**Autenticação com Cognito para Iniciantes em React.**

Neste artigo irei explicar porque aqui na Site Blindado optamos pelo serviço Cognito da Amazon Web Services e mostrar como fazer autenticação na sua própria aplicação React.  
Tudo que irei explicar aqui, basicamente também se aplica ao React-Native, com a pequena diferença na biblioteca que iremos utilizar (aws-amplify-react vs aws-amplify-react-native).

O objetivo final do artigo é criar um app com autenticação, incluindo MFA (two-factor), mostrando diversas formas de realizar através do Cognito e a lib Aws Amplify.

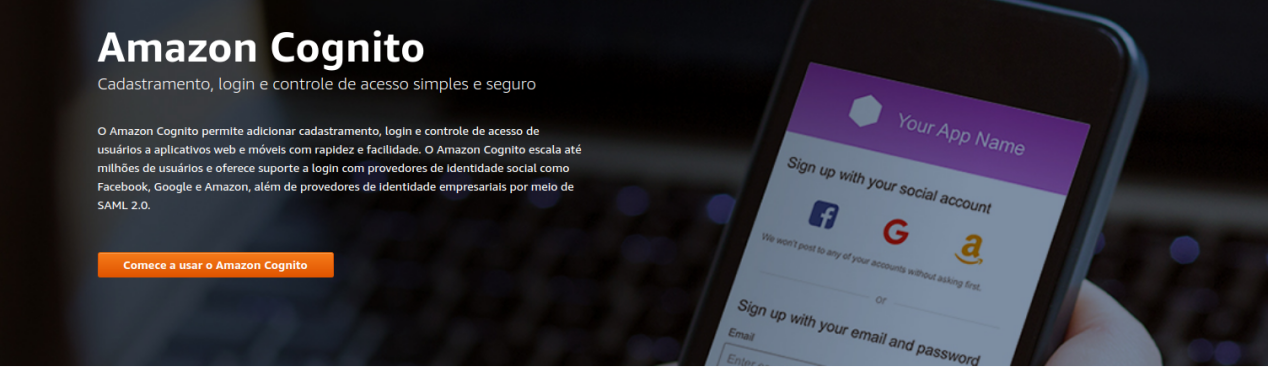
Para nós Devs autenticação é sempre uma dor de cabeça.  
Pensar em estratégias de segurança, banco, criptografia, validações e muitos pequenos detalhes que acabam tirando o nosso sono.  
Além disso temos a LGPD( Lei Geral de Proteção de Dados Pessoais ) batendo na porta em 2020.

A segurança de dados é um assunto tão delicado que pode arriscar até 2% do faturamento bruto da empresa em multas, isso é um risco muito alto para nós desenvolvedores.

Pensando em simplificar isso a AWS (Amazon Web Services) lançou o Cognito, um serviço que permite fazer o gerenciamento de usuários, senhas, dados cadastrais e autenticação de usuários.

O Tier gratuíto contempla 50.000 usuários ativos ou MAUs (como é chamado pelo AWS)

Se um usuário faz login durante o mês, ele é considerado um MAU. Caso contrário ele não é cobrado ou descontado da franquia.



Alguns dos recursos mais interessantes que podemos destacar desta aplicação são:

* Diretório de usuários seguro e escalável
* Federação de identidades sociais e empresariais
* Autenticação baseada em padrões
* Segurança para aplicativos e usuários
* Controle de acesso para recursos da AWS
* Fácil integração com os aplicativos

Mas vamos ao que interessa.

**Pré Requisitos**

* Node.Js
* Yarn
* Create React App - Vou usar o boilerplate como exemplo de autenticação, mas você pode usar da forma como quiser
* Aws-Amplify
* Aws-Amplify-react - Vou mostrar como usar as páginas da lib e como reescreve-las ou criar as suas.

**Começando com create-react-app**

Vamos começar, antes de mais nada criaremos nosso app.

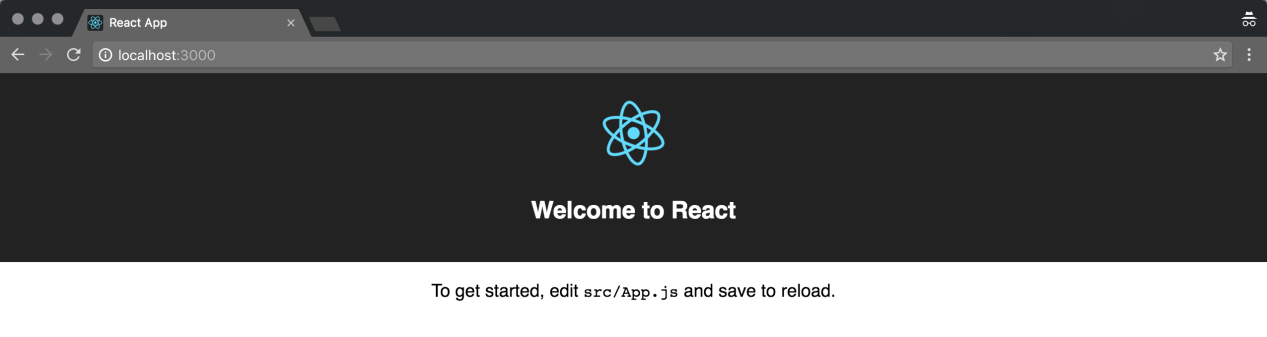
npm install -g create-react-app // Para instalar o create-react-app

npx create-react-app artigo-cognito // Cria o projeto

cd artigo-cognito // Entra na pasta do projeto

npm start // Inicia o projeto

Se tudo der certo você verá uma tela como esta:



**Cognito**

Vamos ver aqui como configurar o Cognito de forma automatizada, via CLI do amplify.

O aws-amplify é uma biblioteca criada pela amazon para facilitar o devops de aplicações da AWS.  
O que isso significa?   
Se você está desenvolvendo um sistema e precisa alocar recursos na AWS, muitas vezes não queremos ficar editando a configuração de todos os recursos na mão.

Então o Amplify próvem de uma ferramenta que faz alocação destes recursos através de scripts.

Ele também provem recursos de acesso, clientes para produtos AWS, cliente GraphQL e bibliotecas de componentes para diversos frameworks como React, React Native, Angular, Ionic e Vue. O amplify éuma ferramenta bastante poderosa.

No nosso projeto vamos usar o Amplify apenas como cliente de acesso ao Cognito.

Aqui na Site Blindado, já utilizamos o Terraform (https://www.terraform.io/) como ferramenta de Devops.

Vamos lá:

npm install aws-amplify // Instala o amplify e a CLI que utilizaremos

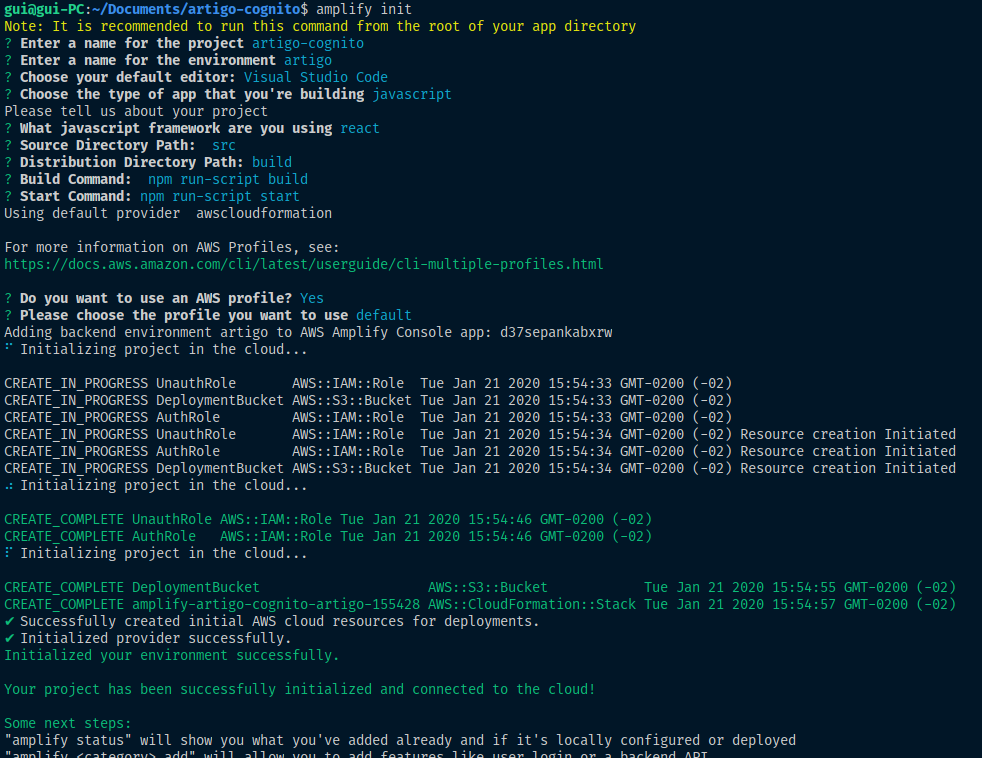
amplify init // Inicia nosso projeto

Logo no inicio, a página de login da AWS será aberta para você fazer login.

Logo você será guiado a criar um usuário, e pegar a access key e a secret key dele.

