

## NÍVEL AVANÇADO



### PROJETO 17

```
(CONTEÚDO DISPONÍVEL) {  
TWITTER;  
(end);  
})();
```

#PORTFÓLIOBOOSTPROGRAM

### CONHECIMENTOS REQUIRIDOS:



**FULL-STACK**

# WIREFRAME

Tela de login utilizando os componentes do Clerk

usuário

senha

Login

Tela Inicial após login

o que está na sua mente?

Postar

@twittername · 3 horas atrás  
mensagem 5 com limite de chars

@twittername · 13 horas atrás  
mensagem 4 com limite de chars

@twittername · 1 dia atrás  
mensagem 3 com limite de chars

@twittername · 2 dias atrás  
mensagem 2 com limite de chars

@twittername · 3 dias atrás  
mensagem 1 com limite de chars

# TWITTER

Clone de uma das mais famosas redes sociais do mundo

## TECH STACK

- React
- TailwindCSS
- nextJS
- Vercel (conhecimento básico de serverless)
- tRPC
- Prisma
- TypeScript



## LIBRARIES

- ClerkJS
- ZOD Validator
- react-hot-toast
- dayJS



## BRIEFING

Nesse projeto iremos criar um **clone do twitter**, mas diferente dos outros projetos, iremos usar um **stack de tecnologia ULTRA recente**. Projeto recomendado para quem já tem conhecimento sobre o **tech stack** e gostaria de aprender **tRPC**.



## NÍVEL 1

No primeiro nível iremos fazer somente a **estrutura básica** de autenticação com a **ferramenta clerk**, uma ferramenta que **terceiriza** toda a responsabilidade de autenticação deixando o projeto um pouco mais simples.

## NÍVEL 2

Nesse nível iremos fazer um **clone do twitter** com todas as suas funcionalidades, desde criação de novo post até listagem de todos os post em um formato de **timeline**.

## NÍVEL 3

Quer correr a **milha extra**? Faça uma **integração de webhook** para receber uma **notificação** a cada vez um tweet novo for feito!

## REQUISITOS DETALHADOS



### Porque TypeScript nesse projeto?

Esse projeto como disse previamente é mais **avançado**, diferente dos outros que eu recomendei **javascript** esse vamos subir de nível e fazer do zero com **typescript**.

Vale lembrar que todos os outros projetos podem ser feitos em **typescript**, eles foram definidos como **default** em **javascript** mas nada impede ele de ser feito em **TS**.





### O que é Clerk?

- ➔ Clerk é uma ferramenta que te entrega uma **infra-estrutura** completa para **user management** da sua UI e API. Utilizando clerk você não precisa se preocupar com toda a complexidade de **autenticação**.

### Qual o downside de clerk?

- ➔ Como você **terceiriza** toda essa **infra**, os usuários e senhas ficam salvos (de forma criptografada) no servidor deles e não sob sua custódia. Entretanto a plataforma é **super simples** de manusear. Veja em:

**CLERK**



- ➔ Recomendo fazer o **tutorial** deles antes de iniciar o projeto. É simples e rápido e vai te dar as **skills** necessárias para tocar o projeto.

### É só mais um clone do twitter?

- ➔ Definitivamente não, pois o mais importante nesse caso não é o projeto em si, **mas como ele foi feito**.

### Create-t3-app

- ➔ Você já ouviu falar do **create-react-app**?

**CREATE-REACT-APP**



- ➡ Bom, se nunca ouviu falar, saiba que é um excelente **boilerplate** para iniciar projetos em **react** rapidamente com todas as bibliotecas necessárias. Em 2022 foi lançado o **create-t3-app** que é uma cópia do **create-react-app**, mas com **Next.js** e **TypeScript**. **Tailwind CSS**, **tRPC**, **Prisma** e **NextAuth.js** vem incluídos no **create-t3-app** também.

### O que é tRPC?

- ➡ **tRPC** é um **framework de RPC (Remote Procedure Call)** que permite que aplicativos da web se comuniquem com outros aplicativos da web de forma rápida e eficiente.
- ➡ **tRPC** usa o protocolo **HTTP** como base e permite que os aplicativos da web chamem métodos **remotos** em outros aplicativos da web usando chamadas **HTTP**.
- ➡ **tRPC** é uma ótima maneira de **conectar** aplicativos da web e pode ser usado para criar aplicativos da web mais complexos e poderosos. Em 2023 o mercado vê o **tRPC** como o futuro do **GraphQL**.
- ➡ **Logo se você tiver isso no seu currículo será um diferencial tremendo, agora imagina ter isso COM um projeto prático debaixo da manga?**
- ➡ **Logo se você tiver isso no seu currículo será um diferencial tremendo, agora imagina ter isso COM um projeto prático debaixo da manga?**
- ➡ Inicie seu projeto no **GitHub**, use o **create-t3-app (npx create-t3-app@latest)**
- ➡ Suba ele para o **Vercel** (com esse **approach** mantemos a segurança **end-to-end** de que o a aplicação não sofrerá problemas para ser **deployada**)

**Teste** a aplicação em produção antes de começar seu desenvolvimento, nem que seja um **"hello world"** em produção.

➡ Configure seu **DB**, seja utilizando **railway.app**:

**RAILWAY.APP**

➡ Para **postgresSQL** ou **planetScale**:

**PLANETSCALE**

➡ Para **MySQL**. Atualize o **DATABASE\_URL** dentro do **.env**. Dessa forma conseguimos manter toda nossa aplicação **serverless**.

➡ Como iremos utilizar uma **nova tecnologia** que provavelmente você ainda não tem a **skill** darei mais detalhes sobre como você pode desenvolver esse app utilizando **TRPC**.

➡ Ao criar o app com o **create-t3-app** e selecionando o **tRPC** para ser instalado na mesma dentro da **folder /server/api/routers** existe um arquivo chamado **example**. Nele tem exemplos de como as rotas funcionam no **tRPC**.

➡ Vamos fazer um **tutorial** antes para “aquecer os motores” e entender de fato **tRPC**. Lembre-se de utilizar as técnicas de estudo que eu ensino para assimilar melhor o conteúdo.

**TUTORIAL**

➡ (Legendas em português)

- ➔ Utilize **Clerk** para te auxiliar na **autenticação** e nos **conditionals renderings** de página caso o usuário esteja autenticado ou não. Veja a documentação em:

## CLERK



- ➔ Dentro das procedures do **trpcRouter** para buscar os posts você pode utilizar o **ctx.prisma.post.findMany**.
- ➔ Para buscar todos os usuários dentro do **clerk** existe uma **builtin function** chamada **getUserList**, procure mais informações sobre ela na documentação pois você irá precisar disso para descobrir o author de cada post
- ➔ Para converter os **timestamps** para tempos em formato legíveis recomendo o **dayJs**.

## DAY.JS



- ➔ Não se esqueça de inserir **loading spinners** enquanto estiver fazendo **fetch** de todos os posts.
- ➔ Existem diversas maneiras de assegurarmos a segurança desses **requests** e garantir que o usuário esteja autenticado para ver e enviar posts. Existem **2 maneiras** bacanas de como isso pode ser feito.
- ➔ A primeira envolve usar o próprio **middleware nativo** do **clerk** para **NEXTJS**. Veja aqui:

## MIDDLEWARE



- ➡ A segunda, mais **complicada**, é utilizando o **trpc**, criando **privateProcedures**. Veja em:

## TRCP



- ➡ Para **validar** os campos iremos utilizar **ZOD**.

## ZOD



- ➡ Para **error-handling** recomendo o:

## REACT-HOT-TOAST



- ➡ Que é um simples **toaster** que irá aparecer na tela do usuário caso ele receba algum erro.
- ➡ Devemos criar uma **rota** para o perfil também, no **frontend** essa será uma tela simples na qual irá fazer **fetch** de todos os posts do user e sua foto de perfil.
- ➡ Seu projeto terá 3 páginas, a **página inicial** que será servida pelo **trpcRouter** posts e a sua **página de perfil**, que terá informações sobre você incluindo sua foto e a **página de login**.
- ➡ Caso queira percorrer um km extra crie uma página (rota) também para cada post.

