Architecture & Design Document

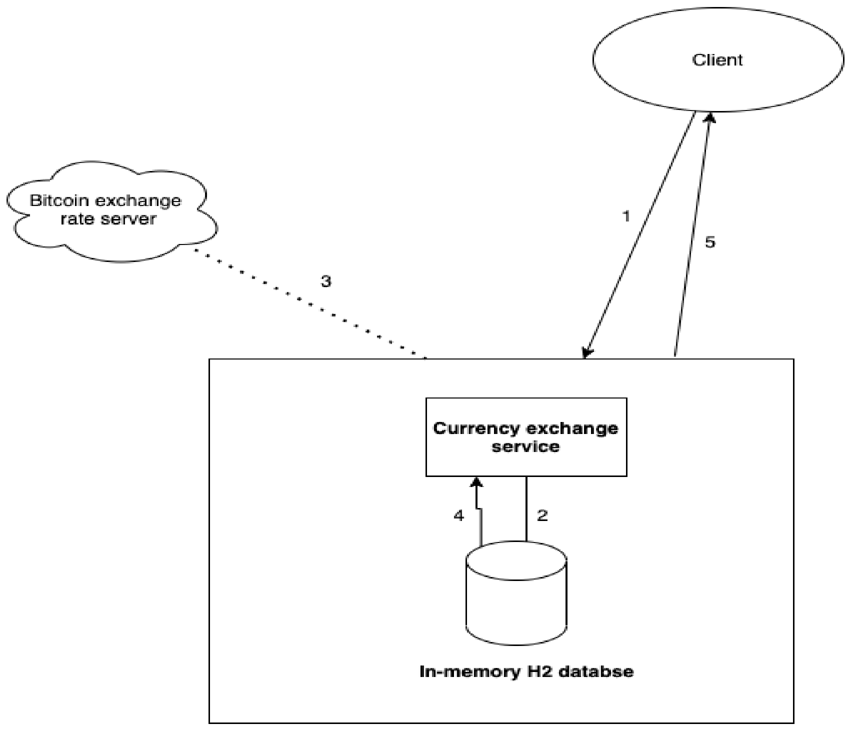
Technologies Used

Frameworks:
- SpringBoot2
- Spring
- Hystrix
- Junit
- Mockito

Infrastructure related:
- Docker
- Maven

Programming Languages
- Java 11
- Bash

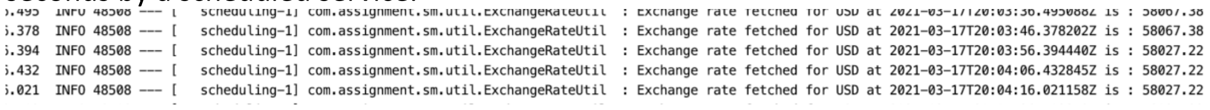
High Level Client-Server Architecture



<u>Below is an overview of how the live exchange rate endpoint works:</u>

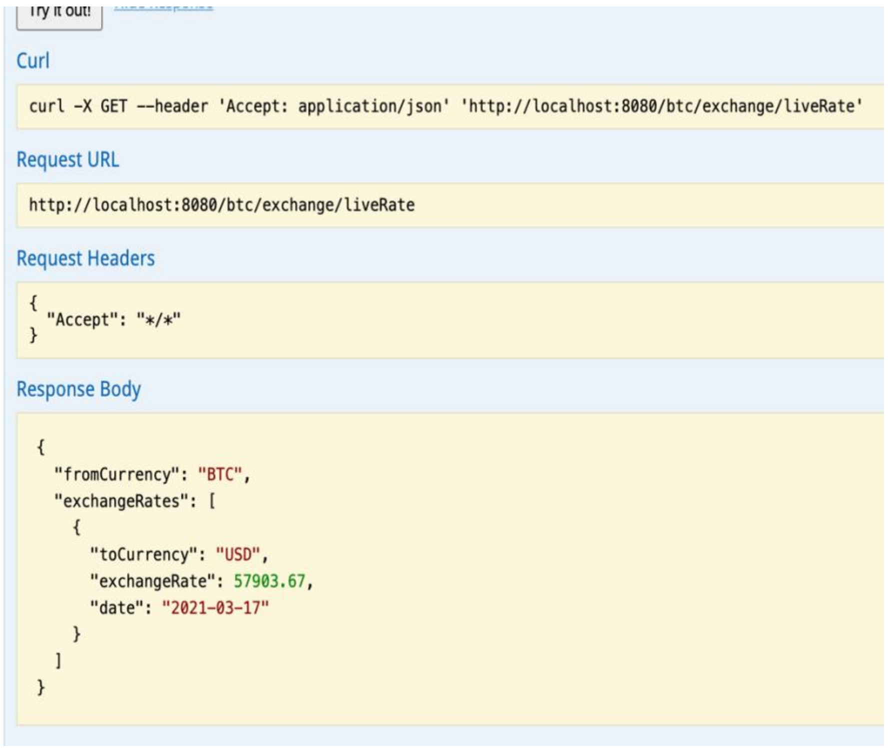
- A scheduled service will repeatedly fetch the real time exchange rates for BTC to USD from an **exchange server** based on a configurable delay parameter in app-properties.
- This check and update happens asynchronously, and the updated rate is saved in a simple java object for fast retrieval by the real-time exchange Rate REST endpoint.

  Below logs demonstrate fetching the exchange rate from exchange server every 10 seconds by a scheduled service.



```
5.495  INFO 48508 --- [   scheduling-1] com.assignment.sm.util.ExchangeRateUtil  : Exchange rate fetched for USD at 2021-03-17T20:03:36.495088Z is : 58067.38
5.378  INFO 48508 --- [   scheduling-1] com.assignment.sm.util.ExchangeRateUtil  : Exchange rate fetched for USD at 2021-03-17T20:03:46.378202Z is : 58067.38
5.394  INFO 48508 --- [   scheduling-1] com.assignment.sm.util.ExchangeRateUtil  : Exchange rate fetched for USD at 2021-03-17T20:03:56.394440Z is : 58027.22
5.432  INFO 48508 --- [   scheduling-1] com.assignment.sm.util.ExchangeRateUtil  : Exchange rate fetched for USD at 2021-03-17T20:04:06.432845Z is : 58027.22
5.021  INFO 48508 --- [   scheduling-1] com.assignment.sm.util.ExchangeRateUtil  : Exchange rate fetched for USD at 2021-03-17T20:04:16.021158Z is : 58027.22
```

  Example of API response:



Curl

```
curl -X GET --header 'Accept: application/json' 'http://localhost:8080/btc/exchange/liveRate'
```

Request URL

```
http://localhost:8080/btc/exchange/liveRate
```

Request Headers

```
{
  "Accept": "*/*"
}
```

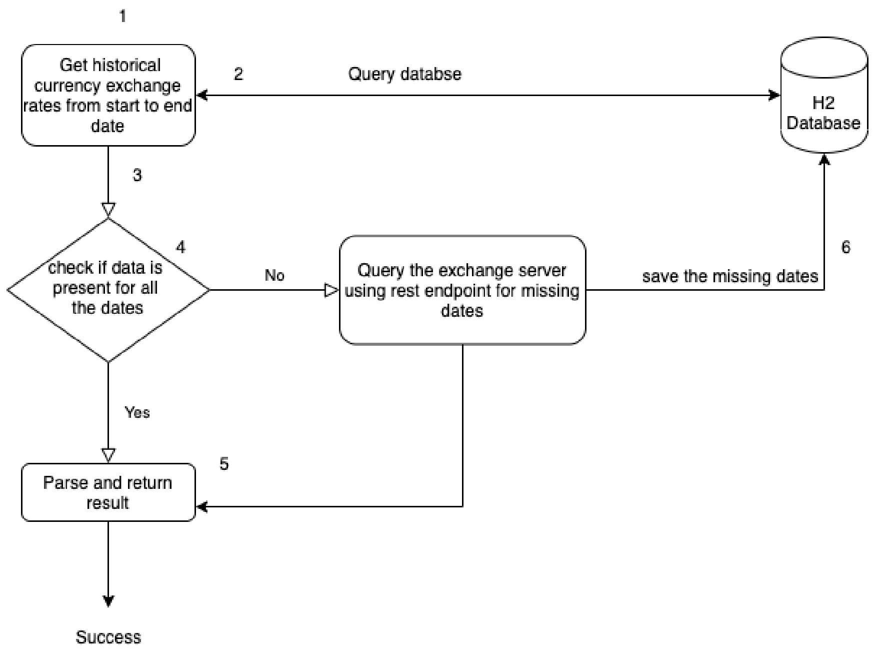
Response Body

```
{
  "fromCurrency": "BTC",
  "exchangeRates": [
    {
      "toCurrency": "USD",
      "exchangeRate": 57903.67,
      "date": "2021-03-17"
    }
  ]
}
```

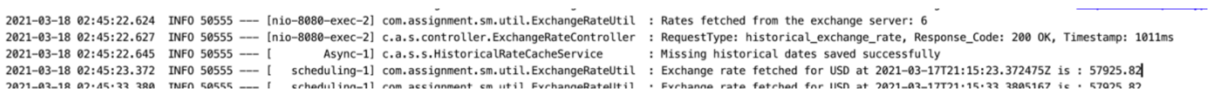
The **fromCurrency** and **toCurrency** values are fetched from the application properties file. Hence in future if there is support for more than one currency then instead of the simple java object used to store the Realtime rates we can use a map like data structure or key-value cache like Redis for sufficing the real time exchange rate endpoint

Below is an overview of how the historical rates endpoint works:

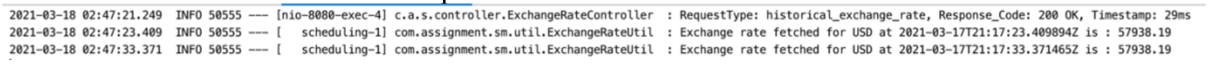
1. Client sends a get request for historical dates using the service REST endpoint.
2. After initial validation, the API checks if the data is available in the in-memory database for all the requested dates.
3. If exchange rates are missing for any dates then the same are fetched from exchange server using its REST endpoint.
4. These missing dates are saved in the in-memory database using an asynchronous request. **This storage will reduce network latency in future for data requested for same set of dates**.
5. The requested historical rate data is sent back to the client.


1
Get historical currency exchange rates from start to end date
2
Query databse
H2 Database
3
4
check if data is present for all the dates
No
Query the exchange server using rest endpoint for missing dates
save the missing dates
6
Yes
Parse and return result
5
Success

Below logs show the response time in milliseconds when the rate data is missing for historical dates

