UNIVERSITÀ DI PISA

Computer Engineering
Artificial Intelligence and Data Engineering

WeSport Project

Distributed Systems and Middleware Technologies
Project Documentation

Gaia Anastasi
Leonardo Poggiani
Riccardo Orrù

Academic Year 2021/2022

Contents

1 Introduction **3**

2 Architecture **4**
2.1 Client Application 4
2.2 Server Application 4
2.2.1 EJB 5

3 Implementation **6**
3.1 JDBC connection pooling 6
3.2 Relation database 6
3.3 Project Structure 8
3.3.1 ejb-interfaces 9
3.3.2 ejb-impl 10
3.3.3 erlang-server 11
3.3.4 WeSport_webapp 13

4 User manual **14**
4.1 How to run the chat application 14
4.2 Ordinary user 15
4.2.1 Registered users 15
4.2.2 Booking 16
4.2.3 Personal profile 18
4.2.4 Chat 19
4.2.5 Chatroom 19
4.3 Admin user 20
4.3.1 Remove a booking 20
4.3.2 Remove an user 21
4.3.3 Chat 22

1 Introduction

WeSport is web application to book fields for various sport activities (futsal, tennis, rugby and basketball). The aim of this application is to allow people to book a field but to give them also the opportunity to interact with other users though a chat service in order to find new users to play with. Once you have find all the people to play with, you can make a reservation. The webapp provides the possibility to chat with a single player but as well the possibility to join a group chat to interact with players who share your same sport interests. A user can:

- book the field
- participate in a game

Each user has its own personal page where the number of games played, the number of sports played and the rating provided by other players on its skill level in various sports are shown.

2 Architecture

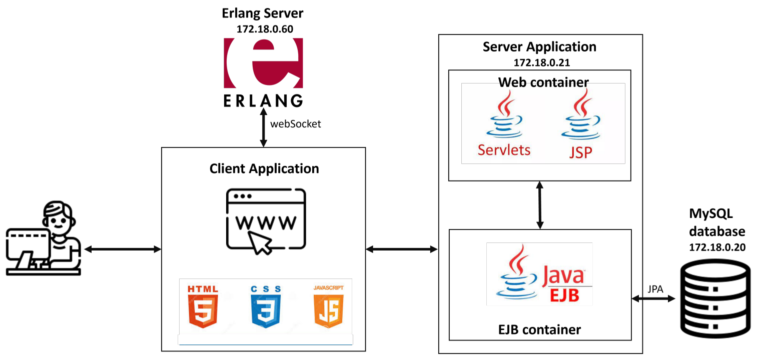


Figure 1: Project architecture

The web application is composed by a *Client Application*, an *Erlang Server* and a *Server Application* which communicates with a *relational database*.The server application is deployed on container 172.18.0.21, mySQL database is on container 172.18.0.20 while the Erlang Server is deployed on the container 172.18.0.60. It is possible to access the container using *UNIPI VPN*. The web application can be accessed through the following link:
http://172.18.0.21:8080/WeSport_webapp/

2.1 Client Application

The client side is developed in *HTML, javascript* and *CSS*. In order to have dynamic data-driven pages *JavaServer Pages (JSP)* is used. The communication with Erlang Server was implemented through *Web Socket*, a messaging protocol allowing asynchronous communication over a TCP connection. The communication is handled using *javascript WebSocket*. The communication with the Server Application was instead implemented using a REST approach which is a method that exploits HTTP as transfer protocol and embeddes the requested resources directly into the HTTP response message.

2.2 Server Application

The server side is implemented through *Java Servlet* and *JavaServer Pages (JSP)* in order to handle HTTP requests and content generation and through *EJBs* it handles the resources made available to the user. It is also connected with a relation database (in our case MySQL), in which information about users and bookings are stored, and it interact with the database by means

of *Java Persistence API (JPA).*
In this project, *GlassFish* is used for application server. Glassifish is the reference implementation for the JEE. In the Java EE server, we can identify the Web Container, that is the container for servlets and JSP. The container interacts with another container for generic components, that in JEE are named EJBs (Enterprise Java Beans).

2.2.1 EJB

About EJBs, in this project Stateless EJBs were used because, given the purpose of the application, there was no need to maintain state information between invocations. The use of stateless EJBs also ensure that there is no need to to deal with synchronization and concurrency problems because each client will obtain a different instance of the EJB upon calling a method because of EJB container.

3 Implementation

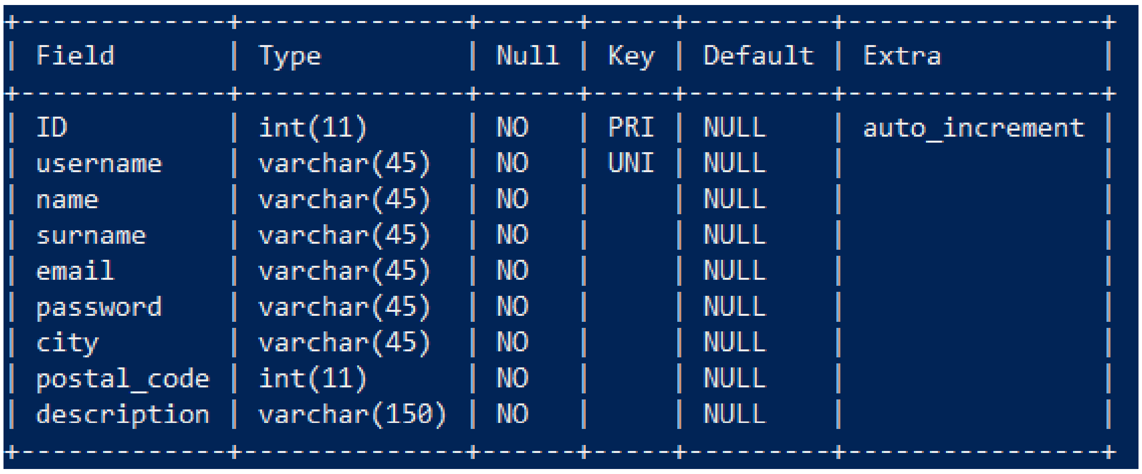
3.1 JDBC connection pooling

For the purpose of the application a JDBC resource was created, identified by an unique JNDI name. The resource was specified using *Glassfish Admin Console* entering *jdbc/wesport_pool*. The advantage of using a JDBC resource is that a resource is associated to a connection pool for a particular database. Exploiting a connection pool will avoid time and computing consumption due to the creation of a new physical connection.

3.2 Relation database

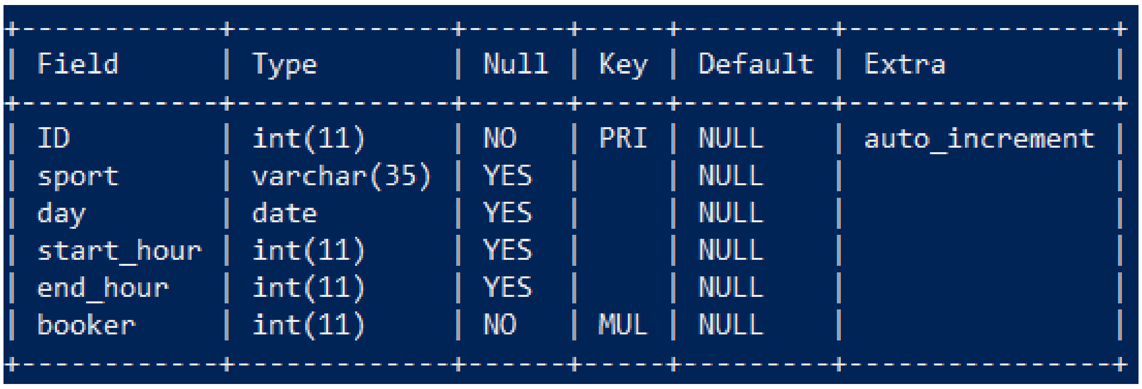
The relational database is implemented using **MySQL** database. The database is composed by 3 tables:

- *User* table : to store information about each user
- *Booking* table: to store information about bookings
- *User_Booking* table: to handle the [N:N] cardinality between user and booking entities.



