Assignment – View Controllers

OVERVIEW

Date Due: Thursday, March 25th (**after break**) by 11:59 p.m. EST. ✧ **Value:** 40 points

Features

In this project, the ToDo application becomes a real usable application. In doing so, you will gain experience working effectively with iPhone view controllers. You will also find out what it takes to make a polished application that handles any curves a user can throw at it.

This embodiment of the ToDo application has the following features.

- Categories
    - Each event can be given a category
    - Events are browsable by category
- Event List
    - Displays events ordered and sectioned by date (you've done this before)
    - Events can be deleted from the event list
    - Events can be edited. Selecting an event brings up a modal editor.
    - Events can be added. Tapping a plus button brings up the modal editor.
- Modal Editor
    - Users can specify and change events properties from this editor
    - Changes should be reflected in each and every displayed view when editing is complete
- Saving
    - Your event "database" should be saved
    - Re-running the application will load in the previous results

You will be given some code and resources that are not related to the goal of learning how to use view controllers. The reference application will be made available so you can play with it to see what it implements and how it behaves. You aren't expected to provide every bell and whistle it has, instead it is meant to serve as a guide to show a full application.

Grading

Writing a complete, and well polished application will require a fair amount of effort. At a minimum, you are expected to implement the features outlined above. The UI behavior, and polish are secondary to the features listed above being implemented and working correctly. After grading feature completeness, application polish will be taken into consideration along with the code architecture. Make good use of MVC and data passing strategies discussed.

Finally, have fun and add any bells and whistles you want! For example customize the UI to match your style. give the application a toolbar with a "today" button that quickly jumps to today's date, or a segmented control to show / hide completed events. Another idea, is to wait to enable the "save" button until the user has changed any values. Explain the additional features you added at the beginning of your README file. **Value**: up to 5 points

SCREENSHOTS

The following screenshots show the main functionality in the application you are to implement.

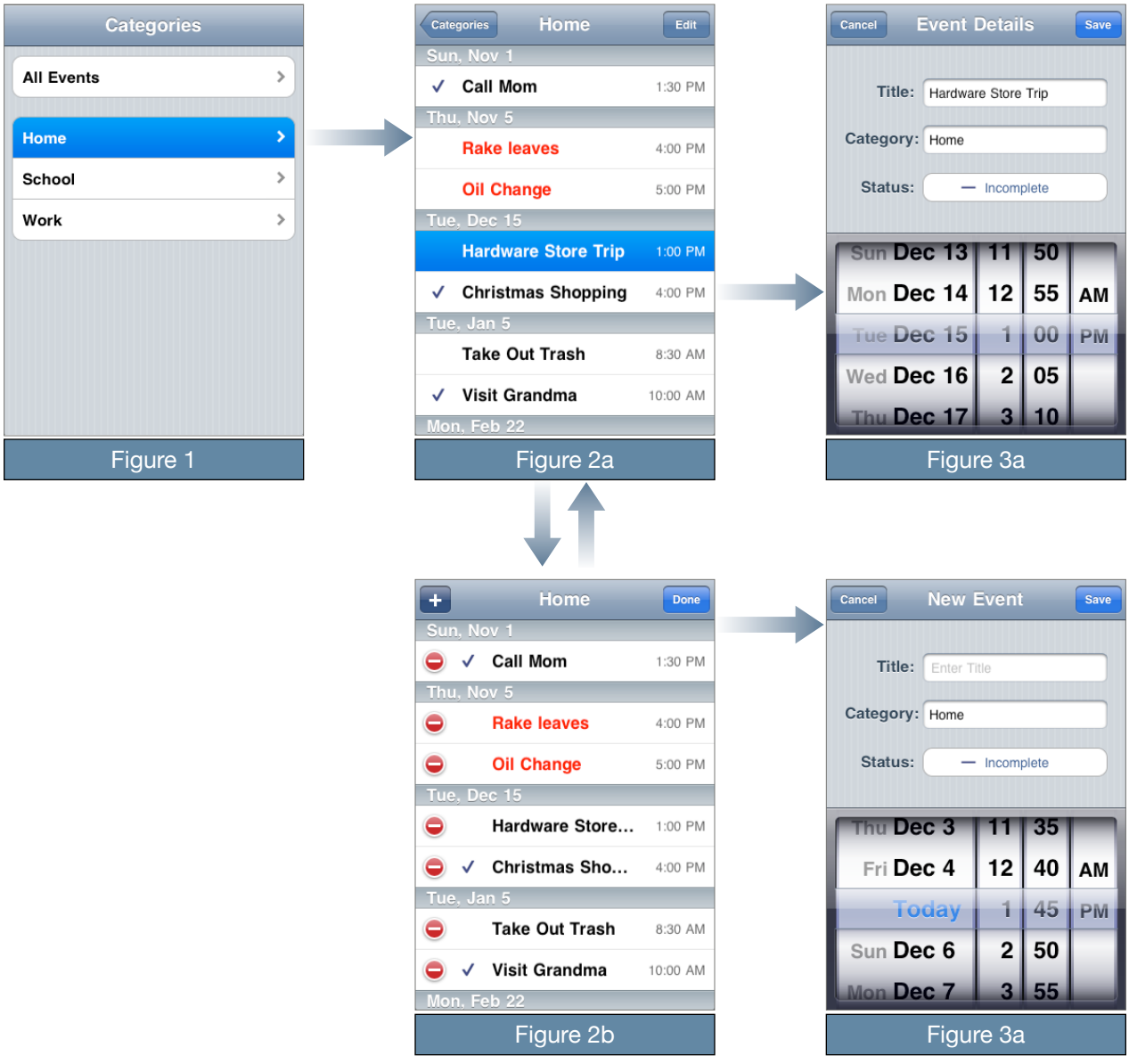

Figure 1 shows the categories listing. Selecting an item in this view takes you to Figure 2a, the event list view. This view presents dates matching the selected category, ordered and sectioned by date. At this point, the user may select an event to edit it (Figure 3a), or bring up the list editing UI (Figure 2b). While in list editing, may delete an event, or choose to add a new item by tapping the "+" button. Transitioning from Figure1 to Figure2 animates by sliding the content in from the right. Transitioning from Figure2 to Figure3 animates by sliding up a modal view controller.

NOTES

Resources

You will be provided starter resources: http://www.cs.umd.edu/class/spring2010/cmsc498i/files/lab5_resources.zip

Basics - Drag these files in to your project and use them where necessary

- FoundationAdditions.[hm] provides a helper function you need for determining where to save data
- NSDateAdditions.[hm] provides functionality needed to display and compare dates.
- ToDoListViewCell.[hm] and ToDoCell.[hm] provides the cells you wil use to display dates in the Event List.
- ToDoEvent.[hm] implements the ToDoEvent model you now know and love! It has 2 additional properties in this lab, a `category` property, and `uniqueID` property.

Data Storage - Use these, or your own event data store and sectioning logic from prevoius labs if you like

- ToDoEventSectionData.[hm] provides section grouping functionality. You can use your own section grouping logic from the previous lab, or if you prefer, use this one.
- ToDoEventStore.[hm] provides storage for a set of events, and is required by ToDoEventSectionData.[hm].

Other Resources

- CompletionState-*.png provide some icons for you to use when representing completion state in the editor.
- SampleEvents.plist provides a sample property list. If you like, place it in your app bundle to use for testing.

The provided code listed above should be usable as if it came from a library. That is, you probably don't need to touch it, and will just use it as a client. However, depending on how you implement things, you could save yourself some time if you add minor changes or features to them. Point being, you can use this code as is, but don't have to do so.

Read the header files and figure out how you will use them in your application. Please give a pass through the implementation of these as well to see the techniques employed. The more Objective-C / Cocoa code you read, the better your own code will be. You will start to recognized standard styles and recurring patterns that will be useful.

Suggested Approach

This week, we give even less instructions with the assumption that you are getting more comfortable figuring out things on your own…

It is advisable to begin by creating all the view controllers in your application and wiring them together. Start by making a navigation based application. Fill out all of the controllers with fake, hard coded data while in this stage, just enough so that you can verify that navigation to every screen works. While you are doing this, be thinking about where data eminates and how it flows around the application. Do you need to define some delgate callbacks? notifications of your own? Make use of the notifications in the provided code too. This is a good time to define the data passing APIs.

