iOS Dev Accelerator
Week 3 Day 4

- Custom Transition Demo
- WebView
- Regex and input validation
- Swift Extensions
- App Ideas

Custom View Controller Transitions

Custom View Controller Transitions

- Debuted with iOS7
- Gives you complete control over the animation while you transition from one VC to another.
- You can even create interactive transitions that are driven by gestures.

Workflow

1. Create an animation controller - This can be any class that conforms to UIViewControllerAnimatedTransitioning protocol. This class will perform the actual animation.
2. Set the transitioning delegate - When you are about to present a view controller, you need to set its transitioning delegate. This is usually just the presenting view controller.
3. Return the animation controller when the transitioning delegate is asked for it.

Animation Controller

- Your animation controller can be any class that conforms to UIViewControllerAnimatedTransitioning protocol
- This protocol has 2 required methods:
  - transitionDuration(transitionContext : UIViewSearchControllerContextTransitioning)
  - animateTransition(transitionContext : UIViewControllerContextTransitioning)

`transitionDuration()`

- A simple method that will return how long the transition will take.
- This should always match up with the total duration of the animations you create in the next method.

animateTransition()

- This method is where you actually implement the animations.
- It has one parameter passed in, the transitionContext.
- The transitionContext gives you access to the both the presenting view controller (fromViewController), and the presented view controller (toViewController:

```
 //getting both the view controller we are presenting from (fromVC) and the one we are
presenting (toVC)
 let fromVC = transitionContext.viewControllerForKey(UITransitionContextFromViewControllerKey)
 let toVC = transitionContext.viewControllerForKey(UITransitionContextToViewControllerKey)
```

animateTransition() cont.

- The transitionContext also provides you with something called the containerView
- The containerView essentially acts as the super view for both the presenting view controller's view and the presented view controllers view.
- UIKit automatically adds the view of the presenting view controller, think of that as your starting state of the transition.
- You then add the view of the toVC to the containerView, and then animate it moving onscreen in some cool way
- Once the animation is complete, be sure to call completeTransition() on the transitionContext.
- You don't have to clean up the containerView, its all done for you once your animations are complete by calling the completeTransition()

TransitionDelegate

- The transitionDelegate is mostly just responsible for providing the animation controller at the appropriate time.
- It does this by implementing this method:

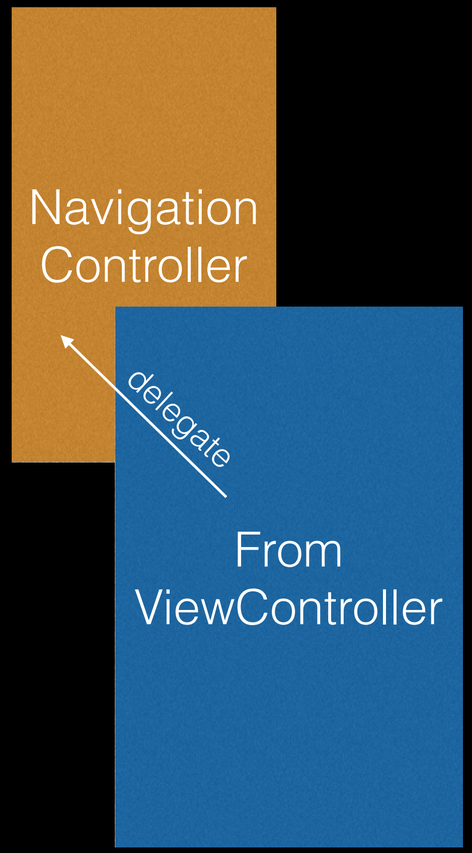
```
func animationControllerForPresentedController(presented: UIViewController, presentingController
 presenting: UIViewController, sourceController source: UIViewController) ->
 UIViewControllerAnimatedTransitioning? {
    return self.animationController
}
```

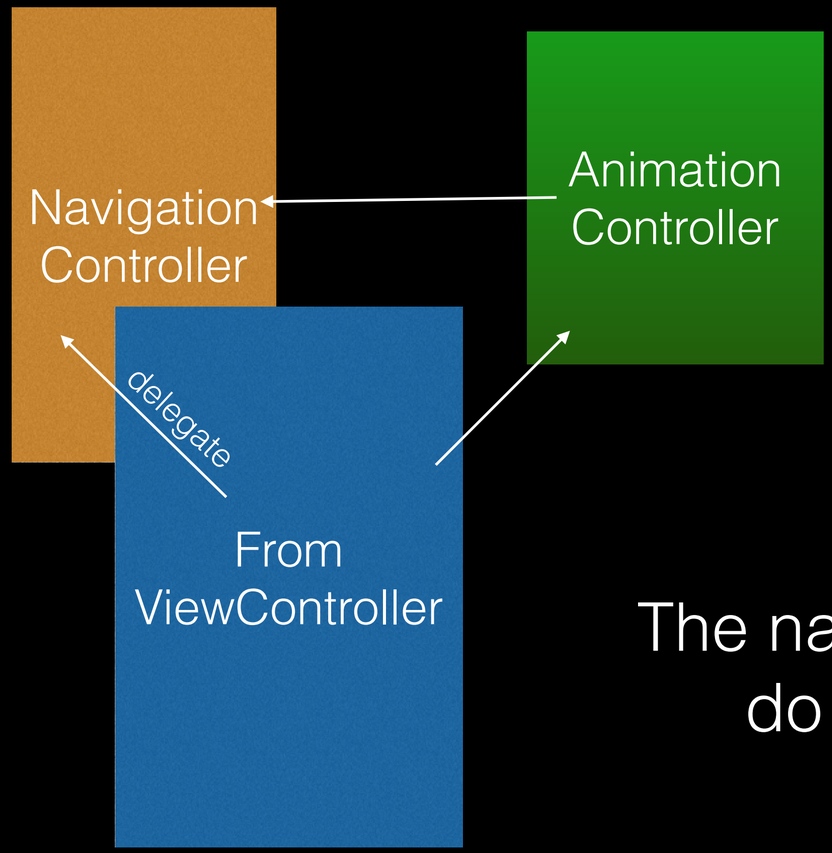
- If you have multiple transitions/segues wired up to this view controller, you would need to inspect the presented view controller to see which one is being presented on screen, and then return the appropriate animation controller

With Navigation Controller

- You can easily create custom transitions for view controllers inside of a navigation stack as well.
- In this case, the navigation controller's delegate will provide the animation controller.
- UINavigationControllerDelegate has a method you will need to implement in order to give it to the animation controller at the appropriate time:

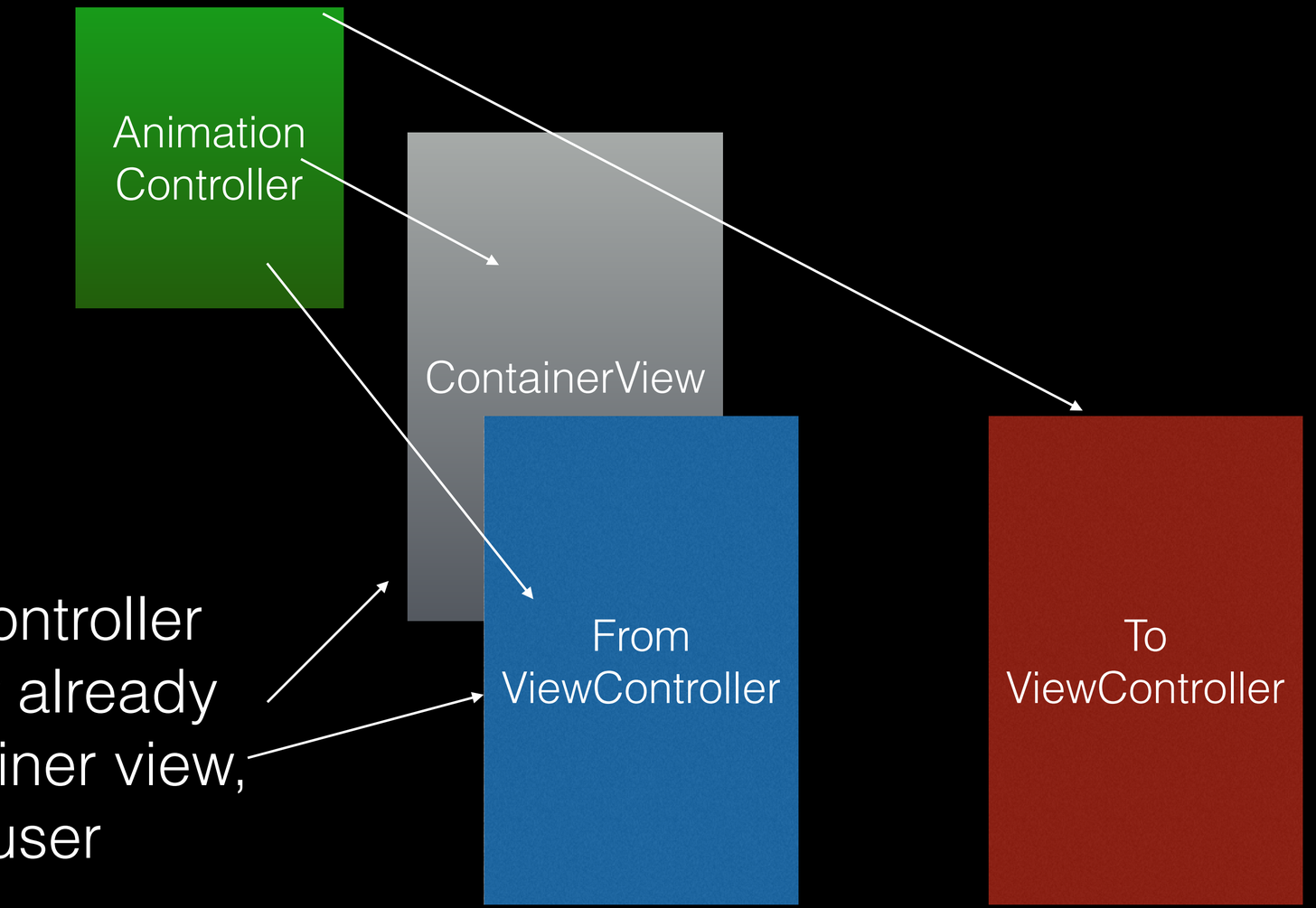
```
//MARK: UINavigationControllerDelegate

func animationControllerForPresentedController(presented: UIViewController, presentingController
    presenting: UIViewController, sourceController source: UIViewController) ->
    UIViewControllerAnimatedTransitioning? {
    return self.animationController
}
```


Navigation
Controller
delegate
From
ViewController



The navigation controller sees its about it to do a transition and asks its delegate for an animation controller to animate the transition

The animation controller takes over from here.
The transition context provided by it has
references to the To and From VC and also the container
View where the transition will take place



The From View Controller
starts with its view already
added to the container view,
visible to the user

Demo

WKWebView

- “Use the WKWebView class to embed web content in your application”
- The simple workflow of a web view:
    1. add web view to the view hierarchy
    2. send it a request to load web content
- Can have a delegate that tracks loading of content, this will come in handy when we do the ‘in-app’ way of doing OAuth
- It’s sort of like a mini browser in your app, and you can customize it to not allow the users to go back or forward.
- Prior to iOS8, we had to use UIWebView, and it was horrible. It leaked memory every time you used it.

Demo

Regex and Input Validation

Input Validation

- “An app sometimes cannot accept the strings entered in textfields and text views without validating the value first”
- In our Github app, we cant have spaces in user’s search queries. We also don’t want them entering in symbols like = or $.
- So how can we ensure our users input doesn't break our app?

Delegate methods on input views

- The go to methods for text validation are:
  - For UITextField : textFieldShouldEndEditing:
  - For UITextView : textViewShouldEndEditing:
  - For UISearchBar : searchBar:shouldChangeTextInRange:
- These methods are called just before the textview/field is about to resign first responder or for the search bar, when a character is about to be entered.
- Returning false prevents that from happening. So return false when the text isn't valid. Also perhaps pop up an alert view or show some indicator explaining why its invalid.
- So how do we check if the text is valid?

Regex

- Regular Expressions are a pattern-matching standard for text validation.
- The regular expression is a pattern that you compare with the text you are validating.
- Regex’s can be used for finding patterns, replacing text, and extracting substrings in strings.

The simplest Regex

- The most simple regex is just a string. Like “seahawks”.
- The regex “seahawks” will find a match on the string “go seahawks”
- The regex “sea hawks” will not find a match on the string “go seahawks”
- In these two examples, we are only using literal characters. So its basically just running a find operation looking for our regex string.
- There are special reserved characters we can use in regex to make it much more powerful

+ and *

- + is used to match the previous character 1 or more times
- so the regex “sea+” will match sea and seaaaaaaaaaa and seaaaa
- * works the same way, except it matches 0 or more times
- so “sea*” matches seaaaaaa, sea, and also se

(Capturing parens)

- Parentheses are used to create a capturing group.
- Capturing groups capture the text matched by the regex inside the parens and put them into a group that can be referenced together.

? optional

- The question mark makes the preceding token in the regular expression optional.
- So “seahawks?” matches both “seahawks” and “seahawk"
- You can use grouping to make groups optional
- So “sea(hawks)?” matches seahawks and sea

[Character Classes]

- With a character class, sometimes called character set, you can have the regex only match one of several characters.
- If you want to match a t or d you will use [td]
- so “foo[td]” will match “foot” and “food”
- You can use a hypen inside the character classes to specify ranges
- “[0-9]” matches any single number
- You can combine ranges “[0-9a-zA-Z]” will match any regular literal character

[Negating Character Classes]

- Adding a caret after the opening square bracket negates the character class.
- This makes it so the character class matches any character that is NOT in the character class.
- So [^0-9a-zA-Z] will match any non regular literal character.
- We also need to add \n in that range because that is what is entered into a textfield/view/searchBar when you hit return/done/search. \n is ascii/unicode for end of line.
- Hey I think we can use [^0-9a-zA-Z] in our app!

