POLITECNICO
MILANO 1863

DD

Design Document

Authors: Gabriele DIGREGORIO
Enrico MASSARO
Vanessa TAMMA

Version: 2.0
Date: 07 February 2021
Professor: Elisabetta Di Nitto

CONTENTS

1 INTRODUCTION

1.1 Purpose

The purpose of the project CLup (Customer Line-up) is to develop a digital system of lining up that saves people from having to stand outside of stores for hours, avoids crowds inside the store, and, more in general, allows to regulate the influx of people in the stores.

The idea is to create a digital version of the traditional mechanism of lining up that is easy to use by everyone. In this way, the system would help to deal with the strict rules imposed by the government due to the global pandemic.

The system should give customers the possibility to line up from their home and approach to store only when their number is close to being called. This mechanism should avoid the situation in which the customers wait for their shift in the proximity of the store that is not an acceptable scenario in a lockdown situation.

1.2 Scope

The software should represent a digital alternative to the situation in which people retrieve a physical number that gives their position in the queue when they want to enter a store.

CLup should provide three main features:

- **Lining up**: allows customers to line up from their homes and avoids crowds outside the stores. It should include a notification system that alerts people when their number is close to being called. These alerts should consider the time customers need to get to the shop from the place they currently are and should be based on precise estimation of the waiting time. Moreover, CLup must provide effective fallback options for people who do not have access to the required technology.
- **Booking**: allows customers to book a visit to the supermarket. Since the time that it takes to visit a supermarket is not uniform, the system should give to user the possibility to specify an estimation of the duration of the visit. Alternatively, it might infer this information analysing the previous visits, if any.
- **Suggestion**: suggests different time slots for visiting the store (also on different days) to deal better with the restriction in the number of people inside the store. Alternatively, the system should propose other available supermarkets to the customers and alerts them in case a new time slot becomes available (e.g. after the deleting of a booking by another customer).

The customer that wants to use the service must be registered. Thanks to this, the system would be able to track the lining up, the booking, and the duration of the previous visits and use this information to manage better the influx of people and estimates with acceptable accuracy the waiting time.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

Time slot	Period or day that can be chosen for a booking by the customers.
Store data	Data about the store like the number of people allowed, opening and closing times, address, name, and photo.
Reservation	A word that might indicate either a booking or a lining up in a specific store.
Active reservation	Lining up or booking that is not yet expired. It means that the reservation has been taken but customers still have to wait for their shift.
Store manager	Manager, cashier, or generic employee of a store.
Available shop	A shop that is registered into the system.

1.3.2 Acronyms

CLup	Customer Line-up
DBMS	DataBase Management System
MVC	Model View Controller
RASD	Requirements Analysis and Specification Document
RDBMS	Relational DBMS
SSL	Secure Sockets Layer
TLS	Transport Layer Security
UI	User interface

1.3.3 Abbreviations

G.n	Goal n-th
R.n	Requirement n-th
C.n	Component n-th

1.4 Revision history

DATE	DESCRIPTION
29/12/2020	First version. Mock-ups and their descriptions.
02/01/2021	Architectural Design: Overview and Component View
03/01/2021	Requirement Traceability
04/01/2021	Algorithms: Pseudocodes and Flowcharts
05/01/2021	Sequence Diagrams integration and comments
06/01/2021	Architectural Styles and Patterns. General improvements.
09/01/2021	Deployment View and general improvements.
10/01/2020	Class Diagram integration. Revision and corrections.
13/01/2020	Formatting improvements. Updating of the sequence diagrams. Changing the version number to consistency with RASD.
30/01/2021	Updates on the integration policy to make the document coherent with the ITD.
07/02/2021	Small changes in the Runtime View and requirements (coherence with the other documents).

1.5 Reference Documents

- *Requirement Engineering and Design Project: goal, schedule, and rules*
- *I&T assignment goal, schedule, and rules*
- Slides of the course *Software Engineering 2*

1.6 Document Structure

The document is composed of the seven following chapters:

- **Chapter 1**: provides an introduction to the purposes and the whole scenario of the software, as already specified in the RASD document. First, it includes the general description of the system, then there is a sufficiently detailed specification of the main features that the system should provide. Lastly, it includes the list of abbreviations, acronyms, and definitions used in the document, the revision history, and the reference documents.
- **Chapter 2**: is the main chapter of this document and includes several sections and diagrams. Here, is provided a precise description of the components of the system and their

interactions. Furthermore, it includes also the Component View, the Deployment View, and the Runtime View. This section is concluded with the Component Interfaces and the overview of the chosen patterns and architectural styles.

- **Chapter 3**: includes some extensions and more details about the *User Interface* (UI) with respect to what was specified in the RASD document. In this section, several new mock-ups and their descriptions are included.
- **Chapter 4**: shows the traceability between the requirements described in the RASD document and the components explained in this document.
- **Chapter 5**: explains the plan for the implementation, integration, and test phases. It includes also the necessary references to the patterns and strategies adopted for these purposes.
- **Chapter 6**: shows the amount of time that each member has spent to produce the document.
- **Chapter 7**: specifies the reference documents and online resources used during the production of this document.

2 ARCHITECTURAL DESIGN

2.1 Overview

From a high-level point of view, a three-tier architecture appears as a good choice for the system. This is a well-known architecture organized in three logical and physical tiers:

- **Presentation tier**: it allows the interaction with the users providing the UI and the needed communication layers. It is able both to show and acquire information about the users that use the system.
- **Application tier**: it is able to modify, add, or delete the data in the data tier. Using some defined business rules, it also processes the information from the presentation tier. It controls the application's functionalities and is also known as *logic tier* or *middle tier*.
- **Data tier**: here the data are stored and managed including the data persistence mechanism. It is also known as *database tier* or *data access tier*.

All communications between presentation and data tiers are possible only using the application tier.

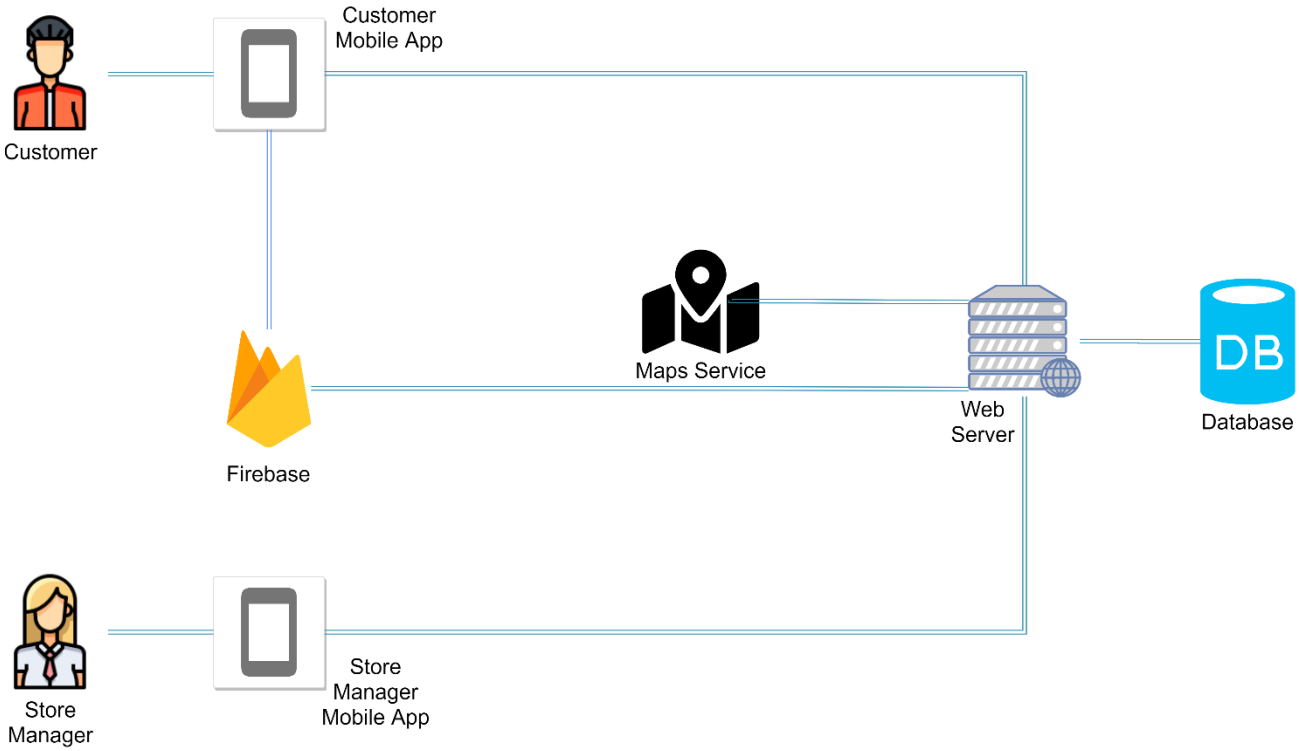


Figure 1: System Architecture

This architecture offers several advantages like the logical and physical separation of functionalities. Moreover, each tier is executed on a dedicated operating system or platform on a dedicated virtual and hardware component. Other strengths are:

- reduction of the developing time thanks to the independence of each tier with respect to the others;
- high scalability: it is possible to scale each tier independently to the others;

- high reliability: it is unlikely that an eventual failure on a specific tier has an impact on the others;
- hight security: the separation of the tiers makes less likely malicious attacks like SQL injections. In particular, the presentation tier and data tier cannot communicate directly.

2.2 Component view

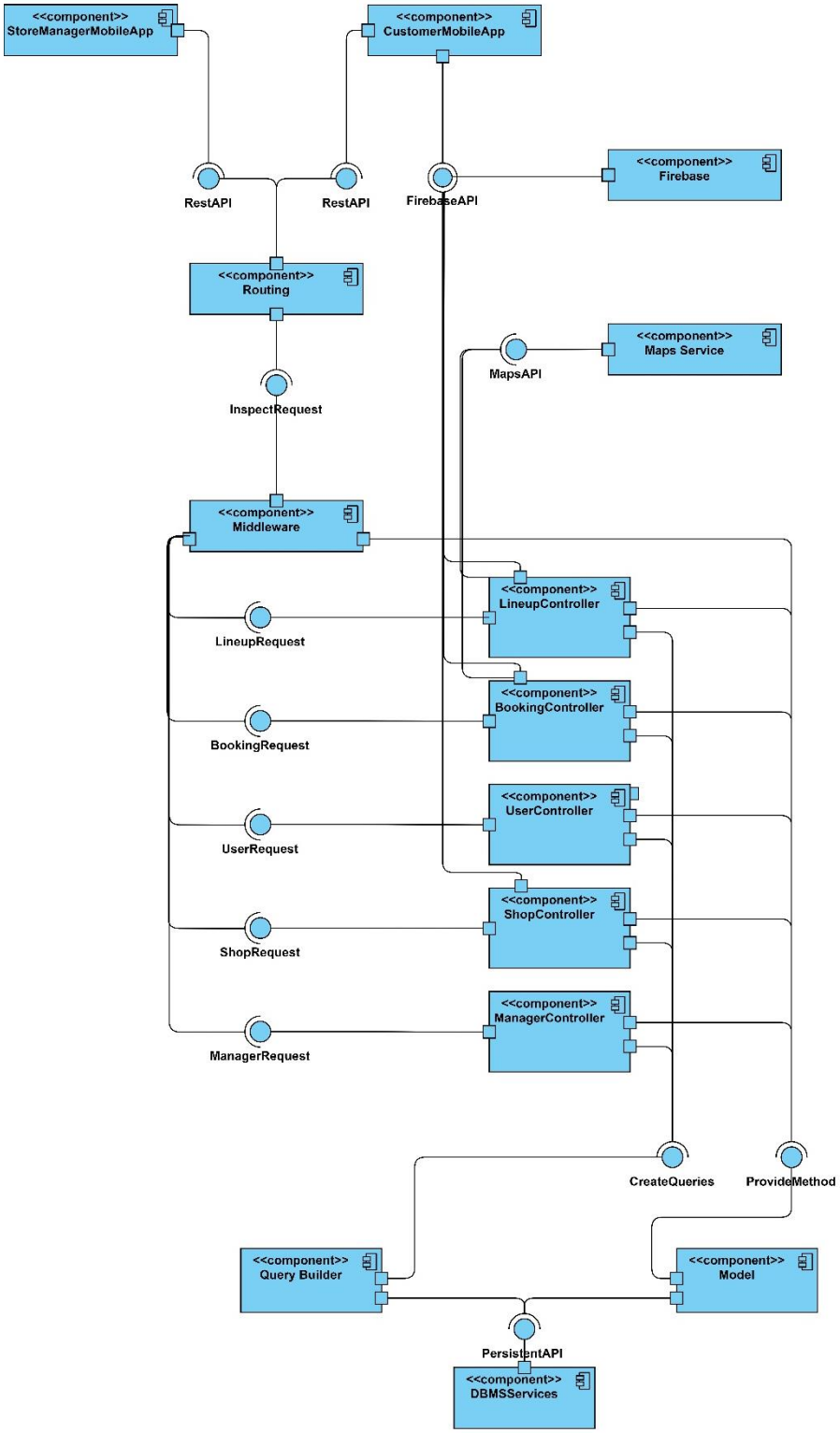


Figure 2: Component Diagram

The Component Diagram represents the internal structure of the system describing the main components and links between them. It shows also how some components are wired together to create larger components. The structure is a reinterpretation of the architecture of the PHP framework Laravel that will be used to develop the system.

The depicted components are:

- **CustomerMobileApp**: mobile application used by the customer to access the system and its functionalities.
- **StoreManagerMobileApp**: mobile application used by the storer manager to access the system and its functionalities.
- **Routing**: this component loads the route files dispatching an HTTP request to a route, a controller, or executing the necessary middlewares.
- **Middleware**: filters and examines the requests according to the specifications of the component. Particular attention is paid to the authentication middleware that checks if the users that use the application are correctly authenticated. If they are so, their request can proceed; otherwise, the request is redirected.
- **UserController**: defines customer requests. For instance, it provides the method for the customer login that checks the inserted email and password and, if they are correct, creates a unique token. This token authorizes the user to request further operations.
- **BookingController**: defines requests that create, update, and delete a booking. Manages also the shifts to enter the store.
- **LineupController**: defines requests that create, update, and delete a lining up. Manages also the shifts to enter the store.
- **ShopController**: defines shop requests. It permits to query all available shops.
- **ManagerController**: defines store manager requests. It allows the store manager to log in to the system.
- **Query Builder**: this component allows to create and run queries communicating with the DBMSService. As specified in the Laravel documentation, it includes a protection mechanism from SQL injection attacks that guarantees a high level of security.
- **Model**: provides methods to access and manage data. It is used by the controller.
- **DBMSService**: manages the creation, manipulation, and querying of the database.

Furthermore, the system uses some external APIs. In particular:

- **Firebase**: it is used to manage push notifications and guarantees a good integration with Android. Moreover, they are easy to implement in the application server.
- **Maps Service**: helps to estimate the needed time to arrive at the store given the current position of the customer. This information is used to provide notifications that alert the customer when her/his shift is near to being called.

2.3 Deployment view

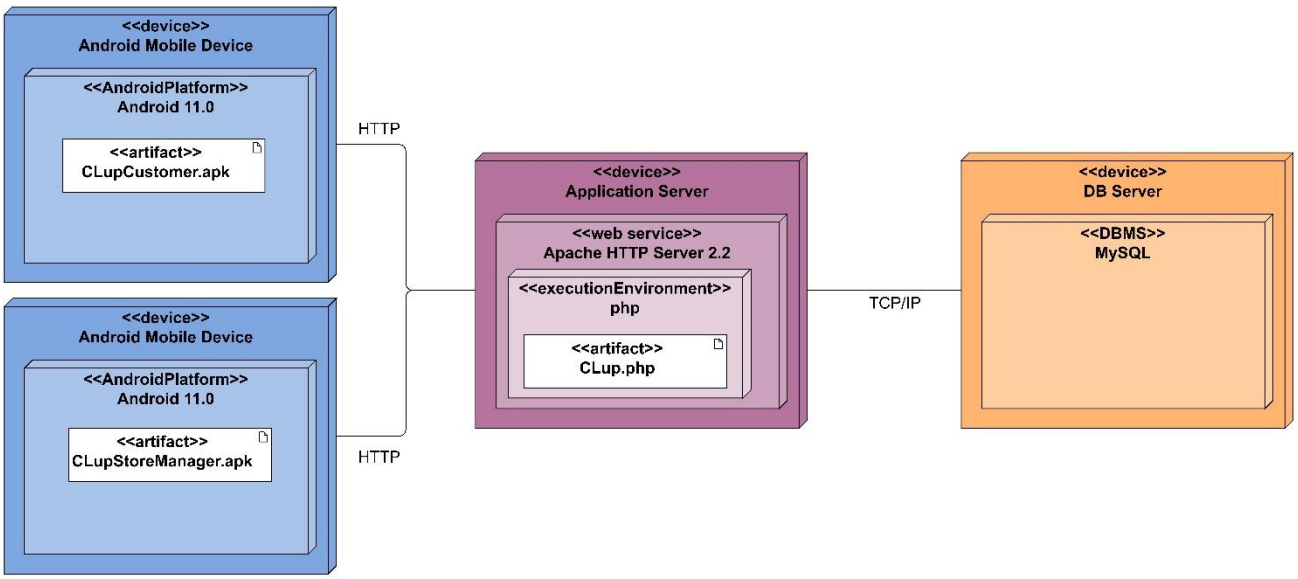


Figure 3: Deployment Diagram

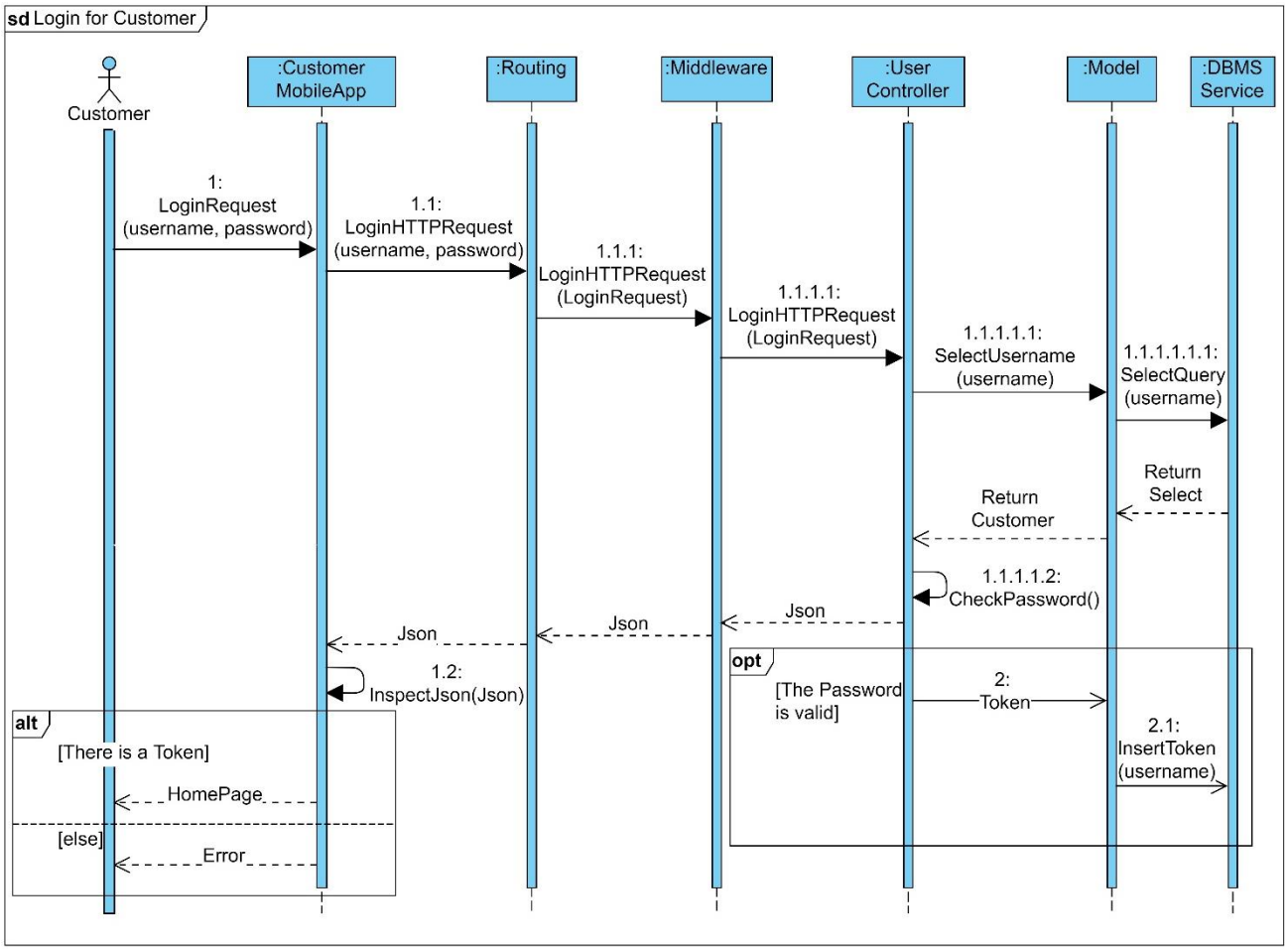
The purpose of this section is to explain the distribution of the components and, possibly, identify which of them might represent bottlenecks. In more detail, the deployment diagram illustrates the topology of the system from the hardware point of view.

In the figure above and from left to right are depicted, respectively, the components of the presentation tier (the blue ones), the component of the application tier (the purple one), and the component of the data tier (the orange one).

- **Smartphone**: suitable mobile device with an internet connection. Initially, the compatibility will be limited to the Android operating system.
- **Application Server**: it contains most of the application logic of the system.
- **DBMS**: MySQL. It is a Relational Database Management System (RDBMS) released with a system of double license (GNU GPL and commercial license). Its strengths include high availability and performance, and good scalability. It allows dealing with the common CRUD (Create Read Update Delete) operations.

2.4 Runtime view

1. Login for Customer



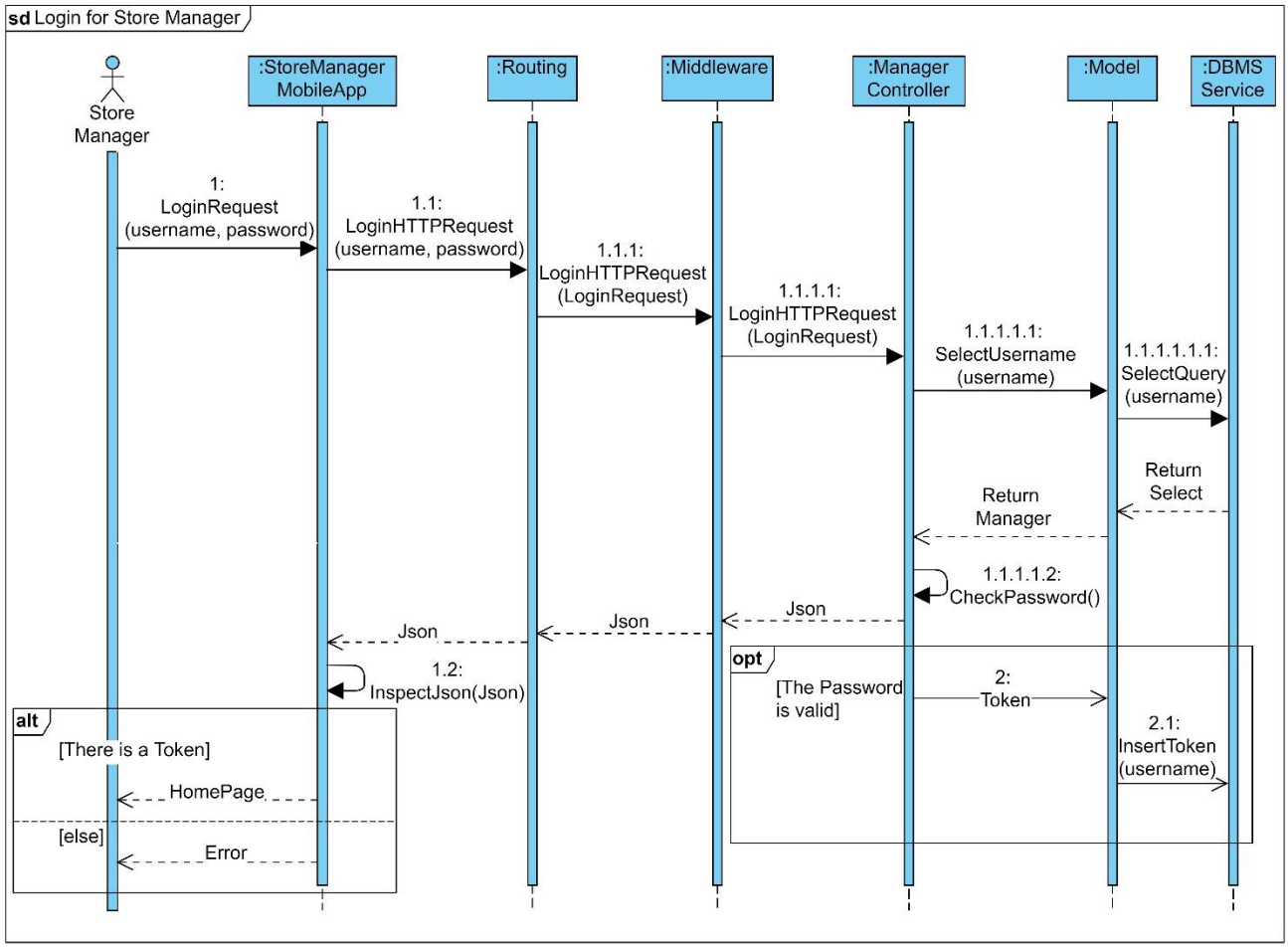
The above sequence diagram shows the login process for a generic customer.

First, the customers insert their credentials using the CustomerMobileApp and then they require the authentication. The application sends an HTTP request that is propagated through the Routing and the Middleware until it reaches the UserController. At this point, the controller requests the customer account data utilizing a query and the functionalities exposed by the Model. These data, returned at the UserController, are used to check the credentials inserted by the customers at the beginning.

If the credentials are correct, a token is generated, it is serialized in a JSON file, and it is returned at the CustomerMobileApp. Here the file is checked and, if there are no errors, the HomePage is displayed to the customers. At the same time, the token is propagated from the UserController to the DBMSService in order to be stored permanently in the database.

Otherwise, if the check generates an error, the JSON file contains the error itself. In this case, the CustomerMobileApp shows an error message to the customers.

2. Login for Store Manager



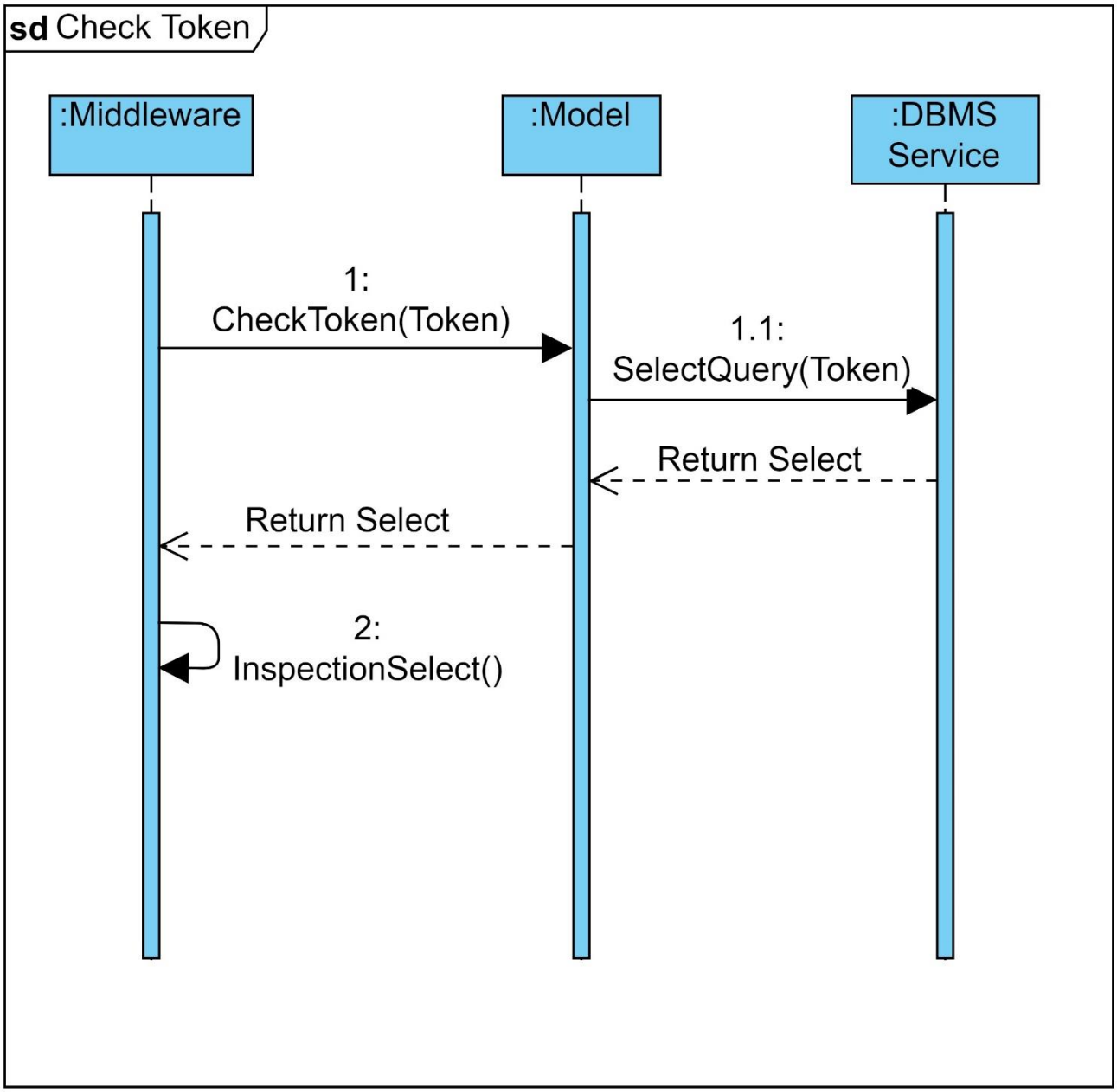
The above sequence diagram shows the login process for the store manager. It is quite similar to the previous one with some differences in the components involved.

First, the store managers insert their credentials using the StoreManagerMobileApp requiring the authentication. Then, the application sends an HTTP request that is propagated through the Routing and the Middleware until it reaches the ManagerController. At this point, the controller requests the store manager account data utilizing a query and the functionalities exposed by the Model. These data, returned at the ManagerController, are used to check the credentials inserted by the store managers at the beginning.

If the credentials are correct, a token is generated, it is serialized in a JSON file, and it is returned at the StoreManagerMobileApp. Here the file is checked and, if there are no errors, the HomePage is displayed to the store managers. At the same time, the token is propagated from the ManagerController to the DBMSService in order to be stored permanently in the database.

Otherwise, if the check generates an error, the JSON file contains the error itself. In this case, the StoreManagerMobileApp shows an error message to the store managers.

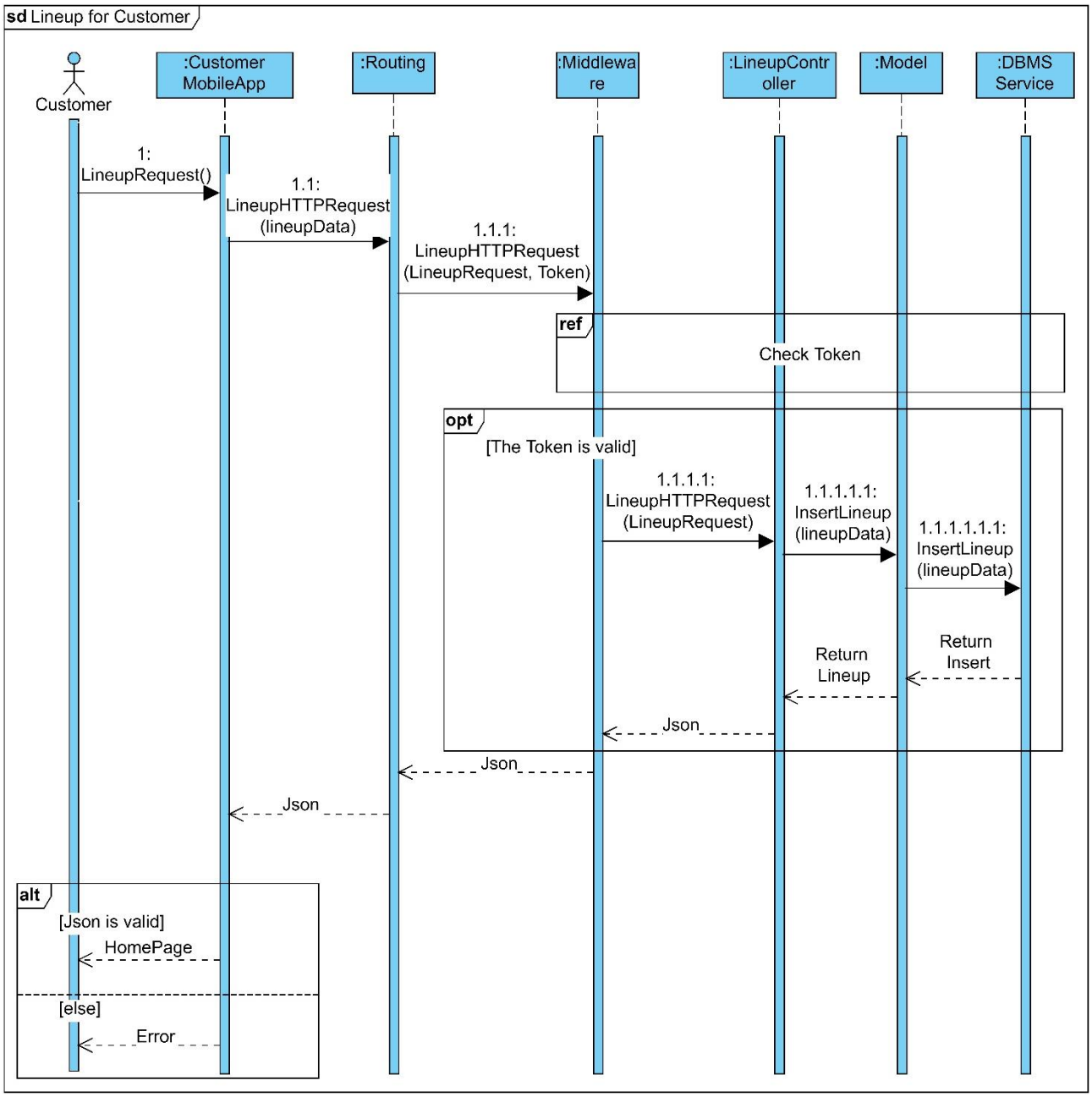
3. Check Token



This sequence will be used in the next diagrams. In particular, the current diagram shows the process of checking the token to authorize the actions required by the users. If the token is not valid, the requests are not forwarded by the Middleware.

First, the Middleware calls a method of the Model passing as a parameter the token itself. This token is then used in a query that is executed by the DBMSService. Following the same path, the result arrives at the Middleware where it is inspected.

4. Lineup for Customer



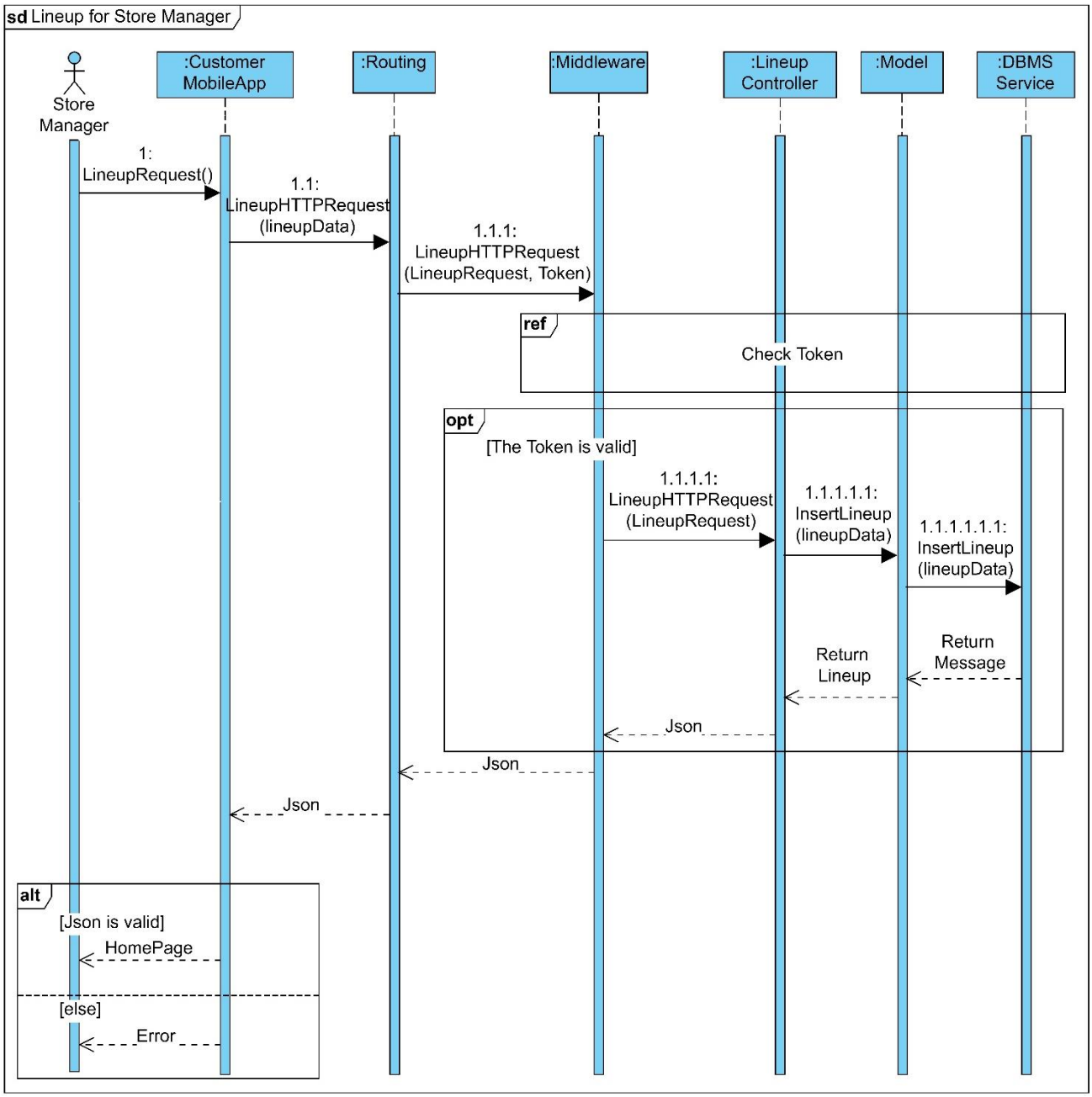
The diagram in this section displays the Lineup sequence for customers.

As usual, the customer requires the action to the CustomerMobileApp. Then, the application sends an HTTP request that reaches the Middleware after has been passed through the Routing component. Here, the token is checked with the CheckToken sequence already described.

If the check is positive, then the request of lining up is propagated until the LineupController. The LineupController calls a method of the Model and then a query is executed in the DBMSService. Following the same path, the information about the Lineup is returned to the LineupController and serialized in a JSON that reaches, in the end, the CustomerMobileApp. Lastly, the application shows the Home Page with the updated list of reservations.

If the check is negative, the Middleware does not allow the forwarding of the request. So, an error message is serialized in the JSON file and it is sent to the CustomerMobileApp. In this case, the application displays an error page instead of the Home Page.

5. Lineup for Store Manager



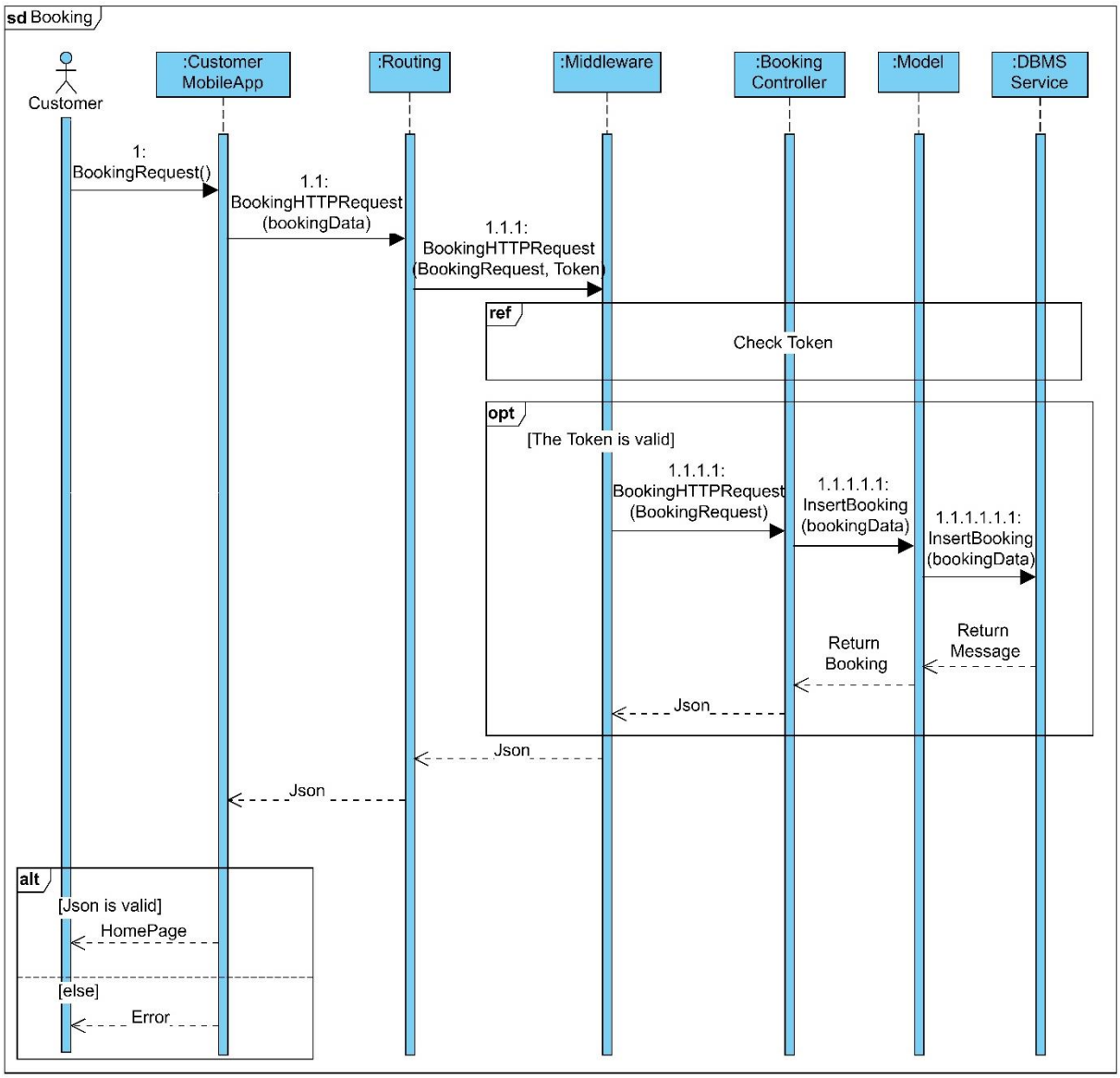
The diagram above depicts the Lineup sequence for store managers. It is quite similar to the one for customers with some differences.

As usual, the customer requires the action to the StoreManagerMobileApp. Then, the application sends an HTTP request that reaches the Middleware after has been passed through the Routing component. Here, the token is checked with the CheckToken sequence already described.

If the check is positive, then the request of lining up is propagated until the LineupController. The LineupController calls a method of the Model and then a query is executed in the DBMSService. Following the same path, the information about the Lineup is returned to the LineupController and serialized in a JSON that reaches, in the end, the StoreManagerMobileApp. Lastly, the application shows the Home Page.

If the check is negative, the Middleware does not allow the forwarding of the request. So, an error message is serialized in the JSON file and it is sent to the StoreManagerMobileApp. In this case, the application displays an error page instead of the Home Page with the updated list of reservations.

6. Booking



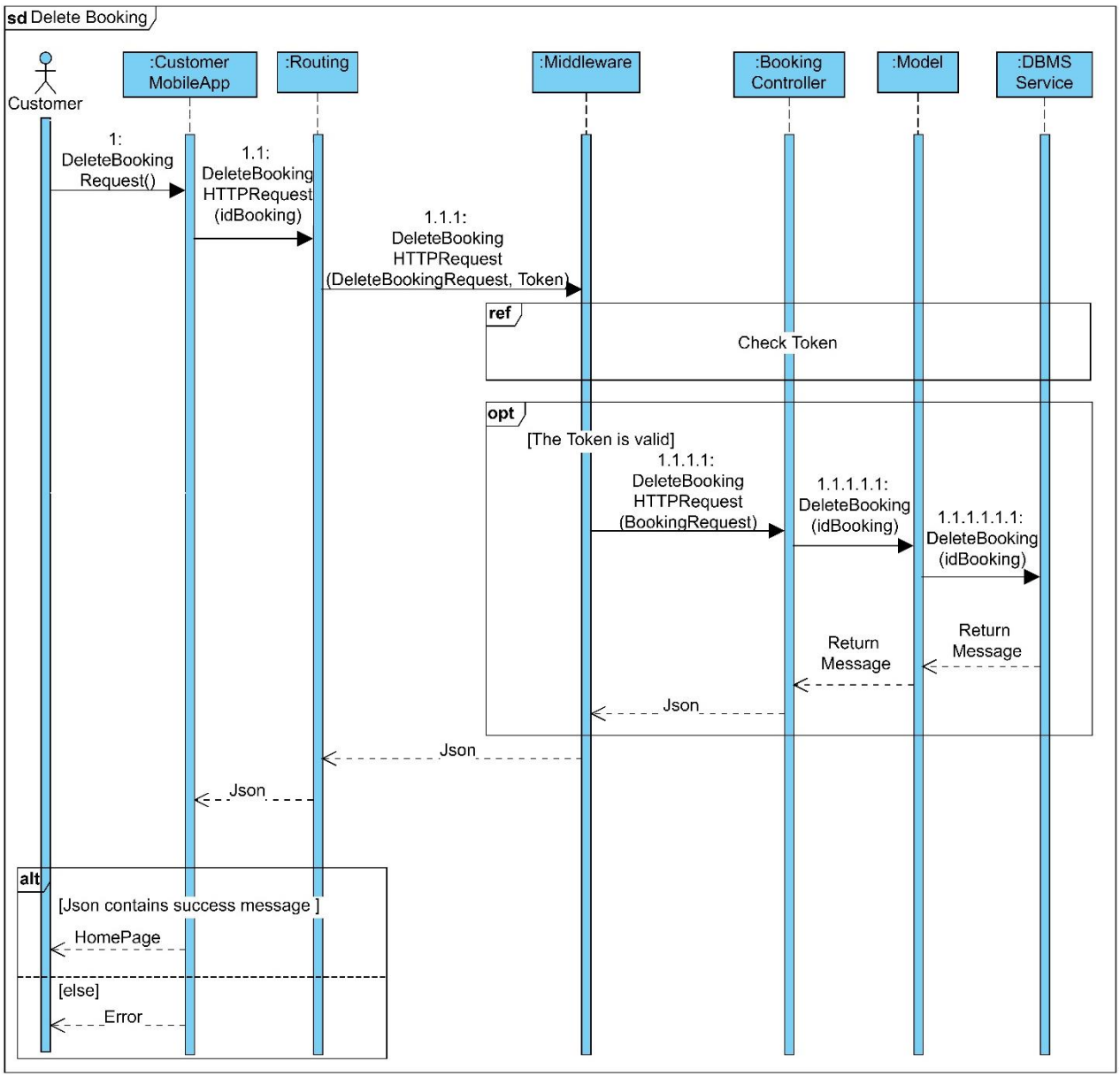
The diagram above shows the Booking sequence that is allowed only for customers.

The interested customer uses the CustomerMobileApp to get a new booking. The CustomerMobileApp sends an HTTP Request that passes through the Routing and arrives at the Middleware. Here, the token is checked with the CheckToken sequence already described.

If the token is valid, then the booking request is propagated until the BookingController. The BookingController calls a method of the Model and then a query is executed in the DBMSService. The ack and the information about the booking are returned to the BookingController and then it is serialized in a JSON file. This file then reaches the CustomerMobileApp where it is checked. The CustomerMobileApp shows, in the end, the HomePage with the updated list of reservations.

If the token is not valid, the Middleware does not forward the request. Next, an error message is serialized in the JSON file, it is sent to the CustomerMobileApp, and the application displays an error page instead of the Home Page.

7. Delete Booking



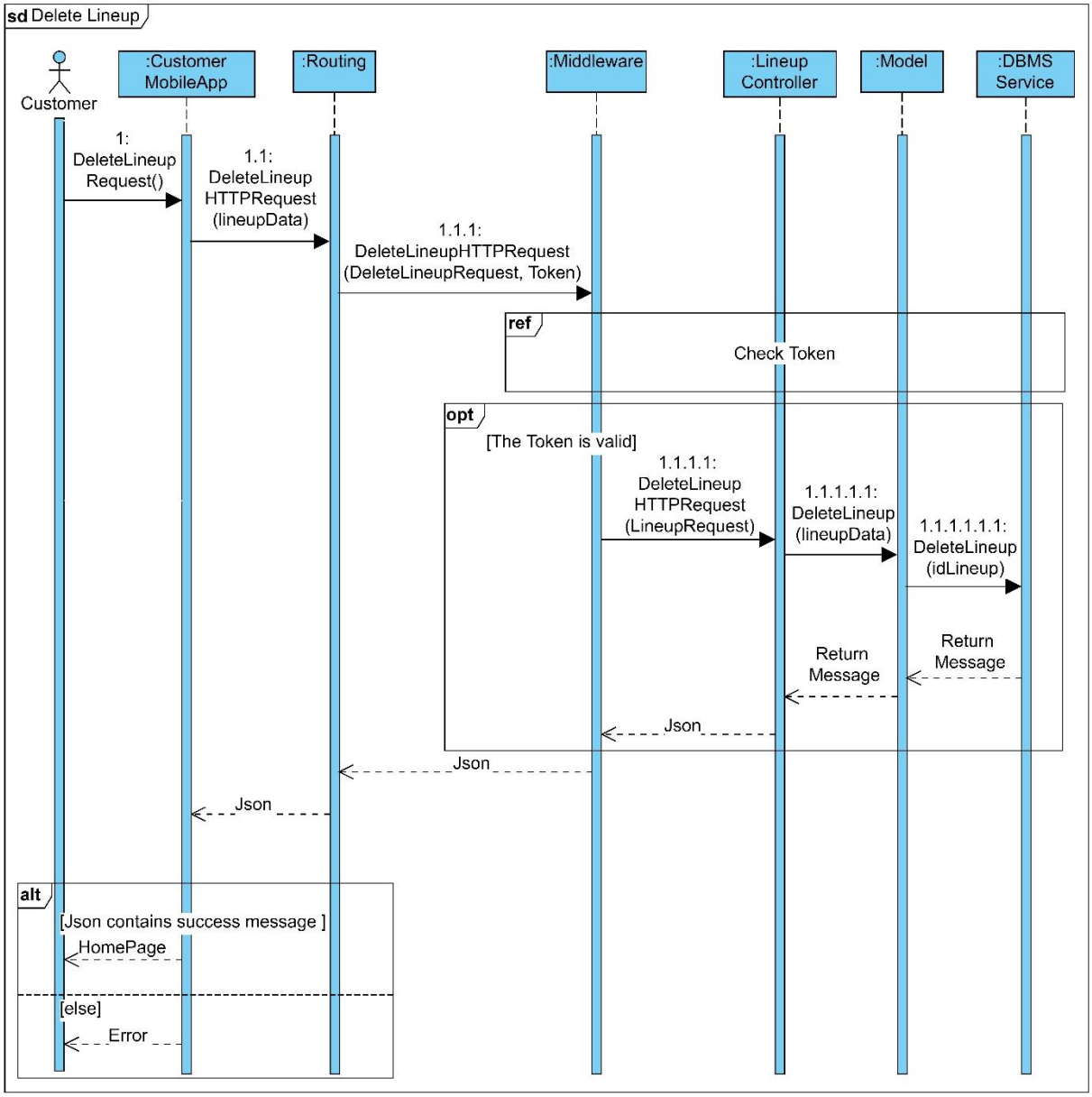
The sequence diagram above shows the process of deleting a customer's booking.

First, customers require the operation using the CustomerMobileApp, then, sends an HTTP request. This request passes through the Routing and the Middleware where the token is checked as described before.

If the token is valid, the HTTP request can be delivered to the BookingController that calls a method of the Model. Then, a query is executed in the DBMSService. A message that contains an ack about the request is backpropagated until the BookingController where it is serialized in a JSON file. This file traverses the Middleware and the Routing and arrives at the CustomerMobileApp. Here the Home Page and the updated list of active reservations are shown.

If the token is not valid the JSON contains an error message. Due to this, the CustomerMobileApp shows an error message too.

8. Delete Lineup for Customer



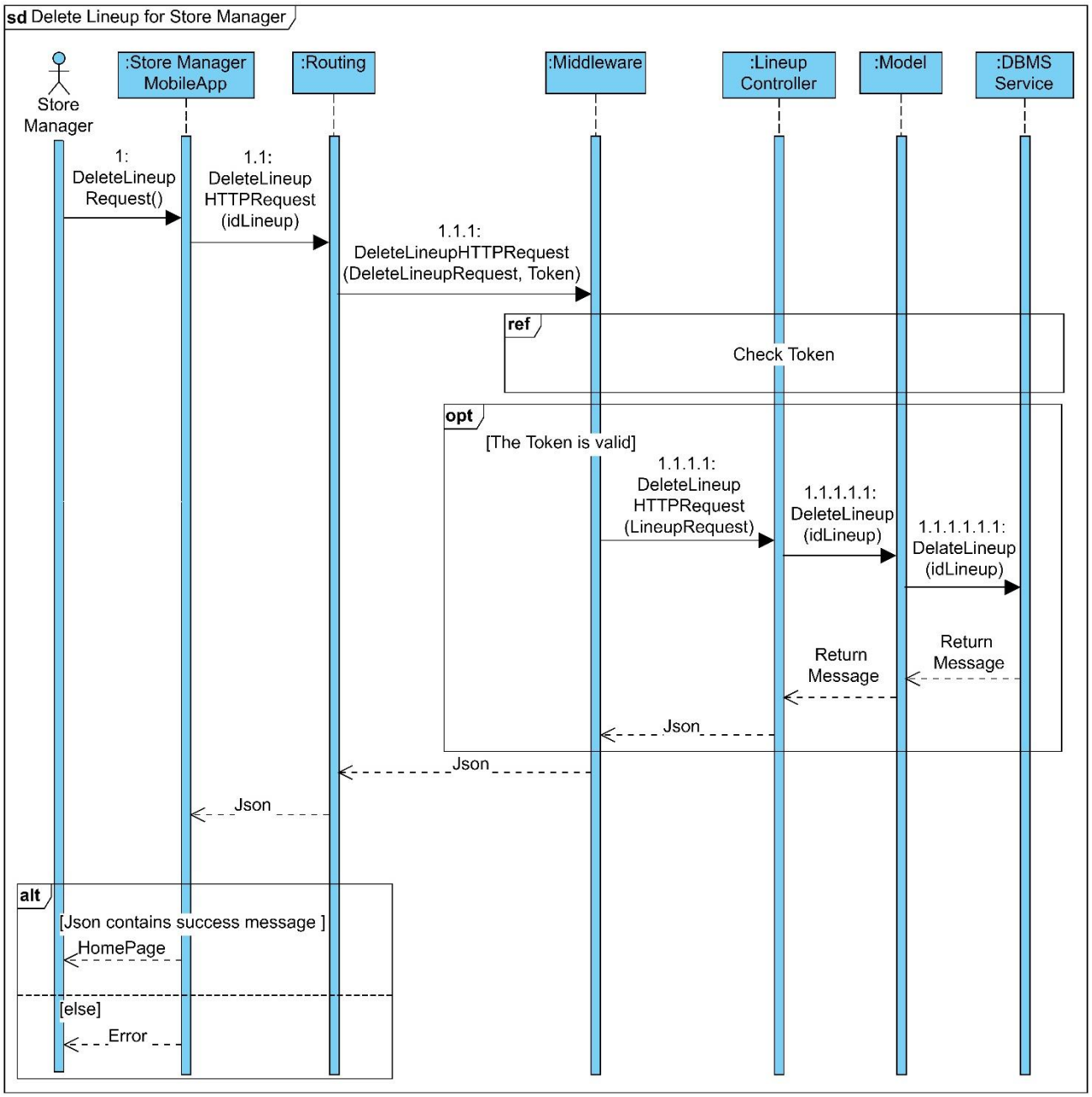
The sequence diagram depicts the phases of deleting a customer's lining up.

The customer requires the deleting by means of the CustomerMobileApp. This one sends an HTTP request that passes through the Routing and the Middleware. Here the token is checked as described before.

If the token is valid, the HTTP request can be forwarded until the LineupController where a method of the Model is called. Then, a query is executed in the DBMSService. An ack message is backpropagated until the LineupController where it is serialized in a JSON file. The JSON arrives at the CustomerMobileApp after it has been passed through the Middleware and the Routing. At this point, the Home Page and the updated list of active reservations are shown to the customer.

If the token is not valid, then the JSON contains an error message. Due to this, the CustomerMobileApp shows an error message instead of the Home Page.

9. Delete Lineup for Store Manager



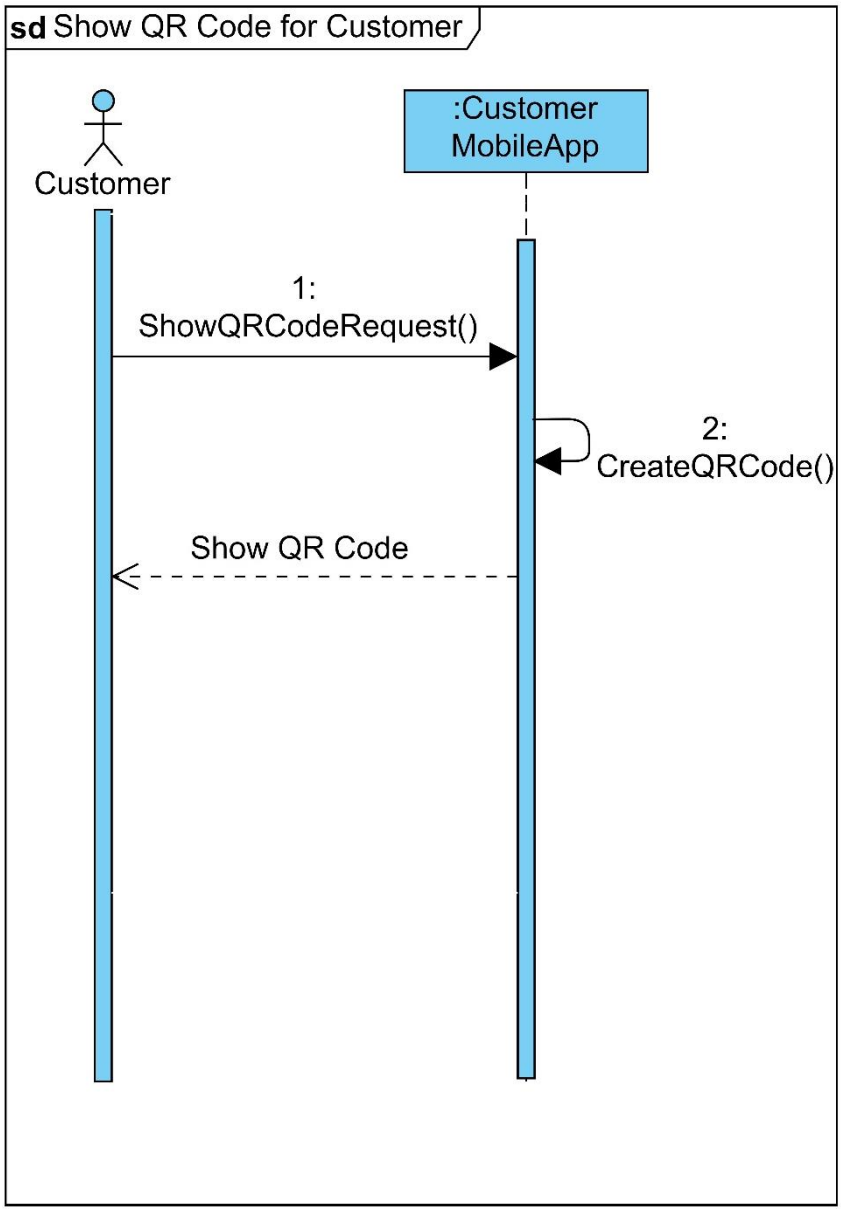
The sequence diagram shows the deleting of a store manager's lining up. The phases are similar to what is already shown for the customer's lineup deleting operation.

The store manager requires the deleting by means of the StoreManagerMobileApp. This one sends an HTTP request that passes through the Routing and the Middleware. Here the token is checked as described before.

If the token is valid, the HTTP request can be forwarded until the LineupController where a method of the Model is called. Then, a query is executed in the DBMSService. An ack message is backpropagated until the LineupController where it is serialized in a JSON file. The JSON arrives at the StoreManagerMobileApp after it has been passed through the Middleware and the Routing. At this point, the Home Page and the updated list of active reservations are shown to the store manager.

If the token is not valid, then the JSON contains an error message. Due to this, the StoreManagerMobileApp shows an error message instead of the HomePage.

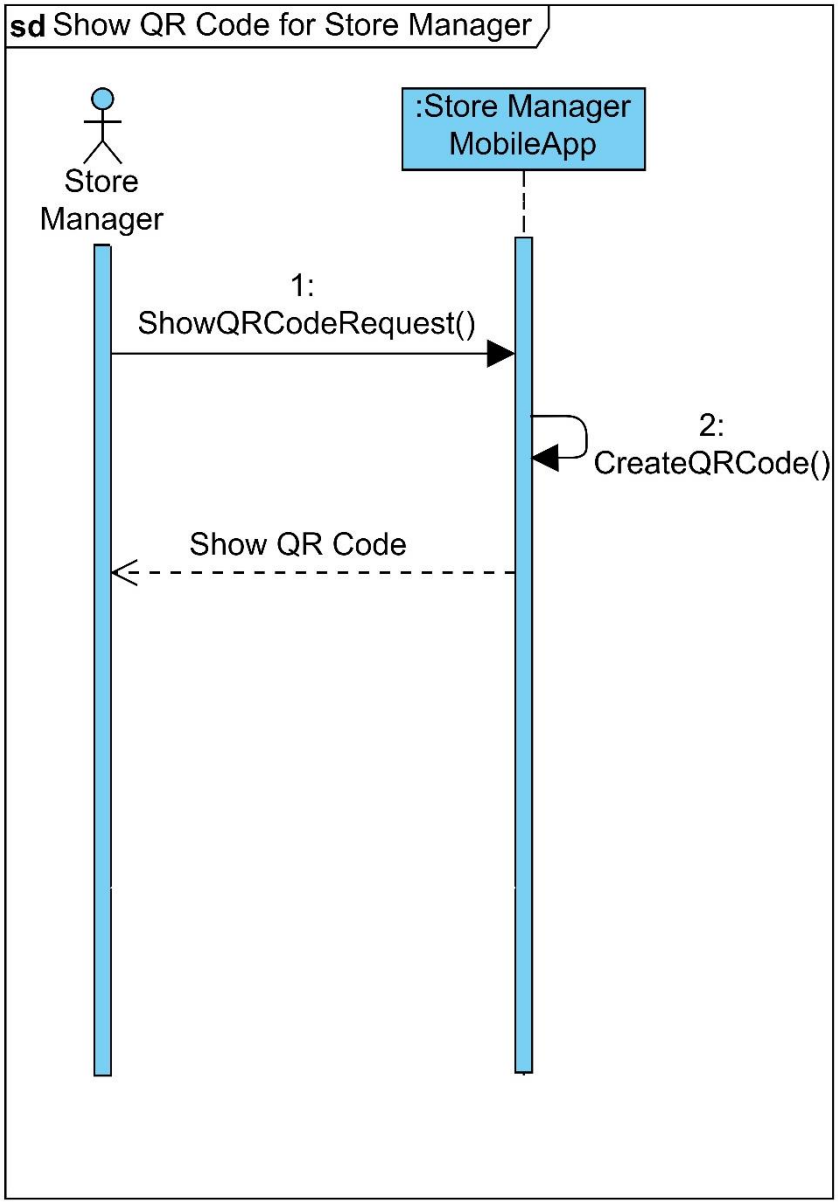
10. Show QR Code for Customer



The diagram shows the phases that allow the system to show the QR Code associated with a specific reservation at a customer. This code can be also printed by the customer.

The request is satisfied entirely on the side of the CustomerMobileApp. First, the customer requests the QR Code using the dedicated functionalities of the application. Then, the CustomerMobileApp generates the code and shows it.