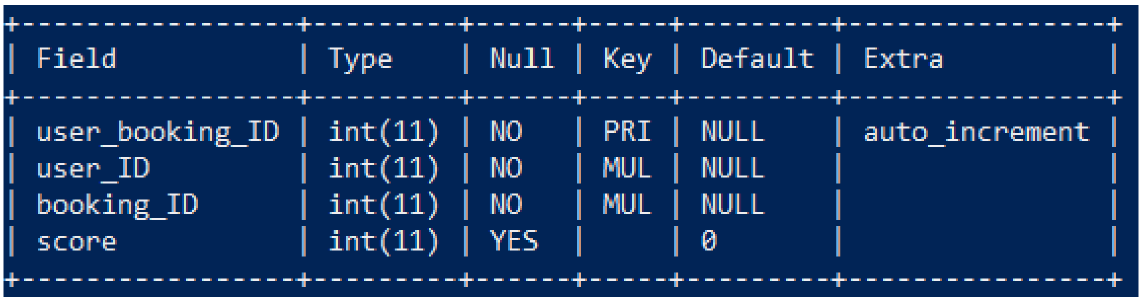
```
+-------------+--------------+------+-----+---------+----------------+
| Field       | Type         | Null | Key | Default | Extra          |
+-------------+--------------+------+-----+---------+----------------+
| ID          | int(11)      | NO   | PRI | NULL    | auto_increment |
| username    | varchar(45)  | NO   | UNI | NULL    |                |
| name        | varchar(45)  | NO   |     | NULL    |                |
| surname     | varchar(45)  | NO   |     | NULL    |                |
| email       | varchar(45)  | NO   |     | NULL    |                |
| password    | varchar(45)  | NO   |     | NULL    |                |
| city        | varchar(45)  | NO   |     | NULL    |                |
| postal_code | int(11)      | NO   |     | NULL    |                |
| description | varchar(150) | NO   |     | NULL    |                |
+-------------+--------------+------+-----+---------+----------------+
```

Figure 2: User table



```
+------------+--------------+------+-----+---------+----------------+
| Field      | Type         | Null | Key | Default | Extra          |
+------------+--------------+------+-----+---------+----------------+
| ID         | int(11)      | NO   | PRI | NULL    | auto_increment |
| sport      | varchar(35)  | YES  |     | NULL    |                |
| day        | date         | YES  |     | NULL    |                |
| start_hour | int(11)      | YES  |     | NULL    |                |
| end_hour   | int(11)      | YES  |     | NULL    |                |
| booker     | int(11)      | NO   | MUL | NULL    |                |
+------------+--------------+------+-----+---------+----------------+
```

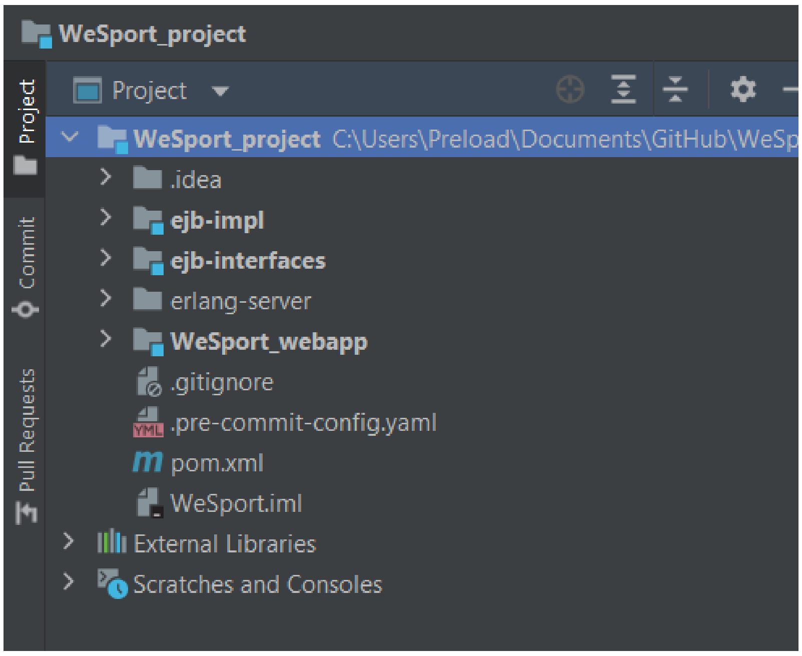
Figure 3: Booking table



```
+-----------------+---------+------+-----+---------+----------------+
| Field           | Type    | Null | Key | Default | Extra          |
+-----------------+---------+------+-----+---------+----------------+
| user_booking_ID | int(11) | NO   | PRI | NULL    | auto_increment |
| user_ID         | int(11) | NO   | MUL | NULL    |                |
| booking_ID      | int(11) | NO   | MUL | NULL    |                |
| score           | int(11) | YES  |     | 0       |                |
+-----------------+---------+------+-----+---------+----------------+
```

Figure 4: User_Booking table

3.3 Project Structure



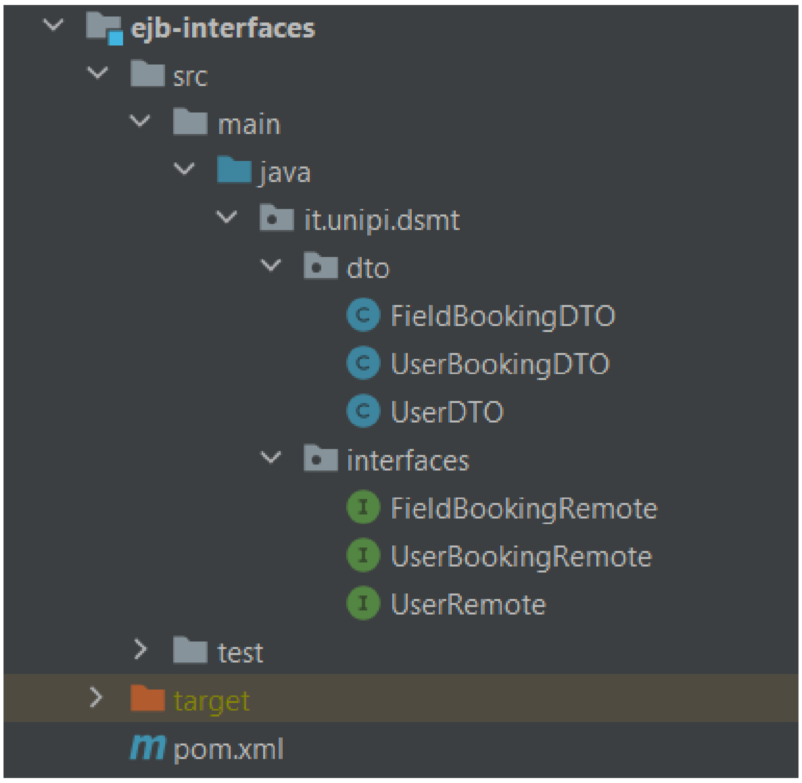
```
WeSport_project C:\Users\Preload\Documents\GitHub\WeSp
  .idea
  ejb-impl
  ejb-interfaces
  erlang-server
  WeSport_webapp
  .gitignore
  .pre-commit-config.yaml
  pom.xml
  WeSport.iml
External Libraries
Scratches and Consoles
```

Figure 5: Project structure

The main folder is WeSport_project inside which the main folders are:

- ejb-impl
- ejb-interfaces
- erlang-server
- WeSport_webapp

3.3.1 ejb-interfaces

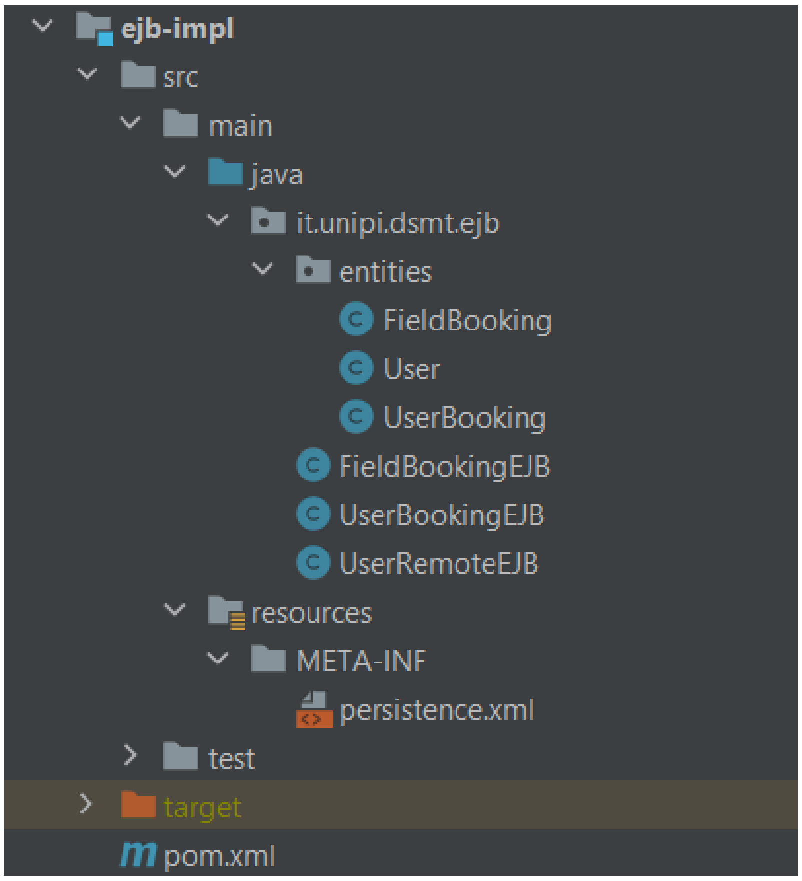


```
ejb-interfaces
  src
    main
      java
        it.unipi.dsmt
          dto
            FieldBookingDTO
            UserBookingDTO
            UserDTO
          interfaces
            FieldBookingRemote
            UserBookingRemote
            UserRemote
    test
  target
  pom.xml
```

Figure 6: ejb-interfaces folder

In the ejb-interfaces folder there are the EJB interfaces which represents the contract with the client. The interfaces are all **remote** interfaces. In this folder there is as well a package called *dto* containing the *Data Type Object* classes corresponding to the entities contained in the ejb-impl module that we will analyze in the next section.

3.3.2 ejb-impl



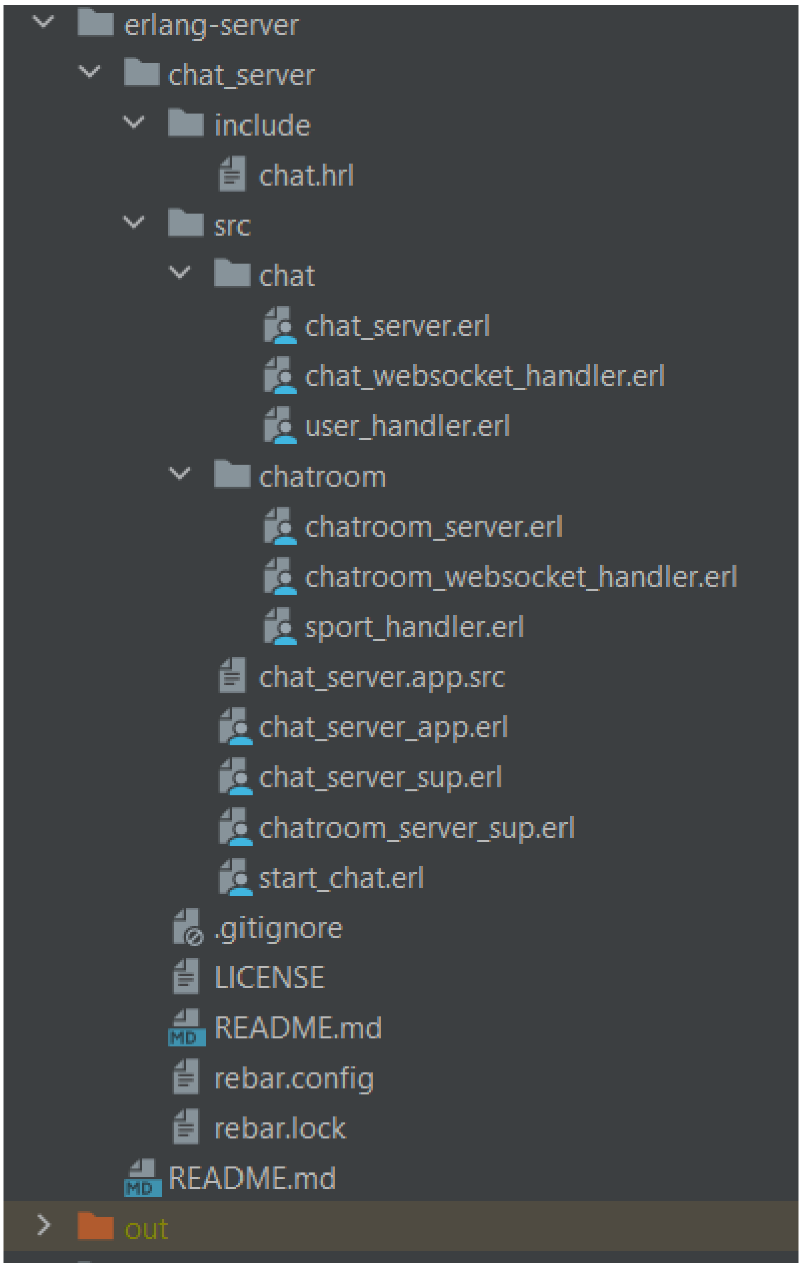
```
ejb-impl
  src
    main
      java
        it.unipi.dsmt.ejb
          entities
            FieldBooking
            User
            UserBooking
          FieldBookingEJB
          UserBookingEJB
          UserRemoteEJB
      resources
        META-INF
          persistence.xml
    test
  target
  pom.xml
```

Figure 7: ejb-impl folder

The ejb-impl folder contains the classes for the ejb-interfaces implementations. As previously mentioned, the type of EJB class implemention chosen is *Stateless*. In the folder a package called *entities* is also present. This package contains the entities used to transform the tables of the database mentioned before into object over which JPA query can be executed.
In the folder is also present the *persistence.xml* file which contains the link for the JDBC resource:

```
<jta-data-source>jdbc/wesport_pool</jta-data-source>
```

3.3.3 erlang-server



```
erlang-server
  chat_server
    include
      chat.hrl
    src
      chat
        chat_server.erl
        chat_websocket_handler.erl
        user_handler.erl
      chatroom
        chatroom_server.erl
        chatroom_websocket_handler.erl
        sport_handler.erl
      chat_server.app.src
      chat_server_app.erl
      chat_server_sup.erl
      chatroom_server_sup.erl
      start_chat.erl
    .gitignore
    LICENSE
    README.md
    rebar.config
    rebar.lock
  README.md
out
```

Figure 8: erlang-server folder

The Erlang server is composed of two parts: one that manages the chat between users in one-to-one mode, the other provides the one-to-many chat service also called *chatroom*. Inside the one-to-one chat it's possible to exchange messages with only one active user at a time while inside the chatroom it's possible to send a message to all the people connected on that specific topic (sport). The Erlang server communicates with the web application through the *Javascript websocket* that allow to send and retrieve messages from the chat server.

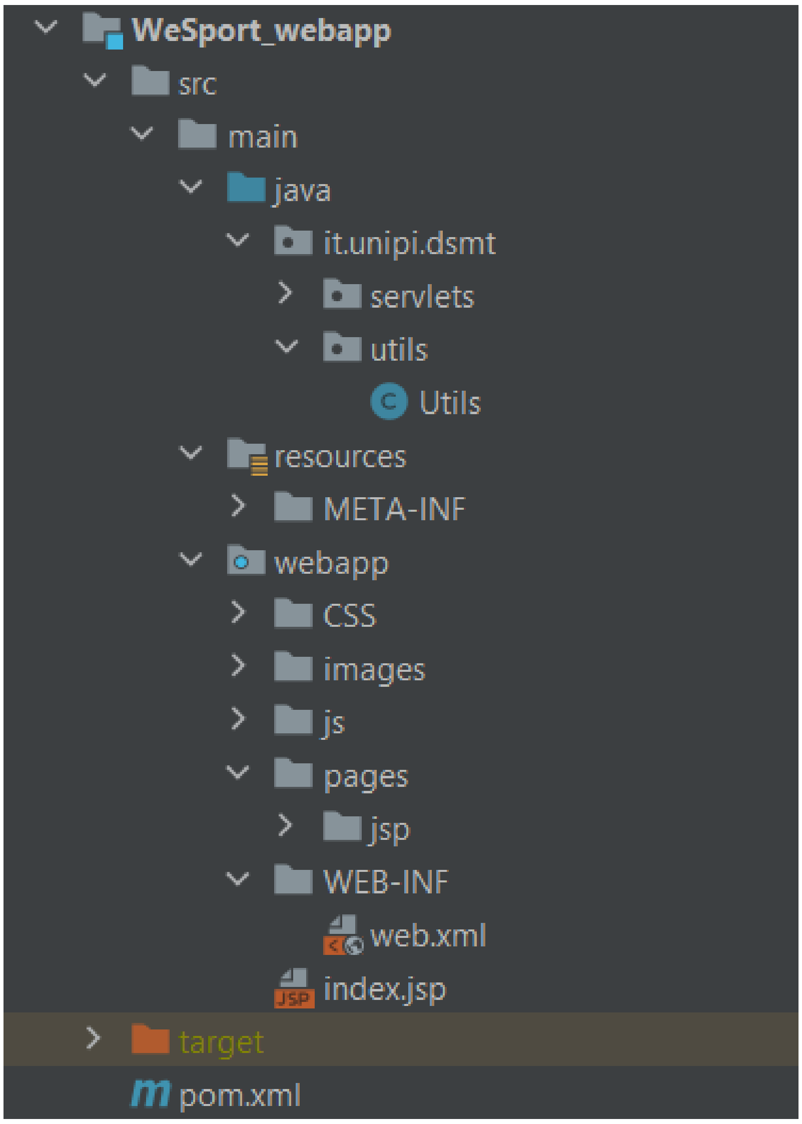
For the management of these two different services (chat and chatroom) the Erlang server is listening on two different ports (3308 and 3307) to make sure that the messages intended for the two different chats are not mixed. This is also because the two chats behave slightly differently from each other: the one-to-one chat shows as online users all those who have entered the chat page (and therefore started the websocket with the server). One-to-many chat, on the other hand, considers online all those users who are inside the chatroom page and have selected the same sport. The message will be delivered according to the sport selected by the user and not according to his username.

Chat: In one-to-one chat, users are stored as they enter the chat page by their username and PID, which is necessary to be able to send back direct messages to the user. Users' usernames and PIDs are stored in an ets structure, so it's possible to check if a user has already logged in or not.
When the user exits the page, the entry related to his username is deleted and the message will not be delivered anymore. In the file *chat_server.erl* is specified the listening port and the handler of the websocket to be used that manages the messages coming from the webapp.

Chatroom: What we said for the chat is also valid for the chatroom with the difference that inside the file *sport_handler.erl* we won't make the insertion and the control on a username and a PID but on a sport and a PID. In fact all the users who connect to the chat page are inserted in an ets structure according to their PID and the chosen sport and when a message will be sent on a chatroom to which they belong, the *chatroom_server* will take care to retrieve the list of connected users and send the message to each user.

The Erlang code can be compiled and executed by following the instructions in the user's manual.

3.3.4 WeSport_webapp



```
WeSport_webapp
  src
    main
      java
        it.unipi.dsmt
          servlets
          utils
            Utils
      resources
        META-INF
      webapp
        CSS
        images
        js
        pages
          jsp
        WEB-INF
          web.xml
        index.jsp
  target
  pom.xml
```

Figure 9: webapp folder

This folder contains the sevlets that are used to handle different client requests in a concurrent and distributed way. In the folder there are as well *JavaServer Pages (JPS)* to implement servlet specification.

4 User manual

4.1 How to run the chat application

To run the chat application you first need to connect to the VPN *students.ovpn*. After logging into the host root@172.18.0.60 you need to enter the folder *erlang-server/chat-server/*. From this folder you must first compile the Erlang files via Rebar with all the necessary dependencies (*cowboy* library) through the command:

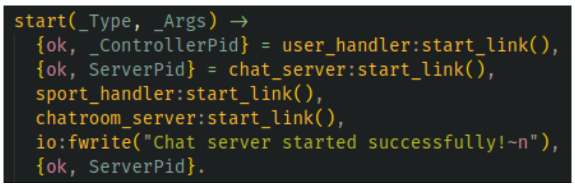
```
# rebar3 compile
```

and finally we can run a shell through the command:

```
# rebar3 shell
```

.

In this way we will have started an Erlang shell with all the dependencies we need and we can then start the chat server through the method *start_chat:start(opt,opt)* which takes care of starting both the one-to-one and the one-to-many chat servers.



```
start(_Type, _Args) ->
  {ok, _ControllerPid} = user_handler:start_link(),
  {ok, ServerPid} = chat_server:start_link(),
  sport_handler:start_link(),
  chatroom_server:start_link(),
  io:fwrite("Chat server started successfully!~n"),
  {ok, ServerPid}.
```

Figure 10: Start chat

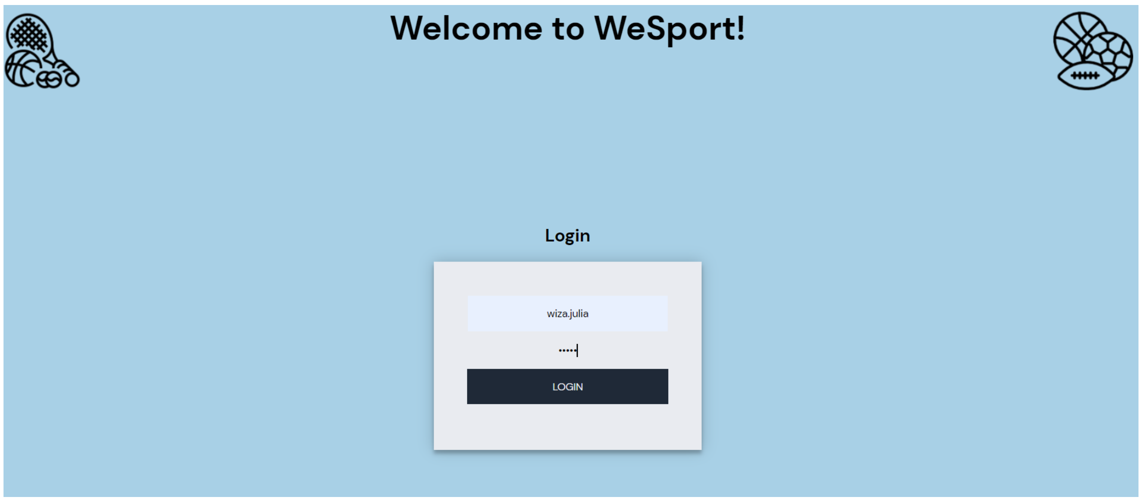


Figure 11: Login page

An user needs to login in to use the service. The admin logs in using *admin, admin* as its credentials.

4.2 Ordinary user

After loggin it, the following homepage is shown to the user. The user can chose which operation to perform among the one listed.

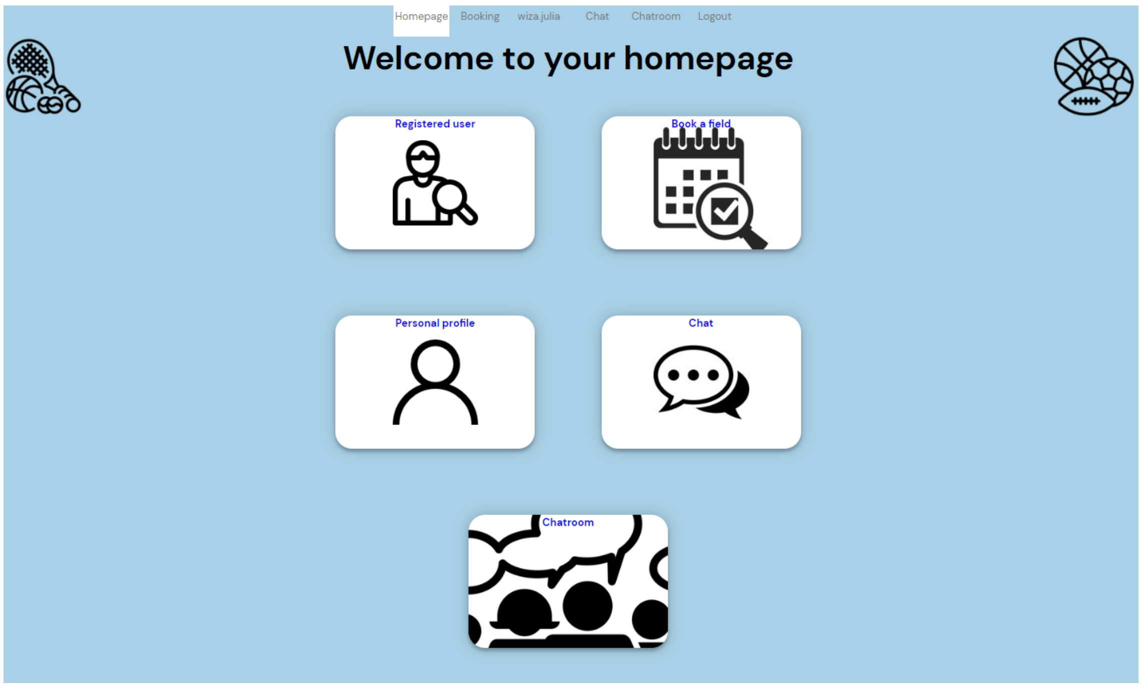
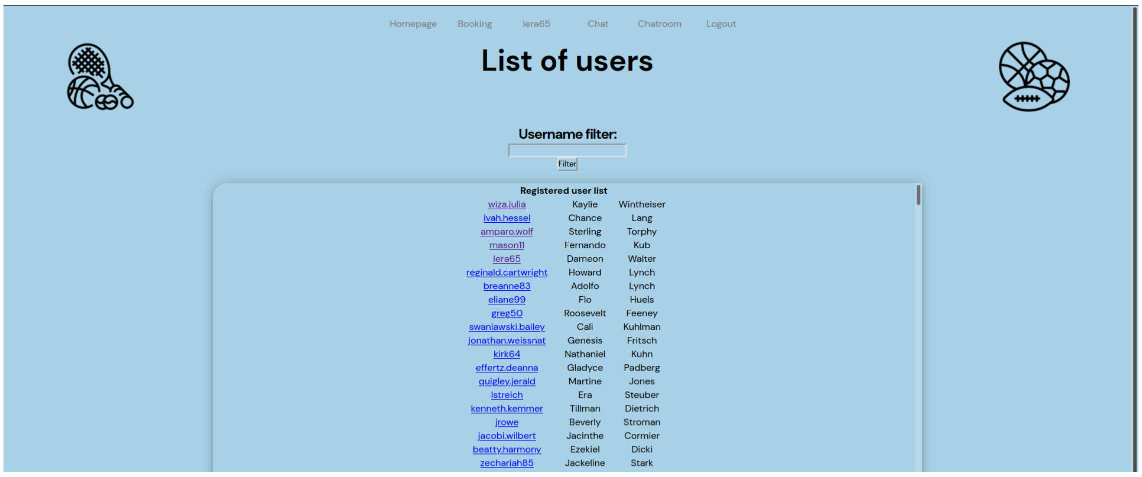


Figure 12: Homepage

4.2.1 Registered users

Clicking on *Registered users* a list of all the registered users will be shown. The user could filtered the list by username.



List of users

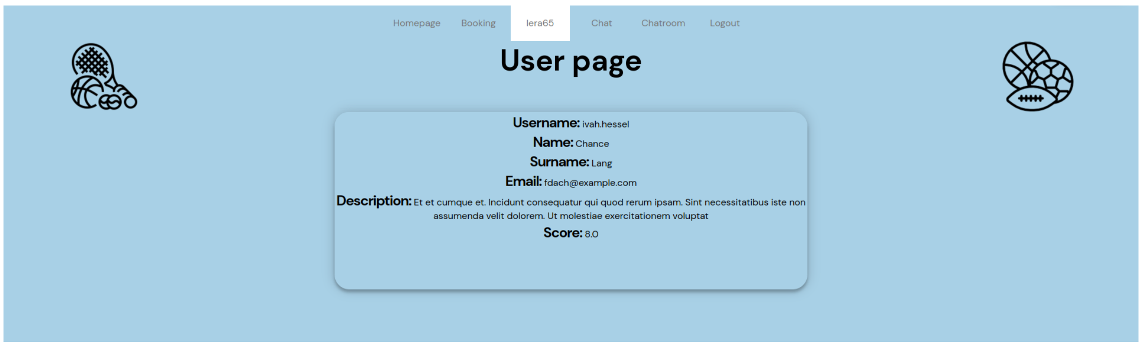
Username filter:

Registered user list

wiza.julia	Kaylie	Wintheiser
ivah.hessel	Chance	Lang
amparo.wolf	Sterling	Torphy
mason11	Fernando	Kub
lera65	Dameon	Walter
reginald.cartwright	Howard	Lynch
breanne83	Adolfo	Lynch
eliane99	Flo	Huels
greg50	Roosevelt	Feeney
swaniawski.bailey	Cali	Kuhlman
jonathan.weissnat	Genesis	Fritsch
kirk64	Nathaniel	Kuhn
effertz.deanna	Gladyce	Padberg
quigley.jerald	Martine	Jones
lstreich	Era	Steuber
kenneth.kemmer	Tillman	Dietrich
jrowe	Beverly	Stroman
jacobi.wilbert	Jacinthe	Cormier
beatty.harmony	Ezekiel	Dicki
zechariah85	Jackeline	Stark

Figure 13: List of user page

If a user clicks on the username it will appear the profile of the selected user.



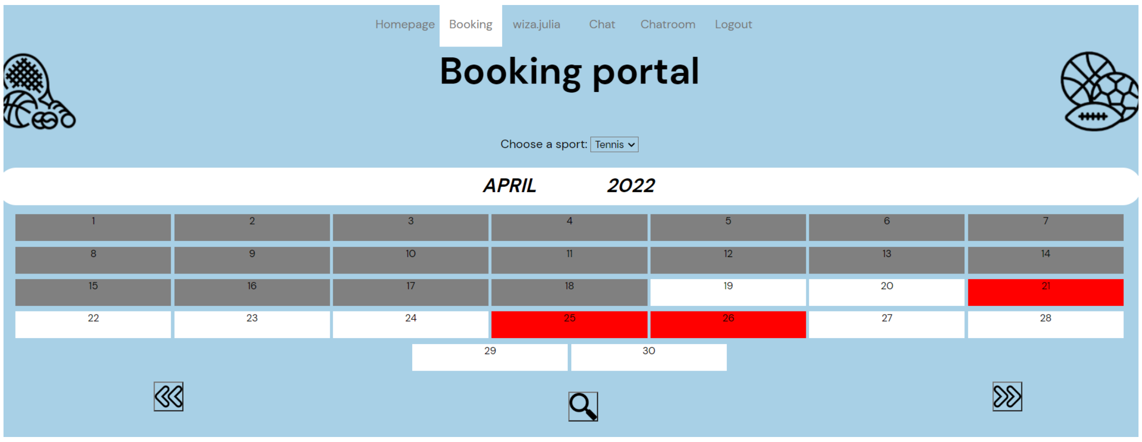
User page

Username: ivah.hessel
Name: Chance
Surname: Lang
Email: fdach@example.com
Description: Et et cumque et. Incidunt consequatur qui quod rerum ipsam. Sint necessitatibus iste non assumenda velit dolorem. Ut molestiae exercitationem voluptat
Score: 8.0

Figure 14: List of user page

4.2.2 Booking

Clicking here the user can insert a new booking. After clicking on it the following page will be shown to the user.



1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

Figure 15: Booking page

The user could now chose the day (s)he wants to book the field among the white ones. After choosing the day the user will be redirect towards the timeslot page to chose which timeslot (s)he prefers among the available ones (the white ones).

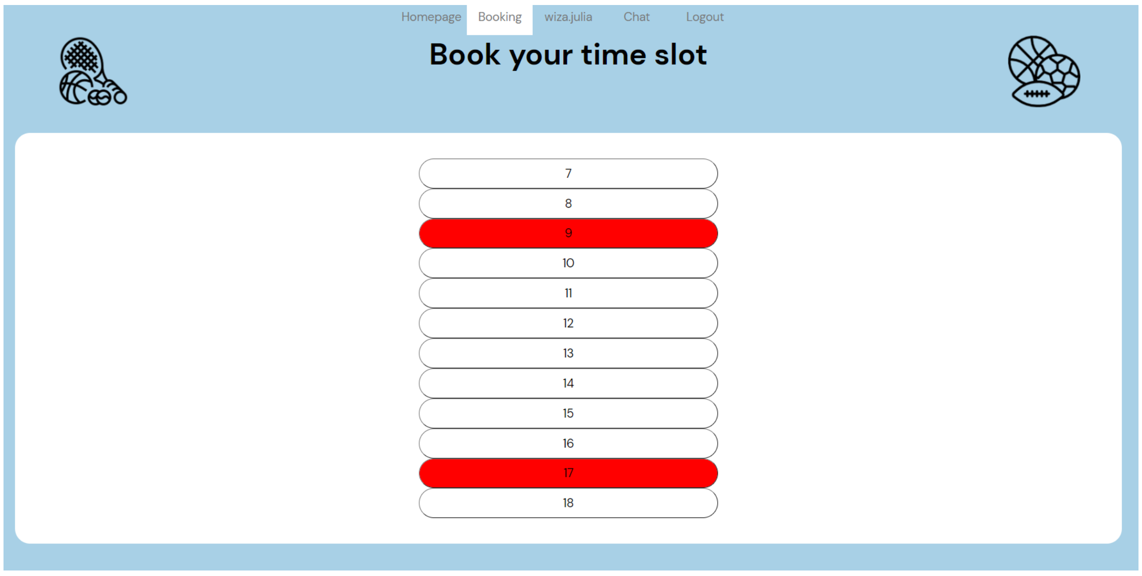
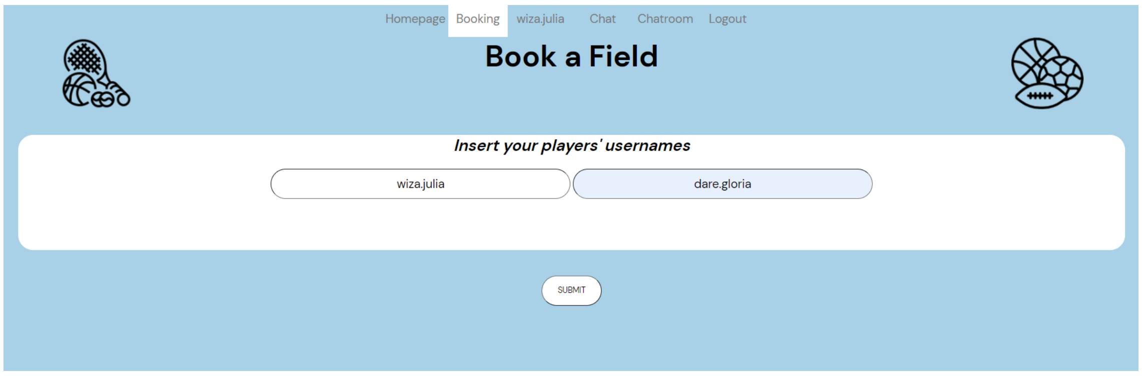


Figure 16: Timeslot page

Finally, after having chosen the timeslot, the user should insert the user-names of the players who will play on the field with.



Book a Field

Insert your players' usernames

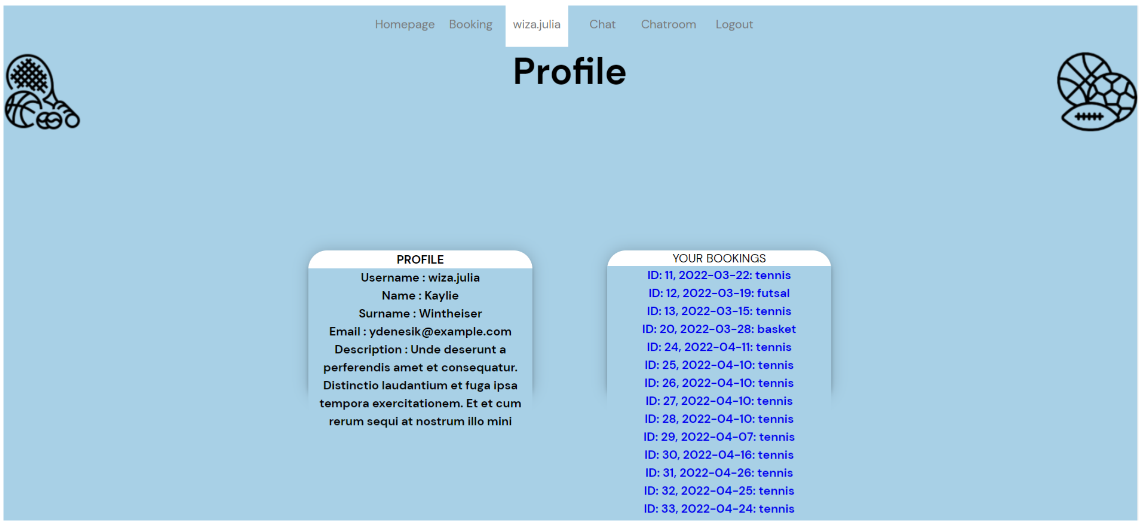
wiza.julia

dare.gloria

Figure 17: Insert players' username page

4.2.3 Personal profile

Using *personal profile* the user could see information about it's account like its username or its description, or the user can see as well a list of its past bookings.



