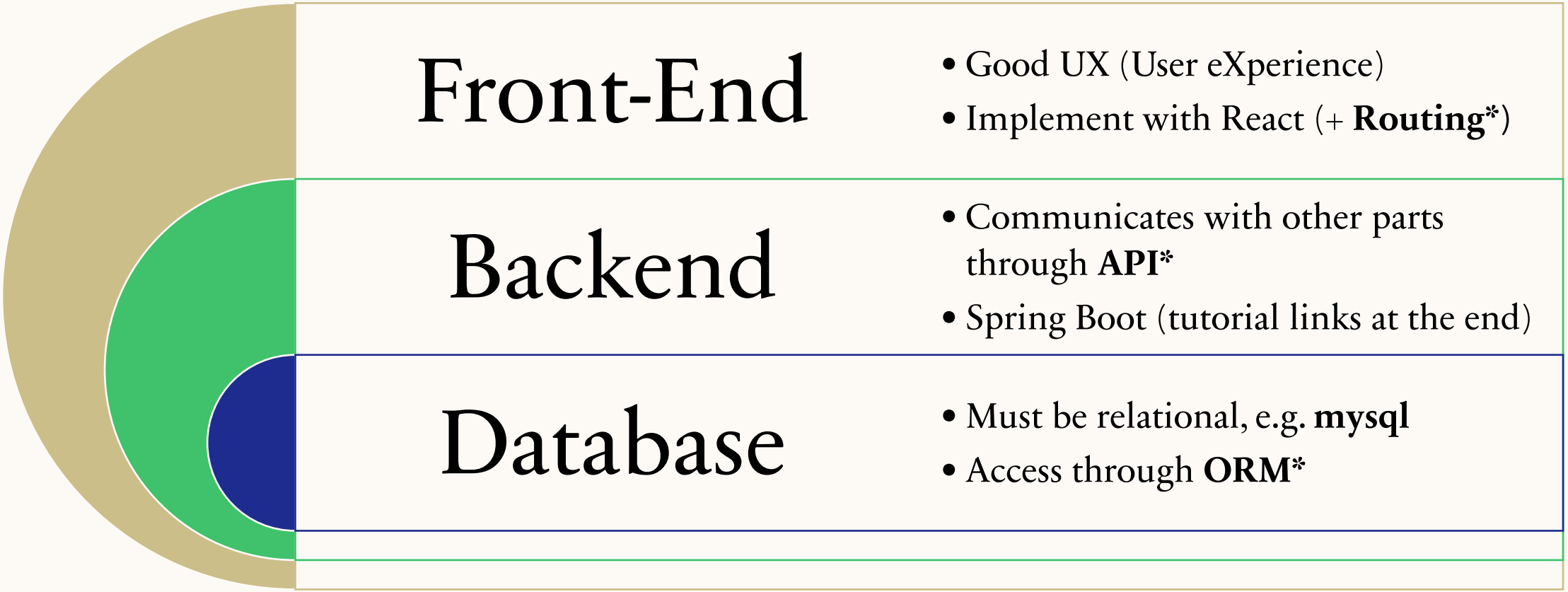
SOFTWARE ENGINEERING PROJECT SPORTSWEAR ESHOP

CONTENTS

1. PROJECT SCOPE



(*) Explanations in later slides

2. IMPLEMENTATION APPROACH

- Team registration by 09/11
- Must follow **Agile/Scrum***, where:
  - Requirements are in the form of **user stories***
  - There must be a **product backlog***
  - One **board*** per **sprint***
  - **sprints have a duration of 15 days**

  (Suggestion: Each user story must be possible to finish in one sprint)

