Test Task

Please implement a composer-based library/package using PHP 7.x.

Final result expected to be provided as a GIT-repository (it's not required to host the repository; zipped folder would be enough).

There is no time limit for the task - feel free to implement it at your own pace.

- Data structure & contracts
  - Value Objects
    - ValueObjectInterface
    - Product
    - Currency
  - Data Contracts
    - BasicFilter
    - Filter
    - CurrencyRate
  - Service Contracts
    - DatabaseConnectionProviderInterface
    - HttpBasicRequestStackInterface
- Utils & helpers
  - Util
    - baseCurrencyFromRequestQuery
    - productsFromRequestQuery
  - DatabaseConnectionProvider
  - DatabaseConnectionFromPdoProvider
  - HttpBasicRequestStack
- Exceptions
  - ValueObject\AbstractInvalidValueException
  - ValueObject\InvalidProductValue
  - ValueObject\InvalidCurrencyValue
  - MissingQueryParameterException
  - MissingCharsetConfigException

Data structure & contracts

Value Objects

ValueObjectInterface

```
/**
  * @return array
  */
public static function getValidValues(): array;

/**
 * @return string
 */
public function getValue(): string;

/**
 * @param ValueObjectInterface $valueObject
 *
 * @return bool
 */
public function equals(ValueObjectInterface $valueObject): bool;
```

Product

implements `ValueObjectInterface`

Valid values: ECOM, POS

Currency

implements `ValueObjectInterface`

Valid values: EUR, GBP, USD the main currencies

Data Contracts

BasicFilter

- from: DateTimeInterface; time is always 00:00:00 (start of day) - DateTimeImmutable should be used to store value
- to: DateTimeInterface; time is always 23:59:59 (end of day) - DateTimeImmutable should be used to store value
- products: Product[]
- previousFrom: DateTimeInterface; time is always 00:00:00 (start of day) - DateTimeImmutable should be used to store value
- previousTo: DateTimeInterface; time is always 23:59:59 (end of day) - DateTimeImmutable should be used to store value

`from`, `to` and `products` should be provided in constructor, but `previousFrom`& `previousTo` automatically calculated using following logic:

- if `from & to` are first and last days of year, then previous year should be set as `previousFrom & previousTo`
- if `from & to` are first and last days of month, then previous month should be set as `previousFrom & previousTo`
- in other cases:
    - `previousFrom = from - (number of days in period "from-to")`
    - `previousTo = to - (number of days in period "from-to")`

Filter

extends BasicFilter

- merchants: string[]

CurrencyRate

- targetCurrency: Currency
- baseCurrency: string
- exchangeRate: string

Service Contracts

DatabaseConnectionProviderInterface

```
public function getConnection(): Aura\Sql\ExtendedPdo
```

HttpBasicRequestStackInterface

```
public function getClient(): Psr\Http\Client\ClientInterface
```

```
public function getRequestFactory(): Psr\Http\Message\RequestFactoryInterface
```

Utils & helpers

Util

baseCurrencyFromRequestQuery

`Util::baseCurrencyFromRequestQuery(array $params): Currency`

`$params` - associative array with query parameters.

`MissingQueryParameterException` must be thrown if there is no base currency parameter in the array.

Must return instance of `Currency`.

productsFromRequestQuery

`Util::productsFromRequestQuery(array $params): Products[]`

`$params` - associative array with query parameters.

Must return `array of Product`or empty array when no products parameter provided.

DatabaseConnectionProvider

implement `DatabaseConnectionProviderInterface`

constructor: `(string $pdoDsn, string $username, string $password)`

`MissingCharsetConfigException` must be thrown in case `charset` is not defined in `$pdoDns`.

DatabaseConnectionFromPdoProvider

implements `DatabaseConnectionProviderInterface`

constructor: `(PDO $pdo)`

If `$pdo` is not an instance of `ExtendedPdo`, then it should be decorated.

HttpBasicRequestStack

implement `HttpBasicRequestStackInterface`

constructor: `(Psr\Http\Client\ClientInterface $client, Psr\Http\Message\RequestFactoryInterface`

`$requestFactory)`

Exceptions

ValueObject\AbstractInvalidValueException

constructor: `(string $value, int $code = 0, \Throwable $previous = null)`

methods:

1. `abstract protected getValueObjectClass(): ValueObjectInterface`

message: `"{{ value }}" is not a valid value for {{ short_class_name }}. Valid values: {{ valid_values }}`; `short_class_name` should be a last component (e.g. `Product`, `Currency`) from the fully-qualified class name taken from `getValueObjectClass()`; `valid_values` should be retrieved from `ValueObjectInterface`

ValueObject\InvalidProductValue

extends ValueObject\AbstractInvalidValueException

ValueObject\InvalidCurrencyValue

extends ValueObject\AbstractInvalidValueException

MissingQueryParameterException

extends \RuntimeException

constructor: `(string $missingParam, int $code = 0, \Throwable $previous = null)`

message: `Parameter "{{ missing_param }}" is missing in the query`

methods:

1. `public getMissingParam(): string`

MissingCharsetConfigException

extends \LogicException

constructor: `(string $dns, int $code = 0, \Throwable $previous = null)`

message: `Parameter "charset" is missing in the dns: "{{ dns }}"`