deltadots

Take-Home Programming Test for Node.js Developers (Hapi, Knex, AWS, PostgreSQL)

This test evaluates real-world API development skills, covering REST API design, Hapi.js, Knex.js (PostgreSQL), authentication, validation, error handling, and best practices. You can reach out if you need any clarification. Follow the instructions – Use best coding practices , clean design pattern will be a plus.

Test Overview

Task:

- Build a simple User Management API with the following features:
- User Registration & Login (JWT Authentication)
- CRUD Operations on Users (Hapi.js, Knex, PostgreSQL)
- Validation using Joi
- Role-Based Authorization- Using CASL is bonus but you can do your own
- Pagination and Search
- Dockerized Setup

Tech Stack:

- Node.js (Hapi.js)
- Knex.js (PostgreSQL)
- JWT Authentication
- Joi for Validation
- Docker (Optional)
- AWS S3 for File Uploads (Bonus)

Test Instructions

Requirements
Create a REST API for user management with the following endpoints:

1️⃣ **User Authentication**

- **Register a new user**

Endpoint: POST /api/register
Request Body:

```
{
  "name": "John Doe",
  "email": "john@example.com",
```

"password": "securepassword"
}

- **Validations:**

o name: Required, at least 3 characters
o email: Required, valid format, unique
o password: Required, min 8 characters

- **Login**

Endpoint: POST /api/login
Request Body:

```
{
  "email": "john@example.com",
  "password": "securepassword"
}
```

- **Response:**

```
{
  "token": "JWT_TOKEN_HERE"
}
```

2️⃣ CRUD Operations on Users (Protected)

- **Get All Users (with Pagination & Search)**

Endpoint: GET /api/users?search=John&page=1&limit=10
Response:

```
{
  "data": [{ "id": 1, "name": "John Doe", "email": "john@example.com" }],
  "pagination": { "total": 100, "page": 1, "limit": 10 }
}
```

- **Get Single User**

Endpoint: GET /api/users/{id}

- **Update User (Only Admins)**

Endpoint: PUT /api/users/{id}
Body: { "name": "New Name" }

- **Delete User (Only Admins)**

Endpoint: DELETE /api/users/{id}

3️⃣ Security & Best Practices

JWT Authentication → Users must be logged in.
Role-Based Access Control (RBAC) → Admins can update/delete users.
Validation with Joi → Proper request validation.
Error Handling → Meaningful HTTP status codes & error messages.

deltadots

Evaluation Criteria

Criteria & What to Check?

✅ **API Design** Follows RESTful conventions (plural nouns, HTTP methods).
✅ **Hapi.js** Usage Uses proper route handling, plugins, and validation.
✅ **Knex.js (PostgreSQL)** Uses migrations, seeds, transactions, parameterized queries.
✅ **Authentication & Security** Uses JWT, hashed passwords, proper access control.
✅ **Error Handling** Returns meaningful status codes and messages.
✅ **Pagination & Search** Implements limit, offset, and LIKE search.
✅ **Code Structure & Cleanliness**Uses services/repositories, avoids bloated controllers.
✅ **Docker & Deployment (Bonus)** Dockerfile, .env for config, AWS S3 for file storage.

Bonus Challenges (Optional)

- File Upload (AWS S3)
- Unit Tests (Jest, Mocha)
- Rate Limiting (Hapi Rate Limit)

Submission Guidelines

- Push your code to GitHub/GitLab (public repo).
- Include a README.md explaining:

o How to set up & run the project.
o API documentation (example requests).
o Any assumptions made.

- Docker setup (docker-compose.yml for PostgreSQL) is a plus!