CLup

Customers Line-Up

DD

Design Document

Software Engineering II Project
Computer Science and Engineering
Politecnico di Milano
Version 1.0 - 10 January 2021

Davide Merli - 10578363
Dario Passarello - 10566467

Deliverable:	DD
Title:	Design Document
Authors:	Davide Merli, Dario Passarello
Version:	1.0
Date:	10 January 2021
Download page:	https://github.com/davidemerli/MerliPassarello/
Copyright:	Copyright © 2020, Davide Merli, Dario Passarello – All rights reserved

Contents

List of Figures

List of Tables

1 Introduction

1.1 Purpose

The purpose of this document is to analyze and describe deeply the design choices of the software and the system to be, which is described in the CLup Requirements Analysis and Specifications Document (RASD).

Design decisions will be further explained together with the architectural structure of the system as a whole as well as its subsystems.

This document will also provide insights about Implementation and Testing, by outlining the characteristics of CLup subsystems and interfaces. Every component will be analyzed and put into perspective with the others, in order to deploy a plan for the developement, the organization and the parallelization of the work.

1.2 Scope

Due to the recent coronavirus global pandemic, many of the human activities have been drastically affected and restricted by the need to contain the virus diffusion. Norms and regulations may vary from contry to country, but almost everywhere the main focus is to mantain social distancing to avoid diffusion from one person to another. One of the most difficult activity to fullfil (yet absolutely essential) is grocery shopping.

Stores are forced to restrain access to avoid too many people inside the building, and this produces endless lines out of the stores. This can both increase the danger for people waiting for their turn and force the shops to regulate customers even outside the structure.

CLup aims to reduce heavily the issues involving customer queues outside of stores by permitting clients to keep track of their position in the store queue or book in a visit in advance with an easy to use application.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Acronyms

- **S2B**: Software To Be
- **RASD**: Requirement Analysis and Specifications Documents
- **REST**: REpresentational State Transfer
- **API**: Application Programming Interface
- **UX**: User Experience
- **UI**: User Interface
- **SSO**: Single sign-on
- **QR code**: Quick Response code
- **OS**: Operating System
- **RAM**: Random Access Memory
- **LAN**: Local Area Network
- **GPS**: Global Positioning System
- **GB**: GigaByte
- **TCP/IP**: Transmission Control Protocol/Internet Protocol
- **HTTPS**: Hypertext Transfer Protocol Secure
- **IoT**: Internet of Things
- **MQTT**: Message Queuing Telemetry Transport
- **RAID**: Redundant Array of Independent Disks
- **UML**: Unified Modeling Language
- **TDD**: Test-driven development

1.3.2 Definitions

- **Access controller**: a subsystem that permits the entrance of customers into the store. It can be a device like a smart turnstile that reads customers tickets or just a person of the store staff manually scanning tickets.
- **Business account**: a CLup account that is destined to store managers or operators and therefore the 'business' side of CLup
- **In-Site ticket**: a ticket that is taken by a customer near the store. It can be both a virtual paperless ti an emitter near the store premises
- **Virtual ticket**: a ticket issued through the CLup application
- **Physical/Paper ticket**: a printed physical ticket, emitted by a printer near the store premises
- **Valid Ticket** (at time X): a ticket that has a code recognized by the CLup system and valid for the specified time
- **Time slot**: a time delta that is associated with a number of bookable tickets (which varies and is customizable from store to store)
- **People-Counting System**: a subsystem that permits the counting of the number of people inside the store. It can comprehend a device like a proximity sensor or a smart turnstile, or it can be a person of the store staff manually counting people.
- **Customer Application**: the CLup mobile application destined to customers that want to shop inside stores adopting CLup
- **Operator Application**: the CLup mobile application destined to store staff to monitor entrances and statistics
- **Store Main System**: the store main server that communicates directly with CLup servers. All store subsystems and smart devices should communicate with it through an Intranet
- **Geocoding API**: Geocoding converts addresses into geographic coordinates to be placed on a map. A Geocoding API allows the use of their services to permit translation between textual addresses and Latitude/Longitude coordinates
- **Map API**: An external services that provides operations of geographics maps and the download of map information, usually of the places in proximity of given geographics coordinates
- **Hashed Password**: When a password has been "hashed" it means it has been turned into a scrambled representation of itself. A user's password is taken and – using a key known to the site – the hash value is derived from the combination of both the password and the key, using a set algorithm.
- **Time to market**: is the length of time it takes from a product being conceived until its being available for sale.

- **Alpha Test**: a trial of machinery, software, or other products carried out by a developer before a product is made available for beta testing.
- **Beta Test**: a trial of machinery, software or other products in the final stages of development, carried out by a party unconnected with the development process.
- **DevOps**: is a set of practices that combines software development (Dev) and IT operations (Ops). It aims to shorten the systems development life cycle and provide continuous delivery with high software quality.
- **Quality Assessment**: is the data collection and analysis through which the degree of conformity to predetermined standards and criteria are exemplified.

1.4 Revision History

- 1.0: 10 January 2021 - First Release

1.5 Reference Documents

- R&DD Assignment A.Y. 2020-2021 - Elisabetta di Nitto, Matteo Giovanni Rossi
- Software Engineering II slides and material - Elisabetta di Nitto, Matteo Giovanni Rossi
- CLup RASD - Davide Luca Merli, Dario Passarello
- Material Design Guidelines
- Convetional Commits Guidelines
- Git - versioning system

1.6 Used Tools

- Umlet - for UML diagrams
- Draw.io - for other diagrams
- Proto.io - for UI Mockups
- Github - for code hosting and version control
- LaTeX - to write this entire document
- Visual Studio Code + Latex Workshop - as a LaTeX environment

1.7 Document Structure

Chapter 1 explains the purpose of this document, summarizes the scope of the CLup project and the relation between this Design Document and the Requirement Analysis and Specifications Document. Acronyms and Definitions used through the whole document are listed and explained. It provides the history of the document revisions and this very description of all the chapters, together with the documents used as a reference.

Chapter 2 presents the system on various levels of detail. An high level view is provided to further frame CLup as a system composed of different software running on different platforms and devices. Component Diagrams illustrate all the components of the various subsystems. Every component is defined and described, exhibiting its role in the system and how it communicates with other internal or external components.

Chapter 3 shows the most meaningful interfaces of the CLup Application, giving a nice idea of what the final product will look like from a customer's perspective.

Chapter 4 associates every component described in section 2 with the Requirements they contribute to satisfy, as a direct reference for developers.

Chapter 5 is another direct reference for developers as it expresses how the developement process will be organized, how and in which order the components of the whole system are to constructed. The same is done with the testing: how it relates to the developement process, how component are tested, and which technologies are to be employed.

Chapter 6 shows the time spent from each member of the team for writing this document.

2 Architectural Design

2.1 Overview

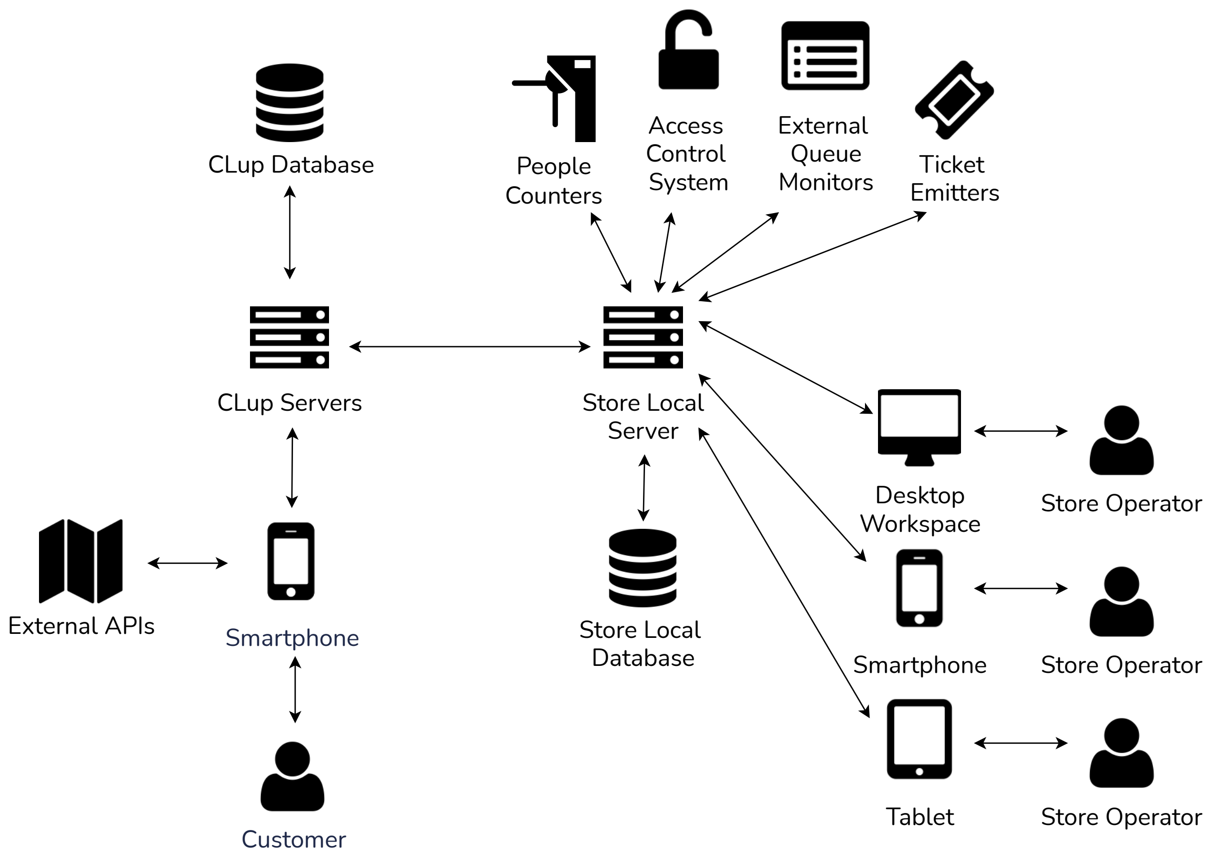


Figure 1: CLup System Overview

This diagram shows the CLup system as a whole, its actors and main components, from an high level perspective.

Store customers interact with the system by using a smartphone, which depends on external third party APIs to visualize and interact with maps. Store operators can interact with the system using different kinds of devices, to better suit their needs when working at the shop: smartphones can be handy when they need to check information about the store queue while moving through departments; tablets make looking at graphs and statistics more easily without loosing the 'mobile' factor, and having a CLup application also on desktops can be convenient for operators working at a desk (without having to switch from one device or another in their workflow).

CLup Customer Application communicates directly with CLup servers through REST APIs, while CLup Operator Application connects to the Store Local Server which retrieves data both from the Store Local Database and from CLup Servers.

Store Devices also connect to the Store Local Server, to enhance the functionalities and automate tasks such as admitting customer entrances, monitoring the people inside the store departments, handing out tickets.

CLup Servers on the other hand only store data that is needed to permit queue regulations, for example information about occupancy limits, live occupancy of the stores, markets locations.

Following sections will analyze every major component more deeply, describing in detail its underlying sub-components and interfaces.

2.2 Component View

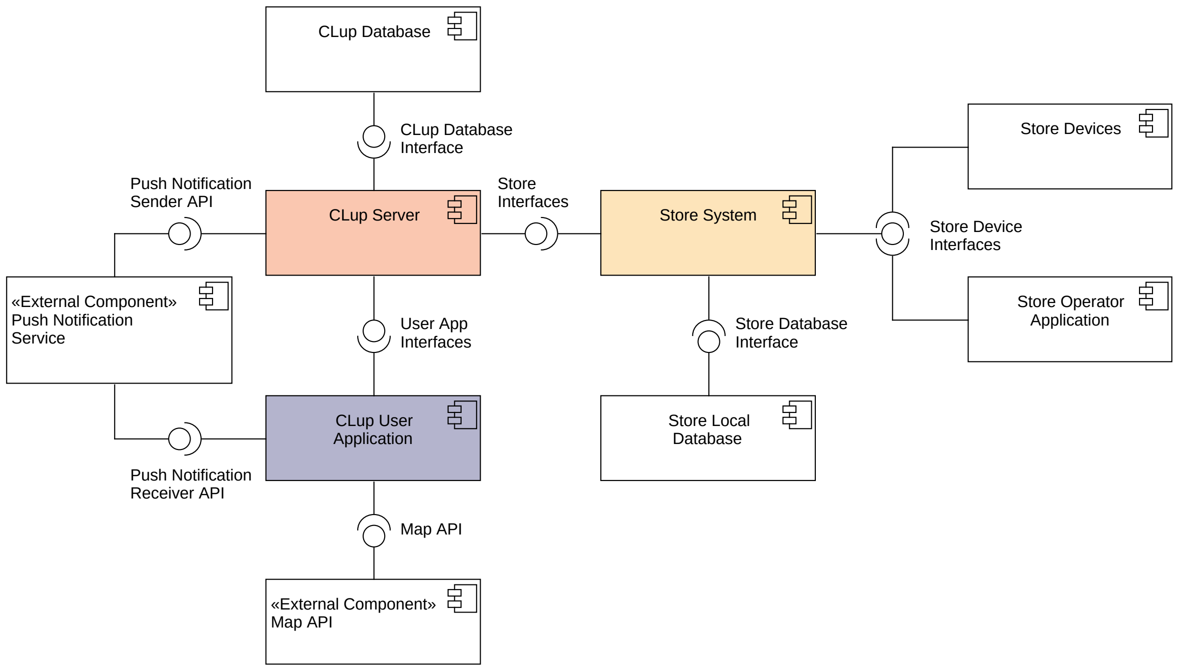


Figure 2: General component diagram

Figure 2 shows a general, simplified, component view of the S2B. The three main components of the system are: the CLup User Application, the CLup Server and the Store System with their databases. CLup system also interacts with external components like Map API Services and other physical components (i.e. store devices).

