POLITECNICO
MILANO 1863

ITD

CLup – Customer Line Up

Rei Barjami – 10588572

Simone Bianchera – 10611808

Niccolò Borrelli – 10615638

Source code: https://github.com/niccoloborrelli/BarjamiBiancheraBorrelli.git
(Server_jar):https://polimi365-my.sharepoint.com/:u:/g/personal/10615638_polimi_it/EaUDzMRKLC9LtNOzpxxBchABTqXCo85fh6GA8pIHy6GxJA?e=745AGK
(Client jar)https://polimi365-my.sharepoint.com/:u:/g/personal/10615638_polimi_it/EejbozHcuZpDjknJf7iqDFcBR_SQDsUL5o7wHbP_agXNXw?e=wCkJRY

1 – Introduction

1.1 – Scope

The scope of this document is to give information about which requirements and functions where actually implemented in the software, to describe which frameworks have been used motivating the good and the bad aspect of which one of them, to describe the structure of the code and to describe how the testing was actually done in the development process.

At last information will be given on how to correctly install and use the software with all the prerequisites and a small description of what every button/form does.

2 – Requirement implemented and not implemented

2.1 – Implemented Requirements:

- **R1**: The application allows customers to make a reservation only if it is possible to enter the store
- **R2**: The system generates a unique code for each reservation
- **R3**: The system frees customers from the queue if they're late more than 5 minutes
- **R5**: The customer who lines up is inserted in the queue of the store
- **R6:** If a customer has already lined up in a store and he/she hasn't left the queue yet, he/she can't line up again in it
- **R7**: The system must allow customers to specify the duration of their visit
- **R8**: The system must allow customers to specify the departments of the items they want to buy
- **R11**: The system must allow customers to cancel their reservation before they enter the store
- **R12**: The system confirms the customer the acceptance of the asked reservation
- **R13**: In case of confirmed reservation the system must insert the customer in the queue of the selected store
- **R14**: The system must reject line up requests if the stores chosen are not available
- **R15**: The system must permit only one line up a time for the same store and the same customer
- **R16**: The application limit access to the store if the number of people is equal to the maximum possible
- **R17**: The system must allow to monitor the entrances
- **R18**: The system must allow to monitor the leavings
- **R23**: The system must keep track of the average time of permanence of customers in the store
- **R25**: After the scan of the unique code the system removes the customer from the queue
- **R26:** After the scan of the unique code the system inserts the customer into the store list of active customers
- **R28**: After an active customer leaves the store (through the re-scanning of the same code) , the same customer is removed from the active customer of the store
- **R29**: When there is a free space in the active customers, the system must decide which is the next customer to enter
- **R30**: If there is a free space available in the store and no online customer can reach the store in time, the first hand out customer in the queue can enter the store
- **R31**: The system permits the customer to select which store he wants to book a visit
- **R32**: The system permits the customer to select when he/she will go to the store(only hour not day)
- **R33**: The system confirms the customer the acceptance of the asked reservation
- **R34**: In case of confirmed reservation the system must insert the customer in the queue of the selected store
- **R36**: The system must allow customers to select a preferred store
- **R37**: The system must reject bookings in which the time chosen is one not available in the store
- **R38**: The system must reject bookings if their times for the same customer intersect each other

2.2 – Not Implemented Requirements:

- **R4**: The system must show the predicted waiting time of the customer
- **R9**: The system provides the customer a list of the stores opened at that time if the preferred one is unavailable
- **R10**: The system must suggest different stores of the same or even different chain if the preferred one isn't available
- **R19**: The system must allow store managers to set the maximum number that can enter the store
- **R20**: The system allows store manager to define the zones of the store
- **R21**: The system allows store manager to define the maximum number of people for every zone in the store
- **R22**: The system must provide the store manager the number of customers for every zone at the moment
- **R24**: The system sends a notification if the new waiting time is less than 20 minutes in case of missed customers
- **R27**: The system must send a notification to the customer when it's her/his time to enter.
- **R35**: The system includes suggestions of alternative slots to customers if the selected time isn't available
- **R39**: The system must calculate the distance between the customer and the store
- **R40**: The system must acquire the customer position
- **R41**: The system must send a notification if the waiting time of the queue is less than 5 minutes plus the arrival time to the store from their position

The implemented requirements were focused on the main logic of the software without including more ornamental parts like notifications and suggestions that weren't implemented mainly due to the fact that for us they wasn't the core of the project.

The implementation gives an extendible structure to implement additional requirements in order to satisfy client requirements. In particular, the development framework guarantees an easy way to extend the possibility of the request.

We wanted to give a prototype that gives the possibility to try a real case, we respected the implementation almost all the requirements declared in the MVP in the RASD except R27, R19, R22 and R9.

3 – Adopted Development Frameworks

- **Java:**

  We decided to implement our code using Java because of its modularity and portability which are, like we all know, its best characteristics.

  The utilized version is Java 11.

- **Spring Framework:**

  We decided to utilize the framework SpringBoot for many reasons:

  - It allows a easier implementation of the RestController which fits with our architecture/design planning based on the MVC pattern.
  - It simplifies the management of the communications between client and server.
  - It guarantees the extendibility of the Controller in terms of new requests.
  - It manages the connection with a database through Spring Data Jpa, allowing insertions, deletions, updates and query through a higher level of abstraction (the concept of repository).
  - It permits the possibility to integrate the test with a custom database created and simulated by a provider chosen (in our case H2) and, also, it's integrated with Junit.
  - The implementation of the Persistence is similar to the JavaEE.
  - The creations of the artifacts are managed by its builder.

  The utilized version is the 2.4.2.

