JOINT-DEV

Developers study Group

SUBMITTED BY: VINCENT KIBET KORIR.

REG NO. : N11/3/0078/020

INSTITUTION: LAIKIPIA UNIVERSITY.

COURSE: COMPUTER SCIENCE.

SUPERVISOR: Dr. ALEX KIBET.

IN PARTIAL FULFILLMENT OF THE REQUIREMENT OF THE DEGREE PROGRAM

January 2024.

i. DECLARATION.

I hereby declare that, the project is the the outcome of my own effort.
The project is submitted to the University of Laikipia. For the partial fulfillment of the Bachelor of Science in Computer Science.

I also declare that these project report have not been previously or will be submitted to any other institution.

Name : Vincent Kibet Korir.

Signature: ____________________.

Date: 28 - Nov - 2023.

ii. **Dedication:**

This project is dedicated to all learners around the world who strive for knowledge and growth, and to the educators and mentors who tirelessly support and guide them on their educational journey.

iii. **Acknowledgement:**

We extend our heartfelt gratitude to all those who contributed to the realization of this project:

- Our esteemed advisors and mentors for their invaluable guidance, encouragement, and support throughout the development process.
- The participants who generously shared their insights, feedback, and suggestions, shaping the direction and features of the Study Group Project.
- The developers and designers who dedicated their expertise, creativity, and hard work to bring this project to fruition.
- The educational institutions and organizations that provided resources, collaboration opportunities, and testing environments.
- Our families, friends, and loved ones for their unwavering support, understanding, and encouragement during the project's development.
- Lastly, to the global community of learners and educators who inspire us with their passion for learning and collaboration, this project is a tribute to your dedication and enthusiasm.

iv. **Abstract:**

JOINT-DEV is a collaborative web application powered by Django, facilitating real-time group discussions among developers. Users register, log in, and create rooms centered around specific topics. They engage in live chat, with options for CRUD operations on rooms and messages. The activity panel showcases recent room interactions, while a search feature enables quick navigation. Each room offers integrated Google Meet functionality for face-to-face interactions, alongside a console window for executing code using Code-Mirror. Additionally, users can access a repository for shared documents, facilitating seamless collaboration. The About page hosts user profiles and bios, leveraging Django admin as the central database repository. JOINT-DEV fosters efficient communication and collaboration within developer communities, enhancing productivity and knowledge sharing.

Table of Contents

List of Tables

List of Figures

CHAPTER 1.

1. Introduction

In the fast-paced world of software development, effective communication and collaboration among developers are crucial for success. JOINT-DEV is a dynamic web application designed to facilitate group discussions and collaboration among developers in real-time. Built on the Django framework, JOINT-DEV offers a comprehensive platform where developers can register, create rooms, discuss topics, share code snippets, and even engage in face-to-face meetings seamlessly.

1.1 Background

Traditional communication tools often lack the specialized features required for developers to efficiently collaborate on projects. Email threads can become cluttered, and instant messaging platforms may not provide the necessary environment for code sharing and discussion. JOINT-DEV addresses these limitations by providing a dedicated platform tailored specifically to the needs of developers.

With JOINT-DEV, developers can create rooms dedicated to specific topics, fostering focused discussions and collaboration. Real-time messaging capabilities ensure that developers can communicate efficiently, brainstorm ideas, and troubleshoot problems effectively. Additionally, features such as code execution within the platform and integration with external tools like Google Meet enhance the collaboration experience further.

1.2 Problem Statement

Despite the abundance of communication tools available, developers often struggle to find platforms that cater specifically to their needs. Existing solutions may lack essential features such as real-time collaboration, code execution, or integration with project management tools. As a result, developers may resort to using multiple disjointed tools, leading to inefficiencies and communication gaps.

The problem lies in the absence of a comprehensive platform that seamlessly integrates essential features for developer collaboration. Developers need a single platform where they can discuss ideas, share code, coordinate meetings, and manage project documents effectively. Addressing this gap requires the development of a specialized solution tailored to the unique requirements of developers.

JOINT-DEV aims to fill this void by providing a feature-rich platform specifically designed to streamline communication and collaboration among developers. By offering a unified environment for discussions, code sharing, and project management, JOINT-DEV seeks to enhance productivity and facilitate smoother workflows for development teams.

1.3 Aim of the Study

The primary aim of this study is to develop JOINT-DEV, a Group Discussion web application tailored to the needs of developers, with a focus on enhancing collaboration, communication, and productivity within development teams.

1.3.1 General Objectives

- To create a user-friendly web application using the Django framework that facilitates real-time group discussions among developers.
- To integrate essential features such as user registration, login, room creation, CRUD functionality for rooms and messages, activity tracking, topic listing, Google Meet integration, code execution, and document management.
- To provide a seamless and intuitive interface for developers to engage in discussions, collaborate on projects, run code, and manage project documents effectively.

1.3.2 Specific Objectives

- To design and implement a robust authentication system for user registration and login.
- To develop functionality for users to create, join, and manage discussion rooms, including CRUD operations for rooms and messages.
- To integrate Google Meet API for scheduling and joining video meetings within discussion rooms.
- To implement a code execution feature using CodeMirror, allowing users to run and execute code within the application.
- To incorporate document management capabilities for storing, accessing, and managing project documents within the application.

1.4 Justification of the Study

The development of JOINT-DEV is justified by the increasing demand for specialized collaboration tools tailored to the needs of developers. By providing a comprehensive platform that integrates essential features such as real-time messaging, code execution, document management, and video conferencing, JOINT-DEV aims to streamline communication and collaboration within development teams, ultimately enhancing productivity and project outcomes.

Furthermore, the study addresses the limitations of existing communication platforms by offering a dedicated solution specifically designed to meet the unique challenges faced by developers working in virtual environments.

1.5 Scope of the Study

The scope of this study encompasses the design, development, and implementation of JOINT-DEV, focusing on the following key features:

- User registration and authentication
- Room creation and management
- Real-time messaging functionality
- Integration of Google Meet for video conferencing
- Code execution using CodeMirror
- Document management capabilities

The study will also include testing, validation, and refinement of the application to ensure usability, functionality, and performance.

1.6 Limitations of the Study

While JOINT-DEV aims to address the communication and collaboration needs of developers, it is important to acknowledge certain limitations:

- The study may face constraints in terms of time, resources, and expertise available for development.
- Integration with third-party APIs, such as Google Meet, may be subject to limitations or changes imposed by the API providers.
- The effectiveness of the application in meeting the diverse needs of development teams may vary depending on factors such as team size, project complexity, and organizational context.
- The study may not cover all possible use cases or scenarios encountered by developers in their day-to-day work.

CHAPTER 2.

2. Literature Review

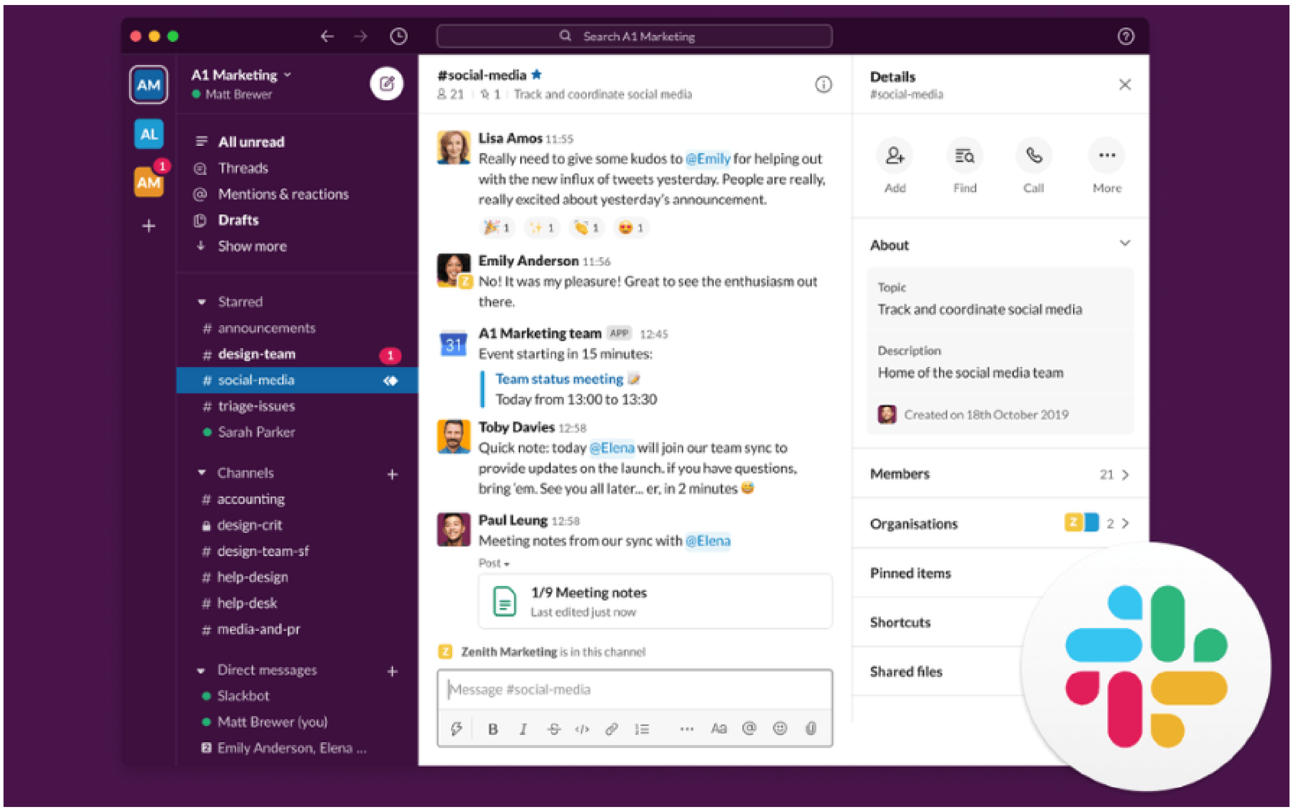
2.1 Introduction

In the realm of online education and collaborative learning, the utilization of web-based platforms has become increasingly prevalent. These platforms aim to bridge the gap between traditional face-to-face interactions and remote learning environments, offering features that facilitate real-time communication, resource accessibility, and community engagement. This literature review explores the efficacy of such platforms by examining relevant case studies that demonstrate their application in various educational contexts.

2.2 Case Studies

2.2.1 Case 1 – Slack

Description: Slack is a widely-used collaboration hub that offers real-time messaging, file sharing, and integration with various third-party services. It allows users to create channels for different topics or teams, facilitating organized discussions and workflow management. Slack's key features include:



Lisa Amos 11:55
Really need to give some kudos to @Emily for helping out with the new influx of tweets yesterday. People are really, really excited about yesterday's announcement.

Emily Anderson 11:56
No! It was my pleasure! Great to see the enthusiasm out there.

A1 Marketing team APP 12:45
Event starting in 15 minutes:
Team status meeting
Today from 13:00 to 13:30

Toby Davies 12:58
Quick note: today @Elena will join our team sync to provide updates on the launch. if you have questions, bring 'em. See you all later... er, in 2 minutes

Paul Leung 12:58
Meeting notes from our sync with @Elena
Post
1/9 Meeting notes
Last edited just now

Zenith Marketing is in this channel

Message #social-media

Topic
Track and coordinate social media

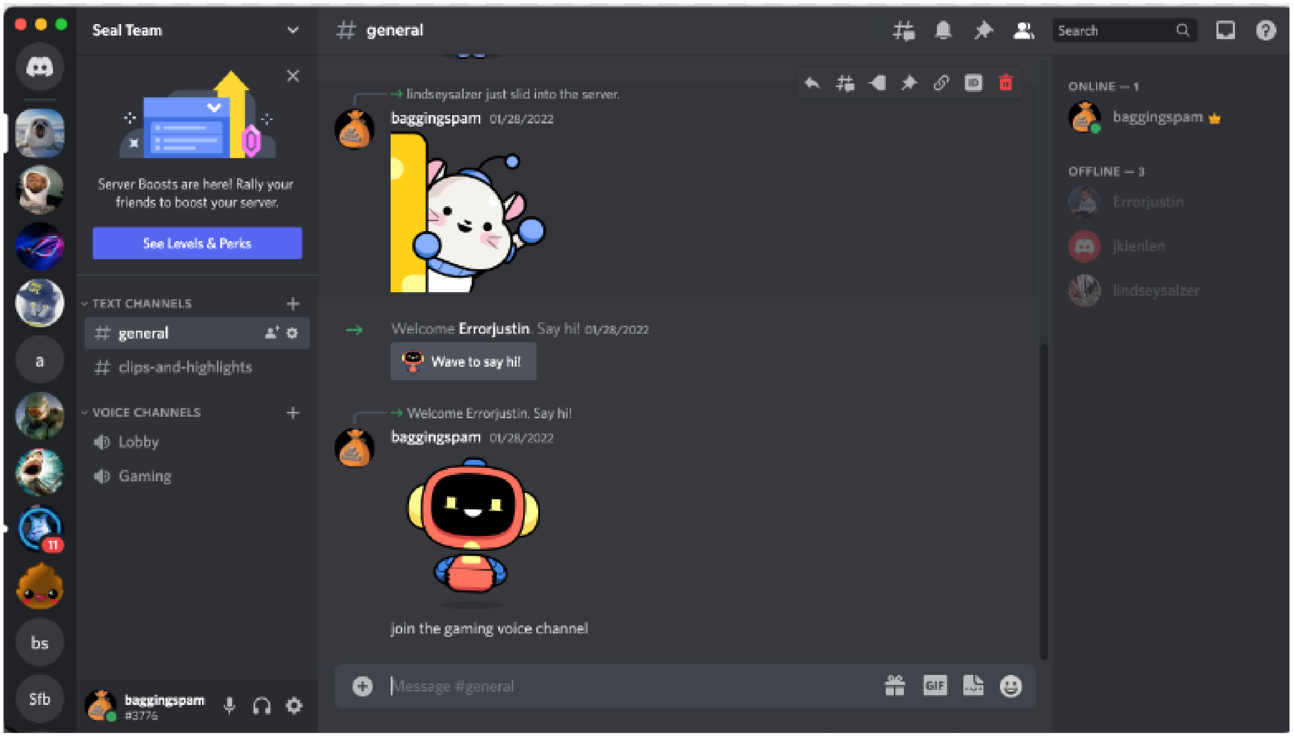
Description
Home of the social media team

Created on 18th October 2019

Figure 1: slack case study

2.2.2 Case 2 – Discord

Description: Originally designed for gamers, Discord has evolved into a multifunctional communication platform suitable for various communities and groups. It offers text, voice, and video chat functionalities, along with customizable servers and roles for managing users' access levels. Discord's robust infrastructure supports large communities and provides features like moderation tools, bots, and extensive customization options, making it ideal for both casual and professional use.



→ lindseysalzer just slid into the server.
baggingspam 01/28/2022

→ Welcome **Errorjustin**. Say hi! 01/28/2022
Wave to say hi!

→ Welcome Errorjustin. Say hi!
baggingspam 01/28/2022

join the gaming voice channel

Figure 2: Discord case study

2.2.3 Case 3: Canvas

Background: Canvas is a learning management system (LMS) widely used in educational institutions for course management, online learning, and collaboration among students and instructors. The Study Group Project, on the other hand, is a dedicated web-based platform designed specifically for collaborative learning, offering features such as real-time chat, resource accessibility, video conferencing, and activity tracking.

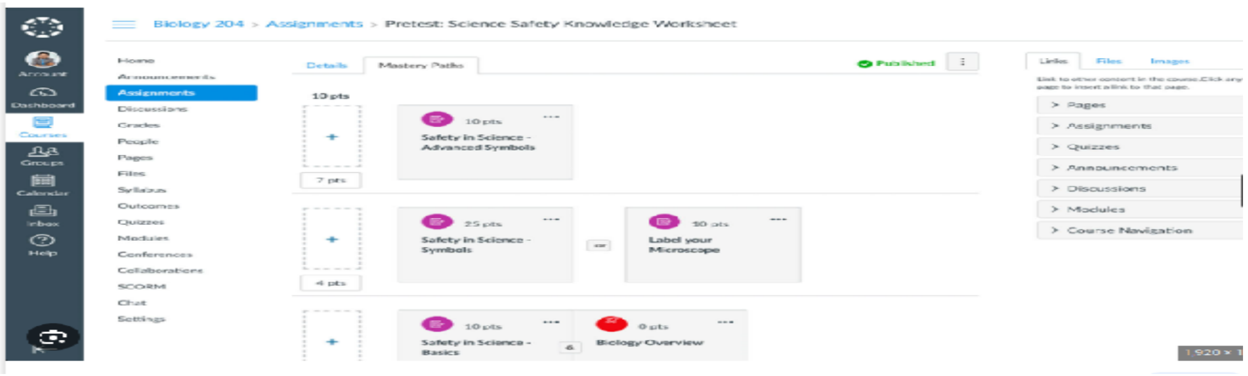


Figure 3: canvas case study

2.2.4 Case 4 – GitHub

Description: GitHub is a widely-used platform for version control and collaboration among developers. It allows users to host and review code, manage projects and workflows, and facilitate discussions through issues and pull requests. GitHub's key features include:

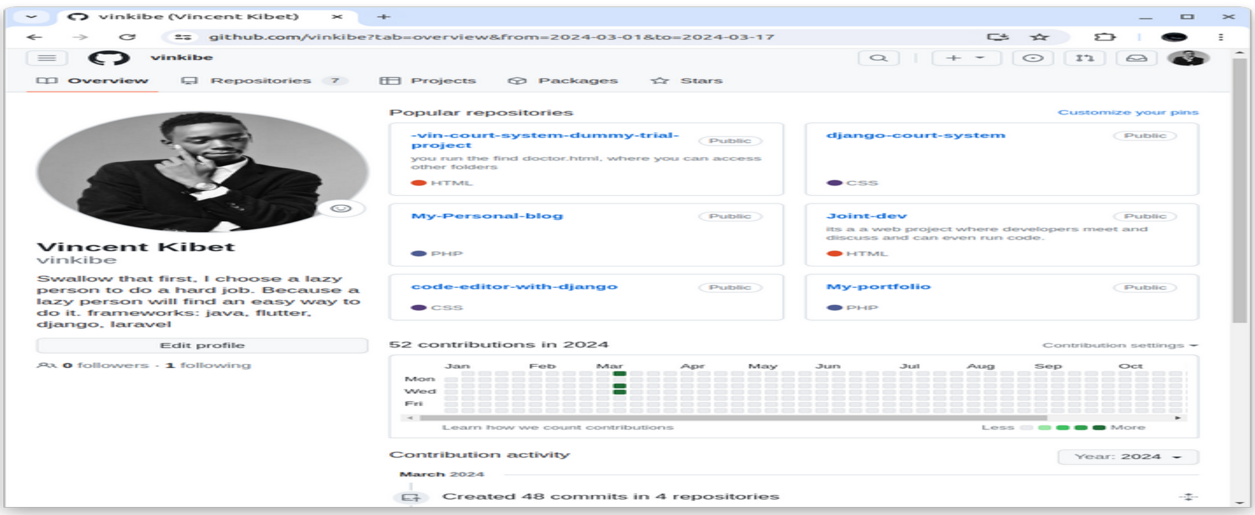

vinkibe (Vincent Kibet)
github.com/vinkibe?tab=overview&from=2024-03-01&to=2024-03-17
vinkibe
Overview
Repositories 7
Projects
Packages
Stars
Popular repositories
Customize your pins
-vin-court-system-dummy-trial-project
Public
you run the find doctor.html, where you can access other folders
HTML
django-court-system
CSS
My-Personal-blog
PHP
Joint-dev
its a a web project where developers meet and discuss and can even run code.
HTML
code-editor-with-django
CSS
My-portfolio
PHP
Vincent Kibet
vinkibe
Swallow that first, I choose a lazy person to do a hard job. Because a lazy person will find an easy way to do it. frameworks: java, flutter, django, laravel
Edit profile
0 followers · 1 following
52 contributions in 2024
Contribution settings
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Mon
Wed
Fri
Learn how we count contributions
Less
More
Contribution activity
Year: 2024
March 2024
Created 48 commits in 4 repositories

Figure 4: Github case study

2.3 Comparison of Features of Reviewed Cases

Feature	Case 1 (Slack)	Case 2 (Discord)	Case 3 (Canvas)	Case 4 (GitHub)	Your Solution (JOINT-DEV)
Real-time messaging					
Video Meetings					
File Sharing					
Code Execution					

JOINT-DEV, integrates various features from the reviewed cases to create a comprehensive platform tailored for developers. By combining real-time messaging, video meetings, file sharing, code execution platform etc. JOINT-DEV aims to provide developers with a unified environment for efficient communication and collaboration on their projects.

CHAPTER 3.

3. Methodology

3.1 Introduction

The methodology for developing the JOINT-DEV project involved a systematic approach to ensure the successful implementation of all features and functionalities. This methodology encompasses various stages, from initial planning to deployment and maintenance.

3.2 Software Process Model Adapted

The process model describes the sequence of phases for the entire lifetime of developing the software. It covers all the activities from the initial commercial idea to the final de-installation or dissembling of the product. **JOINT-DEV** development adopted incremental process model where the system is designed, implemented and tested as a series of incremental builds until the product is finished. Each stage of the incremental model is coded and then integrated into the structure which is tested as a whole.

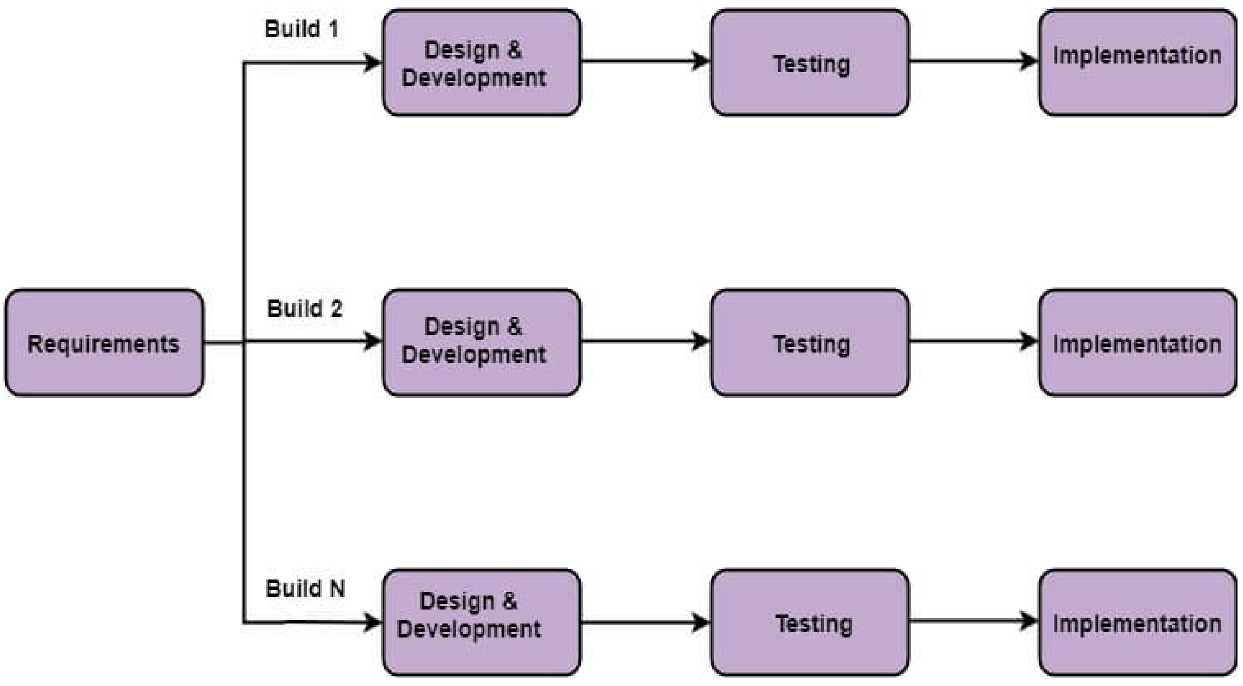


Fig: Incremental Model

Figure 5: Incremental model

Advantage of Incremental Model

- ❍Errors are easy to be recognized.
- ❍Easier to test and debug
- ❍More flexible.
- ❍Simple to manage risk because it handled during its iteration.
- ❍The Client gets important functionality early.

Disadvantage of Incremental Model

- ❍Need for good planning
- ❍Total Cost is high.
- ❍Well defined module interfaces are needed.

3.3 Requirements Gathering Process (Data Collection)

To achieve a great user experience the user should be able to achieve his/her objective when using system. This begins with the developer knowing and understanding what the user wants. A requirement is statement about a projected artifact that stipulates what it should do or even how it should do it. Joint-Dev mission is to have SMART objectives for effective implementation. That is possible to specify, measure, attainable, realistic and able to test them.

3.3.1 Target Users (Population)

The target users of the JOINT-DEV project primarily include developers, software engineers, and anyone involved in collaborative software development activities. These users typically possess technical expertise and engage in group discussions, code collaboration, and project management tasks. The application caters to individuals or teams working on software development projects who require a platform for real-time communication, code sharing, and document management.

3.3.2 Requirements Collection Tools (Data Collection Tools)

To gather requirements effectively, various tools and techniques can be employed:

1. **Guided interview:**

A guided interview was conducted with the various developers. A phone call to illustrate on some issues aided in making this tool effective given the geographical limitations. It was aimed at understanding the operations involved during reservation process and promotion techniques. This was aimed on understanding the challenges they face as a result of failures of the present system. What they saw hasn't been addressed and the various issues that they felt if addressed would improve some of their practices and consequently improve on their collaboration.

Advantages for using interview;

- Face-to-face interviews helped with more accurate screening and clarifications during data collection
- It allows in notation of non-verbal ques. This enabled the interviewer to accurately acquire the issues the respondents put emphasis on.
- It allowed for more detailed information to be asked and collected from the respondents and hence acquire a greater understanding of the industry.
- It is good in solving ambiguities by clarification of some elements of the system requirements that would lead to ambiguities.

Challenges faced during requirements gathering;

- Cost inefficient to travel to respondent destinations and phone calls.

a) Observation

being a developer, I observed that there's a challenge when developers need to understand or develop something together they have to meet physically since there's no tool to cater for there needs especially when doing group projects. JOINT-DEV was to address the issue efficiently.

Observation methods advantages;

- It is more direct, accurate and also a very reliable method to collect the requirements.
- Enabled clear identification of the problem by making an in depth analysis of the problem faced in the dairy industry.
- Proved to be less demanding in nature which made it less biased

Observation method challenges;

- Complete answers to the problems and issues experienced by customers and hotel management couldn't be obtained by observation alone.
- It involved a lot of time as the observer had to wait for an event to happen to study it.

3.3.3 Analysis and Results of Requirements Collection (Results of Data Collection)

The analysis of the requirements collection process yields valuable insights into the needs and preferences of the target users. Based on the data collected through stakeholder interviews, surveys, focus groups, and prototyping sessions, the following key findings emerge:

1. **Feature Prioritization**: Developers prioritize features such as real-time messaging, code collaboration, document management, and integration with external tools like Google Meet for video conferencing.
2. **User Interface Preferences**: Users prefer a clean and intuitive user interface that facilitates easy navigation between rooms, topics, and discussions. They emphasize the importance of responsive design to ensure accessibility across devices.
3. **Security and Privacy**: Users express concerns regarding data security and privacy, especially when sharing sensitive code or project-related information. They expect robust authentication mechanisms and secure data transmission protocols.
4. **Collaborative Tools Integration**: Integrating third-party tools such as Google Meet for video conferencing and CodeMirror for code execution enhances the collaborative experience and streamlines workflow.

CHAPTER 4.

4. System Design

4.1 Introduction

JOINT-DEV is a web application designed to facilitate collaborative development through real-time communication, code execution, document management, and video conferencing. It caters to developers by enabling project discussions, code reviews, and streamlined workflow within a secure, web-based environment.

4.2 Software Specification

4.2.1 Functional Requirements

User Management:

•User Registration and Login: Users can register for accounts with usernames, passwords, and optional profile bios. Login functionality verifies credentials and grants access to the application.

•User Roles: Implement different user roles (e.g., Admin, Developer) to manage user permissions.

Group Management:

•Room Creation: Developers can create new rooms with unique names and specify topics for discussion.

•Room Deletion: The room creator can delete the room, removing all associated discussions, code snippets, documents, and meeting details.

Real-Time Communication:

•Chat Functionality: Implement real-time chat features within rooms, allowing developers to exchange messages instantly.

•Thread Management (Optional): Consider incorporating threading capabilities for focused conversation organization within rooms.

Collaborative Development:

•Code Execution: Integrate an in-room console window with CodeMirror or a similar code editor to enable developers to execute code snippets within the application.

•Document Management: Develop a document repository system within each room. This includes:

•Document Uploads and Downloads: Developers can upload files for collaboration.

•Document Viewing: Provide document viewing capabilities based on file type (text, code, images, etc.).

- Collaborative Editing (Optional): Consider exploring real-time collaborative editing tools for specific document types to enhance teamwork.
- Document Deletion: Developers can delete documents within their room.
- Version Control (Optional): Integrating a lightweight version control system within the document repository would enable version tracking and potential rollbacks.

Video Conferencing Integration:

- Google Meet Integration: Implement seamless integration with Google Meet for real-time video conferencing capabilities.
- Meeting Scheduling: Allow developers to schedule Google Meets directly within rooms.
- Meeting Reminders: Send automated email reminders before scheduled Google Meets.

Search Functionality:

- Room Search: Enable searching for specific rooms based on room name or topic.

Activity Feed (Optional):

- Recent Activity Display: On the home page, consider displaying a component highlighting recent discussions, code snippets, document activity, or upcoming meetings within rooms.

4.2.2 Non-Functional Requirements

Performance:

- Real-time Chat:Ensure seamless, low-latency real-time chat performance.
- Code Execution: Optimize the in-room console's performance for efficient code execution.
- Document Management: Design the document repository and retrieval processes to be efficient and responsive.

Availability:

- Strive for high uptime: Implement redundancy and monitoring mechanisms to ensure application availability.
- Error Handling and Recovery: Design comprehensive error handling and recovery strategies to gracefully handle unexpected situations and maintain system stability.

Usability:

- Intuitive UI: Create an intuitive and user-friendly interface that caters to developers of varying technical backgrounds.
- Clear Navigation: Implement clear and consistent navigation structures to allow users to easily find the features they need.

