João Pedro Pereira Rodrigues - 20629

Jorge Alexandre Pereira Rodrigues – 20632

Documentação GITHUB:

https://github.com/JoaoRodriguesIPCA/OnboardDocs.git

Descrição do Projeto:

Onboard - Aplicação web que tem como principal função o registo das despesas de "deslocações e estadas" dos funcionários das empresas (comercial, entre outros), numa única plataforma web.

Objetivos:

- Aplicativo web permitirá centralizar num só canal todos os registos para disponibilizar ao departamento de contabilidade;
- O aplicativo web permitirá agilizar processo de pagamentos;
- O aplicativo web irá permitir a mais do que uma empresa tenha a possibilidade de ter acesso a esta ferramenta;

Requisitos:

- Registar os seguintes tipos de despesas (data, valor e categoria);
  - Este modulo chama-se "modulo de deslocações e estadas"
- Categorias de registos:
  - Alojamento;
  - Alimentação;
  - Viagens (transportes públicos, combustível, parque estacionamento, etc.);
- Todos os comerciais terão um registo na plataforma, no qual terão permissão para listar, criar, editar e apagar registos;
- Haverá um "role" chamado "admin" que permitirá a criação de utilizadores e apagar utilizadores.

Premissas:

- A entidade onBoard fará a gestão da criação das empresas na base de dados, isto é, a inserção do nome das empresas que aderiram à plataforma;
- Apenas existirá um administrador por empresa;

João Pedro Pereira Rodrigues - 20629

Jorge Alexandre Pereira Rodrigues – 20632

Especificação da API:

RESTFull URL	HTTP Action	Noun	Business Operation
/v1/Users/; <userData>	**POST**	User	createUser
/v1/Users/; { user_id }	**GET**	User	getUser
/v1/Users/; { user_id } <userData>	**PUT**	User	updateUser
/v1/Users/; { user_id } <userData>	**DELETE**	User	deleteUser
/v1/Transactions/; { user_id }	**GET**	Transaction	getUserTransactions
/v1/Transactions/; <transactionData>	**POST**	Transaction	createTransaction
/v1/Transactions/ { transaction_id} < transactionData>	**PUT**	Transaction	updateTransaction
/v1/Transactions/ { transaction_id} < transactionData>	**DELETE**	Transaction	deleteTransaction
/Auth/Signup/; <UserData>	**POST**	Auth	doSignup
/Auth/Signin/; { user_id }	**GET**	Auth	doSignin

CreateUser: Este método pretenderá criar um utilizador através do método POST, no qual enviará informação para os campos company_id, name, email, password, e role_id de utilizador;

getUser: Este método pretenderá consultar a lista de utilizadores através do método GET, através do seu id;

updateUser: Este método pretenderá consultar um determinado utilizador e editar informação através do método PUT, através do seu id;

deleteUser: Este método pretenderá consultar um determinado utilizador e apagá-lo através do método DELETE, através do seu id;

getUserTransactions: Este método pretenderá consultar as tranasações de um determinado utilizador através do método GET, através do seu id;

createTransaction: Este método pretenderá registar uma determinada transação de um utilizador através do método POST, no qual enviará informação para os campos date, category_id e amount;

updateTransaction: Este método pretenderá consultar uma determinada transação do utilizador e editar informação da mesma através do método PUT, através do seu id de transação;

deleteTransaction: Este método pretenderá consultar uma determinada transação do utilizador e apagá-la através do método DELETE, através do seu id de transação;

doSignup: Este método pretenderá criar um utilizador através do método POST, no qual enviará informação para os campos company_id, name, email, password, e role_id de administrador;

doSignin: Este método pretenderá consultar um determinado utilizador através do método GET, através do seu id;

João Pedro Pereira Rodrigues - 20629

Jorge Alexandre Pereira Rodrigues – 20632

Testes:

Onboard Backend GITHUB:

https://github.com/JoaoRodriguesIPCA/OnboardBackend.git

Users:

Teste 1 – Listar todos os utilizadores:

Este teste consiste em listar todos os utilizadores, no qual é necessário fazer um pedido GET, e é expectável receber uma resposta com o status 200 bem como uma resposta maior do que 0.

```
test.skip('Test #1 - List all users', () => {
  return request(app).get(MAIN_ROUTE)
    .set('authorization', `bearer ${user.token}`)
    .then((res) => {
      expect(res.status).toBe(200);
      expect(res.body.length).toBeGreaterThan(0);
    });
});
```

Teste 2 – Inserir utilizadores:

Este teste consiste em inserir utilizadores na base de dados, no qual é necessário fazer um pedido POST, e é expectável receber uma resposta com o status 201. Também é necessário verificar que no body vá a informação do objeto enviado, isto é, a propriedade name, bem como é necessário verificar que a password não esteja visível no body por questões de segurança.

```
test.skip('Test #2 - Insert users', () => {
  return request(app).post(MAIN_ROUTE)
    .send({
      company_id: '1', name: 'João Rodrigues', email: 'joaorodrigues@onboard.com', password: '$2a$10$ieGCSPJoXUdecZwrrwdRbua7an/AizIC1qBREyOHuPSXTZNk1atti', role_id: '1',
    })
    .set('authorization', `bearer ${user.token}`)
    .then((res) => {
      expect(res.status).toBe(201);
      expect(res.body.name).toBe('João Rodrigues');
      expect(res.body).not.toHaveProperty('password');
    });
});
```

Teste 3 – Inserir utilizador sem empresa:

Este teste consiste em inserir um utilizador na base de dados sem o preenchimento do campo company. Para tal é necessário fazer um pedido POST, e é expectável receber uma resposta com o status 400. Também é necessário passar uma mensagem de erro para identificar que no pedido houve um problema, pois o preenchimento do campo company é obrigatório.

```
test.skip('Test #3 - Insert user without company', () => {
  return request(app).post(MAIN_ROUTE)
    .send({
      name: 'João Rodrigues', email: 'joaorodrigues@onboard.com', password: '$2a$10$ieGCSPJoXUdecZwrrwdRbua7an/AizIC1qBREyOHuPSXTZNk1atti', role_id: '1',
    })
    .set('authorization', `bearer ${user.token}`)
    .then((res) => {
      expect(res.status).toBe(400);
      expect(res.body.error).toBe('Company is a mandatory attribute');
    });
});
```

Teste 4 – Inserir utilizador sem nome:

Este teste consiste em inserir um utilizador na base de dados sem o preenchimento do campo name. Para tal é necessário fazer um pedido POST, e é expectável receber uma resposta com o status 400. Também é necessário passar uma mensagem de erro para identificar que no pedido houve um problema, pois o preenchimento do campo name é obrigatório.

```
test.skip('Test #4 - Insert user without name', () => {
  return request(app).post(MAIN_ROUTE)
    .send({
      company_id: '1', email: 'joaorodrigues@onboard.com', password: '$2a$10$ieGCSPJoXUdecZwrrwdRbua7an/AizIC1qBREyOHuPSXTZNk1atti', role_id: '1',
    })
    .set('authorization', `bearer ${user.token}`)
    .then((res) => {
      expect(res.status).toBe(400);
      expect(res.body.error).toBe('Name is a mandatory attribute');
    });
});
```

Teste 5 – Inserir utilizador sem email:

Este teste consiste em inserir um utilizador na base de dados sem o preenchimento do campo email. Para tal é necessário fazer um pedido POST, e é expectável receber uma resposta com o status 400. Também é necessário passar uma mensagem de erro para identificar que no pedido houve um problema, pois o preenchimento do campo email é obrigatório.

```
test.skip('Test #5 - Insert user without email', () => {
  return request(app).post(MAIN_ROUTE)
    .send({
      company_id: '1', name: 'João Rodrigues', password: '$2a$10$ieGCSPJoXUdecZwrrwdRbua7an/AizIC1qBREyOHuPSXTZNk1atti', role_id: '1',
    })
    .set('authorization', `bearer ${user.token}`)
    .then((res) => {
      expect(res.status).toBe(400);
      expect(res.body.error).toBe('Email is a mandatory attribute');
    });
});
```

