**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.

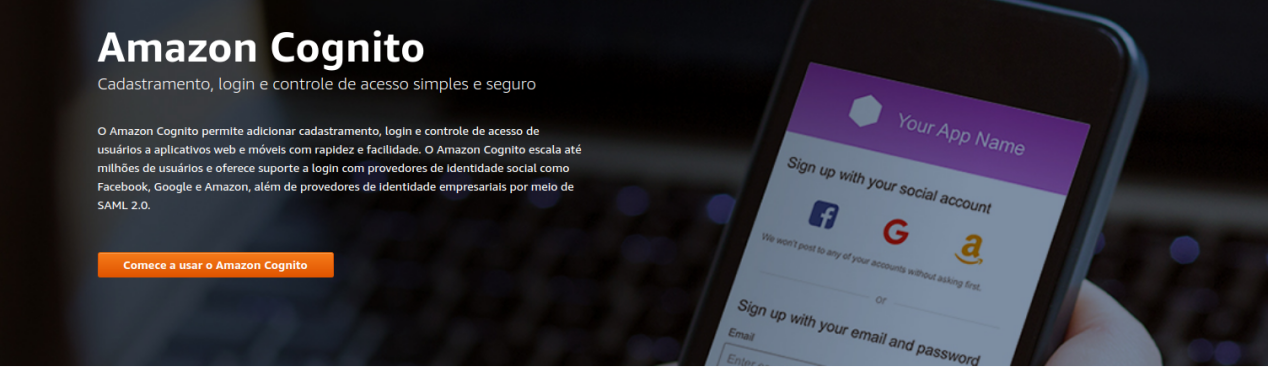
Para nós Devs autenticação é sempre uma dor de cabeça.  
Pensar em estratégias de segurança, banco, criptografia, validações e muitos pequenos detalhes que acabam tirando o nosso sono.  
Além disso temos a LGPD( Lei Geral de Proteção de Dados Pessoais ) batendo na porta em 2020.

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.

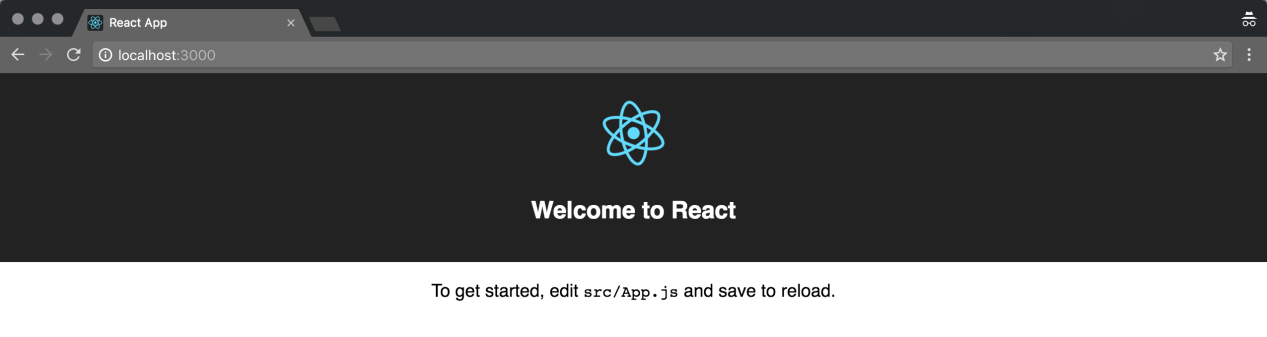
npm install -g create-react-app // Para instalar o create-react-app

