

Test d'intégration

Gestionnaire de Réservation de Salles GRS

Mariam BOUHAJJA

StreamIO

date de création : 20/04/2024

dernière date de modification: 08/06/2024

Sommaire :

Sommaire :	2
Historique de modification :	2
Tests du bon fonctionnement :	3
Exemple de tests manuels :	4
Rapport des tests:	10

Historique de modification :

Réviseur	Date	Description
Mariem Bouhajja	20/04/2024	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 "/AddClient" qui permet d'enregistrer un nouveau client dans la base de donnée

```
//ajouter un client
app.post("/AddClient", async (req, res) => {
  const {Nom, Prenom, Adress_mail, Mot_de_pass, Numero_de_telephone, Date_de_naissance} = req.body;
  const [result] = await pool.query("SELECT * FROM utilisateurs WHERE Adress_mail= ?", [Adress_mail]);
  console.log(result);
  //l'adresse mail est nécessaire
  if (!Adress_mail) {
    return res.status(400).json({ error: "L'adresse e-mail est requise" });
  }

  //le mot de passe est nécessaire
  if (!Mot_de_pass) {
    return res.status(400).json({ error: "Le mot de passe est requis" });
  }

  // Vérifier si le nom est fourni et est une chaîne de caractères
  if (!Nom || typeof Nom !== 'string') {
    return res.status(400).json({ error: "Le nom est requis et doit être une chaîne de caractères" });
