

# MyBank App

A simple app that demonstrates some examples of **clean architecture**, **code organisation**, **loose coupling**, **unit testing** and some of the best practices used in modern iOS programming using *Swift*.

## App Goal:

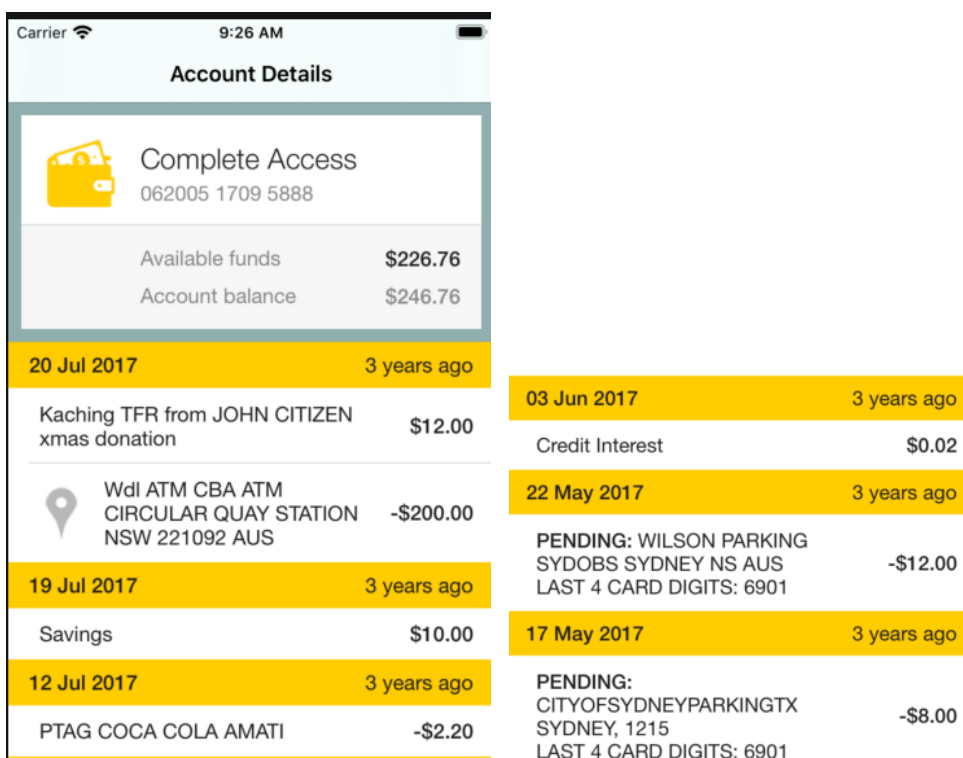
- Show account details after fetching from network
- Load a list of transactions grouped by dates under the account details
- Transactions are divided as cleared and pending but they show together in correct order grouped by dates to show transactions of the same day under one section
- Pull to refresh the view to load latest data and show a loading indicator
- Show error alerts when there is an issue
- Make sure all devices and orientations are supported (use size class)
- Some of the transactions are ATM based and they show location icon and tapping those transactions navigate to the ATM location page on a map view

The data for the `account details` + `transaction history` + `atm locations` are all combined together and comes from the this json file at:



<https://www.dropbox.com/s/tewg9b71x0wrou9/data.json?dl=1>

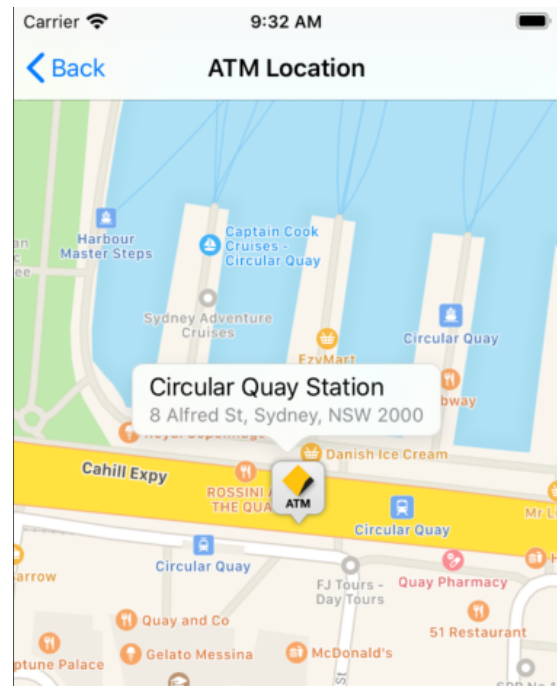
The code also pulls this data locally from a JSON file if needed and used for testing, debugging purposes, Particularly XCTest picks the data from a local stubbed JSON file.

## Screenshots: Remote data




## Screenshots: Local JSON data

Kaching TFR from JOHN CITIZEN xmas donation		\$12.00
<hr/>		
	WdI ATM CBA ATM CIRCULAR QUAY STATION NSW 221092 AUS 🚗🚗	-\$200.00
<hr/>		
04 Dec 2019	9 months ago	
<hr/>		
	WdI ATM CBA ATM TOWN HALL SQUARE NSW 253432 AUS	-\$50.00
<hr/>		
BPAY - Telstra mobile		-\$49.00
<hr/>		
19 Dec 2018	2 years ago	
<hr/>		
Savings		\$10.00
<hr/>		
03 Jan 2016	5 years ago	
<hr/>		
Transfer from REBECCA SHAW Lorem ipsum 😊😎🙏		\$150.00
<hr/>		



## Dark mode support (in landscape mode)

Account Details



Complete Access

062005 1709 5888

Available funds

\$226.76

Account balance


\$246.76

20 Jul 2017

3 years ago

Kaching TFR from JOHN CITIZEN  
xmas donation

\$12.00



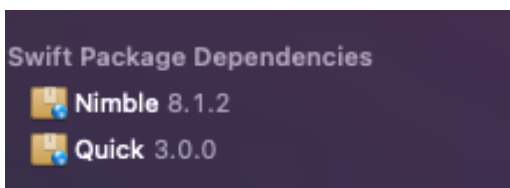
WdI ATM CBA ATM CIRCULAR QUAY STATION NSW  
221092 AUS

-\$200.00

## Installation

---

- Xcode **11.6+** (required)
- Clean /DerivedData folder if any
- Build the project and let the Swift Package Manager pulls two remote libraries used for unit testing – **Quick & Nimble**



## 3rd Party Libraries (only in unit testing)

---

- **Quick** - To unit test as much as possible 🤔
- **Nimble** - To pair with Quick 👯

## Apple frameworks

---

- **Combine** – To do reactive binding when needed 🤔
- **UIKit** – To build as usual everywhere 😊
- **SwiftUI** - To design/build UI fast (*only for the ATM map view*) 🐶

## Clean Architecture

---

- **VIPER** – To get as much loose coupling as possible
- Clean communication between **Display**, **Presenter** and **Router** in the view/scene stack
- Communication between **Interactor**, **Service** in the lower stacks of domain & network layers
- Connectivity of this components are achieved via protocol instances to achieve loose coupling and unit testability
- **View** (i.e. View Controller) is **Display** itself and contacts its Presenter
- **Presenter** may perform view related logic and immediately talk back to Display (for example, talking to interactor to fetch data from somewhere)

- **Presenter** can communicate with underlying **Interactor** layer for more complex task
- **Interactor** decides all Domain level business logic to take care of
- **Interactor** communicates with underlying Network Service layer via abstracted service provider which picks the one that is needed (i.e. real network vs. local stub)
- **Service** implementation communicates to its underlying http client abstracted via `Resource` which handles all networking (stubbed version reads from local JSON)
- **Interactor** gets back information via reactive binding from Service
- **Interactor** parses the data and apply any necessary data transformation
- **Interactor** gives outcome back to Presenter via reactive (AnyPublisher in Combine)
- **Presenter** handles all the presentation related logic necessary for the view
- **Presenter** talks back to **Display**
- **Presenter** also talks to **Router** to navigate from one scene to another when needed

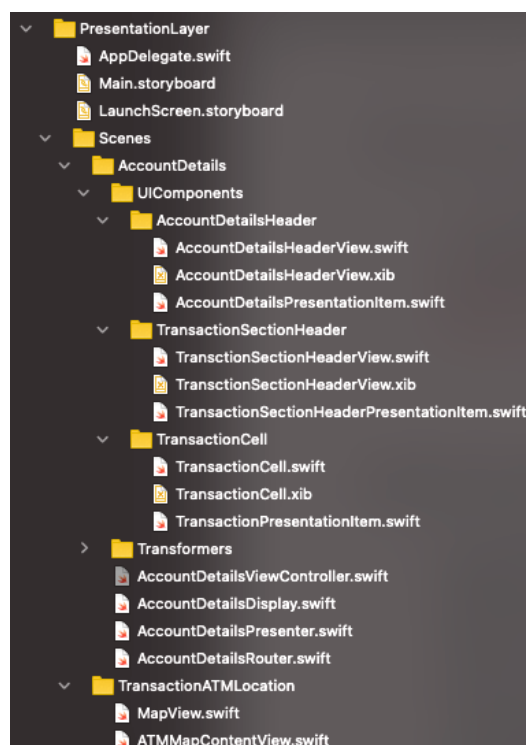
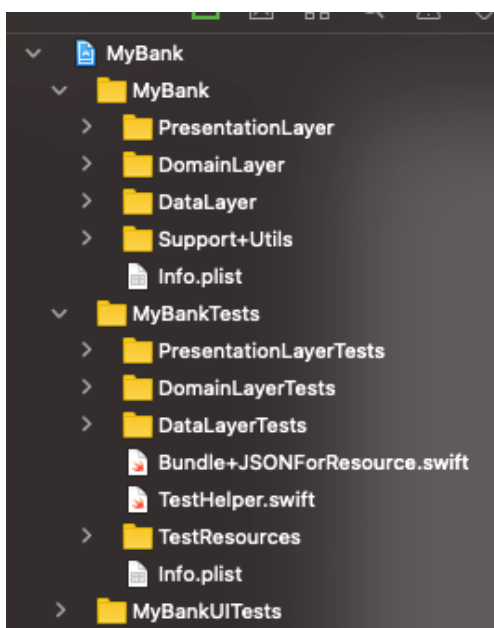
## Code Organisation / Layers / Grouping

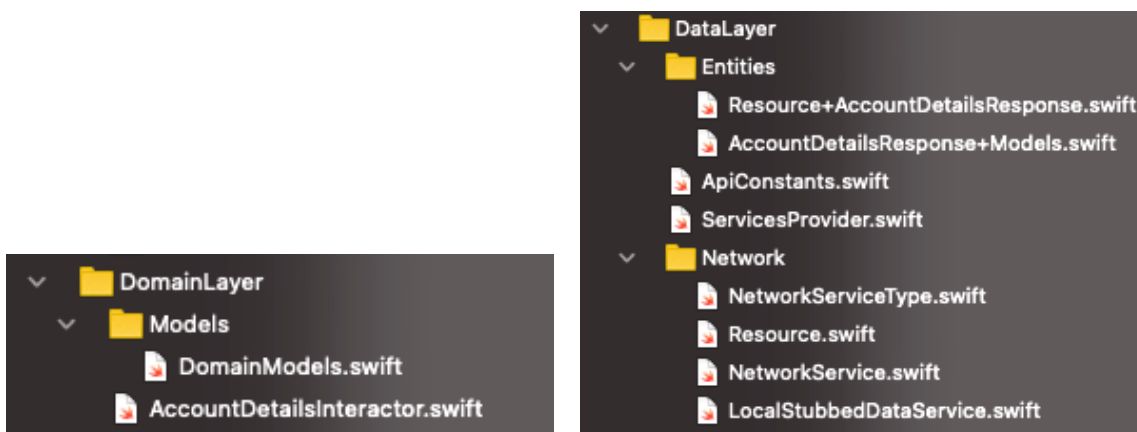
---

Folder / Grouping are done as per below:

Project has 3 targets:

- **MyBank** - The main code
- **MyBankTest** - **Unit testing** of all layers using Quick/Nimble
- **MyBankUITest** - Some automated XCUITest



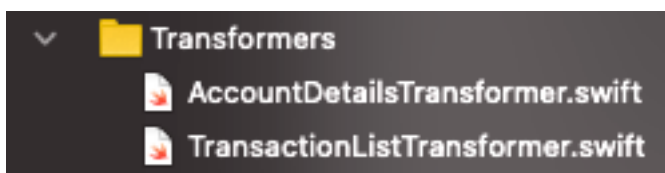


## Business Logic

---

The core business logic of handling the transactions, and grouping the into dates, combining cleared & pending transactions together are done inside the interactor. The low level JSON parsed data models are getting mapped as `[TransactionGroup]` and it has its own `TransactionModel` object that has extra properties and info needed by the presentation layer.

These domain level models get mapped into presentation level models at presentation layer. There are transformers which does the job to create presentation items needed exactly for the UI rendering, such as date formatting, currency formatting, or even bold text injection of "PENDING:" via `NSAttributedString`. They also inject custom accessibility elements such as combined labels needed for the UI.



## Unit Testing

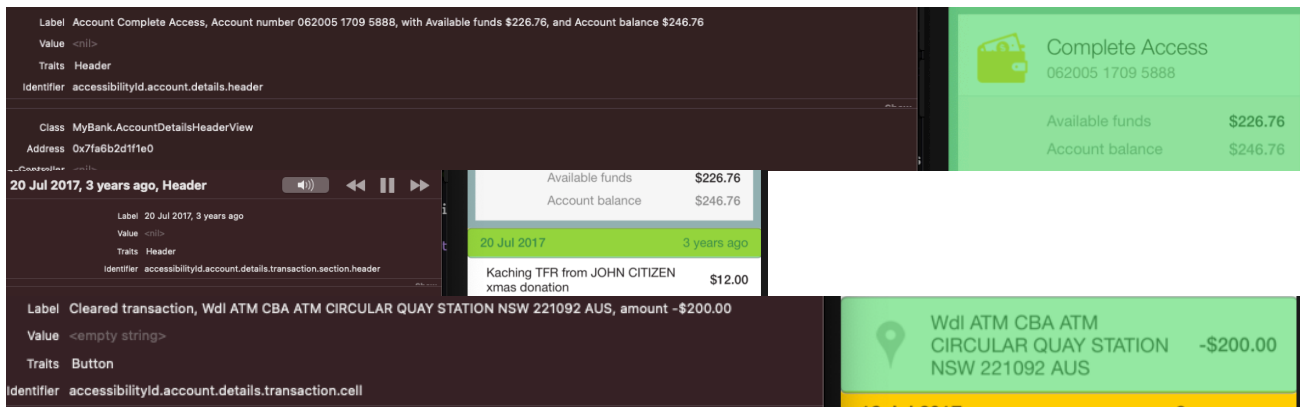
---

- Data Layer logic
- Domain Layer logic

- Presentation layer logic including presenter, display, router communication and transformers are tested individually

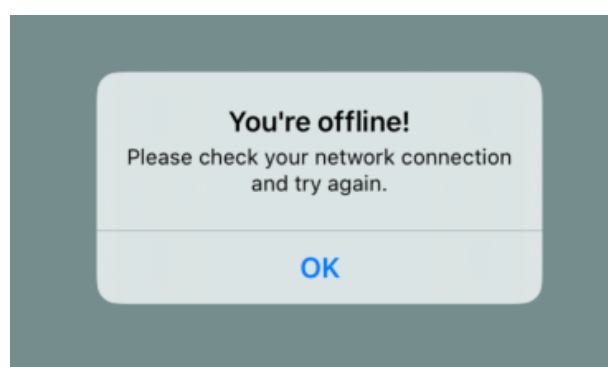
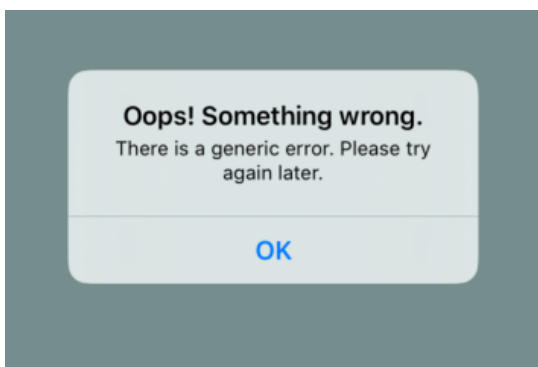
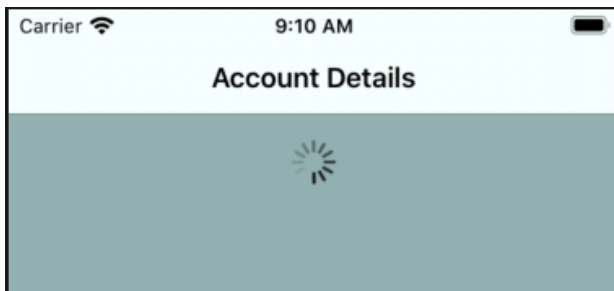
## Accessibility

- Custom accessibility to each element is supported
- They are computed via transformers which are also unit tested



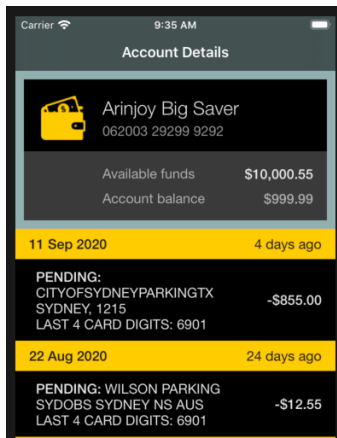
## Custom Error Handling

- The network layer implementation handles most of the known HTTP error codes
- More assumptions can be made in future based server returned code
- All errors fallback to show as generic error on the UI
- Network offline error shows differently with another message
- Presenter logic makes decision on how to handle what error (unit tested)
- There is also a loading indicator on the top and *Pull to refresh* is also supported



## Custom Theme

- All font & colours are used from a central Theme provider
- Colors have dark mode support
- Dynamic scaling of the fonts are supported



## Use of SwiftUI (experimental feature on map)

- A custom SwiftUI view is made for the ATM map
- This follows the coordinator pattern to talk to MapKit delegate
- A custom pin image has been attached to the annotation
- The code is kept very simple and not unit tested