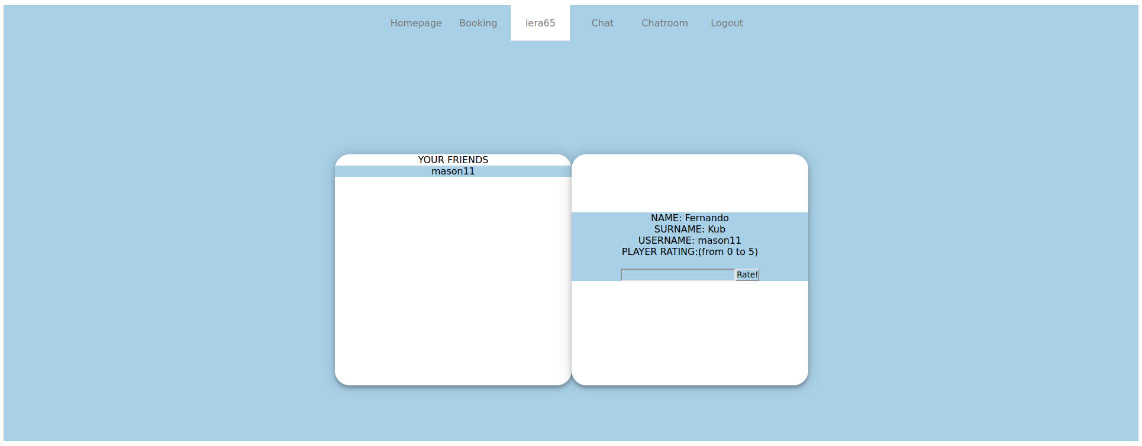
Profile

PROFILE
Username : wiza.julia
Name : Kaylie
Surname : Wintheiser
Email : ydenesik@example.com
Description : Unde deserunt a perferendis amet et consequatur. Distinctio laudantium et fuga ipsa tempora exercitationem. Et et cum rerum sequi at nostrum illo mini

YOUR BOOKINGS
ID: 11, 2022-03-22: tennis
ID: 12, 2022-03-19: futsal
ID: 13, 2022-03-15: tennis
ID: 20, 2022-03-28: basket
ID: 24, 2022-04-11: tennis
ID: 25, 2022-04-10: tennis
ID: 26, 2022-04-10: tennis
ID: 27, 2022-04-10: tennis
ID: 28, 2022-04-10: tennis
ID: 29, 2022-04-07: tennis
ID: 30, 2022-04-16: tennis
ID: 31, 2022-04-26: tennis
ID: 32, 2022-04-25: tennis
ID: 33, 2022-04-24: tennis

Figure 18: Profile page

If the user click on a booking she/he can rate the other players to suggest others to play with them.



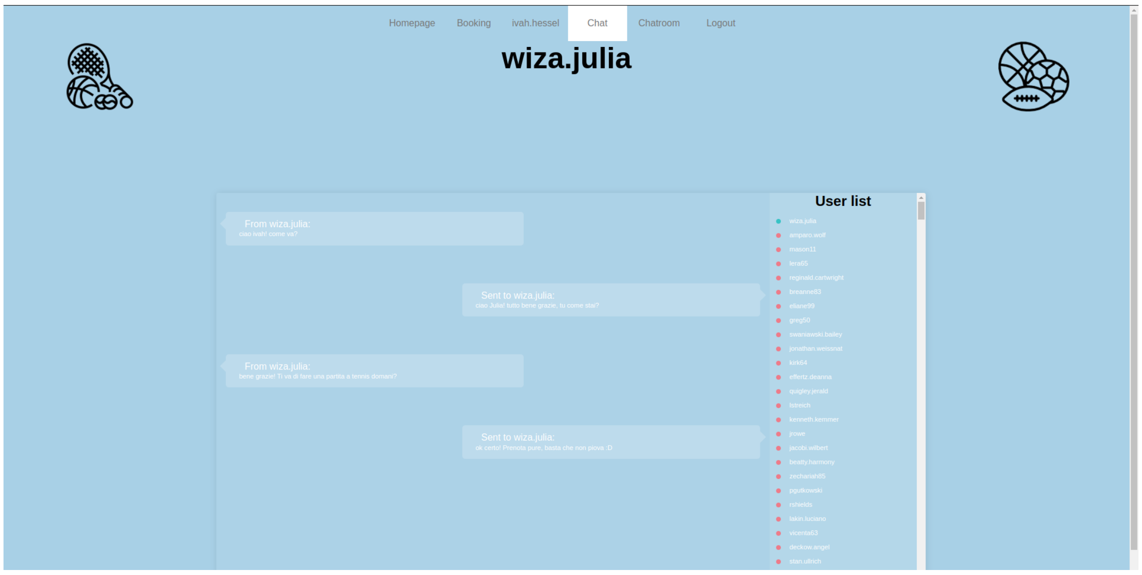
mason11

NAME: Fernando
SURNAME: Kub
USERNAME: mason11
PLAYER RATING:(from 0 to 5)

Figure 19: Profile page

4.2.4 Chat

The chat service can be used by users to interact with other online users. The user just has to select another online user and start typing messages to interact with him(her).



wiza.julia

From wiza.julia:
ciao ivah! come va?

Sent to wiza.julia:
ciao Julia! tutto bene grazie, tu come stai?

From wiza.julia:
bene grazie! Ti va di fare una partita a tennis domani?

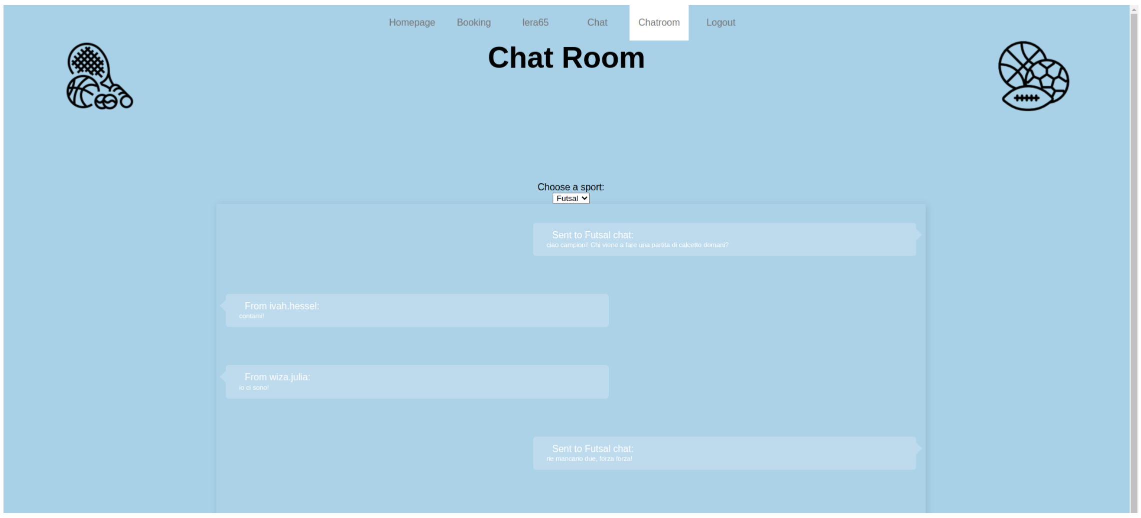
Sent to wiza.julia:
ok certo! Prenota pure, basta che non piova :D

● wiza.julia
● amparo.wolf
● mason11
● lera65
● reginald.cartwright
● breanne83
● eliane99
● greg50
● swaniawski.bailey
● jonathan.weissnat
● kirk64
● effertz.deanna
● quigley.jerald
● lstreich
● kenneth.kemmer
● jrowe
● jacobi.wilbert
● beatty.harmony
● zechariah85
● pgutkowski
● rshields
● lakin.luciano
● vicenta63
● deckow.angel
● stan.ullrich

Figure 20: Chat page

4.2.5 Chatroom

The chatroom service is very similar to chat service with the only difference that a chatroom is a group chat where people who share the interest in the same sport can interact and talk between each other. The user should just select the sport for which (s)he wants to entered the chatroom.



Sent to Futsal chat:
ciao campioni! Chi viene a fare una partita di calcetto domani?

From ivah.hessel:
contami!

From wiza.julia:
io ci sono!

Sent to Futsal chat:
ne mancano due, forza forza!

Figure 21: Chatroom page

4.3 Admin user

Admin is a special user who can delete a booking, or remove an user from the service. It is also provided with a chat service to inform users about events, or to warn people who does not behave correctly. After logging in, the following page is shown to the admin.

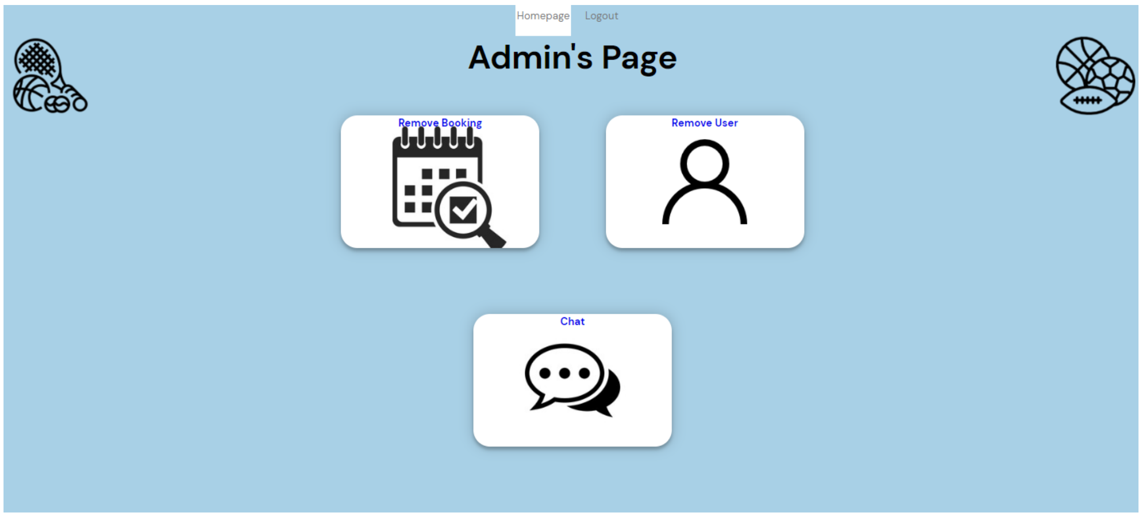
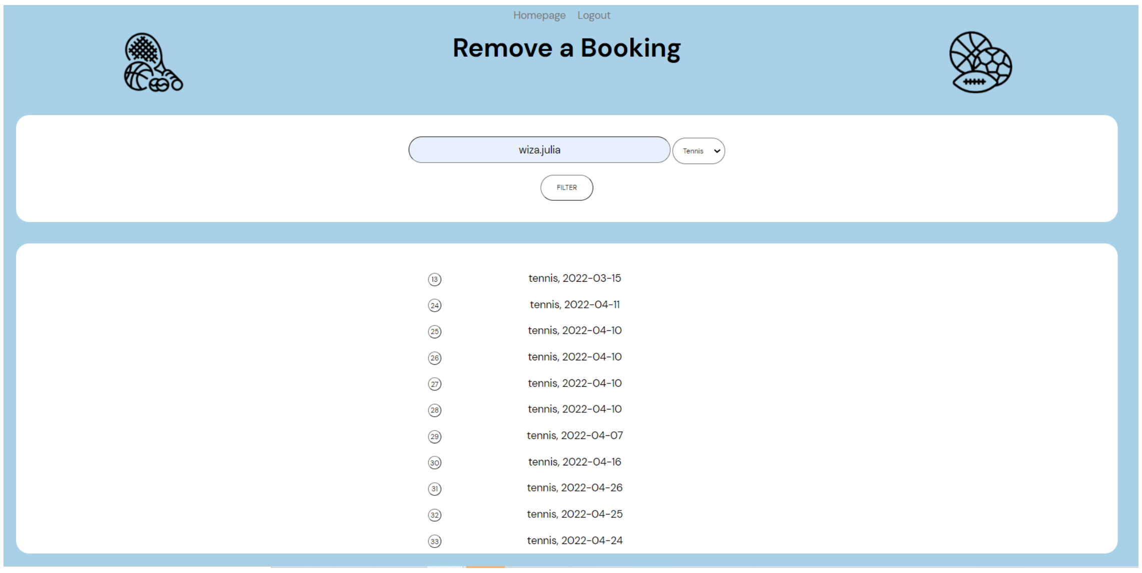


Figure 22: Admin's page

4.3.1 Remove a booking

Admin can remove a booking already inserted using the following page.


Homepage Logout

Remove a Booking

wiza.julia Tennis

FILTER

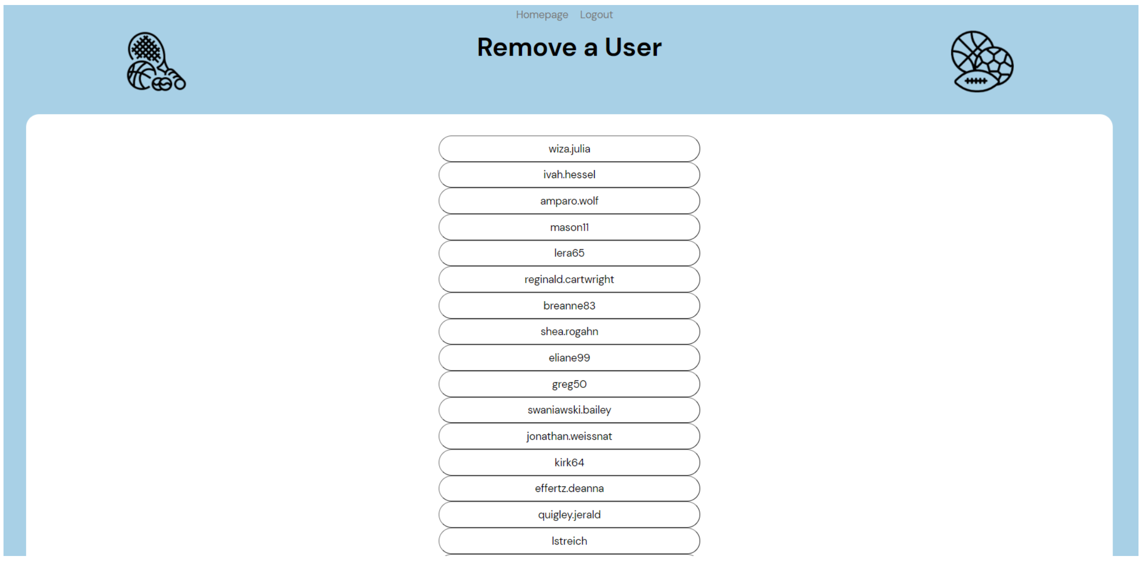
13 tennis, 2022-03-15
24 tennis, 2022-04-11
25 tennis, 2022-04-10
26 tennis, 2022-04-10
27 tennis, 2022-04-10
28 tennis, 2022-04-10
29 tennis, 2022-04-07
30 tennis, 2022-04-16
31 tennis, 2022-04-26
32 tennis, 2022-04-25
33 tennis, 2022-04-24

Figure 23: Remove a booking page

Admin can filtered using both the username and the sport type, or just one of the previously specified attributes and then (s)he needs to select the booking ID of the booking (s)he wants to delete.

4.3.2 Remove an user

Admin can decide to remove an user from the service using this page. When the page is loaded a list of all the registered users is shown. Admin just needs to select the user to remove.


Homepage Logout

Remove a User

wiza.julia
ivah.hessel
amparo.wolf
mason11
lera65
reginald.cartwright
breanne83
shea.rogahn
eliane99
greg50
swaniawski.bailey
jonathan.weissnat
kirk64
effertz.deanna
quigley.jerald
lstreich

Figure 24: Remove an user page

4.3.3 Chat

The chat page is the same of the ordinary user's one. Admin can use this chat to inform users about important information, warn them to behave correctly or inform them about the deletion of a booking.