Xamarin Fundamentals:

Module 3, Lesson 13  
Data Access using Azure Lab

## Overview

Create a data layer by implementing the basic CRUD functions using the Azure Mobile Client SDK for a remote store and SQLite for local store and synchronizing local and remote data.

## Objectives

In this hands-on lab you will learn how to:

* Build a data layer from scratch and replace TodoItem’s data layer (TodoItemManager.cs)
* Implement CRUD functions create, read, update, and delete
* Run TodoItem using remote Azure data access only
* Add a local SQLite database
* Implement synchronization
* Run TodoItem using synced local and remote data

## Prerequisites

The following are required to complete this hands-on lab:

* Completion of the [Lesson 12 Lab](https://github.com/MSFTImagine/computerscience/blob/master/Complimentary%20Course%20Content/Module3/Labs/Module%203%20Lesson%2012%20Azure%20with%20Xamarin%20Lab.docx)- Azure with Xamarin
* Azure account (or free trial account), Visual Studio with Xamarin installed
* Emulator, simulator, or physical device set up for iOS, Android, or UWP
* Internet connectivity is required for the test device(s)

## Exercises

This hands-on lab includes the following exercises:

* Exercise 1: Enhance existing TodoItem app with CRUD functions
* Exercise 2: Implement remote Azure data access
* Exercise 3: Implement local SQLLite DB and sync

Professor’s Note: By the end of this lab there will be three data layers in your solution. The original one is called TodoManager, and can be remote-only or toggled to sync using SQLite/Azure (this lab does not utilize that toggle). Second, there is a remote-only CRUD-based data layer called TodoManagerCRUDaccess. The third is a SQLite/Azure synchronizing data layer called TodoManagerSync. They are swapped in-and-out of the solution by changing references in TodoList.xaml.cs as detailed in the Exercises. TodoManagerCRUDaccess and TodoManagerSync do exactly what TodoManager can do alone, but in a way that is clearer for learning purposes.

## Exercise 1: CRUD Functions

1. Open your existing TodoItem app solution. If you do not have one, complete Lab 4 from Lesson 12 - Mobile Apps with Microsoft Azure App Service.

2. Create a new C# class called TodoItemManagerCRUD.cs.

3. Add these using statements at the top of file.

using System.Collections.Generic;

using System.Collections.ObjectModel;

using System.Threading.Tasks;

using Microsoft.WindowsAzure.MobileServices;

3. Implement CRUD functions in the TodoItemManagerCRUD class.

public class TodoItemManagerCRUD

{  
 MobileServiceClient MobileService;

IMobileServiceTable<TodoItem> todoTable;

public async Task NewTask(TodoItem item) { }

public async Task<ObservableCollection<TodoItem>> GetTasks() { }

public async Task UpdateTask(TodoItem item) { }

public async Task DeleteTask(TodoItem item) { }

}

5. Study this code closely. This CRUD implementation outline is foundation for many data layers found in professional software products.

6. Do not compile yet. Proceed to Exercise 2.

## Exercise 2: Implement CRUD Functions

1. Implement remote Azure data access in TodoItemManagerCRUD by making Azure Mobile Client SDK calls inside each of its CRUD methods.

Begin by initializing the service in the constructor. The URL of the mobile service client can be found in Constants.ApplicationURL, defined in Constants.cs.

private TodoItemManagerCRUD()

{

mobileService = new MobileServiceClient(Constants.ApplicationURL);

this.todoTable = mobileService.GetTable<TodoItem>();

}

1. Implement the Create function.

public async Task NewTask(TodoItem item)

{

await todoTable.InsertAsync(item);

}

1. Implement the Read function. The todoTable query returns unfinished tasks (not Done).

public async Task<ObservableCollection<TodoItem>> GetTasks()

{

IEnumerable<TodoItem> items = await todoTable

.Where(todoItem => !todoItem.Done)

.ToEnumerableAsync();

return new ObservableCollection<TodoItem>(items);

}

1. Implement the Update function.

public async Task UpdateTask(TodoItem item)

{

await todoTable.UpdateAsync(item);

}

1. Lastly, the Delete function. (This is for completeness only. You won’t actually use this method in this lab.)

public async Task DeleteTask(TodoItem item) // unused, here for completeness

{

await todoTable.DeleteAsync(item);

}

1. Some transaction methods call a few CRUD functions at a time, or conditionally, or with different parameters. In this app, the methods are GetTodoItemsAsync and SaveTaskAsync, plus an extra method called IsOfflineEnabled. Enter them now. These are simplified versions of the same-named methods in TodoManager.cs.

public async Task<ObservableCollection<TodoItem>> GetTodoItemsAsync(bool syncItems = false)

{

return await GetTasks();

}

public async Task SaveTaskAsync(TodoItem item)

{

if (item.Id == null)

{

await NewTask(item);

}

else

{

await UpdateTask(item);

}

}

public bool IsOfflineEnabled

{

get { return todoTable is Microsoft.WindowsAzure.MobileServices.Sync.IMobileServiceSyncTable<TodoItem>; }

}

1. Make TodoItemManagerCRUD into a Singleton (a pattern that allows the object to remain in memory, retaining local and remote connections that can be used at different points in the app lifecycle). It uses a static reference to itself to ensure that once it is created, it will remain in memory as long as the app is running.

public static TodoItemManagerCRUD DefaultManager

{

get

{

return defaultInstance;

}

private set

{

defaultInstance = value;

}

}

1. Include TodoItemManagerCRUD in the app by referencing it. Change references from the static class TodoItemManager to your new TodoItemManagerCRUD in the file TodoList.xaml.cs.

public partial class TodoList : ContentPage

{

TodoItemManagerCRUD manager;

public TodoList()

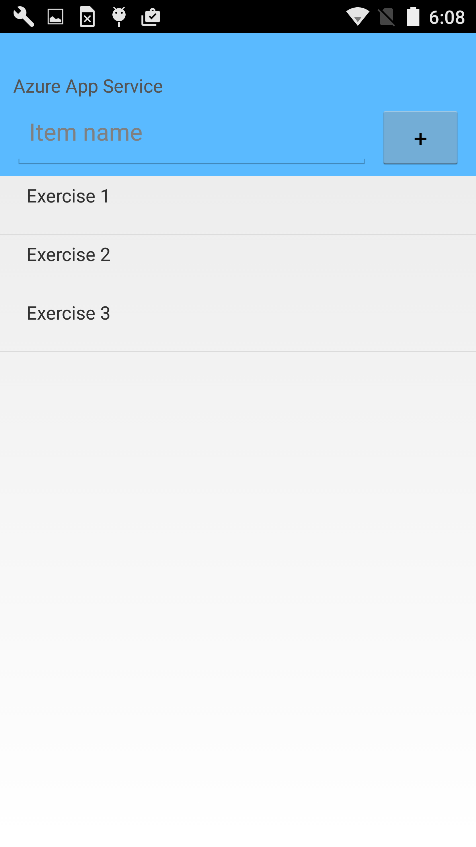
{

InitializeComponent();

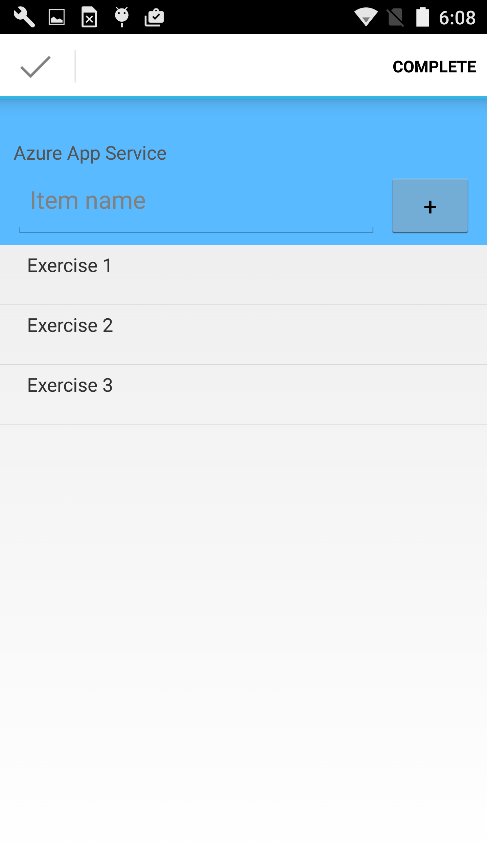
manager = TodoItemManagerCRUD.DefaultManager;

1. Compile and run the solution. Test the app against the Azure SQL Database. Note that there is no local database at this time. Internet connectivity is required.

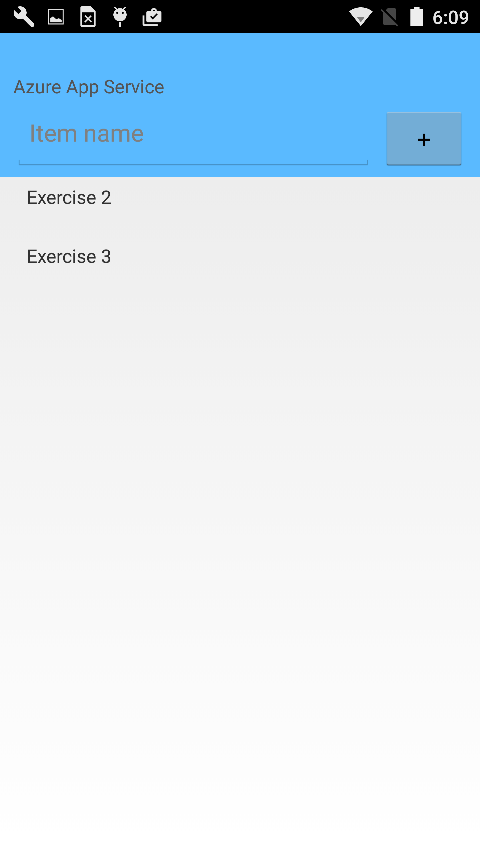
Enter a few tasks.



Mark a task as complete (long press on Android devices/emulators).



The completed task, Exercise 1 in this case, no longer shows in the list due to the query in GetTasks which only retrieves tasks where Done equals false.



This concludes the remote data portion of the lab, which requires the app to be online. The next exercise addresses offline use with online sync.

## Exercise 3: Sync Local Data to Azure

1. Create a copy of TodoItemManagerCRUD.cs and name it TodoItemManagerSync.cs.
2. In TodoItemManagerSync, change all instances of the string TodoItemManagerCRUD to TodoItemManagerSync (there should only be two).
3. In Visual Studio, right-click the solution > **Manage NuGet Packages for Solution...**, then search for and install the **Microsoft.Azure.Mobile.Client.SQLiteStore** NuGet package for all projects in the solution.
4. Ensure that these using statements are at the top of TodoItemManagerSync.cs.

using System.Collections.Generic;

using System.Collections.ObjectModel;

using System.Threading.Tasks;

using Microsoft.WindowsAzure.MobileServices;

using Microsoft.WindowsAzure.MobileServices.SQLiteStore;

using Microsoft.WindowsAzure.MobileServices.Sync;

1. Change the todoTable declaration to a sync-friendly type.

IMobileServiceSyncTable<TodoItem> todoTable;

1. Initialize synchronization between SQLite and Azure.

private TodoItemManagerSync()

{

SQLitePCL.Batteries.Init();

mobileService = new MobileServiceClient(Constants.ApplicationURL);

var path = "localstore.db";

path = SQLitePath.PathName;

var store = new MobileServiceSQLiteStore(path);

store.DefineTable<TodoItem>();

this.mobileService.SyncContext.InitializeAsync(store);

this.todoTable = mobileService.GetSyncTable<TodoItem>();

}

1. Define a the SQLite path and filename by creating a SQLitePath class in the Xamarin.Forms project with a property called PathName.

public class SQLitePath

{

public static string PathName { get; set; } = "localstore.db";

}

Note: For Android, Azure Mobile Client SDK 3.0 brought some changes for synchronization. The SQLite Batteries initialization is now required for all OSes. For Android, the path folder for the MobileServiceClient call is not supplied and must be determined manually. For Xamarin.Forms PCL projects (like the one in this example) this can be done in the Android project and injected back into the Xamarin.Forms PCL. Here’s how:

In Android activity (such as MainActivity.cs)

SQLitePath.PathName = Path.Combine(System.Environment.GetFolderPath(System.Environment.SpecialFolder.Personal), SQLitePath.PathName);

if (!File.Exists(SQLitePath.PathName))

{

File.Create(SQLitePath.PathName).Dispose();

}

1. In TodoItemManagerSync’s GetTodoItemsSync method, add this sync code before the return statement.

if (syncItems)

{

await this.SyncAsync();

}

1. Implement the synchronization method with an explicit Push to catch exceptions.

public async Task SyncAsync()

{

ReadOnlyCollection<MobileServiceTableOperationError> syncErrors = null;

try

{

await this.mobileService.SyncContext.PushAsync();

await this.todoTable.PullAsync("allTodoItems", this.todoTable.CreateQuery());

}

catch (MobileServicePushFailedException exc)

{

if (exc.PushResult != null)

{

syncErrors = exc.PushResult.Errors;

}

}

if (syncErrors != null)

{

foreach (var error in syncErrors)

{

if (error.OperationKind == MobileServiceTableOperationKind.Update && error.Result != null)

{

//Failed, use server data

await error.CancelAndUpdateItemAsync(error.Result);

}

else

{

// Discard change

await error.CancelAndDiscardItemAsync();

}

}

}

}

1. In TodoList.xaml.cs, change the data refresh call in OnAppearing to sync when the app is opened:

await RefreshItems(true, syncItems: true);

1. Include TodoItemManagerSync in the app by referencing it. Change references from the static class TodoItemManager to your new TodoItemManagerSync in the file TodoList.xaml.cs.

public partial class TodoList : ContentPage

{

TodoItemManagerSync manager;

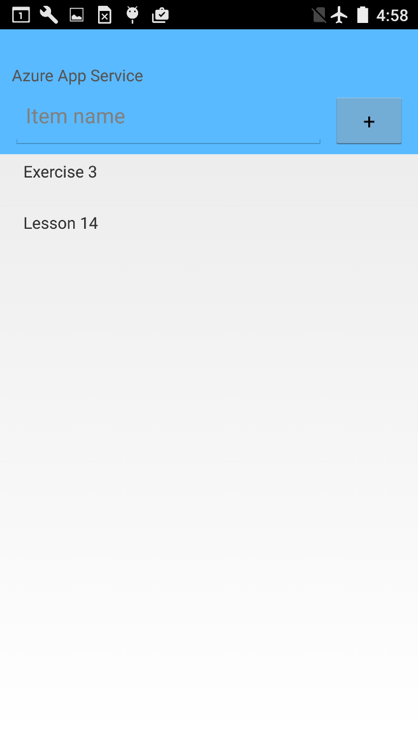
public TodoList()

{

InitializeComponent();

manager = TodoItemManagerSync.DefaultManager;

1. Compile and run. Items added and completed are written to the local SQLite database and synced with the remote Azure data store.



Note: This particular app does not support offline use. That would require code to handle connection failures gracefully. This app does provide the foundation for an online/offline architecture and demonstrates local-to-remote synchronization.

Note: To support Windows 10 devices, download and install [SQLite for the Universal Windows Universal](http://sqlite.org/2016/sqlite-uwp-3120200.vsix). In each Windows app project, right click **References** > **Add Reference...**, expand the **Windows** folder > **Extensions**, then enable the appropriate **SQLite for Windows Runtime** SDK along with the **Visual C++ 2013 Runtime for Windows** SDK. Note that the SQLite SDK names vary slightly with each Windows platform. For a UWP project, install the **Visual C++ 2015 Runtime for Universal Windows Platform apps** SDK.

## Summary

In this hands-on lab, you learned how to:

* Build a data layer from scratch and replace TodoItem’s data layer (TodoItemManager.cs)
* Implement CRUD functions create, read, update, and delete
* Run TodoItem using remote Azure data access only
* Add a local SQLite database
* Implement synchronization
* Run TodoItem using synced local and remote data