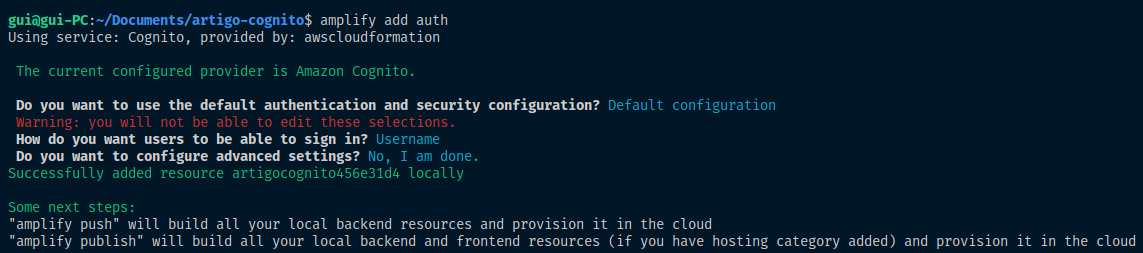
O CLI fará algumas perguntas, como região, nome para usuário amplify, nome do projeto, um nome para o ambiente de desenvolvimento, o editor usado e etc

Basta seguir o passo-a-passo, se tiver alguma dúvida, basta seguir o video: https://www.youtube.com/watch?v=fWbM5DLh25U



Em seguida vamos iniciar o projeto de autenticação.

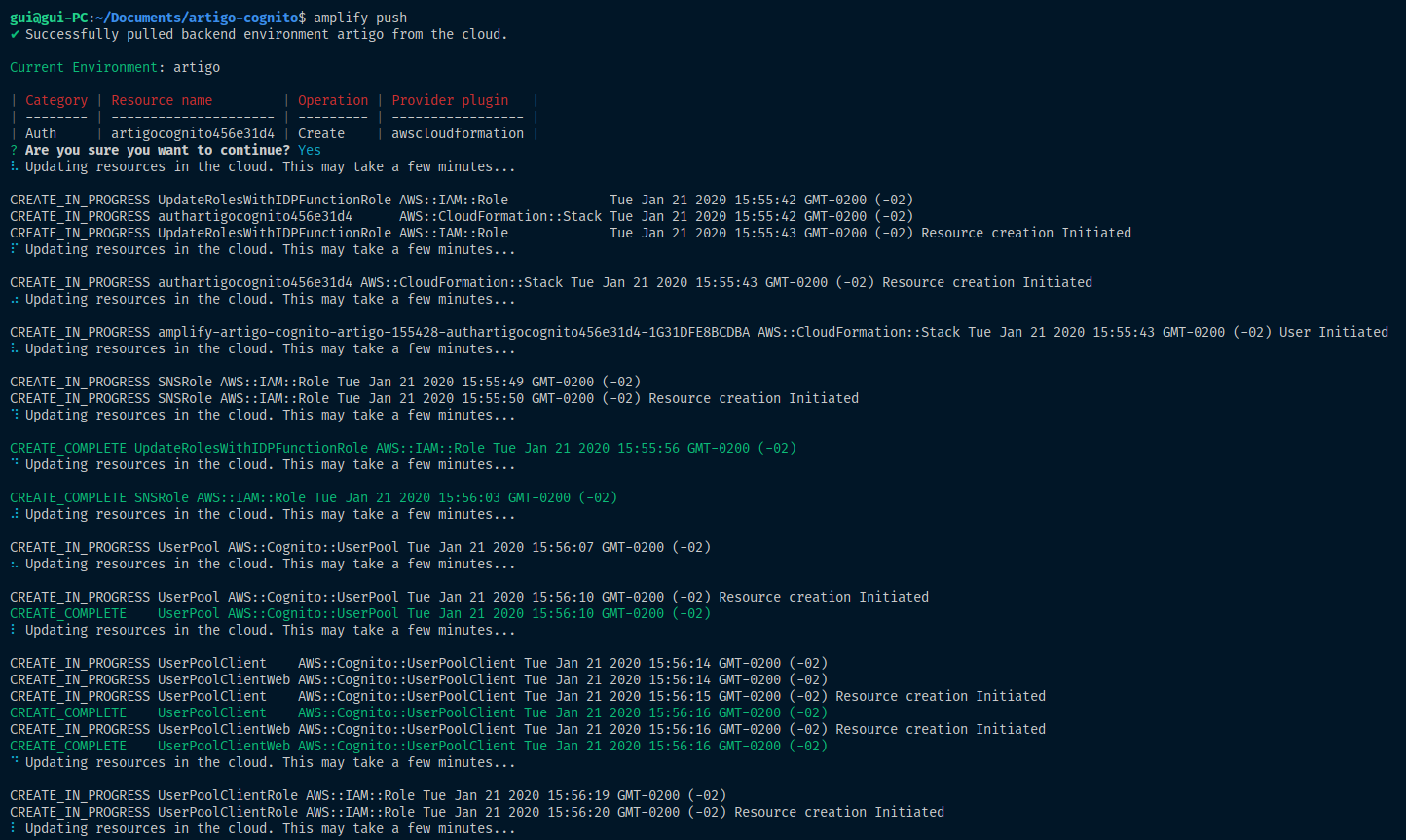
amplify add auth//Inicia o projeto de autenticação



Usaremos a configuração padrão, posteriormente faremos algumas alterações diretamente no painel do cognito para testes.

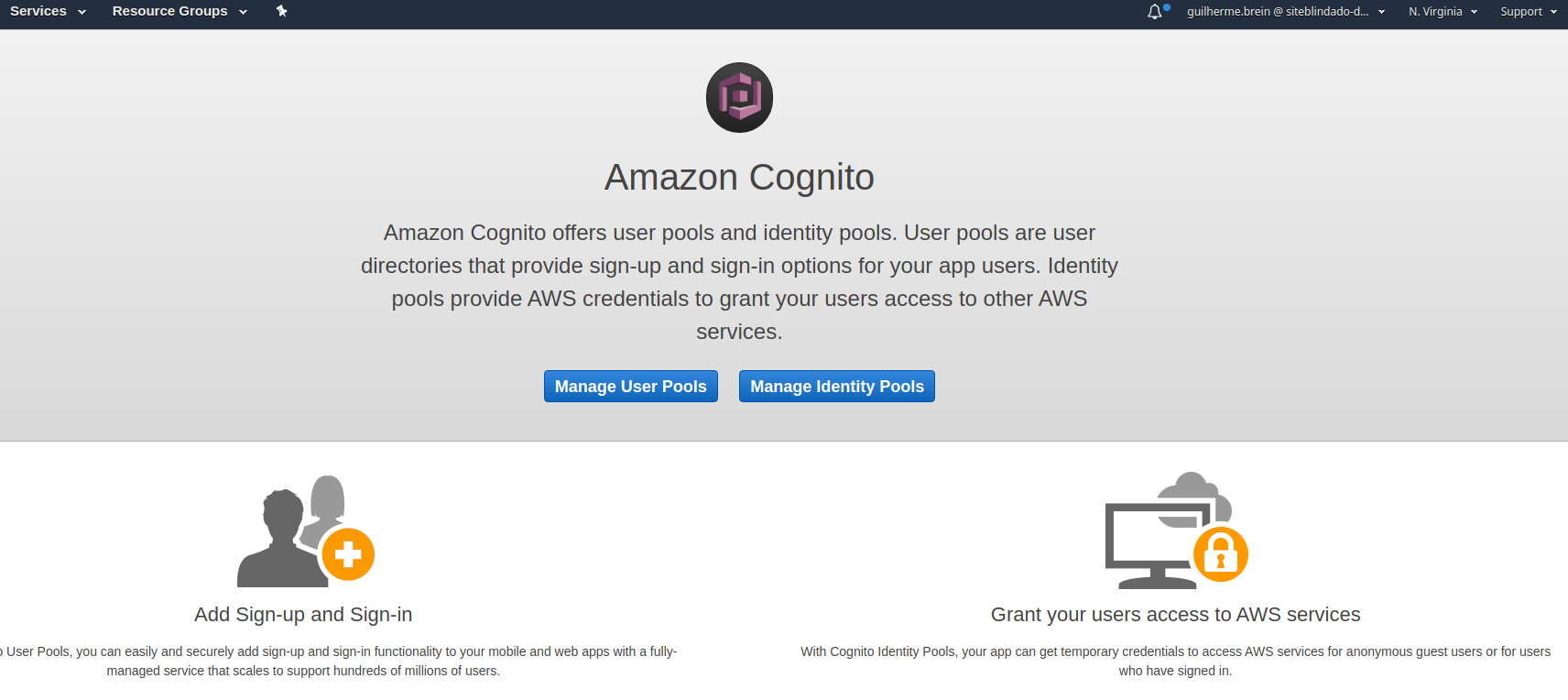
Agora vamos dar um comando que publica via Cloudformation toda infraestrutura que foi projetada pelo amplify

amplify push // publica via cloudformation a infra requisitada



Não entrarei muito em detalhes sobre o funcionamento da CLI do amplify, vamos aqui apenas utiliza-la. Talvez futuramente posso fazer um artigo sobre a CLI.

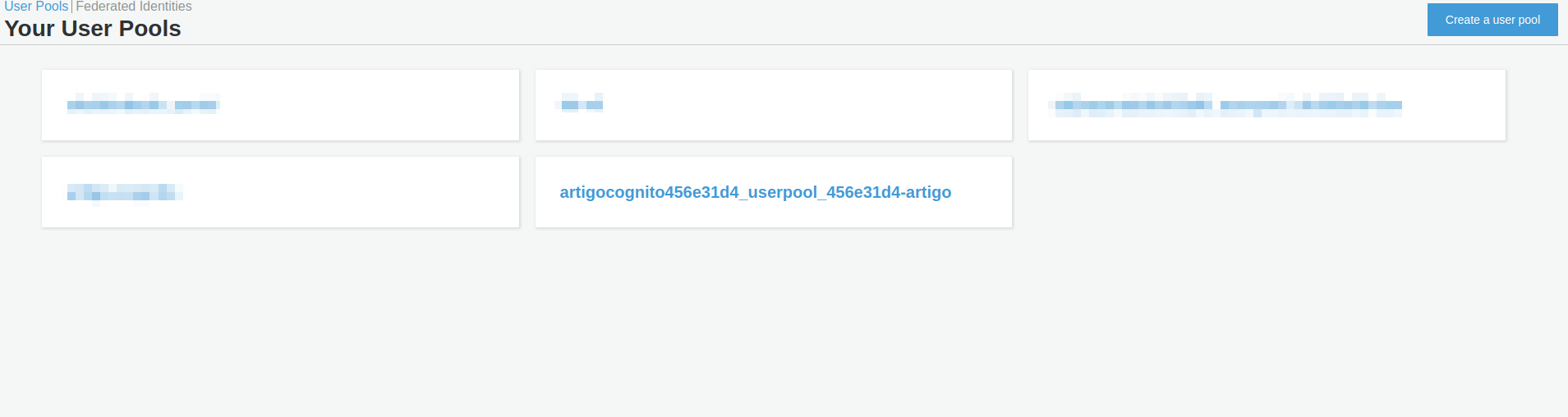
Agora vamos para o painel da AWS e entraremos no serviço do Cognito.



Neste painel temos duas opções.   
  
- Manage User Pools - Ferramenta para criar as User Pools, ou seja, diretórios de usuários com as informações e diretrizes destes. Como um antigo “Active Directory” provido pela amazon.

- Manage Identity Pools - É uma ferramenta de gestão para usuários convidados, logados por ferramentas externas e usuários que receberam um token de identificação.

Clique no botão **Manage User Pools**

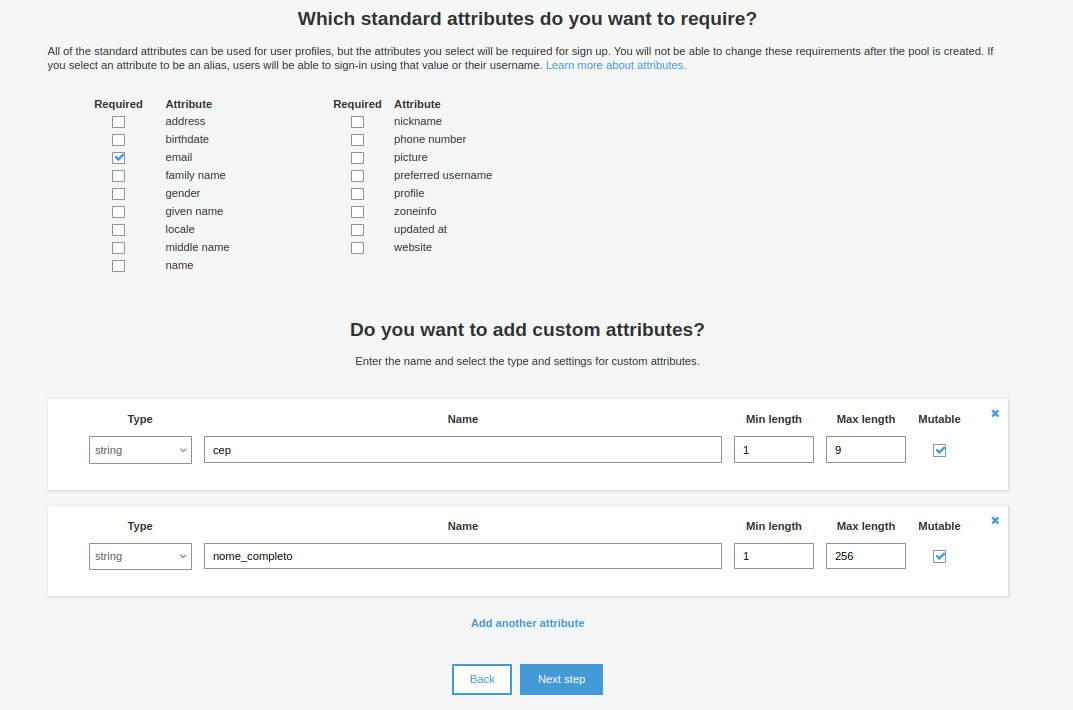


Temos então um **User Pool** criado pelo amplify, entraremos nele

Em seguida vamos a tela de Attributes, no canto esquerdo da tela.

Aqui podemos escolher algumas configurações de nome de usuário, no momento não iremos altera-las.

Mas adicionaremos 2 campos custom na parte de baixo da tela. Um chamado de **cep** e um de **nome\_completo**  
Isso é importante, pois mais pra frente será importante saber como lidar com estes campos customizados.

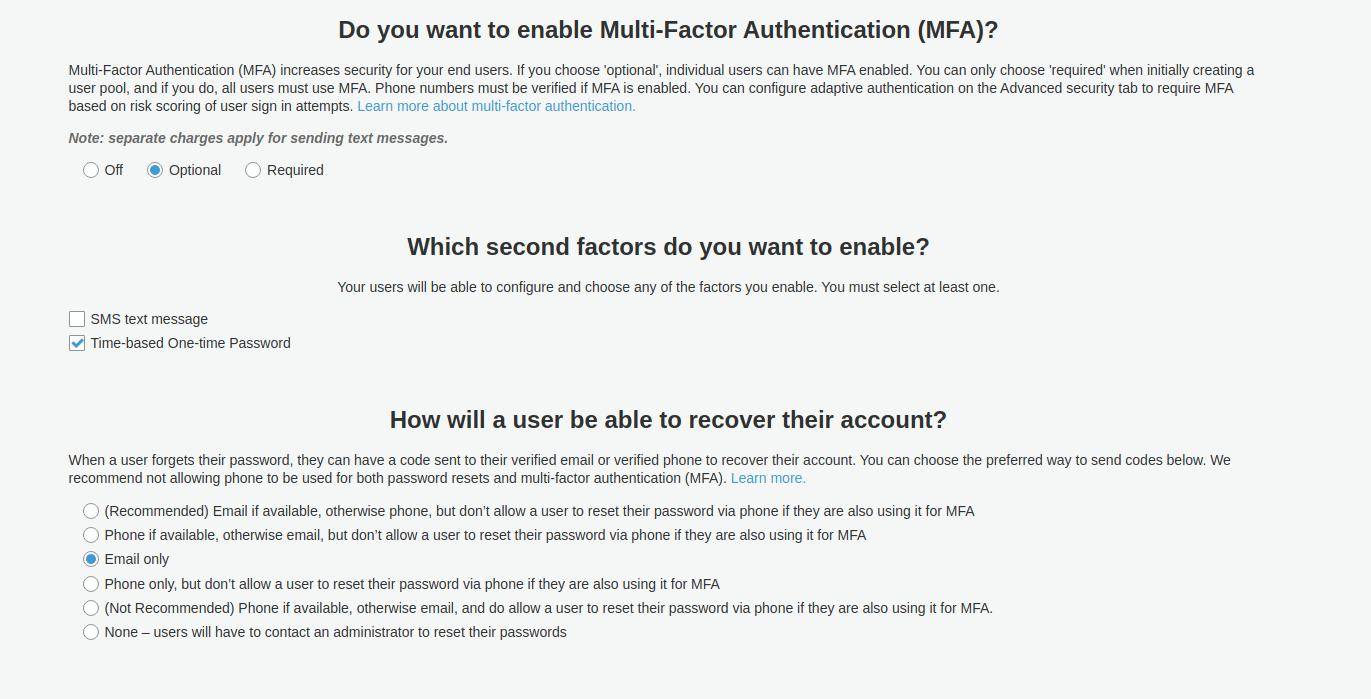


Em seguida salvamos e vamos para a aba MFA and Verifications no menu.

Vamos colocar o MFA (Autenticação de dois fatores) como opcional.

Também vamos marcar Time-Based One-Time Password como fator de autenticação. Isso significa que o usuário deverá utilizar um aplicativo que forneça um token no celular por ex: google authenticator, authy etc...

Por fim marquei a opção de email para recuperação de senha.



Um pequeno adendo é para aba Triggers. É possível disparar funções Lambda (microsserviços) a partir da autenticação.  
No momento não entrarei neste assunto, mas é um recurso fantastico para aplicações.

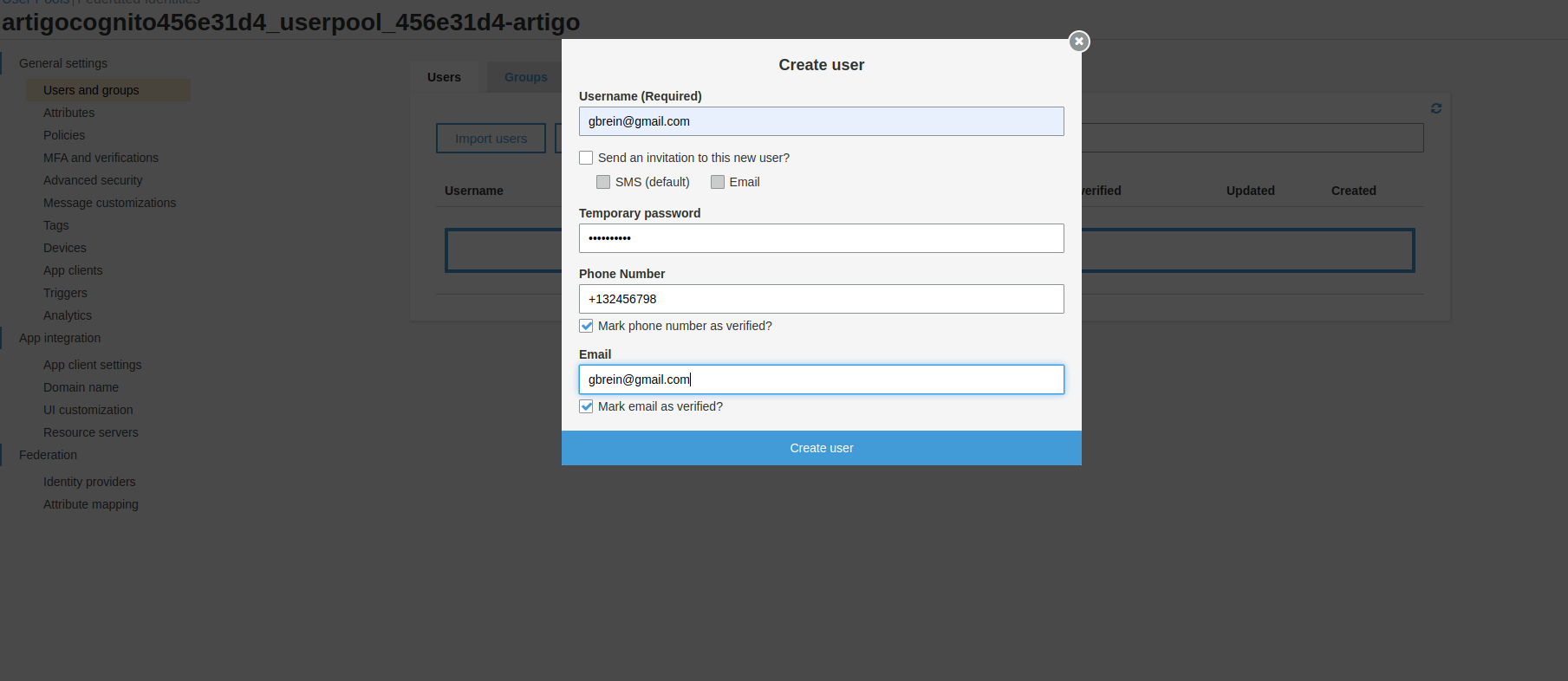
Pronto configuramos nossa User pool.

**Criando um usuário para Teste**

Vamos criar um usuário e ativa-lo direto no cognito, para isso vamos na aba **Users and groups**.

Agora entramos em Create User, colocamos os dados, desmarcamos “Send an invitationto this new user?” e marcamos o telefone e o email como verificados

Na primeira vez que realizarmos login será solicitado a troca de senha

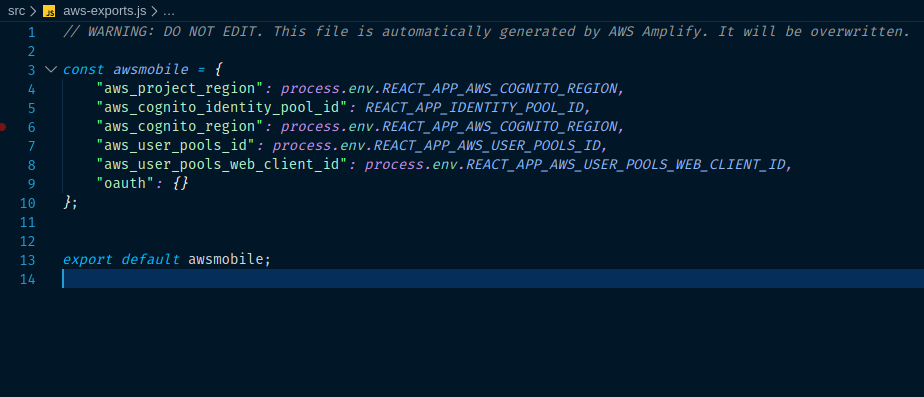


**Configurando Amplify-React**

Vamos começar instalando o aws-amplify, aws-sdk e o cliente do aws-amplify-react:  
npm install aws-amplify-react npm install aws-sdk

Agora vamos ao que interessa e colocar a mão na massa pra valer.

Dentro do arquivo **aws-exports.js** vamos colocar as chaves de valores em um arquivo .env para proteje-los (não esqueça de adicionar no .gitgnore):



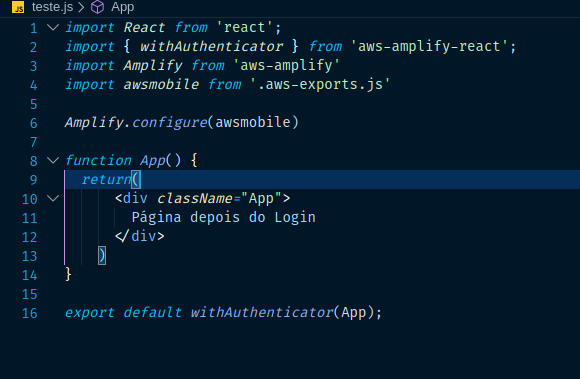
Caso, assim como no meu projeto esteja utilizando o create-react-app, utilize o prefixo REACT\_APP\_ antes do nome da variavel no arquivo .env.

1a forma de utilizar o **Amplify - Via WithAuthorizator:**

Abra o arquivo App.js, em seguida vamos importar o método **withAuthenticator**  da biblioteca aws-amplify-react e o Amplify do aws-amplify.

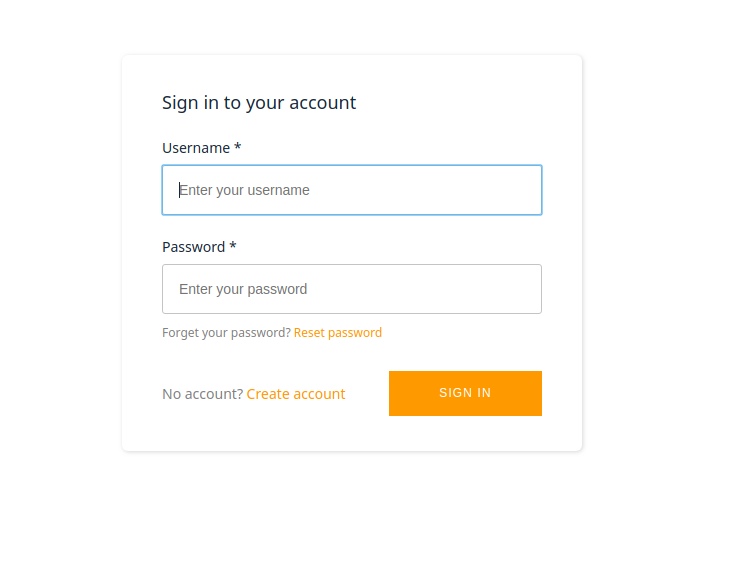
Importamos objeto **awsmobile** do arquivo **aws-export.js**.

Por fim vamos vamos aplicar o WithAuthorizator passando o APP como parametro direto na exportação



Então salvamos o arquivo e iniciamos com **yarn start** (se você mexeu no .env e já estiver com servidor rodando, precisa parar e reiniciar o serviço).

O resultado:



Pronto, temos uma página de login totalmente funcional.

Porém temos alguns problemas. A página roda com layout próprio da **aws-amplify-react**, uma **Component-UI** própria do **amplify**, na maioria dos casos queremos personalizar.

Para isso temos algumas opções, vou tentar abordar as seguintes:

- Reescrever o render das paginas disponiveis do **amplify** aproveitando os métodos e testes já existentes e apenas reescrevendo os componentes renderizados.

- Criar nossas próprias paginas de login, utilizando de métodos próprios.

- Customizar as paginas do **amplify** através de personalização do **theme**

Antes de seguirmos em frente, sugiro criar uma conta para testarmos futuramente.

Basta clickar em Create account.  
Se você preferir pode criar diretamente no **Cognito**.

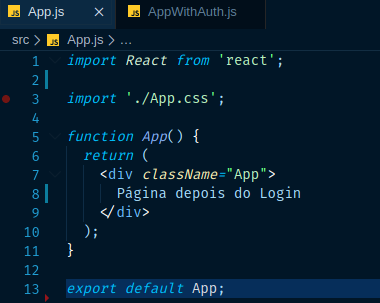
**Utilizando o Authenticator ao invés do WithAuthenticator:**

Antes de customizarmos nossos componentes vamos remover o método **WithAuthenticator** e substituirmos para um componente React chamado **Authenticator** também do **aws-amplify-react**.

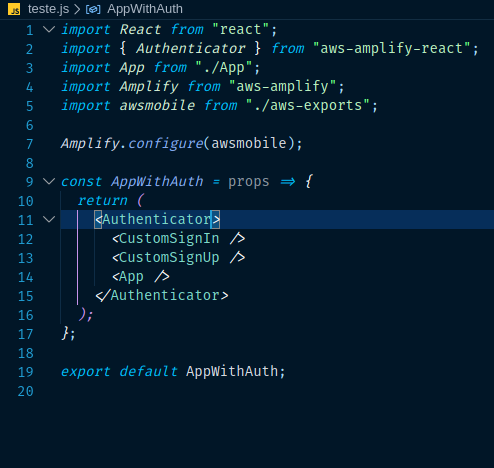
A diferença entre os dois é que o **Authenticator** permite muitas customizações, enquanto o outro é mais restrito.

Para isso vamos criar um arquivo chamado **AppWithAuth** dentro da pasta **src**.

Antes de editar o arquivo vamos no arquivo **App.js** e remover o amplify deste arquivo.

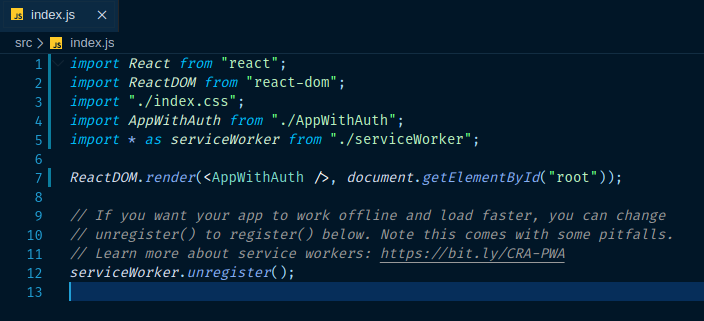


Agora vamos editar o arquivo **AppWithAuth.js.**

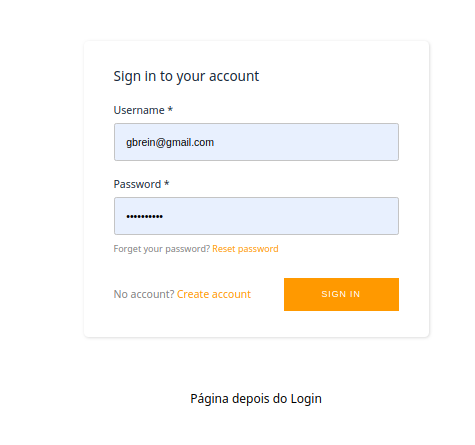


Neste arquivo, basicamente, envolvemos o componente **App** com o componente **Authenticator** do **Amplify**, ele será responsável por transmitir/sobreporas props e rotas de autenticação antes e para o **App**.

