JavaScript Challenge

Duration: 4 hours
We use the exact same test for junior and senior engineers, and we clearly **do not** expect all candidates to complete the entire challenge.
The goal is to spend 4 hours on the test and send what you have done in these 4 hours.

Steps are given in order of importance - please try to complete Step 1 as well as possible before starting Step 2.
Please consider that every aspect of the test is judged, from Git history to READMEs, from code design and comments to the actual working result.
The only aspect that will not be judged is the style (as in CSS) of your submission. Focus on JavaScript/React architecture and clean code, not on the styling of your app.
Make sure your submission is well-presented, with clear instructions, and a clean git history.

Source-code management and hosting:
You should use **Git** to manage your code using **iterative commits**.

Each step is a new branch. At the end of each step, create a Pull Request for your branch to explain what's been done in the step.

Please try to submit a **clean commit history**, as you will be judged on the **presentation of your code**.

Submitting your test:
Once you have completed the test, send us an email at jobs-js@botify.com containing either:
- A link to the Github repository
- Or a git bundle containing your code

If you're hosting your code on GitHub, please invite the following users into your repository:
- Ghislain Le Sergeant - Lead JS Engineer // Github: @g-Ish
- Renan Gehan - Lead JS Engineer // Github: @rgehan
- Timothée Barbot - Lead JS Engineer // Github: @btimo

We very much thank you for the time you spend on this challenge.

botify

JavaScript Challenge

Goal:
The goal is to create an app that shows visualizations of data (almost shippable).
The data displayed will be fetched from NASA's NeoWs (Near Earth Object Web Service).
The test consists of the following steps.

- [Step 1] Create a React application
- [Step 1] Fetch data from NASA's API
- [Step 1] Display that data in a chart using Google Charts
- [Step 2] Implement a user input used to filter the data in your chart
- [Step 3] Implement a Table view of the data with a view switcher
    - [Step BONUS] Implement a "CSV Download" button
    - [Step BONUS] Reimplement your components in Hooks vs Classes

Target:
Evergreen browsers only: you will not be judged on the compatibility of your code for older browsers. Consider that this test will be graded on the latest version of the Chrome browser.

Constraints:
You will use the following libraries, which are parts of the JS stack used at Botify:
- Views and Components: React
- Charts: Google Charts

Tips:
- You can (and probably should) use a **starter kit to bootstrap your project.** create-react-app is a good one to start from.
- Split your code (actions, views, components, utils..) into different files and folders.
- Feel free to use any additional libraries. We are always interested in new ideas.
- As previously mentioned: do not spend too much time on the styling (CSS) for your project, it will not be held against you.

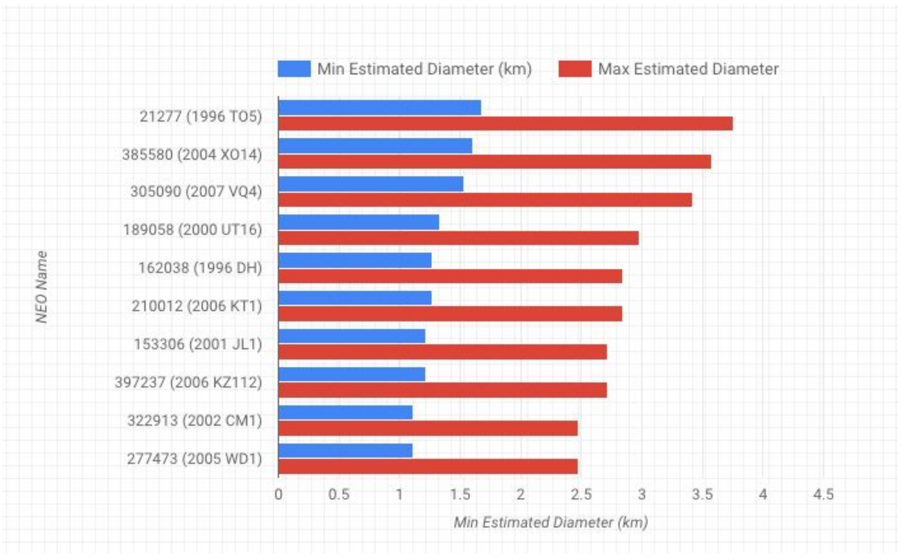
Step 1: Data Visualization

Create a Single Page Application with only one view that shows a visualization of information regarding Near-Earth-Objects travelling around the earth. You will use the Data returned by the following API call: https://api.nasa.gov/neo/rest/v1/neo/browse?api_key=DEMO_KEY

Display a non-Stacked Bar Chart, with 2 Bars for each Near Earth Object from the API representing their min and max estimated diameter, sorted by average estimated diameter descending.

You must handle all of the different possible states and cases during the data fetching.

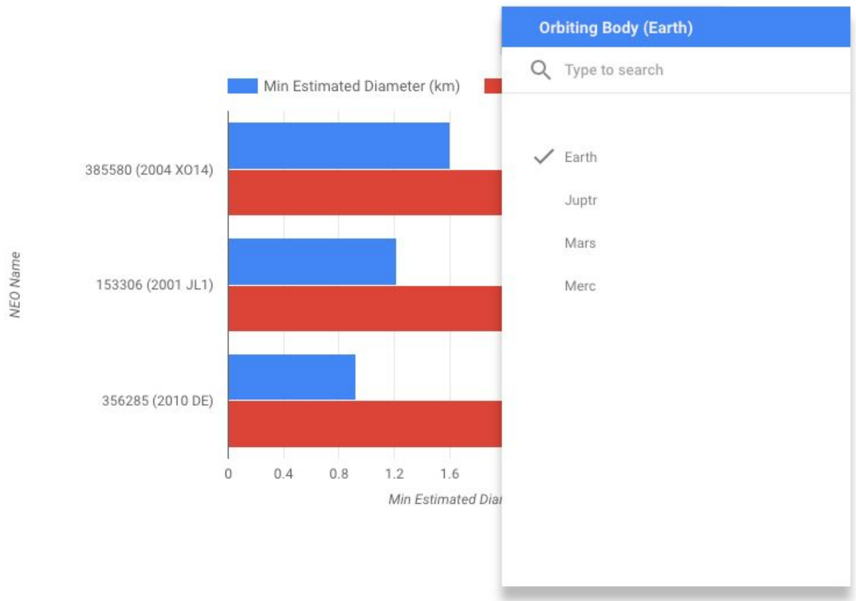
Your chart should look like the following visualization:



Step 2 - Data Filtering

In a new branch, add a dropdown that allows users to change the shown data on your visualization app. This modifier will allow the user to choose to only display NEOs that are orbiting a certain orbital body.

Your UI element should look like the following visualization:



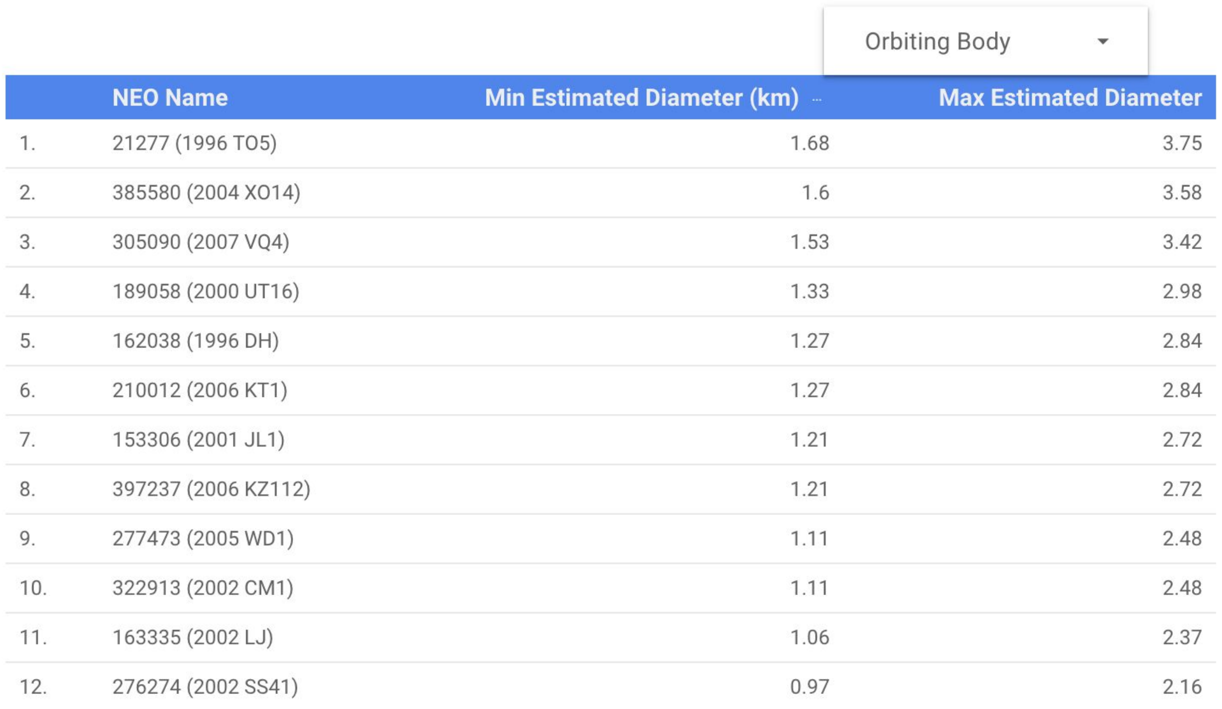
The dropdown should allow the user to select one orbital body name - if an orbital body is selected the chart should only display NEOs that are currently orbiting the selected body.

Tips:

- You will not be judged on the style of the dropdown, only its usability
- You may use any UI framework you see fit to complete this step

Step 3: Display as a Table

Implement a new way to display NASA’s data: as a Table.
Add a UI element that allows the user to switch between Chart view and Table View.



	NEO Name	Min Estimated Diameter (km)	Max Estimated Diameter
1.	21277 (1996 TO5)	1.68	3.75
2.	385580 (2004 XO14)	1.6	3.58
3.	305090 (2007 VQ4)	1.53	3.42
4.	189058 (2000 UT16)	1.33	2.98
5.	162038 (1996 DH)	1.27	2.84
6.	210012 (2006 KT1)	1.27	2.84
7.	153306 (2001 JL1)	1.21	2.72
8.	397237 (2006 KZ112)	1.21	2.72
9.	277473 (2005 WD1)	1.11	2.48
10.	322913 (2002 CM1)	1.11	2.48
11.	163335 (2002 LJ)	1.06	2.37
12.	276274 (2002 SS41)	0.97	2.16

Tips:

- You will not be judged on the style of the UI element or the Table, only its usability
- You may use any UI framework you see fit to complete this step

Do not start this step unless you have a complete and polished result for all other steps and only if you have time remaining on the 4-hour allowance. Candidates will not be penalized for a lack of answer on this step. There are two bonus steps, feel free to pick and choose.

[BONUS] Step BONUS: CSV Export

Add a button that allows the user to download the fetched NASA data as a CSV file. When clicked, this button should trigger a download of a “nasa.csv” file that respects the CSV specification, and contains the same data as your Table in step 3.

[BONUS] Step BONUS: Classes vs Hooks

If you wrote your stateful components of steps 1-3 with Classes, create a branch where you implement them as Hooks.
Inversely, if you wrote your stateful components of steps 1-3 with Hooks, create a branch where you implement them as Classes.

We very much thank you for the time you spent on this challenge.