NativeNews: Plataforma Mobile de Notícias

Ger. Desenvolvimento

José Roberto

https://github.com/joseroberto42

Scrum Master

Diego Silva

https://github.com/diego-humberto

Ger. Configurações

Gabriel

https://github.com/GabsFerr22

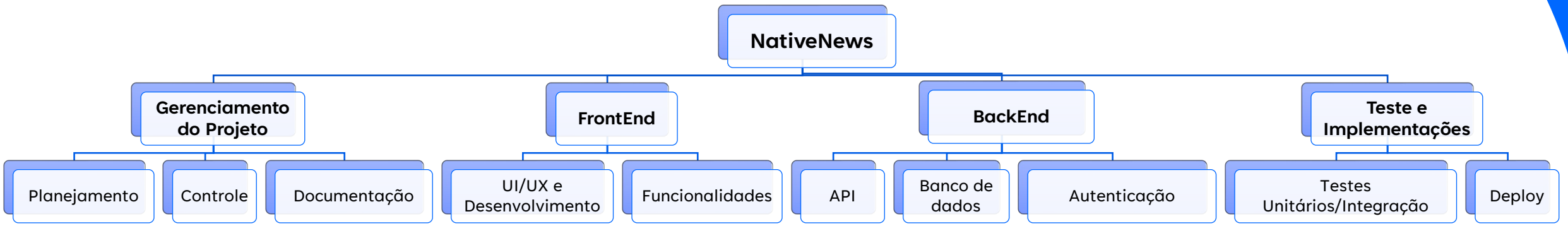
Desenvolvedor

Francisco Macedo

https://github.com/FR-macedo

Sumário

Estrutura Analítica do Projeto



Estrutura Analítica completa do Projeto: Aqui

Atribuição de Tarefas

Scrum Master – Diego Silva

- EAP (1 página, 2 slides)
- Criar cronograma (Gráfico de Gantt)
- Atribuir tarefas e prazos (Google Calendar, Trello, Notion)
- Elaborar planilha de custos (PDF)
- Criar apresentação (PDF: EAP, Cronograma, Tarefas, Protótipos, Demonstração)

Gerente de Configuração – Francisco Macedo

- Criar repositório Git IonicNews (com membros)
- Estruturar repositório: .gitignore, CONTRIBUTING.md, README.md, etc.
- Definir fluxo de trabalho no Git (Branching, Merge)
- Definir ferramentas, bibliotecas e frameworks (PDF)

