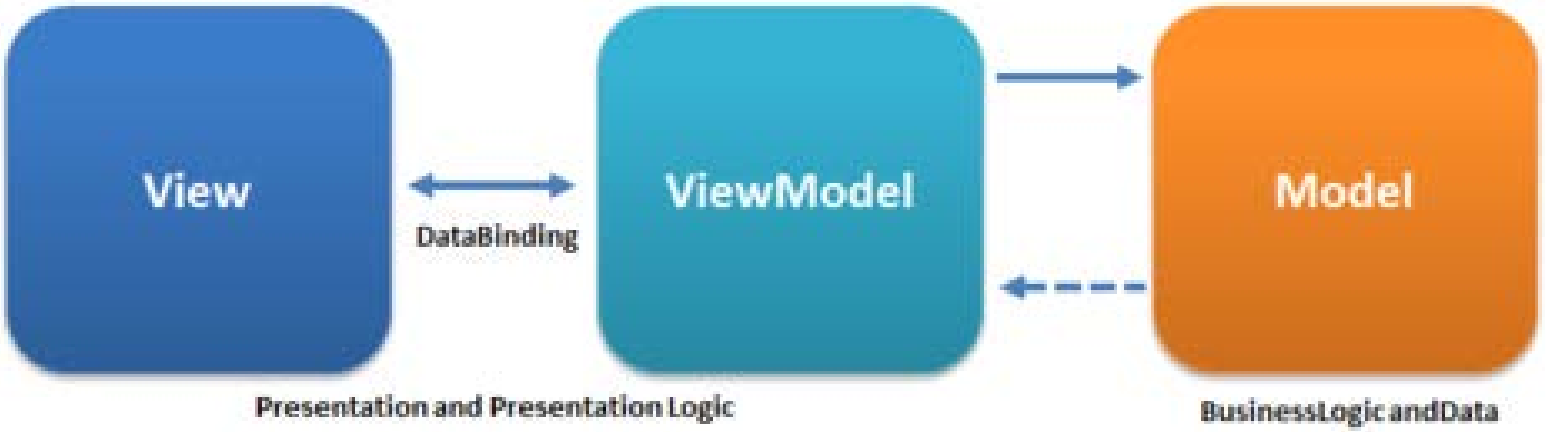
Model–view–viewModel (MVVM)

- MVC nel Web si applicava sviluppando un **controller** che permette alla **View** di manipolare il data **model**
- Indirettamente questo può portare a far comunicare la **View** con il backend
- Con le evoluzioni nel Web, è emerso come l'idea del **controller** nel pattern MVC non sia sempre efficace ed efficiente, ad esempio nelle Single Page Application
- Da questo nasce l'approccio del **ViewModel**, che va a sostituire il **Controller** e che si sposta sul frontend dell'applicazione

Model–view–viewModel (MVVM)

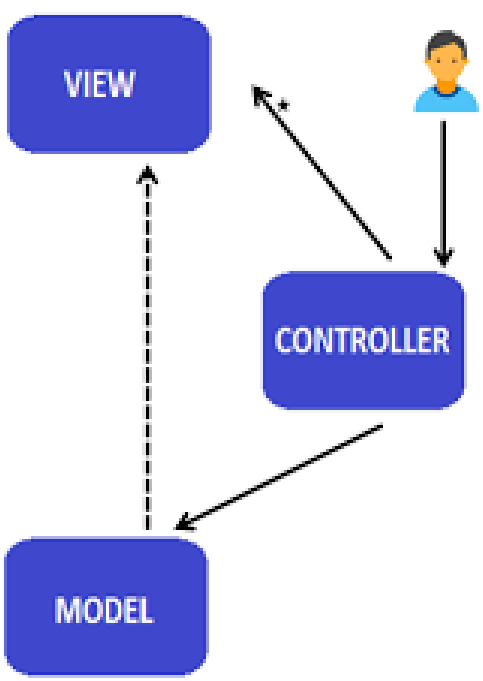
- Il **ViewModel** non è un controller, si comporta come un binder per i dati di **View** e **Model**.
- Quindi il pattern **MVVM** non separa **Model** e **View**, ma con l'approccio del data-binding supporta la comunicazione diretta tra **View** e **Model**
- Svantaggio: **ViewModel** consuma una quantità di memoria considerevole, se paragonato all'approccio con il **controller**. Non è consigliato quindi per semplici interfacce utente, perché considerato eccessivo.
- Per questo motivo il pattern MVVM è per lo più utilizzato da SPA (o single function application) sul Web (l'esecuzione di applicazioni più "consistenti" diventerebbe troppo pesante)

Model–view–viewModel (MVVM)

