### **Backend Task (2 yrs exp.)**

#### **Objective**

Develop a robust backend using Node.js and MongoDB, providing APIs for a Flutter app integration. Ensure secure and efficient data handling, cloud deployment, and CI/CD automation.

#### **Task 1: Build a RESTful API**

**Objective**: Create a simple CRUD API for a product inventory.

**Requirements**:

1. ~~Set up a Node.js project using Express.~~
2. ~~Create endpoints for:~~
   * ~~Creating a new product.~~
   * ~~Retrieving all products.~~
   * ~~Retrieving a single product by ID.~~
   * ~~Updating a product by ID.~~
   * ~~Deleting a product by ID.~~
3. ~~Use a JSON file or an in-memory array to store the data.~~
4. ~~Implement error handling~~ and input validation.

#### **Task 2: Authentication and Authorization**

**Objective**: Implement user authentication using JWT.

**Requirements**:

1. Create endpoints for user registration and login.
2. On successful login, generate and return a JWT.
3. Protect certain routes (e.g., /profile, /settings) to be accessible only to authenticated users.
4. Validate the JWT on protected routes.

#### **Task 3: Data Fetching and Caching**

**Objective**: Fetch data from an external API and implement caching to optimize performance.

**Requirements**:

1. Set up a route that fetches data from an external API (e.g., weather data from a public API).
2. Cache the response using a caching mechanism (e.g., Redis, in-memory cache) for a specific duration (e.g., 10 minutes).
3. Serve the cached response if available and valid; otherwise, fetch new data from the external API.

#### **Task 4: Database Integration**

**Objective**: Integrate a database to store and retrieve data.

**Requirements**:

1. Set up a Node.js project with a connection to a MongoDB database (using Mongoose).
2. Create a schema and model for a sample entity (e.g., users, posts).
3. Implement CRUD operations using the database.

#### **Task 5: Real-time Features**

**Objective**: Implement real-time notifications using WebSockets.

**Requirements**:

1. Set up a WebSocket server using Socket.IO.
2. Create an endpoint for clients to connect to the WebSocket server.
3. Implement a feature where connected clients receive real-time notifications when a specific event occurs (e.g., a new message is posted).

#### **Task 6: Error Handling and Logging**

**Objective**: Implement comprehensive error handling and logging.

**Requirements**:

1. ~~Set up global error handling middleware in an Express app.~~
2. ~~Implement different error classes for different types of errors (e.g., validation errors, database errors).~~
3. Integrate a logging library (e.g., Winston) to log errors and other relevant information.

#### **Evaluation Criteria**

* **API Performance**: Efficient APIs with proper error handling.
* **Security**: Secure endpoints with JWT authentication.
* **Cloud Deployment**: Proper deployment on a cloud platform (Azure, AWS, GCP).
* **CI/CD Implementation**: Functional CI/CD pipeline automating testing and deployment.
* **Code Quality**: Clean, modular, and well-documented code.

#### **Additional Considerations**

* Caching with Redis for optimized performance.
* TypeScript for type safety.
* GraphQL for flexible API queries.

#### **Submission**

* Provide the complete source code in a GitHub repository.
* Include a README with setup, run, and deployment instructions.
* Provide API documentation and a brief architecture/design explanation.

#### **Bonus Task**

* **Advanced Monitoring**: Integrate monitoring and alerting tools to track application health and performance.

This task will assess the developer’s backend development skills, including API integration, cloud deployment, and CI/CD processes.