Lab 02: Mobile Data using sqlite-net

Prerequisites

You will need a Windows PC with the Android SDK and Xamarin tools installed. We will be using the Android emulator to test the code we are building, so make sure to have a virtual device already configured and ready to run. See the Xamarin.Android setup documentation if you need help getting your environment setup.

Downloads

Links to lab sample code (if available).

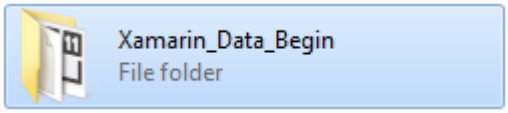
Lab Goals

The goal of this lab will be to introduce integrating local data storage on mobile devices using the sqlite-net ORM.

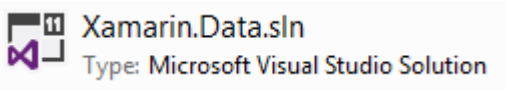
Steps

Open the Starting Solution

1. Launch **Xamarin Studio**
2. Click **Open...** on the Xamarin Studio Welcome and navigate to the **Lab 02 Resources** folder included with this document.
3. Locate the **Xamarin_Data_Begin** folder – make sure it's the **begin** and not the **completed** folder.



4. Inside the **Xamarin_Data_Begin** folder, you will find a **Xamarin.Data.sln** file – double click on this file to open the starter solution

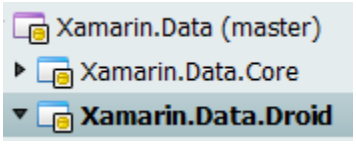


5. Go ahead and build and run the application in the emulator to make sure it compiles and your environment is ready. Let the instructor know if you have any trouble.

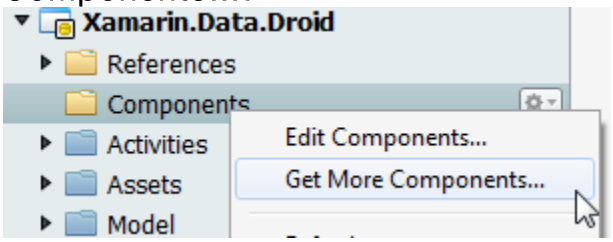
Setting up the SQLite.Net ORM

We are going to reference the SQLite.Net ORM library to help streamline development for local data storage

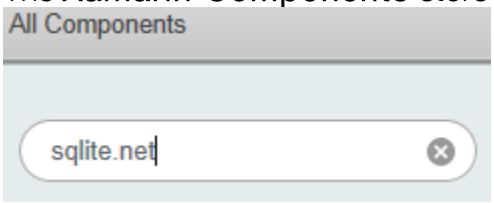
1. In **Xamarin Studio**, locate the project named **Xamarin.Data.Droid**



2. Right click on the components folder and select **Get More Components....**

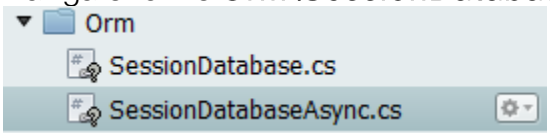


3. The **Xamarin Components** store will open, search for **sqlite.net**



4. From the result screen, select the **SQLite.NET** component and select **Add to App**

5. Navigate to the **Orm\SessionDatabaseAsync.cs** file



6. Locate `//TODO: Step 1 - Setup the Database Path` and uncomment the code within the **DatabaseFilePath** property

```
public static string DatabaseFilePath {
     get {
            var sqliteFilename = "SessionDB.db3";

            #if __ANDROID__

     // Just use whatever directory SpecialFolder.Personal returns

     string libraryPath = Environment.GetFolderPath(Environment.Specia
lFolder.Personal); ;
            #else
```

```
            // we need to put in /Library/ on iOS5.1 to meet Apple's iCloud t
erms
            // (they don't want non-user-generated data in Documents)

            string documentsPath = Environment.GetFolderPath (Environment.Spe
cialFolder.Personal); // Documents folder

            string libraryPath = Path.Combine (documentsPath, "..", "Library"
); // Library folder
            #endif

            var path = Path.Combine (libraryPath, sqliteFilename);

            return path;
        }
}
```

Tip: There is not an app.config or web.config available for Xamarin.Android and Xamarin.iOS applications. Using Preprocessor Directives offers developers an easy way to define different configurations for environments and devices.

