

# T-A-D-D

## APDS README

---

Student Name Tawfiq Mohamed

Student Number ST10042129

Student Name Devania Chetty

Student Number ST10049331

Student Name Dillon Wernich

Student Number ST10038986

Student Name Alison Rees

Student Number ST10038986

## **Overview**

The purpose of this POE for Application Development Security (APDS7311) is to develop and implement a secure international payment site for a bank. The project is split into two parts: a Customer Portal where customers can start and track these transactions (task 2), and an Employee Portal where bank staff verify and manage foreign transactions (task 3). The system architecture must prioritize the implementation of security measures such as SSL for data in transit, hashing and salting for password security, and RegEx whitelisting for input validation (OWASP Foundation, 2023).

Security planning includes safeguarding against vulnerabilities such as DDoS attacks, SQL injection, and cross-site scripting. With a pipeline in place to perform security tests and security tools (ScoutSuite and MobSF) to assess the hosting environment and mobile application, the project also makes use of DevOps principles. This approach aims to ensure that both portals meet stringent operational resilience and data security standards.

<b>Overview .....</b>	<b>1</b>
<b>Table of Contents .....</b>	<b>2</b>
<b>Features .....</b>	<b>3</b>
<b>Technologies .....</b>	<b>4</b>
<b>Installation .....</b>	<b>5</b>
<b>Backend Overview .....</b>	<b>6</b>
<b>Database Setup .....</b>	<b>6</b>
<b>Frontend Overview .....</b>	<b>7</b>
<b>How To Use The Software .....</b>	<b>8</b>
<b>MongoDB .....</b>	<b>14</b>
<b>CircleCi .....</b>	<b>16</b>
<b>SonarQube .....</b>	<b>17</b>
<b>Dev Ops .....</b>	<b>18</b>
<b>Functional Requirements .....</b>	<b>19</b>
<b>Non Functional Requirements.....</b>	<b>20</b>
<b>FAQ's .....</b>	<b>21</b>
<b>Developer Information .....</b>	<b>22</b>
<b>References .....</b>	<b>23</b>

## Features

- **User Authentication:** Secure user registration and login are made possible by password hashing with JWT and bcrypt for session management.
- **Payment Processing:** Securely handle payments.
- **Safe Data Storage:** Use appropriate encryption and hashing algorithms to safely store user, payment, and transaction data in the MongoDB user, payments, and transactions database (Figures 1.6–1.8) (Anderson, 2020).
- **Rate Limiting:** To guard against brute-force attacks and guarantee API security, use express-rate-limit.
- **HTTPS Support:** HTTPS allows for safe data transfer between the frontend and backend.
- **Real-Time Monitoring:** Use MongoDB Atlas to monitor database metrics and performance.
- **Environment Configuration:** When managing sensitive configurations, such as database URIs and API keys, use dotenv.
- **Secure password:** comparison and hashing can be achieved by using bcrypt (Biryukov, Dinu & Khovratovich, 2016).
- **Token-based Authentication:** For safe, stateless user sessions, use JSON Web Tokens (JWT).

## Technologies

The project uses the following key technologies:

- **Backend:** Node.js, Express.js
- **Frontend:** HTML, CSS, JavaScript
- **Database:** MongoDB (Atlas)
- **Security:** HTTPS, Password Hashing (bcrypt), JWT for authentication
- **API Rate Limiting:** express-rate-limit
- **Environment Configuration:** dotenv for managing environment variables
- **Token Authentication:** JSON Web Tokens (JWT)

## **Installation**

### **1. Clone the Repository:**

Clone the repository from GitHub using the following command:

```
git clone https://github.com/IIE-Varsity-College-ST10061721/APDS7311-POE.git
```

*Alternatively, you can download and unzip the folder to your desired directory.*

### **2. Install Dependencies:**

Navigate to the project directory and install the required dependencies:

```
cd employee-payments-portal
```

```
npm install
```

### **3. Run the Backend Server:**

Move to the backend directory and start the backend server:

```
cd backend
```

```
node server.js
```

### **4. Run the Frontend:**

Navigate to the frontend directory and start the frontend service:

```
cd ../frontend
```

```
npm start
```

### **5. Access the Application:**

Once both the backend and frontend are running, open your browser and navigate to the appropriate URL to access the application.

These steps will set up the "Employee Payments Portal" locally for development or testing purposes.

## Backend Overview

The backend of the Employee Payments Portal, which provides RESTful API endpoints to handle crucial functions like user identification, payment processing, and secure connections to the MongoDB database, was built using Node.js and Express.js.

### Key Features:

- **User Authentication:** manages user registration and login processes, employing bcrypt for safe password hashing and comparing plain passwords to the hashed password that has been preserved (Ferraiolo & Kuhn, 1992).
- **JWT Authentication:** uses JSON Web Tokens (JWT) to guarantee safe access to protected routes and securely maintain user sessions.
- **Payment Processing:** facilitates seamless financial transactions within the application by offering API methods for creating and viewing payments.
- **MongoDB Integration:** MongoDB Atlas is a cloud-based database solution that securely stores data while guaranteeing high availability and dependability.
- **Security:** The backend employs encryption methods for sensitive data and makes sure that HTTPS is used for secure connection.

## Database Setup

MongoDB Atlas is the cloud database service used by the Employee Payments Portal. Three essential collections are used by the program to store data:

- **Users Collection:** contains hashed passwords, user roles, account numbers, and other user data including email addresses and usernames. This is employed for user management and authentication. When a user registers, their account number is automatically generated.
- **Payments Collection:** retains information about payments, such as transaction amounts, payment IDs, payment status, and other relevant information.
- **Transaction Collection:** In a somewhat different structure from the payment collection, this keeps track of user transactions. Using the payee account number associated with the user, this saves the transaction.

