POLITECNICO
MILANO 1863

- DD -

Design Document

Computer Science and Engineering

Software Engineering II

A.A. 2020/2021

Daniele Mammone - 10625264
Gianmarco Naro - 10610374
Massimo Parisi - 10583470

Contents

1 Introduction

1.1 Purpose

The main target of this document is to describe the **Customer Line-Up** (*CLup*) design from a more detailed point of view. This document follows faithfully what was defined in the Requirement Analysis and Specification Document and must be read carefully before starting the software implementation in order to understand the software design in detail. Moreover, we tried to maintain independence between *DD* document and *RASD* document in order to allow greater flexibility in case you want to reuse the design model.

1.2 Scope

CLup is a booking application and nowadays this type of applications are more and more widespread in people's everyday life and are becoming more and more indispensable. For this reason, intelligent design choices have been made that extrapolate the positive aspects of existing booking apps, in order to make CLup really "achievable" in real life.

The main purpose of CLup is to facilitate customers to access at a store in **security**, both allowing them to reserve a spot on the queue for entering the store through the app and to book a visit at the store at a determined time of a certain day, selected by the customer. Thanks to this, store managers can manage the **affluence** in their store more easily, and moreover can reduce the crowd in front of the store, that is one of the main purposes of the application. Another important feature deals with obtaining statistics from customers and generic information from stores in order to help customers during their reservation. For this, the system has to store a very large amount of data. This data are mined and used later, which is why it is very important to identify the user's role so that we can provide him with more accurate and suitable information. Indeed a customer wants to provide information about the stores while, for example, a store manager is more interested in information about customer permanence in the store.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

QR Code	Bi dimensional bar code that allows the user to check-in/check-out at the store entries/exits
Reservation	Indicates both booked visits and spots on the queue to enter the store as soon as possible
Customer	The clients of the store, that uses the system to get a reservation to access the store
Store manager	The app user that access to stores' bookings, occupancy and settings, in order to manage the flow of customers
QR Code Reader	Device used to scan customers' *QR Code*
Totem	Electronic device that allows customers to physically get a spot on the queue to enter the store a soon as possible; it allows to specify the same parameters that can be inserted through the app
QR Code Printer	Device used by totems to print *QR Code*
Department	Part of the store that contains the same category of products
Query	Synonym for request

1.3.2 Acronyms

RASD	Requirement Analysis and Specification Document
DD	Design Document
ETA	Estimated Time of Arrival
GPS	Global Positioning System
API	Application Programming Interface
UML	Unified Modeling Language
DBMS	DataBase Management Service
OS	Operative System
HTTPS	HyperText Transfer Protocol over Secure Socket Layer
TCP	Trasmissione Control Protocol
IP	Internet Protocol

1.3.3 Abbreviations

IIT	Implementation, Integration and Testing
Rn	Requirement number n
ASAP	As soon as possible

1.4 Revision History

Version	Date	Changelog
1.0	24/12/2020	First version

1.5 Software and Tools

- LaTeX as software system for docuement preparation
- UMLet for the UML diagrams and other diagrams
- Photoshop for the mockups
- Git & Github as work space. The repository is here.

1.6 Reference Documents

- Specification Document
- Slides of the lectures

1.7 Document Structure

The structure of the document is thought with the intention of allowing simple navigation through it. Also, various abbreviations, highlighted in Abbreviations section, have been used to make the content smoother. Hence, the structure of the document is the following one:

- **Introduction**: this section gives a general view of the problem and describes the scope and purpose of the *DD*, including a set of definitions, acronyms and abbreviations used.
- **Architectural Design**: this section starts with a high level description of the architecture of the system and continues going into details, specifying the components and interfaces used.
- **User Interface Design**: this section presents the mockups of the application, describing how the clients can navigate the application, highlighting the actions they can do.
- **Requirements Traceability**: this section describes the connection between the RASD and the DD, identifying the relation between goals and requirements described previously and the components that allow to realize them.
- **Implementation, Integration and Test Plan**: this section establishes a plan for the development of components, identifying the conditions needed to be met before starting the development process and then maximizing the efficiency of the developer teams with a precise schedule.

- **Effort Spent**: the main focus of this section is to track the time spent to complete this project. In particular, is highlighted the subdivision of the working hours of the various sections.
- **References**: this section is dedicated to all references used in this project.

2 Architectural Design

The aim of this section is to give a look to the architectural aspect of the analyzed system. In doing this, a top-down approach will be followed, starting from a very general view of the system, then going into more detail through **Component Diagrams**, more and more detailed. This allows us to precisely describe each component of *CLup*, so that who will develop the system, well know the behaviour of each component.

2.1 Overview

A first high level overview can be done through the system's **Composition Diagram**. In fact, here there is a first subdivision of the system's component, each one with a different task.

- **Communication Interfaces**: to work properly, the system needs to interface overt the network with both remote applications, and external services. Because of this, there are some components devoted to interact with remote devices:
  - **UserHandler**: to interact with users, both managers and customers, the latter both at *totems*, and on app.
  - **EntranceManagement**: so that the store can process a scanned *QR Code* at entry/exit.
  - **NumberCallingInterface**: to notify the call of a certain number.

  and others deputy to access external services, such as
  - **Map Component**: to access an external **Map Provider**.
  - **Notification Component**: to send notifications to customers through an external **Notification Service**.
  - **Mail Component**: to externally sending mails, avoiding building an internal **Mail Manager**.
- **Business Logic**: here there is the whole Business Logic of the system. Since each store may maintain a different logic from another, there is a Component for each store managed by the system. Each store, manages the following aspects:

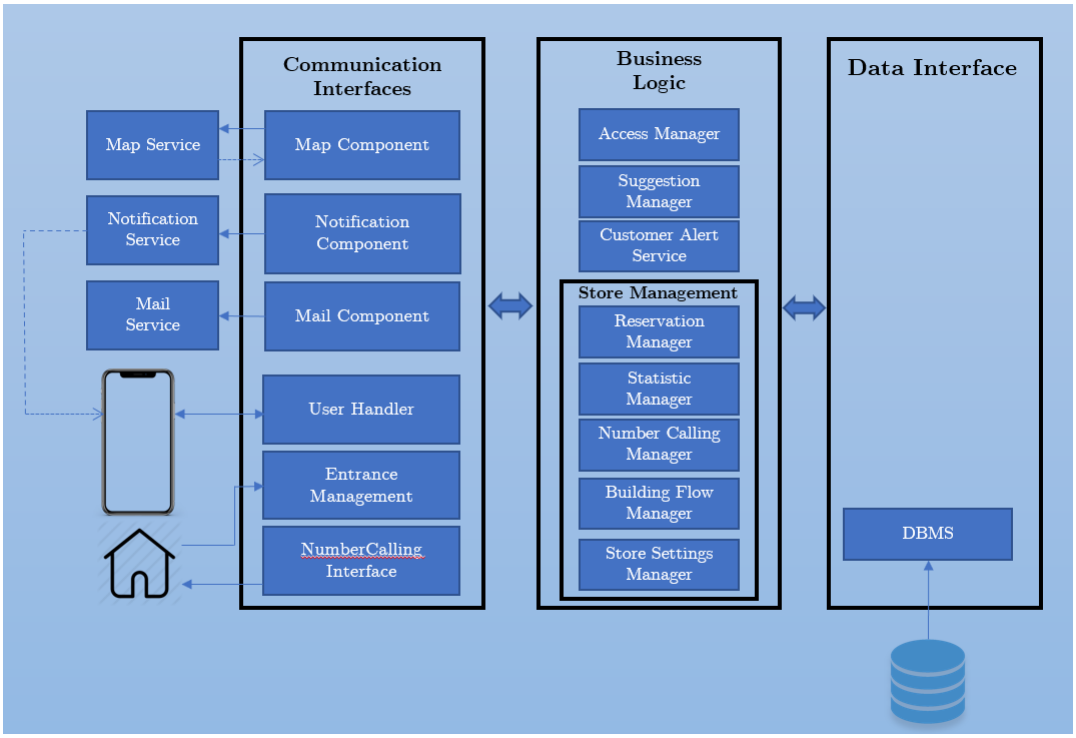

Figure 1: *Number calling system*

- **Reservation Manager**: manages all the things concerning reservations and their making. It's useful also to retrieve some important data, such as *ETA*s and available time intervals to enter the store.
- **Statistic Manager**: it's the part used for the calculation of statistics about customers' shopping time, and about the average time spent in a single department. Since each store is different from the other, the statistics are maintained unique per store. This part is also used by **Reservation Manager** to calculates *ETA*s and time intervals;
- **Number Calling System**: is the component dedicated to admit reserved customers inside the store.
- **Building Flow Manager**: it takes care of processing scanned *QR Codes* at the entry and the exit of the store.
- **Store Settings Manager**: manages all the things concerning stores' settings and parameters, such as working hours and capacity (also for each department).

This subdivision allows to separate better the logic off each store. Moreover, there are component common to each store, so the ones deputy to allow users to request services to the system.

  - **Access Manager**: manages users' registraion and log-in;
  - **Suggestion Manager**: manages the retrieve of suggestions when required, or necessary;
  - **Customer Alert Service**: takes care of sending departing notifications to customers.

  At the end, there is another block: the **Data Interface**: it represents the *Data Tier* of the system and separates the Business Logic from the data. It interacts with the *DBMS* to store and load informations.

2.2 Component View

The aim of this section is to give a look to the architectural aspect of the analyzed system. In doing this, a top-down approach will be followed, starting from a very general view of the system, then going into more detail through **Component Diagrams**, more and more detailed. This allows us to precisely describe each component of *CLup*, so that who will develop the system, well know the behaviour of each component.

- **MobileApp**: it is the component relative to the mobile application used by store managers and customers; in order to access *CLup* services. It uses three interfaces: *CustomerInterface*, *StoreInterface*, used to access functionalities respectively for customers and store managers, and *Access-Interface*, that allows users to log-in and register.
- **CLupServer**: it is the component deputy to manage users' access, and customers requests.
- **StoreComponent**: it is the component that principally manages the business logic of the system, since takes care of the functionalities associated with a single store. Each store will have its own **Store Component** deployed in a dedicated container. The aim of this is to separate the critical functionalities of each store, to reduce the risk of a complete block of the system and to make maintenance easier in case of malfunctionalities of a single store.
- **TotemApp**: it is the component that represents the interface used by a physical totem installed at stores in order to request the issue of a ticket. The totem interface is separated from the Mobile App's one, since a totem can request tickets only to the store where it's installed, directly communicating with it.
- **ScannerApp**: it is the component that represents *QR Scanners*', that asks the associated store to process the scanned *QR Codes*.
- **NumberCallingApp**: it is the component deputy to notify at stores that a number has been called. It's used by the associated store to notify that some customers are admitted to enter the store.

Moreover, to exploit some features, *CLup* relies on some external services, using them through their *API*s:

- **MapsAPI**: exploits the *API*s of an external map services, necessary to compute percurrency times and sort stores depending on the position.
- **NotificationAPI**: it is the interface used to send notifications to customers' devices. Since each mobile *OS* already has its own notification system, *CLup* exploits this in sending notifications, without the need of building a new in-house notification service.
- **EmailAPI**: it is the interface used to communicate with a mail provider service, to send emails to customers.
- **DBMSAPI**: the interface used to access the Data Logic of *CLup*.

