Technical Report - **Product specification**

SnapPark

Course:	IES - Introdução à Engenharia de Software
Date:	Aveiro, 13/11/2023
Students:	108546: Tiago Pereira 108298: Diogo Machado Marto 107186: Vítor Santos 73259: Diogo Gaitas
Project abstract:	Snap Park is an application that aims to provide parking lot owners and users monitoring and managing utilities over them such as viewing current occupation.

Table of contents:

1 Introduction

The following document is a specification report for a IES (Introdução a Engenharia Informática) semester-long project, which focuses on the following objectives, related to the curricular unit's own objectives:

- Develop a software product specification, from its usage requirements/scenarios (user stories) to its technical design.
- Propose, justify and implement a software architecture, based on enterprise *frameworks*.
- Put collaborative work practices for code development and agile project management into practice.

2 Product concept

Vision statement

Snap Park is a web app purposefully designed with parking lot owners in mind. It aims to provide them with several monitoring and managing features that allow them to have a more complete and informed view of what's happening in their parks, including essential information such as the number of parked vehicles. Snap Park is sure to make managing and finding the best park both easier and more efficient.

Personas and Scenarios

Personas

John

John is a forty-year-old portuguese owner of a moderately-sized general store with a degree in management and over 20 years of experience in the business. He's worked with basic retail management systems and general information management tools like Microsoft Excel to track product stock, sales and employee salaries.

John recently acquired the nearest parking lot, being immediately next to the store. He intends to use the parking lot to boost client numbers. He expects this to be the case because this parking lot was restricted to public use for a long period of time and his general store has just a few available parking spots at the front, reserved for his employees. Other parking spots are occupied, most of the time, as this zone of the city is very active.

John wants to enforce a limited free parking period policy of 45 minutes for any driver. Those who remain in the parking lot will be applied the same 5€ fee every 30 minutes, beginning and including the instant they surpass the initial 45 minute limit.

John has searched the parking lot market for solutions to this enforcement, but he simply can't find a simple low-budget solution that is mostly automatic, requiring minimal human supervision and doesn't rely on expensive CCTV systems (not to be confused with wireless network surveillance cameras).

Scenario 1 – Parking Lot Setup

Due to his latest parking lot acquisition, it's in his best interest to monitor every aspect of it so his customers get the best user experience possible and so he can make the most amount of money running it. He wants the process of setting up his new park to be as easy as possible.

To set up a new park, he presses "New Park", gives it a name and it gets registered under his user. He then needs to set up all of his sensors that he has already installed himself or call a company to install them. To set up a new sensor he needs to click his park and then click "New Sensor" where he will scan the sensors and it will be automatically added to his park in the app. He also needs to set up new appliances, like lights, an air purifier, or the wireless network surveillance cameras.

Scenario 2 - Monitor Air Quality

John is worried about his park being overrun by toxic gases when the park is full and with heavy traffic. So he wants to be able to monitor the air quality so he can activate or deactivate air purifiers whenever needed. After having the air quality sensor and the air purifier appliance set up, all he needs to do is press the air quality sensor and all the data is there for him to check. Whenever he feels it's needed he just needs to turn on the air purifier, with the simple click of a button conveniently placed within the air quality sensor page.

Scenario 3 - Monitor Light Levels

To give his customers an enjoyable experience when using his park, he wants to make sure there is always good visibility in his park.

So, to achieve that he needs to be able to know the light levels of his park at all moments so he can turn the lights on

only when needed, so he doesn't waste energy unnecessarily.

Similarly to checking the air quality, the process to check the light levels is in most ways the same, John just has to press the light sensor and all the data will be presented, as well as the button to turn the lights on or off.

Scenario 4 - Monitor Park Movement

Despite not being his main source of revenue, John wants to make a little bit of a profit running his parking lot.

He needs to be able to check the occupancy of the park at all times, as well as check the amount of money each car has to pay when leaving.

Given that he doesn't want every type of vehicle to pay the same amount, because some occupy more space than others, he needs to be able to set different fees for different types of vehicles.

He also doesn't want people to overstay their welcome, and not park their cars there overnight, to do so he needs to be able to apply a fine to people who do that.

To manage this aspect, John has to press the movement tab, where will be displayed the occupancy of the parking lot at that moment, he can also choose to check individual cars, by pressing the cars button, where a list of license plates will appear alongside a search bar to search for an individual one. When a particular car is chosen he can then check when the car entered the park and how much the car has to pay.

While on the movement page, he can also check the revenue by pressing the revenue button. While there the revenue for each month will be displayed and where the money was spent, he can also filter by day and year.

Product requirements (User stories)

Register park epic

John should be able to add a park

As a park owner
I want to add a park to the SnapPark app
So that I can know where parking lots are in my daily life and trips

Acceptance Criteria:
Given there is a park with a basic entry sensor
When I click "Add park" on the main page
Then the park should be visible in my park list and to those I made it visible

Jonh wants to add a sensor to a park

As a park owner
I want to add a sensor to a park in the SnapPark app
So that I can know measure certain statistics in my park

Acceptance Criteria:
Given there is a sensor in the park
When I click "Add sensor" on the park page
Then the sensor should be in the park sensor and it should see the data its recoding

Monitor park epic

John should be able to monitor park movement

As a park owner
I want to monitor vehicles entrances/exits according to their type/plate
So that i can alter park lotation and influence revenue by changing entrance fees.

Acceptance Criteria

Given the user wants to check lotation/change entrance fee
When he alters the value in front-end
Then the system alters the value and has a lotation/charges accordingly from then on.

John should be able to check Revenue

As a park owner
I want to check individual/total parks revenues
So that i can influence revenue.

Acceptance Criteria

Given the user wants to check the revenue of a individual/group of parks
When he selects the desired parks
Then the system will calculate and present the requested revenue.

John wants to monitor air quality

As a park owner
I want to monitor air quality in each park/floor
So that i can check and manipulate temperature/humidity values.

Acceptance Criteria

Given the user wants to monitor air quality
When he alters the values in front-end
Then the system alters the value internally making air quality devices make the needed changes.

John should be able to view the light levels of a park

As a park owner
I want to be able to view and control the light levels of any given park and floor i own
So that there is always good visibility in any floor

Acceptance Criteria

Given the park owner wants to monitor light levels
When he alters the values in front end
Then the system changes the values and adjust the light levels on the light fixtures on the park

3 Architecture notebook

Key requirements and constraints

Significant key requirements for the overall system's architecture:

- All profile and park data must be available to their respective park managers over a browser web application.
- Web frontend must interface with a REST API backend to fetch relevant domain data, which includes park-related and user profile data.
- Under periods of high intensity park traffic, the system should be able to efficiently and effectively track said traffic with no errors or false-positives.
- The system must be highly adaptable and integrate with a variety of different types of sensors (motion sensors, cameras, light sensors, air quality sensors, etc...) and their respective communication protocols.
- Sensors should be cheap, lightweight machines that must operate with complete availability, low latency communication and minimal mistakes.
- Park event delivery must be done with the least latency in communication and no failure or faults, especially with respect to message loss.
- Recent park traffic history, at most during the past 2 weeks, must be readibly available to view with minimal loading time.
- Park managers should be able to view park statistics over the long-term.
- Special events/alarms, e.g. payment infractions, details should be accessible on the long-term.
- The system must integrate with bank accounts to view revenue history and identify payment infractions.

Issues, constraints and expectations to take into account:

- Persisting park event history is expected to be relatively massive compared to user

and park profile information.
- Local park gateways are expected to suffer maintanability problems with the diversity of potential messaging and communication protocols used by different sensors.

Architetural view

Frontend:

- Web UI provided by SvelteKit servers.
    - Provides streamlined developer experience.
    - Flexible web page rendering options, including SSR and CSR.
    - Efficient, fast and scalable [A4].
    - Made for the Svelte UI framework.
    - Relatively new, but growing rapidly in popularity [A5].
- Svelte UI Framework.
    - Fast UI development.
    - Compiles to vanilla javascript, being significantly faster than other common UI frameworks which commonly rely on shipping a heavyweight virtual DOM to the browser.
- WebSockets for real-time notifications.
    - Special event alarms, such as payment infractions.
    - Live view of park traffic without the need for periodic page refreshing.
    - Pub-Sub interactions between the client and backend server.

