Payments API

Tools and frameworks

Solution to this recruitment exercise was written with following tools and frameworks:

- Programming language: Java 8
- Spring Boot, as an all-in-one framework for API development, with extra JPA and test dependencies, providing dependency injection engine by default
- Databases: Postgres for "production" deployment, in-memory H2 database for integration tests
- Packaging tool: Apache Maven

Key design decisions

- **Separation of domain model and resource representation**
  Entity classes, corresponding to database tables, are located in `com.alwozniak.form3.domain` package. They represent the core of the system and their instances are not meant to be directly returned as serialized API resources. This function is fulfilled by classes located in `com.alwozniak.form3.resources` package. Model separation keeps business domain independent from returned resources and allows for each of these layers to evolve independently. Also, some implementation details of domain model are hidden from API users.
- **Layered architecture**
  Solution's architecture consists of 4 layers:
  - Domain model (entities): these should be the classes where most of domain business logic go. Since there was not much information about payments processing and associated business rules, their implementation is kept simple.
  - Repositories: classes responsible for retrieving entities from the database and persisting them
  - Services: in the current implementation there is only one service, `PaymentsService`, responsible for translating between the resource and the domain models.
  - Presentation layer (controller + resources): this layer includes resource classes, and a controller handling HTTP requests
- **Payment id generation**
  Payment ids are automatically generated by an ORM library used in this project (Spring JPA), and as a result, API's does not support `PUT` method for placing a resource under a requested URL with desired UUID. Hence, updating an existing resource is possible only through a `PATCH` request. New resources should be created via `POST` requests.

- **Approach to testing**
  This solution was implemented with Test-Driven Development method, with a goal to keep very high test code coverage.
- Security features (authentication and authorization) are not implemented in this solution.

`/payments` endpoint

API exposes on endpoint: `/payments` with `GET`, `POST`, `PATCH` and `DELETE` methods supported. Example request and response bodies can be found in the solution's repository in `example-requests` folder.

`GET /payments`

Returns a list of payment transactions existing in the system. Current implementation does not provide filtering nor pagination, therefore a full list of persisted payments is returned.

Expected response:

- Status code: `200 OK`
  Body: JSON containing a list of payment resources with a schema matching examples specified in http://mockbin.org/bin/41ca3269-d8c4-4063-9fd5-f306814ff03f

`GET /payments/:id`

Returns a single payment resource. `id` parameter should be a UUID of an existing payment transaction.

Expected responses:

- Status code: `200 OK`
  Body: JSON representing a single payment resource
- Status code: `404 Resource not found`
  Body: Error message indicating that a resource of a given id is not present
- Status code: `400 Bad request data`
  Body: Error message indicating a request error. It may, for example, notify that an id passed in request URL is not a valid UUID.

`POST /payments`

This endpoint can be used to create a new payment entity. Request body should contain a valid single payment resource without id. Check `example-requests` folder for a full example.

Expected response:

- Status code: `201 Created`
  Body: empty

`PATCH /payments/:id`

Allows for updating an existing payment resource. `id` parameter should be a UUID of existing payment, request body should contain fields of this payment that should be updated. There is no need to pass a full resource data in the request body -- only the fields that require modification.

Expected responses:

- Status code: `200 OK`
  Body: JSON representing an updated payment resource
- Status code: `404 Resource not found`
  Body: Error message indicating that a resource of a given id is not present in the system

DELETE /payments/:id

Use this method to delete a payment entity and all associated records.

Expected responses:

- Status code: `204 No content`
  Body: empty
- Status code: `404 Resource not found`
  Body: Error message indicating that a resource of a given id is not present in the system