POLITECNICO
MILANO 1863

CLup - Customer Line Up

Software Engineering 2 Project 2020/21

Authors

- Francesco Attorre - *10618456*
- Thomas Jean Bernard Bonenfant - *10597564*
- Veronica Cardigliano - *10627267*

Deliverable:	DD
Title:	Design Document
Authors:	Francesco Attorre, Thomas Jean Bernard Bonenfant, Veronica Cardigliano
Version:	1.0
Date:	10-January-2021
Download page:	https://github.com/FrancescoAttorre/softeng2-attorre-bonenfant-cardigliano
Copyright:	Copyright © 2021, Francesco Attorre, Thomas Jean Bernard Bonenfant, Veronica Cardigliano - All rights reserved

Contents

1 Introduction

A Purpose

The goal of this Design Document is to provide a general view of the architecture of the software, adding more technical details to the information provided in the RASD document. The architecture will be explained in terms of hardware and software components, the interfaces they provide and the interactions between them. The document is going to face also the integration and testing plans, and the main design patterns to be exploited. Information related too much on the implementation will not be dealt in detail since this document aims to the description of an high level architecture of the system.

B Scope

CLup is an application which aims to manage crowds in a period of global pandemic. It allows store managers to monitor the entrances in a building and to avoid the the formation of queues in front of stores. CLup will allow people to save time and be safer, managing both the accesses of people who use the application directly, and of people who go physically to the store, using store managers as intermediaries. CLup can be accessed by customers, store managers and activities which will register stores available to be visited. Customers using the application, moreover, will be notified when they're supposed to leave for the store, and, when registered, they will have the possibility to book a visit to a store, being informed on alternative time slots or similar stores when nothing is available for their choices. Booking a visit, customers can decide how much time they want spend in the building and, optionally, which specific departments they want to visit, in order to allow a greater number of people to enter when possible.

C Definitions, Acronyms, Abbreviations

C.1 Definitions

- User: with this term we refer to StoreManagers, AppCustomer, Activities which are the ones who use the mobile or web application and its services. They will be associated to an ID in order to be recognized by the system.

C.2 Acronyms

- RASD: Requirements Analysis and Specification Document
- GPS: Global Positioning System
- CLup: Customers Line-up
- DD: Design Document
- HTTP: HyperText Transfer Protocol
- TLS: Transport Layer Security
- JSON: JavaScript Object Notation
- DBMS: Database Management System
- JDBC: Java DataBase Connectivity
- API: Application Programming Interface
- REST: REpresantional State Transfer

C.3 Abbreviations

- Rn: requirement number n
- Cn: component number n

D Revision History

In a first version of the document, a major number of components were created, but then has been considered more appropriate to group them into a few main macro-components with different functions within them.

E Reference Documents

- Specification Document: "R&DD Assignment AY 2020-2021.pdf"
- Slides of the lectures
- UML diagrams: https://www.uml-diagrams.org/
- Database Administration: https://galeracluster.com/library/documentation/deployment-variants.html

F Document Structure

This DD in composed by 7 main sections:

- SECTION 1 is the introduction, containing the scope and the purpose of the system, together with Definitions, Acronyms, Abbreviations, the revision history of the document deployment, the reference documents and the document structure.
- SECTION 2 contains the architectural design of the system, described in terms of components, runtime view with sequence diagrams showing the way the various components interact with each other, the deployment view, component interfaces and the related class diagram and finally an overview of patterns and architectural styles used, plus other design decisions.
- SECTION 3 contains the user interface design, in particular some mockups show the main mobile and web application interfaces.
- SECTION 4 contains requirements traceability showing how the requirements described in the RASD map the design components identified in this document. This table clearly shows if all the components cover at least one requirement and if each requirement is met by at least one component.
- SECTION 5 concerns the implementation, integration and testing. Here it is defined how the subcomponents should be implemented and integrated and which kinds of tests should be carried out on them.
- SECTION 6 contains a table with the effort spent by each member of the group.
- SECTION 7 tools used.

2 Architectural Design

A Overview: High-level components and their interaction

The architectural style chosen to develop the system is a three layer architecture, with a layer of Presentation, one of Business logic/Application and one of Data. This style has been chosen since it allows an easy decoupling of logic and data and of logic and presentation. The presentation level is the one which handles interactions with users, with the interfaces to communicate with them. The Business logic consists in the functions provided to the users. Moreover, this layer handles the communication between the other two layers. The data access layer, instead, manages the access to the database both for storing and retrieving data for the other layers. The hardware architecture chosen is the three-tier one. An advantage of this architecture is that the client tier doesn't communicate directly with the DBMS, so the middle tier guarantee a major level of security. Moreover, in this way the connection with the DBMS will be persistent and consequently less expensive. These application layers are divided into three physical dedicated machines. A mobile device/pc, basing on the type of user, that is a personal computer for the activities and a mobile phone for customers and store managers, is used to interface with the user. The Business logic, instead, is the application server which communicates with the DatabaseServer.

B Component view

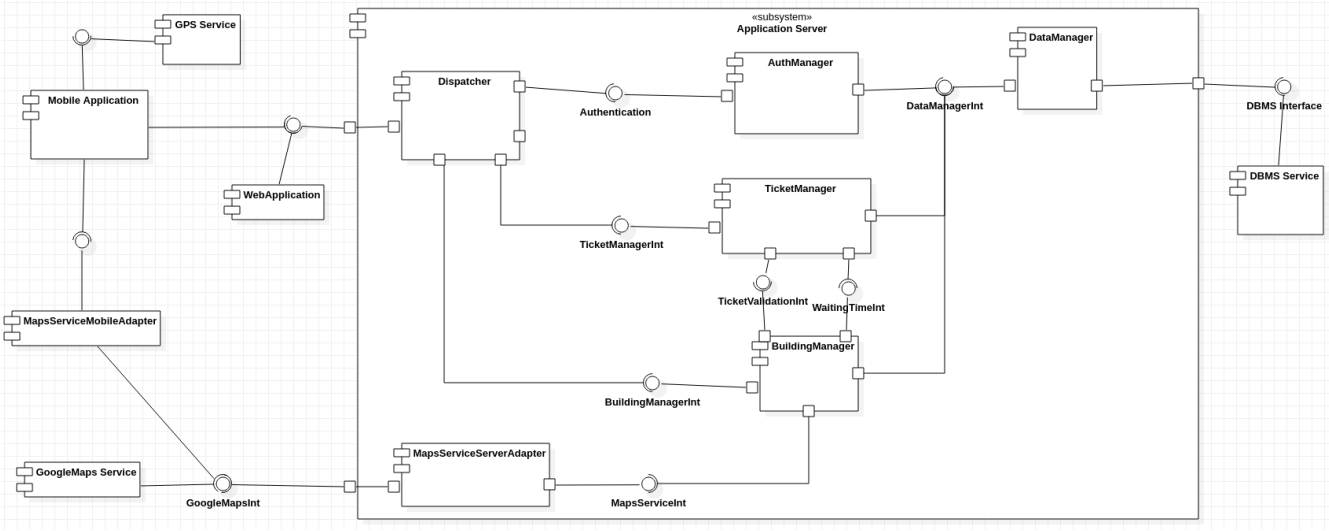


Figure 1: Component Diagram

All the components inside the subsystem "Application Server" are stateless components. Below is provided a description of how they work.

TicketManager

This component provides services to deliver new tickets and communicates with BuildingManager to associate them with a Building. It creates new line up and booking tickets. It manages ticket states and retrieves information about them when required.

AuthManager

This component offers services to authenticate users and deliver them tokens that authorize to use other services. It checks tokens in order to understand whether a user has the authorization to access a requested service, checking also the token validity. The result of this check'll be communicated to the Dispatcher component that'll forward the request to the specific component that provides the service.

BuildingManager

Provides services to add buildings to the system, track the number of people in specific buildings and

manage queues. It appends and removes line up tickets from queues, computes the available slots for a booking ticket and suggestions of alternative buildings and time slots (advanced functionality). It also checks the building capacity to manage the influx of customers, generates the building access code and computes the estimated waiting time for each customer having a digital ticket. This component is in communication with the TicketManager in order to validate a ticket when it's the first one in queue.

GoogleMapsService

External service component that provides web mapping. Used to locate Buildings given their address and compute travel time to reach them with a predefined means of transport. It'll be exploited both by client and server, and having an Adapter component as intermediary with the system it could be easily replaced.

Dispatcher

This component manages the requests received from MobileApplication and WebApplication components and redirects each one to the correct component able to handle it. It interacts with the AuthManager in order to verify if the user is authorized to exploit the requested service, if the check gives a positive response, it proceeds redirecting the request. So, it knows which services and components will handle each request.

MobileApplication

Acts as a Client sending requests to the Dispatcher component. It is a fat client since it contains applicative logic about computation of travel time and sending notifications to the user. It also manages the association of physical tickets with digital ones when used by a store manager.

WebApplication

Web Client sending requests to the Dispatcher like the Mobile Application. It will be thinner than the Mobile Application taking part only at the Presentation Level since it does not contain any Applicative logic. WebApplication is used only by Activities.

DataManager

Manages data structures and object relational mapping. It communicates with DBMS service in order to persist objects in the relational Database.

DBMS Service

Represents the DBMS and the service he offers. It communicates with the DataManager with standard protocols in order to retrieve or get data.

GPS Service

Positioning system used to locate a device. This is an optional component and it would be offered by an external software (Android/iOS) already present in the user device.

MapsServiceMobileAdapter

Adapter that offers to the MobileApplication component an interface for using Mapping services. Useful if the external Mapping service changes since the Mobile Application will always use the same interface.

MapsServiceServerAdapter

This component is an adapter like the previous one, but it is used by the BuildingManager component. Having two different Adapters offers the possibility to use different Mapping services for MobileApplication and BuildingManager.

