Payment Service

This task deals with implementing a payment service that fetches data for a client using two engines, a payment engine and a customer engine.An overall challenge in this task is that the implementation of such a service should follow and map our (undefined) business concepts and logic - going all the way from structuring and naming databases to implementing functionality in our UIs.

Initial thoughts: Payment admin (?)

A user can login and get an overview of all payments and customer information. We will follow a classic crud flow, so a user should be able to list and edit all payments, as well as create and delete.
A payment exists of

- amount
- currency
- timestamp
- customer_name
- customer_birth_date
- customer_birth_place

perhaps we want to add extra less-static customer details like email, address and banking information/payment method. We could also consider having/getting:
-customer types (business/private, subscription/one-time, ..)
-payment type?
-payment status?
-orders/receipts/transactions?
-do we need to be able to manage customer information?
-dealing with payments we want authorization (token for instance), but do we need permissions? Some users can view payments, some users can edit?

Implementation

For the implementation of the architecture I have used

```
>rails plugin new payment_engine --mountable
```

for the payment and customer engine and

```
>rails generate scaffold Payment id:integer amount:integer
currency:string
```

for the payment and customer setup. Otherwise I have added some minor changes:

- a folder for **services** that deals with implementing logic for fetching payments and getting customer information from our customer engine

-the payments engine has views (the controllers should be updated to send json objects), but the customer engine returns json

-I have made some comments/suggestions to the models, controllers and services

Questions/topics

-Do we have a database structure for tables and relations, or should this also be included in the planning?

-Will anybody else in our system need payment and customer data? Should we include these demands in the planning of our apis?

- personal experience
    - pagination, filtering, api agreements
    - heavy queries, n+1
    - single-responsibility
    - clean views, clean controllers

- **APIs**
- **Testing coverage**
- **engine communication**
- **SOLID**
- **Authorization**
- **Client/UI**
    - components and contexts
    - clean apis, filtering
    - style guide, css, …

Notes

This task has been fun and also very challenging for me because I had to set guidelines and limitations myself. I have had a hard time restricting my focus :-)

Hopefully it can function as a starting point for a good conversation. I look forward to talking about it!