**Sample Chat Application**

**Project file :**

Project was generated in **start.spring.io** web page, in this I have created a new Spring boot Maven project and the language was java.

The Spring Boot version was 3.3.5 which was choosen by default. And I have given respected Group Id , Artifact Id , Name , Description and Package name and I choose the package as jar and java version 17.

Later I have added the dependencies, which are:

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

* Allows you to interact with the database using high-level Java objects instead of writing complex SQL queries.
* For example, you can save, retrieve, update, and delete records using repository interfaces.

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-thymeleaf</artifactId>

</dependency>

* Allows the generation of dynamic web pages by embedding server-side data into HTML templates.
* Provides an easy way to bind backend data (using the Model) to frontend HTML components using Thymeleaf syntax. Thymeleaf templates are valid HTML and can be opened in any browser or editor.

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

* Enables handling HTTP requests and responses for building web applications. Includes an embedded web server (Tomcat, by default) to run your application without needing external deployment.
* Provides tools to create RESTful web services using @RestController, @GetMapping, etc. Includes Spring MVC for handling web requests and responses effectively.

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>

<scope>runtime</scope>

<optional>true</optional>

</dependency>

* Detects changes in source code or resources and automatically restarts the application, reducing the need for manual restarts during development.
* Refreshes the browser automatically whenever changes are made to static resources (like HTML, CSS, or JavaScript).

**When to Use It?**

* During the **development phase** of your project to speed up coding, testing, and debugging.
* It is **not used in production** as it is specifically designed for development workflows.

<dependency>

<groupId>com.mysql</groupId>

<artifactId>mysql-connector-j</artifactId>

<scope>runtime</scope>

</dependency>

* Provides the necessary driver to establish a connection between your Java application and a MySQL database.
* The runtime scope ensures the driver is included at runtime but not during the compile phase, optimizing the build process.
* Works seamlessly with Spring Data JPA and JDBC to execute database operations (CRUD, queries, etc.).

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

* Combines multiple testing libraries in one starter for ease of use.

**Key Components Included:**

* **JUnit 5**: Framework for running tests.
* **Spring Test**: Utilities for testing Spring components.
* **Mockito**: Mocking library.
* **AssertJ**: Fluent assertion library.
* **Hamcrest**: Matchers for assertions.

After adding all the dependencies we need to Generate the project it will start downloading in to your local file.

Then open Eclipse which you are using goto **File>click on Import>click on Existing Maven Project>Browse your project in Root directory>Finish.**

**Configuration Phase:**

In **src/main/resources/application.properties** we need to add configuration:

#spring.application.name=simple\_chat\_application

spring.datasource.url=jdbc:mysql://localhost:3306/chatapp

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.datasource.username=root

spring.datasource.password=root

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect

# Enable Thymeleaf template caching for development

spring.thymeleaf.cache=false

spring.thymeleaf.prefix=classpath:/templates/

spring.thymeleaf.suffix=.html

spring.thymeleaf.mode=HTML

spring.aop.auto=true

spring.aop.proxy-target-class=true

logging.level.org.thymeleaf=DEBUG

logging.level.org.springframework=DEBUG

logging.level.org.hibernate=DEBUG

spring.mvc.view.prefix=/WEB-INF/views/

spring.mvc.view.suffix=.jsp

spring.thymeleaf.encoding=UTF-8

server.port=9090

**Welcome page :**

In the welcome page we have displayed a welcome message along with a SignIn button and a Login Button which was directly take us to that particular pages.

**HTML/CSS Code:**

<!DOCTYPE html>

<html lang=*"en"*>

<head>

<meta charset=*"UTF-8"*>

<meta name=*"viewport"* content=*"width=device-width, initial-scale=1.0"*>

<title>Welcome Page</title>

<!-- <link rel="stylesheet" href="welcome1.css"> -->

<style>

/\* General Reset \*/

\* {

margin: *0*;

padding: *0*;

box-sizing: *border-box*;

}

**body** {

font-family: *Arial, sans-serif*;

height: *100vh*;

background: *linear-gradient(135deg, #2a0845, #6441a5)*;

overflow: *hidden*; /\* Prevent scroll due to animation \*/

position: *relative*;

display: *flex*;

justify-content: *center*;

align-items: *center*;

}

/\* Welcome Page Container \*/

*.welcome-container* {

display: *flex*;

justify-content: *center*;

align-items: *center*;

height: *100%*;

width: *100%*;

position: *relative*;

z-index: *1*;

}

*.welcome-box* {

background: *#2a0845*;

border-radius: *15px*;

box-shadow: *0px 5px 15px rgba(0, 0, 0, 0.3)*;

padding: *40px 60px*;

width: *500px*;

text-align: *center*;

color: *white*;

position: *relative*;

}

**h1** {

font-size: *36px*;

margin-bottom: *15px*;

}

**p** {

font-size: *18px*;

margin-bottom: *30px*;

color: *#d1c4e9*;

}

*.button-group* {

display: *flex*;

justify-content: *center*;

gap: *20px*;

}

*.btn* {

display: *inline-block*;

padding: *12px 25px*;

border: *none*;

border-radius: *25px*;

text-decoration: *none*;

font-size: *16px*;

font-weight: *bold*;

color: *white*;

background: *linear-gradient(135deg, #a02fc2, #d62d92)*;

cursor: *pointer*;

transition: *0.3s ease*;

}

*.btn:hover* {

background: *linear-gradient(135deg, #d62d92, #a02fc2)*;

transform: *scale(1.05)*;

}

/\* Bubble Animations \*/

**body***::before* {

content: *""*;

position: *absolute*;

top: *0*;

left: *0*;

width: *100%*;

height: *100%*;

z-index: *0*;

overflow: *hidden*;

}

*.bubble* {

position: *absolute*;

bottom: *-100px*;

width: *40px*;

height: *40px*;

background: *rgba(255, 255, 255, 0.2)*;

border-radius: *50%*;

animation: *bubbleMove 10s infinite ease-in-out*;

animation-delay: *calc(-1s* \* *var(--i))*;

}

@**keyframes** **bubbleMove** {

0% {

transform: *translateY(0) translateX(0)*;

opacity: *0.6*;

}

*50*% {

transform: *translateY(-500px) translateX(50px)*;

opacity: *1*;

}

*100*% {

transform: *translateY(-1000px) translateX(-50px)*;

opacity: *0*;

}

}

/\* Random Bubbles \*/

**body** *.bubble:nth-child*(*1*) {

--i: *0*;

left: *20%*;

width: *60px*;

height: *60px*;

}

**body** *.bubble:nth-child*(*2*) {

--i: *2*;

left: *40%*;

width: *50px*;

height: *50px*;

}

**body** *.bubble:nth-child*(*3*) {

--i: *4*;

left: *60%*;

width: *70px*;

height: *70px*;

}

**body** *.bubble:nth-child*(*4*) {

--i: *6*;

left: *80%*;

width: *40px*;

height: *40px*;

}

**body** *.bubble:nth-child*(*5*) {

--i: *8*;

left: *10%*;

width: *55px*;

height: *55px*;

}

</style>

</head>

<body>

<div class=*"welcome-container"*>

<div class=*"welcome-box"*>

<h1>Welcome!</h1>

<p>We’re excited to have you here. Let’s get started!</p>

<div class=*"button-group"*>

<a href=*"registration.html"* target=*"self"* class=*"btn"*>Sign In</a>

<a href=*"login.html"* target=*"self"* class=*"btn"*>Log In</a>

</div>

</div>

</div>

<!-- Add Bubble Divs -->

<div class=*"bubble"*></div>

<div class=*"bubble"*></div>

<div class=*"bubble"*></div>

<div class=*"bubble"*></div>

<div class=*"bubble"*></div>

</body>

</html>

**Code:**

I have created four packages which are model, repository , service and controller packages.

In model package I have create only blueprints on the objects , and we create an **interface** in the **Repository** package to define data access operations, and we extend it from JpaRepository (or other Spring Data interfaces) to leverage built-in methods for interacting with the database.

**public** **interface** RegistrationRepository **extends** JpaRepository<Registration, Long>

}

Then in Service package create an **class** use @service annotation on top of class declaration, The @Service annotation is typically used to mark service classes that contain business logic. These classes are meant to handle operations that are typically not related directly to database access (which is handled by repositories) but instead to complex business operations, calculations, or integrations.

@Service

**public** **class** MessageService {

@Autowired

**private** MessageRepository messageRepository;

}

In a **Spring MVC controller**, we define methods that handle incoming HTTP requests, process them, and return appropriate responses (e.g., HTML pages, JSON data, etc.). Controllers act as the intermediary between the client (browser or API consumer) and the service layer of your application.

Controllers contain methods annotated with @GetMapping, @PostMapping, @PutMapping, @DeleteMapping, or @RequestMapping to handle specific HTTP requests (GET, POST, PUT, DELETE).

A controller usually calls methods from the service layer (e.g., @Service annotated classes) to handle business logic. Controllers often use the Model or ModelAndView to pass data to views (for web applications) or return data as part of the response (for REST APIs).

@Controller

**public** **class** ChatController {

@Autowired

**private** MessageService messageService;

@GetMapping("/welcome")

**public** String getWelcomePage() {

**return** "welcome";

}

}

* @GetMapping is a Spring MVC annotation used to map HTTP **GET** requests to a specific handler method in a controller. It is a shortcut for @RequestMapping(method = RequestMethod.GET).
* It handles **GET** requests (usually for retrieving data or displaying pages) and routes them to the corresponding method in the controller.
* ("/welcome") this will called as url and it indicates that when we get request it will display particular url html page from template as a response.

**Registration Page:**

In the registration page we were storing user details like name, email, password and confirm password and register, if the user already have an account then they can directly login from welcome page or else they can also login from registration itself.

**HTML/CSS Code:**

<!DOCTYPE html>

<html lang=*"en"*>

<head>

<meta charset=*"UTF-8"*>

<meta name=*"viewport"* content=*"width=device-width, initial-scale=1.0"*>

<title>Registration Page</title>

<style>

/\* General Reset \*/

\* {

margin: *0*;

padding: *0*;

box-sizing: *border-box*;

}

**body** {

font-family: *Arial, sans-serif*;

height: *100vh*;

background: *linear-gradient(135deg, #2a0845, #6441a5)*;

overflow: *hidden*; /\* Prevent scroll due to animation \*/

position: *relative*;

display: *flex*;

justify-content: *center*;

align-items: *center*;

}

/\* Registration Form Container \*/

*.registration-container* {

display: *flex*;

justify-content: *center*;

align-items: *center*;

height: *100%*;

width: *100%*;

position: *relative*;

z-index: *1*;

}

*.registration-box* {

background: *#2a0845*;

border-radius: *15px*;

box-shadow: *0px 5px 15px rgba(0, 0, 0, 0.3)*;

padding: *30px 40px*;

width: *400px*;

color: *white*;

text-align: *center*;

position: *relative*;

}

**h2** {

margin-bottom: *20px*;

}

*.input-field* {

margin: *15px 0*;

position: *relative*;

}

*.input-field* *.icon* {

position: *absolute*;

left: *10px*;

top: *50%*;

transform: *translateY(-50%)*;

color: *#ffffff*;

}

*.input-field* **input** {

width: *100%*;

padding: *10px 40px*;

border: *none*;

border-radius: *25px*;

outline: *none*;

background: *#4b0c6a*;

color: *white*;

}

*.input-field* **input***::placeholder* {

color: *#ffffffb3*;

}

*.register-btn* {

width: *100%*;

padding: *10px*;

border: *none*;

border-radius: *25px*;

background: *linear-gradient(135deg, #a02fc2, #d62d92)*;

color: *white*;

cursor: *pointer*;

transition: *0.3s ease*;

}

*.register-btn:hover* {

background: *linear-gradient(135deg, #d62d92, #a02fc2)*;

}

*.options* {

margin-top: *15px*;

}

*.options* **a** {

color: *#d62d92*;

margin: *0 10px*;

text-decoration: *none*;

font-size: *14px*;

}

*.options* **a***:hover* {

text-decoration: *underline*;

}

/\* Background Bubble Animations \*/

**body***::before* {

content: *""*;

position: *absolute*;

top: *0*;

left: *0*;

width: *100%*;

height: *100%*;

z-index: *0*;

overflow: *hidden*;

}

/\* Bubble Style \*/

*.bubble* {

position: *absolute*;

bottom: *-100px*;

width: *40px*;

height: *40px*;

background: *rgba(255, 255, 255, 0.2)*;

border-radius: *50%*;

animation: *bubbleMove 10s infinite ease-in-out*;

animation-delay: *calc(-1s* \* *var(--i))*;

}

/\* Keyframe for Bubble Movement \*/

@**keyframes** **bubbleMove** {

0% {

transform: *translateY(0) translateX(0)*;

opacity: *0.6*;

}

*50*% {

transform: *translateY(-500px) translateX(50px)*;

opacity: *1*;

}

*100*% {

transform: *translateY(-1000px) translateX(-50px)*;

opacity: *0*;

}

}

/\* Random Bubbles \*/

**body** *.bubble:nth-child*(*1*) {

--i: *0*;

left: *20%*;

width: *60px*;

height: *60px*;

}

**body** *.bubble:nth-child*(*2*) {

--i: *2*;

left: *40%*;

width: *50px*;

height: *50px*;

}

**body** *.bubble:nth-child*(*3*) {

--i: *4*;

left: *60%*;

width: *70px*;

height: *70px*;

}

**body** *.bubble:nth-child*(*4*) {

--i: *6*;

left: *80%*;

width: *40px*;

height: *40px*;

}

**body** *.bubble:nth-child*(*5*) {

--i: *8*;

left: *10%*;

width: *55px*;

height: *55px*;

}

</style>

</head>

<body>

<div class=*"registration-container"*>

<div class=*"registration-box"*>

<h2>Register</h2>

<form action=*"/registration"* method=*"post"* th:object=*"${registration}"*>

<div class=*"input-field"*>

<span class=*"icon"*>👤</span>

<input type=*"text"* placeholder=*"Full Name"* th:field=*"\*{name}"* required>

</div>

<div class=*"input-field"*>

<span class=*"icon"*>📧</span>

<input type=*"email"* placeholder=*"Email"* th:field=*"\*{email}"* required>

</div>

<div class=*"input-field"*>

<span class=*"icon"*>🔒</span>

<input type=*"password"* placeholder=*"Password"* th:field=*"\*{password}"* required>

</div>

<div class=*"input-field"*>

<span class=*"icon"*>🔒</span>

<input type=*"password"* placeholder=*"Confirm Password"* th:field=*"\*{confirmPassword}"* required>

</div>

<button type=*"submit"* class=*"register-btn"*>Register</button>

</form>

<div class=*"options"*>

<a href=*"/login"*>Already have an account? Log In</a>

</div>

</div>

</div>

<!-- Add Bubble Divs -->

<div class=*"bubble"*></div>

<div class=*"bubble"*></div>

<div class=*"bubble"*></div>

<div class=*"bubble"*></div>

<div class=*"bubble"*></div>

</body>

</html>

**Code:**

For Registration I have created a model/DTO class with the help of that class we can able create table in the database and fetch data from webpage to database and store the data.

package com.chatapplication.chatapp.model;

import jakarta.persistence.Entity;

import jakarta.persistence.GeneratedValue;

import jakarta.persistence.GenerationType;

import jakarta.persistence.Id;

import jakarta.persistence.NamedQueries;

import jakarta.persistence.NamedQuery;

import jakarta.persistence.Table;

@Entity

@Table(name = "registration")

public class Registration {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

private String email;

private String password;

private String confirmPassword;

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getEmail() {

return email;

}

public void setEmail(String email) {

this.email = email;

}

public String getPassword() {

return password;

}

public void setPassword(String password) {

this.password = password;

}

public String getConfirmPassword() {

return confirmPassword;

}

public void setConfirmPassword(String confirmPassword) {

this.confirmPassword = confirmPassword;

}

}

I have used the same RegistrationRepository interface for the registration purpose without any changes. And in the service class :

**public** **boolean** isNotExist(Registration newRegistration) {

**return** registrationRepository.findByEmail(newRegistration.getEmail().toLowerCase()).isEmpty();

}

**public** **boolean** saveRegistration(Registration registration) {

**try** {

registrationRepository.save(registration);

**return** **true**;

} **catch** (DataIntegrityViolationException e) {

**return** **false**; // Duplicate entry or other constraint violations

}

}

In the controller class :

@GetMapping("/registration")

**public** String showRegistrationForm(Model model) {

model.addAttribute("registration", **new** Registration());

**return** "registration"; // This corresponds to registration.html in templates

}

// To handle form submission and store data in the database

@PostMapping("/registration")

**public** String registerUser(@Validated Registration registration,BindingResult bindingResult,Model model) {

**if**(bindingResult.hasErrors()) {

**return** "registration";

}

**if**(!registration.getPassword().equals(registration.getConfirmPassword())) {

model.addAttribute("error","Password do not match");

**return** "registration";

}

**if** (!messageService.isNotExist(registration)) {

model.addAttribute("error", "A user with this email already exists!");

**return** "registrationErrorPage";

}

**boolean** isUserCreated=messageService.saveRegistration(registration);

**if**(isUserCreated) {// if(isUserCreated && isNotExist) {

model.addAttribute("message", "Registration successfull!Please log in.");

**return** "redirect:/login";

}**else** {

model.addAttribute("error", "Error in Registration!Please try again.");

**return** "registrationErrorPage";

}

}

If the registration was not done properly due to user gmail already exist or password incorrect or any reasons at that time the error page will be occurred.

