DD - Design Document

POLITECNICO
MILANO 1863

Edoardo Venir - 10570524 - Mat. 962566
Leonardo Ruzza - 10608001 - Mat. 963206

<Version 1.0>

Repository Link

Contents

1 Introduction

1.1 Purpose

The purpose of this document is to provide an overall guidance to the architecture of the CLup software product and therefore it is primarily addressed to the developers.

1.2 Scope

The product aims to function as the digital backbone of the services provided by the CLup company, which will principally include:

- the registration and login,
- the real-time ticketing for a user (including the alert to "reach the store" feature),
- the physical line up using a ticket printer in the store for a visitor,
- the booking in advance of a visit for a user,
- the management of the available stores for a system manager,
- the showing of the current ticket's number on a queue board in a store
- the QR code read from a QR code reader in a store.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

Ticket: represents the reservation of a place in a queue for an available store and always includes a number and a QR Code.

Instant ticket: Ticket associated to the Real-time ticketing, which also includes the waiting time.

Booked ticket: Ticket associated to the booking of a visit, which also includes the reserved date and time.

Physical ticket: Ticket associated to a visitor (so, the physical version of the instant ticket of users), which also includes the waiting time.

Line-up: retrieve a ticket.

Available Stores: shops which adopt CLup, and so, available in the system to CLup's user/visitor who want to reserve a ticket or book a visit to them (for simplicity almost always referred to by writing only "stores", but not to be confused to the generic ones, and it would be clear by the context).

Queue: persons who are waiting to enter in the store.

Real-time ticketing: the supply service of tickets, which allow users to line up for a current queue ("real-time") of a shop.

Book a visit: the reservation of a ticket only in advance, which is different from a real-time reservation.

Current ticket number: the number to be served now (this is the number "calling").

Waiting time: the amount of time reasonably estimated that a user/visitor will wait before his turn.

Travel time: the amount of time reasonably estimated that a user will spend to reach the store.

Permanence time: estimated time the user/visitor will spend in the store. Also, the approximate expected duration of the visit.

DefaultPermanence-time: default estimated time the visitor will spend in the store. This is a system parameter set for every store.

Maximum people capacity: the maximum amount of people that may be present at the same time in a store. This is not computed dynamically by the system but predefined by stores' owners.

CLup store's devices: It include Queue Board, Ticket Printer and QR Reader.

System: the total software and hardware infrastructure which CLup is composed of.

Check-in: passing the QR code of a ticket on the QR code reader of the store to enter.

QR code Reader: a device connected to CLup's system which can read QR codes generated with the tickets in order to check-in users/visitors.

Queue Board: a particular screen connected to CLup's system which show the current ticket's number.

Ticket Printer: a particular printer connected to CLup's system which print the first available ticket for the current queue acting as a proxy for a visitor.

1.3.2 Acronyms

AJP: Apache JServ Protocol

API: Application Programming Interface

DD: Design Document

DBMS: Data Base Management System

EJB: Enterprise Java Beans

JEE: Java Enterprise Edition

JDBC: Java DataBase Connectivity

JPA: Java Persistence API

JSP: Java Server Pages

MVC: Model View Controller

RASD: Requirements Analysis and Specifications Document

RIA: Rich Internet Application

RMI: Remote Method Invocation

UX: User Experience

1.3.3 Abbreviations

SM: System Manager

1.4 Revision history

1.5 Reference Documents

[1] "R&DD Assignment AY 2020-2021"

[2] "RASD"

[3] Slides from lectures of "Software Engineering 2" course AY 2020-2021 at Politecnico of Milan

[4] Slides from lectures of "Ingegneria del Software" course AY 2019-2020 at Politecnico of Milan

[5] "*Fondamenti di Sistemi Informativi per il settore dell'informazione*" - Cinzia Cappiello, Mariagrazia Fugini, Paul Grefen, Barbara Pernici, Pierluigi Plebani, Monica Vitali - 7 Settembre 2018

[6] jetNexus documentation

[7] Oracle19c documentation

[8] TomEE documentation

1.6 Document Structure

The DD is composed by 7 parts:

1. The first part introduces the design document. It explains the utility of the project, text conventions and the framework of the document.
2. This part illustrates the main components of the system and the relationships between them, providing information about their operating principles and deployment. This section will also focus on the main architectural styles and patterns adopted in the design of the system.
3. This part presents mockups and further details about the User Interface.
4. This part associates the decisions taken in the RASD with the ones taken in this DD.
5. This part identifies the order in which is planned to implement the subcomponents of CLup system and also the order in which is planned to integrate such subcomponents and test the integration.
6. This part is accessory and summarizes in detail the hours spent in the document production.
7. This part is the last and is composed by all the references of the tools used to redact this document and its contents.

2 Architectural Design

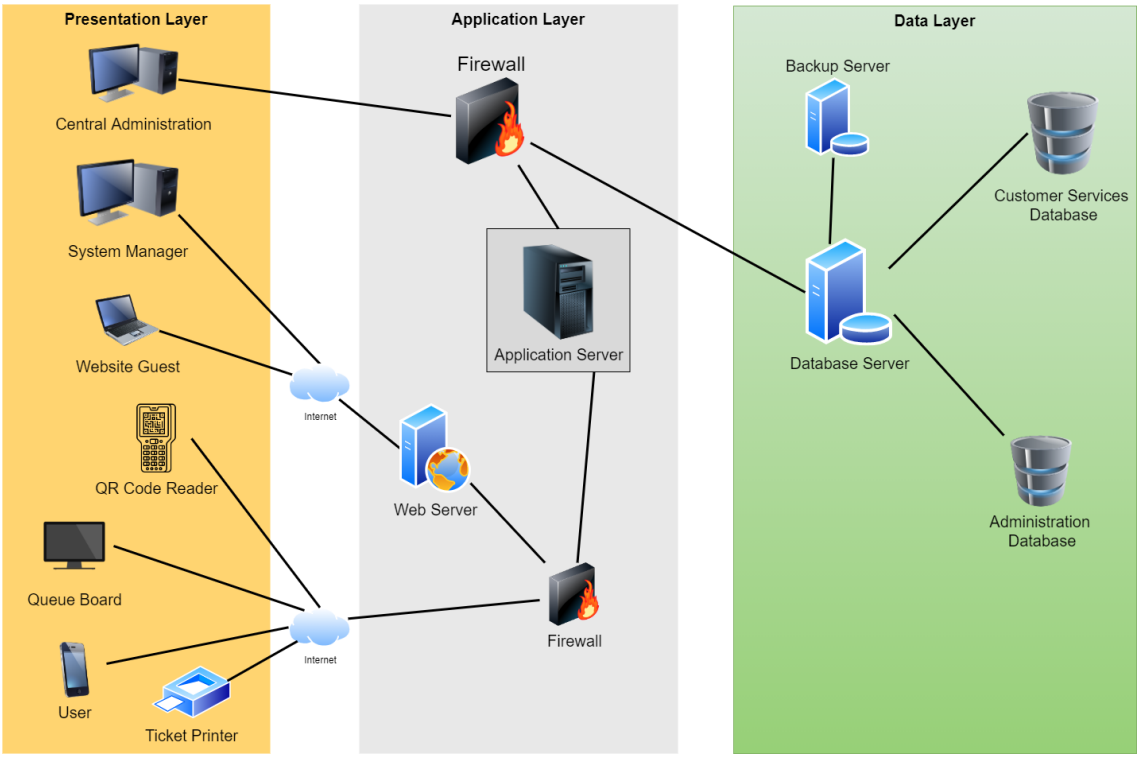
2.1 Overview: High level components and their interaction

The CLup system is based on the Client-Server pattern and it relies on a 3-tier architecture which include the 3 logic layers: Presentation, Application and Data.

The presentation layer can be divided into the client tier and the web tier. While system managers can access the system's functionalities by the official company website through the Web Server (and through the same, Website guests can navigate the official company website), users logged into the mobile application can communicate directly with the Application Server (which is located in a demilitarized zone).

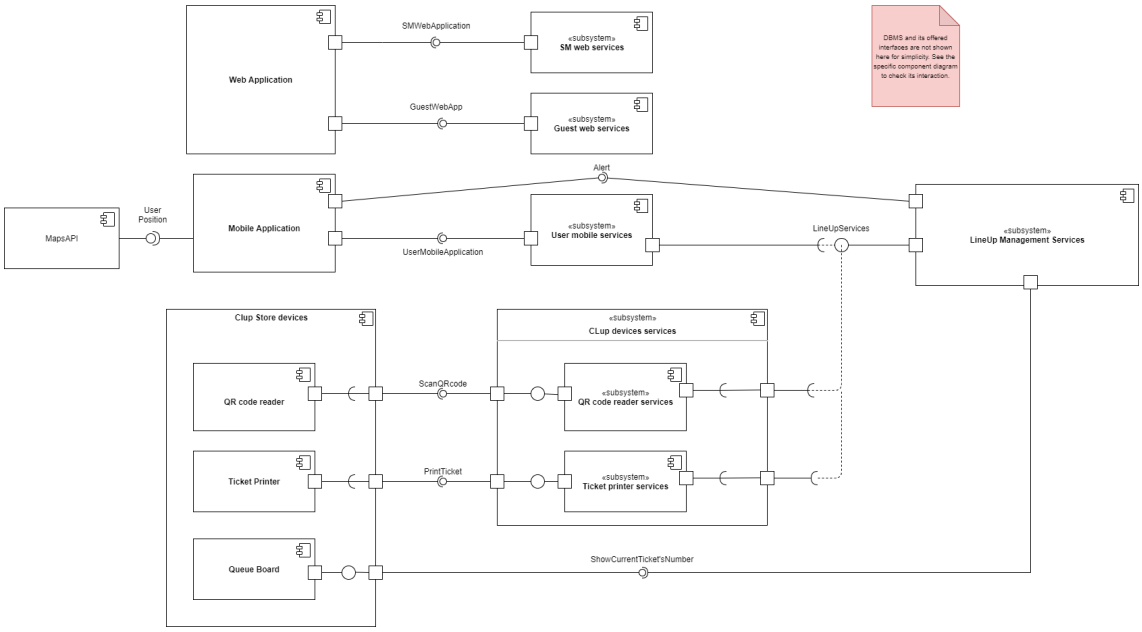
The Customers Service Database store all the information about the CLup's core features (like users' info, available stores' info, system manager's credentials, line up management's info) and not the information of the company general administration which are covered by the Administration Database. On top of that, the Database Server periodically stores critical data in the Backup Server. Both the Central Administration and the Administration Database are implemented through a third-party ERP solution; therefore, no specifics will be provided for those in the following pages.