C Deployment view

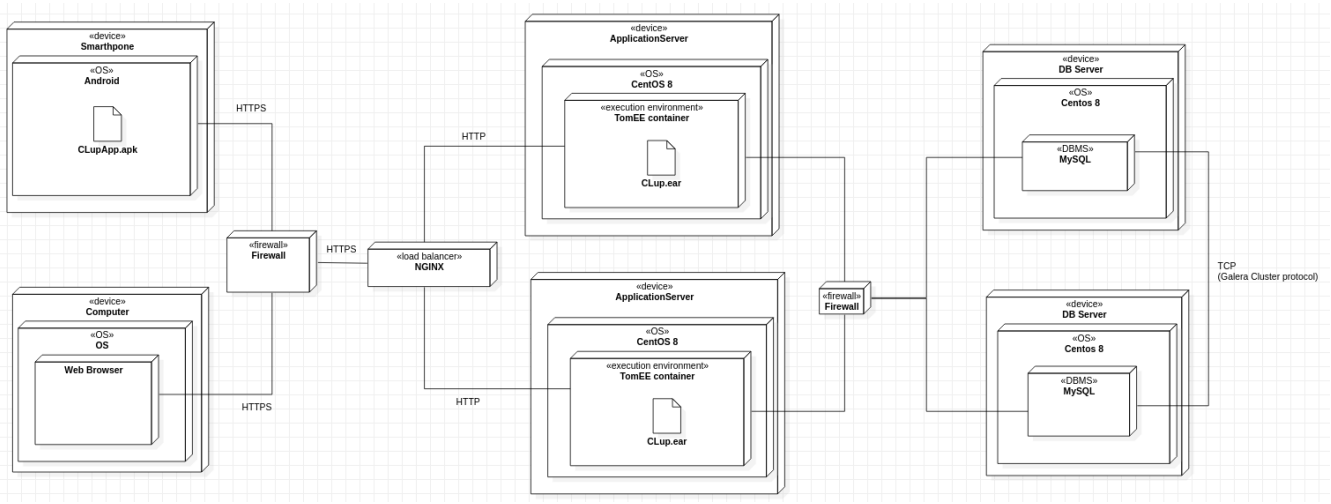


Figure 2: Deployment Diagram

This diagram shows the overall deployment architecture. Here is a description of every element:

- **Smartphone**: Device used by AppCustomers and StoreManagers. It will have to contain the CLup App executable in order to use the system.
- **Computer**: Device used by Activities in order to use the WebApp.
- **Firewall**: Provides safety access to the internal network of the system as part of the safety of the system against external attacks.
- **Nginx**: Nginx server used to balance the load on the multiple Application servers. It is responsible to establish an encrypted connection with the clients through Transport Layer Security (TLS). From this point on the connection will be unencrypted as we consider the internal network as secure.
- **ApplicationServer**: Contains most of the Application Logic. Because of the stateless behavior of the server side logic Application Servers can be easily added to scale out and maintain high availability and reliability. The CLup.ear artifact is the implementation of the ApplicationServer subsystem found in the Component Diagram.
- **DB Server**: It implements the DBMS Service in the Component Diagram. A more detailed description is provided in section G.2.

D Runtime view

Here are proposed some sequence diagrams in order to describe the way components interact with each other to accomplish specific tasks.

Premise

It's not explicitly stated in every sequence diagram that a token checking is performed for every request done from Clients to the Server. In fact, each request (except for authentication requests) should contain a token in order to authorize the sender. The Dispatcher, before processing the request just received, will perform on AuthServer a token validity check, which comprises a verification of services that a Client can request on the Server.
Instead, the identification action performed in some diagrams has the task to retrieve the User information, usually an identifier.

D.1 Authentication Sequence Diagram

Two different authentication procedures can occur, the first is an authentication on a RegisteredAppCustomer, instead a daily access to CLup services can be requested by UnregisteredAppCustomer.
Description
Both of the following processes lead to gain a token, that is a temporary access key for Server functionalities. The second one is Daily in the sense that every operation on server would stay for at most one day.
account based - an authentication request is made by the MobileApplication, after correctly forwarding this request to AuthManager an account check is done. If the user exists and has no token already associated with its account, then a new one will be created. MobileApplication receives a response containing his token.
daily - a daily authentication request is made by the MobileApplication and, after correctly forwarding this request to AuthManager, an access check is performed. If the client can get a token in order to access line up services, a new association between a specific MobileApplication and the new token is made, otherwise a negative response is sent back to the client.

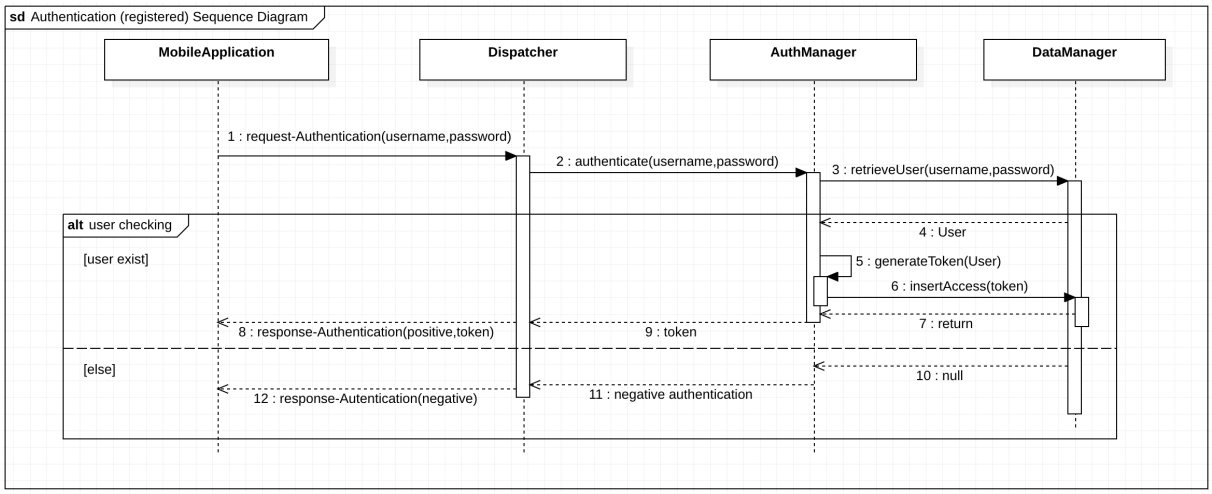


Figure 3: Authentication (registered) Sequence Diagram

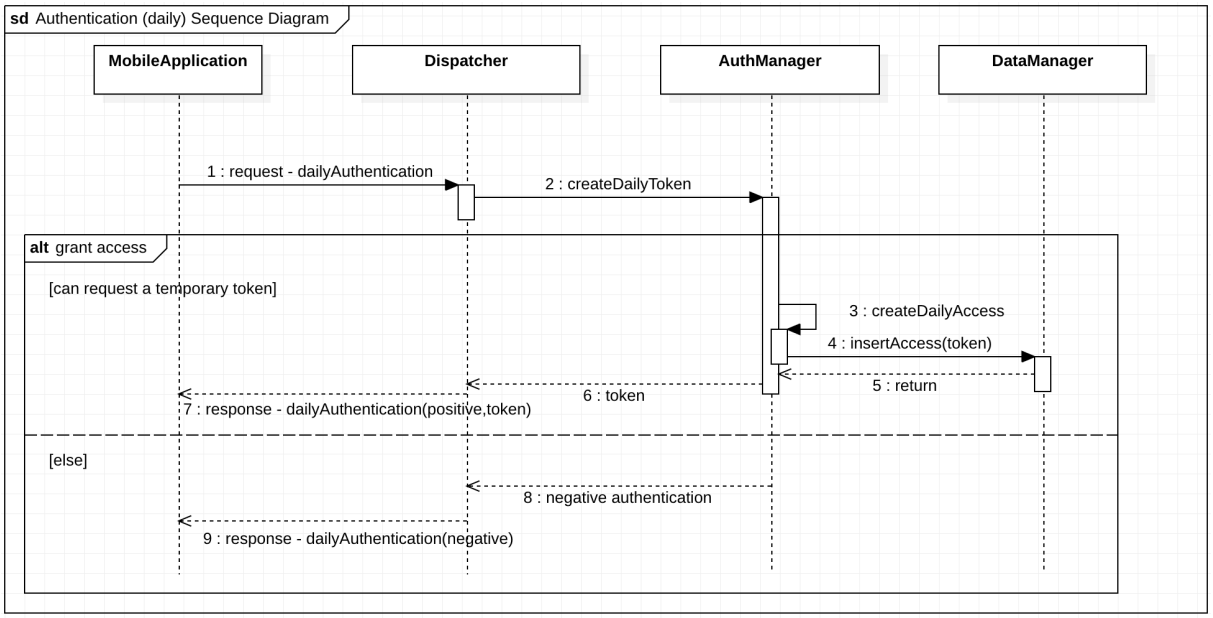


Figure 4: Authentication (daily) Sequence Diagram

D.2 Acquire a LineUpDigitalTicket Sequence Diagram

This diagram explains how an AppCustomer can acquire a LineUpDigitalTicket.

Description

MobileApplication retrieves a position (manually or through GPS) and sends a request in order to receive a list of buildings from which one would be selected. Building retrieval is effectively performed by the DataManager, whereas the MapsServiceAdapter (communication with a maps API) provides to the BuildingManager useful information to correctly pick only reachable buildings.

Once the building is selected, MobileApplication sends another request in order to gain a ticket. An identification is performed on AuthManager to correctly retrieve the UserID (associated to the token) needed in ticket acquisition. Afterwards, to complete the process, if the building has at least one empty space the ticket state is set to Valid, else the ticket is inserted in queue. If ticket insertion goes wrong, a negative acknowledgment is sent back.

Note that only after a Discovery request from the MobileApplication, this ticket will produce a notification (as described in the Discovery diagram).

Variant

MobileApplication ticket request performed by a StoreManager works in the same way, but building list is not requested since it is constrained by the one for which StoreManager is signed in.

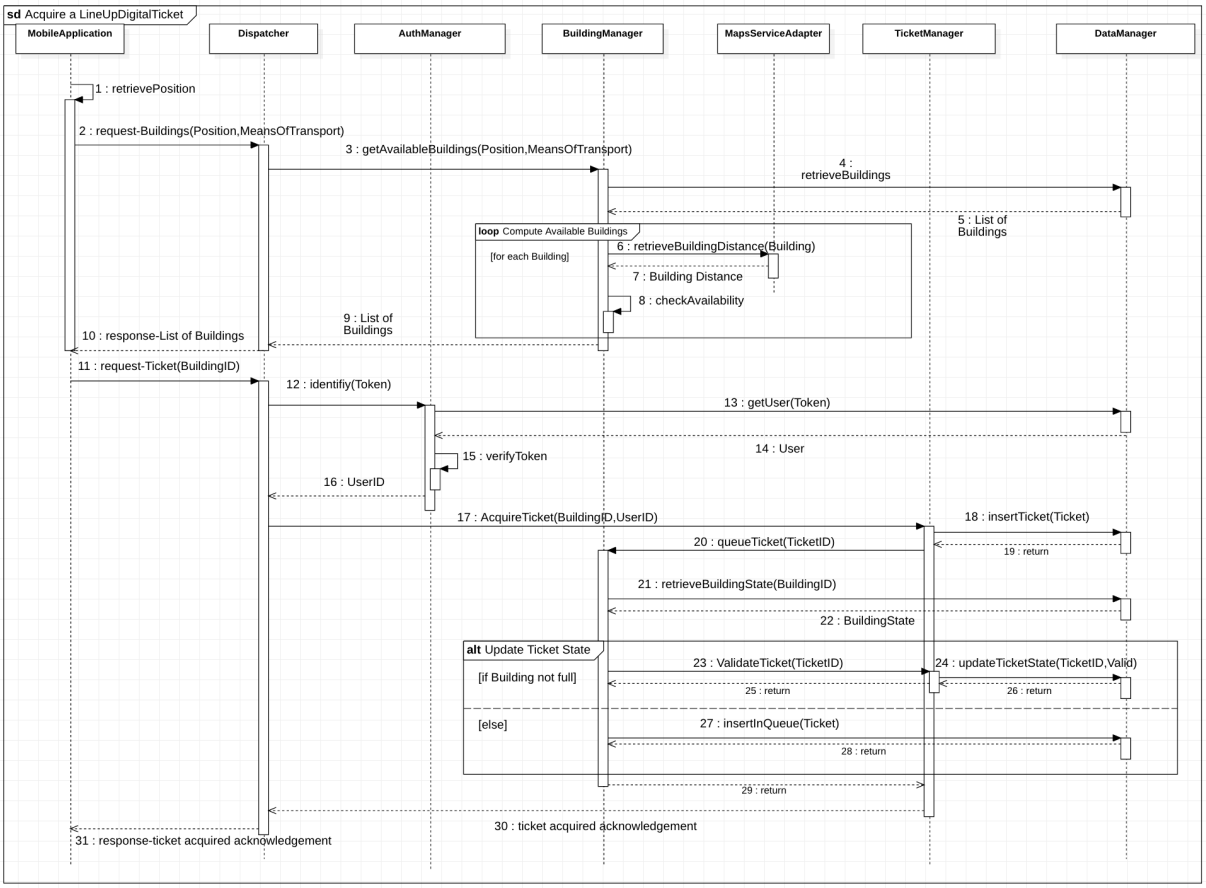


Figure 5: Acquire a LineUpDigitalTicket Sequence Diagram

D.3 Acquire a BookingDigitalTicket Sequence Diagram

This diagram explains how a RegisteredAppCustomer can acquire a BookingDigitalTicket

Description

MobileApplication retrieves a position (manually or through GPS) and sends a request with the purpose to receive a list of buildings from which one would be selected. Building retrieval is effectively performed by the DataManager, whereas the MapsServiceAdapter (communicating with a maps API) provides BuildingManager useful information to correctly pick only reachable buildings.

Once the building is selected, MobileApplication sends another request in order to have a list of TimeSlots. Dispatcher forwards this request to the BuildingManager that primarily retrieves Building which capacity information is needed to compute TimeSlots availability for a specific date.

Lastly MobileApplication requests the Server to gain a ticket. An identification is performed on AuthManager to correctly retrieve the UserID (associated with the token) needed in ticket acquisition. Then a general check validity is performed by BuildingManager, non valid tickets will be not acquired. An acknowledgement is sent back to MobileApplication.

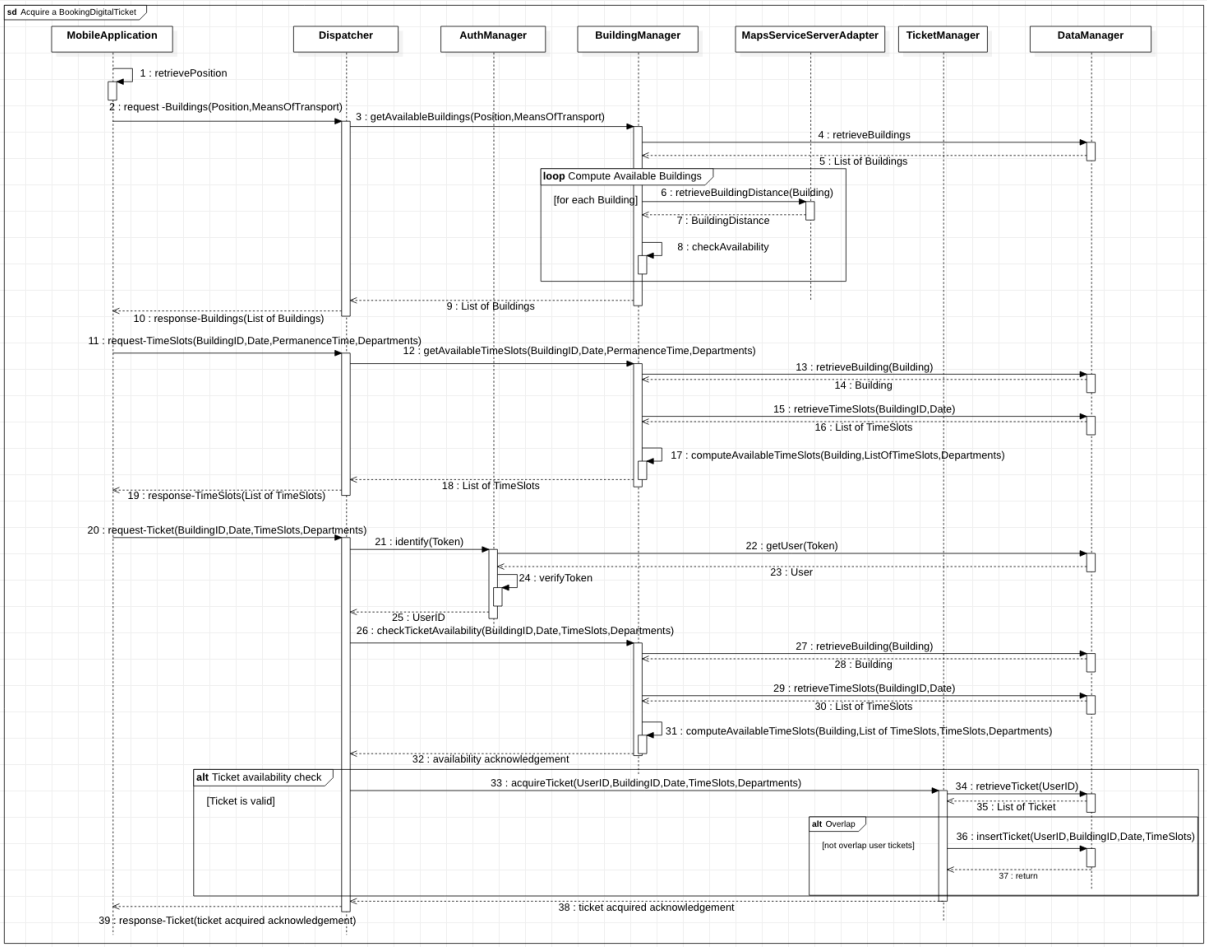


Figure 6: Acquire a BookingDigitalTicket Sequence Diagram

D.4 Discovery Sequence Diagram

This diagram explains how an AppCustomer can be notified about when to start reaching a Building and about its ticket validity or a StoreManager when to inform validity of some PhysicalCustomers's Tickets.

Description

A Discovery request is a polling request made by a MobileApplication in order to receive updates about Tickets. The Dispatcher first identifies the User and then gets an updated waiting time estimation from the TicketManager.

The waiting time computation process is made by means of a statistic (peculiar of each Building and kept up to date on every exit event) and the related position in queue. Waiting time depends also on exiting delays and it would have a particular rounding. Ticket and estimated waiting time for each ticket are sent back to Mobile Application, that will start checking all of them.

For AppCustomers a notification will be received based on the actual position and on ticket estimated waiting time, instead a StoreManager will check only for just validated tickets.

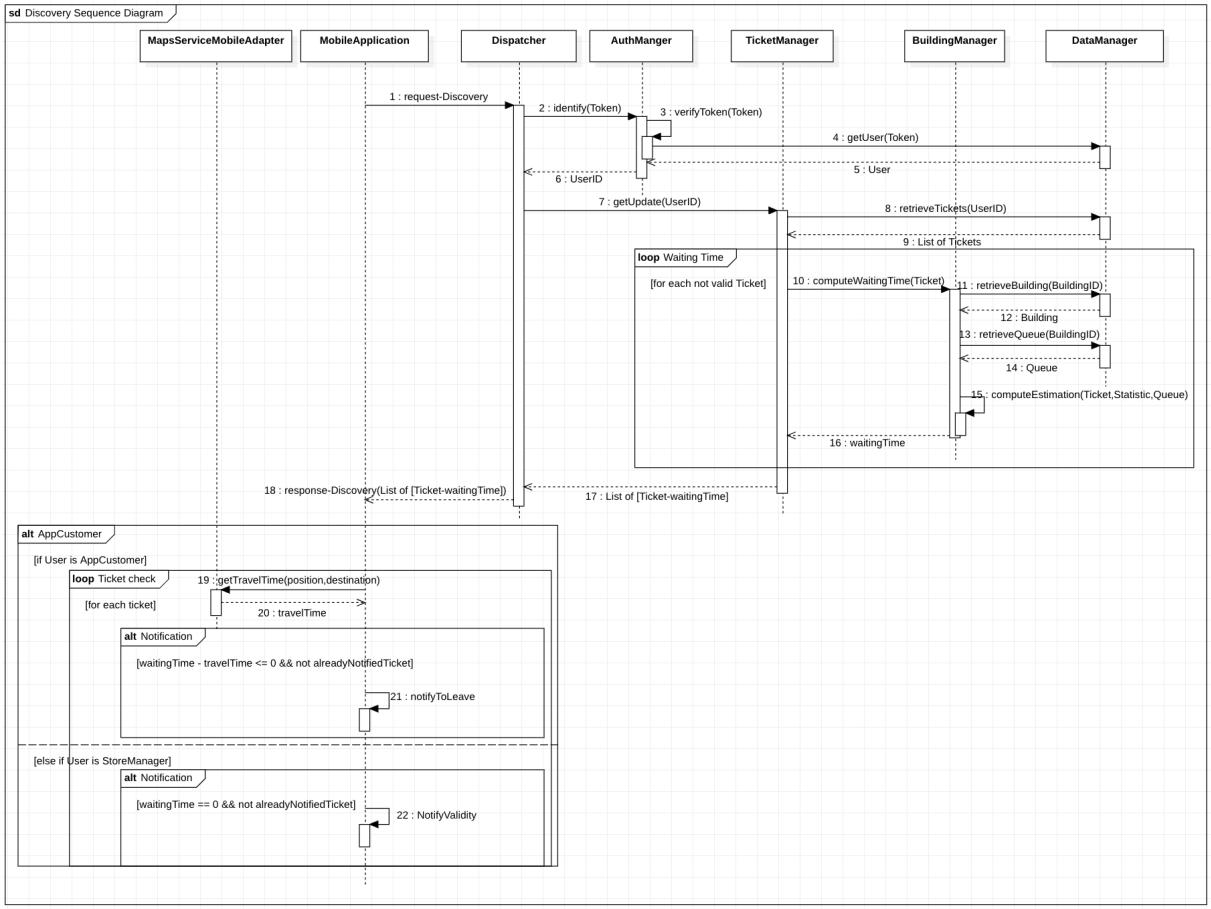


Figure 7: Discovery Sequence Diagram

D.5 Customer Exit Building Sequence Diagram

This diagram explains how a StoreManager can notify CLup about a Customer leaving the Building.
Description
MobileApplication exit request is sent to the Dispatcher, able to identify the User (the StoreManager associated with the Building for which he is signed up for), then the request is forwarded to the BuildingManager.
The latter will keep statistics up to date (related to the average time between two exits) as well as the last exit time for that specific building, then the ticket validation process will start. Queue is retrieved from DataManager and the next ticket in queue (if there is one) will be validated. An acknowledge for successful processing is received by Dispatcher and then MobileApplication.

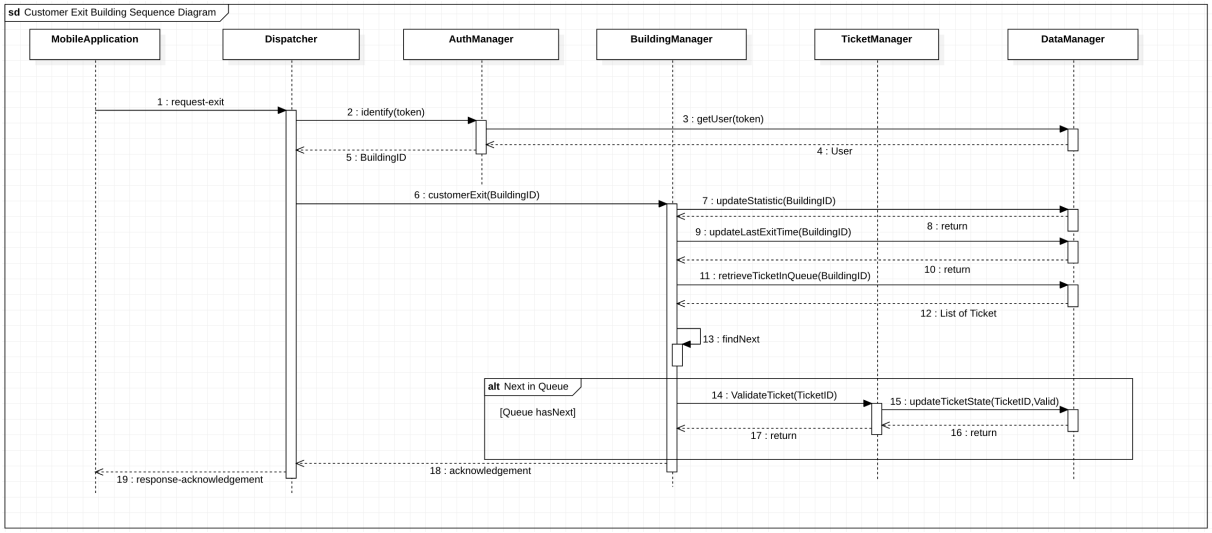


Figure 8: Customer Exit Building Sequence Diagram

D.6 Activity Insert Building Sequence Diagram

This diagram explains how an Activity inserts a new Building in CLup.

Description

WebApplication requests to insert a building providing some parameters, and optionally departments with surplus capacity. Dispatcher receives the request, identifies the user (that should correspond to an Activity) and then forward the insertion request to the BuildingManager, that in order to correctly check if the building can be inserted, it retrieves a list of all buildings and the location, through the MapsServiceServerAdapter, given by the address. At this point if the Building is valid it is persisted. An acknowledgement is sent back to WebApplication.

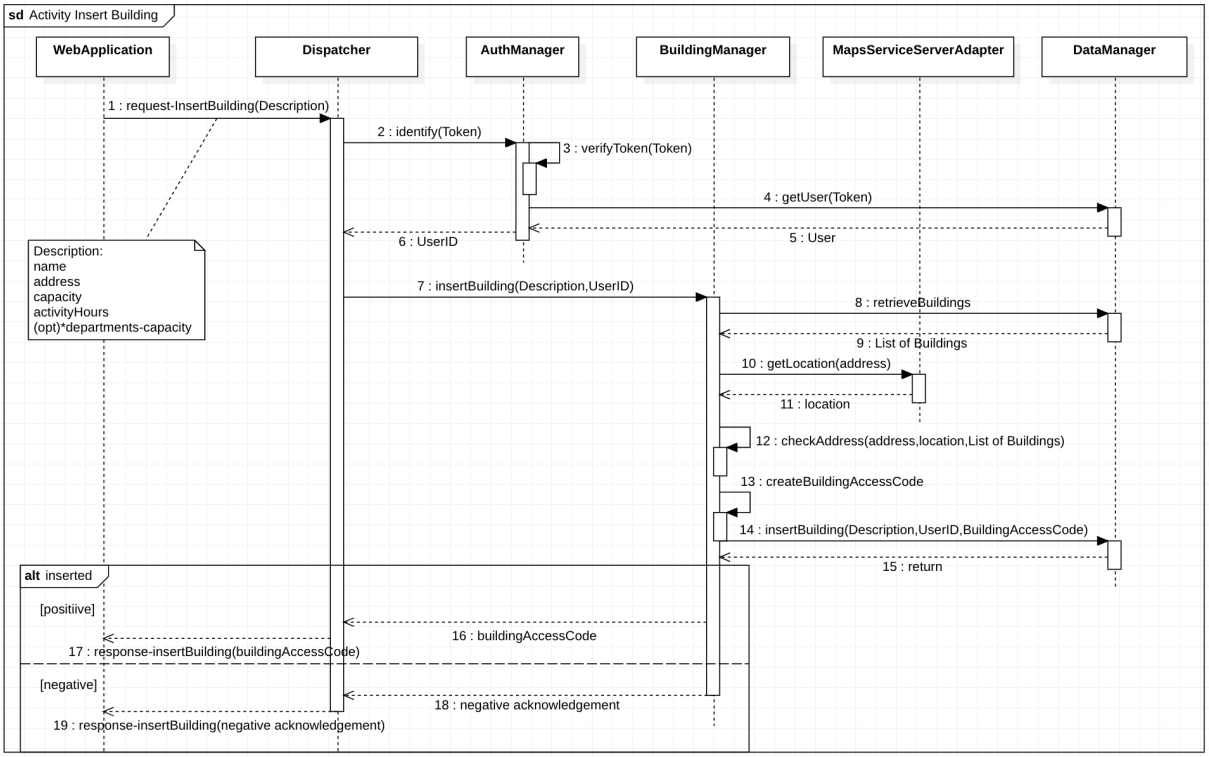


Figure 9: Activity Insert Building Sequence Diagram

E Component Interfaces

The following diagram shows all the component interfaces already exploited in the sequence diagrams together with the dependencies between the various components.

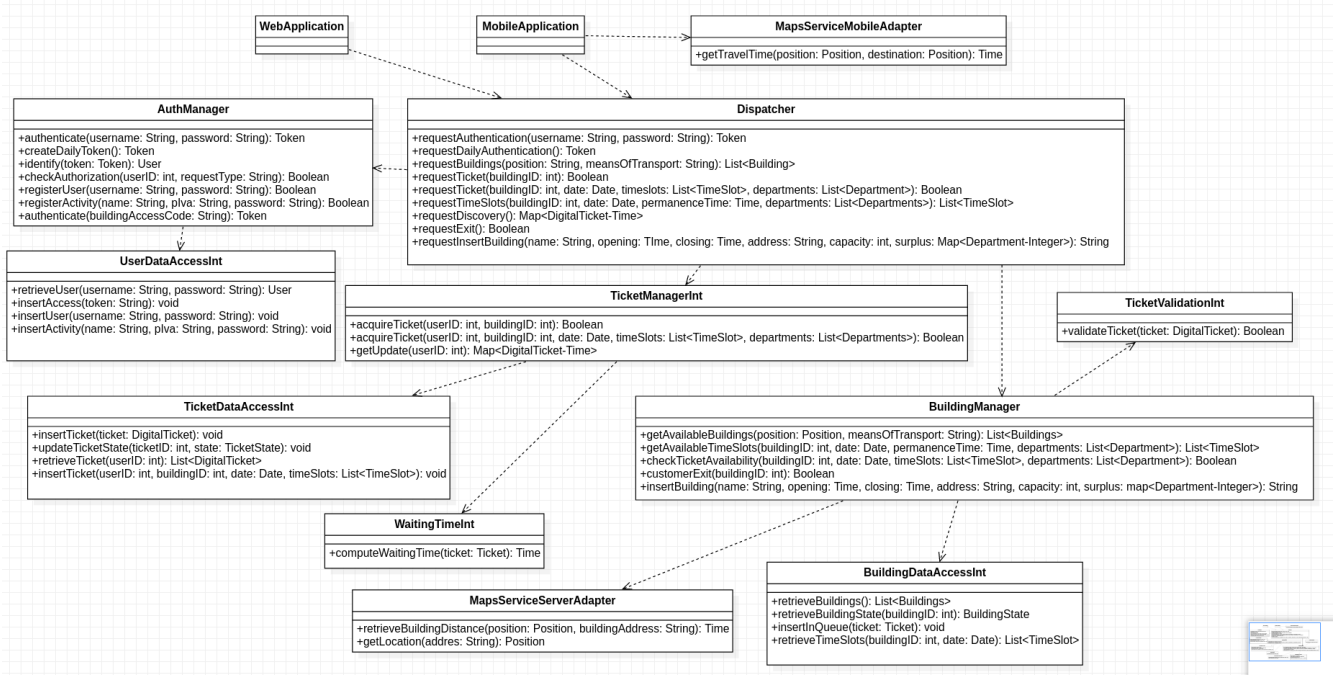


Figure 10: Component Interface Diagram

F Class Diagram

An updated version of the class diagram is shown below, with the various types used by the interfaces.

