**Team Information for International Payments Portal Development**

**Team Members:**

* Aline de Sousa
* Rutendo Chapfika
* Hazel Kaseke
* Fortune Madoda

Github: <https://github.com/APDS-Banking/Banking-App.git>  
Video: https://drive.google.com/drive/folders/1hMAmne4trnLxowDCSIxjlO9fXitkj9Pi?usp=sharing

**Task Division:**

**1. Front-End Development (React)**

**Rutendo Chapfika: Registration & Login Page**

* **Registration Page:**
  + Implemented customer registration with fields for full name, ID number, account number, and password.
  + Applied secure input validation and client-side encryption using bcryptjs to hash passwords before sending them to the server.
  + Implemented authorisation using regular expressions for fields like names, account numbers, and passwords.
* **Login Page:**
  + Allowed customers to log in using their username, account number, and password.
  + Ensured secure handling of login credentials and communication with the API.

**Aline Sousa: Customer Dashboard & Payment Page**

* **Dashboard:**
  + Developed the customer dashboard, displaying a greeting and options for entering payment details like amount, currency, SWIFT provider, and recipient account information.
* **Payment Page:**
  + Implemented the "Pay Now" functionality and ensured smooth data collection before sending data to the backend API.
  + Applied input validation using RegEx patterns for payment amounts, SWIFT codes, and account numbers.

**2. Back-End Development (Node.js/Express API)**

**Hazel Kaseke: API Development & Security**

* **API Development:**
  + Developed API endpoints for customer registration, login, and payment processing.
  + Implemented hashing and salting of passwords using bcrypt, securely storing them in the database.
  + Used SSL (HTTPS) to encrypt all communication between frontend and backend.
  + Secured the backend by using Express Brute for brute force prevention and Helmet.js to protect against common vulnerabilities.
* **Security Measures:**
  + Ensured secure transaction handling, storing all payment data in the database (MongoDB/MySQL).

**Fortune Madoda: Security & DevSecOps**

* **Security:**
  + Worked on SQL injection, cross-site scripting (XSS), and cross-site request forgery (CSRF) attacks.
  + Implemented security best practices to ensure that API endpoints and data flows were encrypted and tamper-proof.
* **DevSecOps Pipeline:**
  + Set up a DevSecOps pipeline using CircleCI for continuous integration and deployment.
  + Utilized SonarQube for code quality checks and security analysis using tools like MobSF and ScoutSuite.
  + Developed automated testing procedures to ensure security and performance.

**Collaboration & Support:**

* The team leveraged tools such as ChatGPT to assist in task breakdown and resolving challenges.
* Collaboration and peer support were essential throughout the project:
  + **Aline** received support from **Hazel** and **Rutendo** for frontend integration.
  + **Rutendo** received help from **Hazel** for the login page.
  + **Aline** was in charge of the documentation page**.**

**Lab Guide: Secure Customer International Payments Portal**

**1. Overview**

This lab covers the front-end and back-end creation of a secure customer payments interface using React and Node.js. With strong security measures including password hashing, input validation, and SSL encryption, the portal concentrates on processing client registration, login, and payment transactions.

**2. Prerequisites**

* React and Node.js installed.
* MongoDB for database setup
* Familiarity with JavaScript
* Tools: Postman, VS Code, Git, Eslint

**3. Lab Objectives**

* Implement safe payment processing and customer authentication.
* Use salting and hashing to secure your passwords.
* Defend the system from XSS, and SQL Injection threats.
* Enable SSL-based HTTPS communication.

**Step-by-Step Guide**

**Step 1: Set Up the Development Environment**

1. **Install Dependencies**:
   * We ensured the required dependencies were installed for both the frontend (React) and backend (Node.js/Express).
2. **Create React Project**:
   * We initialized a new React project, set up the frontend structure, and started the development server.
3. **Backend Setup**:
   * We navigated to the server directory and initialized the backend project, setting up Node.js and Express for handling backend services.
4. **SSL Certificates (Optional)**:
   * For development purposes, we generated self-signed SSL certificates to enable HTTPS communication between the frontend and backend.

**Step 2: Frontend Development (React)**

1. **Customer Registration & Login Pages (Rutendo’s Task)**:
   * We implemented the customer registration form with fields like full name, ID number, account number, and password.
   * To enhance security, we used client-side password hashing before sending data to the server.
   * We ensured that inputs such as names, account numbers, and passwords were validated using appropriate regular expressions for security purposes.
2. **Routing Setup in the Application**:
   * We configured routing in the React application to navigate between different pages, such as registration, login, home, and payment.
3. **Customer Dashboard & Payment Page (Aline’s Task)**:
   * We developed the customer dashboard where users can enter payment details such as amount, currency, provider (SWIFT code), and recipient account information.
   * We implemented the "Pay Now" button that triggers payment processing and integrated it with the backend API for transaction handling.
   * We ensured that all input fields, including payment amounts and SWIFT codes, were validated using regular expressions.

