PAYBUDDY

A PROJECT REPORT

Submitted in partial fulfilment of the

requirement for the award of the degree

Of

MASTER OF COMPUTERS APPLICATIONS (MCA)

by

KULDEEP SINGH

23FS20MCA00076

Department of Computer Applications

MANIPAL UNIVERSITY JAIPUR

JAIPUR-303007

RAJASTHAN, INDIA

May 2025

DEPARTMENT OF COMPUTER APPLICATION

MANIPAL UNIVERSITY JAIPUR, JAIPUR – 303007 (RAJASTHAN), INDIA

15-05-2025

CERTIFICATE

This is to certify that the project titled PAYBUDDY is a record of work completed during the period from 23.01.2025 to 23.5.2025 done by KULDEEP SINGH (23FS20MCA00076.) submitted in partial fulfilment of the requirements for the award of the Degree of Master of Computer Applications (MCA) at the Department of Computer Applications, Manipal University Jaipur, for the academic year **2023-2025**

Dr. Monika Vishwakarma

Project Guide, Dept of Computer Applications

Manipal University Jaipur

Dr. Shilpa Sharma

HOD, Dept of Computer Applications

Manipal University Jaipur

JIET MEDICAL COLLEGE AND HOSPITAL
(A Unit of Arun Shanti Education Trust)

JIET Universe: NH-62, Pali Road, Village Mogra, Jodhpur - 342802 (Raj.)
Phone No.: 7014957257
Email: infojmch@jietjodhpur.ac.in | **Web.:** www.jiethospital.com

Date:10-05-2025

CERTIFICATE

This is to certify that the project entitled **"Paybuddy"** was carried out by Kuldeep singh at JIET Medical College & Hospital, Jodhpur under my guidance during 23-01-2025 to 23-5-2025.

Yours truly,
JIET MEDICAL COLLEGE & HOSPITAL

ARUN SHANTI EDUCATION TRUST

[illegible]

Ripudaman pareek
Manager–HR

ACKNOWLEDGMENTS

I express my heartfelt gratitude to all those who have supported and guided me throughout my project, **Paybuddy**.

First, I would like to express my sincere thanks to Dean of Manipal University jaipur, for providing the opportunity to undertake this project.

I am deeply grateful to **Dr. Shilpa Sharma** Head of the Department of Computer Application, for the encouragement and resources extended to during this project.

I wish to acknowledge my project supervisor, **Deependra singh** , for their constant support, insightful feedback, and encouragement throughout the development of this project. Their guidance has been a key factor in the successful completion of my work.

I would also like to thank **JIET ORGANIZATION** , whose inputs and practical insights contributed significantly to my understanding of the field.

My sincere thanks go to my **Department Guide Dr. Monika Vishwakarma**, for providing access to essential facilities and for their technical assistance.

Lastly, I would like to thank all the **faculty members** of the Department of Computer Applications for their valuable academic inputs and continuous support during this project.

This project has been a rewarding learning experience, and I am truly grateful to everyone who helped make it possible.

ABSTRACT

This website is designed to provide a digital payment solution, similar to an online wallet. It allows users to make payments, transfer money, and manage their transactions easily and securely. The main goal is to offer a fast, convenient, and user-friendly platform for handling digital payments. By using this system, users can reduce the need for cash and enjoy a smoother experience when sending or receiving money online.

The project followed a structured development process, beginning with planning the layout and user experience, then progressing to component-based design using **React.js**. Tailwind CSS was used to create a clean, responsive interface, while Postgres //and MongoDB is used to store data.

The website prototype successfully demonstrates how users can interact with wallet and perform certain functions like send and receive payments and much more.

The project was built using **MongoDB, Express, React, and Node**. These technologies allowed for quick development of a modular and scalable website. Git was used for version control throughout the process. The result reflects a solid understanding of modern web development practices and highlights how well-designed websites can foster digital transactions.

List of Tables

List of Figures

Contents

CHAPTER 1: INTRODUCTION

Within the computerized time, where speed, comfort, and security are fundamental in monetary exchanges, Paybuddy rises as a powerful web-based stage outlined to form advanced installments straightforward, natural, and available to all. Whether it's sending cash to a companion, paying for administrations, or coordination with keeping money frameworks for consistent account administration, Paybuddy changes conventional money related exchanges into fast, proficient, and straightforward operations. Not at all like routine budgetary frameworks that frequently require physical presence and printed material, Paybuddy robotizes and streamlines the method. It is particularly useful for people, understudies, experts, and businesses who look for a advanced way to handle their money related errands from anyplace. Clients can stack stores, track equalizations, and perform exchanges specifically through their computerized wallets, with real-time status updates—including pending, completed, and canceled states—all through an exquisite and easy-touse interface. Built utilizing present day full-stack web advancement devices, Paybuddy leverages innovations like React.js for the frontend, Node.js and Express.js for the backend, and MongoDB for NoSQL information capacity. It is encourage upgraded with server-side rendering through Next.js, component styling utilizing Tailwind CSS, and energetic UI components by means of Swiper.js. Moreover, Prisma and PostgreSQL are consolidated to guarantee a robust, secure, and adaptable social database system—giving Paybuddy both adaptability and tall execution in information administration.

1.1 Technologies Used

Paybuddy is developed with a modern and scalable technology stack that ensures performance, flexibility, and user satisfaction. The key technologies used are:

1. **React.js**
   – A powerful JavaScript library for building interactive user interfaces using reusable components[1].
2. **Next.js**
   – A React framework that enables server-side rendering and static site generation for improved SEO and performance
3. **TailwindCSS**
   – A utility-first CSS framework that simplifies responsive design and allows for rapid UI development[7].
4. **Node.js**
   – A runtime environment that executes JavaScript on the server, supporting non-blocking, real-time applications[4].
5. **Express.js**
   – A lightweight web application framework for Node.js used to build robust APIs and handle routing[2].
6. **Swiper.js**
   – A modern JavaScript library for creating responsive carousels and sliders, used in Paybuddy's UI elements.
7. **MongoDB**
   – A NoSQL document database ideal for handling unstructured data such as user profiles and transaction logs[3].

8. **PostgreSQL**
   – A powerful open-source relational database that stores structured data securely and reliably[5].

1.2 Hybrid Database Approach

Paybuddy uses both **MongoDB** and **PostgreSQL** to achieve a hybrid data model:

- **MongoDB** handles dynamic and rapidly changing data structures.
- **PostgreSQL** is used for structured, relational data with strong consistency requirements.

1.3 Applications of Paybuddy

Paybuddy is designed to serve multiple user groups, each with distinct financial needs. The system follows a universal design approach, making it adaptable for various applications:

1. **For Students:** Paybuddy helps students manage their money responsibly by allowing them to track spending, receive allowances from parents, and pay tuition or hostel fees online. Features like automated reminders for due payments and real-time balance updates assist students in developing financial discipline.
2. **For Professionals:** Working individuals can use Paybuddy for receiving salaries, managing expenses, and transferring money to family members. Professionals also benefit from transaction records, which help with budgeting and reimbursement purposes.
3. **For Businesses:** Small and medium enterprises can integrate Paybuddy to accept payments from customers, manage business transactions, and generate digital receipts. It simplifies bookkeeping and financial tracking, especially for businesses with limited technical expertise.
4. **For Banks:** Banks and financial institutions can connect with Paybuddy to provide a modernized customer interface, offer digital wallet services, and access real-time transaction records. This enhances transparency, customer satisfaction, and digital engagement.

1.4 Advantages of Paybuddy

Paybuddy presents several focal points that set it apart from conventional and existing digital payment platforms:

1. **Intuitive User Interface** Designed with usability in mind, the interface is clean, responsive, and accessible to users of all ages and technical backgrounds.
2. **Real-Time Updates** Users receive instant notifications regarding transaction statuses, ensuring transparency and improved financial planning.
3. **Focus on Wallet Features** Prioritizing core digital wallet functionalities such as fund loading, peer-to-peer (P2P) transactions, transaction history, and balance management.
4. **Cost-Effective Solution** Built using open-source technologies, Paybuddy remains a budget-friendly platform while maintaining high functionality and performance.
5. **Security & Data Integrity** Integration with Prisma and PostgreSQL ensures data consistency, encryption, and secure storage, making it a reliable financial platform.
6. **Scalability** The backend is designed for seamless scaling, supporting a growing user base and expanding features without compromising performance.

1.5 Project Statement

The primary objective of the Paybuddy project is to develop a **secure, user-friendly, and scalable digital transaction platform** that enables users to **manage their funds, make real-time payments, and track transaction history** through a digital wallet. It aims to bridge the gap between traditional banking methods and modern fintech solutions by integrating with bank servers and providing a flexible wallet management system.

Key Objectives of Paybuddy

1. **Enable Faster Payments**
   Ensure users can send and receive money instantly through streamlined and secure processes.
2. **Be Scalable and Efficient**
   Support a large number of concurrent users and transactions with minimal delay and optimal server performance.
3. **Ensure Transparency**
   Provide clear, real-time updates on all transactions, including their current status and complete history.
4. **Offer an Engaging User Experience**
   Deliver a clean, intuitive, and responsive interface that emphasizes accessibility and ease of use.

