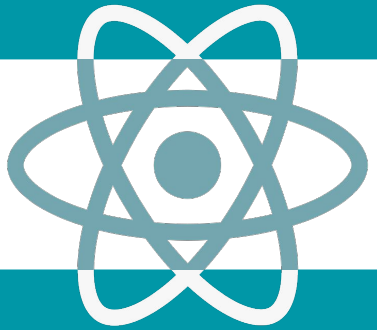
COESIONE ITALIA 21-27 PIEMONTE
Cofinanziato dall'Unione europea
REGIONE PIEMONTE



Programmazione Web

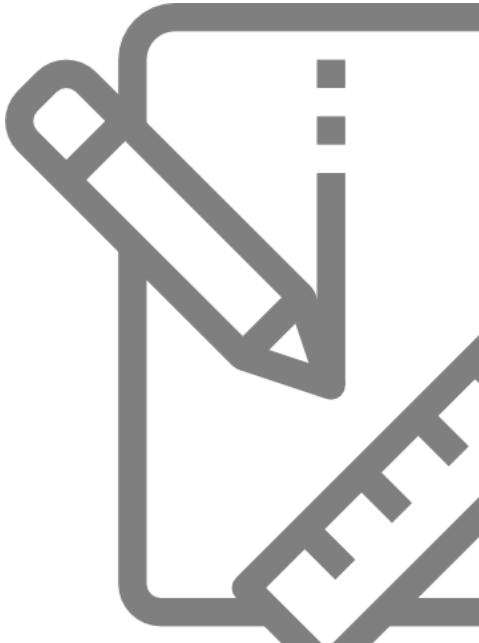
React

Davide Mantovani

synesthesia
the digital experience company

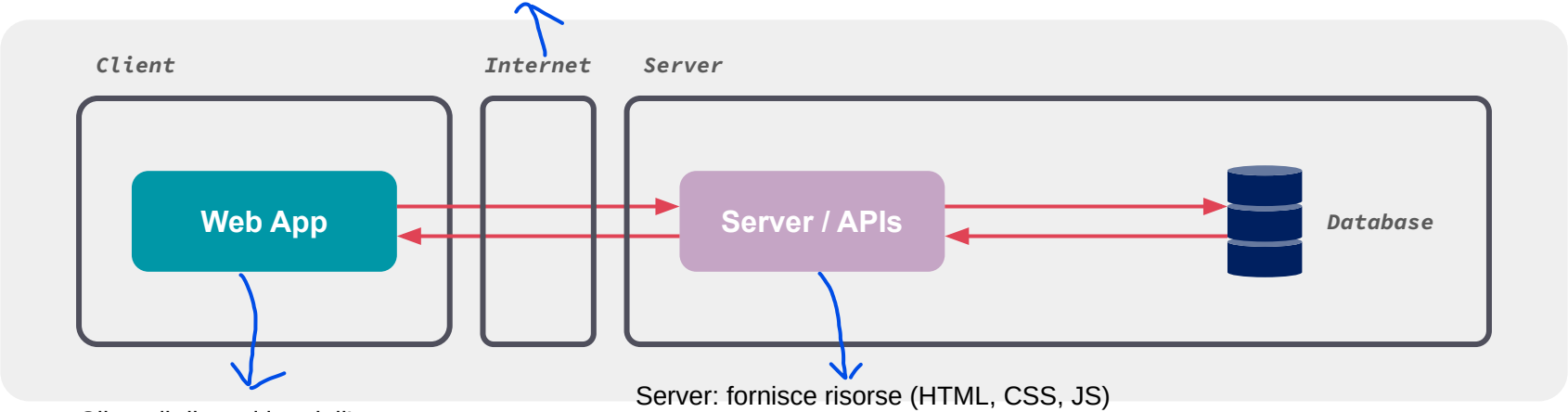
Intervento realizzato da
ITS TECNOLOGIE DELL'INFORMAZIONE E DELLA COMUNICAZIONE

Designing Architectures



Web architecture

Internet: il canale di comunicazione tra client e server.

