Politecnico di Milano

Department of Computer Science and Engineering

Software Engineering 2

CLup – Customers Line-up Design Document

January 5, 2021

Student
Damiano Derin

Student
Jas Valencic

Contents

Chapter 1

Introduction

1.1 Purpose

The purpose of this document is describing the Customers Line-up (CLup) system in a more specific way than the Requirements Analysis and Specification Document (RASD). Therefore, it is suggested to read ahead the RASD. This document is aimed both at stakeholders to view a preview of the final product, and at project managers as a guideline to follow. It describes the system by both showing high-level and in-depth analysis of it, and how these analyses affected the architectural design choices made. It also shows the division of the system in components describing how they work and interact, how they are necessary for the fulfillment of the requirements and how they will be implemented and in which order.

1.2 Scope

Our application helps stores to prevent the diffusion of a virus, in a global pandemic situation, keeping customers in safety conditions. One of means to contribute to the success of this goal is to guarantee the absence of crowds. Considered one store, and established the maximum number of people that are allowed to be inside, our application is an instrument to keep the influx of people below that threshold. It works managing automatically the influx of people inside the store, staggering the entrances in the store by using some parameters to keep them safe. This is done by offering to customers a service that consist in a digital lining up system. It is designed to work remotely, but the possibility to line up physically is offered too. In this way, it should help to avoid crowds outside the store. Moreover, to reach the largest number of people, this application should be easy to use and its functions should work consistently in time. Instead, to the manager of the store it is given a dashboard where are shown significant data like the number of people currently in the store, the status of the queue, the influx of people during different intervals of time, etc. It also allows, when it is necessary, the store manager to make decisions that influences the flux of people (like for example, blocking the entrances for a period or modifying some parameters which the algorithm uses to manage the flux). Furthermore, it is given to the customers a function that allow them

to book a visit to the store specifying a time slot. This is thought to help people with limited availability during the day. This option requires optionally at people to point out which product categories they are going to purchase or the departments they are going to visit. This is used to optimize the algorithm of the system to maximize the number of people inside the store during a certain time, by knowing their distribution. This can also be made automatically by using their data for long-time customers.

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
d.P.C.m	*"decreto del Presidente del Consiglio dei ministri"*
DBMS	Database Management System
DD	Design Document
GUI	Graphical User Interface
JSON	JavaScript Object Notation
MVC	Model-View-Controller
RASD	Requirements Analysis and Specification Document
REST	Representational State Transfer
UX	User Experience

1.4 Revision History

Date	Modifications
January 5, 2021	First version

Table 1.1: Revision history table.

1.5 Reference Documents

- Specification document: "R & DD Assignment AY2020-2021.pdf".
- Slides of the lectures.
- RASD: "CLup - Customers Line-up Requirements Analysis and Specification Document.pdf"
- Design Documents (DDs) of past students.
- Google Maps services: "https://cloud.google.com/maps-platform"
- Google Firebase service: "https://firebase.google.com/"
- Architecture styles: "https://docs.microsoft.com/en-us/azure/architecture/guide/architecturestyles/"

1.6 Documents Structure

Chapter 2

Architectural Design

2.1 Overview

The architecture that was chosen for our application is the three-tier one, because we think that this division is the most natural to represent it and is composed of:

- **Presentation tier**: contains the user interface and the communication layer of the application. Its main purpose is to display information and collect information from the user.
- **Business tier**: is the core of the application. In this tier, information collected in the presentation tier are processed, sometimes against other information in the data tier to offer the different functionality and services.
- **Data tier**: is the tier that contains the database, and so its job is to store persistently and manage data processed by the application tier and make it available back at the application tier.

Since the application interface is light and does not require too much computational power to work. Instead the main services offered by our application requires the coordination of all clients a to manage the queue in real time and requires the elaboration of their real time data, and server data to predict their behavior. It needs so a central component that has a high computational power to manage it and the logic tier describes it perfectly. The application needs to record a lot of data about the customers and so a data tier is required, this data will not only be collected by the application, but it will be also collected from the supermarket chain database. We chose the three-tier architecture also for its properties that are:

- **Scalability**: any tier can be scaled independently as needed without affecting the others.
- **Security**: since the presentation tier and the data tier are not directly connected, it is possible to implement firewalls preventing different kind of unwanted exploits

- **Maintainability**: since the three tiers works independently from each other, updates and maintenance can be done one tier at time without affecting the others and by only one team.

In our case the three tiers will contain the following division:

- **Presentation tier**: will contain the interfaces showed in the mock-ups, and it will allow the users to send request for creating, deleting, and viewing line-up and booking ticket from them application. It will allow the physical spot to create and print the line-up ticket. It will allow the manager to do query on different kind of data and to interact with the realizing of new ticket and people entrances.
- **Business tier**: will contain the function to manage in real time the queue considering the realizing time of the different kind of tickets, the position of the application-users, the presence of booking tickets. It will also contain the function to handle and to redirect the different kinds of the request on the appropriate component. And also, the function to check the QR code and allow user to enter, the function to build the visual representation of manager query, and the sign-up and login services.
- **Data tier**: will contain the information about the users, their behavior (tendency to arrive later or early, typical staying time, etc) and all information that could help to improve the accuracy of predictions.

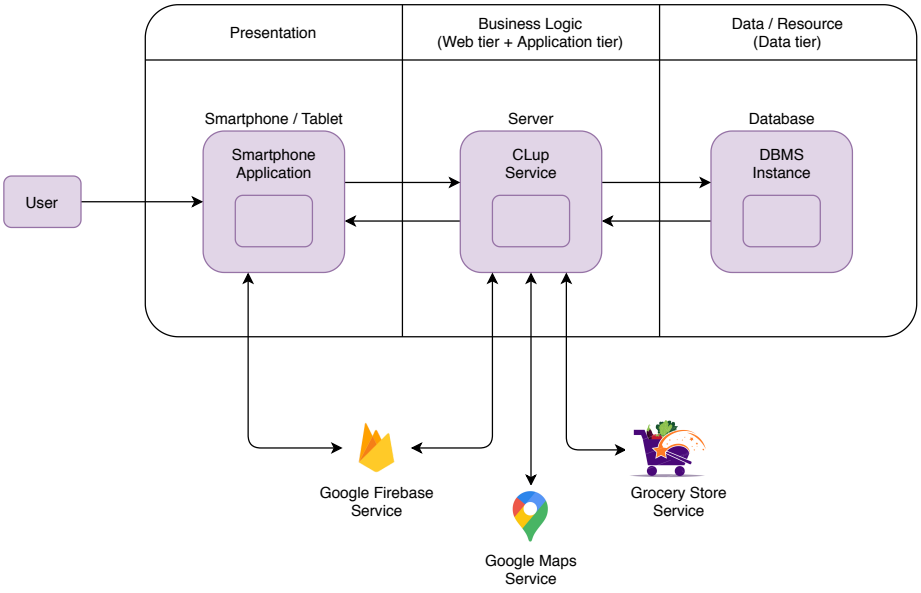


Figure 2.1: System architecture.

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 can be activated by activities that are running in background 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.

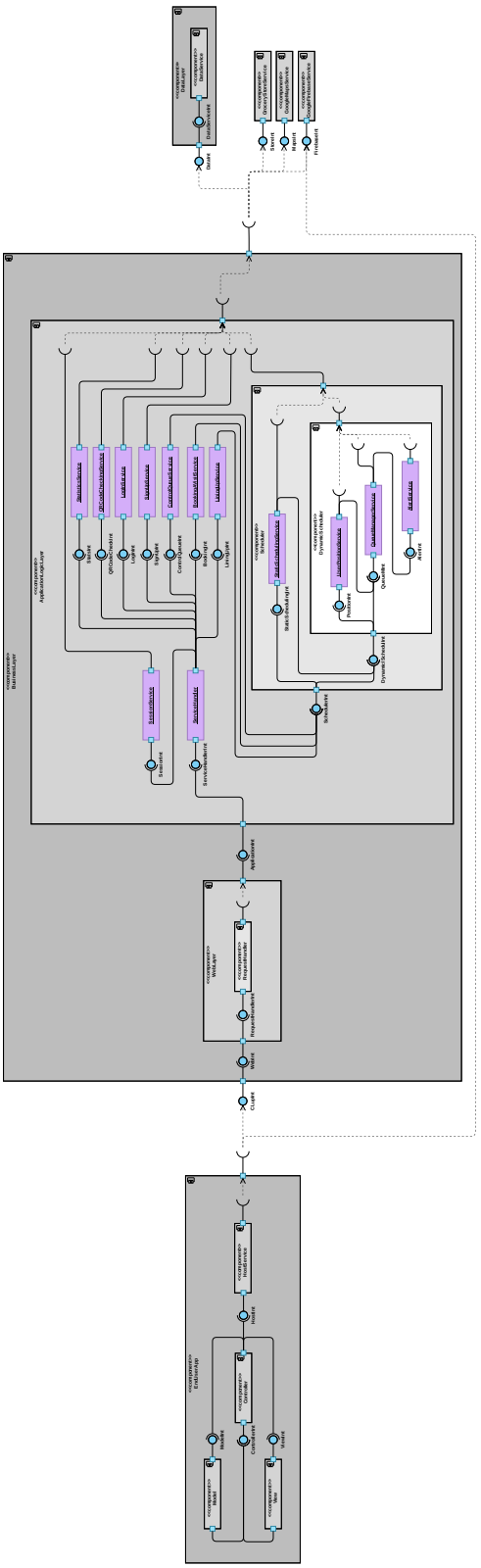


Table 2.1: Component Diagram.

  - **LiningUpService**: is the service that allows customers to line up. As for the BookingAVisitService, it passes data to the Scheduler and the DataLayer.

  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, and it includes the interface of the hardware component of 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

- **Presentation tier**: in this tier we have the interfaces for the different users, the application contains all the interfaces and when the user log in the interfaces change to the proper one. For the customers, the interface permits the user to connect to the BookAVisitService, LiningUpService and ControlQueueService. In a similar way the physical spot interface permits it to connect to the Lining up service and to the feature to print. And for the manager the interface, it permits him to connect and use StaticsService, QRCodeCheckingService and ControlQueueService.

- **Business tier**: in this tier we have the application server that manages all the functions of CLup. Starting from handling all the user request and redirect them to the right service required, to sending request to database to retrieve and record data. Inside this server the function the require more resources are the handling of the request and the scheduling service. Since it requires to schedule first the different kind of user and then if possible, monitor their position in line and compute it to their records of past visit.

- **Data tier**: in this tier the data service handles all the request coming from the application tier. It schedules the different request of transaction from the different services and manages it accordingly. So, it let them store, manage and query data of the Database. The model of DBMS used in this application is the relational one since it is the most well-known and used one. Therefore, there are a lot of tools and APIs up to date to support the work. It also fits the application requirements since it permits to compute quickly complex queries and help the data organize in a well-structured way. The first especially is needed for the scheduler, since it has to re-compute the data a lot of times during a day.

