# WovenLight Development Exercise

## Introduction

For this exercise, we would like you to write a web service using Python.

The service will be used to schedule calls to Transport for London's (TFL) API. TFL provide a set of clean REST APIs that you can use to get data about transport systems in London. For example, the status of a tube line or where Santander bicycles are.

The APIs are well documented and can be found here. For the purpose of this exercise we suggest you use the *Line API* (`https://api.tfl.gov.uk/Line/*`, docs) which provides information on current line disruptions. You will not need an App Key.

An example curl to get disruption information on the Bakerloo and Jubilee lines:

```
$ curl -X GET https://api.tfl.gov.uk/Line/bakerloo,jubilee/Disruption
```

The goal of this exercise is to write a RESTful service for scheduling tasks to run at a specified point in time, that make requests to the Line Disruption API and store the results.

For example, while at work, we might want to schedule a task to run at 1645 (and not before) to get the current disruptions on the Victoria line so we can plan our commute home.

You can assume a single user will use the API.

## Functionality

The service you write should support the following actions:

- Adding tasks with parameters: `schedule_time` and `lines`
- Getting all tasks, or a specific task (by task `id`), with results if they've been run.
- Deleting a task using a task `id`
- Updating `schedule_time` or `lines` of an existing task by task `id`, if it hasn't been scheduled yet.

The format for schedule times will be: '%Y-%m-%dT%H:%M:%S'. You can assume the time is in local timezone (of requester and server); no need to work with UTC, etc.

If an empty `schedule_time` is sent, schedule a task to run immediately.

For simplicity, you should accept TFL line `ids` for line names, for example: `victoria`; `central`; `bakerloo` --- not line names like Central or Victoria. There are 11 lines. You will need to lookup all the line `ids`.

You should assume data is sent in JSON format.

You should include tests for the API.

## Usage

### Basic commands

With the server running on `localhost` and port `5555`, using cURL, we would create a task with a POST request:

```
$ curl -X POST -H 'Content-Type: application/json' -d '{"scheduler_time" :
"2021-11-12T17:00:00", "lines":"victoria"}' http://localhost:5555/tasks
```

We might want information on multiple lines, for example, the Victoria and Central lines:

```
$ curl -X POST -H 'Content-Type: application/json' -d '{"scheduler_time" :
"2021-11-12T17:00:00", "lines":"victoria,central"}'
http://localhost:5555/tasks
```

To get information on all tasks, we would send a GET request:

```
$ curl -X GET http://localhost:5555/tasks
```

To get the results from a scheduled task with ID `id`:

```
$ curl -X GET http://localhost:5555/tasks/<id>
```

### Advanced commands

To update the scheduled time of a scheduled task with ID `id`:

```
$ curl -X PATCH -H 'Content-Type: application/json' -d '{"scheduler_time"
: "2021-11-12T17:00:00"}' http://localhost:5555/tasks/<id>
```

## Tech stack

At WovenLight, we develop using Python `3.x`.

To keep things simple, you could use the `Flask` web framework. If you're more familiar with other frameworks, such as `FastAPI` or `aiohttp`, that's fine too. Inspiration could be drawn from starters like the FastAPI Postgres Template.

To make requests to the TFL API, we recommend using the `requests` library - it's great.

For state, you could use a database like `sqlite` or `PostgreSQL`.

For scheduling, to keep it simple, you could use a library like `apscheduler`.

For the testing framework, why not use `unittest` or `pytest`?

We package and ship our services using Docker. We'd like you to run your services, i.e. API server and database (if using one), using one or multiple Docker containers. Make sure to include all Docker-related files with the source code so we can replicate your setup.

You must include a `README.md` that outlines your API, shows how to run the server and tests, and a list of limitations. Also include how much time you spent on the exercise (i.e. hours) and if you had to learn any new technologies.

## Deliverable

When you're ready, zip up the repository and send it over.

We will take a look at it, and if we decide to proceed, we'll get you to take us through your approach. We'll also discuss implications for multiple users, security, scalability, performance, and fault tolerance.

Note that your submission will be judged based on what you prioritized, not on the total number of features added.

## Finally

We hope these instructions are clear. If not, please don't hesistate to ask questions.

Finally, we hope you find this task both challenging and fun!