

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.

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



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