deti universidade de aveiro departamento de eletrónica, telecomunicações e informática

TQS: Product specification report

Gabriel Hall [102851]*, Tomás Almeida *[103300], Leonardo Flórido [103360]

1 Introduction

1.1 Overview of the project

Our project revolves around the idea of a platform that allows for a network of pickups points.
We will act as a middleman between eCommerce Stores and Pickups. We will have on one side eCommerce sites that partner with us and on the other side Pickups.
This will enhance the economy of the physical stores that partner with us and allow for customer to have a broader delivery choice.

1.2 Limitations

-We missed the mapping of the customer information on the platform. It appears as [Object Object]. Even though this doesn't worsen the functionality of the web application, it is a frontend feature that we didn't implement.
-We had some troubles implementing Continuous Deployment even though everything is Deployed in the Google Cloud.

2 Product concept

2.1 Vision statement

High Level Perspective
A platform that provides eCommerce Stores with pickups, allows for pickups to join the collective association, and allows for pickups to monitor the orders that are of their responsibility.

We planned to build a mock eCommerce Store and the main platform.
The platform will offer some statistics on the number of orders that are being made spread across time and will display the specific details for each order.

2.2 Personas and scenarios

Admin – John Doe – Business Administrator
Goals:
Maintain the platform;
Manage partners;
Have a high-level view of the platform performance;
Stories:
As an admin, I want to see statistics on the orders spread across time of all pickups;
As an admin, I want to accept a new partner;

Pickup Owner – Mary Jane – BookStore Owner
Goals:
Increase traffic on her store; (indirect goal)
Manage orders on the platform;
Have a statistical view over the orders she has received across time;
Stories:
As a pickup Owner, I want to see statistics on the orders spread across time
As an admin, I want to change the order status on the platform;

Customer – Jack Doe – Tech fan
Goals:
Buy a new macbook;
Have the order delivered to the nearest pickup;
Stories:
As a customer, I want to add a macbook to cart and proceed to checkout
As a customer, I want to choose the pickup point
As a customer, I want to get my macbook from the pickup when it arrives

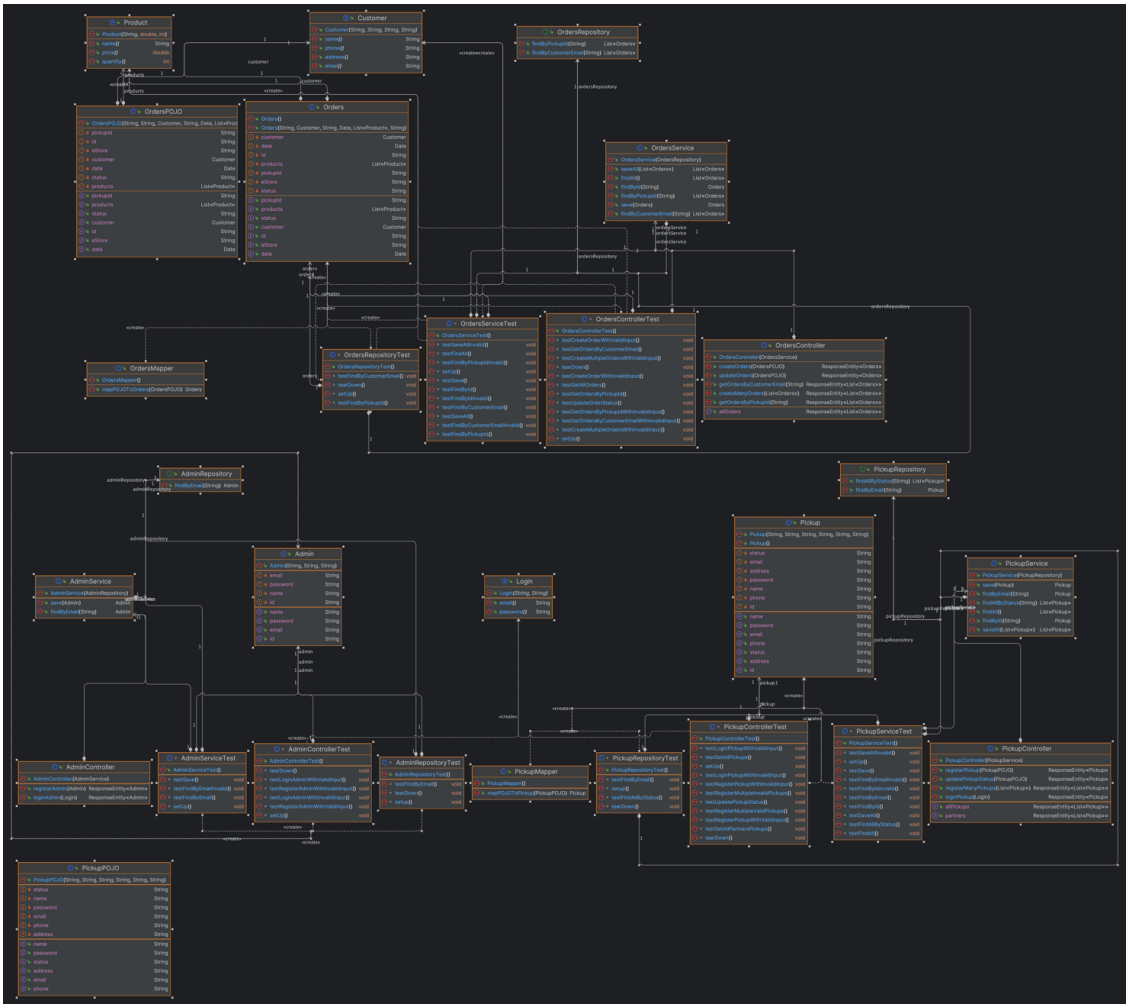
2.3 Project epics and priorities

Sprint	Epic
1	Initial system architecture; Technologies; Work Distribution;
2	Initial prototype development (eStore); Main user stories Defined;
3	Initial frontend development (Pickup Platform); Initial backend development; CI pipeline; SonarCloud;
4	All tests developed; All backend & Frontend developed (Platform);

5	eStore development finished & connected to Pickup platform; Deployment; Bug fix;

3 Domain model

Platform



Models

Name	Attributes	Description
Pickup	Id; phone; name; password; address; email; status;	Entity to represent a pickup
Admin	id; name; password; email	Entity to represent an Admin
Orders	Status; eStore; pickupId; products; id; date; customer	Entity to represent an Order. Products and

		Customer are objects that have their own structure.

Entities

Name	Attributes	Description
Customer	Name; phone; address; email	Entity to represent a customer
Product	Name; price; quantity	Entity to represent a product

Services

We have implemented 3 services in our system: the Pickup Service, the Admin Service, and the Orders Service:

- The Pickup Service is responsible for handling the registration, listing, and retrieval of pickups. It allows users to register new pickups, retrieve a list of all pickups, and search for specific pickups using email or ID.

- The Admin Service is designed for registering and retrieving information about admins. It enables the registration of new admins and provides a search functionality to find admins using their email.

- The Orders Service handles the registration of new orders, updating existing orders, and providing functionality to list and search for orders. Users can register new orders, update existing ones, retrieve a list of all orders, and search for orders based on customer email or pickup ID.

By dividing the functionality into these three services, we have achieved a modular and scalable system that efficiently handles the different aspects of pickups, admins, and orders.

Controllers

We have developed 3 different controllers:

- **Admin Controller**
  - The AdminController handles registration and login functionalities for admins using the **AdminService**.
  - In the /register endpoint, it checks if the admin already exists and saves the admin after encrypting the password.
  - The /login endpoint verifies the existence of the admin, checks the password for correctness, and returns the admin information upon successful authentication.

- **Orders Controller**
  - The OrdersController handles order-related operations and utilises the OrdersService.
  - Endpoints include /create for order creation, /get-all to retrieve all orders, /get-by-pickup/{id} for fetching orders by pickup ID, /update-status to update order status, /get-by-customer/{email} for retrieving orders by customer email, and /create-many to create multiple orders simultaneously.

- **Pickup Controller**
  - The PickupController handles pickup registration and login functionalities using the PickupService.

- The /register endpoint verifies the pickup's existence, encrypts the password, and saves the pickup.
- The /login endpoint checks if the pickup exists, validates its partner status, and verifies the password for authentication.
- The controller also provides endpoints for updating pickup status, retrieving all pickups, registering multiple pickups, and getting pickups with the "Partner" status.

4 Architecture notebook

4.1 Key requirements and constraints

Our system will be developed from scratch, aiming to create a robust platform that includes two main services: an eStore and a pickup point and delivery management platform. Both services will be integrated into the system.

For each service, we will have a web app, a business logic module, and a database.

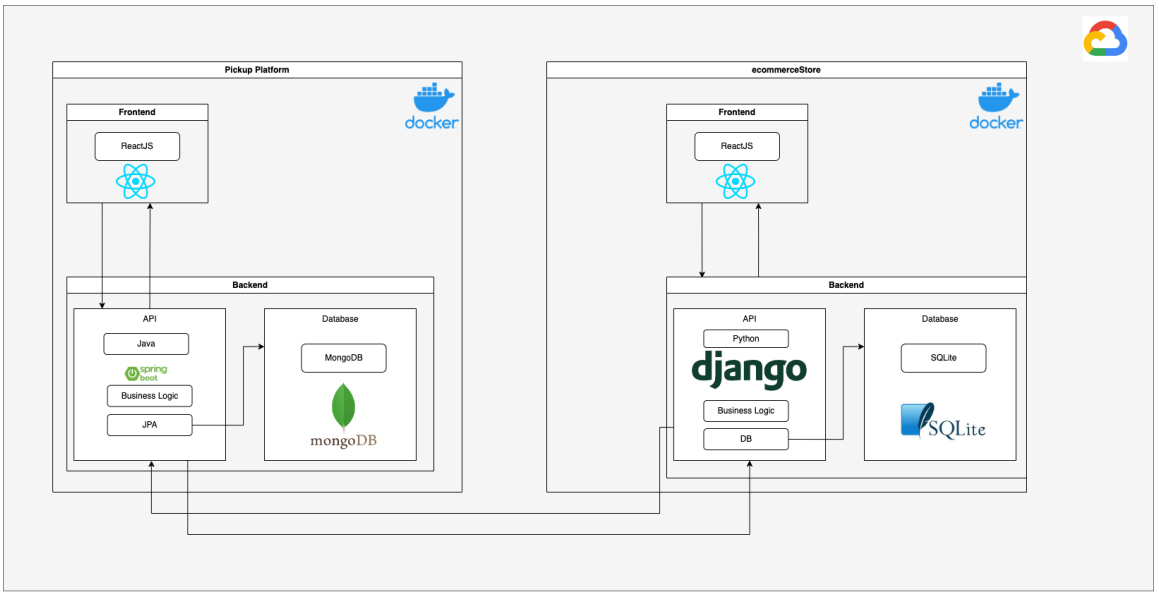
To facilitate communication between the web apps and the business logic, we will utilise HTTP requests to an API. Similarly, the communication between the specific eStore and the pickup point and delivery management platform will occur through an API.

To ensure scalability and portability, each service will be dockerized, and the entire system will be deployed on a virtual machine.

4.2 Architectural view

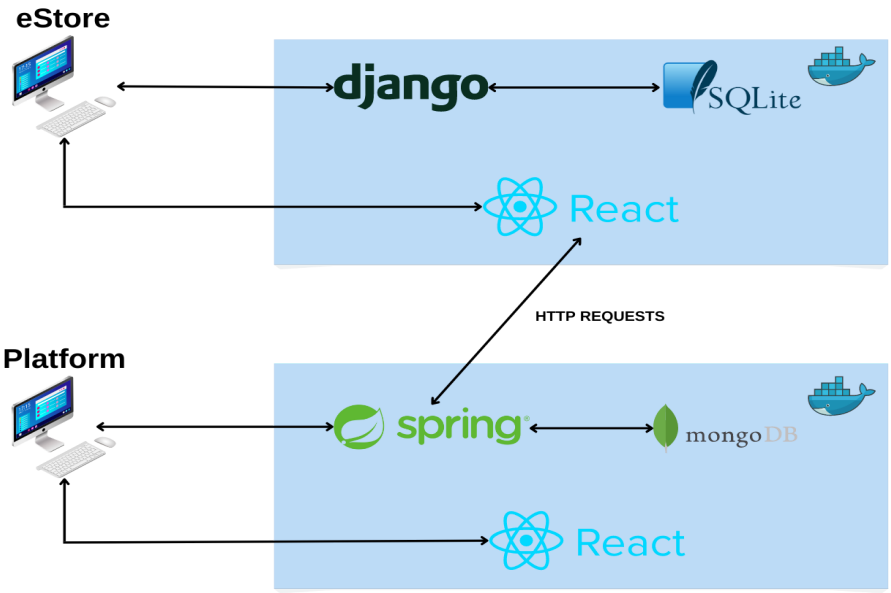
There are two projects in this scenario. The first project is an eStore application that utilizes React for the frontend and a Django API with a SQLite database for the backend. It allows users to view items, add them to the cart, and place orders on the platform.

The second project is a platform built using React for the frontend and a MongoDB database, supported by a SpringBoot API. Depending on the user's login credentials, they can either act as a manager for a specific pickup point, enabling them to change the status of each order associated with that pickup point, or as an admin, who has the authority to accept new partners (pickup points) and view all orders across different pickup points.



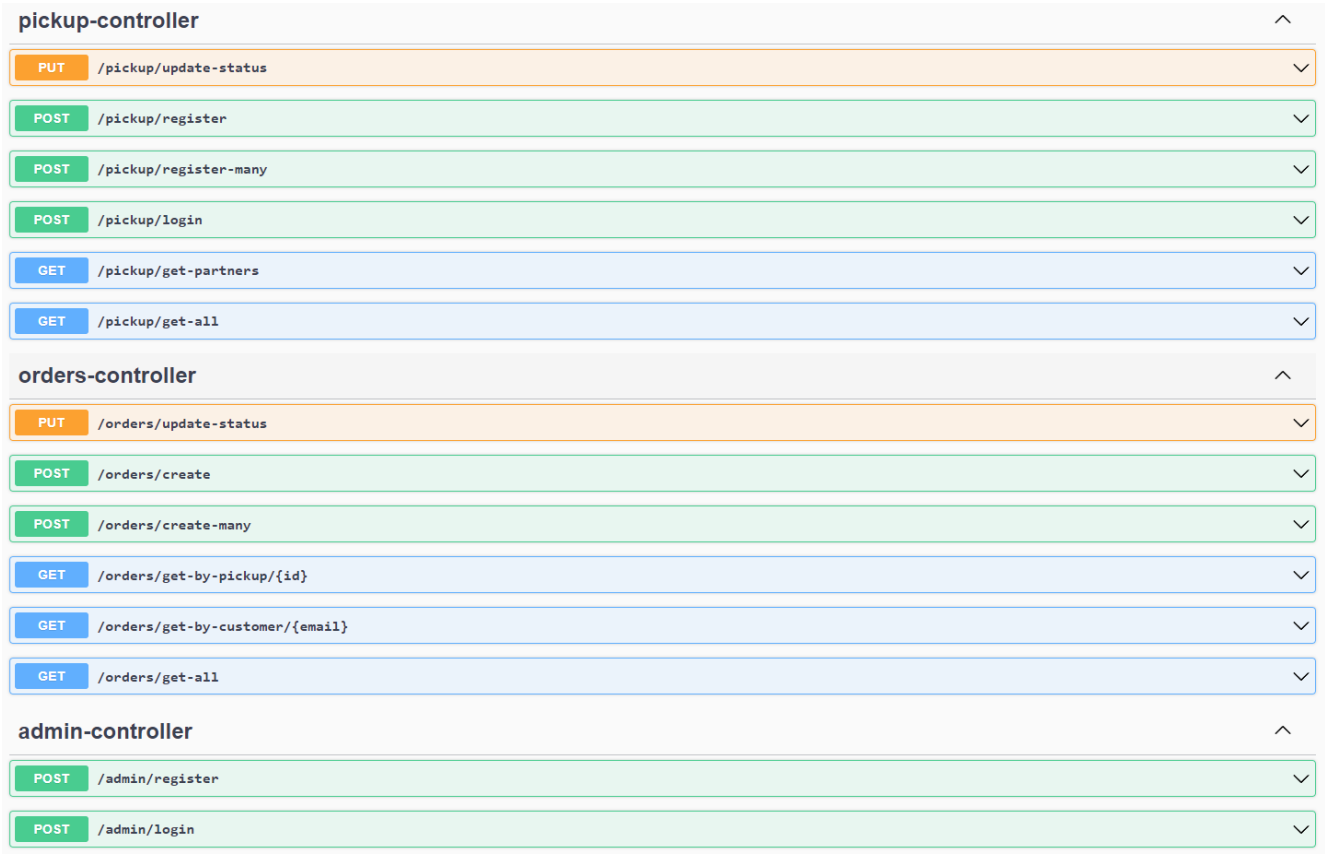
4.3 Deployment architecture

We have 2 docker-compose files that are running in the Google Cloud VM. One of the files represents the Platform and the other is a random eStore.



5 API for developers

5.1 Platform



6 References and resources