Once you have your UI flow working, define IBOutlets / IBActions for the UI. After hooking these up, its now time to pass real data around your application. Get one screen working at a time.

Once you have every screen working, spend some time on polish and testing. Make sure everything work and there are no edgecases you forgot.

NOTES

Saving Documents

When saving documents, use `NSUserDocumentDirectory()` as the directory to put your file into. If you step through gdb and look at what `NSUserDocumentDirectory()` is returning (in FoundationAdditions.m), you may be surprised. You'll see that the documents directory is something like /Users/username/Application Support/iPhone Simulator/User/Applications/D713AFE6-D6B3-4D1E-A1B9-28FD679FD124/Documents. And after each launch, the unique ID in the "/Applications/.../Documents" part changes. Don't worry, between launches, Xcode will copy your documents from the previous location to the new location. The uniqued Documents location is within what is called your "sandbox" area. We'll talk more about this later in the semester. If for some strange reason, Xcode isn't copying files between sandboxes for you, look in `NSUserDocumentDirectory()` for a section of code you can uncomment to force saves to always go to the same location.

Dealing With *nil* Values

String – often you will find Cocoa APIs return an empty string (@"") instead of nil if no string has been set. To insulate yourself from this detail, it can often (it depends on the context of course) be beneficial to check [string length]==0 instead of a check for string == nil.

Objects – For the most part, you don't have to worry about whether or not you are working with an actual object instance or nil. For example, it is usually fine to message nil, and to pass nil as a parameter to some APIs. Normally, when messaging nil, the return value will be 0. However, if the return value is not a simple value, and is instead something like a struct, the return value is undefined. Passing nil can lead to a crash in some scenarios. For example, NSArray's addObject:nil, and NSSet's addObject:nil will lead to a crash. When reading API documentation, be on the look out for information on how nil is handled.

Table Delegate / Data Source Notes

During some table callbacks, UITableView will have problems if you call [tableView reloadData]. Be careful not to call -reloadData when it isn't necessary. For example, while responding to the deletion delegate callbacks, you should not call reloadData. Doing so, could cause UITableView to get out of sync enough to cause a crash due to an exception raise.

Laziness

Recall that UIViewController lazily calculates the view property. This is a good indication how much keeping memory usage down is to the system and your application. Whenever possible, you should try to be lazy. For example, when data changes and you need to recalculate a cache. One approach here is to release the cache and set it to *nil*. If all of your code access the cache through a single method, you can make that method create the cache on demand so that the data is not recalculated until it is really needed.

UITextField, Keyboard, and Editing

The keyboard is automatically shown when a text view like UITextField is made the windows first responder. Tapping in an editable UITextField will automatically make it the first responder. By default, when wiring a UITextField to an action, the action will be configured to be sent on "end editing", which happens when the keyboard is dismissed. So, how do you know when to dismiss the keyboard. In upcoming weeks, we will discuss text editing more. For now, follow these steps to make sure your text fields send there actions when you expect, and give a good user experience.

- In IB, or Code configure the UITextField's delegate to be an object you own

- In the UITextField's delegate, add the following code:

```
-  (void)finishActiveEdits {
    // Call "resign" on all of your text fields...
    [titleField resignFirstResponder];
    [categoryField resignFirstResponder];
}
```

- When the modal view controllers "Done" action method is called, you should first call `finishActiveEdits`. This will make sure any editing in progress fires its action.

- In the UITextField's delegate add the following code. It will make sure that the text field resigns its first responder status when the keyboard's return button is tapped.

```
- (BOOL)textFieldShouldReturn:(UITextField *)textField {
        [textField resignFirstResponder];
        return YES;
}
```

One more thing to watch for with UITextField. When you set up a UITextField using IB, the default properties will cause UITextField to clear the entire text when a user begins editing. That is, a text field that is showing the string "Home" will immediately become empty immediately when the user taps in the text field. In some applications, this makes sense, but we don't want this. To turn off this behavior, go to IB and uncheck the "Clear When Editing Begins" checkbox.