- **Hibernate:**

  We chose it because it's integrated with Spring Data Jpa.

- **JavaFX:**

  For the implementation of the Graphical interface in the View part of the MVC we decided to utilize the JavaFX framework instead of Swing because we already had experience with JavaFX and because its graphical appearance is generally better (than Swing).

4 – Source code structure

The code is, obviously, divided into two main section:

- Server side: it contains all the business logics, the connection to the database and the rest interface.
- Client side: it includes the graphic interface in JavaFx and the part of communication.

Server side

The server side includes different section, each of these represents a set of functions:

- Request Manager: it allows to create a request with the data provided by the user, it will ask the ReservationManager section if it could be accepted or not and then add it to the queue. It's done with a Spring Component with Scope Prototype because it maps the same idea of the Stateful Component in EJB container. So every customer will have a different Request Manager in order to memorize the last request rejected done, so the component can suggest a different store (in a further version). In a following update, it will include the Suggestion Manager.
- ReservationManager: it implements all the logics needed to analyze the situation in a store (customers inside, customers booked or in the queue) in order to manage all the incoming reservations. It will decide if to accept a reservation, and, in case, at which time it can enter. In particular, addRequest and verifyAvailability permit the first function. The method "setNewFirstTime" will decide who will be the next customer through the use of a Timer and of the TimeTask.
  ReservationManager will be a single istance related always with ReservationManagerComponent (that is a singleton, according to the definition of the @Component in SpringBoot) and it will contain always two HashMap containg Timers associated with the stores. Therefore, every store will have two different timers: the timerEntrance which will check when the first customer must be enter, the timerDelay which will manage all the delays of the customer. After 5 minutes, the reservation can't be scanned through the ScanService and the turns will be recalculated.
- ScanService: it can be accessed only by the ScanClerk and provides the functionality of the scan. It will recognize automatically if it's an entrance or a leaving analyzing the list through ReservationManager. It will also update the profile of the store and the customer through Account Manager.
- AccountManager: it's responsible to update the profile of the customer and the parameters of the store, necessary to approximate the duration of the visits.

The access to the database is done with the concept of the repository, indeed in the source code a lot of interfaces will be delegated to this function. The accesses to the service will be always done through interfaces autowired when necessary.

The controller used in the application is a SpringBoot RestController, its method are divided in two types, the ones mapping the HTTPRequest (GET, POST, INSERT, DELETE) from the client and the ones checking that the format of the dates inserted in the requests are valid. Within the controller a redirector is injected by SpringBoot, it is very important because is the next step of the path from the client to the core of the server.

Client side

The connection between client side and server side is managed through HTTPRequest and HTTPResponse.

The SpringBoot WebClient is used to manage connection client side. Before a request is performed the WebClient set what type the request it will be (GET, POST, INSERT, DELETE), the URI of the request and

optionally a header containing useful info for the controller. Depending on the action that has generated the request, the response from the server will be either a string or a list of info to put on screen.

Disclaimer

The source code isn't organized in packages due to difficulty with the generation of the artifacts in Iintellij, the original project would like to create every module/component in different packages.

5 – Testing

Our focussing was on testing the components on the server side, mainly the two Component which had most of the logic of the application, such as Reservation Manager and Request manager, so our testing was focussed in this 2 classes, we also tested the components AccountManager which was lighter in terms of logic, we done the testing for this 3 components because with that we could cover all the main characteristics and functionalities of the application.

The integration is done following the DD section.

We, also, cover different cases in the system testing:

- Registration of almost all type of user (not the hand-out one): all the data are saved in the database, they can also change the preferred store.
- Registration of a new store: only the manager can insert a new store, specifying max capability, chain and address. Some considerations about that:
    - The images in the prototype are only six, so the jar can explode if the chain of store registered isn't some of this: Esselunga, Conad, Coop, Carrefour, Lidl.
    - The value put in the form aren't checked, it's supposed for a prototype that the user won't put "0" in "chain".
    - The selection of a preferred_store is essential to provide the correct functionality (especially for a manager or a scan clerk), without it the two written before can't use their functions.
- Login for the users.
- Creation of a request, both lineUp and booking request: it's possible to see the entry time calculated in the database. It's expressed as millisecond to avoid problems with timezone, it can be converted with a simple equation.
- Remove of a request: it's possible to remove a request if it exists from the profile.
- Entrance scan: it was tested these possible case:
    - The customer can enter (the time of entrance is between the entry time calculated and 5 minutes later)
    - The customer can't enter
- Leaving scan: the customer exits after an entrance.
- Selection of the preferred store for all the users.
- View of the crowd in the store and in the queue by the manager.
- Update of the data of the account (both the store data and the customer data).
- Remove of a request after 5 minutes of delay.

6 – Installation Instructions

First of all you have to install java version 11(or more) then download the 2 jar, one for the server and one for the client in the given link in the first page of this document, remember to run the server jar only one time (to run it just write in the console : java -jar <directory location of jar>\server,jar for the server, java -jar <directory location of jar>\client,jar for the client, please write it without capslock or shift because it is case sensitive, there are no args to be given)

You could also run client by just clicking on the icon of the jar.

To correctly execute the program you also have to install MySQL with you also have to configure MySQL as a windows service.

You have also to extract from the jar the file called application.properties and set spring.datasource.username and spring+.datasource.password according to the root username and root password of your MySQL otherwise you can name the root of MySQL with : root and set his password : root.

Our application has been tested only on a windows environment, try to execute it in a windows PC.

In the same repository of the source code, in the folder “Implementation” it can be found the dump.sql in order to have a database with the correct schemas.