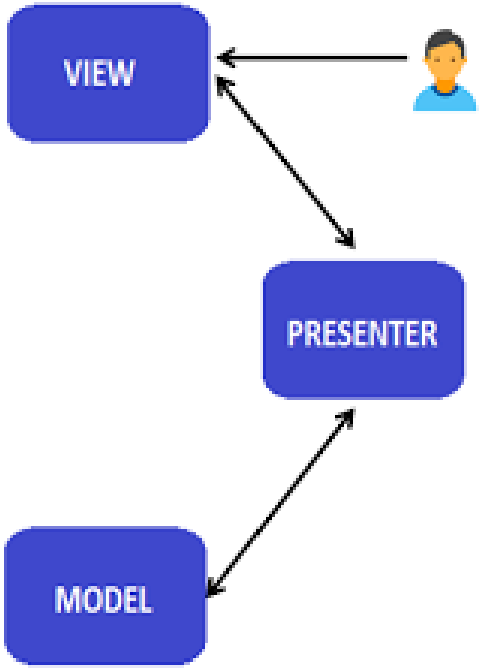


By Ugaya40 - Own work, CC BY-SA 3.0,
https://commons.wikimedia.org/w/index.php?curid=19056842

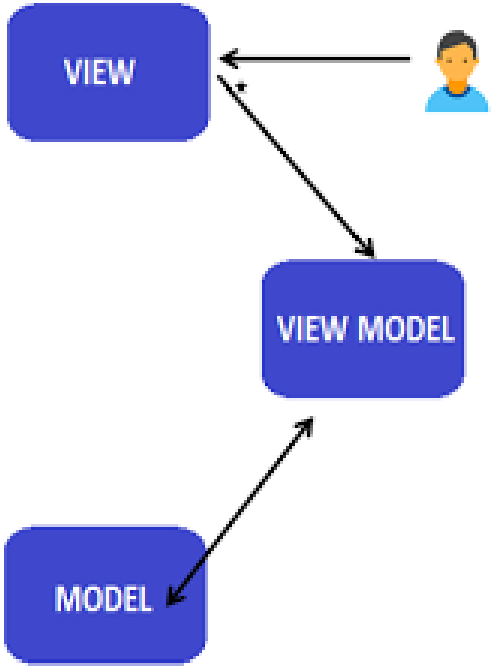
MVC, MVP, MVVM



MVC



MVP



MVVM

https://medium.com/@ankit.sinhal/mvc-mvp-and-mvvm-design-pattern-6e169567bbad

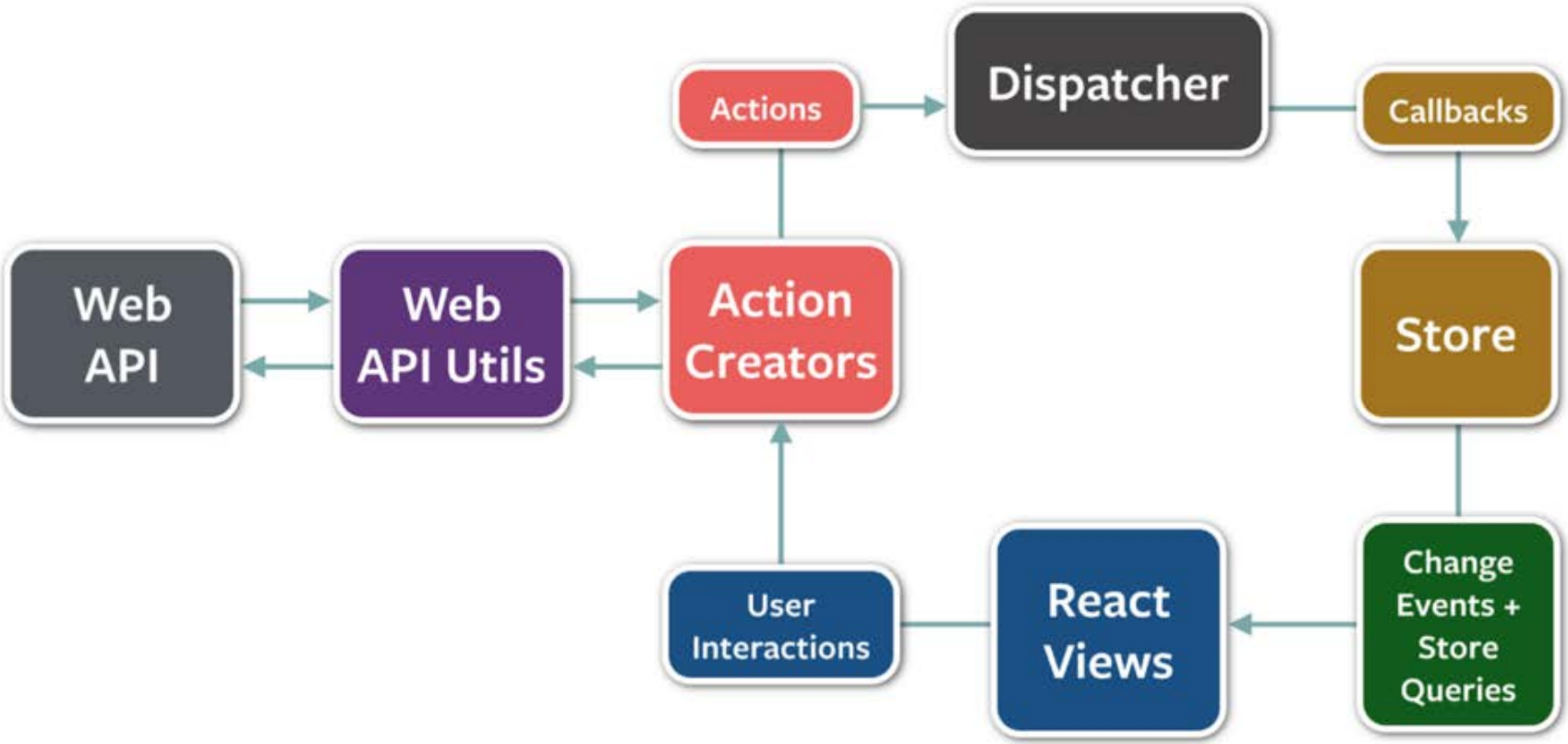
Model-View-Presenter

- Nel pattern MVP, il Presenter riceve l'input dagli utenti attraverso la View, processa i dati dell'utente con il support del Model e passa il risultato alla View
- Il Presenter manipola il Model e aggiorna la View
- View e Presenter sono disaccoppiati, separati tra loro e comunicano attraverso una interfaccia

Flux

- Flux è basato su data flow unidirezionale: man mano che una applicazione cresce in termini di complessità, diventa sempre più difficile gestire i cambi di stato con degli update della **view**, in particolare quando questi cambi derivano da fonti differenti
- React crea una nuova **View** come una funzione di uno stato non modificabile. L'applicazione può cambiare creando una nuova istanza di **Model** nello state tree. Quando React vuole aggiornare l'applicazione, sostituisce una parte del suo state tree con un nuovo oggetto, che «triggera» la creazione di una (o più) nuova **View**
- Mentre il Model in MVC rappresenta i dati che persistono e che saranno renderizzati dalla View, Flux divide le responsabilities del Model:
  - **Actions/APIs** sono usate per la **business logic**
  - **Store** è usato per la gestione dello stato

Flux diagram



Riferimenti

