FX Data Automation

Problem statement:

A Calibre client has asked for FX data to be sourced from:

https://eodhistoricaldata.com/knowledgebase/list-supported-currencies/

and delivered as an email file attachment at 8am, 12pm and 4pm each day.

The FX rates of interest are AUDUSD, AUDNZD, AUDHKD, AUDKRW and AUDJPY

The email file attachment will have:

- a filename of obsval_YYYYMMDD_HHMM.csv e.g. obsval_20191015_0800.csv
- a header row of FOREX, VALUE
- a row for each FX pair e.g. AUDUSD, .65

Design:

There are three distinct functionalities that need to be built in the application:

- Fetching FX Data from the given API
- Scheduling the extract of FX Data
- Delivery of FX Extract via Email

While designing the application we have to be cognizant of the parts of the application that may undergo changes in future. As per open-close design principle, our application should be open for extension and closed for modification.

For Example, a client may want the FX data to be sourced from a different API in future or it may be required to deliver the FX Data extract using a queue or DB.

Similarly, there could be few minor changes in the way the data is fetched from API or delivered via mail and theses should be configured using config:

- Email Id
- File Header
- API URL
- File Name
- Scheduled Time

Development:

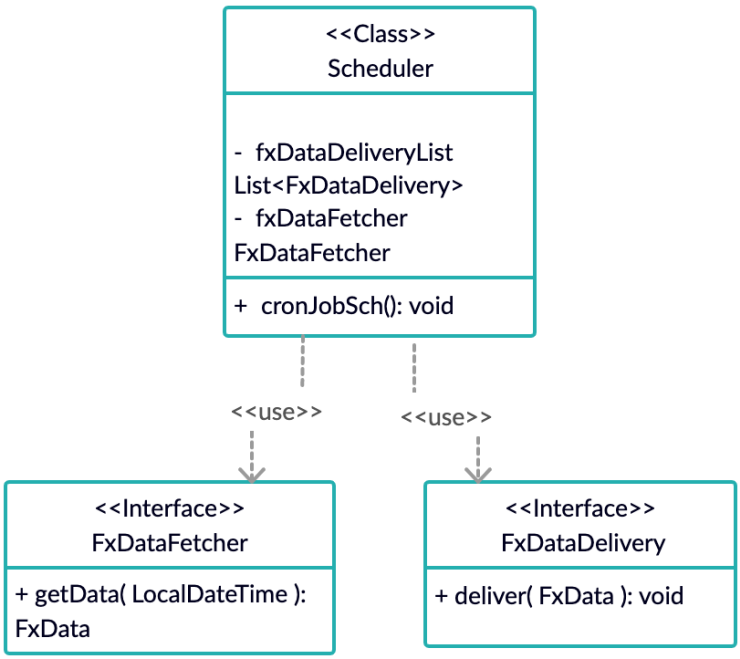
Language: Java 8
Web Service:
Build: Gradle
Logging : Logback
Web Framework: Spring Boot

The project consists of two modules:

- fx_automation – Main module responsible for fetching fx data and sending fx extract via mail. It consists of following packages:
    - input - Used to fetch fx data from external world
    - output – Used to deliver fx data to external world
    - scheduler – Responsible for triggering input and output at specified intervals.
    - mail – Sends mail to configured mail ids
    - config - Used to fetch config from the config server
    - model - Consists of model classes used in application

- fx_config_server - To manage the properties, spring cloud config has been used.

Class Diagram:



As the scheduler consists of interfaces FxDataFetcher, a new implementation for fetching FX Data could easily be plugged in.

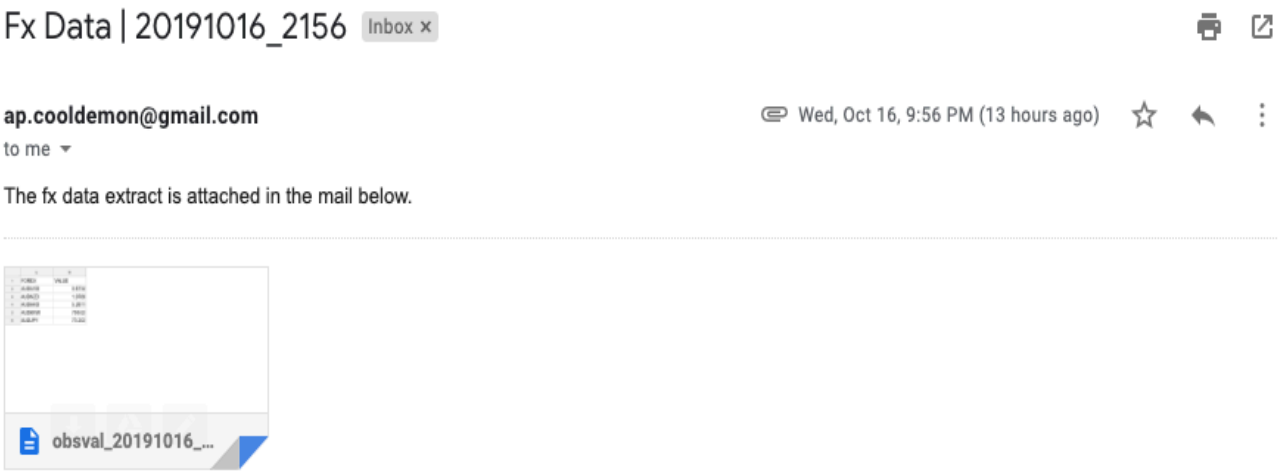
Similarly as it consists of a list of FxDataDelivery , additional implementations to send FX extract to other destinations can be added with ease.

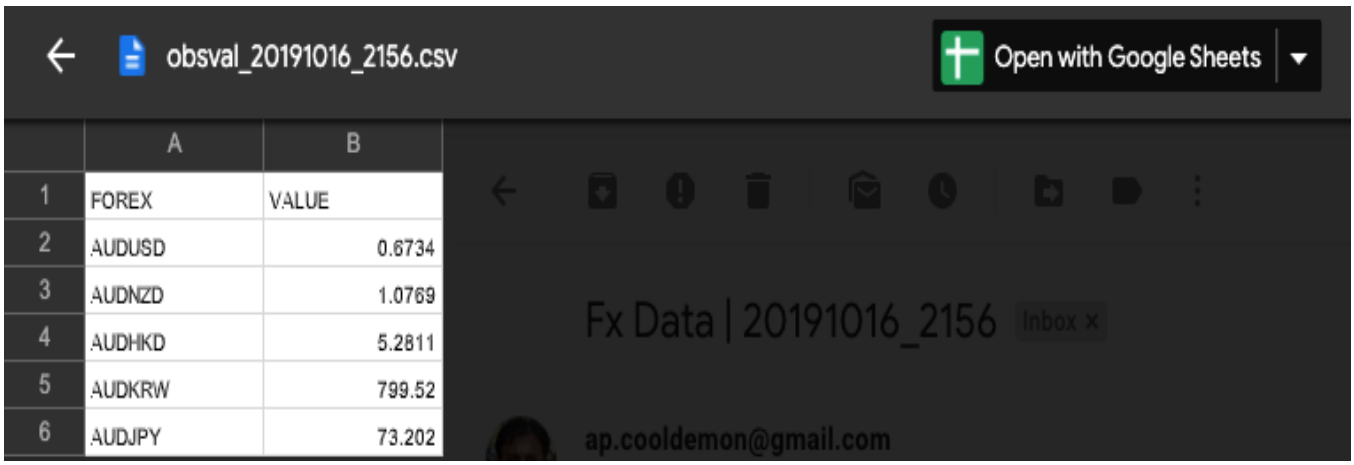
The ExceptionMailSender class is responsible for mapping any exception caught while processing.

Output:

Following are the snapshots of mail received as part of test run:

FX Extract Mail:





Exception Mail:

Exception | FX Data Job Inbox ×

ap.cooldemon@gmail.com 8:43 AM (2 hours ago)
to me

An exception has been encountered while running FX Data job :
org.springframework.web.client.RestClientException: Could not extract response: no suitable HttpMessageConverter found for response type [class com.calibre.rms.fx.model.FxRecord] and content type [text/html;charset=UTF-8]
at org.springframework.web.client.HttpMessageConverterExtractor.extractData(HttpMessageConverterExtractor.java:121)
at org.springframework.web.client.RestTemplate.doExecute(RestTemplate.java:737)
at org.springframework.web.client.RestTemplate.execute(RestTemplate.java:670)
at org.springframework.web.client.RestTemplate.getForObject(RestTemplate.java:311)
at com.calibre.rms.fx.input.FxRecordFetcher.getData(FxRecordFetcher.java:22)
at com.calibre.rms.fx.input.APIFxDataFetcher.getData(APIFxDataFetcher.java:29)
at com.calibre.rms.fx.scheduler.Scheduler.cronJobSch(Scheduler.java:39)
at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
at java.lang.reflect.Method.invoke(Method.java:498)
at org.springframework.scheduling.support.ScheduledMethodRunnable.run(ScheduledMethodRunnable.java:84)
at org.springframework.scheduling.support.DelegatingErrorHandlingRunnable.run(DelegatingErrorHandlingRunnable.java:54)
at org.springframework.scheduling.concurrent.ReschedulingRunnable.run(ReschedulingRunnable.java:93)
at java.util.concurrent.Executors$RunnableAdapter.call(Executors.java:511)
at java.util.concurrent.FutureTask.run(FutureTask.java:266)
at java.util.concurrent.ScheduledThreadPoolExecutor$ScheduledFutureTask.access$201(ScheduledThreadPoolExecutor.java:180)
at java.util.concurrent.ScheduledThreadPoolExecutor$ScheduledFutureTask.run(ScheduledThreadPoolExecutor.java:293)
at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1149)
at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:624)
at java.lang.Thread.run(Thread.java:748)

NB:

Some of the properties would need to be change to run in a new environment.

fx_config
fx-automation-service.templocation – point to local temp folder

spring-config-server
sptring.cloud.config.server.git.uri – point to local git repo

fx_automation
spring.mail.username: <your user name>
spring.mail.password: <password>