BOLOBUZZ

A PROJECT REPORT
for
Mini Project (KCA353)
Session (2024-25)

Submitted by

Shivam Pal
(2300290140168)
Sandeep Sahu
(2300290140158)
Shivam Pandey
(2300290140169)
Syed Ayaz Hasan
(2300290140190)

Submitted in partial fulfilment of the
Requirements for the Degree of

MASTER OF COMPUTER APPLICATION

Under the Supervision of
Ms. Annu Yadav
(Assistant Professor)

Submitted to

DEPARTMENT OF COMPUTER APPLICATIONS
KIET Group of Institutions, Ghaziabad
Uttar Pradesh-201206
(DECEMBER 2024)

CERTIFICATE

Certified that **Shivam Pal 2300290140168, Sandeep Sahu 2300290140158, Shivam Pandey 2300290140169, Syed Ayaz Hasan 2300290140190** have carried out the project work having "**BoloBuzz**" (**Mini-Project-KCA353**) for **Master of Computer Application** from Dr. A.P.J. Abdul Kalam Technical University (AKTU) (formerly UPTU), Lucknow under my supervision. The project report embodies original work, and studies are carried out by the student himself/herself and the contents of the project report do not form the basis for the award of any other degree to the candidate or to anybody else from this or any other University/Institution.

Ms. Annu Yadav	**Dr. Arun Tripathi**
Assistant Professor	**Head**
Department of Computer Applications	**Department of Computer Applications**
KIET Group of Institutions, Ghaziabad	**KIET Group of Institutions, Ghaziabad**

BOLOBUZZ
Shivam Pal
Sandeep Sahu
Shivam Pandey
Syed Ayaz Hasan

ABSTRACT

Effective communication and seamless collaboration are fundamental to the success of any organization. The rapid evolution of technology and the increasing complexity of work environments necessitate robust platforms that enable real-time messaging, efficient collaboration, and streamlined user management. In response to this growing demand, the BoloBuzz project was developed as a cutting-edge solution designed to simplify communication and enhance productivity in professional and organizational settings.

BoloBuzz stands out as an all-in-one platform built with modern technologies, including **Next.js**, **Mantine**, and **TanStack Query**, ensuring a smooth and responsive user experience. The platform's architecture focuses on creating an intuitive and dynamic interface that caters to both end-users and administrators. Key features include user account creation, the ability to join and interact in various workspaces, and real-time messaging capabilities. For administrators, BoloBuzz provides a secure backend interface designed for efficient management of user activities and platform oversight.

The research methodology involved extensive analysis of existing communication platforms to identify gaps in usability, scalability, and security. User experience design principles were employed to ensure an interface that balances simplicity with functionality. Real-time communication was implemented using WebSocket technology to achieve instantaneous interactions, while a modular architecture ensures that the system remains scalable and maintainable. Emphasis was placed on secure user authentication and role-based access control to uphold privacy and ensure a secure environment for all users.

The findings from the development and implementation of BoloBuzz underscore its potential to transform the way teams and organizations collaborate. By addressing critical pain points, such as fragmented communication, inefficient workflows, and lack of scalable management tools, BoloBuzz bridges the gap between complex organizational needs and user-friendly solutions. Additionally, the integration of advanced front-end frameworks and state

management techniques ensures a responsive and robust platform that adapts seamlessly to the evolving demands of its users.

The applications of BoloBuzz extend beyond traditional corporate settings. The platform can be adapted for educational institutions, non-profit organizations, and other collaborative ecosystems where efficient communication and management are essential. By fostering a sense of connectivity and simplifying administrative processes, BoloBuzz has the potential to enhance teamwork and improve organizational efficiency across diverse industries.

Future research and development directions for BoloBuzz include the incorporation of artificial intelligence for predictive analytics, enhanced personalization, and automated task management. Integrating machine learning models could enable the platform to provide intelligent insights, such as detecting communication bottlenecks or suggesting workflow optimizations. Furthermore, exploring decentralized technologies, such as blockchain, could enhance data security and transparency, making the platform even more resilient and trustworthy.

In conclusion, BoloBuzz represents a significant step forward in addressing the challenges of modern communication and collaboration. Its focus on user-centric design, combined with powerful backend capabilities, ensures that it remains a scalable, reliable, and efficient solution for organizations. With continued innovation and development, BoloBuzz is poised to play a pivotal role in reshaping the way teams interact and achieve their objectives in a dynamic and fast-paced world.

ACKNOWLEDGEMENTS

Success in life is never attained single-handedly. My deepest gratitude goes to my project supervisor, **Ms. Annu Yadav** for her guidance, help, and encouragement throughout my project work. Their enlightening ideas, comments, and suggestions.
Words are not enough to express my gratitude to Dr. Arun Kumar Tripathi, Professor and Head, Department of Computer Applications, for his insightful comments and administrative help on various occasions.
Fortunately, I have many understanding friends, who have helped me a lot on many critical conditions.
Finally, my sincere thanks go to my family members and all those who have directly and indirectly provided me with moral support and other kind of help. Without their support, completion of this work would not have been possible in time. They keep my life filled with enjoyment and happiness.

Shivam Pal

Sandeep Sahu

Shivam Pandey

Syed Ayaz Hasan

TABLE OF CONTENTS

Content	Page No.
Certificate	ii
Abstract	iii-iv
Acknowledgements	v
Table of Contents	v-vi
List of Figures	vii
CHAPTER 1 : Introduction	**1-3**
1.1 Overview	1
1.2 Problem Statement	1
1.3 Objective	1
1.4 Scope	2
1.5 Features	2
1.6 Background	3
CHAPTER 2 : Feasibility Study	**4-7**
2.1 Economic Feasibility	4
2.2 Technical Feasibility	5
2.3 Operational Feasibility	5
2.4 Behavioral feasibility	6
CHAPTER 3 : Software requirement specification	**8-12**
3.1 Functionalities	8

3.2 User and Characteristics 9

3.3 Features of project 10

3.4 Feature of Admin 11

3.4 Features of User 12

CHAPTER 4 : System Requirement 13 -22

4.1 Functional Requirement 15

4.2 Non-Functional Requirement 16

4.3 Design Goal 17

4.4 System Sequence Diagram 18

4.5 ER Diagram for BoloBuzz 22

CHAPTER 5 : System Design 23-25

5.1 Primary Design Phase 22

5.2 Secondary Design Phase 23

5.3 User Interface 23

CHAPTER 6 : Architecture 26-30

6.1 Layered Architecture 26

6.2 Real Time Update 28

6.3 Security Architecture 30

CHAPTER 7 : Project Screenshots 31-36

CHAPTER 8 : Code Screenshots 37-55

CHAPTER 9 : Conclusion 56-57

CHAPTER 9 : References 58

LIST OF FIGURES

Figure No.	**Name of Figure**	**Page No.**
4.1	Use Case	17
4.2	E-R Diagram	18
4.3	Data flow	19
4.4	Class Diagram	20
4.5	Level 0 DFD	21
4.6	Level 1 DFD	22
7.1	Login Page	31
7.2	Email Authentication	32
7.3	Home Page	**33**

7.4	Create Channels	34
7.5	Workspace	35
7.6	Account Info	36

CHAPTER 1

INTRODUCTION

1.1 Overview

BoloBuzz is a modern communication platform crafted to enhance team collaboration with its easy-to-use messaging system, smooth file-sharing capabilities, and reliable user management features. The platform is built using cutting-edge technologies like Next.js for the frontend, Mantine for intuitive UI components, and TanStack Query for handling API calls efficiently. Designed with both team members and administrators in mind, BoloBuzz offers a seamless and user-friendly experience to meet the demands of dynamic workplaces.

1.2 Problem Statement

Traditional communication tools often fail to meet the demands of modern team collaboration. Platforms like email or SMS lack real-time messaging, seamless file-sharing capabilities, and efficient workspace management. This creates a need for a unified platform where teams can communicate, collaborate, and track progress in real time.

1.3 Objectives

- Develop a platform that facilitates real-time communication among team members.
- Implement user authentication and workspace management features.
- Create an admin dashboard for user management and activity monitoring.
- Design a responsive and user-friendly interface using Mantine and Tailwind CSS.
- Integrate backend functionality using Next.js and Node.js for optimal performance.

-

1.4 Scope

This project aims to provide essential tools for communication and collaboration, forming a strong foundation for efficient teamwork. Future developments could include features like video conferencing, integration with third-party tools, and scalability improvements to support larger organizations.

1.5 Feature

- **Real-Time Messaging**: Instant messaging for seamless team communication.
- **User Authentication**: Secure user sign-up, login, and profile management.
- **Admin Controls**: Tools for managing users, workspaces, and monitoring activities.
- **User Management**: Capability for users to join multiple workspaces and interact in dedicated channels.

1.6 Hardware and Software Requirements

1.6.1 Hardware Requirements:

- 4 GB RAM or more
- 1 GHz processor or higher
- 10 GB of free storage space

1.6.2 Software Requirements:

- **Operating System**: Windows 10 or higher, Linux, or macOS
- **Frontend**: Next.js, Mantine, Tailwind CSS
- **Backend**: Node.js, TanStack Query
- **Database**: MongoDB or PostgreSQL
- **IDE**: Visual Studio Code

1.7 Background

In today’s fast-moving work environment, effective communication and collaboration are essential for organizational success. Traditional communication methods like emails and phone calls often fail to address the real-time demands of modern teams. BoloBuzz was envisioned to bridge this gap by offering a real-time messaging platform that streamlines team interactions. By leveraging advanced technologies such as Next.js, Mantine, and TanStack Query, BoloBuzz delivers an intuitive user experience and a reliable backend system tailored to meet the needs of both team members and administrators.

