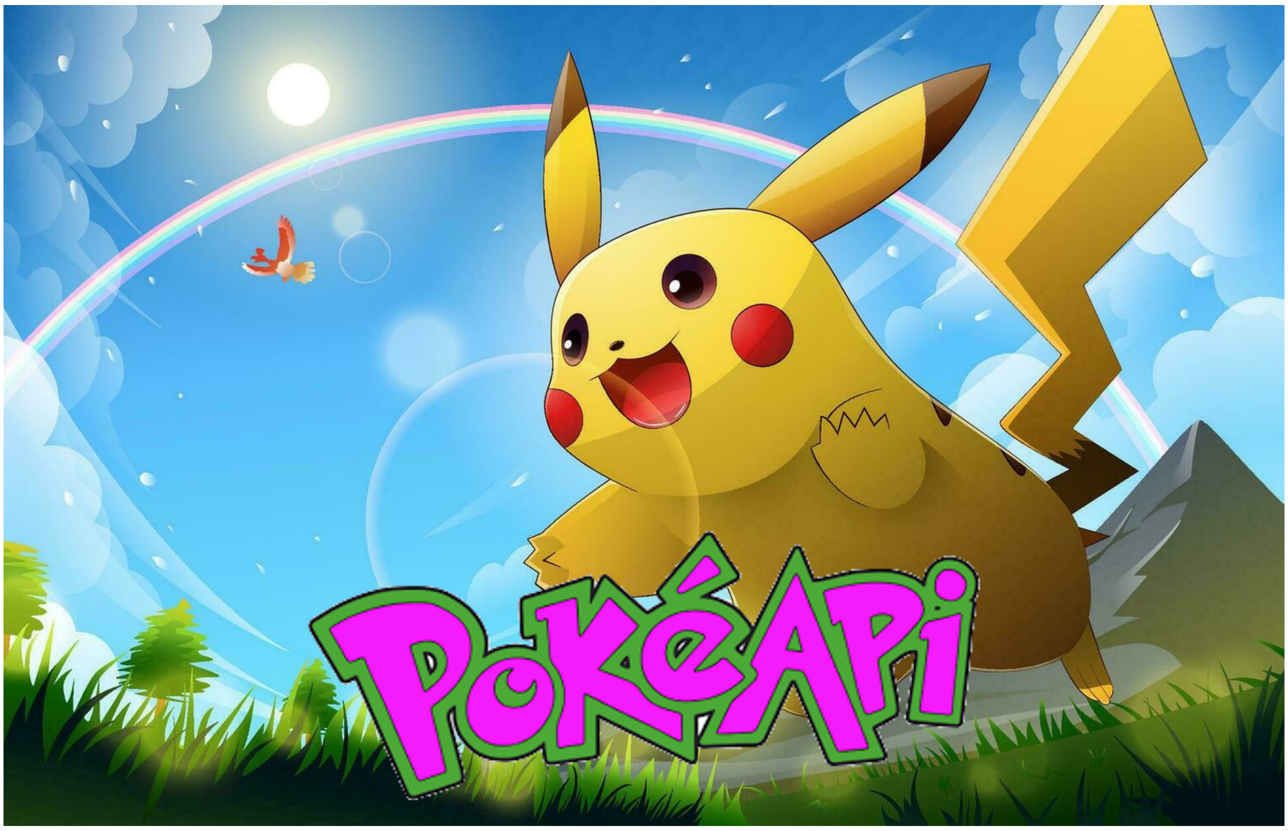

Les tests API

Dans le cadre de ce TP nous aborderons uniquement les tests API. Ces tests étant finalement les plus importants dans l'architecture 3-tiers.

