

NÍVEL AVANÇADO



PROJETO 18

(CONTEÚDO DISPONÍVEL) {
TRELLO;
USANDO IA;
(end);
})();

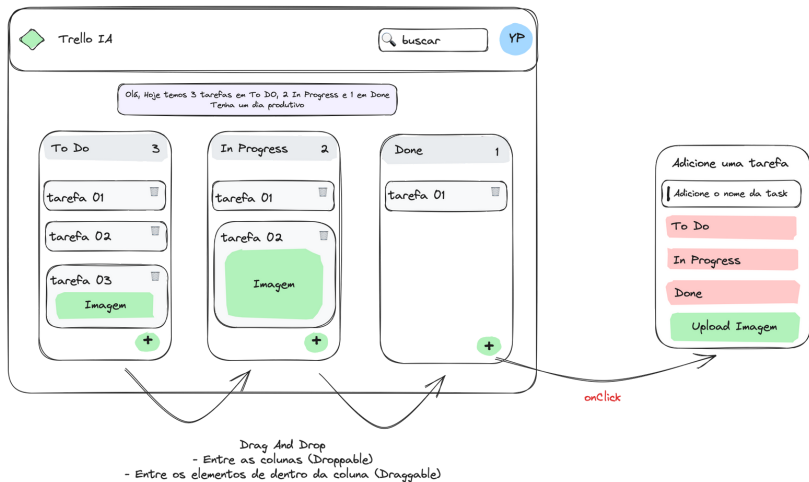
#PORTFÓLIOBOOSTPROGRAM

CONHECIMENTOS REQUIRIDOS:



FULL-STACK

WIREFRAME



Entidades e Types

```

interface BoardState {
  board: Board;
  getBoard: () => void;
  setBoardState: (board: Board) => void;
  updateTodoInDB: (todo: Todo, columnId: TypeColumn) => void;
  newTaskInput: string;
  newTaskType: TypeColumn;
  image: File | null

  searchString: string;
  setSearchString: (searchString: string) => void;

  setNewTaskInput: (input: string) => void;
  setNewTaskType: (columnId: TypeColumn) => void;
  setImage: (image: File | null) => void;
}

```

```

interface Board {
  columns: Map<TypeColumn, Column>
}

type TypeColumn = "todo" | "inprogress" | "done"

```

```

interface Column {
  id: TypeColumn;
  todos: Todo[];
}

```

```

interface Todo {
  $id: string;
  $createdAt: string;
  title: string;
  status: TypeColumn;
  image?: string
}

```

```

interface Image {
  bucketId: string;
  fileId: string;
}

```

OBS: tanto entidades quanto schemas são apenas sugestões para te dar um direcionamento nada do que está escrito aqui deve ser considerado que está escrito em pedra.

Schema

```

model Todo {
  id Int
  title String
  image String
  createdAt DateTime
}

```

TRELLO USANDO IA

Vamos **desenvolver** um dos apps mais famosos para produtividade entretanto com um porém, iremos utilizar **inteligência artificial** para ter um app mais robusto e interessante no seu **portfólio**.

TECH STACK

- NextJs
- Typescript
- TailwindCSS
- Zustand
- APPWRITE (Baas)
- GPT 3.5



LIBRARIES

- HeadlessUI

REACT-BEAUTIFUL-DND



BRIEFING

Imagine você ter em produção o seu próprio **Trello**? Integrando com alguns dos serviços mais em alta no mercado internacional em 2023? Sim, iremos fazer um **trello full-stack** com a **UI** amigável e totalmente escalável.

NÍVEL ÚNICO

Implementar a solução utilizando **NextJS**.

REQUISITOS DETALHADOS

➡ Comece o projeto com **create-next-app**:

CREATE-NEXT-APP



- ➡ Instale **typescript**, **ESLINT**, **tailwind**, e use a **folder apps** (nova atualização do nextJs a partir da v13).
- ➡ Rode o projeto em **dev** e veja se está tudo em ordem.
- ➡ Antes de iniciarmos qualquer código, uma estratégia que gosto de fazer para projetos um pouco mais complexos é fazer toda a instalação de infraestrutura na **cloud** antes de iniciar o código.
- ➡ Porque isso Yuri? Pelo fato de testarmos todas as pequenas variáveis de ponta a ponta e ver se o app está **deployando** sem problemas (**end-to-end**).
- ➡ Durante meus 11 anos de experiência já tive muitas experiências frustrantes de desenvolver o projeto inteiro e depois ter problemas na pipeline de deploy que me tomavam um tempo enorme e me gerava uma dor de cabeça enorme. Eu quero que você faça do jeito certo. **Isso vai te poupar tempo.**

- ➔ Desenvolva um **Header** com um logo similar ao do **Trello** (use nosso wireframe para se balizar). Não esqueça que se a imagem vier de um domínio externa na hora de inserir no **nextJS** você precisa liberar links externos no arquivo **next.config.js** (**whitelist**).
- ➔ Crie o **component Board**, esse componente será aonde iremos mover as tarefas entre os estados “to-do”, “doing”, “finished”
Instale a Lib `react-beautiful-dnd`. Documentação:

REACT-BEAUTIFUL-DND

- ➔ Essa **lib** ira nos auxiliar no **drag-and-drop**, não esqueça de instalar os **types de typescript**.
- ➔ Nosso **BaaS (backend-as-a-service)** usaremos uma ferramenta chamada **APPWRITE**. Diferente das outras ferramentas que usamos aqui como por exemplo **railway** ou **clerk** o **appwrite** é uma solução unificada para diversos serviços no **backend**. Segue a documentação.

APPWRITE.IO

- ➔ Crie um **bucket de images(storage)** e um **database** dentro da interface do **APPWRITE**, iremos usar ela na nossa aplicação.
- ➔ Primeiro é necessário criar um **database** e após uma **collection** dentro. Feito isso é necessário criar os **atributos** (como se você estivesse criando os schemas do banco).
- ➔ Vamos criar **title, status e descrição** (veja o **system design** para **melhor orientação**).
- ➔ Instale algum **state manager** (recomendo **zustand** ou **Jotai**, evite **REDUX**).

👉 Links úteis para você:

GITHUB



JOTAI



ZUSTAND



👉 Quando fizer o **fetch** dos dados do **DataBase** você deverá fazer um **MAP** dos dados para separar exatamente cada status do **ticket** em uma **array separada** (isso será necessário para renderizar as pilhas do "todo", "doing" e "finished").

👉 **Dica:** Dentro da lib ``react-beautiful-dnd`` observe com atenção os componentes que podem ser importados do contexto que são o ``Draggable`` e ``Droppable``.

👉 Crie um **state** somente o **Board** dentro da **store do zustand**.

- ➔ Dentro do **component** do **Board** crie um uma função que irá lidar com o **handle de cada board** (mover a posição dos boards não somente dos cards)
- ➔ A funcionalidade de **search(busca)** busca a lista de **arrays** de todos os itens dentro de todos os **boards**.
- ➔ Vamos usar a **API da OPENAI** para integrarmos nosso app com IA. Caso não tenha uma conta crie em:

OPENAI



- ➔ Vamos utilizar **headlessUI React** para operarmos com os modais, haja visto que ele tem uma **library** muito bem integrada com **TailwindCSS**.

HEADLESSUI

