Challenges

We are assessing the following skills

- Browser familiarity
- JS app proficiency
- Sense of architecture/design patterns
- Style/consistency/following best practices

Allow at least 3 hours to complete each part.

Do not be discouraged if you are unable to complete aspects of the challenge—it is designed to test all levels of ability.

Rules

- Complete the tests on your own.
- Do not disclose any test materials to anyone or post them online.
- Referencing of online resources is expected. You should comment with a reference when you do.
- You are encouraged to ask us clarifying questions. (Your recruiter will forward the questions so expect delays in response.)
- Certain parts of the challenge are intentionally left ambiguous. Use your best judgement to create a solution that aligns with your understanding of the problem. Make sure to state assumptions in the README.
- Note any deviations from the specification in the project readme.
- Be prepared to talk about the challenge in later interview rounds.

Deliverables

- Use a single git project for both parts and include the .git folder in your deliverable.
- Provide an archive (e.g., .zip) of all the project files (frontend/backend).
- Use the following layout:
    - Project root/
        - fe/
            - frontend project files/folders…
        - readme.md
        - other necessary files
        - .git/
- README should include:
    - Instructions on how to run each part of the challenge.
    - Brief description of rationale behind each tool/language/framework of choice.
    - Brief description of key challenges (it's okay to say you are not sure what you did was the best way to solve it. Be sure to justify your decision).
    - Note any caveats of your solution and potential downfall.
    - Note what you would do differently in a production environment.

App Instructions

1. Create a new Angular CLI or React project
2. Setup a data access class (intended to mock a backend service) according to **Database+API Instructions** section.
3. Make 2 new components. You may use any library you deem appropriate. And you may create more subcomponents but be sure to justify the need in your README.

- **Component A**
    - Retrieve data from the data access class described in **Database+API Instructions**.
    - Should have an input box to enter a node path.
    - On each keypress the component should query the API for a subtree matching that path. Inflight requests should be canceled for new ones.
    - Use Component B to render the returned subtree.
- **Component B**
    - Should render a returned node tree structure and all properties.
    - The label of a property should be GREEN if the value is greater than 10

3.

    a. For Angular projects:
        i. Use the Dialog component to make a reusable `Confirm` box.
        ii. Use the above technique to make a `Delete` button with confirmation for each node (this does not need to be connected to the API).
    b. For React projects:
        i. Use the browser-native confirmation dialog (or a library) to make a reusable `Confirm` box.
        ii. Use the above technique to make a `Delete` button with confirmation for each node (this does not need to be connected to the API).

4.

    a. For Angular projects:
        i. Create a pipe that renders how long ago it was since this item was created (e.g. *'created 1 hour ago'*).
        ii. Implement this pipe onto each item in the displayed tree.
    b. For React projects:
        i. Create a helper method that renders how long ago it was since this item was created (e.g. *'created 1 hour ago'*).
        ii. Use this method for each rendered node in Component B.

5. Create a unit test to assert that the color of the Component B label behaves as required.

Database+API Instructions

The database your frontend integrates with has the following structure:

- A rocket (root node) is built from a tree of nodes. Each node has a name. The path of a node can be inferred from the name hierarchy (e.g. *‘/root/parent/child’*).
- Child nodes have no name requirements, and there’s no limit to their depth (i.e. *`/root/parent/child/.../child-n`*).
- Each node can have any number of children nodes and properties. A property is a key value pair, where the key is a string and the value is a decimal number.

1. Create a data access class that supports the following behaviors and seed data. Entries with values are properties—others are nodes. See **API Call Examples** for reference of the backend service you’re mocking. ***Again, please do not use a real database or backend service***

   ### Behaviors

   - Create a node with a specified parent
   - Add a property on a specific node
   - Return the subtree of nodes with their properties for a provided node path

   ### Seed Data

   - "Rocket":
     - "Height": 18.000
     - "Mass": 12000.000
     - "Stage1"
       - "Engine1"
         - "Thrust": 9.493
         - "ISP": 12.156
       - "Engine2"
         - "Thrust": 9.413
         - "ISP": 11.632
       - "Engine3"
         - "Thrust": 9.899
         - "ISP": 12.551
     - "Stage2"
       - "Engine1"
         - "Thrust": 1.622
         - "ISP": 15.110

API Call Examples

Method	Endpoint	Result	description
GET	/Rocket	entire structure above	
GET	/Rocket/Mass	{“Mass”: 12000.000 }	

GET	/Rocket/Stage2/Engine1	{“Engine1”: {
		"Thrust": 1.622,
		"ISP": 15.110
		}
		}
			Returns the entire “engine1” subtree

POST	/Rocket/Stage2/RocketJr	{}	Adds the “RocketJr” node to the “Stage2” node

POST	/Rocket/Stage2/RocketJr
	Request Body: { “foo”: 20.2 }
		{
		“RocketJr”: {
		“foo”: 20.2
		}
		}
			Adds the property “foo” with value 20.2 to the “RocketJr” child node

5.