Dario LORENZONI

Raffele NOCERINO

Marco PETTORALI

Riccardo XEFRAJ

PC BOOKING

Task 1

Index

[Description of the system 3](#_Toc24110262)

[System requirements 3](#_Toc24110263)

[Functional requirements: 3](#_Toc24110264)

[Non-functional requirements: 4](#_Toc24110265)

[Main actors 5](#_Toc24110266)

[Use Cases 5](#_Toc24110267)

[Class Diagram 6](#_Toc24110268)

[ER diagram 7](#_Toc24110269)

[Entities: 7](#_Toc24110270)

[JPA entities and annotations 8](#_Toc24110271)

[Entity PC: 8](#_Toc24110272)

[Entity Room: 9](#_Toc24110273)

[Entity Reservation: 10](#_Toc24110274)

[LevelDB Implementation 12](#_Toc24110275)

[LevelDB Database 12](#_Toc24110276)

[LevelDB content format 12](#_Toc24110277)

[LevelDB collision inside the UserDB file 12](#_Toc24110278)

[UML Diagram 13](#_Toc24110279)

[Classes 14](#_Toc24110280)

[User manual 16](#_Toc24110281)

[Login 16](#_Toc24110282)

[Sign-up 16](#_Toc24110283)

[Working with the application 17](#_Toc24110284)

[Removing a reservation 19](#_Toc24110285)

## Description of the system

The main purpose of this project is to provide to the students of a university a booking system for using computers in informatics rooms of the campus in the spare time. After the registration to the service, students can monitor the available computers. If a user decides to book a computer, the selected machine will provide a login screen in which only that student can put his credentials; otherwise, the system will reject the session. A computer can only be reserved for an hour at a time; that is in order to prevent users to reserve a PC for too many consecutive hours, not letting other users to use the service. The system keeps track of the total number of reservations made for each PC, in order to analyze which computer are used the most and most likely to be worn out.

## System requirements

### Functional requirements:

* Users can sign up to the system by putting their personal information in a sign-up service, obtaining the username that will allow them to use the application.
* A user can sign in using a login service, in order to book computers using his name.
* A user can see in the main page of the application all its reservations, except for the expired ones.
* A user, from the reservations list, can delete one or more future reservations just selecting them. Moreover when a user deletes a reservation the system automatically decrements the statistics counter “NumberOfReservation” of the PC.
* A user can insert his desired date and hour in order to see all the available PCs for each room in that time. If no PCs are available in the specified date and hour, the system displays a message.
* After the insertion of date and hour, a user can see all the available rooms. Choosing a room from the displayed list, a user can reserve a PC in that room. If something occurs during the reservation, the system shows an error message. After the reservation the system automatically increments the statistics counter “NumberOfReservation” of the PC.
* After the reservation, a user can see the map of the room and in particular, it can see the position of the booked PC in the room, which is indicated by the yellow color.

