IONIC 7 CON SQLITE

1. **Debemos comenzar a instalar las dependencias de sqlite:**

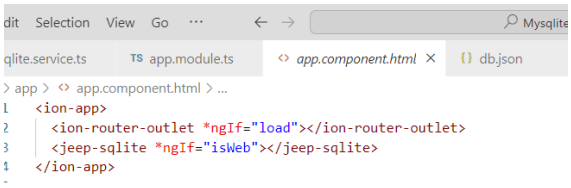
   - npm install --save @capacitor-community/sqlite
   - npm install @capacitor/preferences: este plugin nos permite almacenar datos en nuestro dispositivo móvil o en el navegador.
   - npm install @capacitor/device
   - npm install sql.js: nos permite ejecutar sql en el navegador.

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

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



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

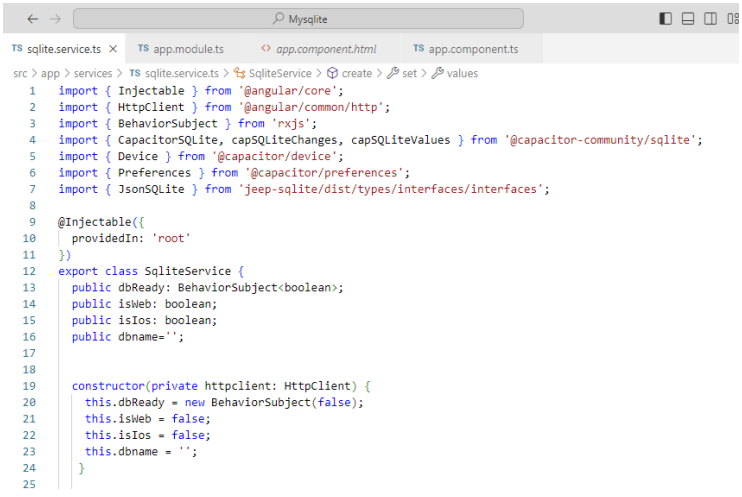
```
import { Component } from '@angular/core';
import { Device } from '@capacitor/device';
import { Platform } from '@ionic/angular';
import { SqliteService } from './services/sqlite.service';

@Component({
  selector: 'app-root',
  templateUrl: 'app.component.html',
  styleUrls: ['app.component.scss'],
})
export class AppComponent {
  public isWeb: boolean;
  public load : boolean;

  constructor(private sqlite: SqliteService,
              private platform: Platform) {
                this.isWeb=false;
                this.load=false;
                this.initApp();
              }

  initApp(){
    this.platform.ready().then( async () => {
      const info = await Device.getInfo();
      this.isWeb = info.platform == 'web';
      this.sqlite.init();
      this.sqlite.dbReady.subscribe(load => {
        this.load = load;
      })
    })
  }
}
```

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.
- Islos: nos indica si estamos utilizando IOS.
- dbname: nos indica el nombre de la base de datos que estamos usando.



```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { BehaviorSubject } from 'rxjs';
import { CapacitorSQLite, capSQLiteChanges, capSQLiteValues } from '@capacitor-community/sqlite';
import { Device } from '@capacitor/device';
import { Preferences } from '@capacitor/preferences';
import { JsonSQLite } from 'jeep-sqlite/dist/types/interfaces/interfaces';

@Injectable({
  providedIn: 'root'
})
export class SqliteService {
  public dbReady: BehaviorSubject<boolean>;
  public isWeb: boolean;
  public isIos: boolean;
  public dbname='';


  constructor(private httpclient: HttpClient) {
    this.dbReady = new BehaviorSubject(false);
    this.isWeb = false;
    this.isIos = false;
    this.dbname = '';
   }
```

Tendremos los siguientes métodos:

Método init():

```
async init(){
    const info = await Device.getInfo();
    const sqlite = CapacitorSQLite as any;

    if (info.platform == 'android'){
      try {
        await sqlite.requestPermissions();
      } catch (error) {
        console.error("Esta app necesita permisos para funcionar")
      }
    } else if (info.platform == 'web') {
        this.isWeb = true;
        await sqlite.initWebStore();
    } else if (info.platform == 'ios') {
      this.isIos = true;
    }
  this.setupDatabase();
 }//finMetodo
```

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

home.page.ts | basedata.json | sqlite.service.ts | home.page.html

src > assets > db > basedata.json > ...

```
{
    "database": "basedatos.db",
    "version": 1,
    "encrypted": false,
    "mode": "full",
    "tables": [
        {
            "name": "usuarios",
            "schema": [
                {"column": "rut", "value": "TEXT PRIMARY KEY"},
                {"column": "email", "value": "TEXT NOT NULL"},
                {"column": "password", "value": "TEXT NOT NULL"}
            ]
        }
    ]
}
```