**Step 3: Backend Development (Node.js & Express)**

1. **API Development (Hazel’s Task)**:
   * We built an API using Node.js and Express to handle customer registration, login, and payment processes.
   * Secure practices like password hashing (with bcryptjs) were used to store passwords securely in the database.
   * We ensured that communication between the frontend and backend was encrypted via HTTPS, utilizing SSL certificates.
2. **Security Measures (Fortune’s Task)**:
   * Various security measures were implemented to prevent brute-force attacks.
   * We secured all API endpoints with CORS and validated inputs at both the frontend and backend levels.
   * The system was protected from CSRF (Cross-Site Request Forgery) attack.

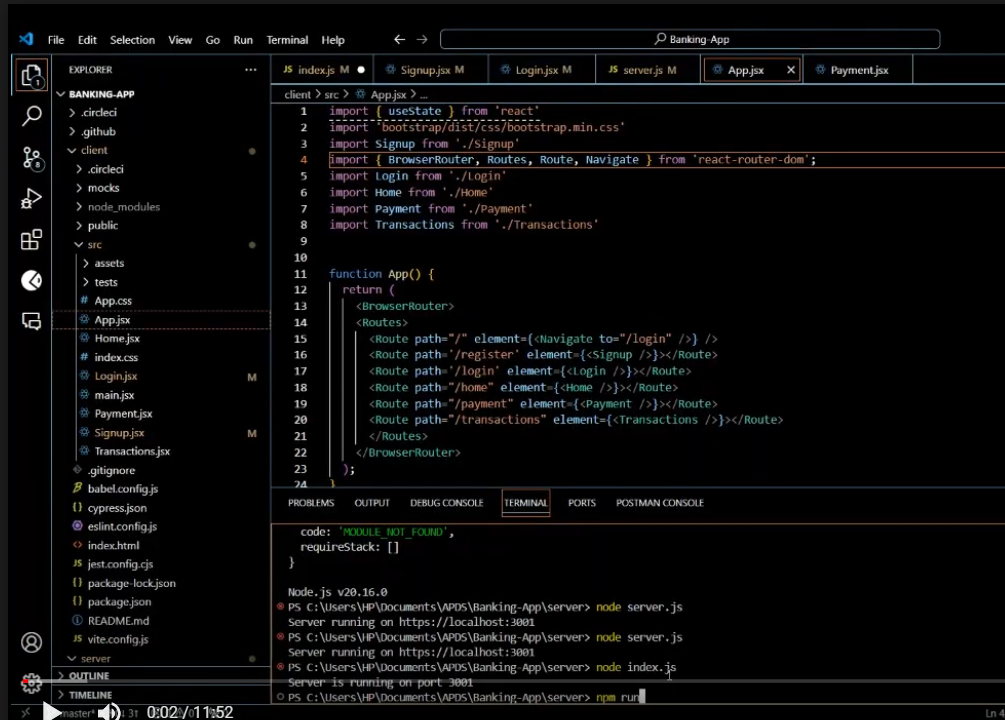
**Step 4: DevSecOps & Continuous Integration**

1. **Set Up CircleCI Pipeline (Fortune’s Task)**:
   * We used CircleCI for automating testing and deployment of the application.
   * The pipeline was configured to include security checks using tools like SonarQube for code quality analysis and MobSF or ScoutSuite for vulnerability scanning.
   * We regularly ran tests to detect vulnerabilities like XSS (Cross-Site Scripting) and SQL Injection.

**Step 5: Testing & Deployment**

1. **Run the Application**:
   * We launched the frontend and backend services, testing the connection between them to ensure smooth communication.
2. **Testing**:
   * We used Postman or a similar tool to test API endpoints for registration, login, and payment processing.
   * Security testing was performed by simulating potential attacks such as XSS and SQL Injection.
3. **Deployment**:
   * The application was deployed to cloud platforms like AWS, ensuring that SSL encryption was enabled to serve traffic securely over HTTPS.

**Screenshots:**

****

In this part we can see the routing setup for the application. It demonstrates how different routes are configured for various pages. These routes allow users to navigate through the various parts of the application seamlessly.

**A screenshot of a computer screen

Description automatically generated**

Here the application is already running, It displays a simple "Register" form interface with fields for entering user information. The form includes the following:

* Name: Enter Name
* Email: Enter email
* Account Number: Enter account number.
* ID Number: Enter ID number.
* Password: Enter password

There is a "Register" button at the bottom, along with a "Login" option for users who already have an account.

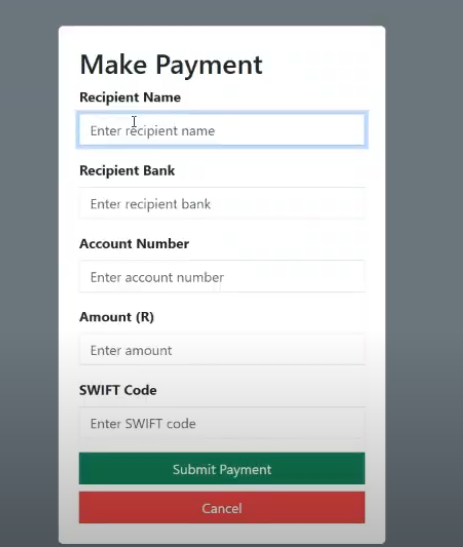
A screenshot of a computer

Description automatically generated

The screenshot below shows a customer dashboard greeting the user, Below the greeting, it displays the account balance.

There are three buttons:

* A button to "Make International Payment.”
* A button to "View Transactions"
* A button to "Logout"



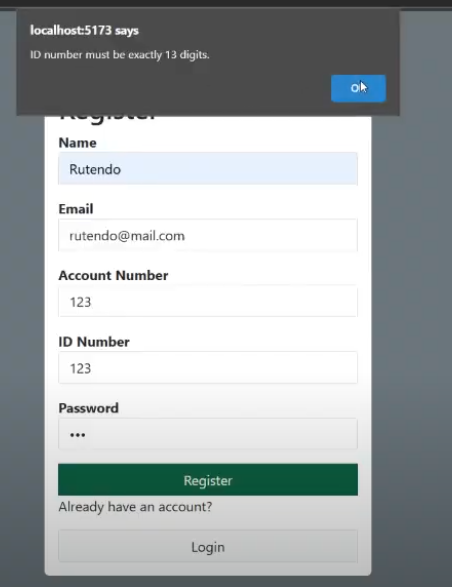
The image shows a page intended for making an international payment. It includes the following input fields:

* Recipient Name
* Recipient Bank
* Account Number
* Amount (R)
* SWIFT Code

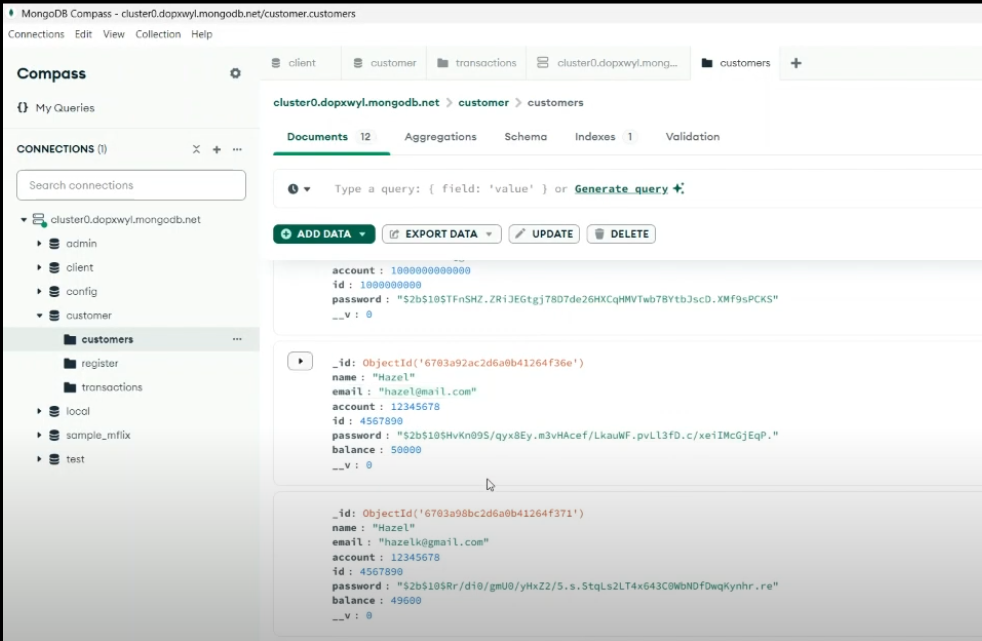
Below the fields, there are two buttons:

* A "Submit Payment" button to confirm and send the payment.
* A "Cancel" button to cancel the process.

The form is designed to collect essential payment details before processing the transaction.

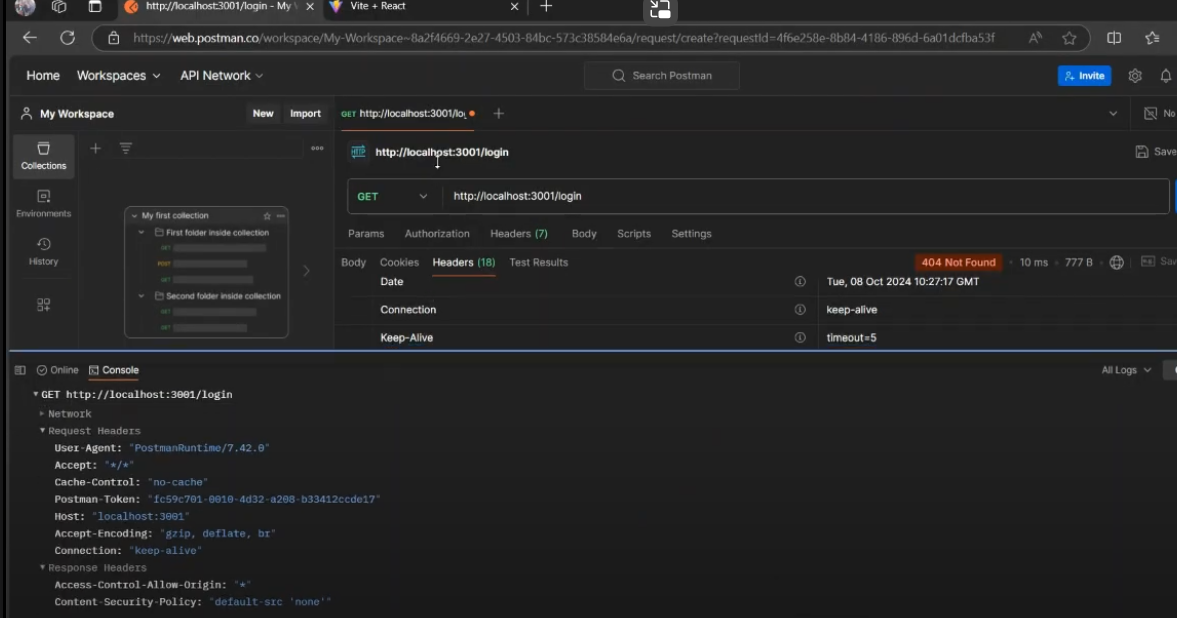


The image shows a "Register" form being filled out, but an error message has appeared saying: "ID number must be exactly 13 digits." This indicates that the user, has entered an ID number which does not meet the requirement of exactly thirteen digits. The system is preventing registration until the correct ID number is entered.

****

The image shows MongoDB Compass, a graphical user interface for interacting with MongoDB databases.

This collection stores customer information, including sensitive data like account details and balances. The interface provides options for adding, exporting, updating, or deleting data, and various queries can be executed on the collection.

****

The image shows a Postman interface, where a GET request is being made to http://localhost:3001/login. The result of the request is shown on the console.

**The link for our portal:**

<https://github.com/APDS-Banking/Banking-App.git>

**Conclusion**

This guide's methods assist in building a safe consumer payments interface that implements security features including input validation, SSL encryption, password hashing, and defense against online threats. As long as you adhere to the rules, you may create a reliable and expandable payments platform.

**References:**

1. **ChatGPT.** (2024),<https://chatgpt.com/share/6703ad69-7ac4-8008-a40b-3c2433cf2a80>
2. **Tech Coffee Break** (2024),”Login and Signup using MERN Stack,Mongo, Express, React and Node Authentication”, <https://youtu.be/cvPIeaogheI?si=tiWYZZH8nVrJzEMW>
3. Koval, V)2020),”My Journey of Understanding Angular vs React. Medium”, <https://medium.com/@vlad.koval/my-journey-of-understanding-angular-vs-react-0c0a5b8373c8>
4. LogRocket, (2022),”Using Helmet in Node.js to secure your application”, LogRocket Blog,<https://blog.logrocket.com/using-helmet-node-js-secure-application/>
5. **Funda Coder** (2021*),*“How to make Static Payment Mode Option in Checkout Form in vb.net”*,* “YouTube”, <https://youtu.be/pSDXq03ut4o?si=Bi-WlyfcUrOHMReR>
6. Integrate Payments (2020),”Javascript Payment Integration Framework for Web Developers”,”YouTube”, <https://youtu.be/Pu_KaJ9hpO0?si=yw26qHjES0Lt349E>
7. Stack Overflow,”Stack Overflow”, <https://stackoverflow.com/>
8. OpenSSL(2024), “OpenSSL req - certificate request and certificate generating utility”, <https://docs.openssl.org/1.1.1/man1/req/>
9. Kaspersky(2024), “What is an SSL certificate – Definition and Explanatio”, <https://www.kaspersky.com/resource-center/definitions/what-is-a-ssl-certificate>
10. Kamunya, T*.(*2023), *“*10 Angular UI Libraries to Create a World-Class User Experience”,”Geekflare”,<https://geekflare.com/angular-ui-libraries/>