

# TQS: Product specification report

Ricardo Martins [112876], Rodrigo Jesus [113526], José Pedro [113403], André Dora [113613]

<b>1</b>	<b>Introduction .....</b>	<b>1</b>
1.1	Overview of the project .....	1
1.2	Known limitations.....	2
1.3	References and resources .....	2
<b>2</b>	<b>Product concept and requirements .....</b>	<b>2</b>
2.1	Vision statement .....	2
2.2	Personas and scenarios.....	3
2.3	Project epics and priorities.....	3
<b>3</b>	<b>Domain model .....</b>	<b>4</b>
<b>4</b>	<b>Architecture notebook.....</b>	<b>5</b>
4.1	Key requirements and constrains .....	5
4.2	Architecture view.....	6
4.3	Deployment view.....	7
<b>5</b>	<b>API for developers .....</b>	<b>8</b>

## 1 Introduction

### 1.1 Overview of the project

This project is developed for the TQS course and therefore its scope encompasses various themes such as CI, CD, automated testing, QA, clear architectural documentation and UML modeling. All things to be implemented and clearly visible in our project CarWash.

WashNow, a web-based platform for car-wash bays, designed for ease of use and for an efficient deployment, our system brings digital convenience to a traditionally offline service.

Thus, allowing drivers to easily locate available bays, reserve one or more time-slots, unlock the jet-wash upon arrival, run the washing program, pay automatically, and retrieve receipts, all within the comfort of our single user-friendly interface, and available for all types of car washes.

Car wash Operators/Owners on the other hand benefit from now having a dashboard to create and manage bays, define schedules and prices, while also monitoring water and energy consumption (available only on a more advanced part of the app development)

## 1.2 Known limitations

To be reviewed and completed by the end of the project >

All hardware integration will be simulated using via API and mocks, because there simply isn't enough time or means to implement using the real thing.

The same goes for the payment service, which also simulates the transactions with no real gateway

## 1.3 References and resources

<document the key components (e.g.: libraries, web services) or key references (e.g.: blog post) used that were really helpful and certainly would help other students pursuing a similar work>

Aqueles que são dados na página de TQS:

Selected books for TQS

a) [Books Playlist](#) at O'Reilly

Maven builds

- Maven [build lifecycle](#)
- Reference POM: [basic JUnit tests](#).
- [Search artifacts](#)/check current versions

Testing libraries - "asserts"

- [JUnit 5](#) Asserts
- [Hamcrest](#) for flexible matchers.
- [AssertJ](#) for fluent (natural to read) assertions.

## 2 Product concept and requirements

### 2.1 Vision statement

Finding a car wash station can be inefficient—drivers waste time searching for available spots, dealing with long queues, or arriving only to find closed facilities. WashNow solves this by providing real-time visibility of nearby car wash stations, their availability, and instant booking options.

Key features:

- b) Interactive Map - Locate car wash stations nearby with filters (self-service, automatic, detailing).
- c) Real-Time Availability – See live occupancy and estimated wait times.
- d) Instant Booking – Reserve a slot and receive confirmation.
- e) Operator Dashboard – Manage station schedules, pricing, and capacity.
- f) Payment Integration – Pay securely via MB Way, PayPal, or card.

## 2.2 Personas and scenarios

John Silver, 32, is a busy sales executive who values convenience and efficiency. He dislikes wasting time in lines, so when it comes to car washes, he looks for quick and easy booking options near his home or workplace. What he really wants is a transparent system that offers real-time status updates and upfront pricing.



Martha Rock, 45, is the owner of a family-run car wash business. She faces several operational challenges, including empty time slots, double bookings that lead to customer complaints, and a lack of online visibility, which causes her to lose potential clients to larger competitors.



Carlos Mendez, 29, is a rideshare driver who relies on his compact car for his livelihood. Driving around the city all day means his car gets dirty fast. With no time to waste, he needs quick, reliable car washes that fit seamlessly into his route between rides.



**Scenario:** John, as a busy sales executive, uses CarWash Tracker to quickly find an available car wash bay near his office. He books a 30-minute slot, and upon arrival, the system unlocks the jet-wash automatically. The payment is done through the app, and he receives a receipt via email. He saves time and avoids waiting in line, seamlessly fitting the wash into his hectic schedule.

**Scenario:** Martha as small car wash owner struggles with double bookings and empty time slots. Using CarWash Tracker, she sets up a digital calendar to manage reservations and prices. The platform reduces errors and increases online visibility, allowing customers to book easily. Martha also monitors water and energy usage to optimize her operations and attract more clients.

**Scenario:** Carlos, a rideshare driver, needs a quick and convenient car wash during his break. Whenever he has some downtime between rides, he uses CarWash Tracker to instantly locate the nearest available wash bay.

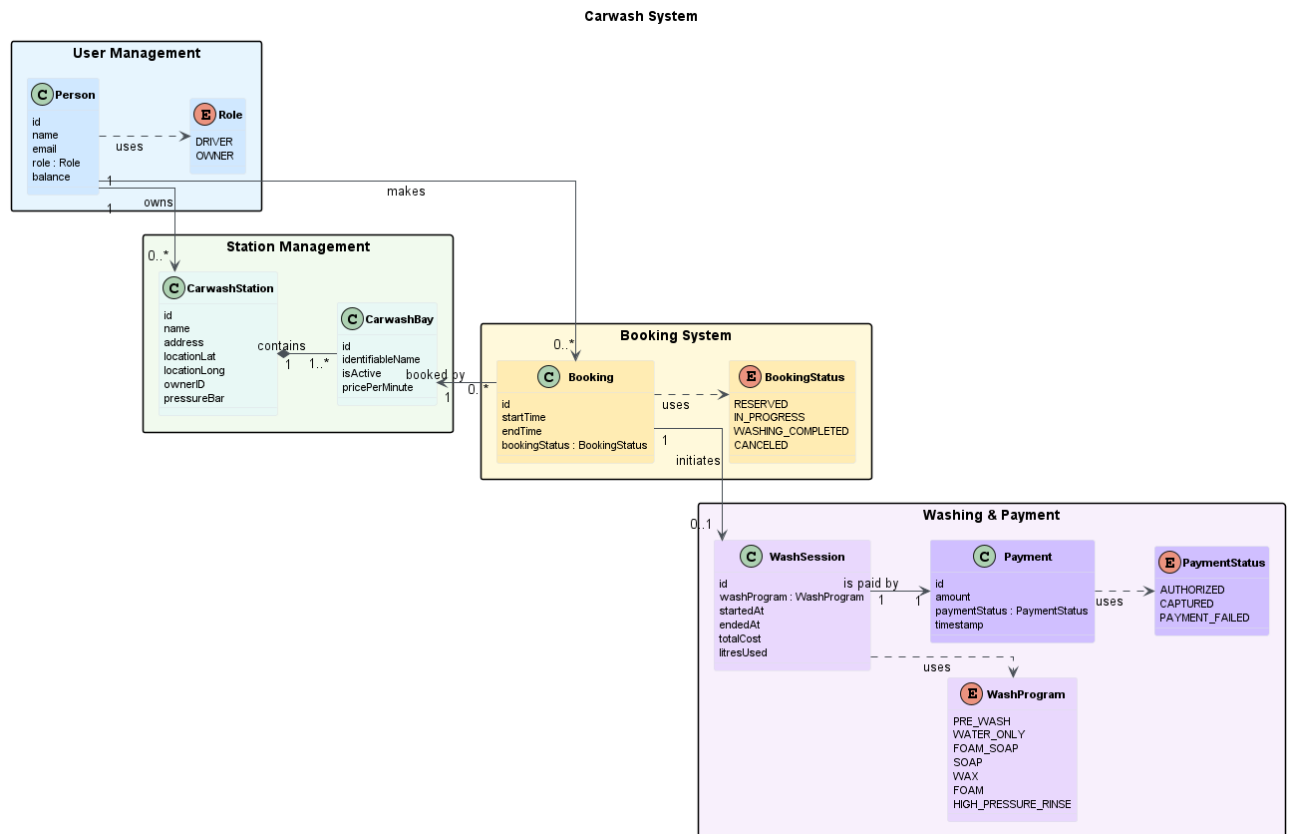
## 2.3 Project epics and priorities

Epic	Priority	Description
Car Wash Station Map	MUST	Interactive map with filters (service type, pricing).
Search and Filtering system	MUST	List of places filtered and sorted.
Booking System	MUST	Slot reservation, confirmation, and notifications.

Occupancy Management	MUST	Real-time updates by operators to prevent overbooking.
User Profiles	SHOULD	Registration/login, booking history.
Payments	SHOULD	Integration with Stripe, MB Way, or PayPal.
Ratings & Reviews	COULD	User feedback system for stations.
Analytics Dashboard	COULD	Usage statistics for operators.

### 3 Domain model

- g) **Person**: id, name, email, role (DRIVER, OWNER), balance.
- h) **CarwashStation**: id, name, address, location (lat and long), ownerID, pressureBar.
- i) **CarwashBay**: id, CarwashStation (FK), Identifiable\_name( talvez algo como Jets-1), isActive, pricePerMinute
- j) **Booking**: id, CarwashBay Id, userId, startTime, endTime, BookingStatus .
- k) **BookingStatus** (Enum): (RESERVED?, IN\_PROGRESS, WASHING\_COMPLETED; CANCELED)
- l) **WashSession**: id, bookingID, Washprogram, startedAt, endedAt, totalCost; [litresUsed] - dado futuro para caso dê e seria o mesmo que o “Monitoring the ecological CO2 footprint” do ficheiro do professor.
- m) **WashProgram** { PRE\_WASH, WATER\_ONLY, FOAM\_SOAP, SOAP, WAX, FOAM, HIGH\_PRESSURE\_RINSE }
- n) **Payment**: id, washSessionID, amount, PaymentStatus , timestamp.
- o) **PaymentStatus** (Enum): (AUTHORIZED, CAPTURED, PAYMENT\_FAILED )



## 4 Architecture notebook

### 4.1 Key requirements and constraints

All interactions with on-site hardware (e.g. bay-unlock controllers, water- and energy-meter feeds) must be encapsulated behind a well-defined service interface (port/adaptor). This abstraction ensures that, when real devices become available, they can be integrated without affecting the core application logic.

In terms of the quality of our platform and availability, the Frontend while not needing to be pretty it needs to be responsive web application with which the user can easily interact and then send API requests to the backend. Also, this backend API should and does follow RESTful principles for creating better code which can also in the future be expanded more easily.

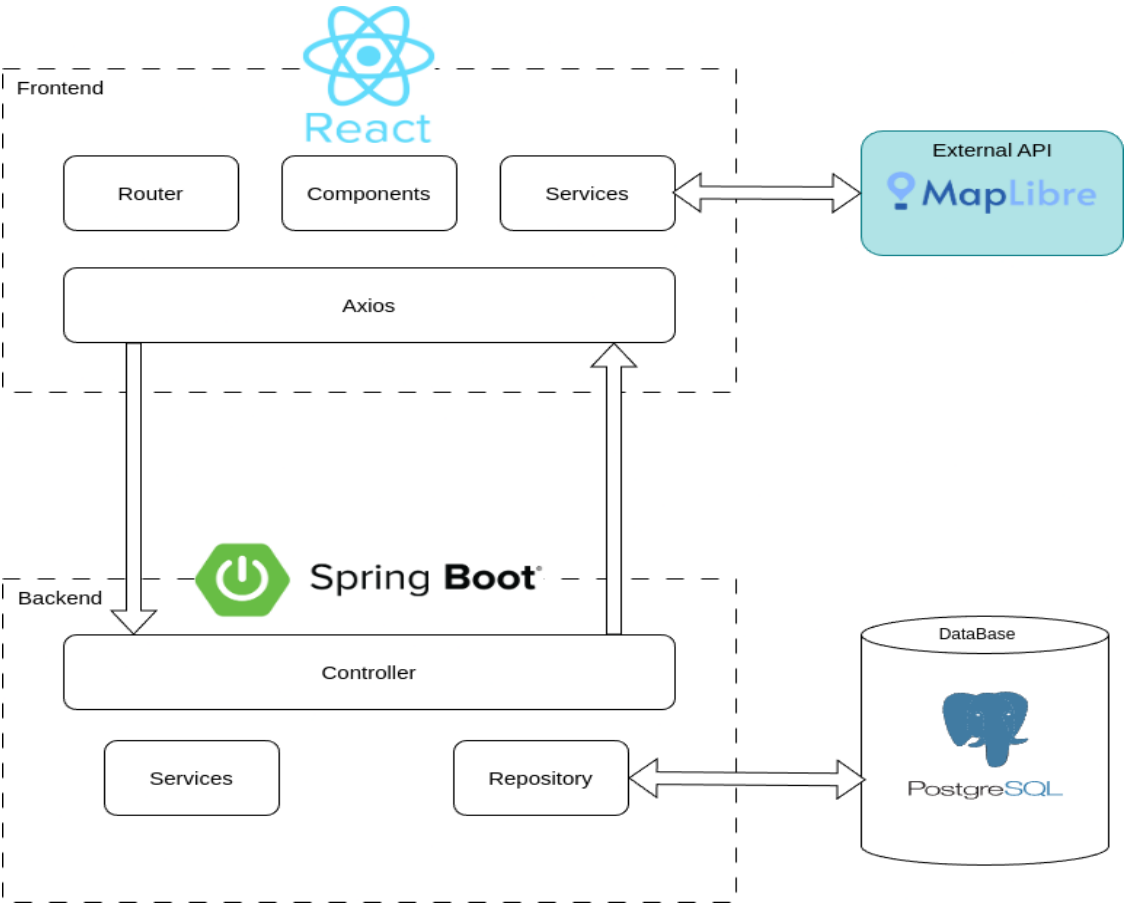
Our system will also be connected to external systems, namely:

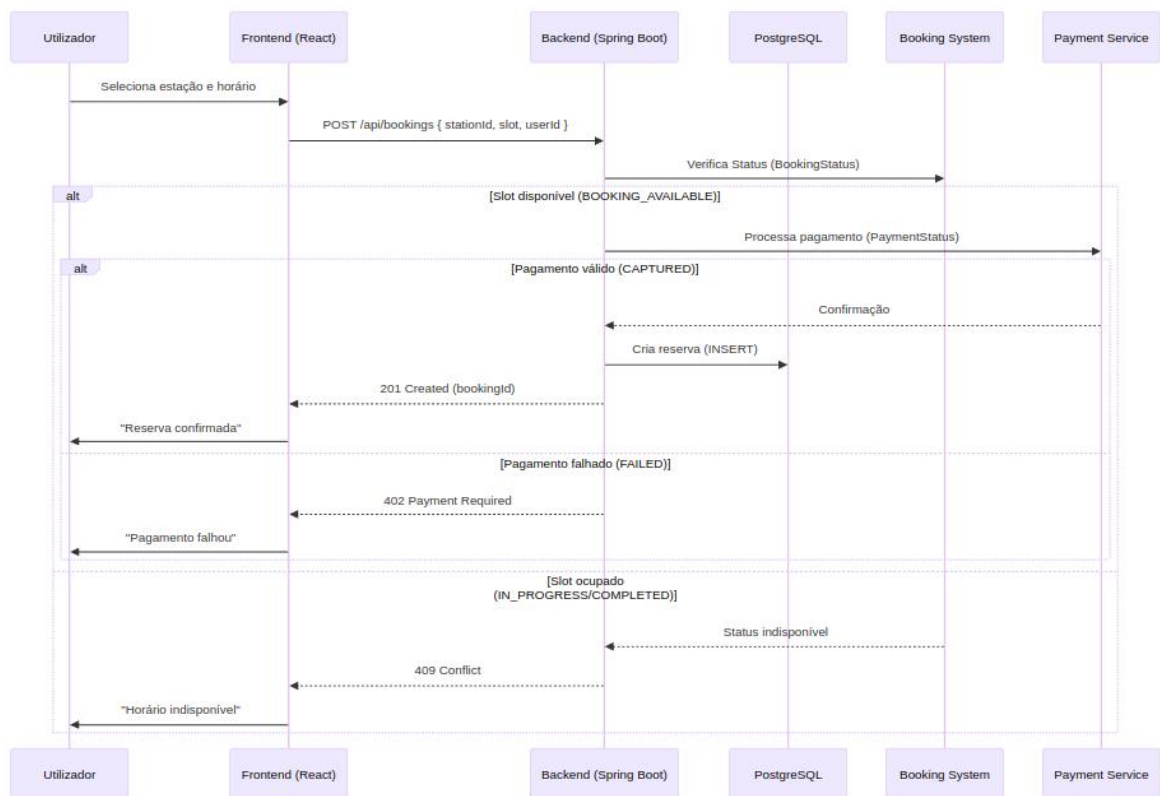
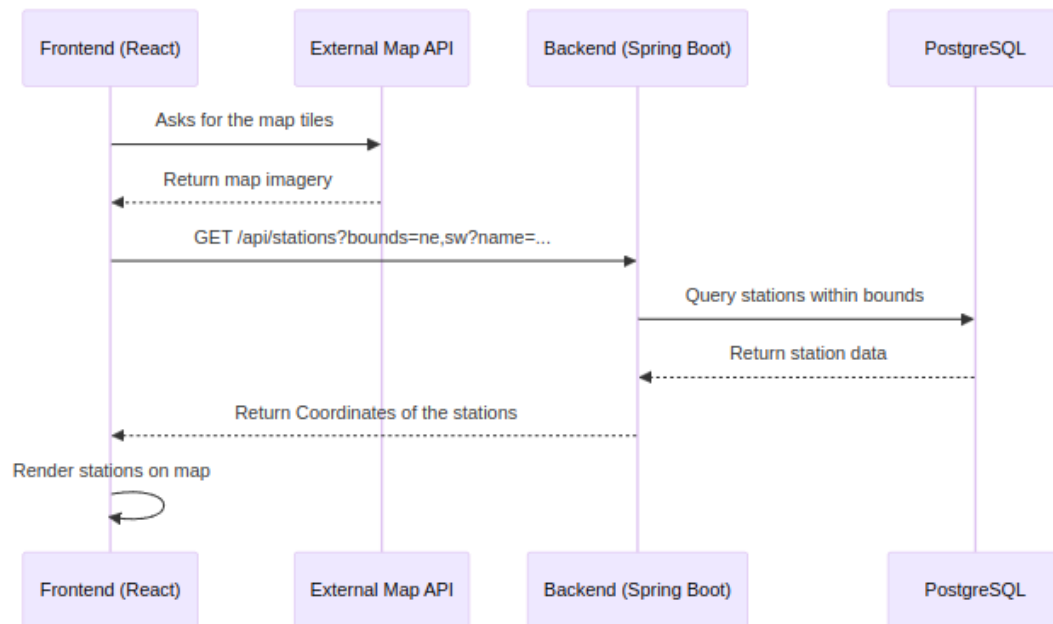
- A monitoring one to expose operational metrics such as request rates, latencies, error counts via Micrometer to Prometheus/Grafana for real-time observability.
- Another one for mapping & geolocation, which uses some kind of provider mapping API layer to locate the stations in the map and possibly calculate routes and distance.
- And finally, we have payment gateways such as MB Way, PayPal and card processors which we'd need to integrate via secure, versioned REST contracts.

In terms of Scalability, we should implement services which can be packaged as containers images and controlled via Docker Compose, to enable for easier scaling and to allow the system to grow and encompass all Carwash stations, for example from entire regions.

Finally for our Architectural characteristics we have thought and designed our system to follow the greats of modularity, extensibility, testability and observability, ensuring that each component can evolve independently, be thoroughly tested independently from external services, and then yield actionable metrics which we can visualize in our dashboards.

4.2 Architecture view





?

### 4.3 Deployment view

[Explicar a organização prevista da solução em termos configuração de produção (*deployment*). Anotar, no diagrama, as tecnologias de implementação, e.g.: colocar o símbolo do PostgreSQL na Base de dados,...]. Indicar a existência de containers (Docker), endereços IP e portos,... Esta parte será completada quando houver efetivamente deployments

## 5 API for developers

Our API organization follows RESTful method, resource-oriented design and it is organized under versioned paths, like we learned in previous classes: `api/v1/SpecificAPI` will be the URL format and inside each one we will have the HTTP methods:

GET for reads; POST for creates; PUT for updates; DELETE for removals

Resource	Endpoints	Description
Person	GET /api/v1/persons GET /api/v1/persons/{id} POST /api/v1/persons	CRUD for users and roles: To first get all users, then also get the info from just one specific user, then finally to add more users
Station	GET /api/v1/stations GET /api/v1/stations/{id} POST /api/v1/stations PUT /api/v1/stations/{id}	Manage car-wash stations: With the ability to get info from all stations, From just one specific station Add more stations to the DB Update the station Info
Bay	GET /api/v1/bays GET /api/v1/stations/{stationId}/bays/{id} PUT /api/v1/stations/{stationId}/bays/{id} GET /api/v1/stations/{stationId}/bays POST /api/v1/stations/{stationId}/bays	List and filter 1.all the wash bays (probably not going to be used); 2. just a specific one, 3. where you can also update its info; 4. Or you can go inside from a station and list all of its bays and then 5. update their info for example from active to close
Booking	POST /api/v1/bookings GET /api/v1/bookings?userId=... POST /api/v1/bookings/{id}/cancel	Endpoints for creating reservations with POST and then also query them using GET. Then since we user must first pay for the reservation before using it and we should also allow the user to cancel the reservation, of course if done far enough away in terms of time-if it wasn't return 409 conflict error, then we would need to return to him is money so it's best for that to create a specific endpoint
Session	POST /api/v1/bookings/{id}/start POST /api/v1/bookings/{id}/end	These endpoints to define the start and end of a wash session (aka unlock n complete)
Payment	GET /api/v1/payments?sessionId=... POST /api/v1/payments DELETE /api/v1/payments/{id}	View payment records and status, proceed with the payment, and then finally DELETE for canceling and deleting a payment