- MDN Web Docs. What is a web server? (https://developer.mozilla.org/en-US/docs/Learn/Common_questions/What_is_a_web_server)
- NGINX Resources. What Is an Application Server vs. a Web Server? (https://www.nginx.com/resources/glossary/application-server-vs-web-server/)
- What are web servers and how do they work (with examples httpd and nodejs) https://www.youtube.com/watch?v=JhpUch6lWMw
- Shruti Kohli. Web server (https://lecture-notes-forstudents.blogspot.com/2011/11/web-server.html)
- Mendel Rosenblum. CS142 Lecture Notes - Web Servers. Standford University (https://web.stanford.edu/class/cs142/)
- Mohamed Zahran. CSCI-UA.0201-003 - Computer Systems Organization. New York University (https://cs.nyu.edu/courses/spring13/CSCI-UA.0201-003/lecture23.pdf)

Riferimenti

- Robert Zhu. From MVC to Modern Web Frameworks (https://hackernoon.com/from-mvc-to-modern-web-frameworks-8067ec9dee65)
- Gamma, Erich. Design patterns: elements of reusable object-oriented software. Pearson Education India, 1995
- Z. Michael Luo. MVC vs. MVVM: How a Website Communicates With Its Data Models (https://hackernoon.com/mvc-vs-mvvm-how-a-website-communicates-with-its-data-models-18553877bf7d)
- Ankit Sinhal. MVC, MVP and MVVM Design Pattern (https://medium.com/@ankit.sinhal/mvc-mvp-and-mvvm-design-pattern-6e169567bbad)
- Reenskaug, Trygve. "The Model-View-Controller (MVC) Its Past and Present Trygve Reenskaug, University of Oslo (trygver@ ifi. uio. no)." (2003). http://heim.ifi.uio.no/~trygver/2003/javazone-jaoo/MVC_pattern.pdf
- Prof. Phuong Nguyen. CECS 475 Application Programming, Spring 2017 Semester (https://web.csulb.edu/~pnguyen/cecs475/leccecs475sp17.html)