Especificações do Projeto - dev.boost(2019.2)

Objetivo:

Desenvolver um **site educacional de vídeos** que utilize o paradigma de **SPA** (**S**ingle **P**age **A**pplication). O site deve possuir uma **página de login**, uma **página de cadastro**, uma **página de vídeos** e uma **página de gerenciamento**, que permita adicionar ou remover um novo vídeo ao site.

Abordagem

O projeto é dividido em duas pastas: **Front-End** e **Back-End**.

Front-End

Tem como função servir de plataforma pública que seja capaz de:

1. **Renderizar** o HTML e a **estilização** do mesmo, criando uma interface com a qual o usuário pode interagir e **enviar informações** que descrevam quais partes da aplicação este quer acessar ou modificar.
2. **Enviar** requisições **HTTP** que especifiquem qual método **(POST, GET, DELETE)** será solicitado e encapsular de maneira estruturada a informação relacionada a esta operação ao **BackEnd**.
3. **Escutar as respostas** geradas pelo Backend quando este responde a uma requisição HTTP.
4. **Aplicar uma lógica interna** que, a partir das respostas do backend, decida de qual maneira a interface do site será manipulada.
5. **Re-Renderizar** a aplicação de maneira a **refletir** para o usuário a **mudança do estado da aplicação** feita durante as aplicações de lógica interna, bem como continuar apta a **novas interações** como usuário.

Back-End

Tem como função servir de plataforma lógica comunicação entre o Front-End o banco de dados, que seja capaz de:

1. **Receber** requisições HTTP enviadas pelo Front-End
2. **Aplicar uma lógica interna** que desestruture a informação recebida pelo protocolo HTTP
3. **Verificar permissão**. Normalmente, um **token** é gerado na criação de usuário e passa a ser enviado no cabeçalho das requisições. Algumas operações podem ser feitas por visitantes, já outras exigem um token de usuário, e outras token de administrador (**roles**)
4. **Enviar ao banco** a requisição, agora tratada pela lógica interna
5. **Escutar a resposta** do banco sobre a requisição feita.
6. **Aplicar uma lógica interna** que trate o conteúdo recebido pela requisição ao banco de maneira a gerar a resposta esperada pelo FrontEnd
7. **Responder a requisição http ao frontend**

Banco de dados

Será utilizado o **Cloud Firestorm do Google Firebase**. Não será necessário o conhecimento de programação de bancos de dados, visto que este processo é automatizado pela ferramenta. Em suma, o banco **recebe e responde requisições** ao Back-End.

Assim como o Back-End, o banco de dados possui uma lógica interna para **autenticação**, caso contrário, seria possível transpassar o backend e enviar solicitações diretamente ao banco. Normalmente, é **verificado** se o IP que solicitou a requisição é o **mesmo do IP do Back-End**. Tal método de verificação **já é implementado** pelo FireBase.

Tecnologia base

Node e NPM

Tanto as tecnologias utilizadas front-end quanto as no back-end ancoram-se no **node.js** e seu gerenciador de pacotes, o **npm**. Em suma, Node trata-se de um **interpretador de código JavaScript** de modo **assíncrono e orientado a eventos**, com objetivo de permitir a execução de código Javascript independentemente do Browser, ou seja, nativamente por um Servidor.

Para rodar um código javascript pelo Node, basta:

```
node index.js
```

Segue um **tutorial de instalação no ubuntu** e um **no Windows**. A versão LTS do Node já instala automaticamente o **npm**, **gerenciador** que permite instalação e distribuição facilitadas de pacotes voltados ao ambiente node.

Playlist fortemente recomendada sobre javascript e node:

Programação funcional vs POO Event Loop JS Assíncrono: Callbacks, Promises e Async/Await Arrow Functions

Git

Ferramenta de **versionamento** que permite manter repositórios com snapshots de diversos momentos de um projeto. **Github** é uma plataforma que permite **compartilhamento online** de repositórios git. **recomendamos fortemente** o uso de git, pois o mesmo facilita o **desenvolvimento remoto** de um mesmo projeto por **vários autores diferentes**, bem como manter um **backup** da **versão atual e anteriores** do projeto.

Em suma, possui **quatro** locais/momentos de desenvolvimento:

1 - WorkSpace:

Local onde os arquivos estão sendo modificados. Para fazer de um diretório um WorkSpace de git, basta utilizar

```
git init
```

Pode ser mais fácil, entretanto, criar primeiramente um repositório no github, e então utilizar:

```
git clone url_do_repositorio
```

Pois isto fará o estágio de push mais prático.

2 - Stage

Local onde se pode acumular diversas alterações de um arquivo, sem de fato salvá-las no repositório local. Serve como uma "caixa intermediária" entre o workspace e o repositório local.

Para adicionar um arquivo ao stage, basta utilizar

```
git add nome_do_arquivo
```

E, para adicionar uma pasta inteira,

```
git add .
```

3 - Repositório Local

Local que mantém a versão mais atualizada do projeto, bem como o histórico de snapshots do mesmo com todas as versões que foram commitadas. Para adicionar o Stage ao Repositório local, basta:

```
git commit -m "Mensagem de commit"
```

4 - Repositório na Nuvem - GitHub

Para subir o repositório local para a nuvem:

1- Caso tenha sido utilizado o `git clone` na etapa 1. , basta utilizar:

```
git push
```

2 - Caso tenha sido utilizado o `git init`, é necessário utilizar na primeira vez:

```
git push --set-upstream url_do_repositorio
```

Após isto, `git push` por sí só poderá ser utilizado para commits futuros.

Para boas práticas e informações mais avançadas sobre git, como Fetch/Rebase e Branches, consultar o **tutorial de Git fortemente recomendado por UCLSanca**

Tecnologias do Front-End

React

Deve ser utilizando o React para a criação da SPA, com a abordagem de **Componentes Funcionais (Hooks)**.

A instalação e a criação de um app React é feita com:

```
npx create-react-app my_app
```

Para iniciar a aplicação:

```
cd my_app
npm start
```

TailWind

Sugere-se o uso do **TailWind** como FrameWork de CSS para a estilização da página. O tutorial da instalação completa em um projeto, bem como exemplos de utilização podem ser encontrado na documentação. Para desenvolvimento, por praticidade, sugere-se a instalação via CDN.

Basta incluir:

```
<link href="https://unpkg.com/tailwindcss@^1.0/dist/tailwind.min.css"
rel="stylesheet">
```

Na tag `<head> </head>` do index.html criado automaticamente na pasta `public`do app pelo React.

Axios

Axios é um cliente que tem por objetivo **enviar requests HTTP e escutar suas respostas**. Segue **uma boa vídeo aula** que exemplifica o uso de axios em React.

Sua instalação também depende do npm, e deve ser feita **Dentro de my_app**

```
npm install axios
```

Por fim, deve ser importado no arquivo .js com

```
import axios from "axios";
```

Entrega 1 - FrontEnd não persistivo

Sugere-se começar o desenvolvimento da aplicação pelo **design** das páginas, bem como sua implementação por HTML/Tailwind/CSS. Caso desejado, é possível desenhar todo o design da aplicação com alguma ferramenta de **prototipação**, como o Marvel App ou Adobe XD. Exemplo de páginas mockadas: Protótipo Site Digimon e Protótipo Ganesh. A aplicação **não precisa ser responsiva** (Mas ficaremos muito felizes se for 😃), ficando a cargo do desenvolvedor escolher desenvolvê-la com foco em navegação **desktop** ou **mobile**.

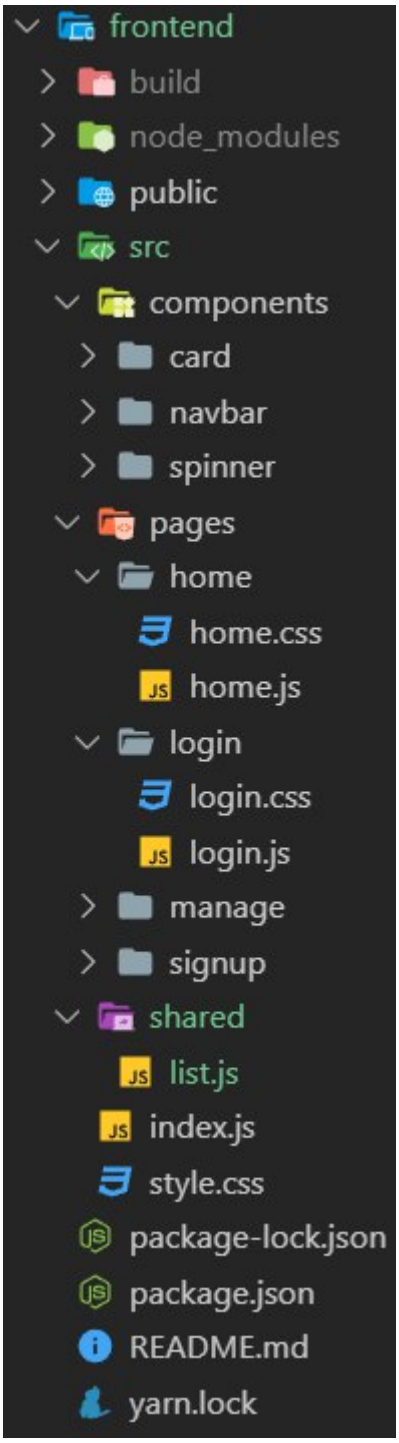
Em seguida, sugere-se desenvolver a lógica da página de **gerenciamento** de vídeos, especificamente as funções de **adicionar** e **remover** vídeos, bem como retornar a lista. Futuramente, a lista de vídeos será requerida no Back-End. Por hora, sugere-se contornar este problema com a criação de um arquivo `lista.js` na pasta pública do projeto. Tal arquivo será importado por `index.js` e deverá possuir, além de uma lista inicialmente, vazias, funcoes **getLista**, **AdicionaItem** e **DeleteItem**. Componentes que se utilizem da lista também deverão importá-lo.

Não é necessário carregar vídeos inteiros na aplicação. A intenção é que se use o `<embed></embed>` de vídeos do youtube. A aplicação apenas salvará links para estes vídeos, mesmo quando o Back-End for implementado. A parte lógica das página de **Login** e **Cadastro** é amarrada ao Back-End e **não é prioridade** nesta etapa de desenvolvimento.

A página gerada **não é persistiva**, isto é, modificações na lista de vídeos se perdem ao se atualizar a página. Entretanto, deverá ser possível adicionar um vídeo pela página de gerenciamento, e acessar a lista de vídeos criados na página de vídeos. Cada página, isto é, **login**, **cadastro**, **vídeos** e **gerenciamento deve possuir seu próprio componente**.

Não é necessário se preocupar com o **Axios** ou com o envio de requisições e recebimento de suas respostas, que serão trabalhadas na entrega 2.

Estrutura sugerida para as pastas do projeto:


frontend
build
node_modules
public
src
components
card
navbar
spinner
pages
home
home.css
home.js
login
login.css
login.js
manage
signup
shared
list.js
index.js
style.css
package-lock.json
package.json
README.md
yarn.lock