

IONIC 7 CON SQLITE

1. Debemos comenzar a instalar las dependencias de sqlite:

- `npm install --save @capacitor-community/sqlite`
- `npm install @capacitor/preferences`: este plugin nos permite almacenar datos en nuestro dispositivo móvil o en el navegador.
- `npm install @capacitor/device`
- `npm install sql.js`: nos permite ejecutar sql en el navegador.

2. Necesitamos copiar el archivo **de node_modules/sql.js/dist el archivo sql-wasm.wasm a la carpeta assets**: es quien permite ejecutar las peticiones que realicemos en la web.

3. Instalar jeep-sqlite: nos permite utilizar sqlite en el navegador, se ejecuta cuando estemos en web: `npm install @jeep-sqlite`.

4. Importamos la biblioteca en el archivo app-module.ts

```
ie.service.ts TS app.modules.ts X {} db.json {} base.json TS app.component.ts
pp > TS app.modules.ts > AppModule
import { CUSTOM_ELEMENTS_SCHEMA, NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { RouteReuseStrategy } from '@angular/router';

import { IonicModule, IonicRouteStrategy } from '@ionic/angular';

import { AppComponent } from './app.component';
import { AppRoutingModule } from './app-routing.module';
import { provideHttpClient } from '@angular/common/http';

import { defineCustomElements as jeepSqlite } from 'jeep-sqlite/loader';
jeepSqlite(window)

@NgModule({
  declarations: [AppComponent],
  imports: [BrowserModule, IonicModule.forRoot(), AppRoutingModule],
  providers: [{ provide: RouteReuseStrategy, useClass: IonicRouteStrategy }, provideHttpClient()],
  bootstrap: [AppComponent],
  schemas: [CUSTOM_ELEMENTS_SCHEMA]
})
export class AppModule {}
```

5. Invocamos el componente en el archivo app.component.html:

```
dit Selection View Go ... <- -> Mysqlite
qlite.service.ts TS app.modules.ts app.component.html X {} db.json
> app > <> app.component.html > ...
1 <ion-app>
2 <ion-router-outlet *ngIf="load"></ion-router-outlet>
3 <jeep-sqlite *ngIf="isWeb"></jeep-sqlite>
4 </ion-app>
```

6. Implementamos la siguiente configuración en app.component.ts:

```
TS sqlite.service.ts TS app.module.ts app.component.html TS app.component.ts X
src > app > TS app.component.ts > AppComponent > initApp
1 import { Component } from '@angular/core';
2 import { Device } from '@capacitor/device';
3 import { Platform } from '@ionic/angular';
4 import { SQLiteService } from '../services/sqlite.service';
5
6 @Component({
7   selector: 'app-root',
8   templateUrl: 'app.component.html',
9   styleUrls: ['app.component.scss'],
10 })
11 export class AppComponent {
12   public isWeb: boolean;
13   public load: boolean;
14
15   constructor(private sqlite: SQLiteService,
16               private platform: Platform) {
17     this.isWeb=false;
18     this.load=false;
19     this.initializeApp();
20   }
21
22   initApp(){
23     this.platform.ready().then( async () => {
24       const info = await Device.getInfo();
25       this.isWeb = info.platform == 'web';
26       this.sqlite.init();
27       this.sqlite.dbReady.subscribe(load => {
28         this.load = load;
29       })
30     })
31   }
32 }
```

CONFIGURACIÓN DEL SERVICIO:

- Implementamos servicio en la **carpeta services: sqlite.service.ts**
El servicio comprobara si la base de datos esta creada, comprobando desde app.component.ts si la base de datos esta preparada.

