Payment Service

Requirements

Provide a design for part of a payments API. A list of payments might look like

http://mockbin.org/bin/41ca3269-d8c4-4063-9fd5-f306814ff03f

Design requirements:

- API should be RESTFUL
- API should be able to:
  - Fetch a payment resource
  - Create, update and delete a payment resource
  - List a collection of payment resources
  - Persist resource state (e.g. to a database)

REST API

The REST API follows the following scheme :

Method	**Path**	**Description**
GET	*/*	Main index
GET	*/payments*	List of payment objects
POST	*/payments*	Create new payment
GET	*/payments/:id/:version*	Get payment for given id and version.
PUT	*/payments/:id/:version*	Replace latest version of payment for given id, with the one sent in the body
PATCH	*/payments/:id/:version*	Update payment for given id and version, with the payment fragment sent in the body
DELETE	*/payments/:id/:version*	

In the above paths, if *:version = 0* then the operation is performed on the latest version.[1]

1 We can add redirects for /*payments*:id to /*payments*:id/0 if necessarry.

The REST API returns a json data structure consisting of :

- An *error-message* and *error-code* (empty if no error has occurred)
- An array of *payment* objects (optional) (see example data file)[2].
- An array of HAL-like *links* for the client to perform related functions on the returned payments.

Domain

The domain is *payments.*

A *payment* is defined as the *credit* of *money* to a *beneficiary.* This *money* is called the *credit money*.

Each *payment* is associated with a *debit* of *money* from a *debtor.* This *money* is called the *debit money*.

Money is a strictly positive *amount* of a specified *currency*. A payment of a zero *amount* of *money* is invalid.

Each *debtor* is defined in terms of an *account* from which the *money* is *debited*.

Each *beneficiary* is defined in terms of an *account* into which the *money* is *credited*.

The *credit-money* and *debit-money* can be in different currencies.

In a payment, if the *credit money* and the *debit money* have the same *currency*, the *amount* must also be the same for both.

An *exchange rate* can be multiplied with *money* to give *money* in a different *currency.* The *exchange rate* is specified in the *foreign exchange (fx)* contract.

In a payment, if the *credit money* and the *debit money* have a different *currency*, the *amount* can be different. The multiplication of the *exchange rate* with the *credit money* should equal the *debit money*.

A *payment* incurs one or more *charges*[3]

Layers

Although the service is very simple, it is assumed that there is probably a rich set of business rules to enforce correctness of each payment. Also it is quite possible that we may need to support more complex types of payment in the future which might not follow the same form. Furthermore, when looking at the example dataset, it is clear that there is some duplicated structure i.e. *beneficiary_party* and *debtor_party* represent some kind of *account*, and in several places we have an (*amount, currency)* pair which hints at a *money* type. Hence, a domain model is probably a worthwhile investment.

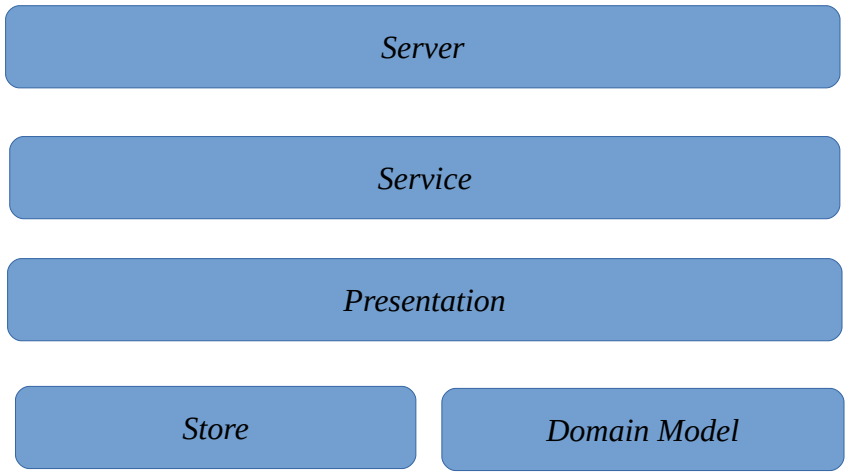
We should also separate the data *store* logic from the domain and the persistence, as we may need to change the back-end data storage in the future.

2 Currently only GET /payments can return more than one payment object.
3 It is not clear what these charges are and two who they apply.

Since our model is likely to have a different structure than the example dataset, we should separate the presentation logic from the business rules in the domain. This also allows us to generate different descriptions of the data for different types of needs in the future, whist keeping the validation logic and persistence logic decoupled from this.

A *service* layer will provide an API to perform the top-level operations on the resources. This *service* layer will be the main entry-point for a *server* layer which will actually implement communications to the client (e.g. the REST API). The *service* layer will be entirely independent of the communications, such that it should be easy to implement a different kind of *server* without changing the *service* layer.

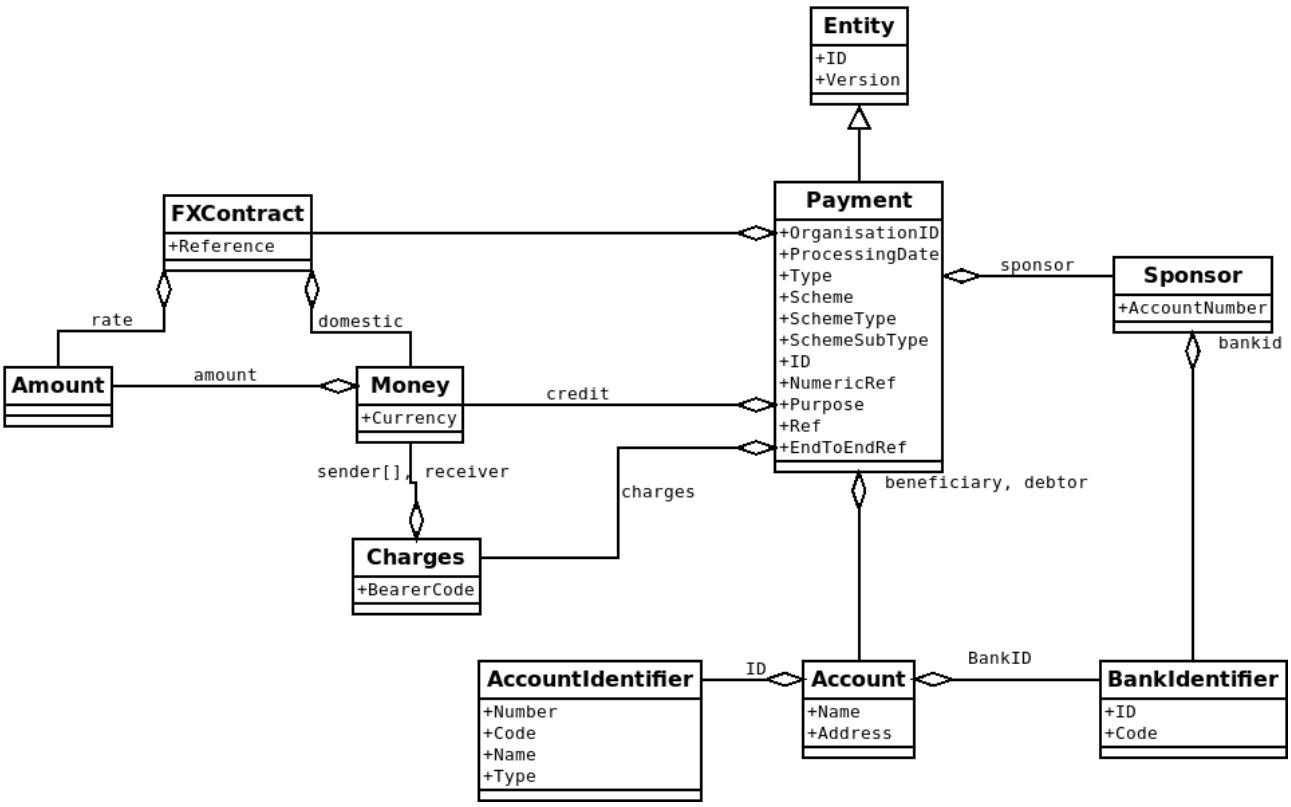
So our layered stack looks like this.



Domain Model

The model is based on a lot of assumptions made by looking at the data. Further input needs to be gathered from the business to correct and enrich the domain model. We attempt to formalise some terminology here.

An outline of the model is shown below :



Organisation ID <Value>

An organisation-id is some opaque, non-empty, external string[4] provided to the service. Hence in this model we don't consider *organisation-id* to be an *entity*, it is simply a *value*.

This means we explicitly make the decision to allow any *organisation-id* to be used in transactions with the domain, without validation. If we did need to validate this field, we would need to some way to bring valid valid *organisation-id*s into domain, which adds complexity.

4 From the data it looks like this is a UUID, but since an organisation entity is outside the scope of the payments domain, we must assume a more general form to avoid coupling.

Payment <Entity>

A payment has a set of attributes, all of which are considered to be *values* :

- *ID* (required) – UUID - immutable
- Version *(required)* - incrementing integer, >=0
- An *organisation-id* (required)
- The *credit-money* which is required
- A *processing-date* which is required
- The *beneficiary* which is required
- The *debtor* which is required
- The *charges* information which is required
- The *foreign-exchange* contract which is not required[5]
- The *sponsor* which is required[6]
- Payment classification information :
  - A *payment-type* which is required[7]
  - A *payment-scheme* which is required
  - A *scheme-payment-type* which is required[8]
  - A *scheme-payment-sub-type* which is required[9]
- External payment identities, references and descriptions :
  - A *payment-id* which is required[10]
  - A *numeric-reference* which is required[11]
  - A *payment-purpose* which is not required
  - A *reference* which is not required
  - An *end-to-end reference* which is not required[12]

5 Making this a required field would make the payment unnecessarily confusing in the case of a payment in the same currency. A fake fx contract would need to be constructed between the same currencies and having an exchange rate of 1.
6 Assumed but could be dependent of payment type
7 This is “Credit” in the data, does it make sense for a payment to be a “debit”?
8 Its not clear if this is free-form text or an enum
9 Its not clear if this is free-form text or an enum
10 Assumed to be external to the model
11 Assumed to be external to the model
12 Assumed

Debtor <Value>

A debtor is an *account.*

The (account_number, bank_id) cannot be the same for both the debtor and beneficiary)

Beneficiary <Value>

A beneficiary is an *account*

The (account_number, bank_id) cannot be the same for both the debtor and beneficiary)

Account <Value>

An *account* is a value consisting of :

- account-identifier (required)
- bank-identifier (required)
- *name* (required) – free-form text
- address (required) – free-form text

All of these field must be set to non-empty values[13].

Account Identifier <Value>

This is a value consisting of :

- an *account-number*
- an *account-code (IBAN, BBAN)*[14]
- *an account-name (required)*
- *an account-type (optional)* integer[15]

The *account-number* string is validated against the internal structure of the *account-code* where possible.

Bank Identifier <Value>

This is a value consisting of a *bank-id* and *bank-id-code.* The internal structure of the *bank-id* is validated where possible[16].

13 It is assumed for simplicity that there is no other validation to be done on these fields
14 https://www.iban.com/glossary
15 It is not known what this integer means
16 http://www.sepaforcorporates.com/swift-for-corporates/list-ncc-national-clearing-codes/

Amount <Value>

An *amount* is a value.

It is a number with a fixed number of decimal places.

An a*mount* can be added to another *amount*, provided it gives a valid *amount.*

An a*mount* can be subtracted from another *amount*, provided it gives a valid *amount.*

An amount be multiplied by another amount, provided it gives a valid *amount*.

An *amount* can be rounded to the nearest precision decimal places.

Currency <Value>

A *currency* is a value representing an international currency[17]. e.g. GBP

Money <Value>

Money is an *amount* of a specified *currency.*

Money is quantified in terms of its smallest denomination e.g. in USD you cannot have fractions of 1 cent.

Money can be added to money of the same currency, provided it gives a valid amount of the same currency[18].

Money can be subtracted from money of the same currency, provided it gives a valid amount of the same currency[19].

Money can be multiplied by an *amount*, provided it gives a valid amount of the same currency.

Exchange Rate <Value>

An exchange rate is the *amount* of the *domestic-currency* required to purchase 1 *amount* of the *foreign-currency*.

An exchange rate with *domestic-currency* X can be applied to *money* of *currency* X, to yield *money* in the *foreign-currency Y*.

17 https://www.iso.org/iso-4217-currency-codes.html
18 In fact we might not actually need this functionality
19 In fact we might not actually need this functionality

Foreign Exchange (FX) Contract <Value>

The FX Contract consists of :

- a contract reference (required) – free-form text
- an *exchange-rate (required)*
- *domestic-money (required)*

In a *payment, t*he *credit-money* multiplied by the *contract exchange-rate* should equal the *domestic-money*.

In a *payment* the *domestic-money* of any *fx-contract* represents the *debit-money*.

Charges <Value>

Charges have a :

- bearer-code (required)
- an array of sender charges as *money*[20] (required)
- receiver-charge as *money* [21] (required)

Sponsor <Value>

A *sponsor* is a value consisting of :

- account-number (required) which is a string[22]
- a *bank-identifier* (required)

20 Presumed to be a single amount for each currency of the payment, but not sure.
21 The data suggests this is in the sender currency, but not sure if this is a restriction.
22 It is not clear if this should be just a string, or an *account-identifier*