

NÍVEL AVANÇADO



PROJETO 19

(CONTEÚDO DISPONÍVEL) {
SOCIAL MEDIA;
GRAPHQL;
(end);
})();

#PORTFÓLIOBOOSTPROGRAM

CONHECIMENTOS REQUIRIDOS:



FULL-STACK

WIREFRAME

Home Login Registro

registro

usuario

email



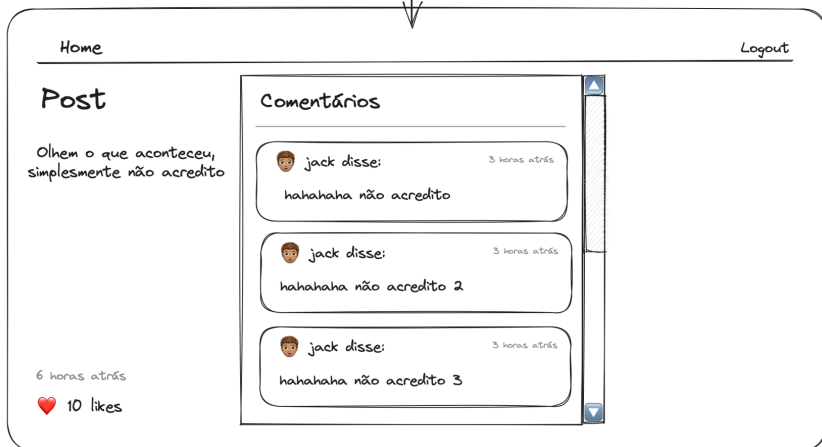
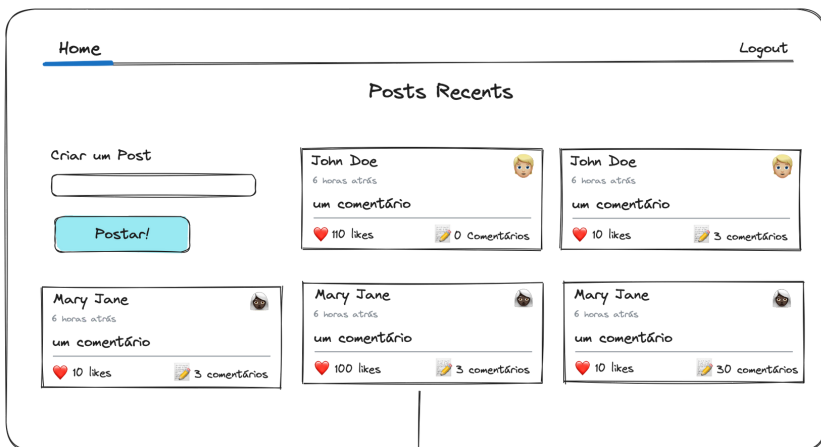
Home Login Registro

Login

usuario

email

WIREFRAME



WIREFRAME

database Schemas

```
const postSchema = new Schema({
  body: String,
  username: String,
  createdAt: String,
  comments: [
    {
      body: String,
      username: String,
      createdAt: String
    }
  ],
  likes: [
    {
      username: String,
      createdAt: String
    }
  ],
  user: {
    type: Schema.Types.ObjectId,
    ref: 'users'
  }
});

const userSchema = new Schema({
  username: String,
  password: String,
  email: String,
  createdAt: String
});
```

GraphQL TypeDefs

```
type Post {
  id: ID!
  body: String!
  createdAt: String!
  username: String!
  comments: [Comment!]
  likes: [Like!]
  likeCount: Int!
  commentCount: Int!
}

type Comment {
  id: ID!
  createdAt: String!
  username: String!
  body: String!
}

type Like {
  id: ID!
  createdAt: String!
  username: String!
}

type User {
  id: ID!
  email: String!
  token: String!
  username: String!
  createdAt: String!
}

input RegisterInput {
  username: String!
  password: String!
  confirmPassword: String!
  email: String!
}

type Query {
  getPosts: [Post!]
  getPost(postId: ID!): Post!
}

type Mutation {
  register(registerInput: RegisterInput!): User!
  login(username: String!, password: String!): User!
  createPost(body: String!): Post!
  deletePost(postId: ID!): String!
  createComment(postId: String!, body: String!): Post!
  deleteComment(postId: ID!, commentId: ID!): Post!
  likePost(postId: ID!): Post!
}

type Subscription {
  newPost: Post!
}
```


SOCIAL MEDIA GRAPHQL

Vamos criar uma rede social utilizando **GraphQL**! Esse projeto recomenda um conhecimento básico de graphql caso não tenha recomendo essa página para estudos:

APOLLOGRAPHQL

TECH STACK

- GraphQL
- TailwindCSS
- React
- BaaS (mongo, MySQL ou Postgres)
- Vercel
- ViteJS
- Apollo-server

LIBRARIES

- Apollo-Client
- DayJs



BRIEFING

Os **GraphQL servers** expõem seus dados como uma **API GraphQL** que seus aplicativos clientes podem consultar. Os GraphQL servers fazem o trabalho pesado de garantir que a quantidade adequada de dados seja recuperada usando o menor número possível de pesquisas de **banco de dados e chamadas de API**.

NÍVEL 1

Implementar uma solução **funcional** que esteja rodando em produção.

NÍVEL EXTRA

Implemente a mesma solução, mas de forma que o consumo de dados do **FE** utilize o cache do **Apollo** e desenvolva **testes unitários**.

REQUISITOS DETALHADOS



Como esse projeto é avançado vou deixar livre a sua escolha de **database** seja tanto o **MySQL** do **planetScale**:

PLANETSCALE





Postgres do [railway](#):

RAILWAY



Ou [mongodb cloud services](#):

MONGODB



O ponto em questão é manter esses [databases](#) na [cloud](#) para mantermos um [approach](#) orientado a [serverless](#).



Vamos criar um [repo](#) do zero sem [nextJS](#) sem nada, somente com [npm init](#) e após isso vamos instalar o [VITEJS](#) que será nosso [module bundler no front](#).

VITEJS



Na camada do server podemos instalar o [apollo-server](#).

APOLLO-SERVER



- ➔ Dentro da **folder** específica para o **backend**(seja **/api** ou **/graphql**) vamos criar uma **folder EXCLUSIVA** para o **graphql**. Dentro dela podemos criar uma **folder /resolvers** e o arquivo de **typeDefs** (**type definitions**).
- ➔ Uma **folder** para as **queries** também (**user, posts e etc**)
Após isso vamos subir nosso projeto no **vercel** e testar o **deploy** somente com um página em branco.
- ➔ Vamos configurar de forma **end-to-end** o nosso projeto para evitar ter problemas de ambiente e configuração no futuro.
- ➔ Nosso registro será via um **MUTATION** utilizando **bcrypt** e **JWT** (ver **typedefs** abaixo), mas resumindo bem teremos os mutations: **register, login, createPost, deletePost, createComment, deleteComment, likePost**.
- ➔ Para estilos recomendo **tailwindcss** pelo grau de facilidade.
- ➔ Teremos as seguintes páginas: **Home, Login e Register**.
- ➔ Crie um **context** usando **createContext** do **react** para persistir os dados do **user** no **FE**.
- ➔ Precisaremos de um **authReducer** também pois não iremos usar nenhuma ferramenta de autenticação, com o propósito de alavancar nosso conhecimento.



- Assim como precisaremos de um **reducer** também iremos precisar de um **provider** e utilizar o **useReducer hook**. Uma dica após criar todos os **mutations** e o **createPost** estiver funcionando como esperado para **atualizar real-time no front-end** podemos consultar o **cache** do **Apollo-server**:

APOLLOGRAPH

- Ou fazer **refetch**.
- Se você quiser correr o km extra implemente o consumo de **dados** do **FE** com cache do Apollo.
- Caso contrário, faça **refetch** do server.
- Crie as respectivas funcionalidades no **FRONT** de acordo com os mutations: **DeletePost**, **CreatePost**, **CreateComment**, **deleteComment** e **likePost**.
- Desenvolva **testes unitários** como parte extra.

