Final Project Report: Plutus

CS-157A: Database Management Systems

Team Members:

- Atishay Jain - 015962713 - atishay.jain@sjsu.edu
- Chandramouli Iyer - 015650063 - chandramouli.iyer@sjsu.edu
- Vishwesh Mashruwala - 016042546 - vishwesh.mashruwala@sjsu.edu
- Het Tikawala - 015947997 - het.tikawala@sjsu.edu
- Devam Sanjay Sheth - 017404218 - devamsanjay.sheth@sjsu.edu

Submission Date: May 10th 2024

Project Overview

In today’s financial ecosystem, a significant number of consumers fail to take advantage of available cashback offers, leaving potential savings untapped. Our project, "Plutus," addresses this issue by utilizing a sophisticated database management system to deliver notifications about cashback opportunities directly to users as they conduct their financial transactions.

The main goal of Plutus is to enhance user engagement and financial management through seamless integration with users' daily banking activities. By ensuring that users receive timely, relevant cashback notifications, Plutus not only aids in financial savings but also enriches the overall user experience.

To achieve this, Plutus combines a user-friendly frontend interface with a robust backend system. The web application is designed to be intuitive and efficient, allowing for quick notifications and minimal user interaction. The backend, developed with Spring Boot, handles data processing and communication with the MySQL database, ensuring that all user transactions are monitored and matched with the best cashback offers available. The frontend, built using React, displays these offers to users in a clear and accessible manner.

Data flow between the frontend and backend is facilitated through a REST API, ensuring that the system remains responsive and capable of handling data transmission effectively. This setup allows Plutus to operate smoothly, providing a reliable service that can scale with user demand.

Plutus leverages advanced technology to solve a practical problem, making financial benefits more accessible and convenient for users. The project aims to bridge the gap between complex database interactions and user-friendly application design, delivering a solution that is both technologically sound and easy to use.

Goals and Description of the Application

The primary goal of "Plutus" is to enhance the financial management experience of users by automating the process of identifying and notifying them about relevant cashback offers. By leveraging structured data stored in a robust database system, Plutus ensures that users receive timely notifications of cashback opportunities that are specifically tailored to their shopping habits and payment methods.

Application Description:

At the heart of Plutus is a comprehensive database designed to manage detailed information about users, their financial instruments, and associated cashback offers. The application is structured around several key tables that maintain data about users, banks, cards, product categories, products, offers, and various types of relationships among these entities.

- Users and Cards: The application stores detailed user profiles including their financial instruments. This data includes information about the cards users hold, which are linked to specific banks and card types. This setup allows Plutus to personalize offers based on the financial products used by the customer.
- Products and Offers: Plutus maintains a detailed categorization of products and the offers associated with them. By correlating offers with product categories and specific products, the system can accurately trigger notifications when users engage in transactions that qualify for cashback.
- Offer Management: Offers are managed dynamically with attributes such as type, validity, and a detailed description. These offers are linked not only to products but also directly to card types and companies, allowing for multifaceted offer strategies that can be tailored to user behaviors and preferences.
- Notifications: The application implements a notification system that connects users with relevant offers through a streamlined process. When an offer relevant to a user's profile or transaction is identified, a notification is generated and sent to the user, ensuring they are aware of potential savings instantaneously.

Plutus is designed to operate efficiently within a complex relational database environment. By interlinking users, transactions, and offers, the system can quickly determine the most beneficial

cashback opportunities for each user, based on transaction data. This not only maximizes financial benefits for users but also drives higher engagement and customer satisfaction.

Application/Functional Requirements and Architecture

Functional Requirements:

A. Notifications: Plutus must fetch and notify users of cashback offers as they perform transactions or when they are detected near participating stores.
B. User Authentication: Secure user authentication system to manage access and maintain user data security.
C. Responsive User Interface: The application must provide a responsive, intuitive interface that is accessible on various devices, displaying offers and notifications effectively.
D. Automatic Offer Matching: Automatically match offers to user transactions based on the cards and stores they use.
E. Data Security and Privacy: Implement comprehensive security measures to protect sensitive user data and transaction details.

Architecture:

Frontend Development (React): Plutus utilizes React for its frontend development to leverage React's robust ecosystem and its capability for building dynamic and responsive user interfaces. The choice of React also facilitates the development of a single-page application (SPA), which enhances the user experience with smooth transitions and immediate updates without the need to reload the page. React’s component-based architecture allows for modular and maintainable code, where each component can be developed and tested independently.

