01

Salvar dados do usuário

Transcrição

Estamos com a nossa aplicação aberta e vou revisar o fluxo do agendamento. Faremos alguns testes de agendamento. Ao repetirmos o processo três, constatamos que tivemos um problema no envio da requisição. Nosso cliente informou que este é um problema do Back-end.

Em situações como esta, precisamos que os dados preenchidos pelo usuário no formulário de agendamento fiquem salvos no dispositivo, desta forma eles poderão ser enviados mais tarde. Separei algumas opções para que você veja como é possível salvar os dados diretamente no mobile.

LocalStorage

WebSQL

SQLite

Como estamos trabalhando com uma aplicação híbrida, o **LocalStorage** rodará em cima de uma WebView - uma forma de browser dentro do dispositivo. A WebView agrupará a chave e dentro desta, ele colocará o valor. Isto resolveria o nosso problema em partes. Porém ele é mais usado para guardarmos o usuário e o token da senha (cuidado para não salvar a senha no LocalStorage, isto não seria bom).

A segunda opção é o **WebSQL**, ele também funciona dentro do WebView. Trata-se de um banco de dados relacional, isto significa que podemos criar tabelas, relacionamentos. É uma opção que atende bem a nossa necessidade.

E temos o **SQLite**, que é um banco de dados relacional bastante parecido com o WebSQL. No entanto, trata-se de uma dependência. Precisaríamos instalá-lo porque ele é um plugin. Casos assim, podemos ter problemas de versionamento, ou seja, pode haver conflitos de versões.

Após estas apresentações, vemos que a melhor opções é o WebSQL, porque ele é nativo e assim, usaremos o banco de dados do browser.

Agora, nós usaremos o WebSQL para salvarmos os dados no dispositivo. Faremos a configuração do banco de dados em um arquivo de valores. Como trabalhamos com o Angular, usaremos um arquivo `values`. Dentro da pasta `js`, geraremos o arquivo `database.value.js`.

Começaremos chamando o `angular.module`.

```
angular.module('starter')
.value('DatabaseValues', {
    bancoDeDados : null,
    setup : function(){
        window.openDatabase('aluraCar', '1.0', 'Banco de dados da aplicacao', 3000)
    }
  })
```

O primeiro parâmetro do `value` é um nome e o segundo, um valor. Inicialmente, o valor será nulo. Como não configuramos o banco, precisamos ver se ele é válido para avaliarmos. Para fazermos a configuração usamos o `setup` e o atributo recebeu uma `function`. Com o `window`, chamamos o o browser ("a janela"), com a função `openDatabase()` que possui quatro parâmetros. Quando trabalhamos com uma aplicação WebSQL, temos a limitação de trabalhar com até 5 megas. Nós usamos no quarto parâmetro o valor `3000`.

Esta é a configuração do banco de dados. Quando chamarmos o `setup`, queremos que ele pegue a variável `bancoDeDados` e setaremos a configuração. Depois que o banco for criado, podemos fazer transações: criar tabelas, inserir, pegar e deletar dados... Precisaremos popular a variável `bancoDeDados`. Para isto, adicionaremos o `this.bancoDeDados`.

```
angular.module('starter')
.value('DatabaseValues', {
    bancoDeDados : null,
    setup : function(){
        this.bancoDeDados = window.openDatabase('aluraCar', '1.0', 'Banco de dados da aplicacao', 3(
    }
  })
```

Em seguida, vamos carregar o arquivo `database.value.js` no `index.html`:

```
<script src="js/database.value.js"></script>
```

Agora, aonde vamos rodar a configuração, para criar o banco de dados? No momento em que iniciamos é o melhor momento, nós já podemos criar o banco de dados. Qual o primeiro método que vamos executar ao iniciarmos a aplicação. No arquivo `app.js`, começamos carregando o módulo `starter` e chamamos o método `run()`.

```
angular.module('starter', ['ionic', 'idf.br-filters', 'ngCordova', ionic-datepicker]) {

  .run(function($ionicPlatform) {
    $ionicPlatform.ready(function() {
      if(window.cordova && window.cordova.plugins.Keyboard) {
        // Hide the accessory bar by default (remove this to show the accessory bar above the keyboa
        // for form inputs)
        cordova.plugins.Keyboard.hideKeyboardAccessoryBar(true);
      }
      if(window.StatusBar) {
        StatusBar.styleDefault();
      }
    });
  })
```

A melhor solução é criarmos o banco de dados no método `run()`. Vamos começar injetando o `DatabaseValues`.

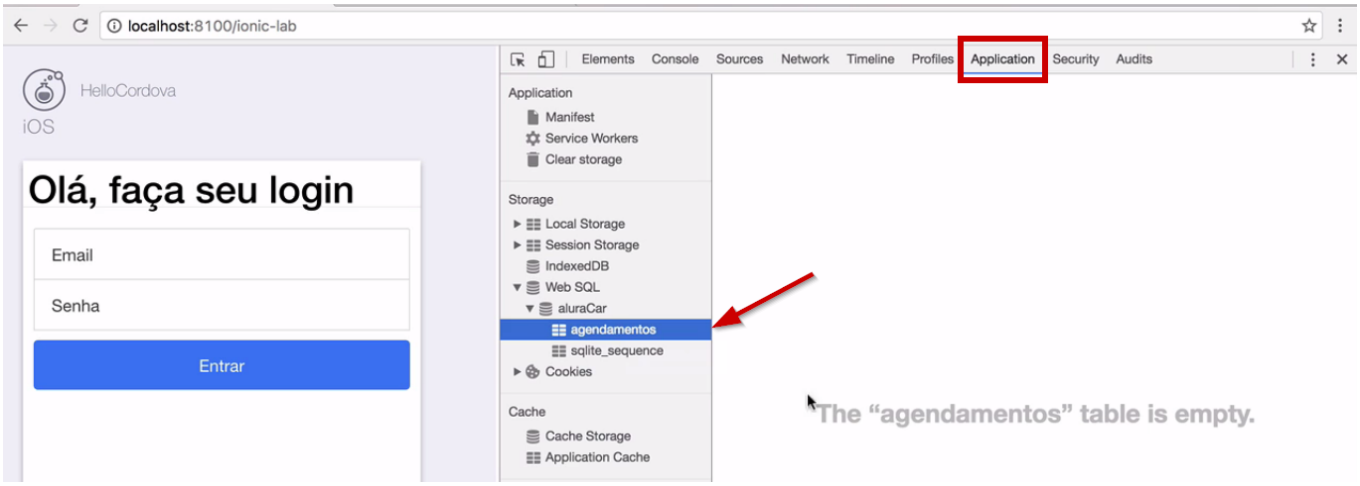
```
angular.module('starter', ['ionic', 'idf.br-filters', 'ngCordova', ionic-datepicker]) {

  .run(function($ionicPlatform, DatabaseValues) {
//...
```

Em seguida, abaixo do `if` , chamaremos o `DatabaseValues` e depois, o `setup` , porque queremos configurar.

```
DatabaseValues.setup();
DatabaseValues.bancoDeDados.transaction(function(transacao){
  transacao.executeSql('CREATE TABLE IF NOT EXISTS agendamentos (id INTEGER NOT NULL PRIMARY KEY AUT
});
```

O `transaction()` recebeu como parâmetro uma `function` e, dentro deste, uma `transacao` . Na `transacao` executamos o Sql (`executeSql`). Adicionaremos em seguida dois parâmetros, uma *string* e o segundo, é um *array* com dados dinâmicos da SQL. Então, o primeiro parâmetro é um SQL. É importante adicionarmos o `CREATE TABLE IF NOT EXISTS` para criarmos uma tabela só uma vez. O `confirmado BOOLEAN` também tem bastante relevância, porque quando tivermos sucesso colocaremos este campo como `true` . Quando tiver retornado com erro, colocaremos `false` . Com este campo saberemos se tivemos sucesso ou não no agendamento.

Vamos testar a aplicação. A seguir, abriremos a ferramenta do desenvolvedor e acessaremos a aba `Application` . Depois, selecionaremos o banco Web SQL -> aluraCar -> agendamentos.



Vemos dentro do "AluraCar", o "sqlite_sequence", é a sequência que ele usa para autoimplementar o *Id*. Não o apague!

Alcançamos nossos objetivos da aula!