UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Web Applications, A.Y. 2020/2021 Master Degree in Computer Engineering Master Degree in ICT for Internet and Multimedia

Homework 1 – Server-side Design and Development

Submission date: 23 April 2021

Last Name	First Name	Badge Number
Candon	Matteo	2020353
Cogato	Matteo	2026966
Peloso	Mario Giovanni	2026828
Piva	Alberto	2030927
Prando	Gianmarco	2019170

Objectives

The objective of the project, in continuation with the one developed during the Foundation of Databases course, is to develop a web interface to manage data for a municipal library. The project is focused on enrolling users and Cultural Association members and giving them the possibility to book seats or rooms in the library during the Covid-19 pandemic.

Main functionalities

This web application will be mainly used to allow registered users to book a seat inside the library (with different booking option based on the logged account) and to enroll for the events that are taken in the conference room.
Also, the organizer of the events will use the web application to book the conference room for events and see who is enrolled for them.
The web interface allows also to modify user's data and add new users.

The website is divided in 4 main areas:

- Public area: this area corresponds to the homepage of the web application and it is accessible to both registered and unregistered users. Here can be found basic information about the library and the events programmed for the present day.
- User area: in this area all different role of users account will be able to:
    - Create new seat reservations;
    - Create new conference reservations;
    - Search for particular reservation or for all its reservations;
    - Update the information about an existing reservation.
- Conference area: this area will be accessible only by an organizer account and he can:
    - Create new conferences providing the title, the description and the date;
    - View the users who booked for the conference;

    - Delete a conference: when this happen all the user reservations for this conference will be deleted.
- Administrator area: this area is accessible only by an “association_admin” role of user account and the functionalities are:
    - Update the role of an existing user: providing the new role (which can be “associaton_member” or “association_admin”) of the user account and with all the information about him the administrator can update the user (it’s used for example when a user decides to become a member of the association);
    - View all the reservation made in a specific day;
    - View all the information about the Cultural Association member.
- Cultural Office area: this area is accessible only by a “cultural_office” role of user account and the functionalities are:
    - Create new organizer account;
    - View all the information about all the organizers.
- Librarian area: this area is accessible only by a “librarian” role of user account and the functionalities are:
    - Create new “librarian” role account;
    - View all the information about the seat bookings;
    - Create a new seat reservation for who is not registered in the web app.

Presentation Logic Layer

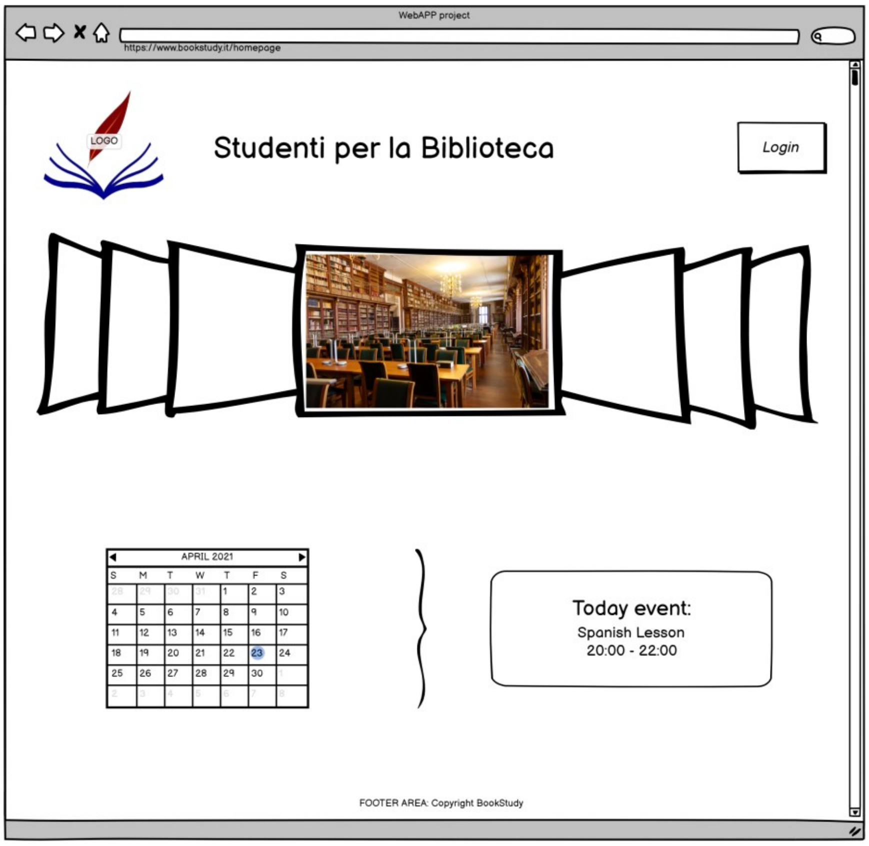
The following pages are an example of the pages available:

