Politecnico di Milano

Department of Computer Science and Engineering

Software Engineering 2

CLup – Customers Line-up Design Document

December 29, 2020

Student
Damiano Derin

Student
Jas Valencic

Contents

Chapter 1

Introduction

1.1 Purpose

1.2 Scope

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

Customer	a person who buys goods from the stores. We will use the term *customers* to refer to natural persons, instead the term *users* will be used to specify the virtual entity served by the application.
Store Manager	a person who is in charge of the store. In our context, we assume that the *store manager* controls the entrances to the store with the help of CLup service. In the real world scenario this activity can be delegated, without loss of generality.
Physical Spot	a digital device positioned outside the store that allows customers to obtain tickets to line up.
User	a virtual entity that interacts with the virtual service offered by CLup. The user can be a customers, a store manager and a physical spot (when it is acting as proxy). In case of ambiguous interpretations, we will specify the real entity name.

Proxy	an intermediary entity that exchanges information between two other entities. In our system, the physical spot can be seen as a proxy, since it allows customers to line up without the necessity to create an user account. From the point of view of the server, the physical spot is seen as an user.
Virtual Queue	a queue of users allocated in the memory of the server. When a user asks for a lining up operation, or a booking a visit operation, it is allocated in this queue.
Physical Queue	a queue of customers outside the store.
Ticket	a piece of paper or a virtual card given to customers to show that they have performed a lining up or a booking a visit operation.
QR code	a matrix composed by white and black squares encoding a string. It is reported on the ticket.
System	we use this term to represent the entire service, composed by smartphone application and servers.
Application	program executable on smartphone.

1.3.2 Acronyms and Abbreviations

API	Application Programming Interface
CLup	Customers Line-up
DBMS	Database Management System
JSON	JavaScript Object Notation
MVC	Model-View-Controller

1.4 Revision History

1.5 Reference Documents

1.6 Documents Structure

Chapter 2

Architectural Design

2.1 Overview

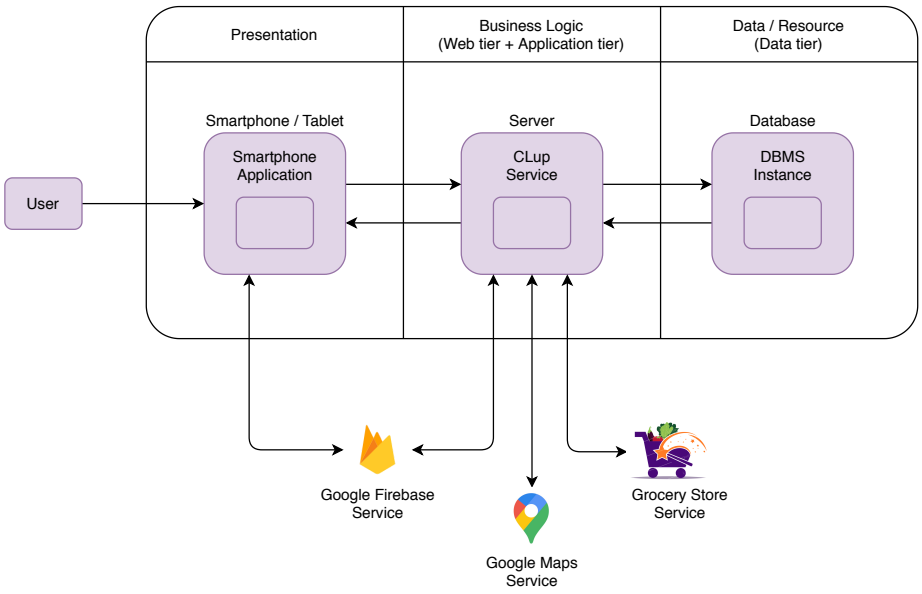

Figure 2.1: System architecture.

2.2 Component View

The previously introduced components will be presented in details in this section by using the component diagram 2.1 that has been divided in three main sub-components to better understand the relation between the three tiers of the system.

The presentation tier has been represented by the **EndUserApp** that generalizes the architecture of the smartphone application. It encapsulates the **Model-View-Controller (MVC)**, that has been chosen as design pattern,

and the **HostService** which is the module that allows the application to use the Application Programming Interfaces (APIs) offered by the device.

