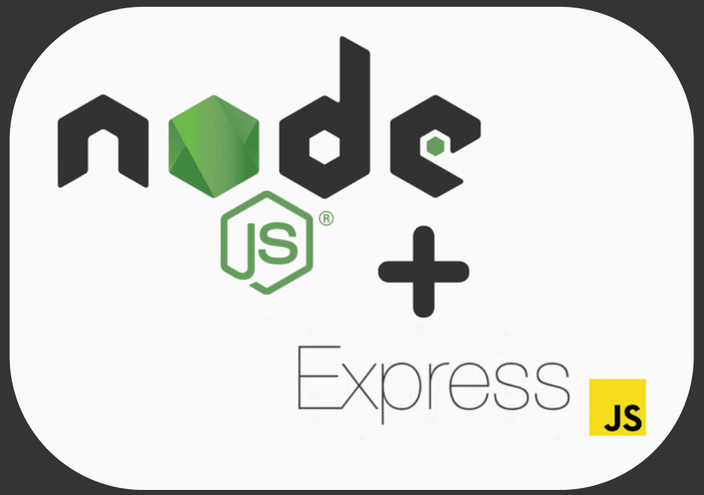
MangaList

Nome: Rodrigo Ruan Souza Viana
Disciplinas: XDES03 - Programação Web

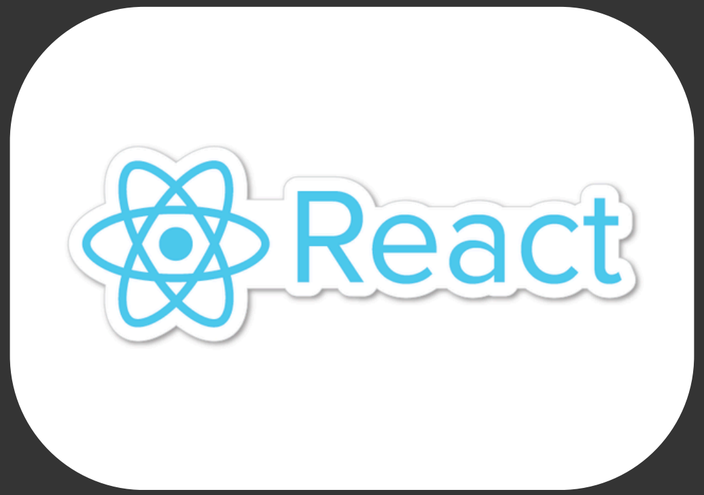
Tecnologias

Server-side (back-end)



Runtime: Node.js
Framework: Express.js

Client-side (front-end)

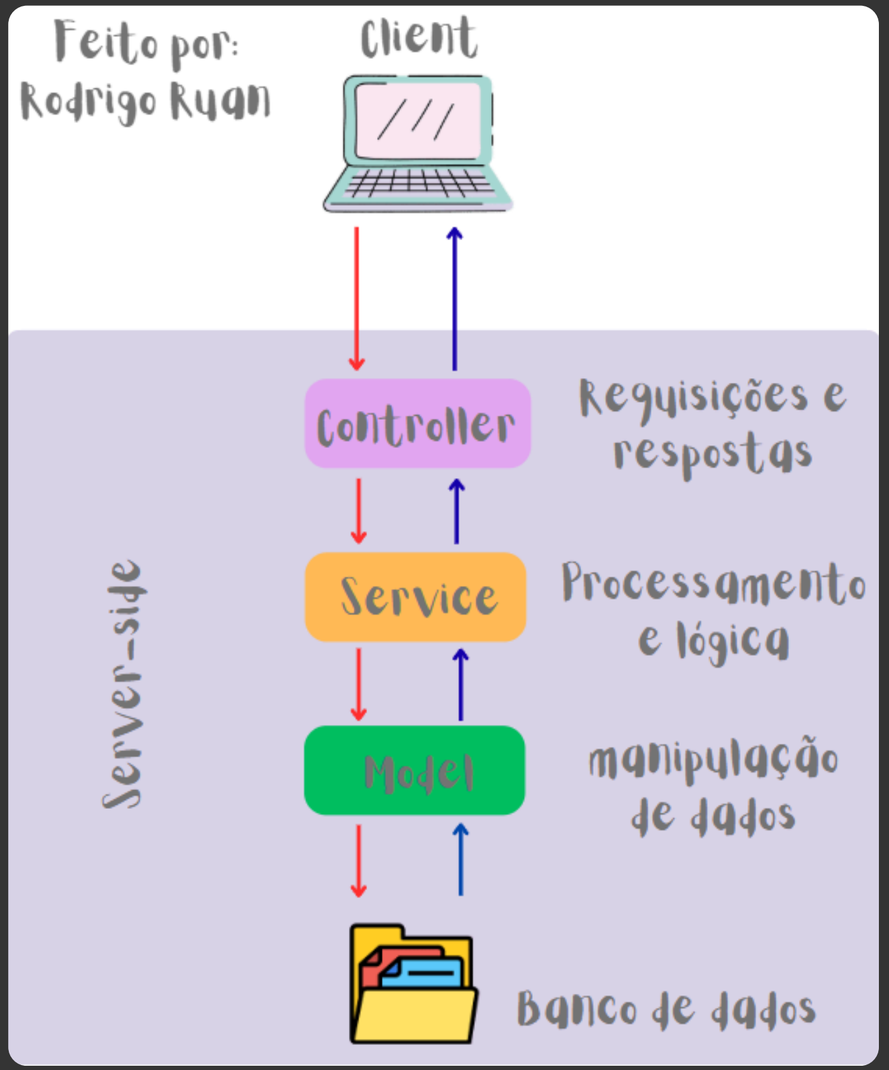


Biblioteca: React.js

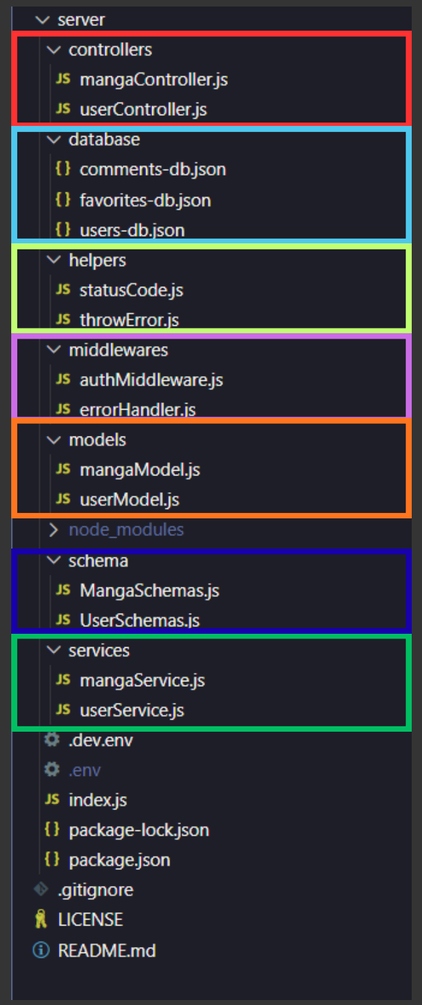
Server-side (Back-end)

- Padrão MSC (Model-Service-Controller):
  - Controller: Recebe requisições e envia respostas;
  - Service: Processamento de dados e regras de negócio;
  - Model: Comunicação com a base de dados.

Ferramentas



Organização dos arquivos



controllers: Recebem requisições e enviam respostas

database: persistência de dados

helpers: funções auxiliares

middlewares: “listeners” que agem ao ouvirem uma ação

models: manipulação e recuperação de dados da database

schema: validação dos dados nos corpos das requisições

services: processamento de dados e regras de negócio

Arquivo base index.js

```js
const express = require("express");
const cors = require("cors");
const cookieParser = require("cookie-parser");
require("dotenv").config({ path: `${__dirname}/.env` }); // Permite acesso ao .env

const app = express();

const PORT = process.env.PORT || 5000;
const CLIENT_PORT = process.env.CLIENT_PORT || 5001;

const UserController = require("./controllers/userController");
const MangaController = require("./controllers/mangaController");
const { AuthMiddleware } = require("./middlewares/authMiddleware");
const { errorHandler } = require("./middlewares/errorHandler");

app.use(express.json()); // Permite receber dados no formato JSON nas requisições
app.use(cookieParser()); // Permite ler token
app.use(
    cors({
        origin: `http://localhost:${CLIENT_PORT}`,
        credentials: true,
    })
); // Permite requisições do client-side
```

```js
// Rotas de usuário
app.post("/user/create", UserController.create);
app.post("/user/login", UserController.login);
app.post("/user/logout", UserController.logout);
app.get("/user/info", AuthMiddleware, UserController.getInfo);

// Rotas de manga
// 1º Favoritos
app.get("/manga/favorite", AuthMiddleware, MangaController.getFavorites);
app.post("/manga/favorite", AuthMiddleware, MangaController.setFavorite);
app.delete("/manga/favorite", AuthMiddleware, MangaController.removeFavorite);
// 2º Comentários
app.post("/manga/comment", AuthMiddleware, MangaController.addComment);
app.delete("/manga/comment", AuthMiddleware, MangaController.removeComment);
app.put("/manga/comment", AuthMiddleware, MangaController.editComment);

// Verificação se o servidor está no ar
app.get("/", (_req, res) => res.send("Hello World!"));
app.listen(PORT, () => console.log(`Server-side rodando na porta: ${PORT}`));

app.use(errorHandler); // Para tratamento de erros global
```

Configurações

No arquivo base index.js, definimos as configurações do nosso servidor, como:

- Inicialização do servidor;
- Rotas utilizadas;
- Permissões de requisição com cors;
- Configurações para uso de cookies;
- Leitura do .env;
- Utilização dos middlewars e controllers.

Middlewares

authMiddleware.js (apenas para rotas específicas)

```
const jwt = require("jsonwebtoken");

const { SECRET } = process.env;

function AuthMiddleware(req, res, next) {
    const error = { code: 401, message: "Token inválido ou expirado" };
    const token = req.cookies.token;

    if (!token) return res.status(error.code).json({ error: error.message });

    try {
        const decoded = jwt.verify(token, SECRET);
        req.token = decoded;
        next();
    } catch (_err) {
        return res.status(error.code).json({ error: error.message });
    }
}

module.exports = {
    AuthMiddleware,
};
```

Verifica se o token enviado pelo usuário é válido e permite continuar determinadas requisições que necessitam de autorização.
Caso seja válido, Seta em **req.token** as informações decodificadas do token: **id** e **email**.

errorHandler.js (global)

```
const statusCode = require("../helpers/statusCode");

function errorHandler(err, _req, res, _next) {
    return res.status(err?.status || statusCode.BAD_REQUEST).json({
        error: err?.message || "Erro interno no servidor",
    });
}

module.exports = {
    errorHandler,
};
```

Caso aconteça algum erro durante uma requisição, este middleware de tratamento de erro é acionado e retorna uma mensagem apropriedade de erro ao usuário.
A assinatura function (err, req, res, next) nos permite implícitamente dizer que é um middleware de tratamento de erros.

Helpers (auxiliares)

statusCode.js

```
module.exports = {
    OK: 200,
    CREATED: 201,
    BAD_REQUEST: 400,
    UNAUTHORIZED: 401,
    NOT_FOUND: 404,
    INTERNAL_ERROR: 500,
    CONFLICT: 409,
};
```

statusCode mantém um objeto do tipo "significado do status: código do status" para melhor reutilização e evitar “magic numbers” nas respostas.

throwError.js

```
const throwError = (code, msg) => {
    const err = new Error(msg);
    err.status = code;
    throw err;
};

module.exports = {
    throwError,
};
```

Função que nos auxilia gerar um erro para ser tratado no Middleware errorHandler. Utilizada quando o email já está cadastrado na criação de uma conta, se um mangá já está favoritado, etc.

Rotas de usuário

1º Requisição POST à rota “/user/create” com { email, username e password }

```
app.post("/user/create", UserController.create);
```

2º Controller.create

```
async function create(req, res) {
    await RegisterSchema.validate(req.body);
    const { email, username, password } = req.body;
    await UserService.create(username, email, password);
    res.status(statusCode.OK).json({
        message: "Usuário criado com sucesso.",
    });
}
```

3º Service.create

```
async function create(username, email, password) {
    const user = await UserModel.findByEmail(email);

    if (user) throwError(statusCode.CONFLICT, "E-mail já utilizado");

    const salt = await bcrypt.genSalt(10);
    const hashedPassword = await bcrypt.hash(password, salt);

    await UserModel.create(username, email, hashedPassword);
}
```

4º Model.create e .findByEmail

```
async function create(username, email, password) {
    const data = JSON.parse(await fs.readFile(DB_PATH, "utf-8"));

    const newUser = {
        id: randomUUID(),
        username,
        email,
        password,
        avatar_url: "https://cdn.noitatnemucod.net/avatar/100x100/zoro_normal/av-zz-10.jpeg",
        active: true,
        created_at: new Date().toISOString(),
        updated_at: new Date().toISOString(),
    };

    data.push(newUser);
    await fs.writeFile(DB_PATH, JSON.stringify(data));
}

async function findByEmail(email) {
    const data = JSON.parse(await fs.readFile(DB_PATH, "utf-8"));
    const user = data.find((user) => user.email === email);
    return user;
}
```

Rotas de usuário

1° Requisição POST à rota “/user/login” com { email, password }

```
app.post("/user/login", UserController.login);
```

2° Controller.login

```
async function login(req, res) {
    await LoginSchema.validate(req.body);
    const { email, password } = req.body;

    const response = await UserService.login(email, password);
    const { token, id, username, avatar_url } = response;

    res.cookie("token", token, {
        httpOnly: true,
        secure: false,
    });
    res.status(statusCode.OK).json({
        message: "Usuário logado com sucesso",
        user: { id, username, avatar_url },
    });
}
```

3° Service.login

```
async function login(email, password) {
    const user = await UserModel.findByEmail(email);
    const error_message = "E-mail ou senha inválidos";

    if (!user) throwError(statusCode.UNAUTHORIZED, error_message);

    const validPassword = await bcrypt.compare(password, user.password);
    if (!validPassword) throwError(statusCode.UNAUTHORIZED, error_message);

    const { id, username, avatar_url } = user;

    const token = jwt.sign({ user: { id, email } }, SECRET, { expiresIn: "7d" });

    return { token, id, username, avatar_url };
}
```

4° Model.findByEmail

```
async function findByEmail(email) {
    const data = JSON.parse(await fs.readFile(DB_PATH, "utf-8"));
    const user = data.find((user) => user.email === email);
    return user;
}
```

Rotas de usuário

1º Requisição POST à rota “/user/logout” sem corpo

```
app.post("/user/logout", UserController.logout);
```

2º Controller.logout

```
async function logout(_req, res) {
    res.clearCookie("token", {
        httpOnly: true,
        secure: false,
    });
    res.status(statusCode.OK).json({
        message: "Usuário deslogado com sucesso",
    });
}
```

Rotas de usuário

1º Requisição GET à rota “/user/info”

```
app.get("/user/info", AuthMiddleware, UserController.getInfo);
```

2º Controller.getInfo

```
async function getInfo(req, res) {
    const data = req.token;

    const response = await UserService.getInfo(data?.user?.id);

    res.status(statusCode.OK).json({
        message: "Informações do usuário carregadas com sucesso",
        user: response,
    });
}
```

3º Service.getInfo

```
async function getInfo(id) {
    const user = await UserModel.findById(id);

    if (!user || !user.active) throwError(statusCode.NOT_FOUND, "Usuário não encontrado");

    const { password, active, ...publicData } = user;

    return publicData;
}
```

4º Model.findByEmail

```
async function findById(id) {
    const data = JSON.parse(await fs.readFile(DB_PATH, "utf-8"));
    const user = data.find((user) => user.id === id);
    return user;
}
```

Rotas de Favoritos

1° Requisição GET à rota “/manga/favorite”

```
app.get("/manga/favorite", AuthMiddleware, MangaController.getFavorites);
```

2° Controller.getFavorites

```
async function getFavorites(req, res) {
    const data = req.token;

    const response = await MangaService.getFavorites(data?.user?.id);

    res.status(statusCode.OK).json({
        user_id: data.user.id,
        favorites: response,
    });
}
```

3° Service.getFavorites

```
const getFavorites = async (user_id) => {
    const userFavorites = await MangaModel.getFavorites(user_id);
    const comments = await MangaModel.getComments(
        userFavorites.map((favorite) => favorite.manga_id)
    );

    const favoritesWithComments = userFavorites.map((favorite) => {
        favorite["comments"] = comments.filter((comment) => favorite.manga_id === comment.manga_id);
        return favorite;
    });

    return favoritesWithComments;
};
```

4° Model.getFavorites e .getComments

```
async function getFavorites(user_id) {
    const data = JSON.parse(await fs.readFile(DB_FAVORITES_PATH, "utf-8"));
    const dataFiltered = data.filter((favorite) => favorite.user_id === user_id);
    return dataFiltered;
}

async function getComments(manga_ids) {
    const data = JSON.parse(await fs.readFile(DB_COMMENTS_PATH, "utf-8"));
    const filteredData = data.filter((comment) => manga_ids.includes(comment.manga_id));
    return filteredData;
}
```

Rotas de Favoritos

1º Requisição POST à rota “/manga/favorite”

```
app.post("/manga/favorite", AuthMiddleware, MangaController.setFavorite);
```

2º Controller.setFavorite

```
async function setFavorite(req, res) {
    await FavoriteSchema.validate(req.body);

    const data = req.token;

    const response = await MangaService.setFavorite(data?.user?.id, req.body?.manga_id);

    res.status(statusCode.OK).json({
        message: "Mangá favoritado com sucesso.",
        data: response,
    });
}
```

3º Service.getFavorites

```
const setFavorite = async (user_id, manga_id) => {
    const userFavorites = await MangaModel.getFavorites(user_id);

    const alreadyFavorite = userFavorites.find(
        (favorite) => favorite.user_id === user_id && favorite.manga_id === manga_id
    );
    if (alreadyFavorite) throwError(statusCode.CONFLICT, "Mangá já favoritado");

    return await MangaModel.setFavorite(user_id, manga_id);
};
```

4º Model.getFavorites e .getComments

```
async function getFavorites(user_id) {
    const data = JSON.parse(await fs.readFile(DB_FAVORITES_PATH, "utf-8"));
    const dataFiltered = data.filter((favorite) => favorite.user_id === user_id);
    return dataFiltered;
}

async function setFavorite(user_id, manga_id) {
    const data = JSON.parse(await fs.readFile(DB_FAVORITES_PATH, "utf-8"));

    const favoritedObject = {
        user_id,
        manga_id,
        favorited_at: new Date().toISOString(),
    };

    data.push(favoritedObject);
    await fs.writeFile(DB_FAVORITES_PATH, JSON.stringify(data));

    return favoritedObject;
}
```

Rotas de Favoritos

1° Requisição DELETE à rota “/manga/favorite”

```
app.delete("/manga/favorite", AuthMiddleware, MangaController.removeFavorite);
```

2° Controller.removeFavorite

```
async function removeFavorite(req, res) {
    await FavoriteSchema.validate(req.body);

    const data = req.token;

    await MangaService.removeFavorite(data?.user?.id, req.body?.manga_id);

    res.status(statusCode.OK).json({
        message: "Manga removido dos favoritos com sucesso.",
    });
}
```

3° Service.removeFavorite

```
const removeFavorite = async (user_id, manga_id) => {
    await MangaModel.removeFavorite(user_id, manga_id);
};
```

4° Model.removeFavorite

```
async function removeFavorite(user_id, manga_id) {
    const raw = await fs.readFile(DB_FAVORITES_PATH, "utf-8");
    const data = JSON.parse(raw);

    const filteredData = data.filter((favorite) => !(favorite.manga_id === manga_id && favorite.user_id === user_id));

    await fs.writeFile(DB_FAVORITES_PATH, JSON.stringify(filteredData));
}
```

Rotas de Comentários

1° Requisição POST à rota “/manga/comment”

```
app.post("/manga/comment", AuthMiddleware, MangaController.addComment);
```

2° Controller.addComment

```
async function addComment(req, res) {
    await AddCommentSchema.validate(req.body);

    const data = req.token;

    const response = await MangaService.addComment(
        data?.user?.id,
        data?.user?.email,
        req.body?.manga_id,
        req.body?.comment
    );

    res.status(statusCode.OK).json({
        message: "Comentário adicionado com sucesso.",
        data: response,
    });
}
```

3° Service.addComment

```
const addComment = async (user_id, email, manga_id, comment) => {
    return await MangaModel.addComment(user_id, email, manga_id, comment);
};
```

4° Model.addComment

```
async function addComment(user_id, email, manga_id, comment) {
    const data = JSON.parse(await fs.readFile(DB_COMMENTS_PATH, "utf-8"));

    const commentObject = {
        id: randomUUID(),
        user_id,
        manga_id,
        comment,
        email,
        created_at: new Date().toISOString(),
        updated_at: new Date().toISOString(),
    };

    data.push(commentObject);
    await fs.writeFile(DB_COMMENTS_PATH, JSON.stringify(data));

    return commentObject;
}
```

Rotas de Comentários

1° Requisição DELETE à rota “/manga/comment”

```
app.delete("/manga/comment", AuthMiddleware, MangaController.removeComment);
```

2° Controller.removeComment

```
async function removeComment(req, res) {
    await RemoveCommentSchema.validate(req.body);

    const data = req.token;

    await MangaService.removeComment(data?.user?.id, req.body?.comment_id);

    res.status(statusCode.OK).json({
        message: "Comentário removido com sucesso.",
    });
}
```

3° Service.removeComment

```
const removeComment = async (user_id, comment_id) => {
    await MangaModel.removeComment(user_id, comment_id);
};
```

4° Model.addComment

```
async function removeComment(user_id, comment_id) {
    const data = JSON.parse(await fs.readFile(DB_COMMENTS_PATH, "utf-8"));
    const filteredData = data.filter((comment) => !(comment_id === comment.id && comment.user_id === user_id));
    await fs.writeFile(DB_COMMENTS_PATH, JSON.stringify(filteredData));
}
```

Rotas de Comentários

1° Requisição PUT à rota “/manga/comment”

```
app.put("/manga/comment", AuthMiddleware, MangaController.editComment);
```

2° Controller.editComment

```
async function editComment(req, res) {
    await EditCommentSchema.validate(req.body);

    const data = req.token;

    const response = await MangaService.editComment(data?.user?.id, req.body?.comment_id, req.body?.comment);

    res.status(statusCode.OK).json({
        message: "Comentário editado com sucesso.",
        data: response,
    });
}
```

3° Service.editComment

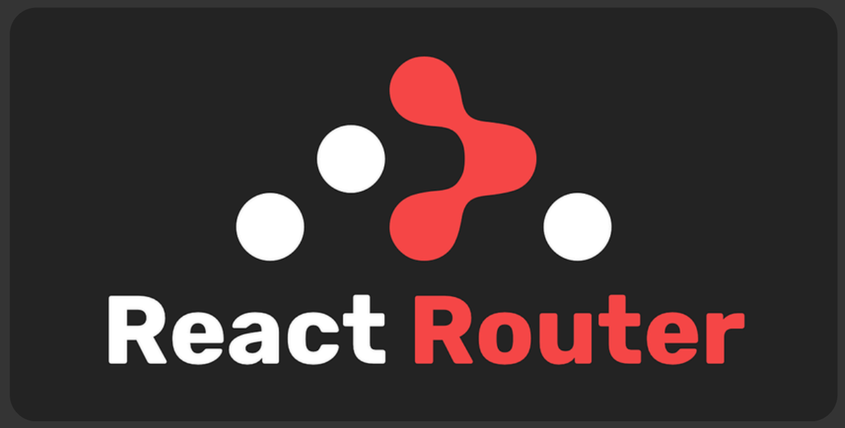
```
const editComment = async (user_id, comment_id, new_comment) => {
    await MangaModel.editComment(user_id, comment_id, new_comment);
};
```

4° Model.addComment

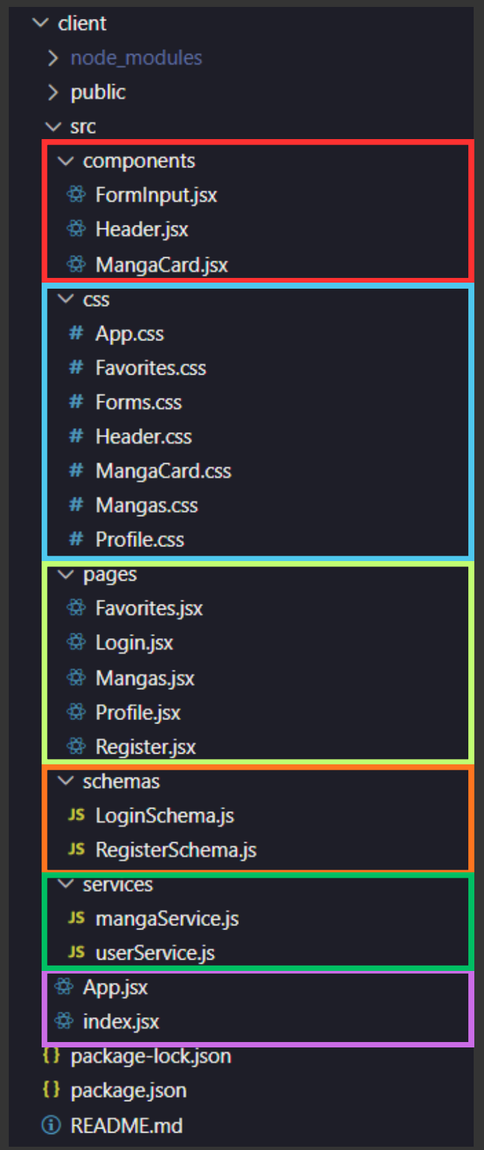
```
async function editComment(user_id, comment_id, new_comment) {
    const data = JSON.parse(await fs.readFile(DB_COMMENTS_PATH, "utf-8"));
    const modifiedData = data.map((comment) => {
        if (comment_id === comment.id && comment.user_id === user_id) {
            comment["comment"] = new_comment;
            comment["updated_at"] = new Date().toISOString();
        }

        return comment;
    });

    await fs.writeFile(DB_COMMENTS_PATH, JSON.stringify(modifiedData));
}
```

Client-side (Front-end)

Ferramentas



Organização dos arquivos



components: Componentes reutilizáveis

css: arquivos de estilização

pages: páginas da aplicação

models: manipulação e recuperação de dados da database

services: requisições aos servidores local e do mangadex

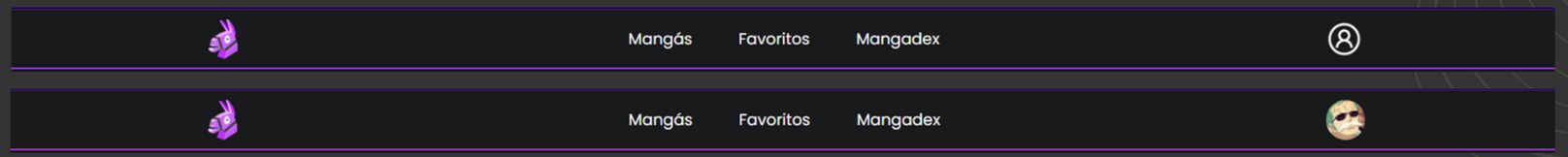
configurações iniciais e declaração das rotas

App.jsx

Configuração das rotas

```jsx
// App.js
import { BrowserRouter, Routes, Route } from "react-router";

import "./css/App.css";

import Mangas from "./pages/Mangas";
import Login from "./pages/Login";
import Register from "./pages/Register";
import Favorites from "./pages/Favorites";
import Profile from "./pages/Profile";

const App = () => (
    <BrowserRouter>
        <Routes>
            <Route path="/" element={<Mangas />} />
            <Route path="/login" element={<Login />} />
            <Route path="/register" element={<Register />} />
            <Route path="/favorites" element={<Favorites />} />
            <Route path="/profile" element={<Profile />} />
        </Routes>
    </BrowserRouter>
);

export default App;
```

Header.jsx




```jsx
function Header() {
    const user_info = JSON.parse(sessionStorage.getItem("user_info"));
    const user_logged = sessionStorage.getItem("user_logged");

    return (
        <header>
            <Link id="logo-container" to="/">
                <img src="https://img.icons8.com/?size=100&id=p6iYcd56bUym&format=png&color=000000"></img>
            </Link>

            <div id="links-container">
                <Link className="nav-link" to="/">
                    Mangás
                </Link>
                <Link className="nav-link" to="/favorites">
                    Favoritos
                </Link>
                <a className="nav-link" target="_blank" href="https://mangadex.org/">
                    Mangadex
                </a>
            </div>

            <div className="user-info">
                <Link id="user-container" to={user_logged ? "/profile" : "/login"}>
                    <img
                        className="user-avatar"
                        src={
                            user_logged && user_info?.avatar_url
                                ? user_info.avatar_url
                                : "https://img.icons8.com/?size=100&id=15263&format=png&color=ffffff"
                        }
                    />
                </Link>
            </div>
        </header>
    );
}
```

MangaCard.jsx

```jsx
function capitalize(word) {
    if (!word) return "";
    return word.charAt(0).toUpperCase() + word.slice(1);
}

function MangaCard({ attributes }) {
    const navigate = useNavigate();
    const [favorited, setFavorited] = React.useState(false);

    async function addToFavorites() {
        if (!sessionStorage.getItem("user_logged")) return navigate("/login");
        if (favorited) return;

        try {
            await addMangaToFavorite(attributes.id);
        } catch (err) {
            const UNAUTHORIZED_STATUS = 401;
            if (err.status == UNAUTHORIZED_STATUS) {
                sessionStorage.clear();
                return navigate("/login");
            }
        }

        setFavorited(true);
    }

    async function removeFavorite() {
        await removeMangaFromFavorite(attributes.id);
        const { favorites, setFavorites } = attributes;
        setFavorites(favorites.filter((favorite) => favorite.id != attributes.id));
    }
```

Berserk

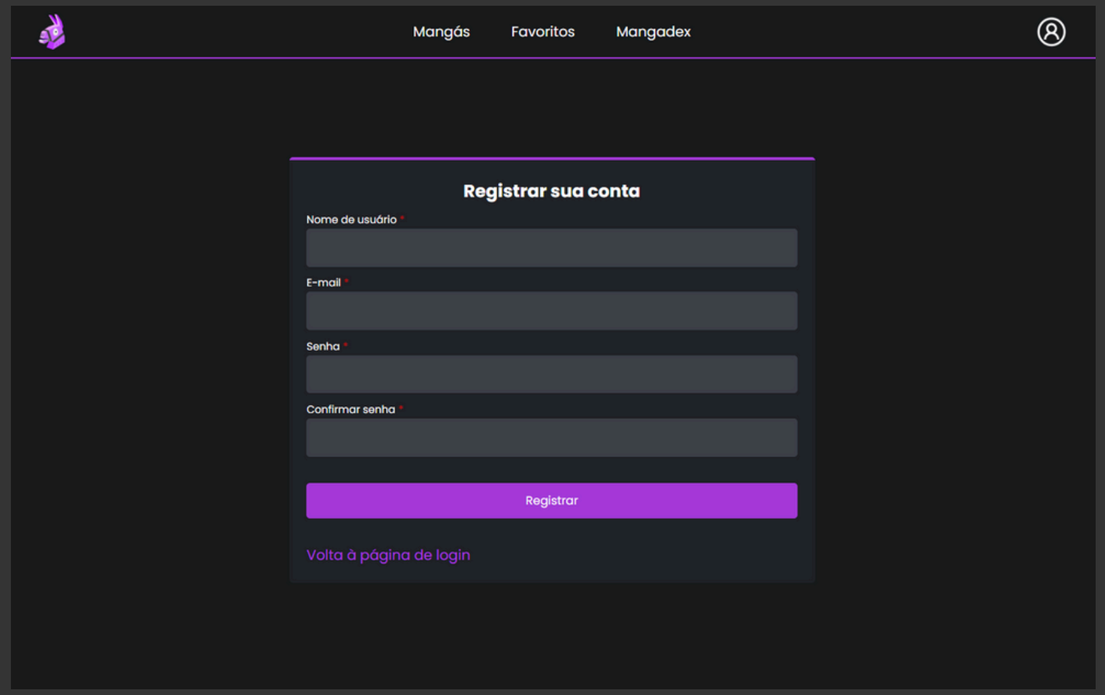
1989 Seinen • Ongoing

Thriller Award Winning Monsters Action

Guts, known as the Black Swordsman, seeks sanctuary from the demonic forces attracted to him and his woman because of a demonic mark on their necks, and also vengeance against the man who branded him as an unholy sacrifice. Aided only by his titanic strength gained from a harsh childhood lived with mercenaries, a gigantic sword, and an iron prosthetic left hand, Guts must struggle against his bleak destiny, all the while fighting with a rage that might strip him of his humanity. ___ Won the 6th Osamu Tezuka Cultural Prize Excellence Award in 2002. Note: Following Miura Kenta's death in 2021, the series has be

```jsx
    return (
        <div className="manga-card">
            <div className="manga-card-img-container">
                <img
                    alt="capa mangá"
                    src={`https://uploads.mangadex.org/covers/${attributes.id}/${attributes.cover}.256.jpg`}
                ></img>
            </div>

            <div className="manga-attributes">
                <div className="title-container">
                    <p className="manga-title">{Object.values(attributes.title)[0]}</p>
                    <span
                        onClick={attributes?.isFavoritesPage ? removeFavorite : addToFavorites}
                        className="favorite-span"
                    >
                        <img
                            alt="icone de favoritar"
                            src={
                                attributes?.isFavoritesPage
                                    ? "https://img.icons8.com/?size=100&id=KPhFC2OwpbWV&format=png&color=FF0000"
                                    : favorited
                                    ? "https://img.icons8.com/?size=100&id=10287&format=png&color=A53BD9"
                                    : "https://img.icons8.com/?size=100&id=581&format=png&color=A53BD9"
                            }
                        />
                    </span>
                </div>
                <div className="manga-info">
                    <p>
                        <img
                            alt="icone do ano de lançamento do mangá"
                            id="calendar"
                            src="https://img.icons8.com/?size=100&id=10053&format=png&color=ffffff"
                        ></img>
                        {attributes.year}
                    </p>
                    <p>{capitalize(attributes?.publicationDemographic)} </p>
                    <p className="manga-card-status">
                        <span className={`${attributes?.status}-status`}>&#9679;</span>
                        {capitalize(attributes?.status)}
                    </p>
                </div>
                <div className="tags-container">
                    {attributes.tags.slice(0, 4).map((tag, idx) => {
                        return <p key={idx}>{tag?.attributes?.name?.en}</p>;
                    })}
                </div>
                <p className="manga-desc">
                    {attributes?.description?.en
                        ? attributes?.description?.en
                        : Object.values(attributes.description)[0]}
                </p>
            </div>
        </div>
    );
}
```

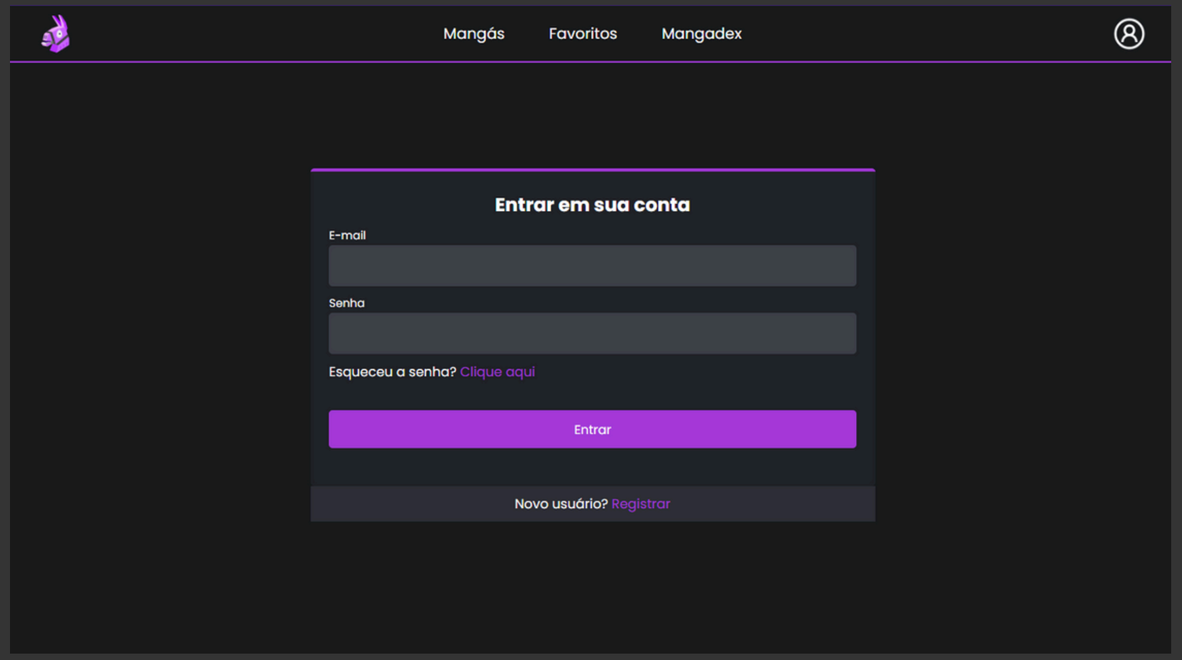
Register.jsx




```jsx
function Register() {
    const navigate = useNavigate();
    const [user, setUser] = React.useState({
        username: "",
        email: "",
        password: "",
        confirmPassword: "",
    });
    const [errorMessage, setErrorMessage] = React.useState("");

    async function submitRegister(e) {
        e.preventDefault();

        try {
            await RegisterSchema.validate(user);
            const response = await createUser(user);
            if (response.status === 200) navigate("/login");
        } catch (err) {
            const serverErrorMessage = err?.response?.data?.error;
            setErrorMessage(serverErrorMessage || err.message);
        }
    }
```

```jsx
    return (
        <>
            <Header />
            <div className="form-body">
                <div className="form-container">
                    <div className="input-section">
                        <h1>Registrar sua conta</h1>
                        <form className="form" onSubmit={(e) => submitRegister(e)}>
                            <FormInput
                                setUser={setUser}
                                user={user}
                                type="text"
                                label="Nome de usuário "
                                id="username"
                            />
                            <FormInput user={user} setUser={setUser} type="email" label="E-mail " id="email" />
                            <FormInput setUser={setUser} user={user} type="password" label="Senha " id="password" />
                            <FormInput
                                setUser={setUser}
                                user={user}
                                type="password"
                                label="Confirmar senha "
                                id="confirmPassword"
                            />
                            <button className="register-button">Registrar</button>
                            <p className="error-message">{errorMessage}</p>
                        </form>
                        <Link id="return-to-login-page" to="/login">
                            Volta à página de login
                        </Link>
                    </div>
                </div>
            </div>
        </>
    );
}
```

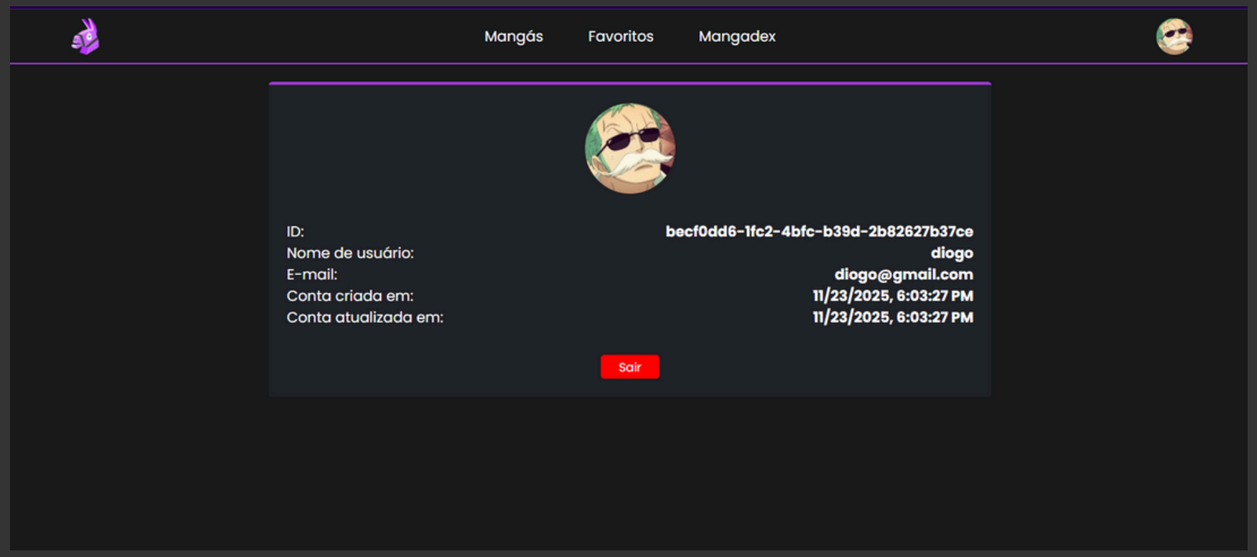
Login.jsx




```jsx
function Login() {
    const navigate = useNavigate();
    const [email, setEmail] = React.useState("");
    const [password, setPassword] = React.useState("");
    const [errorMessage, setErrorMessage] = React.useState("");

    async function submitLogin(e) {
        e.preventDefault();

        try {
            const dataToLogin = { email, password };
            await LoginSchema.validate(dataToLogin);
            const response = await loginUser(dataToLogin);

            if (response.status === 200) {
                const user_info = JSON.stringify(response.data.user);
                sessionStorage.setItem("user_logged", true);
                sessionStorage.setItem("user_info", user_info);
                navigate("/favorites");
            }
        } catch (err) {
            const serverErrorMessage = err?.response?.data?.error;
            setErrorMessage(serverErrorMessage || err.message);
        }
    }
```

```jsx
    return (
        <>
            <Header />
            <div className="form-body">
                <div className="form-container">
                    <div className="input-section">
                        <h1>Entrar em sua conta</h1>
                        <form className="form" onSubmit={(e) => submitLogin(e)}>
                            <label htmlFor="email">E-mail</label>
                            <input id="email" type="email" onChange={(e) => setEmail(e.target.value)}></input>
                            <label htmlFor="password">Senha</label>
                            <input id="password" type="password" onChange={(e) => setPassword(e.target.value)}></input>
                            <p className="forgot-password">
                                Esqueceu a senha?{" "}
                                <span
                                    className="forgot-password-click"
                                    onClick={() => alert("Entre em contato com o administrador!")}
                                >
                                    Clique aqui
                                </span>
                            </p>
                            <button className="login-button">Entrar</button>
                            <p className="error-message">{errorMessage}</p>
                        </form>
                    </div>
                    <div className="section-registrar">
                        <span>Novo usuário? </span>
                        <Link className="register-link" to="/register">
                            Registrar
                        </Link>
                    </div>
                </div>
            </div>
        </>
    );
}
```

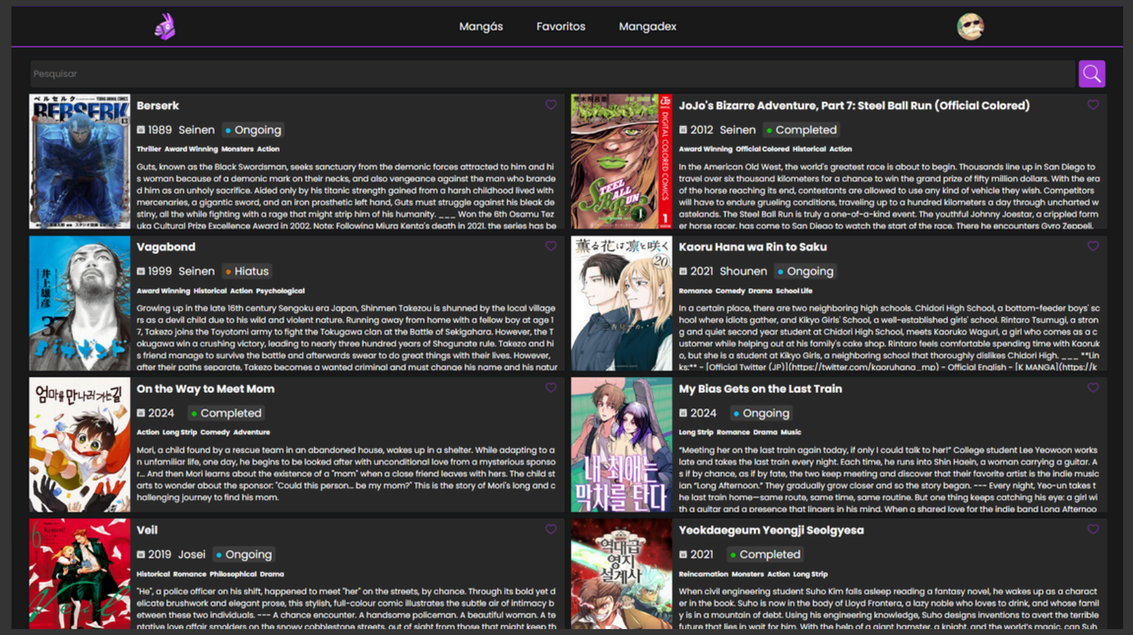
Profile.jsx




```
    return (
        <>
            <Header />
            <div className="profile-body">
                <div className="profile-container">
                    <div className="pfp-container">
                        <img className="profile-picture" src={user.avatar_url} />
                    </div>
                    <div className="pf-info-container">
                        <InfoRow label="ID:" value={user.id} />
                        <InfoRow label="Nome de usuário:" value={user.username} />
                        <InfoRow label="E-mail:" value={user.email} />
                        <InfoRow label="Conta criada em:" value={new Date(user.created_at).toLocaleString()} />
                        <InfoRow label="Conta atualizada em:" value={new Date(user.updated_at).toLocaleString()} />
                        <p className="error-message">{errorMessage}</p>
                        <button onClick={deslogar} className="logout-button">
                            Sair
                        </button>
                    </div>
                </div>
            </div>
        </>
    );
}
```

```
function Profile() {
    const navigate = useNavigate();
    const [user, setUser] = React.useState({});
    const [errorMessage, setErrorMessage] = React.useState(null);

    async function deslogar() {
        try {
            const response = await logoutUser();

            if (response.status === 200) {
                sessionStorage.clear();
                navigate("/");
            }
        } catch (err) {
            setErrorMessage(err.message);
        }
    }

    React.useEffect(() => {
        if (!sessionStorage.getItem("user_logged")) return navigate("/login");

        getUserInfo().then((data) => {
            if (data["error"]) return deslogar();
            setUser(data);
        });
    }, []);

    const InfoRow = ({ label, value }) => {
        return (
            <div>
                <p>{label}</p>
                <p className="info">{value}</p>
            </div>
        );
    };
```

Mangas.jsx




```jsx
function Home() {
    const [mangas, setMangas] = React.useState([]);
    const [page, setPage] = React.useState(0);
    const [loading, setLoading] = React.useState(false);
    const [mangaName, setMangaName] = React.useState("");

    function buscaMangas() {
        setLoading(true);
        getMangas(mangaName, page)
            .then((data) => setMangas(data))
            .finally(() => setLoading(false));
    }

    React.useEffect(() => {
        buscaMangas();
    }, [page]);
```

```jsx
    return (
        <>
            <Header />
            <div className="main-container">
                <div className="mangas-section">
                    <div className="search-bar-container">
                        <input
                            id="search-anime-bar"
                            type="text"
                            placeholder="Pesquisar"
                            onChange={(e) => setMangaName(e.target.value)}
                            onKeyDown={(e) => e.key === "Enter" && buscaMangas()}
                        />
                        <button onClick={buscaMangas}>
                            <img src="https://img.icons8.com/?size=100&id=e4NkZ7kWAD7f&format=png&color=ffffff" />
                        </button>
                    </div>
                    <div className="mangas-container">
                        {loading ? (
                            <p id="loading-msg">Carregando...</p>
                        ) : (
                            mangas.map((manga) => {
                                const { attributes, relationships, id, type } = manga;
                                const cover = relationships.find(({ type }) => type === "cover_art")?.attributes
                                    ?.fileName;
                                const mangaData = { ...attributes, id, type, relationships, cover };
                                return <MangaCard key={id} attributes={mangaData} />;
                            })
                        )}
                    </div>
                </div>
                <div className="navigate-container">
                    <button className="arrow-button" onClick={() => setPage(Math.max(0, page - 1))}>
                        <img src="https://img.icons8.com/?size=100&id=39944&format=png&color=ffffff"></img>
                    </button>
                    <input value={page + 1} type="text" disabled></input>
                    <button className="arrow-button" onClick={() => setPage(page + 1)}>
                        <img src="https://img.icons8.com/?size=100&id=8OOIdPe6NXVi&format=png&color=ffffff"></img>
                    </button>
                </div>
            </div>
        </>
    );
}
```

Favorites.jsx

Mangás Favoritos Mangadex

JoJo's Bizarre Adventure, Part 7: Steel Ball Run (Official Colored)

2012 Seinen • Completed

Award Winning Official Colored Historical Action

In the American Old West, the world's greatest race is about to begin. Thousands line up in San Diego to travel over six thousand kilometers for a chance to win the grand pri ze of fifty million dollars. With the era of the horse reaching its end, contestants are allowed to use any kind of vehicle they wish. Competitors will have to endure grueling co nditions, traveling up to a hundred kilometers a day through uncharted wastelands. The Steel Ball Run is truly a one-of-a-kind event. The youthful Johnny Joestar, a cripple d former horse racer, has come to San Diego to watch the start of the race. There he encounters Gyro Zeppeli, a racer with two steel balls at his waist instead of a gun. John ny witnesses Gyro using one of his steel balls to unleash a fantastical power, compelling a man to fire his gun at himself during a duel. In the midst of the action, Johnny ha ppens to touch the steel ball and feels a power surging through his legs, allowing him to stand up for the first time in two years. Vowing to find the secret of the steel balls, J

Adicione seu comentário aqui... Enviar

Comentários

diogo@gmail.com Editar Excluir

Muito maneiro!

diogo@gmail.com Salvar Excluir

Adorei!!!

Berserk

1989 Seinen • Ongoing

Thriller Award Winning Monsters Action

Guts, known as the Black Swordsman, seeks sanctuary from the demonic forces attracted to him and his woman because of a demonic mark on their necks, and also veng eance against the man who branded him as an unholy sacrifice. Aided only by his titanic strength gained from a harsh childhood lived with mercenaries, a gigantic sword, and an iron prosthetic left hand, Guts must struggle against his bleak destiny, all the while fighting with a rage that might strip him of his humanity. ___ Won the 6th Osam u Tezuka Cultural Prize Excellence Award in 2002. Note: Following Miura Kenta's death in 2021, the series has been taken over by Kouji Mori, who supervises the series with art done by Studio Gaga.

Adicione seu comentário aqui... Enviar

Comentários

Favorites.jsx

```
    return (
        <>
            <Header />
            <div className="favorites-container">
                <p className="favorite-message">{loading ? "Carregando..." : favorites.length == 0 && "Nenhum favorito encontrado"}</p>
                {favorites.map((manga) => {
                    const { attributes, relationships, id, type } = manga;
                    const cover = relationships.find(({ type }) => type === "cover_art")?.attributes?.fileName;
                    const mangaData = { ...attributes, id, type, relationships, cover, isFavoritesPage: 1, setFavorites, favorites, };
                    return (
                        <div key={id} className="manga-card-container">
                            <MangaCard attributes={mangaData} />
                            <form className="manga-review" onSubmit={(e) => addComment(e, id)}>
                                <input
                                    placeholder="Adicione seu comentário aqui..."
                                    type="text"
                                    onChange={(e) => setComments({ ...comments, [id]: e.target.value })}
                                />
                                <button>Enviar</button>
                            </form>
                            <section className="comments-section">
                                <h3>Comentários</h3>
                                <div className="comments-container">
                                    {manga?.comments.map(({ id, user_id, email, comment }) => (
                                        <div key={id}>
                                            <div className="comment-info">
                                                <p>{email}</p>
                                                {user_id == userId && (
                                                    <div className="edit-comment-container">
                                                        <button
                                                            onClick={() => !editMode[id] ? setEditMode({ ...editMode, [id]: true, }) : updateComment(id)}
                                                            className="edit-comment-button">
                                                            {editMode[id] ? "Salvar" : "Editar"}
                                                        </button>
                                                        <button
                                                            onClick={() => removeComment(id)} className="remove-comment-button">
                                                            Excluir
                                                        </button>
                                                    </div>
                                                )}
                                            </div>
                                            {editMode[id] ? (<input onChange={(e) => setComments({ ...comments, [id]: e.target.value })} className="comment-input" defaultValue={comment}/>) : (
                                                <p className="comment" key={id}>{comment}</p>)}
                                        </div>
                                    ))}
                                </div>
                            </section>
                        </div>
                    );
                })}
            </div>
        </>
    );
}
```

```
function Favorites() {
    const navigate = useNavigate();
    const [userId, setUserId] = React.useState(null);
    const [favorites, setFavorites] = React.useState([]);
    const [comments, setComments] = React.useState({});
    const [editMode, setEditMode] = React.useState({});
    const [loading, setLoading] = React.useState(false);

    async function getFavorites() {
        setLoading(true);
        try {
            const favorited_mangas = await getFavoriteMangas();
            const favorite_ids = favorited_mangas?.data?.favorites;
            if (favorite_ids?.length == 0) return;

            const favoritesFetched = await getFavoritesMangadex(favorite_ids);
            const favoritesWithComments = favoritesFetched?.data?.data.map((favorite) => {
                favorite["comments"] = favorite_ids.find(({ manga_id }) => favorite.id == manga_id)?.comments || [];
                return favorite;
            });

            setFavorites(favoritesWithComments);
        } catch (err) {
            const UNAUTHORIZED_STATUS = 401;
            if (err.status == UNAUTHORIZED_STATUS) {
                return navigate("/login");
            }
        } finally {
            setLoading(false);
        }
    }

    async function addComment(e, id) {
        e.preventDefault();
        if (!comments[id]) return;

        await addMangaComment(id, comments[id]);

        e.target[0].value = "";
        comments[id] = "";

        getFavorites();
    }

    async function updateComment(id) {
        if (!comments[id]) return;

        await updateMangaComment(id, comments[id]);

        getFavorites();
        setEditMode({ ...editMode, [id]: false });
    }

    async function removeComment(comment_id) {
        await deleteMangaComment(comment_id);
        getFavorites();
    }

    React.useEffect(() => {
        if (!sessionStorage.getItem("user_logged")) {
            return navigate("/login");
        }
        setUserId(JSON.parse(sessionStorage.getItem("user_info")).id);
        getFavorites();
    }, []);
```

OBRIGADO!