

**CLup project Roberto Buratti, Hrvoje
Hrvoj, Ozan Incesulu**



POLITECNICO
MILANO 1863

Design Document

Deliverable:	DD
Title:	Design Document
Authors:	Roberto Buratti, Hrvoje Hrvoj, Ozan Incesulu
Version:	1.0
Date:	1-1-2021
Download page:	https://github.com/Furcanzo/BurattiIncesuluHrvoj
Copyright:	Copyright © 2021, Roberto Buratti, Hrvoje Hrvoj, Ozan Incesulu – All rights reserved

Contents

Table of Contents	3
List of Figures	5
List of Tables	5
1 Introduction	6
1.1 Purpose	6
1.1.1 Description of the Proposed System	6
1.1.2 Goals	6
1.2 Scope	6
1.3 Scope	6
1.3.1 Targeted Users	7
1.4 Definitions, Acronyms, Abbreviations	8
1.4.1 Definitions	8
1.4.2 Acronyms	9
1.4.3 Abbreviations	9
1.5 Revision history	9
1.6 Reference Documents	9
1.7 Document Structure	9
2 Architectural Design	11
2.1 Overview	11
2.2 Component view	11
2.3 Deployment view	14
2.4 Runtime view	16
2.4.1 Register	16
2.4.2 Login	16
2.4.3 Update User Information	16
2.4.4 Book Future Line Number	16
2.4.5 Retrieve Line Number	17
2.4.6 View Store	17
2.4.7 Add Staff Member	18
2.4.8 Stop for Emergency	19
2.4.9 Update Store Information	20
2.4.10 Monitor Customers	20
2.4.11 Grant Access	21
2.4.12 Generate Guest Ticket	21
2.4.13 Create Store	21
2.5 Component interfaces	21
2.6 Selected architectural styles and patterns	24
2.6.1 Thick-Server	24
2.6.2 Logical Sharding	25
2.6.3 Lazy Loading	25
2.6.4 Service-Oriented Architecture	25
2.7 Other design decisions	25
2.7.1 ORM	25
2.7.2 ACID	25

3	User Interface Design	26
3.1	User Interface Diagram	26
4	Requirements Traceability	27
4.1	Functional Requirements	27
4.2	Non-functional Requirements	27
5	Requirements Traceability	28
6	Effort Spent	29
	References	30

List of Figures

1	Component Diagram for the overall system	11
2	Component Diagram for the Application Server	11
3	Component Diagram for the LineNumberManager	12
4	Component Diagram for the StoreManager	13
5	Component Diagram for the UserManager	14
6	Deployment view diagram	15
7	Sequence Diagram for Book Future Line Number	16
8	Sequence Diagram for Retrieve Line Number	17
9	Sequence Diagram for View store	17
10	Sequence Diagram for Add Staff Member	18
11	Sequence Diagram for Emergency Stop	19
12	Sequence Diagram for Update Store Information	20
13	Sequence Diagram for Monitor State	20
14	Sequence Diagram for Create Store	21
15	Component Interfaces Diagram	22
16	Class Diagram	24
17	USer interface diagram	26

List of Tables

1 Introduction

1.1 Purpose

1.1.1 Description of the Proposed System

The project "CLup - Customer Line up" is a line spot reservation system planned to be used by managers, clerks, and customers of many local vendors and chains. The system proposes a handy solution for the ongoing issue of proper social distancing management, particularly in grocery shopping, by providing assistance to cope with the customer load for managers and helping customers access stores in a safe and controlled manner.

In particular, users will be able to see the stores, get a line number, and book in advance for the grocery stores they would like to visit. Once assigned a line number, the customer will track the estimated time of arrival of the line and wait for the notification that informs about his or her line's forthcoming arrival; hence waiting time in the line in the crowd is minimum. Also, "CLup" provides uniquely generated QR codes per the line number, which can be utilized by the store managers as a proper monitoring tool in the entrances and exits of the locations. The product's general purpose is to keep the congestion levels in line with the stores at a minimum via providing useful features for all the users.

1.1.2 Goals

- G_1 Regulate the influx of people in the stores.
- G_2 Avoid people lining up in front of the stores in order to reduce the spreading of COVID-19.

1.2 Scope

1.3 Scope

The system is expected to implement the following features, which are detailed in the future sections of this document:

- **Generate Line Number:** The customers of the store can generate a line number for themselves as soon as possible to visit a store.
- **Schedule Line Number:** The customers can generate a line number for a future planned visit. Using the CLup app the customers can further define the product categories that they plan to visit to allow finer granularity and better time estimation for a visit. The system thus will allocate them a line number according to the time slot the customer has selected and it will send an email to the customer if the line number is canceled for a reason. From the system, the customers can view the store's location and plan their route accordingly. While scheduling their line numbers, the customers can see an occupancy forecast for the store during time slots so that they can further decide on visiting the store. If added by the manager, the customers can instead prefer to visit a partner store.
- **Print Line Number:** The clerks in the store can generate line number tickets for those customers who have not used the CLup app to generate a line number.
- **Manage Store:** The managers in the store can update the information regarding the store, which are the name and the location of the store. Furthermore, the managers can use this feature to activate or deactivate additional features provided by the system to manage customers more effectively. They can update the timeout interval for a ticket to expire, to manage the queue in front of the store. They can add product categories that their store provides, for the customers to select

before booking their visit as mentioned before. They can assign more managers and clerks if necessary to use the system to its full extent. They can administer the time slots for their customers by setting the opening and closing hours, with possible breaks in between. They can administer the partner stores of their stores if they exist, for their customers to select instead. They can enable the reservation limit to prevent any customer from abusing the system by setting the limit for how many tickets a customer can generate per a given time interval of a month, a week, or a day. The stores are to be created by the Super User, while granting access to the first manager.

The system is expected to be delivered in two parts, a client application, and a server. Given its functionality is depending on the current ongoing pandemic, the app is expected to be implemented before the COVID-19 crisis ends. For this, the client application is expected to be implemented as an app running on a browser, increasing the range of devices being able to run the app, without increasing the development costs, which can then be ported to a mobile application if needed. The server is expected to serve the client application to its users and handle any transaction incoming. To implement the location-specific features of the system, due to the complexity of those, namely location finding, addressing, and route planning tasks, an external Maps API is used.

The system is going to provide the following functions:

- F_1 Customers can issue a line number for a store.
- F_2 Customers can issue line numbers for their future visits.
- F_3 Customers can detail their visits by category.
- F_4 Customers can plan their visit to the store.
- F_5 Customers may prefer to use alternative time slots or partner stores for their visit.
- F_6 Managers can prevent customers from issuing line numbers.
- F_7 Managers can customize the system to allow optimizations for increased granularity, flow control, and time slot forecasting.
- F_8 Clerks and Managers can monitor the customers through their entrance and exits.
- F_9 Customers can obtain printed line number tickets.

1.3.1 Targeted Users

"CLup" aims to resolve the problem of Customers queueing up in front of a store, without control of availability of place in the store and future contact tracing by managers.

Customer:

Customers will be able to obtain specific line numbers for various stores using CLup, which they can track the estimated time available with and also view the location on a map application to plan their visit. Customers can further obtain line numbers for future visits, based on the system's info about the availability of free spots on the specific time intervals. They may prefer to visit a different branch of the same chain. Furthermore, they can provide specific product categories they intend to purchase or set an estimated time for their visit to allow finer granularity. Some customers may also prefer to obtain a line number upon visiting the store physically. To plan their visit in a time slot where the store will be less crowded, the users can see the store's occupancy based on already taken line numbers and forecasts provided by the system.

Clerk:

The clerk (which can be a shopping assistant or a security detail) can monitor customers' flow and manually intervene in case of missing line numbers via performing manual checkout for a specific customer or by printing line numbers physically for some customers.

Manager:

The store manager (which could be an actual manager or someone responsible for handling customer management) can provide details regarding the availability of products and the store in general, by setting the opening hours, maximum allowed customers in the shop, in-shop location of different product categories, the maximum amount of reservations that can be made per customer, line number timeout and the location. Also, for chains and relevant stores, the manager can add chain members for the store.

Super User:

In order to use the CLup application, store or business owners should contact CLup Super User who will give them permission to use the application and create a Manager account for them.

1.4 Definitions, Acronyms, Abbreviations

1.4.1 Definitions

- *Store*: The physical location of the business that uses the line reservation system
- *Manager*: A person in charge of executive action within the store
- *Customer*: A person to visit the store
- *Clerk*: A person in charge of handling the entrance and exit of customers. This user can be replaced by a QR code totem or a similar device for customers to directly interact with instead.
- *Visit Time*: The time interval in which a customer performs a visit to the store
- *Super User*: A user assigned by the company hosting the system to grant manager access to certain users.
- *Line Number*: A number that indicates the ordering of a specific customer in the line
- *Line Number Ticket*: A physical ticket printed that features the line number and the QR code.
- *Time Slot*: Specific intervals of time in which a customer can schedule their visits determined by the opening hours.
- *Partner Store*: A different store included in the same beneficiary chain of command (such as another member of the franchise or store chain) or a mutual agreement with the specific store
- *Product*: Any item, items, service or services demanded by the customer, and provided by the store to the customer.
- *In-store Location*: A location of a specific product category inside the store.
- *Working hours*: The time intervals that the store is open during each day.
- *Maps API*: A third-party mapping service implementation used for location tracking.
- *SSO Provider*: A third-party service providing SSO authentication.
- *ID Token*: A token provided by the SSO provider to verify the user's identity from the server.

1.4.2 Acronyms

- **RASD:** Requirement Analysis and Specification Document
- **QR Code:** Quick Response Code
- **API:** Application Programming Interface
- **SMTP:** Simple Mail Transfer Protocol
- **HTTP:** Hyper-Text Transfer Protocol
- **REST:** REpresentational State Transfer
- **JDBC:** Java DataBase Connectivity
- **DBMS:** DataBase Management System
- **JWT:** JSON Web Token
- **ACID:** Atomicity, Consistency, Isolation, Durability
- **ORM:** Object-Relational Mapping

1.4.3 Abbreviations

- G_n : n^{th} goal
- D_n : n^{th} domain assumption
- R_n : n^{th} functional requirement
- F_n : n^{th} function

1.5 Revision history

1.6 Reference Documents

- **Specification Document: R&DD Assignment AY 2020-2021**

1.7 Document Structure

This document is composed of six sections, each with the purpose described below:

- **Introduction:** This section provides an introduction of the problem, the scope of the project with details regarding the goals, target users and phenomena. The goals of the project are formulated in accordance with the description of actions and actors in the Design Document. Within the project's scope, properties and duties of different users and user groups are described in Target Users section. Furthermore, under the scope of the project, the relevant phenomena of the project is presented through their relevance to the world and the machine.
- **Architectural Design:**
- **User Interface Design:**
- **Requirements Traceability:**
- **Implementation, Integration and Test Plan:**

- **Effort Spent:** This section features the effort table, in which all team members provide a rough estimation on the time spent on creation of the various sections of the document.
- **References:** This section features different reference materials referred inside this document.

2 Architectural Design

2.1 Overview

2.2 Component view

In the following section, components used to implement different functionalities of the system is described, aided with component diagrams demonstrating their separation and interactions within each other and with other external interfaces.

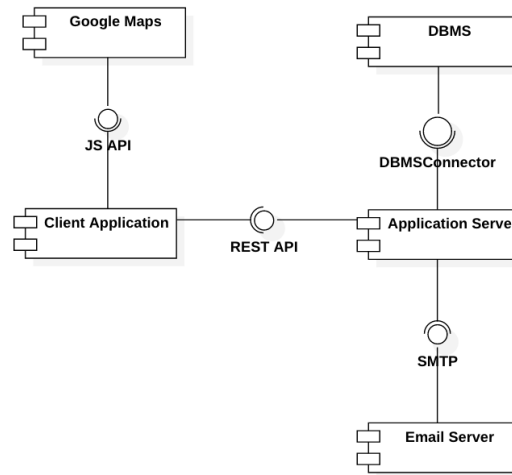


Figure 1: Component Diagram for the overall system

Component Diagram for the overall system provides an overall view to the components present in the system and the connections between these components to correctly realize the decisions provided in this document. The external integrations of the system, with the general communication interfaces they communicate with other components are provided, however the details for the Application Server will be presented following, only, considering that the main application logic is executed through the components residing in it. The client is a thin-client built to interact with and display information directly sent through the endpoints. Therefore, it is provided as one component that is unnecessary to split into sub-components.

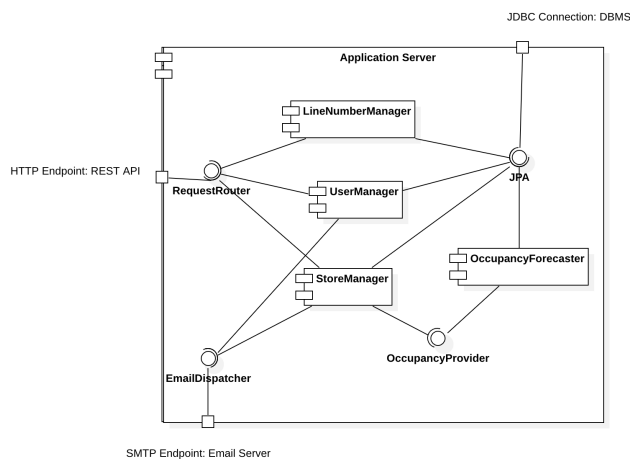


Figure 2: Component Diagram for the Application Server

Component Diagram for the Application Server provides an overall view for the components inside the Application Server. The components separate the functionality into three domains, namely Line Numbers, Users and Stores. All other entities that exist in the system are included into one of these domains based on their relevance. All the user facing functionalities of the system are exposed through the REST endpoint, which uses a router interface to route each request to the domain component it belongs to. All domain components use the ORM to persist their domain data structures, and components that require to send e-mails use the SMTP endpoint to do so. The OccupancyForecaster is a component that features only one function: periodically reading the database for entry and exit records of customers and updating the store accordingly. Therefore, it is not split further into components.

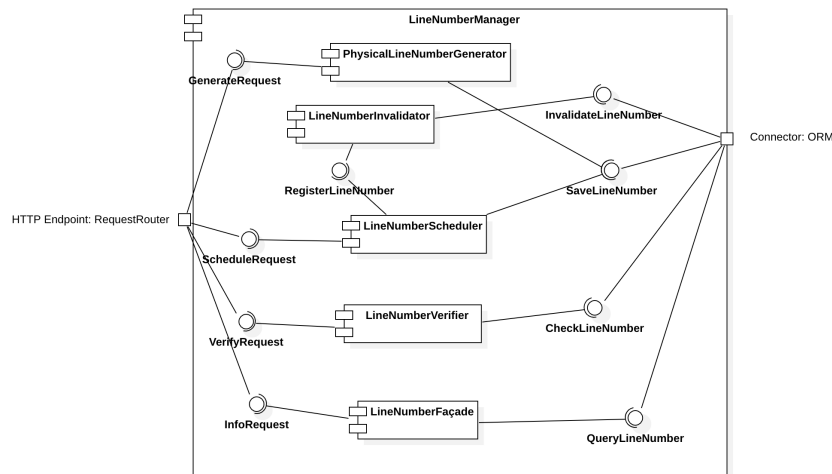


Figure 3: Component Diagram for the LineNumberManager

Component Diagram for the LineNumberManager provides a detailed view over the domain component for line numbers. It allows the management of any sort of query related to line numbers by clerks and customers. This component is further divided into following sub-components to increase the granularity of actions performed on line numbers:

- **PhysicalLineNumberGenerator:** This component exists to provide an interface for clerks to generate line numbers, it's interface handles the incoming request by generating and persisting a new ticket for the to-be-printed line number.
- **LineNumberScheduler:** This component exists to allow the users of the system to book a line number from their home. It realizes its functionality through connecting to the same interface of the persistence provider, however it is capable of handling more complex requests, including custom product category selection and time slot allocation. Furthermore, it registers the ticket to the LineNumberInvalidator to be invalidated after the set timeout minutes have passed from the time slot.
- **LineNumberInvalidator:** This component acts as a helper component to LineNumberScheduler. It schedules the invalidation of the scheduled tickets, so that the invalidation can occur asynchronously. This component is created to separate this responsibility from an active user facing component, all which have the main responsibility to provide a synchronous response to all the users' needs.
- **LineNumberVerifier:** This component is used to handle the transactions regarding customer QR code verification conducted by the Clerk. It is used to register the entrance and exit of customers with their QR codes.

- **LineNumberFaçade:** This component is used by the customers that want to query detailed information regarding the line numbers that they have. The requests handled by this endpoint is directly mapped to the client application's needs.

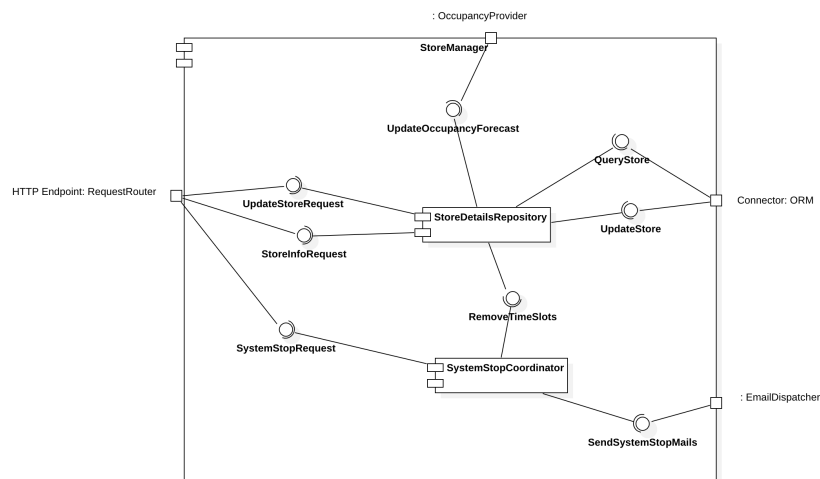


Figure 4: Component Diagram for the StoreManager

Component Diagram for the StoreManager provides a detailed view over the domain component for handling requests related to the store. The wrapping component allows querying all relevant information about the store by all, and furthermore, harbors the logic for the manager users to update different aspects related to the store, such as basic information, time slots, configuration and system stop scheduling. This component, apart from having mailing, routing and database interfaces, exposes an interface to the OccupancyProvider to retrieve updated information about the occupancy forecast. This component is divided into following components to decrease cohesion between tasks to be performed:

- **StoreDetailsRepository:** This component is responsible for carrying out any direct query to the store data and all of the included classes, which are products, categories and time slots. There are two exported interfaces available to be used via HTTP. The query interface allows all the system users to view relevant information for the store, while the manager has access to the other interface allowing them to update the information. The exposed interface to the OccupancyProvider allows the OccupancyForecaster to store updated information regarding the future predictions for the store.
- **SystemStopCoordinator:** This component is responsible for carrying out all the actions that are necessary to perform or schedule a system stop, that are removing the specific time slot and sending e-mails to customers who has already booked those time slots.

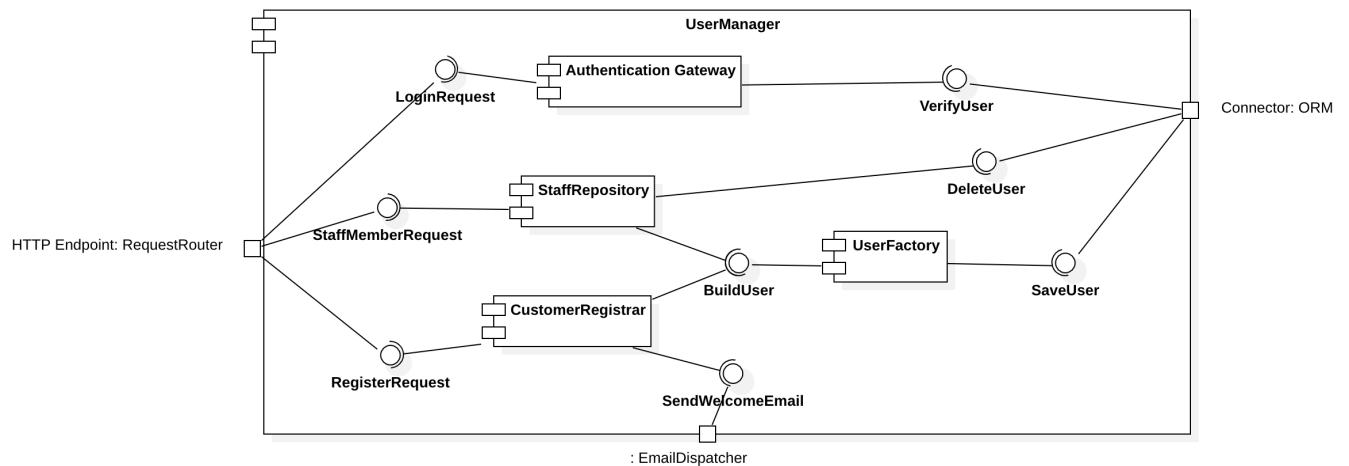


Figure 5: Component Diagram for the UserManager

Component Diagram for the UserManager provides a detailed view of the sub-components related to requests that can be performed on users. This component not only acts as a user registry, but also provides the authentication interfaces required, since its logic encompasses handling user data and authentication is done via user's email address, which relates these functions. To separate responsibility on different actions to be performed on the system, this component is divided into specific sub-components, that are:

- **AuthenticationGateway:** This component acts as a medium to authenticate the user and generate the necessary authorization tokens for future use.
- **StaffRepository:** This component handles all the requests related to adding and removing staff members to stores. Since, in our design, different flavors of users are implemented using the same basis, the creation and removal of manager and clerks are similar from the architectural point of view. Furthermore, since the implementation of user creation is similar to that of the customers, a common factory component is provided to prevent duplication.
- **CustomerRegistrar:** This component handles the requests specific to registering new customers into the system. The component is responsible for calling the shared UserFactory to create a customer user and send the customer a welcome email.
- **UserFactory:** This component acts as a common interface and as a factory to create users. It is used by the components mentioned above to introduce new users to the system.

2.3 Deployment view

The following diagram illustrates the physical architecture and the deployment of the system. Each node represents a piece of hardware which harbours one or more software units. Each piece of hardware could be replicated multiple times to improve performances as specified in later in this document.

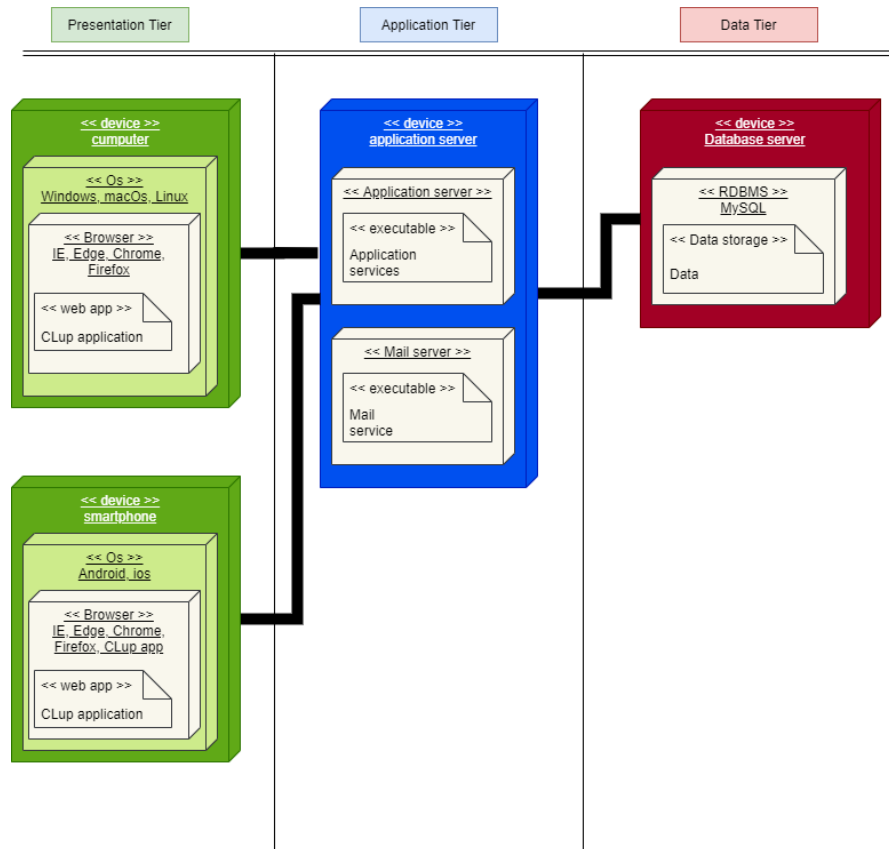


Figure 6: Deployment view diagram

As shown by the diagram, this system is deployed in a three-tier architecture, where the three tiers consist of:

- Presentation tier is the highest-level tier and harbours the presentation logic. The application is thought to be a web app, so it's accessible from every device with a browser, but it's probable that a smartphone should be the most used device to access the application so there will be android and ios applications to access the web app. The smartphones applications will be more similar to a browser than to a real application, therefore in the diagram is represented as a browser. In this tier there will not be application logic, so the architecture should be thin-client.
- Application tier is the core level from the standpoint of logical management of the application services. This tier contains all the application logic, from the scheduling of the line numbers to the management of the store. This tier is connected to the presentation tier through APIs. An API is used for each action of the user, and the APIs are divided based on the role of the user. Each user needs to authenticate using an API and receiving a jwt that needs to be attached to every subsequent request from the same user. The jwt attached to the request is used to determine which API are exposed to that user. This tier contains also a mail Server to notify the users in case their line number is cancelled.

- Data tier (tier 1 in the diagram) is the lowest-level tier as it contains all the information required to fuel the application services, most prominently the query manager. The databases are thought to be managed through an RDBMS (Relational Database Management System), in particular MySQL. Relational databases have the advantage of having near-maximum information density. Relational constraints can improve the quality and integrity of the content of the database.

2.4 Runtime view

2.4.1 Register

2.4.2 Login

2.4.3 Update User Information

2.4.4 Book Future Line Number

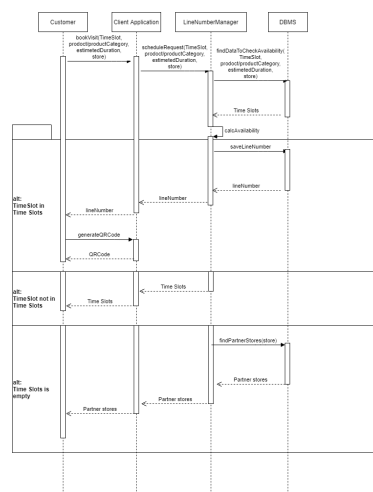


Figure 7: Sequence Diagram for Book Future Line Number

Sequence Diagram for Book Future Line Number represent the flow of events needed to book a line number. When the customer clicks on the "Book a Line Number" button they need to provide the TimeSlot in which they want to book, the product category that they need to buy, an estimated duration of their visit and the store they want to visit. After that the Client application contacts the LineNumberManager that retrieve from the Database the information needed to calculate which timeslots are available in that store, for that product category. If the timeslot that the customer put in their request is available the linenummer is generated and returned to the customer. Otherwise if the timeslot that the customer requested is not available, but there are others timeslots available in that store for that product categories, the system returns the timeslots available to the customer as an advice. If that store has no timeslots available for that product categories, then a list of partner stores is returned to the user as an advice.

2.4.5 Retrieve Line Number

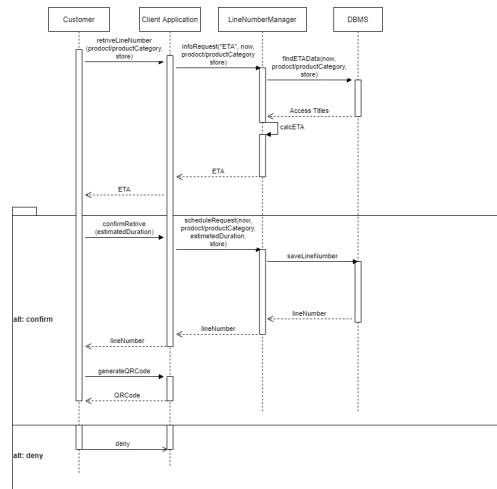


Figure 8: Sequence Diagram for Retrieve Line Number

Sequence Diagram for Retrieve Line Number represent how the system acts when a customer try to retrieve a lineNumber. First of all the customer inserts the request in the client application, providing the store they want to visit and the product category of the products they want to buy. Then the Client application contacts the LineNumberManager in order to have an ETA for that store. The LineNumberManager retrieve the information needed to calculate the ETA, calculate it and returns it tio the customer. The customer now can choose if they want to confirm and actually retrieve the lineNumber, or deny and stops the operations. In case the customer confirms a second request is sent to the LineNumberManager that generate the lineNumber and stores it in the database. Notice that the request of the ETA and the request of generating a new lineNumber are completely independent, this mean that the ETA can increase in the meantime, if someone else retrieve a lineNumber. This is not a problem under the assumption that the time between the two request is short and makes the implementation easier and avoids the handling of a lot of corner cases, such as if the application crashes between the two requests.

2.4.6 View Store

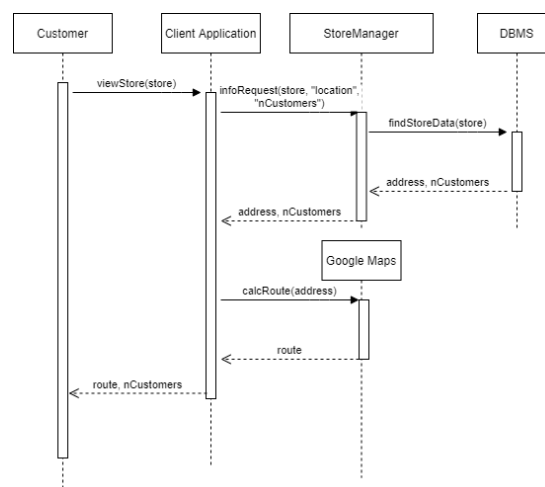


Figure 9: Sequence Diagram for View store

Sequence Diagram for View store shows how the customer can retrieve information about a store. The

customer request to the client application the information of a store, including the number of customers actually in the store. The client application forwards the request to the StoreManager that retrieve the information from the database and returns it to the application. The client application before returning the information to the customer contacts Google Map to calculate the route to the store.

2.4.7 Add Staff Member

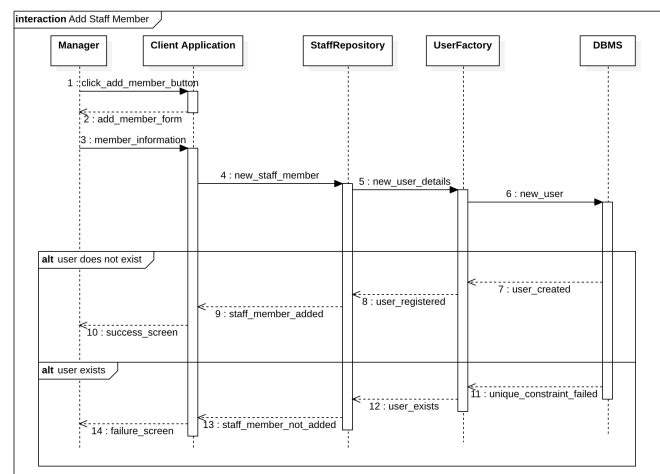


Figure 10: Sequence Diagram for Add Staff Member

Sequence Diagram for Add Staff Member represents the event flow taking place during the process of adding a new staff member to the store by the manager. After the manager clicks the "Add Member" button on the application, the application displays a new form featuring the necessary information to create a new staff member user. When the manager fills in the relevant information and submits the form, the provided information is transferred from the CLup Application to the StaffRepository to be processed. It shall be noted that the verification of the information (such as valid e-mail address and valid password) is performed within the application, which is not represented in this graph for simplicity. Upon the retrieval of the information related to the new member to be added, the StaffRepository calls the UserFactory with the information, to create the new user. The UserFactory then creates the actual user through the DBMS, while applying necessary security measures to the user data, like password hashing. The DBMS might persist the data given, but it might also fail to do so with a unique constraint failure error, if the provided user credentials is already used by another user. In case the user didn't exist in the system before the transaction, DBMS emits a message indicating that the user is created, and that message is propagated through the call stack, finally rendering a success screen on the manager's interface. However, if the user credentials were already used for another user, the error message emitted by the DBMS gets translated into a simpler form and propagated through the components, with a failure screen shown to the manager.

2.4.8 Stop for Emergency

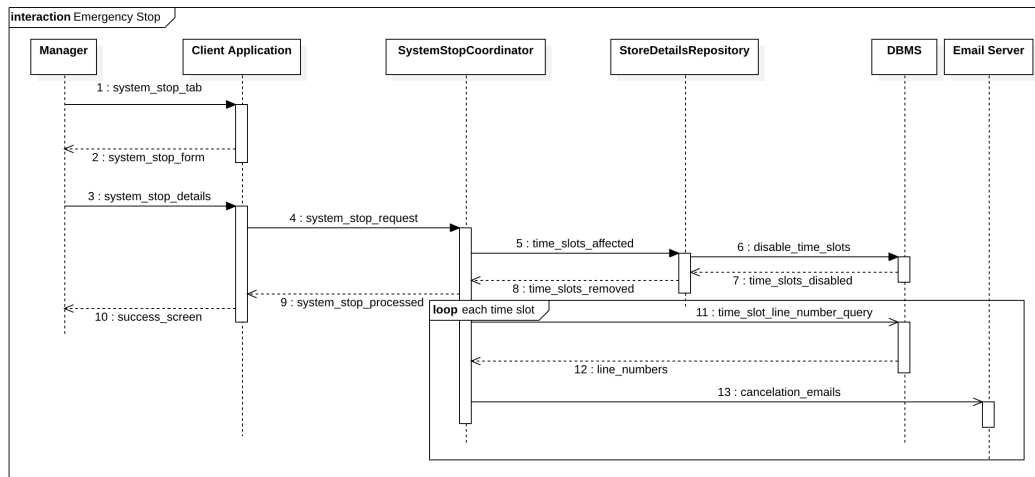


Figure 11: Sequence Diagram for Emergency Stop

Sequence Diagram for Emergency Stop demonstrates the flow of communication between different components of the system to create an emergency system stop, that can be issued immediately or scheduled to a future day, by the manager. When the manager wants to open the "System Stop" tab, the application renders a form providing input fields for details regarding the system stop, which is about whether it is immediate or when it shall be performed. After the manager providing the details of the stop, the application performs a request to the SystemStopCoordinator, the component responsible for handling system stop actions, which includes all the details provided by the manager. The coordinator first initiates the transaction for disabling all the time slots affected by the specific request from the store by sending message to StoreDetailsCoordinator, which in turn sends the disabling query for all the timeslots of the specific store to the DBMS. The DBMS then returns a message with the list of disabled time slots, which then the StoreDetailsRepository propagates back to the SystemStopCoordinator. It shall be noted that, due to various reasons, the actual disabled time slots might be less than the affected time slots, since it can be the case that some of the slots were already disabled before this transaction. After the disabling of the time slots an action receipt is returned to the CLup application indicating completion of the process, which in turn renders a success screen to the manager. Meanwhile, the SystemStopCoordinator also starts sending email notifications for each time slot that has been removed from the store. For each slot, a query to the DBMS is sent to determine the users that have already booked like numbers for the specific slot. When the response from DBMS is retrieved the SystemStopCoordinator starts sending emails to all those users, through the Email Server component.

2.4.9 Update Store Information

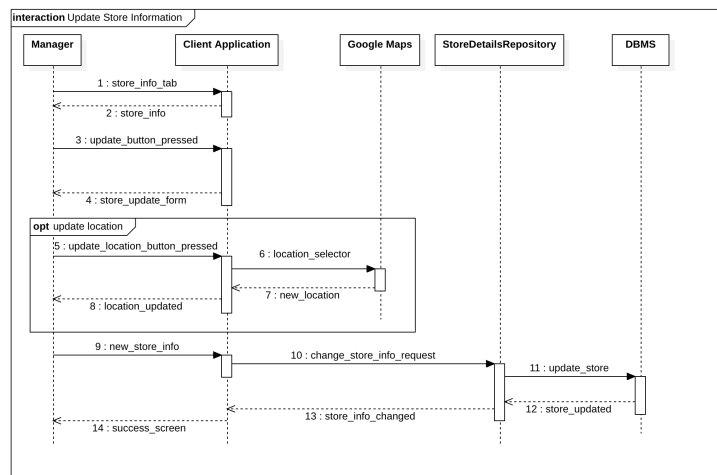


Figure 12: Sequence Diagram for Update Store Information

Sequence Diagram for Update Store Information shows the interactions between the components of the related parts of the system to update store information by the manager. When the manager requests to see the information for the store, the client application displays the related information with an update button. Upon trigger of the button, the CLup Application shows the editing form for the general information about the store, with the current values already pre-filled. Optionally, from this moment, if the manager wants to update the store's location, they can do so by pressing the update location button, which in turn, triggers the Google Maps component to show the location selector. Once the manager sets the new location, the service returns the new location selected by the manager, to be updated on the form. After all the necessary changes are made, the manager submits the form which makes the CLup Application perform a request to the StoreDetailsRepository. The StoreDetailsRepository, after processing the data, calls the DBMS to update the database entry for the store. Once the updated message is retrieved from the DBMS, the message propagates through the calls and a success screen is shown to the manager.

2.4.10 Monitor Customers

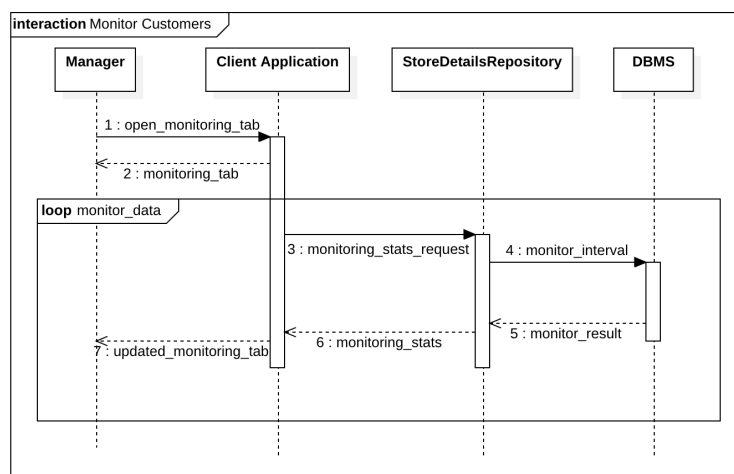


Figure 13: Sequence Diagram for Monitor State

Sequence Diagram for Monitor State displays the interactions between the components of the system required to enable general monitoring of the amount of customers in the store by the manager. When the manager wants to see the monitoring results for the store, the client application shows the store monitoring tab to the manager. Meanwhile, the client application requests the related monitoring statistics from the server continuously, for which the request is handled by the StoreDetailsRepository. The repository queries the database through the monitor interval function, passing the interval requested by the client application. The database then returns the view for the monitoring result given the specific time interval, which in turn is passed to the client app. As the client app retrieves the data, it updates the monitoring tab that the manager sees with new information, and continues with the loop.

2.4.11 Grant Access

2.4.12 Generate Guest Ticket

2.4.13 Create Store

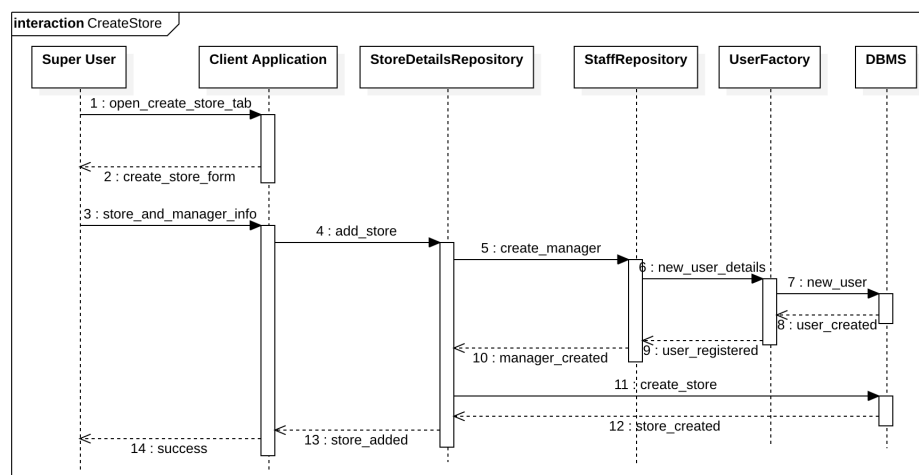


Figure 14: Sequence Diagram for Create Store

Sequence Diagram for Create Store displays the interactions needed to create a new store and its manager by the super user. When the super user requests the create store tab, the application returns the tab with the appropriate form for the information regarding the store and the manager of the store. After the submit, the client application calls the appropriate method in StoreDetailsRepository, which in turn yields the manager information to the method call in the UserRepository. The event flow from there is the same as **Sequence Diagram for Add Staff Member**. When the user is created, the StoreDetailsRepository creates the new store, given the newly created manager and the info provided by the super user. Finally, when the store is created, the update is propagated to the Client Application, which displays a success message to the user.

2.5 Component interfaces

In the following UML Class Diagram, the main interactions between components have been shown. The diagram's scope is mainly the application server and the peripheral services that the server communicates to function correctly. The front-end interfaces are not included as they only act as a mean of communicating the server functions to the end user, thus they do not include any specific functionality implemented. This diagram presents many details regarding the interactions between the system components, however this diagram does not provide a skeleton for the actual implementation, since its sole purpose is to demonstrate the significant integrations between components. Thus, it shall only be seen

as a guide to implement the actual system. In this diagram, all user facing interfaces' (LineNumberManager, StoreManager and UserManager) functions are assumed to retrieve the user calling as an implicit argument, thus authorization aspect is not reflected in this diagram for simplification. In this diagram, many entities in function inputs are represented as they are (like Store, User, TimeSlot, etc . . .), however for optimization purposes the actual implementation may use the specific IDs of each entity instead of passing the whole object. The DateTime used in this diagram is a general name for any implementation for a structure to store a specific time conforming to the standard []. In the diagram, representations for errors that might be returned from the operations are omitted (such as access right problems, network errors), for simplification purposes, along with specific errors (such as conflicts).

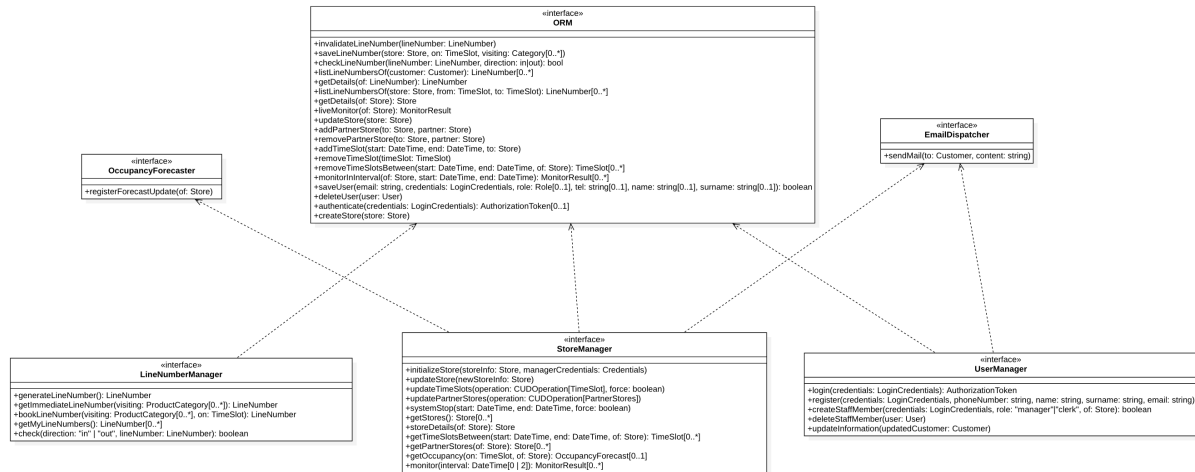


Figure 15: Component Interfaces Diagram

Details regarding the interfaces in **Component Interfaces Diagram** are:

- **ORM:**

This component represents the functionalities that are implemented in the ORM code that is used to communicate with the database. It should be noted that these operations represent the first abstraction layer of the underlying database towards the data model, thus they consist of the most primitive building blocks to realize the actions to be performed on the domain. The underlying realization of the SQL queries while executing these operations are also considered, and one-to-one mapping between the operations and queries are implemented in order to compensate for the atomicity of the calls made to this interface. The *getDetails* methods are implemented to provide an interface to resolve the lazy entities of *LineNumbers* and *Stores*, such as *productCategories* or *partnerStores*.

- **EmailDispatcher:**

This component represents the interface with the E-mail Client to send emails through SMTP. Since it has only one functionality regarding the scope of our system (sending emails to customers), the sole function of it is represented in this diagram.

- **OccupancyForecaster:**

This component updates the OccupancyForecasts of each store whenever is necessary. In order to decrease the overhead of constantly updating the forecasts, this component is to work asynchronously when the transactions on the system are minimal. The only operation exposed by this interface allows the stores to be registered for updates.

- **UserManager:**

This component is responsible for handling all transactions regarding the system's users, including

authentication. The object `LoginCredentials` is implemented in order to provide an abstraction over the authentication method to allow flexibility to swap the method for different markets and times. The method `login` returns an `AuthenticationToken`, which is also unspecified to allow flexibility for different methods, such as Bearer tokens or JWTs. While the customers are created using the `register` operation, managers and clerks of the system are created using the `createStaffMember` operation, while the first manager of a store is created along with the store, in the `initializeStore` function of the `StoreManager`. None of the users can delete themselves, only managers can delete staff from the store that they manage, and all the staff users are linked to a specific store. These decisions are made to ensure that all stores have at least one manager at all times, including their creation, and simplified management of roles (as all users will have at most one role in the system).

- **LineNumberManager:**

This component is responsible for handling all the transactions made with the line numbers on the system. The operations for creating line numbers (`generateLineNumber`, `getImmediateLineNumber` and `bookLineNumber`) may not be able to generate line numbers, due to system being stopped, maximum capacity is reached or set reservation limits are reached. The operation `check` is called by the Clerks to allow customers in and out of the system, having its first argument indicate the direction. The actual realization of this argument could be a boolean value, an enum or a custom typing instead of a string with two values.

- **StoreManager:**

This component is responsible for handling any update or query regarding a store, including creation, forecasting and monitoring. The `updateStore` operation is modeled as it retrieves all the changes to the store, however the operation can also be receiving partial updates of the store data instead of the whole store data, so that the network load for the request can be reduced. The operations to update the related data (`updateTimeSlots` and `updatePartnerStores`) feature *operation* arguments of type `CUDOperation` (CUD stands for Create, Update, Delete), which indicate that the request may not only be about updating a field or a specific instance of the given entity, but also to create new ones and remove existing ones. Some operations may cause an invalidation of customer's tickets, leading to a potential loss of customers. In order to prevent this from happening, the operations that might lead to this (`updateTimeSlots` and `systemStop`) feature a *force* argument. The call would produce an error if the force argument is not given, and there are customers who have booked any line numbers in an affected time slot. The operation `monitor` features an optional argument of 2 *DateTimes*. If these are not provided, the operation is to return the live monitoring result (the flow of customers in realtime).

The UML Class Diagram below describes the model of the system. In order to be consistent with the diagram in the RASD, and to demonstrate views and enumerations, the Class Diagram was preferred over an Entity-Relationship Diagram. For the sake of simplicity, the unique *ids* of entities are not shown. These *ids* don't have any special requirements, thus can be auto-generated using built-in generation mechanisms existing in SQL database implementations. The diagram is similar to the one in RASD, however features more implementation details.

The **LineNumberStatus** indicates the current **LineNumber**'s status. When the line number is generated, it should be initialized with the status *AWAITING* which indicates that the customer didn't yet enter the store. After the entrance of the customer, the line number enters to the state *VISITING* to indicate that the customer has successfully arrived at the store and currently shopping. When the customer leaves, the ticket becomes *VISITED* to indicate a successful exit from the store. If, for any case, the customers' ticket becomes invalid (due to time slot changes, system stop or ticket timeout), the ticket is marked as *EXPIRED*. The *entryTime* and *exitTime* are used to store the time interval that the customer has entered to the store and exited, to allow better monitoring. The features to time out the tickets after some time, reservation limiting, and product category selection to increase granularity of visit can be enabled or disabled by the manager. Thus, on the class **Store** attributes related to these features (*reservationLimit*, *timeoutMinutes* and *productCategories*) have a multiplicity of 0..1 to demonstrate this ability. The representation of the location on the diagram is given as the Latitude and Longitude on the Earth, according to ISO 6709. The latitude and longitude are limited with 4 decimal points, since this provides a precision of at least 12 meters on the Earth, which is in the range provided by D_5 . The **OccupancyForecast** is implemented as an estimated indication of amount of customers per time slot. Since the main purpose of this component is to provide an idea to the customers about how crowded the store is in a certain time slot, the granular implementation is the part of the *MonitorResult* view instead. The *MonitorResult* view provides the amount of customers that has visited the store from and to a specific time, which may or may not correspond to a **TimeSlot**. This view is used to implement a basic monitoring view for the **Manager**. The **ProductCategory** class features an *locatedAt* property in order to help customers know where to find a specific category. The **InStoreLocation** class provides a name for the location in the shop and a description of how to reach there. It may also feature an image of a sketch, diagram or a map indicating the location, considering many large chains already have such a facility. Some store may feature these in an interactive format in their websites or in any other format, however considering that the category location is static, such dynamism is not required. In order to facilitate, a screenshot may be provided in the system of such interactive representation.

2.6.1 Thick-Server

Copyright © 2021, Roberto Buratti, Hrvoje Hrvoj, Ozan Incesulu – All rights reserved 24

2.6.2 Logical Sharding

Since the data on the system is either strictly location dependant (stores and any subsidiaries exist in a specific province or country) or changes location rarely (Customers don't switch countries or provinces often), the data plane can be logically sharded based on location if necessary. With logical sharding, the system can scale without hitting any bottleneck regarding database application limits. Furthermore, the data tier can be provisioned on locations closer to the sharded target location, thus reducing the network latency.

2.6.3 Lazy Loading

Store, the largest entity contains foreign fields that needs multiple accesses to fully retrieve, which would decrease the overall performance of the system, as PartnerStore, TimeSlot, Categories . Thus, this problem necessitates the implementation of lazy loading for such entities on the ORM level.

2.6.4 Service-Oriented Architecture

Many services and options that our system provides can be split into different services, sharing the same database. However, since there is no scalability concern, due to the ability to perform logical sharding, or version compatibility problems, since all system components are planned to exist on same machine, there is no need to implement microservices. Thus, the system is implemented with Service-Oriented Architecture, which allows the developers to split the implementation of functionalities belonging to different domains, while not requiring to implement containerization or provisioning of multiple databases and data brokers. The services are exposed to each other through dependency injection and exposed to outside world via HTTP.

2.7 Other design decisions

2.7.1 ORM

To facilitate the development of the system, instead of hand crafting database queries, an ORM shall be used to further improve development speed and security, as many ORM implementations support the prevention of SQL related attacks.

2.7.2 ACID

The database operation is based on the ACID principles, since the core system doesn't require asynchronous data synchronization with any other external components. Furthermore, as the system will implement Logical Sharding, there wouldn't be a need for increasing the partitions of the database, thus the system will not suffer from the limits of the CAP theorem.

3 User Interface Design

3.1 User Interface Diagram

The following diagram represent the flow of events that the user can choose to follow in order to obtain the service they're looking for. From any window it's always possible to go back to the home window related to the role of the user, this arrows are not explicitated in the diagram for a cleaner diagram. The mockups of the application appearance are presented in the RASD document.

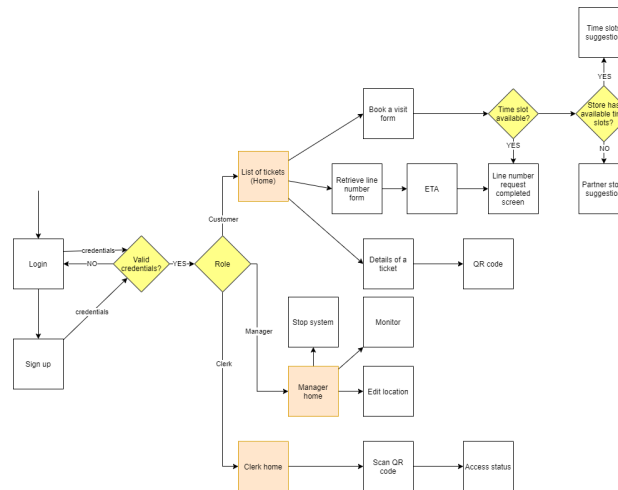


Figure 17: User interface diagram

4 Requirements Traceability

4.1 Functional Requirements

4.2 Non-functional Requirements

5 Requirements Traceability

6 Effort Spent

Date:	Person:	Part:	Time (in hours):	Description:
29/11/2020	Ozan Incesulu	General Structure	1	Imported and built the general document structure, replaced template parts with new project, add sections
07/12/2020	Roberto Buratti	Architectural design	2	Deployment view
08/12/2020	Ozan Incesulu	Component Diagrams	2.5	Created component diagrams for Section 2.2
09/12/2020	Ozan Incesulu	Component Descriptions	1.5	Wrote the component descriptions for Section 2.2
14/12/2020	Ozan Incesulu	Sequence Diagrams	3	Created sequence diagrams for interactions Add Staff Member, Emergency Stop and Update Store Information, with the explanatory texts.
26/12/2020	Ozan Incesulu	Component Interfaces	3	Created component interfaces diagram
28/12/2020	Ozan Incesulu	Class Diagram	3	Created Class Diagram
29/12/2020	Ozan Incesulu	Sequence Diagram for Monitor Store	1	Created the sequence diagram with description text for Monitor Store
29/12/2020	Roberto Buratti	UX diagram	1.5	Added user interface diagram and its description

References