

APÊNDICE B - DIRETRIZ PARA APLICAÇÃO DO TESTE PONTA A PONTA EM UMA APLICAÇÃO WEB

B.1 CENÁRIO DE TESTE

Para o cenário de teste foi desenvolvido uma aplicação exclusiva, utilizando a arquitetura cliente e servidor, a linguagem escolhida foi JavaScript para construir o código de ambas camadas, o que torna mais simples o desenvolvimento pelo fato da linguagem ser isomórfica em ambas camadas.

Para a construção do servidor que é onde realizar as validações necessárias e do banco de dados que é onde armazena fisicamente as informações, enviando e salvando os dados conforme as demandas do cliente. A figura 7 representa as entidades e seus relacionamentos.

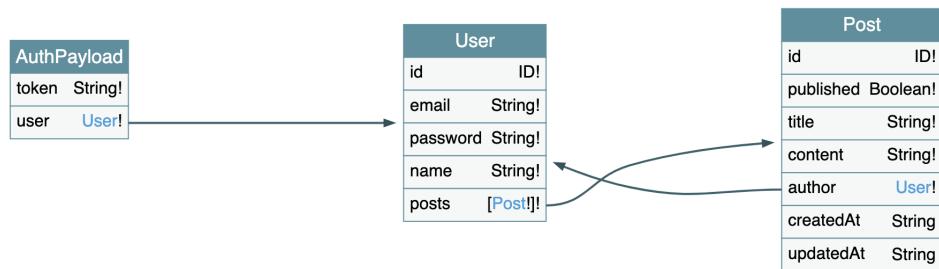


Figura 7: Diagrama Relacional

Na figura 7 fica explícito que um (Usuario = user) contém os atributos (id, email, password, name e posts), os posts representam uma lista de publicações. Uma (Publicação = Post) contém os atributos (id, published, title, content, author, createdAt, updateAt), o (Autor = author) representa o autor da publicação. Por fim o (Carga de Dados de Autenticação = AuthPayload) que contém os atributos (token e user), o user representa o Usuário e o token a chave de acesso pra API.

Observação: Os caracteres [] representam uma lista e o caractere ! representa obrigatoriedade.

O servidor é composto por duas camadas:

- A camada da aplicação é responsável por qualquer funcionalidade que não esteja diretamente relacionada à gravação ou leitura de dados do banco de dados, como por exemplo lógicas que envolva regras de negócio, autenticação e permissões, integrações de terceiros, etc.
- A camada de banco de dados fornece operações CRUD (Create, Read, Update e Delete), em tempo real para o banco de dados e é totalmente cuidada pela Prisma.

Para a construção do cliente foi utilizado uma metodologia baseada em uma analogia científica que nos ajuda a pensar em interfaces de uma maneira mais escalável, tal metodologia é denominada Atomic Design - Design Atômico. O criador Brad Frost desenvolveu esse modelo mental com o objetivo de tornar o desenvolvimento de páginas web mais sofisticado, no que diz a respeito a reaproveitar componentes ao máximo, por isso se preza por componentes fragmentados, e os compõe para formar as páginas (ATOMIC..., 2016).

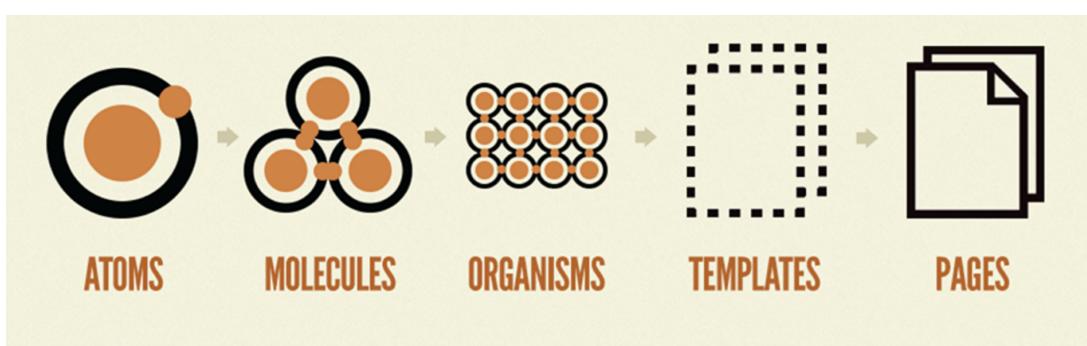


Figura 8: Atomic Design - Design Atômico (ATOMIC..., 2016)

A figura 8 demonstra de forma abstrata o modelo mental proposto. Pensando em uma aplicação web, podemos definir cada um desses elementos como:

- **Atoms:** Uma átomo pode ser um input, button, h1 ou um p do HTML, ou por exemplo um elemento customizado como um Alert, Loading ou Card. O átomo nesse caso é o menor elemento da aplicação.
- **Molecules:** Uma molécula pode ser uma ou mais div, section, table ou nav do HTML ou por exemplo um elemento customizado como um Dropdown, Accordion ou Table. A molécula nesse caso é o segundo menor elemento da aplicação, sendo composta por dois ou mais átomos.

- Organisms:** Um organismo pode ser uma header, aside, footer ou main do HTML ou por exemplo um elemento customizado como um NavBar, TabContent. O organismo nesse caso é o terceiro menor elemento da aplicação, sendo composto por átomos e moléculas.
- Templates:** Um template é o body do HTML ou por exemplo é o que defino o espaço que cada elemento ocupará na página e onde cada elemento será exibido. O organismo nesse caso é o segundo maior elemento da aplicação, sendo composto por átomos moléculas e organismos.
- Pages:** Uma página é o elemento que gerencia a parte lógica e passa as devidas propriedades para cada um dos componentes dentro do template. A página nesse caso é o maior elemento, sendo composta por um template e por dados lógicos e informações necessárias.

Por meio do cliente, o usuário pode realizar um cadastro, acessar a plataforma, cadastrar, editar, publicar e excluir seus rascunhos, editar e excluir suas publicações, visualizar suas publicações e a de outros usuários na página inicial e alterar seus dados pessoais e senha dentro da plataforma.

Segue algumas capturas de tela do cenário de teste:

A figura 9 se refere a tela de Cadastro, onde o usuário poderá se cadastrar.

Figura 9: Página de Cadastrar Usuário

A figura 10 se refere a tela de Login, onde o usuário poderá utilizar o seu cadastro para acessar à plataforma e usufruir dos serviços disponíveis dentro dela.

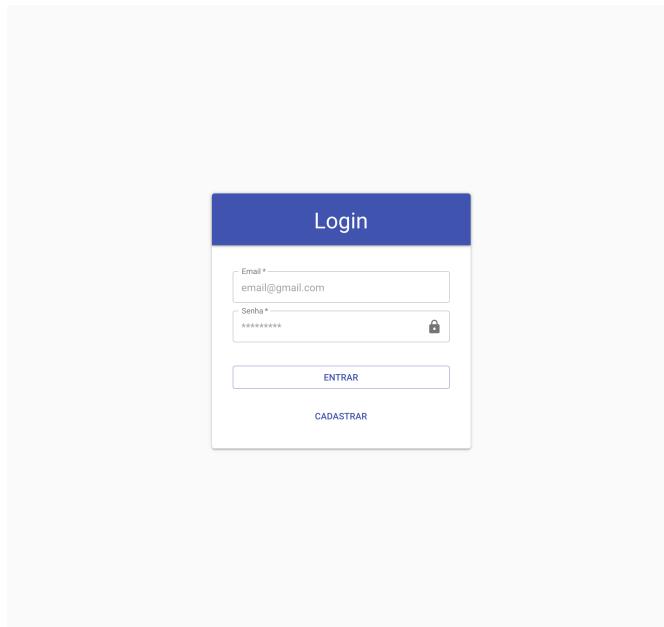


Figura 10: Página de Entrar com Usuário

A figura 11 se refere a tela de Página Inicial, onde o usuário poderá visualizar as publicações escritas e publicadas por ele mesmo ou por outros usuários da plataforma.

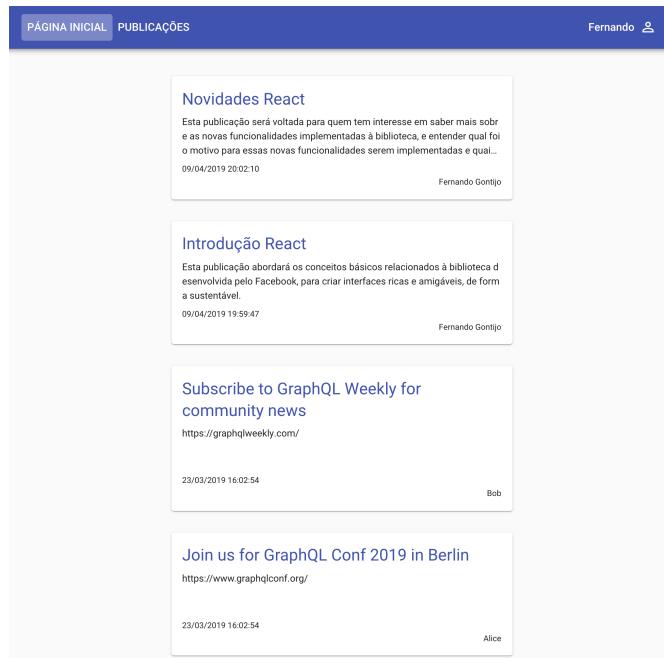


Figura 11: Página de Página Inicial

A figura 12 se refere a tela de Rascunhos, onde o usuário poderá visualizar e

realizar ações nos rascunhos escritos por ele mesmo.

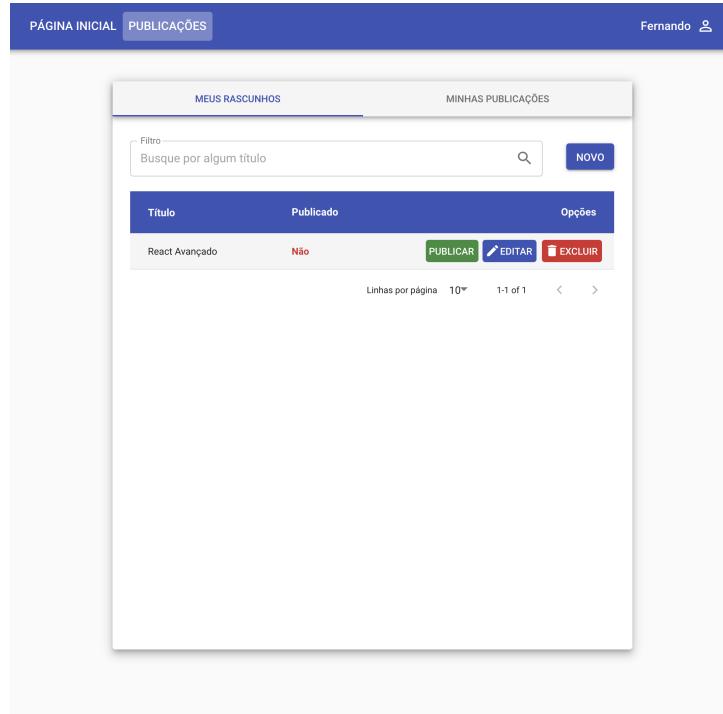


Figura 12: Página de Rascunhos

A figura 13 se refere a tela de Adição de Rascunho, onde o usuário poderá adicionar um rascunho por vez.

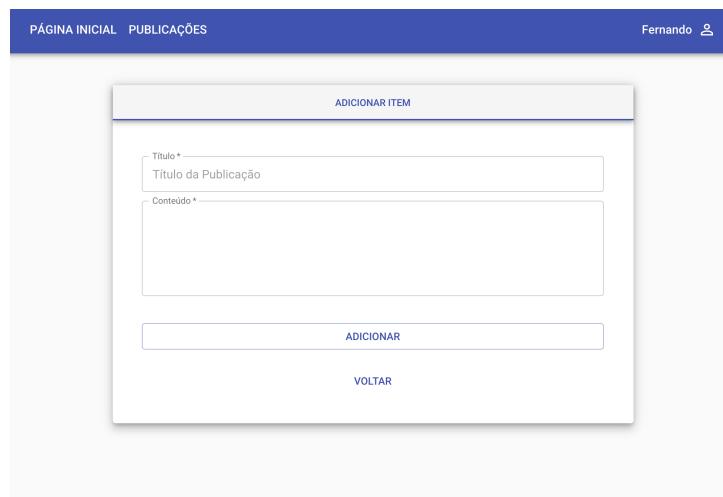


Figura 13: Página de Adicionar Rascunho

A figura 14 se refere a tela de Configurações, onde o usuário poderá visualizar e/ou alterar suas informações pessoais.

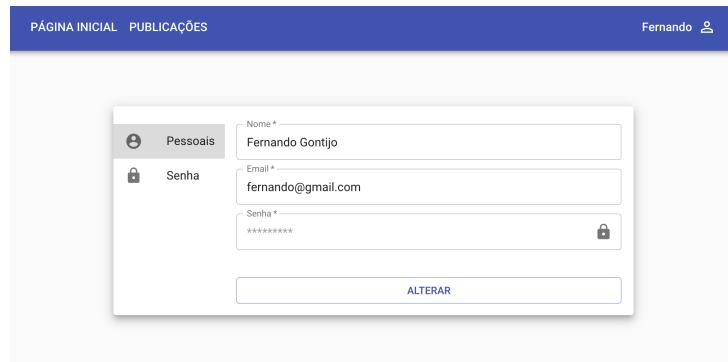


Figura 14: Página de Configurações

B.2 METODOLOGIA

Foi realizado uma pesquisa exploratória sobre uma abordagem relacionada a atividades de VV&T, busca-se entender a essência da abordagem de teste denominada teste *ponta a ponta*, com intuito de responder as três questões de pesquisa elencadas no *Apêndice A*.

Tal pesquisa exploratória consiste em criar 8 casos de testes utilizando a técnica de teste ponta a ponta, onde foi implementado cada uma das rotinas utilizando a ferramenta Cypress para simular a interação de um usuário com a aplicação.

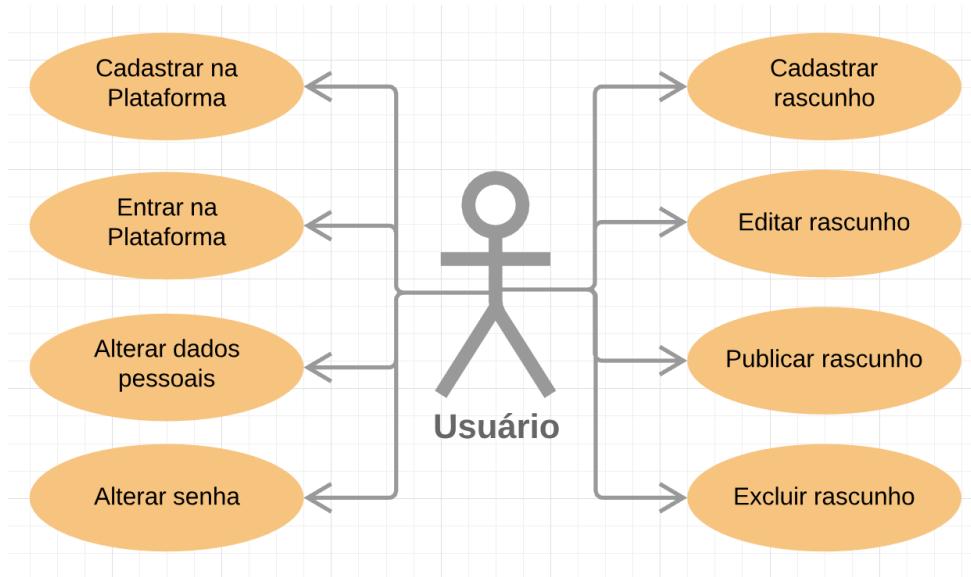


Figura 15: Diagrama de Casos de Uso

O diagrama de Casos de Uso 15 representa o usuário e as interações que ele pode realizar na plataforma.

O cenário de teste foi desenvolvido e anexado no (GITHUB, 2019) como

projeto complementar nos seguintes diretórios **frontend** e **backend**. Os testes utilizando o Cypress foram construídos no diretório **E2E** e também foram disponibilizados no (GITHUB, 2019), segue o link com o código fonte completo do projeto: https://github.com/fernandobd42/projeto_tcc.

Para executar o projeto a primeiro e mais importante atividade é realizar a instalação do Node, para que possamos executar a diretriz. Acesse o site <https://nodejs.org/en/download>, baixe e instale o Node. Para garantir que o node está instalado, abra um terminal e execute:

```
node -v
```

Se tudo estiver ocorrido conforme o esperado você verá um retorno similar a imagem 71:

```
[→ TCC git:(master) ✘ node -v
v10.13.0
```

Figura 16: Versão do Node

Após instalar o Node basta baixar o projeto no link: https://github.com/fernandobd42/projeto_tcc, e entrar no diretório do projeto para instalar as bibliotecas necessárias por meio do comando:

```
npm run install-dependencies
```

Após terminar a instalação das bibliotecas dos 3 projetos (E2E = testes, frontend = cliente, backend = servidor) execute o comando à seguir:

```
npm start
```

O comando acima é utilizado para colocar no ar o cliente e o servidor e abrir a interface do Cypress como na imagem 17:

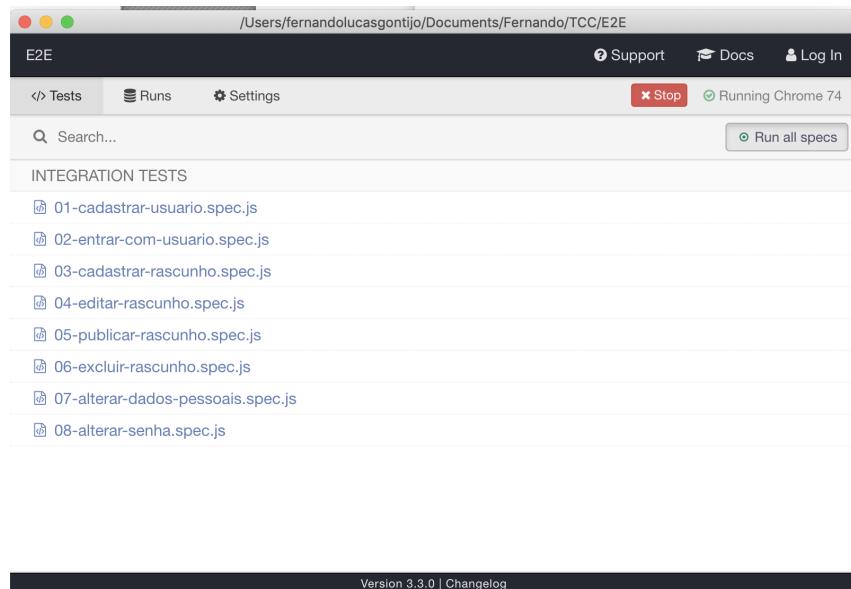


Figura 17: Interface Cypress

A seguir será descrito um conjunto de Casos de Teste com base em cada um dos Casos de Uso exibidos na figura 15 juntamente com a implementação dos testes propriamente ditos utilizando o **Cypress**.

B.3 CONSTRUÇÃO DOS CASOS DE TESTE

B.3.1 TESTE CADASTRAR USUÁRIO: DOIS CASOS DE TESTE.

