Tutorato di:

# Web Programming, Design & Usability

Tutor: Alessio Tudisco


Università di Catania

# Front-End in pillole

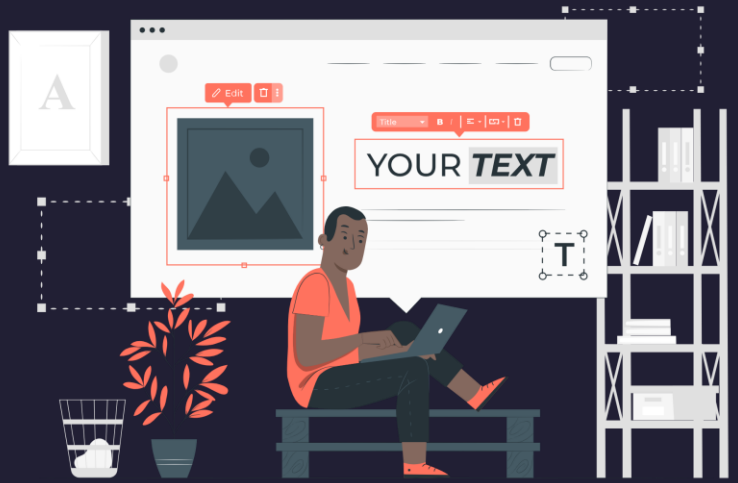
La parte con cui l'utente interagisce

# Front-End

In informatica, il termine «*front-end*» si riferisce alla parte di un'applicazione, di un sistema o di un sito web che è visibile e con cui gli utenti interagiscono direttamente.

Si occupa della presentazione dei dati e dell'interfaccia utente.

Il front-end coinvolge principalmente la *progettazione* e lo *sviluppo* di componenti come *l'interfaccia grafica*, il *layout*, gli *elementi di navigazione* e *l'interazione utente*. Ciò include l'aspetto visivo, il posizionamento degli elementi, i colori, le animazioni e altri aspetti che influenzano l'esperienza dell'utente.




Per creare il front-end di un sito web, si utilizzano una combinazione di linguaggi di markup come HTML per strutturare il contenuto, CSS per definire lo stile e l'aspetto, e JavaScript per aggiungere interattività e gestire eventi.

Con il passare del tempo sono state sviluppate numerose *librerie front-end* che semplificano lo sviluppo fornendo componenti, funzionalità e utilità predefinite. Queste librerie offrono un insieme di strumenti preconfezionati per gestire aspetti come la gestione dello stato, l'interazione utente e la gestione dei dati.

I più famosi al momento sono:

- **Framework**: Angular, React;
- **UI**: Svelte, Vue;
- **Bundler**: WebPack, ParcelJS

# Back-End in pillole

L'*Hyper-Text Markup Language* non è un linguaggio di programmazione!

# Back-End

In informatica, il termine «*back-end*» si riferisce alla parte di un'applicazione, di un sistema o di un sito web che gestisce la *logica di business*, *l'elaborazione dei dati* e la *comunicazione con il database* o *altri sistemi esterni*.


Si occupa delle funzionalità che non sono visibili direttamente agli utenti finali, ma sono *essenziali* per il funzionamento dell'applicazione.

Un aspetto fondamentale è ricoperto dalle API (Application Programming Interface) che consentono la comunicazione e l'interazione tra il front-end e il back-end stesso.

Le API definiscono i *metodi di comunicazione*, *i formati dei dati* e *le operazioni disponibili* per consentire lo scambio di informazioni tra i diversi componenti del software, locali o remote. Definiscono le interfacce attraverso le quali le applicazioni possono richiedere o inviare dati, eseguire operazioni o accedere a funzionalità specifiche offerte da un'altra applicazione o sistema.


Possono essere implementate utilizzando vari protocolli di comunicazione come HTTP, SOAP, GraphQL e altri. Inoltre, possono essere disponibili come *API pubbliche*, accessibili a tutti gli sviluppatori, o come *API private*, accessibili solo a utenti o applicazioni autorizzate.

Con il termine «*CRUD*» si rappresentano le *quattro operazioni di base* che possono essere eseguite sui dati:
**Create** (C), **Read** (R), **Update** (U), **Delete** (D).

