Dell Software Engineering Challenge

Desired Output:
A collection of files, with all the code necessary to solve the 3 challenges proposed here and answer the questions. The code should be properly commented inline.

Topics Covered:

Languages: HTML, CSS, JavaScript, Python

Technologies: Web Scraping, HTTP Requests, Async/Await, JSON, REST APIs, Frontend, Backend.

Exercise 1:

Languages: HTML, CSS, JavaScript

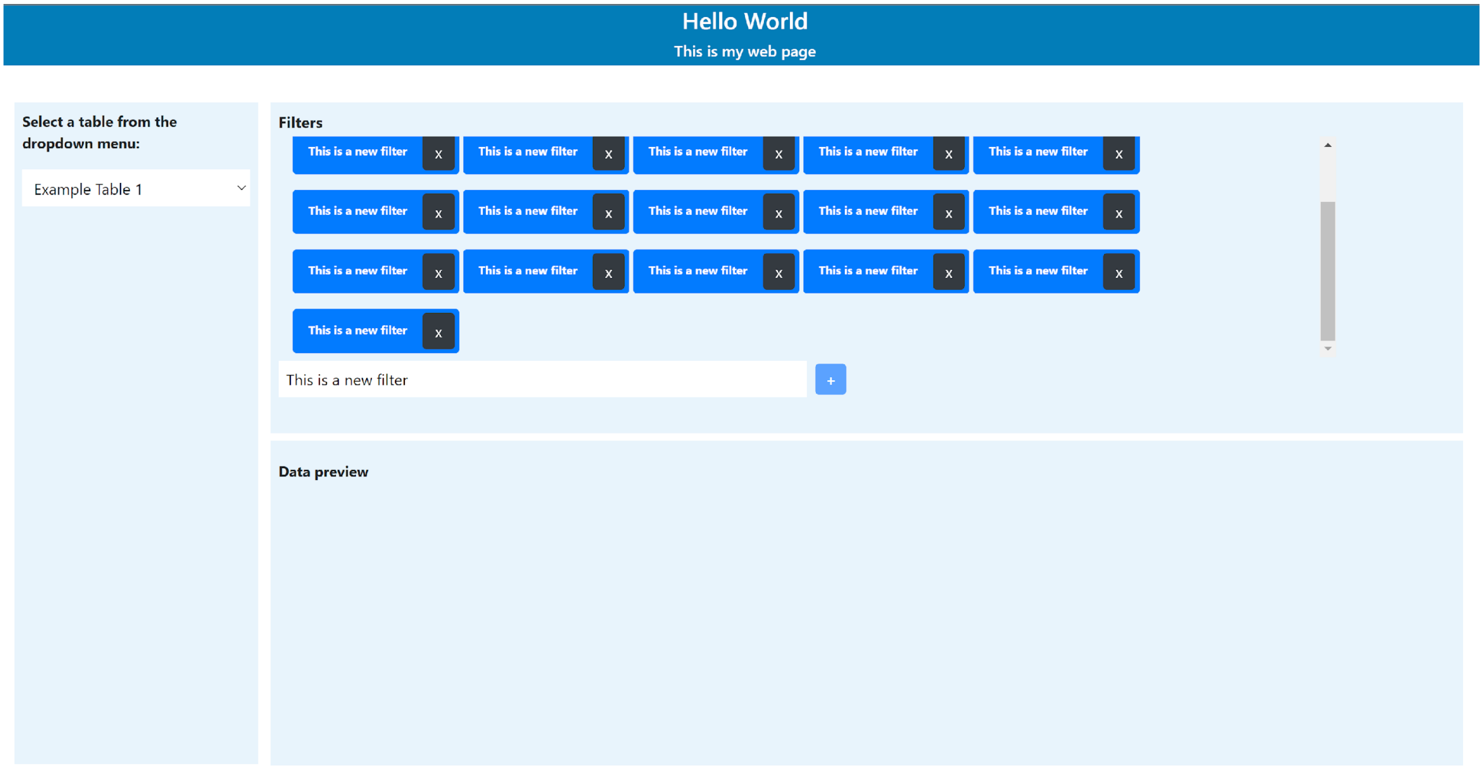
Create a Web Page that looks as similar as possible to the screenshot below.

It should be possible to add a new Filter element in the "Filters" section, using the text input field and the + button.

When the filter elements are too many, a scrollbar will appear on the right side.

You can also delete each Filter element by clicking the X button next to them.

You can use HTML, CSS, vanilla JavaScript, jQuery, Bootstrap or any other styling library and framework.



Hello World

This is my web page

Select a table from the dropdown menu:

Example Table 1

Filters

This is a new filter

Data preview

Questions:

1. How would you deploy this web page so that users can access it online? Talk about the infrastructure, the resources, and the technologies needed.

2. How would you modify the code in order to populate the dropdown menu with values coming from a database? Talk about possible database solutions, and how the backend can interact with the frontend.

Exercise 2:

Language: Python

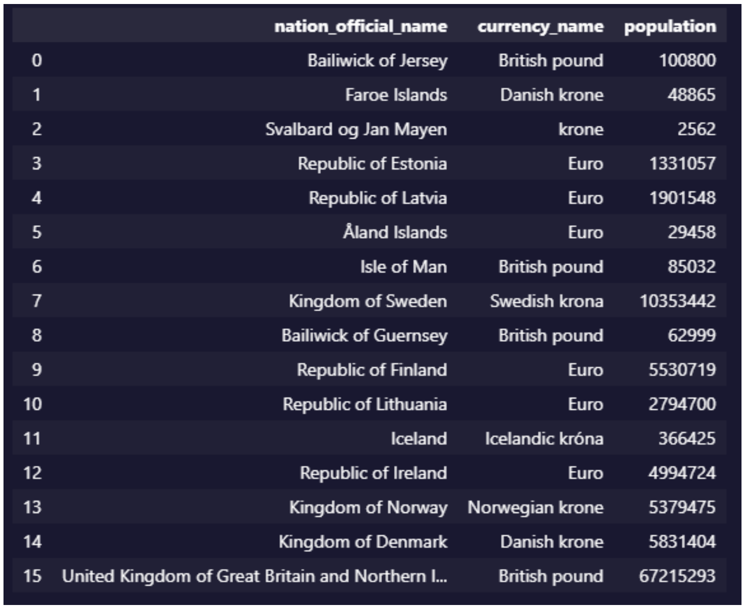
Use the following public REST API.
https://restcountries.com/

Get data only for Northern European countries, and filter only for the following fields:
name
currencies
population

The API call should be asynchronous and encapsulated in a function.

Load the JSON response into a dictionary and then turn it into a single index Pandas dataframe.
The columns should be “nation_official_name”, “currency_name” and “population”.

The dataframe should look like this:



	nation_official_name	currency_name	population
0	Bailiwick of Jersey	British pound	100800
1	Faroe Islands	Danish krone	48865
2	Svalbard og Jan Mayen	krone	2562
3	Republic of Estonia	Euro	1331057
4	Republic of Latvia	Euro	1901548
5	Åland Islands	Euro	29458
6	Isle of Man	British pound	85032
7	Kingdom of Sweden	Swedish krona	10353442
8	Bailiwick of Guernsey	British pound	62999
9	Republic of Finland	Euro	5530719
10	Republic of Lithuania	Euro	2794700
11	Iceland	Icelandic króna	366425
12	Republic of Ireland	Euro	4994724
13	Kingdom of Norway	Norwegian krone	5379475
14	Kingdom of Denmark	Danish krone	5831404
15	United Kingdom of Great Britain and Northern I...	British pound	67215293

then connect to a hypothetical Postgres Database and load the dataframe to a new table, in REPLACE mode.

Questions:

1. If you didn't know the structure of the JSON, and there was an arbitrary level of nesting of arrays and dictionaries, how would you need to change the code to dynamically unnest the data into a single-indexed dataframe?
2. If you had to scale this application to read and load data for hundreds of countries and refresh the database every few minutes, what strategies could be used in terms of coding patterns, technologies, resources and infrastructure?

Exercise 3:

Language: Python

Scrape the S&P 500 companies table from the following Wikipedia page:
https://en.wikipedia.org/wiki/List_of_S%26P_500_companies

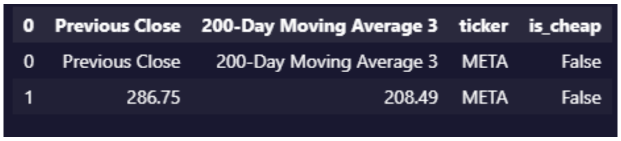
Save the companies ticker symbols into a list. Cut the list to take only the first 50 elements.

For each ticker symbol in the list, call the following API In order to get the Previous Close value for each company:
https://finance.yahoo.com/quote/AAPL?p=AAPLtsrc=fin-srch
Save this value and the ticker symbol in a Pandas dataframe.

For each ticker symbol also call the following API endpoint in order to get the 200-Day Moving Average value:
https://finance.yahoo.com/quote/AAPL/key-statistics?p=AAPL
Save this value in a new column of the same dataframe.

Compute a new column in the dataframe called "is_cheap" with a Boolean value which is True if the Previous Close is lower than the 200-Day Moving Average and False otherwise.

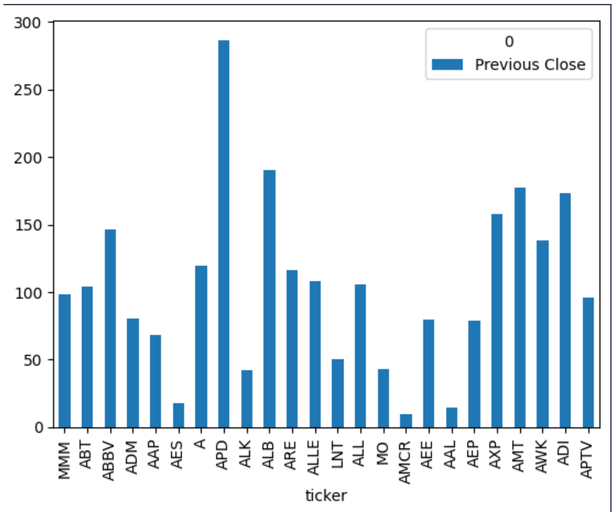
Example:



0	Previous Close	200-Day Moving Average 3	ticker	is_cheap
0	Previous Close	200-Day Moving Average 3	META	False
1	286.75	208.49	META	False

Concatenate all dataframes for all ticker symbols in one.

Display the dataframe on a plot only for the companies where is_cheap = True.
On the X axis should be the Ticker symbol and on the Y axis the Previous Close value.



Questions:

1. If the Wikipedia table was lazy loaded, and only appeared after a few seconds from opening the page, what libraries and strategies could you adopt to get the data?
2. If you had to run this script for thousands of companies instead of 500, what kind of patterns, libraries and/or optimization techniques could you use to keep the process efficient?