Para cobrir o fluxo de Cadastrar Usuário, a primeira atividade foi abrir o arquivo `/frontend/src/components/pages/Auth/Register/index.js`, e adicionar `id` em alguns elementos para que fosse possível identificar e manipular cada um deles. A figura 18 mostra todos os `id`'s necessários para cobrir o fluxo.

```

161      <FormFields>
162        <FormikField
163          id='name'
164          required
165          name='name'
166          label='Name'
167          placeholder='Fulano de tal'
168          component={InputField}
169        />
170        <FormikField
171          id='email'
172          required
173          name='email'
174          label='Email'
175          placeholder='email@gmail.com'
176          component={InputField}
177        />
178        <FormikField
179          id='password'
180          required
181          name='password'
182          type={typePassword}
183          label='Senha'
184          placeholder='*****'
185          component={InputField}
186          endIcon={
187            <MuiTooltip title={tooltipPassword} aria-label={tooltipPassword}>
188              <MuiIconButton id='show-password' onClick={handlingTypePassword(typePassword, setTypePassword, setTooltipPassword)}>
189                <LockIconComponent type={typePassword} />
190              </MuiIconButton>
191            </MuiTooltip>
192          }
193        />
194        <FormikField
195          id='repeatPassword'
196          required
197          name='repeatPassword'
198          type={typePassword}
199          label='Confirmar senha'
200          placeholder='*****'
201          component={InputField}
202        />
203      </FormFields>
204      <CustomButton id='register' type='submit' variant='outlined' color='primary' disabled={loading} onClick={onSubmit}>
205        Cadastrar
206      </CustomButton>

```

Figura 18: Trecho de Código da Página de Cadastrar Usuário

Também foi necessário identificar a mensagem no corpo do modal de alerta para validar que a mensagem de retorno de fato será a esperada. Para identificar a mensagem foi necessário abrir o arquivo `/frontend/src/components/atoms/Alert/index.js` e adicionar o `id` como mostra a figura 19.

```

6   const Alert = (type, title, text) => (
7     Swal.fire({
8       title: title,
9       html: `<div id='text-alert'>${text}</div>`,
10      type: type,
11      timer: 3000,
12      confirmButtonText: 'Fechar',
13      confirmButtonColor: theme.palette.primary.main,
14    })
15  )

```

Figura 19: Trecho de Código do Modal de Alerta

B.3.1.1 PRIMEIRO TESTE

Foi utilizado um usuário válido com as seguintes informações: (Nome = name, E-mail = email, Senha = password), onde foi esperado um retorno de sucesso, após receber o retorno de sucesso foi verificado se de fato o usuário consta na base de dados da aplicação. Veja o Caso de Teste representado pelo Diagrama de Sequência 20 e a exemplificação do teste à seguir.

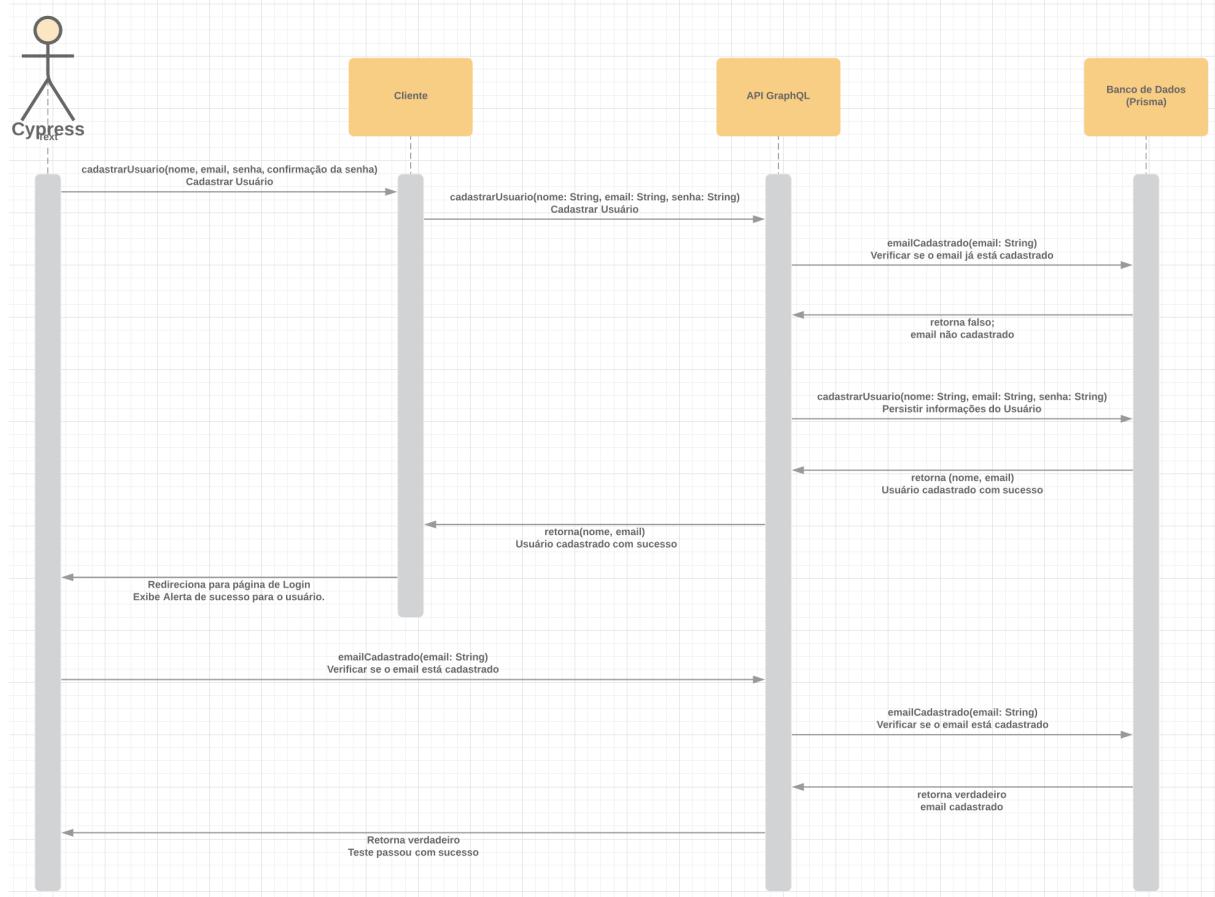


Figura 20: Diagrama de Sequência - Cadastrar Usuário com sucesso

A figura 21 representa o código necessário para implementar o teste Cadastrar

Usuário que pode ser encontrado no arquivo: /E2E/cypress/integration/01-cadastrar-usuario.spec.js.

```

1 import faker from 'faker'
2
3 import { graphql_api } from '../../../../../utils/graphql-request.service'
4
5 const emailUsed = email => `{
6   emailAlreadyUsed(email: "${email}")` 
7 `;
8
9 const name = faker.name.findName()
10 const email = faker.internet.email()
11 const password = faker.internet.password(10)
12
13 describe('Cadastrar Usuário', () => {
14   beforeEach(() => {
15     cy.visit('http://localhost:3000/auth/register')
16
17     cy.get('#name').type(name)
18     cy.get('#email').type(email)
19     cy.get('#password').type(password)
20     cy.get('#repeatPassword').type(password)
21     cy.get('#show-password').click()
22     cy.get('#register').click()
23   })
24
25   it('Usuário válido', async () => {
26     cy.get('#text-alert').should('contain', 'Cadastro realizado com sucesso!')
27     await cy.wait(5000)
28
29     const { emailAlreadyUsed } = await graphql_api(emailUsed(email))
30     expect(emailAlreadyUsed).to.be.true
31   })
}

```

Figura 21: Código de Teste Cypress - Cadastrar Usuário com sucesso

Entrando em mais detalhes do primeiro teste de Cadastrar Usuário, com base na figura 21:

- A linha 1 importa a biblioteca faker;
- A linha 3 importa a função que faz requisições direto para a API;
- Da linha 5 à 7 foi construído uma função que verifica se o **email** passado por parâmetro consta na base de dados da aplicação e retorna um valor booleano (verdadeiro = true, falso = false);
- A linha 9, foi declarado uma constante com um Nome válido utilizando a biblioteca faker, denominada **name**;

- A linha 10, foi declarado uma constante com um E-mail válido utilizando a biblioteca faker, denominada **email**;
- A linha 11, foi declarado uma constante com um Senha válido utilizando a biblioteca faker, denominada **password**;
- A linha 13, foi declarado a função **describe**, a qual é utilizada para descrever o nome que representa de forma geral o conjunto de testes implementados, neste caso, **Cadastrar Usuário**;
- Da linha 14 à 23 foi declarado a função **beforeEach**, a qual é utilizada para definir uma rotina para ser executada antes de cada caso de teste **it** . Neste caso foi definido que:
 - 1.Na linha 15 por meio do método **cy.visit** foi definido qual URL (Localizador Uniforme de Recursos), deve ser acessado;
 - 2.Na linha 17 por meio do método **cy.get** foi definido que o elemento com **seletor: #name** deve ser capturado, e com o método **.type** foi definido que o valor da constante **name** deve ser atribuído ao elemento capturado;
 - 3.Na linha 18 por meio do método **cy.get** foi definido que o elemento com **seletor: #email** deve ser capturado, e com o método **.type** foi definido que o valor da constante **email** deve ser atribuído ao elemento capturado;
 - 4.Na linha 19 por meio do método **cy.get** foi definido que o elemento com **seletor: #password** deve ser capturado, e com o método **.type** foi definido que o valor da constante **password** deve ser atribuído ao elemento capturado;
 - 5.Na linha 20 por meio do método **cy.get** foi definido que o elemento com **seletor: #repeatPassword** deve ser capturado, e com o método **.type** foi definido que o valor da constante **password** deve ser atribuído ao elemento capturado;
 - 6.Na linha 21 por meio do método **cy.get** foi definido que o elemento com **seletor: #show-password** deve ser capturado, e com o método **.click()** foi definido que deve realizar a ação de clique no elemento capturado, neste caso, será exibido as senhas para o usuário;
 - 7.Na linha 22 por meio do método **cy.get** foi definido que o elemento com **seletor: #register** deve ser capturado, e com o método **.click()** foi definido que deve realizar a ação de clique no elemento capturado, neste caso, a função para realizar o cadastro do usuário será invocada;

- Na linha 25 foi definido o primeiro caso de teste por meio da função **it**, denominada **Usuário válido**, tal teste consistiu em:

- 1.Na linha 26 por meio do método **cy.get**, foi definido que o elemento com **seletor: #text-alert** deve ser capturado e com o método **.should** foi validado que a mensagem do alerta deve conter o seguinte texto: "**Cadastro realizado com sucesso!**", uma vez que o usuário é válido pelo fato de ainda não estar cadastrado na aplicação;
- 2.Na linha 27 foi utilizado o método **cy.wait(5000)**, foi definido para esperar 5 segundos antes de prosseguir com a próxima ação;
- 3.Na linha 29 foi enviado uma requisição direta para a API, passando o **email** cadastrado por parâmetro e esperado um valor booleano como retorno na constante **emailAlreadyUsed**;
- 4.Na linha 30 foi realizado a segunda validação, onde a constante **emailAlreadyUsed** deve ser **true**, no caso verdadeiro, uma vez que o **email** verificado foi cadastrado anteriormente e já esta sendo usado;

A figura 22 representa a captura da interface do Cypress, que mostra o resultado do primeiro teste executado, onde foi realizado duas validações.

The screenshot shows the Cypress Test Runner interface with the following details:

- Test Name:** Cadastrar Usuário
- Status:** ✓ Usuário válido
- Test Steps (before each):**
 - VISIT http://localhost:3000/auth/register
 - GET #name (XHR) - GET 200 /sockjs-node/info?t=1556406466250
 - TYPE Dr. Stacey Runolfsdottir
 - GET #email
 - TYPE Garth1@yahoo.com
 - GET #password
 - TYPE xRVLUzeQLY
 - GET #repeatPassword
 - TYPE xRVLUzeQLY
 - GET #show-password
 - CLICK
 - GET #register
 - CLICK
- Test Step (highlighted):**
 - GET #text-alert (NEW URL) http://localhost:3000/auth/login
 - ASSERT expected <div#text-alert> to contain Cadastro realizado com sucesso!
- Test Step (highlighted):**
 - WAIT 5000
 - ASSERT expected true to be true
- Test Status:** ✓ Usuário inválido

Figura 22: Teste Cypress - Cadastrar Usuário com sucesso

B.3.1.2 SEGUNDO TESTE

Foi utilizado as mesmas informações do usuário cadastrado anteriormente, onde foi esperado um retorno de falha, uma vez que um usuário com aquele mesmo (E-mail = email) já está cadastrado na plataforma. Veja o Caso de Teste representado pelo Diagrama de Sequência 23.

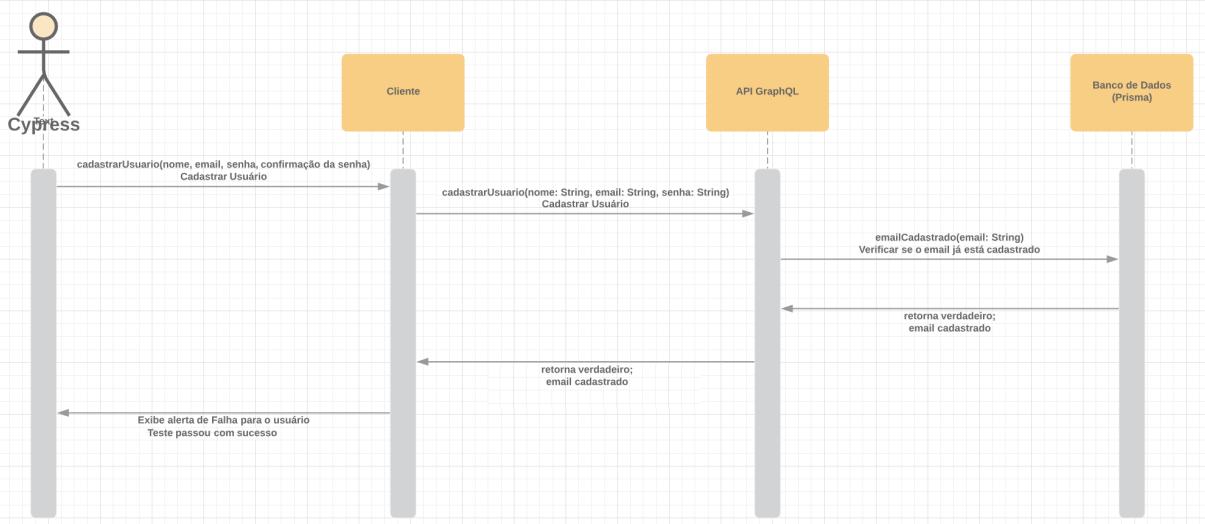


Figura 23: Diagrama de Sequência - Cadastrar Usuário com falha

Entrando em mais detalhes do segundo teste de Cadastrar Usuário, com base na figura 24:

```

33  it('Usuário inválido', () => {
34    cy.get('#text-alert').should('contain', 'Este email já está sendo usado.')
35    cy.wait(5000)
36  })
  
```

Figura 24: Código de Teste Cypress - Cadastrar Usuário com falha

- Na linha 33 foi definido o segundo caso de teste por meio da função **it**, denominada **Usuário inválido**, tal teste consistiu em:

- 1.Na linha 34 por meio do método **cy.get**, foi definido que o elemento com **seletor: #text-alert** deve ser capturado e com o método **.should** foi validado que a mensagem do alerta deve conter o seguinte texto: à "**Este email já esta sendo usado.**", uma vez que o usuário já esta cadastrado na aplicação;
- 2.Na linha 35 foi utilizado o método **cy.wait(5000)**, para esperar 5 segundos antes de prosseguir com a próxima ação;

A figura 25 representa a captura da interface do Cypress, que mostra o resultado do segundo teste executado, onde foi realizado uma validação.

The screenshot shows the Cypress Test Runner interface. The test file is named 'Cadastrar Usuário'. It contains two sections: 'Usuário válido' (successful) and 'Usuário inválido' (invalid). The 'Usuário inválido' section is expanded, showing a 'BEFORE EACH' block and a 'TEST' block. The 'TEST' block contains three steps: 1. GET #text-alert, 2. - ASSERT expected <div#text-alert> to contain Este email já está sendo usado., and 3. WAIT 5000. The second step is highlighted with an orange border, indicating it failed. The assertion message 'Este email já está sendo usado.' is displayed in green text.

```
1 VISIT http://localhost:3000/auth/register
2 GET #name
(XHR) ● GET 200 /sockjs-node/info?t=1556406472625
3 -TYPE Dr. Stacey Runolfsdottir
4 GET #email
5 -TYPE Garth1@yahoo.com
6 GET #password
7 -TYPE xRVLUzeQLY
8 GET #repeatPassword
9 -TYPE xRVLUzeQLY
10 GET #show-password
11 -CLICK
12 GET #register
13 -CLICK

1 GET #text-alert
2 - ASSERT expected <div#text-alert> to contain Este email já está sendo usado.
3 WAIT 5000
```

Figura 25: Teste Cypress - Cadastrar Usuário com falha

B.3.2 TESTE ENTRAR COM USUÁRIO: TRÊS CASOS DE TESTE.

Para cobrir o fluxo de Entrar com Usuário, a primeira atividade foi abrir o arquivo `/frontend/src/components/pages/Auth/Login/index.js`, e adicionar `id` em alguns elementos para que fosse possível identificar e manipular cada um deles por meio das rotinas do Cypress. A figura 26 mostra todos os `id's` necessários para cobrir o fluxo.

```

160 |           <FormFields>
161 |             <FormikField
162 |               id='email'
163 |               required
164 |               name='email'
165 |               label='Email'
166 |               placeholder='email@gmail.com'
167 |               component={InputField}
168 |
169 |             >
170 |             <FormikField
171 |               id='password'
172 |               required
173 |               name='password'
174 |               type={typePassword}
175 |               label='Senha'
176 |               placeholder='*****'
177 |               component={InputField}
178 |               endIcon={(
179 |                 <MuiTooltip title={tooltipPassword} aria-label={tooltipPassword}>
180 |                   <MuiIconButton id='show-password' onClick={() => handlingTypePassword(typePassword, setTypePassword, setTooltipPassword)}>
181 |                     <LockIconComponent type={typePassword} />
182 |                   </MuiIconButton>
183 |                 </MuiTooltip>
184 |               )
185 |             }
186 |           </FormFields>
187 |           <CustomButton id='login' type='submit' variant='outlined' color='primary' disabled={loading} onClick={onSubmit}>
188 |             Entrar
189 |           </CustomButton>

```

Figura 26: Trecho de Código da Página de Entrar com Usuário

B.3.2.1 PRIMEIRO E SEGUNDO TESTE

Foi utilizado um usuário válido com as seguintes informações: (E-mail = email e Senha = password), onde foi esperado um retorno de sucesso, após receber o retorno de sucesso foi verificado se de fato o usuário consta na base de dados da aplicação. Veja o Caso de Teste representado pelo Diagrama de Sequência 27 e a exemplificação do teste à seguir.

A figura 28 representa o código necessário para implementar o teste Entrar com Usuário que pode ser encontrado em: `/E2E/cypress/integration/02-entrar-com-usuario.spec.js`.