SCRUM

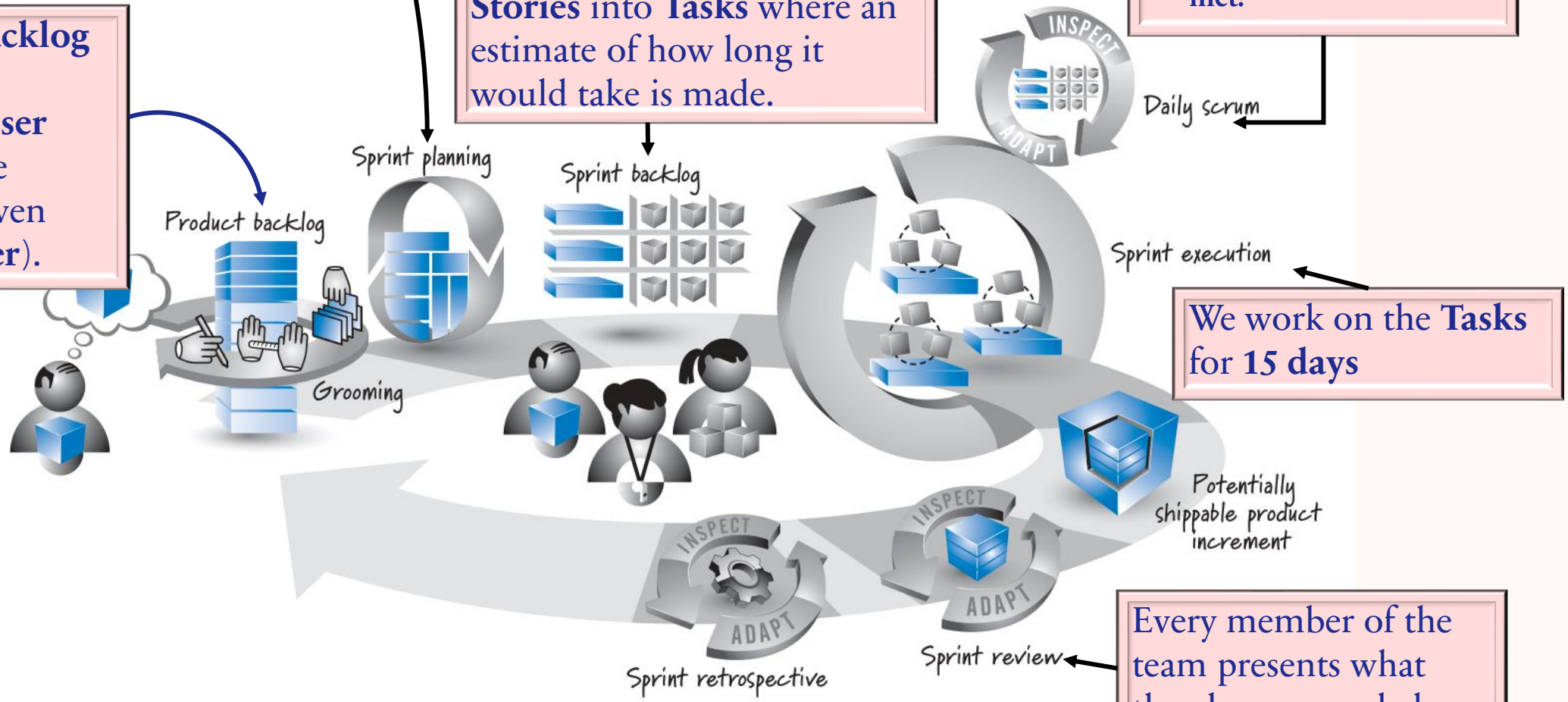
In **Sprint Planning** the **Product Owner** selects which elements of the **backlog** will be completed in the current **sprint** thus creating the **sprint backlog**

The **Product backlog** contains all the requirements (**user stories**) that the customer has given (**Product Owner**).

The **sprint backlog** is basically breaking up **User Stories** into **Tasks** where an estimate of how long it would take is made.