CHAPTER 2

FEASIBILITY STUDY

Every project is feasible when unlimited resources and time are available. However, a feasibility study helps assess how practical and efficient a proposed solution is under real-world constraints. This involves analyzing whether the solution can meet user requirements while remaining flexible for future enhancements. Typically, feasibility studies focus on four key aspects: economic, technical, operational, and behavioral feasibility.

2.1 ECONOMICAL FEASIBILITY

Economic feasibility examines whether the project is financially viable, considering costs for development, maintenance, and operation, along with potential revenue or benefits if the platform is commercialized. Below is a detailed analysis for BoloBuzz:

2.1.1 Development Costs

Technology Stack :

- **Next.js:** As a free, open-source framework, it eliminates expensive licensing fees. Its features, like server-side rendering and static site generation, reduce development time and cost.
- **Mantine:** This React component library accelerates development with pre-designed UI elements, minimizing the need for custom design work and reducing overall costs.
- **TanStack Query:** Simplifies data fetching and caching, lowering the complexity and time required to develop a robust backend system.

Operational Expenses:

- **Cloud Hosting and Storage:** Reliable services like AWS, Google Cloud, or Azure offer scalable hosting options, with costs increasing proportionally to platform growth.
- **Database Costs:** Using managed databases like MongoDB or PostgreSQL ensures efficient data storage at affordable initial rates, though costs may rise with larger datasets.
- **Maintenance and Updates:** Regular updates for bug fixes, security enhancements, and new features will incur ongoing costs, but using open-source tools helps minimize long-term expenses.

2.2 TECHINICAL FEASIBILITY

Technology Stack:

- Next.js ensures high performance with server-side rendering and modular architecture.
- Mantine offers sleek and responsive UI components.
- TanStack Query optimizes data fetching, enhancing performance and user experience.

Real-Time Messaging: The integration of WebSockets or similar protocols supports instant communication without lags.

User Management: Features like authentication, role-based access, and workspace management are implemented using secure and scalable practices.

Scalability: The platform's architecture is designed to handle increased traffic, allowing it to grow with user demand.

2.3 OPERATIONAL FEASIBILITY

Operational feasibility focuses on the platform's usability and its ability to meet organizational requirements:

- **Ease of Use:** Intuitive and user-friendly, designed to require minimal training for new users.

- **Admin Interface:** A robust backend interface simplifies user and workspace management while offering analytics for monitoring activity.

- **Team Collaboration:** Real-time messaging, workspace creation, and file-sharing capabilities make the platform a practical solution for modern teams.

2.4 BEHAVIORAL FEASIBILITY

Behavioral feasibility evaluates how likely stakeholders are to adopt and effectively use the platform.

2.4.1 User Acceptance

- **Ease of Use:** The intuitive design ensures even non-technical users can navigate the platform with ease.
- **Customization:** Options for personalizing workspaces and notifications enhance user satisfaction.

2.4.2 Organizational Readiness

- **Goal Alignment:** BoloBuzz directly supports organizational objectives like improving communication and collaboration.
- **Cultural Fit:** It integrates seamlessly with teams already familiar with platforms like Slack or Microsoft Teams.

2.4.3 Stakeholder Willingness

- **End Users:** Team members will appreciate features like instant messaging and workspace organization, especially in remote or hybrid setups.

- **Administrators:** The backend tools simplify operations, aligning with organizational policies.

- **Decision-Makers:** The cost-effective nature and potential for productivity improvements make the platform appealing to leadership.

CHAPTER 3

SOFTWARE REQUIREMENT SPECIFICATION

The SRS defines the requirements and capabilities of the BoloBuzz platform, offering a clear roadmap for development and ensuring alignment with stakeholders' expectations. Designed to meet the needs of all user types- end-users, admins, and super admins. BoloBuzz aims to enhance real-time collaboration within organizational environments.

3.1 Functionalities:

The core functionalities of the BoloBuzz platform include:

1. User Authentication

- Users can register with an email and password.
- Login and logout functionalities with proper session management.
- Password recovery options for enhanced accessibility.

2. Workspace Creation and Management

- Users can create, join, or leave workspaces.
- Each workspace is assigned a unique identifier to ensure access is restricted to authorized users.

3. Real-Time Messaging

- Instant messaging for both one-on-one and group conversations.
- Ensures seamless and timely communication within workspaces.

4. **User Role Management**

- Admins can assign roles like admin or member to workspace users, ensuring proper access control.

5. **Platform Analytics**

- Admins can view usage statistics, such as active users and messages sent, for better decision-making.

6. **Notifications**

- Real-time alerts for new messages, workspace invitations, or role changes.

7. **Search Functionality**

- Allows users to search for specific messages, files, or members within a workspace.

3.2 User and Characteristics:

1. **End Users (Team Members)**

- *Characteristics:* Employees or collaborators using the platform for communication.
- *Requirements:* A simple, intuitive interface to join workspaces, send messages, and search content.
- *Expected Behavior:* Regular use for team collaboration with basic familiarity with chat tools.

2. **Administrators**

- *Characteristics:* Team leaders or IT personnel managing workspaces and user activities.
- *Requirements:* Advanced tools for user management, analytics, and role controls.
- *Expected Behavior:* Proficient in managing digital tools, ensuring smooth platform operation.

3. **Super Admins (Platform Owners)**

- *Characteristics:* Individuals or teams overseeing the platform's overall performance.

- *Requirements:* Backend-level access to monitor operations, troubleshoot issues, and implement upgrades.
- *Expected Behavior:* Ensures infrastructure reliability and compliance with data security standards.

3.3 Features of the project:

1. **Collaborative Workspaces**

   Dedicated spaces for team or project-specific communication.

2. **Real-Time Communication**

- Instant messaging with negligible lag to facilitate seamless collaboration.

3. **User-Friendly Design**

- Responsive interface optimized for devices like mobiles, tablets, and desktops.

4. **Secure Data Handling**

- Data encryption and secure user authentication methods, such as hashed passwords.

5. **Notifications**

- Alerts for key activities like new messages or role updates.

6. **Modular Architecture**

- Flexible design to allow future enhancements like file sharing or voice/video capabilities.

3.4 Features of Admin:

1. **User Management**

- Add, remove, or ban users.
- Assign and revoke roles such as admin or member.

2. **Analytics Dashboard**

- Monitor usage metrics, including active users and message statistics.

3. Workspace Control

- Create, manage, and moderate workspaces.
- Address reported content or inappropriate messages.

4. Custom Settings

- Configure workspace-specific settings like notification preferences or message retention policies

.

5. Activity Logs

- Maintain logs of user actions for auditing or troubleshooting purposes.

3.5 Features of User:

1. Account Management

- Register, log in, and manage personal profiles.

2. Workspace Participation

- Join or switch between multiple workspaces using invitations or workspace IDs.

3. Real-Time Messaging

- Send and receive instant messages in group or private chats.

4. Search Functionality

- Quickly find messages, files, or members within a workspace.

5. Notifications

- Stay updated on new messages and workspace activities.

6. User Profile Customization

- Set profile pictures, display names, and status updates.

7. Privacy Controls

- Manage visibility settings, such as online status within workspaces.

CHAPTER 4

SYSTEM REQUIREMENT

The development and deployment of BoloBuzz require clear and precise system specifications to ensure smooth operation and optimal performance. These requirements are grouped into hardware and software needs for both the development and production stages.

4.1 System Requirements

4.1.1 Hardware Requirements

1. **Development Environment**

- **Processor**: Intel Core i5 or equivalent
- **RAM**: At least 8 GB
- **Storage**: Minimum of 50 GB free disk space
- **Network**: Reliable high-speed internet connection

2. **Production Environment**

- **Server**: Cloud-hosted or dedicated server with a minimum 4-core CPU
- **RAM**: 16 GB or more
- **Storage**: At least 100 GB SSD for application and data
- **Network**: Stable high-speed internet connection with a static IP address

4.1.2 Software Requirements

1. **Development Environment**

- **Operating System**: Windows 10/11, macOS, or Linux
- **IDE/Code Editor**: Visual Studio Code or similar

- **Node.js**: Version 16 or higher
- **Database**: PostgreSQL or any other compatible relational database
- **Browser**: Google Chrome, Firefox, or similar for testing

2. **Production Environment**

- **Operating System**: Linux-based server (e.g., Ubuntu 20.04)
- **Web Server**: Nginx or Apache
- **Node.js**: Version 16 or above
- **Database**: PostgreSQL with adequate storage
- **Security**: SSL certificate for secure data transmission

4.1.3 Dependencies and Frameworks

- **Frontend**:
  - Next.js: React-based framework for frontend development
  - Mantine: Library for UI components and styling
- **Backend**:
  - TanStack Query: For efficient data fetching and state management
  - Additional libraries for authentication, data validation, and error handling

4.1.4 Optional Tools

- **Docker**: For streamlined deployment and containerization
- **Git**: For version control and collaborative development
- **Postman**: For API testing and validation

4.2 Non-Functional Requirements

To ensure a robust and user-friendly platform, BoloBuzz incorporates non-functional requirements addressing performance, scalability, security, and more.

1. **Performance**

- Real-time messaging must operate with minimal delay, even under high user loads.
- The system should handle up to 1,000 concurrent users per workspace without performance issues.

- API calls should respond within one second during normal usage.

2. **Scalability**

- The platform should scale seamlessly to accommodate increasing user numbers and workspaces.
- Backend architecture should support horizontal scaling for high-traffic scenarios.

3. **Security**
   - All data, including messages and files, must be encrypted during transmission (HTTPS) and storage.
   - Multi-factor authentication (MFA) should be available for added security.
   - Regular security audits should identify and address vulnerabilities.

4. **Availability**

- The system should maintain a 99.9% uptime to ensure continuous service.
- A failover mechanism should be in place for server downtime or maintenance periods.

5. **Usability**

- The platform must be intuitive and user-friendly, catering to both technical and non-technical users.
- Key features like notifications, file sharing, and user management should be easily accessible.

6. **Maintainability**

- The platform should follow a modular design to allow for easy updates and maintenance.
- Comprehensive documentation for code, APIs, and deployment should be readily available.

7. **Compatibility**

- The platform must work smoothly with modern browsers, such as Chrome, Firefox, Edge, and Safari.
- A responsive design should ensure usability across various screen sizes and devices.

8. **Reliability**

- Accurate message delivery should be guaranteed, with retry mechanisms for failures.
- Automated error detection and logging should enable efficient issue resolution.

9. **Compliance**

- The platform must adhere to relevant data protection regulations, such as GDPR, to protect user privacy.

4.3 Design Goal

The design goal of BoloBuzz is to create a secure, scalable, and user-friendly communication platform tailored for modern organizational needs. The primary focus is on delivering seamless real-time messaging, ensuring high performance, and maintaining data security and privacy.

Key objectives include:

- Developing an intuitive interface for easy navigation by all user types.
- Building a robust backend capable of handling large-scale usage.
- Ensuring the platform’s architecture is flexible to allow future enhancements.

