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:	I&T
Title:	Implementation and Testing Document
Authors:	Francesco Attorre, Thomas Jean Bernard Bonenfant, Veronica Cardigliano
Version:	1.0
Date:	07-February-2021
Installation page:	https://github.com/FrancescoAttorre/softeng2-attorre-bonenfant-cardigliano
Source code:	https://github.com/FrancescoAttorre/softeng2-attorre-bonenfant-cardigliano/tree/main/clup
Copyright:	Copyright © 2021, Francesco Attorre, Thomas Jean Bernard Bonenfant, Veronica Cardigliano - All rights reserved

Contents

1 Introduction

A Introduction

The goal of this document (I&T: Implementation and Testing) is to describe the structure of the implementation of CLup, together with the main frameworks adopted for its development. It also includes informations about how the testing plan. This document is preceded by the RASD (Requirements Analysis and Specification Document) and DD (Design Document) documents. In particular, the implementation has been driven by the decisions made in the Design Document, about the main components, interfaces and architectural patterns to exploit, and also implementation, integration and testing plans.

B Scope

This document illustrates how the implementation of CLup system has been carried on. CLup is an application which aims to avoid the occurrence of crowds in a period of global pandemic. To achieve this goal, it allows customers to queue before entering a building without creating long physical queues in front of it, or to book a visit to access a building, through an application. 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 activities, which will register their buildings giving all the necessary information, by store managers who manage the entrances to the building they're related to, and by customers. Booking a visit, customers can also 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.

2 Functions Implemented

In the developed software all the main features and functions of CLup have been implemented, leaving aside additional features that could be easily added at a later time, on the base structure already existent. We present below a list of the implemented features:

- Registration of an Activity to CLup, after providing username, password and P.IVA, and log in of such Activity in order to insert its buildings.
- Buildings insertion, providing name, address, a capacity, opening times and optionally departments with surplus of capacity.
- Generation of a building access code for each building, to allow store managers to sign up for a building.
- Registration of customers after providing username and password, in order to make a booking.
- Login of customers after providing correct credentials.
- Providing clients a list of buildings among which to choose the one to queue or book for, avoiding buildings too far from the customer.
- Provide an estimated travel time depending on the means of transport chosen by the customer, using an external maps service.
- Computation of estimated waiting times in queue for each line up ticket, basing on building's statistics.
- Discovery requests to keep the estimated waiting time as updated as possible.
- Giving physical customers the possibility to acquire a line up ticket having the store manager as an intermediary.
- Queue management.
- Use of departments for each building with additional capacity in order to increase the number of people allowed if directed only to those departments.
- Acquisition of a booking digital ticket, after providing a date, a slot of time, a permanence time and optionally departments.
- Provide a list of available time slots, multiple of 15 minutes, to a customer who wish to book for a building.
- Providing a positive/negative result after a check by the store manager to decide whether a customer can enter or not.
- Making tickets become expired 10 minutes after their validation.
- Allowing store managers to register customer's exits.

Other features have been discarded since they have been considered of minor importance. They are for instance:

- Notification to leave for a building when TravelTime <= WaitingTime, this feature can be developed by the client, once having the information provided by the server and by the external maps service.

- Alternatives of time slots or buildings, which is an additional feature, not one of the main ones necessary for the application to work properly.
- Generation of QR codes with ticket's informations.
- Computation of the estimated permanence time for registered customers, basing on their previous visits to the building, additional feature.

All these features are suggested for a better effectiveness of the application, except the generation of QR code which is particularly critical for the practical use.

3 Frameworks

A Development Frameworks

The main frameworks used are JEE (Java Enterprise Edition) and JPA (Java Persistence API). In the section "Source Code Structure" is better explained the specific structure of the implementation, however the main specifications used have been Enterprise Java Beans, servlets, RESTful Web Services. The EJBs used in the specific are the stateless session beans ones, and each component have been developed as a stateless Bean. Servlets have been used to manage HTTP requests done through the webApp. RESTful Web Services have been used, instead, for the MobileApp, exploiting JAX-RS, also in this case to manage HTTP requests. As explained in section G of the DD document, we used for this implementation many patterns, such as adapter for the external MapsService, a facade pattern with the use of a dispatcher and an MVC architecture. JPA, instead, has been used to develop in a fast and easy way the needed Entities, performing ORM (Object Relational Mapping) in an efficient way. We used as implementation EclipseLink, since the middleware used is TomEE Plume 8.0.6.

A.1 Adopted programming languages

The programming languages used in the implementation have been Java and a little of Javascript. The main advantages of using in Java are Object-Oriented programming, allowing to write standard and reusable code, being platform-independent and makes it easy to write, compile and debug.

A.2 Middleware adopted

The middleware adopted is TomEE, in particular it has been used the TomeEE Plume 8.0.6 version as application web server.

A.3 Other API

Others APIs used are :

- JTA (Java Transaction API): one of the JavaEE API used to manage DataBase transactions.
- JAX-RS (Java RESTful Web Services): JavaEE API for creating web services, exploiting REST architectural pattern, as stated in the DD, section G.

A.4 Authentication Method

To authenticate the different types of users, we have used JWT (JSON Web Tokens), in order to give users digitally signed tokens providing to each of them the authorization to request the proper services. Each token contains the ID of the user, and a string to identify the type of customer related to it.

4 Source Code Structure

A Source Code Structure

The source code is a Maven project divided in packages, which represent the main parts into which the program is divided. They are:

- auth: includes the component AuthManager's class and interface
- building: includes the Building Manager class and interface, modularized adding two more classes which are QueueManager and TimeSlotManager
- data: contains the DataAccess component splitted in Building, Ticket and User Data Access in order to isolate the interactions with the database of the related Managers. In this way each DataAccess object retrieves information of a particular category. It communicates directly with the DBMS Service, exploiting also queries by JPQL. It contains entity classes mapped on auto generated database, used also by all the other components.
- externalServices: this package contains the Adapter interface, which acts as an intermediate between the system and the external maps service. In our implementation, we exploited OpenRouteService, but, thanks to the adapter interface, an eventual change of external service wouldn't represent a problem. In fact, this would lead only to a different implementation of interface's methods.
- ticket: includes Ticket Manager class and interface, which interacts with the related TicketDataAccess class
- web: includes HTTP endpoints that access Auth Manager, Building Manager, Ticket Manager interfaces. It contains also the Dispatcher component.

5 Testing

A Testing

Informations about testing can be found in the section B of the DD, "Integration and test plan". We used a bottom-up approach, we started testing DataAccess components, during the implementation they have been divided in TicketDataAccess, BuildingDataAccess and UserDataAccess in order to better modularize the methods and usage of the dataBase. After that, we proceeded implementing and testing the AuthManager, responsible for creating users and tokens to be used for a limited time, using also Mockito tools in order to simulate the UserDataAccess with which it communicates. Then, it was the turn of implementing and testing Ticket, Building and User Managers. We added also a QueueManager and TimeSlotManager, implementing their specific interfaces in order to modularize BuildingManager tasks. As shown in the Design Document, MapsService component have been developed and tested apart from the rest, verifying the correct use of our external maps service OpenRouteService. It has been integrated in a second moment with the rest of the system, in particular with BuildingManager which exploits its functionalities. Finally, the Dispatcher component. has been implemented.

The main test cases developed are:

BuildingManagetTest

- shouldValidateNextTicket() - when a customer exits it is created a new spot and next ticket in queue should be validated
- shouldRemoveFromQueueNextTicket() - starting from a full building, when a customer exits the next ticket in queue should be removed from it.
- shouldRemoveClosedHoursTimeSlots() - time slot provided to the RegisteredAppCustomer should not include time slots preceding 8.00 opening nor time slots following 21.00 closing
- shouldNotReturnTimeSlotOfFullDepartment() - a sample department has already reached its surplus in time slot 48, then after a request of 2 time slots the 48th and 47th should be not be contained in the available timeslots provided to the user
- shouldPreventCustomerWithInvalidBookingTicketToEnter() - customerEntry method is called with a BookingDigitalTicket with reserved time slot not corresponding to actual time, it should return false to avoid him entering
- bookingDigitalTicketShouldBeAvailable() - occupied time slot 48, then requesting for time slot 36(for all departments) should return true as availability
- bookingDigitalTicketShouldNotBeAvailable() - occupied time slot 48, then requesting for time slot 48 (for all departments) should return false as availability

TicketManagerTest

- checkTicketExpired() - after validating a ticket, this test checks whether, with a validation time older than the the time interval in which a ticket remains valid, the ticket became correctly expired and the waiting time associated set to zero
- correctWaitingTimeInQueue() - after acquiring 3 LineUpDigitalTicket in queue, should check whether their waiting time is computed correctly during the waiting in queue, without anyone entering/leaving the building
- waitingTimeAfterExit() - considering an empty building (with actualCapacity = capacity), creates 4 line up tickets, two of them will enter and two will be set in queue, then it ensures that when a customer exits the building the waiting times are updated correctly

- bookingsMustNotOverlap() - creating more different booking tickets for the same customer, this test ensures that if the time slots of the tickets overlap the method will return false
- sameCustomerCannotAcquire2TicketsSameBuildingSameDay()

6 Installation

A Installation Instructions

The required softwares for the installation are mySQL and TomEE Plume 8.0.6.
First thing, modify tomee.xml in TomEE/conf/tomee.xml (TomEE is the directory where TomEE is installed) as written below:

<?xml version="1.0" encoding="UTF-8"?> <tomee>
<Resource id="clupDS" type="javax.sql.DataSource">
JdbcDriver = com.mysql.cj.jdbc.Driver
JdbcUrl = jdbc:mysql://localhost:3306/clup
UserName = vostrouser
Password = password
</Resource>
<Resource id="clupDSUnmanaged" type="javax.sql.DataSource">
JdbcDriver = com.mysql.cj.jdbc.Driver
JdbcUrl = jdbc:mysql://localhost:3306/clup
UserName = vostroUser
Password = password
JtaManaged = false
</Resource>
</tomee>

Configure username, password and port, then create CLup schema with mySQL.
Before executing, tomcat-users.xml file has to be modified adding username and password.
Then, run TomCat and uncomment the following rows:
<role rolename="tomee-admin" />
<user username="tomee" password="tomee" roles="tomee-admin,manager-gui" />

After that, go to localhost:8080/manager page and load the WAR file, then click on Deploy.