Reet Chahil

rc17yg

6465314

COSC 3P95

A2 – Q2

OpenTelemetry Report

## System Overview

The system I instrumented is a to-do app made with Python and Django. It was pulled at the following link:

<https://github.com/rtzll/django-todolist>

It allows you to register an account so that all your tasks are saved under your account. When you start the app, you can log in to access your existing tasks. You can create multiple lists of tasks which are visible in an overview page. When you click on a list or create a new list, it opens a view of that list and its existing tasks. You can add more tasks to the list. You can also check off tasks in the list.

The following is the project structure.

* Accounts – contains login and registration functionality.
* Lists – contains models and functionalities for to-do lists.
* Todolist – contains settings.py where instrumentation is initialized.

## Description of Instrumentation

I used Open Telemetry’s tracer to create traces through key parts of the system. I started by instrumenting the login and registration features. At the start of each function that made a request, I created a span and gave it a name that represented the purpose of the function. I added an event to the span at each point the logic flow changed. For example, I added an event when the login form was validated, when login was successful, and when login failed. This provided logging information in the traces. I followed the same principle when instrumenting the lists and behaviours such as creating lists and to-do items.

I added metrics using Open Telemetry’s metrics. I initialized the meter in the settings.py file. For each function where a request was processed, I added a counter so I could count how many times each request was made. Each counter was given a name to easily identify what function it was tracking. At set intervals the metrics were printed to the console.

## Description of Data Visualization

The tracing data was exported with the Jaeger exporter. I ran a docker container with Jaeger to visualize the tracing data I collected.

Below is a screenshot of some collected traces.

Graphical user interface

Description automatically generated

Figure : List of Traces in Jaeger

Graphical user interface, application

Description automatically generated

Figure : Selected span from Jaeger Dashboard; User adds a to-do.

otel.library.name has the name of the module in the to-do app where the span is located.

There are two logs due to adding events to the span to show when the form was validated and when the to-do item was added.

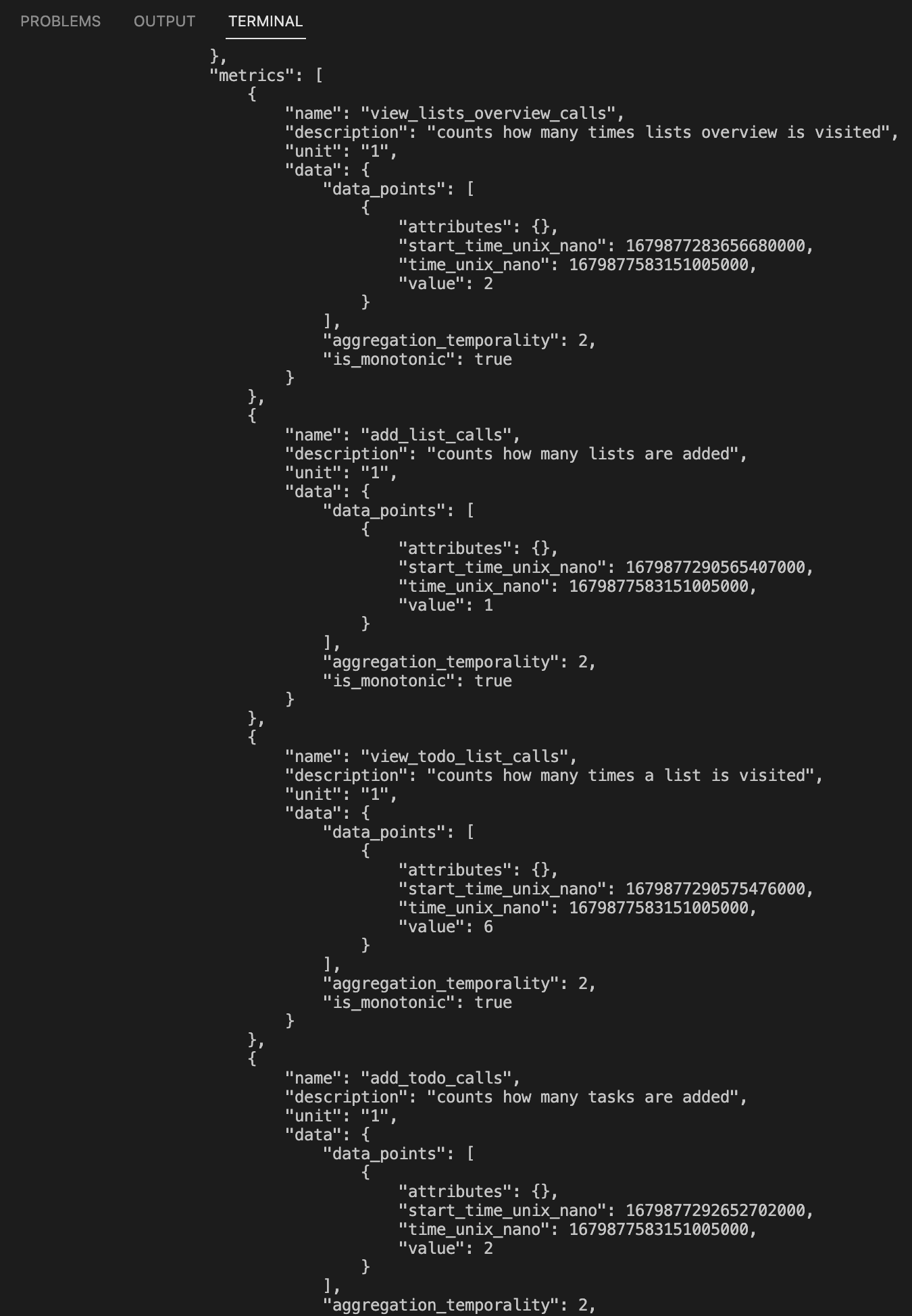
This span took 4.23ms to complete. The form was valid in 146µs, and the to-do item was added in 4.04ms.

Below is the snippet of code that collected this span.

Text

Description automatically generated

I added metrics to count how many times a request was made at set intervals. This can be seen on lines 35 – 37 where I incremented the counter every time the function was called. Below is the output for the metrics.



This is a snippet of the output of the metrics to the console. Each object in the metrics array represents a metric collected. The name and description show what action is being counted.

We can see that a request to view a list was performed 6 times.

A request to add a to-do was made 2 times.