The business tier (web tier & application tier) is the most complex part of the diagram, therefore it has been split in different layers of components:

- **WebLayer**: contains the **RequestHandler** component that receives the requests from remote devices and forwards them to the application tier. The main functionalities are: *decryption*, *parsing* and *forwarding* of the requests; *reformatting* (in JavaScript Object Notation (JSON) format) and *encryption* of the responses before sending them to the clients.
- **ApplicationLogicLayer**: contains the modules for the functionalities offered by the system. They are:
  - **ServiceHandler**: is the module that coordinates the others. It receives the requests from the previous layer and checks if sessions are active, communicating with the SessionService. After that, it calls the other services based on the request type. In the end, it returns the responses to the WebLayer.
  - **SessionService**: is the service that monitors the session by updating and releasing tokens.
  - **StatisticsService**: is the service that creates charts for the store manager to analyze the trend of the queues. It interacts with the DataLayer to retrieve data from the database.
  - **QRCodeCheckingService**: is the service used when customers scan the QR codes to verify that the ticket numbers are the same that have been notified by the system. It is called, in background, by the application running on the store manager device.
  - **LogInService**: is the service that controls and updates the status of the tokens. If an user has been recognized, by the system, it changes the status of the token in the database. In this way, the user will be identified as logged and he will be able to request for services that requires an authentication. This service is used for log out too.
  - **SignUpSerivice**: allows users to create an account.
  - **ControlQueueService**: is the service used by the store manager to change parameters that modify the sequence of events in the Scheduler. From this service the store manager can modify the timing in which tickets are released, he can interrupt the release of tickets and he can change other parameters of the scheduler algorithm. This service communicates with the Scheduler and the DataLayer to store the new parameters.
  - **BookingAVisitService**: is the service that allows customers to book a visit. It handles the data passed from the clients, performs some checks on the inserted information and communicates with the Scheduler and with the DataLayer.
  - **LiningUpService**: is the service that allows customers to line up. As for the BookingAVisitService, it passes data to the Scheduler and the DataLayer.

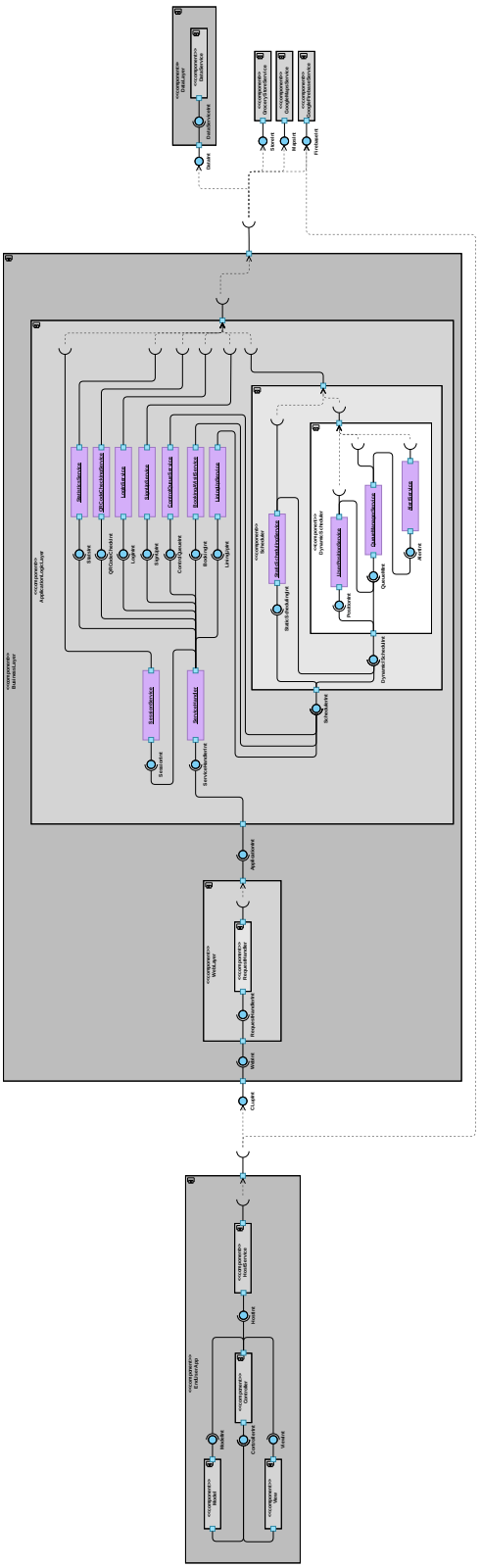

Table 2.1: Component Diagram.

The main core of the ApplicationLogicLayer is the **Scheduler** that implements the algorithm used by the system to create and keep updated the virtual queue of users. It is composed by a StaticSchedulingService and a DynamicScheduler.

- The **StaticSchedulingService** is in charge of performing few checks before asking to the DynamicScheduler to insert customers in the queue. Moreover, it is the first sub-module to be called by the LiningUpService and the BookingAVisitService. It communicates with the external services to compute a raw estimation of the time in which a customer will be called by the system to be authorized to enter in the store.
- The **DynamicScheduler** is composed by different sub-services that are necessary to update the virtual queue in background and to notify customers. The main sub-service is the **QueueManagerService** that handles the queue and coordinates the other modules.

The data tier has been represented by the **DataLayer** that contains the **DataService**. It interacts with the Database Management System (DBMS) to execute queries.

In the diagram has been reported the external services used by the system:

- **GroceryStoreService**: to obtain data about customers from the store.
- **GoogleMapsService**: used to compute the estimated time to arrive to the store from the location of the customer.
- **GoogleFirebaseService**: used to notify customers.

2.3 Deployment View

2.4 Runtime View

In this section we are going to present how the components interact at runtime.

2.4.1 Session Control

The figure [1] 2.2 shows the sequence diagram associated to the operations performed by the SessionService module to generate and to control the validity of a given token. The tokens are part of the request parameters, they are used to keep sessions alive with users. The SessionService checks if a token already exists; if it is then the module controls if it is valid or expired, otherwise it will generate a new one. Authentications and tokens are two different things: an user can have a token but he might be unauthorized (not logged or he could not have an account yet). The authorization status, associated to a token, will be updated by the LogInService or by activities performed in background by the system.

[1] For all the sequence diagrams, a custom notation has been used to show the input parameters of the methods (**data). It follows a programming language notation to represent inputs in a more compact way. In Python style, it can be interpreted as a dictionary data structure.

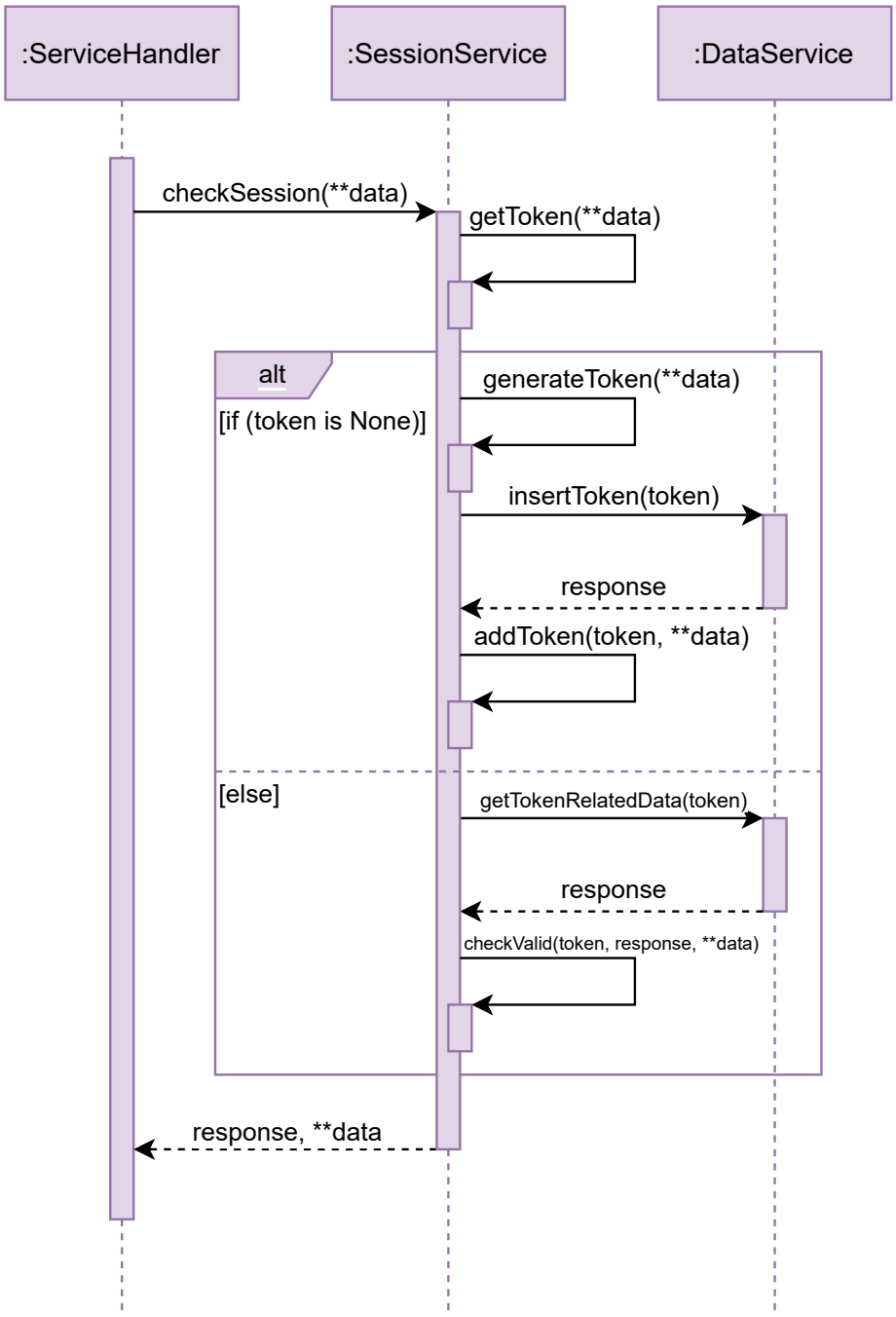

Figure 2.2: Session control sequence diagram.