•Responsiveness: Design the application to be responsive across different devices (desktop, mobile, tablets).

Security:

•User Authentication and Authorization: Implement robust authentication and authorization mechanisms to ensure controlled access to user data, code snippets, and documents.

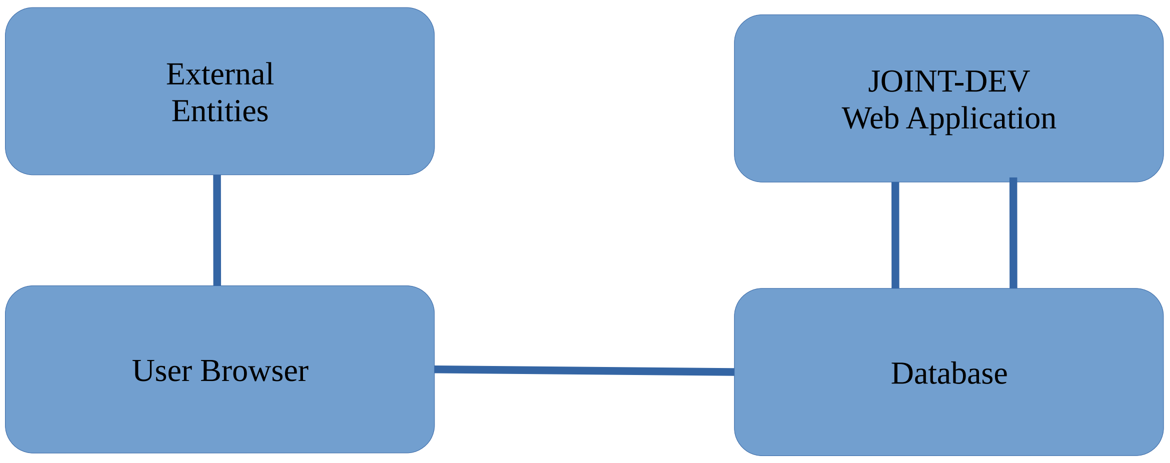
•Data Encryption: Encrypt sensitive data such as passwords, messages, and documents at rest and in transit.

•Regular Updates: Maintain the application's security by applying regular updates to address potential vulnerabilities.

4.3 System Modeling (UML Diagrams)

4.3.1 Context Diagrams:

A context diagram provides a high-level view of the system and its surrounding environment. It shows the interactions between the system and external entities.

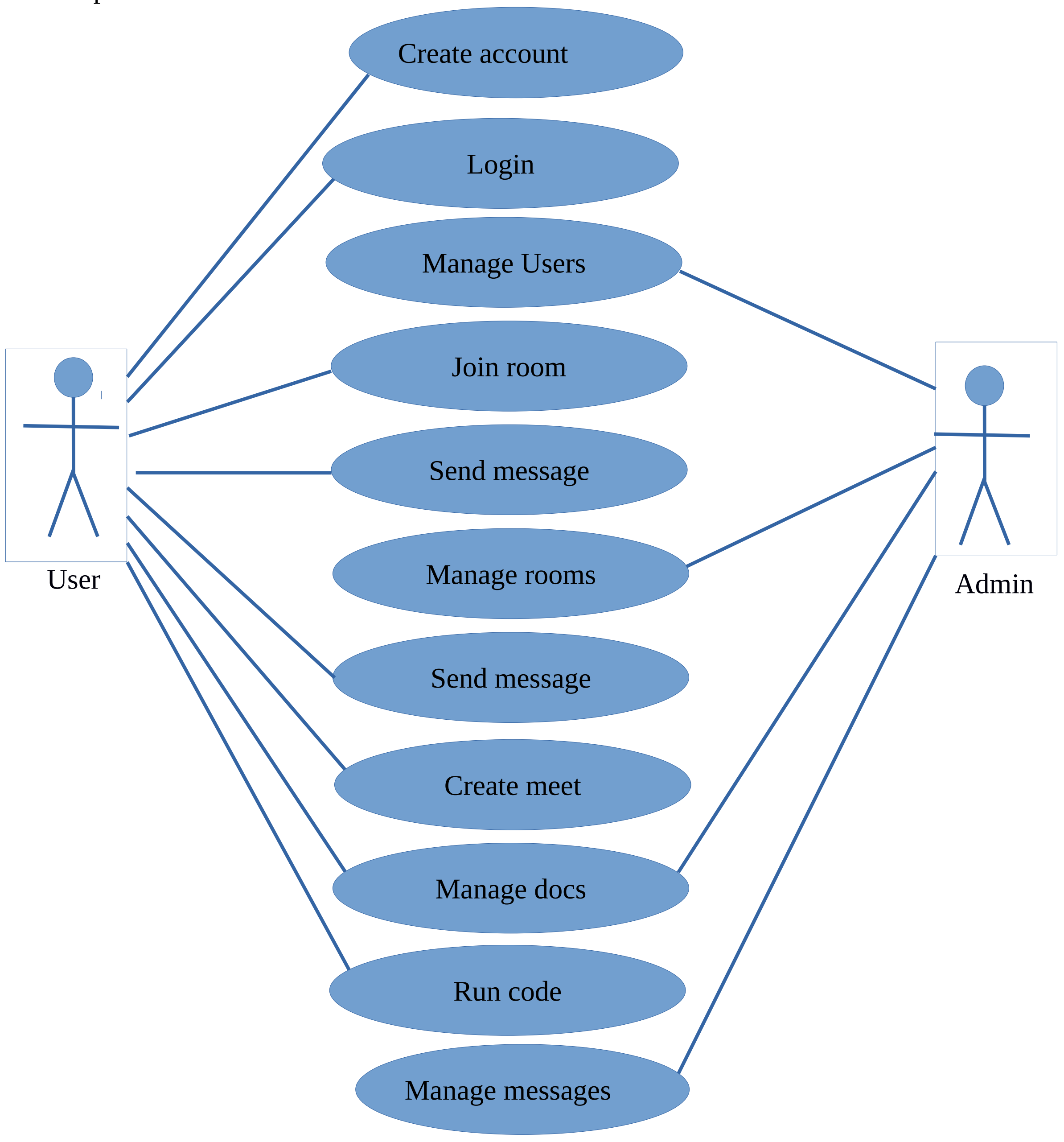


In this context diagram above:

- External entities interact with the JOINT-DEV web application.
- The main external entity shown here is the "User Browser," representing users accessing the system through their web browsers.
- The JOINT-DEV web application interacts with the database to store and retrieve data such as user information, room details, messages, etc.

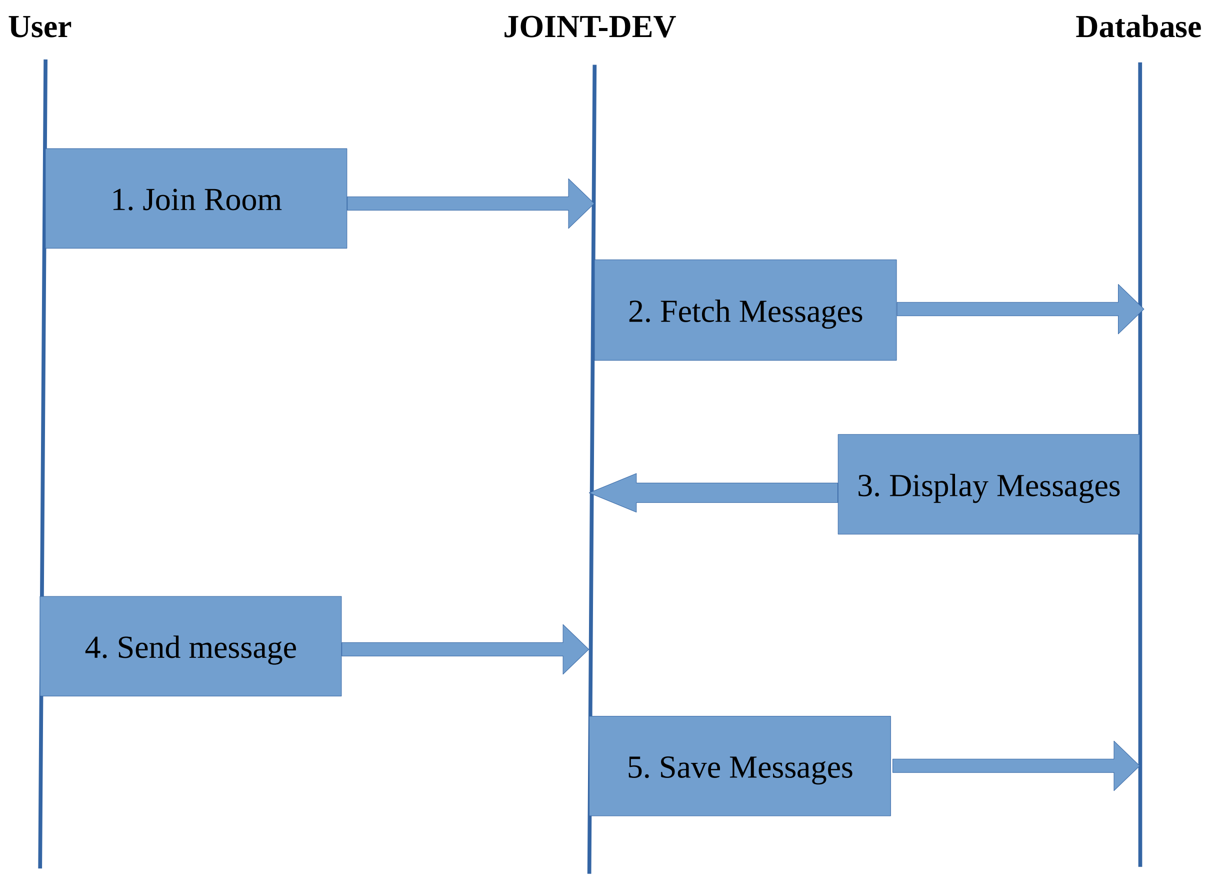
4.3.2 Use Case Diagrams:

Use case diagrams illustrate the interactions between actors and the system. Here, the main actors are developers and administrators.



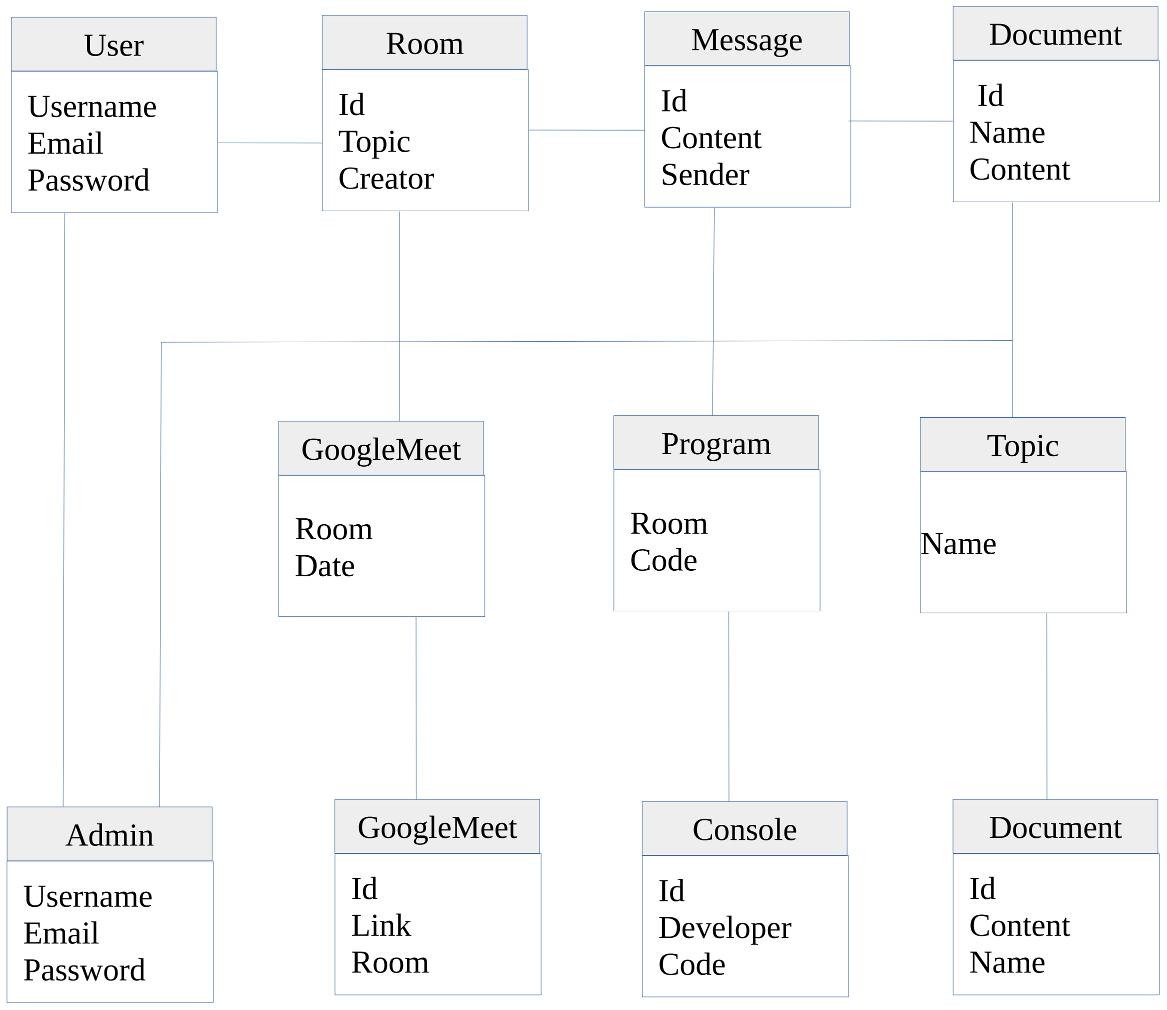
4.3.3 Sequence Diagrams:

Sequence diagrams show interactions between objects over time. Here's a simplified sequence diagram for a user joining a room and sending a message:



4.3.4 Class Diagrams:

Class diagrams depict the static structure of the system, including classes, attributes, methods, and their relationships.



In this diagram above:

- **User**: Represents the users who register and use the system.
- **Admin**: Represents the administrator who manages the system.
- **Room**: Represents the discussion rooms created by developers.
- **Message**: Represents the messages exchanged within a room.
- **Document**: Represents the documents stored within a room.
- **Google Meet**: Represents the Google Meet sessions associated with a room.
- **Console**: Represents the console window where developers can run their programs.
- **Program**: Represents the programs executed within the console.
- **Topic**: Represents the topics discussed within a room.

4.4 Architectural Design

Client-Server Architecture:

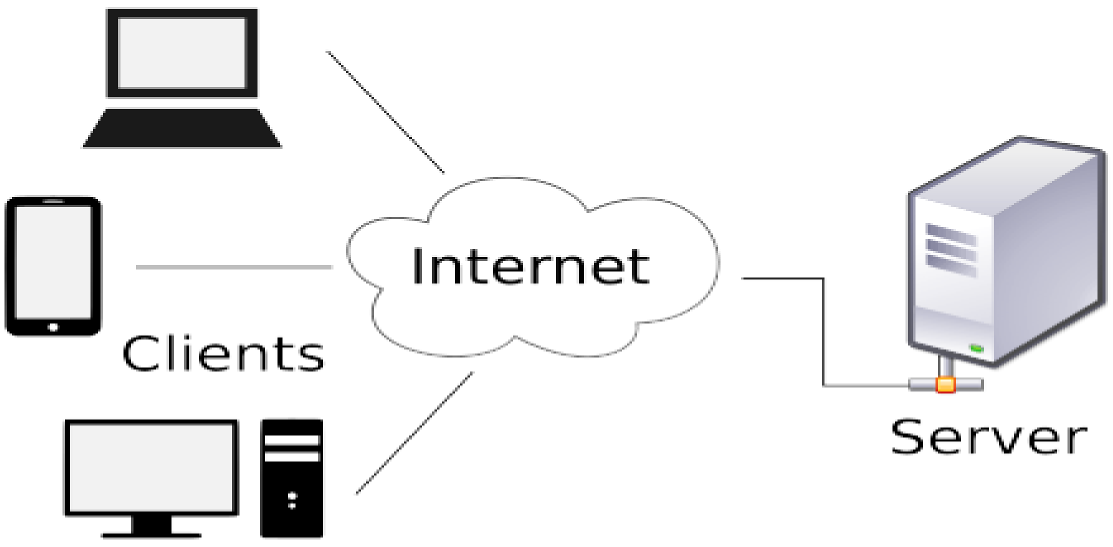


Figure 6: client server architecture

In this architecture, your system is divided into two main components:s

1. **Client Side:**
    - The client side would be the web application accessible to users (developers).

- It will handle the presentation logic and user interaction.
- Users will interact with the application through a web browser.
- This side will include functionalities like user registration, login, creating/joining rooms, sending messages, managing documents, running programs, and accessing profile information.
- Technologies like HTML, CSS, JavaScript (for front-end interactivity), and Django templates can be used to develop the client side.

2. **Server Side:**
    - The server side consists of the backend logic responsible for processing requests from clients and managing data.
    - It will handle authentication, authorization, business logic, data storage/retrieval, and real-time communication.
    - Django framework with Python can be used to build the server side.
    - Real-time communication can be facilitated through technologies like WebSockets, which allow for instant messaging and other real-time functionalities.
    - The server side will interact with the database (e.g., PostgreSQL) to store and retrieve user data, room information, messages, documents, etc.
    - Additionally, integration with third-party services like Google Meet and email services for reminders can be implemented on the server side.

4.5 User Interface Design

4.5.1 Home page

The image below shows the Home screen of JOINT-DEV. One can create his room and create a Topic.

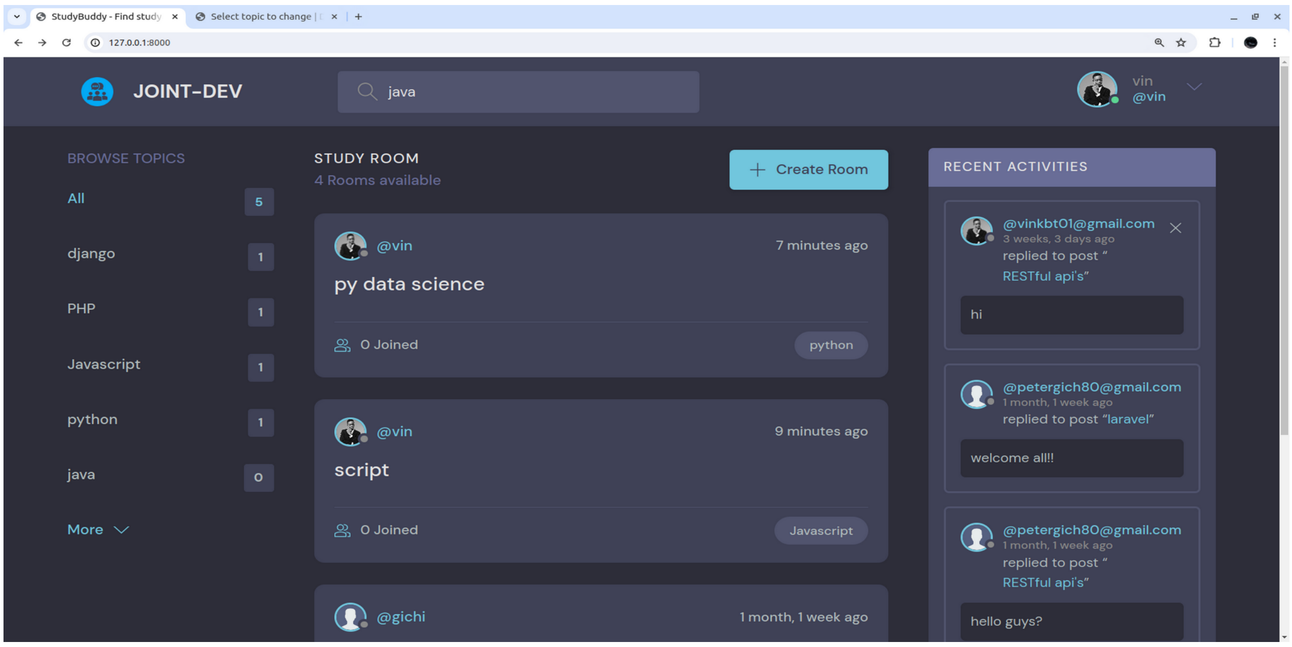


Figure 7: Home Page

4.5.2 Room Page

Once you create a room and the topic you can access the room by clicking on it on the home screen. Inside the room you can send and receive messages to and from other members.

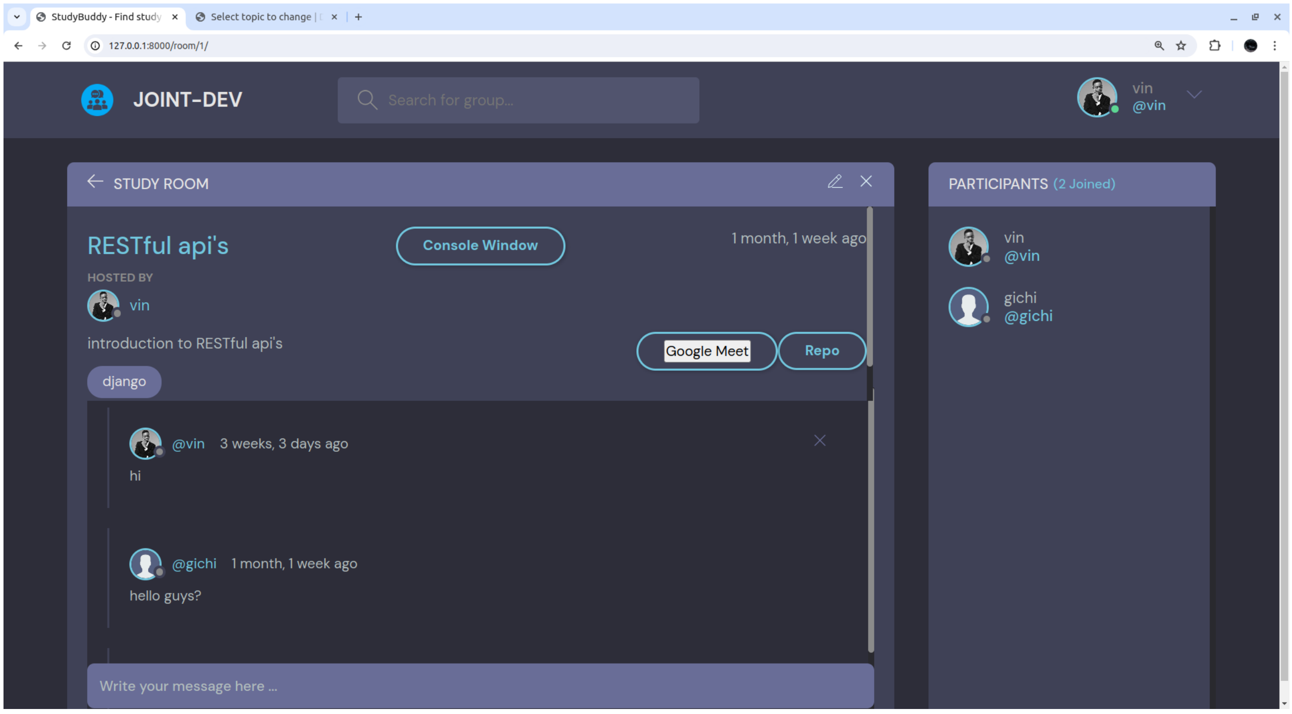
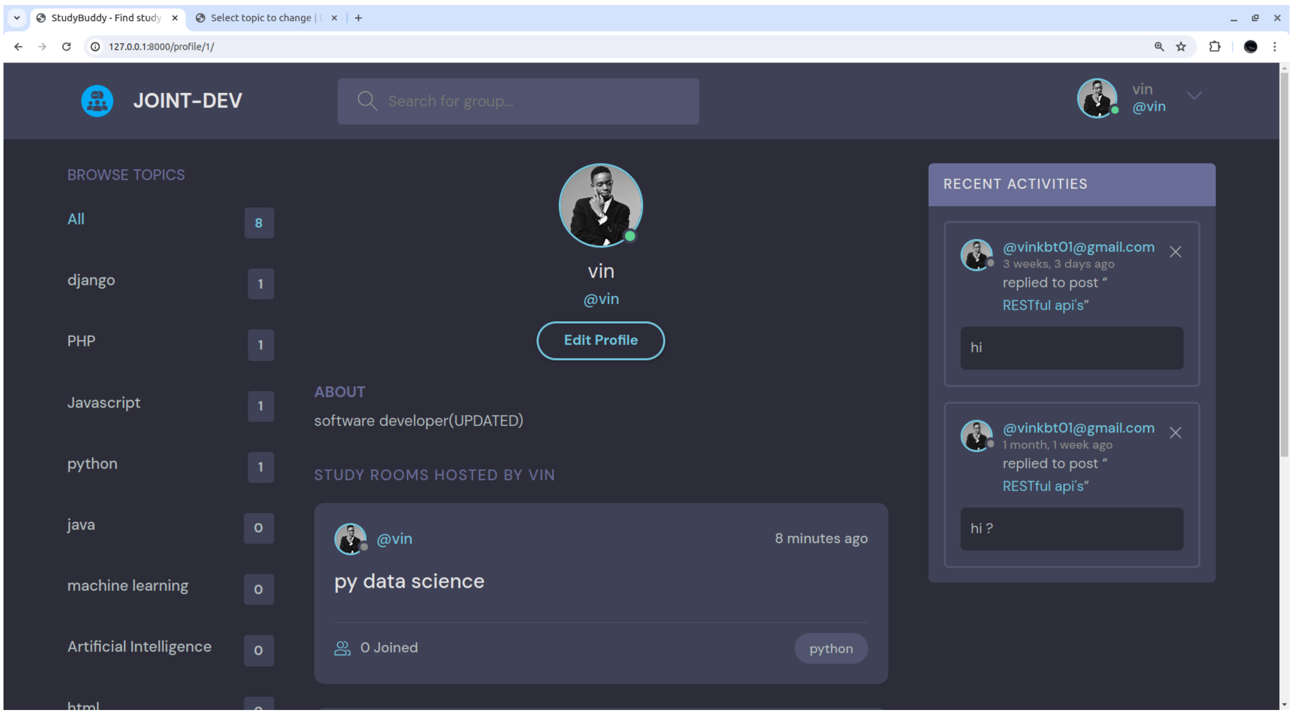


Figure 8: Room Page

4.5.3 About

About page shows the users bio and the groups he created and the message activities in each group



StudyBuddy - Find study | Select topic to change

127.0.0.1:8000/profile/1/

JOINT-DEV

Search for group...

vin @vin

BROWSE TOPICS

All	8
django	1
PHP	1
Javascript	1
python	1
java	0
machine learning	0
Artificial Intelligence	0

vin

@vin

Edit Profile

ABOUT

software developer(UPDATED)

STUDY ROOMS HOSTED BY VIN

@vin 8 minutes ago

py data science

0 Joined python

RECENT ACTIVITIES

@vinkbt01@gmail.com
3 weeks, 3 days ago
replied to post " RESTful api's"

hi

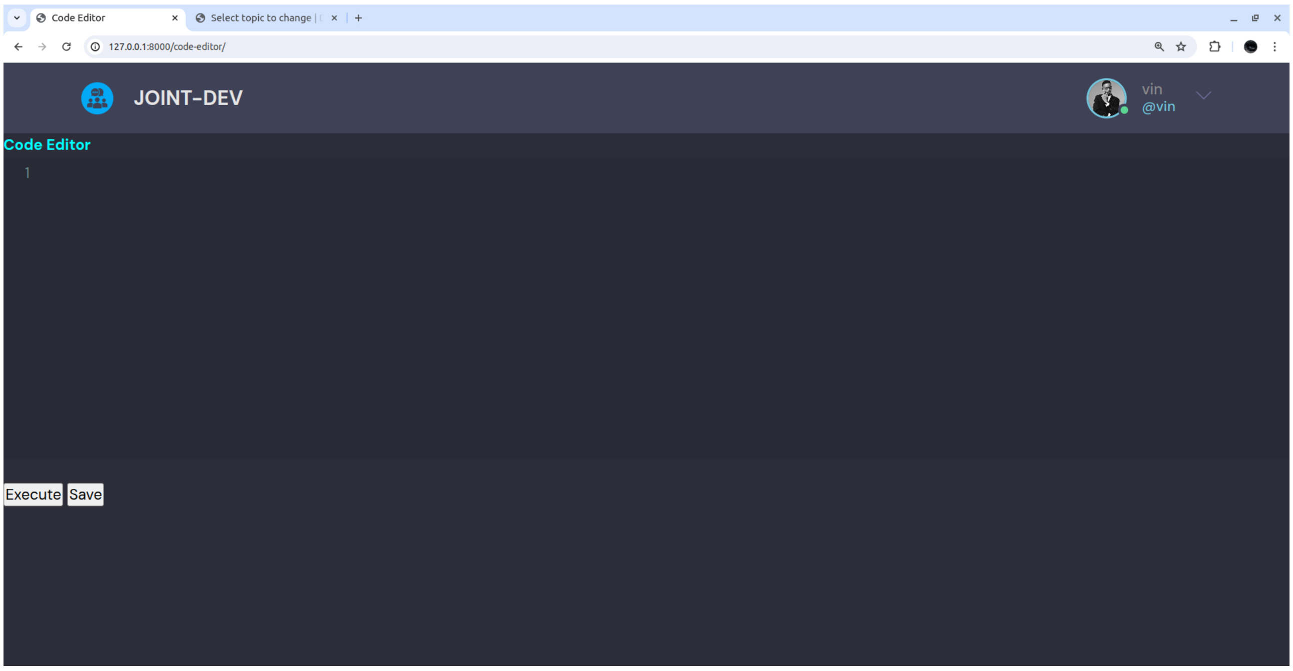
@vinkbt01@gmail.com
1 month, 1 week ago
replied to post " RESTful api's"

hi ?

Figure 9: About Page

4.5.4 Console window.

This is a code execution page inside a room where you can type your code and run it.