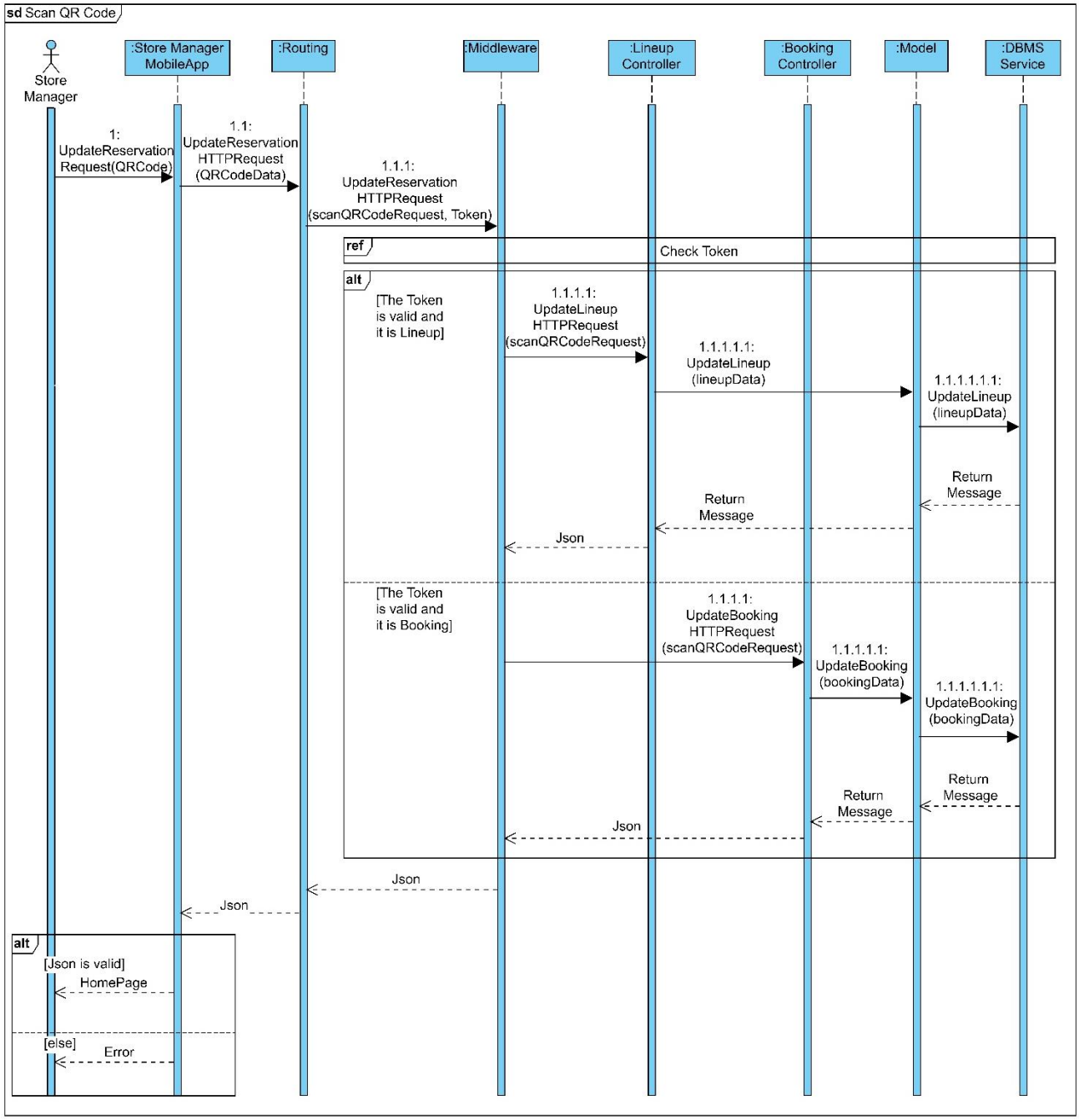
11. Show QR Code for Store Manager



The diagram shows the phases that allow the store manager to view, check, and print the QR Code.

The request is satisfied entirely on the StoreManagerMobileApp side. First, the store manager uses the dedicated functionalities of the application to require the QR code. Then, the StoreManagerMobileApp generates the code and shows it.

12. Scan QR Code



The sequence above depicts the operation of scanning a QR code performed by the store manager to allow customers to enter and exit the store.

As usual, the store manager selects the functionality on the StoreManagerMobileApp. Then the HTTP request is propagated across Routing and Middleware and the cheks on the token are done.

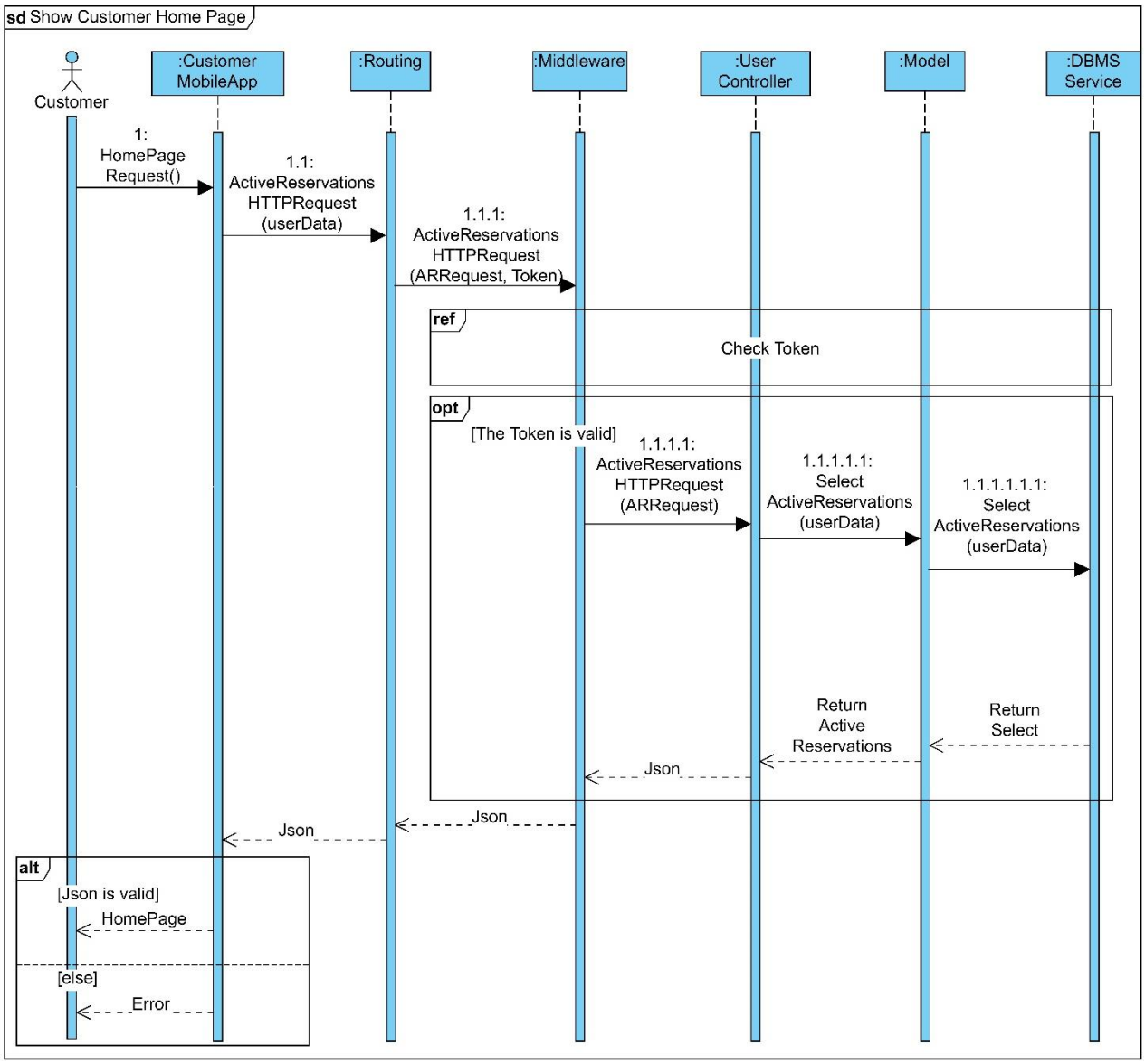
If the token is valid, there are two possibilities.

In the first, the QR code is associated with a lining up, so the request is propagated at the LineupController. Later, a suitable method of the Model is called and an update is done on the database-side. An ack message is returned with some information about the new reservation that is serialized in a JSON file by the LineupController. When this file has arrived at the StoreManagerMobileApp and has been checked, the HomePage is shown.

The second possibility happens when the QR code is associated with a booking. In this case, the previous operations are done referring to the BookingController. So, it calls a method of the Model and the update is required at the DBMS. Then, the information is returned and it is serialized in the JSON file by the BookingController. The file arrives at the StoreManagerMobileApp, it is checked and the Home Page is shown.

Otherwise, if the token is not valid, the reservation request is not propagated and the JSON contains an error message. This message is read by the StoreManagerMobileApp that shows an error message at the store manager.

13. Show Customer Home Page



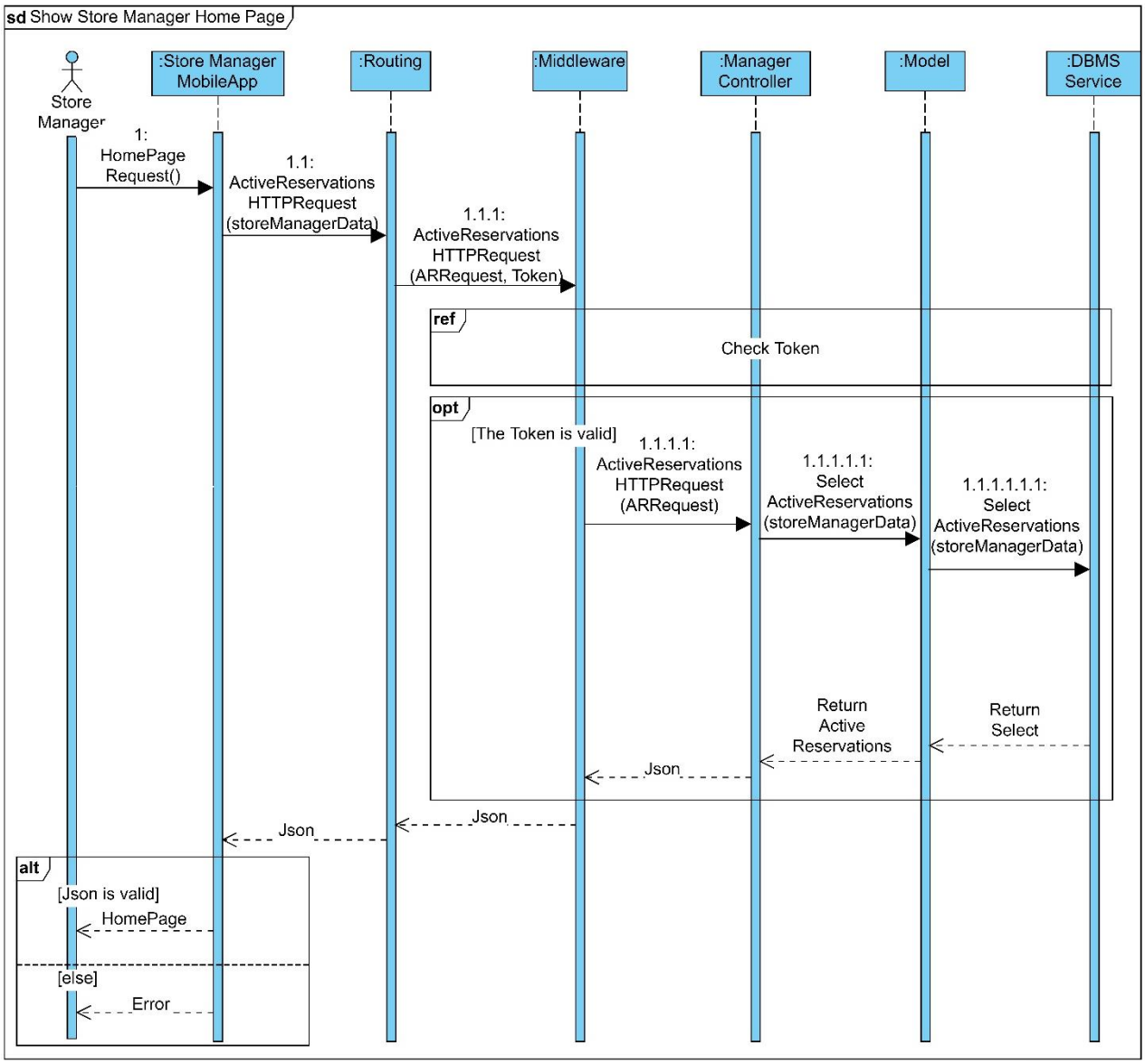
This sequence diagram shows the process that permits the visualization of the Home Page and its information by the customer.

The customer selects the section Home Page in the CustomerMobileApp. This page, as specified in the RASD, contains the list of active reservations. So, the correlated HTTP request is passed to the Routing and the Middleware where the authentication token is checked.

If the token is valid, the request proceeds across UserController and Model, and the select query is performed into the database. The result of the select operation is returned to the model first, and the UserController then. Here, the active reservations are serialized in a JSON file that is propagated through the Middleware and the Routing until the CustomerMobileApp. Lastly, the Home Page with the list of the active reservations is displayed.

If the token is not valid, an error message is serialized and the request is not propagated. The CustomerMobileApp shows an error message too.

14. Show Store Manager Home Page



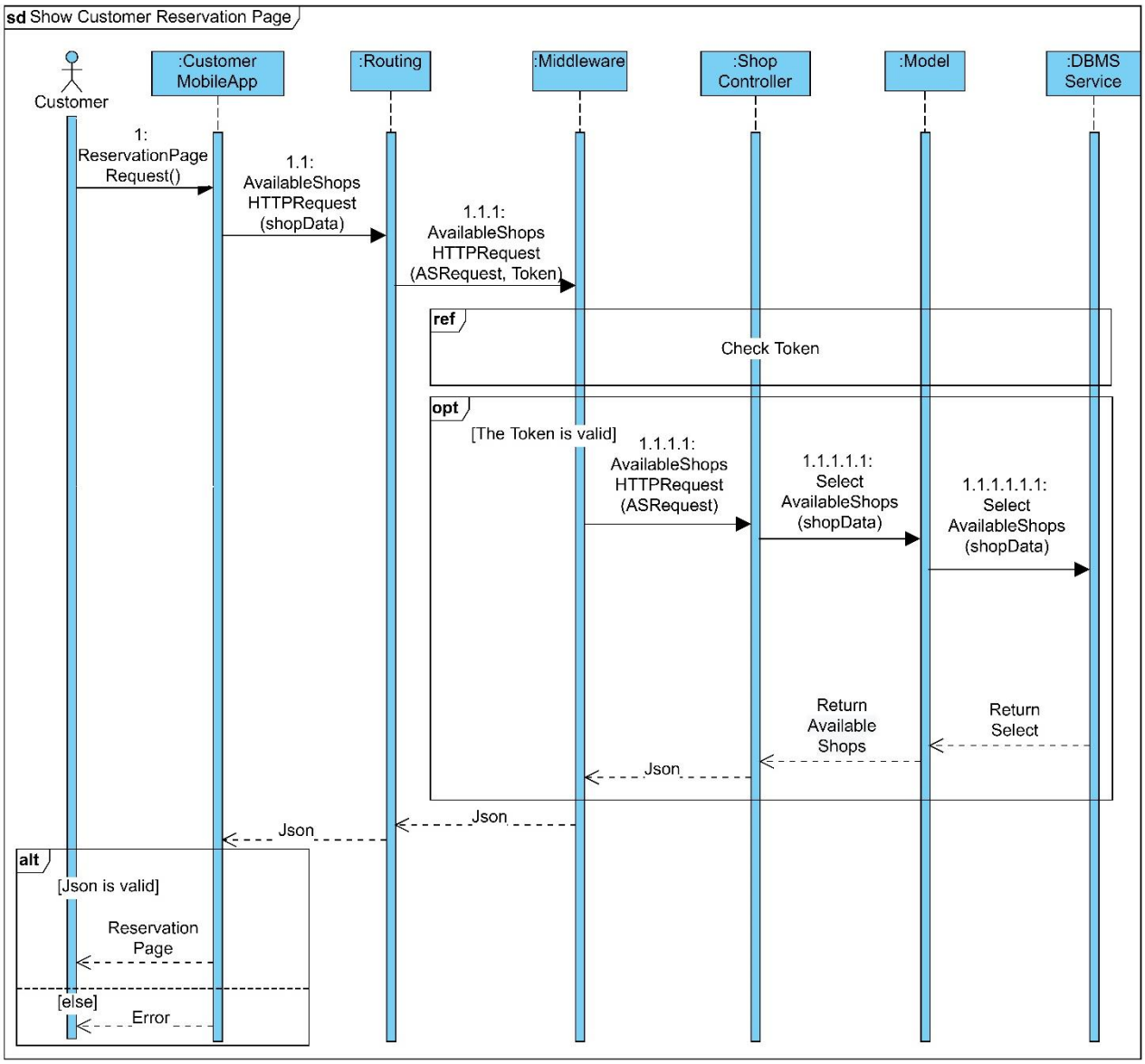
This sequence diagram shows the process that allows the visualization of the HomePage and its information by the store manager.

The store manager selects the Home Page in the StoreManagerMobileApp. This page, as specified in the RASD, contains the list of active reservations. So, the correlated HTTP request is passed to the Routing and the Middleware where the authentication token is checked.

If the token is valid, the request proceeds across ManagerController and Model, and the select query is performed into the database. The result of the select operation is returned to the model first, and the ManagerController then. Here, the active reservations are serialized in a JSON file that is propagated through the Middleware and the Routing until the StoreManagerMobileApp. Lastly, the Home Page with the list of the active reservations is displayed.

If the token is not valid, an error message is serialized and the request is not propagated. The StoreManagerMobileApp shows an error message too.

15. Show Customer Reservation Page



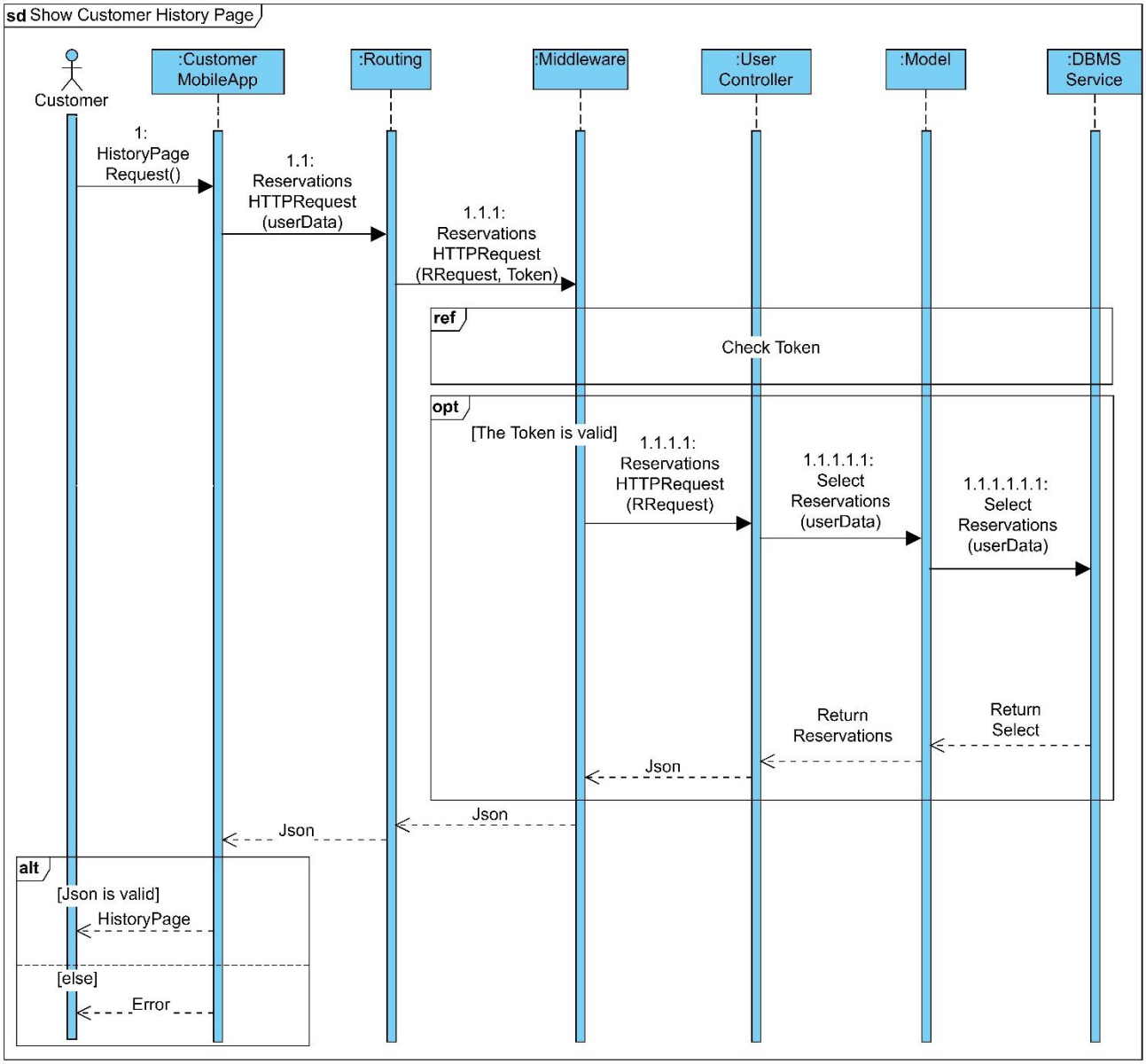
The sequence diagram explains the process to visualize the ReservationPage with the list of the available shops to the customer.

The customer requires the page at the CustomerMobileApp. This one sends an HTTP request to the Routing component that is propagated to the Middleware where the token is checked.

If the token is valid, the request can proceed in direction of the ShopController. It calls a method of the Model and a query to obtain the list of the available shops is performed into the database. The result of the operation is returned to the ShopController across the Model. Here a JSON file containing the information is serialized and it is propagated through the Middleware and the Routing to achieve the CustomerMobileApp. This one shows the ReservationPage with the list of the available shops.

If the token is not valid, the request is not propagated and the JSON contains an error message. When it reaches the CustomerMobileApp, it shows an error message accordingly.

