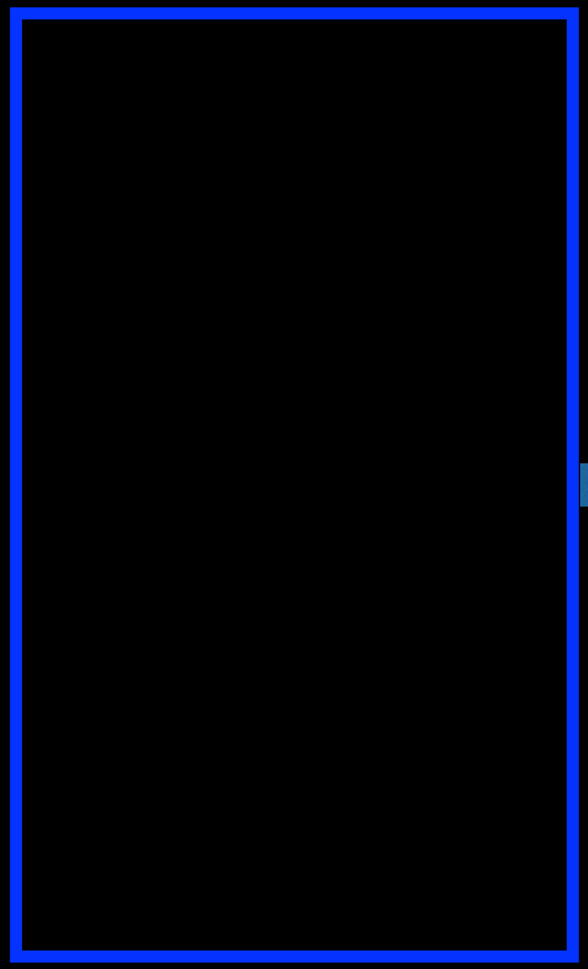
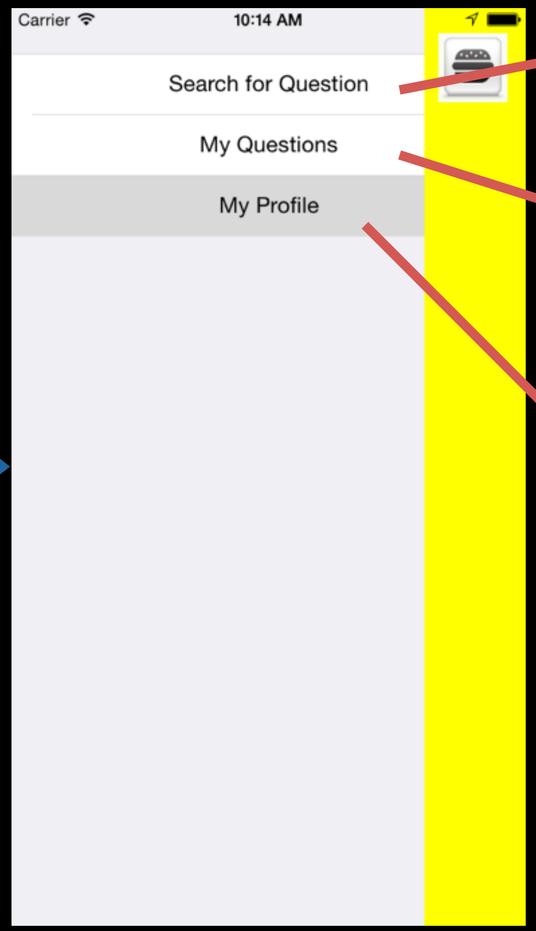
iOS Dev Accelerator
Week 7 Day 1

- Custom Container View Controller
- OAuth with WebViews
- NSError

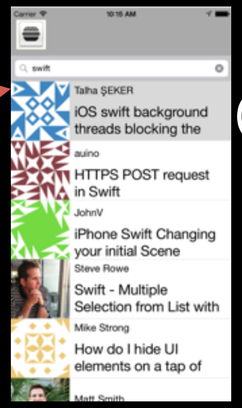
App Wire Frame



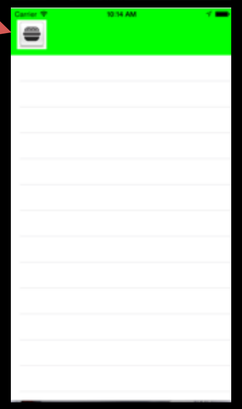
BurgerMenuViewController



MainMenuVC



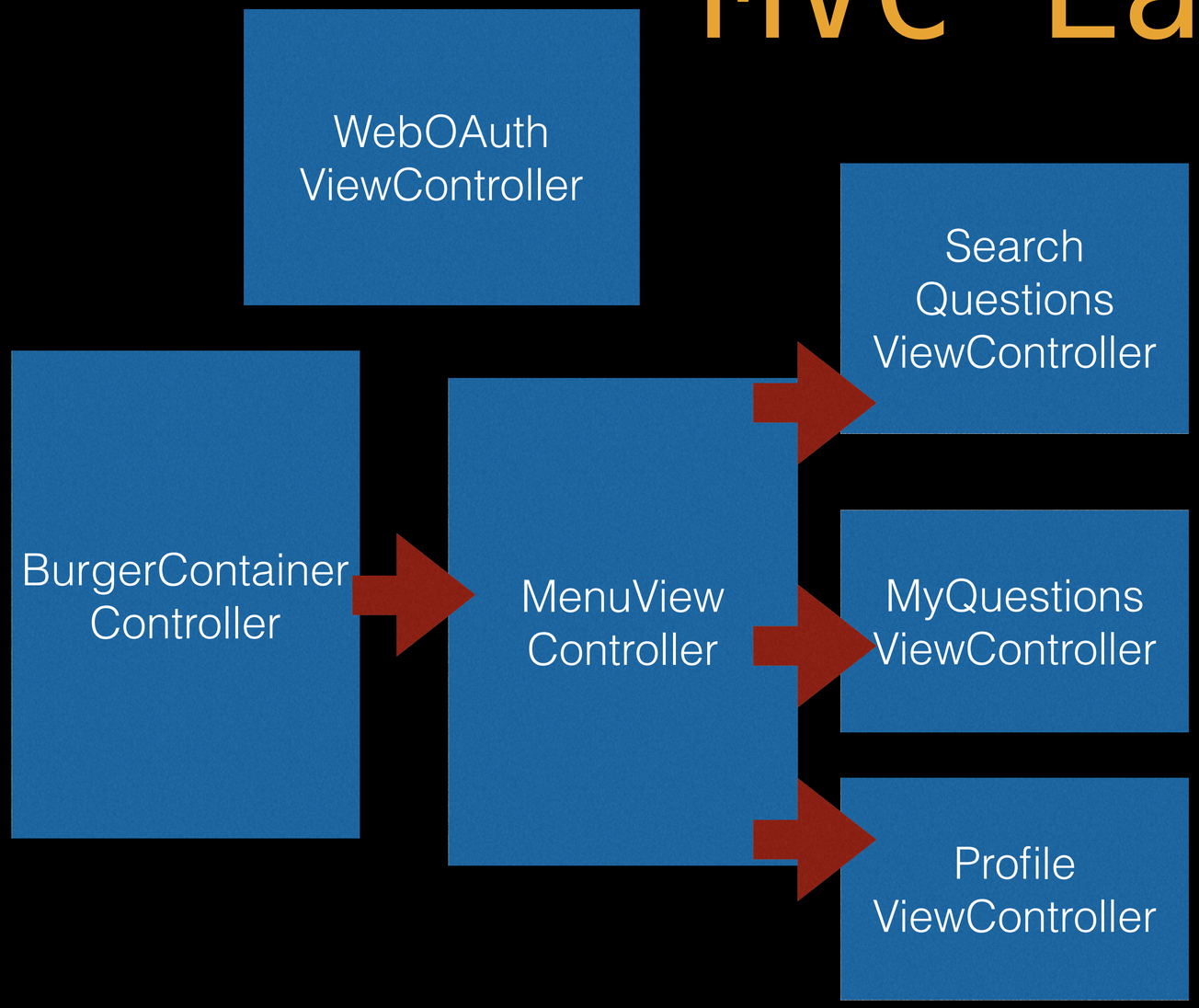
QuestionSearchVC

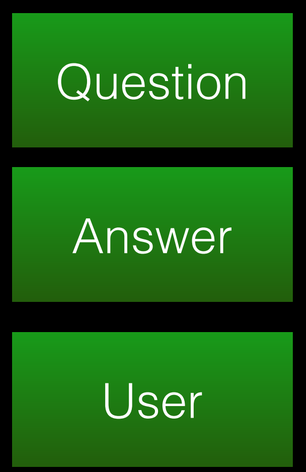


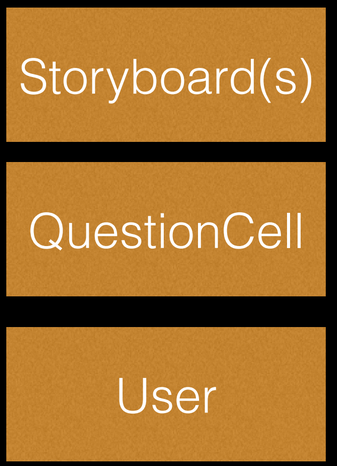
MyQuestionsVC

MyProfileVC

MVC Layout


WebOAuth ViewController
BurgerContainer Controller
MenuView Controller
Search Questions ViewController
MyQuestions ViewController
Profile ViewController
Controllers


Question
Answer
User
Models

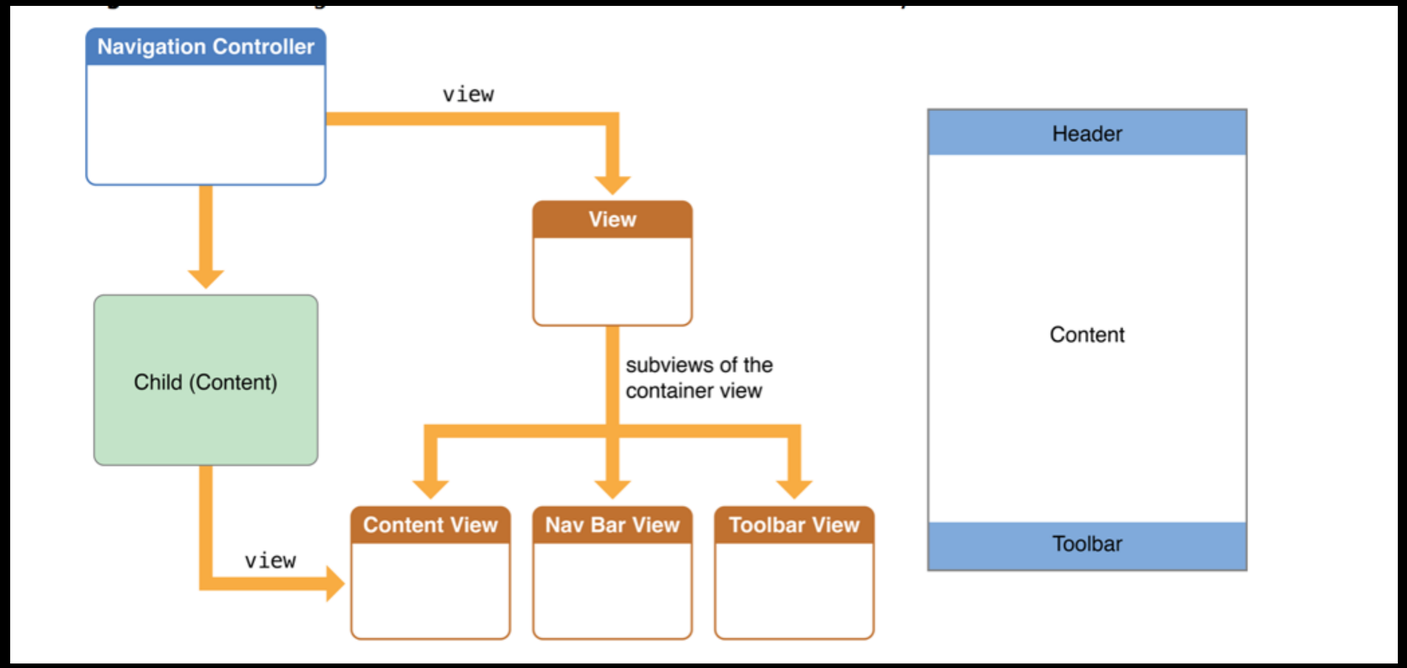

Storyboard(s)
QuestionCell
User
Views

Container View Controllers

- All of the view controllers you have created so far are considered ‘Content View Controllers’
- Container View Controllers are similar to Content View Controllers, except they manage parent-child relationships between the Container VC, the parent, and its Content VC(s), the children.
- The Container VC is in charge of displaying its children VC’s views.

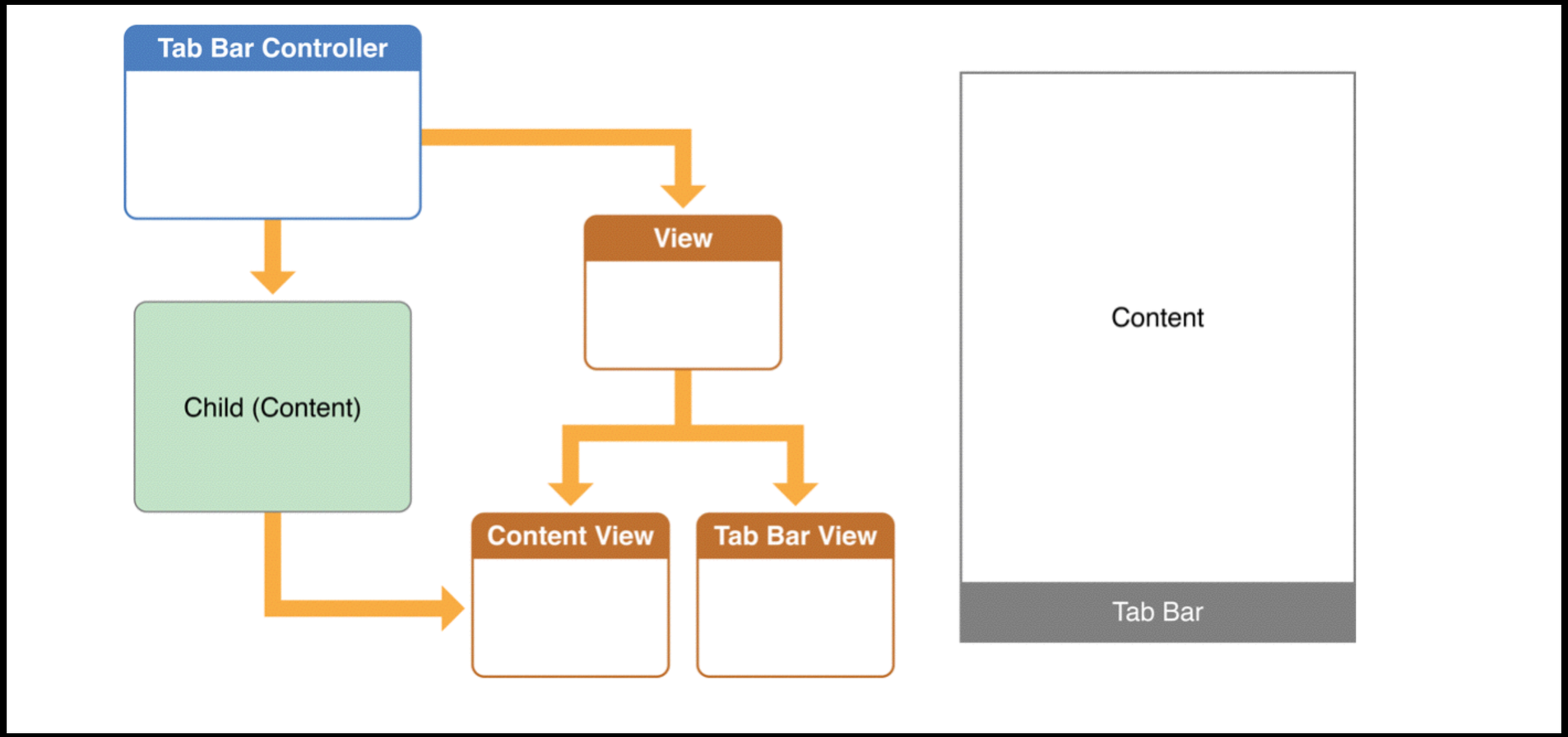
Container View Controllers

- An example of a Container View Controller you have used is the Navigation Controller:



Container View Controllers

- Same Idea with a Tab Bar Controller:



Container View Controllers

- “The goal of implementing a container is to be able to add another view controller’s view as a subview in your container’s view hierarchy”
- When a new VC is added on screen, you can ensure the right events (viewDidLoad, willAppear,etc) are given to all VC’s by associating the new VC as a child of the container VC.

Getting your child on screen

```
func setupChildVC() {
    //create a new VC
    var childVC = UIViewController()
    //tell this VC that we are adding a child VC
    self.addChildViewController(childVC)
    //set the child VC's view's frame, in this case it will
        completely cover the container VC
    childVC.view.frame = self.view.frame
    //add the child view to the parent view
    self.view.addSubview(childVC.view)
    //notify the child vc he is now on screen
    childVC.didMoveToParentViewController(self)
}
```

Taking the child off screen

```
childVC.willMoveToParentViewController(nil)
childVC.view.removeFromSuperview()
childVC.removeFromParentViewController()
```

Demo

OAuth with Webviews

OAuth with Webviews

- When we first learned OAuth, we learned the workflow that actually takes the user completely out of our app and into the web browser.
- Once the the user gives permission for our app, the service provider redirects the user back to our app via the redirect URL.
- This works great, but some web API's don't support mobile app style redirect URL's (appname:\\parameters), which means they they aren't able to redirect the user back to the mobile app.

OAuth with Webviews

- We can get around this by never leaving our app in the first place
- Instead of relying on Safari to show the user the web page that is generated by the service provider to grant permission for our app, we can use a web view.

WebView+Oauth workflow

1. Instantiate a WKWebView and become its delegate
2. Create your initial OAuth URL, which for most web API's will include your clientID and redirect URL
3. implement webView:decidePolicyForNavigationAction:decisionHandler:

webView:decidePolicyForNavigationAction:decisionHandler

- This method is part of the WKNavigationDelegate, and is called anytime the web view is about to load a URL
- We will use this method to listen for each request the web view loads, until the request is loaded that contains the oauth token.
- This method is essentially taking the place of the openURL: method in the app delegate we did for the first oauth workflow.

Special considerations for stack apps

- Make sure the redirect URI you pass in with your initial oauth url is “https://stackexchange.com/oauth/login_success”
- Make sure to put a check in the box “Enable Client Side OAuth Flow”
- Enable Desktop OAuth Redirect URI, which allows non web apps like ours to participate in the OAuth process.
- No need to have the client secret or key in your app at all, yay!
- And there is no 2nd post call we have to make after our initial request. Instead of giving us an intermediate ‘request code’, they gave us the full oauth token after the first request.

Demo

Lazy Loading in Objective-C

Lazy Loading

- During our swift month we learned lazy loading by lazily downloading images only when they were actually needed, instead of downloading them all at once with our initial API request
- You can also apply this same principle to your properties. Leave them uninstantiated until they are actually needed
- We know a property is needed when its getter is called
- This is much more common practice in objective-C simply because we can’t give our properties values inline with their declaration
- You can also do this in swift by simply marking your properties as lazy

Lazy Loading workflow

- Override the getter of any property you want to make lazy and you basically just do 2 things:
    1. first check if the underlying i-var is not nil. If it is, that means the property has already been instantiated, and simply return it.
    2. setup the i-var and return it (this part of the getter will only run if step 1 evaluated to false)

Lazy Loading workflow

1

2

```
- (NSManagedObjectModel *)managedObjectModel {
    // The managed object model for the application. It is a fatal error for the
      application not to be able to find and load its model.
    if (_managedObjectModel != nil) {
        return _managedObjectModel;
    }
    NSURL *modelURL = [[NSBundle mainBundle] URLForResource:@"HotelManager"
        withExtension:@"momd"];
    _managedObjectModel = [[NSManagedObjectModel alloc] initWithContentsOfURL:
        modelURL];

    return _managedObjectModel;
}
```

It is not uncommon to see ALL properties done this way

Demo

NSError

- Every NSError object encodes 3 critical pieces of information:
  - a status code
  - a corresponding error domain
  - additional context provided in the userInfo dictionary week

NSError code

- NSError's code signals the nature of the problem
- Each status code is defined within an error domain
- Status codes are usually defined by constants in an enum
- Heres some error codes from the NSPosixErrorDomain:

```
#define EPERM        1        /* Operation not permitted */
#define ENOENT       2        /* No such file or directory */
#define ESRCH        3        /* No such process */
#define EINTR        4        /* Interrupted system call */
#define EIO          5        /* Input/output error */
#define ENXIO        6        /* Device not configured */
#define E2BIG        7        /* Argument list too long */
#define ENOEXEC      8        /* Exec format error */
#define EBADF        9        /* Bad file descriptor */
```

NSError domain

- “For historical reasons, error codes in OS X (and iOS) are segregated into domains” - Apple Error Handling programming guide
- In addition to major error domains like NSPOSIXErrorDomain and NSCocoaErrorDomain, there are error domains for specific frameworks or even groups of/individual classes.
- For example, within the Foundations framework, URL classes (like NSURL) have their own error domain (NSURLErrorDomain)

Demo

NSError userInfo

- One of the main reasons Apple chose to design error handling as these NSError objects is so the objects can carry extra information
- This extra information is contained inside of a dictionary called userInfo
- There are 3 primary keys in the userInfo dictionary you will find quite useful:
  - localizedDescription -a localized description of the error
  - localizedRecoverySuggestion - a localized recovery suggestion for the error
  - localizedFailureReason - a localized explanation of the reason for the error

Creating your own NSError

- Easiest way to create your own NSError object is with the class factory method errorWithDomain: code: userInfo:
- Lets demo this

Dealing with Errors

- When dealing with NSErrors, you are doing it as either a consumer or a producer
- So far we have only worked with NSErrors from the point of view of a consumer.
- We have consumed an error that was given to us from someone else's framework or class (ex: Apple's NSURLSession or Parse's PFQuery)

Consuming an Error

- Errors usually come in 2 places:
  - As the final parameter to a method. Since Objective-C methods can only return one value, methods cant just return an error object because then the method cant return anything else. Thats why they do the weird parameter & thing.
  - As a parameter to a completion Handler. This is how errors are passed for async methods.

Demo

Producing Errors

- A rule of thumb: If your custom method invokes another method that produces an Error, your custom method should have an NSError parameter.
- These errors on your custom methods could simply forward the errors (if you get one) that you got from the method you invoked inside, or you can create your own custom error domain + error codes.
- For a large app/framework, custom error domains are a great way to simplify your error handling when you need it most (ie when its time to deliver bad news to the user).

Defining a custom error domain and codes

- To define a custom error, simply create a constant string
- Best practice is to use reverse DNS (same as your bundle identifier —ex: com.myCompany.myProject)
- To define error codes, use an enum

Demo