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;
}
```

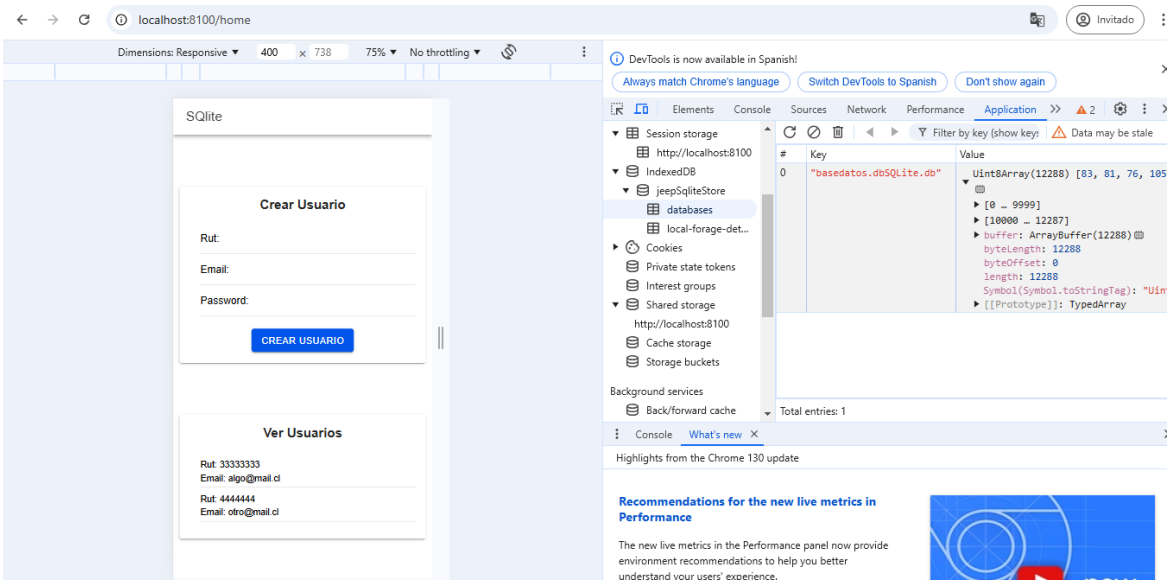
Métodos CRUD:

- **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.isWeb) {
      CapacitorSQLite.saveToStore({ database: dbname });
    }
    return changes;
  }).catch(err => Promise.reject(err))
}
```

- **Método read:**

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
import { Component, OnInit } from '@angular/core';
import { SqliteService } from '../services/sqlite.service';

@Component({
  selector: 'app-home',
  templateUrl: 'home.page.html',
  styleUrls: ['home.page.scss'],
})
export class HomePage implements OnInit {

  public rut: string;
  public email: string;
  public password: string;
  public usuarios=[];

  constructor(private sqliteservice: SqliteService) {}

  ngOnInit(){
    this.read();
  }
```

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

```ts
export class HomePage implements OnInit {


  create(){
    this.sqliteservice.create(this.rut, this.email, this.password).then( (changes) =>{
      console.log(changes);
      console.log("Creado");
      this.rut='';
      this.email = '';
      this.password='';
      this.read(); // Volvemos a leer
    }).catch(err => {
      console.error(err);
      console.error("Error al crear");
    })
  }

  read(){
    this.sqliteservice.read().then( (data: string[]) => {
      this.usuarios = data;
      console.log("Leido");
      console.log(this.usuarios);
    }).catch(err => {
      console.error(err);
      console.error("Error al leer");
    })
  }
}
```

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

```
<ion-header [translucent]="true">
  <ion-toolbar>
    <ion-title>
      SQlite
    </ion-title>
  </ion-toolbar>
</ion-header>

<ion-content [fullscreen]="true">
  <ion-card>
    <ion-card-header>
      <ion-card-title style="font-weight: bold; text-align: center;">Crear Usuario</ion-card-title>
    </ion-card-header>

    <ion-card-content>
      <ion-item>
      <ion-input label="Rut:" type="text" [(ngModel)]="rut" name="rut"></ion-input>
    </ion-item>
    <ion-item>
      <ion-input label="Email:" type="text" [(ngModel)]="email" name="email"></ion-input>
    </ion-item>
    <ion-item>
      <ion-input label="Password:" type="password" [(ngModel)]="password" name="password"></ion-input>
    </ion-item>
    <div style="text-align: center; margin-top: 15px;">
      <ion-button (click)="create()">
        Crear Usuario
      </ion-button>
    </div>
    </ion-card-content>
  </ion-card>
```

```
<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: