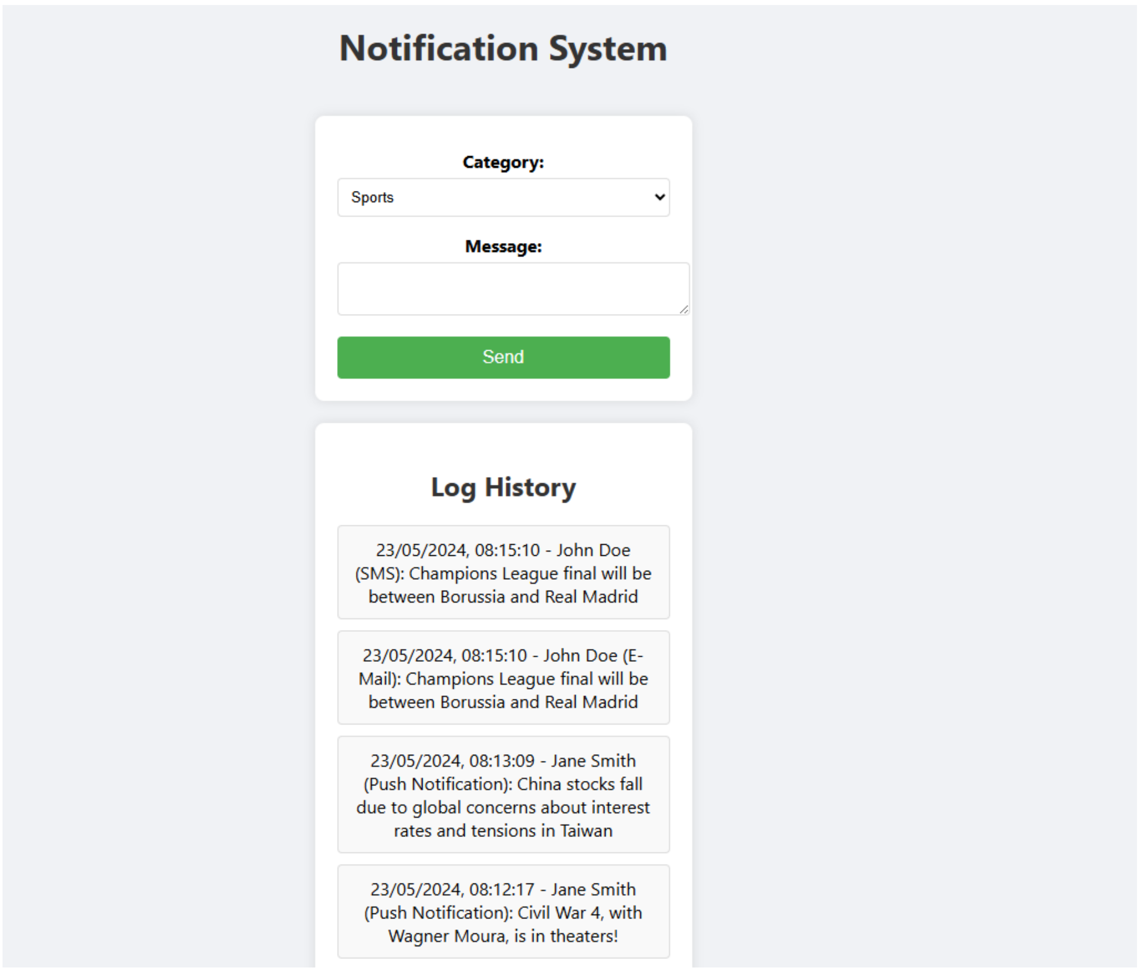
Documentation Notification Test

Project description

The Notification System project is an application developed using Node.js for the backend and React for the frontend. It consists of a notification management system where users can subscribe to different categories and receive notifications via SMS, email, or push notifications. The application follows an MVC (Model-View-Controller) architecture and utilizes MongoDB to store user information and their notification preferences. The goal of the project is to provide a robust and scalable solution for managing user notifications, with an emphasis on flexibility and ease of use.

FrontEnd Example



23/05/2024, 08:15:10 - John Doe (SMS): Champions League final will be between Borussia and Real Madrid

23/05/2024, 08:15:10 - John Doe (E-Mail): Champions League final will be between Borussia and Real Madrid

23/05/2024, 08:13:09 - Jane Smith (Push Notification): China stocks fall due to global concerns about interest rates and tensions in Taiwan

23/05/2024, 08:12:17 - Jane Smith (Push Notification): Civil War 4, with Wagner Moura, is in theaters!

Technologies Used

Backend

- Node.js
- Express.js
- Mongoose
- MongoDB
- dotenv

Frontend

- React
- Axios
- Styled-components

Prerequisites

- Node.js (version 14 or higher)
- MongoDB

Compilation and Execution Instructions

The README.md file present in this same directory has details for compiling, executing and carrying out tests.

Notification System Architecture

Frontend (React)

Components

- NotificationForm: Component for sending notifications.
- LogHistory: Component for viewing the log history of notifications.

API Integration

- Uses axios to make HTTP requests to the backend.

Environment

- Runs on port 3001 during development.

Backend (Node.js + Express)

Controllers

- notificationController.js: Handles the incoming requests for sending notifications and retrieving logs.

Services

- emailService.js: Handles sending notifications via email.
- pushService.js: Handles sending notifications via push.
- smsService.js: Handles sending notifications via SMS.
- notificationService.js: Orchestrates the sending of notifications based on user subscriptions.

Repositories

- logRepository.js: Manages CRUD operations for notification logs in MongoDB.

Models

- log.js: Defines the schema for notification logs.

Routes

- notificationRoutes.js: Defines the API endpoints for notifications.

Environment

- Runs on port 3000 during development.

Database

- MongoDB: Stores user information, subscription details, and notification logs.

Data Flow

User Interaction

- The user interacts with the React frontend to send a notification.
- The form data is submitted to the backend via an API call.

API Request Handling

- The backend Express server receives the request.
- The notificationController processes the request and calls the notificationService.