It has been chosen to not develop these features, since there isn't a real necessity of making them from scratch: the existing ones work well, are already tested and complete (for example, developing an internal map service will require inserting the whole world cartography, while a non *OS*-integrated notification service must deal with aggressive *OS*es' power management). Moreover, this outsourcing allows to reduce the time for developing the system, and the maintenance costs, since it's externally done by the service provider.

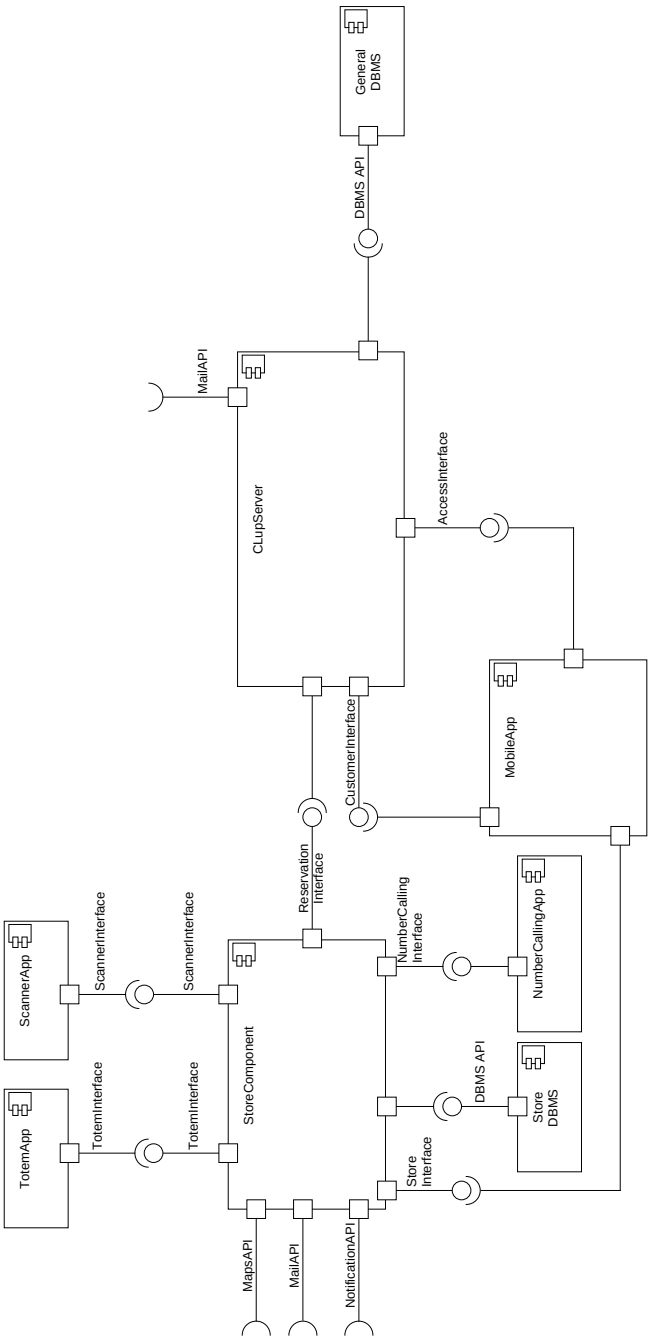

Figure 2: *High Level Component Diagram*

2.2.1 Adapters

The system uses some "Adapters" to access the external services; no *API*s are directly exposed to any component but the adapters. So, it is sufficient to change the corresponding adapter to migrate to another service, without the need of rewriting code of other components. Moreover, if required in some circumstances, it's possible to use different external services: it is sufficient to use the correct adapter in order to access the desidered service.

- **MapManager**: it is the component that takes requests from other components, and redirect them to the chosen external **Map Service**.
- **NotificationComponent**: since each mobile *OS* uses its own notification service, this adapter takes in requests of notification sending, and redirect them to the correct service; the latter will take care of delivering the notification to the customer.
- **MailComponent**: an adapter used to send emails; as the previous ones, allows to change the used mail service painlessly.
- **DataManager**: probably, the most important adapter of the system, since it is the one connecting the other components to the system's data logic.

Here, there is described the one for *DBMS*, since it's the most articulated among those.

2.2.1.1 Data Manager Component The Data Manager is the component that manages the requests for data manipulation. The components analyzed two aspects:

- Data Manipulation
- Query Execution

The components' services are requested through the *DataInterface*, while it access to the *DBMS* through its *API*s. Data can be accessed or written to the *DBMS*. This partition is very important because it allows to avoid writing instead of reading and vice versa.

- **DataRequestor**: it is the component that manages the data reading. Indeed if *DataInterface* asks for a data, this component allows to satisfy this request
- **DataStorer**: it is the component that manages the data writing. Indeed if *DataInterface* asks to write new data in the *DBMS*, this component allows to satisfy this request.
- **QueryExecutor**: it is the component that manages the connection with the *DBMS*. It uses the *DBMSAPI* and both *DataRequestor* and *DataStorer* rely on him to interface directly with the database

