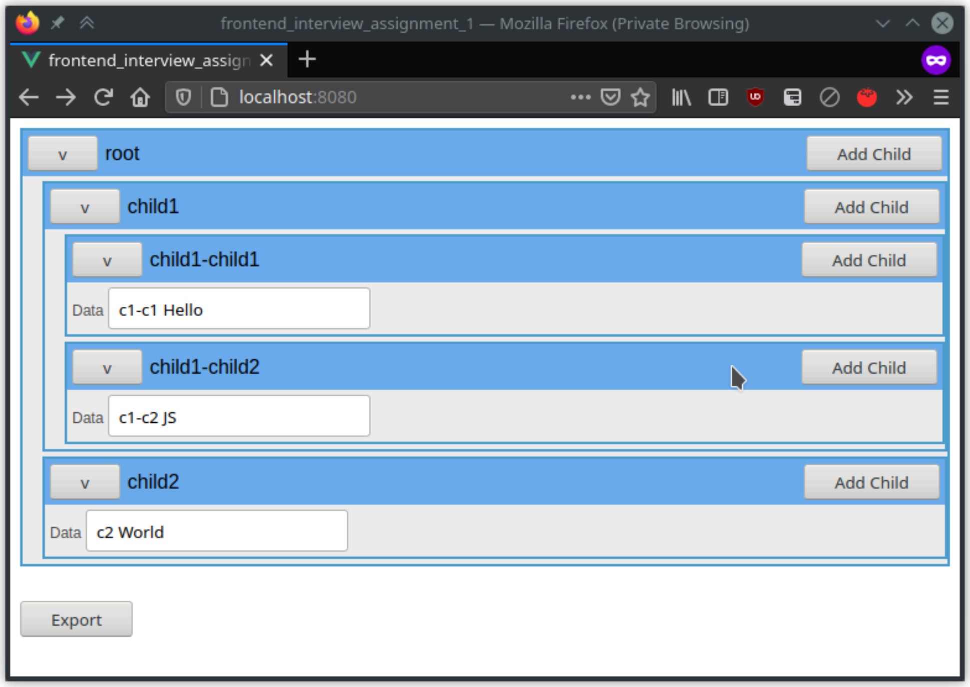
AIMonk Full Stack
Coding Assignment

Description



Create a full stack React/Vue and python web application frontend UI, backend, and database for Nested Tags Tree as shown in the screenshot above. The UI must contain a “TagView” component that supports rendering nested Tags children and text data, and the backend must accept the tree hierarchy from a REST endpoint and save it into the database. The tree hierarchy (data structure) for the Tags Tree is as given below:

```
tree: {
    name: 'root',
    children: [
        {
            name: 'child1',
            children: [
                {name: 'child1-child1', data: "c1-c1 Hello"},
                {name: 'child1-child2', data: "c1-c2 JS"}
            ]
        },
        {name: 'child2', data: "c2 World"}
    ]
}
```

In the above nested tag tree data-structure, each tag element has a “name” property along with either a “children” Array or a “data” String but not both. The Tags elements can be nested recursively inside the children array as shown in the example data structure. In addition, each Tag element must have an “Add Child” button as shown in the screenshot above. On clicking the “Add Child” button, a new TagView component must be instantiated inside its parent by replacing the “Data” textfield and its value.