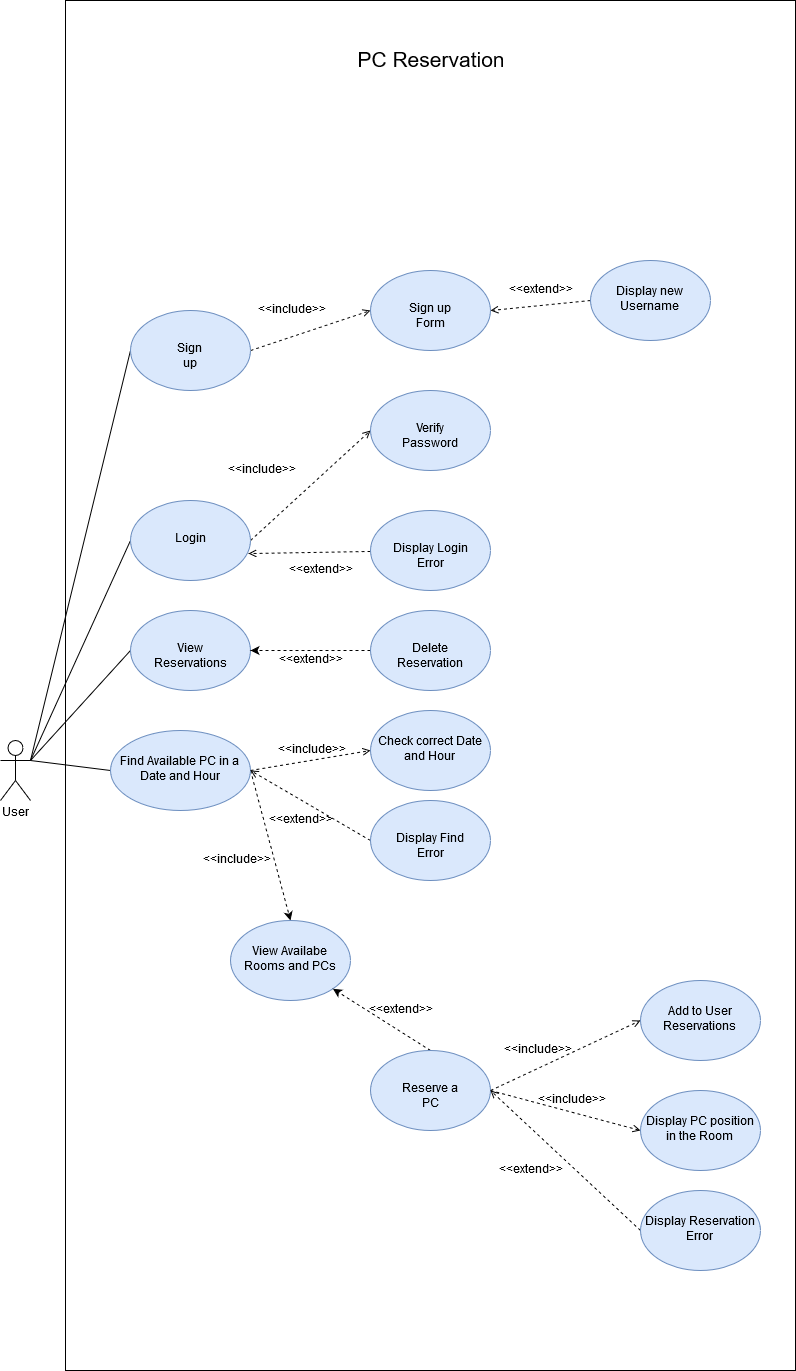
### Non-functional requirements:

* Quality: every time an error occurs the system shows an error message which explain the problem's reason.
* Memory: the database used by the application is developed in order to get a low memory usage.

## Main actors

* User (a student)

## Use Cases

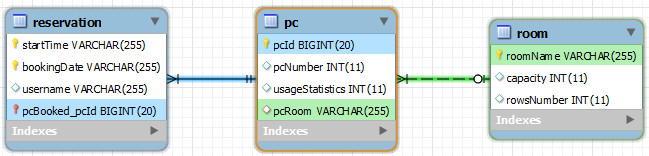


## Class Diagram

Immagine che contiene screenshot

Descrizione generata automaticamente

## ER diagram



### Entities:

* PC: it collects the information about the name and the location of the computer in the college;
* Reservation: it gathers all the reservations made by the students, by storing the users’ id, the selected machine’s id, and the date and time of the booking;
* Room: it indicates name, shape and capacity of each informatics room in the campus.

## JPA entities and annotations

The implementation of our application includes three JPA entity. In this section we are going to explain the implementation of this fundamental part, in particular the in-line code annotation used.

### Entity PC:

1. @Entity
2. public class PC implements Serializable {
4. @Id **[1]**
5. @GeneratedValue(strategy = GenerationType.AUTO) **[2]**
6. @Column(name="pcId") **[3]**
7. private long pcId;
9. @Column(name="pcNumber")**[4]**
10. private int pcNumber;
12. @ManyToOne **[5]**
13. @JoinColumn(name = "pcRoom") **[6]**
14. private Room pcRoom;
16. @OneToMany(mappedBy="pcBooked") **[7]**
17. private Set<Reservation> reservations;
18. //Setter and getter
19. //...
20. }
    1. **@Id** → This annotation is used to point that the attribute pcId will be the primary key of the table.
    2. **@GeneratedValue(strategy = GenerationType.AUTO)** → This annotation specify how to generate the value of pcId. The *GenerationType.A*UTO is the default generation type and lets the persistence provider choose the generation strategy.
    3. **@Column(name="pcId")** → This annotation is used to mention the details of the column related to the attribute pcId in particular the name of this column will be pcId.
    4. **@Column(name="pcNumber")** → has already said above with this annotation we set the column name as pcNumber.
    5. **@ManyToOne** → between the entities Room and PC exists a many to one relation, in facts every PC is associated to one and only one Room and a Room can hold to one or more PC. The *@ManyToOne* annotation is used to create the many-to-one relationship between the Room and PC entities.
    6. **@JoinColumn** → This annotation is used to specify the column for joying an entity association. This column indicate that this entity is the owner of the relationship. It’s also used to specify the name of the column.
    7. **@OneToMany(mappedBy="pcBooked")** → Between PC and Reservation entities exist a many to one relation, in facts every Reservation refer to a PC and a PC can be referred by one or more Reservation. The attribute mappedBy indicates that the entity in this side is the inverse of the relationship, and the owner resides in the "other" entity.

