TQS: Product specification report

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

Table of Contents

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

All hardware integration will be simulated using via API and mocks, because there simply isn't enough time or means to implement using real technology.
The same goes for the payment service, which also simulates the transactions with no real gateway
We weren't able to achieve the following objectives we had set out to do initially:

a) Showing on the map the real or expected concentration of people in certain was stations
b) A complete Owner page where he could see how much each individual station or washbay was consuming in terms of water and energy, and organized by time

1.3 References and resources

The ones given to us in TQS:

Selected books for TQS

c) 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:

d) Interactive Map - Locate car wash stations nearby with filters (self-service, automatic, detailing).

e) Real-Time Availability – See live occupancy and estimated wait times.

f) Instant Booking – Reserve a slot and receive confirmation.

g) Operator Dashboard – Manage station schedules, pricing, and capacity.

h) 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

i) **Person**: id, name, email, role (DRIVER, OWNER), balance.

j) **CarwashStation**: id, name, address, location (lat and long), ownerID, pressureBar.

k) **CarwashBay**: id, CarwashStation (FK), Identafiable_name(talvez algo como Jets-1), isActive, pricePerMinute

l) **Booking**: id, CarwashBay Id, userId, startTime, endTime, BookingStatus .

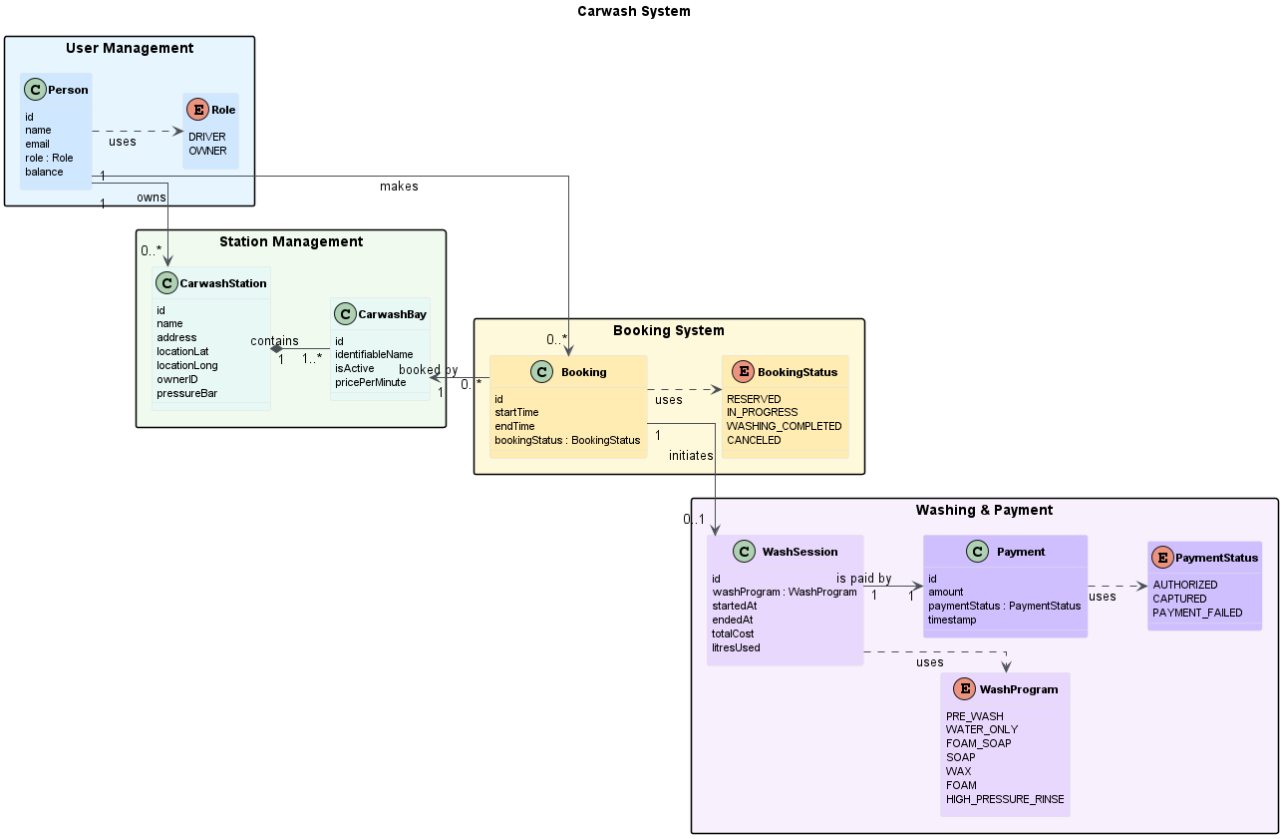
m) **BookingStatus** (Enum): (RESERVED?, IN_PROGRESS, WASHING_COMPLETED; CANCELED)

n) **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.

o) **WashProgram** { PRE_WASH, WATER_ONLY, FOAM_SOAP, SOAP, WAX, FOAM, HIGH_PRESSURE_RINSE }

p) **Payment**: id, washSessionID, amount, PaymentStatus , timestamp.

q) **PaymentStatus** (Enum): (AUTHORIZED, CAPTURED, PAYMENT_FAILED)



4 Architecture notebook

4.1 Key requirements and constrains

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.

