



Product Engineering Task

☰ Tags

Engineering



Congratulations on being selected to progress to the next stage of the process and to complete our technical task. The purpose of this task is to assess the quality of the code that you write, your approach to problem solving, and your ability to reason about the approaches and practices you have implemented.

Context

Orbital Copilot is our newest product; an AI assistant for Real Estate lawyers. Users can ask Orbital Copilot questions about a range of legal documents such as leases. As well as singular questions (e.g. "What is the rent?"), users are also able to ask Orbital Copilot to produce different types of reports (e.g. "Produce a Short Lease Report") which cover many aspects of the documents.

Orbital Copilot works on a consumption basis. Customers will be charged depending on how many "credits" they have consumed. We currently have raw data accessible via APIs, which are explained in the next section, and we need to combine these data sources in order to calculate how many credits are being used. We would then like to display this data to users in a dashboard UI.

The Data

You have access to two API endpoints (imagine that these are other services in our stack) which you will need to use to complete this task. The endpoints are both accessed with HTTP GET requests and do not require authentication.

1. Get raw message data for the current billing period

<https://owpublic.blob.core.windows.net/tech-task/messages/current-period>

👉 This endpoint returns all of the messages sent to Orbital Copilot in the current billing period. Each message will have a numerical ID, a timestamp in ISO 8601 format, the message text that was sent, and optionally, a report ID, which means that Orbital Copilot generated a report in response to this message.

2. Get the name and cost of a report by ID


<https://owpublic.blob.core.windows.net/tech-task/reports/:id>

👉 This endpoint takes a report ID in the URL and returns the name and credit cost of that report

The Challenge

Backend

First, we would like you to build a simple Python API with a single endpoint accessible by making a GET request to /usage that returns the usage data for the current billing period. The endpoint should return a JSON response with the following format.

 **NOTE:** It is extremely important that the API response follows this format exactly - it will be consumed by multiple teams who have agreed to this contract!

```
{
  usage: [
    {
      message_id: number
      timestamp: string
      report_name?: string
      credits_used: number
    }
  ]
}
```

The `usage` array should contain an object for each message returned by the endpoint provided, with its ID and timestamp. It should also include the Report Name for each message if there is one, otherwise this field should be omitted. It should also include the number of credits consumed by that message.

The number of credits consumed by each message is calculated as follows:

- When a message has a `report_id` associated with it, ignore the message text, and the fixed number of credits used to generate that report can be found by querying the `reports/:id` endpoint for that `report_id`.
 - If the `reports/:id` endpoint returns a `HTTP 404` status code for a given `report_id` you must fall back to the calculation method outlined below.

- For messages without a valid `report_id`, the number of credits consumed is based on the message `text` and following rules apply. (**Note:** A "word" is defined as any continual sequence of letters, plus ' and -)
 - **Base Cost:** Every message has a base cost of 1 credit.
 - **Character Count:** Add 0.05 credits for each character in the message.
 - **Word Length Multipliers:**
 - For words of 1-3 characters: Add 0.1 credits per word.
 - For words of 4-7 characters: Add 0.2 credits per word.
 - For words of 8+ characters: Add 0.3 credits per word.
 - **Third Vowels:** If any third (i.e. 3rd, 6th, 9th) character is an uppercase or lowercase vowel (a, e, i, o, u) add 0.3 credits for each occurrence.
 - **Length Penalty:** If the message length exceeds 100 characters, add a penalty of 5 credits.
 - **Unique Word Bonus:** If all words in the message are unique (case-sensitive), subtract 2 credits from the total cost (remember the minimum cost should still be 1 credit).
 - **Palindromes:** If the entire message is a palindrome (that is to say, after converting all uppercase letters into lowercase letters and removing all non-alphanumeric characters, it reads the same forward and backward), double the total cost after all other rules have been applied.

Frontend

Now that we have exposed all of the usage data that we need exposed by a single API, we would like to display this in a dashboard built with React.

The dashboard should query the API you have built and display the data in a table with the same headers and data format as below:

Message ID	Timestamp	Report Name	Credits Used
1000	29-04-2024 02:08	Tenant Obligations Report	79.00
1001	29-04-2024 03:25		5.20
...

As you can see, **Timestamp** should be formatted in a certain way. If the message was not a valid report, the **Report Name** column should be empty. The **Credits Used** column should always be shown to 2 decimal places.

