Backend Development Technical Test

Objective: To evaluate your skills in Node.js, Express, and MongoDB through the development of a RESTful API project.
Time Limit: 3 hours

Instructions:

- Create a GitHub repository for this project and commit your code regularly.
- Provide clear setup instructions in a README file.
- Ensure your code is well-documented and follows best practices.
- Write unit tests for your endpoints.
- Pay attention to performance, security, and code quality.

Project Description

Create a comprehensive RESTful API using Node.js, Express, and MongoDB. The API should include three endpoints with the following requirements:

1. **Complex API: User Management with Role-Based Filtering and Activity Status**
    - **Endpoint:** GET /users
    - **Description:** Fetch a list of users with their roles and activity status. Implement role-based filtering and activity status filtering.
    - **Requirements:**
        - Fetch users along with their roles and activity status.
        - If a role query parameter is provided (e.g., `/users?role=admin`), only users with that role should be returned.
        - If an active query parameter is provided (e.g., `/users?active=true`), filter users based on their activity status.
        - Combine role-based and activity status filtering when both query parameters are provided.
        - Include pagination support with `page` and `limit` query parameters.
        - Include sorting by `created_at` and `name` query parameters.
        - Ensure the API handles large datasets efficiently and includes proper indexing in the database.

Example Request:
http
Copy code
```
GET /users?role=admin&active=true&page=1&limit=10&sort=created_at
```

    -

Example Response:
json
Copy code

```
[
  {
    "id": 1,
    "name": "John Doe",
    "email": "john@example.com",
    "role": "admin",
    "is_active": true,
    "created_at": "2024-01-01T00:00:00.000Z"
  },
  ...
]
```

    -
2. **Create Product API**
    - **Endpoint:** POST /products
    - **Description:** Create a new product in the database.
    - **Requirements:**
        - Insert a new product.
        - Validate the request body to ensure all required fields are provided.
        - Ensure the product category exists in a predefined list of categories.
        - If the product price is above a certain threshold (e.g., $1000), require an additional field `approval_code`.
        - Return the created product's details in the response.

Example Request:
http
Copy code

```
POST /products
{
  "name": "New Product",
  "description": "A brand new product",
  "category": "electronics",
  "price": 199.99,
  "available": true
}
```

    -

Example Request with Approval Code:
http
Copy code

```
POST /products
{
  "name": "Expensive Product",
  "description": "A very expensive product",
  "category": "electronics",
  "price": 1500.00,
  "available": true,
  "approval_code": "APPROVED123"
}
```

-

Example Response:
json
Copy code

```
{
  "id": 4,
  "name": "New Product",
  "description": "A brand new product",
  "category": "electronics",
  "price": 199.99,
  "available": true,
  "created_at": "2024-01-01T00:00:00.000Z"
}
```

-

Implementation Guidelines

1. **Setup Instructions:**
   - Use Node.js and Express for the backend.
   - Use MongoDB as the database.
   - Use Mongoose as the ODM (Object Data Modeling) library.
   - Implement authentication and authorization using JWT (JSON Web Tokens).
2. **Endpoints to Implement:**
   - GET /users with role-based and activity status filtering, pagination, and sorting.
   - GET /customers/orders with a weekly breakdown of orders.
   - POST /products to create a new product.
3. **Requirements:**

- Properly structure the API endpoints with appropriate error handling and validation.
- Implement efficient MongoDB queries and use aggregation pipelines where required.
- Ensure the API handles large datasets and complex queries efficiently.
- Write unit tests for the APIs using a testing framework like Mocha, Chai, or Jest.
- Document the API endpoints and provide setup instructions in a README file.

Evaluation Criteria

1. **Logical Thinking and Problem-Solving:**
   - Ability to design efficient and scalable APIs.
   - Handling of complex filtering, sorting, and pagination requirements.
   - Implementation of dynamic query building based on user input.
2. **Technical Proficiency:**
   - Code quality and structure.
   - Proper use of MongoDB queries, functions, and aggregation pipelines.
   - Error handling, validation, and security measures.
   - Efficient database design and indexing.
3. **Documentation and Readability:**
   - Clarity and completeness of the README file.
   - In-code documentation and comments.
4. **Testing and Validation:**
   - Implementation of unit tests and test coverage.
   - Proper handling of edge cases and input validation.
5. **Performance and Optimization:**
   - Efficient handling of large datasets.