Teste 6 – Inserir utilizador sem password:

Este teste consiste em inserir um utilizador na base de dados sem o preenchimento do campo email. Para tal é necessário fazer um pedido POST, e é expectável receber uma resposta com o status 400. Também é necessário passar uma mensagem de erro para identificar que no pedido houve um problema, pois o preenchimento do campo password é obrigatório.

```
test.skip('Test #6 - Insert user without password', () => {
  return request(app).post(MAIN_ROUTE)
    .send({
      company_id: '1', name: 'João Rodrigues', email: 'joaorodrigues@onboard.com', role_id: '1',
    })
    .set('authorization', `bearer ${user.token}`)
    .then((res) => {
      expect(res.status).toBe(400);
      expect(res.body.error).toBe('Password is a mandatory attribute');
    });
});
```

Teste 7 – Inserir utilizador sem role:

Este teste consiste em inserir um utilizador na base de dados sem o preenchimento do campo role. Para tal é necessário fazer um pedido POST, e é expectável receber uma resposta com o status 400. Também é necessário passar uma mensagem de erro para identificar que no pedido houve um problema, pois o preenchimento do campo role é obrigatório.

```
test.skip('Test #7 - Insert user without role', () => {
  return request(app).post(MAIN_ROUTE)
    .send({
      company_id: '1', name: 'João Rodrigues', email: 'joaorodrigues@onboard.com', password: '$2a$10$ieGCSPJoXUdecZwrrwdRbua7an/AizIC1qBREyOHuPSXTZNk1atti',
    })
    .set('authorization', `bearer ${user.token}`)
    .then((res) => {
      expect(res.status).toBe(400);
      expect(res.body.error).toBe('Role is a mandatory attribute');
    });
});
```

Teste 8 – Gravar com password encriptada:

Este teste consiste em inserir um utilizador na base de dados com todos os campos obrigatórios. Para tal é necessário fazer um pedido POST, e é expectável com a implementação do token, tenhamos uma resposta 201. Também é expectável que quando recebermos a resposta da base dados do utilizador, que a password esteja definida, mas que esta não venha no formato original do utilizador, mas sim no formato encriptado.

```
test.skip('Test #8 - Save encryped password', async () => {
  const res = await request(app).post(MAIN_ROUTE)
    .send({
      company_id: '1', name: 'João Rodrigues', email: 'joaorodrigues@onboard.com', password: '123456', role_id: '1',
    })
    .set('authorization', `bearer ${user.token}`);
  expect(res.status).toBe(201);

  const { id } = res.body;
  const userDB = await app.services.user.findOne({ id });
  expect(userDB.password).not.toBeUndefined();
  expect(userDB.password).not.toBe('123456');
});
```

Teste 9 – Inserir um utilizador duplicado:

Este teste consiste em inserir um utilizador na base de dados com todos os campos obrigatórios. Para tal é necessário fazer um pedido POST, e é expectável que tenhamos uma resposta 400. Também é necessário passar uma mensagem de erro para identificar que no pedido houve um problema, para identificar que foi um problema de email duplicado na base de dados.

```
test.skip('Test #9 - Insert duplicated users', () => {
  return request(app).post(MAIN_ROUTE)
    .send({
      company_id: '1', name: 'João Rodrigues', email: 'joaorodrigues@onboard.com', password: '$2a$10$ieGCSPJoXUdecZwrrwdRbua7an/AizIC1qBREyOHuPSXTZNk1atti', role_id: '1',
    })
    .set('authorization', `bearer ${user.token}`)
    .then((res) => {
      expect(res.status).toBe(400);
      expect(res.body.error).toBe('Email duplicated on BD');
    });
});
```

Teste 10 – Atualizar um utilizador:

Este teste consiste em atualizar um utilizador na base de dados com todos os campos obrigatórios. Para tal é necessário fazer um insert deste objeto, identificá-lo por Id e através do PUT enviar a informação a atualizar nesse objeto. Com isto é expectável recebermos uma resposta com o status 200;

```
test.skip('Test #10 - Update user', () => {
  return app.db('users')
    .insert({
      company_id: '1', name: 'João Rodrigues - Update', email: 'joaorodrigues@onboard.com', password: '$2a$10$ieGCSPJoXUdecZwrrwdRbua7an/AizIC1qBREyOHuPSXTZNk1atti', role_id: '1',
    }, ['id'])
    .then((usr) => request(app).put(`${MAIN_ROUTE}/${usr[0].id}`)
      .set('authorization', `bearer ${user.token}`)
      .send({ name: 'User updated' }))
    .then((res) => {
      expect(res.status).toBe(200);
    });
});
```

Teste 11 – Apagar um utilizador:

Este teste consiste em apagar um utilizador na base de dados com todos os campos obrigatórios. Para tal é necessário fazer um insert deste objeto, identificá-lo por Id e através do DELETE. Com isto é expectável recebermos uma resposta com o status 204;

```
test.skip('Test #11 - Delete user', () => {
  return app.db('users')
    .insert({
      company_id: '1', name: 'João Rodrigues - Remove', email: 'joaorodrigues@onboard.com', password: '$2a$10$ieGCSPJoXUdecZwrrwdRbua7an/AizIC1qBREyOHuPSXTZNk1atti', role_id: '1',
    }, ['id'])
    .then((usr) => request(app).delete(`${MAIN_ROUTE}/${usr[0].id}`)
      .set('authorization', `bearer ${user.token}`)
      .send({ name: 'user updated' }))
    .then((res) => {
      expect(res.status).toBe(204);
    });
});
```

Teste 12 – Restringir o acesso de outro utilizador:

Este teste consiste em restringir que um determinado utilizador tenha acesso a informação de outro utilizador. Para tal é necessário inserir um determinado utilizador2 e passar o token de autenticação de um outro utilizador. Ao efetuar o acesso através do método GET, é expectável que tenhamos uma reposta do tipo 403. Também é necessário passar uma mensagem de erro para identificar que no caso de haver um problema no pedido, seja identificado qual é o problema associado, que neste caso é o utilizador não tem acesso ao recurso solicitado.

```
test.skip('Test #12 - Restrict the access from another user', () => {
  return app.db('users')
    .insert({
      id: user2.id, company_id: '1', name: 'Tiago Rodrigues - #User2', email: 'joaorodrigues@onboard.com', password: '$2a$10$ieGCSPJoXUdecZwrrwdRbua7an/AizIC1qBREyOHuPSXTZNk1atti', role_id: '1',
    }, ['id'])
    .then((usr) => request(app).get(`${MAIN_ROUTE}/${usr[0].id}`)
      .set('authorization', `bearer ${user.token}`))
    .then((res) => {
      expect(res.status).toBe(403);
      expect(res.body.error).toBe('Does not have access to the requested resource');
    });
});
```

João Pedro Pereira Rodrigues - 20629

Jorge Alexandre Pereira Rodrigues – 20632

Teste 13 – Restringir a edição de outro utilizador:

Este teste consiste em restringir que um determinado utilizador tenha permissão para editar a informação de outro utilizador. Para tal é necessário inserir um determinado utilizador2 e passar o token de autenticação de um outro utilizador. Ao efetuar o acesso através do método PUT para edição, é expectável que tenhamos uma reposta do tipo 403. Também é necessário passar uma mensagem de erro para identificar que no caso de haver um problema no pedido, seja identificado qual é o problema associado, que neste caso é o utilizador não tem acesso ao recurso solicitado.

```
test.skip('Test #13 - Restrict edition of another user', () => {
  return app.db('users')
    .insert({ id: user2.id, name: 'Tiago Rodrigues #User2' }, ['id'])
    .then((usr) => request(app).put(`${MAIN_ROUTE}/${usr[0].id}`)
      .set('authorization', `bearer ${user.token}`))
    .then((res) => {
      expect(res.status).toBe(403);
      expect(res.body.error).toBe('Does not have access to the requested resource');
    });
});
```

Teste 14 – Restringir a funcionalidade de apagar a outro utilizador:

Este teste consiste em restringir que um determinado utilizador tenha permissão para apagar a informação de outro utilizador. Para tal é necessário inserir um determinado utilizador2 e passar o token de autenticação de um outro utilizador. Ao efetuar o acesso através do método DELETE para apagar, é expectável que tenhamos uma reposta do tipo 403. Também é necessário passar uma mensagem de erro para identificar que no caso de haver um problema no pedido, seja identificado qual é o problema associado, que neste caso é o utilizador não tem acesso ao recurso solicitado.

```
test.skip('Test #14 - Restrict deletion of another user', () => {
  return app.db('users')
    .insert({ id: user2.id, name: 'Tiago Rodrigues #User2' }, ['id'])
    .then((usr) => request(app).delete(`${MAIN_ROUTE}/${usr[0].id}`)
      .set('authorization', `bearer ${user.token}`))
    .then((res) => {
      expect(res.status).toBe(403);
      expect(res.body.error).toBe('Does not have access to the requested resource');
    });
});
```

Teste 15 – Inserir um utilizador com as devidas permissões:

Este teste consiste em inserir um utilizador com a devida permissão. Para tal é necessário inserir um determinado utilizador e passar o token de autenticação do mesmo. Ao efetuar o acesso através do método POST para aceder, é expectável que tenhamos uma reposta do tipo 201. Além disso é expectável que na posição zero do body da resposta para o parâmetro role_id seja o 1, pois é o que terá a permissão.

João Pedro Pereira Rodrigues - 20629

Jorge Alexandre Pereira Rodrigues – 20632

```
test.skip('Test #15 - Insert an user with the right role_id permitions', () => {
  request(app).post(MAIN_ROUTE)
    .insert({ name: 'Jorge Rodrigues #Role 1', role_id: '1' })
    .set('authorization', `bearer ${user.token}`)
    .then((res) => {
      expect(res.status).toBe(201);
      expect(res.body[0].role_id).toBe('1');
    });
});
```

Teste 16 – Inserir um utilizador sem a devida permissão:

Este teste consiste em restringir que um determinado utilizador tenha permissão para inserir um utilizador novo. Para tal é necessário enviar um determinado utilizador e passar o token de autenticação do utilizador. Ao efetuar o acesso através do método POST para inserir, é expectável que tenhamos uma reposta do tipo 403. Também é necessário passar uma mensagem de erro para identificar que no caso de haver um problema no pedido, seja identificado qual é o problema associado, que neste caso é o utilizador não tem acesso ao recurso solicitado. Também é necessário passar uma mensagem de erro para identificar que no caso de haver um problema no pedido, seja identificado qual é o problema associado, que neste caso é o utilizador ter permissão.

```
test.skip('Test #16 - Insert an user without the role permitions', () => {
  return request(app).post(MAIN_ROUTE)
    .send({
      company_id: '1', name: 'João Rodrigues #Insert', email: 'joaorodrigues@onboard.com', password: '$2a$10$ieGCSPJoXUdecZwrrwdRbua7an/AizIC1qBREyOHuPSXTZNk1atti', role_id: '2',
    })
    .set('authorization', `bearer ${user.token}`)
    .then((res) => {
      expect(res.status).toBe(403);
      expect(res.body.error).toBe('This user does not have permition');
    });
});
```

Teste 16 – Apagar um utilizador sem a devida permissão:

Este teste consiste em restringir que um determinado utilizador tenha permissão para apagar um utilizador. Para tal é necessário enviar um determinado utilizador e passar o token de autenticação do utilizador. Ao efetuar o acesso através do método DELETE para apagar, é expectável que tenhamos uma reposta do tipo 403. Também é necessário passar uma mensagem de erro para identificar que no caso de haver um problema no pedido, seja identificado qual é o problema associado, que neste caso é o utilizador não tem acesso ao recurso solicitado. Também é necessário passar uma mensagem de erro para identificar que no caso de haver um problema no pedido, seja identificado qual é o problema associado, que neste caso é o utilizador ter permissão.

```
test.skip('Test #17 - Restrict delete permitions for roles', () => {
  return app.db('users')
    .insert({
      company_id: '1', name: 'João Rodrigues #Insert', email: 'joaorodrigues@onboard.com', password: '$2a$10$ieGCSPJoXUdecZwrrwdRbua7an/AizIC1qBREyOHuPSXTZNk1atti', role_id: '2',
    }, ['id'])
    .then((usr) => request(app).delete(`${MAIN_ROUTE}/${usr[0].id}`)
      .set('authorization', `bearer ${user.token}`))
    .then((res) => {
      expect(res.status).toBe(403);
      expect(res.body.error).toBe('Does not have the permition to the requested resource');
    });
});
```