Backend:

- Main framework: Spring Boot.
    - Opiniated view of the Spring framework optimized for rapid development and deployment.
    - Elimination of boilerplate and common manually-done configuration with automated configuration.
- REST API Service Controller.
    - Exposes all relevant application data for SvelteKit frontend servers to do SSR.
    - Uses the Service module for this objective.
- WebSocket service.
    - Provides a channel for web clients to subscribe to, where relevant park events

    - are published and propagated to every subscriber using the STOMP messaging protocol.
    - Consumes and publishes events using the event consumer module.
- Event consumer module.
    - Integrated in the backend server as a Java object.
    - Consumes processed events in the form of messages from a RabbitMQ broker.
    - Stores events in a relational database using the service module.
- Service module.
    - Based on Spring Data JPA's service objects.
    - Provides the application's business logic, interacting with the Repository module.
- Repository module.
    - Based on Spring Data JPA's repository objects.
    - Provides the persistence logic of the application.
- Banking Module
    - A generalized interface for revenue monitorization using bank account APIs.

Database

- CockroachDB relational database
    - Highly performant, powerful database with strong ACID requirements compliance.
    - Integrated in Cockroach Labs' distributed database cloud service.

Event Communication

- Message-oriented middleware using message queues hosted in RabbitMQ message brokers.
    - Processed event data is pushed into a message queue using the AMPQ 0-9-1 messaging protocol.
    - Lightweight, easy deployment in the cloud.
    - High variety of different programming language drivers, including Java and Python.

Park Sensors

- Sensor Box.
    - Local gateway for sensors to send their unprocessed data to.
    - Adapted to handle the diversity in communication protocols used by park sensors.
    - Processes raw sensor data into coherent events, pushing them to a message queue.
    - Aggregates processing and communication responsibilities, focusing sensor

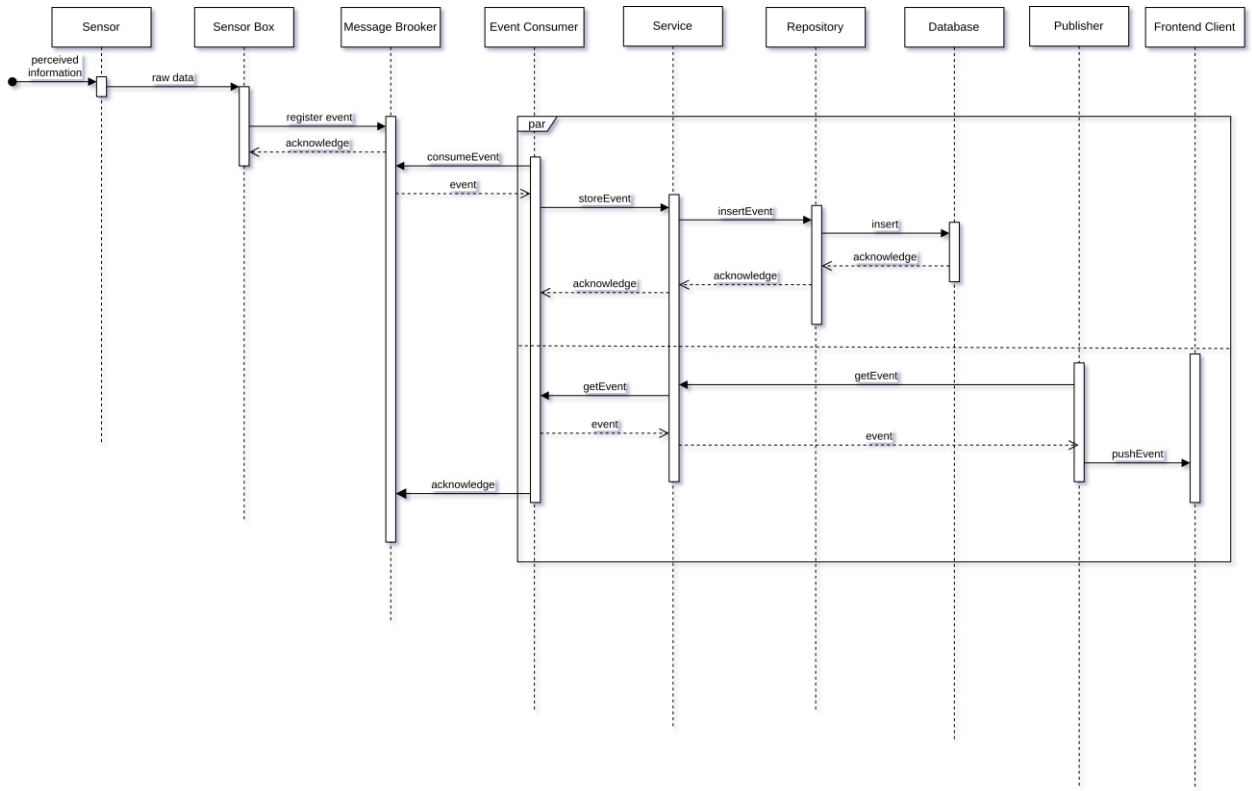
computing power on their main critical tasks.

- Implemented in Go, a language optimized for concurrency tasks and networking, with a vast ecossystem of tools for these purposes.

Module interactions

General Event Handling

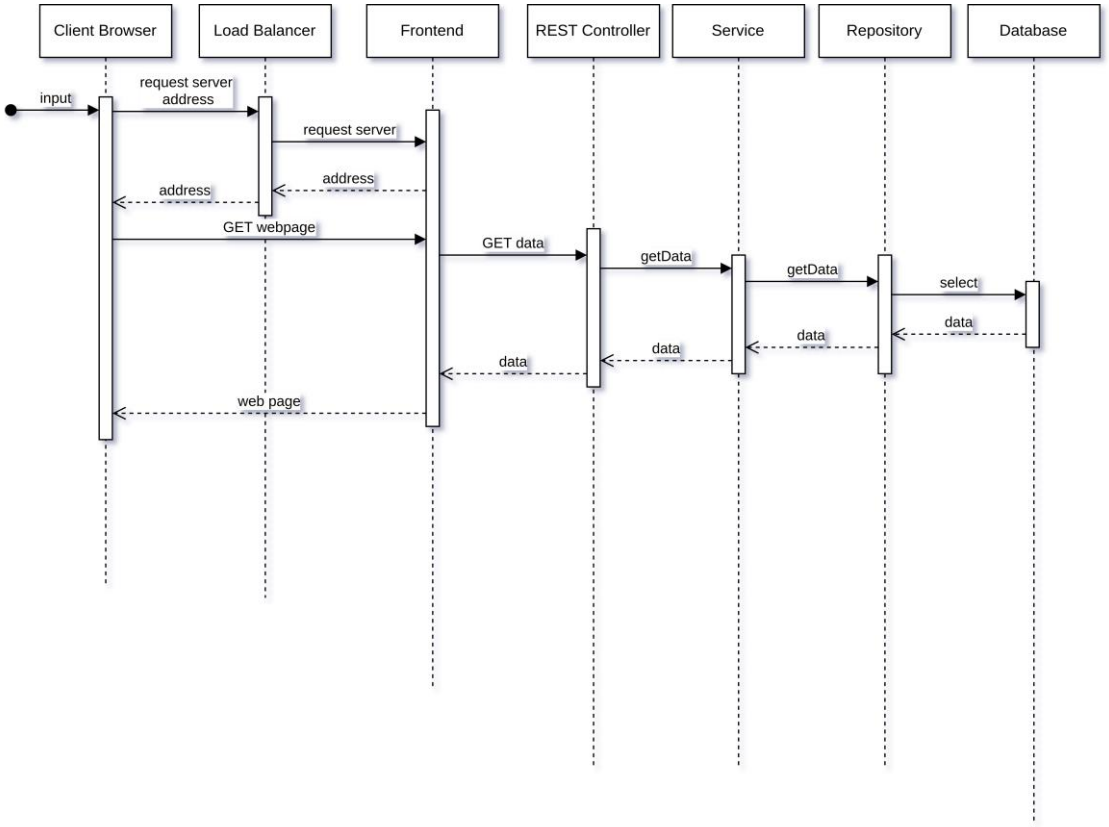
Events come with a certain variety, but all share the same or similar interaction dynamics as represented by the following sequence diagram.



