Test assignment

1. Design a simple database to store user messages. User table should have a minimum of info like user ID and its name. The message table should store message text, message date/time and its sender / receiver user IDs.

2. Create a PHP app which consist of:

    a. Migrations to generate programmatically the tables and relationships for the database designed above in the task #1 (users and their messages).

    b. Command line interface to generate X amount of fake/random users and Y amount of fake/random user messages. For instance if we create a CLI app using Yii2 framework, the sample call might look like this: `***php yii generate-data --user=3 --message=30***`.
        i. Note that inside the CLI input we provide a total number of messages that will be linked with the user sender and user receiver randomly. After running this CLI application, each user will get a random number of messages and the total amount of messages in the database will be equal to the amount of messages provided in the CLI (e.g. **30** like in the sample call above).
        ii. The date/time should also be a random value in the range of 5 years, e.g. the earliest message date could be 2016-06-01 00:00:00 and the latest could be 2021-05-31 23:59:59
        iii. Note that sender ID and receiver ID cannot reference the same user
        iv. Try to use the fastest and performance efficient way to generate and insert the data into the database
        v. Re-running the app will clear (flush) all the tables first and then insert the new data
        vi. You can use any framework like Laravel, Yii or any other to create the app, which means that the CLI input might look different to what we provided as a sample. E.g. it is perfectly fine if it will be `***php artisan generate:data 3 30***` or whatever else you'll come up with

    c. Using the same framework create following web services (HTTP interface):
        i. To return the total number of messages per specified period and grouped by period unit (e.g. day, month, year). So, the HTTP URL samples would look like this:
        ```
        /message/total?period_start=2016-06-01&period_end=2020-06-01&period_group_unit=year
        /message/total?period_start=2020-06-01&period_end=2021-05-31&period_group_unit=month
        /message/total?period_start=2021-05-01&period_end=2021-05-31&period_group_unit=day
        ```
        Note that the result can be multiple values grouped by the period. E.g. sample result:
        ```
        [
            {
                "period_start": "2016-06-01 00:00:00",
                "period_end": "2017-05-31 23:59:59",
                "message_number": 100
            },
            {
                "period_start": "2017-06-01 00:00:00",
                "period_end": "2018-05-31 23:59:59",
                "message_number": 1000
        ```

```
    },
    ...
]
```

ii. Most active/inactive users with a limit and order direction provided through the query params.
E.g. 5 most active users
`/message/user-activity?period_start=2016-06-01&period_end=2020-06-01&limit=5&dir=desc`
E.g. 10 most inactive users
`/message/user-activity?period_start=2020-06-01&period_end=2020-06-01&limit=10&dir=asc`

d. Optionally cover the web service(s) with automation tests (unit or functional, your own preference).

3. Create a simple HTML page to present the total number of messages queried by specified period and grouped by specified period unit (year, month or day). To query the data use the same HTTP interface for total messages you created in the task **2.c**. Allow to input the period start, period end and period group unit values. Present the web service response numbers using any chart type (bar, line) with the help of any chart library you like, e.g. Chart.js. Sample chart:
https://www.chartjs.org/docs/latest/samples/line/line.html

4. Provide a short instruction on how to run your app(s)