Aplicação Web Plan Your Meals

FASE DE IMPLEMETAÇÃO

Vasco Gomes | Tecnologias para a Web e Ambientes Móveis | 26/06/2022

Índice

Introdução

Nesta segunda fase de implementação do projeto desta unidade curricular, irei começar por identificar as 2 tarefas que foram implementadas nesta segunda fase do projeto, sendo estas:

- Criar Template de Refeição
- Mostrar os Templates de Refeição Criados

A API que foi utilizada para a realização deste projeto foi a *Spoonacular API*:

https://spoonacular.com/food-api/docs

Neste projeto foram utilizados 12 *endpoints* para conseguir mostrar todas as informações, devido á divisão de informação por *endpoints* por parte da *API*.

Método	Endpoint	Descrição
GET	https://api.spoonacular.com/mealplanner/generate	Gera um plano alimentar com 3 refeições random
POST	https://api.spoonacular.com/users/connect	Cria uma conta de utilizador
GET	https://api.spoonacular.com/recipes/complexSearch	Pesquisa Receitas
GET	https://api.spoonacular.com/recipes/s{idArray[i]}/information	Pesquisa Detalhes das Receitas
GET	https://api.spoonacular.com/food/ingredients/search	Pesquisa de Ingredientes
GET	https://api.spoonacular.com/food/ingredients/9040/information?	Pesquisa de Detalhes de Ingredientes
GET	https://api.spoonacular.com/food/products/search	Pesquisa de Produtos
GET	https://api.spoonacular.com/food/products/196488	Pesquisa de Detalhes de Produtos
POST	https://api.spoonacular.com/mealplanner/vasco/templates	Criar Plano Alimentar
GET	https://api.spoonacular.com/mealplanner/vasco/templates	Mostrar Templates Criados pelo Utilizador
GET	https://api.spoonacular.com/mealplanner/vasco/templates/5171	Mostrar Detalhes dos Templates Criados pelo Utilizador

Figura 1 - Tabela de Endpoints Utilizados

Neste projeto foi também ainda utilizado uma variedade de contas para ter acesso a chaves da API, uma vez que estas apenas tem um limite de 150 pedidos.

Decisões Globais de Implementação

Nesta secção irei retratar as minhas decisões globais acerca deste projeto.

TECNOLOGIAS DE DESENVOVLVIMENTO UTILIZADAS:

As tecnologias de desenvolvimento que foram utilizadas neste projeto são:

- React - Versão 18.2.0
- Node.js – Versão 8.11.0
- PhpStorm – Versão 212.5284.49
- Extensão Google Chrome Moesif Origin & CORS Changer – Versão 0.4.7

EXPLICAÇÃO SOBRE TECNOLOGIAS QUE SUPORTAM A ESTÉTICA E A INTERFACE GRÁFICA:

A tecnologia que foi utilizada para realizar o design do meu projeto foi um template de bootstrap que se adequa á temática do meu website.

DEFINIÇÕES USADAS PARA ASSEGURAR A RESPONSIVIDADE PARA DIFERENTES DIMENSÕES:

As definições utilizadas para assegurar a responsividade do website para diferentes dimensões de dispositivos é o template bootstrap em conjunto com o JavaScript que vem com o template, que renderiza as páginas de acordo com o tamanho desejado.

DECISÕES GERAIS APLICADAS EM TERMOS DE ACESSIBILIDADE PARA PESSOAS COM NECESSIDADES ESPECIAIS

As definições utilizadas em termos de acessibilidade neste projeto foram alternativas em texto para imagens

```
<div className="carousel-container">
    <img alt="Imagem de Banner" src={require("../img/slide/slide-1.jpg")} />
</div>
```

Figura 2 - Texto Alternativo para Imagem

De seguida foi ainda dado tempo suficiente ao utilizador para que este pudesse ler o conteúdo, como é o caso da figura abaixo que mostra um alerta que diz que a receita foi adicionada com sucesso e que apenas desaparece passados 3 segundos.

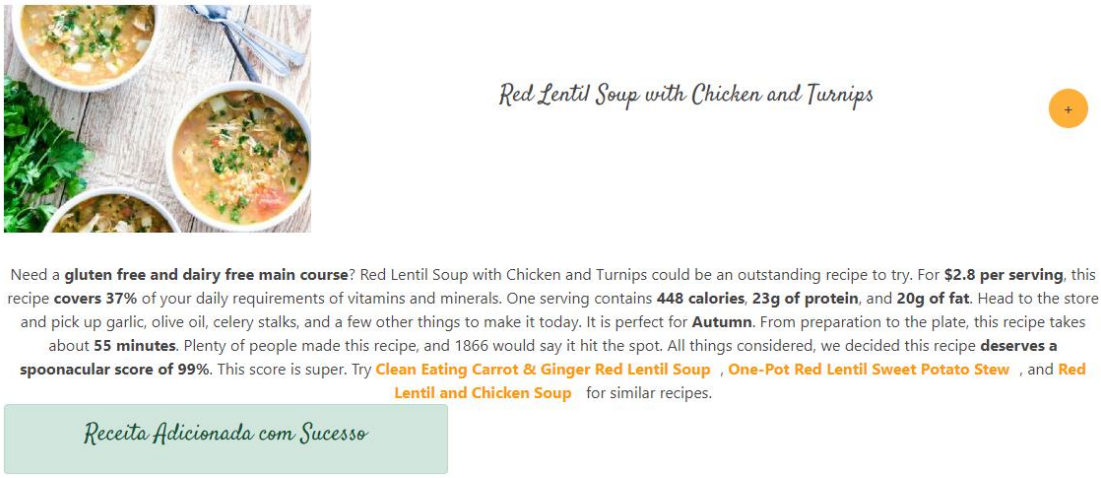

Figura 3 - Ajuda ao Utilizador com Tempo Suficiente

De seguida temos ainda a ajuda á navegação dos utilizadores, como se pode verificar em que temos na barra de navegação um Link para ir para o Espaço do Utilizador, para o Login e para o Sign Up respetivamente, e mais abaixo na página principal também podemos observar os mesmos links.

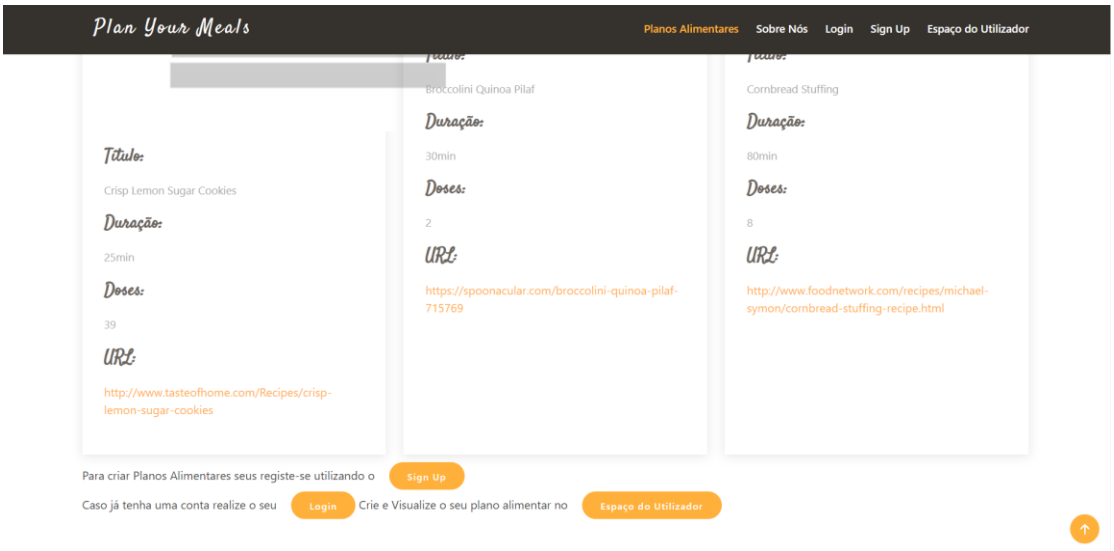

Figura 4 - Ajuda á Navegação

E por fim, temos ainda a ajuda ao utilizador a evitar erros, tendo ao invés de caixas de texto onde o utilizador possa escrever o que lhe apetecer e gerar um erro, existir campos específicos para a seleção da informação, que também foi apontado pelo professor na minha apresentação da fase de análise e que se encontra agora corrigido.

Como podemos observar na imagem seguinte, temos os campos Dia, Tipo de Refeição e Tipo diferentes, sendo que o dia é um calendário, e as restantes são elementos *<select>* com elementos *<option>* dentro e com *size* 3 para mostrar toda a informação lá dentro, á semelhança daquilo que fora pedido no teste desta unidade curricular.

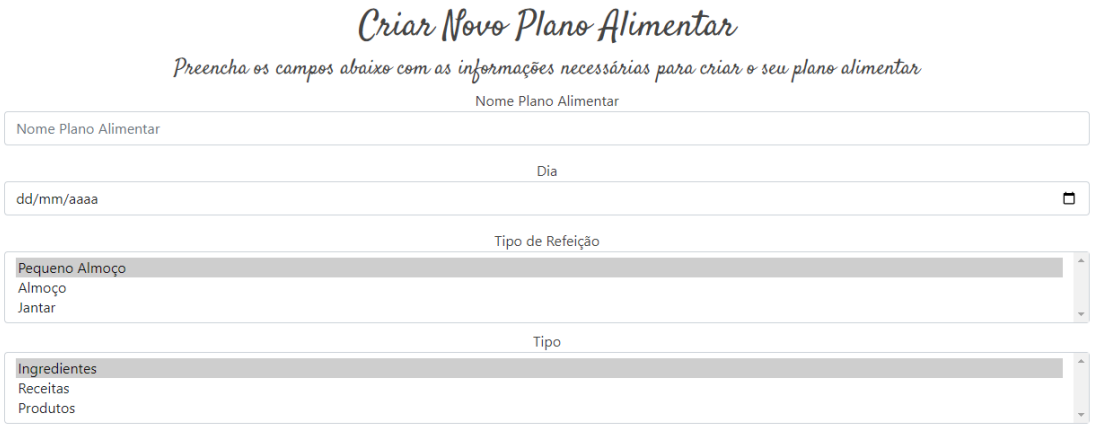

Figura 5 - Ajuda a Evitar Erros

Decisões de Implementação Específicas

Nesta secção irei detalhar acerca das minhas decisões de implementação específicas para cada página do meu website, começando pela página inicial.

PÁGINA INICIAL:

Na página inicial, existem os seguintes componentes como pode ser visto nas figuras abaixo.

