Aether Take-Home, Founding Engineer

Context:

This take-home is extensive. It's up to you how much of the assignment you want to do.

The primary intention of this take-home is to see:

(1) your competency with Python & Django,
(2) your ability to quickly model an app using an MVT architecture
(3) Ability to handle API calls and also set up APIs/Webhooks
(4) Ability to work with basic mathematical concepts related to energy
(5) Overall system design understanding & transaltion

In this exercise, you will need to use the API Keys provided by openEI: https://openei.org/services/ , it's free. Once you have the key, we will work with this API from openEI (Utility rate database, US). https://openei.org/services/doc/rest/util_rates/?version=3

You do not need to host this service online, but during the review, I will want to see in your Django admin or SQLite env, the data as we are going through the app.

Technical Spec: "Part one"

Front-end:

It's up to you on how you build the FE layout, but do use a basic react component library if possible. You don't need to create a separate FE but instead keep it within the Django app templates.

In the UI, allow for the user to:

(1) input a US address.
(2) Enter a kWh Consumption Value (min 1000: max 10000)
(3) Set a percentage escalator from 4% to 10%

We want to take this address input and pass it to openEI in order to retrieve the "utility tariff" for the given address. Precisely, we want to display back to the user:

(1) the average ¢/kWh
(2) the most likely utility tariff name (discussed below)
(3) the full list of utility tariffs with start dates after the last day of year 2021.

(4) Cost of their utility for the first year

(3) Can be a dropdown list with data or a table, it's up to you.

Now, on the initial user input, we want the response to display (2) as the tariff name and its price. But, if the user was to select a tariff from the dropdown then display that tariff's average ¢/kWh. It would be ideal from a UI/UX perspective to know which tariff is currently selected.

We do not want the full "detail" on the API response back from openEI; *please explain to me why not during our call.*

Because the user has not given us their utility details we will need to presume for them a starting utility tariff name using openEI. For our system design, we will call this the "most likely" tariff. Most Likely tariff is the tariff when making the API call to openEI with these params:

- Approved == true
- Is_default == true

Calculating cent/kWh:

Two approaches:

Hard & Easy, do easy 1st and see if it's feasible to do hard.

If it's simply too much time note down why and move on. Do not spend more than 1 hour here. I want to know what was hard about the hard approach.

You will need a load curve to calculate it, use this one:
https://www.researchgate.net/figure/Daily-load-curve-of-residential-consumer_fig2_331013145

Every response from openEI gives you a pricing matrix. Some are extremely simple with only one period and one price and some have multiple pricing periods.

This is the pricing period (at the top) and pricing matrix (at the bottom)

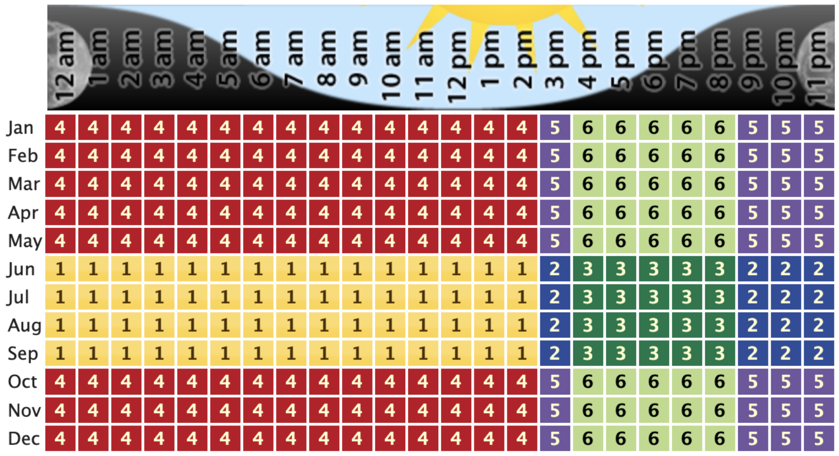
Tiered Energy Usage Charge Structure

Period	Tier	Max Usage ?	Max Usage Units ?	Rate $/kWh ?	Adjustments $/kWh ?	Sell $/kWh ?
1	1		kWh	0.27818		
2	1		kWh	0.48019		
3	1		kWh	0.59068		
4	1		kWh	0.27818		
5	1		kWh	0.44687		
6	1		kWh	0.46357		

Fuel Adjustments Monthly ($/kWh)

Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec

Weekday Schedule



	12 am	1 am	2 am	3 am	4 am	5 am	6 am	7 am	8 am	9 am	10 am	11 am	12 pm	1 pm	2 pm	3 pm	4 pm	5 pm	6 pm	7 pm	8 pm	9 pm	10 pm	11 pm
Jan	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	5	6	6	6	6	6	5	5	5
Feb	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	5	6	6	6	6	6	5	5	5
Mar	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	5	6	6	6	6	6	5	5	5
Apr	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	5	6	6	6	6	6	5	5	5
May	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	5	6	6	6	6	6	5	5	5
Jun	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2	3	3	3	3	3	2	2	2
Jul	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2	3	3	3	3	3	2	2	2
Aug	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2	3	3	3	3	3	2	2	2
Sep	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2	3	3	3	3	3	2	2	2
Oct	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	5	6	6	6	6	6	5	5	5
Nov	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	5	6	6	6	6	6	5	5	5
Dec	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	5	6	6	6	6	6	5	5	5

You want to use the load curve from above to compute against only the weekday schedule at the given hourly interval. The aim is to figure out your Rate c/kWh using the consumption kWh the user inputted into the UI.

Doing this approach will also give you your daily cost of electricity per day.

If the above is hard, or the technical spec is not clear, not this down and move on.

Easy approach:

In the pricing period for the given tariff simply average out all periods (if more than 1) and use this as the cent/kWh.

Graph Output:

Once you are able to grab the tariff's price you have the information needed to make a utility cost graph. This is, for a given year interval X it will have a Y output of yearly utility price.

Make sure this graph accounts for 20 years of utility cost given your users inputs.

<u>What I'm looking at:</u>

- Do you see any flaws with the above implementation?
- I want to know where you struggled so we can discuss and see how we would have approached it if we were working together on this in the production app.

<u>Why are we doing this?</u>

As the role expands, you will need to be comfortable dealing with external systems on getting "atomic levels" of data and running computations on it from my or Shelby's technical specs. Beyond Utility we will deal with electrical data and at the core, you will be more effective if understand the computational data points you are working with.

Part Two: You will most likely do this before Part one. Models.

Within the Django app, make a model that has a User (should be there by default) & Project object.

Every time the user makes a new address input in the UI we want this Django Project Object to capture it as a new object record along with the relevant details we display to the user and capture the latest utility rate selected.

I also want to see another Object with the name, ProposalUtility, which captures for the Project object in a one-to-one relationship the utility tariff response from openEI for the current selected utility. For simplicity's sake only have 5 fields here. Most important is the ID (from openEI), tariff name, and pricing matrix.

Set up a basic Django user auth so we can test this between one or two users.

Part three: Basic webhooks & APIs

Set up an extremely basic API using Django Views which allows us to make POST calls to the project object via an endpoint. Anytime a POST call is made, let's capture using a webhook handler such as https://webhook.site/#!/abad1e6f-3d72-4ead-9f06-e9a519d8b4c9 the event notification.

Webhooks & APIs will be a key feature of a product from the start. The large majority of companies using us as an end-to-end solar tool will use us to power their entire workflow and will want data to enter into our app or exit. Try to think why webhooks become an important asset for us.

Try not to exceed more than 4-5 hours on this exercise.