**Spartan Cove**

|  |
| --- |
| A Project Report Presented to  CMPE-272  Fall, 2024 |

|  |
| --- |
| By |
| Aditya Rajpurohit, Rishi Patel, Rushabh Runwal, Yashishvin Pothuri |
| 12/06/2024 |

|  |
| --- |
| Copyright © 2024 |
| Aditya Rajpurohit, Rishi Patel, Rushabh Runwal, Yashishvin Pothuri |
| ALL RIGHTS RESERVED |

**ABSTRACT**

Spartan Cove

By Aditya Rajpurohit, Rishi Patel, Rushabh Runwal, Yashishvin Pothuri

The proposed project, **Spartan Cove**, is a decentralized real-time chat application designed exclusively for the San Jose State University (SJSU) community, including students, faculty, and staff. By integrating seamlessly with SJSU’s existing infrastructure, this platform aims to enhance campus communication through a secure and feature-rich interface. Spartan Cove leverages a decentralized database architecture, ensuring high data resilience, privacy, and integrity by distributing data across multiple nodes. This design minimizes the risk of data breaches and system failures, while offering real-time messaging capabilities for instant communication. With Google Single Sign-On (SSO) integration, users authenticate via their SJSU email, enabling a streamlined login process and exclusive access to a comprehensive directory of SJSU members for effortless connections.

The application prioritizes security with end-to-end encryption (E2EE), ensuring messages remain private and accessible only to intended recipients. It offers advanced features like group chats, multimedia sharing, and message scheduling to enhance productivity and convenience. Spartan Cove also includes notifications for key activities and an informative bot powered by the SAMMY API, which provides updates on campus events and schedules. By combining robust security, modern communication tools, and SJSU-specific functionalities, Spartan Cove aspires to create a dynamic and secure communication hub for the university community.

**Acknowledgment**

We would like to extend our heartfelt gratitude to our professor, Andrew Bond, for his invaluable guidance, constructive feedback, and continuous support throughout the development of Spartan Cove. Their insights and expertise played a pivotal role in helping us shape the project and overcome challenges.

We are deeply appreciative of our teammates for their hard work, dedication, and collaborative spirit. The combined effort, creativity, and commitment of each team member were essential in delivering a functional and secure application. We would also like to thank the SJSU faculty and staff for providing the necessary resources and infrastructure that supported our research and development.

**Table of Contents**

# Chapter 1 Introduction

* 1. Project goals and objectives
  2. Problem and motivation
  3. Project application and impact
  4. Project results and deliverables

# Chapter 2 Project Background and Related Work

* 1. Background and used technologies
  2. State-of-the-art technologies

# Chapter 3 System Requirements and Analysis

* 1. Domain and business requirements
  2. Customer-oriented requirements
  3. System function requirements
  4. System behavior requirements
  5. System performance and non-function requirements
  6. System context and interface requirements
  7. Technology and resource requirements

# Chapter 4 System Design

* 1. System architecture design
  2. System data and database design
  3. System interface and connectivity design
  4. System user interface design
  5. System component API and logic design
  6. System design problems, solutions, and patterns

# Chapter 5 System Implementation

* 1. System implementation summary
  2. System implementation issues and resolutions
  3. Used technologies and tools

# Chapter 6 System Testing and Experiment

* 1. Testing and experiment scope
  2. Testing and experiment approaches

# Chapter 7 Conclusion and Future Work

* 1. Project summary
  2. Future Work

# List of Figures

Figure 1. Use-Case Diagram 4

Figure 2. System Infrastructure Architecture 8

Figure 3. Database Schema 8

Figure 4. Chat Demo 10

Figure 5. Adding a New User 11

Figure 6. Sparrtan AI Chatbot Demo 11

Figure 7. Testing File Implemented 15

**List of Tables**

Table 1. User Groups and Their Use-Cases 5

Table 2. System Functional Requirements 5

Table 3. System Performance and Non-Functional Requirements 5

Table 4. System Interface Requirements 6

Table 5. Testing Criteria 14

Table 6. Testing Methods and Tools 15

**Chapter 1 Introduction**

* 1. **Project Goals and objectives**

The goal of **Spartan Cove** is to create a secure and AI-enabled chat application exclusively for the SJSU community to enhance communication and collaboration. The project aims to provide seamless real-time messaging, integrated with Google Single Sign-On (SSO) for secure authentication. By incorporating end-to-end encryption and AI-powered assistance, Spartan Cove ensures user privacy, facilitates quick access to campus-related information, and offers features like role-based access control, continuous integration (CI/CD), and scalable infrastructure. This platform aspires to foster a more connected and efficient environment for students, faculty, and staff.

**1.2 Project Goals and objectives**

SJSU’s current communication tools are fragmented, inefficient, and lack dedicated features for secure, real-time interaction. This gap creates delays in information sharing and collaboration. Spartan Cove addresses these issues by offering a secure, decentralized chat platform with AI-powered capabilities. The project is motivated by the need to enhance connectivity, streamline access to university-related data, and ensure data privacy through SSL/TLS encryption and Google SSO. This project contributes to academic advancements in secure enterprise communication and provides technical insights into integrating CI/CD pipelines and role-based access control.

