KNOWLEDGE SHARING SYSTEM

An Industry Oriented Project Report Submitted
In partial fulfillment of the requirements for the award of the degree of

Bachelor of Technology
in
Computer Science and Engineering

by

SAI RACHANA	**22N31A05L7**
AFROZ	**22N31A05M3**
NIHARIKA	**22N31A05P8**

Under the esteemed guidance of

Guide Name
Dr.N.Satheesh Kumar

Department of Computer Science and Engineering

Malla Reddy College of Engineering & Technology
(Autonomous Institution- UGC, Govt. of India)
(Affiliated to JNTUH, Hyderabad, Approved by AICTE, NBA &NAAC with 'A'Grade)
Maisammaguda, Kompally, Dhulapally, Secunderabad – 500100
website: www.mrcet.ac.in

Malla Reddy College of Engineering & Technology

(Autonomous Institution- UGC, Govt. of India)
(Affiliated to JNTUH, Hyderabad, Approved by AICTE, NBA &NAAC with 'A' Grade)
Maisammaguda, Kompally, Dhulapally,Secunderabad – 500100
website: www.mrcet.ac.in

CERTIFICATE

This is to certify that this is the bonafide record of the project entitled "Knowledge Sharing System", submitted by SAI RACHANA (22N31A05L7), AFROZ (22N31A05M3) and NIHARIKA (22N31A05P8) of B.Tech in the partial fulfillment of the requirements for the degree of Bachelor of Technology in Computer Science & Engineering, Department of CSE during the year 2023-2024.

Internal Guide

Dr.N.Satheesh Kumar
Asso. Professor

Head of the Department

Dr. S.Shanthi
Professor

External Examiner

ABSTRACT

The proposed system is a dynamic and user-friendly platform designed to foster a vibrant community of college students eager to share and enhance their knowledge. Functioning as a centralized hub, this platform provides a space for users to pose queries, seek assistance, and engage in collaborative learning. Drawing inspiration from popular platforms like Quora, our application is tailored specifically for the unique needs of college students. Users can effortlessly post questions, doubts, or academic-related inquiries within the application, making them visible to the entire community. This open and transparent environment encourages collaborative problem-solving, allowing anyone with relevant expertise to contribute thoughtful answers.

The platform's intuitive interface ensures that information flows seamlessly, fostering a sense of shared learning and cooperation. Beyond Q&A, our application also serves as a repository for academic works and research papers. Students can showcase their scholarly achievements, share insights from their studies, and contribute to the collective knowledge pool. This feature not only provides a platform for knowledge dissemination but also facilitates networking and collaboration among students with shared academic interests. In summary, our application offers a comprehensive solution for college students seeking a collaborative and supportive digital space. By combining the best features of knowledge-sharing platforms and academic repositories, we aim to create a vibrant community where learning transcends traditional boundaries, fostering a culture of shared knowledge and academic excellence.

TABLE OF CONTENTS

1. INTRODUCTION

1.1 DEFINITION AND SCOPE OF THE PROJECT

DEFINITION:

Knowledge Sharing System is a dynamic online platform designed to facilitate seamless knowledge sharing among users worldwide. It serves as a centralized repository where individuals can pose questions, share insights, and engage in collaborative learning across various domains and disciplines.

SCOPE OF THE PROJECT:

The scope of the Learn and Share project encompasses the development of a robust web-based system that fosters an environment conducive to knowledge exchange. Key components and features include:

- **User Registration and Authentication:** Implementing a secure user registration and authentication system to ensure the integrity of user accounts and protect sensitive information.

- **Question and Answer Functionality:** Providing users with the ability to ask questions on diverse topics and receive answers from a community of contributors. This feature includes mechanisms for categorizing questions, tagging topics, and facilitating discussions.

- **Content Creation and Sharing:** Enabling users to create and share informative content such as articles, tutorials, and resources to enrich the platform's knowledge base.

- **Search and Discovery:** Incorporating advanced search functionalities to allow users to efficiently discover relevant content and information within the platform.

- **Collaborative Tools:** Introducing collaborative tools such as chat functionalities through answering, or group discussions to encourage interaction and collaboration among users.

- **Accessibility and Usability**: Ensuring that the platform is accessible to users of all backgrounds and abilities, with a user-friendly interface that promotes ease of navigation and usability.

- **Security and Data Privacy:** Prioritizing data security and privacy measures to safeguard user information and mitigate risks associated with unauthorized access or data breaches.

The overarching goal of the Knowledge sharing System project is to empower individuals to share, discover, and collaborate on knowledge in a secure and user-friendly online environment, ultimately fostering lifelong learning and collective intelligence.

1.2 EXISTING AND PROPOSED SYSTEM

EXISTIN SYSTEM

- **Informal Knowledge Sharing:** Currently, users may rely on informal methods such as social media platforms, forums, and personal networks to seek and share knowledge. However, these methods lack structure and organization, making it challenging to find relevant information efficiently.

- **Limited Accessibility:** Existing knowledge-sharing platforms may have limitations in terms of accessibility, user interface design, and features, hindering the user experience and adoption rates.

- **Data Fragmentation:** Knowledge may be scattered across multiple platforms and sources, leading to fragmentation and difficulty in consolidating and accessing information effectively.

- **Security Concerns:** Users may have concerns about the security and privacy of their data when sharing sensitive information on public platforms, leading to reluctance in engaging in knowledge sharing activities.

PROPOSED SYSTEM

- **Structured Knowledge Sharing Platform**: The proposed system aims to provide a structured and organized platform for knowledge sharing, offering features such as categorized questions, content tagging, and search functionalities to enhance discoverability and accessibility.

- **User-Friendly Interface**: The user interface of the proposed system will be designed with usability and accessibility in mind, ensuring a seamless and intuitive experience for users across different devices and abilities.

- **Centralized Knowledge Repository:** The proposed system will serve as a centralized repository for knowledge, allowing users to access a diverse range of information from various domains and disciplines in one place.

- **Security Measures:** Security measures will be implemented to protect user data and ensure confidentiality, including encryption, secure authentication mechanisms, and data privacy policies compliant with relevant regulations.

By addressing the limitations of the existing system and introducing innovative features and enhancements, the proposed system aims to create a more effective, secure, and user-friendly environment for knowledge sharing, fostering collaboration, learning, and innovation among users.

2.SYSTEM ANALYSIS

2.1 HARDWARE AND SOFTWARE REQUIREMENTS

Hardware Requirements:

1. **Server Infrastructure:**

- Processor: Multi-core processor with sufficient processing power to handle concurrent user requests.
- Memory (RAM): At least 4GB RAM for smooth operation, but higher RAM capacity may be required based on the expected traffic and database size.
- Storage: Adequate storage space for storing application files, databases, and user-generated content. SSD storage is recommended for improved performance.
- Network Interface: Stable internet connection with sufficient bandwidth to support user interactions and data transfer.

2. **Database Server:**

- MongoDB Database Server: A server capable of hosting MongoDB, with enough storage capacity to accommodate the database and its collections. Ensure adequate memory and processing power to handle database operations efficiently.

3. **Web Server:**

- If deploying the application on a separate web server:
- Processor: Multi-core processor to handle incoming HTTP requests and serve web pages efficiently.
- Memory (RAM): At least 2GB RAM, but higher capacity may be necessary for handling concurrent connections and serving static and dynamic content.
- Storage: Sufficient storage space to host web application files, including HTML, CSS, JavaScript, and EJS templates.
- Network Interface: Stable internet connection with ample bandwidth to handle incoming and outgoing web traffic.

Software Requirements:

1. Operating System:

- Server: Linux-based operating system such as Ubuntu Server, CentOS, or Debian.
- Development: Windows, macOS, or Linux for development environment setup.

2. Web Server:

- Node.js: JavaScript runtime environment for server-side application logic.
- Express.js: Web application framework for Node.js, facilitating the development of web applications and APIs.
- EJS (Embedded JavaScript): Templating language for generating dynamic HTML content.
- Nginx or Apache (optional): Reverse proxy server for load balancing and handling HTTP requests if using a separate web server.

3. Database:

- MongoDB: NoSQL database for storing application data. Install and configure MongoDB server on the designated database server.
- Mongoose.js: MongoDB object modeling library for Node.js, providing a higher-level abstraction for interacting with MongoDB databases.

4. Development Tools:

- Text Editor or Integrated Development Environment (IDE): Choose a text editor or IDE such as Visual Studio Code, Sublime Text, or Atom for writing and editing code.
- Git: Version control system for tracking changes to codebase and collaboration with team members if applicable.
- Package Manager: npm (Node Package Manager) for installing and managing Node.js packages and dependencies.

5. Dependencies and Libraries:

- Various npm packages for handling authentication, session management, form validation, and other functionalities as required by the project specifications.

- Ensure that all software dependencies are up-to-date and compatible with the chosen versions to prevent compatibility issues and ensure smooth functioning of the application. Additionally, consider implementing security best practices and regularly updating software components to mitigate potential security risks.

3. SYSTEM DESIGN

3.1 UML DIAGRAMS

1. Class Diagram:

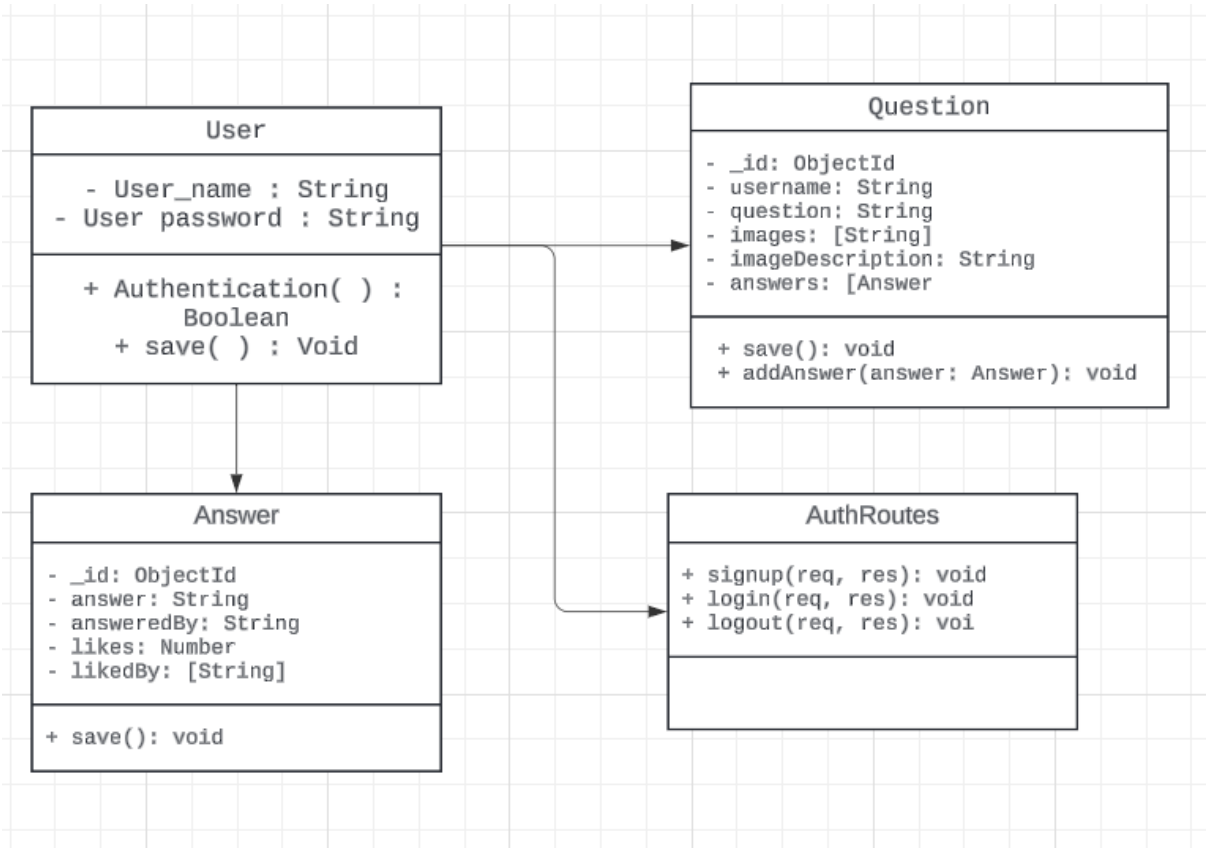
A class diagram provides a static view of the system, showing the system's classes, their attributes, methods, and the relationships among the classes.

Classes:

1. User
2. Question
3. Answer
4. AuthRoutes.

Associations:

1. User-Questions
2. User-Answer
3. User-AuthRoutes

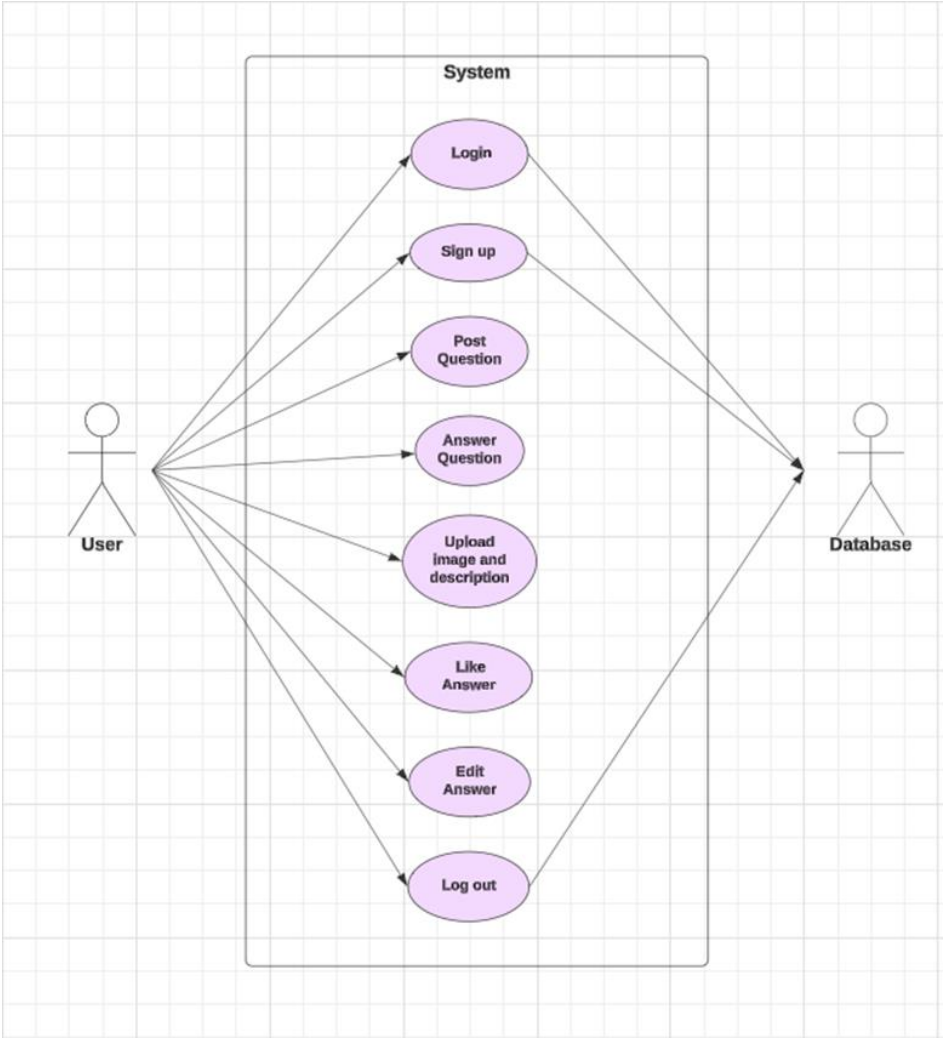


2. Use case Diagram:

Illustrates the functional requirements of the system by showing the interactions between users (actors) and the system. Key use cases include user signup/login, post question, post answer, search, like answer, edit answer, and view profile.

Use cases:

1. Login
2. Sign up
3. Post Question
4. Answer Question
5. Upload image and description
6. like answer
7. Edit Answer
8. Logout

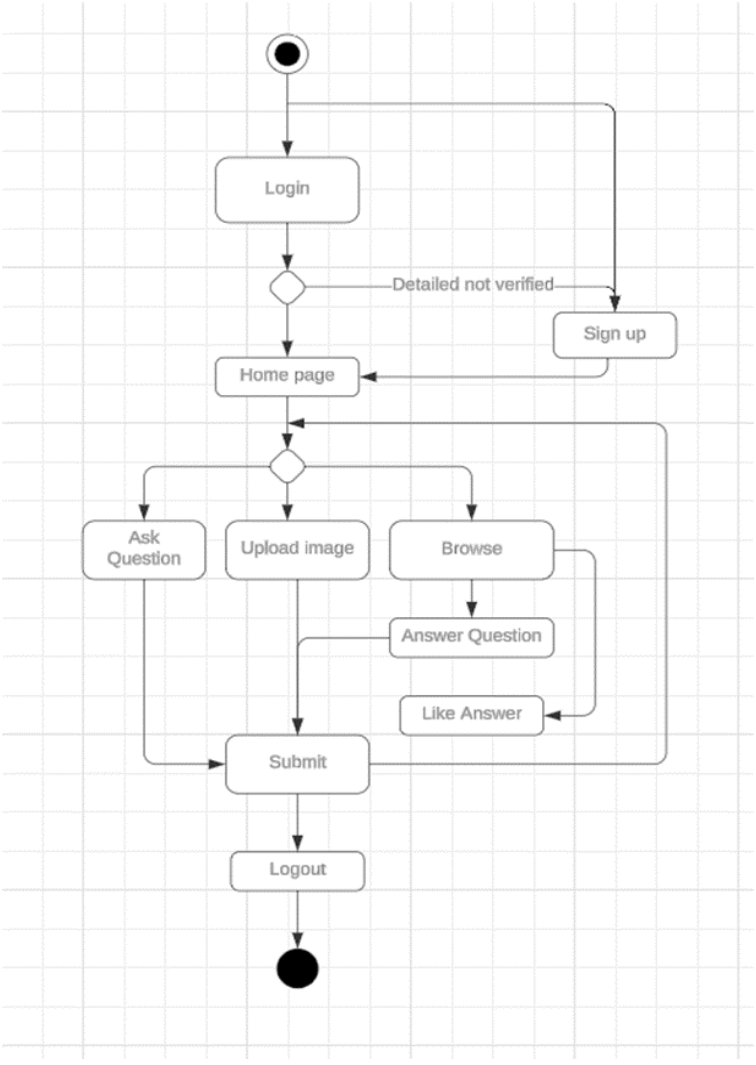


3. Activity Diagram:

- Depicts the dynamic flow of activities within the system. For example, it can show the process of posting a question, from logging in to submitting the question.

Steps followed:

1. User gets to login with the credentials (if exists), else user can sign up
2. Upon entering the detailed it directs to the home page.
3. User can Ask a Question, answer a Question, or Browse the content.
4. User log out from the website.

