

A Capstone Project Report
on
DO CONNECT WEB APPLICATION

Submitted by
C4 -Group - 9



Group Members :

1. M.T.P. Sai Pavan
2. M. Revanth
3. L. Gowtham
4. P. Deepak
5. CH. Surya Teja
6. Bhavi Alkhaniya

Guided by :

Ms. Brahmavidya

INDEX

S.NO	LIST OF CONTENTS	PAGE NO
1	Objective of the Project	3
2	Introduction	3
3	Technologies Used	4
4	Tools Used	7
5	Modules	7
6	Entity Relationship (ER) Diagram	8
7	Unified Modelling Language (UML) Diagram	9
8	Screenshots of Frontend Application	11
9	Screenshots of Backend Application	15
10	Conclusion	22

TECHNICAL REPORT

DO CONNECT WEBSITE

1. OBJECTIVE OF THE PROJECT:

The Objective of the project is to develop a Web Application using different Web technologies to provide a service for an online Web Application. The project should have a simple user interface (UI) convenient for the user to understand and easy to use.

- **Front - End:**

Front End deals with the development of the graphical user interface of a Web Application, through the use of different technologies like HTML, CSS, and JavaScript , So that users can view and interact with that Application.

- **Back – End :**

Back End application deals with the databases from which data is manipulated and the main logic of the App is implemented using technologies like Spring Boot and SQL.

2. INTRODUCTION:

Do Connect is an online Web Application that is capable of helping Users to post the Question and Get Answers. There are two types of Users who can access the Application for different purposes.

They are:

- **User:**

The user has the ability to Login, Logout and Register into the Application. The User has an option to Ask questions faster. There is a Register Option in the application. User can register as a user and can create an account to post their question. As a User He can chat within the Application. He can Search the Questions that he asked for in the form of list.

- **Admin:**

The admin has the ability to access and See the List of Questions that User asks. As an Admin he can get the mail as soon as the new questions are Posted. Admin should only approve the question and answer only then user can see the answer on screen. Additionally, admin can also delete inappropriate Questions & Answers.

3. TECHNOLOGIES USED:

3.1 Angular - Frontend:

Angular is a TypeScript-based free and open-source web application framework led by the Angular Team at Google and by a community of individuals and corporations. Angular is a complete rewrite from the same team that built AngularJS. Angular is a platform and framework for building single-page client applications using HTML and TypeScript. Angular is written in TypeScript. It implements core and optional functionality as a set of TypeScript libraries that you import into your applications.

The structure of Angular application is as follows:

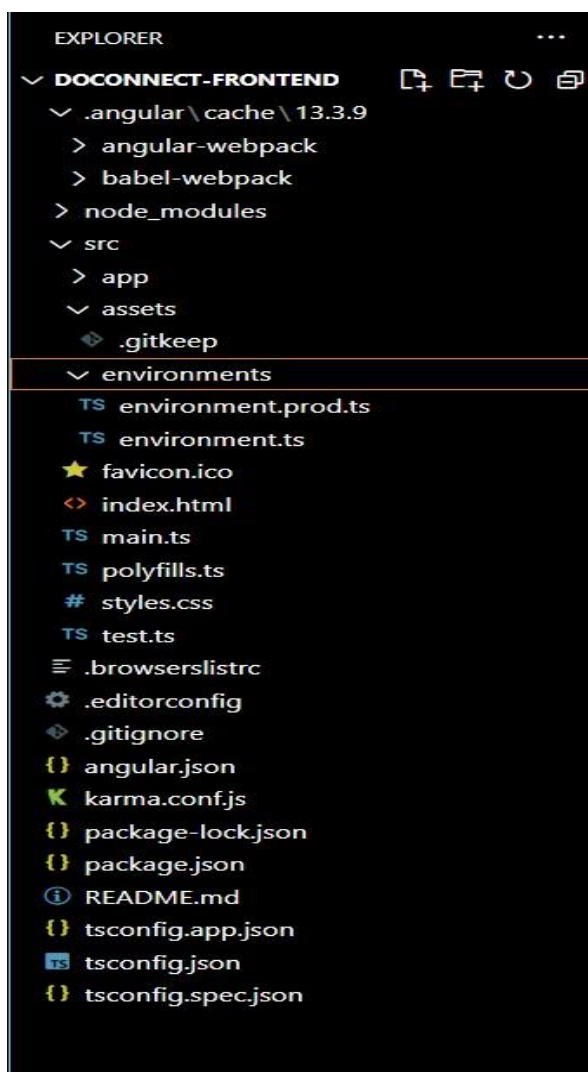
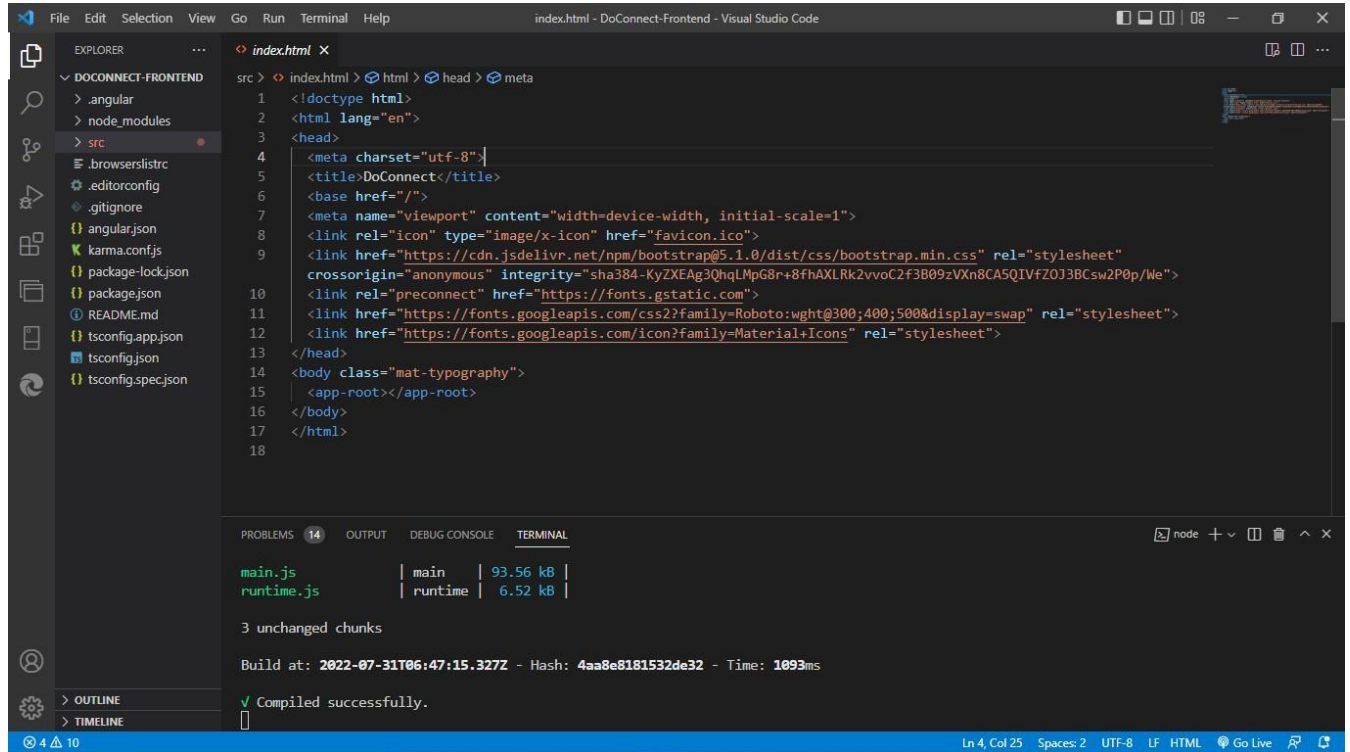
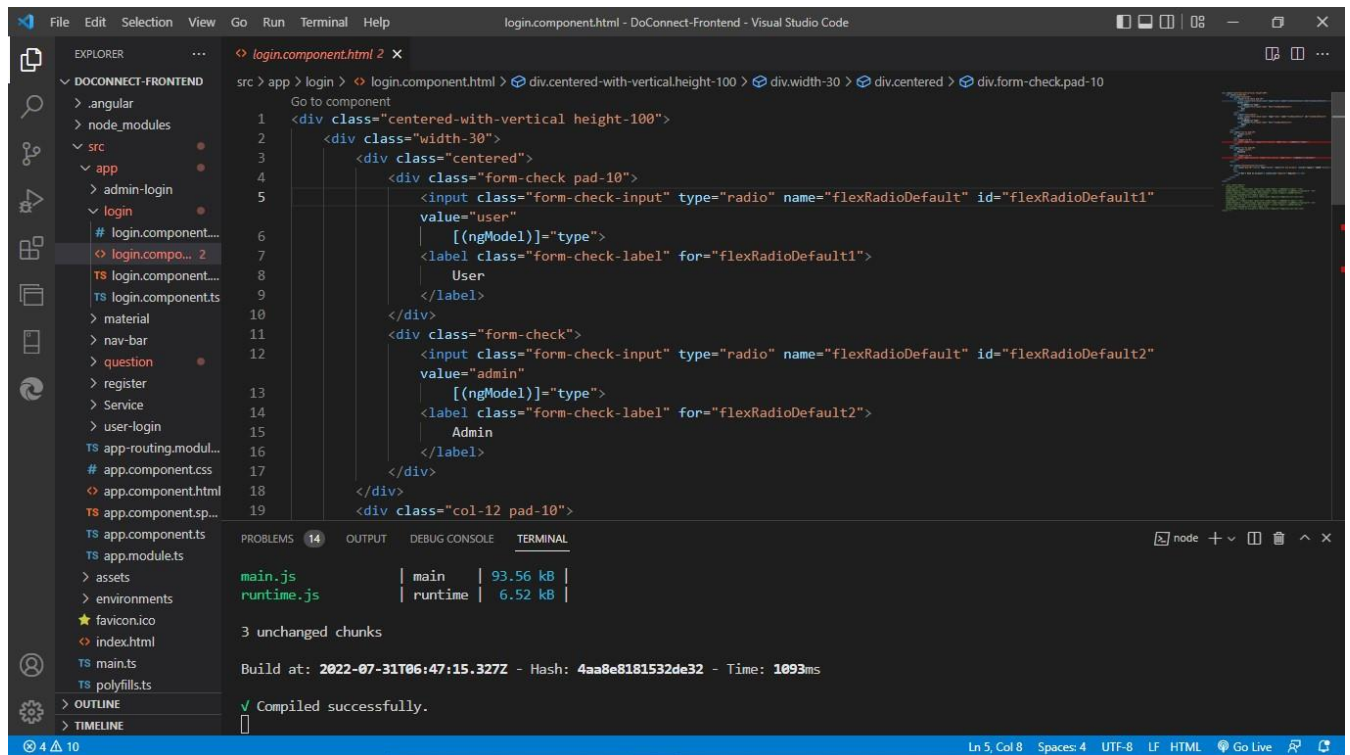


