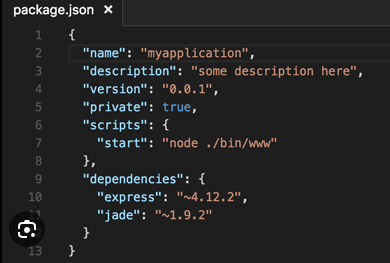
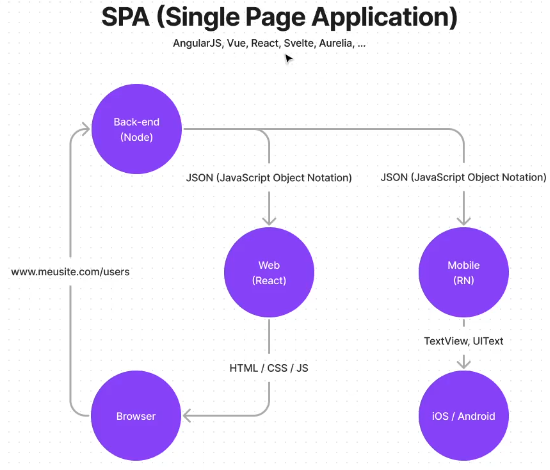
Entender Json: Json e um arquivo universal, que todas as linguagens conseguem utiliza-lo por sua simplicidade, basicamente é uma estrutura simples de arrays, objetos e atributos.

Observe: 

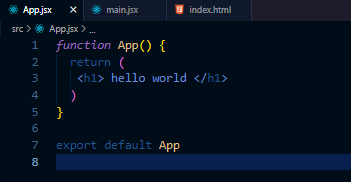
  
Bundlers e compilers

Babel

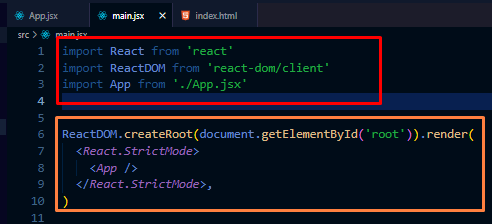
Compilers: compila JS atual para um JS antigo casa seja necessário por conta de um navegador não suportar a versão.

Bundlers:   
Vipe importa módulos na aplicação

**Fundamentos react**



**Observe que a funcao apenas retorna o titulo em html, veja que tem JS e HTML juntos, por conta do .jsx extensao do js.**



Importa o react, ele quem vai controlar os módulos.  
 Importa o reactDOM ((Document Object Model) o controlador que pega os objetos HTML e os representa em JS (queryselector, elmentbyid etc)

Cria e renderiza o elemento principal no html com base no elemento raiz da pagina html, seria o que contem o id root

**Ele renderiza o App, gerando o HTML criado em App.js**

**------------COMPONENTES------------**

**Muito importante!**

Componentes são utilizados em todo o projeto React.

A ideia e ter um Código em uma parte da aplicação reutiliza-lo/repeti-los sempre que necessário na aplicação inclusive com informação diferentes e que ele se torne adaptável a cada uso, também fica mais fácil de dar manutenção, por ser apenas um pedaço da aplicação.

Componente é um arquivo que abstrai parte da interface em outro arquivo e isso pode ser repetido diversas vezes durante o app.

Quando criar componentes? Têm o óbvio:

Reutilização:

Princípio DRY (Don't Repeat Yourself):Se um conjunto específico de HTML, CSS e JavaScript está sendo repetido em vários lugares da sua aplicação, é um forte indicativo de que você deve extraí-lo para um componente.

Organização e Manutenção:

Separação de responsabilidades: Cada componente deve ter uma responsabilidade única. Isso torna o código mais fácil de entender, testar e manter.

Componentes menores e mais focados: Dividir componentes grandes em componentes menores torna o código mais modular e facilita a depuração.

Facilidade de atualização: Ao isolar uma lógica específica em um componente, você pode fazer alterações nesse componente sem afetar outros componentes.

Por exemplo: Um componente que lista usuários tem um botão de upload para carregar novos usuários, é viável separar a lógica do upload em outro componente, pois caso seja necessário atualizar ou fazer manutenção ou atualização na listagem de usuário ou no comportamento do upload, seria melhor e mais ágil, pois a lógica estaria separada.

3. Abstração de Complexidade:

Componentes compostos: Componentes complexos podem ser construídos a partir de componentes menores. Isso permite criar interfaces mais ricas e interativas.

Encapsulamento de lógica: Você pode encapsular lógica complexa dentro de um componente, tornando a interface principal mais simples.

Chield e Parents:  
Chield componente flilho, ex: estou em App.jsx e chamo o componente Header, o componente Header passa a ser filho (chield) do componente App.jsx e da mesma forma o App.jsx passa a ser Parent de Header

List: qualquer coisa que percorre durante o app (iteração), quando se tem uma lista é necessário uma Key = identificação.

Então se der erro:



Basta inserir a key na montagem da lista:



**Por que Keys?**

Porque o React sempre se re-renderiza, ou seja ele carrega todo o componente novamente:  
O React re-renderiza componentes em três situações principais:

1. Quando o estado altera: Quando o estado interno de um componente muda, o React re-renderiza o componente para refletir essa mudança.
2. Quando a propriedade altera: Se uma propriedade passada para o componente é modificada, como quando um post é editado, o React re-renderiza o componente para mostrar o novo conteúdo.
3. Quando um componente pai re-renderiza: Se um componente pai é re-renderizado, todos os seus componentes filhos também serão re-renderizados, mesmo que os filhos em si não tenham sofrido mudanças.

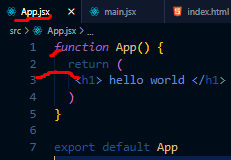
Como as keys ajudam:

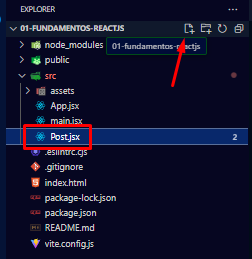
Quando você tem uma lista de itens no React, as keys servem para dar uma "identificação" única para cada item. Isso ajuda o React a saber qual item mudou quando algo na lista é alterado. Assim, em vez de re-renderizar toda a lista, ele só re-renderiza o item que mudou, deixando o restante como está. Isso torna o app mais rápido.

Então, a key deve ser única para que o React possa saber exatamente qual item precisa ser atualizado. Geralmente, o ID do banco de dados é uma boa escolha. Se usar o índice do array como key, pode haver problemas, porque o índice pode não mudar mesmo quando os itens da lista são reordenados ou alterados.

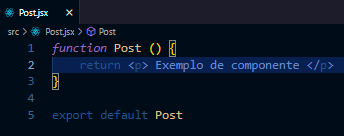
Usando keys corretamente, o React evita renderizações desnecessárias, melhorando a performance do seu aplicativo.

**Identificando componente no React:**

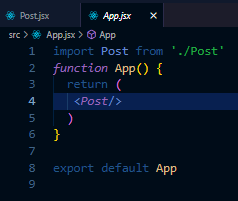
Basicamente uma função que retorna HTML. (JSX = JS + XML):  


Criando, exportando, importando e usando um componente:  


Crie um novo arquivo com a extensão .jsx, exemplo o Post .jsx a cima. **Os componentes sempre devem começar com LETRA MAIÚSCULA**

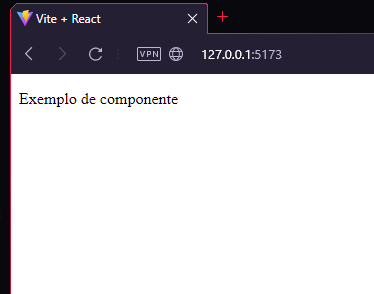


Crie a função com o nome do componente e retorne o HTML desejado, no caso o <p>. Depois no final do código, exporte com o nome da função



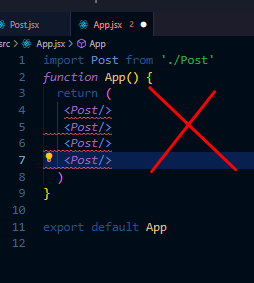
No seu app/pagina onde quer utilizar o componente importe o componente Post (leve em consideração o caminho)

Depois adicione o componente como se fosse uma TAG XML.

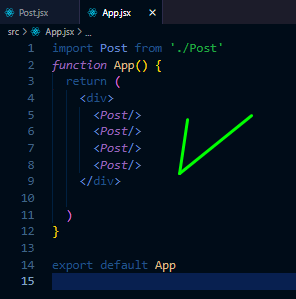


Resultado a cima.

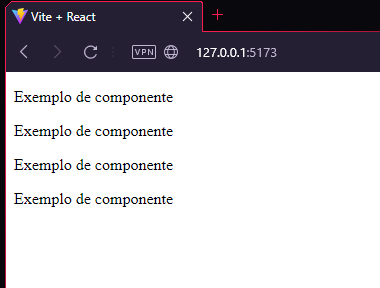
Podemos chamar o componente Post no App quantas vezes quisermos, mas se for utilizar mais de um componente, e necessário ter um elemento pai de referência, ou seja, apenas inserir uma div;



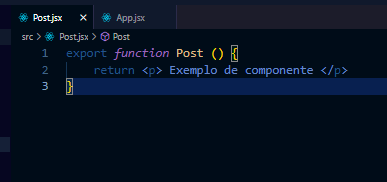
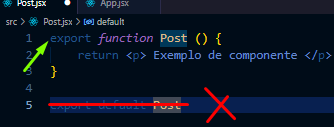
O exemplo a cima não funciona e o baixo funciona.



Resultado:

  
**IMPORTANTE: O método de importação a cima pode acarretar erros, pois o nome do import pode ser alterado e ignorar o nome do export, fazendo com que você possa confundir os components.**

**Seguir método a baixo:**



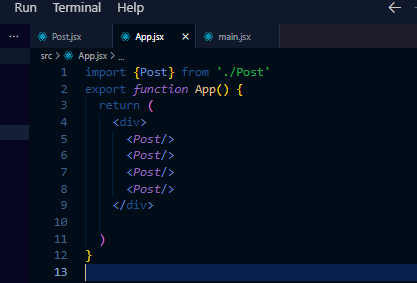
Apenas insira um export antes da function e exclua a exportação padrão.  
Para importar basta inserir um import:

(aperte CTRL + ESPAÇO irá mostrar todos os components disponíveis para utilizar)



(Sempre considere o diretório ./CAMINHO’)

Resultado final:

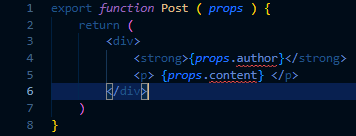


caso você esqueça de alterar o nome do componente na hora de importar, a importação não irá funcionar, diferente do default export, que importaria do mesmo jeito.

**Propriedades dos components:**

Propriedades são informações/conteúdo que eu posso passar aos componentes. Estas propriedades podem ter um visual semelhante o que justifica criar o componente, mas pode conter um conteúdo diferente.

Como no HTML a tag <img src=’exemplo.png’ > tem o search que mostra o caminho da imagem, no mesmo princípio que pode ter a mesma imagem várias vezes ou imagens diferentes as propriedades dos componentes também funcionam da mesma forma.

Na pratica:

|  |
| --- |
| No React o parâmetro **props** (props de propriedade, poderia ser outro nome, mas geralmente e utilizado props pela comunidade React, clean code, boas práticas etc.)  Com isso o componente passa a ser um objeto. |
| Agora podemos deixar cada tag editável, nomeando uma propriedade única para cada parte do componente. Basta chamar o objeto props e adicionar um ‘.nomeDaProriedade’ |
| props.author virou uma propriedade, podendo ser alterada toda vez que se chamar o componente.  E o props.content também. |

No React não se importa CSS no HTML e sim no JSX

Escopo de CSS atrelados ao componente, para estilizar somente aquele componente, sem afetar outros componentes, CSS ESCOPADO: [CSS MODULES](https://github.com/css-modules/css-modules)

Não é necessário baixar pois Vit já contém CSS modules.

