M.Sc. Computer Science and Engineering
Software Engineering 2 Project

CLup - Customers Line-Up

Design Document

Leoni Luca, Locarno Silvia, Minotti Luca

10th January 2021

Version 1.0

GitHub Repository: https://github.com/lucagrammer/LeoniLocarnoMinotti

Contents

Chapter 1
1 Introduction

1.1 Purpose

This document constitutes the Design Document (DD). It provides a more technical overview of the Requirement Analysis and Specification Document (RASD) of the system-to-be, describing the main architectural components, their communication interfaces and their interactions.
It will also present the implementation, integration and testing plan. This type of document is mainly addressed to developers since it provides a guide during the development process through an accurate vision of all parts of the software-to-be.

1.2 Scope

As explained in the RASD, CLup aims to manage the queues to access stores in the Coronavirus era in a simple but effective way. Avoiding dangerous gatherings and unnecessary waste of time, CLup allows you to queue at a store directly from the application and receive a notification when it is time to reach the store, taking into account the time required to get to the shop using the indicated means of transport. Of course, fall-back options are available for people who don't have access to the required technology.
Moreover, the CLup Application also allows you to book a visit to a store in advance by indicating the date, time and product categories you are interested in. In addition, in case the desired slot is full, CLup provides useful suggestions for other slots available at the selected store or similar less crowded stores at the indicated date and time.
Furthermore, CLup can periodically notify you of the availability of slots in the day/time range in which you usually shop.
On the other hand, supermarket managers can easily keep access data under control and effortlessly ensure compliance with safety regulations.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

- **CLup System (or "The System")**: refers to the whole system to be developed.
- **CLup Services (or "Services")**: refers to the functionalities offered by the CLup System, such as the queue management mechanism and the booking service.

- **CLup Application (or "The Application")**: refers to the application that makes CLup Services available everywhere.
- **QR Code**: quick response code, a type of matrix barcode.
- **Reservation ID**: a code that unequivocally represents either a position in the queue or a slot reservation.
- **Physical Ticket Dispenser**: a computer connected to the CLup System that distributes paper tickets. It acts as a proxy for Guests.
- **Guest**: a person who has not access to the CLup Application but still uses the CLup Service to access stores through Physical Ticket Dispensers.
- **Customer**: a person that uses the CLup Application and its services to access stores and book visits.
- **Store Client**: either a Customer or a Guest.

1.3.2 Acronyms

- **RASD**: Requirement Analysis and Specification Document.
- **DD**: Design Document
- **UML**: Unified Modelling Language.
- **API**: Application Programming Interface.
- **PTD**: Physical Ticket Dispenser.
- **GPS**: Global Positioning System.
- **API**: Application Programming Interface.
- **ETA**: Estimated Time of Arrival

1.3.3 Abbreviations

- **[R.i]**: i-th requirement.
- **[C.i]**: t-th component.

1.1 Revision History

Version	Date	Authors	Summary
1.0	20/12/2020	Leoni Luca Locarno Silvia Minotti Luca	First release

1.2 Reference Documents

- Specification document: Project Assignment A.Y. 2020-2021.pdf
- RASD of CLup
- Software Engineering 2 course slides
- Previous project examples:

- Specification document: Project Assignment A.Y. 2019-2020.pdf
- DD to be analyzed.pdf

- IEEE Standard on Requirement Engineering (ISO/IEC/IEEE 29148)

1.5 Document Structure

This document is structured as follows:

1. ***Introduction*** - A general introduction of the system-to-be. It aims at giving general, but exhaustive, information about what this document is going to explain.
2. ***Architectural Design*** **-** An overview of the high-level components and their interactions, with a focus on both static and dynamic view, helped by diagrams.
3. ***User Interface Design*** **-** A representation of how the User Interface will look like.
4. ***Requirements Traceability*** **-** An explanation about how the requirements defined in the RASD map to the design elements defined in this document.
5. ***Implementation, Integration and Test Plan*** **-** Identification of the order in which the sub-components of the system should be implemented, integrated and tested.
6. ***Effort spent*** - Effort spent by all team members shown as the list of all the activities done during the realization of this document.
7. ***References*** - References to documents that this project was developed upon.

Chapter 2
2 Architectural Design

2.1 Overview

The figure below represents a high-level description of the main components which constitute the System. They are organized in a 4-tier architecture thus facilitating maintainability and scalability. Further details about the architectural choices can be found in sections 2.6 and 2.7.

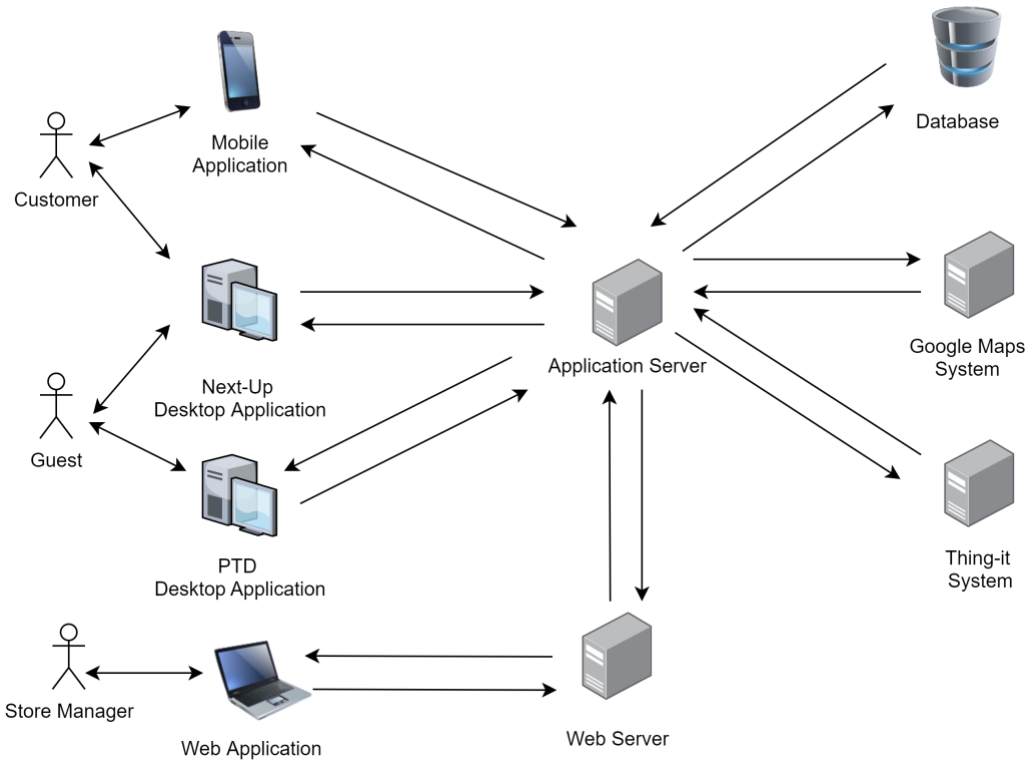


Figure 1: Overall architecture of the system

The main components of the Systems are the following:

- **Web Application**
  A web application accessible through the Store Manager's browser that allows him to access CLup Services. The web app will work with the most modern internet browsers, which communicate with the System by sending requests to the *Web Server*.

- **Customer Mobile Application**
  An application installed on the Customer's smartphone that allows him to use CLup Services anywhere. It communicates with the System forwarding every request directly to the *Application Server*. The mobile application will be available for both iOS and Android devices.

- **PTD Desktop Application**
  An application installed on a touchscreen device located near the entrance to each store that allows Guests to use the "Line Up" Service. It communicates with the System forwarding every request directly to the *Application Server*. This application will be available for Windows devices.

- **Next Up Desktop Application**
  An application installed on a device located near the entrances of each store that allows both Customer and Guests to know which Reservation IDs are allowed to access the store. It communicates with the System through the *Application Server*. This application will be available for Windows devices.

- **Web Server**
  It is the back-end component of the *Web Application* that communicates with the Store Manager's browser on one hand and with the components of the *Application Server* on the other (i.e. the business logic tier).

- **Application Server**
  It is the main back-end component of the System on which the business logic of the application takes place: it elaborates the requests coming from the end-user applications and it interacts with the data layer and the web layer. It also communicates with the various *External Systems*.

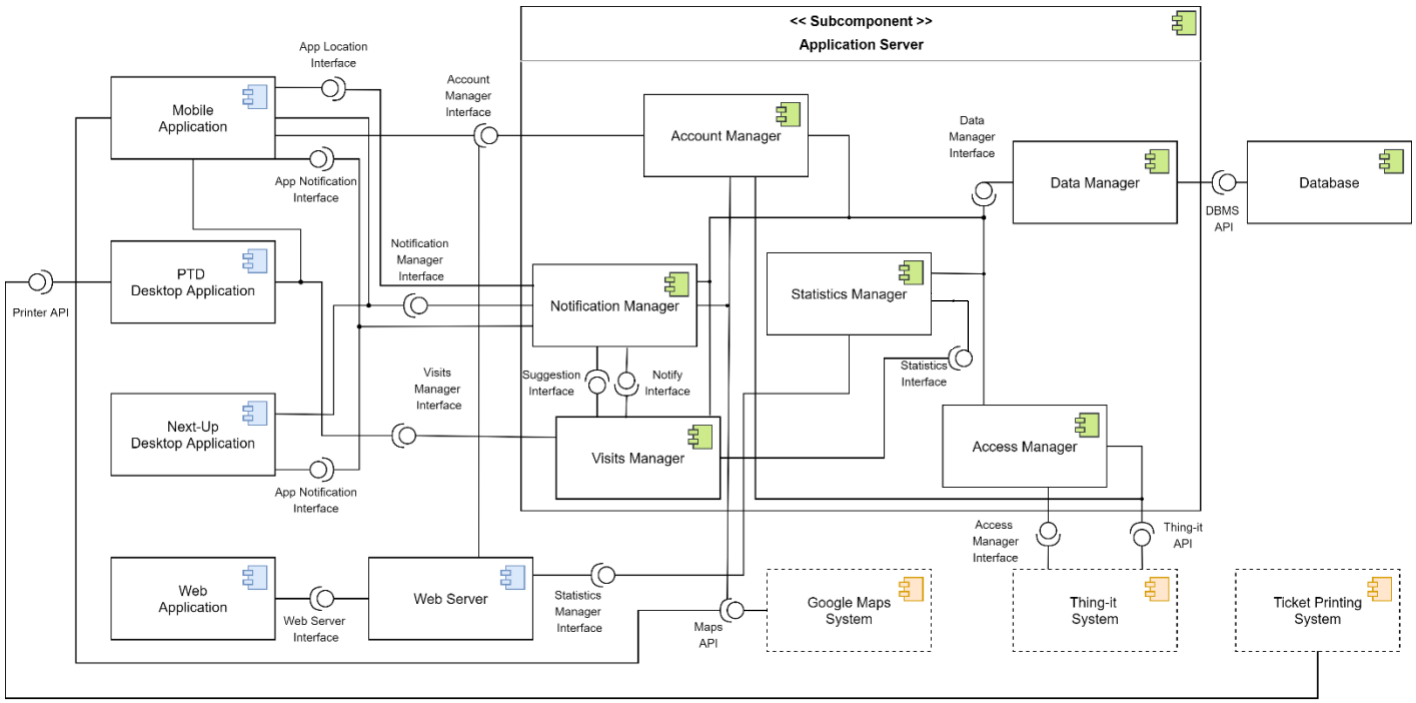
- **Database**
  It is the component responsible for data storage. It can only be accessed by the *Application Server*.

- **External Systems**
  These are systems that provide functionalities that are not internally developed. *External Systems* interact mainly with the *Application Server*.

- **Google Maps System**: this system is responsible for providing the Map Services necessary for computing the ETA and supporting the identification of stores close to a Customer.
- **Thing-it System**: this Smart-Office System allows CLup to remotely control the sliding doors of the store and communicate with the QR code readers.
- **Ticket Printing System**: this system is responsible for printing paper tickets. It's directly connected to the device running the *PTD Desktop Application.*

2.2 Component View

In this section, every high-level component is analysed in terms of its subcomponents. External Systems, such as *Google Maps System, Thing-it System* and *Ticket Printing System*, are presented as black-boxes that expose only the interfaces used by CLup. Further details about the component interfaces can be found in section 2.5.



Component Diagram 1

2.2.1 Application Component

The *Application Component* is the front-end of the System. Its purpose is to render the graphical interfaces (detailed in section 3) that allow the end-user to take advantage of the CLup Services.

The only logic incorporated in the *Application Component* is the ability to perform basic checks, such as the detection of incomplete forms. Indeed, all the computational tasks are carried out by the *Application Server* and, for this reason, the *Application Component* must be able to continuously communicate with it.
As shown in the figure above, the *Application Component* consists of the following sub-elements, already presented in section 2.1:

- **Customer Mobile Application**
  It is the application dedicated to Customers that allows them to line up directly from their smartphone, book visits, access their QR Codes and receive periodic notifications.

- **Next-Up Desktop Application**
  It is the application that runs on a device whose screen is visible from outside each store. It shows which Reservation IDs are allowed to access that store at any given time. Both Guests and Customers can verify whether it is their turn to enter the store via this application.

- **PTD Desktop Application**
  It is the application dedicated to Guests that allows them to line up directly from the store and receive a paper ticket.

- **Web Application**
  It is the application dedicated to Store Managers that allows them to keep access data under control and possibly modify the store information.

2.2.2 Web Server Component

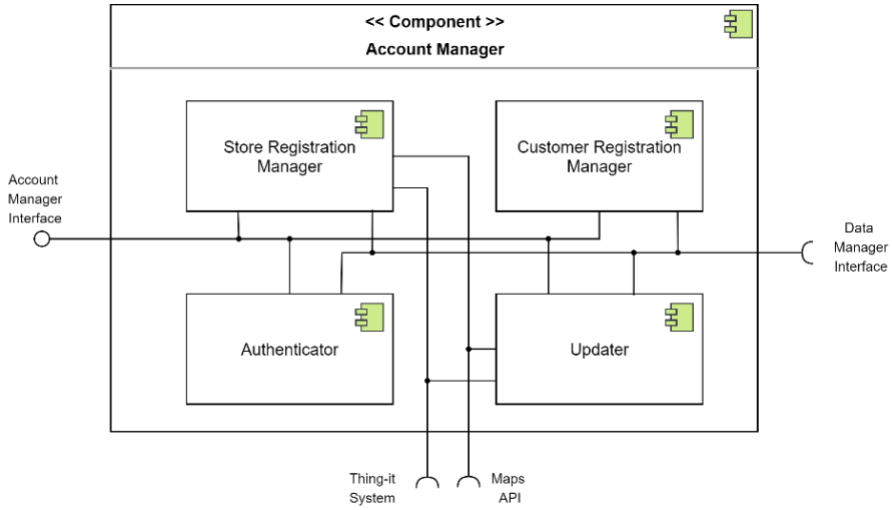
A *Web Server* is required in order to provide the *Web Application* for the Store Managers. This component receives HTTPS requests from the Store Manager's browser and forwards them to the *Application Server* to collect all the data required to generate the dynamic web pages.

2.2.3 Application Server Component

The *Application Server* realizes the business logic, its role is to compute all the needed data and coordinate the flow of information between the application layer and the data layer.
As shown in the component view, the *Application Server Component* consists of the following sub-elements:

- **Account Manager**

  It handles all the operations related to accounts of Store Managers and Customers. It communicates with the *Data Manager* in order to verify, access, store and delete account information. Indeed, this component is responsible for account creation and for the authentication process.



Sub-Component Diagram 1: Account Manager

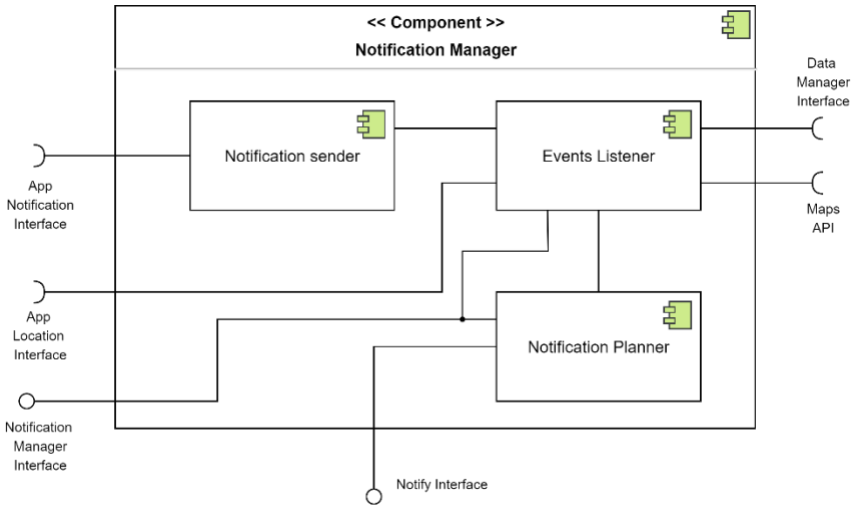
It consists of the following subcomponents:

- **Store Registration Manager***:* it manages the registration of new stores and, in particular, it checks that the indicated Store ID is not already taken. It also interacts with the *Google Maps System* to obtain the coordinates of a store from its address. Furthermore, it interacts with the *Thing-it System* and acquires control of the doors of the newly registered store.
- **Customer Registration Manager***:* it manages the registration of new Customers.
- **Authenticator**: this subcomponent handles the authentication by checking the credentials.
- **Updater:** it is responsible for account data updates and account deletion.

- **Notification Manager**

  It handles notifications to the Customers and the sending of queue updates to the *Next-Up Desktop Application*. Specifically, it exposes an interface to the *Visit Manager* to allow the registration of new events that will trigger the sending of notifications. To manage these events, the

component needs to interface with the *Data Manager* and the *Google Maps System.* An additional interface must be exposed to allow receipt of location updates from the *Mobile Application* and to allow the *Next-Up Desktop Application* to function properly.



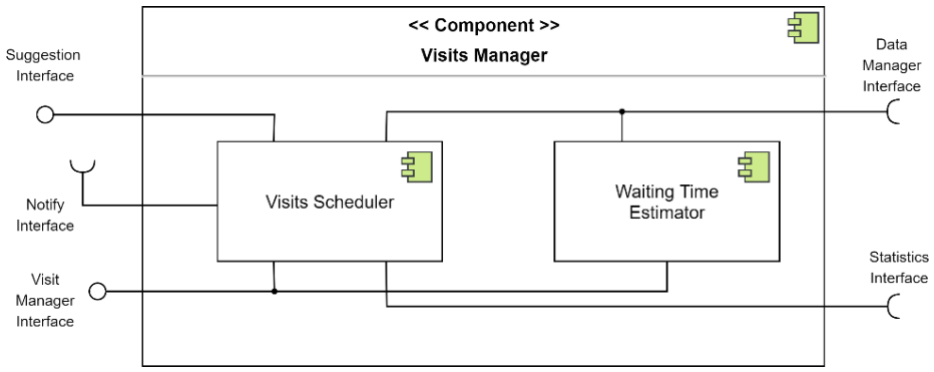
Sub-Component Diagram 2: Notification Manager

It consists of the following subcomponents:

- **Notification Planner***:* it manages the registration of the events that will trigger the sending of notifications.
- **Notification Sender:** it takes care of actually sending notifications to the *Mobile Application* and to the *Next-Up Desktop Application.*
- **Events Listener:** it takes care of detecting the occurrence of the events that trigger the sending. It also manages the functionality of periodic notifications.

- **Visits Manager**

  It manages all the operations related to visit-scheduling. First, it exposes an interface to the *Mobile Application* and the *PTD Desktop Application* to allow the receipt of requests from them. Moreover, it communicates with the *Data Manager* in order to access, store and delete visit information. This component also interacts with the *Notification Manager* to provide the suggestions contained in periodic notifications but also to schedule alerts to the Customer.



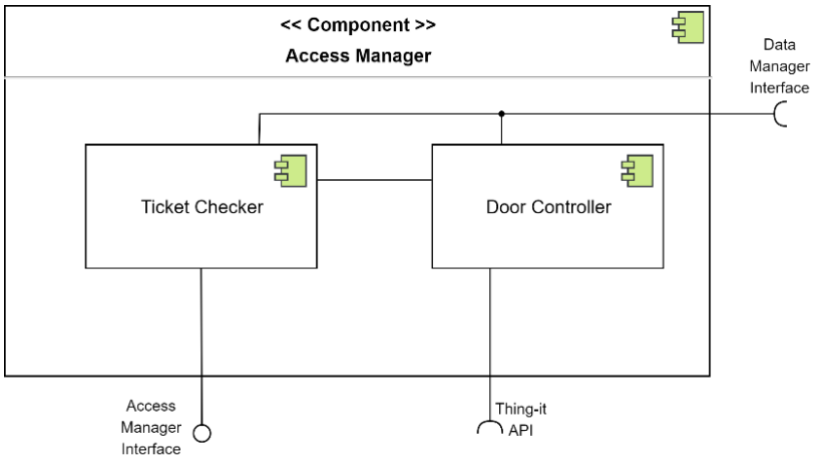
Sub-Component Diagram 3: Visits Manager

It consists of the following subcomponents:

- **Visit Scheduler:** it plans all the visits possibly assigning them an expected duration calculated by the *Statistics Manager*. It is also responsible for computing the visit suggestions.
- **Waiting Time Estimator:** it computes the estimated waiting time based on the scheduled visits stored in the database

- **Statistics Manager**
  It is responsible for computing the access statistics based on the data stored in the database, which is accessed through the *Data Manager.*

- **Access Manager**
  It manages all the access requests to the stores. In particular, it receives the data coming from the QR Code readers and verifies whether the ticket is allowed to access or not. Depending on this, it interacts again with the *Thing-it System* to control the status of the sliding doors.



Sub-Component Diagram 4: Access Manager

It consists of the following subcomponents:

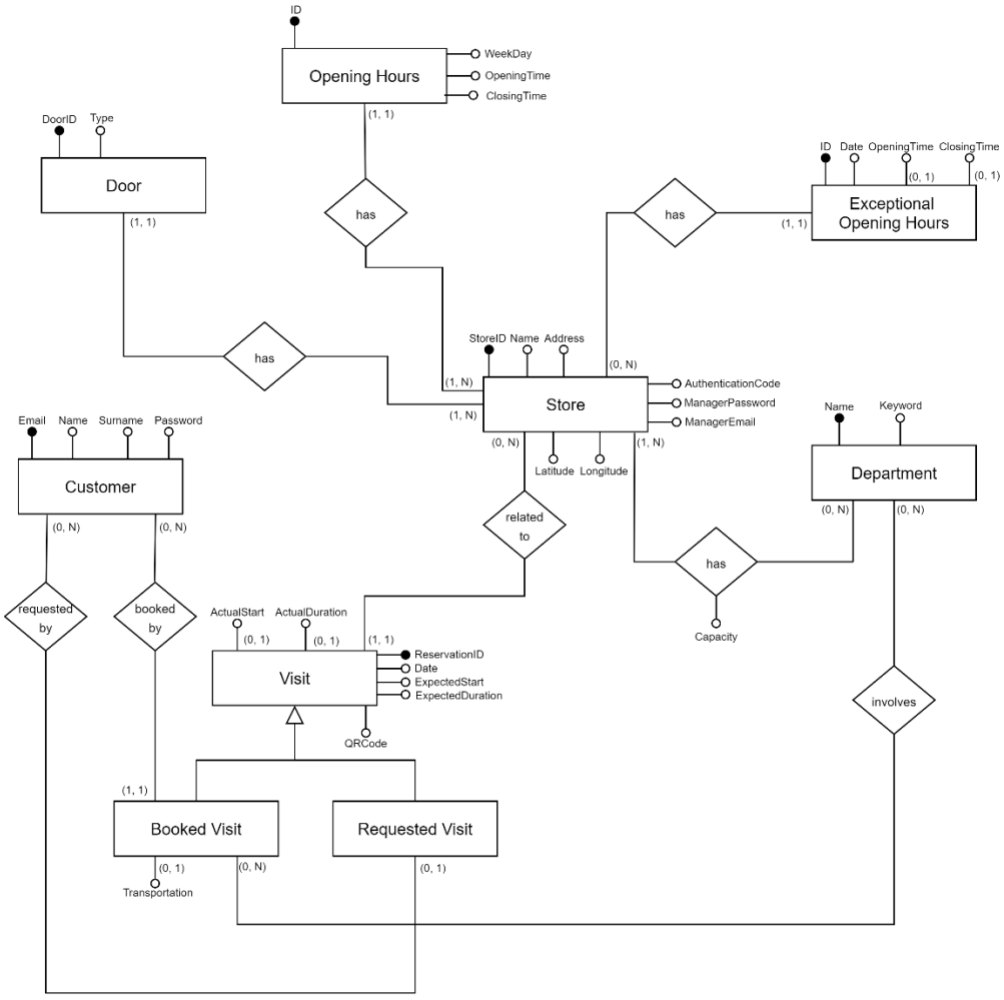
- **Ticket Checker**: it checks whether the read QR Code is associated with a visit scheduled for the current date and time.
- **Door Controller**: it manages the requests to the Smart-Office System for opening and closing the doors.

- **Data Manager**

  It is the only component that interacts with the data layer. Indeed, it receives requests coming from the other components for reading, storing and deleting data from the database.

2.2.4 Data Components

The data layer is composed of a single relational database, which is managed by a DBMS that optimizes and performs the queries. The image below represents the Entity-Relationship (ER) diagram of the database.



CLup Entity-Relationship Diagram

Relevant aspects of the ER schema are the following:

- The *Visit* entity refers to all the scheduled visits, both those booked in advance and those requested through the "Line Up" Service. Both are assigned a Reservation ID, a QR Code and a time slot, possibly estimated by the System. The actual entry and exit time are also stored to allow the computation of statistics. Finally, any information relating to the product categories, in which the Customer is interested, is also stored. If this information has not been specified, the System will assume during the computation that all departments are visited by that Store Client. Note that the visits requested by the Guests will not be connected to any Customer.
- The *Store* entity refers to the stores registered to CLup. For each of them, data relating to departments and opening hours (regular and exceptional) are saved. In order to allow CLup to retrieve the stores near the Customer, the coordinates are also stored. Finally, information on shop doors is saved. This data, provided by the *Thing-it System*, includes a serial code and the type, which defines whether the gate is an exit or an entrance.
  The authentication code, provided by the Store Manager during the registration, is a password required by the *Thing-it* external *System* to grant CLup remote control of the store's devices (further details will be provided in subsection 2.2.5).

From the previous ER schema, it is possible to derive the following logical model:

Customers (Email, Name, Surname, Password)
Store (StoreID, Name, Address, Latitude, Longitude, ManagerEmail, ManagerPassword, AuthCode)
Doors (StoreID, DoorID, Type)
OpeningHours (ID, WeekDay, OpeningTime, ClosingTime, StoreID)
ExceptionalOpeningHours (ID, Date, OpeningTime*, ClosingTime*, StoreID)
Departments (Name, Keywords)
StoreDepartments (StoreID, DepartmentName, Capacity)
Visits (ReservationID, QrCode, Date, ExpectedStart, ExpectedDuration, ActualStart*, ActualDuration*, Transportation*, StoreID, CustomerEmail*)
BookedDepartments (DepartmentName, ReservationID)

2.2.5 External Systems

- Google Maps System

  This external system communicates with the *Application Server* and the *Mobile Application* via APIs. In particular, it provides an interface to insert

a map view, to translate an address into geographical coordinates and to calculate the ETA to reach a store.

- **Thing-it System**

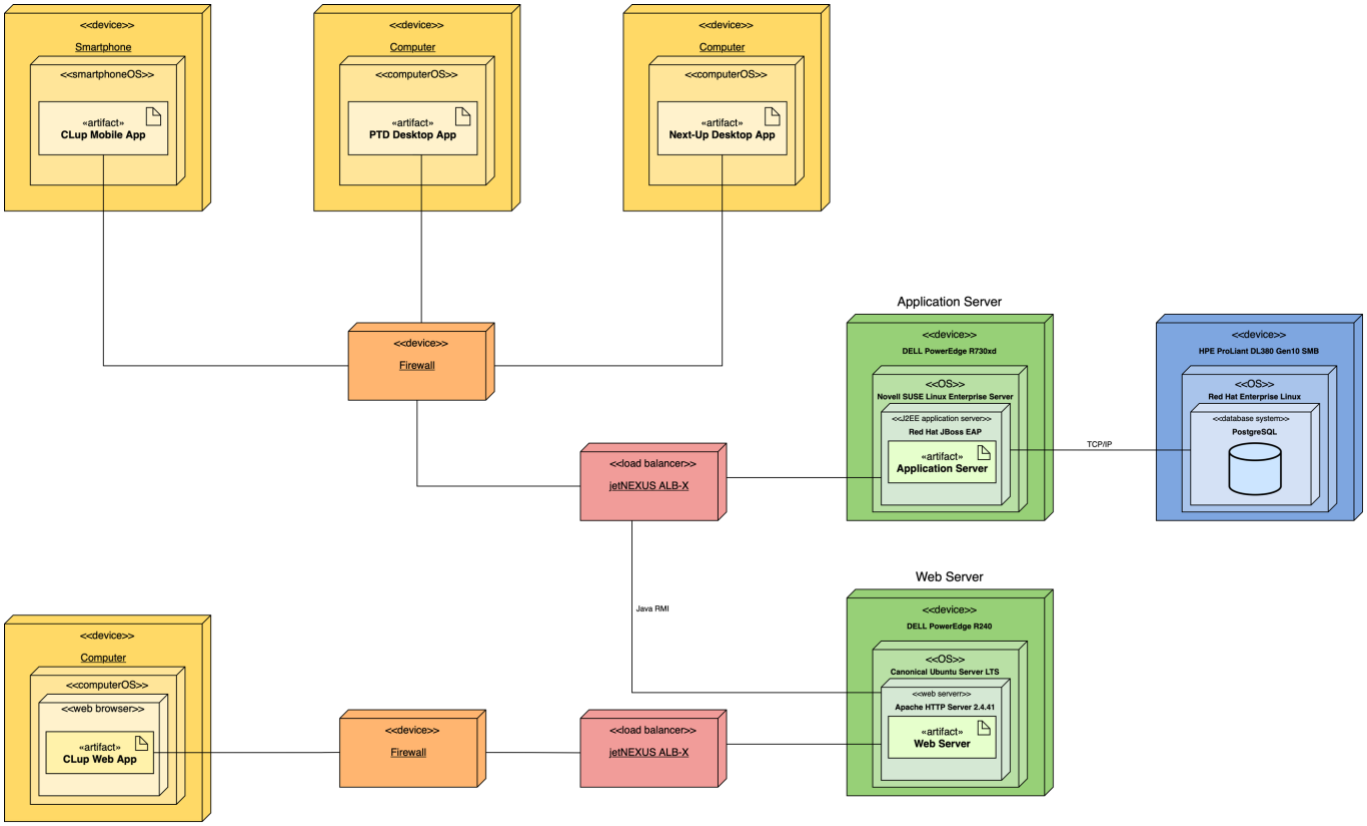
  This external system communicates with the *Application Server* via APIs. In order to use these APIs and thus being able to remotely control doors and QR Code readers of a store, an authentication code is required. This code, provided by the Store Manager during the registration, allows CLup to take control of the devices. This external system also provides APIs to retrieve information on gates and manage them.

- **Ticket Printing System**

  This external system exposes APIs that are used directly by the *PTD Desktop Application* for printing paper tickets.

2.3 Deployment View

This section describes the configuration of the processing nodes and the components that live on them. In the figure below there is a representation of the Deployment diagram for the entire system.



Deployment Diagram

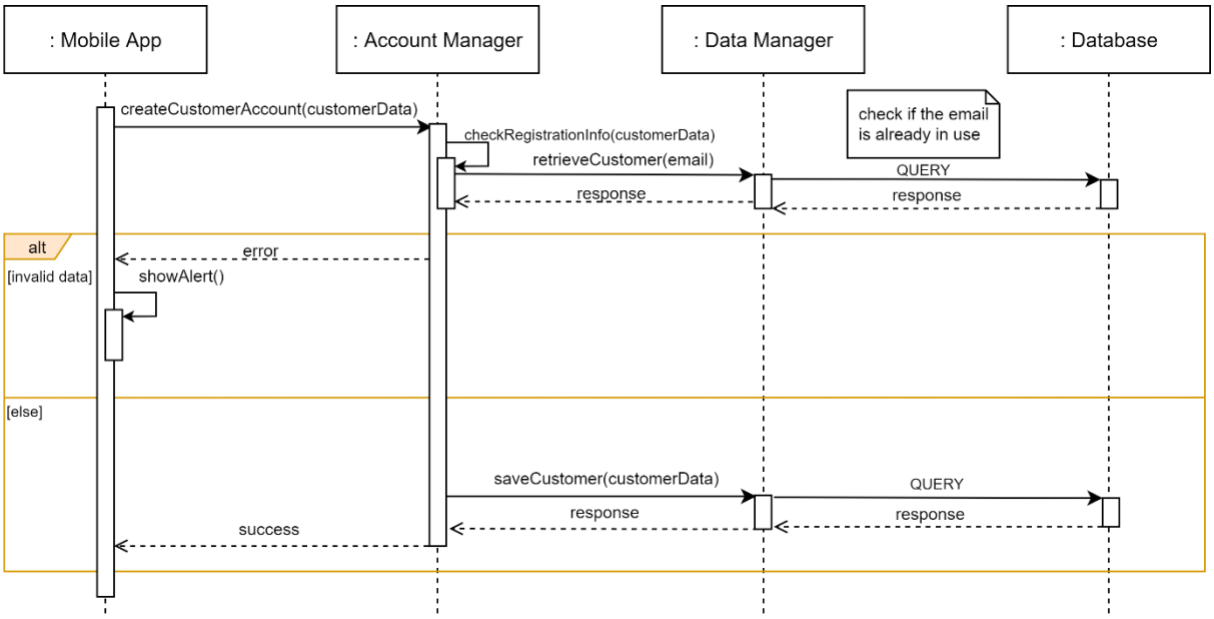
As shown in the above image, *firewalls* and *load balancers* manage the data stream from the devices to the servers. First of all, *firewalls* are in charge of filtering packets received from the Internet. Then, the packets pass through the *load balancer*, where the workload is distributed among the available resources to increase capacity and reliability. Moreover, the *Application Server*, which handles the business logic rules, communicates with the *Web Server* through public interfaces defined in the following paragraphs, while the communication between each *J2EE Application* instance and the *Database Server* is made possible through *PostgreSQL* public API of the database system.

2.4 Runtime View

In this section, the dynamic behaviour of the System is described by showing the major interaction processes between the various components. Further details about the component interfaces can be found in section 2.5.

- **Customer Registration**

In the diagram below is presented the workflow that is executed during the registration of a Customer through the *Mobile Application*. Once the form has been submitted, the application performs a basic check to verify that the required fields have been filled in. If so, it proceeds by forwarding the account creation request to the *Application Server*. Here the *Account Manager* checks the data entered by the Customer and in particular verifies if the email is not already in use. If the checks pass, it proceeds by storing this data.

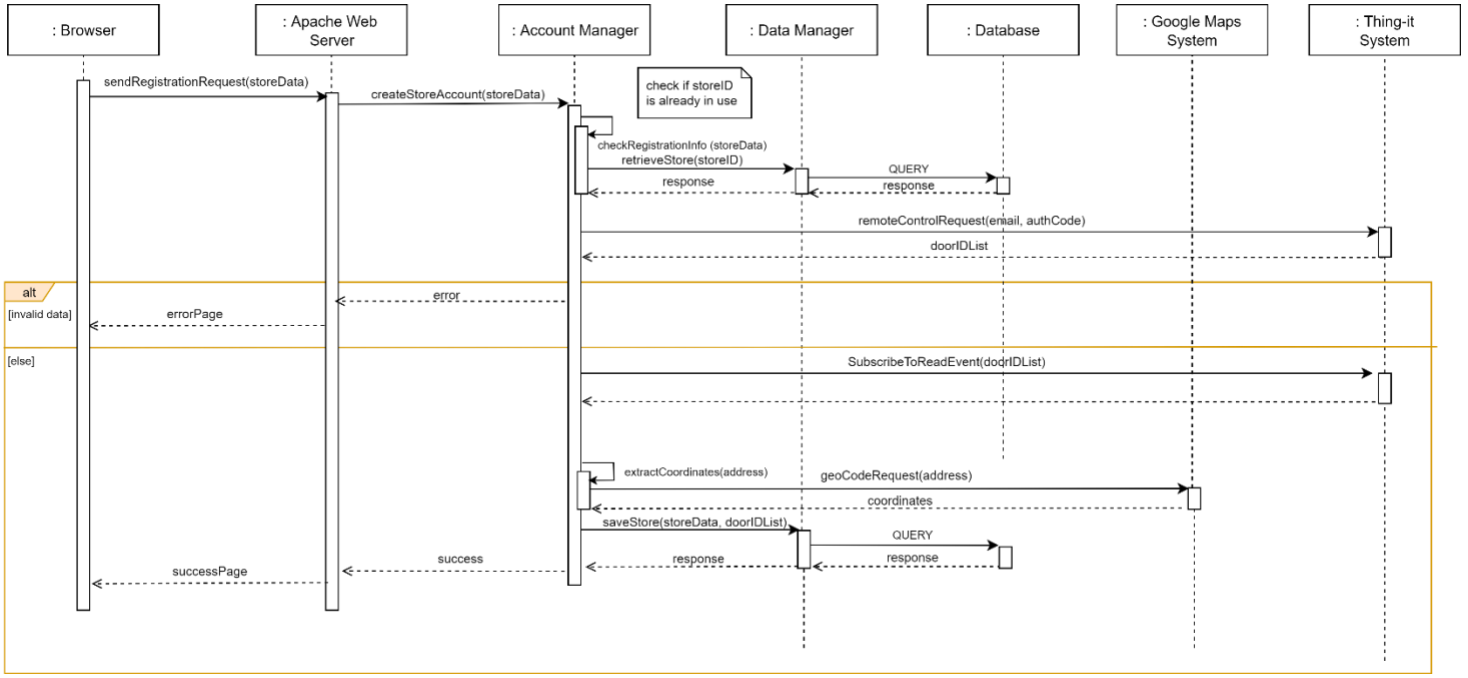


Runtime View 1: Customer Registration

- **Store Manager Registration**

The following diagram represents the workstream that occurs during the registration of a Store Manager through the *Web Application*. Once the form has been submitted, the *Browser* proceeds by forwarding the registration request to the *Web Server*. Here the call to the *Account Manager* takes place. The latter verifies the entered data and in particular checks if the chosen StoreID is not already in use. If the checks pass, it proceeds by connecting to the *Thing-It System* using the authentication code. If successful, the *Account Manager* proceeds by subscribing to events relating to the

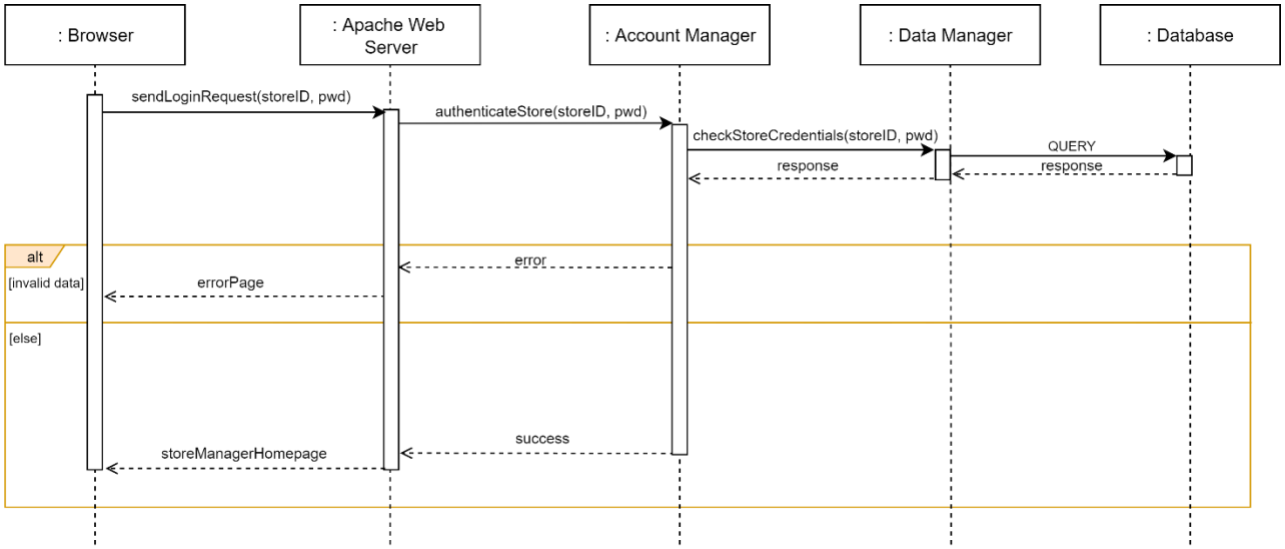
QR code readers. From now on, when a QR code is read by the scanner, CLup will be notified of it. Finally, all the needed data are stored in the database. These data include those provided by *Thing-it* relating to the gates (serial number and type).



