**REACT – REACTGRAM**

**Setup do projeto**

**npm init -y** – instalação de pacotes necessários

reactgram>backend>Terminal: **npm init -y**

**npm i bcryptjs** – biblioteca que permite manipular melhor as senhas(do usuário) que vai ter autenticação;

**cors** – nos permite receber requisições da mesma origem, pois trabalharemos com 2 aplicações localhost;

**dotenv** – onde ficará as variáveis de ambiente da aplicação, poderemos salvar por exemplo, o domínio do banco de dados aqui e conseguir resgatar ele em qualquer lugar facilmente;

**express** – framework back-end para criar de fato a API;

**express-validator** –trabalha entre as requisições para validar os dados, isso tira um pouco a lógica de validações dentro dos controllers;

**jsonwebtoken** –essencial para autenticação, gera e verifica tokens para autenticar usuários e deixar tokens ativos;

**mongoose** –ferramenta que será utilizada para trabalhar com banco de dados(não relacional) - MongoDB

**multer** – será utilizado para trabalhar com upload de imagens no geral.

**npm i --save-dev nodemon –** simulará um servidor local

 "server": "nodemon ./app.js"

Executando pacote nodemon no arquivo app.js

**Configurando o dotenv**

***app.js***

require("dotenv").config(); //nos da acesso ao arquivo '.env' onde estão as variáveis de ambiente

const port = process.env.PORT

***.env***

PORT=5000

**Rota de teste da API**

***routes>Router.js***

const express = require("express"); //chamada express

const router = express(); //chamada router

//test route

router.get('/', (req, res) => {

  //função anônima que aceita req e res é padrão

  res.send("API Working!");

})

***app.js – chamada da rota***

const router = require("./routes/Router.js");

app.use(router); //todas as rotas colocadas em routes>Router.js, servirão como base para nossa aplicação

**Postman**Tela de celular com aplicativo aberto

Descrição gerada automaticamenteTela de celular com aplicativo aberto

Descrição gerada automaticamenteImagem de vídeo game

Descrição gerada automaticamente com confiança média

**Importando middlewares**

***app.js***

//Solve CORS - quando as requisições são executadas pelo mesmo domínio

app.use(cors({ credentials: true, origin: "http://localhost:3000" })); //origin - de onde esta vindo a requisição

//Upload diretório - diretório de upload de imagens

app.use("/uploads", express.static(path.join(\_\_dirname, "/uploads")))//juntando nome do diretório atual(\_dirname) com /uploads

**Conexão com o banco**

- Criado novo projeto com o nome de ReactGram no MongoDB;

***config>db.js***

const dbUser = process.env.DB\_USER;

const dbPassword = process.env.DB\_PASSWORD;

//trazendo de '.env'

const conn = async () => {

  try {

    const dbConn = await mongoose.connect(//conectando ao banco de dados

      `mongodb+srv://${dbUser}:${dbPassword}@cluster.fvnirbj.mongodb.net/`

    );//concatenando variáveis q contém user e senha do mongodbproj

    console.log("Coonectou ao banco!")

    return dbConn;

  } catch (error) {//caso de erro na conexão

    console.log(error);

  }

};

conn();

module.exports=conn

***.env***

DB\_USER=aninha

DB\_PASSWORD=iYZJQzn7qExUPv7n

**Criando Model do usuário**

***models>User.js***

const {Schema} = mongoose

//o models é dividido em duas partes: //Schema- esquema(como ele é constituído)

//model - objeto que possui os métodos(inserir, deletar, ler)

const userSchema = new Schema({//dados do usuário

    name: String,

    email: String,

    password: String,

    profileImage: String,//foto de perfil do usuário

    bio: String

}, {

    timestamps:true//2 campós serão criados no model(createdate/updatedate)-com isso qnd usuário é criado e atualizado, o horário e a data são marcados

})

const User = mongoose.model("User", userSchema); //definindo model com o nome de "User" e com o Schema 'userSchema'

module.exports = User;//exportando user

**Criando Model da foto**

***models>Photo.js(mesma estrutura do usuário)***

const photoSchema = new Schema(

  {

    image: String, //será salvo o caminho da imagem, pois no bd não salva imagens

    title: String,

    likes: Array, //array porque os interesses são a quantidade e informações do usuário, isso nos leva a um array de objetos

    comments: Array, //qtd+conteudo\_coment+infor.user

    userId: mongoose.ObjectId, //id-usuario que inseriu a foto - .ObjectId: indica que não é uma string comum e sim de idmongoose(mais complexo)

    userName: String, //nome usuario

  },

  {

    timestamps: true,

  }

);

**Criando Controller do usuário**

***controllers>UserController.js***

***.env***

JWT\_SECRET=thisisoursecret

Quando trabalha-se com JWT é preciso ter um **SECRET**, é uma forma das pessoas que também usam JWT não hackearem o token.

Não há um valor ideal, mas quanto mais complexa(string), o token fica mais seguro.

Isso será utilizado na criação do token e na hora de descriptografar o token.