Technology stack

- SpringBoot 1.5.10
- Java (Oracle) 1.8
- JPA
- Hibernate 5.0.12
- H2 embedded Database
- Junit
- Mockito
- MVN (3.5.2) optional command line build

Overall

In the interests of following the requirements, various open source components were used to provide speedier development and libraries.

SpringBoot framework was used, with an eclipse development environment. The REST-API was provided by SpringBoot. The database was designed by using entity models and a Repository. Hibernate was setup to auto create the database on start-up and H2 was used as an embedded database.

A hybrid BDD/TDD development approach was undertaken. A high level design goal driving a TDD development phase driving detailed code production. Some template code was produced with no function, for further development.

Access to the REST API is via the following URLz

/payment/all – return all payments

/payment/delete/{id} – delete a payment, id is the payment id

/payment/update/{id} – update payment with supplied id. Data should be fully supplied

/payment/get/{id} – get single payment for payment id supplied

Detail Implementation

The project has been developed in line with standard REST-API package structure.

The data models are in **com.sample.f3.payment.model**.

The database interfaces are in **com.sample.f3.payment.repository**

The controllers are in **com.sample.f3.payment.controller**.

The service interfaces for autowiring are in **com.sample.f3.payment.service**

The service implementations are in **com.sample.f3.payment.serviceImpl**.

Other miscellaneous utilities are in **com.sample.f3.payment.utils**

The data was broken down in to the following structures.

- `Payment`
  - `- Attributes`
  - `- Beneficiary_party`
  - `- Debtor_party`
  - `- Sponsor_party`
  - `- Charges_information`
    - `- Charge_items`
  - `- Fx`

Data verification

Within each **serviceImpl**, there are functions to verify the data. Some basic verification is being performed at the moment in most classes. This needs to be clarified what is valid and what is not so more detailed verification can be conducted.

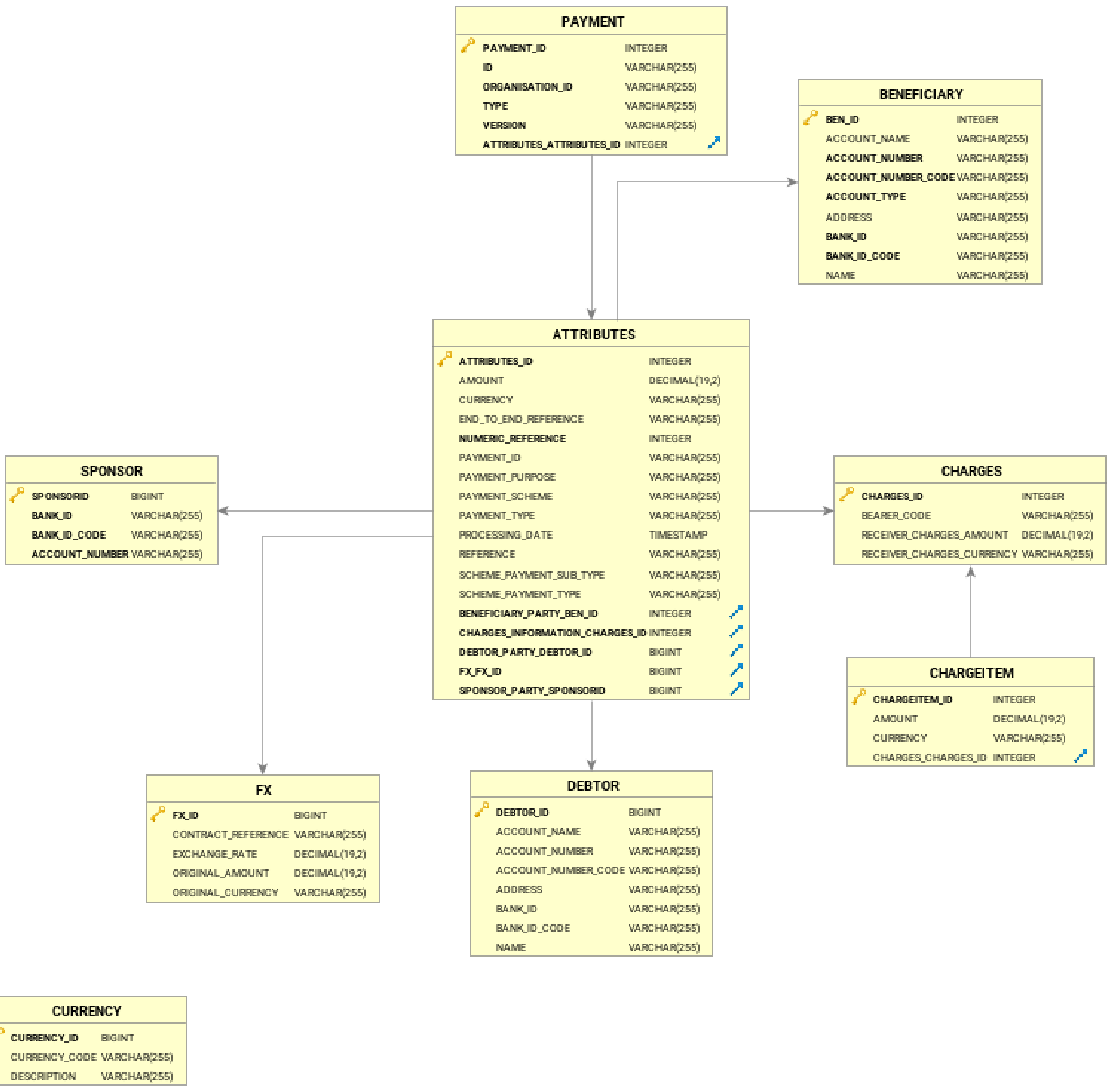
Some further assumptions were made at the input level to decide what data is a requirement for input.

Currency information is checked against a list of valid currencies. This approach would be used more widely once detailed requirements are understood.

Testing

Unit tests were written to drive the function of the code. Where the cases are simple, Junit test cases are used. Where they are dependent on other modules Mockito has been used to restrict the scope of tests. For the controllers a full test is being conducted with the test cases launching the server and requesting REST resources. The data returned has been verified on a sample basis.

Database Structure


PAYMENT
PAYMENT_ID INTEGER
ID VARCHAR(255)
ORGANISATION_ID VARCHAR(255)
TYPE VARCHAR(255)
VERSION VARCHAR(255)
ATTRIBUTES_ATTRIBUTES_ID INTEGER
BENEFICIARY
BEN_ID INTEGER
ACCOUNT_NAME VARCHAR(255)
ACCOUNT_NUMBER VARCHAR(255)
ACCOUNT_NUMBER_CODE VARCHAR(255)
ACCOUNT_TYPE VARCHAR(255)
ADDRESS VARCHAR(255)
BANK_ID VARCHAR(255)
BANK_ID_CODE VARCHAR(255)
NAME VARCHAR(255)
ATTRIBUTES
ATTRIBUTES_ID INTEGER
AMOUNT DECIMAL(19,2)
CURRENCY VARCHAR(255)
END_TO_END_REFERENCE VARCHAR(255)
NUMERIC_REFERENCE INTEGER
PAYMENT_ID VARCHAR(255)
PAYMENT_PURPOSE VARCHAR(255)
PAYMENT_SCHEME VARCHAR(255)
PAYMENT_TYPE VARCHAR(255)
PROCESSING_DATE TIMESTAMP
REFERENCE VARCHAR(255)
SCHEME_PAYMENT_SUB_TYPE VARCHAR(255)
SCHEME_PAYMENT_TYPE VARCHAR(255)
BENEFICIARY_PARTY_BEN_ID INTEGER
CHARGES_INFORMATION_CHARGES_ID INTEGER
DEBTOR_PARTY_DEBTOR_ID BIGINT
FX_FX_ID BIGINT
SPONSOR_PARTY_SPONSORID BIGINT
SPONSOR
SPONSORID BIGINT
BANK_ID VARCHAR(255)
BANK_ID_CODE VARCHAR(255)
ACCOUNT_NUMBER VARCHAR(255)
CHARGES
CHARGES_ID INTEGER
BEARER_CODE VARCHAR(255)
RECEIVER_CHARGES_AMOUNT DECIMAL(19,2)
RECEIVER_CHARGES_CURRENCY VARCHAR(255)
CHARGEITEM
CHARGEITEM_ID INTEGER
AMOUNT DECIMAL(19,2)
CURRENCY VARCHAR(255)
CHARGES_CHARGES_ID INTEGER
FX
FX_ID BIGINT
CONTRACT_REFERENCE VARCHAR(255)
EXCHANGE_RATE DECIMAL(19,2)
ORIGINAL_AMOUNT DECIMAL(19,2)
ORIGINAL_CURRENCY VARCHAR(255)
DEBTOR
DEBTOR_ID BIGINT
ACCOUNT_NAME VARCHAR(255)
ACCOUNT_NUMBER VARCHAR(255)
ACCOUNT_NUMBER_CODE VARCHAR(255)
ADDRESS VARCHAR(255)
BANK_ID VARCHAR(255)
BANK_ID_CODE VARCHAR(255)
NAME VARCHAR(255)
CURRENCY
CURRENCY_ID BIGINT
CURRENCY_CODE VARCHAR(255)
DESCRIPTION VARCHAR(255)

The sample JSON data, was broken down into blocks represented by tables above. In some cases there appears to be duplication of data structures and those might be better placed in a different table. However it was decided not to change the JSON structure and so data has been kept true to the model supplied.

A currency table has been added, as a list of items that a valid for currency. It would be prudent to extend this to introducing tables for payment types, bank codes, and other items where a fixed range of values exist. Dependent on performance requirements this might be better served with a fixed list of data.

To Do

- Add useful logging messages
- Add useful return messages
- Add data validation
- Performance testing
- Reduce DB I/O
- Increase Test Coverage for more cases at controller level
- Add Test coverage for Model