Fig. 3.1.1 Angular Directory Structure



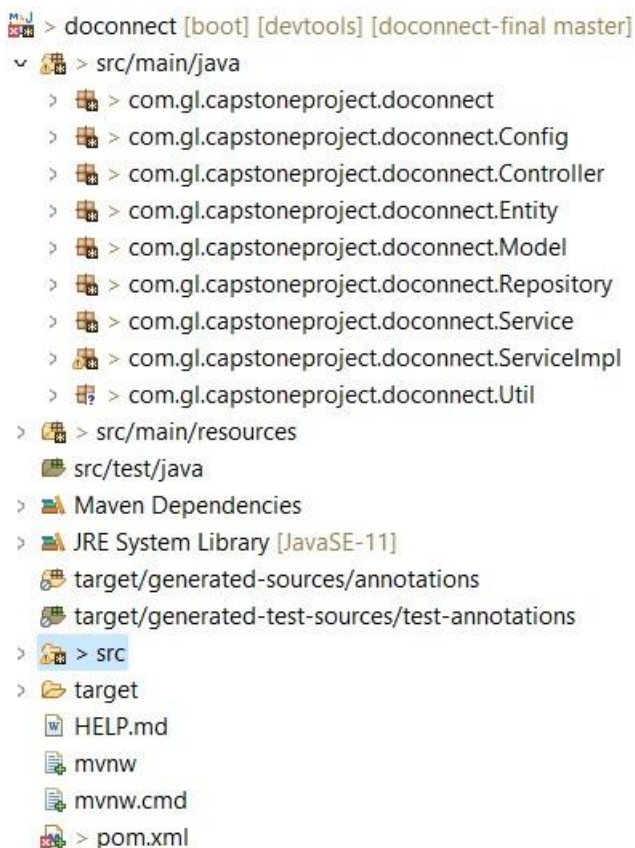
LOGIN COMPONENT



3.2 Spring Boot - Backend:

Java Spring Boot (Spring Boot) is a tool that makes developing web application with Spring Framework faster and easier through core capabilities. It is used to build standalone and production ready spring applications. Spring Boot works well with several servlet containers. Spring Boot helps developers create applications that just run. Specifically, it lets you create standalone applications that run on their own, without relying on an external web server, by embedding a web server such as Tomcat into your app during the initialization process.

The structure of Spring Boot application is as follows:



```
M:\J > doconnect [boot] [devtools] [doconnect-final master]
v > src/main/java
    > com.gl.capstoneproject.doconnect
    > com.gl.capstoneproject.doconnect.Config
    > com.gl.capstoneproject.doconnect.Controller
    > com.gl.capstoneproject.doconnect.Entity
    > com.gl.capstoneproject.doconnect.Model
    > com.gl.capstoneproject.doconnect.Repository
    > com.gl.capstoneproject.doconnect.Service
    > com.gl.capstoneproject.doconnect.ServiceImpl
    > com.gl.capstoneproject.doconnect.Util
> src/main/resources
  src/test/java
> Maven Dependencies
> JRE System Library [JavaSE-11]
  target/generated-sources/annotations
  target/generated-test-sources/test-annotations
> > src
> target
  HELP.md
  mvnw
  mvnw.cmd
  > pom.xml
```

Fig. 3.2.1 Spring Boot Directory Structure

4.TOOLS USED:

The tools used to develop this project include:

- **Visual Studio Code Editor** for Angular (**Front End**)
- **J2EE Eclipse IDE** for Spring Boot (**Back End**)
- **MySQL database** to store the data of the Web Application. .

5.MODULES:

The modules that have been implemented in the project are:

- **User:**
The user module contains all the fields of the users stored in the MySQL database and Vs Code its implementation using the data stored in the database.
- **Admin:**
The admin module contains all the fields of the admins stored in the MySQL database and Vs Code its implementation using the data stored in the database.
- **Login:**
The Login module contains all the fields of the values stored in the MySQL database and its implementation using the data stored in the database.
- **Register:**
The Register module contains all the fields of the values stored in the MySQL database and its implementation using the data stored in the database.
- **Question:**
The Question module contains all the fields of the values stored in the MySQL database and its implementation using the data stored in the database.

- **Search bar:**
The Search Bar module contains all the fields of the values stored in the MySQL database and its implementation using the data stored in the database.
- **Email:**
The email module contains all the fields of the values stored in the MySQL database and its implementation using the data stored in the database.

6. **ENTITY RELATIONSHIP (ER) DIAGRAM:**

The Entity Relationship diagram or the ER diagram is used to visually represent all the entities in the application and relationship between the entities. The entities in the project are:

- **User**
- **Admin**
- **Login**
- **Register**
- **Question**
- **Email**

The ER diagram of our project:

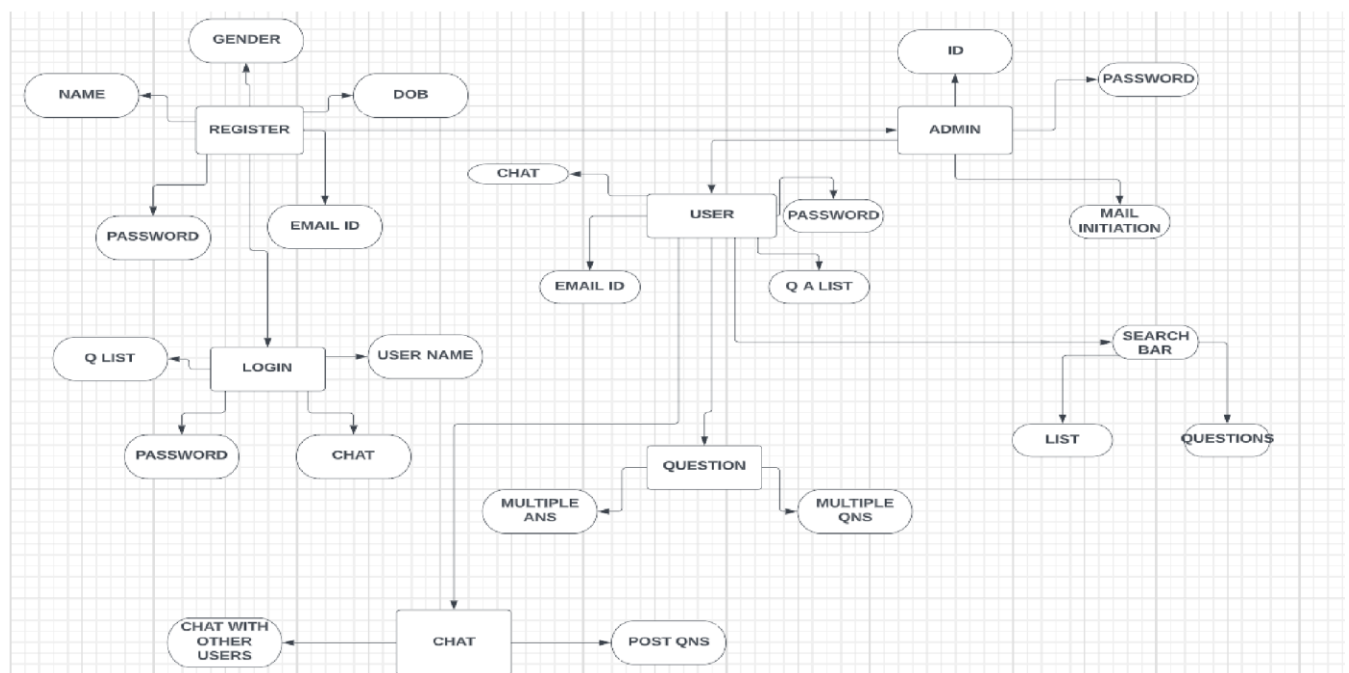


Fig. 6.1 Do Connect ER Diagram

7. UNIFIED MODELLING LANGUAGE (UML) DIAGRAMS:

The Unified Modelling Language diagrams or the UML diagrams are general-purpose, developmental, modeling language in the field of software engineering that is intended to provide a standard way to visualize the design of a system. Few types of UML diagrams are:

- **Use case diagram:**

A use case diagram is a graphical depiction of a user's possible interactions with a system. A use case diagram shows various use cases and different types of users the system has and will often be accompanied by other types of diagrams as well. The use cases are represented by either circles or ellipses.

- **Class diagram:**

A class diagram in the Unified Modeling Language is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations, and the relationships among objects.

- **Sequence diagram:**

A sequence diagram or system sequence diagram shows process interactions arranged in time sequence in the field of software engineering. It depicts the processes involved and the sequence of messages exchanged between the processes needed to carry out the functionality.

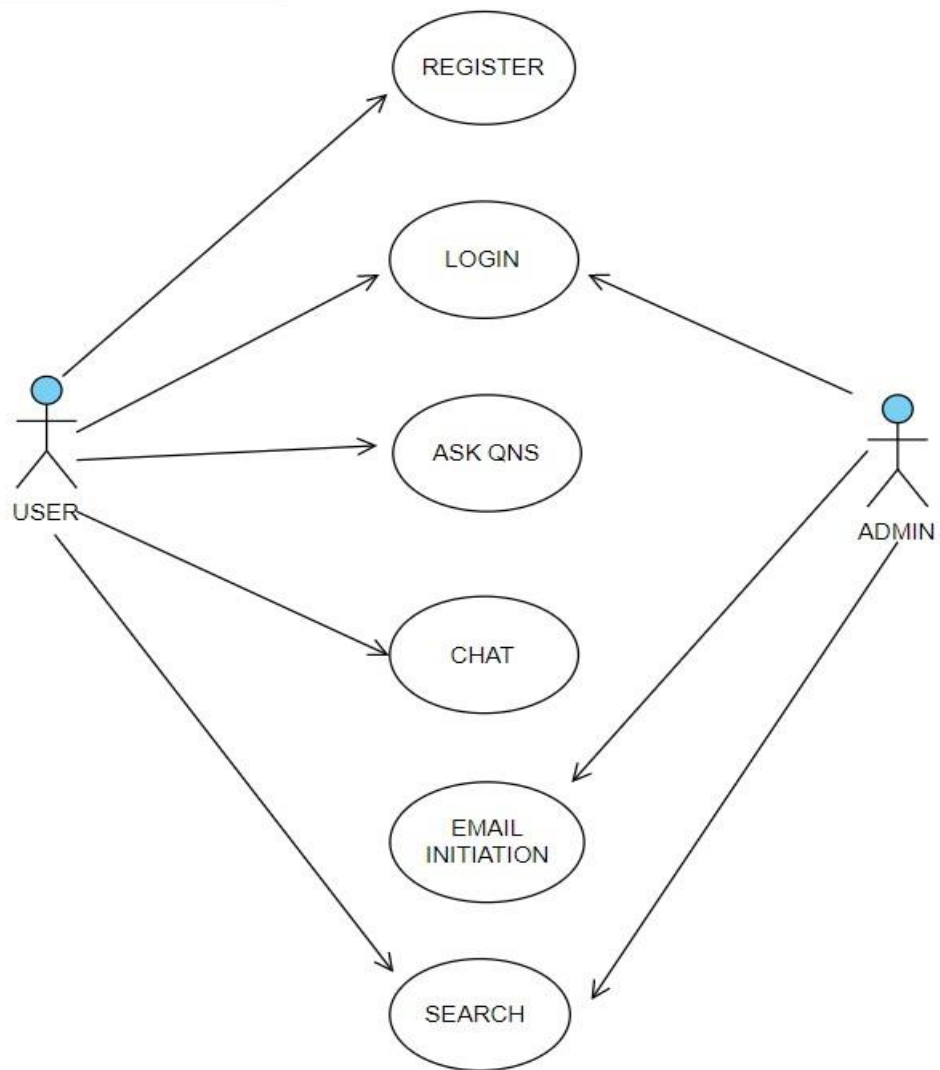
- **Activity diagram:**

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency.

- **State chart diagram:**

A State chart diagram describes a state machine. State machine can be defined as a machine which defines different states of an object and these states are controlled by external or internal events

7.1 Use Case Diagram:



8. SCREENSHOTS OF FRONT END APPLICATION:

The frontend application runs on localhost with port number 4200.

The screenshots of the localhost are as follows:

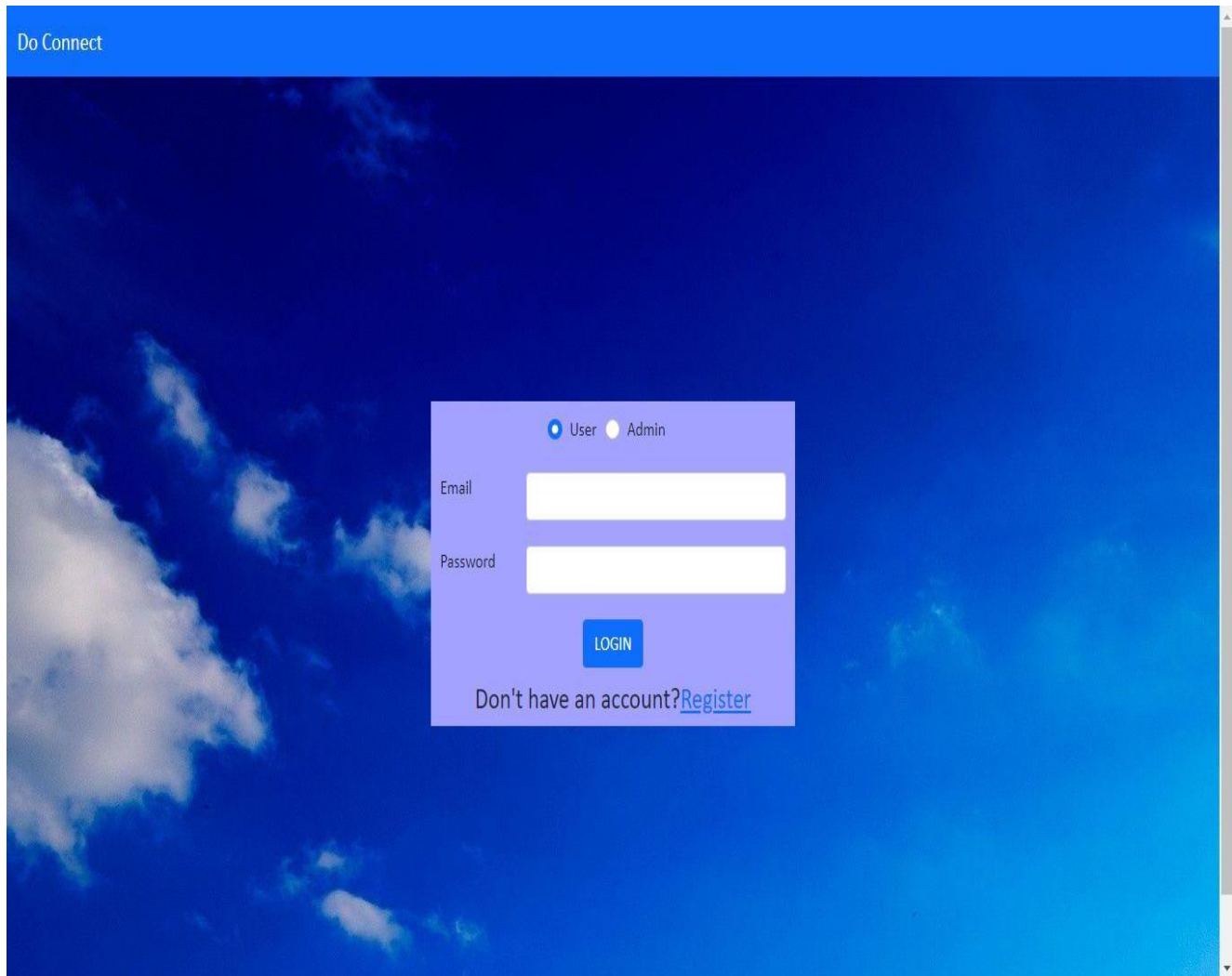
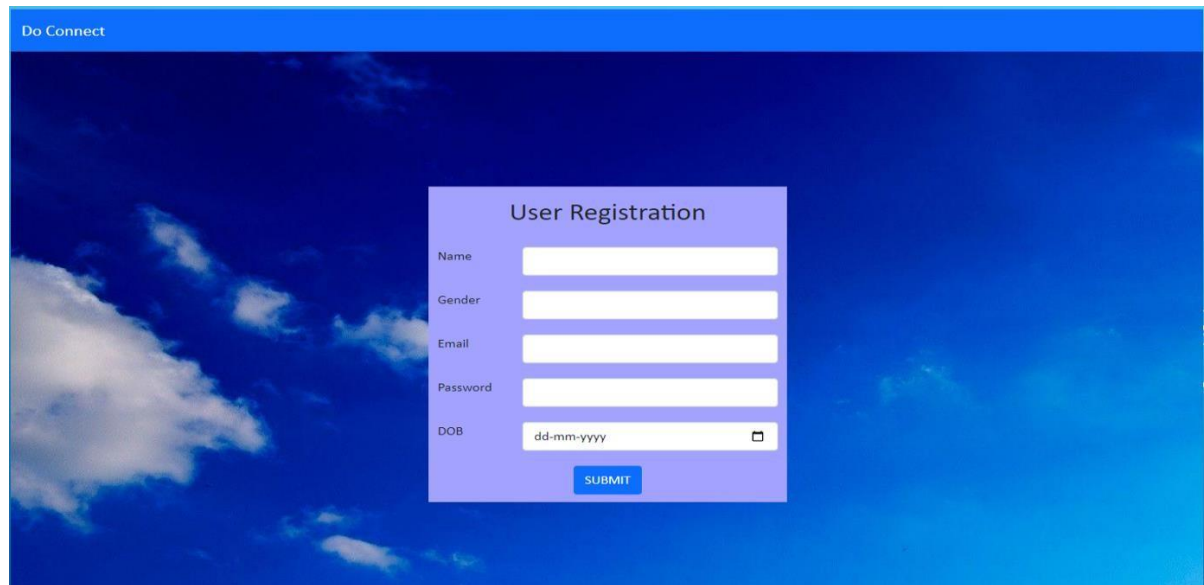


Fig. 8.1 Home Page



The registration form is titled "User Registration" and is set against a background of a blue sky with white clouds. It includes input fields for Name, Gender, Email, Password, and a Date of Birth (DOB) field with a "dd-mm-yyyy" placeholder and a calendar icon. A "SUBMIT" button is located at the bottom of the form.

Do Connect

User Registration

Name

Gender

Email

Password


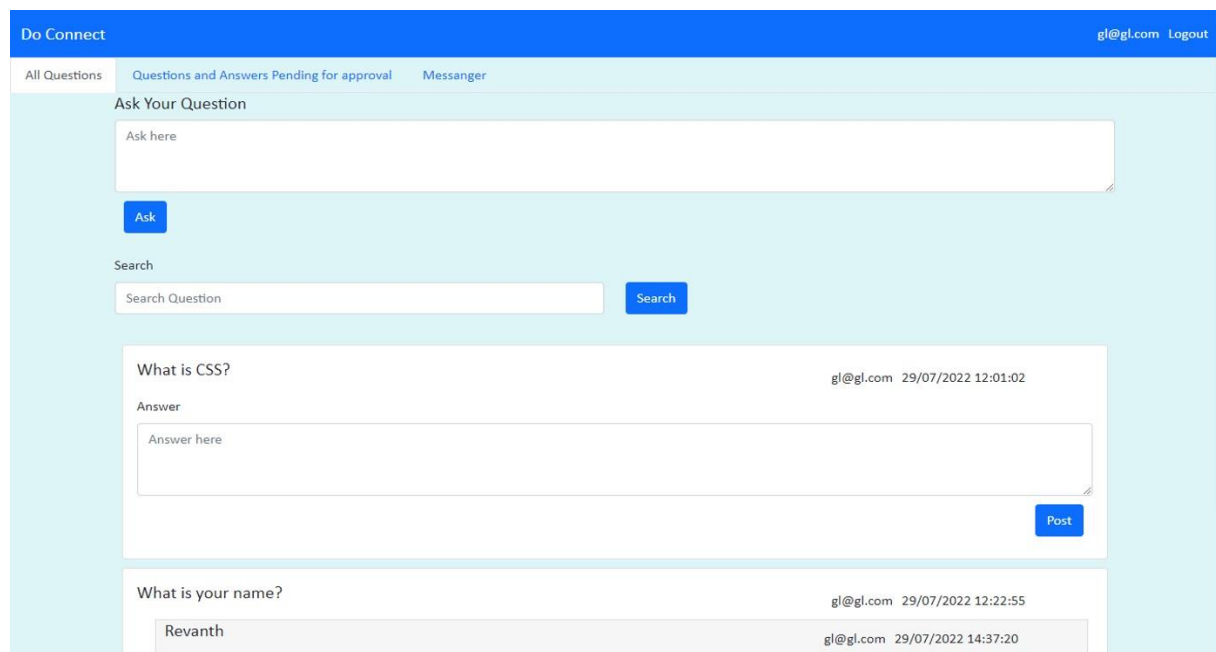
DOB 

Fig. 8.2 Registration Page



The page features a navigation bar with "Do Connect" on the left and "gl@gl.com Logout" on the right. Below the navigation bar are tabs for "All Questions", "Questions and Answers Pending for approval", and "Messenger". The main content area is titled "Ask Your Question" and contains a large text input field with the placeholder "Ask here" and an "Ask" button. Below this is a search section with a "Search Question" input field and a "Search" button. The page displays two question entries. The first entry is "What is CSS?" by "gl@gl.com" at "29/07/2022 12:01:02", with an "Answer" section containing an "Answer here" input field and a "Post" button. The second entry is "What is your name?" by "gl@gl.com" at "29/07/2022 12:22:55", with a response from "Revanth" by "gl@gl.com" at "29/07/2022 14:37:20".

Do Connect gl@gl.com Logout

All Questions Questions and Answers Pending for approval Messenger

Ask Your Question

Ask here

Search

Search Question

What is CSS? gl@gl.com 29/07/2022 12:01:02

Answer

Answer here

What is your name? gl@gl.com 29/07/2022 12:22:55

Revanth gl@gl.com 29/07/2022 14:37:20

Fig. 8.3 Question & Answer Page

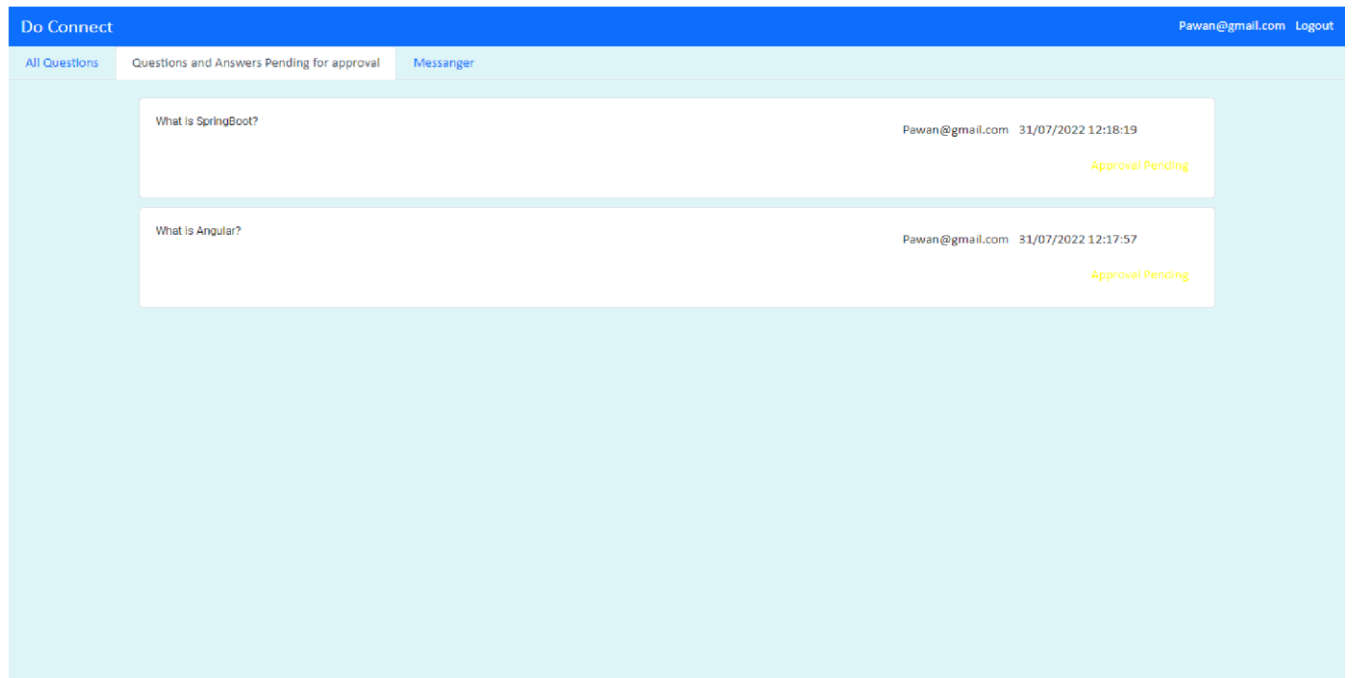


Fig. 8.4 UI for Q & A For Approval

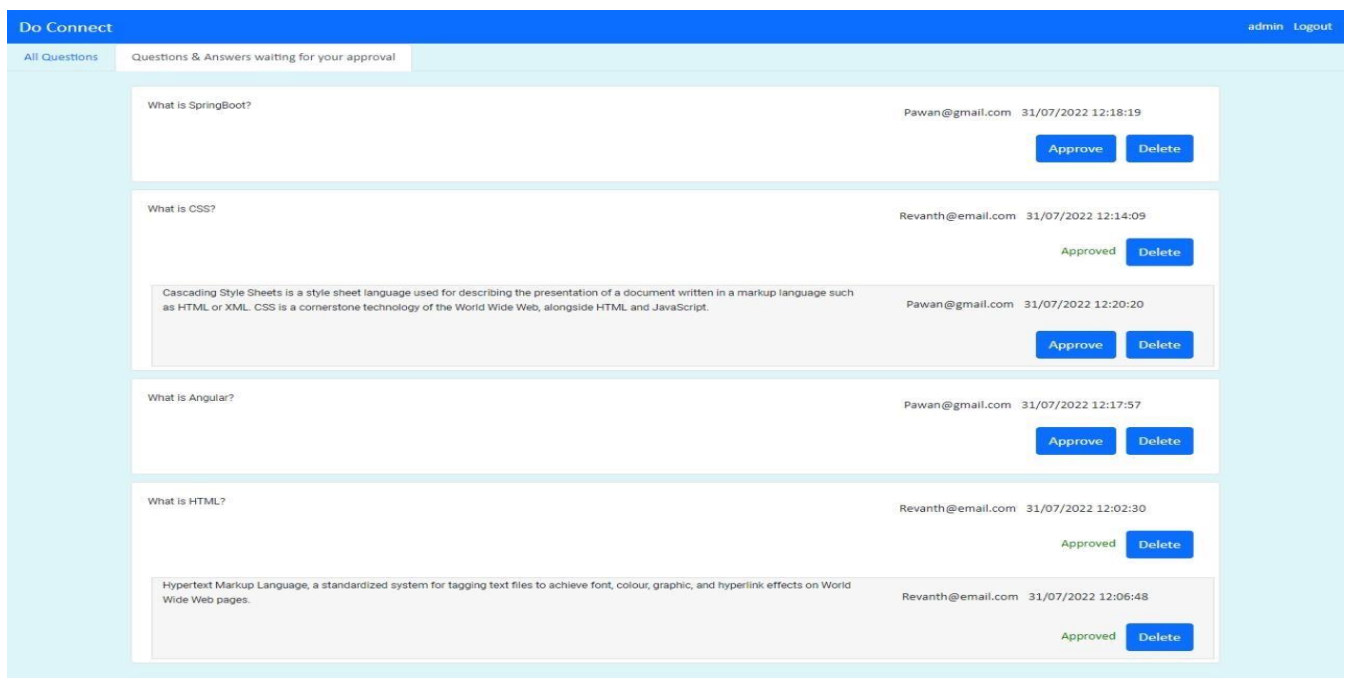


Fig 8.5 Admin Q & A Approval Interface

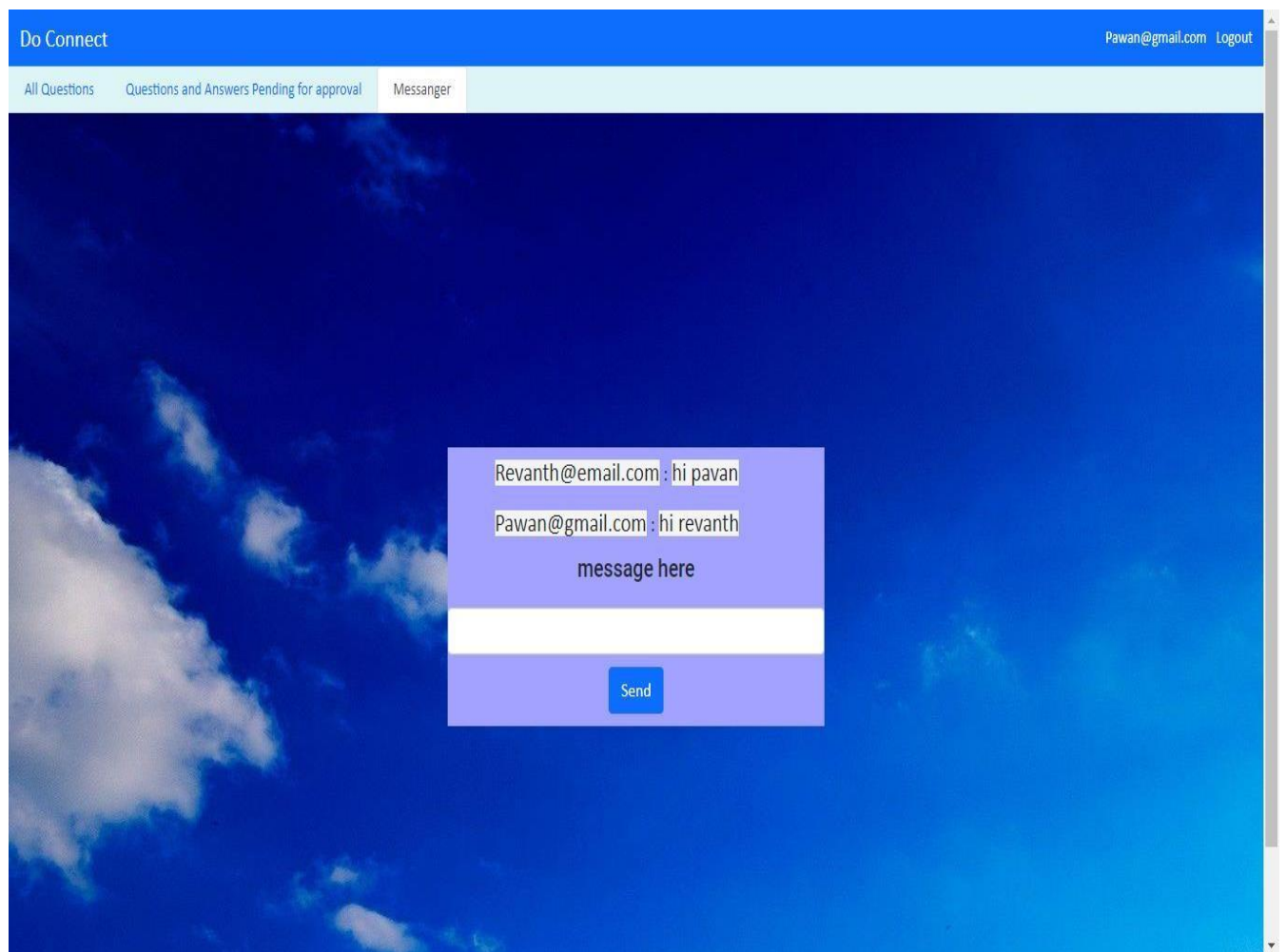


Fig. 8.6 App Chat Function

9. SCREENSHOTS OF BACK END APPLICATION:

The backend of the application is developed using Java Spring boot on localhost with port 8081. The screenshots of it are as follows:

A screenshot of an IDE showing the code for DoconnectApplication.java. The code is a Spring Boot application. It starts with a package declaration 'com.gl.capstoneproject.doconnect;', followed by an import 'org.springframework.boot.SpringApplication;'. The class is annotated with '@SpringBootApplication' and is a public class 'DoconnectApplication'. It contains a 'main' method that calls 'SpringApplication.run(DoconnectApplication.class, args);'.

```
1 package com.gl.capstoneproject.doconnect;
2
3 import org.springframework.boot.SpringApplication;
4
5
6 @SpringBootApplication
7 public class DoconnectApplication {
8
9     public static void main(String[] args) {
10         SpringApplication.run(DoconnectApplication.class, args);
11     }
12 }
```

Fig. 9.1 App Connection

A screenshot of an IDE showing the code for AdminService.java. The code defines a package 'com.gl.capstoneproject.doconnect.Service;', imports 'com.gl.capstoneproject.doconnect.Model.Admins;', and defines a public interface 'AdminService'. The interface has two methods: 'adminLogin(Admins admin);' and 'adminRegister(Admins admin);'.

```
1 package com.gl.capstoneproject.doconnect.Service;
2
3 import com.gl.capstoneproject.doconnect.Model.Admins;
4
5 public interface AdminService {
6
7     public String adminLogin(Admins admin);
8     public String adminRegister(Admins admin);
9 }
10
```

Fig. 9.2 Admin Login


```

1 package com.gl.capstoneproject.doconnect.ServiceImpl;
2
3 import org.slf4j.Logger;
4
5 @Service
6 public class AdminServiceImpl implements AdminService {
7
8     private static final Logger LOGGER = LoggerFactory.getLogger(UserServiceImpl.class);
9
10    @Autowired
11    private AdminRepository adminRepo;
12
13    public String adminLogin(Admins admin) {
14        try {
15            Admin adminlogin = adminRepo.findByEmailAndPassword(admin.getEmail(), admin.getPassword());
16            if (adminlogin.getEmail().equals(admin.getEmail())
17                && adminlogin.getPassword().equals(admin.getPassword())) {
18                return "Login Success";
19            } else {
20                return "User does not exist! Please register :>";
21            }
22        } catch (Exception e) {
23            LOGGER.error(e.getMessage());
24            return "User does not exist! Please register :>";
25        }
26    }
27
28    public String adminRegister(Admins admin) {
29        try {
30            Admin adminRegister = new Admin();
31            adminRegister.setId(admin.getId());
32            adminRegister.setName(admin.getName());
33            adminRegister.setDateOfBirth(admin.getDateOfBirth());
34            adminRegister.setGender(admin.getGender());
35            adminRegister.setPassword(admin.getPassword());
36            adminRegister.setEmail(admin.getEmail());
37
38            adminRepo.save(adminRegister);
39            return "registered successfully";
40        } catch (Exception e) {
41

```

Fig. 9.3 Admin Register

```

1 package com.gl.capstoneproject.doconnect.ServiceImpl;
2
3 import org.slf4j.Logger;
4
5 @Service
6 public class UserServiceImpl implements UserService {
7
8     private static final Logger LOGGER = LoggerFactory.getLogger(UserServiceImpl.class);
9
10    @Autowired
11    private UserRepository userRepo;
12
13    public String login(String body) {
14        try {
15            ObjectMapper mapper = new ObjectMapper();
16            UserLogin login = mapper.readValue(body, UserLogin.class);
17            User user = userRepo.findByEmailAndPassword(login.getEmail(), login.getPassword());
18
19            if (null != user) {
20                if (user.getEmail().equals(login.getEmail()) && user.getPassword().equals(login.getPassword())) {
21                    return "Login Success";
22                } else {
23                    return "User does not exist! Please register :>";
24                }
25            } else {
26                return "User does not exist! Please register :>";
27            }
28        } catch (Exception e) {
29            LOGGER.error(e.getMessage());
30            return null;
31        }
32    }
33
34    public String register(String body) {
35        try {
36            ObjectMapper mapper = new ObjectMapper();
37            UserLogin register = mapper.readValue(body, UserLogin.class);
38            User user1 = new User();
39            user1.setId(register.getId());
40

```

Fig. 9.4 User Register


```

1 package com.gl.capstoneproject.doconnect.Model;
2
3 import lombok.Data;
4
5 @Data
6 public class UserLogin {
7
8     public int getId() {
9         return id;
10    }
11    public void setId(int id) {
12        this.id = id;
13    }
14    public String getName() {
15        return name;
16    }
17    public void setName(String name) {
18        this.name = name;
19    }
20    public String getGender() {
21        return gender;
22    }
23    public void setGender(String gender) {
24        this.gender = gender;
25    }
26    public String getDateOfBirth() {
27        return dateOfBirth;
28    }
29    public void setDateOfBirth(String dateOfBirth) {
30        this.dateOfBirth = dateOfBirth;
31    }
32    public String getEmail() {
33        return email;
34    }
35    public void setEmail(String email) {
36        this.email = email;
37    }
38    public String getPassword() {
39        return password;
40    }
41    public void setPassword(String password) {

```

Fig. 9.5 User Login

```

1 package com.gl.capstoneproject.doconnect.Config;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4
5 @Component
6 public class MailConfig {
7
8     @Autowired
9     private JavaMailSender mail;
10
11     @Value("${spring.mail.username}")
12     private String fromMail;
13
14     public void sendMail(String toEmail, String subject, String body) {
15
16         SimpleMailMessage message = new SimpleMailMessage();
17         message.setFrom(fromMail);
18         message.setTo(toEmail);
19         message.setText(body);
20         message.setSubject(subject);
21         mail.send(message);
22     }
23 }

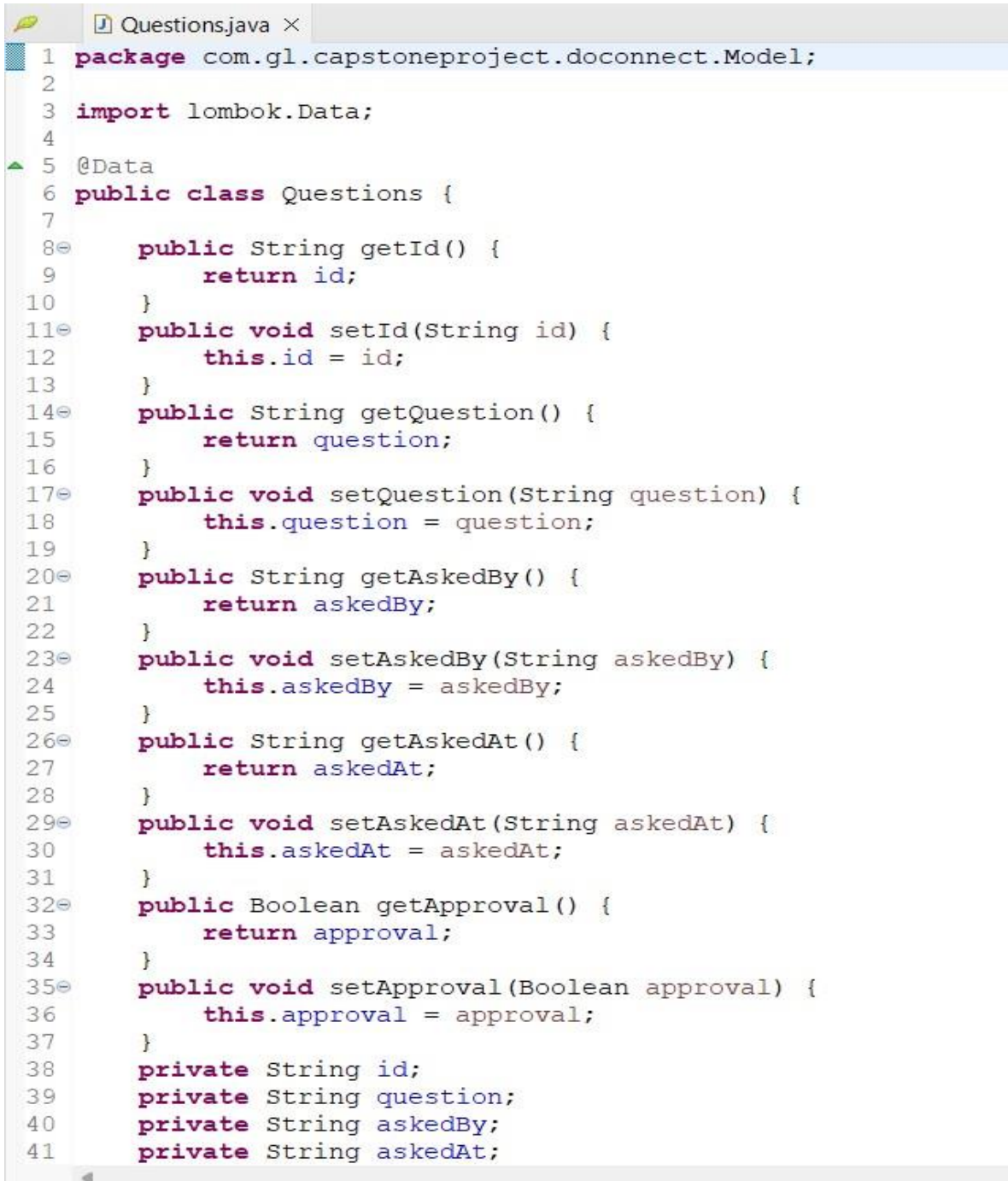
```

```

1 package com.gl.capstoneproject.doconnect.Model;
2
3 public class Answers {
4
5     public String getId() {
6         return id;
7     }
8
9     public void setId(String id) {
10         this.id = id;
11     }
12
13     public String getQuestionId() {
14         return questionId;
15     }
16
17     public void setQuestionId(String questionId) {
18         this.questionId = questionId;
19     }
20
21     public String getAnswer() {
22         return answer;
23     }
24
25     public void setAnswer(String answer) {
26         this.answer = answer;
27     }
28
29     public String getAnsweredBy() {
30         return answeredBy;
31     }
32
33     public void setAnsweredBy(String answeredBy) {
34         this.answeredBy = answeredBy;
35     }
36
37     public String getAnsweredAt() {
38         return answeredAt;
39     }
40
41     public void setAnsweredAt(String answeredAt) {
42         this.answeredAt = answeredAt;
43     }
44
45     public Boolean getApproval() {
46         return approval;
47     }
48
49     public void setApproval(Boolean approval) {
50         this.approval = approval;
51     }
52 }

```

Fig. 9.8 Answer Service



```

1 package com.gl.capstoneproject.doconnect.Model;
2
3 import lombok.Data;
4
5 @Data
6 public class Questions {
7
8     public String getId() {
9         return id;
10    }
11    public void setId(String id) {
12        this.id = id;
13    }
14    public String getQuestion() {
15        return question;
16    }
17    public void setQuestion(String question) {
18        this.question = question;
19    }
20    public String getAskedBy() {
21        return askedBy;
22    }
23    public void setAskedBy(String askedBy) {
24        this.askedBy = askedBy;
25    }
26    public String getAskedAt() {
27        return askedAt;
28    }
29    public void setAskedAt(String askedAt) {
30        this.askedAt = askedAt;
31    }
32    public Boolean getApproval() {
33        return approval;
34    }
35    public void setApproval(Boolean approval) {
36        this.approval = approval;
37    }
38    private String id;
39    private String question;
40    private String askedBy;
41    private String askedAt;

```

Fig. 9.9 Questions Service

```

1 package com.gl.capstoneproject.doconnect.Service;
2
3 import com.gl.capstoneproject.doconnect.Model.Admins;
4
5 public interface AdminService {
6
7     public String adminLogin(Admins admin);
8     public String adminRegister(Admins admin);
9 }
10

```

Fig. 9.10 Admin Service

```

1 package com.gl.capstoneproject.doconnect.Controller;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4
5 @CrossOrigin(origins = "http://localhost:4200")
6 @RestController
7 public class DoConnectController {
8
9     @Autowired
10     private UserService userService;
11
12     @Autowired
13     private QuestionsService quesService;
14
15     @Autowired
16     private AnswerService ansService;
17
18     @Autowired
19     private AdminService adminService;
20
21     @PostMapping("user/login")
22     public String login(@RequestBody String body) {
23         return userService.login(body);
24     }
25
26     @PostMapping("user/register")
27     public String registration(@RequestBody String body) {
28         return userService.register(body);
29     }
30
31     @PostMapping("admin/login")
32     public String adminLogin(@RequestBody Admins admin) {
33         return adminService.adminLogin(admin);
34     }
35
36     @PostMapping("admin/register")
37     public String adminRegister(@RequestBody Admins admin) {
38         return adminService.adminRegister(admin);
39     }
40
41     @PostMapping("question/ask")
42

```

Fig 9.11 App Controller

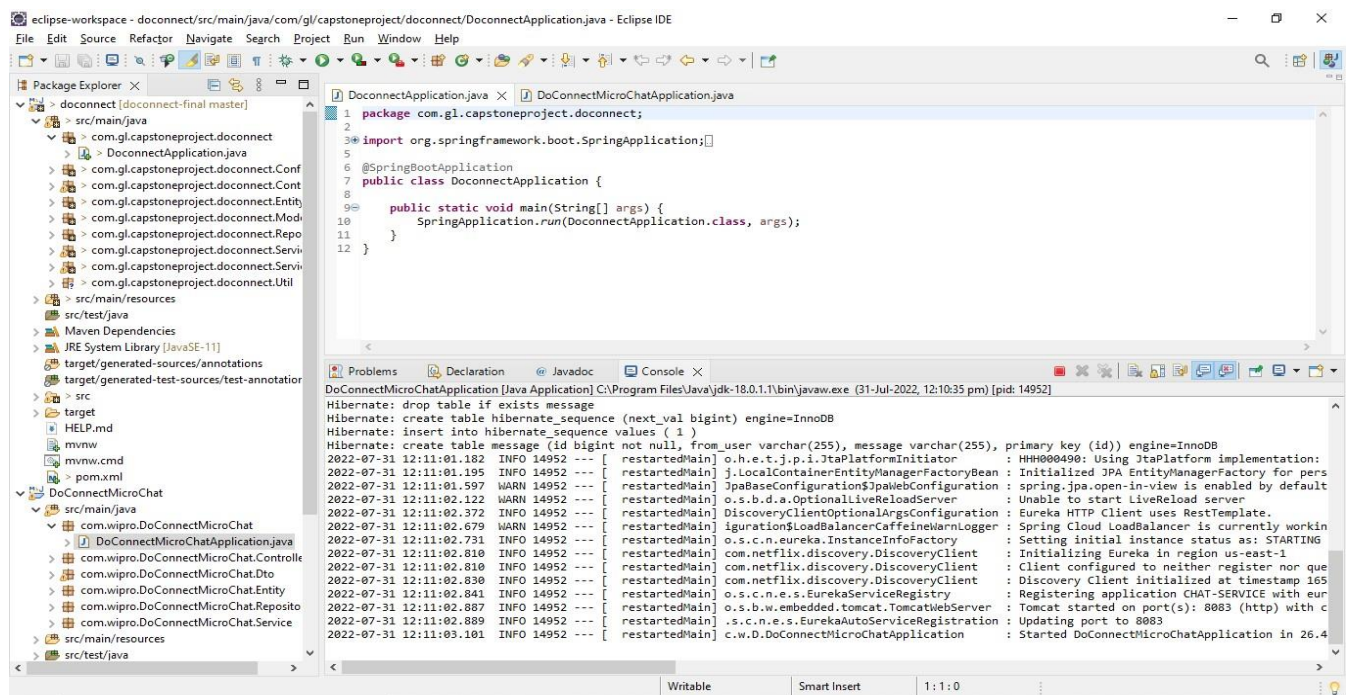
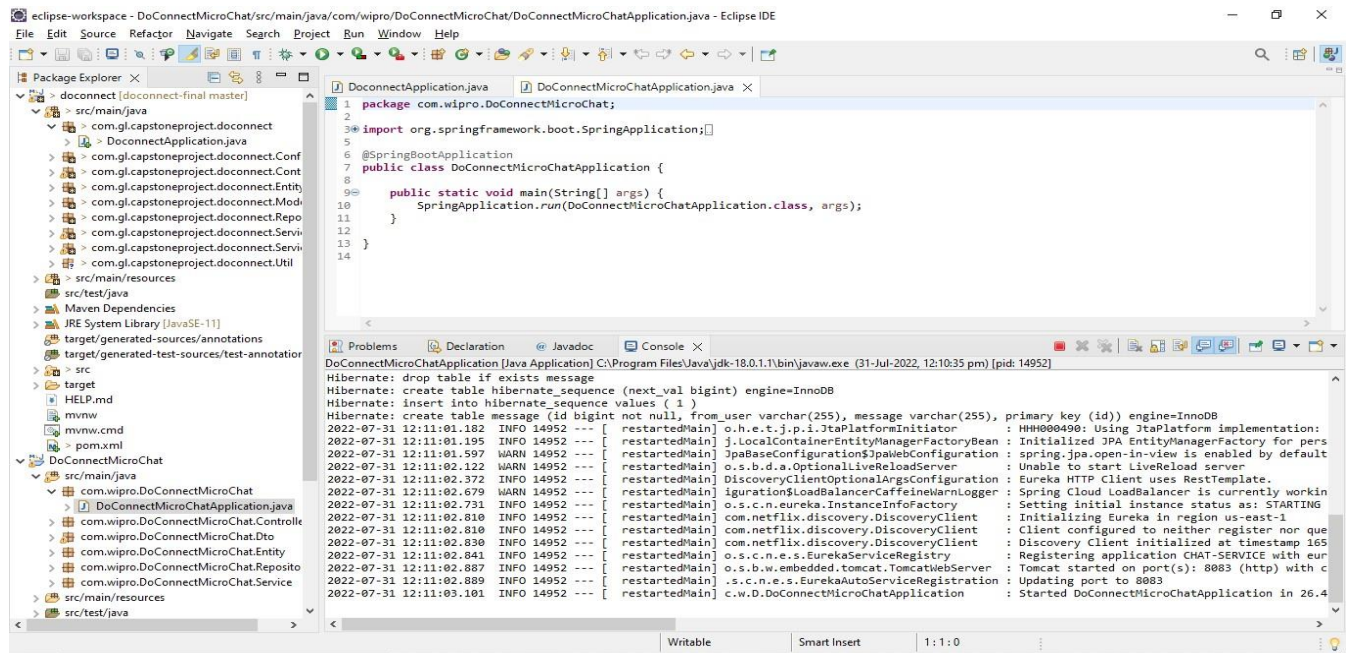


Fig 9.12 App Chat Function

10. CONCLUSION:

This project has been implemented by using Angular integrated with Spring Boot covering both the front end and the back end of the project. The application implements an Online Web Application where users can ask questions and chat with other users as per their own desires. The project DO CONNECT is developed for Users who have Q&A issues and they can easily Post their Question many times.