Lab 01: iOS Performance

Prerequisites

This lab is based on an iOS project so you will need an OSX based development environment, or the proper Windows setup with an OSX based build host.

Assets

This exercises uses an existing application which is included with these materials – look for it at:

Content/Exercises/PerformanceiOS/Lab Resources

Lab Goals

The goal of this lab will be to demonstrate different performance issues you may encounter in an iOS project. You will need to run the program on a live device to see most of the issues, as the simulator will hide them.

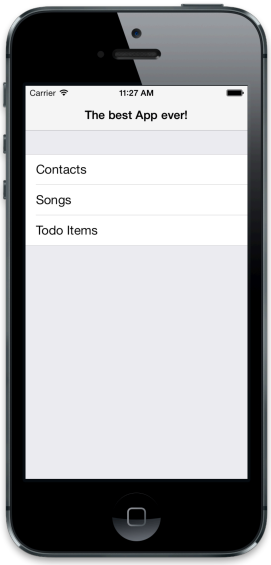
Unlike other labs, this will not walk you through a solution. Instead, it will point out some of the issues with the application and let you attempt to fix them.

If you cannot figure out the problem, or possible solution, there is a Completed form of this lab that has fixes with comments labeled as TODO items so you can easily locate the changes. For the best learning experience, we recommend you play with the lab though – it's instructive to see what fixes issues and what doesn't.

Known Issues with "The Best App Ever"

Open the Starting Solution

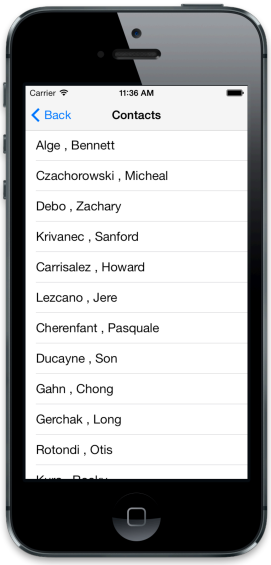
1. Launch Xamarin Studio and open the starting solution in the begin folder – Lab3.TheBestAppEver.sln.
2. Run the program – preferably on a real iOS device. It will show a table view with three choices:



3. Each item corresponds to a specific view controller, which has an associated, view model and model. Each also has at least one common performance issue.

Contacts (TODO: Step 1a)

4. Try the Contacts item first. It displays a list of contacts as shown below:



5. However, if you try to scroll the contacts, you will find it scrolls quite slowly – pausing and scrolling in a "jerky" fashion. It is almost a certainty that if you deployed this app, Apple would either reject it outright, or you would have several complaints from users and poor reviews. The problem lies in the data source for the table view, this line in particular from ContactsViewController.cs:

```
var items = Database.Contacts.Table<Person> ();
```

6. Look at the data type being returned – can you tell why this would be slow, particularly when the TableView has to pull items towards the middle end of the data? See if you can fix it.

Songs (TODO: Step 2a, 2b)

1. Next, we have a songs list. This pulls a list of songs from the same JSON file you used when we were playing with the async / await support, and if you look at the web service class used here, you'll see our fixed code in place.

   However, this application is also very slow in scrolling and updating – if you launch it, it will download the JSON data and parse it out asynchronously, but then begin to load the UI (which must be done on the UI thread). While it's loading the data, it will be painful to scroll it on the device. In addition, the progress bar will update fairly slowly.

Hint: watch the output window to see the update activity.

ToDo (TODO: Step 3a – 3e)

1. Finally, we have a ToDo task list. This also scrolls poorly on the device, for several reasons. In addition, if you pull the list down, it will refresh the data on the screen; which will also slow everything down.

Hint #1: look at the contents in the cell, and specifically how the cell is loaded and setup in the **TodoItemCell** class.

Hint #2: look at the **TodoViewModel**, which provides the data, it, raises events to notify the UI that things have changed. How often is it doing it?

Summary

Congratulations on working through some real world (although simple) code which has performance issues! You can look at the pre-canned solution in the completed project but keep in mind that there are multiple ways to solve the issues in this project so the completed version isn't a definitive solution – just an example way to solve some of the issues we have.