Physical Architecture Diagram



2.2 Component view

The following diagrams show the main components of the system and the interfaces through which they interact.



- The client side is made of 4 components which refer to the SM Web Application, the Mobile Application, the Guest Web Application, the CLup store devices.
- The server side has several components: in order to get more comprehensibility, they are gathered into subsystems:
  - the SM web services component will support the stores management performed by the system managers.
  - the User mobile services component provides the interfaces to get instant/booked tickets (and see related information).
  - the Lineup management services component provides the interfaces to manage the "queues" for the various stores (for example: checking availability of ticket, calculate waiting times etc.).
  - the Ticket printer services component provides the interface to get a physical ticket for visitors.
  - the QR code reader services component provides the interface to scan a QR code of a ticket and give the consequential feedback.
  - the Guest web service component will provide access to static contents of the website such as rules, tutorials, and other information about the

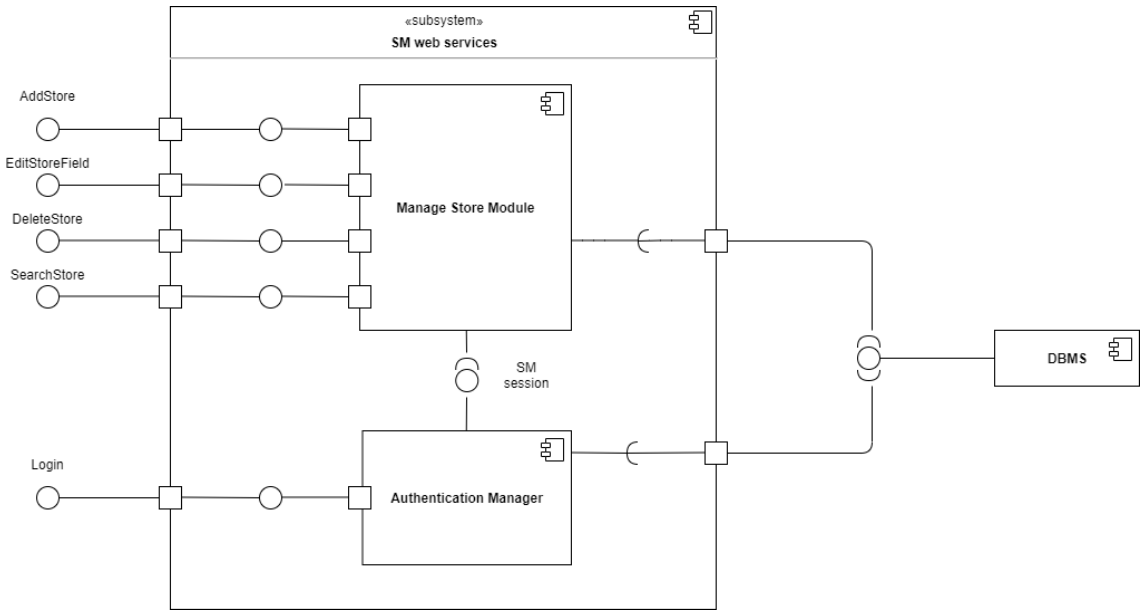
CLup service. This part of the system will not be further analysed in other sections of the document.

2.2.1 SM web services component

SM web services subsystem contains the following component:

- Manage Store Module: This allow the SM to do all the operations on the system for updating and maintenance
- Authentication Manager: This handle the registration process of a SM

These components provide to the SM web application the following interfaces: *Login, SearchStore, AddStore, DeleteStore, EditStoreField*. To fulfil their related goals these components needs to communicate with the DBMS to check the SM credentials, and to update (add, delete, edit) store's information. This subsystem doesn't need to use other subsystems' interfaces.

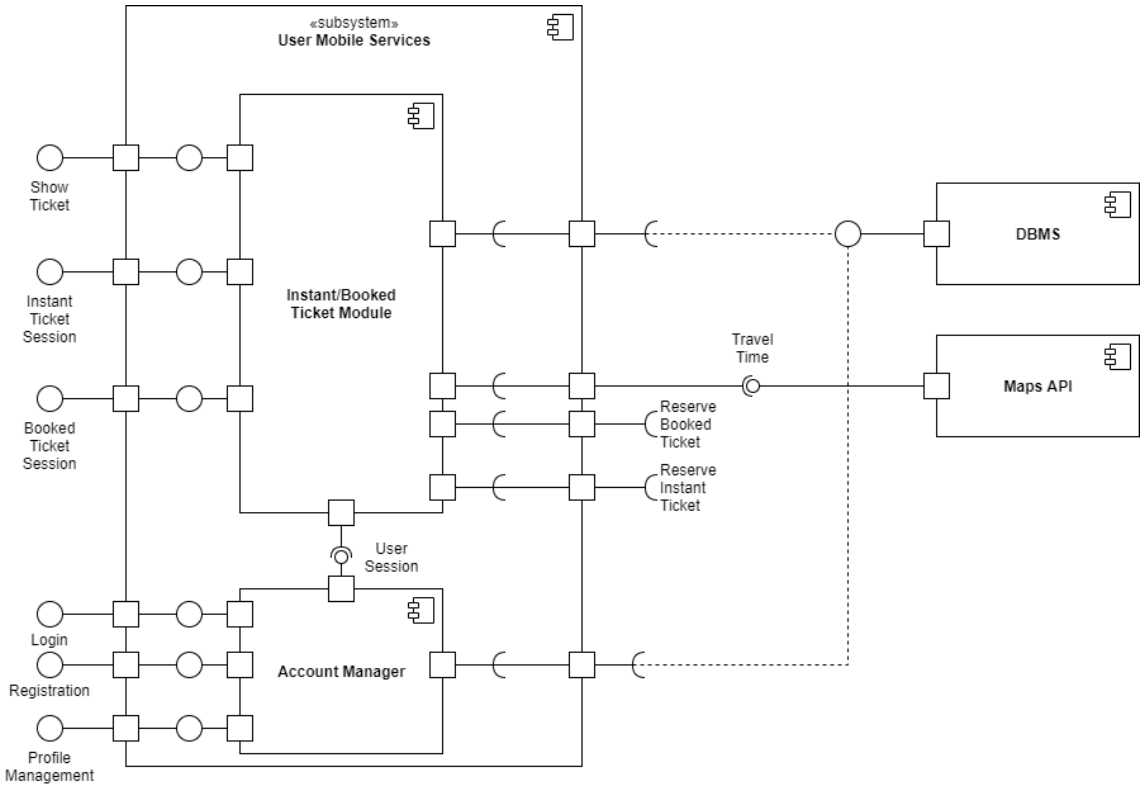


2.2.2 User mobile services projection

User mobile services subsystem contains the following component:

- Instant/booked Ticket Module: This oversees the management of the user's choices with during the reservation process. Moreover, it calculates the suggested permanence time and the travel time, it sends the alerts to reach the store to the user, and it retrieves reserved tickets information for the user.
- Account Manager: This handles the registration and login process of a user.

These components provide to the user mobile application the following interfaces: *show ticket, login, profile management, instant ticket session, booked ticket session.* In order to fulfil their related goals these components need to communicate with the DBMS and Maps API. It also needs Reserve Instant Ticket and Reserve Booked Ticket interfaces, provided by lineup management services component, to really check the availability and then to eventually manage the ticket in the "queue" of the store.



