

# Programmare server

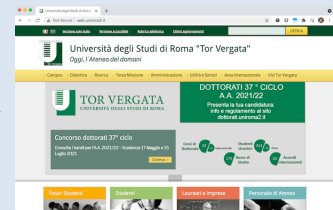
# Frontend e Backend

# Frontend e Backend

BROWSER

# Frontend e Backend

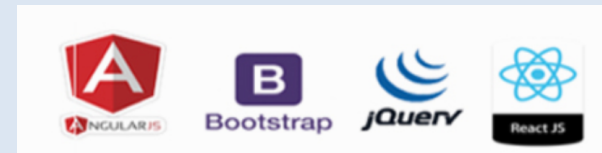
BROWSER



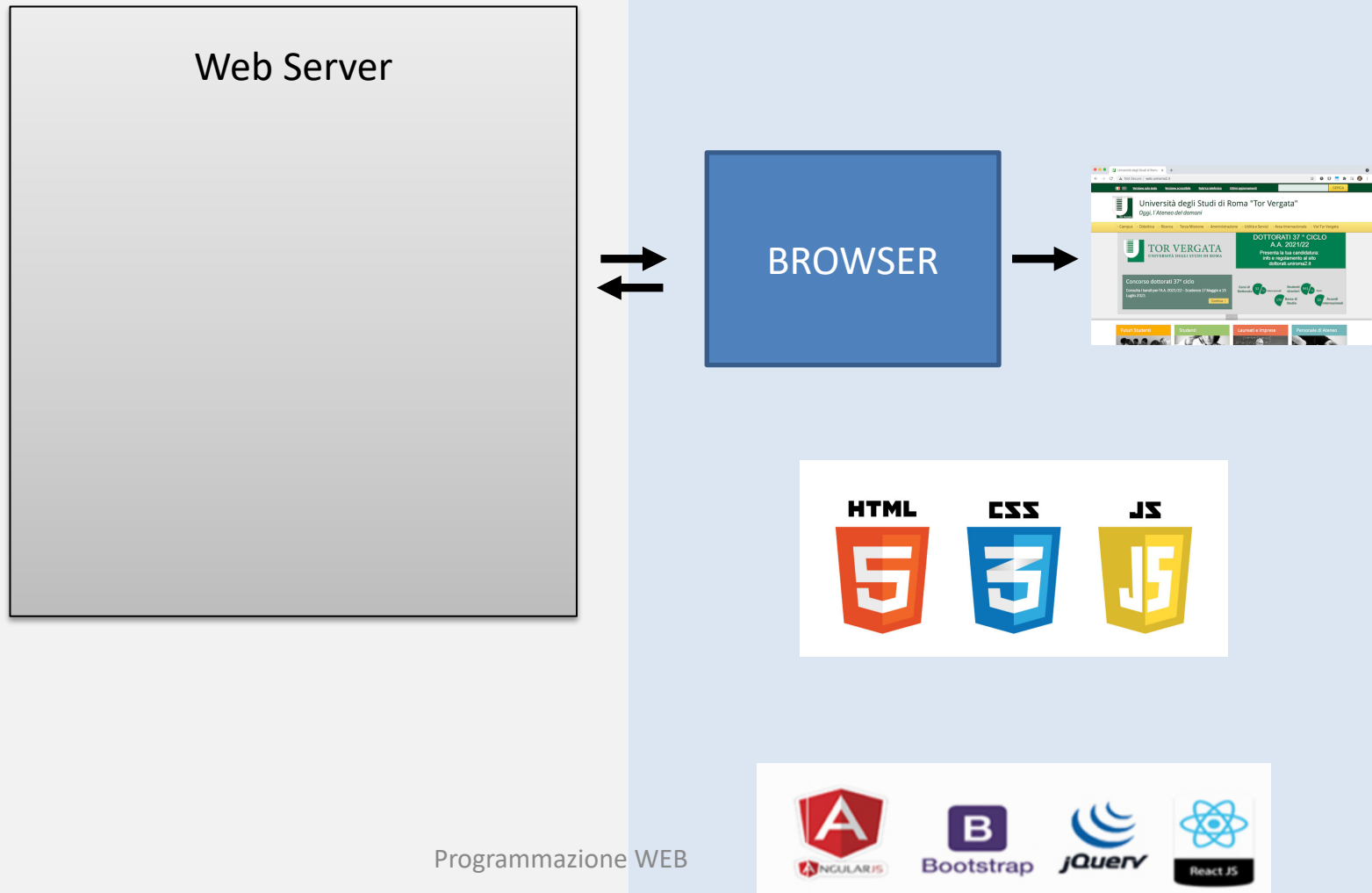
# Frontend e Backend



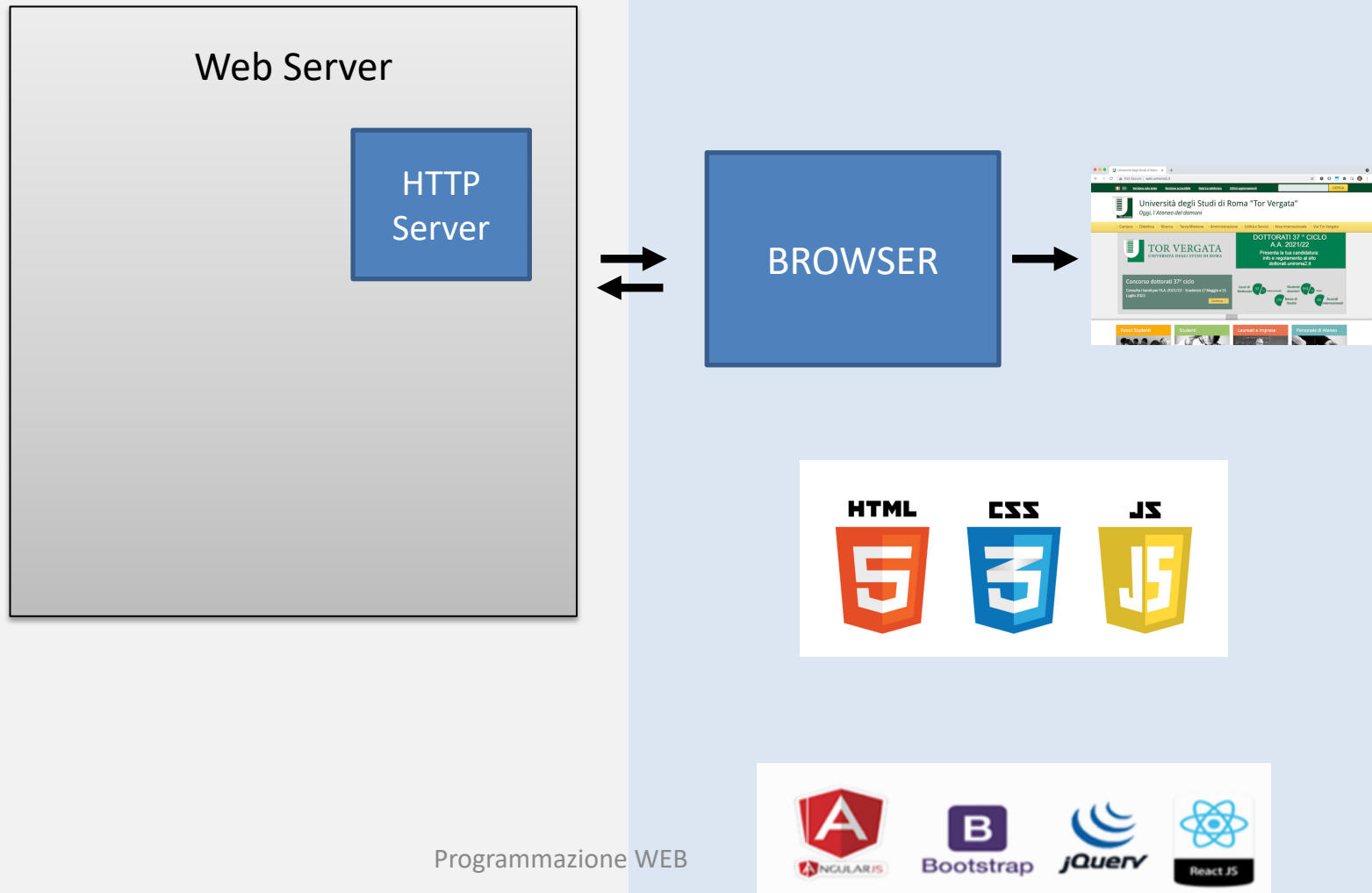
