Final Semester Exam Report for Platform-Based Programming Course: API for Event Organizer

To fulfill the Final Semester Exam assignment for Platform-Based Programming Course, we have developed an API for Event Organizer. This API is designed to facilitate the management of various aspects related to organizing events, such as event category management, event management, ticket booking, email sending, and so on. This API is built using Node.js technology with the Express.js framework and MySQL database. The following is a detailed explanation of the API that we have created.

1. Architecture and Technology Used

This API is built using Node.js as the runtime environment and Express.js as the framework for building the server. The database used is MySQL, with the help of the mysql2 library to connect the Node.js application to the database. To manage user authentication, JSON Web Token (JWT) is used which is implemented with the json webtoken library. In addition, this API is also equipped with an email sending feature using the nodemailer library.

Some other libraries used include:

- bcrypt: To encrypt user passwords.
- dotenv: To manage environment variables.
- express-validator: To validate input data.
- cors: To allow cross-origin resource sharing.
- axios: To make HTTP requests to external APIs.
- express: Framework for building APIs.
- jsonwebtoken: Authentication and authorization with JWT.
- mysql2: Interaction with MySQL databases.
- nodemailer: Send email.
- nodemon: Monitor file changes and auto-restart servers.

2. Directory and File Structure

This API has a well-organized directory structure, which makes it easy to develop and maintain the code. Here is a brief explanation of the directory structure and files:

- config/: Contains configuration files such as database connections (db.js).
- controllers/: Contains controller files that handle business logic for each API endpoint, such as category-controller.js, event-controller.js, order-controller.js, etc.
- models/: Contains model files that interact directly with the database, such as category.js, event.js, order.js, etc.
- routes/: Contains route files that define API endpoints, such as category-route.js, event-route.js, order-route.js, etc.
- middlewares/: Contains middleware files such as auth-middleware.js used for user authentication.
- services/: Contains additional logic such as weather services.
- index.js: The main file that runs the server and initializes all routes.

3. Objectives

The objectives of creating this API are:

- To provide an integrated system to manage events, categories, locations, weather, users, and orders.
- To facilitate users in performing CRUD (Create, Read, Update, Delete) operations on various entities related to the event organizer.
- To provide an endpoint to send notification emails to event participants.
- To provide weather information based on the event location.

4.API Features and Endpoints

This API provides various features needed to manage event organizers. Here are some of the main features along with the available endpoints:

a. Event Category Management

GET /api/category: Get all event categories.

GET /api/category/:id : Get event category details based on ID.

POST /api/category: Add a new event category (requires authentication).

PUT /api/category/:id : Update event categories based on ID (requires authentication).

DELETE /api/category/:id : Delete event categories based on ID (requires authentication).

b. Event Management

GET /api/event: Get all events.

GET /api/event/:id : Get event details based on ID.

POST /api/event: Add a new event (requires authentication).

PUT /api/event/:id : Update events based on ID (requires authentication).

DELETE /api/event/:id : Deletes an event based on ID (requires authentication).

c. Order Management

GET /api/order: Gets all orders.

GET /api/order/:id : Gets order details based on ID.

POST /api/order: Adds a new order (requires authentication).

PUT /api/order/:id : Updates an order based on ID (requires authentication).

DELETE /api/order/:id : Delete an order based on ID (requires authentication).

d. User Management

POST /api/users/register: Registers a new user.

POST /api/users/login: Logins a user and gets a JWT token.

GET /api/users: Gets all users (requires authentication).

GET /api/users/:id : Get user details based on ID (requires

authentication).

PUT /api/users/:id : Update user data based on ID (requires authentication).

DELETE /api/users/:id : Delete user based on ID (requires

authentication).

e. Email Sending

POST /api/email: Send email (requires authentication). This endpoint is used to send confirmation or notification emails to users.

f. Additional Services

GET /api/location: Get location information based on a given address.

GET /api/weather: Get weather information based on a given city.

5.Authentication and Security

This API is equipped with an authentication system using JSON Web Token (JWT). Every user who successfully logs in will get a token that can be used to access endpoints that require authentication. This token has a validity period that can be set (in this case, the token is valid for 1 hour).

In addition, the user's password is encrypted using the bcrypt library before being saved to the database, thereby increasing the security of user data. Input validation is also performed using express-validator to ensure that the data received by the API is valid and secure.

6. API Testing

This API has been tested using tools such as Postman to ensure that all endpoints are working properly. Testing includes:

- CRUD (Create, Read, Update, Delete) testing for each entity (category, event, order, user).
- Authentication and authorization testing.
- Email sending testing.
- Integration testing with external services such as OpenWeatherMap for weather information.

7. Conclusion

The API for Event Organizer that we have created is a comprehensive solution for managing various aspects of event management. With an organized structure and complete features, this API can be used as a backend for event organizer applications, both web-based and mobile. The use of modern technologies such as Node.js, Express.js, and MySQL ensures that this API can run efficiently and scalably. In addition, the implementation of strict authentication and input validation makes this API safe from common attacks such as SQL injection and XSS. Thus, this API is ready to be used in a production environment and can be further developed as needed.