FullStack #WebDevelopment Bootcamp

Course Week 2

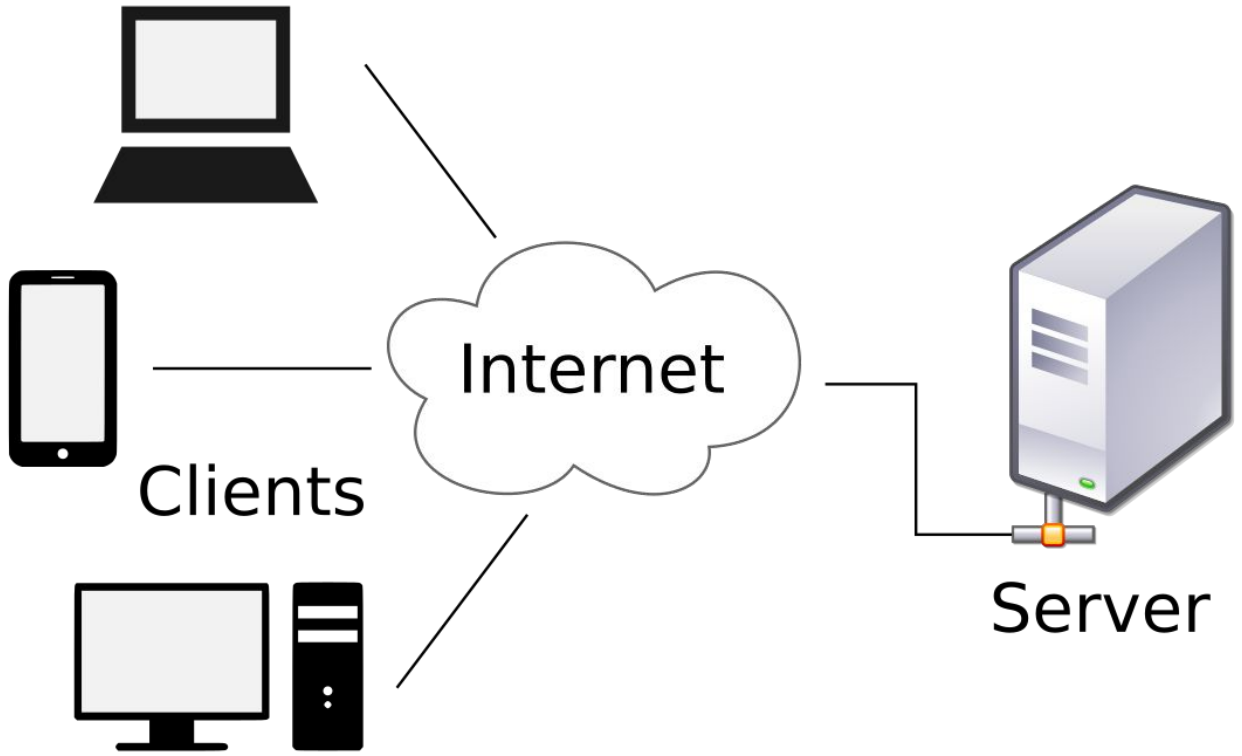
Content

- *Theory: Client Server Architecture*
- *Theory: HTTP(s) and other types of Web Communication (WS, WebRTC)*
- *Theory: Dependency Injection*
- *Theory: Our Application Architecture*
- *Practical: JavaScript & TypeScript Introduction*
  - *Variables, Loops, Functions, Array & String Manipulation, Why do people hate it (== vs ===)?*
- *Practical: CSS Introduction*
- *Practical: Git*
  - *Clone/Create a repository*
  - *Commit & Push*
  - *Create Branch & Make a Pull/Merge Request (PR/MR)*

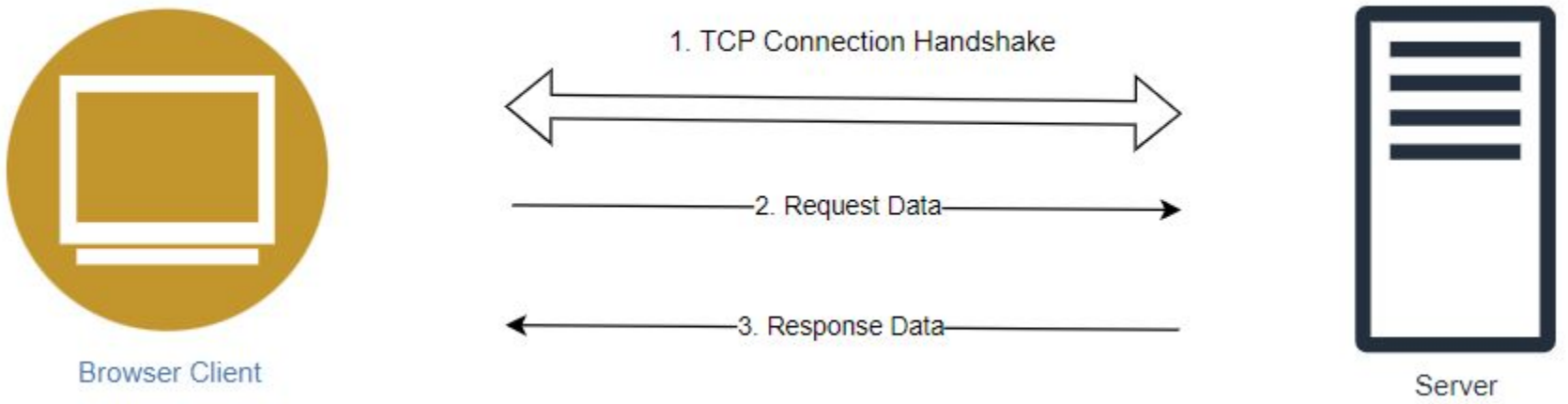
Client-Server Architecture

- The **most common** architecture in Web Development
- You have the following:
  - A “**Server**” which serves some kind of functionality (an API)
  - One or more **clients** which **communicate** with “**Server**” for that functionality

- A “Client” can be of different types (browser, mobile app, another server)
- Note that there are also other types of architectures: Peer-to-Peer



Communication: HTTP



- The most common Protocol in the Web
- Is built over **TCP** and has an **initial handshake activity** (send and receive acknowledgement signals)
- It has a secure version called HTTPS using Server-Side Digital Certificates (SSL)
- This is a **unidirectional** connection (the server does not have knowledge of all the possible clients)

Communication: HTTP Request Structure

Request Line

Method: “GET”/ “POST”/ “PUT” / “DELETE”, “PATCH”, “HEAD”, “OPTIONS”
Target: “http://localhost:8080” / “https://www.google.com/search?q=http+communication”
Version

Headers

Authorization Token
Cookies
Security Headers
etc...

Body
Binary data / Simple Text Data / JSON data / Any other kind of encoding

Communication: HTTP Response Structure

Will be similar to the Request Structure with the addition of:

- **Status Code:** 200/401/404/etc

More info at: https://developer.mozilla.org/en-US/docs/Web/HTTP/Status

S.N.	Code and Description
1	**1xx: Informational** It means the request was received and the process is continuing.
2	**2xx: Success** It means the action was successfully received, understood, and accepted.
3	**3xx: Redirection** It means further action must be taken in order to complete the request.
4	**4xx: Client Error** It means the request contains incorrect syntax or cannot be fulfilled.
5	**5xx: Server Error** It means the server failed to fulfill an apparently valid request.

Communication: HTTP Request Example

Communication: Other types of protocols

WebSockets

- A **bidirectional** connection between a client and a server
- Has the “ws” or “wss” protocol instead of http
- Built over TCP
- Usually used when you want **the server** to **notify** clients of some changes
- **Example**: when a client made some modification to a resource, you want the other clients to see the resource change instantly without a refresh of the tab

WebRTC

- A realtime peer-to-peer communication protocol usually used for media streaming (video/voice/etc)
- Built over UDP (has also some implementations over TCP)
- **Example**: Browser Skype/Teams/Discord

Dependency Injection (DI): Inversion of Control

```
export class B {
    // functionality
}

export class A {
    constructor(private b: B) {}
    // use class B functionality
}
```

Dependency Injection (DI): Inversion of Control

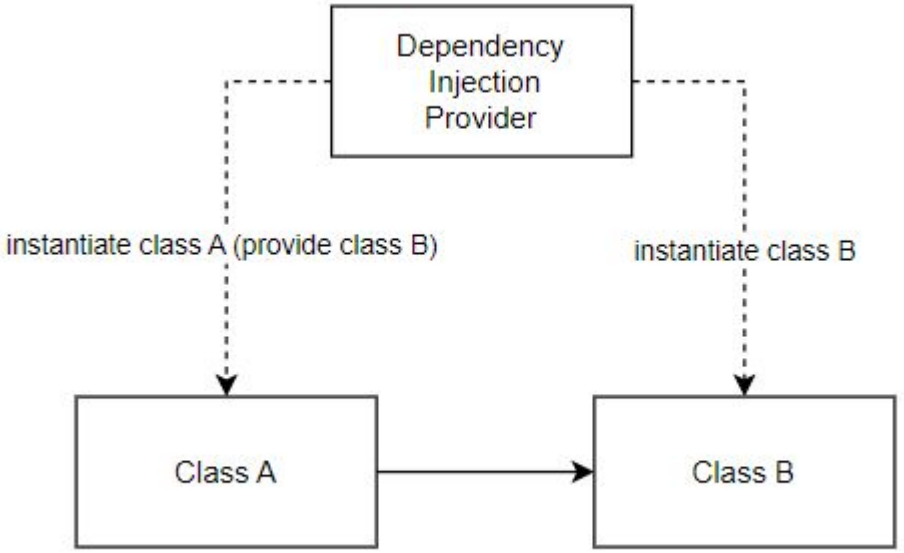
How is it done by default:

- We will need to instantiate class B
- Then pass it when you want to instantiate class A

```
const b = new B();
const a = new A(b);
```

This is extremely complex when you have to manage multiple dependencies for multiple classes

Dependency Injection (DI): Inversion of Control



Inversion of Control = a pattern in which you provide a “callback” (which “implement” and/or controls reaction), instead of acting directly

In other words: you redirect control to an external handler/controller

In our case: you redirect who instantiates our classes to a Dependency Injection Handler

Dependency Injection (DI): How we use it

Angular / Nest.js

```typescript
@Injectable()
export class B {
    // functionality
}

@Injectable()
export class A {
    constructor(private b: B) {}
    // use class B functionality
}
```

Spring (Java)

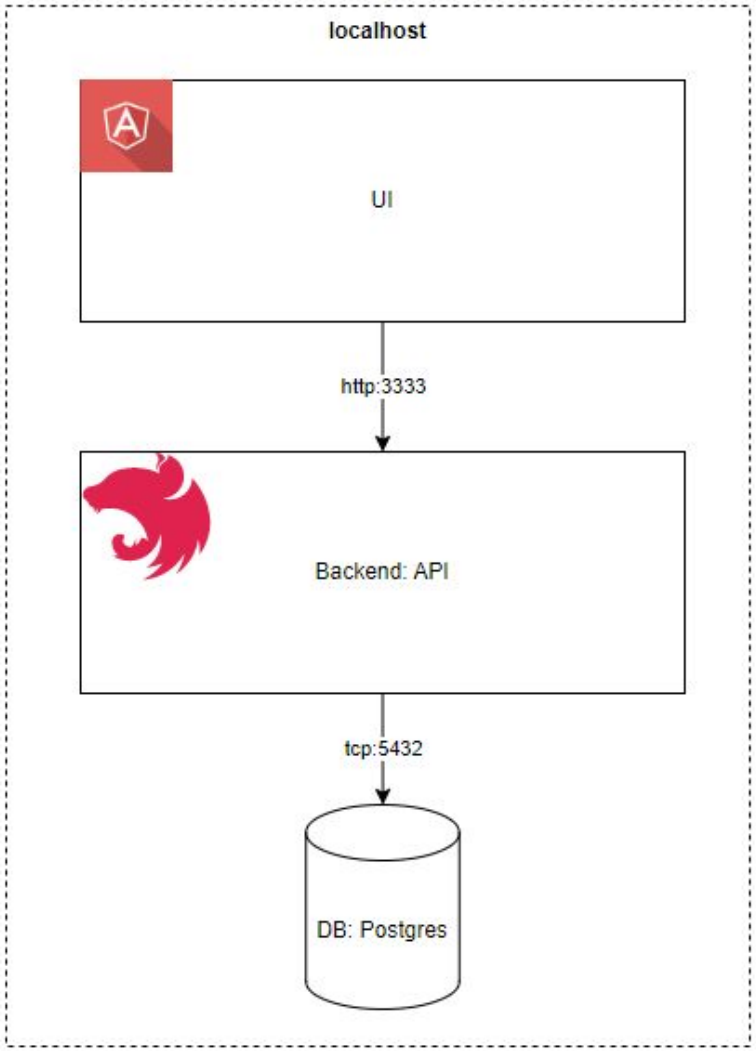
```java
public class UserService {

    @Autowired
    private UserRepository userRepository;
}
```

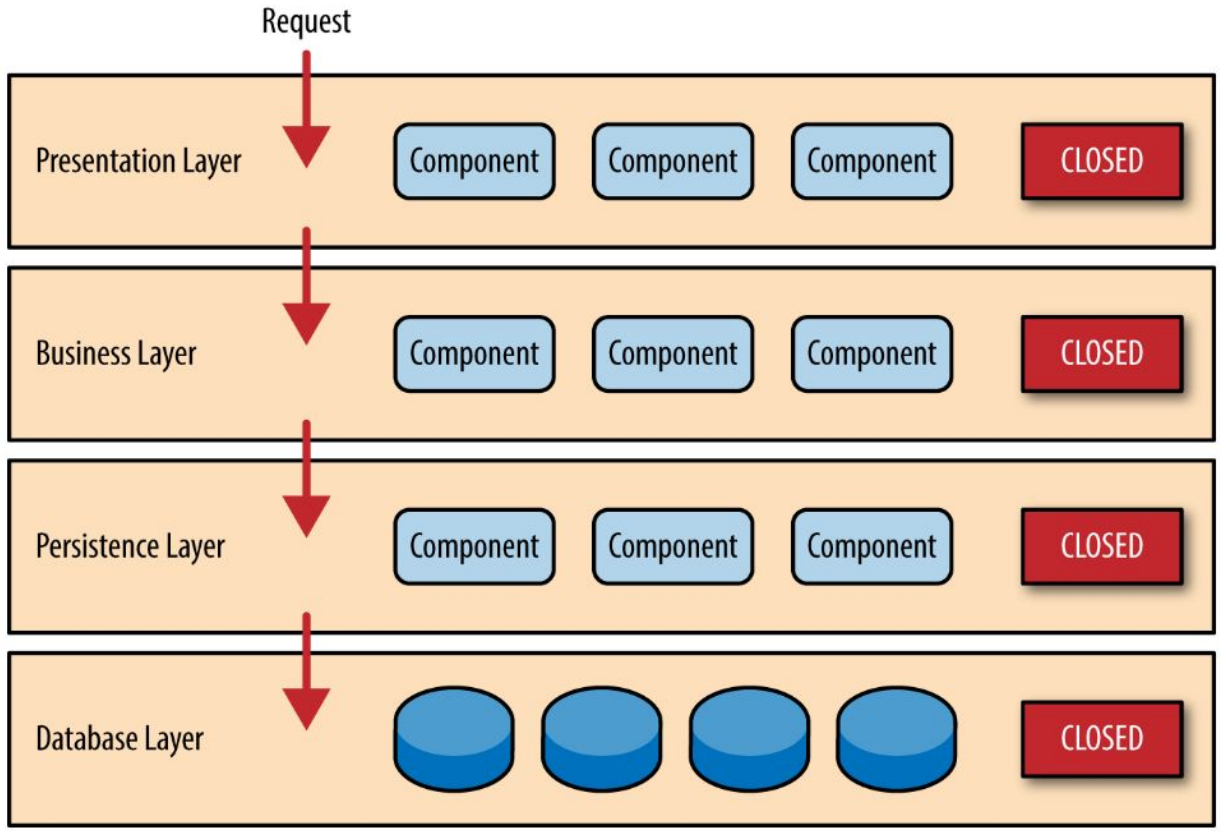
```java
@Component("fooFormatter")
public class FooFormatter {
    public String format() {
        return "foo";
    }
}
```

```java
@Component
public class FooService {
    @Autowired
    private FooFormatter fooFormatter;
}
```

Our Application Architecture

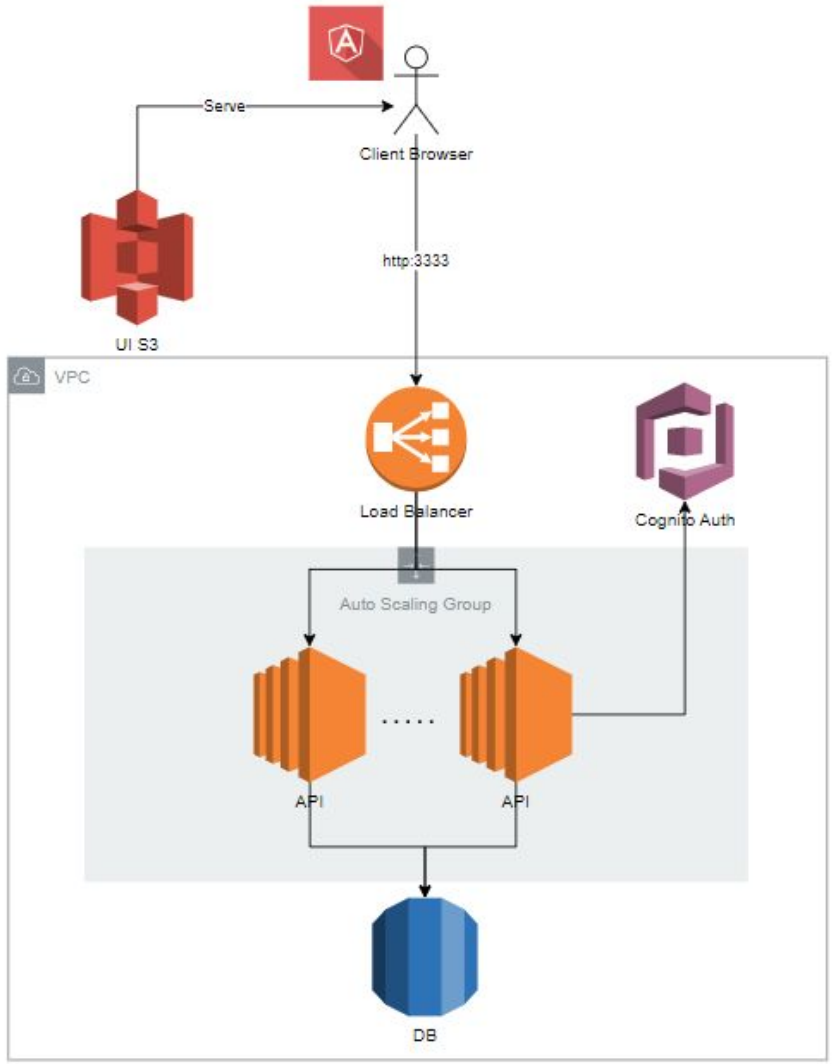


Backend Layers Architectural Pattern



https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch01.html

Our Application Architecture in Real-Life Example (AWS)



- Scale based on user traffic
- Use a third-party authentication system
- Serve your UI from a service
- Use a managed database system