**1.3 Project application and impact**

The **Spartan Cove** application directly impacts the SJSU community by improving campus communication, facilitating efficient collaboration, and enhancing user experience through AI-enabled features. Academically, it demonstrates advanced integration of secure authentication, AI, and CI/CD processes. For industry, it serves as a case study for developing scalable, decentralized chat applications with enterprise-level security. Societally, the project offers a model for other educational institutions aiming to modernize communication while prioritizing data privacy and user engagement.

**1.4 Project results and expected deliverables**

The key outcomes of Spartan Cove include a fully functional AI-powered chat application with Google SSO integration, end-to-end encryption, and real-time messaging. Deliverables include a detailed project plan, comprehensive project report with design patterns and test cases, a live project presentation, and a GitHub code repository (RAYR\_Spartan\_Cove). Additional integrations, such as Jenkins CI/CD, layered security, and role-based access, will support ongoing development and maintenance, ensuring Spartan Cove is a robust and scalable communication solution for SJSU.

**Chapter 2 Background and Related Work**

**2.1 Project Goals and objectives**

The Spartan Cove project leverages modern web and cloud technologies to create a secure, real-time, AI-enabled chat application for the SJSU community. The application utilizes a Full-Stack MERN architecture (MongoDB, Express.js, React.js, Node.js) to provide a responsive and scalable platform. MongoDB handles flexible data storage for users, groups, and messages, while Express.js powers the backend API for efficient handling of requests. The frontend, built with React.js, offers a seamless user experience with real-time updates, and Node.js supports server-side operations with asynchronous processing. Socket.io enables real-time chat functionality with instant message delivery, typing indicators, and real-time user presence tracking. The AI assistant, powered by xAI’s Grok 2, provides accurate, context-specific information to users. Additionally, Google SSO ensures secure user authentication, and infrastructure provisioning on AWS is managed using Terraform, with Auto Scaling Groups and security configurations for high availability and network segmentation.

**2.2 State-of-the-art**

The current market offers several real-time chat applications and AI-powered communication tools, each serving different needs:

1. **General Chat Applications**:
   * **Slack**: A popular workplace communication tool with real-time messaging, channels, and integrations, but lacks university-specific AI assistance.
   * **Microsoft Teams**: Offers chat, video conferencing, and collaborative tools, integrated with Microsoft 365. However, it is primarily designed for enterprises.
2. **University-Specific Platforms**:
   * **Piazza**: Facilitates Q&A and discussions for academic courses but lacks real-time chat and AI integration.
   * **Canvas Messaging**: Integrated with learning management systems for basic communication but lacks advanced features like AI assistance and real-time group chat.
3. **AI Chatbots**:
   * **ChatGPT**: A versatile AI chatbot but not specifically tailored to university environments or integrated with real-time communication tools.
   * **IBM Watson Assistant**: Provides AI-driven support but requires significant customization for educational institutions.

**Spartan Cove** stands out by combining the best of these platforms: real-time group chat, AI-powered assistance specific to SJSU, secure Google SSO authentication, and modern infrastructure designed for scalability and high availability. This comprehensive approach addresses the unique needs of the SJSU community, providing a tailored and efficient communication platform.

**Chapter 3 System Requirements and Analysis**

**3.1 Domain and Business Requirements**

The **Spartan Cove** application falls within the domain of university communication and collaboration platforms. Business requirements include secure real-time messaging, AI-assisted information retrieval, and seamless integration with SJSU’s existing infrastructure. A **UML 2 Activity Diagram** can illustrate the overall process, such as user login, AI query, and message delivery. Process decomposition diagrams detail tasks like authentication via Google SSO, chat room creation, and AI response retrieval. A **Domain Class Diagram** would include classes like User, ChatRoom, Message, and AI-Response, with attributes such as UserID, RoomID, Content, and Timestamp. **State Machine Diagrams** can depict state transitions for key classes, such as User states (logged in, logged out, active, inactive) and Message states (sent, delivered, read).

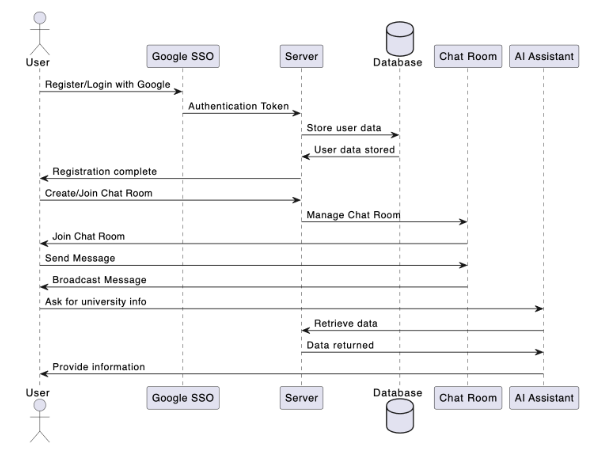


Fig 1: Use-Case Diagram

**3.2 Customer-oriented requirements**

The primary user groups for **Spartan Cove** include **SJSU students, faculty, and staff**. Use cases for students involve sending messages, querying the AI assistant, and joining chat rooms. Faculty use cases include creating group chats for classes, sharing materials, and communicating securely. Staff may use the platform for administrative communications and event announcements. The following table summarizes the key user groups and their use cases:

|  |  |
| --- | --- |
| **User Group** | **Use Cases** |
| Students | Send messages, join chat rooms, AI queries |
| Faculty | Create chat rooms, share resources, AI queries |
| Staff | Manage announcements, AI queries, chat support |