  }

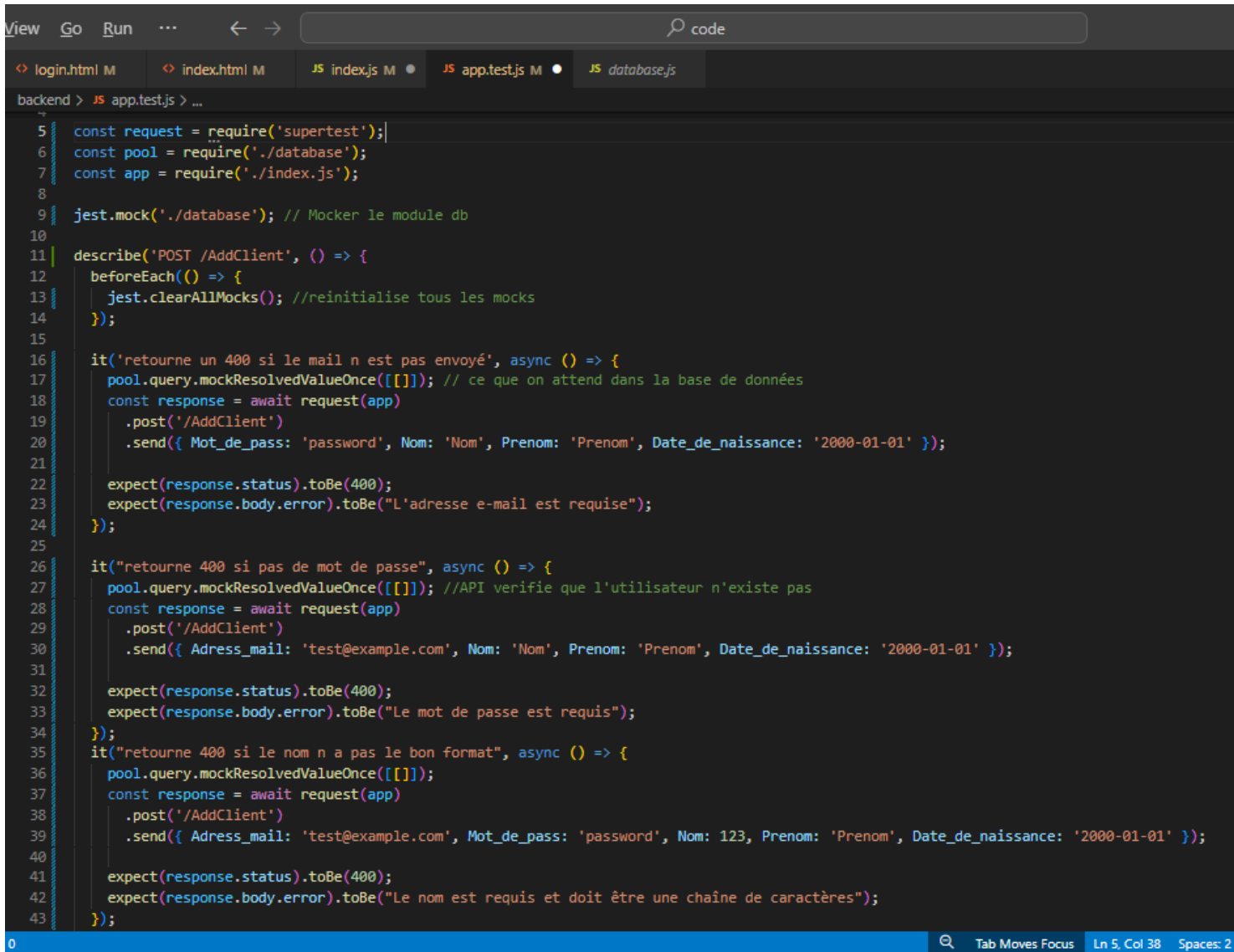
  // Vérifier si le prénom est fourni et est une chaîne de caractères
  if (!Prenom || typeof Prenom !== 'string') {
    return res.status(400).json({ error: "Le prénom est requis et doit être une chaîne de caractères" });
  }

  // Fonction pour vérifier si une date est au format 'yyyy-mm-dd'
  function isValidDateFormat(dateString) {
    const regEx = /^\\d{4}-\\d{2}-\\d{2}$/;
    return regEx.test(dateString);
  }

  //vérifier le format de la date de naissance
  if (!Date_de_naissance || !isValidDateFormat(Date_de_naissance)) {
    return res.status(400).json({ error: "La date de naissance est requise et doit être au format 'yyyy-mm-dd'" });
  }

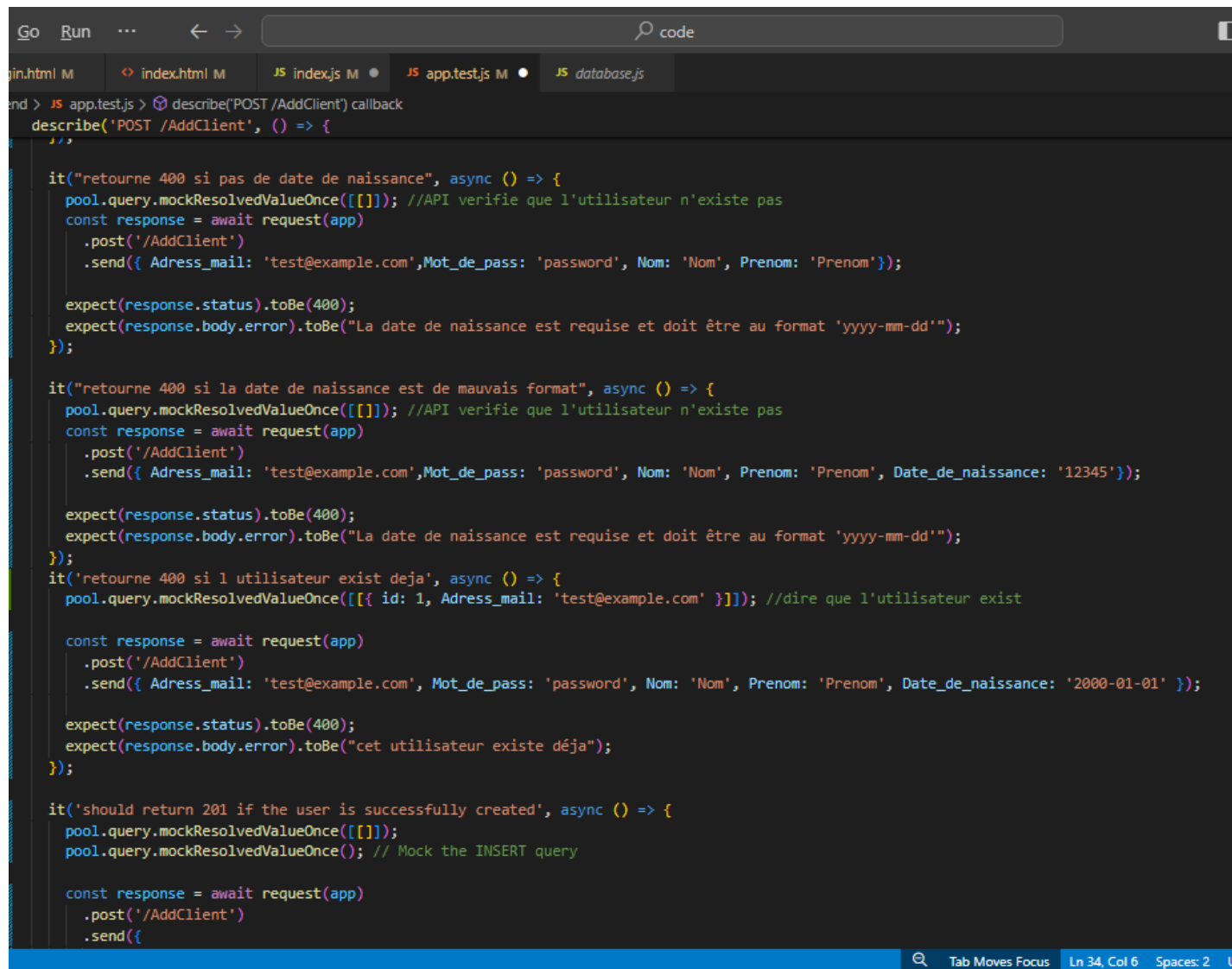
  if (result.length > 0){
    return res.status(400).json({error : "cet utilisateur existe déjà"})
  }
  //hacher le mot de pass
  const hashedPassword = await bcrypt.hash(Mot_de_pass, 4);
  await pool.query("INSERT INTO `utilisateurs`(`Nom`, `Prenom`, `Adress_mail`, `mot_de_pass`, `Numero_de_telephone`, `Date_de_naissance`) VALUES (?, ?, ?, ?, ?, ?)", [Nom, Prenom, Adress_mail, hashedPassword, Numero_de_telephone, Date_de_naissance]);
  res.status(201).json({ 'ajout réussi' });
});
```

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



```
View Go Run ... ← → code
login.html M index.html M JS index.js M JS app.test.js M JS database.js
backend > JS app.test.js > ...
5  const request = require('supertest');
6  const pool = require('../database');
7  const app = require('./index.js');
8
9  jest.mock('../database'); // Mock le module db
10
11 describe('POST /AddClient', () => {
12   beforeEach(() => {
13     jest.clearAllMocks(); //reinitialise tous les mocks
14   });
15
16   it('retourne un 400 si le mail n est pas envoyé', async () => {
17     pool.query.mockResolvedValueOnce([]); // ce que on attend dans la base de données
18     const response = await request(app)
19       .post('/AddClient')
20       .send({ Mot_de_pass: 'password', Nom: 'Nom', Prenom: 'Prenom', Date_de_naissance: '2000-01-01' });
21
22     expect(response.status).toBe(400);
23     expect(response.body.error).toBe("L'adresse e-mail est requise");
24   });
25
26   it("retourne 400 si pas de mot de passe", async () => {
27     pool.query.mockResolvedValueOnce([]); //API verifie que l'utilisateur n'existe pas
28     const response = await request(app)
29       .post('/AddClient')
30       .send({ Adress_mail: 'test@example.com', Nom: 'Nom', Prenom: 'Prenom', Date_de_naissance: '2000-01-01' });
31
32     expect(response.status).toBe(400);
33     expect(response.body.error).toBe("Le mot de passe est requis");
34   });
35   it("retourne 400 si le nom n a pas le bon format", async () => {
36     pool.query.mockResolvedValueOnce([]);
37     const response = await request(app)
38       .post('/AddClient')
39       .send({ Adress_mail: 'test@example.com', Mot_de_pass: 'password', Nom: 123, Prenom: 'Prenom', Date_de_naissance: '2000-01-01' });
40
41     expect(response.status).toBe(400);
42     expect(response.body.error).toBe("Le nom est requis et doit être une chaîne de caractères");
43   });
44 }
```