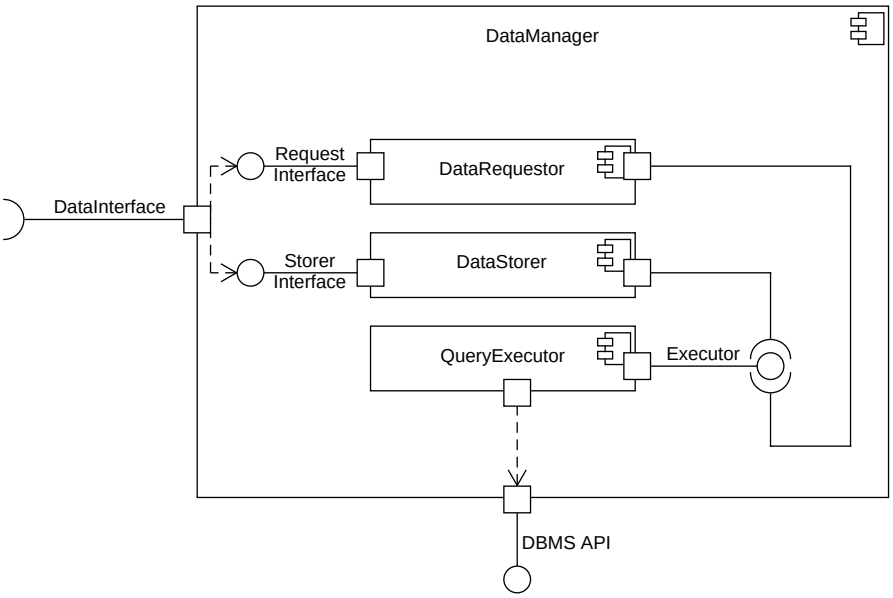

Figure 3: *Data Manager Component View*

2.2.2 CLup Server

As said before, it manages customers' requests and access operations, through the two interfaces it expose:

- **CustomerInterface**: expose all the methods relative to services a customer can access, such as making a reservation and retrieving already made reservations.
- **AccessManager**: the interface deputy to manage login and registration of users. After a login, will send to the user the correct set of operations the user can do, depending on their role: customers or store managers.

In the following paragraphs, there is a description of the most relevant component contained in **CLup Server**.

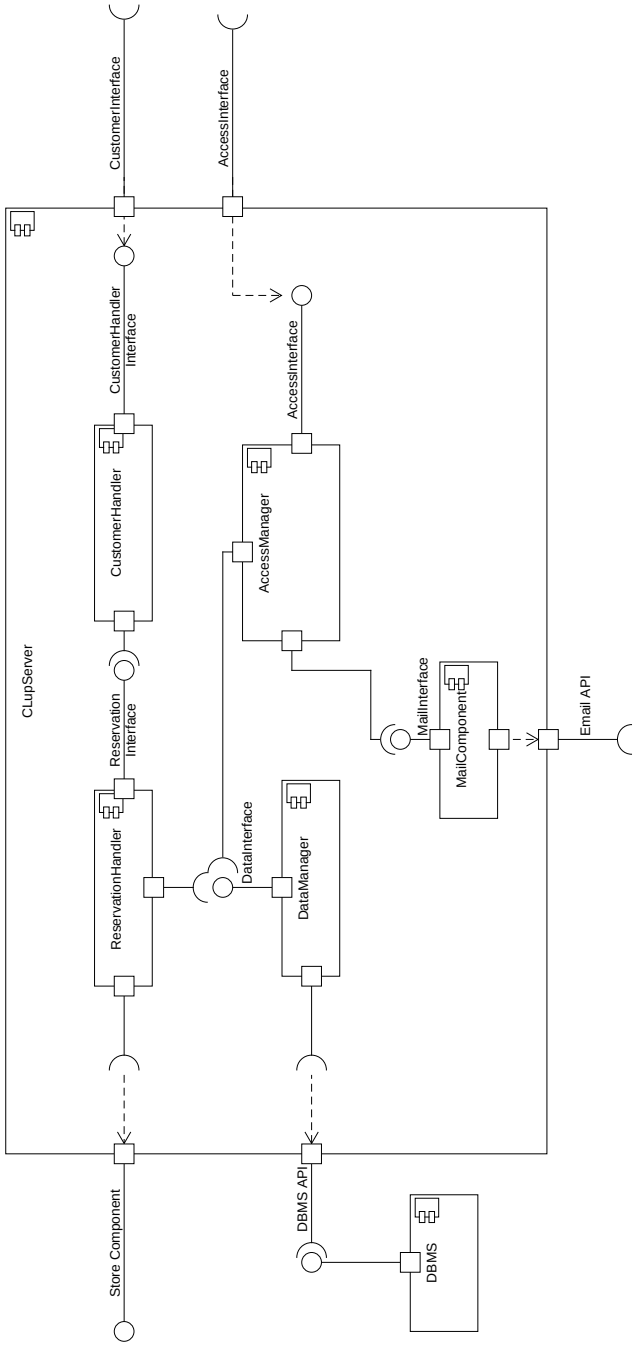

Figure 4: *Server Component View*

2.2.2.1 Customer Handler Component The CustomerHandler is the component that manages all the actions that a customer can make within the application. The component analyze the aspects concerning reservations.

This component is used by invoking methods on the *CustomerHandlerInterface*. This component is divided in some subcomponents:

- **ASAPManager**: it is the component that manages the *ASAP* reservations and it is formed from two sub-component: **DeleteRes** and **AddRes**. The first one deals with deleting an *ASAP* reservation while the second one deals with creating a new *ASAP* reservation.
- **BookingManager**: it is the component that manages the booking via app and it is formed from three sub-component: **DeleteRes**, **ModifyRes** and **AddRes**. The first one deals with deleting a reservation, the second one deals with modifying a reservation while the third one deals with adding new reservations.
- **RetriveReservation**: it is the component that is used to retrieve and update all user's reservations.

All of these components uses the *ReservationInterface* to complete their tasks.