UI Fetching

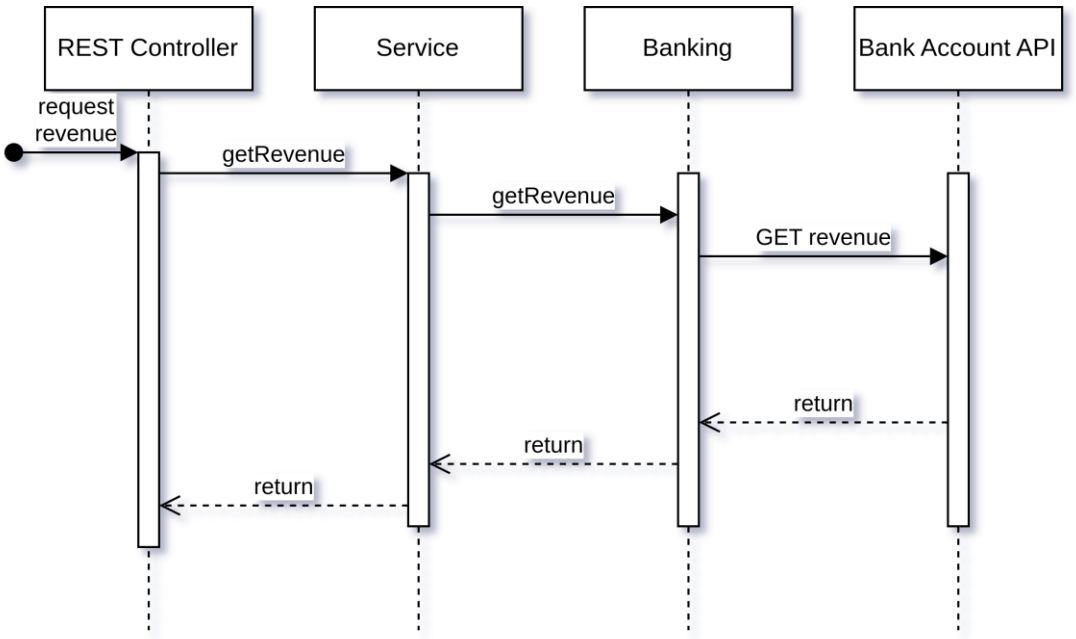
Fetching the required data (park data, user data, etc...) to render on the server side and delivering the web page to the client. The following representation doesn't include the event notification pushing mechanism that was already represented in the previous diagram.

This diagram is meant to represent data fetching interactions in a more generalized form, as fetching park data and user profile data present very similar module dynamics.



Revenue Statistics

To monitor revenue, the application system must rely on an external bank account's API.



→ dicuss more advanced app design issues: integration with Internet-based external services, data synchronization strategy, distributed workflows, push notifications mechanism, distribution of updates to distributed devices, etc.>

Advanced Issues

There are some architecture details that need further discussion due to their more intricate and complex nature in this context:

- Bank account integration.
  - Implies broad compatibility with many banking systems.
  - Should be built on popular banking solutions first.
- Database Sharding
  - Database sharding should be done for each park. This is because parks, combined with their event history, aggregate the most data and relations out

of all other relational entities, excluding the manager.
    - This also makes it significantly easier to geographically distribute instances, as they will be instantiated as close to the park's real location as possible.
    - Sharding by manager isn't viable as a manager will most likely be responsible for several parks, at which point it may be an excessive load on one single instance.
    - Replicating manager profile data across instances is acceptable as its volume is predicted to be low, while also offering redundancy and a consequent lower latency when fetching user data.
    - Sharding can be done manually and automatically via Cockroach Labs' cloud service.
- Message Broker Distribution.
    - Message brokers should be distributed to offer the majority of sensor boxes low latency communications.
    - The distribution strategy should seek to identify, for N brokers, N regions of sensors boxes.
    - For each region, place a broker in a position that minimizes its distance with all of its sensor boxes.
- Message Broker Replication
    - With respect to relatively high traffic brokers, a cluster of multiple replicated and synchronized brokers should be setup for fault tolerance and redundancy purposes.
- Frontend and Backend server distribution
    - Servers should be distributed using a similar strategy to Message Broker distribution, now focused around client connections.

4 Information perspetive

5 References and resources

USAGE SCENARIOS:

https://www.atlassian.com/agile/project-management/epics-stories-themes

HARDWARE:

https://www.tomshardware.com/reviews/raspberry-pi-4

ARCHITECTURE:

[A1] https://www.highgo.ca/2021/08/09/horizontal-scalability-options-in-postgresql/

[A2] https://phoenixnap.com/kb/distributed-database

[A3] https://www.turing.com/resources/backend-frameworks#factors-to-consider-when-choosing-a-backend-framework

[A4] https://dev.to/ahmed_onour/sveltekit-what-is-it-and-why-should-i-care-63j

[A5] https://naturaily.com/blog/why-svelte-is-next-big-thing-javascript-development

[A6] https://www.gartner.com/reviews/market/cloud-database-management-systems/vendor/cockroach-labs/product/cockroachdb/likes-dislikes

[A7] https://medium.com/@radoslav.vlaskovski/cockroachdb-serverless-free-tier-analysis-70747ec64ad8