Entrando em mais detalhes do primeiro teste de Entrar com Usuário, com base na figura 28:

- A linha 1 importa a biblioteca faker;

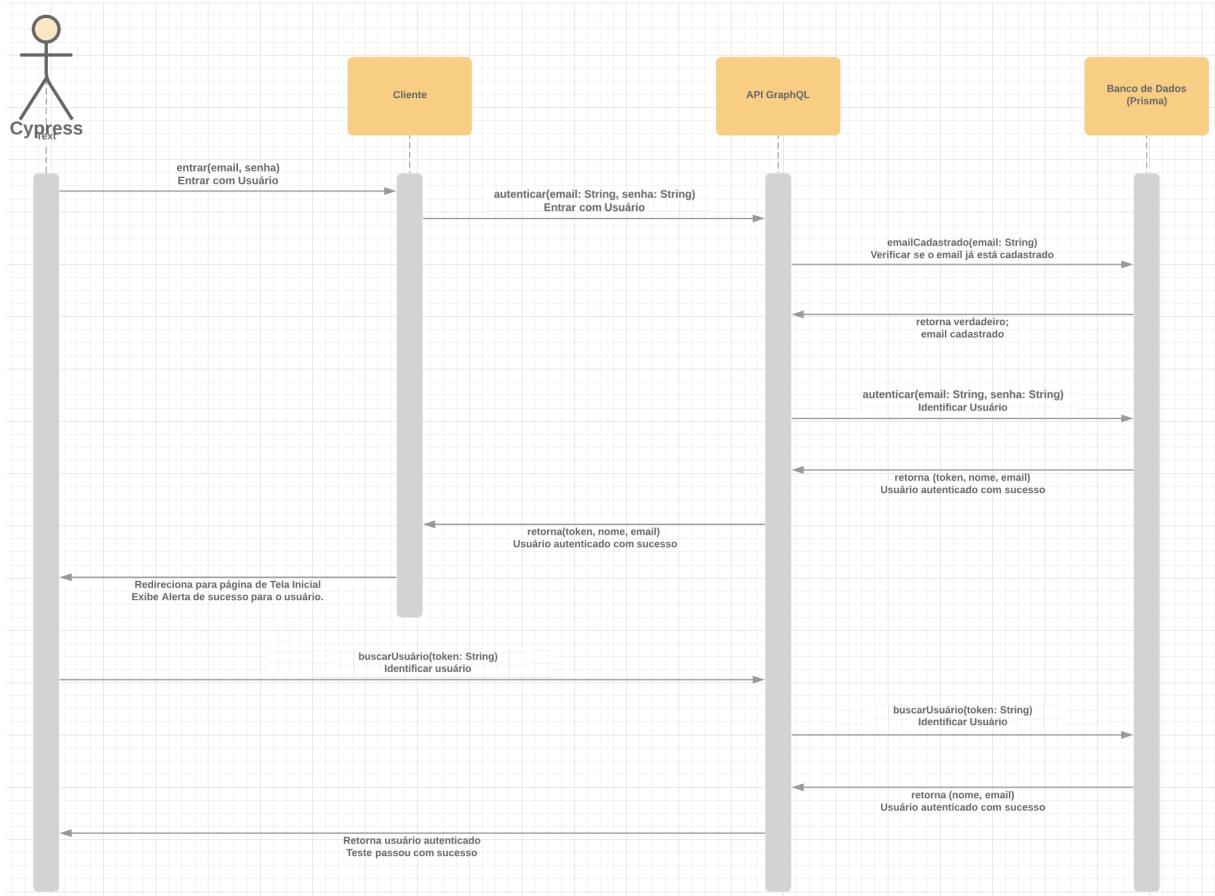


Figura 27: Diagrama de Sequência - Entrar com Usuário com sucesso

- A linha 3 importa a função que faz requisições direto para a API;
- Da linha 5 à 7 foi construído uma função que verifica se o usuário (**email** e **password**) passado por parâmetro consta na base de dados da aplicação;
- A linha 9, foi declarado uma constante com um E-mail válido, denominada **email**;
- A linha 10, foi declarado uma constante com um E-mail inválido utilizando a biblioteca faker, denominada **emailInvalid**;
- A linha 11, foi declarado uma constante com a senha válida do E-mail válido, denominada **password**;
- A linha 12, foi declarado uma constante com uma senha utilizando a biblioteca faker, denominada **passwordInvalid**;
- A linha 14, foi declarado a função **describe**, a qual é utilizada para descrever o nome que representa de forma geral o conjunto de testes implementados à seguir;

```

1 import faker from 'faker'
2
3 import { graphql_api } from '../../../../../utils/graphql-request.service'
4
5 const getToken = (email, password) => `{
6   token(email: "${email}", password: "${password}")
7 }
8
9 const email = 'fernando@gmail.com'
10 const emailInvalid = faker.internet.email()
11 const password = '%fernando%123'
12 const passwordInvalid = faker.internet.password(10)
13
14 describe('Entrar com Usuário', () => {
15   beforeEach(() => {
16     cy.visit('http://localhost:3000/auth/login')
17   })
18
19   it('Usuário válido', () => {
20     cy.get('#email').type(email)
21     cy.get('#password').type(password)
22     cy.get('#show-password').click()
23     cy.get('#login').click()
24
25     cy.get('#text-alert').should('contain', 'Login realizado com sucesso!')
26     cy.location("pathname").should("eq", "/admin/feed")
27     cy.wait(5000)
28   })
29
30   it('Validar usuário', async () => {
31     const { token } = await graphql_api(getToken(email, password))
32     expect(!token).to.have.true
33     expect(token.length).to.be.greaterThan(154)
34   })
}

```

Figura 28: Código de Teste Cypress - Entrar com Usuário com sucesso

- Da linha 15 à 17 foi declarado a função **beforeEach**, a qual é utilizada para definir uma rotina que será executada antes de cada caso teste **it**. Neste caso foi definido que:

- 1.Na linha 16 por meio do método **cy.visit** foi definido qual URL (Localizador Uniforme de Recursos), deve ser acessado;;

- Na linha 19 foi definido o primeiro caso teste por meio da função **it**, denominada **Usuário válido**, tal teste consistiu em:

- 1.Na linha 20 por meio do método **cy.get** foi definido que o elemento com **seletor**:

#email deve ser capturado, e com o método **.type** foi definido que o valor **email** deve ser atribuído ao elemento capturado;

2.Na linha 21 por meio do método **cy.get** foi definido que o elemento com **seletor:**

#password deve ser capturado, e com o método **.type** foi definido que o valor **password** deve ser atribuído ao elemento capturado;

3.Na linha 22 por meio do método **cy.get** foi definido que o elemento com **seletor:**

#show-password deve ser capturado, e com o método **.click()** foi definido que deve realizar a ação de clique no elemento capturado, neste caso, será exibido a senha para o usuário;

4.Na linha 23 por meio do método **cy.get** foi definido que o elemento com **seletor:**

#login deve ser capturado, e com o método **.click()** foi definido que deve realizar a ação de clique no elemento capturado, neste caso, a função para realizar o login do usuário será invocada;

5.Na linha 25 por meio do método **cy.get**, foi definido que o elemento com **seletor:** **#text-alert** deve ser capturado e com o método **.should** foi validado

que a mensagem do alerta deve conter o seguinte texto: "**Login realizado com sucesso!**", uma vez que o usuário é válido pelo fato de já estar cadastrado na aplicação;

6.Na linha 26 foi utilizado o método **cy.location("pathname")** para capturar

e validar se o caminho da URLpós entrar na plataforma contém o caminho:

"/admin/feed";

7.Na linha 27 foi utilizado o método **cy.wait(5000)** foi definido para esperar 5

segundos antes de prosseguir com a próxima ação;

Entrando em mais detalhes do segundo teste de Entrar com Usuário, com base na figura 28:

- Na linha 30 foi definido o segundo caso teste por meio da função **it**, denominada **Validar usuário**, tal teste consistiu em:

- 1.Na linha 31 foi enviado uma requisição direta para a API, passando o usuário válido **email** e **password** por parâmetro e esperado um **token** em formato de string como retorno;

- 2.Na linha 32 foi realizado a primeira validação, onde o **token** foi validado utilizando a dupla negação da string, uma vez que ao negar duas vezes uma string com mais de 1 caractere à transforma em booleano verdadeiro;

3.Na linha 33 foi realizado a segunda validação, onde foi validado se a string do **token** contém mais que 154 caracteres, uma vez que um token JWT (JSON Web Token) válido contém 155 caracteres;

A figura 29 representa a captura da interface do Cypress, que mostra o resultado do primeiro e do segundo teste executado, onde foi realizado quatro validações.

```

    ▼ Entrar com Usuário
      ✓ Usuário válido
        ▼ BEFORE EACH
          1 VISIT http://localhost:3000/auth/login
        ▼ TEST
          1 GET #email
            (XHR) ● GET 200 /sockjs-node/info?t=1556406684017
          2 -TYPE fernando@gmail.com
          3 GET #password
          4 -TYPE %fernando%123
          5 GET #show-password
          6 -CLICK
          7 GET #login
          8 -CLICK
          9 GET #text-alert
          10 -ASSERT expected <div#text-alert> to contain Login realizado com sucesso!
            (NEW URL) http://localhost:3000/admin
            (NEW URL) http://localhost:3000/admin/feed
          11 LOCATION pathname
          12 -ASSERT expected /admin/feed to equal /admin/feed
          13 WAIT 5000

      ✓ Validar usuário
        ▼ BEFORE EACH
          1 VISIT http://localhost:3000/auth/login
        ▼ TEST
          (XHR) ● GET 200 /sockjs-node/info?t=1556406692337
          1 ASSERT expected true to be true
          2 ASSERT expected 155 to be above 154

      ✓ Usuário inválido
  
```

Figura 29: Teste Cypress - Entrar com Usuário com sucesso

B.3.2.2 TERCEIRO TESTE

Foi utilizado informações de um usuário inválido, onde foi esperado um retorno de falha, uma vez que não existe um usuário com aquele mesmo (E-mail = email e Senha = password) cadastrado na plataforma. Veja o Caso de Teste representado pelo Diagrama de Sequência 30.

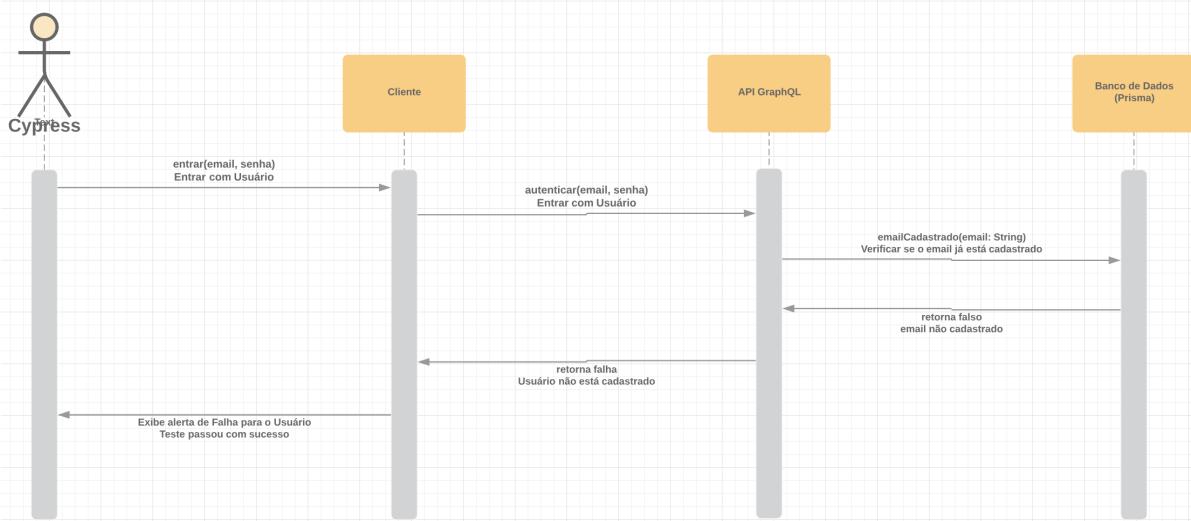


Figura 30: Diagrama de Sequência - Entrar com Usuário com falha

Entrando em mais detalhes do terceiro teste de Entrar com Usuário, com base na figura 31:

```

36   it('Usuário inválido', () => {
37     cy.get('#email').type(emailInvalid)
38     cy.get('#password').type(passwordInvalid)
39     cy.get('#show-password').click()
40     cy.get('#login').click()
41
42     cy.get('#text-alert').should('contain', 'Email não cadastrado!')
43     cy.wait(5000)
44
45     cy.get('#email').clear()
46     cy.get('#email').type(email)
47     cy.get('#login').click()
48
49     cy.get('#text-alert').should('contain', 'Senha incorreta, tente novamente com outra senha.')
50     cy.wait(5000)
51   })
52 }
  
```

Figura 31: Código de Teste Cypress - Entrar com Usuário com falha

- Na linha 36 foi definido o terceiro caso teste por meio da função **it**, denominada **Usuário inválido**, tal teste consistiu em:

- 1.Na linha 37 por meio do método **cy.get** foi definido que o elemento com **seletor: #email** deve ser capturado, e com o método **.type** foi definido que o valor **emailInvalido** inválido deve ser atribuído ao elemento capturado;
- 2.Na linha 38 por meio do método **cy.get** foi definido que o elemento com **seletor: #password** deve ser capturado, e com o método **.type** foi definido que o valor **passwordInvalido** inválido deve ser atribuído ao elemento capturado;

- 3.Na linha 39 por meio do método **cy.get** foi definido que o elemento com **seletor: #show-password** deve ser capturado, e com o método **.click()** foi definido que deve realizar a ação de clique no elemento capturado, neste caso, será exibido a senha para o usuário;
- 4.Na linha 40 por meio do método **cy.get** foi definido que o elemento com **seletor: #login** deve ser capturado, e com o método **.click()** foi definido que deve realizar a ação de clique no elemento capturado, neste caso, a função para realizar o login do usuário será invocada;
- 5.Na linha 42 por meio do método **cy.get** foi definido que o elemento com **seletor: #text-alert** deve ser capturado e com o método **.should** foi validado que a mensagem do alerta deve conter o seguinte texto: "**Email não cadastrado!**", uma vez que não existe nenhum usuário com este E-mail cadastrado na aplicação;
- 6.Na linha 43 foi utilizado o método **cy.wait(5000)** foi definido para esperar 5 segundos antes de prosseguir com a próxima ação;
- 7.Na linha 45 por meio do método **cy.get** foi definido que o elemento com **seletor: #email** deve ser capturado, e com o método **.clear** foi definido que deve limpar/apagar o valor do campo em questão;
- 8.Na linha 46 por meio do método **cy.get** foi definido que o elemento com **seletor: #email** deve ser capturado, e com o método **.type** foi definido que o valor **email** inválido deve ser atribuído ao elemento capturado;
- 9.Na linha 47 por meio do método **cy.get** foi definido que o elemento com **seletor: #login** deve ser capturado, e com o método **.click()** foi definido que deve realizar a ação de clique no elemento capturado, neste caso, a função para realizar o login do usuário será invocada;
- 10.Na linha 49 por meio do método **cy.get** foi definido que o elemento com **seletor: #text-alert** deve ser capturado e com o método **.should** foi validado que a mensagem do alerta deve conter o seguinte texto: "**Senha incorreta, tente novamente com outra senha.**", uma vez que a senha daquele E-mail não é compatível com a senha cadastrada na aplicação;
- 11.Na linha 50 foi utilizado o método **cy.wait(5000)** foi definido para esperar 5 segundos antes de prosseguir com a próxima ação;

A figura 32 representa captura da interface do Cypress, que mostra o resultado do segundo teste executado, onde foi realizado duas validações.

```
▼ Entrar com Usuário
  ✓ Usuário válido
  ✓ Validar usuário
  ✓ Usuário inválido
    ▼ BEFORE EACH
      1 VISIT http://localhost:3000/auth/login
    ▼ TEST
      1 GET #email
      (XHR) ● GET 200 /sockjs-node/info?t=1556406693489
      2 - TYPE Zakary.Muller@yahoo.com
      3 GET #password
      4 - TYPE BK_DKw3600
      5 GET #show-password
      6 - CLICK
      7 GET #login
      8 - CLICK
      9 GET #text-alert
      10 - ASSERT expected <div#text-alert> to contain Email não cadastrado!
      11 WAIT 5000
      12 GET #email
      13 - CLEAR
      14 GET #email
      15 - TYPE fernando@gmail.com
      16 GET #login
      17 - CLICK
      18 GET #text-alert
      19 - ASSERT expected <div#text-alert> to contain Senha incorreta, tente novamente com outra senha.
      20 WAIT 5000
```

Figura 32: Teste Cypress - Entrar com Usuário com falha

B.3.3 TESTE DE CADASTRAR RASCUNHO: DOIS CASOS DE TESTE.

Para cobrir o fluxo de cadastrar rascunho, a primeira atividade foi abrir o arquivo `/frontend/src/components/organisms/AppBar/index.js`, e adicionar `id` no elemento como na figura 33 para acessar a página de rascunhos.

```
52     <CustomButton id='publications' onClick={redirectToPublications} disabled={route === 'publications'}>
53       <ButtonText variant='subtitle1'>
54         Publicações
55       </ButtonText>
56     </CustomButton>
```

Figura 33: Trecho de Código Componente Menu de Navegação

Em seguida foi preciso abrir o arquivo `/frontend/src/components/organisms/TabContent/index.js`, e adicionar `id` no elemento como na figura 34 para acessar a página de cadastrar rascunho.

```
87     <MuiButton id='new-draft' variant='contained' color='primary' onClick={redirectToNewDraft}>Novo</MuiButton>
```

Figura 34: Trecho de Código da Página de Listar Rascunhos

Por fim foi preciso abrir o arquivo `/frontend/src/components/pages/Admin/NewEditItem/index.js`, e adicionar `id` em alguns elementos para que fosse possível identificar e manipular cada um deles. As figuras 33, 34 e 35 mostra todos os `id`'s necessários para cobrir o fluxo.

```
130
131   <Form>
132     <FormFields>
133       <FormikField
134         id='title'
135           required
136           name='title'
137           label='Título'
138           placeholder='Título da Publicação'
139           component={InputField}
140         />
141       <FormikField
142         id='content'
143           required
144           name='content'
145           label='Conteúdo'
146           component={InputField}
147           multiline={true}
148           rows={5}
149           rowsMax={20}
150         />
151     </FormFields>
152     <CustomButton id='save-draft' type='submit' variant='outlined' color='primary' onClick={onSubmit}>
153       {id.length > 25 ? 'Editar' : 'Adicionar'}
154     </CustomButton>
155     <Footer>
156       <CustomButton type='button' color='primary' onClick={redirectToPublications}>
157         Voltar
158       </CustomButton>
159     </Footer>
</Form>
```

Figura 35: Trecho de Código da Página de Cadastrar Rascunho

B.3.3.1 PRIMEIRO E SEGUNDO TESTE

Foi utilizado um rascunho com as seguintes informações: (Título = title e Conteúdo = content), onde foi esperado um retorno de sucesso, logo em seguida foi realizado uma outra tentativa de inserir outro rascunho com o mesmo Título onde foi esperado um retorno de falha. Após receber o retorno da segunda validação, foi verificado se de fato o rascunho com aquele determinado Título consta na base de dados da aplicação. Veja o Caso de Teste representado pelo Diagrama de Sequência 36 e a exemplificação do teste à seguir.

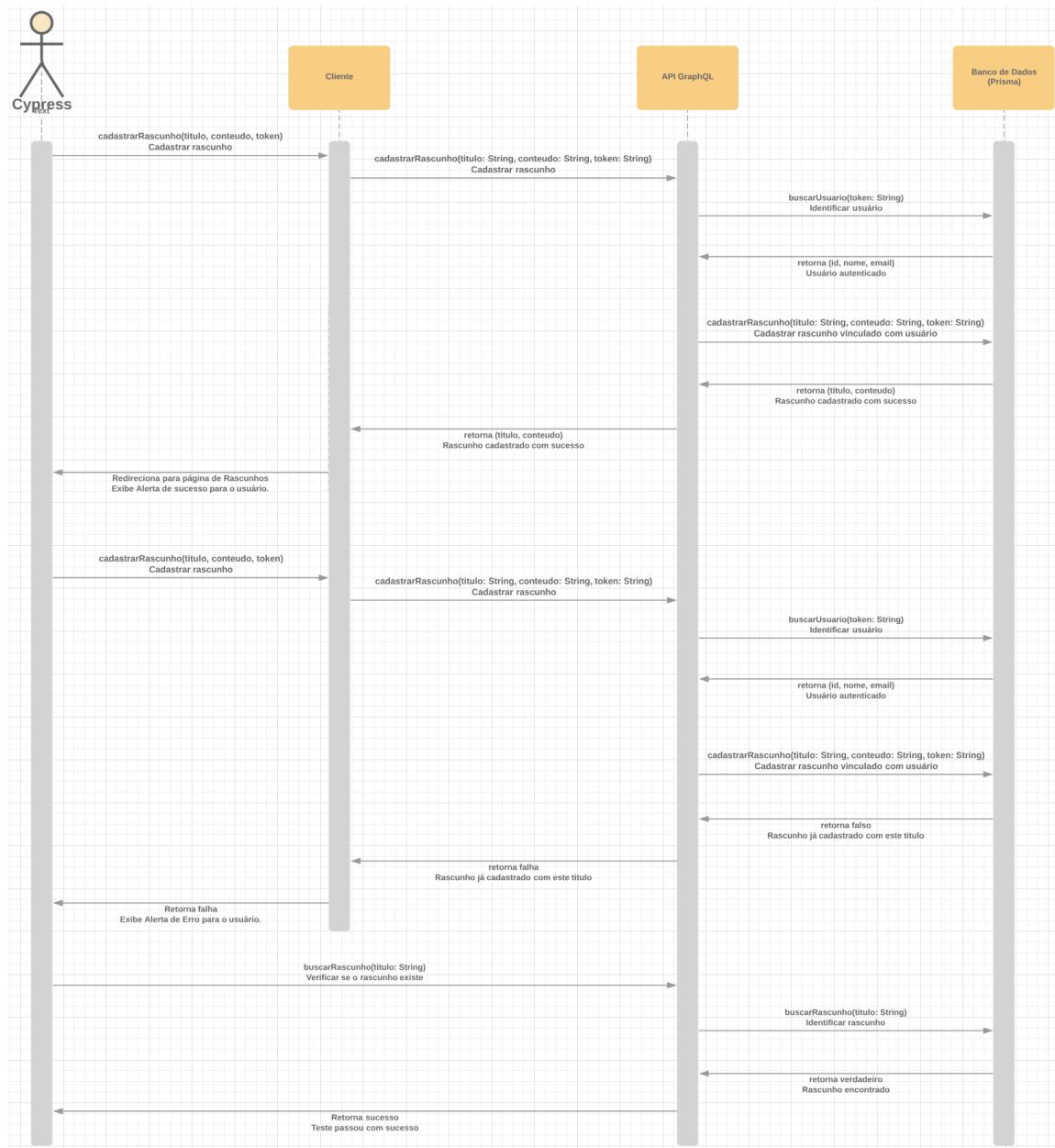


Figura 36: Diagrama de Sequência - Cadastrar Rascunho

A figura 37 representa a abstração do método Entrar com Usuário, implementada para auxiliar nos próximos testes, que pode ser encontrado em: [/E2E/cypress/support/commands.js](#).

```

1  Cypress.Commands.add('login', (email = 'fernando@gmail.com', password = "%fernando%123") => {
2    cy.visit('http://localhost:3000/auth/login')
3    cy.get('#email').type(email)
4    cy.get('#password').type(password)
5    cy.get('#show-password').click()
6    cy.get('#login').click()
7    cy.wait(5000)
8  })