To set up the database:

1. **Create a MongoDB Atlas Cluster:** Set up a MongoDB Atlas cluster on your account.
2. **Get Your Connection URI:** After creating the cluster, get the connection string (URI) for your MongoDB instance.
3. **Update the .env File:** Place the MongoDB Atlas connection URI in the .env file under the MONGODB\_URI variable to allow the application to connect to the database.

Ensure that the collections are properly configured and indexed in MongoDB Atlas for optimal performance.

## Frontend Overview

The frontend of the Employee Payments Portal is a user-friendly React application designed to provide a faultless experience for both customers and employees (W3C, 2017). It facilitates the creation of payments, secure user management, and transaction tracking.

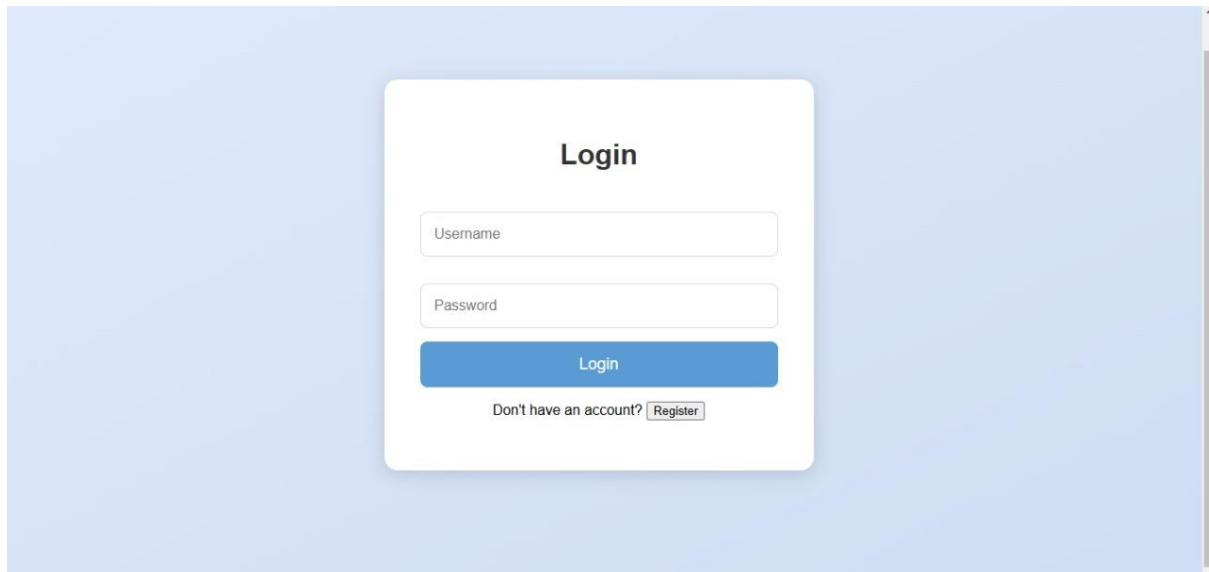
### Key Features:

- **Registration and Login Pages:**
  - **Customers** can securely register and log in to access their accounts.
  - **Employees** can log in directly because they are automatically assigned the "employee" role after pre-registering on MongoDB.
- **Employee Dashboard:**
  - Upon login, employees are directed to the dashboard where they can manage payments, verify them, and submit them to SWIFT.
- **Payment Interface for Customers:**
  - Customers can create new payments using an easy-to-use payment form.
- **Payment Management for Employees:**
  - Employees can view, handle, and confirm consumer payments.
  - Employees can send payments to SWIFT for processing once they have verified.
- **Transaction History:**
  - Both customers and employees can view a list of previously processed payments, pulled from the backend database.
- **Secure Authentication:**
  - Consumers access their accounts using secure login information and registration.
  - To ensure safe access to the employee dashboard, employees use their pre-registered login credentials.

The application, which was developed using React, ensures smooth, real-time backend interfaces for data submission and retrieval. Additionally, it provides a clear and easy-to-use interface for both user roles. Customers can easily create payments, while staff members have the necessary tools to manage and validate payments.



## How To Use The Software



*Figure 1.1*

The purpose of the login screen is to control customer and staff authentication. Customers can use the system after completing the registration process, however staff members have pre-registered accounts (see pre-registered account details below) and log in directly using their login credentials.

### **Customer Login:**

On the login screen, users must input their username (which could be an email address or a unique username) and password. Customers must first register an account before they can log in. Upon registration, they will be required to provide a username and password. At least one special character (such as!, @, or #), at least one number (such as 1, 2, or 3), caps, and small letters must all be included in the password, which must have a character count of three to twenty. The username can contain both special characters and digits, but it must be between three and twenty characters. After registration, customers can log in by entering their login information on the login screen.

### **Example Customer Login Details:**

- **Username:** user3@example.com
- **Password:** securePass!45

### Employee Login:

Employees are not obliged to register for accounts on the portal, though. To access their pre-registered accounts, they use the username and password that were generated with the employee role (Ferraiolo & Kuhn, 1992). After logging in, staff members can use the Employee Dashboard to manage payments, monitor transactions, and perform administrative duties. The employee login process is the same as the customer login process, using credentials that are provided to them during setup.

