**Project 2: Temple Flashcard App**

**Objective**

In this assignment you will practice using collection views and table views; you will also practice creating view animations and custom views; finally, you will use Autolayout to make your app run nicely on any size iPhone, but only in landscape orientation. The main task of this project is to write a fun temple flashcard app that lets users study the temple photos and try their hand at matching photos to names.

**Required Tasks**

1. Write a flashcard app that displays temple photos in a collection view, temple names in a table view, and lets users select a temple photo and the corresponding name of that temple.
   * Alternative: you can choose a technique other than using a collection view or table view if you’d like. For example, you could decide to display temple photos one at a time, using a single view for the photo and a table view for the list of temple names. Or you could use a single view for the temple photo and a stack of buttons with a limited number of temple names for the user to choose from.
2. Display a border of some kind around each temple photo. The point is to dress up the temple photo and make it look nice. Think about different elements that would make the border look good.
3. When the user makes a valid match, remove those elements from the corresponding collection and table views.
   * Alternative: if you’re not using a collection view and/or table view, you’ll need a way for the user to advance to the next step in the game. For example, tapping on the correct name could advance to the next temple, but tapping on an incorrect name could mark that name as wrong and let the user try again. Or maybe you just tell them they guessed wrong and move to the next temple. You decide on the game flow you would like.
4. Keep track of how many correct and incorrect matches the user makes, and display their current correct/incorrect tallies.
5. Display a “correct” or “incorrect” message when the user attempts a match.
6. Provide a “reset” button that clears the tallies and loads all photos and temple names like it was when the app first launched.
7. Also include a “study” mode that does not display a separate list of temple names, but instead lists temple names on top of the corresponding photos. This mode should only display a collection view of temple image cards that the user can scroll through so he or she can study the temple photos. The user shouldn’t be able to select photos in this mode.
   * Alternative: if you’re not using a collection view to show temples, you’ll need a mechanism in the UI for advancing forward (and perhaps backward) through the list of temples to study them. This could be a button, a swipe gesture, or some other technique. You decide.
8. Display all temple photos at a fixed height, but preserve the image perspective. So different photos might have a different width.
9. Do not use a tab bar controller to move between “study” and “match” modes. Instead, provide a button (e.g. labeled “Study” while in match mode, or “Match” while in study mode) that reconfigures your MVC. Provide smooth animation of the UI configuration changes. In study mode the reset button should be disabled and there should be no separate list of temple names.
   * Alternative: if you’re not using a collection view, make a reasonable interpretation of this requirement, and determine how your study mode is going to work. Make smooth animations between different UI configurations.

Show your app to at least one other person and get his/her feedback on the user experience you have created. How would they suggest you improve the user experience? I encourage you also to share a meaningful temple-related experience you’ve had. I’ve given you a particular set of temple images. Also include in your app images of any temples that have special meaning to you. You can find temple photos on LDS.org. In your README file in the root of your project folder, include a report on these items.

How to turn this in: rename your top-level project folder “Project 2 Lastname Firstname” using your own last and first names. Then right-click and compress the folder to “Project 2 Lastname Firstname.zip”. Upload that zip file to Learning Suite.

**Alternative Requirements**

If you would prefer to do a different project, you could instead make a card game based on the Rook Card demo series. For example, you could make a game of Hearts that uses Rook cards instead of standard playing cards, or you could choose any other simple card game you’d like. You could create your own alternative card style if you don’t want to use Rook cards, but you are welcome to use the code from the Rook Card demo series if you’d like. To support multiple players, you will need to have some mechanism that lets the user indicate that they’re done with their turn so the UI hides their cards; then they hand the device to the next person who indicates that they are ready to begin their turn, so it shows their cards; and so on. One easy way to think about dividing up the cards is to have different sections of a collection view represent different groups of cards, like those that are visible face-up in the center of the table, those for each player, and so on. If there is a draw pile, it could be represented by a face-down card in the “center of the table” section, while other cards that have been played are shown face-up in that section. Be creative, and make a nice user experience. It may be a bit too ambitious based on what I’ve taught so far to manage a card table where the cards are laid out more realistically like a real card table. I suggest using a collection view if you choose this card-game alternative for Project 2.