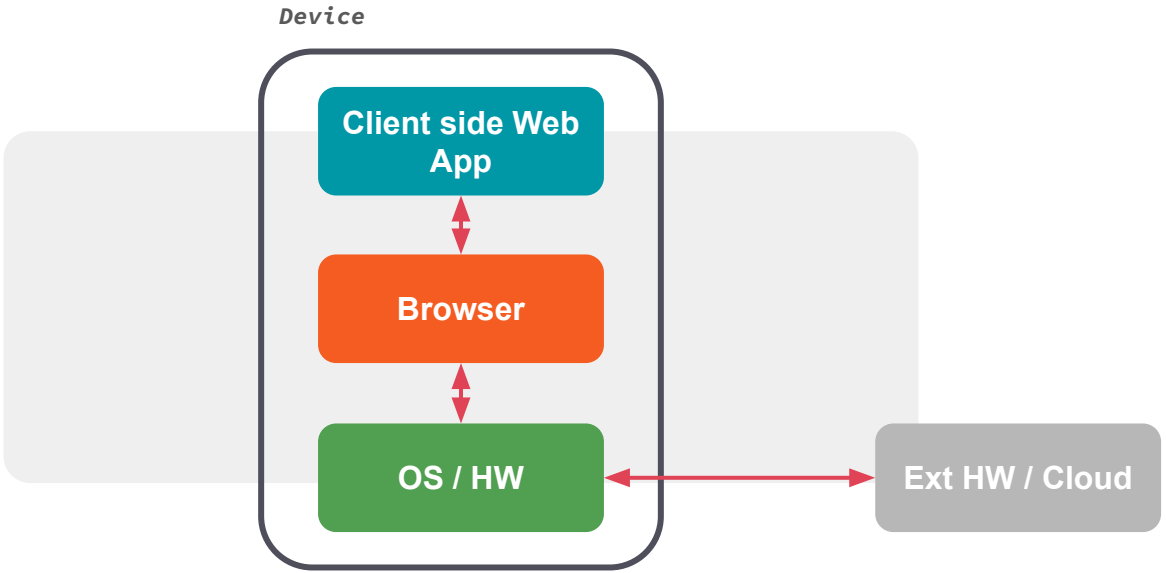


Client: il dispositivo dell'utente
(browser, sistema operativo, hardware).

Server: fornisce risorse (HTML, CSS, JS)
e gestisce i dati attraverso API e database.

Warning: this scheme is not exhaustive, but deliberately schematized broadly to represent the concept.