In the **Daily Scrum** everyone answers:

- **What they did?**
- **What they will do?**
- **What obstacles they've met?**



We work on the **Tasks** for **15 days**

Every member of the team presents what they have succeded

METHODOLOGY

1° Βήμα

- Create the system
- Create the **user stories**

2° Βήμα

- Gather all **user stories** in a **backlog** (**Jira**)
- And start **sprint planning**

3° Βήμα

- Choose high-priority **user stories**
- Execute **sprints** while setting **sprint goals** at the start

USER STORIES

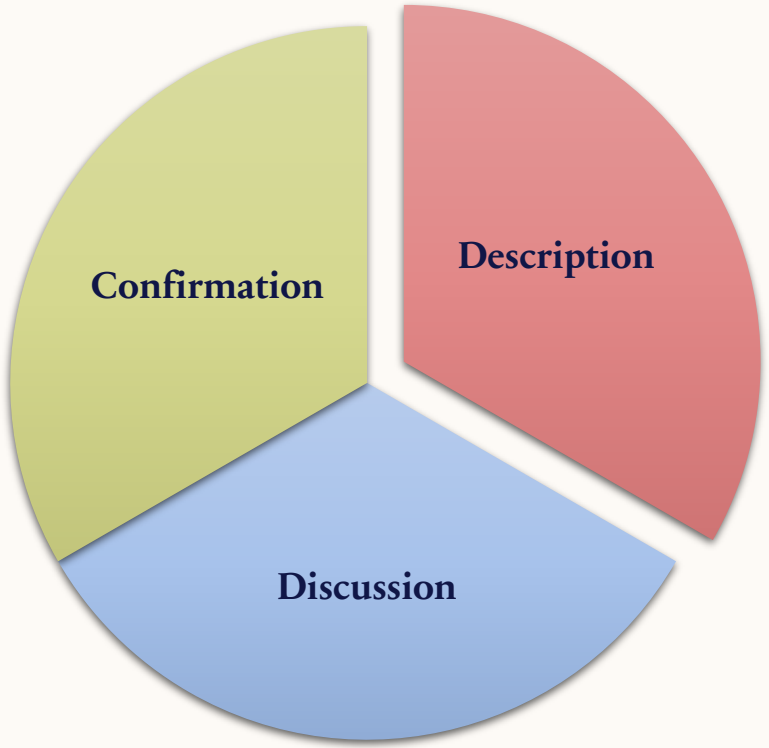
- **user stories** are made up of 3 parts:

1. **Description:** Short description answering the following 3 basic questions
   - **Who (user role)**
   - **What (goal)**
   - **Why (reason)**

   As a [user role] I want to [goal] so I can [reason]

   Example:

   As a **registered user** I want to **log in** so I can **access subscriber-only content**
2. **Discussion / Details**
   - Additional details will come up after talking to the client.
3. **Confirmation**
   - What checks must be done to verify that this user story has been implemented as desired.

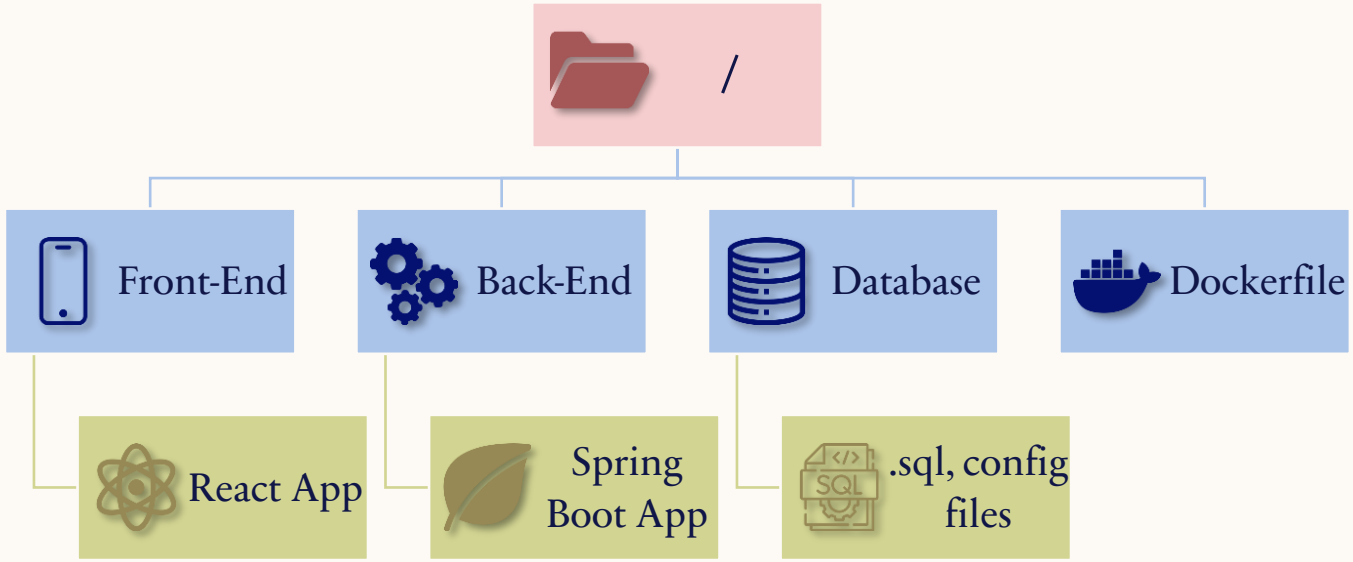


3. REPOSITORY

GitHub (Public ή Private ?) & Git

Project Structure (Example):

(This structure allows for a smoother work with docker, allowing our project to be made as microservices)



Icons source: Freepik - Flaticon.com

4. SYSTEM ARCHITECTURE

Front-End

- Communicates with business logic via **Restful API web services**
- Must be made as an **SPA** (Single Page Application)
- **Routing library** will be needed for **React**. e.g. **React Router**, **Next.js**, etc..

Back-End

- OOP language, e.g. **Java**
- **Spring Boot**, **Spring Security**
- **API consisting of 3 layers** (**controllers**, **business logic**, **data**)(examples in later slides)
- **Dependency Injection** (Explanation and examples in later slides)

4. ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΣΥΣΤΗΜΑΤΟΣ

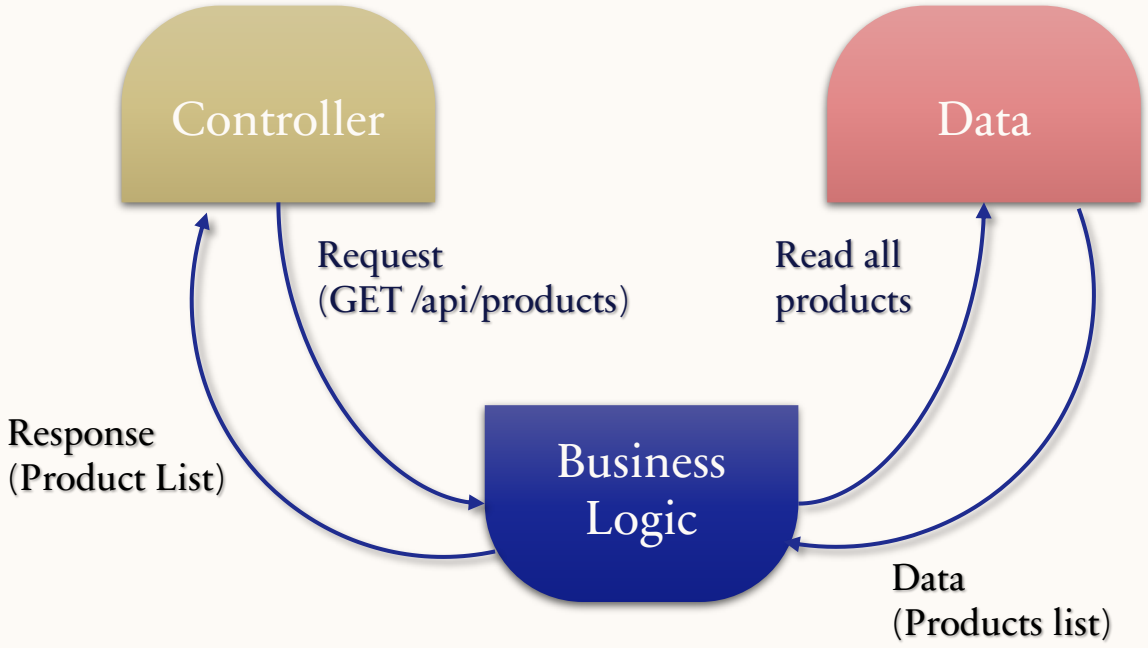
Database

- Must be relational, hence **mySQL** (**MariaDB, MySQL**)
- Communicates with business logic via **ORM*** (Object Relational Model) (**Hibernate**)
- We must first make the database, it will then be deployed to the server so that all members of the team have easier access.

