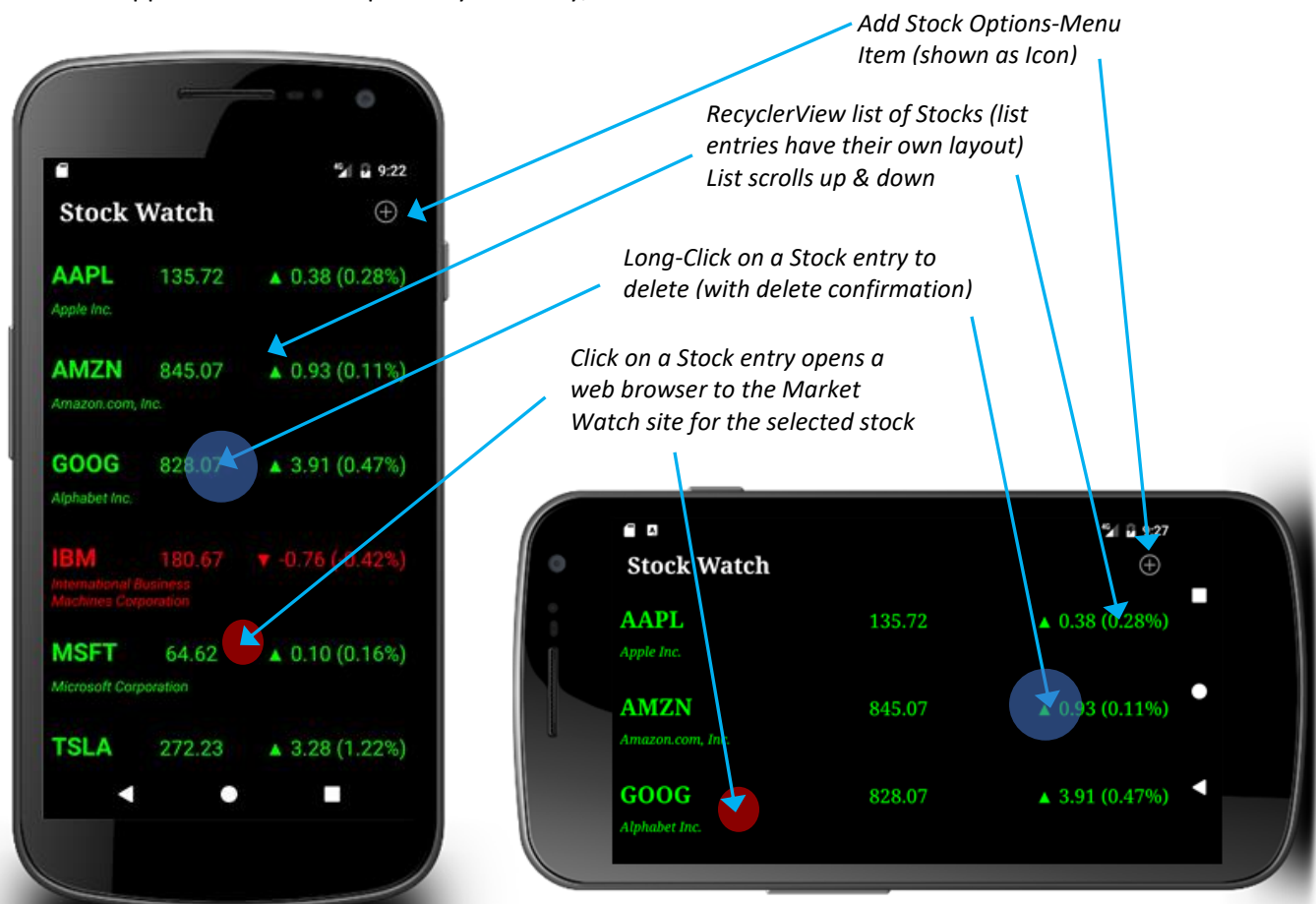


**CS 442: Mobile Applications Development****Assignment 3 – Stock Watch (300 pts)**

Uses: Internet, RecyclerView, Option-Menus, Multiple AsyncTasks,  
JSON Data, Swipe-Refresh, Dialogs, SQLite Database

**App Highlights:**

- This app allows the user to display a sorted list of selected stocks. List entries include the stock symbol (and company name), the current price, the daily price change amount and price percent change.
- There is no need to use a different layout for landscape orientation in this application – the same layout should work in any orientation.
- Selected stock symbols and the related names should be stored in the device's SQLite Database.
- A Stock class should be created to represent each individual stock in the application. Required data includes: Stock Symbol (String), Company Name (String), Price (double), Price Change (double), and Change Percentage (double).
- Clicking on a stock opens a browser displaying the *Market Watch* web page for that stock
- Swipe-Refresh (pull-down) refreshes stock data.
- The application is made up of only 1 activity, shown below:



## A) Internet Data:

Downloading data for a stock symbol requires 2 downloads – one download to acquire the full set or supported stock symbol and company names, and a second download to acquire the financial data for a particular stock.

### Download 1: Stock Symbol & Company Data

When started, your app should initiate a download of the full set or supported stock symbol and company names. This data is saved, and then used whenever the user adds a new stock.

**Download Source:** <https://api.iextrading.com/1.0/ref-data/symbols>

### Download Results Example:

Results are returned in JSON format, as a JSONArray containing the results data from the query. *The data we are interested in is the stock “symbol” and “name”.* Below is a sample of the JSON you will download :

```
[{
  "symbol": "A",
  "name": "Agilent Technologies Inc.",
  "date": "2018-09-21",
  "isEnabled": true,
  "type": "cs",
  "iexId": "2"
}, {
  "symbol": "AA",
  "name": "Alcoa Corporation",
  "date": "2018-09-21",
  "isEnabled": true,
  "type": "cs",
  "iexId": "12042"
}, {
  "symbol": "AAAU",
  "name": "Perth Mint Physical Gold",
  "date": "2018-09-21",
  "isEnabled": true,
  "type": "N/A",
  "iexId": "14924"
}, {
  "symbol": "AABA",
  "name": "Altaba Inc.",
  "date": "2018-09-21",
  "isEnabled": true,
  "type": "cs",
  "iexId": "7653"
}, {
  "symbol": "AAC",
  "name": "AAC Holdings Inc.",
  "date": "2018-09-21",
  "isEnabled": true,
  "type": "cs",
  "iexId": "9169"
}, {
  "symbol": "ICXUSD",
  "name": "ICON USD",
  "date": "2018-09-21",
  "isEnabled": true,
  "type": "crypto",
  "iexId": "10000013"
}, {
  "symbol": "NEOUSD",
  "name": "NEO USD",
  "date": "2018-09-21",
  "isEnabled": true,
  "type": "crypto",
  "iexId": "10000014"
}, {
  "symbol": "VENUSD",
  "name": "VeChain USD",
  "date": "2018-09-21",
  "isEnabled": true,
  "type": "crypto",
  "iexId": "10000015"
}, {
  "symbol": "XLMUSD",
  "name": "Stellar Lumens USD",
  "date": "2018-09-21",
  "isEnabled": true,
  "type": "crypto",
  "iexId": "10000016"
}, {
  "symbol": "QTUMUSD",
  "name": "Qtum USD",
  "date": "2018-09-21",
  "isEnabled": true,
  "type": "crypto",
  "iexId": "10000017"
}
]
```



## Download 2: Stock Financial Data

When you have the desired stock symbol (and company name), you use the *stock symbol* to download financial data for that stock. For this you will need an API key. You can get this by registering at:

<https://iexcloud.io/cloud-login#/register>

- Go to: <https://iexcloud.io/cloud-login#/register>
- Select "Individual"
- Enter your name, email address, and select a password.
- Click the terms checkbox and click "Create account"
- Select the "START" plan (0/mo) - click Select Start
- Go to your email - look for message with Subject "IEX Cloud Email Verification"
- Click the iexcloud.io link in the email.
- From the page that link opens, click "API Tokens" in the upper-left (under Home)

**Query Format:** [https://cloud.iexapis.com/stable/stock/STOCK\\_SYMBOL/quote?token=API\\_KEY](https://cloud.iexapis.com/stable/stock/STOCK_SYMBOL/quote?token=API_KEY)

For example, if the selected stock symbol was TSLA and your API token was pk\_123abc, then your full URL would be: [https://cloud.iexapis.com/stable/stock/TSLA/quote?token=pk\\_123abc](https://cloud.iexapis.com/stable/stock/TSLA/quote?token=pk_123abc)

## Download Results:

Results are returned in JSON format, as a JSONObject containing the results data from the query. The data we are interested in is highlighted below (Example using search text "TSLA"):

```
{
  "symbol": "TGT",
  "companyName": "Target Corp.",
  "primaryExchange": "New York Stock Exchange",
  "calculationPrice": "previousclose",
  "open": null,
  "openTime": null,
  "close": null,
  "closeTime": null,
  "high": null,
  "low": null,
  "latestPrice": 116.63,
  "latestSource": "Previous close",
  "latestTime": "February 14, 2020",
  "latestUpdate": 1581656400000,
  "latestVolume": null,
  "iexRealtimePrice": null,
  "iexRealtimeSize": null,
  "iexLastUpdated": null,
  "delayedPrice": null,
  "delayedPriceTime": null,
  "oddLotDelayedPrice": null,
  "oddLotDelayedPriceTime": null,
  "extendedPrice": null,
  "extendedChange": null,
  "extendedChangePercent": null,
  "extendedPriceTime": null,
  "previousClose": 116.63,
  "previousVolume": 3289074,
  "change": 0,
  "changePercent": 0,
  "volume": null,
  "iexMarketPercent": null,
  "iexVolume": null,
  "avgTotalVolume": 5489347,
  "iexBidPrice": null,
  "iexBidSize": null,
  "iexAskPrice": null,
  "iexAskSize": null,
  "marketCap": 59100736310,
  "peRatio": 18.48,
  "week52High": 130.24,
  "week52Low": 70.03,
  "ytdChange": -0.0941,
  "lastTradeTime": 1581714000046,
  "isUSMarketOpen": false
}
```

If the requested stock symbol does not exist, the following is returned: (Example using search text "ZZZZZ"):

*Response Code: 404 Not Found*

The **symbol**, **companyName**, **latestPrice**, **change** & **changePercentage** make up the data for one stock. Your code should parse these 5 data elements.

Using these 5 data elements, you can create a Stock object with all data reflecting the user's choice.

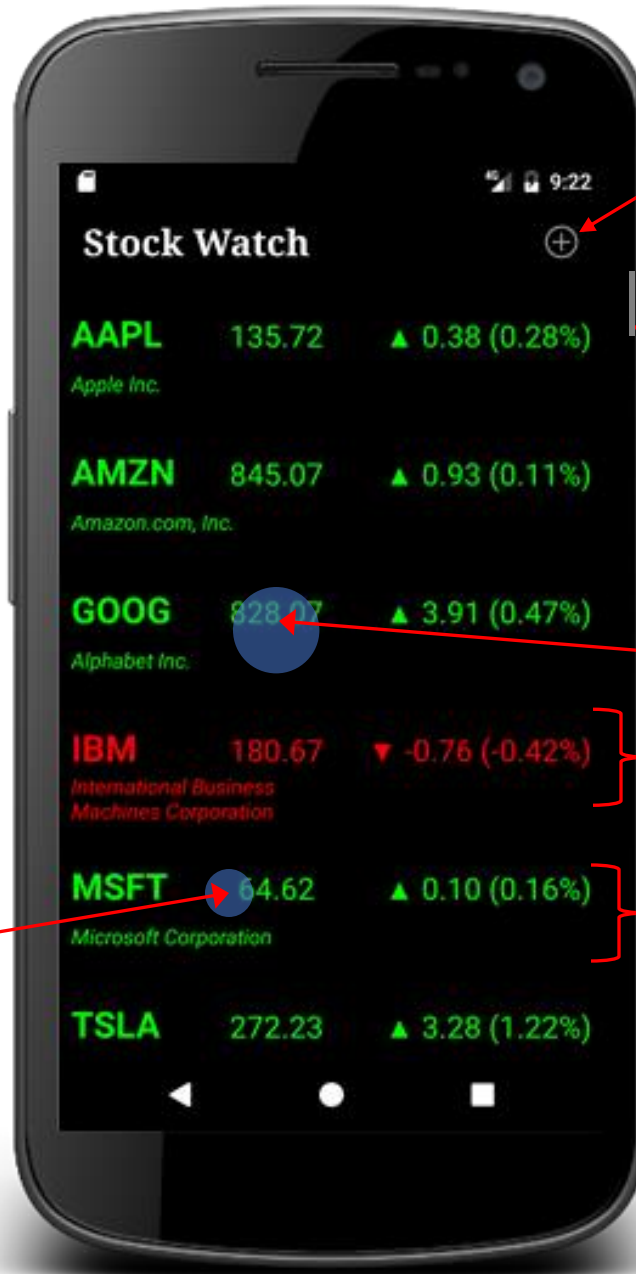
## B) Application Behavior Diagrams:

### 1) App MainActivity

Each stock entry contains the Stock Symbol (i.e., AAPL), the company name (i.e., Apple Inc.), the Last Trade Price (135.72), the price change direction (▲ for positive Price Change Amount, ▼ for negative Price Change Amount), the Price Change Amount (0.38), and the Price Change Percentage (0.28%) in parentheses.

If the stock's Price Change Amount is a positive value, then entire entry should use a green font. If the Price Change Amount is a negative value, then entire entry should use a red font.

Clicking on a Stock entry opens a web browser to the Market Watch website site for the selected stock



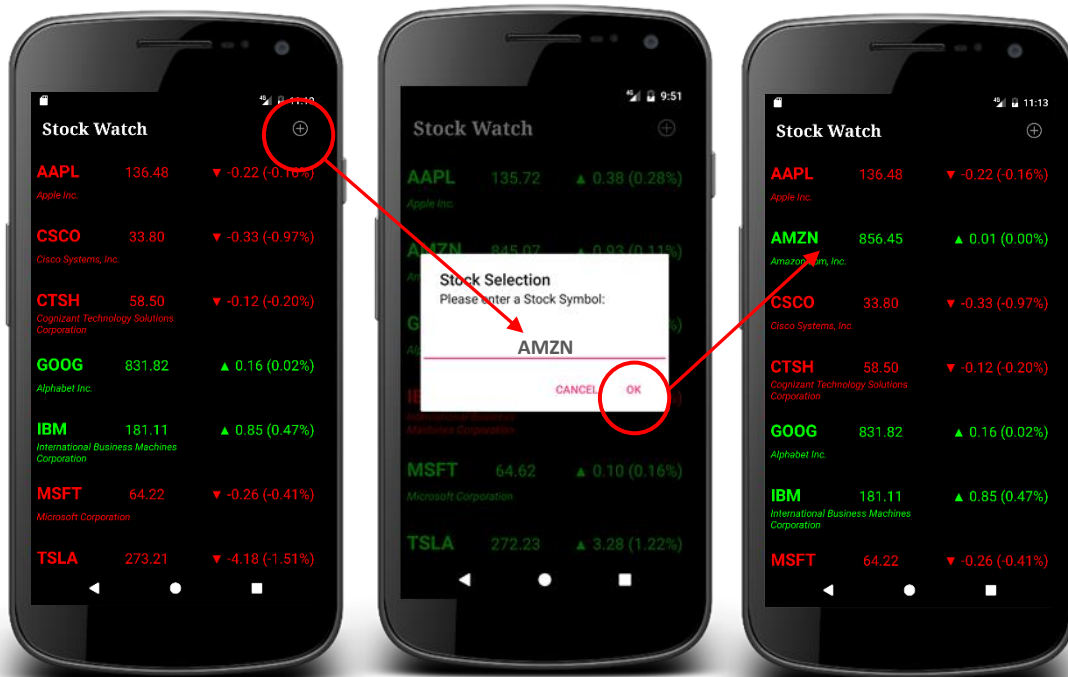
"Add Stock" action is an Options-Menu with a single item (shown as Icon)

A scrollbar should be present along the right-hand side

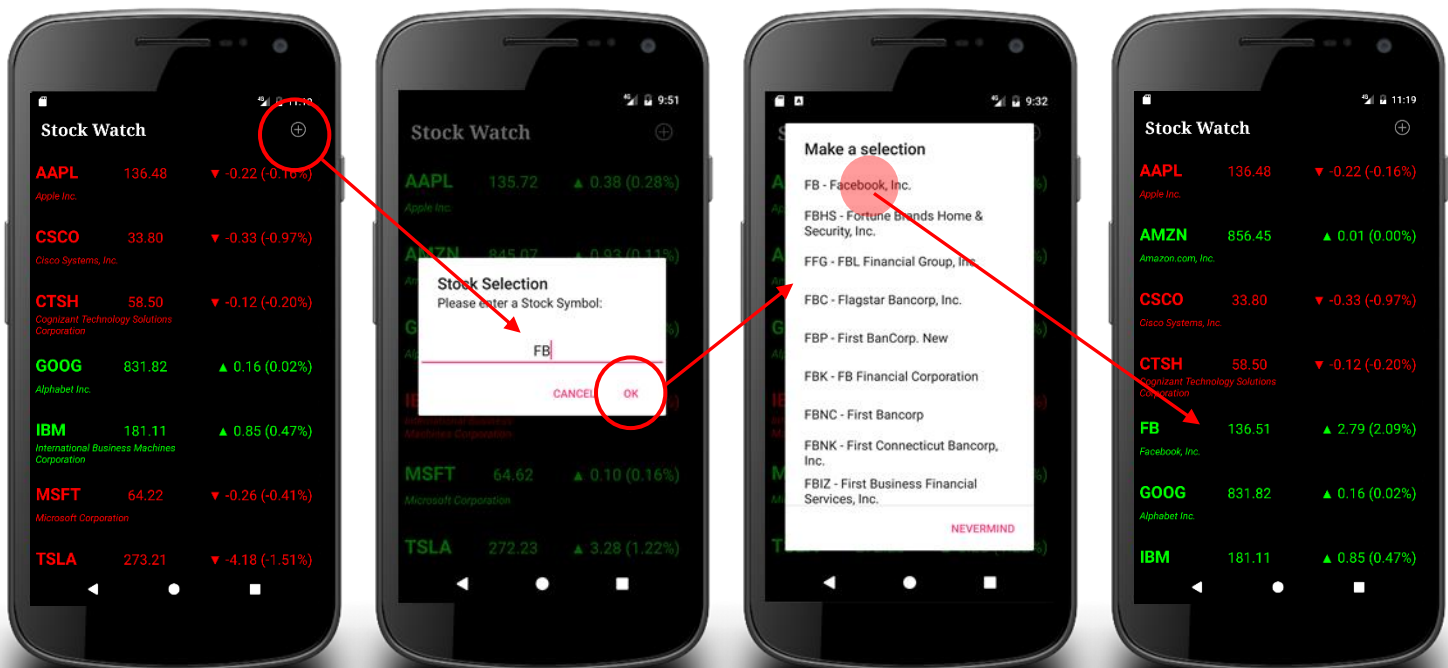
Long-Click on a Stock entry to delete (with delete confirmation)

RecyclerView list entries have their own layout

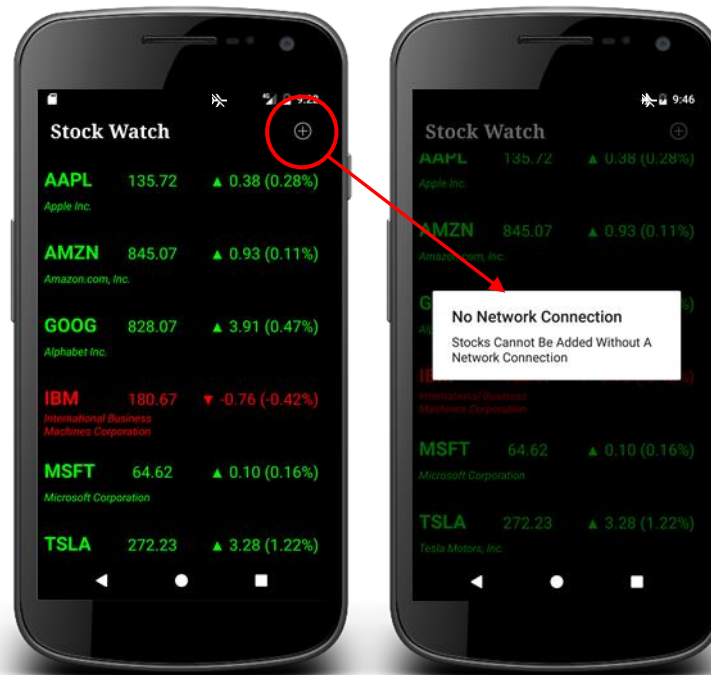
- 1) Adding a stock – when only **one stock matches** the search symbol/name search string (*NOTE: The Stock Selection dialog should only allow capital letters*):



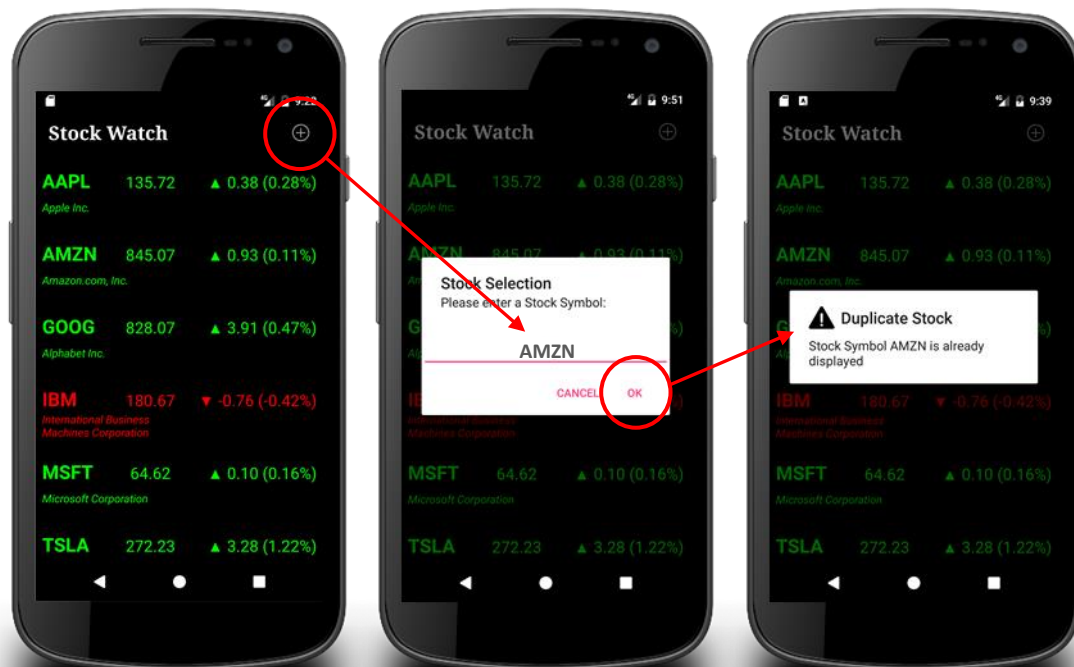
- 2) Adding a stock – when **multiple stocks matched** the search string (*Stock Selection dialog should only allow capital letters, stock selection dialog should display the stock symbol and company name*):



- 3) Adding a stock with no Network Connection – test using “Airplane Mode” (No buttons on the error dialog):

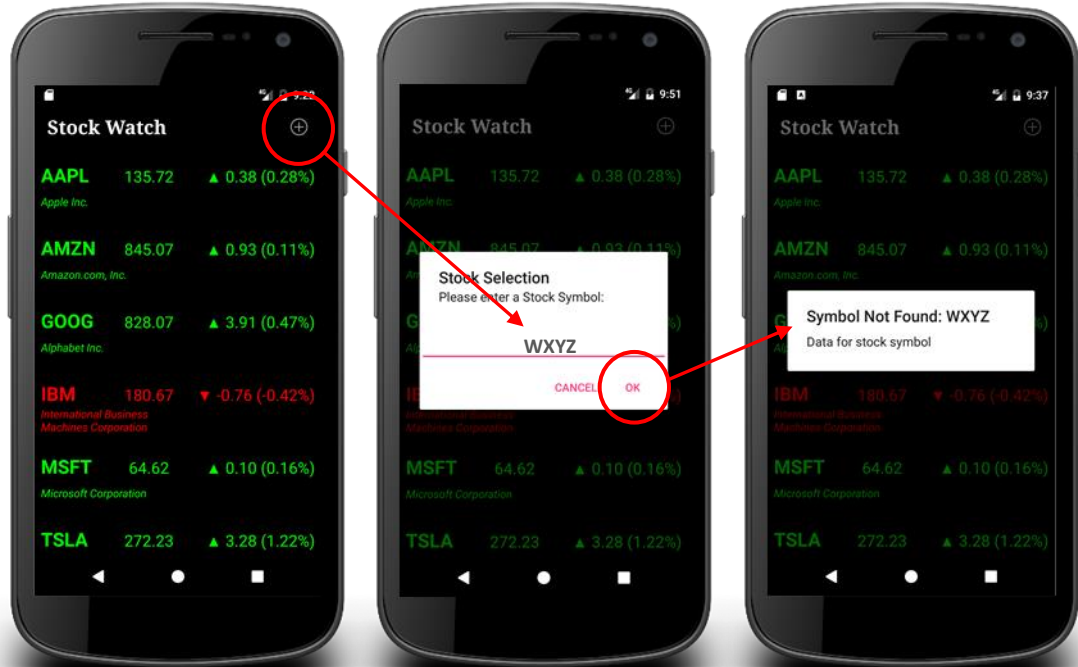


- 4) Adding a stock – specified stock is a duplicate (Stock Selection dialog should only allow capital letters, No buttons on the warning dialog):

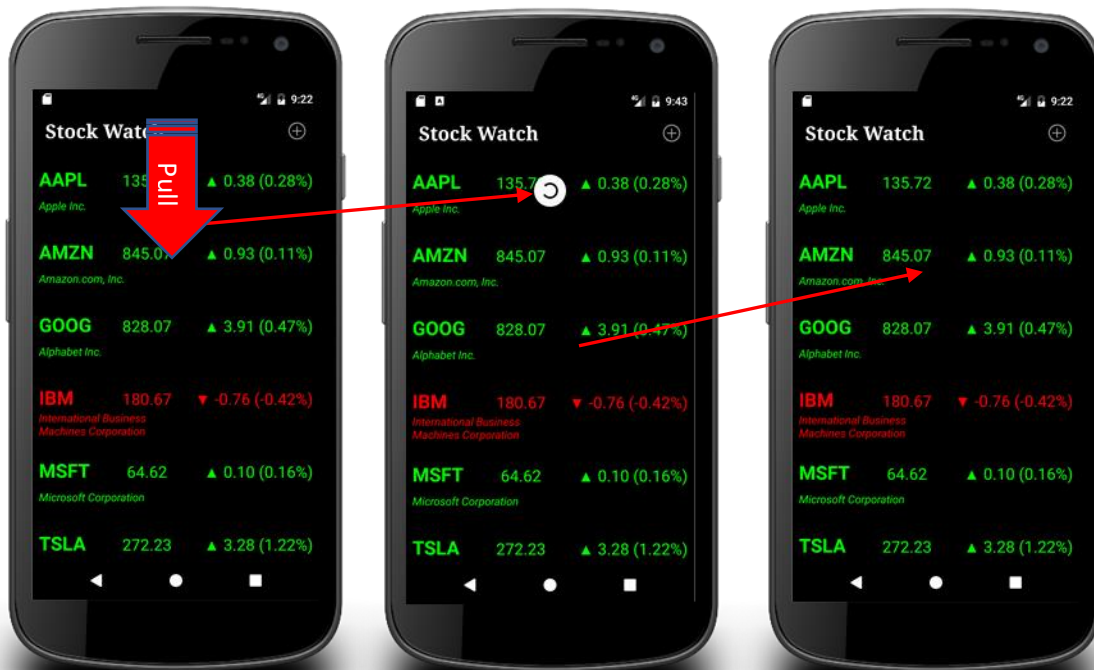




- 1) Adding a stock – specified stock is not found (*Stock Selection dialog should only allow capital letters, No buttons on the dialog*):

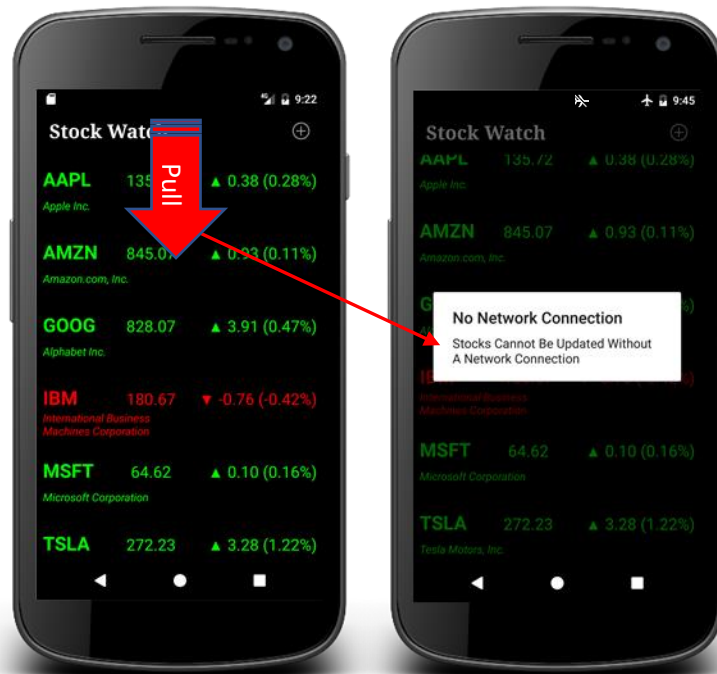


- 6) Swipe-Refresh (pull-down) reloads (re-downloads) all stock data:

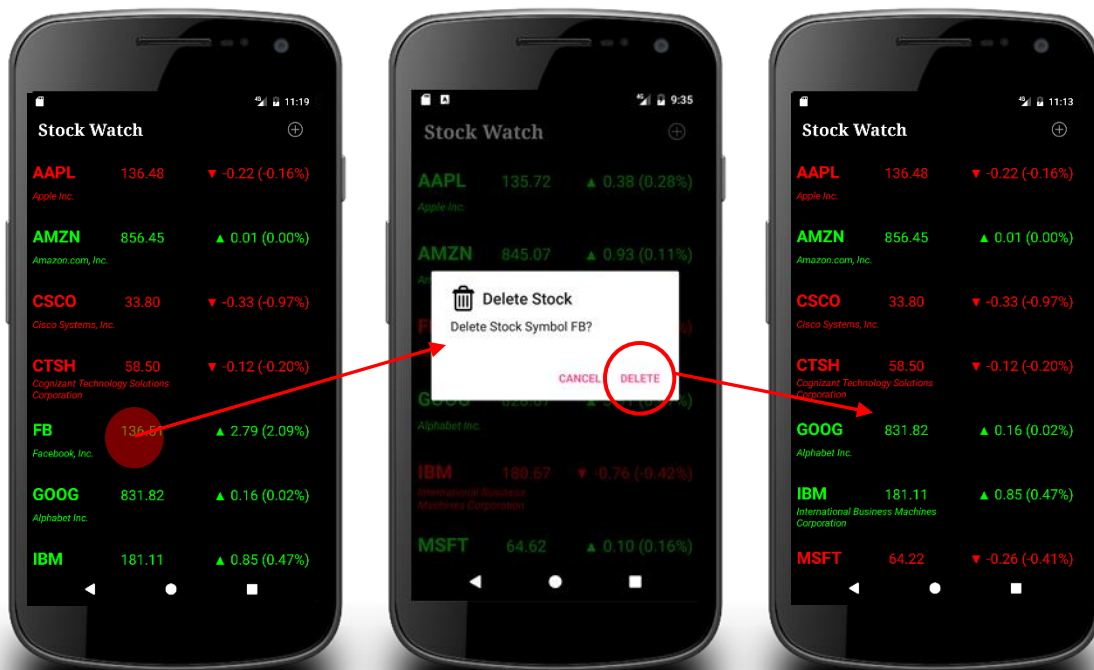




7) Swipe-Refresh attempt with no network connection (*No buttons on the error dialog*):



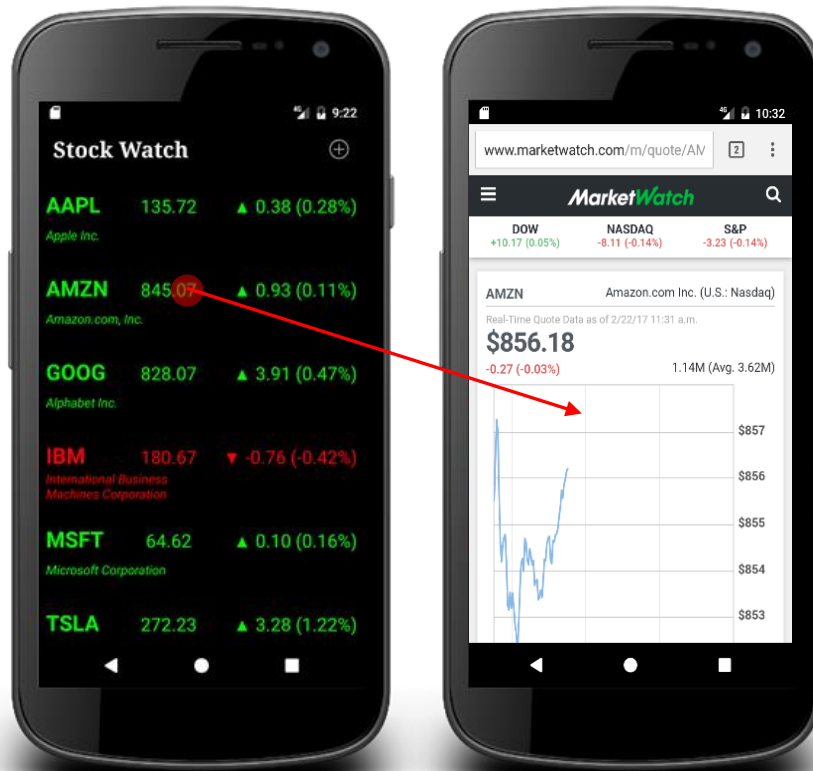
8) Long-Press on a stock to delete it:



9) Tap on a stock to open the *MarketWatch.com* website entry for the selected stock:

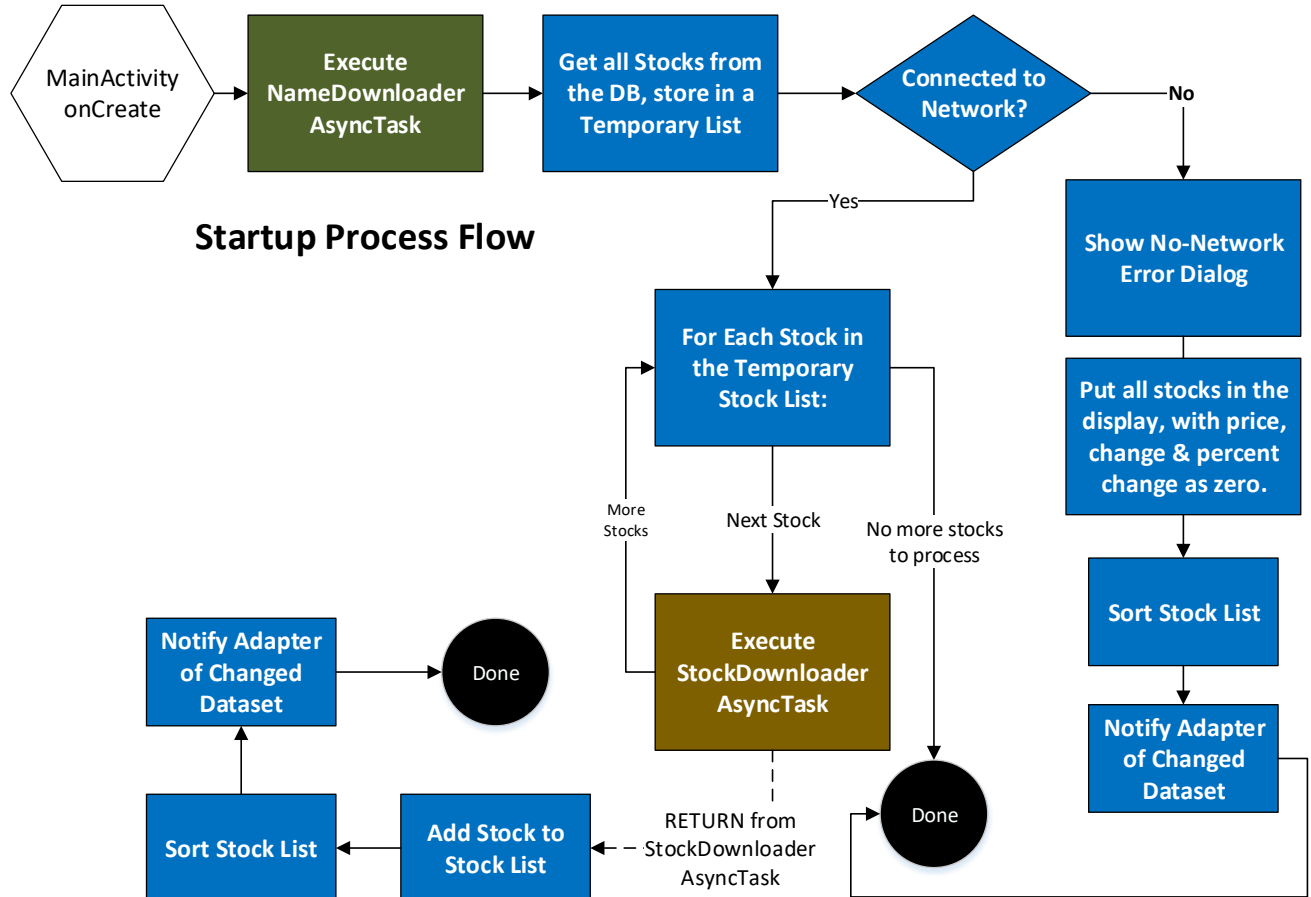
MarketWatch URL's are in the form: [http://www.marketwatch.com/investing/stock/some\\_stock](http://www.marketwatch.com/investing/stock/some_stock)

For example: <http://www.marketwatch.com/investing/stock/TSLA>

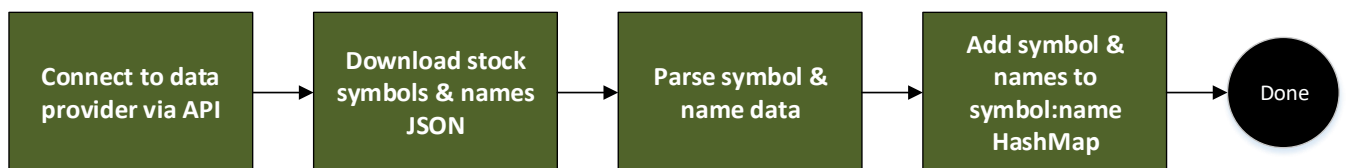


C) Application Behavior Flowcharts:

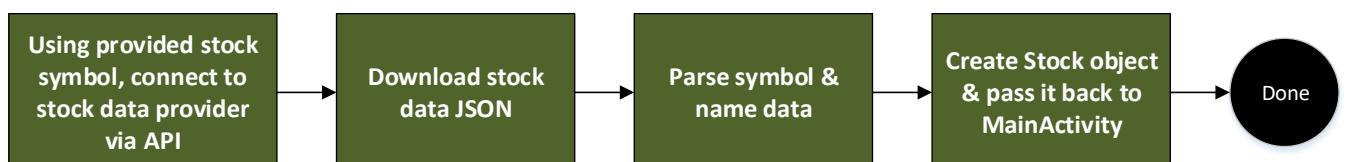
a) App Startup



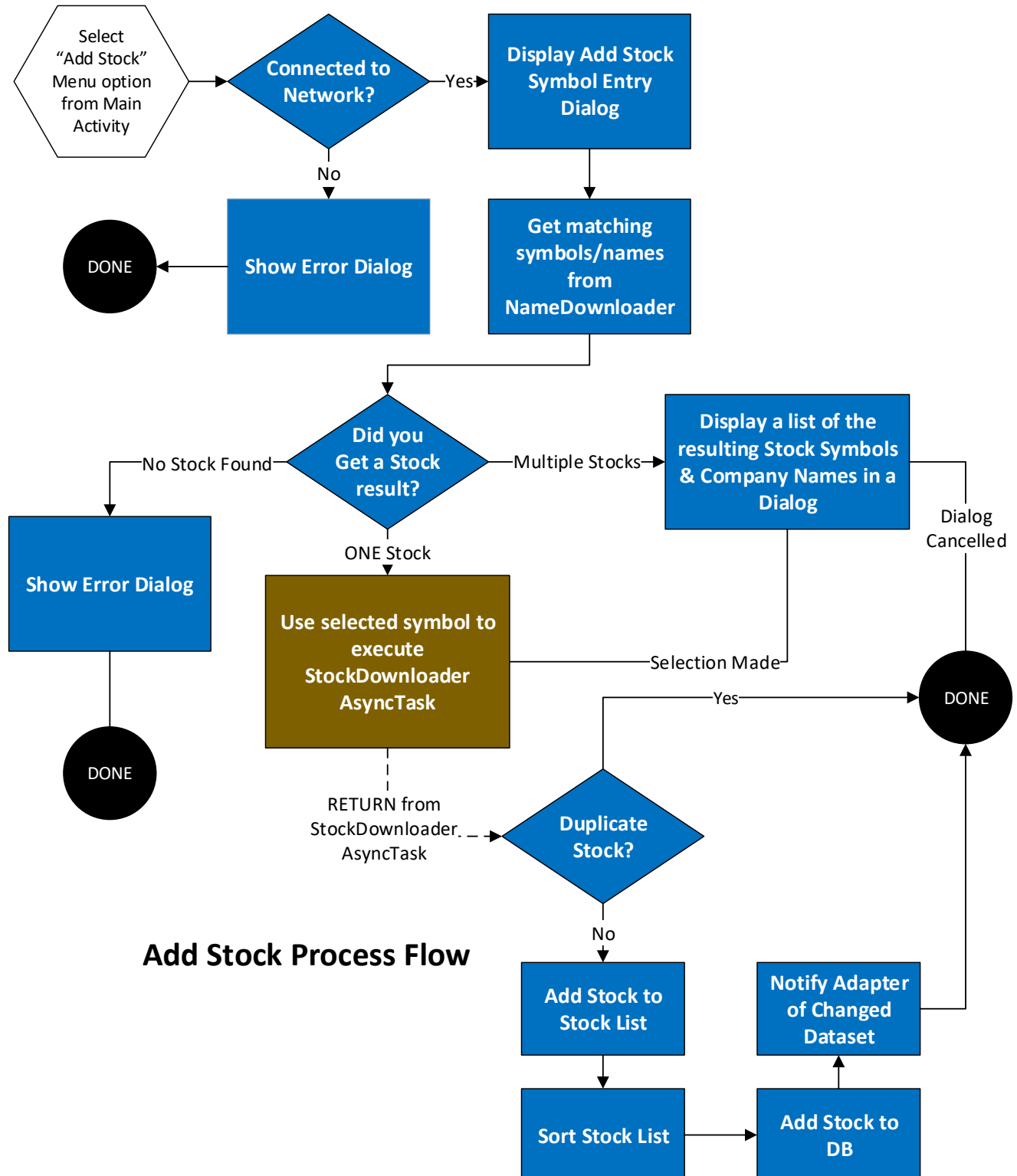
b) NameDownloader (AsyncTask)



c) StockDownloader (AsyncTask)

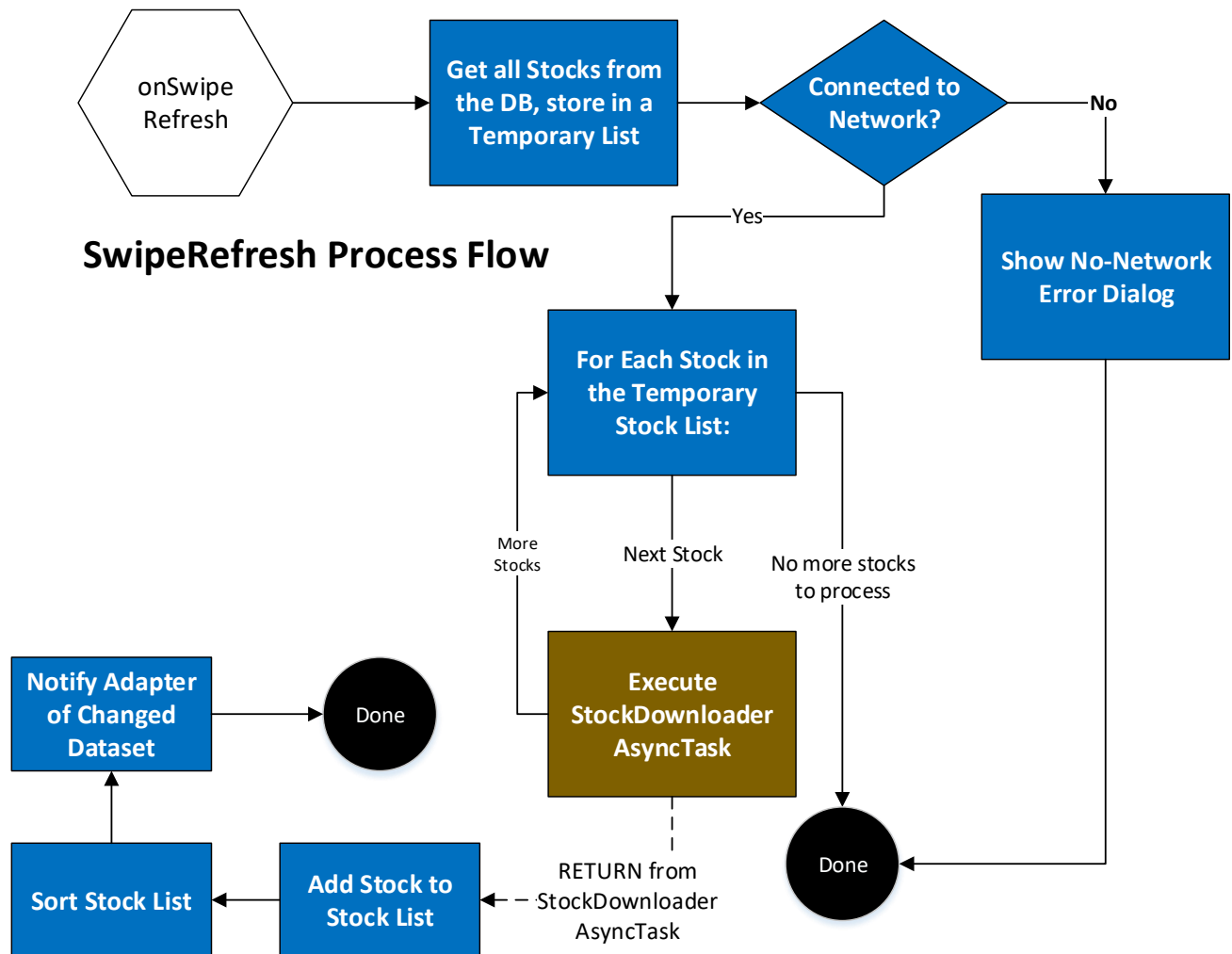


d) Add New Stock Process

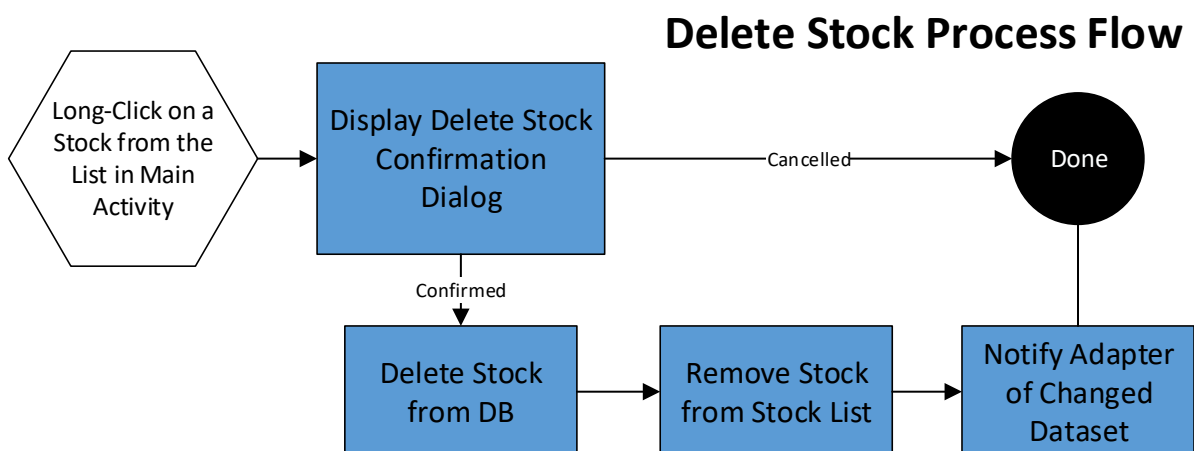


Add Stock Process Flow

e) Swipe-Refresh (pull-down) List:



f) Long-Press Delete Stock:





## D) Database

Your application must store the Stock Symbol and Company Name in the android device's SQLite database. You will need a Database handler class as we have done in class (you have a posted example of a Database handler that you can use as a model).

StockWatchTable	
StockSymbol	CompanyName
AAPL	Apple, Inc
AMZN	Amazon.com, Inc.
...	...

```
private static final String DATABASE_NAME = "StockAppDB";
private static final String TABLE_NAME = "StockWatchTable";
private static final String SYMBOL = "StockSymbol";
private static final String COMPANYY = "CompanyName";
```

- DB creation (done in onCreate):

```
CREATE TABLE TABLE_NAME (
    SYMBOL TEXT not null unique,
    COMPANYY TEXT not null)
```

- DB Add (Sample method to add a stock to the DB):

```
public void addStock(Stock stock) {
    Log.d(TAG, "addStock: Adding " + stock.getSymbol());

    ContentValues values = new ContentValues();
    values.put(SYMBOL, stock.getSymbol());
    values.put(COMPANYY, stock.getCompany());

    database.insert(TABLE_NAME, null, values);

    Log.d(TAG, "addStock: Add Complete");
}
```

- DB Delete (Sample method to delete a stock from the DB):

```
public void deleteStock(String symbol) {
    Log.d(TAG, "deleteStock: Deleting Stock " + symbol);

    int cnt = database.delete(
        TABLE_NAME, "SYMBOL = ?", new String[] { symbol });

    Log.d(TAG, "deleteStock: " + cnt);
}
```





- DB Load All (Sample method to get all stock-company entries from the DB):

```
public ArrayList<String[]> loadStocks() {  
  
    ArrayList<String[]> stocks = new ArrayList<>();  
  
    Cursor cursor = database.query(  
        TABLE_NAME, // The table to query  
        new String[]{SYMBOL, COMPANY}, // The columns to return  
        null, // The columns for the WHERE clause  
        null, // The values for the WHERE clause  
        null, // don't group the rows  
        null, // don't filter by row groups  
        null); // The sort order  
  
    if (cursor != null) {  
        cursor.moveToFirst();  
  
        for (int i = 0; i < cursor.getCount(); i++) {  
            String symbol = cursor.getString(0);  
            String company = cursor.getString(1);  
            stocks.add(new String[]{symbol, company});  
            cursor.moveToNext();  
        }  
        cursor.close();  
    }  
  
    return stocks;  
}
```

## E) Development Plan

- 1) Create the base app:
  - a. MainActivity with RecyclerView & SwipeRefreshLayout
  - b. Stock Class
  - c. RecyclerView Adapter
  - d. RecyclerView ViewHolder
  - e. Create fake “dummy” stocks to populate the list in the MainActivity onCreate.
  - f. Add the onClick and onLongClick methods. The onLongClick should delete an entry. The onClick can open a Toast message for now.
  - g. Add Stock options-menu opens dialog, on confirmation you can open a Toast message for now.
  - h. SwipeRefresh callback method can open a Toast message for now.
  
- 2) Add the database elements:
  - a. Create the database handler.
  - b. Add database handler calls to MainActivity (load, add, delete)
  - c. Code the onClick method to open the browser to the stock’s Market Watch site.
  
- 3) Add the internet elements and final integration:
  - a. Create the Stock Symbol - Company Name downloader/parser AsyncTask (<http://d.yimg.com/aq/autoc>)
  - b. Create the Stock Financial Data downloader/parser AsyncTask ([finance.google.com](http://finance.google.com))
  - c. Add a method to MainActivity that allows the Stock Symbol - Company Name downloader/parser AsyncTask to send the newly downloaded Stock Symbol & Company Name data back to MainActivity. This method should create and execute the Stock Financial Data downloader/parser AsyncTask.
  - d. Add a method to MainActivity that allows the Stock Financial Data downloader/parser AsyncTask to send the newly created Stock back to MainActivity.
  - e. Implement the Add Stock feature (this uses the results of the above tasks)
  - f. Implement the SwipeRefresh callback to re-download the Stock Financial Data for the loaded stocks
  - g. Add alerts when startup, add, & refresh are attempted when no internet connection is available.

### Assignment Assistance

The TAs for our course is available to assist you with your assignment if needed. Questions on assignment requirements and course concepts can be sent to the instructor.

### Submissions & Grading

- 1) Submissions must consist of your zipped project folder (*please execute Build =>Clean Project before generating the zip file*).
- 2) Submissions should reflect the concepts and practices we cover in class, and the requirements specified in this document.
- 3) Late submissions will be penalized by 10% per class late. (i.e., from one second late to 1 class late: 10% penalty, from one class plus one second late to 2 classes late: 20% penalty, etc.).
- 4) Grading will be based upon the presence and proper functionality of *all features and behaviors* described in this document.

### NOTE

**This assignment is worth 300 points. This means (for example) that if you get 89% on this assignment, your recorded score will be:**

**(89% \* 300 points = 267 points)**

***Note that this also means that the 10% late submission penalty will be 10% \* 300 points = 30 points.***

*If you do not understand anything in this handout, please ask.*

*Otherwise the assumption is that you understand the content.*

***Unsure? Ask!***