Use Case Diagram

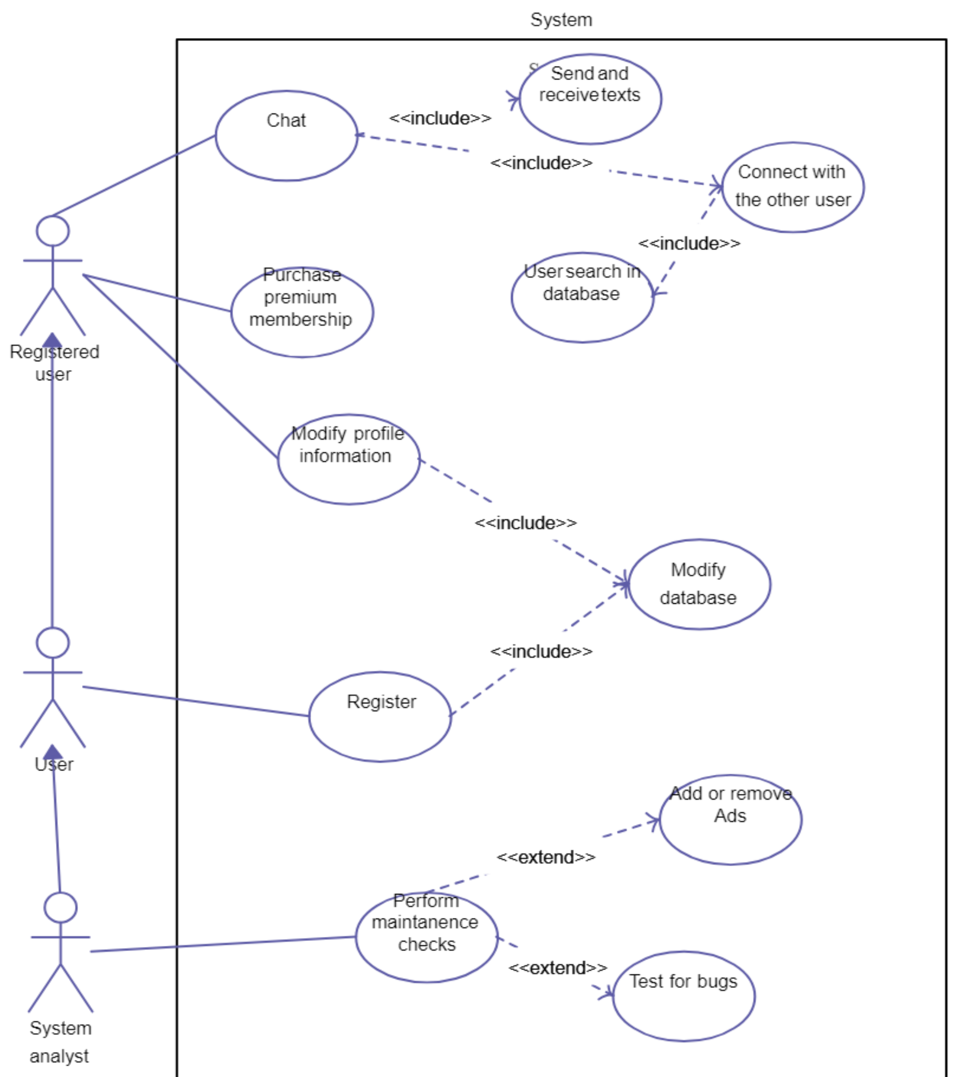


Fig 4.1: Use Case Diagram

ER-Diagram

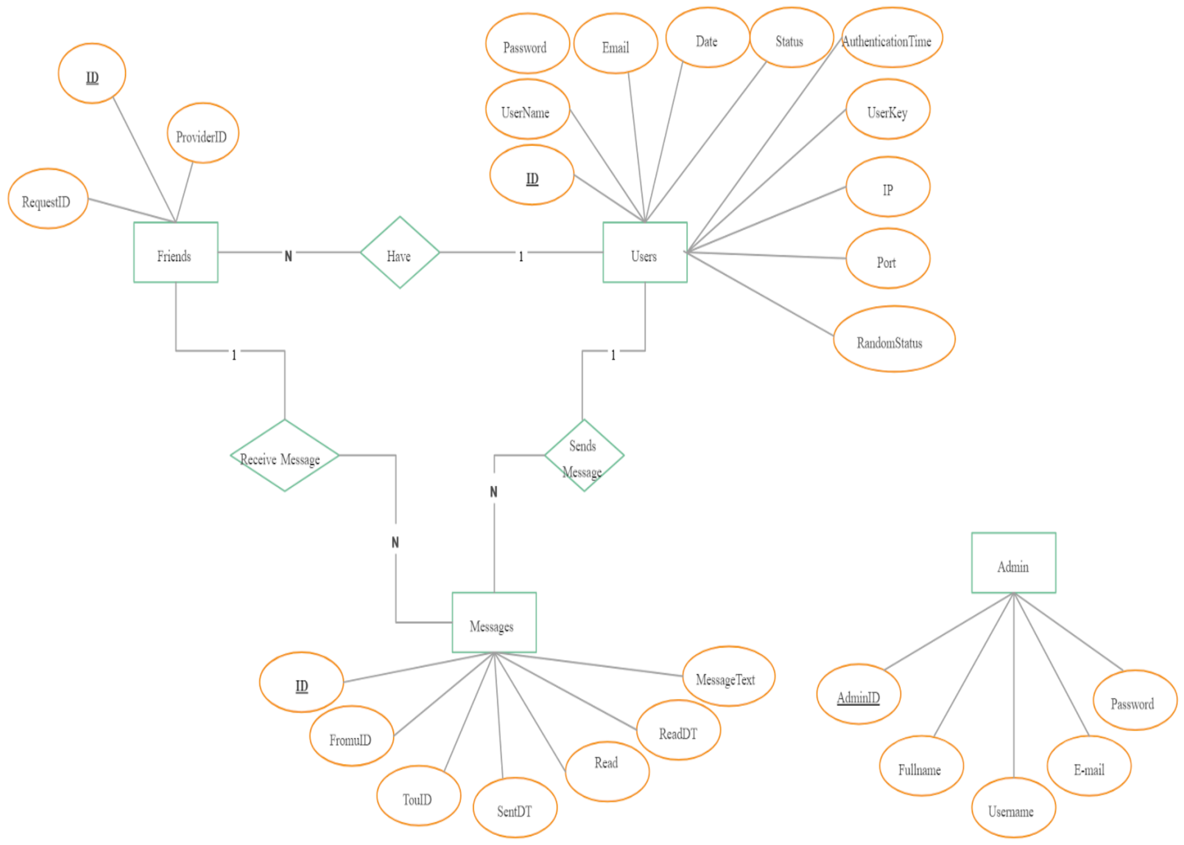


Fig 4.2: ER- Diagram

Data Flow Diagram

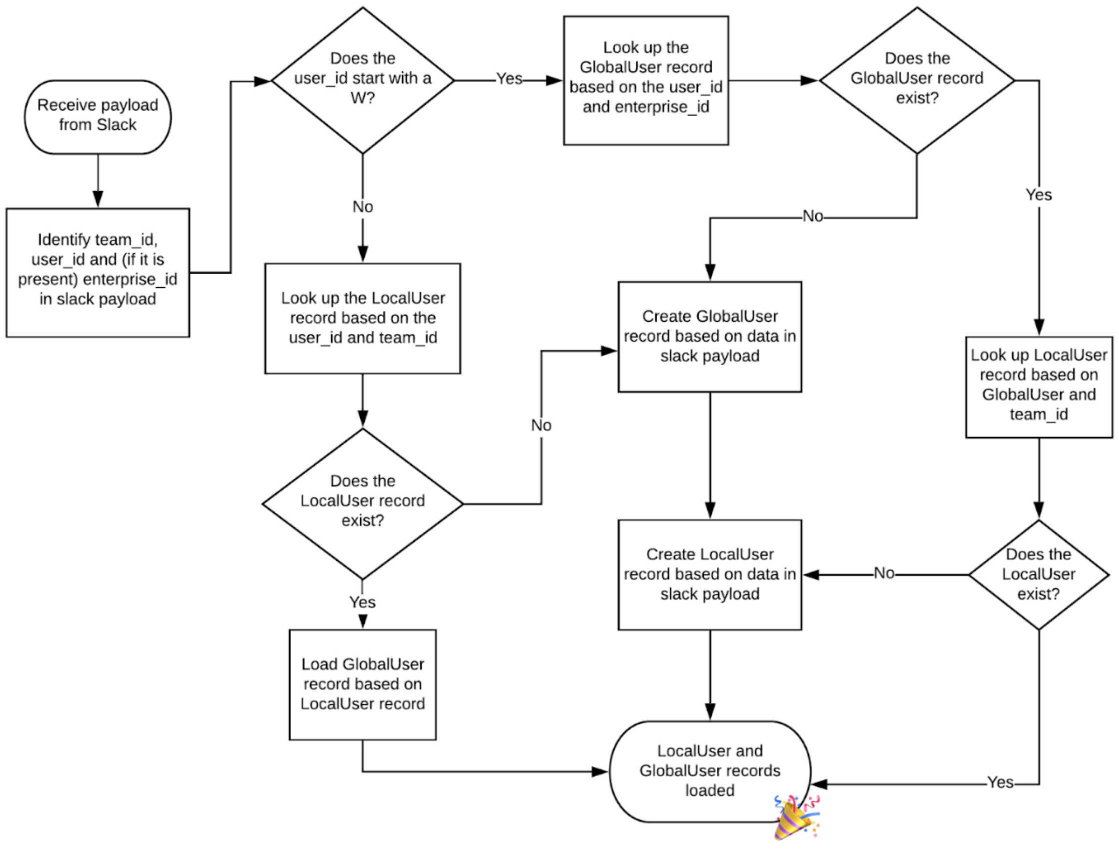

Receive payload from Slack
Identify team_id, user_id and (if it is present) enterprise_id in slack payload
Does the user_id start with a W?
Yes
Look up the GlobalUser record based on the user_id and enterprise_id
Does the GlobalUser record exist?
Yes
No
No
Look up the LocalUser record based on the user_id and team_id
Create GlobalUser record based on data in slack payload
Look up LocalUser record based on GlobalUser and team_id
No
Does the LocalUser record exist?
Create LocalUser record based on data in slack payload
No
Does the LocalUser exist?
Yes
Load GlobalUser record based on LocalUser record
LocalUser and GlobalUser records loaded
Yes

Fig 4.3: Data Flow Diagram

Class Diagram

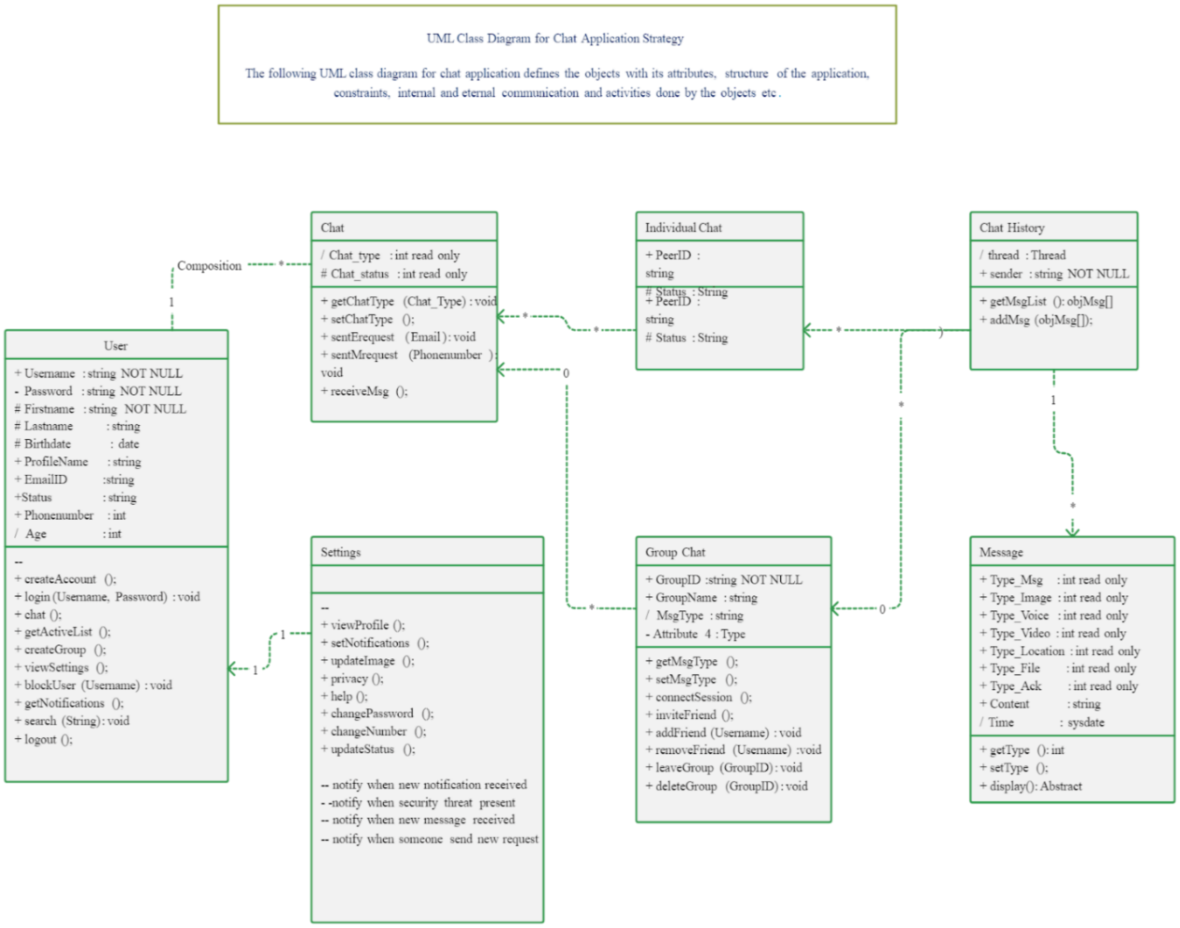

UML Class Diagram for Chat Application Strategy
The following UML class diagram for chat application defines the objects with its attributes, structure of the application, constraints, internal and eternal communication and activities done by the objects etc.
Chat
/ Chat_type : int read only
# Chat_status : int read only
+ getChatType (Chat_Type) : void
+ setChatType ();
+ sentErequest (Email) : void
+ sentMrequest (Phonenumber) : void
+ receiveMsg ();
Individual Chat
+ PeerID : string
# Status : String
+ PeerID : string
# Status : String
Chat History
/ thread : Thread
+ sender : string NOT NULL
+ getMsgList (): objMsg[]
+ addMsg (objMsg[]);
User
+ Username : string NOT NULL
- Password : string NOT NULL
# Firstname : string NOT NULL
# Lastname : string
# Birthdate : date
+ ProfileName : string
+ EmailID :string
+Status : string
+ Phonenumber : int
/ Age : int
--
+ createAccount ();
+ login (Username, Password) : void
+ chat ();
+ getActiveList ();
+ createGroup ();
+ viewSettings ();
+ blockUser (Username) : void
+ getNotifications ();
+ search (String) : void
+ logout ();
Settings
--
+ viewProfile ();
+ setNotifications ();
+ updateImage ();
+ privacy ();
+ help ();
+ changePassword ();
+ changeNumber ();
+ updateStatus ();
-- notify when new notification received
- -notify when security threat present
-- notify when new message received
-- notify when someone send new request
Group Chat
+ GroupID :string NOT NULL
+ GroupName : string
/ MsgType : string
- Attribute 4 : Type
+ getMsgType ();
+ setMsgType ();
+ connectSession ();
+ inviteFriend ();
+ addFriend (Username) : void
+ removeFriend (Username) :void
+ leaveGroup (GroupID) : void
+ deleteGroup (GroupID) : void
Message
+ Type_Msg : int read only
+ Type_Image : int read only
+ Type_Voice : int read only
+ Type_Video : int read only
+ Type_Location : int read only
+ Type_File : int read only
+ Type_Ack : int read only
+ Content : string
/ Time : sysdate
+ getType (): int
+ setType ();
+ display(): Abstract
Composition
*
1
1
1
*
*
0
*
*
0
*
1
*

Fig 4.4 Class Diagram

Level 0 DFD Diagram

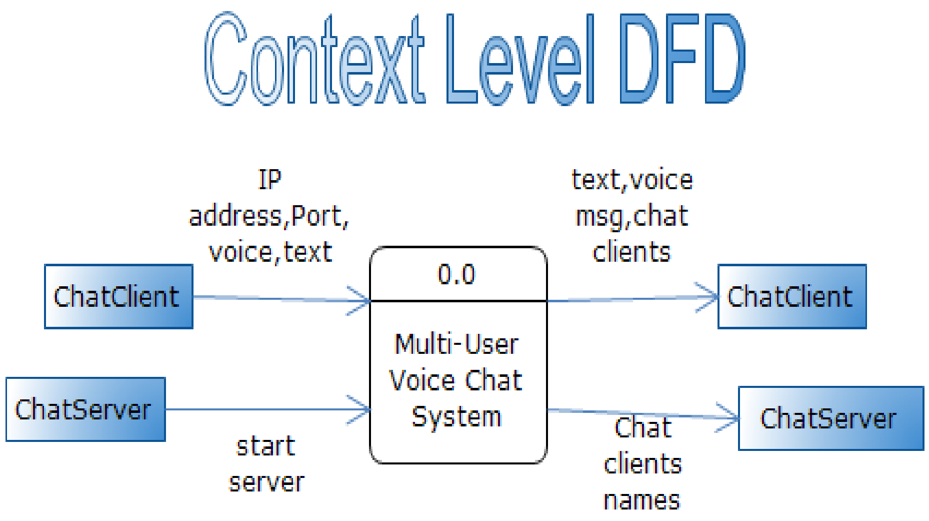


Fig 4.5 Level 0 DFD Diagram

Level 1 DFD Diagram

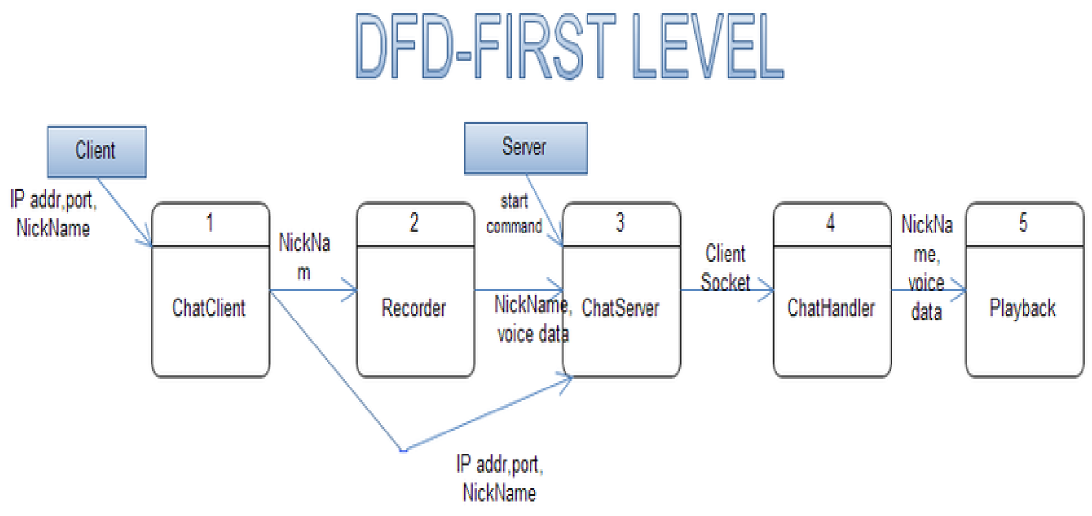


Fig 4.6 Level 1 DFD Diagram

CHAPTER 5

SYSTEM DESIGN

The system design of the BoloBuzz platform emphasizes creating a modular, scalable, and secure architecture. This approach supports seamless real-time communication, effective user management, and a user-friendly experience. The design incorporates modern technologies to ensure the platform is efficient and reliable.

5.1 Architecture Overview

BoloBuzz adopts a client-server architecture where the frontend communicates with the backend via APIs. This approach ensures a clear separation of responsibilities, enhances scalability, and simplifies maintenance.

5.2 Frontend Design

- **Technology**: The frontend is built using Next.js, leveraging server-side rendering (SSR) and static site generation (SSG) to enhance performance and search engine optimization (SEO).
- **UI Framework**: The Mantine library is utilized to create visually appealing, responsive, and modern user interfaces.
- **Key Features**: The frontend supports user authentication, messaging, file sharing, and workspace management, ensuring a smooth and intuitive experience for users.

5.3 Backend Design

- **Technology**: The backend is developed using Node.js and integrates TanStack Query for efficient data fetching and state management.
- **Database**: A relational database, such as PostgreSQL, is used to securely store user data, messages, and workspace details.
- **APIs**: RESTful APIs enable secure and efficient communication between the frontend and backend.

5.4 Real-Time Communication

- **WebSocket Integration**: WebSockets enable real-time messaging, ensuring messages are delivered instantaneously.
- **Event Management**: The backend uses an event-driven design to handle real-time notifications and updates efficiently.

5.5 Admin Interface

- **Features**: The admin panel provides tools for managing users, monitoring workspaces, and analyzing platform usage.
- **Security**: Role-based access control (RBAC) restricts admin functionalities to authorized users, enhancing platform security.

5.6 Scalability and Reliability

- **Load Balancing**: Traffic is distributed evenly across servers using a load balancer, maintaining consistent performance even during peak usage.
- **Database Optimization**: Techniques like indexing and query optimization ensure the platform handles large volumes of data efficiently.

5.7 Security Measures

- **Data Encryption**: Data is encrypted using HTTPS for secure transmission and AES for secure storage, safeguarding user privacy.
- **Authentication**: OAuth 2.0 ensures secure user authentication while enabling integration with third-party platforms.

5.8 Deployment and Hosting

- **Containerization**: Docker containers streamline deployment, making it easier to manage and scale the application.
- **Hosting**: The platform is hosted on a reliable cloud provider like AWS or Azure, ensuring high availability and robust infrastructure support.

CHAPTER 6

ARCHITECTURE

The architecture of the BoloBuzz platform is based on a layered approach, ensuring a clear separation of responsibilities, modularity, scalability, and ease of maintenance. This design divides the system into distinct layers, each focusing on specific functions to streamline development and simplify future enhancements.

6.1 Presentation Layer (Frontend)

This layer is the user-facing component, responsible for managing all interactions between the users and the system. It provides a responsive and intuitive interface to ensure accessibility across different devices.

- **Technologies Used**:
  - **Next.js**: Supports server-side rendering and static site generation to improve performance and SEO.
  - **Mantine**: Offers pre-designed and customizable UI components for building an appealing and user-friendly interface.

- **Responsibilities**:

- Displays workspaces, chats, user profiles, and notifications to users.
- Captures user inputs like login credentials, messages, and workspace details, and forwards them to the backend via APIs.
- Manages client-side state efficiently for real-time responsiveness using TanStack Query.
- Delivers real-time updates such as incoming messages or notifications using WebSocket technology.

6.2 Application Layer (Backend Logic)

This layer acts as the bridge between the user interface and the database, processing user requests and applying business logic.

- **Technologies Used**:
  - **Node.js**: Serves as the backend framework for handling requests and implementing logic.
  - **WebSocket Integration**: Enables instant real-time communication between users

.

- **Responsibilities**:
  - Handles core operations like user authentication, workspace creation, and message routing.
  - Enforces business rules, such as access permissions for workspaces or user roles.
  - Secures data handling during frontend-backend interactions.
  - Manages WebSocket connections to ensure uninterrupted real-time communication.

6.3 Data Access Layer (Database Management)

This layer focuses on efficient and secure data storage and retrieval, serving as an abstraction over the database.

- **Technologies Used**:
  - **PostgreSQL**: Stores user data, messages, and workspace details.

- **Prisma/Sequelize**: Simplifies database interactions with structured query-building tools.

- **Responsibilities**:
  - Stores sensitive user information securely, including hashed passwords and activity logs.
  - Manages workspace details and chat history with structured indexing for quick retrieval.
  - Ensures data consistency and integrity with constraints like unique workspace IDs.

6.4 Communication Layer

This layer facilitates real-time communication and ensures smooth data exchange between system components.

- **Technologies Used**:
  - **WebSocket Protocol**: Manages live messaging and instant updates.
  - **REST APIs**: Handles standard operations like authentication, workspace management, and non-real-time interactions.

- **Responsibilities**:
  - Manages WebSocket connections for instant communication.
  - Provides RESTful APIs for performing CRUD (Create, Read, Update, Delete) operations.
  - Ensures fast and reliable data transfer with minimal latency.

6.5 Security Layer

Although not a traditional layer, security is a critical component integrated across all system levels.

- **Responsibilities**:
  - Implements authentication and authorization using token-based systems like JWT.
  - Encrypts sensitive data during transmission with HTTPS and ensures password security using hashing techniques.

- Protects against common vulnerabilities, including SQL injection, cross-site scripting (XSS), and cross-site request forgery (CSRF).

Advantages of a Layered Architecture in BoloBuzz

1. **Separation of Concerns**: Each layer focuses on specific tasks, simplifying development, testing, and maintenance.
2. **Scalability**: Layers can be scaled independently, such as enhancing the database for more users or optimizing the frontend for better performance.
3. **Modularity**: Individual layers can be updated or replaced without affecting the rest of the system.
4. **Reusability**: Components within each layer can be reused across different areas of the application or in other projects.
5. **Enhanced Security**: Multiple layers provide opportunities to integrate robust security measures, ensuring comprehensive protection.

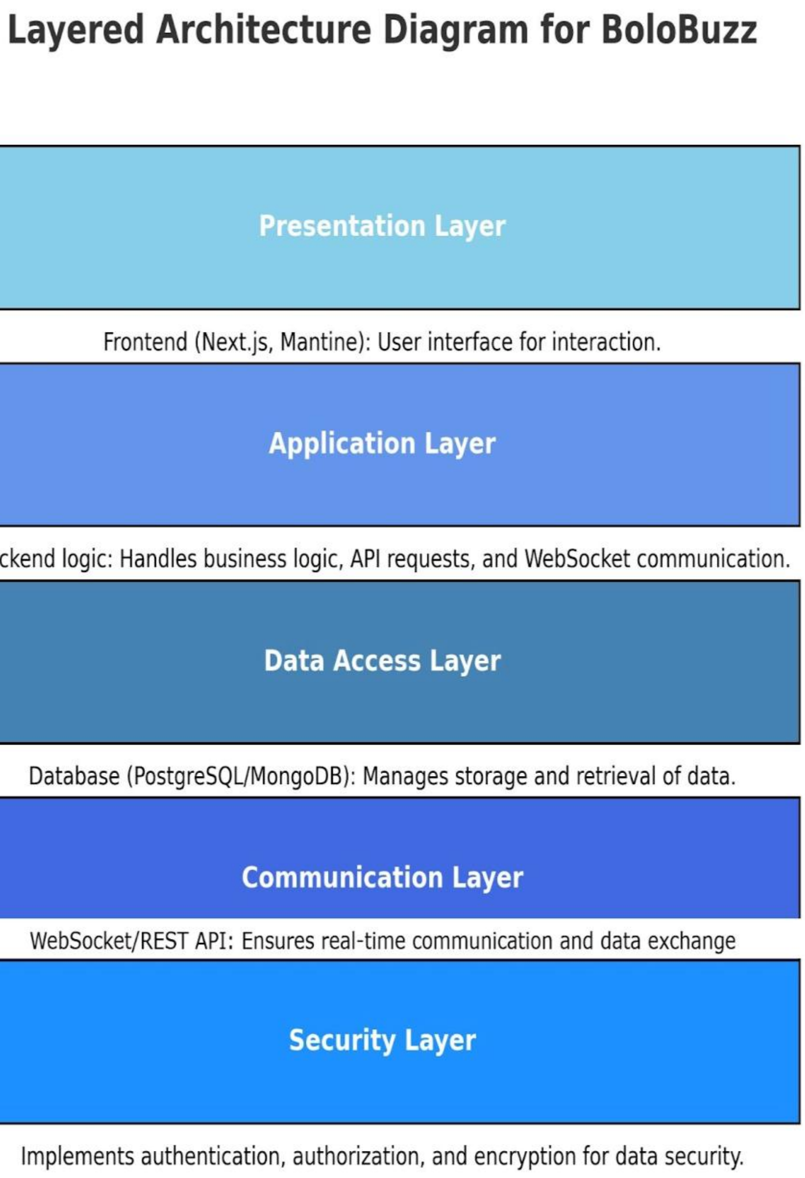

Layered Architecture Diagram for BoloBuzz
Presentation Layer
Frontend (Next.js, Mantine): User interface for interaction.
Application Layer
Backend logic: Handles business logic, API requests, and WebSocket communication.
Data Access Layer
Database (PostgreSQL/MongoDB): Manages storage and retrieval of data.
Communication Layer
WebSocket/REST API: Ensures real-time communication and data exchange
Security Layer
Implements authentication, authorization, and encryption for data security.

Fig 6.1: Architecture of BoloBuzz web app

Chapter 7

Project Screenshots

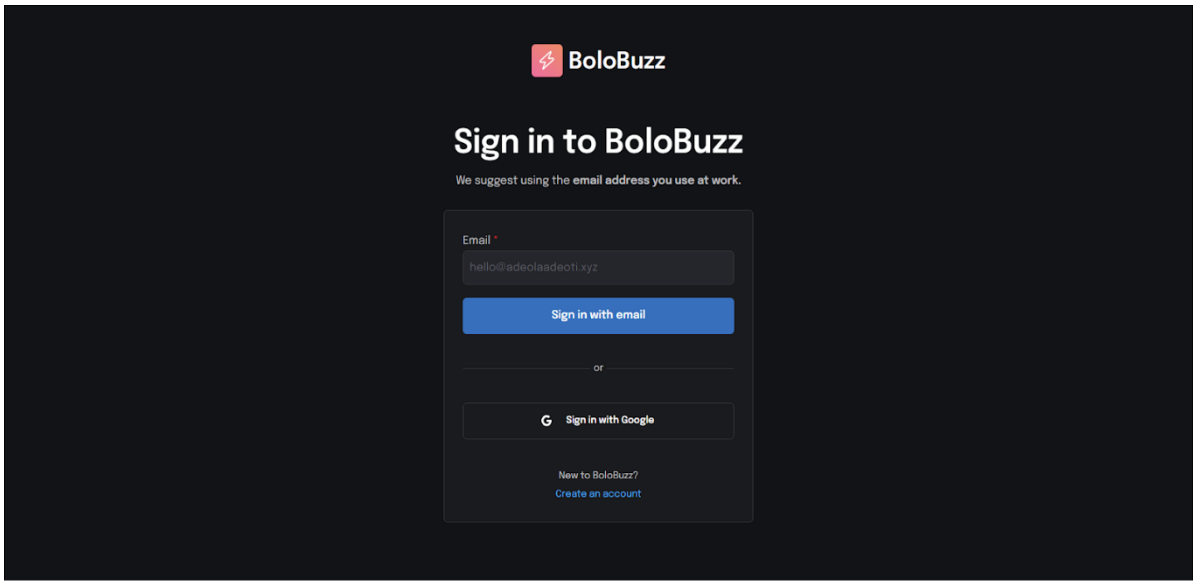
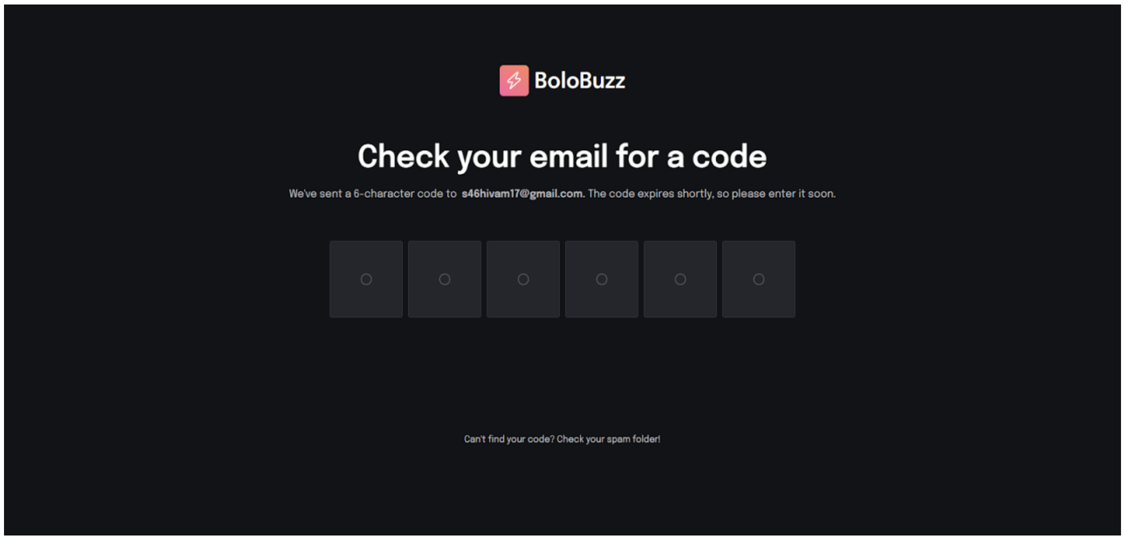


Fig 7.1 Login Page

The sign-in module allows users to access BoloBuzz conveniently through various methods while ensuring data security.

Options Available:

Email & Password: Traditional method with secure encryption



BoloBuzz

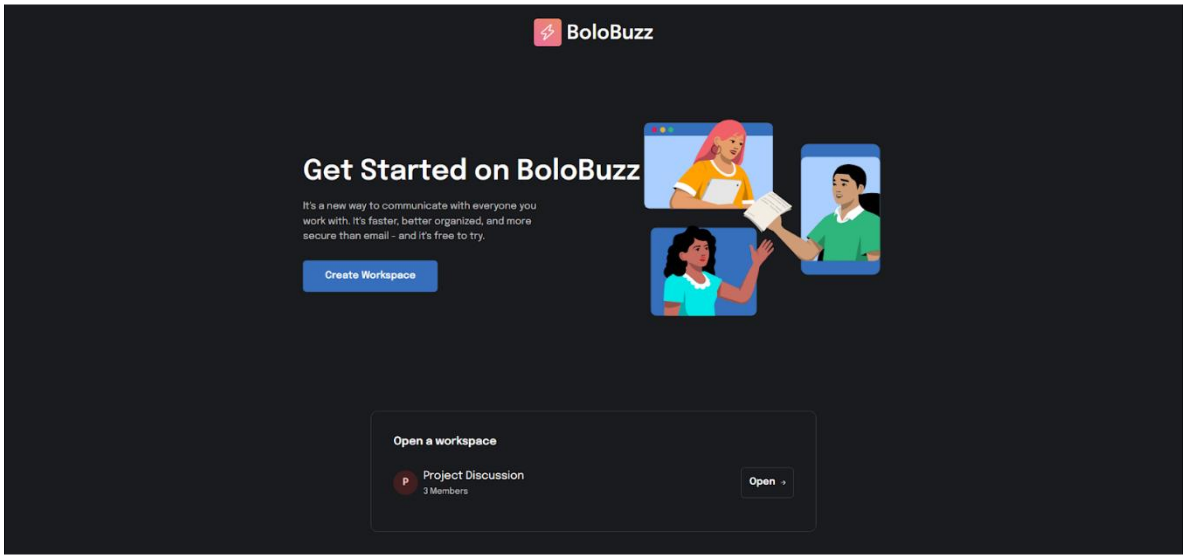
Check your email for a code

We've sent a 6-character code to **s46hivam17@gmail.com.** The code expires shortly, so please enter it soon.

Can't find your code? Check your spam folder!