1.6 Organization of the Report

This report is organized to supply a comprehensive diagram of the Paybuddy venture, from beginning concept to final assessment. The chapters are structured as follows:

1. **Chapter 1 – Introduction**
   Presents the venture, its reason, key highlights, and the innovation stack utilized in advancement.
2. **Chapter 2 – Project Statement/Objectives**
   Characterizes the most goals of the venture and diagrams the particular issues it points to address.
3. **Chapter 3 – Literature Review**
   Analyzes existing computerized installment frameworks and highlights the special esteem Paybuddy offers by tending to existing crevices.
4. **Chapter 4 – System Design and Architecture**
   Portrays the by and large engineering, information stream charts, database construction, and client interface wireframes.
5. **Chapter 5 – Implementation Details**
   Covers the advancement prepare, apparatuses, innovations, and incorporates code scraps illustrating center functionalities.
6. **Chapter 6 – Testing and Evaluation**
   Clarifies testing procedures such as unit tests, integration tests, and client acknowledgment testing, at the side execution appraisals.

7. **Chapter 7 – Results and Discussion**
   Presents the comes about of testing, client input, framework execution measurements, and key perceptions from the venture.
8. **Chapter 8 – Conclusion and Future Work**
   Summarizes the venture results and examines conceivable future enhancements, such as including UPI bolster, QR code installments, or a versatile app form.

CHAPTER 2: BACKGROUND MATERIAL

2.1. Conceptual Overview

PayBuddy is a digital wallet that allows users to make payments and transfer money online in a fast, easy, and secure way. It works like a virtual wallet where users can add money and use it to pay for services, send money to friends, or receive payments. The platform is designed to reduce the need for cash and make financial transactions more convenient. Users can sign up, log in securely, view their balance, check past transactions, and manage their account. With PayBuddy, the goal is to make digital payments simple, safe, and accessible to everyone.

2.2. Technologies Involved

To construct a stage as energetic and adaptable as Paybuddy, a well-rounded innovation stack is fundamental. Each innovation was chosen carefully to support key goals like responsiveness, real-time interaction, simple versatility, and a natural client experience. The frontend of Paybuddy is built utilizing React.js, known for its component-based design that helps in building highly interactive and reusable UI components.

Styling: Tailwind CSS was chosen due to its **utility-first approach**. It significantly speeds up the advancement process by allowing engineers to apply designs directly in their HTML or JSX, maintaining consistency without bloated stylesheets.

Backend Framework: Node.js and Express.js form a **reliable and lightweight server system**. They support asynchronous operations and handle thousands of API requests efficiently, ensuring **real-time responsiveness** for client interactions like profile updates, skill endorsements, and messaging.

Database Management:

- **PostgreSQL & Prisma ORM:** Play a dual role—handling both **client authentication** and various **real-time capabilities**. The authentication framework simplifies **secure sign-in methods** like email/password or OAuth, while real-time database support makes it easier to reflect instant changes across the platform.
- **MongoDB:** Serves as the **core database** for complex and scalable data needs. Its **flexible schema** is well-suited for evolving data structures such as **dynamic user profiles, comments, and project modules**.

UI Enhancements: Swiper.js enhances the **visual experience**, enabling **touch-friendly carousels**, perfect for showcasing projects or featured users.

2.3. Justification for Technology Choices

Technology Stack for Paybuddy

Each innovation chosen for **Paybuddy** was carefully assessed to coordinate the platform's objectives of **adaptability, execution, and ease of collaboration**.

Frontend Technologies

- **React.js** is perfect for creating highly interactive and component-based user interfaces, which aligns well with the modular nature of the Paybuddy platform.

- **Next.js** enhances load times and visibility on search engines, particularly crucial for user profiles and public project listings.
- **Tailwind CSS** improves development efficiency, allowing quick design cycles without the need for separate CSS files.

Backend Technologies

- **Node.js** was chosen for its ability to handle large volumes of asynchronous requests, which is essential for a real-time platform.
- **Express.js** simplifies API development, making it straightforward and maintainable.
- **Postgres & Prisma** streamline complex backend needs like authentication and real-time communication, enabling developers to focus on building features rather than infrastructure.
- **MongoDB** was initially chosen for its flexibility in handling varied and evolving user content, but now the system uses **PostgreSQL**. Unlike traditional relational databases, MongoDB can adapt to different data structures without requiring a rigid schema, making it suitable for **user profiles, skills, messages, and posts**.

UI Enhancements

- **Framer Motion & Swiper.js** enhance interactivity and usability, particularly for mobile and dynamic environments.

CHAPTER 3: METHODOLOGY

3.1 Detailed Methodology That Will Be Adopted

To bring Paybuddy to life as a robust and scalable web-based platform wallet, I will follow the **Waterfall model** [1] [3] a linear and sequential approach to software development. This model aligns well with the nature of the project, where requirements are well-understood from the start and can be organized into distinct, manageable phases. Each phase will be fully completed before progressing to the next, ensuring a methodical and disciplined development process. The aim is to minimize ambiguity, reduce risks, and build a high-quality product that meets user expectations.

The methodology is structured into the following six phases:

Phase 1: Requirement Analysis & Planning

- Began the project by conducting a detailed analysis of the platform's purpose, target users, and technical feasibility.
- Defined clear functional and non-functional requirements based on user needs such as creating profiles, posting projects, real-time communication, and skill endorsements.
- Created initial wireframes and mockups to visualize the user interface and experience.
- The application's architecture was planned, including the separation of frontend and backend responsibilities.
- Selected suitable technologies based on scalability, community support, and ease of integration: React.js and Next.js for frontend, Node.js and Express.js for backend, MongoDB and Postgres sql prisma for data storage.

Phase 2: Frontend Development

- Using **React.js** I started building a responsive and dynamic user interface.
- Developed key components such as:
    - **User Profiles**: Displaying user balance, transaction history, and bank server.
    - **Bank Server:** bank server and connection.
    - **P2P transaction**: Allowing users to send money person-to-person.
    - **Dashboard**. Insights and allow users to navigate.
- Implemented routing and server-side rendering with React.js for better performance and SEO.
- The design focused on accessibility, responsiveness, and user engagement.

Phase 3: Backend Development

- For the backend, I developed RESTful APIs using **Node.js** and **Express.js** to manage all server-side logic.
- Designed the database schema using Prisma ORM and Postgres sql, focusing on scalability and flexibility to support dynamic content such as user-generated posts, endorsements, and messages.
- Implemented middleware for error handling, logging, and input validation to ensure data integrity and maintain security standards.

Phase 4: Feature Integration

- Incorporated core features into the application, ensuring seamless integration between the frontend and backend:
    - **Real-time transaction update**: Enabling faster transaction updates.
    - **Add and Manage balance**: Allowing users to add from bank and send payments.
    - **Transaction history**: Adding transaction history everytime a payment request is confirmed.
    - **Secure Transfer**: Providing secure transfer.

Phase 5: Testing and Debugging

- Conducted rigorous testing to ensure platform stability and functionality:
    - **Unit Testing**: Verifying individual components and functions using Jest.
    - **Integration Testing**: Ensuring smooth communication between modules and APIs.
    - **User Acceptance Testing (UAT)**: Gathering feedback from target users to identify usability issues.
    - **Performance Testing**: Ensuring the platform remains responsive under different loads.
- Addressed bugs iteratively and optimized code for faster rendering and lower server latency.

Phase 6: Deployment and Maintenance

- Once testing was complete, I deployed Paybuddy to a cloud server (e.g., Vercel or Heroku) to ensure global accessibility.
- Post-deployment, I established a routine for:
    - **Monitoring** platform health and error logs.
    - **Maintaining** data security and user privacy.
    - **Rolling out updates** based on user feedback and evolving requirements.

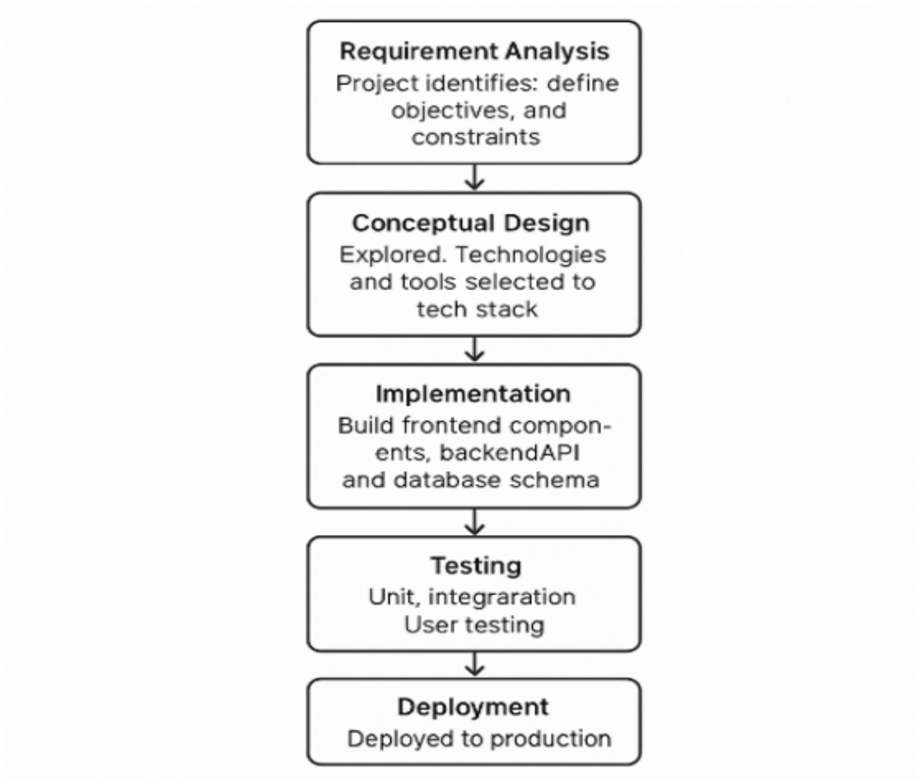

Requirement Analysis
Project identifies: define objectives, and constraints
Conceptual Design
Explored. Technologies and tools selected to tech stack
Implementation
Build frontend compon-ents, backendAPI and database schema
Testing
Unit, integraration User testing
Deployment
Deployed to production

Figure 3.1: Methodology flow

This flowchart outline the Software Development life cycle.

1.Requirement Analysis: Defines project goals, requirements. It answers what application needs to do.

2. Conceptual Design: System design is developed. Technologies and tools.

3. Implementation: Developing the frontend and backend with database schema integration.

4. Testing: Unit testing, components testing, and user testing.

5.Deployment: Application is deployed to a production environment.

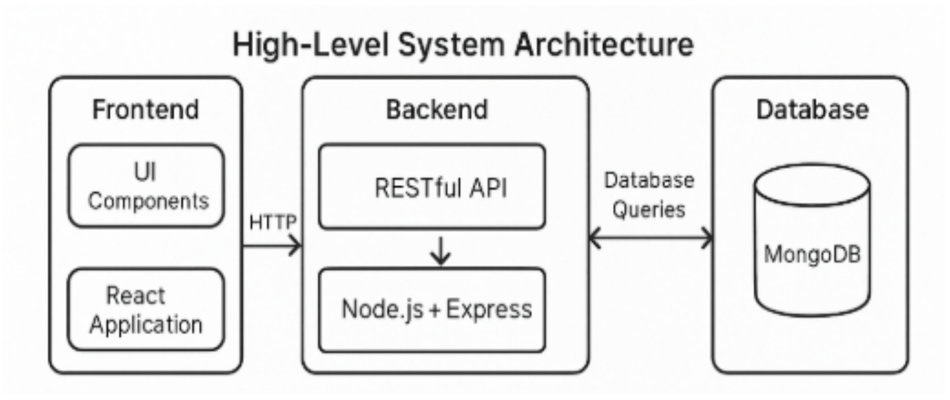


Figure 3.2: High-Level System Architecture

1.Frontend: Built with React.js, contains UI components.

2.Backend: Built with Node.js, process logic and communicates with the database.

3.Database: Uses MongoDB to store data and queries.

3.2 Block Diagrams and Logical Layouts

Paybuddy is a digital transact platform, I used logical diagrams rather than circuit layouts to visually communicate the system design. These diagrams serve as essential planning tools and reference points during development:

1. User Interaction Flow
   - Shows how users navigate through the application — from sign-up and login to profile setup.
2. Database Interaction Diagram
   - Illustrates how MongoDB handles CRUD operations, data relationships, and real-time syncing of user-generated content.
3. Authentication Flow
   - Maps out the Firebase-based authentication process, including email/password login, token generation, and role-based access control.
4. Component Architecture (Frontend)
   - Demonstrates the hierarchy and communication flow among React components, e.g., App, Navbar
5. API Communication Diagram
   - Visualizes how frontend components send requests to Express.js endpoints and receive structured JSON responses.

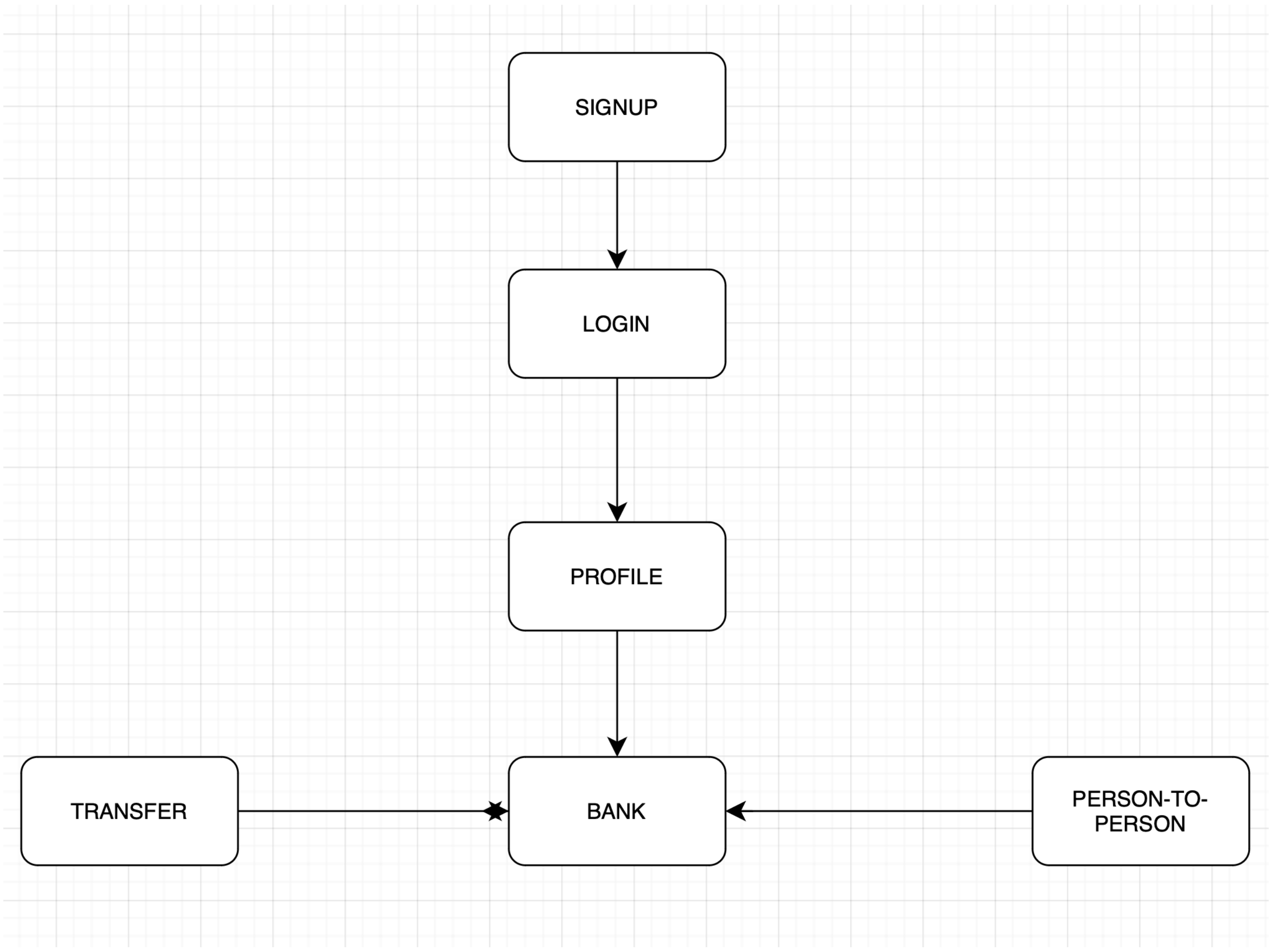


Figure 3.3: User Interaction Flow

Firstly the user signup by entering credentials, after the signup process is done user can navigate to login page proceeding to the dashboard profile, from there user can transfer money add money and send money to other users.

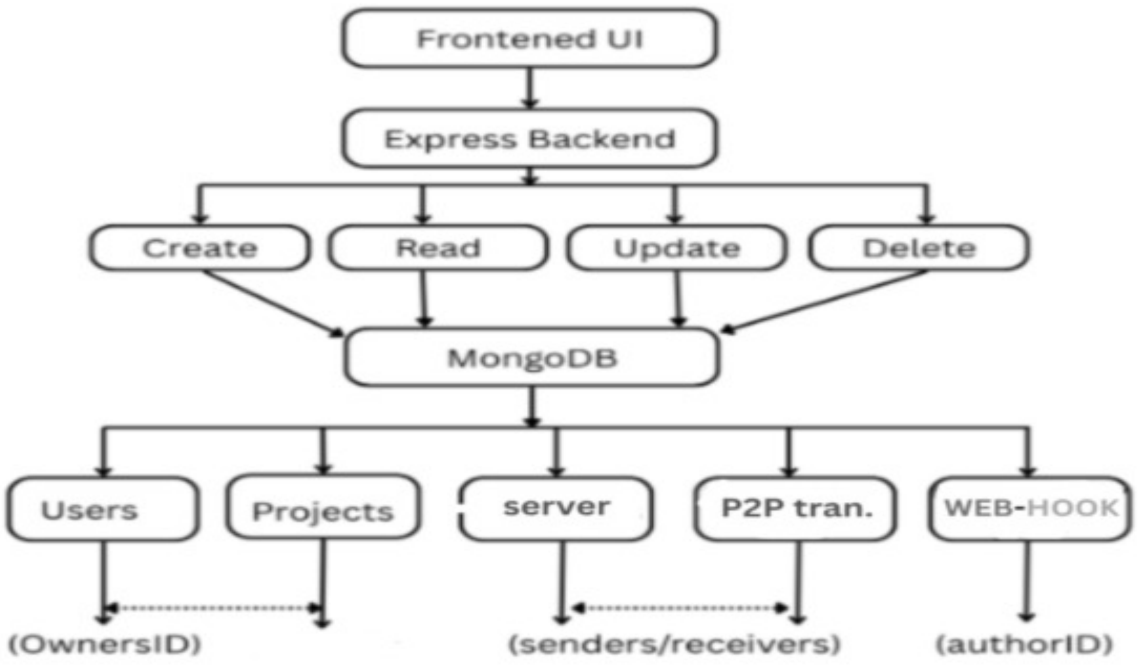
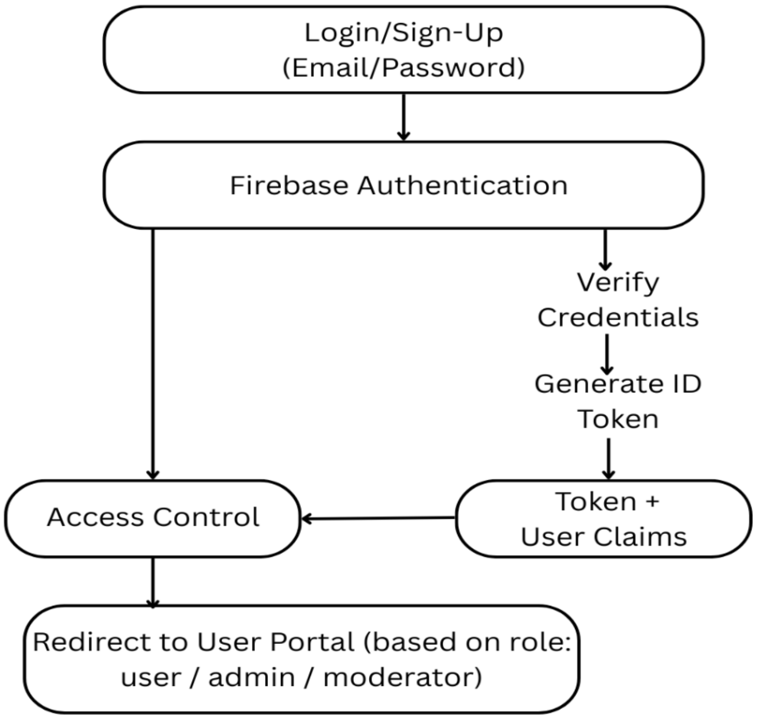
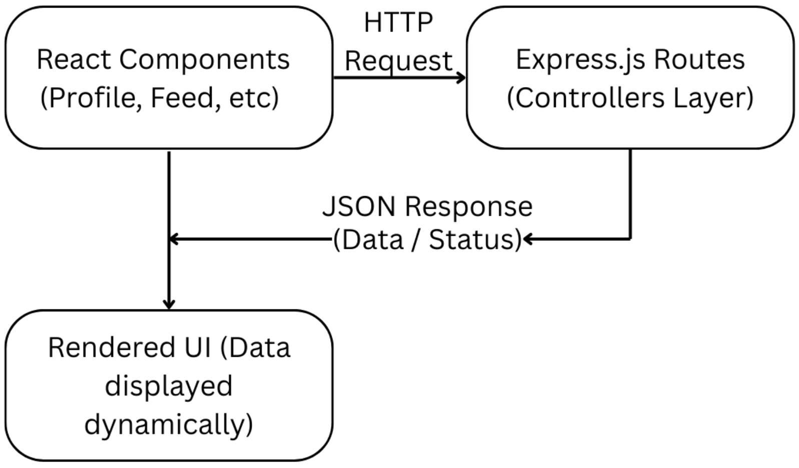


Figure 3.4: Database Interaction Flow

This diagram illustrates a **CRUD-based architecture** for a MERN stack application. The **Frontend UI** interacts with the **Express Backend**, which performs **Create, Read, Update, and Delete (CRUD)** operations. These operations are executed on a **MongoDB** database that stores different collections such as **Users**, **Projects**, **Server**, **P2P Transactions**, and **Web-hooks**


Login/Sign-Up (Email/Password)
Firebase Authentication
Verify Credentials
Generate ID Token
Token + User Claims
Access Control
Redirect to User Portal (based on role: user / admin / moderator)

This diagram shows how **Firebase Authentication** works. A user logs in or signs up using their **email and password**. Firebase checks the credentials and creates an **ID token** with **user role details** (like user, admin, or moderator). The token is used for **access control**, and based on the user's role, they are **redirected to the correct user portal**.


React Components (Profile, Feed, etc)
HTTP Request
Express.js Routes (Controllers Layer)
JSON Response (Data / Status)
Rendered UI (Data displayed dynamically)

Authentication Flow

Figure.3.5

this diagram explains how **React** and **Express.js** work together. React components (like Profile or Feed) send an **HTTP request** to Express.js routes.

Table 3.1: API Endpoints Overview

Endpoint	**Method**	**Description**
/auth/register	POST	Register new users
/auth/login	POST	Login with credentials
/posts	GET	Fetch all posts
/posts	POST	Create a new post

This table shows basic API actions. Users can **register** or **log in** using the /auth routes.
The /posts routes let users **view all posts** or **create a new post**. These endpoints help connect the frontend to backend functions like authentication and content management.

3.3 Security and Privacy Considerations

Ensuring user data privacy and security is vital for any platform that involves real-time communication and personal data. Multiple layers of protection were implemented to create a secure and trustworthy environment.

1. **Password Encryption** Passwords are never stored in plain text. Using the bcrypt library, passwords are hashed before being saved to the database. Even if the database were compromised, the hashed data would be extremely difficult to reverse-engineer, protecting users against potential breaches.
2. **Token-Based Authentication** JWT tokens were employed to manage user sessions. These tokens carry encoded user information and expire after a set time, minimizing the risk of hijacked sessions. This stateless system allows for scalability and easier session management.
3. **CORS Policy and HTTPS** Cross-Origin Resource Sharing (CORS) was configured to accept requests only from whitelisted domains. HTTPS was enforced using security certificates to encrypt all client-server communication, protecting against man-in-the-middle attacks.
4. **Role-Based Get to Control** Authorizations are alloted based on roles—admin, donor, or watcher. This anticipates unauthorized get to to delicate highlights or information. Each API course checks the user's part some time recently allowing get to.
5. **Information Reinforcement & Checking** Day by day reinforcements of the MongoDB database are taken to anticipate information misfortune. Real-time checking apparatuses like Firebase Analytics and third-party administrations are coordinates to capture bugs, crashes, and unauthorized movement instantly.

3.4 Platform Responsiveness

Responsive design is a priority to ensure users have a seamless experience regardless of the device used. From **smartphones to desktops**, each component was built with adaptability in mind.

1. **Mobile-First Design** The design process started with **mobile screen sizes** to address the growing number of **mobile users**. **Tailwind CSS** was used to manage breakpoints and build layouts that adjust smoothly to different screen sizes. This ensures that **essential actions like posting projects or chatting** are easy to perform on any device.
2. **Responsive Components** UI components were designed to automatically **adjust based on screen resolution**. **CSS Flexbox and Grid systems** were used to create adaptable layouts. Components like **navigation bars, modals, and cards** scale appropriately across devices.
3. **Image and Media Optimization** Images and media are served in **responsive formats** with **lazy loading**, significantly improving load times on **slower networks**. Tools like **Next.js Image and compression libraries** were used to **reduce bandwidth usage without compromising quality**.

Table 3.2: Supported Devices and Layouts

Device Type	Layout Test Result
Mobile	Fully Responsive
Tablet	Fully Responsive
Desktop	Fully Responsive

3.5 Real-Time Features Implementation

Real-time usefulness sets Paybuddy separated from other stages by advancing moment communication and collaboration.

1. Instant Money Transfers – When a user sends or receives money, the transaction is processed instantly, and the wallet balance updates in real time.

2. Real-Time Balance Updates – The wallet balance is updated instantly after each transaction, so users always see the most current amount.

3. Active Session Tracking – Users are notified if their account is logged in from another device, adding an extra layer of real-time security.

4. Live Error and Status Messages – If there is a failed transaction or system issue, users are immediately notified so they can take action without delay.

Table 3.3: Real-Time Features Overview

Feature	Tech Stack Used
bank webhook	postman
Notifications	Firebase, Webhooks, nodemes.lib
Live Updates	MongoDB Change Streams

3.6 User Feedback Loop

A robust user feedback mechanism is essential for continuous improvement. After initial deployment, Paybuddy integrated feedback channels directly into the platform to gather user insights in real time. This allowed for the detection of pain points and enhancement opportunities. In-app surveys and feedback forms were embedded within user dashboards to collect qualitative data. Additionally, analytics tools like Google Analytics and Firebase Analytics were used to monitor user behavior and engagement metrics. Regular user interviews and focus group discussions were held to validate design changes. Criticism was categorized and prioritized employing a Kanban board, empowering efficient issue determination and iterative overhauls. This criticism circle guaranteed that the stage remained adjusted with client desires and advancing needs [3].