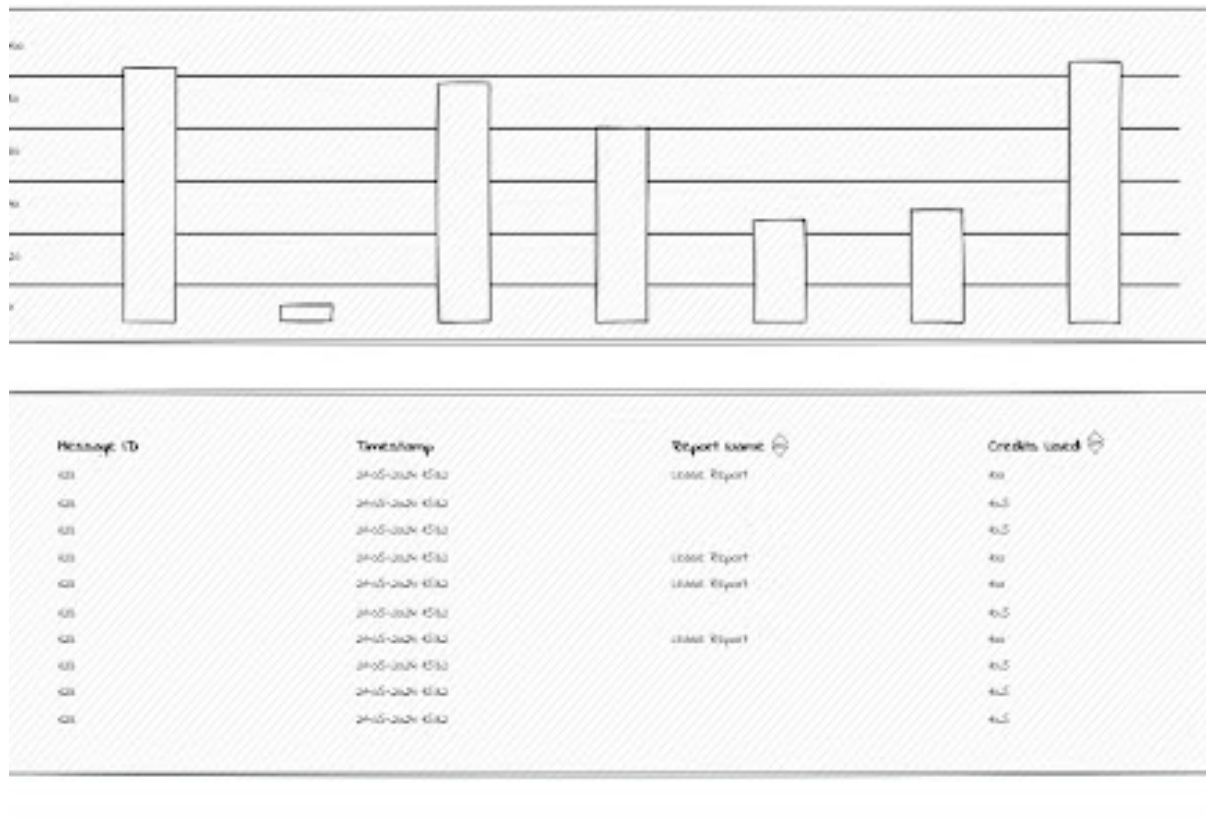
The **Report Name** and **Credits Used** columns should be sortable. First click sorts it ascending, second click sorts it descending and a third click returns to its original sort order. It should be possible to have sorts applied to both columns at the same time.

The user should be able to share the URL to colleagues and see the same results, with the same sorting applied.

Above the table, you should display a simple bar chart with date on the x-axis and credits on the y-axis. Display a bar for each date from the earliest date in the data to the latest, which represents the total number of credits consumed on that day.

Note from designer: "The final designs aren't ready yet - sorry! But please use the wireframe below to guide you. Feel free to add your own creative flair!"

Usage Dashboard



Guidelines

- Please provide a README that contains instructions for how we can run your solution, any decisions you made that you feel are worth highlighting and concessions you had to make due to the time constraints, as well as anything else you think we should be aware of when reviewing your code.
- It is critical that the usage data we provide to our customers is accurate, so please consider how you can be confident that your solution produces correct results.
- We are just as interested in your technical approach as we are in completeness, so please document your implementation decisions, for example in comments.
- If you are asked to attend a face-to-face interview, there will be an opportunity to go into more detail so please aim to spend a maximum of 3-4 hours on the exercise.



IMPORTANT NOTE: You can use any tools you prefer to help you complete the task requirements above, **BUT** if you are progressed to an interview, part of the interview will include a collaborative session where we will ask you to walk us through your take-home task whilst asking questions about why you made certain decisions. This is to ensure that you have a full understanding of the solution you have created so please take this into