MICROFINANCE BANKING AND PAYMENT SYSTEM

MSc Computing & IT Management

Sauronil Das

C22107670

dasS19@cardiff.ac.uk

Supervisor: Sandy Gould

School of Computer Science and Informatics

Cardiff University

Acknowledgements

I want to offer my heartfelt thanks to my supervisor, Prof. Sandy Gould. His guidance, knowledge, and moral support were instrumental in helping me to successfully complete this project over the last 12 weeks. I am deeply grateful for his mentorship and will always be indebted to him for his invaluable contributions to my academic journey.

I would like to extend my sincerest thanks to Aryan Lakdawala and Yedukrishna Jayan, who were always available to discuss ideas and provide critical feedback on my work. Their insights and suggestions were invaluable in helping me to refine my arguments and improve the overall quality of my dissertation.

I must also acknowledge the invaluable contributions of Tanvi Patwardhan and Shiwali Santosh Charjan, who served as an excellent rubber duck for my ideas and generously offered to proofread my report. Their keen eye for detail and thoughtful suggestions helped me to polish my work and present it in the best possible light.

Last but certainly not least, I would like to express my heartfelt gratitude to Abacws Building who itself feels like a person who has been like a pillar, supporting me during the challenging period of completing my dissertation. Its facilities provided me with the setup and the right atmosphere to persevere through the difficult times.

Declaration

I hereby declare that this written submission is a true and accurate representation of my own ideas, expressed in my own words. In instances where I have incorporated the ideas or words of others, I have taken care to properly cite and reference the original sources. Furthermore, I affirm that I have adhered to the highest standards of academic honesty and integrity in the preparation of this submission, and have not engaged in any form of misrepresentation, fabrication, or falsification of ideas, data, facts, or sources.

In addition, I declare that all software used in the preparation of this submission is licensed under an open-source license. Open-source software is software that is freely available for use, modification, and distribution by anyone. It is developed collaboratively by a community of developers who share a common goal of creating high-quality software that meets the needs of its users. By using open-source software in my work, I am supporting the principles of transparency, collaboration, and community-driven development that are at the heart of the open-source movement.

I understand that any violation of these principles will be subject to disciplinary action by the Institute and may also result in legal consequences from the original sources if proper citation or permission has not been obtained. I take full responsibility for the contents of this submission, and I am committed to upholding the values of academic integrity and ethical conduct in all my scholarly pursuits.

Abstract

The report outlines the development process of a microfinance banking application, a project spurred by the increasing demand for digital financial services in remote and underserved areas. The initiative was inspired by personal observations of financial challenges faced by small business owners in India. The report maps out the project's objectives, which pivot around offering a cloud-based and open-source solution as a server-client web application, fortified with robust user verification and transactional capabilities. Leveraging Django's principles and PostgreSQL's features, the project adapts a user-centric approach, encapsulating user stories to carve out the foundational requirements. The report further unfolds the project's journey, highlighting the methodology adopted, technology stack utilized, and the step-by-step implementation strategy for the most vital section. The report also sheds light on the results and evaluation segment before paving the way for potential future developments. A reflective note is added, encapsulating the learnings and experiences during the project lifecycle.

Table of Contents

Table of Figures

Table of Tables

1. Introduction

In today's world, there is an increasing demand for digital financial services that can reach more people in remote and underserved areas. One way to meet this demand is by developing a microfinance banking app that can be accessed via mobile devices or other affordable handheld devices. This app will provide users with a convenient and secure way to manage their finances, helping them to achieve greater financial stability and independence. Microfinance stands as a pivotal banking service catering to the financial needs of marginalised individuals and underserved groups who would otherwise remain devoid of access to formal financial avenues.

The rationale behind delivering microfinance services to the unemployed or those with meagre incomes is rooted in the fact that conventional financial institutions often fail to accommodate individuals crippled in poverty or possessing limited monetary resources. Despite their exclusion from mainstream banking services, individuals subsisting on as little as £10 per day demonstrate a persistent inclination towards saving, borrowing, securing credit, and obtaining insurance. This often pushes them towards informal networks like friends, family, and high interest charging loan sharks. In this context, microfinance emerges as a safe environment, enabling individuals to access modest yet sensible business loans while adhering to ethical lending norms. Though the global footprint of microfinance is widespread, its predominant impact is observed in developing nations such as Bangladesh, Cambodia, India, Afghanistan, the Democratic Republic of Congo, Indonesia, and Ecuador (Williams & Nguyen, 2021).

Notably, a substantial segment of microfinance initiatives is dedicated to uplifting women, recognising their significant role in these economies (Garcia & Martinez, 2017). Microfinance entities engender a plethora of interventions ranging from fundamental services like basic checking and savings accounts to seeding initial capital for nascent entrepreneurial ventures. Microfinance embodies a beacon of financial inclusion, casting its transformative influence on the lives of those typically relegated to the economic fringes.

The underlying aspiration of microfinance is to foster self-sufficiency among impoverished communities, thereby presenting them with an avenue to transcend their economic challenges. The project aims to develop a web-based client-server payment and loan system that facilitates seamless financial transactions and management. This system would be accessible through browsers, making it easy for clients and administrators to access it from various devices without the need for specialized software.

By leveraging the power of technology, this microfinance banking app has the potential to transform the lives of countless individuals and small businesses. By providing them with access to essential financial services, the app will empower them to take control of their financial futures and achieve their goals. Whether they are looking to start a new business, invest in their education or healthcare, or simply manage their daily expenses more effectively, this app will provide them with the tools they need to succeed. By transitioning from traditional pen-and-paper methods to a digital platform, small entrepreneurs can start their own microfinance banks and provide essential financial services to their communities.

2. Aims and Objectives

Before delving into the intricate details of the system components and functionalities, let me first provide an overview of the core objectives and targets of the project. The initiative is aimed to create a comprehensive financial management platform that combines with seamless user authentication processes with personalised data storage solutions.

In this introductory glimpse, it's worth noting that the platform intends to incorporate critical features such as Know Your Customer (KYC) forms and integrations with credit card and transaction facilities within the bank for initial development. The financial integrity with user convenience and email verification and to create a platform where users can manage their financial activities with confidence and ease, all within a singular, intuitive client-server environment.

As we progress, we will elaborate on the technical aspects and functionalities in greater detail. At this stage, it is important to outline the core aims and objectives of the client server application. It is designed to be deployed on any cloud platform. This strategic choice promises benefits such as scalability, security, and reliability, not to mention a cost-effective approach by the "pay-as-you-use" model and open-source software stack.

1. **Comprehensive User Management:**

   Objective: Create a secure and efficient system for user registration, authentication, and management.
   - The system uses email as the primary identification method.
   - Account statuses include active, pending, and inactive.
   - User personal details, such as marital status and gender, can be stored.
   - A Know Your Customer (KYC) form is present for verification of users.

2. **Core Financial and Transactional Capabilities:**

   Objective: Provide users with a platform to conduct various financial operations and maintain their financial data.
   - The system supports different transaction types, such as transfers, withdrawals, and payment requests.
   - Transaction statuses include failed, completed, pending, and processing.
   - Credit card management functions are present. Users can add or manage their cards for transactions.

3. **Deployment and Custom Domain Name:**

   Objective: Streamline application deployment and provide a user-friendly access point through a custom domain.
   - The application is deployed on a cloud platform, ensuring scalability, resilience, and high availability.

- Custom domain integration for the platform's brand image, making it easily recognizable and accessible.

2.1 Work Plan

To ensure the systematic and efficient development of the client-server application, we have devised a detailed six-sprint work plan spanning twelve weeks. This agile approach allows for iterative development, regular feedback incorporation, and a structured progression from initial research to the final product submission. Each sprint, lasting two weeks, focuses on specific milestones, ensuring that every phase of the project is given due attention. The following breakdown outlines the tasks, objectives, and deliverables for each sprint, providing a clear roadmap for the project's lifecycle.

Sprint 1 (Weeks 1-2): Background Research & Initial Setup

- Conduct background research on financial management platforms and similar projects.
- Identify and document project requirements, priorities, and potential challenges.
- Set up the development environment, including version control, database setup, and tool integrations.

Sprint 2 (Weeks 3-4): Core Development - User Management & Authentication

- Develop the custom user model with email as the primary identifier.
- Implement user registration, login, and email verification functionalities.
- Start working on the KYC form and related backend logic.

Sprint 3 (Weeks 5-6): Core Development - Financial Management & Operations

- Develop features related to credit card management.
- Implement transactional capabilities, including transfers, refunds, and payment requests.
- Incorporate transaction status tracking and user notifications.

Sprint 4 (Weeks 7-8): Interface Development & Report Writing

- Design and develop core views like "sign-in" and "index page".
- Refine the user interface based on best practices.
- Begin drafting the project report, detailing methodologies, software stack used, challenges faced, and solutions implemented.

Sprint 5 (Weeks 9-10): Testing, Debugging & Finalizing Report

- Conduct tests and integration tests on various functionalities.
- Address and fix identified bugs and issues.
- Finalise the project report, incorporating insights from the testing phase.

Sprint 6 (Weeks 11-12): Reflective Essay & Final Touches

- Write a reflective essay, discussing personal growth, challenges, learnings, and the overall development experience.
- Prepare for project submission, ensuring all deliverables are well-organized and documented.

3. Background

When I was living in India, I noticed a particular problem. My uncle owned a small mobile phone shop and interacted with individuals working in blue-collar jobs, such as taxi drivers, rickshaw pullers, conductors, and electricians. These individuals often lacked formal education and were financially illiterate. Many did not have bank accounts or proper identification, earned daily wages, and had little opportunity to save money. As a result, it was difficult for them to obtain loans to improve their financial situation. They often fell victim to high-interest loans from loan sharks or had to rely on family and friends for support. My uncle wanted to start a business that would provide small loans to these individuals, with daily payments and low interest rates tailored to their needs. This would help them avoid exploitation and improve their financial situation. By providing access to affordable credit, my uncle hoped to uplift these individuals and help them achieve greater financial stability.

However, there were several challenges that my uncle faced while trying to implement this business model. Firstly, all transactions were recorded using pen and paper, which made it difficult to keep track of payments and defaults. Secondly, if an individual defaulted on their loan, someone had to physically go and collect the daily amount from them. Thirdly, there were no proper guidelines or laws in place to safeguard the bank in case of defaults. Finally, the business model lacked scalability, making it difficult to expand the business and reach more people in need.

Despite these challenges, my uncle remained determined to help these individuals and provide them with access to affordable credit. He believed that by doing so, he could make a positive impact on their lives and help them achieve greater financial stability. Hence, the need for open-source banking software to facilitate business expansion and maintain accurate records of data. The adoption of open-source technologies is driven by their cost-effectiveness and ease of distribution to multiple small communities.

Before designing and developing the application, I conducted research on currently available software that provided a similar service and their limitations and drawbacks. I have found two existing alternatives which seemed the most viable option going forward in my research in developing a solution with skills that I have acquired.

3.1 Existing Alternatives

3.1.1 – Mambu

Mambu is a German software company founded in Berlin and headquartered in Amsterdam. It provides infrastructure for banks and financial service providers according to the software as a service (SaaS) model. In December 2021, Mambu reached a company valuation of $5.3 billion, making it one of the unicorns of Germany (Mambu, 2021).

Mambu was founded in 2011 by Frederik Pfisterer and Eugene Danilkis, who had developed technical infrastructure for microfinanciers in Latin America and Africa. Mambu's cloud-native lending engine enables banks, fintechs, retailers, corporates, and others to build a variety of loan offerings tailored to customer needs. From embedded finance, buy now pay later and mortgages to SME lending and purchase financing (Mambu, 2021).

Mambu's customers include fintech unicorns, top-tier banks, and everyone in between. They work with pioneers of composable – and digital-first financial services more generally – who come in all shapes and sizes (Mambu, 2021). Mambu's platform is used by over 100 microfinance organizations, in 26 countries worldwide (Mambu, 2021).

3.1.1.1 – SWOT Analysis

Strength:

1. Mambu is a market-leading SaaS banking platform that provides financial institutions of all sizes with the agility to rapidly design, launch, service and scale their banking and lending portfolio.
2. Mambu has a strong API suite that makes integrations with third-party systems easy.
3. Mambu has a quick and easy installation process, allowing new environments to be set up in less than a day.
4. Mambu is priced competitively according to the market standards and has delivered year-on-year growth of more than 120% in Q3 of 2021.

Weaknesses:

1. Migration of existing loans is difficult due to the inability of Mambu to import historical transactions from a point in time.
2. The inability to set up a payment plan in Mambu is a notable limitation, especially in the microfinance sector where structured and predictable repayment schedules are essential for clients managing financial obligations with limited resources.
3. Adjustments to loan terms force clients into a new loan agreement, complicating the process and hindering flexible financial management.

Opportunities:

1. The banking and securities enterprise software market is over £80 billion in size and experiencing double-digit annual growth.
2. The tools are at hand for incumbents to address the challenges posed by legacy platforms, including the emergence of new generation cloud-native core banking platforms such as Mambu.

3. Mambu could further accelerate innovation in its next-generation platform and expand its global footprint.

Threats:

1. Competition in the banking industry is intensifying, with neo-banks winning market share and serving customers at around one third of the cost of traditional banks.
2. FinTech's are targeting lucrative niches in the value chain, while big tech players with large customer bases pose a real threat.
3. Incumbent banks are increasingly concerned about the limitations of their own core architectures and their relatively slow pace of change.

3.1.2 – Finastra Fusion Microfinance

Finastra is a leading financial technology company that provides a range of software and services to financial institutions worldwide. One of its products is Fusion Microfinance, which is designed to help microfinance institutions (MFIs) manage their operations more efficiently (Finastra, n.d.).

Fusion Microfinance is a comprehensive solution that covers all aspects of microfinance operations, from loan origination and disbursement to repayment and reporting. It is designed to help MFIs streamline their processes, reduce costs, and improve the quality of their services (Finastra, n.d.).

One of the key features of Fusion Microfinance is its flexibility. The solution can be customized to meet the specific needs of each MFI, allowing them to tailor the system to their unique business processes. This flexibility also extends to the product's deployment options, as it can be implemented on-premises or in the cloud (Finastra, n.d.).

3.1.2.1 SWOT Analysis

Strengths:

1. Fusion Microfinance is designed specifically for microfinance institutions, catering to their unique needs and challenges. This specialization led to more tailored features and better alignment with microfinance processes.
2. The software offers a wide range of features including client management, loan origination, repayment tracking, and portfolio management. This comprehensive suite of tools help streamlines operations and improve efficiency.
3. Fusion Microfinance likely includes risk assessment and management features that can help microfinance institutions analyse and mitigate potential risks, such as defaults and non-repayments.
4. Finastra is a well-established financial technology company, suggesting that Fusion Microfinance is likely built with scalability in mind. This is crucial for microfinance institutions aiming to grow their operations over time.

Weaknesses:

1. While comprehensive features are beneficial, they might also contribute to software complexity. This could potentially lead to a steeper learning curve for new users and require more training and support.
2. While being a specialized solution is a strength, it could also lead to limitations in terms of customization. Microfinance institutions might have unique processes that cannot be easily accommodated within a pre-built solution.

Opportunities:

1. The microfinance industry is experiencing growth due to increasing financial inclusion efforts worldwide. Fusion Microfinance could tap into this expanding market by offering a solution that caters to the specific needs of microfinance institutions.
2. Finastra could explore partnerships with other fintech companies or financial institutions to offer a more holistic suite of services. Integration with complementary technologies could enhance the value proposition of Fusion Microfinance.

Threats:

1. The microfinance industry is subject to regulatory changes that can impact operations and software requirements. Fusion Microfinance must remain adaptable to evolving regulatory landscapes in various regions.

3.1.2 Conclusion

Mambu and Finastra Fusion Microfinance stand as substantial contenders in the financial technology sector, each bringing to the table a wealth of features and capabilities tailored to the needs of financial institutions and microfinance entities respectively. In the context of our project, implementing a SaaS solution was a strategic decision, driven by the desire to provide scalable, cloud-based solutions that can quickly adapt to the evolving needs and demands of the market. The SaaS model facilitates seamless updates and maintenance, reducing the burden on IT resources and allowing for a more agile and responsive service delivery.

It is important to note that being SaaS products, they inherently carry a significant limitation - a continual financial commitment due to perpetual payment structures. These platforms exhibit a pronounced vendor service lock-in, a scenario where customers become heavily dependent on the vendor's infrastructure and services. This not only makes the process of transitioning to a different service provider cumbersome but may also pose challenges in data migration and integration with other systems. While the offerings are robust and cater to a growing market, potential adopters should carefully weigh the long-term implications of service lock-in, and the continuous financial commitments associated with utilising these SaaS products.

3.2 Limitation and Challenges of Microfinance Banking

Challenges faced by microfinance banking is inappropriate laws and regulations. In some countries, the legal and regulatory framework may not be conducive to the growth and development of microfinance institutions. This can make it difficult for these institutions to operate effectively and efficiently and can limit their ability to reach those who need their services the most.

Profitability is another challenge faced by microfinance institutions. To be sustainable, these institutions need to be able to generate sufficient revenue to cover their costs. However, this can be difficult, particularly in markets where competition is intense, and margins are thin. Default risk inherited from borrowers is another challenge faced by microfinance institutions. These institutions lend money to individuals who may not have a steady source of income or collateral. This means that there is a risk that these borrowers may default on their loans, which can have serious consequences for the institution.

Less awareness about the microfinance program by the borrowers is another challenge. For microfinance programs to be effective, borrowers need to understand how they work and how they can benefit from them. However, many borrowers may not have access to this information, which can limit the effectiveness of these programs.

The infrastructure gap in developing countries is another challenge faced by microfinance institutions. In many developing countries, the infrastructure needed to support microfinance programs may not be in place. This can make it difficult for these institutions to reach those who need their services the most. Internal technological constraints are another challenge faced by microfinance institutions. These institutions need to have access to modern technology to operate effectively and efficiently. However, many smaller institutions may not have the resources needed to invest in this technology.

Small finance banks also face challenges in building a low-cost liability portfolio, managing technology, and balancing regulatory compliances. These challenges need to be addressed for small finance banks to operate effectively and efficiently.

3.3 Ethical Implication and Fraud Instances

Loan fraud is a serious issue in India, with many victims suffering financial and emotional hardship as a result (BBC News, 2022). One type of loan fraud that has become increasingly common in India is the digital loan scam. These scams lure victims in with an easy qualification process, often through a mobile app, but then demand exorbitant repayment amounts and use threats and intimidation to extract money from their victims (BBC News, 2022).

Between January 1, 2020, and March 31, 2021, a study by the Reserve Bank of India (RBI) identified 600 illegal lending apps. During that period, Maharashtra state recorded the highest number of complaints relating to lending apps, with 572 reported to the RBI (BBC News, 2022). These apps promise hassle-free loans and quick money, but often result in the victim’s phone being hacked, their data being stolen, and their privacy being compromised (BBC News, 2022).

Banks operating in India have also reported significant amounts of loan fraud. As of March 31, 2021, banks in India reported fraud of Rs 4.92 trillion, representing nearly 4.5% of the total bank credit (Business Standard, 2021).

It is important for individuals to be aware of the risks associated with loan fraud and to take precautions to avoid becoming victims. This can include being cautious when dealing with unfamiliar lenders or loan apps, verifying the legitimacy of lenders before providing personal information or money, and being wary of offers that seem too good to be true.

4 Requirements

In this section, we will explore the foundational requirements for our project. We'll begin by examining user stories, which offer insights into the software's desired features from the end-user's viewpoint. Described in everyday language, these stories give us a glimpse into the anticipated functionality and value. We'll also delve into the acceptance criteria, defining the conditions that signify a user story's completion. This ensures a shared understanding between developers and stakeholders. Lastly, our focus will shift to the software product quality requirements, emphasizing the crucial standards the software must meet, encompassing attributes like performance, security, and usability.

4.1 User Story and Acceptance Criteria

These two user stories will illustrate the potential use case of microfinance banking.

User Story 1 – “I’m a daily wage earner, and I want to avail a small credit to fix my car engine which broke down, and I want to pay the credit amount daily with significantly less Interest Rates.”

In the first story, a taxi driver needs a small credit to fix his car engine. With a microfinance banking app, he could easily apply for and receive a credit with low interest rates and flexible repayment options, such as daily payments. This would allow him to quickly get back on the road and continue earning a living.

User Story 2 – “I’m an entrepreneur who wants to create a small-scale microfinance bank in my local area to provide cheap credits to customers who wish to avail a credit in cash or direct transfer”.

In the second story, an entrepreneur wants to start a small-scale microfinance bank in his local area. With a microfinance banking app, he could easily set up and manage his bank, providing affordable credits to customers in need. This would not only help the entrepreneur grow his business, but also provide a valuable service to his community.

Both stories demonstrate how microfinance banking can empower individuals and small businesses, providing them with access to essential financial services and helping them achieve their goals.

Acceptance Criteria for Admin/Entrepreneur:

1. The system should have a dashboard for admin to view all users.
2. The system should have the ability to view KYC information uploaded and make account active or in-active.
3. The system should have the ability to create users in case a user does not want to use the web application.
4. The system should be able to update user password, change PIN when requested.
5. The system should have a notification stream which allows the admin to view all changes in the system.
6. The system should have the ability to add user to groups of elevated permissions to perform certain transactions or roll back.
7. The system should have the ability to generate credit card with certain amount for user who has requested credit from the bank.

Acceptance Criteria for User:

1. The system should allow users to create account using sign-up page with valid email and password.
2. The system should allow users to login using their correct credentials.
3. The system should allow user to upload their Profile Photo, Identification Image, and Signature for KYC verification.
4. The system should allow the user to apply for credit card after account verification.
5. The system should allow the user to make transaction from their account to other users within the bank and to the bank.
6. The system should have security to prompt user to ask for security key or PIN while making transaction.
7. The system should have transaction history for the user to check.
8. The system should have notification alert when they receive payment.

