**A PROJECT ON Pariwar WEB APP**

SUBMITTED BY

Success Dai

STUDENT ID: LC00016000

FINAL YEAR PROJECT REPORT SUBMITTED IN

PARTIAL FULFILLMENT FOR THE DEGREE OF

BACHELOR OF INFORMATION TECHNOLOGY

FACULTY OF COMPUTER SCIENCE IN MULTIMEDIA

LINCOLN UNIVERSITY COLLEGE

MALAYSIA

JULY 202

# Declaration

We hereby declare that this report is based on our own independent work, except for Quotation and summaries which have been dully acknowledged. I also declare that no part of this work has been submitted for any degree to this or any other university.

|  |  |
| --- | --- |
| Date: 12 June 2024 | Signature: |
|  | Name: |

Supervisor:

Faculty of Computer Science in Multimedia,

Nepal Business College,

Lincoln University

Supervisor:

# Acknowledgement

# Abstract

# Table of Contents

Catalog

[Declaration 3](#_Toc26865)

[Acknowledgement 4](#_Toc550)

[Abstract 5](#_Toc9386)

[Table of Contents 6](#_Toc20713)

[Chapter one Introduction 9](#_Toc13884)

[1.1 Background of the Study 9](#_Toc6082)

[1.2 Problem Statement 9](#_Toc4000)

[1.3 Project Objectives 10](#_Toc8161)

[1.4 Significance of the Project 10](#_Toc24572)

[1.5 Limitations of the Project 11](#_Toc961)

[1.6 Organisation of the Project 11](#_Toc2081)

[Chapter Two Literature Review 13](#_Toc6186)

[CHAPTER THREE THORETICAL BACKGROUND OF THE PROJECT 15](#_Toc3078)

[1 Web Development Frameworks: 15](#_Toc4668)

[2 Database Management Systems: 15](#_Toc12590)

[3 User Experience (UX) Design: 16](#_Toc31433)

[In summary 16](#_Toc14612)

[CHAPTER FOUR PROPOSED METHODOLOGY 17](#_Toc16779)

[4.1 Requirement Analysis 17](#_Toc24592)

[4.2 System Design 18](#_Toc5877)

[4.3 Technology Stack Selection 18](#_Toc19541)

[4.4 Development Phase 19](#_Toc10679)

[4.5 Testing and Quality Assurance 19](#_Toc13818)

[4.6 Deployment 19](#_Toc3369)

[4.7 Maintenance and Future Enhancements 20](#_Toc6185)

[Chapter Five Result and Discussion 21](#_Toc30770)

[5.1 System Analysis 21](#_Toc31978)

[5.2 Use Case Diagram 24](#_Toc18551)

[5.3 Context Diagram 25](#_Toc16421)

[Fig 5.4: ContextDiagram 25](#_Toc21446)

[5.4 ER Diagram 26](#_Toc19408)

[Fig 5.4: ER Diagram 26](#_Toc8735)

[5.5 Activity Diagram 27](#_Toc7774)

[Chapter Six Api Documentaion 28](#_Toc15564)

[1 Accounts API 28](#_Toc13810)

[1. Login API 28](#_Toc25440)

[ Endpoint: /api/login/ 28](#_Toc3556)

[ Method: POST 28](#_Toc15774)

[2. Register API 28](#_Toc2447)

[ Endpoint: /api/register/ 28](#_Toc6542)

[ Method: POST 28](#_Toc16012)

[2 Profile API 29](#_Toc32591)

[1. Get Profile API 29](#_Toc25026)

[3 Issue Segment 29](#_Toc28773)

[1. Post Issue API 29](#_Toc30195)

[ Endpoint: /api/post\_issue/ 30](#_Toc5785)

[ Method: POST 30](#_Toc4010)

[2. Get All Issues API 30](#_Toc1693)

[ Endpoint: /api/get\_all\_issues/ 30](#_Toc25023)

[ Method: POST 30](#_Toc7446)

[3. Get Issues by Character API 31](#_Toc7880)

[ Endpoint: /api/get\_issue\_character/ 31](#_Toc28862)

[ Method: POST 31](#_Toc24360)

[4. Get Issues by User API 31](#_Toc27315)

[ Endpoint: /api/get\_issue\_by\_user/ 31](#_Toc30888)

[ Method: POST 31](#_Toc24118)

[5. Get Specific Issue API 32](#_Toc23480)

[ Endpoint: /api/get\_specific\_issue/ 32](#_Toc867)

[ Method: POST 32](#_Toc27588)

[4 Reply / Suggestion Segment 32](#_Toc5289)

[1. Post Reply API 33](#_Toc13134)

[ Endpoint: /api/post\_reply/ 33](#_Toc3344)

[ Method: POST 33](#_Toc6565)

[2. Get Issue Replies API 33](#_Toc13704)

[ Endpoint: /api/get\_issue\_reply/ 33](#_Toc12137)

[ Method: POST 33](#_Toc25047)

[3. Get All Replies API 34](#_Toc4171)

[ Endpoint: /api/get\_reply\_list/ 34](#_Toc485)

[ Method: POST 34](#_Toc2979)

[5 Relation Segment 35](#_Toc4724)

[1. Update Relation API 35](#_Toc4815)

[ Endpoint: /api/relation\_update/ 35](#_Toc4230)

[ Method: POST 35](#_Toc9084)

[2. Get User Relations API 35](#_Toc27899)

[ Endpoint: /api/get\_particular\_relation/ 35](#_Toc21831)

[ Method: POST 36](#_Toc10808)

[6 Chat API 36](#_Toc11609)

[Get Chat by Relation ID 36](#_Toc23498)

[Post Chat 37](#_Toc12266)

[Chapter Seven Conclusion and Recommendation 39](#_Toc16485)

[7 Conclusion 39](#_Toc9933)

# List of tables

# List of figures

# List of abbreviations

# Chapter one Introduction

## 1.1 Background of the Study

In today's fast-paced world, students often face challenges and pressures that can be difficult to navigate. While family support is crucial, many students find it challenging to communicate their issues directly with their parents due to generational gaps, fear of misunderstanding, or cultural norms that discourage open discussion of certain topics. This communication gap can lead to a sense of isolation, anxiety, and a lack of necessary guidance.

Recognizing the importance of family support, the Pariwar web application was conceived as a solution to bridge this gap. Pariwar provides a platform where students can seek advice and support from other family members or individuals playing familial roles, allowing for more open and targeted communication. By facilitating these connections, the application aims to create a supportive environment where students can receive the guidance they need to navigate their challenges effectively.

## 1.2 Problem Statement

Students often struggle to share their personal issues with their parents, either due to fear of judgment, generational differences, or cultural constraints. This communication barrier can prevent them from receiving the support and guidance they need during critical times. Without an effective way to communicate their problems, students may feel isolated, leading to increased stress and potential long-term negative consequences.

The Pariwar web application addresses this issue by providing a platform where students can anonymously or publicly share their concerns with other family members or individuals in familial roles. By enabling targeted communication and fostering a supportive environment, the application seeks to bridge the gap between students and the support network they need, ensuring that they are not left to face their challenges alone.

## 1.3 Project Objectives

The objectives of the Pariwar web application are:

* **Facilitate Communication:** Provide a platform for students to share concerns comfortably, especially when direct communication with parents is difficult.
* **Encourage Family Support:** Engage extended family members in offering guidance and support, leveraging diverse perspectives within the family.
* **Enable Anonymous Posting:** Allow students to post concerns anonymously, reducing fear of judgment.
* **Offer Targeted Advice:** Enable users to seek advice from specific familial roles for more relevant support.
* **Build Relationships:** Foster stronger family connections through ongoing communication.
* **Create a Supportive Space:** Cultivate a safe environment for open discussion and feedback.

## 1.4 Significance of the Project

* Bridging Communication Gaps: Provides an alternative way for students to seek family support when direct communication is challenging.
* Promoting Well-being: Helps reduce student stress and isolation by facilitating access to guidance and support.
* Strengthening Family Bonds: Encourages stronger family connections through involvement in the support process.
* Empowering Students: Gives students the tools to seek help comfortably, enabling them to tackle challenges effectively.
* Culturally Sensitive: Offers a solution that respects cultural barriers to open communication within families.

## 1.5 Limitations of the Project

**Lack of Fake User Detection:** The system currently lacks robust mechanisms to detect and prevent fake users from spreading misinformation or engaging in harmful chats.

**Potential Misuse:** Users may exploit the platform for unintended purposes, such as creating false identities or misrepresenting their roles.

**Limited Moderation:** The platform may face challenges in effectively moderating content and ensuring that all interactions remain supportive and constructive.

**Dependence on User Integrity:** The success of the application relies heavily on users' honesty and genuine participation, which cannot always be guaranteed.

## 1.6 Organisation of the Project

The report is organized as follows:

* **Chapter 1:** Introduction, which includes the abstract, background of the study, problem statement, project objectives, significance, limitations, and organization of the report.
* **Chapter 2:** Literature Review, providing an overview of existing solutions and related research in the field of room rental applications.
* **Chapter 3:** Theoretical background, provides the theory of techniques used in the project
* **Chapter 4:** Methodology, detailing the design and development process of the Room Finder Web App, including technology stack, system architecture, and implementation steps.
* **Chapter 5:** Conclusion and Discussion, presenting the outcomes of the project, testing results, and user feedback.
* **Chapter 6:** Conclusion and Future Work, summarizing the project’s achievements, limitations, and potential areas for future enhancement.
* **References:** A list of all sources cited in the report.
* **Appendices:** Additional material supporting the report, such as code snippets, screenshots, and user manuals.

# Chapter Two Literature Review

The concept of family-based support systems has been extensively explored in both psychological and sociological research, emphasizing the critical role that family dynamics play in an individual’s emotional and mental well-being. Studies indicate that open communication within families contributes significantly to the development of healthy coping mechanisms in young adults, particularly in navigating the stresses associated with academic and social pressures. However, research also highlights the challenges posed by generational gaps, cultural norms, and varying communication styles, which can create barriers to effective dialogue between parents and their children. This gap is particularly pronounced in societies where discussing personal problems, especially those related to mental health, remains stigmatized.

In response to these challenges, technology-mediated solutions have gained traction, providing platforms for individuals to seek advice and support outside of traditional face-to-face interactions. These platforms often utilize online communities, where anonymity and targeted communication can reduce the fear of judgment and encourage more open discussion. However, the effectiveness of these platforms hinges on their ability to foster trust and maintain a supportive environment, free from the negative influences of misinformation and toxic interactions.

The Pariwar web application draws on these insights, aiming to bridge the communication gap within families by leveraging technology to create a safe and supportive space for students. By allowing users to take on familial roles, the platform seeks to replicate the protective and nurturing environment traditionally provided by family structures, while also addressing the communication barriers identified in existing literature. Despite the potential benefits, the literature also warns of the risks associated with such platforms, particularly in terms of user authenticity and the potential for misuse. These concerns underscore the need for ongoing research and development to enhance the security and effectiveness of family-oriented support applications like Pariwar.

# CHAPTER THREE THORETICAL BACKGROUND OF THE PROJECT

The Pariwar web application is grounded in several theoretical frameworks relevant to web development, database management, and user experience design. Understanding these frameworks helps in appreciating how the application’s technology stack supports its objectives and enhances its functionality.

## Web Development Frameworks:

Web development frameworks play a crucial role in building scalable and efficient web applications. React is a JavaScript library for building user interfaces, particularly for single-page applications. React’s component-based architecture enables the development of interactive and dynamic user experiences by managing and updating user interfaces efficiently. Django, a high-level Python web framework, provides a robust backend infrastructure with features such as authentication, routing, and database management. Django’s emphasis on reusability and rapid development supports the creation of secure and maintainable web applications. The combination of React and Django facilitates a clear separation between frontend and backend functionalities, allowing for a modular and scalable application architecture.

## Database Management Systems:

Effective database management is essential for handling and storing data in web applications. SQLite is a lightweight, serverless database engine that is ideal for applications with moderate data needs. Its simplicity and ease of integration make it a suitable choice for managing user data, such as user profiles, issue postings, and feedback in the Pariwar application. SQLite’s ability to handle transactions and queries efficiently supports the application’s data management requirements, ensuring that user interactions and content are stored and retrieved reliably.

## User Experience (UX) Design:

User experience design focuses on creating intuitive and engaging interactions for users. Tailwind CSS is a utility-first CSS framework that enables the rapid development of visually appealing and responsive user interfaces. Its approach to styling emphasizes the use of predefined classes to achieve consistent and flexible design, enhancing the overall user experience. By applying Tailwind CSS, the Pariwar application ensures a user-friendly interface that is both aesthetically pleasing and functional. UX design principles are crucial in creating an accessible and enjoyable platform where users can easily navigate, post concerns, and provide feedback.

In summary, the theoretical background of the Pariwar web application incorporates principles from web development frameworks, database management systems, and user experience design. The use of React and Django frameworks supports a modular and efficient application structure, while SQLite manages data effectively. Tailwind CSS enhances the user interface, ensuring a positive user experience. Together, these technologies and theories form the foundation for a well-designed and functional application that meets the needs of its users.

# CHAPTER FOUR PROPOSED METHODOLOGY

## 4.1 Requirement Analysis

The Pariwar web application is designed to address the communication barriers between students and their family members by providing a platform for seeking advice and support. The primary functional requirements include:

* **User Role Management:** Users should be able to sign up and identify themselves with roles such as father, mother, brother, etc., to offer targeted advice.
* **Issue Posting:** Students must be able to post their concerns either anonymously or publicly, seeking suggestions from family members.
* **Feedback Mechanism:** Family members should be able to provide feedback and suggestions on posted issues.
* **Private Messaging:** Users should be able to establish private communication channels based on received feedback.

Non-functional requirements include:

* **Security:** The platform must ensure secure handling of user data and privacy, especially when dealing with anonymous postings.
* **Scalability:** The system should be able to handle a growing number of users and interactions.
* **Usability:** The user interface must be intuitive and accessible to a diverse user base, including those with limited technical skills.

## 4.2 System Design

The system architecture of the Pariwar web application consists of the following components:

* **Frontend:** Built using **React** for a dynamic and interactive user interface. React’s component-based architecture allows for efficient rendering and updating of user interfaces. **Tailwind CSS** is used for styling, ensuring a responsive and visually appealing design.
* **Backend:** Powered by **Django**, which provides a robust framework for handling authentication, routing, and database interactions. **SQLite** is used as the database management system, offering a lightweight solution for storing user data and application content.

The architecture follows a client-server model where the React frontend communicates with the Django backend via RESTful APIs. This separation of concerns ensures modularity and scalability.

## 4.3 Technology Stack Selection

The selection of the technology stack was guided by the following criteria:

* **React** was chosen for its efficient rendering capabilities and component-based structure, which are ideal for building dynamic user interfaces.
* **Django** was selected for its comprehensive backend support, including built-in features for user management and security.
* **REST API** was employed to facilitate smooth communication between the frontend and backend components.
* **Tailwind CSS** was used for its utility-first approach to styling, which simplifies the creation of responsive and attractive designs.
* **SQLite** was chosen for its simplicity and ease of integration, making it suitable for managing the application’s data requirements.

## 4.4 Development Phase

The development process follows an **Agile methodology**, which allows for iterative development and continuous feedback. Key aspects include:

* **Task Management:** Development tasks are divided among team members based on their roles, with regular sprints and reviews.
* **Code Collaboration:** Tools such as Git for version control and collaborative platforms like GitHub or GitLab are used to manage code and facilitate teamwork.

## 4.5 Testing and Quality Assurance

Testing and quality assurance involve:

* **Unit Testing:** Testing individual components to ensure they function correctly.
* **Integration Testing:** Verifying that different components work together as expected.
* **User Testing:** Gathering feedback from real users to identify usability issues and areas for improvement.
* **Tools:** Testing frameworks such as Jest for frontend testing and Django’s built-in testing tools for backend validation.

## 4.6 Deployment

The deployment strategy includes:

* **Hosting:** The application is deployed on a cloud platform such as AWS or Heroku to ensure scalability and reliability.
* **Continuous Deployment:** Automated deployment processes are set up to streamline updates and manage application versions.

## 4.7 Maintenance and Future Enhancements

Ongoing maintenance involves:

* **Monitoring:** Regular monitoring of system performance and user feedback to address issues promptly.
* **Updates:** Implementing updates based on user feedback and emerging technological advancements.
* **Future Enhancements:** Potential future features include advanced user analytics, additional support roles, and integration with other communication tools.

This methodology provides a structured approach to developing, deploying, and maintaining the Pariwar web application, ensuring that it meets the needs of its users while allowing for future growth and improvement.

# Chapter Five Result and Discussion

## 5.1 System Analysis

The system analysis of the Pariwar web application includes a detailed examination of both functional and non-functional requirements, ensuring that the application meets its intended goals effectively.

### 1.1 Functional Requirements

**User Authentication and Management:**

* 1. **Sign-Up Process:** Users must be able to register by providing their username, email, age, selecting a familial role (father, mother, son, etc.), and setting a password. They will receive an OTP for account verification.
  2. **Login:** Users can log in using their credentials and access their personalized dashboard.
  3. **Profile Management:** Users should be able to update their profile information, including their role and personal details.

**Issue Posting and Management:**

* 1. **Post Issues:** Users can submit issues by providing a title, description, and selecting the preferred role for suggestions.
  2. **View Issues:** Users can view their posted issues and track the status through a dedicated section in their dashboard.
  3. **Receive Notifications:** Users will receive notifications upon issue submission and when new suggestions or responses are received.

**Suggestion and Feedback:**

* 1. **View Feed:** Users can see issues and suggestions from others based on their selected familial role in their feed.
  2. **Provide Suggestions:** Users can respond to issues by providing feedback or suggestions, which will be visible to the original poster.

**Communication and Relationship Building:**

* 1. **Messaging:** Users can send and receive messages from other users based on suggestions or mutual interests.
  2. **Establish Relationships:** Users can initiate and manage relationships with others by viewing suggested connections and clicking on the "Relate" button.

**User Interface and Navigation:**

* 1. **Dashboard:** Users will have access to a dashboard with options to view and manage their issues, feedback, and messages.
  2. **Navigation Bar:** The navigation bar will include links to key sections such as "My Issues," "Feed," "Relations," and "Settings."

### 1.2 Non-Functional Requirements

**Security:**

* 1. **Data Protection:** Ensure the protection of user data, especially personal and sensitive information, through encryption and secure storage practices.
  2. **Authentication:** Implement secure authentication mechanisms, including OTP verification during sign-up.

**Performance:**

* 1. **Scalability:** The application must be able to handle increasing numbers of users and interactions without performance degradation.
  2. **Response Time:** Ensure that the application responds quickly to user inputs and actions, providing a seamless user experience.

**Usability:**

* 1. **User Experience:** The interface should be intuitive and easy to navigate, allowing users to perform actions with minimal effort.
  2. **Accessibility:** The application should be accessible to users with varying levels of technical proficiency and comply with accessibility standards.

**Reliability:**

* 1. **Availability:** Ensure high availability of the application, minimizing downtime and disruptions to user access.
  2. **Error Handling:** Implement robust error handling and logging mechanisms to manage and troubleshoot issues effectively.

### 1.3 Use Case Diagram

· **Create Issue:** User navigates to the issue posting page, fills out the issue details, and submits it.

· **View Issues:** User views their posted issues and status updates on their dashboard.

· **View Feed:** User accesses the feed to view issues and suggestions.

· **Respond:** User selects an issue and provides feedback or suggestions.

· **Build Relations:** User views suggested connections, initiates relationships, and interacts with connected users.

· **Send Message:** User navigates to the messaging section, selects a recipient, and sends a message.

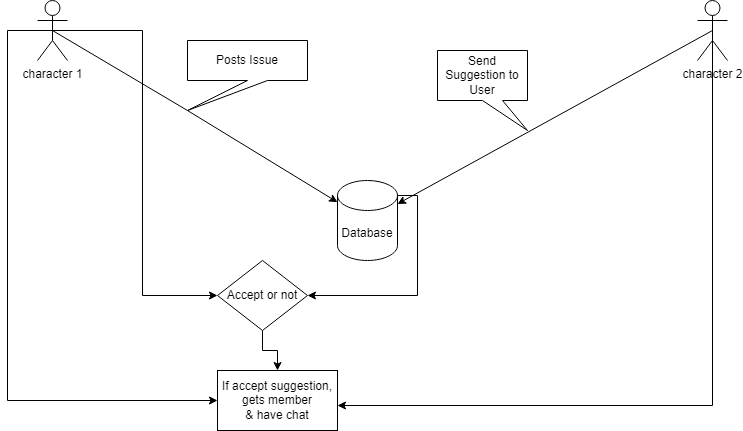


Fig 5.1 Use case diagram

### 1.4 Context Diagram

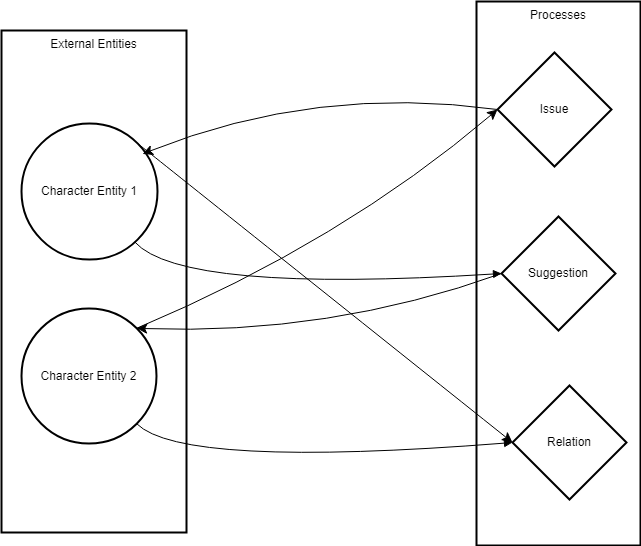


Fig 5.4: ContextDiagram

### 1.5 ER Diagram

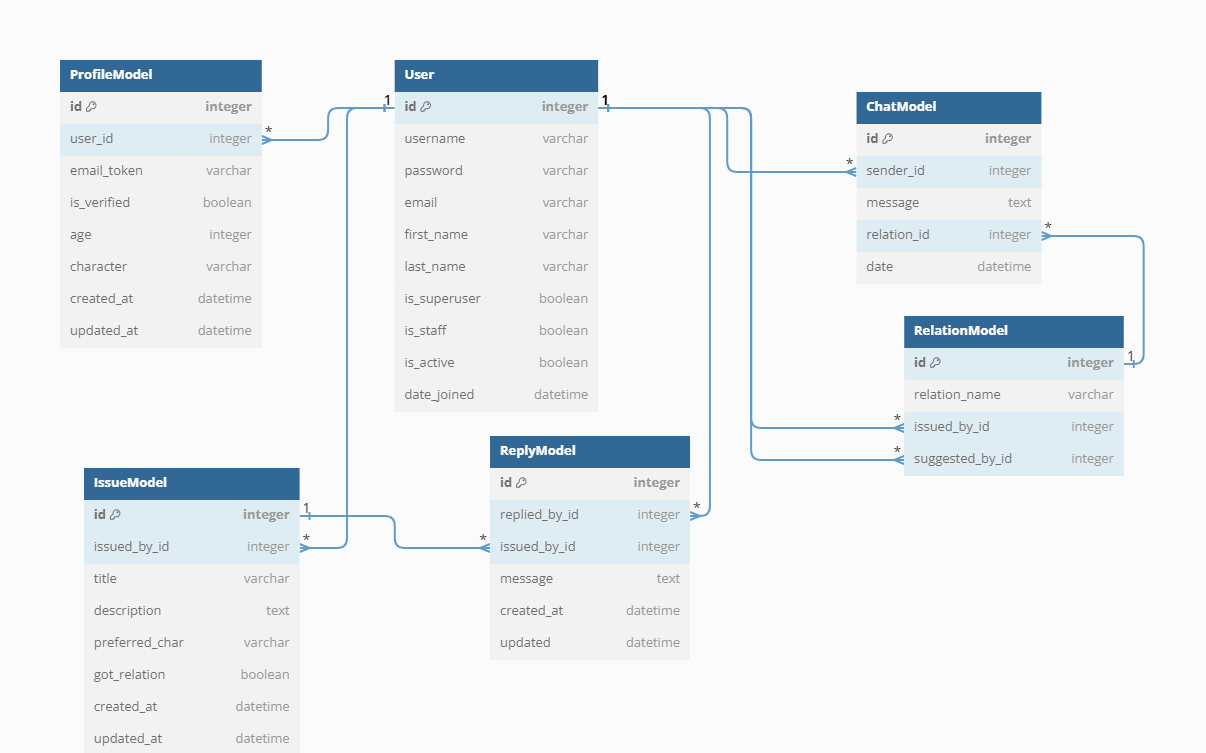
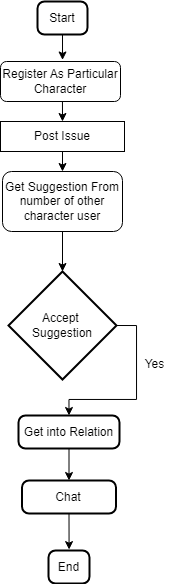


Fig 5.4: ER Diagram

### 1.6 Activity Diagram

****

## 5.2 System Design

* System design involves creating a detailed plan for implementing the Pariwar Web App, focusing on both the software architecture and the user experience. The design phase translates the requirements and analysis into a blueprint that guides the development process. Below are the key components of the system design:

### 5.2.1 System Architecture:

The Pariwar Web App is designed using a three-tier architecture:

#### Presentation Layer (Front-End):

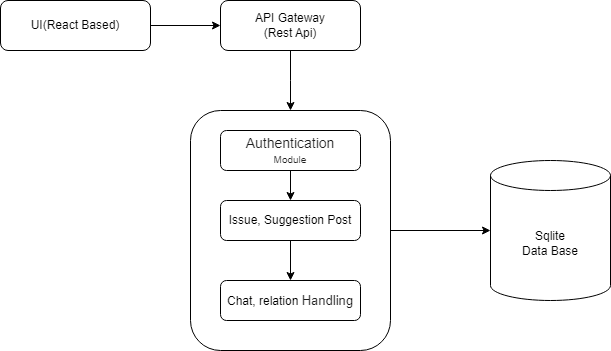
* + Developed using Next.js (React framework) for building the user interface.
  + Handles the user interface, including forms, navigation, and content display.
  + Communicates with the application layer through RESTful APIs.

#### Application Layer (Back-End):

* + Built using Django to manage server-side logic.
  + Processes user requests, handles business logic, and interacts with the database.
  + Implements authentication, authorization, and data validation.

#### Data Layer (Database):

* + Uses Sqlite as the primary database to store user profiles, Issue post, Post suggestion, Create relation and Chat.
  + Implements database indexing for optimized search queries.

  
  
Fig 5.6: System Architecture Design

### 5.2.2 User Interface (UI) Design

UI design is centered on creating an intuitive, user-friendly interface that ensures a positive user experience across all devices.

* **Consistency:** Maintain a uniform design language throughout the app, with consistent use of colors, fonts, and button styles.
* **Components:** The contents in the web app are broken down into components to ensure code reuse and efficiency.
* **Responsive Design:** The UI will adapt to various screen sizes and orientations, ensuring a seamless experience on devices like desktops, tablets, and mobile devices.

