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 Talker Manager!

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

Neste projeto, verificamos se você é capaz de:

- Realizar operações assíncronas utilizando callbacks;
- Realizar operações assíncronas utilizando Promises;
- Ler e escrever arquivos localmente com NodeJS;
- Escrever seus próprios scripts que criam e consomem Promises;
- Reescrever código que usa callbacks para que use Promises;
- Realizar chamadas de funções de forma consciente;
- Entender os conceitos básicos de como o JavaScript funciona;
- Detectar e solucionar problemas no código de forma mais objetiva;
- Entender a diferença entre execução síncrona e assíncrona;
- Entender o que é o HTTP, o que é uma API e o que os dois têm a ver com o Express;
- Escrever APIs utilizando Node e Express;
- Entender a estrutura de uma aplicação Express e como organizar seu código;
- Criar rotas e aplicar middlewares.

Entregáveis

Para entregar o seu projeto você deverá criar um Pull Request neste repositório.

Lembre-se que você pode consultar nosso conteúdo sobre Git & GitHub sempre que precisar!

O que deverá ser desenvolvido

Você vai desenvolver uma API de um CRUD (**C**reate, **R**ead, **U**pdate e **D**elete) de palestrantes. Você vai desenvolver alguns endpoints que irão ler e escrever em um arquivo, isso utilizando o módulo `fs`.

Desenvolvimento

Data de Entrega

O projeto tem até a seguinte data: `08/09/2021 - 14:00h`. Para ser entregue a avaliação final. Serão dois dias de projeto.

Instruções para entregar seu projeto

Antes de começar a desenvolver:

1. Clone o repositório

- `git clone git@github.com:tryber/sd-010-a-project-talker-manager.git`.
- Entre na pasta do repositório que você acabou de clonar:
  - `cd sd-010-a-project-talker-manager`

2. Instale as dependências [**Caso existam**]

- `npm install`

3. Crie uma branch a partir da branch `master`

- 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` do seu projeto
  - Você deve criar uma branch no seguinte formato: `nome-de-usuario-nome-do-projeto`
  - Exemplo: `git checkout -b joaozinho-sd-010-a-project-talker-manager`

4. Adicione as mudanças ao *stage* do Git e faça um `commit`

- Verifique que as mudanças ainda não estão no *stage*
  - Exemplo: `git status` (deve aparecer listada a pasta *joaozinho* em vermelho)
- Adicione o novo arquivo ao *stage* do Git
  - Exemplo:
    - `git add .` (adicionando todas as mudanças - *que estavam em vermelho* - ao stage do Git)
    - `git status` (deve aparecer listado o arquivo *joaozinho/README.md* em verde)
- Faça o `commit` inicial
  - Exemplo:
    - `git commit -m 'iniciando o projeto x'` (fazendo o primeiro commit)
    - `git status` (deve aparecer uma mensagem tipo *nothing to commit*)

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

- Usando o exemplo anterior: `git push -u origin joaozinho-sd-010-a-project-talker-manager`

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

- Vá até a página de *Pull Requests* do repositório no GitHub
- 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

Durante 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 (opcional):

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

- 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`.

Caso tenha alguma dúvida, aqui tem um video explicativo.

Revisando um pull request

Use o conteúdo sobre Code Review para te ajudar a revisar os *Pull Requests*.

Como desenvolver:

Sobre o avaliador

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?

Requisitos do projeto

Linter

Usaremos o ESLint para fazer a análise estática do seu código.

Este projeto já vem com as dependências relacionadas ao *linter* configuradas nos arquivos `package.json`.

Para poder rodar o `ESLint` em um projeto basta executar o comando `npm install` dentro do projeto e depois `npm run lint`. Se a análise do `ESLint` encontrar problemas no seu código, tais problemas serão mostrados no seu terminal. Se não houver problema no seu código, nada será impresso no seu terminal.

Você pode também instalar o plugin do `ESLint` no `VSCode`, bastar ir em extensions e baixar o plugin `ESLint`.

Lista de requisitos

Observações

1. Com exceção do requisito 3, todos os outros requisitos deverão ser feitos utilizando o módulo `fs`.

2. O arquivo `talker.json` será utilizado como base para fazer as requisições da API. As operações de leitura e escrita dos requisitos devem ser feitas nesse arquivo usando os métodos da biblioteca `fs`.

3. Há um arquivo `index.js` no repositório. Não remova, nele, o seguinte trecho de código:

```
app.get('/', (_request, response) => {
  response.status(HTTP_OK_STATUS).send();
});
```

Isso está configurado para o avaliador funcionar.

4. Caso os testes falhem seu arquivo `talker.json` não será restaurado, para isso utilize `npm run restore`.

5. Ao se deparar com o erro de que a porta já está em uso: `EADDRINUSE: address already in use 0.0.0.0:3000`, execute em seu terminal `killall node` isso finaliza todas as execuções do node.

1 - Crie o endpoint GET `/talker`

Os seguintes pontos serão avaliados:

- O endpoint deve retornar um array com todas as pessoas palestrantes cadastradas. Devendo retornar o `status 200`, com o seguinte corpo:

```
[
  {
    "name": "Henrique Albuquerque",
    "age": 62,
    "id": 1,
    "talk": { "watchedAt": "23/10/2020", "rate": 5 }
  },
  {
    "name": "Heloísa Albuquerque",
    "age": 67,
    "id": 2,
    "talk": { "watchedAt": "23/10/2020", "rate": 5 }
  },
  {
    "name": "Ricardo Xavier Filho",
    "age": 33,
    "id": 3,
    "talk": { "watchedAt": "23/10/2020", "rate": 5 }
  },
  {
    "name": "Marcos Costa",
    "age": 24,
    "id": 4,
    "talk": { "watchedAt": "23/10/2020", "rate": 5 }
  }
]
```

- Caso não exista nenhuma pessoa palestrante cadastrada o endpoint deve retornar um array vazio e o `status 200`.

2 - Crie o endpoint GET `/talker/:id`

- O endpoint deve retornar uma pessoa palestrante com base no id da rota. Devendo retornar o `status 200` ao fazer uma requisição `/talker/1`, com o seguinte corpo:

```
{
  "name": "Henrique Albuquerque",
  "age": 62,
  "id": 1,
  "talk": { "watchedAt": "23/10/2020", "rate": 5 }
}
```

- Caso não seja encontrada uma pessoa palestrante com base no id da rota, o endpoint deve retornar o `status 404` com o seguinte corpo:

```
{
  "message": "Pessoa palestrante não encontrada"
}
```

3 - Crie o endpoint POST `/login`

Os seguintes pontos serão avaliados:

- O endpoint deve ser capaz de retornar um token aleatório de 16 caracteres que deverá ser utilizado nas demais requisições.
  - O endpoint deverá retornar um código de `status 200` com o token gerado, com o seguinte corpo:

```
{
  "token": "7mqaVRXJSp886CGr"
}
```

- O corpo da requisição deverá ter o seguinte formato:

```
{
  "email": "email@email.com",
  "password": "123456"
}
```

- O campo `email` deverá ser um email válido. Ele é obrigatório.
  - Caso o campo não seja passado ou esteja vazio retorne um código de `status 400` , com o seguinte corpo:

```
{
  "message": "O campo \"email\" é obrigatório"
}
```

  - Caso o email passado não seja um email válido retorne um código de `status 400` , com o seguinte corpo:

```
{
  "message": "O \"email\" deve ter o formato \"email@email.com\""
}
```

- O campo `password` deverá ter pelo menos 6 caracteres.
  - Caso o campo não seja passado ou esteja vazio retorne um código de `status 400` , com o seguinte corpo:

```
{
  "message": "O campo \"password\" é obrigatório"
}
```

- Caso a senha não tenha pelo menos 6 caracteres retorne um código de `status 400` , com o seguinte corpo:

```
{
  "message": "O \"password\" deve ter pelo menos 6 caracteres"
}
```

4 - Crie o endpoint POST `/talker`

Os seguintes pontos serão avaliados:

- O endpoint deve ser capaz de adicionar uma nova pessoa palestrante ao seu arquivo;
- O corpo da requisição deverá ter o seguinte formato:

```
{
  "name": "Danielle Santos",
  "age": 56,
  "talk": {
    "watchedAt": "22/10/2019",
    "rate": 5
  }
}
```

- A requisição deve ter o token de autenticação nos headers.
  - Caso o token não seja encontrado retorne um código de `status 401` , com o seguinte corpo:

```
{
  "message": "Token não encontrado"
}
```

  - Caso o token seja inválido retorne um código de `status 401` , com o seguinte corpo:

```
{
  "message": "Token inválido"
}
```

- O campo `name` deverá ter no mínimo 3 caracteres. Ele é obrigatório.
  - Caso o campo não seja passado ou esteja vazio retorne um código de `status 400` , com o seguinte corpo:

```
{
  "message": "O campo \"name\" é obrigatório"
}
```

  - Caso o nome não tenha pelo menos 3 caracteres retorne um código de `status 400` , com o seguinte corpo:

```
{
  "message": "O \"name\" deve ter pelo menos 3 caracteres"
}
```

- O campo `age` deverá ser um inteiro e apenas pessoas maiores de idade (pelo menos `18 anos`) podem ser cadastrados. Ele é obrigatório.
  - Caso o campo não seja passado ou esteja vazio retorne um código de `status 400` , com o seguinte corpo:

```
{
  "message": "O campo \"age\" é obrigatório"
}
```

  - Caso a pessoa palestrante não tenha pelo menos 18 anos retorne `status 400` , com o seguinte corpo:

```
{
  "message": "A pessoa palestrante deve ser maior de idade"
}
```

- O campo `talk` deverá ser um objeto com as seguintes chaves:
  - A chave `watchedAt` deve ser uma data no formato `dd/mm/aaaa` .
    - Caso a data não respeito o formato `dd/mm/aaaa` retorne `status 400` , com o seguinte corpo:

```
{
  "message": "O campo \"watchedAt\" deve ter o formato \"dd/mm/aaaa\""
}
```

- A chave `rate` deve ser um inteiro de 1 à 5.
  - Caso a nota não seja um inteiro de 1 à 5 retorne `status 400` , com o seguinte corpo:

```
{
  "message": "O campo \"rate\" deve ser um inteiro de 1 à 5"
}
```

- O campo `talk` é obrigatório e nenhuma das chaves citadas anteriormente podem ser vazias.
  - Caso o campo não seja informado, esteja vazio ou então alguma de suas chaves não tenham sido informadas retorne `status 400` , com o seguinte corpo:

```
{
  "message": "O campo \"talk\" é obrigatório e \"watchedAt\" e \"r
}
```

- Caso esteja tudo certo, retorne o `status 201` e a pessoa cadastrada.
- O endpoint deve retornar o `status 201` e a pessoa palestrante que foi cadastrada, da seguinte forma:

```
{
  "id": 1,
  "name": "Danielle Santos",
  "age": 56,
  "talk": {
    "watchedAt": "22/10/2019",
    "rate": 5
  }
}
```

5 - Crie o endpoint PUT `/talker/:id`

Os seguintes pontos serão avaliados:

- O endpoint deve ser capaz de editar uma pessoa palestrante com base no id da rota, sem alterar o id registrado.
- O corpo da requisição deverá ter o seguinte formato:

```
{
  "name": "Danielle Santos",
  "age": 56,
  "talk": {
    "watchedAt": "22/10/2019",
    "rate": 5
  }
}
```

- A requisição deve ter o token de autenticação nos headers.
  - Caso o token não seja encontrado retorne um código de `status 401`, com o seguinte corpo:

    ```
    {
      "message": "Token não encontrado"
    }
    ```

  - Caso o token seja inválido retorne um código de `status 401`, com o seguinte corpo:

    ```
    {
      "message": "Token inválido"
    }
    ```

- O campo `name` deverá ter no mínimo 3 caracteres. Ele é obrigatório.
  - Caso o campo não seja passado ou esteja vazio retorne um código de `status 400`, com o seguinte corpo:

    ```
    {
      "message": "O campo \"name\" é obrigatório"
    }
    ```

  - Caso o nome não tenha pelo menos 3 caracteres retorne um código de `status 400`, com o seguinte corpo:

    ```
    {
      "message": "O \"name\" ter pelo menos 3 caracteres"
    }
    ```

- O campo `age` deverá ser um inteiro e apenas pessoas maiores de idade (pelo menos `18 anos`) podem ser cadastrados. Ele é obrigatório.

- Caso o campo não seja passado ou esteja vazio retorne um código de `status 400`, com o seguinte corpo:

```
{
  "message": "O campo \"age\" é obrigatório"
}
```

- Caso a pessoa palestrante não tenha pelo menos 18 anos retorne `status 400`, com o seguinte corpo:

```
{
  "message": "A pessoa palestrante deve ser maior de idade"
}
```

- O campo `talk` deverá ser um objeto com as seguintes chaves:
  - A chave `watchedAt` deve ser uma data no formato `dd/mm/aaaa`.
    - Caso a data não respeito o formato `dd/mm/aaaa` retorne `status 400`, com o seguinte corpo:

```
{
  "message": "O campo \"watchedAt\" deve ter o formato \"dd/mm/aaaa\""
}
```

  - A chave `rate` deve ser um inteiro de 1 à 5.
    - Caso a nota não seja um inteiro de 1 à 5 retorne `status 400`, com o seguinte corpo:

```
{
  "message": "O campo \"rate\" deve ser um inteiro de 1 à 5"
}
```

  - O campo `talk` é obrigatório e nenhuma das chaves citadas anteriormente podem ser vazias.
    - Caso o campo não seja informado, esteja vazio ou então alguma de suas chaves não tenham sido informadas retorne `status 400`, com o seguinte corpo:

```
{
  "message": "O campo \"talk\" é obrigatório e \"watchedAt\" e \"r
}
```

- Caso esteja tudo certo, retorne o `status 200` e a pessoa editada.
- O endpoint deve retornar o `status 200` e a pessoa palestrante que foi editada, da seguinte forma:

```
{
  "id": 1,
 "name": "Danielle Santos",
  "age": 56,
  "talk": {
    "watchedAt": "22/10/2019",
    "rate": 4
  }
}
```

6 - Crie o endpoint DELETE `/talker/:id`

Os seguintes pontos serão avaliados:

- A requisição deve ter o token de autenticação nos headers.
  - Caso o token não seja encontrado retorne um código de `status 401` , com o seguinte corpo:

```
{
  "message": "Token não encontrado"
}
```

  - Caso o token seja inválido retorne um código de `status 401` , com o seguinte corpo:

```
{
  "message": "Token inválido"
}
```

- O endpoint deve deletar uma pessoa palestrante com base no id da rota. Devendo retornar o `status 200` , com o seguinte corpo:

```
{ "message": "Pessoa palestrante deletada com sucesso" }
```

7 - Crie o endpoint GET `/talker/search?q=searchTerm`

Os seguintes pontos serão avaliados:

- O endpoint deve retornar um array de palestrantes que contenham em seu nome o termo pesquisado no queryParam da URL. Devendo retornar o `status 200`, com o seguinte corpo:

```
/search?q=Da
```

```
[
  {
    id: 1,
    name: "Danielle Santos",
    age: 56,
    talk: {
      watchedAt: "22/10/2019",
      rate: 5,
    },
  }
];
```

- A requisição deve ter o token de autenticação nos headers.
  - Caso o token não seja encontrado retorne um código de `status 401`, com o seguinte corpo:

```
{
  "message": "Token não encontrado"
}
```

  - Caso o token seja inválido retorne um código de `status 401`, com o seguinte corpo:

```
{
  "message": "Token inválido"
}
```

- Caso `searchTerm` não seja informado ou esteja vazio, o endpoint deverá retornar um array com todos as pessoas palestrantes cadastradas, assim como no endpoint GET `/talker`, com um `status 200`.
- Caso nenhuma pessoa palestrante satisfaça a busca, o endpoint deve retornar o `status 200` e um array vazio.

Dica é importante ter atenção se essa rota não entra em conflito com as outras, já que a ordem das rotas faz diferença na interpretação da aplicação

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
Create a new release

Packages

No packages published
Publish your first package

Contributors 3


jeanpsv Jean Paulo Silva Vasconcelos


phramos07 Pedro Ramos


GabrielCoruja Gabriel Dalseco

Languages

- **JavaScript** 100.0%