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

cd artigo-cognito // Entra na pasta do projeto

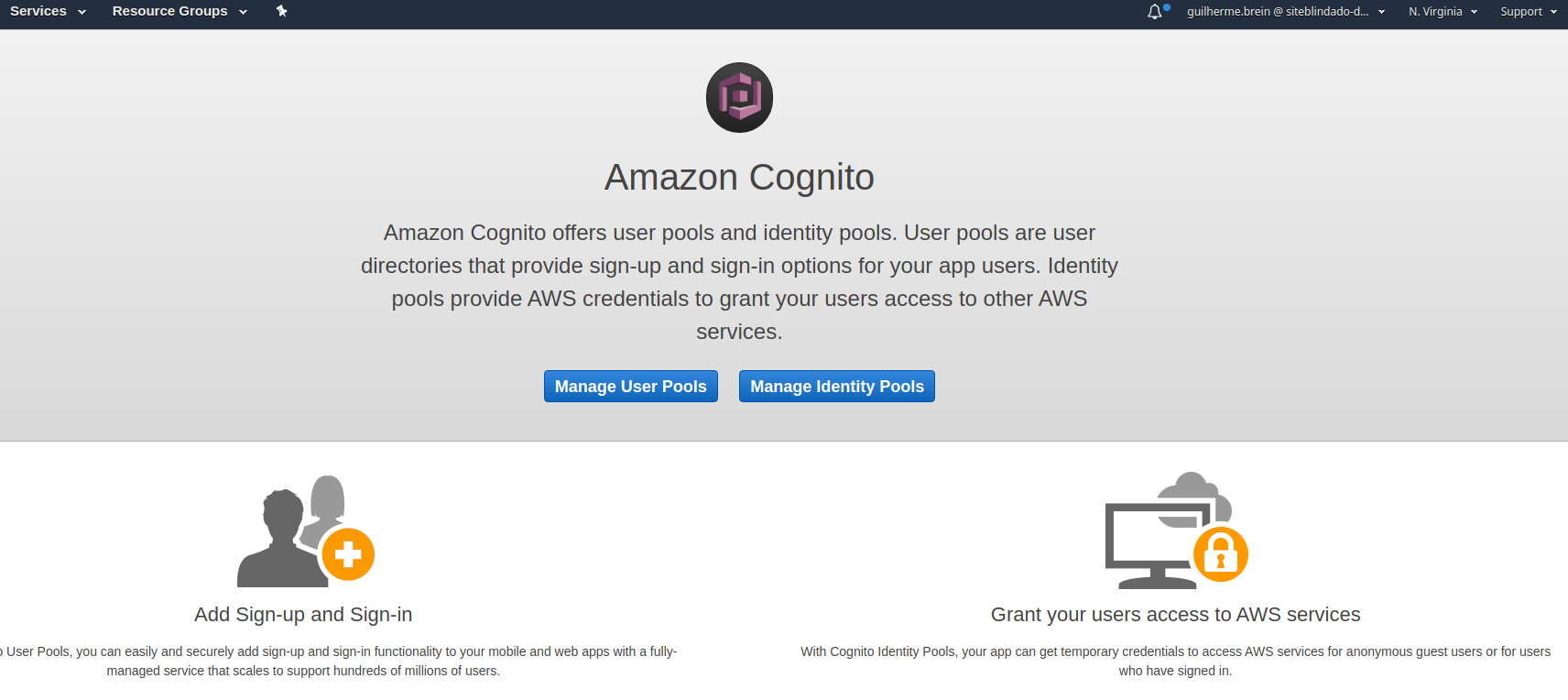
npm start // Inicia o projeto

Se tudo der certo você verá uma tela como esta:



**Cognito**Vamos ver aqui como configurar o Cognito manualmente.  
Poderiamos realizar esta tarefa diretamente via Script/Amplify, mas deixarei este tema para outro tópico.

Vamos acessar o painel do Cognito no AWS.



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.

É isto o que vamos utilizar na sequencia.

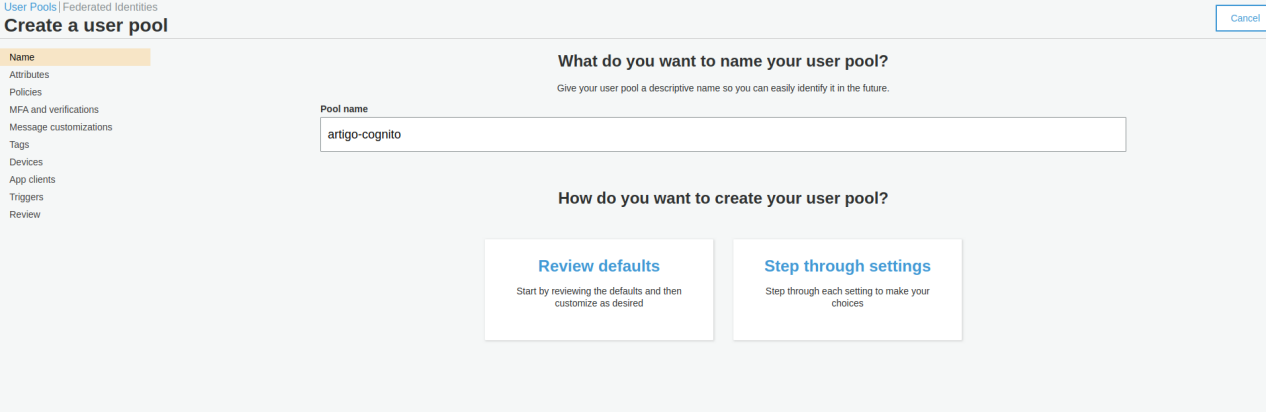
- Manage Identity Pools - É uma ferramenta de gestão para usuários convidados, logados por ferramentas externas e usuários que receberam um token de identificação.

Não mexeremos nisso agora.

Agora vamos criar nossa User Pool.

Clique no botão **Create a user pool.**

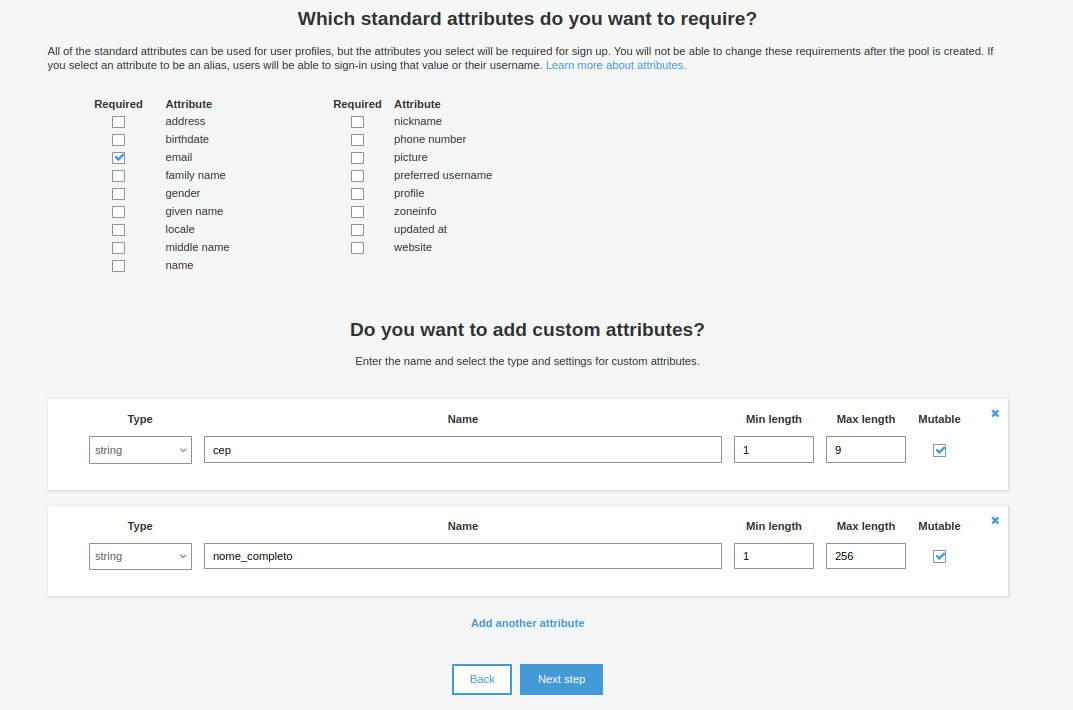
Em seguida vamos dar um nome para nossa User Pool,



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.  
Isso é importante, pois mais pra frente será importante saber como lidar com estes campos customizados.