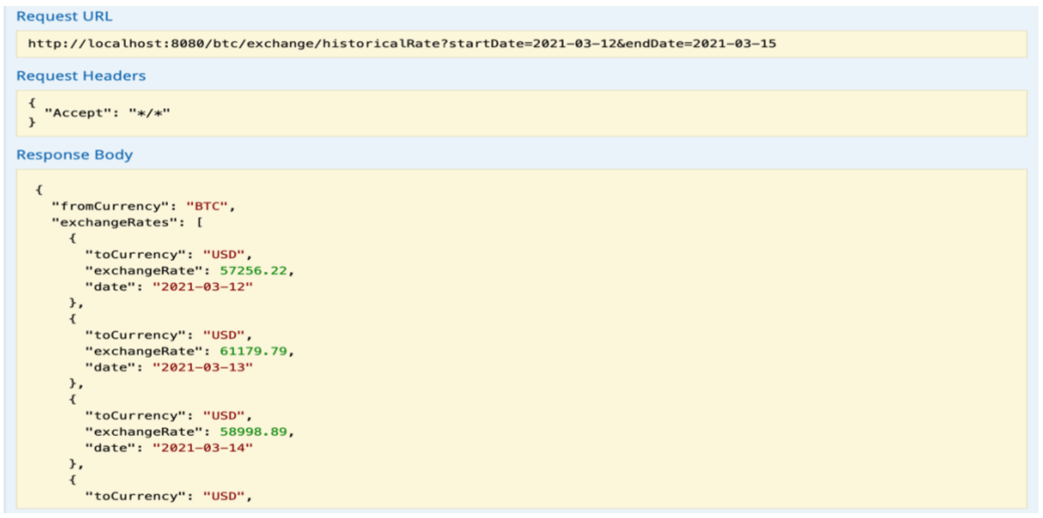

```
2021-03-18 02:45:22.624  INFO 50555 --- [nio-8080-exec-2] com.assignment.sm.util.ExchangeRateUtil  : Rates fetched from the exchange server: 6
2021-03-18 02:45:22.627  INFO 50555 --- [nio-8080-exec-2] c.a.s.controller.ExchangeRateController  : RequestType: historical_exchange_rate, Response_Code: 200 OK, Timestamp: 1011ms
2021-03-18 02:45:22.645  INFO 50555 --- [         Async-1] c.a.s.s.HistoricalRateCacheService       : Missing historical dates saved successfully
2021-03-18 02:45:23.372  INFO 50555 --- [    scheduling-1] com.assignment.sm.util.ExchangeRateUtil  : Exchange rate fetched for USD at 2021-03-17T21:15:23.372475Z is : 57925.82
2021-03-18 02:45:33.380  INFO 50555 --- [    scheduling-1] com.assignment.sm.util.ExchangeRateUtil  : Exchange rate fetched for USD at 2021-03-17T21:15:33.380516Z is : 57925.82
```

Below logs show the reduced response time when the same rate data is fetched for the historical dates for consecutive requests.


```
2021-03-18 02:47:21.249  INFO 50555 --- [nio-8080-exec-4] c.a.s.controller.ExchangeRateController  : RequestType: historical_exchange_rate, Response_Code: 200 OK, Timestamp: 29ms
2021-03-18 02:47:23.409  INFO 50555 --- [    scheduling-1] com.assignment.sm.util.ExchangeRateUtil  : Exchange rate fetched for USD at 2021-03-17T21:17:23.409894Z is : 57938.19
2021-03-18 02:47:33.371  INFO 50555 --- [    scheduling-1] com.assignment.sm.util.ExchangeRateUtil  : Exchange rate fetched for USD at 2021-03-17T21:17:33.371465Z is : 57938.19
```

An important thing to note is that since we are relying on exchange server APIs for fetching the historical dates, if the exchange server is down or not available then there is a simple Hystrix circuit breaker which will break the request after 2 minutes. This can be further enhanced as per the need.

Below is an example of historical rates endpoint :



Request URL

```
http://localhost:8080/btc/exchange/historicalRate?startDate=2021-03-12&endDate=2021-03-15
```

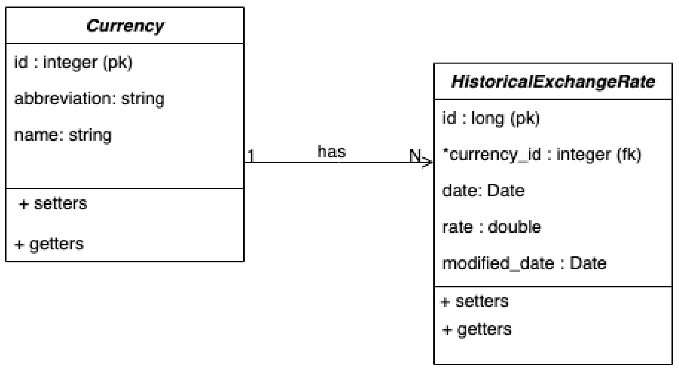
Request Headers

```
{
  "Accept": "*/*"
}
```

Response Body

```
{
  "fromCurrency": "BTC",
  "exchangeRates": [
    {
      "toCurrency": "USD",
      "exchangeRate": 57256.22,
      "date": "2021-03-12"
    },
    {
      "toCurrency": "USD",
      "exchangeRate": 61179.79,
      "date": "2021-03-13"
    },
    {
      "toCurrency": "USD",
      "exchangeRate": 58998.89,
      "date": "2021-03-14"
    },
    {
      "toCurrency": "USD",
```

Class Diagram



A single Currency object with abbreviation as USD is initially saved in the Currency table at application startup because currently only currency rate conversion is supported.
This currency entity has one to many relationship with HistoricalExchangeRate table where currency id is the foreign key.

Brief description of the Service APIs:

<u>ExchangeRateService</u> - the main engine of this application supported by helper utility methods of ExchangeRateUtil.

1. **getHistoricalExchangeRates**
   - This service method receives the targetCurrencyAbbreviation, startDate and EndDate from the controller endpoint
   - Gets the currency object using targetCurrencyAbbreviation from Currency Table and uses it find the available rate data based on the startDate & endDate from HistoricalExchangeRate table.

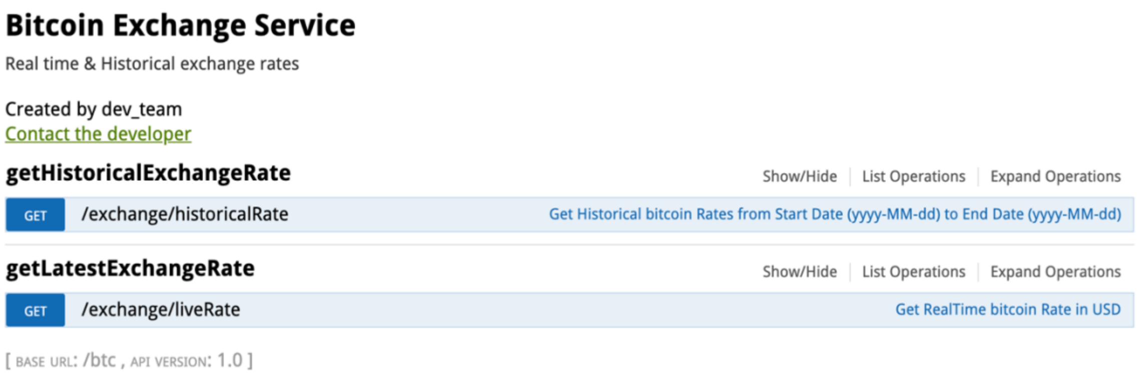
- Checks if any requested dates are missing from the data retrieved from Database. If no, then returns it by converting to POJO using a helper utility.
- If any dates are missing then the exchange rates for those days are fetched using an exchange server using the exchange server endpoint.
- The helper utility plays a major role in forming the request URL and parsing the response from the exchange server.
- These missing exchange rates are then merged with the dB output using the utility method and sent back as response
- Asynchronously, this missing rate data is also saved to the in-memory database for future requests.

2. **getHistoricalRatesFromServer**
   - This service method is responsible for fetching the missing historical exchange rates from an exchange server.
   - It generates the RestEndpoint URL to be used to query exchange server depending the startDate, endDate and the historicalDates present in our DB.
   - Now the required URL needs the last date to be fetched and it will fetch the required missing number of dates above it using a limit parameter which is calculated using a utility method.
   - The limit and the lowest missing date has many cases depending on the range of missing historical dates.
   - After these missing dates have been fetched an asynchronous save call is made to a cache service while simultaneously this data is merged and sent back to the requested endpoint

3. **fetchBitcoinExchangeRate**
   - This is an asynchronous method scheduled with a fixed Rate as per the value configured in the application properties. There is no initial delay.
   - It fetches the real time exchange rates for BTC and saves the corresponding value against USD in a local CurrencyExchange Object.
   - This object is read whenever the realTime exchange rate for BTC to USD is needed.

Swagger:

Swagger endpoint:
http://localhost:8080/btc/swagger-ui.html



Bitcoin Exchange Service

Real time & Historical exchange rates

Created by dev_team
Contact the developer

getHistoricalExchangeRate

/exchange/historicalRate — Get Historical bitcoin Rates from Start Date (yyyy-MM-dd) to End Date (yyyy-MM-dd)

getLatestExchangeRate

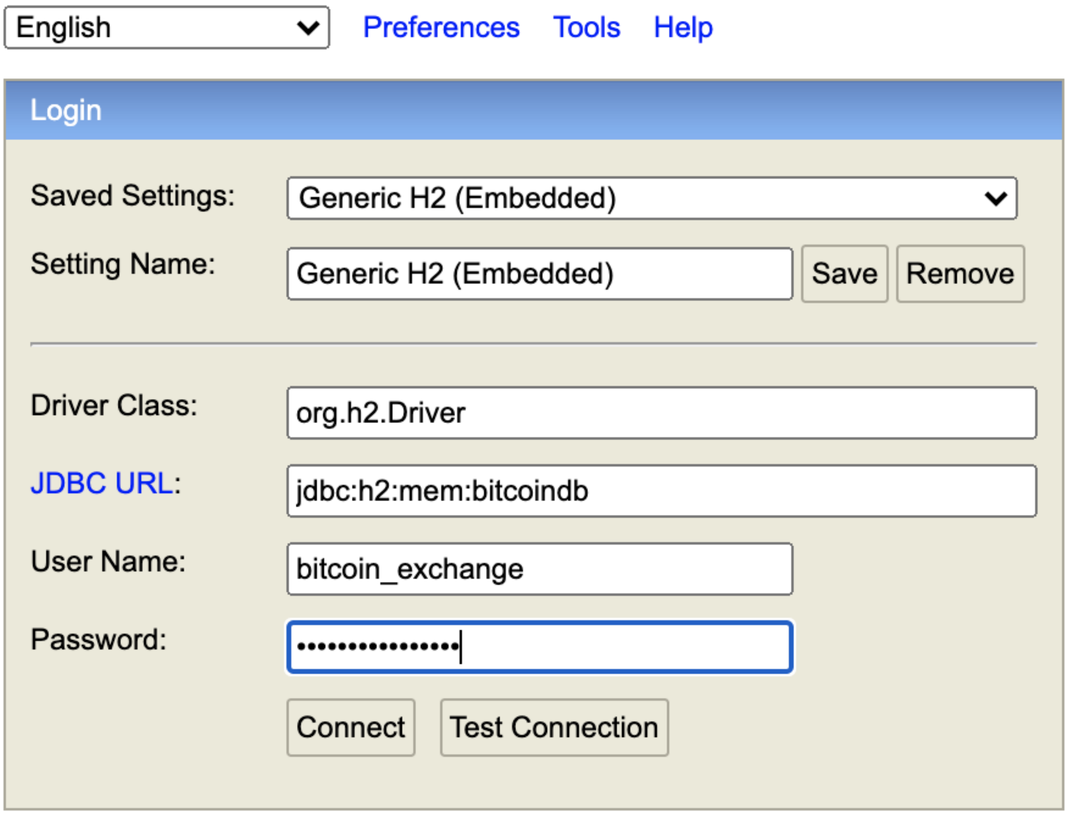
/exchange/liveRate — Get RealTime bitcoin Rate in USD

[BASE URL: /btc , API VERSION: 1.0]

To access h2-console:
http://localhost:8080/btc//h2-console/login.jsp

jdbc url - jdbc:h2:mem:bitcoindb
username - bitcoin_exchange
password - bitcoin_exchange

The same parameters are changeable from application.properties file.



Saved Settings:	Generic H2 (Embedded)
Setting Name:	Generic H2 (Embedded)
Driver Class:	org.h2.Driver
JDBC URL:	jdbc:h2:mem:bitcoindb
User Name:	bitcoin_exchange
Password:	•••••••••••••••

Steps to start the application :

Note : Maven and Java 11 is needed to run the application

After unzipping, Navigate inside the project directory.

1. Run the bash script file in the base directory.
   $ sh start.sh
   This will create a docker image and run the application in a container.

2. Alternatively, it can be run using maven without a docker container as well using
   mvn spring-boot:run

3. Also, we can start it from an IDE like eclipse or IntelliJ once the project has been imported and built using maven.