7. Navigate to the **Model\Session.cs** class
8. Add the following code under `//TODO: Step 2 - Add a primary key` and add in attributes to define a primary key

```
[SQLite.PrimaryKey, SQLite.AutoIncrement, SQLite.Column("_id")]
```

9. Add the following code under `//TODO: Step 3 - Mark column that you will not persist`

```
[SQLite.Ignore()]
```

Tip: sqlite-net has a lot more features than we have seen here. Please reference the sqlite-net wiki for more information.

10. Navigate to `//TODO: Step 4 - Create tables` and add in the following code. This code will create any tables that may not be there when we connect to the database

```
// create the SQLite database tables based on the Model classes
var createTableSession = CreateTableAsync<Session> ();
var createTableSpeaker = CreateTableAsync<Speaker>();

Task.WaitAll(createTableSession, createTableSpeaker);
```

11. Navigate to `//TODO: Step 5 - Select All Records from Table` and add in the following code to query for all data in the table.

```
return await Table<Session>().ToListAsync();
```

12. Navigate to `//TODO: Step 6 - Select records from table using Linq` and add in the following code to query for the first record and return the first match

```
return await Table<Session>().Where(s => s.Id == id).FirstOrDefaultAsy
nc();
```

13. Navigate to `//TODO: Step 7 - Perform an insert or update to the database` and add in the following code to perform an insert or update

```
if (item.Id <= 0)
    return await InsertAsync(item);

await UpdateAsync(item);
return item.Id;
```

14. Navigate to `//TODO: Step 8 - Delete a record from the table` and in the following code to perform a deletion

```
return await DeleteAsync(session);
```

15. Navigate to `//TODO: Step 9 - Delete all data using SQL commands` and add in the following code to delete all data from a table

```
await this.ExecuteAsync ("DELETE FROM Session;");
```

Tip: You can still call standard SQL code when using the sqlite-net code.

16. Save and Build the project and ensure that there are no errors
17. Open the **Activities/SessionsActivity.cs** file
18. Navigate to `//TODO: Step 10 - Delete and save new data` and in the following code to clear and insert data into the sessions adapter

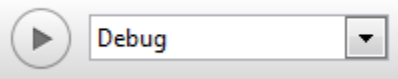
```
await App.Database.DeleteSessionsAsync ();
await App.Database.SaveSessionsAsync (sessions);
```

19. Navigate to `//TODO: Step 11 - Load from database and display` and add the following code to query the database and load data into our adapter

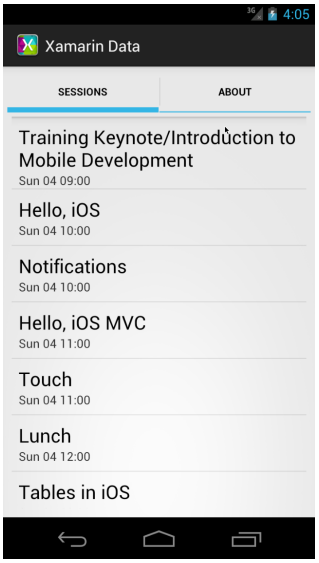
```
var sessions = await App.Database.GetSessionsAsync ();
adapter = new Adapters.SessionsAdapter(this, sessions);
ListView.Adapter = adapter;
```

Testing the Data App

1. Ensure your Android Emulator is running and from Xamarin Studio, Debug or Run the app



2. The application will load data from the database and present the results into the list view



Summary

In this lab, we learned how to integrate a sqlite database to assist us with persisting data locally in our applications. Using the sqlite-net ORM, we generated our tables, created a client, queried for and took action against data in our tables.