# Frontend e Backend



# Frontend e Backend

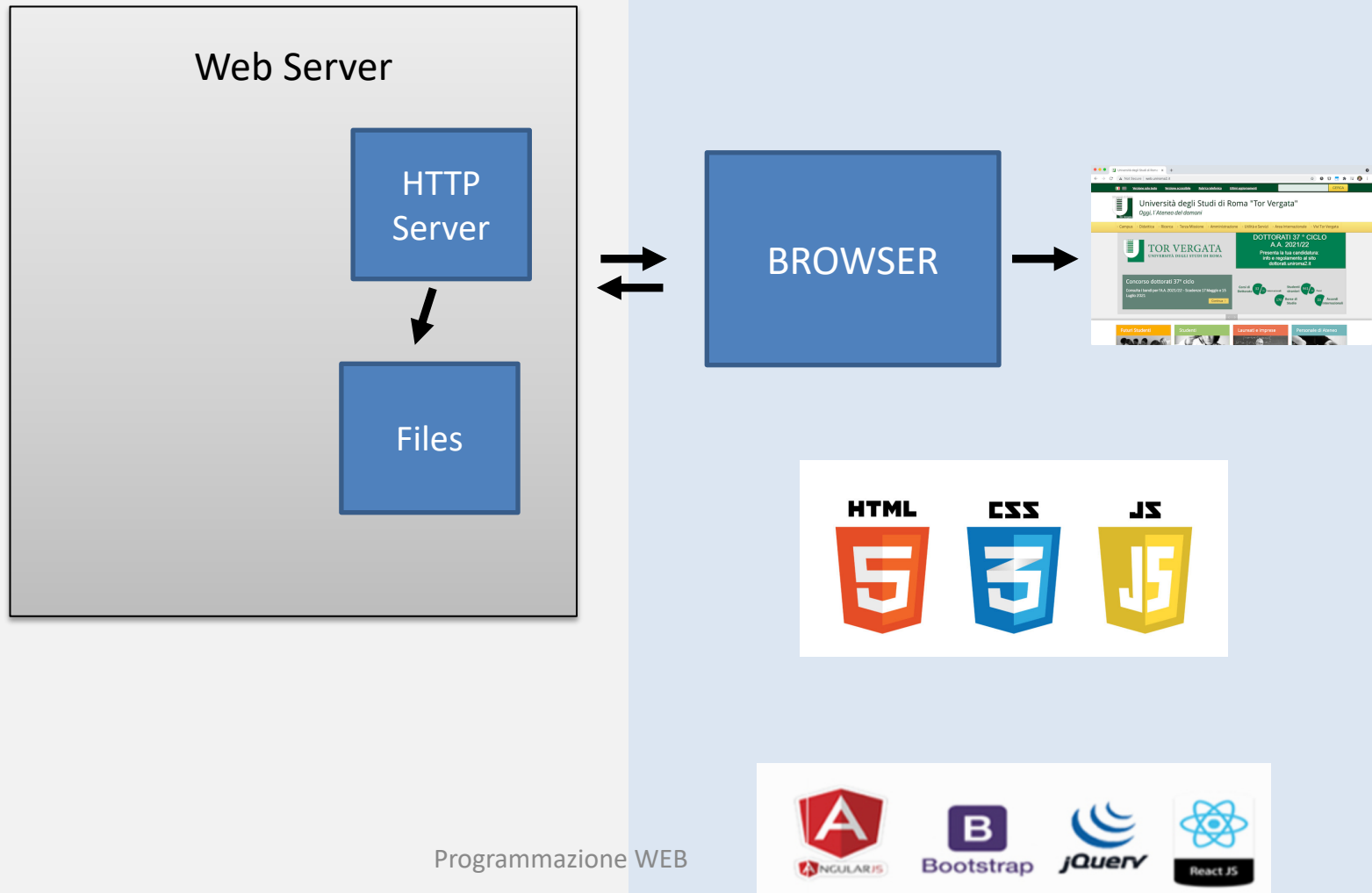


# Frontend e Backend

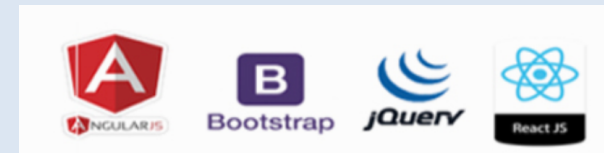
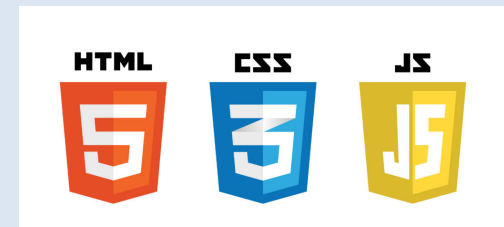
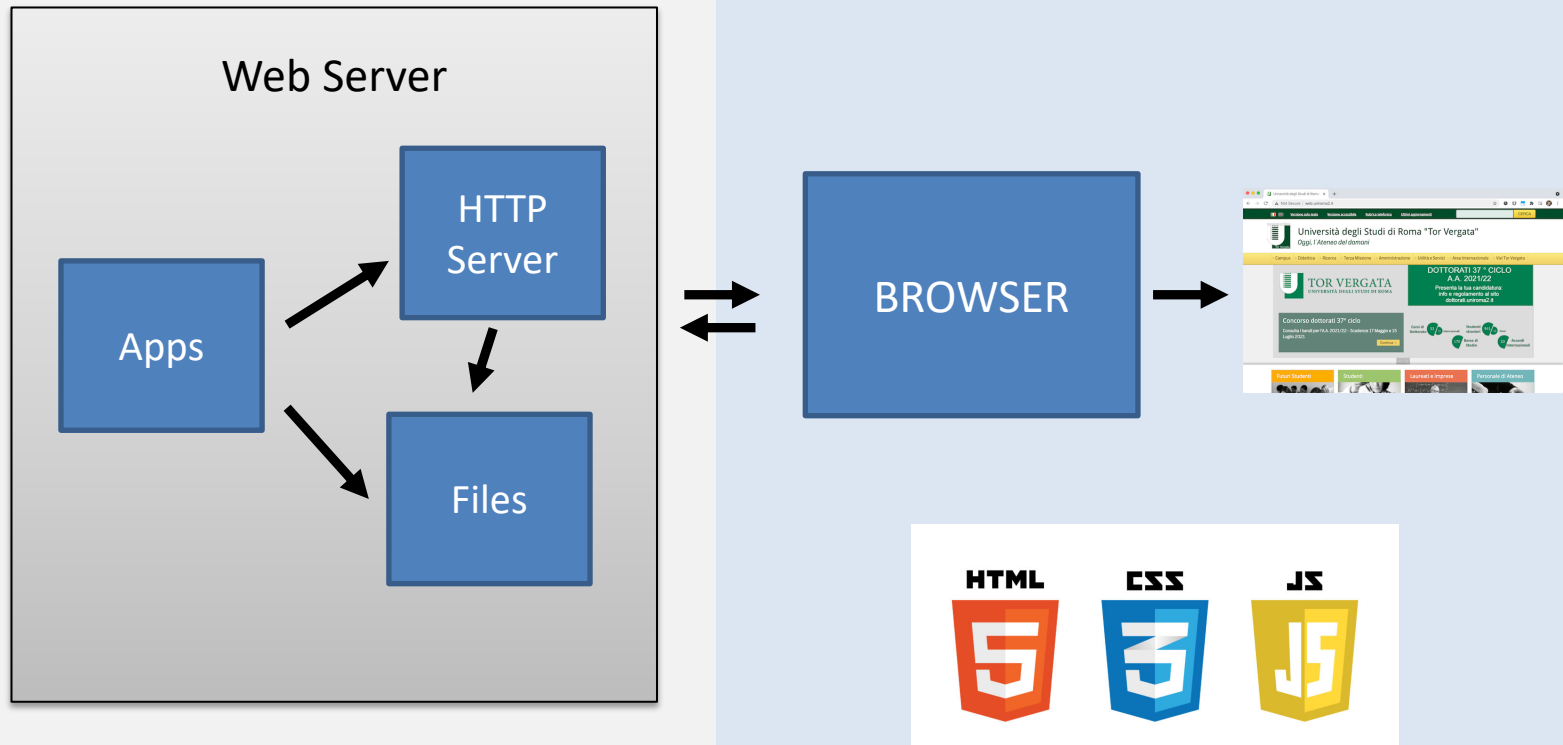




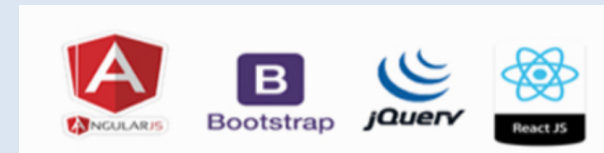
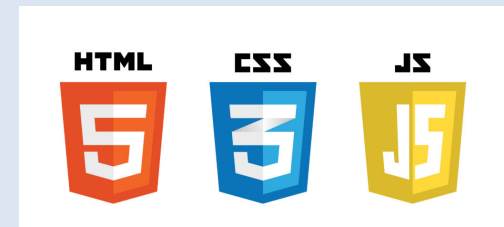
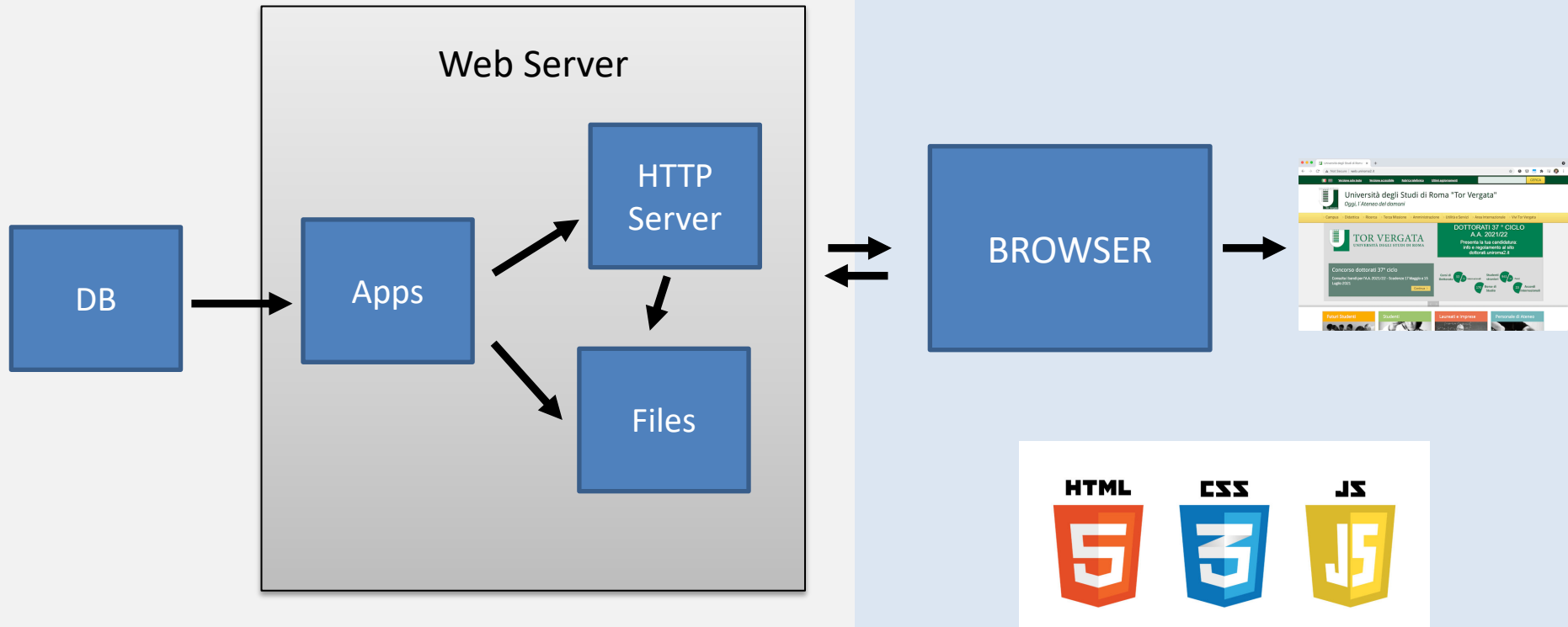
# Frontend e Backend



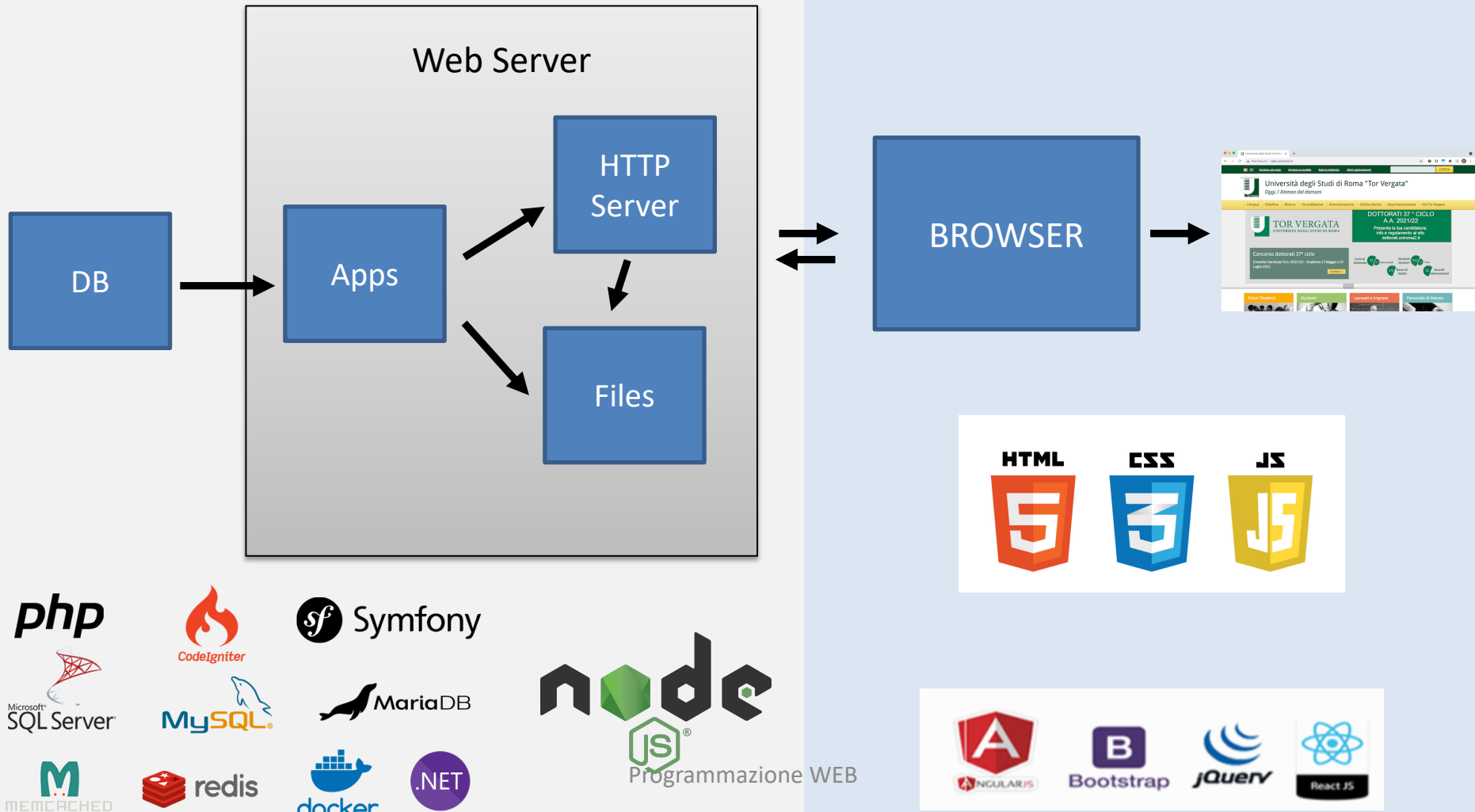
# Frontend e Backend



# Frontend e Backend



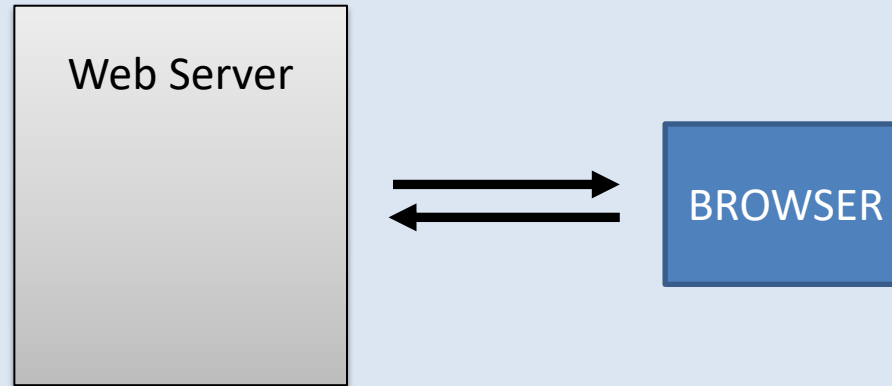
# Frontend e Backend



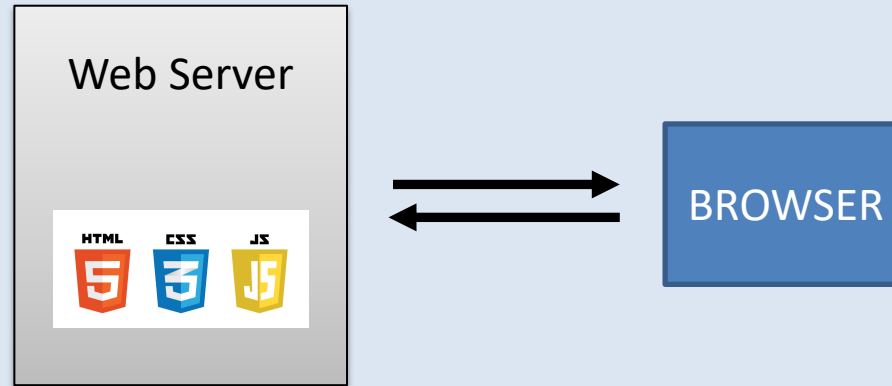
# Siti statici vs dinamici

BROWSER

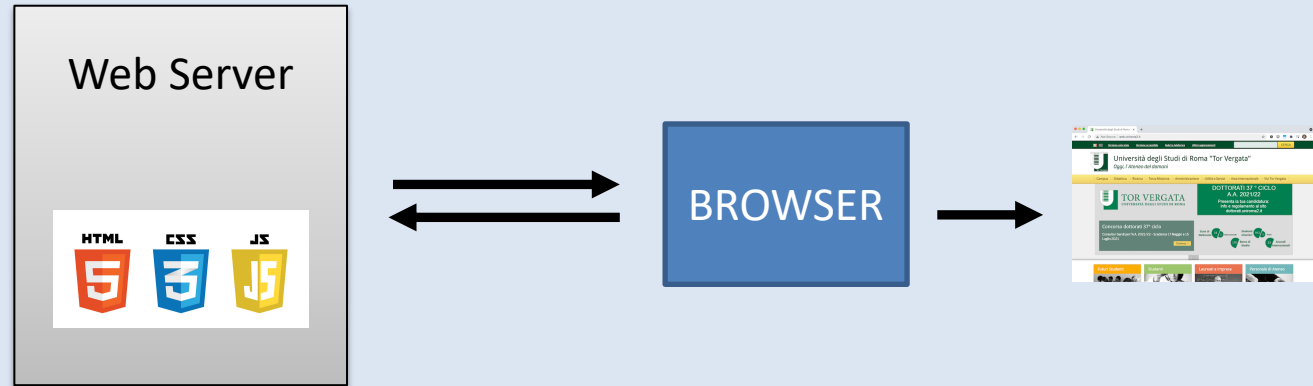
# Siti statici vs dinamici



# Siti statici vs dinamici

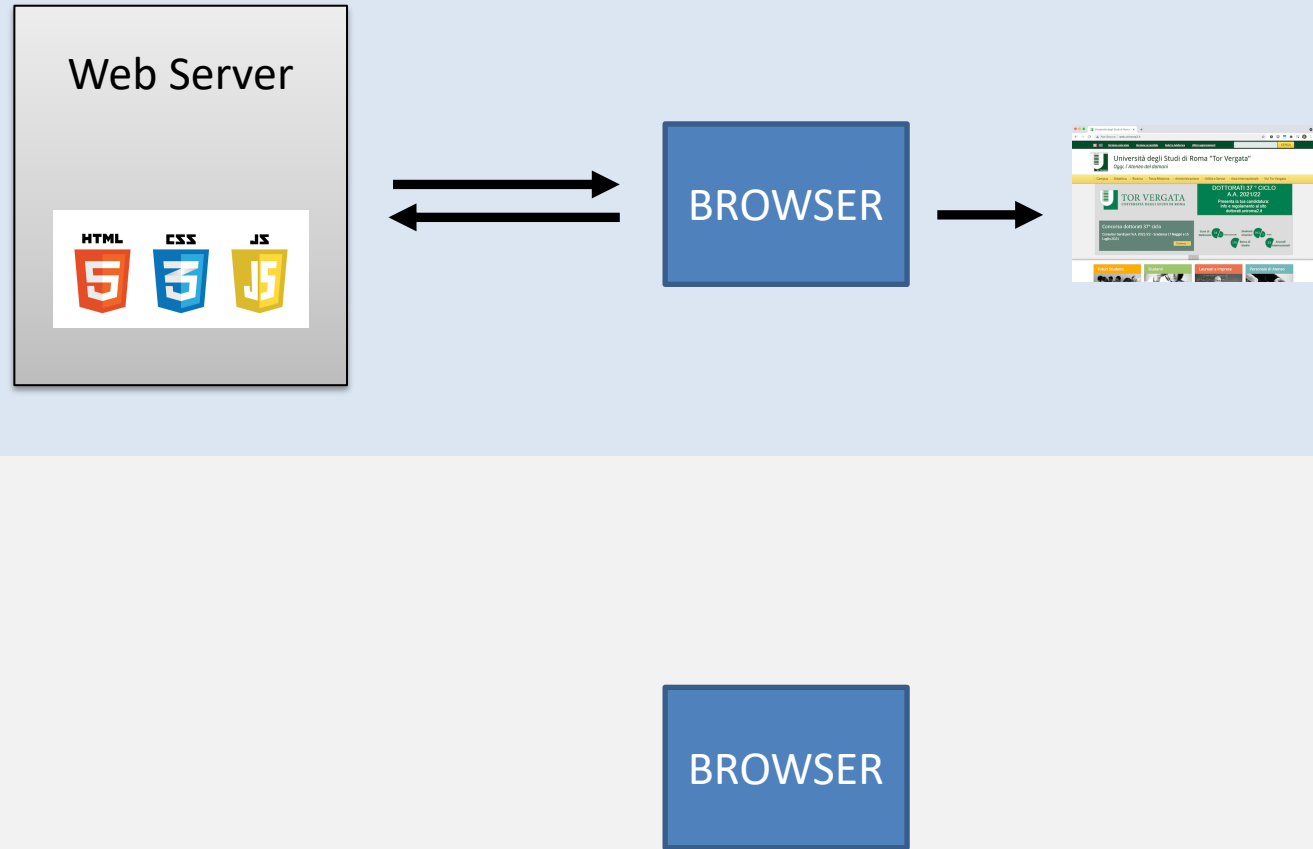


# Siti statici vs dinamici

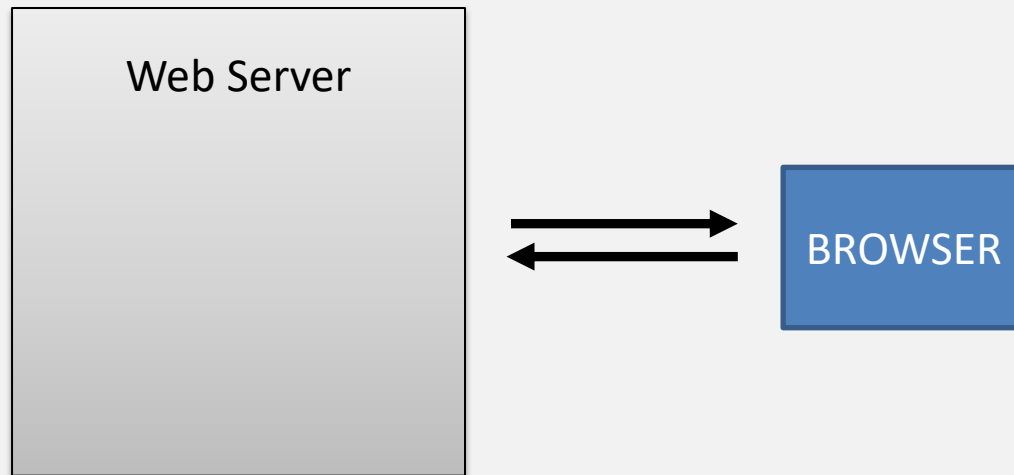
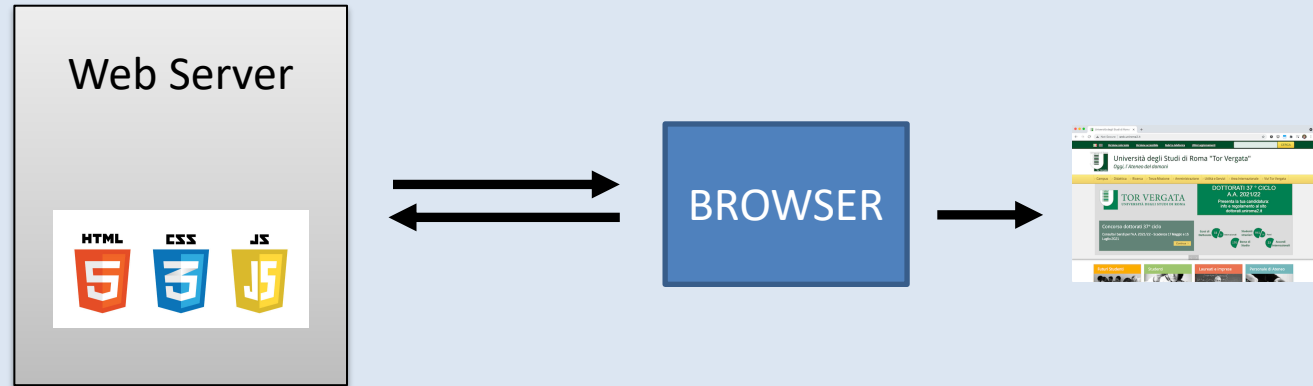




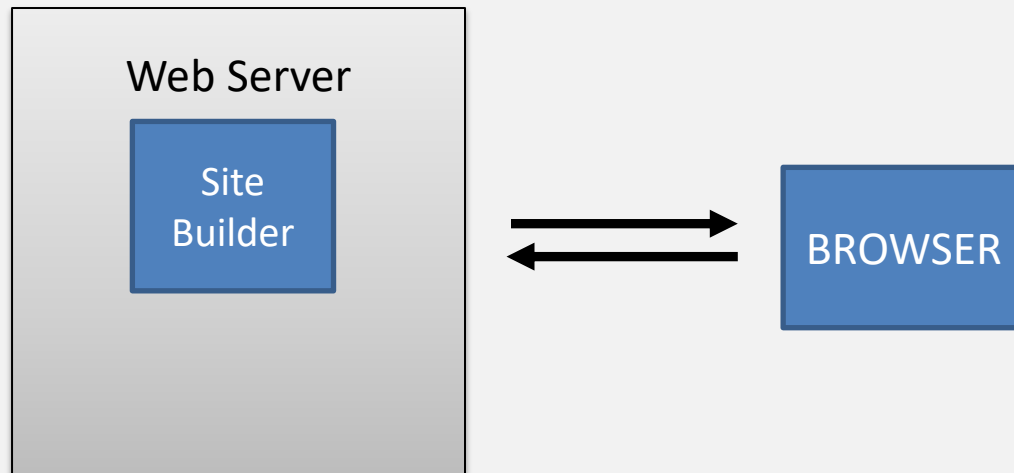
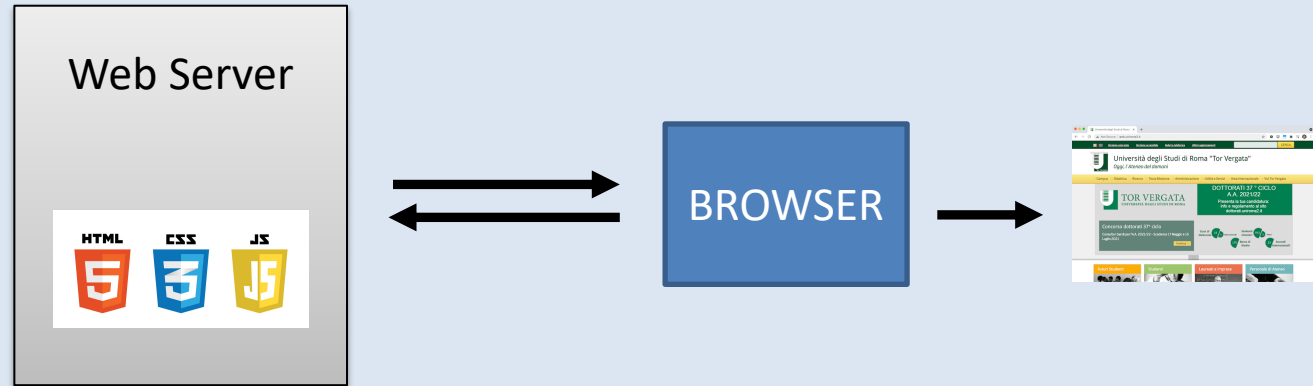
# Siti statici vs dinamici



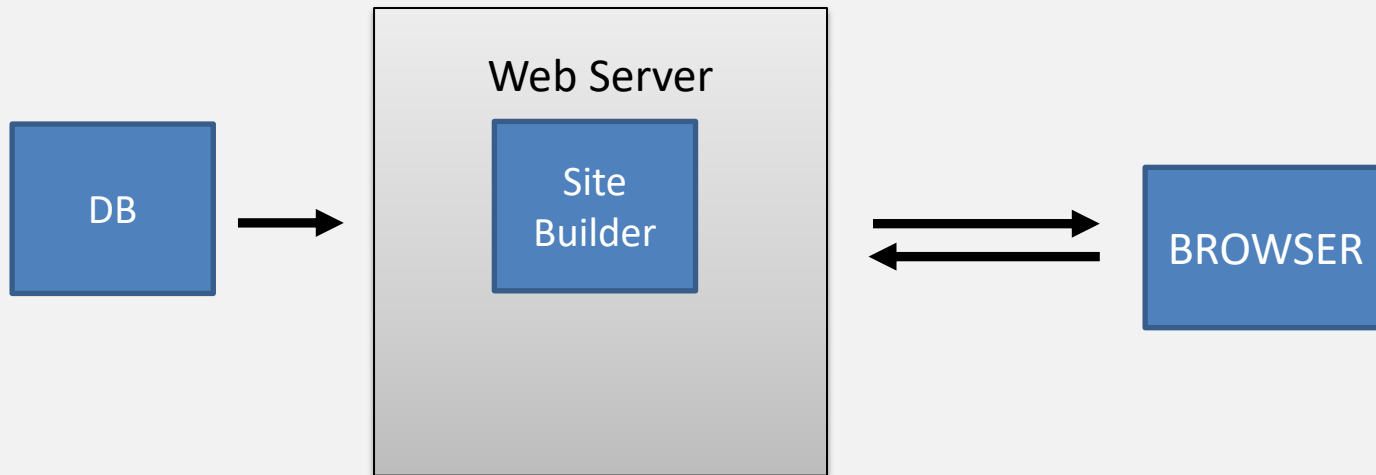
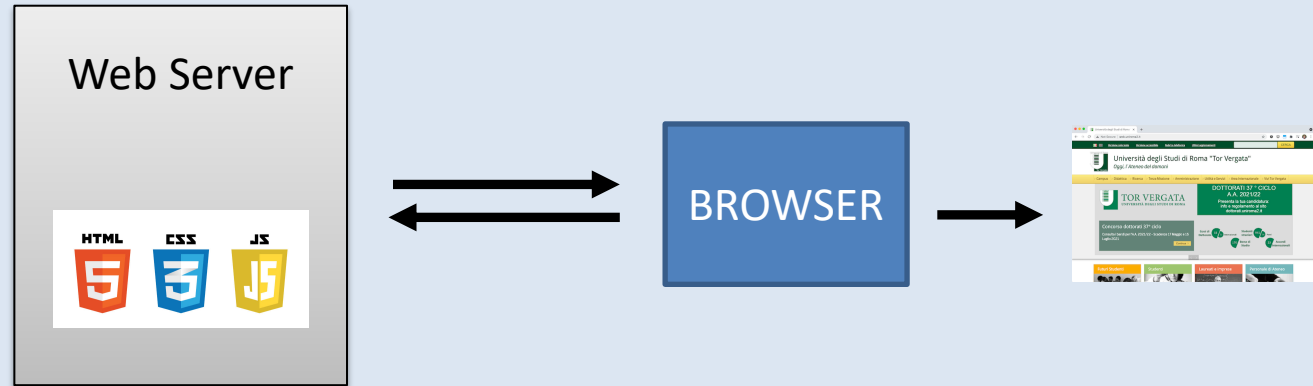
# Siti statici vs dinamici



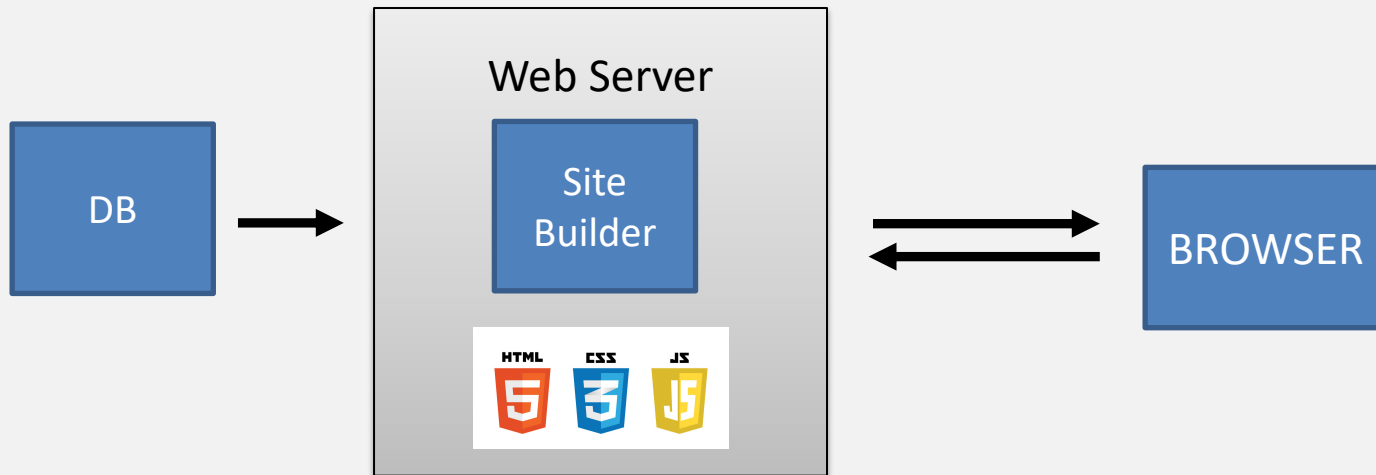
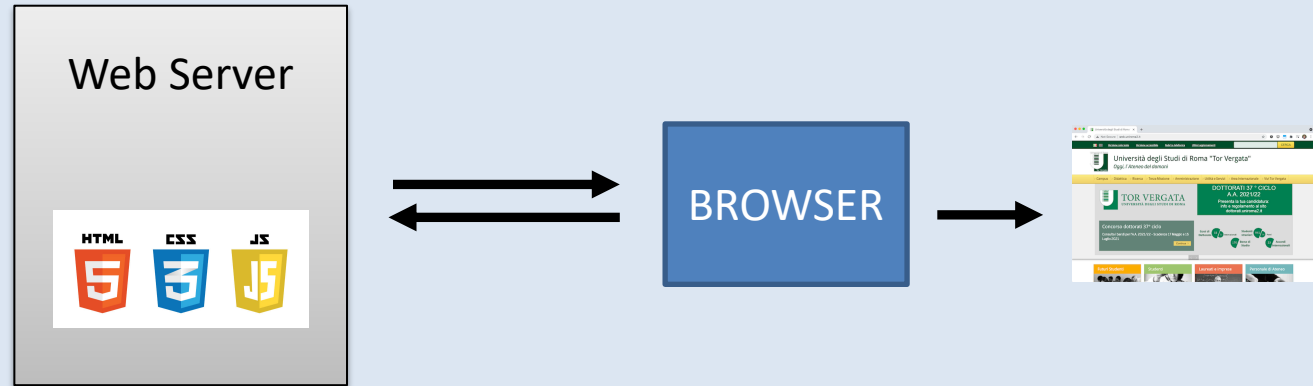
# Siti statici vs dinamici



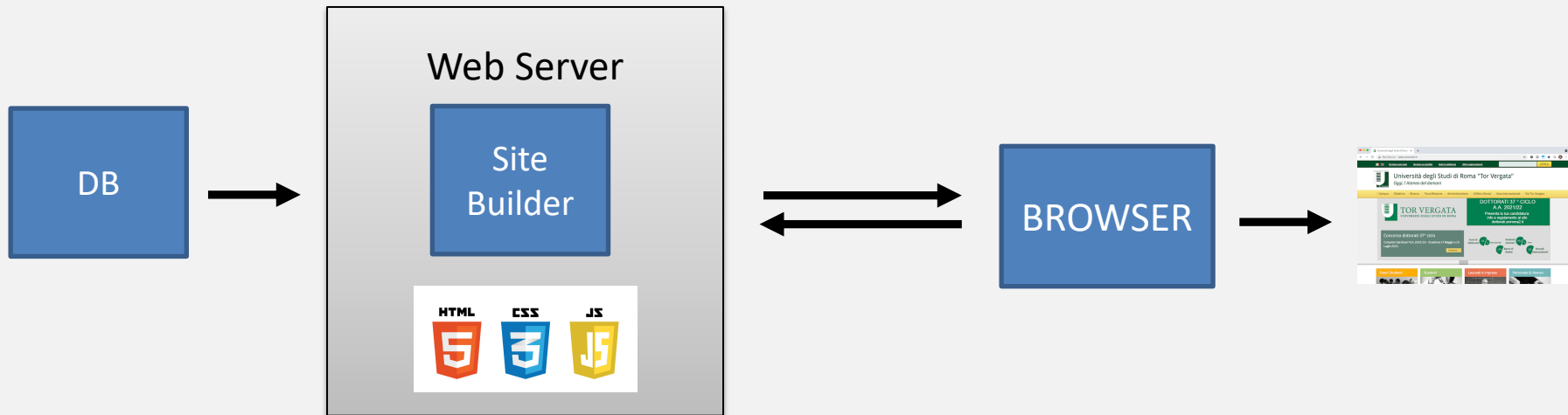
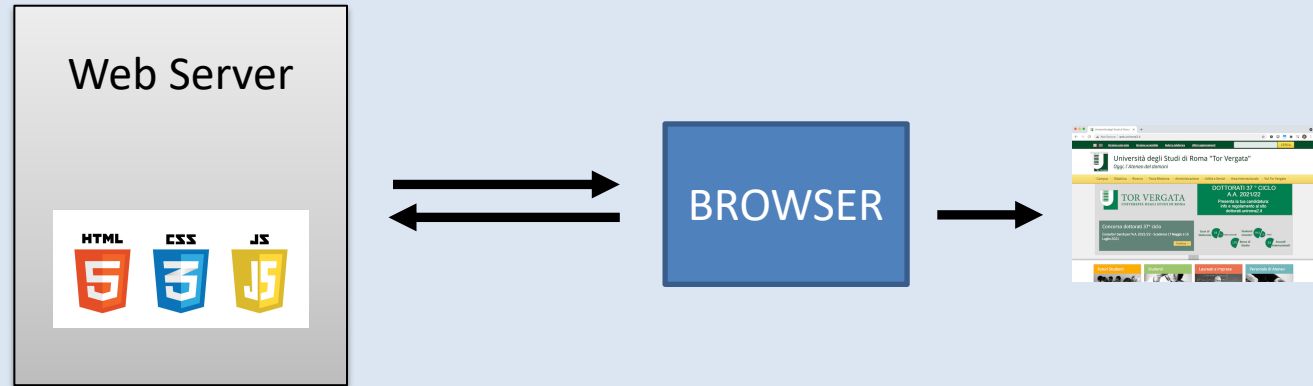
# Siti statici vs dinamici