Table 1: User Groups and Their Use Cases

**3.3 System (or component) function requirements**

The **Spartan Cove** system’s core functional components include real-time chat, AI-powered assistance, user authentication, and chat room management. The table below outlines the high-level functional requirements:

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature** | **Inputs** | **Behaviour** | **Outputs** |
| User Authentication | SJSU email credentials (Google SSO) | Verifies user identity | Login success/failure |
| Real-Time Messaging | Text, multimedia | Sends messages instantly | Delivered messages |
| AI Assistance | User queries | Retrieves SJSU-specific information | Relevant AI responses |
| Chat Room Management | Chat room details | Creates/joins chat rooms | Chat room access |

Table 2: System Functional Requirements

**3.4 System performance and non-function requirements**

The **Spartan Cove** system must meet performance and non-functional requirements to ensure reliability and security. The following table summarizes these requirements:

|  |  |
| --- | --- |
| **Requirement Type** | **Description** |
| Performance | Support up to 5,000 concurrent users with low-latency messaging |
| Availability | 99.9% uptime with AWS auto-scaling across multiple zones |
| Security | End-to-end encryption, Google SSO, role-based access |
| Scalability | Handle growing user base and message volume |
| Compliance | Adhere to FERPA and university data privacy policies |

Table 3: System Performance and Non-Functional Requirements

**3.5 System behavior requirements**

The system behavior requirements focus on how the application responds to user actions and system states. **State diagrams** can capture behaviors such as user login/logout, message sending/delivery, and AI query processing. For instance, a user state diagram includes transitions like **Logged In → Active → Inactive → Logged Out**. A Message state diagram includes states like **Draft → Sent → Delivered → Read**. These diagrams help visualize system-level behaviors and interactions.

**3.6 Context and interface requirements**

The development and deployment of **Spartan Cove** will be supported by environments such as **AWS** for cloud hosting, **GitHub** for version control, and **Jenkins** for CI/CD. The system interfaces include a responsive web portal accessible on both desktop and mobile devices. The user interface will feature sections for **chat rooms, AI queries, user profiles, and settings**. The following table outlines key interface requirements:

|  |  |
| --- | --- |
| **Interface Element** | **Description** |
| Chat Interface | Real-time chat rooms with message input box |
| AI Assistant Interface | Query input field and response display |
| User Profile Interface | Profile details and avatar settings |
| Authentication Interface | Google SSO login screen |

Table 4: System Interface Requirements

**3.7 Technology and resource requirements**

**Spartan Cove** requires a set of modern technologies and resources to ensure functionality and scalability. The technologies include **MongoDB** for data storage, **Express.js** for backend API, **React.js** for the frontend, and **Node.js** for server-side operations. Real-time messaging is powered by **Socket.io**, and AI responses are provided by **xAI’s Grok 2**. Infrastructure is provisioned using **Terraform** on **AWS**, utilizing services like **EC2**, **Auto Scaling Groups**, and **VPCs**. Additional tools include **Google SSO** for authentication, **GitHub** for code management, and **Jenkins** for CI/CD.

**Chapter 4 System Design**

**4.1 System architecture design**

The **Spartan Cove** system architecture follows a **Full-Stack MERN (MongoDB, Express.js, React.js, Node.js)** design, ensuring scalability and efficient real-time communication. The architecture comprises a **frontend component** built with **React.js** for delivering a responsive user interface, a **backend component** using **Express.js** for handling API requests and business logic, and a **MongoDB** database for flexible and scalable data storage. Real-time messaging is facilitated by **Socket.io**, enabling WebSocket-based bidirectional communication. **Google SSO** provides secure authentication, and the system is hosted on **AWS infrastructure**, managed via **Terraform** for provisioning and scaling across multiple availability zones.

The system architecture for this project leverages AWS cloud services to provide a secure, scalable, and highly available infrastructure. The following diagram illustrates the core components and their interactions, including user authentication, request validation, backend compute services, and database storage.

**Key Elements of the Architecture:**

1. **Monitoring and Event Management**
   * **AWS Cloud Watch** is configured to monitor the health and performance of Front-End and Back-End Servers it is set to trigger automated alarms and notifications in case of server failure.
   * **AWS Event Bridge** is configured to detect instance failure events and trigger workflows to restart the affected instances.
2. **Backend and Frontend Services**
   * **EC2 Instances** handle business logic and processing. These instances are part of an **Auto Scaling Group** to maintain high availability and performance.
   * The backend services are deployed across multiple **Availability Zones** to ensure fault tolerance and reduce latency.
   * An Elastic Load Balancer is configured to distribute traffic evenly across EC2 instances ensuring high availability and efficient resource utilization.
3. **Database**
   * **MongoDB** stores application data, and access to it is tightly controlled using **IAM Roles** and **Resource-Based Policies** to restrict database writes to authorized backend servers only.
   * A read replica is configured for the primary database to offload read-intensive queries, enhancing performance and scalability.
4. **Networking and Security**
   * **Security Groups** are configured to control traffic between services, ensuring that only necessary ports are open.
   * The architecture incorporates a security-focused design by splitting resources across public and private subnets.

The diagram below provides a visual representation of the components discussed above.