Por fim vamos editar o arquivo **index.js**, alterando o caminho nativo do primeiro componente do React, do **App.js** para **AppWithAuth.js**



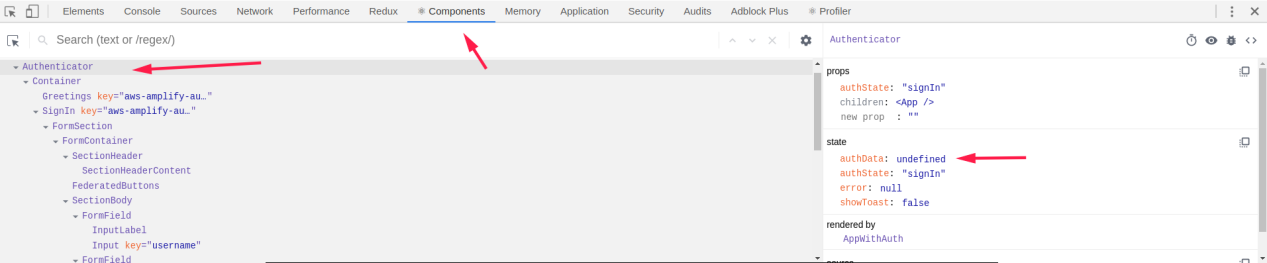
Salvando teremos como resultado



Estamos exibindo a mesma pagina de Login de antes, porém o componente **App** que deveria ser exibido pós login ainda está sendo exibido.

Agora vamos resolver este problema de uma forma simple.

Utilizando a extensão do chrome React Developer Tools vamos dar um inspecionar na página e olhar a aba Componentes:

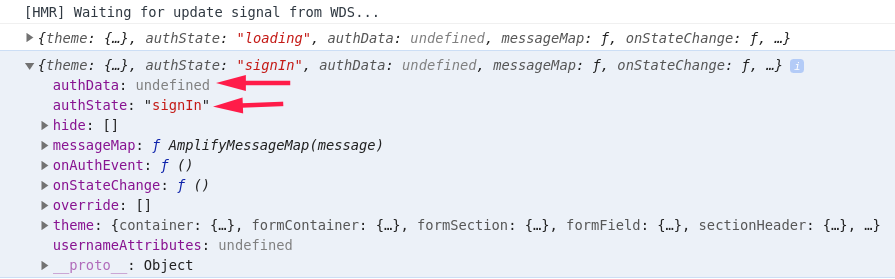


Vemos que o componente **Authenticator**, envolve todo o restante do código, e como estado temos **authState** e **authData**.

O **authState** funciona como um guia para nossas rotas. Dependendo deste estado o Authenticator exibirá uma determinada página ou outra.

O **authData** é um estado que apenas é preenchido quando o login é efetuado com sucesso. É o conteúdo do **JWT** que o cliente recebe, já parseado.

Então agora vamos imprimir (console.log) as props que o nosso componente App está recebendo:



Como podemos ver, recebemos os estados citados acima como props.

Então vamos editar o App.js

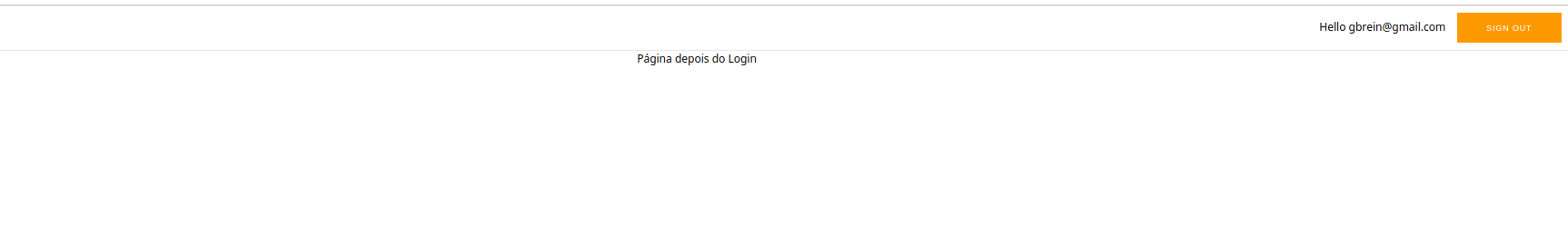
Para organização do código coloquei a prop **authState** em uma constante e criei duas outras constantes, uma com o conteúdo do App que será exibido caso esteja logado e uma com o conteúdo vazio para ser exibido caso esteja deslogado.

Por fim colocaremos um IF para verificar se o **authState** tem valor ‘signedIn’, o que significa que o usuário está logado.



Agora temos uma exibição correta na nossa página.

Recomendo testarem para ver a página funcionando.



\*Pagina depois de logada com o header padrão do amplify (componente chamado de greetings) para logoff.

Não entrarei neste tema agora, mas se você checar o componente **Authenticator**, verá que o estado **authData**, está preenchido com o JWT decodificado.



**Alterando o Theme:**

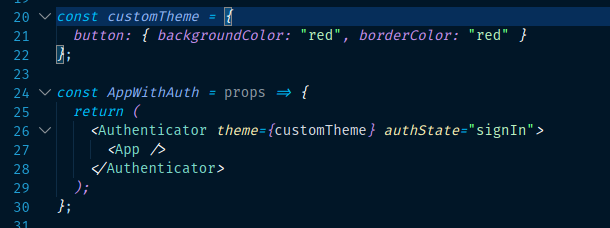
Vamos criar um tema diferente para o SignIn, alterando por exemplo a cor do botão de login e alterar os textos dos labels.

Para isso vamos apenas alterar algumas coisas no arquivo **AppWithAuth.js**.

- Customizando o **Theme** em si:

Para alterar caracteristicas do CSS sem ter que criar uma nova página, apenas sobreescrevendo o componente-UI do **amplify**, vamos criar um objeto chamado **customTheme**, dentro dele passaremos o parametro que desejamos mudar e o CSS novo.

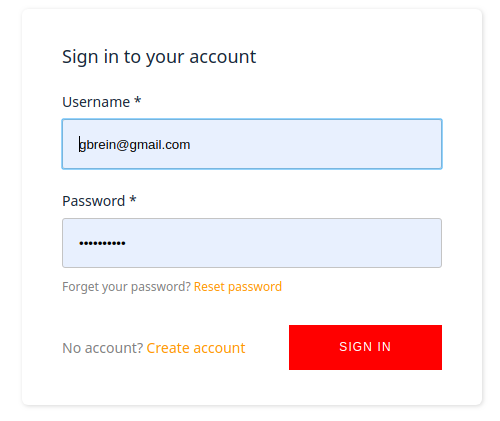
Também vamos passar este parametro no componente **Authenticator**.



E como resultado teremos o botão vermelho, conforme o CSS que foi reescrito

Se tiver alguma dúvida, pode usar o link a seguir como referencia:

https://github.com/aws-amplify/amplify-js/blob/master/packages/aws-amplify-react/src/Amplify-UI/Amplify-UI-Theme.tsx



- Para alterar o texto dos labels, por exemplo, vamos importar o **l18n** do **aws-amplify**.

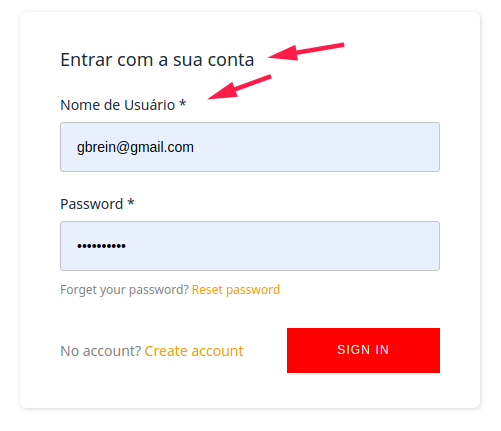
Em seguida criaremos um objeto, com outro objeto dentro e valores para tradução.

Para cada chave, sendo string que consta no componente do amplify-react, passamos um valor novo.

Em seguida aplicamos o idioma do objeto (objeto de dentro) e inserimos no método **putVocabularies** do **l18n**, parece mais complicado do que é na prática, veja o código.



Como resultado, onde tinhamos a string “Sign in to your Account” agora temos o valor em português, o mesmo se aplica ao “Username”, resultado:

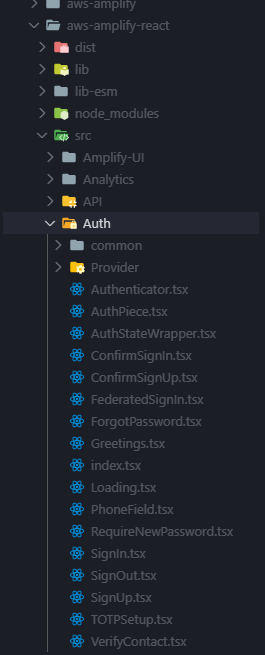


**Reescrevendo o render do método padrão do amplify**

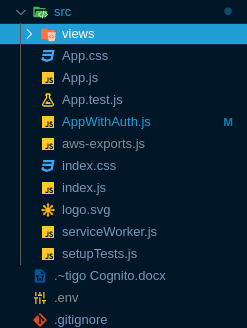
Está é uma abordagem diferente da anterior. Gosto dela pois aqui utilizaremos diversos métodos, verificações, testes que já foram abordados pelo time da AWS e apenas substituiremos o Render destas classes.

Porém em contrapartida não tem um código tão limpo, o código precisa ser muito bem comentado para alguma pessoa, futuramente, que não tenha mexido no mesmo entender.

Antes de colocar a mão na massa, deem uma olhada no src da pasta aws-amplify-react no node\_modules. Lá se encontra uma pasta de auth que será a base do código usado nesta sessão.



Dentro do nosso src, do nosso projeto, vou criar uma pasta de views para guardar os novos renders.



Vamos criar uma tela personalizada de Login sem personalizar o **theme**.

Vou utilizar o material-UI para fazer a tela de login. Usei os exemplos da própria material-UI para customizar a minhas telas (<https://github.com/mui-org/material-ui>).

Antes vamos instalar o material-ui e o styled-components:

npm install @material-ui/core

npm install @material-ui/icons

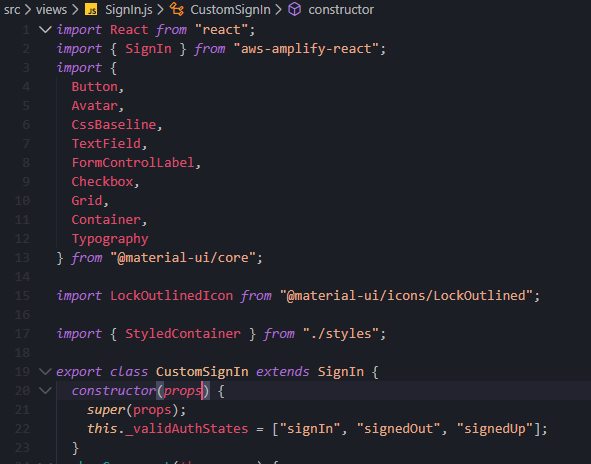
npm install styled-components

Dentro de views vamos criar um arquivo **SignIn.js** e umarquivo **Styles.js**. Vamos começar importando o React, alguns componentes do Material-UI, e por fim e mais importante, vamos importar um componente **SignIn** do **aws-amplify-react.**

Neste arquivo vamos instanciar uma classe (**CustomSignIn)** que herda os métodos deste componente **SignIn** do amplify.

Por fim, vamos inicar um construtor com o super(como sempre que temos uma classe herdando métodos de outra classe) e referir o estado **\_validAuthStates,** este é uma propriedade que o cliente do **amplify** passa para os componentes filhos.

Aqui vamos setar que os argumentos que quando este estado possuír, o componente será renderizado (neste caso “signIn”, “signedOut”, “signedUp”).



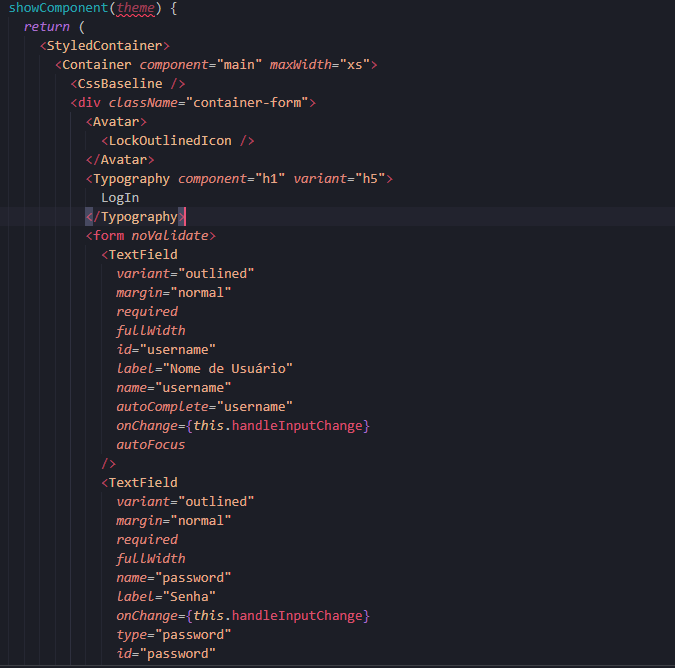
Como disse anteriormente se formos na pasta node\_modules/aws-amplify-react/src/auth, teremos a classe SignIn que estamos herdando. Vale a pena dar uma checada nos métodos e implementação disponível lá.  
Nos ajudará a entender que argumentos devemos passar para os métodos que existem lá.



Vemos aqui que a implementação do método signIn, que envia os dados para cógnito, recebe um evento disparado por um botão e um formulário.

Este evento deverá passar dois argumentos do formulário(input name): **username** e **password**

Então vamos reescrever o método **showComponent** que basicamente será o render desta classe:

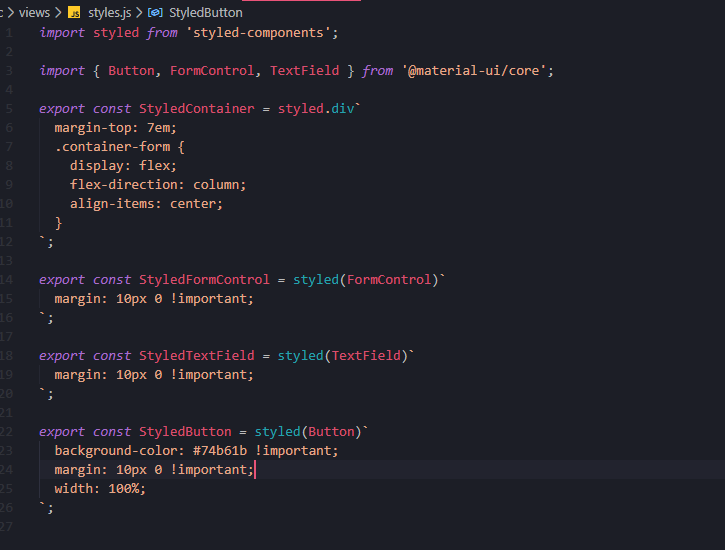




Agora que criamos a nossa tela, o método que irá disparar o formulário é uma função que possuí o evento como paramentro. Através do super, acessamos o método signIn do aws-amplify-react (classe herdada) passando como parâmetro o próprio evento.

Para complementar, adicionei dois outros botões, que disparam métodos do signIn de mudança de estado. Lembra que falei que o componente é renderizado de acordo com os argumentos do estado **\_validAuthStates**? Então, esse changeState altera esse estado para que outro componente seja renderizado no lugar. Quando houver “signUp” como estado, outro componente com um formulário de cadastro será exibido. Ou quando mudarmos o estado para “forgotPassword”, o formulário de recuperação de senha será exibido.

Agora vamos editar o arquivo **styles.js.** Faremos apenas algumas pequenas customizações

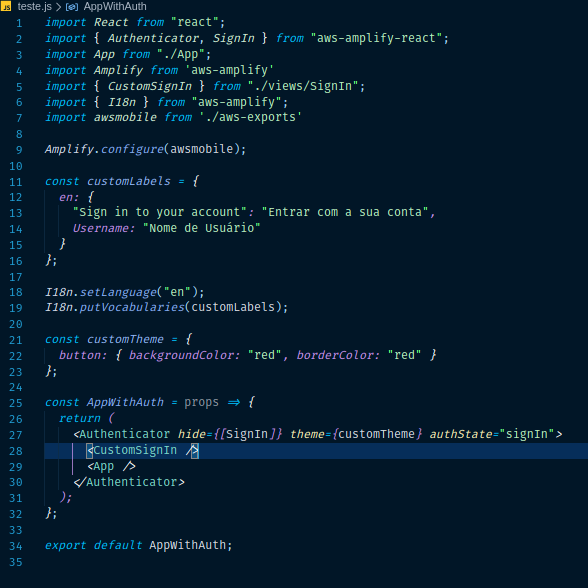


Agora, por fim, voltamos no nosso **AppWithAuth.js.**

Primeiro vamos importar nosso novo componente **CustomSignIn** e o **SignIn** do aws-amplify-react. Em seguida dentro do componente renderizado **Authenticator** colocaremos a prop **hide** passando um objeto com um array dentro. Todos os componentes que colocarmos dentro deste hide, não serão exibidos diretamente pelo Authenticator.

Por exemplo, se colocarmos o ForgotPassword dentro desta array, o amplify não exibira mais nativamente o formulário de reset de senha, cabendo a nós importarmos ele.

Em seguida vamos renderizar o componente **CustomSignIn** antes do nosso **App.**



Salvamos e damos start, eis o resultado:



Terminamos nossa renderização. Agora vamos ao passo final.

**Criando nossas próprias páginas, utilizando de métodos próprios.**

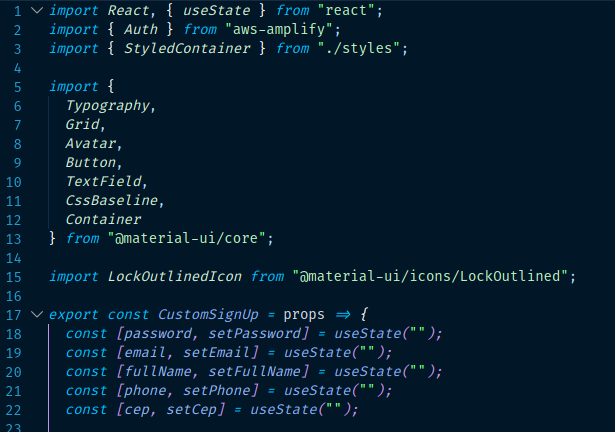
Vou fazer um pouco diferente agora. Em vez de renderizar a página de SignIn, vou fazer a página de SignUp (Cadastro).

Escolhi esta tela, pois aqui existem algumas especificidades, como criamos alguns campos customizados no Cognito existe um “pulo do gato” que demorei algum tempo para descobrir.