Runtime View 2: Store Manager Registration

- **Store Manager Login**

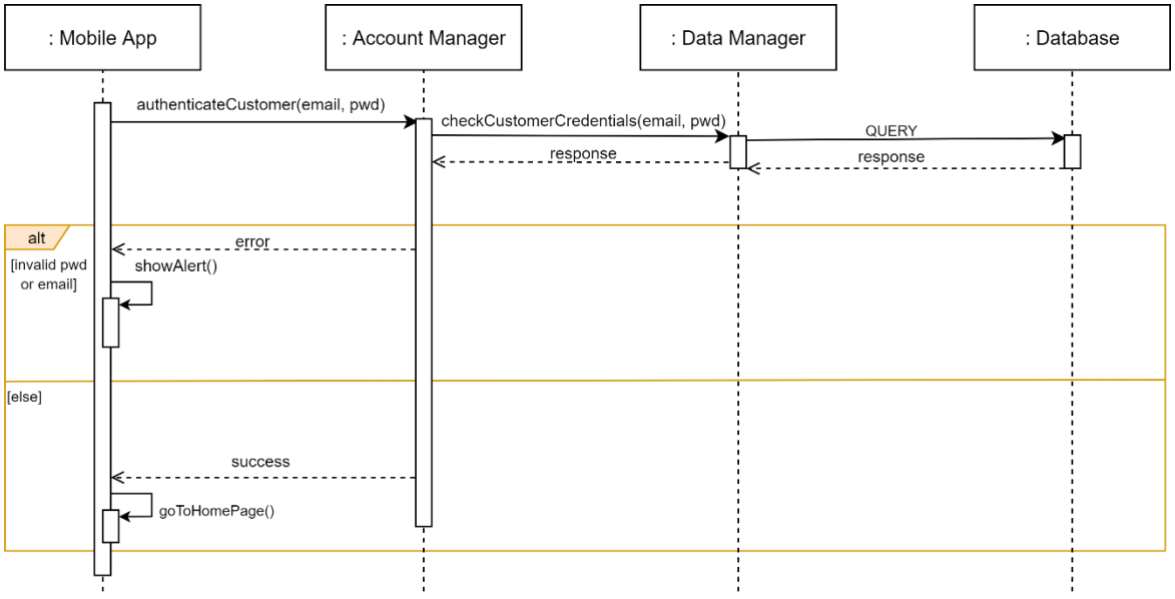
In the sequence diagram below is detailed the workflow that occurs during the login of the Store Manager. Once the form has been submitted, the *Browser* sends the request to the *Web Server* which forwards it to the *Account Manager*. This component verifies the credentials by interacting with the *Data Manager*. If correct, the *Web Server* redirects the Store Manager to the homepage otherwise it shows an error page.



Runtime View 3: Store Manager Login

- **Store Manager Login**

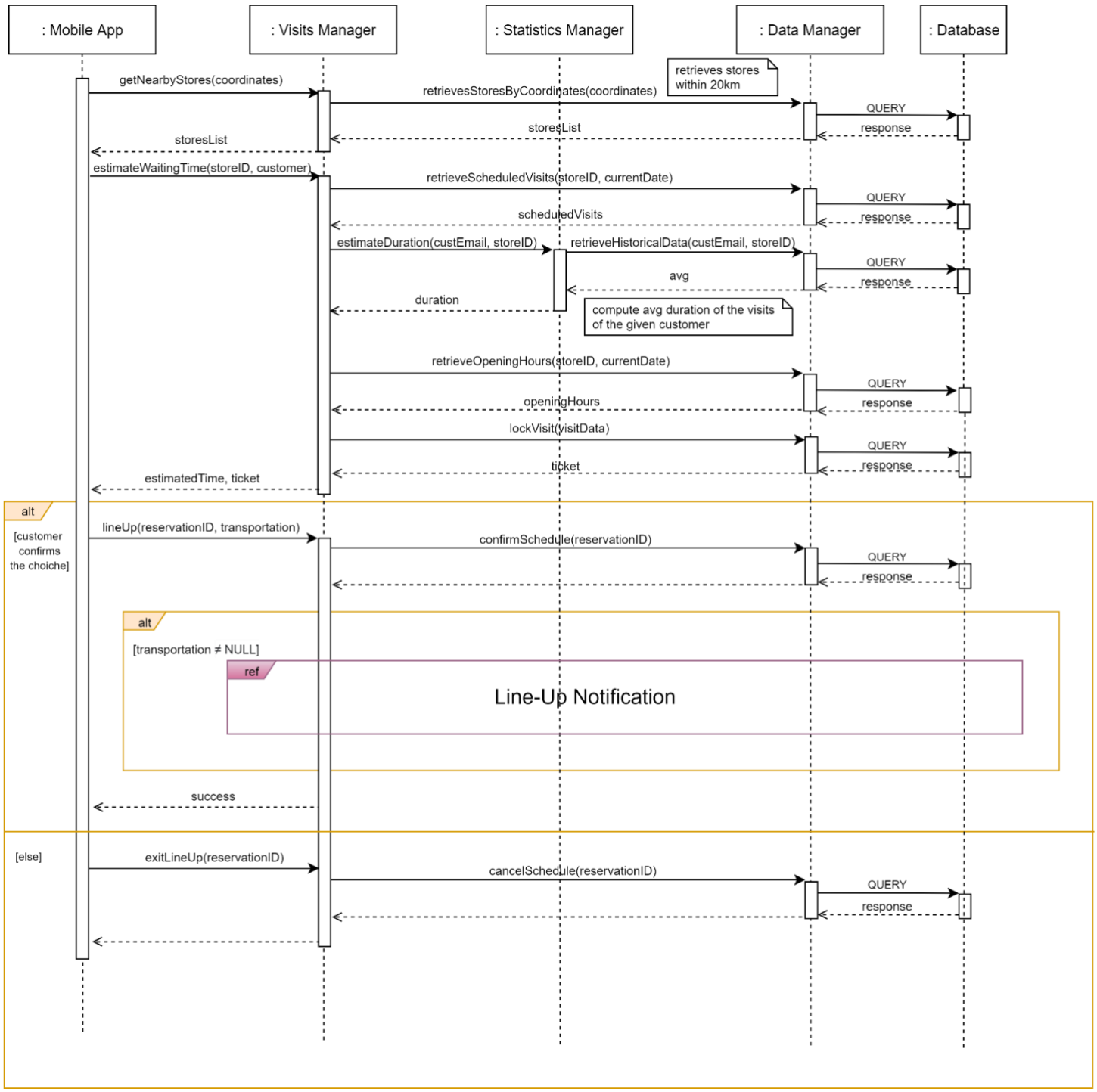
In the image below is presented the workflow that is executed during the login of a Customer through the *Mobile Application*. Once the login form has been submitted, the application performs a basic check to verify that the fields have been filled in. If so, it proceeds by forwarding the request to the *Application Server*. Here the *Account Manager* checks the credentials and returns the result.



Runtime View 4: Customer Login

- **Customer Line-Up**

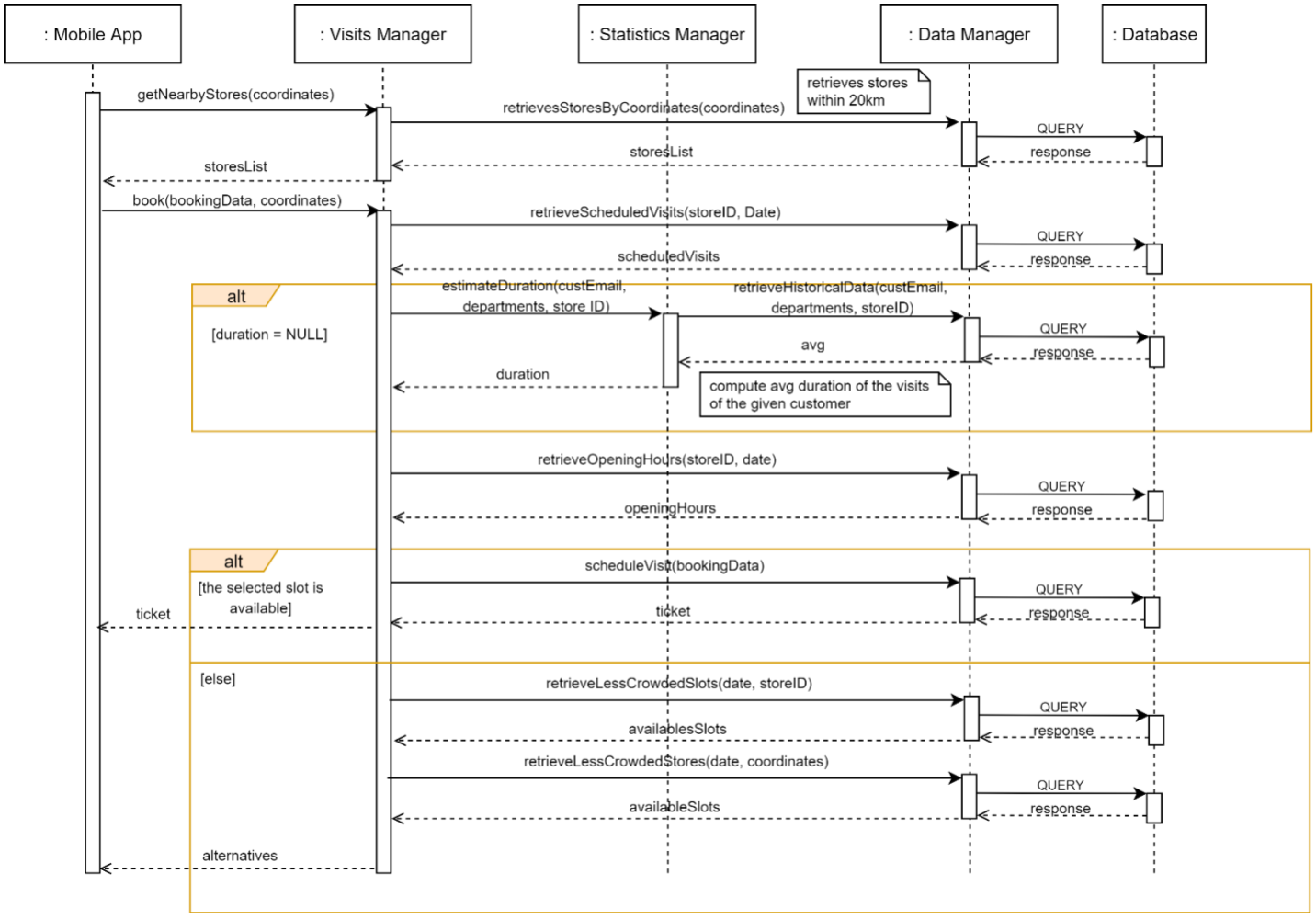
The following diagram represents the workstream that occurs when a Customer tries to line up at a store from the *Mobile Application*. When the Customer opens the line-up interface, the application retrieves the nearby stores by forwarding a request to the *Visits Manager*. Once the store has been selected, the application asks the *Application Server* to estimate the waiting time. The *Visit Manager* then makes the estimation. First, it retrieves the data of the already scheduled visits and the opening hours of the store, then estimates the average duration of the Customer's visit through the *Statistic Manager*. This component analyses the historical data relating to the Customer. Once the estimation has been computed, a visit to the shop is provisionally reserved for the Customer. If the Customer decides to proceed, the visit is effectively reserved, otherwise the reservation is cancelled. If the Customer has also indicated the means of transport that he will use to reach the store, the process specified in the Runtime View 10 starts.



Runtime View 5: Customer Line-Up

- **Customer Books a Visit**

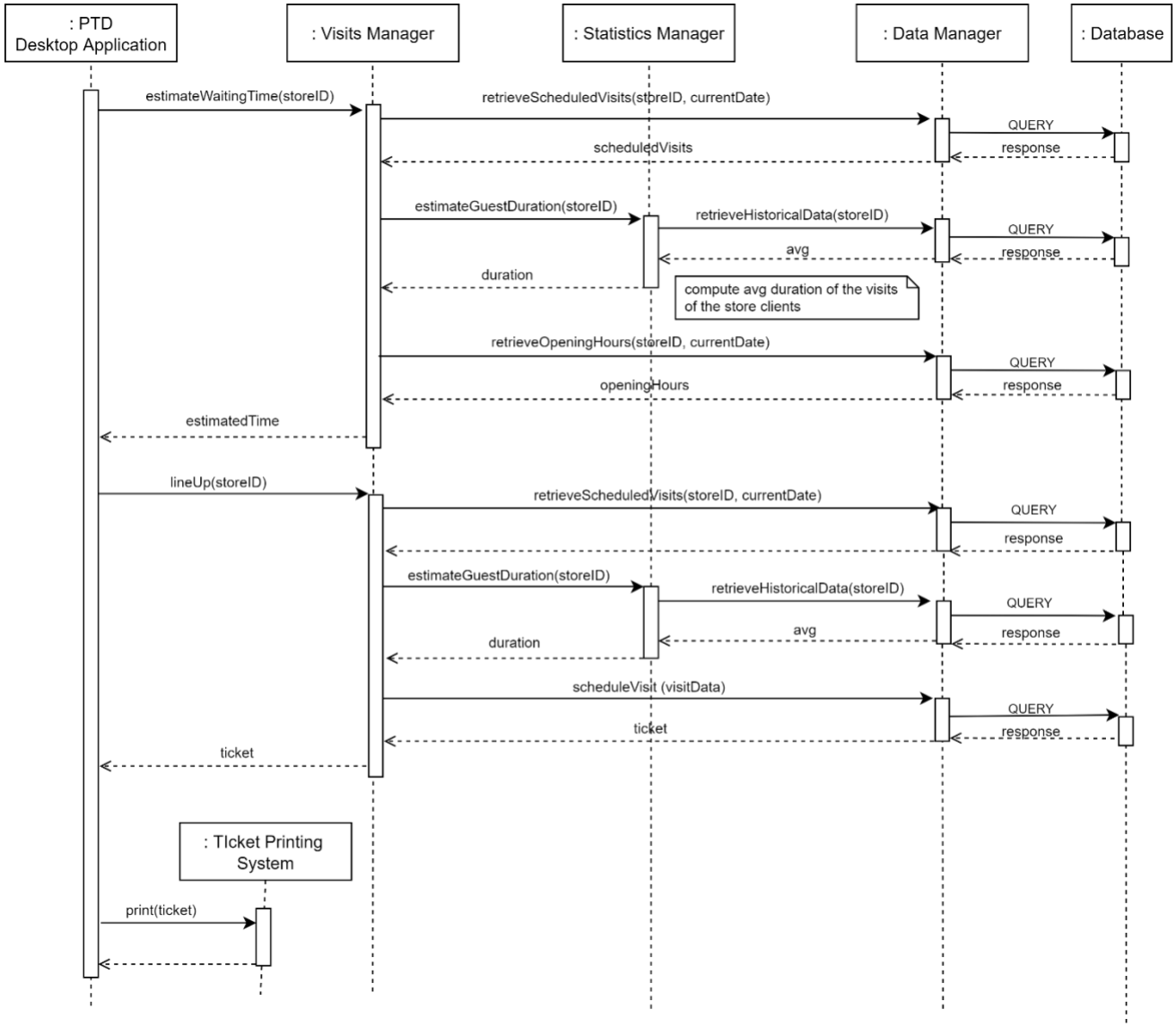
In the diagram below is presented the workflow that is executed during the booking of a visit through the *Mobile Application*. When the Customer opens the booking interface, the application retrieves the nearby stores by forwarding a request to the *Visits Manager*. Once the store and the slot have been selected, the application asks the *Application Server* to book the slot. The *Visit Manager* then checks the availability of the slot. First, it retrieves the data of the visits already scheduled and the opening hours of the store, then estimates the average duration of the Customer's visit through the *Statistic Manager*. This component analyses the historical data relating to the Customer. Once the estimation has been computed, a visit to the shop is booked for the Customer. If the slot is not available, the Visit Manager compute the available alternatives.



Runtime View 6: Customer Books a Visit

- **Guest Line-Up**

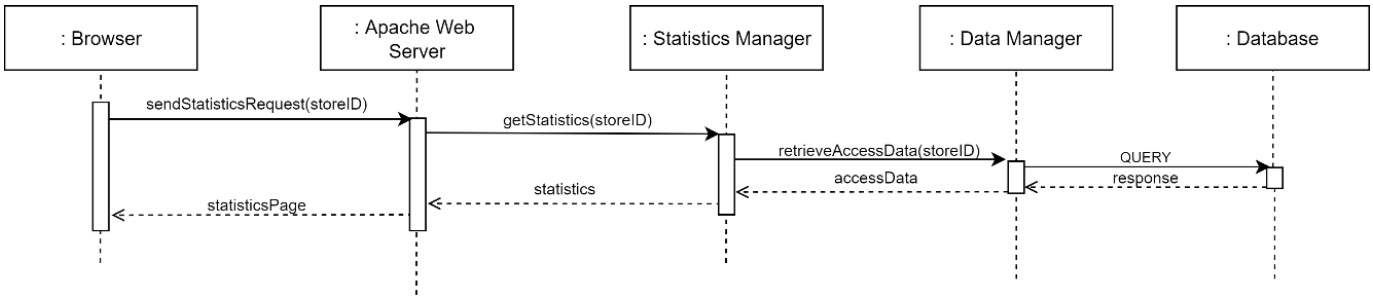
In the following image is presented the workflow that occurs during the line-up of a Guest through the *PTD Desktop Application*. When a Guest reaches the touchscreen monitor, the application contacts the *Application Server* to compute the estimated waiting time. This is done by the *Visit Manager*. First, it retrieves the visits already scheduled and the opening hours of the store, then estimates the average duration of the visits for that shop. When the Guest decides to queue, another request is sent to the *Visit Manager* that, after calculating the slot to be assigned, stores the assignment through the *Data Manager*. At this point the *PTD Desktop Application* orders the *Ticket Printing System* to print the paper ticket.



Runtime View 7: Guest Line-Up

- **Store Manager Analyses Access Data**

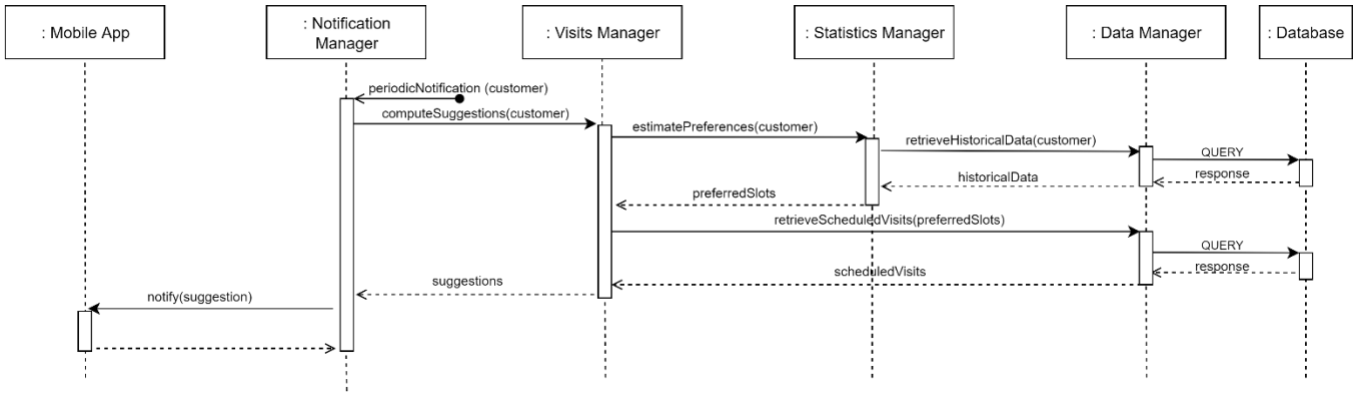
  In the diagram below is presented the workflow that is executed when the Store Manager wants to analyse the access data through the *Web Application*. The *Browser* sends a request to the *Web Server* to access the statistics report. The latter is generated by the *Statistics Manager*. Finally, the report is included in the web page that is returned to the *Browser.*



Runtime View 8: Store Manager Analyses Access Data

- **Periodic Notification**

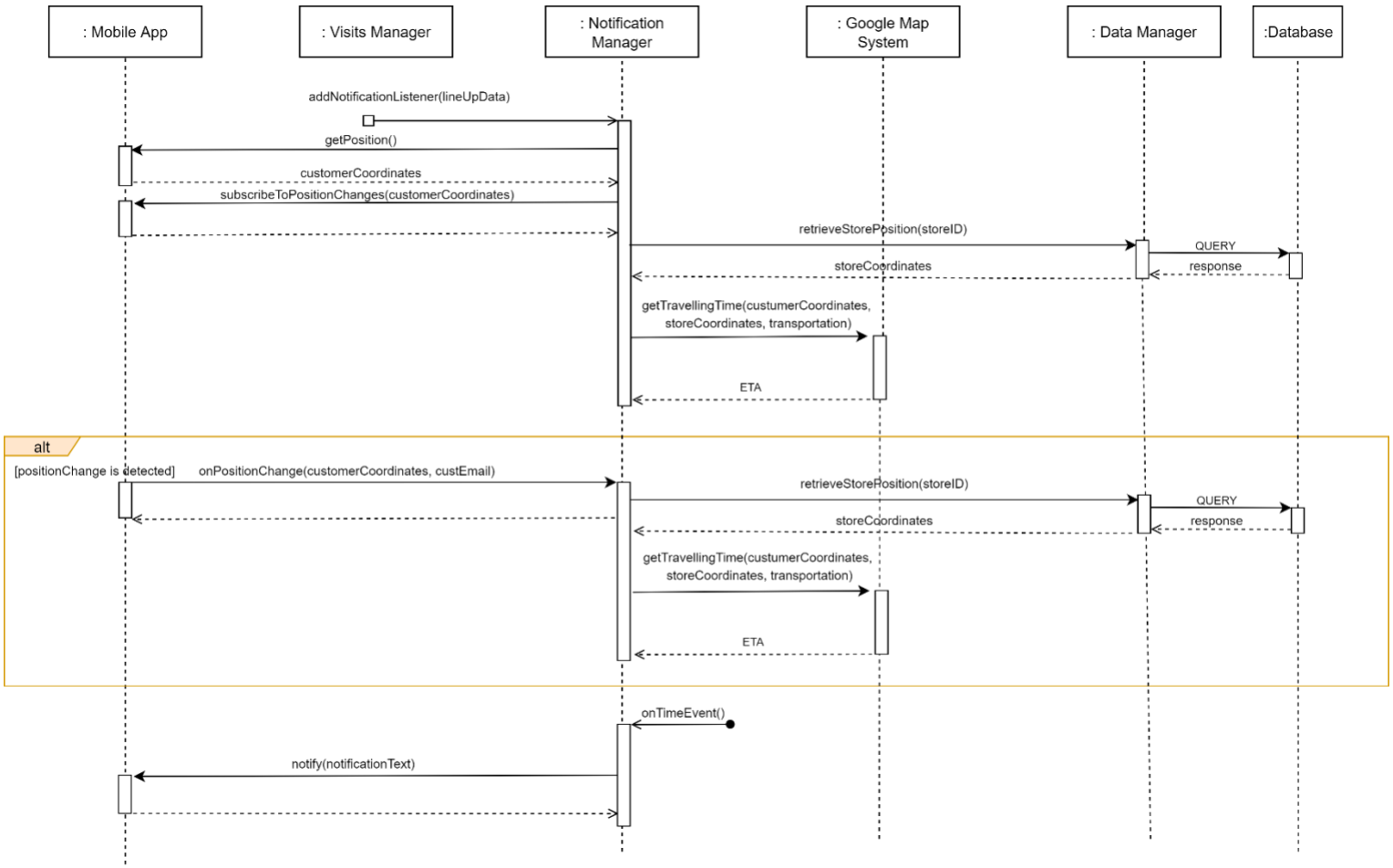
The sequence diagram below explains how the System works to periodically notify the Customer and provide him useful suggestions. Periodically, the *Notification Manager* is activated to check for the presence of available slots that could be of interest to a Customer. In order to do this, it interacts with the *Visits Manager* that, after computing the slots, checks their availability. Once the computation is complete, a notification is sent to the *Mobile Application*.



Runtime View 9: Periodic Notification

- **Line-Up Notification**

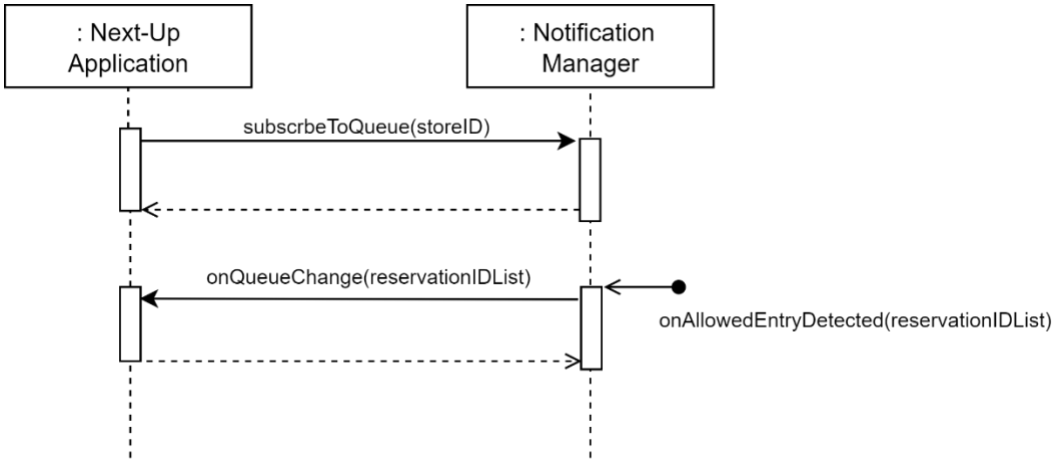
The following diagram explains how the System works to notify the Customer when it is time to reach the store. The *Visits Manager* delegates the notification management to the *Notification Manager*. Once retrieved the position of the Customer and that of the store, this component contacts the *Google Maps System* to estimate the departure time. If the *Mobile Application* detects that the Customer has moved, it communicates the new position to the server and the computation is repeated. When the *Notification Manager* detects that it is time to notify the Customer, it sends a notification to the *Mobile Application*.



Runtime View 10: Line-Up Notification

- **Update of "Next-Up" Interface**

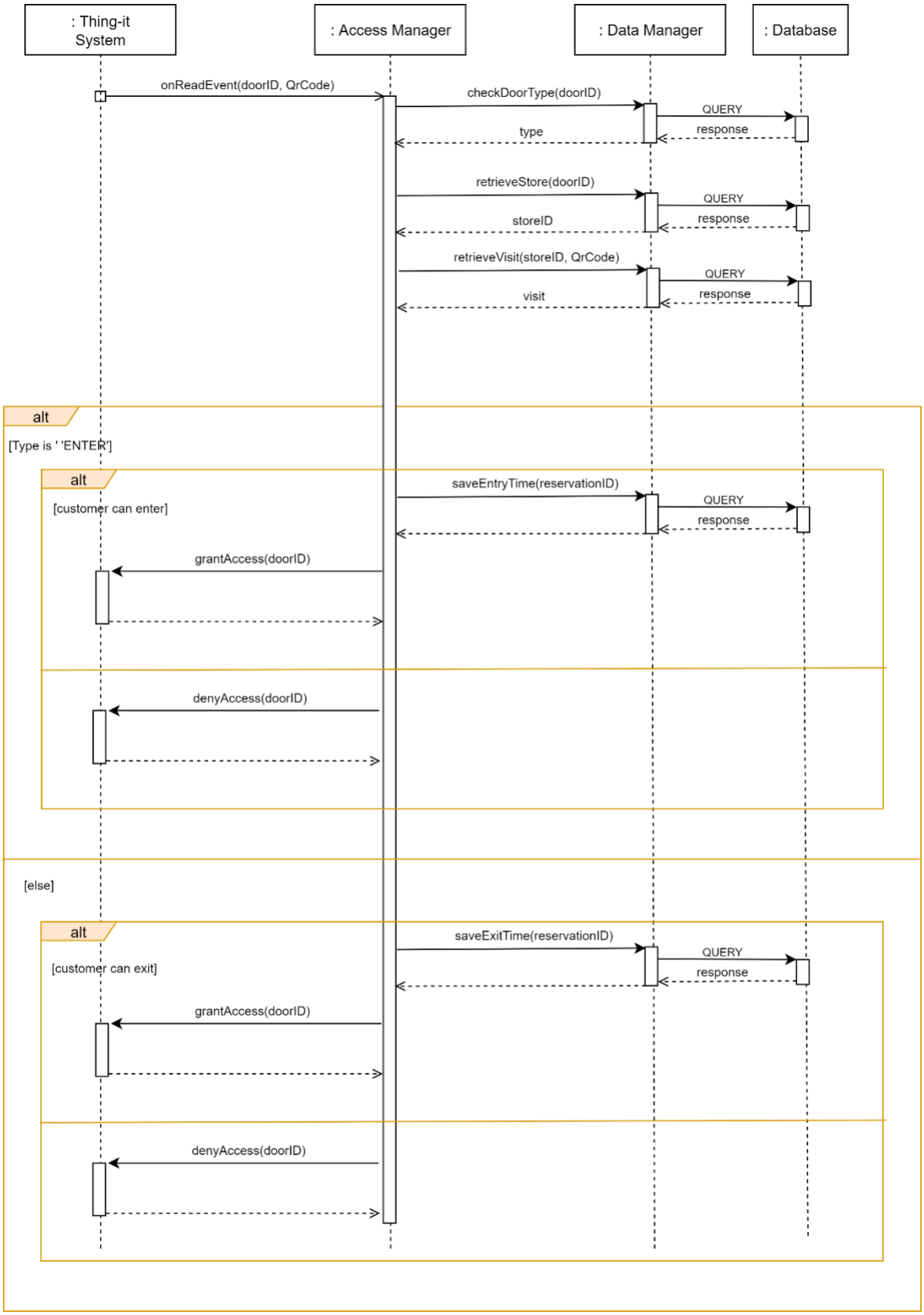
The following diagram explains how the System works to periodically update the Next-Up Interface that shows the Reservation IDs that are allowed to enter a given store. When the application is started, it contacts the *Application Server* to subscribe to updates related to visits for that store. In this way, once the *Notification Manager* has detected that other Customers can access the store, a notification is sent to the application and the new Reservations IDs appear on the interface.



Runtime View 11: Update of "Next-Up" Interface

- **Store Client Enters or Leaves a Store**

The runtime view below explains the workflow that occurs when a Customer or a Guest tries to enter or exit a store. When a code is scanned by the QR reader, the *Thing-It System* notifies the *Application Server* by forwarding the read data. Here, the *Access Manager* checks the data of the door from which the QR Code was read and then verifies if the code is assigned to a ticket of a scheduled visit to that store. In this case, it allows access and updates the statistics, otherwise it returns an error.



Runtime View 12: Customer/Guest Enters or Leaves a Store

2.5 Component interfaces

This section provides a high-level description of the methods offered by the interfaces that the various system components expose.

2.5.1 Data Manager Interfaces

The *Data Manager* component exposes a single external interface that is used by all the other *Application Server* components to interact with the database. Its main methods are the followings:

- **saveCustomer**(customerData)
  It stores the information of a new Customer account in the database. Throws an exception in case an error occurs.

- **saveStore**(storeData, doorsList)
  It stores the information of a new store account in the database. Throws an exception in case an error occurs.

- **checkStoreCredentials**(storeID, pwd)
  It verifies the correctness of the Store Manager login credentials passed as a parameter.

- **checkCustomerCredentials**(email, pwd)
  It verifies the correctness of the Customer login credentials passed as a parameter.

- **retrieveCustomer**(email)
  It returns the information of the Customer whose email is the specified one. If none is found, the null value is returned.

- **retrieveStore**(storeID)
  It returns the information of the store whose username is the specified one. If none is found, the null value is returned.

- **retrieveStoresByCoordinates**(coordinates)
  It returns the information of the stores located within a radius of 20km from the specified coordinates.

- **retrieveScheduledVisits**(storeID, date)
  It returns information about scheduled visits to a specific store on a specific date.

- **retrieveScheduledVisits**(slots)
  It returns information about scheduled visits in certain slots (each of them is identified by shop, date and time)

- **retrieveHistoricalData**(storeID)
  It returns the average duration of a visit to the specified store. This value is computed using the historical data relating to both Guests and Customers.

- **retrieveHistoricalData**(customerEmail, storeID)
  It returns the average duration of a visit to the specified store of a specific Customer. This value is computed using his historical data.

- **retrieveHistoricalData**(customerEmail, departments, storeID)
  It returns the average duration of a visit to the specified store of a specific Customer. This value is computed using the historical data relating to his visits which include the specified set of departments.

- **retrieveHistoricalData**(customerEmail)
  It returns the slots that the Customer prefers with respect to the historical data of the visits. A slot defines a store and an hourly range.

- **retrieveOpeningHours**(storeID, date)
  It returns the opening hours on the specified date of the store whose username is the one passed as parameter. This computation takes into account any exceptional changes to timetables.

- **retrieveStorePosition**(storeID)
  It returns the coordinates of the store whose username is the specified one.

- **checkDoorType**(doorID)
  It returns the type (entrance or exit) of the gate whose identifier is the specified one.

- **retrieveStore**(doorID)
  It returns the store to which the specified door belongs.

- **retrieveVisit**(storeID, qrCode)
  It retrieves information about the visit associated with the specified QR code. An exception is thrown if the visit is not in the schedule of the store.

- **saveEntryTime**(reservationID)
  It stores the current time as the entry time for the specified visit.

- **saveExitTime**(reservationID)
  It stores the current time as the exit time for the specified visit.

- **lockVisit**(visitData)
  It temporarily reserves a store visit for a Customer. This visit must then be confirmed to be considered valid.

- **confirmSchedule**(reservationID)
  It confirms a previous temporary slot assignment.

- **cancelSchedule**(reservationID)
  It cancels the visit that has as id the specified one from the schedule.

- **scheduleVisit**(bookingData)
  It stores a booking for a visit to a specified store.

- **retrieveLessCrowdedSlots**(date, storeID)
  It retrieves time slots of a specified store that are still available for booking on a specified date.

- **retrieveLessCrowdedStores**(date, coordinates)
  It retrieves stores within a radius of 20km from the specified location that are still available for booking on a specified date.

- **retrieveAccessData**(storeID)
  It retrieves the data needed to compute the statistics for the Store Manager.

2.5.2 Account Manager Interfaces

The *Account Manager* component exposes a single external interface that is accessible from the *Mobile Application* and the *Web Server*. Its most important methods are listed below.

> **createCustomerAccount**(customerData)
Create a new account for a Customer if the provided data are correct, otherwise throw an appropriate exception.

> **createSroreAccount**(storeData)
Create a new account for a Store if the provided data are correct, otherwise throw an appropriate exception. It also makes the first connection to the *Thing-it System* and retrieves information about the shop gates.

> **authenticateStore**(storeID, pwd)
It handles the login request of a Store Manager.

> **authenticateCustomer**(email, pwd)
It handles the login request of a Customer.

2.5.3 Notification Manager Interfaces

The *Notification Manager* component exposes an external interface that is accessible from the *Mobile Application* and the *Next-Up Desktop Application*. Its most relevant methods are the followings:

> **onPositionChange**(coordinates, email)
This method is called when the *Mobile Application* detects a significant position's change. The coordinates passed as a parameter represent the new Customer's position.

> **subscribeToQueue**(storeID)
This method is called by the *Next-Up Desktop Application* to receive all the updates related to the access queue of the specified store.

In addition, an internal interface is made available for the *Visits Manager.* This interface mainly exposes the following method:

> **addNotificationListener**(lineUpData)
This method allows the Visit Manager to schedule notifications for Customers who want to be notified when they should leave to reach the store.

2.5.4 Statistics Manager Interfaces

The *Statistics Manager* component exposes an external interface that is accessible by the *Web Server*. Its most important method is the following:

> **getStatistis**(storeID)
  It returns all the statistical data relating to the given store.

Moreover, an internal interface is made available for the *Visits Manager.* This interface mainly exposes the following methods:

> **estimateDuration**(email, storeID)
  It returns the estimated duration of a visit of a specified Customer to a given store.

> **estimateDuration**(email, departments, storeID)
  It returns the estimated duration of a visit of a specified Customer to a given store that involves a specified set of departments.

> **estimateGuestDuration**(storeID)
  It returns the estimated duration of a visit of a generic Store Client to a given store.

> **estimatePreferences**(email)
  It returns a list of slots (store and time pair) that could be of interest to the Customer specified as a parameter.

2.5.5 Visits Manager Interfaces

The *Visits Manager* component exposes an external interface that is used by the *Mobile Application* and the *PTD Desktop Application*. Its most relevant methods are the ones listed below.

> **getNearbyStores(coordinates)**
  It returns the stores near the location specified as a parameter.

> **estimateWaitingTime(storeID,email)**
  It returns the estimated waiting time for the specified Customer at the given store and temporarily reserves a visit for the Customer.

> **estimateWaitingTime(storeID)**
  It returns the estimated waiting time for a generic Store Client at the given store.

> **lineUp(reservationID,transportation)**
  It handles a Customer's request to queue at a store by confirming the previous temporary reservation. It also schedules the departure notification, if required.

> **exitLineUp(reservationID)**
  It cancels a previously made temporary reservation whose identifier is passed as a parameter.

> **lineUp(storeID)**
  It handles a Guest's request to queue at a store by scheduling a new visit.

> **book(bookingData,coordinates)**
  It handles a Customer's request to book a visit to a store or, if the selected slot is full, provides alternative slots in the same store or at the same time.

Additionally, an internal interface is made available for the *Notification Manager.* This interface mainly exposes the following method:

> **computeSuggestions(email)**
  It computes the slot suggestions for the periodic notification functionality.

2.5.6 Access Manager Interfaces

The *Access Manager* component exposes an external interface that is used by the *Thing-it System*. The most important method is the following one:

> **onReadEvent**(doorID, qrCode)
  This method is called by the *Thing-it System* whenever a QR Code is read by the scanners. When called, it checks the read code.

2.5.7 Web Server Interfaces

The *Web Server* component exposes an external interface that is used by the *Web Browser* of the Store Manager to access the *Web Application*.

> **sendRegistrationRequest**(storeData)
  It handles the Store Manager's registration request. It redirects him to either a success page or an error page depending on the outcome of the request.

> **sendLoginRequest**(storeID,pwd)
  It handles the Store Manager's login request. It redirects him to either the homepage or an error page depending on the outcome of the request.

> **sendStatisticsRequest**(storeID)
  It handles the request to visualize the access statistics. It returns the page containing the various graphs and the statistics data.

2.5.8 Google Maps System Interfaces

The *Google Maps System* provides various APIs to allow access to its features. Below is reported a simplification of those that the *CLup System* will use.

> **geocodeRequest**(address)
  It returns the latitude and longitude coordinates of the address passed as parameter.

> **getTravellingTime**(sourceCoord, destinationCoord, transportation)
  It returns the time required to reach a specified destination from a given location using a specified means of transport.

2.5.9 Thing-it System Interfaces

The *Thing-it System* provides various APIs to allow access to its features. Below is reported a simplification of those that the *CLup System* will use.

> **remoteControlRequest**(email, authCode)
  This function allows CLup to control the doors of a store. The data passed as parameters represent the Store Manager's email and an authentication code that he provides to authorize the transfer of control of the gates to CLup.

> **subscribeToReadEvent**(doorIDList)
  This function allows CLup to be notified every time a QR code is read by the scanners of a store for which control has been granted to Clup using the previous method. When a reading event is detected, *Thing-it* makes a call to the previously specified **onReadEvent()** method.

> **grantAccess**(doorID)
  It opens the gate whose identifier is the one passed as a parameter.

> **denyAccess**(doorID)
  It denies the request to open the door whose identifier is the one passed as a parameter.

2.5.10 Ticket Printing System Interfaces

The *Ticket Printing System* provides various APIs to allow access to its features. Below is reported a simplification of the one that the *CLup System* will use.

> **printTicket**(ticket)
>
> This method allows CLup to request the issuance of a paper ticket. The data of the ticket are passed as a parameter.

2.5.11 Database Interfaces

The *DBMS* exposes appropriate methods for querying the database and accessing query results.

2.5.12 Mobile Application Interfaces

The *Mobile Application* exposes two external interfaces that are used by the *Notification Manager* component of the *Application Server* to send notifications to the Customer and to access his position. Its most relevant methods are the ones listed below.

> **notify**(suggestions)
>
> It allows the *Notification Manager* to send a notification containing information about some suggested slots to the Customer's smartphone.

> **notify**(notificationText)
>
> It allows the *Notification Manager* to send a notification containing a text string to the Customer's smartphone.

> **getPosition**()
>
> It returns the latitude and longitude coordinates relating to the position of the Customer's smartphone.

> **subscribeToPositionChanges**(coordinates)
>
> This method allows the *Notification Manager* to be alerted of any significant change in position with respect to the given coordinates. When a location change is detected, the *Mobile Application* contacts the *Application Server* by calling the **onPositionChange()** method.

2.5.13 Next-Up Desktop Interfaces

The *Next-Up Desktop Application* exposes an external interface that is used by the *Notification Manager* component of the *Application Server* to update the list of the reservationIDs that can access a specific store.

> **onQueueChange**(reservationIDList)
  This function is used to update the list of the reservationIDs that are shown on the display.

2.6 Selected Architectural Styles and Patterns

In this section are explained the most relevant architectural choices. In particular, the reasons that led to their adoption are detailed here.

2.6.1 Four tiers client-server

The System is organized according to the well-known and most used architectural style for distributed applications: ***Client-Server architecture with four tiers***.
As already explained in the previous sections, it consists of the following tiers:

- The ***client tier*** or ***presentation tier*** consists of application clients that make requests to the server, which processes the requests and returns a response. Clients can be a web browser or a standalone application. The first communicates with the web tier the latter interacts directly with the business tier.
- The ***web tier*** consists of components that handle the interaction between the browser and the business tier. It collects input from the browser and returns appropriate results from the components in the business tier. It also controls the flow of pages and maintains the state of the session.
- The ***business tier*** consists of components that provide the business logic for the application. In a properly designed enterprise application, the core functionality exists in the business tier components.
- The ***enterprise information systems*** (EIS) ***tier*** or ***data tier*** consists of the database server. These resources are accessed by components on the business tier.

This modular structure increase maintainability, flexibility and scalability. Indeed, all the parts of the System can be upgraded independently. The 4-tier model is architected to create a foundation for excellent performance, device-tailored experiences, and allows for the integration of both internal services as well as third-party services and APIs.

Moreover, the main reason behind the segmentation between *Web Server* and *Application Server* is related to the greater security that this solution in general offers. This choice also has positive implications on the speed of implementation of the System, allowing different teams to work in parallel on the different tiers.

2.6.2 Stateless Components

The components of the *Application Server* do not maintain any state or context between requests. Each of them contains all the information needed for the provision of the service. This property is ideal in high-volume applications, increasing performance by removing the server load caused by retaining session information.

2.6.3 Data Access Component

The functionalities to store and access data elements are provided by a special component that isolates the complexity of data access and enables additional data consistency. This component is the *Data Manager*. Indeed, it coordinates all data manipulation. In case a storage offering shall be replaced or the interface of a storage offering changes, the data access component is the only component that has to be adjusted.

2.6.4 Provider Adapter

The provider interfaces are encapsulated and mapped to unified interfaces. These are used to separate provider interaction concerns from application functionality. Specifically, the *Door Controller* component takes care of encapsulating the specific implementation of the *Thing-it external System.*

2.6.5 Observer

The observer pattern is used at several points in the system architecture. Firstly, for the proper functioning of the *Notification Manager* component. When a Customer declares that he wants to be notified when it is time to reach the store, the *Notification Manager* stores the choice and sends him a notification when it detects that it is actually time to leave. Similarly, the *Notification Manager* observes the Customer's location through the *Mobile Application*, which notifies the *Application Server* when it detects a significant change in location. Finally, the pattern is also used to communicate with the *Thing-it System.* In fact, when a new store registers to CLup, the *Application Server* contacts *Thing-it* in order to subscribe to the reading events of the QR Code readers. When a code is read, *Thing-it* notifies the *Access Manager* by forwarding the read data.

2.7 Other design decisions

2.7.1 Password Storing and User Authentication

In order to guarantee confidentiality, the login to CLup requires a personal identifier and a password. For Customers, the id coincides with the e-mail address entered during registration. For Store Managers, it corresponds to the username chosen for their store (StoreID).
For each registered entity, the access passwords are first hashed using SHA512 algorithm and then stored in the *Database*. No passwords are stored in clear text in the *Database*.

2.7.2 Relational Database

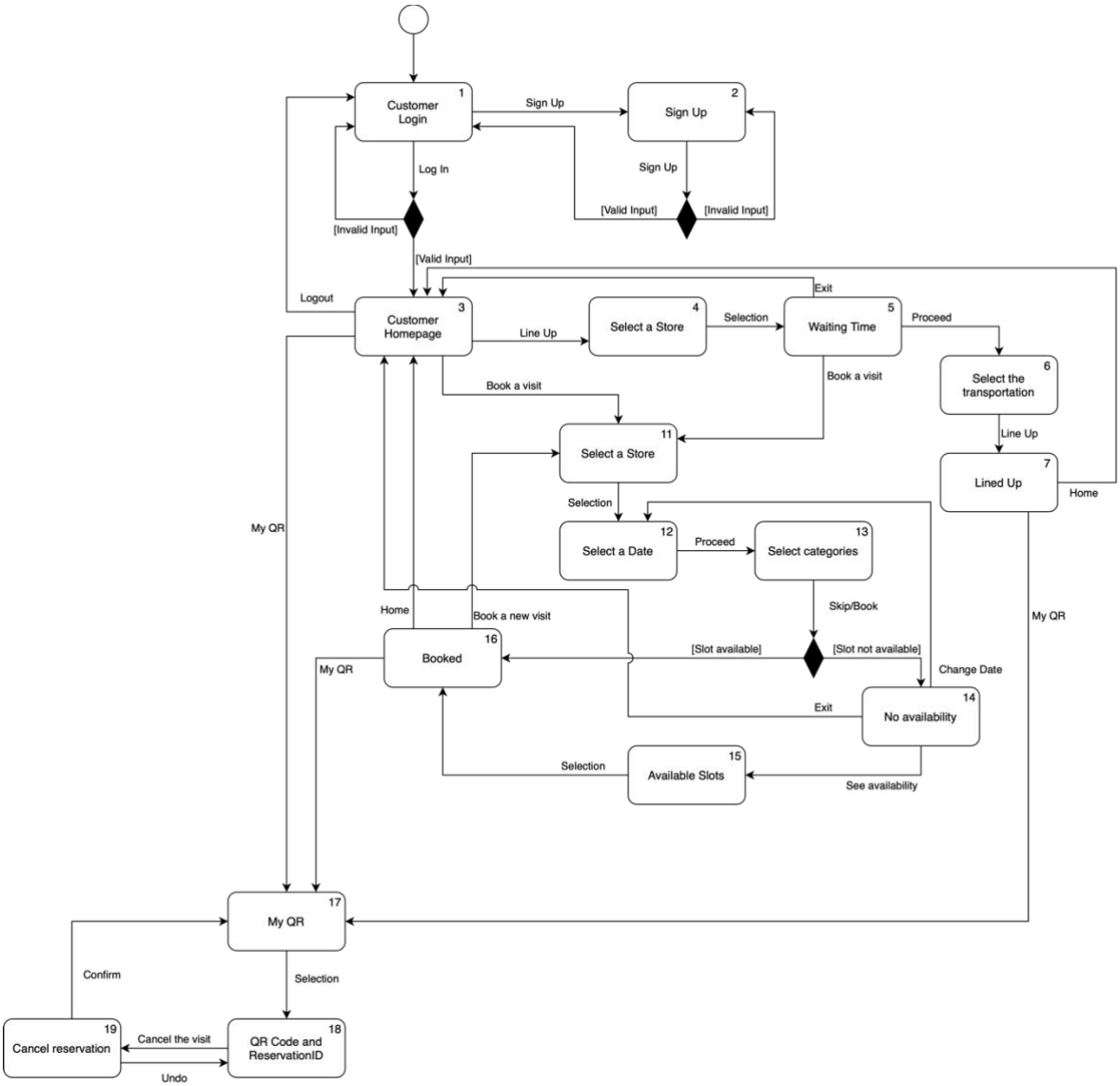
The data management of the system will be handled by a Relational DBMS. Indeed, a relational database is transactional and the so-called ACID properties are guaranteed. No partial execution is allowed, every state is consistent after a transaction execution and each transaction is isolated. Even in case of failures, changes in the database persist.

Chapter 3
3 User Interface Design

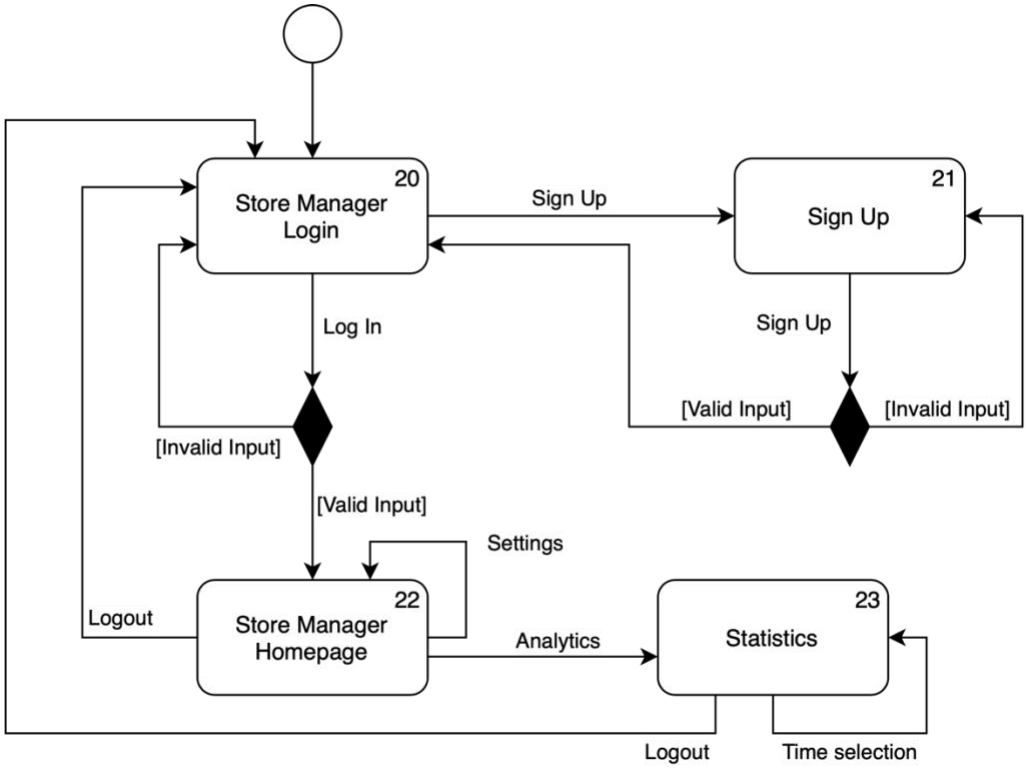
3.1 UX Diagrams

All the interfaces have already been presented in the RASD. For this reason, in the next sections flow graphs of the application are provided.
In the following graphs, nodes represent the screens, referenced by figure number in the RASD; arcs represent buttons and square brackets represent conditional branches.

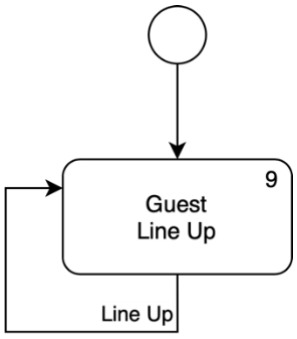
3.1.1 Mobile Application Flow Graph



3.1.2 Web Application Flow Graph



3.1.3 PTD Desktop Application Flow Graph



Chapter 4
4 Requirements Traceability

In this section, the mapping between the components of the CLup System and the requirements described in the RASD is shown. As explained, the System consists of the following components:

[C.1] Data Manager
[C.2] Account Manager
[C.3] Notification Manager
[C.4] Statistics Manager
[C.5] Visits Manager
[C.6] Access Manager
[C.7] Web Server
[C.8] Google Maps System
[C.9] Thing-it System
[C.10] Ticket Printing System
[C.11] Database
[C.12] Mobile Application
[C.13] PTD Desktop Application
[C.14] Next-Up Desktop Application
[C.15] Web Application

Requirement		Components
ID	Description	
[R.1]	The System must allow Customer to register to CLup by filling in a form containing a set of fields	Account Mobile Application
[R.2]	The System must store the data relating to the Customers	Data Database
[R.3]	The System must allow Customer to login to CLup by entering his email and password	Data Account Database Mobile Application

[R.4]	The System must allow Store Manager to register to CLup by filling in a form containing a set of fields	Account Web Server Web Application
[R.5]	The System must store the data relating to the supermarkets	Data Database
[R.6]	The System must allow Store Manager to login to CLup by entering his Store ID and password	Data Account Web Server Database Web Application
[R.7]	The System must ask the Customer for permission to access his local position	Mobile Application
[R.8]	The System must ask the Customer for permission to send him notifications	Mobile Application
[R.9]	The System must be able to generate electronic tickets	Visits
[R.10]	The System must be able to emit paper tickets	Visits Ticket Printing System PTD Desktop Application
[R.11]	The System must be able to associate a unique QR code to each issued ticket	Data Visits Database
[R.12]	The System must be able to associate a unique Reservation ID to each issued ticket	Data Visits Database
[R.13]	The System must store the data relating to the queues of the shops	Data Database
[R.14]	The System must be able to read a QR code at the entrance of a store	Access Thing-it System
[R.15]	The System must be able to read a QR code at the exit of a store	Access Thing-it System
[R.16]	The System must be able to control the sliding doors of a store	Access Thing-it System

[R.17]	The System must be able to show which Reservation IDs can access a store at any given time	Notification Next-Up Desktop Application
[R.18]	The System must be able to inhibit or grant access to a store for issued tickets	Data Access Thing-it System Database
[R.19]	The System must allow Guest to require a paper ticket through the PTD	Visits PTD Desktop Application
[R.20]	The System must allow Guest to view the estimated waiting time from the PTD	Visits PTD Desktop Application
[R.21]	The System must keep stored historical data relating to the time of access and exit from the store	Data Database
[R.22]	The System must allow Customer to require an electronic ticket through the Application	Visits Mobile Application
[R.23]	The System must periodically estimate the waiting time	Data Statistics Visits Database
[R.24]	The System must allow Customer to visualise the estimation of the waiting time from the Application	Visits Mobile Application
[R.25]	The System must periodically check the traffic conditions	Notification Google Maps System
[R.26]	The System must be able to send a notification to Customer	Notification Mobile Application
[R.27]	The System must allow Customer to send a request to book a visit by filling in a form containing a set of fields	Visits Mobile Application

[R.28]	The System must store the data relating to the booking of visits	Data Database
[R.29]	The System must be able to compute the average time of visit for long-term Customers	Data Statistics Database
[R.30]	The System must allow Customer to visualise his valid electronic tickets	Data Visits Database Mobile Application
[R.31]	The System must periodically compute the statistics relating to the stored access data	Data Statistics Database
[R.32]	Store Manager must be able to visualise the statistics relating to the access data	Statistics Web Server Web Application
[R.33]	Store Manager must be able to change the departments and the capacity of each of them	Account Web Server Web Application

	[C.1]	[C.2]	[C.3]	[C.4]	[C.5]	[C.6]	[C.7]	[C.8]	[C.9]	[C.10]	[C.11]	[C.12]	[C.13]	[C.14]	[C.15]
[R.1]		X										X			
[R.2]	X										X				
[R.3]	X	X									X	X			
[R.4]		X					X								X
[R.5]	X										X				
[R.6]	X	X					X				X				X
[R.7]												X			
[R.8]												X			
[R.9]					X										
[R.10]					X					X			X		
[R.11]	X				X						X				
[R.12]	X				X						X				
[R.13]	X										X				
[R.14]						X			X						
[R.15]						X			X						
[R.16]						X			X						
[R.17]			X											X	
[R.18]	X					X			X		X				
[R.19]					X								X		
[R.20]					X								X		
[R.21]	X										X				
[R.22]					X							X			
[R.23]	X			X	X						X				
[R.24]					X							X			
[R.25]			X					X							
[R.26]			X									X			
[R.27]					X							X			
[R.28]	X										X				
[R.29]	X			X							X				
[R.30]	X				X						X	X			
[R.31]	X			X							X				
[R.32]				X			X								X
[R.33]		X					X								X

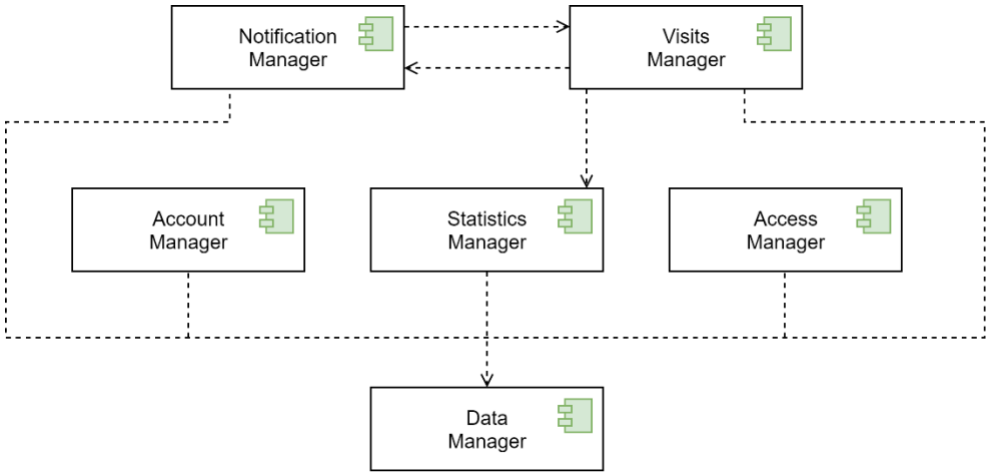
Chapter 5
5 Implementation, Integration and Test Plan

5.1 Development Process

The four tiers that constitute the System can be implemented in parallel and integrated at the end of development together with the *External Systems*. This choice is motivated by the different nature of the various tiers and by the need to speed up the development process to solve the problems associated with gatherings in supermarkets as soon as possible. The business logic layer is the most critical and difficult to implement, and the following sections focus on its development.

A ***bottom-up approach*** will be adopted for the implementation and integration procedure: starting with the primitive elements provided by the implementation language and ending when the desired system is reached. At each stage, the available elements are used in the construction of new, more powerful elements. These new elements will in turn be used in the next phase in the construction of even more powerful elements, and so on until the available elements can be used directly in the construction of the whole system.

The following diagram details the dependencies between the various components of the *Application Server*.



Dependency Diagram 1

The following table represents the main components of the *Application Server*, along with the difficulty of implementing them.

Component	Implementation complexity
[C.1] Data Manager	Low
[C.2] Account Manager	Medium
[C.3] Notification Manager	High
[C.4] Statistics Manager	Medium
[C.5] Visits Manager	High
[C.6] Access Manager	Medium

5.2 Implementation Plan

The implementation order for the development of the *Application Server* is the following one:

- **Data Manager:** this is the first module to be implemented. Indeed, all the other components of the *Application Server* rely on it to communicate with the *Database*. Therefore, it is a module of crucial importance but characterized by a not high level of complexity as it only implements the queries to the database.

- **Statistics Manager:** it could be implemented in parallel with the *Account Manager* and the *Access Manager* as they are mutually independent. All three components have the same levels of complexity, but this one is essential for the correct functioning of the *Visits Manager.* Given the high complexity of the latter, higher priority will be given to the development of the *Statistics Manager*.

- **Account Manager:** the implementation of this module follows that of the *Statistics Manager*. Indeed, this module is necessary for the correct configuration of the *Thing-it System* and therefore it will precede the development of the *Access Manager*.

- **Access Manager:** the development of this part follows the implementation of the *Account Manager* as it is a module that is particularly critical and given the considerable complexity of its development caused by the interaction with an external system (*Thing-it System*).

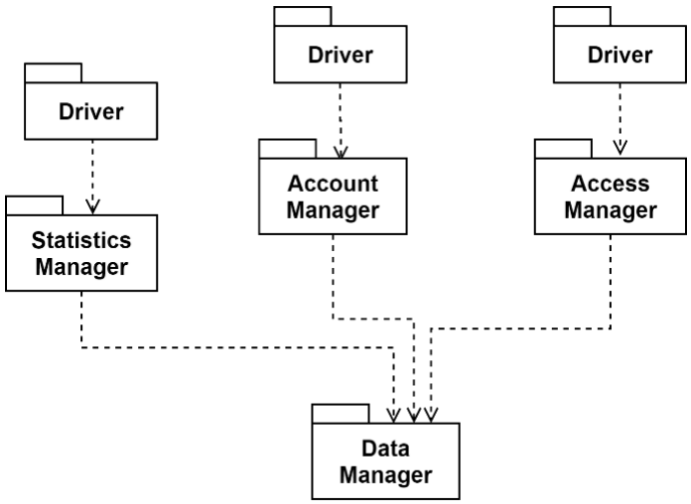
- **Visits Manager:** this component is then implemented. It represents a critical point for the success of the entire project, and, at the same time, it requires to interact with the components listed above.

- **Notification Manager:** once the above modules are completed, the implementation of the *Notification Manager* takes place. This choice is due to the fact that, although this component is particularly complex to implement, it only handles additional or marginal functions of the System.

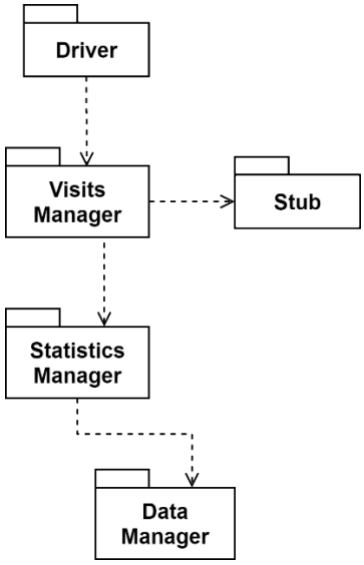
5.3 Integration Sequence

This section briefly describes the integration plan for the system to be developed. Each component must first undergo ***Unit Testing***. The integration testing process occurs incrementally to allow for bug tracking. Every time a component of a dependency level is completed, it is integrated with the modules of the lower level to test the behaviour of the developed subsystem.

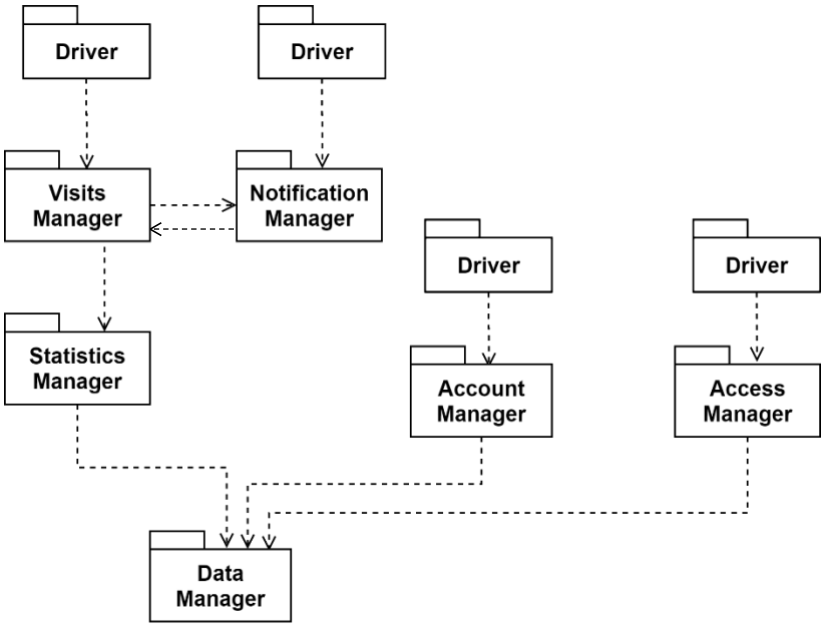
The following diagrams illustrate the integration process at the various levels of dependency. Note that it is not possible to apply a pure bottom-up approach due to the mutual dependency relationship between the *Visits Manager* and the *Notification Manager*.



Integration Diagram 1: integration of Data Manager, Statistics Manager and Access Manager.



Integration Diagram 2: integration of the Visits Manager component



Integration Diagram 3: final Application Server integration

5.4 System Testing

Once all system components have been integrated, ***System Testing*** begins. This process aims at verifying functional and non-functional requirements and must take place in a testing environment that is as close as possible to the production environment. Specifically, *CLup System* will be subjected to the following tests:

- **Functional testing**: It ensures that the requirements and the specifications defined in the RASD are properly satisfied by the System.

- **Performance testing**: The main purpose is to identify and eliminate the performance bottlenecks in the software application affecting response time, utilization, throughput.

- **Load testing**: It aims at detecting bugs such as memory leaks, mismanagement of memory and buffer overflows. It also identifies the maximum operating capacity of the application.

- **Stress testing**: It verifies stability and reliability of software application. The goal is measuring software on its robustness and error handling capabilities under extremely heavy load conditions and ensuring that software doesn't crash under crunch situations. It even tests beyond normal operating points and evaluates how software works under extreme conditions.

- Chapter 6

6 Effort Spent

The following tables summarize the effort spent by each member of the team to create the DD document.

6.1 Leoni Luca

Description of the task	Hours
Document Introduction: Purpose and Scope Summary	3
Identification of External Services, Patterns and Architectural styles	3
Definition of the General Architecture and Components Identification	4
Deployment of the System: definition and diagram	3
General definition of Runtime Views: identification and description	12
Runtime View Refinements	5
Description of the Patterns and Architectural Choices adopted	2
UX Diagrams: definition and realization	2
Full Revision of Chapter 2	4
Requirements Traceability Definition	4
Full Revision of Chapters 3 and 4	3
Definition of the Development Process	3
Revision of the Document and further improvements	4
Fixes to the Runtime View section	4

6.2 Locarno Silvia

Description of the task	Hours
Document Introduction: Purpose and Scope Summary	3
Identification of External Services, Patterns and Architectural styles	3
Definition of the General Architecture and Components Identification	4
Architecture Overview and Component View Diagrams	2,5
General definition of Runtime Views: identification and description	12
Actual creation of Runtime View Diagrams	5
Runtime View Refinements	5
Description of the Patterns and Architectural Choices adopted	1.5
Full Revision of Chapter 2	4
Requirements Traceability Definition	4
Full Revision of Chapters 3 and 4	3
Definition of the Development Process	2
Revision of the Document and further improvements	4
Fixes to the Runtime View section	4

6.3 Minotti Luca

Description of the task	Hours
Document Introduction: Purpose and Scope Summary	3
Identification of External Services, Patterns and Architectural styles	3
Definition of the General Architecture and Components Identification	4
Detailed Description of the Components and their Composition	5
Detailed Description of the Component Interfaces	5
General definition of Runtime Views: identification and description	12
Runtime View Refinements	5
ER schema definition	1
Description of the Patterns and Architectural Choices adopted	2
Full Revision of Chapter 2	4
Requirements Traceability Definition	4
Definition of the Development Process	2
Revision of the Document and further improvements	4
Fixes to the Runtime View section	4

Chapter 7
7 References

- E. Di Nitto. Lecture Slides. Politecnico di Milano.
- E. Di Nitto. Project Assignment AY 2020-2021. Politecnico di Milano.
- Uml Diagrams. uml-diagrams.org
- Cloud computing patterns. cloudcomputingpatterns.org
- Oracle - Tiered Applications. docs.oracle.com
- Google Maps Platform. developers.google.com/maps
- Thing-it. thing-it.com