MEAN: FULL STACK WEB DEVELOPMENT LABRATORY

PROJECT REPORT

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE AWARD OF THE DEGREE OF

BACHELOR OF TECHNOLOGY

(Information Technology)

JAN-MAY, 2025

SUBMITTED BY:

NAME: Devesh Sharma
UNIVERSITY ROLL NO: 2203818

NAME: Mayank
UNIVERSITY ROLL NO: 2203855

DEPARTMENT OF INFORMATION TECHNOLOGY

GURU NANAK DEV ENGINEERING COLLEGE LUDHIANA

(An Autonomous College Under UGC ACT)

ACKNOWLEDGMENT

I would like to express my sincere gratitude to my project guide, Dr. Palwinder Kaur, for her invaluable guidance and support throughout this project. Her expertise in web development and MEAN stack technologies has been instrumental in shaping this work and enhancing my understanding of modern web application architecture.

I am deeply thankful to Guru Nanak Dev Engineering College for providing the necessary resources and infrastructure that made this project possible. The emphasis on practical learning and modern technologies has greatly contributed to the successful completion of this work.

Finally, I would like to thank my family and friends for their constant encouragement and support throughout my academic journey.

Contents

1 Introduction

This project implements a modern web-based AI chat application that integrates with Ollama's local Large Language Models (LLMs) while featuring user authentication and chat history persistence. The application follows the MEAN stack architecture: MongoDB for the database, Express.js for the backend server, AngularJS for the frontend interface, and Node.js as the runtime environment.

The application allows users to register, log in, create new chat sessions, and continue previous conversations with the AI. All user information and chat history are stored securely in MongoDB, providing a persistent and personalized experience. By leveraging the power of modern web technologies, this project demonstrates the practical application of full-stack development principles in creating a functional and user-friendly AI chat interface.

2 Objectives

- **To** implement a full-stack web application using the MEAN stack architecture
- **To** integrate Ollama's local LLM capabilities for AI-powered conversations
- **To** implement a secure user authentication system with session management
- **To** create a persistent chat history system with MongoDB
- **To** allow users to manage their chat sessions (create, load, update, delete)
- **To** develop a responsive and intuitive user interface using AngularJS

3 Technologies Used

- **Database:** MongoDB for data persistence
- **Backend:** Node.js with Express.js framework
- **Frontend:** AngularJS for the user interface
- **Authentication:** bcrypt for password hashing, express-session for session management

- **AI Integration:** Ollama API for LLM interactions
- **Additional Libraries:**
  - connect-mongo for session storage
  - axios for HTTP requests
  - mongoose for MongoDB object modeling
- **Development Tools:** Visual Studio Code, Git

4 System Architecture

The application follows the MEAN stack architecture with four main components:

- **MongoDB Database:** Stores user information and chat history
- **Express.js Backend:** Provides API endpoints for authentication, chat management, and AI interaction
- **AngularJS Frontend:** Delivers the user interface and client-side application logic
- **Ollama API:** External service that provides LLM capabilities

4.1 Database Schema

The application uses two main MongoDB collections:

4.1.1 User Schema

Stores user credentials and authentication information:

```
const UserSchema = new mongoose.Schema({
  username: {
    type: String,
    required: true,
    unique: true,
    trim: true,
  },
  password: {
    type: String,
    required: true,
  },
  createdAt: {
    type: Date,
    default: Date.now,
  }
});
```

4.1.2 Chat Schema

Stores chat sessions and message history:

```
const MessageSchema = new mongoose.Schema({
  role: {
    type: String,
    enum: ["user", "assistant"],
    required: true,
  },
  content: {
    type: String,
    required: true,
  },
  timestamp: {
    type: Date,
    default: Date.now,
  }
});

const ChatSchema = new mongoose.Schema({
  userId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: "User",
    required: true,
  },
  title: {
    type: String,
    default: function() {
      return "Chat " + this.createdAt.toLocaleDateString();
    }
  },
  messages: [MessageSchema],
  createdAt: {
    type: Date,
    default: Date.now,
  },
  updatedAt: {
    type: Date,
    default: Date.now,
  }
});
```

5 Implementation Details

5.1 Authentication System

The application implements a secure authentication system using bcrypt for password hashing and express-session for session management.

5.1.1 User Registration

The registration endpoint handles new user creation with password hashing:

```
router.post("/register", async (req, res) => {
  try {
    const { username, password } = req.body;

    // Check if user exists & hash password
    const existingUser = await User.findOne({ username });
    if (existingUser) {
      return res.status(400).json({ error: "Username already exists" });
    }

    const hashedPassword = await bcrypt.hash(password, 10);

    // Create user & start session
    const user = new User({ username, password: hashedPassword });
    await user.save();
    req.session.userId = user._id;

    res.json({ success: true, username: user.username });
  } catch (err) {
    res.status(500).json({ error: "Registration failed" });
  }
});
```

5.1.2 User Login

The login endpoint authenticates users and establishes sessions:

```
router.post("/login", async (req, res) => {
  try {
    const { username, password } = req.body;

    // Find user & verify password
    const user = await User.findOne({ username });
    if (!user) return res.status(400).json({ error: "Invalid
       credentials" });

    const isMatch = await bcrypt.compare(password, user.password);
    if (!isMatch) return res.status(400).json({ error: "Invalid
       credentials" });

    // Set session and return user info
    req.session.userId = user._id;
    res.json({ success: true, username: user.username });
  } catch (err) {
    res.status(500).json({ error: "Login failed" });
  }
});
```

5.2 Chat Management System

The application provides complete CRUD operations for chat management:

5.2.1 Creating a New Chat

This endpoint allows authenticated users to create new chat sessions:

```
router.post("/", isAuthenticated, async (req, res) => {
  try {
    const { title, messages } = req.body;
    const chat = new Chat({
      userId: req.session.userId,
      title: title || undefined,
      messages: messages || []
    });
    await chat.save();
    res.json(chat);
  } catch (err) {
    res.status(500).json({ error: "Failed to create chat" });
  }
});
```

5.2.2 Loading Existing Chats

This endpoint retrieves a list of a user's previous chat sessions:

```
router.get("/", isAuthenticated, async (req, res) => {
  try {
    // Find user's chats, select minimal fields, sort by recent first
    const chats = await Chat.find({ userId: req.session.userId })
      .select("_id title updatedAt")
      .sort({ updatedAt: -1 });
    res.json(chats);
  } catch (err) {
    res.status(500).json({ error: "Failed to get chats" });
  }
});
```

5.3 Integration with Ollama API

The application integrates with Ollama's API to generate responses from the LLM and automatically saves responses to the user's chat history:

```
app.post("/api/chat", async (req, res) => {
  try {
    // Send user messages to Ollama API
    const response = await axios.post("http://localhost:11434/api/chat",
       {
      model: "llama3.2:3b",
      messages: req.body.messages,
      stream: false,
    });

    // Format AI response
    const aiMessage = {
      role: "assistant",
      content: response.data.message?.content || response.data.response,
    };

    // Save to chat history if user is authenticated
    if (req.session.userId && req.body.chatId) {
      const Chat = require("./models/Chat");
      const chat = await Chat.findOne({
        _id: req.body.chatId,
        userId: req.session.userId
      });

      if (chat) {
        chat.messages.push(aiMessage);
        chat.updatedAt = Date.now();
        await chat.save();
      }
    }

    res.json({ message: aiMessage });
  } catch (err) {
    console.error(err);
    res.status(500).json({ error: "Error connecting to Ollama" });
  }
});
```

6 Features

- **User Authentication:** Secure registration and login system
- **Session Management:** Persistent user sessions with MongoDB storage
- **Chat Creation:** Ability to start new chat conversations
- **Chat History:** Saving and loading previous conversations
- **Chat Management:** Updating chat titles and deleting unwanted chats
- **AI Integration:** Seamless interaction with Ollama's LLM
- **Responsive UI:** User-friendly interface with AngularJS

7 Key Components

7.1 Frontend Interface

The application features a responsive and intuitive user interface built with AngularJS that adapts based on authentication state:

```
<!-- Main application container with conditional views -->
<div class="app-container">
    <!-- Header with user info when logged in -->
    <div class="header">
        <h1>Ollama Chat</h1>
        <div class="user-info" ng-if="chat.user">
            <span>Welcome, {{chat.user.username}}!</span>
            <button ng-click="chat.logout()">Logout</button>
        </div>
    </div>
```

```html

    <!-- Authentication section shown when not logged in -->
    <div class="auth-container" ng-if="!chat.user">
        <div class="auth-tabs">
            <div class="auth-tab" ng-class="{'active': chat.authTab ===
                'login'}">Login</div>
            <div class="auth-tab" ng-class="{'active': chat.authTab ===
                'register'}">Register</div>
        </div>
        <!-- Login/Register forms would be here -->
    </div>

    <!-- Main chat interface shown when logged in -->
    <div class="main-content" ng-if="chat.user">
        <!-- Sidebar with chat history -->
        <div class="sidebar">
            <button ng-click="chat.newChat()">New Chat</button>
            <ul class="chat-list">
                <li ng-repeat="chatItem in chat.chatList"
                    ng-click="chat.loadChat(chatItem._id)">
                    {{chatItem.title}}
                </li>
            </ul>
        </div>

        <!-- Chat area with messages -->
        <div class="chat-container">
            <!-- Message display and input would be here -->
        </div>
    </div>
</div>
```

7.2 AngularJS Controller

The application's frontend logic is managed by an AngularJS controller that handles authentication, chat loading, and message sending:

```
angular.module("chatApp", []).controller("ChatController", function
    ($http) {
  var vm = this;

  // State variables
  vm.messages = [];
  vm.input = "";
  vm.loading = false;
  vm.user = null;
  vm.authTab = 'login';
  vm.chatList = [];
  vm.currentChat = null;

  // Core functionality
  vm.sendMessage = function() {
    if (!vm.input.trim()) return;

    // Add user message to display
    var userMsg = { role: "user", content: vm.input.trim() };
    vm.messages.push(userMsg);
    vm.input = "";
    vm.loading = true;

    // Send to API with current chat ID if available
    var reqData = { messages: vm.messages };
    if (vm.currentChat) reqData.chatId = vm.currentChat._id;

    $http.post("/api/chat", reqData)
      .then(function(response) {
        vm.messages.push(response.data.message);
        // Save updated chat if needed
        if (vm.currentChat) vm.updateChatTitle();
      })
      .finally(function() { vm.loading = false; });
  };

  // Initialize by checking auth status
  vm.checkAuthStatus();
});
```

8 Deployment Process

The source code is available on GitHub at https://github.com/mayank0304/mean-mini-project.

1. Install Node.js and MongoDB on the target system
2. Clone the repository containing the application code
3. Install dependencies:

   ```
   npm install
   ```

4. Ensure MongoDB is running:

   ```
   mongod --dbpath=/path/to/data/directory
   ```

5. Install and start Ollama service with the required model:

   ```
   ollama run llama3.2:3b
   ```

6. Start the application:

   ```
   node server.js
   ```

7. Access the application at http://localhost:3000

9 Challenges and Solutions

- **Challenge:** Integrating MongoDB with Express for session storage
- **Solution:** Used connect-mongo package to seamlessly store sessions in MongoDB
- **Challenge:** Implementing secure authentication
- **Solution:** Utilized bcrypt for password hashing and express-session for session management
- **Challenge:** Managing real-time chat updates
- **Solution:** Implemented efficient state management in AngularJS controller to maintain chat state
- **Challenge:** Synchronizing chat history between database and frontend
- **Solution:** Developed a robust API to handle create, read, update, and delete operations for chat sessions
- **Challenge:** Handling asynchronous API calls to Ollama
- **Solution:** Used async/await with proper error handling to ensure reliable communication

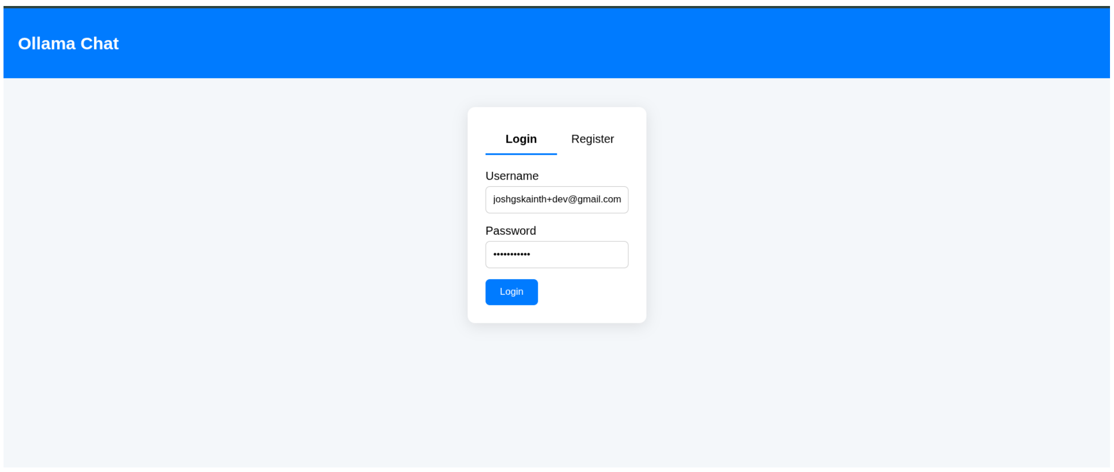
10 Future Enhancements

- **User Profile Management:** Allow users to update their profile information and preferences
- **Multiple AI Model Selection:** Enable users to choose from different Ollama models
- **Chat Categories and Tags:** Implement a system to categorize and tag chat sessions
- **Advanced Authentication:** Add features like password reset, email verification, and social login

- **Real-time Collaboration:** Enable shared chat sessions between multiple users
- **Enhanced UI Features:** Implement message formatting, file sharing, and code highlighting
- **Analytics Dashboard:** Provide users with insights into their chat history and usage patterns

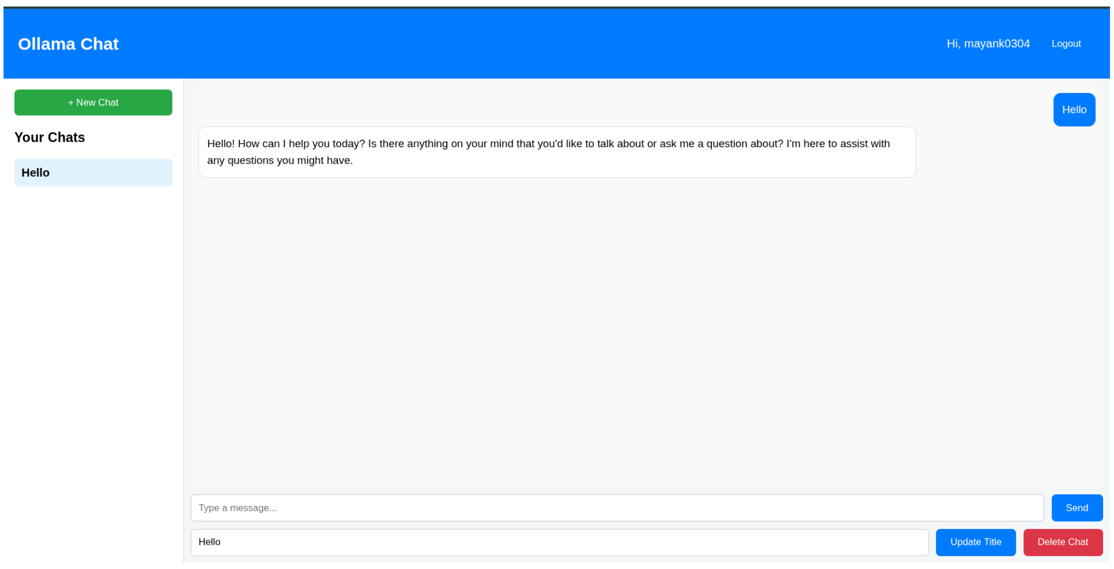
11 Application Screenshots

This section presents key screenshots of the application interface to demonstrate its functionality and user experience.



joshgskainth+dev@gmail.com

Figure 1: Login and Registration Screen



Hello

Hello! How can I help you today? Is there anything on your mind that you'd like to talk about or ask me a question about? I'm here to assist with any questions you might have.

Hello

Figure 2: Main Chat Interface with Sidebar

12 Conclusion

This project successfully demonstrates the implementation of a full-stack web application using the MEAN stack architecture. The integration of MongoDB provides a robust foundation for user authentication and chat history persistence, while Express.js delivers efficient API endpoints for both client-server communication and integration with the Ollama LLM service.

The application showcases several key aspects of modern web development:

- Effective use of NoSQL database design for flexible data storage
- Implementation of secure authentication and session management
- Creation of a responsive and intuitive user interface with AngularJS
- Integration with external AI services through RESTful APIs
- Complete CRUD operations for managing persistent user data

The project provides a solid foundation for further development and enhancement of AI-powered chat applications, demonstrating the capabilities of the MEAN stack in creating modern, interactive web experiences.