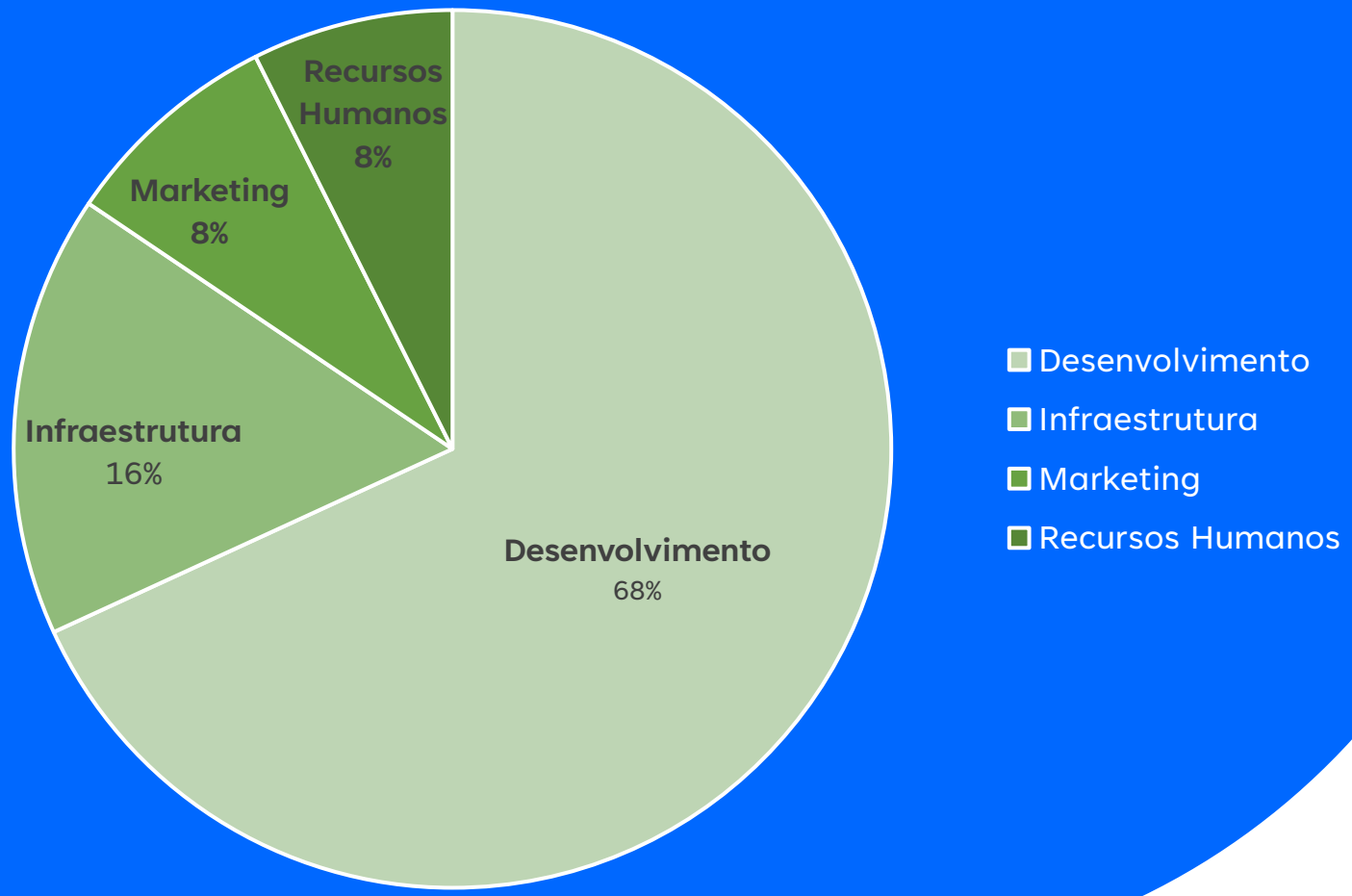
Documentadores – Gabriel

- Elaborar Casos de Uso (PDF)
- Criar protótipos (Canva, Pencil)
- Elaborar Diagramas UML/ER (PDF)

Desenvolvedores – José Roberto e Francisco Macedo

- Codificação da solução (Front-end e Back-end)

Distribuição de Custos do Projeto



Detalhamento Completo dos Custos e Investimentos

Mão de Obra		
Cargo	**Horas**	**Custo**
Scrum Master	60h	R$ 3.000,00
Gerente de Configuração	40h	R$ 2.000,00
Documentadores	100h	R$ 5.000,00
Desenvolvedores	300h	R$ 15.000,00

Infraestrutura e APIs	
Item	**Custo Mensal**
Hospedagem Backend (Firebase/Node.js)	R$ 100,00
NewsAPI	Gratuito
Notificações Push	R$ 50,00
Armazenamento Local	Incluso

Ferramentas e Licenças	
Ferramenta	**Custo**
Ionic Framework	Gratuito
VSCode	Gratuito
WebStorm (opcional)	R$ 250,00/ano
Mermaid	Gratuito
PowerPoint	R$ 359,00

Marketing e Outros	
Marketing e Divulgação	R$ 500,00
Licenciamento	Gratuito

Cronograma

Fase de Planejamento (25/10 - 05/11)				
Atividade	**Data Início**	**Data Fim**	**Duração**	**Fase**
Kickoff do Projeto	25/10	25/10	1 dia	Planejamento
Análise de Requisitos	25/10	27/10	3 dias	Planejamento
Documentação	28/10	29/10	2 dias	Planejamento
Definição Técnica	30/10	31/10	2 dias	Planejamento
Prototipação	02/11	05/11	4 dias	Planejamento

