MAJOR PROJECT REPORT

(230NMCR-753)

BY

RAJDEEP MITRA

(O23MCA110396)

In the partial fulfilment of requirements for the award of degree in Masters of Computer Application

(Batch from 2023-2025)

Under the Guidance of

CHANDIGARH UNIVERSITY

NH-05 Chandigarh-Ludhiana Highway,
Mohali, Punjab (INDIA)

CERTIFICATE

This is to certify that Mr Rajdeep Mitra bearing Registration No. O23MCA110396 of Centre for Distance & Online Learning, Chandigarh University has worked on the project entitled “SpeakMate - A Language Learning” Platform under the supervision of Mr/Mrs _____ of Chandigarh University. The project was carried out from May 7th to May 30th.

The project is hereby accepted by the School of Information Technology, Chandigarh University, in partial fulfilment of the requirements for the award of Degree in Master of Computer Application.

DECLARATION

I hereby declare that the work recorded in this project report entitled “SpeakMate - A Language Learning Platform”, in partial fulfilment for the requirements for the award of Degree in Master of Computer Application from Chandigarh University, is a faithful and Bonafide work carried out under the supervision and guidance of Mr/Mrs _____ from May 7th to May 30th. The results of this investigation reported in this project have so far not been reported for any other Degree / Diploma or other technical forum. DECLARATION The assistance and help received during the course of the investigation have been duly acknowledged.

Name: Rajdeep Mitra

Registration No.: O23MCA110396

ACKNOWLEDGMENT

I would like to sincerely thank my project guide, Mr/Mrs_____, Chandigarh University for their invaluable support, expert guidance, and encouragement throughout the duration of this project. Their insightful feedback and constructive suggestions played a crucial role in the successful completion of this work.

I am also grateful to the faculty members of the Master of Computer Application at Chandigarh University whose teachings and resources provided a strong foundation for undertaking this project.

Finally, I express my heartfelt appreciation to my family for their unwavering support, patience, and motivation during the development of this project.

ABSTRACT

SpeakMate is a web-based language learning platform designed to connect individuals who share a common interest in learning new languages. The platform encourages peer-to-peer communication through real-time chat rooms, allowing users to practice and improve their language skills with fellow learners. A key feature of SpeakMate is its integration of an AI-powered chatbot, which provides instant assistance for grammar, vocabulary, and general language-related queries.

In addition to its social and AI-driven elements, SpeakMate offers personalized tutorials based on the language selected by each user. These tutorials are designed to cater to different learning levels and styles, making the learning process more engaging and efficient. The platform is built using modern technologies including React.js for the frontend, Express.js and Node.js for the backend, and Gemini's GPT for intelligent responses.

SpeakMate aims to make language learning more interactive, collaborative, and accessible, providing users with both human and AI support to enhance their fluency and confidence in a new language.

TABLE OF CONTENTS

NO.	TITLE	PAGE NO.
1.	Title Page	1 - 1
2.	Certificate	2 - 2
3.	Declaration	3 - 3
4.	Acknowledgment	4 - 4
5.	Abstract	5 - 5
6.	Table of Contents	6 - 6
7.	Introduction	7 - 11
8.	SDLC	12 – 16
9.	System Design	17 - 20
10.	Coding & Implementation	21 - 51
11.	Testing	52 – 56
12.	Application	57 – 58
13.	Conclusion	59 - 60
12.	Bibliography (APA Style)	61 - 61

INTRODUCTION

1. Background

With the increasing globalization of economies and cultures, the demand for multilingual communication skills has grown rapidly. Traditional methods of language learning often lack interactivity, engagement, and real-world conversation practice. Learners frequently struggle with motivation and opportunities to use their target language in meaningful contexts. SpeakMate aims to bridge this gap by providing a collaborative, interactive, and intelligent platform for individuals passionate about learning new languages.

2. Motivation

The inspiration for developing SpeakMate arose from observing the limitations of existing language learning platforms, which often focus solely on pre-recorded lessons or static exercises. Real learning happens through real interaction. SpeakMate not only connects learners from around the world to practice languages together, but also supports them with an AI chatbot that offers guidance, explanations, and examples anytime they need it. This combination of human connection and artificial intelligence provides a uniquely effective learning experience.

3. Objectives

- To provide a platform where users can connect with others learning the same language.
- To integrate AI technology for instant, 24/7 language learning support.
- To deliver personalized tutorials based on the user's selected language and proficiency level.
- To support goal tracking and progress monitoring for self-paced learning.

4. Scope

SpeakMate is a web-based platform accessible from any modern browser. It supports real-time chat rooms, AI-driven conversation support, and tutorial content delivery. The platform is designed to be scalable and adaptable, supporting the addition of more languages and features over time. Initially, the focus is on commonly spoken languages, with plans to expand as the user base grows.

5. Problem Statement

Language learners often face three core challenges: lack of practice partners, lack of personalized feedback, and lack of motivation. SpeakMate addresses these issues by creating a socially engaging learning environment, enhanced with intelligent AI support and structured tutorials. The platform fosters an ecosystem where users can teach, learn, and support each other.

SOFTWARE DEVELOPMENT LIFECYCLE (SDLC)

The development of the SpeakMate platform followed the **Agile Model** of SDLC. This approach enabled iterative development, continuous feedback, and regular adaptation to evolving requirements. The project was broken into sprints, with new features and refinements added progressively based on feedback and testing.

1. Requirement Analysis

The first phase involved gathering and analyzing both functional and non-functional requirements for the platform. This included:

- Functional Requirements:
    - User registration and login
    - Real-time chat between users
    - AI chatbot integration
    - Language selection and tutorials
    - Profile and progress tracking
- Non-functional Requirements:
    - Scalability
    - Security (JWT-based authentication)

- High availability
- Responsive design

Stakeholder feedback and competitive analysis helped shape a detailed Software Requirements Specification (SRS).

2. Feasibility Study

A feasibility analysis was conducted to evaluate:

- **Technical Feasibility:** The chosen technology stack (MERN, OpenAI API) was validated for implementation ease and community support.
- **Operational Feasibility:** The platform's core functions (chat, tutorials, AI support) were tested via small-scale prototypes.
- **Economic Feasibility:** The project was developed using open-source technologies, keeping costs minimal while allowing future commercial scaling.

3. System Design

This phase focused on converting the requirements into a visual and structural system plan. Key design activities included:

- Creating UML diagrams (Use Case, Class, Activity)
- Designing the database schema and ERD
- Developing wireframes and UI mock-ups
- Architecting the frontend-backend interaction flow

Design decisions prioritized modularity, reusability, and performance optimization.

4. Implementation

The implementation phase involved full-stack development in stages:

- **Frontend:** Built using React.js and Tailwind CSS with reusable ShadCN UI components.
- **Backend:** Implemented in Express.js with RESTful APIs for user authentication, chat, and tutorial data.
- **AI Bot:** Integrated OpenAI's GPT-3.5 model for interactive chatbot functionality.
- **Database:** MongoDB used for scalable and flexible data modeling.

Each module was developed independently, version-controlled using Git.

5. Deployment

After successful testing, the platform was deployed on the following services:

- **Frontend:** Deployed via Vercel for seamless CI/CD.
- **Backend:** Hosted on Render for scalable API hosting.
- **Database:** Managed via MongoDB Atlas.

The platform was made accessible to selected users for beta testing and feedback collection.

SYSTEM DESIGN

System design is a crucial phase that transforms the project requirements into a blueprint for development. For SpeakMate, the system design focuses on a modular, scalable, and maintainable structure that supports real-time communication, AI interaction, and personalized learning experiences.

1. System Architecture

SpeakMate follows a **three-tier architecture**:

1. **Presentation Layer (Frontend):**
   Built using **React.js** and **Tailwind CSS**, it manages the user interface and experience. It interacts with backend services via RESTful APIs and WebSocket for real-time features.
2. **Application Layer (Backend):**
   Developed using **Express.js**, this layer handles all business logic, including user authentication, AI chat processing, tutorial delivery, and messaging logic. It exposes APIs to the frontend and interacts with the database.
3. **Data Layer (Database & AI Integration):**
   - **MongoDB** stores users, chat logs, tutorials, and session data.

- **OpenAI API** is integrated to provide AI responses for language-related queries.

The architecture ensures separation of concerns and allows for independent scaling of components.

