Adamur User Management System

By: Joseph Gathithi Wathome, wathomejosephgathithi@gmail.com

Documentation

Table of Contents

Project Overview

The Adamur User Management System is designed to facilitate user registration, authentication, account verification, and password management for an educational platform. The system leverages modern technologies and best practices to ensure a seamless user experience while maintaining high standards of security and scalability.

Technologies Used

- **Node.js**: JavaScript runtime for building scalable server-side applications.
- **Express**: Web framework for Node.js, simplifying the development of APIs.
- **GraphQL**: Query language for APIs, providing a more flexible and efficient way to interact with data.
- **Prisma**: ORM for Node.js, allowing seamless interaction with the database.
- **TypeScript**: Superset of JavaScript, adding static typing to enhance code quality and maintainability.
- **smtplib**: Python Module for sending emails, facilitating account verification and password reset functionality.
- **dotenv**: Module for managing environment variables, enhancing security by keeping sensitive data out of source control.

Development Process

The development process for the Adamur User Management System was structured into several phases:

1. **Requirement Gathering**: We identified the core functionalities needed for user management, including registration, login, account verification, password reset, and token management.
2. **Architecture Design**: The system was architected using a modular structure, allowing for separation of concerns. The main components include:
    - **Services**: Contain business logic (e.g., userService, authService).
    - **Resolvers**: Handle GraphQL queries and mutations.
3. **Implementation**: Features were implemented incrementally, with regular testing and refactoring to ensure code quality. We utilized TypeScript to catch errors during development and improve maintainability.
4. **Testing**: Comprehensive unit and integration tests were written for all major functionalities, ensuring robustness and reliability of the system.

Key Decisions

Several key decisions were made during the development of this project:

- **Choice of GraphQL**: We chose GraphQL over REST due to its flexibility and ability to fetch only the required data, reducing bandwidth usage and improving performance.
- **Use of Prisma**: The decision to use Prisma as an ORM was based on its ease of use, strong type safety with TypeScript, and support for migrations, which streamline database schema management.
- **Email Verification and Password Reset**: Implementing email verification and password reset functionality adds a layer of security, ensuring that only legitimate users can access the system.

API Endpoints

The Adamur User Management System exposes several GraphQL endpoints for interacting with user data:

post graph request to. http://localhost:4000/graphql

User Registration

Mutation: register

- **Input**:
    - username: String
    - email: String

    - password: String

```

mutation {
  register(input: {username: "us3edddsrr2389",email: "tresddsdtt@example.com",
password: "password123" }) {
    token
    user {
      id
      email
      isVerified
    }
  }
}

```

- **Response**: Returns user data or an error message.

  **```Check your email for the verification token expires in 24 hours.```**

```

{
    "data": {
        "register": {
            "token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOiJjNWZiY2FjZC1kZjVhLTQyOTItOGY1OS0w
MWVmMTRmYzEwNDkiLCJpYXQiOjE3Mjg2MDA4NzMsImV4cCI6MTcyODY4NzI3M30.J_nJtuMNp3lptIJLcNyfd3
C8NW0dhjoagxwtVnmqTts",
            "user": {
                "id": "c5fbcacd-df5a-4292-8f59-01ef14fc1049",
                "email": "tresddsdtt@example.com",
                "isVerified": false
            }
        }
    }
}

```

Account Verification

Mutation: verifyAccount/verifyEmail

- **Input**:

    - otp: String (verification token sent to the user's email)

query:

```
mutation {
  verifyEmail(otp: "594208") {
    id
    email
    isVerified
  }
}
```

- **Response**: Returns success message or an error message.

```
{
    "data": {
        "verifyEmail": {
            "id": "c5fbcacd-df5a-4292-8f59-01ef14fc1049",
            "email": "tresddsdtt@example.com",
            "isVerified": true
        }
    }
}
```

User Login

Mutation: loginUser

- **Input**:
    - email: String
    - password: String

```
mutation {
  login(input: { email: "tresddsdtt@example.com", password: "password123" }) {
    token
    user {
      id
      email
      isVerified
    }
  }
}
```

- **Response**: Returns a JWT token and user data or an error message.

```

{
    "data": {
        "login": {
            "token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOiJjNWZiY2FjZC1kZjVhLTQyOTItOGY1OS0w
MWVmMTRmYzEwNDkiLCJpYXQiOjE3Mjg2MDEyNjIsImV4cCI6MTcyODY4NzY2Mn0.MCYpXadIxjqDG8C-
FtQx9gRI1JoYm0WGSxN_CJHLem0",
            "user": {
                "id": "c5fbcacd-df5a-4292-8f59-01ef14fc1049",
                "email": "tresddsdtt@example.com",
                "isVerified": true
            }
        }
    }
}
```

Password Reset Request

Mutation: requestPasswordReset

- **Input**:
  - email: String

```

mutation {
  requestPasswordReset(email: "tresddsdtt@example.com")
}
```

- **Response**: Returns success message or an error message.

~~~
```

{
    "data": {
        "requestPasswordReset": true
    }
}

```
~~~

``` **Check your email and spam box to get the reset code**. Something like: Reset token: 657378faf071c9524e2899b9f16656492ac8cde01ba3a88c8484e7cf1e45be17 `` Token valid for 1 hour

Password Reset

Mutation: resetPassword

- **Input**:
    - token: String (password reset token)
    - newPassword: String

````
```

mutation {
  resetPassword(input: { resetToken:
"657378faf071c9524e2899b9f16656492ac8cde01ba3a88c8484e7cf1e45be17", newPassword:
"newPassword123" }) {
    id
    email
    isVerified
  }
}

```
~~~

- **Response**: Returns success message or an error message.

**```New password set```**

```
{
    "data": {
        "resetPassword": {
            "id": "c5fbcacd-df5a-4292-8f59-01ef14fc1049",
            "email": "tresddsdtt@example.com",
```
~~~

```
            "isVerified": true
        }
    }
}
```

## Scalability Considerations

Scalability was a key consideration throughout the development process. The following strategies were implemented to ensure the system can handle increased loads:

- **Modular Architecture**: The modular design allows for easy scaling of individual components without affecting the entire system. New features can be added as separate modules, promoting maintainability and flexibility.
- **Database Optimization**: Using Prisma as an ORM allows for efficient database queries. We also considered indexing frequently queried fields to improve performance as the user base grows.
- **Caching Mechanisms**: Implementing caching strategies for frequently accessed data can significantly reduce database load and improve response times.
- **Load Balancing**: In a production environment, using load balancers can distribute incoming requests across multiple server instances, ensuring no single server is overwhelmed.

## Security Features

Security is paramount in the design of the Adamur User Management System. The following features were implemented to protect user data and ensure secure interactions:

- **Data Encryption**: User passwords are hashed using a secure algorithm (bcrypt) before being stored in the database. This protects user credentials in the event of a data breach.
- **Token Management**: JWT tokens are used for authentication, providing a secure means of verifying user identity. Tokens are signed to prevent tampering and can be set to expire, reducing the risk of unauthorized access.
- **Input Validation**: Input data is validated to prevent common attacks such as SQL injection and cross-site scripting (XSS).
- **Environment Variables**: Sensitive information, such as database connection strings and secret keys, is stored in environment variables using dotenv, keeping them out of the source code.

## Conclusion

The Adamur User Management System was developed with a focus on modern technologies, modular architecture, and best practices for security and scalability. The implementation of robust APIs, comprehensive testing, and strong security measures positions the system to effectively manage user data for an educational platform. Ongoing maintenance and regular updates will ensure that the system remains reliable, secure, and scalable as user needs evolve.