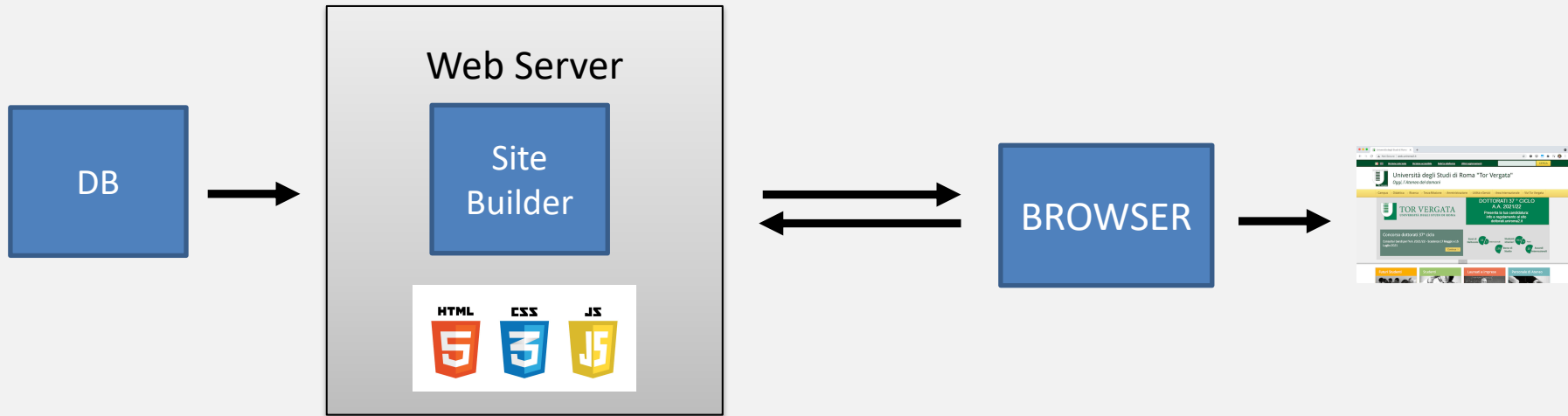
# Siti statici vs dinamici



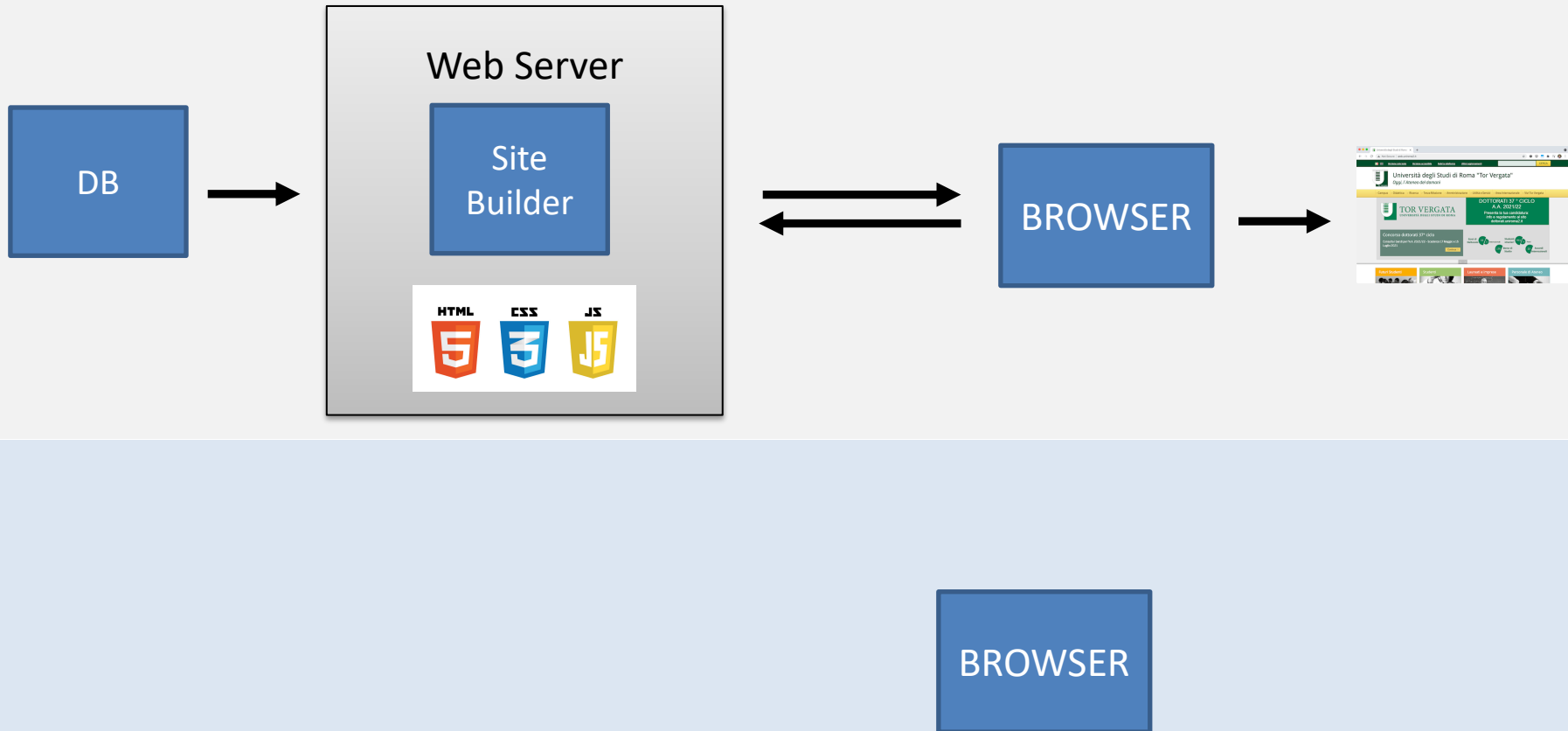
# Siti statici vs dinamici



# Siti dinamici vs API based

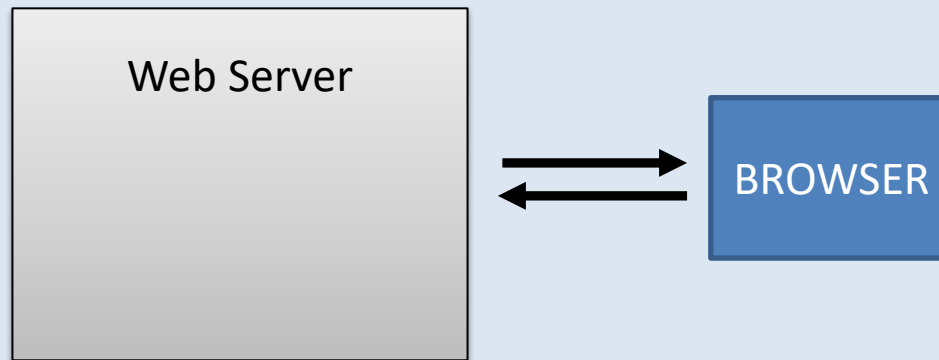
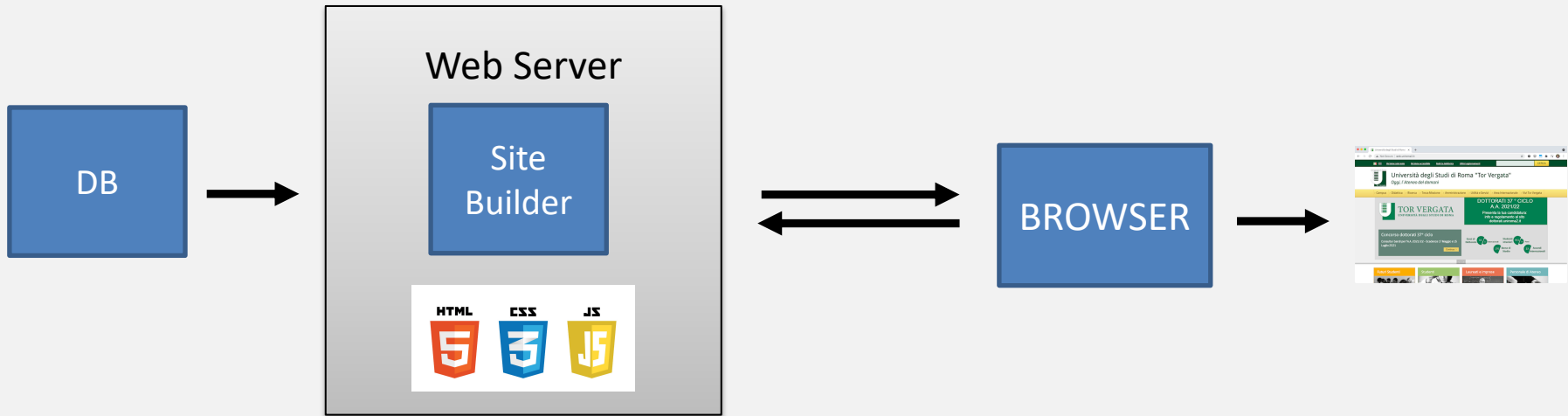


# Siti dinamici vs API based

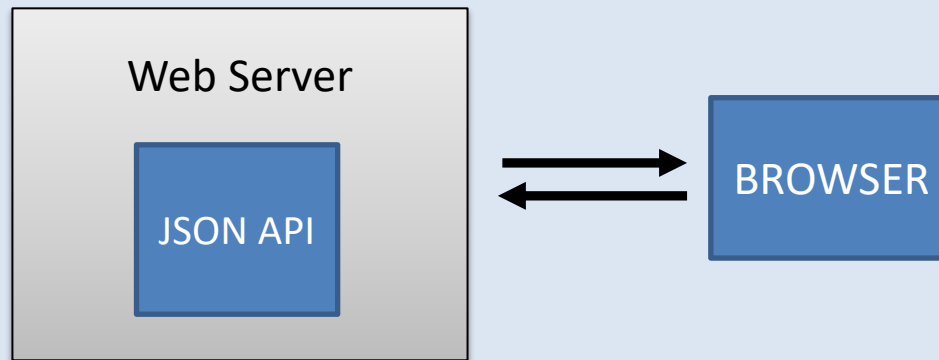
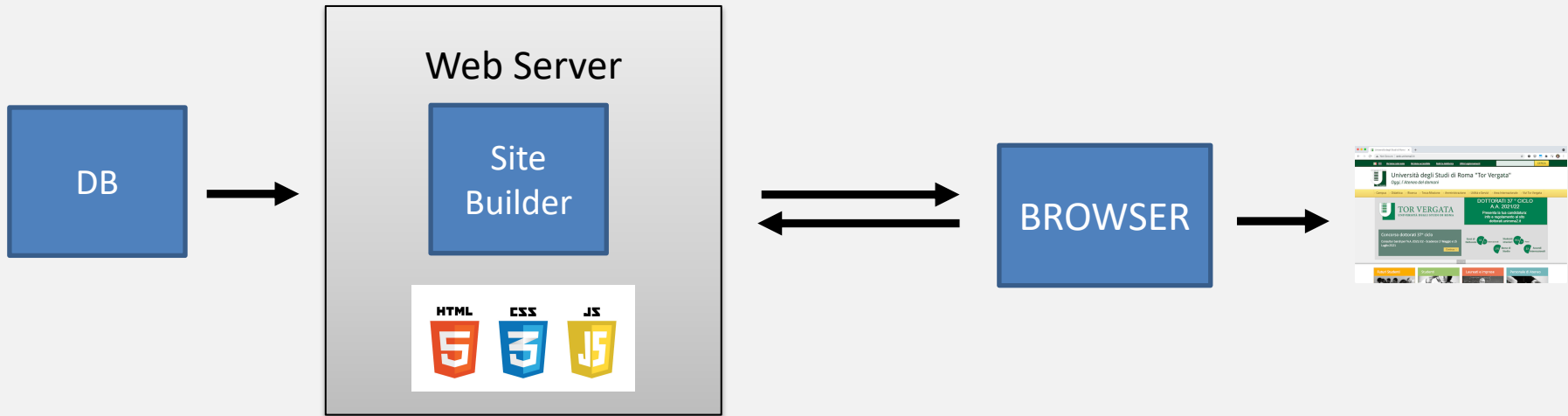




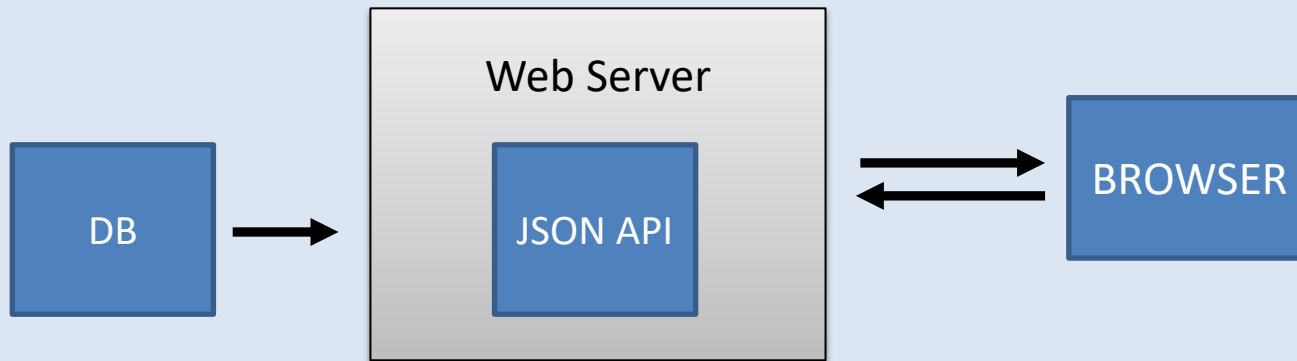
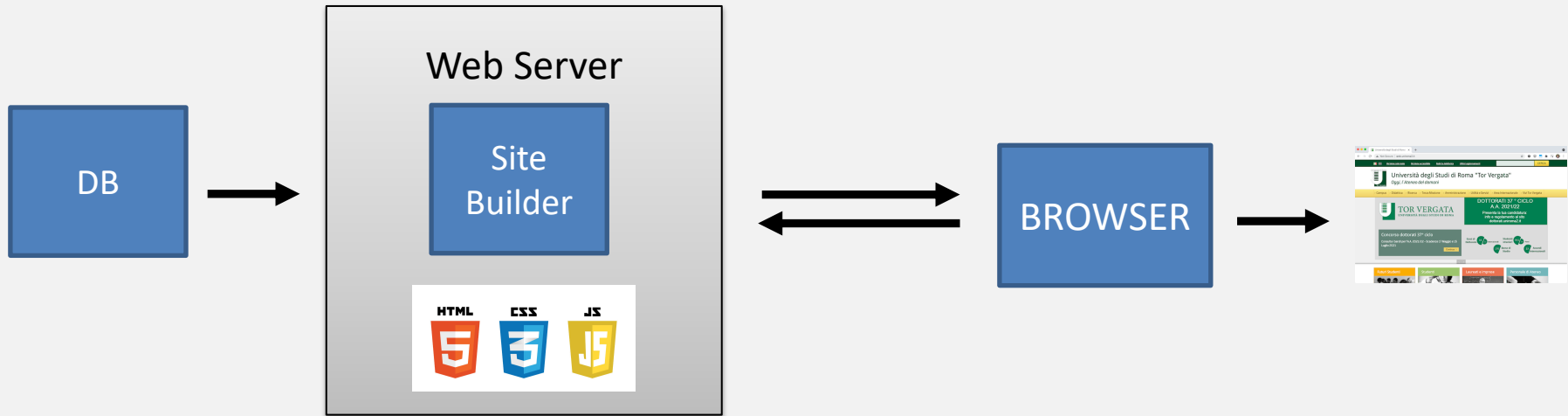
# Siti dinamici vs API based



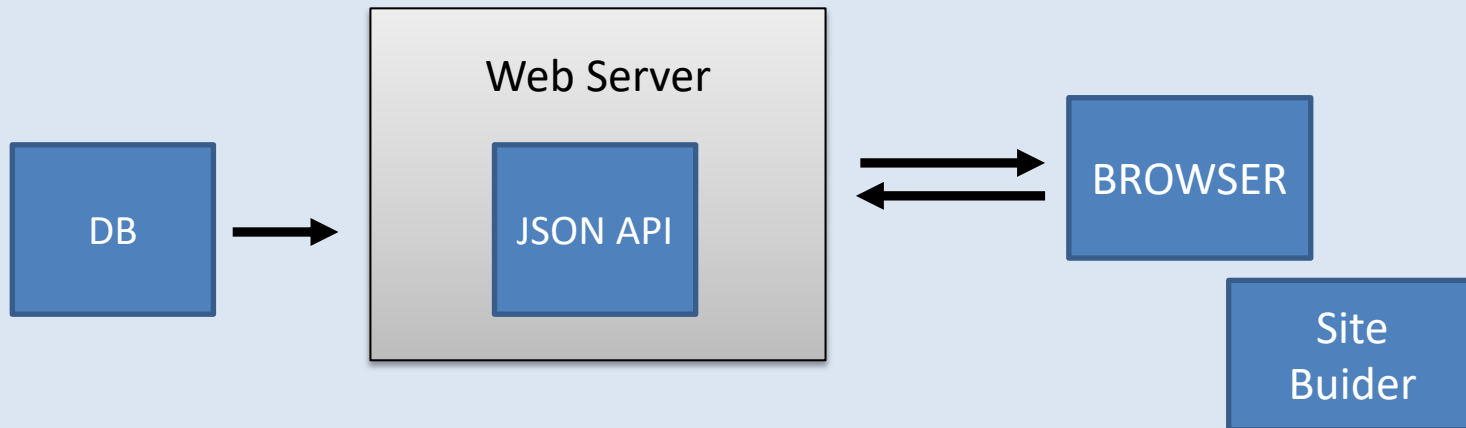
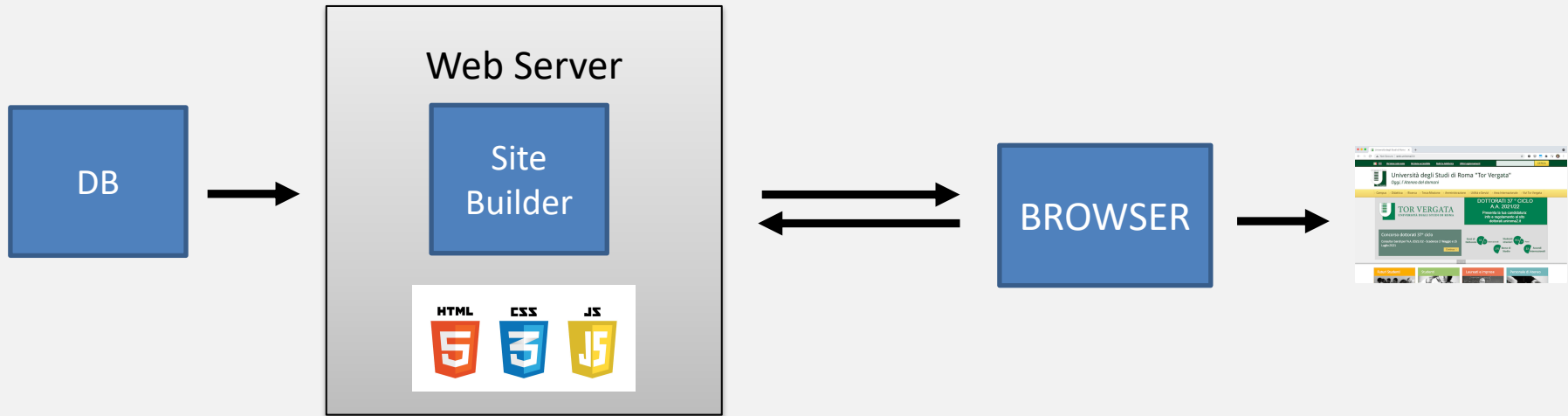
# Siti dinamici vs API based



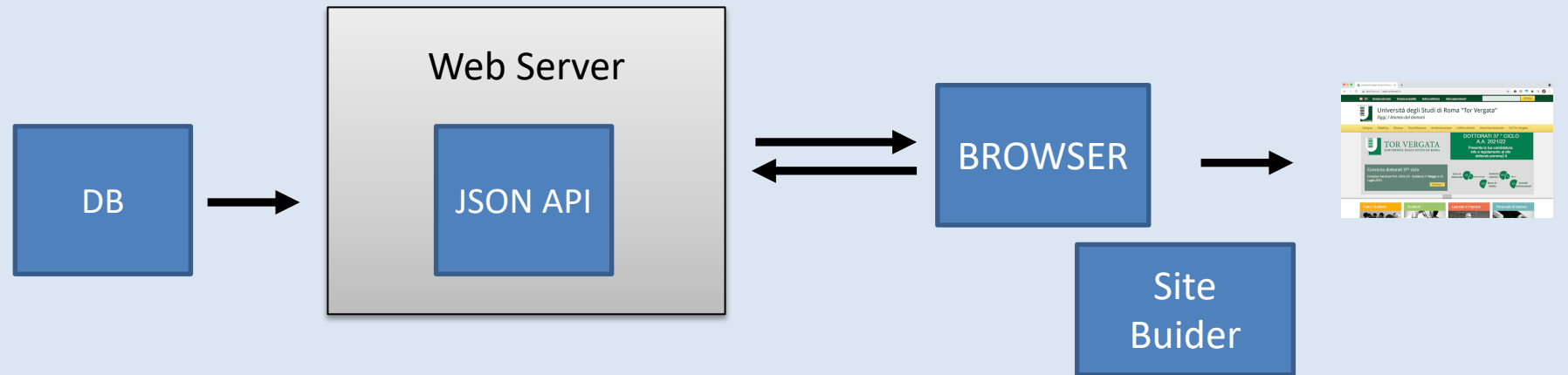
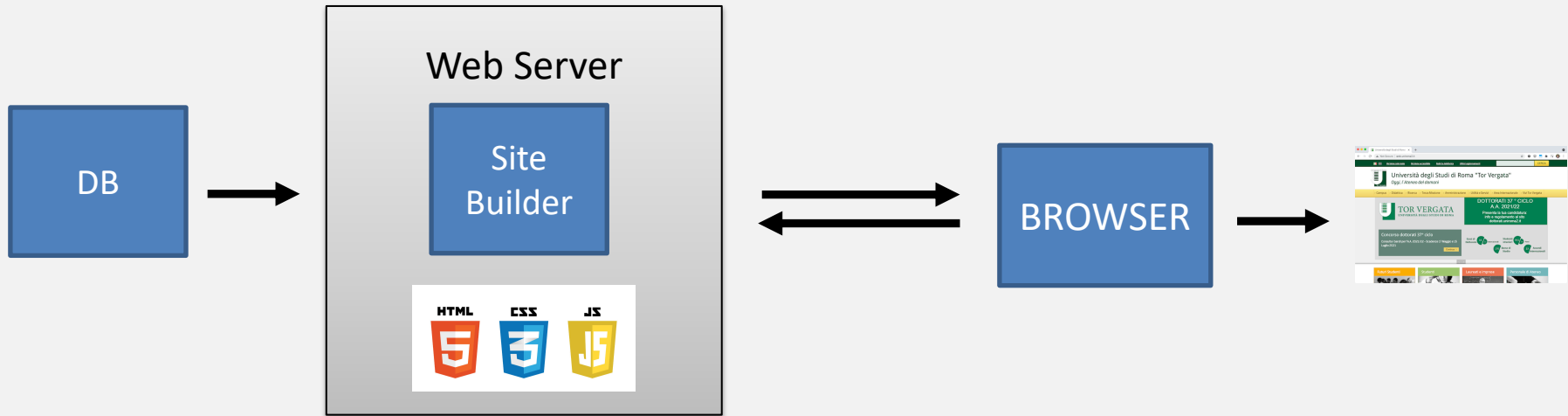
# Siti dinamici vs API based



