Introduction

Code Repository

The source code for the *Lonsec* project is available at https://github.com/johnfarnell/lonsec. Checkout the *Master* branch to a local folder <ROOT_FOLDER>

Prerequisites

To build and run the software the following is required

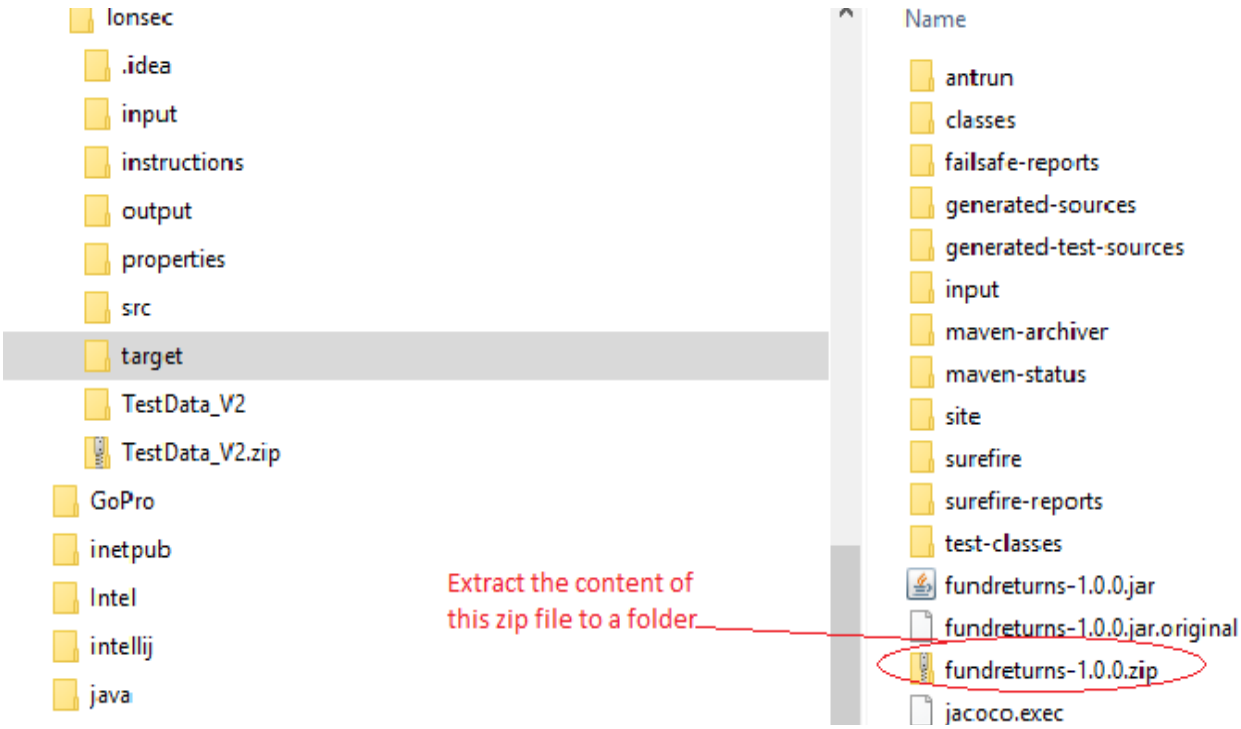
1. Maven 3
2. Java 1.8

Build the software

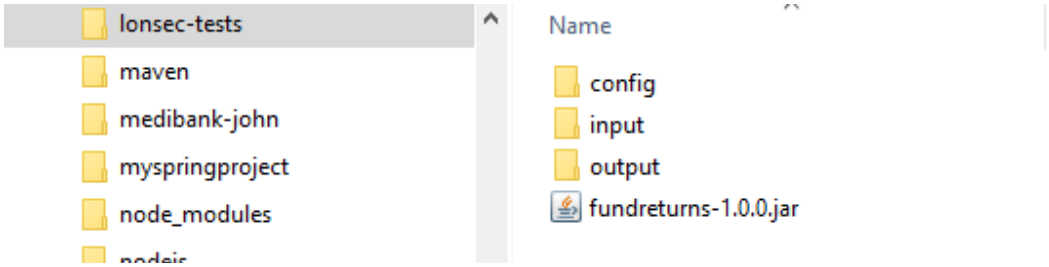
From the command line at the <ROOT_FOLDER>, enter *mvn clean install*

