

[tryber / sd-010-a-stranger-things](#)

Public

generated from betrybe/sd-0x-stranger-things

☆ 0 stars ⚡ 1 fork

[Star](#)[Watch](#)[Code](#)[Issues](#)[Pull requests](#)[Actions](#)[Projects](#)[Wiki](#)[Security](#)[Insights](#)[master](#)

...



vinicius-vasconcelos ...

on 28 Sep

[View code](#)

README.md

Termos e acordos

Ao iniciar este projeto, você concorda com as diretrizes do Código de Ética e Conduta e do Manual da Pessoa Estudante da Trybe.

Boas vindas ao repositório do projeto Stranger Things!

Você já usa o GitHub diariamente para desenvolver os exercícios, certo? Agora, para desenvolver os projetos, você deverá seguir as instruções a seguir. Fique atento a cada passo, e se tiver qualquer dúvida, nos envie por *Slack!* #vqv

Aqui você vai encontrar os detalhes de como estruturar o desenvolvimento do seu projeto a partir deste repositório, utilizando uma branch específica e um *Pull Request* para colocar seus códigos.

Sumário

- [Habilidades](#)
- [Entregáveis](#)
 - [O que deverá ser desenvolvido](#)
 - [Desenvolvimento](#)

- Data de entrega
- Instruções para entregar seu projeto:
 - Antes de começar a desenvolver:
 - Durante o desenvolvimento
 - Depois de terminar o desenvolvimento
- Como desenvolver
 - Linter
 - Testes
 - Deploy - Stranger Things
 - Backend
 - API
 - Resposta
 - Filtros
 - Modo upside down - Backend
 - Frontend
 - Comunicação com o backend
 - Modo upside down - Frontend
 - Monitoramento
 - Desenvolvimento
 - Dicas
- Requisitos do projeto
 - Backend
 - 1 - Variáveis de ambiente
 - 2 - Módulo PM2
 - 3 - Ecosystem
 - 4 - Scripts package.json
 - 5 - Procfile
 - 6 - Deploy no Heroku
 - Frontend
 - 7 - Variáveis de Ambiente
 - 8 - Deploy do frontend no Heroku
 - Bônus
 - 9 - Multi-ambientes e Development Mode.
- Avisos Finais

Habilidades

Nesse projeto, você será capaz de:

- Publicar aplicações através do Heroku ;
- Visualizar logs das suas aplicações publicadas;
- Monitorar suas aplicações publicadas;
- Utilizar variáveis de ambiente dentro do Heroku ;
- Instalar, utilizar e aproveitar os principais recursos do PM2 ;
- Fazer deploy no Heroku aproveitando recursos de um process manager.

Entregáveis

Para entregar o seu projeto você deverá criar um Pull Request em **cada um destes** repositórios:

- [Repositório com o backend](#).
- [Repositório com o frontend](#);

Lembre-se que você pode consultar nosso conteúdo sobre [Git & GitHub](#) sempre que precisar!

O que deverá ser desenvolvido

Você vai adaptar e configurar os projetos descritos nesse README para que seja feito o deploy deles. Você vai colocar os projetos frontend e backend no ar com o Heroku , utilizando o PM2 para gerenciar e monitorar os processos. Além disso, você vai alterar alguns comportamentos aplicando os conceitos vistos no conteúdo.

Desenvolvimento

Adapte e configure os projetos descritos nesse *README* para que seja feito o deploy por meio do Heroku e monitorado pelo gerenciador do PM2 .

Data de entrega

- Serão 2 dias de projeto.
- Data de entrega para avaliação final do projeto: 11/10/2021 - 14:00h .

Instruções para entregar seu projeto:

Antes de começar a desenvolver:

1. Clone os **dois** repositórios

- `git clone https://github.com/tryber/sd-010-a-stranger-things-backend.git .`
- `git clone https://github.com/tryber/sd-010-a-stranger-things-frontend.git .`

2. Navegue entre as pastas dos repositórios que você acabou de clonar

- `cd sd-010-a-stranger-things-backend`
- `cd sd-010-a-stranger-things-frontend`

3. Instale as dependências dos dois projetos

- `npm install`

3. Para rodar localmente os projetos, execute o script de start do `package.json`.

- `npm start`

4. Crie uma branch a partir da branch `master` para cada um dos repositórios.

- Verifique que você está na branch `master`
 - Exemplo: `git branch`
- Se não estiver, mude para a branch `master`
 - Exemplo: `git checkout master`
- Agora crie uma branch à qual você vai submeter os `commits` dos seus projetos
 - Você deve criar uma branch no seguinte formato: `nome-de-usuario-nome-do-projeto`
 - Exemplo:
 - `git checkout -b joaozinho-sd-010-a-stranger-things-backend`
 - `git checkout -b joaozinho-sd-010-a-stranger-things-frontend`

5. Adicione a sua branch com o novo `commit` ao repositório remoto

- Usando o exemplo anterior:
 - `git push -u origin joaozinho-sd-010-a-stranger-things-backend`
 - `git push -u origin joaozinho-sd-010-a-stranger-things-frontend`

6. Crie um novo Pull Request (*PR*)

- Vá até a página de *Pull Requests* dos repositórios no GitHub: [Backend](#) | [Frontend](#)
- Clique no botão verde “*New pull request*”
- Clique na caixa de seleção “*Compare*” e escolha a sua branch **com atenção**
- Clique no botão verde “*Create pull request*”

- Adicione uma descrição para o *Pull Request* e clique no botão verde "Create pull request"
- **Não se preocupe em preencher mais nada por enquanto!**
- Volte até a [página de Pull Requests do repositório](#) e confira que o seu *Pull Request* está criado
- Volte até a [página de Pull Requests do repositório](#) e confira que o seu *Pull Request* está criado
- **⚠️ Observação: Os PRs não devem ser abertos neste repositório, apenas nos outros dois repositórios. ⚠️**

Durante o desenvolvimento

- **⚠️ PULL REQUESTS COM ISSUES NO LINTER NÃO SERÃO AVALIADAS, ATENTE-SE PARA RESOLVÊ-LAS ANTES DE FINALIZAR O DESENVOLVIMENTO!**
⚠️
- Faça commits das alterações que você fizer no código regularmente
- Lembre-se de sempre após um (ou alguns) commits atualizar o repositório remoto
- Os comandos que você utilizará com mais frequência são:
 - i. git status (*para verificar o que está em vermelho - fora do stage - e o que está em verde - no stage*)
 - ii. git add (*para adicionar arquivos ao stage do Git*)
 - iii. git commit (*para criar um commit com os arquivos que estão no stage do Git*)
 - iv. git push -u nome-da-branch (*para enviar o commit para o repositório remoto na primeira vez que fizer o push de uma nova branch*)
 - v. git push (*para enviar o commit para o repositório remoto após o passo anterior*)

Depois de terminar o desenvolvimento

Para "entregar" seu projeto, siga os passos a seguir:

- Vá até a página **DO SEU Pull Request**, adicione a label de "code-review" e marque seus colegas
 - No menu à direita, clique no link "**Labels**" e escolha a label **code-review**
 - No menu à direita, clique no link "**Assignees**" e escolha **o seu usuário**

- No menu à direita, clique no link "**Reviewers**" e digite students , selecione o time tryber/students-sd-010-a

Se ainda houver alguma dúvida sobre como entregar seu projeto, [aqui tem um vídeo explicativo](#).

⚠️ Lembre-se que garantir que todas as *issues* comentadas pelo Linter estão resolvidas! ⚠️

Revisando um pull request

À medida que você e as outras pessoas que estudam na Trybe forem entregando os projetos, vocês receberão um alerta via Slack para também fazer a revisão dos Pull Requests dos seus colegas. Fiquem atentos às mensagens do "Pull Reminders" no Slack!

Use o material que você já viu sobre [Code Review](#) para te ajudar a revisar os projetos que chegaram para você.

Como desenvolver

Linter

Para garantir a qualidade do código, vamos utilizar neste projeto o linter ESLint. Assim o código estará alinhado com as boas práticas de desenvolvimento, sendo mais legível e de fácil manutenção! Para rodá-los localmente no projeto, execute o comando abaixo:

- `npm run lint`

⚠️ PULL REQUESTS COM ISSUES DE LINTER NÃO SERÃO AVALIADAS. ATENTE-SE PARA RESOLVÊ-LAS ANTES DE FINALIZAR O DESENVOLVIMENTO! ⚠️

Testes

Backend

Todos os requisitos do projeto serão testados **automaticamente** por meio do Jest . Basta executar o comando abaixo:

```
npm test
```

Frontend

Todos os requisitos do projeto serão testados **automaticamente** por meio do Cypress . Basta executar um dos comandos abaixo:

```
npm run cypress:open // (Com interface)  
npm run cy // (via CLI)
```

Deploy - Stranger Things

Esse repositório contém as instruções e os requisitos para o projeto de Deploy com Heroku e PM2. O código base para o desenvolvimento do projeto está dividido em duas partes: uma API de backend utilizando Node.js e Express e um frontend com React. Abaixo estão disponíveis os links de acesso aos respectivos repositórios:

- [Repositório com o frontend](#);
- [Repositório com o backend](#).

A seguir, temos algumas explicações sobre a estrutura base e alguns comportamentos dessas aplicações. Você explorará esses pontos durante o projeto, alterando o código preexistente.

Lembre-se de que tratam-se de projetos base. Sendo assim, ao longo do desenvolvimento, você vai identificar pontos a serem alterados com o objetivo de aplicar as práticas que vimos durante o curso. Mas não se preocupe, pois no README deste repositório esses pontos estão especificados.

Backend

O Backend possui a seguinte estrutura:

```
├── README.md  
├── index.js  
├── data  
│   ├── dataset  
│   │   └── stranger-things-characters.json  
│   └── repository  
│       └── StrangerThings.js  
└── services  
    └── StrangerThings.js  
├── package-lock.json  
└── package.json
```

API

A API consiste em um serviço simples com um endpoint / que retorna uma listagem com os personagens da série **Stranger Things**.

Resposta

A lista de personagem possui os seguintes campos:

- **name**: Nome do personagem;
- **status**: se o personagem está vivo ou não na série. Os valores possíveis são alive , deceased ou unknown ;
- **origin**: o local de origem do personagem.

A resposta é em formato JSON , e o retorno é sempre um array de objetos. Abaixo, um exemplo:

```
[  
  {  
    "name": "Will",  
    "status": "Alive",  
    "origin": "Byers Family"  
  }  
]
```

Filtros

Todos os campos da estrutura de retorno da resposta podem ser utilizados como filtros. Para isso, basta passar na query string o filtro desejado. No exemplo abaixo, estamos filtrando pelo nome do personagem:

```
localhost:3000?name=el
```

Os filtros são feitos utilizando uma regex , buscando pelo texto passado na query string em qualquer parte do campo, sem diferenciar maiúsculas de minúsculas.

Nesse caso o retorno seria:

```
[  
  {  
    "name": "Russell",  
  }
```

```

    "status": "Alive",
    "origin": "Hawkings Middle School"
},
{
  "name": "Eleven",
  "status": "Alive",
  "origin": "Hopper Family"
}
]

```

Modo upside down - Backend

Na API, no arquivo `index.js`, há a seguinte variável de controle:

```
const hereIsTheUpsideDown = true;
```

Caso ela seja `true`, a API ativará o modo "Mundo Invertido", conforme a série, e começará a responder desta forma:

```

[
  {
    "name": "lləssRnus",
    "origin": "Hawkiñgs Middlē Schooł",
    "status": "Alivé"
  },
  {
    "name": "Eleven",
    "origin": "Hopper Famíly",
    "status": "Alivé"
  }
]

```

Frontend

O frontend consiste em um projeto simples utilizando React, que renderizará o seguinte layout:



Trata-se de um frontend bem simples que vai se comunicar com a nossa API. Ele possui as seguintes funcionalidades:

- Uma tabela para exibição da resposta da nossa API;
- Um campo de pesquisa para filtro de **nome**;
- Botões de navegação para paginação;
- Um botão para ativar o modo `Mundo Invertido` no frontend.

Todas essas funcionalidades estão implementadas no componente `StrangerThings`.

Comunicação com o backend

Na estrutura do projeto, temos um diretório `services`. Nesse diretório, há um arquivo `charactersAPI.js`, onde são feitas as chamadas `HTTP` para a nossa API, utilizando o `axios`.

O service é uma classe que espera receber a URL da nossa API e um timeout para a requisição:

```
{
  url: 'localhost:3003',
  timeout: 30000
}
```

Esses parâmetros estão pré-programados de maneira fixa no componente (alteraremos esse comportamento durante o projeto):

```
const strangerThingsConfig = {  
  url: 'localhost:3002',  
  timeout: 30000,  
};  
  
const upsideDownConfig = {  
  url: 'localhost:3003',  
  timeout: 30000,  
};  
  
const charactersService = new CharactersService(strangerThingsConfig);  
const charactersUpsideDownService = new CharactersService(upsideDownConfig);
```

Modo upside down - Frontend

Assim como no backend, o frontend também possui um modo "Mundo Invertido". Esse modo é ativado e desativado com o botão "Mudar de Realidade".

A ideia é a seguinte: inicialmente, o frontend possui uma imagem de background e utiliza o service instanciado com a configuração contida na variável `strangerThingsConfig`. Ao ativar o Mundo Invertido, a imagem de background é alterada e passamos a utilizar o serviço instanciado com a configuração `upsideDownConfig`.

Desse modo, ao "alternar entre os universos", vamos realizar chamadas a API's diferentes.

No exemplo pré-programado, em um dos "universos", chamamos um serviço na porta 3002 e o outro serviço na porta 3003. Exploraremos esse comportamento durante o projeto.

Monitoramento

Para monitorar sua aplicação no heroku usando o dashboard do PM2, siga os passos abaixo:

1.Crie um novo bucket no dashboard de monitoramento web do PM2 . Em seguida, pelo dashboard, adicione as chaves criadas aos apps do Heroku criados anteriormente.

2.Deverão ser adicionadas três variáveis para cada app:

- O nome do server. No caso, utilizaremos um nome diferente para cada um dos apps;
- Chave pública;
- Chave privada.

3.Verifique no Dashboard se os processos estão sendo exibidos e monitorados.

Desenvolvimento

O código não está utilizando variáveis de ambiente. Você vai configurá-lo para utilizá-las, tornando parametrizáveis a porta e o *modo upside down*.

Em seguida, você deverá adicionar o modulo `PM2` ao projeto utilizando o arquivo `ecosystem`, fazendos as adaptações necessárias no `package.json`.

Você vai realizar o deploy do projeto (frontend e backend) no *Heroku*. Para isso, você deverá prepará-los, adicionando os respectivos `Procfiles`. Eles deverão apontar para os scripts adicionados ao `package.json` que utilizam o `PM2`. Por último, você deverá realizar o deploy do projeto, conforme requisitos, no *Heroku*. Os comandos utilizados deverão ser adicionados no `README.md` de cada projeto. Para mais informações sobre - [Profiles](#) e [ecosystem](#) visite o Course.

Todos esses passos estão detalhados nos requisitos abaixo.

Dicas

Para publicar seu frontend React, utilize o buildpack [mars/create-react-app](#).

Lembre-se de que é possível testar o comportamento definindo as variáveis de ambiente em sua máquina. Você pode fazê-las apontar tanto para o backend rodando localmente em sua máquina, quanto para as APIs já publicadas nos requisitos anteriores.

 **Dica:** Lembre-se que ao importar uma variável de ambiente com `process.env.nomeDaVariavel`, seu tipo é `string`. 

Requisitos do projeto

 Lembre-se que o seu projeto só será avaliado se estiver passando por **todos os checks** do **Linter**. Utilize o comando `npm run lint` no seu terminal para verificar os `checks` do **Linter** 😊 

Os requisitos estão agrupados por `frontend` e `backend`. Realize as alterações nos respectivos diretórios [disponibilizados para você](#).

Backend

1 - Verifica as variáveis de ambiente

Altere o backend para utilizar variáveis de ambiente para contrololar os seguintes comportamentos:

1. A porta que a API escutará, essa variável deve ter, nescessariamente, o nome PORT e o seu valor deve ser 3000.
2. O modo "upsideDown". Essa variável espera um valor booleano e deverá se chamar UPSIDE DOWN_MODE. Lembre-se que as variáveis de ambiente são strings .

O que será testado:

- Se existe a variável de ambiente PORT.
- Se a variável de ambiente UPSIDE DOWN_MODE existe e se ela é um booleano.

Importante: Para esse projeto, as variáveis de ambiente devem ser definidas em um arquivo .env e o arquivo deve ser enviando no seu PR(Pull Request). ISSO NÃO É UMA PRÁTICA DE MERCADO, o arquivo .env deve ser sempre incluido do .gitignore pois contém informações sensíveis, aqui será enviado apenas por motivo de avaliação.

2 - Verifica se o módulo pm2 foi instalado no projeto

Adicione o módulo PM2 à API.

O que será testado:

- Se o módulo pm2 esta instalado nas dependências.

3 - Verifica a configuração do ecosystem.config.yml

Adicione o [arquivo](#) ecosystem.config.yml . O arquivo deverá realizar as seguintes configurações:

1. Ativar o Modo Cluster;
2. Subir duas instâncias do processo;
3. Não assistir à alterações no diretório (modo watch desativado);
4. Reiniciar o processo caso ele consuma mais de 200MB de memória.

importante: O arquivo ecosystem deve ter a extensão yml e não yaml.

O que será testado:

- Se o ecosystem tem a propriedade name
- Se o script a ser executado é o index.js.
- Se o modo de execução está configurado para cluster.

- Se o numero de instancias está definido como 2.
- Se o modo watch esta configurado para estar desativado.
- Se a reiniciação de memória máxima esta configurada como 200M. [Documentação do pm2](#)

4 - Verifica se os scripts do package.json estão corretos

Adicione/altere dois scripts no package.json :

1. start : Deverá iniciar o server utilizando o módulo do PM2 e apontando para o arquivo ecosystem criado;
2. start:dev : Deverá iniciar o server utilizando o módulo do PM2 , **sem** apontar para o arquivo ecosystem e com o parâmetro para "observar alterações no diretório" **ativado**.

Execute ambos em sua máquina para validar se o comportamento é o esperado.

O que será testado:

- Se o comando start inicia o server com pm2 e se usa o ecosystem.
- Se o comando start:dev inicia o server com pm2, se não usa o ecosystem e abre em watchMode.

5 - Verifica a configuração do arquivo Procfile

Defina um [arquivo Procfile](#) , utilizando a mesma configuração do script start do package.json : iniciar o server utilizando o módulo do PM2 , apontando para o arquivo ecosystem criado anteriormente.

Lembre-se: como nossos serviços receberão acessos HTTP externos, precisamos definir os Dynos como sendo do tipo web .

O que será testado:

- Se o dyno é do tipo web.
- Se o script inicia o server com pm2 e se usa o ecosystem.

6 - Verifica o Deploy no Heroku

IMPORTANTE: Uma variável de ambiente com o nome GITHUB_USER deverá ser criada com o seu usuário do github.

IMPORTANTE: O heroku limita o tamanho do nome de uma aplicação para ter no máximo **30 caracteres**, se o seu usuário do GitHub possuir mais que 27 caracteres você não conseguirá criar uma aplicação com o nome no padrão solicitado pelo requisito. Caso esteja nessa condição, avise nosso time de instrução que iremos ajudá-lo.

1. Crie dois apps do Heroku a partir do mesmo código fonte (código da API). O nome do seu app no heroku deverá conter seu nome de usuário no github seguido de "-bk" ou "-bd". Por exemplo, se seu nome de usuário no github for "student" seus app deverão ter o nome "student-bk" e "student-bd" e as urls abaixo:

- <https://student-bk.herokuapp.com/> -para o app hawkins
- <https://student-bd.herokuapp.com/> -para o app upside-down

2. Configure a variável de ambiente criada para controlar o modo `upsideDown`. Cada um dos apps deverá ter valores distintos, da seguinte maneira:

- O app hawkins deverá ter o valor `false` ;
- O app upside-down deverá ter o valor `true` .

3. Utilizando o `git`, faça o deploy de ambos os apps no Heroku.

Acesse os URLs geradas e valide se ambas as API's estão no ar e funcionando como esperado. **Importante:** As URLs deverão ser geradas pelo heroku e não devem ser modificadas.

O que será testado:

- Se ao fazer uma requisição do tipo GET para o endpoint da api Hawkins serão retornadas as informações corretas.
- Se ao fazer uma requisição do tipo GET para o endpoint da api upsideDown serão retornadas as informações corretas.

Frontend

7 - Verifica as variáveis de ambiente do frontend

Altere o frontend para utilizar variáveis de ambiente para controlar as **URLs** e **Timeouts** de comunicação com a API.

Perceba que o código está esperando por duas **APIs**. Uma para o modo `upside-down` e a outra para o modo normal.

O nome das variáveis deve ser o seguinte:

- Para seu back-end hawkins:
 - `REACT_APP_HAWKINS_URL` para a URL;
 - `REACT_APP_HAWKINS_TIMEOUT` para o timeout;
- Para seu back-end UPSIDEDOWN:

- REACT_APP_UPSIDEDOWN_URL para a URL;
- REACT_APP_UPSIDEDOWN_TIMEOUT para o timeout;

O que será testado:

- Se existem as 4 variáveis de ambiente citadas acima.

Importante: Para esse projeto, as variáveis de ambiente devem ser definidas em um arquivo .env e o arquivo deve ser enviado no seu PR(Pull Request). ISSO NÃO É UMA PRÁTICA DE MERCADO, o arquivo .env deve ser sempre incluído do .gitignore pois contém informações sensíveis, aqui será enviado apenas por motivo de avaliação.

8 - Verifica se foi feito o deploy do frontend no Heroku

IMPORTANTE: Assim como no backend, a variável de ambiente GITHUB_USER

deverá ser criada com o seu usuário do github.

Faça o deploy do front-end:

1. Crie um app do Heroku com o front-end. Não é necessário a criação do Procfile aqui. Vamos deixar o Heroku utilizar as configurações padrões. No momento de criar o app do Heroku, utilize o buildpack descrito abaixo, em **Dicas**.
2. O nome do seu app no heroku deve ser seu nome de usuário do github seguido de "-ft". Por exemplo, se o seu usuário do github for "student", o nome do seu app será "student-ft" e a url **precisar ser <https://student-ft.herokuapp.com/>**.
3. Configure as variáveis de ambiente do app para apontar para as API's publicadas.
4. Faça o deploy com o git.

Dicas:

Para publicar seu frontend React, utilize o buildpack [mars/create-react-app](#).

Lembre-se de que é possível testar o comportamento definindo as variáveis de ambiente em sua máquina. Você pode fazê-las apontar tanto para o backend rodando localmente em sua máquina, quanto para as APIs já publicadas nos requisitos anteriores.

O que será testado:

- Se ao visitar sua página no heroku, o botão de mudar de realidade existe.
- Se a pesquisa funciona como deveria, fazendo chamada a API externa.
- Se o botão de mudar de realidade funciona.
- Se os botões de próxima página e página anterior funcionam.

Bônus

9 - Verifica os multi-ambientes e modo de desenvolvimento.

Utilize a estratégia de multi-ambientes no frontend. Para isso:

- Renomeie o *remote* atual para `development` ;
- Refaça o deploy com um item no frontend que identifique o layout como rodando em modo de "desenvolvimento". Esse tag item **deve** conter o texto "Em desenvolvimento"
- Crie um novo app no heroku cujo nome deve ser seu nome de usuário do github seguido de "-pd". Por exemplo, se o seu usuário do github for "student", o nome do seu app será "student-pd" e a url **precisar ser** <https://student-pd.herokuapp.com/>.
- Lembre-se que a boa prática para essa situação é criar uma variável de ambiente para controlar esse comportamento, configurando-a para ter um valor diferente em cada um dos ambientes.

O que será testado:

- Se ao acessar o frontend de desenvolvimento, haverá a tag com o texto "Em desenvolvimento"
- Se ao acessar o frontend de produção, não haverá a tag.

Depois de terminar o desenvolvimento

Para sinalizar que o seu projeto está pronto para o "Code Review" de colegas, faça o seguinte:

- Vá até a página **DO SEU Pull Request**, adicione a label de "code-review" e marque as pessoas de quem quer receber o review:
 - No menu à direita, clique no link "**Labels**" e escolha a label **code-review**;
 - No menu à direita, clique no link "**Assignees**" e escolha **o seu usuário**;
 - No menu à direita, clique no link "**Reviewers**" e digite `students` , selecione o time `tryber/students-sd-010-a` .

Caso tenha alguma dúvida, [aqui tem um vídeo explicativo](#).

Revisando um pull request

Use o conteúdo sobre [Code Review](#) para te ajudar a revisar os *Pull Requests*.

#VQV 

Avisos Finais

Ao finalizar e submeter o projeto, não se esqueça de avaliar sua experiência preenchendo o formulário. Leva menos de 3 minutos!

Link: [FORMULÁRIO DE AVALIAÇÃO DE PROJETO](#)

O avaliador automático não necessariamente avalia seu projeto na ordem em que os requisitos aparecem no readme. Isso acontece para deixar o processo de avaliação mais rápido. Então, não se assuste se isso acontecer, ok?

Você sabia que o LinkedIn é a principal rede social profissional e compartilhar o seu aprendizado lá é muito importante para quem deseja construir uma carreira de sucesso? Compartilhe esse projeto no seu LinkedIn, marque o perfil da Trybe (@trybe) e mostre para a sua rede toda a sua evolução.

Releases

No releases published

Packages

No packages published