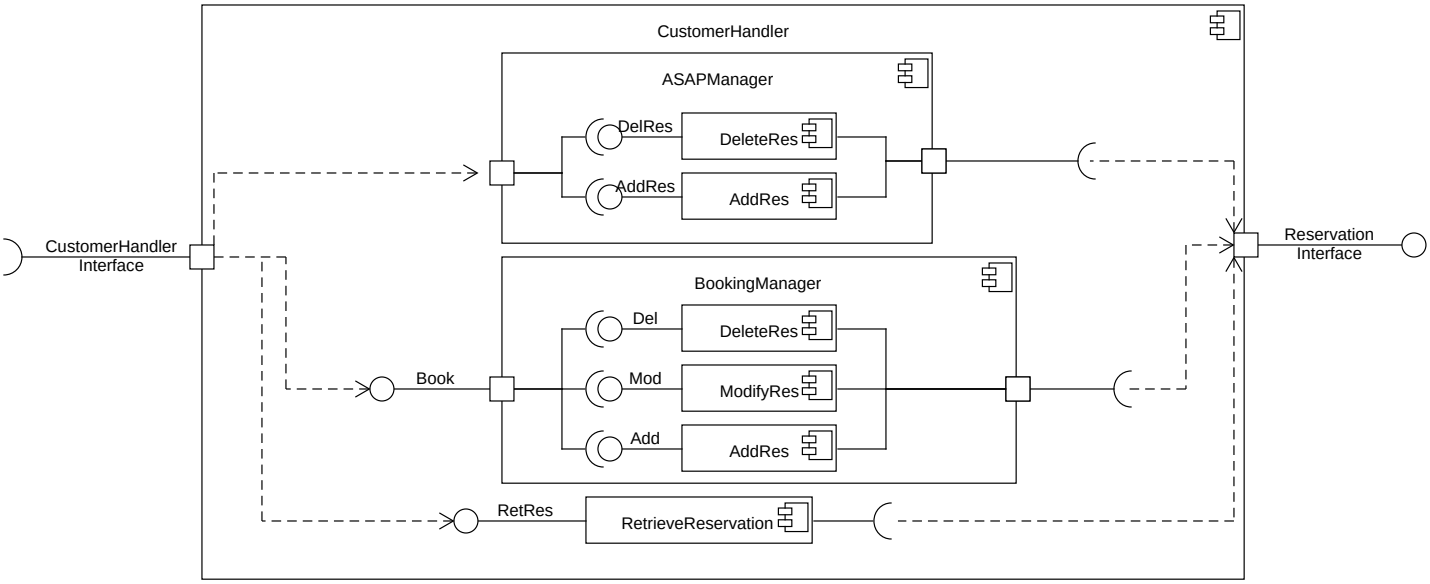

Figure 5: *Customer Handler Component View*

2.2.2.2 Access Manager Component The Access Manager is the component that manages all the issues concerning the access of the users. Indeed, as mentioned above, it is very important to recognize the role of the users in the app in order to provide them correct information, and only the functionalities/-data they are authorized to use. The component analyze two main aspects:

- Login
- Registration

The component uses two very important interfaces: *EmailAPI* and *DataInterface*. The first one is used for the registration, in order to send confirmation messages, while the second one is used for storing registration data, and to check credentials for log-in operations. Furthermore, its services are requested through the *AccessInterface*, that implements the interfaces exposed by **Login Component**, and **Registration Component**. This component is invoked by *StoreManageAccessInterface* and *CustomerAccessInterface* in case an user wants to log in or register

- **LoginManager**: it is the component that manages user's login. Indeed, if a user logs in, the component checks if the credentials are correct and then allows the user to use their personal area.
- **RegistrationManager**: it is the component that manages the user's registration. Indeed, if a user makes a registration, this component, at the beginning, checks through *DataInterface* if the data inserted by the user are valid (e.g. uniqueness of the email, or certification for store managers). Then, if the data are correct, the user's data are stored in the *DBMS*.
  - **RegistrationChecker**: it is the component that checks if the information is inserted during the registration, from the user, are valid. In case of customers, this component checks if the email is valid while in case of store manager checks if the store certification is valid and the registration *ID* is unique.
  - **RegistrationStorer**: it is the component that stores the new data in the *DBMS* after checking.

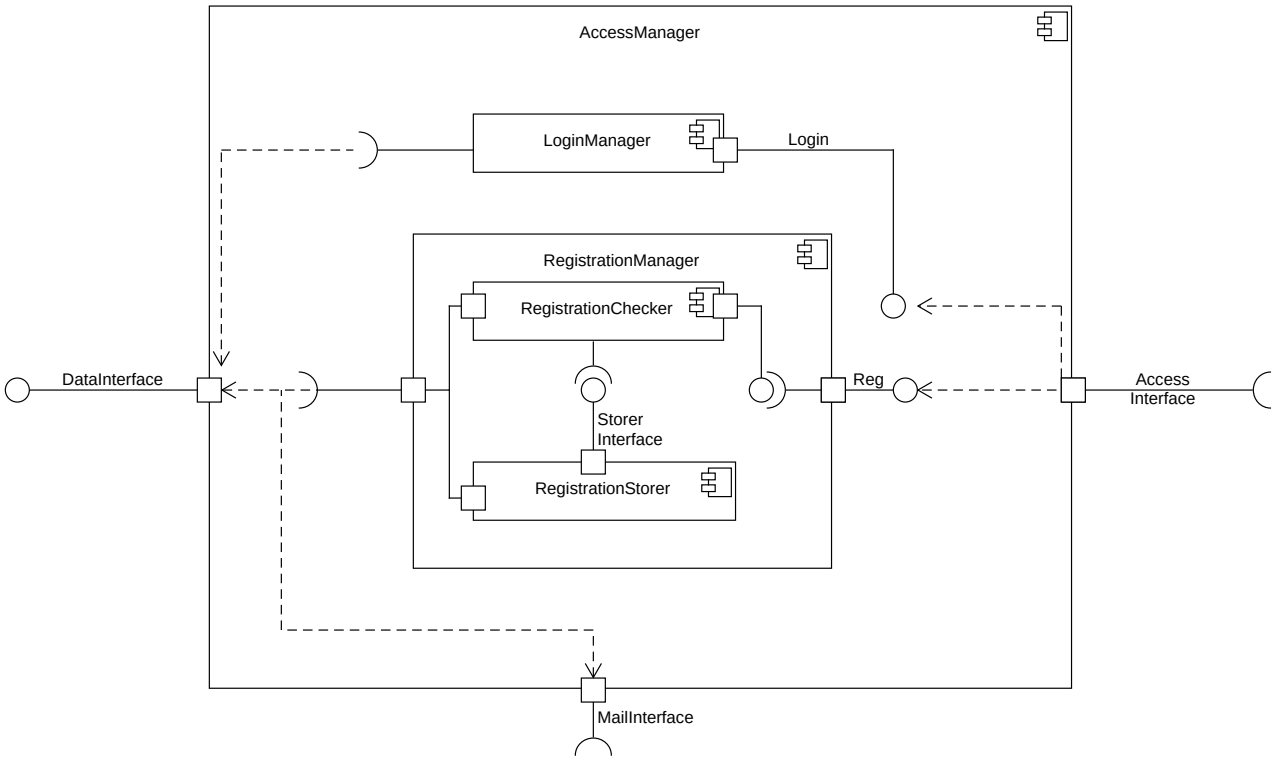

Figure 6: *Access Manager Component View*

2.2.2.3 Reservation Handler Component The Reservation Handler is the component that manages the customer's reservations. The component analyzes two main aspects:

- Suggestions
- Reservation Management

The component uses *ReservationInterface* (that implements *QueryInterface* and *SuggestionInterface*) to accept requests that are processed by this component , and **Store Component**, used to make requests to a specific store. This component is invoked when the customer tries to make or modify a reservation, or tries to retrieve his ones. Each time a reservation is retrieved, the customer will receive the updated information (such us *ETA* to enter the store).

- **SuggestionComponent**: it is the component that manages the suggestions that are made available to the customer at the time of reservation.

It will query each store registered on the platform, to retrieve availability information, depending on the preferences inserted by the customer and the type of reservation they are making (so, the component will use the right **Query Component**); it's invoked by both the *ReservationInterface* when a customer asks for suggestions on bookings, or by **ASAPQueryComponent** while the *ETA* for entering the store *ASAP* is too high.

- **StoreQueryComponent**: it is the component that manages the queries that are made at a particular store, indeed this component uses the **Store Component**. The queries they can be of two types:
  - **ASAPQueryComponent**: it is the component that manages the *ASAP* reservation, asks the server for waiting times for accessing the store, notifies the store if a reservation is made and manage the deletion of *ASAP* reservations.
  - **BookingQueryComponent**: it is the component that manages the booking via app, asking the server time slots to enter the store, in case a reservation is confirmed, notify the store to save it, and manage all the requests about modification and cancellation of booking reservations.

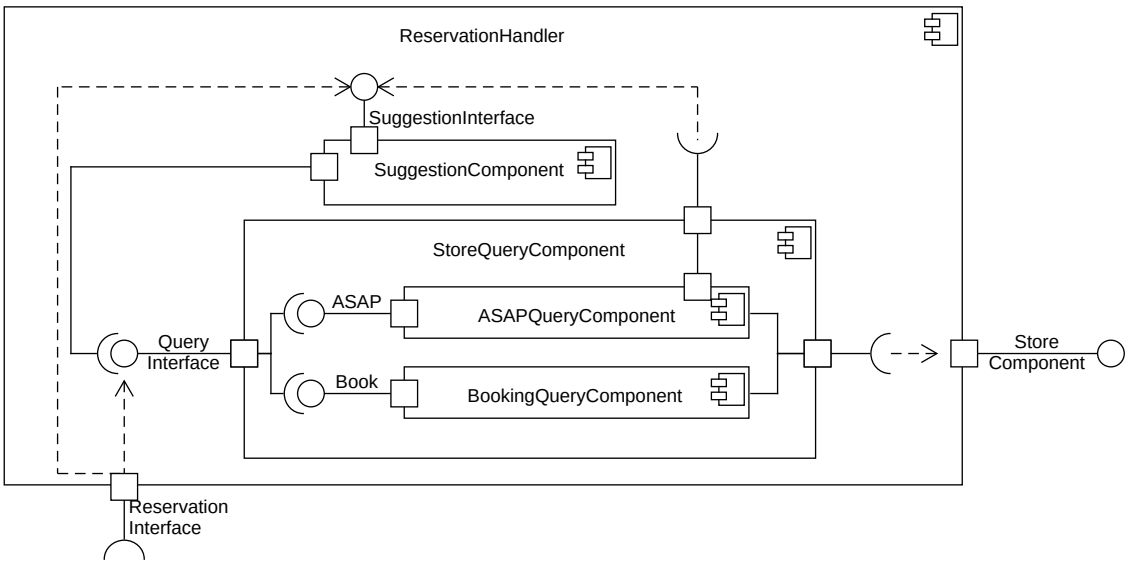

Figure 7: *Reservation Handler Component View*

2.2.3 Store Component

The Store is the component that manages the store from all points of view and it is the most important component of the whole architecture, since implements the whole business tier. The component analyze many important aspect, but the most important are:

- Reservation
- Admit Reservations Inside
- Store Parameters
- Statistical Values

As said before, it's been preferred to separate the logic governing each store by designing a component for each different store. Also here, the store component have been designed in a modular way, to separate the different features offered by a single store; these are:

- **FlowComponent**: this component manages the functionalities that are requested when a *QR Code* is scanned at the store entry, and uses the **Calling Number Component** to authorize the entry, and uses the **Department Component** to register the entrance/exit.
- **StatisticComponent**: it is the component dedicated to building customer statistics. It's used by:
  - **EntranceComponent**: in order to update visiting statistics at customers' exit
  - **ReservationComponent**: invokes the methods when, during a reservation process, it's necessary to infer the estimation of customers' shopping time.
- **RearrangeComponent**: it is the component that rearrange the reservations when a store manager modifies the capacity of some part of the store. Interacts with the **Calling Number Component** in order to rearrange the queue. In case of delays or cancellations, the customers will receive a notification.
- **StoreDataManager**: it is the adapter between the store and its dedicated *DBMS*. The choice of dedicating a *DBMS* for each store is due to efficiency and security reasons: if a *DBMS* broke, it wouldn't affect the whole system, while a store can access only its data.
- **ParameterManager**: it is the component that manages the parameters of the store, the settings and the situation of each department of the store. It's used by:

  - **EntranceComponent**: is used to update Departments' situation.
  - **StoreManagerHandler**: is used to update store parameters, and to access the real-time situation of the store.

  Moreover, uses **RearrangeReservation** when a parameter is modified, to make coherent the store's status

- **ReservationComponent**: it is the component that is dedicated to the management of reservations. It's used to retrieve reservations made in that store, to retrieve waiting times, and to store customers' reservations. Uses the **Calling Number Component** to notify the creation of a new reservation, and it's used by the latter to initialize the queue at store opening. It also handles modifications made by customers and store managers to reservations. Furthermore, a store manager uses it to contact customers about some specific reservation. Because of this, the components interact with **Notification Component** and **Mail Component** to interact with customers. To logically separate the management of the two types of reservations, the component is divided in two subcomponents: **BookingComponent**, and **ASAPComponent**.

- **CallingNumberComponent**: it is the component deputy to call in reservations, and to notify customers they should depart for the store. To do this, it interfaces with *MapInterface* to calculate travel time, and with *NotificationInterface* to send customers notification. It's divided in two subcomponents, each carrying out different functionalities:

  - **CallableReservationComponent**: checks whenever a reservation in the queue is callable, checking the store affluence, through the bfseries Parameter Manager Component; when it notice that a customer should be alerted to depart for the store, sends a notification through the **Notification Manager Component**.
  - **CallNumberComponent**: takes a queue of reservations to be called, and when it's possible, a reservation is called to enter. It's alerted by the **Entrance Component** that somebody exited the store.

The core store functionalities are requested through the *ReservationInterface* (dedicated to customers' features) and the *StoreInterfaces*, dedicated to store managers; their requests pass through an handler, here described. One note: the *StoreComponentInterfaces* implements all the interfaces of the single services requested by the **Store Manager Handler**.

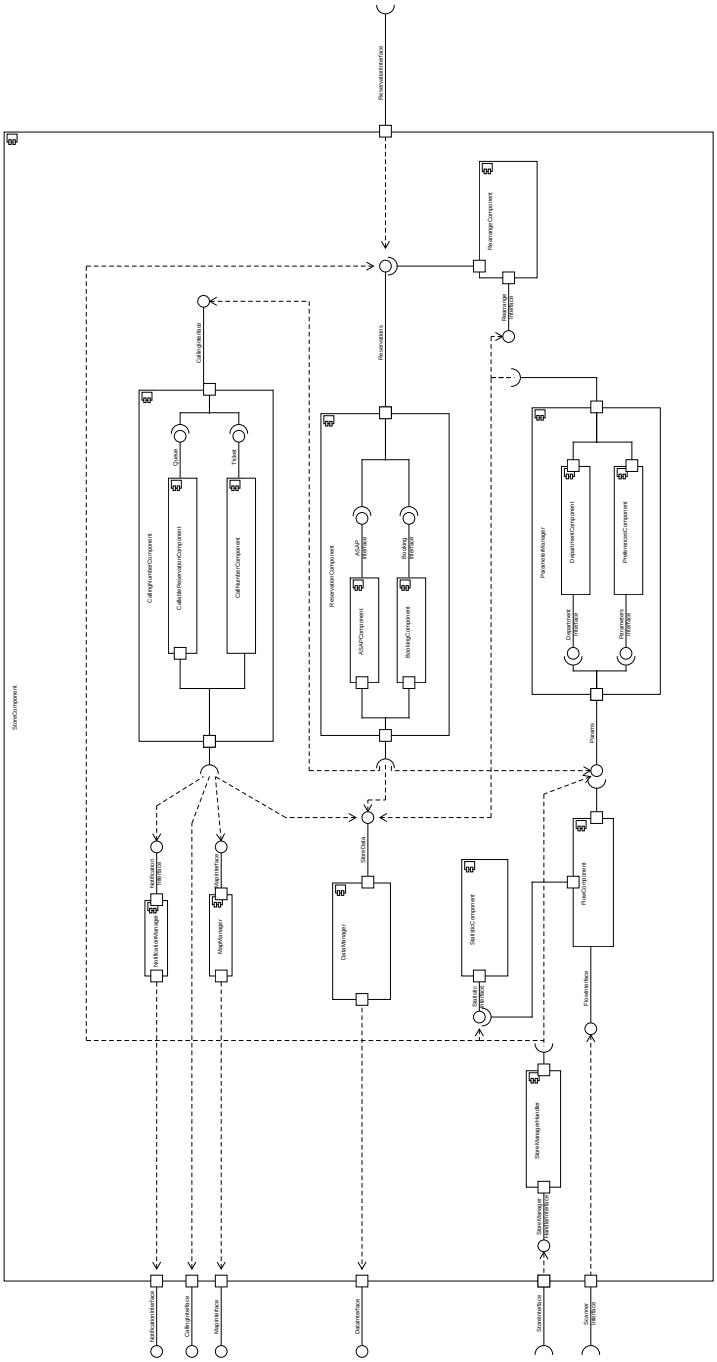

Figure 8: *Store Component View*

2.2.4 Store Manager Handler Component

The Store Manager Handler is the component that manages all the actions that a store manager can make within the application. The component analyze various aspects:

- Reservations
- Modify Parameters

This component handles all the possible settable options of a store manager, it is invoked on the *StoreInterface*, from the mobile application, and it interfaces with *StoreComponentInterface*, that manages all information regarding parameters and reservations.

- **ManagerReservationHandler**: it is the component that manages the store reservations and it is divided in two sub-component:
  - **DeleteRes**: it is the component that manages the elimination of the reservations.
  - **ModifyRes**: it is the component that manages the modification of the reservations.
- **ModifyParameters**: it is the component that manages the store parameter. They are of two types:
  - **DepartmentManager**: it is the component that manages all the parameters which concern the individual departments. Indeed, the store manager can add or delete a new department and can modify the maximum capacity of each department.
  - **StoreSettingManager**: it is the component that manages all the parameters which concern the entire store. Indeed, the store manager, can change the maximum capacity of the people inside the store and can modify the opening and closing hours.
- **RetriveReservation**: it is the component that manages all the reservation of a specific store.
- **SituationManager**: it is the component that manages the requests for the real time situation of the store.

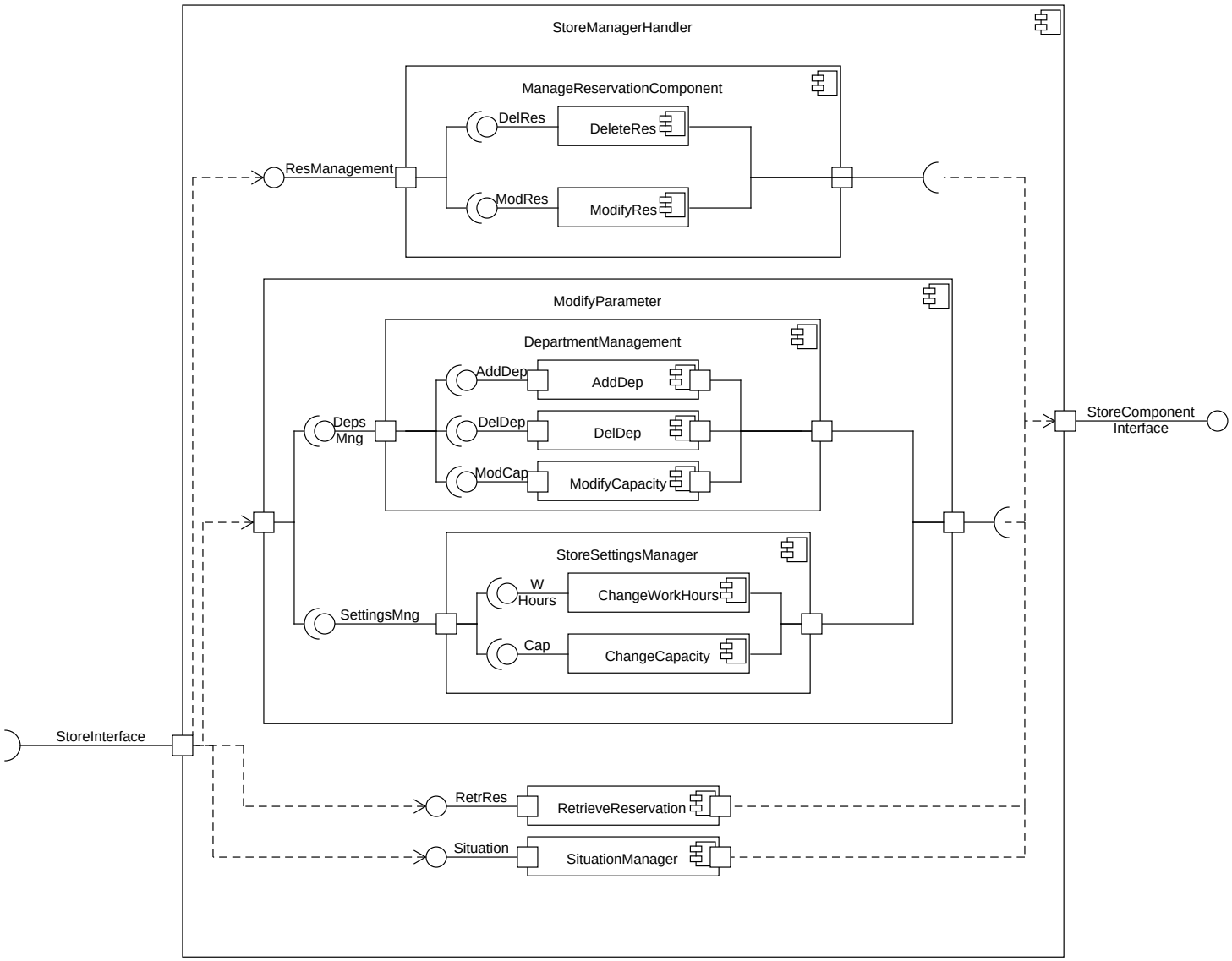

Figure 9: *Store Manager Handler Component View*