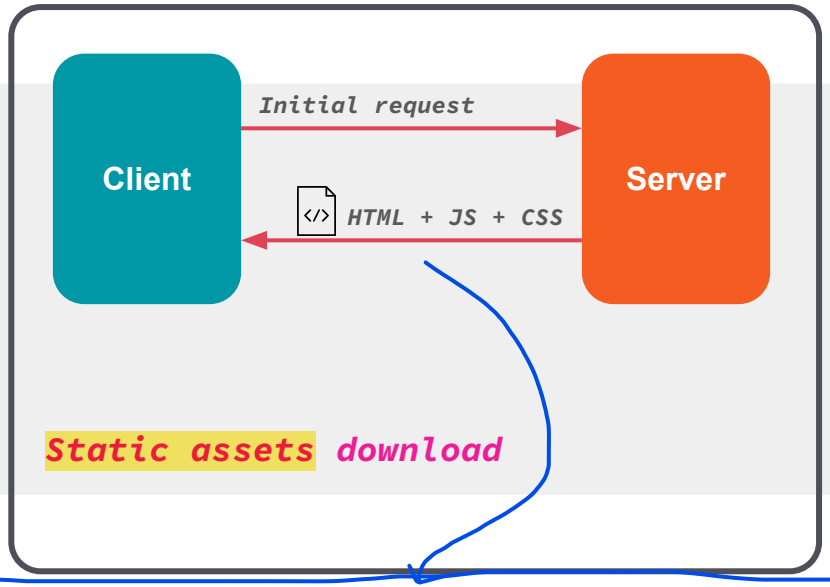
Device stack

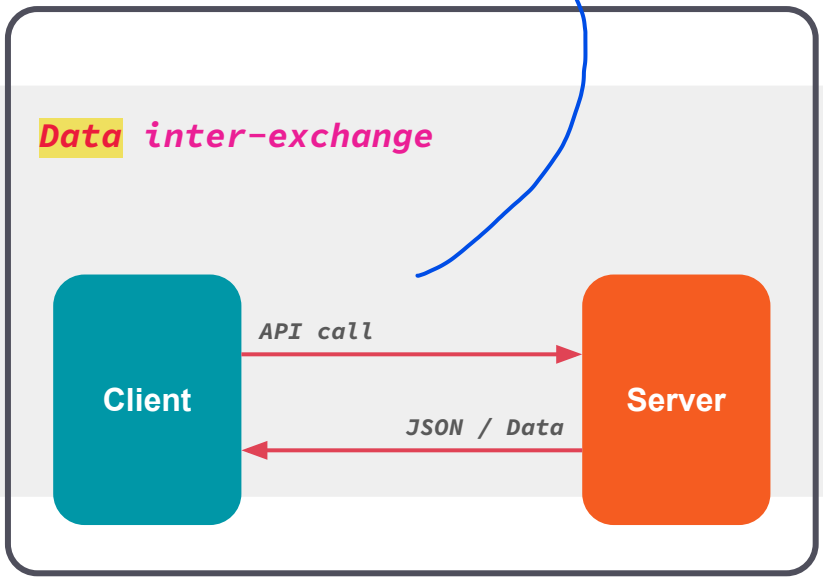


Warning: this scheme is not exhaustive, but deliberately schematized broadly to represent the concept.

Client/Server Interaction

il client invia chiamate API per scambiare dati in formato JSON o altri formati.



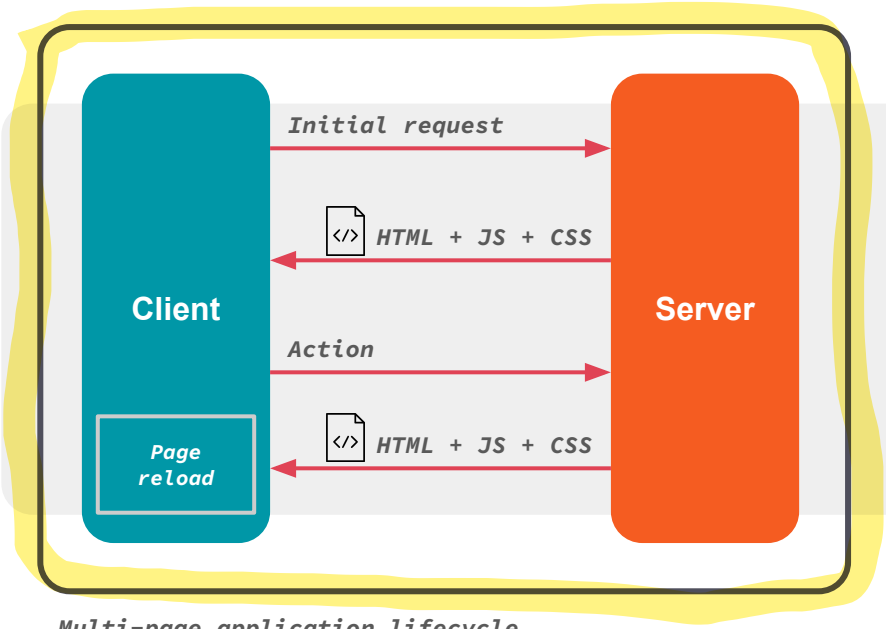


Durante la prima richiesta, il server invia risorse statiche (HTML, JS, CSS) al client.

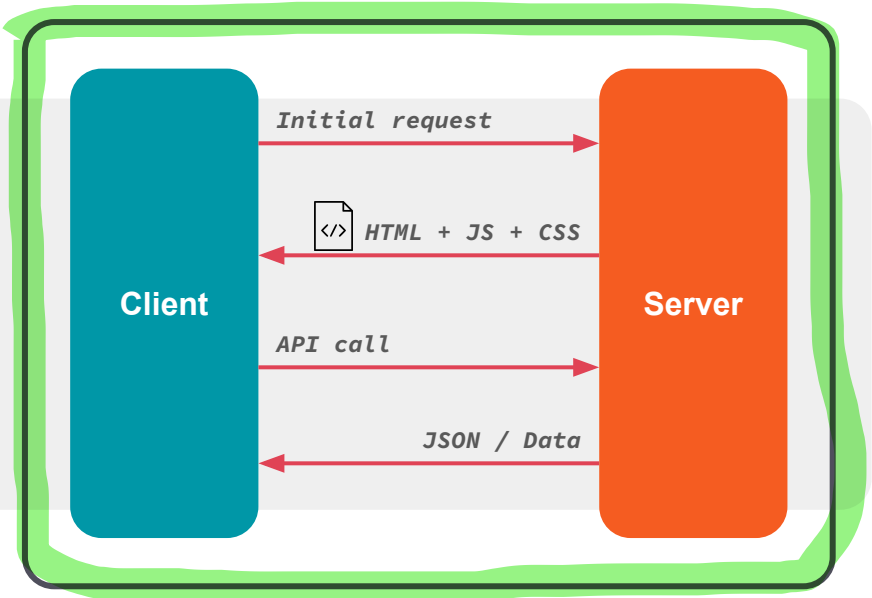
Warning: this scheme is not exhaustive, but deliberately schematized broadly to represent the concept.

MPA vs SPA

- MPA: Ogni azione richiede un RICARICAMENTO della pagina, inviando al server una nuova richiesta.
- SPA: Solo la prima richiesta carica risorse statiche; le azioni successive AGGIORNANO DINAMICAMENTE il contenuto tramite API.



Multi-page application lifecycle



Single page application lifecycle

Warning: this scheme is not exhaustive, but deliberately schematized broadly to represent the concept.

MV* patterns

Il pattern favorisce la separazione delle responsabilità per migliorare manutenzione e testabilità.

Model Recupera, modifica e salva i dati, collegandoli alle sorgenti (API, JSON, ecc.).

⇢ Retrieves, changes, saves the data that feed the application

⇢ Links the source of data (API, JSON, …) to the application

View Rappresenta visivamente i dati e consente interazioni con l'utente.