Regex and iOS

- You can use the NSRegularExpression class to natively use regex in your app.
- You instantiate an instance of NSRegularExpression with your regex pattern, options, and an error pointer:

```
let regex = NSRegularExpression(pattern: "[^0-9a-zA-Z]", options:
    nil, error: nil)
```

Search for matches

- NSRegularExpression has methods for returning the total count of matches, enumerating through each match, returning an array of matches, returning the first match, and the range of the first match.
- In our app we can just use the number of matches method, and if its greater than 0 we know the user typed in something invalid:

```
let regex = NSRegularExpression(pattern: "[^0-9a-zA-Z]", options:
    nil, error: nil)
let match = regex.numberOfMatchesInString(self, options: nil, range:
    NSRange(location:0, length: countElements(self)))
if match > 0 {
    return false
}
return true
```

Demo

Swift Extensions

Extensions

- Extensions add new functionality to pre-existing classes, structs, or enums.
- You can even do this on types you do not have access to the source code.
- Similar to categories in Objective-C, except extensions dont have names.

Things you can add with extensions:

- computed properties
- instance methods
- type methods
- new inits
- subscripts
- nested types
- make an existing type conform to a protocol

Extension Syntax

```
extension SomeType {
    // new functionality to add to SomeType goes
        here
}
```

Demo

Using Git with a Team

Setting up your repo

1. **Don’t use Github’s app!!!!!!**
2. Init a git repo inside it’s directory
3. Setup a .gitignore and place it in the directory
4. Stage the .gitignore FIRST, with ‘git add .gitignore’
5. Stage & commit the gitignore
6. Create your project in Xcode
7. Setup your remote repo on github, add it as a remote in your local git repo, and then push it up

So what should your .gitignore contain?

.gitignore

- There are a number of files generated by Xcode that you are going to want to put in your .gitignore in order to have a smooth source control experience
- The easiest way to setup your .gitignore file is to go to https://www.gitignore.io and have it generate a Swift .gitignore for you

Getting others on your repo

1. On github, the original owner needs to add all of the team members as collaborators (they now have read/write access)
2. Now the collaborators can clone the repo down to their local machine and have the ability to push changes back up
3. The clone command automatically hooks up the remote repo as origin
4. Begin working

Git team workflow

1. When you are going to start work on a new feature, create a new branch
2. Do your work in that branch while periodically pulling from master to ensure your code works with the latest changes to master
3. When you are ready to push your changes to master, do one final pull from master to resolve any merge conflicts.
4. Push up to your remote feature branch, and then initiate a pull request to master
5. Have someone review the changes, and then accept or reject the pull request
6. Rinse and repeat

Do your pushing and pulling in Xcode, since Xcode has a great merge tool

Merge Conflicts

- Occasionally the merge process wont go as smoothly as we think it will.
- If you changed the same part of a file on the two branches you are merging, this will be a merge conflict.
- git will tell you theres conflicts in specific files, that the merge failed, and to fix the conflicts and then commit the results.
- Essentially git pauses the merge process until the conflicts are resolved.
- At anytime during a halted merge, you can run git status to see which files are still unresolved.

Resolving conflicts

- There are 2 ways to resolve the conflicts.
- Manually: Open each conflicted file and fix the conflicts line by line.
- Merge Tool: Use a merge tool that lets you choose which file's version of the conflicted code you want. This way is much less error prone. Xcode has a built in merge tool.

Manual Resolution

- Git adds conflict-resolution markers to the files that have conflicts.
- Heres what they will look like when you open them manually:

```
<<<<<<< HEAD
<div id="footer">contact : email.support@github.com</div>
=======
<div id="footer">
  please contact us at support@github.com
</div>
>>>>>>> iss53
```

- The <<<<< HEAD denotes this is the beginning of the code that our local HEAD branch contains.
- The ====== signifies of the end of HEAD's version and the beginning of the branch we are trying to merge from.
- Finally, the >>>>>>iss53 signifies the end of the version of the code branch iss53 had
- Once we get rid of all the conflict markers (<<<<,======,>>>>>) in a file, we are ready to mark this file as resolved.
- Run git add on the file to mark it as resolved. staging the file tells git the conflicts have been resolved.

Merge Tool

- You can use a merge tool for a graphical interface based conflict resolution process
- use the git mergetool command to fire up the appropriate merge tool
- opendiff is the default merge tool if you havent configured git to use a different one.

Xcode's pbxproj and git

- pbxproj is a file that is contained in your Xcode projects
- it manages the file structure of your project
- so anytime a team member adds, removes, or rearranges files in your Xcode project, pbxproj changes
- git cannot automatically merge pbxproj, and sometimes even its merge tool will crash Xcode while you are trying to resolve the merge conflicts
- If that happens, you will have to manually resolve the conflicts by opening the pbxproj and removing the conflict markers yourself

Demo