# Breve cenno sull'autenticazione

L'*autenticazione* delle API è un *processo critico* che **verifica l'identità e l'autorizzazione** di un entità che cerca di accedere ad un'API, al fine di garantire che solo entità autorizzate possano accedere alle risorse o alle funzionalità fornite dall'API. Ci sono *diversi meccanismi di autenticazione* comunemente utilizzati per proteggere le API. Tra i più comuni ci sono:

- **API KEY**: è un *identificatore univoco* assegnato a un'entità autorizzata per accedere a un'API. Viene solitamente inviata come parametro (*GET o POST*) nella richiesta API. Può essere considerata come una forma di autenticazione leggera, ma può non essere sicura se viene compromessa;
- **Token**: viene generato dal server di autenticazione dopo un processo di login riuscito e inviato all'utente. Successivamente viene inviato come parte dell'intestazione (*header*) della richiesta API successive. Si possono implementare meccanismi di scadenza e refresh del token per una maggiore sicurezza in caso di leak;
- **OAuth2**: è un protocollo di autorizzazione. È un po' complicato e sconsigliato per chi è alle prime armi;

# Node.js

JavaScript esce dal browser

# Node.js

È un ambiente *di runtime open-source* basato su JavaScript che consente di eseguire codice JavaScript al di fuori dei browser web. È costruito sul *motore JavaScript V8* di Google Chrome e consente di creare applicazioni server-side altamente scalabili e ad alte prestazioni.

Utilizza un *modello di I/O non bloccante* e ad eventi, che lo rende particolarmente adatto per le applicazioni in tempo reale e le applicazioni che richiedono una gestione efficiente delle connessioni simultanee. Questo modello consente al server di elaborare richieste asincrone in modo efficiente, evitando il blocco o l'attesa di operazioni di I/O.

Il gestore dei pacchetti di Node.js è chiamato *npm* (*Node Package Manager*). Consiste in un repository online di pacchetti e moduli JavaScript pronti per l'uso che possono essere facilmente installati e gestiti all'interno di un progetto Node.js.

Permette di utilizzare librerie di terze parti, framework e strumenti senza doverli sviluppare da zero, semplificando notevolmente la gestione delle dipendenze del progetto e focalizzando lo sviluppo dell'applicazione piuttosto che la gestione manuale dei pacchetti.

Esistono package manager alternativi che con il tempo si sono sempre più affermati grazie alle loro caratteristiche peculiari:

- **yarn**: creato da Facebook per affrontare alcune limitazioni e migliorare le prestazioni rispetto a npm;
- **pnpm**:introduce il concetto di cache dei package, al fine di limitare la dimensione della cartella *node_modules*.

# Comandi essenziali di npm

| | |
|---|---|
| **npm init** | Inizializza un nuovo progetto npm creando un file "package.json". Durante il processo di inizializzazione, vengono richieste informazioni sul progetto, come nome, versione, autore e dipendenze iniziali; |
| **npm install** | Installa le dipendenze definite nel file "package.json". Questo comando scarica e installa i pacchetti elencati nel file "dependencies". È possibile specificare i pacchetti da installare come argomenti aggiuntivi; |
| **npm install --save <nome-pacchetto>** | Aggiunge un pacchetto come dipendenza al progetto e lo salva nel file "package.json". Il flag *--save* è opzionale in npm v. 5 e successive, poiché le dipendenze vengono automaticamente salvate nel file "package.json" |
| **npm install --global <nome-pacchetto>** | Installa un pacchetto globalmente, rendendolo disponibile in tutto il sistema. Questo è utile per strumenti di linea di comando o utility che si desidera utilizzare in vari progetti. |
| **npm uninstall <nome-pacchetto>** | Rimuove un pacchetto dal progetto e lo disinstalla. Questo comando rimuove anche il pacchetto dal file "package.json" |
| **npm update** | Aggiorna i pacchetti installati alla loro versione più recente. Questo comando verifica se sono disponibili nuove versioni dei pacchetti e li scarica e installa se necessario |
| **npm run <nome-script>** | Esegue uno script definito nel file "package.json". È possibile definire script personalizzati nel campo "scripts" del file "package.json" e utilizzarli per automatizzare operazioni come l'avvio del server di sviluppo o la compilazione del progetto |

# Node.js Modules

## Express.js

# Express.js

È un framework web leggero e flessibile per Node.js. È uno dei framework web più popolari per lo sviluppo di applicazioni server-side utilizzando JavaScript. Rispetto al modulo *http* nativo di Node.js,  Express.js semplifica la creazione di server HTTP e la gestione delle richieste e delle risposte.

