
Software Design Document

Term 3 Project: Business Intelligence System

Author: John Burchell

Version 2.0

Table of Contents

| | |
|--|----|
| 1 Revision History..... | 3 |
| 2 Introduction..... | 4 |
| 2.1 Purpose..... | 4 |
| 2.2 Scope..... | 4 |
| 2.3 Intended Audience..... | 4 |
| 3 Overview..... | 4 |
| 4 System Architecture..... | 5 |
| 4.1 Desktop..... | 5 |
| 4.2 Android..... | 5 |
| 4.3 Website..... | 5 |
| 4.4 ETL..... | 5 |
| 5 System Design..... | 6 |
| 6 Data Design..... | 7 |
| 6.1 Overview..... | 7 |
| 6.2 Database..... | 7 |
| 6.2.1 Interaction..... | 9 |
| 6.3 JSON Structure..... | 9 |
| 7 User interface Design..... | 10 |
| 8 Design Justification and Comments..... | 10 |
| 8.1 ETL – Overview..... | 10 |
| 8.2 ETL – Version 1..... | 10 |
| 8.3 ETL – Version 2..... | 10 |
| 8.4 Google Action Script..... | 11 |
| 8.5 Yahoo Finance..... | 12 |
| 8.6 Markitondemand..... | 12 |
| 9 SCM Information..... | 12 |
| 10 References..... | 12 |
| 11 Glossary..... | 12 |

Software Design Document

1 Revision History

| Author | Changes | Revision | Date |
|---------------|---|----------|------------|
| John Burchell | First Draft | 1.0 | 2013-12-16 |
| John Burchell | Included a new paragraph, JSON structure. | 1.1 | 2013-12-16 |
| John Burchell | Incorporated new template based on IEEE standard. | 2.0 | 2013-12-17 |

2 Introduction.

This document is written to the standards of the [IEEE 1016](#) standard. Unfortunately the standard is blocked behind a pay wall by the IEEE and as such I've had to use an alternative version along with templates provided by external sources. These sources can be found in the References section with their respective URLs.

2.1 Purpose.

The purpose of this document is to show the design and design decisions that went into creating the Adapa Stock Watcher (ASW). The document is to be read in conjunction with other documents created by the development team. The respective pages are listed throughout the document in the relevant sections. It also describes some information regarding our source control system.

2.2 Scope.

The scope of this document is to show how the software was designed, a brief overview of the architecture and user interfaces while including design decisions made regarding the data the application uses.

2.3 Intended Audience

The intended audience of this document are the stakeholders who can and will use this document during or post development.

3 Overview.

The system is documented in much greater detail in the quality management report. There is also information included in the Project Plan and Evaluation document. These documents include the requirements, aims and objectives of the project as a whole.

As a brief summary, the ASW is intended to be an application which fetches stock data hourly. The idea being that someone interested in his or her stocks can use the interfaces to build their own portfolio of stocks that they are interested in.

4 System Architecture.

The whole system architecture is covered in much greater detail in the SAD documented. Please refer to this for further in depth information about each module.

Below is a brief description and architectural overview of each part of the application.

4.1 System overview.

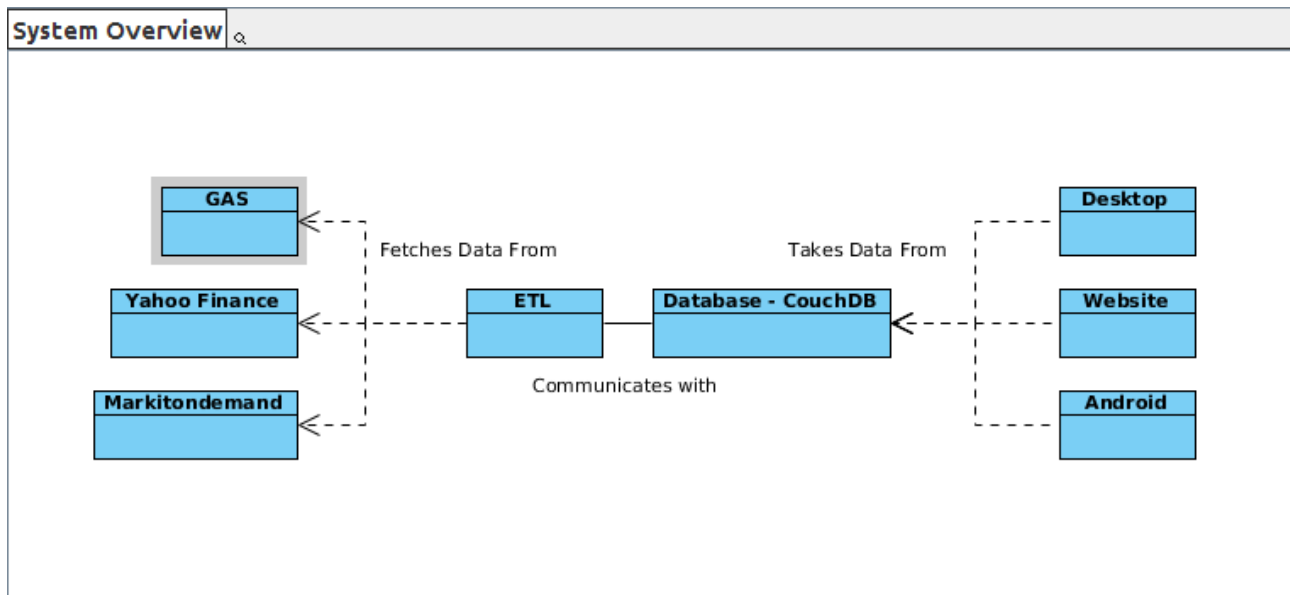
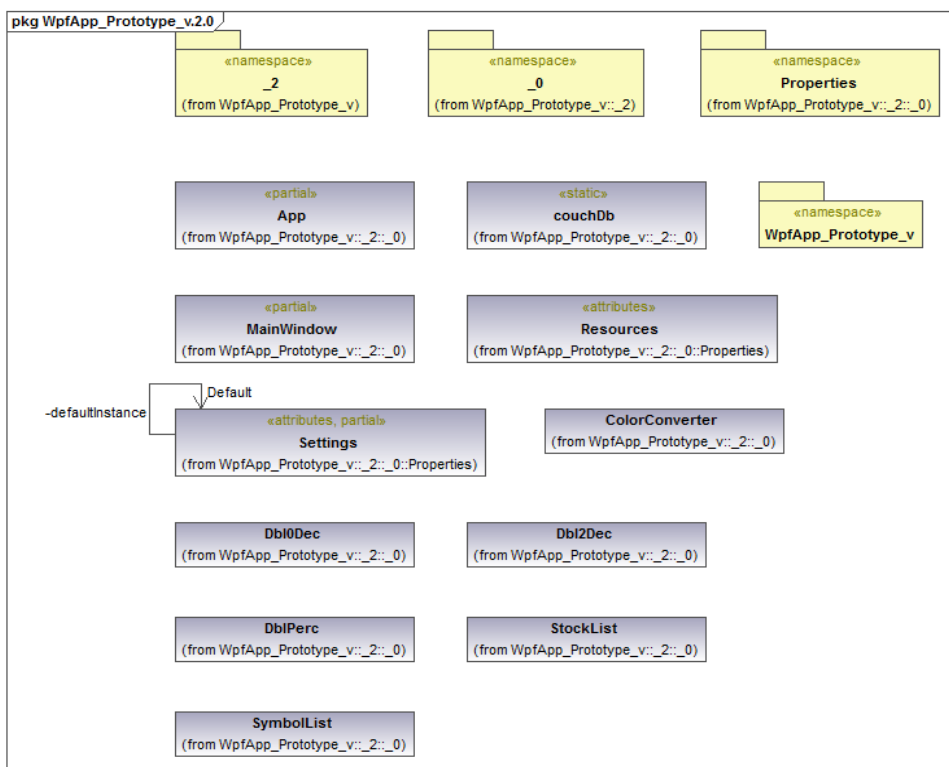


Fig 4.1.1 – System Package Overview.

4.2 Desktop.



Software Design Document

Fig 4.2.1 – Desktop App including packages.

4.3 Android.

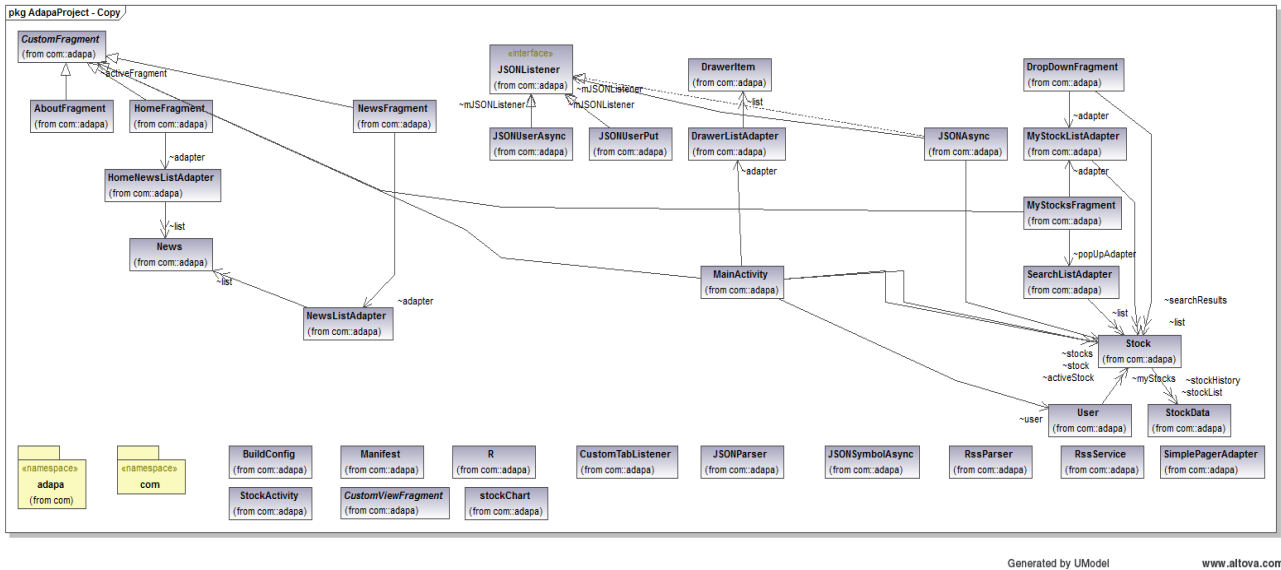


Fig 4.3.1 – Android Class Diagram including packages.

4.4 ETL.

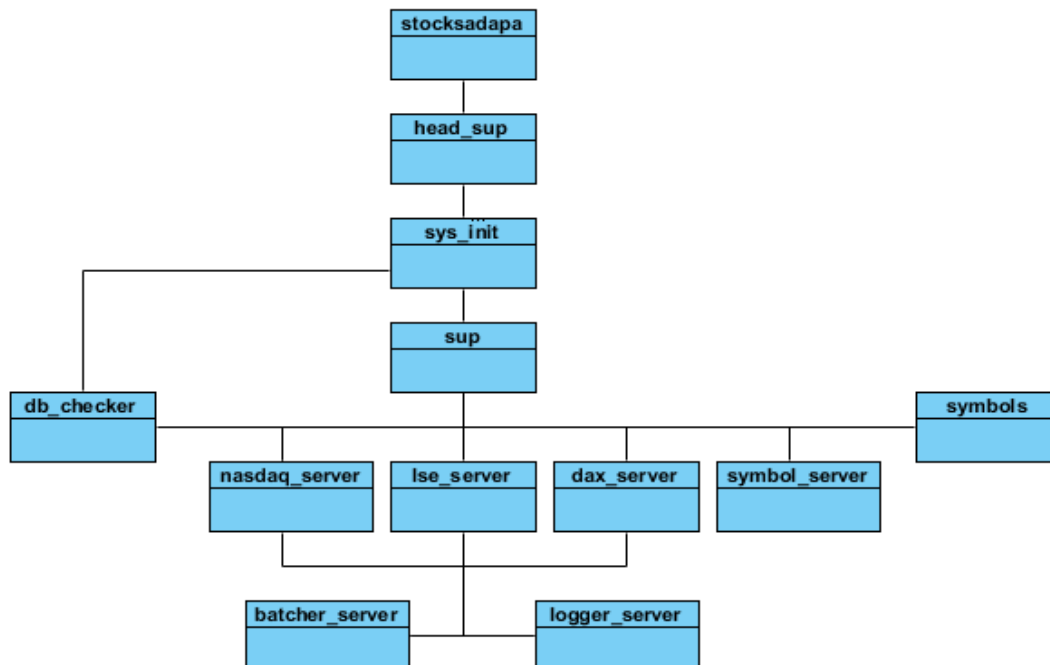


Fig 4.4.1 – ETL Overview.

5 System Design.

The system was designed so that every module would be highly dynamic, flexible and maintainable. We have tried to keep communication centred around CouchDB as it is so simple to communicate with. The ETL is kept separate and only feeds information to CouchDB, no information is ever sent to the ETL explicitly from anything other than HTTP request or requests made to CouchDB.

The interfaces themselves connect and communicate with the database via HTTP requests too, taking full advantage of CouchDB's restful interface and use of JSON. This allowed us to keep the data the same across all platforms.

Here is a list of all components of the system:

- ETL – Back end, fetches data and sends it to the database to be stored.
- Database – CouchDB, stores data and serves it, also hosts small JavaScript functions in the form of “Views”.
- Desktop Application – Written in C# and XAML.
- Android Application – Written in Java/Android.
- Website – Written in Ajax/JQuery.
- Markitondemand – A source for Stock Data.
- Google Action Script – A source for Stock Data.
- Yahoo Finance – A source for Stock Data.
- Symbol Data URLS – Sources where symbols are taken.

Further design details can be found in the SAD document.

5.1 Best Practises and standards.

For the languages we are use, we will be following some best practices / conventions which have been documented for future reference and use. The documents are included with this documentation. They are intended to be guidelines and as such are up to the discretion of the developer to adhere to them. One solid rule we have is that if you are to change someone's code, keep the style consistent which is expressed strongly within those documents. They are titled:

1. C# - Coding Standards.
2. Erlang – Coding Standards.
3. Java_Android – Coding Standards.

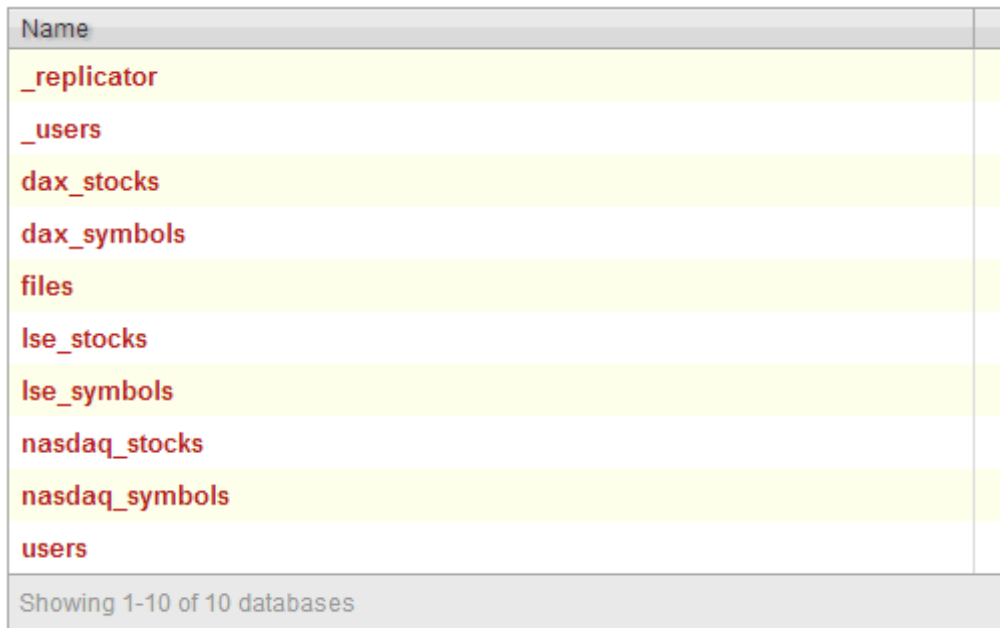
6 Data Design.

6.1 Overview.

This section will show what data structures we are using between the applications. Primarily it will focus upon those that are sent between the different components as opposed to those that are used within each application. It will also provide a brief overview of the database design.

6.2 Database.

The database we are using is CouchDB. CouchDB uses both JSON and JavaScript. We have written half a dozen views in JavaScript to server up data to requesters in specific ways. The stocks are stored using their stock symbol as their ID. CouchDB normally assigns unique Id's for each document but we decided not to use this method.



| Name |
|--------------------------------|
| _replicator |
| _users |
| dax_stocks |
| dax_symbols |
| files |
| lse_stocks |
| lse_symbols |
| nasdaq_stocks |
| nasdaq_symbols |
| users |

Showing 1-10 of 10 databases

Fig 6.1 – Database overview.

The above figure shows the current layout of the database. We have a separate database for each market, a separate database for their stocks, a database for user configurations and a database to hold some common files.

Software Design Document

| Key ▲ |
|--------------------------|
| "ADS.DE" ID: ADS.DE |
| "ALV.DE" ID: ALV.DE |
| "BAS.DE" ID: BAS.DE |
| "BAYN.DE" ID: BAYN.DE |
| "BEI.DE" ID: BEI.DE |
| "BMW.DE" ID: BMW.DE |
| "CBK.DE" ID: CBK.DE |
| "CON.DE" ID: CON.DE |
| "DAI.DE" ID: DAI.DE |
| "DB1.DE" ID: DB1.DE |
| Showing 1-10 of 36 rows |

Fig 6.2 – Database Structure – Stock arrangement.

| Field | Value |
|-------|--|
| _id | "ADS.DE" |
| _rev | "129-4ee16da0519a3ea5fe16f05836687d21" |
| Data | <div><div>Name "ADIDAS N"</div><div><div>StockData</div><div>0<div><div>TimeStamp "Tue Dec 17 2013 09:21:33 GMT+0100"</div><div>LocalTimestamp "Tue, 17 Dec 2013 08:21:31 GMT"</div><div>Price 86.41</div><div>Change -0.13</div><div>ChangePercent -0.15</div><div>High 86.62</div><div>Low 86.44</div><div>Volume 14444</div><div>Close 86.41</div><div>Open 86.62</div></div></div><div><div>1</div><div>2</div><div>3</div><div>4</div><div>5</div><div>6</div><div>7</div><div>8</div><div>9</div></div></div><div><div>Historical</div><div>0<div><div>TimeStamp "Fri Dec 06 2013 19:21:33 GMT+0100"</div><div>LocalTimestamp "Fri, 06 Dec 2013 18:21:31 GMT"</div><div>Price 87.98</div><div>Change 0.21</div><div>ChangePercent 0.24</div><div>High 88.83</div><div>Low 87.82</div><div>Volume 642592</div><div>Close 87.98</div><div>Open 88.43</div></div></div><div><div>1</div><div>2</div><div>3</div><div>4</div><div>5</div><div>6</div><div>7</div><div>8</div><div>9</div><div>10</div></div></div></div> |

Fig 6.3 – Stock Data Example.

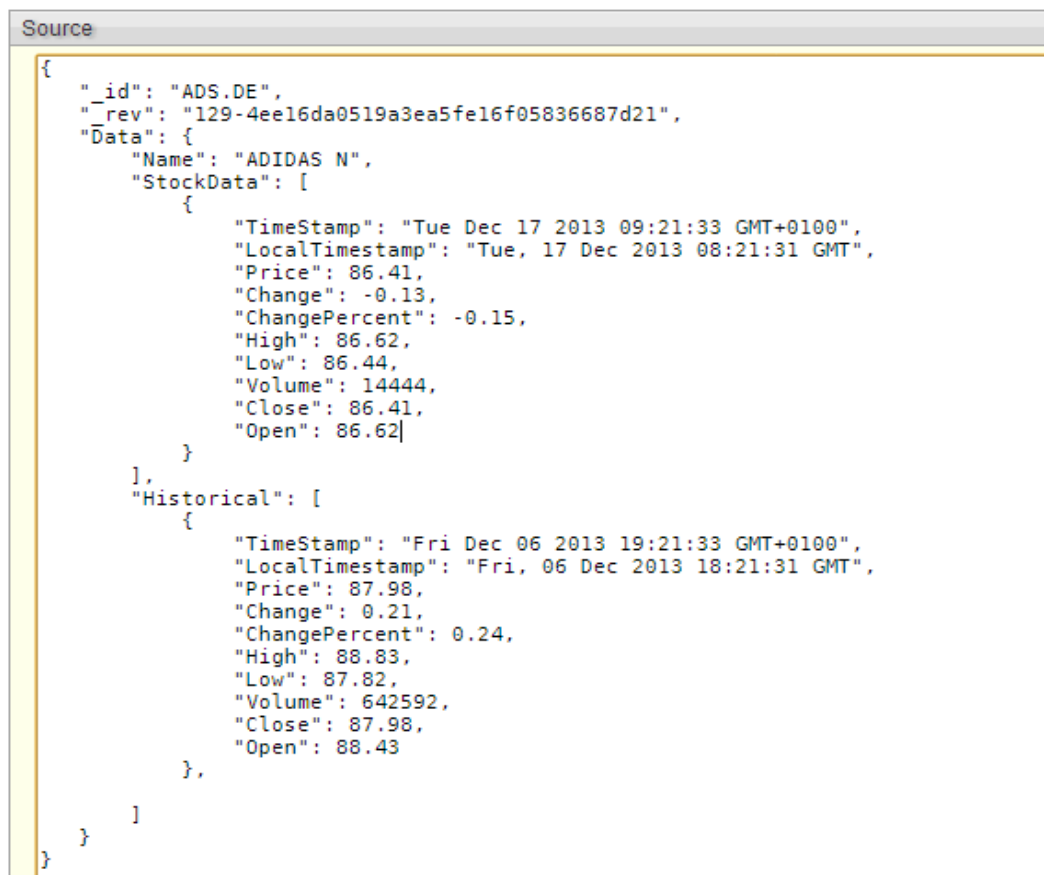
6.2.1 Interaction.

We wanted to use couch mainly for its restful interface. Having never used anything other than SQL we thought this was a pretty interesting idea. We primarily interact with CouchDB via HTTP requests through Erlang and the interfaces. It is very simple to query, write and delete from. Updating was a slight issue that we had to tackle but we solved it with the use of update handlers.

We realised early on that we would not need a dedicated middle layer because Couch acts as the interaction point between the ETL and the clients. This was made all the more simpler due to the view functionality that CouchDB provides. Data is retrieved via views which are written in JavaScript. We aimed to keep them as simple as possible by having single views containing multiple related functions. The views can be queried by HTTP requests to the location of the view. This also was the case for our update handler which takes care of inserting or updating documents into the database.

6.3 JSON Structure.

Couchdb uses JSON for everything, and as such we had to supply it with proper JSON formats. Below is what we designed and considered worthwhile data for our use, data that we didn't use was not saved to the database. Fig 1.0 shows what the structure of JSON is like for an individual stock.



```
{
  "_id": "ADS.DE",
  "_rev": "129-4ee16da0519a3ea5fe16f05836687d21",
  "Data": {
    "Name": "ADIDAS N",
    "StockData": [
      {
        "TimeStamp": "Tue Dec 17 2013 09:21:33 GMT+0100",
        "LocalTimeStamp": "Tue, 17 Dec 2013 08:21:31 GMT",
        "Price": 86.41,
        "Change": -0.13,
        "ChangePercent": -0.15,
        "High": 86.62,
        "Low": 86.44,
        "Volume": 14444,
        "Close": 86.41,
        "Open": 86.62
      }
    ]
  },
  "Historical": [
    {
      "TimeStamp": "Fri Dec 06 2013 19:21:33 GMT+0100",
      "LocalTimeStamp": "Fri, 06 Dec 2013 18:21:31 GMT",
      "Price": 87.98,
      "Change": 0.21,
      "ChangePercent": 0.24,
      "High": 88.83,
      "Low": 87.82,
      "Volume": 642592,
      "Close": 87.98,
      "Open": 88.43
    }
  ]
}
```

Fig 6.4 - JSON Structure.

7 User interface Design.

The user interface document was created early on and has been used extensively for the creation of all 3 user interfaces. Please read the “UI Research Report” for more information about the interfaces.

8 Design Justification and Comments.

8.1 ETL – Overview.

The ETL was designed primarily on a trial and error prototype process. As the group was inexperienced with Erlang, we didn't really have any ideas how to start writing the ETL. We did start with some initial prototypes and tests which started to form the basis of the ETL itself. Unfortunately we did not employ any design patterns during the creation of the software. The Software Architecture course which is running along side the project didn't have any patterns that were particularly appropriate for use in a functional language.

8.2 ETL – Version 1.

The first iteration used a non OTP style. An architectural overview can be found in the corresponding architecture documentation. Our initial ideas were to have the collectors running from a schedule which was an OTP module. This resulted in us having a lot of different modules doing nearly the same things. The ETL soon became rather bloated and we decided to scrap what we'd done so far to start again.

There's not much more to say about this version other than it worked well as a prototype. We learnt a lot from creating this and I think that had we stuck with this version our system would be weaker.

8.3 ETL – Version 2.

With this iteration of the ETL we were set on a few key principles that we wanted to achieve:

1. Use more OTP style modules.
2. Have the fetchers run on the same schedule, but have them controlled by a supervisor.
3. Make it as dynamic as possible while having high maintainability.
4. Make use of definitions

With these in mind we created the second version. An architectural view of version 2 is available in the architecture documentation or can be seen above in fig 2.4. I feel that this version was designed and implemented in a much cleaner and efficient way. Since we completed the ETL we've had to make some changes and they have been very simple to make because we followed these steps.

The main idea behind this version was to utilise supervisor control and OTP. We have 3 different servers for the different markets we are using. We didn't make them into a single module as each has a different way of querying, extracting and dealing with their respective data. If we did have it in a single module it would have been much harder to maintain. It would have also been double the size.

The fetchers all run a time related to their stock markets, we don't bother querying when they are closed as the data is just going to be the same anyway. This means that if we were to move the

Software Design Document

servers to a different time-zone, you would have to change a few values in the definitions to accommodate the new time. They take the lists of symbols stored on the database and they then create a process for each symbol. These processes are slept individually for a random time before they execute the query to their source. This was done to avoid the sources from blocking us off. We decided that an upper limit of around 10 minutes was acceptable as speed was not a big issue for us. This can be changed and tweaked to find a “sweet spot” but for consistency, we decided to stick with this.

The symbols for the LSE and Nasdaq are obtained from 2 websites. They are collected by a symbols server and are updated daily. They are gathered, parsed and then sent to the database sequentially. This also happens during the initialisation of the database, but more about this is covered below. This is the module that we would clean up or re-write since we've learnt about regular expressions. It was one of the first to be made and as such is a little less elegant than the rest, but it works so we decided to leave it be.

We also introduced a “Batcher” and a “Logger”. The logger provided some error logging and success logging to get some more information about failed request to the sources. The batcher was initially going to batch the information into groups of 200 and then send them to the database. Couch doesn't have update handlers for the batch API so this was scrapped, but we kept the name. The batcher instead just takes messages from the fetcher servers and depending on who sent it adds them to their respective databases.

Lastly, we created some initialisation modules to get the database up and running. It was getting a bit tedious to create everything by hand all the time so this seemed like a good idea. The module simply sends some requests or queries to the databases that we expect to be there and if they reply with fails we create them.

This module `sys_init` also uploads the symbols if they're missing. We tried to make it as flexible and maintainable as possible. It uses a lot of information from the definitions file which usually means that changes need only happen in one place.

While not necessarily part of the ETL, we created an Erlang make file and a regular makefile to handle the building of the system for Unix systems. The makefile sets up the application into either development or release mode. It also makes Erlang calls to do the compilation. We could have extended this to start the Erlang module a daemon but have decided not to so that we could test it more easily. One final thing we added to this was to create Erlang docs, you should find in the docs folder a HTML page which acts like other generated documentation (such as Java-docs). To find the Edocs, navigate to the folder where you have the application. It should be under the following folder:

ETL Development > Version 2 > Doc

8.4 Source API Interaction.

Below is a brief description of how we interact with our sources.

8.4.1 Google Action Script.

We decided to use GAS as the old google finance API is now deprecated. You can still query it but the information it returns is not what we wanted. The way we found that we could do this was to use GAS. We simply host a small script via google drive and query it to get the stock information for a symbol and its given market. We also use it to fetch historical data for

specific stocks. One drawback is the limited amount of traffic and queries that we can make per day. It was sometimes an issue during development. To cope with this we created 2 scripts on different accounts in case one of them becomes blocked. The inclusion of this meant that we had to also use the Crypto and SSL libraries in Erlang, while not a difficulty it was something that we had to keep in mind.

8.4.2 Yahoo Finance.

We decided to use yahoo finance as another source because it allows custom querying. We could specify certain symbols and could as a result get the exact information that we required. One problem with it was that the data was only sent back in CSV, this meant that a small parser had to be written to use it in the main program.

8.4.3 Markitondemand.

Our third and final source was found first and became the standard to which we compared the others. It provided all the information we needed and sent it back in JSON so there were no problems here. Like the other sources, we simply send a HTTP request to the URL provided by their API along with a stock symbol and it returns its data.

9 SCM Information.

We decided to use Bitbucket to host our data. Initially we had some issues getting used to using Git but in the end it was relatively straight forwards to use. The windows and mac users in the group used Source tree while the Linux user used git on the command line. Each part of the project had their own branch to work on and after development they were all merged into the master branch. The interface branches shall be left open in case of further iterations.

10 References.

Copy of the IEEE Standard - <http://www.iso-architecture.org/ieee-p1016/IEEE-P1016-d5.0b.pdf>

Example Templates - http://www.ceng.metu.edu.tr/media/course/ceng490/sdd_template.pdf?rev=https://docs.google.com/document/d/1pgMutdDasJb6eN6yK6M95JM8gQ16IKacxxhPXgeL9WY/preview

Repo URL - <https://bitbucket.org/adapa/adapaterm3projectgit>

11 Glossary.

ASW – Adapa Stock Watcher.

OTP – Open Telecom Platform, part of Erlang.

GAS – Google Action Script.

SCM – Source Control Manager.