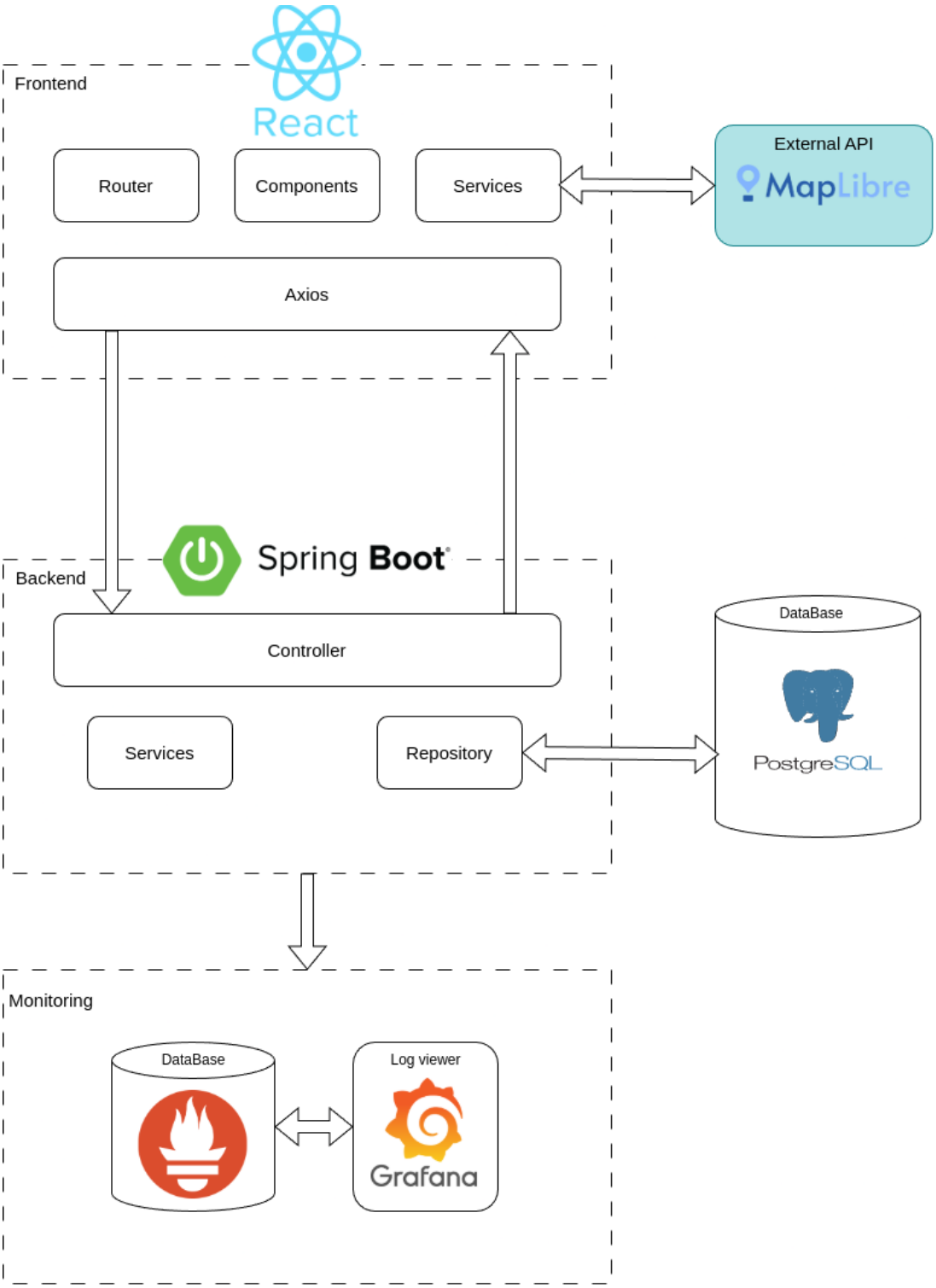
In terms of the quality of our platform and availability, the Frontend while not needing to be pretty it needs to be responsive web application with which the user can easily interact and then send API requests to the backend. Also, this backend API should and does follow RESTful principles for creating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

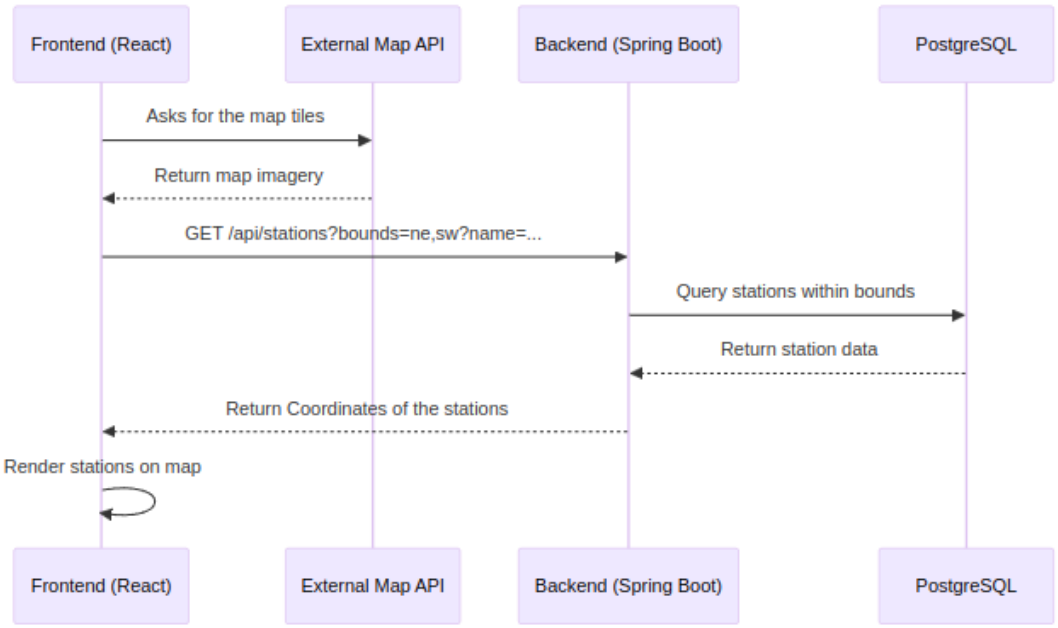
Our system will also be connected to external systems, namely:
– A monitoring one to expose operational metrics such as request rates, latencies, error counts via Micrometer to Prometheus/Grafana for real-time observability.
– Another one for mapping & geolocation, which uses some kind of provider mapping 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.

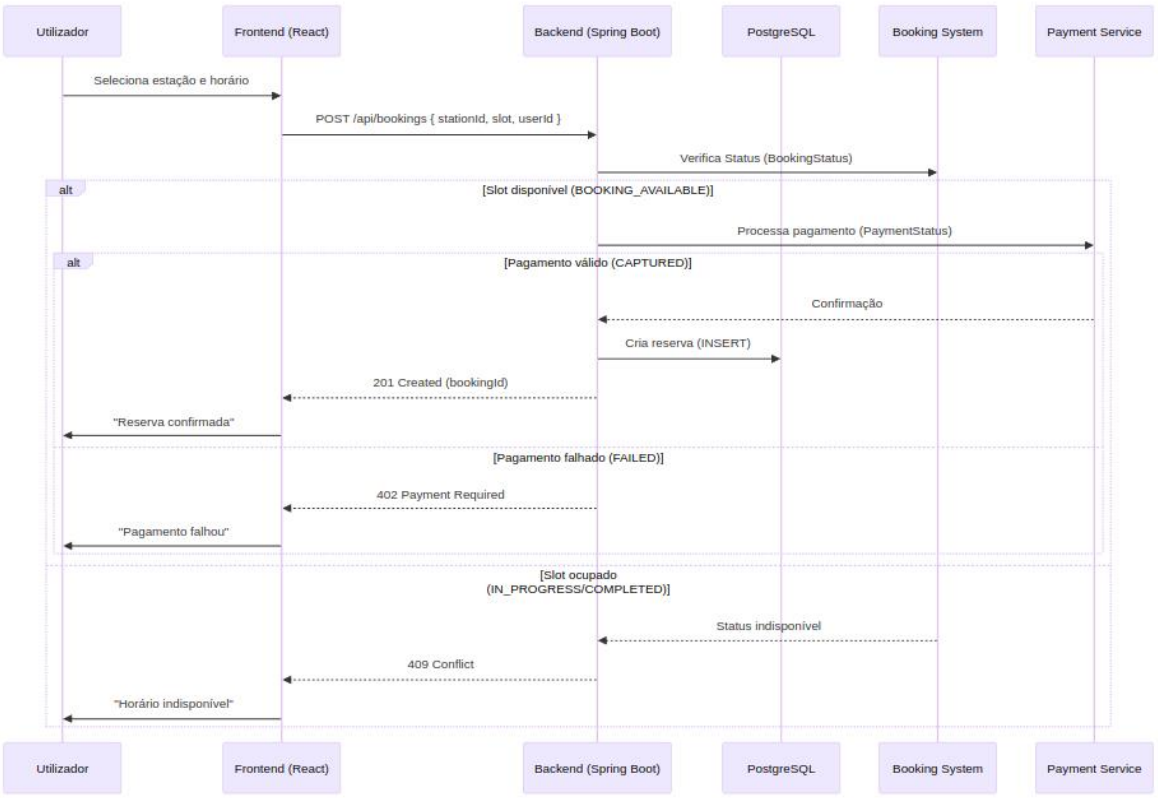
In terms of Scalability, we should implement services which can be packaged as containers images and controlled via Docker Compose, to enable for easier scaling and to allow the system to grow and encompass all Carwash stations, for example from entire regions.