2.4.2 Sign Up

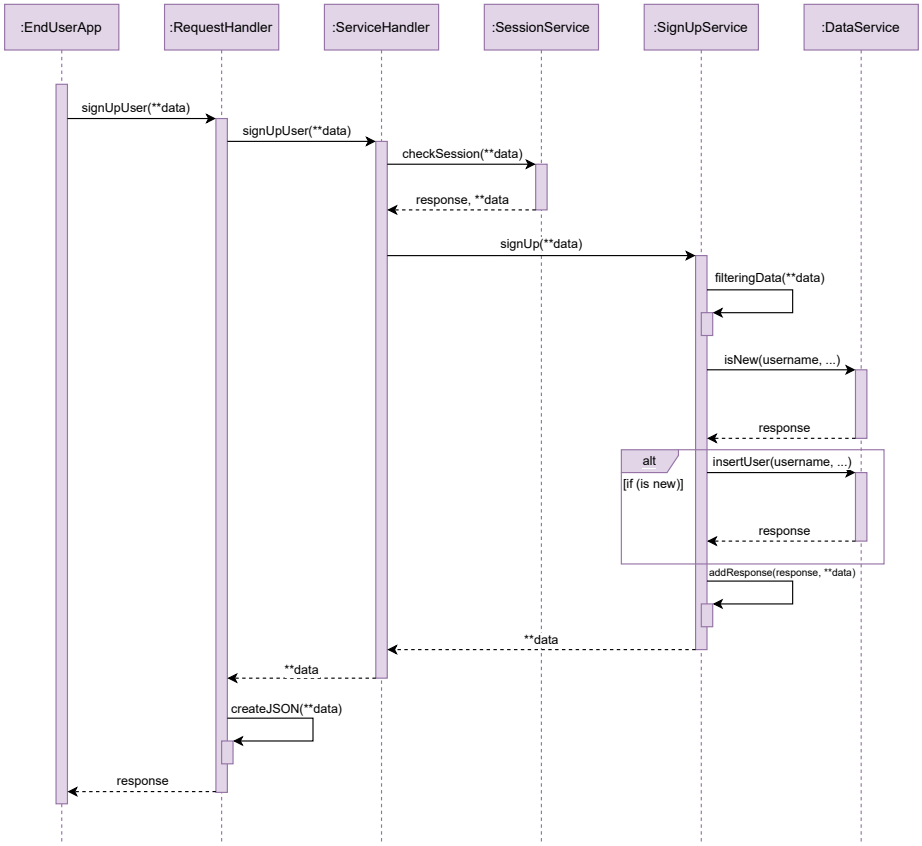

Figure 2.3: Sign Up sequence diagram.

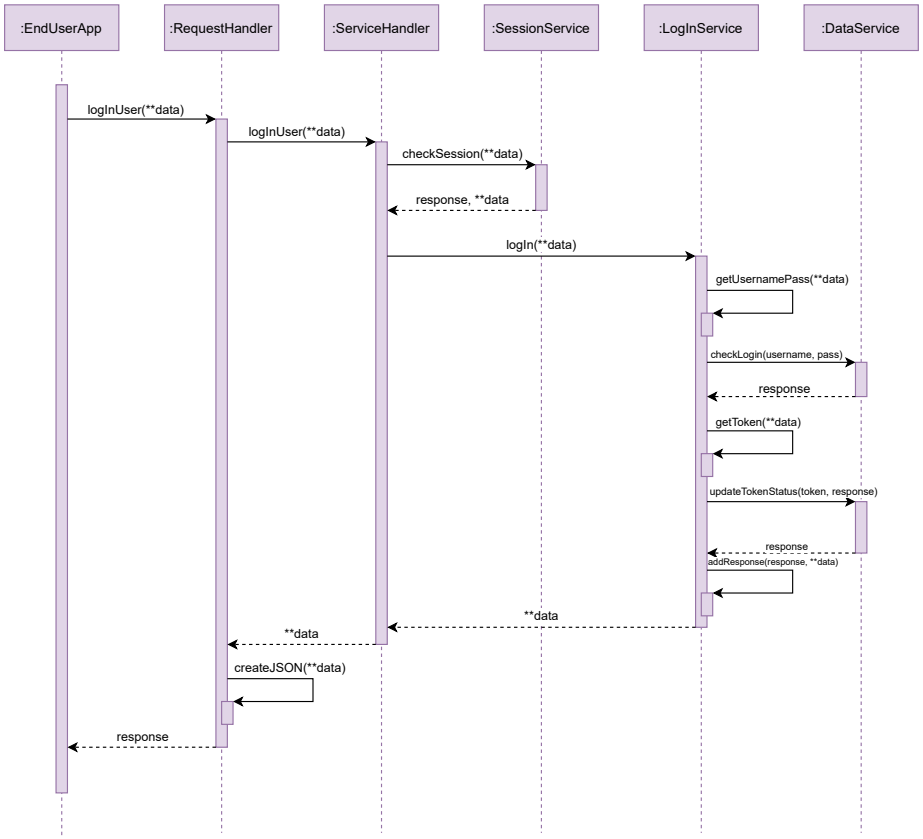

Figure 2.4: Log In sequence diagram.