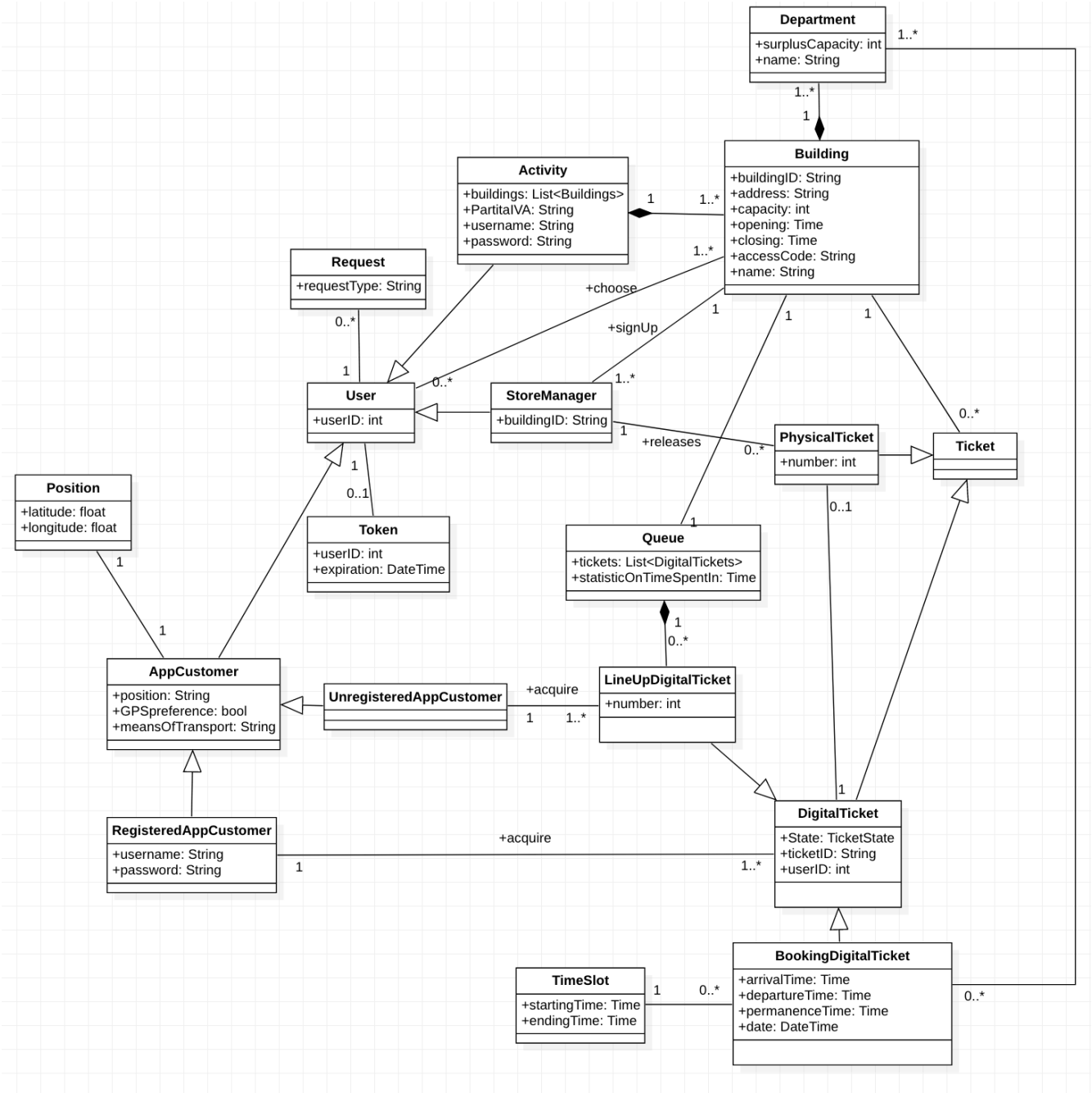


Figure 11: Class Diagram

G Selected architectural styles and patterns

The architectural style selected is a three-tier client-server architecture, in order to have a good decoupling of logic, data and presentation, increasing reusability, flexibility and scalability. Moreover, components in the application server have to be developed mainly with low coupling among modules in order to make the system more comprehensible and maintainable. About the components, they have been designed to maintain a stateless logic as much as possible, that is, they should not contain an internal state, but refer to the database to get the necessary information. This is important since instances of components can fail and nothing must go lost in this eventuality. In this context the scalability of the database is very important, and DataAccessManager will play a leading role. The protocol used to send requests is HTTP, which is a good choice to implement a RESTful architecture to meet the above objectives of having a stateless and low coupling system. Other advantages are that it would be cacheable and with a uniform interface. To ensure a secure and reliable communication between client and server, HTTPS is used with TLS encryption. Data are transmitted in JSON, which is one of the simplest and most easily customizable protocol. It is also easy readable and allows fast parsing.
Finally, we've decided to use some design patterns in order to exploit existing models to solve recurrent problems. This benefits the reusability and maintainability of the code, as well as making it easier for designers to understand how the system works. Below the patterns used:

G.1 Model View Controller - MVC

MVC is a widely used pattern, particularly suitable for the development of applications written in object-oriented programming languages such as java. MVC is based on three main roles which are: the Model that contains all the methods to access the data useful to the application, the View that visualizes the data contained in the model and deals with the interaction with users and the Controller which receives the commands of the user and executes them them modifying the other two components. In CLup, the Controller logic is in the Dispatcher component, the Model is represented by services offered by the other application server's components, plus the DBMSAccessService, instead the view is in the Mobile and Web Applications.

G.2 Adapter pattern

Adapter is a structural pattern that aims to match interfaces of different classes. The interface of the Adapter is interposed between the system and the Adaptee, that is the object to be adapted. In this way, whoever has to use a method of the Adaptee sees only an interface (or an abstract class) which would be implemented according to the component to be adapted. In the case of CLup, the components MapsServiceMobileAdapter and MapsServiceServerAdapter act as the adapter for the mobile application and the server, and GoogleMapsService is the component to be adapted. In this way, even assuming that the external service is changed, the internal system will not undergo any changes, since the new API will be handled by these components.

G.3 Facade pattern

Facade is also a structural pattern, which consists in a single class representing the entire subsystem. In the case of CLup, the Dispatcher takes all the requests from the client and then directs them to the specific component of the AppServer. The aim of this component is to mask the complexity of the entire subsystem, with which you can communicate via a simple interface.

H Other design decisions

H.1 Thin Client

Having most of the business logic on server side, our client can be defined thin, although there are small pieces of business logic also in the client side. In fact, the client has a MapsServiceMobileAdapter to access the external component in order to compute independently the travel time in a real time way, in case it has decided to activate the GPS services. An advantage of this choice is a lighter communication with the server, which is not constantly updated on the position of the client when not necessary. Also the control on when to notify the client to leave for the building is made client side, given such information about position and time to reach it, avoiding to overload the server. Another advantage of having a thin client with a little of business logic is that, having an application which works mainly online, in any case it will always be connected to the server having the main logic, but even if the connection would be interrupted, some services encapsulated in the client will still remain available.

H.2 Database

We will use Galera Cluster, a synchronous replication solution, transparent to the Application Servers to improve availability and performance of the DBMS Service. The Cluster will be deployed in a distributed load balancing configuration: each Application server will have a JDBC connector configured for load balancing; this avoids to have between the Application Servers and the Database Servers a separated load balancer that could be a single point of failure (if an ulterior backup load balancer is not added) and a potential bottleneck.

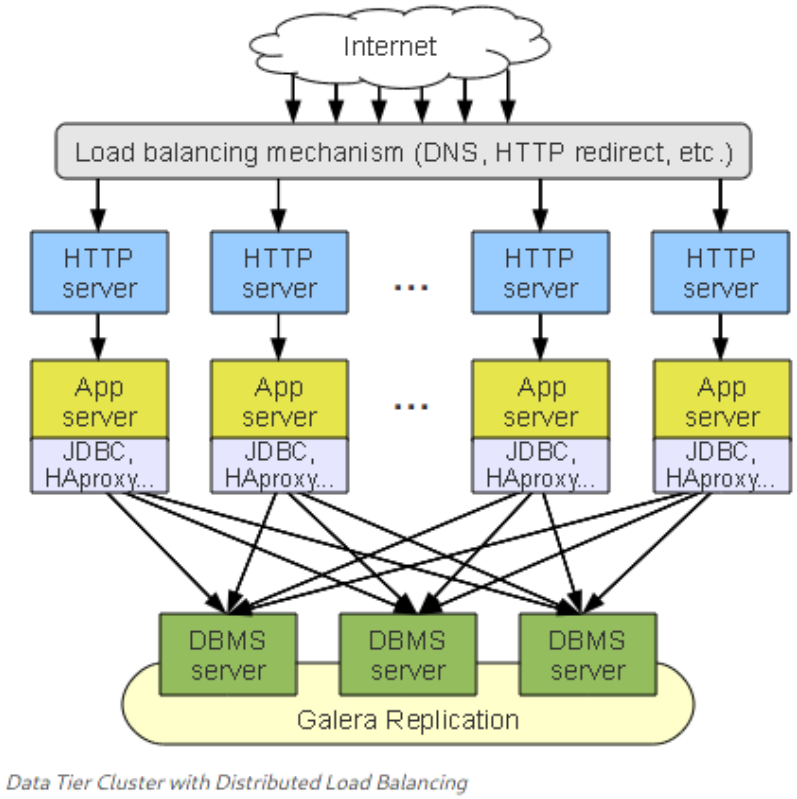


Figure 12: Galera Cluster's configuration

3 User Interface Design

A UserMobileApp Interfaces

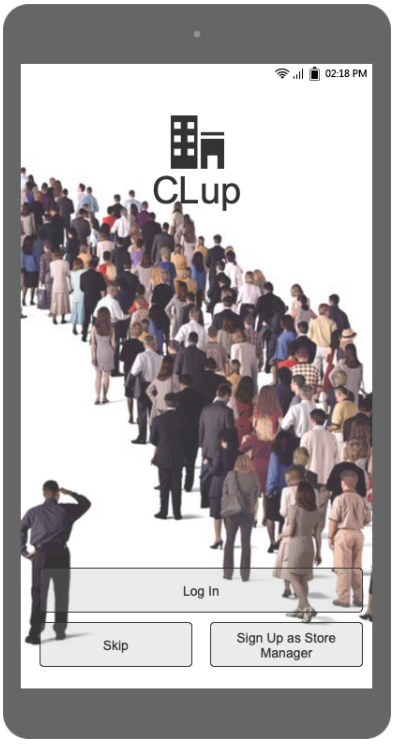


Figure 13: Starting page

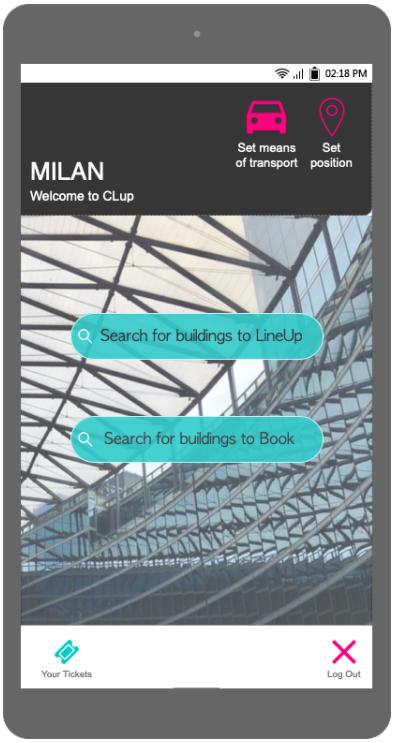


Figure 14: General home page of a registered customer

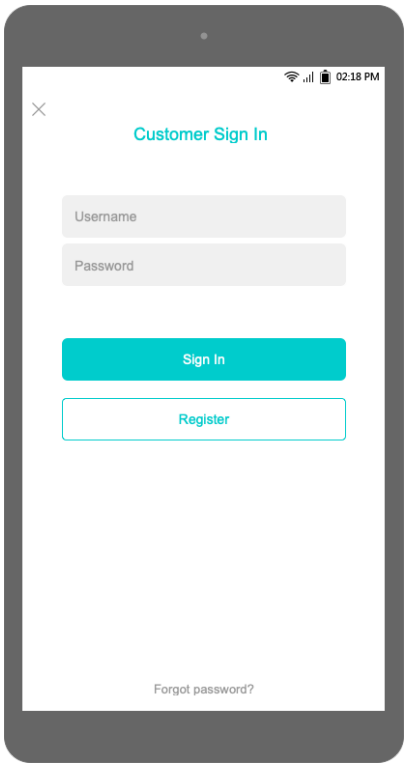


Figure 15: Interface for customers that want to sign in

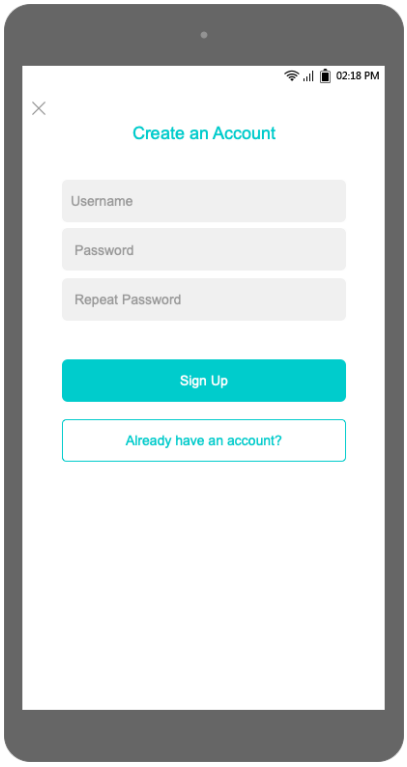


Figure 16: Interface for customers that want to register to CLup

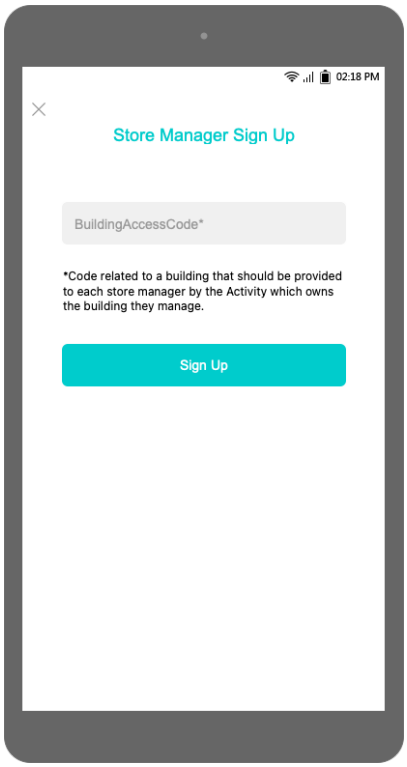


Figure 17: Interface for store managers to sign up

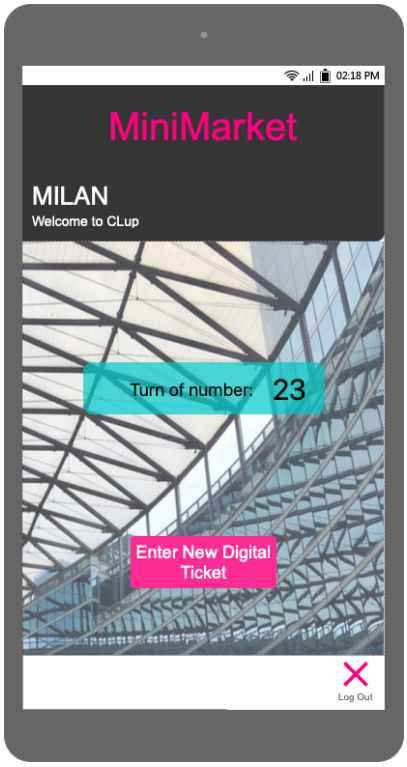


Figure 18: Home Page for store managers that have signed up

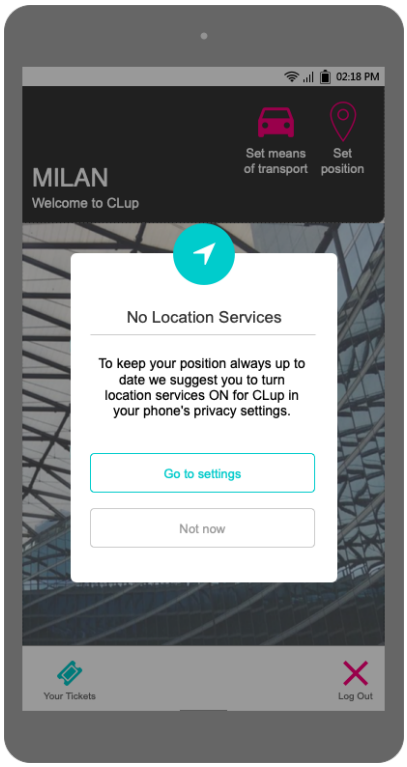


Figure 19: Pop-up to activate GPS services

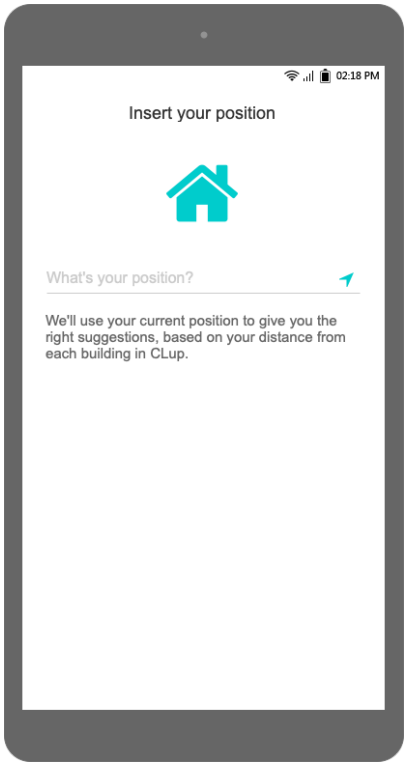


Figure 20: Setting page to set a position manually

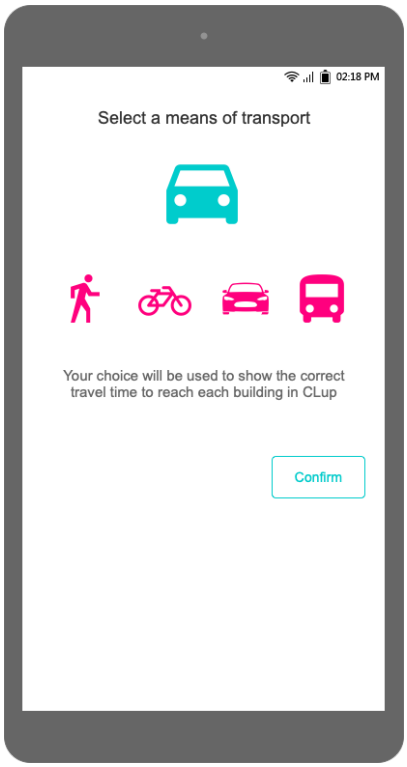


Figure 21: Setting page to choose a means of transport

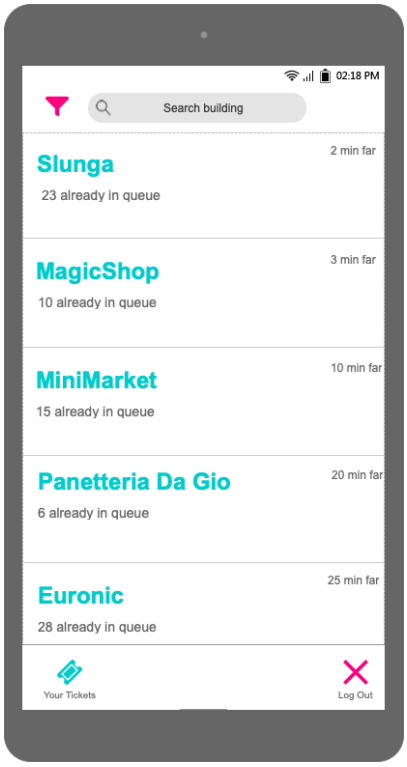


Figure 22: List of available buildings

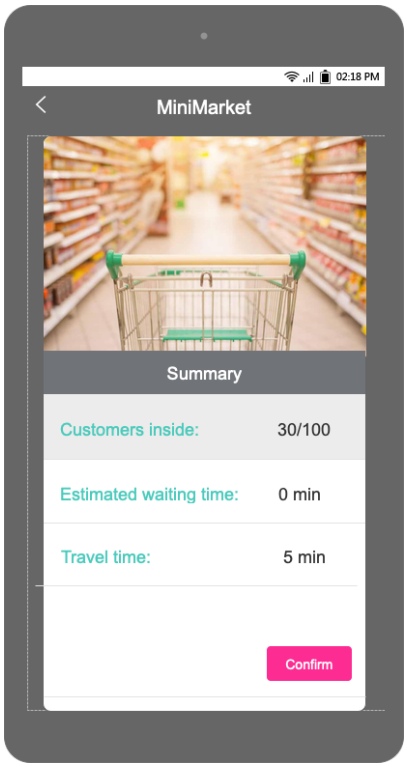


Figure 23: Summary page after the choice of a building to line up for.

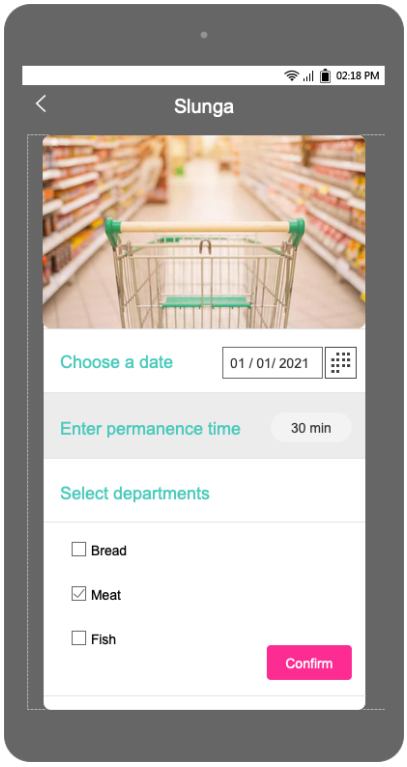


Figure 24: Options to set in order to book a time slot for the chosen building

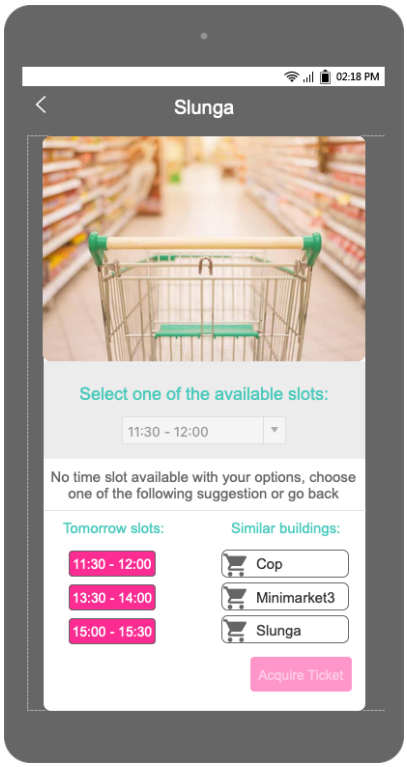


Figure 25: Interface to decide one of the available time slot/see alternative suggestions

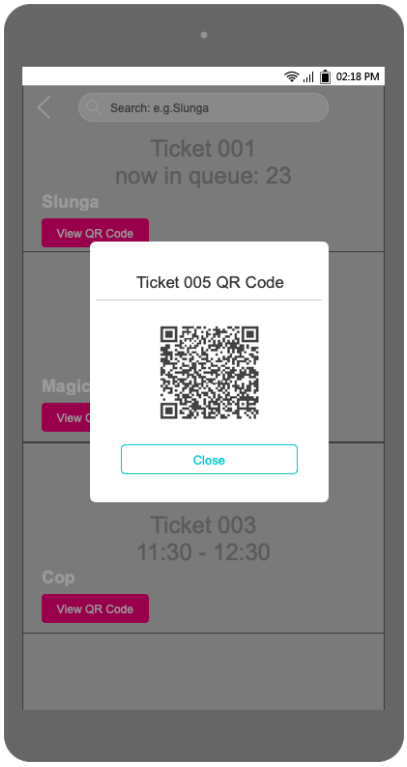


Figure 26: List of tickets of a customers, with the possibility to view the related QR code

B WebApp Interfaces

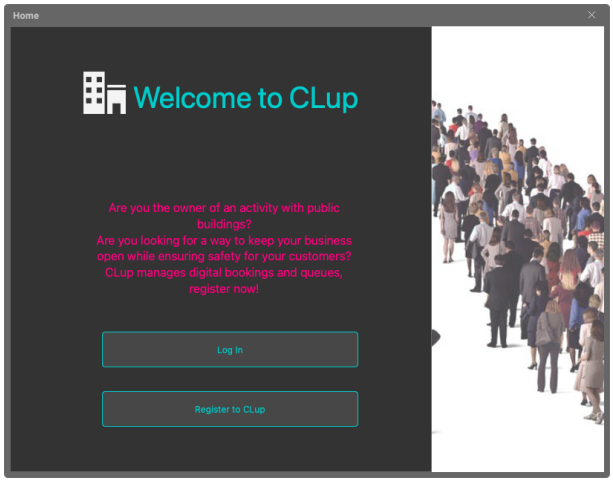


Figure 27: List of tickets of a customers, with the possibility to view the related QR code

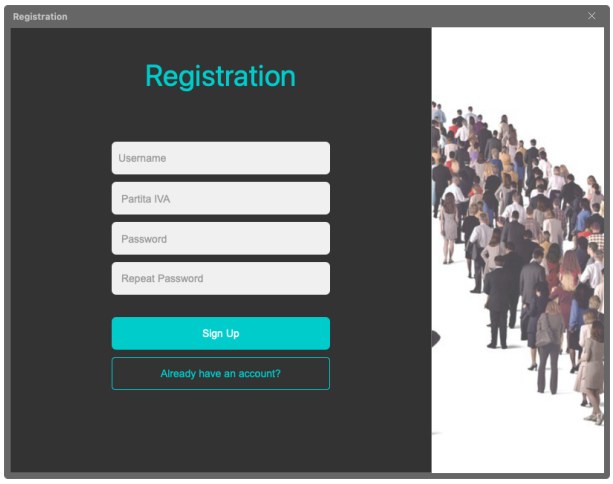


Figure 28: List of tickets of a customers, with the possibility to view the related QR code

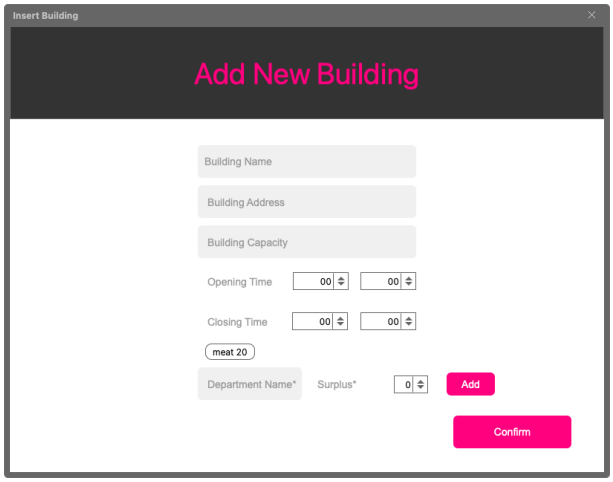


Figure 29: List of tickets of a customers, with the possibility to view the related QR code

4 Requirements Traceability

A External Interface Requirements

Here we provide a mapping of RASD document requirements with the Component provided in the architectural design section. The components are numerated as follows:

[C 1]. TicketManager

[C 2]. AuthManager

[C 3]. BuildingManager

[C 4]. GoogleMapsService

[C 5]. DataManager

[C 6]. MapsServiceMobileAdapter

[C 7]. Dispatcher

[C 8]. MapsServiceServerAdapter

[C 9]. DBMS Service

[C 10]. GPS Service

[C 11]. Mobile Application

[C 12]. Web Application

*GPS is an optional service, which can be or not exploited by the MobileApp user.

-	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10*	C11	C12
R1												X
R2		X					X					X
R3		X					X					X
R4		X			X		X		X			X
R5			X				X					X
R6			X				X					X
R7			X		X				X			
R8			X				X					X
R9			X		X				X			
R10		X			X		X		X		X	
R11		X			X				X			
R12		X					X				X	
R13											X	
R14		X					X				X	
R15		X			X				X			
R16		X			X							
R17		X					X				X	
R18			X	X			X	X			X	
R19			X	X	X			X	X		X	
R20			X				X			X	X	
R21			X				X				X	
R22			X				X				X	
R23			X		X		X		X		X	
R24			X		X		X		X		X	
R25		X	X		X		X		X		X	
R26		X	X		X		X		X		X	
R27			X	X	X		X	X	X		X	
R28				X		X				X	X	
R30			X		X		X		X		X	
R31				X		X				X	X	
R32	X	X	X	X		X				X	X	
R33				X		X					X	
R35			X		X				X			
R36	X	X					X				X	

-	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10*	C11	C12
R37	X	X					X				X	
R38	X						X				X	
R39	X						X				X	
R40	X						X				X	
R41	X		X		X				X			
R42	X						X				X	
R43	X						X				X	
R44	X						X				X	
R45			X				X				X	
R47	X						X				X	
R48			X				X				X	
R49			X		X		X		X		X	
R50	X		X		X		X		X		X	
R51	X		X				X		X		X	
R52			X		X		X		X		X	
R53			X		X		X				X	
R54	X				X		X		X		X	
R55	X				X		X		X		X	
R56	X				X		X		X		X	
R57	X	X			X		X		X		X	
R58	X											
R59	X						X				X	
R60	X	X			X		X		X		X	
R61	X	X			X		X		X		X	
R62	X	X			X		X		X		X	
R63	X				X				X			
R64	X				X		X		X		X	
R65	X				X		X		X		X	
R66	X				X		X		X		X	
R68	X	X			X		X		X		X	
R69	X	X			X		X		X		X	
R70	X	X			X		X		X		X	
R71	X				X		X		X		X	
R72		X			X		X		X		X	
R73			X		X		X		X		X	
R74	X				X		X		X		X	
R76	X				X		X		X		X	
R77	X				X		X		X		X	
R78											X	

R29, R34, R67, R75 are not present in the table since they have been duplicated/skipped in the RASD.

5 Implementation, Integration and Test Plan

A Implementation

The entire system will be implemented exploiting a **bottom-up** approach. This approach is chosen for both the server side and the client side. The server side will be the first to be implemented and tested. A bottom-up implementation is incremental and gives the possibility to test whenever possible every small software unit regardless of the others facilitating bug tracking.

The first step can be implementing the DataManager which will be communicating with the DBMSService (MySQL) thanks to JDBC. This will guarantee to others components depending on the DataManager's interfaces to access to persistent objects mapped to the relational database.

After that we will implement the AuthManager component which depends only on the DataManager.
The next step consists in implementing the TicketManager.
MapsServiceServerAdapter will be implemented and tested before the BuildingManager as the latter depends on it.

Finally, for the server side, the dispatcher will be implemented.

The order in which we will implement our components in the application server is:

1 DataManager

2 AuthManager

3 TicketManager

4 MapsServiceServerAdapter

5 BuildingManager

6 Dispatcher

In parallel we will implement the MapsServiceMobileAdapter and the MobileApplication components.

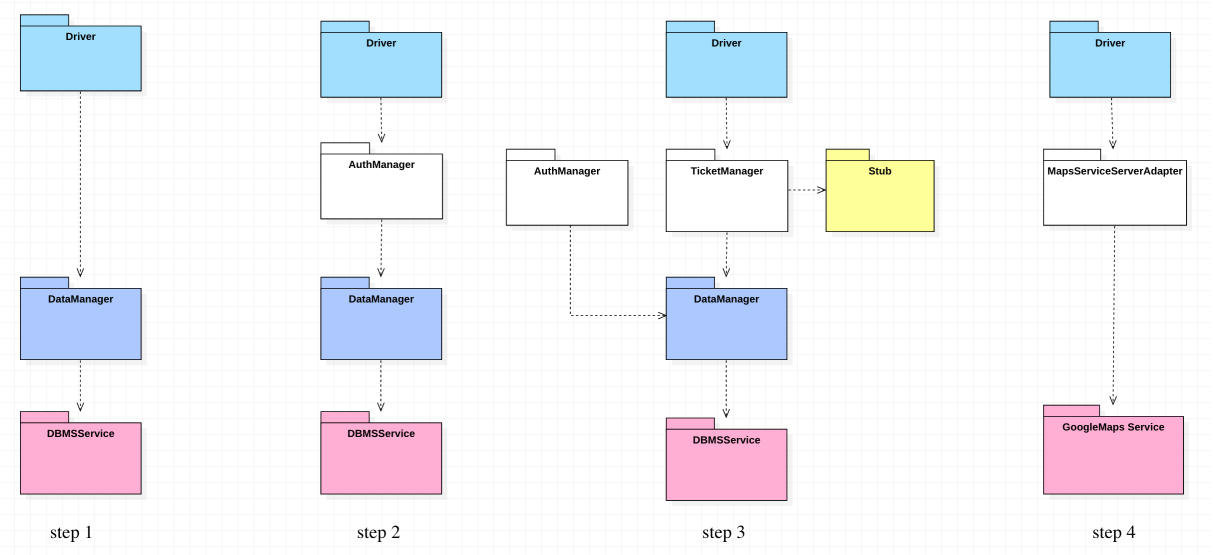
DBMSService is not implemented as it is the DBMS software and neither is GoogleMapsService as it is an external service component.

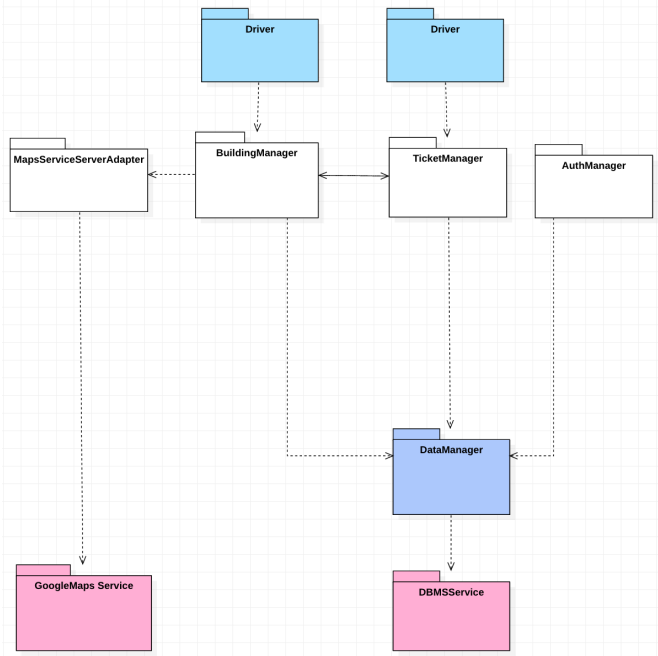
B Integration and Test Plan

As stated previously we will use a bottom-up approach.
The following diagrams illustrates in each step how we will combine each component and which utility components will be used to test them.

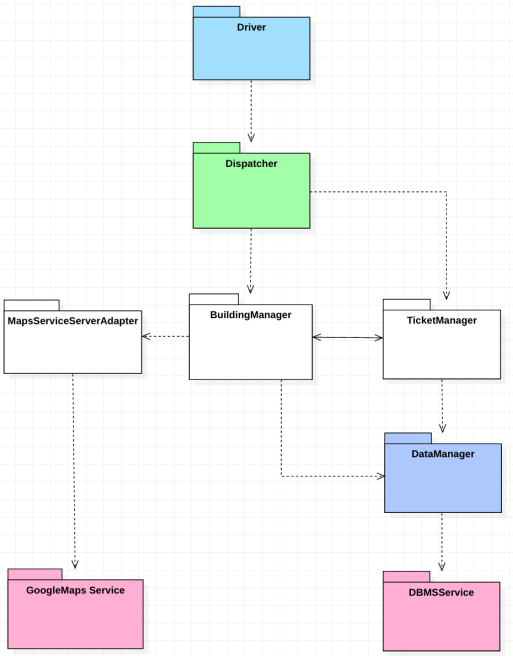
Server side
Step 1: The DataManager is tested with a Driver class that uses its interfaces.
Step 2: After that it's the turn of AuthManager as it depends only on the DataManager already tested.
Step 3: When integrating TicketManager we need a Stub Component in order to simulate the BuildingManager's WaitingTimeInt interface.
Step 4: The MapsServiceServerAdapter, on which the BuildingManager will depend, is tested separately with the GoogleMapsService and a driver Class to simulate the calls to its offered methods.
Step 5: Now that MapsServiceServerAdapter is implemented and tested we can integrate the BuildingManager and remove the stub class. This step is crucial as it handles most of the business logic's implementation. The Stub class is replaced with the BuildingManager component.
Step 6: Finally the Dispatcher can be implemented, integrated and tested with the rest of the server side system.

Client Side
Step 7: The MapsServiceMobileAdapter is implemented and tested with the GoogleMapsService and a Driver class to simulate the calls to its methods.
Step 8: Final integration Step. MobileApplication and WebApplication are integrated into the whole system.

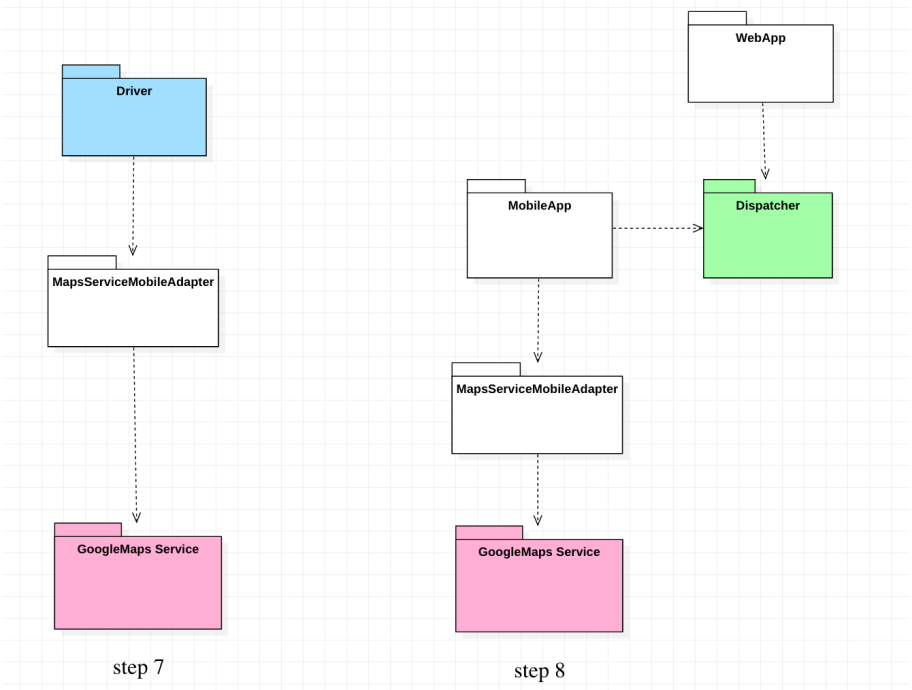




step 5



step 6



step 7

step 8

6 Effort Spent

Task	*Name*	*Time spent*
Discussion on DD	All Components	4h
Sequence Diagrams	Francesco Attorre	12h
Component Diagram	Thomas Jean Bernard Bonenfant	3h
Review of component diagram	Francesco Attorre	3h
Mockups	Veronica Cardigliano	10h
RuntimeView description	Francesco Attorre	2h
Write Introduction	Veronica Cardigliano	1h
Requirements Traceability	Veronica Cardigliano	3h
Discussion on overview and high-level architecture	All components	4h
Overview and high-level architecture	Veronica Cardigliano	1h
Deployment diagram	Thomas Jean Bernard Bonenfant	3h
Description of deployment diagram	Thomas Jean Bernard Bonenfant	1h
Architectural styles and patterns	Veronica Cardigliano	1h
Other design decisions	Thomas Jean Bernard Bonenfant	1h
Discussion on implementation, integration, testing	All Components	2h
Implementation, integration, testing	Thoma Jean Bernard Bonenfant	3h
Document review	All Components	3h

7 Used Tools

Tools used to create this RASD document:

- StarUML: for all the UML diagrams
- LaTeX: to create the pdf
- GitHub: for the repo of the project
- GoogleDoc: for a shared editing of the document