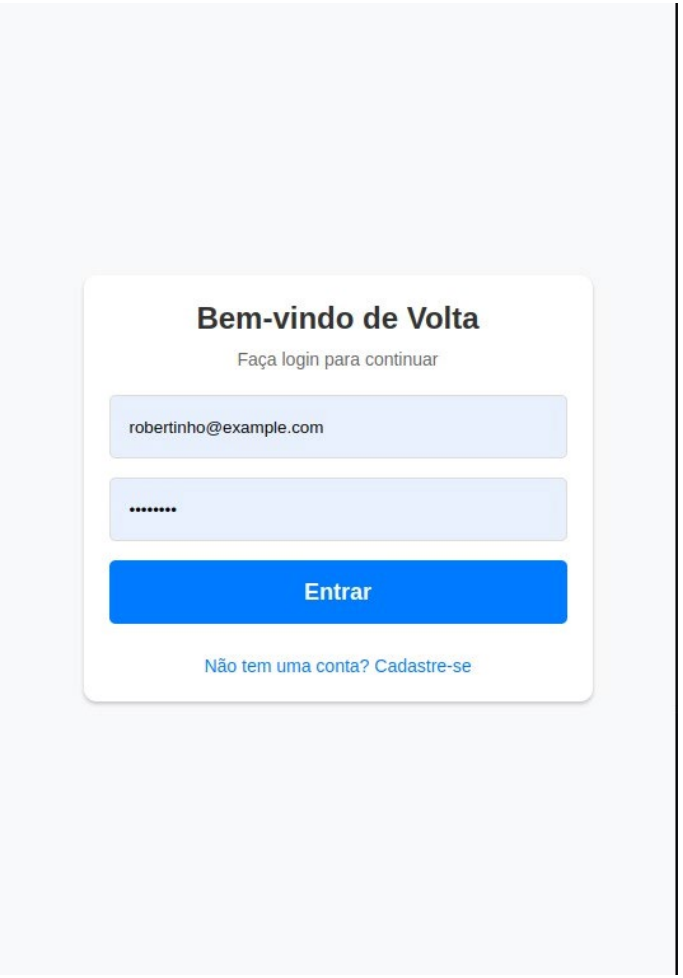
Fase de Design (06/11 - 12/11)				
Atividade	**Data Início**	**Data Fim**	**Duração**	**Fase**
Design System	06/11	07/11	2 dias	Design
Layout e Revisão	08/11	12/11	5 dias	Design

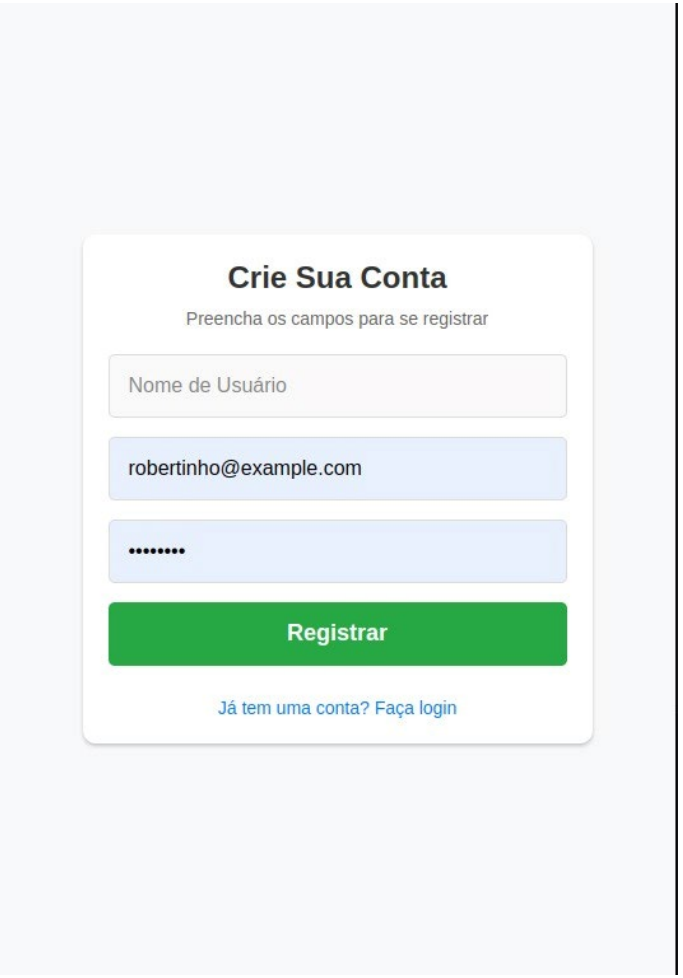
Fase de Desenvolvimento (13/11 - 26/11)				
Atividade	**Data Início**	**Data Fim**	**Duração**	**Fase**
Setup	13/11	14/11	2 dias	Desenvolvimento
Desenvolvimento Principal	15/11	21/11	7 dias	Desenvolvimento
Integração	22/11	26/11	5 dias	Desenvolvimento