2.2.3 Lineup management services projection

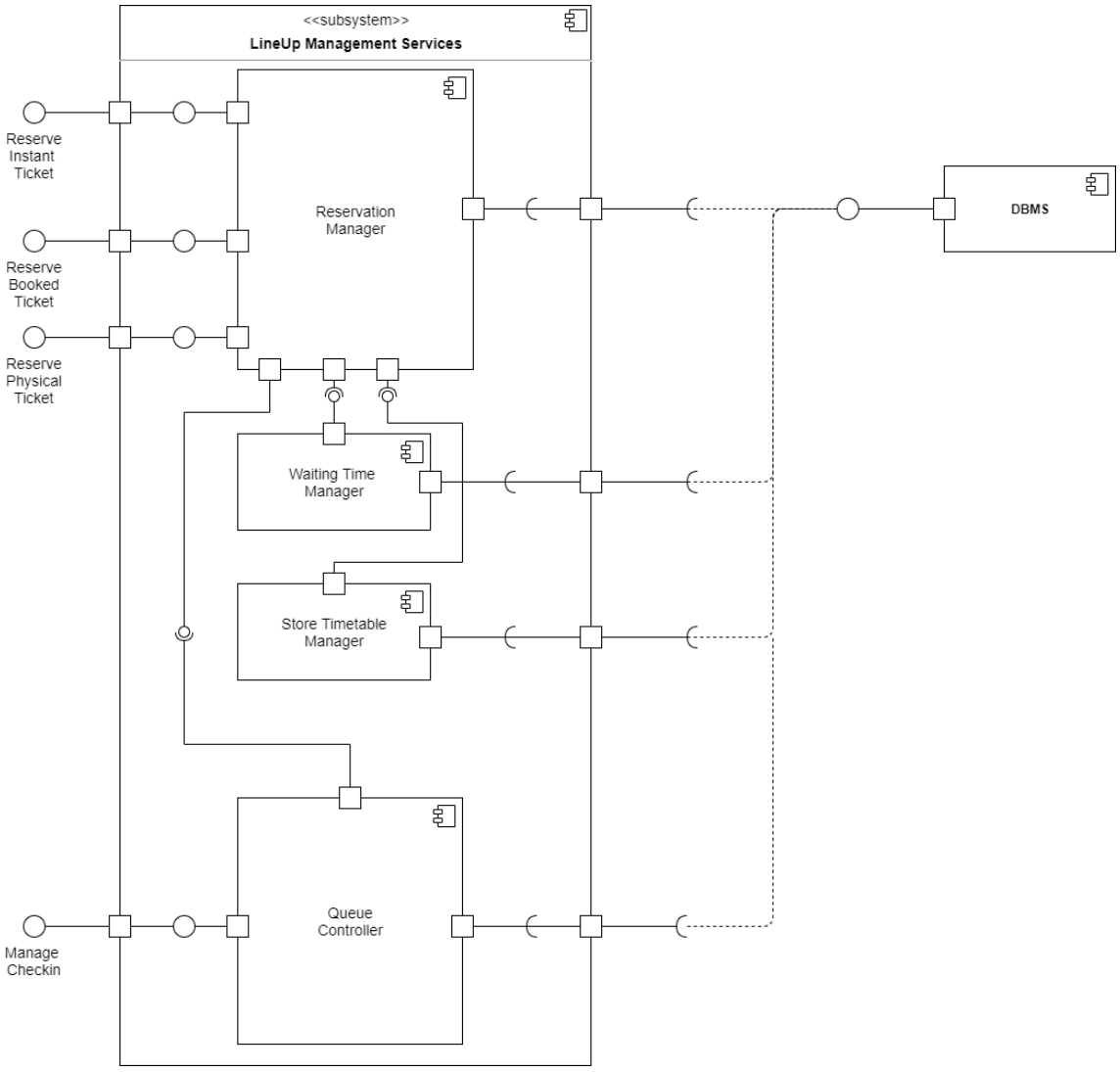
Lineup management services subsystem contains five main components:

- Reservation Manager: It oversees the effective insertion in the system of any type of ticket, during a reservation, and updates also the Queue Controller (only about instant/physical tickets).
- Queue Controller: It manages all the tickets in the daily queue and some related information such as the countdown of the waiting time. It is the main component to know which tickets are current at a certain moment and so who can be accepted or no in the store. Is important to underline that at the start of the day it takes all the booked tickets for the stores exploiting the DBMS and start to control their lifecycle, then for every new reservation notified by the Reservation Manager does the same.

- Waiting Time Manager: It compute the waiting time when a new ticket needs to be released, considering all the permanence time of the tickets in the current queue and the Maximum People capacity of the store.
- Store Timetable Manager: Check if the waiting time associated to the ticket to be released, does not exceed store's timetable. Besides, it checks that a booked ticket to be released respects all the necessary constraints.

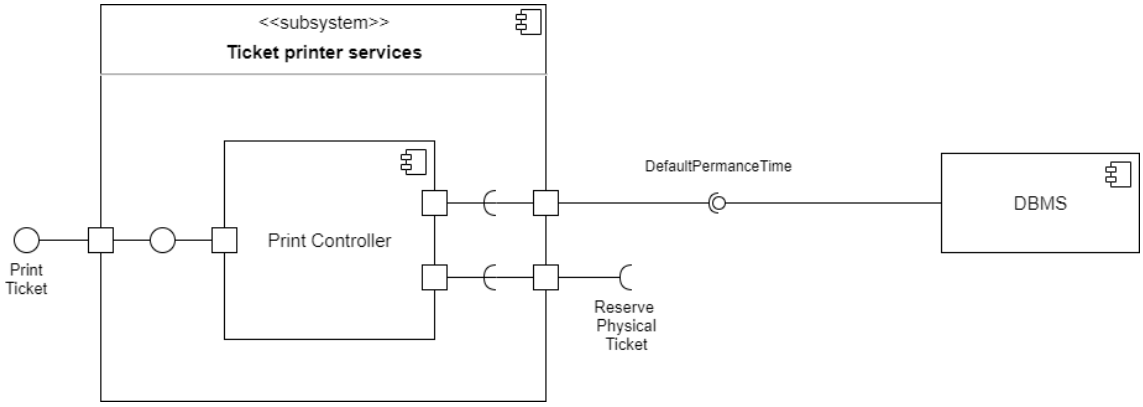
These components provide some interfaces to other subsystems.

- To the user mobile services: *reserve instant ticket and reserve booked ticket.*
- To the ticket printer services: *reserve physical ticket.*
- To the QR code reader services: *manage check in.*



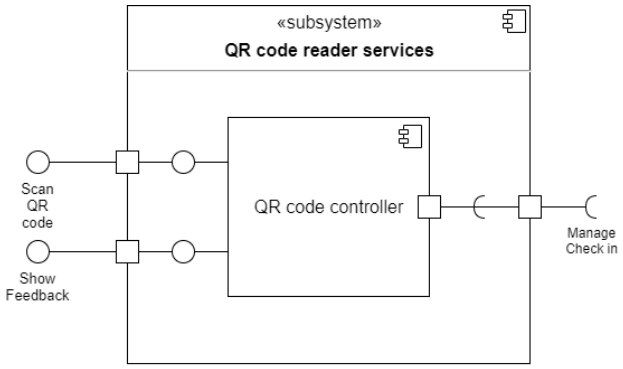
2.2.4 Ticket printer services projection

In the ticket printer services there is one main component: PRINT CONTROLLER. Trivially the print controller provides the interface print ticket to retrieve a physical ticket after a "request" from a visitor, if there is one available for the store. It also needs to communicate with the DBMS (to retrieve the default permanence time of a store), and the Reserve Physical Ticket interface, provided by line up management services component, to really check the availability and then to eventually manage the ticket in the "queue" of the store.



2.2.5 QR code reader services projection

In the qr code reader services there is one main component: QR CODE CONTROLLER. The qr code controller provides the interfaces *scan qr code* to scan a ticket's qr code of a visitor or user, and show feedback to show if the qr code's ticket is valid or not in order to enter the store in that moment. It also needs *Manage Check In* interface, provided by lineup management services component, precisely, to know if the ticket's qr code is or not among the acceptable.



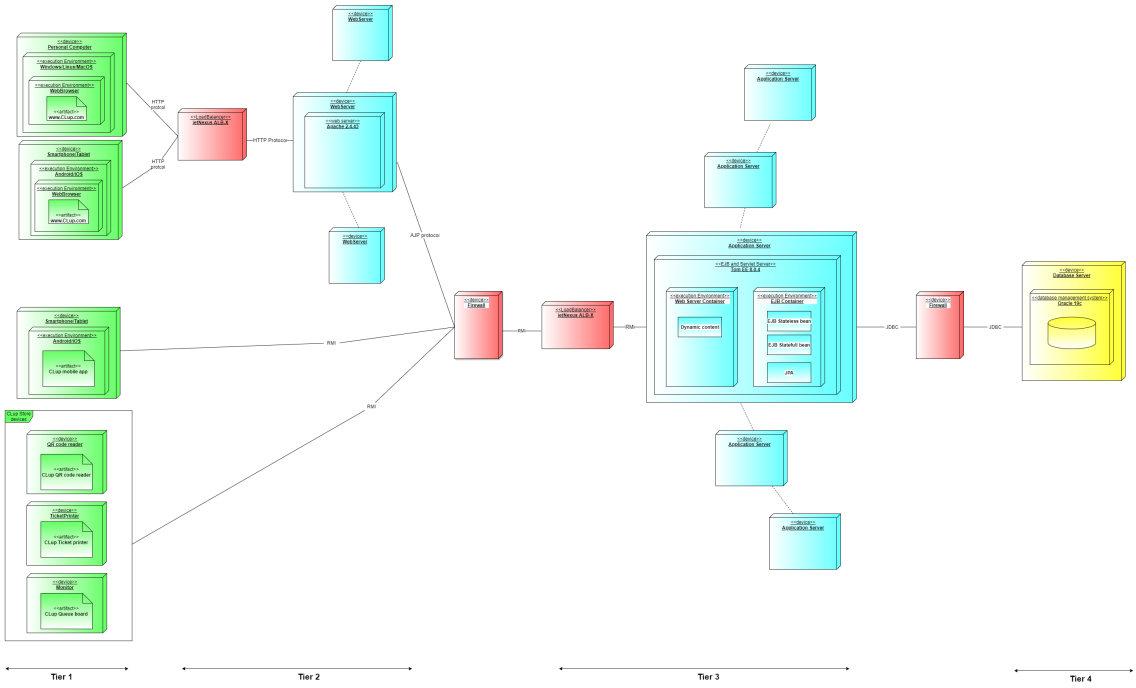
2.2.6 Queue board projection

The queue board provides the interface *Show Current Ticket's Number* to show the number of the current ticket of the queue board's store after an update, if there is one.

2.2.7 Mobile Application

The mobile application provides the interface *Alert* to notify a user to reach the store for which he/she has reserved an Instant Ticket in time.

2.3 Deployment view



The system architecture is divided into 4 tiers (notice that the Web tier and the Application tier are the exploiting of the unique Application Layer) and it is based on the JEE framework.

