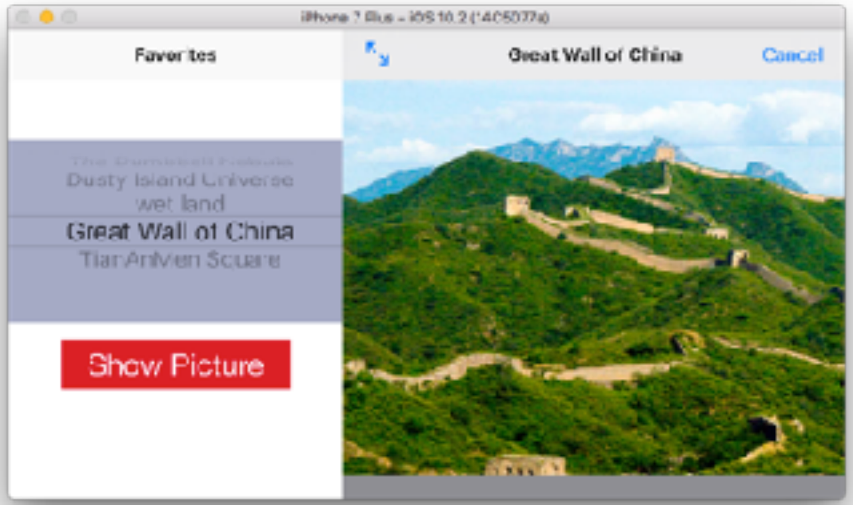
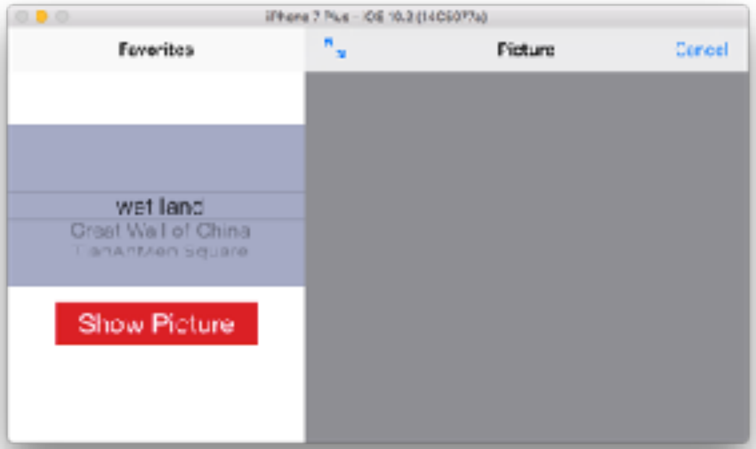
Networking — the “ImageFetcher”

ImageFetcher is an app that fetches a user’s favorite images from the Internet. This app’s UI looks as follows. It’s mainly a split view. On the left side, the user can pick a photo, and the app will download the photo from the Internet and display the picture on the right side.



To start with, first download the starter project. Build and run, the starter UI should look like the following:



Defining the Model Class for Networking

Next, we are going to define a model class that performs networking. Create a new class named *IFImage*, and then define the following properties and the init method.

```
import Foundation
import UIKit
```

```
class IFImage: NSObject {

    var imageName: String
    var imageURL: URL?

    var image: UIImage? {
        if imageData != nil {
            return UIImage(data: imageData!)
        } else {
            fetchAsync() //start asynchronous image fetching from the Internet

            return UIImage(named: "photoalbum")
        }
    }

    // stores the image data
    private var imageData: Data?
    var isFetching = false

    init(_ name: String, _ url: String) {
        imageName = name
        imageURL = URL(string: url)

        print("init image \(imageName)")

        super.init()
    }
}
```

The property *image* is a read-only computed property representing the image. The actual image data is stored in a private property *imageData*. When the caller asks for an image, if *imageData* has value, the *image* property returns the UIImage object created from the *imageData*. Otherwise, the *image* property returns a default image *photoalbum*, and at the same time, starts to fetch the image asynchronously from the Internet. Since the downloading may take some time. During this period, other parties may attempt to access the image again. To prevent the image from downloading multiple times, we define a flag *isFetching*. The downloading will only start when this flag is *false.*

