

Mobile Engineer Candidate Code Exercise

Write a sample app that fetches and displays data from a RESTful Web API.

Applicant: Edward C Ganges



This document is submitted as part of a job application for Comcast internal review purposes only.
Document Copyright © 2019 Edward C Ganges. All rights reserved.

Mobile Engineer Candidate Code Exercise.....	1
Original Requirement Specification:.....	3
Implementation:	4
Tasks Completed:	4
Organizational Decisions:.....	5
Master-Detail based template:.....	5
Core-Data as the persistent information store:	5
Networking:	5
Application :	5
Services (API specific):.....	5
WebClient (shared):.....	5
Service Errors (shared):.....	5
Reachability (shared):	5
Folder Structures:	5
Shared, vs. Target folders:.....	6
TARGET_ Simpsons and TARGET_TheWire Folders:.....	7
File Implementation Decisions:	8
App Features and Feature templates:.....	8
CONTROLLERS folder:.....	8
AppDelegate.swift:.....	8
AppConfiguration.swift:.....	8
AppDefinitions.swift:	8
MasterViewController.swift:	8
MasterViewController.swift:	9
DetailViewController.swift:.....	10
FavoritesViewController.swift:.....	10
SlideMenuController.swift:	10
EXTENSIONS folder:.....	10
NSLayoutController+Extension.swift:	10
NSAttributedString+SearchHighlights.swift:.....	10
MODELS folder:.....	11
ShowCharacter.swift	11
ComcastCodeChallenge.xcdatamodeld	11
VIEWS folder(s):	12
Main.storyboard	12
LaunchScreen.storyboard	12
NETWORKING folder:.....	12
/Networking/Services —	12
/Networking/Shared —	12
RESOURCES folder(s):.....	12
Features and Screen Shots:	13
Main Menu Feature:	13
Collection View Feature:	14
Search Feature:.....	15
Favorites Feature:	16
Testing:	17
Enhancements:.....	17
End Of Document	17

Original Requirement Specification:

Purpose:

In order to best assess your fit with Xfinity Mobile, we'd like to get an idea of how you approach application development. The purpose of this challenge is for you to demonstrate your best work, knowledge of application architecture and development best practices. There is no time limit for this challenge, and the time in which you complete it does not affect our consideration. Your goal with the challenge is not simply to finish it -- we consider it a given that you can meet the requirements -- rather, the goal is to complete the challenge with the highest possible quality so that we understand what you will bring to our team.

Documentation Instructions:

Please include a separate **NOTES** document (pdf, .doc, etc) outlining the major decisions you made while building the app and the motivations behind them, and, for any libraries you chose to use, explanations of what they do and why you chose them. (We're interested in the decision making process you use while developing)

App Requirements

Write a sample app that fetches and displays data from a RESTful Web API.

The app should be able to display the data as a text only, scrollable list of titles, and on phones, should be toggle-able from the list to a scrollable grid of item images. The title and description for each item should each be parsed out of the data in the "Text" field. Images should be loaded from the URLs in the "Icon" field. For items with blank or missing image URLs, use a placeholder image of your choice.

Clicking on an item should load a Detail view, including the item's image, title, and description. You choose the layout of the Detail view. On tablets, the app should show the list and detail views on the same screen. For phones, the list and detail should each be full screen. The app should have an tool-bar that displays:

For Phone - The name of the app on the item list screen, and the title of the item on the detail screen

For Tablets - The name of the app

In addition, two versions of the app should be created. Each version has a different Name, package-name, and url that it pulls data from. interested in your methodology for creating multiple apps from a shared codebase)

Version One Name: Simpsons Character Viewer

Data API: <http://api.duckduckgo.com/?q=simpsons+characters&format=json>

Package name: com.sample.simpsonsviwer

Version Two Name: The Wire Character Viewer

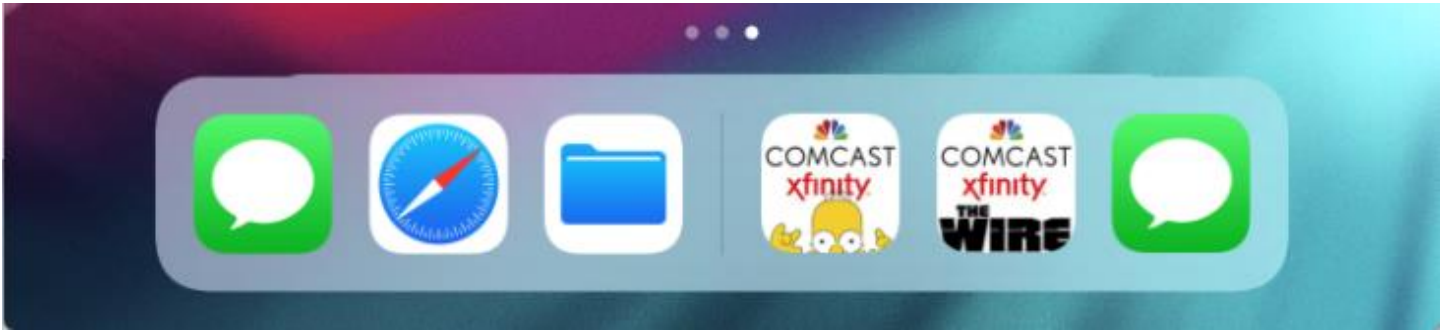
Data API: <http://api.duckduckgo.com/?q=the+wire+characters&format=json>

Package name: com.sample.wireviewer

As a bonus, implement one of the following (not required):

1. Local search functionality that filters the item list according to items whose titles or descriptions contain the query text
2. Functionality to favorite characters, and a drawer layout for navigating between a view of all characters, and a view of only favorited items
3. Animated transitions between the item list and detail screen (for phones), or animated detail-pane loading on tablets. You choose the complexity and character of the animations.

Implementation:



Tasks Completed:

For this exercise, I completed the primary App Requirements, plus Additional Tasks numbers 1 and 2.

Basic App Requirements:

- iPhone
 - iPhone plus Collection View
- iPad
- version one - Simpsons
- version two – The Wire

Additional Tasks:

1. Search
2. Favorites
3. ~~Animated Transitions~~

The application is bundled into a .zip file, attached with this documentation.

The application includes icons and launch screens for each screen device and each version.

This application was created by Edward Ganges. For information or questions, please contact Edward Ganges at eganges@yahoo.com or via phone at 610-733-8433.

Organizational Decisions:

Master-Detail based template:

Chosen to facilitate iPad/Universal deployment. Although not clearly stated, this controller works for both iPhone and iPad; the simultaneous Master-Detail view is suppressed if iPhone orientations do not include rotate left/right.

Core-Data as the persistent information store:

Chosen as the most efficient way to manage large amounts of data, especially if the data should persist across reboots, as is the case when an app supports “favorites”.

Core Data also provides the most efficient algorithms for searching and filtering results within large datasets as well, so provides the most scalable solution in the event of future targets with MUCH larger data sets, or embedded objects larger than the images included in these samples.

For this project, I created a simple CDShowCharacter entity, and load our JSON response into that. This data entity also contains Icon image data which is downloaded separately, and flags for tracking and cancelling background downloading tasks.

Networking:

I implement a loosely coupled networking design pattern here, separating the ReSTful API requests into logical components:

Application :

which initiates the API specific requests and implements final transformation of the data

Services (API specific):

which handle the preparation of params for calling the API via the WebClient, and perform validation/parsing of the results before returning them to the Application. Since all targets in this project call the same base URL, I only implement one service.

WebClient (shared):

which handles the boilerplate of making the Request, getting the Response, parsing Errors...

Service Errors (shared):

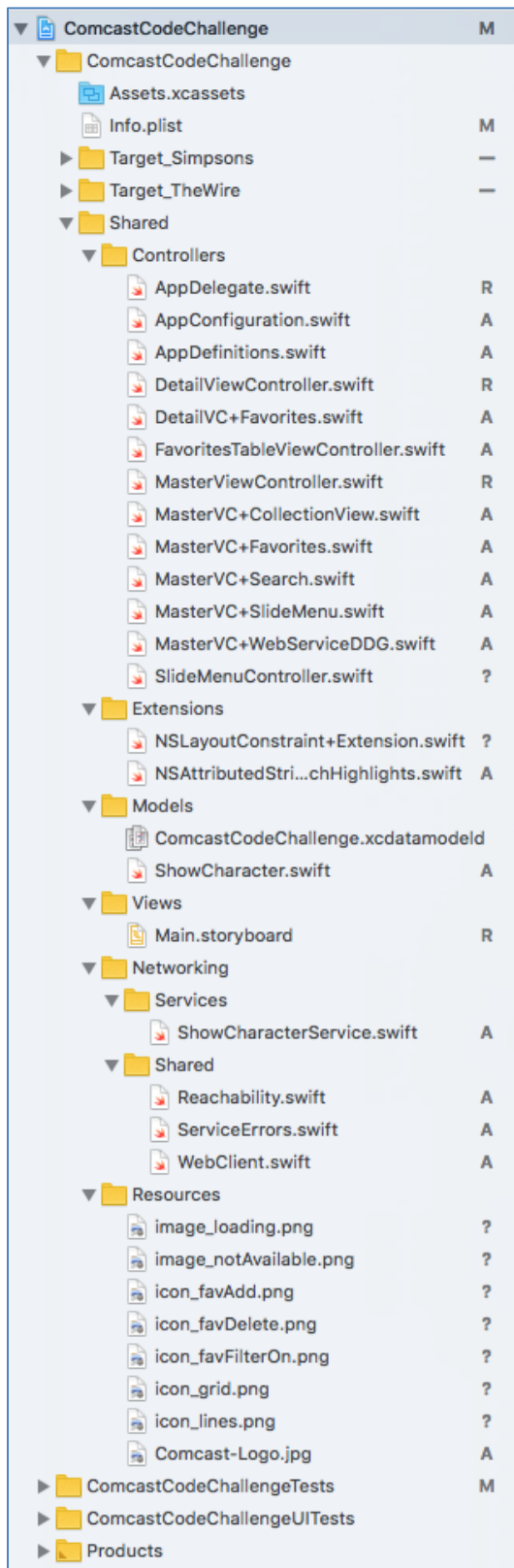
boilerplate to manage NSError returned from the Web API calls, such as the “Network Unavailable” message.

Reachability (shared):

useful check before implementing WebClient; common utility component, easily downloaded via StackOverflow.

Folder Structures:

I organized the file system by Models, Views, Controllers, Extensions, Resources, Networking. I find this provides quicker access to files, and a logical structure to assist others who follow behind.



Shared, vs. Target folders:

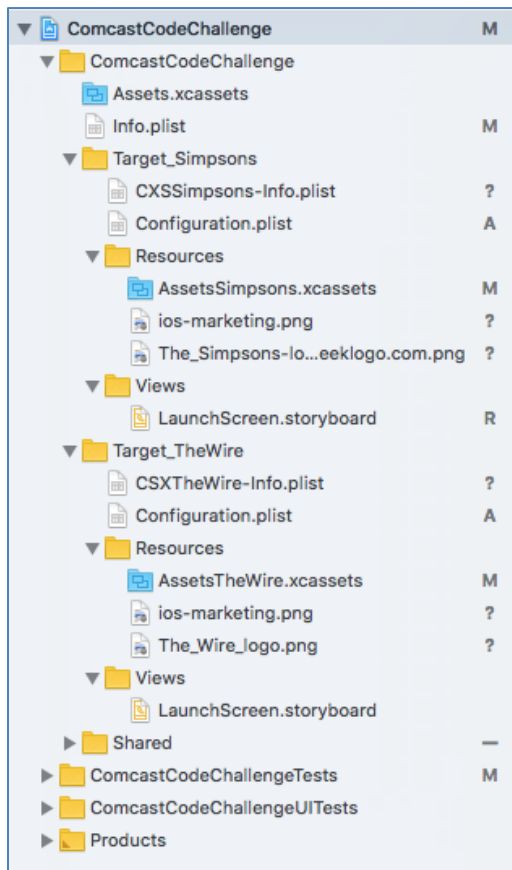
For this project, I placed the majority of the files into the Shared folder, knowing I wouldn't make the Target_ folders until after I completed the baseline functionality and cloned the Master solution into new version specific targets.

Once I had a working prototype, I created Target_Simpsons and Target_TheWire, which are described below.

All other customization is handled within AppConfig.swift, based on the contents of the target specific Configuration.plist files, as described below.

TARGET_ Simpsons and TARGET_TheWire Folders:

Target specific files and assets, organized in a folder structure emulating that of the master Shared folder structure.



Ultimately, the only Target specific files for this project are:

- CSX[Target]-Info.plist -
the app store information file
- Configuration.plist –
the customization override settings for this target
- Assets.xcassets –
the icons and legacy launch screens for this target
- LaunchScreen.storyboard -
the modern launch image for this target

All other app customizations are handled within AppConfigurationswift, based on the contents of the target specific Configuration.plist files.

File Implementation Decisions:

App Features and Feature templates:

For this project, the app features have strong overlapping dependencies on the master presentation controller(s). For this reason, for each app feature, I tried to implement a Decoration (VC+Extension) design pattern against the containing View Controller where the feature is implemented. This allows for a clean modularization and maintenance of code, and avoids a “supersized single template” design, while also avoiding setting up overly complex attempts at loose coupling.

CONTROLLERS folder:

AppDelegate.swift:

Here I simply added a convenience variable for cleaner reference to the AppDelegate Singleton. I also added an extra method to make it more convenient for Favorites and Image downloads to access persistence.

AppConfiguration.swift:

I use this to access and manage Target specific variables, URLs, images, and behavior changes to the baseline. These values are loaded from Configuration.plist files and store in the AppConfiguration.shared() instance.

AppDefinitions.swift:

This centralizes constants, strings,. Etc, and defines strings and keys used throughout the application. This template also compliments the Configuration pattern by defining a more loosely coupled mapping from AppConfiguration.swift to the Configuration.plist.

MasterViewController.swift:

Inheritance:

To better implement the Side Drawer functionality for Favorites, I “fake” multiple inheritance by rearranging and “stacking” the required inheritances to allow MasterViewController to be both a SlideMenuController and an UITableViewController.

1. First I change MasterViewController’s inheritance from UITableViewController to SlideViewController
2. Second, I also change SlideViewController’s inheritance from UIViewController to UITableViewController.
3. This allows MasterViewController to inherit both, and keeps the resulting code as DRY as possible, and avoids pasting the entirety of SlideViewController into a MasterViewController extensions.

MasterViewController.swift:

Decoration:

MasterViewController ships with Apple's Master-Detail application template, and is tightly coupled into the SplitViewController, Main.storyboard and DetailsViewController.swift implementation. Here I decide that the easiest way to leverage this template is to use a Decoration pattern to extend its functionality on a feature-by-feature basis, via class extensions.

Starting with the default template, I augment MasterViewController.swift with MasterVC+extension files per feature, in an attempt to keep the code modular and DRY. The Class extension files include:

- **MasterVC +CollectionView –**
 - here I extend MasterViewController as a delegate of CollectionView.
 - This controls the custom header which includes the CollectionView toggle button.
 - Configuration.plist to control Image Aspect Ratio and cell background color on a per Target basis, allowing a better overall presentation to the user, per Target.
- **MasterVC +Favorites –**
 - here I extend MasterViewController to manage the images and Tap functionality for the Favorites buttons which are spread across the Title Bar, Search Bar, Table View, and Collection View – all of which are also extensions of MasterViewController.
 - I could have created a static Favorites class, but with 90% of the favorites requirements passing through MasterViewController, that approach seemed artificial.
- **MasterVC +Search –**
 - here I extend MasterViewController to manage the Search Bar.
 - Again, as Search effects MasterVC's TableView and CollectionView, extending it for Search provides tight integration to MasterVC's reusable components, specifically, it's embedded fetchedResultsController.
 - I chose SearchBar over SearchBarController to because of the preexisting efficiency of the CoreData and the already embedded fetchedResultsController, which is easily filtered via an NSPredicate based on the SearchBar text.
 - For user clarity when moving from Search results to the Detail View, I also implement a highlighting of the search term as it appears in the description.
- **MasterVC +SlideMenu –**
 - I implement MVC+SlideMenu as an extension to support SlideMenuDelegate, which is consistent with my ongoing design pattern, and allows me fine tune layer performance of the Slide Drawer, used by MVC+Favorites.
 - NOTE: this is more thoroughly discussed in relation to the SlideMenuController.swift file, described later.
- **MasterVC +WebServiceDDG –**
 - Here, I extend MasterVC to support the Application, WebService, WebClient Networking pattern I described adopting earlier. Having this extension basically isolates the Application layer into it's own module for better readability and maintenance.
 - This extension decouples our logic from our MVC presentation and links us to the external DuckDuckGo (character) API. Upon receiving the data, it maps it's contents into MasterVC's persistent store.

DetailViewController.swift:

Implementation:

Here I make sure to set an aesthetically pleasing default presentation, in case of invalid access or if iPad Portrait defaults to an uninitialized controller.

Decoration:

Just like MasterViewController, I decide here that the easiest way to leverage this template it to use a Decoration design pattern to extend it's functionality via a Class+extensions naming pattern.

- **DetailVC+Favorites –**
 - Simple update of the persistent store if the icon is tapped.

FavoritesViewController.swift:

I made this a simple TableViewController using the shared persistent store for displaying the Favorites Table View in the Slide Drawer. I made this as dumb as possible, passing the selected entity back to it's delegate, MasterViewController.swift.

SlideMenuController.swift:

This is an open source implementation of the Side Drawer functionality. I chose this template for it's open source nature, it's single file nature, and it's ability to be inherited and utilized via Storyboards.

EXTENSIONS folder:

I borrowed some LayoutConstraint code here, to allow for easier experimentation and final placement of the CollectionView swap button and other Layout components.

NSLayoutController+Extension.swift:

a useful utility for mapping NSLayoutConstraints as defined via Apple's Layout Markup Language.

NSAttributedString+SearchHighlights.swift:

a quick algorithm highlighting matches in a NSAttributedString, with attribution to Bohdan Savych. I chose this one for it's potential scalability re: it's ability to highlight an array of search terms.

MODELS folder:




This folder stores the CoreData model file, and my prototype/Facade data model.








ShowCharacter.swift

I use this model as part of a Façade pattern to transition/transformation object between the Web API JSON response and the final CD object. This decouples the CoreData and Web API models. I went with this approach to allow for a cleaner transform path and more consistency with non-CD adaptations of the implemented Networking pattern.

ComcastCodeChallenge.xcdatamodeld

This is the Core Data repository for this project. Nothing to see here.

ENTITIES	
 CDSShowCharacter	
 Event	
FETCH REQUESTS	
CONFIGURATIONS	
 Default	

Attributes	
Attribute	Type
 details	String
 iconAddress	String
 iconData	Binary Data
 iconTaskID	String
 isFavorite	Boolean
 linkAddress	String
 title	String
+ -	

For tracking the current Icon Retrieval Task, as it relates to reusable cells and lazy loading, I use a string identifier (iconTaskID) to get around the issue where using Type = Transformable as?

[URLSessionDataTask](#) is not NSCoder compliant, and thus fails the `persistentObject.save()` request. Therefore, I store the Task's unique HashValue instead.

VIEWS folder(s):

I place most storyboards, here. This folder exists within the Shared folder structure, plus each of the Target folder structures.

Main.storyboard

This is a shared file, across all Targets.

Main.storyboard ships with Apple's Master-Detail application template, and is tightly coupled into the SplitViewController, MasterViewController.swift, and DetailsViewController.swift implementation.

The main VIEW presented to the user is via MasterViewController, which implements the VIEW as a protected UITableView, which does not allow arbitrary drag-and-drop additions to its view hierarchy (exceptions are SearchBar, SearchBarController; not allowed includes collectionView,ActivityIndicatorView, etc).

I find the easiest way to leverage this template it to add a SearchBar to the tableview (managed by MaserVC+Search), and add CollecionView, activityIndicatorView components as peers, which I can integrate into UITableView later via code; adjusting the dynamic layout via the Visual Format Language.

LaunchScreen.storyboard

This file is implemented as Target specific.

I design this storyboard specific to each target, using the target specific branding assets and colors.

NETWORKING folder:

/Networking/Services —

contains the xyz specific logic for each external service provider. In the pattern I use here, each service provider would get their own xyzService.swift file. For this solution, only one provider endpoint is used for both shows (Simpsons, The Wire), so only one ShowCharacterService.swift file is required.

/Networking/Shared —

contains the common components to support this networking design pattern.

RESOURCES folder(s):

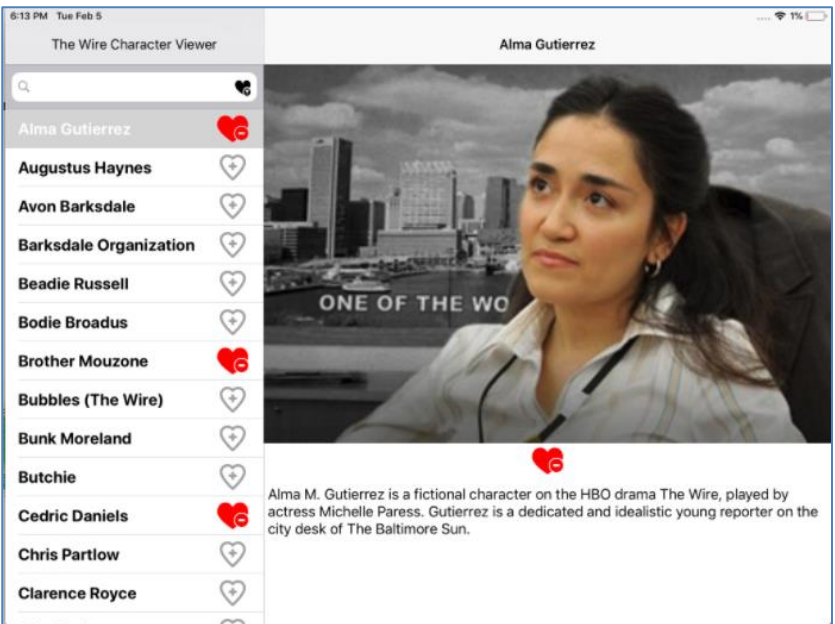
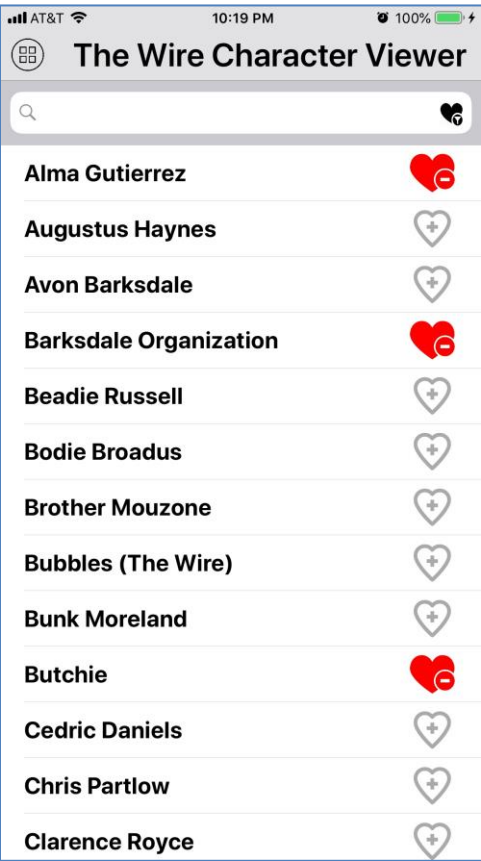
This folder exists within the Shared folder structure, plus each of the Target_ folder structures. For the Targets in this project, only a custom Show logo and a copy of the target app icon (ios-marketing.png) are needed. The large icon is used as the default Details icon, as seen on iPad.

Features and Screen Shots:

Below are implementation screenshots for the project, which again, may be compiled and executed via the attached .zip file.

Main Menu Feature:

Functionality includes: Collection View Toggle Option (iPhone), Custom Title View, Search Bar, Favorites Drawer Toggle Option and Text Only list, iPad Split View Open,iPad Split View Closed, iPad Split View Landscape.



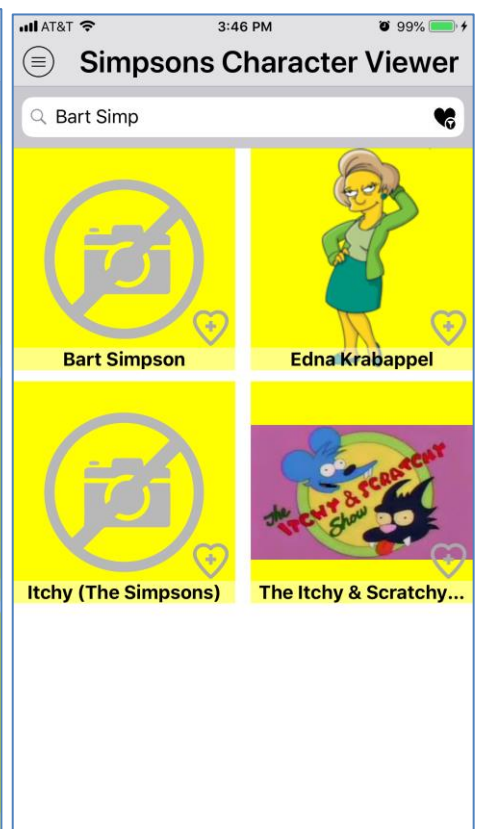
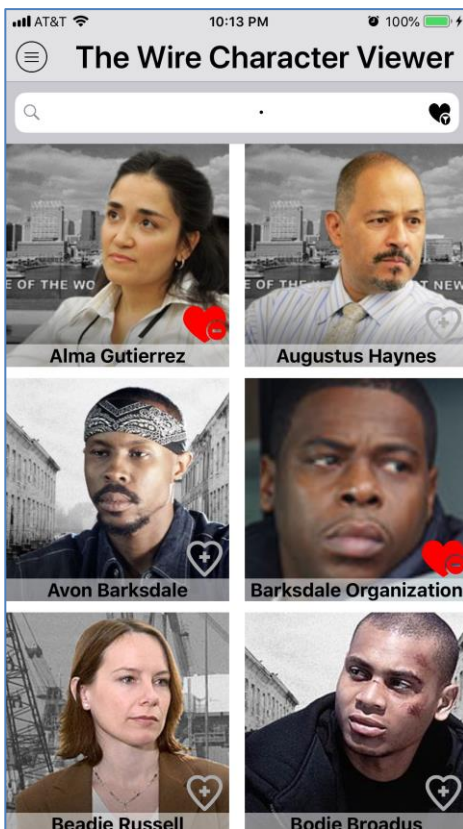
Collection View Feature:

Functionality includes Toggle-able CollectionView, customizable cell background colors, customizable image aspect ratios, integration with DetailsView, Search, and Favorites.

Design and Coding Choices:

(As also noted above in the description for MasterVC+CollectionView.swift)

- here I extend MasterViewController as a delegate of CollectionView.
- I create a custom header which includes the CollectionView toggle button.
- I use Configuration.plist to control Image Aspect Ratio on a per Target basis, allowing a better overall presentation to the user, per Target.
- I use Configuration.plist to control Cell BackgroundColor on a per Target basis, allowing a better overall presentation to the user, per Target.
- I make sure that CollectionView is responsive to Search.
- I address implementation incompatibilities between CollectionVew and FavoritesDrawerView by assuring that FavoritesView is intelligently preferred over CollectionView.



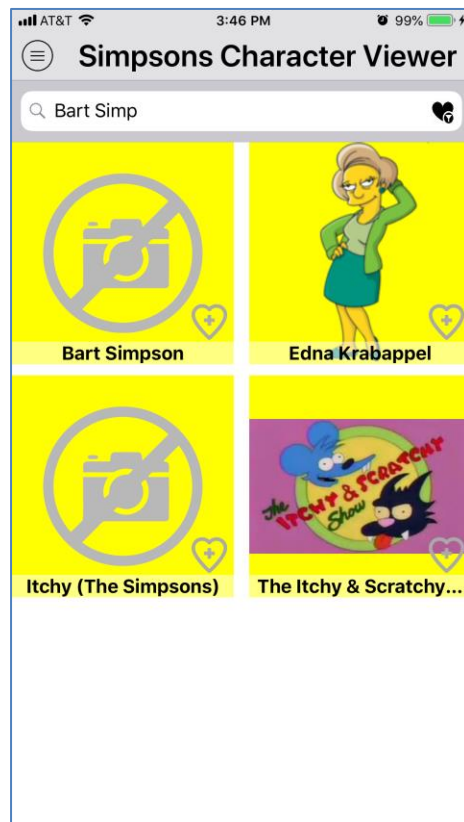
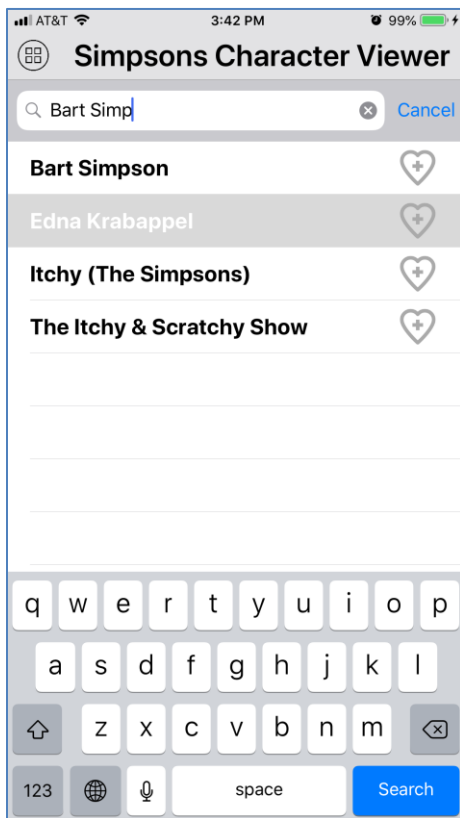
Search Feature:

Functionality includes TableView search, CollectionView search, DetailView search results.

Design and Coding Choices:

(As also noted above in the description for MasterVC+Search.swift)

- here I extend MasterViewController to manage the Search Bar.
- Again, as Search effects MasterVC's TableView and CollectionView, extending it for Search provides tight integration to MasterVCs reusable components, specifically, it's embedded fetchedResultsController.
- I chose searchBar over SearchBarController because of the preexisting efficiency of the already embedded CoreData fetchedResultsController, which is easily filtered via an NSPredicate based on the searchBar text.
- For user clarity when moving from Search results to the Detail View, I also implement a highlighting of the search term as it appears in the description to smooth out the experience and help users to better understand their search results as well.

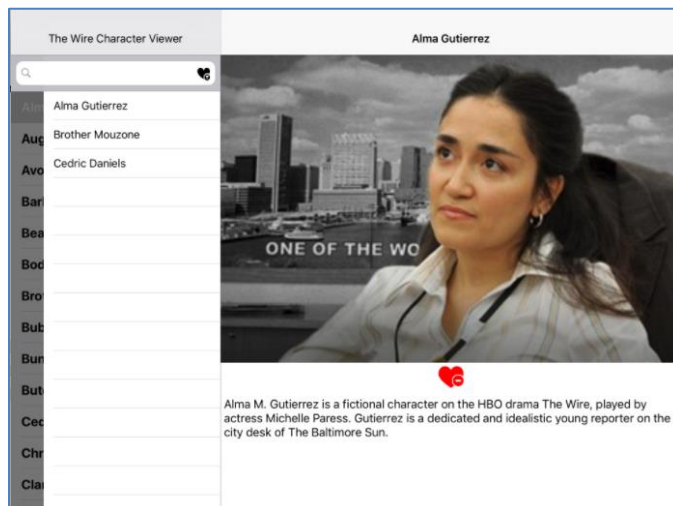
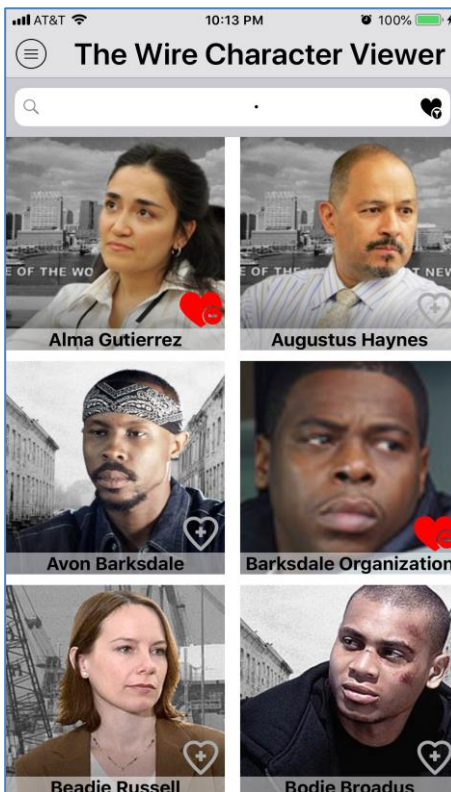
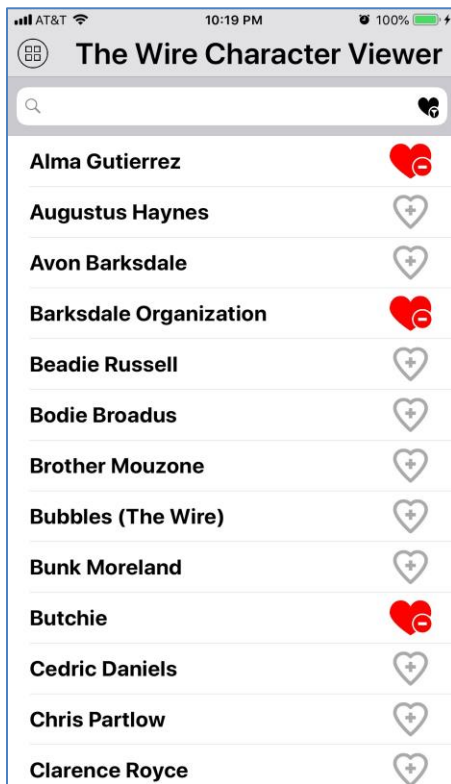


Favorites Feature:

Functionality includes Text Only view with favorites, CollectionView with favorites (iPhone), Favorites Drawer extended iPhone, and Favorites Drawer extended iPad Landscape

Design Choices:

- here I extend MasterViewController to manage the images viewed and tap functionality for the Favorites buttons which are spread across the Title Bar, Search Bar, Table View, and Collection View – all of which are also extensions of MasterVeiwController.
- I toggle the Favorites Drawer via the Accessory Icon embedded in the Search bar. The list can be toggled closed, swiped closed, or tapped closed via the inactive grey area.
- I set the Favorites list to not show above the CollectionView, but instead toggles the CollectionView to hidden, if necessary. Likewise, the CollectionView toggle switch will hide/show the CollectionView above the favorites list.



Testing:

No automated testing scripts have been added to this project at this time.
All testing has been performed via manual QA.

Enhancements:

The following recommended functionality enhancements were NOT implemented to avoid (or to avoid the impression of) Scope Creep. They are listed here in the spirit of creativity, sharing, and the pursuit of perfect software.

- Search Results Highlighting:
OK, I couldn't help myself - I left this one in, as it was already implanted in order to more readily verify search results during testing.
- DetailView Previous, Next Navigation:
Once on the details page, it would be nice to be able to swipe, edge tap, or icon tap to move forward and backward through characters, dependent on a full or Search limited set of Show Characters. This would be nice as in infinite scroll, looping at either end.
- SiriKit Integration
It would be nice to add an Audio icon to the toolbar housing the Favorites icon, and have Siri read the description to the user.

End Of Document

Thanks for the Exercise!
I look forward to your feedback.

Best regards,
Edward Ganges