Chapitre 6
Open API

Qu’est ce qu’Open API

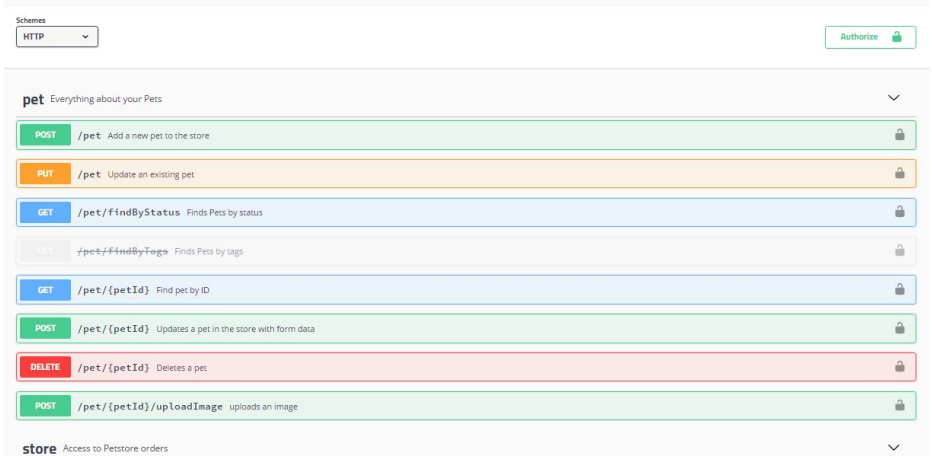
- Spécification standard pour décrire une API REST
- Aussi appelé Swagger
- Décrit notamment :
  - Endpoints
  - Paramètres
  - Réponses
  - Modèles
- Fichier généralement au format JSON

Pourquoi utiliser OpenAPI

- Documentation claire et standardisée, source de “vérité” de l’API
- Tester l’API directement depuis le navigateur
- Meilleure collaboration front / back

Swagger c’est quoi ?

- Ce sont les outils autour d’OpenAPI
- Nous utiliserons Swagger UI (interface web interactive)



Swagger pour Express.js

- Pour documenter une API Express, on installe les modules :
  a. express
  b. swagger-ui-express
  c. swagger-jsdoc

Configuration swagger (swagger.js)

```
const swaggerJsdoc = require("swagger-jsdoc");

const options = {
  definition: {
    openapi: "3.0.0",
    info: {
      title: "Mon API Express",
      version: "1.0.0",
      description: "Documentation de mon API"
    },
  },
  apis: ["./routes/*.js"],
};

module.exports = swaggerJsdoc(options);
```

Intégration dans express

```
const express = require("express");
const swaggerUi = require("swagger-ui-express");
const swaggerSpec = require("./swagger");

const app = express();

app.use("/api-docs", swaggerUi.serve, swaggerUi.setup(swaggerSpec));

app.listen(3000, () => {
  console.log("API running on http://localhost:3000");
});
```

Documenter une route

```
/**
 * @swagger
 * /users:
 *   get:
 *     summary: Récupère la liste des utilisateurs
 *     responses:
 *       200:
 *         description: Succès
 */
router.get("/users", (req, res) => {
  res.json([]);
});
```

Bonnes pratiques

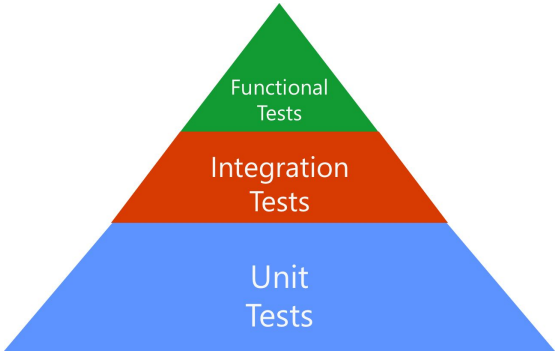
- Documenter toutes les routes
- Définir des schémas/modèles pour les objets
- Garder la doc à jour avec le code

Exercices

Chapitre 7
Tester son code

Tester son code

- Adopter une approche structurée
- Couvre différents types de tests
- S’assurer que toutes les parties fonctionnent comme prévu



Tests unitaires

- Ciblent des parties spécifiques du code, comme des méthodes spécifiques.
- Vérifient que la logique métier fonctionne correctement de manière isolée.

Tests d’intégration

- Vérifient que les différentes parties de l'application fonctionnent bien ensemble.
- Simulent l’interaction entre les couches (contrôleurs, BDD, services…).

Tests fonctionnels

- Vérifient le comportement de l'application du point de vue de l'utilisateur final
- Simulent des actions utilisateur, comme remplir un formulaire ou naviguer sur le site

Tests unitaires avec Jest

- describe : groupe de tests
- it : un comportement
- expect : assertion

math.service.js

```
function add(a, b) {
  return a + b;
}

module.exports = { add };
```

math.service.test.js

```
const { add } = require('../../src/services/math.service');

describe('add()', () => {
  it('additionne deux nombres', () => {
    expect(add(2, 3)).toBe(5);
  });

  it('gère les nombres négatifs', () => {
    expect(add(-1, 1)).toBe(0);
  });
});
```

Tests unitaires avec Jest

computer.service.ts

```ts
const createComputer = async (name) => {
    return await Computer.create({ name: name.trim() });
};

const updateComputer = async (id, name) => {
    const computer = await Computer.findByPk(id);
    computer.name = name.trim();
    await computer.save();
    return computer;
};
```

computer.service.spec.ts

```ts
jest.mock('../../src/models/computer.model', () => ({
  create: jest.fn(),
  findByPk: jest.fn()
}));

const Computer = require('../../src/models/computer.model');
const { createComputer, updateComputer } = require('../../src/services/computer.service');

describe('Computer Service (unit)', () => {

  afterEach(() => {
    jest.clearAllMocks();
  });

  it('crée un computer avec un nom trim', async () => {
    Computer.create.mockResolvedValue({ id: 1, name: 'PC' });

    const computer = await createComputer('  PC  ');

    expect(Computer.create).toHaveBeenCalledWith({ name: 'PC' });
    expect(computer.name).toBe('PC');
  });

  it('met à jour le nom du computer', async () => {
    const mockComputer = {
      id: 1,
      name: 'Old',
      save: jest.fn()
    };

    Computer.findByPk.mockResolvedValue(mockComputer);

    const updated = await updateComputer(1, '  New  ');

    expect(Computer.findByPk).toHaveBeenCalledWith(1);
    expect(mockComputer.name).toBe('New');
    expect(mockComputer.save).toHaveBeenCalled();
    expect(updated).toBe(mockComputer);
  });
});
```

diginamic FORMATION

Exercices

TP 4 : Validation des acquis