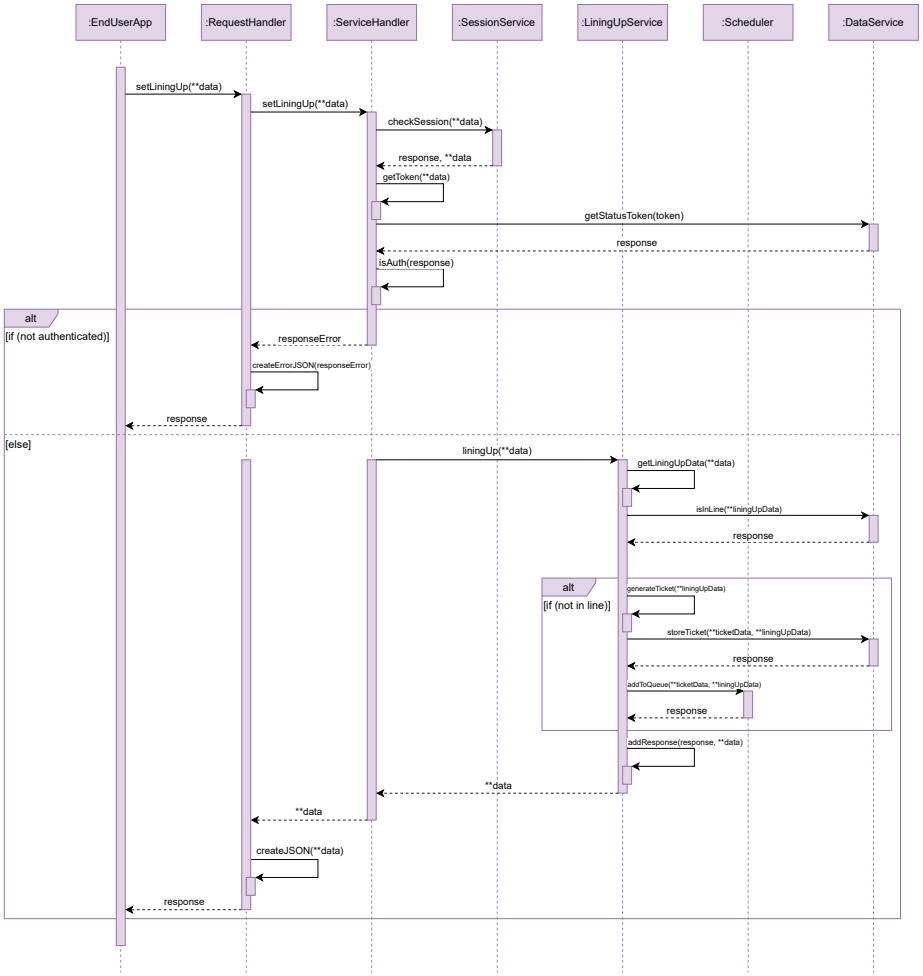

Figure 2.5: Lining Up sequence diagram.