Tendremos 4 parámetros:

- dbready: observable que nos indica si la base de datos esta preparada.
- isWeb: nos indica si se está en modo web.
- IsIos: nos indica si estamos utilizando IOS.
- dbname: nos indica el nombre de la base de datos que estamos usando.

```
← → MySQLite
TS sqlite.service.ts X TS app.module.ts app.component.html TS app.component.ts
src > app > services > TS sqlite.service.ts > SQLiteService > create > set > values
1 import { Injectable } from '@angular/core';
2 import { HttpClient } from '@angular/common/http';
3 import { BehaviorSubject } from 'rxjs';
4 import { CapacitorSQLite, capSQLiteChanges, capSQLiteValues } from '@capacitor-community/sqlite';
5 import { Device } from '@capacitor/device';
6 import { Preferences } from '@capacitor/preferences';
7 import { JsonSqlite } from 'jeep-sqlite/dist/types/interfaces/interfaces';
8
9 @Injectable({
10   providedIn: 'root'
11 })
12 export class SQLiteService {
13   public dbReady: BehaviorSubject<boolean>;
14   public isWeb: boolean;
15   public isIos: boolean;
16   public dbname='';
17
18   constructor(private httpClient: HttpClient) {
19     this.dbReady = new BehaviorSubject(false);
20     this.isWeb = false;
21     this.isIos = false;
22     this.dbname = '';
23   }
24
25 }
```

Tendremos los siguientes métodos:

Método init():

```
25
26 async init(){
27   const info = await Device.getInfo();
28   const sqlite = CapacitorSQLite as any;
29
30   if (info.platform == 'android'){
31     try {
32       await sqlite.requestPermissions();
33     } catch (error) {
34       console.error("Esta app necesita permisos para funcionar")
35     }
36   } else if (info.platform == 'web') {
37     this.isWeb = true;
38     await sqlite.initWebStore();
39   } else if (info.platform == 'ios') {
40     this.isIos = true;
41   }
42   this.setupDatabase();
43 } //finMetodo
44
45
```

Método setupDatabase(): nos permite comprobar si necesitamos crear la base de datos o la debemos recuperarla:

```
async setupDatabase(){
  const dbSetup = await Preferences.get({ key: 'first_setup_key' })
  if (!dbSetup.value) {
    this.downloadDatabase();
  } else {
    this.dbname = await this.getDbName();
    await CapacitorSQLite.createConnection({ database: this.dbname });
    await CapacitorSQLite.open({ database: this.dbname })
    this.dbReady.next(true);
  }
}
```

Método downloadDatabase(): si la base de datos la debemos crear, la descargamos de la carpeta assets/db.json (contiene la estructura de la base de datos).

```
downloadDatabase() {
  this.httpClient.get('assets/db/basedata.json').subscribe(async (jsonExport: JsonSqlite) => {
    const jsonString = JSON.stringify(jsonExport);
    const isValid = await CapacitorSQLite.isJsonValid({ jsonString });
    if (isValid.result) {
      this.dbname = jsonExport.database;
      await CapacitorSQLite.importFromJson({ jsonString });
      await CapacitorSQLite.createConnection({ database: this.dbname });
      await CapacitorSQLite.open({ database: this.dbname })
      await Preferences.set({ key: 'first_setup_key', value: '1' })
      await Preferences.set({ key: 'dbname', value: this.dbname })
      this.dbReady.next(true);
    }
  })
} //fin
```

Archivo basedata.json: implementamos estructura para usuarios

```
TS home.page.ts {} basedata.json X TS sqlite.service.ts <> home.page.html
src > assets > db > {} basedata.json > ...
1 {
2   "database": "basedatos.db",
3   "version": 1,
4   "encrypted": false,
5   "mode": "full",
6   "tables": [
7     {
8       "name": "usuarios",
9       "schema": [
10        { "column": "rut", "value": "TEXT PRIMARY KEY"},
11        { "column": "email", "value": "TEXT NOT NULL"},
12        { "column": "password", "value": "TEXT NOT NULL" }
13      ]
14    }
15  ]
16 }
```

Método getDbname(): nos permite traer el nombre de la base de datos

```
async getDbName() {
  if (!this.dbname) {
    const dbname = await Preferences.get({ key: 'dbname' })
    if (dbname.value) {
      this.dbname = dbname.value
    }
  }
  console.log(this.dbname);
  return this.dbname;
}
```

Métodos CRUD:

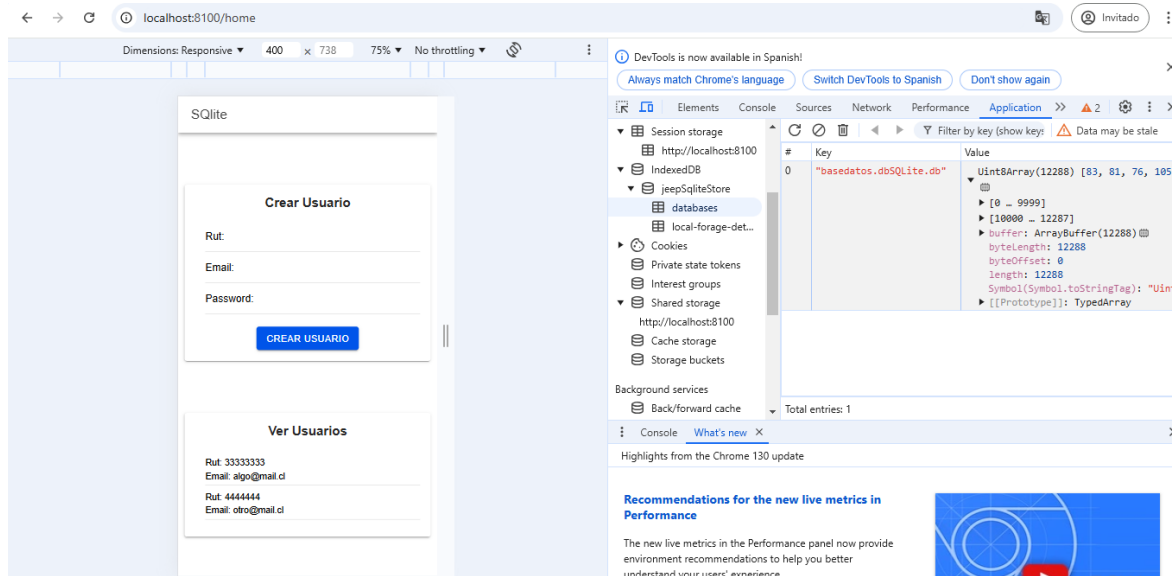
▪ **Método create:**

```
async create(rut: string, email: string, password:string) {
  let sql = 'INSERT INTO usuarios VALUES(?,?,?)';
  const dbname = await this.getDbName();
  return CapacitorSQLite.executeSet({
    database: dbname,
    set: [
      {
        statement: sql,
        values: [
          rut,
          email,
          password
        ]
      }
    ]
  }).then((changes: capSQLiteChanges) => {
    if (this.isIos) {
      CapacitorSQLite.saveToStore({ database: dbname });
    }
    return changes;
  }).catch(err => Promise.reject(err))
}
```

▪ **Método read:**

```
TS sqlite.service.ts x {} db.json TS app.module.ts TS app.component.ts
src > app > services > TS sqlite.service.ts > $ SqliteService > read > then() callback
12 export class SqliteService {
110
111
112   async read() {
113     let sql = 'SELECT * FROM usuarios';
114     const dbname = await this.getDbName();
115     return CapacitorSQLite.query({
116       database: dbname,
117       statement: sql,
118       values: [] // necesario para android
119     }).then((response: capSQLiteValues) => {
120       let usuarios = [];
121
122       if (this.isIos && response.values.length > 0) {
123         response.values.shift();
124       }
125       for (let index = 0; index < response.values.length; index++) {
126         const usuario = response.values[index];
127         usuarios.push(usuario);
128       }
129       return usuarios;
130     }).catch(err => Promise.reject(err))
131   }
132
133 }
```

Implementamos el servicio en el page:



[Home.page.ts](#): importamos el servicio y definimos los parámetros para crear formulario que permita crear a un usuario para luego visualizarlos en un ion-list:

```
TS home.page.ts x {} basedata.json TS sqlite.service.ts home.page.html home.page.s
src > app > home > TS home.page.ts > ...
1 import { Component, OnInit } from '@angular/core';
2 import { SqliteService } from '../services/sqlite.service';
3
4 @Component({
5   selector: 'app-home',
6   templateUrl: 'home.page.html',
7   styleUrls: ['home.page.scss'],
8 })
9 export class HomePage implements OnInit {
10
11   public rut: string;
12   public email: string;
13   public password: string;
14   public usuarios=[];
15
16   constructor(private sqliteservice: SqliteService) {}
17
18   ngOnInit(){
19     this.read();
20   }
21 }
```

Implementamos dos métodos que permiten crear un usuario y ver usuarios:

```
TS home.page.ts x {} basedata.json TS sqlite.service.ts home.page.html home.page.s
src > app > home > TS home.page.ts > HomePage
9 export class HomePage implements OnInit {
10
11
12
13
14
15
16
17
18
19
20
21
22
23   create(){
24     this.sqliteservice.create(this.rut, this.email, this.password).then( (changes) =>{
25       console.log(changes);
26       console.log("Creado");
27       this.rut="";
28       this.email = "";
29       this.password="";
30       this.read(); // Volvemos a leer
31     }).catch(err => {
32       console.error(err);
33       console.error("Error al crear");
34     })
35   }
36
37   read(){
38     this.sqliteservice.read().then( (data: string[]) => {
39       this.usuarios = data;
40       console.log("Leído");
41       console.log(this.usuarios);
42     }).catch(err => {
43       console.error(err);
44       console.error("Error al leer");
45     })
46   }
47
48 }
```

Home.page.html: creamos formulario para registrar y luego vemos los usuarios almacenados:

```
TS home.page.ts {} basedata.json TS sqlite.service.ts < home.page.html X home.page.scss
src > app > home > home.page.html > ion-content > ion-card > ion-card-header
1 <ion-header [translucent]="true">
2   <ion-toolbar>
3     <ion-title>
4       SQLite
5     </ion-title>
6   </ion-toolbar>
7 </ion-header>
8
9 <ion-content [fullscreen]="true">
10  <ion-card>
11    <ion-card-header>
12      <ion-card-title style="font-weight: bold; text-align: center;">Crear Usuario</ion-card-title>
13    </ion-card-header>
14
15    <ion-card-content>
16      <ion-item>
17        <ion-input label="Rut:" type="text" [(ngModel)]="rut" name="rut"></ion-input>
18      </ion-item>
19      <ion-item>
20        <ion-input label="Email:" type="text" [(ngModel)]="email" name="email"></ion-input>
21      </ion-item>
22      <ion-item>
23        <ion-input label="Password:" type="password" [(ngModel)]="password" name="password"></ion-input>
24      </ion-item>
25      <div style="text-align: center; margin-top: 15px;">
26        <ion-button (click)="create()">
27          Crear Usuario
28        </ion-button>
29      </div>
30    </ion-card-content>
31  </ion-card>
32 </ion-content>
```

```
home.page.ts {} basedata.json TS sqlite.service.ts < home.page.html X home.page.scss
src > app > home > home.page.html > ion-content > ion-card > ion-card-header
<ion-content [fullscreen]="true">
  <ion-card>
    <ion-card-header>
      <ion-card-title style="font-weight: bold; text-align: center;">Ver Usuarios</ion-card-title>
    </ion-card-header>
    <ion-card-content>
      <ion-list>
        <ion-item *ngFor="let usuario of usuarios" style="margin-bottom: 5px;">
          <p>
            <ion-label> Rut: {{usuario.rut}}</ion-label>
            <ion-label>Email: {{usuario.email}}</ion-label>
          </p>
        </ion-item>
      </ion-list>
    </ion-card-content>
  </ion-card>
</ion-content>
```

Desde Android:

