Technical Task - Backend Engineer

End-users of this product will be embedding an image into their pages to track visits to their sites:

```
<img src="http://pixel-service/track" alt="SuperTracker Pixel" />
```

The solution captures requests to that image and stores visitors asynchronously in the plain text log for further analysis. The solution consists of two services (projects): one collects the data, other handles writing to the file.

Pixel Service

Should be ASP.NET Core 6+ **Minimal API**.

This API has only one endpoint `GET /track` that returns a transparent **1-pixel image in GIF format** and collects the following information:

1. `Referrer` header
2. `User-Agent` header
3. Visitor IP address

After the information has been collected, it sends it to the **Storage Service**.

Storage Service

Receives event and stores it in the **append-only file**. The path to the file is defined via `appsettings.json` with the defaults to `/tmp/visits.log`.

The format of the file is “date-time of the visit in ISO 8601 format in UTC | referrer | user-agent | ip“.

The IP address is the only mandatory value. The rest can be substituted with `null` when empty.

Example:

```
2022-12-19T14:16:49.9605280Z|https://google.com|SomeUserAgent 1.2.3|192.168.1.1
2022-12-19T14:17:49.9605280Z|https://bing.com|AnotherUserAgent 4.5.6|10.0.0.1
2022-12-19T14:16:49.9605280Z|null|null|8.8.8.8
...
```

Constraints

1. There could be **multiple instances of Pixel Service** sending events simultaneously, but the **Storage Service is always one**. No need to handle any concurrency issues other than writing to the file. The order of writes does not matter.
2. Pixel Service runs without any HTTP balancers / WAFs in front of it. There’s no need to handle proxied headers.
3. Chose the communication protocol between Pixel and Store services that you think makes sense for that task. Pixel and Store services might be running inside different networks without direct access to each other, but both have access to any required third-party services. The connection string or any connection parameters will be passed via the `appsettings.json` or get overridden by the environment variables.
4. Keep things as simply as possible. If something is not clear from the task description, make an assumption and leave a comment in your code.
5. **Extra task**: write unit tests for both services.
6. **Extra task**: write separate Dockerfile for two services with build and runtime split into two stages.

Deliverables

Public GitHub repository with the test task result.

Constraints marked as extra tasks are optional.

We are not going to run or even build the code and do the E2E testing but only look into the repository and the logic of your solution.