CountryVote - Documentation

Overview

This project consists of a full-stack application, "**CountryVote**", that allows users to vote for their favorite countries and view a ranking of top 10 favorite countries. The project is divided into two main parts: the backend service and the frontend interface.

It is estimated that the time invested to the solution is of 16 hours
Backend: 2h (capacitation) + 7h development = 9h
Frontend: 1h (capacitation) + 6h development = 7h

Backend

The backend is built with Node.js, Express, and TypeScript, interacting with the RestCountries API to retrieve country details. It follows clean architecture principles like: low coupling, high cohesion, and separation of concerns.

Functional Requirements

- **User Registration**: Users can register by providing their name, email, and favorite country.
- **Vote Limitation**: Each email can vote once.
- **Top 10 Favorite Countries**: The system counts the votes and displays the top 10 favorite countries along with their details: name, official name, capital city, region, and sub-region.

Non-Functional Requirements

- **Scalability**: It is designed to be easily extendable, allowing future features like database integration or additional services.
- **Modularity**: Each part of the system (controllers, services, repositories, models) is modular, making the codebase easier to maintain and test.
- **Error Handling**: The system includes basic error handling, ensuring that errors are logged and users receive feedback.
- **Security**: Basic security practices, such as input validation and the use of environment variables for configuration, are in place.
- **Persistence:** The system is designed to use MySQL as a storage solution, ensuring that user data, such as votes, is stored and can be retrieved at any time.

Design Choices

1. Architecture

The application follows a layered architecture with separation of concerns:

- **Controllers**: Handle HTTP requests and responses, delegating business logic to the services.
- **Services**: Contain the business logic, such as user registration, country voting, and aggregation of results.

- **Repositories**: Manage data persistence. For this project, we used an in-memory repository with the option to easily integrate a database in the future.

2. Use of Interfaces

Interfaces are used throughout the project to define contracts between components (services and repositories). This approach promotes loose coupling, as depending on interfaces rather than concrete implementations, promotes low coupling. This makes it easier to swap out implementations, for example replacing in-memory storage with a database, without changing the dependent code.

3. Async/Await for Asynchronous Operations

The system uses async/await to handle asynchronous tasks, making the code easier to read and maintain. This approach keeps the program clear and organized. Using try/catch blocks with await ensures that errors during these operations are handled correctly, improving overall stability and reliability.

4. Dependency Injection

Dependency injection is used to decouple the components, making it easier to maintain. This design allows for easy swapping of implementations, such as replacing the in-memory repository with a database.

5. Rationale for Choosing MySQL

MySQL was chosen to manage persistence in this system due to:

- Reliable relational database management system with a history of use in production environments.
- Offers great performance for read-heavy applications, which is useful for retrieving and managing user votes.
- **C**apable of handling large volumes of data and can scale both vertically and horizontally as the application grows.

6. Fallback to In-Memory Storage

The system is designed to run with MySQL, but, in order to allow the application to be tried and tested without needing to set up a database, if the server fails to connect to MySQL, it will automatically use an in-memory storage solution. It is not intended for production use.

The localUserRepository is implemented with singleton to ensure that all parts of the application share the same instance. This choice was made to prevent issues with data consistency and to simplify the management of user data.

Compromises

1. In-Memory Storage

Due to time limitation, the application uses in-memory storage for user data. While this approach is sufficient for a small-scale project, it is not scalable for production environments. The decision was made to simplify development, with the understanding that a database integration would be necessary if it was to be deployed to production.

2. Simplified Security
Basic security practices, such as checking the email, were implemented, but more advanced security measures such as authentication, authorization were not included. The focus was on getting the core functionality working correctly and securely.

3. Error Handling
While error handling is implemented, it is simplified. For example, the system logs errors and returns simplified error messages to the client. In a full production system, more granular error logging, and user-friendly responses would be needed.

API Endpoints
Base URL: http://localhost:3000

1. Register a User
Endpoint: /api/users
- **Method:** POST
- **Description:** Registers a new user by providing their name, email, and favorite country.

Request Body:
- **name**: string
- **email**: string (must be unique)
- **country**: string

Success Response: 201 Created
Error Response: 400
1. "message": "This email has already voted."
2. "message": "Request body must include name, email, and country."

2. Get Top 10 Favorite Countries
Endpoint: /api/countries
- Method: GET
- Description: Retrieves the top 10 favorite countries based on user votes. Shows details from the countries: name, official name, capital city, region, and sub-region.

Request Body: None
Success Response: 200
1. Ranking of top 10 countries
2. "message": "No votes have been cast yet."

Error Response: 500
1. "message": "An error occurred while retrieving the top favorite countries."

Future Enhancements
- **Authentication**: Implement user authentication and authorization.

- **Detailed Logging**: Improve logging and monitoring for better error tracking and debugging.
- **Unit Testing:** Add tests to ensure code quality and reliability.

Frontend

Overview

The frontend runs with React and Next.js, using Tailwind for styling. The application provides users with a form to submit their votes and displays the top 10 most voted countries in a table.

Components

1. Voting Form

This component allows users to vote for their favorite country by submitting their name, email, and country of choice. The form processes the inputs and sends the data to the API. Upon successful submission, the form displays a success message, and automatically the table of top countries is refreshed.

Country Display Table

The country display table component shows a ranking of the ten most voted countries. It retrieves data from the API and allows users to filter the displayed countries by their name, capital city, region, or subregion.

Responsiveness

The interface is built to be responsive, ensuring a great user experience across various devices, from mobile phones to desktops. Elements automatically adjust their layout to fit the screen size.

Integration with Backend

Error Handling: Both components handle errors. The voting form shows appropriate error messages if the submission fails, while the table shows an error if the data retrieval fails.

Country Service: There is a country service module that centralizes the interactions with the API, making the code modular and maintainable. It contains the following functions:

- **fetchTopCountries**: Retrieves the top 10 most voted countries from the backend.
- **submitVote**: Sends the user’s vote (name, email, country) to the backend for processing.