**SERVIÇO NACIONAL DE APRENDIZAGEM COMERCIAL**

**SENAC**

**CURO DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS**

**PROJETO INTEGRADOR: DESENVOLVIMENTO DE SISTEMAS ORIENTADO A DISPOSITIVOS MÓVEIS E BASEADOS NA WEB**

Integrantes do grupo:

Fernando Fraguas Carneiro

Jéssica Silva Azevedo

Kevin de Jesus Souza

Rafael de Morais Lopes

Romualdo Souza de Souza

Victor Hugo Milone

EAD - ENSINO À DISTÂNCIA - 2024

**PROJETO INTEGRADOR: DESENVOLVIMENTO DE SISTEMAS ORIENTADO A DISPOSITIVOS MÓVEIS E BASEADOS NA WEB**

Jean Carlo Wagner

**TRABALHO PARA APROVAÇÃO EM DISCIPLINA**

EAD - ENSINO À DISTÂNCIA - 2024

**Resumo**

O objetivo do trabalho é desenvolver um aplicativo de saúde e fitness que auxilie os usuários a monitorarem e acompanhar suas atividades físicas de maneira simples e acessível, promovendo um estilo de vida ativo e saudável. A metodologia utilizada inclui a integração de diferentes modalidades de exercício e a personalização da experiência do usuário, adaptando-se às suas preferências e necessidades específicas. Os resultados alcançados demonstram que a plataforma pode facilitar o registro e monitoramento das atividades físicas, contribuindo para o aumento da motivação e comprometimento com a prática regular de exercícios. Conclui-se que a combinação entre tecnologia e saúde oferece uma solução eficaz para superar barreiras enfrentadas por muitas pessoas que precisam de ajuda para manter hábitos mais saudáveis em suas rotinas diárias.

**Palavras-chave:** saúde, aplicativo, personas, exercícios físicos.

**Sumário**

[1. Banco de Dados usado e configurações 5](#_Toc183439434)

[1.1. Configuração do Banco de Dados 5](#_Toc183439435)

[1.2. Criação de Tabelas 5](#_Toc183439436)

[1.3. Explicação das Tabelas 6](#_Toc183439437)

[1.4. Scrits de Operações CRUD 6](#_Toc183439438)

[1.5. Configuração do Banco de Dados 7](#_Toc183439439)

[2. Qual arquitetura de software foi usada 7](#_Toc183439440)

[2.1. Arquitetura Cliente-Servidor Local (Offline) 8](#_Toc183439441)

[3. Quais APIs foram usadas 8](#_Toc183439442)

[3.1. Apache Cordova API 8](#_Toc183439443)

[3.2. SQLite Plugin para Cordova 8](#_Toc183439444)

[3.3. API de Navegação HTML (DOM) 9](#_Toc183439445)

[4. Resumo das APIs utilizadas: 9](#_Toc183439446)

[4.1. Cordova API 9](#_Toc183439447)

[4.2. SQLite Plugin 10](#_Toc183439448)

[4.3. API de Navegação HTML (DOM) 10](#_Toc183439449)

[5. Link do GitHub 10](#_Toc183439450)

[**Conclusão** 11](#_Toc183439451)

[Referências 12](#_Toc183439452)

# Banco de Dados usado e configurações

O banco de dados usado no aplicativo é o SQLite, que é um banco de dados relacional leve e local, amplamente utilizado em aplicativos móveis e outros ambientes que necessitam de armazenamento persistente sem a necessidade de um servidor remoto.

Nesse código é utilizado o plugin cordova-sqlite-storage, que permite o uso do SQLite diretamente em dispositivos móveis com Apache Cordova. Serão detalhadas a configuração e os scripts que serão utilizados para configurar o banco de dados e as tabelas.

## Configuração do Banco de Dados

O banco de dados é criado e manipulado usando o window.sqlitePlugin, que é o objeto fornecido pelo plugin SQLite do Cordova.

No presente código, o banco de dados é aberto com o seguinte comando:

const db = window.sqlitePlugin.openDatabase({ name: 'fitness.db', location: 'default' });

Aqui:

* **fitness.db**: É o nome do banco de dados que será utilizado. Se o banco de dados não existir, ele será criado.
* **location: 'default'**: Define o local onde o banco de dados será armazenado. No caso, o banco será armazenado no diretório padrão do dispositivo.

## Criação de Tabelas

Dentro do banco de dados tem três tabelas principais: **usuarios**, **objetivos** e **anotacoes**. A criação dessas tabelas é feita usando a transação SQL que roda quando o evento deviceready é disparado, garantindo que o banco de dados seja manipulado após a inicialização do Cordova.

***Script para Criação de Tabelas:***

No arquivo app.js, a criação das tabelas é realizada dentro de uma transação:

db.transaction(tx => {

// Tabela de usuários

tx.executeSql('CREATE TABLE IF NOT EXISTS usuarios (id INTEGER PRIMARY KEY, nome TEXT, email TEXT, senha TEXT)');

// Tabela de objetivos

tx.executeSql('CREATE TABLE IF NOT EXISTS objetivos (id INTEGER PRIMARY KEY, userId INTEGER, objetivo TEXT)');

// Tabela de anotações

tx.executeSql('CREATE TABLE IF NOT EXISTS anotacoes (id INTEGER PRIMARY KEY, userId INTEGER, texto TEXT)');

});

## Explicação das Tabelas

**1. Tabela usuarios:**

1. a **id:** Chave primária que identifica de forma única cada usuário.

b. **nome**: Armazena o nome do usuário.

**c. email**: Armazena o email do usuário.

d. **senha**: Armazena a senha do usuário (deveria ser tratada adequadamente em termos de segurança, como com hashing).

**2. Tabela objetivos:**

1. a. **id:** Chave primária.

b. **userId**: Chvave estrangeira que relaciona o objetivo a um usuário específico (referência ao id da tabela usuários).

c. **objetivo**: Armazena o o objetivo do usuário (como “Ganhar Massa Muscular”, “Emagrecer”, etc.).

**3. Tabela anotações:**

1. a. **id:** Chave primária.

b. **userId**: Chave estrangeira que relaciona a anotação a um usuário específico (referência ao id da tabela usuarios).

c. **texto**: Armazena o conteúdo da anotação.

## Scrits de Operações CRUD

* 1. Além da criação das tabelas, estamos utilizando o SQLite para realizar operações de **CRUD** (Create, Read, Update, Delete) no banco de dados. Aqui estão alguns exemplos de como são manipulados os dados:
  2. ***Criando um novo usuário:***
  3. tx.executeSql('INSERT INTO usuarios (nome, email, senha) VALUES (?, ?, ?)', [nome, email, senha], () => {
  4. alert('Cadastro realizado com sucesso!');
  5. }, () => {
  6. alert('Erro ao cadastrar. Tente novamente.');
  7. });
  8. ***Inserindo um objetivo:***
  9. tx.executeSql('INSERT INTO objetivos (userId, objetivo) VALUES (?, ?)', [currentUser.id, objetivo], () => {
  10. listarObjetivoAtual();
  11. });
  12. ***Listando anotações:***
  13. tx.executeSql('SELECT \* FROM anotacoes WHERE userId = ?', [currentUser.id], (tx, result) => {
  14. for (let i = 0; i < result.rows.length; i++) {
  15. const item = result.rows.item(i);
  16. // Processar cada anotação
  17. }
  18. });

## Configuração do Banco de Dados

O banco de dados **fitness.db** é criado no diretório de armazenamento padrão do dispositivo. No caso do plugin SQLite do Cordova, o banco é armazenado de maneira persistente e não depende de uma conexão de rede, funcionando de forma totalmente offline.

**Resumo:**

* **Banco de Dados**: **SQLite**, gerido pelo plugin cordova-sqlite-storage.
* Tabelas Criadas: usuarios: Armazena dados do usuário (id, nome, email, senha).
  1. objetivos: Relaciona objetivos com usuários (id, userId, objetivo).
  2. anotacoes: Armazena anotações do usuário (id, userId, texto).

Este banco de dados local permite que o aplicativo funcione sem uma conexão com a internet, armazenando dados no dispositivo de forma eficiente e persistente.

# Qual arquitetura de software foi usada

## Arquitetura Cliente-Servidor Local (Offline)

Cliente: O cliente é o próprio aplicativo, que está rodando no dispositivo móvel, usando o Apache Cordova para criar uma interface com HTML, CSS e JavaScript.

O cliente é responsável pela interação com o usuário, incluindo formulários de login, cadastro, objetivo e anotações.

Servidor Local (Banco de Dados Local): Em vez de fazer chamadas a um servidor remoto, o aplicativo armazena os dados no dispositivo, utilizando o SQLite, um banco de dados local. O banco de dados armazena informações de usuários, objetivos e anotações, permitindo que o aplicativo funcione offline.

Arquitetura Monolítica

A arquitetura pode ser considerada monolítica, uma vez que toda a lógica e a interface do usuário estão contidas no mesmo pacote do aplicativo.

Não há divisão clara de camadas entre frontend (interface) e backend (servidor remoto), uma vez que o banco de dados e a lógica de negócios estão todos no dispositivo.

# Quais APIs foram usadas

## Apache Cordova API

O Apache Cordova fornece uma plataforma de desenvolvimento para criar aplicativos móveis utilizando tecnologias web. Algumas APIs específicas que estamos utilizando são:

* **cordova.js**: O arquivo cordova.js é incluído no HTML para integrar o Apache Cordova ao seu código JavaScript. Ele fornece acesso a recursos nativos dos dispositivos móveis (como o armazenamento local e APIs de dispositivo).
* **deviceready**: Você usa o evento deviceready para garantir que o aplicativo só interaja com os recursos nativos do dispositivo após a inicialização completa do Cordova. Isso é feito com o código:

document.addEventListener('deviceready', () => {

// código que depende de APIs nativas

});

## SQLite Plugin para Cordova

**cordova-sqlite-storage**: O principal plugin utilizado para interagir com o banco de dados SQLite. Este plugin fornece uma interface JavaScript para acessar e manipular bancos de dados SQLite diretamente no dispositivo. Ele é usado no seu código para criar tabelas, inserir, atualizar e consultar dados. Usa-se os comandos como tx.executeSql() para executar comandos SQL no banco de dados.

Exemplos de operações com SQLite:

db.transaction(tx => {

tx.executeSql('CREATE TABLE IF NOT EXISTS usuarios (id INTEGER PRIMARY KEY, nome TEXT, email TEXT, senha TEXT)');

tx.executeSql('INSERT INTO usuarios (nome, email, senha) VALUES (?, ?, ?)', [nome, email, senha]);

});

## API de Navegação HTML (DOM)

* Embora não seja uma API externa, o aplicativo também depende fortemente das APIs do **DOM (Document Object Model)** para gerenciar a navegação entre telas.
* O código JavaScript manipula a visibilidade das diferentes seções (divs) do aplicativo utilizando o document.querySelectorAll() e document.getElementById(), além de métodos como style.display para mostrar ou ocultar as telas:

const showScreen = (screenId) => {

document.querySelectorAll('#app > div').forEach(screen => {

screen.style.display = 'none';

});

document.getElementById(screenId).style.display = 'block';

};

# Resumo das APIs utilizadas:

## Cordova API

a. cordova.js: Inicializa e fornece a integração com o Cordova e APIs nativas.

b. deviceready: Evento para garantir que o Cordova foi carregado antes de interagir com o dispositivo.

## SQLite Plugin

a. window.sqlitePlugin.openDatabase(): Abre o banco de dados SQLite.

b. tx.executeSql(): Executa comandos SQL (para criar tabelas, inserir e consultar dados).

## API de Navegação HTML (DOM)

a. Manipulação do DOM (document.querySelectorAll(), document.getElementById()) para controlar a navegação entre as telas do aplicativo.

Essas APIs permitem que o aplicativo funcione corretamente, armazenando dados localmente (SQLite) e interagindo com a interface do usuário (DOM e Cordova).

# Link do GitHub

<https://github.com/VictorMilone/2-ENTREGA-PROJETO-INTEGRADOR/tree/main>

# **Conclusão**

Em conclusão, percebe-se a crescente preocupação com a saúde e o bem-estar de um público que tem impulsionado a busca por um estilo de vida mais saudável. No entanto, há desafios para as pessoas manterem uma rotina consistente de atividades físicas, devido a diversos motivos. Com a evolução tecnológica, surgem novas oportunidades de superar essas barreiras.

Nesse sentido, visando esse público que se preocupa com o bem-estar, o desenvolvimento do aplicativo de saúde e fitness preenche essa lacuna, proporcionando uma plataforma simples e acessível para monitorar e acompanhar as atividades físicas, sendo uma resposta direta à demanda crescente por soluções inteligentes que ofereçam mais controle e organização da rotina. Assim sendo, o aplicativo atende os seus objetivos ao integrar diferentes modalidades de exercício e oferecer uma experiência personalizada, levando motivação e comprometimento dos usuários, ajudando-os a alcançar a alcançar suas metas de saúde e a manter hábitos mais saudáveis e organizados.

# Referências

Logo do software não tem direitos autorais e foi desenhada por Rafiico Creative Studio Co., Ltd. https://www.iconfinder.com/icons/7195758/safe\_excercise\_healthcare\_gym\_fitness\_weight\_training\_trainer\_icon