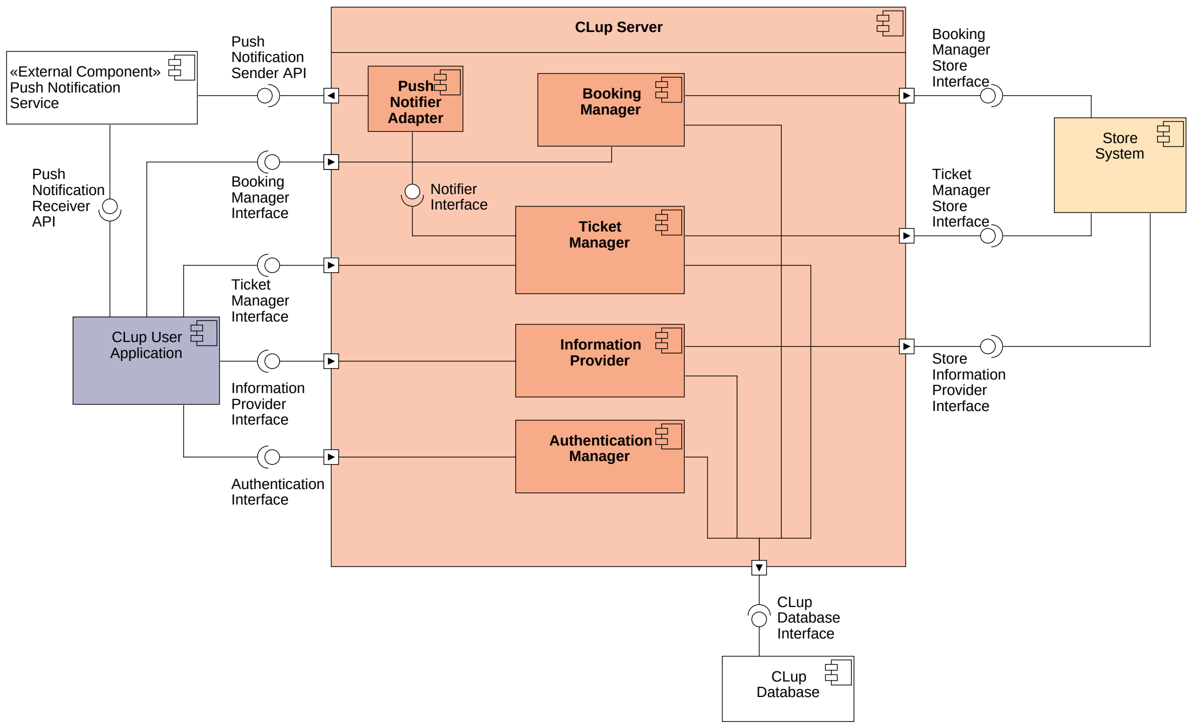
Customers use the **CLup User application** to check supermarkets near them and eventually book a visit or retrieve a ticket. The User application interfaces with the **CLup server**.

Customers need to receive notifications about their ticket, and to allow this the CLup Server and the application need to use the Push Notification service. Each OS manifacturer provides its dedicated API for sending and receiving push notifications.

The CLup server is the central component of the system, mediating between the customers and the stores. It manages customer authentication, tickets issued remotely by the CLup application, and hands out to the public information about stores to the customers (i.e. opening hours, waiting times, etc).

The **Store system** provides an interface for all the physical devices in the store, collecting and processing data from them and then routing the data to the CLup server or keeping it in the local database. The system communicates with a local database to store temporary data about tickets, the statistics collected and the authentication details of the store operators.

2.2.1 CLup Server component view



CLup Server
«External Component» Push Notification Service
Push Notification Sender API
Push Notifier Adapter
Booking Manager
Booking Manager Store Interface
Store System
Push Notification Receiver API
Booking Manager Interface
Notifier Interface
Ticket Manager
Ticket Manager Store Interface
Ticket Manager Interface
CLup User Application
Information Provider
Store Information Provider Interface
Information Provider Interface
Authentication Manager
Authentication Interface
CLup Database Interface
CLup Database

Figure 3: Component diagram for CLup server

The CLup server provides interfaces to the CLup User application and to each Store System. This component stores the persistent data in the CLup Database component communicating with it using a common Database interface.

The CLup Server has these internal components:

- **Authentication Manager**: handles customer registration and authentication. To do this task this component needs to communicate tightly with the CLup database which stores the user data and their hashed passwords;
- **Information Provider**: hands out information to the customers requesting them. These information can be of various nature, like opening hours or the store crowd-

edness in various times of the day. This data is retrieved from the CLup database or from the Store System which uses another interface to push them periodically.

- **Ticket Manager**: issues virtual tickets when customers requests them. This component interfaces directly with the Store System, adding the created tickets to the queue. The component has the responsibility to notify the customers when it's time to approach the store entrance; it uses the notification interface, provided by the Push Notifier Adapter component.
- **Booking Manager**: handles the visit booking. Queries the database about free slots in the day when a customer wants to make a reservation. Requests are eventually finalized by attaching the shop list to the booking if and when the customer adds it from the application. This components also pushes updates to the Store System about changes regarding the bookings (e.g. cancellation, shopping list changes, etc).
- **Push Notifier Adapter**: handles all the aspects about sending push notifications to the users, taking into account that the CLup application can be running on different operating systems and that they provide different services to receive these notifications. The notifier needs to communicate with the external Push Notification Service, using the interface provided by that service.

2.2.2 CLup Customer Application component view

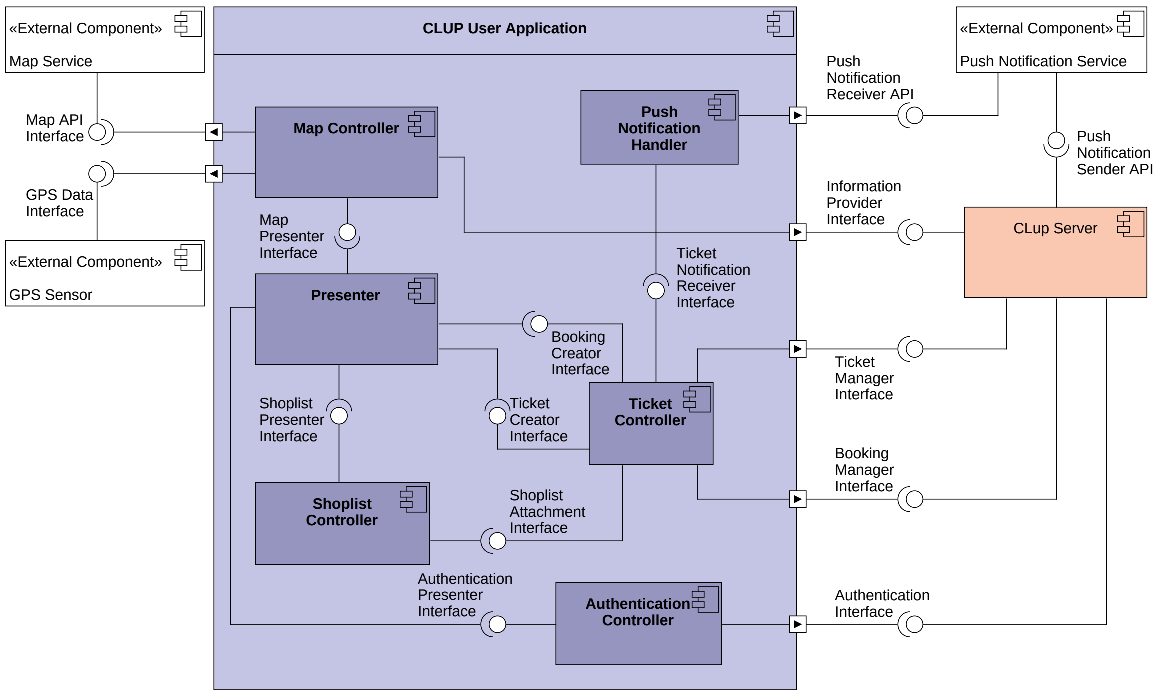

«External Component» Map Service
CLUP User Application
«External Component» Push Notification Service
Push Notification Receiver API
Map API Interface
Map Controller
Push Notification Handler
Push Notification Sender API
GPS Data Interface
Information Provider Interface
CLup Server
Map Presenter Interface
«External Component» GPS Sensor
Ticket Notification Receiver Interface
Presenter
Booking Creator Interface
Ticket Manager Interface
Shoplist Presenter Interface
Ticket Creator Interface
Ticket Controller
Booking Manager Interface
Shoplist Controller
Shoplist Attachment Interface
Authentication Presenter Interface
Authentication Controller
Authentication Interface

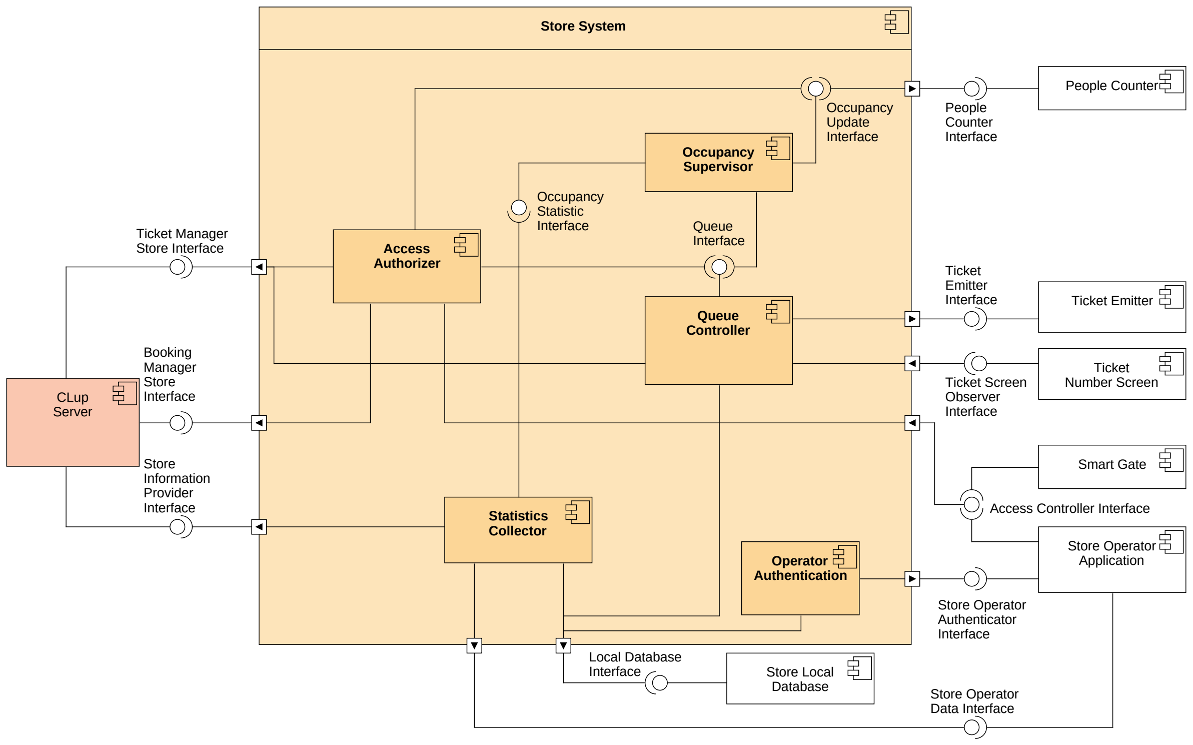
Figure 4: Component diagram for CLup User Application

The CLup user application provides the functionalities of CLup to store's customers. The applications needs to interface with the GPS Sensors and to an external map service that provides a map containing all the stores. The application also needs some controllers that will communicate with the CLup server to retrieve store information, authenticate, book a visit or create a queue ticket.

The user application contains these sub-components:

- **Authentication Controller**: allows the user to create a CLup account and to log-in with an existing account.
- **Map Controller**: Handles the communication with the external Map Service and the GPS. Exploits the Map API provided by the Map Service to get the map data near the user position; then it will request data from the CLup server in order to populate the map with nearby stores. It needs to access data about the location of the user. For this task the component will use the Information Provider Interface offered by the CLup server component and GPS Data Interface.
- **Presenter**: Handles the GUI of the applications and the events from the user like the push of a button or the text boxes. Needs to relate with all the controllers components that handle the different aspects and functionalities of the application. To do this this component exposes many interfaces to be able to display information coming from all the different controllers that allow the application to function correctly.
- **Ticket Controller**: Handles all the steps needed to retrieve a Numbered Ticket and to Book a visit to the store. It communicates directly with the CLup Server using one of the two interfaces provided from the CLup Server (Ticket Manager Interface, Booking Manager Interface). It also ensures that the user could not book another ticket when has a ticket already active and also it handles the cancellation of a booking or a ticket.
- **Shop-list Controller**: manages the creation of shop lists that will be attached to visit bookings in order to estimate the occupancy of the various department of the store.
- **Push Notifications Handler**: handles and displays to the user the push notification received from the CLup Server about the bookings or the tickets previously created. This component is provided by the underlying Operating System, and the interface is accessible through standard libraries of the application programming language.

2.2.3 Store System component view



Store System
People Counter
People Counter Interface
Occupancy Update Interface
Occupancy Supervisor
Occupancy Statistic Interface
Queue Interface
Ticket Manager Store Interface
Access Authorizer
Ticket Emitter Interface
Ticket Emitter
Queue Controller
Ticket Number Screen
Ticket Screen Observer Interface
Booking Manager Store Interface
CLup Server
Smart Gate
Access Controller Interface
Store Information Provider Interface
Statistics Collector
Store Operator Application
Operator Authentication
Store Operator Authenticator Interface
Local Database Interface
Store Local Database
Store Operator Data Interface

Figure 5: Component diagram for Store system

The Store System receives data from the store physical devices and combines it with the information retrieved by the CLup Servers. This component uses the interfaces provided by the CLup Server and lays out its own interfaces that are used by the physical devices located in the store. This component also interfaces with a local database containing store statistics, store operators authentication data and local data that need persistency.

- **Operator Authentication**: allows store operators to authenticate using their store credential. This component communicates with the Store Operator Application, which is used to manually check tickets and register the customer admissions in the store. The component fetches the authentication data in the store local database, using the interface provided by the database.
- **Occupancy Supervisor**: checks that the occupancy limits are not violated using all the data that it receives about people entering and leaving the store. This data can be provided directly by devices like People Counters, or from the Access Authorizer. The latters increments the counter of the number of people inside the store when an access is authorized. The Occupancy Supervisor checks information about the bookings and notifies the Queue Controller when there is space to admit new customers with a numbered ticket (without a booked visit).
- **Queue Controller**: handles the queueing of the customers by combining the data about virtual numbered tickets and paper numbered tickets, using the proper interfaces to communicate with the CLup Server. The Queue Controller is notified by

the Occupancy Supervisor when customers in line can be called to the entrance; it then updates the ticket numbers on the screen placed at the store entrance (with the new numbers obtained using its interface).

- **Access Authorizer**: receives the code read by the Access Controller (that can be either a Smart Gate or a store operator), checks if that code is valid and allowed to enter, then notifies back the Access Controller. Even if the ticket is valid, before admitting a customer, this component needs to check that the occupancy limits of the store will not be violated by letting the customer inside the store, consulting the Occupancy Supervisor. The Access Authorizer will check (requesting information from the CLup server) if the ticket refers to a pre-booked ticket; otherwise it will check if the ticket is on top of the queue using the Queue Controller interface.

- **Statistics Collector**: collects and periodically elaborates statistics with data from the Occupancy Supervisor and Queue Controller. Statistics are saved in the store local database. Brief elaborated data that can be shown to the customer (i.e. indications of the crowdedness of the store, estimated waiting times,...) are sent periodically to the CLup Server using the Store Information Provider interface.

2.3 Deployment View

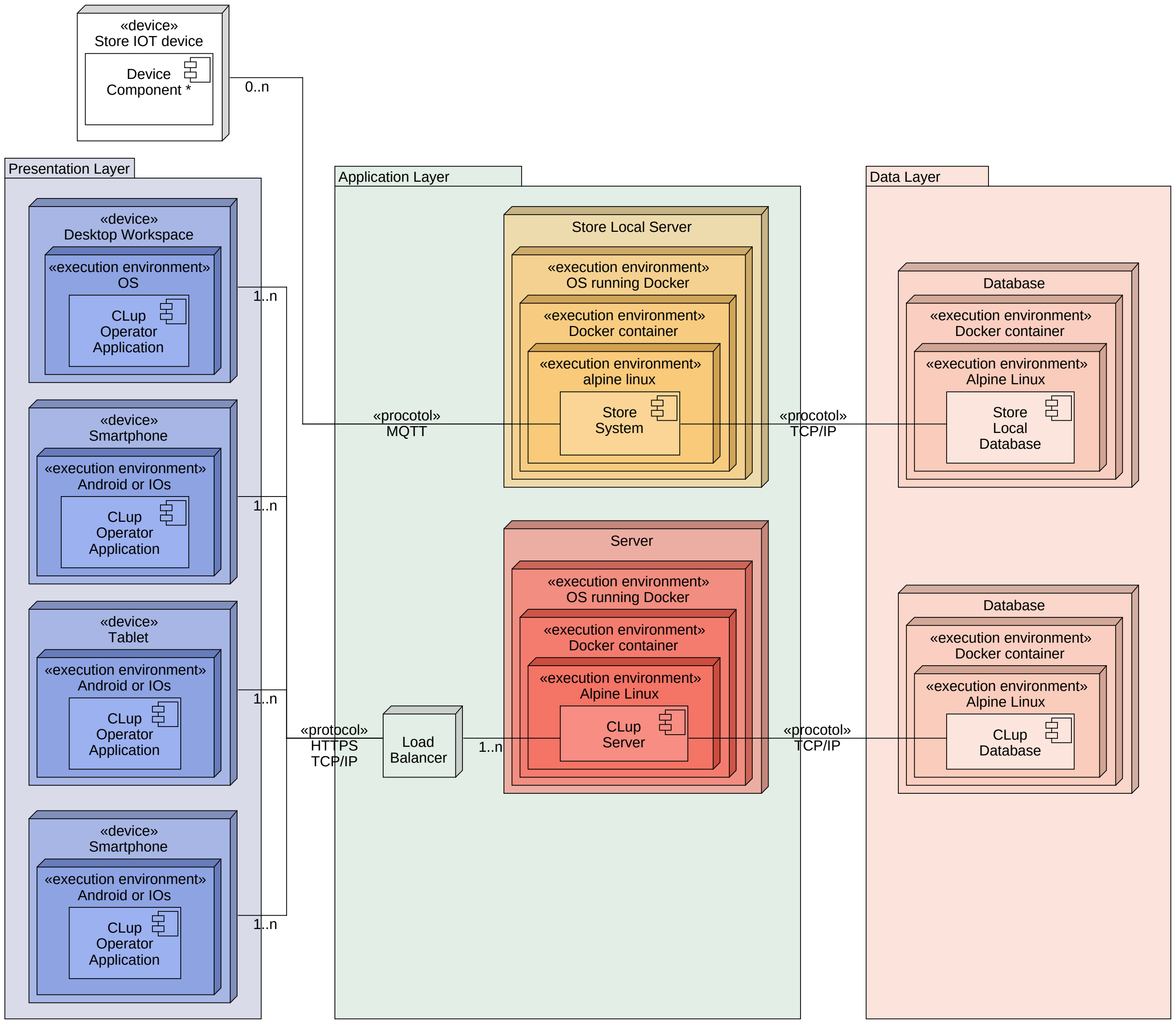


Figure 6: Deployment Diagram
** a single Device Component is shown for the semplicity of the representation. Also details on the operating system and/or environment are not provided since there is no particular need to prefer one solution to another. The preferred choice is made on the protocol side and it is MQTT.*

The system is structured in a Three Tier Architecture, and the Deployment Diagram visually presents the system divided in the three different layers, **Data**, **Application** and **Presentation**.

In the Application and Data layers, we chose to exploit the functionalities provided by Docker, and use containers running Alpine Linux to further improve the standardization of the deployment process, simplify the testing on real hardware, and adhere to best practices. Alpine Linux has been chosen since it is designed for security, simplicity, and resource efficiency and it is vastly used to deploy Docker containers with.

A Load Balancer is essential to provide a reliable service, and splits the traffic over multiple instances of the Web Server; a possible candidate for the load balancing task is Nginx.

The Web Server is not presented as a component since it is not strictly required when describing the functionalities of the system, but it is needed since the system heavily relies on REST APIs. Furthermore, if the traffic is really intense and/or variable, it could be an economically viable choice to outsource the Web Server on a Serverless Architecture. Allocating machine resources on demand can be a desired behaviour for an application like CLup, which can possibily see an exponential increase in users over short periods of time, and very different loads depending on the time of the day. The developement has been organized in very stand-alones components, so migrating from one solution to another will be as simple as possible.

2.4 Runtime View

The aim of the following sequence diagrams is to show how the components (the actors in the diagrams) interact between them at runtime.

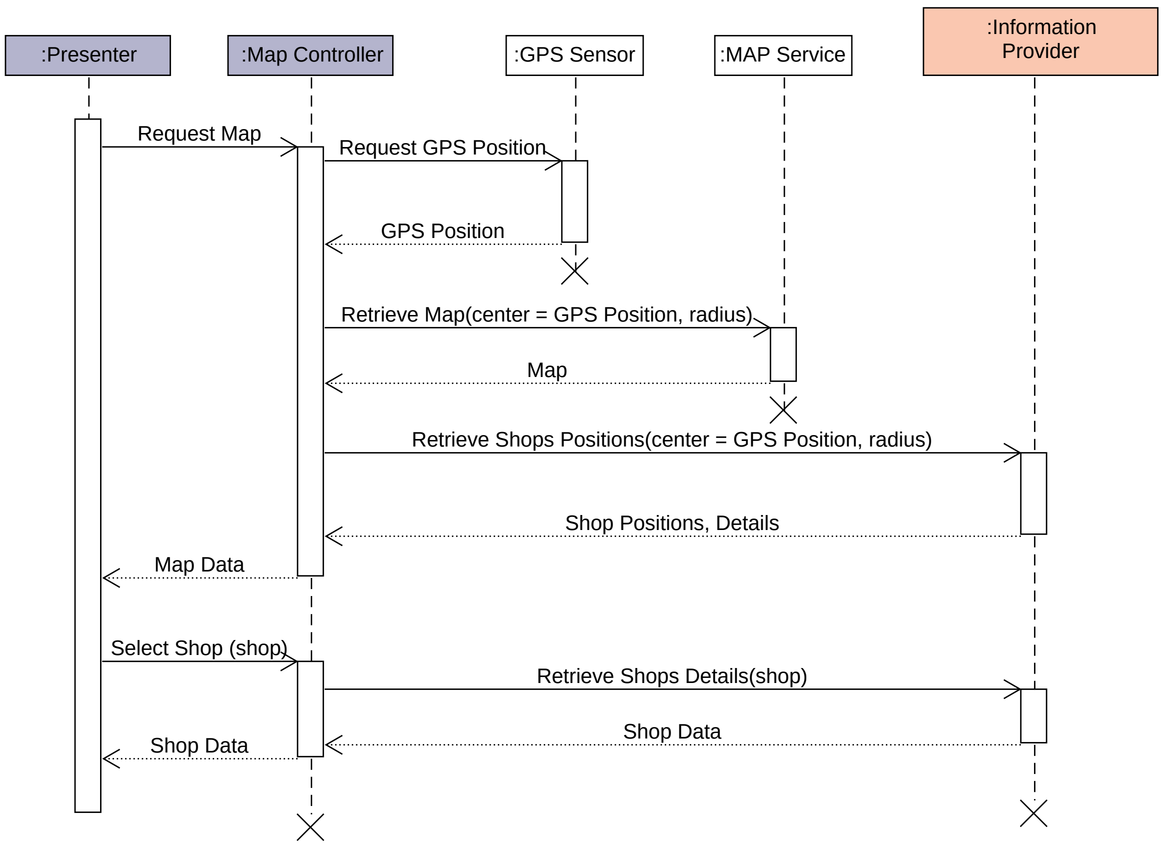


Figure 7: Sequence diagram for store map loading

The diagram in Figure 7 shows how the system loads the map of all stores and shows it to the user[1].

To determine which part of the map to load, the system needs to acquire the user position consulting the GPS sensor external component (external from the CLup system view, but built in most smartphones). The MAP API is then called to get the map data; concurrently a call to the Information Provider has to be done in order to get the positions and the basic information of all the stores to show in the map. When the Map Controller has all the data at its disposal, the map can finally be rendered by the Presenter.

[1]UC3 - Customer searches for the store page, refer to RASD

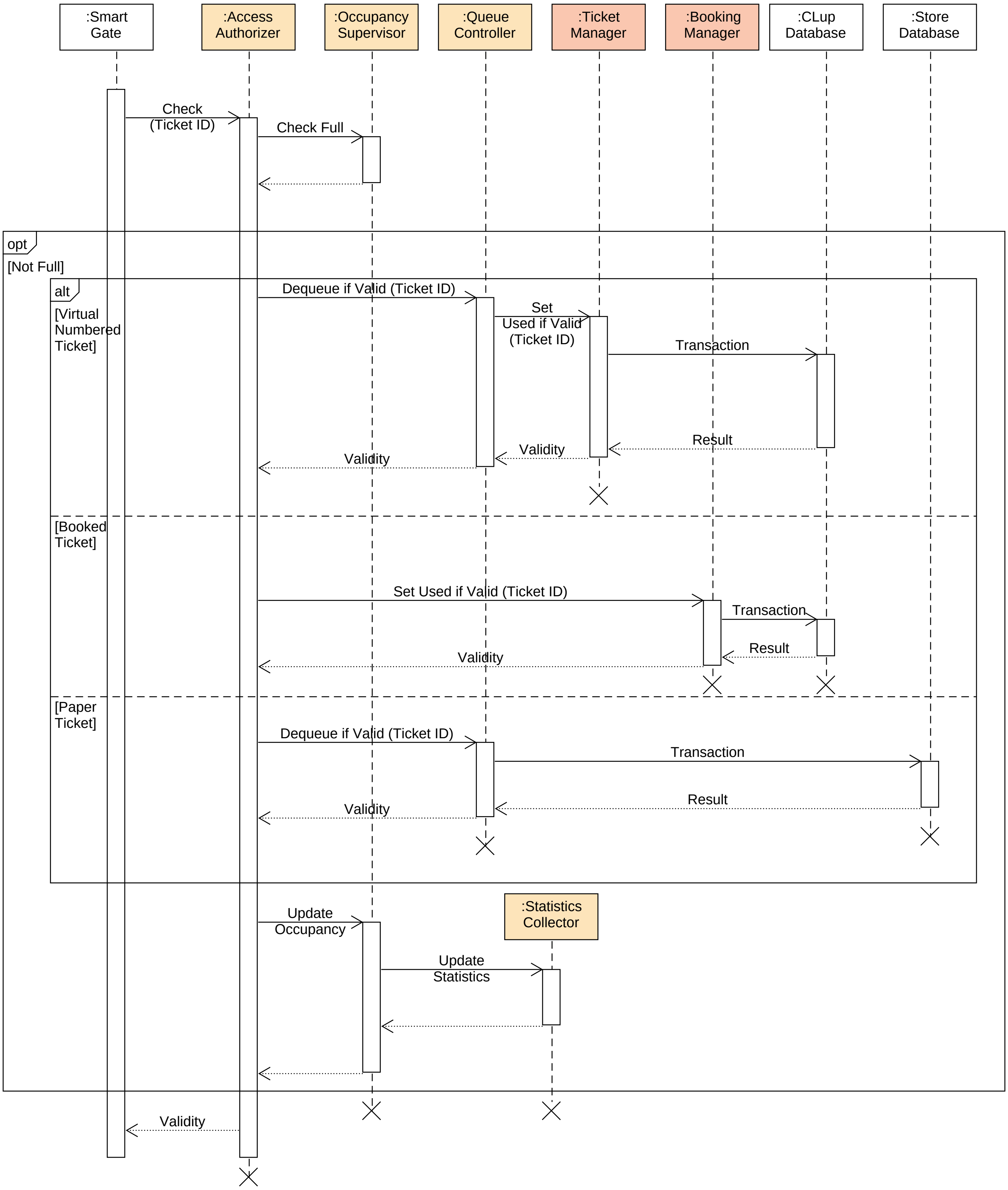


Figure 8: Sequence diagram for ticket scanning

The diagram shown in Figure 8 shows how the system behaves when a ticket is scanned[2]. The Access Authorizer receives the Ticket ID scanned by a Smart Gate (or an Operator). Before checking if the ticket ID exists and is valid, the store live occupancy is checked. If the supermarket is full the entrance can be denied (based on the supermarket policies).

Then the Access Authorizer figures from the ID which type of ticket has been scanned; this could be achieved, for example, by reserving a digit of the ticket ID to differentiate ticket types. Depending on the ticket type different actions are carried out:

- Virtual Numbered Ticket: if the ticket is valid it is removed from the store queue and the ticket status is set to ‘Inside Shop’ on the CLup main database
- Booked Ticket: the Booking Manager is in charge of checking if the ticket is valid at the time of the scan
- Paper Ticket: if the ticket is valid it is removed from the store queue and the ticket is set to ’used’ on the CLup main database

After this, the Occupancy Supervisor is called again and updates the people count, then the Access Controller grants the customer the access to the store.

[2]UC10 - Customer scans the ticket through an access control system to enter, refer to RASD

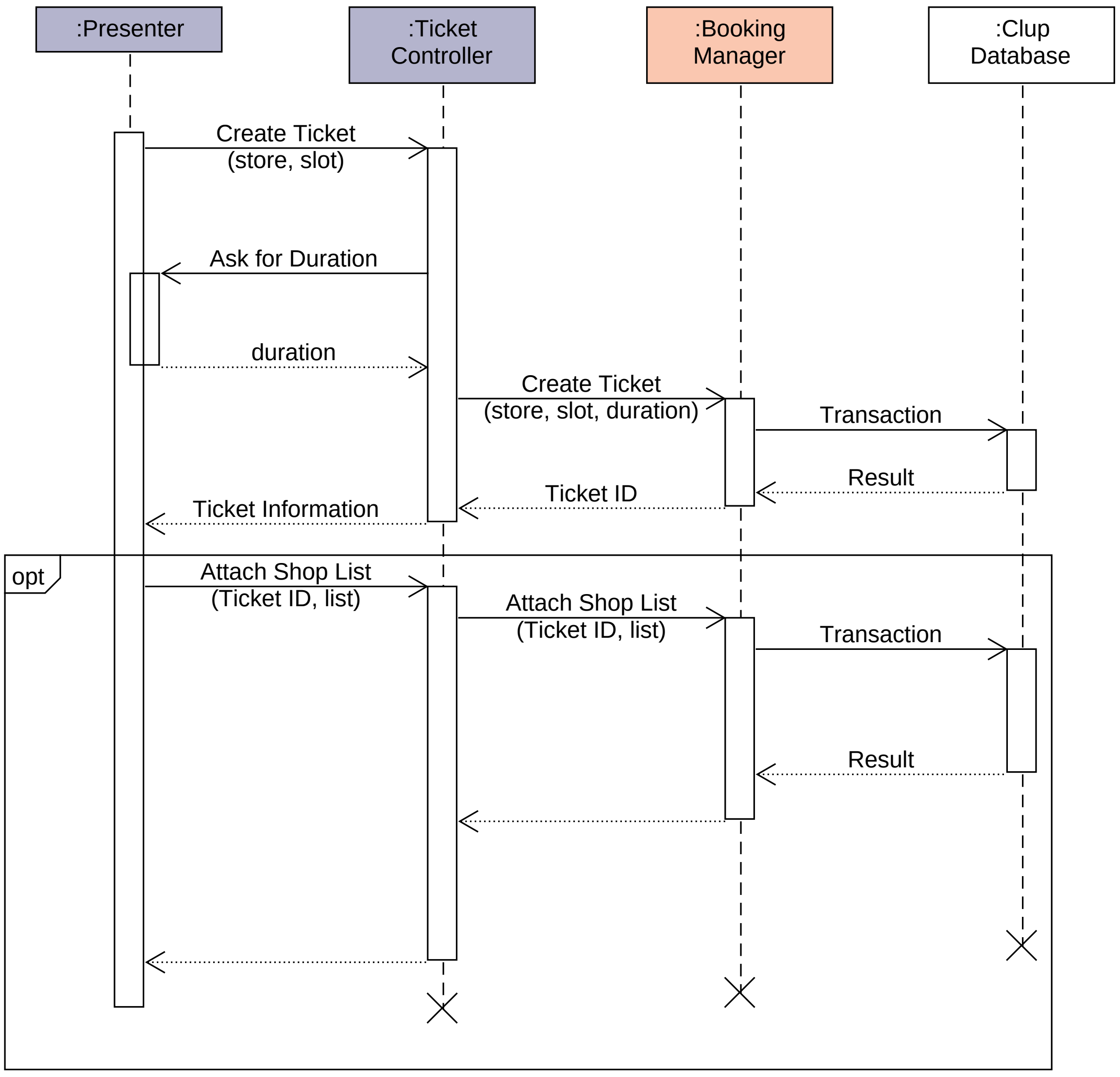


Figure 9: Sequence diagram of book a visit functionality

The diagram shown in Figure 9 shows how the system behaves when the user requests to book a visit in a store timeslot[3]

The presenter asks the Ticket Controller to generate a reservation; after asking the user the estimated duration of the visit, the Booking Manager component in the CLup Server is called. When the transaction in the CLup Database has succeeded, the Ticket ID will be returned to the Ticket Controller, which will send the information needed to render the Ticket View to the Presenter, and show it to the user. After the booking is done, the user can optionally attach a shopping list to the reservation he just made.

[3]UC 5 - Customer books a visit in a Store, refer to RASD

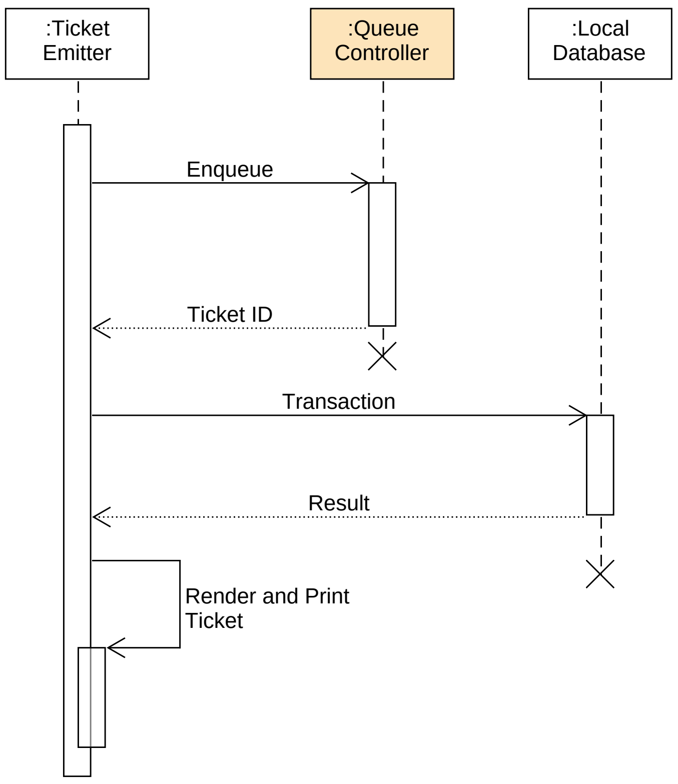


Figure 10: Sequence diagram for printed tickets

The diagram shown in Figure 10 explains how the system behaves when a customer goes to the store and retrieves a ticket directly from a ticket emitter[4]. Before printing the ticket the emitter adds the customer to the queue, then this ticket is temporarily stored in the store local database. After this the ticket emitter renders the ticket and prints it.

[4]UC 8 -Customer creates a physical numbered ticket, refer to RASD

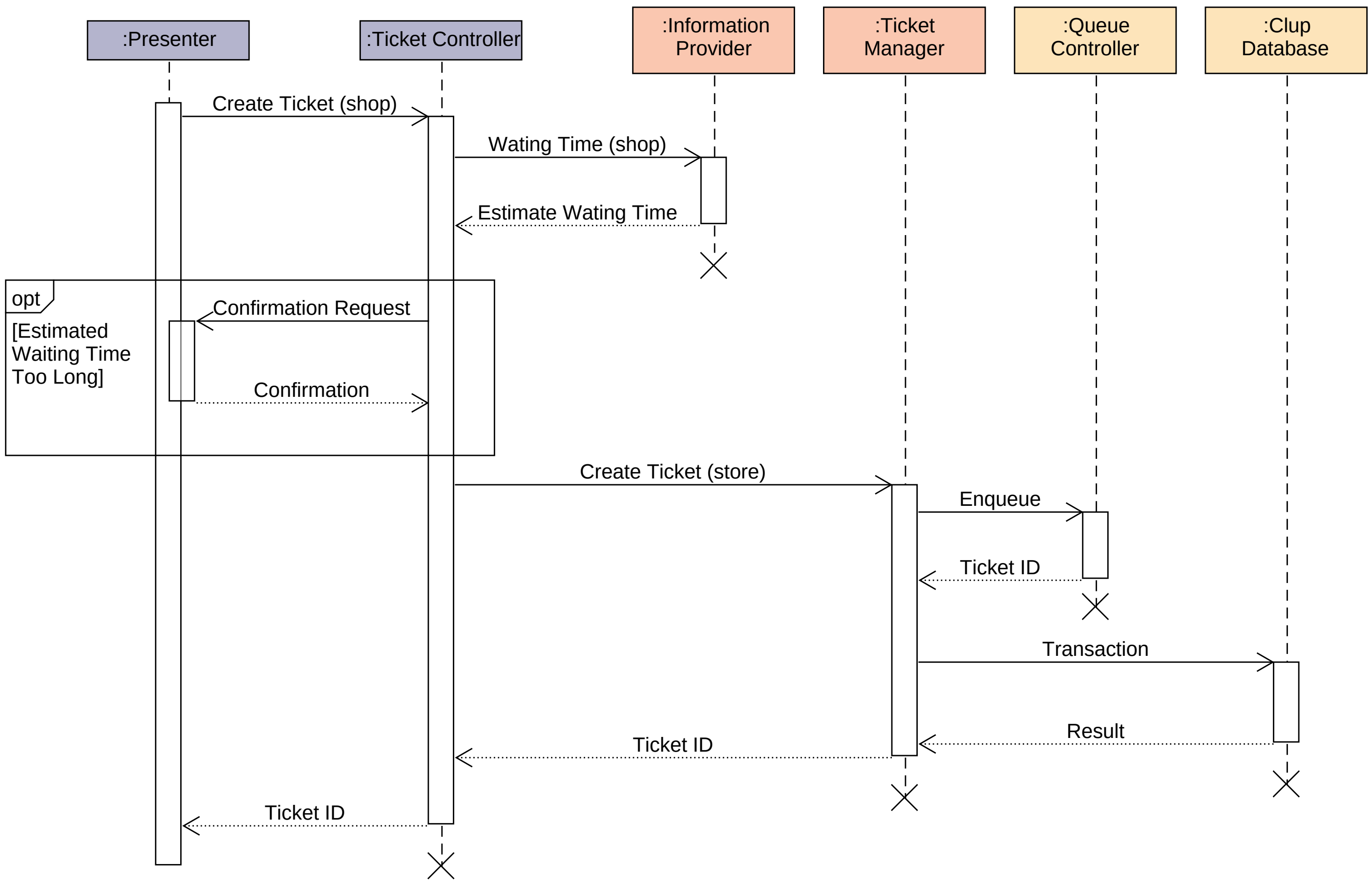


Figure 11: User creates a virtual numbered ticket, runtime view

The diagram shown in Figure 19 illustrates the process for creating a virtual numbered ticket to enter the store, from the User Application[5]. The transaction is initiated when the presenter calls the Ticket Controller; this component first asks the information provider how much queue time the user has to wait. If this time is higher than a certain threshold, the user is informed and has to explicitly confirm the request. The ticket manager then calls the Ticket Manager, which sends an enqueue request to the store Queue Controller. Finally the ticket is saved to the CLup main database.

[5]UC 7 -Customer creates a numbered virtual ticket to enter a store as soon as there is a place available, refer to RASD

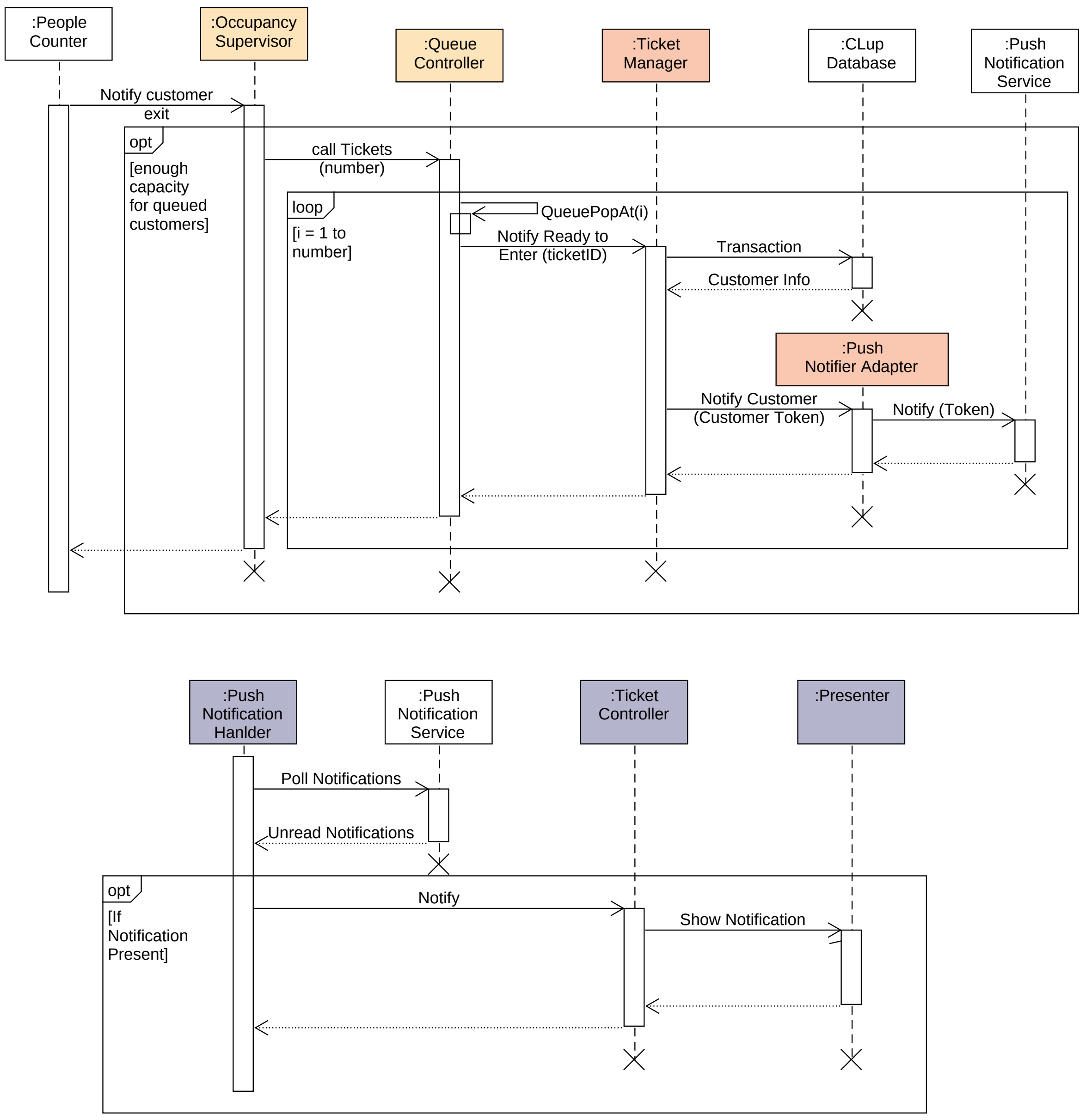


Figure 12: The store notifies the first customers in queue when they can enter, runtime view

The diagram shown in Figure 12 shows the notification process when there is enough capacity for allowing queue customer.

This notification is usually started by a People Counter when it detects that someone left the store. When this event happens, the People Counter calls the occupancy supervisor that calculates how many queued people can enter the store, taking into account the actual occupancy and how many people have booked visits in the incoming slots. If more than zero people are allowed to enter the shop, then a number of tickets is popped from the Queue Controller, starting the first. For each popped ticket, if that ticket is virtual then the Ticket Manager is called to notify that user using the Push Notification Service (and the user notification token bound to the ticket), passing through the adapter component.

The customer device periodically checks for new notifications, calling the API provided by the Push Notification service; if a Notification related to CLup is detected, the Ticket Controller is called and will call the Presenter that will show the push notification.

2.5 Component Interfaces

Here are presented all the interfaces already mentioned in section 2.2. Every interface is described in detail with the data that is passed from one component to the other, showing the type of expected response from an interface, and the data required to provide when performing a request.

The interfaces are written in pseudocode using a Java-like syntax. In addiction, some details of the specification are:

- **square brackets** indicate optional parameters (used for example when responses return an error, the returned value only has the responseType since no valid data could be retrieved)
- **interfaces** keyword is used when more than one interface have the same structure
- **class** keyword is used to represent aggregations of data, they do not necessary reflect as objects or classes, but are implementation specific. This notation can be compared to definitions of NamedTuples in Python
- variable types are specified only when defining "classes" or when type specification is needed to explain the behaviour of the interface
- functions returning **Response** usually do not need to return any kind of data, but only a response about the result of the operation. This can be extended if needed during implementation
- '**...**' are used when the interface is provided by an external component, or when a 'class' or method can have more parameters to further specify/give other details about the passed information which are not strictly necessary

```
/**
 * It is possible to login using another valid authToken
 * provided by services like login with a social network account
 */
interfaces AuthenticationInterface, StoreOperatorApplication,
    AuthenticationPresenterInterface {

    class LoginResponse(responseType, [authToken]);

    LoginResponse loginWithCredentials(email, password);

    //For using an Single Sign On service
    LoginResponse loginWithAnotherAuthService(authenticator, authToken);
}
```

```
/**
 * The Information Provider yields publicly available data,
 * so authentication is not required
 */
interfaces InformationProviderInterface, StoreInformationProviderInterface {
   class StoreCharacteristicsResponse(responseType,
       [storeID, name, address, gpsPosition, clupStoreSettings]);
   class StoreLiveDataResponse(responseType, [storeID, occupancyData, ...]);

   StoreCharacteristicsResponse getStoreCharacteristics(storeID);
   StoreLiveDataResponse getStoreLiveData(storeID);
   List<storeID> getStoresNearLocation(gpsPosition, kmRadius);
}

interface TicketManagerInterface {
   class TicketDetails(ticketNumber, queuePosition, storeID, issueDate, validFrom, validTo);
   class TicketRequestResponse(responseType, [ticketID, TicketDetails]);
   class TicketRevokeResponse(responseType, [ticketID]);
   class TicketDetailsResponse(responseType, [TicketDetails]);

   TicketDetailsResponse getTicketInformation(authToken, ticketID, storeID);
   TicketDetailsResponse getNextIssuableTicketInformation(authToken, storeID);
   TicketRequestResponse requestTicket(authToken, storeID);
   TicketRevokeResponse revokeTicket(authToken, storeID, ticketID);
}

interface BookingManagerInterface {
   class BookingDetails(ticketNumber, storeID, issueDate, validFrom, validTo);
   class TimeSlot(timeSlotID, validFrom, validTo, availableReservations);
   class BookingRequestResponse(responseType, [bookingID, BookingDetails]);
   class BookingRevokeResponse(responseType, [bookingID]);
   class TimeSlotsRequestResponse(responseType, [List<TimeSlot>]);

   TimeSlotsRequestResponse getTimeSlots(authToken, storeID, fromDate, toDate);
   BookingRequestResponse requestBooking(authToken, storeID, timeSlotID);
   BookingRevokeResponse revokeBooking(authToken, storeID, timeSlotID);
}

/**
 * Both interfaces are provided by the device OS CLup application is running on.
 * The Notification Adapter transparently handles multiple types
 * of Push Notification APIs for the other components.
*/
interfaces PushNotificationSenderAPI, PushNotificationReceiverAPI {...}

interface BookingManagerStoreInterface {
   Response cancelBooking(authToken, timeSlotID);
   Response shoplistUpdate(authToken, timeSlotID, shoppingList);
}

interface TicketManagerStoreInterface {
   class TicketDetails(ticketNumber, queuePosition, storeID, issueDate, validFrom, validTo);

   Response cancelTicket(authToken, ticketID);
   Response updateTicket(authToken, ticketID, TicketDetails);
}

// Transactional Database interfaces, provided by the chosen Database technology
interfaces CLupDatabaseInterface, LocalDatabaseInterfaces {...}
```

```
// Map API interface, provided by the chosen Map Provider Service
interface MapAPIInterface {...}

// GPS Interface, provided by the GPS sensor built on
// the device using CLup Application
interface GPSDataInterface {
    ... getCurrentPositions();
}

interface MapPresenterInterface {
    class MapResponse(responseType, [mapData]);

    MapResponse getMap(gpsPosition, kmRange, [filters]);
    Response selectShop(shopID);
    Response addToFavorites(shopID);
    Response removeFromFavorites(shopID);
}

interface ShoplistPresentInterface {
    Response addShoppingList(shoppingList);
    Response addItemToShoppingList(shoppingList, newItem);
}

interface TicketCreatorInterface {
    class TicketDetails(ticketNumber, queuePosition, storeID, issueDate, validFrom, validTo);
    class TicketRequestResponse(responseType, [ticketID, TicketDetails]);
    class TicketRevokeResponse(responseType, [ticketID]);

    TicketRequestResponse generateTicket(storeID);
    TicketRevokeResponse cancelTicket(storeID, ticketID);
}

interface BookingCreatorInterface {
    class BookingDetails(ticketNumber, storeID, issueDate, validFrom, validTo);
    class BookingRequestResponse(responseType, [bookingID, BookingDetails]);
    class BookingRevokeResponse(responseType, [bookingID]);

    BookingRequestResponse generateBooking(storeID, timeSlotID);
    BookingRevokeResponse cancelBooking(storeID, bookingID);
}

interface ShoplistAttachmentInterface{
    Response updateShoplist(storeID, bookingID, shopList);
}

interface TicketNotificationReceiverInterface {
    void showNotification(notificationContent);
}

interface NotifierInterface {
    Response sendNotification(customerID, notificationContent);
}

interfaces OccupancyUpdateInterface, PeopleCounterInterface {
    Response increaseCounter(authToken, amount, [customerIDs]);
    Response decreaseCounter(authToken, amount, [customerIDs]);
}
```

```
interface OccupancyStatisticsInterface {
    Response pullOccupancyStatistics();
}

interface QueueInterface {
    Response popUserFromQueue();
    Response pushUserToQueue();
}this

interface TicketScreenObserverInterface {
    Response updateNumbers(calledNumbers, nextToBeCalledNumbers);
    Response updateQueueDetails(estimatedQueueTime, queueLength, ...);
}

interface AccessControllerInterface {
    Response requestEntrance(ticketID/bookingID);
    Response forceEntrance(ticketID/bookingID);
}

interface StoreOperatorDataInterface {
    Response getStoreStatistics();
    Response getStoreLiveStastics();
}

interface TicketEmitterInterface {
    Response printTicket(ticketID, queueNumber, ticketNumber, estimatedWaitingTime);
}
```

2.6 Selected Architecture Styles and Patterns

Distributed Model View Controller Behavioral Pattern

From the behavioral standpoint, the system adopts a Distributed Model View Controller Design Pattern. In this pattern an event triggered in the view calls the corresponding action provided by the controller interface. The controller checks if that action is legit and then calls the model, which will act depending on his initial state and the action provided by the controller. After the model has computed his new state it updates the view accordingly.

The system components map in the MVC pattern as stated below:

- View: Presenter
- Controller: Map Controller, Shop list Controller, Ticket Controller, Authentication Controller, Push Notification Handler
- Model: All the components inside the CLup Server and inside the Store System

The pattern is 'distributed' because the components run on different machines (the view and the controller on the customers' smartphones, the model on the servers). Also, the entire model is distributed because it runs in the CLup server and in the Store system and according to the deployment diagram, these two macro components run in different containers.

Three Tier Architecture Architectural Pattern

The system adopts on a three tier architectural pattern.

A three tier architecture is a special client–server architecture in which presentation, application processing and data management functions are physically separated. In the case of a Three Tier application each of this three function runs in separated components.

The CLup user mobile application handles the **Presentation Layer** and interfaces directly with the Application Layer, this case is referred as 'Thick Client'.

Otherwise, if the S2B can also provide a Web Application to be accessed through a browser, the Presentation layer is handled partly by the browser (that renders the HTML page) and partly by the Web Server that generates that HTML, filling a template with the data elaborated from the Application Layer. This case is referred as 'Thin Client'.

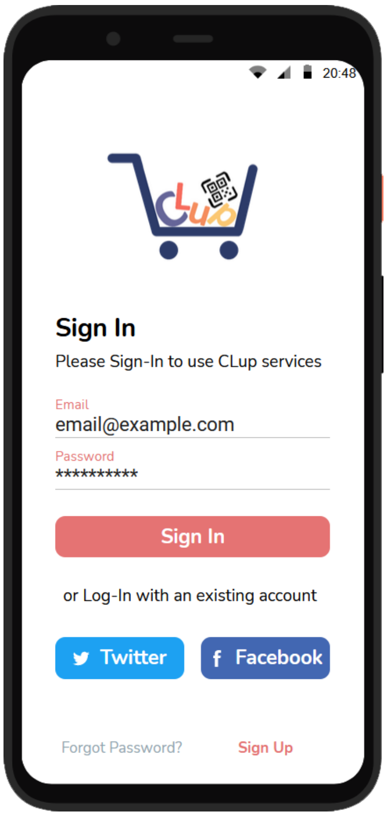
The **Application Layer** processes the data persisted in the Data Layer and executed Business Logic related operation, like managing the booking of a ticket, or retrieve the information requested from a client.

The presentation layer communicates with the application layer through a **Web Server** using the HTTP protocol. The Web Server allows to provide Web Pages to Thin Clients (i.e. a Browser) and to provide application Data to Thick Clients (i.e. a Mobile Application) via a **RESTful API**.

The **Data Layer** manages the store, the consistency and the persistance of the Data. The Data Layer is implemented from the Database and the DBMS, it handles the transactions started and controlled from the Application Layer, it handles the consistency of the data and its physical reliable and secure storage system.

3 User Interface

Following the design choices explained in section *3.1.1 User Interfaces* of the RASD, mockups of all the main sections of the application are provided. The components follows Google's Material Design guidelines[6].



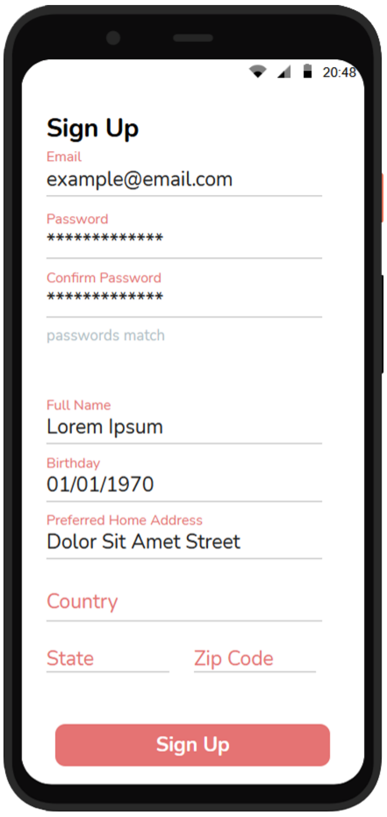
Sign In

Please Sign-In to use CLup services

Email
email@example.com

Password
****

(a) Login



Sign Up

Email
example@email.com

Password
****

Confirm Password
****

passwords match

Full Name
Lorem Ipsum

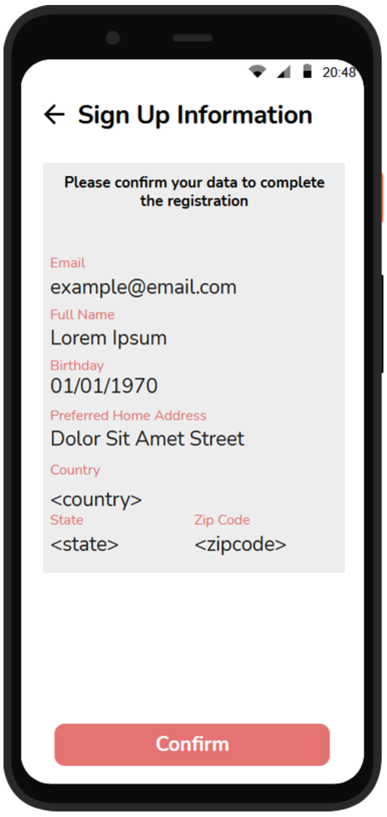
Birthday
01/01/1970

Preferred Home Address
Dolor Sit Amet Street

Country

State | Zip Code

(b) Registration



Sign Up Information

Please confirm your data to complete the registration

Email
example@email.com

Full Name
Lorem Ipsum

Birthday
01/01/1970

Preferred Home Address
Dolor Sit Amet Street

Country
<country>

State
<state>

Zip Code
<zipcode>

(c) Confirm Registration

Figure 13: Authentication Process

[6]https://material.io/design/guidelines-overview

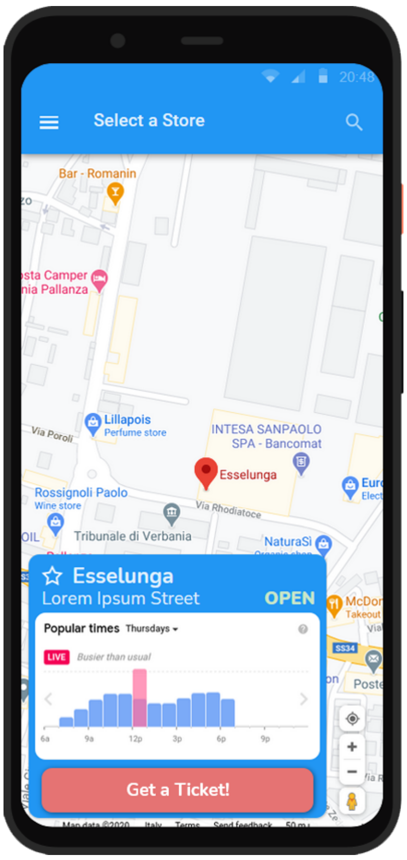
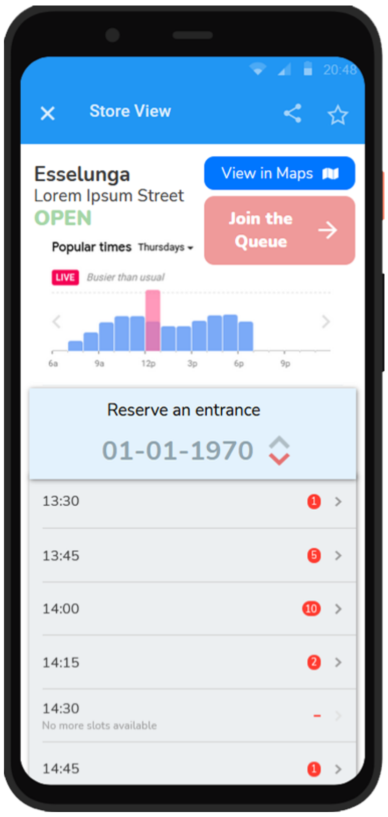
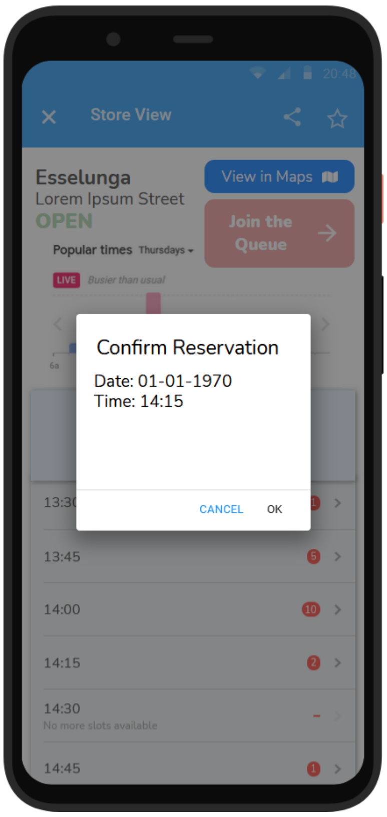


Figure 14: Map



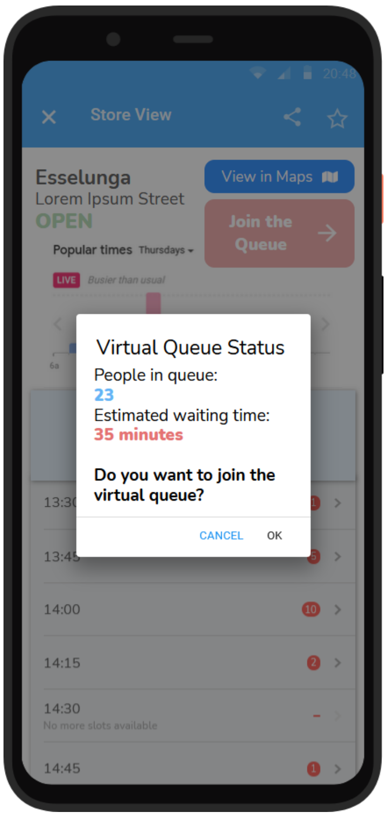
13:30	1
13:45	5
14:00	10
14:15	2
14:30 No more slots available	-
14:45	1

(a) Store View


Confirm Reservation

Date: 01-01-1970
Time: 14:15

(b) Confirm Reservation

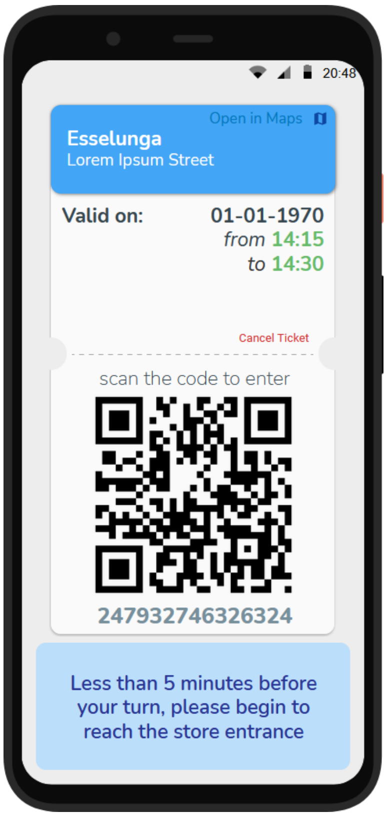

Virtual Queue Status

People in queue:
23
Estimated waiting time:
35 minutes

Do you want to join the virtual queue?

(c) Confirm Joining Queue

Figure 15: Ticket Retrieving



20:48

Open in Maps

Esselunga
Lorem Ipsum Street

Valid on: 01-01-1970
from 14:15
to 14:30

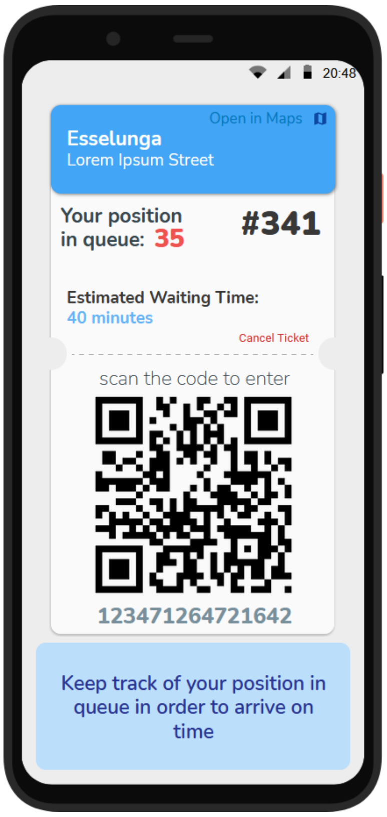
Cancel Ticket

scan the code to enter

247932746326324

Less than 5 minutes before your turn, please begin to reach the store entrance

(a) Reservation Ticket View



20:48

Open in Maps

Esselunga
Lorem Ipsum Street

Your position in queue: 35 #341

Estimated Waiting Time:
40 minutes

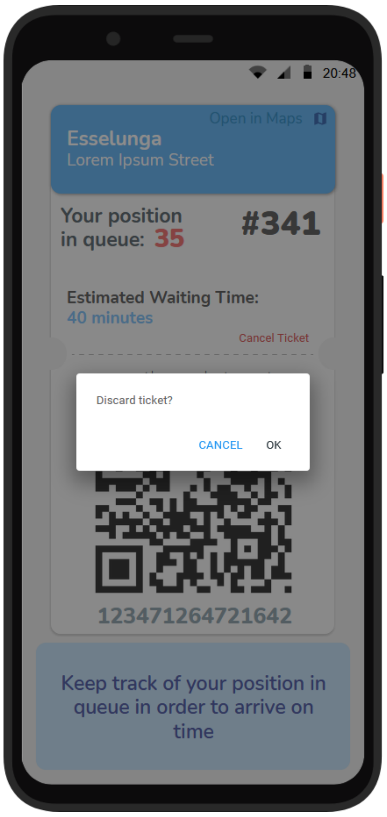
Cancel Ticket

scan the code to enter

123471264721642

Keep track of your position in queue in order to arrive on time

(b) Queue Ticket View



20:48

Open in Maps

Esselunga
Lorem Ipsum Street

Your position in queue: 35 #341

Estimated Waiting Time:
40 minutes

Cancel Ticket

Discard ticket?

CANCEL OK

123471264721642

Keep track of your position in queue in order to arrive on time

(c) Confirm Cancel Ticket

Figure 16: Ticket Interfaces

4 Requirement Traceability

In this section each component is mapped with the requirement that it contributes to satisfy. Obviously this is not a one to one mapping but each requirement is satisfied from a combination of components and each component contributes to satisfy one or more requirement.

It is guaranteed that each requirement is covered by at least one component and, vice versa, each component contributes to the satisfaction of at least one requirement.

4.1 CLup Server Component Traceability

- **Push Notifier Adapter**
  - **R18** The system must notify the customers with a virtual queue ticket when it's time to approach the store entrance
  - **R34** The system must push notifications to user devices with update information on the store he has a ticket for
- **Push Notifier Adapter (External Component)**
  - **R18** The system must notify the customers with a virtual queue ticket when it's time to approach the store entrance
  - **R34** The system must push notifications to user devices with update information on the store he has a ticket for
- **Ticket Manager**
  - **R17** The system must allow a customer to create a numbered virtual queue ticket and notify them if he can enter immediately (if the store is not full) or provide them a waiting time estimation
  - **R18** The system must notify the customers with a virtual queue ticket when it's time to approach the store entrance
  - **R35** The system must associates tickets with line numbers
- **Booking Manager**
  - **R10** The system must provide an interface for user to compile a shopping list
  - **R12** The system must allow the store-admin account to create and edit entrance time intervals
  - **R13** Each time interval must have a number of bookable slots fewer than the store capacity
  - **R14** The system must allow authenticated users to book a visit in a desired time interval
  - **R15** The system must not allow a user to book a slot in an already full time interval

  - **R16** The system must not allow a user to book a visit if he has already reserved another visit
  - **R20** The system must ask the customer to provide the estimated visit time when booking a time slot

- **Information Provider**
  - **R4** For each store the system must allow the users to retrieve information about location and business hours
  - **R27** The customer CLup application must show brief statistics about average occupancy of each stores during different days of the week
  - **R30** The stores adopting CLup must be displayed on a map

- **Authentication Manager**
  - **R36** The system allows customer to register an user account
  - **R37** The system allows registered customers to authenticate

- **CLup Database**
  - **R1** The system must keep general information and contacts about the store chains adopting CLup
  - **R4** For each store the system must allow the users to retrieve information about location and business hours
  - **R14** The system must allow authenticated users to book a visit in a desired time interval
  - **R17** The system must allow a customer to create a numbered virtual queue ticket and notify them if he can enter immediately (if the store is not full) or provide them a waiting time estimation
  - **R26** The system must record periodically and store statistics about the occupancy of each store
  - **R27** The customer CLup application must show brief statistics about average occupancy of each stores during different days of the week
  - **R30** The stores adopting CLup must be displayed on a map
  - **R35** The system allows customer to register an user account
  - **R36** The system allows registered customers to authenticate

4.2 CLup User Application Mappings

- **Map Controller**
  - **R4** For each store the system must allow the users to retrieve information about location and business hours
  - **R27** The customer CLup application must show brief statistics about average occupancy of each stores during different days of the week
  - **R30** The stores adopting CLup must be displayed on a map
  - **R31** The CLup customer application allows user to mark stores as favorite in order to access them quickly
- **Presenter**
  - **R29** The customer CLup application must be cross-platform and must work on the majority of the devices
  - **R32** The CLup customer app allows to book visit and retrieve tickets directly from the store page
- **Shop List Controller**
  - **R10** The system must provide an interface for user to compile a shopping list
- **Push Notification Handler**
  - **R18** The system must notify the customers with a virtual queue ticket when it's time to approach the store entrance
  - **R34** The system must push notifications to user devices with update information on the store he has a ticket for
- **Ticket Controller**
  - **R14** The system must allow authenticated users to book a visit in a desired time interval
  - **R15** The system must not allow a user to book a slot in an already full time interval
  - **R16** The system must not allow a user to book a visit if he has already reserved another visit
  - **R17** The system must allow a customer to create a numbered virtual queue ticket and notify them if he can enter immediately (if the store is not full) or provide them a waiting time estimation
  - **R18** The system must notify the customers with a virtual queue ticket when it's time to approach the store entrance
  - **R20** The system must ask the customer to provide the estimated visit time when booking a time slot
  - **R32** The CLup customer app allows to book visit and retrieve tickets directly from the store page

  - **R34** The system must push notifications to user devices with update information on the store he has a ticket for

- **Authentication Controller**
  - **R36** The system allows customer to register an user account
  - **R37** The system allows registered customers to authenticate
- **Map Service (External Component)**
  - **R30** The stores adopting CLup must be displayed on a map
- **GPS Sensor (External Component)**
  - **R30** The stores adopting CLup must be displayed on a map

4.3 CLup Store System Mappings

- **Access Authorizer**
  - **R7** The system will let a customer enter the store if and only if they have a a valid ticket
  - **R8** The system will use the occupancy data retrieved from the store to control the store access
  - **R9** The system must provide an interface to communicate with the store access control
  - **R19** The store operator application must allow an authenticated operator to manually admit customers
- **Statistics Collector**
  - **R23** The system must try to estimate waiting time based on store capacity, reservations and the current number of people with numbered tickets waiting in line
  - **R25** The system must let the store-admin accounts retrieve statistics collected from CLup regarding their store
  - **R26** The system must record periodically and store statistics about the occupancy of each store
  - **R27** The customer CLup application must show brief statistics about average occupancy of each stores during different days of the week
- **Occupancy Supervisor**
  - **R5** The system stores information about capacity of each market
  - **R6** The system won't let anyone enter the store if the maximum capacity has been reached
  - **R11** The system must take in consideration shopping list data and historic data from previous user visits to reduce store crowdedness per department
  - **R26** The system must record periodically and store statistics about the occupancy of each store
  - **R28** The operator CLup application must show to an authenticated operator the real time occupancy of the store
  - **R33** The system must provide an interface for automated control devices to communicate to CLup data about store entrances, store leavings and crowdedness in the various departments
- **Queue Controller**
  - **R17** The system must allow a customer to create a numbered virtual queue ticket and notify them if he can enter immediately (if the store is not full) or provide them a waiting time estimation

  - **R18** The system must notify the customers with a virtual queue ticket when it's time to approach the store entrance
  - **R21** The system must allow stores to hand out numbered physical queueing tickets to those that do not use the CLup application
  - **R22** The system must allow the access to the store to customer with numbered tickets using a 'First Come First Served' logic, treating virtual and physical ticket owner equally
  - **R24** The system should interface with an screen placed at the entrance of the store to notify customer which ticket numbers will enter in the next called batch
  - **R34** The system must push notifications to user devices with update information on the store he has a ticket for
  - **R35** The system must associates tickets with line numbers

- **Operator Authentication**
  - **R2** The system must provide each store a store-admin account
  - **R3** The store-admin account must allow the creation of store-operator accounts
  - **R25** The system must let the store-admin accounts retrieve statistics collected from CLup regarding their store

- **People Counter**
  - **R8** The system will use the occupancy data retrieved from the store to control the store access
  - **R33** The system must provide an interface for automated control devices to communicate to CLup data about store entrances, store leavings and crowdedness in the various departments

- **Ticket Emitter**
  - **R21** The system must allow stores to hand out numbered physical queueing tickets to those that do not use the CLup application
  - **R35** The system must associates tickets with line numbers

- **Ticket Number screen**
  - **R24** The system should interface with an screen placed at the entrance of the store to notify customer which ticket numbers will enter in the next called batch

- **Smart Gate**
  - **R6** The system won't let anyone enter the store if the maximum capacity has been reached
  - **R7** The system will let a customer enter the store if and only if they have a a valid ticket

  - **R8** The system will use the occupancy data retrieved from the store to control the store access

- **Store Operator Application**
  - **R6** The system won't let anyone enter the store if the maximum capacity has been reached
  - **R7** The system will let a customer enter the store if and only if they have a a valid ticket
  - **R8** The system will use the occupancy data retrieved from the store to control the store access
  - **R19** The store operator application must allow an authenticated operator to manually admit customers
  - **R25** The system must let the store-admin accounts retrieve statistics collected from CLup regarding their store
  - **R28** The operator CLup application must show to an authenticated operator the real time occupancy of the store

- **CLup Store Database**
  - **R2** The system must provide each store a store-admin account
  - **R3** The store-admin account must allow the creation of store-operator accounts
  - **R21** The system must allow stores to hand out numbered physical queueing tickets to those that do not use the CLup application
  - **R25** The system must let the store-admin accounts retrieve statistics collected from CLup regarding their store
  - **R28** The operator CLup application must show to an authenticated operator the real time occupancy of the store
  - **R35** The system must associates tickets with line numbers

5 Implementation, Integration and Test Plan

In this section are discussed the details about the implementation of the system, showing a possible (but not unique) implementation plan, an unit and integration testing strategy and some useful DevOps practices that could ease and speedup the system development process.

5.1 Implementation Plan

As shown in the Figure 2 there are three macro component to implement from scratch. The other components either are up and running (i.e. a Map Service) or were already developed and the only step to do is setup, configure and deploy them. These components will be tested along the developed components in the Integration Testing part.

The three macro-components to be implemented are the CLup User Application, the CLup Server and the Store System. Because all the common interfaces were previously specified (See section 4) each of these three components can be implemented independently. This allows three different teams to work in parallel on the implementation of each macro component, speeding up the development process. Even though a full parallel implementation reduces the overall development time, in this section will be presented a plan that aims to develop and test the core features of the system and then expands these features. This strategy reduces the time to prototype and the time to market of the system and enables the opportunity of doing an alpha test of the system running in a real context, even if the features initially offered are reduced. A secondary objective of the presented plan is to schedule in a smart way the development of the components in order to reduce the number *drivers* and *stubs*. This decision reduces the testing code size while not hindering the quality of the tests.

Therefore the proposed implementation is divided into different iterations. The first iteration aims to produce a working system. The output of the first iteration is, in fact, a full stack working prototype of the system with only the core features. The second iteration refines the features presented in the first iteration and add other important features; new components will be integrated to the system, and optionally some components already developed in the first iteration are expanded. The successive iterations add quality of life features and polish the system which will be ready for the system test before the release.

For each iteration, a graph is provided to show the precedence relations between the components developed or refined during that iteration.

5.1.1 First Iteration

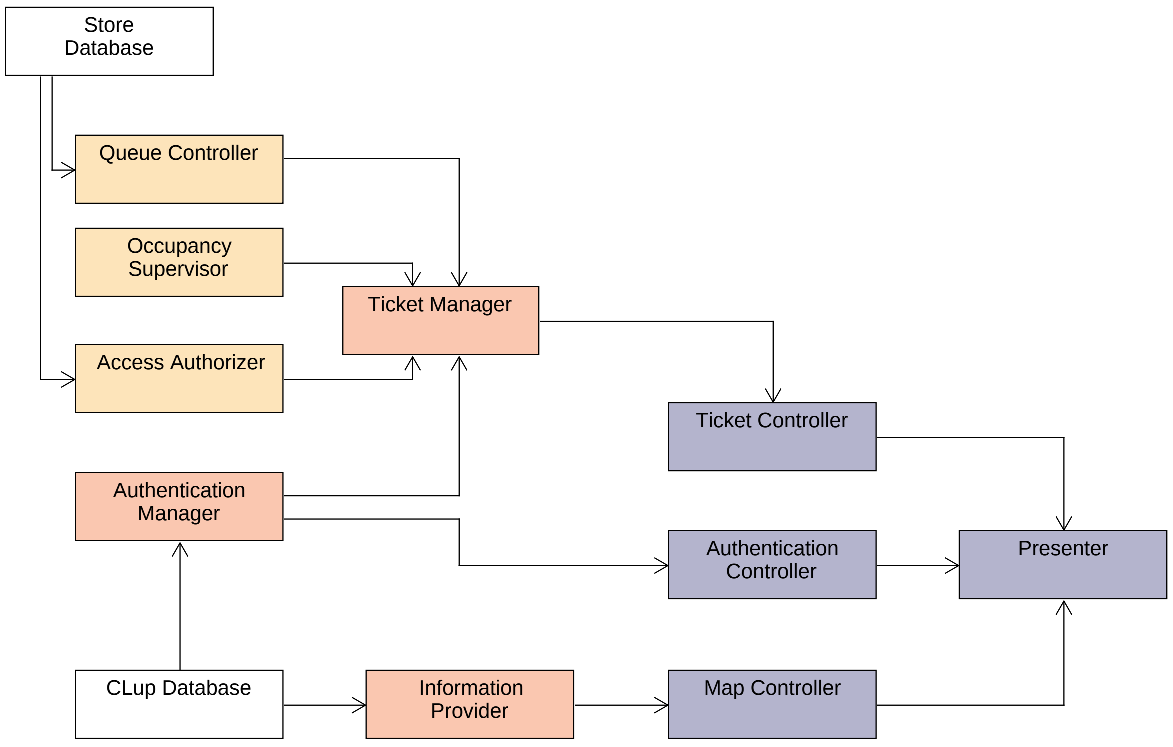


Figure 17: First Iteration of Implementation Plan

The first iteration sets up the system to provide the functionality for issuing a numbered ticket, either using the application or the ticket emitter. The first components to be set-up are the databases, then the development can continue with the essential backend components (CLup Server and Store System). After this, the development shifts on the frontend (CLup user application) essential features.

5.1.2 Second Iteration

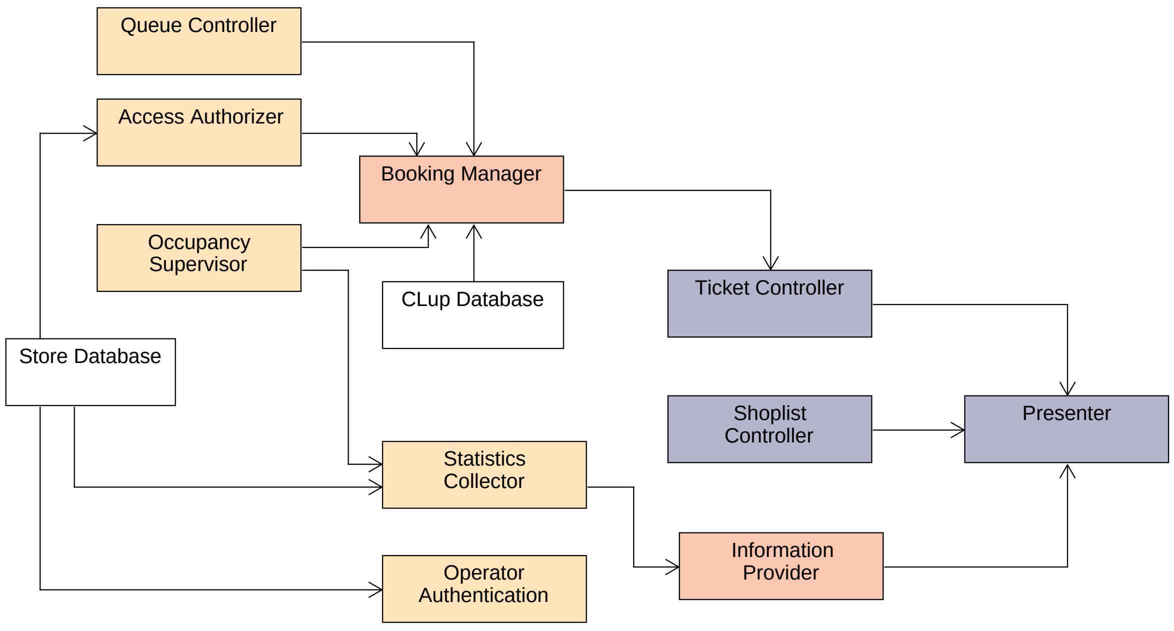


Figure 18: Second Iteration of Implementation Plan

In the second iteration the ‘Book a visit’ feature is implemented. At first the required data structures are created in the CLup Database and the subsequent components are created (if they not exist yet) or expanded to host the new features.

5.1.3 Third Iteration

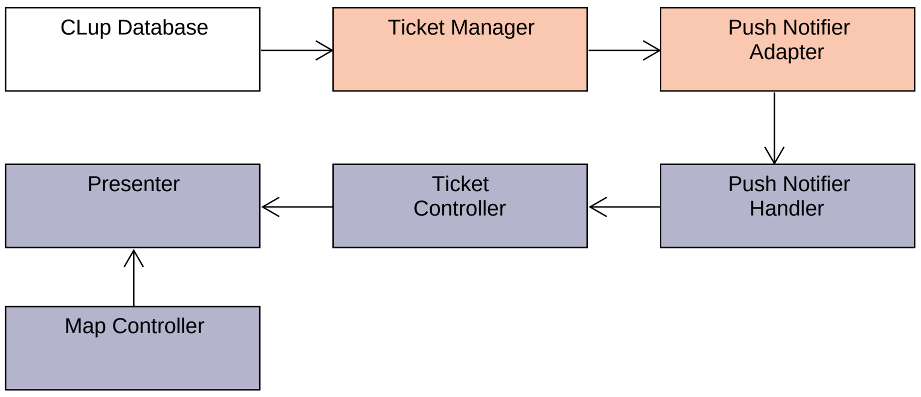


Figure 19: Third Iteration of Implementation Plan

The last iteration adds nice to have features in the system like ‘push notifications’ that require adding some new components and the ‘add store to favorites’ which requires the expansion of the Map Controller and the Presenter components.

5.2 Unit Testing

Each component will be developed using mainly Test Driven Development (TDD) strategy. This strategy consist in writing the unit test cases before writing the application code, forcing the developer to use black box testing over white box testing. In this way the developer has to think at all combinations of the inputs before writing the code. When or after writing the code the developer can always add some white box test cases if they feel that this addition could improve the coverage and the effectiveness of the test cases.

The unit testing should not pose a constraint on the programming languages chosen to implement the system, because nowadays almost all mainstream languages provide a unit testing library (i.e. JUnit for Java).

5.3 Integration and Testing

The integration of the components can follow the implementation plan with its precedence relations. When a component is implemented and unit tested, the component can finally be integrated and tested with all the other previously developed components.

To start a new iteration, the integration test has to be passed in the previous iteration.

The integration test can start in parallel to the implementation of the components, in fact a Quality Assessment team can start to integrate and test a component together with its predecessors when its implementation is concluded.

5.4 Other Development Tools

To develop the S2B a **versioning system**, like git, must be used. This allows to keep the history of the implementation in a repository, track issues in an easier way, revert changes that created errors and share code between the developers in a straightforward way.

To add a change in the repository the developers perform 'commits'; these commits include a message that explains briefly what are the contents of the commit. It's preferable to use a convention in these messages, to allow a program to automatically generate changelogs, and to force the developers to create small commits with small changes, instead of using big and monolithic commits that are difficult to understand. A widely used standard for commit messages is the **Conventional Commits**[7] Standard.

Another useful set of tools are **Continuous Integration (CI)** frameworks. These frameworks allow the developers to automate testing when pushing to a versioning system repository, which saves time and motivates developers to push code only when it passes all unit tests. CI works really well with a Test Driven Development Strategy.

[7]https://www.conventionalcommits.org/en/v1.0.0/

6 Effort Spent

Dario Passarello

- Initial discussion and objectives definitions: 4h
- Component Diagram, Component View Section: 5h
- Sequence Diagrams, Runtime View Section: 5h
- Style Architecture: 3h
- Requirements traceability: 2h
- Testing, Integration and Implementation Plan: 5h
- Final Review: 1h

Total: 25h

Davide Luca Merli

- Initial discussion and objectives definitions: 4h
- Setup DD Repository: 0.5h
- First chapter: 2h
- Overview: 1h
- Interfaces: 6h
- Deployment View: 1.5h
- Design Adjustments of the whole document: 6h
- Final Review: 5h

Total: 26h