### Example Employee Login Details:

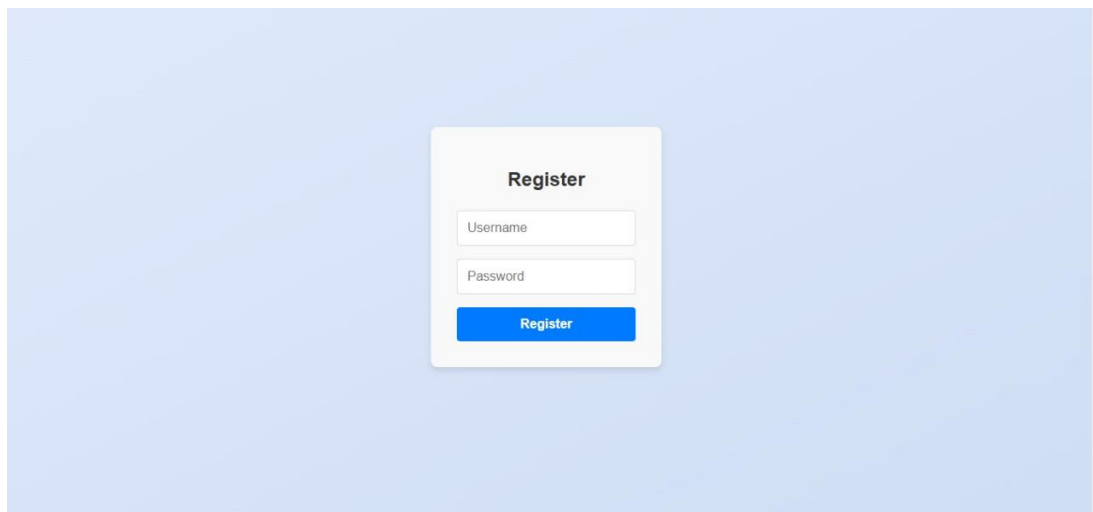
- **Username:** tester
- **Password:** AnotherPass!78

### Security Features:

To enforce strict security standards, REGEX validation is used to safeguard the login process. The following guidelines are examined to make sure the username and password fields are following them:

- **Username:** Can contain special characters and digits, but it must be between three and twenty characters.
- **Password:** It must have at least one capital letter, one number, and one special character, and it must be between three and twenty characters long.

This guarantees that when users enter the portal, both clients and staff utilize safe, correctly formatted login credentials.



The image shows a registration form titled "Register". It has two input fields: "Username" and "Password". Below the "Password" field is a blue button with the text "Register". The form is centered on a light blue background.

*Figure 1.2*

To access the system, users can create a new account on the Register Screen. Workers are pre-registered with specific duties and qualifications. Customers must complete the registration process to create a valid account. After that, they can use their newly created login information to log in.

### **Customer Registration:**

By choosing the Register option on the login page, customers can access the registration form. The following details are needed on the form:

- **Username:** The username must be between three and twenty characters long, however it may include special characters and numbers.
- **Password:** There are security standards that must be met by the password's complexity and strength. It must be between three and twenty characters long, contain at least one special character (such as !, @, or #), and at least one numeric digit (such as 1, 2, or 3).

To create an account, the client can fill out the registration form. After successfully registering, the user can access their dashboard by providing their login information.

### **REGEX Validation:**

REGEX is used to validate the password and username fields to make sure the entered data adheres to the necessary format:

- **Username:** The username can contain special characters and digits, but it must be between three and twenty characters.
- **Password:** The password must contain at least one special character and one number, and it must be between three and twenty characters long.

### **Example Customer Registration Details:**

- **Username:** user3@example.com
- **Password:** securePass!45

Customers can access the functionalities of the application by logging in with their credentials after successfully completing the registration process.

The screenshot shows a web interface titled "Payments". On the left, there is a form with four input fields: "Amount", "Currency (e.g., USD)", "SWIFT Code", and "Recipient Account". Below these fields is a blue button labeled "Submit Payment". On the right, there is a section titled "Payment History" which currently displays the text "No payments found.".

*Figure 1.3*

Customers can enter and submit transaction data for a new payment on the Customer Payments page. Its user-friendly layout makes it simple for clients to follow the steps involved in making payments.

#### **Payment Details Section:**

In the **Payment Details** section, customers can fill in the required information for a new payment. This includes:

- **Amount:** The amount of money to be transferred.
- **Currency:** The currency of the payment (e.g., USD, EUR, ZAR AUE).
- **SWIFT Code:** The SWIFT code of the recipient's bank, which ensures that the payment is routed correctly.
- **Recipient Account:** The account number of the recipient where the funds will be deposited.

Customers can start the payment process by clicking the Submit button once they have filled out all the necessary information. As a result, the system will generate a new payment record, which will be handled appropriately.

#### **Payment History Section:**

A list of all prior payments made by the client is shown in the second section of the page, Payment History. "No payments found" will appear at the beginning of the section if the users is new to the system, signifying that no transactions have been completed yet. A list of previous transactions will automatically appear in this section as the customer makes payments.

Each transaction will display the following details (see figure 1.4):

- **Amount:** The payment amount.
- **Currency:** The currency in which the payment was made.
- **SWIFT Code:** The SWIFT code associated with the transaction.

- **Recipient Account:** The account number to which the payment was made.

This history allows customers to track their past transactions and ensures they have a clear view of their payment history within the system.

The screenshot displays a web interface for payments. On the left, under the heading "Payments", there is a form with four input fields: "Amount", "Currency (e.g., USD)", "SWIFT Code", and "Recipient Account". Below these fields is a blue "Submit Payment" button. A confirmation message "Payment created successfully!" is visible below the button. On the right, the "Payment History" section shows five transaction cards. Each card lists the Amount, Currency, SWIFT Code, and Recipient Account.

Payment History			
Amount: 1000	Amount: 500	Amount: 250	
Currency: ZAR	Currency: ZAR	Currency: ZAR	
SWIFT Code: HYD5647FR	SWIFT Code: HAG5647FYG	SWIFT Code: HAG5647FYG	
Recipient Account: 5647653426	Recipient Account: Test	Recipient Account: John	
Amount: 400	Amount: 400		
Currency: ZAR	Currency: ZAR		
SWIFT Code: AS43DR567	SWIFT Code: HAG5647FYG		
Recipient Account: TEST	Recipient Account: John		

*Figure 1.4*

A thorough record of the user's previous transactions is shown in the Payment History section of the Customer Payments page. Customers can track and examine their prior payments in this part, which acts as a crucial reference point.

#### **Payment History Section:**

In this section, each payment is listed with the following details:

- **Amount:** The total value of the transaction.
- **Currency:** The currency used for the transaction (e.g., USD, EUR).
- **SWIFT Code:** The SWIFT code of the recipient's bank, ensuring the payment is routed correctly.
- **Recipient Account:** The account number of the recipient that received the funds.

Every item in the Payment History gives a clear picture of the transaction, giving clients convenient access to their previous payment information. Five previous payments are given in the example to show the function of the payment history.

#### **User Interface:**

The UI of the payment system is simple and easy to use. The Payment History area appears next to the Payment Details section, where clients enter fresh payment details (such as the amount, currency, SWIFT code, and recipient account). As additional funds are received, this section will automatically update, adding each transaction to the list below.

A confirmation message notifies the consumer that their payment has been successfully created after it has been submitted. The new payment and its related information will then be instantly shown in the Payment History section.

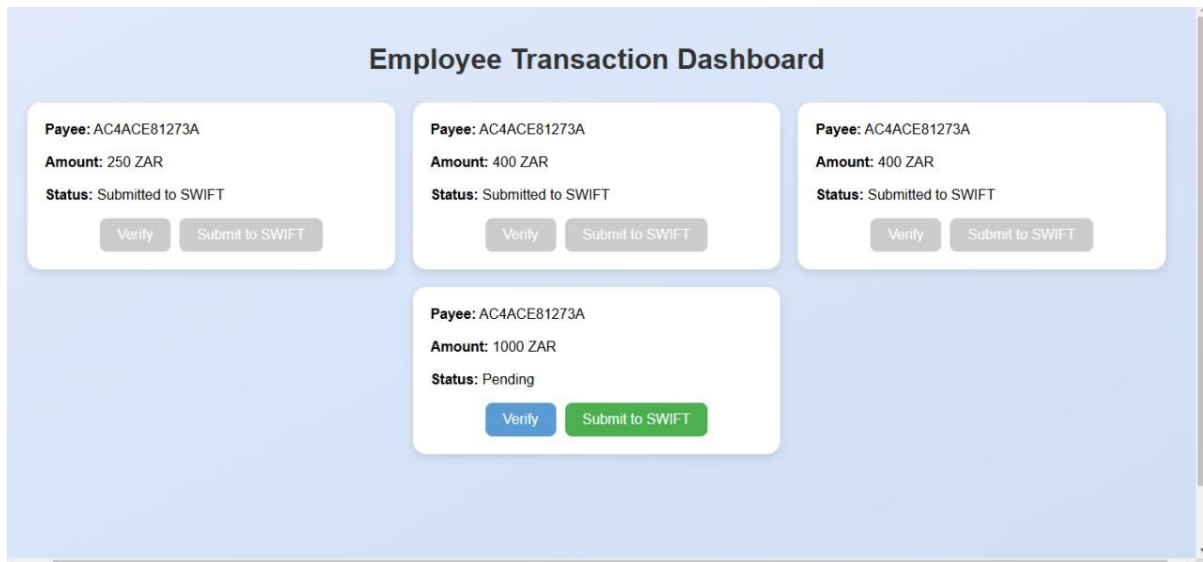


Figure 1.5

The purpose of the Employee Transaction Dashboard is to make managing and processing financial transactions easier. Employees can view and engage with customer transactions, confirm customer information, and complete payments using this interface. With distinct visual cues for each transaction's progress, the dashboard is well-structured and simple to use.

#### Transaction Cards:

Each transaction is displayed in its own card, providing detailed information at a glance:

- **Payee (Customer Account Number):** The account number of the customer who is the recipient of the payment.
- **Amount:** The total value of the transaction.
- **Status:** The current stage of the transaction, indicating whether it is in progress or has completed certain steps. Possible statuses include:
  - **Pending:** The transaction is awaiting verification or approval.
  - **Verified:** The transaction has been confirmed by the employee.
  - **Submitted to SWIFT:** The transaction has been finalised and sent to the SWIFT network for processing.

#### Action Buttons:

Each transaction card includes two key action buttons:

- **Verify (Blue Button):** This button is used to confirm the details of the transaction. Once clicked, the employee can ensure that all the payment details are accurate and complete. The status is updated as needed when this is selected.
- **Submit to SWIFT (Green Button):** This button is used to finalise and submit the transaction to the SWIFT network for processing. It is only enabled when the transaction is in a **Pending** status. The status is updated as needed when this is selected.

## User Interface Design:

The light blue background of the Employee Transaction Dashboard's simple, easy-to-use design, keeping to the theme of the whole app. To the layout's ease of use, staff members can handle several transactions at once without feeling overwhelmed. In addition to improving usability, the colour-coded action buttons (green for submission, blue for verification) help staff members navigate every stage of the transaction process.

## MongoDB

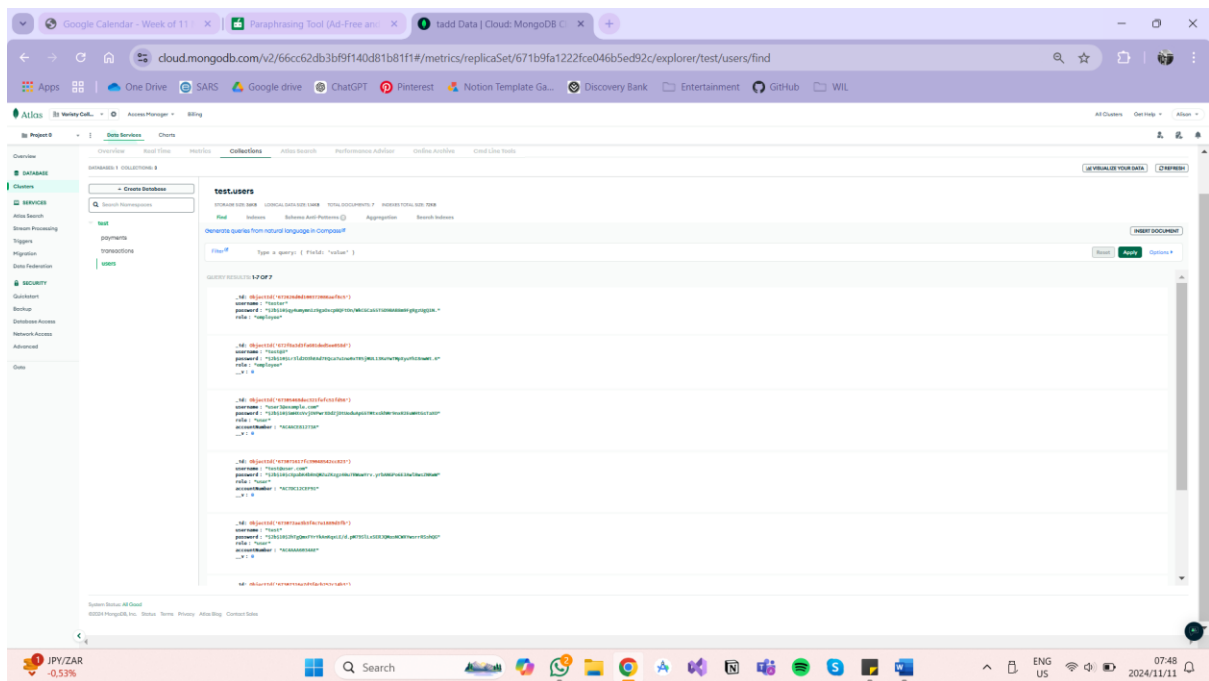


Figure 1.6

This demonstrates that users have been successfully saved to the MongoDB database (MongoDB Atlas, 2023).

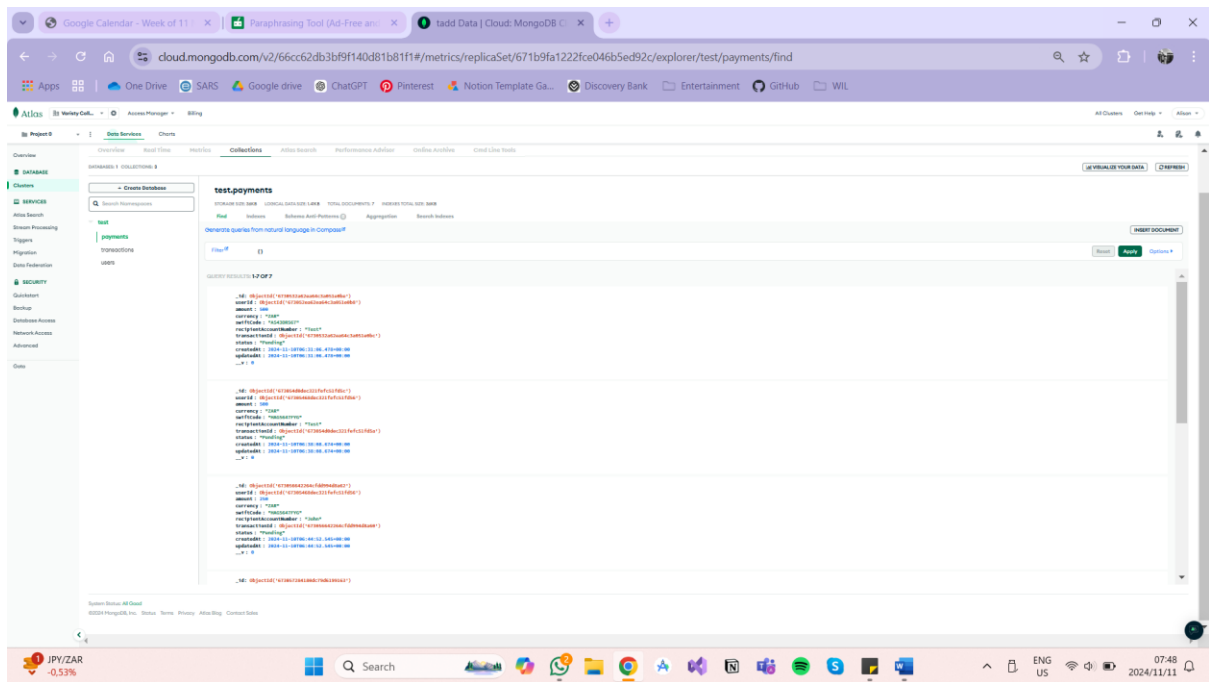


Figure 1.7

This demonstrates that payments have been successfully saved to the MongoDB database (MongoDB Atlas, 2023).

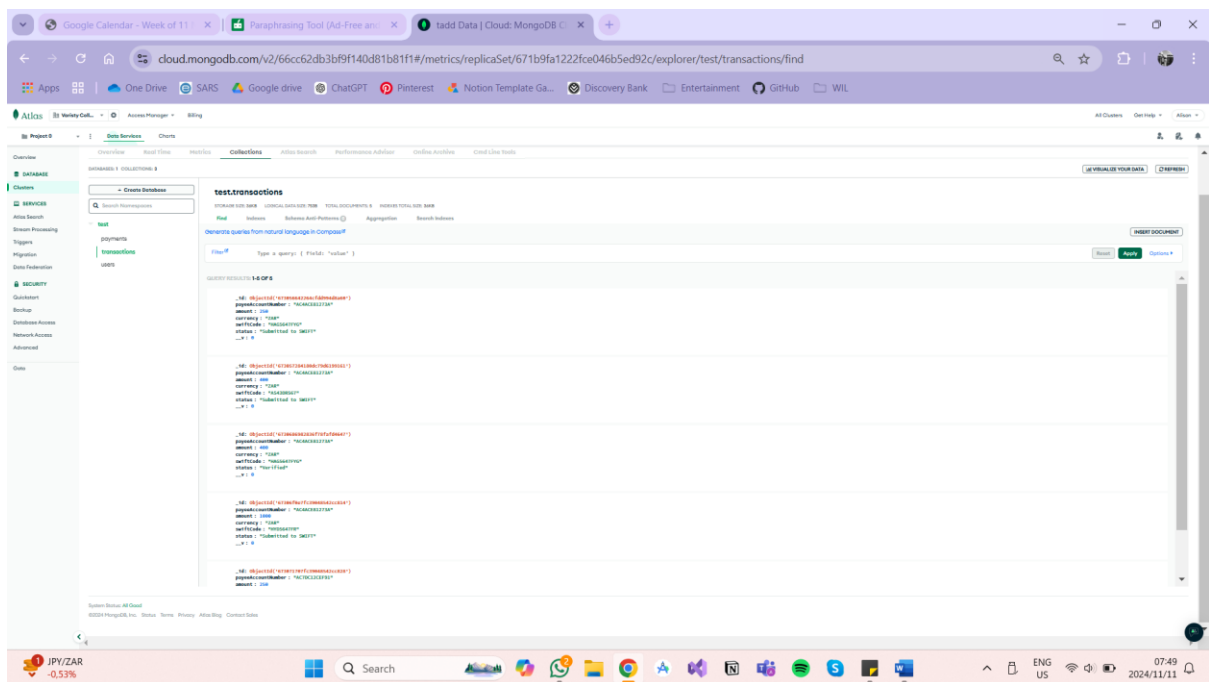


Figure 1.8

This demonstrates that transactions have been successfully saved to the MongoDB database (MongoDB Atlas, 2023).



## CircleCI

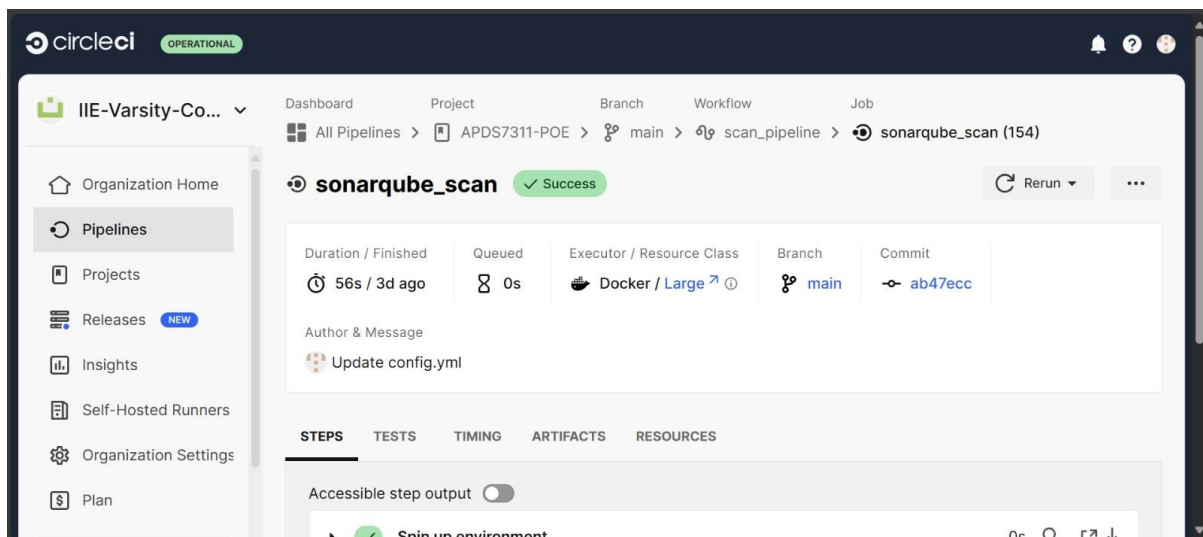


Figure 2

To guarantee uniform security and functionality throughout the application, CircleCI has been set up for continuous integration. To detect possible defects or security vulnerabilities prior to deployment, CircleCI automatically launches tests and initiates a SonarQube scan for static code analysis with each code modification (Anderson, 2020). This procedure guarantees that important security procedures like password hashing, validation, and whitelisting are applied accurately and operating as intended (Anderson, 2020). The `sonarsource/sonar-scanner-cli` Docker image is used by the CircleCI pipeline to check out the code, install dependencies, and execute the SonarQube scan. This ensures that only the pertinent code is examined for quality and security by analyzing the source code and removing extraneous files (Anderson, 2020). By anticipating and resolving problems early on, this configuration allows for quicker, more dependable updates and contributes to the stability of the program.

```
version: 2.1
jobs:
  sonarqube_scan:
    docker:
      - image: sonarsource/sonar-scanner-cli:latest # Official SonarQube scanner image
    steps:
      - checkout # Check out code from your GitHub repository
      - run:
          name: "Install Dependencies" # Only necessary if you have dependencies for the scan
          command: npm install --prefix employee-payments-portal/client # Example path; adjust if needed
      - run:
          name: "Run SonarQube Scan"
          command: |
            sonar-scanner \
              -Dsonar.projectKey="$SONAR_PROJECT_KEY" \
              -Dsonar.organization="$SONAR_ORG" \
              -Dsonar.host.url="$SONAR_HOST_URL" \
              -Dsonar.login="$SONAR_TOKEN" \
              -Dsonar.sources="employee-payments-portal" \
              -Dsonar.branch.name="${CIRCLE_BRANCH}" \
              -Dsonar.exclusions="*/c,*/.cpp,*/.h,*/tests/*" \
              -Dsonar.c.file.suffixes=- \
              -Dsonar.cpp.file.suffixes=- \
              -Dsonar.objc.file.suffixes=-

workflows:
  version: 2
  scan_pipeline:
    jobs:
      - sonarqube_scan
```

## SonarQube

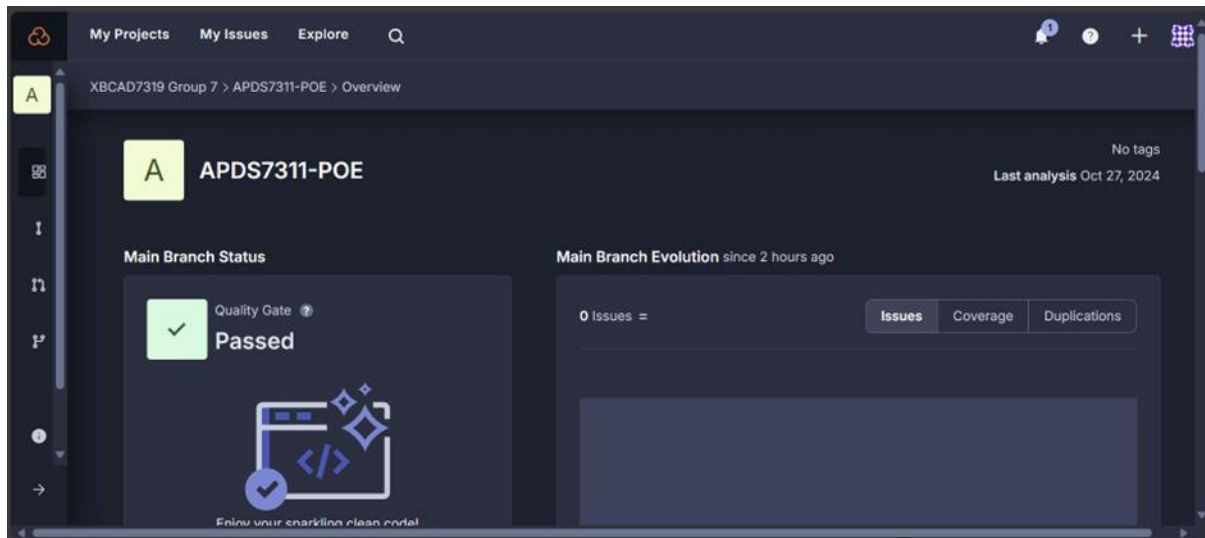
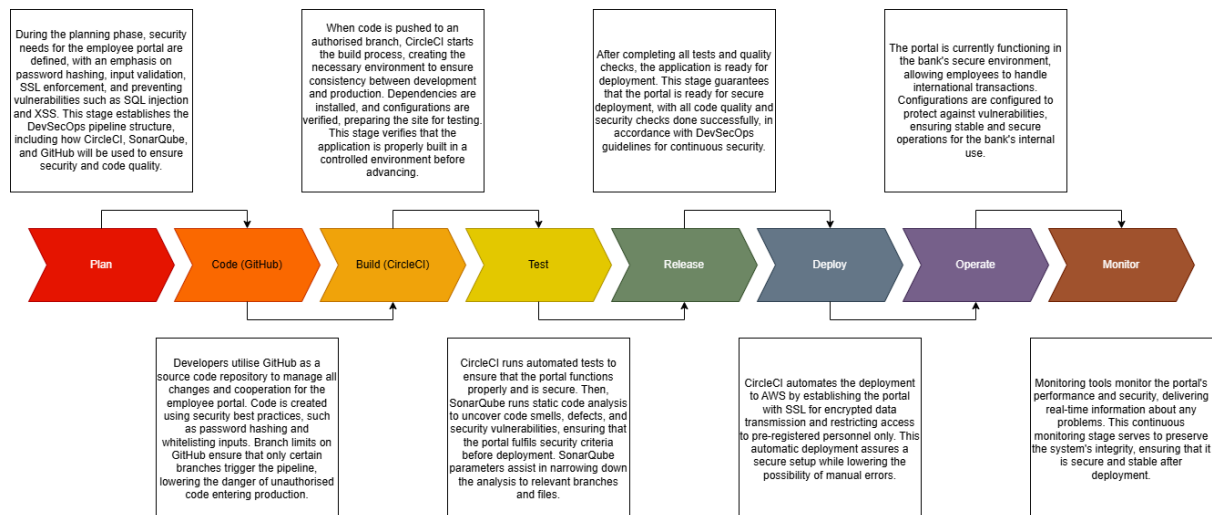


Figure 2.1

SonarQube is integrated into the APDS7311-POE project to ensure the use of automated static code analysis to guarantee ongoing code security and quality. A SonarQube scan is one of the automatic tests that are triggered by every code update when CircleCI is used for continuous integration. The project's codebase is assessed by the SonarQube scan using several quality measures, including problems, coverage, and duplications. It contributes to upholding high levels of quality by scanning for vulnerabilities, code smells, and maintainability problems. The CircleCI dashboard, which tracks the state of the main branch, shows the SonarQube results. The code passes the "Quality Gate," guaranteeing that it is safe and dependable prior to deployment if it satisfies the established quality requirements. This integration offers a proactive method of preserving code quality and stability throughout the development lifecycle by guaranteeing that any problems are identified early.

## Dev Ops



The secure employee portal's DevOps pipeline incorporates security at every stage of development to give workers a dependable and secure platform. It begins with the planning stage, where important security measures are set up, including input validation, SSL encryption, password hashing, and defence against typical vulnerabilities like XSS and SQL injection (Anderson, 2020). Unauthorized changes are prevented by branch protection measures and secure coding practices once the code is managed in GitHub. By automating the build process, CircleCI makes sure that configurations are validated and consistent across environments.

SonarQube analyses static code to identify possible vulnerabilities early, and automated tests verify both functionality and security (Anderson, 2020). The gateway is deployed to AWS after passing these tests, guaranteeing SSL encryption for safe data transfer, and limiting access to only authorized individuals. Tools for continuous monitoring are used to keep tabs on system integrity, detect security risks, and track performance. This DevOps pipeline offers a thorough method for developing, testing, and deploying the employee portal safely, guaranteeing that it continues to be a reliable and secure setting for crucial processes.

## **Functional Requirements**

### **User Authentication:**

- Employees log in with pre-registered credentials.
- Customers register and log in with valid username and password.
- Username and password validated using regex patterns.
- bcrypt used for password hashing and matching.
- Employees are redirected to the employee dashboard upon login.

### **User Roles:**

#### **Customer:**

- Register an account before logging in.
- Can create and view payments.

#### **Employee:**

- Pre-registered accounts with employee role.
- Can manage, verify, and submit payments to SWIFT.
- View customer payments and their status.

### **Payment Management:**

- Customers can create and submit payment details.
- Employees can verify and submit payments to SWIFT.
- Payment status: Pending, Verified, Submitted to SWIFT.

### **Transaction History:**

- Customers can view past payments with details like amount, currency, SWIFT code, and recipient account.
- Employees can view and manage payment history.

### **Security:**

- Passwords securely stored using bcrypt.
- SSL encryption for secure communication.
- Input fields validated with regex to prevent weak or invalid credentials.
- Protection against XSS and SQL injection.

### **User Interface:**

- Simple and responsive interface for customers and employees.
- Easy navigation between payment creation and transaction history.

### **System Monitoring and Logging:**

- Logs all important actions like logins, payment submissions, and verifications.
- Admins and employees can access logs for tracking activities.

### **Performance:**

- Fast loading times for payment data.
- Efficient handling of multiple concurrent requests.

## **Non Functional Requirements**

**Scalability:** The system should support up to 100 concurrent users without performance issues (Anderson, 2020).

**Security:** All communication between the client and server must use HTTPS. Passwords must be hashed using bcrypt before storage (Anderson, 2020).

**Performance:** Payment processing and user authentication responses must complete within 2 seconds under normal load (Anderson, 2020).

**Reliability:** The database uptime should be 99.9% to ensure the availability of user and payment data.

**Usability:** The user interface must be intuitive and accessible across desktop and mobile browsers.

**Maintainability:** The codebase should follow the best practices to facilitate easy updates and maintenance.

**Availability:** The system must be available 24/7, except for scheduled maintenance, with a maximum downtime of 0.1%.

**Compatibility:** The portal should be compatible with major browsers (Chrome, Firefox, Safari, Edge) and be responsive on both desktop and mobile devices.

**Compliance:** The system must comply with GDPR for data protection and privacy (European Parliament, 2016).

**Backup and Recovery:** The system should perform regular automated backups of critical data and allow for quick recovery in the event of a failure.

## **FAQ's**

### **What if MongoDB Atlas connection fails?**

The first thing to do if the MongoDB Atlas connection doesn't work is to make sure the .env file contains the right MongoDB connection URI. Verify that the URI is legitimate and identifies the appropriate MongoDB cluster. Additionally, make sure that no firewall settings are preventing the connection to MongoDB and look for any problems with network connectivity. Additionally, you should check the status of your MongoDB Atlas cluster, since it can be undergoing maintenance or going through downtime that could result in problems connecting.

### **Why am I getting a CORS error?**

When the backend server is not set up to accept requests from the frontend's origin, a CORS (Cross-Origin Resource Sharing) issue usually happens. Make sure the backend server has the appropriate CORS settings to accept requests from the frontend URL to fix issue. To ensure that the requests are successfully executed, you might need to modify the server's CORS configuration to specifically include the frontend's domain.

### **What to do if HTTPS setup fails?**

The most frequent reason for HTTPS setup failures is an SSL certificate issue. First, make sure the SSL certificate is set up correctly and in the appropriate server directory. Make that the SSL certificate and domain name match; if they don't, HTTPS may not work. Additionally, confirm that the certificate is current and hasn't expired, and that the server is appropriately waiting for secure (HTTPS) traffic.

### **How can I reset a forgotten admin password?**

You can use a database management tool like MongoDB Atlas to change an admin password if you've forgotten it. Change the admin user account's password by going straight to the database. Depending on the tool you're using, you might have to manually reset the credentials via the user interface or use a query to change the password field. Make that the new password complies with the security specifications of the system.

### **What should I do if npm modules fail to install?**

Try using the command with administrator rights if npm install is unable to install the necessary modules. This can fix problems with authorization. Additionally, confirm that the versions of Node.js and npm you are running are compatible with the project's documentation. Try the installation again after clearing the npm cache with `npm cache clean --force` if the problem continues.

## **Developer Information**

**Dillon Wernich**

Email: st10061721@vcconnect.edu.za

**Devania Chetty**

Email: st10049331@vcconnect.edu.za

**Alison Rees**

Email: st10038986@vcconnect.edu.za

**Tawfiq Mohamed**

Email: st10042129@vcconnect.edu.za

## References

Anderson, R. (2020) Security Engineering: A Guide to Building Dependable Distributed Systems. 3rd ed. New York: Wiley.

Biryukov, A., Dinu, D. and Khovratovich, I. (2016) 'Argon2: The memory-hard function for password hashing and other applications', Journal of Cryptographic Engineering, 6(2), pp. 89–111.

Ferraiolo, D.F. and Kuhn, D.R. (1992) 'Role-Based Access Controls', Proceedings of the 15th NIST-NCSC National Computer Security Conference, pp. 554–563.

OWASP Foundation (2023) OWASP Top Ten Web Application Security Risks. Available at: <https://owasp.org/Top10/> (Accessed: 27 October 2024).

Rescorla, E. (2018) The Transport Layer Security (TLS) Protocol Version 1.3. Internet Engineering Task Force. Available at: <https://tools.ietf.org/html/rfc8446> .

Verizon (2023) Data Breach Investigations Report. Available at: <https://verizon.com/dbir/> (Accessed: 27 October 2024).