Setup

Document de référence: https://axios-http.com/docs/intro
Document de référence: https://jestjs.io/fr/docs/api
Pour commencer, nous allons créer un programme complétement séparé du reste de notre projet. Ce programme aura l'unique responsabilité d'exécuter des tests. Ce choix s'explique essentiellement par la séparation des responsabilités. En effet si notre technologie API change pour une raison "x" alors cela n'auras aucun impact sur le programme de tests qui resteras toujours valide. Dans notre cas, les tests seront écrit en `NodeJS` avec `Jest` et `Axios`. Cela permettras de ré-utiliser le code source des requêtes `Axios` dans notre front. Créons un projet de base `NodeJS`:

```
# npm init
# npm install jest
# npm install axios
```

Configurer le package.json

```json
{
  "name": "Test API",
  "version": "1.0.0",
  "module": "esnext",
  "scripts": {
    "start": "node index.js",
    "test": "jest --runInBand"
  },
  "devDependencies": {
    "chai": "^5.1.1",
    "jest": "^29.7.0"
  },
  "dependencies": {
    "axios": "^1.7.5"
  }
}
```

La ligne `"test": "jest --runInBand"` permet de lancer la commande `npm run test`. Cette commande exécutera l'ensemble des tests. Il faut savoir que **Jest** exécute uniquement les fichiers qui sont post-fixés par `.test.js`. Alors créons un fichier `init.test.js` qui contiendra nos tests:

```javascript
const { expect } = require('@jest/globals');
const axios = require('axios');

const Axios = axios.create({
  baseURL: 'http://localhost:8000/api',
  headers: {
    Accept: 'application/json'
  }
});

const user = {};

describe("User Login", () => {
  test("Vérification de l'authentification", async () => {
    await login(user, {
      email: 'user@pokedex.com',
      password: 'test123'
    });
  });
});
```

```
// ---------------------------------------------------------------------------
// UTILS
// ---------------------------------------------------------------------------

async function login(user, credentials) {
  await Axios.get('/logout', {
    baseURL: 'http://localhost:8000'
  });

  const res = await Axios.get('/sanctum/csrf-cookie', {
    baseURL: 'http://localhost:8000'
  });

  Axios.defaults.headers.cookie = res.headers['set-cookie'];
  Axios.defaults.headers.common['X-XSRF-TOKEN'] = parseCSRFToken(res.headers['set-
cookie']);
  Axios.defaults.headers.common['Origin'] = 'http://localhost:8000';
  Axios.defaults.headers.common['Referer'] = 'http://localhost:8000';

  const auth = await Axios.post('/authenticate', credentials, {
    headers: { 'Content-Type': 'application/x-www-form-urlencoded' },
    baseURL: 'http://localhost:8000',
  });

  Axios.defaults.headers.cookie = auth.headers['set-cookie'];
  Axios.defaults.headers.common['X-XSRF-TOKEN'] = parseCSRFToken(auth.headers['set-
cookie'])

  for (let key in auth.data.user) {
    user[key] = auth.data.user[key];
  }
}

function parseCSRFToken(cookies) {
  const startAt = cookies[0].indexOf('=');
  const endAt = cookies[0].indexOf(';');
  const csrf = cookies[0].substring(startAt + 1, endAt - 3);
  return csrf;
}
```

Ce code est certes un peu barbare mais pas d'inquiétude. Cette partie là sera la même pour toutes vos API basé sur Laravel. Ce code initialise une connexion en tant qu'utilisateur. Les tests suivant se feront donc dans le contexte de cet utilisateur là:

```
describe("Creatures GET", () => {
  test('Récupération de la liste des créatures', async () => {
    const res = await Axios.get('/creatures');
    expect(res.data.length).toBeGreaterThanOrEqual(2);
  });

  test('Get Show', async () => {
    const creatures = await Axios.get('/creatures');
    const res = await Axios.get('/creatures/' + creatures.data[0].id);
    expect(res.data.name).toBeTruthy();
    expect(res.data.pv).toBeGreaterThanOrEqual(1);
    expect(res.data.pv).toBeLessThanOrEqual(100);
  });

  test('Get Paginate', async () => {
    const res = await Axios.get('/creatures-paginate?page=1');
    expect(res.data.maxPages).toBeGreaterThanOrEqual(0);
    expect(res.data.page).toBe('1');
    expect(res.data.creatures).toHaveLength(3);
  });
});

describe("Creatures PUT", () => {
  test("Update as user owner", async (data = { name: 'New name', _method: 'PUT' }) =>
{
    const res = await Axios.get('/creatures');
    const creature = res.data.find(c => c.user_id == user.id);
    const updateRes = await Axios.post('/creatures/' + creature.id, data);
    expect(updateRes.data.name).toBe('New name');
  });

  test("Update as not user owner", async (data = { name: 'New name', _method: 'PUT'
}) => {
    const res = await Axios.get('/creatures');
    const creature = res.data.find(c => c.user_id != user.id);
    const updateRes = await Axios.post('/creatures/' + creature.id, data, {
validateStatus: () => true });
    expect(updateRes.status).toBe(403);
    expect(updateRes.data.name).not.toBe('New name');
  });
});

describe("Creatures POST", () => {
  test("Create with good data", async (data = { name: 'Pika', pv: 10, atk: 10, def:
10, speed: 3, type: 'ELECTRIK', race: 'MOUSE', capture_rate: 4 }) => {
    const old = await Axios.get('/creatures');
    const oldNumCreatures = old.data.length;
    // before
    const createRes = await Axios.post('/creatures', data);
    // after
    const cur = await Axios.get('/creatures');
    const curNumCreatures = cur.data.length;
    expect(createRes.data.name).toBe('Pika');
    expect(curNumCreatures).toBe(oldNumCreatures + 1);
  });

  test("Create with bad data", async (data = { name: 'Pika', pv: 120, atk: 10, def:
10, speed: 3, type: 'ELECTRIK', race: 'MOUSE', capture_rate: 4 }) => {
    const old = await Axios.get('/creatures');
    const oldNumCreatures = old.data.length;
    // before
    const createRes = await Axios.post('/creatures', data, { validateStatus: () =>
true });
    // after
    const cur = await Axios.get('/creatures');
    const curNumCreatures = cur.data.length;
    expect(createRes.status).toBe(422);
    expect(curNumCreatures).toBe(oldNumCreatures);
  });
});

describe("Creatures DELETE", () => {
  test("Delete as user owner", async () => {
    const old = await Axios.get('/creatures');
    const creature = old.data.find(c => c.user_id == user.id);
    const oldNumCreature = old.data.length;
    // before
    const deleteRes = await Axios.delete('/creatures/' + creature.id);
    // after
    const cur = await Axios.get('/creatures');
    const curNumCreature = cur.data.length;
    expect(deleteRes.status).toBe(200);
    expect(curNumCreature).toBe(oldNumCreature - 1);
  });

  test("Delete as not user owner", async () => {
    const old = await Axios.get('/creatures');
    const creature = old.data.find(c => c.user_id != user.id);
    const oldNumCreature = old.data.length;
    // before
    const deleteRes = await Axios.delete('/creatures/' + creature.id, {
validateStatus: () => true });
```

```
    // after
    const cur = await Axios.get('/creatures');
    const curNumCreature = cur.data.length;
    expect(deleteRes.status).toBe(403);
    expect(curNumCreature).toBe(oldNumCreature);
  });
});
```

Lancer vos tests:

```
# php artisan migrate:fresh --seed
# npm run test
```

Enchainons sur les tests en tant qu'administrateur:

```
describe("Admin Login", () => {
  test("Vérification de l'authentification", async () => {
    await login(user, {
      email: 'admin@pokedex.com',
      password: 'test123'
    });
  });
});

describe("Admin Creatures PUT", () => {
  test("Update as admin", async (data = { name: 'New name', _method: 'PUT' }) => {
    const res = await Axios.get('/creatures');
    const creature = res.data.find(c => c.user_id != user.id);
    const updateRes = await Axios.post('/creatures/' + creature.id, data);
    expect(updateRes.data.name).toBe('New name');
  });
});

describe("Admin Creatures DELETE", () => {
  test("Delete as admin", async () => {
    const old = await Axios.get('/creatures');
    const creature = old.data.find(c => c.user_id != user.id);
    const oldNumCreature = old.data.length;
    // before
    const deleteRes = await Axios.delete('/creatures/' + creature.id);
    // after
    const cur = await Axios.get('/creatures');
    const curNumCreature = cur.data.length;
    expect(deleteRes.status).toBe(200);
    expect(curNumCreature).toBe(oldNumCreature - 1);
  });
});
```