Fase de Testes (27/11 - 03/12)				
Atividade	**Data Início**	**Data Fim**	**Duração**	**Fase**
Testes Unitários	27/11	29/11	3 dias	Testes
Testes Integrados	30/11	03/12	4 dias	Testes

Fase de Implantação (04/12 - 06/12)				
Atividade	**Data Início**	**Data Fim**	**Duração**	**Fase**
Deploy	04/12	05/12	2 dias	Implantação
Entrega Final	06/12	06/12	1 dia	Implantação

Demonstração de uso



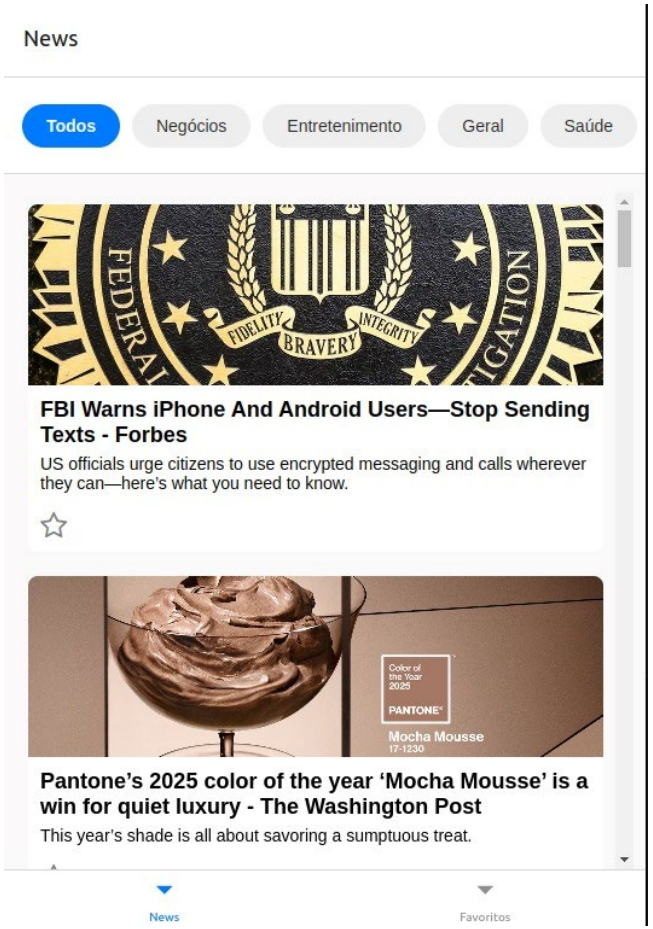


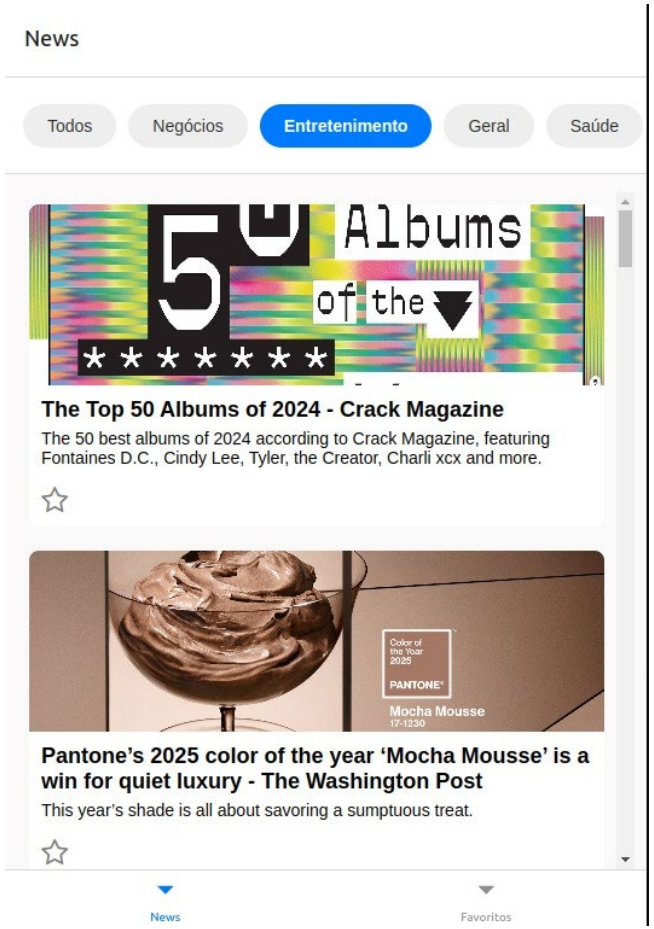
Extratos dos códigos

```
const login = async (req, res) => {
    const { email, password } = req.body;
    if (!email || !password) {
        return res.status(400).json({ error: 'Email e senha são obrigatórios' });
    }
    try {
        // Verificar se o usuário existe
        const [user] = await database_1.default.execute('SELECT * FROM users WHERE email = ?', [email]);
        if (user.length === 0) {
            return res.status(400).json({ error: 'Usuário não encontrado' });
        }
        // Comparar a senha fornecida com a senha criptografada no banco de dados
        const isMatch = await bcryptjs_1.default.compare(password, user[0].password);
        if (!isMatch) {
            return res.status(400).json({ error: 'Senha incorreta' });
        }
        // Criar o token de autenticação
        const token = jsonwebtoken_1.default.sign({ id: user[0].id, email: user[0].email }, process.env.JWT_SECRET, { expiresIn: '1h' });
        res.status(200).json({
            message: 'Login realizado com sucesso',
            token: token,
        });
    }
    catch (error) {
        // Verificar se o erro é uma instância de Error
        if (error instanceof Error) {
            console.error('Erro ao realizar login:', error.message);
            return res.status(500).json({ error: 'Erro ao realizar login', details: error.message });
        }
        else {
            console.error('Erro desconhecido:', error);
            return res.status(500).json({ error: 'Erro desconhecido ao realizar login' });
        }
    }
};
```

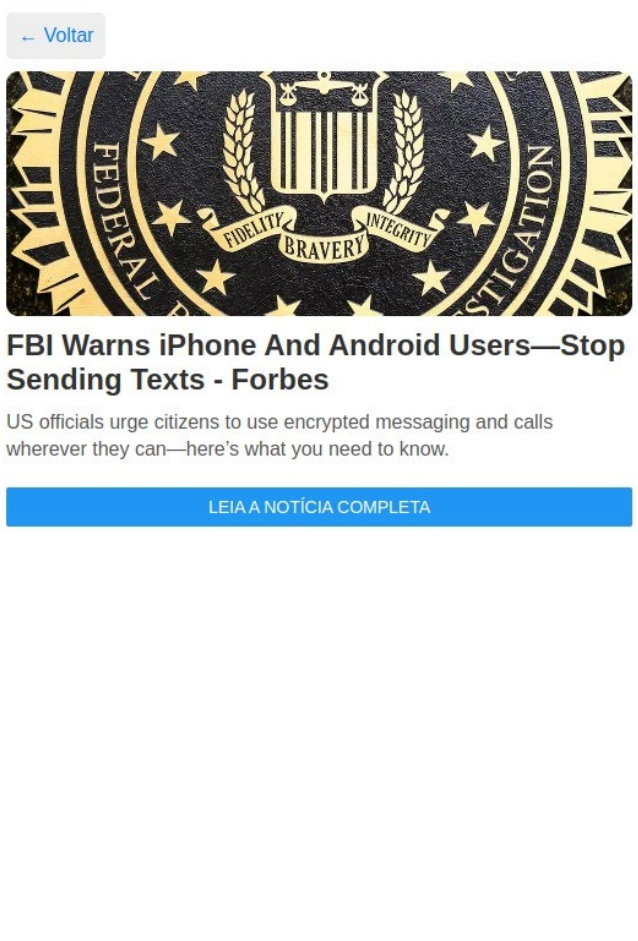
```
const register = async (req, res) => {
    const { name, email, password } = req.body;
    if (!name || !email || !password) {
        return res.status(400).json({ error: 'Todos os campos são obrigatórios' });
    }
    try {
        // Verificar se o usuário já existe
        const [existingUser] = await database_1.default.execute('SELECT * FROM users WHERE email = ?', [email]);
        if (existingUser.length > 0) {
            return res.status(400).json({ error: 'O email já está em uso.' });
        }
        // Criptografar a senha
        const hashedPassword = await bcryptjs_1.default.hash(password, 10);
        // Inserir o usuário
        await database_1.default.execute('INSERT INTO users (name, email, password) VALUES (?, ?, ?)', [name, email, hashedPassword]);
        res.status(201).json({ message: 'Usuário registrado com sucesso' });
    }
    catch (error) {
        console.error('Erro ao registrar usuário:', error);
        res.status(500).json({ error: 'Erro ao registrar usuário' });
    }
};
```

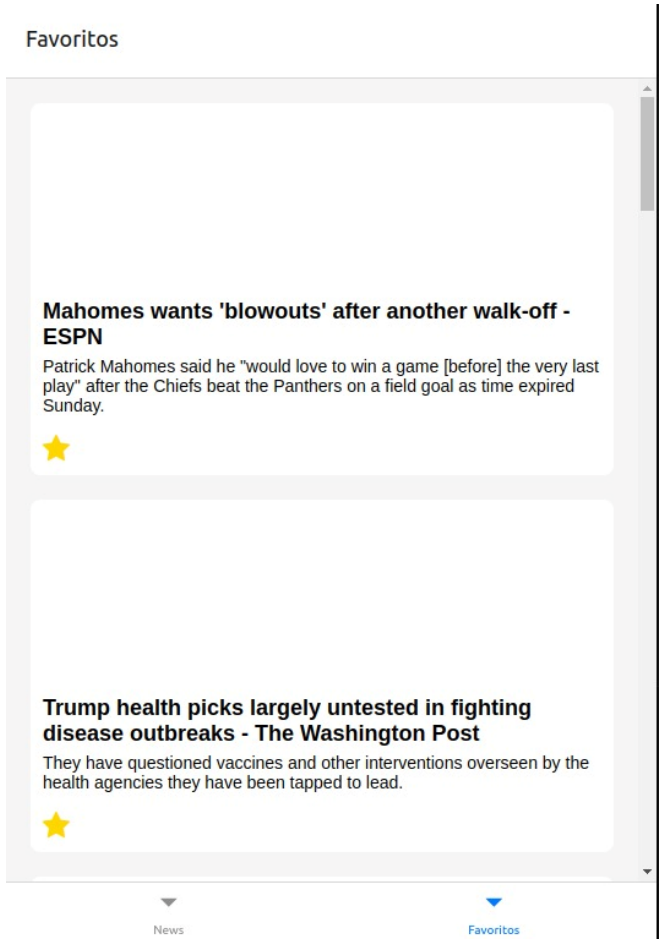
Demonstração de uso





Demonstração de uso





Extratos dos códigos

```
const addFavorite = async (req, res) => {
    const { id: userId } = req.user; // Desestruturando userId de req.user
    const { title, description, imageUrl, newsUrl } = req.body;
    // Verificar se todos os campos necessários estão presentes
    if (!title || !description || !imageUrl || !newsUrl) {
        return res.status(400).json({ error: 'Os campos "title", "description", "imageUrl" e "newsUrl" são obrigatórios.' });
    }
    try {
        // Verificar se o item já foi adicionado aos favoritos
        const [existingFavorite] = await database_1.default.execute('SELECT * FROM favorites WHERE user_id = ? AND newsUrl = ?', [userId, newsUrl]);
        if (existingFavorite.length > 0) {
            return res.status(400).json({ error: 'Esse item já está nos seus favoritos.' });
        }
        // Inserir favorito com as informações detalhadas da noticia
        const result = await database_1.default.execute('INSERT INTO favorites (user_id, title, description, imageUrl, newsUrl) VALUES (?, ?, ?, ?, ?)', [userId, title, description, imageUrl, newsUrl]);
        console.log('Resultado da inserção:', result);
        res.status(201).json({ message: 'Favorito adicionado com sucesso' });
    }
    catch (error) {
        console.error('Erro ao adicionar favorito:', error);
        res.status(500).json({
            error: 'Erro ao adicionar favorito',
            details: error instanceof Error ? error.message : 'Erro desconhecido'
        });
    }
};
```

```
"use strict";
var __importDefault = (this && this.__importDefault) || function (mod) {
    return (mod && mod.__esModule) ? mod : { "default": mod };
};
Object.defineProperty(exports, "__esModule", { value: true });
const express_1 = __importDefault(require("express"));
const favoritosControler_1 = require("../controle/favoritosControler");
const authMiddleware_1 = __importDefault(require("../middleware/authMiddleware"));
const router = express_1.default.Router();
// Rota para adicionar um favorito
router.post('/favorites', authMiddleware_1.default, favoritosControler_1.addFavorite);
// Rota para listar os favoritos do usuário
router.get('/favorites', authMiddleware_1.default, favoritosControler_1.getFavorites);
// Rota para remover um favorito
router.delete('/favorites', authMiddleware_1.default, favoritosControler_1.removeFavorite);
exports.default = router;

```

Lições Aprendidas

Aprendizados em Gestão

- Importância do planejamento inicial
- Comunicação efetiva entre equipe
- Gestão de tempo e prioridades
- Resolução de conflitos
- Adaptação a mudanças de requisitos

Desafios Técnicos

- Integração com APIs externas (NewsAPI)
- Implementação de autenticação segura
- Desenvolvimento de interface responsiva
- Gerenciamento de estado da aplicação
- Trabalho com armazenamento local e cache

Referências

Documentações Técnicas	Ferramentas Utilizadas
Expo Framework: https://docs.expo.dev/	**Microsoft PowerPoint**: Criação de slides e apresentação
React: https://react.dev/learn/	**Visual Studio Code**: IDE de desenvolvimento
TypeScript: https://www.typescriptlang.org/docs	**Git & GitHub**: Controle de versão
NewsAPI: https://newsapi.org/docs	

Agradecimentos

Professor Orientador: Prof. **João Ferreira da Silva Junior** -
- *Disciplina de Mobile*

UNINASSAU e **Programa Embarque Digital** *Por proporcionar experiências únicas de aprendizado e desenvolvimento profissional*

E a toda equipe do projeto

Gratidão a todos que fizeram parte desta jornada de aprendizado e desenvolvimento.