A screenshot of a computer

Description automatically generated

Fig 2: System Infrastructure Architecture

**4.2 System data and database design**

The database design for **Spartan Cove** uses **MongoDB** to store users, messages, and chat room details. The primary collections include Users, Messages, ChatRooms, and AI-Responses, each with defined attributes like UserID, Username, Avatar, MessageContent, RoomID, and Timestamp. Relationships between these collections are established via references, such as linking messages to users and chat rooms. An **Entity-Relationship Diagram (ERD)** visualizes these relationships, illustrating how users can belong to multiple chat rooms and send various messages within those rooms.

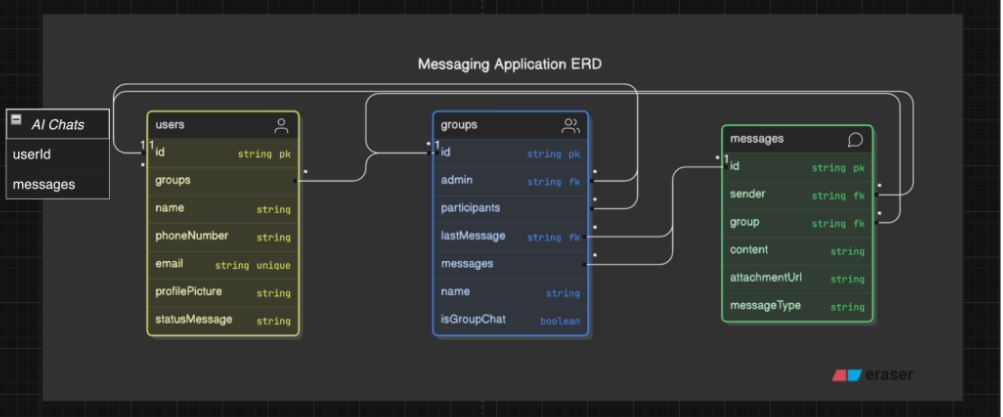


Fig 3: Database Schema

**4.3 System interface and connectivity design**

**Spartan Cove** interfaces with external systems like **Google SSO** for authentication and **xAI’s Grok 2** for AI responses. The system also supports integration with **GitHub** and **Jenkins** for CI/CD pipelines. User interfaces include a **responsive web portal** for chat functions, profile management, and AI queries. Connectivity is managed via **WebSocket** protocols (using **Socket.io**) for real-time messaging, ensuring instant communication and live updates between users and the server.

**4.4** **System user interface design**

The **Spartan Cove** user interface consists of a clean, responsive design that supports both desktop and mobile devices. The interface includes sections for **chat rooms**, **AI assistant queries**, and **user profiles** with avatars. The chat interface features a message input box, real-time message delivery, and typing indicators. The AI assistant interface allows users to submit queries and view responses. The operation flow ensures intuitive navigation between login, chat rooms, and user settings, with a consistent layout and easy-to-access features.

**4.5** **System component API and logic design**

The backend API, built with **Express.js**, provides endpoints for user authentication, message handling, chat room management, and AI queries. Key components include APIs for **sending messages**, **retrieving chat history**, **creating chat rooms**, and **fetching AI responses**. The logic for each endpoint ensures secure data handling and validation. The API is documented with **Swagger** for clarity. **Sequence diagrams** illustrate interactions between components, such as user login, message sending, and AI response retrieval, ensuring clear logic flow and component interaction.

**4.6** **Design problems, solutions, and patterns**

Several design challenges were addressed during the development of **Spartan Cove**. To ensure **data security and privacy**, we implemented **end-to-end encryption** and **Google SSO** for authentication. For **scalability**, the system uses **AWS Auto Scaling Groups** and a **decentralized database** architecture. The need for real-time communication was met using **WebSocket protocols** with **Socket.io**. To handle complex UI updates, **React.js** was chosen for its component-based architecture. These solutions, combined with design patterns like **Model-View-Controller (MVC)** and **Repository Pattern**, ensure a robust, maintainable, and scalable system.

**Chapter 5 System Implementation**

**5.1 System implementation summary**

The **Spartan Cove** project has been successfully implemented with core functionalities including secure Google SSO authentication, real-time messaging, AI-powered assistance, and user profile management. The system follows a **Full-Stack MERN architecture**, ensuring a responsive and scalable chat platform for SJSU students and faculty. Real-time communication is enabled through **Socket.io**, and the AI assistant powered by **xAI’s Grok 2** provides accurate, university-specific information. The application has undergone system testing and is now ready for deployment, offering a robust, secure, and user-friendly communication tool tailored to the SJSU community.

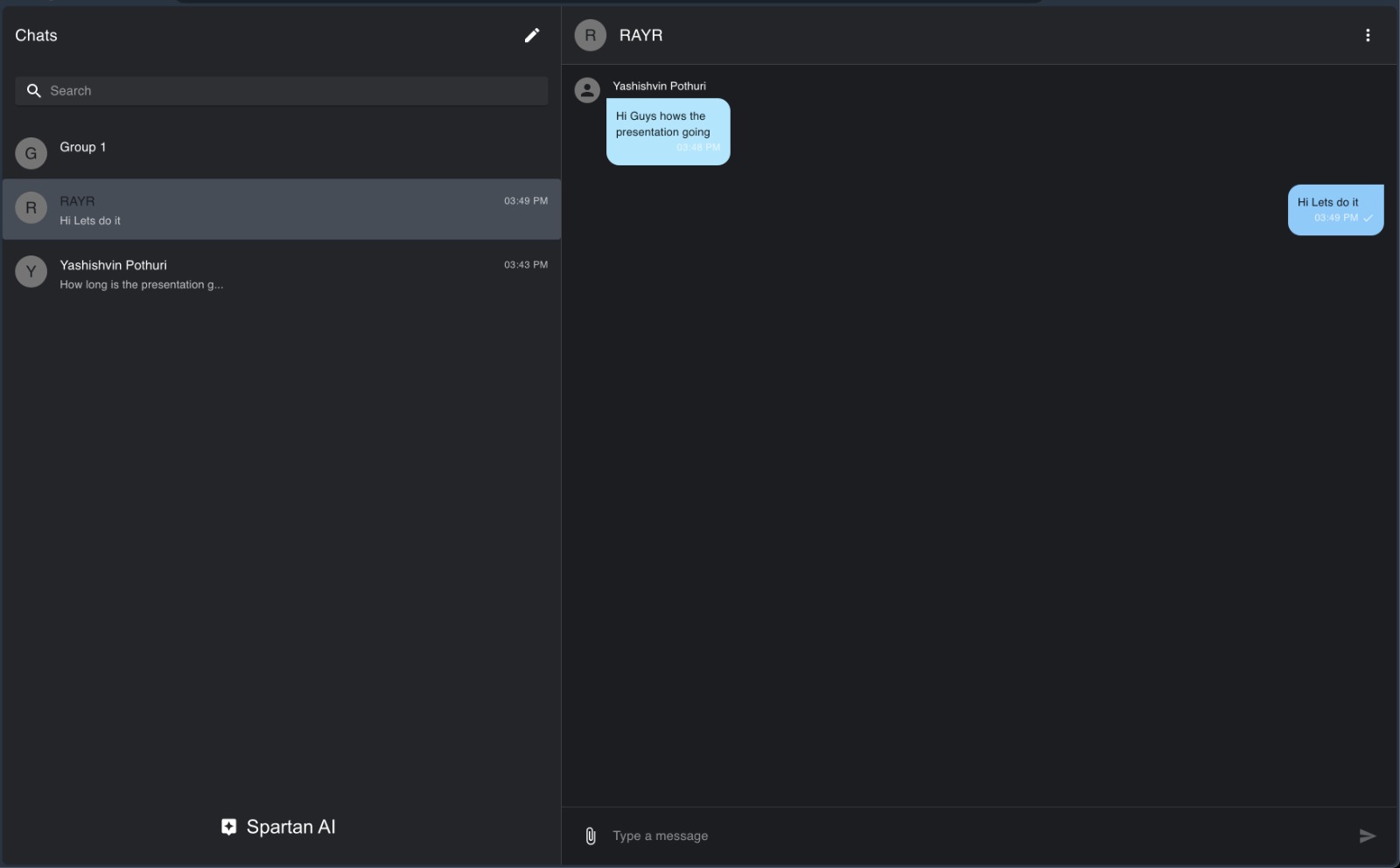


Fig 4: Chat Demo

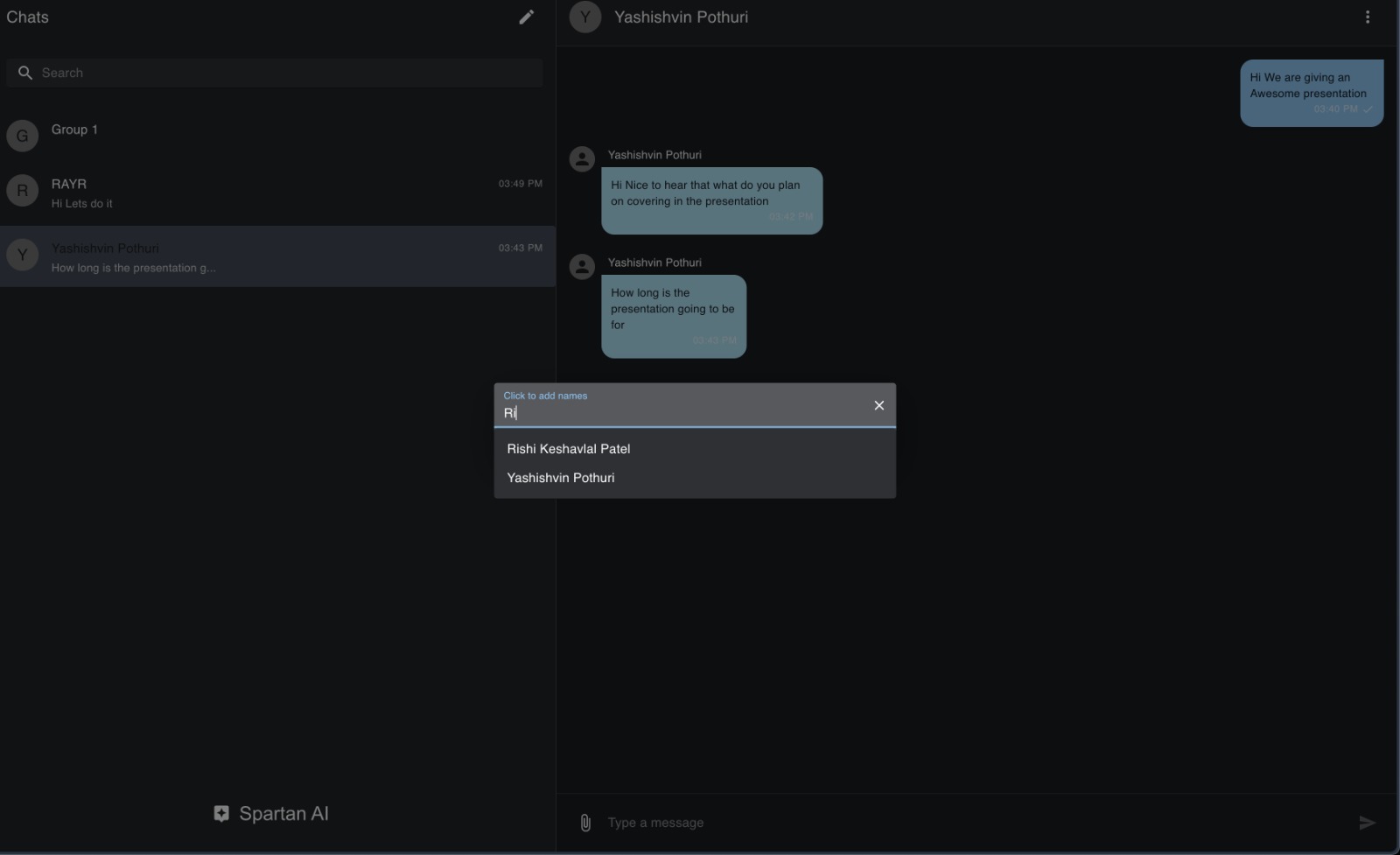


Fig 5: Adding a New User

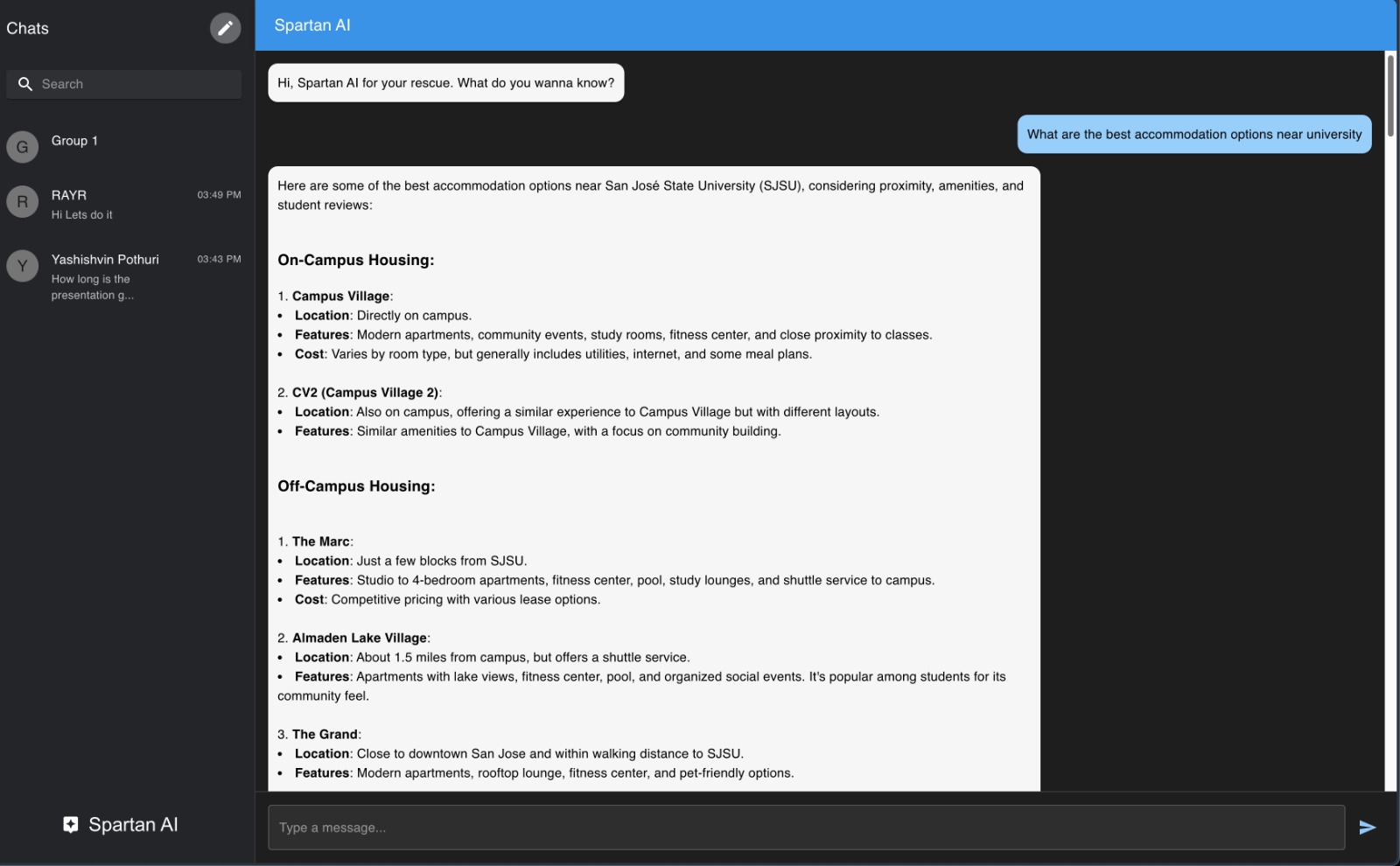


Fig 6: Spartan AI Chatbot Demo

**5.2 System implementation issues and resolutions**

During implementation, several challenges arose, including **Google SSO integration**, **real-time communication consistency**, and **scalability** concerns. For Google SSO, issues with token verification were resolved by implementing strict SSL/TLS encryption and validating tokens server-side. Real-time messaging occasionally faced delays, which were mitigated by optimizing **WebSocket connections** and ensuring persistent connections with **Socket.io**. To address scalability, **AWS Auto Scaling Groups** were configured to handle high traffic loads efficiently, and **MongoDB indexing** was optimized to improve database performance. These solutions ensured the system met performance, security, and reliability requirements.

**5.3 Used technologies and tools**

The **Spartan Cove** project utilized a set of carefully selected technologies and tools to ensure efficient development, security, and scalability:

1. **Frontend**:
   * **React.js**: For building a responsive and dynamic user interface. Chosen for its component-based architecture and real-time UI updates.
2. **Backend**:
   * **Node.js** and **Express.js**: For handling server-side operations and API requests. Selected for their performance with asynchronous processing.
3. **Database**:
   * **MongoDB**: A NoSQL database offering flexible schemas and scalability. Used for storing user data, messages, and chat room details.
4. **Real-Time Communication**:
   * **Socket.io**: For WebSocket integration, enabling instant message delivery, typing indicators, and real-time updates.
5. **AI Integration**:
   * **xAI’s Grok 2**: Provides real-time AI responses tailored to SJSU-related queries.
6. **Authentication**:
   * **Google SSO**: For secure and seamless user authentication. Ensures only verified SJSU users can access the platform.
7. **Infrastructure**:
   * **AWS (EC2, Auto Scaling Groups, VPC)**: For hosting the application with high availability and network segmentation.
   * **Terraform**: For provisioning and managing AWS resources efficiently.
8. **CI/CD and Version Control**:
   * **GitHub**: For version control and collaborative development.
   * **Jenkins**: For continuous integration and deployment (CI/CD) pipelines.

These technologies were chosen for their reliability, scalability, and ability to integrate seamlessly, ensuring the Spartan Cove application meets both technical and user requirements.

**Chapter 6 System Testing and Experiment**

**6.1 Testing and experiment scope**

The testing process for **Spartan Cove** focused on ensuring functionality, security, performance, and user experience. The scope included both **component-level** and **system-level** testing. At the component level, tests were conducted for individual modules such as **user authentication**, **real-time messaging**, and **AI assistant queries**. System-level tests involved end-to-end validation of interactions between these modules to ensure seamless performance. The primary objectives were to verify **Google SSO integration**, **Socket.io real-time messaging**, and the **AI-powered assistant** response accuracy. Test criteria included **functionality**, **performance (response time under load)**, **security (encryption and access control)**, and **usability**. The test process involved creating unit tests, integration tests, and user acceptance tests (UAT) to ensure all features met the requirements. The following table summarizes the key test criteria:

|  |  |  |
| --- | --- | --- |
| **Test Focus** | **Objective** | **Criteria** |
| Functionality | Ensure core features work as intended | 100% feature coverage |
| Performance | Measure response time and system load handling | Response time < 2 seconds |
| Security | Verify data privacy and secure access | No vulnerabilities detected |
| Usability | Ensure intuitive user experience | Positive user feedback |

Table 5: Testing Criteria

**6.2 Testing and experiment approaches**

The testing approach for **Spartan Cove** followed a structured **test plan** that included **unit testing**, **integration testing**, **system testing**, and **user acceptance testing (UAT)**. For unit testing, **Jest** was used to validate individual components like **Google SSO authentication**, **Socket.io real-time communication**, and **AI assistant queries**. **Integration testing** was conducted using **Mocha** to ensure seamless interaction between the frontend, backend, and database. **System testing** involved end-to-end verification of the full application flow, including login, messaging, AI queries, and profile management. For performance testing, **Apache JMeter** was used to simulate multiple concurrent users and measure system response times. Security tests focused on verifying **SSL/TLS encryption** and protection against vulnerabilities like **SQL injection** and **XSS attacks**. The following table summarizes the test methods and tools used:

|  |  |  |
| --- | --- | --- |
| **Test Method** | **Tool Used** | **Focus** |
| Unit Testing | Jest | Individual components |
| Integration Testing | Mocha | Component interactions |
| System Testing | Manual/Automated | End-to-end workflows |
| Performance Testing | Apache JMeter | Load and response time |
| Security Testing | OWASP ZAP | Vulnerability checks |
| User Acceptance Testing | Manual (User Feedback) | User experience and functionality |

Table 6: Test Methods and Tools

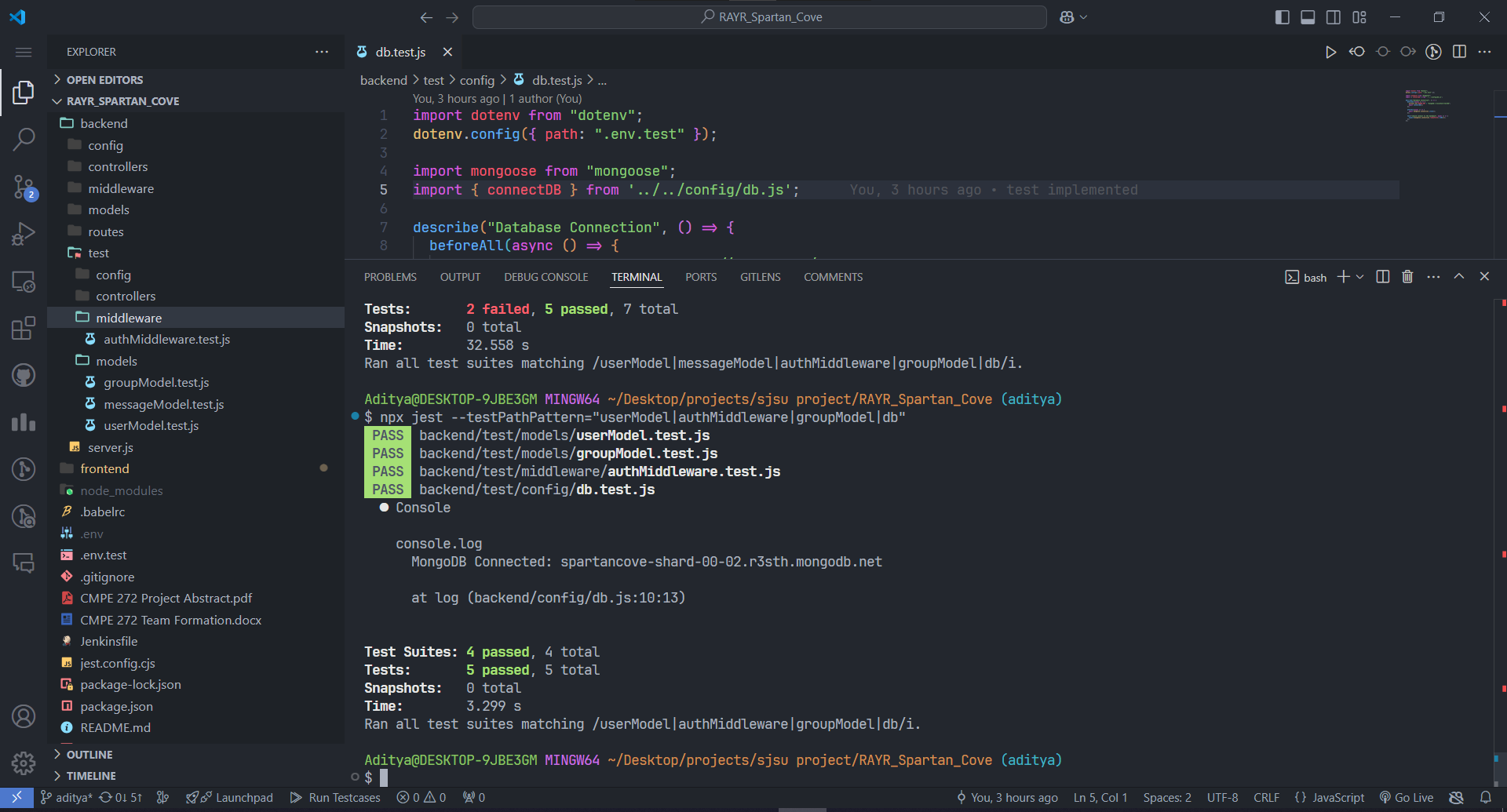


Fig 7: Testing File Implemented

**Chapter 7 Conclusion and Future Work**

**7.1 Project summary**

The **Spartan Cove** project successfully achieved its goal of delivering a secure, AI-enabled chat application tailored for the SJSU community. The platform integrates **Google Single Sign-On (SSO)** for authentication, **Socket.io** for real-time messaging, and **xAI’s Grok 2** for AI-powered assistance. The system was implemented using a **Full-Stack MERN architecture** and deployed on **AWS infrastructure** with Terraform for scalability and high availability. Comprehensive testing ensured the application met functionality, performance, security, and usability criteria. The final deliverables included a working prototype, detailed documentation, and a live presentation. The project provided valuable insights into real-time communication, secure system design, and AI integration. Key lessons learned include the importance of robust encryption for user privacy, optimizing real-time messaging for performance, and ensuring seamless user authentication. Collaboration, consistent testing, and iterative development played crucial roles in the project's success.

**7.2 Future Work**

Future enhancements for **Spartan Cove** aim to expand its capabilities and improve user experience. Potential directions include developing a **mobile application** to complement the web platform, enhancing accessibility for on-the-go communication. Additional integrations, such as **calendar synchronization** for class schedules and **event notifications**, can provide even more value to users. Improving the AI assistant by incorporating **natural language understanding (NLU)** and expanding its knowledge base will offer more accurate and personalized responses. To further enhance security and scalability, implementing features like **multi-factor authentication (MFA)** and **load balancing** across additional servers is planned. Finally, gathering ongoing user feedback and incorporating **machine learning-based personalization** can ensure the platform continues to evolve according to the needs of the SJSU community.