Programação Móvel

Bancos de Dados Móveis

Prof. Dr. Diego R. Antunes
drantunes@utfpr.edu.br

Departamento de Computação
Universidade Tecnológica Federal do Paraná

Anteriormente

Componentes Visuais

- *Componentes de Interação / Interface;*
- *Componentes de Navegação;*
- *Componentes de Feedback;*

Problema

Todas as aplicações móveis desenvolvidas até o momento são voláteis, ou seja, perdem os dados quando fechamos a aplicação.

Problema

Para evitar este problema, precisamos persistir os dados após o aplicativo fechar. Ao abrir o aplicativo novamente, podemos acessar os dados salvos.

Persistência

Tipos de Persistência

- *Local*
- *Externa*

Persistência Local

Podemos utilizar o sistema de arquivos do próprio dispositivo para armazenar dados. Para gerenciar a memória interna, geralmente utilizamos algum sistema de banco de dados móvel.

Persistência Externa

Outra maneira é utilizar um banco de dados remoto, utilizando a API de um Serviço Web.

Persistência Externa

Por meio de chamadas remotas via HTTP ao Serviço Web, seu aplicativo poderá realizar todas as operações necessárias sobre os dados.

Persistência Externa

Além do uso de Serviços Web, existem alguns bancos de dados que funcionam totalmente via Internet, por exemplo, o Firebase.

Persistência Externa

Neste caso, um aplicativo pode conectar diretamente ao banco de dados, que está online. A vantagem é a simplicidade e não ser necessário a construção de um serviço web.

Banco de Dados

Persistência Local

www.sqlite.org

Persistência Externa

firebase.google.com

Como Selecionar

- *Suporte para sua plataforma de desenvolvimento;*
- *Segurança e Autenticação;*
- *Flexibilidade no Modelo de Dados (Tipos, Estrutura, etc);*
- *Sincronização (e.g. Bancos de Dados em Tempo Real);*

Outros Bancos de Dados

Database	Type of data stored	License	Supported platforms
BerkeleyDB	relational, objects, key-value pairs, documents	AGPL 3.0	Android, iOS
Couchbase Lite	documents	Apache 2.0	Android, iOS
LevelDB	key-value pairs	New BSD	Android, iOS
SQLite	relational	Public Domain	Android, iOS, Windows Phone, Blackberry
UnQLite	key-value pairs, documents	BSD 2-Clause	Android, iOS, Windows Phone

https://www.developereconomics.com/five-popular-databases-for-mobile

SQLite

SQLite

É uma biblioteca de banco de dados auto-contido, sem necessidade de servidor e configuração e que fornece um mecanismo transacional de SQL.

Funcionalidades

- *Transações consistentes e duráveis;*
- *Configuração Zero – sem setup ou administração;*
- *Compatibilidade com SQL;*
- *Armazenado em um único arquivo multi-plataforma;*

SQLite

É gratuito para uso pessoal ou comercial. Além disso, SQLite é o banco de dados móvel mais utilizado para aplicações móveis. Empresas como Google, Facebook, Dropbox, Skype, entre outros usam SQLite.

http://www.sqlite.org/famous.html

SQLite

A grande vantagem do SQLite é não ser necessário manter um servidor somente para o banco de dados, pois seu mecanismo opera diretamente em um arquivo no disco.

SQLite

Assim como qualquer SGBD, o SQLite fornece um mecanismo para transações SQL (select, insert, update, índices, triggers, entre outros).

SQLite

O projeto do SQLite executa um pacote de milhares de testes para garantir um tratamento correto de SQL, bem como tratar problemas de memória e I/O no disco.

Situações de Uso

- *Aplicativos com Dados Embarcados;*
- *Internet das Coisas (IoT);*
- *Cache local de dados para SGBD Remoto;*
- *Dados Temporários (e.g. teste de aplicativo pelo usuário)*

Tipos de Dados

- *NULL*
- *INTEGER*
- *REAL*
- *TEXT*
- *BLOB*

SQLite + Ionic

SQLite no Ionic

O Ionic Native fornece suporte ao SQLite por meio de um plugin do Cordova chamado cordova-sqlite-storage.

http://ionicframework.com/docs/v2/native/sqlite/

SQLite no Ionic

Como o SQLite é um plugin do Ionic Native, isso quer dizer que só pode ser utilizado no dispositivo físico ou em um emulador Android ou iOS.

SQLite no Ionic

A API do SQLite é muito simples e similar à API do WebSQL e IndexedDB do HTML5, utilizados nos navegadores modernos para cache e armazenamento local.

Ionic SQLStorage

Para auxiliar no desenvolvimento de aplicativos que usam o SQLite, resolvendo o problema da necessidade de uso do dispositivo físico ou emulador, o Ionic criou o SQLStorage.

Ionic SQLStorage

O SQLStorage consiste de um adaptador (wrapper) que usa o SQLite quando executado nativamente (dispositivo ou emulador) ou usa do WebSQL do Navegador quando executado pelo ionic serve no browser.

Ionic SQLStorage

Além disso, o SQLStorage fornece uma API ainda mais simples para manipulação de SQL. Todas as SQL são executadas sobre o método query.

http://ionicframework.com/docs/v2/api/platform/storage/SqlStorage/

Ionic SQLStorage

Adicionamente, este módulo também fornece funções para armazenamento chave-valor, para simplesmente utilizar set/get em valores simples (e.g. um token).

Como utilizar

Primeiro, precisamos adicionar o plugin do SQLite ao projeto Ionic. Para isso, na pasta do projeto executamos o comando ionic plugin add ***cordova-sqlite-storage****.*

```
→  aula_15_09_sqlite ionic plugin add cordova-sqlite-storage
Fetching plugin "cordova-sqlite-storage" via npm


Installing "cordova-sqlite-storage" for android


installing external dependencies via npm

npm install of external dependencies ok
```

Então, na página ou componente que você precisar acessar o banco de dados, você precisará importar dois módulos.

```
import { Storage, SqlStorage } from 'ionic-angular';

@Component({
    templateUrl: 'build/pages/home/home.html'
})
export class HomePage {
    protected db: any;
    constructor() {
        this.db = new Storage(SqlStorage, {
            name: "meu_banco"
        });
        let tabela = "tarefas(id INT PRIMARY KEY AUTOINCREMENT, titulo TEXT, status INT)";
        this.db.query(`CREATE TABLE IF NOT EXISTS ${tabela}`).then((data) => {
            console.log("Database Criada" + data);
        });
    }
```

Depois você deve criar uma instância do Storage no construtor. É aqui que você pode informar o nome do Banco

```
import { Storage, SqlStorage } from 'ionic-angular';

@Component({
    templateUrl: 'build/pages/home/home.html'
})
export class HomePage {
    protected db: any;
    constructor() {
        this.db = new Storage(SqlStorage, {
            name: "meu_banco"
        });
        let tabela = "tarefas(id INT PRIMARY KEY AUTOINCREMENT, titulo TEXT, status INT)";
        this.db.query(`CREATE TABLE IF NOT EXISTS ${tabela}`).then((data) => {
            console.log("Database Criada" + data);
        });
    }
```

Nota: Se você utilizar o banco em vários locais você poderia criar uma classe específica de banco de dados, para que não haja a necessidade de criar uma instância a todo momento.

Em seguida você precisa criar a tabela, se ela não existe, de acordo com um esquema de dados.

```
import { Storage, SqlStorage } from 'ionic-angular';

@Component({
    templateUrl: 'build/pages/home/home.html'
})
export class HomePage {
    protected db: any;
    constructor() {
        this.db = new Storage(SqlStorage, {
            name: "meu_banco"
        });
        let tabela = "tarefas(id INT PRIMARY KEY AUTOINCREMENT, titulo TEXT, status INT)";
        this.db.query(`CREATE TABLE IF NOT EXISTS ${tabela}`).then((data) => {
            console.log("Database Criada" + data);
        });
    }
```

O esquema é definido da seguinte maneira:
nome_da_tabela(atributos) conforme mostrado na imagem:

```
import { Storage, SqlStorage } from 'ionic-angular';

@Component({
    templateUrl: 'build/pages/home/home.html'
})
export class HomePage {
    protected db: any;
    constructor() {
        this.db = new Storage(SqlStorage, {
            name: "meu_banco"
        });
        let tabela = "tarefas(id INT PRIMARY KEY AUTOINCREMENT, titulo TEXT, status INT)";
        this.db.query(`CREATE TABLE IF NOT EXISTS ${tabela}`).then((data) => {
            console.log("Database Criada" + data);
        });
    }
```

Por comodidade, definimos parte da string na variável tabela e adicionamos ela no comando query.

```
import { Storage, SqlStorage } from 'ionic-angular';

@Component({
    templateUrl: 'build/pages/home/home.html'
})
export class HomePage {
    protected db: any;
    constructor() {
        this.db = new Storage(SqlStorage, {
            name: "meu_banco"
        });
        let tabela = "tarefas(id INT PRIMARY KEY AUTOINCREMENT, titulo TEXT, status INT)";
        this.db.query(`CREATE TABLE IF NOT EXISTS ${tabela}`).then((data) => {
            console.log("Database Criada" + data);
        });
    }
```

Após a inicialização do Storage e a criação da tabela, se necessário, você pode criar métodos para manipular a tabela do banco de dados. Ou seja, para as operações de insert, delete, select, entre outros.

Por exemplo, temos a função salvar definida em nossa classe (ts). Este método recebe um parâmetro e então executa o INSERT INTO na tabela tarefas.

```
23      salvar(titulo) {
24        this.db.query("INSERT INTO tarefas(titulo, status) VALUES (?,?)", [titulo, 0]).then(
25            (resposta) => { console.log("ID: " + resposta.res.insertId) }
26        );
27      }
```

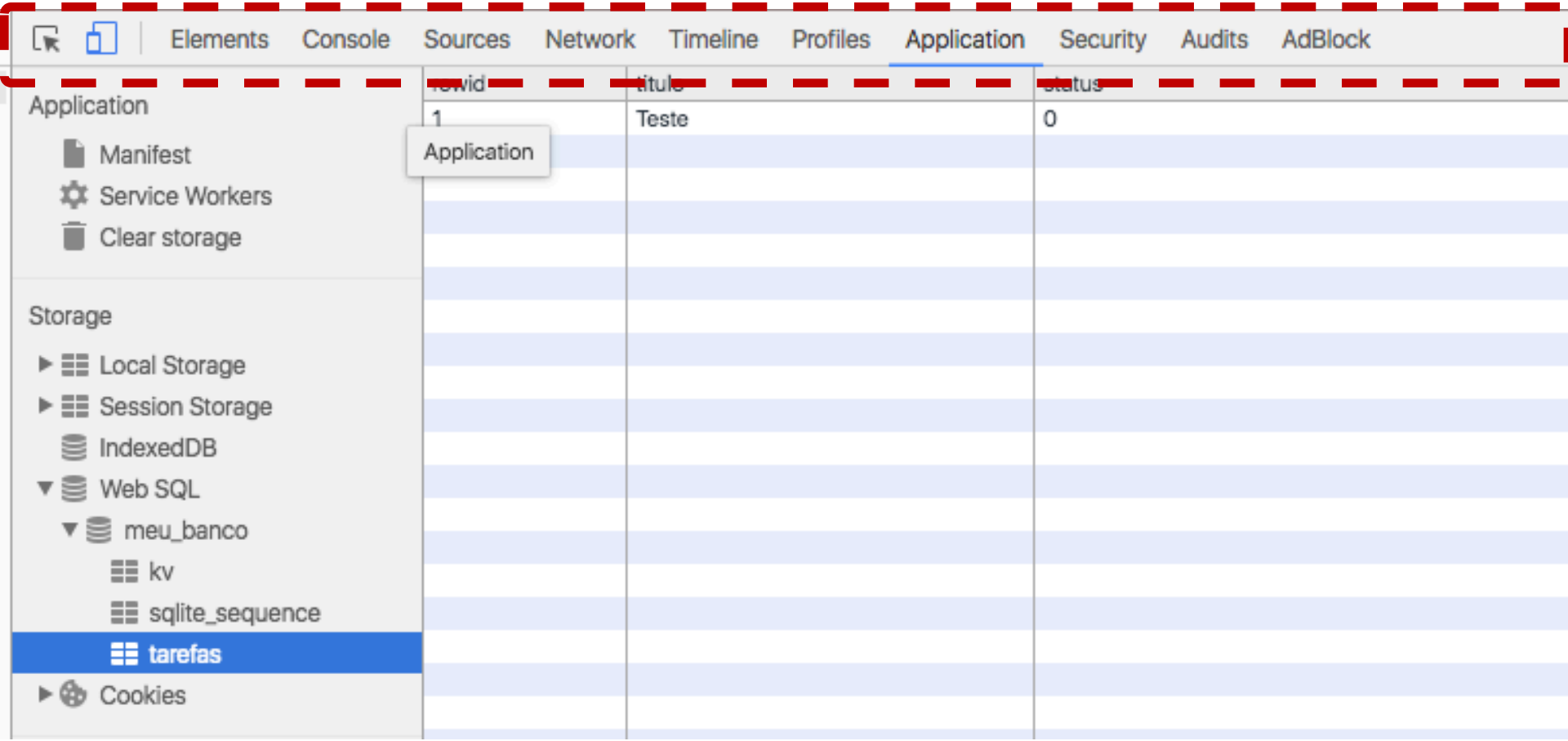
Repare que o comando QUERY recebe dois parâmetros: o primeiro a SQL e o segundo um array com os valores que devem ser substituídos no array ao invés do ?

```
23      salvar(titulo) {
24        this.db.query("INSERT INTO tarefas(titulo, status) VALUES (?,?)", [titulo, 0]).then(
25            (resposta) => { console.log("ID: " + resposta.res.insertId) }
26        );
27      }
```

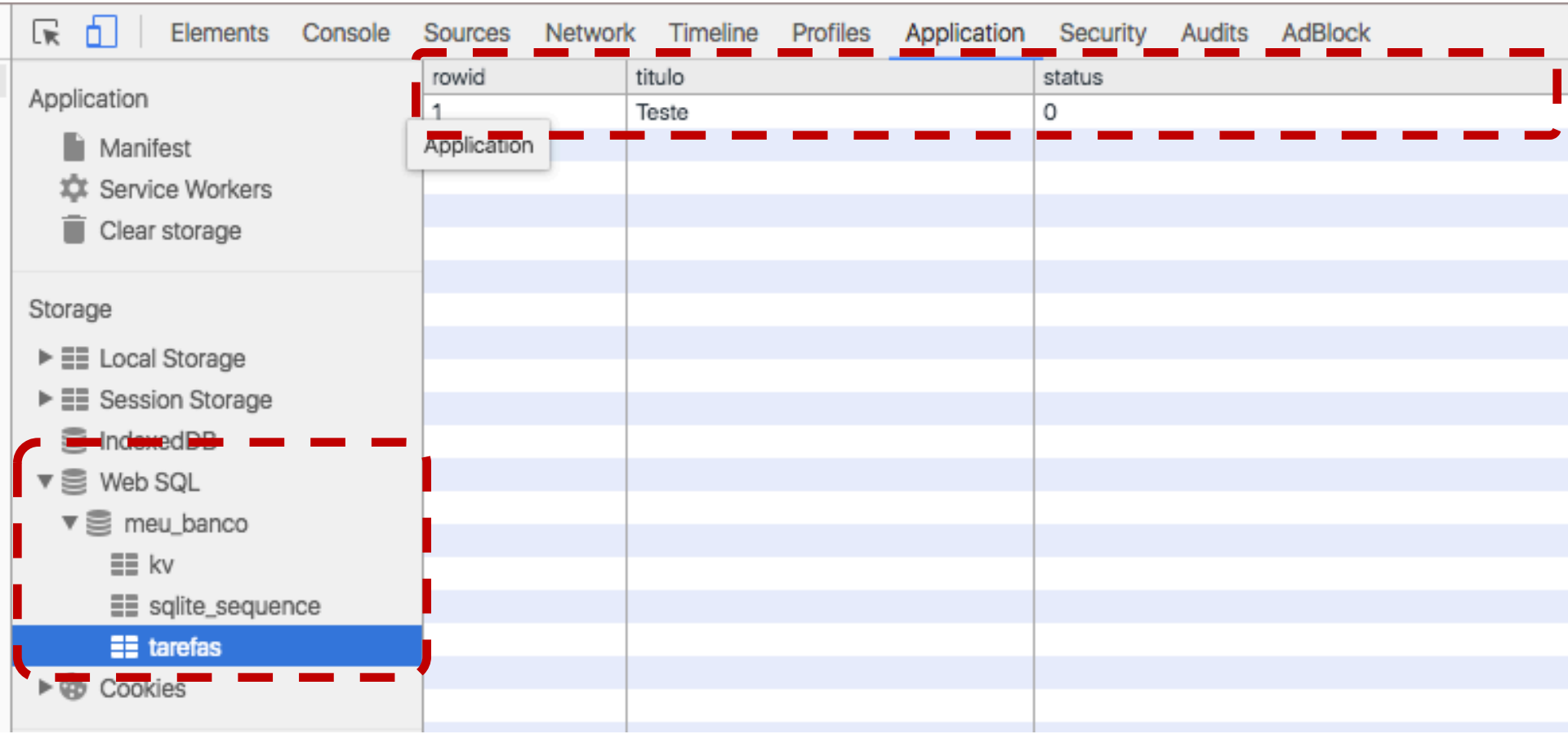
Então, você executa a query com o método then. Assim, você pode executar uma função anônima com a resposta vinda da query. Neste exemplo, no retorno da query temos o ID da tarefa inserida.

```
23      salvar(titulo) {
24          this.db.query("INSERT INTO tarefas(titulo, status) VALUES (?,?)", [titulo, 0]).then(
25              (resposta) => { console.log("ID: " + resposta.res.insertId) }
26          );
27      }
```

Ao testar e executar este método, conseguimos criar e inserir dados no banco de dados. Ao testar no navegador (Chrome) é possível acessar via Painel de Desenvolvedor, na aba Applications o WebSQL



Repare que a tabela tarefas foi criada com sucesso. E que temos à direita os dados salvos nesta tabela.



Outro exemplo: vamos usar o SELECT para carregar uma lista de tarefas a partir do banco de dados. Primeiro, executamos a SQL.

```
34      getTarefas() {
35          this.db.query("SELECT * FROM tarefas").then((resposta) => {
36              let resultado = resposta.res.rows;
37              for (let i = 0; i < resultado.length; i++) {
38                  this.tarefas.push({
39                      id: resultado.item(i).id,
40                      titulo: resultado.item(i).titulo,
41                      status: resultado.item(i).status
42                  });
43              }
44          });
45      }
```

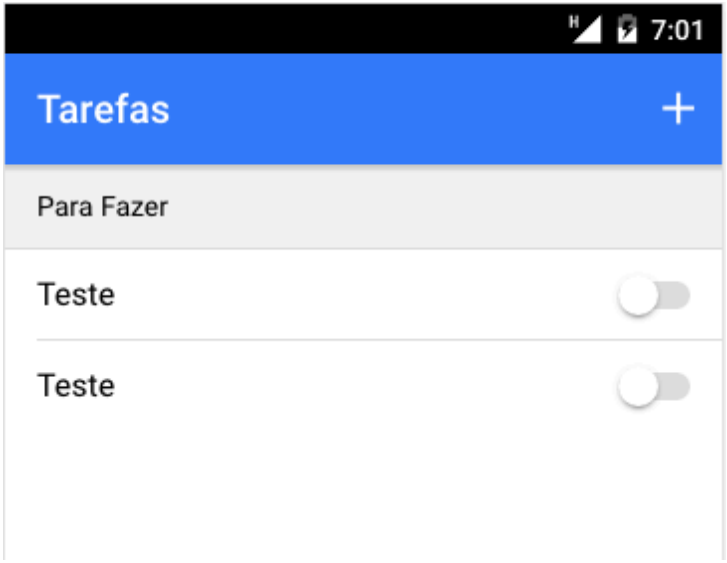
Então, podemos processar a resposta usando um for, e adicionar em um vetor chamado tarefas. Os resultados são salvos em resposta.res.rows.

```
34      getTarefas() {
35          this.db.query("SELECT * FROM tarefas").then((resposta) => {
36              let resultado = resposta.res.rows;
37              for (let i = 0; i < resultado.length; i++) {
38                  this.tarefas.push({
39                      id: resultado.item(i).id,
40                      titulo: resultado.item(i).titulo,
41                      status: resultado.item(i).status
42                  });
43              }
44          });
45      }
```

Em seguida, para cada valor de resposta, podemos adicionar uma tarefa no vetor tarefas na forma de um objeto. Para acessar os dados, fazemos resultado.item(i)

```
34      getTarefas() {
35          this.db.query("SELECT * FROM tarefas").then((resposta) => {
36              let resultado = resposta.res.rows;
37              for (let i = 0; i < resultado.length; i++) {
38                  this.tarefas.push({
39                      id: resultado.item(i).id,
40                      titulo: resultado.item(i).titulo,
41                      status: resultado.item(i).status
42                  });
43              }
44          });
45      }
```

*Agora você pode processar esta lista no HTML usando o *ngFor e mostrar os valores da lista e o status da tarefa.*




```
<ion-item *ngFor="let tarefa of tarefas">
  <ion-label>{{ tarefa.titulo }}</ion-label>
  <ion-toggle checked="{{ tarefa.status }}"></ion-toggle>
</ion-item>
```

Agora é a sua vez: teste os exemplos apresentados na aula em um projeto Ionic. Além disso, crie funções para testar o UPDATE e o DELETE.