4.2 Software Product Quality Requirements

The concept of software product quality requirements has its roots in the broader field of quality assurance and management. As software development emerged as a significant discipline in the mid-20th century, the importance of ensuring software quality became evident. Early software was riddled with bugs and lacked standardization, leading to the recognition of the need for quality requirements (Daniel and Galin, 2018).

ISO (International Organization for Standardization) and IEC (International Electrotechnical Commission) have jointly developed standards that focus on software product quality. The ISO/IEC 25010:2011 is a notable standard that defines and categorizes software quality attributes. This standard evolved from earlier ones, such as ISO/IEC 9126, and provides a comprehensive framework for understanding and evaluating software product quality.

4.2.1 Functional Requirements

1. User Management:
    - Users should be able to register with email verification.
    - Users should be able to log in using their credentials.
    - Users should be able to log out.
    - Users should be able to reset their passwords.
2. Email Verification:
    - Upon registration, users should receive an email verification link.
    - Clicking the link should successfully verify the user’s email.
    - Invalid or expired links should provide appropriate error messages.
3. Admin Interface:
    - Admins should have a customized dashboard (Jazzmin).
    - Admins should be able to perform CRUD operations on all models.
    - Admins should be able to manage user accounts.
4. Deployment:
    - The application should be deployable on cloud platforms like Railway.
    - Database migrations should be easily executable post-deployment.

4.2.2 Non-Functional Requirements

1. Performance:
    - The application should respond quickly to user actions.
    - Database operations should be optimized and not introduce bottlenecks.
2. Security:
    - User passwords should be stored securely (hashed and salted).
    - Email verification tokens should be securely generated.
    - The application should guard against common web vulnerabilities (SQL injection, CSRF, XSS, etc.).
3. Usability:
    - The user interface should be intuitive and user-friendly.
    - Error messages should be clear and guide users on what actions to take.
4. Availability:
    - The application should aim for high uptime.
    - Critical operations (like registration and login) should be reliable.
5. Scalability:
    - The application should handle an increasing number of users and data.
6. Maintainability:
    - The codebase should be well-organized and modular.
    - Proper documentation should be provided for developers.
7. Compatibility:
    - The application should be compatible with modern web browsers.
    - The application should be responsive and work on both desktop and mobile devices.

5 Approach and Methodology

In this section, I'll discuss the software lifecycle methodologies employed in our project, along with the chosen technology stack. We'll delve into the advantages and disadvantages of this stack, and we will conclude by explaining my rationale for selecting this specific set of technologies for the development of this project.

5.1 Agile Development and Notion as Methodology for Product Creation

Agile development stands as a cornerstone in my project's methodology, fostering a rapid and adaptive approach to product creation. This iterative strategy, rooted in the realms of software engineering, emphasizes collaboration and customer engagement, making it particularly potent in the dynamic market environments that manufacturing companies often navigate (Singh et al., 2021).

In the context of the project, agile development translates into a flexible, yet focused approach to building a robust financial management system. This method allows me to continually adapt to changing requirements, a feature that is essential in the ever-evolving financial sector. It not only ensures speed and efficiency in development but also actively involves stakeholders (here my supervisor) at each stage, fostering a collaborative environment that is catered to real-time market demands (Riesener et al., 2021).

The integration of lean processes enhances the agility of the development approach, promising significant reductions in lead time and thus saving time. This combination of lean and agile methodologies is geared towards overcoming the unique challenges presented in physical systems development, where acceptance and perceived value can vary considerably across different branches (Heimicke, Kaiser, and Albers, 2021).

To facilitate this agile approach, we have chosen Notion as our central tool, a decision inspired by its compatibility with various agile practices including Scrum and Kanban. In the framework of our project, Notion serves as a versatile canvas, streamlining task management and sprint planning. It acts as a vital nexus, housing all project-related documentation, from user stories to intricate technical specifics, enhancing team collaboration and alignment with the project's evolving objectives (Gridfiti, 2023).

By tailoring this agile methodology to the specific demands and context of our project, we aim to create a product that is not only resilient and adaptable but also finely tuned to the needs and expectations of our target audience.

Figure – Sample notion board

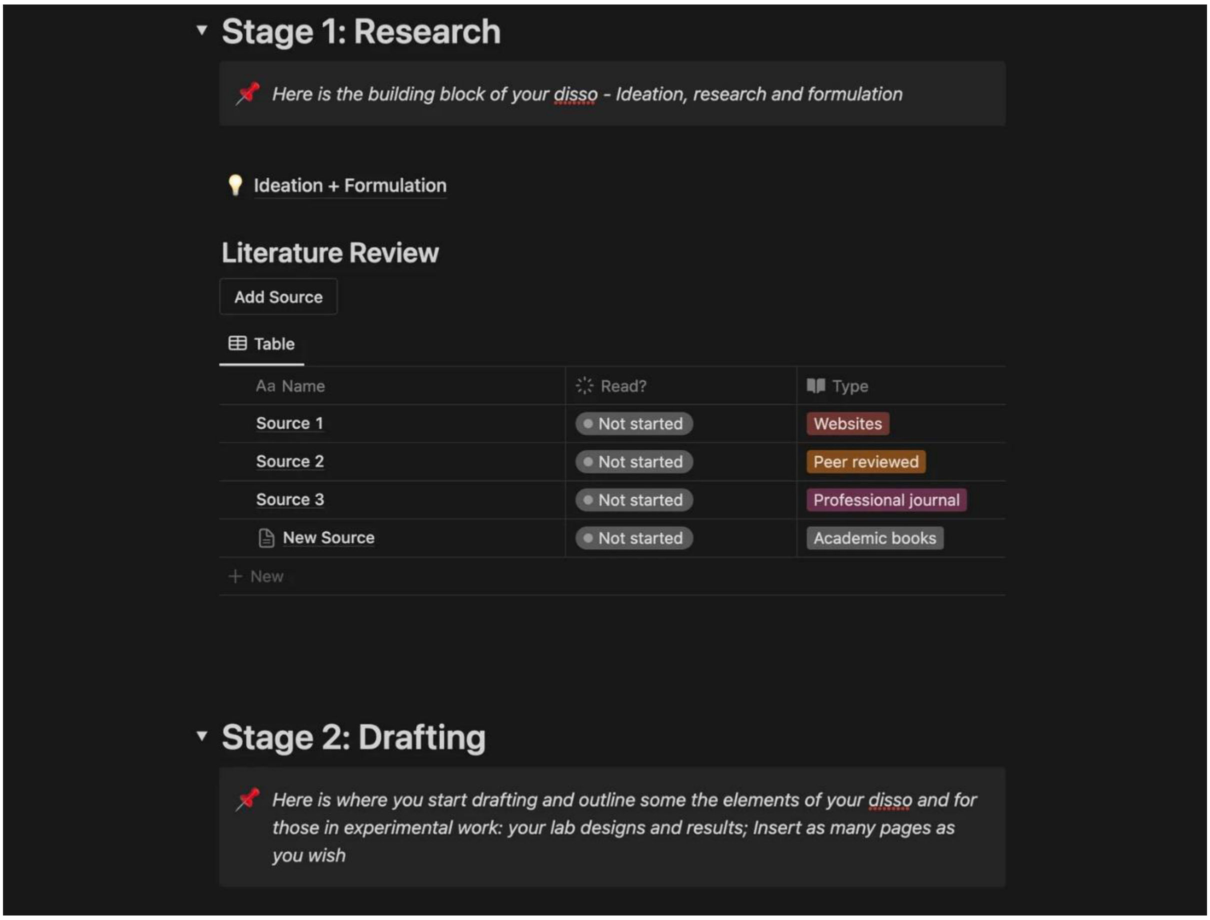


Figure 1 Notion Template for Dissertation

5.2 Technology Stack

5.1.1 – Backend Development- Django

Django was developed by Adrian Holovaty and Simon Willison while they were working at the Lawrence Journal-World newspaper. It was released to the public as an open-source project in 2005 (Django Software Foundation, 2021). Django is based on the "batteries-included" philosophy, which means it provides a wide array of functionalities out-of-the-box. The main goal of Django is to enable developers to build web applications quickly without the need to rebuild common structural elements from scratch.

Benefits:

1. Admin Interface: Django includes a built-in admin interface, which makes it easier to manage content and perform CRUD (Create, Read, Update, Delete) operations (Django Software Foundation, 2021).
2. DRY Principle: Django believes in the "Don't Repeat Yourself" (DRY) principle, which means writing the code once and reusing it, leading to less redundancy and easier maintenance.
3. ORM: Django's Object-Relational Mapping (ORM) allows developers to interact with databases using Python code rather than writing SQL queries (Django Software Foundation, 2021).
4. Batteries Included: Django provides a wide array of built-in features like authentication, URL routing, and template engine, which speeds up the development process.
5. Security: Django emphasizes security and helps developers avoid many common security mistakes, like SQL injection and cross-site scripting (XSS) (Django Software Foundation, 2021).
6. Scalability: Django has been used in building high-traffic websites and can scale to handle millions of users.

Drawbacks:

1. Learning Curve: For beginners, especially those not familiar with Python, Django might present a steep learning curve.
2. Monolithic: Django follows a "monolithic" approach, which can sometimes make it less flexible compared to micro-frameworks like Flask or FastAPI.
3. Performance: While Django is generally fast, certain real-time applications might require a more lightweight and faster framework.
4. Too Much Magic: Django does a lot behind the scenes, which can sometimes be confusing for developers who want more control over the application's behaviour.
5. Not always suitable for small projects: Given the comprehensive features that Django offers; it might be an overkill for smaller, simpler projects.

Conclusion:

Having faced a choice between Django and Node.js, I gravitated towards Django primarily due to my existing expertise in Python. This was bolstered by the fact that Django shares striking similarities with Flask, a framework that formed a significant part of our academic curriculum. Diving into a completely new technology like Node.js, which embraces a set of programming paradigms distinct from what I'm accustomed to, seemed like a potential challenge that could divert focus from the main project. Additionally, Django's longstanding reputation as a robust and reliable framework in the web development sphere gave me the confidence that I was deciding grounded in the experiences of countless developers before me. With its wide array of features and open-source community that has vouched for its capabilities over the years, Django seemed to be the most fitting choice for my web development endeavours.

5.2 Frontend Technology- Bootstrap

Bootstrap was initially developed by Twitter engineers Mark Otto and Jacob Thornton and was released as an open-source project in 2011 (Otto & Thornton, 2021). The framework provides pre-styled components and utilities that facilitate faster and more consistent web development. Over the years, Bootstrap has gained immense popularity among web developers and designers for its ease of use and versatility.

Benefits:

1. Responsive Design: Bootstrap offers a grid system and components that automatically adjust to different screen sizes, ensuring a seamless user experience across devices (Otto & Thornton, 2021).
2. Customizable: Developers can easily customize Bootstrap to fit their project's specific needs, using its extensive list of variables and configurations.
3. Consistency: Originally built to maintain design and functional consistency at Twitter, Bootstrap ensures that elements and components maintain a uniform look and feel across platforms and browsers.
4. Community Support: Being open-source and widely adopted, Bootstrap boasts a large community. This means extensive documentation, frequent updates, and a plethora of third-party plugins and themes.
5. Integration: Bootstrap can be integrated with various other platforms and frameworks, both on the front-end (like Angular or React) and the back end (like Django or ASP.NET).
6. Pre-styled Components: Bootstrap comes with a range of pre-designed components such as navigation bars, modals, and alerts, which speed up the development process.

Drawbacks:

1. File Size: Even though Bootstrap can be customized to include only the required components, the full package can be relatively large, potentially affecting page load times.
2. Similarity in Design: Since many websites use Bootstrap, there's a risk of them looking similar unless developers invest time in customization.
3. Learning Curve: While Bootstrap simplifies web design, newcomers might still face a learning curve, especially if they are not familiar with its grid system or responsive design principles.
4. Overhead: For simple projects, Bootstrap might introduce unnecessary overhead due to its comprehensive features.
5. JavaScript Dependencies: Some of Bootstrap's components rely on JavaScript which might not be ideal for all projects due to potential security risks or to enhance site load speed, especially for users with slower internet connections.

Conclusion:

In the initial stages of the project, several front-end frameworks were under consideration to ensure the optimum balance of aesthetics and functionality for the user interface. Apart from Bootstrap, alternatives like Tailwind CSS caught attention due to their respective strengths in facilitating responsive design and offering customizable components.

However, after a careful evaluation, I chose Bootstrap because of its pre-styled components and seamless integration capabilities. It stood out for its ease of use, allowing for the efficient management of the templates' aesthetics. This feature significantly eased the process of reusing templates for similar functions, enabling a focused effort on refining the backend processes, thereby streamlining the development workflow.

5.1.3 – Database – Sqlite3 for Local Development

SQLite is a software library that provides a relational database management system (RDBMS) in a C-language programming library. Unlike traditional database management systems which operate as standalone server processes, SQLite is serverless and requires no configuration (SQLite Development Team, 2021). It offers a simple and lightweight mechanism for persistent data storage, making it a popular choice for embedded database solutions and small to medium applications.

Benefits:

1. Serverless and Self-Contained: SQLite doesn't require a server to run, and its operations are fully contained in a single disk file (SQLite Development Team, 2021).
2. Lightweight: With a small binary size, SQLite is well-suited for applications where minimal overhead and resource usage is critical, such as mobile apps or embedded systems.
3. Zero Configuration: SQLite doesn't require any setup, installation, or configuration, making it easy to integrate with applications.
4. Public Domain: SQLite is in the public domain, which means it can be used for any purpose, including private, commercial, or academic, without any restrictions.

Drawbacks:

1. Scalability: While SQLite is efficient for small to medium-sized applications, it may not be suitable for large-scale applications or high-concurrency scenarios.
2. Write Operations: SQLite locks the entire database during write operations, which can be a bottleneck in scenarios with frequent concurrent writes (SQLite Development Team, 2021).
3. Lacks Some SQL Features: SQLite might not support some advanced SQL features or extensions that other databases offer.
4. Not Ideal for Large BLOBs: While SQLite might not be the best choice for managing large BLOBs (binary large objects) such as videos or high-resolution images, it is worth noting that such data types are generally better stored directly on the filesystem to optimize performance and organization. This approach promotes a more efficient and structured data management system, minimizing the potential strain on the database.

Conclusion:

For my local development and testing, I chose Sqlite3 due to its serverless and self-contained nature. It was ideal for code testing, allowing for easy data insertion, modification, and updates. Given that scalability wasn't a requirement during development, Sqlite3 paired well with tools like DB Browser and SQL workbench, facilitating my work on Django's models.py. My familiarity with Sqlite3, having been introduced to it during our coursework for both database management and SQL query writing, made it a natural selection. Moreover, its usage in our "Fundamentals of Programming" course, where we worked with Flask, further solidified my preference for it.

5.1.4 – Database – PostgreSQL for Deployment

PostgreSQL has its roots in the Ingres project at the University of California, Berkeley. It has over 30 years of active development, making it one of the most advanced database systems. PostgreSQL is not just a relational database but also incorporates features of object-relational databases, which allows it to store complex data structures and support custom data types (PostgreSQL Global Development Group, 2021).

Benefits:

1. Extensibility: Users can define their own data types, operators, and functions. This extensibility has led to the development of features like PostGIS for spatial databases.
2. Active Community: Being open-source, PostgreSQL boasts a strong and active community that contributes to its continuous development, offers support, and develops a range of third-party add-ons and tools.
3. Cross-Platform: PostgreSQL can run on various platforms including Linux, Windows, macOS, and more.
4. ACID Compliant: PostgreSQL is fully ACID compliant, ensuring data reliability in case of system failures.

Drawbacks:

1. Operational Overhead: Compared to simpler database solutions, PostgreSQL require more operational overhead in terms of maintenance and tuning.
2. Less Commercial Support: While the community support is robust, PostgreSQL does not have as much commercial support or proprietary tools as some other commercial RDBMS systems.

Conclusion:

When considering the database options for deployment on railway.app, various alternatives were evaluated including MySQL, MongoDB, and Redis, each bringing their own set of features and community support. These alternatives were known for their reliability and widespread use in various types of projects, from small to large scale.

However, I ultimately selected PostgreSQL as the primary deployment database, swayed by its open-source foundation and the robust support from a vibrant and dedicated community. A significant factor that influenced this choice was railway.app's exceptional ability to bridge the gap between sqlite3 and PostgreSQL, which guarantees fluid and complication-free data migrations. This seamless transition not only simplifies the database migration process but also ensures data integrity and security are maintained.

The user-friendly nature of the railway.app platform was a decisive factor. It simplifies complex tasks such as configuring connection parameters, engines, hosts, and ports, transforming them into straightforward procedures managed through a simple text file. Moreover, the platform offers an adept handling of .env files, a critical feature that not only safeguards sensitive information but also fortifies the overall security framework, thus making PostgreSQL an optimal choice for this project.

5.1.5 – Version Control- Git and GitHub

Git was developed by Linus Torvalds in 2005 for the development of the Linux kernel, with the help of other developers. Unlike centralized version control systems, Git provides each developer a local copy of the entire development history, and changes are copied from one such repository to another (Torvalds & Hamano, 2021). This means that users can make changes locally and then push those changes to remote repositories.

GitHub, founded in 2008 by Chris Wanstrath, P. J. Hyett, Tom Preston-Werner, and Scott Chacon, is a platform that offers distributed version control and source code management (SCM) functionality of Git combined with its own features (GitHub, Inc., 2021). GitHub allows multiple users to collaborate on projects, making it easier to manage and track work on shared repositories.

Benefits:

1. Collaboration: GitHub makes it easy for multiple users to collaborate on a single project. Features like pull requests allow users to contribute to projects, making open-source collaboration highly efficient (GitHub, Inc., 2021).
2. Integration: GitHub offers integration with numerous tools and platforms, enhancing the development workflow. This includes continuous integration tools, project management tools, and cloud services.
3. GitHub Actions: A feature that allows automation of software workflows, making it easier to test, build, and deploy applications directly from within GitHub.

Drawbacks:

1. Complexity for Beginners: Git, as a version control system, has a steep learning curve, and while GitHub adds a layer of user-friendliness, beginners might still find it challenging.
2. Centralization Concerns: Even though Git is decentralized, GitHub itself is a centralized platform. This means potential risks of service outages affecting all users.

Conclusion:

I selected GitHub as my primary tool for version control, given the advantages it offers. Cloud service providers, including AWS, GCP, Azure, and Railway, frequently employ it, underscoring its ubiquity and trustworthiness. GitHub facilitates effective code management, allowing for separate development branches where testing can be executed. The main or master branch can then serve as the deployment or production hub. New functionalities can be added and tested in these branches and once deemed stable, merged into the main branch. Another feature of GitHub is its provision for private repositories. This feature is useful for safeguarding confidential files, such as .env files, adding to security.

5.1.6 – Deployment – Railway.app

Railway.app is a cloud-based platform that helps developers ship software quickly and easily. It is designed to handle code written in any language and for projects of any size. The platform takes the complexity out of shipping software by providing tooling that extends your app with plugins and variable management, so environments are the same live as local (Railway, n.d.).

Benefits:

1. Simplified Deployment: Railway aims to make the deployment process straightforward, allowing developers to focus on coding rather than the intricacies of deployment.
2. Integrated Environment: Railway provides an environment that integrates various services and tools, which can simplify the process of connecting and managing different parts of an application.
3. Scalability: Railway offer easy scalability options, letting applications grow without significant reconfigurations.

Drawbacks:

1. Learning Curve: Every platform introduces its own set of tools, configurations, and workflows. New users might face a learning curve understanding and navigating these nuances.
2. Limitations: While platforms like Railway offer a range of tools and integrations, they might not support every tool or service a developer wants to use. This can lead to limitations in how applications are developed or deployed.
3. Cost: As applications grow, hosting and service costs might increase.
4. Platform Dependency: Relying heavily on a specific platform can lead to a form of "vendor lock-in", where migrating to another platform in the future might become challenging.

Conclusion:

I gravitated towards railway.app due to its user-centric design and straightforward approach, which made it developer-friendly. As a student, its free tier starter plan further piqued my interest, offering an affordable entry point. My intention was to bridge the knowledge gap, leveraging a cloud platform to showcase a functional web application. This would effectively illustrate my proficiency in deploying web applications on the cloud, complete with a relevant domain name. An added bonus with railway.app is its capability to handle most of the infrastructure intricacies, sparing me from unnecessary tinkering. The skills and insights I've gained from this experience are invaluable. They can be seamlessly applied to platforms like Heroku or AWS, underscoring the transferable nature of cloud deployment knowledge.

5.1.7 – SendGrid for Email Verification

SendGrid, which was acquired by Twilio in 2018, is a cloud-based email delivery service that assists businesses in sending transactional and marketing emails. SendGrid's platform offers solutions to help companies with their email outreach, whether it's shipping notifications, password resets, promotional emails, or other forms of communication.

Benefits:

1. Scalability: SendGrid is built to handle large-scale email sending, making it an ideal choice for businesses of all sizes.
2. Reliability: With its robust infrastructure, SendGrid boasts high deliverability rates. This ensures that your emails reach the inboxes of your recipients.
3. Detailed Analytics: SendGrid provides detailed metrics about your emails, including open rates, click-through rates, and more. This helps businesses refine their email strategies.

4. Flexible APIs: SendGrid offers robust APIs that allow developers to integrate email sending capabilities directly into their applications or websites.
5. Email Template Designs: Users can choose from a variety of pre-designed templates or create their own using SendGrid's design editor.
6. Security: SendGrid offers features like two-factor authentication and domain authentication to enhance security.

Drawbacks:

1. Learning Curve: Some users find SendGrid's interface and features a bit overwhelming initially, especially if they are new to email marketing.
2. Pricing: While SendGrid offers a free tier, it has limitations. Some users feel that the higher-tier plans can be expensive, especially for small businesses or startups.
3. Deliverability Issues: Some users have reported occasional deliverability issues, where emails land in the spam folder of recipients.
4. Support Concerns: Some customers have expressed dissatisfaction with SendGrid's customer support, stating that responses can sometimes be slow or not as helpful as expected.
5. Limitations in Design Editor: While SendGrid offers an email design editor, some users find it less intuitive or flexible compared to other platforms.