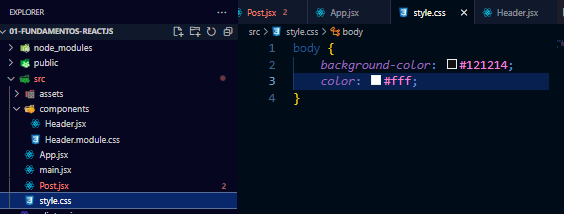
Cria o arquivo com a extensão module e CSS ex: style.module.css

Quando utilizar .module sempre chame os marcadores por classe .nomedaClasse, sempre que possível.

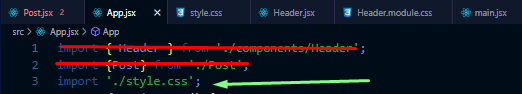
No COMPONENTE.JSX Utilizar ClassName em vez de class do HTML pois o React pode confundir o ‘class’ que também existe no JS.  
Sempre que utilizar nome.modules.css e necessário nomear o import do style, diferente se usar apenas o nome.css que basta apenas inserir o caminho.

CSS Modules criam classes com nomes únicos usando hashes, evitando que estilos de um componente afetem outros. Funciona de forma parecida das propriedades do React.

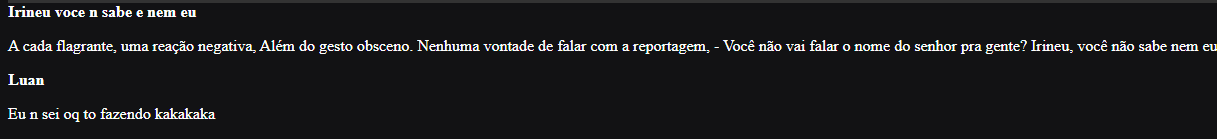
Passo a passo, estilização simples no CSS:

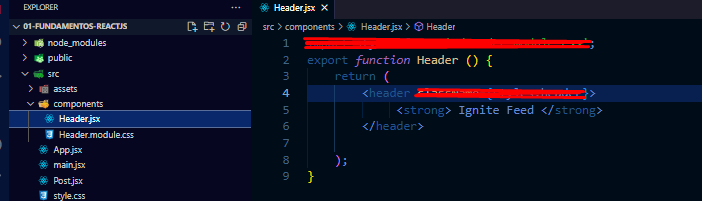
Primeiro crie um arquivo style.css em SRC, mesmo nível no arquivo principal App.jsx e faça uma simples estilização.  


Importamos ele diretamente no App.jsx e não no HTML.

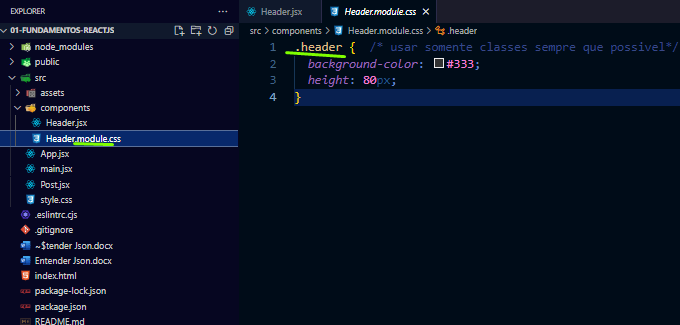


Observe que é apenas necessário listar o caminho.  
Com isso já obtemos o resultado desejado:



Porém esse método irá estilizar o App.jsx como um todo. O React + Vit nos permite editar cada componente independentemente sempre que quisermos e reutilizar esses componentes com estilos diferentes, parecido as propriedades que vimos anteriormente.  
utilizando module.css, veja:  
Primeiro criamos um novo componente.  


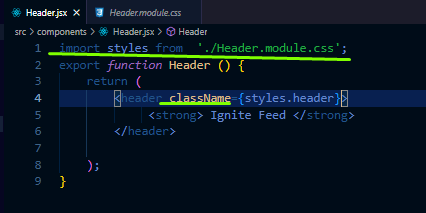
Agora criamos o arquivo style, mas desta vez, adicionamos a extensão .module.css em vez de apenas .css:

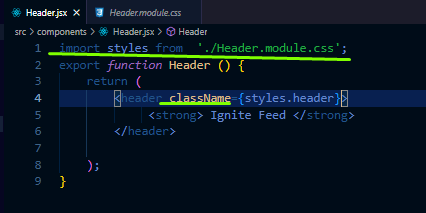
E inserimos a estilização desejada.  


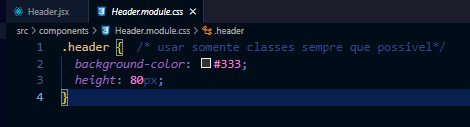
Observe que chamamos pelo identificador de classe .header  
Sempre que possível iremos chamar pela classe por boas práticas.

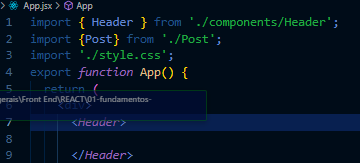
De volta ao componente iremos importar o css, mas desta vez o import precisa de um nome.

E ao inserir o identificador de ‘class’ iremos colocar de ‘className’ para o React não confundir com class de JS.

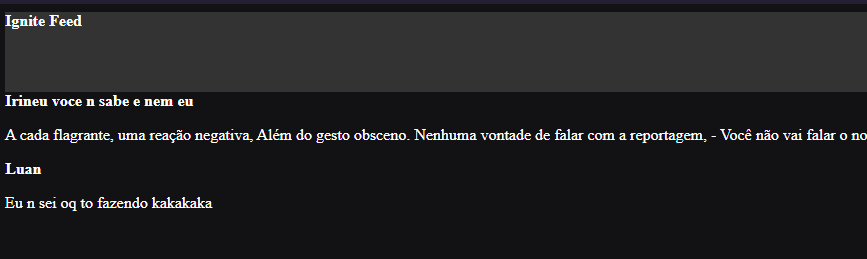


Observe que importamos o style igual um componente e também o utilizamos igual uma propriedade, usando o nomeDaImportação.nomeDaClasseCss.  
observe:

Parte inferior do formulário

Agora basta chamar o componente criado no App.jsx, e ver o resultado:  


Resultado:

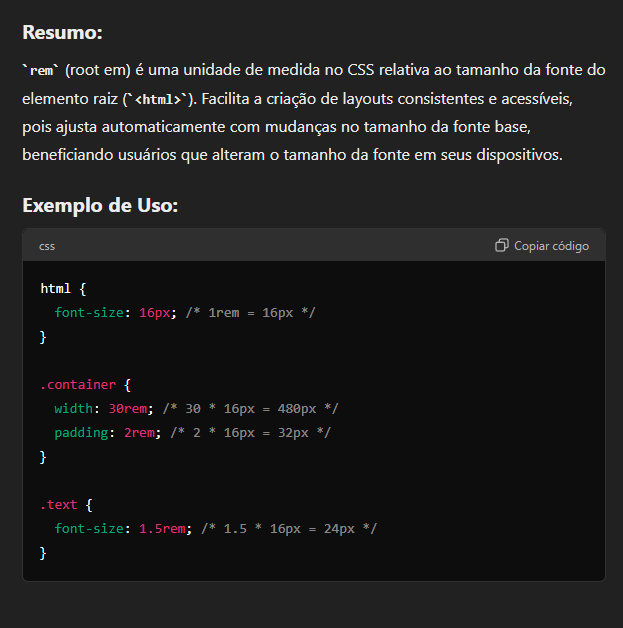


Utilizar variáveis para todas as cores.

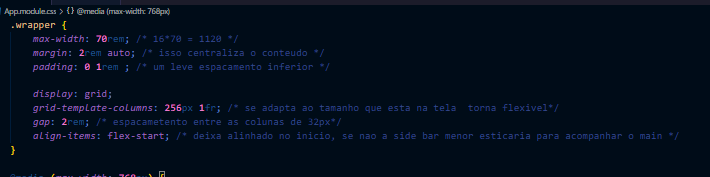
Se tiver mais de um tom, faça uma numeração de 100 a 1000 para definir o tom do mais claro ao mais escuro ex: --gray-100 --gray-300 etc.

Use os links do google diretamente no HTML, mas deixe eles em primeiro, para garantir que eles carreguem antes da aplicação.

Body, input, textarea, button: Padronizar o tamanho unidades de medidas relativas ex: 1rem, por acessibilidade, por exemplo pessoas aumentam a fonte do celular, nosso app também vai acompanhar.



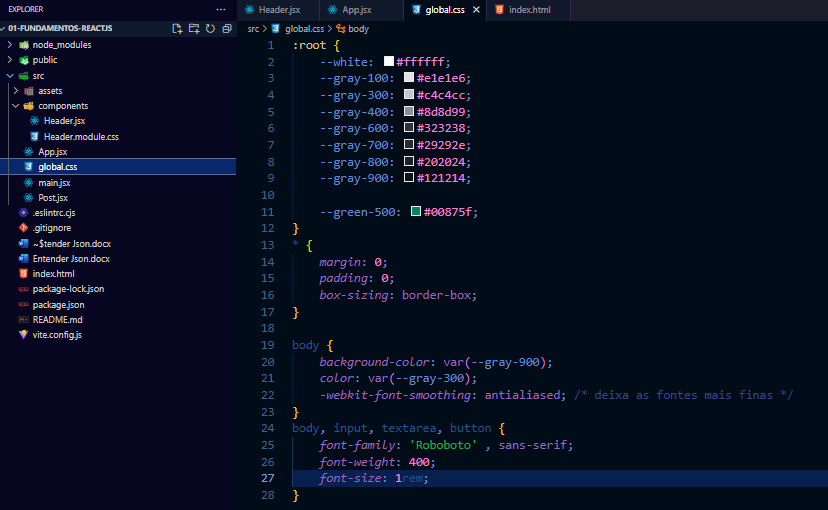
Iniciando o projeto:

Primeiro definimos algumas estilizações globais em App.module.css:  


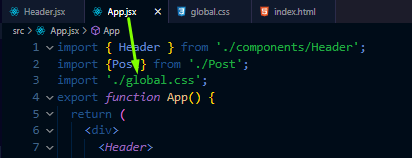
Agora criaremos algumas variáveis (outras serão inseridas conforme necessidade)

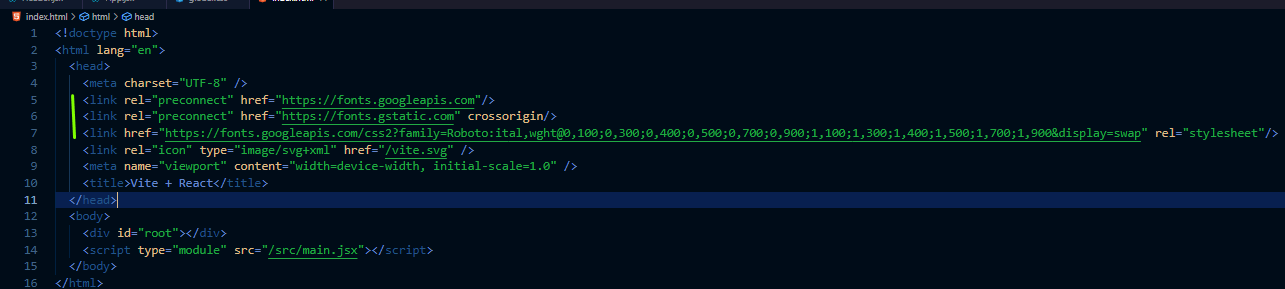
Algumas coisas na prática fica um padrão, por isso iremos renomear o arquivo style.css criado antes para global.css