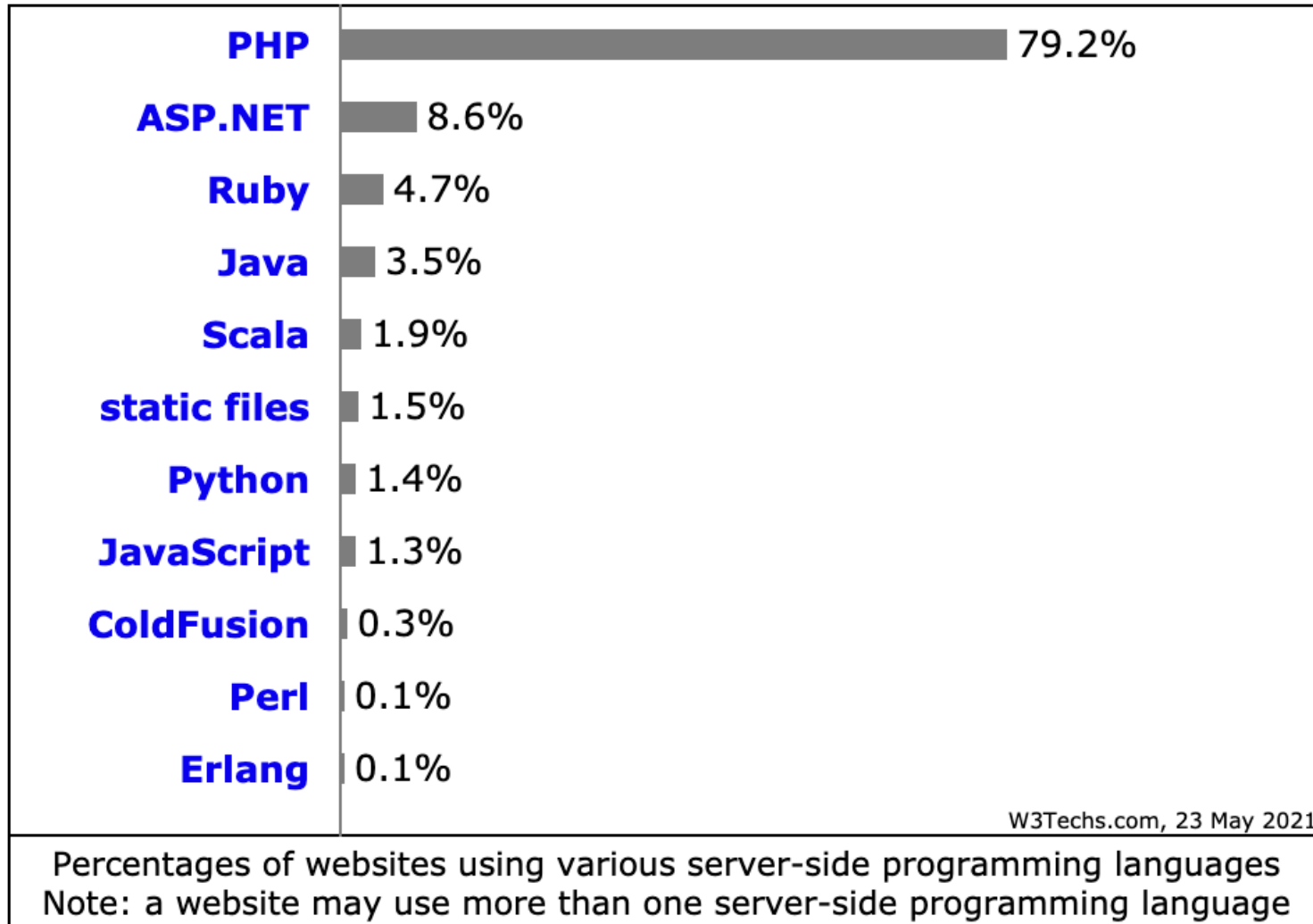
# Siti dinamici vs API based

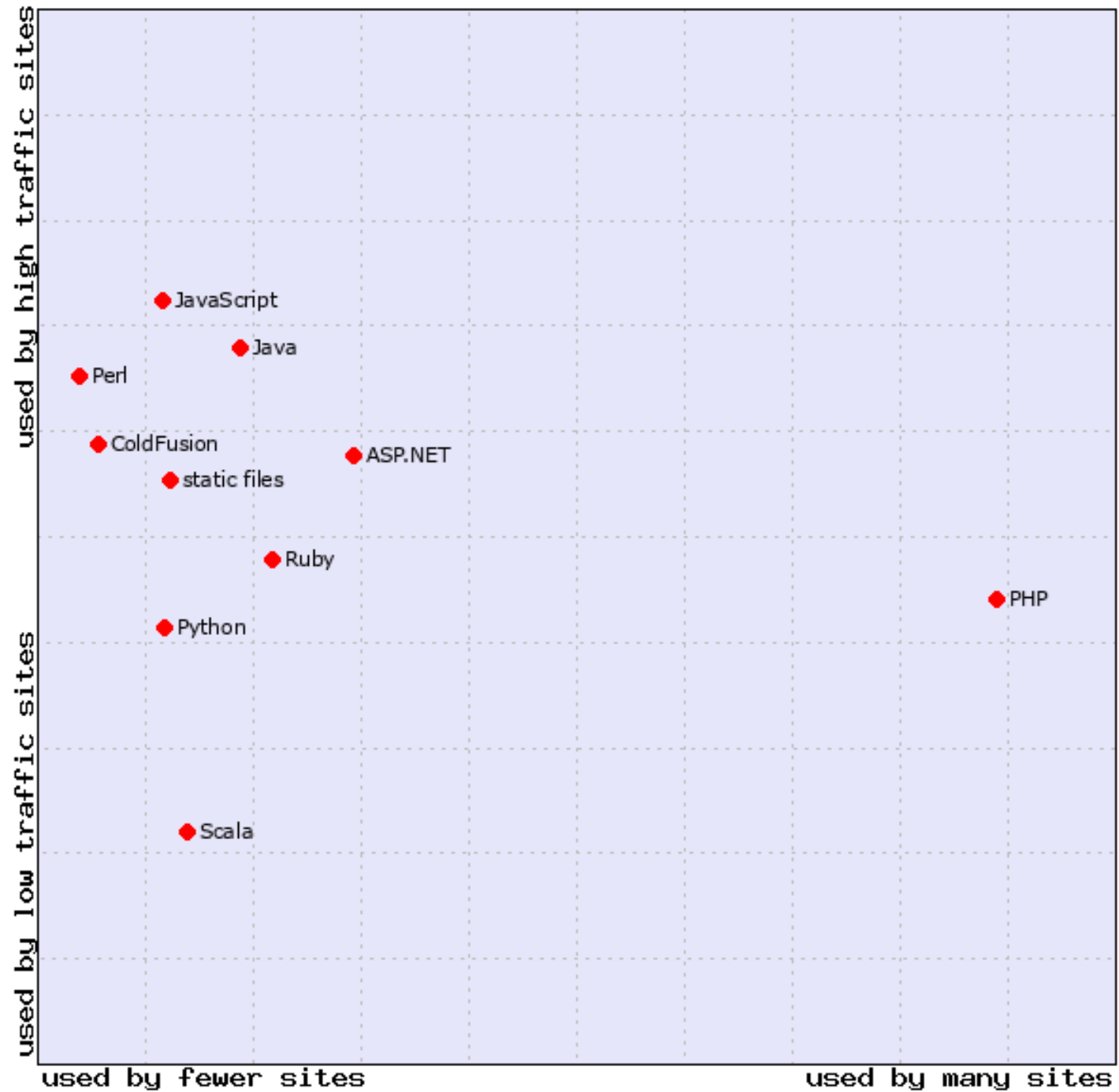


# Siti dinamici vs API based



# Tecnologie per backend

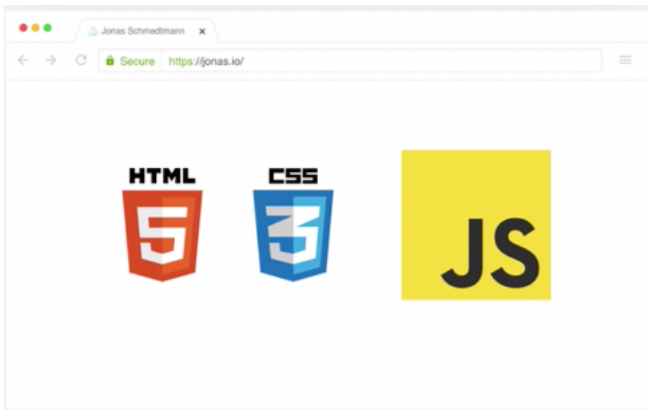




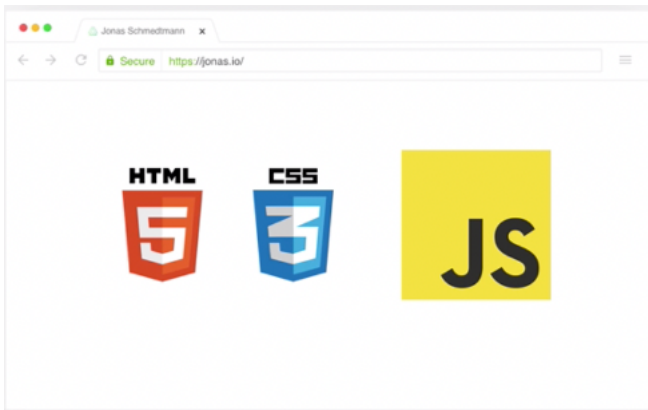
# NodeJS



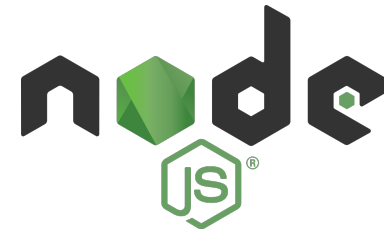
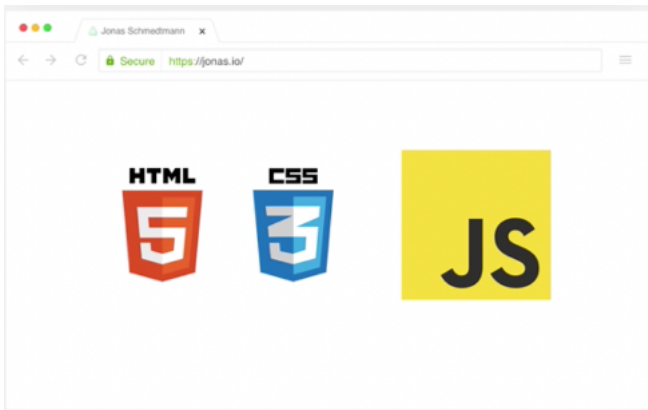
# node.js



# node.js



# node.js



- Può essere considerato come un ambiente runtime per JavaScript costruito sopra il motore V8 di Google.
- Ci fornisce un contesto dove possiamo scrivere codice JavaScript su qualsiasi piattaforma dove Node.js può essere installato
- L'ambiente ideale dove usare node.js è il server

# Architettura node.js



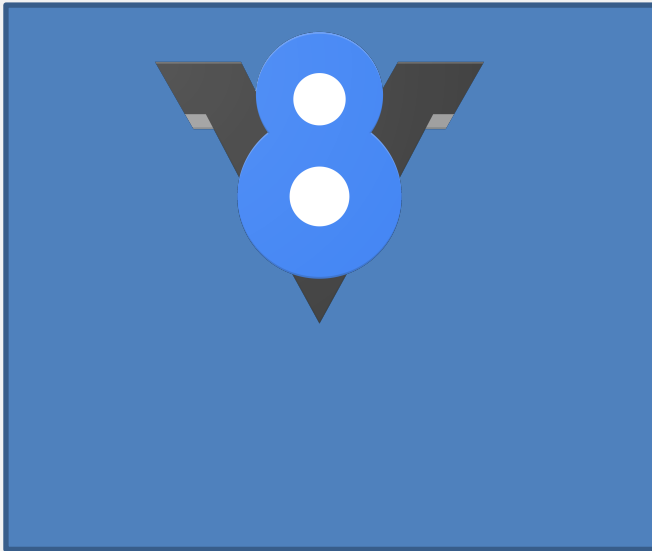
# Architettura node.js



# Architettura node.js



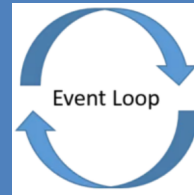
# Architettura node.js



# Architettura node.js



libuv



http-parser

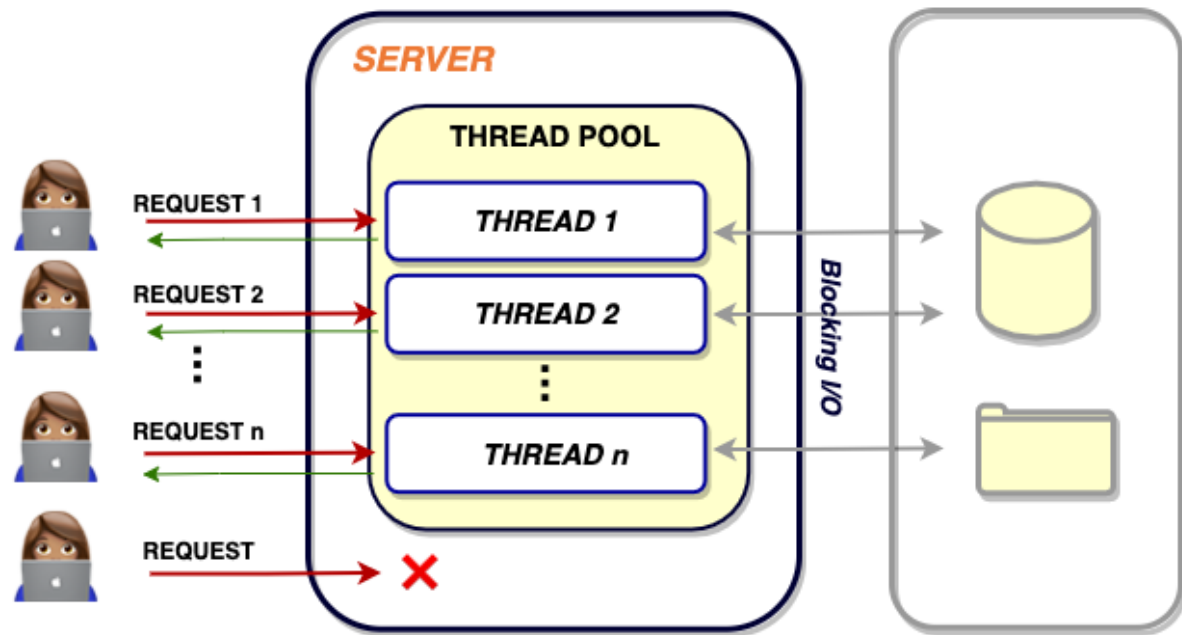
z-lib

OpenSSL

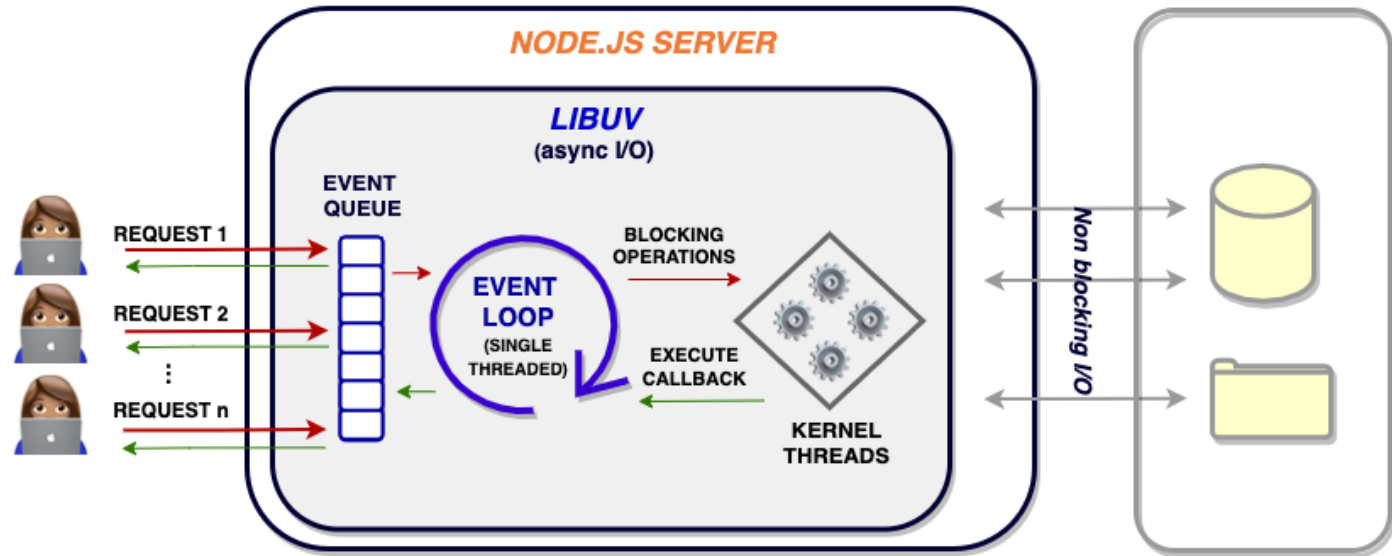
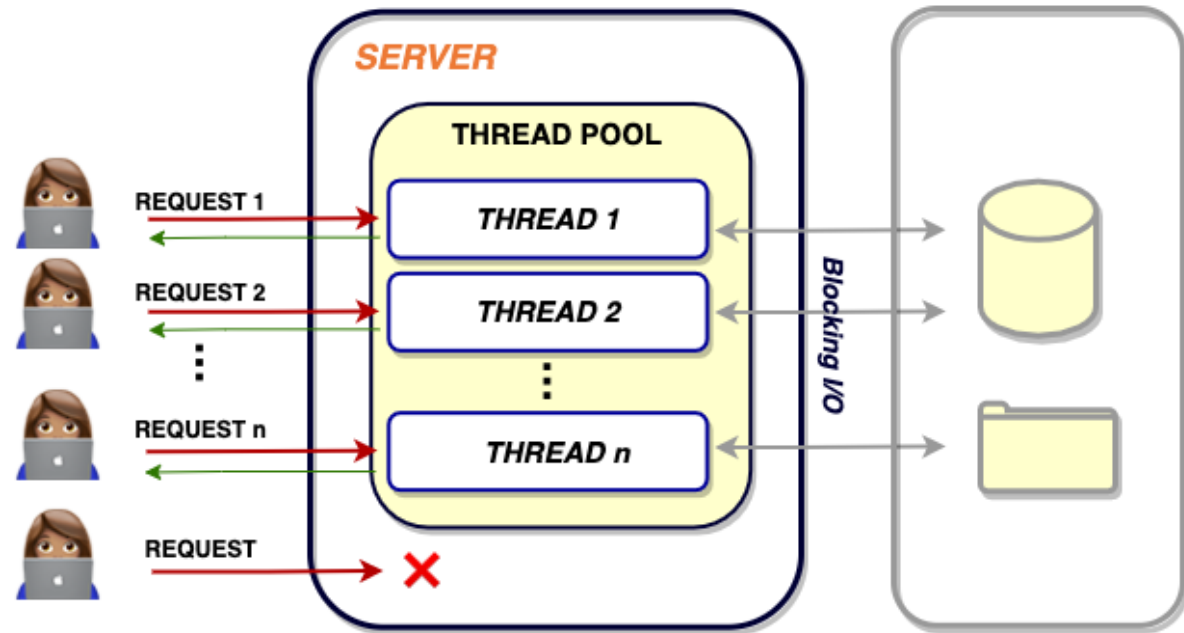
c-ares



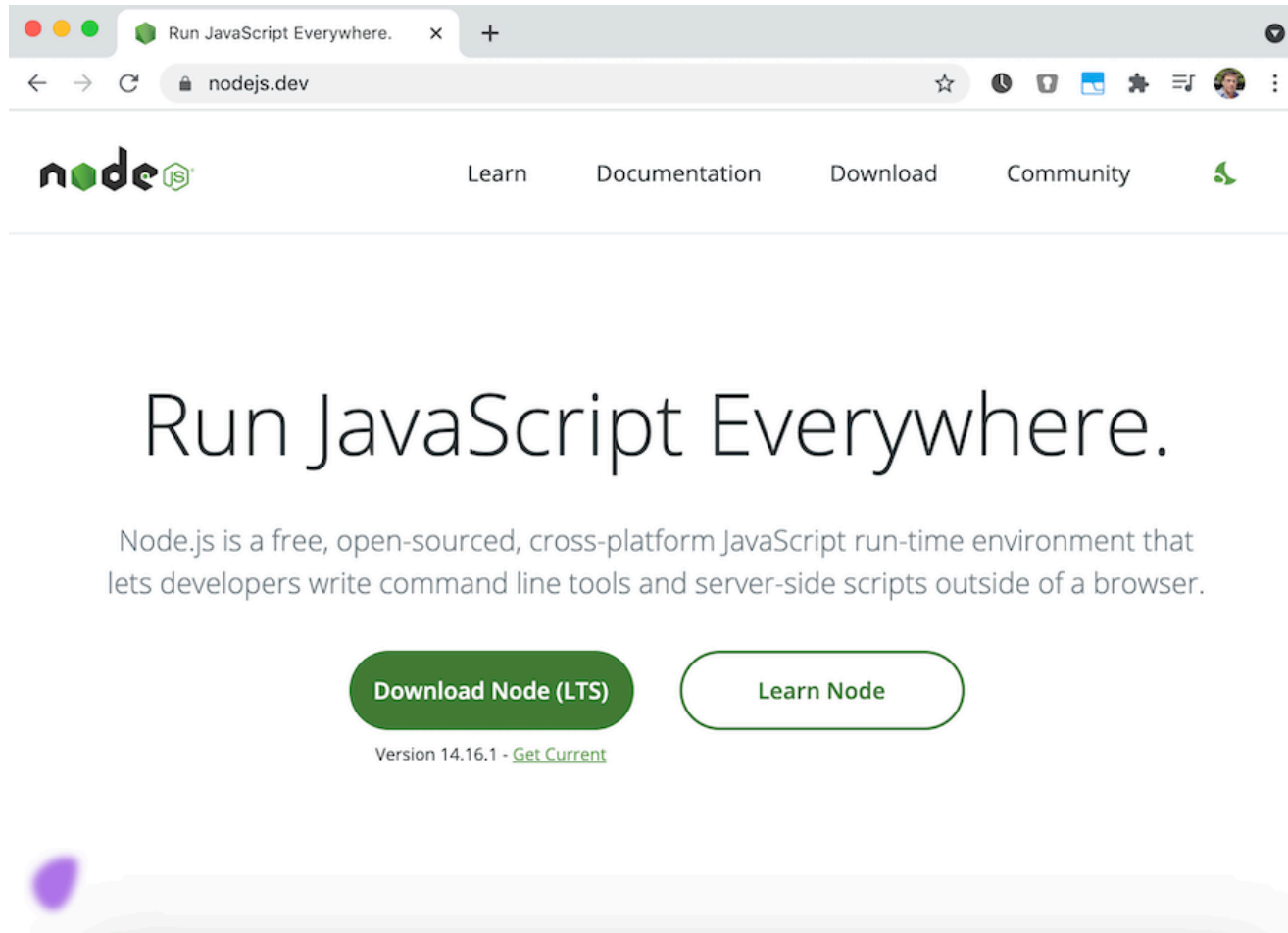
# Architettura nodejs



# Architettura nodejs



# Installazione



# node hello world!!

JS es1.js



```
1 console.log('Hello World!!');
```

```
2
```

TERMINAL

PROBLEMS

OUTPUT

DEBUG CONSOLE

```
(base) PPL3:basic loreti$ node es1.js
```

```
Hello World!!
```

```
(base) PPL3:basic loreti$
```

# Moduli

- A reusable piece of code that encapsulates implementation detail
- Modules can load each other and use special directives to export and import functionality
- Moduli in JS
  - AMD – one of the most ancient module systems, initially implemented by the library [require.js](#).
  - CommonJS – the module system created for Node.js server.
  - ES6 Moduls - language-level module system appeared in the standard in 2015

# Node Module

- Core Modules
  - di sistema installati con node
- Local Modules
  - Li creiamo localmente
- Third Party Modules
  - li dobbiamo installare (con npm)

# Core Module

Core Module	Description
<a href="#"><u>http</u></a>	http module includes classes, methods and events to create Node.js http server.
<a href="#"><u>url</u></a>	url module includes methods for URL resolution and parsing.
<a href="#"><u>querystring</u></a>	querystring module includes methods to deal with query string.
<a href="#"><u>path</u></a>	path module includes methods to deal with file paths.
<a href="#"><u>fs</u></a>	fs module includes classes, methods, and events to work with file I/O.
<a href="#"><u>util</u></a>	util module includes utility functions useful for programmers.

# Esempio fs

JS es2.js ●

```

1  const fs = require('fs');
2
3  fs.readFile('./data/input.txt', 'utf-8', (err, data) => {
4    console.log('Async');
5    console.log(data);
6  });
7
8  const data = fs.readFileSync('./data/input.txt', 'utf-8');
9  console.log(data);
10 console.log('----\n');
```



# Esempio web server

```

1  const http = require('http');
2
3  const server = http.createServer((req, res) => {
4      const pathName = req.url;
5      if (pathName === '/' || pathName === '/home') {
6          res.end('Home page');
7      } else if (pathName === '/contatti') {
8          res.end('Contatti');
9      } else {
10         res.writeHead(404, {
11             'Content-type': 'text/html',
12         });
13         res.end('<h1>404 - Page Not foud</h1>');
14     }
15 });
16
17 const port = 8000;
18
19 server.listen(port, '127.0.0.1', () => {
20     console.log(`Server listening on port ${port}`);
21 });

```

# Routing

```
const pathName = req.url;
if (pathName === '/' || pathName === '/home') {
  res.end('Home page');
} else if (pathName === '/contatti') {
  res.end('Contatti');
} else if (pathName === '/info') {
  res.end('Info Page');
} else if (pathName === '/api') {
  res.writeHead(404, {
    'Content-type': 'application/json',
  });
  res.end('Info Page');
} else {
  res.writeHead(404, {
    'Content-type': 'text/html',
  });
  res.end('<h1>404 - Page Not found</h1>');
}
```

Per **Routing** si intende determinare come un'applicazione risponde a una richiesta client a un endpoint particolare, il quale è un URI (o percorso) e un metodo di richiesta HTTP specifico (GET, POST e così via).

# Creare moduli

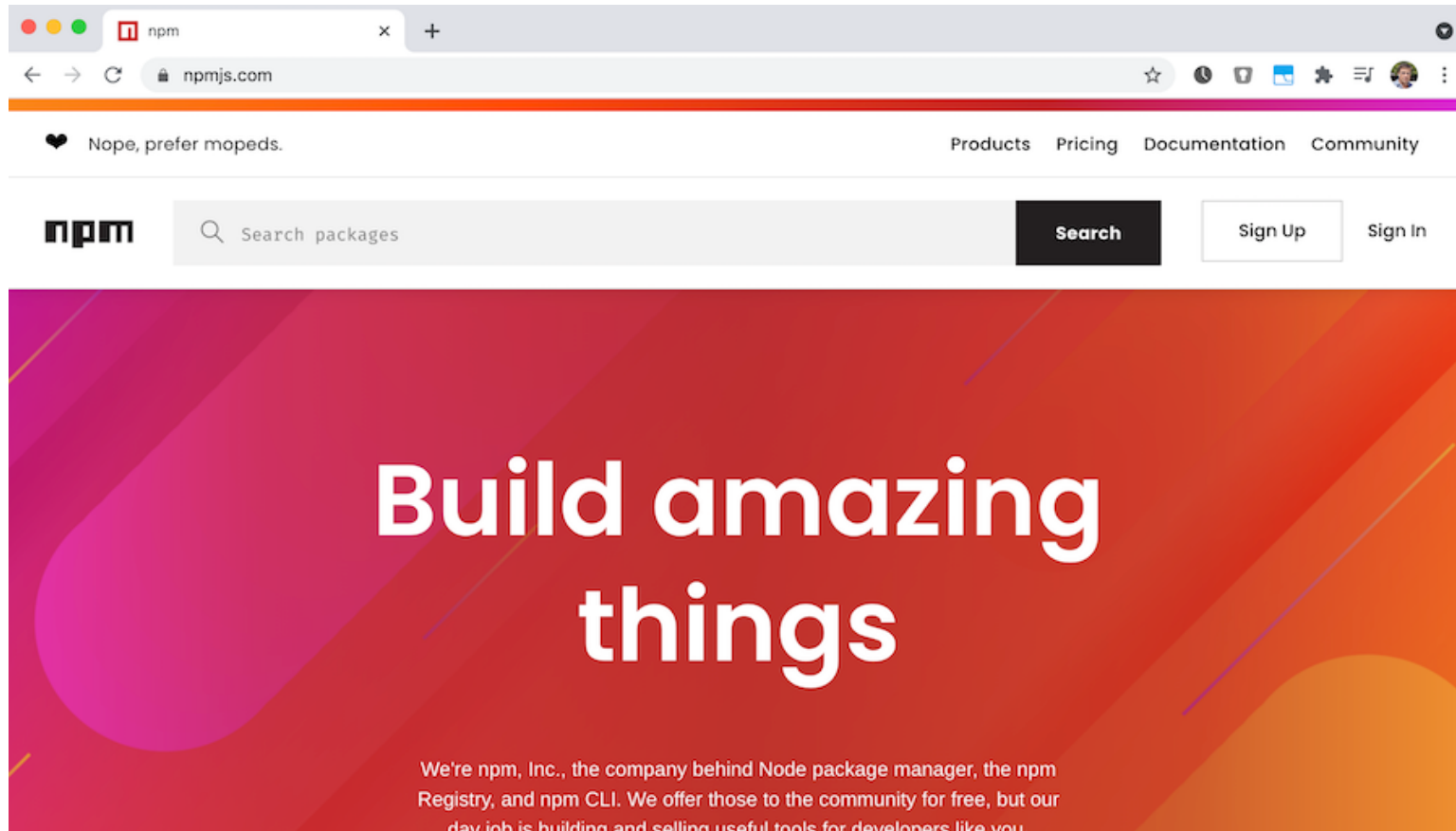
```

1  const fs = require('fs');
2
3  const data = JSON.parse(
4    fs.readFileSync(`${__dirname}/data/data.json`, 'utf-8')
5  );
6
7  console.log(data);
8  exports.getAll = function () {
9    return JSON.stringify(data);
10 };
11
12 exports.getItem = function (index) {
13   return JSON.stringify(data.find((el) => el.id == index));
14 };

```

```
const { getAll, getItem } = require('./lista');
```

# Import terze parti e npm



# Comandi base npm

# install

npm i <package>

# uninstall

npm un <package>

# update

npm up <package>

# init

npm init

# run

npm run <script>

# Versions

Code status	Stage	Rule	Example version
First release	New product	Start with 1.0.0	1.0.0
Backward compatible bug fixes	Patch release	Increment the third digit	1.0.1
Backward compatible new features	Minor release	Increment the middle digit and reset last digit to zero	1.1.0
Changes that break backward compatibility	Major release	Increment the first digit and reset middle and last digits to zero	2.0.0

# update

- Patch releases: 1.0 or 1.0.x or ~1.0.4
- Minor releases: 1 or 1.x or ^1.0.4
- Major releases: \* or x

```
"dependencies": {
  "my_dep": "^1.0.0",
  "another_dep": "~2.2.0"
},
```

# Resolving and loading

- Risolvere il path e decidere il modulo da caricare

## 1. **Core Module**

## 2. Se path inizia con './' o '../' -> **Developer Module**

- prima lo script con il nome
- poi la folder con dentro index.js

## 3. Entra in modules\_core e cerca gli **Installed Module**



# Wrapping

```
(function exports require module __filename __dirname {  
    // Module code lives here...  
});
```

- Il modulo viene "wrappato" in una IIFE ed eseguito
- **require**: funzione per importare moduli
- **module**: riferimento al modulo corrente
- **export**: riferimento a module.export
- **\_\_filename**: path assoluto del modulo
- **\_\_dirname**: path della dir del modulo

# Esecuzione, exports e caching

- Il codice del modulo viene eseguito
- La funzione require torna gli exports
- Il risultato dell'esecuzione è cachato e ritornato alle esecuzioni successive del modulo