Bankme Receivables Management System

Bankme has introduced a new feature related to receivables management. Every day, Bankme's client deals with numerous receivables, and our operations team was overwhelmed with the manual entry of these transactions. Receivables are digital representations of documents that simulate debts to be received. It's crucial for Bankme to incorporate this information into its commercial flow with its client.

Project Structure

The project has been structured into two main parts: the backend and the frontend.

Backend

Technologies Used:

- NestJS: Used to create the API.
- Prisma: Used to handle the persistence layer and create an SQLite database.
- RabbitMQ: Used for message queueing.
- Nodemailer: Used for sending emails.
- Docker: Used to encapsulate the development environment.

Implementations:

Implementations:

1. **Data Validation**: We implemented an API using NestJS that receives data for a receivable and a cedent. We ensured that no field is null, that IDs are of type UUID, and that strings do not exceed the defined character limit.

2. **Data Persistence**: We used Prisma to create an SQLite database and store information about receivables and cedents. We implemented routes for registration, editing, deletion, and retrieval of receivables and cedents.

3. **Authentication and Access Control**: We implemented an authentication system on all routes. Users can log in and receive a JWT token with a 1-minute expiration time. This token is required to access protected routes.

4. **Permission Management**: We created a permissions management system where login and password credentials are stored in the database, the password is encrypted with bcrypt. The system only generates JWTs if the credentials are registered in the database.

5. **Message Queueing**: We utilized RabbitMQ for handling message queueing, particularly for batch processing of payments.

6. **Email Notifications**: We integrated Nodemailer to send email notifications, such as alerts for successful batch processing or errors encountered during processing.

PROF

7. **Documentation and Infrastructure**: We created a Dockerfile for the API and a docker-compose.yaml to start the project.

Frontend

Technologies Used:

- Next.js: Used to create the user interface.
- Tailwind CSS: Used for styling the user interface.
- Axios: Used to make HTTP requests to the API.

Implementations:

1. **Registration of Receivables and Cedents**: We created an interface where users can register receivables and cedents. The interface prevents the registration of empty fields or fields that do not meet the defined rules.

2. **Connection to the API**: We connected the frontend to the created API and ensured that the registration of a receivable is reflected in the API. We also implemented the cedent registration screen.

3. **Listing of Receivables**: We created a page for listing receivables, showing only id, value, and emission date. For each item in the list, we included a link that shows the details of the receivable. We also implemented options for editing and deleting.

4. **User Authentication**: We implemented a login and password system to access routes in an authenticated manner. The JWT token is stored in the browser's localStorage.

5. **Testing**: We created tests for the frontend application, ensuring that all functionalities are working correctly.

How to run the Project Locally

PROF

Pre requisites:

- Docker installed on your machine
- Git installed on your machine

Steps:

1. **Clone the repository:**

   ```
   git clone https://github.com/your-username/project-repo.git
   ```

2. **Navigate to the project directory:**

   ```
   cd bankme-samueldeotti
   ```

3. **Start the Docker containers:**

   ```
       docker-compose up --build
   ```

   This command will build and start the necessary Docker containers for the project. It may take a few minutes on the first run as it depends on your internet connection speed to download the required Docker images.

4. **Access the application:**
   Once the previous command has completed successfully, you can access the application at the following URLs:

   - Frontend: http://localhost:3000
   - Backend: http://localhost:8080
   - RabbitMQ: http://localhost:5672 and http://localhost:15672

5. **Interact with the application:**
   You can now interact with the frontend application through your web browser. The backend will be running in Docker containers, serving requests made by the frontend.

6. **Stop the containers:**
   To stop the containers, press Ctrl + C in the terminal where the containers are running or execute the following command in the project directory:

   ```
   docker-compose down
   ```