- Homepage: Contains information about the programmed events and a link to the login page and the registration page. Developed via jsp.
- Login page: Contains the form to login on the webapp, using the phone number and password of the user. Developed via jsp.
- Create User Page: Allows all the stakeholders of the web app to create a new user account, which has “user” role and it cannot be modified from this page. Developed via jsp.
- Create Librarian Page: Allows the “librarian” users’ role of the web app to create a new user account which can have “librarian” role. Developed via jsp.
- Create Organizer Page: Allows the “cultural_office” users’ role of the web app to create a new organizer account. Developed via jsp.
- Upgrade User Role Page: allows the “association_admin” users’ role to update the role of an existing user account from the role “user” to “association_member” or “association_admin” role. Developed via jsp.
- Insert Seat Reservation Page: Allows users to make a new seat reservation in the library. Developed via jsp.
- Search Reservation Page: Allows users to search for their own seat reservations and see information about. Developed via jsp.
- Update Reservation Page: allows users to register the entry and exit time by submitting the alphanumeric code of the reservation. The system will automatically register the time when the submit request is sent. Developed via jsp.
- Insert Conference Page: allows an organizer account to create a conference in a specific day. Developed via rest.

- Delete Conference Page: allows an organizer account to delete a conference. Developed via rest.
- Insert Conference Reservation Page: allows a user to book a reservation for a conference. Developed via jsp.

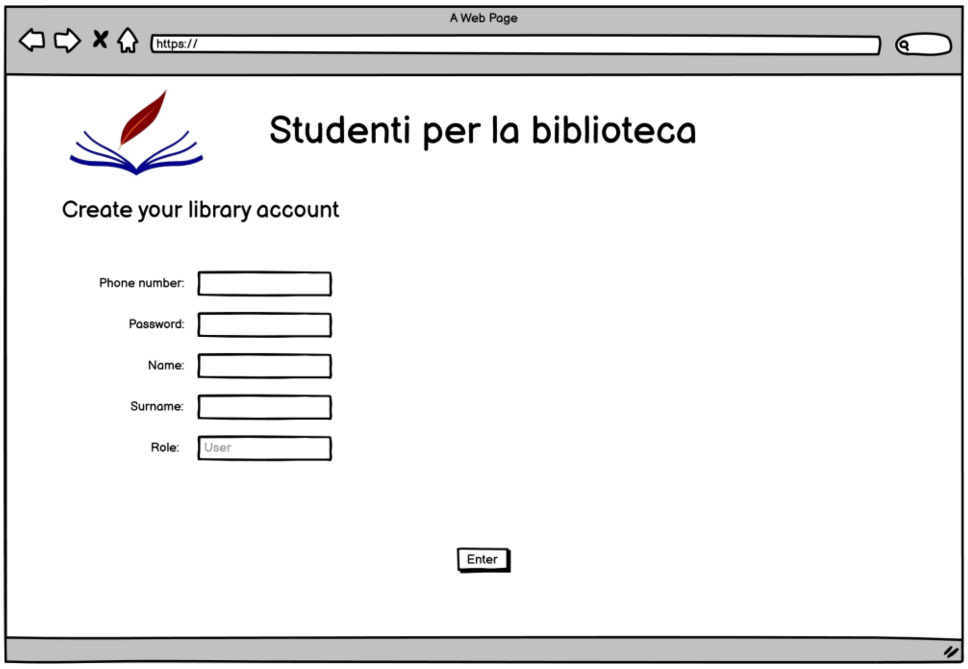
Home Page (Interface Mockup)

The homepage contains a graphic representation of the library's room. It also contains information about the actual programmed event and a link to the login page.
The homepage does not contain any other link because all the operations are supposed to be done after the login.


WebAPP project
https://www.bookstudy.it/homepage
LOGO
Studenti per la Biblioteca
Login
APRIL 2021
Today event:
Spanish Lesson
20:00 - 22:00
FOOTER AREA: Copyright BookStudy

Registration Page (Interface Mockup)

In case the user creates the account by himself, the following form will be displayed and the role is by default set to “user” (and it cannot be changed using this form).



A Web Page

https://

Studenti per la biblioteca

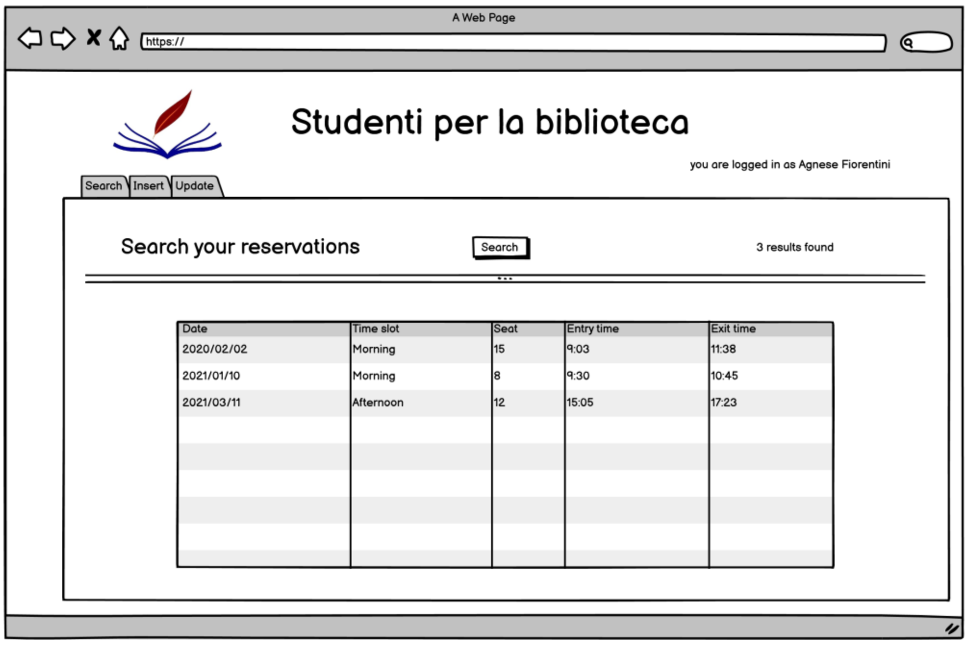
Create your library account

Phone number:	
Password:	
Name:	
Surname:	
Role:	User

Enter

Search Reservation Page (Interface Mockup)

The “Search Reservation Page” contains a form that show the users’ reservations. Note that a user can only see his own reservations.



A Web Page

https://

Studenti per la biblioteca

you are logged in as Agnese Fiorentini

Search | Insert | Update

Search your reservations

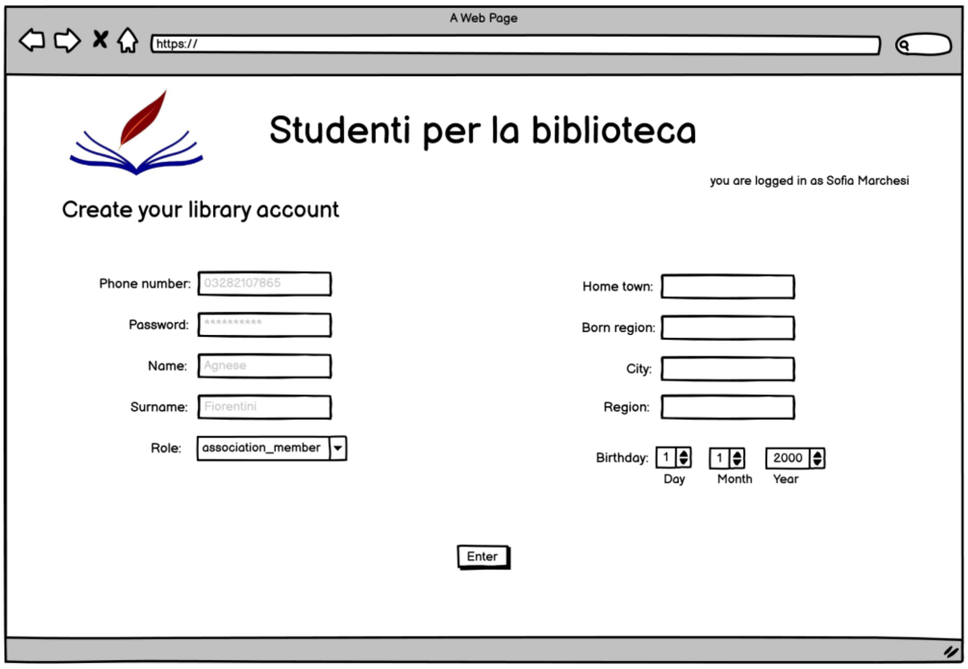
Search

3 results found

Date	Time slot	Seat	Entry time	Exit time
2020/02/02	Morning	15	9:03	11:38
2021/01/10	Morning	8	9:30	10:45
2021/03/11	Afternoon	12	15:05	17:23

Update Role Page (Interface Mockup)

The Update Role Page contains the form to update from “user” role of user account to a different role (“association_admin” role and “association_member” role). It can be done only by users account that are logged in with role “association_admins”, who will modify the role and add necessary additional information to the account.



Studenti per la biblioteca

you are logged in as Sofia Marchesi

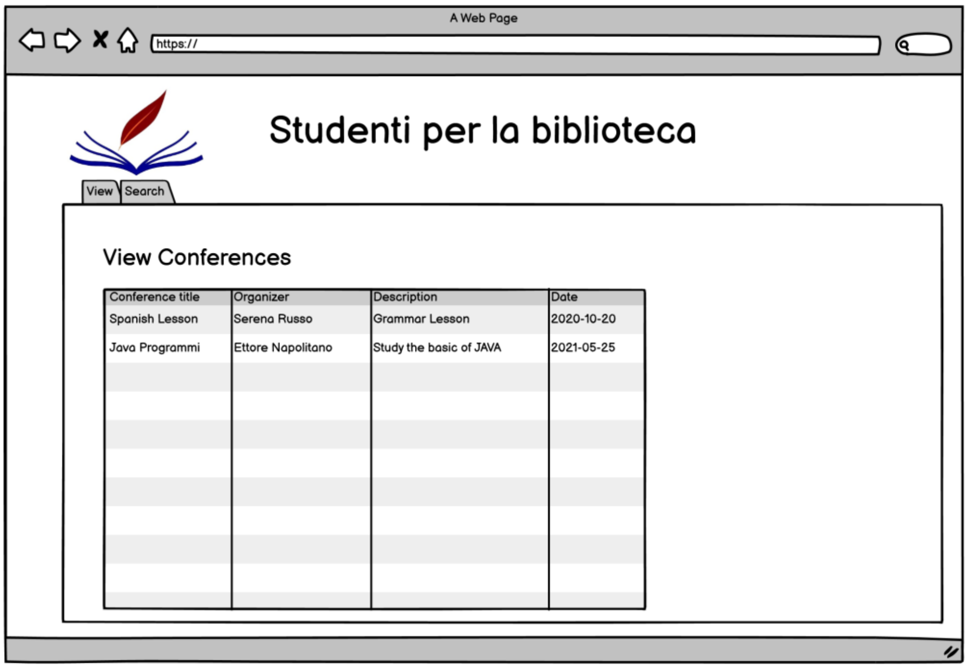
Create your library account

Phone number:	03282107865	Home town:	
Password:	**********	Born region:	
Name:	Agnese	City:	
Surname:	Fiorentini	Region:	
Role:	association_member	Birthday:	1 (Day) 1 (Month) 2000 (Year)

Enter

View Conference (Interface Mockup)

In this page can be found all the information about future conferences. This page can be accessed by also the unregistered users.



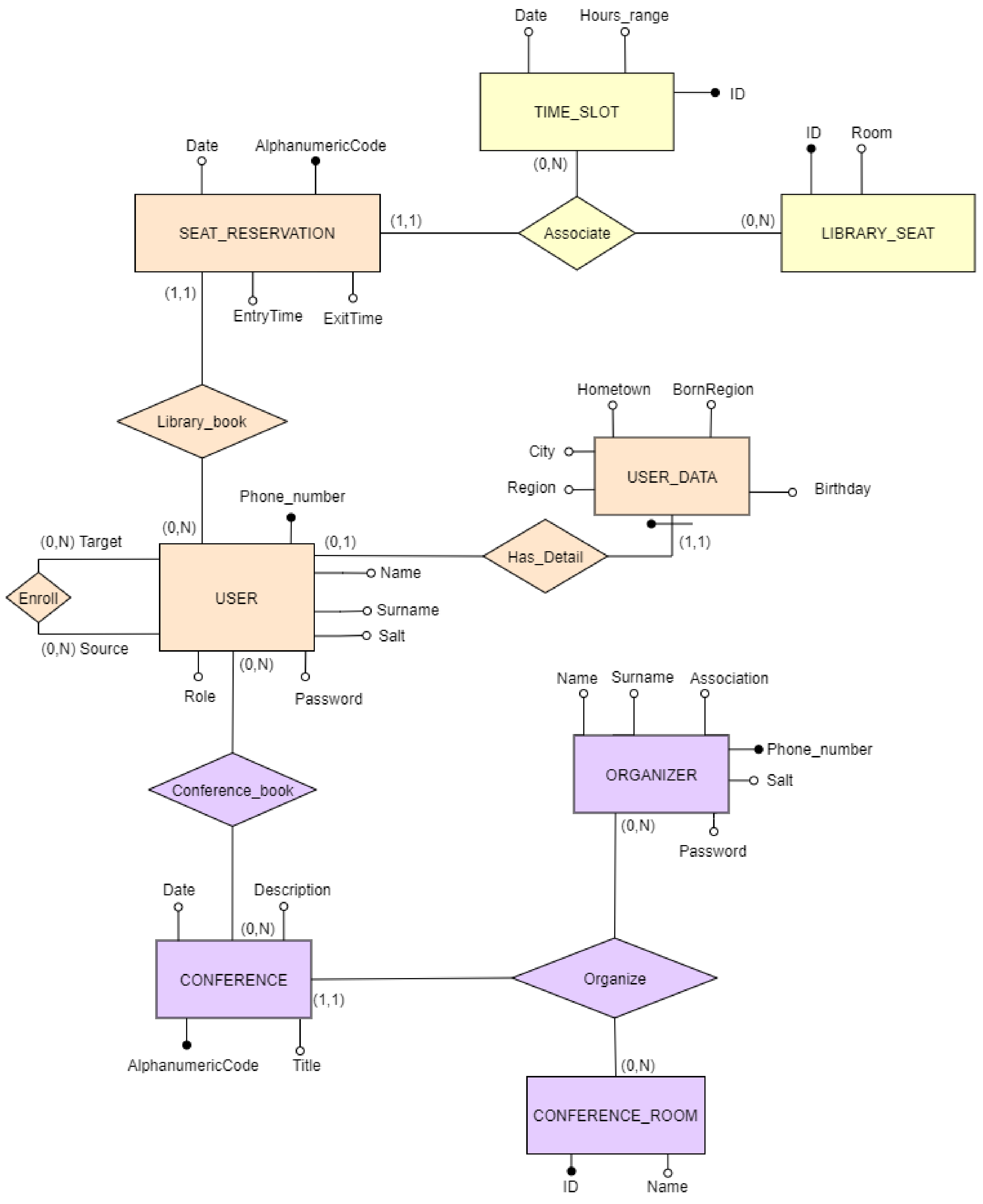
Studenti per la biblioteca

View Conferences

Conference title	Organizer	Description	Date
Spanish Lesson	Serena Russo	Grammar Lesson	2020-10-20
Java Programmi	Ettore Napolitano	Study the basic of JAVA	2021-05-25

Data Logic Layer

Entity-Relationship Schema



The Entity-Relationship schema contains 8 main entities:

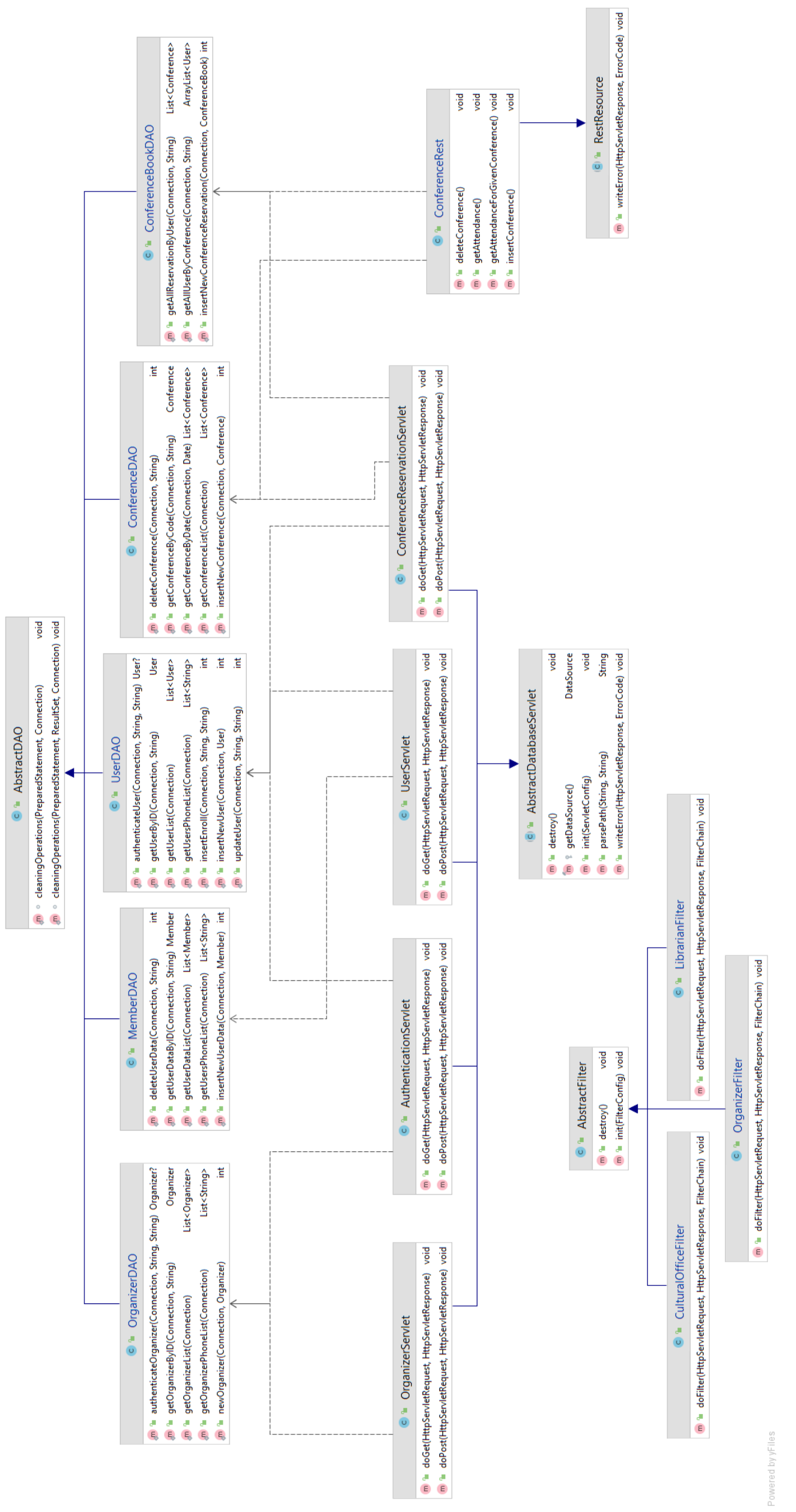
- **User**: contains the basic information about users. The primary key is the phone number, which is a CHAR field. The other attributes are name (CHAR), surname (CHAR), password (CHAR), salt (CHAR) and role (ENUM). Role types are "user", "librarian", "cultural office", "association admin" and "association member". The enroll relationship defines which user has enrolled another user into the database.
- **User_data**: contains other information about the user. The primary key is phone number from the user entity (external key), while the other attributes are the hometown (CHAR), born region (CHAR), birthday (DATE), city (CHAR) and region (CHAR).
- **Seat_reservation**: describe reservations made by users. They are uniquely identified by an alphanumeric code. Every reservation is also defined by the date, together with an entry and exit time.
- **Time_slot**: defines the slot of time in which the reservation is. The primary key is an INTEGER ID, while the other attributes define the date of the reservation and at which time slot correspond ("Morning", "Afternoon" or "Evening").
- **Library_seat**: defines the seat inside the library. Each seat is uniquely identified by an INTEGER ID, and the other attribute defines the room in which the seat is.
- **Conference**: describe conference reservations made by organizers. The primary key is an alphanumeric code, while the other attributes are the date, the title and a brief description of the conference argument.
- **Organizer**: conferences are made by organizers, who are able to book the conference room. As users of the system, organizers are identified by their phone number. The other attributes are name, surname, password, salt and the association they are part of.
- **Conference_room**: defines the conference rooms. The primary key is a CHAR id and the only other attribute is the name associated to the conference room.

Other Information

Conference_room, Time_slot and Library_Seat are the only entities that cannot be interacted with. It is possible to interact with all the other entities via interface (insert and update).

The salt attribute in the User and Organizer entities carries no information about the users since it is only used for password encryption.

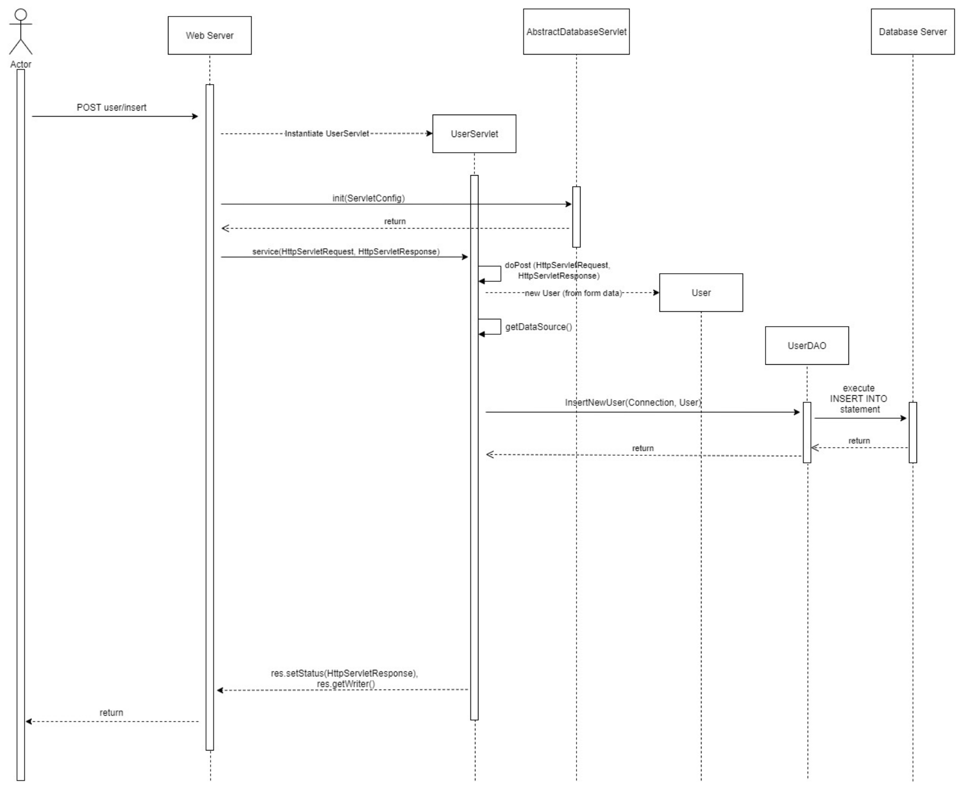
Business Logic Layer


AbstractDAO
cleaningOperations(PreparedStatement, Connection) void
cleaningOperations(PreparedStatement, ResultSet, Connection) void
ConferenceBookDAO
getAllReservationByUser(Connection, String) List<Conference>
getAllUserByConference(Connection, String) ArrayList<User>
insertNewConferenceReservation(Connection, ConferenceBook) int
ConferenceDAO
deleteConference(Connection, String) int
getConferenceByCode(Connection, String) Conference
getConferenceByDate(Connection, Date) List<Conference>
getConferenceList(Connection) List<Conference>
insertNewConference(Connection, Conference) int
UserDAO
authenticateUser(Connection, String, String) User?
getUserByID(Connection, String) User
getUserList(Connection) List<User>
getUsersPhoneList(Connection) List<String>
insertEnroll(Connection, String, String) int
insertNewUser(Connection, User) int
updateUser(Connection, String, String) int
MemberDAO
deleteUserData(Connection, String) int
getUserDataByID(Connection, String) Member
getUserDataList(Connection) List<Member>
getUsersPhoneList(Connection) List<String>
insertNewUserData(Connection, Member) int
OrganizerDAO
authenticateOrganizer(Connection, String, String) Organizer?
getOrganizerByID(Connection, String) Organizer
getOrganizerList(Connection) List<Organizer>
getOrganizerPhoneList(Connection) List<String>
newOrganizer(Connection, Organizer) int
ConferenceRest
deleteConference() void
getAttendance() void
getAttendanceForGivenConference() void
insertConference() void
RestResource
writeError(HttpServletResponse, ErrorCode) void
ConferenceReservationServlet
doGet(HttpServletRequest, HttpServletResponse) void
doPost(HttpServletRequest, HttpServletResponse) void
UserServlet
doGet(HttpServletRequest, HttpServletResponse) void
doPost(HttpServletRequest, HttpServletResponse) void
AuthenticationServlet
doGet(HttpServletRequest, HttpServletResponse) void
doPost(HttpServletRequest, HttpServletResponse) void
OrganizerServlet
doGet(HttpServletRequest, HttpServletResponse) void
doPost(HttpServletRequest, HttpServletResponse) void
AbstractDatabaseServlet
destroy() void
getDataSource() DataSource
init(ServletConfig) void
parsePath(String, String) String
writeError(HttpServletResponse, ErrorCode) void
AbstractFilter
destroy() void
init(FilterConfig) void
LibrarianFilter
doFilter(HttpServletRequest, HttpServletResponse, FilterChain) void
CulturalOfficeFilter
doFilter(HttpServletRequest, HttpServletResponse, FilterChain) void
OrganizerFilter
doFilter(HttpServletRequest, HttpServletResponse, FilterChain) void
Powered by yFiles

The class diagram contains (some of) the classes used to handle four types of resources: users, members, organizers and conferences. It is possible to observe that there is only one resource handled by only one Servlet, that is Member. All servlets implement the doGet and doPost methods, as sublcasses of the HttpServlet class (here omitted).
Regarding the Organizer, there are two Servlets used to handle the resource.
In particular, OrganizerServlet handles the registration of new organizers and allows to see a list of all the organizers, together with their information. The AuthenticationServlet handles the logins in the web application, in particular its doGet method handles the request for the logout, meanwhile the doPost method handles the login (or in other terms, the authentication) of the users in the application. Once the servlet has processed the request, it forwards it to the proper DAO class (Organizer or User). Concerning the Member, the only servlet that handle this resource is UserServlet. Such Servlet implements the doGet and doPost methods, that handle different operations. The doGet method, through a URI request, allows to retrieve a list of all the users with their information or the information of a single user given the phone number. The doPost method, on the other hand, manages the registration of a new user and the upgrade of a pre-existing user's role. Regarding the Conferences and the conference bookings, these resources are handled through a servlet and a REST class. The Servlet, called ConferenceReservationServlet, has a doGet method that allows to retrieve a list of reservations given a parameter (a date or a user phone number), and a doPost method to create a new reservation for a conference. The REST class, called ConferenceRest, manage the REST call about conference. This class contains four methods, used to delete and insert conferences and to retrieve a list of the attendances for all conferences or a specific one. ConferenceRest is a sub-class of RestResource and is invoked from RestDispatcherServlet.
To handle the interaction with the database, we have five Data Access Objects (DAO), one for each resource plus one to handle conference bookings: OrganizerDAO, MemberDAO, UserDAO, ConferenceDAO and ConferenceBookDAO. Each DAO implements the methods to retrieve, insert and update the different resources. ConferenceDAO is the only one that allows the delete operation.
Each resource is mapped into a specific data class, that has been omitted due to space reasons. Instances of such classes are instantiated by and passed between servlets and DAOs.

One important notation is that because of the project still in developing way all the date inserted in the form are not checked and so a user can also select a past date. We think to implement the necessary constraints in the front-end web application part.

Sequence Diagram


Actor
Web Server
AbstractDatabaseServlet
Database Server
POST user/insert
Instantiate UserServlet
UserServlet
init(ServletConfig)
return
service(HttpServletRequest, HttpServletResponse)
doPost (HttpServletRequest, HttpServletResponse)
new User (from form data)
User
getDataSource()
UserDAO
execute INSERT INTO statement
InsertNewUser(Connection, User)
return
return
res.setStatus(HttpServletResponse), res.getWriter()
return

Here is reported the sequence diagram for the operation about the insert of a simple user. The user executes a POST request to the web server. The web server instantiates the UserServlet, get the information about the datasource from the init method of the AbstractDatabaseServlet. After that it calls the doPost method of the UserServlet, passing the HttpServletRequest and the HTTP Servlet response. The UserServlet analyzes the request and recognizes that it is an insert operation, thus it creates an object User with the request parameters. At last, with the using of the insertNewUser method of the UserDAO class, the servlet inserts the User in the database. After that, the UserServlet replies to the web server with the methods setStatus and getWriter that notify if everything goes well or not.

REST API Summary

The list of the rest API for this homework are still "in progress" because we have to decide according to the frontend where is better to use a REST call or not.
Anyway, there are some resources that are just available with REST that are reported in the next table.

URI	Method	Description
conference/list	GET	Get the list of all conference for a given organizer (that is found with the session)
conference/view/{confID}	GET	Get the list of all users that are booked for the conference identified by the confID passed through the URL
conference/manage	POST	Create a new conference according with the data passed through json
Conference/manage/{confID}	DELETE	Delete a conference identified by the confID passed through the URL

REST API Error Codes

Here is the list of errors defined in the application.

Error Code	HTTP Status Code	Description
-100	WRONG_FORMAT	Wrong format of the request
-102	EMPTY_INPUT_FIELDS	One or more input fields are empty
-105	WRONG_CREDENTIALS	Submitted credentials are wrong
-109	UNRECOGNIZED_ROLE	Unrecognized role of a user
-113	WRONG_REST_FORMAT	Wrong rest request format
-116	CONFERENCE_NOT_FOUND	Conference not found
-117	USER_NOT_FOUND	User not found
-120	BADLY_FORMATTED_JSON	The input json is in the wrong format
-130	CONFERENCE_ALREADY_BOOKED	You are already booked for this conference
-200	OPERATION_UNKNOWN	Operation unknown
-500	METHOD_NOT_ALLOWED	The method is not allowed
-998	SQL_ERROR	SQL Exception
-999	INTERNAL_ERROR	Internal Error

REST API Details

We report here 3 different type of resources that our web applications handle.

User's reservations for a given conference

The following endpoint allows to get a list of all users who have reserved themselves for a given conference.

- **URL**

  /conference/view/{confID}

- **Method**

  GET

- **URL Params**

  Required:
  - confID= {string}
    The identifier of the conference

- **Data Params**

  No data params required.

- **Success Response**

  Code: 200
  Content:

```
{"resource-list":
[{"user":{"phoneNumber":"03336239657","name":"Valerio","surname":"Bianchi","role":"associa
tion_member"}
},
{"user":{"phoneNumber":"3920473377","name":"Gianmarco","surname":"Prando","role":"user"
}
}
]}
```

- Error Response

  Code: 302 REDIRECT
  When: The URI is not accessible because the user is not logged in, so it will be redirect to the
login page.

  Code: 401 UNAUTHORIZED
  When: The URI is accessible only to logged users with the role "organizer".

Code: 404 NOT FOUND
Content: {"error":{"code":-116,"message":"Conference not found."}}
When: The conference not exist

Code: 500 INTERNAL_SERVER_ERROR
Content: {“error”: {“code”: -999, “message” : “Internal Error.”}}
When: if there is a SQLException or a NamingException, this error is returned.

Insert new conference

The following endpoint allow an organizer to create a new conference.

- URL

  /conference/manage

- Method

  POST

- URL Params

  No URL params needed.

- Data Params

  The json representation of the conference.

- Success Response

  Code: 200
  Content:
  {“result”:”successful”}

- Error Response

  Code: 302 REDIRECT
  When: The URI is not accessible because the user is not logged in, so it will be redirect to the login page.

  Code: 400 BAD REQUEST
  Content: {“error”: {“code”: -102, “message” : “One or more input fields are empty.”}}
  When: if there are some input fields that are empty.

  Code: 401 UNAUTHORIZED
  When: The URI is accessible only to logged users with the role “organizer”.

Code: 500 INTERNAL_SERVER_ERROR
Content: {“error”: {“code”: -999, “message” : “Internal Error.”}}
When: if there is a SQLException or a NamingException, this error is returned.

Delete a conference

The following endpoint allows the organizer to delete one of his conference.

- URL

  /conference/manage/{confID}

- Method

  DELETE

- URL Params

  Required:
  - confID= {string}
    The identifier of the conference

- Data Params

  No data params required.

- Success Response

  Code: 200
  Content:
  {"result" : ”successfull”}

- Error Response

  Code: 302 REDIRECT
  When: The URI is not accessible because the user is not logged in, so it will be redirect to the login page.

  Code: 400 BAD REQUEST
  Content: {"error":{"code":-113,"message":”Wrong rest request format"}}
  When: We not provide a conference to delete

  Code: 401 UNAUTHORIZED
  When: The URI is accessible only to logged users with the role “organizer”.

  Code: 404 NOT FOUND
  Content: {"error":{"code":-116,"message":"Conference not found."}}
  When: The conference not exist

Code: 500 INTERNAL_SERVER_ERROR
Content: {“error”: {“code”: -999, “message” : “Internal Error.”}}
When: if there is a SQLException or a NamingException, this error is returned.