CONNECTIFY

PROJECT REPORT SUBMITTED TO MAHATMA GANDHI UNIVERSITY, IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF BACHELOR OF SCIENCE IN COMPUTER APPLICATIONS

BY

AMAL MATHEW JOSE

220021083188

DEPT. OF COMPUTER SCIENCE

B.V.M HOLY CROSS COLLEGE

CHERPUNKAL, KOTTAYAM 686 584

March 2025

DEPARTMENT OF COMPUTER SCIENCE

BVM HOLY CROSS

Certificate

Certified that the report entitled “CONNECTIFY” is the bonafide record of the project work done by Mr. AMAL MATHEW JOSE(220021083188) under our guidance and supervision and is submitted in partial fulfillment of the Bachelor degree in Computer Applications, awarded by Mahatma Gandhi University, Kerala.

Mrs. Dithy Mary Thomas

Project Guide

Mr. Binu M. B

Head of the Department

Rev. Dr. Baby Sebastian Principal

Submitted for Project Evaluation on ------/-----/------

Internal Examiner

External Examiner

DECLARATION

I here by declare that the project work entitled CONNECTIFY submitted in partial fulfillment of the requirements for the award of the Bachelor of degree in Computer Applications from BVM Holy Cross College, Cherpunkal, is record of bonafide work done under the guidance of Mrs. Dithy Mary Thomas (Assistant professor, Department of computer applications).

Place: (Signature)

Date: Name:Amal Mathew Jose

Reg. no:220021083188

ACKNOWLEDGMENT

Dedicating this main project to the Almighty of God whose abundant grace and mercies enabled is successful completion, we would like to express our profound gratitude to all the people who had inspired and motivated us to make this main project a success. We express our heartfelt thanks to our principal Dr. Baby Sebastian Thonikuzhi for his warming support and suggestions in accomplishing our project. We would like to place our deep sense of gratitude to Mr. Binu M.B (Head of department of Computer Applications), for his guidance in carrying out this project work and project guidance for his valuable guidance, suggestions and assessment throughout the project. We also extent our sincere thanks to all other faculty members of the department of Computer Applications for their assistance and encouragement throughout the project. Last, we would like to thank our friends for their co-operations and encouragement.

ABSTRACT

This project is a web-based real-time chat application named **Connectify**, developed using Django for the backend and React for the frontend. It aims to deliver a fast, responsive, and secure messaging experience that closely mimics the core features of WhatsApp, while maintaining simplicity, performance, and a great user experience. The application facilitates smooth communication between users through one-on-one chats and group interactions.

Connectify allows users to send and respond to friend requests, create public or private groups (with join keys), and engage in real-time conversations. The chat system supports rich messaging features such as delivery/read indicators, typing and online status, real-time message history, and the ability to send various file types, including images, videos, and documents. Users can also delete messages in real time, ensuring full control over their conversations.

For data storage and efficiency, SQLite is used for lightweight user authentication, MongoDB handles chat and group data, and Redis is used for storing user/group metadata and managing WebSocket-based message brokering. JWT is used for secure authentication, and all file uploads are handled via WebSockets with strict server-side validation.

Connectify also provides personalization options such as custom UI themes and backgrounds to enhance the user experience. The frontend leverages React for building dynamic components, ensuring a clean and intuitive interface. Privacy is a key focus—administrators have limited capabilities, such as viewing or deleting users, without access to private messages.

By combining real-time performance, rich features, and a privacy-first approach, **Connectify** stands as a modern, scalable solution for social communication in a digital-first world.

CONTENTS

1. INTRODUCTION

1.1 PROJECT OVERVIEW

The main objective of the **Connectify** is to create a fast, intuitive, and feature-rich messaging platform inspired by WhatsApp, tailored for seamless communication between users. The system aims to simplify and enhance the experience of chatting, group interactions, and media sharing in real time, with a strong focus on scalability, performance, and modern UI/UX.

Key Features:

- **User Connectivity:** Allows users to send and receive friend requests, accept or reject them, and maintain a curated friend list for private one-on-one messaging.
- **Group Management:** Users can create public groups (open to all) or private groups (joinable via a unique key). Group members can view participant lists and interact in real time.
- **Authentication & Security:** Implements JWT-based authentication for secure login and session management, ensuring personalized and protected access.
- **Real-Time Messaging:** Powered by Django Channels and WebSockets, the chat system supports instant messaging with typing indicators, online status, read and delivered receipts, and message deletion in real time.
- **File Sharing & Media Support:** Enables users to send any type of file—including images, videos, and documents—up to 80MB directly via WebSocket, with server-side validation for file size and basic metadata tracking.
- **Custom User Experience:** Offers customizable UI themes and chat backgrounds for a more personalized experience, along with a smooth interface using React, Zustand, and Framer Motion for responsive interactions.
- **Data Handling**: Utilizes MongoDB for efficient storage of chat data and Redis for handling user/group metadata and real-time message brokering, while SQLite is reserved solely for authentication.

The system not only enhances operational efficiency but also ensures a smooth and transparent transaction process, contributing to a better dining experience for the college community.

1.2 ORGANIZATION PROFILE

Organization Name	**:**	BVM Holy Cross College
Location	**:**	Cherpunkal, Kottayam, Kerala, India
Year of Establishment	**:**	1995
Type of Institution	**:**	Self-Financing College
Affiliation	**:**	Mahatma Gandhi University
Management	**:**	Holy Cross Forane Church, Cherpunkal
Mission	**:**	To provide quality education that promotes academic excellence, holistic development, and social commitment

Core Values:

- Academic Integrity
- Student-Centric Approach
- Social Responsibility
- Inclusiveness
- Excellence in all Endeavours

Contact Information:

BVM Holy Cross College

Cherpunkal, Kottayam, Kerala686529

Phone :04822-267520,

9446640157

Email: bvmhcc@gmail.com Website: https://bvmcollege.com

2. SYSTEM CONFIGURATION

2.1 HARDWARE CONFIGURATION

The selection of hardware is very important in the existence and proper working of any of the software. When selecting hardware, the size and capacity requirements are also important. The hardware must suit all application developments.

Processor	: Intel i5 10th gen
RAM	: 8 GB
Storage	: 512 GB
Monitor	: Standard display resolution

2.2 SOFTWARE CONFIGURATION

One of the most difficult tasks is selecting software, once the system requirement is found out then we have to determine whether a particular software package fits for those system requirements. This section summarizes the application requirement.

Operating System	: Windows 11
Web Server	: Daphne (ASGI Server)
Database	: Mongo DB,
Browser	: Chrome, Microsoft Edge
Text Editor	: VS Code
Front-End	: React, JavaScript
Back-end	: Django (Django Channels)

3. SYSTEM ANALYSIS

3.1 PRELIMINARY INVESTIGATION

System Analysis is the process of gathering information, identifying system requirements, and evaluating existing workflows in order to design an improved solution. A system is a collection of interacting components working together to achieve specific goals. In the context of this connectify application, the analysis focused on addressing limitations in traditional messaging systems and creating a modern, scalable alternative.

The main objective of the system analysis was to evaluate existing real-time communication platforms and identify areas for improvement. The investigation led to the decision to develop a mini chat system with features such as real-time messaging, friend and group management, media sharing, and an enhanced user interface.

3.2 EXISTING SYSTEM

Many traditional messaging systems lack real-time performance and modern features expected by users today. Messaging often suffers from latency, poor delivery status handling, and limited support for file sharing or presence indicators. Group chat capabilities are usually basic, without advanced access controls like private keys or participant management.

