Prog Disp Móveis

Competências:

- Desenvolver aplicativo para dispositivos móveis.

Indicadores de Competências:

- Desenvolve aplicativos para o contexto de dispositivos móveis;
- Utiliza APIs para manipulação dos componentes do dispositivo móvel;
- Integra aplicativo móvel com serviços web;
- Publica aplicativos móveis em serviço de distribuição digital.

Bases Tecnológicas, científicas e instrumentais (conteúdos):

- Persistência de dados no dispositivo móvel;

Situação de Aprendizagem:

- Aplicações para Dispositivos Móveis

Senac

SQLite - www.sqlite.org

SQLite is a C-language library that implements a small, fast, self-contained, high-reliability, full-featured, SQL database engine.

Prog Disp Móveis

- Linguagem padrão para consulta a bancos de dados
- Construções básicas
  - SELECT Campo1, Campo2, ... FROM Tabela1, Tabela2 WHERE *<condição>* ORDER BY CampoX;
  - INSERT INTO Tabela(Campo1, Campo2, ...) VALUES (Valor1, Valor2, ...);
  - DELETE FROM Tabela WHERE <condição>;
  - UPDATE Tabela SET Campo1 = <expressao1>, Campo2 = <expressao2> WHERE <condição>;
- Criação e exclusão de tabelas []=opcional
  - CREATE TABLE Tabela (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    Campo1 Tipo1 [[UNIQUE] NOT NULL],
    Campo2 Tipo2 [NOT] NULL,
    ...);
  - DROP TABLE IF EXISTS Tabela

SQLite

`expo-sqlite` dá ao seu aplicativo acesso a um banco de dados que pode ser consultado por meio de uma API semelhante a WebSQL . O banco de dados é mantido durante as reinicializações do seu aplicativo.

Está disponível um exemplo de aplicativo de lista de tarefas que usa este módulo para armazenamento.

Compatibilidade de plataforma

Dispositivo Android	Android Emulator	Dispositivo iOS	simulador iOS	Rede
✅	✅	✅	✅	❌

Prog Disp Móveis

Expo-sqlite

Primeiro, você precisa instalá-los em seu projeto:

```
expo install expo-sqlite
```

Dentro do projeto

```
import * as SQLite from 'expo-sqlite';
```

`SQLite.openDatabase(name, version, description, size)`

Abra um banco de dados, criando-o se ele não existir, e retorne um `Database` objeto. No disco, o banco de dados será criado sob o aplicativo de diretório de documentos , ou seja `${FileSystem.documentDirectory}/SQLite/${name}` .

Argumentos

- **name (*string*)** - Nome do arquivo de banco de dados a ser aberto.

Os argumentos `version` , `description` e `size` são ignorados, mas são aceitos pela função para compatibilidade com a especificação WebSQL.

Devoluções

Retorna um `Database` objeto, descrito a seguir.

- `db.transaction(callback, error, success)` Execute uma transação de banco de dados.

  Parâmetros

  - **callback (*function*)** - Uma função que representa a transação a ser executada. Toma um `Transaction` (veja abaixo) como seu único parâmetro, no qual pode adicionar instruções SQL para executar.
  - **erro (*função*)** - Chamado se ocorrer um erro no processamento desta transação. Obtém um único parâmetro que descreve o erro.
  - **sucesso (*função*)** - Chamado quando a execução da transação no banco de dados é concluída.

- `tx.executeSql(sqlStatement, arguments, success, error)` Enfileire uma instrução SQL para executar na transação. Os autores são fortemente recomendados a usar o `?` recurso de espaço reservado do método para evitar ataques de injeção de SQL e nunca construir instruções SQL dinamicamente.
  Parâmetros
  - **sqlStatement (*string*)** - Uma string contendo uma consulta ao banco de dados para executar expressa como SQL. A string pode conter `?` marcadores de posição, com valores a serem substituídos listados no `arguments` parâmetro.
  - **argumentos (*matriz*)** - uma matriz de valores (números ou strings) para substituir `?` marcadores de posição na instrução SQL.
  - **sucesso (*função*)** - Chamado quando a consulta é concluída com sucesso durante a transação. Aceita dois parâmetros: a própria transação e um `ResultSet` objeto (veja abaixo) com os resultados da consulta.
  - **erro (*função*)** - Chamado se ocorrer um erro ao executar essa consulta específica na transação. Aceita dois parâmetros: a própria transação e o objeto de erro.

Prog Disp Móveis

`ResultSet` os objetos são retornados por meio do segundo parâmetro do `success` retorno de chamada para o `tx.executeSql()` método em a `Transaction` (veja acima). Eles têm o seguinte formato:

```
{
  insertId,
  rowsAffected,
  rows: {
    length,
    item(),
    _array,
  },
}
```

- **insertId (*número*)** - O ID da linha da linha que a instrução SQL inseriu no banco de dados, se uma linha foi inserida.
- **rowsAffected (*number*)** - O número de linhas que foram alteradas pela instrução SQL.
- **rows.length (*number*)** - O número de linhas retornadas pela consulta.
- **row.item (*function*)** - `rows.item(index)` retorna a linha com o dado `index`. Se não houver tal linha, retorna `null`.
- **linhas. *array (_ number)*** - O array real de linhas retornado pela consulta. Pode ser usado diretamente em vez de passar as linhas `rows.item()`.

Prog Disp Móveis

```
import React from 'react';
import { View, Text, TouchableOpacity, ScrollView, StyleSheet,
TextInput, Button } from 'react-native';

import * as SQLite from 'expo-sqlite'
const db = SQLite.openDatabase('db.testDb') // returns Database
object
```

Importação da biblioteca recai

Prog Disp Móveis

```
import React from 'react';
import { View, Text, TouchableOpacity, ScrollView, StyleSheet,
TextInput, Button } from 'react-native';

import * as SQLite from 'expo-sqlite'
const db = SQLite.openDatabase('db.testD  ') // returns Database
object
```

Importação dos componentes utilizados no desenvolvimento

Prog Disp Móveis

```
import React from 'react';
import { View, Text, TouchableOpacity, ScrollView, StyleSheet,
TextInput, Button } from 'react-native';

import * as SQLite from 'expo-sqlite'
const db = SQLite.openDatabase('db.testDb') // returns Database
object
```

Importação da biblioteca do SQLite

Prog Disp Móveis

```
import React from 'react';
import { View, Text, TouchableOpacity, ScrollView, StyleSheet,
TextInput, Button } from 'react-native';

import * as SQLite from 'expo-sqlite'
const db = SQLite.openDatabase('db.testDb')
```

Definição da base a ser utilizada.

Prog Disp Móveis

```
class App extends React.Component {
  state = {
    nome: '',
    telefone: '',
  }
  setNome = (n) => {
    this.setState({nome: n})
  }
  setTelefone = (t) => {
    this.setState({telefone: t})
  }

  constructor(props) {
    super(props)
   db.transaction(tx => {
      tx.executeSql(
        'CREATE TABLE IF NOT EXISTS agenda (id INTEGER PRIMARY KEY AUTOINCREMENT,
nome TEXT, telefone TEXT)'
      )
    })
```

Criação da base

Prog Disp Móveis

```
class App extends React.Component {
  state = {
    nome: '',
    telefone: '',
  }
  setNome = (n) => {
    this.setState({nome: n})
  }
  setTelefone = (t) => {
    this.setState({telefone: t})
  }

  constructor(props) {
    super(props)
   db.transaction(tx => {
      tx.executeSql(
        'CREATE TABLE IF NOT EXISTS agenda (id INTEGER PRIMARY KEY AUTOINCREMENT,
nome TEXT, telefone TEXT)'
      )
    })
```

Campos de entrada no aplicativo

Prog Disp Móveis

```
fetchData = () => {
    db.transaction(tx => {
     tx.executeSql('SELECT * FROM agenda', null,
       (txObj, { rows: { _array } }) => this.setState({ data: _array })
       )
    })
  }
```

Recuperação de todos os campos como array

Prog Disp Móveis

```
  newItem = (nome, telefone) => {
    console.log('Inserido: ', nome);
    db.transaction(tx => {
      tx.executeSql('INSERT INTO agenda (nome, telefone) values (?, ?)', [nome,
telefone],
        (txObj, resultSet) => this.setState({ data: this.state.data.concat(
            { id: resultSet.insertId, nome: nome, telefone: telefone }) }),
        (txObj, error) => console.log('Error', error))
    })
  }
```

Inserção de um novo item

Prog Disp Móveis

```
delete = (id) => {
  db.transaction(tx => {
    tx.executeSql('SELECT nome FROM agenda WHERE id = ?', [id],
    (txObj, resultSet) => {
        let newList = this.state.data.filter(data => {
          if (data.id === id) console.log('Removido: ', data.nome)
        })
    })
    tx.executeSql('DELETE FROM agenda WHERE id = ? ', [id],
      (txObj, resultSet) => {
        if (resultSet.rowsAffected > 0) {
          let newList = this.state.data.filter(data => {
            if (data.id === id)
              return false
            else
              return true
          })
          this.setState({ data: newList })
        }
      })
  })
}
```

Inserção do item escolhido

Prog Disp Móveis

```
<TextInput
  style={styles.input}
  placeholder="Entre com o nome"
  onChangeText={this.setNome}/>

<TextInput
  style={styles.input}
  placeholder="Entre com o telefone
  onChangeText={this.setTelefone}/>

<TouchableOpacity
  style={styles.botao}
  onPress={()=>this.newItem(this.stat ome, this.state.telefone)}>
      <Text style={styles.rotuloBotao} lvar</Text>
</TouchableOpacity>
```

TextInput de entrada do nome

Prog Disp Móveis

```
<TextInput
  style={styles.input}
  placeholder="Entre com o nome"
  onChangeText={this.setNome}/>

<TextInput
  style={styles.input}
  placeholder="Entre com o telefone"
  onChangeText={this.setTelefone}/>

<TouchableOpacity
  style={styles.botao}
  onPress={()=>this.newItem(this.state.nome, this.state.telefone)}>
      <Text style={styles.rotuloBotao}>Salvar</Text>
</TouchableOpacity>
```

TextInput de entrada do telefone

Prog Disp Móveis

```
<ScrollView >
{
    this.state.data && this.state.data.map(data =>
    (
        <View key={data.id} style={styles.item}>
        <TouchableOpacity
            onPress={() => this.delete(data.id)}>
            <Text>Nome:{data.nome} - Telefone:{data.telefone}</Text>
        </TouchableOpacity>
        </View>
    )
)}
</ScrollView>
```

ScrollView para apresentação dos dados