Development

- **Front-End & Back-End** -> **Local**, each member on their own **Workstation**
- **Database** -> **Remote** (**each member will be supplied with IP and credentials**)
- **Server** -> **Open 24/7** , At the end we will deploy the project with docker compose

4. SYSTEM ARCHITECTURE



What is **API** in 3 **layers ?**

Basically what we are asked to do, is the **standard way** in which **CRUD applications** are made. We have 3 layers:

- **Controller**
  - **Accepts requests** from the Front-End and **sends** back the **responses.**
- **Business Logic (Service)**
  - **Decides what to do, based on the request.**
- **Data Layer (or Repository)**
  - Communicates with the database via the ORM method and executes CRUD actions.

4. SYSTEM ARCHITECTURE - EXAMPLE

Controller

```
@RestController
@RequestMapping("/api/products")
public class ProductController {
  private final ProductService service;
  public ProductController(ProductService service) {
    this.service = service;
  }
  @GetMapping // GET
  public List<Product> getAllProds() {
    return service.findAll();
  }
}
```

4. SYSTEM ARCHITECTURE - EXAMPLE

Business Logic (Service)

```
@Service
public class ProductService {
  private final ProductData repo; // ORM
  public ProductService(ProductData repo) {
    this.repo = repo;
  }
  public List<Product> findAll() {
    return repo.findAll();
  }
}
```

4. SYSTEM ARCHITECTURE - EXAMPLE

Data (ORM) - Repository

```
@Repository
public interface ProductRepository extends JpaRepository<Product, Long> {
}

@Entity
public class Product { // Class is mapped directly to the equivalent table
    @Id @GeneratedValue
    private Long id;
    private String name;
    private double price;
}
```

5. AUTHORIZATION / AUTHENTICATION

Authorization

Check what functions are available to the specific user.

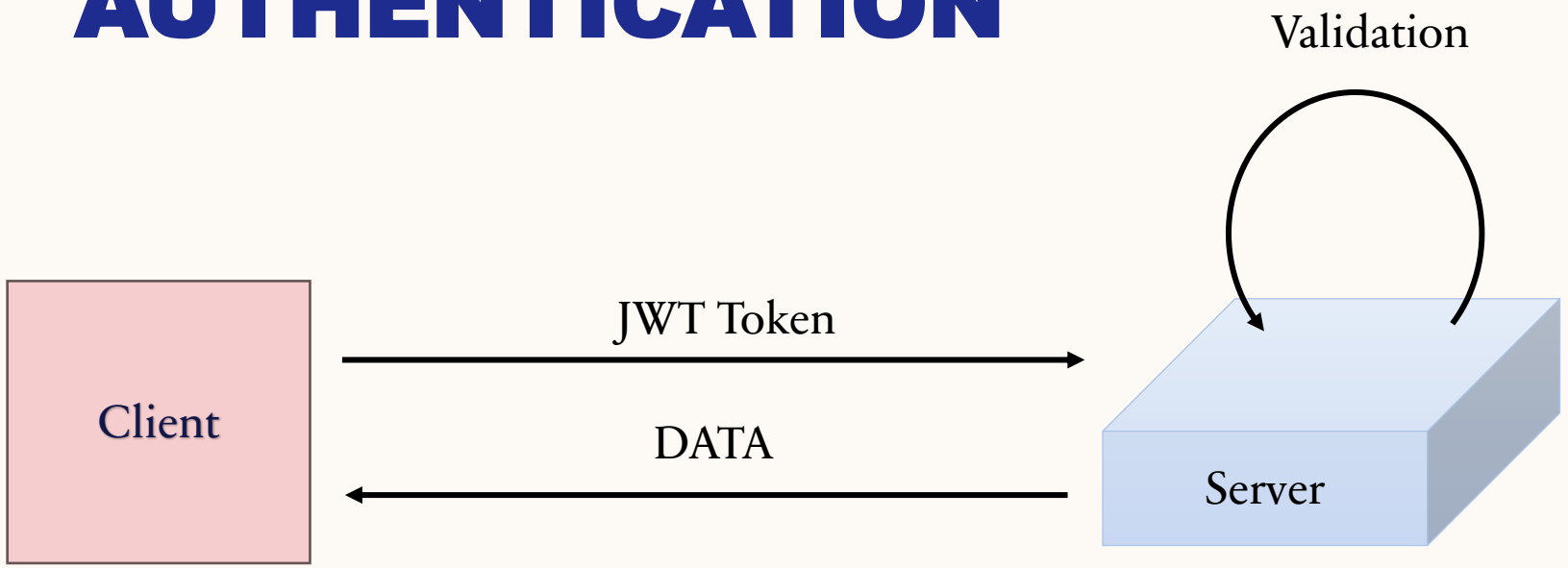
Authentication

Verify the users identity based on the credentials supplied.

Info

JWT (JSON Web Token): Basically after the **Authentication** a **JWT token** is created which will provide access to protected **REST Endpoints** and can be combined with which endpoints each role will have access via **RBAC (Role Based Access Control**

5. AUTHORIZATION / AUTHENTICATION

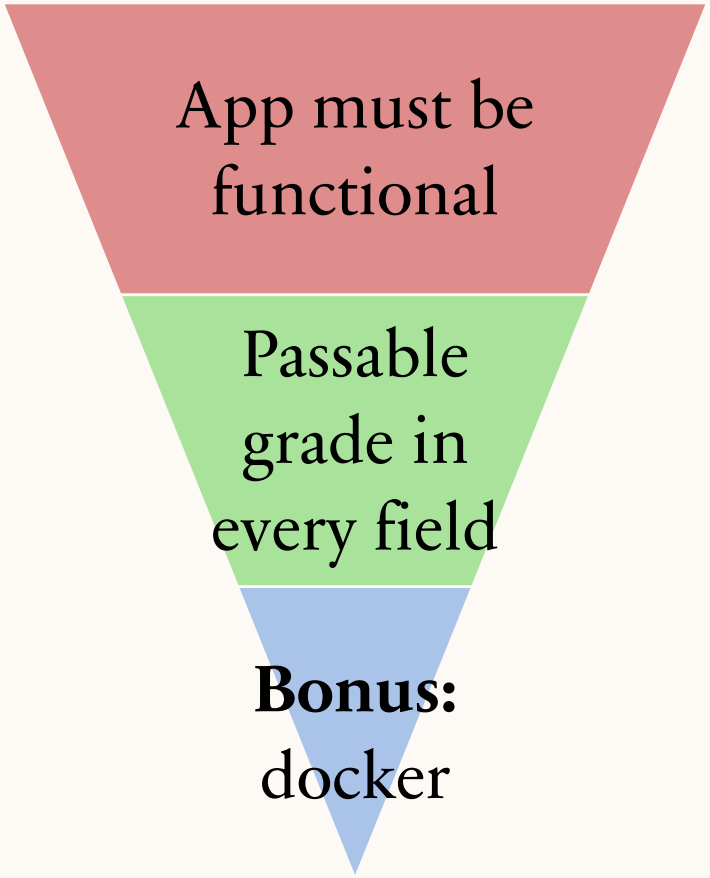


What is JWT? (~5mins)

6. TESTING

- Only a few integrated end-to-end tests at the end of the semester. We will see them later on.

7. EVALUATION



Field	Points
Use of repository	2
Use of planning tools for Ύπαρξη εργαλείου για την παρακολούθηση των sprints και χρήση αυτού	2
Usage of Dependency Injection	1
Usage of ORM	1
Integration Tests	2
Code Quality / UI/UX	2

THANK YOU

Valentino Velchev
[Your phone]
[Your email]