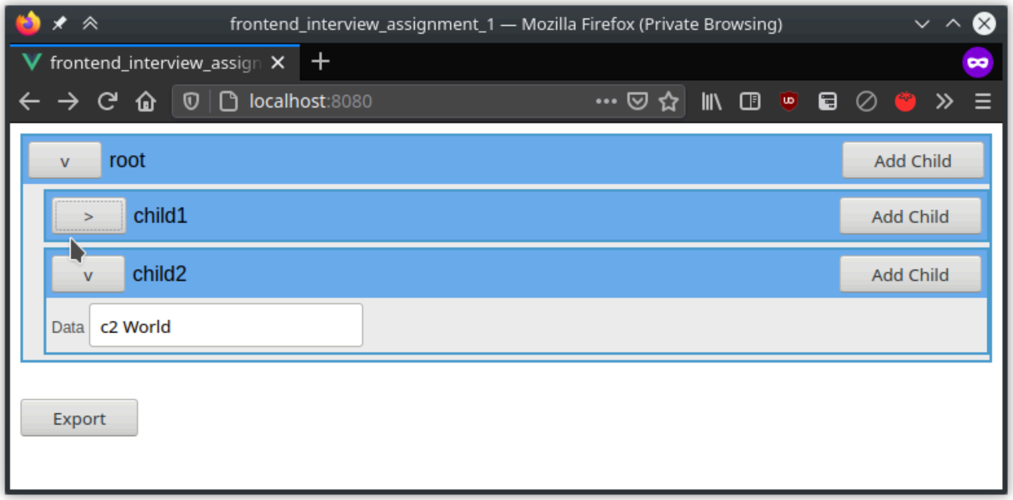
Very Important: The above data-structure is an example data structure. The tree hierarchy (data structure) can be altered by adding new children using the “Add Child” button or by modifying the tree object in the source code. The UI must update and render the new hierarchy as and when new children are added.

Once the tree hierarchy is saved in the database by the backend, the UI must fetch and load the previously saved tree hierarchy and display it on the frontend UI.

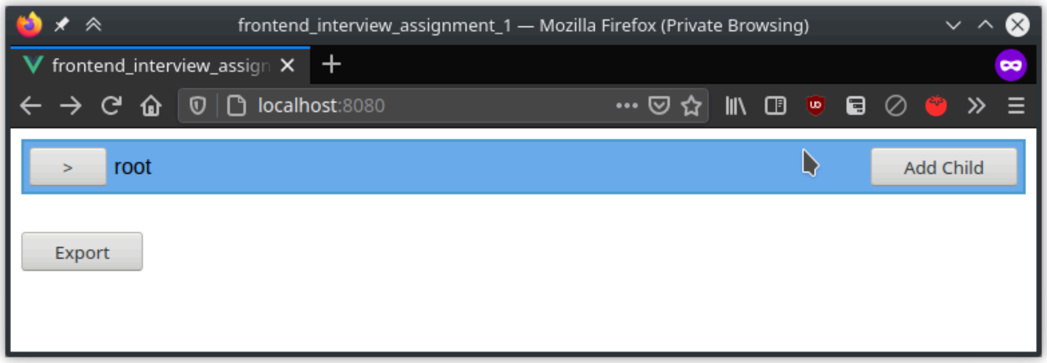
Requirements

Frontend UI

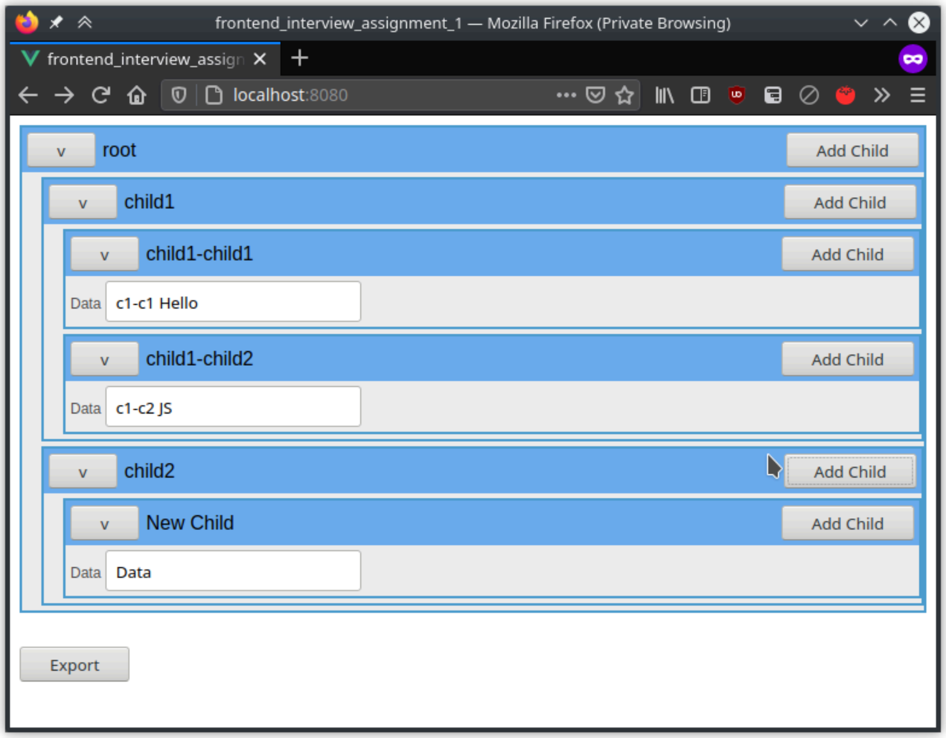
1. The UI for your solution must be implemented in a web framework such as React.js, Vue.js, Next.js, etc or any other similar component based web frontend framework.
2. The “data” String inside each Tag should be rendered using a text input field. Editing the contents of the text input elements should modify the contents of the internal tree data structure object.
3. Each Tag must be collapsible on clicking the “v” or “>” button next to the tag name header. The collapsed Tag must display only the “name” header but hide its “children” and “data” string as shown in the screenshot below.


frontend_interview_assignment_1 — Mozilla Firefox (Private Browsing)
frontend_interview_assign
localhost:8080
v
root
Add Child
>
child1
Add Child
v
child2
Add Child
Data
c2 World
Export

Note: Every Tag element must be collapsable recursively, including the root tag as shown in the below screenshot. If a tag is collapsed, then the leftmost button should show the string “>” else it must show “v”.

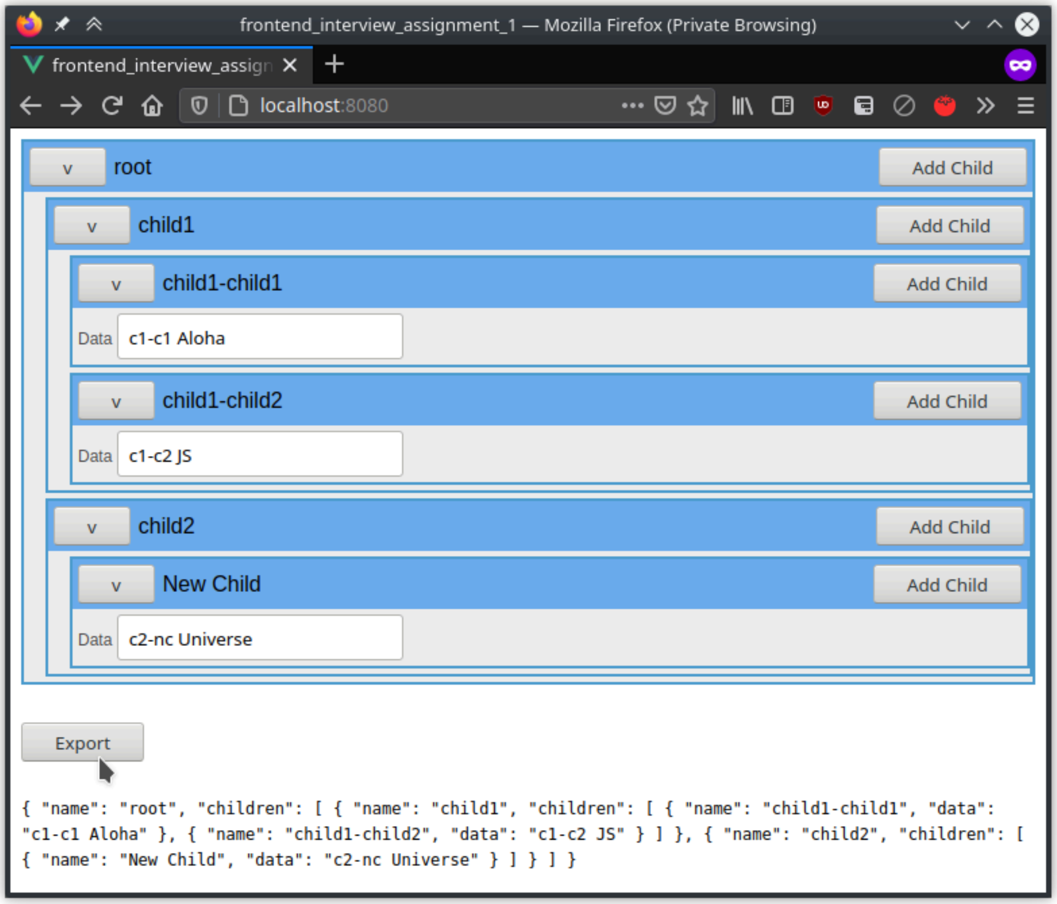

frontend_interview_assignment_1 — Mozilla Firefox (Private Browsing)
frontend_interview_assign
localhost:8080
>
root
Add Child
Export

