Payments API Design

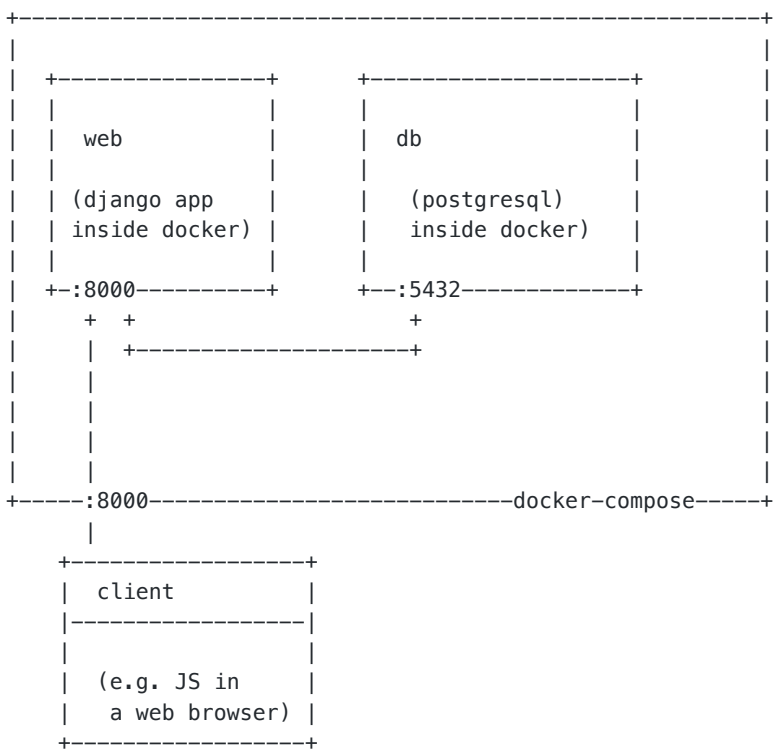
Endpoints

HTTP server providing the following endpoints:

Method	URL	Description
GET	/api/v0/payments/	returns list of payments e.g. [{"id": "244....}, {"id": "987...}]
POST	/api/v0/payments/	create a new payment resource - expects JSON object
GET	/api/v0/payment/<uuid>/	returns a single payment as a JSON object
PATCH	/api/v0/payment/<uuid>/	updates an existing payment resource. It expects a partial payment JSON object. Updated object is returned
DELETE	/api/v0/payment/<uuid>/	deletes an existing payment resouce.

Architecture

For development and tests application and database is containerised using docker.



Rationale

I decided to use a transactional database as projects deals with money transactions - payments. Even it might not scale easily it guarantees that after client request its payment is saved into the disk and not just intermediate queue or a cache. Payments are typical CRUD matrix application so Django+DjangoRestFramework and classical SQL database serves well as are easy to set up. Using docker is a convinient lightweight solution for local development, testing and deploying application (on e.g. Kuberentes, EKS, OpenStack). Database model is denormalised by the use of JSONField. Model correctness is verified on the API level (djangorestframework serializers) and not on the ORM-level. It was simply faster to implement a nested dictionary rather than set of models and their relations. In case of introducing api endpoints representing Organisation or Charges then normalising the database model into individual classes/tables would make sense.

There's not authentication and authorization as the task description didn't mention anything about Users and priviliges. JWT could be use to authenticate JavaScript requests.