# Routing Express.JS

Express.js offre *un sistema di routing* semplice ed intuitivo che permette di definire le *route* (percorsi URL) e le relative azioni da eseguire quando viene richiesta una di quest'ultime. Tali route possono essere: *statiche, parametriche e di pattern matching.*

- **app.[get-post-put-delete](path, callback)**: Definisce una route con il metodo HTTP selezionato sulla *path specificata*. Quando tale path viene raggiunta da una richiesta di tipo desiderato verrà eseguita la funzione di callback specificata;
- **app.all(path, callback)**: simile alla precedente, ma per la path specificata accetta qualunque verbo HTTP;
- **app.use(path, middleware)**: Aggiunge un *middleware* globale o locale per tutte le richieste che corrispondono al percorso specificato. Il middleware può essere una funzione di callback o un'altra istanza di Express.js;

```
const express = require("express");
const app = express();

app.get("/", (req, res) => {
    res.status(200).send("Home Page");
});

app.get("/about", (req, res) => {
    res.status(200).send("About");
});

// * means everything
app.all("*", (req, res) => {
    res.status(404).send("<h1>404</h1>")
});

app.listen(8080);
```

# Middleware

Un middleware è una funzione che viene eseguita durante il ciclo di vita di una richiesta HTTP. Funziona come un intermediario tra la richiesta del client e la risposta del server, consentendo di eseguire controlli di sicurezza o aggiungere funzionalità extra alla gestione delle richieste.

La definizione di un middleware necessita di 3 parametri:

- **Request**: rappresenta la richiesta HTTP che è stata ricevuta dal server;
- **Response**: rappresenta un oggetto risposta con cui ritornare qualcosa all'utente;
- **Next:** rappresenta l'API che l'utente cercava di raggiungere.

```
const express = require('express')
const app = express()

app.use((req, res, next) => {
  console.log('Time:', Date.now())
  next()
})
```

# Metodi principali di Request

| | |
|---|---|
| **req.params** | Restituisce un oggetto contenente i parametri di route definiti nella route corrente.<br>Ad esempio, se definita una route come «*/users/:id*», si può accedere all'ID utilizzando *req.params.id;* |
| **req.query** | Restituisce un oggetto contenente i parametri di query della richiesta.<br>Ad esempio, per una richiesta GET come «*/users?name=John&age=25*», si può accedere ai parametri utilizzando *req.query.name* e *req.query.age;* |
| **req.body** | Restituisce i dati inviati nel corpo della richiesta, utilizzato per le richieste POST.<br>È necessario utilizzare un *middleware di analisi del corpo* come *express.json()* o *express.urlencoded().*<br>Esempio: *app.use(express.json())* |
| **req.headers** | Restituisce un oggetto contenente gli header della richiesta. Si può accedere a un singolo header utilizzando *req.headers['nome-header'];* |
| **req.path** | Restituisce il percorso della richiesta (escluso il dominio e la query string); |
| **req.method** | Restituisce il metodo HTTP della richiesta; |

# Metodi principali di Response

| | |
|---|---|
| **res.send(data)** | Invia una risposta al client. Il parametro data può essere una stringa, un oggetto JavaScript o un array JSON. Express.js determina automaticamente il tipo di contenuto in base al tipo di dati fornito; |
| **res.json(data)** | Invia una risposta JSON al client. Il parametro data può essere un oggetto JavaScript o un array JSON. Express.js imposta automaticamente l'intestazione *Content-Type* sulla corretta *applicazione/json;* |
| **res.status(code)** | Imposta il *codice di stato HTTP* della risposta; |
| **res.redirect(path)** | Reindirizza il client a un'altra *route* o URL specificato dal percorso (*path*); |
| **res.sendFile(path, options)** | Invia un file al client. Il parametro *path* è il percorso del file da inviare. È possibile specificare opzioni come il tipo di contenuto (*options.contentType*) e il controllo della cache (*options.cache*); |
| **res.header(field, value)** | Imposta un'intestazione personalizzata nella risposta. Il parametro *field* è il nome dell'intestazione, mentre *value* è il valore dell'intestazione; |
| **res.status(code).send(data)** | Combina l'impostazione del codice di stato e l'invio della risposta in un'unica chiamata; |

# Exercise Time

# Il Quiz Client-Server

## Struttura

Stessa del Quiz precedente

## JavaScript

Il question-set in formato json deve essere ottenuto mediante richiesta GET ad una API fatta tramite express.js, le domande devono essere inserite nel DOM dinamicamente.

## Logica

L'API deve prevedere un middleware di autenticazione che controlli l'utilizzo di una API_KEY. In caso di entità autorizzata, si può proseguire con la route.

## Tips

Si prediliga un codice che non dipenda da un numero fisso di domande... usate l'iterabilità!