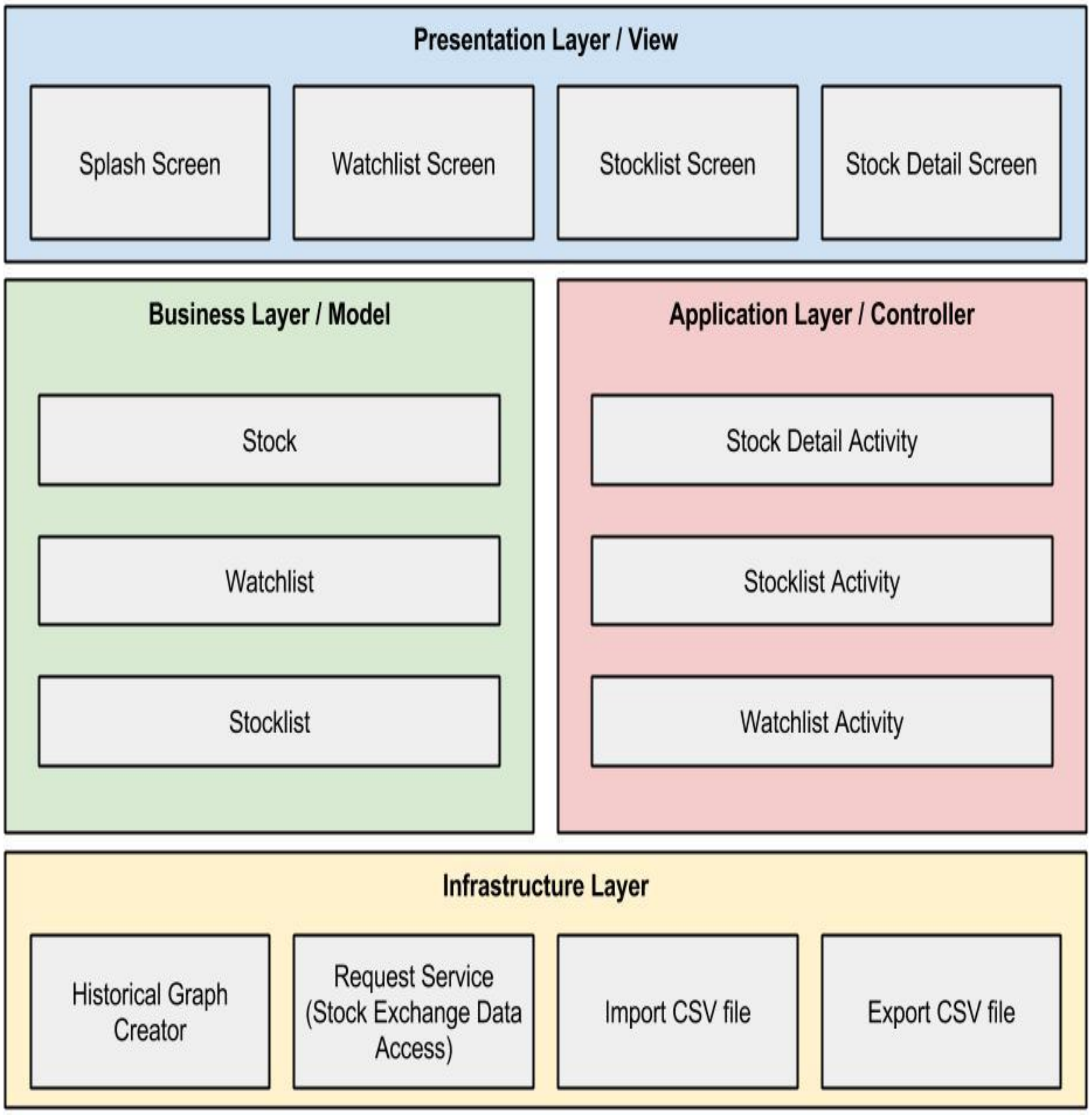
Team Name	**Sprinters**
Application Name	**Bullzx**
Client Reference	**Nick Lehtola**
Version	**1.0**
Status	**Ready for Review**

Table of Contents

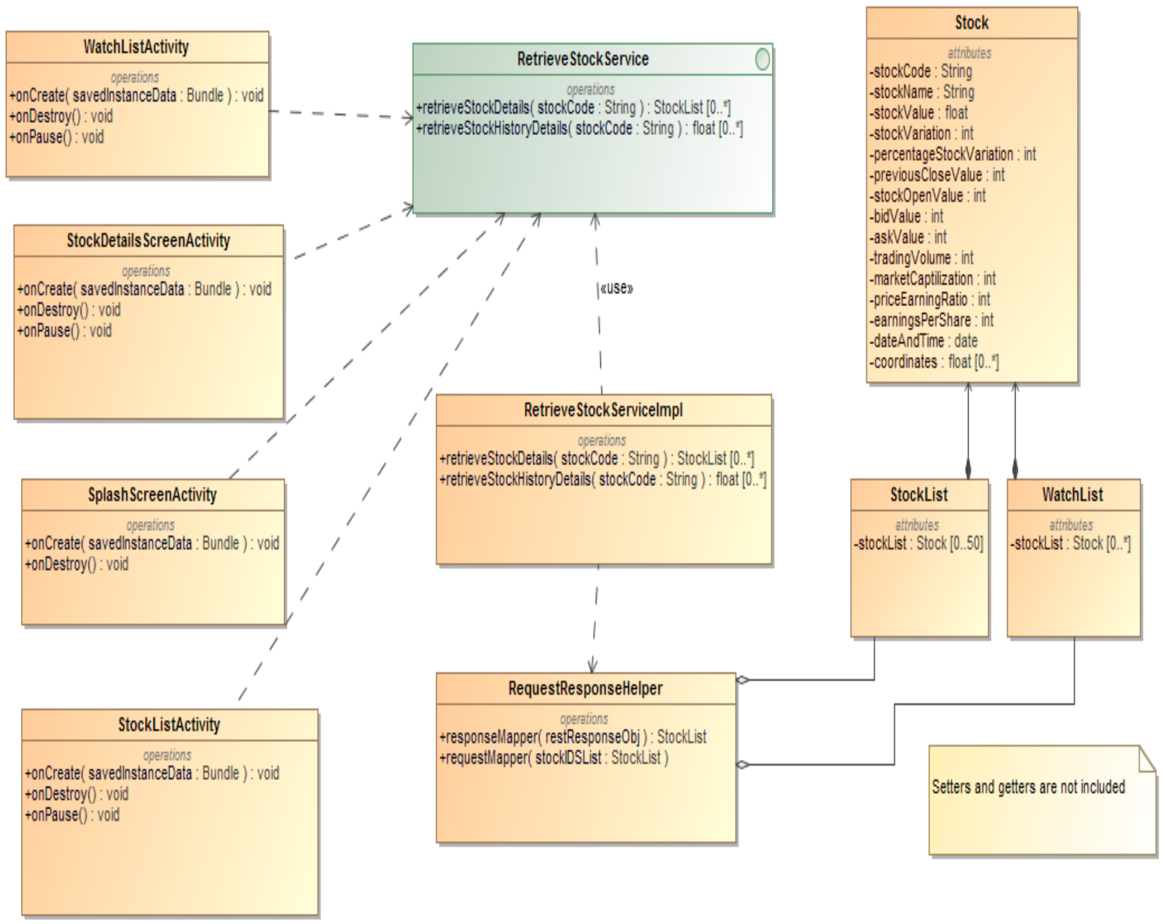
Software Architecture Diagram



ClassDiagram

The uml is attached below. The snapshot is attached for reference.

NewZealandStockExc
hange.xml


WatchListActivity
operations
+onCreate(savedInstanceData : Bundle) : void
+onDestroy() : void
+onPause() : void
RetrieveStockService
operations
+retrieveStockDetails(stockCode : String) : StockList [0..*]
+retrieveStockHistoryDetails(stockCode : String) : float [0..*]
Stock
attributes
-stockCode : String
-stockName : String
-stockValue : float
-stockVariation : int
-percentageStockVariation : int
-previousCloseValue : int
-stockOpenValue : int
-bidValue : int
-askValue : int
-tradingVolume : int
-marketCaptilization : int
-priceEarningRatio : int
-earningsPerShare : int
-dateAndTime : date
-coordinates : float [0..*]
StockDetailsScreenActivity
operations
+onCreate(savedInstanceData : Bundle) : void
+onDestroy() : void
+onPause() : void
«use»
RetrieveStockServiceImpl
operations
+retrieveStockDetails(stockCode : String) : StockList [0..*]
+retrieveStockHistoryDetails(stockCode : String) : float [0..*]
SplashScreenActivity
operations
+onCreate(savedInstanceData : Bundle) : void
+onDestroy() : void
StockList
attributes
-stockList : Stock [0..50]
WatchList
attributes
-stockList : Stock [0..*]
RequestResponseHelper
operations
+responseMapper(restResponseObj) : StockList
+requestMapper(stockIDSList : StockList)
StockListActivity
operations
+onCreate(savedInstanceData : Bundle) : void
+onDestroy() : void
+onPause() : void
Setters and getters are not included

There are four activity classes StockListActivity, SplashScreenActivity, StockDetailsScreenActivity and WatchListActivity corresponding to the screens in the application.

A Service interface 'RetrieveStockService' implementation is provided by 'RetrieveStockService' class in order to fetch data from an External System using a 'Rest' web service.

Stock class contains all the parameters of a stock and StockList and WatchList has the list of stock objects.

Design Decisions

Generic Objects:

The use of generic objects, such as the stock object we plan to implement, helps to increase extensibility since the stock objects could be used to represent a stock found on any exchange in the world.

Creating stock objects this way helps us increase Reusability, Portability and Modularity, since they are able to capture all the information about required to describe the current state of a stock, they can be used in many other applications.

Data Access:

Implementing a global service to access real time data increases extensibility by allowing the easy integration of other stock markets into the application.

Layered Architecture:

The use of a layered architecture increases the applications modularity since each layer is only dependent on the layer below. Also, each layer can be substituted in for another as long as it respects each layers access points.

In terms of our application, we have split it up into the Presentation Layer, which holds the views of the interface.

- The Application Layer which comprises of the activities of each screen.
- The Business Layer which has the objects that will store data gathered from the infrastructure layer.
- The Infrastructure Layer which has classes that perform functions for the business layer.

Model View Controller Pattern:

Using the MVC pattern increases the extensibility and modularity of the application in much the same way that the use of a layered architecture does. The MVC pattern helps separate functions found in each section of the pattern allowing them to be independent. This allows for the addition of different ways to gather data for the business layer as well as different implementations for certain classes found in the application by providing a view that isn't bound to a single implementation.

Model:

The model of our application is the business layer. These are the Stock, Watch list and Stock List classes.

View:

The interface screens that the user will interact with make up the view of our application. This includes the Splash screen, the Watch list screen, the Stock List screen, and the Stock Details screen.

Controller:

Each screen presented in a view will have its own activity which will act as the controller. Our application will implement a main controller that will call upon the screen's activity when required. These "controllers" will also call functions found in the applications infrastructure layer to help change the application's model.

Singleton Pattern:

The infrastructure layer which consists of calls to the web service is made singleton. This layer is stateless and one instance of it is sufficient throughout the application. This will save time to initiate the objects and wire them.

The Activity classes are also made singleton. So every time the user clicks on the application icon without closing a previously running version of the app, the same running instance of the app is returned, without opening a new application every time.

Android's Fragment:

Usage of fragments in the design of an activity enables a modular design. A particular fragment can be reused across multiple activities.
E.g.: Menu option is a Fragment and can be reused.

Generic Objects:

The use of generic objects, such as the stock object we plan to implement, helps to increase extensibility since the stock objects could be used to represent a stock found on any exchange in the world.

Creating stock objects this way helps us increase Reusability, Portability and Modularity, since they are able to capture all the information about required to describe the current state of a stock, they can be used in many other applications.

Data Access:

Implementing a global service such as Yahoo finance using REST API to access real time data increases extensibility by allowing easy integration of other stock markets into the application.

Optimal storage

Using internal storage of mobile only for user preferences and personalized watch list reduces storage overhead, increasing robustness, performance.

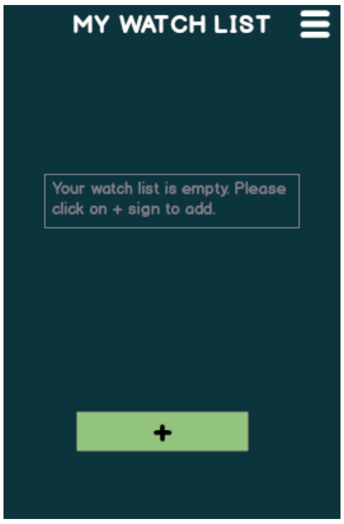
User Interface

The application opens up with a splash screen showcasing the application's logo. The major capabilities of the application are achieved through the following three screens

1. My Watch List screen
2. Stock List screen
3. Stock Detail screen

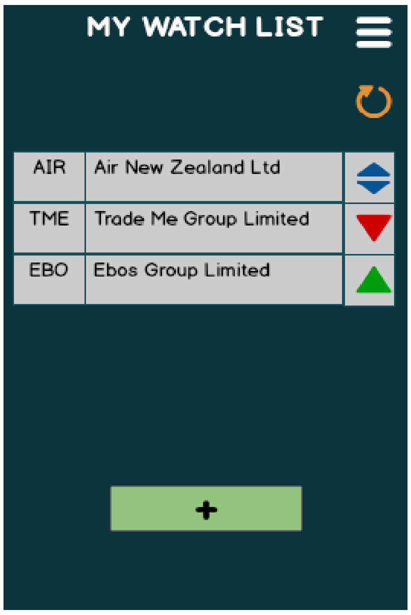
My Watch List Screen

The My Watch List screen contains no stocks listed in it for the first time. It has a '+' sign that will allow the users to look from NZX50 group of companies and add them to their watch list.

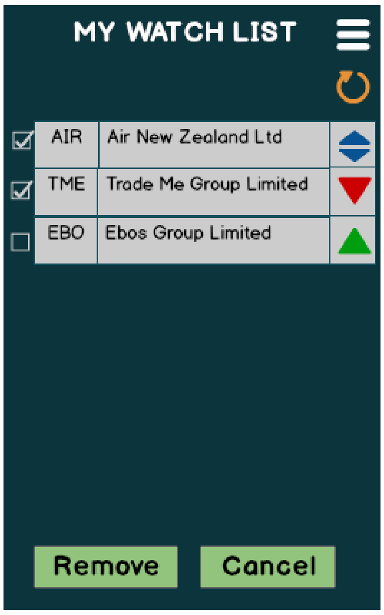


Your watch list is empty. Please click on + sign to add.

Once the user adds stock to the watch list it will be available in the user watch list. There is an option in the Menu at the top right from where the user can remove stocks from the watch list.



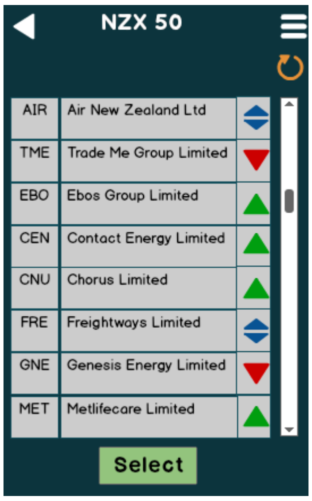
AIR	Air New Zealand Ltd	⬍
TME	Trade Me Group Limited	▼
EBO	Ebos Group Limited	▲



☑	AIR	Air New Zealand Ltd	⬍
☑	TME	Trade Me Group Limited	▼
☐	EBO	Ebos Group Limited	▲

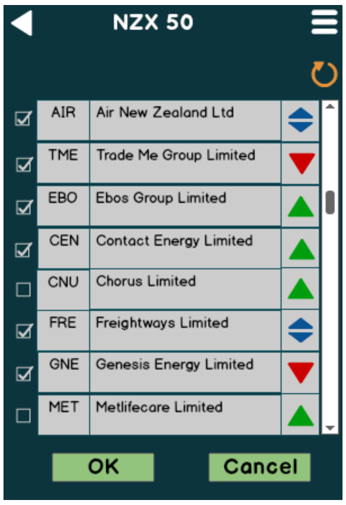
Stock List Screen

The Stock List Screen displays the list of all companies in the NZX 50 group of companies. This provides options to the user to add companies to their personalized watch list by clicking on the 'Select' button.



NZX 50

AIR	Air New Zealand Ltd	⬍
TME	Trade Me Group Limited	▼
EBO	Ebos Group Limited	▲
CEN	Contact Energy Limited	▲
CNU	Chorus Limited	▲
FRE	Freightways Limited	⬍
GNE	Genesis Energy Limited	▼
MET	Metlifecare Limited	▲



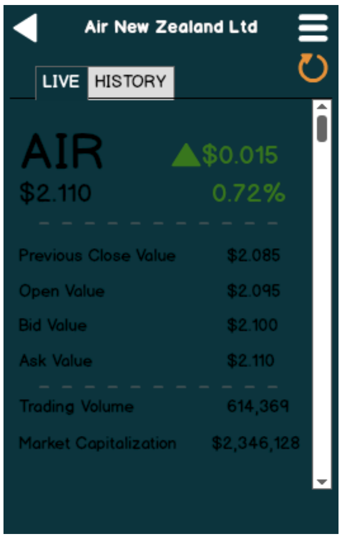
NZX 50

☑	AIR	Air New Zealand Ltd	⬍
☑	TME	Trade Me Group Limited	▼
☑	EBO	Ebos Group Limited	▲
☑	CEN	Contact Energy Limited	▲
☐	CNU	Chorus Limited	▲
☑	FRE	Freightways Limited	⬍
☑	GNE	Genesis Energy Limited	▼
☐	MET	Metlifecare Limited	▲

The users are allowed to view details of the particular stocks from the NZX 50 list.

Stock Detail Screen

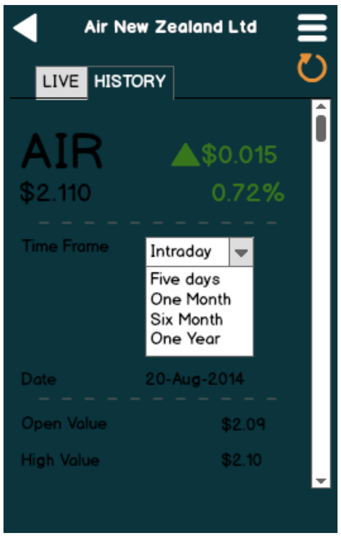
The Stock detail screen displays the real time and historical data of a particular stock. This screen is reachable from the user watch list screen and stock list screen.



Air New Zealand Ltd

AIR ▲$0.015
$2.110 0.72%

Previous Close Value	$2.085
Open Value	$2.095
Bid Value	$2.100
Ask Value	$2.110
Trading Volume	614,369
Market Capitalization	$2,346,128



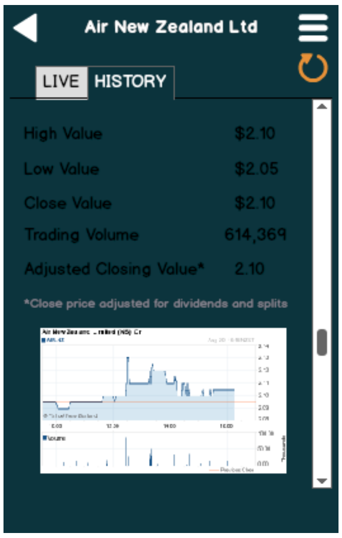
Air New Zealand Ltd

AIR ▲$0.015
$2.110 0.72%

Time Frame: Intraday

Date	20-Aug-2014
Open Value	$2.09
High Value	$2.10

s



Air New Zealand Ltd

High Value	$2.10
Low Value	$2.05
Close Value	$2.10
Trading Volume	614,369
Adjusted Closing Value*	2.10

*Close price adjusted for dividends and splits

Note: Refer *UX_NZX.pdf* for Interactive User Interface design templates.