### Entity Room:

1. @Entity
2. public class Room implements Serializable {
4. @Id **[1]**
5. @Column(name = "roomName")**[2]**
6. private String roomName;
8. @Column(name = "capacity")**[3]**
9. private int capacity;
11. @Column(name = "rowsNumber")**[4]**
12. private int rowsNumber;
14. @OneToMany(mappedBy = "pcRoom")**[5]**
15. private Set<PC> PCs;
17. @Transient **[6]**
18. private int availablePCs;
20. //Getter and setter
21. //…
22. }

* 1. **@Id** → This annotation is used to point that the attribute roomName will be the primary key of the table.
  2. **@Column(name="** **roomName ")** → This annotation is used to mention the details of the column related to the attribute roomName in particular the name of this column will be roomName.
  3. **@Column(name="** **capacity ")** → as above this annotation is used to mention the details of the column related to the attribute capacity in particular the name of this column will be capacity.
  4. **@Column(name="** **rowsNumber ")** → again this annotation is used to mention the details of the column related to the attribute roomName in particular the name of this column will be roomName.
  5. **@OneToMany(mappedBy="pcRoom")** → as said before between the entities PC and Room exist a many to one relation that is own by the entity PC and so we use the attributed mapped by = “pcRoom”.
  6. **@Transient** → this annotation is used to indicate that the field availablePCs has not to be persisted in the database. We use this transient field in order to show the room’s number of available PCs in a certain date and time, in the specific table view inside the application.

### Entity Reservation:

1. @Entity
2. public class Reservation implements Serializable {
4. @Id **[1]**
5. @Column(name = "startTime") **[2]**
6. private String startTime;
7. @Id **[3]**
8. @Column(name = "bookingDate") **[4]**
9. private String bookingDate;
10. @Id **[5]**
11. @ManyToOne **[6]**
12. private PC pcBooked;
14. @Transient **[7]**
15. private String roomN;
17. @Transient **[8]**
18. private int pcnumb;
20. @Column(name = "username") **[9]**
21. private String username;
22. //Getter and setter
23. //...
24. }
    1. **@Id** → This annotation is used to point that the attribute startTime will be a part of the primary key of the table.
    2. **@Column(name="** **startTime ")** → This annotation is used to mention the details of the column related to the attribute startTime in particular the name of this column will be startTime.
    3. **@Id** → This annotation is used to point that the attribute bookingDate will be a part of the primary key of the table.
    4. **@Column(name="** **startTime ")** → This annotation is used to mention the details of the column related to the attribute startTime in particular the name of this column will be startTime.
    5. **@Id** → This annotation is used to point that the attribute pcBooked will be a part of the primary key of the table
    6. **@ManyToOne** → as explained before between the entities Reservation and PC exists a many to one relation. The *@ManyToOne* annotation is used to create the many-to-one relationship between the Reservation and PC entities that is own by this entity.
    7. **@Transient** → as said above this annotation is used to indicate that the field roomN is not be persisted in the database. We use this transient field just to show the reserved PC’s room name in the user reservations’ TableView.
    8. **@Transient** → again this annotation is used to indicate that the field pcnumb is not be persisted in the database. We use this transient field just to show the reserved PC’s number in the user reservations’ TableView.
    9. **@Column(name="** **username ")** → This annotation is used to mention the details of the column related to the attribute username in particular the name of this column will be username.

## LevelDB Implementation

### LevelDB Database

As shown in the feasibility study, we decided to implement the key-value Database for the User table. A new class ‘LDBManager’ has been implemented to manage the data referred to the users’ information. This class creates (if not exists) a new file named UserDB.

### LevelDB content format

The data for each user has been formatted in a human readable string:

user:<userId>:username=<userUsername>

user:<userId>:password=<userPassword>

user:<userId>:firstName=<userFirstName>

user:<userId>:lastName=<userLastName>

user:<userId>:matriculationNumber=<userMatriculationNumber>

### LevelDB collision inside the UserDB file

The userId is formatted in this way: FirstName\_FirstCharacter.LastName

According to the format shown before, it can happen that two or more different users have the same userId. In order to avoid that, we generate the userIds in this way: during the sign-up process, if the generated userId happens to be already in the database, we set an auto-incremental integer parameter counting the number of users that share the same couple (FirstName\_FirstCharacter, LastName).