**HTML/CSS Code:**

<!DOCTYPE html>

<html lang=*"en"*>

<head>

<meta charset=*"UTF-8"*>

<meta name=*"viewport"* content=*"width=device-width, initial-scale=1.0"*>

<title>Welcome Page</title>

<style>

/\* General Reset \*/

\* {

margin: *0*;

padding: *0*;

box-sizing: *border-box*;

}

**body** {

font-family: *Arial, sans-serif*;

height: *100vh*;

background: *linear-gradient(135deg, #2a0845, #6441a5)*;

overflow: *hidden*; /\* Prevent scroll due to animation \*/

position: *relative*;

display: *flex*;

justify-content: *center*;

align-items: *center*;

}

/\* Welcome Page Container \*/

*.welcome-container* {

display: *flex*;

justify-content: *center*;

align-items: *center*;

height: *100%*;

width: *100%*;

position: *relative*;

z-index: *1*;

}

*.welcome-box* {

background: *#2a0845*;

border-radius: *15px*;

box-shadow: *0px 5px 15px rgba(0, 0, 0, 0.3)*;

padding: *40px 60px*;

width: *500px*;

text-align: *center*;

color: *white*;

position: *relative*;

}

**h1** {

font-size: *36px*;

margin-bottom: *15px*;

}

**p** {

font-size: *18px*;

margin-bottom: *30px*;

color: *#d1c4e9*;

}

*.button-group* {

display: *flex*;

justify-content: *center*;

gap: *20px*;

}

*.btn* {

display: *inline-block*;

padding: *12px 25px*;

border: *none*;

border-radius: *25px*;

text-decoration: *none*;

font-size: *16px*;

font-weight: *bold*;

color: *white*;

background: *linear-gradient(135deg, #a02fc2, #d62d92)*;

cursor: *pointer*;

transition: *0.3s ease*;

}

*.btn:hover* {

background: *linear-gradient(135deg, #d62d92, #a02fc2)*;

transform: *scale(1.05)*;

}

/\* Bubble Animations \*/

**body***::before* {

content: *""*;

position: *absolute*;

top: *0*;

left: *0*;

width: *100%*;

height: *100%*;

z-index: *0*;

overflow: *hidden*;

}

*.bubble* {

position: *absolute*;

bottom: *-100px*;

width: *40px*;

height: *40px*;

background: *rgba(255, 255, 255, 0.2)*;

border-radius: *50%*;

animation: *bubbleMove 10s infinite ease-in-out*;

animation-delay: *calc(-1s* \* *var(--i))*;

}

@**keyframes** **bubbleMove** {

0% {

transform: *translateY(0) translateX(0)*;

opacity: *0.6*;

}

*50*% {

transform: *translateY(-500px) translateX(50px)*;

opacity: *1*;

}

*100*% {

transform: *translateY(-1000px) translateX(-50px)*;

opacity: *0*;

}

}

/\* Random Bubbles \*/

**body** *.bubble:nth-child*(*1*) {

--i: *0*;

left: *20%*;

width: *60px*;

height: *60px*;

}

**body** *.bubble:nth-child*(*2*) {

--i: *2*;

left: *40%*;

width: *50px*;

height: *50px*;

}

**body** *.bubble:nth-child*(*3*) {

--i: *4*;

left: *60%*;

width: *70px*;

height: *70px*;

}

**body** *.bubble:nth-child*(*4*) {

--i: *6*;

left: *80%*;

width: *40px*;

height: *40px*;

}

**body** *.bubble:nth-child*(*5*) {

--i: *8*;

left: *10%*;

width: *55px*;

height: *55px*;

}

</style>

</head>

<body>

<div class=*"welcome-container"*>

<form class=*"welcome-box"* action=*"/registrationErrorPage"*>

<h1>Error!</h1>

<p>Above Email address was already exist try with another email or do login!</p>

<div class=*"button-group"*>

<a href=*"/registration"* class=*"btn"*>Sign In</a>

<a href=*"/login"* class=*"btn"*>Log In</a>

</div>

</form>

</div>

<!-- Add Bubble Divs -->

<div class=*"bubble"*></div>

<div class=*"bubble"*></div>

<div class=*"bubble"*></div>

<div class=*"bubble"*></div>

<div class=*"bubble"*></div>

</body>

</html>

**Login Page:**

In the login page we are doing validation weather the user already existing or not, comparing the email and password with data present in the database.

**HTML/CSS Code:**

<!DOCTYPE html>

<html xmlns:th=*"http://www.thymeleaf.org"*>

<head>

<meta charset=*"UTF-8"*>

<meta name=*"viewport"* content=*"width=device-width, initial-scale=1.0"*>

<title>Login Page</title>

<style>

\* {

margin: *0*;

padding: *0*;

box-sizing: *border-box*;

}

**body** {

font-family: *Arial, sans-serif*;

height: *100vh*;

background: *linear-gradient(135deg, #2a0845, #6441a5)*;

animation: *gradientAnimation 6s infinite alternate*;

display: *flex*;

justify-content: *center*;

align-items: *center*;

overflow: *hidden*;

}

/\* Animated Gradient Background \*/

@**keyframes** **gradientAnimation** {

0% {

background: *linear-gradient(135deg, #2a0845, #6441a5)*;

}

*50*% {

background: *linear-gradient(135deg, #3c096c, #7f00ff)*;

}

*100*% {

background: *linear-gradient(135deg, #4b0082, #ff0099)*;

}

}

*.login-container* {

display: *flex*;

justify-content: *center*;

align-items: *center*;

height: *100%*;

width: *100%*;

position: *relative*;

}

*.login-box* {

background: *#2a0845*;

border-radius: *15px*;

box-shadow: *0px 5px 15px rgba(0, 0, 0, 0.3)*;

padding: *30px 40px*;

width: *400px*;

color: *white*;

text-align: *center*;

position: *relative*;

}

**h2** {

margin-bottom: *20px*;

}

*.input-field* {

margin: *15px 0*;

position: *relative*;}

*.input-field* *.icon* {

position: *absolute*;

left: *10px*;

top: *50%*;

transform: *translateY(-50%)*;

color: *#ffffff*;

}

*.input-field* **input** {

width: *100%*;

padding: *10px 40px*;

border: *none*;

border-radius: *25px*;

outline: *none*;

background: *#4b0c6a*;

color: *white*;

}

*.input-field* **input***::placeholder* {

color: *#ffffffb3*;

}

/\* Animated Button Hover \*/

*.login-btn* {

width: *100%*;

padding: *10px*;

border: *none*;

border-radius: *25px*;

background: *linear-gradient(135deg, #a02fc2, #d62d92)*;

color: *white*;

cursor: *pointer*;

transition: *transform 0.3s ease, background 0.3s ease*;

}

*.login-btn:hover* {

background: *linear-gradient(135deg, #d62d92, #a02fc2)*;

transform: *scale(1.1)*;

}

*.options* {

margin-top: *15px*;

}

*.options* **a** {

color: *#d62d92*;

margin: *0 10px*;

text-decoration: *none*;

font-size: *14px*;

}

*.options* **a***:hover* {

text-decoration: *underline*;

}

/\* Floating Background Circles Animation \*/

*.login-container::before***,**

*.login-container::after* {

content: *""*;

position: *absolute*;

border-radius: *50%*;

z-index: *-1*;

}

*.login-container::before* {

width: *300px*;

height: *300px*;

background: *#5e0e6b*;

top: *-50px*;

left: *-50px*;

animation: *float 8s ease-in-out infinite alternate*;

}

*.login-container::after* {

width: *400px*;

height: *400px*;

background: *#4e007e*;

bottom: *-80px*;

right: *-100px*;

animation: *float 10s ease-in-out infinite alternate-reverse*;

}

/\* Keyframe for Floating Effect \*/

@**keyframes** **float** {

0% {

transform: *translate(0, 0)*;

}

*50*% {

transform: *translate(20px, -20px)*;

}

*100*% {

transform: *translate(-20px, 20px)*;

}

}

</style>

</head>

<body>

<div class=*"login-container"*>

<div class=*"login-box"*>

<h2>Login</h2>

<form action=*"/login"* method=*"post"* th:object=*"${login}"*>

<div class=*"input-field"*>

<span class=*"icon"*>📧</span>

<input type=*"email"* placeholder=*"Email"* th:field=*"\*{email}"* required>

</div>

<div class=*"input-field"*>

<span class=*"icon"*>🔒</span>

<input type=*"password"* placeholder=*"Password"* th:field=*"\*{password}"* required>

</div>

<button type=*"submit"* class=*"login-btn"*>Log In</button>

</form>

<div class=*"options"*>

<a href=*"/login"*>Login</a>

<a href=*"/registration"*>Sign Up</a>

</div>

</div>

</div>

</body>

</html>

In the Repository class we need to override some methods are:

Optional<Registration> findByEmail(String email);

@Query("SELECT r FROM Registration r WHERE r.email = :email AND r.password = :password")

Optional<Registration> findByEmailAndPassword(@Param("email") String email,@Param("password") String password);

In the Service class :

**public** **boolean** loginDetails(String email, String password) {

Optional<Registration> result = registrationRepository.findByEmailAndPassword(email,

password);

**return** result.isPresent();

}

In the Controller class:

@GetMapping("/login")

**public** String showLoginForm(Model model) {

model.addAttribute("login", **new** Login()); // Bind the form to the Login object

**return** "login"; // Return the Thymeleaf template name

}

@PostMapping("/login")

**public** String processLogin(@ModelAttribute("login") Login login , BindingResult result) {

**if** (result.hasErrors()) {

**return** "login"; // If validation fails, return to the form view

}

String email = login.getEmail();

String password = login.getPassword();

**boolean** isValidUser = messageService.loginDetails(email, password);

**if** (isValidUser) {

**return** "redirect:/chat";// Redirect to a dashboard or appropriate page

}**else** {

**return** "loginErrorPage";

}

}

If the validation went wrong with incorrect email id or password then it shows login error page :

**HTML/CSS Code:**

<!DOCTYPE html>

<html lang=*"en"*>

<head>

<meta charset=*"UTF-8"*>

<meta name=*"viewport"* content=*"width=device-width, initial-scale=1.0"*>

<title>Welcome Page</title>

<style>

/\* General Reset \*/

\* {

margin: *0*;

padding: *0*;

box-sizing: *border-box*;

}

**body** {

font-family: *Arial, sans-serif*;

height: *100vh*;

background: *linear-gradient(135deg, #2a0845, #6441a5)*;

overflow: *hidden*; /\* Prevent scroll due to animation \*/

position: *relative*;

display: *flex*;

justify-content: *center*;

align-items: *center*;

}

/\* Welcome Page Container \*/

*.welcome-container* {

display: *flex*;

justify-content: *center*;

align-items: *center*;

height: *100%*;

width: *100%*;

position: *relative*;

z-index: *1*;

}

*.welcome-box* {

background: *#2a0845*;

border-radius: *15px*;

box-shadow: *0px 5px 15px rgba(0, 0, 0, 0.3)*;

padding: *40px 60px*;

width: *500px*;

text-align: *center*;

color: *white*;

position: *relative*;

}

**h1** {

font-size: *36px*;

margin-bottom: *15px*;

}

**p** {

font-size: *18px*;

margin-bottom: *30px*;

color: *#d1c4e9*;

}

*.button-group* {

display: *flex*;

justify-content: *center*;

gap: *20px*;

}

*.btn* {

display: *inline-block*;

padding: *12px 25px*;

border: *none*;

border-radius: *25px*;

text-decoration: *none*;

font-size: *16px*;

font-weight: *bold*;

color: *white*;

background: *linear-gradient(135deg, #a02fc2, #d62d92)*;

cursor: *pointer*;

transition: *0.3s ease*;

}

*.btn:hover* {

background: *linear-gradient(135deg, #d62d92, #a02fc2)*;

transform: *scale(1.05)*;

}

/\* Bubble Animations \*/

**body***::before* {

content: *""*;

position: *absolute*;

top: *0*;

left: *0*;

width: *100%*;

height: *100%*;

z-index: *0*;

overflow: *hidden*;

}

*.bubble* {

position: *absolute*;

bottom: *-100px*;

width: *40px*;

height: *40px*;

background: *rgba(255, 255, 255, 0.2)*;

border-radius: *50%*;

animation: *bubbleMove 10s infinite ease-in-out*;

animation-delay: *calc(-1s* \* *var(--i))*;

}

@**keyframes** **bubbleMove** {

0% {

transform: *translateY(0) translateX(0)*;

opacity: *0.6*;

}

*50*% {

transform: *translateY(-500px) translateX(50px)*;

opacity: *1*;

}

*100*% {

transform: *translateY(-1000px) translateX(-50px)*;

opacity: *0*;

}

}

/\* Random Bubbles \*/

**body** *.bubble:nth-child*(*1*) {

--i: *0*;

left: *20%*;

width: *60px*;

height: *60px*;

}

**body** *.bubble:nth-child*(*2*) {

--i: *2*;

left: *40%*;

width: *50px*;

height: *50px*;

}

**body** *.bubble:nth-child*(*3*) {

--i: *4*;

left: *60%*;

width: *70px*;

height: *70px*;

}

**body** *.bubble:nth-child*(*4*) {

--i: *6*;

left: *80%*;

width: *40px*;

height: *40px*;

}

**body** *.bubble:nth-child*(*5*) {

--i: *8*;

left: *10%*;

width: *55px*;

height: *55px*;

}

</style>

</head>

<body>

<div class=*"welcome-container"*>

<form class=*"welcome-box"* action=*"/registrationErrorPage"*>

<h1>Error!</h1>

<p>Incorrect Email/Password!Please try again</p>

<div class=*"button-group"*>

<a href=*"/registration"* class=*"btn"*>Sign In</a>

<a href=*"/login"* class=*"btn"*>Log In</a>

</div>

</form>

</div>

<!-- Add Bubble Divs -->

<div class=*"bubble"*></div>

<div class=*"bubble"*></div>

<div class=*"bubble"*></div>

<div class=*"bubble"*></div>

<div class=*"bubble"*></div>

</body>

</html>

**Chat Page:**

When the user details was matched with the registered details then he chat page will be accessed and in the chat page it was mainly focusing on providing two way communication with multiple users with the help of WebSocket API this was mainly used to provide to way communication.

**WebSocket API:**

The WebSocket API enables a full-duplex communication channel over a single, long-lived connection between a client (e.g., a browser) and a server. Unlike HTTP, it allows for real-time, two-way data exchange without repeatedly establishing connections.

**Uses:**

* **Real-time updates**: For applications like chat apps, live sports updates, or stock price tracking.
* **Interactive tools**: Multiplayer games or collaborative editing tools.
* **IoT communication**: For devices needing constant data exchange.

**Features:**

1. **Full-duplex communication**: Simultaneous sending and receiving of data.
2. **Low latency**: Reduces overhead by avoiding frequent handshakes.
3. **Efficient**: Maintains a single connection for ongoing data exchange.
4. **Cross-platform**: Works in browsers, mobile apps, and servers.
5. **Protocol support**: Works on ws:// or wss:// (secure WebSocket) protocols.

**HTML/CSS/JS/WS Code:**

<!DOCTYPE html>

<html lang=*"en"*>

<head>

<meta charset=*"UTF-8"*>

<meta name=*"viewport"* content=*"width=device-width, initial-scale=1.0"*>

<title>Chat Application</title>

<style>

/\* General Reset \*/

\* {

margin: *0*;

padding: *0*;

box-sizing: *border-box*;

}

**body** {

font-family: *Arial, sans-serif*;

height: *100vh*;

background: *linear-gradient(135deg, #2a0845, #6441a5)*;

display: *flex*;

justify-content: *center*;

align-items: *center*;

color: *white*;

overflow: *hidden*;

}

/\* Chat Page Container \*/

*.chat-container* {

background: *#2a0845*;

border-radius: *15px*;

box-shadow: *0px 5px 15px rgba(0, 0, 0, 0.3)*;

padding: *30px*;

width: *500px*;

display: *flex*;

flex-direction: *column*;

align-items: *center*;

}

**h1** {

font-size: *28px*;

margin-bottom: *15px*;

text-align: *center*;

}

*#chatWindow* {

border: *2px solid #d1c4e9*;

border-radius: *10px*;

background: *rgba(255, 255, 255, 0.1)*;

width: *100%*;

height: *300px*;

overflow-y: *auto*;

padding: *10px*;

margin-bottom: *20px*;

color: *white*;

}

*.message* {

padding: *5px*;

margin-bottom: *5px*;

border-radius: *8px*;

word-wrap: *break-word*;

}

*.message.sender* {

color: *#a8e6cf*;

text-align: *right*;

}

*.message.receiver* {

color: *#ff8b94*;

text-align: *left*;

}

**form** {

display: *flex*;

width: *100%*;

gap: *10px*;

}

**input**[type="text"] {

flex: *1*;

padding: *10px*;

border: *2px solid #d1c4e9*;

border-radius: *25px*;

background: *rgba(255, 255, 255, 0.2)*;

color: *white*;

font-size: *16px*;

outline: *none*;

}

**button** {

padding: *10px 20px*;

border: *none*;

border-radius: *25px*;

background: *linear-gradient(135deg, #a02fc2, #d62d92)*;

color: *white*;

font-size: *16px*;

font-weight: *bold*;

cursor: *pointer*;

transition: *transform 0.3s ease, background 0.3s ease*;

}

**button***:hover* {

background: *linear-gradient(135deg, #d62d92, #a02fc2)*;

transform: *scale(1.05)*;

}

</style>

</head>

<body>

<div class=*"chat-container"*>

<h1>Chat Application</h1>

<div id=*"chatWindow"*></div>

<form id=*"chatForm"*>

<input type=*"text"* id=*"messageInput"* placeholder=*"Type a message..."* required />

<button type=*"submit"*>Send</button>

</form>

</div>

<script src=*"https://cdn.jsdelivr.net/npm/sockjs-client/dist/sockjs.min.js"*></script>

<script src=*"https://cdn.jsdelivr.net/npm/stompjs/lib/stomp.min.js"*></script>

<script>

**const** chatWindow = document.getElementById('chatWindow');

**const** messageInput = document.getElementById('messageInput');

**const** chatForm = document.getElementById('chatForm');

// WebSocket connection

**const** socket = **new** SockJS('/ws');

**const** stompClient = Stomp.over(socket);

// Fetch messages from the server on page load

fetch('/api/messages')

.then(response => {

**if** (!response.ok) {

**throw** **new** Error('Failed to fetch messages');

}

**return** response.json();

})

.then(messages => {

messages.forEach(displayMessage);

})

.**catch**(error => {

console.error('Error fetching messages:', error);

});

// Connect to WebSocket

stompClient.connect({}, **function** () {

stompClient.subscribe('/topic/public', **function** (messageOutput) {

**const** message = JSON.parse(messageOutput.body);

displayMessage(message);

});

});

// Send message on form submission

chatForm.addEventListener('submit', **function** (e) {

e.preventDefault();

**const** messageContent = messageInput.value.trim();

**if** (messageContent) {

**const** message = { sender: 'User1', content: messageContent };

stompClient.send('/app/chat.sendMessage', {}, JSON.stringify(message));

messageInput.value = '';

}

});

// Display a message in the chat window

**function** displayMessage(message) {

**const** messageDiv = document.createElement('div');

messageDiv.className = `message ${message.sender === 'User1' ? 'sender' : 'receiver'}`;

messageDiv.textContent = `${message.timestamp || ''} - ${message.sender}: ${message.content}`;

chatWindow.appendChild(messageDiv);

chatWindow.scrollTop = chatWindow.scrollHeight;

}

</script>

</body>

</html>

In repository, I have created a separate repository interface for chat :

**package** com.chatapplication.chatapp.repository;

**import** org.springframework.data.jpa.repository.JpaRepository;

**import** org.springframework.stereotype.Repository;

**import** com.chatapplication.chatapp.model.Messages;

@Repository

**public** **interface** MessagesRepository **extends** JpaRepository<Messages, Long>{

}

In Service class :

First I have @Autowired the repository interface in the service class

**private** **final** MessagesRepository messagesRepository;

@Autowired

**public** MessageService(RegistrationRepository registrationRepository, MessagesRepository messagesRepository) {

**this**.registrationRepository = registrationRepository;

**this**.messagesRepository=messagesRepository;

}

And then written some method like getAllMessages(), getMessagesById(), deleteMessagesById() to perform various operation on the messages,

**public** Messages saveMessages(Messages messages) {

// Example of additional logic: Add a timestamp before saving

**public** Messages saveMessages(Messages messages) {

messages.setTimestamp(System.*currentTimeMillis*());

**return** messagesRepository.save(messages);

}

// Get all messages

**public** List<Messages> getAllMessages() {

**return** messagesRepository.findAll(Sort.*by*(Sort.Direction.***ASC***, "timestamp"));

}

// Find a message by ID

**public** Optional<Messages> getMessagesById(Long id) {

**return** messagesRepository.findById(id);

}

// Delete a message by ID

**public** **void** deleteMessagesById(Long id) {

**if** (messagesRepository.existsById(id)) {

messagesRepository.deleteById(id);

} **else** {

**throw** **new** RuntimeException("Message with ID " + id + " not found.");

}

}

In Controller class :

@GetMapping("/chat")

**public** String home() {

**return** "chat";

}

@MessageMapping("/chat.sendMessage")

@SendTo("/topic/public")

**public** Messages sendMessage(Messages message) {

Messages savedMessage = messageService.saveMessages(message);

message.setTimestamp(LocalDateTime.*now*().toString());

**return** message;

}

// Endpoint to fetch all messages

@GetMapping

**public** List<Messages> getAllMessages() {

**return** messageService.getAllMessages();

}

And for webpage configuration create a new class for WebSocket configuration to fetch data from the chat page:

**package** com.chatapplication.chatapp.config;

**import** org.springframework.context.annotation.Configuration;

**import** org.springframework.messaging.simp.config.MessageBrokerRegistry;

**import** org.springframework.web.socket.config.annotation.EnableWebSocketMessageBroker;

**import** org.springframework.web.socket.config.annotation.StompEndpointRegistry;

**import** org.springframework.web.socket.config.annotation.WebSocketMessageBrokerConfigurer;

@Configuration

@EnableWebSocketMessageBroker

**public** **class** AppConfig **implements** WebSocketMessageBrokerConfigurer {

@Override

**public** **void** configureMessageBroker(MessageBrokerRegistry config) {

config.enableSimpleBroker("/topic");

config.setApplicationDestinationPrefixes("/app"); }

@Override

**public** **void** registerStompEndpoints(StompEndpointRegistry registry) {

registry.addEndpoint("/ws").withSockJS();

}

}

And the final class was SimpleChatApplication chat to run our project :

**package** com.chatapplication.chatapp;

**import** org.springframework.boot.SpringApplication;

**import** org.springframework.boot.autoconfigure.SpringBootApplication;

**import** org.springframework.boot.autoconfigure.domain.EntityScan;

**import** org.springframework.context.annotation.ComponentScan;

**import** org.springframework.data.jpa.repository.config.EnableJpaRepositories;

@SpringBootApplication

@EnableJpaRepositories(basePackages = "com.chatapplication.chatapp.repository")

@EntityScan(basePackages = "com.chatapplication.chatapp.model")

**public** **class** SimpleChatApplication {

**public** **static** **void** main(String[] args) {

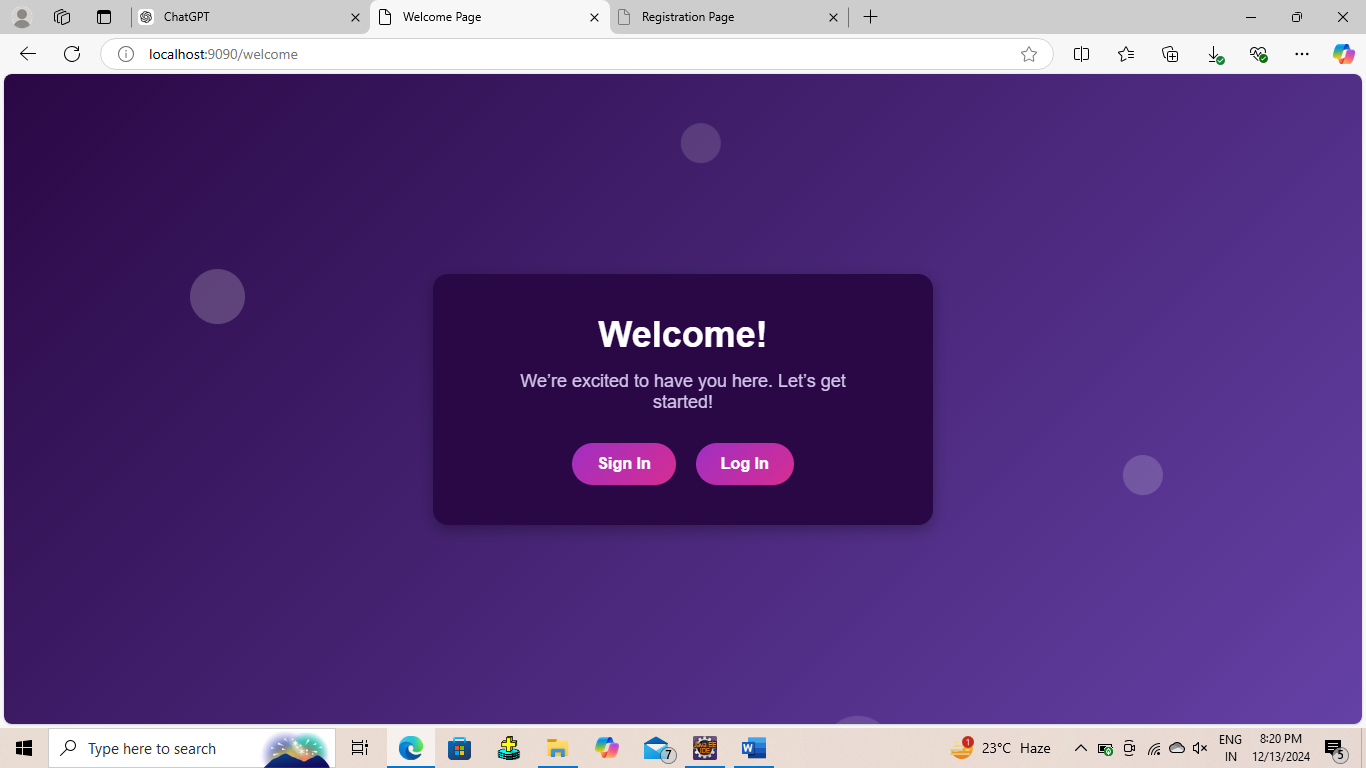
SpringApplication.*run*(SimpleChatApplication.**class**, args);

}

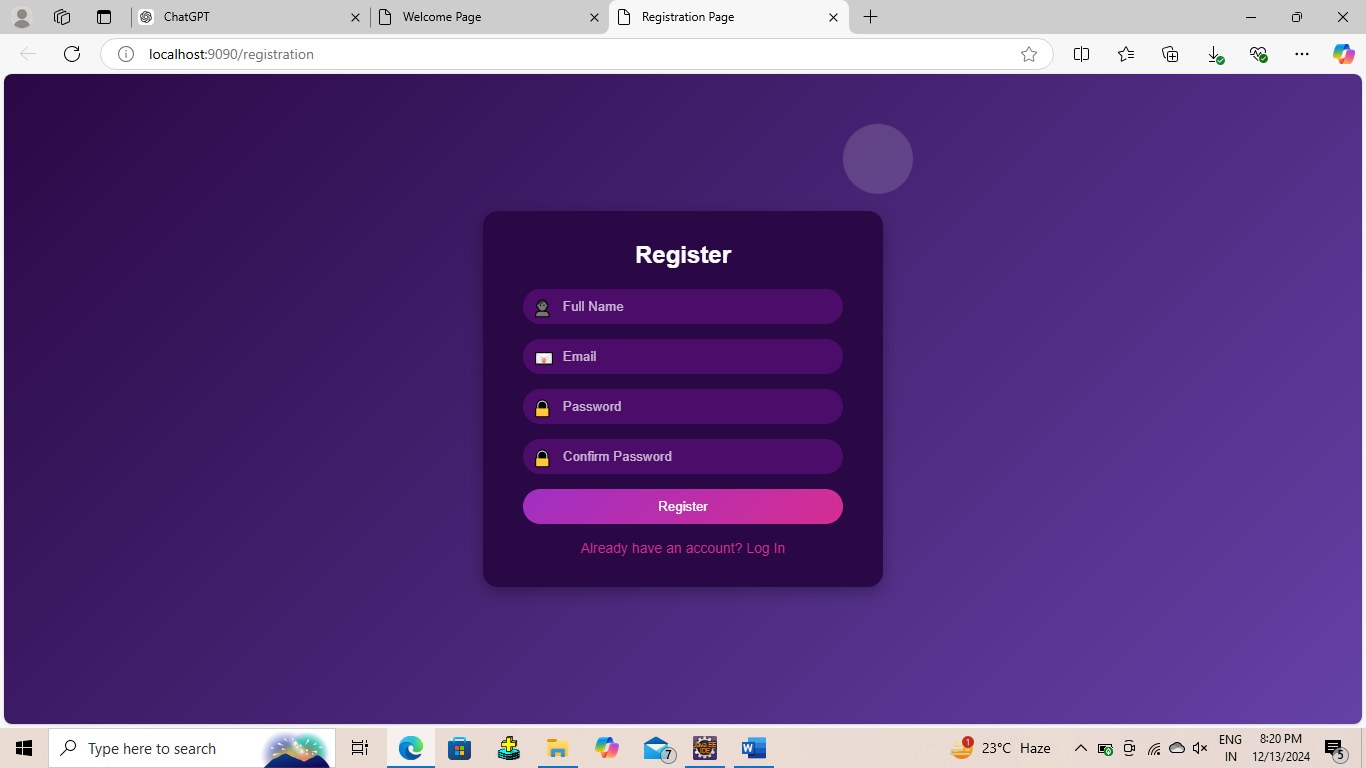
}

**Screenshots:**

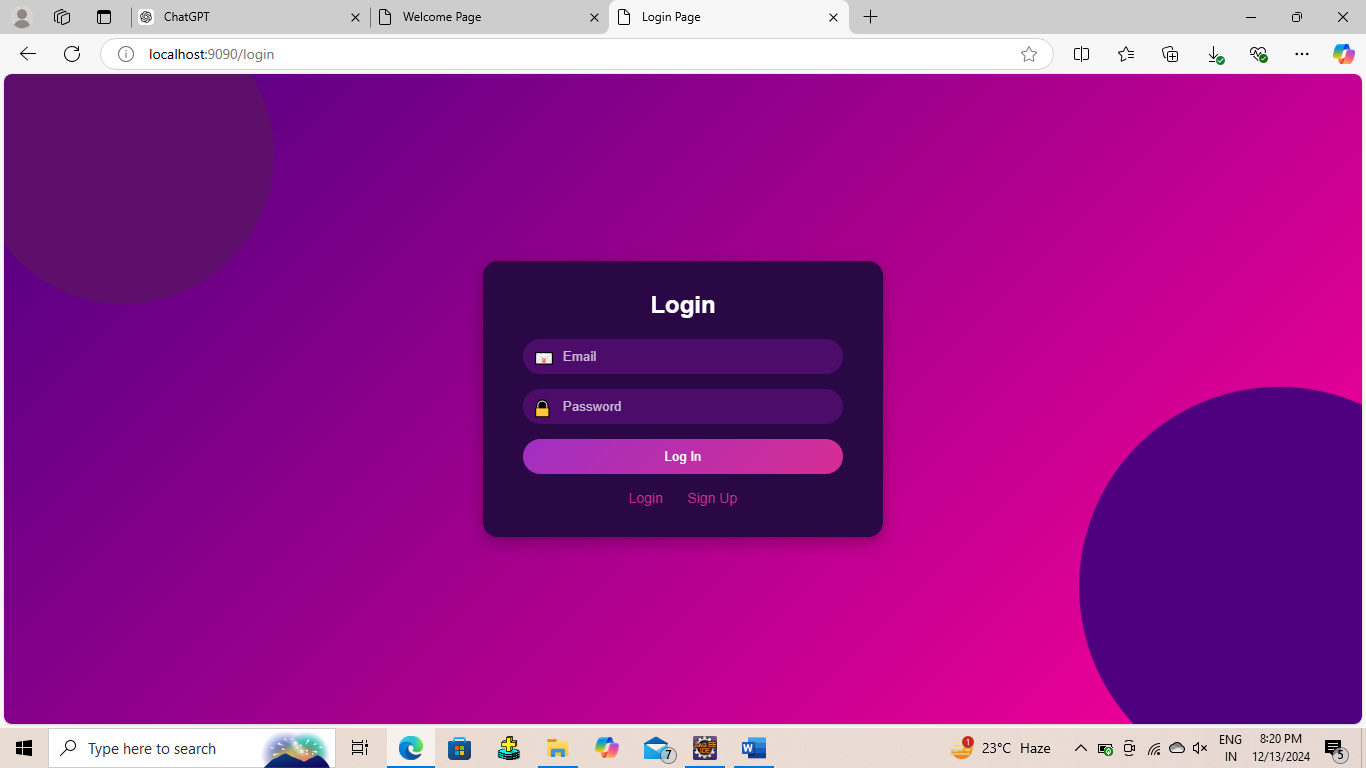
**Figure 1: welcome page**



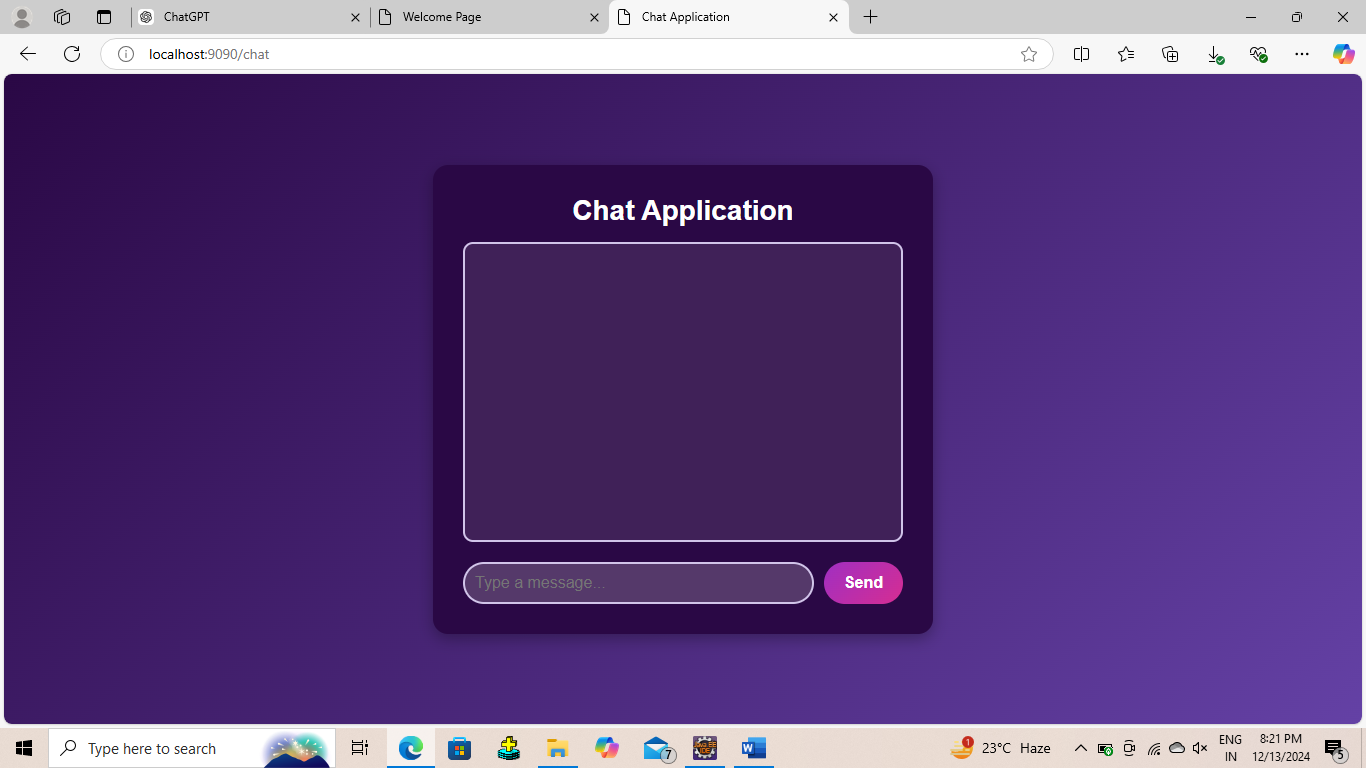
**Figure 2: Registration page**



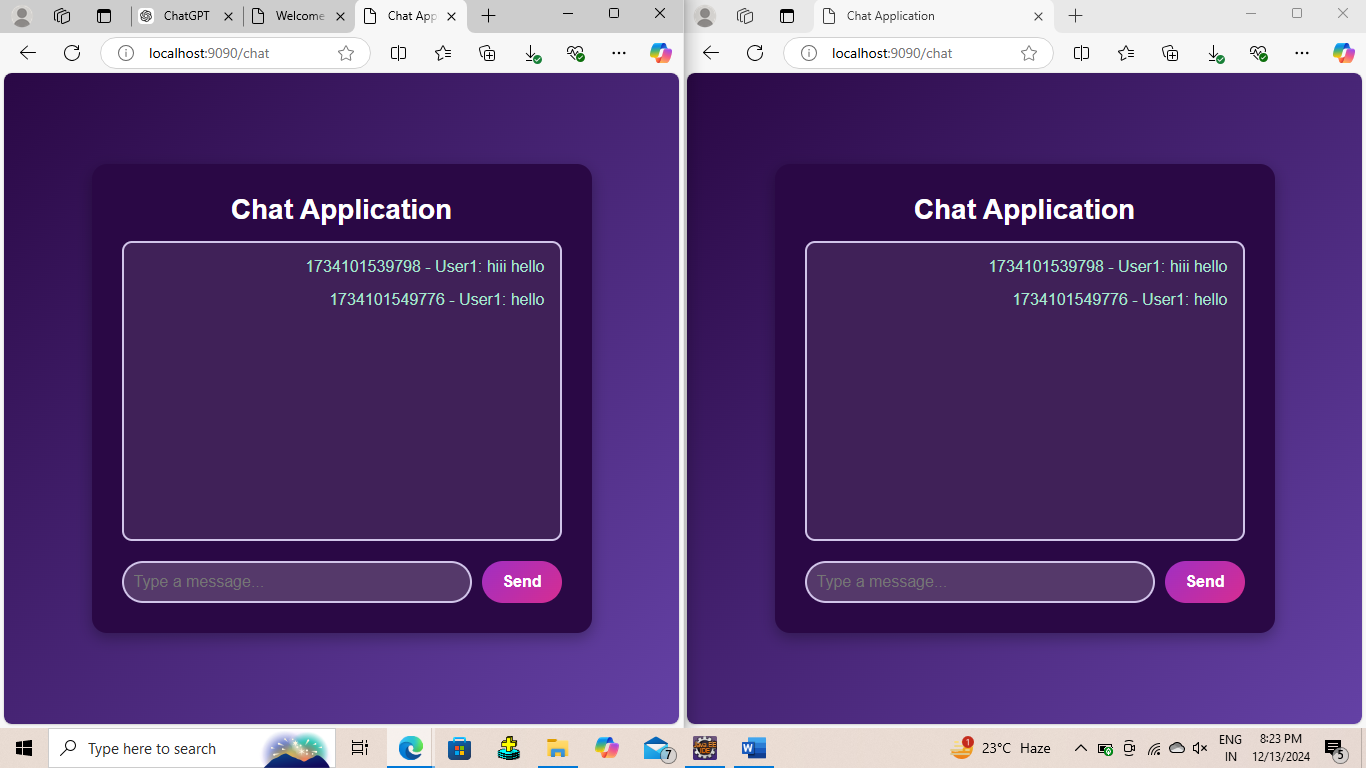
**Figure 3: Login page**



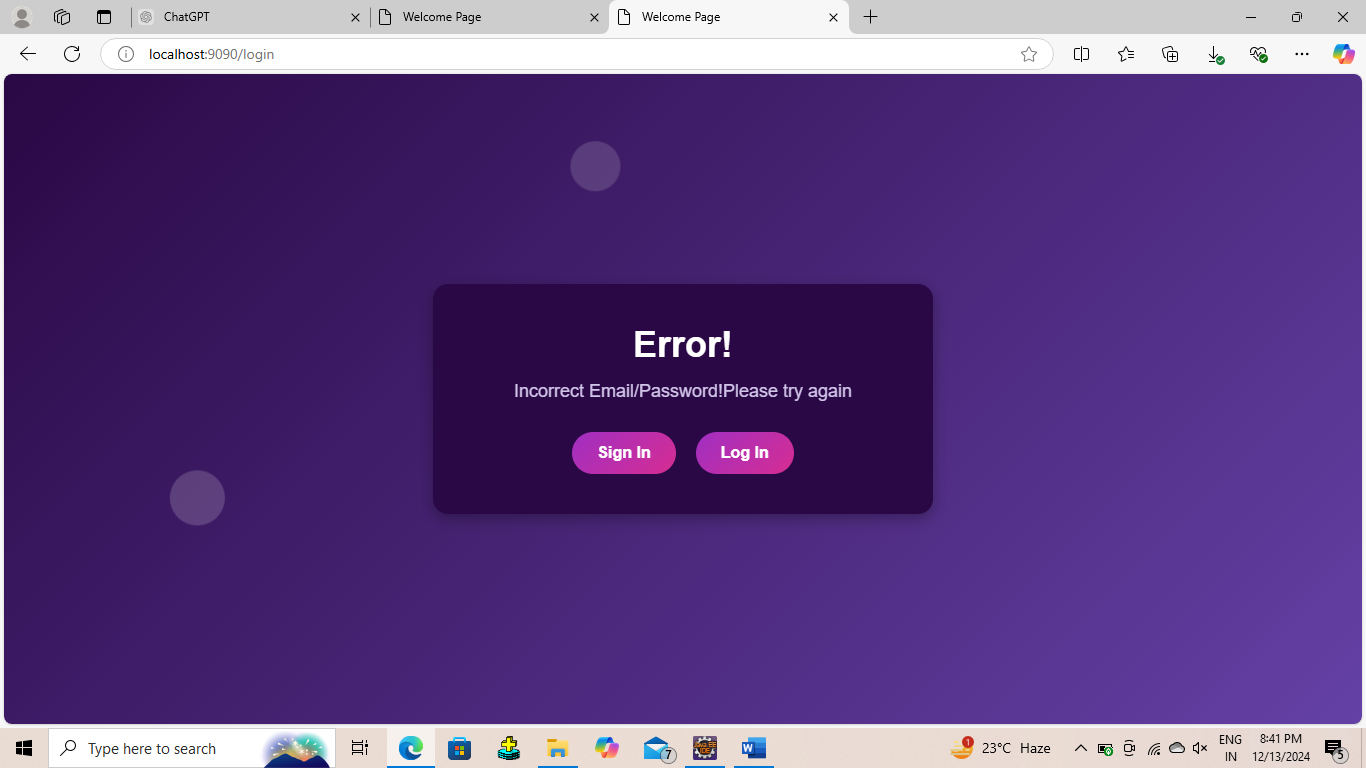
**Figure 4: Chat page**

****

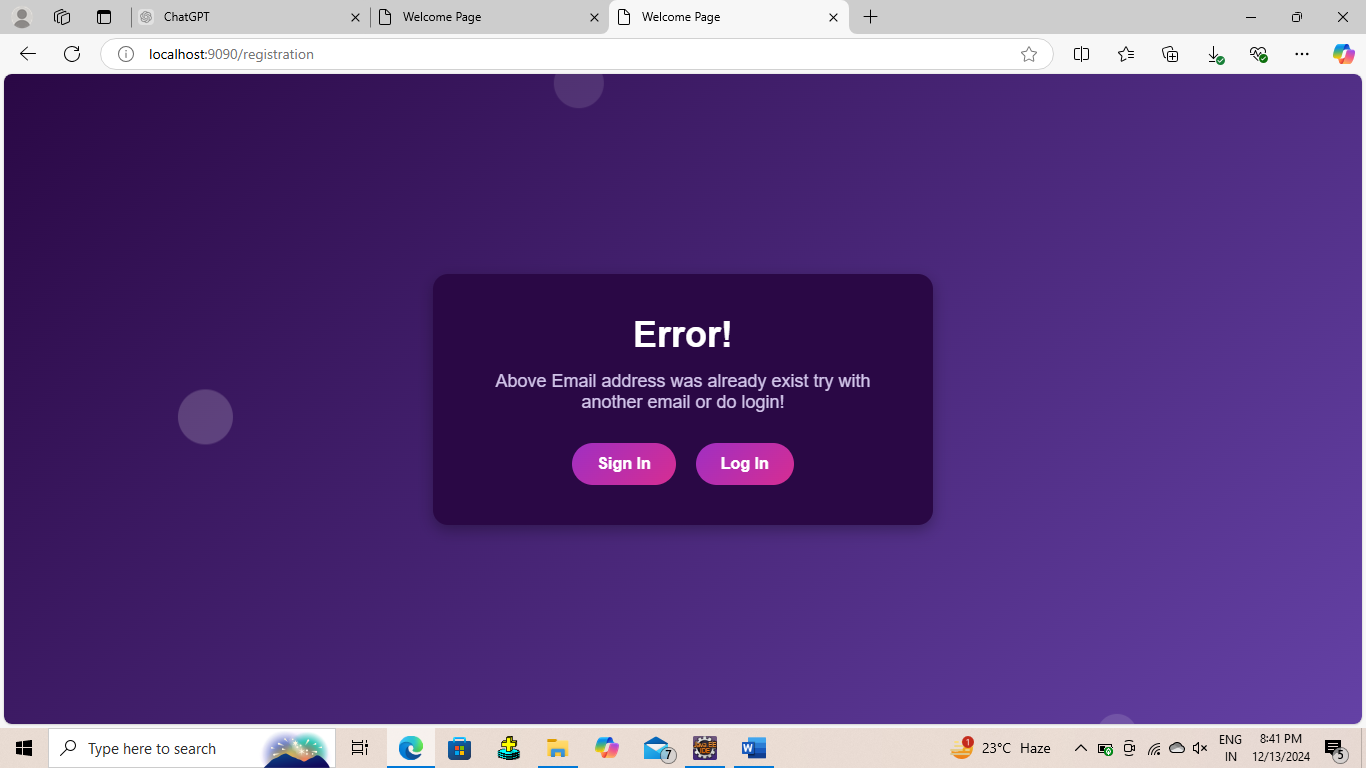
**Figure 5:**

****

**Figure 6: Login Error page**



**Figure 7: Registration Error page**



**FutureEnhancement:**

1. **Load Previous Messages**

* **Problem**: Messages disappear when the page is refreshed.
* **Solution**:
  + Create an API (@GetMapping("/chat/messages") to fetch all messages from the database.
  + Use fetch in JavaScript to get and display messages when the page loads.

**Code**:

* + Backend:

java

Copy code

@GetMapping("/chat/messages")

public List<Messages> getAllMessages() {

return messageService.getAllMessages();

}

* + Frontend:

javascript

Copy code

window.onload = function () {

fetch('/chat/messages')

.then(response => response.json())

.then(messages => {

messages.forEach(displayMessage); // Add each message to the chat window

});

};

2. **Add Usernames**

* **Problem**: All users are treated the same.
* **Solution**:
  + Add a login screen to capture the username.
  + Send the username with each message.

3. **Private Chats**

* **Problem**: All messages are visible to everyone.
* **Solution**:
  + Add private chat channels (e.g., /topic/{userId}).
  + Allow users to choose whom they want to chat with.

4. **Improve UI**

* Show timestamps for each message.
* Highlight the sender and receiver with different colors.
* Make it mobile-friendly with responsive design.

5. **Notifications**

* Show browser notifications for new messages when the user is not on the chat page

**Simple Document Structure**

1. **Current Features**:Describe what works now (real-time chat, saving messages, etc.).
2. **Issues**:Mention key problems: messages lost on refresh, no usernames, no private chat, etc.
3. **Enhancements**:Briefly describe each improvement (e.g., loading old messages, adding usernames, private chats, UI upgrades).
4. **Future Plans**:Mention ideas like file sharing, multilingual chat, or integration with other services.

**Conclusion:**

In conclusion, the chat application successfully facilitates real-time communication using WebSocket and SpringBoot, with messages being stored in the database for potential persistence. To enhance functionality, implementing message retrieval on page reload, user authentication, private chats, and UI improvements will greatly improve user experience and scalability. These enhancements ensure the application remains robust, user-friendly, and ready for future expansion.