- State Management: State management is handled using React Context or Redux to manage the states across multiple components, ensuring that the state is synchronized across the entire application without prop-drilling.
- Routing: React Router is used for handling in-app routing to different views such as user profiles, notification lists, and settings.

- User Interface: The use of Material-UI or similar libraries provides a set of ready-to-use components that are both aesthetically pleasing and functionally effective, speeding up the development process while maintaining consistency across the application.

Backend Development (Spring Boot): The backend of Plutus is developed using Spring Boot, chosen for its extensive support for developing RESTful services quickly and its large array of configurable options. Spring Boot facilitates the rapid setup and configuration of applications, which is crucial for a project with a complex domain model like Plutus.

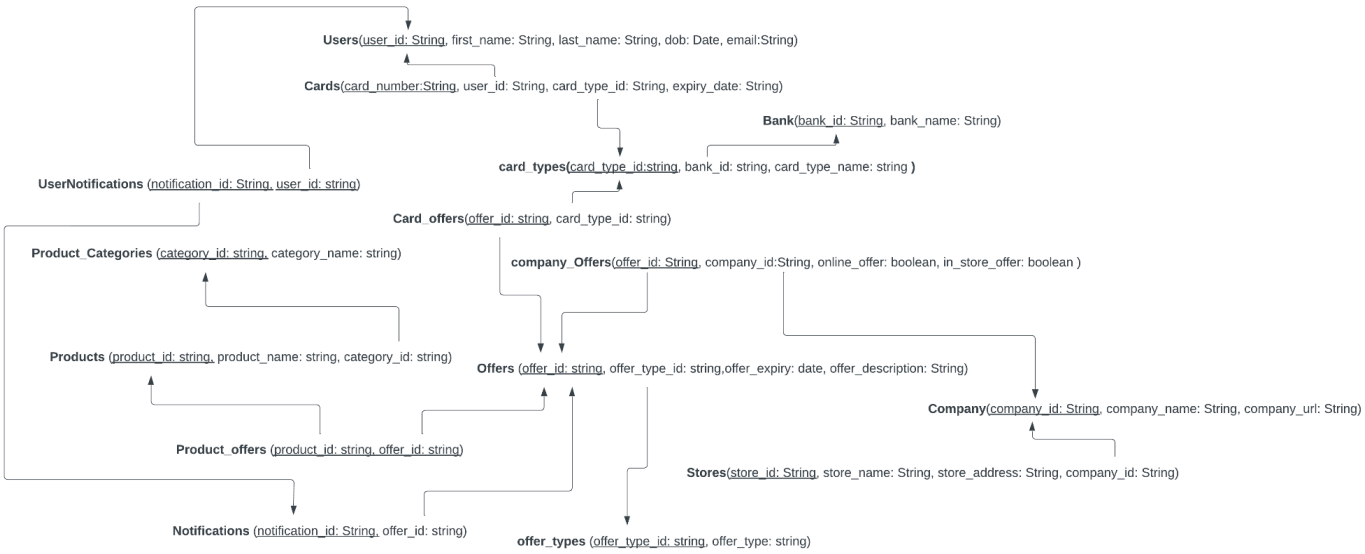
A. Data Handling: Spring Data JPA is utilized for database interactions, which simplifies the implementation of database repositories and reduces boilerplate code significantly. It automates the mapping between Java objects and database tables and provides intuitive methods for querying the database.
B. Security: Spring Security is integrated to handle authentication and authorization, securing REST endpoints, and ensuring that data is accessible only to authenticated users.
C. Business Logic: Business services are implemented to handle the core logic of matching offers to transactions, generating notifications, and managing user profiles and data.

Communication (REST API with JSON): Communication between the frontend and backend is managed through a RESTful API, with JSON as the data interchange format. This setup is chosen for its simplicity and effectiveness in handling asynchronous communications, which is essential for an application like Plutus.

A. API Design: The API endpoints are designed to be RESTful, ensuring that they are intuitive and resource-oriented. This includes standard HTTP methods (GET, POST, PUT, DELETE) to interact with the resource endpoints.
B. Data Format: JSON is used for sending and receiving data due to its lightweight nature and easy integration with both React and Spring Boot, facilitating fast data serialization and deserialization.
C. Error Handling: Robust error handling mechanisms are implemented to provide clear error responses for API calls, improving the debugging process.