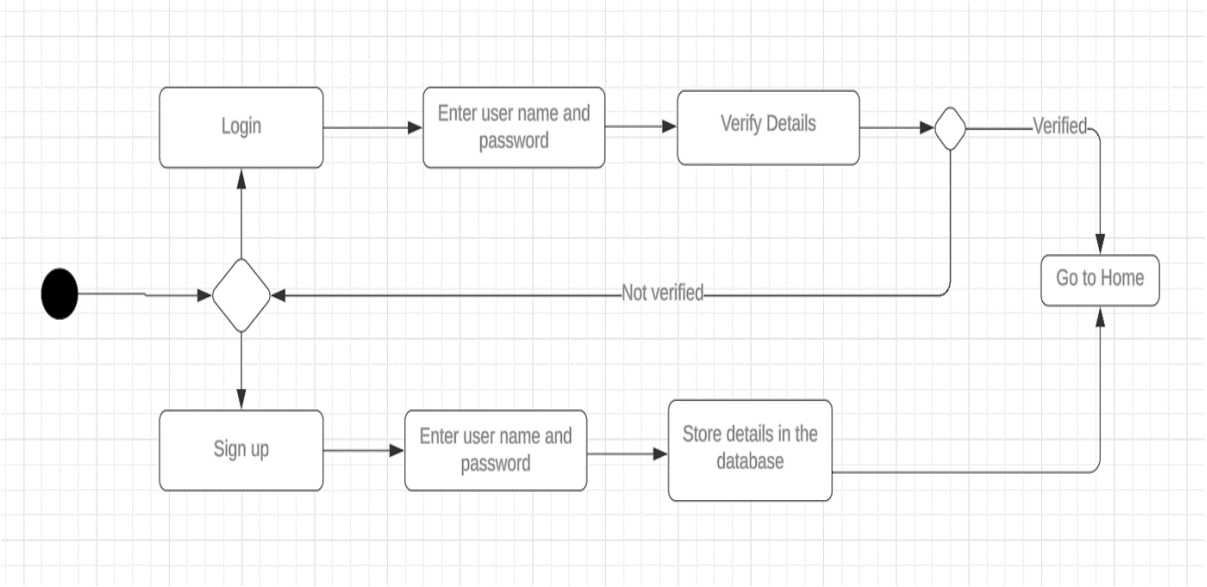


4. State chart Diagram:

State chart diagram is about how the system handles the logins of the user

Steps Followed:

1. User gets two options either to login or sign up
2. Upon login with the correct detailed, Home page is being displayed.
3. If details are incorrect, it displays an error message.
4. Else, when the user signs up the new entry is being made in the database.
5. And finally, it directs to the Home page.



5. Sequential Diagram:

The sequence diagram shows how objects interact in a particular scenario of a use case, focusing on the order of messages exchanged.

Sign Up:

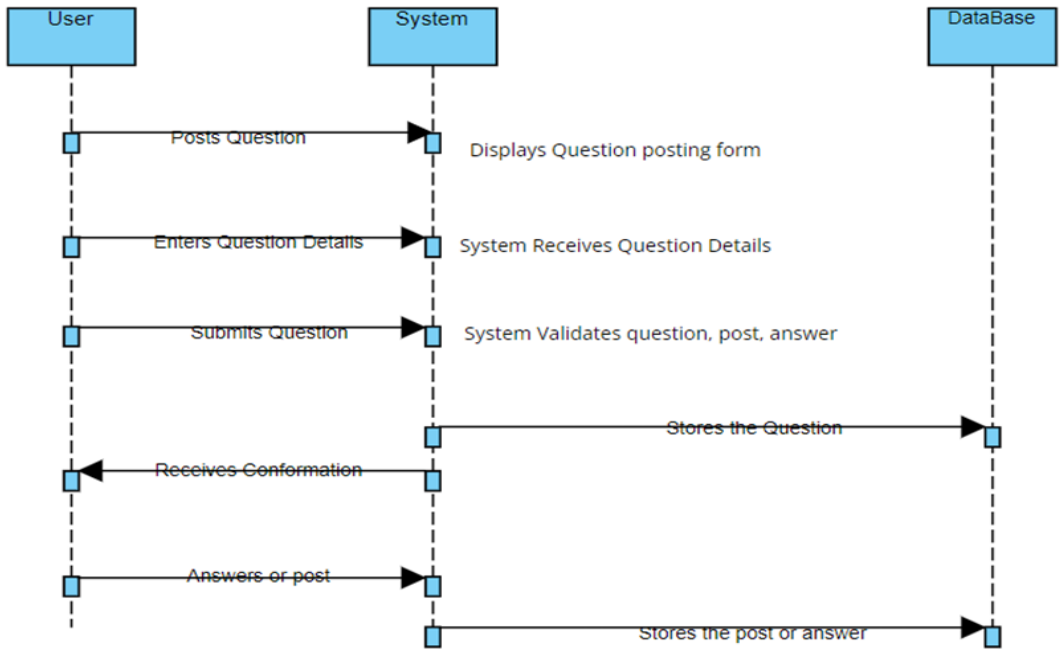
1. User sends sign up request.
2. System validates user data.
3. System saves user data.
4. System creates user session.
5. User receives confirmation and is redirected to home.

Post Question:

1. User sends question data.
2. System validates question content and images.
3. System saves question data.
4. System updates home page with new question.
5. User sees updated home page.

Answer Question:

1. User sends answer data.
2. System validates answer content.
3. System saves answer data.
4. System updates question with new answer.
5. User sees updated question with new answer.



4. IMPLEMENTATION

Ejs code for home page:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Home</title>
   <link rel="stylesheet" href="home.css">
   <script src="https://kit.fontawesome.com/4f58a2fa2b.js"
crossorigin="anonymous"></script>
</head>
<body>
   <div id="name">
      <h2><a style="background-color:#F45B69; border: #F45B69; color: #22181C; padding:
0;" href="/home">? Knowledge Sharing System</a></h2>
   </div>
   <div class="buttons">
      <div id="searchbar">
      <form action="/search" method="get">
         <input type="text" name="search" id="search" placeholder="Search.." required>
         <button type="submit"><i class="fa fa-search"></i></button>
      </form>

      </div>
   </div>
   <div id="logoutbt">
      <a href="/auth/logout" style="color: black; height: 12px;">Logout</a>
   </div>
   <div id="probut">
      <a href="/profile">Profile</a>
   </div>
   <div id="ask_que">
      <form action="/post" method="post" enctype="multipart/form-data">
         <input type="text" id="askquestion" name="askquestion" placeholder="Ask
Question..">
         <button type="submit">Submit</button> <br><br><br>
         <label for="questionImage">Upload an image:</label>
         <input type="file" id="questionImage" name="questionImage" accept="image/*">
         <input type="text" name="imageDescription" placeholder="Image Description
(optional)">
         <button type="submit">Submit</button>
      </form>
   </div>
   <div class="container">
      <h2>Questions : </h2>
      <% questions.forEach(question => { %>
```

```
        <div class="question-box">
          <p><strong>User:</strong> <%= question.username %></p>
          <p><strong>Question:</strong> <%= question.Question %></p>
          <% if (question.image) { %>
            <img src="<%= question.image %>" alt="Question Image" style="max-width:
200px;">
            <% if (question.imageDescription) { %>
              <p><strong>Description:</strong> <%= question.imageDescription %></p>
            <% } %>
          <% } %>
          <% if (question.answers && question.answers.length > 0) { %>
            <% question.answers.forEach(answer => { %>
              <div class="answer-box">
                <p><strong>Answer:</strong> <%= answer.answer %></p>
                <p><strong>By:</strong> <%= answer.answeredBy %></p>
                <p><strong>Likes:</strong> <%= answer.likes %></p>
                <form action="/like" method="post" style="display: inline;">
                  <input type="hidden" name="questionId" value="<%= question._id
%>">
                  <input type="hidden" name="answerId" value="<%= answer._id %>">
                  <button type="submit" style="background: none; border: none; color:
blue; cursor: pointer;">
                    <i class="fa-regular fa-thumbs-up"></i>
                  </button>
                </form>
                <% if (answer.answeredBy === session.username) { %>
                  <button onclick="toggleEditForm('<%= answer._id %>')">Edit</button>
                  <form id="edit-form-<%= answer._id %>" action="/edit-answer-inline"
method="post" style="display:none;">
                    <input type="hidden" name="questionId" value="<%= question._id
%>">
                    <input type="hidden" name="answerId" value="<%= answer._id
%>">
                    <textarea name="newAnswer" rows="4" cols="50"><%=
answer.answer %></textarea>
                    <button type="submit">Save</button>
                  </form>
                <% } %>
              </div>
            <% }) %>
          <% } else { %>
            <p>No answers yet.</p>
          <% } %>
          <form class="answer-form" action="/answer" method="post">
            <input type="hidden" name="questionId" value="<%= question._id %>">
            <input type="text" name="answer" placeholder="Your answer...">
            <button type="submit">Submit Answer</button>
          </form>
        </div>
      <% }) %>
```

```
    </div>

    <script>
        function toggleEditForm(answerId) {
            const form = document.getElementById('edit-form-' + answerId);
            if (form.style.display === 'none') {
                form.style.display = 'block';
            } else {
                form.style.display = 'none';
            }
        }
    </script>
</body>
</html>
```

Js code:

```
const express = require('express');
const mongoose = require('mongoose');
const path = require('path');
const session = require('express-session');
const multer = require('multer');
const User = require('./models/user');
const Question = require('./models/Question');
const authRoutes = require('./routes/authRoutes');
const port = 3000;

const app = express();

const storage = multer.diskStorage({
    destination: './uploads/',
    filename: function(req, file, cb) {
        cb(null, file.fieldname + '-' + Date.now() + path.extname(file.originalname));
    }
});

const upload = multer({
    storage: storage,
    limits: { fileSize: 1000000 }
}).single('questionImage');

app.use('/uploads', express.static('uploads'));
app.use(express.static(path.join(__dirname, 'public')));
app.use(express.urlencoded({ extended: true }));
app.use(express.json());
app.use(session({
    secret: 'your_secret_key',
    resave: false,
```

```
    saveUninitialized: true
}));

app.set('view engine', 'ejs');
app.set('views', path.join(__dirname, 'views'));

app.use('/auth', authRoutes);

function isAuthenticated(req, res, next) {
    if (req.session.username) {
        return next();
    } else {
        res.redirect('/');
    }
}

mongoose.connect('mongodb://localhost:27017/your_database_name')
    .then(() => {
        console.log('Connected to MongoDB');
    })
    .catch((error) => {
        console.error('Error connecting to MongoDB:', error.message);
    });

app.get('/profile', isAuthenticated, async (req, res) => {
    try {
        const username = req.session.username;
        const questions = await Question.find({ username });
        res.render('profile', { username, questions });
    } catch (error) {
        res.status(500).send('Error fetching profile data: ' + error.message);
    }
});

app.post('/post', (req, res) => {
    upload(req, res, async (err) => {
        if (err) {
            res.render('home', { msg: err });
        } else {
            try {
                const newQuestion = new Question({
                    username: req.session.username,
                    Question: req.body.askquestion || '', // Make question optional
                    image: req.file ? `/uploads/${req.file.filename}` : '',
                    imageDescription: req.body.imageDescription || '' // Default to empty string if no
description provided
                });
                await newQuestion.save();
                res.redirect('/home');
            } catch (err) {
```

```
            res.render('home', { msg: 'Error: Question could not be saved!' });
        }
    }
  });
});

app.post('/answer', async (req, res) => {
  const { questionId, answer } = req.body;
  try {
    const question = await Question.findById(questionId);
    const newAnswer = {
      answer: answer,
      answeredBy: req.session.username
    };
    question.answers.push(newAnswer);
    await question.save();
    res.redirect('/home');
  } catch (err) {
    res.status(500).send('Error saving answer: ' + err.message);
  }
});

app.post('/edit-answer-inline', async (req, res) => {
  const { questionId, answerId, newAnswer } = req.body;
  try {
    const question = await Question.findById(questionId);
    const answer = question.answers.id(answerId);
    if (answer.answeredBy === req.session.username) {
      answer.answer = newAnswer;
      await question.save();
      res.redirect('/home');
    } else {
      res.status(403).send('You are not authorized to edit this answer');
    }
  } catch (err) {
    res.status(500).send('Error editing answer: ' + err.message);
  }
});

app.post('/like', isAuthenticated, async (req, res) => {
  const { questionId, answerId } = req.body;
  try {
    const question = await Question.findById(questionId);
    if (!question) {
      return res.status(404).send('Question not found');
    }

    const answer = question.answers.id(answerId);
    if (!answer) {
      return res.status(404).send('Answer not found');
```

```
        }

        if (answer.likedBy.includes(req.session.username)) {
            return res.status(400).send('You have already liked this answer');
        }

        answer.likes += 1;
        answer.likedBy.push(req.session.username);
        await question.save();

        res.redirect('/home');
    } catch (err) {
        res.status(500).send('Error liking the answer: ' + err.message);
    }
});

app.get('/search', isAuthenticated, async (req, res) => {
    try {
        const query = req.query.search;
        const questions = await Question.find({
            Question: new RegExp(query, 'i')
        });
        res.render('home', { questions, session: req.session });
    } catch (err) {
        res.status(500).send('Error fetching search results: ' + err.message);
    }
});


app.get('/home', isAuthenticated, async (req, res) => {
    try {
        const questions = await Question.find({});
        res.render('home', { questions, session: req.session });
    } catch (err) {
        res.status(500).send('Error fetching questions: ' + err.message);
    }
});

app.get('/', (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'index.html'));
});

app.listen(port, () => {
    console.log(`Server is running on port ${port}`);
});
```

Ejs code for profile:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Your Profile</title>
   <link rel="stylesheet" href="profile.css">
   <link rel="preconnect" href="https://fonts.googleapis.com">
   <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
   <link
href="https://fonts.googleapis.com/css2?family=Caveat:wght@400..700&display=swap"
rel="stylesheet">
</head>
<body>
   <div id="profilebox">
      <div id="goback">
         <a href="/home" style="float: right; border: #22181C;">Back</a>
      </div>
      <img id="img" src="/assets/profile.jpg" alt="Profile Image">
      <p style="color: #F45B69"><b>Your profile</b></p><br>
      <div id="hello" style="background-color: #F45B69; height: 130px; text-align: center;">
         <img style="height: 100px; width: 100px; margin-left: 150px; margin-top: 15px;
float: left;" src="/assets/hi.png" alt="hi">
         <pre style="font-family: Caveat Brush, cursive;font-style: normal; font-size:xx-
large;margin-inline: 20px;">
            <b>Hello there., <%= username %></b></pre>
         <pre style="font-size: large;">                    Thank you for using our Website.</pre>
      </div>

      <p><br><br><b style="color: #F45B69;">Your Questions:</b></p>
      <ul>
         <% questions.forEach(question => { %>
            <li>
               <p><strong>Question:</strong> <%= question.Question %></p>
               <% if (question.image) { %>
                  <img src="<%= question.image %>" alt="Question Image" style="max-width:
200px;">
               <% } %>
               <% if (question.answers && question.answers.length > 0) { %>
                  <% question.answers.forEach(answer => { %>
                     <div class="answer-box">
                        <p><strong>Answer:</strong> <%= answer.answer %></p>
                        <p><strong>By:</strong> <%= answer.answeredBy %></p>
                        <p><strong>Likes:</strong> <%= answer.likes %></p>
                        <% if (answer.likedBy.includes(username)) { %>
                           <span><i class="fa-solid fa-thumbs-up"></i> <%= answer.likes
%></span>
                        <% } else { %>
                           <span><i class="fa-regular fa-thumbs-up" onclick="likeAnswer('<%=
answer._id %>')"></i> <%= answer.likes %></span>
```

```
                        <% } %>
                        <% if (answer.answeredBy === username) { %>
                            <button onclick="toggleEditForm('<%= answer._id
%>')">Edit</button>
                            <form id="edit-form-<%= answer._id %>" action="/edit-answer-
inline" method="post" style="display:none;">
                                <input type="hidden" name="questionId" value="<%= question._id
%>">
                                <input type="hidden" name="answerId" value="<%= answer._id
%>">
                                <textarea name="newAnswer" rows="4" cols="50"><%=
answer.answer %></textarea>
                                <button type="submit">Save</button>
                            </form>
                        <% } %>
                    </div>
                <% }) %>
            <% } else { %>
                <p>No answers yet.</p>
            <% } %>
            <p>.....</p>
        </li>
    <% }) %>
  </ul>
 </div>
 <script>
  function toggleEditForm(answerId) {
    const form = document.getElementById('edit-form-' + answerId);
    if (form.style.display === 'none') {
      form.style.display = 'block';
    } else {
      form.style.display = 'none';
    }
  }

  function likeAnswer(answerId) {
    fetch('/like', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json'
      },
      body: JSON.stringify({ answerId })
    }).then(response => {
      if (response.ok) {
        location.reload();
      }
    });
  }
 </script>
</body>
```

```
</html>
```

User.js file (to handle user details in database):

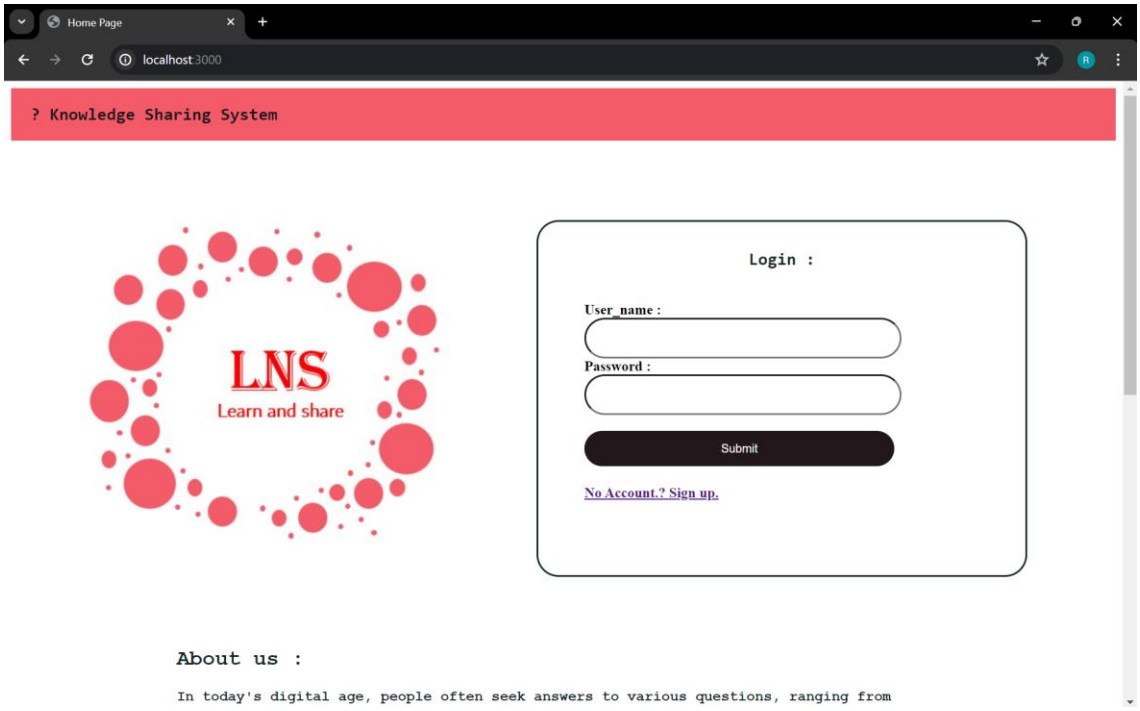
```
const mongoose = require('mongoose');

const userSchema = new mongoose.Schema({
   username: { type: String, required: true },
   password: { type: String, required: true }
});

// Check if the model is already defined to prevent overwrite error
const User = mongoose.models.User || mongoose.model('User', userSchema);

module.exports = User;
```

Question.js file (to handle Question storing in database):

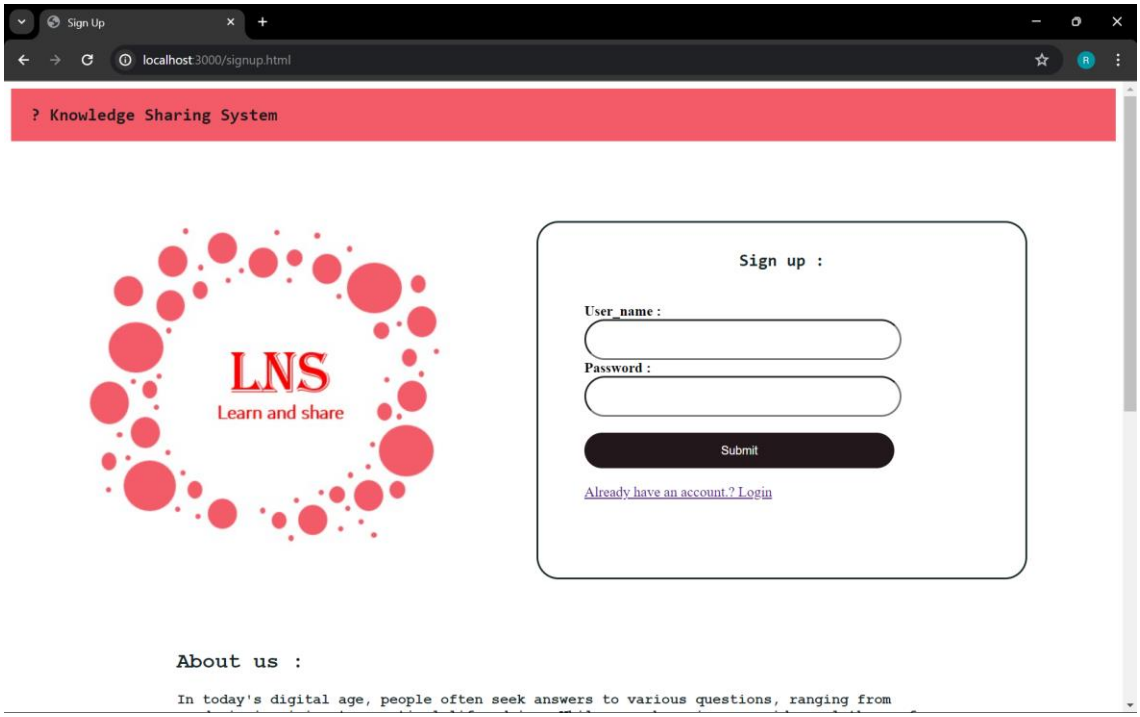
```
const mongoose = require('mongoose');

const answerSchema = new mongoose.Schema({
   answer: String,
   answeredBy: String,
   likes: { type: Number, default: 0 },
   likedBy: [String]
});

const questionSchema = new mongoose.Schema({
   username: String,
   Question: { type: String, required: true },
   image: String,
   imageDescription: String,
   answers: [answerSchema]
});

// Create an index on the Question field for faster searches
questionSchema.index({ Question: 'text' });

const Question = mongoose.model('Question', questionSchema);

module.exports = Question;
```

4.2 OUTPUT SCREENS

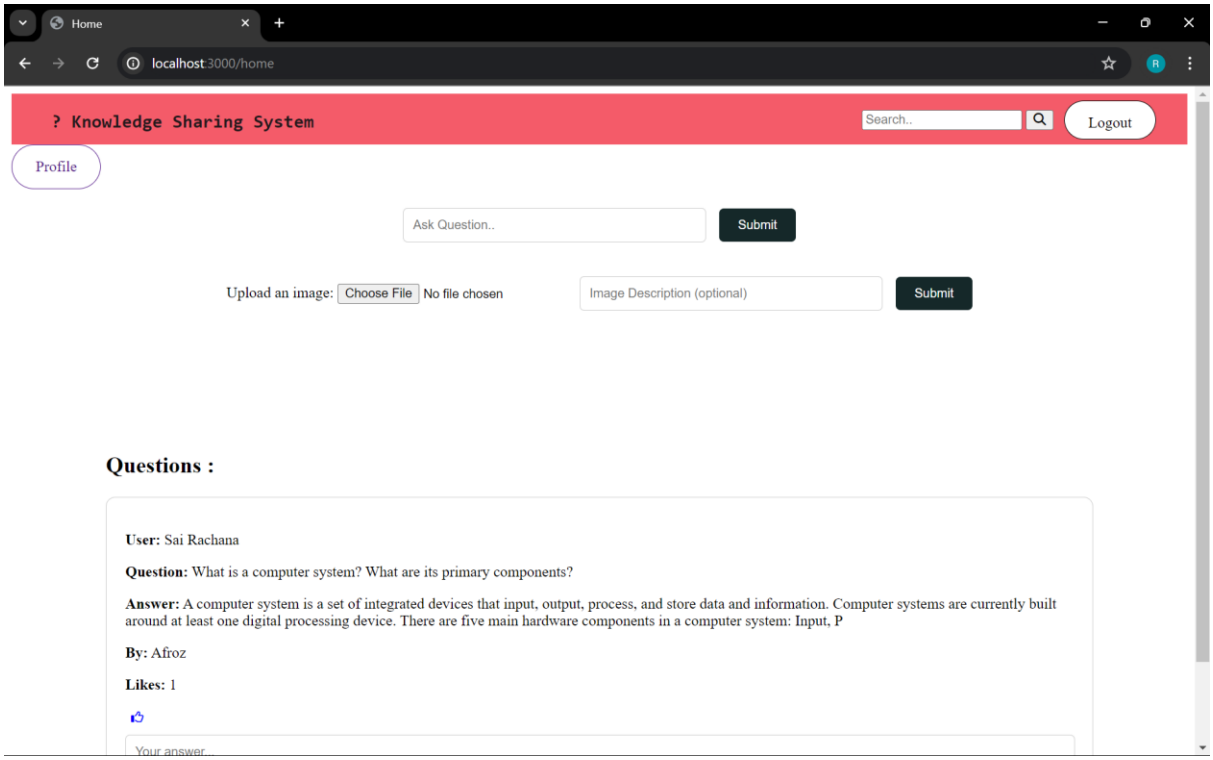
Login page:

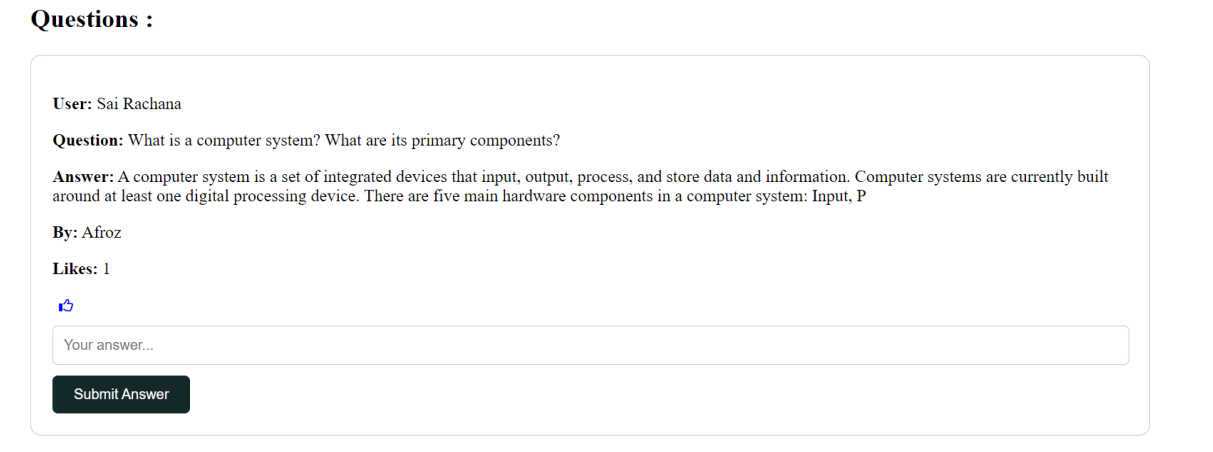


Sign-up page

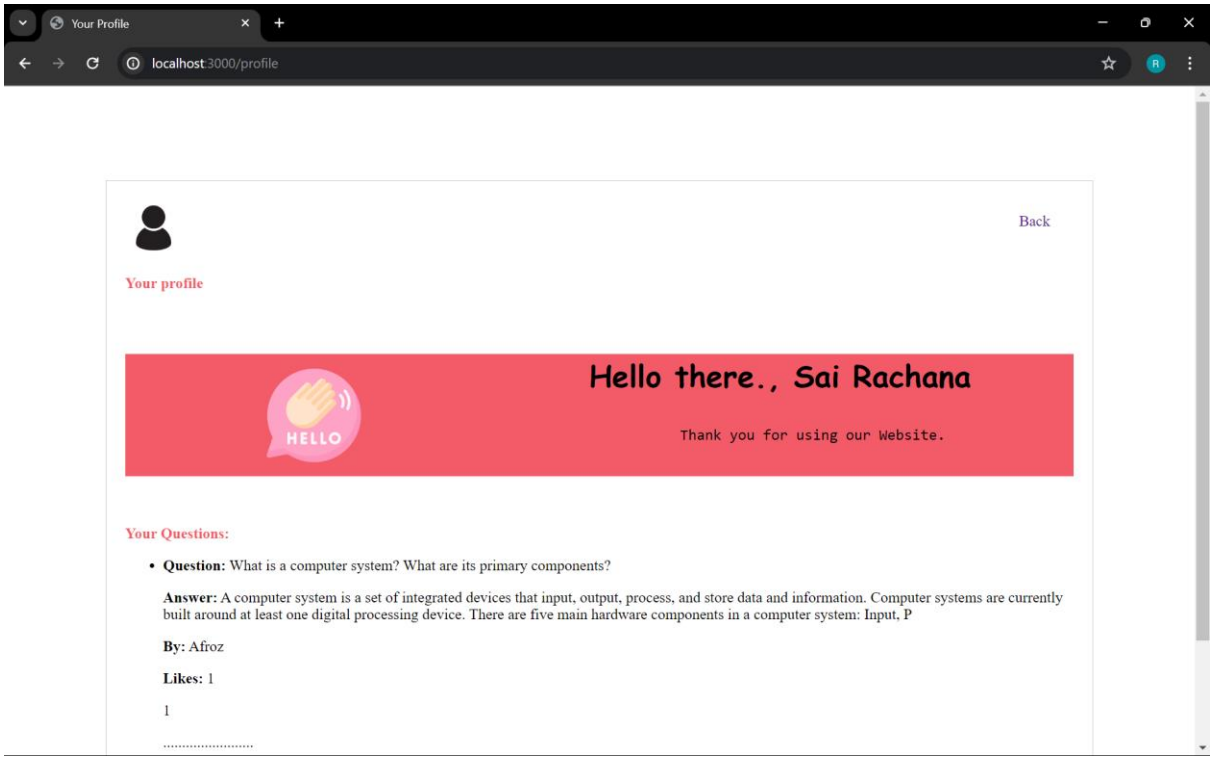


Home page :


Home
localhost:3000/home
? Knowledge Sharing System
Search..
Logout
Profile
Ask Question..
Submit
Upload an image:
Choose File
No file chosen
Image Description (optional)
Submit
Questions :
User: Sai Rachana
Question: What is a computer system? What are its primary components?
Answer: A computer system is a set of integrated devices that input, output, process, and store data and information. Computer systems are currently built around at least one digital processing device. There are five main hardware components in a computer system: Input, P
By: Afroz
Likes: 1
Your answer...


Questions :
User: Sai Rachana
Question: What is a computer system? What are its primary components?
Answer: A computer system is a set of integrated devices that input, output, process, and store data and information. Computer systems are currently built around at least one digital processing device. There are five main hardware components in a computer system: Input, P
By: Afroz
Likes: 1
Your answer...
Submit Answer

Profile page :



5. Conclusion

The Knowledge Sharing System project successfully implements a web-based platform where users can share knowledge by asking questions and providing answers. The project leverages modern web technologies and follows a robust architectural pattern to ensure a seamless user experience. Here are the key takeaways and accomplishments of the project:

1. User Authentication and Session Management:
   - The system allows users to securely sign up, log in, and log out. Passwords are hashed using bcrypt to enhance security.
   - User sessions are managed efficiently to keep users logged in across different pages.

2. Question and Answer Functionality:
   - Users can post questions with images, which are stored and retrieved from the server using Multer for file handling.
   - Users can provide answers to questions, view all answers, and like answers to indicate their usefulness.

3. Dynamic Content Rendering:
   - The application uses EJS templating to dynamically render content on the front-end, providing a responsive and interactive user interface.
   - Users can edit their answers inline, enhancing the usability of the system.

4. Keyword Search:
   - The search functionality allows users to find relevant questions by entering keywords, improving the discoverability of information within the system.

5. Backend and Database Integration:
   - The project is built on Express.js, a robust backend framework that handles routing, middleware, and session management.
   - MongoDB is used as the database to store user information, questions, and answers, enabling efficient data retrieval and manipulation.

6. File Handling and Image Uploads:
   - The integration of Multer enables users to upload multiple images with their questions, enhancing the clarity and context of the questions posted.

7. Interactive User Interface:
   - The use of HTML, CSS, and JavaScript provides a user-friendly interface. Features like toggling edit forms and liking answers are implemented using JavaScript to offer a smooth and engaging user experience.

BIBLIOGRAPHY

During the development of the Knowledge Sharing System project, a variety of resources were utilized to gather information, learn new concepts, and solve problems effectively. Key references included W3Schools, which provided comprehensive tutorials and references on HTML, CSS, JavaScript, and various other web technologies, essential for building the front-end of the application. GeeksforGeeks offered detailed articles and code examples on a wide range of programming topics, particularly beneficial for backend development involving Express.js, MongoDB, and Multer for handling file uploads. YouTube was a valuable source for video tutorials and walkthroughs, offering in-depth explanations and visual demonstrations of web development concepts, facilitating the implementation of complex features. ChatGPT, an AI model developed by OpenAI, provided real-time guidance, code suggestions, and explanations on various aspects of the project, aiding in debugging issues and optimizing code. Mozilla Developer Network (MDN Web Docs) offered detailed documentation and examples for web technologies, ensuring the use of modern standards and best practices. The official Express.js documentation provided crucial guides and API references for server setup, routing, and middleware, while MongoDB documentation offered extensive information on database setup, schema design, and querying, vital for the backend development. Additionally, Multer documentation provided detailed instructions on configuring and using Multer for file uploads in an Express.js application.

Collectively, these resources were instrumental in the successful completion of the Knowledge Sharing System project, and the assistance from the OpenAI platform was particularly invaluable. The contributions from various community tutorials, blog posts, and open-source code also played a significant role in the project's development.