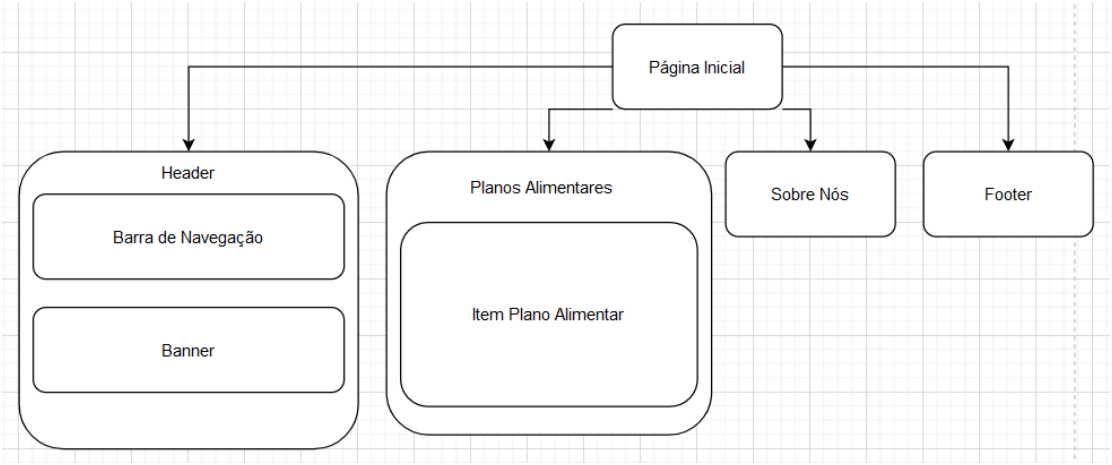

Figura 6 - Diagrama de Componentes da Página Inicial

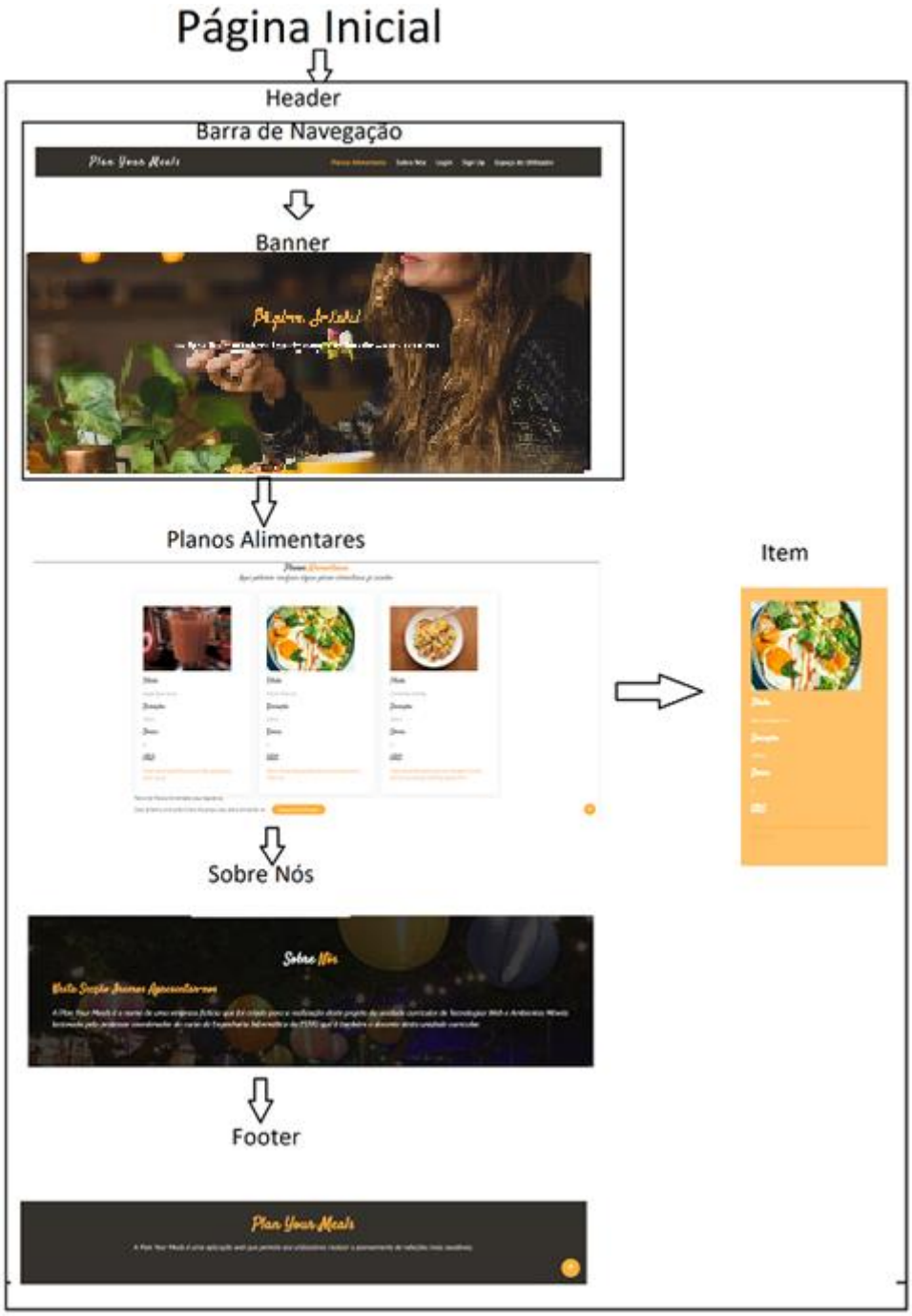

Figura 7 - Esquema de Componentes da Página Inicial

Agora irei explicar o código que faz esta página funcionar, que como se pode verificar na figura seguinte, e como podemos observar existe um elemento a ser passado para o header, que aqui no *React* são as *props*, que são basicamente os argumentos de uma função de *Javascript* por exemplo, e que servem para passar dados para dentro dos componentes.

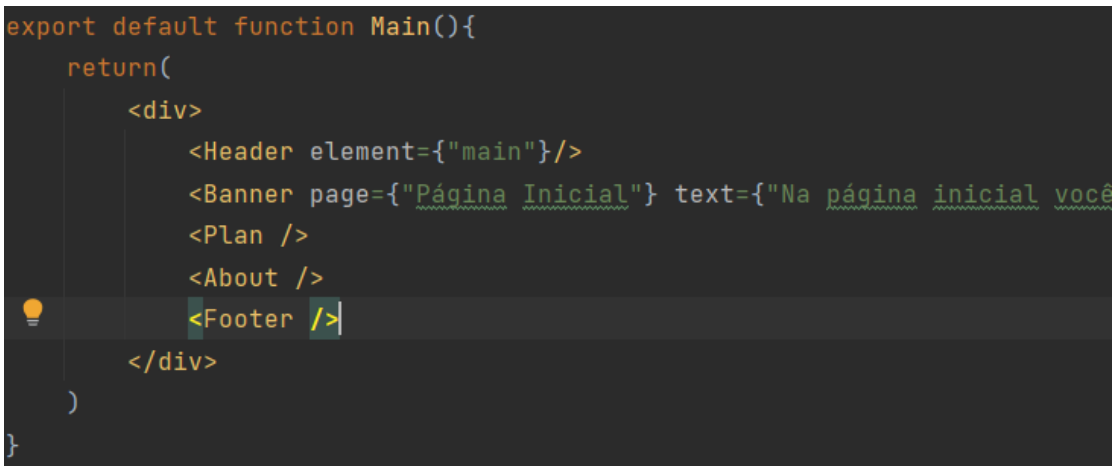

Figura 8 - Este é o código em React que Renderiza a Página Inicial, Chamando Todos os Outros Componentes

Header

Nesta secção é onde está a barra de navegação está a ser renderizada condicionalmente com o resultado dos props que foi falado anteriormente, uma vez que o Header está presente noutras 4 páginas, mas a barra de navegação é diferente para 3 delas.

De seguida podemos verificar a existência de um elemento <*link*> que redireciona os utilizadores para a página inicial caso este seja clicado, como podemos verificar na figura seguinte.

```
export default function Header(props){
    let nav_component=""
    if(props.element === "main"){
        nav_component = <Nav />
    } else if(props.element === "espaco_utilizador"){
        nav_component = <EspacoUtilizadorNav />
    }
    else {
        nav_component = <LoginNav />
    }
    return(
        <header id="header" className="fixed-top d-flex align-items-center header-transparent">
            <div className="container-fluid container-xl d-flex align-items-center justify-content-between">
                <div className="logo me-auto">
                    <h1><Link to={'/'} className="nav-link">Plan Your Meals</Link></h1>
                </div>
                {nav_component}
            </div>
        </header>
    )
}
```

Figura 9 - Ficheiro Header

Barra de Navegação

Nesta secção, iremos ver o código por trás da barra de navegação da página inicial apenas.

Como podemos verificar na figura seguinte existem 2 *tags* <*a*>, e outras 2 <*link*>, e isto porque para ser possível navegar entre secções da mesma página eu utilizei as *tags* <*a*>, mas para sair já não posso utilizar as mesmas, tenho de utilizar sim o sistema de rotas através das *tags* <*link*> e passando a página de destino no atributo "*to*", como é demonstrado na figura seguinte

```
export default function Nav(){
    return(
        <nav id="navbar" className="navbar order-last order-lg-0">
            <ul>
                <li><a className="nav-link scrollto active" href="#planos_alimentares">Planos Alimentares</a></li>
                <li><a className="nav-link scrollto" href="#sobre_nos">Sobre Nós</a></li>
                <li><Link to={'./LoginPage'} className="nav-link">Login</Link></li>
                <li><Link to={'./SignUpPage'} className="nav-link">Sign Up</Link></li>
                <li><Link to={'./EspacoUtilizadorPage'} className="nav-link">Espaço do Utilizador</Link></li>
            </ul>
        </nav>
    )
}
```

Figura 10 - Barra de Navegação

Vale ainda apena referir que as *tags* <*link*> não funcionariam se não existisse um sistema de rotas, algo que no tutorial do *Scrimba* não estava mencionado, logo, foi este o meu primeiro grande desafio descobrir como fazer rotas em *React*, que ficaram como é apresentado na figura abaixo, e que podem ser encontradas no ficheiro *App.js*.

```
class App extends Component{
    render(){
      return (
          <Router>
            <div className="App">
                <Routes>
                    {/*Rotas -> https://www.codingame.com/playgrounds/6517/react-router-tutorial*/}
                    <Route exact path='/' element={<Main />} />
                    <Route exact path='/LoginPage' element={<LoginPage />} />
                    <Route exact path='/SignUpPage' element={<SignUpPage />} />
                    <Route exact path='/EspacoUtilizadorPage' element={<EspacoUtilizadorPage />} />
                </Routes>
            </div>
          </Router>
      );
    }
}

export default App;
```

Figura 11 - Rotas para o Funcionamento dos Links

Banner

A secção do *Banner* não é nada mais nada menos do que uma secção cheia *de tags* <*div*> por conta do *template bootstrap* com uma *tag* <*img*> lá dentro que mostra uma imagem, e ainda em que página o utilizador está através de props, uma vez que as imagens são iguais na Página Inicial no Espaço de Utilizador como pode ser demonstrado na figura seguinte.

```jsx
export default function Banner(props){
    return(
        <section id="hero">

            <div className="hero-container">
                <div id="heroCarousel" data-bs-interval="5000" className="carousel slide carousel-fade"
                     data-bs-ride="carousel">
                    <ol className="carousel-indicators" id="hero-carousel-indicators"/>
                    <div className="carousel-inner" role="listbox">
                        <div className="carousel-item active">
                            <div className="carousel-container">
                                <img alt="Imagem de Banner" src={require("../img/slide/slide-1.jpg")} />
                            </div>

                            <div id="banner-text" className="carousel-content">
                                <div id="banner-text-1" className="p-3 mb-2 bg-dark text-white">
                                    <h2 className="animate__animated animate__fadeInDown"><span>{props.page}</span>
                                    </h2>
                                    <p className="animate__animated animate__fadeInUp">{props.text}</p>
                                </div>
                            </div>
                        </div>
                    </div>
                </div>
            </div>
        </section>
    )
}
```

Figura 12 - Banner

Planos Alimentares

Na secção dos planos alimentares começam as interações com a *API* escolhida, mas antes de chegarmos a esse ponto, podemos verificar que no componente *Plan*, temos o texto que compõem o componente e ainda uma chamada ao componente *Plan_Item*.

```jsx
export default function Plan(){
    return(
        <section id="planos_alimentares" className="why-us">
            <div className="container">
                <div className="section-title">
                    <h2>Planos <span>Alimentares</span></h2>
                    <h4>Aqui podemos verificar alguns planos alimentares já criados.</h4>
                </div>
                <div className="row">
                    <Plan_Item />
                </div>
                <p>Para criar Planos Alimentares seus registe-se.</p>
                <p>Caso já tenha uma conta Crie e Visualize o seu plano alimentar no <a href={'./EspacoUtilizadorPage'} className="book-a-table-btn scrollto">Espaço do Utilizador</a></p>
            </div>
        </section>
    )
}
```

Figura 13 - Componente Plan

Agora sim no componente *Plan_Item*, podemos verificar como é feita a renderização dos 3 planos apresentados na página inicial.

Começamos por definir um *state* com o nome de "*food*", este *state* é o que vai conter o array de objetos que depois vai ser iterado com recurso á função *map()*.

De seguida realizamos uma chamada a uma função da classe *React*, que é a função *useEffect()*, que diz ao React que o componente necessita de ser atualizado, e neste caso específico necessita de ser atualizado com os dados provenientes do *fetch*, passando-lhe o seguinte endpoints:

GET
https://api.spoonacular.com/mealplanner/generate?apiKey=f5028b2a056a4853b0a8134bc6b67523

Por sua vez após ter recebido a resposta do primeiro *.then(),* que é uma função que é executa logo após a conclusão do *fetch*, os dados vem num formato *JSON* e é necessário extraí-los para uma variável a que eu chamei de *foodArray*, para não ter de andar com um comboio de nomes atrás.

De seguida é feito uma atualização do estado da variável *food* através do método *setFood()*, passando-lhe o *array* criado anteriormente.

Por fim temos a função *return()* que retorna o componente renderizado e como já foi dito acima utiliza a função *.map()* que é como se tivéssemos a fazer um ciclo *for* no componente, e a informação que queremos renderizar é extraída através de uma variável que é a *foodAtributes*, que contem os mesmos dados que o *state food*, como podemos observar na figura seguinte.

```
export default function Plan_Item(){
    const [food, setFood] = React.useState( initialState: [])

    //https://api.spoonacular.com/mealplanner/generate?apiKey=f5028b2a056a4853b0a8134bc6b67523
    React.useEffect( effect: () => {
        fetch( input: "https://api.spoonacular.com/mealplanner/generate?apiKey=f5028b2a056a4853b0a8134bc6b67523") Promise<Response>
            .then(res => res.json()) Promise<any>
            .then(data => {
                let foodArray = data.week.friday.meals

                console.log("Food Array = "+foodArray)
                setFood(foodArray)
            })
    }, deps: [])
    return(
        food.map(foodAtributes => (
            <div className="col-lg-4">
                <div className="box" >
                    {/* IMG End Point https://spoonacular.com/recipeImages/{ID}-{SIZE}.{TYPE}*/}
                    <img src={`https://spoonacular.com/recipeImages/${foodAtributes.id}-312x231.${foodAtributes.imageType}`} alt="Imagem de Planos Alimentares"/>
                    <h4>Título:</h4><p>{foodAtributes.title}</p>
                    <h4>Duração:</h4><p>{foodAtributes.readyInMinutes}min</p>
                    <h4>Doses:</h4><p>{foodAtributes.servings}</p>
                    <h4>URL:</h4><a href={foodAtributes.sourceUrl}>{foodAtributes.sourceUrl}</a>
                </div>
            </div>
        ))
    )
}
```

Figura 14 - Componente Plan_Item

Sobre Nós

A secção Sobre Nós é apenas um componente que mostra algumas informações, como pode ser visualizado na figura abaixo.

```
export default function About(){
    return(
        <section id="sobre_nos" className="events">
            <div className="container">

                <div className="section-title">
                    <h2>Sobre <span>Nós</span></h2>
                </div>

                <div className="events-slider swiper">
                    <div className="swiper-wrapper">
                        <div className="swiper-slide">
                            <div className="row event-item">
                                    <h3>Nesta Secção Iremos Apresentar-nos</h3>
                                    <div className="price">
                                    </div>
                                    <p className="fst-italic">
                                        A <span>Plan Your Meals</span> é o nome de uma empresa fictícia que foi criada
                                        para a realização deste projeto da unidade curricular de Tecnologias Web e Ambientes
                                        Móveis lecionada pelo professor coordenador do curso de Engenharia Informática da ESTIG
                                        que é também o docente desta unidade curricular.
                                    </p>

                            </div>
                        </div>
                    </div>
                    <div className="swiper-pagination"></div>
                </div>

            </div>
        </section>
```

Figura 15 - Componente Sobre Nós

Footer

Por fim temos o footer que apresenta mais algumas informações sobre o sistema, como pode ser visto na figura seguinte.

```
export default function Footer(){
    return(
        <footer id="footer">
            <div className="container">
                <h3>Plan Your Meals</h3>
                <p>A Plan Your Meals é uma aplicação web que permite aos utilizadores realizar
            </div>
        </footer>
    )
}
```

Figura 16 - Componente Footer

SING UP

De seguida passamos para a secção do *Sing Up*, destinada á realização do registo no *website*, e podemos ainda ver as diferentes partes que o constituem, como podemos verificar na figura seguinte.

Figura 17 - Esquema de Componentes Página de Sing Up

Como podemos verificar pela figura anterior existem aqui algumas diferenças, e isso deve-se ao facto do componente que é invocado quando se chama a página de *Sing Up*, não possuir a mesma estrutura da página inicial, sendo que a diferença que salta logo mais á vista é o facto de não existir o *Banner*, uma vez que não existe imagem.

Na figura seguinte podemos verificar a composição da página de *Sing Up*.

```
export default function SignUpPage(){
    return(
        <div>
            <Header element={"sign_up"}/>
            <SignUp />
            <Footer />
        </div>
    )
}
```

Figura 18 - Composição da Página de Sign Up

Uma vez que já foi falado do *Header* e do *Footer*, irei passar esses componentes á frente, e seguir direto para a Barra de Navegação.

Barra de Navegação

A barra de navegação do *Sign Up* é um pouco diferente da que foi mostrada na página inicial, uma vez que não fazia sentido ter links de navegação a apontar para componentes que não existem nesta página web.

Como pode ser visto na figura seguinte, este componente tem o nome de *LoginNav*, isto é porque este componente é utilizado tanto nas páginas de *Login* como na de *Sign Up*, para reaproveitar o componente, uma vez que não faz sentido ter 2 componentes iguais com 2 nomes diferentes.

```
export default function LoginNav(){
    return(
        <nav id="navbar" className="navbar order-last order-lg-0">
            <ul>
                <li><Link to={'../LoginPage'} className="nav-link"> Login </Link></li>
                <li><Link to={'../SignUpPage'} className="nav-link">Sign Up</Link></li>
            </ul>
        </nav>
    )
}
```

Figura 19 - Componente LoginNav

Formulário de Sign Up

Nesta secção, irei demonstrar como fiz o *Sign Up* funcionar, como podemos verificar na figura seguinte, temos um state com o nome de *formData*, que é um *array* de objetos com os atributos que estão presentes no formulário.

De seguida temos a função *changeFormData()*, que é executada quando os dados do formulário são mudados, uma vez que esta função está associada a todos os campos do formulário através da função *onChange()*.

```
export default function SignUp(){

    const [formData, setForm] = React.useState( initialState: {
        name:"",
        lastName:"",
        userName:"",
        email:"",
        password:"",
        repeatPassword:""
    })


    function changeFormData(event){
        console.log(formData)

        const {name, value, type} = event.target
        setForm( value: prevFormData => ({
            ...prevFormData,
            [name]: value
            })
        )
    }
```

Figura 20 - Modificação de Valores do Formulário do Componente SignUp

Como podemos constatar na figura abaixo, esta é a maior função, que está presente neste ficheiro, e que como o próprio nome indica realiza a submissão dos dados que estão presentes no formulário para a *API*, tratando-se por isso mesmo de um *POST*:

POST:

https://api.spoonacular.com/users/connect?apiKey=2c7f4036297b414cb84a5fc55432e437

Na primeira linha vemos algo que ainda não se tinha visto, essa linha o que faz é que faz com que o formulário não dê *refresh* quando se submete os dados.

E de seguida são verificadas se as passwords introduzidas no formulário são iguais e é realizado o *POST* das informações para a *API*.

Á semelhança do *GET* que vimos anteriormente, ficamos á espera da resposta da submissão dos dados, e quando estes forem submetidos, coloco os mesmos dentro de um ficheiro para que possam ser utilizados para realizar o *login* do utilizador assim como a criação de planos alimentares e a visualização dos mesmos.

```
function submitData(event){
    event.preventDefault()

    if(formData.password === formData.repeatPassword){
        // Send data to the API via POST
        fetch( input: 'https://api.spoonacular.com/users/connect?apiKey=2c7f4036297b414cb84a5fc55432e437', init: {
            method: 'POST',
            headers: {'Content-Type': 'application/json'},
            body: JSON.stringify( value: {
                "username":formData.userName,
                "firstName":formData.firstName,
                "lastName":formData.lastName,
                "email":formData.email
            }) // body data type must match "Content-Type" header
        }).then((response : Response ) => {
            return response.json();
        }).then((parsedData) => {
            console.log("User Data Status = " + parsedData.status)
            console.log("User Data Username = " + parsedData.username)
            console.log("User Data spoonacularPassword= " + parsedData.spoonacularPassword)
            console.log("User Data hash= " + parsedData.hash)
            //Push Data Inside Array
            Utilizador.setUtilizadorData(parsedData.username, formData.email, formData.password, parsedData.spoon
        })
    }
}
```

Figura 21 - Função submitData do Componente SignUp

Ficheiro de Dados do Utilizador

Uma vez que falamos no ficheiro do Utilizador, vamos agora dar uma espreitadela nele.

Como podemos verificar na figura seguinte, a estrutura do mesmo é bastante similar a uma classe, temos os atributos definidos para serem preenchidos, e de seguida temos a função que coloca os dados dentro do ficheiro, como foi visto na secção de trás.

De seguida verificamos que existem vários *getters* para retirar os dados do ficheiro e puderem ser utilizados.

```
var Utilizador = (function () {

    var username = ""
    var email = ""
    var password = ""
    var spoonacularPassword = ""
    var hash = ""
    var utilizadorArrayObject = []

    function setUtilizadorData (username1, email1, password1, spoonacularPassword1, hash1){
        //Put Data in Array
        username = username1
        email = email1
        password = password1
        spoonacularPassword = spoonacularPassword1
        hash = hash1
    }

    var getUtilizadorUsername = function (){
        return username
    }

    var getUtilizadorEmail = function (){
        return email
    }

    var getUtilizadorPassword = function (){
        return password
    }
```

Figura 22 – Início Ficheiro Utilizador

Na figura seguinte conseguimos mais alguns *getters* e por fim uma função de *return*, em que esta dá o retorno dos métodos que podem ser utilizados.

```
    var getUtilizadorSpoonacularPassword = function (){
        return spoonacularPassword
    }

    var getUtilizadorHash = function (){
        return hash
    }

    return {
        setUtilizadorData: setUtilizadorData,
        getUtilizadorUsername: getUtilizadorUsername,
        getUtilizadorEmail : getUtilizadorEmail,
        getUtilizadorPassword : getUtilizadorPassword,
        getUtilizadorSpoonacularPassword: getUtilizadorSpoonacularPassword,
        getUtilizadorHash : getUtilizadorHash
    }
})()

export default Utilizador
```

Figura 23 - Final Ficheiro Utilizador

Com o ficheiro de dados do utilizador apresentado, podemos prosseguir normalmente com o *Sign Up*, e na figura seguinte podemos verificar a função *renderComponent()*, que inverte o *state* da notificação que está declarada logo acima da função.

Esta função serve para mostrar a mensagem de sucesso quando um utilizador realiza um registo.

```
const [notificacao, setNotificacao] = React.useState( initialState: false)
function renderComponent(){
    setNotificacao( value: prevState => !prevState)
}
```

Figura 24 - Função renderComponent

Por fim chegamos á função *return()* que retorna o formulário de *Sing Up* que vemos, com vários campos para preencher, como podemos verificar na figura seguinte.

```
return(
    <form  className="php-email-form" onSubmit={submitData}>
        <div className="text-center">
            <h1>Sign Up</h1>
            <h4>Preencha os campos do formulário para se registar</h4>
        </div>
        <div className="text-center">
            <div className="form-group">
                <label>Name</label>
                <input
                    type="name"
                    name="name"
                    className="form-control"
                    id="name"
                    placeholder="Name"
                    onChange={changeFormData}
                    value={formData.name}
                />
            </div>
            <div className="form-group mt-3">
                <label>Last Name</label>
                <input type="lastName"
                       className="form-control"
                       name="lastName"
                       id="lastName"
                       placeholder="Last Name"
                       onChange={changeFormData}
                       value={formData.lastname}
                />
            </div>
```

Figura 25 - Início Função return()

Para não ficar muito extenso, vou saltar direto para o final da função, e com isso podemos verificar na figura seguinte que todos os campos tem a função *onChange* que foi falada anteriormente, e conseguimos ver também que temos um campo *value* com o nome do atributo do *state*, para que seja possível ir buscar o valor do mesmo na hora de atualizar o campo.

Por fim temos ainda o botão que tem como função *onClick* a função *renderComponent()* que já foi falada anteriormente e por baixo disso temos a renderização condicional de um alerta com o texto a ser passado através de *props*, ou seja quando existe valores no *state* notificação o alerta pode ser visto, como podemos verificar na figura seguinte.

```
            <div className="form-group mt-3">
                <label>Repeat Password</label>
                <input type="password"
                       className="form-control"
                       name="repeatPassword"
                       id="repeatPassword"
                       placeholder="Repeat Password"
                       data-msg="Please enter at least 4 chars"
                       onChange={changeFormData}
                       value={formData.repeatPassword}
                />
            </div>
        </div>
        <div className="text-center">
            <button id="select-tipo" className="book-a-table-btn scrollto" onClick={renderComponent}>Sign Up</button>
            {notificacao && <Alert id="alert" props={"Utilizador Registado com Sucesso"}/>}
        </div>
    </form>
)
```

Figura 26 - Fim Função return()

E uma vez que falamos do alerta, deixo aqui uma figura com o código do mesmo.

```
export default function Alert(props){
    return(
        <div classs="container p-5">
            <div className="row no-gutters">
                <div className="col-lg-5 col-md-12">
                    <div className="alert alert-success fade show" role="alert">
                        <h4 className="alert-heading">{props.props}</h4>
                    </div>
                </div>
            </div>
        </div>
    )
}
```

Figura 27 - Ficheiro de Alerta

LOGIN

Uma vez que já falamos do *Sing Up*, não faz sentido não falar do *login* também, e da mesma maneira, começamos pelo diagrama que constitui a página de *Login*, como pode ser visto na figura seguinte.

Figura 28 - Esquema de Componentes de Formulário de Login

Antes de começar a falar dos componentes que constituem a página de *Login*, falta mostrar como estes estão organizados no código, como podemos observar na figura seguinte.

```
export default function LoginPage(){
    return(
        <div>
            <Header element={"login"}/>
            <Login />
            <Footer />
        </div>
    )
}
```

Figura 29 - Componente LoginPage

Formulário de Login

Uma vez que a barra de navegação é igual á barra de navegação que foi falada na secção anterior, vamos saltar diretamente para o Formulário de *Login*.

Á semelhança do formulário de *Sign Up*, temos um *state formData* que guarda os dados do formulário de *login*, nomeadamente o *email*, e a *password*, e de seguida temos também uma função já conhecida que guarda o *state* do formulário a cada alteração feita, pois está associada a uma função *onChange* em cada campo do formulário.

```
export default function Login(){

    const [formData, setForm] = React.useState( initialState: {
        email:"",
        password:""
    })

    function changeFormData(event){
        console.log(formData)

        const {name, value, type} = event.target
        setForm( value: prevFormData => ({
                ...prevFormData,
                [name]: value
            })
        )
    }
```

Figura 30 - Função changeFormData Formulário de Login

De seguida temos a função *submitData()*, que também já é conhecida nossa, mas agora tem aqui umas modificações, como por exemplo o uso do ficheiro *Utilizador*, assim como dos seu *getters* que foram falados anteriormente.

Posteriormente a isso, temos uma verificação dos dados que foram submetidos pelo formulário de *Login*, e os dados que vieram do ficheiro *Utilizador*, que foram inseridos no formulário de *Sign Up*, e os dados sejam iguais, é feito um redireccionamento para a página do *Espaço do Utilizador* de seguida, uma vez que a *API* não possui nenhum *endpoint* para realizar esta verificação.

```
const navigate = useNavigate()
function submitData(event){
    event.preventDefault()
    console.log("Data Submitted")

    console.log("UserName From Sing UP = " + Utilizador.getUtilizadorUsername())
    console.log("Email From Sing UP = " + Utilizador.getUtilizadorEmail())
    console.log("Password From Sing UP = " + Utilizador.getUtilizadorPassword())
    console.log("Spoonacular Password from Sing UP = " + Utilizador.getUtilizadorSpoonacularPassword())
    console.log("Hash from Sing UP = " + Utilizador.getUtilizadorHash())
    console.log(" ")

    if(Utilizador.getUtilizadorEmail() === formData.email){
        if(Utilizador.getUtilizadorPassword() === formData.password){
            //Redirect to Espaco Utilizador
            navigate('/EspacoUtilizadorPage')
        }
    }
}
```

Figura 31 - Função submitData() Componente Login

Por fim, temos a função *return()* que retorna o formulário que pode ser vista nas seguintes figuras.

```
return(
    <form  className="php-email-form" onSubmit={submitData}>
        <div className="text-center">
            <h1>Login</h1>
            <h4>Preencha o formulário para realizar o seu Login</h4>
        </div>
        <div className="text-center">
            <div className="form-group">
                <label>Email</label>
                <input
                    type="email"
                    name="email"
                    className="form-control"
                    id="email"
                    placeholder="Email"
                    onChange={changeFormData}
                    value={formData.email}
                />
            </div>
            <div className="form-group mt-3">
                <label>Password</label>
                <input type="password"
                       className="form-control"
                       name="password"
                       id="password"
                       placeholder="Password"
                       onChange={changeFormData}
                       value={formData.password}
                />
            </div>
```

Figura 32 - Início da Função Return do Formulário de Login

```
    </div>
    <div className="text-center">
        <button id="select-tipo" className="book-a-table-btn scrollto">Login</button>
    </div>
</form>
```

Figura 33 - Finalização da Função Return do Formulário de Login

ESPAÇO DO UTILIZADOR

Da mesma maneira que temos feito para todas as páginas, iremos ver agora o diagrama de componentes utilizados para esta página, como mostra a figura seguinte.

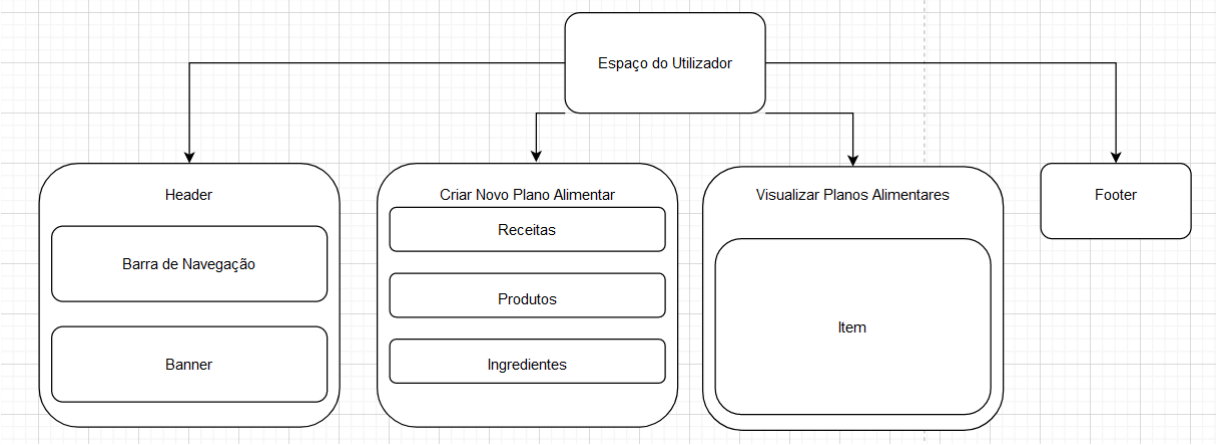

Figura 24 - Diagrama de Componentes do Espaço de Utilizador

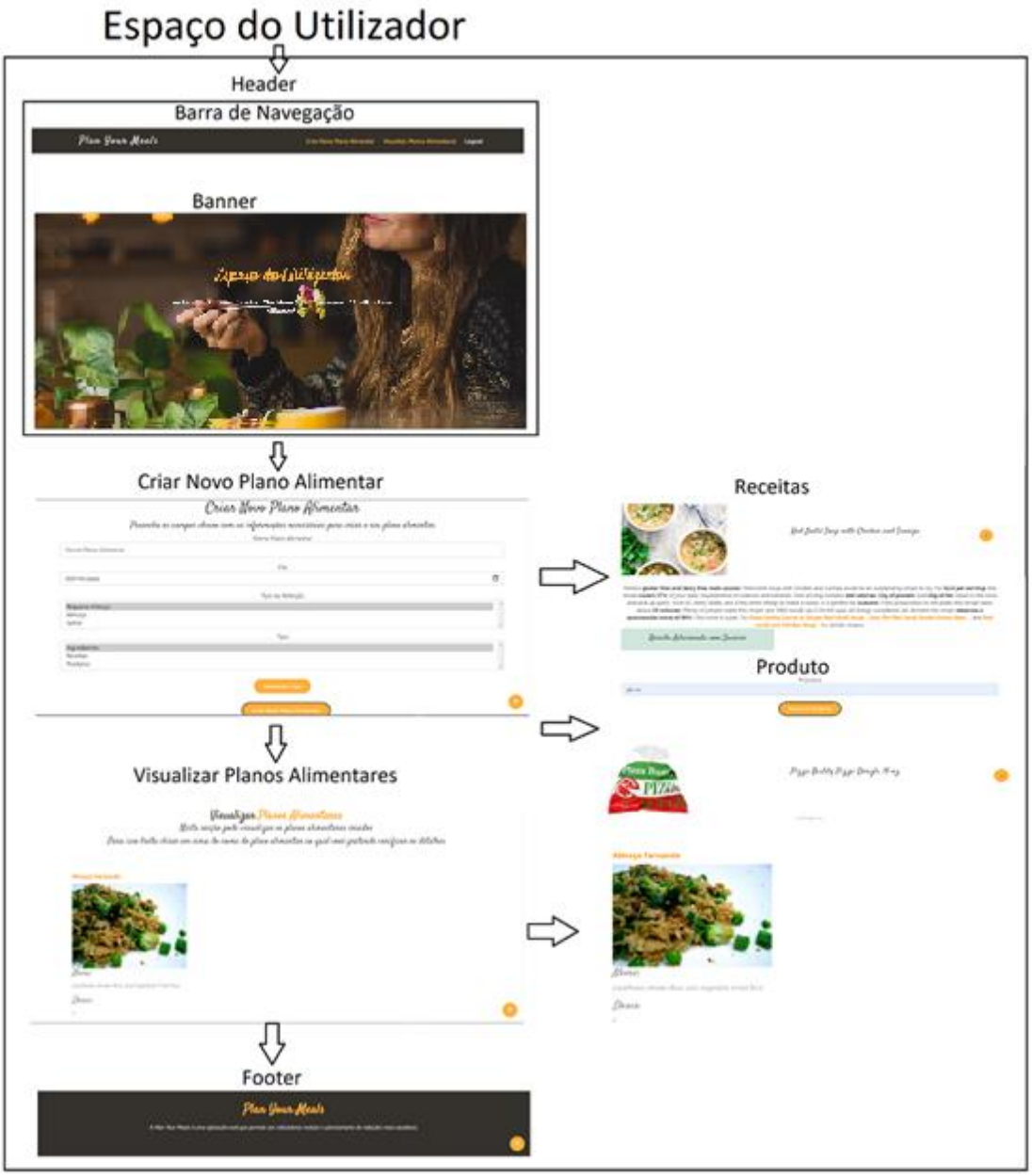

Figura 35 - Esquema de Componentes do Espaço do Utilizador

De seguida podemos verificar a estrutura do Espaço de Utilizador, como podemos verificar na figura seguinte.

```
export default function EspacoUtilizadorPage(){
    return(
        <div>
            <Header element={"espaco_utilizador"}/>
            <EspacoUtilizador />
            <Footer />
        </div>
    )
}
```

Figura 36 - Estrutura do Espaço de Utilizador

Barra de Navegação

Ao contrário da construção de uma casa em que não se começa pelo telhado, aqui mais uma vez começamos com a Barra de Navegação, que podemos verificar que esta barra de navegação é diferente da que consta nas páginas anteriores, na figura seguinte podemos verificar o código do mesmo.

```
export default function EspacoUtilizadorNav(){
    return(
        <nav id="navbar" className="navbar order-last order-lg-0">
            <ul>
                <li><a className="nav-link scrollto active" href="#criar-planos-alimentares"> Criar Novo Plano Alimentar </a></l
                <li><a className="nav-link scrollto active" href="#visualizar-planos_alimentares">Visualizar Planos Alimentares<
                <li><Link to={'/LoginPage'} className="nav-link">Logout</Link></li>
            </ul>
        </nav>
    )
}
```

Figura 37 - Barra de Navegação do Espaço de Utilizador

Irei saltar as secções *Header*, e *Footer* para não se tornar repetitivo.

Criar Novo Plano Alimentar

Agora chegamos á tarefa nº1 do projeto, que é a criação de um plano alimentar, utilizando vários componentes, e em primeiro vem o formulário da criação de planos alimentares.

Este formulário é similar aos restantes que já foram apresentados, uma vez que usa um *state formData* que guarda os dados, assim como um *state alert* para mostrar quando os dados são inseridos com sucesso.

De seguida temos a função *changeFormData()* que está associada a eventos *onChange* dos campos do formulário e que guarda no *state* os valores do formulário.

Figura 38 - Inicio da Função changeFormData

Depois como podemos verificar na figura seguinte, temos a declaração de mais 3 variáveis do tipo *state* que tem como estado inicial *false*, uma vez que nenhum deles está a ser mostrado, pois isso apenas acontece quando se carrega na caixa de seleção tipo e se carrega no botão "Selecionar Tipo", a função *selectType()* é ativada, e vai verificar qual das 3 opções foi clicada, e após isso inverte o valor do *state* de *false* para *true* e mostra o componente renderizado.

```
const [receitas, setReceitas] = React.useState( initialState: false)
const [produtos, setProdutos] = React.useState( initialState: false)
const [ingredientes, setIngredientes] = React.useState( initialState: false)

/**
 * Resume: Function that shows the selected component from select element
 */
function selectType(){
    //https://bobbyhadz.com/blog/react-onclick-show-component
    if(formData.tipo === "receitas"){
        setReceitas( value: currentValue => !currentValue)
    } else if(formData.tipo === "produtos"){
        setProdutos( value: currentValue => !currentValue)
    } else {
        setIngredientes( value: currentValue => !currentValue)
    }
}

/**
 * Resume: Function that closes a component
 * @param event
 */
function closeField(event){
    console.log("Event = " + event.target.name)
    if(event.target.value === receitas){
        setReceitas( value: false)
    } else if(event.target.value === produtos){
        setProdutos( value: false)
    } else {
        setIngredientes( value: false)
```

Figura 39 - Função selectType e Renderização do Mesmo

Na figura seguinte podemos visualizar as interfaces gráficas dos componentes mencionados.

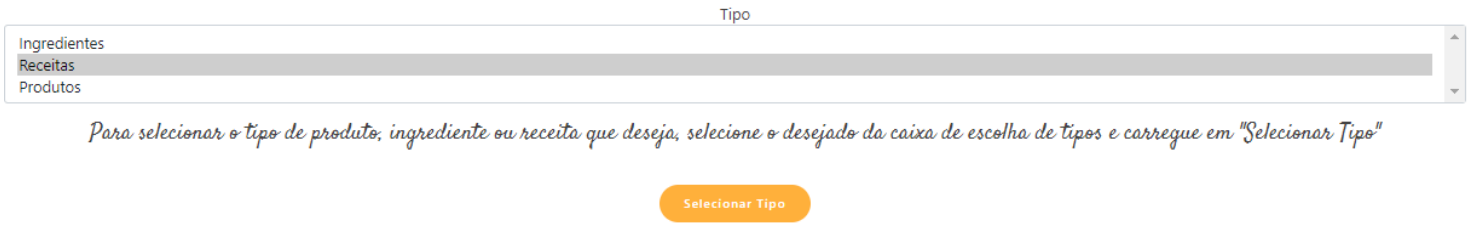

Figura 40 - Seleção do Tipo

Como podemos ver no caso das receitas, ele faz automático, isto porque o *endpoint* da *API* me mostra as receitas automaticamente, como podemos observar na figura seguinte.

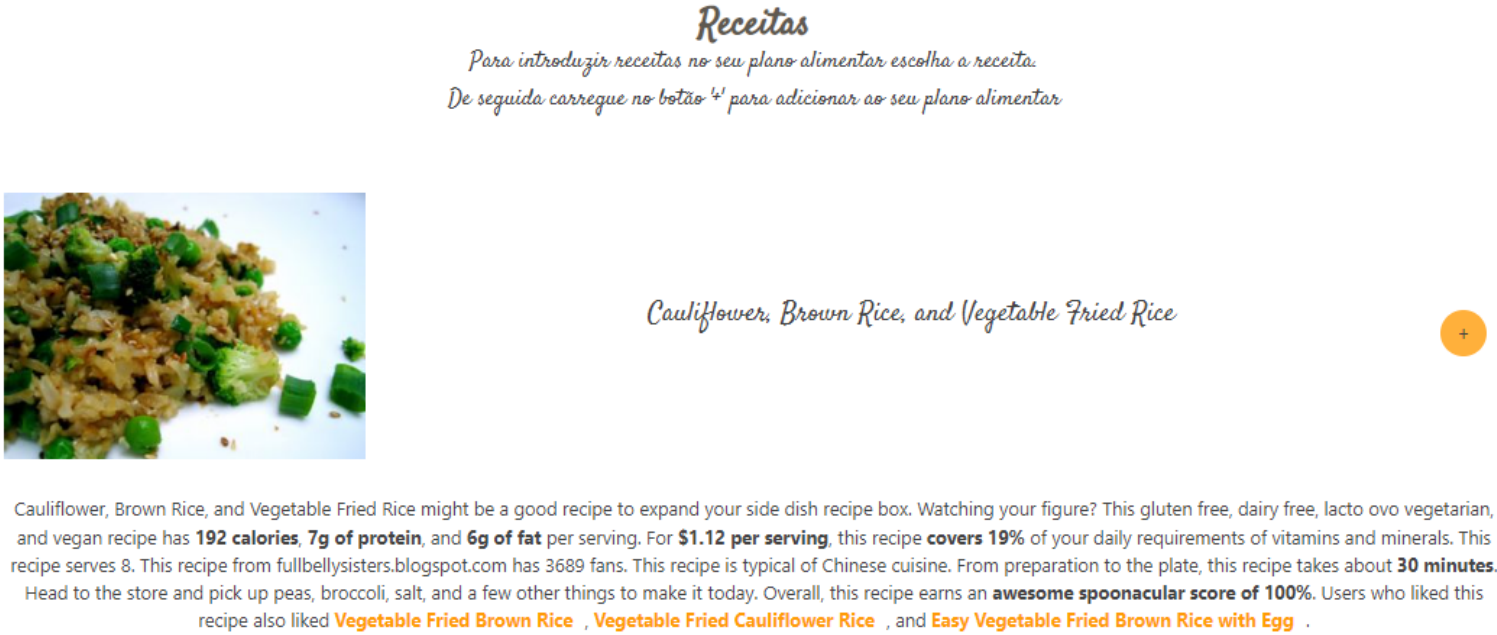

Figura 41 - Visualizar Receitas

No caso dos produtos e dos ingrediente, a história já é diferente, uma vez que a *API* não me fornece os dados como me fornece as receitas, ou seja quando quero disponibilizar receitas basta ir ao *endpoint* da *API* e buscar os dados automaticamente, mas no caso dos produtos e dos ingredientes, a *API* não fornece esses dados automaticamente, tem de ser o utilizador a pesquisar.

Eu queria ter feito algo similar ao que fiz com as receitas, uma vez que o utilizador realizar a pesquisa foi algo que o professor disse que não ficava bem no trabalho, quando fiz a apresentação da fase de desenho do projeto.

Figura 42 - Pesquisar e Visualizar Produtos

Após esta breve explicação, passaremos então a ver como é constituído o componente das receitas, uma vez que a única coisa que muda é a existência de um formulário para realizar a procura desejada.

Componente Espaço do Utilizador Receitas

Vamos agora visualizar como é feito o componente das receitas, e podemos desde já visualizar que tem um *state* com o nome de receitas, que vai servir para guardar os dados que vem da função *GET*, e de seguida temos a declaração de um *array* com o nome de *receitasArray*.

É utilizado a função do *React* com o nome de *useEffect()*, que diz ao componente que necessita de ser atualizado, e dentro dessa mesma função temos o *GET* que realiza a pesquisa neste caso das receitas.

GET

https://api.spoonacular.com/recipes/complexSearch?apiKey=${apiKey}

Após receber a resposta da *API*, os resultados são colocados num *array* de *id's*, para que sejam iterados por um ciclo for para ir buscar os detalhes das receitas, como as imagens, e o sumário, após isso o *state* das receitas é atualizado com os dados, como pode ser visto na figura seguinte.

GET

https://api.spoonacular.com/recipes/${idArray[i]}/information?apiKey=${apiKey}&includeNutrition=false

```
export default function EspacoUtilizadorReceitas(){

    let apiKey = "c936f6781d2f41c293bf0959d2574d9a"
    const [receitas, setReceitas] = React.useState( initialState: [])
    let receitasAlimentaryPlan = []
    let receitasArray = []
    //https://api.spoonacular.com/recipes/complexSearch?apiKey=f5028b2a056a4853b0a8134bc6b67523
    React.useEffect( effect: () => {
        fetch( input: `https://api.spoonacular.com/recipes/complexSearch?apiKey=${apiKey}` ) Promise<Response>
            .then(res => res.json()) Promise<any>
            .then(data => {
                receitasArray = data.results
                    let idArray = receitasArray.map(receitasAtributes2 => (receitasAtributes2.id))
                    for(let i = 0; i < idArray.length; i++) {
                        fetch( input: `https://api.spoonacular.com/recipes/${idArray[i]}/information?apiKey=${apiKey}
                            .then(res => res.json()) Promise<any>
                            .then(data => {
                                //Adds Elements to an Object
                                receitasArray[i] = {
                                    ...receitasArray[i],
                                    summary:data.summary
                                }

                                if(i === receitasArray.length - 1){
                                    setReceitas(receitasArray)
                                }
                            })
                    }
            })
    }, deps: [])
```

Figura 43 - Método GET Receitas

Seguidamente, podemos verificar como é realizada a renderização do componente, na função *return()*.

Como pode ser visto na figura seguinte, uma vez que as receitas vem através da *API*, é realizado um *loop* com a função *map()* para renderizar os itens das receitas, e isto tem uma razão de ser, uma vez que quando se clica no botão "+" para adicionar a receita ao plano alimentar, de seguida aparece uma notificação durante 3 segundos, apenas queremos que seja disponibilizado para a receita que foi clicada e não todas elas, que era o que acontecia, então para ficar tudo certo e como manda a lei ainda teve de ser criado outro componente que é o item específico.

Algo que também pode ser visto em várias fases do curso do *Scrimba*, que foi onde eu descobri como solucionar o problema em questão.

```
return(
    <section id="menu" className="menu">
        <div className="container">

            <div className="section-title">
                <h2>Receitas</h2>
                <h5>Para introduzir receitas no seu plano alimentar escolha a receita.</h5>
                <h5>De seguida carregue no botão '+' para adicionar ao seu plano alimentar</h5>
            </div>

            {receitas.map(receitasAtributes => {
                return(
                    //How to Convert Function to Stat to be able to use componentDidUpdate
                    //https://www.youtube.com/embed/DPdc5Z-Tf4U?autoplay=1&html5=1&enablejsapi=1
                    <EspacoUtilizadorReceitasItem receitasAtributes={receitasAtributes}/>
                )
            })}
        </div>
    </section>
)
```

Figura 44 - Função return() do Content Receitas

Componente Receitas Itens

Para finalizar o componente das receitas, iremos agora olhar para o item específico, e olhando para a figura seguinte, podemos logo à partida perceber que está diferente de todos os outros ficheiros.

Algo que salta logo à vista é a presença de um construtor, o que indica que estamos na presença de uma classe, e porquê uma classe e não um ficheiro normal como tem sido até agora que funciona tão bem?

Tudo tem uma razão de ser, e a resposta a essa pergunta é para ter acesso á função *componentDidUpdate()*, que tem como objetivo remover o alerta que é dado ao utilizador quando uma receita é adicionada ao plano alimentar.

Mais acima podemos verificar que existe a declaração de um *array*, que guarda os itens quando o botão "+" é clicado e a função *addReceitas()* é ativada, podemos também ver que existe ainda o ficheiro de dados relativo ás receitas que vai guardar os itens com a função *setter*, e que vão ser utilizados nas funções respetivas á criação de planos alimentares, que iremos retomar dai a nada.

```
export default class EspacoUtilizadorReceitasItem extends React.Component{

    constructor(props) {
        super(props);
        this.receitasAlimentaryPlan = []
        this.state = {alert:false}
    }

    addReceitas(receitasAtributes){
        this.receitasAlimentaryPlan.push(receitasAtributes)

        //How to Pass Data from a Component to Another
        //https://stackoverflow.com/questions/42420531/what-is-the-best-way-to-manage-a-users-session-in-react
        ReceitaAlimentaryPlan.setReceitaArray(this.receitasAlimentaryPlan)

        this.setState( state: {alert:true})
    }

    componentDidUpdate(){
        setTimeout( handler: () => this.setState( state: {alert:false}),  timeout: 3000);
    }
```

Figura 45 - Função addReceitas do Componente EspacoUtilizadorReceitasItem

De seguida falta renderizar o componente, que também é feito de maneira diferente, uma vez que é utilizada a função *render()* antes da função *return()*, uma vez que se trata de uma classe, podemos também observar que o acesso aos atributos no construtor é feito com a palavra *this* atrás, e finalmente conseguimos ainda ver a renderização condicional do componente de alerta, que é o motivo de se ter separado mais a estrutura da página em mais 1 componente, como pode ser visualizado na figura seguinte.

```
render(){
    return(
        <div className="row menu-container">
            <div className="menu-item filter-starters">
                <div className="menu-content">
                    <img alt="Imagem Relativa a Receita" src={this.props.receitasAtributes.image}/>
                    <h4 style={{"margin-top":"7%"}}>{this.props.receitasAtributes.title}</h4><span
                </div>
                <div className="menu-content">
                    {/* Set Dangerously HTML https://blog.logrocket.com/using-dangerouslysetinnerht
                    <div dangerouslySetInnerHTML={{__html: this.props.receitasAtributes.summary}} />

                </div>
                {this.state.alert && <Alert id="alert" props={"Receita Adicionada com Sucesso"}/>}
            </div>
        </div>
    )
}
```

Figura 46 - Função render() do Componente EspacoUtilizadorReceitasItem

Tendo o componente dos itens sido acabado, podemos regressar ao componente responsável por fazer a criação do plano alimentar, e mais especificamente á função *submitData()*, em que pode ser visto a declaração de um *array* com o nome de *alimentaryPlan* a ser preenchido com os getters dos ficheiros de dados respetivos aos ingrediente, aos produtos e ás receitas, que irão ser utilizados no método *POST*.

```
function submitData(event) {
    event.preventDefault()

    if(formData.tipoRefeicao === ""){
        formData.tipoRefeicao = 1
    }
    let alimentaryPlanArray = [IngredicentAlimentaryPlan.getIngredienteArrayObject(), ProductAlimentaryP

    //CORS ERROR
    //https://stackoverflow.com/questions/20035101/why-does-my-javascript-code-receive-a-no-access-contr
    //https://web.dev/i18n/en/cross-origin-resource-sharing/
    //https://youtu.be/j1xJaIjEkxg
```

Figura 47 - Função submitData() do Componente EspacoUtilizadorCriarPlanoAlimentar

De seguida passamos para a o método *POST* em si, e este está envolvido em dois ciclos *for*, o primeiro é para percorrer o *array* de planos alimentares e ir buscar os ingredientes, as receitas e os produtos, e o segundo é para ir buscar os elementos que se encontram em cada 1 desses elementos, uma vez que se formos fazer um pequeno almoço não vamos só beber um café, normalmente come-se sempre mais qualquer

coisa e dai ter mais um ciclo for, para se tivermos 2 ingredientes num plano alimentar ele conseguir ir buscar os 2.

O método *POST* leva os dados do formulário, que vem do *state formData*, e por fim quando recebem a resposta da *API*, é gerado um alerta a indicar que o plano alimentar foi criado com sucesso.

POST

https://api.spoonacular.com/mealplanner/${username}/templates?apiKey=87e224df57a14f439d4c4192f67bdb4c&hash=${hash}

```
for (let i = 0; i < alimentaryPlanArray.length; i++) {
    for (let k = 0; k < alimentaryPlanArray[i].length; k++){
        fetch( input: `https://api.spoonacular.com/mealplanner/${username}/templates?apiKey=8
            method: 'POST',
            headers: {
                'Content-Type': 'application/json;charset=utf-8',
                "Access-Control-Allow-Origin": "*"
            },
            body: JSON.stringify( value: {
                "name": formData.nomePlanoAlimentar,
                "items": [
                    {
                        "day": parseInt(formData.dia),
                        "slot": formData.tipoRefeicao,
                        "position": 0,
                        "type": "RECIPE",
                        "value": {
                            "id": 296213,
                            "servings": 1,
                            "title": alimentaryPlanArray[i][k],
                            "imageType": "jpg"
                        }
                    }
                ]
            }), // body data type must match "Content-Type" header
        }).then((response : Response ) => {
            return response.json();
        }).then((parsedData) => {
            console.log("Meal Plan ID = " + parsedData.mealPlan.id)
            setAlert( value: prevState => !prevState)
        })
```

Figura 48 - Método POST Função submitData do Componente EspacoUtilizadorCriarPlanoAlimentar

E por fim sem mais demoras temos a função *return()*, que renderiza o formulário com os seus diversos campos, como pode ser visto na figura seguinte.

```
return(
    <div>
        <Banner />
        <form  className="php-email-form" onSubmit={submitData}>
            <div className="text-center">
                <h1 id="criar-planos-alimentares">Criar Novo Plano Alimentar</h1>
                <h4>Preencha os campos abaixo com as informações necessárias para criar o seu plano alimentar</h4>
            </div>
            <div className="text-center">
                <div className="form-group">
                    <label>Nome Plano Alimentar</label>
                    <input
                        type="nomePlanoAlimentar"
                        name="nomePlanoAlimentar"
                        className="form-control"
                        id="nomePlanoAlimentar"
                        placeholder="Nome Plano Alimentar"
                        onChange={changeFormData}
                        value={formData.nomePlanoAlimentar}
                    />
                </div>
```

Figura 49 – Inicio da Função return() do Componente EspacoUtilizadorCriarPlanoAlimentar

Para não ficar muito extenso saltei logo para o final da função de *return()*, uma vez que no meio tem os campos que são bastante similares ao que aparece na figura acima.

Para finalizar, temos as renderizações condicionais dos elementos para visualizar as receitas, os produtos e os ingredientes, quando o tipo é selecionado.

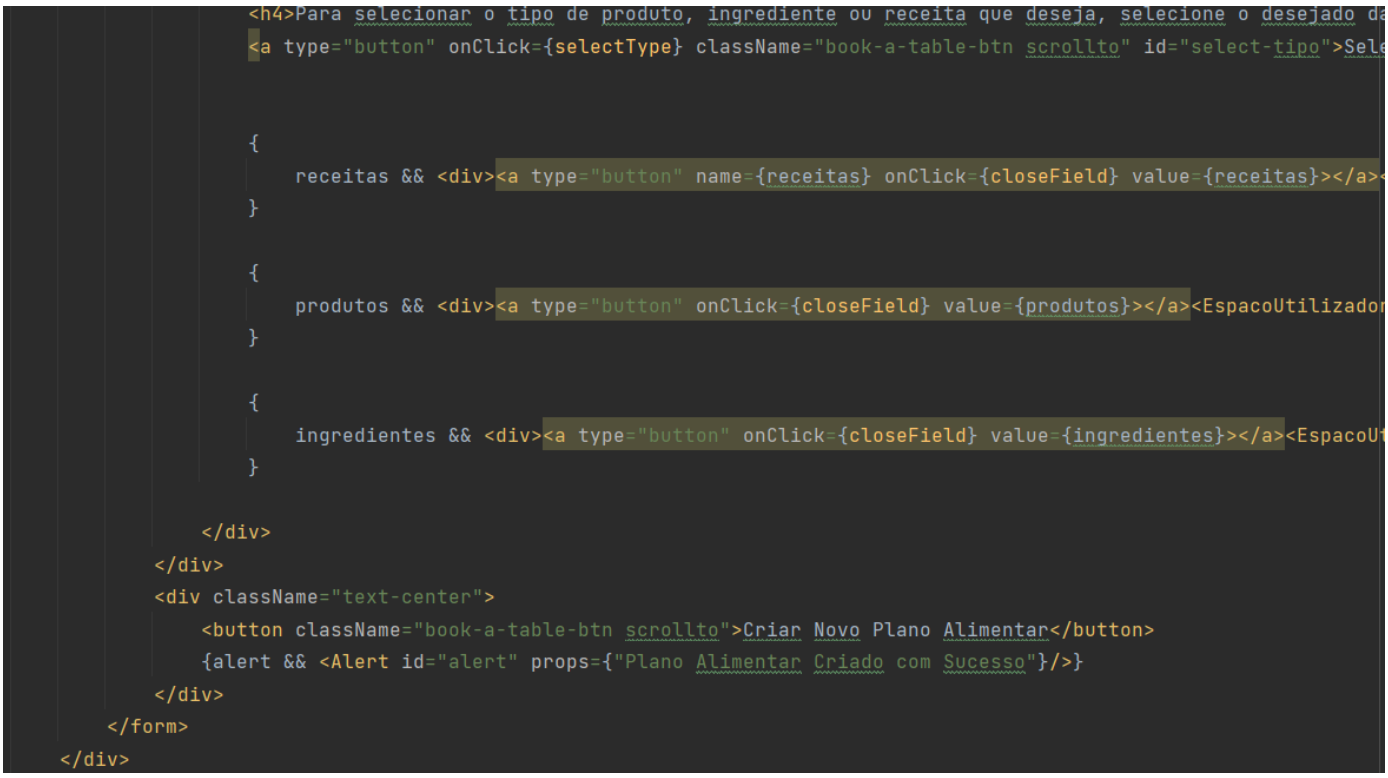

Figura 50 - Fim da Função return() do Componente EspacoUtilizadorCriarPlanoAlimentar

Visualização de Planos Alimentares e seus Detalhes

Por fim chegamos a 2º tarefa do meu sistema *web*, que é a visualização dos planos alimentares que foram criados na secção acima, como podemos verificar na figura seguinte.

```
export default function EspacoUtilizador(){

    return(
    <div>
        <EspacoUtilizadorCriarPlanoAlimentar />
        <EspacoUtilizadorVisualizarPlanosAlimentares />
    </div>
    )
}
```

Figura 51 - Estrutura da Página do Espaço do Utilizador

Como pode ser visto na figura abaixo, para ser possível visualizar a lista de planos alimentares, temos de utilizar novamente um state, que neste caso tem o nome de

planos, e de seguida através da função *React useEffect()* que também é utilizada na página inicial, é renderizada a lista dos planos do utilizador específico através de um método *GET*.

GET

https://api.spoonacular.com/mealplanner/${username}/templates?apiKey=${apiKey}&hash=${hash}

Uma vez feito o *GET* e tendo chegado a resposta, o *state* planos pode ser atualizado através da função *setPlanos()*, com os dados vindos da resposta, como pode ser visto na figura seguinte.

```
export default function EspacoUtilizadorVisualizarPlanosAlimentares(){

    const [planos, setPlanos] = React.useState( initialState: [])
    let apiKey = "12c5cff8e3444d849402a3d0c65bc985"

    //let username = Utilizador.getUtilizadorEmail()
    //let username = "admin1"
    let username = "teste2"
    //let hash = Utilizador.getUtilizadorHash()
    let hash = "089f099d4782ca799d117074f9709514a4a3059b"
    //let hash = "9f0397217c22828795be05bbbae07c0e8b6637cc"

    React.useEffect( effect: () => {
        fetch( input: `https://api.spoonacular.com/mealplanner/${username}/templates?a
            .then(res => res.json()) Promise<any>
            .then(data => {
                setPlanos(data.templates)
            })
    }, deps: [])
```

Figura 52 - Função useEffect do Componente EspacoUtilizadorVisualizarPlanosAlimentares

Para finalizar este componente temos ainda a função *return()*, que mostra as informações acerca do componente visualizar planos alimentares criados pelo utilizador, através de um *loop* com a função *map()* pelos elementos que se encontram no *state* planos , como pode ser visto na figura seguinte.

```
return(
    <section id="menu" className="menu">
        <div className="container">

            <div className="section-title">
                <h2 id="visualizar-planos_alimentares">Visualizar <span>Planos Alimentares</span></h2>
                <h4>Nesta secção pode visualizar os planos alimentares criados</h4>
                <h4>Para isso basta clicar em cima do nome do plano alimentar ao qual você pretende verificar os detalhes</h4>
            </div>

            <div className="row menu-container">

                {planos.map(planosAtributes => {
                    return (
                        <EspacoUtilizadorVisualizarDetalhesPlanosAlimentares planosAtributes={planosAtributes}/>
                    )
                })}
            </div>
        </div>
    </section>
)
```

Figura 53 - Função return() do Componente EspacoUtilizadorVisualizarPlanosAlimentares

E como não poderia faltar, de que é que serve visualizar o nome do plano alimentar sem poder ver os seus detalhes? Isso é igual a nada, logo temos um componente específico para executar essa tarefa, para não acontecer quando se carregar num elemento para ver os seus detalhes mostrar os detalhes de todos os elementos na lista.

Visualizar Plano Alimentar Item

Como já foi dito anteriormente, este componente vem a resolver um *bug* que existia durante a execução do *website*.

Como pode ser visto na figura seguinte, temos 2 variáveis *state*, uma dela vai ser utilizada para indicar se é possível mostrar os detalhes do plano alimentar selecionado, e a outra é para guardar esses mesmo detalhes.

```
export default function EspacoUtilizadorVisualizarDetalhesPlanosAlimentares(props){

    //Renderizar Detalhes 1 de Cada Vez
    //https://scrimba.com/learn/learnreact/boxes-challenge-part-31-local-state-co0264ad6929a556e38a6a10f
    const [plansDisplayInfo, setPlansDisplayInfo] = React.useState( initialState: false)
    const [plansInfo, setPlansInfo] = React.useState( initialState: [])
    let apiKey = "12c5cff8e3444d849402a3d0c65bc985"
    let username = "teste2"
    let hash = "089f099d4782ca799d117074f9709514a4a3059b"
```

Figura 54 - Definição de state's no Componente EspacoUtilizadorVisualizarDetalhesPlanosAlimentares

De seguida temos a função *getPlansDetails()*, que tem como objetivo ir buscar os dados do plano alimentar selecionado através de um método *GET*.

GET

https://api.spoonacular.com/mealplanner/${username}/templates/${id}?apiKey=${apiKey}&hash=${hash}

Após receber a resposta da *API*, o state que vai mostrar a informação é passado de *false* para *true*, e é feita a verificação se o campo *name* está *undefined*, pois tem a ver com os atributos que um plano com uma receita tem que os restantes não têm, e de seguida o *state* que contem a informação dos planos é por fim atualizado.

```
function getPlansDetails(event, id){
    //https://api.spoonacular.com/mealplanner/{username}/templates/{id}
    fetch( input: `https://api.spoonacular.com/mealplanner/${username}/templates/${id}?apiKey=${apiKey}
        .then(res => res.json()) Promise<any>
        .then(data => {
            console.log("LENGHT = " + data.days.length)
            console.log("Plans ID INFO = " + data.id)
        if(data.days.length !== undefined || data.days.length > 0){
            for (let i = 0; i < data.days.length; i++){
                for (let j = 0; j < data.days[i].items.length; j++) {

                    if(data.days[i].items[j].value.title.name === undefined){
                        setPlansDisplayInfo( value: prevData => !prevData)
                        let dataArray = [data.days[i].items[0].value.title.title, data.days[i].items[
                        if(data.days[i].items[0].value.title.summary === undefined){
                            data.days[i].items[0].value.title.summary = ""
                        }
                        setPlansInfo(dataArray)
                    } else {
                        setPlansDisplayInfo( value: prevData => !prevData)
                        let dataArray = [data.days[i].items[0].value.title.name, data.days[i].items[0
                        if(data.days[i].items[0].value.title.consistencia === undefined){
                            data.days[i].items[0].value.title.consistencia = ""
                        }
                        setPlansInfo(dataArray)
                    }
                }
            }
        }
        })
```

Figura 55 - Função getPlansDetails() no Componente EspacoUtilizadorVisualizarDetalhesPlanosAlimentares

E por fim temos a função *return()* que tem uma *tag <a>* que tem como função *onClick*, e chama a função que acabamos de ver, e por fim temos a renderização condicional dos detalhes do plano alimentar.

```
return(
    <div className="menu-item filter-starters">
        <div className="menu-content">
            {
                //How to pass data to a function as argument in react
                //https://bobbyhadz.com/blog/react-onclick-pass-event-and-parameter
            }
            <a onClick={event => getPlansDetails(event, props.planosAtributes.id)}>{props.planosAtributes.name}</a>
        </div>
        <div className="menu-ingredients">
            {/* Renders Image Conditionally*/}
            {plansDisplayInfo && <div><img alt="Imagem Relativa ao Plano Alimentar" src={plansInfo[3].includes( value:
            {plansDisplayInfo && <div><h4>Nome: </h4><p>{plansInfo[0]}</p></div>}
            {plansDisplayInfo && <div><h4>Doses: </h4><p>{plansInfo[1]}</p></div>}
        </div>
    </div>
)
```

Figura 56 - Função return() no Componente EspacoUtilizadorVisualizarDetalhesPlanosAlimentares

A *tag <img>* tem ainda um *if* ternário, pois as imagens possuem que vem nas receitas vem com o caminho completo, mas as imagens que vem de um ingrediente ou de um produto não, tendo que se adicionar o caminho, como pode ser visto na figura seguinte.

```
src={plansInfo[3].includes( value: "http") ? plansInfo[3] : "https://spoonacular.com/cdn/ingredients_100x100/"+plansInfo[3]}
```

Figura 57 - Tag <img> com If Ternário

Avaliação das Interfaces / Testes com Utilizadores

Nesta secção, iremos realizar a avaliação das interfaces com recurso aos testes com utilizadores.

Os vídeos estão disponibilizados numa pasta com o nome de "Testes com Utilizadores".

GUIÃO

Para começar a testar o website, irei agradecer a presença do utilizador em questão.

De seguida, irei pedir ao utilizador para observar a página principal, e depois que se registe na aplicação

Com esse passo terminado, irei-lhe indicar para realizar o login na aplicação.

Agora já no Espaço do Utilizador, peço ao utilizador para realizar a criação de um novo plano alimentar

Após o utilizador ter criado o plano alimentar digo-lhe para retornar á página inicial, e para entrar no Espaço do Utilizador novamente para que possa ver o plano alimentar que fora criado.

APRESENTAÇÃO DE RESULTADOS

Após terem sido feitos os 5 testes necessários, cheguei a esta tabela que se apresenta de seguida.

Utilizador	Tempo Tarefa 1	Nº de Clicks Tarefa 1	Tempo Tarefa 2	Nº de Clicks Tarefa 2	Respostas Corretas
1	02:03	19	00:23	3	100%
2	02:17	20	00:42	3	100%
3	02:00	21	00:29	3	100%
4	02:19	19	00:40	2	100%
5	02:30	24	00:30	3	100%
Min	02:00	19	00:23	2	100%
Média	02:13	20,6	00:32	2,8	100%
Máximo	02:30	24	00:42	3	100%
Desvio Padrão	0,008553233	2,073644135	0,005542519	0,447213595	0

Figura 58 - Tabela de Apresentação de Resultados

De seguida, temos os vídeos dos testes com utilizadores

Teste 1:

https://drive.google.com/file/d/1kBg3_ThibxoHcTwS5Tij5C5H7WryWfHt/view?usp=sharing

Teste 2:

https://drive.google.com/file/d/1LvL9Sj6Zmmbve78GoJRhoBxftmmcfW1A/view?usp=sharing

Teste 3:

https://drive.google.com/file/d/1LvL9Sj6Zmmbve78GoJRhoBxftmmcfW1A/view?usp=sharing

Teste 4:

https://drive.google.com/file/d/1sDVCY3UaKcIeCWAx9aaSjoJZW3bKIwLy/view?usp=sharing

Teste 5:

https://drive.google.com/file/d/1YNUOLW0Zb404Sk4mwDeIWA6wQihNU7OV/view?usp=sharing

Foram ainda levantadas algumas questões durante os testes em que alguns utilizadores não conseguiram identificar bem se se encontravam na página inicial ou no espaço do utilizador, e é algo que deve ser resolvido.

Conclusões Finais

Posso concluir que este projeto foi bastante desafiante pois encontrei alguns problemas ao longo do caminho, uns por culpa da *API* não ter sido bem desenvolvida como o de não conseguir adicionar nada a um plano alimentar já criado, quando eles têm lá o *endpoint*, mas não faz nada, eu ainda cheguei a mandar um *email* na esperança de alguém me ajudar como se verifica na figura seguinte, mas não obtive resposta.

Hello Good Morning.

I'm using your API to do some college work and I'm finding difficult to adding Ingredients, Products or Recepies to an already created User Meal Plan, I have been testing the endpoints with Postman and I didn't had any success.

Is there any problem with the endpoint or is it just me doing it wrong?
I'm saying this because when I want to add an Ingredient to a User Meal Plan I should need to provide the user's meal plan id with in the endpoint does not mention anything about it.
There is also good to refer that I'm using the syntax of the correct endpoint.

Thanks in Advance Vasco Gomes

Figura 59 - Email Acerca de Problema no Endpoint da API

De seguida tive problemas com um dos métodos *POST*, e a resolução foi utilizar a extensão do *Google Chrome Moesif Origin & CORS Changer*, e ativá-la durante a execução do trabalho.

Posso ainda dizer que trabalhar com componentes facilita muito a vida de um programador, mas foi algo difícil de lhe apanhar o jeito só na parte final do projeto é que comecei a perceber exatamente como é que se processava, porque quanto mais se faz mais se aprende.

Para terminar posso dizer que tentei resolver os problemas da melhor forma possível de modo a ter um produto final com qualidade.

Webgrafia

[1] "Google - What is a Hook in React," [Online]. Available: https://www.google.com/search?q=what+is+a+hook+in+react&oq=what+is+a+Hook+in+rea&aqs=chrome.1.69i57j0i512j0i22i30l8.6951j0j7&sourceid=chrome&ie=UTF-8. [Acedido em 27 Junho 2022].

[2] "Reactjs - Hooks Effect," [Online]. Available: https://reactjs.org/docs/hooks-effect.html. [Acedido em 27 Junho 2022].

[3] "Spoonacular - Gerar Plano Alimentar," [Online]. Available: https://api.spoonacular.com/mealplanner/generate. [Acedido em 27 Junho 2022].

[4] "Spoonacular - Criar Conta de Utilizador," [Online]. Available: https://api.spoonacular.com/users/connect. [Acedido em 27 Junho 2022].

[5] "Spoonacular - Pesquisar Receitas," [Online]. Available: https://api.spoonacular.com/recipes/complexSearch. [Acedido em 27 Junho 2022].

[6] "Pesquisar Detalhes das Receitas," [Online]. Available: https://api.spoonacular.com/recipes/${idArray[i]}/information. [Acedido em 27 Junho 2022].

[7] "Pesquisa de Ingredientes," [Online]. Available: https://api.spoonacular.com/food/ingredients/search. [Acedido em 27 Junho 2022].

[8] "Spoonacular - Pesquisar Detalhes de Ingredientes," [Online]. Available: https://api.spoonacular.com/food/ingredients/9040/information?. [Acedido em 27 Junho 2022].

[9] "Spoonacular - Pesquisa de Produtos," [Online]. Available: https://api.spoonacular.com/food/products/search. [Acedido em 27 Junho 2022].

[10] "Spoonacular - Pesquisa de Detalhes de Produtos," [Online]. Available: https://api.spoonacular.com/food/products/196488. [Acedido em 27 Junho 2022].

[11] "Spoonacular - Criar Plano Alimentar," [Online]. Available: https://api.spoonacular.com/mealplanner/vasco/templates. [Acedido em 27 Junho 2022].

[12] "Spoonacular Mostrar Templates Criados Pelo Utilizador," [Online]. Available: https://api.spoonacular.com/mealplanner/vasco/templates. [Acedido em 27 Junho 2022].

[13] "Spoonacular - Mostrar Detalhes dos Templates Criados Pelos Utilizadores," [Online]. Available: https://api.spoonacular.com/mealplanner/vasco/templates/5171. [Acedido em 27 Junho 2022].