Additional Resources

Setup Prerequisites:

- Node.js v14+ and npm
- MongoDB v4.4+

Environment Variables:

Create a `.env` file with the following variables:
env
Copy code
```
MONGODB_URI=mongodb://localhost:27017/yourdbname
JWT_SECRET=your_jwt_secret
```

-

Database Seed Data: Use the following script to populate the MongoDB collections:

javascript
Copy code

```
// seed.js
const mongoose = require('mongoose');
const { Schema } = mongoose;

const roleSchema = new Schema({ name: { type: String, unique: true }
});
const userSchema = new Schema({
  name: String,
  email: { type: String, unique: true },
  role_id: { type: Schema.Types.ObjectId, ref: 'Role' },
  created_at: { type: Date, default: Date.now }
});
const userActivitySchema = new Schema({
  user_id: { type: Schema.Types.ObjectId, ref: 'User' },
  last_active: Date,
  is_active: { type: Boolean, default: true }
});
const customerSchema = new Schema({
  name: String,
  email: { type: String, unique: true },
  created_at: { type: Date, default: Date.now }
});
const orderSchema = new Schema({
  customer_id: { type: Schema.Types.ObjectId, ref: 'Customer' },
  total_amount: Number,
  created_at: { type: Date, default: Date.now }
});
const productSchema = new Schema({
  name: String,
  description: String,
  category: String,
  price: Number,
  available: { type: Boolean, default: true },
  created_at: { type: Date, default: Date.now }
});
```

```
const Role = mongoose.model('Role', roleSchema);
const User = mongoose.model('User', userSchema);
const UserActivity = mongoose.model('UserActivity',
userActivitySchema);
const Customer = mongoose.model('Customer', customerSchema);
const Order = mongoose.model('Order', orderSchema);
const Product = mongoose.model('Product', productSchema);

mongoose.connect(process.env.MONGODB_URI, { useNewUrlParser: true,
useUnifiedTopology: true });

async function seedData() {
  await Role.deleteMany({});
  await User.deleteMany({});
  await UserActivity.deleteMany({});
  await Customer.deleteMany({});
  await Order.deleteMany({});
  await Product.deleteMany({});

  const roles = await Role.insertMany([
    { name: 'admin' },
    { name: 'user' },
    { name: 'guest' }
  ]);

  const users = await User.insertMany([
    { name: 'Admin User', email: 'admin@example.com', role_id:
roles[0]._id },
    { name: 'Regular User', email: 'user@example.com', role_id:
roles[1]._id },
    { name: 'Guest User', email: 'guest@example.com', role_id:
roles[2]._id }
  ]);

  await UserActivity.insertMany([
    { user_id: users[0]._id, last_active: new Date(), is_active: true
},
```

```
    { user_id: users[1]._id, last_active: new Date(), is_active: true },
    { user_id: users[2]._id, last_active: new Date(), is_active: false }
  ]);

  const customers = await Customer.insertMany([
    { name: 'Customer One', email: 'customer1@example.com' },
    { name: 'Customer Two', email: 'customer2@example.com' }
  ]);

  await Order.insertMany([
    { customer_id: customers[0]._id, total_amount: 100.00 },
    { customer_id: customers[0]._id, total_amount: 200.00 },
    { customer_id: customers[1]._id, total_amount: 300.00 }
  ]);

  await Product.insertMany([
    { name: 'Product One', description: 'Description One', category: 'electronics', price: 500.00, available: true },
    { name: 'Product Two', description: 'Description Two', category: 'home', price: 150.00, available: true }
  ]);

  mongoose.disconnect();
}

seedData().catch(err => console.error(err));
```

To run the seed script, create a `seed.js` file and execute it using `node seed.js`.

Running Tests: To run your tests, use the following command:

bash
Copy code
`npm test`

Good luck, and we look forward to reviewing your submission!