For instance, if the user “Marco Rossi” tries to sign up, but the user “Mario Rossi” already exists in the database (whose userId is “m.rossi”), Marco Rossi’s userId will be generated as follows:

Mario Rossi sign up -> userId=”m.rossi”

Marco Rossi sign up -> collision with “m.rossi” -> userId=”m.rossi1”

## UML Diagram

Immagine che contiene testo, mappa

Descrizione generata automaticamente

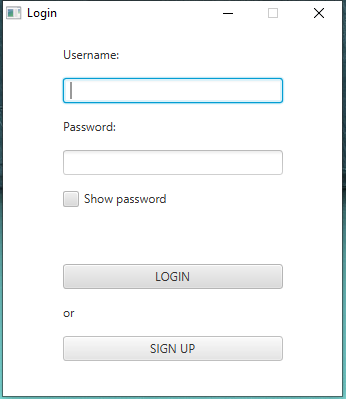
## Classes

|  |  |  |
| --- | --- | --- |
| CLASS NAME | | DESCRIPTION |
| FRONT-END | **MainPaneGUI** | It displays the GUI of the application and contains the methods for the event handlers |
| **SignupPaneGUI** | It displays the GUI of the signup service and contains the methods for the event handler |
| **LoginPaneGUI** | It displays the GUI of the login service and contains the method for the event handler |
| **PCIcon** | It represents each pc position in the map of a room |
| **AvailableRoomsTable** | It contain the methods to show to the user the status of each room in the campus |
| **ResevationsTable** | It contain the methods to show each future reservation made by the user |
| MIDDLEWARE | **PCBookingApplicationController** | It’s the main class of the application. It allows the connection between the Front-end classes and the back-end ones |
| BACK-END | **Room** | It’s a JPA entity that defines the structure of Room’s table in the db and collects all the information about a specific room in the campus |
| **User** | It collects the personal data of a specific user |
| **PC** | It’s a JPA entity that defines the structure of PC’s table in the db and collects the information about a PC in a certain room |
| BACK-END | **Reservation** | It’s a JPA entity that defines the structure of Room’s table in the db and collects the data of a reservation made by a specific user |
| **JPAManager** | This class is used to manage the functionalities of JPA. It contains the needed methods for making the application relate with the DB using JPA. |
| **LDBManager** | It’s the class which implements the functionalities of key-value databases. It allows to store information about the users and contains the definitions of the methods for login and sign up |

## User manual

### Login

When the application is launched, the system will display a login form. The user has to put in its username and password and then to click on “LOGIN” button. If desired, the user can also see its password as plain text by clicking on the “Show/Hide Password” radio button.



### Sign-up

If the user doesn’t have credentials for the application, he can register himself to the service pressing the button SIGN UP. He has to insert his First Name, Last Name, Matriculation number and the desired password. After pressing the button REGISTER, the system will show the assigned username.

Immagine che contiene screenshot

Descrizione generata automaticamente

### Working with the application

After the login, the user is given a list of all its reservation made for future days. The data is displayed in a table, showing the room, the pc number the date and time of the reservation.

Immagine che contiene screenshot

Descrizione generata automaticamente

The user has to choose a date and time in the form in the left upper corner and then to click the “FIND” button to watch the status of the rooms in the campus at the specified moment.

The table “Available Rooms” will be filled with a row for each room in the db, indicating some information about the room itself, like the capacity and the number of the available PCs in the room

Immagine che contiene screenshot

Descrizione generata automaticamente

The user has now to click on a room from the table and click on the “RESERVE” button, to create a new reservation in his name. A map of the room will be displayed, highlighting the position of the PC assigned to the user. Moreover, the tables will be updated: “Your Reservations” table will display another row including the information about the reservation, while “Available Rooms” table will show the updated value of the available PCs in the rooms.

Immagine che contiene screenshot

Descrizione generata automaticamente

### Removing a reservation

Any time the user wants to delete a reservation, it simply has to click on the reservation that it wants to delete from “Your Reservations” table, then just to click on the “DELETE” button; the tables are updated accordingly.

Immagine che contiene screenshot

Descrizione generata automaticamente