In the application tier we have two main device that can contain the application, in particular the smartphone is associated to customers and the tablet to the physical spot and the store manager. The smartphone application also interacts via Google API with Google Firebase Service to receive the notifications from the system. The application of both the devices can connect to the application server, which must first pass through a firewall, to increase the security of the system and then through a load balancer to distribute the workload. The application server is connected to both Google services also through a firewall to retrieve the position of the user in line and to send notifications. It is connected to the grocery store service to retrieve information about user's behavior. And it is connected to also to CLup database to retrieve stats, queue information, reservation, etc.

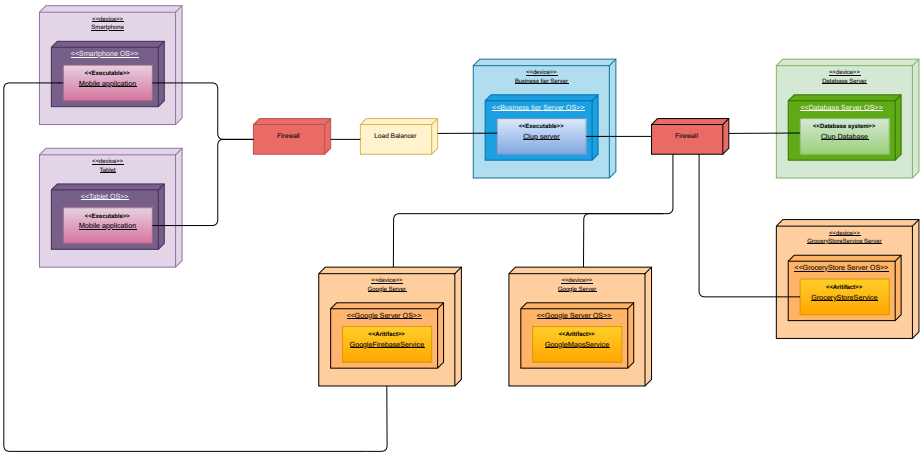


Figure 2.2: Deployment diagram

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] 2.3 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.

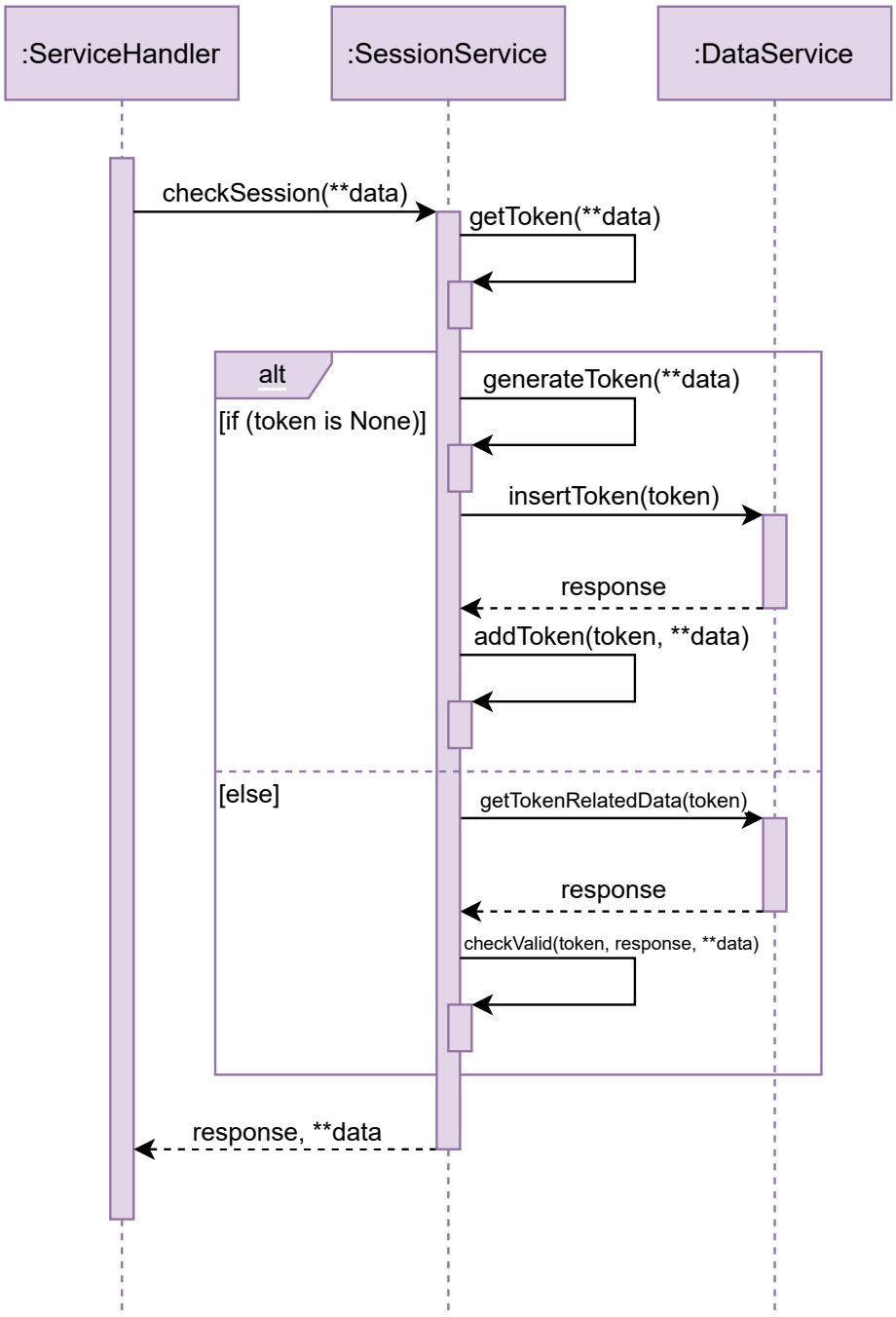


Figure 2.3: Session control sequence diagram.

2.4.2 Sign Up

In figure 2.4 has been shown the procedure to register a new user. The core of the sequence diagram is the SignUpService that filters the data inserted by

the customers and controls if the credentials have already been used for other accounts. If possible, the service creates the new account by inserting the credentials, and the other customers information, in the database. Results of the queries and the other executed methods are appended to the data that are back-propagated to the RequestHandler that creates the JSON response.

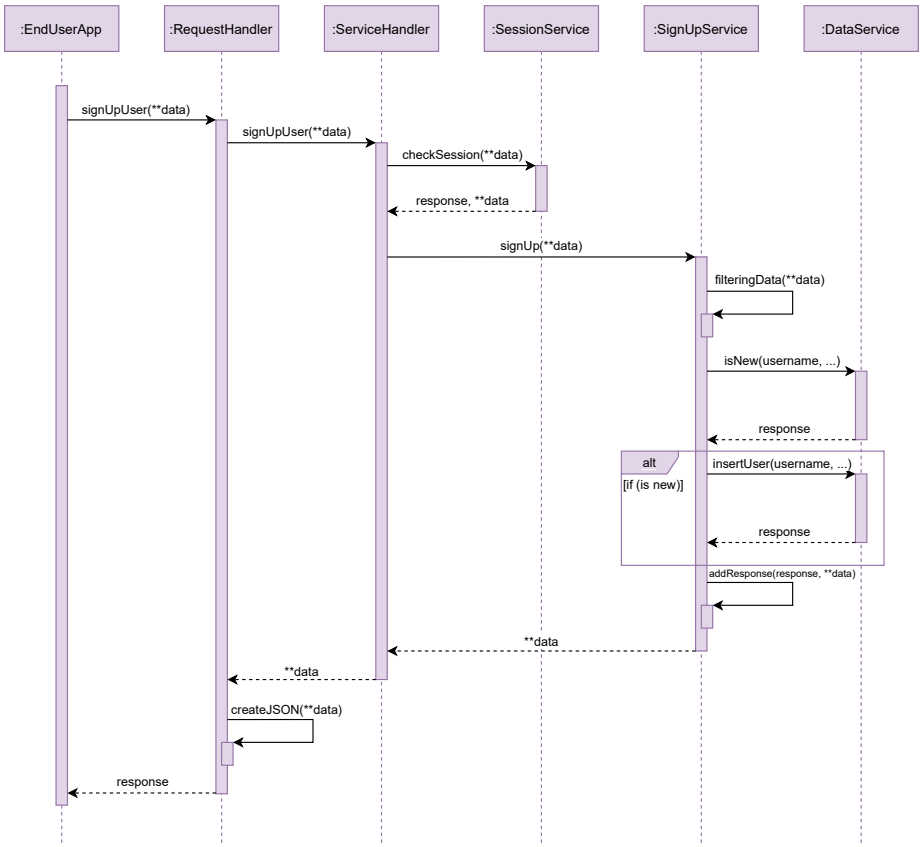


Figure 2.4: Sign Up sequence diagram.

2.4.3 Log In

Similarly to the sign up, the figure 2.5 shows how the log in is performed. In this case the credentials are parsed from the request parameters and a verification is performed. If the pair username-password is present in the database, then the status of the token, assigned to the customer, will be updated to remember that the user has been authenticated.

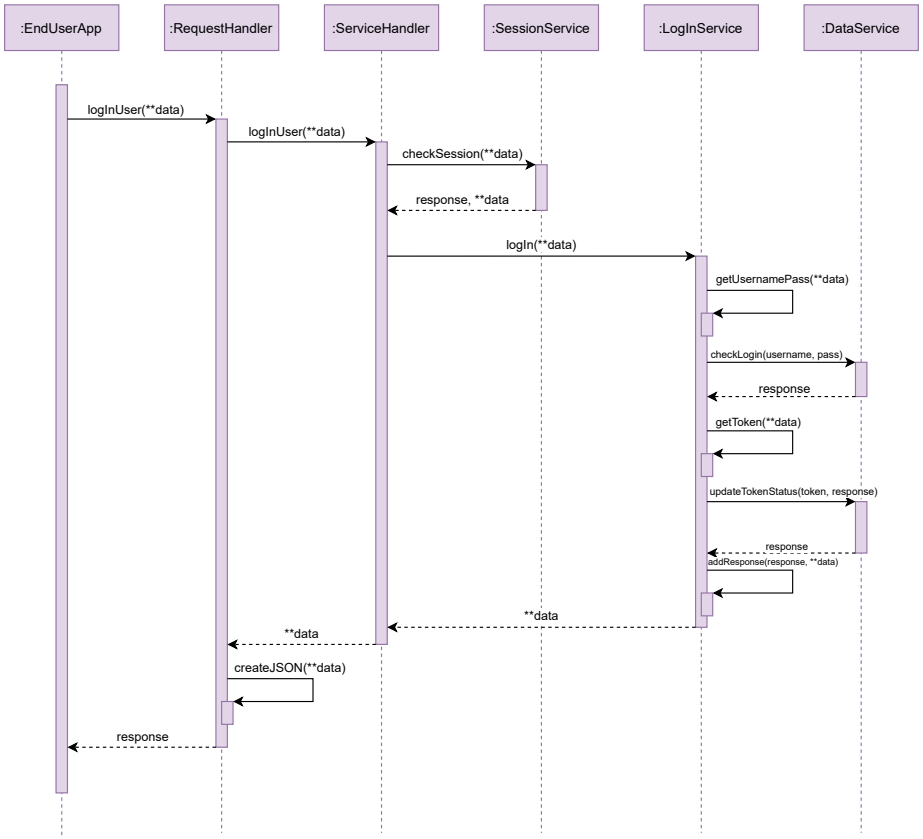


Figure 2.5: Log In sequence diagram.