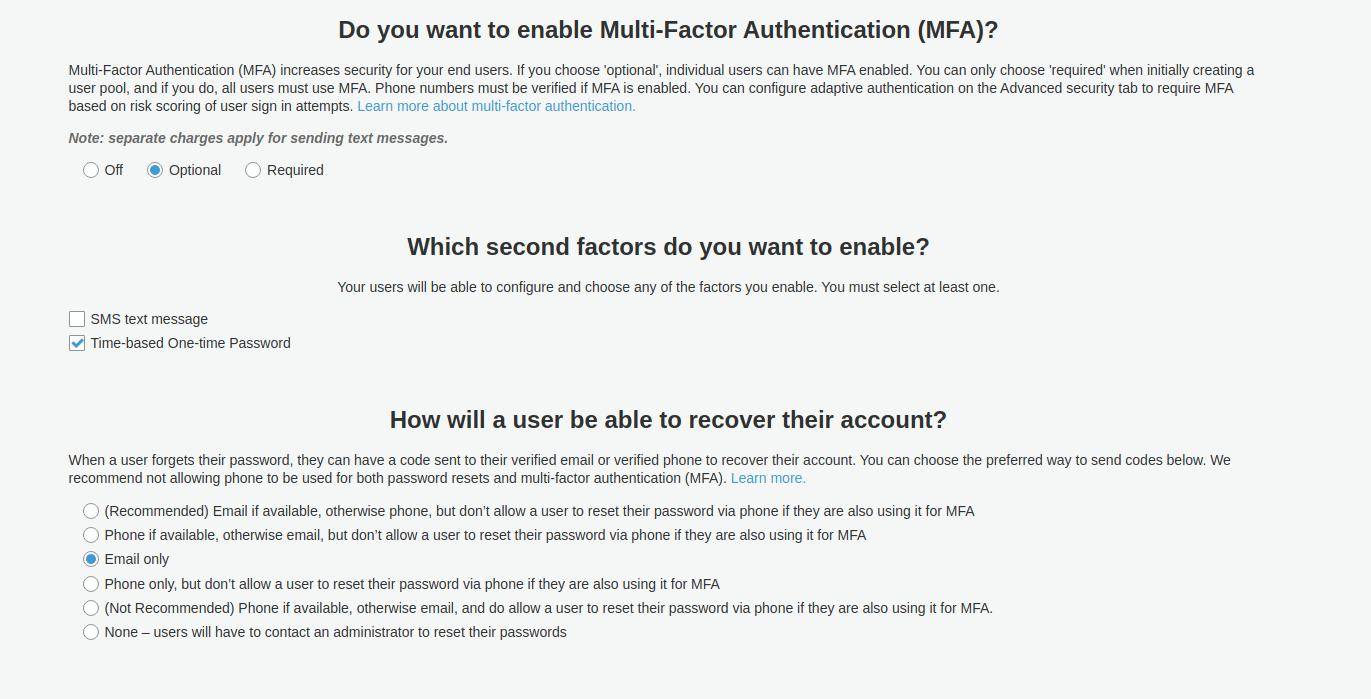


Em seguida vamos para a aba MFA and Verifications no menu.

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...

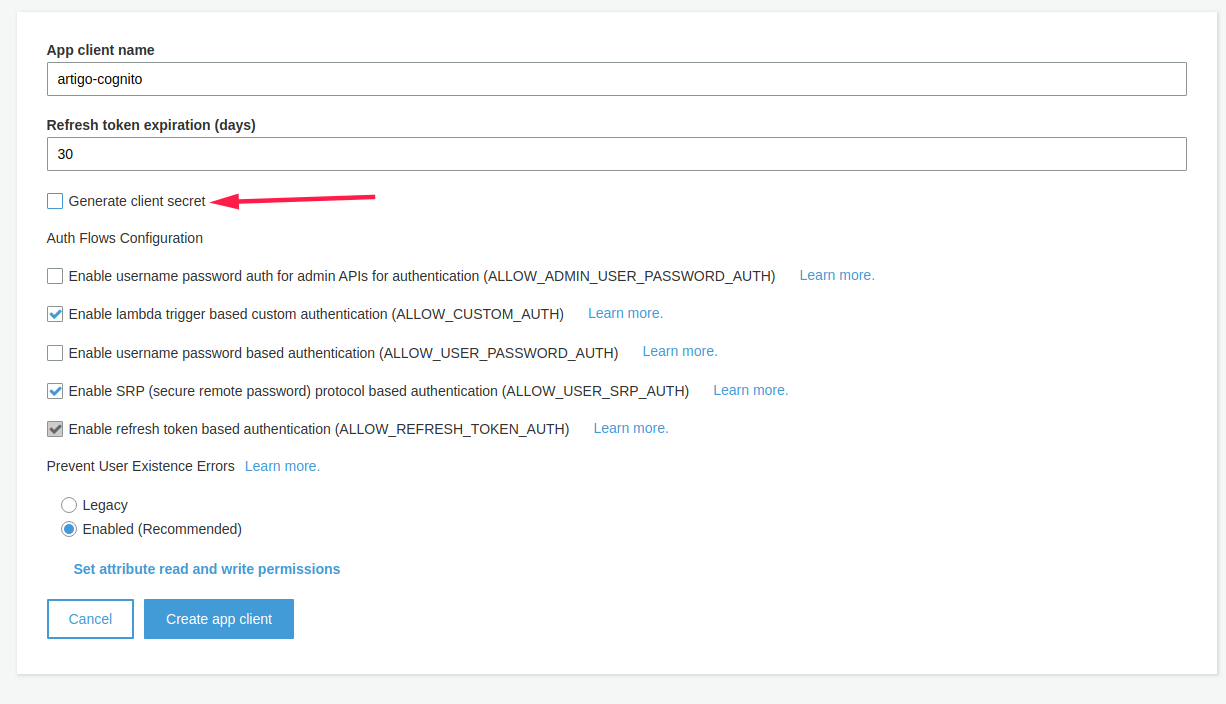
Por fim marquei a opção de email para recuperação de senha.