Fig 7.2 Email Authentication Page

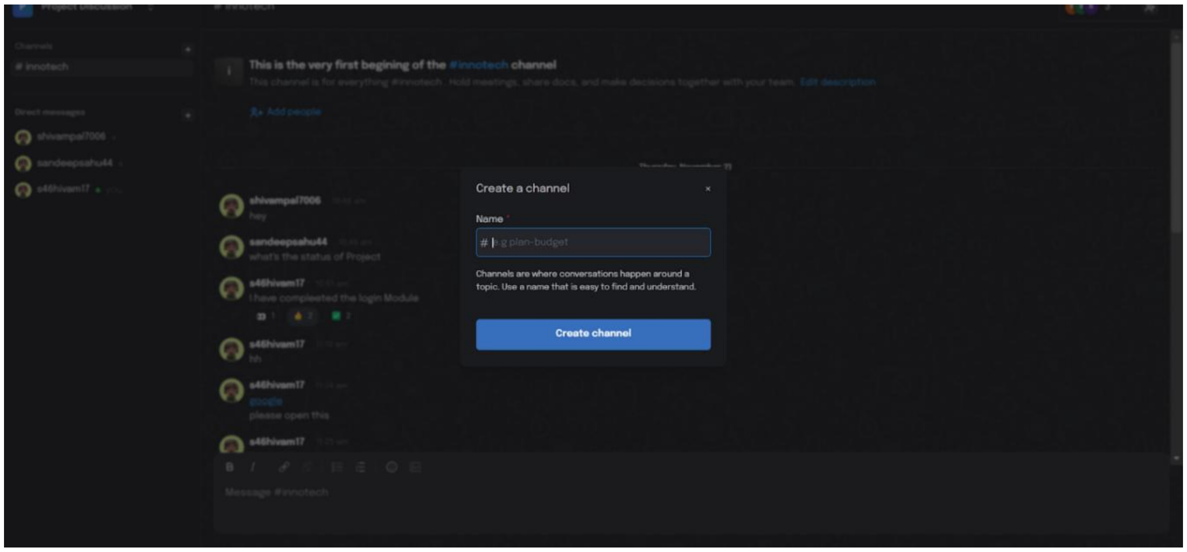
Ensure the user’s email is valid and secure by sending a One-Time Password through OTP for verification.



It's a new way to communicate with everyone you work with. It's faster, better organized, and more secure than email - and it's free to try.

Fig 7.3 Home Page

View and manage all your workspaces in one place, with the option to create a new workspace effortlessly



Create a channel

Name *

e.g plan-budget

Channels are where conversations happen around a topic. Use a name that is easy to find and understand.

This is the very first begining of the #innotech channel

shivampal7006: hey

sandeepsahu44: what's the status of Project

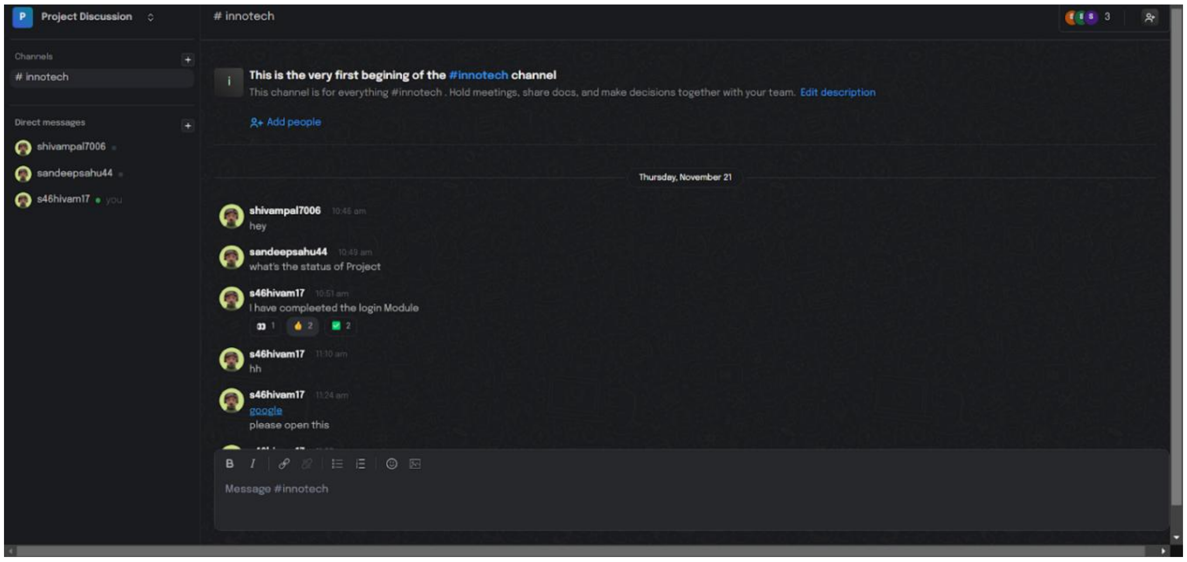
s46hivam17: I have compleeted the login Module

s46hivam17: hh

s46hivam17: google
please open this

Fig 7.4 Create Channel Functionality

Easily create a channel to start meaningful conversations and engage with your audience on BoloBuzz.



innotech

This is the very first begining of the #innotech channel
This channel is for everything #innotech . Hold meetings, share docs, and make decisions together with your team. Edit description

Thursday, November 21

shivampal7006 10:46 am
hey

sandeepsahu44 10:49 am
what's the status of Project

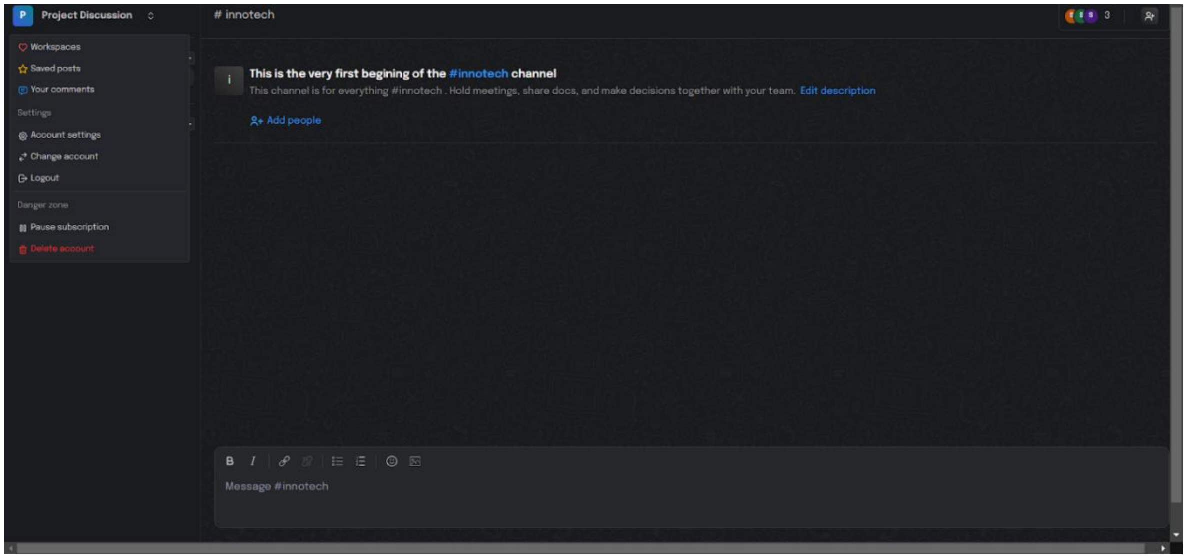
s46hivam17 10:51 am
I have compleeted the login Module

s46hivam17 11:10 am
hh

s46hivam17 11:24 am
google
please open this

Fig 7.5 Workspace Page

Engage in real-time conversations, share ideas, and interact with others through organized channels on BoloBuzz.



This is the very first begining of the #innotech channel

This channel is for everything #innotech . Hold meetings, share docs, and make decisions together with your team. Edit description

Fig 7.6 Account Info Menu

View and update your personal details, account settings, and preferences to customize your BoloBuzz experience.

CHAPTER 8

Code Screenshots

```
import mongoose from 'mongoose'

import jwt from 'jsonwebtoken'

import crypto from 'crypto'

import randomize from 'randomatic'


export interface UserSchemaType {

  _id: mongoose.Types.ObjectId

  username?: string

  email?: string

  role?: string

  phone?: string

  profilePicture?: string

  isOnline?: boolean

  loginVerificationCode?: string

  loginVerificationCodeExpires?: Date

  googleId?: string

  getSignedJwtToken?: () => string
  getVerificationCode?: () => string
}

const userSchema = new mongoose.Schema<UserSchemaType>(
  {
    username: {
      type: String,
      default() {
        return this.email.split('@')[0]
      },
    },
    email: {
      type: String,
      required: [true, 'Please enter your email'],
      unique: true,
      // match: [/^\w+@[a-zA-Z_]+?\.[a-zA-Z]{2,3}$/, 'Please enter a valid email'],
    },
    googleId: String,
    isOnline: Boolean,
    role: String,
    phone: String,
    profilePicture: String,

    loginVerificationCode: String,

    loginVerificationCodeExpires: Date,

  },

  {

    timestamps: true,

    versionKey: false,

  }

)


// Sign JWT and return

userSchema.methods.getSignedJwtToken = function () {

  return jwt.sign({ id: this._id }, process.env.JWT_SECRET, {

    expiresIn: process.env.JWT_EXPIRE,

  })

}


// Generate login verification code

userSchema.methods.getVerificationCode = function () {

  const verificationCode = randomize('Aa0', 6)
  this.loginVerificationCode = crypto
    .createHash('sha256')
    .update(verificationCode)
    .digest('hex')

  this.loginVerificationCodeExpires = Date.now() + 10 * 60 * 1000 // 10 minutes

  return verificationCode
}

export default mongoose.model('User', userSchema)
```

This code snippet define a mongoose schema for user model

```
import mongoose from 'mongoose'

interface MessageSchemaType {
  sender: mongoose.Schema.Types.ObjectId
  content: string
  channel: mongoose.Schema.Types.ObjectId
  organisation: mongoose.Schema.Types.ObjectId
  conversation: mongoose.Schema.Types.ObjectId
  collaborators: mongoose.Schema.Types.ObjectId[]
  reactions: {
    emoji: string
    reactedToBy: [mongoose.Schema.Types.ObjectId]
  }[]
  threadReplies: mongoose.Schema.Types.ObjectId[]
  threadRepliesCount: number
  threadLastReplyDate: Date
  isBookmarked: boolean
  isSelf: boolean
  hasRead: boolean
  type: string
  createdAt: Date
  updatedAt: Date
}

const messageSchema = new mongoose.Schema<MessageSchemaType>(
  {
    sender: {
      type: mongoose.Schema.Types.ObjectId,
      ref: 'User',
    },
    content: String,
    channel: {
      type: mongoose.Schema.Types.ObjectId,
      ref: 'Channel',
    },
    organisation: {
      type: mongoose.Schema.Types.ObjectId,
      ref: 'Organisation',
    },
    conversation: {
      type: mongoose.Schema.Types.ObjectId,
      ref: 'Conversation',
    },
    collaborators: [
      {
        type: mongoose.Schema.Types.ObjectId,
        ref: 'User',
      },
    ],
    reactions: [
      {
        emoji: String,
        reactedToBy: [
          {
            type: mongoose.Schema.Types.ObjectId,
            ref: 'User',
          },
        ],
      },
    ],
    threadReplies: [
      {
        type: mongoose.Schema.Types.ObjectId,
          ref: 'User',
        },
      ],
      threadRepliesCount: Number,
      threadLastReplyDate: Date,
      isBookmarked: {
        type: Boolean,
        default: false,
      },
      isSelf: {
        type: Boolean,
        default: false,
      },
      hasRead: {
        type: Boolean,
        default: false,
      },
      type: String,
    },
    {
      timestamps: true,
      versionKey: false,
    }
  )

  export default mongoose.model('Message', messageSchema)
```

This code snippet defines a mongoose schema for message model

```
import { Request, Response, NextFunction } from 'express'
import User from '../models/user'
import sendEmail from '../helpers/sendEmail'
import crypto from 'crypto'
import { verificationHtml } from '../html/confirmation-code-email'
import passport from 'passport'
import { Strategy as GoogleStrategy } from 'passport-google-oauth20'

// Configure Google OAuth strategy
passport.use(
  new GoogleStrategy(
    {
      clientID: process.env.GOOGLE_CLIENT_ID,
      clientSecret: process.env.GOOGLE_CLIENT_SECRET,
      callbackURL: `${process.env.API_URL}/auth/google/callback`,
      scope: ['profile', 'email'],
    },
    async (accessToken, refreshToken, profile, done) => {
      try {
        // Find or create a user based on the Google profile information
        let user = await User.findOne({ googleId: profile.id })
        if (!user) {
          user = new User({
            googleId: profile.id,
            username: profile.displayName,
            email: profile.emails[0].value,
          })
          await user.save()
        }

        return done(null, user)
      } catch (error) {
        return done(error, false)
      }
    }
  )
)

passport.serializeUser((user, done) => {
  done(null, user)
})

passport.deserializeUser((user, done) => {
  done(null, user)
})

// @desc    Register user
// @route   POST /api/v1/auth/register
// @access  Public
```

```
    },




export const register = async (
  req: Request,
  res: Response,
  next: NextFunction
) => {
  const { username, email } = req.body

  if (!email) {
    return res.status(400).json({
      success: false,
      data: {
        name: 'Please provide your email address',
      },
    })
  }

  const emailExist = await User.findOne({ email })

  if (emailExist) {
    return res.status(400).json({
      success: false,
      data: {
        name: 'User already exists',
      },
    })
```

```
}

const user = await User.create({
  username,
  email,
})
try {
  const verificationToken = user.getVerificationCode()
  await user.save()
  sendEmail(
    email,
    'BoloBuzz confirmation code',
    verificationHtml(verificationToken)
  )

  res.status(201).json({
    success: true,
    data: {
      name: 'Verification token sent to email',
    },
  })
} catch (err) {
  user.loginVerificationCode = undefined
  user.loginVerificationCodeExpires = undefined
  await user.save({ validateBeforeSave: false })
```

```
    next(err)
  }
}

// @desc    Signin user
// @route   POST /api/v1/auth/signin
// @access  Public
export const signin = async (
  req: Request,
  res: Response,
  next: NextFunction
) => {
  const { email } = req.body

  if (!email) {
    return res.status(400).json({
      success: false,
      data: {
        name: 'Please provide your email address',
      },
    })
  }

  const user = await User.findOne({ email })
```

,

```
if (!user) {
  return res.status(400).json({
    success: false,
    data: {
      name: "User doesn't exist",
    },
  })
}

try {
  const verificationToken = user.getVerificationCode()
  await user.save()

  sendEmail(
    email,
    'Slack confirmation code',
    verificationHtml(verificationToken)
  )

  res.status(201).json({
    success: true,
    data: {
      name: 'Verification token sent to email',
    },
  })
  } catch (err) {
    user.loginVerificationCode = undefined
    user.loginVerificationCodeExpires = undefined
    await user.save({ validateBeforeSave: false })
    next(err)
  }
}

// @desc    Verify user
// @route   POST /api/v1/auth/verify
// @access  Public
export const verify = async (
  req: Request,
  res: Response,
  next: NextFunction
) => {
  try {
    if (!req.body.loginVerificationCode) {
      return res.status(400).json({
        success: false,
        data: {
          name: 'Please provide verification token',
        },
      })
    }
```

```
// Get hashed token
const loginVerificationCode = crypto
  .createHash('sha256')
  .update(req.body.loginVerificationCode)
  .digest('hex')

const user = await User.findOne({
  loginVerificationCode,
  loginVerificationCodeExpires: { $gt: Date.now() },
})

if (!user) {
  return res.status(400).json({
    success: false,
    data: {
      name: 'Invalid verification token',
    },
  })
}

res.status(200).json({
  success: true,
  data: {
    username: user.username,
    email: user.email,
        token: user.getSignedJwtToken(),
      },
    })

    user.loginVerificationCode = undefined
    user.loginVerificationCodeExpires = undefined
    await user.save({ validateBeforeSave: false })
  } catch (err) {
    next(err)
  }
}

// @desc    Google OAuth Callback
// @route   GET /auth/google/callback
// @access  Public
export const googleCallback = async (
  req: Request,
  res: Response,
  next: NextFunction
) => {
  passport.authenticate('google', (err, user) => {
    if (err) {
      // Handle error
      return next(err)
    }
    if (!user) {
      // Handle user not found
      return res.status(401).json({ message: 'Authentication failed' })
    }

    const token = user.getSignedJwtToken()

    res.redirect(
`${process.env.CLIENT_URL}?token=${token}&email=${user.email}&username=${user.username
}`
    )
  })(req, res, next)
}
```

This code snippet defines authentication controller using google and email and password

CHAPTER 8

Conclusion

With this mini-project, BoloBuzz we show a great success in building a real-time messaging and collaboration platform that is designed to cater to the increasing need for seamless digital communication in the workplace. By using up to date technologies like Next. js, Mantine, and TanStack Query, providing a responsive and user-friendly interface with current industry standards. The project covers communication priorities, such as workspace management, user roles, and secure messaging.

The one feature that sets this project apart is its layered architecture which follows modularity and scalability. The exchange of code between presentation layer and application logic, client and server is ensured seamlessly while the data storage and security protocols work independently with clearly defined APIs between the components. Such a design make development and testing much easier, also sets a decent base for future upgrades and scaling when user requirements change.

From a technical perspective, BoloBuzz showcases an efficient use of web development practices. The frontend delivers a rich user experience, while the backend ensures robust handling of real-time data exchange through WebSockets and API communication. Furthermore, the platform's data layer is designed to ensure integrity, consistency, and security by leveraging reliable database solutions. Security remains a cornerstone of the system, with measures like encrypted communication, secure authentication, and access controls to safeguard user data and prevent vulnerabilities.

The development of BoloBuzz has also served as an enriching learning experience for the team. It provided a practical understanding of full-stack web application development, real-time communication protocols, and user interface design. The project required collaboration, problem-solving, and iterative development, reinforcing essential skills for real-world software engineering.

Moreover, the project has potential for future enhancements, such as integrating file sharing, video conferencing, and advanced analytics. These features could further expand the platform’s utility and appeal to a wider audience. The scalability of the architecture ensures that these additions can be implemented without disrupting the existing functionalities.

In conclusion, BoloBuzz successfully fulfills its objectives of providing a streamlined, intuitive, and secure communication platform for professional environments. It not only serves as a proof of concept for real-time messaging platforms but also underscores the importance of teamwork, planning, and the effective application of modern technologies in software development. This mini-project stands as a testament to the team’s dedication and technical competence, paving the way for future projects of greater complexity and impact.

CHAPTER 9

Reference

To support the development and implementation of the BoloBuzz platform, several technical resources and documentation were utilized. These references provided detailed insights into the technologies and frameworks employed throughout the project.

1. **Next.js Documentation**: Comprehensive guide and resources for understanding and implementing server-side rendering, static site generation, and other core functionalities of Next.js

2. **Mantine Documentation**: Detailed instructions and examples for utilizing Mantine's pre-built UI components, enabling the creation of a responsive and aesthetically pleasing interface.

3. **TanStack Query Documentation**: An in-depth resource for managing server-state and API data fetching with TanStack Query, ensuring efficient and real-time data handling.

4. **Node.js Documentation**: Official documentation covering the backend framework, including its modules, APIs, and event-driven architecture for building scalable server-side applications.

5. **MongoDB Documentation**: A valuable resource for understanding the structure and implementation of MongoDB, focusing on efficient data storage and retrieval practices.