0 Tab Moves Focus Ln 5, Col 38 Spaces: 2



The screenshot shows a code editor with several tabs: 'index.html M', 'JS index.js M', 'JS app.test.js M', and 'JS database.js'. The active tab is 'JS app.test.js M'. The code is a Jest test suite for the 'AddClient' endpoint. It includes three test cases: 1) 'retourne 400 si pas de date de naissance' (returns 400 if no birth date), 2) 'retourne 400 si la date de naissance est de mauvais format' (returns 400 if birth date is in bad format), and 3) 'retourne 400 si l'utilisateur existe déjà' (returns 400 if user already exists). Each test case uses 'pool.query.mockResolvedValueOnce' to mock database responses and 'request(app)' to simulate HTTP requests. The tests use 'expect' to verify the status code and the error message in the response body. The fourth test case, 'should return 201 if the user is successfully created', is partially visible at the bottom.

```
Go Run ... ← → code
index.html M JS index.js M JS app.test.js M JS database.js
and > JS app.test.js > describe('POST /AddClient' callback
describe('POST /AddClient', () => {
  it("retourne 400 si pas de date de naissance", async () => {
    pool.query.mockResolvedValueOnce([]); //API verifie que l'utilisateur n'existe pas
    const response = await request(app)
      .post('/AddClient')
      .send({ Address_mail: 'test@example.com', Mot_de_pass: 'password', Nom: 'Nom', Prenom: 'Prenom' });

    expect(response.status).toBe(400);
    expect(response.body.error).toBe("La date de naissance est requise et doit être au format 'yyyy-mm-dd'");
  });

  it("retourne 400 si la date de naissance est de mauvais format", async () => {
    pool.query.mockResolvedValueOnce([]); //API verifie que l'utilisateur n'existe pas
    const response = await request(app)
      .post('/AddClient')
      .send({ Address_mail: 'test@example.com', Mot_de_pass: 'password', Nom: 'Nom', Prenom: 'Prenom', Date_de_naissance: '12345' });

    expect(response.status).toBe(400);
    expect(response.body.error).toBe("La date de naissance est requise et doit être au format 'yyyy-mm-dd'");
  });

  it('retourne 400 si l'utilisateur existe déjà', async () => {
    pool.query.mockResolvedValueOnce([{ id: 1, Address_mail: 'test@example.com' }]); //dire que l'utilisateur exist

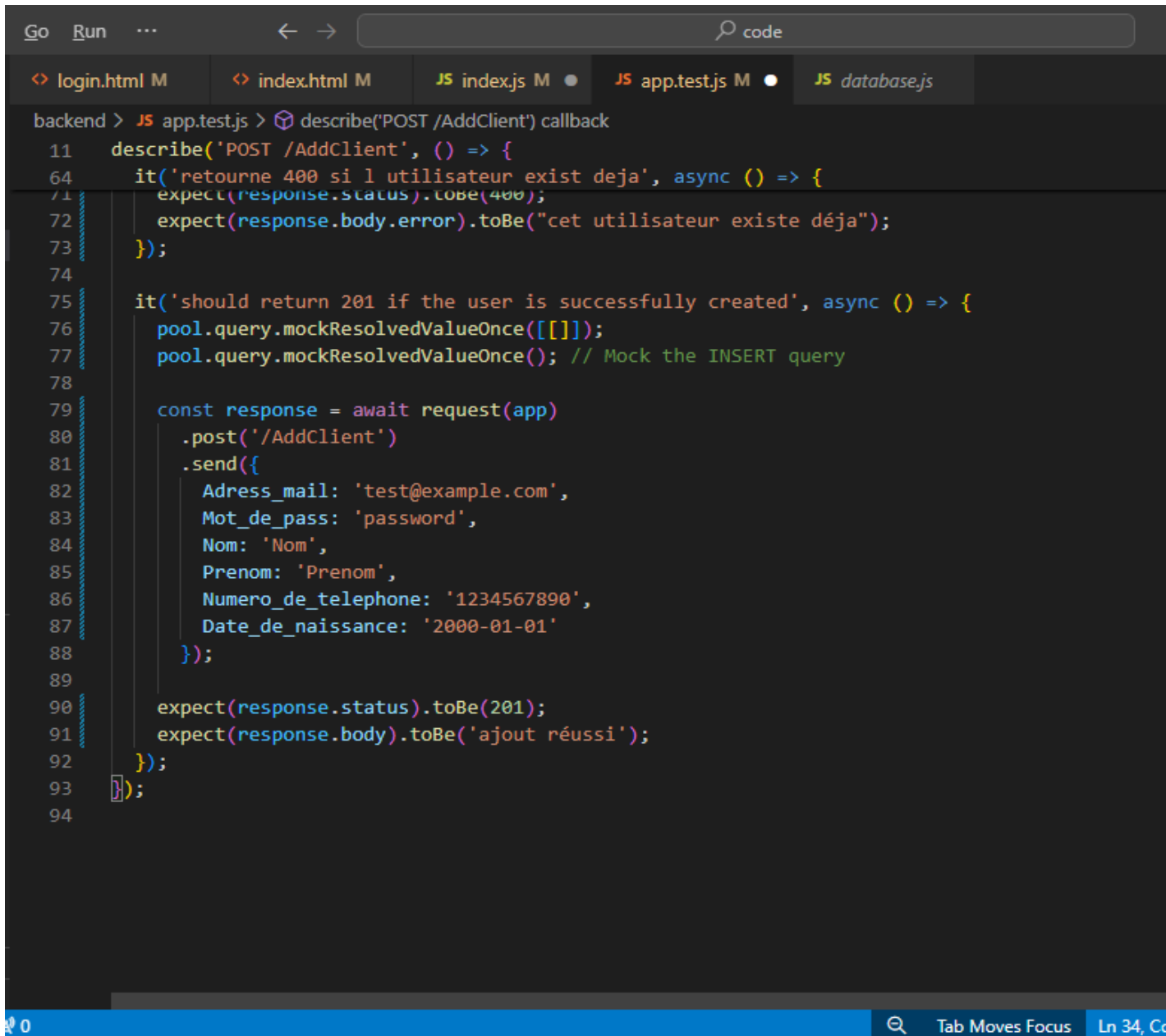
    const response = await request(app)
      .post('/AddClient')
      .send({ Address_mail: 'test@example.com', Mot_de_pass: 'password', Nom: 'Nom', Prenom: 'Prenom', Date_de_naissance: '2000-01-01' });

    expect(response.status).toBe(400);
    expect(response.body.error).toBe("cet utilisateur existe déjà");
  });

  it('should return 201 if the user is successfully created', async () => {
    pool.query.mockResolvedValueOnce([]);
    pool.query.mockResolvedValueOnce(); // Mock the INSERT query

    const response = await request(app)
      .post('/AddClient')
      .send({
```

Ln 34, Col 6 Spaces: 2 U



```
Go Run ...  ← →  code

login.html M  index.html M  JS index.js M  JS app.test.js M  JS database.js

backend > JS app.test.js > describe('POST /AddClient') callback
11  describe('POST /AddClient', () => {
64    it('retourne 400 si l utilisateur exist deja', async () => {
71      expect(response.status).toBe(400);
72      expect(response.body.error).toBe("cet utilisateur existe déjà");
73    });
74
75    it('should return 201 if the user is successfully created', async () => {
76      pool.query.mockResolvedValueOnce([]);
77      pool.query.mockResolvedValueOnce(); // Mock the INSERT query
78
79      const response = await request(app)
80        .post('/AddClient')
81        .send({
82          Adress_mail: 'test@example.com',
83          Mot_de_pass: 'password',
84          Nom: 'Nom',
85          Prenom: 'Prenom',
86          Numero_de_telephone: '1234567890',
87          Date_de_naissance: '2000-01-01'
88        });
89
90      expect(response.status).toBe(201);
91      expect(response.body).toBe('ajout réussi');
92    });
93  });
94
```

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 ne fournit pas son adress mail : L'API s'arrête et envoie une erreur 400 avec le message "L'adresse e-mail est requise"

```
it('retourne un 400 si le mail n est pas envoyé', async () => {  
  pool.query.mockResolvedValueOnce([]); // ce que on attend dans la base de données  
  const response = await request(app)  
    .post('/AddClient')  
    .send({ Mot_de_pass: 'password', Nom: 'Nom', Prenom: 'Prenom', Date_de_naissance: '2000-01-01' });  
  
  expect(response.status).toBe(400);  
  expect(response.body.error).toBe("L'adresse e-mail est requise");  
});
```

2. Dans le cas où l'utilisateur ne fournit pas son mot de passe: L'API s'arrête et envoie une erreur 400 avec le message "Le mot de passe est requis"

```
it("retourne 400 si pas de mot de passe", async () => {  
  pool.query.mockResolvedValueOnce([]); //API verifie que l'utilisateur n'existe pas  
  const response = await request(app)  
    .post('/AddClient')  
    .send({ Adress_mail: 'test@example.com', Nom: 'Nom', Prenom: 'Prenom', Date_de_naissance: '2000-01-01' });  
  
  expect(response.status).toBe(400);  
  expect(response.body.error).toBe("Le mot de passe est requis");  
});
```

3. Dans le cas où l'utilisateur fournit un nom au mauvais format (autre que chaîne de caractères) : L'API s'arrête et envoie une erreur 400 avec le message "Le nom est requis et doit être une chaîne de caractères"


```

    it("retourne 400 si le nom n a pas le bon format", async () => {
      pool.query.mockResolvedValueOnce([]);
      const response = await request(app)
        .post('/AddClient')
        .send({ Adress_mail: 'test@example.com', Mot_de_pass: 'password', Nom: 123, Prenom: 'Prenom', Date_de_naissance: '2000-01-01' });

      expect(response.status).toBe(400);
      expect(response.body.error).toBe("Le nom est requis et doit être une chaîne de caractères");
    });

```

4. Dans le cas où l'utilisateur fournit une date de naissance au mauvais format (autre que chaîne de caractères sous forme de YYYY-MM-DD) : L'API s'arrête et envoie une erreur 400 avec le message "La date de naissance est requise et doit être au format 'yyyy-mm-dd'"

```

it("retourne 400 si la date de naissance est de mauvais format", async () => {
  pool.query.mockResolvedValueOnce([]); //API verifie que l'utilisateur n'existe pas
  const response = await request(app)
    .post('/AddClient')
    .send({ Adress_mail: 'test@example.com', Mot_de_pass: 'password', Nom: 'Nom', Prenom: 'Prenom', Date_de_naissance: '12345' });

  expect(response.status).toBe(400);
  expect(response.body.error).toBe("La date de naissance est requise et doit être au format 'yyyy-mm-dd'");
});

```

5. Dans le cas où l'utilisateur existe déjà (il essaye de s'inscrire avec une address mail qui existe dans la base de donnée) : L'API s'arrête et envoie une erreur 400 avec le message "cet utilisateur existe déjà"

```

it('retourne 400 si l utilisateur exist deja', async () => {
  pool.query.mockResolvedValueOnce([[{ id: 1, Adress_mail: 'test@example.com' }]]); //dire que l'utilisateur exist

  const response = await request(app)
    .post('/AddClient')
    .send({ Adress_mail: 'test@example.com', Mot_de_pass: 'password', Nom: 'Nom', Prenom: 'Prenom', Date_de_naissance: '2000-01-01' });

  expect(response.status).toBe(400);
  expect(response.body.error).toBe("cet utilisateur existe déjà");
});

```

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

```
it('should return 201 if the user is successfully created', async () => {
  pool.query.mockResolvedValueOnce([]);
  pool.query.mockResolvedValueOnce(); //mocker l'insertion

  const response = await request(app)
    .post('/AddClient')
    .send({
      Adress_mail: 'test@example.com',
      Mot_de_pass: 'password',
      Nom: 'Nom',
      Prenom: 'Prenom',
      Numero_de_telephone: '1234567890',
      Date_de_naissance: '2000-01-01'
    });

  expect(response.status).toBe(201);
  expect(response.body).toBe('ajout réussi');
});
```

Rapport des tests:

```
PASS ./app.test.js
  POST /AddClient
    ✓ retourne un 400 si le mail n est pas envoyé (182 ms)
    ✓ retourne 400 si pas de mot de passe (111 ms)
    ✓ retourne 400 si le nom n a pas le bon format (12 ms)
    ✓ retourne 400 si pas de date de naissance (14 ms)
    ✓ retourne 400 si la date de naissance est de mauvais format (12 ms)
    ✓ retourne 400 si l utilisateur exist deja (15 ms)
    ✓ should return 201 if the user is successfully created (18 ms)

Test Suites: 1 passed, 1 total
Tests:       7 passed, 7 total
Snapshots:   0 total
Time:        2.828 s, estimated 3 s
Ran all test suites matching /app.test.js/i.
```