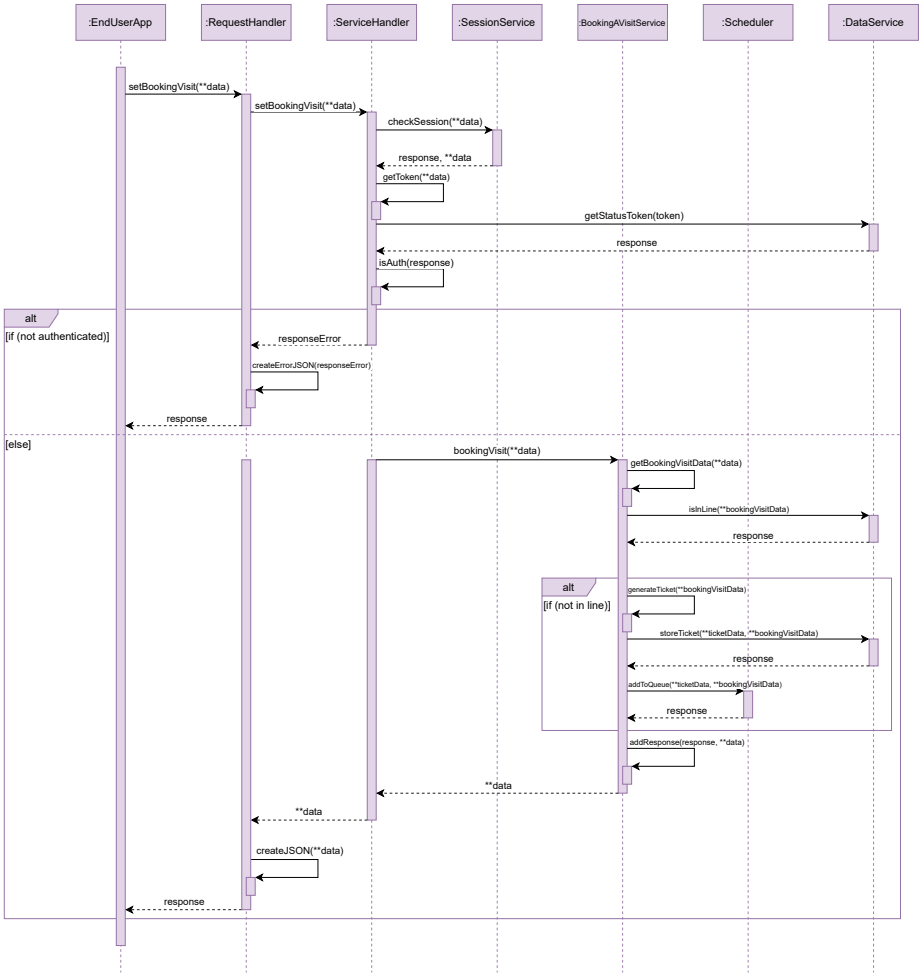

Figure 2.6: Booking a Visit sequence diagram.

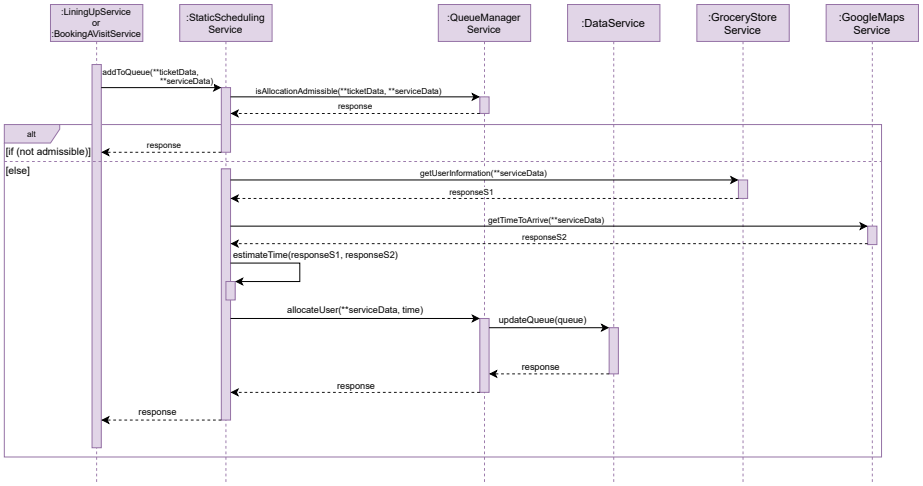

Figure 2.7: Scheduler sequence diagram.

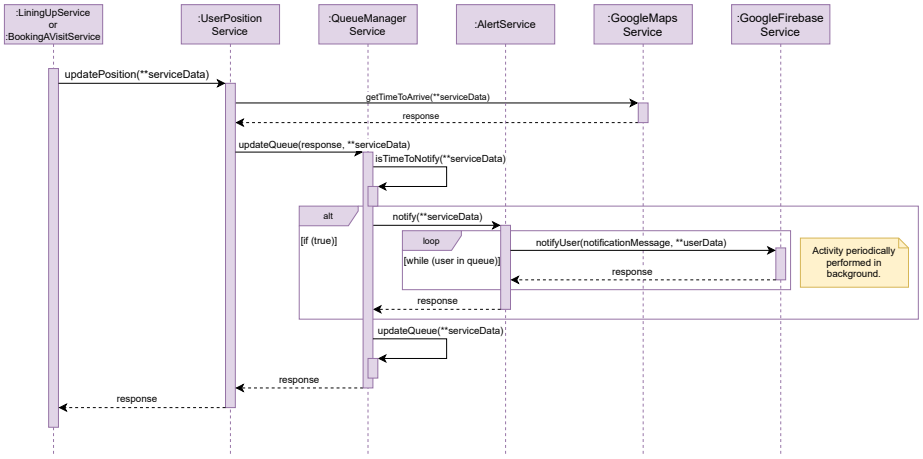

Figure 2.8: Dynamic Scheduler sequence diagram.