Conclusion:

I selected SendGrid as my preferred platform for several strategic reasons that align with the security objectives of my project. One of the primary advantages was the opportunity to leverage their network to dispatch emails without incurring any costs during the initial development phase. This was a pivotal resource in crafting a web application where email verification is a central feature.

Utilizing SendGrid not only allowed me to create email verification system but also significantly bolstered the security of the application. It provides an added layer of protection against potential spam bot incursions, helping to prevent unauthorized access and safeguard user data. By incorporating SendGrid's advanced email verification features, the application can effectively filter out spam and phishing emails, thereby enhancing the security posture of the entire system.

The integration of SendGrid showcases my proficiency in melding external APIs into my application seamlessly, a skill that is crucial in modern web development for building secure and efficient systems. My supervisor emphasized the importance of this approach, noting that leveraging trusted external platforms like SendGrid can enhance the security infrastructure by reducing vulnerabilities associated with email-based attacks. This, in turn, ensures a safer and more secure user experience, reflecting a proactive stance towards protecting user data and maintaining the integrity of the application.

5.1.8 Amazon S3 Bucket for Storage

Amazon S3 (Simple Storage Service) is a scalable object storage service offered by Amazon Web Services (AWS). It is designed to store and retrieve any amount of data from anywhere on the web. It is widely used by organizations of various sizes for backup and recovery, content distribution, big data analytics, and many other applications. This report aims to elucidate the benefits and drawbacks of utilizing Amazon S3 (Amazon Web Services, n.d).

Benefits:

1. Durability and Availability: Amazon S3 is designed to deliver 99.999999999% (11 9's) durability over a given year, ensuring that the data is highly reliable. It also guarantees 99.99% availability, meaning that the data is accessible whenever needed.
2. Scalability: Amazon S3 can handle an unlimited amount of data, which can range from a few bytes to several terabytes, thus offering excellent scalability options to meet the growing demands of businesses.
3. S3 offers robust security features including data encryption, integrated access controls, and multi-factor authentication to protect sensitive information from unauthorized access.
4. Wide Range of Use Cases: From serving website content to hosting mobile apps and big data analytics, S3 can accommodate a wide range of use cases, making it a versatile choice for various business needs.

Drawbacks:

1. Complexity of Pricing: The pricing structure of Amazon S3 can be complex to understand due to various factors such as data transfer fees, request costs, and different storage class pricing. This can lead to unexpected costs if not carefully managed.
2. Learning Curve: New users may face a steep learning curve, especially if they are not familiar with AWS or cloud computing concepts. Understanding the various features and configurations of S3 might require a significant amount of time and effort.
3. Data Transfer Costs: Data transfer out of S3 to the internet or to a different AWS region can incur additional costs, which can add up quickly for businesses with high data transfer needs.

Conclusion:

I opted for the Amazon S3 bucket as my preferred storage solution, primarily to address the inherent limitations presented by the railway.app, which unfortunately does not retain data formats such as PDFs and images beyond a 48-hour window, subsequently resulting in data loss. To circumvent this challenge, I decided to deploy and manage all static files within a storage bucket. Furthermore, my choice to utilize AWS over other prominent vendors in the market, such as Azure or Google Cloud, was significantly influenced by my existing familiarity with AWS's system. My past experience with executing projects using this technology instilled a level of comfort and trust in its capabilities, resulting a smooth transition and integration into the current project framework.

6 Design

In this section, we will explore the systematic approach employed in the design of the entire project. This overview aims to demonstrate the foundational elements that underwent the project's blueprint. Central to the design is the use case diagram, which serves as a visual representation of the interactions between the user and the system, the main functionalities and shedding light on various user roles and their corresponding actions. A database schema, detailing the underlying structure of the database. This design shows how tables and fields are connected and helps keep the data accurate while making searches faster. Description of the overall architecture, offering a bird’s eye view of the project, encompassing the high-level components, their interactions, and the architectural patterns adopted.

6.1 Use Case Diagram

Figure 1 demonstrates the use case diagram.

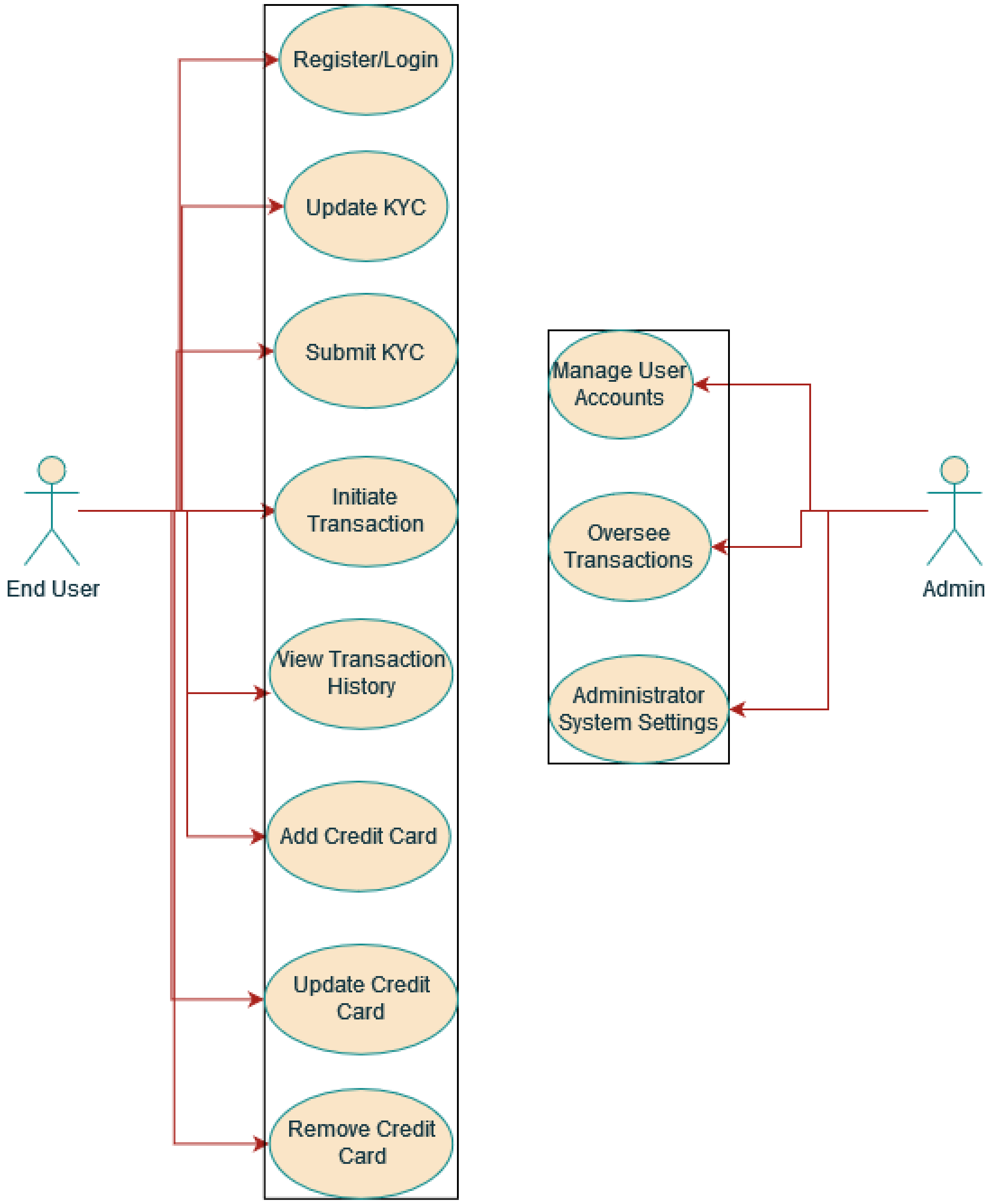


Figure 2 Use Case Diagram

Within the system, there are two primary actors: the **End User** and the **Admin**. The End User, a standard user of the system, can engage in a range of activities such as registering or logging in, initiating transactions, viewing their transaction history, managing their credit card details (including adding, updating, or removing credit cards), and handling their Know Your Customer (KYC) details by submitting them. In contrast, the admin, embodying the role of an administrative user or system overseer, has broader responsibilities. They oversee managing all user accounts, overseeing the entirety of transactions, and configuring system-wide settings. Collectively, this diagram underscores the distinction in functionalities accessed by regular users versus system administrators, emphasizing the latter's broader control and oversight capabilities.

6.2 System Architecture

Figure 2 represents a bird's eye view of System Architecture

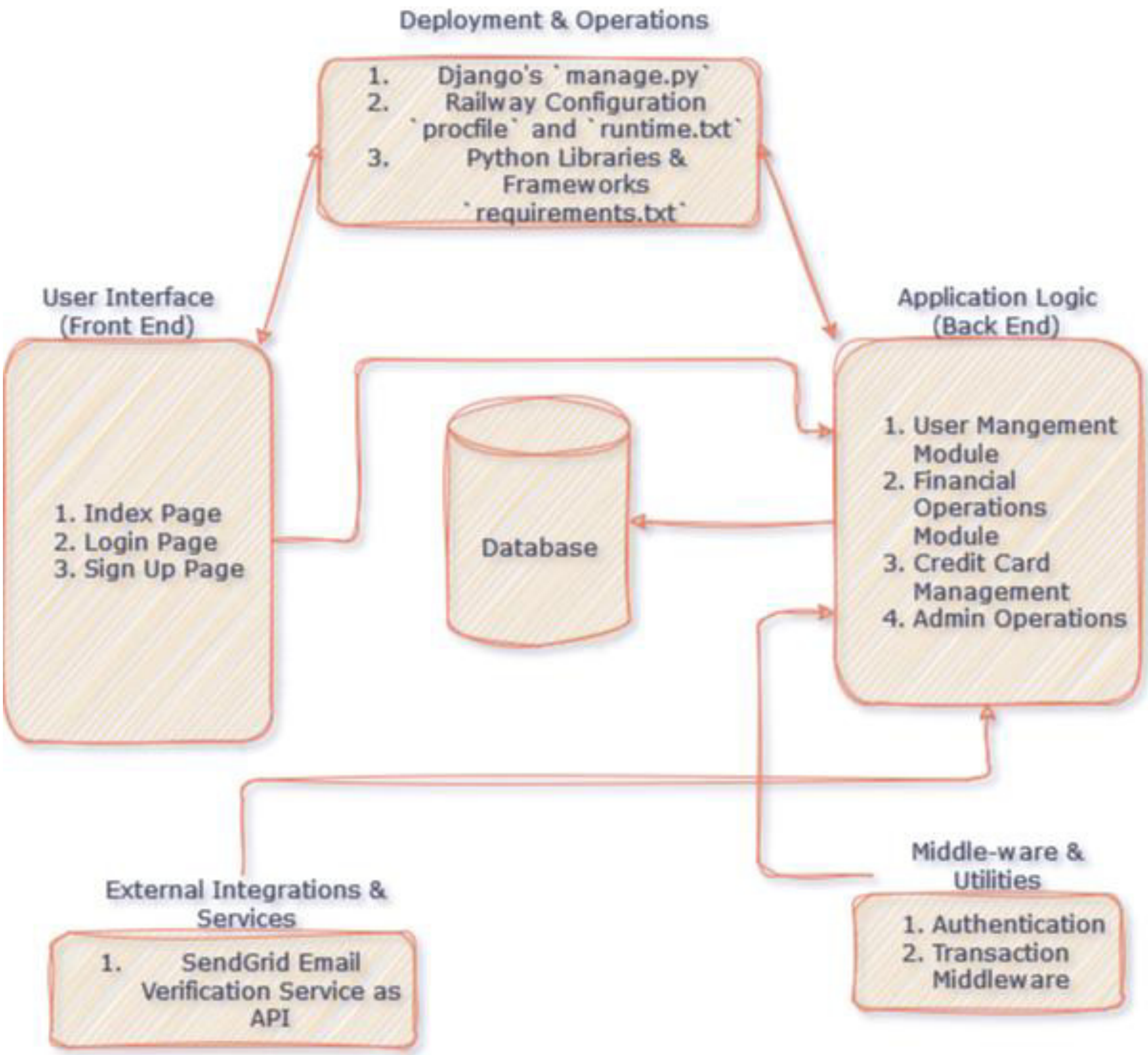


Figure 3 System Architecture

1. Database

The system incorporates a Database component, commonly responsible for:

- Storing persistent data
- Ensuring data reliability, consistency, and availability
- Supporting CRUD (Create, Read, Update, Delete) operations.

2. Middleware & Utilities

This layer can be perceived as the intermediary software that provides services to applications outside of what's offered by the operating system.

- Authentication: Manages user identification, ensuring that users are authenticated and authorized to access specific resources.
- Transaction Middleware: This component oversees transactions, ensuring consistency and atomicity. It also supports functionalities like rollback, commit, and isolation.

3. Application Logic (Back End)

This segment represents the heart of the application, providing the core business logic and functionalities. It encompasses:

- User Management Module: Manages user-related tasks such as registration, profile management, password resets, and more.
- Financial Operations Module: As the name suggests, this module facilitates financial transactions, reporting, and related operations.
- Credit Card Management: Dedicates itself to operations related to credit card information, such as storage (likely in an encrypted form), transactions, and verification using PIN.
- Admin Operations: This component appears to cater to administrative tasks. This encompasses functionalities like user role management, system configurations, oversight of application metrics, and more.

4. User Interface (Front End)

Serving as the visual and interactive layer of the application, the front end is the point of direct contact for users. It includes:

- Index Page: Acts as the primary entry point or landing page of the application, offering an overview and possibly latest updates or features.
- Login Page: Facilitates user login, likely integrating with the Authentication component in the Middleware & Utilities layer.
- Sign Up Page: Enables new users to create accounts, gathering requisite information, and initiating the user registration process.

6.3 Database Schema

We would like to draw the attention to several pivotal aspects that underscore the system's functionality and security. The "`core_transaction`" and "`account_kyc`" tables stand as the backbone of the system, meticulously recording transaction details and maintaining vital Know Your Customer (KYC) information, respectively. A closer look at the "`core_creditcard`" table reveals it as a crucial repository for credit card information, playing a significant role in the financial transactions occurring within the system.

We would like to emphasize the interrelationships and dependencies between these tables in the diagram, offering a window into the database's complex operational intricacies. Highlighting the roles of Django's authentication and session frameworks, as represented by tables such as "`auth_group`" and "`django_session`". The "`userauths_user`" table, passwords are stored as hashes, a technique that greatly enhances security by ensuring that actual password strings are not stored in the database, thus providing a robust defence against potential data breaches.

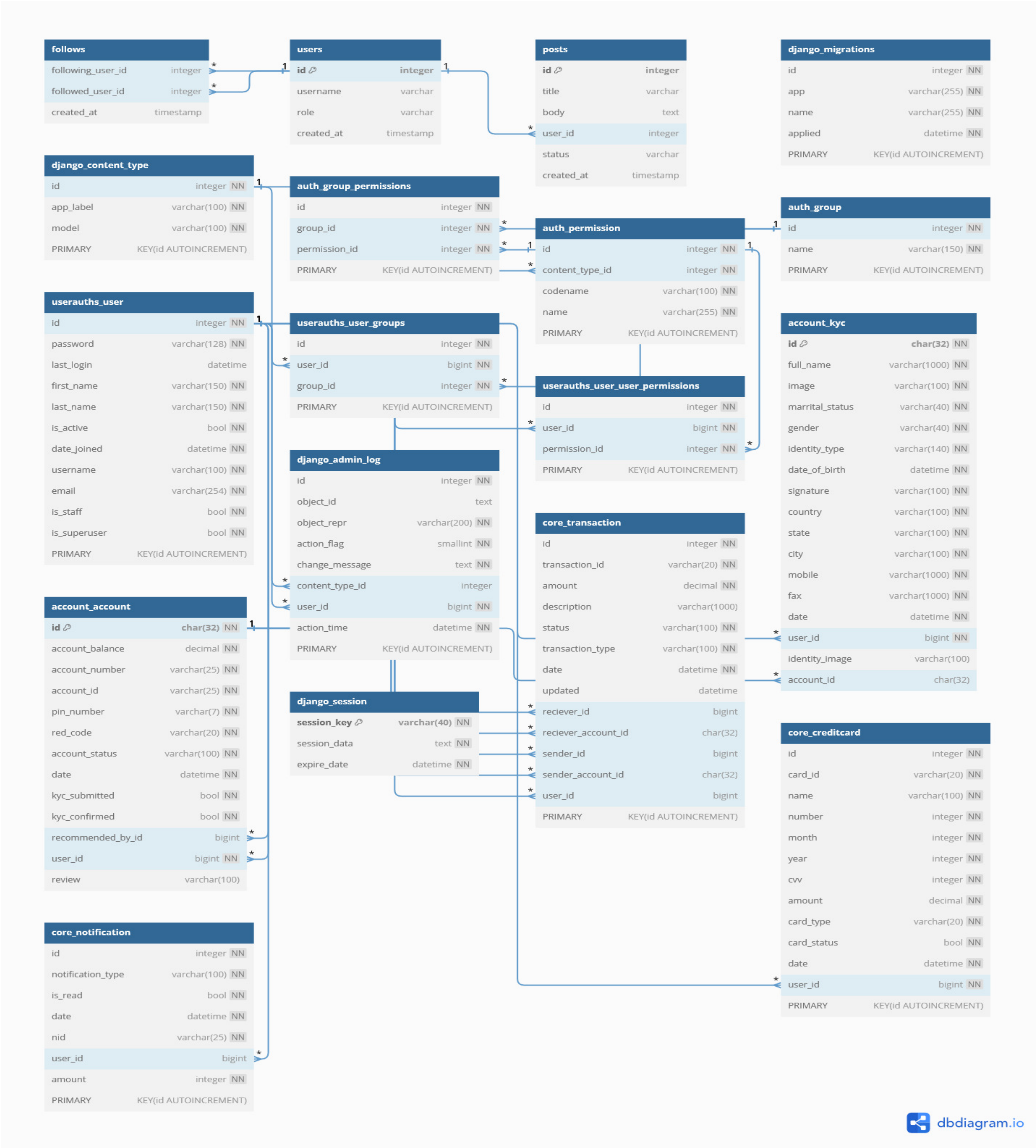


Figure 4 Database Schema

The overview of the database schema is as follows:

1. `account_kyc`

   Purpose: Stores identity verification data for users.

   Key Columns: `full_name`, `date_of_birth`, `identity_type`, `country`, `user_id`.

2. `account_account`

   Purpose: Manages user financial accounts.

   Key Columns: `account_balance`, `account_number`, `account_status`, `kyc_submitted`,`user_id`.

3. `core_transaction`

   Purpose: Records financial transactions.

   Key Columns: `transaction_id`, `amount, description`, `transaction_type`,`reciever_id`,`sender_id`.

4. `core_creditcard`

   Purpose: Details user credit cards.

   Key Columns: `card_id`, `name`, `number`, `month`, `year`, `card_type`, `user_id`.

5. `core_notification`

   Purpose: Handles user notifications.

   Key Columns: `notification_type`,`is_read`, amount, `user_id`.

6.4 Deployment Architecture

Figure 4 highlights the birds eye view of the deployment architecture.

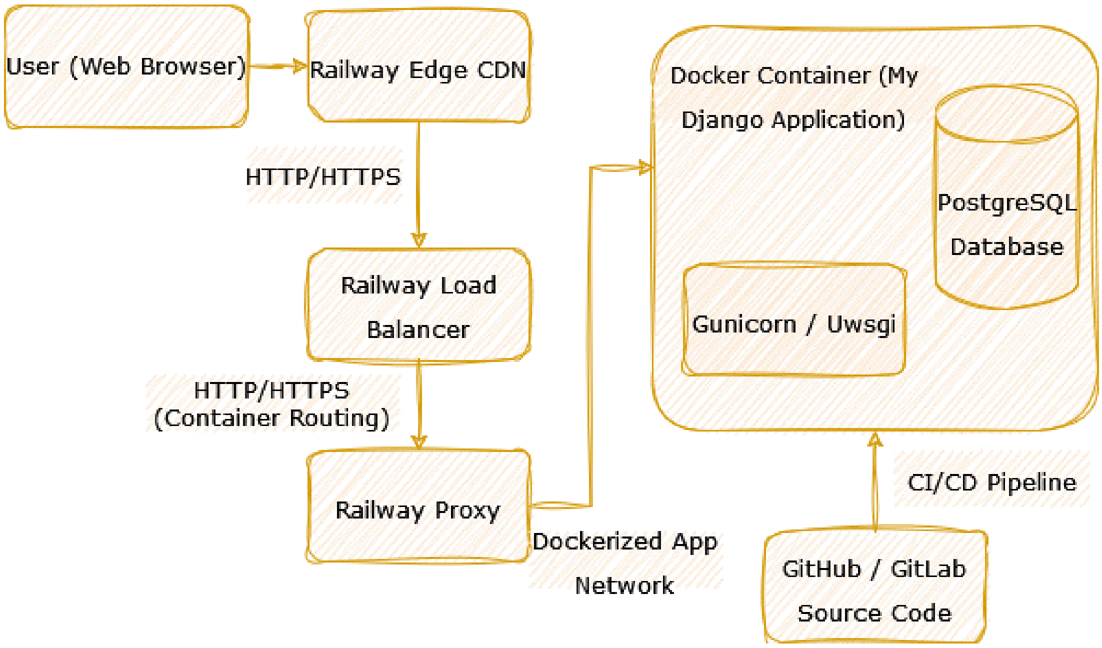


Figure 5 Deployment Architecture

The overview of the deployment architecture is as follows:

1. User Interaction Layer:
    - User's Device: Represents the end-users' devices, such as desktops, laptops, mobile phones, etc., which interact with your system.
    - Web Browser/Client: This is the application or platform from which users send requests to the server, e.g., Chrome, Firefox, mobile apps, etc.
2. Front-end Infrastructure:
    - CDN (Content Delivery Network): This is used to cache and deliver content to users based on their geographical location, ensuring faster response times. CDNs often store static assets like images, CSS, and JavaScript files.
    - Load Balancer: Distributes incoming traffic across multiple servers to ensure no single server is overwhelmed with too many requests. This provides redundancy and high availability.
3. Application Servers:
    - These are the servers where the application runs. They handle user requests, execute application logic, interact with databases, etc.

4. Backend Services:
    - Database: This is where the application's data is stored. This could be relational databases like MySQL, PostgreSQL, or NoSQL databases like MongoDB.
    - Cache: Systems like Redis or Memcached to cache data and reduce the load on databases, ensuring faster response times.
    - Third-party Services/APIs: External services the application interacts with, such as payment gateways, email services, etc.

5. Infrastructure & Network:
    - Firewalls: Protect the servers and infrastructure from malicious attacks.

- DNS: Resolve the domain name to the IP address of the servers.
- Networking Components: Ensure communication between different parts of the system, like routers, switches, etc.

6. Development & CI/CD:
    - Version Control: Systems like Git, where the codebase is versioned and managed.
    - CI/CD: Continuous Integration and Continuous Deployment/Delivery pipelines automate the testing and deployment of your code.

7 Implementation

In this section I'm going to discuss the step-by-step core implementation of my project, a microfinance banking application designed and catered to do payments and provide a credit card which serves as the primary source of credit application.

`Note:` The entire project was developed in Linux (Kubuntu 22.04 LTS) with Bourne Again SHell (BASH). Therefore, all commands are written following Linux terminologies.

7.1 Setup Project

7.1.1 – Installed Apps / Custom Modules

The project uses several Django apps and has added custom apps and third-party packages for extended functionalities.

Figure 5 shows all installed apps in this project and custom apps created to build the application.

```python
INSTALLED_APPS = [
    'jazzmin',

    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'django.contrib.humanize',

    # Custom Apps
    'core',
    'userauths',
    'account',
    "storages",
]
```

Figure 6 Installed Apps

7.1.5 – Static and Media File Configuration

- **Static Files:** These are our CSS, JavaScript, and image files. The configuration involve setting `STATIC_URL` and `STATICFILES_DIRS` or `STATIC_ROOT` in the `settings.py` file.
- **Media Files:** These are user-uploaded files. For them, `MEDIA_URL` and `MEDIA_ROOT` is defined in the `settings.py` file. Additionally, in the `urls.py` file, there are configurations to serve these media files during development.

Figure 6 shows the directory design or file structure.

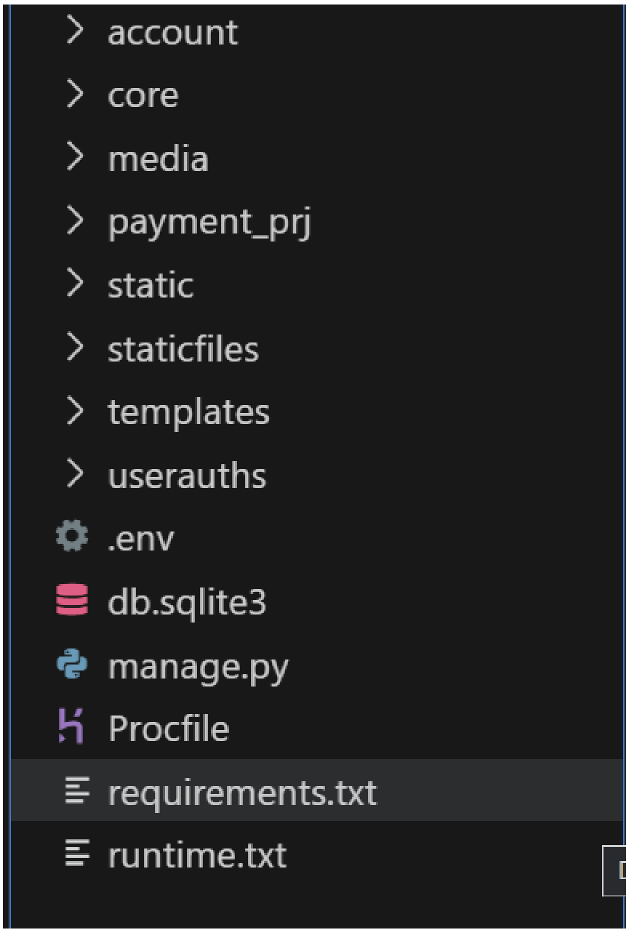


Figure 7 File Structure

```
STATIC_URL = '/static/'
STATICFILES_DIRS = [os.path.join(BASE_DIR, 'static')]
STATIC_ROOT = os.path.join(BASE_DIR, 'staticfiles')

MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
```

Figure 8 Static and Media Folder Configuration

7.1.7 – Template Inheritance and Partials

Django's templating system supports inheritance, allowing for a DRY (Don't Repeat Yourself) approach to templating. This would typically involve:

- **Base Template:** A base template `` `base.html` `` that defines common structures like headers, footers, and navigation.
- **Child Templates:** These templates extend the base template and fill in the content blocks. The `{% extends "base.html" %}` tag is used at the beginning of these templates.
- **Partials:** For reusable components across templates, partials can be defined. These are smaller templates that can be included in other templates using the `{% include "partial_name.html" %}` tag.

7.2 Jazzmin and Admin Dashboard

Django Jazzmin is an intuitive and modern configurable admin dashboard for Django projects. It is a built as a drop-in theme, it integrates seamlessly with Django projects, offering developers a range of customization options to tailor the dashboard to their preferences and project requirements. Famed for its vibrant and modern design, Django Jazzmin allows developers to transform the traditional Django admin interface into a dynamic and interactive workspace. The customizable UI components, coupled with additional features such as a live search and responsive design, have positioned it as a must-have tool for developers looking to elevate their Django projects. Leveraging the capabilities of libraries such as Bootstrap 4 and FontAwesome, Django Jazzmin promises not only a visual overhaul but also a functional enhancement of the Django admin interface, facilitating a more streamlined and efficient project management experience (Farr & Contributors, 2021).

To enhance the web application, I leveraged the capabilities of Jazzmin to craft a tailored admin interface. This not only elevates the visual aesthetics of our administrative dashboard but also streamlines user interactions, resulting in a more intuitive experience. Django's framework was employed to establish a superuser. This step provides privileged access to the admin dashboard, fostering efficient management and oversight.

Admin Login Page

In figure 8, shows how the admin login page looks like.

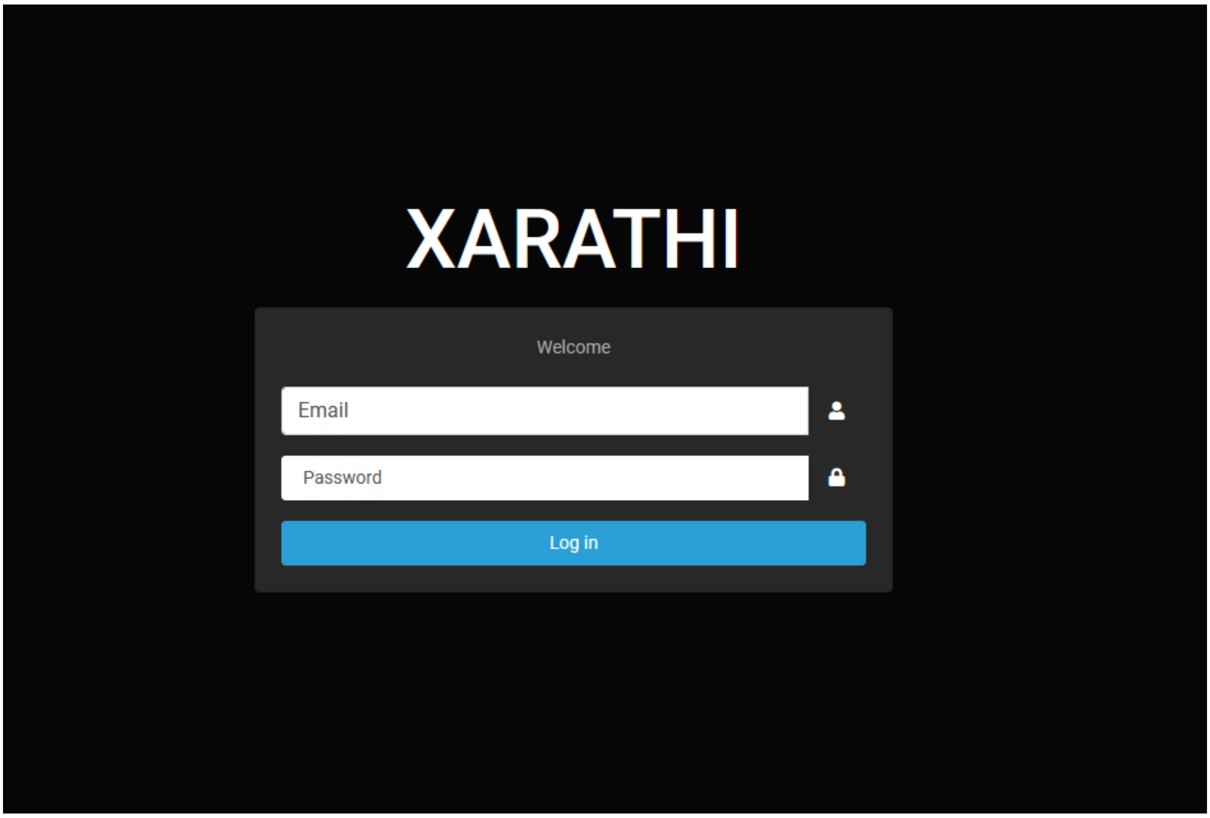


Figure 9 Admin Login Page

This login page can be tailored to specific preferences utilizing the Jazzmin configuration. To access it, a superuser can be set up using the command `django-admin manage.py createsuperuser` provided by Django.

In figure 9 shows how the Admin Dashboard looks like.

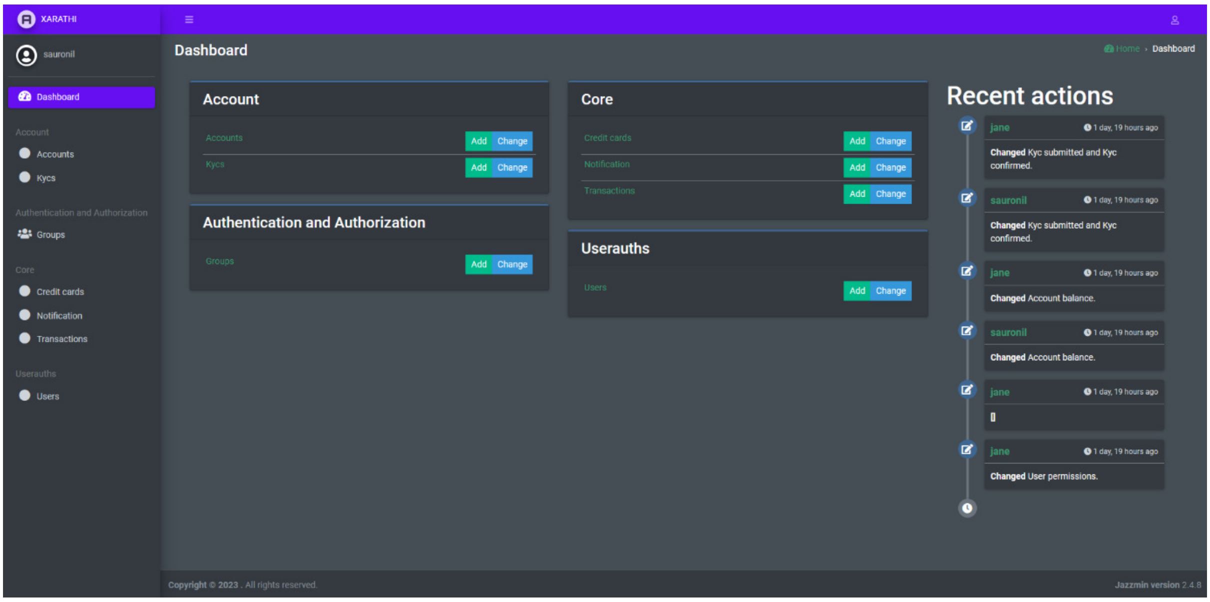


Figure 10 Admin Dashboard

The administrative dashboard is a comprehensive interface, equipped with various sections to facilitate efficient management. It showcases the "`accounts`" segment, which offers in-depth details like account number, status, and balance. The "`KYC`" portion facilitates either approval or discontinuation processes. With "`Groups`", administrators can delegate specific credentials to their counterparts. Vital data, such as credit card information, transaction histories, and notifications, reside under the "`Core`" section. The "`userauth`" domain maintains user data, allowing actions like termination, particularly in instances of extended inactivity or inaccurate KYC entries.

The dashboard extends the flexibility for administrators to manually input account and KYC details. This is especially beneficial for users who submit hard copy forms, opting for a more traditional registration process over the digital platform. Admins also possess the capability to incorporate credit card details, setting the stage for potential credit acquisitions and determining credit values. An intuitive "`notifications`" section keeps admins informed of recent activities, be it KYC modifications, balance adjustments, bank-to-customer transactions, or alterations in user permissions.

7.3 `core` App

1. Choices – There are several choice fields which dictate the predefined options for specific model fields.

```python
TRANSACTION_TYPE = (
    ("transfer", "Transfer"),
    ("recieved", "Recieved"),
    ("withdraw", "withdraw"),
    ("refund", "Refund"),
    ("request", "Payment Request"),
    ("none", "None")
)

TRANSACTION_STATUS = (
    ("failed", "failed"),
    ("completed", "completed"),
    ("pending", "pending"),
    ("processing", "processing"),
    ("request_sent", "request_sent"),
    ("request_settled", "request settled"),
    ("request_processing", "request processing"),

)
```

```python
CARD_TYPE = (
    ("master", "master"),
    ("visa", "visa"),
    ("verve", "verve"),

)

NOTIFICATION_TYPE = (
    ("None", "None"),
    ("Transfer", "Transfer"),
    ("Credit Alert", "Credit Alert"),
    ("Debit Alert", "Debit Alert"),
    ("Sent Payment Request", "Sent Payment Request"),
    ("Recieved Payment Request", "Recieved Payment Request"),
    ("Funded Credit Card", "Funded Credit Card"),
    ("Withdrew Credit Card Funds", "Withdrew Credit Card Funds"),
    ("Deleted Credit Card", "Deleted Credit Card"),
    ("Added Credit Card", "Added Credit Card"),

)
```

Figure 11 Core App Choices

2. Transaction Model: This model captures the details of transactions between users:

```python
class Transaction(models.Model):
    transaction_id = ShortUUIDField(unique=True, length=15, max_length=20, prefix="TRN")
    user = models.ForeignKey(User, on_delete=models.SET_NULL, null=True, related_name="user")
    amount = models.DecimalField(max_digits=12, decimal_places=2, default=0.00)
    description = models.CharField(max_length=1000, null=True, blank=True)
    reciever = models.ForeignKey(User, on_delete=models.SET_NULL, null=True, related_name="reciever")
    sender = models.ForeignKey(User, on_delete=models.SET_NULL, null=True, related_name="sender")
    reciever_account = models.ForeignKey(Account, on_delete=models.SET_NULL, null=True,
    related_name="reciever_account")
    sender_account = models.ForeignKey(Account, on_delete=models.SET_NULL, null=True, related_name="sender_account")
    status = models.CharField(choices=TRANSACTION_STATUS, max_length=100, default="pending")
    transaction_type = models.CharField(choices=TRANSACTION_TYPE, max_length=100, default="none")
    date = models.DateTimeField(auto_now_add=True)
    updated = models.DateTimeField(auto_now_add=False, null=True, blank=True)

    def __str__(self):
        try:
            return f"{self.user}"
        except:
            return f"Transaction"
```

Figure 12 Transaction Model

3. Notification Model: This model is responsible for storing notifications related to users.

```
class Notification(models.Model):
    user = models.ForeignKey(User, on_delete=models.SET_NULL, null=True)
    notification_type = models.CharField(max_length=100, choices=NOTIFICATION_TYPE, default="none")
    amount = models.IntegerField(default=0)
    is_read = models.BooleanField(default=False)
    date = models.DateTimeField(auto_now_add=True)
    nid = ShortUUIDField(length=10, max_length=25, alphabet="abcdefghijklmnopqrstuvxyz")

    class Meta:
        ordering = ["-date"]
        verbose_name_plural = "Notification"

    def __str__(self):
        return f"{self.user} - {self.notification_type}"
```

Figure 13 Notification Model

4. Credit Card Model: This model stores the credit card details of users.

```
class CreditCard(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    card_id = ShortUUIDField(unique=True, length=5, max_length=20, prefix="CARD", alphabet="1234567890")

    name = models.CharField(max_length=100)
    number = models.IntegerField()
    month = models.IntegerField()
    year = models.IntegerField()
    cvv = models.IntegerField()

    amount = models.DecimalField(max_digits=12, decimal_places=2, default=0.00)

    card_type = models.CharField(choices=CARD_TYPE, max_length=20, default="master")
    card_status = models.BooleanField(default=True)

    date = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return f"{self.user}"
```

Figure 14 Credit Card Model

7.4 `userauths` app

The `userauths` app provides a custom user model that extends Django's built-in `AbstractUser` model.

In this figure 14 shows User Model

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


class User(AbstractUser):
    username = models.CharField(max_length=100)
    email = models.EmailField(unique=True)
    is_staff = models.BooleanField(default=False)
    is_superuser = models.BooleanField(default=False)
    email_verified = models.BooleanField(default=False)
    verification_token = models.CharField(max_length=100, blank=True, null=True)

    USERNAME_FIELD = 'email'
    REQUIRED_FIELDS = ['username']

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

Figure 15 User Model

- `username`: Represents the user's username.
- `email`: The email field of the user, set as unique, ensuring each user has a distinct email.
- `is_staff` and `is_superuser`: Flags indicating if the user has staff or superuser privileges.
- `email_verified`: A boolean field indicating if the user's email address has been verified.
- `verification_token`: Token used for email verification purposes.
- `USERNAME_FIELD`: Specifies that the email is used for authentication instead of the default username.
- `REQUIRED_FIELDS`: Indicates that the username is a required field upon user creation.

This custom user model provides flexibility for user authentication using email and includes additional fields to manage email verification via sendgrid.

7.5 `account` app

1. Choices: There are several choice fields which dictate the predefined options for specific model fields:

```python
ACCOUNT_STATUS = (
    ("active", "Active"),
    ("pending", "Pending"),
    ("in-active", "In-active")
)

MARITAL_STATUS = (
    ("married", "Married"),
    ("single", "Single"),
    ("other", "Other")
)

GENDER = (
    ("male", "Male"),
    ("female", "Female"),
    ("other", "Other")
)


IDENTITY_TYPE = (
    ("national_id_card", "National ID Card"),
    ("drivers_licence", "Drives Licence"),
    ("international_passport", "International Passport")
)
```

Figure 16 Account Choices

2. Account Model: This model captures the details of a user's account, including their balance, account number, and KYC status.

```python
class Account(models.Model):
    id = models.UUIDField(primary_key=True, unique=True, default=uuid.uuid4, editable=False)
    user =  models.OneToOneField(User, on_delete=models.CASCADE)
    account_balance = models.DecimalField(max_digits=12, decimal_places=2, default=0.00) #123 345 789 102
    account_number = ShortUUIDField(unique=True,length=10, max_length=25, prefix="217", alphabet="1234567890")
    #2175893745837
    account_id = ShortUUIDField(unique=True,length=7, max_length=25, prefix="DEX", alphabet="1234567890")
    #2175893745837
    pin_number = ShortUUIDField(unique=True,length=4, max_length=7, alphabet="1234567890") #2737
    red_code = ShortUUIDField(unique=True,length=10, max_length=20, alphabet="abcdefgh1234567890") #2737
    account_status = models.CharField(max_length=100, choices=ACCOUNT_STATUS, default="in-active")
    date = models.DateTimeField(auto_now_add=True)
    kyc_submitted = models.BooleanField(default=False)
    kyc_confirmed = models.BooleanField(default=False)
    recommended_by = models.ForeignKey(User, on_delete=models.DO_NOTHING, blank=True, null=True,
    related_name="recommended_by")
    review = models.CharField(max_length=100, null=True, blank=True, default="Review")


    class Meta:
        ordering = ['-id']

    def __str__(self):
        return f"{self.user}"
```

Figure 17 Account Model

7.5 KYC (Know Your Customer) Registration Form

Upon successful registration, every user is directed in the onboarding process: completing the Know Your Customer (KYC) form. This step is a comprehensive verification method designed to ensure the authenticity of our platform's users and to provide a secure environment for all. The KYC form structured such that it requires users to provide a spectrum of personal details. To begin with, users are prompted to upload a clear, recent photograph of themselves. This serves as a visual identifier, aiding in the verification process.

Next, they are required to provide their full legal name. It's essential that this name matches the name on the official documents they will later upload, ensuring consistency and authenticity. The mobile number field is more than just a contact detail; it often serves as a quick verification tool, especially in multi-factor authentication processes. Geographical details such as country, state, and city ensuring compliance with regional regulations and laws. Additionally, the date of birth is required so that the platform's services are being accessed by users of legal age. In the realm of personal details, the KYC form dives deeper, seeking information about the user's gender and marital status. Such details, while seemingly simple, can be instrumental in tailoring user experiences in the future.

One of the most critical sections of the KYC form revolves around account verification. Users are required to specify the type of identification document they are providing, be it a passport, driver's license, or any other valid ID. To corroborate the details provided, users are then prompted to upload two distinct files: an image of the chosen identification document and a clear image of their signature. These uploads play a quintessential role in the verification process, ensuring the legitimacy of the provided information.