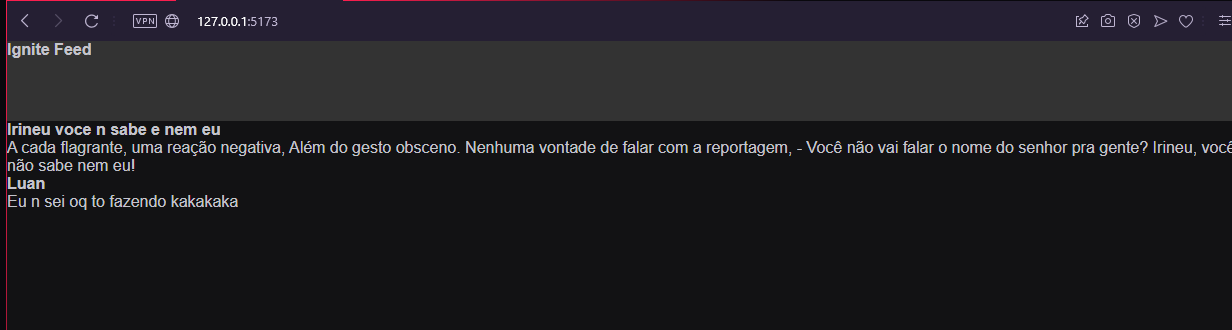
E inserir variáveis de estilo padrão e alguns padrões de inputs, textos, botão etc:



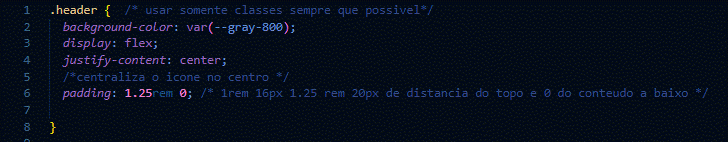
Na importação do App.jsx também alteraremos o nome.



Iremos também pegar a fonte no google fonts roboto e importar no HTML antes de todos os links para evitar erros:  


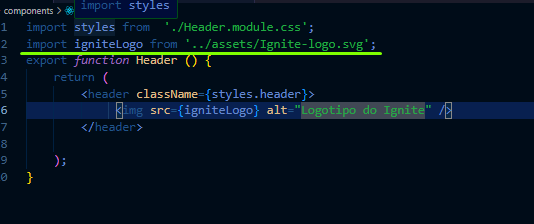
Resultado:  


Agora iremos construir esse layout do figma: [Layout do curso.](https://www.figma.com/design/3wo7UATnhuqTyZcdxSGSkF/Ignite-Feed-(Community)?node-id=1-35&t=r2y9HH0M5lp0XgMp-0)  
Começando pelo básico o logo do cabeçalho. (header).

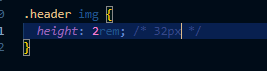
Em Header.module.css  


Esse trecho centraliza o cabeçalho e deixa o conteúdo distanciado do topo.

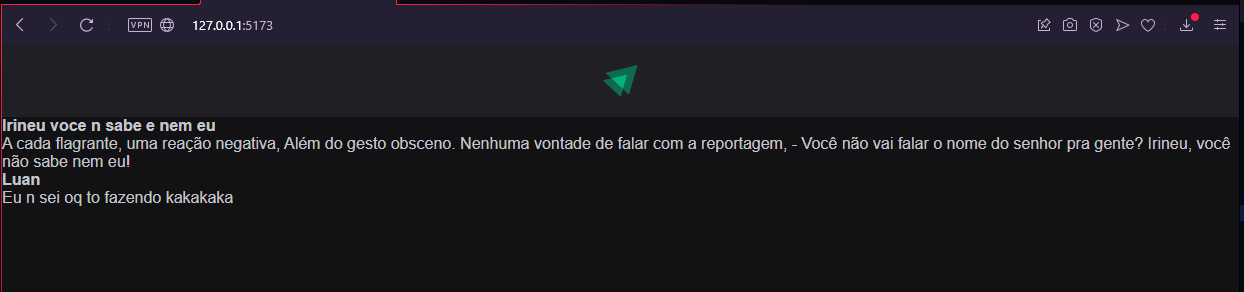
exportamos o logo do figma no formado svg e salvamos em assets e renomeamos para ‘Ignite-logo’

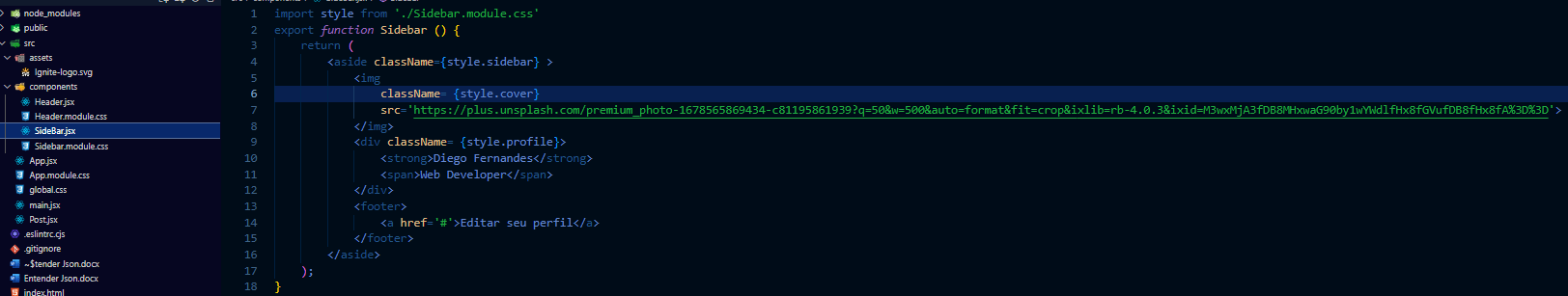
Para usarmos a imagem, importamos ela igual um estilo ou um componente:  


E Chamamos ela como uma variável como nos componentes e estilos.

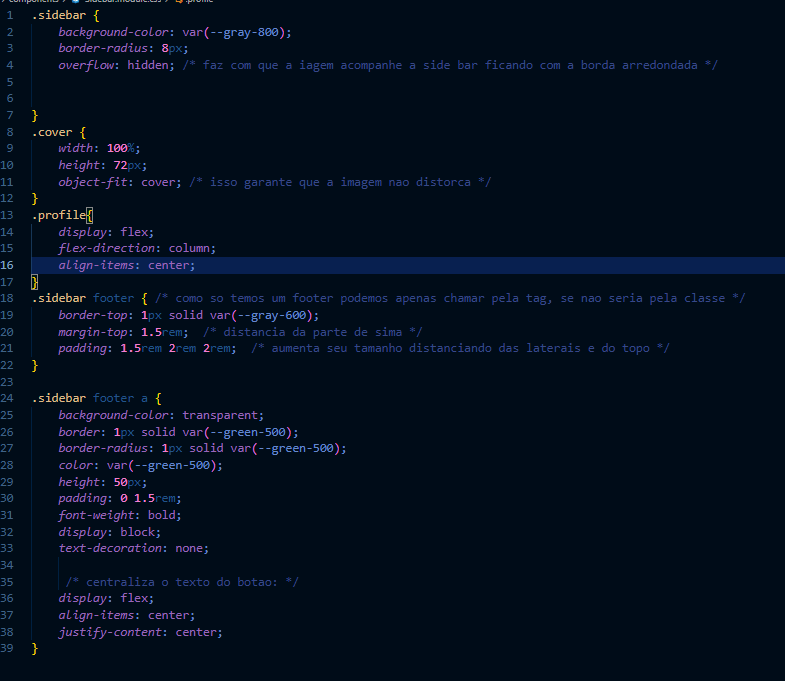
Vamos diminuir um pouco o tamanho do logo:  


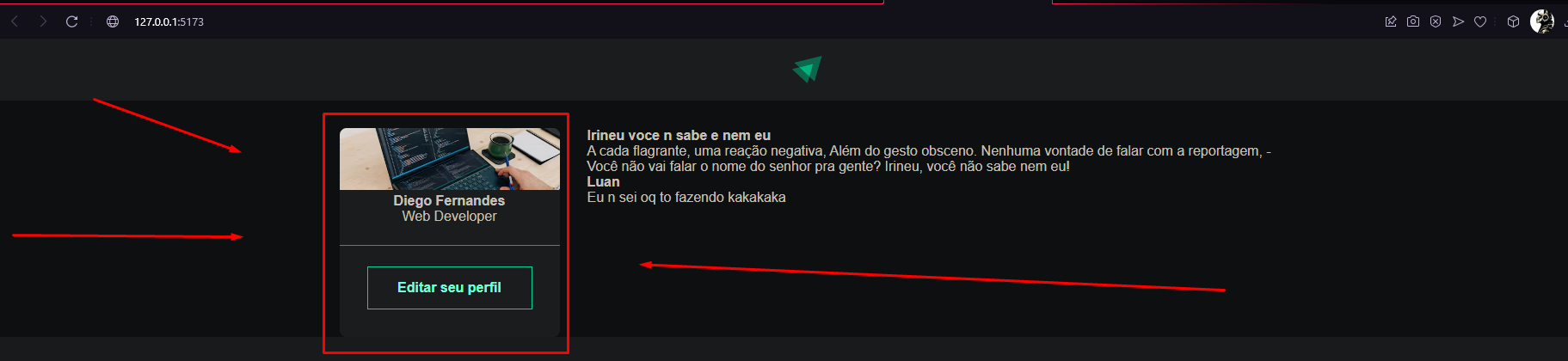
Resultado:



Continuando criando o componente SideBar (um espaço onde fica o perfil).  
Crie o SideBar.jsx e Sidebar.css  
em SideBar.jsx:  


A imagem uma <img> seria a imagem que o usuário colocaria de fundo imagem retirada de bancos de dados gratuitos, depois apenas um título que vai conter o nome e o cargo.



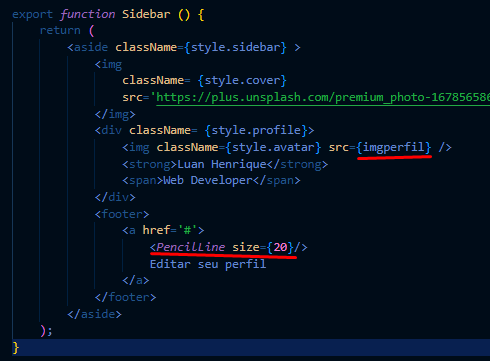
O css a cima separa a side bar e sempre utilizo unidade de medidas rem  
resultado destacado em vermelho:  


Continuando irei inserir uma imagem de perfil e um ícone.

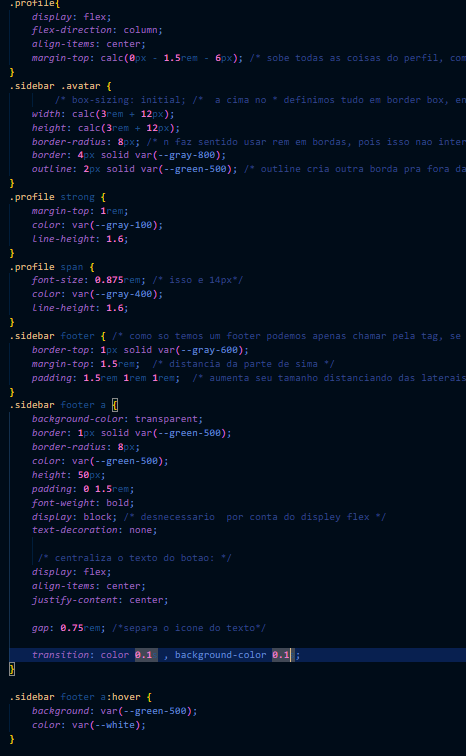
No caso usei o [phosphor-icons](https://github.com/phosphor-icons/homepage), onde apenas e necessário instala-lo no diretório raiz do projeto com npm i phosphor-react e depois importamos o ícone exato que queremos, em vez de importar uma pagina inteira de ícones, só importamos o que iremos usar.