```

Figura 37: Código de Comando do Cypress - Entrar com Usuário

Entrando em mais detalhes do método login com base na figura 37:

- A linha 1 por meio do método **Comamands** foi definido uma função global, onde o primeiro parâmetro no caso o **login** é o nome utilizado para invocar a função e o segundo parâmetro são os parâmetros de fato passados para a função no momento da invocação, neste caso temos 2 parâmetros com valores padrão, o (E-mail = email e a senha = password);
- Na linha 2 por meio do método **cy.visit** foi definido qual URL deve ser acessado;
- Na linha 3 por meio do método **cy.get** foi definido que o elemento com **seletor: #email** deve ser capturado, e com o método **.type** foi definido que o valor **email** deve ser atribuído ao elemento capturado;
- Na linha 4 por meio do método **cy.get** foi definido que o elemento com **seletor: #password** deve ser capturado, e com o método **.type** foi definido que o valor **password** deve ser atribuído ao elemento capturado;
- Na linha 5 por meio do método **cy.get** foi definido que o elemento com **seletor: #show-password** deve ser capturado, e com o método **.click()** foi definido que deve realizar a ação de clique no elemento capturado, neste caso, será exibido a senha para o usuário;
- Na linha 6 por meio do método **cy.get** foi definido que o elemento com **seletor: #login** deve ser capturado, e com o método **.click()** foi definido que deve realizar a ação de clique no elemento capturado, neste caso, a função para realizar o login do usuário será invocada;

- Na linha 7 foi utilizado o método `cy.wait(5000)` foi definido para esperar 5 segundos antes de prosseguir com a próxima ação;

A figura 38 representa o código necessário para implementar o teste Cadastrar Rascunho que pode ser encontrado em: [/E2E/cypress/integration/03-cadastrar-rascunho.spec.js](#).

```

1 import faker from 'faker'
2
3 import { graphql_api } from '../../../../../utils/graphql-request.service'
4
5 const titleAlreadyExists = title => `
6   titleExists(title: "${title}")
7 `
8
9 const title = faker.lorem.words(4)
10 const content = faker.lorem.paragraphs(2)
11
12 describe('Cadastrar rascunho', () => {
13   before(() => {
14     cy.login()
15     cy.get('#publications').click()
16     cy.wait(3000)
17   })
18
19   it('Novo rascunho', () => {
20     cy.get('#new-draft').click()
21     cy.location("pathname").should("eq", "/admin/newDraft")
22
23     cy.get('#title').type(title)
24     cy.get('#content').type(content)
25     cy.get('#save-draft').click()
26
27     cy.get('#text-alert').should('contain', 'Rascunho adicionado com sucesso!')
28     cy.wait(3000)
29
30     cy.get('#new-draft').click()
31
32     cy.get('#title').type(title)
33     cy.get('#content').type(content)
34     cy.get('#save-draft').click()
35
36     cy.get('#text-alert').should('contain', 'Já existe um rascunho com este título, tente outro.')
37     cy.wait(3000)
38   })
39
40   it('Validar rascunho', async () => {
41     const { titleExists } = await graphql_api(titleAlreadyExists(title))
42     expect(titleExists).to.be.true
43   })
44 })

```

Figura 38: Código de Teste Cypress - Cadastrar Rascunho

Entrando em mais detalhes do primeiro teste de Cadastrar Rascunho, com base na figura 38:

- A linha 1 importa a biblioteca faker;

- A linha 3 importa a função que faz requisições direto para a API;
- Da linha 5 à 7 foi construído uma função que verifica se o **title** do rascunho passado por parâmetro consta na base de dados da aplicação;
- A linha 9, foi declarado uma constante com um Título válido utilizando a biblioteca faker, denominada **title**;
- A linha 10, foi declarado uma constante com um Conteúdo válido utilizando a biblioteca faker, denominada **content**;
- A linha 12, foi declarado a função **describe**, a qual é utilizada para descrever o nome que representa de forma geral o conjunto de testes implementados à seguir;
- Da linha 13 à 17 foi declarado a função **before**, a qual é utilizada para definir uma rotina que será executada antes dos casos de testes **it**. Neste caso foi definido que:
 - 1.Na linha 14 por meio do método **cy.login**, foi invocada a função exemplificada na figura 37, a qual é utilizada para acessar a plataforma;
 - 2.Na linha 15 por meio do método **cy.get** foi definido que o elemento com **seletor: #publications** deve ser capturado, e com o método **.click()** foi definido que deve realizar a ação de clique no elemento capturado, neste caso, a função para realizar o redirecionamento para a página Publicações será invocada;
 - 3.Na linha 16 foi utilizado o método **cy.wait(3000)** foi definido para esperar 3 segundos antes de prosseguir com a próxima ação;
- Na linha 19 foi definido o primeiro caso de teste por meio da função **it**, denominada **Novo rascunho**, tal teste consistiu em:
 - 1.Na linha 20 por meio do método **cy.get** foi definido que o elemento com **seletor: #new-draft** deve ser capturado, e com o método **.click()** foi definido que deve realizar a ação de clique no elemento capturado, neste caso, a função para realizar o redirecionamento de página Novo Rascunho será invocada;
 - 2.Na linha 21 foi utilizado o método **cy.location("pathname")** para capturar e validar se o caminho da URLpós navegar para página de Novo Rascunho contém o caminho: **"/admin/newDraft"**;
 - 3.Na linha 23 por meio do método **cy.get** foi definido que o elemento com **seletor: #title** deve ser capturado, e com o método **.type** foi definido que o valor **title** deve ser atribuído ao elemento capturado;

- 4.Na linha 24 por meio do método **cy.get** foi definido que o elemento com **seletor: #content** deve ser capturado, e com o método **.type** foi definido que o valor **content** deve ser atribuído ao elemento capturado;
- 5.Na linha 25 por meio do método **cy.get** foi definido que o elemento com **seletor: #save-draft** deve ser capturado, e com o método **.click()** foi definido que deve realizar a ação de clique no elemento capturado, neste caso, a função para realizar o cadastro do rascunho será invocada;
- 6.Na linha 27 por meio do método **cy.get**, foi definido que o elemento com **seletor: #text-alert** deve ser capturado e com o método **.should** foi válido que a mensagem do alerta deve conter o seguinte texto: "**Rascunho adicionado com sucesso!**", uma vez que o rascunho é válido pelo fato de não existir outro com o mesmo título cadastrado na aplicação;
- 7.Na linha 28 foi utilizado o método **cy.wait(3000)** foi definido para esperar 3 segundos antes de prosseguir com a próxima ação;
- 8.Na linha 30 por meio do método **cy.get** foi definido que o elemento com **seletor: #new-draft** deve ser capturado, e com o método **.click()** foi definido que deve realizar a ação de clique no elemento capturado, neste caso, a função para realizar o redirecionamento de página Novo Rascunho será invocada;
- 9.Na linha 32 por meio do método **cy.get** foi definido que o elemento com **seletor: #title** deve ser capturado, e com o método **.type** foi definido que o valor **title** deve ser atribuído ao elemento capturado;
- 10.Na linha 33 por meio do método **cy.get** foi definido que o elemento com **seletor: #content** deve ser capturado, e com o método **.type** foi definido que o valor **content** deve ser atribuído ao elemento capturado;
- 11.Na linha 34 por meio do método **cy.get** foi definido que o elemento com **seletor: #save-draft** deve ser capturado, e com o método **.click()** foi definido que deve realizar a ação de clique no elemento capturado, neste caso, a função para realizar o cadastro do rascunho será invocada;
- 12.Na linha 36 por meio do método **cy.get**, foi definido que o elemento com **seletor: #text-alert** deve ser capturado e com o método **.should** foi válido que a mensagem do alerta deve conter o seguinte texto: "**Já existe um rascunho com este título, tente outro.**", uma vez que o rascunho é inválido pelo fato de já existir outro com o mesmo título cadastrado na aplicação;

13.Na linha 37 foi utilizado o método **cy.wait(3000)** foi definido para esperar 3 segundos antes de prosseguir com a próxima ação;

Entrando em mais detalhes do segundo teste de Cadastrar Rascunho, com base na figura 38:

- Na linha 40 foi definido o segundo caso de teste por meio da função **it**, denominada **Validar rascunho**, tal teste consistiu em:

- 1.Na linha 41 foi enviado uma requisição direta para a API, passando o (Título) do rascunho cadastrado por parâmetro e esperado um **titleExists** em formato de booleano como retorno;
- 2.Na linha 42 foi realizado a validação, onde o **titleExists** deve ser verdadeiro, uma vez que deve existir um rascunho com este título cadastrado na aplicação;

A figura 39 representa a captura da interface do Cypress, que mostra o resultado do primeiro e do segundo teste executado, onde foi realizado quatro validações.

```

    ▼ Cadastrar rascunho
      ✓ Novo rascunho
        ▶ BEFORE ALL ...
        ▶ TEST
          1 GET      #new-draft
          2 - CLICK
          (NEW URL) http://localhost:3000/admin/newDraft
          3 LOCATION pathname
          4 - ASSERT expected /admin/newDraft to equal /admin/newDraft
          5 GET      #title
          6 - TYPE   fuga repudiandae est ipsa
          7 GET      #content
          8 - TYPE   Nesciunt eos provident tempore. Magni rem repellendus dolorem aut exercitation...
          9 GET      #save-draft
          10 - CLICK
          11 GET      #text-alert
          12 - ASSERT expected <div#text-alert> to contain Rascunho adicionado com sucesso!
          (NEW URL) http://localhost:3000/admin/publications
          13 WAIT    3000
          14 GET      #new-draft
          15 - CLICK
          (NEW URL) http://localhost:3000/admin/newDraft
          16 GET      #title
          17 - TYPE   fuga repudiandae est ipsa
          18 GET      #content
          19 - TYPE   Nesciunt eos provident tempore. Magni rem repellendus dolorem aut exercitation...
          20 GET      #save-draft
          21 - CLICK
          22 GET      #text-alert
          23 - ASSERT expected <div#text-alert> to contain Já existe um rascunho com este título, tente outro.
          24 WAIT    3000

      ✓ Validar rascunho
        ▶ TEST
          1 ASSERT expected true to be true
    
```

Figura 39: Teste Cypress - Cadastrar Rascunho

B.3.4 TESTE DE EDITAR RASCUNHO: DOIS CASOS DE TESTE.

Para cobrir o fluxo de editar rascunho, a primeira atividade foi abrir o arquivo `/frontend/src/components/molecules/Table/TableBody/index.js`, e adicionar `id` nos elementos como nas figura 40 e 41 para acessar página de editar rascunho.

```
132 <CustomTableRow key={item.allObject.id} id='table-row'>
```

Figura 40: Trecho de Código Componente Tabela

```
163 <CustomButton
164   size='small'
165   id={!item.published ? 'editar-rascunho' : 'editar-publicao'}
166   disabled={loadingPublish || loadingDeletePost}
167   btnColor={theme.palette.primary.main}
168   right={7}
169   onClick={redirectToEditItem(item.allObject.id)}
170 >
171   <MuiIconEdit fontSize='small' />
172   Editar
173 </CustomButton>
```

Figura 41: Trecho de Código Componente Tabela

Neste caso não foi preciso abrir o arquivo `/frontend/src/components/pages/Admin/NewEditItem/index.js`, e adicionar `id` no elementos, uma vez que no teste anterior adicionamos todos os `id`'s necessários para cobrir o fluxo como na figura 35.

B.3.4.1 PRIMEIRO E SEGUNDO TESTE

Foi utilizado o primeiro rascunho da lista de rascunhos, onde após alterar o (Título = title) foi esperado um retorno de sucesso. Em seguida foi verificado se de fato o rascunho com o Título alterado consta na base de dados da aplicação. Veja o Caso de Teste representado pelo Diagrama de Sequência 42 e a exemplificação do teste escrit à seguir.

A figura 43 representa o código necessário para implementar o teste Editar Rascunho que pode ser encontrado em: `/E2E/cypress/integration/04-editar-rascunho.spec.js`.

Entrando em mais detalhes do primeiro teste de Editar Rascunho, com base na figura 43:

- A linha 1 importa a função que faz requisições direto para a API;

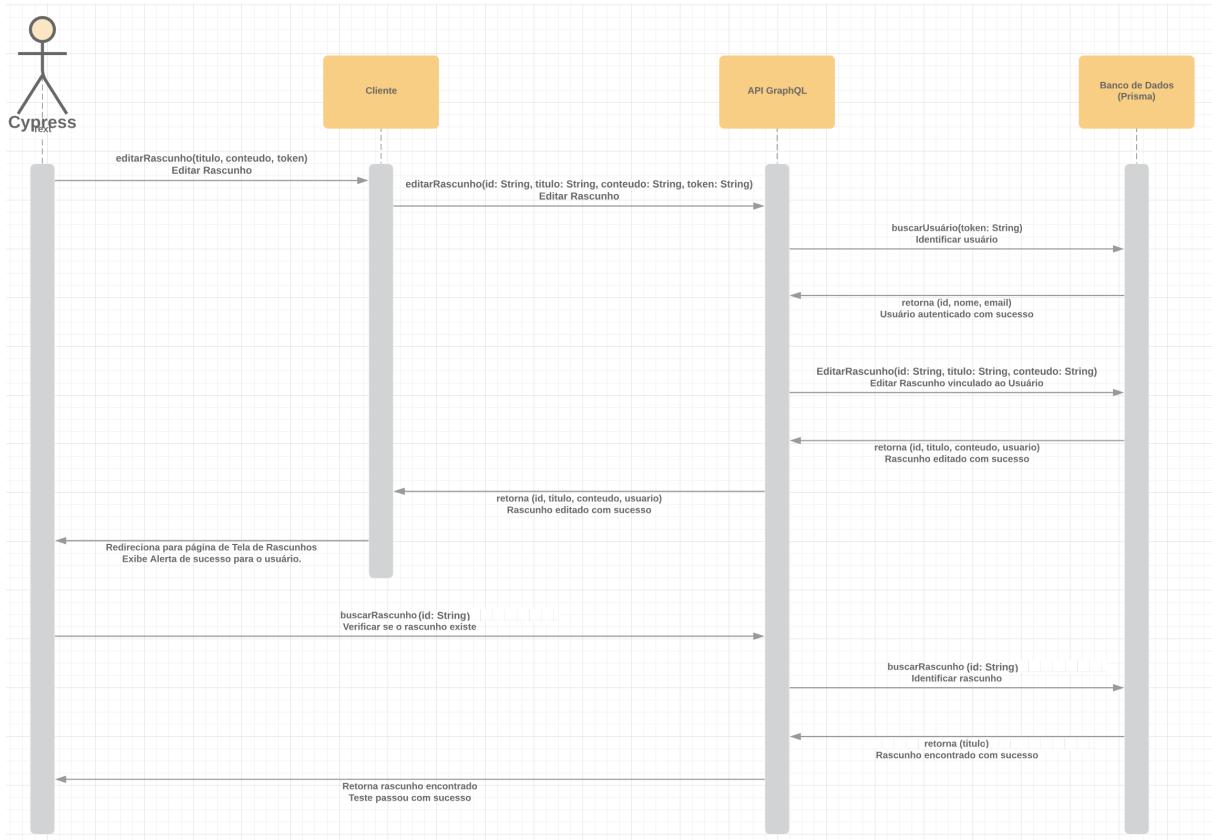


Figura 42: Diagrama de Sequência - Editar Rascunho

- Da linha 3 à 5 foi construído uma função que busca na base de dados da aplicação o Título do rascunho passando por parâmetro o **id**;
- A linha 7, foi declarado duas variáveis que posteriormente serão utilizadas para armazenar o novo **titleEdited** e o **id** do rascunho editado;
- A linha 9, foi declarado a função **describe**, a qual é utilizada para descrever o nome que representa de forma geral o conjunto de testes implementados à seguir;
- Da linha 10 à 15 foi declarado a função **before**, a qual é utilizada para definir uma rotina que será executada antes dos casos de testes **it**. Neste caso foi definido que:
 - 1.Na linha 11 por meio do método **cy.login**, foi invocada a função exemplificada na figura 37, a qual é utilizada para acessar a plataforma;
 - 2.Na linha 12 por meio do método **cy.get** foi definido que o elemento com seletor: **#publications** deve ser capturado, e com o método **.click()** foi definido que deve realizar a ação de clique no elemento capturado, neste caso, a função para realizar o redirecionamento para a página Publicações será invocada;

```

1 import { graphql_api } from '../../../../../utils/graphql-request.service'
2
3 const getDraft = id => `{
4   titleDraft(id: "${id}")
5 }
6
7 let titleEdited, id
8
9 describe('Editar rascunho', () => {
10   before(() => {
11     cy.login()
12     cy.get('#publications').click()
13     cy.wait(3000)
14     cy.get('#table-row:first-child #editar-rascunho').click()
15   })
16
17   it('Editando rascunho', () => {
18     cy.location("pathname").should("contain", "/admin/editItem")
19
20     cy.get('#title').type(' editado')
21     cy.get('input[name="title"]')
22       .invoke('val')
23       .then(text => titleEdited = text)
24     cy.window()
25       .then(win => id = win.location.href.split('/').pop())
26
27     cy.get('#save-draft').click()
28     cy.get('#text-alert').should('contain', 'Rascunho editado com sucesso!')
29     cy.wait(3000)
30   })
31
32   it('Validar rascunho', async () => {
33     const { titleDraft } = await graphql_api(getDraft(id))
34     expect(titleDraft).to.be.equal(titleEdited)
35   })
36 })

```

Figura 43: Código de Teste Cypress - Editar Rascunho

- 3.Na linha 13 foi utilizado o método **cy.wait(3000)** foi definido para esperar 3 segundos antes de prosseguir com a próxima validação;
- 4.Na linha 14 por meio do método **cy.get** foi definido que o elemento com seletor: **#table-row:first-child #editar-rascunho** deve ser capturado, e com o método **.click()** foi definido que deve realizar a ação de clique no elemento capturado, neste caso, a função para realizar o redirecionamento de página Editar Rascunho será invocada;

- Na linha 17 foi definido o primeiro caso de teste por meio da função **it**, denominada **Editando rascunho**, tal teste consistiu em:

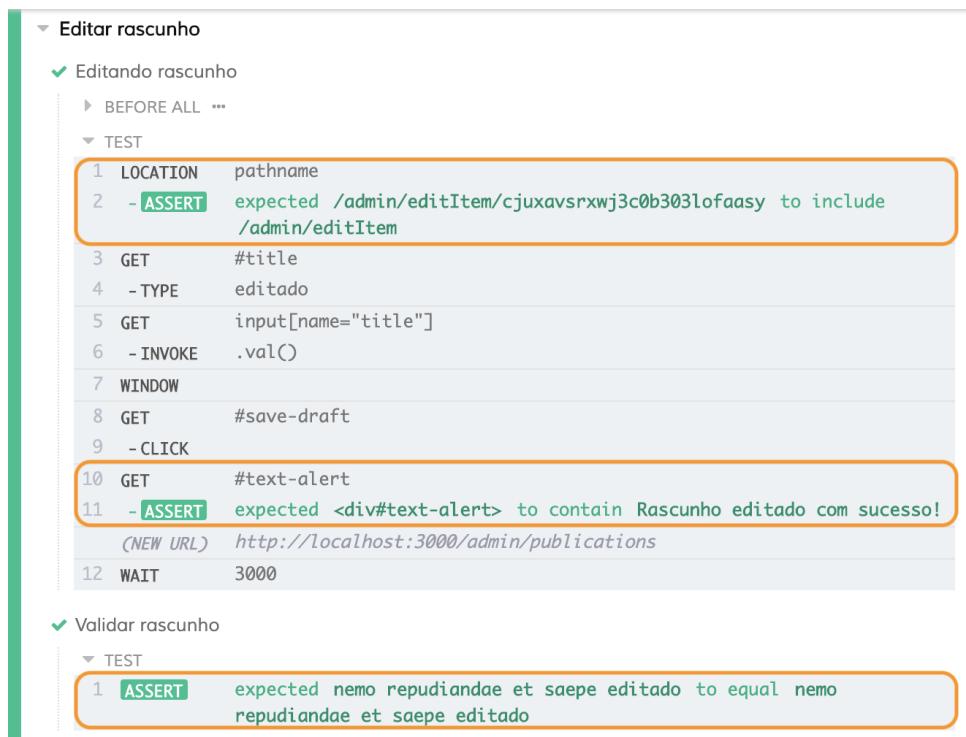
- 1.Na linha 18 foi utilizado o método **cy.location("pathname")** para capturar e validar se o caminho da URLpós navegar para página de Editar Rascunho contém o caminho: **"/admin/editItem"**;
- 2.Na linha 20 por meio do método **cy.get** foi definido que o elemento com **seletor: #title** deve ser capturado, e com o método **.type** foi definido que o valor **"editado"** deve ser inserido no elemento capturado;
- 3.Das linhas 21 à 23 por meio do método **cy.get** foi definido que o elemento com **seletor: input[name]="title"** deve ser capturado, com o método **.invoke** foi definido que o valor do campo deve ser capturado, e com o método **then** foi definido que a variável **titleEdited** vai receber o valor do campo, ou seja, o Título editado;
- 4.Nas linhas 24 e 25 por meio do método **cy.window** foi definido que a função "window" nativa do javascript será invocada, e com o método **then** foi definido que a variável **id** vai receber o último parâmetro passado na URL da página, ou seja, o ID do rascunho editado;
- 5.Na linha 27 por meio do método **cy.get** foi definido que o elemento com **seletor: #save-draft** deve ser capturado, e com o método **.click()** foi definido que deve realizar a ação de clique no elemento capturado, neste caso, a função para realizar o cadastro do rascunho será invocada;
- 6.Na linha 28 por meio do método **cy.get**, foi definido que o elemento com **seletor: #text-alert** deve ser capturado e com o método **.should** foi validado que a mensagem do alerta deve conter o seguinte texto: **"Rascunho editado com sucesso!"**, uma vez que o rascunho é válido pelo fato de não existir outro com o mesmo título cadastrado na aplicação;
- 7.Na linha 29 foi utilizado o método **cy.wait(3000)** foi definido para esperar 3 segundos antes de prosseguir com a próxima ação;

Entrando em mais detalhes do segundo teste de Editar Rascunho, com base na figura 43:

- Na linha 32 foi definido o segundo caso de teste por meio da função **it**, denominada **Validar rascunho**, tal teste consistiu em:

- 1.Na linha 33 foi enviado uma requisição direta para a API, passando o **id** do rascunho por parâmetro e esperado o **titleDraft** do rascunho como retorno;
- 2.Na linha 34 foi realizado a validação, onde o **titleDraft** deve ser igual ao **titleEdited**, uma vez que o Título foi alterado por meio da aplicação;

A figura 44 representa a captura da interface do Cypress, que mostra o resultado do primeiro e do segundo teste executado, onde foi realizado três validações.



The screenshot shows the Cypress Test Runner interface. It displays two test suites: 'Editar rascunho' and 'Validar rascunho'. The first suite contains two tests, both of which have their assertions highlighted with orange boxes. The first test's assertion is: 'expected /admin/editItem/cjuxavsrwj3c0b303lofaasy to include /admin/editItem'. The second test's assertion is: 'expected <div#text-alert> to contain Rascunho editado com sucesso!'. The second suite also contains one test with an assertion: 'expected nemo repudiandae et saepe editado to equal nemo repudiandae et saepe editado'.

```
1 LOCATION pathname
2 - ASSERT expected /admin/editItem/cjuxavsrwj3c0b303lofaasy to include /admin/editItem
3 GET #title
4 - TYPE editado
5 GET input[name="title"]
6 - INVOKE .val()
7 WINDOW
8 GET #save-draft
9 - CLICK
10 GET #text-alert
11 - ASSERT expected <div#text-alert> to contain Rascunho editado com sucesso!
(CNEW URL) http://localhost:3000/admin/publications
12 WAIT 3000

✓ Editando rascunho
  ▶ BEFORE ALL ***
  ▶ TEST
    1 LOCATION pathname
    2 - ASSERT expected /admin/editItem/cjuxavsrwj3c0b303lofaasy to include /admin/editItem
    3 GET #title
    4 - TYPE editado
    5 GET input[name="title"]
    6 - INVOKE .val()
    7 WINDOW
    8 GET #save-draft
    9 - CLICK
    10 GET #text-alert
    11 - ASSERT expected <div#text-alert> to contain Rascunho editado com sucesso!
    (CNEW URL) http://localhost:3000/admin/publications
    12 WAIT 3000

✓ Validar rascunho
  ▶ TEST
    1 ASSERT expected nemo repudiandae et saepe editado to equal nemo repudiandae et saepe editado
```

Figura 44: Teste Cypress - Editar Rascunho

B.3.5 TESTE DE PUBLICAR RASCUNHO: DOIS CASOS DE TESTE.

Para cobrir o fluxo de Publicar Rasunho, a primeira atividade foi abrir o arquivo `/frontend/src/components/molecules/Table/TableBody/index.js`, e adicionar `id` no elemento como na figura 45 para abrir o modal de publicar rascunho.

```

149   <CustomButton
150     size='small'
151     id='publicar-rascunho'
152     disabled={loadingPublish || loadingDeletePost}
153     right={12}
154     btnColor={theme.palette.success[700]}
155     onClick={() =>
156       AlertConfirm(
157         'Rascunho publicado com sucesso.',
158         publishDeleteItem(publish, item, refetch, 'publicar'),
159       )
160     }
161     >
162       Publicar
163     </CustomButton>
```

Figura 45: Trecho de Código Componente Tabela

Em seguida foi preciso abrir o arquivo `/frontend/src/components/pages/Admin/NewEditItem/index.js`, e adicionar `id` no elemento como na figura 46 para cancelar a edição do rascunho e voltar para listagem de rascunhos.

```

155   <CustomButton id='cancel' type='button' color='primary' onClick={redirectToPublications}>
156     Voltar
157   </CustomButton>
```

Figura 46: Trecho de Código da Página de Editar Rascunho

Por fim foi preciso abrir o arquivo `/frontend/src/components/atoms/AlertConfirm/index.js`, e adicionar `class` no elemento como na figura 47 para confirmar a publicação do rascunho.

```

15   customClass: {
16     confirmButton: 'confirm-button'
17   }
```

Figura 47: Trecho de Código do Modal de Decisão

B.3.5.1 PRIMEIRO E SEGUNDO TESTE

Foi utilizado um rascunho cadastrado previamente para ser publicado e foi esperado um retorno de sucesso, após receber o retorno de sucesso foi verificado se de fato o que era uma rascunho se transformou em uma publicação conforme a base de dados da aplicação. Veja o Caso de Teste representado pelo Diagrama de Sequência 48 e a exemplificação do teste à seguir.

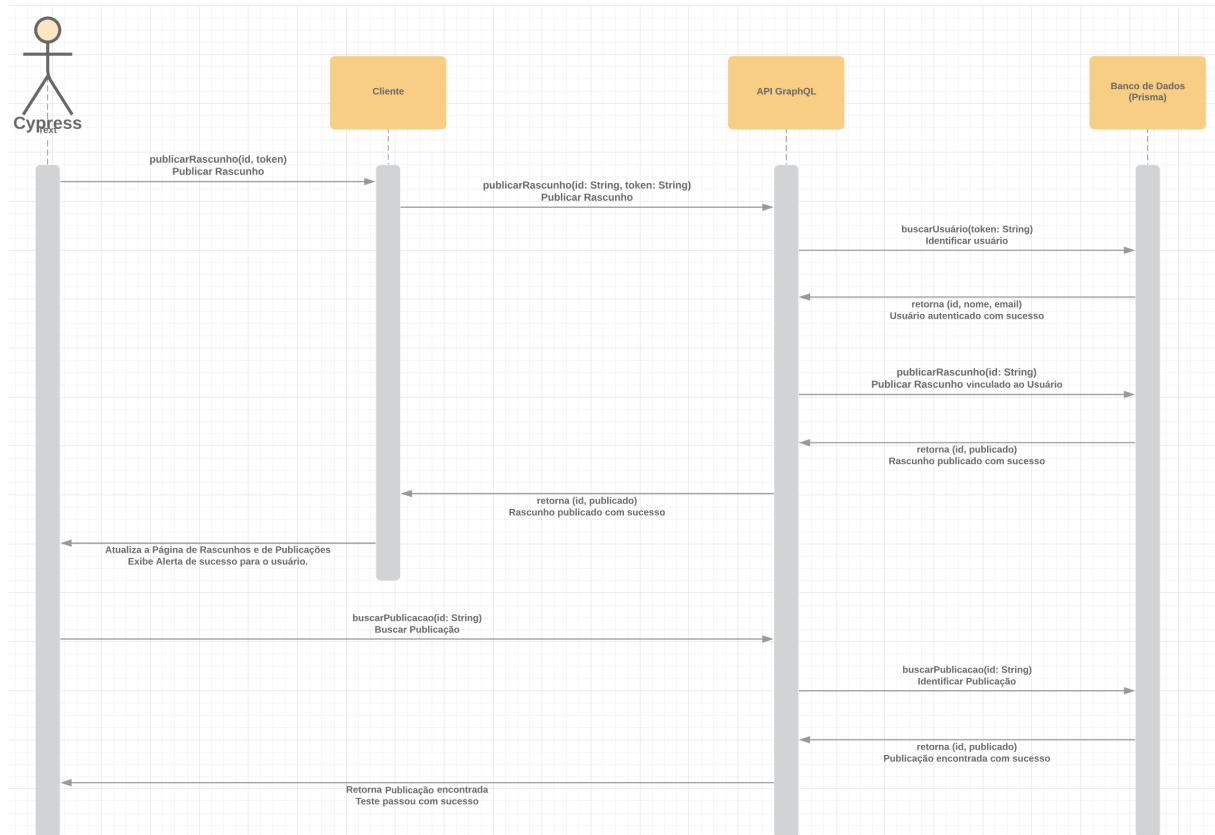


Figura 48: Diagrama de Sequência - Publicar Rascunho

A figura 49 representa o código necessário para implementar o teste Publicar Rascunho que pode ser encontrado em: [/E2E/cypress/integration/05-publicar-rascunho.spec.js](#).

Entrando em mais detalhes do primeiro teste de Publicar Rascunho, com base na figura 49:

- A linha 1 importa a função que faz requisições direto para a API;
- Da linha 3 à 7 foi construído uma função que busca na base de dados da aplicação o status da publicação passando por parâmetro o **id**, onde se espera um retorno booleano;

```

1 import { graphql_api } from '../../../../../utils/graphql-request.service'
2
3 const getPost = id => `{
4   post(id: "${id}") {
5     published
6   }
7 }` 
8
9 let id
10
11 describe('Publicar rascunho', () => {
12   before(() => {
13     cy.login()
14     cy.get('#publications').click()
15     cy.wait(3000)
16     cy.get('#table-row:first-child #editar-rascunho').click()
17     cy.window()
18       .then(win => id = win.location.href.split('/').pop())
19     cy.wait(3000)
20     cy.get('#cancel').click()
21   })
22
23   it('Publicando rascunho', () => {
24     cy.get('#table-row:first-child #publicar-rascunho').click()
25     cy.wait(3000)
26     cy.get('.confirm-button').click()
27     cy.get('#text-alert').should('contain', 'Rascunho publicado com sucesso.')
28     cy.wait(3000)
29   })
30
31   it('Validar publicação', async () => {
32     const { post } = await graphql_api(getPost(id))
33     expect(post.published).to.be.true
34   })
35 })

```

Figura 49: Código de Teste Cypress - Publicar Rascunho

- A linha 9 foi declarado uma variável que posteriormente será utilizadas para armazenar o ID do rascunho que será publicado;
- A linha 11, foi declarado a função **describe**, a qual é utilizada para descrever o nome que representa de forma geral o conjunto de testes implementados à seguir;
- Da linha 12 à 21 foi declarado a função **before**, a qual é utilizada para definir uma rotina que será executada antes dos casos de testes **it**. Neste caso foi definido que:
 - 1.Na linha 13 por meio do método **cy.login**, foi invocada a função exemplificada na figura 37, a qual é utilizada para acessar a plataforma;
 - 2.Na linha 14 por meio do método **cy.get** foi definido que o elemento com seletor: **#publications** deve ser capturado, e com o método **.click()** foi definido que

deve realizar a ação de clique no elemento capturado, neste caso, a função para realizar o redirecionamento para a página Publicações será invocada;

3.Na linha 15 foi utilizado o método **cy.wait(3000)** foi definido para esperar 3 segundos antes de prosseguir com a próxima validação;

4.Na linha 16 por meio do método **cy.get** foi definido que o elemento com **seletor: #table-row:first-child #editar-rascunho** deve ser capturado, e com o método **.click()** foi definido que deve realizar a ação de clique no elemento capturado, neste caso, a função para realizar o redirecionamento de página Editar Rascunho será invocada;

5.Nas linhas 17 e 18 por meio do método **cy.window** foi definido que a função "window" nativa do javascript será invocada, e com o método **then** foi definido que a variável **id** vai receber o último parâmetro passado na URL da página, ou seja, o ID do rascunho que será publicado;

6.Na linha 19 foi utilizado o método **cy.wait(3000)** foi definido para esperar 3 segundos antes de prosseguir com a próxima validação;

7.Na linha 20 por meio do método **cy.get** foi definido que o elemento com **seletor: #cancel** deve ser capturado, e com o método **.click()** foi definido que deve realizar a ação de clique no elemento capturado, neste caso, a função para realizar o redirecionamento para voltar para a página Publicações será invocada;

- Na linha 23 foi definido o primeiro caso de teste por meio da função **it**, denominada **Publicando rascunho**, tal teste consistiu em:

1.Na linha 24 por meio do método **cy.get** foi definido que o elemento com **seletor: #table-row:first-child #publicar-rascunho** deve ser capturado, e com o método **.click()** foi definido que deve realizar a ação de clique no elemento capturado, neste caso, a função para abrir o modal de decisão será invocada;

2.Na linha 25 foi utilizado o método **cy.wait(3000)** foi definido para esperar 3 segundos antes de prosseguir com a próxima ação;

3.Na linha 26 por meio do método **cy.get** foi definido que o elemento com **seletor: .confirm-button** deve ser capturado, e com o método **.click()** foi definido que deve realizar a ação de clique no elemento capturado, neste caso, a função para confirmar a publicação do rascunho será invocada;

4.Na linha 27 por meio do método **cy.get**, foi definido que o elemento com **seletor: #text-alert** deve ser capturado e com o método **.should** foi válido que a mensagem do alerta deve conter o seguinte texto: "**Rascunho publicado com sucesso.**", uma vez que o rascunho é válido e pode ser publicado dentro da aplicação;

5.Na linha 28 foi utilizado o método **cy.wait(3000)** foi definido para esperar 3 segundos antes de prosseguir com a próxima ação;

Entrando em mais detalhes do segundo teste de Publicar Rascunho, com base na figura 49:

- Na linha 31 foi definido o segundo caso de teste por meio da função **it**, denominada **Validar publicação**, tal teste consistiu em:

- 1.Na linha 32 foi enviado uma requisição direta para a API, passando o **id** da publicação por parâmetro e esperado um objeto **post** como retorno;
- 2.Na linha 33 foi realizado a validação, onde o **post.published** deve ser verdadeiro, uma vez que o rascunho foi publicado na aplicação;

A figura 50 representa a captura da interface do Cypress, que mostra o resultado do primeiro e do segundo teste executado, onde foi realizado duas validações.

```

    ▼ Publicar rascunho
      ✓ Publicando rascunho
        ▶ BEFORE ALL ...
        ▼ TEST
          1 GET      #table-row:first-child #publicar-rascunho
          2 -CLICK
          3 WAIT     3000
          4 GET      .confirm-button
          5 -CLICK
          6 GET      #text-alert
          7 -ASSERT  expected <div#text-alert> to contain Rascunho publicado com
                     sucesso.
          8 WAIT     3000

      ✓ Validar publicação
        ▼ TEST
          1 ASSERT  expected true to be true
    
```

Figura 50: Teste Cypress - Publicar Rascunho

B.3.6 TESTE DE EXCLUIR PUBLICAÇÃO: DOIS CASOS DE TESTE.

Para cobrir o fluxo de Excluir Rascunho, a primeira atividade foi abrir o arquivo `/frontend/src/components/molecules/Table/TableBody/index.js`, e adicionar `id` no elemento como na figura 51 para abrir o modal de excluir rascunho.

```

174   <CustomButton
175     size='small'
176     id={!item.published ? 'excluir-rascunho' : 'excluir-publicao'}
177     disabled={loadingPublish || loadingDeletePost}
178     btncolor={theme.palette.danger[700]}
179     onClick={() =>
180       AlertConfirm(
181         'Item excluido com sucesso.',
182         publishDeleteItem(deletePost, item, refetch, 'excluir')
183       )
184     }
185     <MuiIconDelete fontSize='small' />
186     Excluir
187   </CustomButton>
```

Figura 51: Trecho de Código Componente Tabela

B.3.6.1 PRIMEIRO E SEGUNDO TESTE

Foi cadastrado um novo rascunho para posteriormente ser excluído e foi esperado um retorno de sucesso, após receber o retorno de sucesso foi verificado se de fato o rascunho foi excluído conforme a base de dados da aplicação. Veja o Caso de Teste representado pelo Diagrama de Sequência 52 e a exemplificação do teste à seguir.

A figura 53 representa o código necessário para implementar o teste de Excluir Rascunho que pode ser encontrado em: `/E2E/cypress/integration/06-excluir-rascunho.spec.js`.

Entrando em mais detalhes do primeiro teste de Excluir Rascunho, com base na figura 53:

- A linha 1 importa a biblioteca faker;
- A linha 3 importa a função que faz requisições direto para a API;
- Da linha 5 à 7 foi construído uma função que busca na base de dados da aplicação o Título do rascunho passando por parâmetro o `id`;
- A linha 9, foi declarado uma constante com um Título válido utilizando a biblioteca faker, denominada `title`;

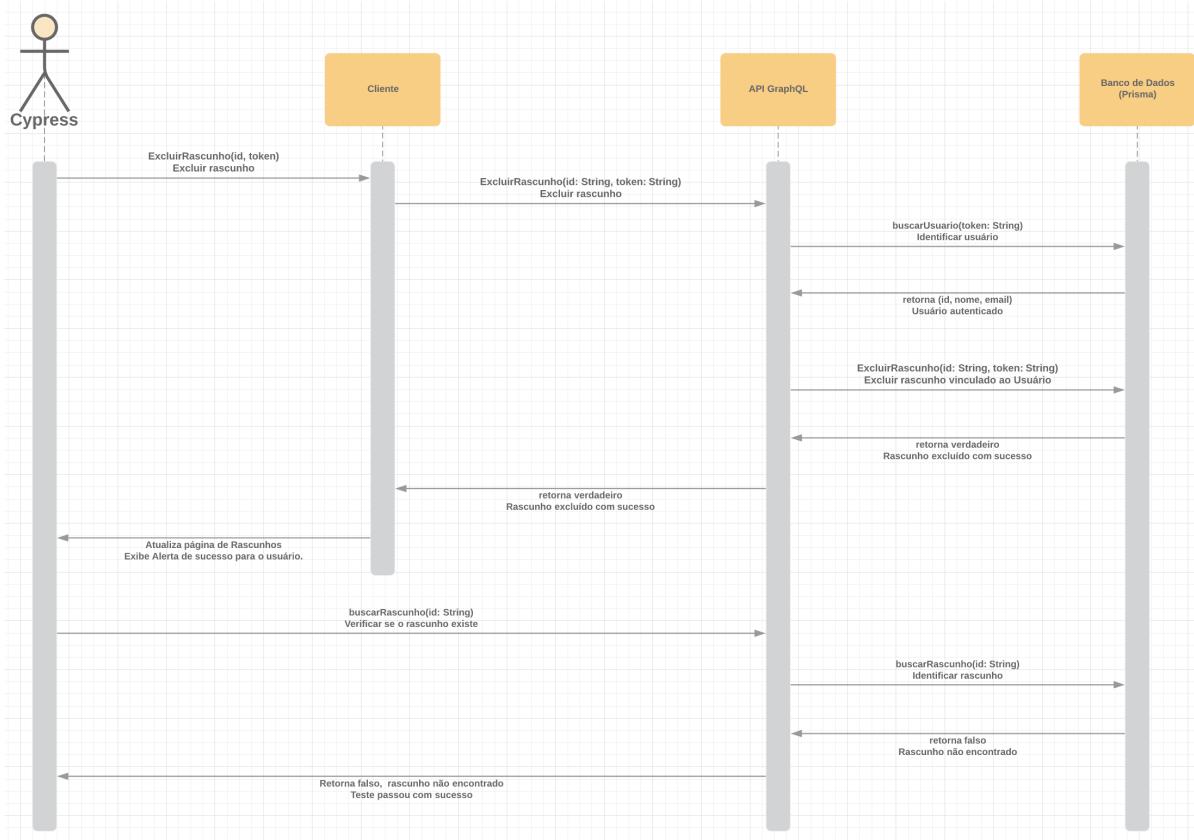


Figura 52: Diagrama de Sequência - Excluir Rascunho

- A linha 10, foi declarado uma constante com uma Conteúdo válido utilizando a biblioteca faker, denominada **content**;
- A linha 11 foi declarado uma variável que posteriormente será utilizadas para armazenar o **id** do rascunho que será publicado;
- A linha 13, foi declarado a função **describe**, a qual é utilizada para descrever o nome que representa de forma geral o conjunto de testes implementados à seguir;
- Da linha 14 à 30 foi declarado a função **before**, a qual é utilizada para definir uma rotina que será executada antes dos casos de testes **it**. Neste caso foi definido que:
 - 1.Na linha 15 por meio do método **cy.login**, foi invocada a função exemplificada na figura 37, a qual é utilizada para acessar a plataforma;
 - 2.Na linha 16 por meio do método **cy.get** foi definido que o elemento com **seletor: #publications** deve ser capturado, e com o método **.click()** foi definido que deve realizar a ação de clique no elemento capturado, neste caso, a função para realizar o redirecionamento para a página Publicações será invocada;

```

1 import faker from 'faker'
2
3 import { graphql_api } from '../../../../../utils/graphql-request.service'
4
5 const getDraft = id => `{
6   titleDraft(id: "${id}")
7 }` 
8
9 const title = faker.lorem.words(4)
10 const content = faker.lorem.paragraphs(2)
11 let id
12
13 describe('Excluir rascunho', () => {
14   before(() => {
15     cy.login()
16     cy.get('#publications').click()
17     cy.wait(3000)
18
19     cy.get('#new-draft').click()
20     cy.get('#title').type(title)
21     cy.get('#content').type(content)
22     cy.get('#save-draft').click()
23     cy.wait(3000)
24
25     cy.get('#table-row:first-child #editar-rascunho').click()
26     cy.window()
27       .then(win => id = win.location.href.split('/').pop())
28     cy.wait(3000)
29     cy.get('#cancel').click()
30   })
31
32   it('Excluindo rascunho', () => {
33     cy.get('#table-row:first-child #excluir-rascunho').click()
34     cy.wait(3000)
35     cy.get('.confirm-button').click()
36     cy.get('#text-alert').should('contain', 'Item excluído com sucesso.')
37     cy.wait(3000)
38   })
39
40   it('Validar rascunho', async () => {
41     const { titleDraft } = await graphql_api(getDraft(id))
42     expect(titleDraft).to.be.equal('Rascunho não existe')
43   })
44 })

```

Figura 53: Código de Teste Cypress - Excluir Rascunho

- 3.Na linha 17 foi utilizado o método **cy.wait(3000)** foi definido para esperar 3 segundos antes de prosseguir com a próxima ação;
- 4.Na linha 19 por meio do método **cy.get** foi definido que o elemento com **seletor: #new-draft** deve ser capturado, e com o método **.click()** foi definido que deve realizar a ação de clique no elemento capturado, neste caso, a função para realizar o redirecionamento de página Novo Rascunho será invocada;
- 5.Na linha 20 por meio do método **cy.get** foi definido que o elemento com **seletor: #title** deve ser capturado, e com o método **.type** foi definido que o valor **title** deve ser atribuído ao elemento capturado;
- 6.Na linha 21 por meio do método **cy.get** foi definido que o elemento com **seletor: #content** deve ser capturado, e com o método **.type** foi definido que o valor **content** deve ser atribuído ao elemento capturado;
- 7.Na linha 22 por meio do método **cy.get** foi definido que o elemento com **seletor: #save-draft** deve ser capturado, e com o método **.click()** foi definido que deve realizar a ação de clique no elemento capturado, neste caso, a função para realizar o cadastro do rascunho será invocada;
- 8.Na linha 23 foi utilizado o método **cy.wait(3000)** foi definido para esperar 3 segundos antes de prosseguir com a próxima ação;
- 9.Na linha 25 por meio do método **cy.get** foi definido que o elemento com **seletor: #table-row:first-child #editar-rascunho** deve ser capturado, e com o método **.click()** foi definido que deve realizar a ação de clique no elemento capturado, neste caso, a função para realizar o redirecionamento de página Editar Rascunho será invocada;
- 10.Nas linhas 26 e 27 por meio do método **cy.window** foi definido que a função "window" nativa do javascript será invocada, e com o método **then** foi definido que a variável **id** vai receber o último parâmetro passado na URL da página, ou seja, o ID do rascunho editado;
- 11.Na linha 28 foi utilizado o método **cy.wait(3000)** foi definido para esperar 3 segundos antes de prosseguir com a próxima validação;
- 12.Na linha 29 por meio do método **cy.get** foi definido que o elemento com **seletor: #cancel** deve ser capturado, e com o método **.click()** foi definido que deve realizar a ação de clique no elemento capturado, neste caso, a função para realizar o redirecionamento para voltar para a página Publicações será invocada;

- Na linha 32 foi definido o primeiro caso de teste por meio da função **it**, denominada **Excluindo rascunho**, tal teste consistiu em:

- 1.Na linha 33 por meio do método **cy.get** foi definido que o elemento com **seletor: #table-row:first-child #excluir-rascunho** deve ser capturado, e com o método **.click()** foi definido que deve realizar a ação de clique no elemento capturado, neste caso, a função para abrir o modal de decisão será invocada;
- 2.Na linha 34 foi utilizado o método **cy.wait(3000)** foi definido para esperar 3 segundos antes de prosseguir com a próxima validação;
- 3.Na linha 35 por meio do método **cy.get** foi definido que o elemento com **seletor: .confirm-button** deve ser capturado, e com o método **.click()** foi definido que deve realizar a ação de clique no elemento capturado, neste caso, a função para confirmar a exclusão do rascunho será invocada; **seletor: #text-alert** deve ser capturado e com o método **.should** foi validado que a mensagem do alerta deve conter o seguinte texto: "**Item excluído com sucesso.**", uma vez que o rascunho foi excluído da aplicação;
- 4.Na linha 28 foi utilizado o método **cy.wait(3000)** foi definido para esperar 3 segundos antes de prosseguir com a próxima ação;

Entrando em mais detalhes do segundo teste de Excluir Rascunho, com base na figura 53:

- Na linha 40 foi definido o segundo caso de teste por meio da função **it**, denominada **Validar rascunho**, tal teste consistiu em:

- 1.Na linha 41 foi enviado uma requisição direta para a API, passando o **id** do rascunho por parâmetro e esperado o **titleDraft** do rascunho como retorno;
- 2.Na linha 42 foi realizado a validação, onde o **titleDraft** deve ser igual à "**Rascunho não existe**" uma vez que o rascunho foi excluído por meio da aplicação;

A figura 54 representa a captura da interface do Cypress, que mostra o resultado do primeiro e do segundo teste executado, onde foi realizado duas validações.



```
Excluir rascunho
  ✓ Excluindo rascunho
    ▶ BEFORE ALL ...
      ▶ TEST
        1 GET      #table-row:first-child #excluir-rascunho
        2 -CLICK
        3 WAIT     3000
        4 GET      .confirm-button
        5 -CLICK
        6 GET      #text-alert
        7 -ASSERT  expected <div#text-alert> to contain Item excluído com sucesso.
        8 WAIT     3000

  ✓ Validar rascunho
    ▶ TEST
      -ASSERT  expected Rascunho não existe to equal Rascunho não existe
```

Figura 54: Teste Cypress - Excluir Rascunho

B.3.7 TESTE DE ALTERAR DADOS PESSOAIS: DOIS CONJUNTOS DE TESTE.

Para cobrir o fluxo de Alterar Dados Pessoais do usuário, a primeira atividade foi abrir o arquivo `/frontend/src/components/molecules/DropdownMenu/index.js`, e adicionar `id` nos elementos como nas figura 55 e 56 para acessar página de Configurações.

```

73   |   <MuiButton
74     |     id='dropdown'
75     |     onClick={profileToggle(openMenu, setOpenMenu)}
76     |     buttonRef={setProfileAnchorEl}
77   |   >

```

Figura 55: Trecho de Código do Componente DropdownMenu

```

91   |   <MuiMenuItem>
92     |     <MuiMenuItem id='settings' onClick={profileToggle(openMenu, setOpenMenu, redirectToSettings)}>Configurações</MuiMenuItem>
93     |     <MuiMenuItem onClick={logout(setUser, redirectToLogin)}>Sair</MuiMenuItem>
94   | </MuiMenuItemList>

```

Figura 56: Trecho de Código do Componente DropdownMenu

Em seguida foi preciso abrir o arquivo `/frontend/src/components/pages/Admin/Settings/PersonalTab/index.js`, e adicionar `id` em alguns elementos para que fosse possível identificar e manipular cada um deles. As figuras 55, 56 e 57 mostram todos os `id's` necessários para cobrir o fluxo.

```

125  |   <Form>
126    |     <FormFields>
127      |       <FormikField
128        |         id='name'
129        |         required
130        |         name='name'
131        |         label='Nome'
132        |         placeholder='Fulano de tal'
133        |         component={InputField}
134      |     >
135      |     <FormikField
136        |       id='email'
137        |       required
138        |       name='email'
139        |       label='Email'
140        |       placeholder='email@gmail.com'
141        |       component={InputField}
142      |     >
143      |     <FormikField
144        |       id='password'
145        |       required
146        |       name='password'
147        |       type={typePassword}
148        |       label='Senha'
149        |       placeholder='*****'
150        |       component={InputField}
151        |       endIcon={(
152          |           <MuiTooltip title={tooltipPassword} aria-label={tooltipPassword}>
153            |             <MuiIconButton id='show-password' onClick={handlingTypePassword(typePassword, setTypePassword, setTooltipPassword)}>
154              |               <LockIconComponent type={typePassword} />
155            |             </MuiIconButton>
156          |         </MuiTooltip>
157        |       )
158      |     >
159    |   </FormFields>
160    |   <CustomButton id='save-settings' type='submit' variant='outlined' color='primary' disabled={loading} onClick={onSubmit}>
161      |     Alterar
162    |   </CustomButton>
163  | </Form>

```

Figura 57: Trecho de Código da Página de Alterar Dados Pessoais

B.3.7.1 PRIMEIRO CONJUNTO DE TESTE

Foi utilizado uma senha inválida, para alterar as seguintes informações: (Nome = newName e E-mail = newEmail), onde foi esperado um retorno de falha, após receber o retorno de falha foi verificado se de fato a mensagem de erro no modal era a esperada, em seguida, foi utilizado a senha válida, onde foi esperado um retorno de sucesso, após receber o retorno de sucesso foi verificado se de fato as informações do usuário em questão foram alteradas conforme a base de dados da aplicação. Veja o Caso de Teste representado pelos Diagramas de sequência 58 e 59 e a exemplificação do teste à seguir.

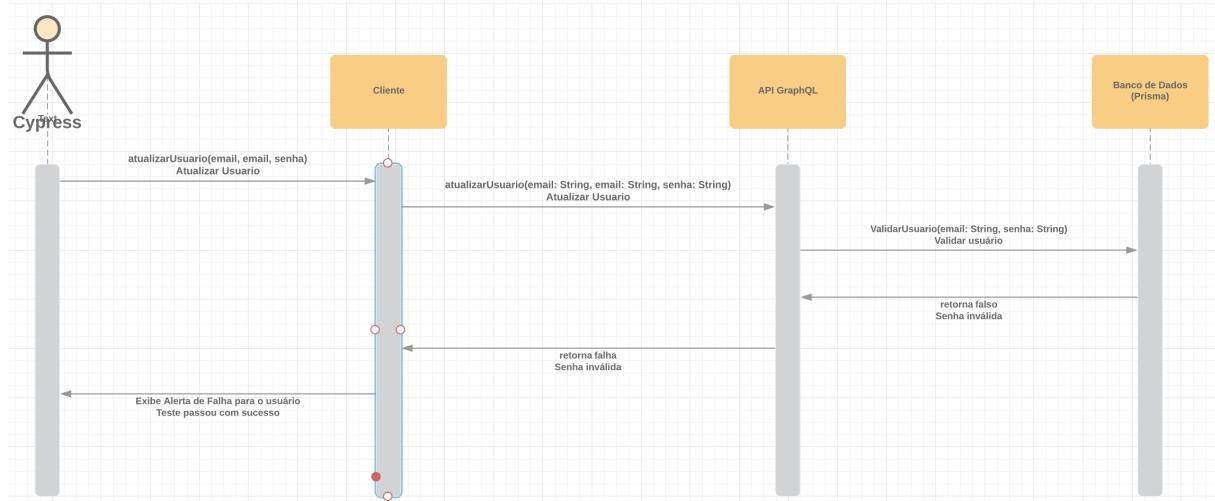


Figura 58: Diagrama de Sequência - Alterar Dados Pessoais - falha

A figura 60 representa o código necessário para implementar o teste Altera Dados Pessoais que pode ser encontrado em: [/E2E/cypress/integration/07-alterar-dados-pessoais.spec.js](#).

Entrando em mais detalhes do primeiro teste de Alterar Dados Pessoais, com base na figura 60:

- A linha 1 importa a biblioteca faker;
- A linha 3 importa a função que faz requisições direto para a API;
- Da linha 5 à 10 foi construído uma função que busca na base de dados da aplicação o Usuário passando por parâmetro o **email** e a **password**;
- A linha 12, foi declarado uma constante com um E-mail válido, denominada **email**;
- A linha 13, foi declarado uma constante com a senha válida do E-mail válido, denominada **password**;

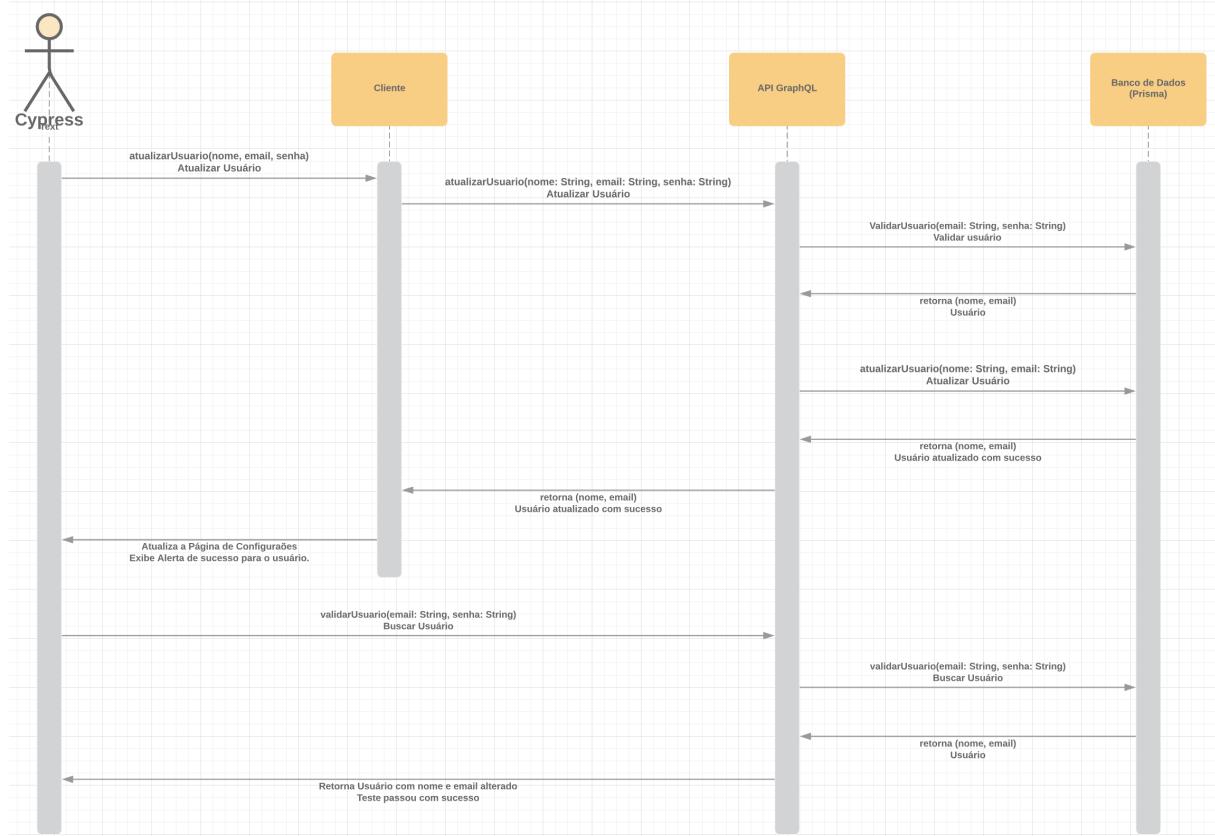


Figura 59: Diagrama de Sequência - Alterar Dados Pessoais - sucesso

- A linha 14, foi declarado uma constante com um novo E-mail utilizando a biblioteca faker, denominada **newEmail**;
- A linha 15, foi declarado uma constante com uma senha inválida utilizando a biblioteca faker, denominada **wrongPassword**;
- A linha 16, foi declarado duas variáveis que posteriormente serão utilizadas para armazenar o Nome atual e o novo Nome do usuário alterado;
- A linha 18, foi declarado a função **describe**, a qual é utilizada para descrever o nome que representa de forma geral o conjunto de testes implementados à seguir;
- Da linha 19 à 25 foi declarado a função **before**, a qual é utilizada para definir uma rotina que será executada antes dos casos de testes **it**. Neste caso foi definido que:
 - 1.Na linha 20 por meio do método **cy.login**, foi invocada a função exemplificada na figura 37, a qual é utilizada para acessar a plataforma;
 - 2.Na linha 21 por meio do método **cy.get** foi definido que o elemento com **seletor: #dropdown** deve ser capturado, e com o método **.click()** foi definido que

```

1 import faker from 'faker'
2
3 import { graphql_api } from '../../../../../utils/graphql-request.service'
4
5 const getUser = (email, password) => `{
6   user(email: "${email}", password: "${password}") {
7     name
8     email
9   }
10}`
11
12 const email = 'fernando@gmail.com'
13 const password = '%fernando%123'
14 const newEmail = faker.internet.email()
15 const wrongPassword = faker.internet.password(12)
16 let name, newName;
17
18 describe('Alterar dados pessoais', () => {
19   before(() => {
20     cy.login()
21     cy.get('#dropdown').click()
22     cy.wait(1000)
23     cy.get('#settings').click()
24     cy.wait(1000)
25   })
26
27   it('Alterar usuário', () => {
28     cy.location("pathname").should("eq", "/admin/settings")
29
30     cy.get('input[name="name"]')
31       .invoke('val')
32       .then(text => name = text)
33
34     cy.get('#name').type(' editado')
35
36     cy.get('input[name="name"]')
37       .invoke('val')
38       .then(text => newName = text)
39
40     cy.get('#email').clear()
41     cy.get('#email').type(newEmail)
42     cy.get('#password').type(wrongPassword)
43     cy.get('#show-password').click()
44     cy.get('#save-settings').click()
45
46     cy.get('#text-alert').should('contain', 'Senha incorreta, tente novamente com outra senha.')
47     cy.wait(3000)
48
49     cy.get('#password').clear()
50     cy.get('#password').type(password)
51     cy.get('#save-settings').click()
52     cy.get('#text-alert').should('contain', 'Dados pessoais alterados com sucesso!')
53     cy.wait(3000)
54   })
55
56   it('Validar alteracao', async () => {
57     try {
58       await graphql_api(getUser(email, password))
59     } catch (error) {
60       expect(error.response.errors[0].message).to.be
61         .equal(`No such user found for email: ${email}`)
62     }
63
64     const { user } = await graphql_api(getUser(newEmail, password))
65     expect(user.name).to.be.equal(newName)
66     expect(user.email).to.be.equal(newEmail)
67   })
68 })

```

Figura 60: Código de Teste Cypress - Alterar Dados Pessoais

deve realizar a ação de clique no elemento capturado, neste caso, a função para abrir o menu e exibir as opções será invocada;

3.Na linha 22 foi utilizado o método **cy.wait(1000)** foi definido para esperar 1 segundo antes de prosseguir com a próxima validação;

4.Na linha 23 por meio do método **cy.get** foi definido que o elemento com **seletor: #settings** deve ser capturado, e com o método **.click()** foi definido que deve realizar a ação de clique no elemento capturado, neste caso, a função para realizar o redirecionamento para a página Configurações será invocada;

5.Na linha 24 foi utilizado o método **cy.wait(1000)** foi definido para esperar 1 segundo antes de prosseguir com a próxima validação;

- Na linha 27 foi definido o primeiro caso de teste por meio da função **it**, denominada **Alterar usuário**, tal teste consistiu em:

1.Na linha 28 foi utilizado o método **cy.location("pathname")** para capturar e validar se o caminho da URLpós navegar para página de Editar Rascunho contém o caminho: **"/admin/settings"**;

2.Das linhas 30 à 32 por meio do método **cy.get** foi definido que o elemento com **seletor: input[name]="name"** deve ser capturado, com o método **.invoke** foi definido que o valor do campo deve ser capturado, e com o método **then** foi definido que a variável **name** vai receber o valor do campo, ou seja, o Nome atual do usuário;

3.Na linha 34 por meio do método **cy.get** foi definido que o elemento com **seletor: #name** deve ser capturado, e com o método **.type** foi definido que o valor **"editado"** deve ser inserido no elemento capturado;

4.Das linhas 36 à 38 por meio do método **cy.get** foi definido que o elemento com **seletor: input[name]="name"** deve ser capturado, com o método **.invoke** foi definido que o valor do campo deve ser capturado, e com o método **then** foi definido que a variável **newName** vai receber o valor do campo, ou seja, o novo Nome do usuário;

5.Na linha 40 por meio do método **cy.get** foi definido que o elemento com **seletor: #email** deve ser capturado, e com o método **.clear** foi definido que deve limpar/apagar o valor do campo em questão;

6.Na linha 41 por meio do método **cy.get** foi definido que o elemento com **seletor: #email** deve ser capturado, e com o método **.type** foi definido que o valor

- newEmail** inválido deve ser atribuído ao elemento capturado;
- 7.Na linha 42 por meio do método **cy.get** foi definido que o elemento com **seletor: #password** deve ser capturado, e com o método **.type** foi definido que o valor **wrongPassword** inválido deve ser atribuído ao elemento capturado;
- 8.Na linha 43 por meio do método **cy.get** foi definido que o elemento com **seletor: #show-password** deve ser capturado, e com o método **.click()** foi definido que deve realizar a ação de clique no elemento capturado, neste caso, será exibido a senha para o usuário;
- 9.Na linha 44 por meio do método **cy.get** foi definido que o elemento com **seletor: #save-settings** deve ser capturado, e com o método **.click()** foi definido que deve realizar a ação de clique no elemento capturado, neste caso, a função para salvar as informações do usuário será invocada;
- 10.Na linha 26 por meio do método **cy.get**, foi definido que o elemento com **seletor: #text-alert** deve ser capturado e com o método **.should** foi validado que a mensagem do alerta deve conter o seguinte texto: "**Senha incorreta, tente novamente com outra senha**", uma vez que a senha do usuário é inválida;
- 11.Na linha 47 foi utilizado o método **cy.wait(3000)** foi definido para esperar 3 segundos antes de prosseguir com a próxima ação;
- 12.Na linha 49 por meio do método **cy.get** foi definido que o elemento com **seletor: #password** deve ser capturado, e com o método **.clear** foi definido que deve limpar/apagar o valor do campo em questão;
- 13.Na linha 50 por meio do método **cy.get** foi definido que o elemento com **seletor: #password** deve ser capturado, e com o método **.type** foi definido que o valor **password** inválido deve ser atribuído ao elemento capturado;
- 14.Na linha 51 por meio do método **cy.get** foi definido que o elemento com **seletor: #save-settings** deve ser capturado, e com o método **.click()** foi definido que deve realizar a ação de clique no elemento capturado, neste caso, a função para salvar as informações do usuário será invocada;
- 15.Na linha 52 por meio do método **cy.get**, foi definido que o elemento com **seletor: #text-alert** deve ser capturado e com o método **.should** foi validado que a mensagem do alerta deve conter o seguinte texto: "**Dados pessoais alterados com sucesso!**", uma vez que a senha do usuário é válida;

16.Na linha 53 foi utilizado o método **cy.wait(3000)** foi definido para esperar 3 segundos antes de prosseguir com a próxima ação;

Entrando em mais detalhes do segundo teste de Alterar Dados Pessoais, com base na figura 60:

•Na linha 56 foi definido o segundo caso de teste por meio da função **it**, denominada **Validar alteração**, tal teste consistiu em:

- 1.Na linha 58 foi enviado uma requisição direta para a API, passando o (**email** e **password**) do usuário por parâmetro e esperado uma objeto de **error** como retorno;
- 2.Nas linhas 60 e 61 foi realizado a validação, onde o **error** deve ser um objeto que contém a seguinte mensagem "**No such user found for email: fernando@g mail.com**", uma vez que o E-mail do usuário foi alterado na aplicação;
- 3.Na linha 64 foi enviado uma requisição direta para a API, passando o (**newEmail** e **password**) do usuário por parâmetro e esperado uma objeto **user** como retorno;
- 4.Na linha 65 foi realizado a validação, onde o **user** deve ser um objeto do usuário e foi verificado se o atributo **user.name** é igual ao atributo **newName**, como esperado após realizar a alteração dos dados pessoais por meio da aplicação;
- 5.Na linha 66 foi realizado a validação, onde o **user** deve ser um objeto do usuário e foi verificado se o atributo **user.email** é igual ao atributo **newEmail**, como esperado após realizar a alteração dos dados pessoais por meio da aplicação;

A figura 61 representa a captura da interface do Cypress, que mostra o resultado do primeiro conjunto de teste executado, onde foi realizado seis validações.

```

▼ Alterar dados pessoais
  ✓ Alterar usuário
    ▶ BEFORE ALL ...
    ▶ TEST
      1 LOCATION pathname
      2 - ASSERT expected /admin/settings to equal /admin/settings
      3 GET input[name="name"]
      4 - INVOKE .val()
      5 GET #name
      6 - TYPE editado
      7 GET input[name="name"]
      8 - INVOKE .val()
      9 GET #email
     10 - CLEAR
     11 GET #email
     12 - TYPE Yoshiko_Boyer@hotmail.com
     13 GET #password
     14 - TYPE B1NL11beAo7I
     15 GET #show-password
     16 - CLICK
     17 GET #save-settings
     18 - CLICK
      19 GET #text-alert
      20 - ASSERT expected <div#text-alert> to contain Senha incorreta, tente novamente com outra senha.
      21 WAIT 3000
      22 GET #password
      23 - CLEAR
      24 GET #password
      25 - TYPE %fernando%123
      26 GET #save-settings
      27 - CLICK
      28 GET #text-alert
      29 - ASSERT expected <div#text-alert> to contain Dados pessoais alterados com sucesso!
      30 WAIT 3000

  ✓ Validar alteração
    ▶ TEST
      1 ASSERT expected No such user found for email: fernando@gmail.com to equal No such user found for email: fernando@gmail.com
      2 ASSERT expected Fernando Gontijo editado to equal Fernando Gontijo editado
      3 ASSERT expected Yoshiko_Boyer@hotmail.com to equal Yoshiko_Boyer@hotmail.com

  ▶ Voltar dados pessoais
    ✓ Voltar usuário
    ✓ Validar alteração

```

Figura 61: Teste Cypress - Alterar Dados Pessoais

B.3.7.2 SEGUNDO CONJUNTO DE TESTE

Foi utilizado a senha válida para alterar as seguintes informações: (Nome = name e E-mail = email), onde foi esperado um retorno de sucesso, após receber o retorno de sucesso foi verificado se de fato as informações do usuário em questão foram voltadas conforme a base de dados da aplicação. Veja o Caso de Teste representado pelo Diagrama de Sequência 59 e a exemplificação do teste à seguir.

A figura 62 representa o código necessário para implementar o teste Voltar Dados Pessoais que pode ser encontrado em: [/E2E/cypress/integration/07-alterar-dados-pessoais.spec.js](#).

```

70  describe('Voltar dados pessoais', () => {
71    before(() => {
72      cy.login()
73      cy.get('#dropdown').click()
74      cy.wait(1000)
75      cy.get('#settings').click()
76      cy.wait(1000)
77    })
78
79    it('Voltar usuário', () => {
80      cy.get('#name').clear()
81      cy.get('#name').type(name)
82      cy.get('#email').clear()
83      cy.get('#email').type(email)
84      cy.get('#password').type(password)
85      cy.get('#show-password').click()
86      cy.get('#save-settings').click()
87
88      cy.get('#text-alert').should('contain', 'Dados pessoais alterados com sucesso!')
89      cy.wait(3000)
90    })
91
92    it('Validar alteracao', async () => {
93      try {
94        await graphql_api(getUser(newEmail, password))
95      } catch (error) {
96        expect(error.response.errors[0].message).to.be
97        .equal(`No such user found for email: ${newEmail}`)
98      }
99
100     const { user } = await graphql_api(getUser(email, password))
101     expect(user.name).to.be.equal(name)
102     expect(user.email).to.be.equal(email)
103   })
104 })

```

Figura 62: Código de Teste Cypress - Voltar Dados Pessoais

Entrando em mais detalhes do primeiro teste de Voltar Dados Pessoais,

com base na figura 62:

- A linha 70, foi declarado a função **describe**, a qual é utilizada para descrever o nome que representa de forma geral o conjunto de testes implementados à seguir;
- Da linha 71 à 77 foi declarado a função **before**, a qual é utilizada para definir uma rotina que será executada antes dos casos de testes **it**. Neste caso foi definido que:
 - 1.Na linha 72 por meio do método **cy.login**, foi invocada a função exemplificada na figura 37, a qual é utilizada para acessar a plataforma;
 - 2.Na linha 73 por meio do método **cy.get** foi definido que o elemento com **seletor: #dropdown** deve ser capturado, e com o método **.click()** foi definido que deve realizar a ação de clique no elemento capturado, neste caso, a função para abrir o menu e exibir as opções será invocada;
 - 3.Na linha 74 foi utilizado o método **cy.wait(1000)** foi definido para esperar 1 segundo antes de prosseguir com a próxima validação;
 - 4.Na linha 75 por meio do método **cy.get** foi definido que o elemento com **seletor: #settings** deve ser capturado, e com o método **.click()** foi definido que deve realizar a ação de clique no elemento capturado, neste caso, a função para realizar o redirecionamento para a página Configurações será invocada;
 - 5.Na linha 76 foi utilizado o método **cy.wait(1000)** foi definido para esperar 1 segundo antes de prosseguir com a próxima validação;
- Na linha 79 foi definido o primeiro caso de teste por meio da função **it**, denominada **voltar usuário**, tal teste consistiu em:
 - 1.Na linha 80 por meio do método **cy.get** foi definido que o elemento com **seletor: #name** deve ser capturado, e com o método **.clear** foi definido que deve limpar/apagar o valor do campo em questão;
 - 2.Na linha 81 por meio do método **cy.get** foi definido que o elemento com **seletor: #name** deve ser capturado, e com o método **.type** foi definido que o valor **name** deve ser inserido no elemento capturado;
 - 3.Na linha 82 por meio do método **cy.get** foi definido que o elemento com **seletor: #email** deve ser capturado, e com o método **.clear** foi definido que deve limpar/apagar o valor do campo em questão;

- 4.Na linha 83 por meio do método **cy.get** foi definido que o elemento com **seletor: #email** deve ser capturado, e com o método **.type** foi definido que o valor **email** deve ser inserido no elemento capturado;
- 5.Na linha 84 por meio do método **cy.get** foi definido que o elemento com **seletor: #password** deve ser capturado, e com o método **.type** foi definido o que valor **password** deve ser inserido no elemento capturado;
- 6.Na linha 85 por meio do método **cy.get** foi definido que o elemento com **seletor: #show-password** deve ser capturado, e com o método **.click()** foi definido que deve realizar a ação de clique no elemento capturado, neste caso, será exibido a senha para o usuário;
- 7.Na linha 86 por meio do método **cy.get** foi definido que o elemento com **seletor: #save-settings** deve ser capturado, e com o método **.click()** foi definido que deve realizar a ação de clique no elemento capturado, neste caso, a função para salvar as informações do usuário será invocada;
- 8.Na linha 88 por meio do método **cy.get**, foi definido que o elemento com **seletor: #text-alert** deve ser capturado e com o método **.should** foi validado que a mensagem do alerta deve conter o seguinte texto: "**Dados pessoais alterados com sucesso!**", uma vez que a senha do usuário é válida;
- 9.Na linha 89 foi utilizado o método **cy.wait(3000)** foi definido para esperar 3 segundos antes de prosseguir com a próxima ação;

Entrando em mais detalhes do segundo teste de Voltar Dados Pessoais, com base na figura 61:

- Na linha 92 foi definido o segundo caso de teste por meio da função **it**, denominada **Voltar alteração**, tal teste consistiu em:
 - 1.Na linha 94 foi enviado uma requisição direta para a API, passando o (**email** e **password**) do usuário por parâmetro e esperado uma objeto de **error** como retorno;
 - 2.Nas linhas 96 e 97 foi realizado a validação, onde o **error** deve ser um objeto que contém a seguinte mensagem "**No such user found for email: novo email gerado pelo faker**", uma vez que o E-mail do usuário foi alterado na aplicação;

- 3.Na linha 100 foi enviado uma requisição direta para a API, passando o (**email** e **password**) do usuário por parâmetro e esperado uma objeto **user** como retorno;
- 4.Na linha 101 foi realizado a validação, onde o **user** deve ser um objeto do usuário e foi verificado se o atributo **user.name** é igual ao atributo **name**, como esperado após realizar a alteração dos dados pessoais por meio da aplicação;
- 5.Na linha 102 foi realizado a validação, onde o **user** deve ser um objeto do usuário e foi verificado se o atributo **user.email** é igual ao atributo **email**, como esperado após realizar a alteração dos dados pessoais por meio da aplicação;

A figura 63 representa a captura da interface do Cypress, que mostra o resultado do segundo conjunto de teste executado, onde foi realizado quatro validações.

```

▶ Alterar dados pessoais ...
  ▾ Voltar dados pessoais
    ✓ Voltar usuário
      ▶ BEFORE ALL ...
      ▾ TEST
        1 GET      #name
        2 - CLEAR
        3 GET      #name
        4 - TYPE   Fernando Gontijo
        5 GET      #email
        6 - CLEAR
        7 GET      #email
        8 - TYPE   fernando@gmail.com
        9 GET      #password
        10 - TYPE  %fernando%123
        11 GET      #show-password
        12 - CLICK
        13 GET      #save-settings
        14 - CLICK
        15 GET      #text-alert
        16 - ASSERT expected <div#text-alert> to contain Dados pessoais alterados com sucesso!
        17 WAIT    3000

    ✓ Validar alteracao
      ▾ TEST
        1 ASSERT expected No such user found for email: Yoshiko_Boyer@hotmail.com to equal No such user found for email: Yoshiko_Boyer@hotmail.com
        2 ASSERT expected Fernando Gontijo to equal Fernando Gontijo
        3 ASSERT expected fernando@gmail.com to equal fernando@gmail.com
  
```

Figura 63: Teste Cypress - Voltar Dados Pessoais

B.3.8 TESTE DE ALTERAR SENHA: DOIS CONJUNTOS DE TESTE.

Para cobrir o fluxo de Alterar Senha do usuário, a primeira atividade foi abrir o arquivo `/frontend/src/components/pages/Admin/Settings/index.js`, e adicionar `id` no elemento como na figura 64 para acessar página de alterar senha.

```

49     <ListItemIcon id='change-password' button onClick={handleTab(1, setTab)} selected={tab === 1}>
50       <ListItemIconIcon>
51         | <Lock />
52       </ListItemIconIcon>
53       <ListItemText primary='Senha' />
54     </ListItemIcon>

```

Figura 64: Trecho de Código do Componente Tabs de Configuração

Em seguida foi preciso abrir o arquivo `/frontend/src/components/pages/Admin/Settings/PersonalTab/index.js`, e adicionar `id` em alguns elementos para que fosse possível identificar e manipular cada um deles. A figura 65 mostra todos os `id's` necessários para cobrir o fluxo.

```

120   <Form>
121     <FormFields>
122       <FormikField
123         id='new-password'
124         required
125         name='newPassword'
126         type={typePassword}
127         label='Nova Senha'
128         placeholder='*****'
129         component={InputField}
130         endIcon={
131           <MuiTooltip title={tooltipPassword} aria-label={tooltipPassword}>
132             <MuiIconButton id='show-password' onClick={() => handlingTypePassword(typePassword, setTypePassword, setTooltipPassword)}>
133               <LockIconComponent type={typePassword} />
134             </MuiIconButton>
135           </MuiTooltip>
136         }
137       >
138       <FormikField
139         id='confirm-new-password'
140         required
141         name='confirmNewPassword'
142         type={typePassword}
143         label='Confirmar senha'
144         placeholder='*****'
145         component={InputField}
146       >
147       <FormikField
148         id='current-password'
149         required
150         name='currentPassword'
151         type={typePassword}
152         label='Senha atual'
153         placeholder='*****'
154         component={InputField}
155       >
156     </FormFields>
157     <CustomButton id='save-password' type='submit' variant='outlined' color='primary' disabled={loading} onClick={onSubmit}>
158       Alterar
159     </CustomButton>
160   </Form>

```

Figura 65: Trecho de Código da Página de Alterar Senha

B.3.8.1 PRIMEIRO CONJUNTO DE TESTE

Foi utilizado uma senha inválida, para alterar a seguinte informação: (Senha = password), onde foi esperado um retorno de falha, após receber o retorno de falha foi verificado se de fato a mensagem de erro no modal era a esperada, em seguida, foi utilizado a senha válida onde foi esperado um retorno de sucesso, após receber o retorno de sucesso foi verificado se de fato a Senha do usuário foi alterada conforme a base de dados da aplicação. Veja o Caso de Teste representado pelos Diagramas de sequência 66 e 67 e a exemplificação do teste à seguir.