2.4.4 Lining Up

In figure 2.6 the lining up operation has been reported. The user, to be allowed to perform a lining up, has to be registered and authenticated, therefore the ServiceHandler checks the token validity, communicating with the SessionService, and controls if the user is authenticated by getting the status of the token from the DataService. In case of valid token with positive status (user before logged in), the request is forwarded to the LiningUpService, which after few controls, can decide to pass the lining up information to the Scheduler. In any case the RequestHandler will reply to the clients with a response that it will contain lining up information (such as the QR code) or an error message.

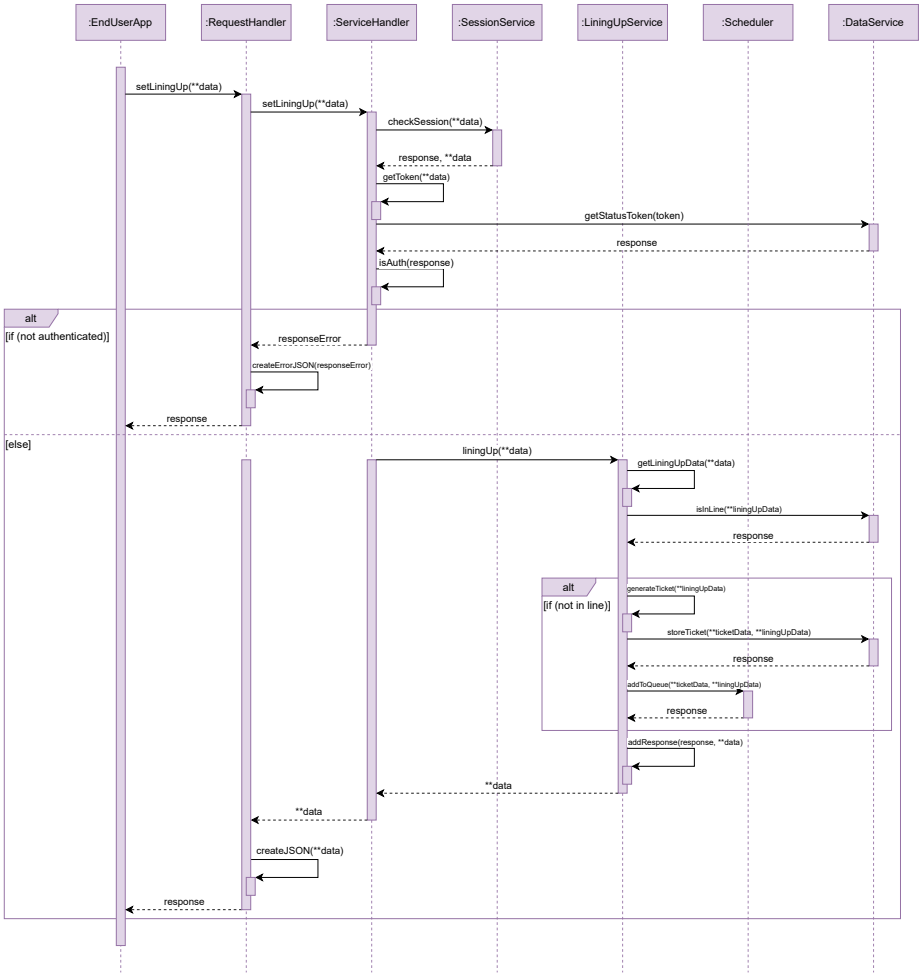


Figure 2.6: Lining Up sequence diagram.

2.4.5 Booking a Visit

The sequence diagram 2.7 is similar to the previous one. The differences are in the kind of information passed from the EndUserApp.

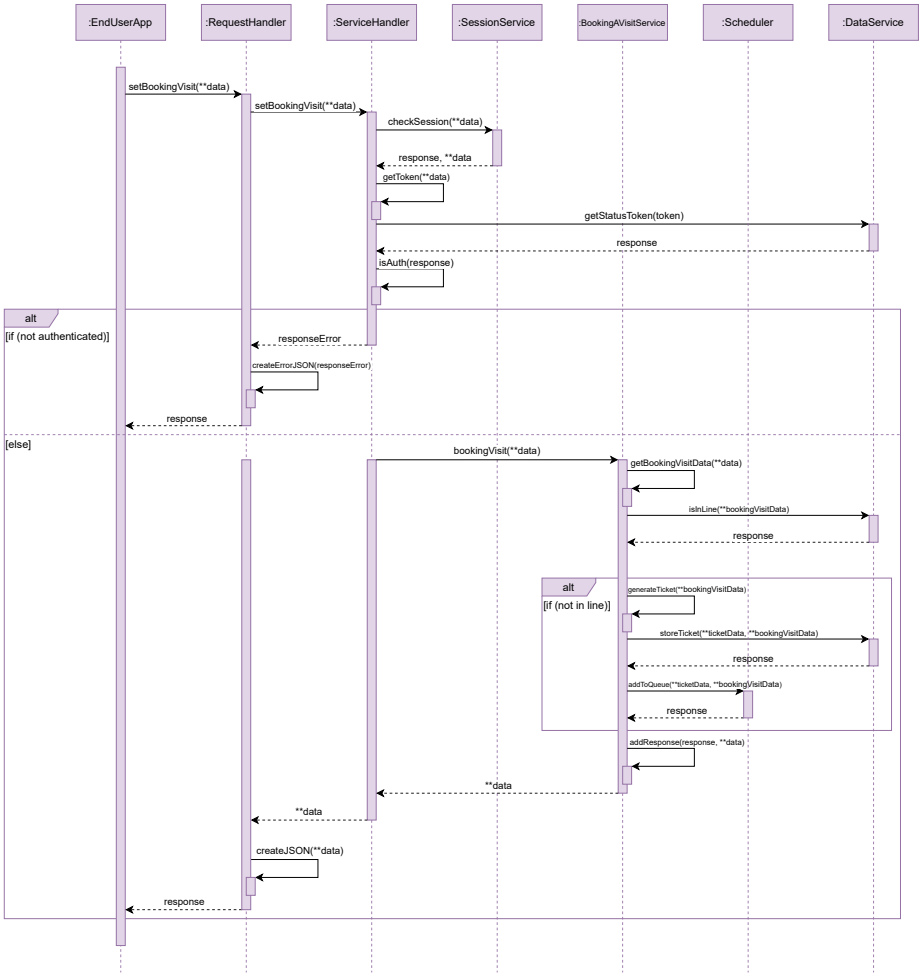


Figure 2.7: Booking a Visit sequence diagram.

2.4.6 Static Scheduler

Figure 2.8 illustrates the sequence of events that occur in the Scheduler after that the LiningUpService, or BookingAVisitService, has been activated. It is called *static* since these operations are performed only one time. The main purpose of this sequence is to show how the system allocates tickets in the virtual queue. It estimates the time in which customers will be called by collecting data from the history of purchases of customers (GroceryStoreService) and the needed time to arrive to the store (GoogleMapsService). Once an estimation is computed, the allocation in the queue is determined.

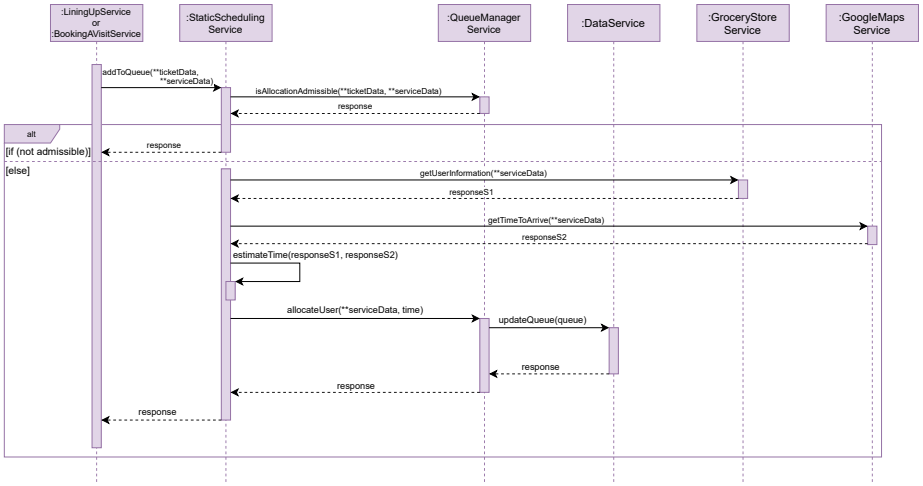


Figure 2.8: Static Scheduler sequence diagram.

2.4.7 Dynamic Scheduler

In contrast to the previous paragraph, in figure 2.9 there are activities that are performed mainly in background by the system to reschedule the virtual queue and to notify customers; for this reason it is called *dynamic*. The EndUserApp periodically sends information to the server about the location of the customers (in case of active tickets). These kind of requests are forwarded to the UserPositionService through the LiningUpService, or the BookingAVisitService. The UserPositionService is used to estimate the time to arrive of customers and to update the queue through the QueueManagerService. The QueueManagerService checks if it should notify users about timing and delays, moreover it updates the queue if tickets are expired.

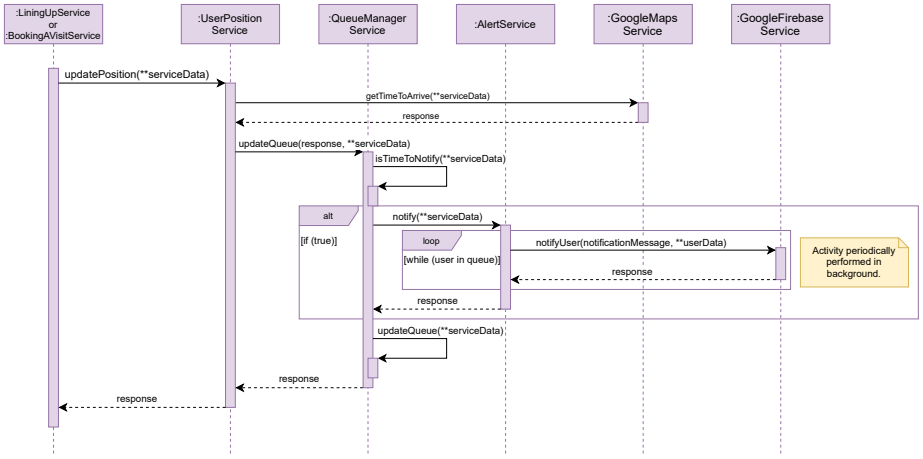


Figure 2.9: Dynamic Scheduler sequence diagram.

2.4.8 Control Queue

This service is used by the store manager to modify the parameters of the scheduler algorithm. Figure 2.10 follows the sequence of events to achieve this task. Since this operation can be activated only by store managers, the ControlQueueService verifies that the request is coming from a store manager account. In case of positive response, it will continue with a sequence of activities to update the parameters of the algorithm.

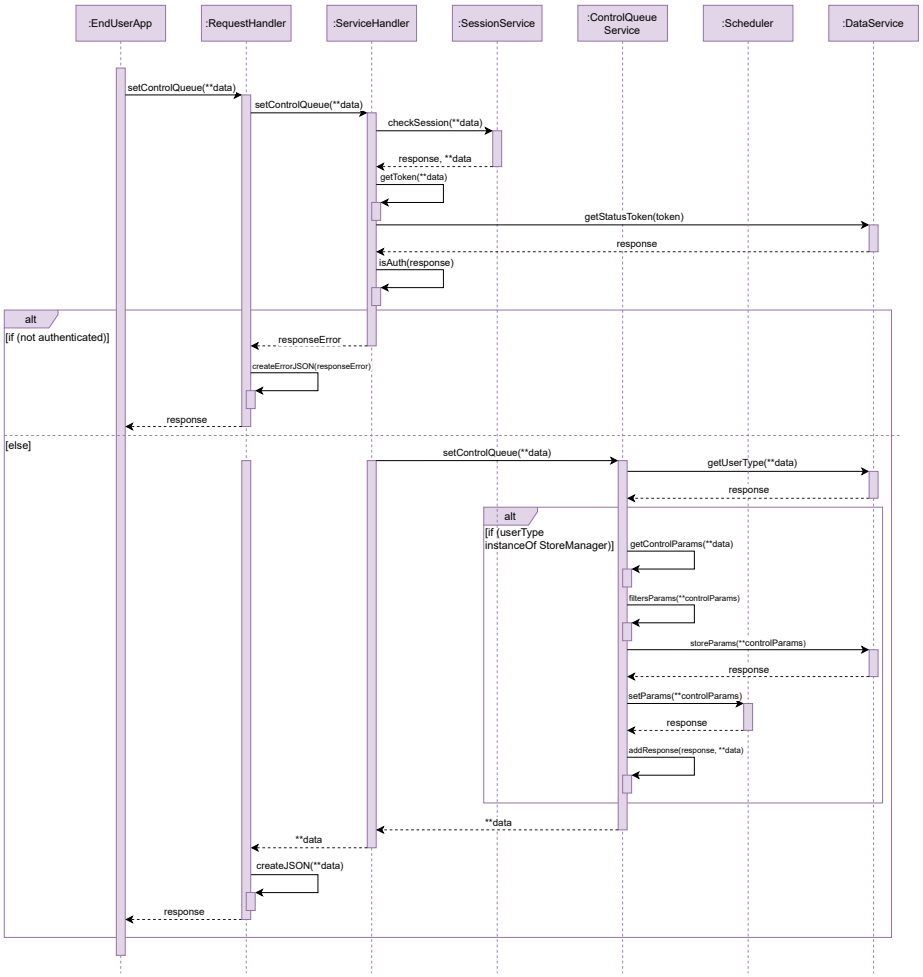


Figure 2.10: Control Queue sequence diagram.

2.4.9 Show Stats

Similarly to the previous paragraph, figure 2.11 shows how the server retrieves data for analytical purposes. As before, this is an activity allowed only by store managers.

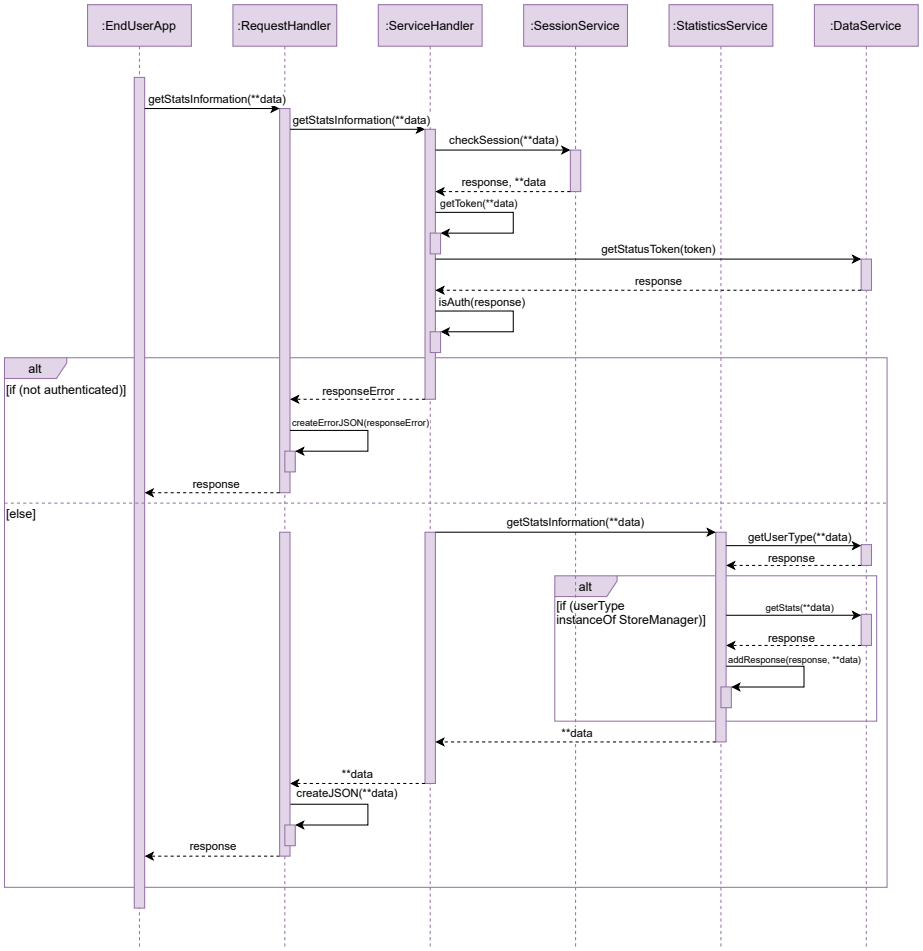


Figure 2.11: Show Stats sequence diagram.

2.4.10 QR Code Checking

When customers scan the QR codes, the device of the store manager, in background, communicates with the server to authorize customers to enter in the store. Therefore, figure 2.12 reports the actions performed by the system to check that the scanned QR code is the same that has been called by the system. This sequence diagram is strongly related to the corresponding one reported in the RASD.

After having scanned the QR code, the EndUserApp of the store manager sends the ticket number to the server that compares it with the expected number. If it is the same, the server reply to the EndUserApp that unlocks the turnstile and sends a confirmation to the server that save and update the status of the queue.

This is an activity that can be executed only by the store manager account, thus a verification must be performed as in figures 2.10 and 2.11 on the entity type. In this sequence, that verification wasn't be reported to avoid an overly crowded image.

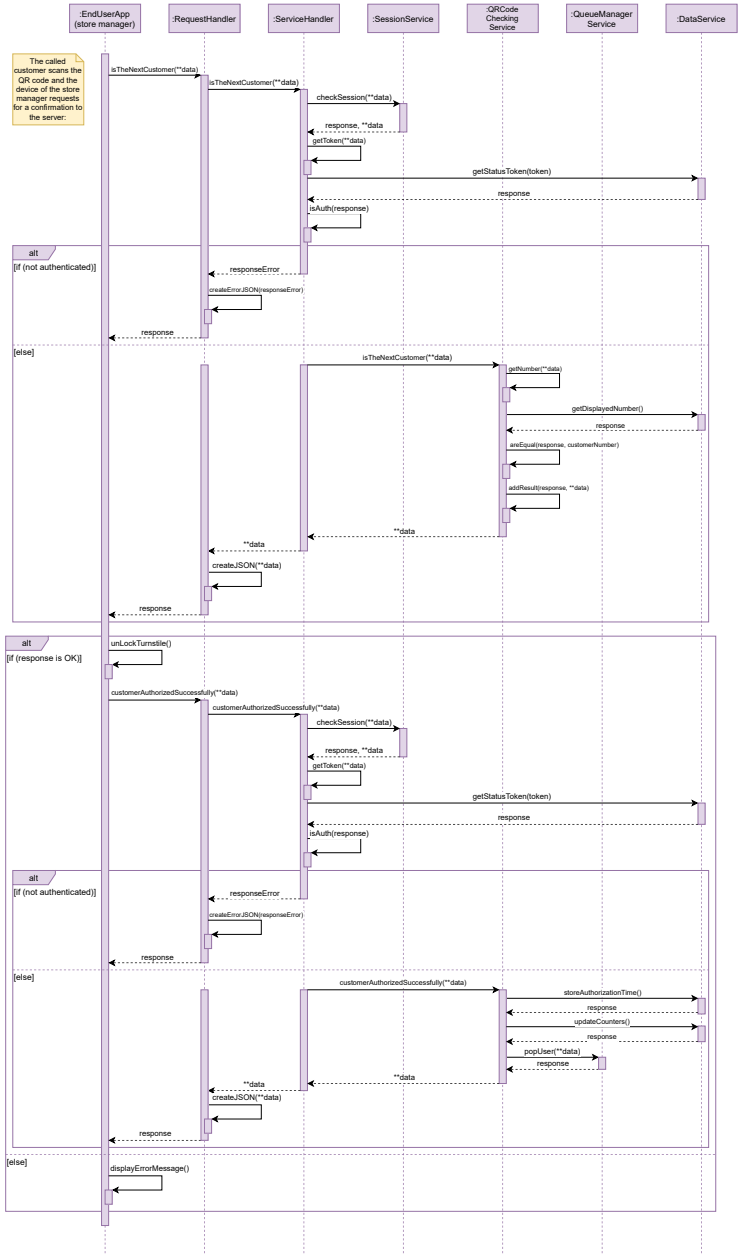


Figure 2.12: QR Code Checking sequence diagram.

2.5 Component Interfaces

In this section, it is proposed a diagram that shows the different interface and the method used. It also shows which of them may interact during different operations.
The methods here shown are not supposed to be final but to give a hint of their function and suggest a possible way to implement them.

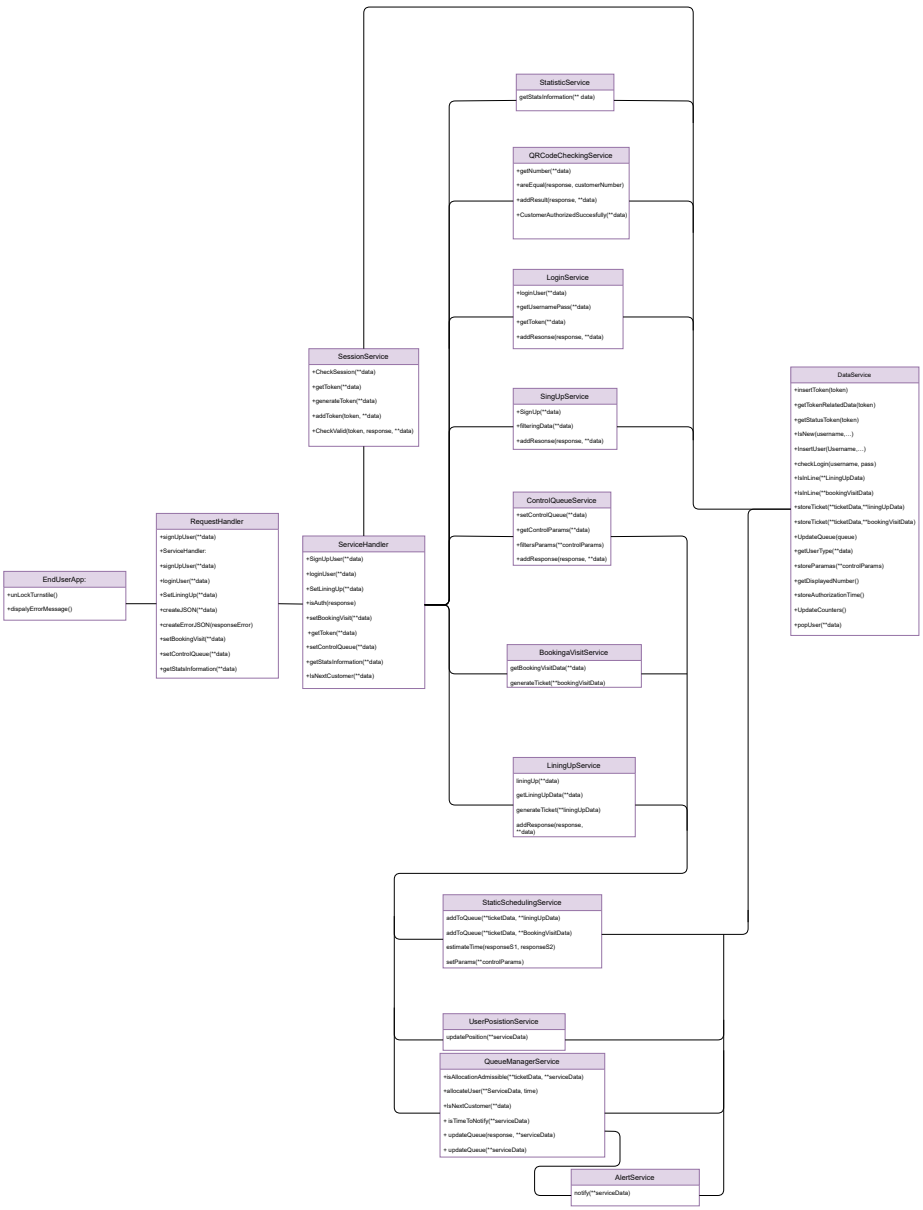


Figure 2.13: Component interface diagram.

2.6 Selected Architectural Styles and Patterns

2.6.1 Overall Architecture

The most suitable architecture for CLup is the three tiers architecture since it is a well known design pattern and a well established software architecture. It allows to distribute the service over multiple independent sub-systems.

In previous sections we already explained the functionalities offered by the

different tiers, now we are focusing the attention on the architectural point of view.

- **Presentation tier**: is the tier that allows users to interact with the system. It communicates with the business tier by using the Representational State Transfer (REST) architecture. As reported in figure 2.14, there are three types of entity that allow users to interact with the system through a Graphical User Interface (GUI). Only customers' devices communicate with the Google Firebase service to receive live notifications.

- **Business tier**: is the core of the service. It is protected by a firewall; it is composed by a data balancer and two physical servers to increase the availability and reliability of the system. Servers include the web and the application tiers. In case of necessity, the two sub-tiers can be divided over different servers. The tier communicates with external services: Google Maps, Google Firebase and the Grocery Store service.

- **Data tier**: is protected by another firewall and a load balancer is used to distribute queries between the two database servers. Only the business tier can communicate with the data tier.

The architectural choice has been influenced by the benefits that follows: scalability, performance and availability are the main. They are consequences of the distribution of the system over multiple independent platforms. Moreover, independence allows a parallel development of the system, reducing the needed time to deploy the service, playing an important role in a critic situation like the virus pandemic.

2.6.2 Design Patterns

The design patterns used by the system are:

- **Proxy**: is used as firewall to protect the business tier and the data tier. It provides a well known way to control accesses to the servers.

- **Observer** or **Publish/subscribe**: is the best choice in an environment in which there are one-to-many interactions. In CLup, the business tier acts as publisher when it notifies users about timing and delays. This pattern allows to update multiple objects (users' devices) after a state change in another one (server).

- **Producer/consumer**: is used in systems in which there are objects that offer a service and others that use the service. In this particular case, the users' devices are consumers and servers are producers. This design pattern allows to build a scalable and asynchronous system that allows to break the dependency between the producers and the consumers.

- **MVC**: is used in two architectural levels. First, in the smartphone application, as reported in figure 2.1; second, in the entire system as design pattern of the three tiers architecture since it is able to split the presentation tier from the business tier. We propose to use this pattern since it is the most common choice and because it has several benefits such as: fast development process, possibility to have multiple views and, having

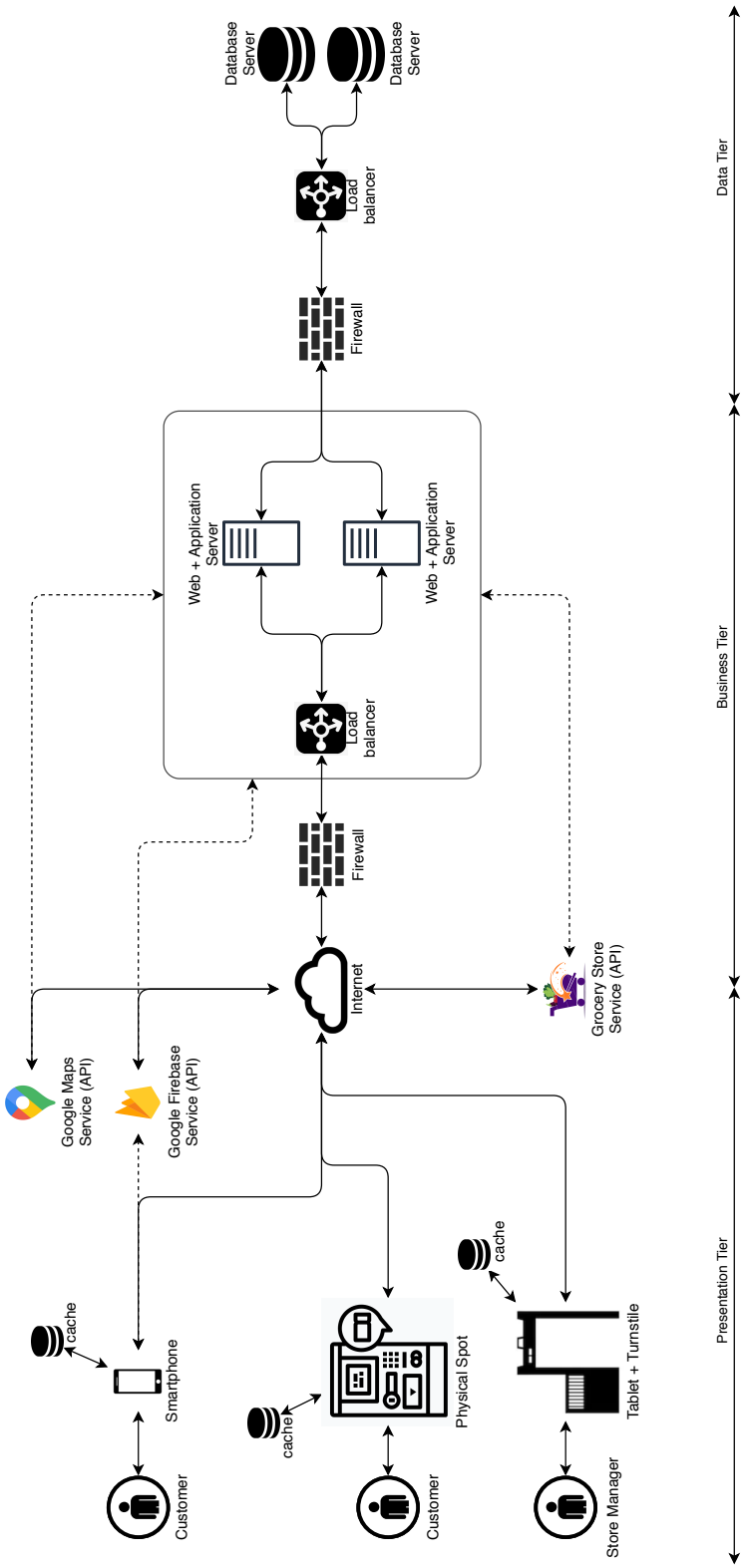


Figure 2.14: Architecture components.

different sub-modules, it allows to modify one component without changes in the others.

- **Adapter**: is used as design pattern to integrate external services.

2.7 Other Design Decisions

Few design decisions have been influenced by the integration of the external services. As shown in figure 2.14, Google Maps, Google Firebase and Grocery Store service have been integrated following the guidelines described in their documentations.

Chapter 3

User Interface Design

Different mock-ups of the application have been shown in the RASD. In the current document we are going to present the User Experience (UX) diagram 3.1 to help the reader in the understanding of the sequence of events and the relations between interfaces.

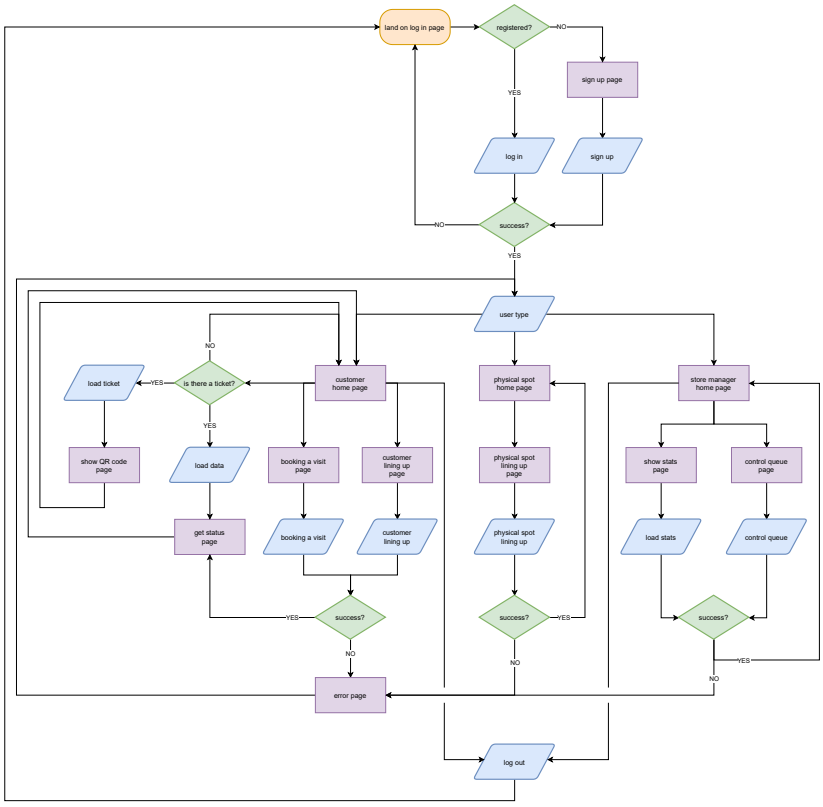


Figure 3.1: UX diagram. *Yellow* boxes represent **start**ing and **end**ing **points**. *Green* boxes are for **decisions**. *Violet* boxes are for **processes**. *Blue* boxes are for **inputs/outputs**.

Chapter 4

Requirements Traceability

- **[R1]**: The system has to schedule entrances to the store.
  Component:
  - Scheduler
- **[R2]**: The system has to compute the maximum capacity of the store w.r.t. the social distances imposed by the *"decreto del Presidente del Consiglio dei ministri"* (d.P.C.m) in force.
  Component:
  - DataService
  - StatisticService
- **[R3]**: The system has to monitor the customers residence time in the store. Component:
  - DataService
  - GroceryStoreService
- **[R4]**: The system has to allow authorized customers to enter in the store.
  Component:
  - DataService
  - QRCheckingService
  - LiningUpService
  - BookingService
  - EndUserApp
- **[R5]**: The system has to deny unauthorized customers to enter in the store. Component:
  - DataService
  - QRCheckingService
- **[R6]**: The system has to know when a customer enters in the store. Component:

  - DataService
  - QRCheckingService

- **[R7]**: The system has to know when a customer has left the store. Component:
  - DataService
  - GroceryStoreService

- **[R8]**: The system has to estimate the residence time, of a customer, in the store. Component:
  - DataService
  - StaticSchedulingService
  - QueueManagerService

- **[R9]**: The system has to infer the residence time of the customers based on past purchases. Component:
  - DataService
  - GroceryStoreService
  - StaticSchedulingService
  - QueueManagerService

- **[R10]**: The system has to estimate the time needed to arrive, to the store, from the position of the customer. Component:
  - UserPositionService
  - GoogleMapsService

- **[R11]**: The system has to track the global position of the customers. Component:
  - UserPositionService
  - GoogleMapsService

- **[R12]**: The system has to allow store managers to limit the number of QR codes released. Component:
  - Dataservice
  - ControlQueueService
  - EndUSerApp

- **[R13]**: The system has to allow the store manager to monitor the status of the queue. Component:
  - Dataservice
  - ControlQueueService
  - EndUSerApp

- **[R14]**: The system has to notify customers about the remaining time to be authorized to enter in the store. Component:
  - QueueManagerService
  - AlertService
  - GoogleFirebaseService
- **[R15]**: The system has to communicate which is the next served QR code number. Component:
  - EndUserApp
  - statisticService
- **[R16]**: The system has to allow customers to register to the application. Component:
  - DataService
  - SingUpService
  - EndUserApp
- **[R17]**: The system has to allow users to login to the application. Component:
  - DataService
  - LoginService
  - EndUserApp
- **[R18]**: The system has to release QR codes to the customers through the application. Component:
  - DataService
  - LiningUpService
  - BookingAVisitService
  - EndUserApp
- **[R19]**: The system has to alert customers if the queue is full. Component:
  - EndUserApp
  - LiningUpService
  - StaticSchedulingService
- **[R20]**: The system has to encode the ticket number in the QR code. Component:
  - LiningUpService
- **[R21]**: The system has to allow customers to watch the QR code from the application. Component:
  - EndUserApp
  - LiningUpService

- **[R22]**: The system has to allow customers to watch the ticket number encoded in the QR code. Component:
  - EndUserApp
  - LiningUpService
- **[R23]**: The system has to allow customers to watch the remaining time to be authorized to enter in the store. Component:
  - EndUserApp
  - LiningUpService
- **[R24]**: The system has to update the remaining time showed to the customers. Component:
  - QueueManager
  - LiningUpService
- **[R25]**: The system has to allow customers to delete a ticket. Component:
  - EndUserApp
  - LiningUpService
  - BookingAVisitService
- **[R26]**: The system has to notify customers about the validation status of the QR code. Component:
  - QueueManagerService
  - AlertService
  - GoogleFirebaseService
- **[R27]**: The system has to check if users have Internet connection active. Component:
  - EndUserApp
- **[R28]**: The system has to check if users have allowed the permissions requested by the application. Component:
  - EndUserApp
- **[R29]**: The system has to register store managers in the application. Component:
  - DataService
- **[R30]**: The system has to allow store managers to monitor the number of customers inside the store. Component:
  - EndUserApp
  - StatisticService
  - ControlQueueService

- **[R31]**: The system has to scan the QR codes of the customers. Component:
  - QRCodeCheckingService
- **[R32]**: The system has to allow store managers to modify the timing parameters of the scheduler. Component:
  - DataService
  - StaticsService
  - EndUserApp
- **[R33]**: The system has to allow unregistered customers to line up. Component:
  - EndUserApp
  - LiningUpService
- **[R34]**: The system has to release QR codes from a physical spot. Component:
  - DataService
  - LiningUpService
  - EndUserApp
- **[R35]**: The system has to print QR codes on a paper tickets. Component:
  - LiningUpService
- **[R36]**: The system has to alert when the paper and toner of the physical spot is going to finish. Component:
  - LiningUpService
- **[R37]**: The system has to allow customers to specify the date and time for a visit to the store. Component:
  - EndUserApp
- **[R38]**: The system has to allow customers to specify the category of grocery they want to buy. Component:
  - EndUserApp

Chapter 5

Implementation, Integration and Test Plan

5.1 Overview

In this section we will focus on steps to be conducted for the implementation, integration and test plan. Firstly, we ensure all the single units are created and tested on their own. Then we combine these unary modules into clusters that will perform a specific software sub-function. This will allow us to start the integrated testing of this clusters and test the interfaces between the units. This is crucial since the logic implemented in the modules differ from developer to developer. So, by testing them we ensure they work properly, and in case it allows us to detect possible errors in the interfaces. This process is repeated clustering the new formed units, until we reach the end. The software integration test is to be repeated again several times to ensures that no new previously undetected errors spawn. It serves also to ensure that the correct performance level is reached.

5.2 Integration testing approach:

The approach chosen to the integration testing is the bottom-up one. It starts testing from the lowest unit of the application and gradually moves towards integrating and testing the upper ones until all the units are integrated as one. Then the system as whole is tested. This process require the use of drivers that are disposable components made ad hoc to be placed on a higher level of the component tested to call their function. This approach was chosen because it allows us to localize in an easier way the possible errors in the lower levels, where is present the most complex component, the scheduler. The approach divides the component in five ordered blocks.

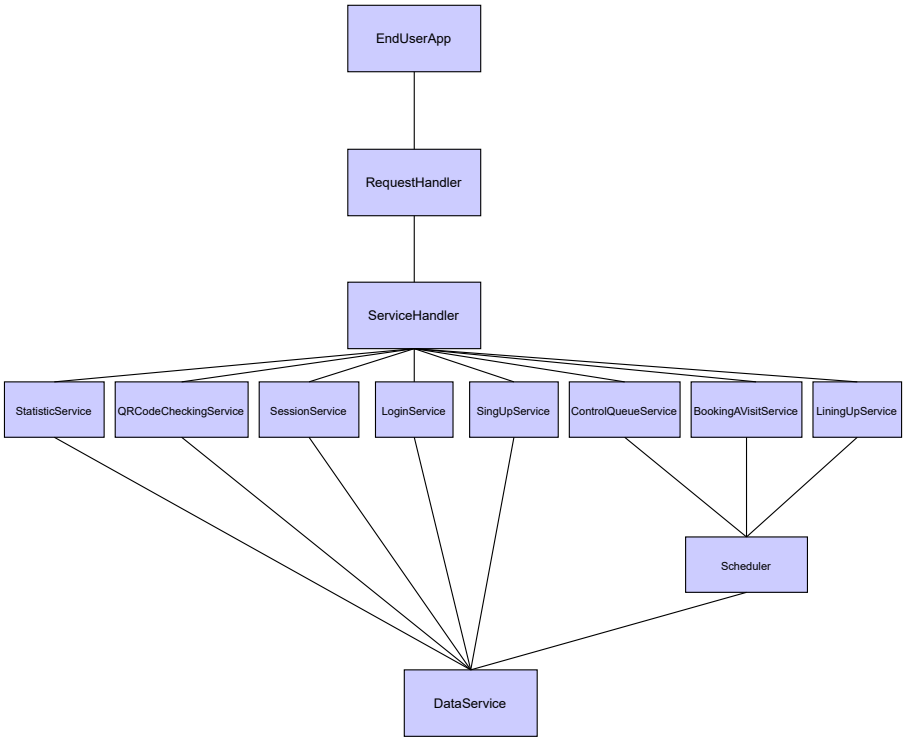
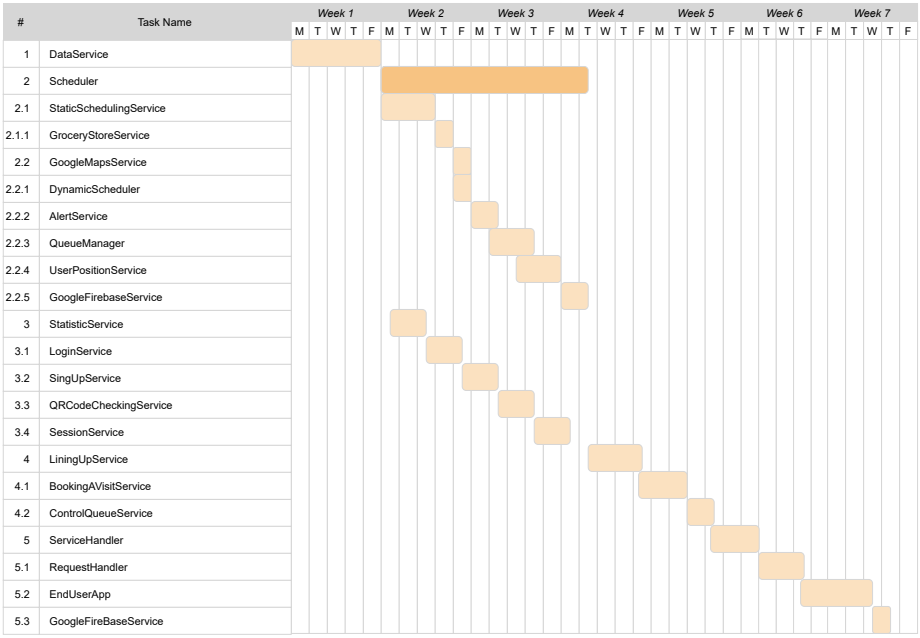


Figure 5.1: hierarchy of components.

As shown in the hierarchy of the components, the lowest component of the system and so the first one to be tested and implemented is the DataService. Every other component directly or not, rely on this one since they require to store and retrieve data to function. On the other hand, the data service does not rely on other components and so it can function in isolation.
The Scheduler should be implemented next and so as second block, since it is the

most complex system of the bottom level ones that use different component to function. Its function is to schedule both statically and dynamically for the app users, the order of their turn in the queue. It requires to consider both people that lined up and booked a visit, and possibly their data and position. It also ineteracts through the FireBaseService with the app-users to notify them when it is their turn. Inside the scheduler the first component to be implemented is the StaticSchedulingService. Since it is the one that handles first the request from the higher level and could function independently of the DynamicScheduler. It requires to build the drivers of the basic function of the upper modules and the integration with GoogleMapsServices and GroceryStoreService. After that the DynamicScheduler has to be implemented starting from the AlertService and a drive of the QueueManager to verify that the notification request are sent correctly to the GoogleFireBaseService. Then QueueManagerService and UserPositionService can be implemented and integrated in parallel.
The third block commence after the start of the implementation of the scheduler, the component to be integrated in parallel or immediately following are the StatisticService, QRCodeCheckingSerivice, LoginService, SingUpService and SessionService. Since all of them work independently to each other their order is also interchangeable, but the order of implementation chosen is the following: The first one is StatisticService, it offers function to the store manager and it is not strictly bond with the client interface. The second and third are LoginService and SingUpService, one serves to allow user to login and the other to allow user to sign in. The fourth one checks that the information scanned by the turnstile are correct and match the ones in the database. The last one, SessionService manages the session of the different users and so it more dependent on the higher levels. All of them require the use and so the develop of a driver of the ServiceHandler.
Subsequently the fourth block begin, after the scheduler is integrated, the component to implements are ControlQueueService, BookingAVisitService and LiningUpService. All of them need a driver for the ServiceHandler and must interact with the scheduler. The first one to change parameters and block the release of the new tickets, the second and third ones allow user to get respectively booking ticket and lining up ticket. The BookingAVisitService permits also to insert more information about the visit while the LiningUpService has also to check the user type to decide to monitor their position. The order of implementation of this block of component is the following: First the LiningUpService since it is the most important of the three then BookingAVisitService and lastly the ControlQueueService because it will be easier to check if it works as expected if the LiningUpService is already implemented.
And in the end the last block to be integrated is composed in order of the ServiceHandler, RequestHandler and the EndUserApp. The order of integration is clear since they are in different levels, it require to develop the RequestHandler and then EndUserApp drivers when proceeding on the implementation. Since they are the last ones to be implemented the testing should be done more meticulously to ensure that the almost complete system works as thought. Especially because these components are crucial to its functioning, they allow so send the request from the device, elaborate them and redirect on the right component.
The interfaces of the hardware component of the store and so turnstile, scanner and printer are included in the GroceryStoreSerivice. And so even though they are already implemented and tested, they should by further tested on site to

verify that the acquisition of data, the output data and the state transition are correct.



#	Task Name
1	DataService
2	Scheduler
2.1	StaticSchedulingService
2.1.1	GroceryStoreService
2.2	GoogleMapsService
2.2.1	DynamicScheduler
2.2.2	AlertService
2.2.3	QueueManager
2.2.4	UserPositionService
2.2.5	GoogleFirebaseService
3	StatisticService
3.1	LoginService
3.2	SingUpService
3.3	QRCodeCheckingService
3.4	SessionService
4	LiningUpService
4.1	BookingAVisitService
4.2	ControlQueueService
5	ServiceHandler
5.1	RequestHandler
5.2	EndUserApp
5.3	GoogleFireBaseService

Figure 5.2: Gantt chart.

5.3 Integration strategy :

As stated in the previous section,the approach chosen it the bottom-up one so the first component to be unit tested is the DataService. After it is implemented on top of that the scheduler and a driver to simulate the inputs.

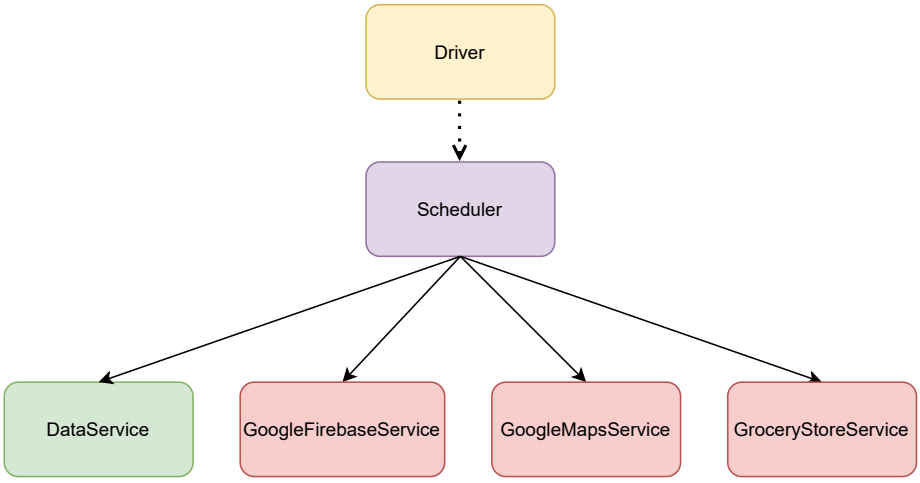


Figure 5.3: Integration step 1.

The scheduler being a complex component includes first the implementation of the StaticSchedulingService with the DataService and the two external services. Next the AlertService has to be implemented with GoogleFirebaseService

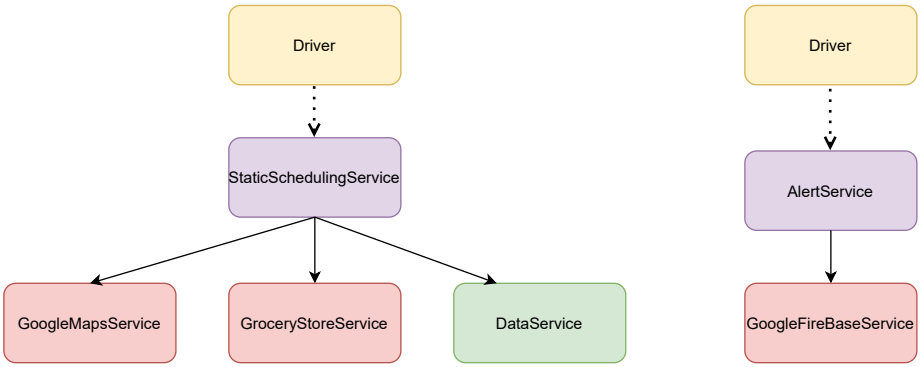


Figure 5.4: Integration step 2.

Then the QueueManagerService is implemented and integrated with the StaticSchedulingService and AlertService. Which causes the merge of the two small clusters.

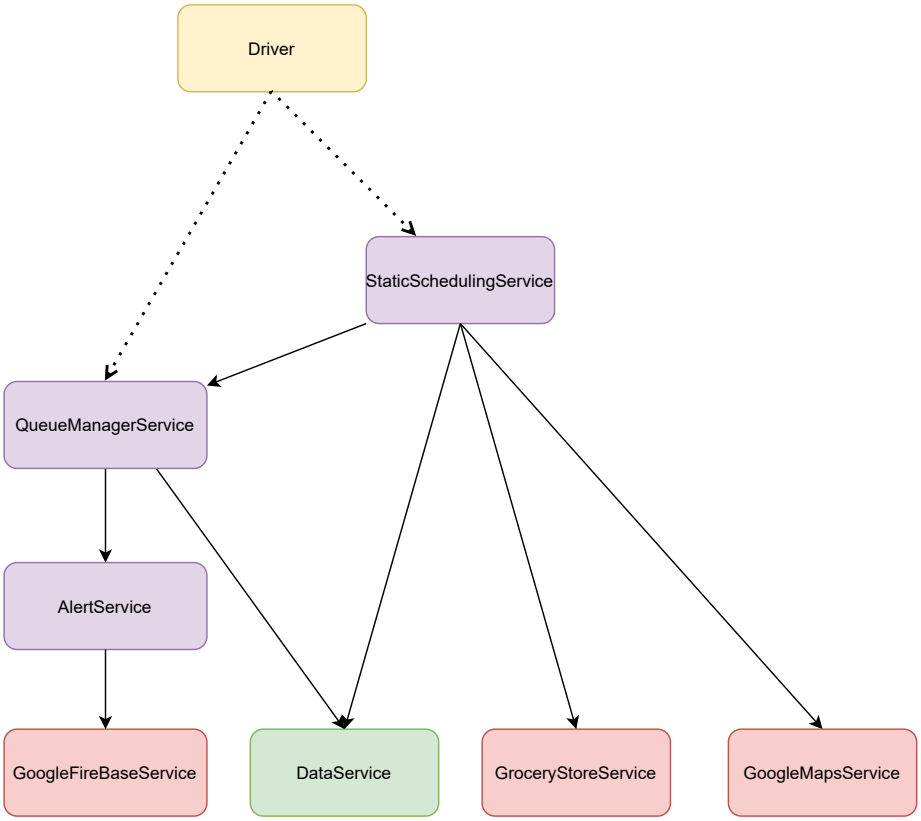


Figure 5.5: Integration step 3.

The UserPositionSerivice is implemented for last inside the scheduler and it completes it.

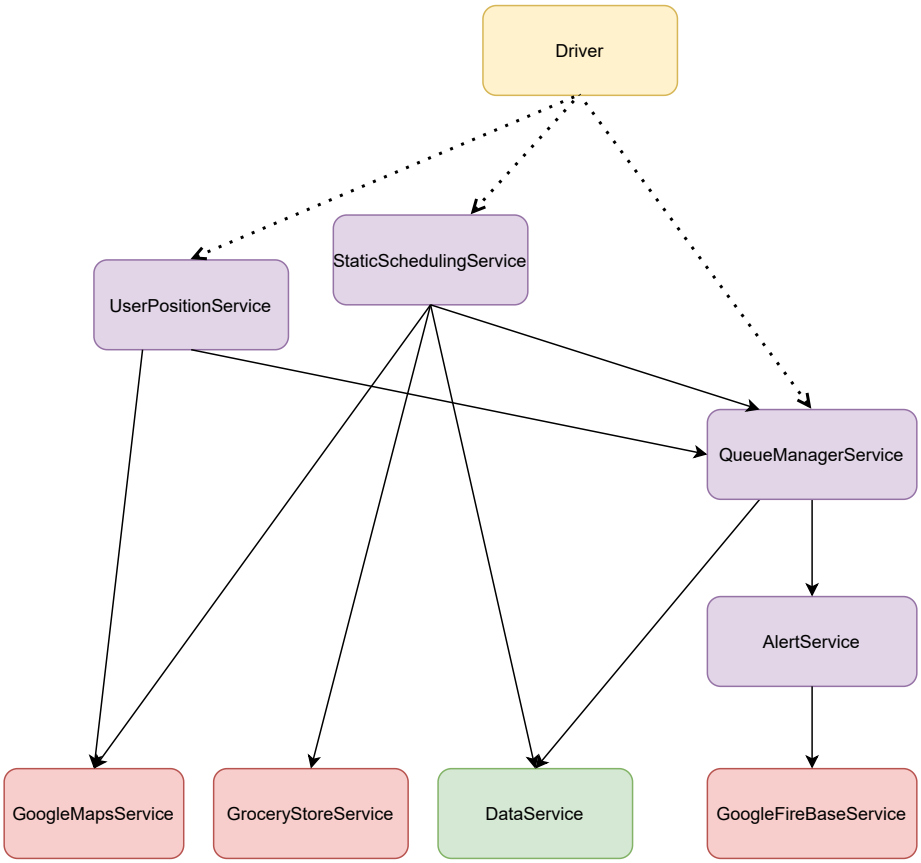


Figure 5.6: Integration step 4.

Simultaneously the components StatisticService, LoginService, SignUpService, QRCodeCheckingService and SessionService are implemented on the DataService in this order,with a driver of the ServiceHandler.

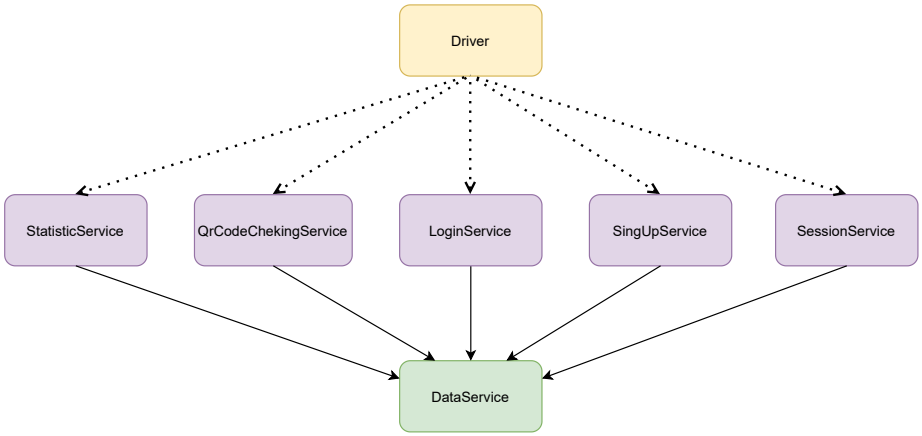


Figure 5.7: Integration step 5.

After the integration of the scheduler, the components that are implemented on top are in order LiningUpService, BookingAVisitService, ControlQueueService.

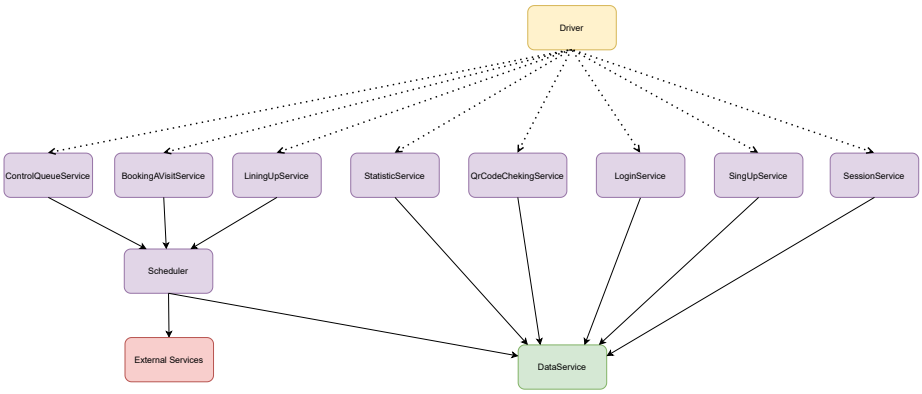


Figure 5.8: Integration step 6.

Lastly, the component implemented are the ServiceHandler with the driver on top, then the RequestHandler with the driver and finally the EndUserApp with the GoogleFirebaseService.

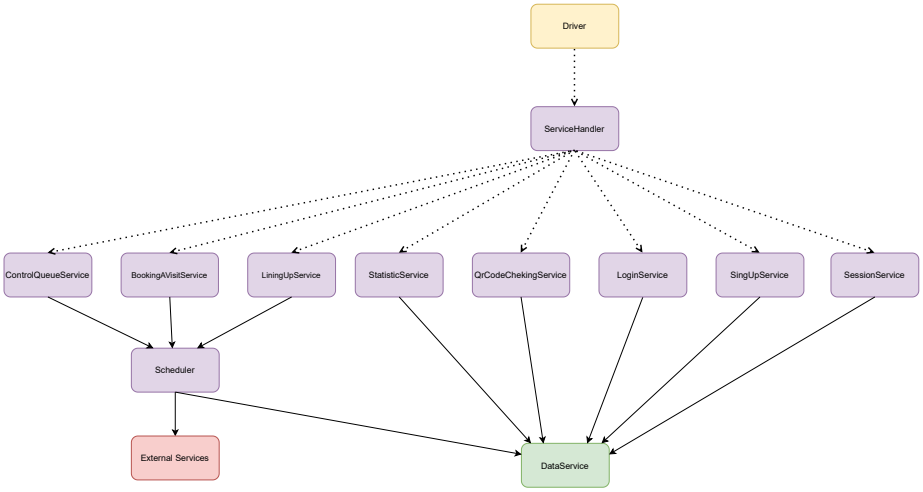


Figure 5.9: Integration step 7.

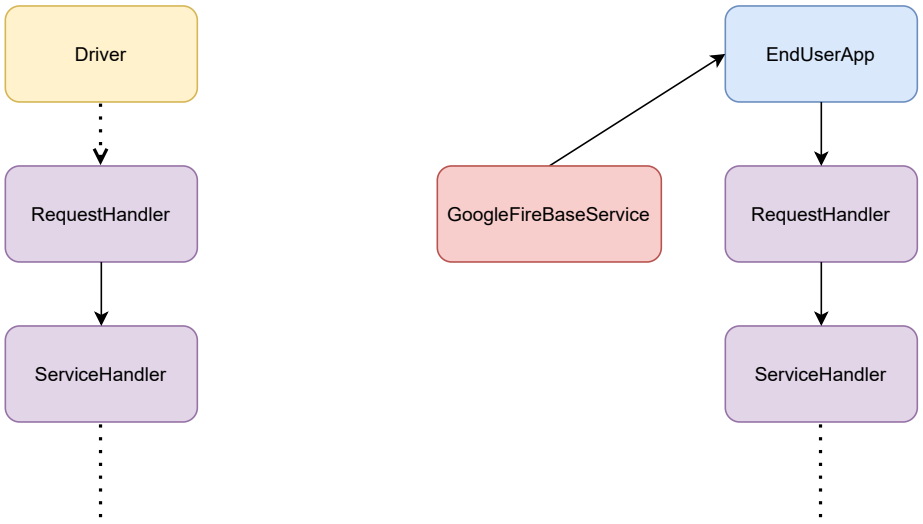


Figure 5.10: Integration step 8.

Chapter 6

Effort Spent

Topic	**Hours**
Preliminary Discussion	6
Introduction	4
Product Perspective	6
Product Functions	1
User Characteristics	2
Assumptions, Dependencies and Constraints	1
External Interface Requirements	2
Functional Requirements	5
Performance Requirements / Design Constraints / Software System Attributes	5
Alloy Code	12
Revision	4
Total:	48

Table 6.1: Effort spent by Jas Valencic.

Topic	**Hours**
Preliminary Discussion	6
Introduction	3
Product Perspective	3
Product Functions	3
User Characteristics	1
Assumptions, Dependencies and Constraints	2
External Interface Requirements	8
Functional Requirements	10
Performance Requirements / Design Constraints / Software System Attributes	3
Alloy Code	5
Revision	4
Total:	48

Table 6.2: Effort spent by Damiano Derin.

Chapter 7

References

- Specification document: "R & DD Assignment AY2020-2021.pdf".
- Slides of the lectures.
- RASD: "CLup - Customers Line-up Requirements Analysis and Specification Document.pdf"
- DDs of past students.
- Google Maps services: "https://cloud.google.com/maps-platform"
- Google Firebase service: "https://firebase.google.com/"
- Architecture styles: "https://docs.microsoft.com/en-us/azure/architecture/guide/architecturestyles/"

Bibliography

[1] Athena-fidus. https://www.asi.it/it/flash/telecomunicazioni-e-navigazione/athena-fidus, 2018.

[2] Dirk Gómez Depoorter, Omar Raissouni, Elisenda Temprado Garriga, and Oliver Lücke. The across testbed for the future aeronautical data communications. Technical report, IEEE, 2016.

[3] SESAR, Joint Undertaking. *Single Programming Document*, 2017-2019.