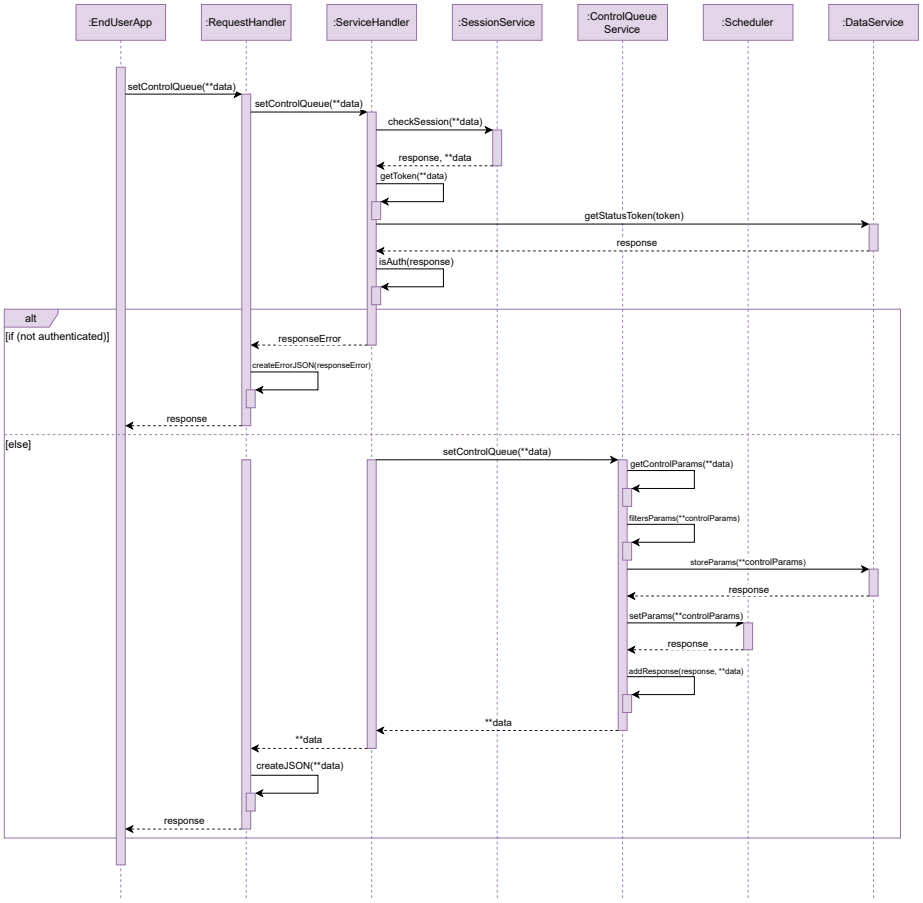

Figure 2.9: Control Queue sequence diagram.

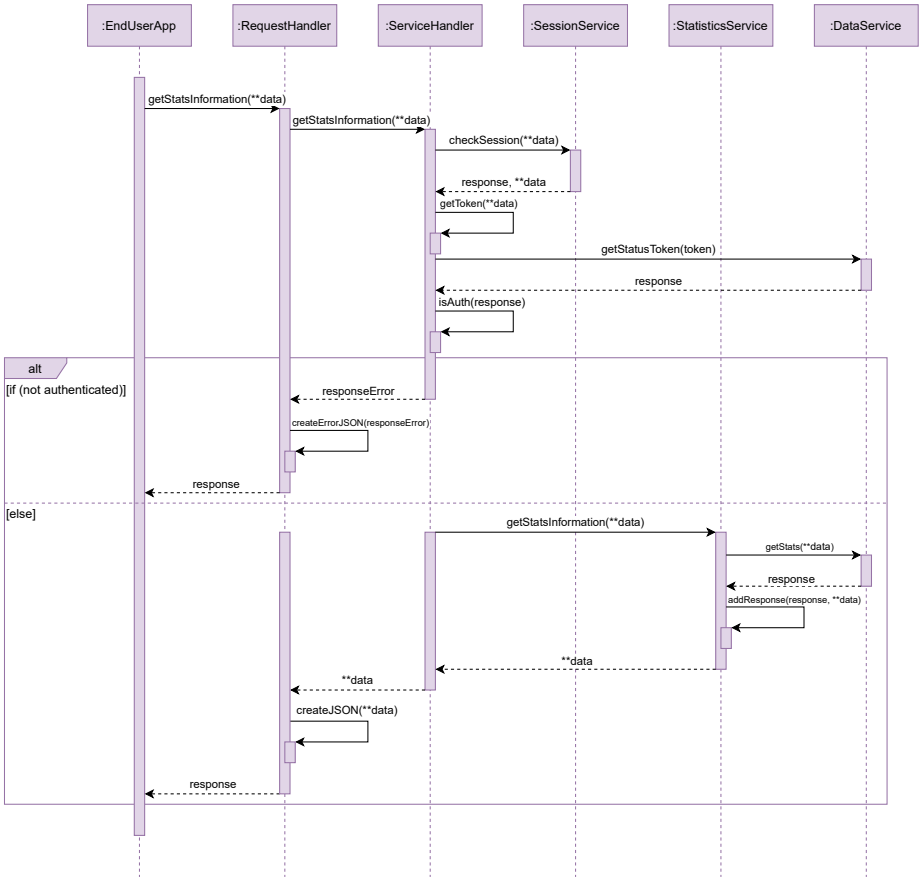

Figure 2.10: Show Stats sequence diagram.

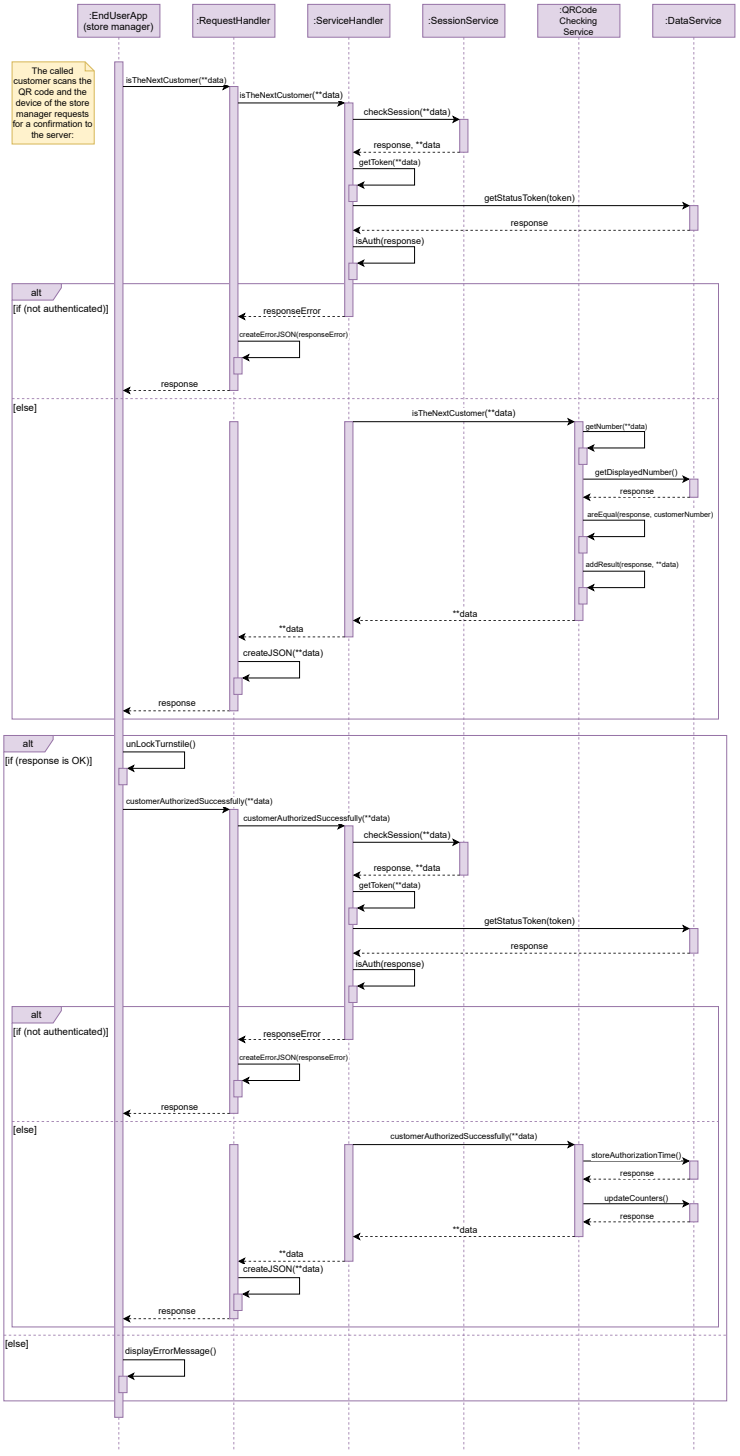

Figure 2.11: QR Code Checking sequence diagram.

2.5 Component Interfaces

2.6 Selected Architectural Styles and Patterns

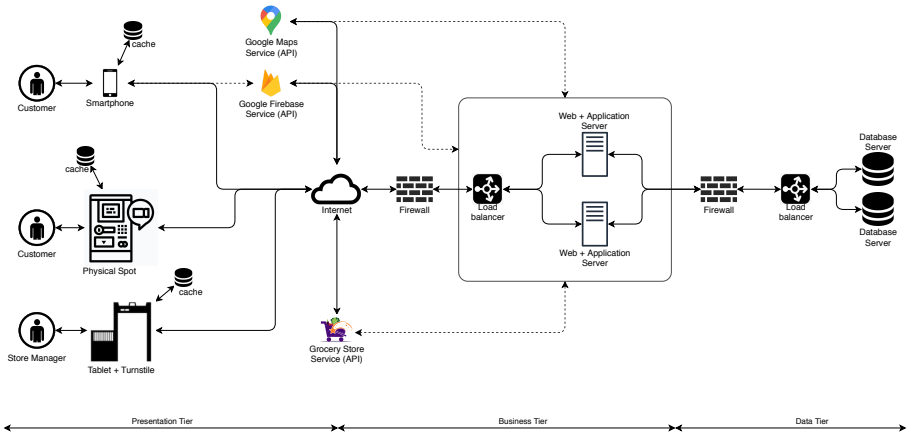

Figure 2.12: Architecture components.

2.7 Other Design Decisions

Chapter 3

User Interface Design

Chapter 4

Requirements Traceability

Chapter 5

Implementation, Integration and Test Plan

Chapter 6

Effort Spent

Chapter 7

References

- Specification document: "R & DD Assignment AY2020-2021.pdf".
- Slides of the lectures.