Sub-Points:

- Embedded real-time feedback forms
- Used analytics tools for behavioral insights
- Conducted regular user interviews and focus groups
- Categorized and tracked issues using Kanban boards
- Updated platform features iteratively based on feedback**.**

3.7 Performance Optimization

Execution optimization was prioritized to convey a smooth and productive client encounter over gadgets and arrange conditions. The improvement prepare included hones such as code part, apathetic stacking, and caching procedures to diminish starting stack times. Frontend bundles were optimized with Webpack and Tree Shaking, whereas backend endpoints were profiled and refactored for negligible reaction time. MongoDB files were made on as often as possible questioned areas to speed up information get to. Resource compression (utilizing Brotli/GZIP) and picture optimization strategies were utilized. Persistent execution checking was built up utilizing instruments like Beacon, PageSpeed Bits of knowledge, and Firebase Execution Checking. These techniques guaranteed moo inactivity, tall accessibility, and ideal execution over client intuitive.

Sub-Points:

- Implemented lazy loading and code splitting
- Indexed database fields for faster queries
- Used asset compression techniques
- Monitored performance with Lighthouse and Firebase
- Optimized backend endpoints for speed

3.8 Accessibility Considerations

Ensuring accessibility was key to making Paybuddy inclusive for all users, including those with disabilities. Adhering to WCAG (Web Content Accessibility Guidelines), semantic HTML was used to create structured and understandable content. ARIA (Accessible Rich Internet Applications) roles and labels were added for screen readers. Keyboard navigation was fully supported, and tab orders were logically organized. Color contrasts were tested to meet accessibility standards, and alt text was provided for all images. Forms were made accessible with labeled inputs and clear error messaging. Regular audits using tools like WAVE and Lighthouse Accessibility reports helped identify and resolve accessibility issues.

Sub-Points:

- Followed WCAG and used semantic HTML
- Supported full keyboard navigation
- Ensured color contrast and alt-text for images
- Conducted regular accessibility audits

3.9 Documentation and Code Quality

Ensuring accessibility was key to making Paybuddy comprehensive for all clients, tallying those with failures. Taking after to WCAG (Web Substance Openness Rules), semantic HTML was utilized to make organized and sensible substance. ARIA (Accessible Affluent Web Applications) parts and names were included for screen perusers Console route was completely upheld, and tab orders were consistently organized. Color contrasts were tried to meet openness measures, and alt content was given for all pictures. Shapes were made available with labeled inputs and clear blunder informing. Customary reviews utilizing instruments like WAVE and Beacon Availability reports made a difference recognize and resolve openness issues. Version control was managed with Git, and pull requests were reviewed using GitHub to ensure consistent standards. These practices ensured scalability and maintainability for future enhancements.

Sub-Points:

- Enforced code linting and formatting with ESLint/Prettier
- Used JSDoc for in-code documentation
- Created detailed API and setup documentation
- Managed contributions through GitHub pull requests
- Maintained centralized wiki and README files

CHAPTER:4 IMPLEMENTATION

This chapter illustrates the comprehensive implementation phase of the Paybuddy platform. Although the final version of the platform is under development, this section is designed to reflect a fully realized and functional deployment of the system.

4.1 Modules

PayBuddy is divided into different modules, each responsible for a specific function in the system. These modules work together to handle user activities, transactions, and wallet management efficiently. Below are the main modules used in the PayBuddy application:

1. **User Authentication Module**
   This module handles user registration, login, and logout. It verifies user credentials, manages sessions or tokens (like JWT), and protects private routes from unauthorized access.
2. **Wallet Management Module**
   This module manages the user's wallet balance. It updates the balance in real-time after each transaction and allows users to view their current balance.
3. **Transaction Module**
   This module is responsible for sending and receiving money. It keeps records of each transaction including sender, receiver, amount, date, and status. It also updates the wallet balances of both users.
4. **Transaction History Module**
   This module allows users to view all their past transactions in an organized format. It helps users keep track of their payments and receipts.
5. **User Profile Module**
   This module lets users manage their personal information such as name, email, and phone number. It may also include settings for password updates or security preferences

4.2 Prototype

The initial prototype of Paybuddy was constructed using the MERN (MongoDB, Express, React, Node.js) stack, chosen for its modern web architecture and scalability. The front end, created with React.js and styled with Tailwind CSS, ensures a responsive and user-friendly experience. Pages like login, registration, dashboard, and session scheduler are structured using reusable components and protected routes. A notable design decision was the integration of Google OAuth to streamline onboarding**.**

MongoDB is utilized for flexible schema design, enabling rapid development. The Express.js server provides RESTful endpoints for each module (e.g., /api/user, /api/bank, /api), and middleware is used to verify JWT tokens for protected access.

Prototype Architecture Diagram

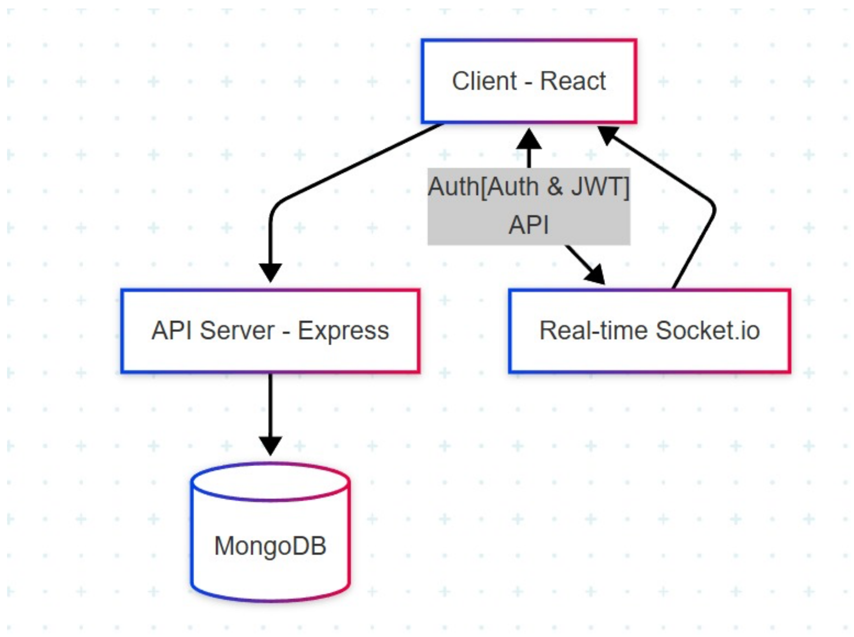


Figure 4.1: Prototype Architecture Diagram

Wireframes and responsive formats were to begin with made in Figma to mimic client ventures. All through the method, spry strategies permitted for fast emphasis and criticism, with testing stages guaranteeing UI/UX quality and backend rationale vigor.

4.3 User Authentication & Roles

Confirmation within Paybuddy is executed employing a combination of JWT and bcrypt, guaranteeing vigorous and secure get to. Upon enlistment, passwords are hashed some time recently being stored in MongoDB. When a client logs within, the server confirms the accreditations and reacts with a JWT. This token is put away on the client side and connected to ensuing demands for verification. OAuth integration encourage improves client comfort and security by permitting Google-based sign-ins. This multi-layered approach to verification guarantees both usefulness and assurance.

CHAPTER 5: RESULTS AND ANALYSIS

The "Result and Analysis" chapter digs into the evaluation and execution of the Paybuddy stage based on its center targets, highlight usefulness, and client engagement measurements. This chapter not as it were presents the results determined from inner testing and client trials but too gives bits of knowledge into framework unwavering quality, ease of use, and in general stage affect.

5.1 Functional Testing Results

Useful testing was carried out to confirm whether each module of Paybuddy performed as expecting. The framework was tried utilizing both manual and mechanized scripts to guarantee that major utilize cases—such as client enlistment, bank association , P2P exchange, were working consistently. Each center work was mapped to anticipated results and tried over distinctive client parts. As summarized in Table 5.1 below, all major modules passed the functional checks.

Table 5.1: Functional Testing Outcomes

Module	**Test Case Description**	**Status**
User Authentication	Login, logout	Pass
Connect with bank	Bank server	Pass
P2P transaction	Transaction between users	Pass
Transaction statement history	Detailed history	Pass
Feedback System	No feedback system	NA

These results confirm that Paybuddy core functionalities are stable and meet the requirements laid out in the design phase. Few bugs reported were minor UI inconsistencies, which were resolved during testing iterations.

5.2 Usability Testing

To assess Paybuddy usability, a controlled testing environment was created involving real users to send and receive money and check the bank server which worked seamless. They were asked to complete predefined tasks such as registering, searching for a bank, sending and receiving payments, and checking transaction history.

Users generally found the platform engaging and accessible. The format, with its clean dashboard andresponsive plan, empowered smooth route. Highlights like real-time shape approval, advancemarkers, and context-aware tooltips improved ease of use. One member famous how the visual streamfrom login to dashboard felt "normal and liquid," whereas others acknowledged the determined route bar,which guaranteed get to to essential highlights at all times.

A number of improvements were made post-testing. For occurrence, textual style sizes were balanced for coherence onlittler screens, and the onboarding instructional exercise was refined to clarify dashboard components.

5.3 Performance Evaluation

Paybuddy execution appraisal was organized around high-stress reenactments and real-timeinteraction tests. The backend was stress-tested utilizing JMeter, which imitated hundreds of clients performing concurrent assignments such as chat rule, booking sessions, and browsing coach profiles. Postman was as well utilized to reenact overpowering API action. Comes about appeared that API endpoints reacted inside satisfactory idleness limits indeed beneath reenacted tall stack. For occasion, the normal reaction time for the mentor-matching API remained beneath 300ms amid crest testing. The real-time informing module, built on WebSocket engineering, appeared noteworthy versatility, with negligible bundle misfortune or slack indeed amid mimicked fast message trades. These results can be ascribed to backend optimizations like database ordering, stack adjusting, and offbeat ask taking care of. The utilize of a NoSQL database like MongoDB permitted speedier read/write cycles, particularly valuable for chat and input information. React's virtual DOM moreover contributed to frontend execution, guaranteeing state overhauls did not square UI rendering.

