Feature:

Create a simple RESTful API service to manage bank transactions.

Scenario:

A client can create a bank transaction through the given API.

Bank transaction common data should be stored in a `bank transaction` table, each part of the bank transaction should be stored in `bank transaction part` table.

Transactions have different parts, distinguished by reasons. Each part has its own type.

Transaction must have at least one part.

Given:

1. Entities:
    a. `bank transaction` entity:
    ```
    'id': <Int>
    'uuid': <Uuid>
    'amount': <Decimal>
    'booking date': <DateTime>
    ```

    b. 'bank transaction part' entities:
    ```
    'id': <Int>
    'bank_transaction_id': <BankTransaction>
    'amount': <Decimal>
    'reason': <String>
    ```
2. Reasons of transaction parts and their types:
    a. `debtor_payback`: <DebtorPayback>
    b. `bank_charge`: <BankCharge>
    c. `payment_request`: <PaymentRequest>
    d. `unidentified`: <Unidentified>

When: Client sends a transaction with parts
Then: Transaction and its parts are properly stored in the database
And: Client receives a proper response from API

Scenario:

A client provides uuid of a transaction and wants to receive its data together with parts and parts data.

Each part in PHP code should be represented as a different type, according to its reason (for example, `debtor_payback` bank transaction part is a <DebtorPayback> type).

When: Client sends a request for an explicit transaction
Then: Client receives the transaction with its data
And: Transaction has a list of parts containing their data

Example of a transaction data sent to API:

```
{
'amount': 9.99,
'booking_date': '2018-01-01 12:00:01',
'parts': [
  {
   'reason': 'debtor_payback',
   'amount': 2.00
  },
  {
   'reason': 'bank_charge',
   'amount': 1.00
  },
  {
   'reason': 'payment_request',
   'amount': 1.50
  },
  {
   'reason': 'unidentified',
   'amount': 1.50
  },
  {
   'reason': 'unidentified',
   'amount': 2.00
  },
  {
   'reason': 'debtor_payback',
   'amount': 1.99
  }
 ]
}
```

It means, that we should have array collection with:
- 2x DebtorPayback objects (and records in DB, with different data)
- 1x BankCharge object
- 1x PaymentRequest object
- 2x Unidentified objects

Required functionality:

- …is described in scenarios above

Technical constraints:

- Use all possible, known to you, best practices of coding.
- Use PHP 7.x and any framework you prefer
- Save the entities in a database. You can use any relational or NoSQL database.
- The exceptions need to be properly handled.
- Cover your code with tests. The technology and the type of testing is up-to-you to choose.

Delivering the task:

- Provide a straight-forward method of running your project.
- Use Github to share your project, we would love to see your commits ;-)

Bonus points:

- A working Dockerfile or Docker-compose configuration to run your project and any dependencies, like the database is provided.
- A proper API-Documentation is written.
- The Cache headers, including the Etag header, are used.
- The data which is sent by a client is properly validated