Install and run the software

After successfully building the software, find the *fundreturns-1.0.0.zip*

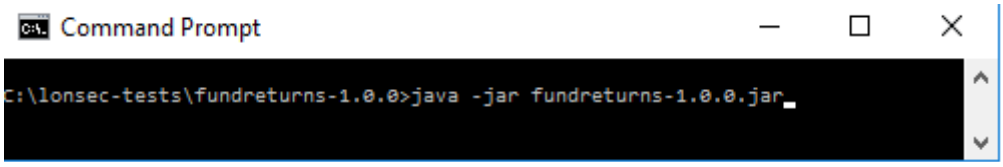


and extract the contents to a folder (referred to later as the <DEPLOYMENT_FOLDER> – below is an image of what the zip file contains



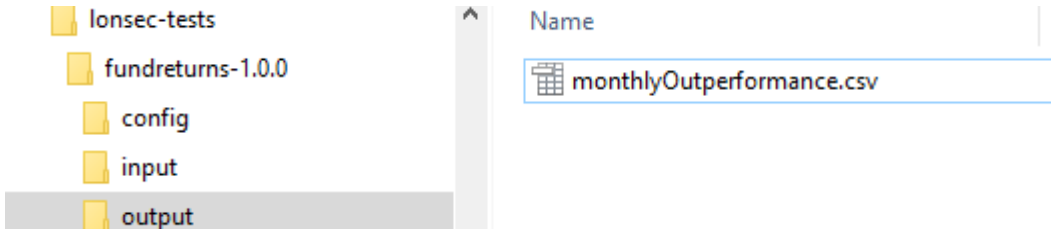
Navigate to the folder and, from the command line run

java -jar fundreturns-1.0.0.jar



The console log will show the following, and the application will complete :-

Upon successful completion, look into the *<DEPLOYMENT_FOLDER>/output* folder, the application will have generated an output csv file ***monthlyOutperformance.csv***



	A	B	C	D	E	F
1	FundName	Date	Excess	OutPerformance	Return	Rank
2	Pengana Global Small Companies Fund	30/11/2016	4	out performed	3.87	1
3	Vanguard Australian Shares Index ETF	30/11/2016	2.87	out performed	2.74	2
4	Vanguard All-World Ex-US Shares Index ETF	30/11/2016	0.87		0.75	3
5	Vanguard Wholesale Australian Fixed Interest Index	30/11/2016	-0.01		-1.47	4
6	Vanguard Wholesale Emerging Markets Shares Index	30/11/2016	-1.74	under performed	-1.86	5
7	Goldman Sachs Emerging Leaders Fund	30/11/2016	-3.88	under performed	-4.01	6
8	Vanguard Wholesale Emerging Markets Shares Index	31/10/2016	5.37	out performed	0.74	1
9	Vanguard All-World Ex-US Shares Index ETF	31/10/2016	3.57	out performed	-1.07	2
10	Pengana Global Small Companies Fund	31/10/2016	3.45	out performed	-1.18	3
11	Vanguard Wholesale Australian Fixed Interest Index	31/10/2016	-0.02		-1.32	4
12	Vanguard Australian Shares Index ETF	31/10/2016	2.39	out performed	-2.24	5
13	Goldman Sachs Emerging Leaders Fund	31/10/2016	-4.22	under performed	-8.85	6

So what just happened?

The application used the csv files in the *Input* folder, to derive the *monthlyOutperformance.csv.* This output csv shows the data in the format requested in the interview requirement specification …

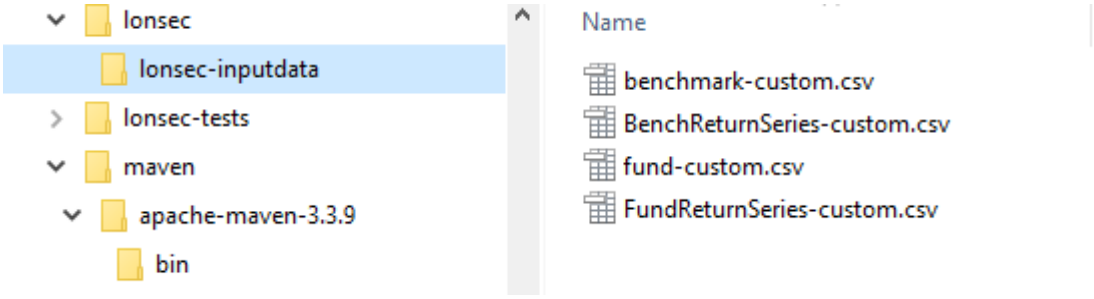
"*The application will produce one output file called monthlyOutperformance.csv*

The output is to be sorted by [Date] descending, then sub-sorted by [Rank] ascending..... Your solution should contain data for
multiple months, as described further down in this document.

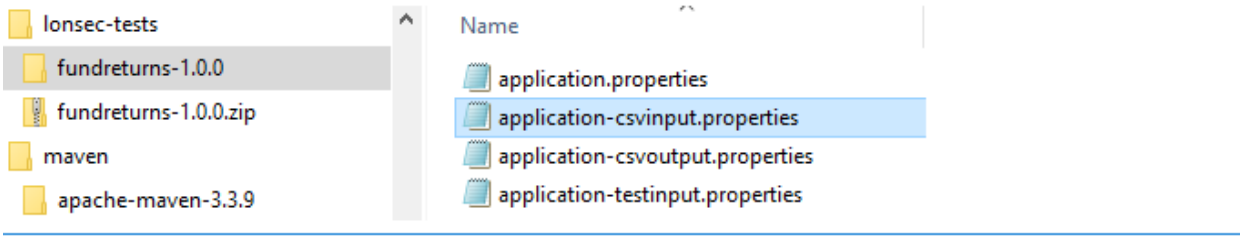
The above is the default behaviour, next I will explain the various ways in which the application can be configured.

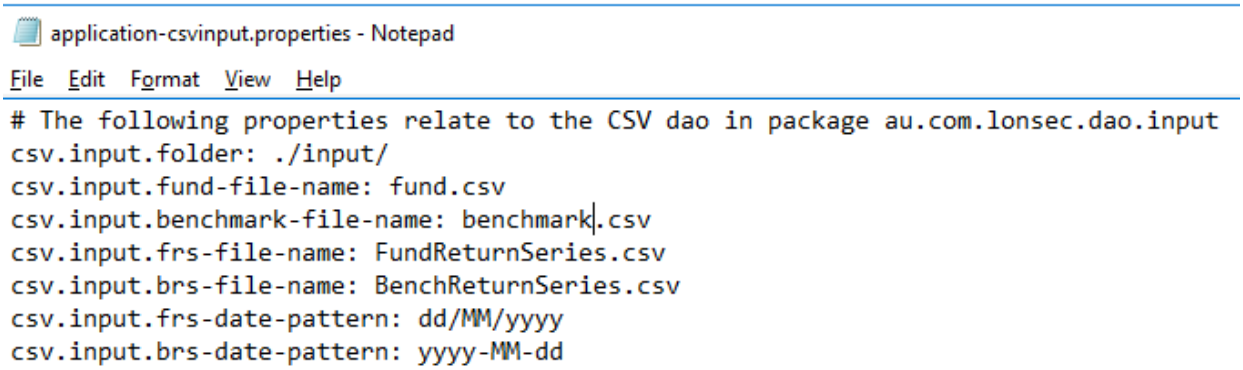
Configure the location and the name of the input files

Assume you now want to run input files from a different location. In my example this will be *C:\lonsec\lonsec-inputdata*. Note the file names are different (they have the literal *-custom* at the end).



How is this achieved?
Go back to *<DEPLOYMENT_FOLDER>/config* and open the file *application-csvinput.properties*.





Change *csv.input.folder* to ***C://lonsec/lonsec-inputdata/*** *(*Note the trailing "\")
Change each of the *file-name* fields to have the literal *-custom*
e.g. ***benchmark.csv → benchmark-custom.csv***

```
# The following properties relate to the CSV dao in package au.com.lonsec.dao.input
csv.input.folder: C://lonsec/lonsec-inputdata/
csv.input.fund-file-name: fund-custom.csv
csv.input.benchmark-file-name: benchmark-custom.csv
csv.input.frs-file-name: FundReturnSeries-custom.csv
csv.input.brs-file-name: BenchReturnSeries-custom.csv
csv.input.frs-date-pattern: dd/MM/yyyy
csv.input.brs-date-pattern: yyyy-MM-dd
```

Save the file and, again from the *<DEPLOYMENT_FOLDER>* in the command line, run *java -jar fundreturns-1.0.0.jar*

In a similar fashion, the location and the name of the output csv file can be changed via the *<DEPLOYMENT_FOLDER>/config/application-csvoutput.properties* file

```
# The following properties relate to the CSV dao in package au.com.lonsec.dao.input
csv.output.folder: ./output/
csv.monthly.performance.file-name:monthPerformance.csv
csv.monthly.performance.fundname.col:FundName
csv.monthly.performance.date.col:Date
csv.monthly.performance.return.col:Return
csv.monthly.performance.rank.col:Rank
csv.monthly.performance.columns:FundName,Date,Excess,OutPerformance,Return,Rank
csv.monthly.performance.date.format:dd/MM/yyyy
csv.monthly.performance.decimal.format:#,##0.00
```

As you can see, there are various other configuration settings which control the layout of the input and output files.
The *column* names and the order they appear in the output csv can be configured. **Note: Please do not change the Excess and OutPerformance** settings without also changing their equivalent values in the *application.properties* file

Configure the sort order of the output csv file

As discussed, the default sort order of the *MonthlyOutperformance.csv* file is Date Descending/Rank Ascending.

The software has been developed to allow the sort order to be easily changed. Package *au.com.lonsec.sort* contains all of the sorting available. The class *au.com.lonsec.sort.SortByDateDescAndRankAsc* provides the default, but, by using Spring Boot's @ConditionalOnProperty annotation, there are other alternative provided.

For example going to *<DEPLOYMENT_FOLDER>/config* and open the file *application.properties* and change

sort.return.percent:default to
sort.return.percent:DateAscReturnDesc

and re-run :

java -jar fundreturns-1.0.0.jar and see that the sort order has changed

	A	B	C	D	E	F
1	FundName	Date	Excess	OutPerformance	Return	Rank
2	Vanguard Wholesale Australian Fixed Interest Index	30/06/2016	-0.02		1.29	1
3	Vanguard Wholesale Emerging Markets Shares Index	30/06/2016	3.03	out performed	1.22	2
4	Vanguard Australian Shares Index ETF	30/06/2016	-0.73		-2.54	3
5	Vanguard All-World Ex-US Shares Index ETF	30/06/2016	-1.77	under performed	-3.58	4
6	Goldman Sachs Emerging Leaders Fund	30/06/2016	-2.42	under performed	-4.23	5
7	Pengana Global Small Companies Fund	30/06/2016	-2.97	under performed	-4.78	6
8	Goldman Sachs Emerging Leaders Fund	31/07/2016	0.3		8.07	1
9	Vanguard Australian Shares Index ETF	31/07/2016	-1.78	under performed	5.98	2
10	Vanguard Wholesale Emerging Markets Shares Index	31/07/2016	-5.01	under performed	2.75	3
11	Pengana Global Small Companies Fund	31/07/2016	-5.05	under performed	2.72	4

If there was a future requirement to sort the output in a different way that is currently covered, then a different implementation of the sort could easily be created without changing any existing code or touching the code framework

Likewise, the package a*u.com.lonsec.rank* has a similar design to allow the mechanism by which a fund is ranked. The current default implementation is best return for a particular date – see a*u.com.lonsec.rank.FundServiceRankByBestReturnForSameDate*

Outperformance Text

As per the requirement, there are properties in *application.properties* .which allow the Outperformance text to be configured – the default values are held in *application.properties*

```
# The following properties are assigned to au.com.lonsec.description.DisplayReturnServiceOutperformanceLiteral.class
display.return.service.output.performance.label:OutPerformance
display.return.service.output.over.performance.literal:out performed
display.return.service.output.under.performance.literal:under performed
display.return.service.output.matched.performance.literal:
display.return.service.output.performance-underlimit:-1.00
display.return.service.output.performance.overlimit:1.00
```

Change these values, and rerun the application and you will see the Outperformance literal change in the *monthlyOutperformance.csv*

It is also possible to change the *underperformance* and *overperformance* limits from the default of -1.00 and 1.00 respectively

Configuring Calculations

As per the requirement, the application has been designed to allow new calculations to be dependency injected into the logic without changing any existing code. The current requirement is to have a calculation of *Excess* by "*subtracting the benchmark return from the fund return for a particular date and rounding this to 2 digits and rounding half-down*"

To meet this requirement, and allow for future calculations to be seamlessly injected into the application, an interface has been created called

au.com.lonsec.calculation.FundReturnSeriesCalculation

Implementations of this interface will automatically be injected into the application by the Spring Boot framework, and called to perform their calculation. This can then be configured (as per *Excess*) to then appear in the output csv. Please view *au.com.lonsec.calculation.CalculateReturnPercentageDifference* and note the settings in *application.properties*

calc.return.percent.diff.enabled:true
calc.return.percent.diff.dec.places:2 – the number of decimal places
calc.return.percent.diff.label:Excess – this is the label that will appear on the output csv

Bad Data

Any bad data will result in a FundsException being thrown with a message, and the application will terminate

Testing

Mockito and junit have been used to test the individual classes. By running
 mvn clean jacoco:prepare-agent install jacoco:report from the <ROOT_FOLDER> and opening <ROOT_FOLDER>/\target\site\jacoco*index.html* you can see a breakdown of the test coverage.