Auths:

Teste 1 – Inserir um utilizador com o signup:

João Pedro Pereira Rodrigues - 20629

Jorge Alexandre Pereira Rodrigues – 20632

Este teste consiste em inserir um utilizador acedendo à rota de signup, enviando o objeto do utilizador a criar. É expectável que tenhamos como resposta o status 201, que a propriedade body.name seja igual à propriedade do objeto enviado, que o body tenha a propriedade email, e que não tenha a propriedade email no body.

```
test.skip('Test #1 - Create user with signup', () => {
  const email = `${Date.now()}@onboard.com`;
  return request(app).post('/auth/signup')
    .send({
      company_id: '1', name: 'Jorge Rodrigues', email, password: '123456', role_id: '1',
    })
    .then((res) => {
      expect(res.status).toBe(201);
      expect(res.body.name).toBe('Jorge Rodrigues');
      expect(res.body).toHaveProperty('email');
      expect(res.body).not.toHaveProperty('password');
    });
});
```

Teste 2 – Receber o token ao autenticar:

Este teste consiste em inserir as credenciais de um utilizador acedendo à rota de signin, enviando o objeto do utilizador que quer aceder. É expectável que tenhamos como resposta o status 200 e que o body da resposta tenha a propriedade token.

```
test.skip('Test #2 - Recieve token when authenticating ', () => {
  const email = `${Date.now()}@onboard.com`;
  return app.services.user.save(
    {
      company_id: '1', name: 'Jorge Rodrigues', email, password: '123456', role_id: '1',
    },
  ).then(() => request(app).post('/auth/signin')
    .send({ email, password: '123456' }))
    .then((res) => {
      expect(res.status).toBe(200);
      expect(res.body).toHaveProperty('token');
    });
});
```

Teste 3 – Tentativa falhada de autenticação por password errada:

Este teste consiste em inserir as credenciais de um utilizador acedendo à rota de signin, enviando o objeto do utilizador que quer aceder, mas com a password errada. É expectável que tenhamos como resposta o status 400, e além disso, que passemos uma mensagem de erro para identificar que ouve um problema de password ou utilizador errados.

```
test.skip('Test #3 - Wrong authentication attempt with wrong password', () => {
  const email = `${Date.now()}@onboard.com`;
  return app.services.user.save(
    {
      company_id: '1', name: 'Jorge Rodrigues', email, password: '123456', role_id: '1',
    },
  ).then(() => request(app).post('/auth/signin')
    .send({ email, password: '654321' }))
    .then((res) => {
      expect(res.status).toBe(400);
      expect(res.body.error).toBe('Invalid user or password');
    });
});
```

Teste 4 – Tentativa falhada de autenticação por password errada:

Este teste consiste em inserir as credenciais de um utilizador acedendo à rota de signin, enviando o objeto do utilizador que quer aceder, mas com o email errado. É expectável que tenhamos como resposta o status 400, e além disso, que passemos uma mensagem de erro para identificar que ouve um problema de password ou utilizador errados.

```
test.skip('Test #4 - Wrong authentication attempt with wrong user', () => {
  return request(app).post('/auth/signin')
    .send({ email: 'doesnotexist@mail.com', password: '654321' })
    .then((res) => {
      expect(res.status).toBe(400);
      expect(res.body.error).toBe('Invalid user or password');
    });
});
```

Teste 5 – Aceder a rotas protegidas:

Este teste consiste em proteger as rotas, e para tal é necessário fazer um pedido GET à rota /v1/users e o expectável é que tenhamos uma resposta de 401.

```
test.skip('Test #5 - Access to protected routes', () => {
  return request(app).get('/v1/users')
    .then((res) => {
      expect(res.status).toBe(401);
    });
});
```

Transactions:

Teste 1 – Listar transações de um utilizador:

João Pedro Pereira Rodrigues - 20629

Jorge Alexandre Pereira Rodrigues – 20632

Este teste consiste em listar as transações de um utilizador. Para tal é necessário fazermos insert de dois utilizadores distinguidos por user.id e user2.id e fazer um pedido GET com o token do user. Assim é expectável recebermos um status 200, ter uma resposta no body com o comprimento de 1 e verificar que a primeira posição da propriedade amount é de facto a correspondente ao user.

```
test.skip('Test #1 - List only user transactions', () => {
  return app.db('transactions').insert([
    {
      user_id: user.id, date: new Date(), category_id: '1', amount: 60,
    },
    {
      user_id: user2.id, date: new Date(), category_id: '3', amount: 200,
    },
  ]).then(() => request(app).get(MAIN_ROUTE)
    .set('authorization', `bearer ${user.token}`)
    .then((res) => {
      expect(res.status).toBe(200);
      expect(res.body).toHaveLength(1);
      expect(res.body[0].amount).toBe('60.00');
    }));
});
```

Teste 2 – Inserir transações de um utilizador:

Este teste consiste em inserir uma transação de um utilizador. Para tal é necessário fazermos insert de uma determinada transação de um utilizador através de um pedido POST com o token do user. Assim é expectável recebermos uma resposta com o status 201, ter uma verificação que o user_id do body corresponde ao id do user e que a propriedade amount é correspondente ao do utilizador.

```
test.skip('Test #2 - Insert user transactions', () => {
  return request(app).post(MAIN_ROUTE)
    .set('authorization', `bearer ${user.token}`)
    .send({
      user_id: user.id, date: new Date(), category_id: '1', amount: 60,
    })
    .then((res) => {
      expect(res.status).toBe(201);
      expect(res.body.user_id).toBe(user.id);
      expect(res.body.amount).toBe('60.00');
    });
});
```

Teste 3 – Validação da criação de uma transação:

João Pedro Pereira Rodrigues - 20629

Jorge Alexandre Pereira Rodrigues – 20632

Este teste consiste em inserir uma determinada transação de um utilizador na base de dados. Para tal serão feitas tentativas de inserção sem cada um dos campos obrigatórios, isto é, a date, a category_id e o ammount, através do método POST.O expectável será receber uma resposta com o status 400. Também é necessário passar uma mensagem de erro para identificar que no pedido houve um problema, pois o preenchimento do campo de cada um destes campos é obrigatório.

```
describe('Test #3 - Validation of transaction creation', () => {
  const testTemplate = (newData, errorMessage) => {
    return request(app).post(MAIN_ROUTE)
      .set('authorization', `bearer ${user.token}`)
      .send({
        user_id: user.id, date: new Date(), category_id: '1', amount: 60, ...newData,
      })
      .then((res) => {
        expect(res.status).toBe(400);
        expect(res.body.error).toBe(errorMessage);
      });
  };
  test.skip('Test #3.1 - Insert transaction without Date', () => testTemplate({ date: null }, 'DATE is a mandatory attribute'));

  test.skip('Test #3.2 - Insert transaction without Category', () => testTemplate({ category_id: null }, 'CATEGORY is a mandatory attribute'));

  test.skip('Test #3.3 - Insert transaction without Amount', () => testTemplate({ amount: null }, 'AMOUNT is a mandatory attribute'));
});
```

Teste 4 – Listar as transações de utilizadores por transaction_id:

Este teste consiste em listar as transações de um utilizador por transaction_id. Para tal é necessário fazermos insert de uma transação e fazer um pedido GET com o token do user. Assim é expectável recebermos um status 200, ter uma resposta no body com o primeiro resultado e que o amount seja de acordo com a transação enviada.

```
test.skip('Test #4 - List user transactions by transaction id', () => {
  return app.db('transactions').insert({
    user_id: user.id, date: new Date(), category_id: '1', amount: 60,
  }, ['id'])
    .then((result) => request(app).get(`${MAIN_ROUTE}/${result[0].id}`)
      .set('authorization', `bearer ${user.token}`)
      .then((res) => {
        expect(res.status).toBe(200);
        expect(res.body.id).toBe(result[0].id);
        expect(res.body.amount).toBe('60.00');
      }));
});
```

Teste 5 – Atualizar uma transação:

Este teste consiste em atualizar uma transação na base de dados com todos os campos obrigatórios. Para tal é necessário fazer um insert deste objeto, identificá-lo por Id e através do PUT enviar a informação a atualizar nesse objeto. Com isto é expectável recebermos uma resposta com o status 200, que a propriedade id no body da resposta seja a mesmo id do resultado na posição 0 de resultados e que a propriedade amount no body da resposta seja a mesma do objeto enviado;

```
test.skip('Test #5 - Update transaction', () => {
  return app.db('transactions').insert({
    user_id: user.id, date: new Date(), category_id: '1', amount: 60,
  }, ['id'])
    .then((result) => request(app).put(`${MAIN_ROUTE}/${result[0].id}`)
      .set('authorization', `bearer ${user.token}`)
      .send({ amount: '100.00' })
      .then((res) => {
        expect(res.status).toBe(200);
        expect(res.body.id).toBe(result[0].id);
        expect(res.body.amount).toBe('100.00');
      }));
});
```

Teste 6 – Apagar uma transação:

Este teste consiste em apagar uma transação na base de dados. Para tal é necessário fazer um insert deste objeto com todos os campos obrigatórios., identificá-lo por Id e através do DELETE fazer a operação. Com isto é expectável recebermos uma resposta com o status 204;

```
test.skip('Test #6 - Remove transaction ', () => {
  return app.db('transactions').insert({
    user_id: user.id, date: new Date(), category_id: '1', amount: 60,
  }, ['id'])
    .then((result) => request(app).delete(`${MAIN_ROUTE}/${result[0].id}`)
      .set('authorization', `bearer ${user.token}`)
      .then((res) => {
        expect(res.status).toBe(204);
      }));
});
```

Teste 7 – Restringir o acesso a uma transação de outro utilizador:

Este teste consiste em restringir que um determinado utilizador tenha acesso a transações de outro utilizador. Para tal é necessário inserir um determinado utilizador2 e passar o token de autenticação de um outro utilizador. Ao efetuar o acesso através do método GET, é expectável que tenhamos uma reposta do tipo 403. Também é necessário passar uma mensagem de erro para identificar que no caso de haver um problema no pedido, seja identificado qual é o problema associado, que neste caso é que o recurso não pertence ao utilizador.

```
test.skip('Test #7 - Access transaction from another user', () => {
  return app.db('transactions').insert({
    user_id: user2.id, date: new Date(), category_id: '1', amount: 60,
  }, ['id'])
    .then((result) => request(app).delete(`${MAIN_ROUTE}/${result[0].id}`)
      .set('authorization', `bearer ${user.token}`)
      .then((res) => {
        expect(res.status).toBe(403);
        expect(res.body.error).toBe('This resource does not belong to this user');
      }));
});
```

Teste 8 – O valor da transação deve ser positivo:

Este teste consiste em garantir que o valor da transação é sempre positivo. Para tal é necessário inserir uma transação negativa com o send. Ao efetuar o envio através do método POST, é expectável que tenhamos uma reposta do tipo 201, que o id de utilizador presente no body da resposta seja o mesmo do objeto enviado e que o amount seja com valor positivo.

```
test.skip('Test #8 - Transaction ammount must be positive', () => {
  return request(app).post(MAIN_ROUTE)
    .set('authorization', `bearer ${user.token}`)
    .send({
      user_id: user.id, date: new Date(), category_id: '1', amount: -60,
    })
    .then((res) => {
      expect(res.status).toBe(201);
      expect(res.body.user_id).toBe(user.id);
      expect(res.body.amount).toBe('60.00');
    });
});
```