ER Data Model and Database Design

(the ER Data Model)



The ER model for Plutus is structured to represent the complex relationships between various entities such as users, banks, cards, products, offers, and stores. This model facilitates the tracking of relationships such as which user holds which cards, which offers apply to which products, and which stores are running specific promotions.

- Entities: Key entities include Users, Cards, Banks, Offers, Products, Stores, and Notifications. These are connected through relationships that allow for the application to efficiently organize and retrieve data.
- Relationships: For example, a User is linked to multiple Cards, and each Card is linked to specific Offers. This design allows the system to quickly determine which offers are relevant to transactions made by the user with particular cards.

Database Design

The database is designed to support this ER model using MySQL, a relational database management system known for its reliability and efficiency. The schema is designed to optimize data retrieval and updates, which are critical for the functionality of Plutus.

- Normalization: Tables are normalized to BCNF to reduce redundancy and improve data integrity. This involves decomposing tables to eliminate functional dependencies, a process that simplifies data maintenance and enhances query performance. The process of normalisation is further discussed in the report.
- Indexes: Indexes are used to speed up data retrieval processes, particularly for frequently accessed data such as user credentials, current offers, and transaction records. For example, indexes on user_id in the Users table or offer_id in the Offers table ensure quick lookups.

Data Flow from MySQL to Spring Boot and React Frontend

Data flows through the system starting from MySQL, passing through the Spring Boot backend, and finally being displayed on the React frontend. Here's how this process works:

Data Retrieval from MySQL: When a transaction is detected, the Spring Boot application queries the MySQL database to fetch relevant user and card information. Concurrently, it retrieves any applicable offers based on the transaction details such as location, store, and card type used.

Data Processing in Spring Boot: The backend processes this data, applying business logic to match offers with user transactions. This may involve complex queries joining multiple tables such as Cards, Offers, and Users. Once the relevant offers are identified, they are formatted into a JSON structure suitable for transmission over the network.

Data Transmission to React Frontend: The processed data is sent to the React frontend via a REST API. The API ensures that data is securely transmitted using HTTPS, returning JSON-formatted responses to the frontend. The frontend then parses this JSON data and dynamically updates the UI to display the offers to the user. This interaction is managed without reloading the page, providing a seamless user experience.

Normalisation

The Process

1. **First Normal Form (1NF)**

- Tables: All Plutus database tables
- Criteria: No repeating groups or arrays; all entries are atomic.
- Verification: All tables in the schema have columns with atomic values and no multi-valued attributes, thereby satisfying 1NF.

2. **Second Normal Form (2NF)**

- Tables: Users, Banks, Card_Types, Cards, Product_Categories, Products, Offer_Types, Offers, Product_Offers, Card_Offers, Companies, Company_Offers, Stores, Notifications, User_Notifications
- Criteria: All attributes in each table must be fully functionally dependent on the whole of a composite primary key (if any).
- Verification: Each table's non-primary key attributes are dependent on the primary key, confirming 2NF compliance. No partial dependencies on composite keys are present as most keys are simple (single attribute).

3. **Third Normal Form (3NF)**

- Tables: As listed above.
- Criteria: No transitive dependencies exist between non-key attributes and the primary key.
- Verification: Each attribute is dependent only on the primary key, not on other non-key attributes, thus upholding 3NF.

4. **Boyce-Codd Normal Form (BCNF)**

- Focus Areas: Potential overlaps between candidate keys and proper superkeys.
- Challenges: Ensuring every determinant is a candidate key.
- Verification & Adjustments:
- Users, Banks, Product_Categories, Offer_Types, Companies, Stores: These tables each contain a straightforward primary key with no complex dependencies, confirming they are already in BCNF.

- Card_Types, Cards, Products, Offers, Product_Offers, Card_Offers, Company_Offers, Notifications, User_Notifications: Each of these tables is reevaluated for potential hidden dependencies and determinant non-candidate key issues.
- Cards: Initially, user_id and card_type_id could seem like joint determinants for attributes in other tables. However, since card_number is the primary key and the only determinant, this table is in BCNF.
- Company_Offers: offer_description was duplicative and dependent on offer_id. By ensuring that descriptions are only stored in the Offers table, we remove redundant data and uphold BCNF.
- Card_Types, Product_Offers, Card_Offers: Foreign keys that form composite primary keys with other attributes are checked to ensure they are part of candidate keys and not mere determinants of non-primary attributes. Adjustments were made to ensure all determinants are candidate keys.

Recommendations

1. **Indexing Foreign Keys**

- Justification: Indexes on foreign key columns can drastically improve the performance of JOIN operations, which are common in relational databases involving multiple tables.
- Impact: Faster query response times, especially for complex joins involving large datasets.

2. **Consistent Data Types**

- Justification: Ensuring consistent use of data types across similar fields prevents errors and inefficiencies. For instance, using VARCHAR for all postal code fields accommodates international formats.
- Impact: Reduces potential errors during data integration and improves application compatibility with international standards.

3. **Optional Fields Review**

- Justification: Reviewing and standardizing the optionality of fields can lead to better data consistency and less erroneous data entry.
- Impact: Streamlines data entry processes and simplifies backend logic, improving overall data quality.

4. **Audit Trails for Sensitive Data**

- Justification: Implementing audit trails for tables containing sensitive information like Users and Cards ensures compliance with data security regulations and aids in forensic analysis.
- Impact: Enhances security measures, provides accountability, and supports regulatory compliance.

5. **Normalization Beyond BCNF**

- Justification: Applying 4th Normal Form (4NF) and 5th Normal Form (5NF) where applicable can further reduce redundancy and improve data integrity in cases of complex relationships.
- Impact: Ensures that the database structure is optimally designed to handle intricate data scenarios without compromising on performance.

6. **Timestamps for Record Tracking**

- Justification: Adding creation and modification timestamps to each table helps in tracking changes over time and is essential for maintaining historical data accuracy.
- Impact: Provides better insights into data lifecycle, supports audit requirements, and enhances data traceability.

Major Design Decisions

In developing Plutus, critical design decisions were implemented to optimize functionality and scalability:

1. **Different Types of Offers as Subsets of One Common Offer Relation:** We adopted a unified Offers table model to manage diverse types of offers effectively. This approach simplifies the management of offer data and enhances the system's scalability:
    a. Unified Offer Table: Centralizes common attributes like offer IDs and descriptions, simplifying maintenance and queries.
    b. Subtype Relationships: Additional tables such as Product_Offers, Card_Offers, and Company_Offers extend the Offers table, storing specific attributes and linking back via foreign keys. This allows for efficient, targeted queries based on user transactions and activities.

    c. Flexibility and Extensibility: The structure supports various offer types and is adaptable for future enhancements, enabling the addition of new tables without impacting the existing database structure.
2. **User Notifications Separate from Common Notification Relation:** To enhance personalization and manage notifications effectively, User_Notifications are maintained separately from the general Notifications table:
    a. Personalized Notifications: The User_Notifications table links notifications directly to users based on eligible offers, ensuring relevance and personalization.
    b. Scalable Notification Management: This separate structure supports scalability and efficient querying, maintaining performance as user numbers grow.
    c. Enhanced Data Integrity and Privacy: Separation helps protect user data, enhancing security by controlling data access and reducing exposure.

Implementation Details

The implementation of Plutus involved a series of structured steps from database schema design to frontend development, ensuring robustness and efficiency at each stage. Here's how each component was developed:

Logical Schema Designing:

Data Relationships and Structures: We began by defining the logical schema, outlining how data elements interact within the system. This phase included the structuring of entities such as Users, Offers, Cards, and their interrelations, which are crucial for the core functionalities of Plutus.

Physical Schema Designing:

Optimizing Storage and Performance: The logical schema was then translated into a physical schema that details the storage of data in the MySQL database. This step involved specifying table structures, indexes, and storage parameters to optimize data retrieval and performance.

Normalization: Reducing Redundancy and Enhancing Integrity: We applied normalization principles up to Boyce-Codd Normal Form (BCNF) to minimize redundancy and enhance data

integrity. This process ensured that the database remains efficient and reliable, reducing the chance of data anomalies.

Implementation in MySQL:

Table Creation and Relationship Definition: The physical schema was implemented in MySQL by creating tables and defining relationships. This included setting primary and foreign keys, constraints, and triggers to maintain data accuracy and consistency.

Optimized Query Performance: Indexes were strategically created on frequently accessed columns to speed up query performance, essential for the application's functionality.

Backend Development in Java:

Server Setup and API Integration: The backend was developed using Java, leveraging frameworks like Spring Boot for setting up RESTful APIs and managing server-side logic.

Database Integration: Java's robust ecosystem facilitated efficient integration with MySQL, handling complex queries and transactions necessary for processing user requests and managing data flows.

Business Logic Implementation: Core application logic, including offer matching and notification handling, was implemented in Java. This allowed for robust processing capabilities and seamless management of user interactions.

Frontend Development in React:

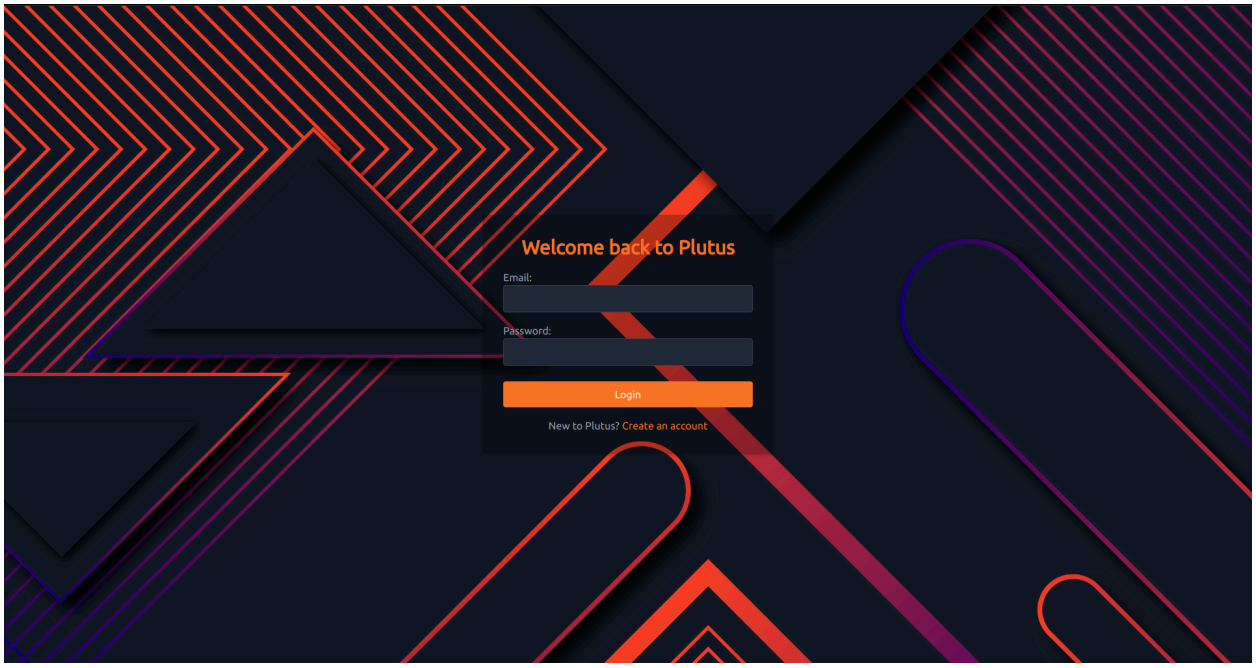
Interactive UI Components: The frontend was built using React, focusing on creating responsive and interactive UI components. This involved developing components that are not only visually appealing but also efficient in fetching and displaying data dynamically from the backend.

React's Ecosystem: Utilizing tools and libraries available in React’s ecosystem, such as Redux for state management and Axios for HTTP requests, enhanced the frontend’s capability to handle state and communicate effectively with the backend.

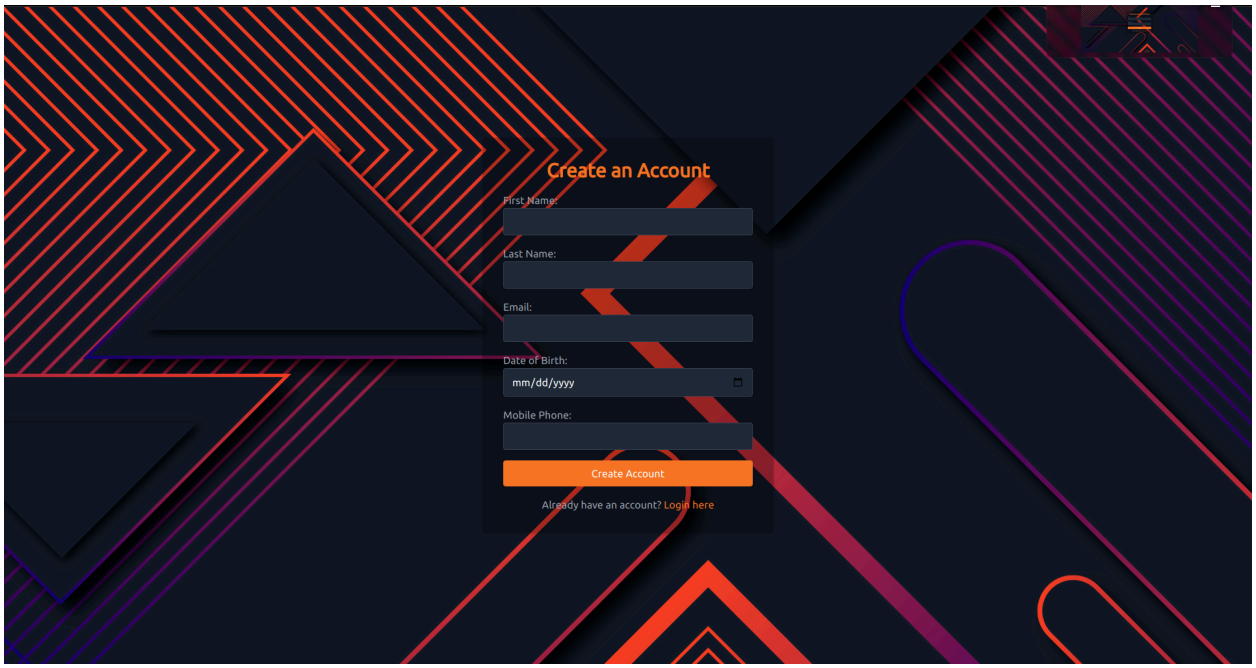
Overall System Integration:

Seamless Data Flow: The entire architecture was designed to ensure seamless flow of data from the MySQL database, through the Java backend, and to the React frontend. This integration supports the dynamic requirements of Plutus, enabling data updates and interactions across the application.

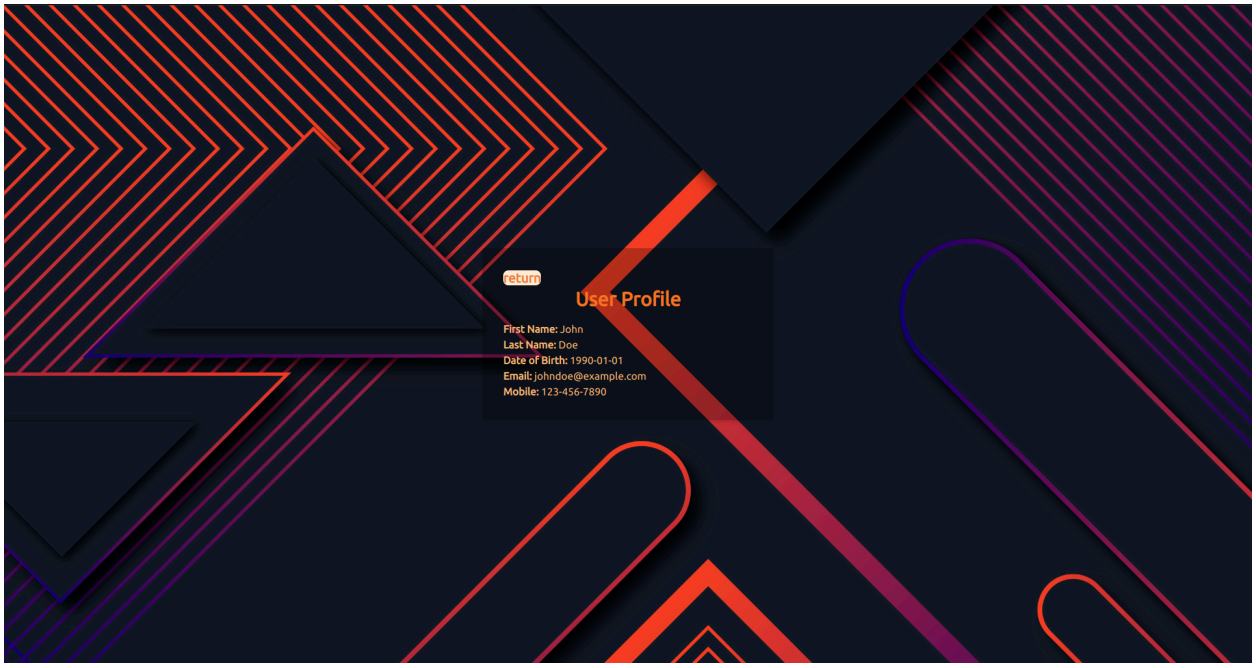
Demonstration of Example System Run



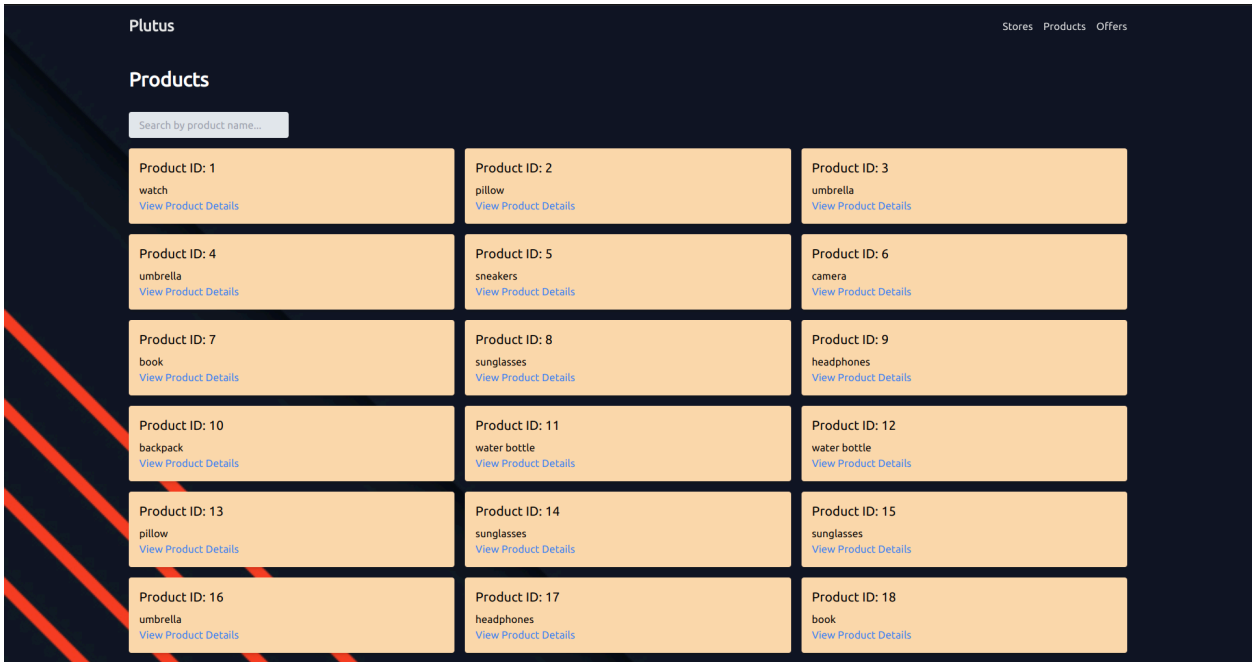
Provides a secure interface for users to log into their accounts. The form fields for email and password are straightforward, emphasizing ease of use and accessibility.



