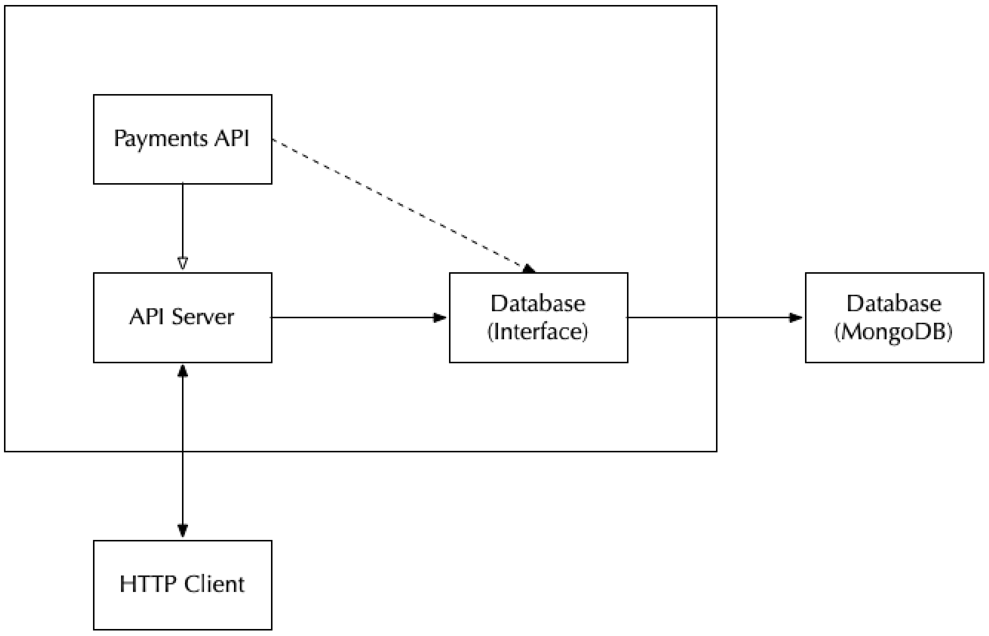
Payments API: Design Overview

Table of contents

Introduction

This document aims at providing an overview of the design of a payments API. The architecture and the data model itself are kept deliberately simple, but are designed with extensibility in mind.

Architecture



Overview of the internal and external architecture

API Server
Based on the assumption that the product of this design may need to be extended in the future in order to provide APIs for managing resources other than payments, an *API server* is to be introduced as the core component of the solution. The API server is a simple HTTP server to which APIs such as the *Payments API* can be plugged (*registered*). The API server is responsible for handling concerns such as (un)marshaling data, logging and tracing requests.

Since most (if not all) of the APIs will require storage at some point, the API server is to be provided with an *interface* to a database. This allows the API server to provide storage capabilities to all registered APIs without tightly coupling them to a particular database engine. The API server is able to understand and report whether the target database is online or not.

The host and port at which the API server serves requests is to be provided at startup time via a command-line flag.

Database
For the sake of simplicity, MongoDB is chosen as the actual database engine used to power the API server (and hence the *Payments API*). However, and as mentioned above, this represents an implementation detail since access to the database by the API server is to be performed via an interface. It must be possible to replace MongoDB entirely with a different database engine with minimal changes to the codebase.

The URL at which MongoDB can be reached is to be provided to the API server at startup time via a command-line flag. The same happens with respect to the name of the database to use.

Payments API
The *Payments API* is a RESTful API which will allow CRUD operations on payments. All endpoints that receive or return a `Payment` object are to support `application/json` alone.

The `Payment` object
An example of a full `Payment` object is provided below:

```
{
    "id": "5cc9b9b2c3c45afaf2855941",
    "beneficiary": {
        "account_number": "1234",
        "bank_id": "4321",
        "name": "Bruno"
    },
    "debtor": {
        "account_number": "5678",
        "bank_id": "8765",
        "name": "Dave"
    },
    "amount": 314.15,
    "currency": "EUR",
    "date": "2019-04-30T22:30:00Z",
    "description": "Order #1"
}
```

As mentioned above, the model is kept deliberately simple, but can be extended in the future shall the need for that ever arise.

Creating a payment

To create a payment, one is to send an HTTP `POST` request to `/payments` with a `Payment` object as the request body. All fields besides `id` are mandatory. A `201 CREATED` HTTP response is to be returned upon successful creation. The response is to carry the full `Payment` object (including its ID) in its body.

Listing payments

To list all registered payments, one is to send an HTTP `GET` request to `/payments`. A `200 OK` HTTP response containing a (possibly empty) list of `Payment` objects is to be returned.

Getting a payment

To get a payment by its ID, one is to send an HTTP `GET` request to `/payments/<id>`, where `<id>` is the ID of the payment. A `200 OK` HTTP response containing the full `Payment` object is to be returned when a payment with the provided ID exists. Otherwise, a `404 NOT FOUND` HTTP response is to be returned.

Updating a payment

To update a payment by its ID, one is to send an HTTP `PUT` request to `/payments/<id>`, where `<id>` is the ID of the payment. The HTTP request is to contain the ***full***, updated `Payment` object in its body. A `200 OK` HTTP response containing the full, updated `Payment` object is to be returned when a payment with the provided ID exists and can be successfully updated. Otherwise, a `404 NOT FOUND` HTTP response is to be returned.

Deleting a payment

To delete a payment by its ID, one is to send an HTTP `DELETE` request to `/payments/<id>`, where `<id>` is the ID of the payment. A `204 NO CONTENT` HTTP response is to be returned when a payment with the provided ID exists and can be successfully deleted. Otherwise, a `404 NOT FOUND` HTTP response is to be returned.