- Client tier (tier1 in the figure): it is composed by the mobile devices(smartphones and tablets), on which the mobile application run, and the store's devices which communicate directly to the application layers through the RMI system. In addition, this tier includes the devices (mobiles and PCs) with which CLup system can be accessed by the SM.

- Web tier (tier2 in the figure): it is composed by the web server implemented with the Apache HTTP platform which is composed of the static content module,

and mod_jk and mod_proxy_balancer, connectors which are used respectively for the connection with the application server and the load balancer.

- Application tier (tier3 in the figure): it is composed by the application server implemented with Tomcat EE which communicate with the web server through the AJP. More in detail, 2 containers are inside the application server; the Web server container, which is in charge of the dynamic content generation for the web application, and the EJB container which include the Java Beans(Stateless and Stateful) and the JPA.

- Data tier (tier4 in the figure): it is composed of the Database Server implemented with Oracle19c DBMS. This tier communicates with the application one with the JDBC connector, using the JDBC protocol.

Both in the Web Tier and Application Tier a Load balancer is used to better divide the computation load among the various nodes.

2.3.1 Recommended implementation

- Client tier: o the customer mobile app may be implemented using a cross-platform mobile development tool such as Xamarin, platforms which allows to develop native Android, iOS and Windows Phone apps in C#; also Flutter may be a good choice since it allows to create cross-platform application which are closer to native application in terms of performance. o The store devices software may be implemented using either Xamarin or any other Android development tool.

- Web tier: the web pages may be implemented using HTML 5.0 + CSS and JavaScript.

- Application tier: The EJB application server uses stateless, stateful java beans and stores the data (and the state with the client) on the database using JPA and mapping the object with the data through entity beans. TomcatEE 8.0.4 may be a good choice as runtime environment. To generate dynamic content, Servlet + JSP can be enough, despite using a template engine like Thymeleaf is strongly advised.

- Data tier: the database may be implemented with Oracle database 19c.

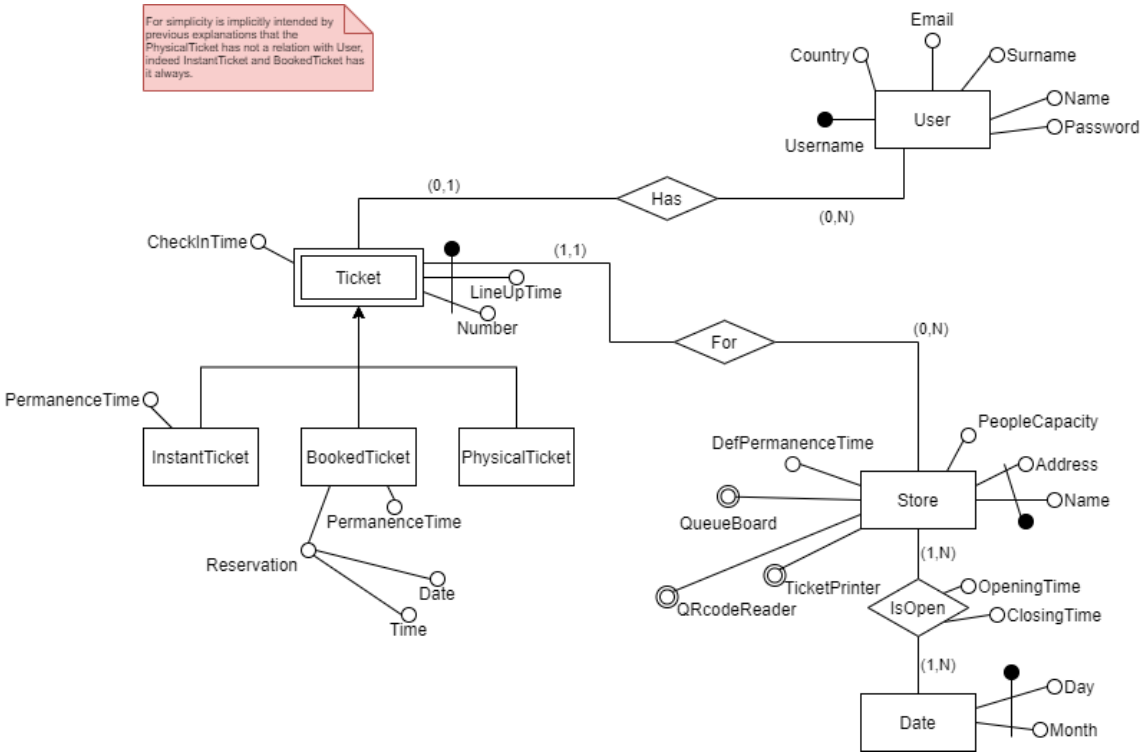
2.3.1.1 Physical Implementation

- Load Balancers: they may be implemented with the jetNexus ALB-X series by Edgenexus, which assures great reliability and performances.

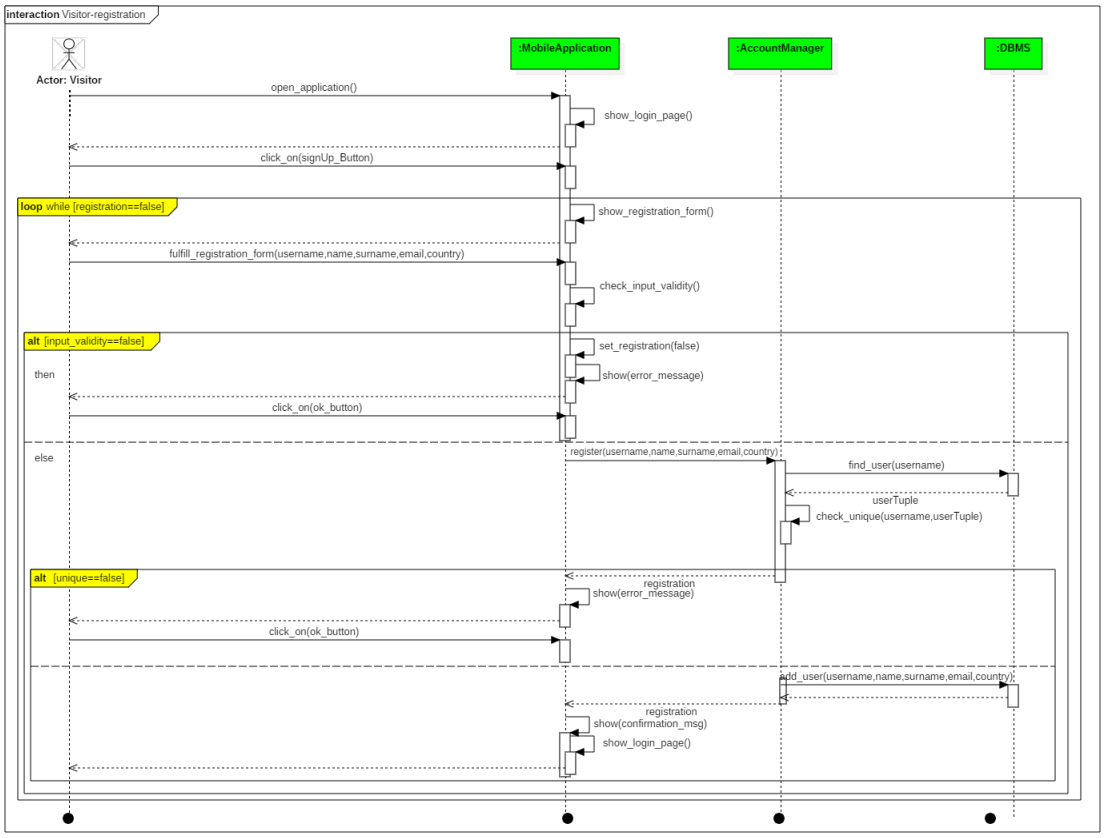
- Web, application, and data tier: a server such as Oracle SPARC T8 may be a good solution because it suits well with JEE applications and Oracle DBMS.

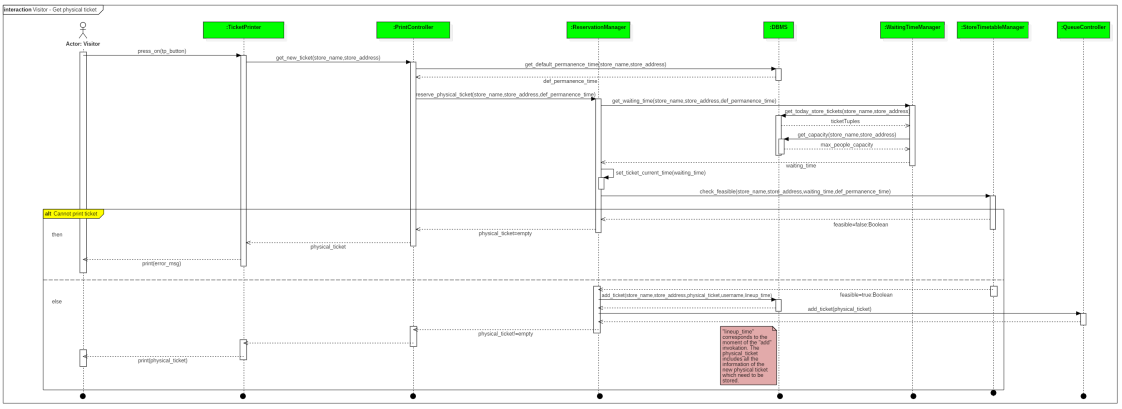
2.4 Database design (ER-diagram)

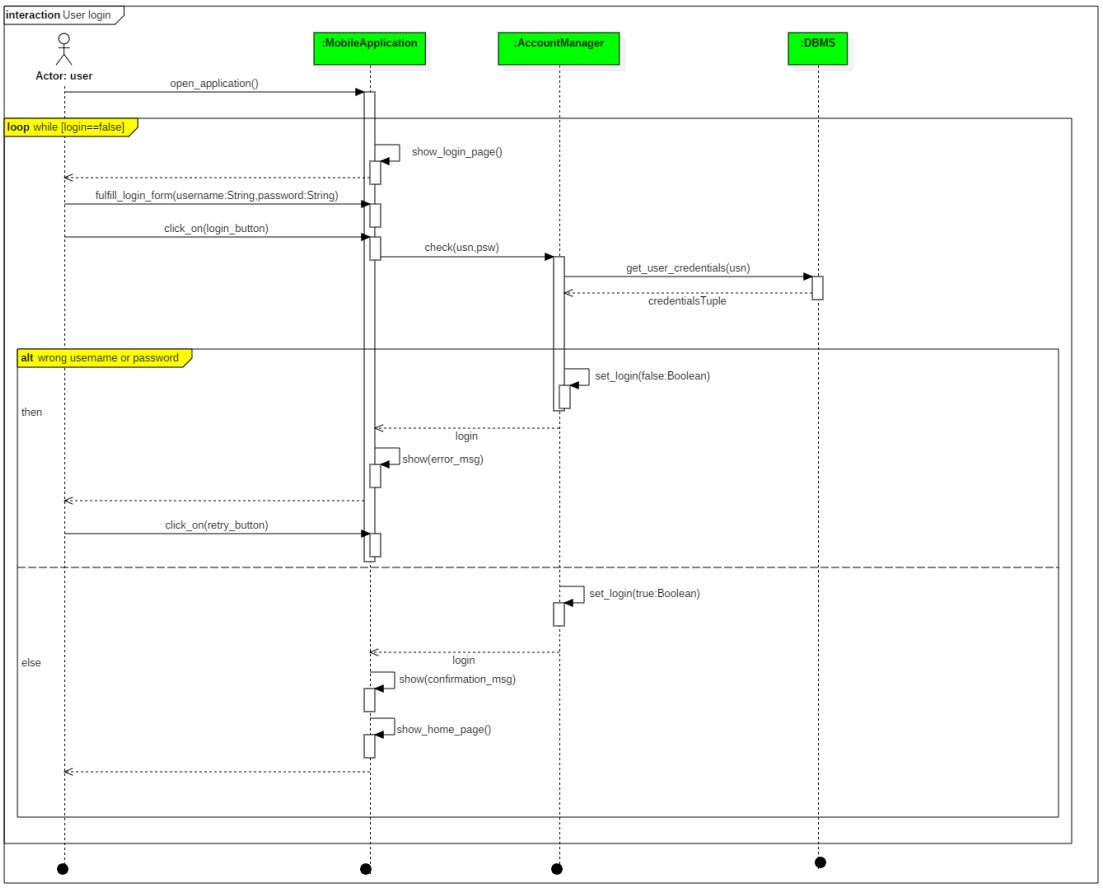
Note that the following diagram expose the design only for the Customer Services Database.

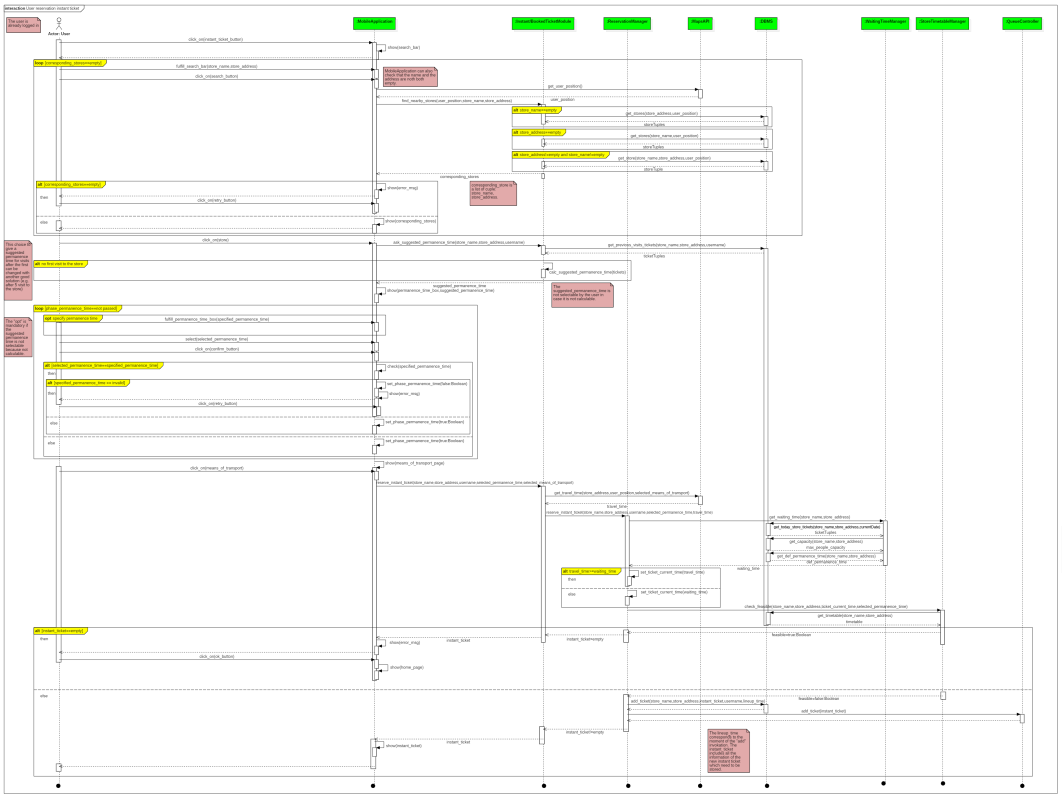


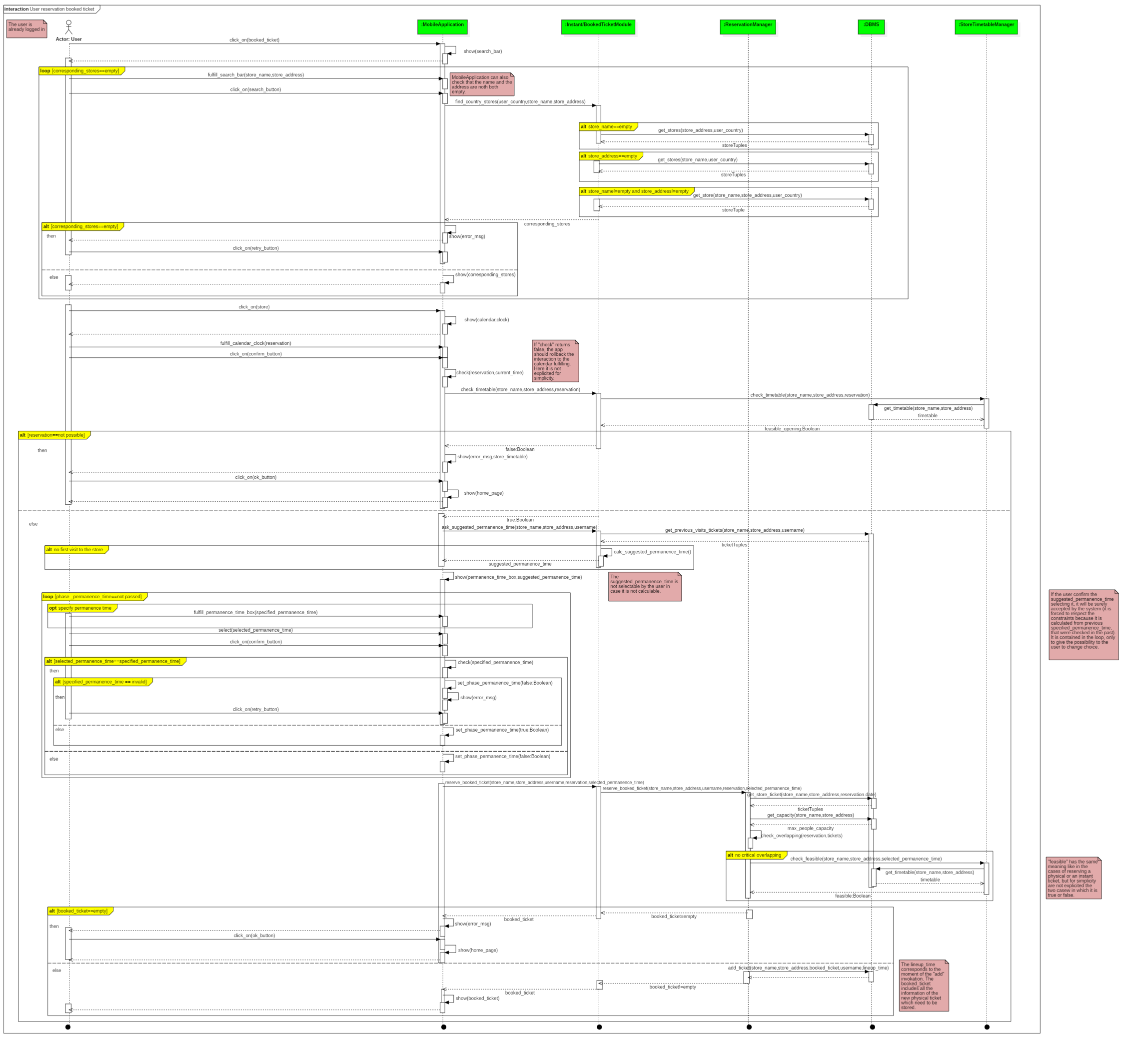
2.5 Runtime view

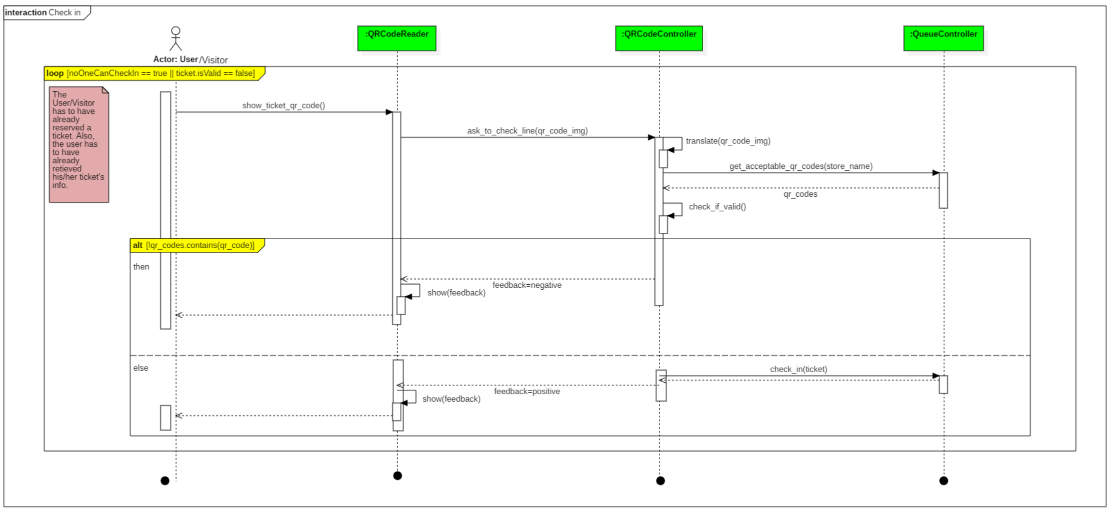


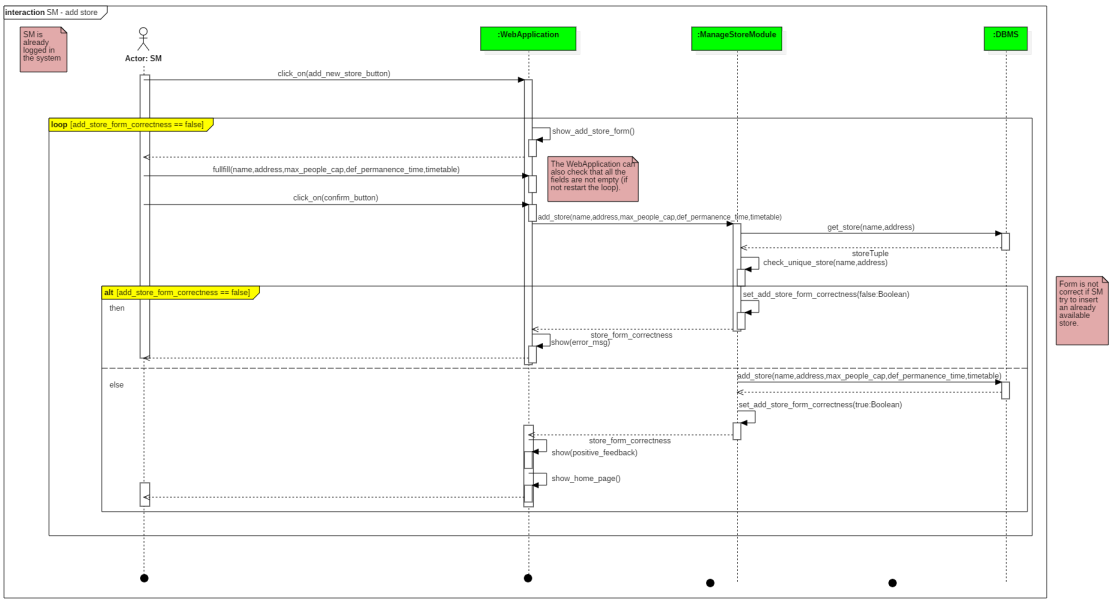


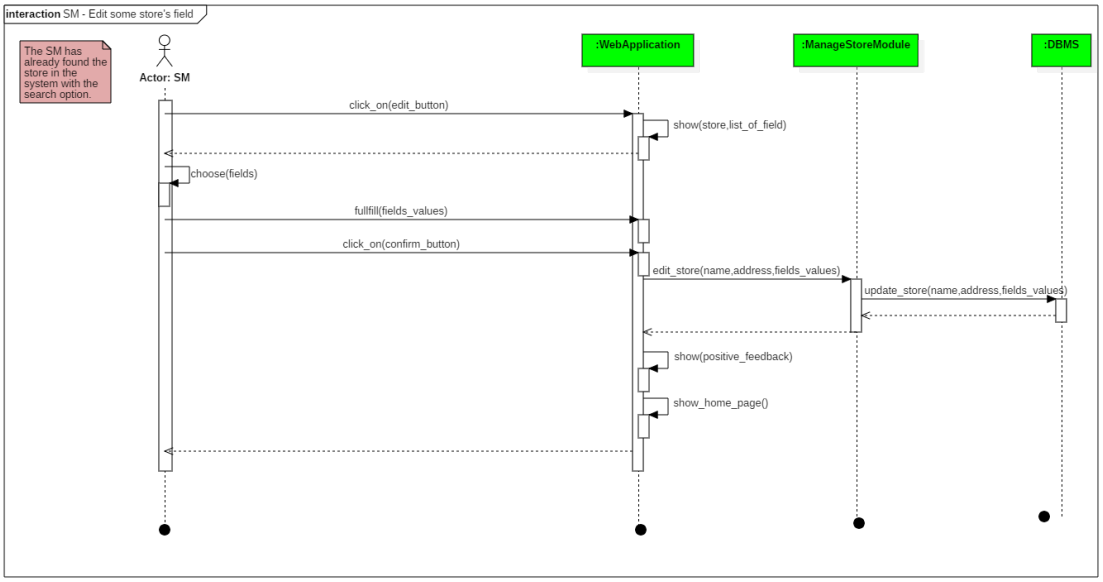

interaction User login
Actor: user
:MobileApplication
:AccountManager
:DBMS
open_application()
loop while [login==false]
show_login_page()
fulfill_login_form(username:String,password:String)
click_on(login_button)
check(usn,psw)
get_user_credentials(usn)
credentialsTuple
alt wrong username or password
set_login(false:Boolean)
then
login
show(error_msg)
click_on(retry_button)
set_login(true:Boolean)
else
login
show(confirmation_msg)
show_home_page()




interaction User reservation booked ticket
Actor: User
MobileApplication
InstantBookedTicketModule
ReservationManager
DBMS
StoreTimetableManager

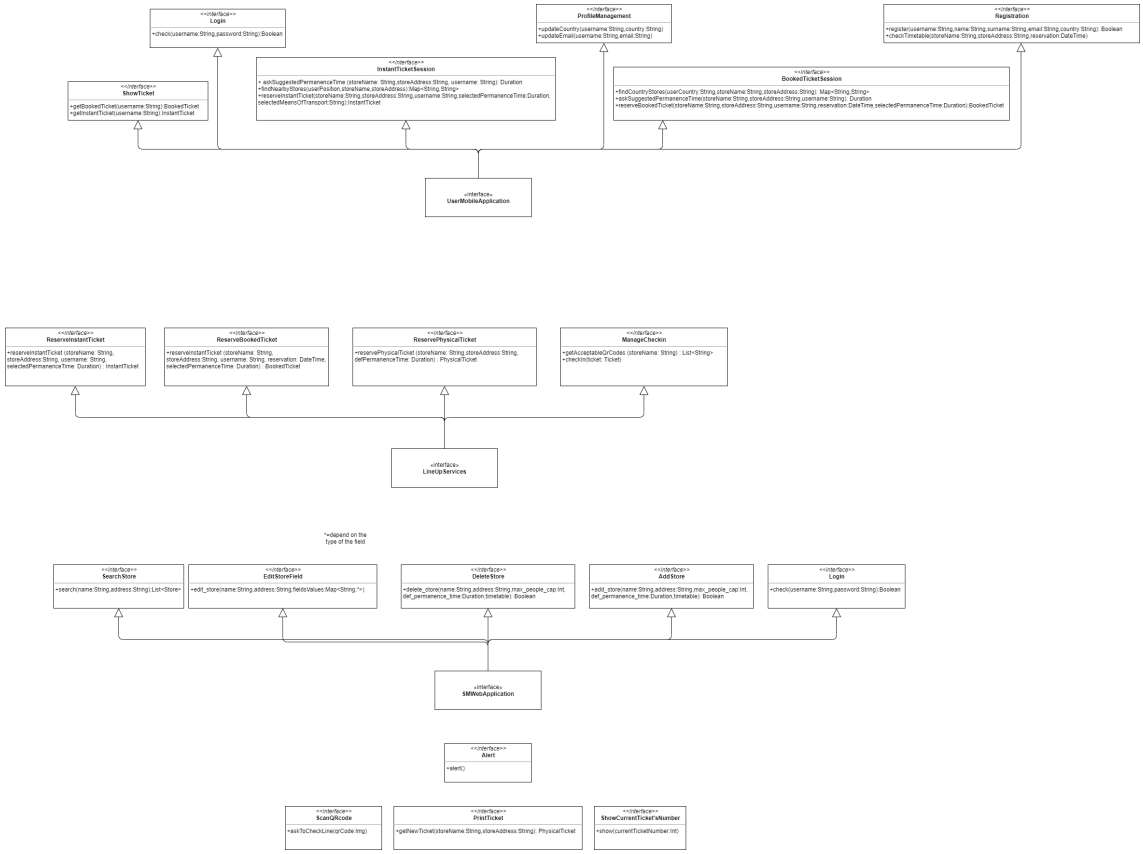

interaction Check in
Actor: User/Visitor
:QRCodeReader
:QRCodeController
:QueueController
loop [noOneCanCheckIn == true || ticket.isValid == false]
The User/Visitor has to have already reserved a ticket. Also, the user has to have already retieved his/her ticket's info.
show_ticket_qr_code()
ask_to_check_line(qr_code_img)
translate(qr_code_img)
get_acceptable_qr_codes(store_name)
qr_codes
check_if_valid()
alt [!qr_codes.contains(qr_code)]
then
feedback=negative
show(feedback)
else
check_in(ticket)
feedback=positive
show(feedback)


interaction SM - add store
SM is already logged in the system
Actor: SM
:WebApplication
:ManageStoreModule
:DBMS
click_on(add_new_store_button)
loop [add_store_form_correctness == false]
show_add_store_form()
The WebApplication can also check that all the fields are not empty (if not restart the loop).
fullfill(name,address,max_people_cap,def_permanence_time,timetable)
click_on(confirm_button)
add_store(name,address,max_people_cap,def_permanence_time,timetable)
get_store(name,address)
storeTuple
check_unique_store(name,address)
alt [add_store_form_correctness == false]
then
set_add_store_form_correctness(false:Boolean)
store_form_correctness
show(error_msg)
else
add_store(name,address,max_people_cap,def_permanence_time,timetable)
set_add_store_form_correctness(true:Boolean)
store_form_correctness
show(positive_feedback)
show_home_page()
Form is not correct if SM try to insert an already available store.


interaction SM - Edit some store's field
The SM has already found the store in the system with the search option.
Actor: SM
:WebApplication
:ManageStoreModule
:DBMS
click_on(edit_button)
show(store_list_of_field)
choose(fields)
fullfill(fields_values)
click_on(confirm_button)
edit_store(name,address,fields_values)
update_store(name,address,fields_values)
show(positive_feedback)
show_home_page()

2.6 Component interfaces

The following diagram provide a list of methods offered by the various component's interfaces.



Interfaces offered by the subsystem's component to other subsytems.

- SMWebApplication: It includes all interfaces (and their respective methods) needed by the SM web app to perform updates on the system and to login in the system. Precisely these are the interfaces provided:
  - *AddStore*
  - *EditStoreField*
  - *DeleteStore*
  - *SearchStore*
  - *Login*
- UserMobileApplication: It includes all interfaces (and their respective methods) needed by a user of the Mobile Application to login in the system, provide information to reserve Instant and Booked tickets, retrieve information about reserved tickets and update some of his/her credentials.

  - *ShowTicket*
  - *InstantTicketSession*
  - *BookedTicketSession*
  - *Login o Registration*
  - *ProfileManagement*

- LineUpServices: It includes all interfaces (and their respective methods) needed to add a reservation for all type of tickets, and to handle a check in seeing if it is possible or not.
  - *ReserveBookedTicket*
  - *ReserveInstantTicket*
  - *ReserveBookedTicket*
  - *ManageCheckIn*
- ScanQRcode: It provides methods needed by the store's QR code readers to permit check in of user/visitors by their ticket's QR codes.
- PrintTicket: it provides methods needed by the store's ticket printers to ask to emit new physical tickets when a visitor requires one.
- ShowCurrentTicket'sNumber: it provides methods needed by the system to show current tickets' number of a certain store on the related queue boards.
- Alert: it provides methods needed by the system to alert a user to reach the store associated to an instant ticket already reserved from him/herself.

Interfaces offered by the MAPS API.

- UserPosition: It provides methods needed by the Mobile Application to get the current User Position thanks to the GPS.
- TravelTime: It provide methods needed to calculate the travel time from current User Position to the chosen Store during the reservation of an Instant ticket, considering the means of transport selected by the user.

2.7 Selected architectural styles and patterns

2.7.1 Styles and pattern

- Framework: JEE has been chosen as framework in which develop the Server-side components of CLup since it allows to create compact, safe, and reliable application. Beside it allows the use of EJB for managing in a finer way the state of transactions with the clients, and JPA so to let the communication with the databases easy and more efficient.

- THIN CLIENT: it allows CLup app to run on mobile devices without requiring a heavy computational load, which is handled mainly by the server. This also prevent to release too many new versions of the app in case of changes in requirements/functionalities, since only the presentation, handled by the client, is going to be changed.
- RIA: To ensure a better experience while using the Web Application, Javascript will be used. This is also important to reduce server requests and improve the general system performance.
- STATE PATTERN: ties object circumstances to its behaviour, allowing the object to behave in different ways based upon its internal state (used in the QR code reader state chart diagram).
- FACADE PATTERN: supplies a single simplified interface to a set of other components/end adopters (see the component diagrams). This hides the complexities of the system and allow an easy maintenance and reusability of the code.
- SERVER FARM + SCALE-OUT PATTERN: both the Application Server and the Web Server are cloned on more nodes, to increase/decrease the computational power following the current demand; while keeping the cost low, following the principle of the downsizing, this also prevents System's unavailability, increasing performances and reliability.
- ELASTICITY PATTERN: thanks to the presence of Load balancers, it allows to better divide the computational load between the various nodes, so to be able to respond to different numbers of users at different moments. This also guarantee to implement in a finer way the scale-out pattern thanks to a better use of resources, avoiding waste of computational power or bottle necks.

2.7.2 Suggested Architectural styles

- SHARED DISK/CLUSTER + STATELESS COMPONENTS: various clones of the Application Server shares a common device for secondary storage. This is particularly good for the S2B since it will be quite "write-intensive". Moreover, in case of a component failure, no critical data are lost. Of course, ad-hoc backups must be implemented for the storage devices.

2.7.3 Recommended Design patterns for implementation

- MODEL-VIEW-CONTROLLER (MVC) PATTERN: divides a given software application into three interconnected parts, to separate internal representations (Model) of information from the ways that information is presented to or accepted (View) from the user. This is one of the most common and effective ways to low the level of coupling between the various parts of system.

- FACTORY PATTERN: exposes a method for creating objects, allowing subclasses to control the actual creation process to fit better different scenarios. It is particularly useful if applied in combination with the MVC pattern. • Observer pattern: lets one or more objects be notified of state changes in other objects within the system. It is essential for the application of the MVC pattern.
- VISITOR PATTERN: separate algorithms from the set of objects to which it is applied. So, it is easy to modify or improve the 2 parts separately, lowing the coupling of components in the system.

2.8 Other design decisions

2.8.1 Maps API

Considering that the app will be a cross-platform native mobile application, different APIs will be used to compute the travel time for an Instant ticket. This means that travel time feature will be implemented using Apple Maps API on iOS, Google Maps API on Android.

2.8.2 Dynamic content generation

To generate the dynamic content from the Server Side, the use of a template engine like Thymeleaf is strongly recommended. In fact, it allows Graphic Designer to work on HTML pages without caring about the dynamic content that will be injected later by the programmers; this will result in an easier system's maintenance and in general in a decoupling of system's parts.

2.8.3 Atomicity

Both in the components definition and their provided interfaces, functionalities offered to ensure requirements have been split as much as possible to ensure an easier system maintenance and a decoupling of parts, and also to make the integration and implementation process easier and faster.

3 User Interface Design

See the RASD associated to this project development.

4 Requirements traceability

While making the design choices presented in this document, the main task was to fulfil in a complete and correct way the goals specified in the RASD. The following list provides a mapping between goals and requirements defined in the RASD and system components illustrated in the DD.

[G1] Allow a person to become a registered User providing some basic personal info (like Name, Surname etc.).

- User mobile services: **R1, R2**
  - Account manager
- DBMS

[G2] Allow a store's Visitor to pick up a Physical ticket from the store itself.

- Ticket printer services: **R3, R7**
  - Print controller
- LineUp management services: **R4, R5, R6, R25**
  - Reservation manager
  - Waiting time manager
  - Store timetable manager
  - Queue controller
- DBMS

[G3] Allow a User to reserve in real-time a ticket for the current queue of a particular store:

- Mobile Application: **R12**
- User mobile services: **R8, R10, R11, R13**
  - Instant/Booked ticket module
  - Account manager
- LineUp management services: **R5, R6, R9, R25**
  - Reservation manager
  - Waiting time manager
  - Store timetable manager
  - Queue controller

- Maps API: **R11, R12**
- DBMS

[G4] Allow a User to book a visit to a store:

- User mobile services: **R8, R10, R13, R16**
  - Instant/Booked ticket module
  - Account manager
- LineUp management services: **R9, R14, R15, R17**
  - Reservation manager
  - Store timetable manager
- DBMS

[G5] Allow a User/Visitor to enter in the store within the use of QR code when it is his/her turn.

- User mobile services: **R8, R24**
  - Instant/Booked ticket module
  - Account manager
- LineUp management services: **R18, R19, R20, R21, R22**
  - Queue controller
- QR code reader services: **R23**
  - QR code controller
- Queue board: **R36**

[G6] Allow a User to search and find available stores:

- Mobile Application: **R12**
- User mobile services: **R8, R13, R16**
  - Instant/Booked ticket module
  - Account manager
- Maps API: **R12**
- DBMS

[G7] The system should provide Users/Visitors with a reasonably precise estimation of the waiting time associated to their respectively instant and physical tickets.

- User mobile services: **R24**
  - Instant/Booked ticket module
- LineUp management services: **R22, R25**

  - Reservation manager
  - Waiting time manager
- DBMS

[G8] The system should alert in time a User to reach the store, for which he took the instant ticket, taking into account the time he need to get to the store from the place he currently is.

- Mobile Application: **R12, R26**
- User mobile services: **R26, R28**
  - Instant/Booked ticket module
- LineUp management services: **R27, R28**
  - Queue controller
- Maps API: **R12, R26**
- DBMS

[G9] Allow a System Manager to do operations on the system for updating and maintenance:

- SM web services: **R29, R30, R31, R32, R33, R34, R35**
  - Manage store module
  - Authentication manager
- DBMS

[G10] Allow a User/Visitor to see the current ticket number on the store's queue board.

- LineUp management services: **R19, R20, R21, R22**
  - Queue controller
- Queue board: **R36**
- DBMS

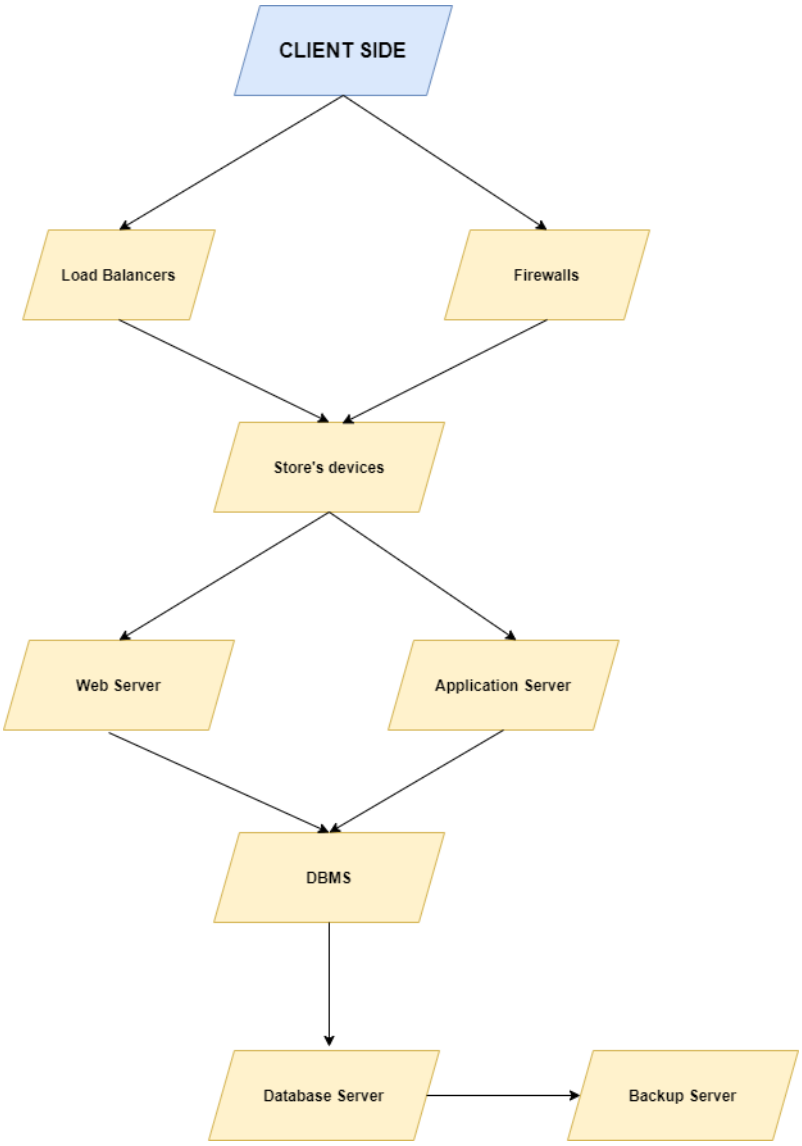
5 Implementation, Integration, Test Plan

5.1 Overview

In this last part of the DD is discussed how to take on the Implementation and Integration of the various subsystems (and their relative components), both in the client and server side, to realize the final system with its various artifacts, that will be delivered to the stakeholders.

Also, it is discussed the Verification and Validation (VaV) process should be carried on both in a static and dynamic way, to be sure do deliver a product as much as possible bug free and that matches all the requirements. However here the focus is on the dynamic VaV, where the main used approach is testing, which will be done starting from simple unit testing of various components (and their relative interfaces), finally arriving to the overall system.

5.2 Implementation plan



The implementation of the system will proceed with a bottom-up approach both for the server side and the client side, that will be implemented in parallel. Using this approach, the system could be done in an incremental way so that also the testing can proceed in parallel with the implementation.

Obviously, there are some components that rely on some others so a priority among the components is present, and it will be more clarify in this section.

5.2.1 Server-side

The first step can be implementing the DBMS, which is the component providing all methods that allow to access to the Customer Services Database and perform queries and updates (e.g. insert, delete, update) on it.

After that, it is possible to proceed to the implementation of the Waiting Time Manager and the Store Timetable Manager in parallel, which are completely independent between them. Then it will be the moment to implement the ReservationManager and right after that, the QueueController which is strictly related to the first one.

At this point, the implementation of all the components of CLup Devices Services will proceed in parallel w.r.t. all the components of User Mobile Services and w.r.t. all the components of SM Web Services. In particular, the first is developed implementing in parallel the Print Controller and the QR Code Controller. The second will proceed in the following order, the Account Manager and then the Instant/Booked Ticket Module. The third is implemented starting from the Authentication Manager and ending with the Manage Store Module.

5.2.2 Client-side

The implementation of Mobile Application, Web Application, Ticket Printer, QR Code Reader, and Queue Board, as just said, could be done in parallel w.r.t. the components described before.

There is no mention about the implementation of MapsAPI because it is an external service provided by a trusted company: thus, it will be tested only for integration test and not for the unity test.

It is important that the verification and validation phases start as soon as the development of the system begins to find errors as quickly as possible. The program testing to find bugs must proceed in parallel with the implementation: unit testing has to be performed on the individual components and, as soon as the first, even partial, versions of two components that have to be integrated are implemented, the integration is performed and tested.

5.3 Testing and integration plan

Note that the entire Testing process is based on the V-model.

A Bottom Up approach will be used for the entire Testing process, starting from the back-end functionalities offered by the server, and arriving to the modules which will communicate directly with the Client side.

Various types of testing methodologies will be used.

1. First, we start with the Unity Testing, which include white box and black box testing for the components. While the black box will be applied to each component to check that every methods offered (to other subsystem or to other component) works properly, white box testing will be applied only to the most critical components, the ones on which system's security and reliability depends most such as Queue Controller or Reservation Manager. In this part of testing main goal is to show the presence of bugs, but never to show their absence.

2. Then we proceed with Integration Testing, starting from components in the same subsystem and arriving to integration from various subsystems. It is convenient to use the black box approach here, since we are interested in interaction between components and not on how they work internally (this has been checked with the Unity Testing). A graph is provided to show the component and their dependences. The components in the lower part are the ones which must be implemented first since they offer functionalities to the upper ones (following the Bottom-Up approach). To do that, drivers will be used to emulate the presence of the upper components which have not been integrated yet, and only in few cases (such as Queue Board and Mobile Application, but only for the Alert interface interaction) it will be useful emulate the presence of lower components trough stubs.

3. Then, we arrive to Subsystem/System Testing, where there will be:
   - Functional testing: verifies if the system or part of it satisfies the respective functional requirements and goals specified in the RASD.
   - Performance testing: identifies bottlenecks affecting response time, utilization, throughput and establishes a performance baseline and possibly compares it with different versions of the same product or a different competitive product (benchmarking). Thanks to this it is possible to identify the presence of inefficient algorithms, query optimization possibilities or hardware/network issues.
   - Load testing exposes bugs such as memory leaks, mismanagement of memory, buffer overflows and identifies upper limits of components.
   - Stress testing: makes sure that the system recovers gracefully after failure.

4. Finally, we arrive to the System Testing with Clients where the final artifacts are presented to the stakeholders, to check that artifacts meet their needs.

6 Effort Spent

6.1 Table of work days

Day	People	Time	Rasd Section	Topic
14/11/20	Group	2h	1.Introduction	
15/11/20	Group	2h	2.Architectural Design	
16/11/20	Edo	1h	2.Architectural Design	Overview
	Leo	2h	2.Architectural Design	Component View
17/11/20	Edo	2.30h	2.Overall Description	Deployment view
18/11/20	Leo	1h	2.Overall Description	Component View
19/11/20	Group	1h	-	1. and 2. General Review
19/11/20	Edo	1h.30m		Build DD with LyX
20/11/20	Group	2h.30m	2.Overall Description	Runtime view
21/11/20	Edo	1h.30	2.Overall Description	Component interfaces
	Leo	2h	2.Overall Description	Runtime view
22/11/20	Edo	2h.30m	2.Overall Description-	Selected architectural styles and patterns
	Leo	1h.30m	2.Overall Description	Runtime view
23/11/20	Edo	1h.30m	2.Overall Description	Other design decision
24/11/20	Edo	3h	4.Requirements traceability	Build DD with LyX
	Leo	2h	2.Overall Description	Component interfaces
	Group	2h	-	General review

25/11/20	Edo	1.30h	5.Implementation, Integration and Test Plan	Testing and integration plan
	Leo	1.30h	5.Implementation, Integration and Test Plan	Implementation plan
	Group	1.30h	5.Implementation, Integration and Test Plan	General
27/11/20	Group	1.30h	5.Implementation, Integration and Test Plan	Geeneral review
28/11/20	Edo	4h	-	Build DD with LyX
	Leo	2h	-	General Review

6.2 Total work hour per DD section

- **1.Introduction** = 2h
- **2.Architectural design** = 20.5h
- **4. Requirements traceability** = 3h
- **5.Implementation, Integration and Test Plan** = 4.5h

6.3 Total work hour per person

- **Edo** = 18h.30m
- **Leo** = 14h.30m
- **Group** =10h

7 References

7.1 Used tools

- LyX - v2.3.5.2
- Microsoft Word
- Adobe Reader PDF
- StarUML - v3.2.2
- draw.io - Diagrams.net 13.9.5