4. When the “Add Child” button is clicked, then a new child must be created inside the parent. If the parent had a “data” property and textfield, then the “data” property and textfield must be replaced with a “children” property and a single new child must be added inside it. The new child Tag element must have the name property set to “New Child” and the “data” property must be “Data” as shown in the screenshot below.



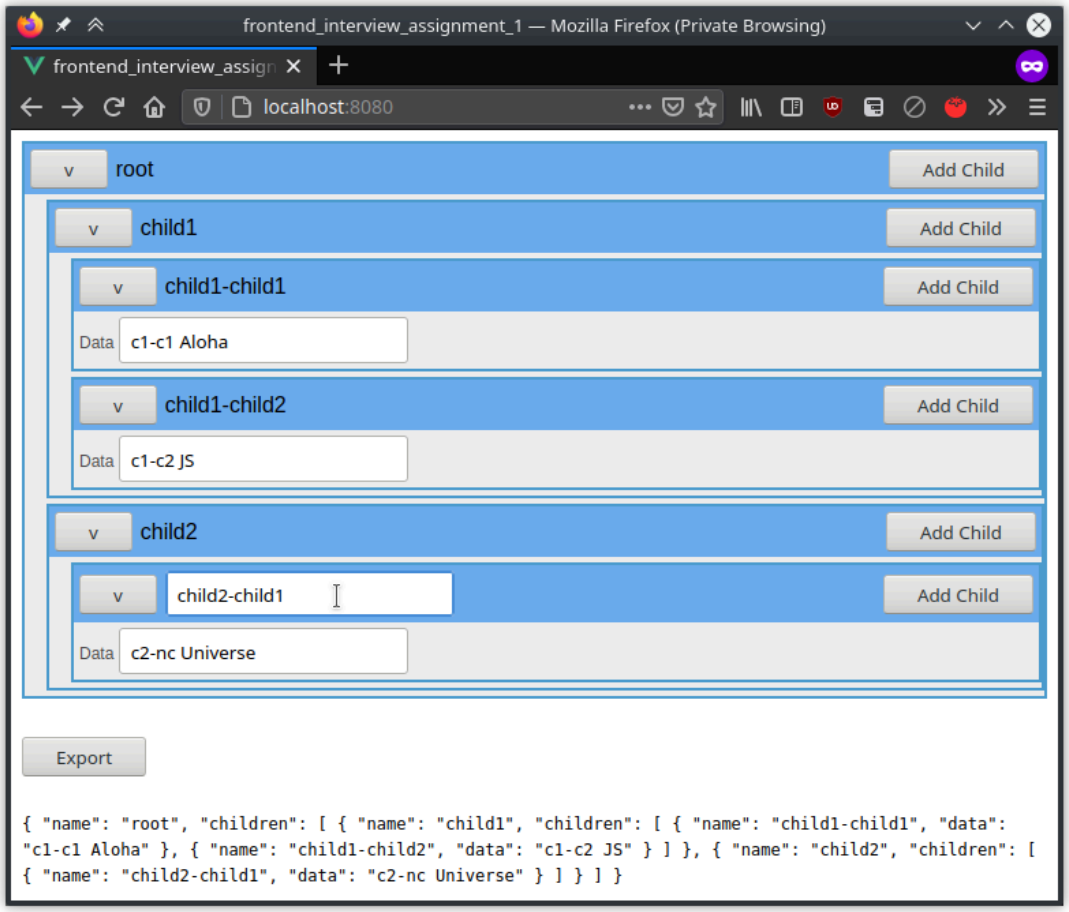
5. There must be an “Export” button which must output the current data and JSON structure of the tree hierarchy on click as shown in the screenshot below. It should also call the REST API that you implement and save the tree hierarchy to the database.

   (Hint: You can use JSON.stringify to convert the tree object to string, the necessary properties can be extracted recursively. Also, you must only output the “name”, “children”, and “data” properties of each Tag, all other internal values must not be present in the exported string.)


frontend_interview_assignment_1 — Mozilla Firefox (Private Browsing)
frontend_interview_assign
localhost:8080
v root Add Child
v child1 Add Child
v child1-child1 Add Child
Data c1-c1 Aloha
v child1-child2 Add Child
Data c1-c2 JS
v child2 Add Child
v New Child Add Child
Data c2-nc Universe
Export

```
{ "name": "root", "children": [ { "name": "child1", "children": [ { "name": "child1-child1", "data":
"c1-c1 Aloha" }, { "name": "child1-child2", "data": "c1-c2 JS" } ] }, { "name": "child2", "children": [
{ "name": "New Child", "data": "c2-nc Universe" } ] } ] }
```

6. All modified values of the "data" property for each Tag must be reflected in the exported string on clicking the export button. For example, the "data" string for the first textfield has been modified to "c1-c1 Aloha" which is reflected in the exported string as shown in the screenshot above.
7. You can either style the Tag Elements as shown in the screenshots or you may use your own creativity to style them using any CSS framework (Tailwind CSS, Material Design, etc) of your choice to make it look nicer.
8. Bonus Question: It should be possible to modify the name of any Tag element by clicking on the tag name displayed in the blue header as shown in the screenshot below. A new text input form element should be shown in place of the name. The user must be able to type a new name in the textfield and must press the Enter key on the keyboard to set the new name and hide the input text field. The newly typed name must be shown and updated on the UI and JSON exported string.



```
{ "name": "root", "children": [ { "name": "child1", "children": [ { "name": "child1-child1", "data":
"c1-c1 Aloha" }, { "name": "child1-child2", "data": "c1-c2 JS" } ] }, { "name": "child2", "children": [
{ "name": "child2-child1", "data": "c2-nc Universe" } ] } ] }
```

Backend and Database

1. You can use any SQL based database (PostgreSQL, MySQL etc) to store the tree hierarchy data. In addition, you can either use any ORM or direct queries to access data from the backend.
2. Identify and create the most suitable database schema to store the tree hierarchy data.
3. You must implement the backend in a python based web framework (FastAPI, Django, Flask, etc).
4. In the backend, you must implement a POST API endpoint to save a new tree hierarchy record.
5. When the UI of the app is opened, it must initially call the backend GET API to fetch all of the previously saved tree hierarchies. If there are multiple tree records saved in the DB, then this API must return all the records as a list, and the frontend UI must display each tree separately one below the other on the same page.

6. When an existing tree hierarchy is loaded on the UI, and changes are made to the tree hierarchy, then clicking on the "Export" button should call the backend PUT API endpoint to update the existing tree hierarchy JSON with the updated value.

Note

Please share your solution in a zip file containing the full project along with all the frontend and backend source files and package.json/requirements.txt files. You can also share it as github repository links. Bonus points for deploying your solution on a hosted link such as Vercel, AWS or any other hosting solution using CI/CD integrations or Github Actions.

Feel free to reach out if you have any questions.