**Hints**

1. When you start this project, select iPhone, not iPad and not “universal”. Be sure to disable all portrait orientations for your app, but support both landscape orientations. This will make it easier to control the variations in UI you have to support. You’re welcome to write a universal app if you prefer, but be aware that it takes a bit more Autolayout skill.
2. I’m providing a list of temples and photos I’ve downloaded from LDS.org/media-library. You’re welcome to expand the list and get more photos from LDS.org. You can also trim the list I’ve given you. This list is a bit out of date. You can get the latest directly from LDS.org.
3. Remember to adhere to good MVC architecture principles. Start by sketching what you think your UI will be, then design the model, then populate your model with photos and temple names, and then build your storyboard and view controller.
4. For required tasks #2 and #7, you will need to create a custom subclass of UIView, like I showed for RookCardView. Feel free to use my code as a guide. Be creative and try to make the photo collection look aesthetically pleasing. Don’t just slap up a 1-point solid black border and call it good.
5. For required task #7, think about how to make your text visible on top of any image. The closed captioning features of televisions might offer some inspiration on this matter.
6. Don’t forget to create an appropriate outlet for the custom views in your collection view. A common mistake is to forget to connect the outlet, and then when you’re trying to configure the cell, you end up with a null pointer.
7. Also don’t forget to assign your view controller as the data source and delegate for both your collection view and for your table view. This is another common mistake. You may need to spend some time reading the Apple class reference documentation for those protocols (UICollectionViewDataSource, UICollectionViewDelegate, UITableViewDataSource, UITableViewDelegate). You’ll want your view controller to implement all four of these protocols. Perhaps Swift extensions could help you organize the implementation of all these protocols more effectively.
8. There’s a UICollectionViewDelegateFlowLayout protocol that will let you tell the UICollectionView how big each cell should be. When you’re displaying images of different sizes, you will probably want to implement this protocol’s method:

func collectionView(\_ collectionView: UICollectionView,

layout collectionViewLayout: UICollectionViewLayout,

sizeForItemAt indexPath: NSIndexPath) -> CGSize

Just look up the image for the given index path (indexPath.row is the 0-based index), and use that image to calculate the proper rectangle size of the desired height.

1. A helpful widget in the iOS library is UIToolbar. You can drag UIBarButtonItems of several types into the toolbar: buttons (which can have either a custom text title or a built-in icon type), fixed space, and flexible space. A button that has no associated action in your Controller is a lot like a label. Yes, you can still tap it, but it does nothing other than acknowledge the tap by animating a fade cycle. You can create outlets and actions for UIBarButtonItems just like you would for regular UIButtons.
2. Remember the following pattern for animation of Autolayout constraints:

view.layoutIfNeeded()

// Make desired constraint changes here

UIView.animateWithDuration(0.5, delay: 0,

options: UIViewAnimationOptions.CurveEaseInOut,

animations: {

// put non-constraint view adjustments to animate here

self.view.layoutIfNeeded()

}, completion: { (Bool) in

// events to perform on completion of the animation

}

)

**Evaluation**

In all of the assignments, writing quality code that builds without warnings or errors, and then testing the resulting application and iterating until it functions properly is the goal.

Here are the most common reasons assignments are marked down:

* Project does not build.
* Project does not build without warnings.
* One or more Required Tasks was not completed correctly.
* A fundamental concept was not understood.
* Code is sloppy and hard to read (e.g. indentation is not consistent, etc.).
* Your solution is difficult for someone reading the code to understand due to lack of comments, poor variable/method names, poor solution structure, etc.
* UI is a mess. Things should be lined up and appropriately spaced to “look nice.” Xcode gives you those dashed blue guidelines so there should be no excuse for things not being lined up, etc. Get in the habit of building aesthetically balanced UIs from the start of this course.
* Assignment was turned in late.

A common question is, “How much commenting of my code do I need to do?” The answer is that your code must be easily and completely understandable by anyone reading it. You can assume that the reader knows the SDK, but should not assume that they already know the (or a) solution to the problem.

**Need More of a Challenge?**

Here are a few of ideas for some things you could do to get some more experience.

1. Allow the user to view a larger temple photo by double-tapping a photo in the collection view. Be sure to smoothly animate any UI changes.
2. Learn about the iOS settings mechanism, and let the user choose a photo size (small, medium, large) to control how many photos appear on the screen at a time in the collection view.
3. Display the user’s high score in the iOS settings app.
4. Make a nicer (and HIG-compliant) launch screen.