Code Editor | Select topic to change

127.0.0.1:8000/code-editor/

JOINT-DEV

vin @vin

Code Editor

Execute Save

4.5.5 Login/register page

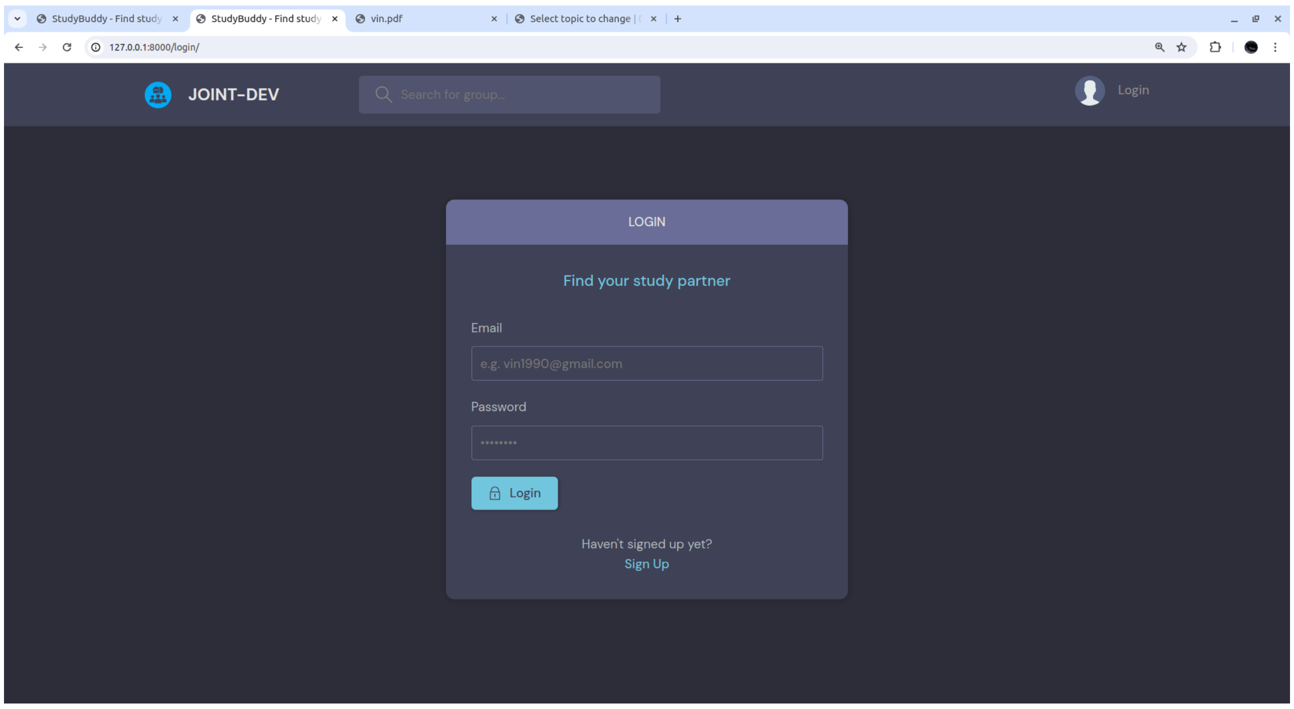
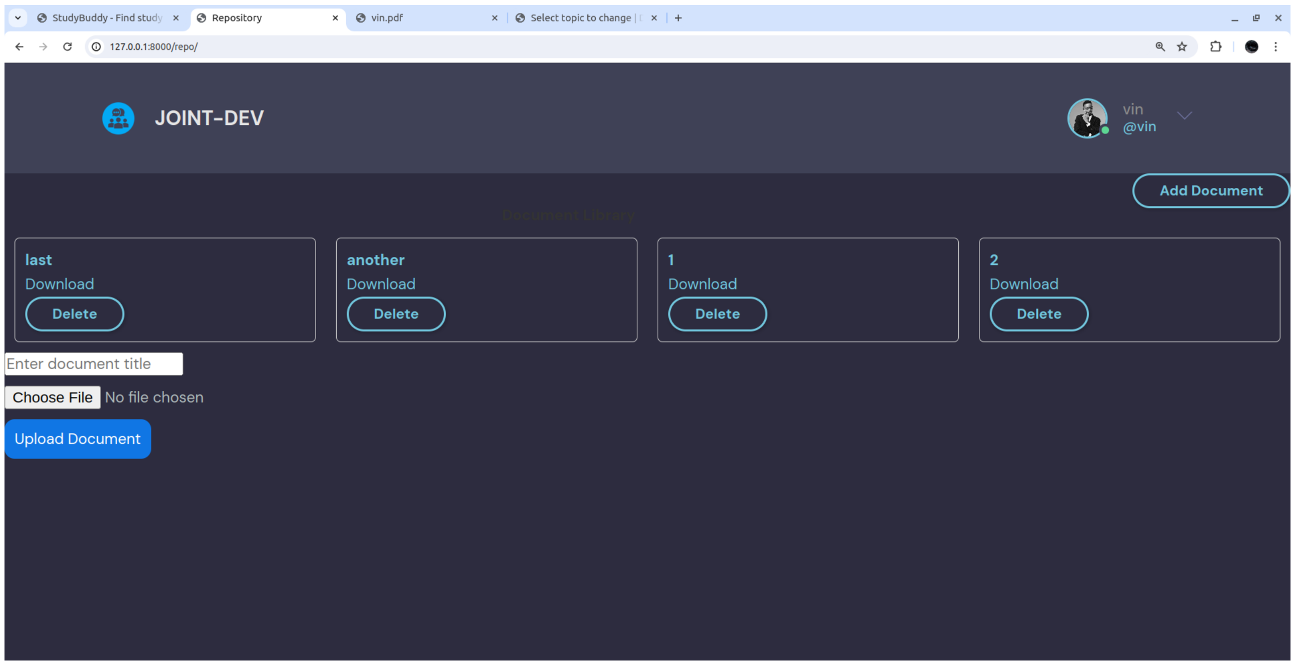


Figure 11: Login/Register page

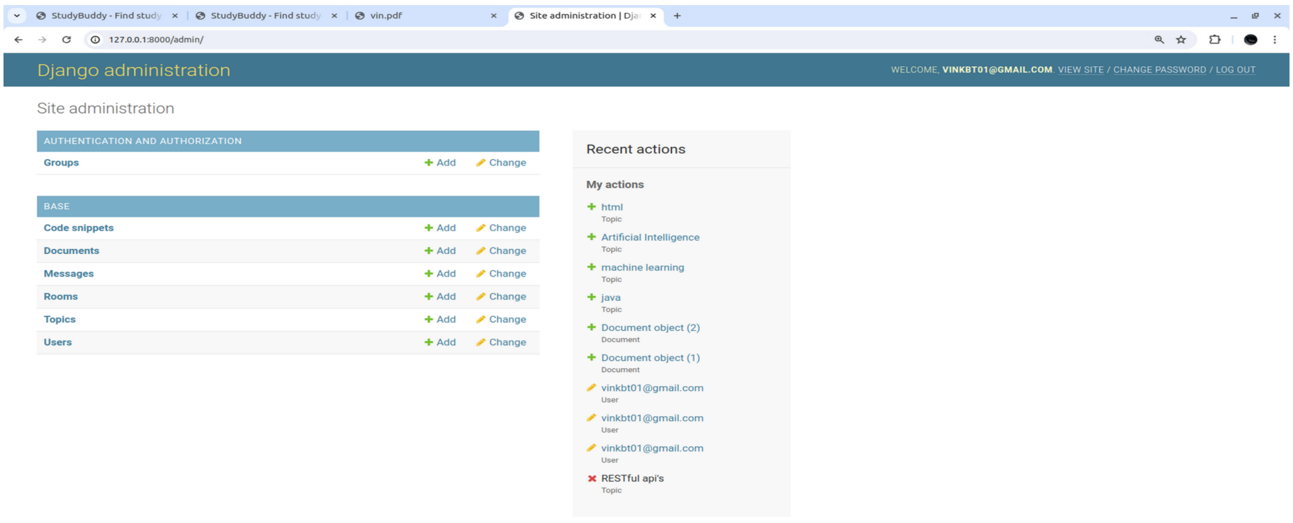
4.5.5 Document page.

This is where members access, add, delete documents. It displays all the documents4.6 Database



4.6.1 Base

this is a database page where it shows all the pages for our Joint-Dev application.


Django administration

WELCOME, VINKBT01@GMAIL.COM. VIEW SITE / CHANGE PASSWORD / LOG OUT

Site administration

AUTHENTICATION AND AUTHORIZATION		
Groups	Add	Change

BASE		
Code snippets	Add	Change
Documents	Add	Change
Messages	Add	Change
Rooms	Add	Change
Topics	Add	Change
Users	Add	Change

Recent actions

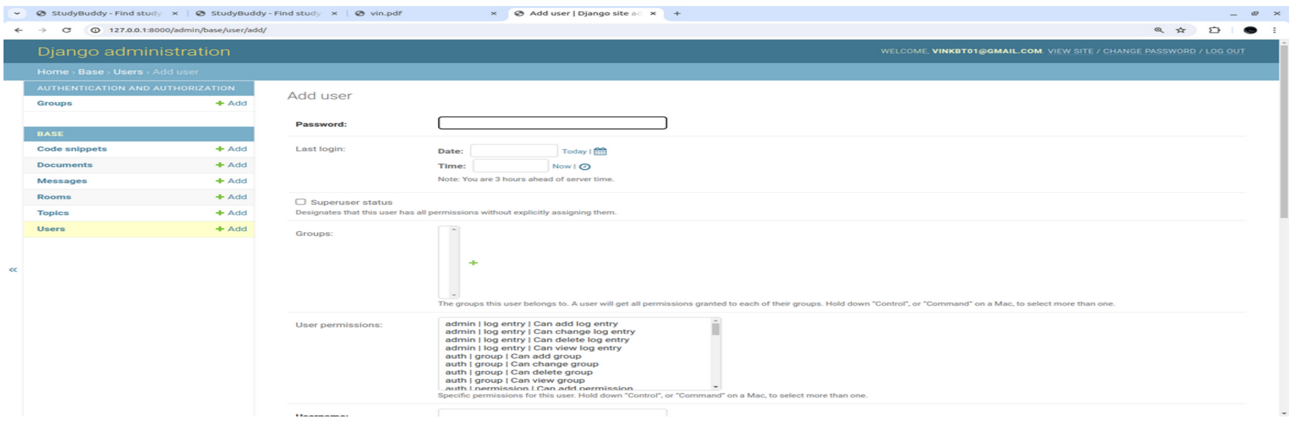
My actions

- html
  Topic
- Artificial Intelligence
  Topic
- machine learning
  Topic
- java
  Topic
- Document object (2)
  Document
- Document object (1)
  Document
- vinkbt01@gmail.com
  User
- vinkbt01@gmail.com
  User
- vinkbt01@gmail.com
  User
- RESTful api's
  Topic

Figure 13: database

4.6.2 User

this is where you can add users or delete users and also it displays all the users using our application. One can also enable user permission such as superuser


Django administration

WELCOME, VINKBT01@GMAIL.COM. VIEW SITE / CHANGE PASSWORD / LOG OUT

Home › Base › Users › Add user

Add user

Password:

Last login: Date: Today | Time: Now |
Note: You are 3 hours ahead of server time.

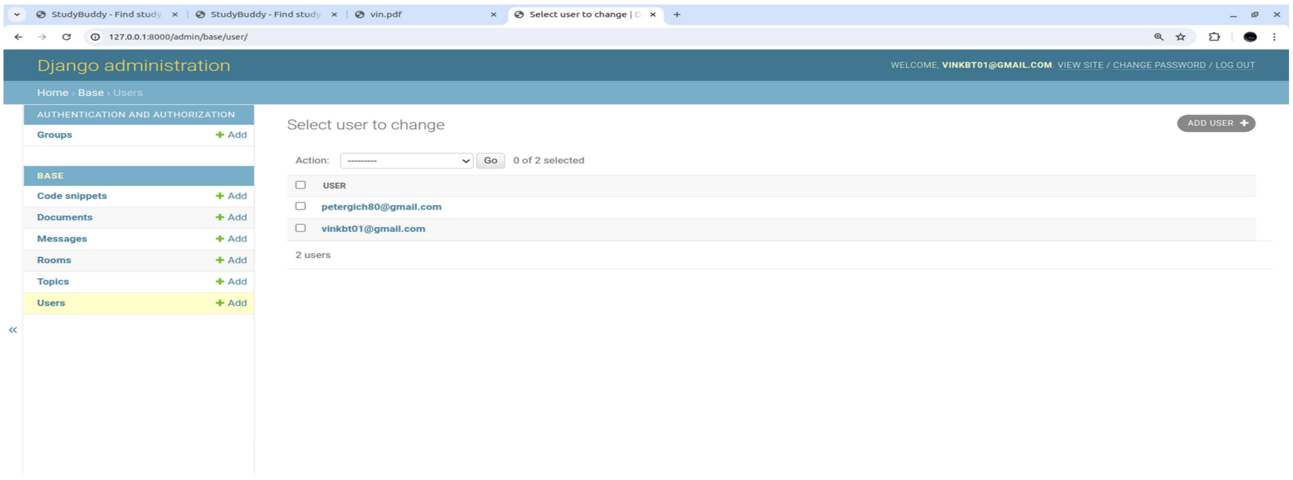
Superuser status
Designates that this user has all permissions without explicitly assigning them.

Groups:
The groups this user belongs to. A user will get all permissions granted to each of their groups. Hold down "Control", or "Command" on a Mac, to select more than one.

User permissions:

```
admin | log entry | Can add log entry
admin | log entry | Can change log entry
admin | log entry | Can delete log entry
admin | log entry | Can view log entry
auth | group | Can add group
auth | group | Can change group
auth | group | Can delete group
auth | group | Can view group
auth | permission | Can add permission
```

Specific permissions for this user. Hold down "Control", or "Command" on a Mac, to select more than one.


Django administration

WELCOME, VINKBT01@GMAIL.COM. VIEW SITE / CHANGE PASSWORD / LOG OUT

Home › Base › Users

Select user to change

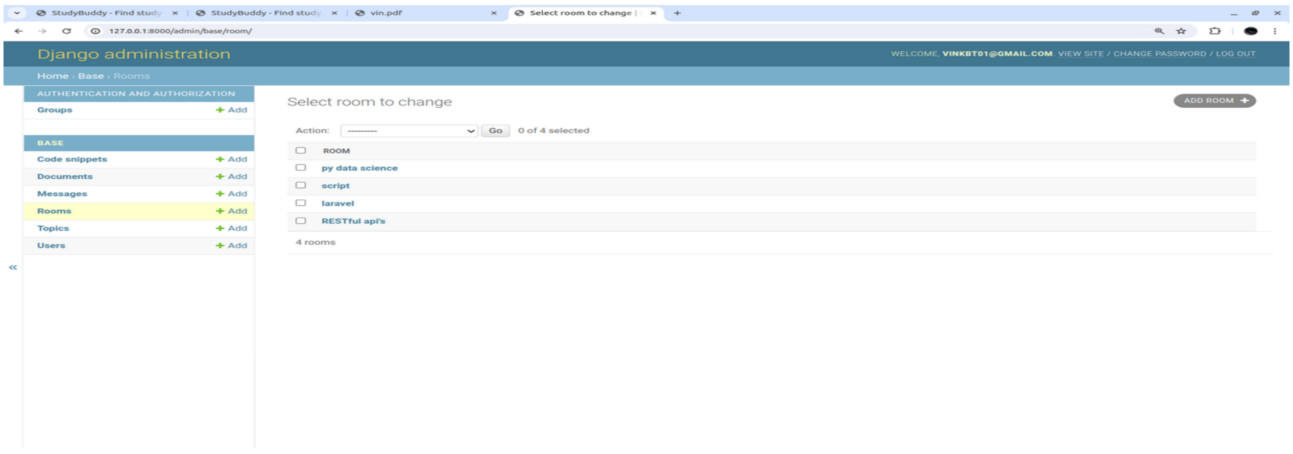
Action: --------- Go 0 of 2 selected

USER
petergich80@gmail.com
vinkbt01@gmail.com

2 users

4.6.3 Room

This a page where a room is stored. You can also create a room here with its properties e.g Topics, Name, Description.



Django administration

WELCOME, VINKBT01@GMAIL.COM. VIEW SITE / CHANGE PASSWORD / LOG OUT

Home › Base › Rooms

AUTHENTICATION AND AUTHORIZATION: Groups + Add

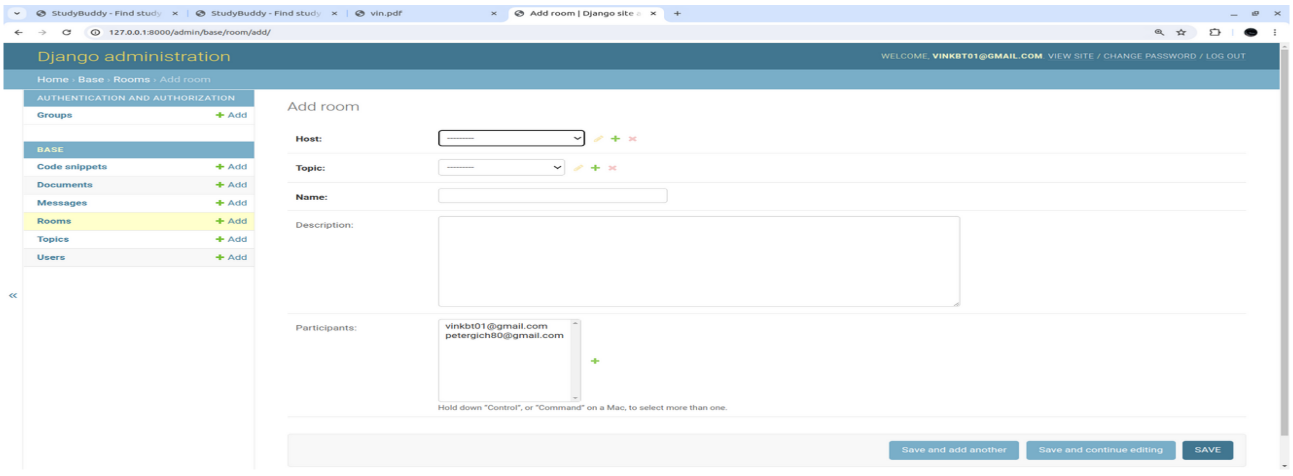
BASE: Code snippets + Add; Documents + Add; Messages + Add; Rooms + Add; Topics + Add; Users + Add

Select room to change — ADD ROOM +

Action: --------- Go 0 of 4 selected

ROOM
py data science
script
laravel
RESTful api's

4 rooms



Django administration

WELCOME, VINKBT01@GMAIL.COM. VIEW SITE / CHANGE PASSWORD / LOG OUT

Home › Base › Rooms › Add room

Add room

Host: ---------

Topic: ---------

Name:

Description:

Participants: vinkbt01@gmail.com, petergich80@gmail.com

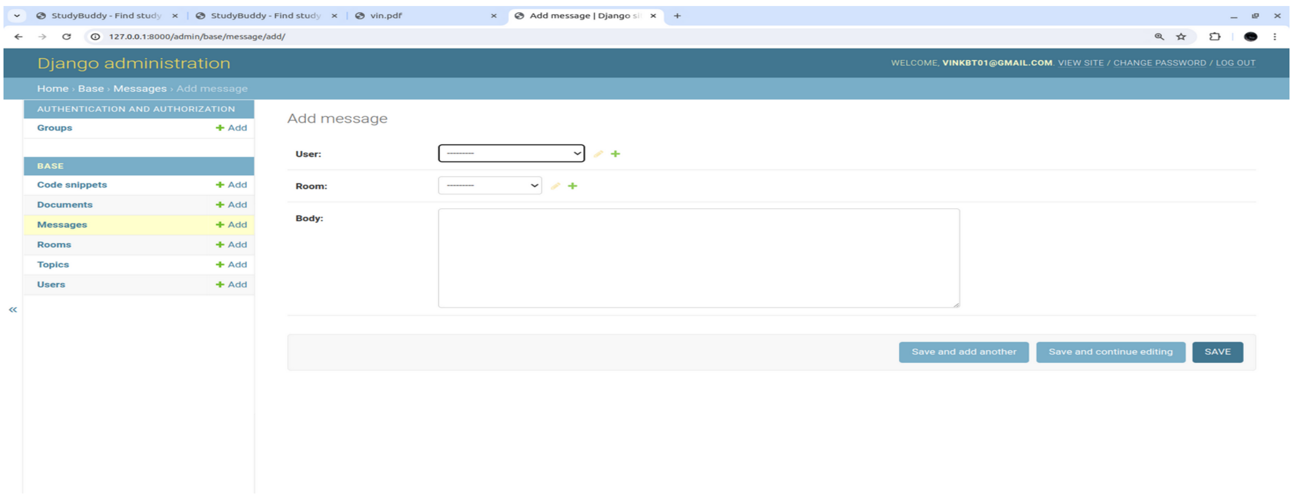
Hold down "Control", or "Command" on a Mac, to select more than one.

Save and add another | Save and continue editing | SAVE

Figure 15: Room Table

4.6.4 Message

This is where each message in each group is stored. It links with all users, topics and the rooms.



Django administration

WELCOME, VINKBT01@GMAIL.COM. VIEW SITE / CHANGE PASSWORD / LOG OUT

Home › Base › Messages › Add message

Add message

User: ---------

Room: ---------

Body:

Save and add another | Save and continue editing | SAVE

CHAPTER 5.

5. System Implementation and Testing

5.1 Implementation

5.1.1 Hardware Requirements

The section of hardware configuration is an important task related to the software development. Insufficient random access memory (RAM) may affect adversely on the speed and efficiency of the entire system. The hard disk should have sufficient capacity to store the file and application. For the case of this system the following minimum system requirement can be recommended.

Processor: Pentium IV and above

Processor speed: 1.4 GHz Onwards

RAM: 512 MB (Minimum)

Hard disk: 80 GB

5.1.2 Software Requirements

“A major element in building a system is the section of compatible software since the software in the market is experiencing in geometric progression. Selected software should be acceptable by the firm and one user as well as it should be feasible for the system”

Operating System

- ❖ Windows 95/98/NT or later versions
- ❖ Macintosh System 7 or later
- ❖ Most UNIX systems.

Web Browser

- ❖ Mozilla Firefox
- ❖ Safari on Macintosh
- ❖ Safari 5.1.7 for Windows
- ❖ Netscape 4.0 and earlier
- ❖ Internet Explorer 5.5 and later on windows

5.2 Testing

5.2.1 Test Cases

1. **User Authentication:**
    - Test user registration with valid and invalid inputs.
    - Test user login with correct and incorrect credentials.
    - Test user logout functionality.
2. **Room Creation and Management:**
    - Test room creation by registered users.
    - Test CRUD operations on rooms (create, read, update, delete).
    - Test room deletion by the creator.
3. **Real-time Messaging:**
    - Test real-time messaging functionality within a room.
    - Test message CRUD operations (create, read, update, delete).
    - Test message notification and updates.
4. **Google Meeting Integration:**
    - Test creating Google Meetings from within the room.
    - Test sending reminders for scheduled meetings via email.
5. **Code Execution:**
    - Test code execution using CodeMirror within the console window.
    - Test various programming languages supported for code execution.
6. **Document Management:**
    - Test adding, opening, and deleting documents within the repository.
    - Test document access control based on room permissions.
7. **Profile and Bio Editing:**
    - Test editing user profile information and bio.

- Test updating user profile picture.

5.2.2 Test Results

- All test cases pass successfully without any critical issues.

5.3 Deployment

5.3.1 Deployment Plan:

1. **Server Setup:**
    - Choose a cloud hosting provider (e.g., AWS, Azure, DigitalOcean).
    - Provision a server instance with appropriate specifications based on expected traffic.
    - Install necessary software dependencies including Python, Django, and web server.
2. **Database Setup:**
    - Configure the database server (e.g., PostgreSQL) and create the required database schema.
    - Migrate the Django application's database schema to the production environment.
3. **Application Deployment:**
    - Clone the application codebase from the version control system (e.g., Git).
    - Configure environment variables for sensitive information (e.g., database credentials, API keys).
    - Set up static file serving and media file storage configurations.
    - Configure the web server (e.g., Nginx) to serve the Django application.
4. **Testing in Production-like Environment:**
    - Perform thorough testing in the production-like environment to ensure stability and performance.
5. **Continuous Integration/Continuous Deployment (CI/CD):**
    - Set up CI/CD pipelines for automated testing and deployment.
    - Integrate with version control to automate deployment whenever changes are pushed to the repository.

6. **Monitoring and Scaling:**
    - Implement monitoring tools (e.g., Prometheus, Grafana) to monitor application performance and server health.
    - Set up auto-scaling policies to handle increased traffic and load.
7. **Security Measures:**
    - Implement SSL/TLS certificates for secure communication.
    - Regularly update system packages and dependencies to patch security vulnerabilities.
    - Implement secure coding practices to prevent common web vulnerabilities (e.g., SQL injection, XSS).
8. **Backup and Disaster Recovery:**
    - Set up regular backups of the database and application data.
    - Implement disaster recovery procedures to mitigate the impact of unexpected failures.
9. **User Training and Support:**
    - Provide documentation and training for users on how to use the application effectively.
    - Offer support channels for users to report issues and seek assistance.
10. **Launch and Monitoring:**
    - Coordinate the launch with stakeholders and announce the availability of the application.
    - Monitor the application closely post-launch for any issues or performance bottlenecks and address them promptly.

CHAPTER 6.

6. Discussion and Conclusion:

6.1 Discussion:

The JOINT-DEV project presents a comprehensive solution for facilitating group discussions among developers, integrating various features such as real-time messaging, Google Meet integration, code execution console, and document management. The application leverages the Django framework to provide a robust web platform for developers to collaborate effectively.

One of the notable features of JOINT-DEV is its emphasis on real-time communication within discussion rooms. This feature enhances collaboration by allowing participants to engage in instant discussions, share ideas, and provide feedback promptly. Additionally, the integration of Google Meet functionality enables face-to-face interactions, further fostering effective communication and teamwork.

The inclusion of a code execution console within discussion rooms offers a unique advantage to developers. This feature allows participants to experiment with code snippets, test solutions, and collaborate on coding challenges in real-time. By leveraging CodeMirror, an advanced code editor component, users can write and execute code seamlessly within the application environment, enhancing the development process.

Furthermore, JOINT-DEV provides a centralized platform for document management, enabling users to store, access, and collaborate on project-related documents efficiently. The repository functionality simplifies document organization and version control, ensuring that all members have access to the latest project resources.

6.2 Limitations:

Despite its comprehensive feature set, JOINT-DEV may have certain limitations that need to be addressed:

- **Scalability**: While the application is designed to accommodate group discussions, scalability issues may arise as the user base grows. Ensuring optimal performance and scalability will be crucial for handling increased traffic and user interactions.

- **Security**: As JOINT-DEV involves real-time communication and document sharing, ensuring robust security measures is imperative to protect user data and sensitive information. Implementing encryption protocols, access controls, and authentication mechanisms will be essential to mitigate security risks.
- **User Experience**: Enhancing the user experience is an ongoing process that requires continuous feedback and iteration. Addressing usability issues, improving navigation, and optimizing UI/UX design elements will contribute to a more intuitive and engaging user experience.
- **Platform Compatibility**: Compatibility with various devices and browsers is essential to ensure accessibility for all users. Conducting thorough compatibility testing and optimizing the application for different platforms will enhance usability and reach.

6.3 Recommendations:

To address the identified limitations and further enhance the JOINT-DEV platform, the following recommendations are proposed:

- **Performance Optimization**: Implement caching mechanisms, database optimizations, and load balancing techniques to improve application performance and scalability. Conduct regular performance testing to identify bottlenecks and optimize resource utilization.
- **Security Enhancements**: Continuously monitor security threats and vulnerabilities, and implement proactive measures such as security audits, penetration testing, and vulnerability assessments. Regularly update security protocols and enforce best practices to safeguard user data and privacy.
- **User Feedback Integration**: Establish channels for collecting user feedback and suggestions to identify areas for improvement. Conduct surveys, user interviews, and usability testing to gather insights into user preferences and pain points, and prioritize feature enhancements accordingly.
- **Cross-Platform Compatibility**: Ensure compatibility with a wide range of devices and browsers by adopting responsive design principles and leveraging compatibility testing tools. Conduct thorough testing across different platforms to identify and address any compatibility issues proactively.

6.4 Conclusion:

In conclusion, the JOINT-DEV project offers a comprehensive solution for facilitating group discussions and collaboration among developers. With its array of features including real-time messaging, Google Meet integration, code execution console, and document management, the platform empowers developers to communicate effectively, share knowledge, and collaborate on projects seamlessly. While certain limitations exist, such as scalability and security concerns, proactive measures and continuous improvement efforts can address these challenges and further enhance the platform's utility and usability. Overall, JOINT-DEV holds promise as a valuable tool for fostering collaboration and innovation within the developer community.

REFERENCES

Bentley, L., & Whitten, J. (2007). In *System Analysis & Design for the Global Enterprise* (p. 147).

Bentley, T. (2008). *System Analysis and Design for Global Enterprise*. Higher Education Press. Retrieved from http:/www.books.google.co.ke/books/about/systems_design_for_the_Global_E.html?id=Aga6qWAACAAj&redir_esc=y.

client server model. (2016, 4). Retrieved from technopedia: http://www.technopedia.com/definition/18321/client-server-model

dybka, p. (2015, december 9). *chen ERD notaion*. Retrieved from vertabelo: www.vertabela.com/blog/technical-articles/chen-erd-notaion

Eric, G., & H., R. (1994). *Elements of Re-usable Object Oriented Software*. Edison Wesley.

J, M. (2016, MARCH 22). *networking 3-TIER CLIENT/SERVER architecture*. Retrieved from CCM: http://cc.net/contents/151-networking-3-tier-client-server-architecture

Mifsud, J. (2013, May 20). *Usability Geek*. Retrieved from Requirements Gathering: A step By Step Approach Foa Better user experience: usabilitygeek.com/requirements-gathering-user-experience-pt1

pluta, j. (2004, october 24). *client server architecture*. Retrieved from MC|PRESS online: http://www.mcpressonline.com/networking/general/clientserver-architecture.html

Recal, j. (2015, september 22). *Computer Science*. Retrieved from Athabasca University: http://www.athabasca.ca/syllabi/comp/comp361.html

techopedia. (2016, jan). *system design*. Retrieved from techopedia: http:/www.techopedia.com/definition/29998/system-design

use case diagram in uml. (2015, sep 22). Retrieved from lucidchart: http://www.lucidchart.com/pages/uml/use-case-diagram

8. APPENDICES

A.1: Database Schema

- User table
- Room table
- Message table
- Topic table
- Document table

A.2: Django Application Structure

- `accounts`: Handles user authentication and profile management.
- `rooms`: Manages rooms, messages, topics, and documents.
- `console`: Provides functionality for running programs in a console.
- `meeting`: Handles Google Meet integration and reminder emails.
- `documents`: Manages documents associated with rooms.
- `templates`: HTML templates for rendering frontend views.
- `static`: CSS, JavaScript, and media files.

Interview Guide

Introduction

- Introduce yourself and the purpose of the interview.
- Explain that the interview aims to gather feedback on the JOINT-DEV platform.

Questions

1. How would you rate your overall experience with JOINT-DEV?
2. What features did you find most useful or enjoyable?
3. Were there any difficulties you encountered while using the platform?
4. How do you think JOINT-DEV could be improved or expanded?
5. Did you face any issues with user registration or authentication?
6. How intuitive was the interface for creating and managing rooms?

7. Did you find the real-time messaging feature convenient?
8. Were there any features missing that you expected to see?
9. How was your experience with the Google Meet integration?
10. Did you receive reminder emails for scheduled meetings?
11. How useful was the console feature for running programs?
12. Were you able to easily manage documents within rooms?
13. How would you rate the performance and responsiveness of the platform?
14. Any additional comments or suggestions for JOINT-DEV?

Questionnaire

Section 1: User Experience

(tick the correct answer).

1. How satisfied are you with the registration process?
   - Very satisfied
   - Satisfied
   - Neutral
   - Dissatisfied
   - Very dissatisfied
2. Rate the ease of navigating through the platform.
   - Very easy
   - Easy
   - Neutral
   - Difficult
   - Very difficult
3. How would you rate the responsiveness of the platform?
   - Very responsive

- Responsive
- Neutral
- Unresponsive
- Very unresponsive

Section 2: Features

4. Which features did you find most useful?
    - Real-time messaging
    - Google Meet integration
    - Console for running programs
    - Document management
    - Other (please specify)
5. Were there any features missing that you expected to see?
    - Yes
    - No
    - Not sure
6. How satisfied are you with the overall feature set?
    - Very satisfied
    - Satisfied
    - Neutral
    - Dissatisfied
    - Very dissatisfied

Section 3: Improvements

7. How do you think JOINT-DEV could be improved?
    - More intuitive interface
    - Additional features
    - Better performance

- Improved documentation
- Other (please specify)

User Manual

Registration and Login

1. Navigate to the JOINT-DEV website.
2. Click on the "Register" button and fill in the required information.
3. Once registered, log in using your credentials.

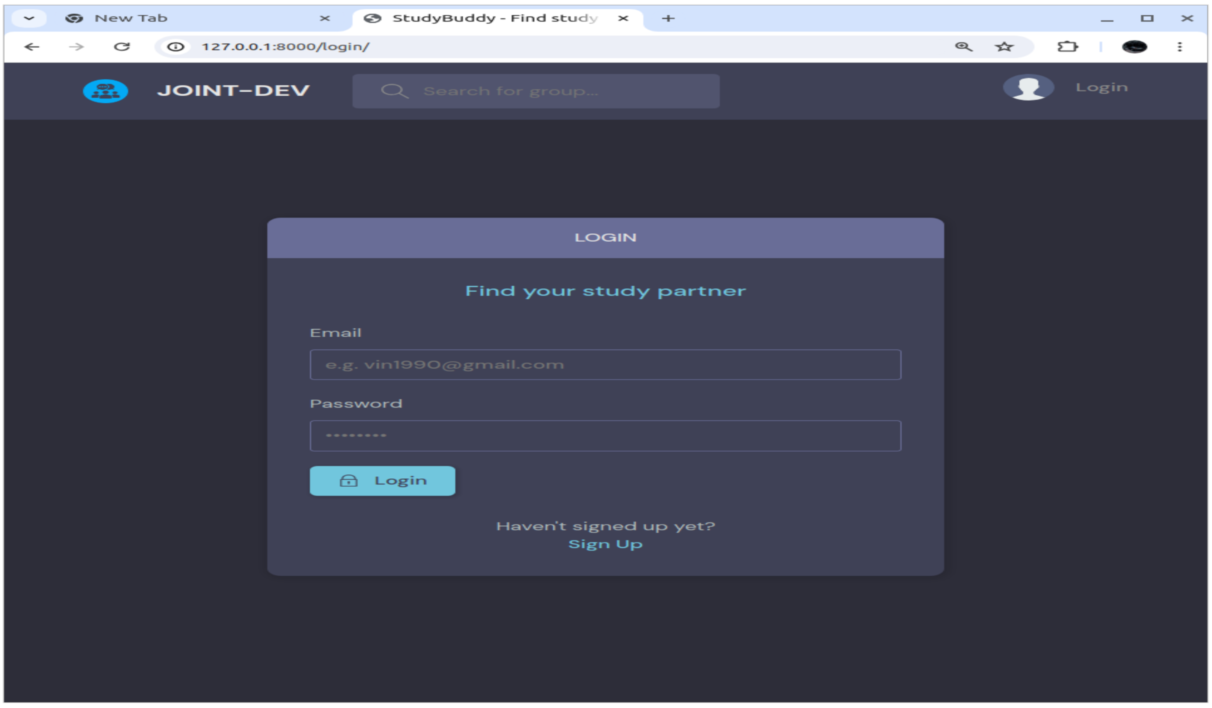


Figure 17: Login/Registration

Creating a Room

1. After logging in, click on the "Create Room" button.
2. Enter the room details including the topic and description.
3. Click "Create Room" to finalize the room creation.

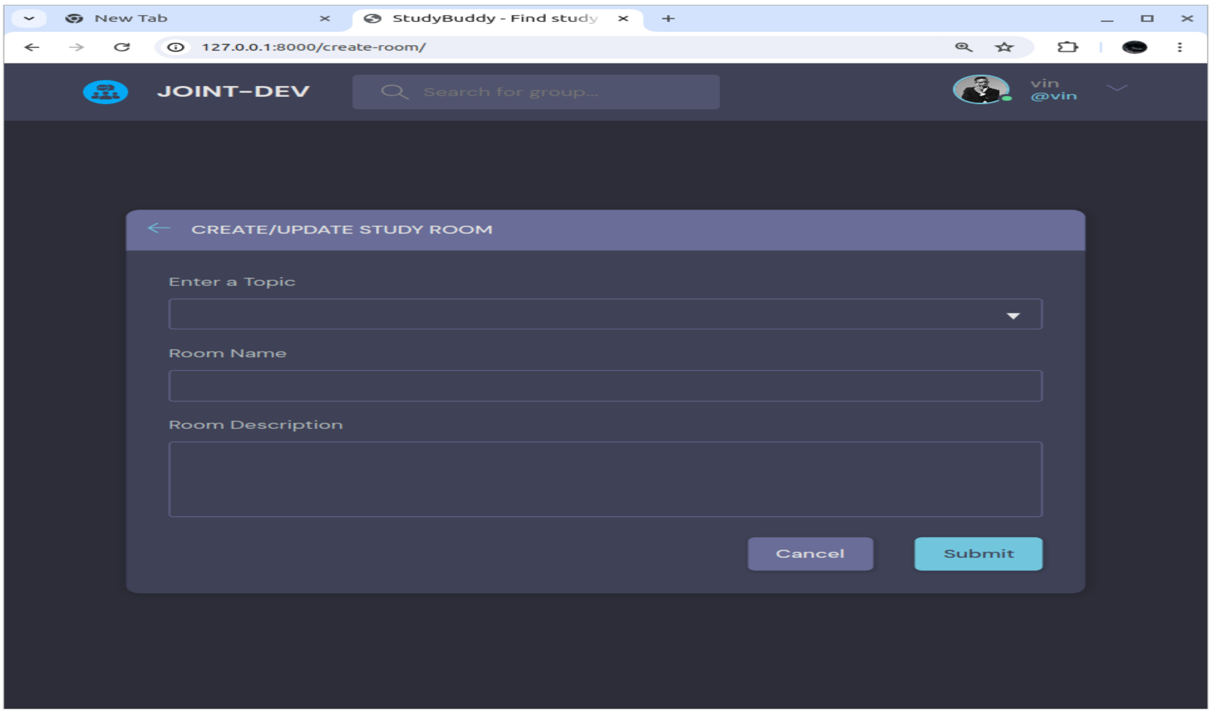


Figure 18: Room creation

Real-time Messaging

1. Enter a room by selecting it from the list.
2. Use the chat interface to send and receive messages in real-time.
3. You can also delete your own messages if necessary.

Google Meet Integration

1. Inside a room, click on the "Create Meeting" button.
2. Follow the prompts to set up a Google Meet session.
3. Reminders for the meeting will be sent via email.

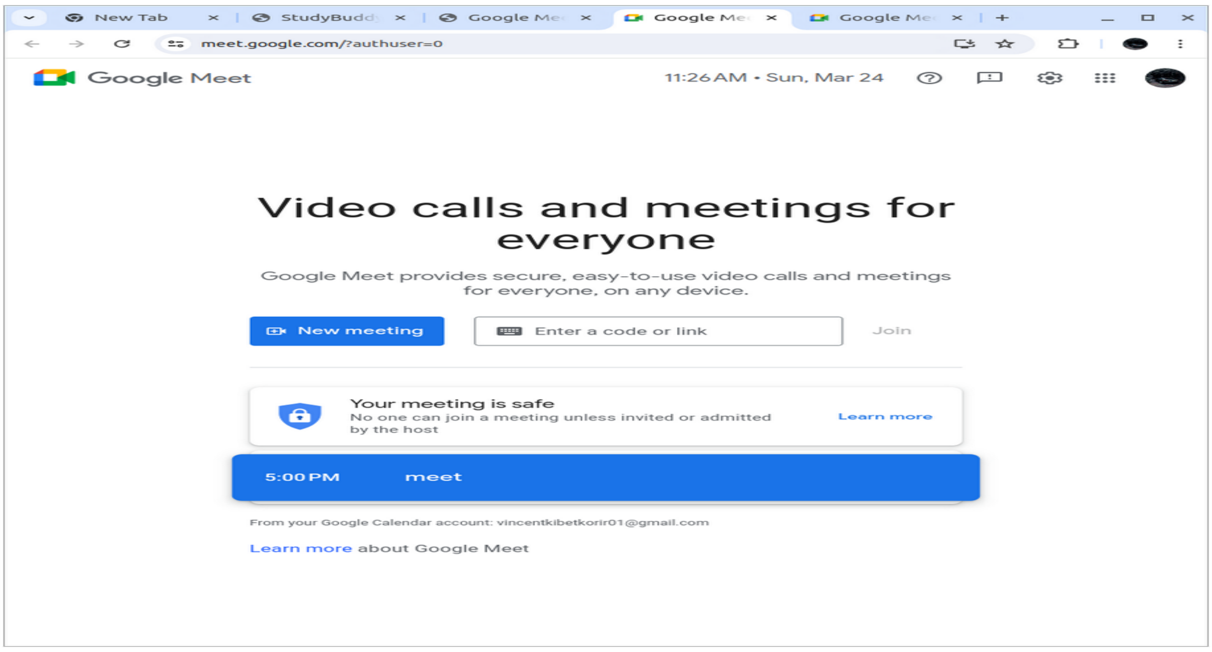

New Tab
StudyBudd
Google Me
meet.google.com/?authuser=0
Google Meet
11:26 AM • Sun, Mar 24
Video calls and meetings for everyone
Google Meet provides secure, easy-to-use video calls and meetings for everyone, on any device.
New meeting
Enter a code or link
Join
Your meeting is safe
No one can join a meeting unless invited or admitted by the host
Learn more
5:00 PM
meet
From your Google Calendar account: vincentkibetkorir01@gmail.com
Learn more about Google Meet

Figure 19: Google meet UI

Console for Running Programs

1. Inside a room, locate the console feature.
2. Click on the "Console" button to open the console window.
3. Write your program code using CodeMirror and execute it.

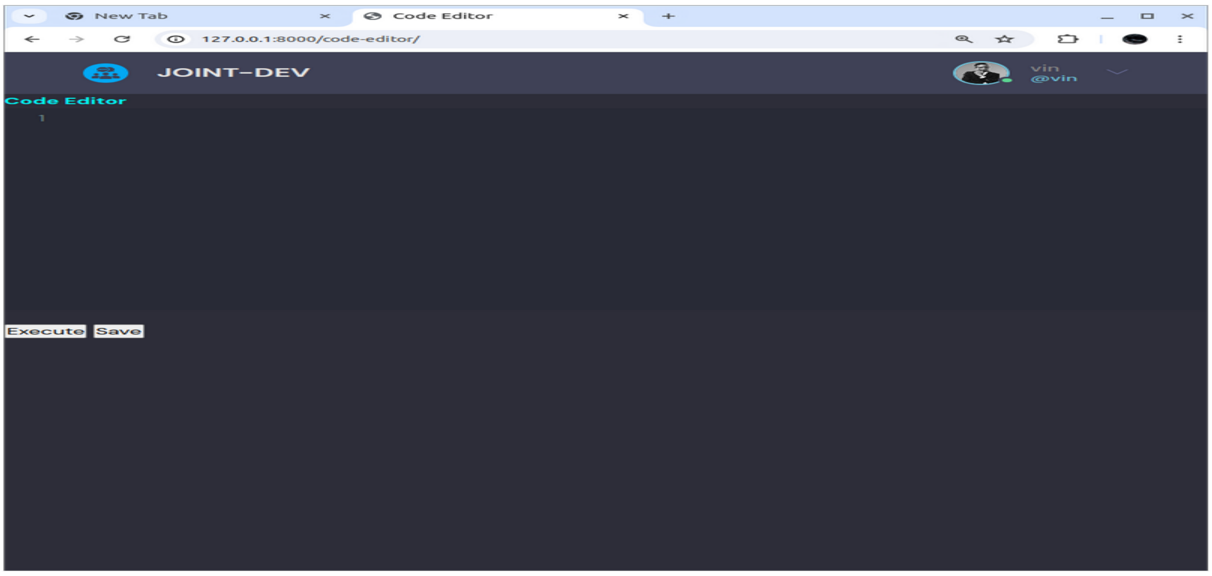


Figure 20: Console Window

Document Management

1. In the room page, find the "Repo" button.

2. Clicking on it will direct you to a page where documents are stored.

3. You can add, open, or delete documents as needed.

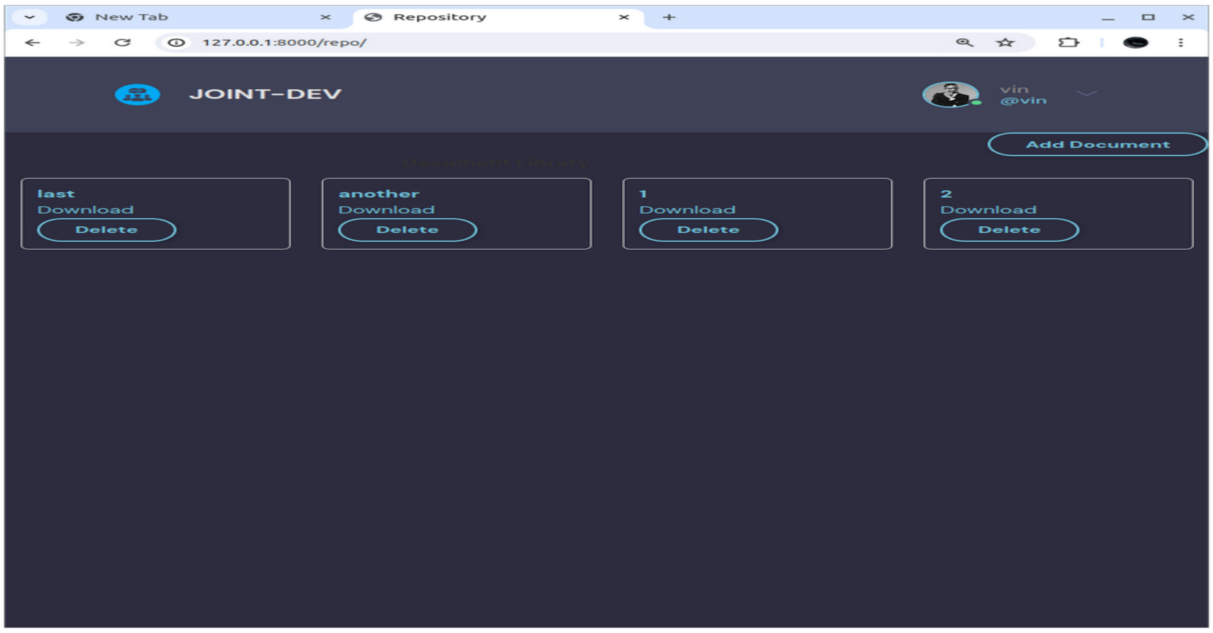
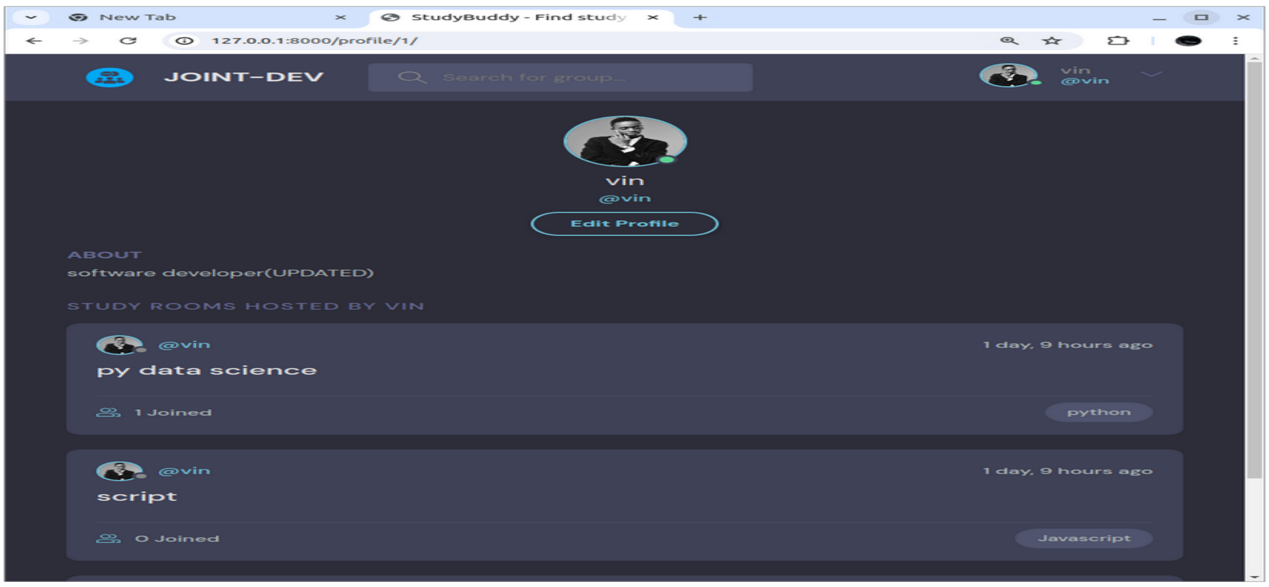


Figure 21: Document mngmnt Page

Profile and Bio

1. Click on profile to open about page.

2. Edit your profile information and bio as desired.



ABOUT
software developer(UPDATED)

STUDY ROOMS HOSTED BY VIN

@vin 1 day, 9 hours ago
py data science
1 Joined python

@vin 1 day, 9 hours ago
script
0 Joined Javascript

Figure 22: Abou

Crucial code parts.

Views.py

```
from django.shortcuts import render, redirect
from django.http import HttpResponse
from django.contrib import messages
from django.contrib.auth.decorators import login_required
from django.db.models import Q
from django.contrib.auth import authenticate, login, logout
from .models import Room, Topic, Message, User, Document
from .forms import RoomForm, UserForm, MyUserCreationForm, CodeSnippetForm, DocumentForm
from django.http import JsonResponse
from google.oauth2 import id_token
from google.auth.transport import requests as google_requests
import traceback

def loginPage(request):
page = 'login'
if request.user.is_authenticated:
return redirect('home')

if request.method == 'POST':
email = request.POST.get('email').lower()
password = request.POST.get('password')

try:
user = User.objects.get(email=email)
except:
messages.error(request, 'user does not exist')

user = authenticate(request, email=email, password=password)

if user is not None:
login(request, user)
return redirect('home')
else:
messages.error(request, 'username or password does not exist')

context = {'page':page}
return render(request, 'base/login_register.html', context)

def logoutUser(request):
logout(request)
return redirect('home')

def registerPage(request):
form = MyUserCreationForm()

if request.method == 'POST':
form = MyUserCreationForm(request.POST)
if form.is_valid():
user = form.save(commit=False)
user.username = user.username.lower()
user.save()
```

```
login(request, user)
return redirect('home')
else:
messages.error(request,'Error occured during registration')

return render(request, 'base/login_register.html', {'form': form})

def home(request):
q = request.GET.get('q') if request.GET.get('q') != None else ''
rooms = Room.objects.filter(
Q(topic__name__icontains=q) |
Q(name__icontains=q) |
Q(description__icontains=q)
)

topics = Topic.objects.all()[0:5]

room_count = rooms.count()

room_messages = Message.objects.filter(Q(room__topic__name__icontains=q))

context = {
'rooms': rooms,
'topics': topics,
'room_count': room_count,
'room_messages': room_messages
}
return render(request, 'base/home.html', context)

def room(request, id):

room = Room.objects.get(id=id)
room_messages = room.message_set.all()
participants = room.participants.all()

if request.method == 'POST':
message = Message.objects.create(
user=request.user,
room = room,
body = request.POST.get('body')
)
room.participants.add(request.user)
return redirect('room', id=room.id)

context = {
'room' : room ,
'room_messages' : room_messages,
'participants' : participants
}
return render(request, 'base/room.html', context)

def code_editor(request):
form = CodeSnippetForm(request.POST or None)
output = None
if request.method == 'POST':
```

```
if form.is_valid():
code = form.cleaned_data['code']
try:
# Redirect stdout to capture output
import sys
from io import StringIO
original_stdout = sys.stdout
sys.stdout = StringIO()
exec(code)
output = sys.stdout.getvalue()
# Restore stdout
sys.stdout = original_stdout
except Exception as e:
output = f"Error: {e}<br>{traceback.format_exc()}"
return JsonResponse({'output': output})
return render(request, 'base/code-editor.html', {'form': form})


def save_code(request):
if request.method == 'POST':
form = CodeSnippetForm(request.POST)
if form.is_valid():
code_snippet = form.save() # Save the code snippet to the database
return JsonResponse({'success': True, 'id': code_snippet.id})
else:
return JsonResponse({'success': False, 'errors': form.errors})
else:
return JsonResponse({'success': False, 'errors': 'Invalid request method'})

def google_meet_auth(request):
# Generate authorization URL
auth_url = "https://accounts.google.com/o/oauth2/auth?"
auth_url += "response_type=code"
auth_url += "&client_id=484745872524-0bspvhjro0fon5o907qpoeirhuuusccb.apps.googleusercontent.com"
auth_url += "&redirect_uri=https://www.joint-dev.com"
auth_url += "&scope=https://www.googleapis.com/auth/meet.readonly"

return redirect(auth_url)

def google_meet_callback(request):
# Get authorization code from request
code = request.GET.get('code')

# Exchange authorization code for access token
token_url = "https://oauth2.googleapis.com/token"
token_payload = {
"code": code,
"client_id": "484745872524-0bspvhjro0fon5o907qpoeirhuuusccb.apps.googleusercontent.com",
"client_secret": "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",
"redirect_uri": "https://www.joint-dev.com",
"grant_type": "authorization_code"
}
response = google_requests.post(token_url, data=token_payload)
token_data = response.json()
access_token = token_data.get("access_token")
```

```
return HttpResponse("Successfully authenticated with Google Meet.")

def list_upcoming_meetings(access_token):
url = "https://www.googleapis.com/calendar/v3/users/me/calendarList"
headers = {
"Authorization": f"Bearer {access_token}"
}
response = requests.get(url, headers=headers)
return response.json()

def google_meet_view(request):
meet_url = "https://meet.google.com/?authuser=0"
return render(request, 'base/meet.html', {'meet_url':meet_url})

def joinMeeting(request):
return render(request, 'base/video_conference.html')

def userProfile(request, id):
user = User.objects.get(id=id)
rooms = user.room_set.all()
room_messages = user.message_set.all()
topics = Topic.objects.all()
context = {'user': user,
'rooms': rooms,
'room_messages': room_messages,
'topics': topics
}
return render(request, 'base/profile.html', context)

@login_required(login_url='login')
def createRoom(request):
form = RoomForm()
topics = Topic.objects.all()
if request.method == 'POST':
topic_name = request.POST.get('topic')
topic, created = Topic.objects.get_or_create(name=topic_name)

Room.objects.create(
host=request.user,
topic=topic,
name=request.POST.get('name'),
description=request.POST.get('description'),
)
return redirect('home')

context = {'form': form, 'topics': topics, 'room': room}
return render(request, 'base/room_form.html', context)

@login_required(login_url='login')
def updateRoom(request, id):
room = Room.objects.get(id=id)
form = RoomForm(instance=room)
topics = Topic.objects.all()
```

```
if request.user != room.host:
return HttpResponse(' Forbidden!!')

if request.method == 'POST':
topic_name = request.POST.get('topic')
topic, created = Topic.objects.get_or_create(name=topic_name)
room.name = request.POST.get('name')
room.topic = topic
room.description = request.POST.get('description')
room.save()
return redirect('home')

context = {'form': form, 'topics': topics}
return render(request, 'base/room_form.html', context)

@login_required(login_url='login')
def deleteRoom(request, id):
room = Room.objects.get(id=id)

if request.user != room.host:
return HttpResponse(' Forbidden!!')

if request.method == 'POST':
room.delete()
return redirect ('home')
return render (request, 'base/delete.html', {'obj':room})

@login_required(login_url='login')
def deleteMessage(request, id):
message = Message.objects.get(id=id)

if request.user != message.user:
return HttpResponse(' Forbidden!!')

if request.method == 'POST':
message.delete()
return redirect ('home')
return render (request, 'base/delete.html', {'obj':message})

@login_required(login_url='login')
def updateUser(request):
user = request.user
form = UserForm(instance=user)

if request.method == 'POST':
form = UserForm(request.POST, request.FILES, instance=user)
if form. is_valid():
form.save()
return redirect('user-profile', id=user.id)

return render(request, 'base/update-user.html', {'form':form})


def topicsPage(request):
q = request.GET.get('q') if request.GET.get('q') != None else ''
```

```
topics = Topic.objects.filter(name__icontains=q)
return render(request, 'base/topics.html', {'topics':topics})

def activityPage(request):
room_messages = Message.objects.all()
return render(request, 'base/activity.html', {'room_messages':room_messages})


def repo(request):
documents = Document.objects.all()
# notes = Note.objects.all()
return render(request, 'base/repo.html', {'documents': documents})

def add_document(request):
if request.method == 'POST':
form = DocumentForm(request.POST, request.FILES)
if form.is_valid():
form.save()
return redirect('repo')
else:
form = DocumentForm()
return render(request, 'base/add_document.html', {'form': form})

def delete_document(request, document_id):
document = Document.objects.get(id=document_id)
document.delete()
return redirect('repo')
```

urls.py

```
from django.urls import path
from . import views

urlpatterns = [
path('login/', views.loginPage, name='login'),
path('logout/', views.logoutUser, name='logout'),
path('register/', views.registerPage, name='register'),

path('', views.home, name='home'),
path('room/<str:id>/', views.room, name='room'),
path('profile/<str:id>/', views.userProfile, name='user-profile'),
path('code-editor/', views.code_editor, name='code-editor'),
path('save/', views.save_code, name='save_code'),
path('join-meet/', views.joinMeeting, name='join-meet'),
path('google_meet/', views.google_meet_view, name='google_meet_view'),
path('auth/google_meet/', views.google_meet_auth, name='google_meet_auth'),
path('auth/google_meet/callback/', views.google_meet_callback, name='google_meet_callback'),

path('create-room/', views.createRoom, name='create-room'),
path('update-room/<str:id>/', views.updateRoom, name='update-room'),
path('delete-room/<str:id>/', views.deleteRoom, name='delete-room'),
path('delete-message/<str:id>/', views.deleteMessage, name='delete-message'),
path('update-user/', views.updateUser, name='update-user'),

path('topics/', views.topicsPage, name='topics'),
path('activity/', views.activityPage, name='activity'),

path('repo/', views.repo, name='repo'),
path('add_document/', views.add_document, name='add_document'),
path('delete_document/<int:document_id>/', views.delete_document, name='delete_document'),
]
```

Models.py

```
from django.db import models
from django.contrib.auth.models import AbstractUser

class User(AbstractUser):
name = models.CharField(max_length=100, null=True)
email = models.EmailField(unique=True, null=True)
bio = models.TextField(null=True)

avatar = models.ImageField(null=True, default="avatar.svg")

USERNAME_FIELD = 'email'
REQUIRED_FIELDS = []

# Create your models here.

class Topic(models.Model):
name = models.CharField(max_length=200)

def __str__(self):
return self.name

class Room(models.Model):
host = models.ForeignKey(User, on_delete=models.SET_NULL, null=True)
topic = models.ForeignKey(Topic, on_delete=models.SET_NULL, null=True)
name = models.CharField(max_length=200)
description = models.TextField(null=True, blank=True)
participants = models.ManyToManyField(User, related_name='participants', blank=True)
updated = models.DateTimeField(auto_now=True)
created = models.DateTimeField(auto_now_add=True)

class Meta:
ordering = [
'-updated', '-created'
]

def __str__(self):
return self.name
```

```
class CodeSnippet(models.Model):
code = models.TextField()


class Document(models.Model):
title = models.CharField(max_length=100)
file = models.FileField(upload_to='repo/')
created = models.DateTimeField(auto_now_add=True)


class Message(models.Model):
user = models.ForeignKey(User, on_delete=models.CASCADE )
room = models.ForeignKey(Room, on_delete=models.CASCADE)
body = models.TextField()
updated = models.DateTimeField(auto_now=True)
created = models.DateTimeField(auto_now_add=True)

class Meta:
ordering = [
'-updated', '-created']

def __str__(self):
return self.body[0:50]
```