⇢ Allows user interaction

⇢ It's also a visual representation of models

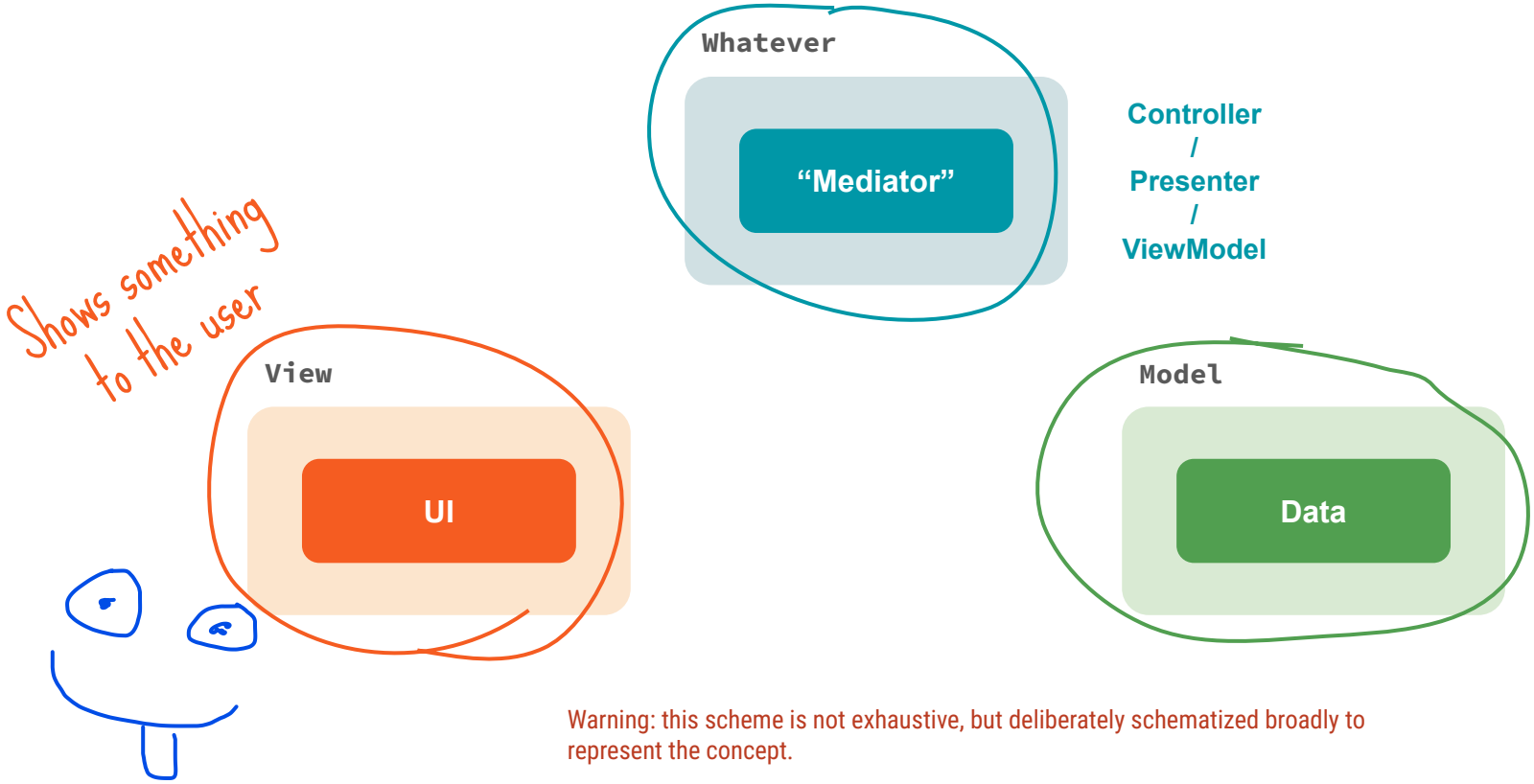
*** (Whatever)** (Whatever):

⇢ Controller: manipulates models data and serves them to the views - Controller: gestisce i dati e li invia alla view.

⇢ Presenter: holds the user-interface business logic - Presenter: contiene la logica dell'interfaccia utente.

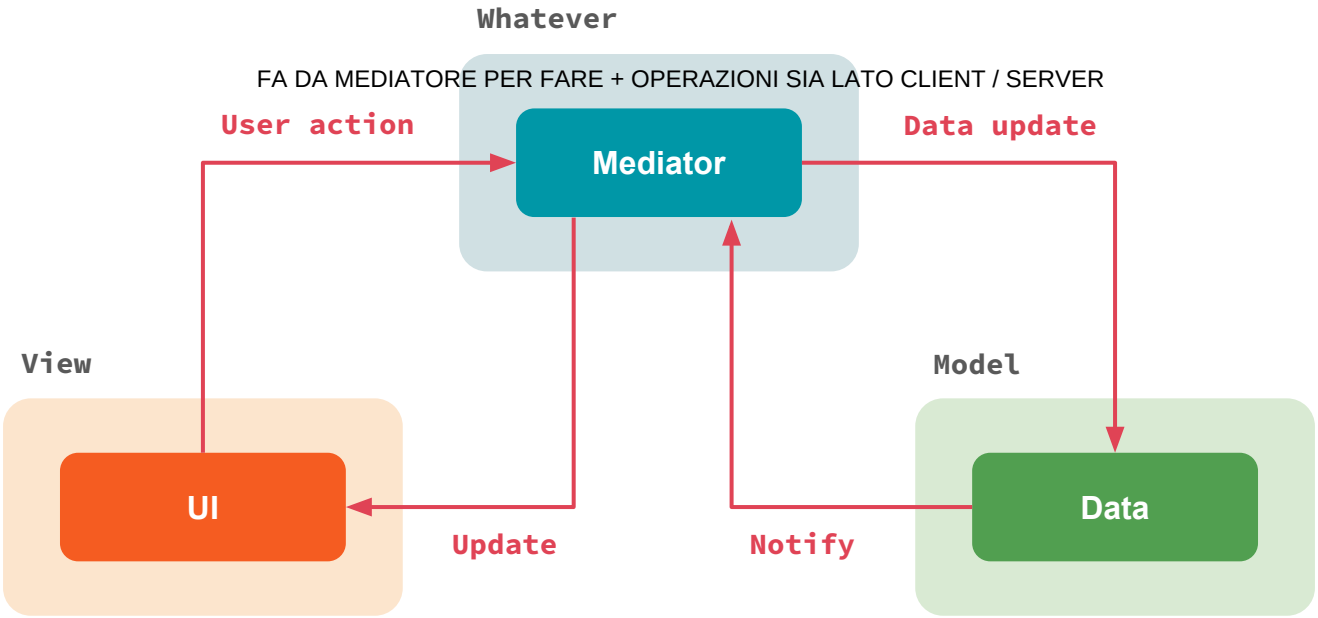
⇢ ViewModel: transfers data and events between models and views - ViewModel: media tra i dati e la vista.

MV* patterns



Warning: this scheme is not exhaustive, but deliberately schematized broadly to represent the concept.

MV* patterns



Warning: this scheme is not exhaustive, but deliberately schematized broadly to represent the concept.

- Deve essere modulare, ottimizzata e reattiva.
- Organizzata in componenti riutilizzabili.
- Supporta la separazione delle responsabilità evitando logica interna complessa.

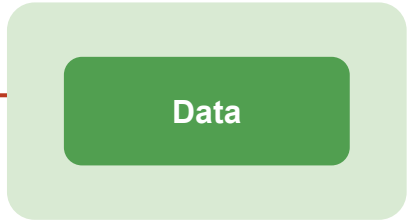
UI

(Interfaccia Utente)

- **Should not contain logic**, to ensure separation of concerns (SoC).
- **Should be organised** in reusable components.
- **Should be responsive** to support screen resolutions.
- **Should be optimised** to be fast.
- **Should be tracked** to catch errors.
- **Could be chunked in micro frontends**, useful for large apps across many teams.

Data

- Devono essere strutturati e validati.
- Possono essere centralizzati o temporanei a seconda del caso d'uso.



- **Should be defined by data structure**, to be inspected.
- **Should integrate data validation**, to ensure input cleaness.
- **Could be disposable**, when data is used for a limited time.
- **Could be centralised**, when data is used across components and pages.

Mediator

- Contiene la logica della vista (view logic).
- È testabile e tracciabile per garantire affidabilità.

"Mediator"

- **Should contain view logic**, to ensure separation of concerns (SoC) and centralisation.
- **Should be tracked** to catch errors.
- **Could be easily tested**, to support TDD and regression tests.

Designing an application

1. Paper and pencil **flowchart**
2. Detect the main **containers** in which happens interaction
3. Break the UI in **Layouts**, **Containers** and **Components**, Atoms if needed
4. Detect the main **data sources** and their format
5. Use **data mocks**
6. Develop the **static version**
7. Identify where user interaction has **effects** on the application
8. Attach **state management** and **data fetching**

Build your **own flow** but stick to a **plan**!