

Test d'intégration

Calendrier de Réservation de Salles CRS

Mariam BOUHAJJA

StreamIO

date de création : 16/12/2023

dernière date de modification: 08/06/2024

Sommaire :

Sommaire :	2
Historique de modification :	2
Tests du bon fonctionnement :	3
Exemple de tests unitaires :	4
Rapport des tests:	7

Historique de modification :

Réviseur	Date	Description
Mariem Bouhajja	16/12/2023	Création du document
Mariem Bouhajja	08/06/2024	Relecture et correction de l'orthographe

Tests du bon fonctionnement :

Pour m'assurer du bon fonctionnement de mon application j'ai effectué plusieurs types de tests :

- **des tests unitaires** : en utilisant Jest, Jest.Mock(pour simuler ma base de données) et supertest (pour pouvoir tester les routes d'une API)
- **des tests en utilisant Postman** : pour pouvoir voir les réponses des requêtes
- **des test manuels** : effectuer une action dans le Front et voir s' il y a les changements attendus dans la base de données..

Exemple de tests unitaires :

Pour expliquer les tests manuels je vais utiliser l'exemple d'une des route de l'API "/login" qui permet la connexion des clients à l'application

Voici un exemple de test de la route "/AddClient" en utilisant Jest, Supertest et Mock :

```

View Go Run ... ← → code
index.html M JS index.js M JS app.test.js M X reserHbdo.html U JS supprimCompte.js U JS reserHbdo.js U
backend > JS app.test.js > describe('POST /login') callback > it('retourne un statut 201 et message "sucess" si le mot de passe et le mail est correct') callback
97 describe('POST /login', () => {
98   beforeEach(() => {
99     jest.clearAllMocks();
100   });
101
102   //tests a faire : l'utilisateur n'existe pas
103   //mauvais mot de passe
104   //test du bon fonctionnement
105   it('retourne un statut 401 et message "Utilisateur non trouvé" si le mail ne se trouve pas dans la bdd',
106     async () => {
107       pool.query.mockResolvedValueOnce([]);
108       const response = await request(app)
109         .post('/login')
110         .send({ Adress_mail: 'test@example.com', Mot_de_pass: 'password'});
111       expect(response.status).toBe(401);
112       expect(response.body.error).toBe("Utilisateur non trouvé");
113     }
114   );
115 }
116
117 it('retourne un statut 401 et message "mot de passe incorrect!" si le mot de passe est incorrect', async () => {
118   const user = {
119     Id_Personnes: 1,
120     Adress_mail: 'test@example.com',
121     mot_de_pass: await bcrypt.hash('mdpcorrect', 4), // hash d'un mot de passe correct
122   };
123
124   pool.query.mockResolvedValueOnce([user]);
125   bcrypt.compare = jest.fn().mockResolvedValueOnce(false); // mot de passe incorrect
126
127   const response = await request(app)
128     .post('/login')
129     .send({ email: 'test@example.com', mdp: 'mdpincorrect' });
130
131   expect(response.status).toBe(401);
132   expect(response.body.error).toBe("mot de passe incorrect!");
133 });
134
135 it('retourne un statut 201 et message "sucess" si le mot de passe et le mail est correct', async () => {

```

```

ew Go Run ... ← → code
index.html M JS index.js M JS app.test.js M X reserHbedo.html U JS supprimCompte.js U JS reserHbedo.js U
backend > JS app.test.js > describe('POST /login') callback > it('retourne un statut 201 et message "sucess" si le mot de passe et le mail est correct') callback
117 describe('POST /login', () => {
129   it('retourne un statut 401 et message "mot de passe incorrect!" si le mot de passe est incorrect', async () => {
130     .send({ email: 'test@example.com', mdp: 'mdpincorrect' });
131
132     expect(response.status).toBe(401);
133     expect(response.body.error).toBe("mot de passe incorrect!");
134   });
135
136   it('retourne un statut 201 et message "sucess" si le mot de passe et le mail est correct', async () => {
137     const user = {
138       Id_Personnes: 1,
139       Adress_mail: 'test@example.com',
140       mot_de_pass: await bcrypt.hash('mdpcorrect', 4), // hash d'un mot de passe correct
141     };
142
143     pool.query.mockResolvedValueOnce([user]);
144     bcrypt.compare = jest.fn().mockResolvedValueOnce(true); // mot de passe incorrect
145
146     const response = await request(app)
147       .post('/login')
148       .send({ email: 'test@example.com', mdp: 'mdpcorrect' });
149
150     expect(response.status).toBe(201);
151     expect(response.body.message).toBe("sucess");
152     expect(response.body.pass).toBe(true);
153     expect(response.body.id).toBe(user.Id_Personnes);
154   });
155 });
156
157

```

J'ai utilisé Jest.mock pour simuler ma base de données définie
 “./database”

J'ai testé les cas d'utilisations suivants :

1. Dans le cas où l'utilisateur n'existe pas dans la base de données:
 L'API s'arrête et envoie une erreur 401 avec le message
 “Utilisateur non trouvé!”

```

//mauvais mot de passe
//test du bon fonctionnement
it('retourne un statut 401 et message "Utilisateur non trouvé" si le mail ne se trouve pas dans',
  async()=>{
    pool.query.mockResolvedValueOnce([[]]);
    const response = await request(app)
      .post('/login')
      .send({ Adress_mail: 'test@example.com', Mot_de_pass: 'password' });
    expect(response.status).toBe(401);
    expect(response.body.error).toBe("Utilisateur non trouvé");
  })

```

2. Dans le cas où l'utilisateur fournit un mot de passe incorrect :
L'API s'arrête et envoie une erreur 401 avec le message "mot de passe incorrect !"

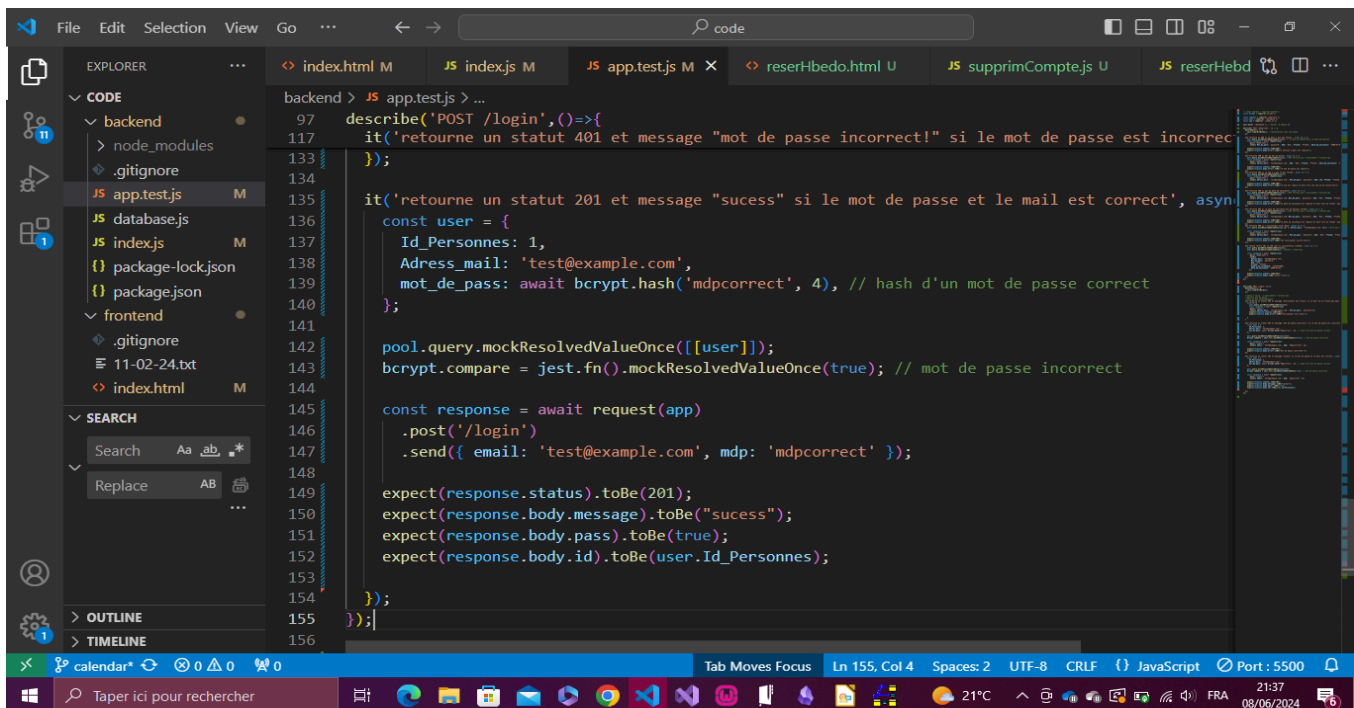
```
it('retourne un statut 401 et message "mot de passe incorrect!" si le mot de passe est incorrect', async () => {
  const user = {
    Id_Personnes: 1,
    Address_mail: 'test@example.com',
    mot_de_pass: await bcrypt.hash('mdpcorrect', 4), // hash d'un mot de passe correct
  };

  pool.query.mockResolvedValueOnce([user]);
  bcrypt.compare = jest.fn().mockResolvedValueOnce(false); // mot de passe incorrect

  const response = await request(app)
    .post('/login')
    .send({ email: 'test@example.com', mdp: 'mdpincorrect' });

  expect(response.status).toBe(401);
  expect(response.body.error).toBe("mot de passe incorrect!");
});
```

3. Test de cas de réussite : L'API effectue la requête et envoie une réponse 201 avec le message "ajout réussi"



```
describe('POST /login', () => {
  it('retourne un statut 401 et message "mot de passe incorrect!" si le mot de passe est incorrect', async () => {
    // ... (previous test code) ...
  });

  it('retourne un statut 201 et message "sucess" si le mot de passe et le mail est correct', async () => {
    const user = {
      Id_Personnes: 1,
      Address_mail: 'test@example.com',
      mot_de_pass: await bcrypt.hash('mdpcorrect', 4), // hash d'un mot de passe correct
    };

    pool.query.mockResolvedValueOnce([user]);
    bcrypt.compare = jest.fn().mockResolvedValueOnce(true); // mot de passe correct

    const response = await request(app)
      .post('/login')
      .send({ email: 'test@example.com', mdp: 'mdpcorrect' });

    expect(response.status).toBe(201);
    expect(response.body.message).toBe("sucess");
    expect(response.body.pass).toBe(true);
    expect(response.body.id).toBe(user.Id_Personnes);
  });
});
```

Rapport des tests:

PASS ./app.test.js

POST /login

✓ retourne un statut 401 et message "Utilisateur non trouvé" si le mail ne se trouve pas dans la bdd (149 ms)

✓ retourne un statut 401 et message "mot de passe incorrect!" si le mot de passe est incorrect (115 ms)

✓ retourne un statut 201 et message "sucess" si le mot de passe et le mail est correct (25 ms)

Test Suites: 1 **passed**, 1 total

Tests: 3 **passed**, 3 total

Snapshots: 0 total

Time: 3.142 s