On the backend, monolithic systems relying solely on relational databases struggle to manage unstructured chat data efficiently. There's often no dedicated infrastructure for message brokering or metadata tracking, resulting in poor scalability and user experience—highlighting the need for a more modern, real-time, and responsive communication system.

User experience is another area where existing systems underperform. Limited customization, outdated interfaces, and lack of responsiveness often lead to a subpar experience—especially when compared to modern expectations shaped by platforms like WhatsApp and Telegram.

Overall, the limitations of these existing systems underscore the need for a more advanced, modular, and real-time solution that leverages modern tech stacks and delivers both performance and usability at scale.

3.3 PROPOSED SYSTEM

The proposed Connectify Application is a real-time, full-stack communication platform designed to replicate the core experience of WhatsApp in a lightweight, browser-based environment. Unlike traditional messaging systems that offer only basic chat functionality, this system introduces a well-structured and interactive communication flow between users and groups, supported by modern technologies to ensure speed, reliability, and scalability.

The application provides features such as sending and receiving friend requests, creating and joining both public and private groups, and exchanging real-time messages with delivery and read indicators. Additional enhancements include typing and online presence indicators, real-time message deletion, and access to message history. Users can also share various types of media and files directly through WebSocket-based transfers, ensuring seamless performance.

Powered by Django Channels on the backend with MongoDB for chat storage, Redis for metadata and message brokering, and JWT-based authentication via SQLite, the system is optimized for real-time responsiveness. The frontend, built with React, Axios, Zustand, and Framer Motion, provides a smooth and customizable user experience with support for custom themes and backgrounds. By addressing the limitations of older systems, this chat app delivers a modern, efficient, and highly engaging communication platform for today's users.

3.4 FEASIBILITY ANALYSIS

Feasibility refers to the degree to which a proposed system can be realistically developed and successfully implemented. To evaluate this, a feasibility study was conducted for the Connectify to determine whether the proposed features and technologies are practical, efficient, and suitable for deployment. The study considered critical factors such as resource availability, development cost, long-term maintenance, and user expectations.

The objective was to ensure the system is not only technically sound but also user-acceptable, cost-effective, and adaptable to future changes. The feasibility study examined whether the

application could be built using the current tech stack—Django Channels, React, MongoDB, Redis, and WebSockets—while staying within resource and timeline constraints.

Various other objectives of the feasibility study are listed below:

- Ensuring the system meets communication needs similar to modern messaging platforms.
- Verifying that existing infrastructure supports the use of real-time technologies like WebSockets.
- Confirming the possibility of integrating components like JWT authentication, custom UI themes, and file sharing.

Various types of feasibility that we checked include technical feasibility , operational feasibility, and economic feasibility.

3.4.1 TECHNICAL FEASIBILITY

Technical feasibility focuses on evaluating whether the existing technological resources—such as frameworks, libraries, and developer expertise—are sufficient to successfully implement the chat application within the given timeline and constraints. This involves assessing the suitability and reliability of selected tools, as well as the team's proficiency in using them.

In this project, the development team has solid expertise in Django, Django Channels, and WebSocket-based communication, along with frontend skills in React, Axios, Zustand, and Framer Motion. Additionally, the backend stack includes MongoDB and Redis, which are well-suited for handling real-time data and message brokering. Given this strong technical foundation, the system is deemed fully feasible from a technical standpoint.

- **Backend (Django)**: Django is a powerful, secure, and scalable Python web framework, ideal for building real-time web apps with built-in features like routing, ORM, and middleware support.
- **Database (SQLite + MongoDB + Redis)**: SQLite handles user authentication. MongoDB stores chat and group data, while Redis manages metadata and real-time message brokering.

- **Frontend (React)**: React is used to build fast, responsive, and dynamic user interfaces for the chat application

3.4.2 OPERATIONAL FEASIBILTY

Operational feasibility evaluates how effectively the proposed chat application addresses user needs and integrates into expected usage patterns, taking into account system usability and user acceptance. It involves assessing whether users can easily adapt to the platform and whether the system supports smooth daily operations.

The chat system is designed with a focus on user experience, offering a clean, responsive interface and using familiar messaging conventions such as chat lists, message status indicators, and customizable UI elements. The platform has received positive feedback during internal testing, indicating strong user acceptance and adaptability. As a result, the system is considered operationally feasible and well-suited for real-world deployment.

3.4.3 ECONOMIC FEASIBILITY

Economic feasibility evaluates whether the benefits of the proposed chat application justify the associated costs. This includes assessing development, maintenance, and operational costs in comparison to the expected benefits, such as increased efficiency and user engagement.

While there are ongoing expenses for server hosting, database management, and minor maintenance, these costs are outweighed by the system's efficiency in automating communication, reducing the need for manual interventions, and enhancing user experience. The long-term benefits, such as scalability, reduced operational overhead, and user satisfaction, make this project economically feasible.

3.5 ADVANTAGES OF PROPOSED SYSTEM

- **Real-Time Communication**: Automation simplifies the ordering process, reducing wait times and minimizing errors associated with manual order taking.

- **Enhanced Privacy and Security**: By utilizing JWT for authentication and WebSocket technology for real-time communication, the system ensures secure and private conversations.
- **Customizable User Experience**: Users can personalize their chat experience with custom UI preferences and backgrounds, which enhances engagement and user satisfaction.
- **Efficient Group Management**: The system supports both public and private groups with secure join-key mechanisms, making group interactions more organized and accessible.
- **Real-Time Message Indicators**: With features like read/delivery status and typing indicators, users are kept informed about the status of their messages.
- **Scalable and Future-Proof**: Leveraging a robust backend with MongoDB, Redis, and Django Channels ensures the application can handle increased user traffic and scale effortlessly.
- **Media Sharing Capabilities**: The system supports sharing various media files, enabling users to exchange diverse types of content without limitations.

3.6 REQUIREMENT SPECIFICATION

The Software Requirement Specification (SRS) defines the technical specifications and functional requirements for the proposed chat application, ensuring a clear understanding between the developer and the end users. This document formalizes the system's behavior, user interactions, and validation criteria, ensuring that the application meets user expectations and adheres to performance and security standards.

The proposed system has the following requirements:

- Secure user authentication and account management using JWT.

- Management of friendship and groups with request handling and join mechanisms.
- Real-time messaging with history, typing indicators, and delivery/read status.
- Media sharing support for files, images, and videos via WebSocket.
- Real-time message deletion and search functionality across chats and users.
- Customizable UI with strong privacy and security measures.

4. SYSTEM DESIGN

4.1 INTRODUCTION

The system design phase is focused on converting the identified requirements into a logical and structured architecture for the connectify application. This stage is critical to laying out how each component of the system will interact, perform, and deliver the intended functionality. It is typically divided into two main sub-phases:

1. **Primary Design Phase:** At this level, the system is broken down into major functional blocks such as user management, messaging service, media handling, group management, and real-time communication. Each block is designed to minimize external dependencies and internalize closely related operations to improve performance and reduce complexity.

2. **Secondary Design Phase:** This phase involves designing each block in detail. For example, the messaging module would include real-time delivery, read receipts, and message deletion mechanics, while the media module would define file size checks, metadata handling, and socket-based transmission logic.

   The general tasks involved in the design process are the following:

   1. Designing the core blocks such as user authentication, chat system, group management, and file handling.

   2. Breaking down each block into smaller, well-defined modules for easier implementation and testing.

   3. Structuring databases—using SQLite for auth and MongoDB for chats and metadata—to store user, group, and message data efficiently.

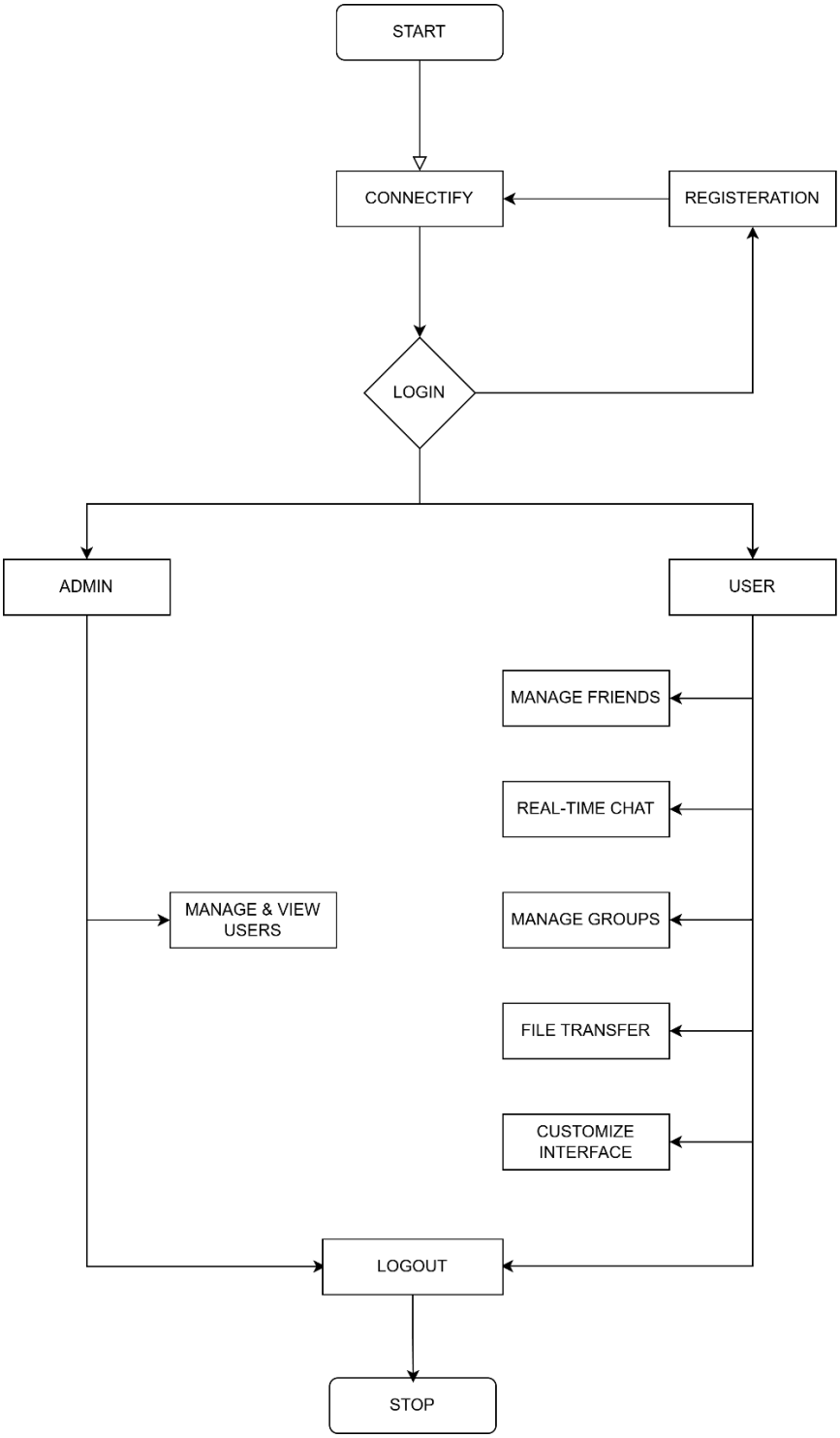
   4. Defining program logic and workflows that will power real-time messaging and user interactions.

   5. Designing the data flow between the frontend and backend, and specifying the structure of user inputs and system outputs.

   6. Creating detailed documentation of the overall system architecture and design choices.

   7. Reviewing the system design for performance, scalability, and security considerations.

4.2 SYSTEM FLOWCHART



4.3 DATABASE DESIGN

Database design is a crucial aspect of developing the Connectify chat application. It involves creating a structured model to efficiently organize and manage data across the system. In this project, SQLite is used for lightweight user authentication, while MongoDB is used to store and manage dynamic data such as messages, group information, and chat history. This separation ensures better performance, scalability, and data integrity. The design aims to reduce redundancy, enhance data retrieval speed, and ensure accuracy, privacy, and security. By structuring the databases efficiently, the system supports real-time communication and a seamless user experience.

PRIMARY KEY

In SQLite, primary keys uniquely identify records with non-null constraints. In MongoDB, each document has a unique _id field that serves the same purpose.

FOREIGN KEY

SQLite supports foreign keys to reference primary keys of other tables. In MongoDB, references are made by storing the _id of related documents, enabling flexible linking without strict foreign key constraints.

NORMALIZATION

Normalization is applied in SQLite to reduce redundancy and maintain data integrity through structured tables. In MongoDB, data is often denormalized for performance, but referencing is used to manage relationships efficiently.

FIRST NORMAL FORM (1NF)

In 1NF, each field should hold only atomic values with unique column names. In SQLite, this ensures no repeating groups. In MongoDB, documents are structured to avoid arrays of unrelated data within a field.

SECOND NORMAL FORM (2NF)

2NF eliminates partial dependencies in SQLite, ensuring non-key fields fully depend on the primary key. MongoDB achieves similar structure by organizing related data within nested documents or separate collections.

THIRD NORMAL FORM (3NF)

In 3NF, SQLite ensures that non-key fields are only dependent on the primary key, not on other non-key fields. MongoDB maintains clean design by avoiding unnecessary nested dependencies and separating concerns across collections.

COLLECTIONS

Collection 1: user_mongo

To store the basic details of user

Field name	Data type	Width	Description
_id	ObjectId	24 chars	Unique MongoDB document ID
sqlite_id	UUID (String)	36 chars	User auth UUID
user_name	String	255	Name of the user
email	String	254	User's unique email address
password	String	100	Hashed password
profile_picture	String	500	Image Link of User Profile
created_at	ISODate		User Sign up TimeStamp

Primary Key: _id

Collection 2: user_data_mongo

To store the user additional data

Field name	Data type	Width	Description
_id	ObjectId	24 chars	Unique MongoDB document ID
friends	Array[ObjectId]		User friend list
request	Array[ObjectId]		User friend requests

notifications	Array[String]		User notifications

Primary Key: _id

Collection 3: message_mongo

To store the user message details

Field name	Data type	Width	Description
_id	ObjectId	24 chars	Unique MongoDB document ID
sender	ObjectId	24 chars	Sender user ID
receiver	ObjectId	24 chars	Receiver user ID
c_id	String	48 chars	Conversation ID
content	String		Message content
type	Integer	1 digit	Message Type
status	Integer	1 digit	Message Status
time	ISODate		Message Time

Primary Key: _id

Foreign key: sender, receiver

Collection 4: conversations_mongo

To store the each user to user conversations

Field name	Data type	Width	Description
_id	ObjectId	24 chars	Unique MongoDB document ID
c_id	String	48 chars	Conversation
prtcpnt	Array[ObjectId]	2 items	List of participants
lst_m	String		Last Message
l_s	Integer	1 digit	Last Message Status
time	ISODate		Last Message time

Primary Key: _id

Foreign key: prtcpnt

Collection 5: chat_group_mongo

To store the details groups

Field name	Data type	Width	Description
_id	ObjectId	24 chars	Unique MongoDB document ID
name	String	100 chars	Group name
group_key	String	100 chars	Group key
join_key	String	100 chars	Join key
created_at	ISODate		Group Create time
created_by	String		Group Creator
is_private	Boolean	1	Group type

Primary Key: _id

4.4 DATA FLOW DIAGRAM

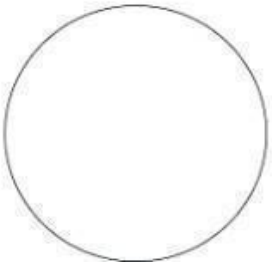
A Data Flow Diagram (DFD) is a visual representation that illustrates how data moves through the Connectify system. It maps out the flow of information from external users, how it is processed in various modules (like chat, groups, and authentication), and how it's stored in databases such as SQLite and MongoDB. Key DFD components include external entities (users/admin), processes (message handling, friend requests), data stores (authentication and chat storage), and data flows (arrows indicating movement). DFDs help simplify and visualize the system logic by focusing on how data interacts rather than technical implementation

The symbols used in DFD are shown below:

It represents data source or destination

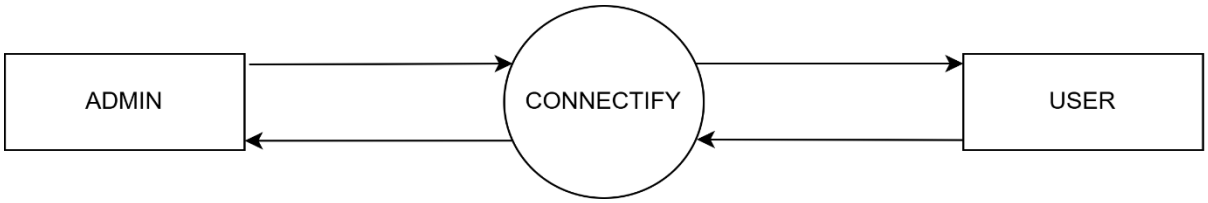
It represents the data store

It represents the data flow

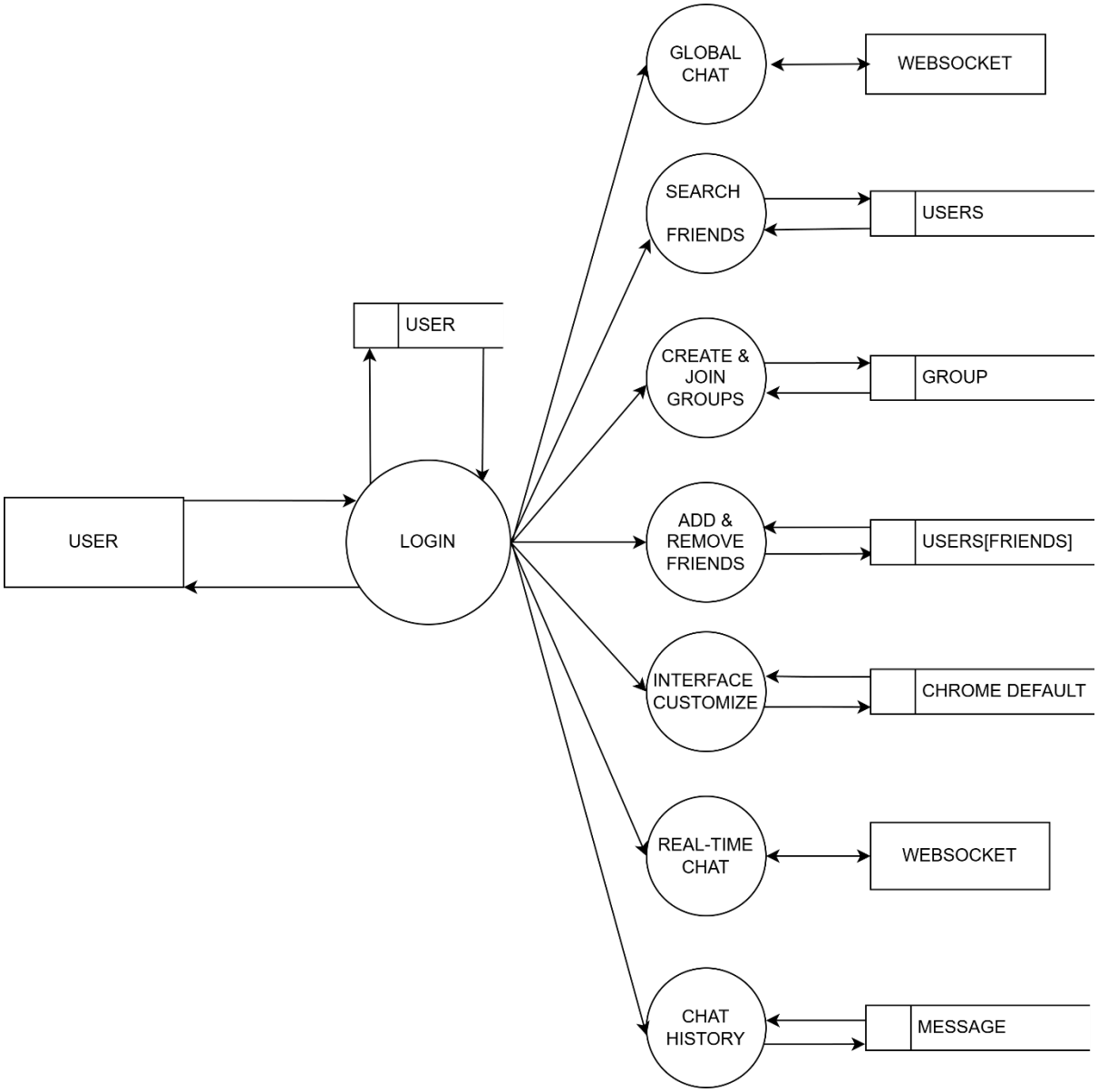


It represents a process

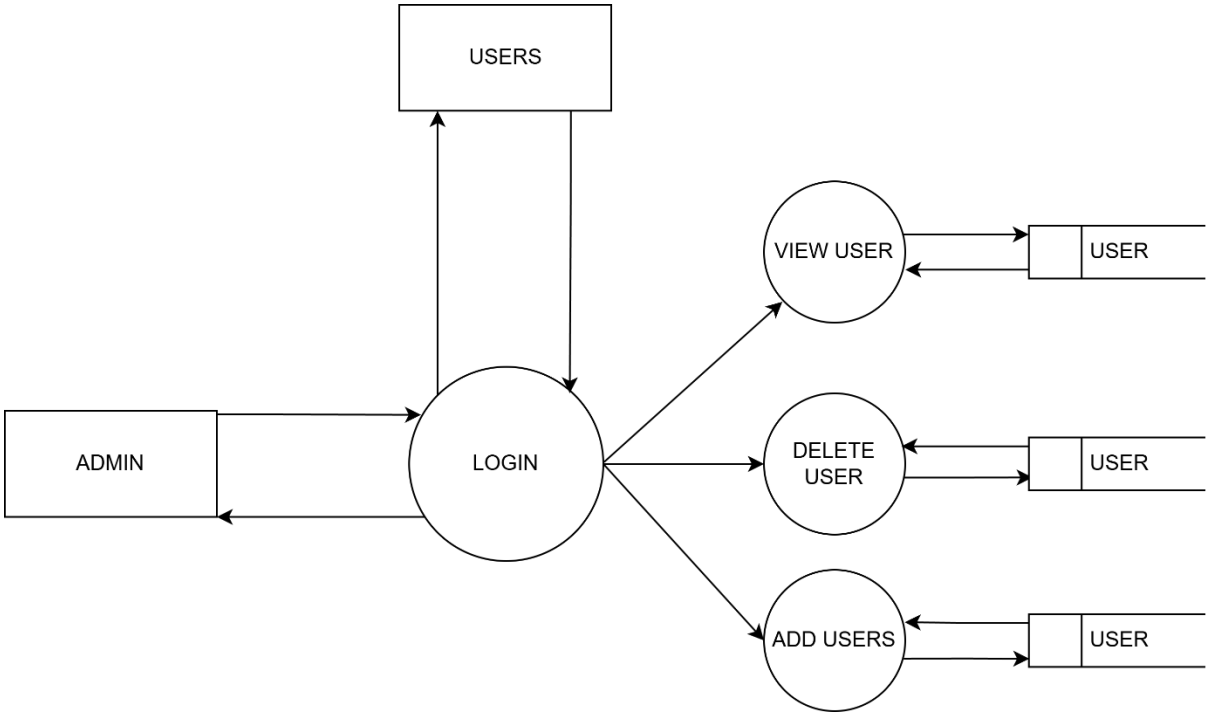
CONTEXT DIAGRAM



LEVEL 1 USER



LEVEL 1 ADMIN



4.5 INPUT DESIGN

User interface design plays a critical role in any application, especially one focused on real-time communication like our chat app. Input design is the process of translating user-friendly data input into a format the system can understand. The interface allows users to interact smoothly, whether they're logging in, sending messages, uploading files, or creating groups.

Forms and input fields in the system are designed to be intuitive, with labels and guidance to help users provide accurate information. Input validation is applied where necessary—like when sending media, authenticating users, or joining groups with a key—ensuring only valid and relevant data enters the system. The design emphasizes ease of use, interactivity, and consistent layouts that prevent user errors and improve efficiency. This is especially important in our React-based frontend, where dynamic and responsive input handling enhances the user experience.

The basic steps involved in input design are:

- Converts user actions into system-readable format.
- Clean, simple forms for login, chat, groups, etc.
- Prompts and validations for error-free input.
- Consistent screen design across all modules.
- Real-time validation for secure data entry.
- React-based input, connected via API/WebSocket.

4.6 OUTPUT DESIGN

The output design of the connectify application focuses on delivering the right information to users in a clear and timely manner. In a real-time messaging environment, outputs play a crucial role in enhancing the user experience by ensuring that all communication and updates are instantly visible and accessible. Whether it's the display of messages, delivery/read indicators, online statuses, or group participant lists, the system is designed to present this information in an intuitive and user-friendly way.

The primary aim of the output design is to keep users informed about chat activities, friend interactions, and group updates in real-time. With the integration of React on the frontend, outputs are rendered dynamically, ensuring smooth transitions and instant updates without the need to reload the page. This design approach not only improves responsiveness but also supports decision-making and interaction by clearly conveying ongoing conversation status, file uploads, and message histories.

Efficient output design in this system supports both individual and group interactions, ensuring that users receive accurate, visually organized, and timely feedback for every action performed within the app.

5. SYSTEM DEVELOPMENT

5.1 INTRODUCTION

The Connectify application has been built using a structured software development approach that supports future updates and maintenance. It leverages modern technologies like Django, Daphne (ASGI), MongoDB, Redis, and React to ensure a scalable, efficient, and maintainable system.

A dual-database setup is used—SQLite for authentication and MongoDB for dynamic data such as messages and group info. Redis handles metadata and message brokering, enabling real-time performance. This architecture balances flexibility with performance and supports smooth transitions across all development phases.

Beyond code logic, the system emphasizes real-time communication, user privacy, smooth UX, and long-term scalability, making it adaptable to both agile and structured workflows.

5.2 MENU LEVEL DESCRIPTION

Admin Module:

- User Management View and manage registered users with limited access—primarily for user viewing and deletion to maintain privacy.
- Group Monitoring: Monitor group creation activity, view public and private groups.
- Privacy Respect: Admin access is intentionally limited to preserve user confidentiality and autonomy.

The admin's role in this chat app is minimal and privacy-respecting, focused on system hygiene rather than control over user content.

User Module:

- Registration & Login: Sign up and log in securely using JWT-based auth.
- Friend System: Send, accept, reject, and remove friend requests.
- Group Features: Join public groups or enter join key to access private groups.

- Chat System: Real-time messaging with delivery, read, typing, and online indicators.
- Media Sharing: Send text, files, images, videos.
- Chat History: View message history with real-time deletion and sync.

5.3 PROCESS SPECIFICATION

The Connectify Application System enhances real-time communication by allowing users to register with a username, email, and password, and securely log in using JWT authentication. After logging in, users can send and manage friend requests, initiate one-on-one or group chats, and view participant statuses. Both private and public groups are supported; users can join public groups freely or enter a key to access private ones. Users can send messages, files, images, and videos, with real-time indicators for message delivery, read status, and typing activity.

Messages are sent and received over WebSocket connections powered by Django Channels, allowing instant communication. Message history is maintained and can be retrieved at any time, with real-time deletion features synced across all clients. File uploads are directly transmitted over sockets, with server-side validation for size and metadata only.

Admins have limited controls focused on privacy—mainly viewing and deleting user accounts. Administrative tools prioritize user autonomy and data protection. The system uses Redis for real-time metadata like online presence, MongoDB for storing messages and group data, and SQLite for managing authentication. Error handling is built-in to manage invalid operations, connection issues, and file upload limits. Overall, the system offers an efficient and user-respecting approach to modern chat applications.

6. SYSTEM TESTING

6.1 TESTING METHODS

WHITE-BOX TESTING

White-box testing, also known as glass-box testing, is a method that tests the internal logic and structure of the code by examining control flow, conditions, and loops. Using white-box testing methods, the software engineer can derive test cases that

- Guarantee that all independent paths within a module have been exercised at least once.
- Exercise all logical decision on their true and false sides.
- Execute all loops at their boundaries and within their operational bounds.
- Exercise internal data structure to ensure their validity.

White-box testing was successfully conducted on our system. All independent paths within a module have been executed at least once and all logical decisions have been exercised on their true and false sides.

BLACK-BOX TESTING

Black-box testing also called behavioural testing, focuses on the functional requirements of the software. It is a complementary approach that is likely to uncover a different class of errors than white-box methods.

Black-box testing attempts to find errors in the following categories.

- Incorrect or missing functions
- Interface-level errors across modules.
- Errors in accessing data from MongoDB or Redis.
- Behavioral or performance-related issues.
- Initialization and termination handling problems.

Black-box testing was successfully conducted on our system. The system was divided into a number of modules and testing was conducted on each module. We have tested the system for incorrect or missing functions and interface errors. Performance errors and the flow of information between modules ensuring interface.

ALPHA TESTING:

Alpha testing, also known as acceptance testing, is performed to validate the system against user requirements before final release. It is typically conducted in-house by the development team in collaboration with the client or stakeholders.

BETA TESTING:

Beta testing is used when the system is treated as a software product and involves releasing it to a limited group of users outside the development team. In Connectify, the app was provided to selected real users who tested the platform in actual usage conditions. These users reported any issues or unexpected behavior to the developers. This testing helped uncover real-world errors and usage patterns that were not identified during internal testing

UNIT TESTING

Unit testing involves testing individual components of the system by the developer before integration. In Connectify, each module—such as authentication, friend management, group features, and messaging—was tested separately during development. This helped in identifying and fixing bugs early. Exhaustive unit testing ensured data validity and improved overall system reliability. Testing each part at the time of development made it easier to catch and correct issues immediately.

INTEGRATION TESTING

Integration testing is done after unit testing to verify how well individual modules work together. In Connectify, after testing modules like user authentication, chat, groups, and file handling separately, they were combined and tested as a whole. This ensured smooth data flow and communication between dependent components. Some different types of integration testing are big-bang, mixed (sandwich), risky-hardest, top-down, and bottom-up. Unit testing uses modules for testing purpose, and these modules are combined and tested in integration testing. The main goal was to confirm that integrated parts functioned correctly as a unified system and met all functional requirements before system-level testing.

ACCEPTANCE TESTING

Acceptance testing is performed to ensure the system meets its defined requirements through functional, performance, and stress tests. In Connectify, this testing confirmed that all core features like messaging, group handling, and file sharing performed as expected under normal and high-load conditions. Both developer-defined and user-focused tests were used. Stress tests were also carried out to check the system's limits, such as handling multiple simultaneous users or large file transmissions.

VALIDATION TESTING

Validation testing is done to ensure complete assembly of the error-free software. Validation can be termed successful only if it functions in manner, reasonably expected by the user under validation is alpha and beta testing. In Connectify, this included both alpha and beta testing phases. We verified whether inputs passed to various services—such as login, chat, file uploads, and group joins—were correctly handled. Invalid inputs were intentionally entered to test the system's response, and all errors were properly caught and displayed through validations.

Here we have checked whether the data passed to each service is valid or not. For that we have entered incorrect values and did the validation testing and checked whether the errors are being considered and incorrect values are discarded. The errors were rectified. In this System, verifications are done correctly. So, there is no chance for users to enter incorrect values. It will give error messages by using different validations. The validation testing is done very clearly and found it is error free.

OUTPUT TESTING

After successful validation testing, output testing was performed to ensure that the system produces correct and expected results. In Connectify, outputs such as message displays, friend request statuses, group participant lists, and file previews were checked. Users were consulted about the preferred formats, and testing was done for both on-screen outputs and downloadable/print-friendly formats. The system generated outputs accurately and in a user-friendly manner.

6.2 TEST PLAN ACTIVITIES

Testing is the process of running a program to identify and fix errors. A good test has a high likelihood of uncovering undiscovered issues. Testing for **Connectify** was carried out in multiple stages, starting from individual components and progressing to the full system integration. Each page, functionality, and module—such as user authentication, friend requests, and chat features—was tested independently. The overall system integrity was also thoroughly checked. Some of testing strategies applied for the system are listed here.

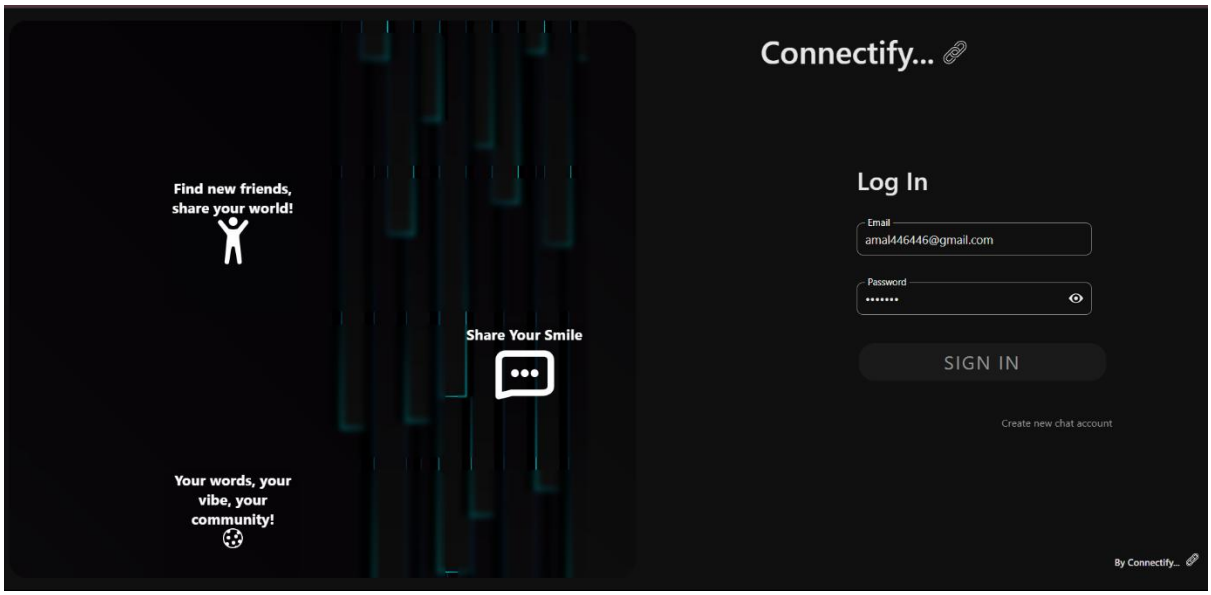
A test plan should contain the following:

- Test unit specification
- Features to be tested
- Approach for testing
- Test deliverables
- Testing schedule

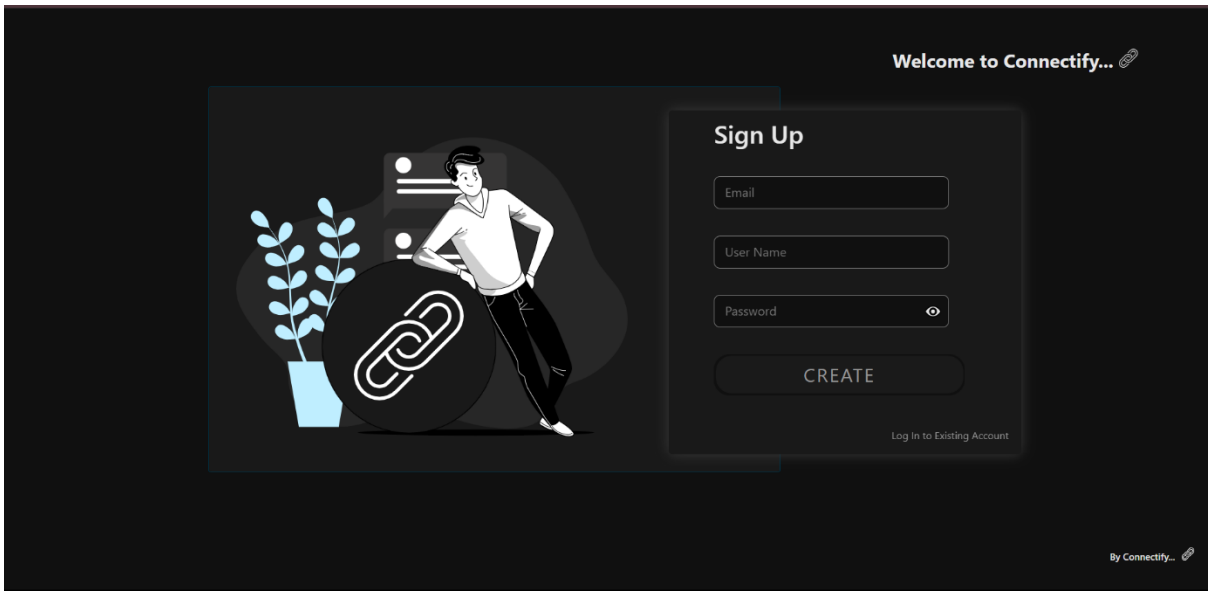
These strategies ensured that all aspects of the system were tested before final acceptance.

6.3 SCREEN LAYOUTS

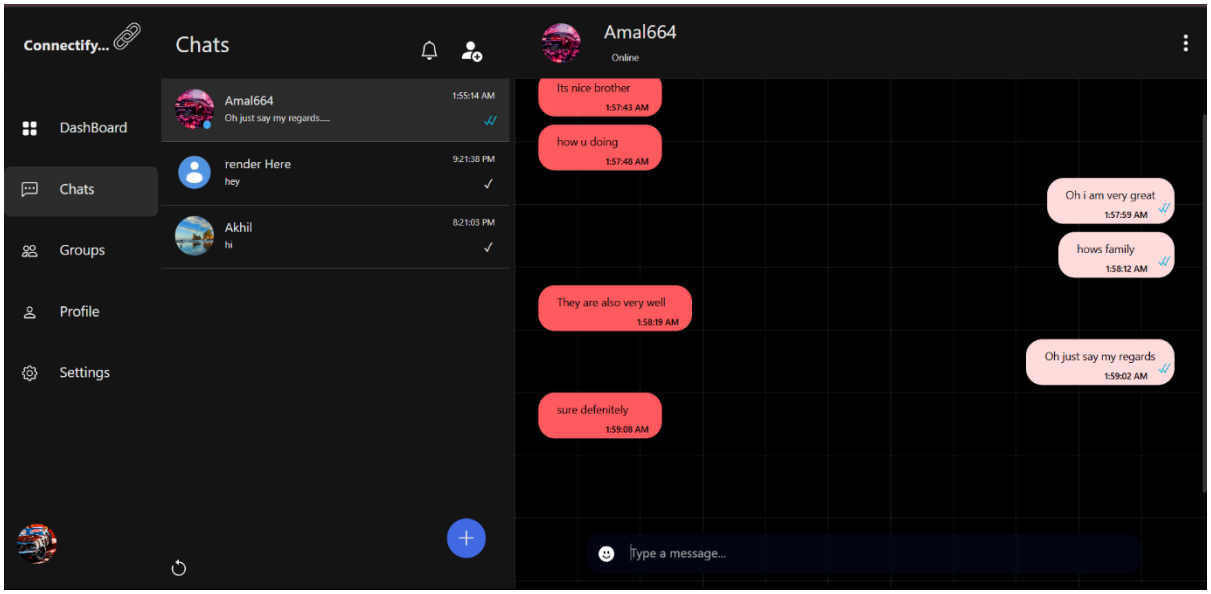
LOGIN PAGE



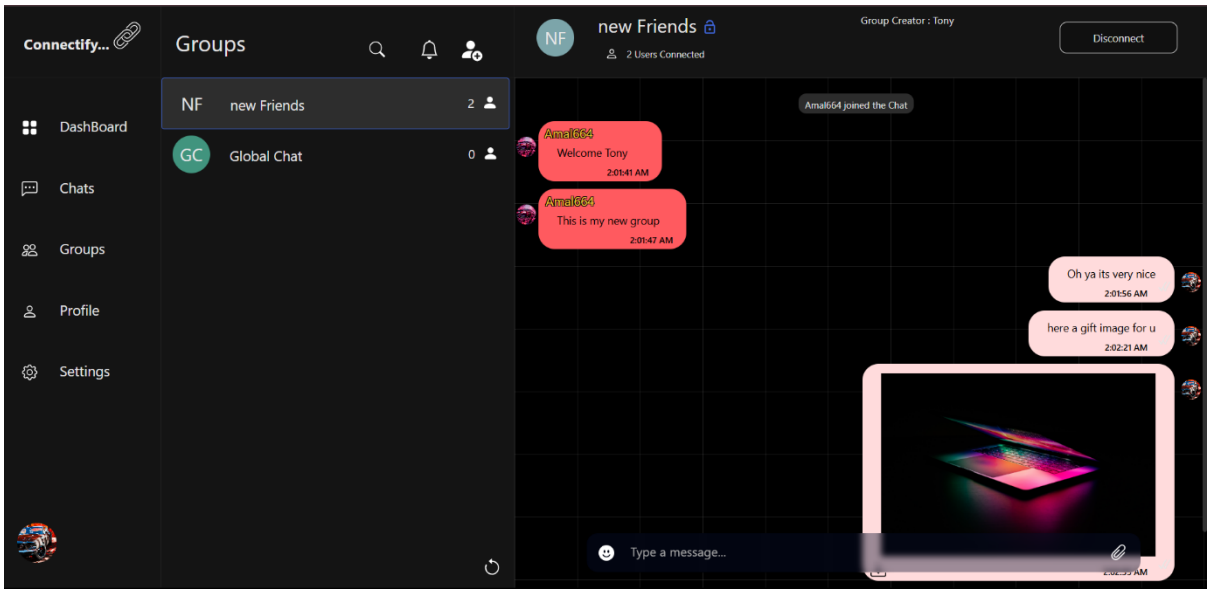
SIGN UP PAGE



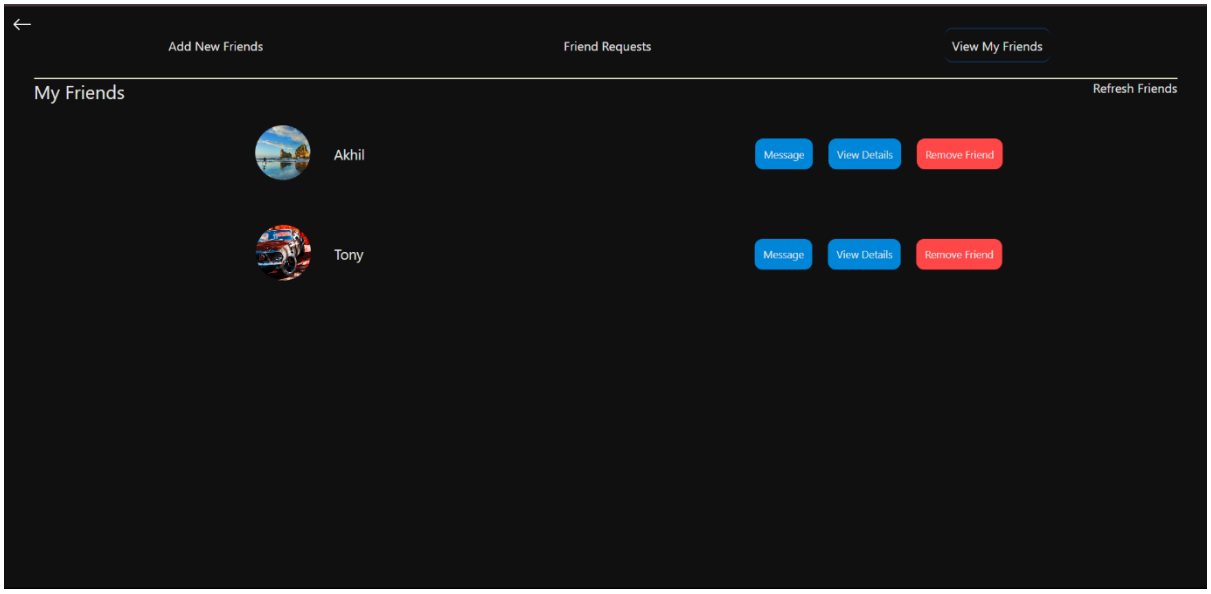
HOME PAGE – CHAT



HOME PAGE – GROUPS

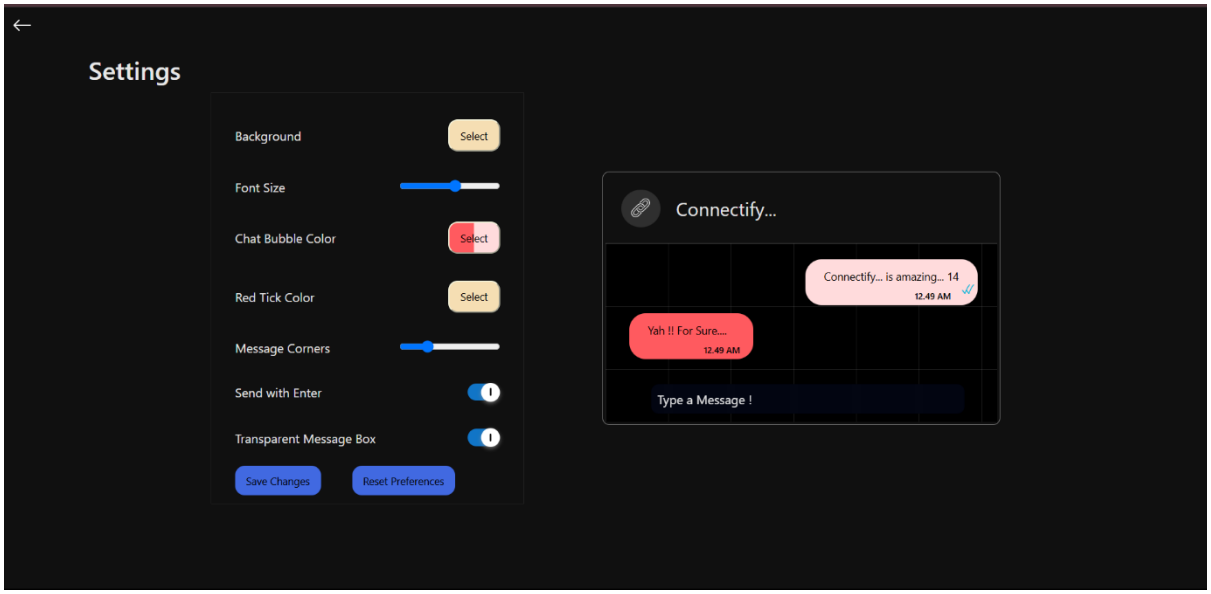


FRIENDS PAGE



PROFILE PAGE

SETTINGS PAGE



7. SYSTEM IMPLEMENTAION

Implementation is the process of converting a new or a revised system design into an operational one.

Conversion means changing from one system to another.

The objective is to put the system into operation while holding costs, risks and personnel irritation to minimum. It involves:

- Creating necessary files and databases compatible with the system.
- Training staff and users on the new system functionalities.
- Installing any required hardware or terminals.

Implementation involves converting from an old system to a new one. This can mean replacing a manual or automated system, or making significant modifications to an existing one. For **Connectify**, proper implementation is critical to ensure a reliable system that meets user and organizational requirements. While successful implementation doesn't always guarantee improvement, improper installation can hinder progress. The implementation method and timeline were planned in advance. Following this, the system underwent thorough testing, and users were trained on new procedures to ensure smooth adoption of the updated system.

Implementation procedures: Implementation of software involves installing the system in its real environment to meet the needs of users and ensure smooth operation. During the initial stages, users may have doubts about the software, but it's essential to address their concerns and build confidence in the system. Proper training and user support are key to preventing resistance and ensuring users are comfortable using the software.

Post implementation: Post-implementation is the final step of the system approach, where the success of the solution is reviewed after deployment. Even after successful implementation, the solution may fail to fully address the problem. The post-implementation review process ensures that the results are monitored, evaluated, and any issues are addressed to improve the system's effectiveness.

8. CONCLUSION AND SCOPE FOR FUTURE ENHANCEMENT

CONCLUSION

Connectify provides a comprehensive and efficient solution for users to manage their communication needs, offering seamless features like friend requests, real-time messaging, file sharing, group management, and more. By focusing on simplicity, real-time interactions, and user privacy, the system delivers an intuitive and engaging user experience. The integration of robust features like custom UI preferences and advanced message indicators ensures that users have full control over their communication environment.

Looking ahead, there is significant potential for expanding the capabilities of **Connectify**. Future enhancements could include the addition of end-to-end encryption for heightened security, more advanced group management features, and improved AI-driven recommendations for content and contacts. Additionally, incorporating mobile support for both iOS and Android could significantly increase accessibility, allowing users to stay connected on-the-go. Enhanced data analytics for real-time insights and improving scalability to handle larger user bases are also areas for potential growth.

As the needs of the user community evolve, **Connectify** is well-positioned to adapt and offer an even more personalized, efficient, and secure communication platform, ensuring sustained success and user satisfaction in the ever-evolving landscape of digital communication.

FUTURE ENHANCEMENTS

Connectify has been built with a solid foundation and offers numerous possibilities for future improvements, including:

1. **Mobile App Support:** Building dedicated Android and iOS apps for better on-the-go communication.
2. **End-to-End Encryption:** Enhancing message privacy and user trust with advanced encryption.
3. **Voice and Video Calls:** Adding real-time calling features within private chats and groups.
4. **AI-Powered Suggestions:** Suggesting contacts, replies, and message categories using smart AI.

5. **Advanced Admin Tools:** Offering more control and visibility to group admins while keeping user privacy intact.
6. **Improved File Previewing:** Enabling more advanced in-app preview of shared images, videos, and documents.
7. **Group Polls and Announcements:** Supporting interactive group tools like polls or pinned announcements.
8. **Status Feature:** Letting users post temporary updates similar to WhatsApp's Status or Instagram Stories.
9. **Multi-Device Login:** Allowing users to stay connected across multiple devices with sync.
10. **Localization and Language Support:** Providing multilingual interface options for broader accessibility.

These enhancements would help **Connectify** grow into an even more robust and modern chat platform, aligned with user expectations and technological trends.

BIBLIOGRAPHY

REFERENCES

- Django for Beginners: Build Websites with Python and Django
- Django 1.5 documentation
- Database management system – Raghu Ramakrishan and Johannes Gehrke, McGraw hill, 2003

WEBSITES

- https://docs.djangoproject.com/en/5.2/
- https://www.w3schools.com/
- https://docs.mongoengine.org/
- https://channels.readthedocs.io/en/latest/
- https://pymongo.readthedocs.io/en/stable/
- https://react.dev/learn