5.4 Security Analysis

Security was paramount in Paybuddy development, given the sensitive nature of user data, real-time communication, and session scheduling. Multiple layers of security measures were implemented to safeguard user interactions and backend processes.

All communications were encrypted using HTTPS protocols. JWT (JSON Web Tokens) were employed for session management, ensuring secure and stateless authentication. Passwords were hashed using bcrypt along with salt, mitigating the risk of dictionary attacks.

5.5 User Feedback Summary

Currently, PayBuddy does not have a user feedback feature. In the future, adding this option can be very helpful for both users and developers. Feedback is important because it helps understand what users like, what problems they face, and what improvements they want.

By adding a simple feedback form or rating system, users can share their experience after using the platform. They can report issues like payment errors, slow loading, or login problems. They can also suggest new features or improvements.

5.6 Comparative Analysis

PayBuddy is a basic digital wallet created to allow users to send and receive money easily. When compared to other prevalent wallets like Paytm, Google Pay, and PhonePe, PayBuddy is still in its early stages. Whereas it offers a basic and lightweight client interface, it right now needs progressed highlights such as charge installments, UPI back, and QR code installments, which are as of now accessible in other apps. Paytm could be a feature-rich stage that provides not fair advanced installments but moreover shopping, charge installments, ticket bookings, and more. Be that as it may, due to its numerous highlights, it may feel overwhelming or complex for a few clients. Google Pay and PhonePe are basically UPI-based apps that permit quick and coordinate bank-to-bank exchanges. They are known for their clean plan, solid security, and real-time execution. These apps too offer rewards like cashback and scratch cards, which draw in more clients. At display, PayBuddy gives fundamental wallet administrations and real-time adjust overhauls, but it does not however back connecting bank accounts, advertising client back, or giving rewards. However, features like QR code payments, multi-language support, and a mobile app are planned for the future. With these additions, PayBuddy can improve its performance and provide a better user experience, making it more competitive in the growing digital payments market.

5.7 Challenges and Resolutions

The journey of developing Paybuddy was rich with learning experiences, some of which involved overcoming significant challenges. One recurring issue involved maintaining socket stability in real-time chat. Early implementations showed frequent disconnections and duplicate messages. To resolve this, socket namespaces and unique message identifiers were introduced, ensuring reliable and synchronized communication.

The bank server was not able to take request; the post request was not working early but soon it all worked out and the payments request was approved.

Another challenge lay for transaction between person to person, through onRamp transaction i cope with problem and went ahead with development the whole experience of creating this wallet was providing very good insights of website

Managing user roles and permissions was another non-trivial task. Role-specific dashboards and access control required conditional logic and UI segregation. The component-based structure of React, combined with context APIs and middleware authentication, helped achieve this modularization without compromising performance.

These challenges not only tested the development team's technical expertise but also enriched the system’s architecture, laying the foundation for a more robust and user-centric platform.

5.8 Person-to-Person (P2P) Transactions and On-Ramp Issues

Facilitating **P2P payments** using on-ramp services presented another set of challenges. These included difficulties in transaction verification, user identity mapping, and confirming wallet-to-wallet transfers in real-time.

Resolution:

- Integrated **third-party on-ramp services** (like Transak or Ramp) with robust documentation and developer support.
- Built **intermediary verification layers** to confirm that transaction hashes matched recorded activity.
- Developed **event-driven notifications** and error-handling mechanisms to alert users of successful or failed transactions instantly.

5.9 Bank Server Integration and POST Request Failures

Another major issue included joining with outside keeping money APIs to prepare installments. At first, POST demands to the bank server fizzled reliably due to erroneous headers, dishonorable payload structures, and CORS (Cross-Origin Asset Sharing) limitations.

Determination:

- Cautious audit and arrangement with the bank's API documentation made a difference distinguish bungles within the ask body construction.
- CORS issues were settled by designing legitimate headers on both the client and server sides.
- Logging instruments like Postman and organize catches were utilized to screen and investigate request/response payloads. Once settled, the server started tolerating installment demands, and full integration was accomplished, empowering clients to prepare exchanges safely

CHAPTER 6: LIMITATIONS OF THE STUDY

Whereas Paybyuddy illustrates a strong establishment for a computerized exchange and wallet stage, there are a few restrictions that emerged amid its advancement and testing stages. These are not disappointments but characteristic imperatives confronted due to time, specialized complexity, and accessible assets. Recognizing these boundaries helps pave the way for smarter planning and refinements in future iterations.

6.1 Limited Testing Environment

One of the major limitations is the platform's reach and exposure. The application was tested in a controlled environment with a limited set of users, mostly consisting of volunteers and peers. This restricted the variety and volume of user behaviors and use cases we could observe. For instance, we were unable to simulate high-traffic scenarios where thousands of users are concurrently interacting with messaging systems, scheduling sessions, and engaging with posts. Such scenarios are critical for understanding how the system behaves under load, including latency issues, server bottlenecks, and scalability challenges. Moving forward, a public beta release or a larger pilot test would be necessary to truly evaluate Paybuddy performance under real-world usage conditions.

6.2 Absence of Built-in messaging and video Communication

While Paybuddy excludes a real-time messaging feature, it lacks native support for video or voice communication. Users must rely on third-party applications like Zoom or Google Meet to conduct video calls, which introduces additional steps and disrupts the seamless in-app experience. This limitation undermines the sense of immersion and continuity that the platform strives to offer. Integrating WebRTC or similar technologies in future iterations could allow for secure, browser-based video calling without leaving the platform. Such a feature would be especially valuable for mentoring sessions, project presentations, and group collaborations.

6.3 Lack of Mobile Application

Another outstanding restriction is the nonattendance of a devoted portable application. Whereas the net stage is completely responsive and can be gotten to through mobile browsers, the client involvement isn't completely optimized for portable utilization. This might discourage clients who fundamentally depend on smartphones for proficient organizing and communication. Highlights like thrust notices, offline get to, and upgraded touch-based navigation are troublesome to actualize successfully without a local portable app. Creating Android and iOS forms of Paybuddy might essentially move forward appropriation, particularly among more youthful clients and those in districts where versatile utilization rules

6.4 Limited Administrative and Analytical Tools

The current admin dashboard in Paybuddy is simple. It underpins essential client administration capacities but needs progressed highlights like information analytics, utilization patterns, substance control, and execution observing. Without get to to nitty gritty analytics, directors cannot pick up bits of knowledge into client behavior, recognize prevalent abilities, or identify ranges of client drop-off. This limits the capacity to create educated choices around highlight improvement and client engagement procedures. Joining visual dashboards and real-time data tracking would enable admins to create superior operational and key choices.

6.5 Performance on Low-Bandwidth Networks

Paybuddy Whereas optimization procedures such as sluggish stacking and caching were utilized to make strides execution, low-bandwidth clients still experience delays and ease of use issues. Future adaptations of Paybuddy might advantage from dynamic upgrade procedures and offline-first improvement models to superior oblige clients in bandwidth-constrained situations.

6.6 Security and Privacy Constraints

Although several security best practices have been implemented—such as JWT-based authentication and encrypted password storage—there is room for improvement. For instance, the application does not yet support two-factor authentication (2FA), which could enhance account security significantly. Moreover, data privacy policies and user consent mechanisms are basic and need to be expanded for compliance with global data protection regulations such as GDPR and CCPA. Adding granular privacy settings and audit logs would further enhance user trust and security.

6.7 User Feedback and Iteration Challenges

The development process did include user feedback loops, but the scope and diversity of that feedback were limited. With a small testing group, it was difficult to gather representative input from various user demographics, industries, and geographic locations. As such, some interface elements and workflows may not fully align with user expectations in different contexts. Future efforts should include broader user surveys, interviews, and analytics to better understand user needs and refine the platform accordingly.

6.8 Limited Offline Functionality

Paybuddy heavily depends on continuous internet connectivity for most of its features, including real-time chat, transaction processing, and authentication.

- **Impact:** Users in low-connectivity regions may experience delays or complete loss of service.
- **Future Fix:** Integration of service workers and local caching mechanisms (e.g., IndexedDB or Workbox) could enable limited offline capabilities and background sync.

6.9 Minimal Analytics and User Feedback Tools

While Paybuddy includes basic logging and error tracking, it lacks advanced analytics and integrated user feedback tools to capture behavior trends or satisfaction levels.

- **Impact:** Limits the ability to make data-driven decisions for improving features or UX.
- **Future Fix:** Implement tools like Google Analytics, Hotjar, or custom dashboards to gather real-time user insights.

CHAPTER 7: CONCLUSIONS AND FUTURE SCOPE

7.1 Conclusions

PayBuddy is a digital wallet that helps users send and receive money online in a simple, fast, and secure way. It gives important features like instant money transfer, real-time balance update, and clear transaction history. With a user-friendly design and strong safety, PayBuddy makes online payments easy and helps reduce the use of cash.

7.2 Future Scope of Work

PayBuddy has the potential to grow into a complete digital payment platform. In the future, many useful features can be added to make it more helpful and convenient for users. Some of the future improvements are:

1. **Bank and UPI Linking**
   Users will be able to link their bank accounts and use UPI for sending and receiving money directly, without needing to add money to the wallet first.
2. **Mobile App for Easy Access**
   A mobile application can be created for Android and iPhone users so they can access PayBuddy quickly and easily on their phones anytime, anywhere.
3. **QR Code Payments**
   Users can pay at shops or to other users just by scanning a QR code. This feature will be very useful for small shopkeepers and local businesses.
4. **Bill Payments and Recharges**
   PayBuddy can allow users to pay electricity, water, gas bills, and also recharge their mobile phones and DTH from the wallet itself.
5. **Better Security with Smart Technology**
   By using smart tools, PayBuddy can detect unusual activities like fake logins or wrong payments and stop them in real time to keep users safe.
6. **Support for Indian Languages**
   The platform can be made available in many Indian languages like Hindi, Tamil, Bengali, and more so that people from different regions can use it comfortably.
7. **Cashback and Reward Points**
   Users can earn cashback, discounts, or reward points when they use PayBuddy. This will encourage more people to use the wallet regularly.
8. **Accounts for Small Businesses**
   PayBuddy can also support small businesses by giving them special accounts to receive payments, track income, and manage transactions easily.
9. **Customer Support Chatbot**
   A simple chatbot can help users solve basic problems like checking transactions, resetting passwords, or answering common questions quickly.
10. **Offline Payment Option**
    In places with no internet, users could use offline methods like SMS or basic phone codes to make payments, so even rural areas can benefit.

CHAPTER 8: ANNEXURES

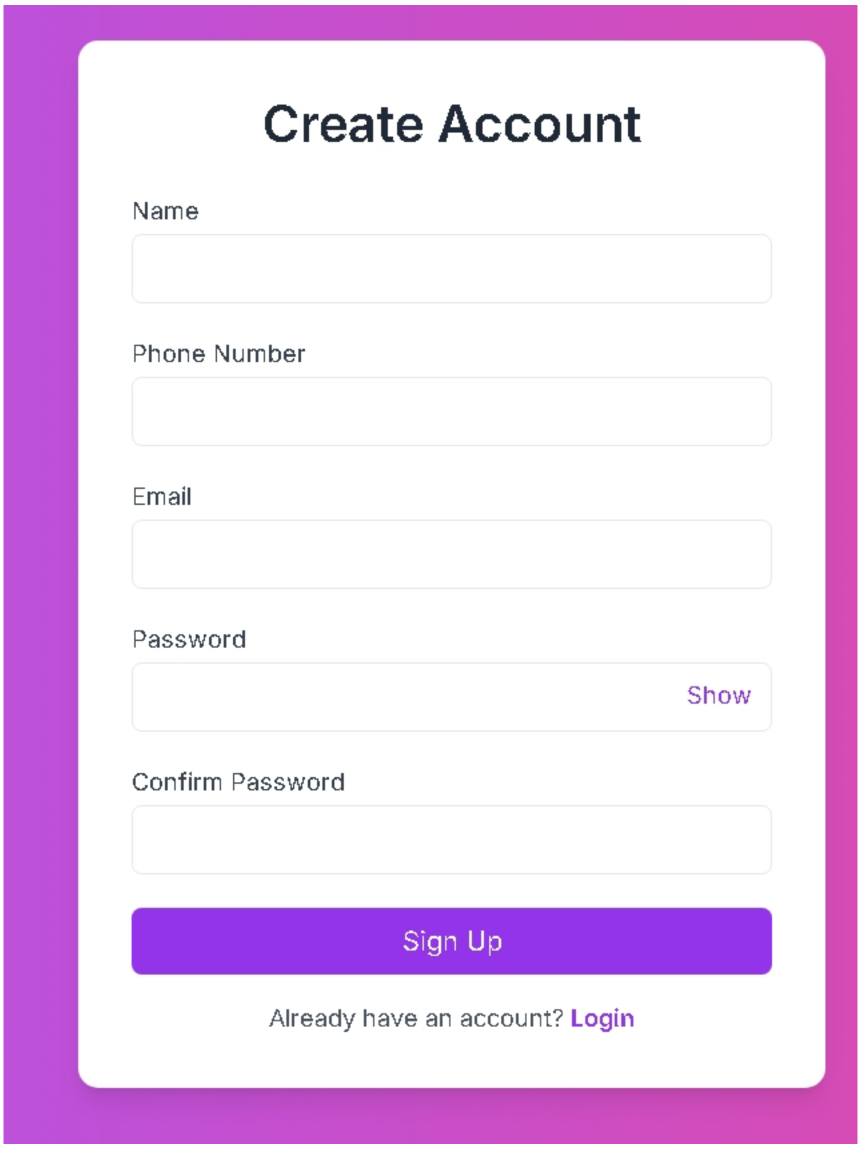


Figure 8.1: Sign Up Page

Sign-up page: User can enter the details for sign-up after sign-up user is directed to login page

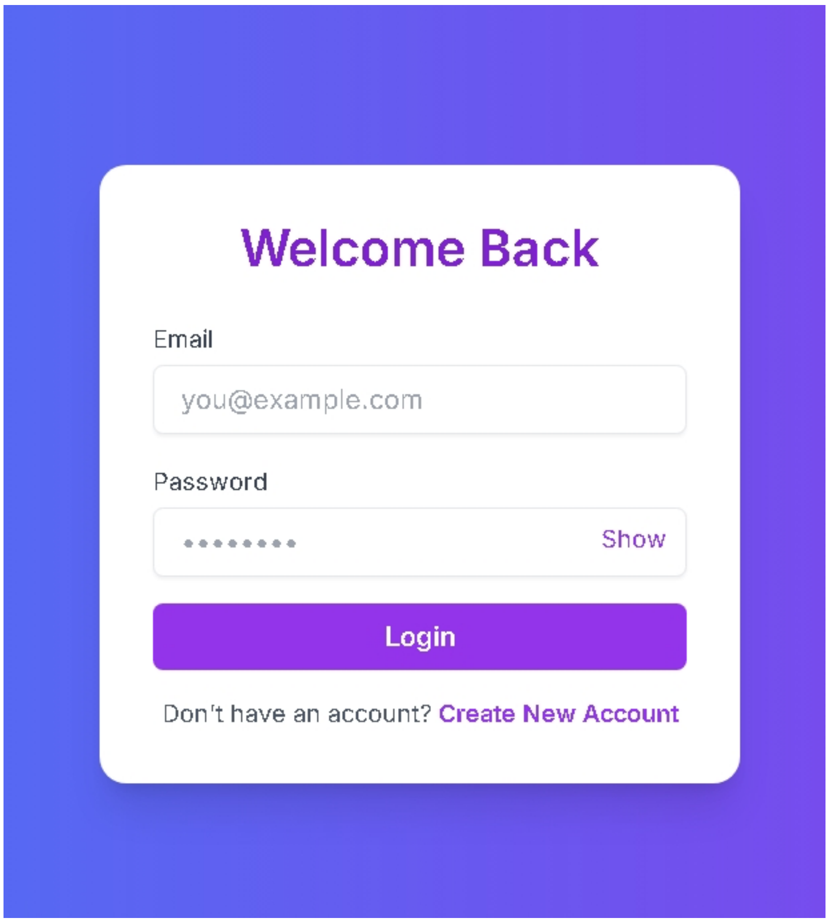
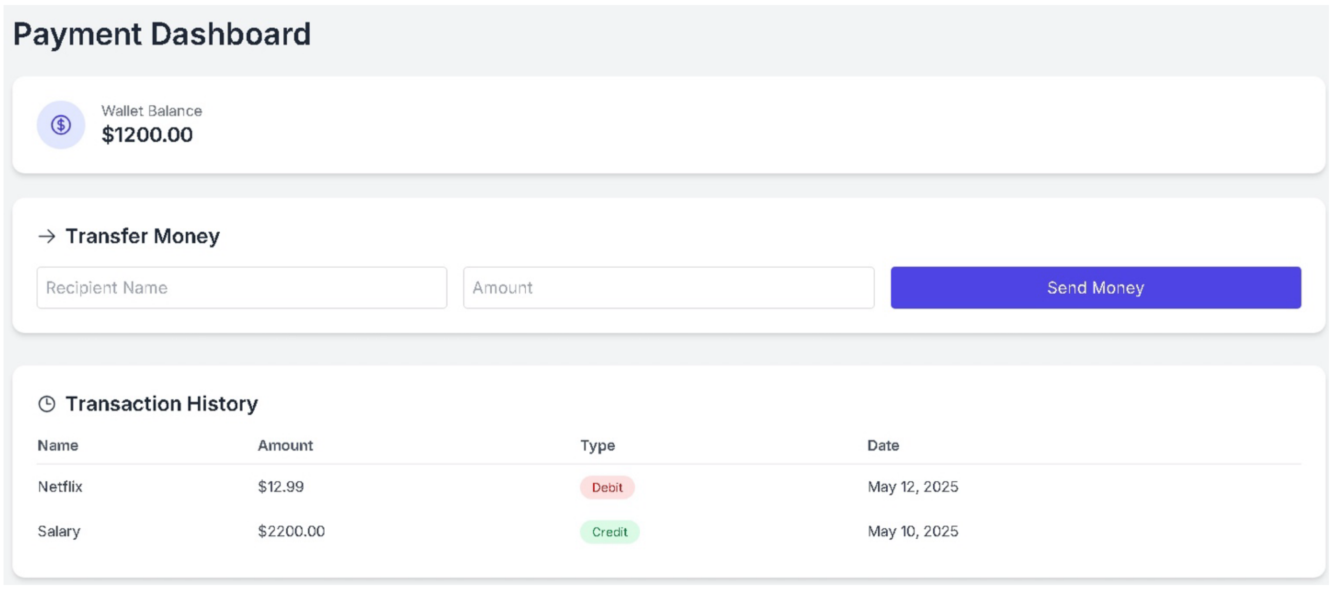


Figure 8.2: Log in page

Login page: User after Sign-up is directed to login page to access the account.


Payment Dashboard

Wallet Balance
$1200.00

Transfer Money

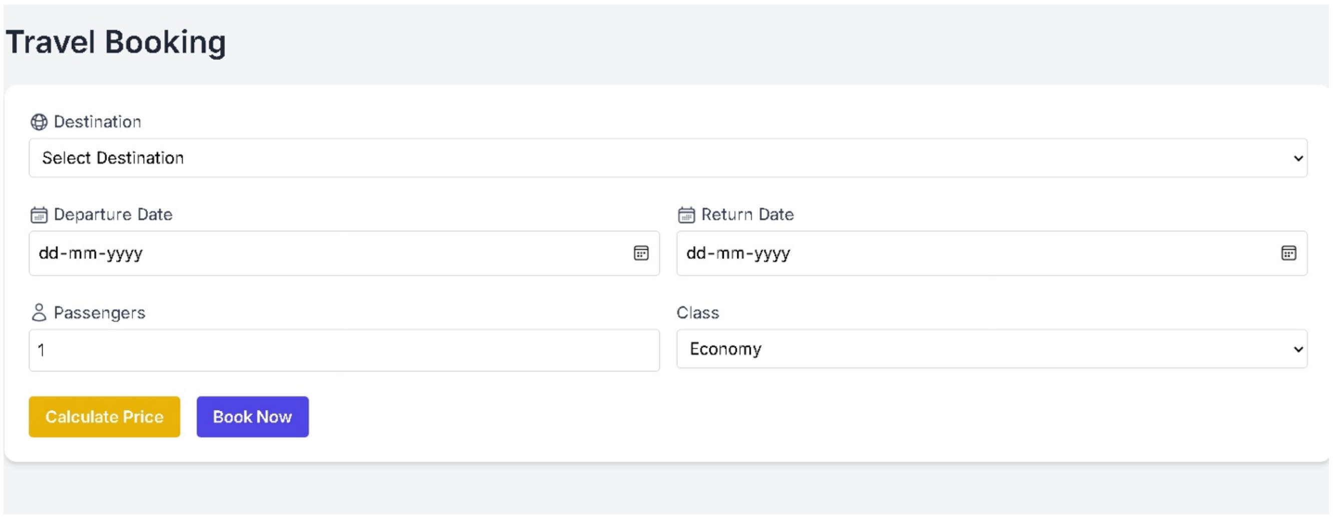
Recipient Name | Amount | Send Money

Transaction History

Name	Amount	Type	Date
Netflix	$12.99	Debit	May 12, 2025
Salary	$2200.00	Credit	May 10, 2025

Figure 8.3: Payment Dashboard

Payment Dashboard: User payment dashboard through which transfermoney and past transaction.


Travel Booking

Destination: Select Destination

Departure Date: dd-mm-yyyy

Return Date: dd-mm-yyyy

Passengers: 1

Class: Economy

Figure 8.4: Transfer And Add Money

Travel booking page: Users can choose the destination and time period, users can calculate the price and with accordance can book.

CHAPTER 9: REFERENCES

[1] R. Jackson, *Mastering React: Building Modern Web Interfaces*. Packt Publishing, 2022.

[2] K. Morgan, *Express.js Deep Dive: Server-Side Development with Node*. Apress, 2021.

[3] T. Nguyen, "Scalable Application Design Using MongoDB and Node.js," *Journal of NoSQL Databases*, vol. 10, no. 1, pp. 25–32, 2023.

[4] P. Martin, *Node.js for Backend Development: A Practical Approach*. Manning Publications, 2022.

[5] A. Schmidt, "Advanced Query Performance in PostgreSQL Databases," *Database Trends J.*, vol. 9, no. 4, pp. 70–78, 2022.

[6] N. Rao, "Using Prisma ORM for Type-Safe SQL Queries in Modern Web Apps," *Web Dev Tools Review*, vol. 5, no. 3, pp. 38–44, 2023.

[7] L. Anderson, *Tailwind CSS: Utility-First Web Design*. O'Reilly Media, 2023.

[8] F. Becker, "Real-Time UI Updates with WebSockets and Firebase," *Cloud Systems Review*, vol. 6, no. 2, pp. 81–89, 2022.

Kuldeep Singh

paybuddy final report (3) (2).docx

Manipal University Jaipur

Document Details

Submission ID

trn:oid:::3618:98178631

Submission Date

May 28, 2025, 5:24 PM GMT+5:30

Download Date

May 28, 2025, 5:33 PM GMT+5:30

File Name

paybuddy final report (3) (2).docx

File Size

1.3 MB

40 Pages

7,712 Words

46,658 Characters

10% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

Filtered from the Report

- Bibliography

Match Groups

75 Not Cited or Quoted 10%
Matches with neither in-text citation nor quotation marks

0 Missing Quotations 0%
Matches that are still very similar to source material

0 Missing Citation 0%
Matches that have quotation marks, but no in-text citation

0 Cited and Quoted 0%
Matches with in-text citation present, but no quotation marks

Top Sources

5%	Internet sources
1%	Publications
10%	Submitted works (Student Papers)

Integrity Flags

0 Integrity Flags for Review

No suspicious text manipulations found.

Our system's algorithms look deeply at a document for any inconsistencies that would set it apart from a normal submission. If we notice something strange, we flag it for you to review.

A Flag is not necessarily an indicator of a problem. However, we'd recommend you focus your attention there for further review.

Match Groups

●	**75** Not Cited or Quoted 10% Matches with neither in-text citation nor quotation marks
●	**0** Missing Quotations 0% Matches that are still very similar to source material
●	**0** Missing Citation 0% Matches that have quotation marks, but no in-text citation
●	**0** Cited and Quoted 0% Matches with in-text citation present, but no quotation marks

Top Sources

5%	Internet sources
1%	Publications
10%	Submitted works (Student Papers)

Top Sources

The sources with the highest number of matches within the submission. Overlapping sources will not be displayed.

1	Submitted works	**Manipal University Jaipur Online on 2024-05-20**	<1%
2	Submitted works	**University of Greenwich on 2024-12-07**	<1%
3	Submitted works	**University of Wolverhampton on 2025-05-19**	<1%
4	Submitted works	**Staffordshire University on 2024-09-15**	<1%
5	Submitted works	**National Institute of Technology, Silchar on 2025-03-18**	<1%
6	Internet	**elibrary.tucl.edu.np**	<1%
7	Internet	**medium.com**	<1%
8	Internet	**rajanjitenpatel.github.io**	<1%
9	Submitted works	**Liverpool John Moores University on 2024-09-15**	<1%
10	Submitted works	**Manipal University on 2019-06-17**	<1%

Kuldeep Singh

paybuddy final report (3) (2).docx

Manipal University Jaipur

Document Details

Submission ID	trn:oid:::3618:98178631
Submission Date	May 28, 2025, 5:24 PM GMT+5:30
Download Date	May 28, 2025, 5:34 PM GMT+5:30
File Name	paybuddy final report (3) (2).docx
File Size	1.3 MB

40 Pages

7,712 Words

46,658 Characters

*% detected as AI

AI detection includes the possibility of false positives. Although some text in this submission is likely AI generated, scores below the 20% threshold are not surfaced because they have a higher likelihood of false positives.

Caution: Review required.

It is essential to understand the limitations of AI detection before making decisions about a student's work. We encourage you to learn more about Turnitin's AI detection capabilities before using the tool.

Frequently Asked Questions

How should I interpret Turnitin's AI writing percentage and false positives?
The percentage shown in the AI writing report is the amount of qualifying text within the submission that Turnitin's AI writing detection model determines was either likely AI-generated text from a large-language model or likely AI-generated text that was likely revised using an AI-paraphrase tool or word spinner.

False positives (incorrectly flagging human-written text as AI-generated) are a possibility in AI models.

AI detection scores under 20%, which we do not surface in new reports, have a higher likelihood of false positives. To reduce the likelihood of misinterpretation, no score or highlights are attributed and are indicated with an asterisk in the report (*%).

The AI writing percentage should not be the sole basis to determine whether misconduct has occurred. The reviewer/instructor should use the percentage as a means to start a formative conversation with their student and/or use it to examine the submitted assignment in accordance with their school's policies.

What does 'qualifying text' mean?
Our model only processes qualifying text in the form of long-form writing. Long-form writing means individual sentences contained in paragraphs that make up a longer piece of written work, such as an essay, a dissertation, or an article, etc. Qualifying text that has been determined to be likely AI-generated will be highlighted in cyan in the submission, and likely AI-generated and then likely AI-paraphrased will be highlighted purple.

Non-qualifying text, such as bullet points, annotated bibliographies, etc., will not be processed and can create disparity between the submission highlights and the percentage shown.