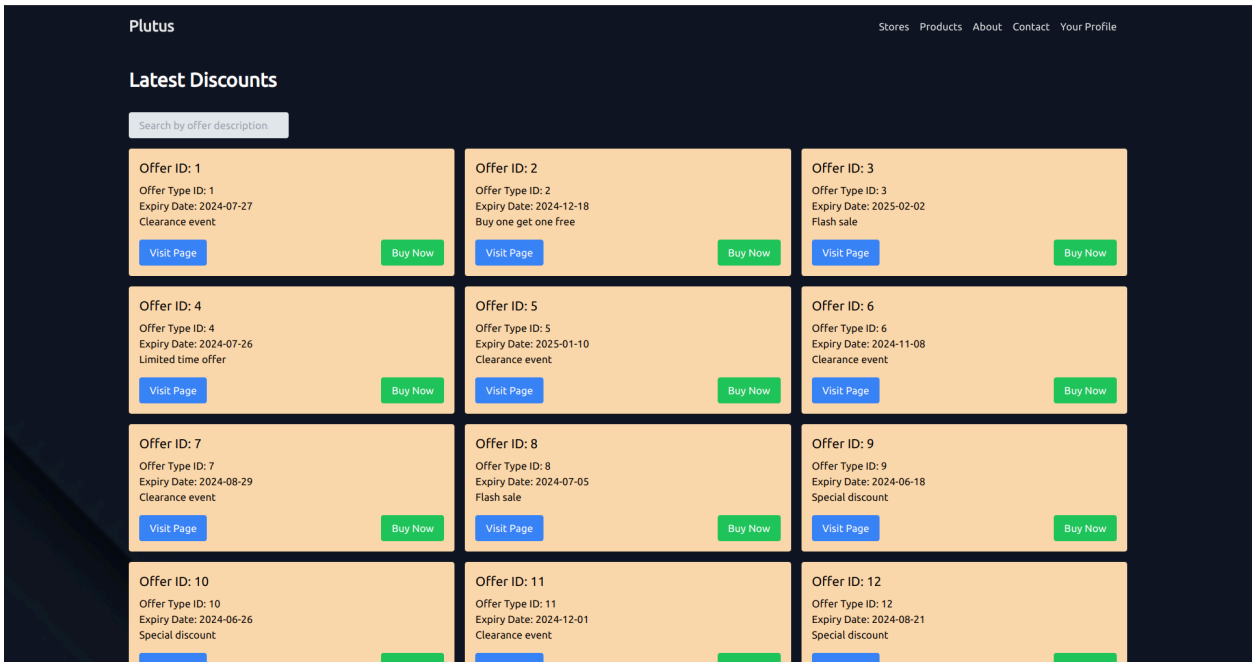
Offers new users the ability to create an account by entering details such as first name, last name, email, date of birth, and mobile phone. This page ensures a user-friendly account registration process.



Shows the user's profile details including first name, last name, date of birth, email, and mobile number, with a stylish, minimalistic design that enhances user experience.



Lists all products in a grid layout where each card provides the product ID, name, and a link to view more details about the product. This page allows users to browse through various items available for purchase.



Displays a grid of current offers available to users, with each card showing the offer ID, type, expiry date, and a brief description. Users can navigate to detailed pages or directly engage with offers via 'Visit Page' or 'Buy Now' buttons.

Lessons Learned

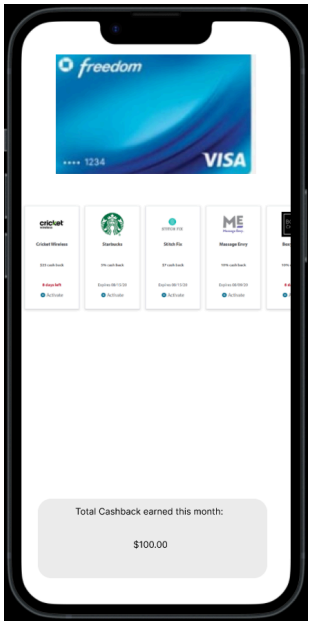
Importance of Schema Design: The critical role of thorough logical and physical schema design in supporting system functionality and performance was a key takeaway. Proper normalization and indexing proved essential for efficient data handling and retrieval.

Integration of Technologies: Integrating multiple technologies (React, Spring Boot, MySQL) required careful planning and execution. This project underscored the importance of seamless integration across different layers to ensure smooth data flow and user interactions.

User-Centric Design: Focusing on user experience from the outset—considering how users would interact with the interface and what would make their experience more intuitive—helped shape a more engaging and accessible platform.

Possible Improvements

Mobile Application Development: Recognizing the increasing reliance on mobile devices, transitioning Plutus into a comprehensive mobile application is a significant next step. This will not only improve accessibility for users on-the-go but also ensure that the platform remains competitive and relevant in a mobile-first world.



Enhanced Security Features: Based on our experiences and user feedback, implementing additional layers of security protocols, such as two-factor authentication and advanced encryption, will be crucial in safeguarding user data further.

User Interface Upgrades: While the current interface is functional and user-friendly, ongoing refinements are planned to enhance navigation and interactions. This includes optimizing UI design and functionality to make the application even more intuitive.

Conclusion

The development of Plutus has resulted in a robust and user-friendly platform that enhances the financial experience by providing, relevant cashback notifications directly to users. The project's success is largely attributable to its strong database design, effective integration of frontend and backend technologies, and a keen focus on user experience.

Furthermore, Plutus has set a solid foundation for a scalable and secure platform that effectively meets the needs of its users. The lessons learned from this project will guide future enhancements, ensuring that Plutus continues to evolve in line with technological advancements and user expectations.