ZERO WASTE

Zero Waste

Autores: Lucas Lima Souza, Guilherme Gavioli, Raquel Anicio dos Santos.

#1 Teste unitário (Descrição de solicitação):

```
TestonDescriptionInputChange_descricao_invalida(){
    this.valid_description = false;
    this.descriptionInput.nativeElement.content = "dslankds";
    if (this.valid_description != false){
      console.log("Erro!");
    }
  }

  TestonDescriptionInputChange_descricao_valida(){
    this.valid_description = false;
    this.descriptionInput.nativeElement.content =
"dnalçflasfdnlaflaflwilfliaefgaeilgflaenglealglrlnglaiengilaelginaliewg
lneangaenglianeg";
    if (this.valid_description == false){
      console.log("Erro!");
    }
  }
```

Ambos os casos servem para testar se uma descrição de uma solicitação é válida no primeiro uma descrição com menos de 85 caracteres é inserida num campo de texto e isso ativa uma função que avalia a validez do que foi inserido e muda a variável “valid_description”, depois disso é checado se a variável ainda está falsa já que a descrição inserida é inválida.

No segundo caso uma descrição maior que 85 caracteres é inserida e novamente é checada a validez da descrição para ver se agora a variável mudou para verdadeira.

#2 Teste unitário (Email cadastro):

```
 TestonEmailInputChange_invalido() {
    this.valid_email = false;
    this.emailContainer.nativeElement.content = 'valido@email.com';
    if (this.valid_email == false){
      console.log("Erro!");
    }
```

```
  }

  TestonEmailInputChange_valido() {
    this.valid_email = false;
    this.emailContainer.nativeElement.content = 'invalido';
    if (this.valid_email != false){
      console.log("Erro!");
    }
  }
```

Ambos os casos servem para testar se um email no cadastro são válida no primeiro um email sem a formatação correta é inserido num campo de texto e isso ativa uma função que avalia a validez do que foi inserido e muda a variável “valid_email”, depois disso é checado se a variável ainda está falsa já que a descrição inserida é inválida.

No segundo caso um email com a formatação correta é inserido e novamente é checada a validez da descrição para ver se agora a variável mudou para verdadeira.

#3 Teste unitário (Favoritar ong):

```
   async TestlikeOng(){
    var id_ong = '0';
   await this.likeOng(id_ong,0);
   const res = await fetch(`http://localhost:3000/mylikes`, {
    credentials: 'include',
    method: 'GET',
   })
    const data = await res.json()
    if (data && res.status === 200){
      this.my_likes = data;
      var x = this.my_likes.find(ong.id: id_ong);
      if (x == null){
        console.log('Erro!');
      }
      else{
        this.unlikeOng(id_ong,0);
      }
    }
  }
```

Este teste serve para avaliar se a função de favoritar ONG está funcionando normalmente primeiro é definido o id da ong que é um id pre-salvo de uma ONG de teste então é executada a função para favoritar a ONG depois é buscado no banco de dados os favoritos do usuário, o id da ong é procurado dentro do que é retornado e por fim remove o favorito da ONG.

#1 Teste de componente (Fazer um agendamento):

```
async makeAppointmentTest(){
```

```
    this.order_id = 0;
    this.inputData = ['arroz','feijão'];
    dia = 'ter';
    await makeAppointment();
    const res = await fetch(`http://localhost:3000/checkappointment`, {
            credentials: 'include',
            body: JSON.stringify({order_parent_id: this.order_id}),
            method: 'GET',
      });
      if (res.text() != 'ok'){
        console.log('Ocorreu um erro');
      }
      const res = await
fetch(`http://localhost:3000/deleteappointment`, {
            credentials: 'include',
            body: JSON.stringify({order_parent_id: this.order_id}),
            method: 'GET',
      });
  }
```

O teste consiste em acessar o código de uma ong pré-salva que fica oculta própria para testes então acessar o pedido de index ‘0’ e fazer criar um agendamento que cadastre itens que contém dentro dessa solicitação pré-salva e por fim escolher um dia em que está ONG estará aberta (informação que está salva dentro da ONG de teste previamente) e então criar a solicitação, depois checar o banco de dados para ter certeza que o agendamento foi criado e por fim apagar a solicitação.

#2 Teste de componente (Criar solicitação):

```
 async TestcreateOrder() {
    var solicitacao_id = 0;
    this.items = ['arroz', 'feijão', 'macarrão'];
    this.name = 'solicitação teste';
    this.description = 'descrição teste';
    await this.createOrder();

    const res = await fetch(`http://localhost:3000/getorder`, {
      credentials: 'include',
      method: 'GET',
      body: JSON.stringify({id: solicitacao_id})
    });
    if (res.status === 200) {
      const res = await fetch(`http://localhost:3000/deleteorder`, {
        credentials: 'include',
        method: 'GET',
```

```
        body: JSON.stringify({id: solicitacao_id})
      });
    } else {
      console.log('erro solicitação não encontrada!');
    }
  }
```

Uma ONG de teste criada previamente é acessada, as especificações da solicitação são criadas nome, descrição e alimentos a serem adicionados, com isso a solicitação é criada e uma busca no banco de dados é feita para ver se a solicitação foi realmente criada depois disso a solicitação é apagada.

Teste de Sistema:

```
import {LoginComponent} from './login/login.component';
import {FazerAgendamentoComponent} from
'./fazer-agendamento/fazer-agendamento.component';
import {AuthService} from './auth.service';
export class testeSistema{
    public constructor(private Login:LoginComponent, private
Agendar:FazerAgendamentoComponent, private Autenticacao: AuthService){}
    email = "";
    password = "";
    public order_id: string = '';
    public inputData: any[] = [0,0,0,0,0,0,0];
    async TestSystem(){
        this.email = "teste@email.com";
        this.password = "123";
        try{
        await this.Login.login();
        }catch{
            return 'login falhou';
        }
        this.order_id = '0';
        this.inputData = ['arroz','feijão'];
        try{
        await this.Agendar.makeAppointment();
        }catch{
            return 'agendamento falhou';
        }
        var agendamento = '';
        try{
            const res = await
fetch(`http://localhost:3000/getappointment`, {
```

```
                credentials: 'include',
                body: JSON.stringify({order_parent_id: this.order_id}),
                method: 'POST',
              })
              agendamento = (await res.text());
        }catch{
            return  'não achou o agendamento!';
        }
        try{
            const res = await
fetch(`http://localhost:3000/gerarcomprovante`, {
                credentials: 'include',
                body: JSON.stringify({appointment_id: agendamento}),
                method: 'POST',
              })
              console.log(await res.text())
        }catch{
            return 'não conseguiu gerar o comprovante!';
        }
        this.Autenticacao.logout();
    }
}
```

Teste feito para avaliar a “jornada do usuário” ou do doador, primeiro passo se logar dentro de uma conta pré-salva para testes esse é o primeiro “break-point” caso isso falhe irá retornar uma mensagem de erro dizendo que não foi possível se logar na conta, segundo passo vai na ONG de testes e cria um agendamento segundo break-point caso isso falhe outra mensagem irá aparecer dizendo que não foi possível fazer o agendamento, terceiro passo achar o id do agendamento que foi gerado dentro do banco de dados outro break-point caso isso não dê certo também aparecerá uma mensagem avisando, quarto passo gerar o comprovante para e enviá-lo por email para o usuário break-point caso não dê certo e por fim desloga o usuário da sessão.

Nesse teste uma única função chama o sistema inteiro porém é controlado pelos break-points para que não tenha mensagens genéricas de erros e sim momentos específicos de onde o sistema está falhando.