16. Show Customer History Page



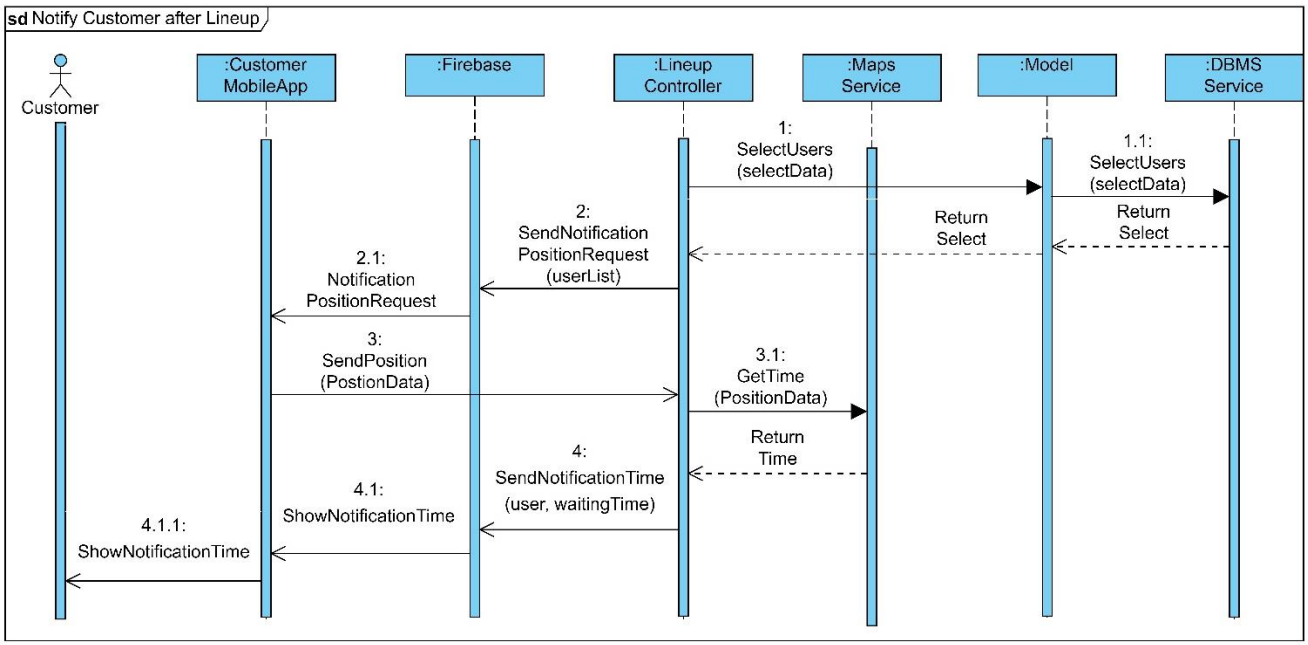
The diagram below depicts the process to show the HistoryPage to the customer.

First, the customer selects the dedicated section on the CustomerMobileApp. Then, the request is propagated through the Routing component until the Middleware where the token is checked as already seen in the other diagrams.

If the token is valid, the request is transmitted to the UserController. It calls a method of the Model and a select query is done into the database. The result of the query is the list of the reservations and they are returned first at the Model and then at the UserController. The list is serialized in a JSON file and sends through the Middleware and the Routing to the CustomerMobileApp. Here the file is inspected and, if valid, the CustomerMobileApp shows the HistoryPage with the list of the reservations.

If the token is not valid, then the request is not propagated as usual and the JSON file contains an error message. This error is received and read by the CustomerMobileApp that displays an error message to the customer.

17. Notify Customer after Lineup



The diagram shows the process of notification to alert the customer that her/his turn is near to being called. To be more precise, it shows the case in which a customer that has lined up exit the store.

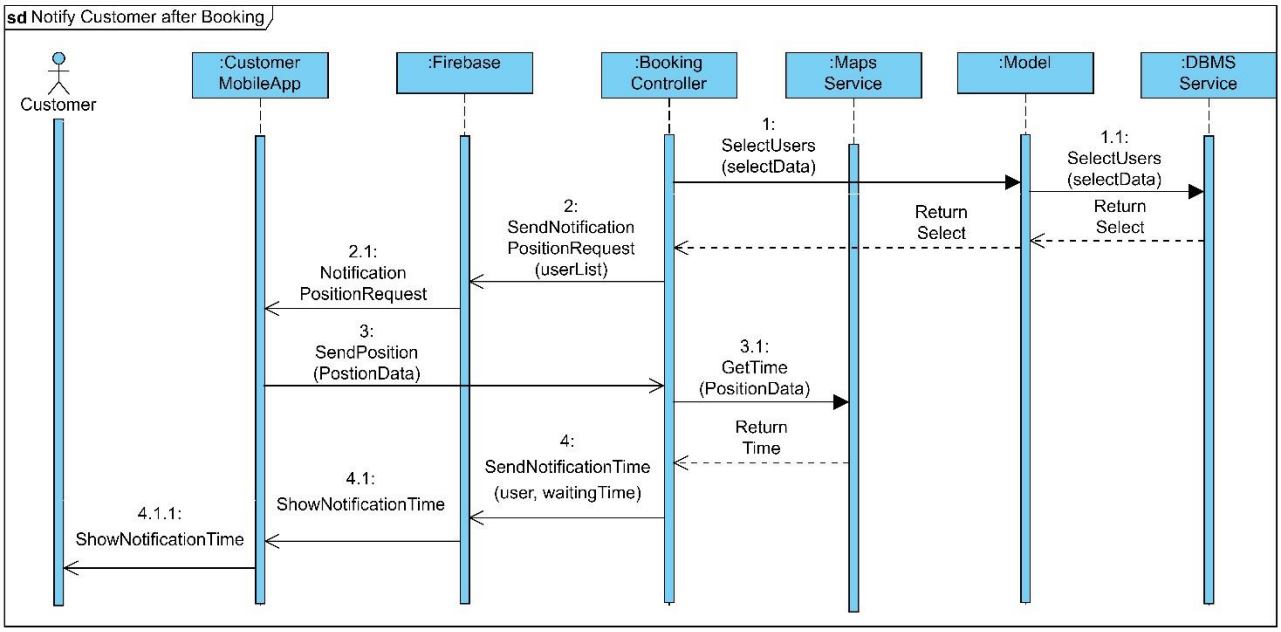
After the QR code of the customer that exits the store has been scanned, the LineupController calls a method of the Model to check the next customers who want to enter the store. Notice that the check is performed both on customers that have got a booking and who have got a lining up. The choice to involve the LineupController depends on the nature of the reservation of the customer that exits the store.

Then, a query is executed in the database and the result is returned to the Model and the LineupController. Through Firebase, a message is received by the CustomerMobileApp. The application sends back the information about the current position of the user to the LineupController that calls the MapsService to obtain an estimation of the time required to approach the store. Then, the time estimation reaches the CustomerMobileApp passing through Firebase.

Lastly, the application shows a notification to the customer. In particular, the notification should display an estimation of the departure time needed to guarantee that the customer arrives in time for her/his shift.

For the sake of simplicity and brevity, the diagram shows the process for a single CustomerMobileApp and a single customer. Actually, each time a customer exit the store, more than one customer is alerted that their shifts are near to being called. At this level, it is possible to assume that the operation is executed involving the next three customers who are waiting to enter the store.

18. Notify Customer after Booking



The diagram shows the process of notification to alert the customer that her/his turn is near to being called. To be more precise, it shows the case in which a customer that has booked a visit exit the store.

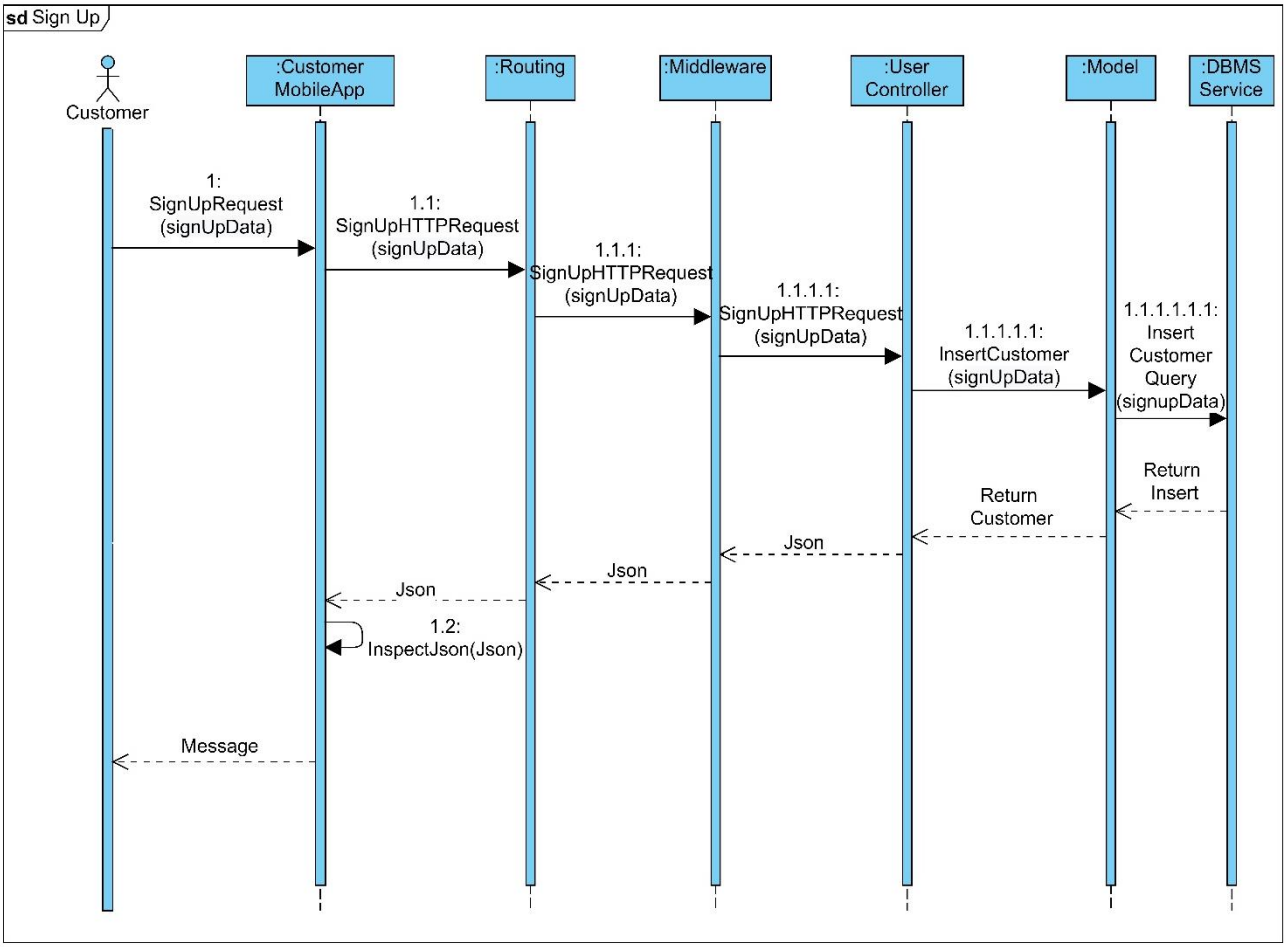
After the QR code of the customer that exits the store has been scanned, the BookingController calls a method of the Model to check the next customers who want to enter the store. Notice that the check is performed both on customers that have got a booking and who have got a lining up. The choice to involve the BookingController depends on the nature of the reservation of the customer that exits the store.

Then, a query is executed in the database and the result is returned to the Model and the BookingController. Through Firebase, a message is received by the CustomerMobileApp. The application sends back the information about the current position of the user to the BookingController that calls the MapsService to obtain an estimation of the time required to approach the store. Then, the time estimation reaches the CustomerMobileApp passing through Firebase.

Lastly, the application shows a notification to the customer. In particular, the notification should display an estimation of the departure time needed to guarantee that the customer arrives in time for her/his shift.

For the sake of simplicity and brevity, the diagram shows the process for a single CustomerMobileApp and a single customer. Actually, each time a customer exit the store, more than one customer is alerted that their shifts are near to being called. At this level, it is possible to assume that the operation is executed involving the next three customers who are waiting to enter the store.

19.SignUp



The above sequence diagram shows the SignUp process for a generic customer.

First, the customers insert the data for the SignUp using the CustomerMobileApp and then they require the registration. The application sends an HTTP request that is propagated through the Routing and the Middleware until it reaches the UserController.

The UserController calls a method of the Model and then a query is executed in the DBMSService. The ack and the information about the SignUp are returned to the UserController and then it is serialized in a JSON file. This file then reaches the CustomerMobileApp where it is checked. The CustomerMobileApp shows, in the end, a confirmation message.

If the data are not valid, the JSON file contains an error message that is shown to the customer.

2.5 Component interfaces

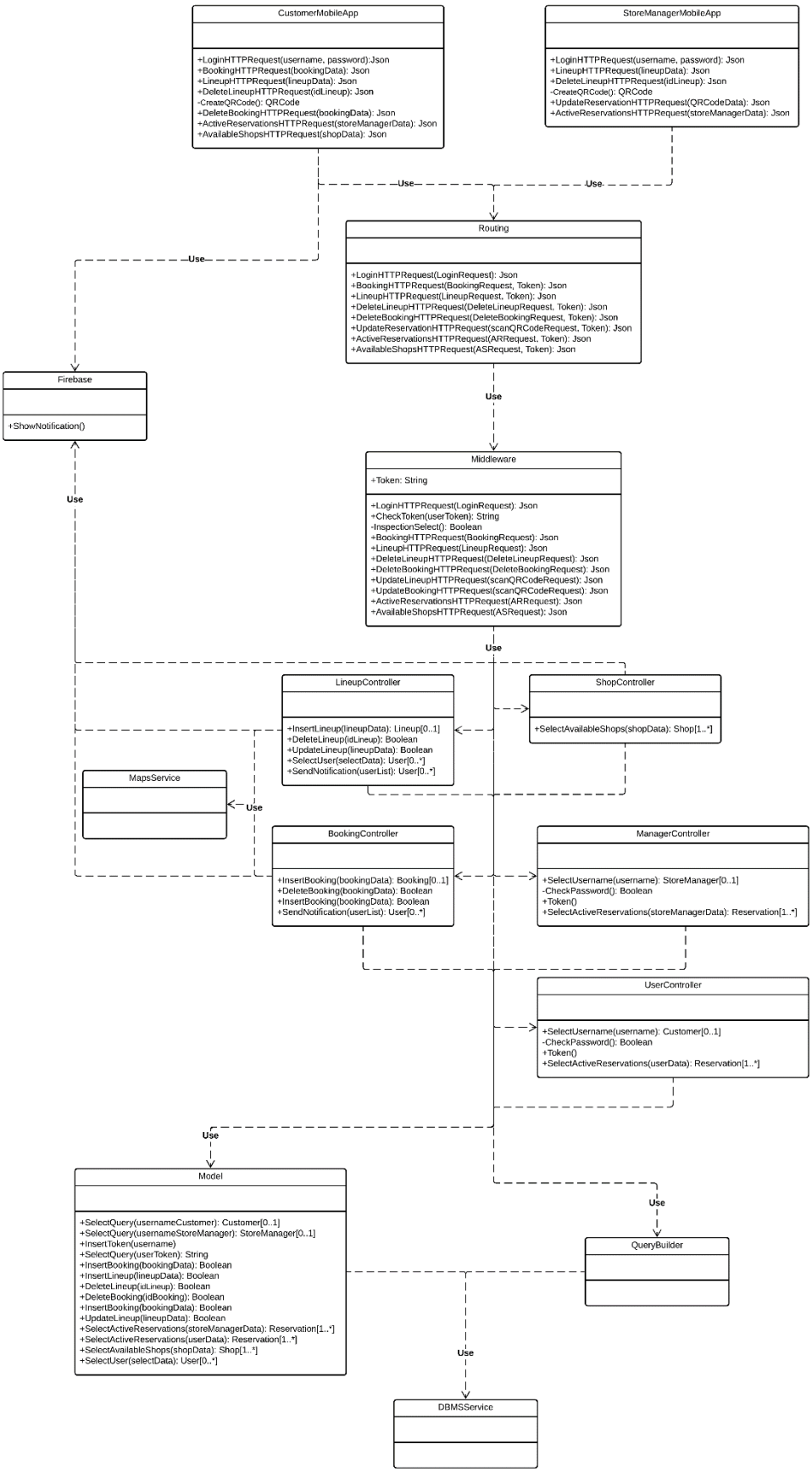


Figure 4: Component Interfaces Diagram

The diagram above shows the most important methods for the components described in this document and the main interfaces between them.

Please notice that it is a high-level representation. The actual implementation of the components might impose some variation in the names and details about methods and interfaces.

2.6 Selected architectural styles and patterns

The system is based on a three-tier architecture, which is an architecture style composed of three main parts: presentation tier, application tier, and data tier. In more details:

- the **presentation tier** is represented by the CustomerMobileApp and StoreManagerMobileApp;
- the **application tier** includes the Routing, the Middleware, the Controllers, and the Models;
- the **data tier** is composed of the database.

This choice is given because the application should guarantee high reliability, high security, and high scalability.

As described in the previous sections of this document, the communication between the components on the client-side and the ones on the server-side is provided by means of HTTP requests.

Each controller can also serialize files in JSON format to return meaningful information to the CustomerMobileApp or StoreManagerMobileApp. This information might be the token used for the authentication, the confirmation and error messages (e.g. successful sign up), the data about the reservations (e.g. active reservations which are shown in the HomePage, history of reservations), and so on.

Moreover, the connection between the application and the server must be safe. For this purpose, it is used the TLS (*Transport Layer Security*) protocol and the needed SSL/TSL certificate.

Following two patterns on which the system is based.

Model View Controller (MVC)

The MVC is a pattern that permits a good level of decoupling of the code. It is applied by the definition of three distinct blocks:

- **Model**: allow to access and manage data through a series of methods. It is used by the controller;
- **View**: presents the data in a readable and understandable form for the user;
- **Controller**: takes the inputs and forward them to the Model after some operations like validation actions.

Singleton

A token is a unique string needed anytime the presentation tier wants to communicate with the application tier. This token and other user information that are useful for the presentation tier have to be stored in an object that has to be kept for the entirely CustomerMobileApp or

StoreManagerMobileApp lifecycle. This pattern avoids the instantiation of this object in each view of the CustomerMobileApp and StoreManagerMobileApp that would be a very inefficient choice.

2.7 Other design decisions

The system will be implemented using the Laravel framework.

Laravel is a PHP web application framework that provides a starting point for the development of applications and projects. Some strengths are that Laravel:

- provides several tools for dependency injection, unit testing, queues, and real-time events;
- hight scalability, especially horizontal scalability;
- huge support from the community.

Moreover, it is a good choice because Laravel makes easier and more efficient future developments. For instance, it is easy to develop a web application that allows the store managers to manage the store thanks to the possibility to build both the front-end and the back-end of the system with Laravel.

Lastly, there is available a great number of libraries that allow saving time in the development of common features encouraging the reuse of well-tested solutions. In other words, Laravel aims to the reusability of the code.

2.8 Algorithms

2.8.1 Pseudocode

Each reservation is associated with a status. This status can assume three different values:

- **Wait**: the reservation is active and the customer is waiting for her/his shift;
- **Enter**: the customer is into the shop;
- **Exit**: the customer has left the shop and the reservation is not active anymore.

Following is shown the pseudocode that sketches the operation that the system has to do each time a customer leaves the shop.

```
if (Exit the store){
  if(Is Booking?){
    //change the status of the booking
    booking.status=Exit;
    //update the counter of current people into the store
    counter--;
  }
  elseif(Is Lineup?){
    //change the status of the lineup
    lineup.status=Exit;
    //update the counter of current people into the store
    counter--;
  }
}
```

Figure 5: Exit the Store - Pseudocode

Each time a customer exit the store, a new one takes her/his place. The following pseudocode illustrates the basic operations that manage this process. The ID of a reservation is a unique, incremental code, so it suffices to check this value to understand the order of entering. Capacity is the maximum number of people that can enter the store at the same time.

```
while (counter<capacity){
    if(Is there a booking?) {
      //select the minimum ID from the active bookings
      getMinBookingID;
      //authorize the booking to enter
      booking.status=Enter;
      //update the counter of current people into the store
      counter++;
    }
    elseif(Is there a line up?) {
      //select the minimum ID from the active lining ups
      getMinLineUpID;
      //authorize the lining up to enter
      lineup.status=Enter;
      //update the counter of current people into the store
      counter++;
    }
}
```

Figure 6: It is Your Turn - Pseudocode

2.8.2 Flowcharts

Lastly, a series of flowcharts corresponding with the above pseudocode examples.

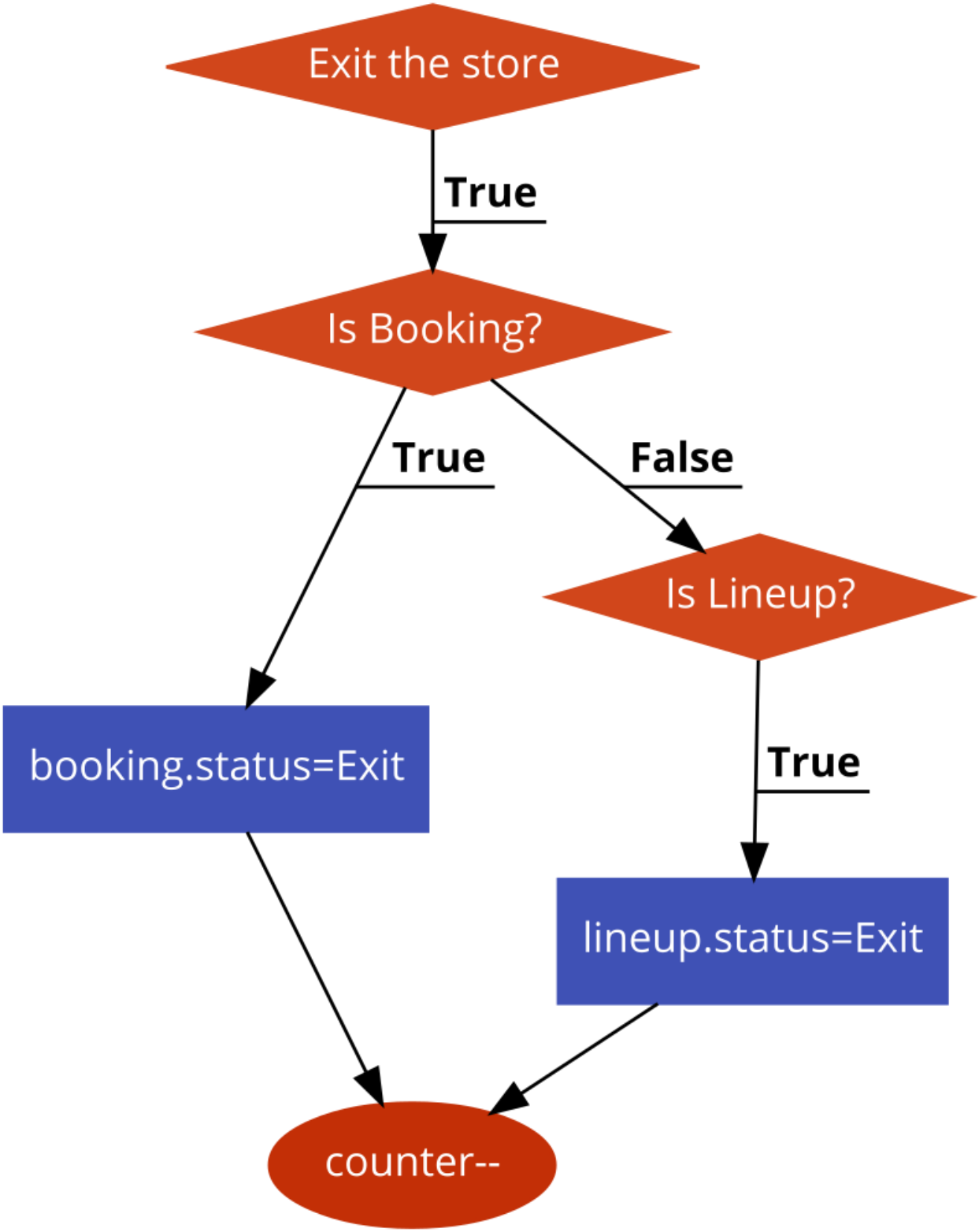


Figure 7: Exit the Store – Flowchart

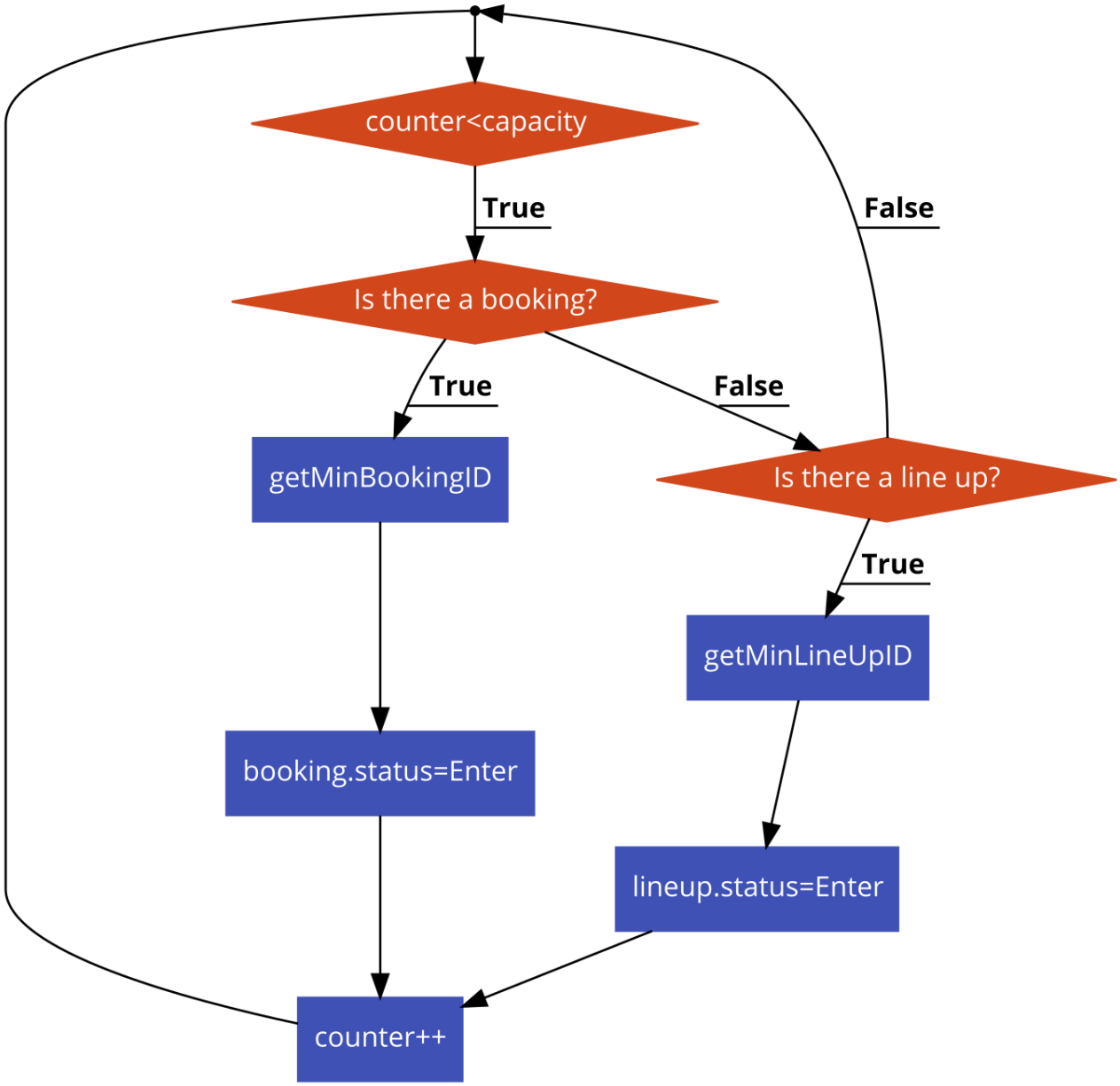


Figure 8: It is Your Turn – Flowchart

3 USER INTERFACE DESIGN

In this section of the document, it is provided an overview of how the *User Interface* (UI) of the system will look like. Let us assume that the application is divided into several pages dedicated to the different main functionalities and that there are two versions of the application. One version is dedicated to the customers, while the other one is focused on the store manager. The following images show an idea of these pages.

3.1 Mock-up Customer Application

Figure 10: Customer Initial Page mock-up

Figure 11: Customer Login Page mock-up

Figure 10 represents the page that CLup shows to customers during their very first access. On this page, they are able to reach the Login Page if they are already registered to the platform or create a new account if it is the first time that they use the software.

Figure 11 illustrates the page that allows customers to insert their credentials and then access the functionality of CLup. The required credentials are the email and the password set during the Sign Up process.

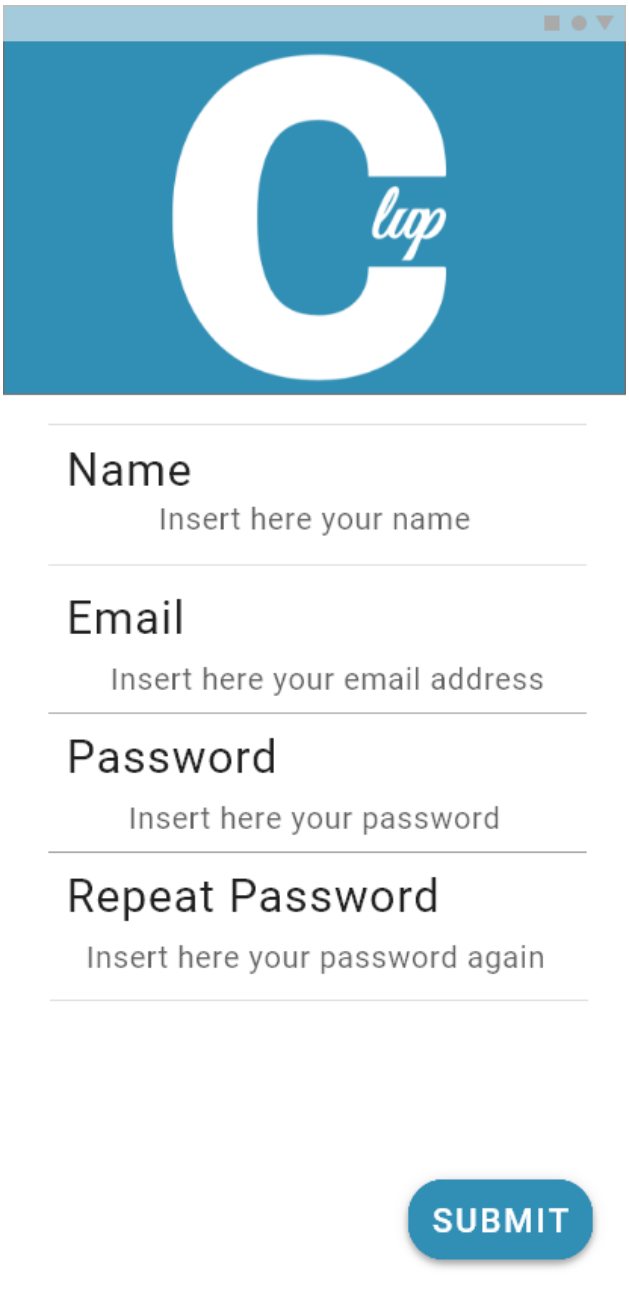


Figure 12: Customer Sign Up Page mock-up

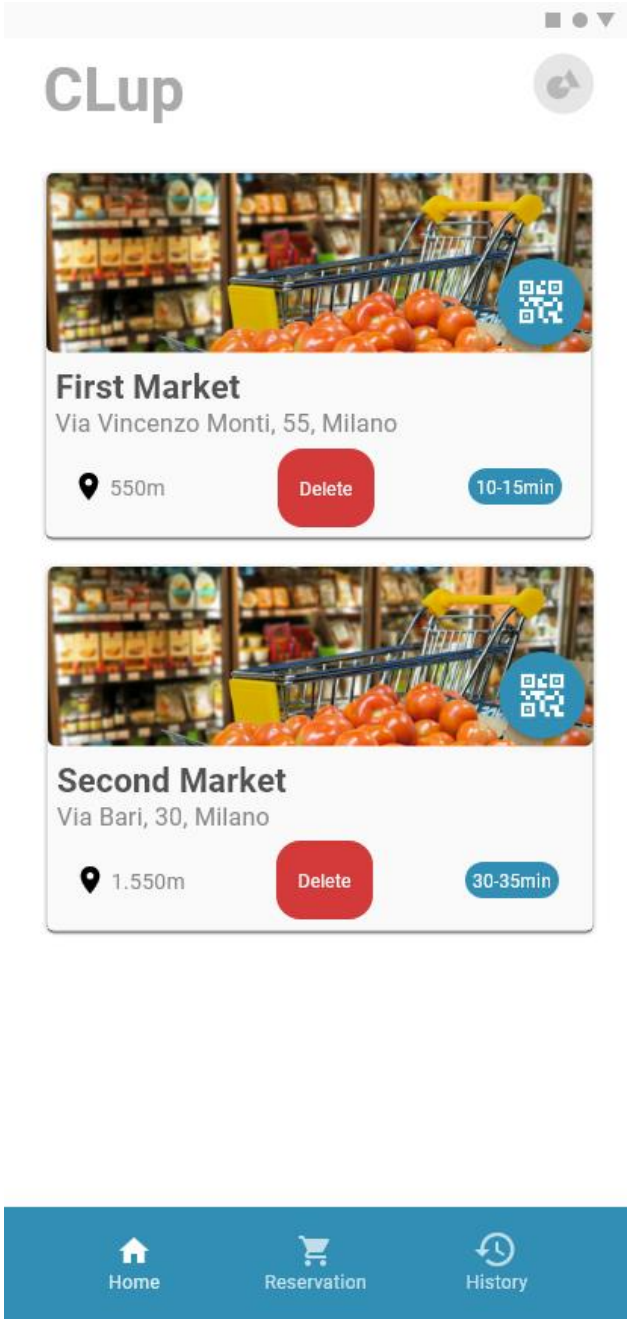


Figure 13: Customer Home Page mock-up

As anticipated before, customers can also submit their data and create a new account that allows access to the platform. The user profile is used to associate to each user the list of reservations (both active and non-active) and the duration of the previous visits to the stores used to improve the automatic estimation of the waiting time. An idea of the dedicated page is provided in *Figure 12*.

Due to the focus on privacy, the required data are only the essential ones. They are:

- Name
- Email
- Password
- Confirm password (to prevent typing mistakes)

Moreover, *Figure 13* depicts a mock-up of the Home Page of the application. It contains the main features like the list of active reservations with the possibility to delete them or to see the expected waiting time and some data about the shops. Furthermore, on this page, customers can open and use the QR code associated with each reservation.

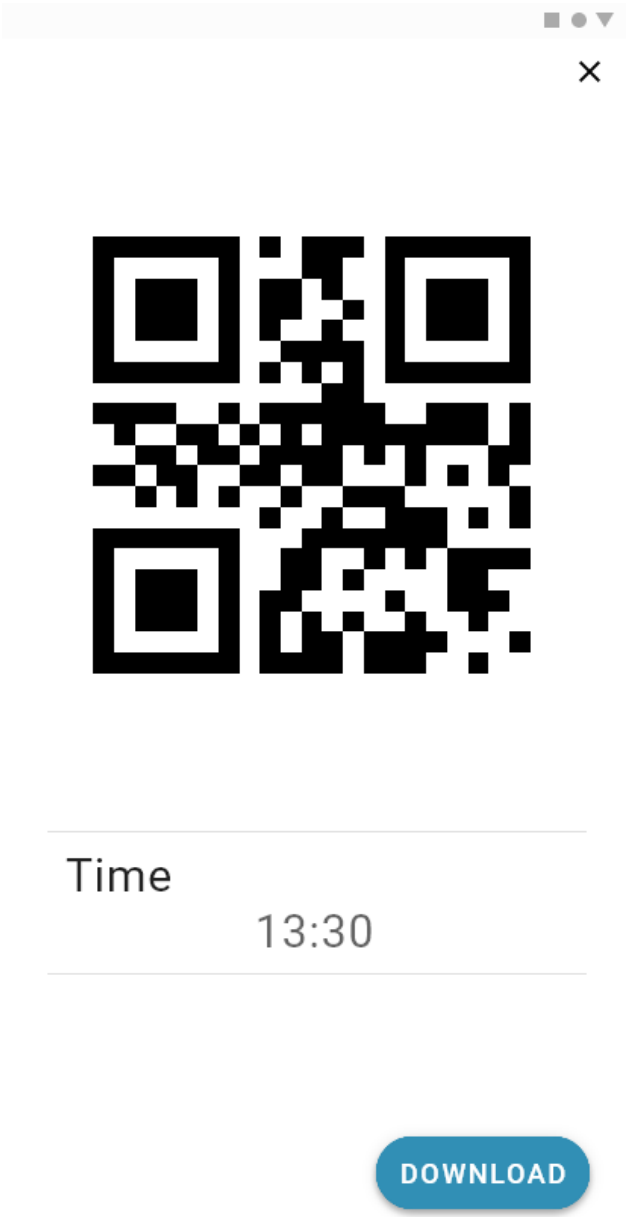


Figure 14: Customer QR Page mock-up

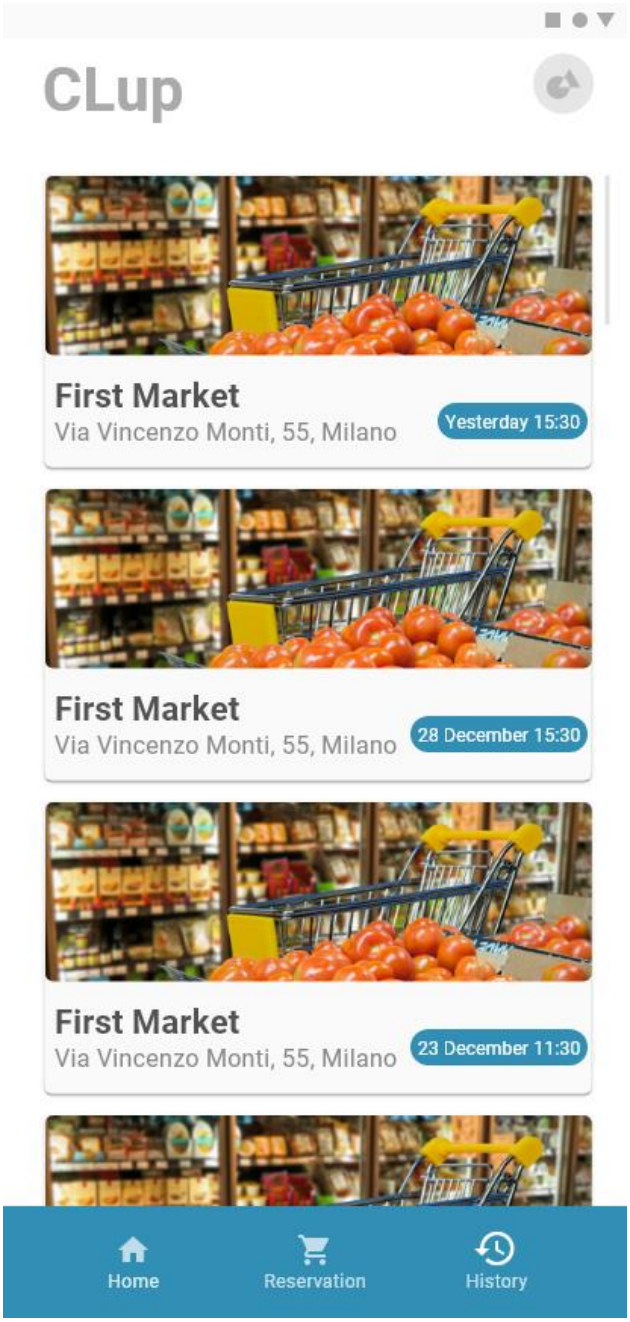


Figure 15: Customer History Page mock-up

On the left (*Figure 14*) a sketch of the page that shows the QR code associated with each reservation. Thanks to this page, the customers can scan their code when they enter and exit the store as expected. Moreover, as specified during the phase of requirements definition, specific functionality is provided to allow downloading the QR code in the PDF format to make easier the printout.

On the right (*Figure 15*) is shown a possible design of the History Page. Here, customers can access information about expired reservations with the indication of the shop (and its general data), the date and time of the reservation.

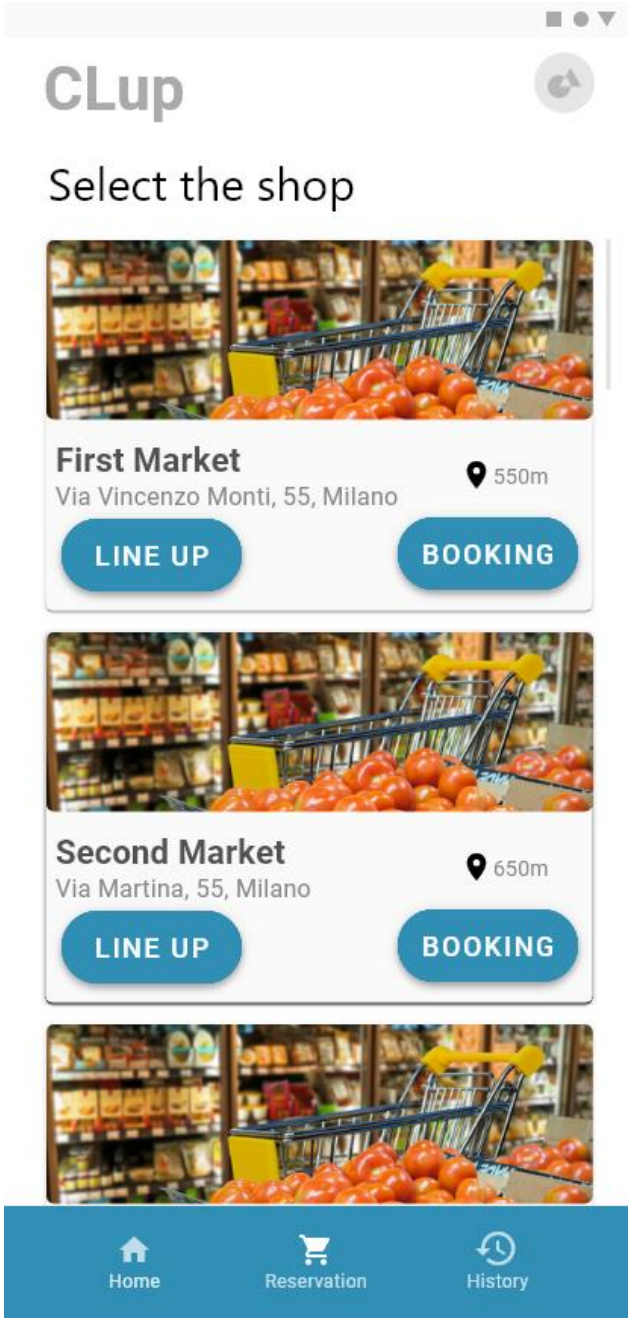


Figure 16: Customer Select Shop mock-up

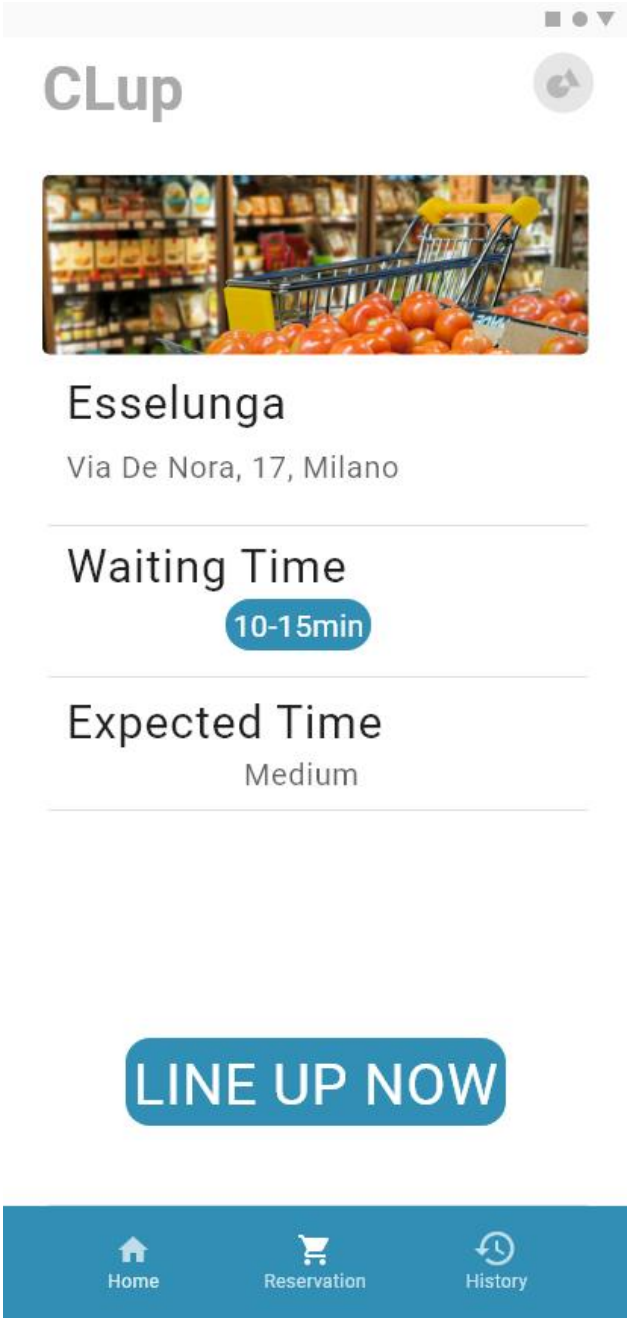


Figure 17: Customer Line Up Page mock-up

Figure 16 represents the Reservation Page that contains the list of active shops to permit customers to select the desired one. Each card of each store contains some information about the supermarket (e.g. position, photo) and the links to access lining up and booking functionalities.

Figure 17 shows the Line Up Page where customers can specify an estimation of the duration of the visit that is used by the system to better infer the waiting time.

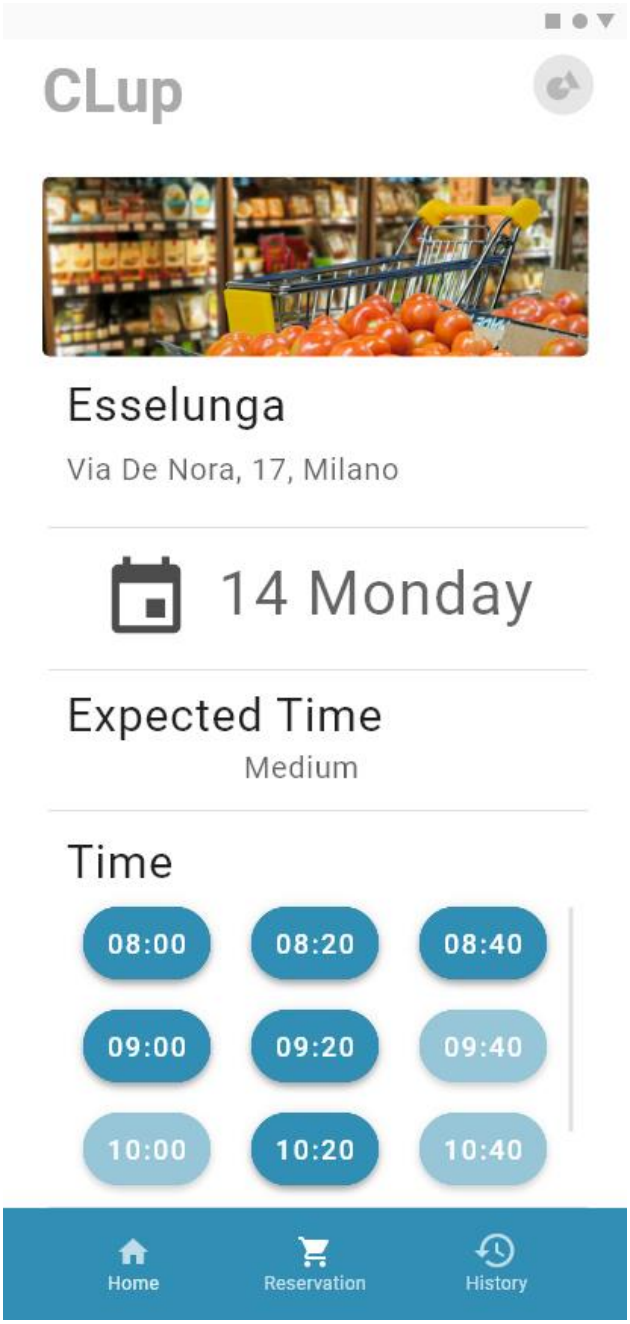


Figure 18: Customer Booking Page mock-up

Figure 18 gives a representation of how the Booking Page will look like. This page allows to select the date and the time slot that is desired by the customer. Customers must also specify the expected duration of the visit.

3.2 Mock-up Store Manager Application

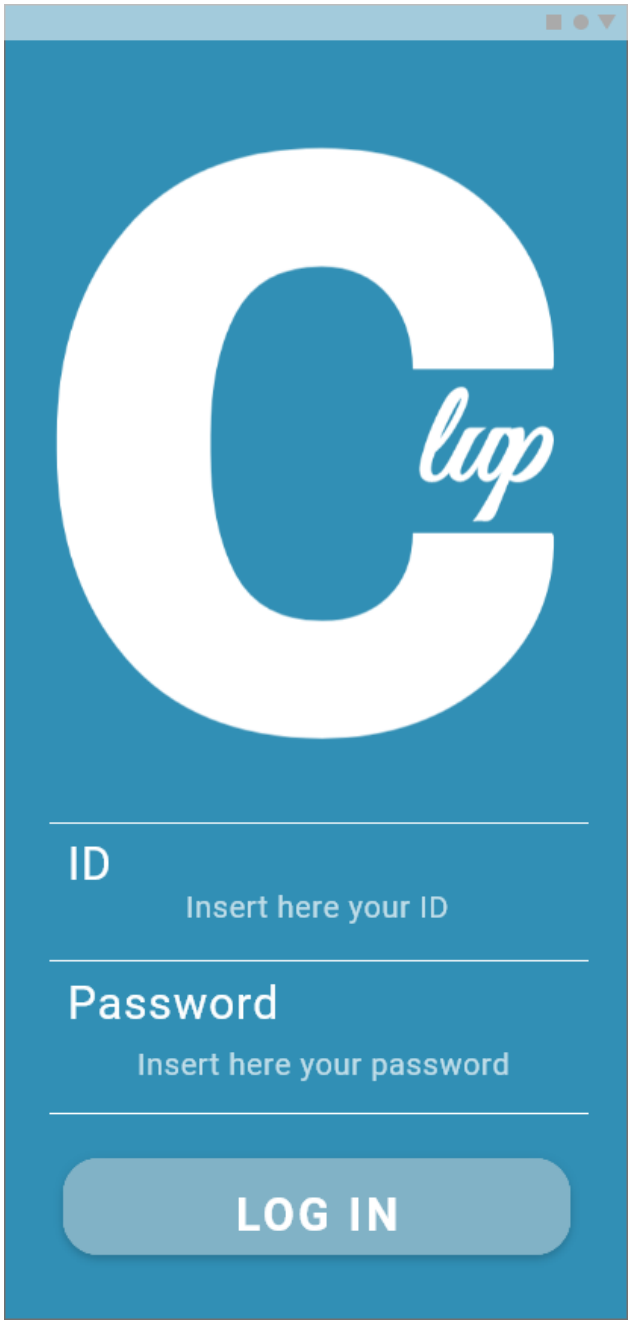


Figure 19: Store Manager Login Page mock-up

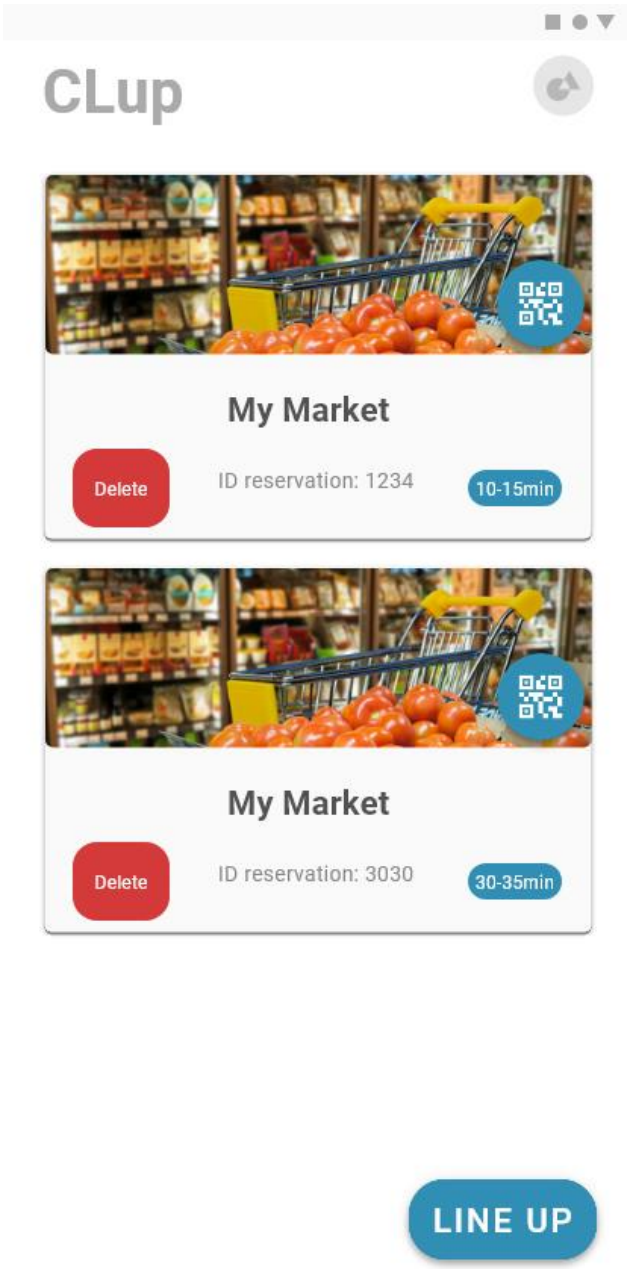


Figure 20: Store Manager Home Page mock-up

Figure 19 shows the version of the Login Page for Store Manager. In this case, the required credentials are the unique ID and the password.

On the other hand, *Figure 20* depicts the Home Page of the application dedicated to store managers. This page contains some functionalities already seen in the version for customers like the possibility to delete a lining up and to check the expected waiting time. Moreover, in this version, the page contains also a direct link to the Line Up Page and an indication of the identifier of each reservation. This identifier is printed with the QR code and delivered to the customer and is used to associate each Line Up to each customer without requires her/his data (e.g. name). Lastly, on this page, store managers can open and print the QR code associated with each lining up.

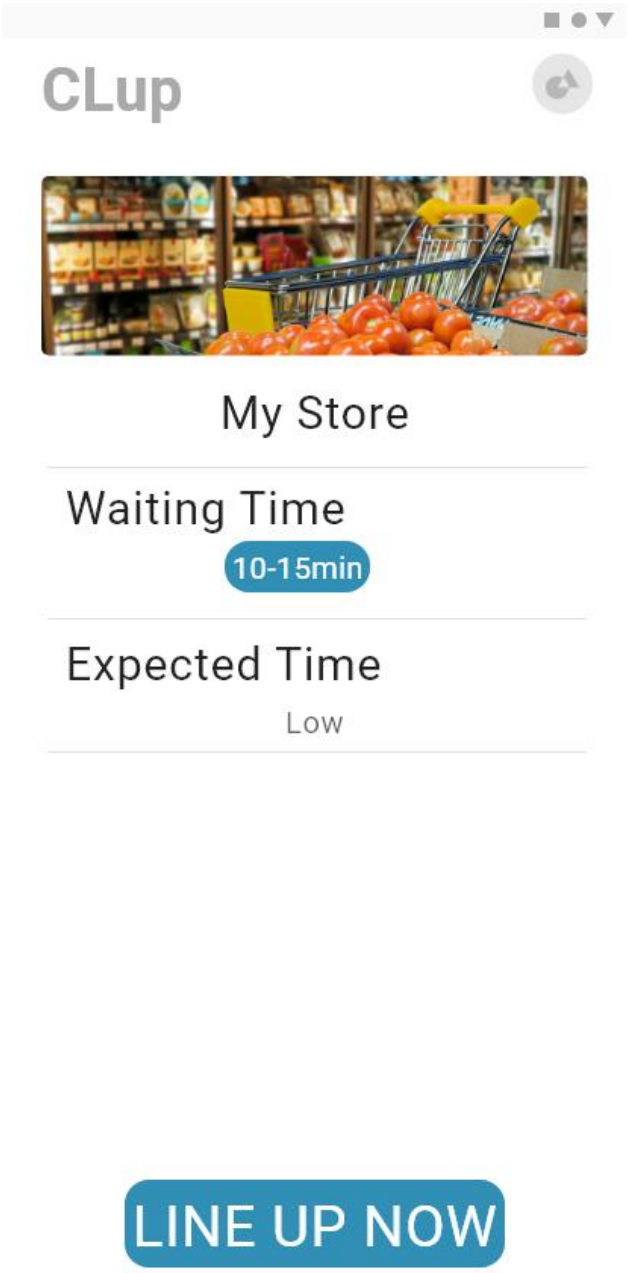


Figure 21: Store Manager Line Up Page mock-up

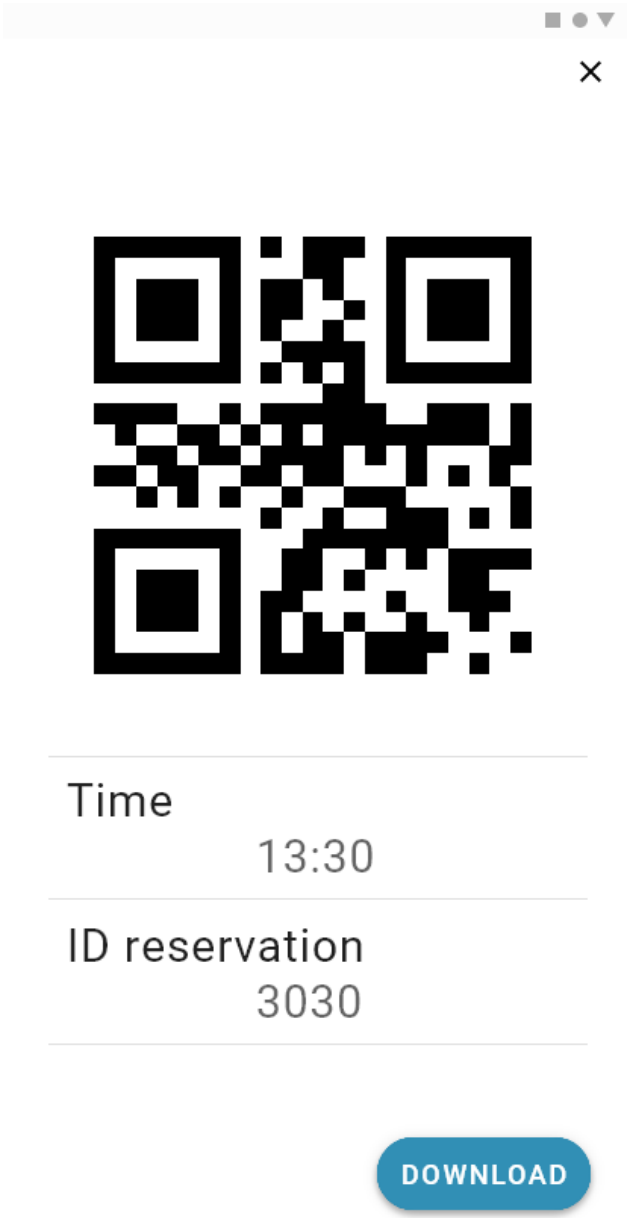


Figure 22: Store Manager QR Page mock-up

On the left (*Figure 21*) the Line Up Page. Here, the information about the waiting time is provided. Moreover, the page requires an estimation of the duration of the visit that is used by the system to better infer the waiting time.

On the right (*Figure 22*) an idea of the QR Page associated with each reservation. Thanks to this page, the store manager can provide the customer with an indication of the ID reservation and the estimated time. Furthermore, it is given the possibility to download the QR code in PDF format to make easier the printout.

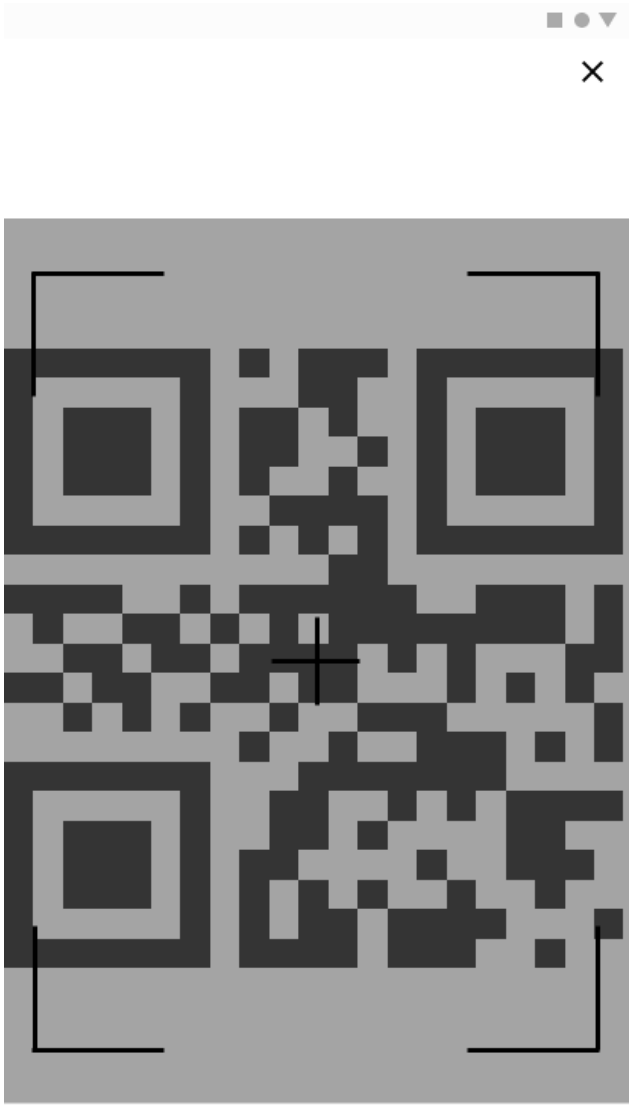


Figure 23: Store Manager QR Scanner Page mock-up

Figure 23 represents a possible UI of the QR code scanner. The store manager can scan the QR code shown by the customer to authorize the entering to or the exit from the store.

4 REQUIREMENTS TRACEABILITY

In this section, it is provided the mapping between the requirements defined in the RASD document and the components defined above. First, it is provided a general overview that gives the possibility to verify that all requirements are satisfied with the described components. Then, a more detailed analysis is presented.

ID	COMPONENT
C.1	Customer Mobile App
C.2	Store Manager Mobile App
C.3	Routing
C.4	Middleware
C.5	UserController
C.6	BookingController
C.7	LineupController
C.8	ShopController
C.9	ManagerController
C.10	Query Builder
C.11	Model
C.12	DBMSService
C.13	Firebase
C.14	Maps Service

	C.1	C.2	C.3	C.4	C.5	C.6	C.7	C.8	C.9	C.10	C.11	C.12	C.13	C.14
R.1	X	X												
R.2	X	X	X	X		X	X			X	X	X		
R.3	X	X	X	X			X			X	X	X		
R.4	X		X	X		X				X	X	X		
R.5	X		X	X		X	X			X	X	X	X	X
R.6		X	X	X			X			X	X	X		
R.7	X		X	X				X		X	X	X		
R.8	X		X	X				X		X	X	X		
R.9	X	X	X	X		X	X			X	X	X		
R.10	X		X	X	X					X	X	X		
R.11		X	X	X			X			X	X	X		
R.12	X		X	X				X		X	X	X	X	
R.13	X		X	X				X		X	X	X		
R.14	X		X	X		X		X		X	X	X		
R.15	X	X												
R.16	X	X	X	X	X				X	X	X	X		
R.17	X	X	X	X	X					X	X	X		
R.18	X		X	X	X					X	X	X		
R.19	X	X	X	X		X	X			X	X	X		
R.20	X		X	X			X			X	X	X		
R.21	X		X	X	X					X	X	X		
R.22		X	X	X		X	X			X	X	X		

G.1	Allows to regulate the influx of people that enter the building.
R.1	The system generates a single QR code to enter and exit the store for each booking or lining up.
R.2	The system allows to get a reservation for the supermarket.
R.6	The system allows people (who do not have access to the required technology) to digitally line up directly when they are at the store.
R.17	The system shows active reservations.
R.19	The system allows users to delete a reservation.
R.20	The system allows customers only one lining up at a time for each shop.
R.22	The system allows the store manager to scan the QR codes.
C.1	CustomerMobileApp
C.2	StoreManagerMobileApp
C.3	Routing
C.4	Middleware
C.5	UserController
C.6	BookingController
C.7	LineupController
C.8	ShopController
C.10	Query Builder
C.11	Model
C.12	DBMSService

G.2	Avoids that customers must line up and wait outside of stores for hours.
R.1	The system generates a single QR code to enter and exit the store for each booking or lining up.
R.2	The system allows to get a reservation for the supermarket.
R.3	The system provides customers a precise estimation of the waiting time.

R.5	The system alerts the customers before their shift according to the geolocation information.
R.6	The system allows people (who do not have access to the required technology) to digitally line up directly when they are at the store.
R.9	The system allows customers to insert the approximated expected duration of the visit.
R.10	The system infers customers' expected duration of the visit based on an analysis of the previous visits.
R.16	The system requires a sign up/login.
R.21	The system uses information about the customer that exit the store to infer better the waiting time.
R.22	The system allows the store manager to scan the QR codes.
C.1	CustomerMobileApp
C.2	StoreManagerMobileApp
C.3	Routing
C.4	Middleware
C.5	UserController
C.6	BookingController
C.7	LineupController
C.9	ManagerController
C.10	Query Builder
C.11	Model
C.12	DBMSService
C.13	Firebase
C.14	Maps Service

G.3	Guarantees that everyone can shop, even people who do not have access to the required technology.
R.4	The system allows customers only one booking at a time for each shop.

R.6	The system allows people (who do not have access to the required technology) to digitally line up directly when they are at the store.
R.7	The system suggests alternative time slots for visiting the store when the desired one is not available.
R.8	The system suggests alternative stores when the desired one is not available.
R.14	The system shows the available time slots for each grocery.
R.15	The system provides a QR code printing service.
R.16	The system requires a sign up/login.
R.22	The system allows the store manager to scan the QR codes.
C.1	CustomerMobileApp
C.2	StoreManagerMobileApp
C.3	Routing
C.4	Middleware
C.5	UserController
C.6	BookingController
C.7	LineupController
C.8	ShopController
C.9	ManagerController
C.10	Query Builder
C.11	Model
C.12	DBMSService

G.4	Provides smart managing of lining up and booking with a digital system.
R.1	The system generates a single QR code to enter and exit the store for each booking or lining up.
R.2	The system allows to get a reservation for the supermarket.
R.6	The system allows people (who do not have access to the required technology) to digitally line up directly when they are at the store.

R.7	The system suggests alternative time slots for visiting the store when the desired one is not available.
R.8	The system suggests alternative stores when the desired one is not available.
R.9	The system allows customers to insert the approximated expected duration of the visit.
R.10	The system infers customers' expected duration of the visit based on an analysis of the previous visits.
R.11	The system allows store managers to get a lining up only for own store.
R.12	The system provides periodic notifications of available time slots in a day/time range.
R.13	The system shows the list of shops.
R.15	The system provides with a QR code printing service.
R.18	The system shows the history of reservation of each customer.
C.1	CustomerMobileApp
C.2	StoreManagerMobileApp
C.3	Routing
C.4	Middleware
C.5	UserController
C.6	BookingController
C.7	LineupController
C.8	ShopController
C.10	Query Builder
C.11	Mode
C.12	DBMSService
C.13	Firebase

5 IMPLEMENTATION, INTEGRATION, AND TEST PLAN

In this section, it is proposed an overview of the plain that will be followed during the implementation of the system. Moreover, it is explained the process of integration of the already described components and their testing phases.

5.1 Implementation

The chosen strategy for the implementation (and testing) of the system is of thread type. This is an approach based on the concept of thread, which is composed of portions of different modules that together provide specific functionalities. Considering this definition, the thread approach focuses its attention first on the integration of one thread, then on another thread, and so on.

Please notice that the QueryBuilder is a component that will not be used in this phase of the development. Anyway, it is integrated into Laravel and it will be useful for future development. For these reasons, it is present in the analysis of the previous chapters but it is not planned in the implementation plan below. Also, the StoreManagerMobileApp, which will perform the same operations as the CustomerMobileApp, but adapted for the store managers, is not expected during the early development phase. It is left as a future improvement. Similar reasoning for the ManagerController which will allow the store manager to log in to the system.

The order of implementation will be the following:

1. **Database**: includes the creation of the needed table and the configuration of the DBMS. As already mentioned, the chosen DBMS is MySQL. It is the basic component from which all the others depend, so it will be configurated from the very beginning in order to guarantee the correct integration and testing with the other components.
2. **Model**: in this phase, all the Models that will map a table will be implemented. The first one will be the UserModel that will represent the user table. As the database, also the models are fundamental for the next components. On the other hand, they depend on the database, so they will be developed immediately after the database.
3. **External Components**: they are Firebase and the service for the maps. Firebase will handle the system of push notifications used to alert the customer when her/his turn is near to being called. The map service will manage the current position of the user and infer the time required to arrive at the shop. This phase is to intend as configuration only. The actual integration will be performed during the development of the Controllers, the CustomerMobileApp, and the StoreManagerMobileApp.
4. **UserController**: the first controller to be developed will be the UserController. It defines the method for the customer login that checks the inserted email and password and, if they are correct, creates a unique token. It will include also the method to retrieve the active reservations associated with the customer.
5. **Middleware**: configuration of the Middlewares provided by Laravel. It will include checking on the validity of the token that is necessary for the authorization of the requests. A requests filter will also be configurated in this phase. Due to its importance and centrality in the integration, implementation, and testing of the various features, it will be configurated in this phase.

6. **Routing**: it will define the routes to obtain the required functionalities and satisfy the requests.
7. **CustomerMobileApp**: it will present the data in a readable form for the customer. It will provide the access to the main functionalities described in this document and the RASD. With the StoreManagerMobileApp it is at the highest level component, hence its development will be continued in parallel with the following one.
8. **ShopController**: it will define the shop requests and allow to query the list of available shops.
9. **LineupController**: it will manage the creation, updating, and deleting of lining ups. From a conceptual point of view, it will be linked to the UserController and the ShopController and develop after them. However, during the development, it should be kept in mind that a high decoupling between the different components is preferred and required.
10. **BookingController**: it will provide the same functionalities offered by the LineupController, but for the bookings. In other words, it will manage the creation, updating, and deleting of bookings.

5.2 Integration

In this paragraph, more details about the order of the integration of the components are provided. This order should be coherent with the implementation order previously defined and follow the thread approach.

The descriptions are accompanied by some diagrams for clarity.

First Phase

During this stage, the basic portions of several components will be developed to permit the first tests. More in detail, the involved components will be the Database, the Model, the External Components, the Middleware, the Routing, and the CustomerMobileApp. Moreover, the only controller developed at this stage will be the User Controller, while the others will be delayed until the following phases.

Second Phase

During the second phase, it will be possible to develop the second Controller, i.e. the ShopController. Moreover, the components already developed will be updated to allow the correct integration.

Third Phase

At this point, the LineupController and the correlated features can be developed, integrated, and tested.

Fourth Phase

Lastly, it will be possible to integrate and test the BookingController.

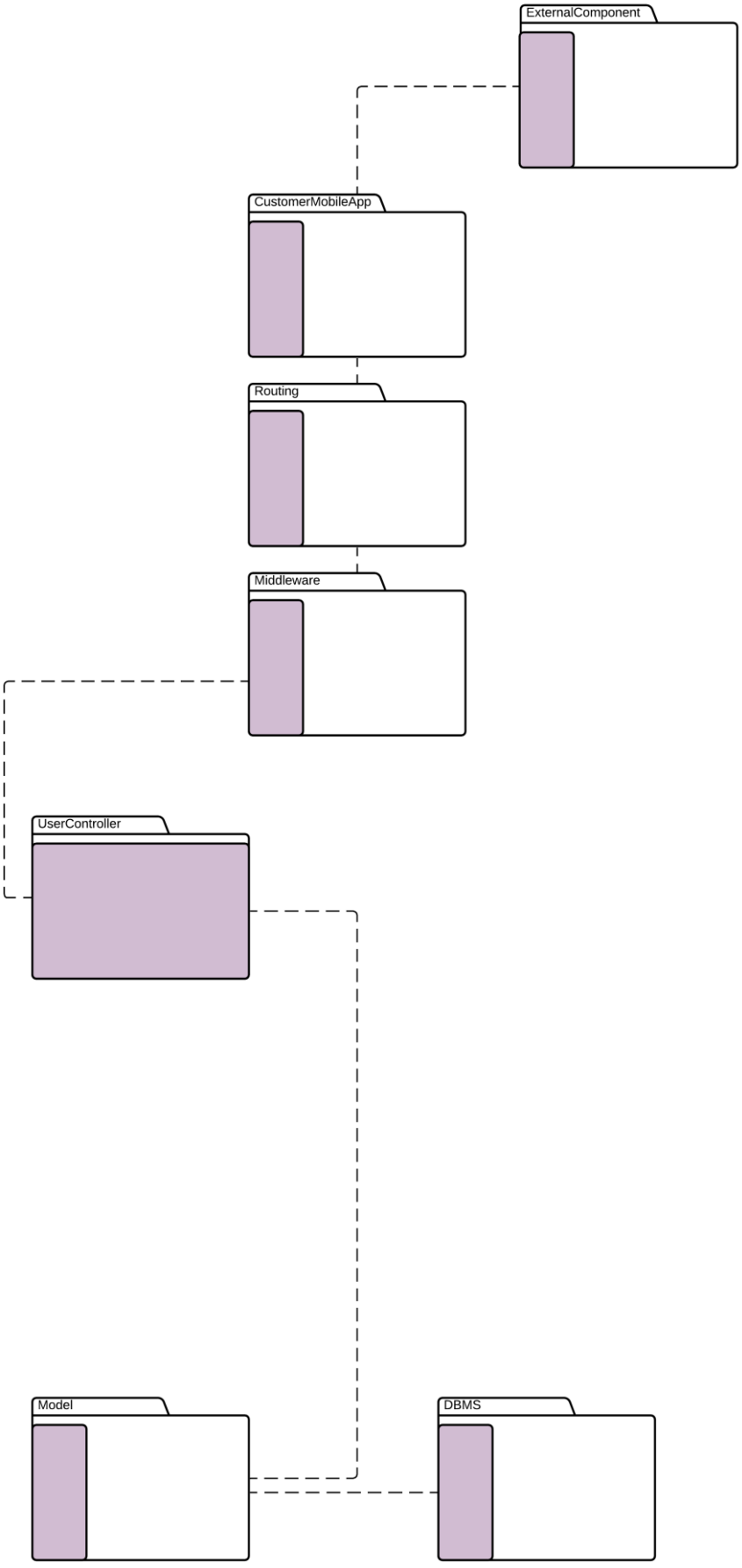


Figure 24: First Phase

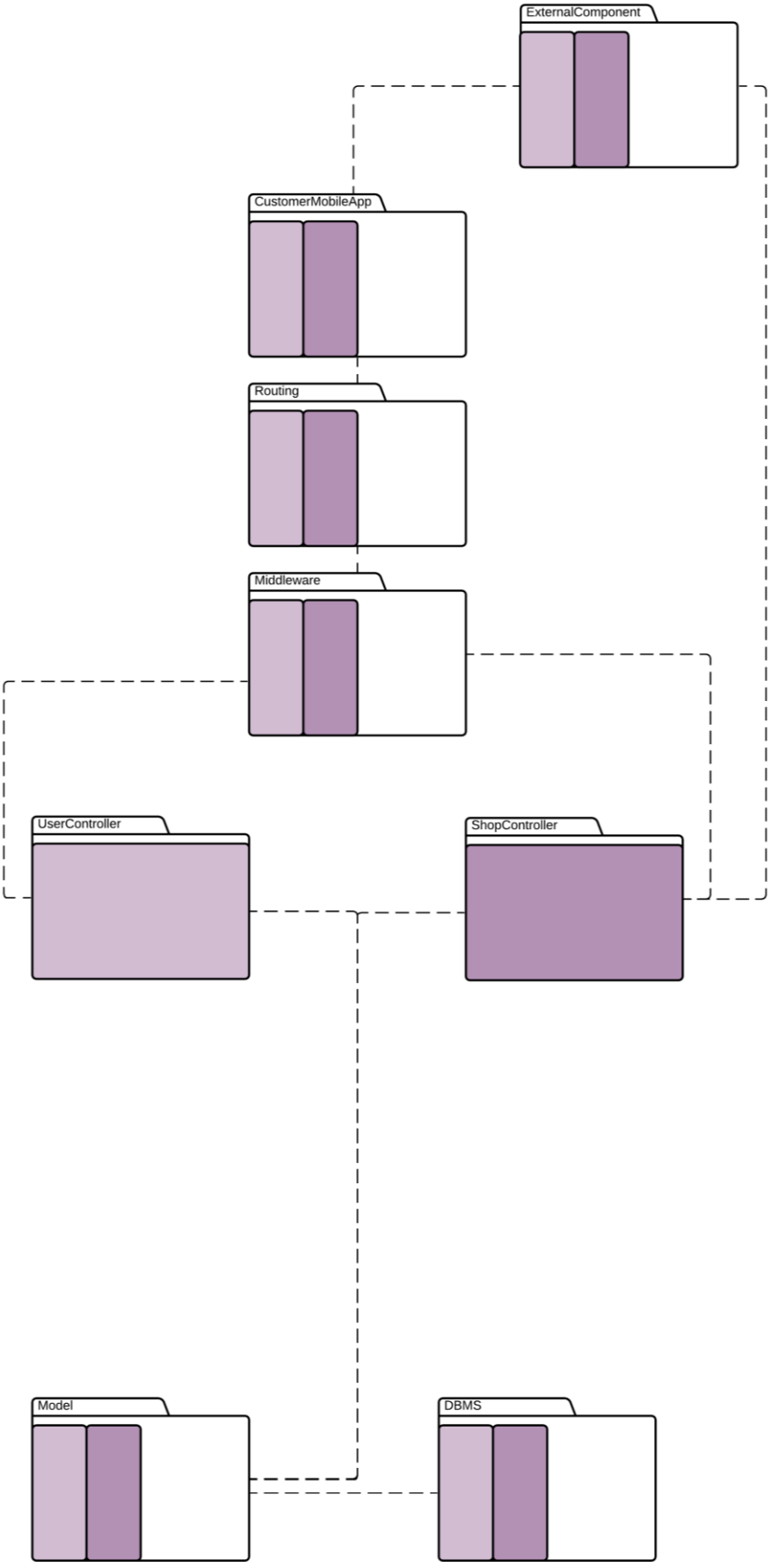


Figure 25: Second Phase

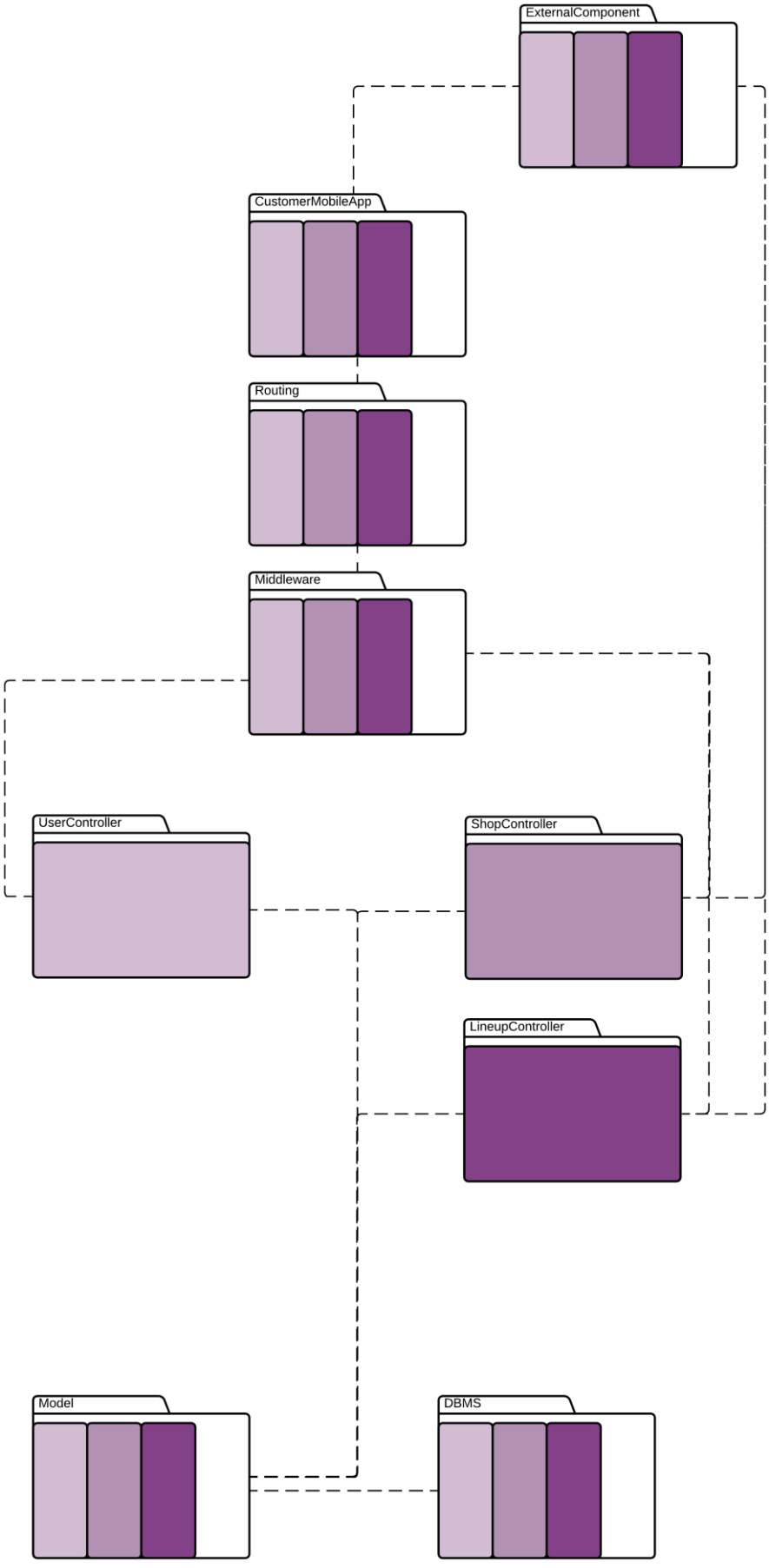


Figure 26: Third Phase

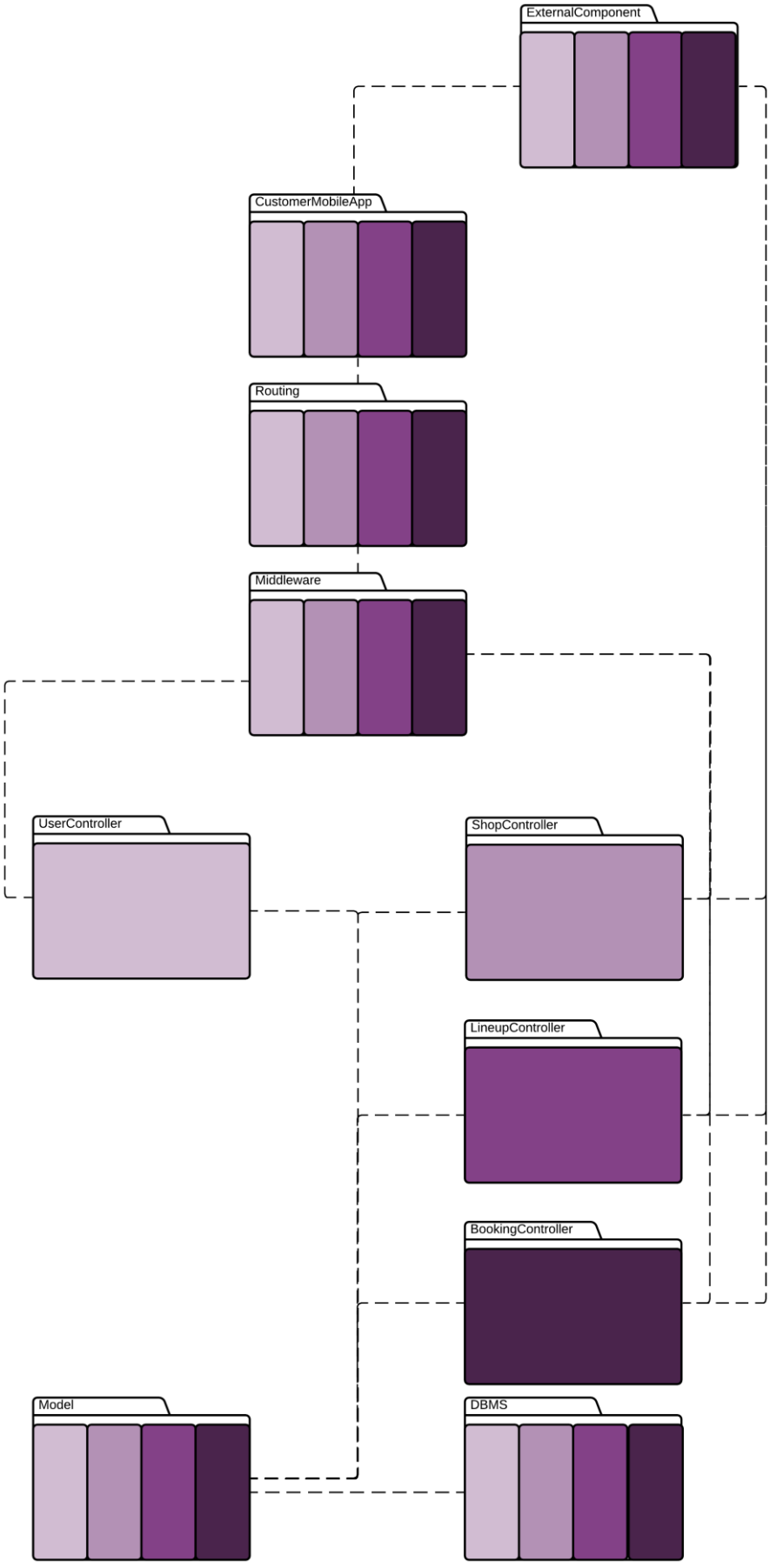


Figure 27: Fourth Phase

5.3 Testing

The focus will mainly on server-side integration tests following the order that has been defined in the previous section. In other words, it will occur while the various components will be developed and released.

As soon as a version of a component will be developed and integrated with the others already present in the system, the integration will be tested. During this operation, another component should be developed. If the test will be passed, the new components can be integrated and tested as well.

Moreover, a form of system testing will be also conducted. This allows testing the complete integrated system.

In particular, it will be provided functional and performance testing. The former analyses the outputs provided by the system, given a series of known inputs. The inner structure of the system is not considered. The latter identifies possible bottlenecks that might affect the system and influence some critical aspects like response time. It allows identifying hardware and network issues, inefficient algorithms, and so on.

6 EFFORT SPENT

This section shows the amount of time that each member has spent to produce the document. Please notice that each section, diagram, and specification is the result of coordinated work. The column *Member* specifies only the main contributor (or contributors, if more than one) for each topic but should not be interpreted as a lack of participation by other team members for that topic.

TOPIC	MEMBER	HOURS
New mock-ups	Digregorio	3h
Creation of the document and integration of mock-ups	Digregorio	5h
Component definition and sketch of Component Diagram	Digregorio, Massaro, Tamma	4h
Component Diagram improvements	Tamma	3.5h
Research on architectures and patterns	Massaro	2h
Component Diagram integration and textual description	Massaro	2h
Architectural Design: Overview	Massaro	2h
Sequence Diagram definition and sketch	Massaro, Tamma	5h
Sequence Diagrams	Tamma	13h
Requirement traceability	Digregorio, Massaro	2h
Sequence Diagrams integration and comments	Digregorio	6.5h
General brainstorming and further decisions	Digregorio, Massaro, Tamma	3h
Algorithms: pseudocodes and flowcharts	Digregorio, Massaro	3.5h
Architectural Styles and Patterns	Massaro	2.5h
Implementation, Integration, and Test Plan	Digregorio, Massaro	4.5h
Component Interfaces sketch and definition	Massaro	3h
Component Interfaces Diagram	Tamma	4h
Correction and integration on *Chapter 2*	Digregorio, Massaro	1.5h
Final revision and improvements	Digregorio, Massaro, Tamma	6h

7 REFERENCES

- The diagrams have been made with: https://www.visual-paradigm.com/ and https://lucid.co/it
- The mockups have been made with: Adobe XD
- Sequence Diagram Reference: https://www.uml-diagrams.org/sequence-diagrams-reference.html
- *UML - Imparare a descrivere sistemi orientati agli oggetti graficamente e in modo standard*, APOGEO, Enrico Amedeo
- Wikipedia - https://en.wikipedia.org/wiki/Main_Page
- IBM – Three-Tier Architecture - https://www.ibm.com/cloud/learn/three-tier-architecture
- Laravel - https://laravel.com/