2. Entity Relationship Diagram (ERD)

The ERD represents the core entities and relationships in SpeakMate:

Entities:

- **User** (id, name, email, password, languageSelected, progress)
- **ChatRoom** (id, roomName, participants[], createdAt)
- **Message** (id, chatRoomId, senderId, content, timestamp, type)
- **Tutorial** (id, language, title, content, level)

Relationships:

- One **User** can participate in many **ChatRooms**
- Each **ChatRoom** can have many **Messages**
- **User** can access multiple **Tutorials**

3. UML Diagrams

Use Case Diagram

Shows the main interactions between the system and users:

Actors: User (Learner)

Use Cases:

- Register/Login
- Select language
- Join chat room
- Chat with AI
- View tutorials
- Track progress

Class Diagram

Classes:

- User
- ChatRoom
- Message
- Tutorial
- AIService

Relationships:

- User *associates* with ChatRoom
- ChatRoom *aggregates* Message

- User *requests* Tutorial
- AIService *interacts* with User for query resolution

Activity Diagram: AI Chat Interaction

1. User sends message →
2. Backend receives →
3. AI service processes input →
4. Returns AI-generated reply →
5. Message shown in UI

CODING & IMPLEMENTATION

1. Hardware Used:

- AMD Ryzen 5000 Processor
- 12 GB RAM
- 512 GB ROM
- 150 mbps WI-FI

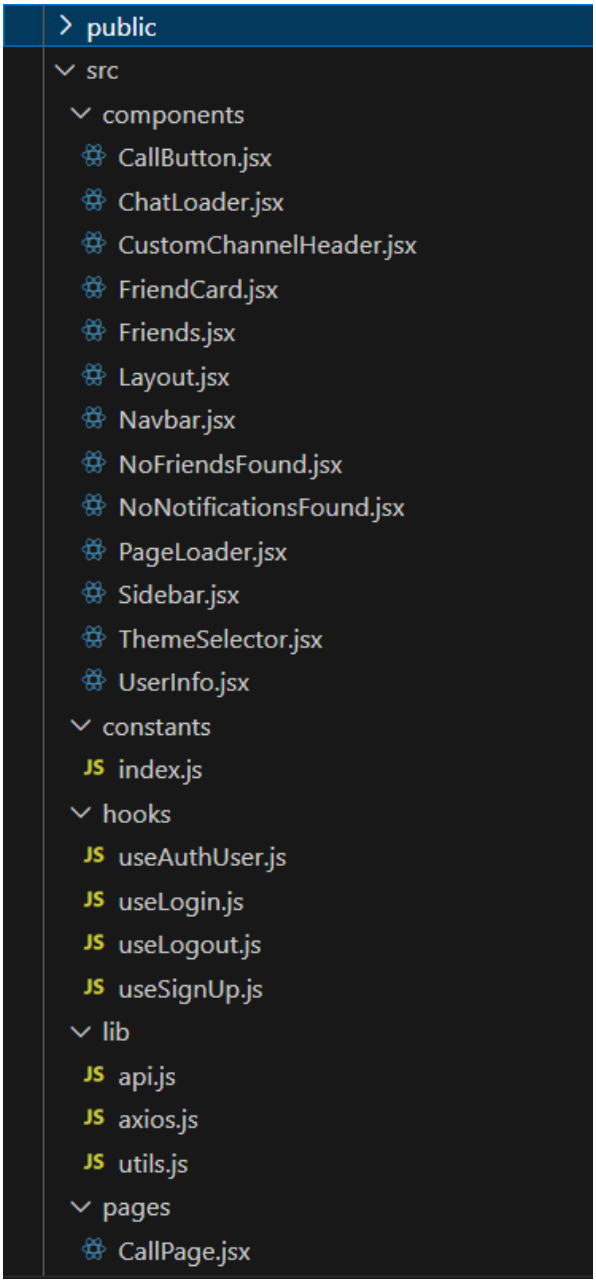
2. Software Used:

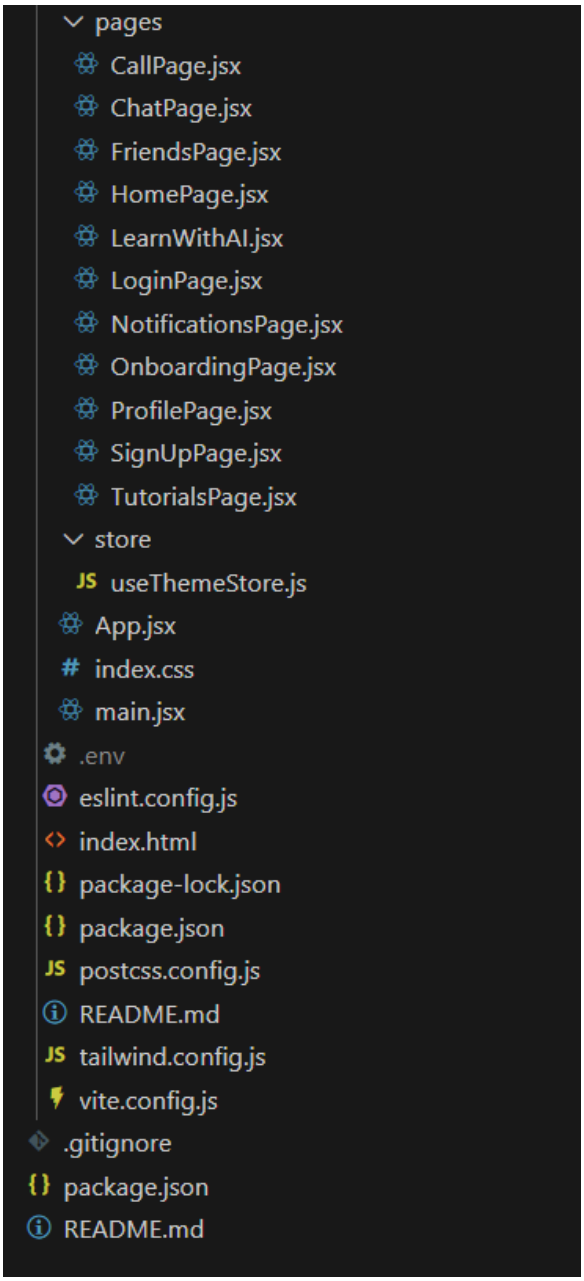
- OS – Windows-11
- Nodejs
- MongoDB
- Reactjs
- Visual Studio Code
- Git & Github

3. Technology Stack Overview

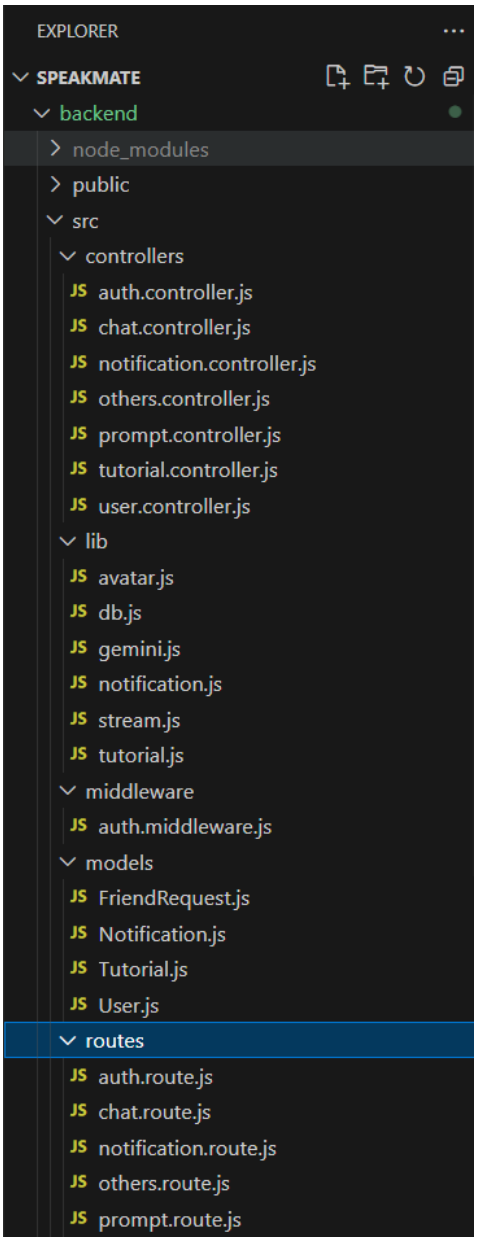
Layer	Tools / Tech	Purpose
Frontend	React.js, Tailwind CSS, ShadCN UI	UI/UX, responsive components
Backend	Node.js, Express.js	API development, AI logic, auth
Database	MongoDB (via Mongoose)	Stores users, messages, tutorials
AI Integration	Gemini GPT API	Smart chatbot for language learning help
Auth & Security	Bcrypt, JWT	Secured login/session management
Deployment	Vercel, Render, MongoDB Atlas	Hosting and DB management

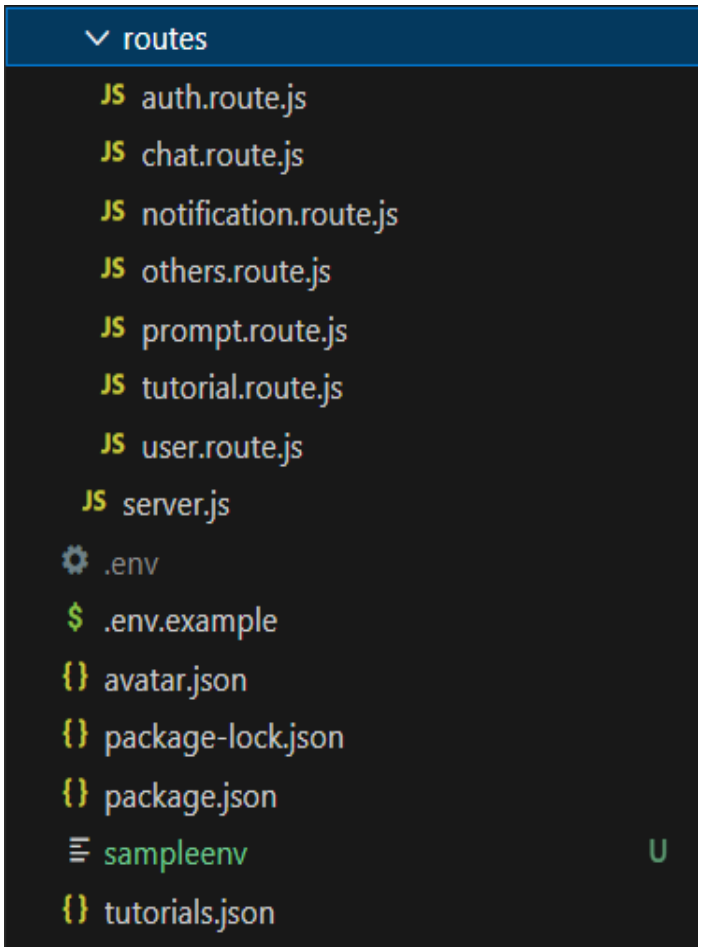
4. Frontend Project Structure



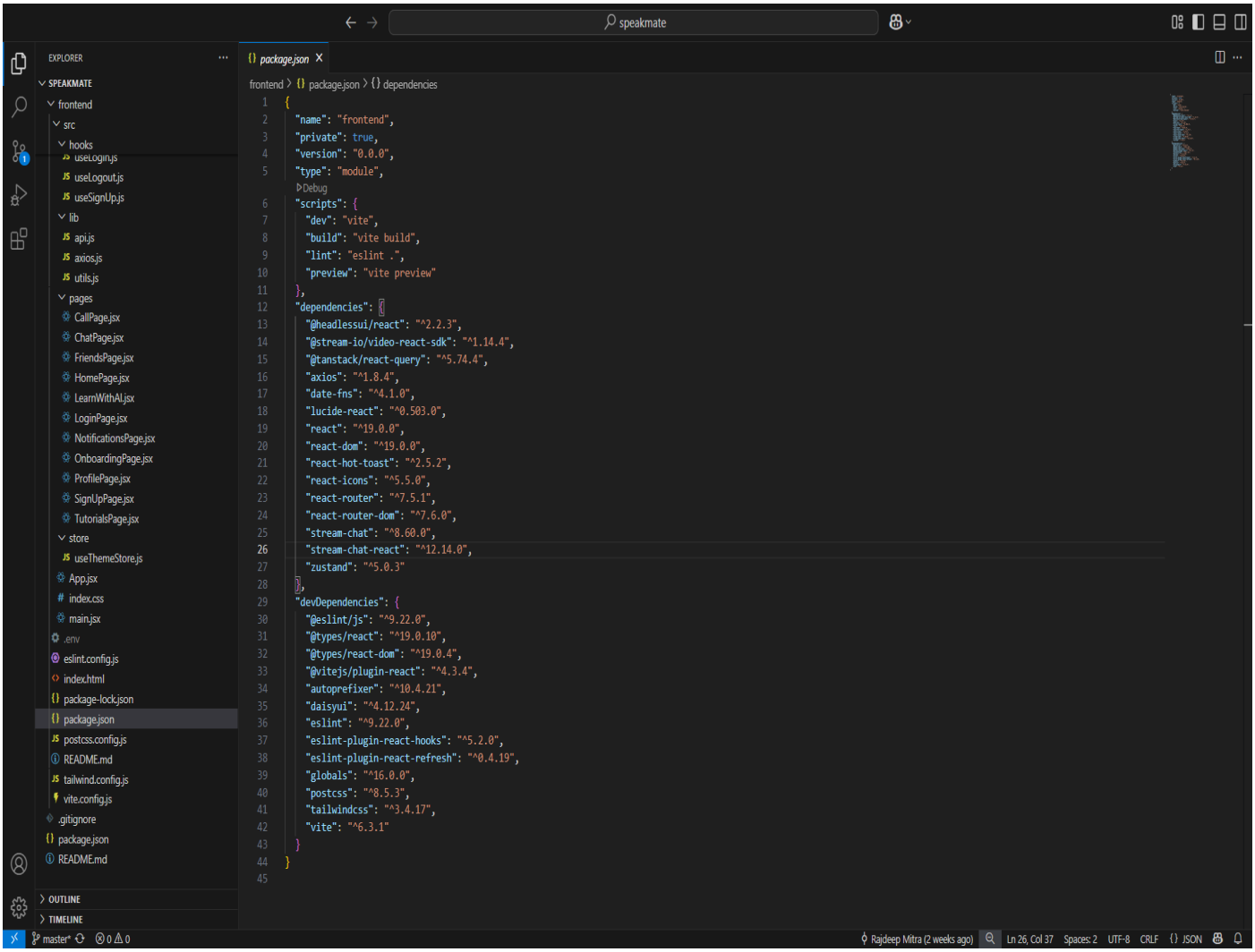


5. Backend Project Structure

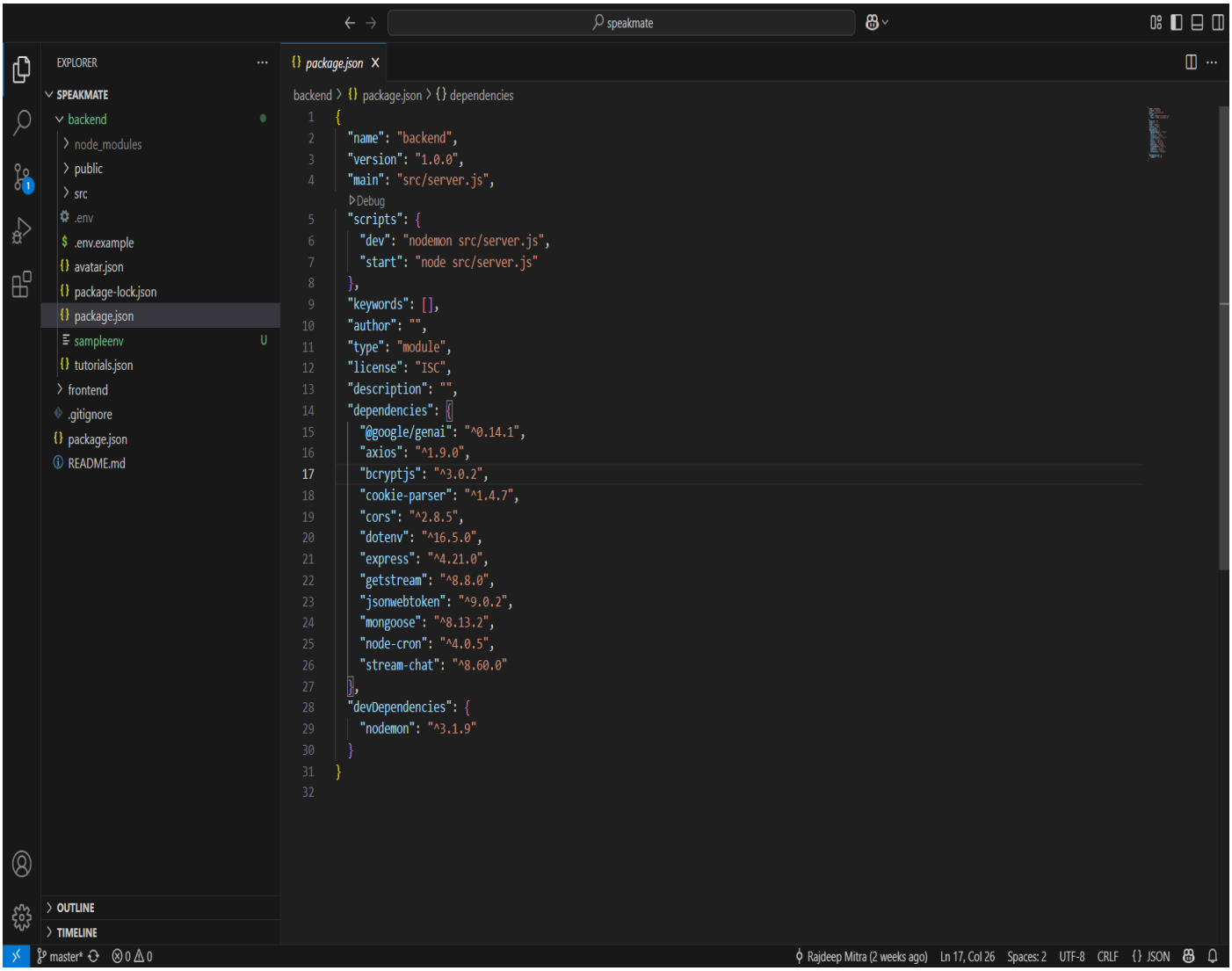




6. Frontend Packages Used



7. Backend Packages Used



8. Database Schema

a. User Model

```
{
  fullName: {
    type: String,
    required: true,
  },
  email: {
    type: String,
    required: true,
    unique: true,
  },
  password: {
    type: String,
    required: true,
    minlength: 6,
  },
  bio: {
    type: String,
    default: "",
  },
  gender: {
    type: String,
    default: "",
  },
```

```
    profilePic: {
      type: String,
      default: "",
    },
    nativeLanguage: {
      type: String,
      default: "",
    },
    learningLanguage: {
      type: String,
      default: "",
    },
    location: {
      type: String,
      default: "",
    },
    isOnboarded: {
      type: Boolean,
      default: false,
    },
    friends: [
      {
        type: mongoose.Schema.Types.ObjectId,
        ref: "User",
      },
    ],
  },
```

b. Friend Request

```
{
    sender: {
      type: mongoose.Schema.Types.ObjectId,
      ref: "User",
      required: true,
    },
    recipient: {
      type: mongoose.Schema.Types.ObjectId,
      ref: "User",
      required: true,
    },
    status: {
      type: String,
      enum: ["pending", "accepted"],
      default: "pending",
    },
  },
```

c. Tutorial Model

```
{
    videoId: {
      type: String,
      required: true,
      unique: true, // prevent duplicate entries
    },
    title: {
      type: String,
      required: true,
    },
    channel: {
      type: String,
      required: true,
    },
    thumbnail: {
      type: String,
    },
    language: {
      type: String,
      required: true,
    },
  },
```

d. Notification Model

```
{
  recipient: {
    type: mongoose.Schema.Types.ObjectId,
    ref: "User",
    required: true,
  },
  sender: {
    type: mongoose.Schema.Types.ObjectId,
    ref: "User",
  },
  type: {
    type: String,
    enum: ["friend_request"],
    required: true,
  },
  message: {
    type: String,
  },
  entityId: {
    type: mongoose.Schema.Types.ObjectId,
    required: false,
  },
  isRead: {
    type: Boolean,
    default: false,
  },
  actionTaken: {
    type: Boolean,
    default: false,
  },
 },
```

9. User Registration

```
export async function signup(req, res) {
  const { email, password, fullName } = req.body;

  try {
    if (!email || !password || !fullName) {
      return res.status(400).json({ message: "All fields are required" });
    }

    if (password.length < 6) {
      return res
        .status(400)
        .json({ message: "Password must be at least 6 characters" });
    }

    const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;

    if (!emailRegex.test(email)) {
      return res.status(400).json({ message: "Invalid email format" });
    }

    const existingUser = await User.findOne({ email });
    if (existingUser) {
      return res
        .status(400)
        .json({ message: "Email already exists, please use a diffrent one" });
```

```
    }

    const idx = Math.floor(Math.random() * 50) + 1; // 1-50 included
    const randomGender =
      Math.floor(Math.random() * 2) === 0 ? "male" : "female"; // Randomly choose
between male or female
    const randomAvatar = path.join("avatars", `${randomGender}-${idx}.png`);
    const newUser = await User.create({
      email,
      fullName,
      password,
      profilePic: randomAvatar,
    });

    try {
      await upsertStreamUser({
        id: newUser._id.toString(),
        name: newUser.fullName,
        image: newUser.profilePic || "",
      });
      console.log(`Stream user created for ${newUser.fullName}`);
    } catch (error) {
      console.log("Error creating Stream user:", error);
    }

    const token = jwt.sign(
      { userId: newUser._id },
      process.env.JWT_SECRET_KEY,
```

```
      {
        expiresIn: "7d",
      }
    );

    res.cookie("jwt", token, {
      maxAge: 7 * 24 * 60 * 60 * 1000,
      httpOnly: true, // prevent XSS attacks,
      sameSite: "strict", // prevent CSRF attacks
      secure: process.env.NODE_ENV === "production",
    });

    res.status(201).json({ success: true, user: newUser });
  } catch (error) {
    console.log("Error in signup controller", error);
    res.status(500).json({ message: "Internal Server Error" });
  }
}
```

10. User Login

```
export async function login(req, res) {
  try {
    const { email, password } = req.body;

    if (!email || !password) {
      return res.status(400).json({ message: "All fields are required" });
    }

    const user = await User.findOne({ email });
    if (!user)
      return res.status(401).json({ message: "Invalid email or password" });

    const isPasswordCorrect = await user.matchPassword(password);
    if (!isPasswordCorrect)
      return res.status(401).json({ message: "Invalid email or password" });

    const token = jwt.sign({ userId: user._id }, process.env.JWT_SECRET_KEY, {
      expiresIn: "7d",
    });

    res.cookie("jwt", token, {
      maxAge: 7 * 24 * 60 * 60 * 1000,
      httpOnly: true, // prevent XSS attacks,
      sameSite: "strict", // prevent CSRF attacks
      secure: process.env.NODE_ENV === "production",
```

```
    });

    res.status(200).json({ success: true, user });
  } catch (error) {
    console.log("Error in login controller", error.message);
    res.status(500).json({ message: "Internal Server Error" });
  }
}
```

11. User Logout

```
export function logout(req, res) {
  res.clearCookie("jwt");
  res.status(200).json({ success: true, message: "Logout successful" });
}
```

12. User Onboarding

```
export async function onboard(req, res) {
  try {
    const userId = req.user._id;

    const {
      fullName,
      bio,
      gender,
      nativeLanguage,
      learningLanguage,
      location,
    } = req.body;

    if (
      !fullName ||
      !bio ||
      !gender ||
      !nativeLanguage ||
      !learningLanguage ||
      !location
    ) {
      return res.status(400).json({
        message: "All fields are required",
        missingFields: [
          !fullName && "fullName",
```

```
      !bio && "bio",
      !gender && "gender",
      !nativeLanguage && "nativeLanguage",
      !learningLanguage && "learningLanguage",
      !location && "location",
    ].filter(Boolean),
  });
} else if (bio.length >= 100) {
  return res
    .status(400)
    .json({ message: "Bio must be less than equal to 100 characters" });
} else if (learningLanguage === nativeLanguage) {
  return res
    .status(400)
    .json({ message: "Learning and native languages must be different" });
}

const updatedUser = await User.findByIdAndUpdate(
  userId,
  {
    ...req.body,
    isOnboarded: true,
  },
  { new: true }
);

if (!updatedUser)
  return res.status(404).json({ message: "User not found" });
```

```
    try {
      await upsertStreamUser({
        id: updatedUser._id.toString(),
        name: updatedUser.fullName,
        image: updatedUser.profilePic || "",
      });
      console.log(
        `Stream user updated after onboarding for ${updatedUser.fullName}`
      );
    } catch (streamError) {
      console.log(
        "Error updating Stream user during onboarding:",
        streamError.message
      );
    }

    res.status(200).json({ success: true, user: updatedUser });
  } catch (error) {
    console.error("Onboarding error:", error);
    res.status(500).json({ message: "Internal Server Error" });
  }
}
```

13. One-to-One / Group Chat

```
export async function getStreamToken(req, res) {
  try {
    const token = generateStreamChatToken(req.user.id);


    res.status(200).json({ token });
  } catch (error) {
    console.log("Error in getStreamToken controller:", error.message);
    res.status(500).json({ message: "Internal Server Error" });
  }
}
```

14. AI Chat Generation

```
export async function generatePrompt(req, res) {
  try {
    const { prompt } = req.body;
    if (!prompt) {
      return res.status(400).json({ message: "Prompt is required" });
    }
    const result = await generateGeminiPrompt(prompt);
    if (!result) {
      return res.status(500).json({ message: "Error generating prompt" });
    }
    res.status(200).json({ result });
  } catch (error) {
    // console.error("Error in generatePrompt controller", error.message);
    res.status(500).json({ message: "Internal Server Error" });
  }
}
```

15. Fetch Tutorials

```
export async function getTutorials(req, res) {
  try {
    const { page = 1, limit = 6 } = req.query;
    console.log(req.user.learningLanguage.toLowerCase());
    const query = req.user.learningLanguage
      ? {
          language: {
            $regex: req.user.learningLanguage,
            $options: "i",
          },
          title: {
            $regex: req.user.learningLanguage,
            $options: "i",
          },
        }
      : {};

    const skip = (Number(page) - 1) * Number(limit);

    const [videos, total] = await Promise.all([
      Tutorial.find(query)
        .sort({ createdAt: -1 }) // Latest first
        .skip(skip)
        .limit(Number(limit)),
      Tutorial.countDocuments(query),
```

```
    ]);

    const totalPages = Math.ceil(total / Number(limit));

    res.status(200).json({
      videos,
      page: Number(page),
      totalPages,
      totalResults: total,
      hasNextPage: Number(page) < totalPages,
      hasPrevPage: Number(page) > 1,
    });
  } catch (error) {
    res.status(500).json({
      message: "Error fetching tutorials",
      error: error.message,
    });
  }
}
```

16. Get Notifications

```
export const getNotifications = async (req, res) => {
  try {
    const userId = req.user.id;

    const page = parseInt(req.query.page) || 1;
    const limit = parseInt(req.query.limit) || 10;
    const sortDirection = req.query.sort === "asc" ? 1 : -1;

    const skip = (page - 1) * limit;

    const [notifications, total] = await Promise.all([
      Notification.find({ recipient: userId })
        .sort({ createdAt: sortDirection })
        .skip(skip)
        .limit(limit)
        .populate("sender", "username profilePicture"),

      Notification.countDocuments({ recipient: userId }),
    ]);

    res.status(200).json({
      notifications,
      pagination: {
        totalItems: total,
        currentPage: 1,
```

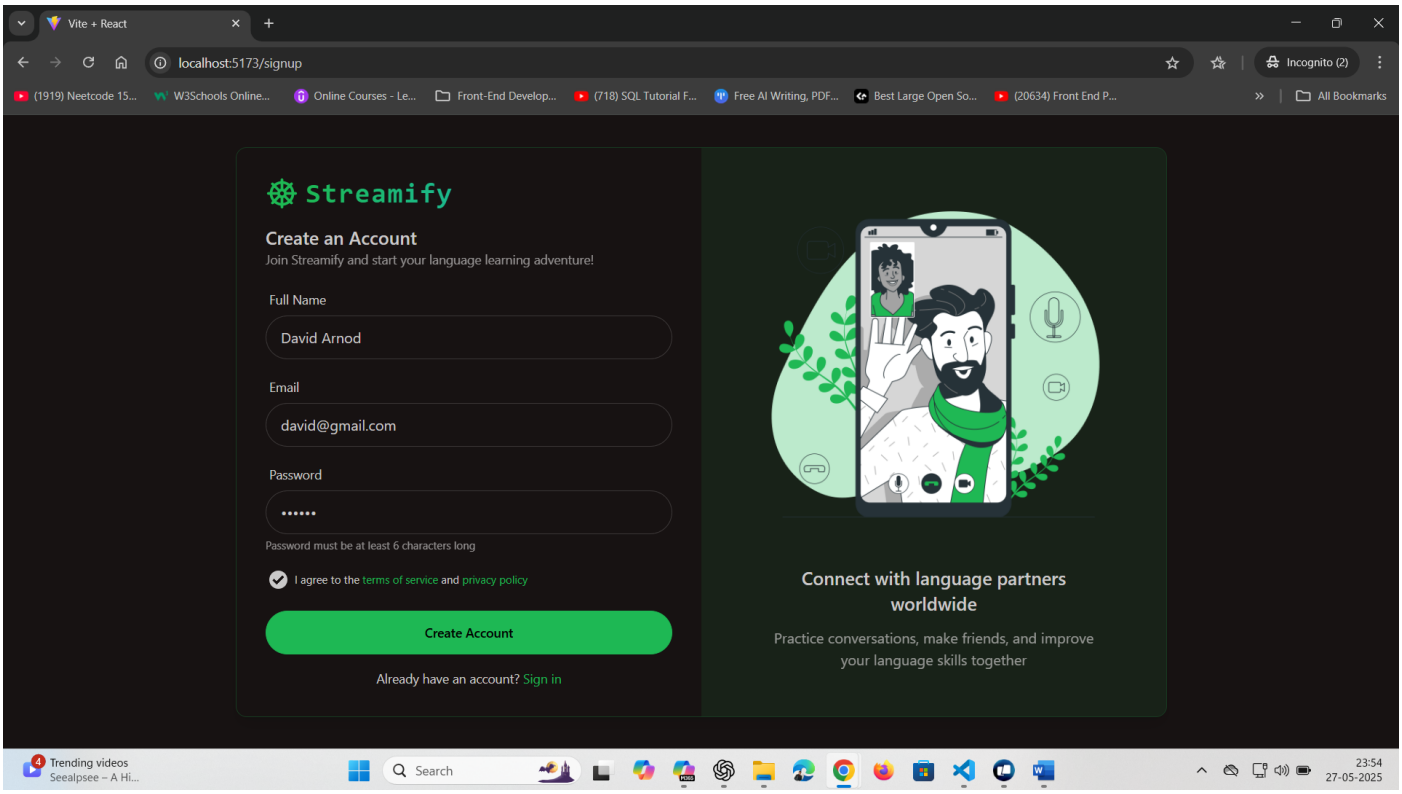
```
        itemsPerPage: limit,
        totalPages: Math.ceil(total / limit),
      },
    });
  } catch (err) {
    res.status(500).json({ error: "Failed to fetch notifications" });
  }
};
```

17. Get Unread Notifications Count

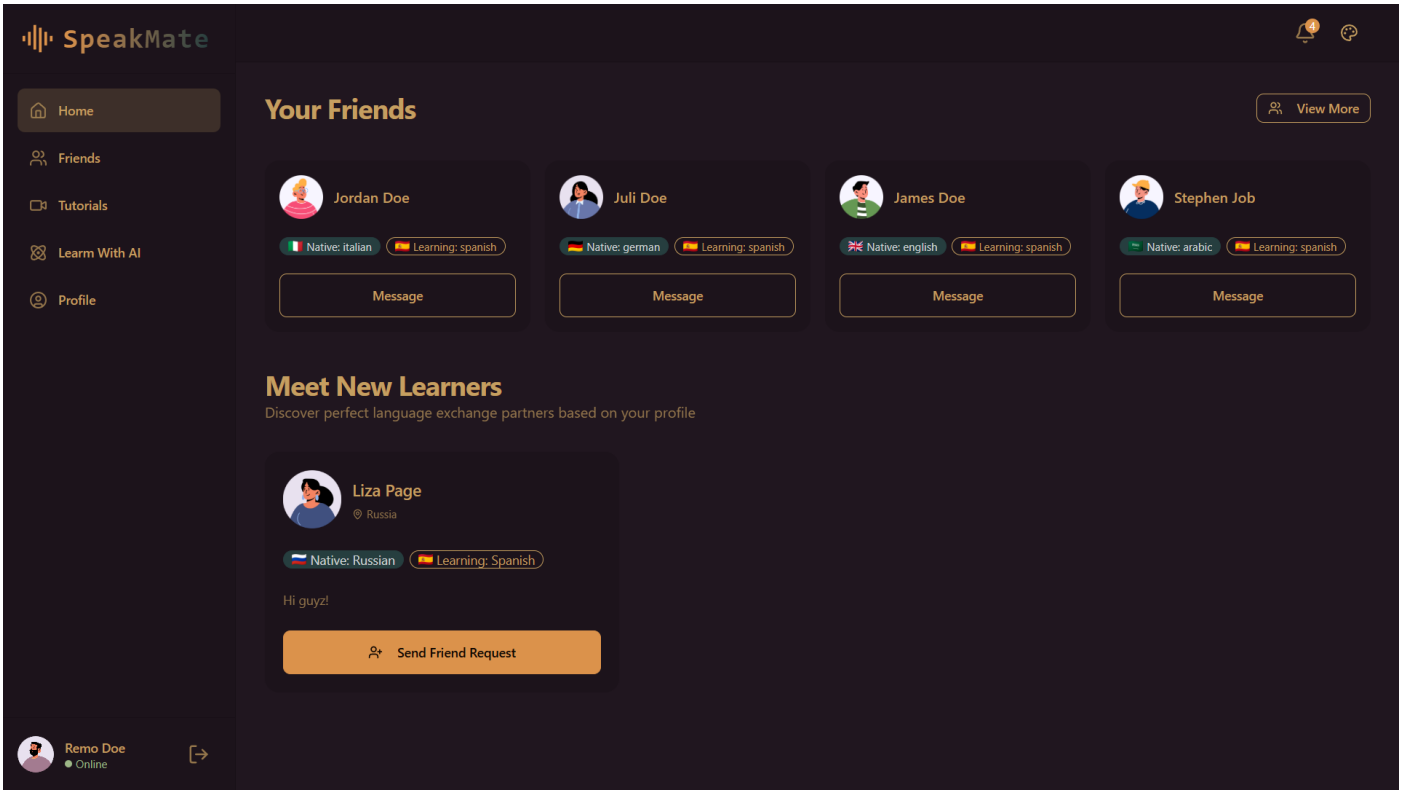
```
export const getUnreadNotificationCount = async (req, res) => {
  try {
    const userId = req.user.id;

    const count = await Notification.countDocuments({
      recipient: userId,
      isRead: false,
    });

    res.status(200).json({ unreadCount: count });
  } catch (err) {
    res.status(500).json({ error: "Failed to fetch unread count" });
  }
};
```

18. UI Outputs

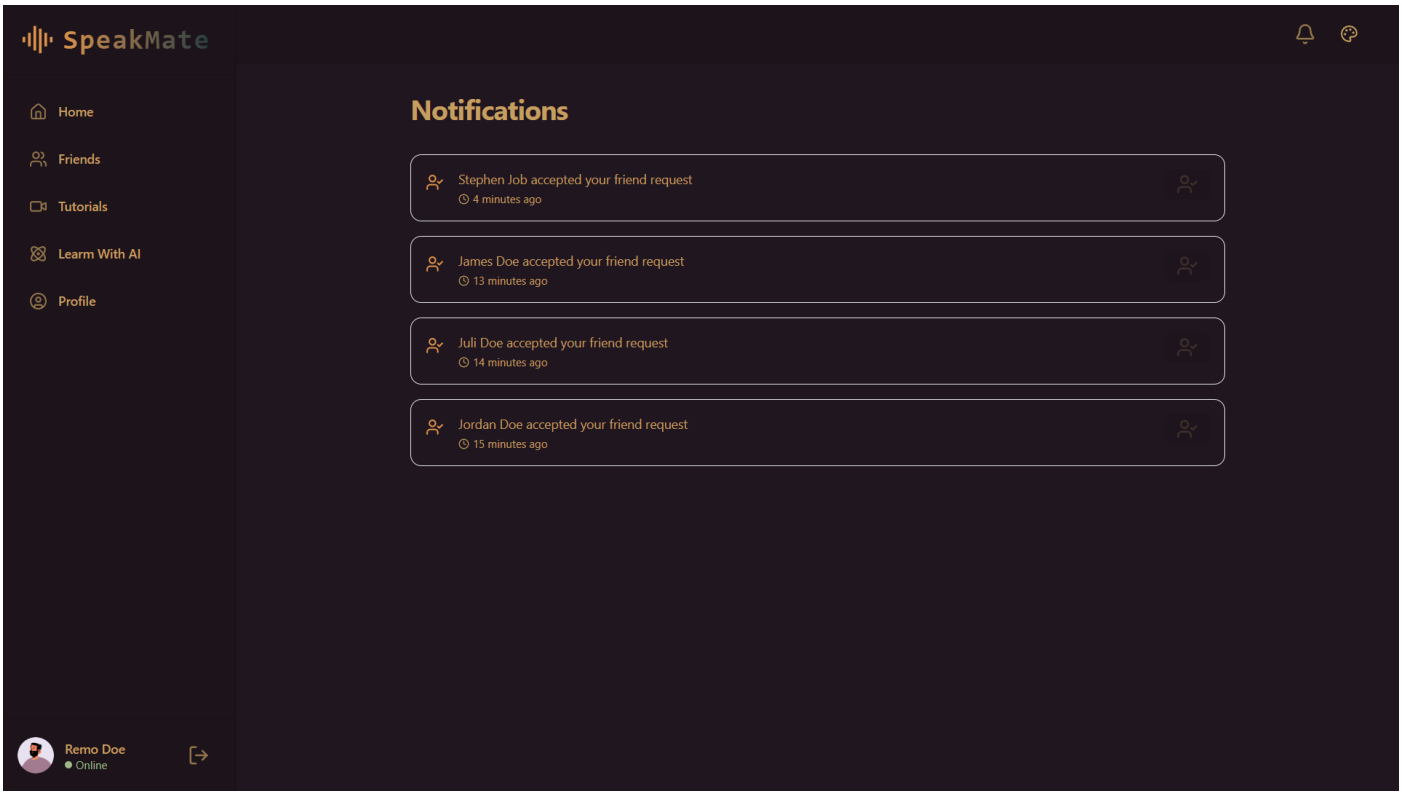
a. User Registration



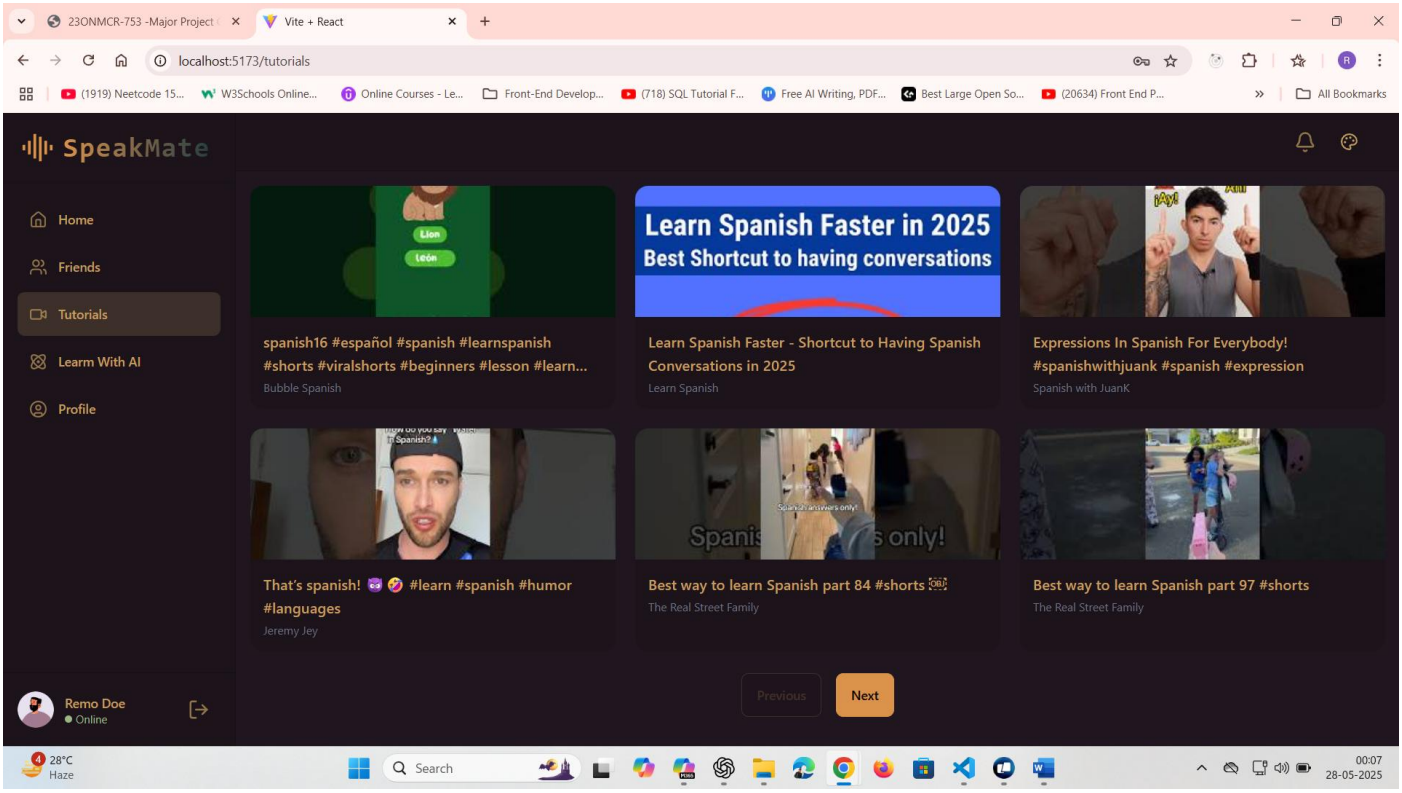
b. Dashboard

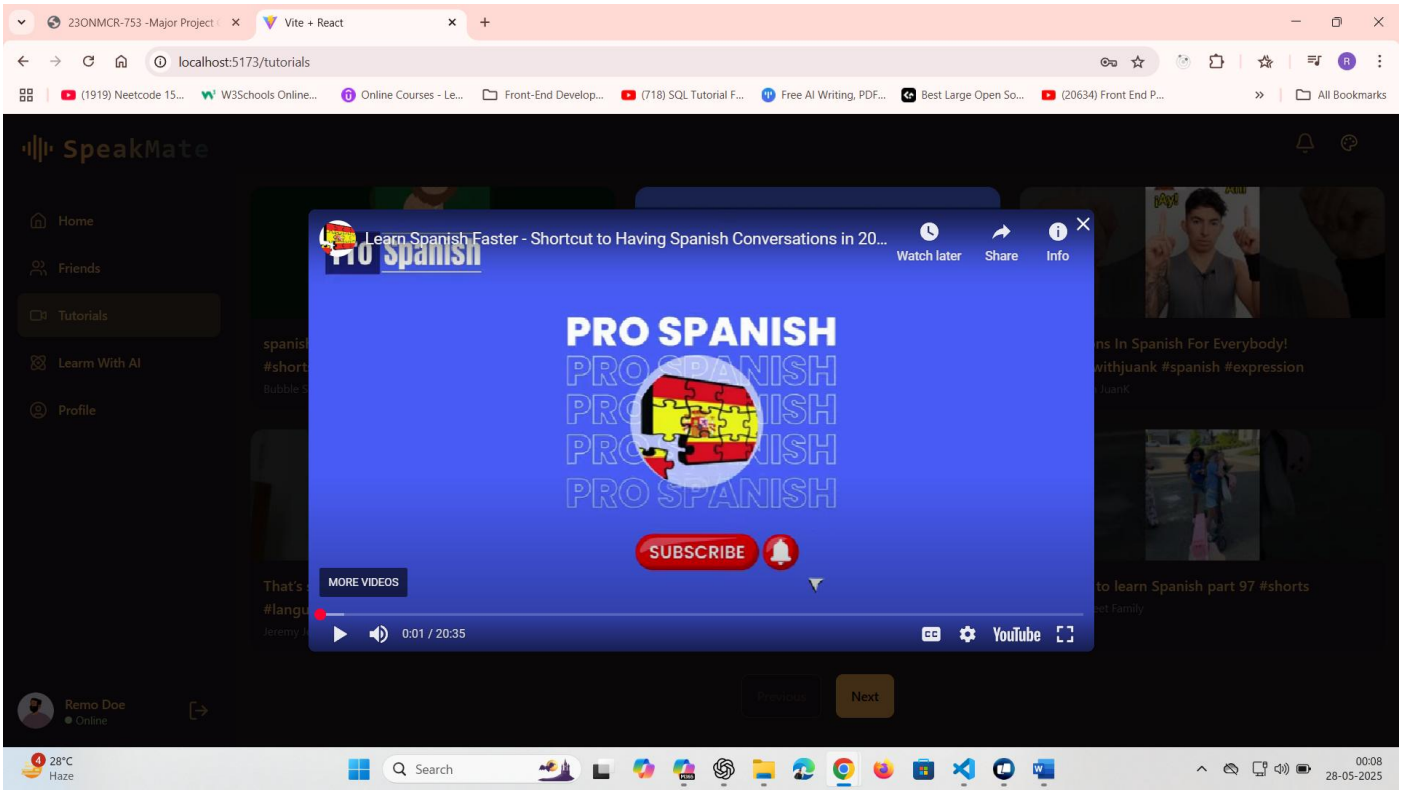

SpeakMate
Home
Friends
Tutorials
Learm With AI
Profile
Remo Doe
Online
Your Friends
View More
Jordan Doe
Native: italian
Learning: spanish
Message
Juli Doe
Native: german
Learning: spanish
Message
James Doe
Native: english
Learning: spanish
Message
Stephen Job
Native: arabic
Learning: spanish
Message
Meet New Learners
Discover perfect language exchange partners based on your profile
Liza Page
Russia
Native: Russian
Learning: Spanish
Hi guyz!
Send Friend Request

c. Notification Page

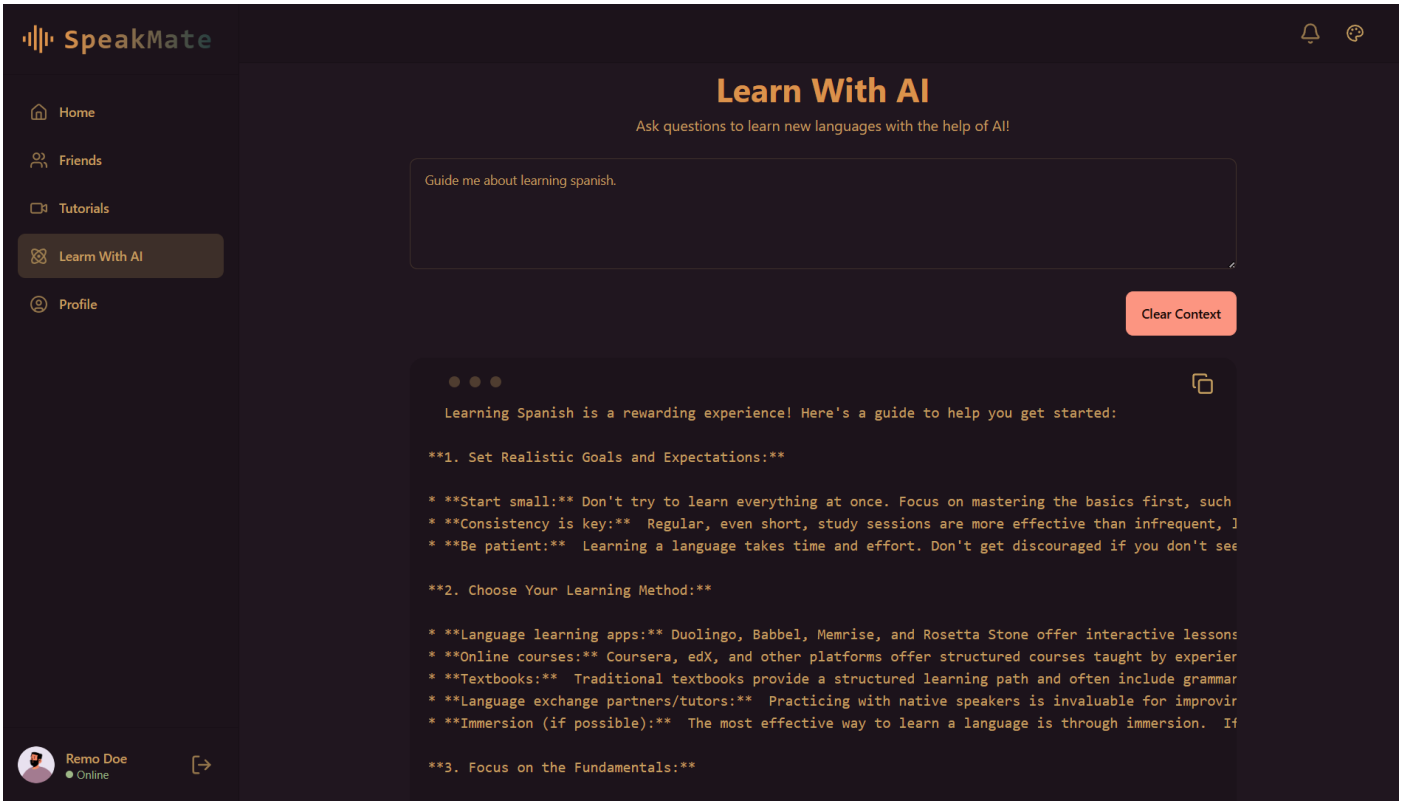


d. Tutorials Page





e. Learn with AI



19. Deployed Links

Vercel => https://speakmate-bj4d.vercel.app/

GitHub => https://github.com/mitrarajdeep2001/speakmate

TESTING

Testing is an essential phase in the software development lifecycle to ensure the reliability, performance, and correctness of the system. For SpeakMate, testing was carried out at multiple levels including unit tests, integration tests, manual testing, and user acceptance testing (UAT).

1. Testing Strategy

The following strategies were adopted:

- **Unit Testing:** To test individual functions and components.
- **Integration Testing:** To test the interactions between frontend, backend, and database.
- **End-to-End Testing (Manual):** To test complete user flows such as registration, chatting, AI interactions, and tutorial access.
- **Performance Testing:** To ensure the platform could handle concurrent users.

Tools Used:

- **Jest:** For unit testing backend logic.
- **React Testing Library:** For testing React components.
- **Postman:** For API testing.
- **Socket.io Tester:** For real-time chat simulation.

2. Unit Testing Examples

Example 1: AI Chat Controller

```
it('should return a response from OpenAI', async () => {
  const res = await request(app)
    .post('/api/ai-chat')
    .send({ message: 'What is a verb?' });

  expect(res.status).toBe(200);
  expect(res.body.reply).toContain('verb');
});
```

Example 2: Tutorial Filtering Logic

```
it('should return tutorials based on language', async () => {
  const res = await request(app)
    .get('/api/tutorials?lang=Spanish');

  expect(res.status).toBe(200);
  expect(res.body.length).toBeGreaterThan(0);
});
```

3. Integration Testing

- Tested real-time message delivery using mock Socket.io clients.
- Verified session handling between frontend and backend.
- Ensured AI API integration returned consistent results.

Integration Test: Socket.io Chat

```
socket.emit('joinRoom', 'roomId123');
socket.emit('message', { room: 'roomId123', text: 'Hola!' });
```

Result: Message was received by all clients in the room.

4. Manual Testing & Use Cases

Scenario	Expected Result	Status
User signs up	Redirect to dashboard	✅
AI responds to English grammar question	Returns structured answer	✅
User sends message in chat room	Message appears instantly	✅
View tutorial in selected language	Correct content displays	✅

5. Bug Tracking

Bugs were tracked via GitHub Issues with appropriate labels:

- **Type:** bug / enhancement / question
- **Status:** open / in progress / resolved
- **Severity:** low / medium / high

SpeakMate passed all critical test cases and use scenarios. Comprehensive testing across components helped ensure that the platform performs reliably in real-world usage. Continuous testing and bug tracking allowed the system to improve iteratively.

APPLICATION

The SpeakMate platform was developed to provide real-world language learning experiences through both peer interaction and AI assistance. This section outlines the practical applications of the system, key user-facing features, and real-world use scenarios.

1. Key Features

1. **AI-Powered Chatbot**
    - Offers real-time help on grammar, vocabulary, and usage.
    - Responds in selected languages.
    - Works 24/7 for on-demand learning.
2. **Real-Time Chat Rooms**
    - Users can join language-specific rooms.
    - Peer-to-peer conversations enhance learning through practice.
    - Built using Socket.io for real-time communication.
3. **Interactive Tutorials**
    - Personalized tutorials based on selected language.
    - Topics include common phrases, grammar rules, sentence construction.
    - Markdown-formatted and fetched dynamically.
4. **Progress Tracking**
    - Users can view their learning history.
    - Tutorial completion and chat frequency are logged.
5. **User Authentication**

- Secured login using Auth0.
- User-specific data and preferences stored securely.

2. Real-World Use Cases

- **Beginner learner:** Joins Spanish chat room and practices with peers, then uses the AI for feedback.
- **Intermediate user:** Reads tutorials on sentence structure and uses AI for grammar clarification.
- **Busy professional:** Chats with AI during short breaks to revise vocabulary.

CONCLUSION

The SpeakMate project was designed and developed as an innovative platform to make language learning more interactive, collaborative, and intelligent. By integrating AI-driven chatbots with real-time communication and structured tutorial delivery, it successfully bridges the gap between traditional learning methods and modern digital education.

The development process followed a structured SDLC approach, enabling clear phases of requirement analysis, system design, implementation, testing, and deployment. The use of a modern technology stack such as MERN and the OpenAI API empowered the platform with a rich user experience and smart assistance capabilities.

Through rigorous testing and feedback iterations, the application proved to be functional, scalable, and user-friendly. SpeakMate provides learners with access to real-time practice and instant explanations, which are vital for effective language acquisition.

1. Limitations

- **Limited Language Support:** Currently supports only a few major languages.
- **Dependency on External APIs:** AI chatbot performance relies heavily on the availability and responsiveness of OpenAI services.
- **No Mobile App Yet:** Platform is currently available only on web browsers.

2. Future Enhancements

- **Mobile Application:** Develop a cross-platform mobile app using React Native or Flutter.
- **Gamification:** Add quizzes, badges, and leaderboards to motivate learners.
- **Voice Chat:** Enable voice-based conversation for pronunciation practice.
- **Language Expansion:** Add support for less common languages like Korean, Dutch, and Hindi.
- **Offline Mode:** Allow users to download tutorials for offline study.

In conclusion, SpeakMate is a promising platform with strong foundations and great potential. It empowers learners to build confidence, engage with peers, and receive intelligent support—all within a single environment.

BIBLIOGRAPHY (APA STYLE)

- Gemini. (2023). *Gemini API Documentation*. Retrieved from https://ai.google.dev/gemini-api/docs
- Mozilla. (2023). *MDN Web Docs*. Retrieved from https://developer.mozilla.org/
- React. (2023). *React Documentation*. Retrieved from https://reactjs.org/
- Express. (2023). *Express.js Guide*. Retrieved from https://expressjs.com/
- Tailwind Labs. (2023). *Tailwind CSS Documentation*. Retrieved from https://tailwindcss.com/
- MongoDB, Inc. (2023). *MongoDB Manual*. Retrieved from https://www.mongodb.com/docs/manual/
- Vercel. (2023). *Vercel Deployment Guides*. Retrieved from https://vercel.com/docs
- Render. (2023). *Render Deployment Documentation*. Retrieved from https://render.com/docs
- Socket.io. (2023). *Socket.io Documentation*. Retrieved from https://socket.io/docs/