Kernolab PHP developer task

Story(use case)

Company X which has an EMI license, has a social system where users can top up their balance and spend money for boosting their social experience. Company X decides to allow users do funds withdrawal, so they now need system which will allow users make transactions(payments) to their bank accounts. They hire you, to develop separate API for those transactions(payments). Also, company X requires you to use at least two transaction providers(processors) for different processing depending on currencies.

Goal

Your goal is to build a **STATELESS** API which allows transaction creation and processing via persistent steps:

1. Transaction persisting to database with fee calculation. (note: Total transaction amount is amount + fee.)
2. Transaction submission with 2FA (assuming code is 111 always for development purposes) code, only new transaction can be submitted.
3. Transaction processing via command, **only transaction which is submitted can be processed**. Processing means that transaction status is completed after processing command is finished.
4. Transaction pulling(getting) for data view, which shows transaction information, such as status, fee, etc.
5. API error on unsuccessful transaction creation.

Rules(transaction creation/processing logics)

1) User can make only ten transactions per hour. Otherwise he gets API error.
2) Each transaction has a fee of 10% unless **total amount without fees** of user daily transactions (excluding current one) are more than 100 (in any of currencies), then fee is 5%. Even if transaction is not submitted(submission is in goal 2) it counts.

For example: user has total daily amounts for all of his transactions: 10 eur and 110 usd. Then for each next transaction, fee will be only 5%, even if currency is not the one with amount > 100(in this case 10). So now new transaction with amount of 40 will have a fee of 2(5%) units(currency in which transaction is created).

3) Maximum total amount (total available balance per user for all time) users can transfer is 1000 in one currency (so 1000 eur, 1000 usd, etc. is max). Total amount is calculated by summing total amounts of transactions. Total amount of transaction is amount + fee (example: 100 + 100 * 10% = 110 total amount)

4) When currency is "eur" then we use provider called "megacash", when transaction is in other currency, we use provider called “supermoney". Each of them apply own rules to processing:

Megacash provider substrings details of transaction to length of 20
Supermoney provider appends random integer to transaction details

Main notes (kind of FAQ)

1) Let’s assume user is authenticated, so we can skip authentication part for API.
2) More providers can be added by custom rules(from management for example) in the future
3) For unsuccessful responses we have to return formatted(JSON) response like we do with successful ones.
4) Background task/script is needed to process confirmed (submitted with code 111) transactions. It's possible to make a call to web to process all transactions too, but background task is preferred.

5) No relation of user needed. We assume that system knows user. We calculate amounts by transactions assigned to some user_id.
6) Fees are calculated on the transaction creation, so when user creates transaction he receives fee in response.

Success flow

1) User submits data via API about transaction, which consist of:

user_id (identifier of user in integer)
details (string)
receiver_account (string)
receiver_name (string)
amount (decimal)
currency (string, EUR, USD, etc.)

2) Then user confirms (submits) transaction with code, lets assume code is 111.
3) Background task, completes transaction.
4) We pull transaction data from API and see status completed. Also, we see fee applied to that transaction.

Example transaction initiation request (step1):

```
{
	"user_id": 1,
	"details": "Transaction number one",
	"receiver_account": "12345",
	"receiver_name": "Name Surname",
	"amount": 20.00,
	"currency": "eur"
}
```

Example transaction submit request (step2)

```
{
	"code": 111
}
```

Example response when pulling transaction after completion (step4):

```
{
	"transaction_id": 1,
	"details": "Transaction number one",
	"receiver_account": "12345",
	"receiver_name": "Name Surname",
	"amount": 20.00,
	"currency": "eur",
	"fee": 2.00,
	"status": "completed"
}
```

Technical requirements:

Task has to have README how to launch everything
PHP 7+ preferred
Only OOP in business logic
Code must follow best practices (design patterns, SOLID, YAGNI, etc.)
Database (MySQL or PostgreSQL)
API must be stateless REST
Request/Response in application/json

Tests (functional + unit), coverage to at lest see how those are written

Notes:

Task can be done with any framework, main goal is to see hard and soft skills, business logic and how those are written, code style, best practices, architecture, etc.
If there is a possibility, vagrant image or docker image is preferred, for faster launch.
Frontend is not necessary, up to you.

This task is just an idea and rules for you to apply using your knowledge. If you don't know how to do any of task parts, or think that rule is not like you would do, then do it your own way and how you think it should be. Just don't forget to metntion that in README.

For any unfinished(if you think so) part add notes to README or to email body.

Solution:

For any questions relating the task contact via sergej@kernolab.com. Solution can be sent via zip or other (vcs, dockerfile, etc.) methods you prefer.

Time for solution is one(up to 2 weeks if some delays occur) week, if you see that you have some delays, please inform.