Notification Processing

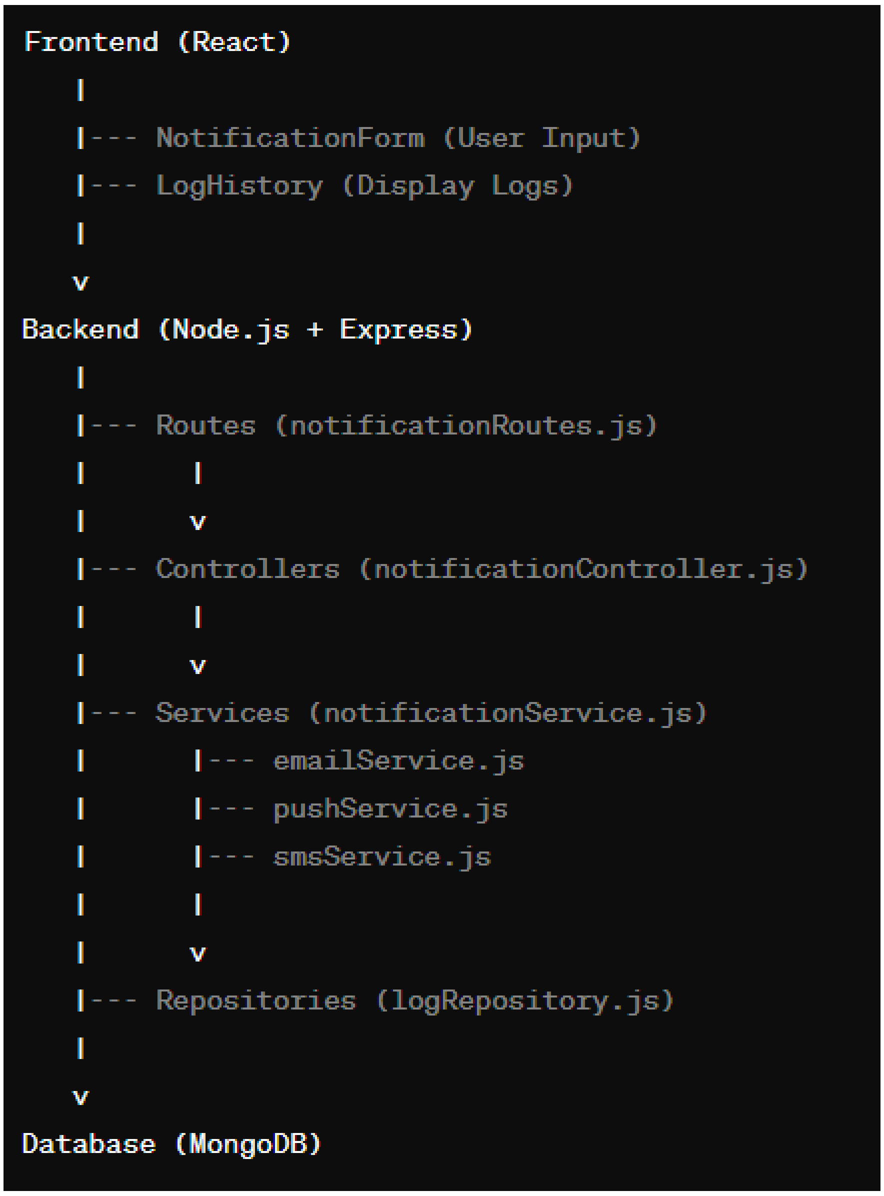
- The notificationService checks user subscriptions and decides which notification channels (email, push, SMS) to use.
- Calls corresponding services (emailService, pushService, smsService) to send notifications.

Logging

- After sending notifications, the backend logs the notification details using logRepository and saves them in MongoDB.

Log Retrieval

- The user can request the log history through the frontend.
- The request is handled by the notificationController, which retrieves the logs from MongoDB using logRepository.



```
Frontend (React)
   |
   |--- NotificationForm (User Input)
   |--- LogHistory (Display Logs)
   |
   v
Backend (Node.js + Express)
   |
   |--- Routes (notificationRoutes.js)
   |       |
   |       v
   |--- Controllers (notificationController.js)
   |       |
   |       v
   |--- Services (notificationService.js)
   |       |--- emailService.js
   |       |--- pushService.js
   |       |--- smsService.js
   |       |
   |       v
   |--- Repositories (logRepository.js)
   |
   v
Database (MongoDB)
```

Diagram representing the text before.

Examples of Design Patterns Applied on the Project

1. Factory Pattern

- Example: Creating instances of different notification services (email, SMS, push).

2. Strategy Pattern

- Example: Defining a family of algorithms for sending notifications, encapsulating each one, and making them interchangeable.

3. Singleton Pattern

- Example: Ensuring a single instance of the MongoDB connection.

4. Observer Pattern

- Example: Logging system that acts as an observer for notification events.

Key Features of the Notification System

1. Sending Notifications

Description: The system allows sending notifications to users based on their category subscriptions. Notifications can be sent via three different channels: SMS, email, and push notifications.

How it works:

- Users subscribe to different notification categories.
- The system sends notifications to users via their preferred channels.
- Notification services (EmailService, PushService, SMSService) handle the actual sending process.

Related Components:

- NotificationForm (frontend component): Allows users to input the category and message for the notification.
- notificationController (backend controller): Handles the API request to send notifications.
- notificationService (backend service): Orchestrates the sending process, ensuring notifications are sent via the appropriate channels.

2. Logging Sent Notifications

Description: The system logs every notification sent, including details such as the user, category, message, channel, and timestamp. This helps in tracking and auditing the notifications.

How it works:

- Each time a notification is sent, a log entry is created and saved to the database.
- The logging system is triggered via events emitted during the notification sending process.

Related Components:

- logRepository (backend repository): Manages CRUD operations for notification logs in MongoDB.
- notificationEmitter (backend utility): Emits events when notifications are sent.
- notificationService (backend service): Emits events for logging after sending notifications.

3. Interface for Sending Notifications

Description: The system provides a user-friendly interface for sending notifications. Users can select the notification category and input the message they want to send.

How it works:

- The frontend provides a form where users can enter the notification details.
- The form data is submitted to the backend via an API call.

Related Components:

- NotificationForm (frontend component): The form for sending notifications.
- api.js (frontend utility): Manages API calls to the backend.

Unit Testing in the Notification System

The Notification System project incorporates comprehensive unit tests to ensure the reliability and correctness of its components. Unit tests are applied to various parts of the backend, including the notification services (`EmailService`, `PushService`, `SMSService`), the notification controller, and the repository responsible for logging notifications.

Using the Jest testing framework, these tests verify that each service correctly handles the sending of notifications and that the logging functionality operates as expected. Mock data and functions are employed to simulate real-world scenarios without requiring actual network requests or database interactions.

This approach allows for the detection of potential issues early in the development process, promoting code quality and robustness. For instance, the `notificationService` is tested to ensure that notifications are sent to the appropriate channels based on user subscriptions, while the `logRepository` is tested to confirm that logs are correctly saved and retrieved from MongoDB.

Overall, the unit tests play a crucial role in maintaining the integrity of the Notification Test, enabling developers to confidently extend and modify the codebase.

Final consideration

Considering that this was a short project without direct client contact for potential feedback and guidance, I have outlined below some challenges faced and points for improvement. In a work environment with deadlines and stakeholder interactions, these can certainly be developed further to aim for the best final product!

1. **Challenges**:

- Data Validation: Ensuring that the data provided by users is valid and that all business rules are correctly applied can be a challenge. The current implementation has simpler logic, but it could become a significant challenge depending on the system's requirements.

- Asynchronous Operations: Managing asynchronous operations, such as sending notifications through different channels (email, SMS, push notifications) and logging these actions in the database, posed a challenge. Ensuring that these operations were performed efficiently and that the system could handle multiple concurrent requests required the use of asynchronous programming techniques and proper error handling to maintain reliability and performance.

- Integration Between Frontend and Backend: Ensuring seamless communication between the React frontend and the Node.js backend was another challenge. This involved configuring the proxy settings correctly, and ensuring that the API endpoints were correctly defined and accessible. Thorough testing was required to ensure that data flowed correctly between the frontend and backend.

2. **Points of Improvement**:

Scalability Enhancements:

- Current State: The system is designed with basic scalability in mind.
- Improvement: Implement horizontal scaling strategies by deploying the backend on container orchestration platforms like Kubernetes. Use load balancers to distribute traffic efficiently and ensure the system can handle increased load and user traffic seamlessly.

Security Improvements:

- Current State: Basic security measures are in place.
- Improvement: Enhance security by implementing JWT-based authentication, encrypting sensitive data, using HTTPS for all communications, and conducting regular security audits and penetration testing to identify and mitigate potential vulnerabilities.

Frontend Optimization and User Experience:

- Current State: The frontend provides basic functionality for sending notifications and viewing logs.
- Improvement: Improve the user experience by optimizing the UI/UX design, implementing responsive design principles for better mobile compatibility, and adding features like real-time updates using WebSockets to provide instant feedback and updates to users.

Continuous Integration and Deployment (CI/CD):

- Current State: Manual deployment processes are likely used.

- Improvement: Implement a CI/CD pipeline using tools like GitLab or Jenkins to automate testing, building, and deployment processes. This will ensure faster and more reliable deployments and streamline the development workflow.