This is automatically generated report from HTML page. It is recommended to see the report on the **About** page of the application.

## Briefing

This web application is a time-tracking app, which tracks the time spent on users' tasks, saves them, and provides some statistics and analysis of users' productivity.

The main inspiration for this app is the Toggl Track app, which is a very popular time-tracking application.

Most important features of this app are:

- Time tracking - user can create a task and run it with a timer until it is stopped. Also, the user can change the name of the task while it is running just by typing a new name and pressing enter key. What is most important - you can close the browser and open it again - and time entry still will be running. You can stop it in one browser, and the other browser will automatically react and will stop it too. Same with starting a new time entry - the timer is synchronized with the server.
- Time entries table - user can see time entries in the table, and manipulate with them - filtering time entries with the date and time ranges, and also sorting columns. Also, user can create new time entries and delete existing ones. Input validation is also supported.
- Visualization - on the analysis page user can see some useful charts which provide some statistics about users' productivity.
- In overall, this app is a web application that uses MVC architecture. Also, docker support is provided and app can be easily deployable.

## Realization

This app is more about used technologies and frameworks than about algorithms. So there won't be complex algorithms or math.

**Architecture:**

As it was mentioned - this app has MVC architecture. But because the app is small, backend and frontend are run on the same server. Therefore, frontend part - views, which render HTML pages using templates, use directly same services that the backend uses. But still, the user's browser is the client, which runs javascript code, which calls the backend's API and requests data.

**Used frameworks:**

- **Backend side:**
    - FastAPI is used for creating API and views. Application is run by Uvicorn - ASGI server for python.
    - SQLAlchemy is used for ORM and communicating with the PostgreSQL database, which is run in a docker container.
    - Pydantic is used for data validation and creating schemes (aka DTO - data transfer objects). Also Pydantic provides configuration for the whole application - special settings class.
    - Pandas is used for time entries statistics service, which allows to easily manipulate with data.
    - Plotly is used in combination with pandas to visualize users productivity in charts.
- **Frontend side:**
    - HyperApp is a tiny javascript framework used for timer feature.
    - Jinja 2 template language is used to render HTML pages with provided data.
    - Bootstrap CSS framework is used for a nice UI.
    - It is worth to mention, than HyperApp and Bootstrap are downloaded by CDN links - their libraries are download directly to the users' browser.

**Timer:**

The timer feature is created using the HyperApp framework. This framework uses states. Each event or user action creates a new state. There is a timer that every half of second updates the current timer count. It is worth mentioning, that it doesn't add seconds. In the database, the time entry entity stores the start DateTime and end DateTime. Until the end DateTime is null, time entry is still running, and HyperApp just calculates a difference between the current time now and the start time, which is stored in the database. Therefore there is impossible any significant deviation from the real timer count.

The main problem was to ensure syncronized behaivior. For that WebSockets framework was used. It connects user to the server and the server periodically sends to the client the state of the currently running entry. If it is still running or not. That ensures that timer is syncronized.

For starting time entry or updating its name, HyperApp just uses API calls to backend.

**Time entries table:**

Just on the same page as the timer, the time entries table allows the user to manipulate the data. In backend, there is a time entries statistics service, which uses the Pandas framework. Pandas dataframe is loaded directly from the database table.

Creating and deleting operations use the backend API, called by JavaScript code in the user's browser.

**Visualization**

Plotly framework is used to create charts and plots directly from pandas dataframe. Then they are converted to HTML and rendered by Jinja 2 template language.

**API:**

The API of the application is automatically generated by FastAPI - and is located on the /docs page. It provides not only information about API but also allows to make those API calls with provided data.

## Results

**Timer:**

The timer is working and does what it should be doing. It tracks time, saves it to database, it is synchronized with the server. It works very similar like Toggl Track timer.

But of course, there are still problems with it - mainly with milliseconds issue that you don't see as user - sometimes it stores time with one second less than it should be.

What could be improved? Certainly the stability, and also allowing to update not only the title of the running task but the start date-time too, so the entry can be moved, extended, or shrinked.

**Time entries table:**

It uses Pandas and allows user to manipulate with the data. But still, there is a problem, it doesn't have update operations - only create, read and delete. Next time that can be improved.

**Visualization:**

Charts and plots are a very good feature. They really provide useful information for user, and even allows user to interact with the charts.

But there could be more charts with much more advanced statistics. This is what could be improved.

**Overall:**

This application uses a lot of different technologies and frameworks. It provides all basic features for the tracking app.

## References

- FastAPI
- SQLAlchemy
- Pydantic
- Bootstrap
- HyperApp
- Jinja 2
- Pandas
- Plotly
- Toggl Track