OPENCLASSROOMS

Corrigé

P1C4 - Testez vos routes API Next.js avec Vitest

Commencez par préparer un environnement propre avant chaque test :

```typescript
import { resetStore } from "@/lib/data/store";


beforeEach(() => {
 resetStore();
});
```

Vous pouvez ensuite écrire le test de connexion réussie :

```typescript
it("should return a token when credentials are valid", async ()
=>    {
 const req = new Request(
   "<http://localhost/api/auth/login>",
   {
     method: "POST",
     body: JSON.stringify({
       email: "test@example.com",
       password: "secret"
```

```
    })
  }
);

const res = await POST(req);

expect(res.status).toBe(200);

const body = await res.json();

expect(body.token).toBeDefined();
});
```

Ajoutez ensuite le cas d'échec :

TypeScript

```
it("should return 401 when credentials are invalid", async () => {
  const req = new Request(
    "<http://localhost/api/auth/login>",
    {
      method: "POST",
      body: JSON.stringify({
        email: "test@example.com",
        password: "wrong-password"
      })
```

```
    }
  );

  const res = await POST(req);

  expect(res.status).toBe(401);
});
```

Une fois le jeton récupéré, vous pouvez construire le scénario de création de tâche. Le jeton retourné par la route de connexion est placé dans l'en-tête `Authorization` des requêtes suivantes. La route `POST /api/tasks` crée la tâche, puis `GET /api/tasks` permet de vérifier que cette tâche est bien présente dans la liste retournée.

TypeScript

```
import { POST as login } from "@/app/api/auth/login/route";
import { GET, POST } from "@/app/api/tasks/route";

it("should create a task and then return it in the list", async
() => {
  // On se connecte pour récupérer un jeton.
  const loginRes = await login(
    new Request("<http://localhost/api/auth/login>", {
      method: "POST",
            body: JSON.stringify({ email: "test@example.com",
password: "secret" }),
    }),
  );
```

```
  const { token } = await loginRes.json();


   // On crée une tâche en passant le jeton dans l'en-tête
Authorization.
  const creation = await POST(
    new Request("<http://localhost/api/tasks>", {
      method: "POST",
      headers: { Authorization: `Bearer ${token}` },
      body: JSON.stringify({ title: "Ma tâche de test" }),
    }),
  );
  expect(creation.status).toBe(201);


  // On vérifie que la tâche figure bien dans la liste.
  const liste = await GET(
    new Request("<http://localhost/api/tasks>", {
      headers: { Authorization: `Bearer ${token}` },
    }),
  );
  expect(liste.status).toBe(200);


  const taches = await liste.json();
   expect(taches.some((t: { title: string }) => t.title === "Ma
tâche de test")).toBe(true);
 });
```

Pour provoquer volontairement un échec, modifiez une assertion :

```typescript
expect(res.status).toBe(201);
```

alors que la route retourne en réalité un code `200`.

Relancez les tests. Vitest affichera alors un message indiquant précisément la valeur attendue, la valeur reçue et la ligne concernée. Cette lecture doit vous permettre d'identifier immédiatement l'origine de l'échec avant toute modification du code.

Une fois l'analyse terminée, rétablissez l'assertion correcte et vérifiez que l'ensemble de la suite de tests repasse au vert.