# **Tokoin Mobile Test**

* This repository contains all source code for Tokoin Application .
* The application have this feature:  
      a. News List with image  
      b. News detail with image  
      c. Link to open original news  
      d. Have 3 tab view at home and it will show list about:  
          i. Top Headline news with image  
          ii. Custom news based on user preferences (user must be presented with keyword selection from: bitcoin, apple, earthquake, animal. User can only choose one keyword)  
          iii. Profile  
  User can register with username at profile and data (user preferences) will be saved on local storage.

## Installation

Pull or download project from GitHub url.

## Development

To develop application in this repository, ensure that you have at least the following software:

* Xcode 10.3 (or later)
* CocoaPods 1.7.2 (or later)

### **Running Unit Tests**

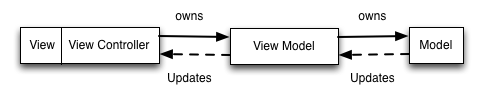
Select a scheme and press Command-u to build a component and run its unit tests.

### **Running Apps**

* Open **tokointest.xcworkspace** with Xcode
* Open Terminal and navigate to project folder by dragging project from Xcode to Terminal , after that run command : **pod install** to install framework/library from Podfile .
* Select a Simulator and press Command-R to run application on simulator. ( Remember choose **Automatically manage signing** from **Signing & Capabilities** Tab to auto sign app with current bundle identifier )

### **App Structure**

**MVVM**



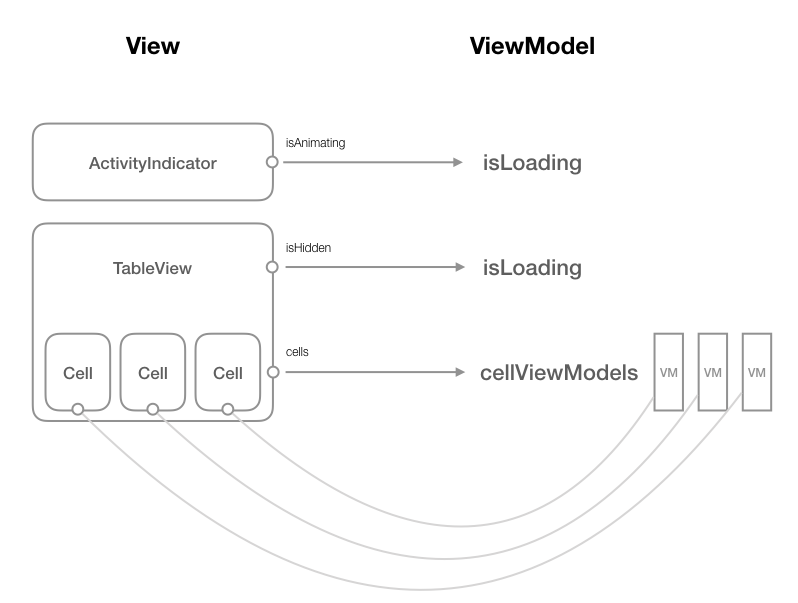
We are going to:

1. Design a set of interfaces for binding.
2. Move the presentational logic and controller logic to the ViewModel.

First thing first, let’s take a look at the UI components in the View:

1. activity Indicator (loading/finish)
2. tableView (show/hide)
3. cells (title, description, created date)

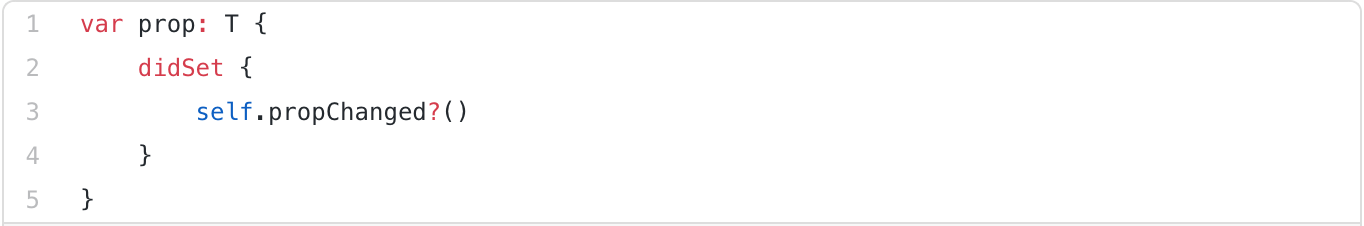
So we can abstract the UI components to a set of canonical representations:



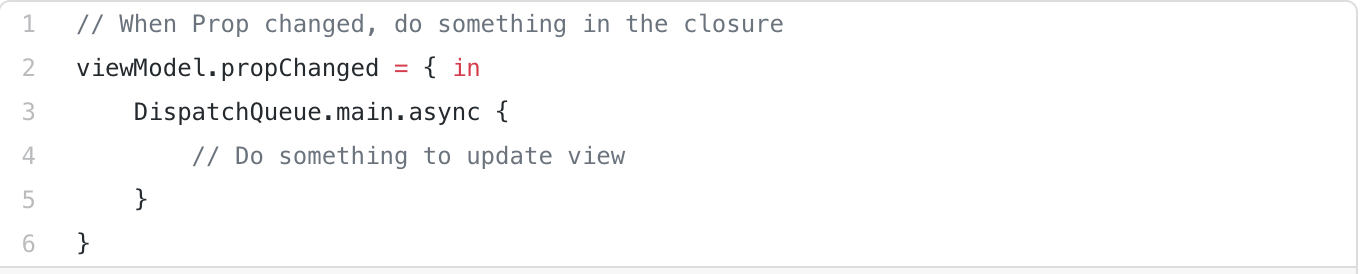
Each UI component has a corresponding property in the ViewModel. We can say that what we will see in the View should be the same as what we see in the ViewModel.

**Implement the Binding with Closure**

We bind things using a closure. Practically, in the ViewModel an interface/property for binding looks like this:



On the other hand, in the View, we assign a closure to the propChanged as a callback closure for value updates.



Every time the property prop is updated, the propChanged is called. So we are able to update the View according to the change of the ViewModel.

**Interfaces for binding in ViewModel**

Now, let’s start to design our ViewModel, the Article**ListViewModel**. Given the following three UI components:

## tableView

## cells

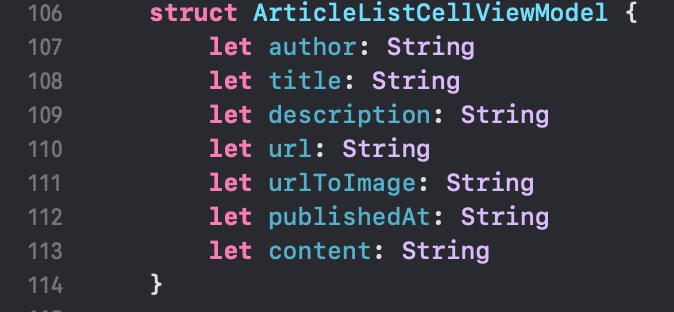
## activity indicator

## We create the interfaces/properties for binding in the ArticleListViewModel:

## 

Each **ArticleListCellViewModel** object forms a canonical representation of a cell in the table view. It provides data interfaces for rendering a UITableView cell. We put all **ArticleListCellViewModel** objects into an array **cellViewModels**, the number of cells is exactly the number of items in that array. We can say that the array, **cellViewModels**, represents the table view. Once we update the **cellViewModels** in ViewModel, the closure **reloadTableViewClosure** will be called and the View updates correspondingly.

A single **ArticleListCellViewModel** looks like this:



As you can see, the properties of the **ArticleListCellViewModel** provide interface for binding to UI components in the View.

**Bind the View with the ViewModel**

With the interfaces for binding, now we’ll focus on the View part. First, in the **ArticleListViewController**, we initialize callback closures in viewDidLoad:



The data flow now becomes:

1. The ArticleListViewModel starts to fetch data.
2. After the data fetched, we create **ArticleListCellViewModel** objects and update the **cellViewModels.**
3. The **ArticleListViewController** is notified of the update and then layouts cells using the updated **cellViewModels.**

It could be depicted as the following figure:

Fetch Data

ArticleListCellViewModel

Bind

Bind

Update

ArticleListViewModel

ArticleListViewController