Web Architectures

`assignment 2`

# Chat system - Web Application

Sebastiano Chiari - 220527

`sebastiano.chiari@studenti.unitn.it`

October 17, 2021

## 1 Introduction

The aim of this assignment is to develop a web application that implements a chat system, using the MVC pattern.

## 2 Implementation

### 2.1 Presentation layer

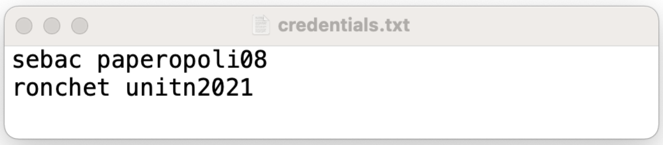
The **login page** (Figure 1) displays a title that remember the user to log in and a form in which the user has to submit username and password (which are required fields). Validation on client side is performed, so if the user doesn't input one of those field the form will not be submitted. Once both fields have been filled, the form submits a POST request to the `LoginServlet` (section 2.3.1) to perform authentication. If the login process fails, an invalid feedback will be displayed at the top of the page (Figure 6).

The **homepage** is displayed after the user completes the authentication process. It is composed of two sections:

- a list of available room names, which are clickable. All the room names are links that bring to the correspondent room page. If there are no rooms, this section will display a paragraph saying "Sorry, no rooms are available, but you can create one" (Figure 2). If rooms are available, this paragraph is substituted with "Enter in a room or create a new one" (Figure 7).
- a form which allows creating a new room, giving it a name (required field). The POST request is processes by the `IndexServlet` (section

The **room page** shows its name, a form that allows adding new messages in the room and all the messages in reversed chronological order (newest first). Every message is composed of the name of the author, the text and the timestamp of the message creation. The room page reloads itself automatically every 15 seconds. There are a button for manually reloading the page and another one ("Leave room") to return to the index page.

The **admin page** (Figure 9) can only be accessed by the admin user (thanks to the `AdminFilter`, section 2.4). This page shows the list of users with their credentials and allows adding new users through a form. New users are saved in the `credentials.txt` file and are persistent.



```
sebac paperopoli08
ronchet unitn2021
```

All pages, except the authentication one, show a banner with the username and a "Log out" button. This button invalidates the authentication and brings the user back to the login page. More, if the user is an admin, there will be also a button next to the "Logout" one, which allows the user to get to the admin page. The banner is a stand-alone component (which can be found in the `components` folder and it is included dynamically in each page through `<jsp:include page="..." />`.

In order to perform conditional controls and display multiple instances of the same collection, part of the JSTL library has been used: in particular, the core tags `<c:if>` and `<c:forEach>`, which do not violate the MVC pattern.

## 2.2 Classes

Three objects have been implemented to keep track of all complex structures within the web application. No collections (for example, a class containing the list of users) have been implemented as they are considered redundant. All classes have private fields, a public constructor and getter and setter methods, together with some specific public methods.

The `User` class represents a single user within the web application. It contains its username, its password and if the user is an admin. For loggin purposes, the method `toString()` was overrided.

The `Room` class represents a single room. It contains the room name, the username of the creator of the room and the list of messages within the room. A public method `addMessage(Message message)` has been implemented in order to add a single message in the room, while the `orderNewest()` method reorder the list of messages from the newest to the oldest based upon the messages timestamps.

The `Message` class represents a single message. It has the text of the message, the username of its own author and a timestamp when the message was created. More, the `getFormattedTime()` method returns the timestamp formatted in the following way `day-month-year hours:minutes` for visualization purposes.

## 2.3 Servlets

Servlets are Java programming language classes that are used to extend the capabilities of servers that host applications accessed by means of a request-response programming model.

### 2.3.1 LoginServlet

The `LoginServlet` manages the authentication process.

The `doGet()` method redirect the user to the login page.

The `doPost()` method gets the login parameters, username and password, and checks if the given username corresponds to any existing user: if so, the same check is performed over the password of the given username and if it matches the user is logged in, session is established and he gets redirected to the index page; otherwise, the user request is forwarded to the login page.

### 2.3.2 `LogoutServlet`

The `LogoutServlet` manages the de-authentication process.

The `doPost()` method retrieves the current session and invalidates it. Then, the user is redirected to the login page.

### 2.3.3 `IndexServlet`

The `IndexServlet` manages the home page, also called user page.

The `init()` function, invoked only the first time the servlet is called, creates the empty room collection and sets a servlet context attribute to render the correct message in the index page.

The `doGet()` forward the user to the `index.jsp` view.

The `doPost()` is invoked when a user creates a new room: it retrieves the parameters, `roomName` and `user`, create a new room with the given parameters and adds it to the rooms collection. Since this operation is editing an instance variable, this process is placed inside a synchronized block in order to prevent problems due to concurrent accesses. Then, the servlet context variables are updated and the user is redirected to the homepage.

### 2.3.4 `RoomServlet`

The `RoomServlet` manages the room page, where all the information about one room is displayed and the user can post messages.

The `doGet()` method is called when a specific room is requested by the user. The servlet retrieves the parameter `roomName` and finds the corresponding room object from the rooms collection stored in the servlet context. If no room is found, the servlet throws a 404 error and the user is redirected to the error page. On the other hand, if the servlet found a correspondence, the correct room is retrieved and this object is set as a request attribute: then, the user request is forwarded to the room page, which will display dynamically all the information.

The `doPost()` method is invoked when a user send the form for a new message on a given room. The servlet retrieves the request parameters, `roomID` and `corpus`, and build a new `Message` object. The message is added to the corresponding room and the list is reordered in order to display the latest messages first in a synchronized block, given the fact that we are going to edit a shared resource which can be edited concurrently otherwise. Then, the servlet context variable rooms is updated and the user is redirected to the same room page, with the updated messages.

### 2.3.5 `AdminServlet`

The `AdminServlet` deals with the admin section of this web application and the creation of new users.

The `doGet()` method forward the user to the `admin/users.jsp` page.

The `doPost()` method is invoked when the admin fills in the form for creating a new user. The servlet gets the parameter, `username` and `password`, and builds a new `User` object. Then, the new user is added to the list of current users and it is saved into the `credentials.txt` file in order to assure persistence. As in the `RoomServlet` case, all the operations that involve editing shared variables are placed inside a synchronized block, to prevent issues for concurrent accesses.

## 2.4 Filters

Filters dynamically intercept requests and responses to transform or use the information contained in the requests or responses. In this web application, two filters have been implemented: the `AuthenticationFilter` and the `AdminFilter`.

### 2.4.1 `AuthenticationFilter`

This filter handles the access to all pages of the web application: to access any page, users must authenticate themselves, while unauthenticated users must be directed to the login page.

In the `init` method, the filter imports all the users from the configuration file (in `.txt` format) and creates the admin user: all users are added to a list and this collection is set as a `ServletContext` attribute. The password of the admin user and the path to the configuration file are given to the filter as `initParams`, respectively `admin` and `usersFilePath` in the configuration of the filter.

In the `doFilter` method, this filter checks if a session exists and if the session has a `user` attribute (this means that the current user is logged in or has a previous session open). If the users is logged in and the request coincides with the login page, then, the user is redirected to the homepage. If the user is not logged in, the filter will redirect the user to the login page. Any other case will forward the user's request.

### 2.4.2 `AdminFilter`

This filter handles all the requests that belongs to the admin category user. In the `doFilter` method, the filter retrieves the session and checks the value of the `isAdmin` session attribute. If the user is an admin, then the request to the admin resource will be accepted, while on the other hand the user will be redirected to the homepage.

## 3 Comments and notes

Given the non existence of a database, all the process that involves creating resources, copying resources and storing them in the `ServletContext` (in order to make everything accessible for everybody) is very expensive on the computational side. More, in order to prevent issues due to concurrent accesses, the synchronized blocks reduce even more the optimization.

The `web.xml` has been edited in order to redirect all errors to a custom error page (`error.jsp`): this page will contain a link to return to the homepage or, if the user is not logged in already, to the login page (this is managed through the `AuthenticationFilter`).

All the servlet and filter configurations are not in the `web.xml` file but can be found in the headers of the corresponding Java classes: I think this approach is more clean and less chaotic.

Despite the suggestion to use a dedicated function to URL encoding, it seems to me (with different browsers, such as Firefox, Chrome and Safari) that spaces in URL do not give any trouble: browsers automatically convert the space character in the special one `%20` and vice versa, so no further encoding has to be performed.

### 3.1 Security constraints

Since the goal of this project was not to build a robust web application with regards to security, it was not performed a thorough implementation of security checks. The forms (especially the login one) call a POST request and the parameters are not passed in a visible way to the user. On the other hand, no encryption methods have been implemented to store safely passwords and to perform controls without exhibit the actual password. More, basic validation has been put client and server side but forms are definitely not safe against injection of malicious code.

Another control which was not implemented is the creation of duplicate resources, such as users or rooms. Each object is not associated to an ID (as it would be with a database), so its "primary key" is the name itself. Since iterating though all the objects in order to find duplicates would be very expensive and since it was not required, I skipped this check.

## 4 Results

In the following section, there are screenshots of the running application with descriptions that documents the various steps and scenarios.

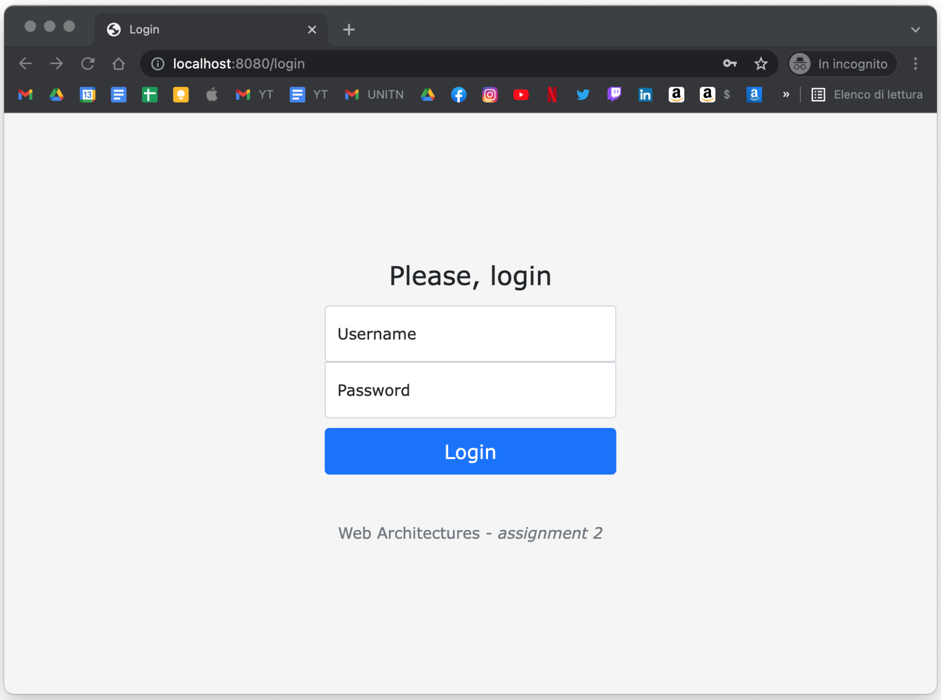
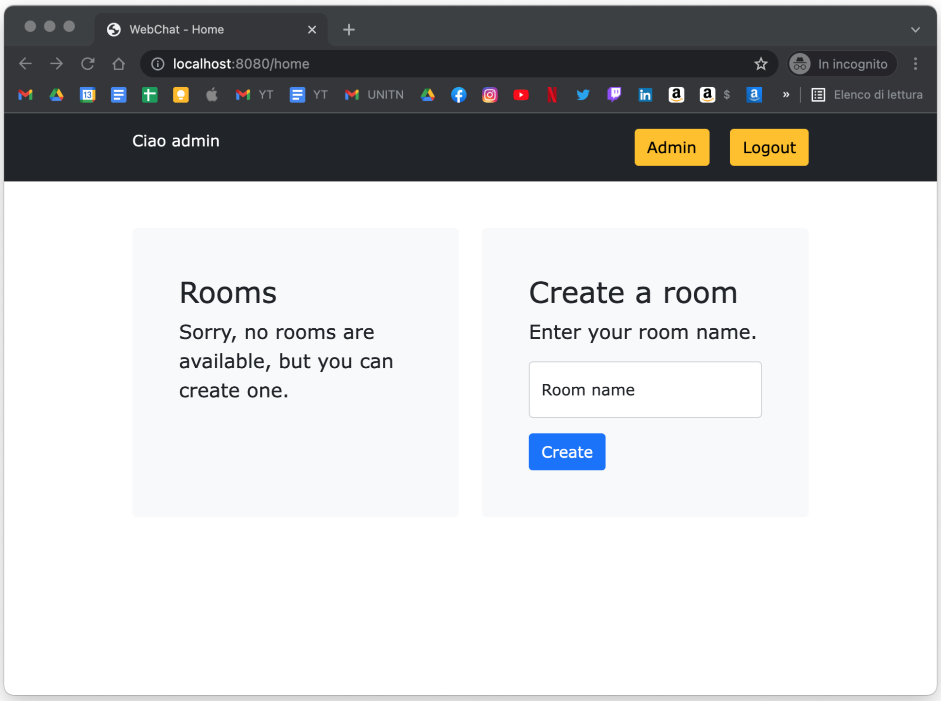


Figure 1: Landing page for a non authenticated user



Sorry, no rooms are available, but you can create one.

Enter your room name.

Figure 2: Homepage for the admin user, with no rooms created

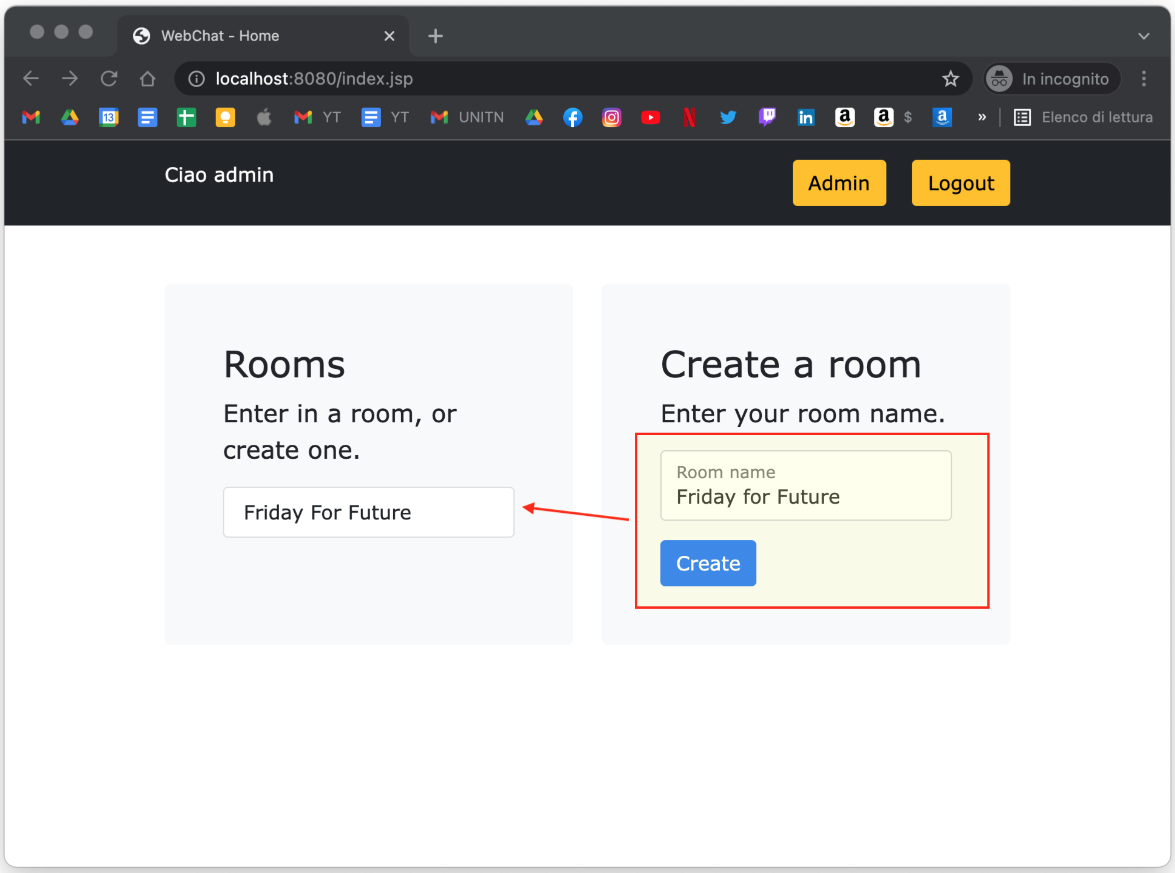


Figure 3: Page creation, two step process: fill the form and then the page will refresh displaying the updated list of available rooms

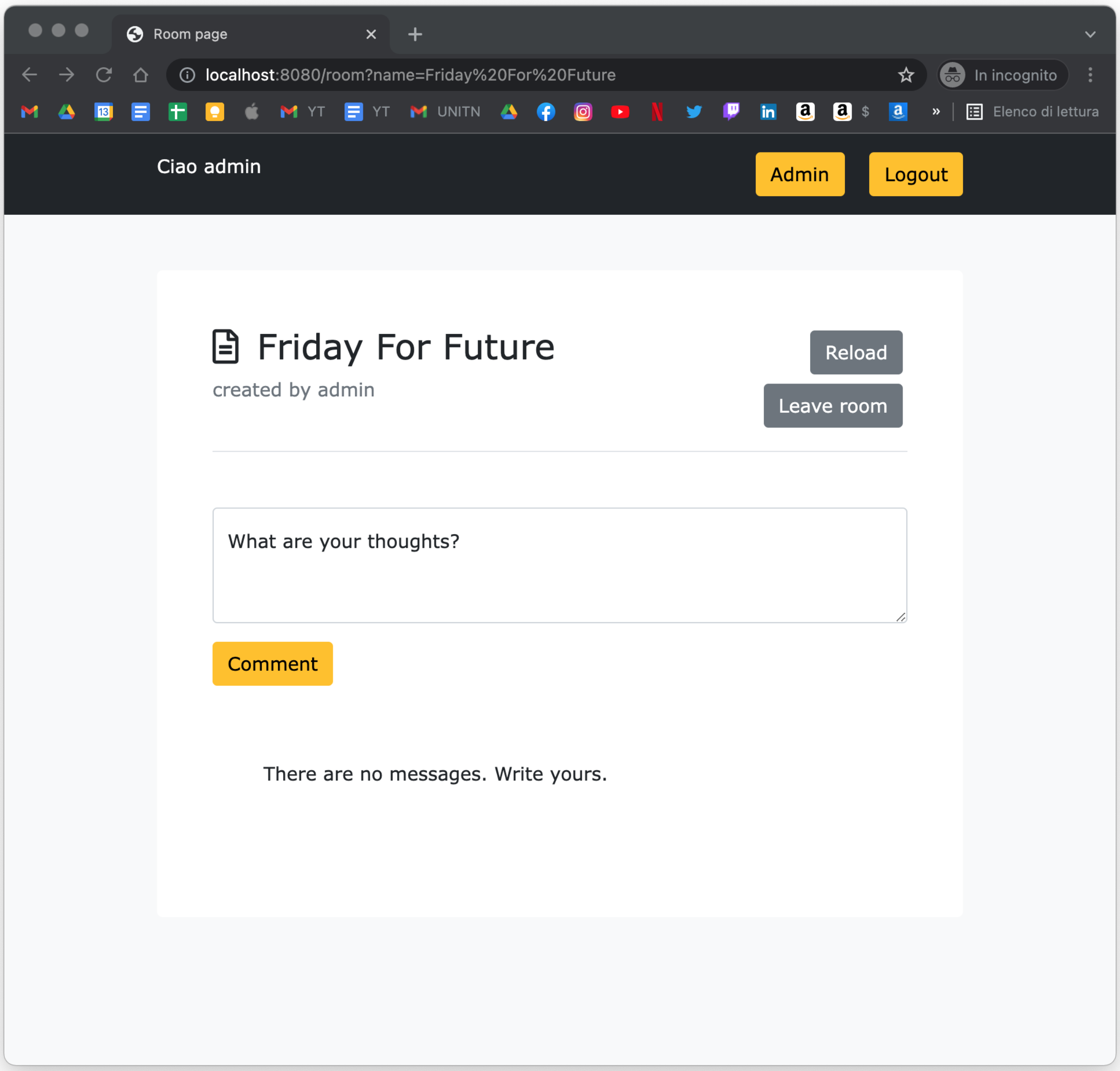
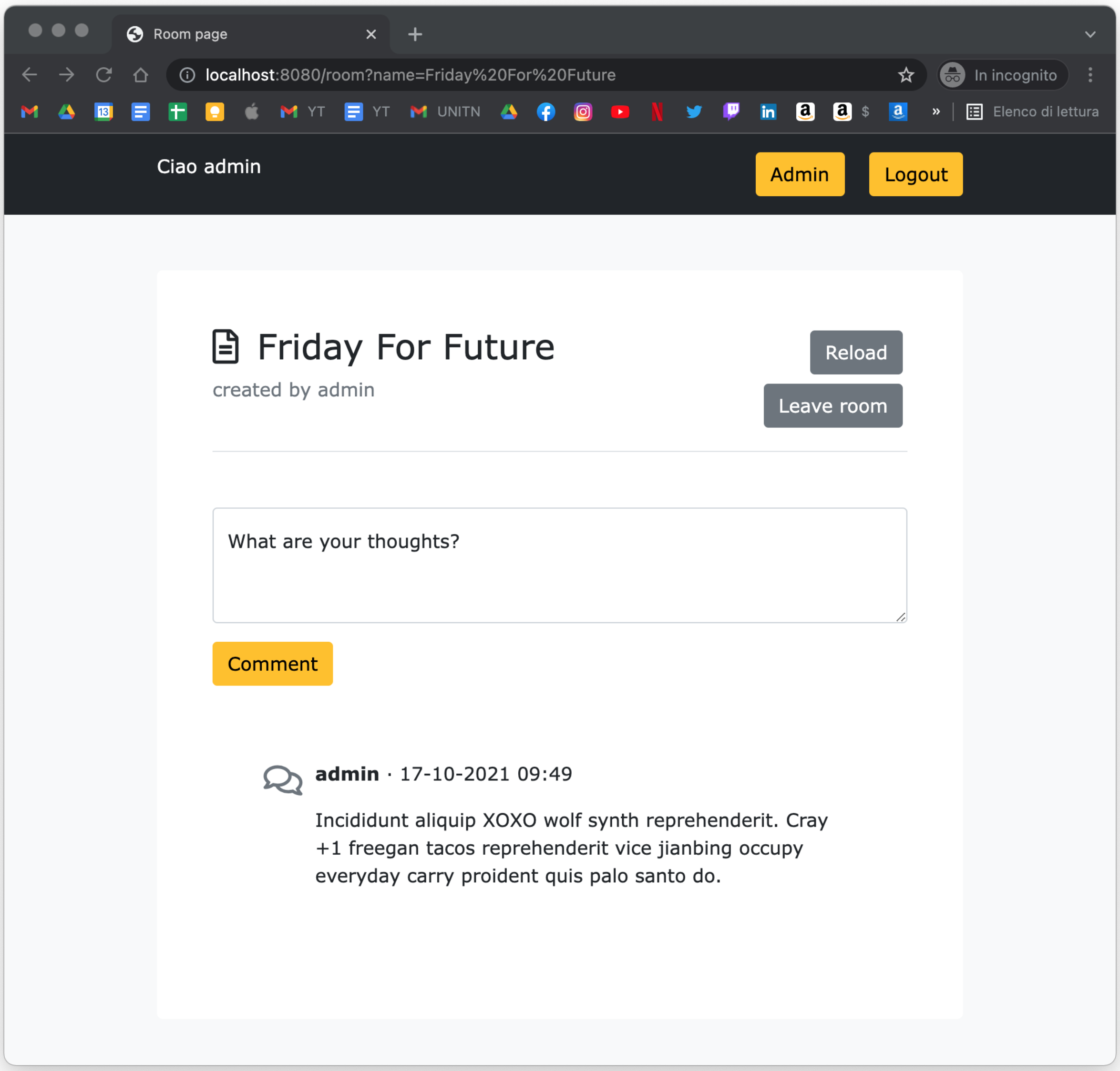


Figure 4: Empty room with no messages



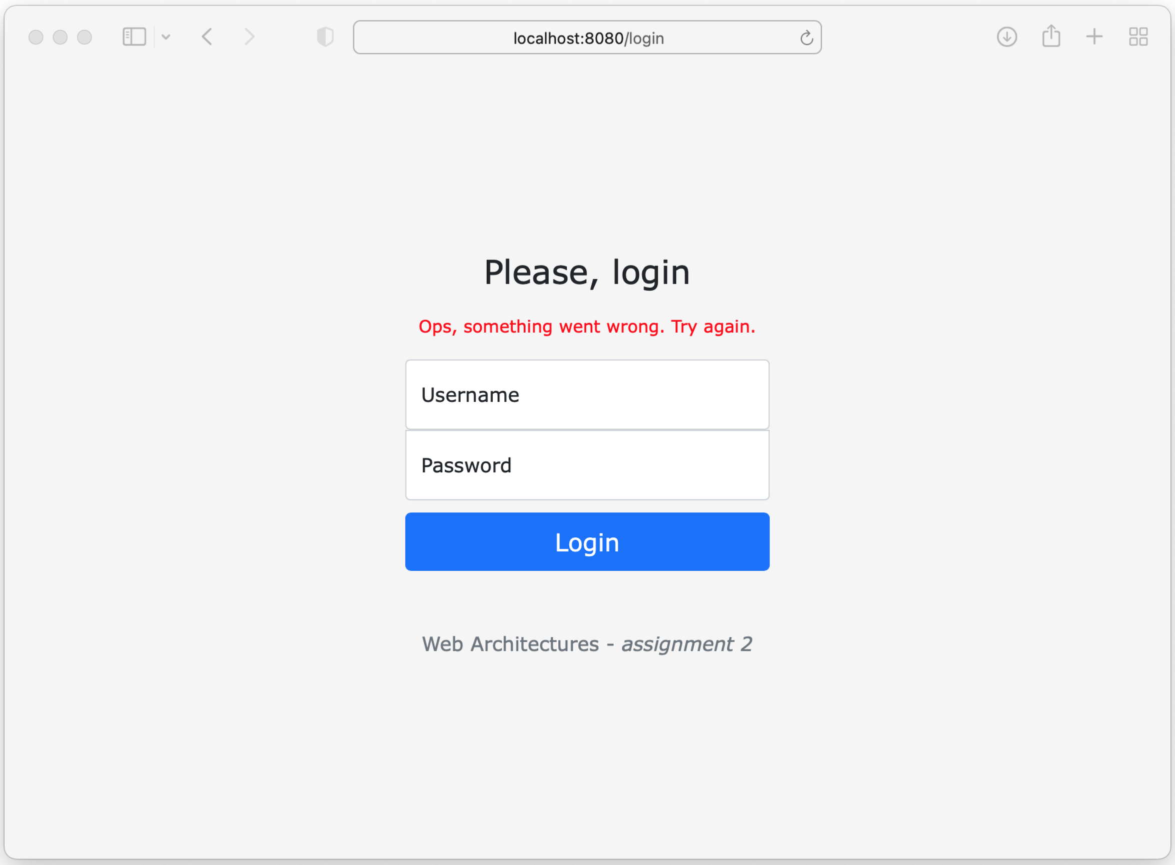
Friday For Future

created by admin

**admin** · 17-10-2021 09:49

Incididunt aliquip XOXO wolf synth reprehenderit. Cray +1 freegan tacos reprehenderit vice jianbing occupy everyday carry proident quis palo santo do.

Figure 5: Room with a posted message

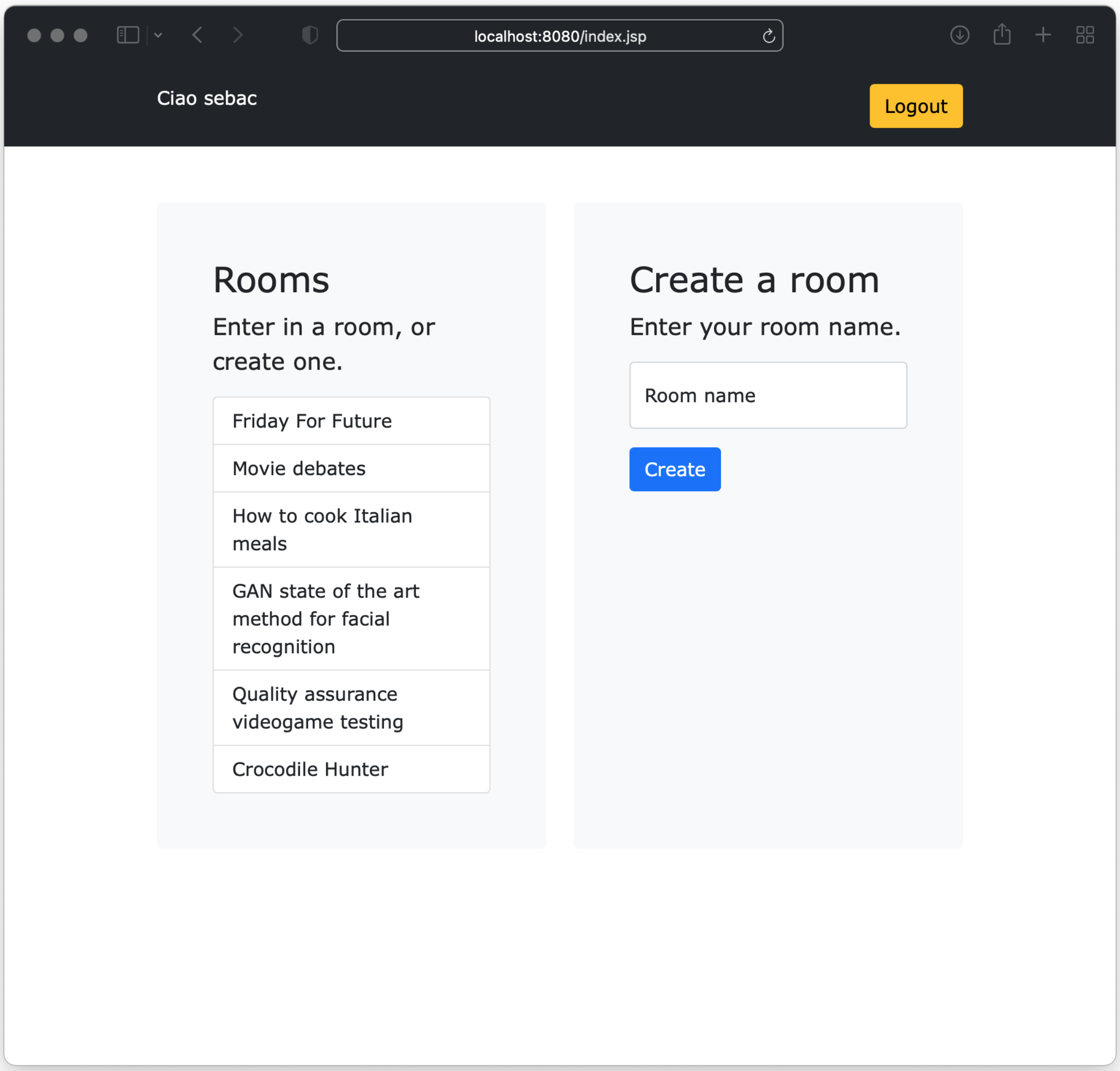

localhost:8080/login

Please, login

Ops, something went wrong. Try again.

Web Architectures - *assignment 2*

Figure 6: Other user (different browser) tries to authenticate with wrong credentials


localhost:8080/index.jsp

Ciao sebac

Logout

Rooms

Enter in a room, or create one.

- Friday For Future
- Movie debates
- How to cook Italian meals
- GAN state of the art method for facial recognition
- Quality assurance videogame testing
- Crocodile Hunter

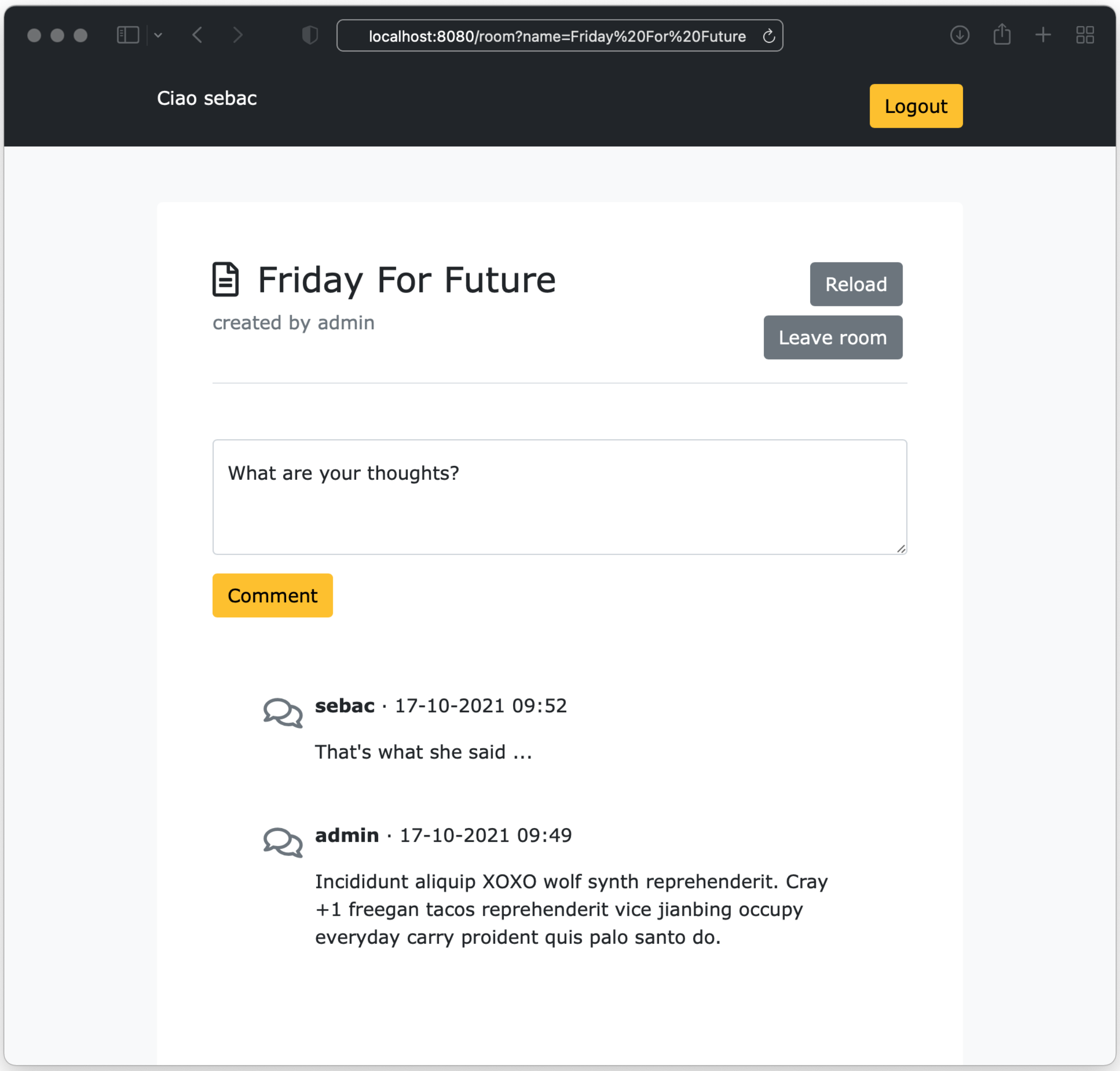
Create a room

Enter your room name.

Room name

Create

Figure 7: User homepage with lots of rooms



Friday For Future

created by admin

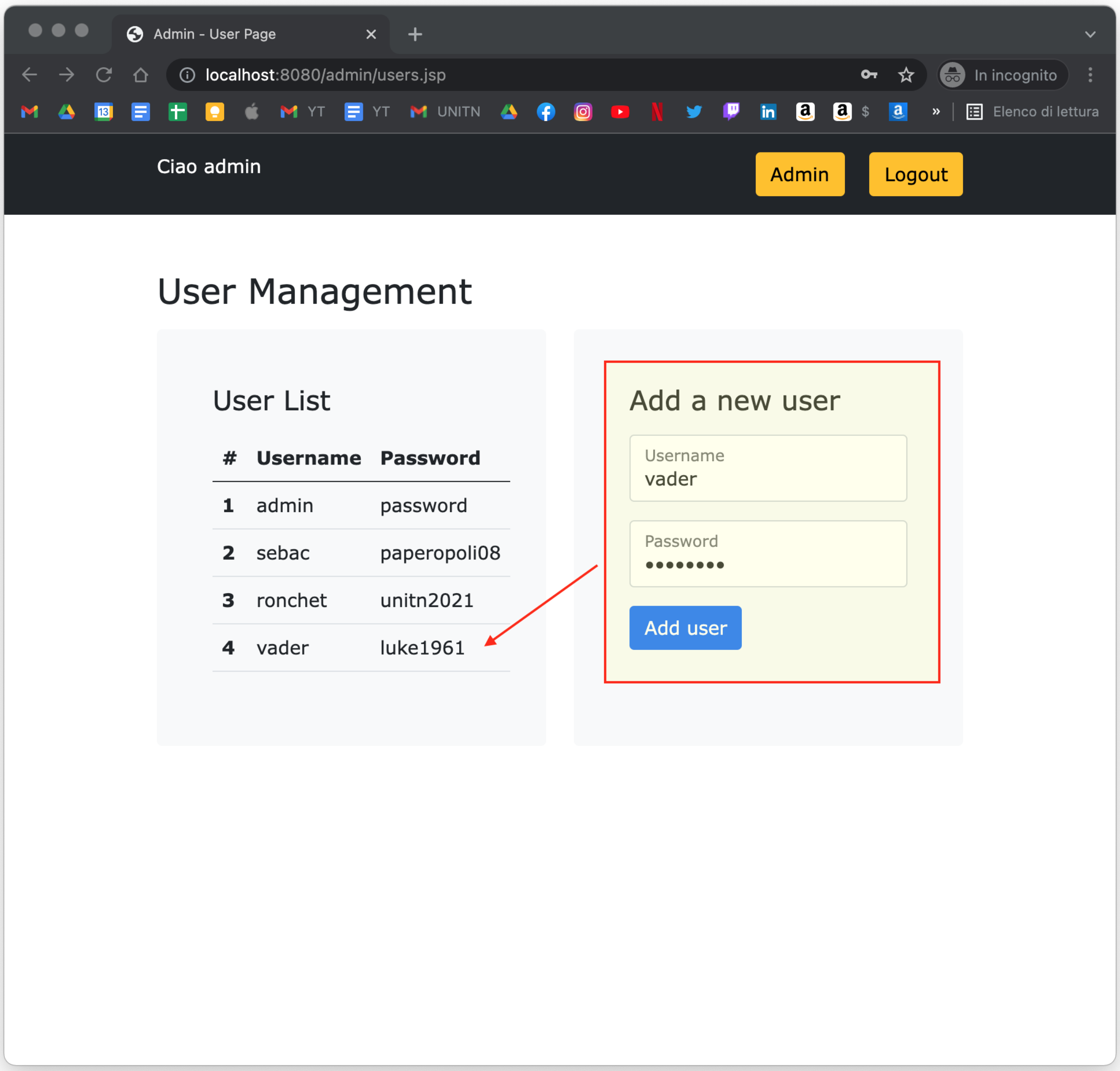
**sebac** · 17-10-2021 09:52

That's what she said ...

**admin** · 17-10-2021 09:49

Incididunt aliquip XOXO wolf synth reprehenderit. Cray +1 freegan tacos reprehenderit vice jianbing occupy everyday carry proident quis palo santo do.

Figure 8: User posts a message on the same room as before



Admin - User Page

localhost:8080/admin/users.jsp

User Management

User List

| # | Username | Password |
|---|---|---|
| 1 | admin | password |
| 2 | sebac | paperopoli08 |
| 3 | ronchet | unitn2021 |
| 4 | vader | luke1961 |

Add a new user

Username: vader

Password: ••••••••

Figure 9: Admin section and new user creation