Beyond the KYC form, the user dashboard is a hub of functionalities. It offers a detailed transaction history, providing users with a chronological account of their financial activities. The "pay" option facilitates seamless transactions to other users, while the "receive" function allows them to accept funds. Furthermore, the dashboard provides a repository of saved recipients, making future transactions swift and hassle-free.

```python
class KYC(models.Model):
    id = models.UUIDField(primary_key=True, unique=True, default=uuid.uuid4, editable=False)
    user =  models.OneToOneField(User, on_delete=models.CASCADE)
    account =  models.OneToOneField(Account, on_delete=models.CASCADE, null=True, blank=True)
    full_name = models.CharField(max_length=1000)
    image = models.ImageField(upload_to="kyc", default="default.jpg")
    marrital_status = models.CharField(choices=MARITAL_STATUS, max_length=40)
    gender = models.CharField(choices=GENDER, max_length=40)
    identity_type = models.CharField(choices=IDENTITY_TYPE, max_length=140)
    identity_image = models.ImageField(upload_to="kyc", null=True, blank=True)
    date_of_birth = models.DateTimeField(auto_now_add=False)
    signature = models.ImageField(upload_to="kyc")

    # Address
    country = models.CharField(max_length=100)
    state = models.CharField(max_length=100)
    city = models.CharField(max_length=100)

    # Contact Detail
    mobile = models.CharField(max_length=1000)
    fax = models.CharField(max_length=1000)
    date = models.DateTimeField(auto_now_add=True)
    def __str__(self):
        return f"{self.user}"
    class Meta:
        ordering = ['-date']
```
Figure 19 KYC Model

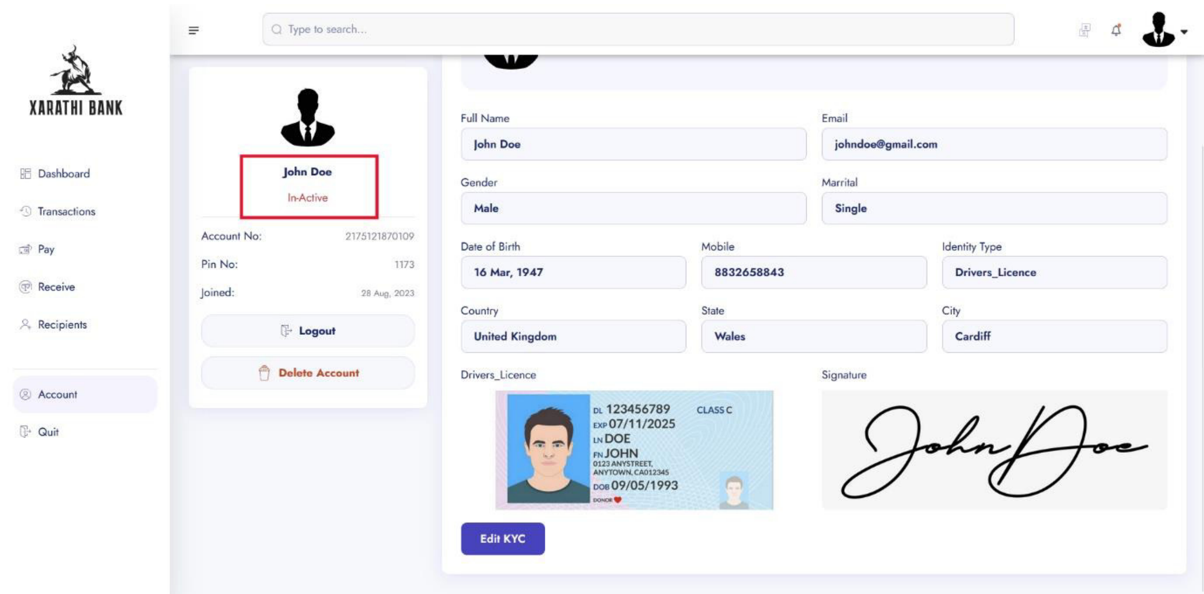


Figure 18 KYC View When Account Created and KYC Submitted

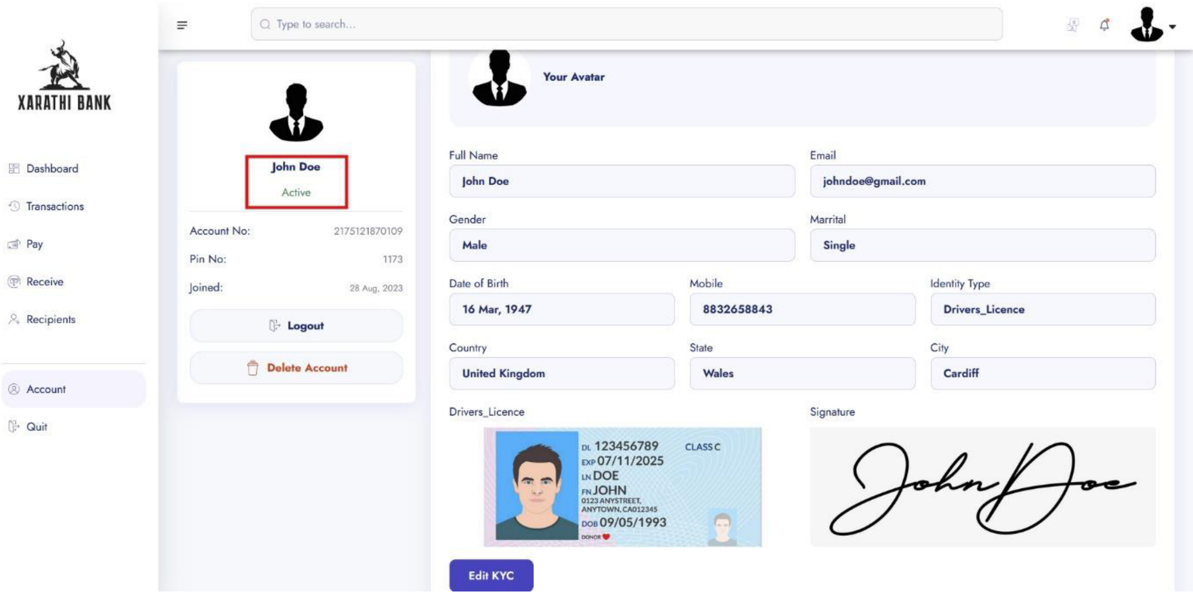


Figure 20 KYC View after KYC is Verified

7.6 Email Verification using SendGrid.

Email verification serves as a crucial security and authenticity measure, helping to prevent fraudulent sign-ups and ensuring that users have provided a valid and active email address. This feature not only safeguards a platform from potential misuse but also fosters a secure and trustable environment for its user community.

A system for email verification involves several components. First, there's a unique token or code associated with each user to validate their email address. This system also requires an email sending mechanism to dispatch the verification link or code to the intended user. Upon receipt, when the user clicks on the verification link, a specific view and URL pattern are designed to manage the email verification process. Once successfully verified, the user's profile or account is updated to mark the email as confirmed.

The implementation is as follows:

1. Unique Token Generation: In the `userauths` app's `models.py`, the `User` model has a field `verification_token`. This field stores the unique token for email verification. The unique token is created by using cryptographically secure random number generator, ensuring that it cannot be predicted or replicated.

```
verification_token = models.CharField(max_length=100, blank=True, null=True)
```

Figure 21 Code Snippet for verification token

2. Email Sending Mechanism: In the `userauths` app's `views.py` (specifically the `RegisterView`), there is a mechanism to send an email to the user upon successful registration. Here, `send_email` is Django's built-in function for sending emails, and `verification_link` contains the unique URL with the token for the user to verify their email.

```
send_mail(
    "Verify Your Email",
    f'Click on the link to verify your email: {verification_link}',
    "sauronildas@protonmail.com",
    [new_user.email],
    fail_silently=False
)
```

Figure 22 Custom Email Template

3. Email Verification View: When a user clicks on the verification link in their email, this view checks the token, and if valid, sets the `email_verified` field of the user to `True`.

```python
def email_verification_view(request, token):
    try:
        user = User.objects.get(verification_token=token)

        if default_token_generator.check_token(user, token):
            user.email_verified = True
            user.verification_token = None
            user.save()

            messages.success(request, "Your email has been successfully verified!")
            return redirect("account:account")

        else:

            messages.error(request, "Email verification link is invalid or has expired. Please request a new one.")
            return redirect("userauths:sign-in")

    except User.DoesNotExist:
        messages.error(request, "Invalid verification link.")
        return redirect("userauths:sign-in")
```

Figure 23 Code Snippet - Email Verification View

4. URL Pattern for Email Verification: In the `userauths` app's `urls.py`, there's a URL pattern for email verification.

```python
urlpatterns = [
    path("sign-up/", views.RegisterView, name="sign-up"),
    path("sign-in/", views.LoginView, name="sign-in"),
    path("sign-out/", views.logoutView, name="sign-out"),
    path('email_verification/<str:token>/', views.email_verification_view, name='email_verification'),
]
```

Figure 24 Email Verification URL pattern

5. API Configuration: In the `settings.py` configure the API for sendgrid, which communicates between my application to sendgrid.

```python
#SENDGRID

EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
SENDGRID_API_KEY = [redacted]
EMAIL_HOST = 'smtp.sendgrid.net'
EMAIL_HOST_USER = 'apikey'
EMAIL_HOST_PASSWORD = SENDGRID_API_KEY
EMAIL_PORT = 587
EMAIL_USE_TLS = True
```

Figure 25 API Configuration

7.7 Deployment on Railway.app

To successfully transition my project from a localized setup to a more accessible and professional platform, I opted for deployment on Railway. This choice wasn't merely a technical preference but a strategic decision to enhance the project's visibility and accessibility to a broader audience. Running the project on localhost would have sufficed for a basic demonstration but deploying it on a platform like Railway amplifies its reach, enables real-time testing, which is instrumental in refining the project further.

The first step towards this transition was creating an account on Railway, a renowned deployment platform. Setting up an account here was a precursor to leveraging the features it offers for project deployment. Post account setup, my focus shifted to preparing my project for seamless integration with Railway. This involved uploading all project files to GitHub, a move that ensured all essential components were readily accessible for the subsequent steps.

Once the project was securely hosted on GitHub, I facilitated a connection between my repository and Railway. This linkage permitted Railway to fetch the project files directly from the repository, setting the stage for an efficient deployment process.

The next phase was the database configuration. Given the reliance of my project on a PostgreSQL database, setting up the relevant connections such as the `DATABSE_URI`, `PGHOST`, `PGPASSWORD`, and other connection. This step was crucial to guarantee a fluid data flow and unhindered access during the deployment phase.

To enhance user experience and enhance the project's professional appeal, I assigned a unique domain name to the application. This step transcended mere aesthetics, aiming to craft an identity that users could easily remember and access. Following the creation of the domain, I configured it to point directly to the application hosted on Railway, thereby establishing a public portal that invited users to interact with the application seamlessly and professionally.

The link to the website - https://xarathi-bank.up.railway.app

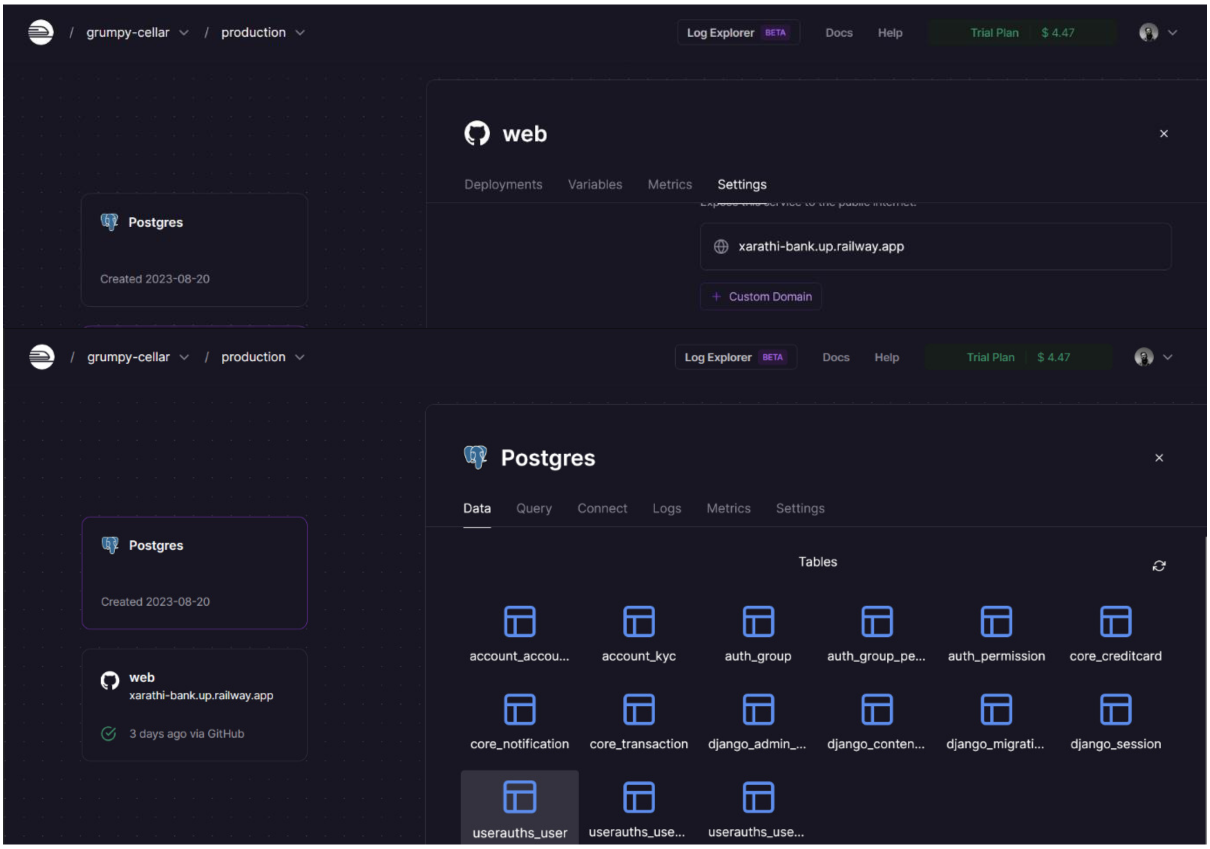


Figure 26 Deployment Dashboard

7.8 Amazon S3 Bucket Configuration

To efficiently utilize an Amazon S3 bucket for storing static files, it's essential to first set up a root account on AWS. This account will grant you access to various services available on the AWS platform.

Next, prepare the Python environment to communicate effectively with AWS by incorporating three crucial libraries: `boto3`, `botocore`, and `django-storages`. These libraries will serve as a bridge, facilitating seamless interaction between your Python applications and the AWS services.

Following the software setup, create an IAM user to manage secure and regulated access to the AWS services. Grant this user sufficient permissions, including S3FullAccess, to ensure a wide-ranging control over S3 bucket operations. Next, we establish an S3 bucket and link it with the IAM user to create a secure pathway for storing and retrieving your static files. Subsequently, adjust the AWS bucket policy to allow public access, facilitating smoother data exchanges. Additionally, configure the CORS (Cross Origin Resource Sharing) settings to enable PUT, GET, and POST methods, establishing rules that govern how various web applications interact with your bucket.

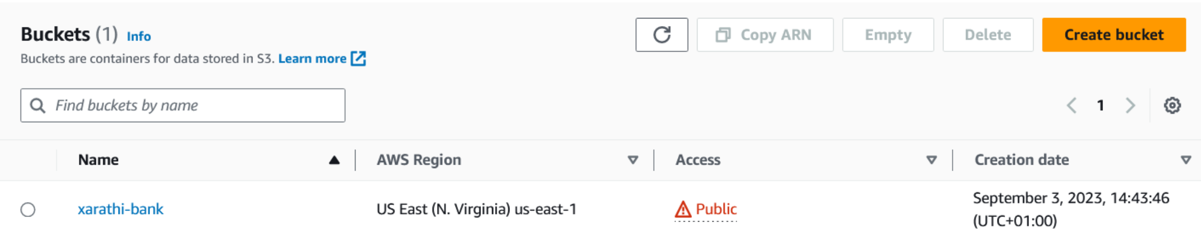


Figure 27 S3 Bucket

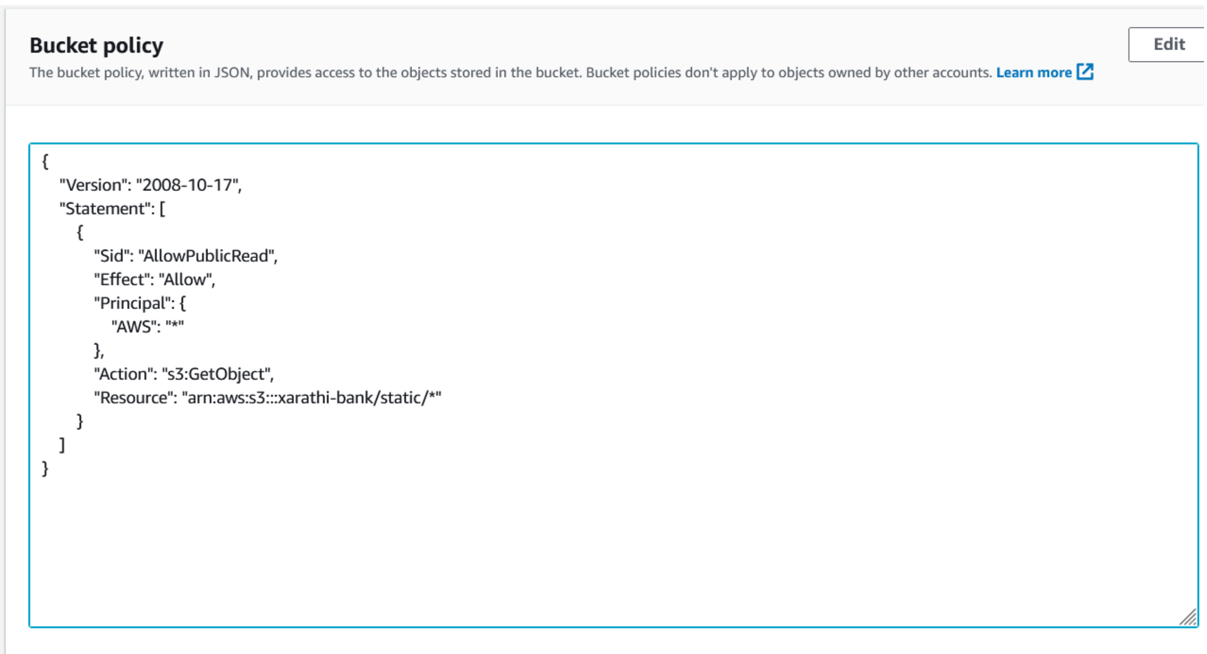


Figure 28 Bucket Policy

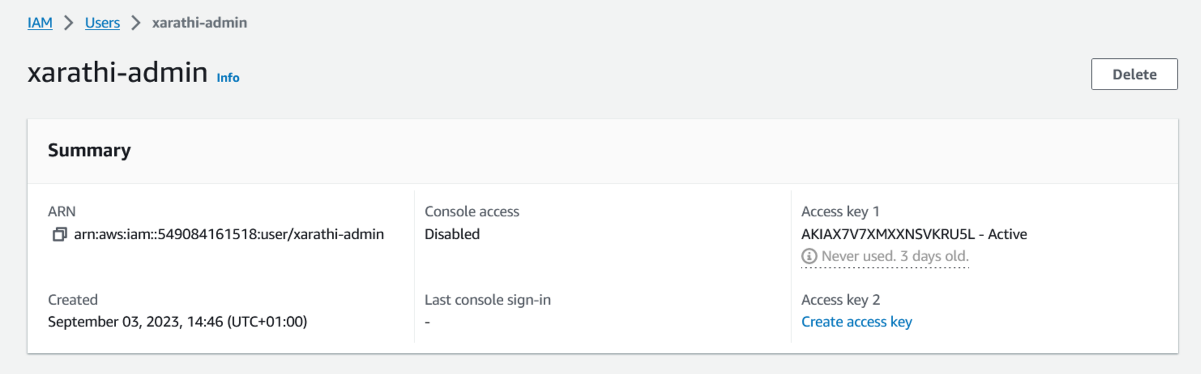


Figure 29 IAM user

Finally, integrate all the established connections into your project's settings.py file, ensuring a cohesive and efficient workflow, where data management is streamlined and secure.

```
# Amazon S3

AWS_ACCESS_KEY_ID = "AKIAX7V7XMXXIHD2U4HL"

AWS_SECRET_ACCESS_KEY = [redacted]

AWS_STORAGE_BUCKET_NAME = "xarathi-bank"

AWS_S3_FILE_OVERWRITE = False

AWS_DEFAULT_ACL = None

DEFAULT_FILE_STORAGE = 'storages.backends.s3boto3.S3Boto3Storage'

STATICFILES_STORAGE = 'storages.backends.s3boto3.S3Boto3Storage'

AWS_S3_OBJECT_PARAMETERS = {'CacheControl': 'max-age=86400'}

AWS_S3_CUSTOM_DOMAIN = f'{AWS_STORAGE_BUCKET_NAME}.s3.amazonaws.com'

AWS_LOCATION = 'static'

STATIC_LOCATION = 'static'

STATIC_URL = f'https://{AWS_S3_CUSTOM_DOMAIN}/{STATIC_LOCATION}/'
```

Figure 30 Configuration of S3 and Django Application

8 Results and Evaluation

This section discusses the methodology used to test the product, including the presentation of test cases and their results and finally whether all Functional and Non-Functional Requirements were met.

8.1 Testing Methodology

Black box testing is a method of testing where the internal workings of a system are not known or considered. It focuses on the inputs and outputs of the system without any knowledge of its internal structure or code. This type of testing is useful when the source code is not available or when testing needs to be done from a user's perspective. Black box testing can be enhanced by incorporating randomness and learning models to improve coverage and accelerate the testing process (Esparza and Grande, 2023). It can also be applied to RESTful APIs, where the source code is not accessible, by using swarm intelligence algorithms to generate test suites and maximize test coverage (N. et.el, 2023). Additionally, black box testing can be done on software using only its binary code by using execution fingerprints to identify different execution paths (Roi, et.el, 2023).

I chose the black box testing methodology to test my project because it allows me to focus on the functionality of the system without getting bogged down in the details of its internal workings. This approach is particularly useful when testing complex systems, as it allows me to identify issues that may not be immediately apparent from an examination of the code. Additionally, black box testing is a great way to ensure that the system meets the needs of its users, as it allows me to test the system from the perspective of an end user. By providing inputs and observing the outputs generated by the system, I can identify any usability or reliability issues that may impact the user experience.

8.2 Test Cases

8.2.1 User Management

Table 1 User Management - Black Box Testing

Test Case ID	Description	Preconditions	Steps	Expected Results	Status
1.	Registration with Valid Details	User is on the registration page.	1. Enter valid user. credentials. 2. Click the “Register” button.	User account is created successfully, and a verification email is sent.	Pass

Evidence:

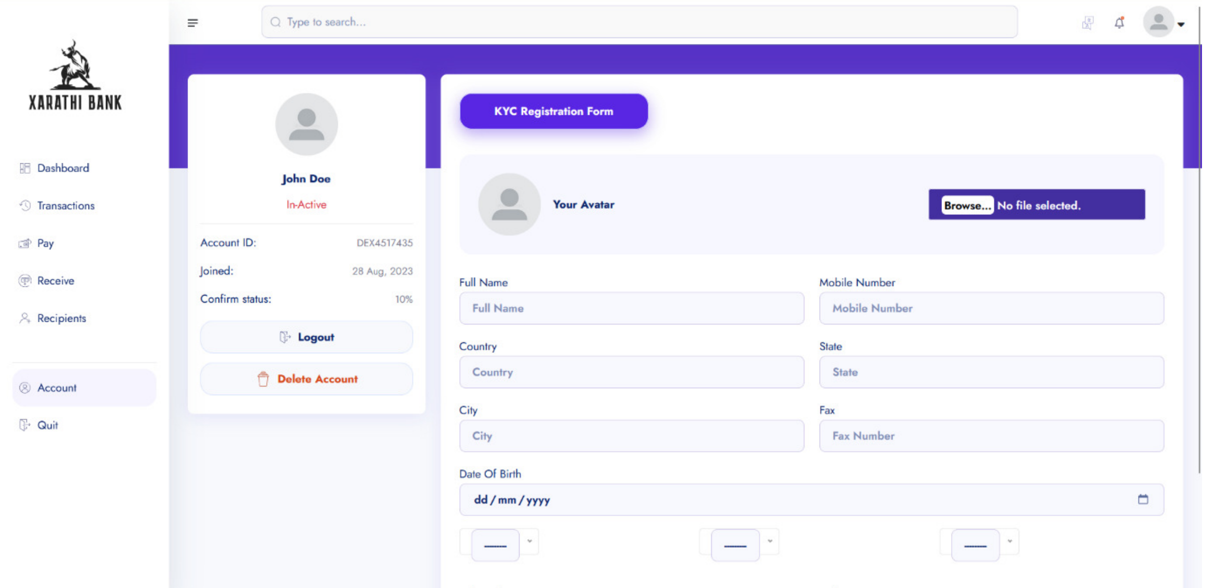


Figure 31 Account Created

2.	Register with Incorrect Password	User is on the registration page.	1. Enter two different passwords (ex – Abcd@1234 and abcd@1234)	Error message indicating passwords 1 and 2 does not match.	Pass
Evidence:					

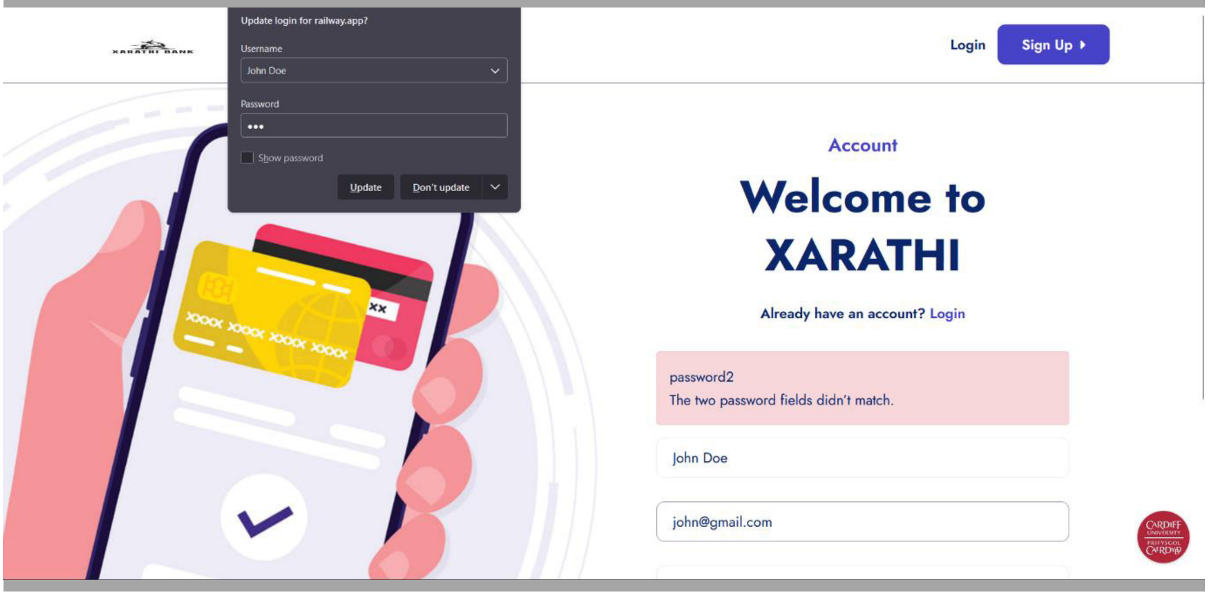


Figure 32 Passwords 1 and 2 Does not match.

3.	Login with Valid Credentials	User has an account.	1. Enter registered email and correct password. 2. Click the "login" button.	User is logged into their account.	Pass

Evidence:

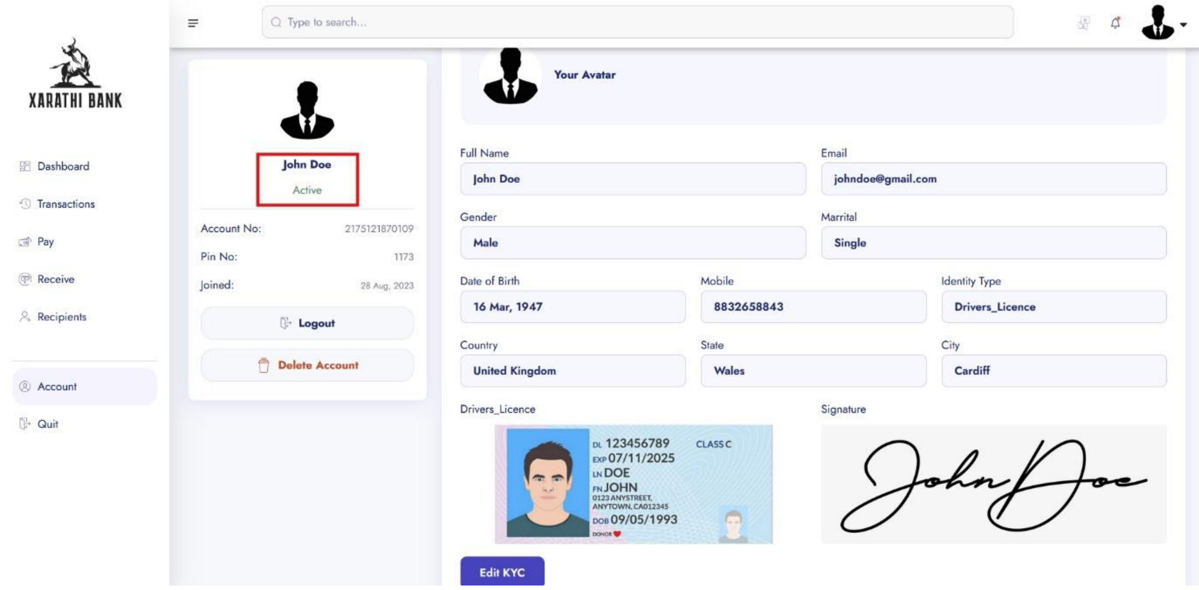


Figure 33 Account Logged in Page

8.2.2 KYC

Table 2 KYC - Black Box Testing

Test Case ID	Description	Preconditions	Steps	Expected Results	Status
1.	Upload Valid ID Proof	User is on the KYC Verification Page	1. Click the “Upload ID” button. 2. Select valid ID proof (e.g., Passport, Driver’s License)	ID proof is uploaded successfully, and the user is notified. KYC status changes to “Pending Verification”	Pass

Evidence:

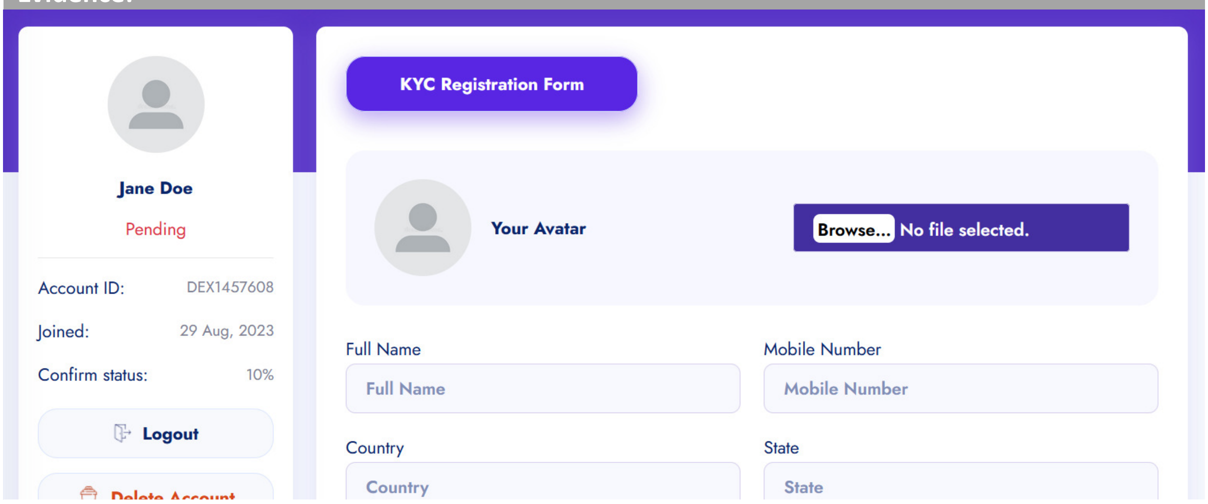


Figure 34 KYC Verification Pending

2.	Upload Invalid ID Proof Format	User is on the KYC Verification page.	1. Click the "Upload ID" button. 2. Select an invalid format (e.g., .exe). 3. Submit.	Error message indicating incorrect credentials and admin is not granted access.	Fail
Evidence: Testing Failed – Cannot Provide Evidence					
4.	KYC Verification Approval	Admin is reviewing a valid KYC submission.	1. Review the uploaded ID proof. 2. Click the "Approve" button.	KYC status for the user is set to "Active".	Pass
Evidence:					

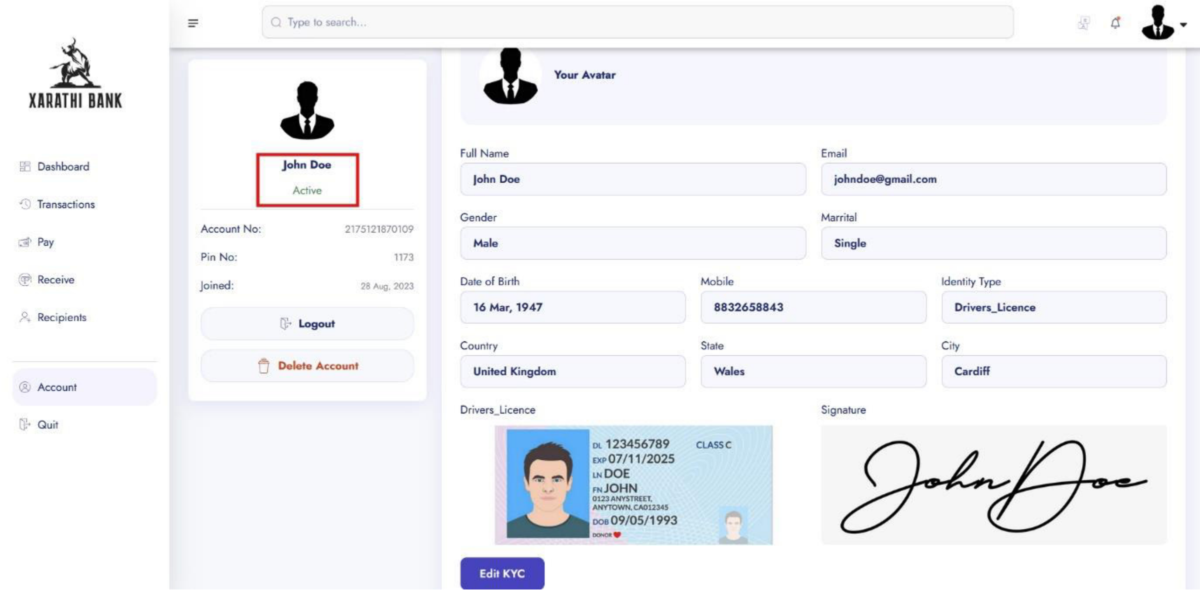


Figure 35 KYC Verified and account Active.

8.2.3 Email Verification

Table 3 Email Verification - Black Box Testing

Test Case ID	Description	Preconditions	Steps	Expected Results	Status
1.	Email Verification Link Functionality	User has received a verification email.	1. Click the email verification link sent to the registered email.	User's email is verified, and a success message is displayed.	Pass
Evidence:					

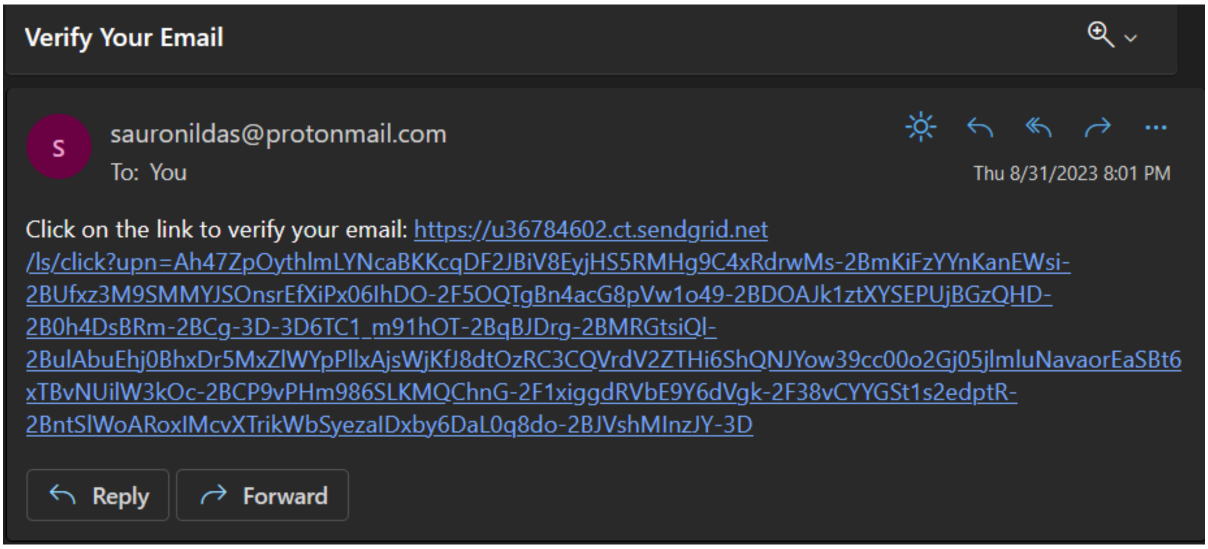


Figure 36 Email Verification Link

2.	Logged into Account	Use has clicked on the verification link.	1. Login into the system after verification Link.	Message saying your email has been verified	Pass
Evidence:					

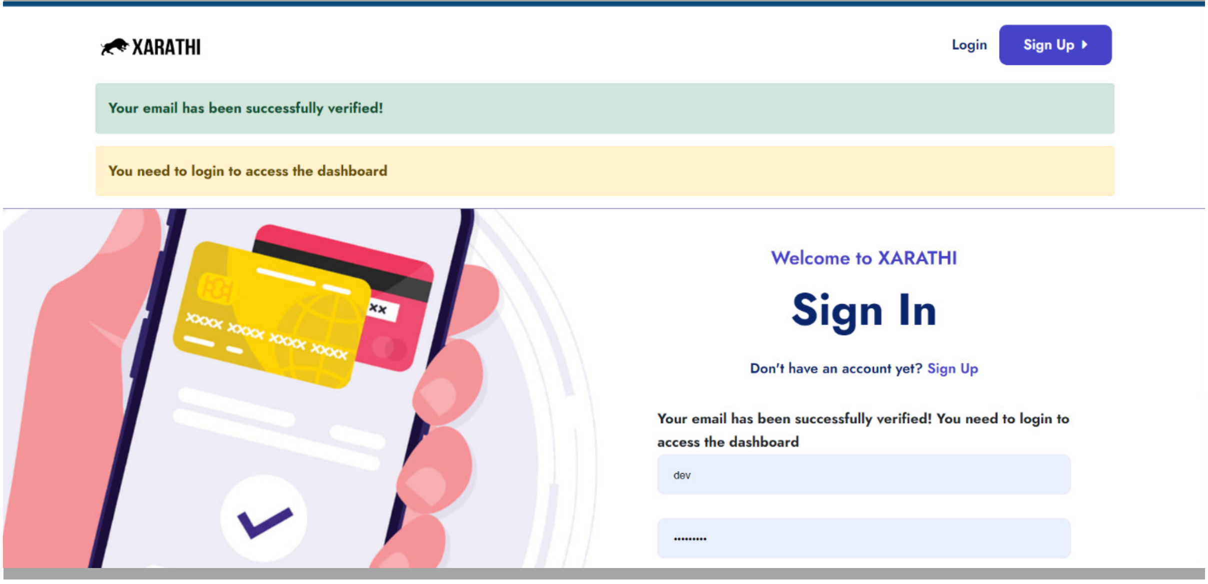


Figure 37 Email Verified

8.2.4 Admin Interface

Table 4 Admin Interface - Black Box Testing

Test Case ID	Description	Preconditions	Steps	Expected Results	Status
1.	Access Admin Dashboard	Admin is on the login page	1. Enter valid admin credentials.	Admin is directed to the dashboard and can view all	Pass

			2. Click on “login” button.	available functionalities.	

Evidence:

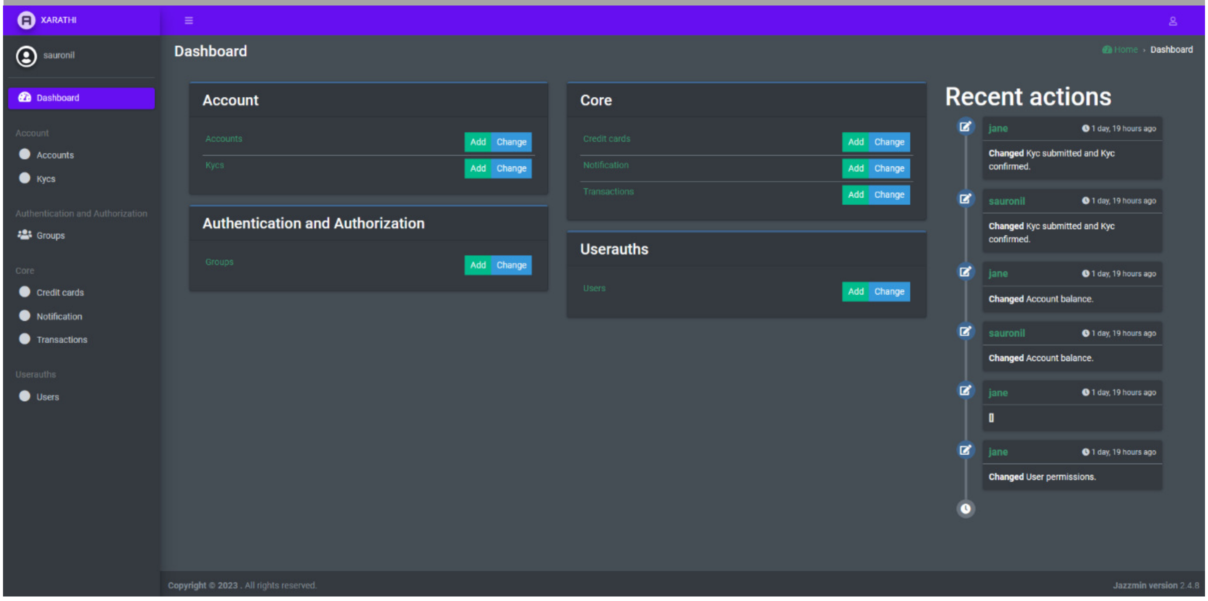


Figure 38 Admin Dashboard

2.	Access Admin Dashboard with Invalid Credentials	Admin is on the login page.	1. Enter invalid admin credentials. 2. Click the ‘’login” button	Error message indicating incorrect credentials and admin is not granted access.	Pass

Evidence:

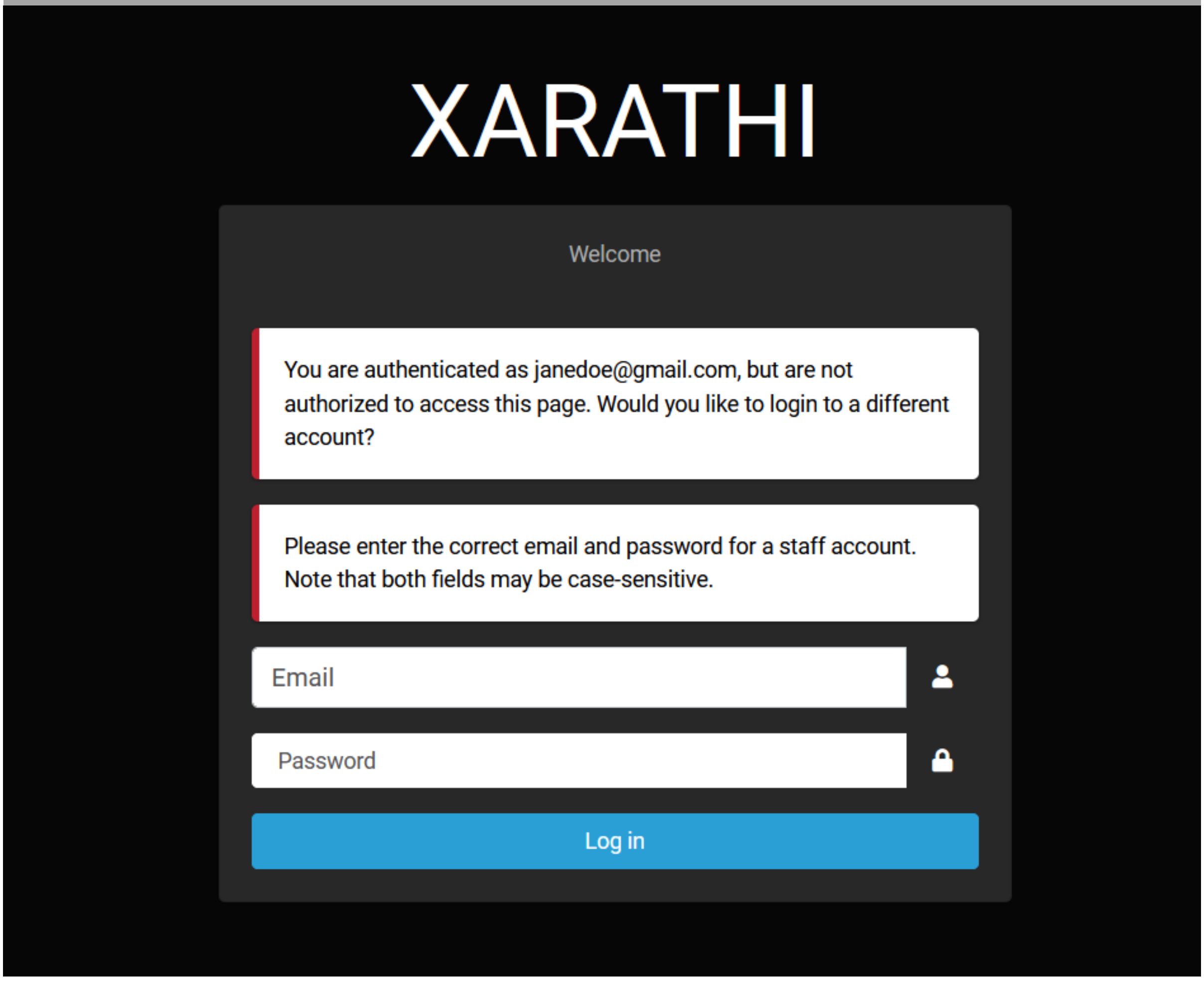


Figure 39 Admin Login Page Error Message

3.	CRUD Operations on a Model (e.g, User)	Admin is logged into the dashboard	1. Navigate to the "Users" section. 2. Add, edit, view, and delete user entries.	All CRUD operations are executed successfully without errors.	Pass

Evidence:

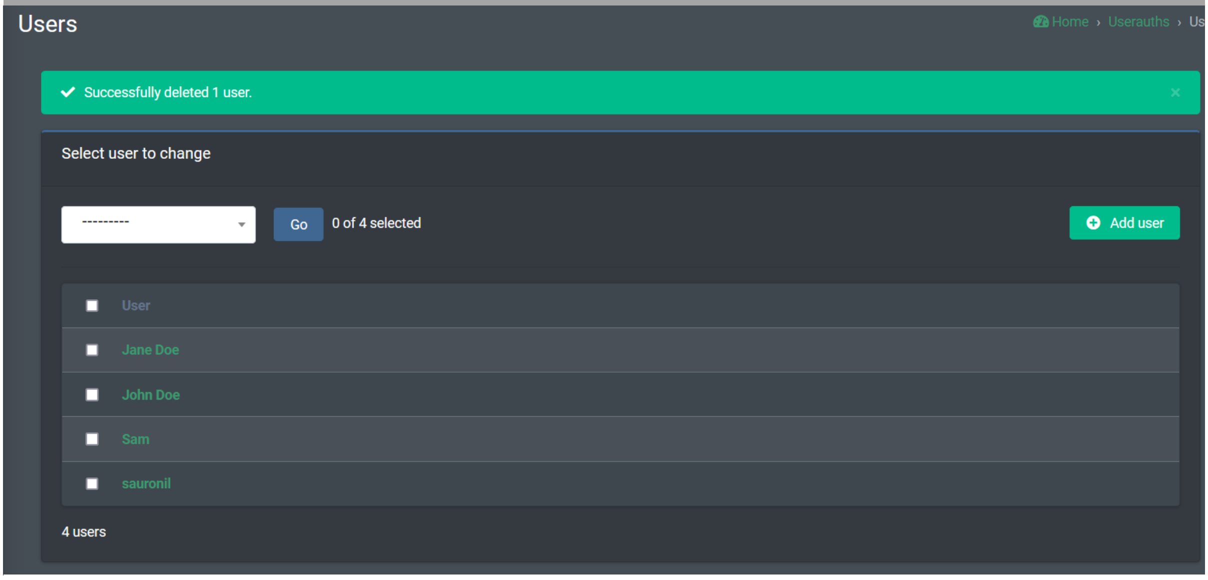


Figure 40 CRUD Operation

4.	View System Logs and Activities	Admin is logged into the dashboard	1. Navigate to the “logs” or “Activities” section.	Admin can view all system logs and activities, errors, and system events.	Pass

Evidence:

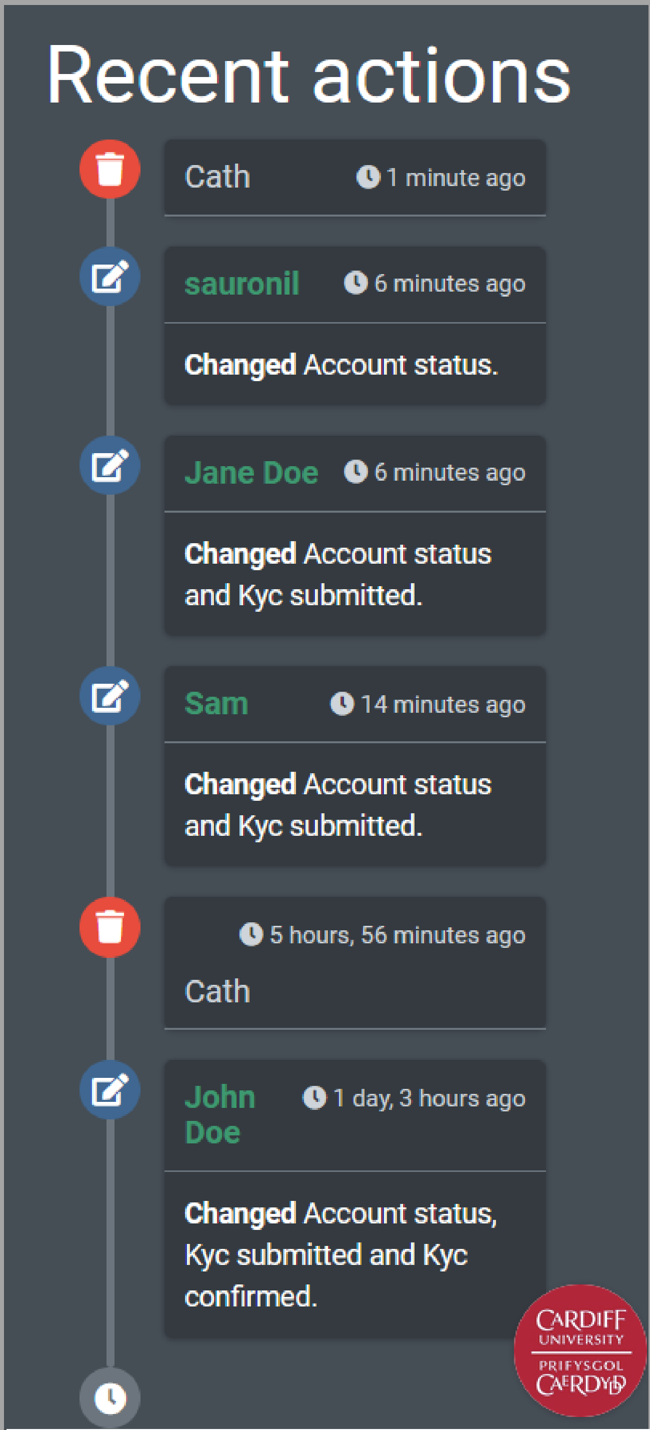


Figure 41 Notification Panel

5.	Logout from Admin Dashboard	Admin is logged into the dashboard.	1. Click on the “logout” option.	Admin is successfully logged out and redirected to the index page.	Pass

Evidence:

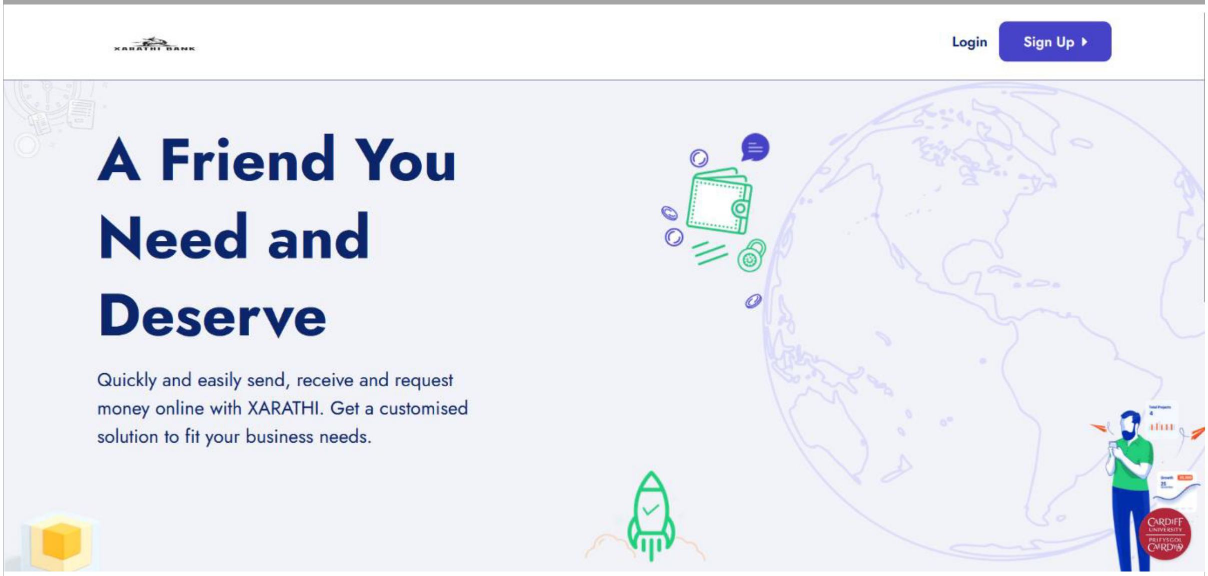


Figure 42 Admin Logout

8.2.5 Transaction Control

Table 5 Transaction Control - Black Box Testing

Test Case ID	Description	Preconditions	Steps	Expected Results	Status
1.	Successful Transaction	User initiates a valid transaction (e.g., transferring).	1. Select Account ID. 2. Provide valid PIN	Transaction is successful, and relevant data is updated.	Pass
Evidence:					

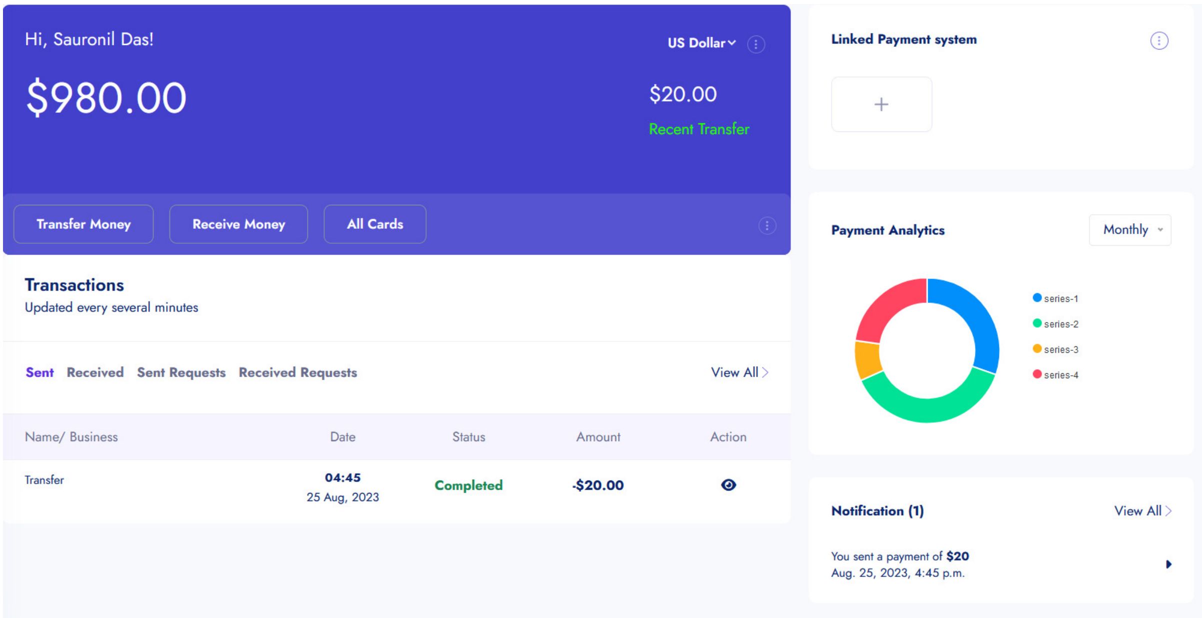


Figure 43 Successful Payment

2.	Transaction History	User is on transactions history.	1. Select Transactions	All past transaction is present.	Pass

Evidence:

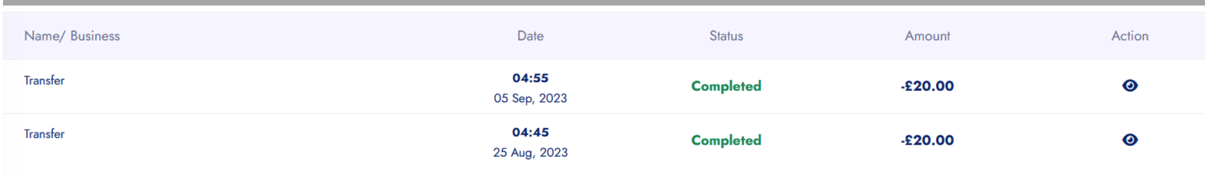


Figure 44 Transaction History

3.	Notification	User is on his account page.	1. Navigate to notification	User can view notifications.	Pass

Evidence:

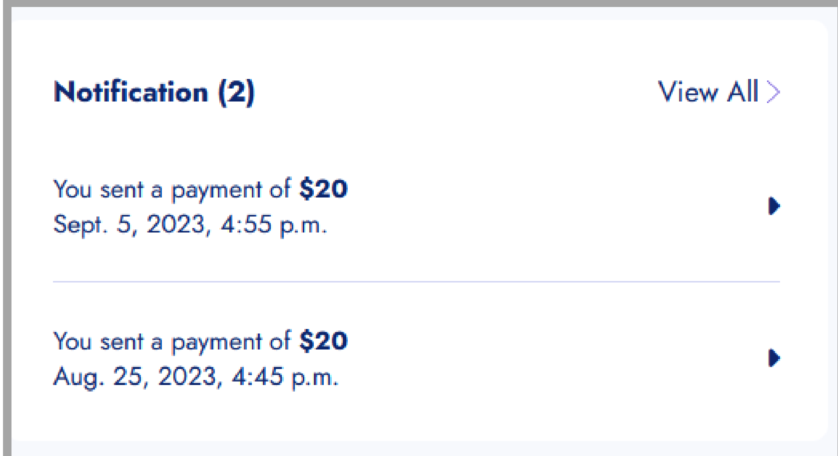


Figure 45 Notification

8.2.6 Deployment

Table 6 Deployment - Black Box Testing

Test Case ID	Description	Preconditions	Steps	Expected Results	Status
1.	Initial Deployment	Fresh environment with no previous of the app.	1. Execute the deployment script or process. 2. Start application	Application starts without errors and is accessible to users.	Pass

Evidence:

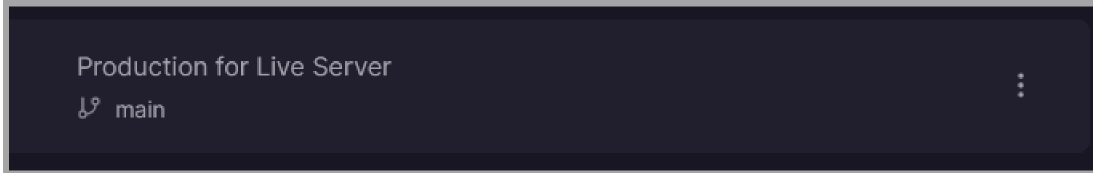


Figure 46 Initial Deployment

Test Case ID	Description	Preconditions	Steps	Expected Results	Status
2.	Database Migration during Deployment	New version contains database schema changes	1. Execute the deployment script or process. 2. Run database migrations.	Database schema is updated without errors, and data integrity is maintained.	Pass

Evidence:

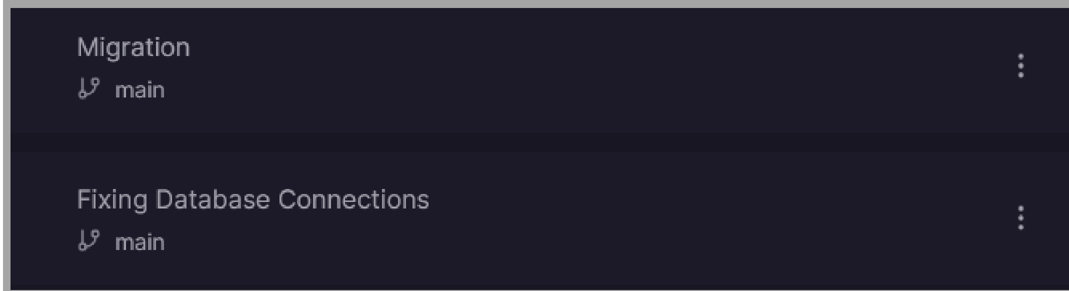


Figure 47 Database Migration

Test Case ID	Description	Preconditions	Steps	Expected Results	Status
3.	Rollback Deployment	Deployment of a new version fails or contains errors.	1. Trigger the rollback process to the previous stable version	Application reverts to the previous stable version without data loss or downtime.	Pass

Evidence:

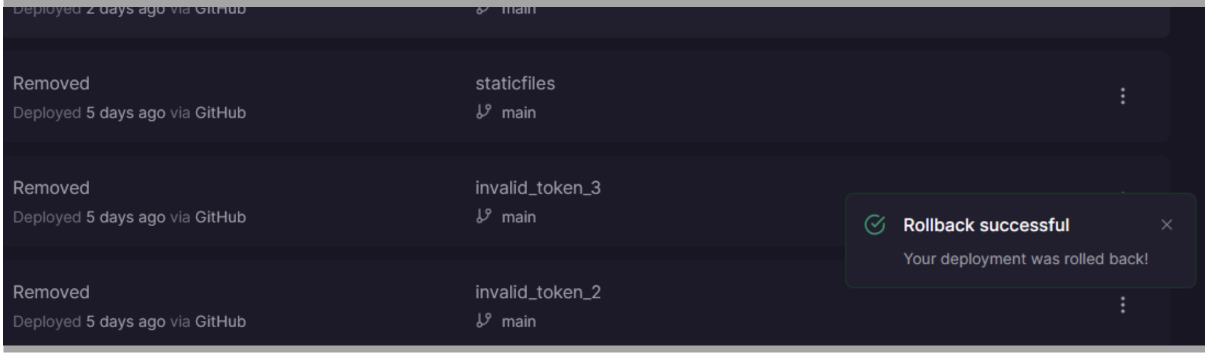


Figure 48 Rollback Deployment

8.3 Google Lighthouse Report

Google Lighthouse is an open-source, automated tool developed by Google to aid developers in improving the quality of their web pages. It offers a comprehensive analysis of web applications, scrutinising various critical parameters such as performance, accessibility, SEO, and best practices for modern web development (Google, 2021). The primary objective of this tool is to provide insights that can help in optimizing websites, thereby ensuring a better user experience.

It is essential to note that Lighthouse operates in the background, analysing web pages through a series of audits for web performance and other vital metrics. It can be run in various ways: as a Chrome Extension, from the Chrome DevTools, from the command line, or as a Node module. When initiated, Lighthouse simulates a user visiting the webpage on a mobile device and conducts a series of evaluations based on established performance metrics and best practices. It then generates a report detailing the analysis with scores in different categories, along with recommendations for improvements (Lighthouse Developers, n.d.).

Utilizing Google Lighthouse in the development process can significantly assist developers in identifying issues that might impede the performance or accessibility of a website, offering actionable insights to optimize the web page for a superior user experience.

Below are series of images from the Google Lighthouse Report.

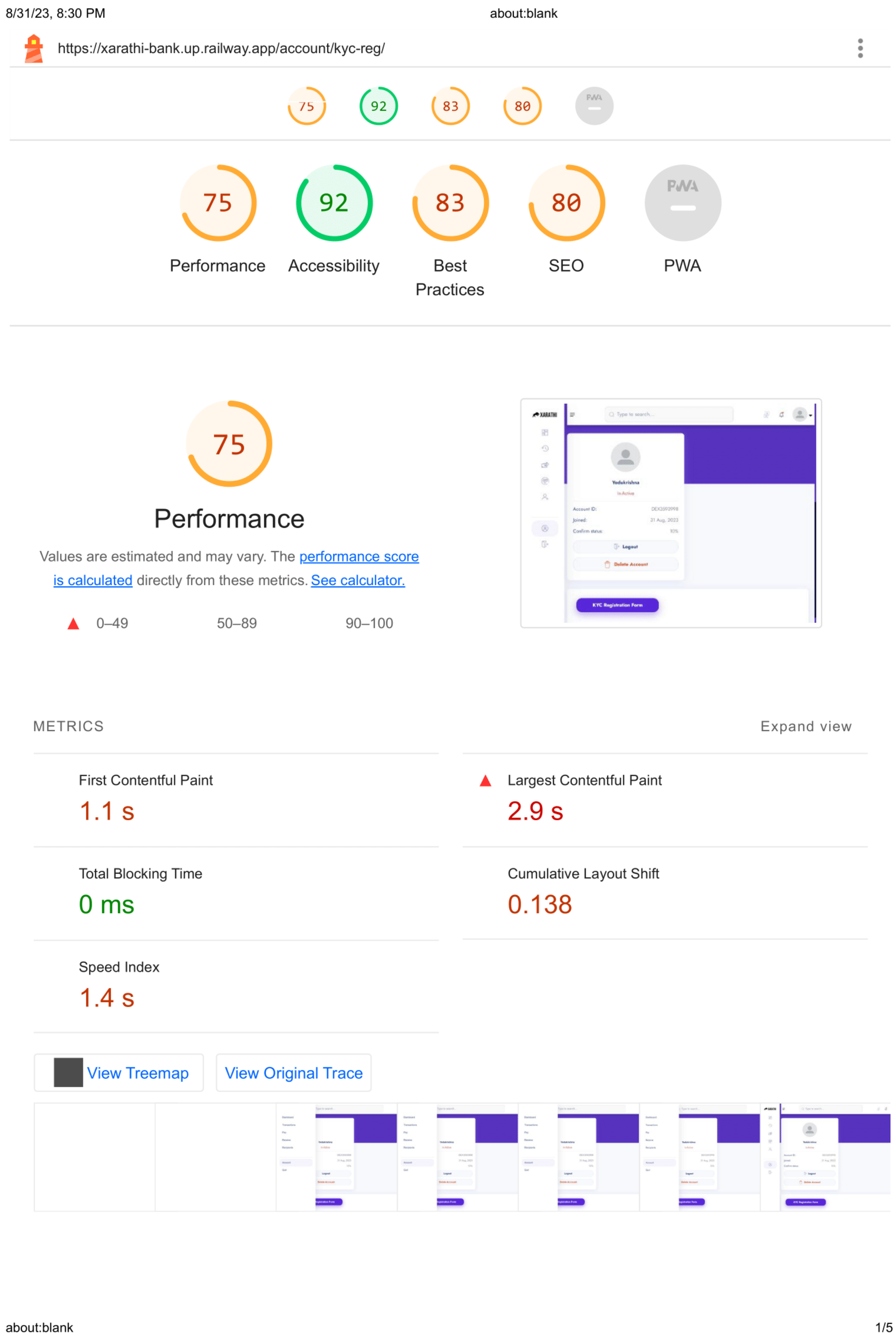
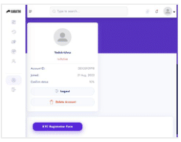


Figure 49 Google Lighthouse Page 1



Show audits relevant to: All FCP LCP TBT CLS

OPPORTUNITIES

Opportunity	Estimated Savings
▲ Enable text compression	1.72s
▲ Reduce unused JavaScript	1.28s
Minify JavaScript	0.76s
Eliminate render-blocking resources	0.66s
Reduce unused CSS	0.48s

These suggestions can help your page load faster. They don't directly affect the Performance score.

DIAGNOSTICS

- ▲ Serve static assets with an efficient cache policy — 38 resources found
- ▲ Image elements do not have explicit `width` and `height`
- Avoid enormous network payloads — Total size was 2,812 KiB
- ○ Avoid chaining critical requests — 17 chains found
- ○ User Timing marks and measures — 2 user timings
- ○ Keep request counts low and transfer sizes small — 46 requests • 2,812 KiB
- ○ Largest Contentful Paint element — 2,900 ms
- ○ Avoid large layout shifts — 5 elements found
- ○ Avoid long main-thread tasks — 1 long task found
- ○ Avoid non-composited animations — 1 animated element found

More information about the performance of your application. These numbers don't directly affect the Performance score.

PASSED AUDITS (25)

Show

Figure 50 Google Lighthouse Page 2

92

Accessibility

These checks highlight opportunities to improve the accessibility of your web app. Only a subset of accessibility issues can be automatically detected so manual testing is also encouraged.

NAMES AND LABELS

- ▲ Form elements do not have associated labels
- ▲ Links do not have a discernible name

These are opportunities to improve the semantics of the controls in your application. This may enhance the experience for users of assistive technology, like a screen reader.

ADDITIONAL ITEMS TO MANUALLY CHECK (10) Show

These items address areas which an automated testing tool cannot cover. Learn more in our guide on conducting an accessibility review.

PASSED AUDITS (22) Show

NOT APPLICABLE (24) Show

83

Best Practices

USER EXPERIENCE

- ▲ Serves images with low resolution

GENERAL

- ▲ Browser errors were logged to the console

Figure 51 Google Lighthouse Page 3

- Detected JavaScript libraries
- ▲ Missing source maps for large first-party JavaScript

TRUST AND SAFETY

- Ensure CSP is effective against XSS attacks

PASSED AUDITS (10) Show

NOT APPLICABLE (1) Show

80

SEO

These checks ensure that your page is following basic search engine optimization advice. There are many additional factors Lighthouse does not score here that may affect your search ranking, including performance on Core Web Vitals. Learn more about Google Search Essentials.

CONTENT BEST PRACTICES

- ▲ Document does not have a meta description

Format your HTML in a way that enables crawlers to better understand your app's content.

CRAWLING AND INDEXING

- ▲ Links are not crawlable

To appear in search results, crawlers need access to your app.

ADDITIONAL ITEMS TO MANUALLY CHECK (1) Show

Run these additional validators on your site to check additional SEO best practices.

PASSED AUDITS (8) Show

NOT APPLICABLE (4) Show

Figure 52 Google Lighthouse Page 4

PWA

These checks validate the aspects of a Progressive Web App. Learn what makes a good Progressive Web App.

INSTALLABLE

- Web app manifest or service worker do not meet the installability requirements — 1 reason

PWA OPTIMIZED

- Does not register a service worker that controls page and `start_url`
- Is not configured for a custom splash screen Failures: No manifest was fetched.
- Does not set a theme color for the address bar.
  Failures: No manifest was fetched, No `<meta name="theme-color">` tag found.
- Content is sized correctly for the viewport
- Has a `<meta name="viewport">` tag with `width` or `initial-scale`
- Manifest doesn't have a maskable icon No manifest was fetched

ADDITIONAL ITEMS TO MANUALLY CHECK (3) Show

These checks are required by the baseline PWA Checklist but are not automatically checked by Lighthouse. They do not affect your score but it's important that you verify them manually.

- Captured at Aug 31, 2023, 8:28 PM GMT+1
- Initial page load
- Emulated Desktop with Lighthouse 10.3.0
- Custom throttling
- Single page load
- Using Chromium 116.0.0.0 with devtools

Generated by **Lighthouse** 10.3.0 | File an issue

Figure 53 Goole Lighthouse Page 5

8.4 Conclusion

In the development of this project, my primary goal was to design a microfinance-banking application. This application has the capability to allow for the creation of user profiles. What makes it unique is its admin panel, which is equipped with an extensive suite of control features. Administrators can manually create user profiles, delete existing ones, upload KYC documents, and even adjust the balance of accounts as needed.

While users of the application have the flexibility to execute transactions internally, it's worth noting that the system currently operates in isolation from real-world banks. It doesn't interface with the international SWIFT network. The standalone nature of the application a detailed history of all transactions, providing transparency and accountability which can be expanded in the future. One of the primary intentions behind this application was to emphasize the importance of developing a resilient server and payment portal as the first step. I later focused on integrating banking functionalities and due to time constrains a lot of functionalities was left out.

I wanted to showcase my technical expertise in cloud-based solutions. I've successfully deployed this application to the cloud, making full use of the CI/CD pipeline, showcasing not just the application's capabilities but also my proficiency in modern deployment strategies.

Certain non-functional requirements unfortunately had to be sidelined due to pressing time constraints. These included essential features such as the ability to reset passwords, effectively handle expired tokens, and ensure compatibility across mobile and tablet devices. While the foundation for scalability was laid, the full potential of leveraging a pay-as-you-use cloud services model hasn't been realized. Furthermore, the project would have greatly benefited from more comprehensive documentation of the codebase, which would facilitate better understanding and future modifications.

In terms of security, the application's foundation rests largely on Django's built-in modules, with a particular emphasis on utilizing crispy forms for input sanitation. While these built-in functionalities offer a good starting point for security, it's important to recognize that they are not entirely foolproof. Sole reliance on them could inadvertently introduce vulnerabilities, making the application potentially vulnerable to targeted exploits. Moving forward, it would be better to consider additional layers of security measures to boost the system's defences.

8.4.1 Functional Requirements

Table 7 User Management Requirements

User Management	**Status**
1. User should be able to register with email verification	Achieved
2. Users should be able to log in using their credentials.	Achieved
3. Users should be able to log out	Achieved
4. Users should be able to reset their passwords	Not Achieved

Table 8 Email Verification Requirements

Email Verification	**Status**

1. Upon Verification, users should receive an email verification link.	Achieved
2. Clicking the link should successfully verify the user’s email.	Achieved
3. Invalid or expired links should provide appropriate error messages.	Not Achieved

Table 9 Admin Interface Requirements

Admin Interface	**Status**
1. Admin should have a customised dashboard	Achieved
2. Admin should be able to perform CRUF operations.	Achieved
3. Admin should be able to manager user accounts	Achieved

Table 10 Deployment Requirements

Deployment	**Status**
1. The application should be deployable on cloud platforms.	Achieved
2. Database Migrations should be easily executable post-deployment	Achieved

8.4.2 Non-Functional Requirements

Table 11 Performance Requirements

Performance	**Status**
1. The application should respond quickly to user actions.	Partially – Achieved (refer to Google Lighthouse Report)
2. Database operations should be optimised and not introduce bottlenecks	Achieved

Table 12 Security Requirements

Security	**Status**
1. User passwords should be stored securely (hashed and salted)	Achieved
2. Email verification tokens should be securely generated	Achieved
3. The application should guard against common web vulnerabilities (SQL injection, CSRF, XSS, etc.).	Achieved

Table 13 Usability Requirements

Usability	**Status**
1. The user interface should be intuitive and user-friendly.	Deprioritized
2. Error Messages should be clear and guide users on what actions to take.	Deprioritized

Table 14 Availability Requirements

Availability	Status
1. The application should aim for high uptime.	Not Achieved (Current Deployment on Free – Tier Model)
2. Critical operations (like registration and login) should be reliable.	Achieved

Table 15 Scalability Requirement

Scalability	Status
1. The application should handle an increasing number of users and data.	Achieved

Table 16 Maintainability Requirements

Maintainability	Status
1. The codebase should be well-organised and modular.	Achieved
2. Proper documentation should be provided for developers.	Not Achieved

Table 17 Compatibility Requirements

Compatibility	Status
1. The application should be compatible with modern web browsers.	Achieved
2. The application should be responsive and work on both desktop and mobile devices.	Not Achieved

9 Future Work

In this following section, I will provide features that I have identified as potential enhancements for the system and the features and functionalities I could not implement within the project. These features, when integrated, can greatly boost the system's capabilities, and create a complete product. Due to time constraints, skill gap and ethical approval, I was unable to implement these functionalities.

9.1 Genuine Stakeholder

1. Engaging with a genuine stakeholder allows for the incorporation of customized features essential for bank operations.
2. Securing the necessary ethical clearances is vital for launching a bank that offers legitimate financial services.
3. It's crucial to furnish proper legal documentation and safeguard user rights. Loan terms, whether monthly, daily, or quarterly, should be determined based on government regulations and appropriate interest rates.

4. Compliance with regard to GDPR (General Data Protection Regulation).

9.2 Advanced KYC Verification

1. Instead of just image-based verification, I can implement a video KYC process where users record a short video, which can improve the authenticity of the verification.
2. Integrate Optical Character Recognition (OCR) tools to auto-fetch details from ID proofs, reducing manual data entry errors.

9.3 API Integration

1. Integrate with banking APIs to fetch real-time account balance, transaction history, and more.
2. For a seamless transaction experience, integrate with popular payment gateways.
3. Integration with popular mobile wallets for seamless transaction experience.
4. User login using direct gmail, apple ID, outlook and other major email providers.

9.4 Security Features

1. Introduce 2FA for an added layer of security during logins and critical transactions.
2. Implement fingerprint or facial recognition for mobile app versions.
3. Implement ReCaptcha API to avoid additional bot spam attacks.
4. Ability to implement user to reset password using email verification links.
5. Implement backend logic to serve invalid, expired links and to provide appropriate error messages for additional security.
6. Properly use. evn file and secret variables and hide secret keys from settings.py. Due to time constraints and error while deploying did not manage to hide away secrets properly.

9.5 UI/UX Features

1. Fix UI/UX for mobile viewing.
2. Create custom CSS and JavaScript tailored to the needs of the application.
3. User analytics to show user graphs and charts.
4. Better error messages and guidance.

9.6 Performance and Usability

1. Use Google's Lighthouse report recommendation and fix all audits failed.
2. Reduce overhead by using custom specialised cloud providers and provide tailored load balancer solutions.
3. Get ethical approval to conduct user survey on various features and services.

9.7 Maintainability and Compatibility

1. Documentation for developers to be created according to the ISO/IEC 25010:2011 standards.
2. Conduct extensive user tests for compatibility on all devices available such as mobile, tablets etc.

10 Reflection

While working on my dissertation and developing the Microfinance Banking and Payment system, I realised that 12 weeks is a short period of time. Managing part-time, working on the project and fighting my mental block took willpower, determination and resilience. Nevertheless, it has helped me develop my research, technical, and project management skills; the biggest would be my personal development.

10.1 What Went Well

The core reasoning behind developing the project was to create software that benefits my community back home and provides them with the technology to uplift them. The amount of ground I covered learning Django and Bootstrap was tremendous. I immersed myself in Django, acquiring substantial knowledge and skills applicable to this high-level Python Web framework. This experience has laid a solid foundation for me, enabling me to construct better web applications with an enhanced understanding of the framework's capabilities and functionalities.

My interaction with Git has not only been about understanding its basic functionalities but also extending to mastering advanced operations such as rolling back to previous commits and recovering to a working state. This skill has proven a safety net, allowing me to experiment without fear of irreversible errors while deploying and testing. I have also honed my ability to manage branches efficiently, a critical aspect of version control that facilitates parallel development without conflicts. The skill to successfully incorporate fixes and manage pull requests has made me a better defensive programmer. The journey into cloud technologies has been brilliant. Notably, I managed to integrate CI/CD pipelines within Railway.app, which culminated in the successful deployment of a functioning application. This accomplishment marked a significant milestone in my skills, showcasing cloud technologies' potential to enhance development workflows. As an aspiring DevOps Engineer, this skill would be invaluable in my career. I also ventured into AWS S3, where I succeeded in hosting static files and storing user images in an S3 bucket. This experience has expanded my understanding of cloud storage solutions, showcasing their versatility and efficiency in web application development.

My project also saw the integration of external APIs like SendGrid, which played a pivotal role in enhancing the project's security and facilitating email communication. This integration has not only boosted the functionality of my project but has also opened avenues for exploring other potent APIs that can add value to my project.

10.2 What Went Wrong

During the project, I embarked on the ambitious task of incorporating a working loan facility with variable interest rates. However, I needed to pay more attention to the depth of research required to navigate the complex terrain of ethical and governmental regulations surrounding such a feature. Consequently, I had to abandon this initiative midway, recognising the need for ethical approval and extensive consultation with legal and stakeholder advisors. This experience has been a stern reminder of the importance of thorough research and preparedness before venturing into complex features, especially when intersecting with sensitive areas such as finance and ethics.

I envisioned delivering a project of substantial scale and complexity within a three-month window, an ambition that, in retrospect, needed to be fully aligned with my capabilities at the time. This miscalculation highlighted the necessity of realistic goal setting, considering the multifaceted nature of project development and the time required for each phase. While using the AWS S3 bucket to store Jazzmin (admin panel) static files, I encountered a stumbling block. A persistent 403 authentication error, which stemmed from Jazzmin's authentication issues with AWS, proved to be a significant hindrance. This forced me to pivot and adopt a different strategy, serving jazzmin files from the Railway.app while retaining AWS for user data storage. This experience has made me realise the importance of flexibility in project development, adapting quickly to unforeseen technical challenges and finding alternative solutions. Despite my intentions to utilise the test cases provided by the Django framework, I could not implement them as planned. Consequently, I had to revert to Black Box testing, which consumed considerable time. This setback emphasised the critical role of testing in project development and the need to allocate sufficient time and resources for this phase to ensure a robust and reliable application.

As an international student who ventured into an entirely different educational sphere and teaching style for my master's program, embarking on a dissertation of this scale was uncharted territory. This endeavour marks my adventure into independently crafting an entire client-server web application. Despite the challenges, I have a profound pride in the milestones I have reached and the knowledge I garnered throughout this project's journey. Much like a young Padawan evolving into a Jedi Master, I am confident in my ability to navigate future projects more adeptly, ensuring a more aligned and successful outcome.

References

BBC News, 2022. India's loan scams leave victims scared for their lives. [online] Available at: https://www.bbc.com/news/world-asia-india-59451589 [Accessed 6 September 2023].

Business Standard, 2021. Banks in India report loan fraud worth Rs 5 trn, SBI's amount largest. [online] Available at: https://www.business-standard.com/article/finance/banks-in-india-report-loan-fraud-worth-rs-5-trn-sbi-s-amount-largest-122032200030_1.html [Accessed 6 September 2023].

Capterra, 2023. Mambu Reviews. [online] Available at: https://www.capterra.com/p/122029/Mambu/reviews/ [Accessed 16 August 2023].

Daniel, G., 2018. Software Product Quality Metrics. In: , pp.346-374. doi: 10.1002/9781119134527.CH16.

Django Software Foundation, 2023. Django documentation. [online] Available at: https://docs.djangoproject.com/en/3.2/ [Accessed 23 August 2023].

Esparza, J. and Grande, V., 2023. Black-box Testing Liveness Properties of Partially Observable Stochastic Systems. arXiv. Cornell University. Available at: https://doi.org/10.48550/arxiv.2303.03292.

Fogler, R., Cohen, I. and Peled, D., 2023. Accelerating Black Box Testing with Light-Weight Learning, pp.103-120. doi: 10.1007/978-3-031-32157-3_6.

Farr, F. and Contributors, 2023. Django Jazzmin: A jazzy skin for the Django Admin-Interface. [online] GitHub. Available at: https://github.com/farridav/django-jazzmin.

Finastra, 2023. About Us. [online] Available at: https://www.finastra.com/about/our-story [Accessed 16 August 2023].

Garcia, M. and Martinez, E., 2017. Women's Empowerment through Microfinance Initiatives: A Global Perspective. Gender & Development, 25(3), pp.355-370.

GitHub, Inc., 2023. About GitHub. [online] Available at: https://docs.github.com/en/github/getting-started-with-github/about-github [Accessed 23 August 2023].

Google, 2023. Lighthouse. [online] Available at: https://developers.google.com/web/tools/lighthouse.

Gridfiti, 2023. 13 Best & Most Aesthetic Note-Taking Apps in 2023 (Desktop & Mobile). [online] Available at: https://gridfiti.com/aesthetic-note-taking-apps/ [Accessed 26 August 2023].

Heimicke, J., Kaiser, S. and Albers, A., 2021. Agile product development: an analysis of acceptance and added value in practice. Procedia CIRP, 100, pp.768–773. doi: 10.1016/j.procir.2021.05.046.

Jones, R. and Patel, S., 2019. Financial Inclusion and Poverty Alleviation: A Comparative Analysis of Microfinance Institutions. International Journal of Finance Studies, 7(4), pp.112-128.

Lighthouse Developers, n.d. Lighthouse. [online] GitHub. Available at: https://github.com/GoogleChrome/lighthouse.

Mambu, 2023. About Us. [online] Available at: https://www.mambu.com/en/about-us/ [Accessed 16 August 2023].

Kumar, N.J. et al., 2023. Black Box Testing Design Intended for Vehicle Surveillance and Tracking. [online] doi: 10.1109/icidca56705.2023.10099824. [Accessed 29 August 2023].

NerdWallet, 2023. What Is Mambu?. [online] Available at: https://www.nerdwallet.com/article/banking/mambu-review [Accessed 16 August 2023].

Otto, M. and Thornton, J., 2023. About Bootstrap. [online] Available at: https://getbootstrap.com/docs/5.0/about/overview/ [Accessed 23 August 2023].

Riesener, M., Doelle, C., Perau, S., Lossie, P. and Schuh, G., 2021. Methodology for iterative system modeling in agile product development. Procedia CIRP, 100, pp.439–444. doi: 10.1016/j.procir.2021.05.101.

Robinson, J., 2016. The Microfinance Revolution: Sustainable Finance for the Poor. Washington, DC: World Bank.

Singh, K., 2021. Agile Methodology for Product Development: A Conceptual Study. International Journal of Recent Technology and Engineering, 10(1), pp.209–215. doi: 10.35940/ijrte.a5899.0510121.*

SQLite Development Team, 2023. About SQLite. [online] Available at: https://www.sqlite.org/about.html [Accessed 23 August 2023].

Williams, L. and Nguyen, T., 2021. Microfinance in Developing Economies: Trends and Impacts. Journal of Global Economic Dynamics, 5(1), pp.78-92