Now, we are going to define the *fetchAsync* methods with various networking APIs, starting from wrapping the synchronous blocking networking methods with Grand Central Dispatch functions. We will also follow the *Asynchronous Networking Design Pattern*. In the *fetchAsync()* method, the *Data(contentsOf:)* function is a synchronous blocking networking API. To make it asynchronous and non-blocking, we run this method as an asynchronous task in the built-in *global* dispatch queue. Any task in this queue will be scheduled and executed in a worker thread so that the main thread is not blocked. Once the data is received from the Internet, we follow the asynchronous networking design patter to update the UI-related properties in the main thread by dispatching the related tasks in the *main* queue, which is executed on the main thread. In this app, we use the *Notification* mechanism to notify the other interested parties that the image has been fetched. We also post the notification in the main thread since it is UI-related.

```
if !isFetching, let url = imageURL {
            print("start feteching image \(self.imageName)")

                isFetching = true
            DispatchQueue.global(qos: .userInitiated).async() { [weak self] in
                do {
                    let imageData = try Data(contentsOf: url)
                    print("received data for image \(self?.imageName)")

                    DispatchQueue.main.async {
                        if let strongSelf = self {
                            strongSelf.imageData = imageData

                            NotificationCenter.default.post(name: NSNotification.Name("ImageFetched"), object:
strongSelf)

                            strongSelf.isFetching = false
                        }
                    }
                } catch {
                    print("error fetching image \(self?.imageName) error: \(error)")

                    DispatchQueue.main.async {
                        if let strongSelf = self {
                            strongSelf.isFetching = false
                        }
                    }
                }
            }
        }
```

Defining the View Controller for Displaying the Image

Now we are going to define the *ImageViewController* class that displays the fetched image. In this class, first define a property for the image:

```
var ifImage: IFImage?
```

This view controller requests the image from the *ifImage* object every time its view is about to be shown, which is in the *viewWillAppear* method. It is possible that the image is not available at that time. In that case, the *ifImage.image* returns a default image. The view controller will display this default image, and at the same time, starts to listen to the notification message. The *NotificationCenter.default.addObserver* method registers the view controller as the listener to the *ImageFetched* message which will be sent by this specific *ifImage* object. When this message is posted by this *ifImage* object, the method *imageFetched()* will be called, which gets the image from the *ifImage*.

```
override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(animated)

        self.imageView.image = ifImage?.image
        self.imageView.sizeToFit()
        self.scrollView.contentSize = self.imageView.bounds.size

        NotificationCenter.default.addObserver(self, selector: #selector(imageFetched), name:
NSNotification.Name("ImageFetched"), object: ifImage)
}
```

The view controller stops listening to the notification message when its view becomes invisible, which is in the *ViewWillDisappear()* method:

```
override func viewWillDisappear(_ animated: Bool) {
        super.viewWillDisappear(animated)

        NotificationCenter.default.removeObserver(self)
    }
```

One last piece to make the image fetching work is to modify the model array in the *ListViewController* class as follows. First, replace the *image* array with *IFImage* objects rather than the image info dictionary.

```
let images = [IFImage("wet land", "http://127.0.0.1/PhotoAlbum/b01.jpg"),
                  IFImage("Great Wall of China", "http://up.qin180.com/2014/0326/20140326110836781.jpg"),
                  IFImage("TianAnMen Square", "http://military.people.com.cn/NMediaFile/2016/0102/
MAIN201601021010000399701943600.jpg")]
```

Secondly, in *prepare(for segue)* method, replace the *TODO:* part with the following two lines of code:

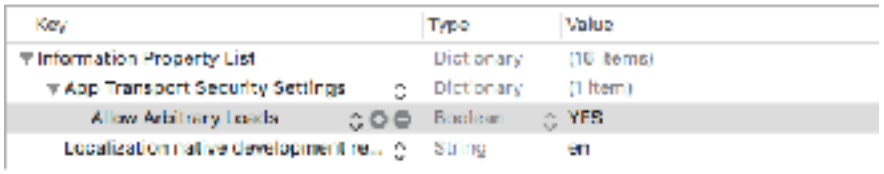
```
imageVC.ifImage = images[rowIndex]
imageVC.title = images[rowIndex].imageName
```

Lastly, in the *pickerView(titleForRow)* method, replace `images[row][“imageName”]` with `return images[row].imageName`.

Build and run the app. The app may crash when you click an image from the list to download. The error message says:

"App Transport Security has blocked a cleartext HTTP (http://) resource load since it is insecure. Temporary exceptions can be configured via your app's Info.plist file."

The purpose of *App Transport Security* is to improve the security of connections between an app and web services by enforcing the use of secure connections. With ATS, all network requests should now be sent over HTTPS. If you make a network connection using HTTP, ATS will block the request and display the error. To resolve this issue, open *Info.plist*, right click the editor and select *Add Row*. For the key column, enter *App Transport Security Settings*. Then add the *Allow Arbitrary Loads* key with the type *Boolean* . By setting the key to *YES* , you explicitly disable App Transport Security.



Key	Type	Value
▼ Information Property List	Dictionary	(10 items)
▼ App Transport Security Settings	Dictionary	(1 item)
Allow Arbitrary Loads	Boolean	YES
Localization native development re...	String	en

Build and run the app again. This time, you should be able to download and shown the image.

Fetching the Image with URLSession Tasks

Now, we will re-define the *fetchAsync()* method to use the asynchronous API *URLSessionDataTask* to download the image.

```
func fetchAsync() {
        if !isFetching, let url = imageURL {
            print("start feteching image \(self.imageName)")

            isFetching = true

            let dataTask = URLSession.shared.dataTask(with: url) { [weak self] data, response, error in
                if let error = error {
                    print("error downloading image \(self?.imageName) error: \(error)")

                    DispatchQueue.main.async {
                        if let strongSelf = self {
                            strongSelf.isFetching = false
                        }
                    }
                } else if let httpResponse = response as? HTTPURLResponse, httpResponse.statusCode == 200,
let imageData = data {

                    print("received data for image \(self?.imageName)")

                    DispatchQueue.main.async {
                        if let strongSelf = self {
                            strongSelf.imageData = imageData

                            NotificationCenter.default.post(name: NSNotification.Name("ImageFetched"), object:
strongSelf)

                                strongSelf.isFetching = false
                            }
                    }
                } else {
                    print("failed downloading image \(self?.imageName) error: \(response)")

                    DispatchQueue.main.async {
                        if let strongSelf = self {
                            strongSelf.isFetching = false
                        }
                    }
                }
            }
            dataTask.resume()
        }
    }
```

We can also use the *NSURL Download Task* to fetch the image. In this case, the *fetchAsync()* method can be defined as:

```
func fetchAsync() {
        if !isFetching, let url = imageURL {
            print("start feteching image \(self.imageName)")

            isFetching = true

            let downloadTask = URLSession.shared.downloadTask(with: url,
                                                  completionHandler: { [weak self] url, response, error in
```

```
                if let error = error {
                    print("error downloading image \(self?.imageName) error: \(error)")

                    DispatchQueue.main.async {
                        if let strongSelf = self {
                            strongSelf.isFetching = false
                        }
                    }
                } else if let httpResponse = response as? HTTPURLResponse, httpResponse.statusCode == 200,
let url = url {
                     do {
                         let imageData = try Data(contentsOf: url)
                         print("received data for image \(self?.imageName)")

                         DispatchQueue.main.async {
                             if let strongSelf = self {
                                strongSelf.imageData = imageData

                                NotificationCenter.default.post(name: NSNotification.Name(rawValue:
"ImageFetched"), object: strongSelf)
                                strongSelf.isFetching = false
                             }
                         }
                     } catch {
                         print("error fetching data for image \(self?.imageName) error: \(error)")

                         DispatchQueue.main.async {
                         self?.isFetching = false
                         }
                     }
                } else {
                    print("failed downloading image \(self?.imageName) error: \(response)")

                    DispatchQueue.main.async {
                        if let strongSelf = self {
                            strongSelf.isFetching = false
                        }
                    }
                }
            })
            downloadTask.resume()
        }
    }
```

The UISplitViewControllerDelegate Methods

Currently, if this app is running on an iPhone and the iPhone is in the *Portrait* position, the app will show the main page. We would like the app to show the picture page instead. We can implement this function by adding the following code in the *TODO* part of the following *UPSplitViewControllerDelegate* method, defined in *AppDelegate.swift.* This method will show the main page only when the image page does not contain an image.

```
func splitViewController(_ splitViewController: UISplitViewController, collapseSecondary
secondaryViewController:UIViewController, onto primaryViewController:UIViewController) -> Bool {
        guard let secondaryAsNavController = secondaryViewController as? UINavigationController else
{ return false }
        guard let topAsDetailController = secondaryAsNavController.topViewController as?
ImageViewController else { return false }
        if topAsDetailController.ifImage == nil {
            // Return true to indicate that we have handled the collapse by doing nothing; the secondary
controller will be discarded.
            return true
        }
        return false
    }
```