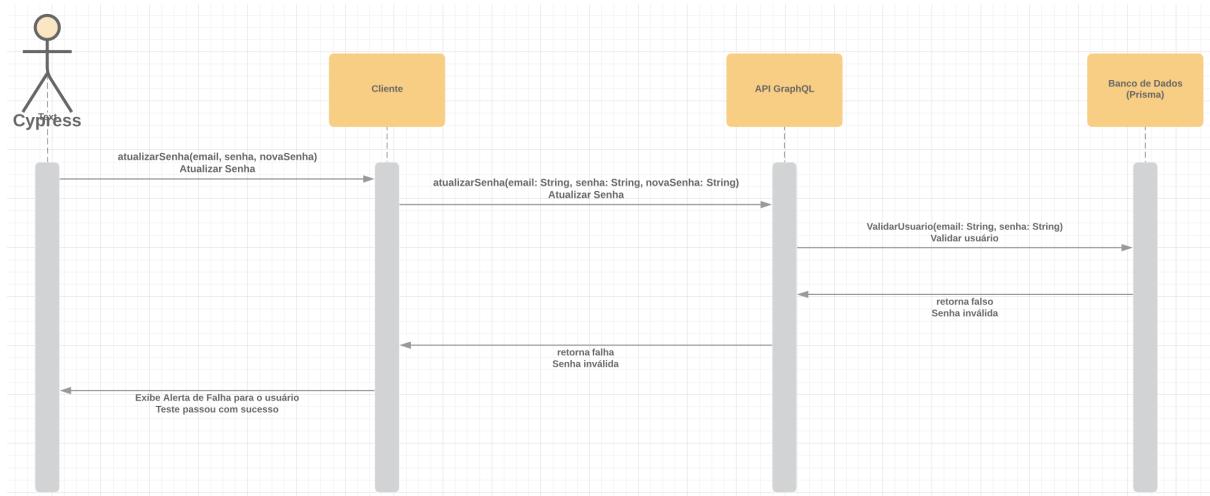


Figura 66: Diagrama de Sequência - Alterar Senha - falha

A figura 68 representa o código necessário para implementar o teste Alterar Senha que pode ser encontrado em: [/E2E/cypress/integration/08-alterar-senha.spec.js](#).

Entrando em mais detalhes do primeiro teste de Alterar Senha, com base na figura 68:

- A linha 1 importa a biblioteca faker;
- A linha 3 importa a função que faz requisições direto para a API;
- Da linha 5 à 9 foi construído uma função que busca na base de dados da aplicação o E-mail do Usuário passando por parâmetro o **email** e a **password**;
- A linha 11, foi declarado uma constante com um E-mail válido, denominada **email**;
- A linha 12, foi declarado uma constante com a Senha válida do E-mail válido, denominada **password**;

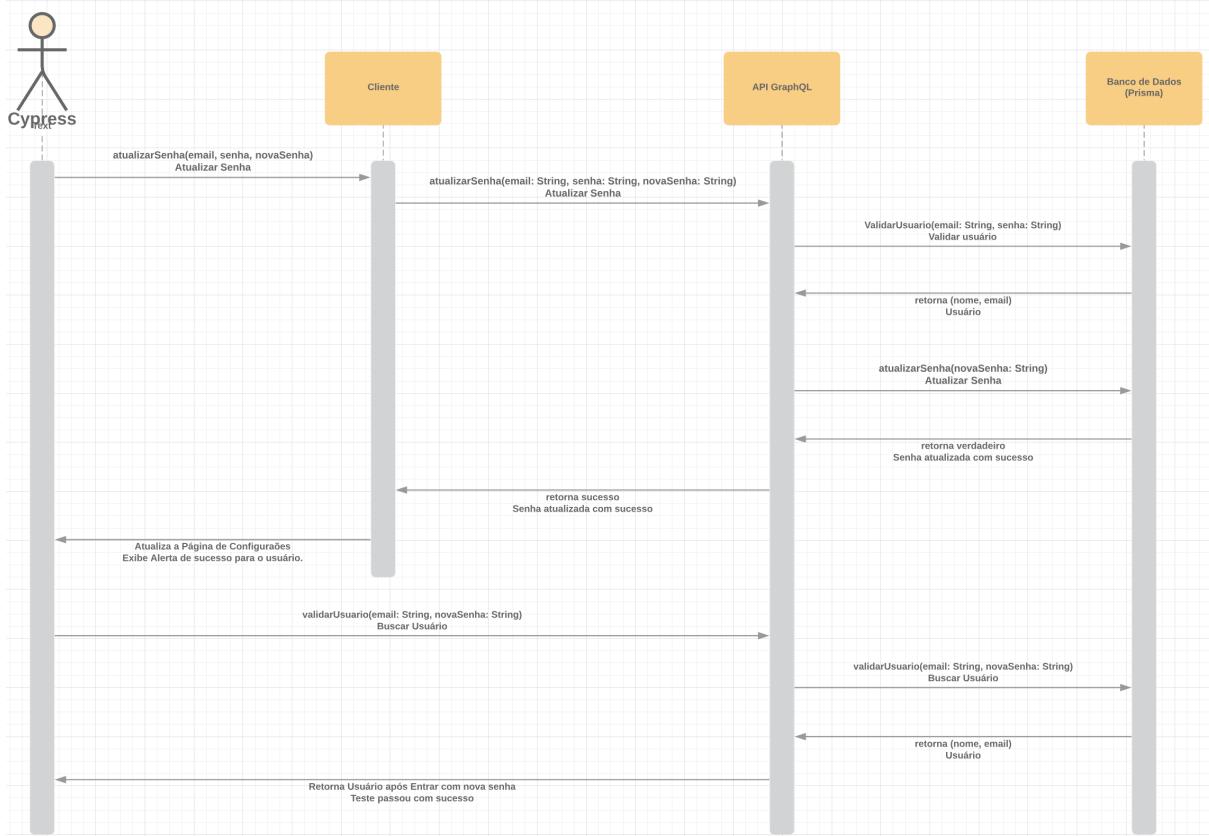


Figura 67: Diagrama de Sequência - Alterar Senha - sucesso

- A linha 13, foi declarado uma constante com uma nova Senha utilizando a biblioteca faker, denominada **newPassword**;
- A linha 15, foi declarado a função **describe**, a qual é utilizada para descrever o nome que representa de forma geral o conjunto de testes implementados à seguir;
- Da linha 16 à 22 foi declarado a função **before**, a qual é utilizada para definir uma rotina que será executada antes dos casos de testes **it**. Neste caso foi definido que:
 - 1.Na linha 20 por meio do método **cy.login**, foi invocada a função exemplificada na figura 37, a qual é utilizada para acessar a plataforma;
 - 2.Na linha 21 por meio do método **cy.get** foi definido que o elemento com **seletor: #dropdown** deve ser capturado, e com o método **.click()** foi definido que deve realizar a ação de clique no elemento capturado, neste caso, a função para abrir o menu e exibir as opções será invocada;
 - 3.Na linha 22 foi utilizado o método **cy.wait(1000)** foi definido para esperar 1 segundo antes de prosseguir com a próxima validação;

```

1 import faker from 'faker'
2
3 import { graphql_api } from '../../../../../utils/graphql-request.service'
4
5 const getUser = (email, password) => `{
6   user(email: "${email}", password: "${password}") {
7     email
8   }
9 }`
10
11 const email = 'fernando@gmail.com'
12 const password = '%fernando%123'
13 const newPassword = faker.internet.password(12)
14
15 it('Alterar senha', () => {
16   before(() => {
17     cy.login()
18     cy.get('#dropdown').click()
19     cy.wait(1000)
20     cy.get('#settings').click()
21     cy.wait(1000)
22   })
23
24   it('Alterar senha do usuário', () => {
25     cy.get('#change-password').click()
26
27     cy.get('#new-password').type(newPassword)
28     cy.get('#confirm-new-password').type(newPassword)
29     cy.get('#current-password').type(password)
30     cy.get('#show-password').click()
31     cy.get('#save-password').click()
32
33     cy.get('#text-alert').should('contain', 'Senha incorreta, tente novamente com outra senha.')
34     cy.wait(3000)
35
36     cy.get('#current-password').clear()
37     cy.get('#current-password').type(newPassword)
38     cy.get('#save-password').click()
39     cy.get('#text-alert').should('contain', 'Senha alterada com sucesso!')
40     cy.wait(3000)
41   })
42
43   it('Validar alteração', async () => {
44     try {
45       await graphql_api(getUser(email, password))
46     } catch (error) {
47       expect(error.response.errors[0].message).to.be.equal('Invalid password')
48     }
49
50     const { user } = await graphql_api(getUser(email, newPassword))
51     expect(user.email).to.be.equal(email)
52   })
53 })

```

Figura 68: Código de Teste Cypress - Alterar Senha

- 4.Na linha 23 por meio do método **cy.get** foi definido que o elemento com **seletor: #settings** deve ser capturado, e com o método **.click()** foi definido que deve realizar a ação de clique no elemento capturado, neste caso, a função para realizar o redirecionamento para a página Configurações será invocada;
- 5.Na linha 24 foi utilizado o método **cy.wait(1000)** foi definido para esperar 1 segundo antes de prosseguir com a próxima validação;
- Na linha 27 foi definido o primeiro caso de teste por meio da função **it**, denominada

Alterar senha do usuário, tal teste consistiu em:

- 1.Na linha 25 por meio do método **cy.get** foi definido que o elemento com **seletor: #change-password** deve ser capturado, e com o método **.click()** foi definido que deve realizar a ação de clique no elemento capturado, neste caso, a função para realizar o redirecionamento para a aba de Alterar Senha será invocada;
- 2.Na linha 27 por meio do método **cy.get** foi definido que o elemento com **seletor: #new-password** deve ser capturado, e com o método **.type** foi definido que o valor **newPassword** deve ser atribuído ao elemento capturado;
- 3.Na linha 28 por meio do método **cy.get** foi definido que o elemento com **seletor: #confirm-new-password** deve ser capturado, e com o método **.type** foi definido que o valor **newPassword** deve ser atribuído ao elemento capturado;
- 4.Na linha 29 por meio do método **cy.get** foi definido que o elemento com **seletor: #current-password** deve ser capturado, e com o método **.type** foi definido que o valor **newPassword** deve ser atribuído ao elemento capturado;
- 5.Na linha 30 por meio do método **cy.get** foi definido que o elemento com **seletor: #show-password** deve ser capturado, e com o método **.click()** foi definido que deve realizar a ação de clique no elemento capturado, neste caso, será exibido a senha para o usuário;
- 6.Na linha 31 por meio do método **cy.get** foi definido que o elemento com **seletor: #save-password** deve ser capturado, e com o método **.click()** foi definido que deve realizar a ação de clique no elemento capturado, neste caso, a função para salvar a senha do usuário será invocada;
- 7.Na linha 33 por meio do método **cy.get**, foi definido que o elemento com **seletor: #text-alert** deve ser capturado e com o método **.should** foi validado que a mensagem do alerta deve conter o seguinte texto: "**Senha incorreta, tente novamente com outra senha**", uma vez que a senha do usuário é inválida;
- 8.Na linha 34 foi utilizado o método **cy.wait(3000)** foi definido para esperar 3 segundos antes de prosseguir com a próxima ação;
- 9.Na linha 36 por meio do método **cy.get** foi definido que o elemento com **seletor: #current-password** deve ser capturado, e com o método **.clear** foi definido que deve limpar/apagar o valor do campo em questão;

- 10.Na linha 50 por meio do método **cy.get** foi definido que o elemento com **seletor: #current-password** deve ser capturado, e com o método **.type** foi definido que o valor **password** inválido deve ser atribuído ao elemento capturado;
- 11.Na linha 51 por meio do método **cy.get** foi definido que o elemento com **seletor: #save-password** deve ser capturado, e com o método **.click()** foi definido que deve realizar a ação de clique no elemento capturado, neste caso, a função para salvar as informações do usuário será invocada;
- 12.Na linha 52 por meio do método **cy.get**, foi definido que o elemento com **seletor: #text-alert** deve ser capturado e com o método **.should** foi validado que a mensagem do alerta deve conter o seguinte texto: "**Senha alterada com sucesso!**", uma vez que a senha do usuário é válida;
- 13.Na linha 53 foi utilizado o método **cy.wait(3000)** foi definido para esperar 3 segundos antes de prosseguir com a próxima ação;

Entrando em mais detalhes do segundo teste de Alterar Senha, com base na figura 67:

- Na linha 43 foi definido o segundo caso de teste por meio da função **it**, denominada **Validar alteração**, tal teste consistiu em:
 - 1.Na linha 45 foi enviado uma requisição direta para a API, passando o (**email** e **password**) do usuário por parâmetro e esperado uma objeto de **error** como retorno;
 - 2.Nas linhas 47 e 61 foi realizado a validação, onde o **error** deve ser um objeto que contém a seguinte mensagem "**Invalid password**", uma vez que a Senha do usuário foi alterado na aplicação;
 - 3.Na linha 50 foi enviado uma requisição direta para a API, passando o (**email** e **newPassword**) do usuário por parâmetro e esperado uma objeto **user** como retorno;
 - 4.Na linha 51 foi realizado a validação, onde o **user** deve ser um objeto do usuário e foi verificado se o atributo **user.email** name é igual ao atributo **email**,

A figura 69 representa a captura da interface do Cypress, que mostra o resultado do primeiro conjunto de teste executado, onde foi realizado seis validações.

```

▼ Alterar senha
  ✓ Alterar senha do usuário
    ▶ BEFORE ALL ...
    ▼ TEST
      1 GET      #change-password
      2 - CLICK
      3 GET      #new-password
      4 - TYPE   U1cGUCwWy_1Z
      5 GET      #confirm-new-password
      6 - TYPE   U1cGUCwWy_1Z
      7 GET      #current-password
      8 - TYPE   U1cGUCwWy_1Z
      9 GET      #show-password
     10 - CLICK
     11 GET      #save-password
     12 - CLICK
     13 GET      #text-alert
     14 - ASSERT expected <div#text-alert> to contain Senha incorreta, tente
                 novamente com outra senha.

      15 WAIT    3000
      16 GET      #current-password
      17 - CLEAR
      18 GET      #current-password
      19 - TYPE   %fernando%123
      20 GET      #save-password
      21 - CLICK
      22 GET      #text-alert
      23 - ASSERT expected <div#text-alert> to contain Senha alterada com sucesso!
      24 WAIT    3000

  ✓ Validar alteracao
    ▼ TEST
      1 ASSERT expected Invalid password to equal Invalid password
      2 ASSERT expected fernando@gmail.com to equal fernando@gmail.com

  ▼ Voltar senha
    ✓ Voltar senha do usuário
    ✓ Validar alteracao

```

Figura 69: Teste Cypress - Alterar Senha

B.3.8.2 SEGUNDO CONJUNTO DE TESTE

Foi utilizado a senha válida para alterar a seguinte informação: (Senha = password), onde foi esperado um retorno de sucesso, após receber o retorno de sucesso foi verificado se de fato a Senha do usuário em questão foi voltada conforme a base de dados da aplicação. Veja o Caso de Teste representado pelo Diagrama de Sequência 67 e a exemplificação do teste à seguir.

A figura 70 representa o código necessário para implementar o teste Voltar Senha que pode ser encontrado em: [/E2E/cypress/integration/08-alterar-senha.spec.js](#).

```

55  describe('Voltar senha', () => {
56    before(() => {
57      cy.login()
58      cy.get('#dropdown').click()
59      cy.wait(1000)
60      cy.get('#settings').click()
61      cy.wait(1000)
62    })
63
64    it('Voltar senha do usuário', () => {
65      cy.get('#change-password').click()
66
67      cy.get('#new-password').type(password)
68      cy.get('#confirm-new-password').type(password)
69      cy.get('#current-password').type(newPassword)
70      cy.get('#save-password').click()
71      cy.get('#text-alert').should('contain', 'Senha alterada com sucesso!')
72      cy.wait(3000)
73    })
74
75    it('Validar alteracao', async () => {
76      try {
77        await graphql_api(getUser(email, newPassword))
78      } catch (error) {
79        expect(error.response.errors[0].message).to.be.equal('Invalid password')
80      }
81
82      const { user } = await graphql_api(getUser(email, password))
83      expect(user.email).to.be.equal(email)
84    })
85  })

```

Figura 70: Código de Teste Cypress - Voltar Senha

Entrando em mais detalhes do primeiro teste de Voltar Senha, com base na figura 70:

- A linha 55, foi declarado a função **describe**, a qual é utilizada para descrever o nome que representa de forma geral o conjunto de testes implementados à seguir;
- Da linha 56 à 62 foi declarado a função **before**, a qual é utilizada para definir uma rotina que será executada antes dos casos de testes **it**. Neste caso foi definido que:
 - 1.Na linha 57 por meio do método **cy.login**, foi invocada a função exemplificada na figura 37, a qual é utilizada para acessar a plataforma;
 - 2.Na linha 58 por meio do método **cy.get** foi definido que o elemento com **seletor: #dropdown** deve ser capturado, e com o método **.click()** foi definido que deve realizar a ação de clique no elemento capturado, neste caso, a função para abrir o menu e exibir as opções será invocada;
 - 3.Na linha 59 foi utilizado o método **cy.wait(1000)** foi definido para esperar 1 segundo antes de prosseguir com a próxima validação;
 - 4.Na linha 60 por meio do método **cy.get** foi definido que o elemento com **seletor: #settings** deve ser capturado, e com o método **.click()** foi definido que deve realizar a ação de clique no elemento capturado, neste caso, a função para realizar o redirecionamento para a página Configurações será invocada;
 - 5.Na linha 61 foi utilizado o método **cy.wait(1000)** foi definido para esperar 1 segundo antes de prosseguir com a próxima validação;
- Na linha 64 foi definido o primeiro caso de teste por meio da função **it**, denominada **Voltar senha do usuário**, tal teste consistiu em:
 - 1.Na linha 65 por meio do método **cy.get** foi definido que o elemento com **seletor: #change-password** deve ser capturado, e com o método **.click()** foi definido que deve realizar a ação de clique no elemento capturado, neste caso, a função para realizar o redirecionamento para a aba de Alterar Senha será invocada;
 - 2.Na linha 67 por meio do método **cy.get** foi definido que o elemento com **seletor: #new-password** deve ser capturado, e com o método **.type** foi definido que o valor **password** deve ser atribuído ao elemento capturado;
 - 3.Na linha 68 por meio do método **cy.get** foi definido que o elemento com **seletor: #confirm-new-password** deve ser capturado, e com o método **.type** foi definido que o valor **password** deve ser atribuído ao elemento capturado;
 - 4.Na linha 69 por meio do método **cy.get** foi definido que o elemento com **seletor: #current-password** deve ser capturado, e com o método **.type** foi definido que o valor **newPassword** deve ser atribuído ao elemento capturado;

- 5.Na linha 70 por meio do método **cy.get** foi definido que o elemento com **seletor: #save-password** deve ser capturado, e com o método **.click()** foi definido que deve realizar a ação de clique no elemento capturado, neste caso, a função para salvar a senha do usuário será invocada;
- 6.Na linha 71 por meio do método **cy.get**, foi definido que o elemento com **seletor: #text-alert** deve ser capturado e com o método **.should** foi validado que a mensagem do alerta deve conter o seguinte texto: "**Senha alterada com sucesso!**", uma vez que a senha do usuário é válida;
- 7.Na linha 72 foi utilizado o método **cy.wait(3000)** foi definido para esperar 3 segundo antes de prosseguir com a próxima validação;

Entrando em mais detalhes do segundo teste de Voltar Senha, com base na figura 70:

- Na linha 75 foi definido o segundo caso de teste por meio da função **it**, denominada **Voltar alteração**, tal teste consistiu em:
 - 1.Na linha 77 foi enviado uma requisição direta para a API, passando o (**email** e **newPassword**) do usuário por parâmetro e esperado uma objeto de **error** como retorno;
 - 2.Nas linha 79 foi realizado a validação, onde o **error** deve ser um objeto que contém a seguinte mensagem "**Invalid password**", uma vez que a Senha do usuário foi alterado na aplicação;
 - 3.Na linha 82 foi enviado uma requisição direta para a API, passando o (**email** e **password**) do usuário por parâmetro e esperado uma objeto **user** como retorno;
 - 4.Na linha 83 foi realizado a validação, onde o **user** deve ser um objeto do usuário e foi verificado se o atributo **user.email** é igual ao atributo **email**, como esperado após realizar login meio da aplicação;

A figura 71 representa a captura da interface do Cypress, que mostra o resultado do segundo conjunto de teste executado, onde foi realizado três validações.

```
▶ Alterar senha ...
  ▾ Voltar senha
    ✓ Voltar senha do usuário
      ▶ BEFORE ALL ...
      ▾ TEST
        1 GET      #change-password
        2 - CLICK
        3 GET      #new-password
        4 - TYPE   %fernando%123
        5 GET      #confirm-new-password
        6 - TYPE   %fernando%123
        7 GET      #current-password
        8 - TYPE   U1cGUCwWY_1Z
        9 GET      #save-password
       10 - CLICK
       11 GET      #text-alert
       12 - ASSERT expected <div#text-alert> to contain Senha alterada com sucesso!
       13 WAIT     3000

    ✓ Validar alteracao
      ▾ TEST
        1 ASSERT  expected Invalid password to equal Invalid password
        2 ASSERT  expected fernando@gmail.com to equal fernando@gmail.com
```

Figura 71: Teste Cypress - Voltar Senha