Em seguida vamos na aba App Clients.

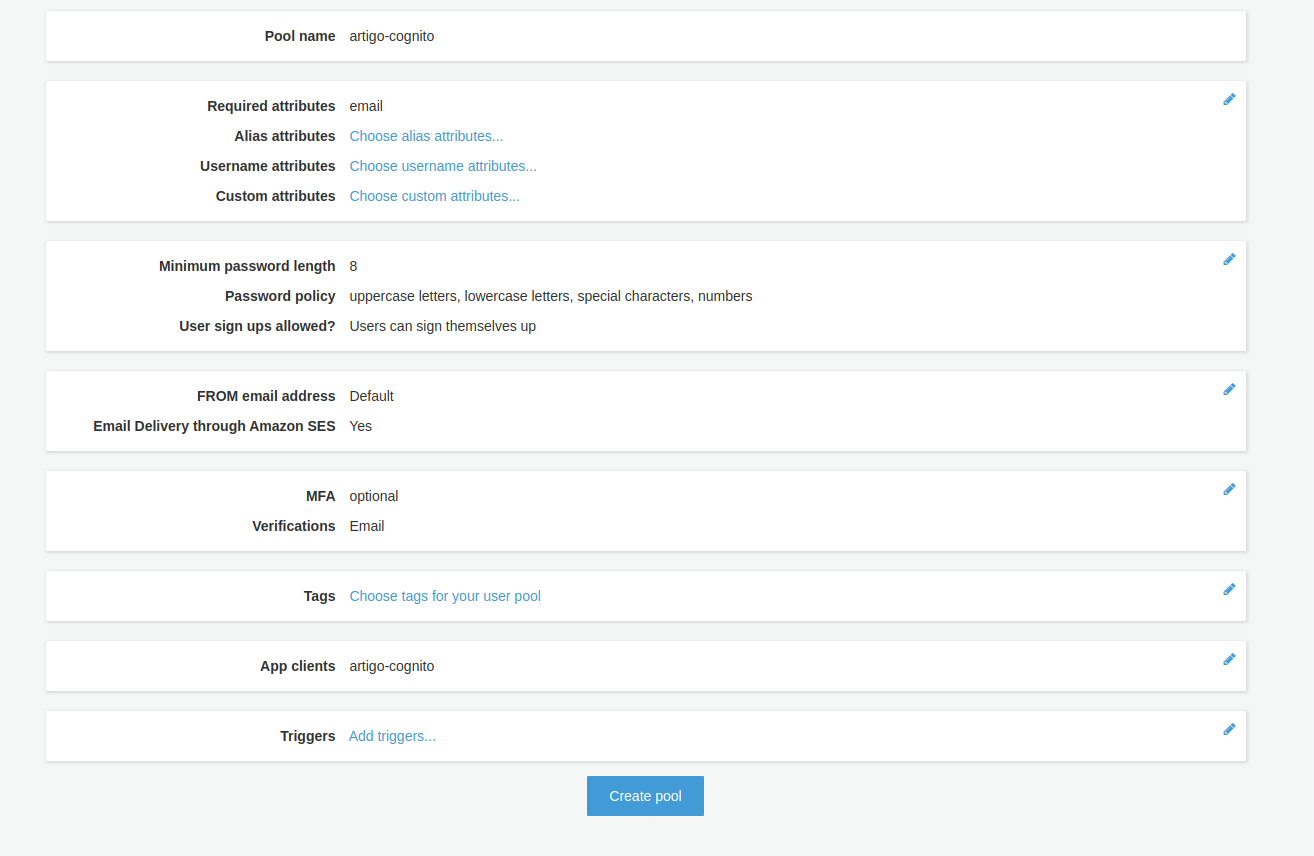
Devemos criar um App Client para que nossa aplicação React possa acessar o cognito e realizar a autenticação.

Por enquanto vamos apenas dar um nome e criar o App Client, apenas **desabilite** o campo **Generate client secret (MUITO IMPORTANTE)**, sem isso o app não vai funcionar.



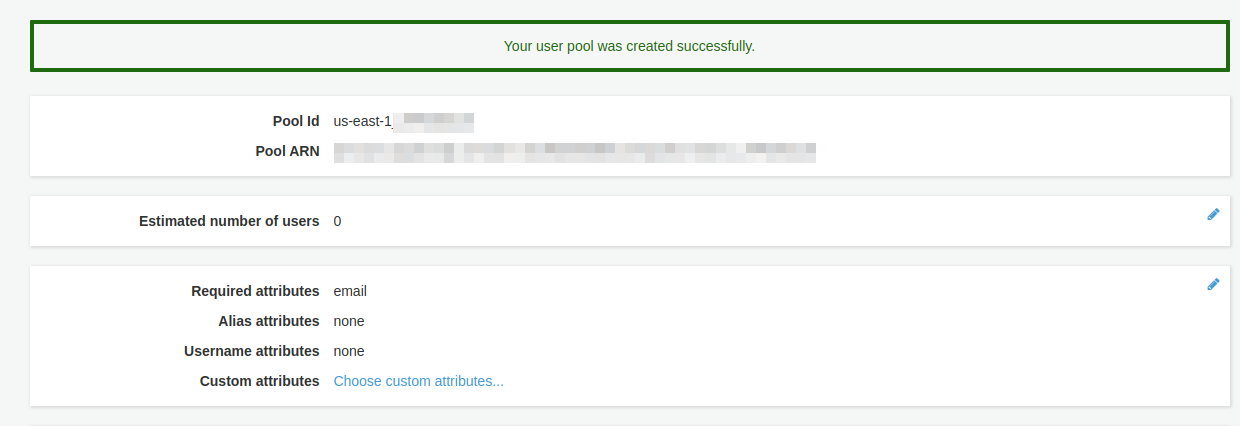
Um pequeno adendo é para aba Triggers. É possível disparar funções Lambda (microsserviços) a partir da autenticação.  
No momento não entrarei neste assunto, mas é um recurso fantastico para aplicações.

Por fim vamos a pagina final, se tudo estiver correto pode clickar em Create Pool

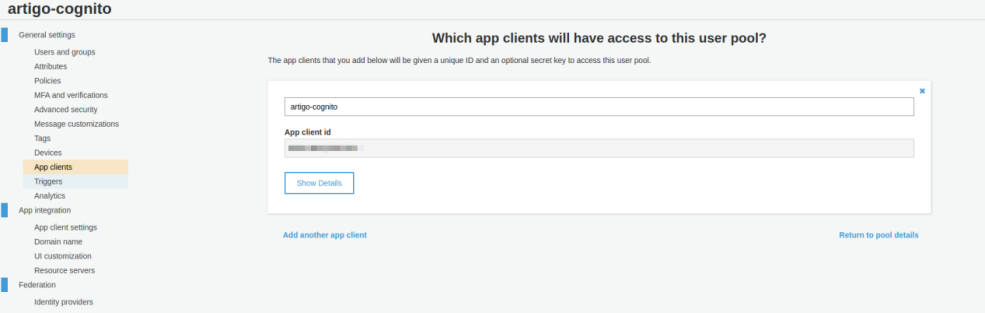


Pronto criamos nossa User pool, agora vamos anotar 3 dados.

Em General Settings:  
Pool Id e a região, que nada mais é do que o inicio do Pool Id (no meu caso - us-east-1).



Em seguida vamos em App Clients e teremos o App Client Id



Pronto! Temos tudo que precisamos do Cognito (Region, Pool Id e App Client ID).

Vamos agora ver como fazer nossa aplicação conectar com isso.

**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.

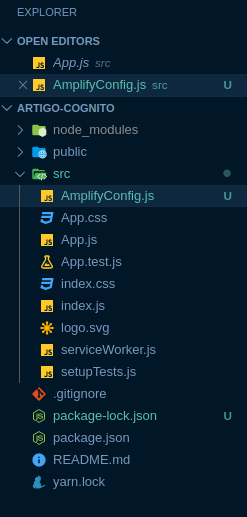
Aqui na Site Blindado, já utilizamos o Terraform (https://www.terraform.io/) como ferramenta de Devops.

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

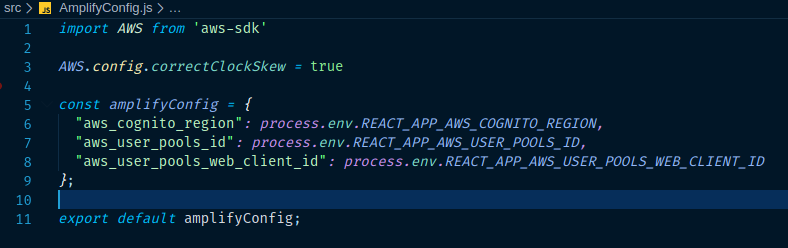
npm install aws-amplify  
npm install aws-amplify-react

npm install aws-sdk

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

Dentro da nossa pasta **src** vamos criar um arquivo chamado **AmplifyConfig.js**

Dentro do seguinte arquivo vamos colocar o seguinte código:



O que estamos fazendo aqui?

Primeiro importamos a biblioteca da aws-sdk, então aplicamos a o método **correctClogSkew** que serve para corrigir a diferença de hora entre o cliente e o servidor (fuso horário distinto).

Depois criamos o objeto **amplifyConfig** e colocamos as informações que guardamos no passo anterior quando criamos nosso pool do cognito.

Primeiro a região, depois o pool\_id e por ultimo o app client Id.

Por fim exportamos o objeto **amplifyConfig.**

Recomendo ue não deixem as chaves e ids abertos em projetos como github e etc.   
Utilize um arquivo .env para esconder suas chaves e coloque o mesmo no .gitignore.

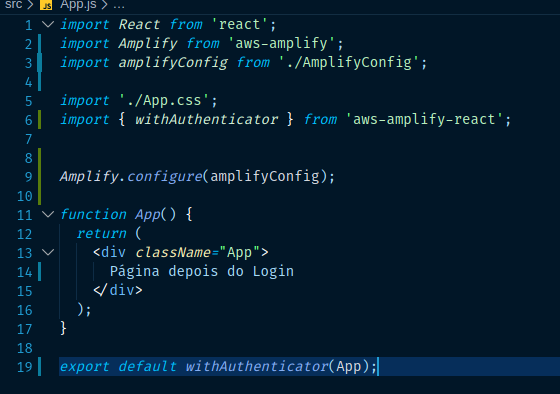
Caso, assim como no meu projeto esteja utilizando o create-react-app, utilize o prefixo REACT\_APP\_ antes do nome da variavel no arquivo .env.

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.

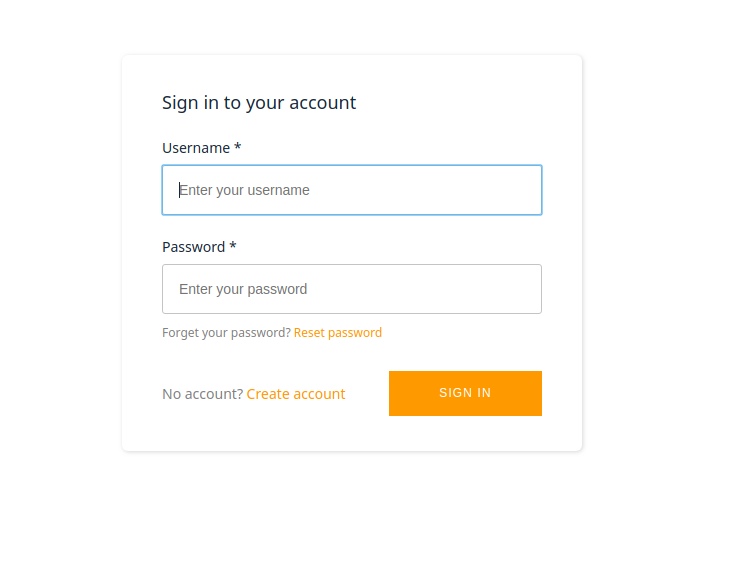
Por fim nosso objeto **amplifyConfig** do arquivo com mesmo nome que criamos anteriormente.

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**.

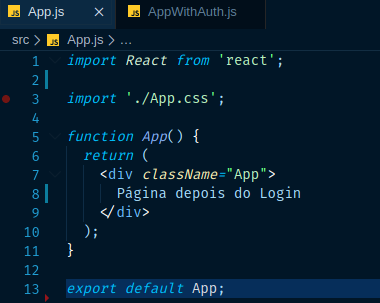
**Utilizando o Authenticator ao invés do WithAuthenticator:**

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

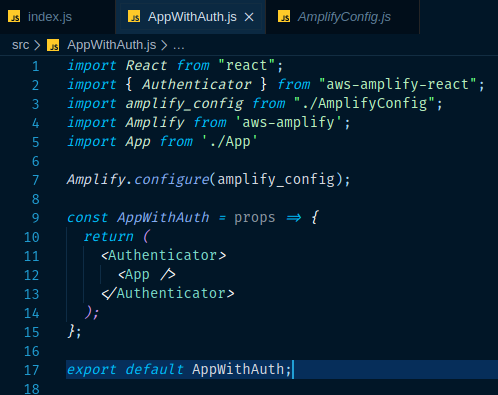
A diferença entre os dois é que o **Authenticator** permite muitas customizações, enquanto o outro é mais restrito.

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.

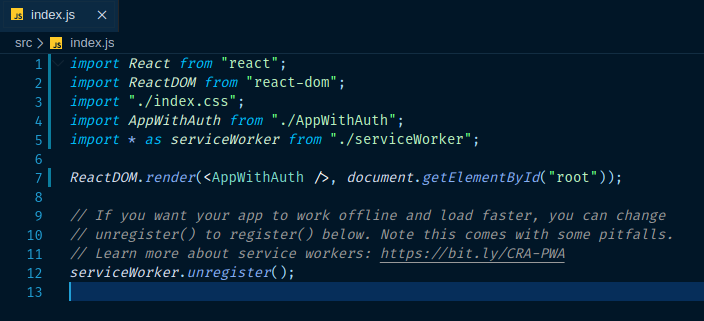


Agora vamos editar o arquivo **AppWithAuth.js.**

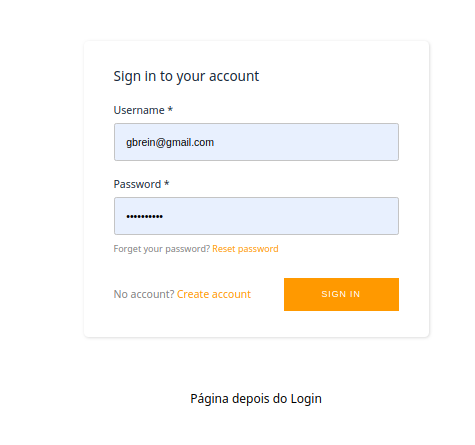


Neste arquivo, basicamente, envolvemos o componente **App** com o componente **Authenticator** do **Amplify**, ele será responsável por transmitir/sobreporas props e rotas de autenticação antes e para o **App**.

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



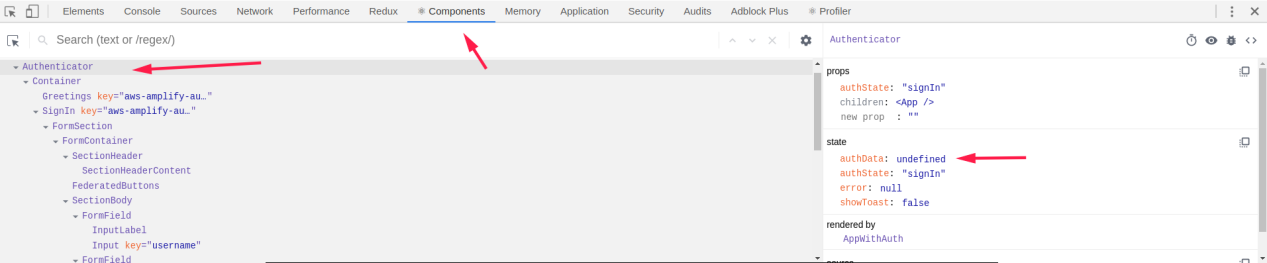
Salvando teremos como resultado



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:

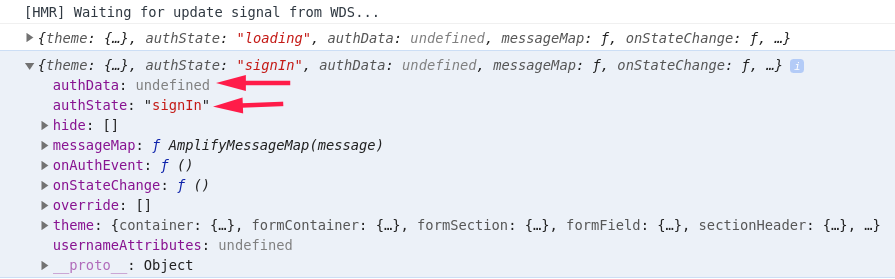


Vemos que o componente **Authenticator**, envolve todo o restante do código, e como estado temos **authState** e **authData**.

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

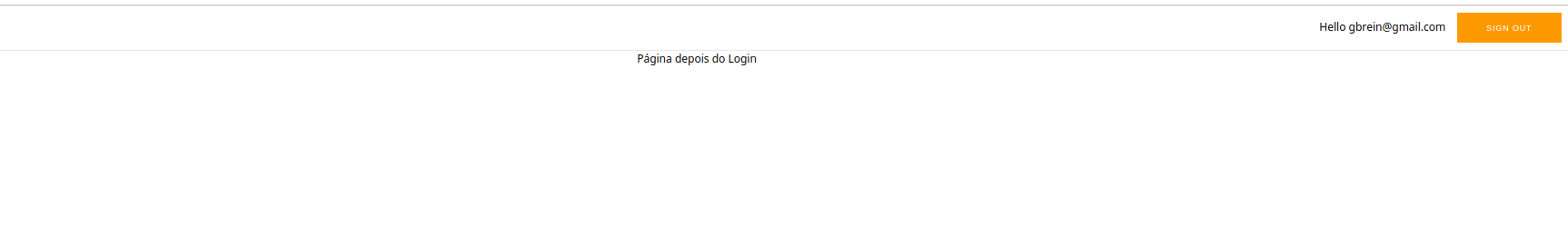
Para organização do código coloquei a prop **authState** em uma constante e criei duas outras constantes, uma com o conteúdo do App que será exibido caso esteja logado e uma com o conteúdo vazio para ser exibido caso esteja deslogado.

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:**

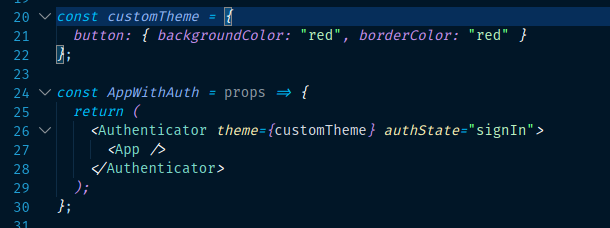
Vamos criar um tema diferente para o SignIn, alterando por exemplo a cor do botão de login e alterar os textos dos labels.

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.

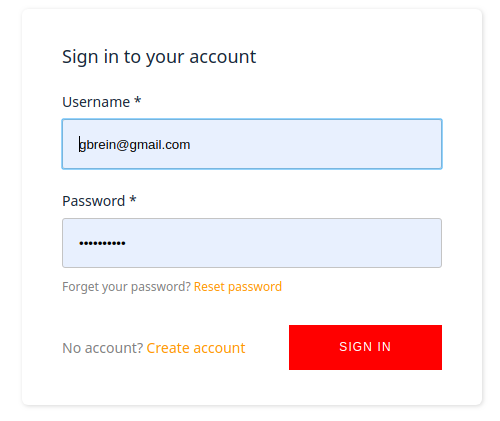
Também vamos passar este parametro no componente **Authenticator**.



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

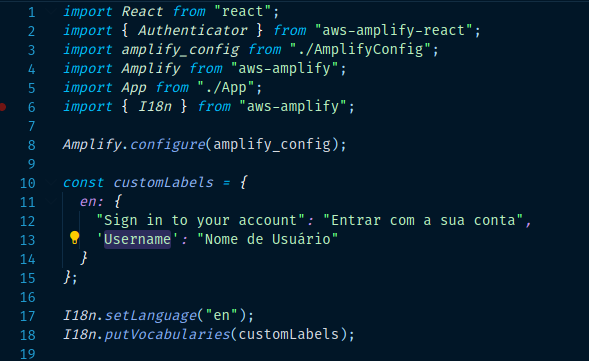


- Para alterar o texto dos labels, por exemplo, vamos importar o **l18n** do **aws-amplify**.

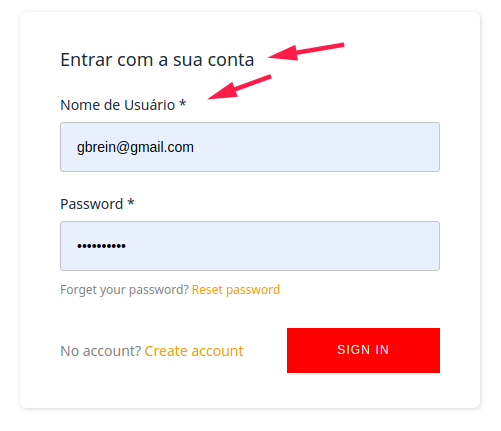
Em seguida criaremos um objeto, com outro objeto dentro e valores para tradução.

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:

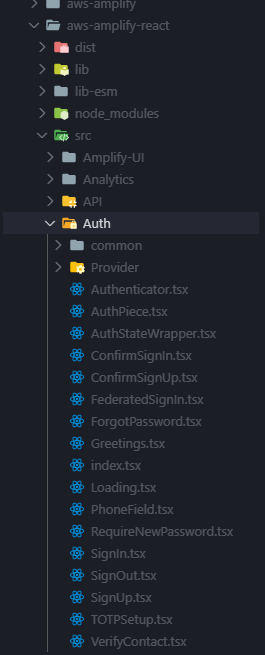


**Reescrevendo o render do método padrão do amplify**

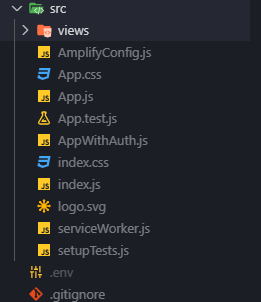
Está é uma abordagem diferente da anterior. Gosto dela pois aqui utilizaremos diversos métodos, verificações, testes que já foram abordados pelo time da AWS e apenas substituiremos o Render destas classes.

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

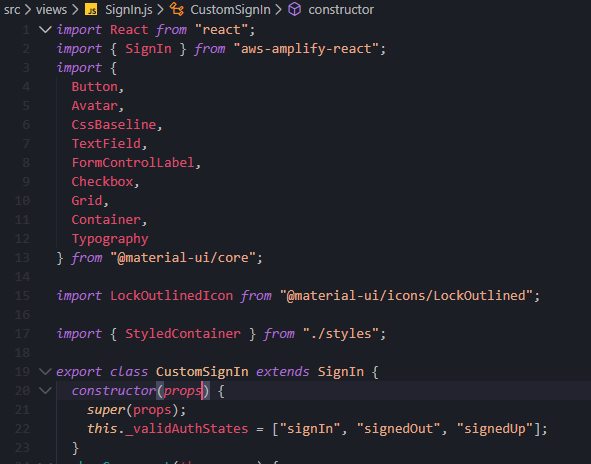
npm install styled-components

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”).



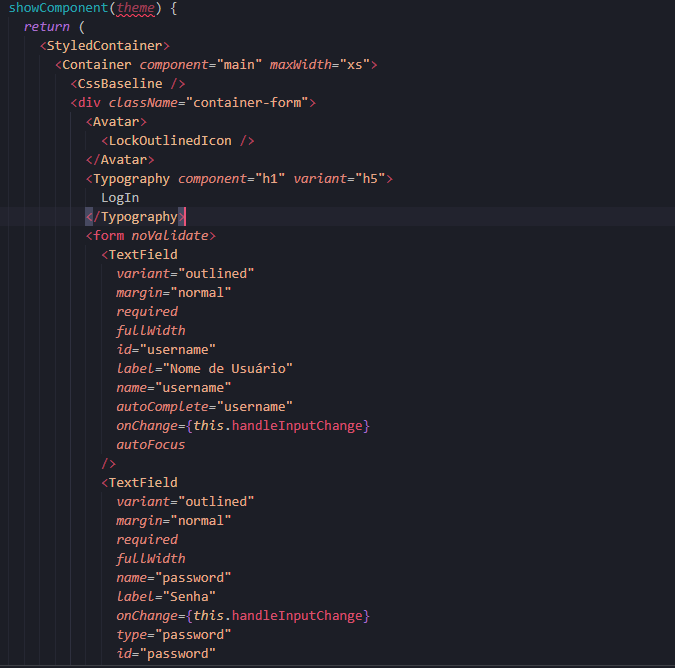
Como disse anteriormente se formos na pasta node\_modules/aws-amplify-react/src/auth, teremos a classe SignIn que estamos herdando. Vale a pena dar uma checada nos métodos e implementação disponível lá.  
Nos ajudará a entender que argumentos devemos passar para os métodos que existem lá.



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:

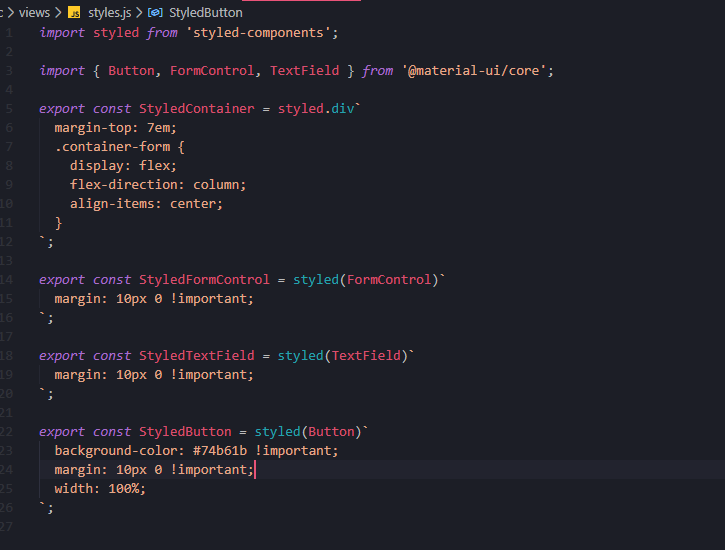




Agora que criamos a nossa tela, o método que irá disparar o formulário é uma função que possuí o evento como paramentro. Através do super, acessamos o método signIn do aws-amplify-react (classe herdada) passando como parâmetro o próprio evento.

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

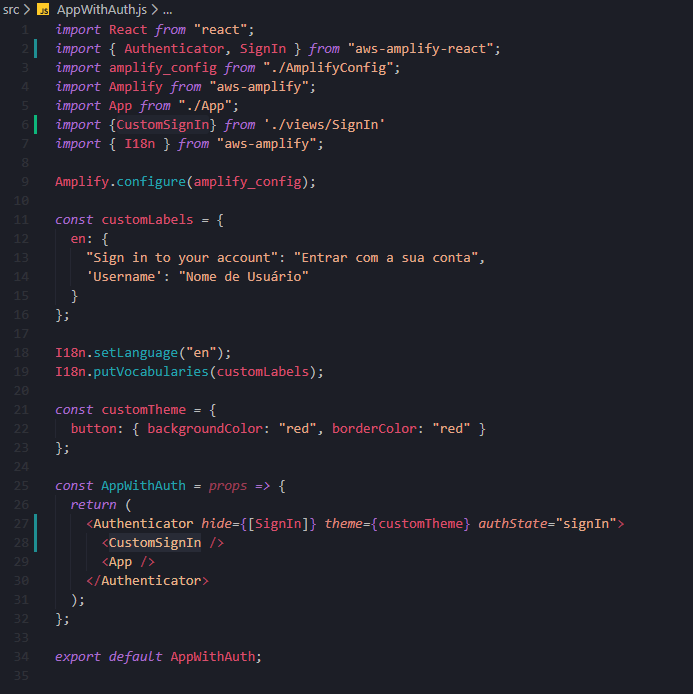


Agora, por fim, voltamos no nosso **AppWithAuth.js.**

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).