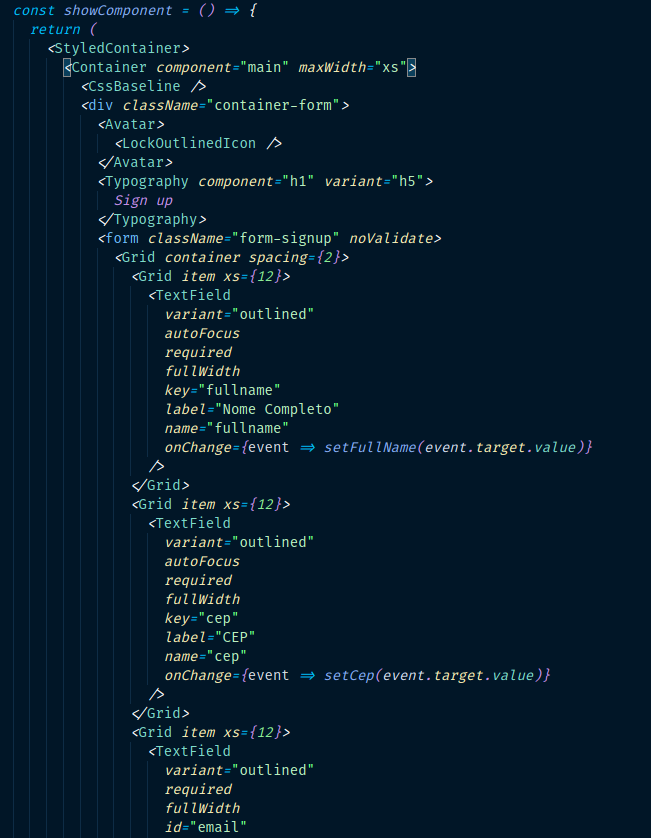
Também vou aproveitar este componente para escreve-lo com componente funcional (functional component).

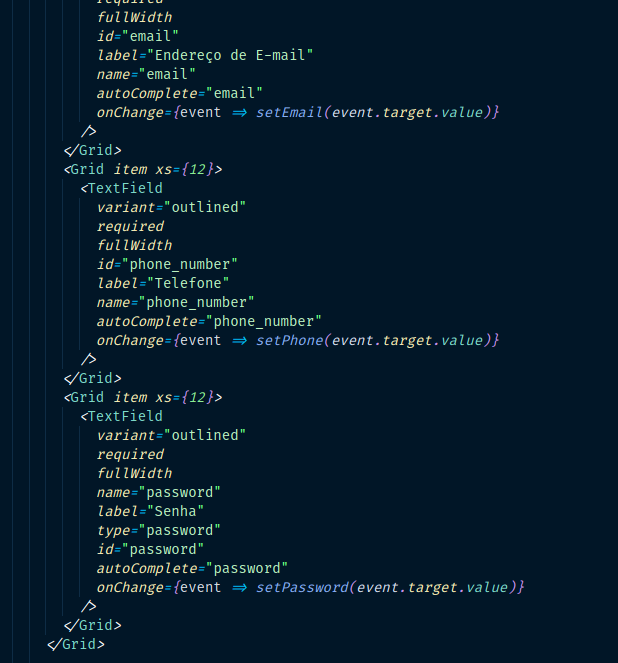
Vamos começar importando nossas dependencias, criando nosso componente e criando um estado através de useState para cada campo de cadastro.

A unica novidade é a importação de um módulo chamado **Auth** do **aws-amplify**, ele é nosso cliente de troca de informações com o servidor do cognito.



Em seguida vamos criar uma função chamada showComponent(seguindo o padrão de nomes do amplify).   
Dentro dela teremos nosso formulário e para cada campo vamos colocar a função setState correspondente para armazenar os valores:

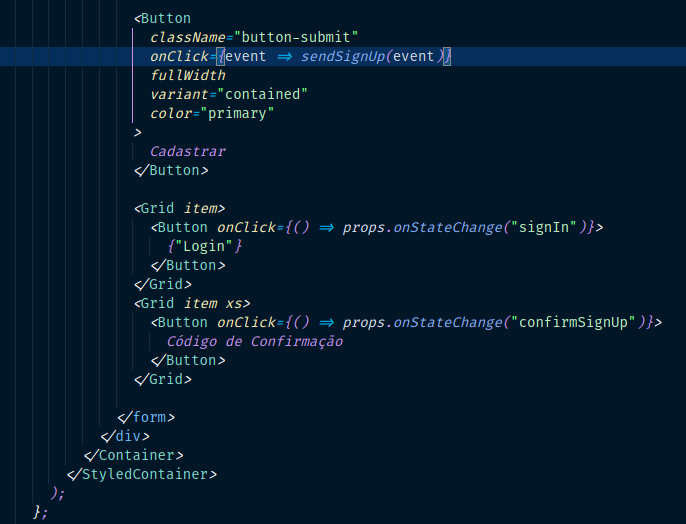




Por fim um botão de submit que deverá disparar uma função chamada **sendSignUp**, passando o evento como parametro. Esta função será responsavel por fazer o submit para o client do **amplify**.

Também adicionei 2 botões, um para voltar a tela inicial e um caso o usuário já tenha tentado criar uma conta.

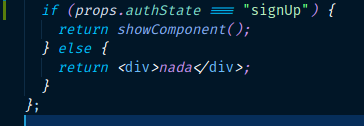
Para isso acontecer utilizaremos uma função recebida nas props do componente, passada pelo Authenticator, chamada onStateChange, que altera os valores do estado **authState**.



Escrevemos o formulário de cadastro, falta criar a função **sendSignUp** e configurar em que situações a tela deverá ser renderizada.

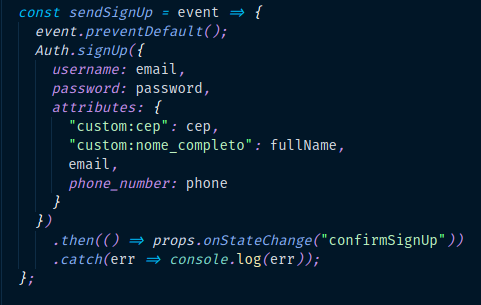
Vamos começar pelo segundo caso. Como disse quando comentava sobre a tela de SignIn, o estado **authState**, nos ajuda a identificar qual tela será renderizada.

Neste caso faremos um simples If, se o **authState** for “signUp” então renderizamos o showComponent(), caso contrario renderizamos uma div vazia.



Pronto, agora falta apenas enviar as infos para o cliente.

Vamos ver a função de **sendSignUp**:



Primeiro damos um event.preventDefault() para evitar execuções não solicitadas.

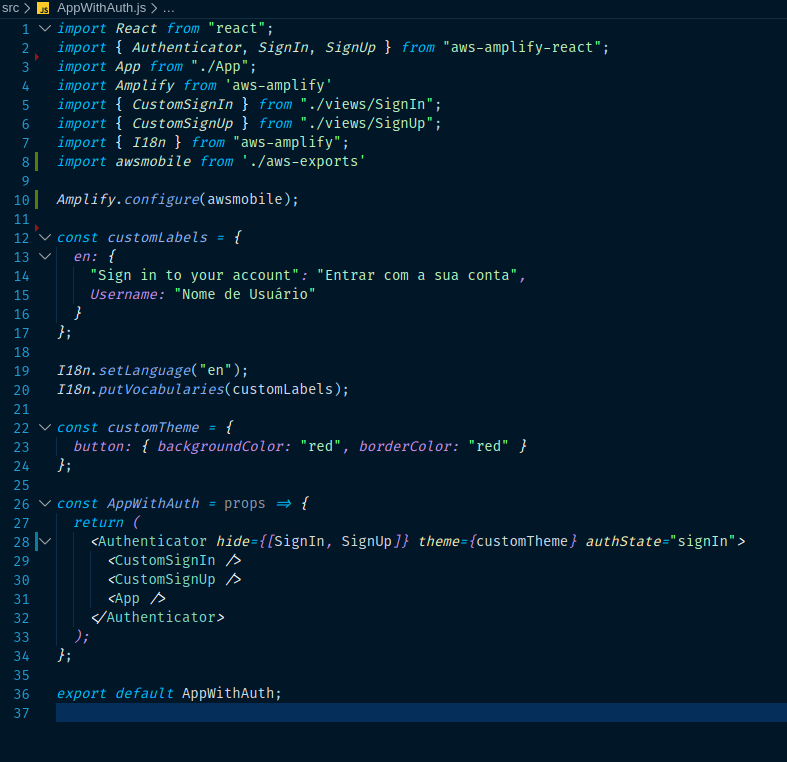
Em seguida utilizamos o método Auth.signUp passando como parametros os dados dos inputs.

Lembram que disse sobre um “pulo do gato” que me tomou algum tempo.

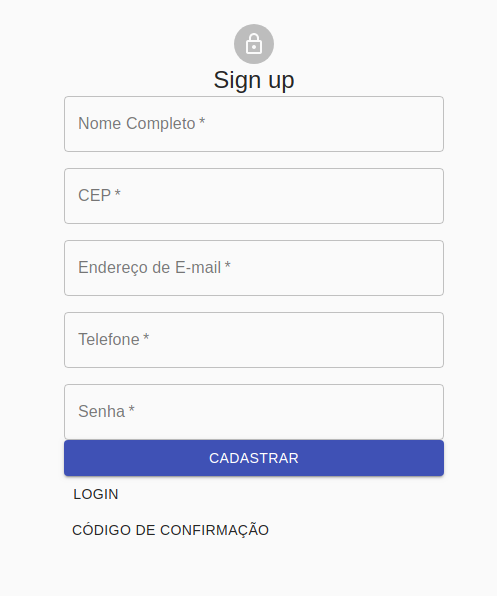
Os campos customizados devem seguir este padrão da imagem: “custom:nome\_do\_campo”: valor,

Sem isso não dá certo.

Antes de terminarmos, vamos dar um hide no componente original do **aws-amplify-react**  no arquivo **AppWithAuth.js**. Assim como fizemos antes com SignIn.



Vamos então testar esta tela.



**Autenticação com MFA**

Para que nossos usuarios loguem com MFA, precisamos disponibilizar uma tela de setup de MFA.

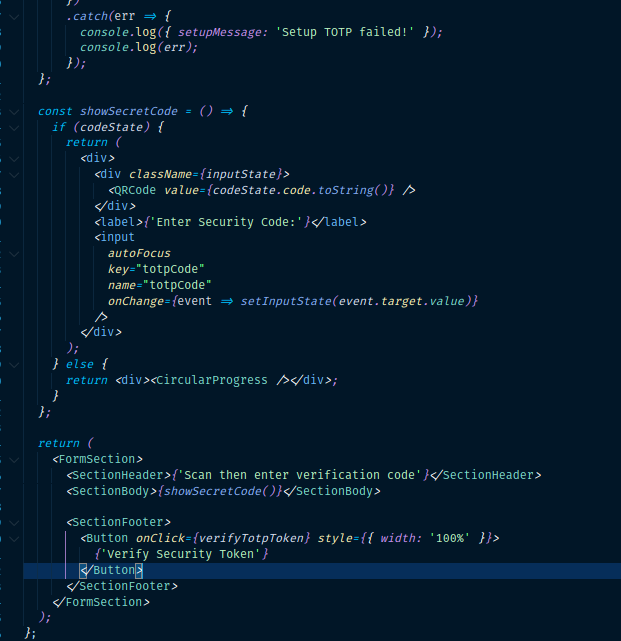
Este talvez seja a tela mais complicada de montar.

Não vou explicar ela detalhadamente, mas vou disponibilizar um exemplo de uma pagina de Setup de MFA.

Para criar um QRcode com os dados do MFA do usuário, para o mesmo cadastrar na sua aplicação favorita (google auth, authy, etc...) vamos usar a biblioteca **qrcode.react**.

O resultado do Setup de MFA estará disponivel via **console.log**, mas podemos alterar futuramente para redirecionar a alguma rota.





Se estiver com dificuldades todo código está disponibilizado no meu Github:

<https://github.com/gbrein/artigo-cognito>

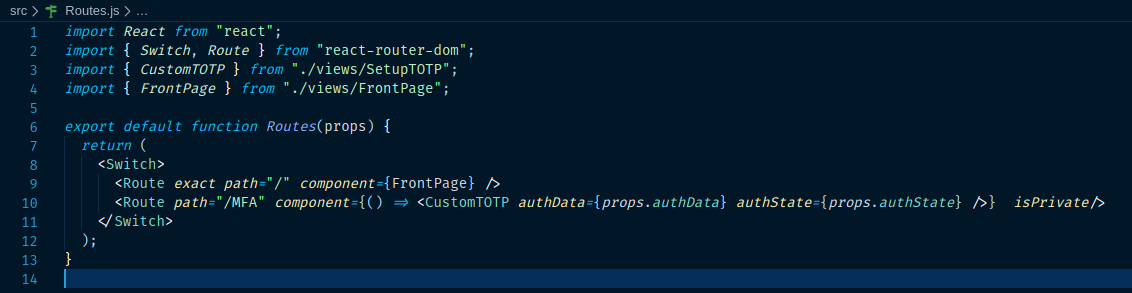
Por fim, apenas colocamos este setup em uma rota (usando react-router), que mostrarei a seguir.

npm install qrcode.react // instala o pacote que comentei mais acima

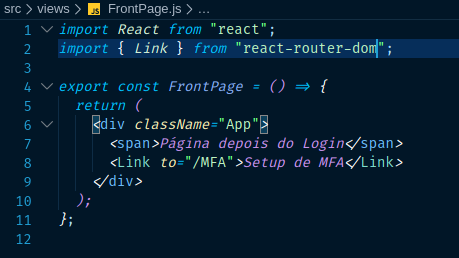
npm install react-router-dom // instala o react router

Também criaremos o arquivo Routes.js na pasta src e um FrontPage.js na pasta views.

No arquivo Routes criaremos 2 rotas uma para pagina inicial e uma para o setup de MFA criado anteriormente, passando o **authData** e o **authState** como props.

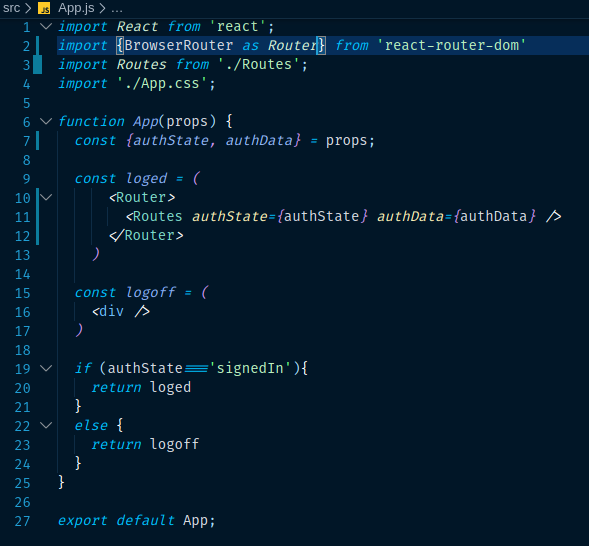


No **FrontPage.js** vamos iniciar uma pagina simples com um link para a pagina de MFA.



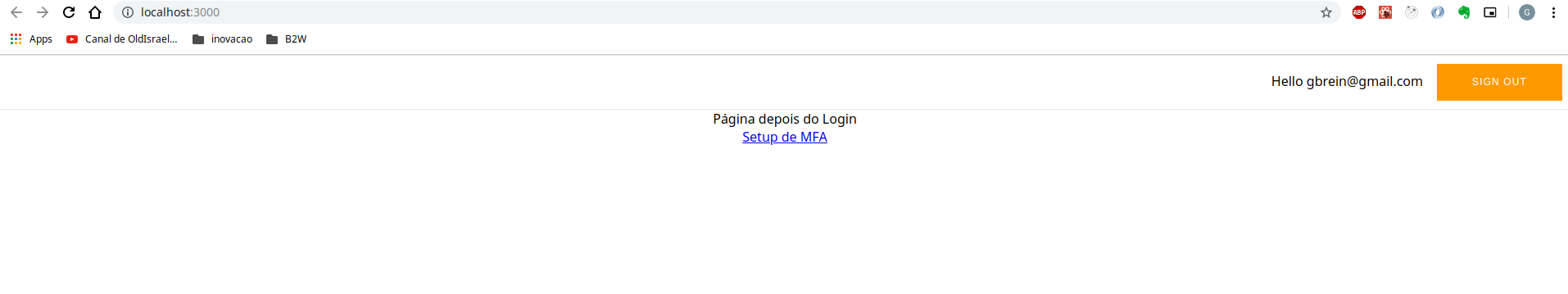
Então vamos alterar o App.js para configurar nossas rotas, também passei para o arquivo de rotas a prop **authData** comentei anteriormente, mas basicamente é nosso JWT decodificado.

Uma informação que pode ser importante para os demais componentes, use com cuidado, pois podem haver dados sensiveis dependendo do que você disponibilizar.

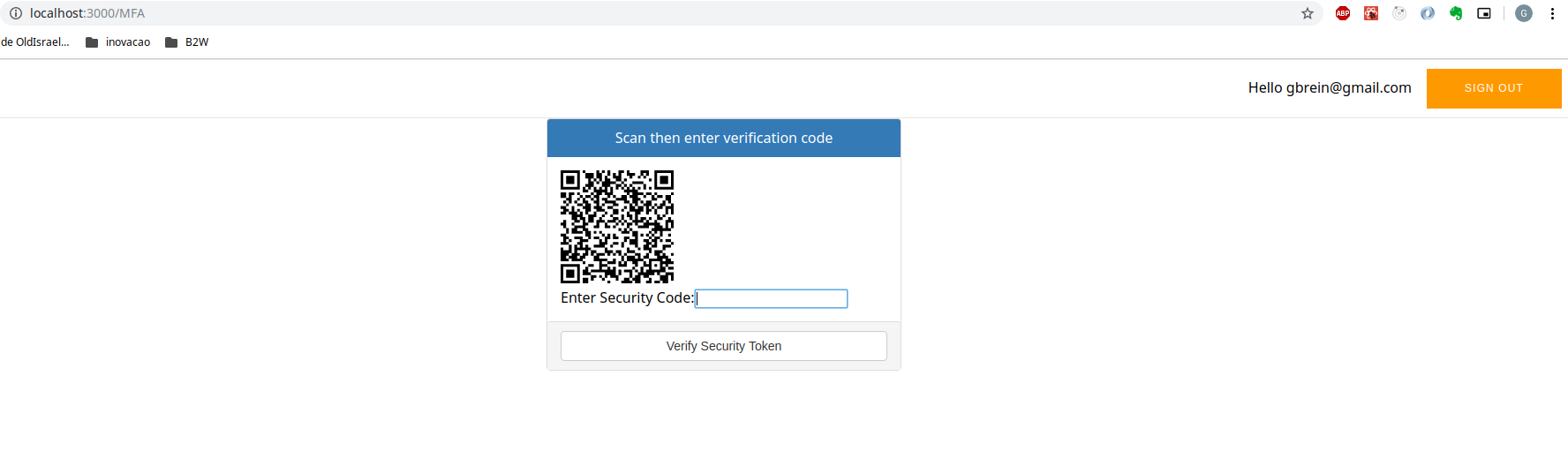


Então temos o resultado final:

Tela inicial



Tela de MFA



Referencias: