Xfinity Coding Exercise documentation

Environment:

This project is built with Xcode 10.1 using Swift language. You should be able to build this project in Xcode 9.3 and above and able to run on iOS SDK 10.0 and above. Do a "pod install" if you see any sandbox out of syn errors.

Approach:

The key approach to display different layouts for master and detail screens iPad and iPhone is to use a UISplitViewController.

Architecture:

MVVM design pattern is used to design this whole project.

Third Party Frameworks:

Alamofire and Alamofireimage Cocoapods are used to make API calls and load images asynchronously. This makes network layer coding easy, we can use URLSession API as well.

Targets:

There are two targets in this project, "SimpsonsViewer" and "WireViewer" each hitting different API end points from duckduckgo.com

Description of Files:

Model:

`XMCharacterList` Contains list of `XMCharacter` objects. `XMCharacter` Is the actual model we will be using to show title, description and images of characters we fetched from API

ViewModel:

`XMCharactersListViewModel` Takes care of making API call and refreshing characters data inside collection view on master screen.

View:

`XMSplitViewController` Is the basic split viewcontroller that is responsible for showing master and detail screens. It has two view navigation controllers inside it, one for master and another for detail.
`XMCharcatersMasterViewController` Is root view controller for Master and is *`XMCharacterDetailViewController`* root view controller for detail. `XMCharcaterSelectionDelegate` Is a protocol that is implemented by detail to load its data based on object that is selected on master

Network Layer:

`XMCharctersApiManager` Is the core api manager that makes API calls to supplied urls using Alamofire framework. This class leverages the compiler flags of the appropriate target and decides which url it needs to use to render the data. Example, `SIMPSONVIEWER` is compiler flag for Simsonsviewer target and `WIREVIEWER` is compiler flag for WireViewer target.

Image Caching and Rendering:

`XMImageCell` Uses asynronous image loading function from *`XMImageViewExtension`* to lazy load the images from character urls. It will first look in image cache and if it is not cached it will request the image from supplied url and cahes the image into our imageview cache so as to not make API call every time it want to load image.