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




Frontend (React)
External Map API
Backend (Spring Boot)
PostgreSQL
Asks for the map tiles
Return map imagery
GET /api/stations?bounds=ne,sw?name=...
Query stations within bounds
Return station data
Return Coordinates of the stations
Render stations on map
Frontend (React)
External Map API
Backend (Spring Boot)
PostgreSQL


Utilizador
Frontend (React)
Backend (Spring Boot)
PostgreSQL
Booking System
Payment Service
Seleciona estação e horário
POST /api/bookings { stationId, slot, userId }
Verifica Status (BookingStatus)
alt
[Slot disponível (BOOKING_AVAILABLE)]
Processa pagamento (PaymentStatus)
alt
[Pagamento válido (CAPTURED)]
Confirmação
Cria reserva (INSERT)
201 Created (bookingId)
"Reserva confirmada"
[Pagamento falhado (FAILED)]
402 Payment Required
"Pagamento falhou"
[Slot ocupado (IN_PROGRESS/COMPLETED)]
Status indisponível
409 Conflict
"Horário indisponível"
Utilizador
Frontend (React)
Backend (Spring Boot)
PostgreSQL
Booking System
Payment Service

4.3 Deployment view

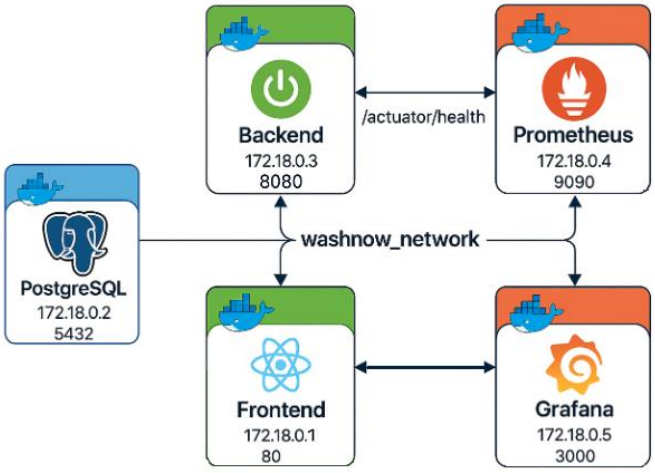
The deployment architecture of WashNow consists of multiple services running in isolated containers, connected using Docker via the washnow_network, ensuring our system is able to scale, deploy easily to other machines and is also able to integrate safely with external services.
So we have a container for our PostgreSQL database, another for Spring Boot backend, one for the React frontend, and then two more for the monitoring layers with Prometheus and Grafana.

For our Database we have PostgreSQL running on the default port 5432, which manages all of the data persistence operations, such as storing user profiles, booking records, and payment transactions. The database is initialized with an administrative user (admin) and a dedicated database named washnow.

The Backend is implemented in Spring Boot and listens on port 8080 and exposes a RESTful API for the frontend to interact with, while also integrating with external systems, such as mapping APIs. The backend health status is also monitored via an /actuator/health endpoint scraped to Prometheus

To provide real-time observability, our system includes a Prometheus container running on port 9090 and another for Grafana running on port 3000. This way we can collect, store and visualize metrics from our system easily, which help us for the process of monitoring.

Finally we have the Frontend, which is built in React and is running on it's own container, mapped to the port 80.



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/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/persons GET /api/persons/{id} POST /api/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/stations GET /api/stations/{id} POST /api/stations PUT /api/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/bays GET /api/stations/{stationId}/bays/{id} PUT /api/stations/{stationId}/bays/{id} GET /api/stations/{stationId}/bays POST /api/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/bookings GET /api/bookings?userId=... POST /api/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/bookings/{id}/start POST /api/bookings/{id}/end	These endpoints to define the start and end of a wash session (aka unlock n complete)
Payment	GET /api/payments?sessionId=... POST /api/payments DELETE /api/payments/{id}	View payment records and status, proceed with the payment, and then finally DELETE for canceling and deleting a payment