Penciline e o nome do ícone, chamamos ele como um componente em nosso código e a imagem como variável:



Alteramos alguns detalhes para o tamanho da imagem atravessar a div de sima, colocando uma margem negativa, com um cálculo e uma animação de transição



O css a cima estiliza o sideBar, não tem algum CSS muito complexo para explicação:

Continuando a análise do CSS da barra lateral:

**profile:** Define um container para os elementos do perfil, organizando-os em uma coluna e centralizando-os horizontalmente. A margem negativa aplicada ajusta a posição do perfil em relação à imagem de capa, criando um efeito de sobreposição.

**sidebar .avatar:** Estiliza o avatar do usuário, definindo tamanho, forma, bordas e contorno. O cálculo do tamanho utiliza a função calc() para garantir um espaçamento consistente.

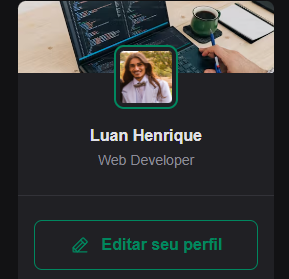
**profile strong:** Aplica estilos ao nome do usuário, como cor, espaçamento entre linhas e margem superior.

**profile span:** Estiliza uma descrição ou outro texto relacionado ao perfil, definindo fonte, cor e espaçamento entre linhas.

**sidebar footer:** Cria um rodapé para a barra lateral, adicionando uma borda superior e espaçamentos internos.

**sidebar footer a:** Estiliza o link dentro do rodapé, que provavelmente serve como um botão de ação. Define a aparência do botão, incluindo cor, borda, padding, transições e centralização do conteúdo.  
**.sidebar footer a:hover:** Define o estilo do botão quando o mouse passa sobre ele, alterando a cor de fundo e da fonte.

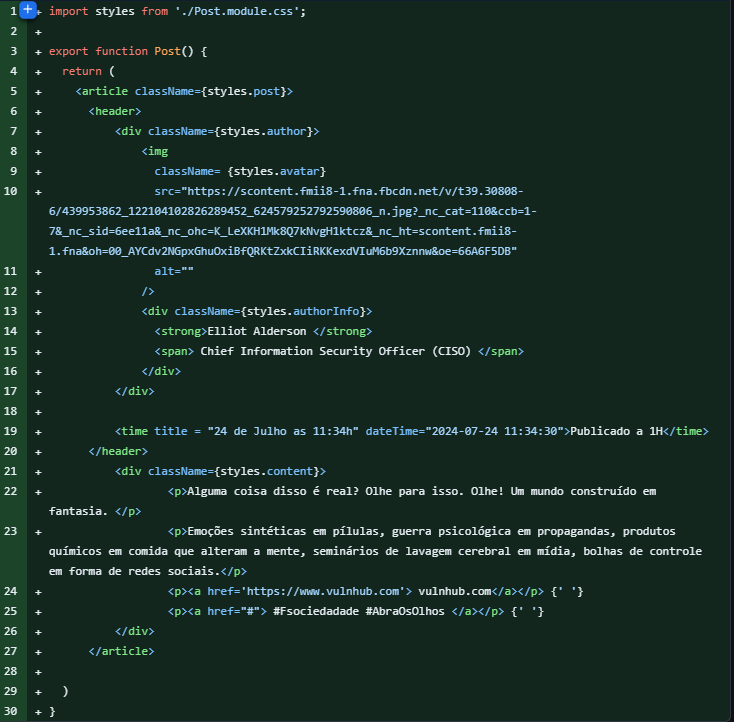
Resultado:



[Commit: Side Bar](https://github.com/luanhsr/ignite-reactjs-01-fundamentos-react-ALUNO/commit/74b9ecb19dedf2ee5f7ad055fdb646e445e48109)

Criamos agora o Post.jsx e Post.module.css em componentes e excluímos o antigo Post.jsx usado apenas para aprender conceitos de componentes.

O componente Post.jsx oferece uma estrutura sólida e flexível para a criação de posts em aplicações React. Ele combina a semântica do HTML com a personalização do CSS para criar uma experiência de usuário consistente e acessível.

Sua estrutura HTML:  


import styles from './Post.module.css';: Importa as classes CSS definidas no arquivo Post.module.css, permitindo aplicar estilos personalizados ao componente.

Função do Componente:

export function Post(): Define a função que representa o componente Post.

Estrutura do Post:

**article:** O elemento principal article envolve todo o conteúdo do post, indicando semanticamente que se trata de um conteúdo independente.

**header:** Contém as informações do cabeçalho do post, como autor, data de publicação e outras metadados.

**div.authorInfo:** Contém as informações sobre o autor do post, incluindo a imagem do avatar e o nome.

**img.avatar:** Renderiza a imagem do avatar do autor.

**div.authorInfo:** Contém o nome do autor e sua função.

**time:** Representa a data e hora de publicação do post, com atributos title e dateTime para fornecer informações adicionais aos mecanismos de acessibilidade e aos bots de busca.

**div.content:** Contém o conteúdo principal do post, que pode incluir texto, links e hashtags.

+ SOBRE A TAG TIME E ACESSIBILIDADE:  
Para usuários que precisam de acessibilidade vizual utilizando um leitor de tela para navegar em um site. Ao encontrar a tag <time>, o leitor irá interpretar a data e a hora de publicação do post de forma clara e concisa para o usuário. Isso significa que pessoas com deficiência visual poderão saber exatamente quando um post foi publicado.

Outros detalhes que ao passarmos o cursor em cima da tag time, ela também aparece visualmente o horário exato do post.

[Commit: Estrura Post HTML](https://github.com/luanhsr/ignite-reactjs-01-fundamentos-react-ALUNO/commit/c6b3d93efe5ef01164ed5541fe62fb1f59e281ca)



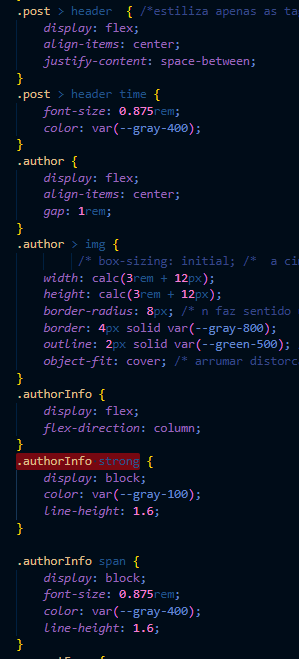
**.post:** Define o estilo geral do post, incluindo fundo, bordas arredondadas e espaçamento interno. Aplica margem superior aos posts subsequentes.

**.content:** Estiliza o conteúdo do post, definindo cor, espaçamento de linhas e margens.

**.content p, .content a:** Estiliza parágrafos e links dentro do conteúdo.

**.post + .post** Essa tag, faz apenas estilizar o post adjacente, ou seja, apenas o post que esteja após outro post.

**.content a:hover** Troca a cor de fundo do link após passar o mause em cima.



.post > header: Define o layout do cabeçalho do post, posicionando elementos horizontalmente.

.post > header time: Estiliza a informação de data e hora do post.

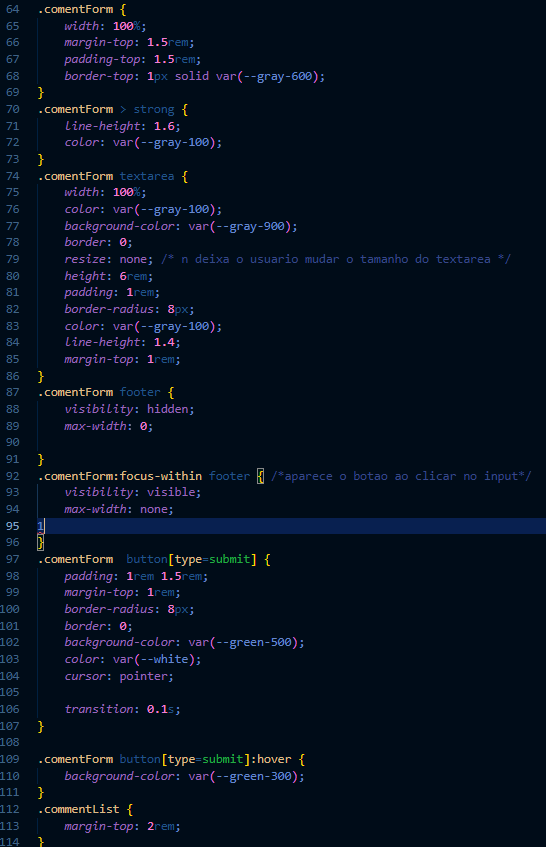
.author: Controla o layout da informação do autor, incluindo espaçamento entre imagem e texto.

.author > img: Estiliza a imagem do avatar do autor.

.authorInfo: Organiza o nome e a função do autor em uma coluna.

.authorInfo strong: Define a estilização do nome do autor (em negrito).

.authorInfo span: Define a estilização da função do autor.



.comentForm: Define o estilo geral do formulário de comentário.

.comentForm > strong: Estiliza o título do formulário de comentário.

.comentForm textarea: Estiliza o campo de texto para o comentário.

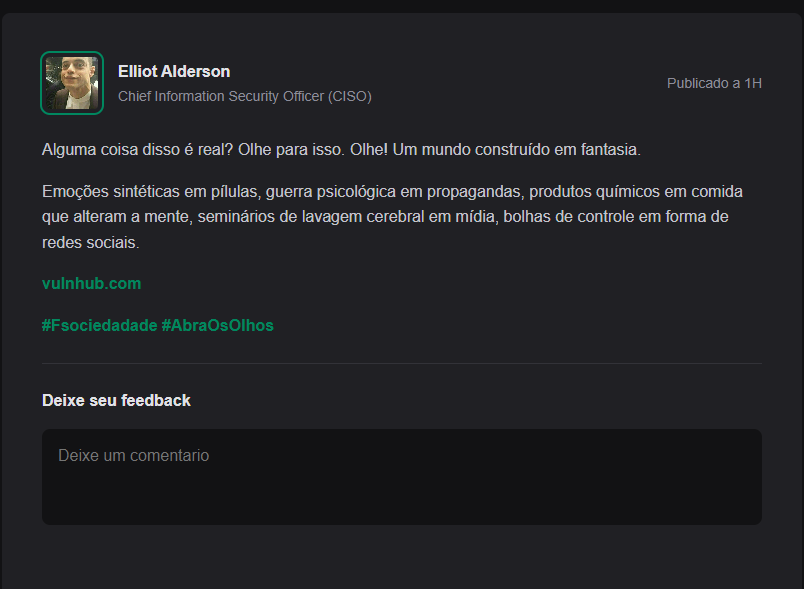
.comentForm footer: Controla a visibilidade do rodapé do formulário (onde fica o botão de enviar).

.comentForm:focus-within footer: Mostra o rodapé do formulário quando o usuário clica dentro dele.

.comentForm button[type=submit]: Estiliza o botão de envio do comentário.

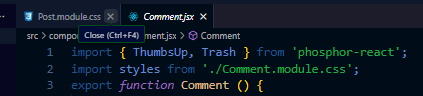
.comentForm button[type=submit]:hover: Altera o estilo do botão quando o mouse passa sobre ele.

.commentList: Define a margem superior da lista de comentários.

Resultado:  


[Commit: Estilização do Componente Post](https://github.com/luanhsr/ignite-reactjs-01-fundamentos-react-ALUNO/commit/ee11a71cf30e630b775018cc5c28298c9fbac9c9)

Agora criamos o componente comment.jsx e o estilo comente.module.css para podermos criar a parte que os usuários comemtam:

****Importa os ícones de curtir (ThumbsUp) e excluir (Trash) da biblioteca Phosphor-React, que serão utilizados para ações do usuário.  
Importa os estilos CSS definidos no arquivo Comment.module.css, que serão aplicados aos elementos do componente.  
E inicia a função principal do componente comentário.

**<div className={styles.comment}>:** Cria o container principal do comentário, aplicando os estilos definidos em Comment.module.css.

**<img src="...">:** Insere a imagem do perfil do usuário que fez o comentário.

**<div className={styles.commentBox}>:** Cria um container para o conteúdo do comentário, incluindo o cabeçalho, o corpo e o rodapé.

**<div className={styles.commentContent}>:** Contém o conteúdo principal do comentário.

<header>: Contém as informações do cabeçalho do comentário.

**<div className={styles.authorAndTime}>:** Contém o nome do autor e a data do comentário.

**<strong>:** Exibe o nome do autor em negrito.

**<time>:** Exibe a data e hora do comentário.

**<button>:** Botão para excluir o comentário (<*Trash* /> ícone de Lixeira).

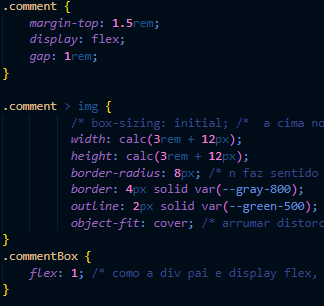
**<p>:** Contém o texto do comentário.

**<footer>:** Contém os botões de interação do usuário.

**<button>:** Botão para curtir o comentário (<*ThumbsUp* /> ícone de para cima).

[Commit: estrutura componente comment](https://github.com/luanhsr/ignite-reactjs-01-fundamentos-react-ALUNO/commit/783e487174dfce473f0d7d360545387a99c4f1b9)

Agora estilizamos:



**.comment:** Define o container principal do comentário, estabelecendo o espaçamento entre a imagem do usuário e o conteúdo do comentário.

**.commentBox:** Ocupa o espaço restante dentro do container principal e serve como um wrapper para o conteúdo do comentário.

**.comment > img:** Define o estilo da imagem do usuário, incluindo tamanho, bordas, formato e posicionamento.



**.commentContent:** Define o estilo do conteúdo do comentário, incluindo background, bordas e padding.

**.commentContent header:** Define o estilo do cabeçalho do comentário, posicionando os elementos horizontalmente.

**.authorAndTime:** Controla o layout do nome do autor e da data do comentário.

**.authorAndTime strong:** Define o estilo do nome do autor.

**.authorAndTime time:** Define o estilo da data do comentário.

**.commentContent header button:** Estiliza os botões do cabeçalho (excluir), removendo estilos padrão e adicionando um cursor para indicar interação.

**.commentContent p:** Define o estilo do parágrafo do comentário.

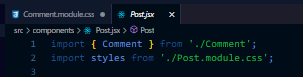
**.commentBox footer:** Define o estilo do rodapé do comentário, onde se encontra o botão de curtir.

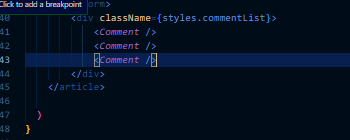
**.commentBox footer button:** Estiliza o botão de curtir, similar aos botões do cabeçalho.

**.commentBox footer button svg:** Estiliza o ícone dentro do botão de curtir.

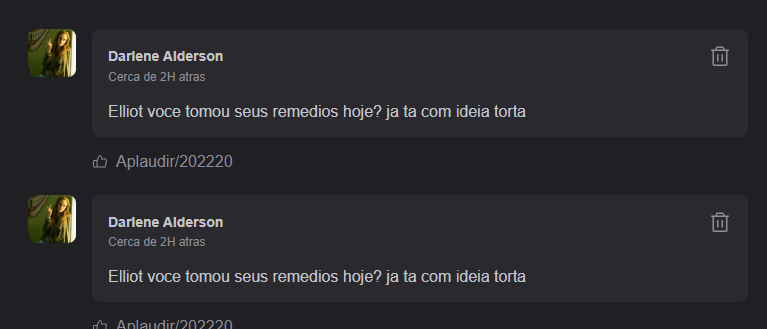
**.commentBox footer button span::before:** Adiciona um conteúdo antes do texto "Aplaudir", simulando um ícone.

Após conclusão, vamos ao Post.jsx e importamos e inserimos o commente.jsx:





Resultado:



[Commit: Estilização componente comentário](https://github.com/luanhsr/ignite-reactjs-01-fundamentos-react-ALUNO/commit/ccd6abdb114de83a33312bd952a90273ab19ec8b)

Observe que já criamos o Avatar 3 vezes no nosso projeto da mesma maneira, com exceção do detalhe, do comentário onde não possuí borda.

Seguindo a regra dos componentes, vamos criar um componente Avatar

Crie Avatar.jsx e avatar.module.css

Em Avatar.jsx:



O componente acima usamos a props com desestruturação.

Basicamente, as propriedades é caminho da imagem **src** e a propriedade **hasBorder** que por padrão vem como **True**.

No className temos a lógica se o **hasborder** existir, ativa o estilo styles.avatarWithBorder se não ativa o estilo styles.avatar

Esses estilos foram criados em Avatar.module.css

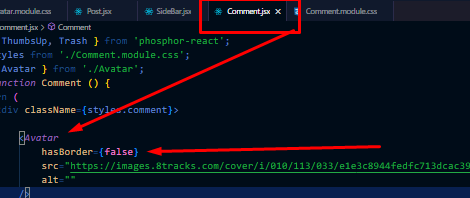
Veja abaixo:



Com a borda e sem a borda.

Agora trocamos as imagens que representavam os avatares dos componentes e inserimos o componente avatar.



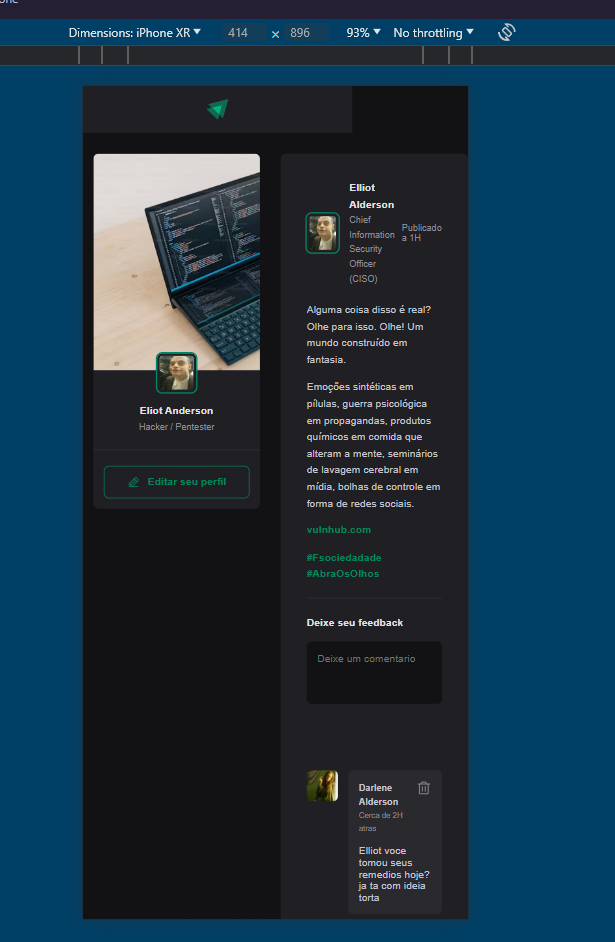


Observe que apenas no comentário chamo a propriedade **hasBorder** e atribuímos **false** para deixa-la sem borda.

[Commit: Componente Avatar](https://github.com/luanhsr/ignite-reactjs-01-fundamentos-react-ALUNO/commit/888fec61906c303945433e4f161f6f15b61a1669)

**RESPONSIVIDADE**

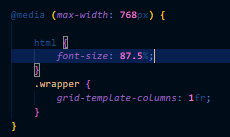
Chegamos em uma parte crucial em todo projeto web, responsividade, pra ter ideia o Google não prioriza sites que não possuem responsividade, observe que não está responsivo atualmente:



Assim seria a visão de um usuário de dispositivo móvel.

Como utilizamos a unidade relativa rem em quase todo o projeto, vai ficar bem mais fácil e poderoso de se editar a responsividade.

Aplicando a responsividade:

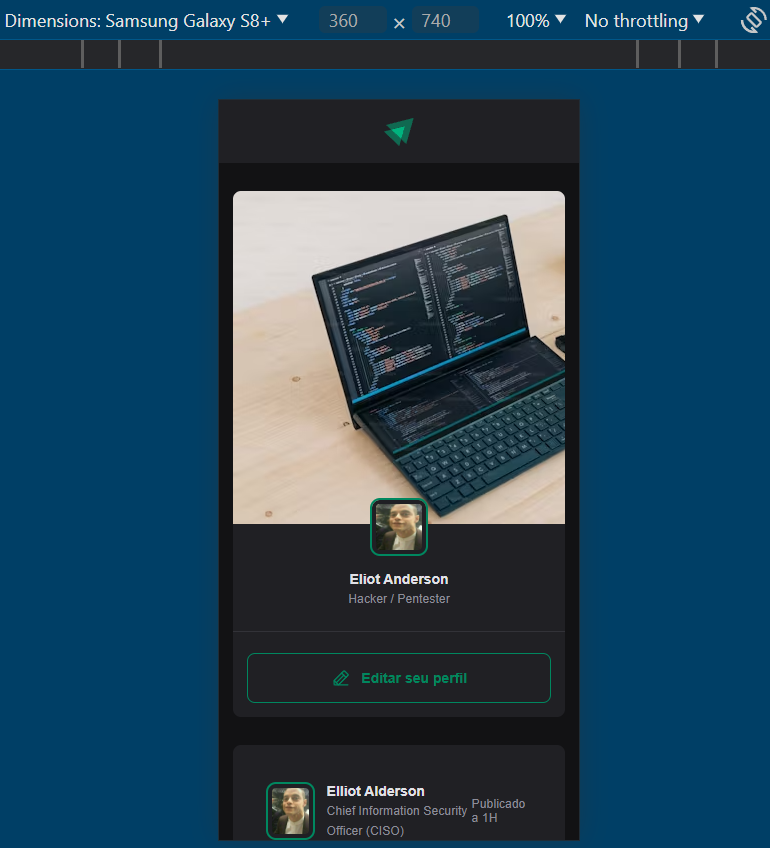
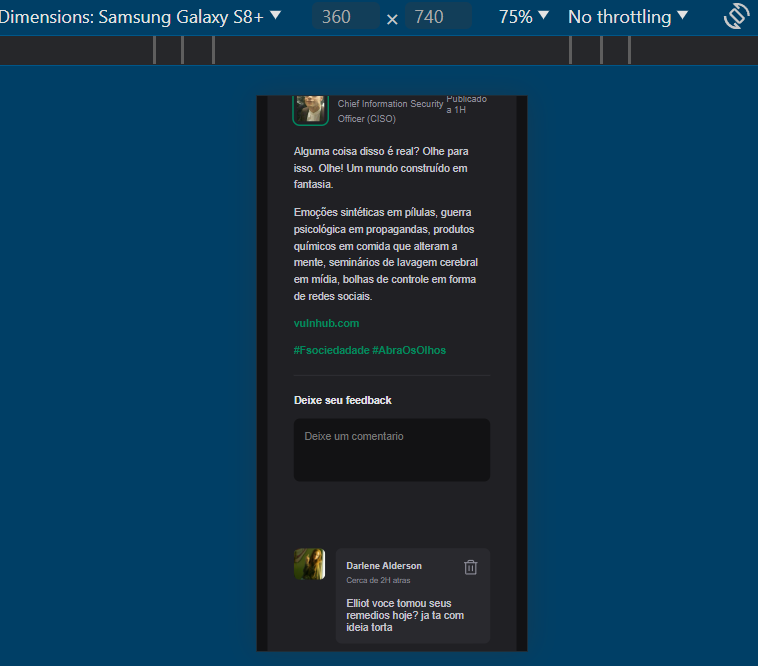
Em nosso projeto vamos em App.module.css e inserimos o seguinte código:  


O Max width seria para telas pequena (dispositivos móveis)

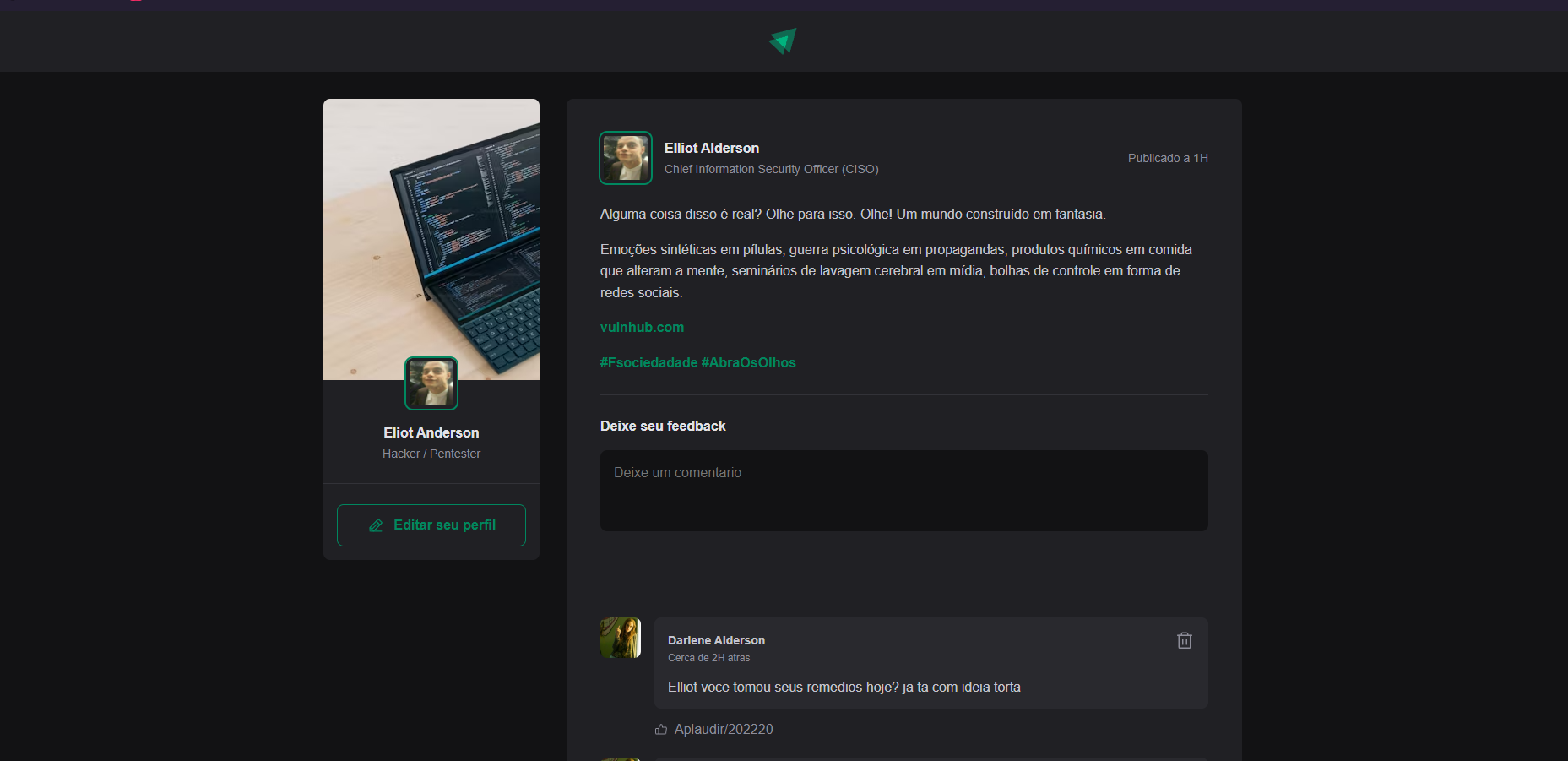
Ajustamos o `rem em dispositivos móveis para garantir que o tamanho do texto se adapte às preferências do usuário, promovendo a acessibilidade. Ao usar porcentagem em vez de `px` o layout responde às configurações de fonte do usuário, como aumentar o texto para facilitar a leitura, todo o app se ajuste proporcionalmente. (imagens, margens, tamanho de objetos etc, pois usamos REM em quase todo app).

O grid-template-colluns, em vez de dividir o app em 2 colunas e realizar todos aqueles ajustes, apenas fazemos uma coluna, observe:

Veja agora a versão responsiva:

Versão de desktop:



[Commit: Aplicando responsividade](https://github.com/luanhsr/ignite-reactjs-01-fundamentos-react-ALUNO/commit/eb138e1c983c0aede2216b0f4cff46c90dbe64af)

**Começando o backeend**

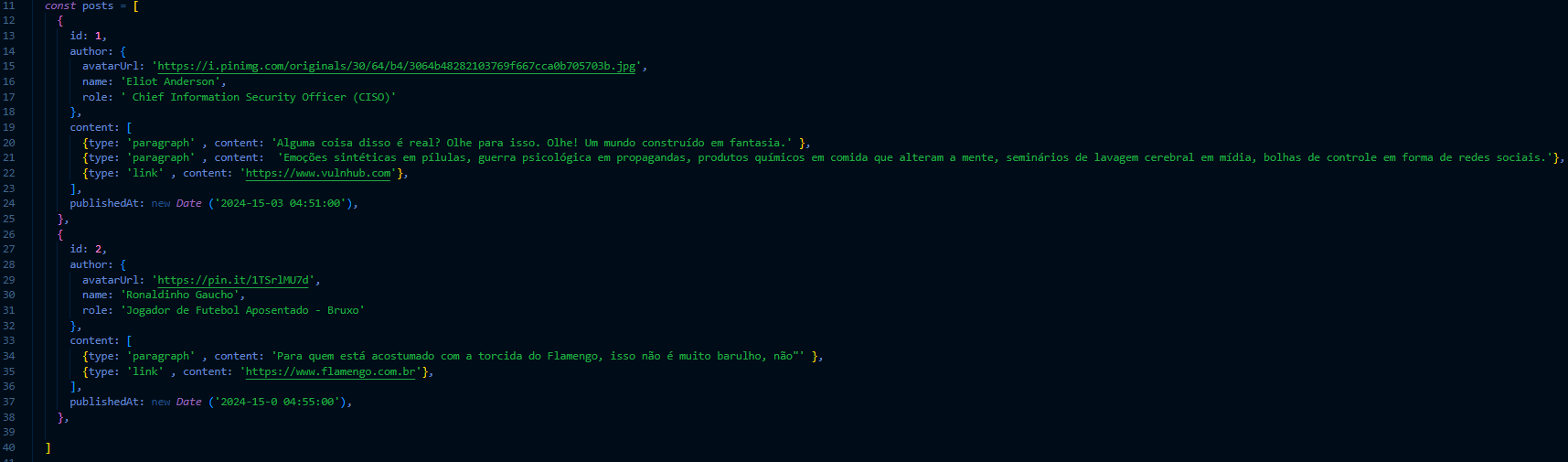
Agora que temos uma boa parte do front finalizada, iremos iniciar o backend, primeiramente um planejamento, começando pelo componente Post



O componente post é composto pelas partes acima, o autor que contém uma imagem, o nome e o cargo, a data da publicação e o conteúdo do post.

Entenda que os dados viriam de um banco de dados e não do array a seguir, mas o exemplo é bem próximo da saída de um banco de dados.

Relembre o conteúdo do Post.jsx em caso de dúvida [Post.jsx](#Post)

O código abaixo cria uma estrutura de dados que simula uma lista de posts, com cada post contendo informações como autor, conteúdo e data de publicação. Essa estrutura é comum em aplicações React para representar dados que seriam obtidos de uma API ou banco de dados.

**const** **posts** = [...]: Cria um array chamado posts que contém objetos, onde cada objeto é uma postagem.

Cada objeto no array representa uma única postagem com as seguintes propriedades:

**id:** Um identificador único para a postagem.

**author:** Um objeto que contém informações sobre o autor da postagem:

**avatarUrl:** URL para a imagem do avatar do autor.

**name:** Nome do autor.

**role:** Papel ou título do autor.

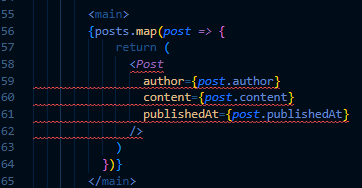
**content:** Um array de objetos que descreve o conteúdo da postagem:

**type:** Tipo de conteúdo, como 'paragraph' (parágrafo) ou 'link' (um link).

content: O texto ou URL específico.

**publishedAt:** A data e hora em que a postagem foi publicada, utilizando o objeto Date.

Agora ao chamar o componente Usamos o método map para iterar o array.  
O componente Post é chamado dentro da função App. A função App é um componente React que retorna a estrutura da interface do aplicativo. Dentro dela, o componente Post é renderizado várias vezes, uma para cada objeto dentro do array posts.



{posts.map(post => { ... })}: Esse trecho de código usa o método map para iterar sobre o array posts. Para cada item no array (cada postagem), ele executa uma função que retorna o componente Post.

<Post ... />: Dentro do map, o componente Post é renderizado. Ele recebe algumas propriedades (ou props) específicas:

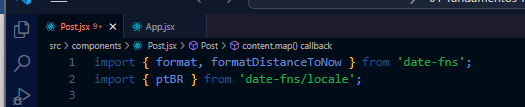
**author**={**post**.author}: Passa o objeto author da postagem atual como uma prop para o componente Post.

**content**={**post**.content}: Passa o array content da postagem atual como uma prop para o componente Post.

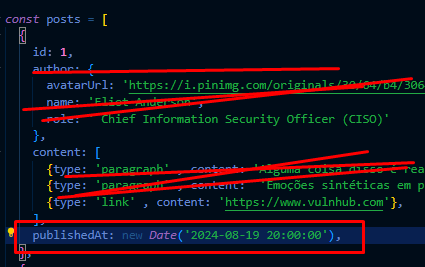
**publishedAt**={**post**.publishedAt}: Passa a data e hora da postagem como uma prop para o componente Post.

(forEach não funcionaria pois esse método apenas itera o array mas não retorna nada, map cria outro array a cada iteração e retorna o conteúdo do índice)

[Commit: Começando o Backend](https://github.com/luanhsr/ignite-reactjs-01-fundamentos-react-ALUNO/commit/b51e3481291787c139a0c929a39a8b10abc6b08e)

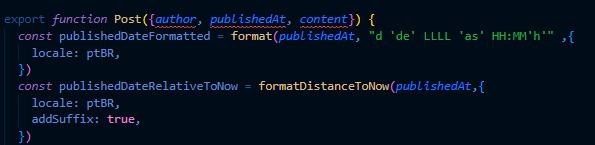
Agora iremos começar a gerenciar o componente post, e consumir os dados do array que simula o banco de dados. Iremos usar uma biblioteca para editar os elementos tipo date, dentro da pasta do projeto insira o código npm install date-fns.  
após a instalação importamos em app.jsx:  


O import é referente a formatação da data inserida no array e formatação da data em comparação com a data atual, também importamos a linguagem pt/br da localização da data.  
Lembre-se que o array representa um banco de dados.



A iremos usar a biblioteca para formatar esse valor e exibir em tela.

Então no nosso Post.jsx começamos a desestruturar a props



Criamos duas constantes uma para armazenar a data formatada e outra para formatar a data com base na distância.

Em caso de duvida consulte a [documentação da API date-fns](https://date-fns.org/v3.6.0/docs/format)

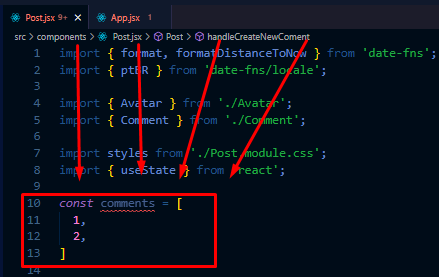
Continuando:  


Apenas inserimos a propriedade **author.avatarURL** e **author.name** e **author.role** no lugar do texto fixo que havia antes.

Agora no conteúdo é necessário percorrer o array **content** com um map para poder tratar cada linha em caso de paragraph apenas exibir o conteúdo em um parágrafo em caso de link inserir o conteúdo dentro de um link dentro de um parágrafo.

[Propriedades do Post](https://github.com/luanhsr/ignite-reactjs-01-fundamentos-react-ALUNO/commit/1a8bc8116d142aeabc2a83341d899c4b938b28c2#diff-99b43677898b2cfde5078535f04c45d16397dfc63a77d39447b901e4980fb359)

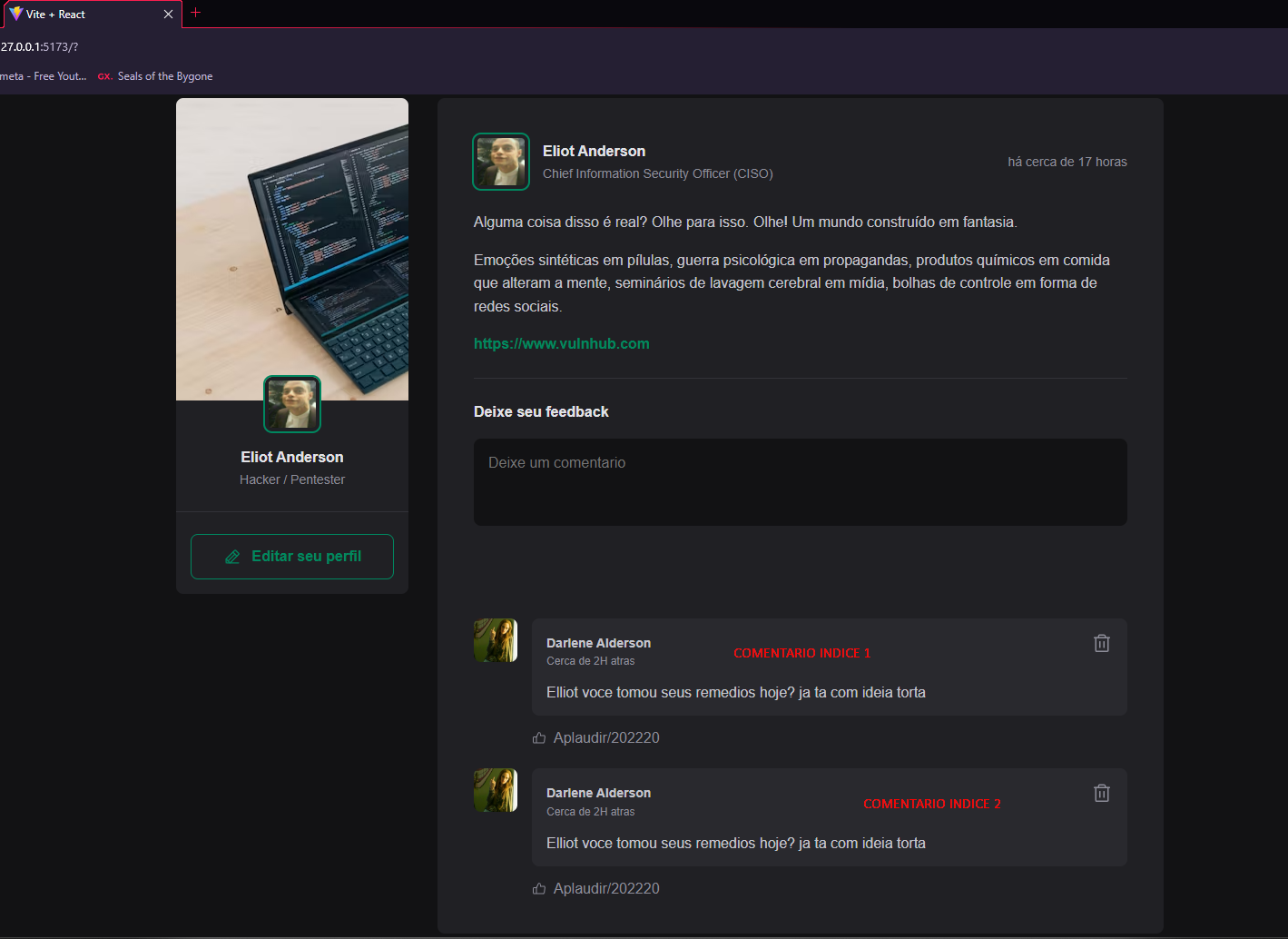
Criando novos comentários, a baixo não iremos fazer a funcionalidade em si de criar comentários, mas a lógica de adicionar um comentário.

Inicialmente como feito anteriormente, criamos um array para representar o banco de dados em App.jsx:  


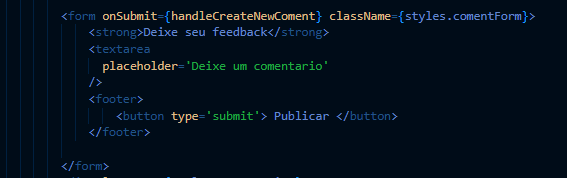
No final do componente onde tínhamos 3 <**Comment**/> iremos usar a mesma função MAP para percorrer o array e retornar um componente comentário para cada índice do array comentário a cima:  


Caso adicione outro índice qualquer no array a cima, irá adicionar outro comentário.

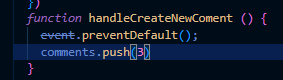
O array representa como se existisse dois comentários:



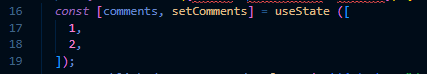
Para atualizarmos o comentário sempre que quisermos precisaremos suar o useState (veja a explicação no comentário do Microsoft Word ao lado).

Criamos a chamada submit no form:  


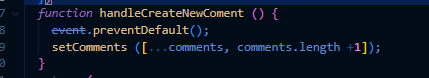
Função:



Apesar dessa função inserir um novo array, não irá atualizar no display, pois o react não fica escutando cada alteração de array ou de variáveis, isso traria vários problemas de performance.

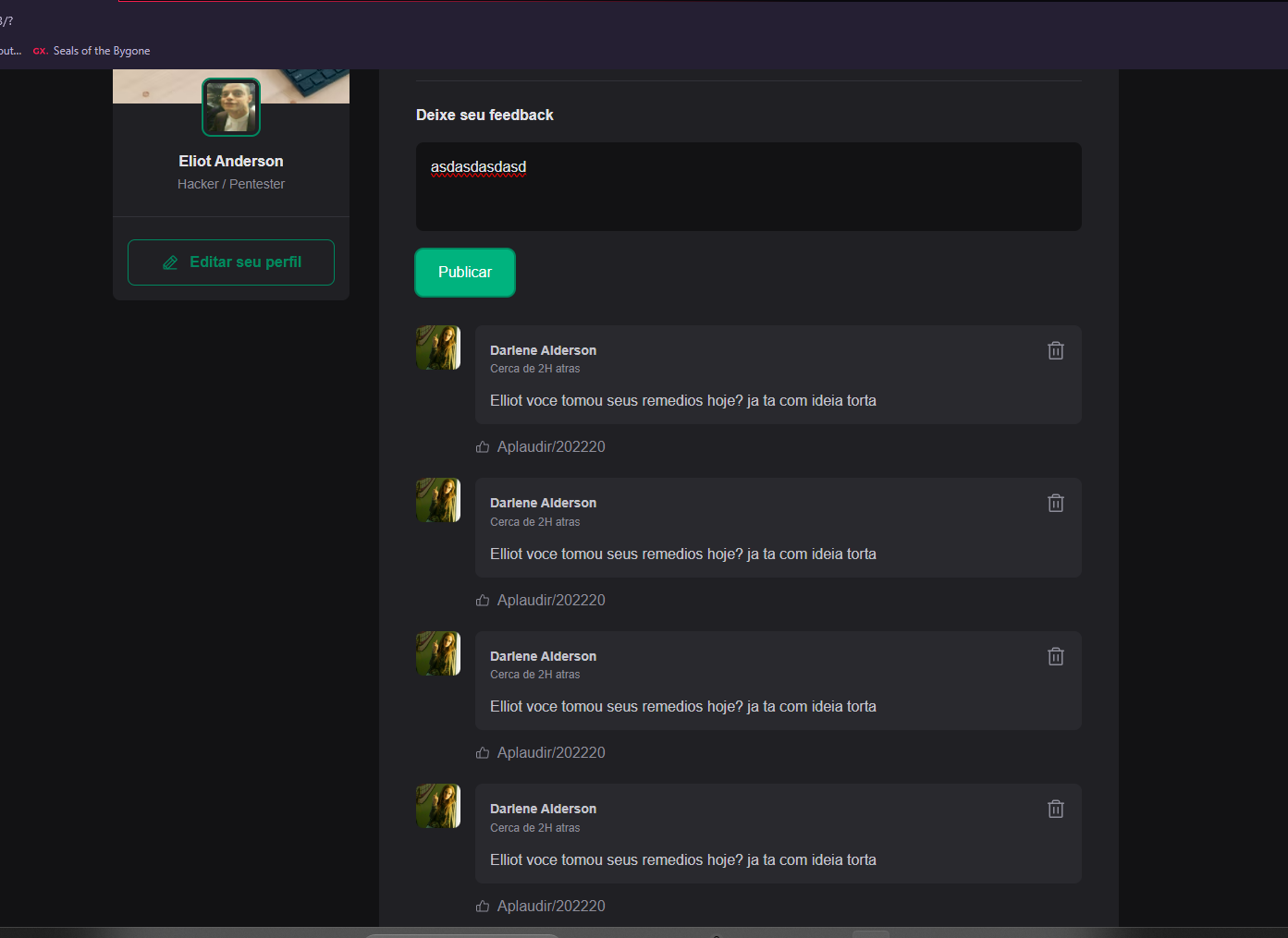
Usamos o use state no array:   


Cada vez que o array for alterado ira disparar a função setComments que agora iremos inserir na função de insert comente:  
então chamamos a função setComent do use state pra cada vez que o usuário inserir um comentário:



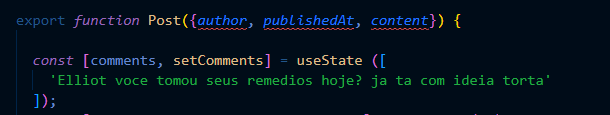
O set Comments verifica o array até o final e adiciona mais um comentário.

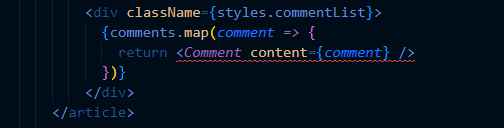
Resultado:

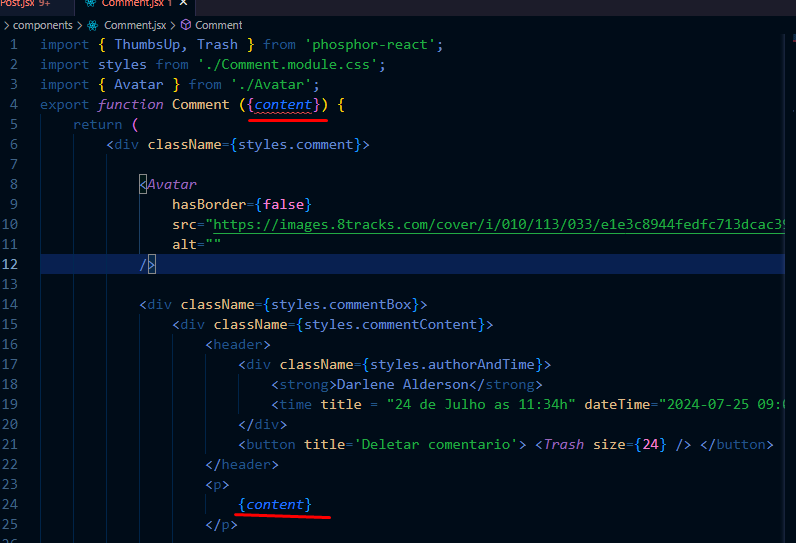


Cada vez que é chamado o submit outro comentário é adicionado.  
[Commit: Adicionando useState no comentario](https://d.docs.live.net/5807ea9bad67f871/Documentos/ESTUDOS/Rockseat.docx)

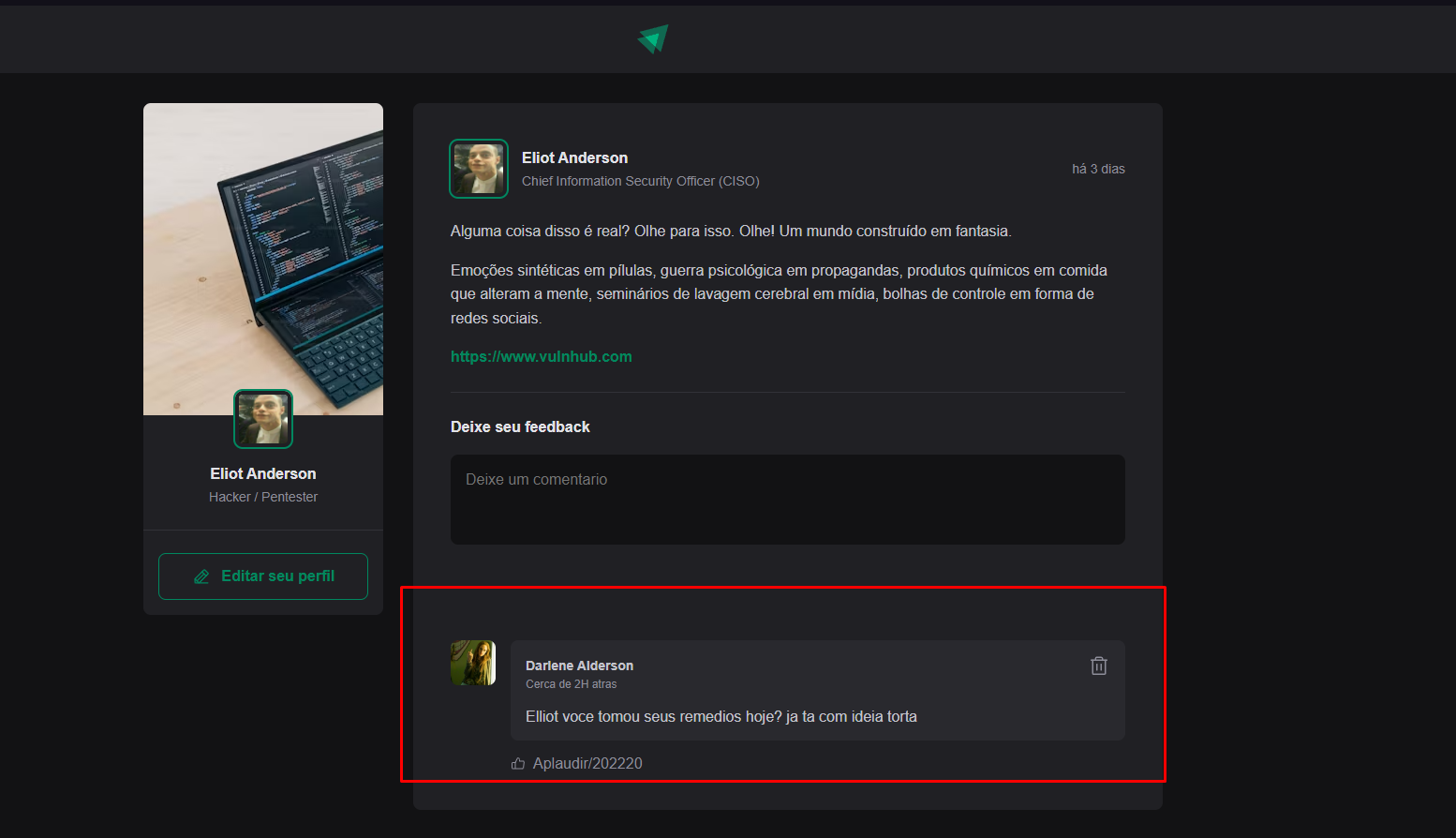
Agora em Post.jsx iremos inserir os comentários conforme é adicionado no nosso array de comentários e conforme o usuário digita.

Primeiro para representar um comentário, iremos trocar o texto do array:  


Nos comentários iremos realizar a função map, porém iremos inserir a propriedade contente que irá receber o conteúdo do comentário:  


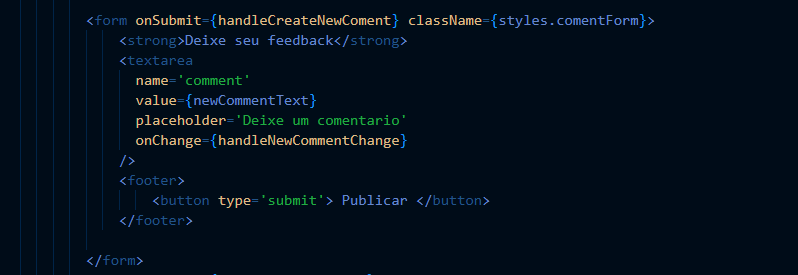
Em Comment.jsx, iremos retirar o comentário estático e iremos inserir a propriedade content (que seria o conteúdo do comentário) mas já iremos desestruturar pois posteriormente iremos inserir outras propriedades:  


Ao olhar o conteúdo já se alterou para o conteúdo que está no useState a cima:

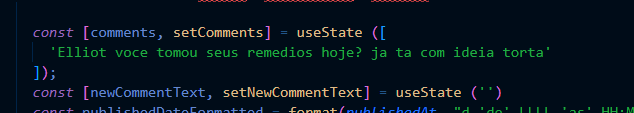


Se Comentarmos agora, irá apenas continuar com o comportamento de iterar o Array:

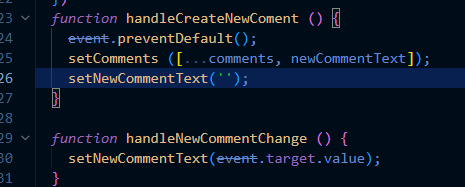
Agora iremos inserir alguns detalhes no formulário do Post.jsx:



* OnSubmit é a função nativa JS/React que irá executar nossa função **handleCreateNewContent** assim que o usuário der um submit
* Adicionamos o name =‘comment para selecionarmos o conteúdo do HTML
* **Value** = {newCommentText} será para armazenar o valor do comentário.
* **onChange** executará uma função sempre que a textarea mudar no caso a função **handleNewCommentChange**



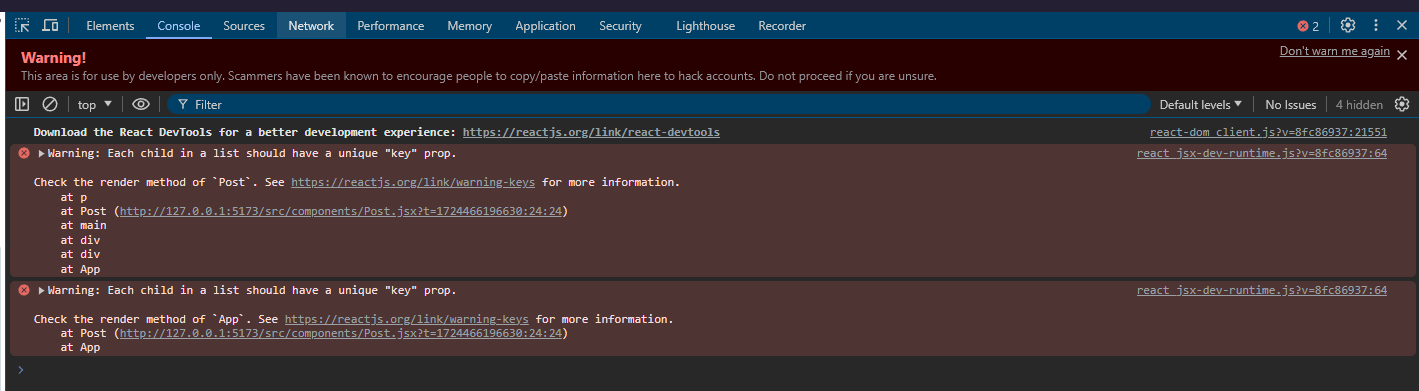
Adicionamos um state ao NewComentText para monitorar nosso comentário



* a Função **handleNewCommentChange**() e disparada sempre que o usuário digita a variável **setNewComentText** recebe o conteúdo desse campo de texto (event.target.value)
* A função handleCreateNewContent é executada quando o usuário envia o Formulário
* Ela tira o evento default de redirecionar a página
* Usa a função setComents do useState pegando o estado atual e executando a função NewComent adicionando um novo comentário em quanto deixa a text box vazia

**Corrigindo warnings:**

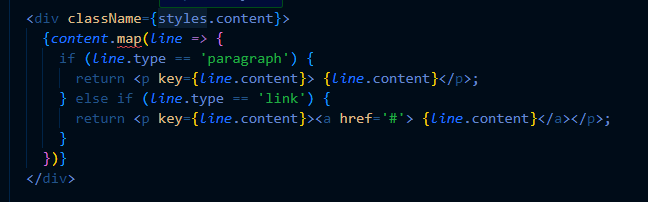
Os warnings:



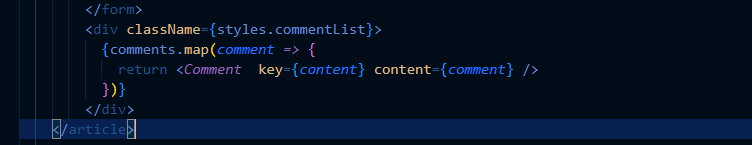
Os erros a cima não impedem o app de funcionar, mas é um aviso de problema de performance, caso aja dúvidas volte no início referente a Keys e renderização do React.

Mas basicamente os erros são por haver listas, e toda vez que o React ter que renderizar ele vai ter que carregar todas as listas mesmo as que não foram alteradas e isso resulta em má performance, para corrigir, basta inserir uma Key única em cada lista presente, dessa forma quando o React for re-renderizar ele vai ignorar as Keys presentes e não re-renderizalas e apenas adicionar a nova ou renderizar a key específica que foi alterada:  
Em App.jsx temos uma lista no componente Post:  


Basta inserir a Key do id.

Em Post.jsx:  


Temos essa lista, como não temos banco de dados para inserir o ID, apenas inserimos o conteúdo do Post.

Também temos outra lista em baixo:  


Commit: