Payment API

Functional design

The application exposes basic RESTful CRUD functionalities for payments.
A payment is defined as follows:

- Id
- Payment type (defaulted to Payment)
- Version number
- Organisation ID
- Attributes

Attributes contain the following categories:

- Payment information (amount, currency, type, reference, scheme)
- Beneficiary party (the payment receiver)
- Debtor party (the payment sender)
- Charges information (information about additional charges for sender and receiver, plus currency and bearer)
- Fx (information about currency exchange)
- Sponsor party (bank details)

The application will provide functionality to insert, update and delete a payment. It will also be possible to retrieve a list of all payments in the system as well as a single payment given its id.

The application will validate the input and reject the request when all mandatory fields have not been provided. For now we will assume only payment id, type, version number and organisation ID will be mandatory. We will extend validation to the attributes in the future when required.

When an error occurs, be it a validation issue with the input provided or with the operation the user is trying to execute (i.e. trying to retrieve a non existing payment), the application will provide the appropriate feedback to the user in the form of one or more error messages.

Technical design

The application will be written in Java and Spring Boot framework. It will be structured following the layout described below.

Resource

Top layer where the RESTful endpoints will be mapped. The app will expose the following resources:

- **GET** /v1/payments - will return a list of all payments in the system
- **GET** /v1/payments/:paymentId - will return a specific payment, given its id, or an error if the payment doesn't exist
- **POST** /v1/payments - will create a payment and return the newly created payment, or an error if a payment with the same id already exists
- **PUT** /v1/payments/:paymentId - will update an existing payment and return the newly updated payment, or an error if a payment with the given id doesn't exist
- **DELETE** /v1/payments/:paymentId - will delete an existing payment and return the newly deleted payment, or an error if a payment with the given id doesn't exist

The resource layer will also be responsible for validating the input provided

Service

This layer will be responsible for routing the requests from the resource layer and fetch/push resources to the DAO layer.

The service layer will also be responsible for mapping between model objects (those which provide the API representation) and domain object (the ones that provide the repository format, i.e. what is written in the database).

DAO

It will consist of an interface providing basic methods to create, read, update, and delete from a repository. It will also have a database implementation providing connectivity to a database.
The database implementation will use Spring Boot's support for JPA/Hibernate. Hibernate will be configured to automatically create the schema, if not present. This option will have to be revised when deploying to a production environment.
For now, only a MySql version will be provided, but the app will be able to support every database that provides a JDBC connector library.

Testing

The app will have comprehensive unit test coverage, plus will be provided with full stack integration tests. Spring Boot provides several utilities to write integration tests, so a version with an embedded database will also be provided.