## 5.3 Implementation

The implementation phase is the critical stage where the design specifications are translated into functional code. In this chapter, we will explore the detailed process of implementing the Pariwar Web App, focusing on the technologies, tools, and methodologies used. The implementation is aligned with the structure and components outlined in the design phase and leverages modern web development practices to ensure efficiency, scalability, and maintainability.

### 5.3.1 Technology Stack

The choice of technology stack plays a pivotal role in the overall performance and user experience of the web app. For "Pariwar," the following technologies were selected:

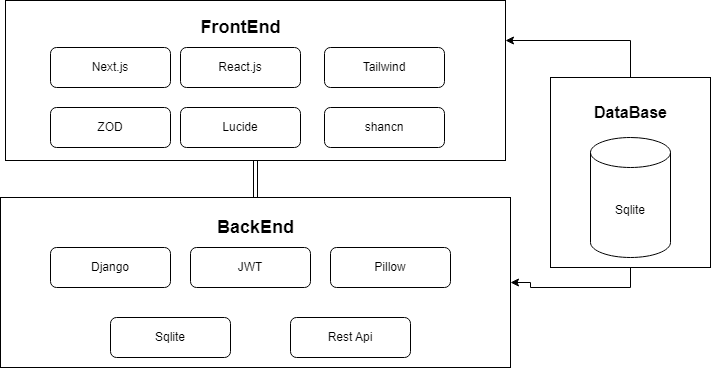


Fig 5.9: Technology Stack

#### Front-End Technology Stack

* + **Next.js (React Framework)**: Provides a server-side rendering capability, ensuring faster load times and better SEO performance. It also offers a flexible file-based routing system, making it easier to manage different pages within the app. (Docs | Next.js, 2024)
  + **React.js**: Used for building reusable UI components, enabling a dynamic and responsive user interface. React’s component-based architecture allows for efficient development and easier maintenance.
  + **Tailwind CSS**: A utility-first CSS framework that simplifies styling by providing a set of predefined classes, allowing for rapid UI design and ensuring a consistent look and feel across the application.
  + **Lucide React**: Lucide React was used to incorporate a wide range of customizable icons into the React components, significantly improving the app's visual appeal and user experience. (Lucide React, 2024)
  + **Shadcn**: It is a component library that provides react components that can be easily integrated to Next.js application. Forms, cards, toast, dialog, accordion, buttons, lists, alert, etc. are implemented in this project with the help of shadcn components. (shadcn, 2024)
  + **ZOD**: By defining schemas with Zod, the project can enforce strict validation rules on user inputs and API responses, preventing invalid or malicious data from being processed. This is particularly useful in form handling and API requests, where Zod automatically checks that incoming data meets the required structure and types. (Zod | Documentation, 2024)

#### Back-End Technology Stack:

The backend technology stack for the Pariwar web application has been carefully selected to provide a robust, secure, and scalable foundation for the application. Below are the key components of the backend technology stack:

**Django (Python):**

* 1. **Framework Overview:** Django is a high-level Python web framework that encourages rapid development and clean, pragmatic design. It follows the Model-View-Template (MVT) architectural pattern, which allows for efficient separation of concerns.
  2. **Role in the Project:** Django serves as the core framework for the backend, handling URL routing, request processing, and rendering views. It provides built-in features for authentication, session management, and security, making it an ideal choice for the development of a secure and scalable web application.

**REST API Framework:**

* 1. **Framework Overview:** Django REST framework (DRF) is a powerful and flexible toolkit for building Web APIs. It provides a simple, yet highly customizable, framework for creating RESTful APIs that can interact with the frontend.
  2. **Role in the Project:** The REST API framework is used to create the API endpoints that allow the frontend (built with React) to communicate with the backend. This includes endpoints for user authentication, issue posting, suggestion management, and more. DRF simplifies the process of serializing data and handling complex queries, ensuring smooth data exchange between the client and server.

**JWT (JSON Web Tokens):**

* 1. **Technology Overview:** JSON Web Tokens (JWT) are a compact, URL-safe means of representing claims to be transferred between two parties. They are commonly used for securely transmitting information between the client and server.
  2. **Role in the Project:** JWT is used for user authentication and authorization in the Pariwar application. After a user successfully logs in, a JWT is generated and sent to the client, where it is stored and included in subsequent API requests to verify the user’s identity. This stateless authentication method enhances security and scalability.

**Pillow:**

* 1. **Library Overview:** Pillow is a Python Imaging Library (PIL) fork that adds image processing capabilities to your Python applications. It is widely used for handling image files and performing tasks like resizing, cropping, and format conversion.
  2. **Role in the Project:** Pillow is used in the backend for handling user profile images and any other image-related functionality within the application. It ensures that images are appropriately processed and stored, maintaining the quality and performance of the application.

#### Data Base on ( ****SQLite ) :****

* 1. **Database Overview:** SQLite is a self-contained, serverless, zero-configuration, transactional SQL database engine. It is highly efficient and easy to set up, making it a popular choice for many web applications.
  2. **Role in the Project:** SQLite is used as the database management system for the Pariwar application. It stores user data, issues, suggestions, relationships, and other relevant information. Its lightweight nature makes it ideal for this project, where the focus is on simplicity and ease of deployment.

## 5.3.2 Frontend Implementation

The front-end of the Room Finder Web App was implemented using Next.js and React.js. The development process began by setting up the Next.js project, configuring the necessary files, and integrating Tailwind CSS for styling.

* **Page Structure**: The web app's pages are structured according to Next.js's file-based routing system. Key pages include the home page, search results, room details, user profile, and booking confirmation. Each page corresponds to a component in the React architecture, ensuring that the UI is built in a modular and reusable manner.
* **Component Development**: React components were developed for each section of the UI, including the header, footer, room listing, and search bar. Components were designed to be reusable, meaning that the same component could be used in different parts of the app with varying data inputs. For instance, the RoomCard component is used both on the search results page and the user's profile page, but with different data sources.
* **State Management**: React's useState and useEffect hooks were extensively used to manage state within components, ensuring that the UI reacts to changes in data in real-time. For more complex state management across different components, the Context API was used, allowing for a global state that could be accessed by any component in the tree.
* **Styling**: Tailwind CSS was integrated to handle the styling of the web app. The utility-first approach of Tailwind allowed for rapid development, with classes being applied directly to elements to control layout, spacing, typography, and more. This also made it easier to maintain a consistent design across the app.
* **Icons:** Lucide React was used to incorporate a wide range of customizable icons into the React components, significantly improving the app's visual appeal and user experience. By using Lucide React, the project could easily include scalable vector icons that are consistent in style, lightweight, and responsive across different devices. This helped in creating intuitive and aesthetically pleasing UI elements, such as buttons, navigation menus, and status indicators, which are crucial for user interaction.

## 5.3.3 Backend Implementation

The backend of the Pariwar Web App was meticulously designed and implemented using Django, a powerful and flexible web framework. The backend is responsible for managing data, business logic, and communication with the frontend through APIs. Below is a breakdown of the key components involved in the backend implementation:

**API Development:**

* 1. **Overview:** The Pariwar Web App uses Django REST Framework (DRF) to develop RESTful APIs. These APIs serve as the communication bridge between the frontend (React) and the backend, facilitating data exchange and user interactions.
  2. **Implementation Details:**
     1. **Endpoints:** Various endpoints were created to handle user authentication, issue posting, suggestion management, messaging, and relationship building.
     2. **Serialization:** DRF's serializers were used to convert complex data types, like Django QuerySets, into JSON, which is easily consumed by the frontend.
     3. **Routing:** The API routes were defined using Django's URL routing system, ensuring a clean and organized structure that aligns with RESTful principles.
     4. **Security:** JWT (JSON Web Tokens) were integrated to secure API requests, ensuring that only authenticated users can access protected resources.

**Database Design:**

* 1. **Overview:** The database design was based on SQLite, a lightweight and efficient relational database. The design focused on ensuring data integrity, normalization, and efficient data retrieval.
  2. **Implementation Details:**
     1. **Models:** Django's ORM (Object-Relational Mapping) was used to define models representing different entities such as users, issues, suggestions, and relationships.
     2. **Relationships:** Proper relationships (one-to-many, many-to-many) were established between models to represent real-world connections, such as users and their posts or feedback.
     3. **Migrations:** Django’s migration system was utilized to manage schema changes over time, allowing for smooth database evolution as the application grows.
     4. **Data Constraints:** Constraints such as unique fields, foreign keys, and non-nullable columns were enforced to maintain data consistency.

**OTP-Based Authentication:**

* 1. **Overview:** To enhance security and ensure user authenticity, an OTP (One-Time Password) based authentication system was implemented during user registration.
  2. **Implementation Details:**
     1. **OTP Generation:** A secure, random OTP is generated and sent to the user’s email during the registration process.
     2. **Verification Process:** The user is required to enter the OTP to verify their email address and activate their account.
     3. **Expiry and Resend:** OTPs have a limited validity period to enhance security, and users can request a new OTP if needed.

**Image Handling:**

* 1. **Overview:** The handling of user-uploaded images, such as profile pictures, was managed using the Pillow library, integrated with Django.
  2. **Implementation Details:**
     1. **Image Upload:** Users can upload images as part of their profile or issue posts. These images are processed and stored securely on the server.
     2. **Processing:** Pillow is used to resize and optimize images before saving them, ensuring they are efficiently managed in terms of storage and load time.
     3. **Validation:** Image files are validated for type, size, and dimensions to prevent uploading of inappropriate or excessively large files.

**Error Handling and Validation:**

* 1. **Overview:** Robust error handling and input validation were implemented to ensure the backend operates smoothly and securely.
  2. **Implementation Details:**
     1. **Form Validation:** User inputs are validated at both the form and API levels to prevent invalid data from being processed or stored.
     2. **Error Handling:** Custom error handlers were created to manage different types of exceptions, providing meaningful error messages to the frontend and logging issues for debugging purposes.
     3. **Response Codes:** Appropriate HTTP status codes are returned with API responses to inform the frontend of the success or failure of requests.

Chapter Six   
  
Api Documentaion

## Accounts API

### 1. Login API

* **Endpoint:** /api/login/
* **Method:** POST

#### ****Request Body:****

{

"username": "string",

"password": "string"

}

#### ****Response:****

{"posted": true,"user\_id": integer,"user\_name": "string","access": "string","refresh": "string"

}

### 2. Register API

* **Endpoint:** /api/register/
* **Method:** POST

#### ****Request Body:****

{"username": "string","email": "string","password": "string","age": integer,"character": "string"

}

#### ****Response:****

{"posted": true,"id": integer

}

## Profile API

### 1. Get Profile API

* **Endpoint:** /api/get\_profile/
* **Method:** POST

#### ****Request Body:**** User id

{"id": integer

}

#### ****Response:****

{"success": true,"data": {

"id": integer,

"username": "string",

"email": "string",

"is\_verified": boolean,

"age": integer,

"character": "string"

}

}

## Issue Segment

### 1. Post Issue API

* **Endpoint:** /api/post\_issue/
* **Method:** POST

#### ****Request Body:****

{"id": integer,"title": "string","description": "string","preferred\_char": "string"

}

#### ****Response:****

{"posted": true,"payload": {

"id": integer,

"title": "string",

"description": "string",

"preferred\_char": "string"

}

}

### 2. Get All Issues API

* **Endpoint:** /api/get\_all\_issues/
* **Method:** POST

#### ****Request Body:****

{"id": integer

}

#### ****Response:****

{"success": true,"payload": [

{

"id": integer,

"title": "string",

"description": "string",

"preferred\_char": "string"

},

{

"id": integer,

"title": "string",

"description": "string",

"preferred\_char": "string"

},

...

]

}

### 3. Get Issues by Character API

* **Endpoint:** /api/get\_issue\_character/
* **Method:** POST

#### ****Request Body:****

{"id": integer

}

#### ****Response:****

{"success": true,"payload": [

{

"id": integer,

"title": "string",

"description": "string",

"preferred\_char": "string"

},

{

"id": integer,

"title": "string",

"description": "string",

"preferred\_char": "string"

},

...

]

}

### 4. Get Issues by User API

* **Endpoint:** /api/get\_issue\_by\_user/
* **Method:** POST

#### ****Request Body:****

{"id": integer

}

#### ****Response:****

{"success": true,"payload": [

{

"id": integer,

"title": "string",

"description": "string",

"preferred\_char": "string"

},

{

"id": integer,

"title": "string",

"description": "string",

"preferred\_char": "string"

},

...

]

}

### 5. Get Specific Issue API

* **Endpoint:** /api/get\_specific\_issue/
* **Method:** POST

#### ****Request Body:****

{"issue\_id": integer

}

#### ****Response:****

{"success": true,"payload": {

"id": integer,

"title": "string",

"description": "string",

"preferred\_char": "string"

}

}

## Reply / Suggestion Segment

### 1. Post Reply API

* **Endpoint:** /api/post\_reply/
* **Method:** POST

#### ****Request Body:****

{"issued\_id": integer,"reply\_user\_id": integer,"message": "string"

}

#### ****Response:****

{"posted": true,"payload": {

"id": integer,

"message": "string",

"created\_at": "datetime",

"updated": "datetime",

"replied\_by": {

"id": integer,

"username": "string"

},

"issued\_by": integer

}

}

### 2. Get Issue Replies API

* **Endpoint:** /api/get\_issue\_reply/
* **Method:** POST

#### ****Request Body:****

{"id": integer

}

#### ****Response:****

{"success": true,"payload": [

{

"id": integer,

"message": "string",

"created\_at": "datetime",

"updated": "datetime",

"replied\_by": {

"id": integer,

"username": "string"

},

"issued\_by": integer

},

{

"id": integer,

"message": "string",

"created\_at": "datetime",

"updated": "datetime",

"replied\_by": {

"id": integer,

"username": "string"

},

"issued\_by": integer

},

...

]

}

### 3. Get All Replies API

* **Endpoint:** /api/get\_reply\_list/
* **Method:** POST

#### ****Request Body:****

{"issued\_id": integer

}

#### ****Response:****

{"success": true,"payload": [

{

"id": integer,

"message": "string",

"created\_at": "datetime",

"updated": "datetime",

"replied\_by": {

"id": integer,

"username": "string"

},

"issued\_by": integer

},

{

"id": integer,

"message": "string",

"created\_at": "datetime",

"updated": "datetime",

"replied\_by": {

"id": integer,

"username": "string"

},

"issued\_by": integer

},

...

]

}

## Relation Segment

### 1. Update Relation API

* **Endpoint:** /api/relation\_update/
* **Method:** POST

#### ****Request Body:****

{"suggested\_by": integer,"issued\_by": integer

}

#### ****Response:****

{"posted": true,"payload": {

"id": integer,

"suggested\_by": integer,

"issued\_by": integer,

"relation\_name": "string"

}

}

### 2. Get User Relations API

* **Endpoint:** /api/get\_particular\_relation/
* **Method:** POST

#### ****Request Body:****

{"id": integer

}

#### ****Response:****

{"posted": true,"payload": [

{

"id": integer,

"suggested\_by": {

"id": integer,

"username": "string"

},

"issued\_by": {

"id": integer,

"username": "string"

},

"relation\_name": "string"

},

{

"id": integer,

"suggested\_by": {

"id": integer,

"username": "string"

},

"issued\_by": {

"id": integer,

"username": "string"

},

"relation\_name": "string"

},

...

]

}

## Chat API

### Get Chat by Relation ID

* **Endpoint:** /api/chat/get/
* **Method:** POST

#### ****Request Body:****

{"relation\_id": integer

}

#### ****Response (Success):****

{"success": true,"payload": [

{

"username": "string",

"sender": {

"id": integer,

"username": "string"

},

"message": "string",

"relation": integer,

"date": "datetime"

},

...

]

}

#### ****Response (Failure):****

{"success": false,"message": "string"

}

### Post Chat

* **Endpoint:** /api/chat/post/
* **Method:** POST

#### ****Request Body:****

{"sender\_id": integer,"message": "string","relation\_id": integer

}

#### ****Response (Success):****

{"success": true

}

#### ****Response (Failure):****

{"success": false,"message": "string"

}

# Chapter Seven Conclusion and Recommendation

## 7.1 Conclusion

## 7.2 Limitations

## 7.3 Recommendations

# Chapter Eight References