# VectorShift

## Frontend Technical Assessment Instructions

Thank you for taking the time to interview with us at VectorShift! As part of the interview process, we would like you to complete a frontend technical assessment. You can find all files necessary for the assignment in the `/frontend/src` and `/backend` folders. Feel free to make any changes to the provided files, including adding new files, deleting existing files, installing new packages, and modifying any provided code. Please use JavaScript/React for the frontend and Python/FastAPI for the backend.

The assessment consists of four parts, which are detailed below. You are encouraged to read all parts of the assessment before starting so that you can plan your approach. You can run the frontend code by navigating to the `/frontend` directory and running (1) `npm i` and (2) `npm start`. You can run the backend code by navigating to the `/backend` directory and running `uvicorn main:app --reload`.

Feel free to reach out to recruiting@vectorshift.ai if you have any questions.

### Part 1: Node Abstraction

In `/frontend/src`, you will find a folder called nodes. This folder contains JavaScript files for four types of nodes (inputs, outputs, LLMs, and text). Each of these nodes contains different text, content, and input/output connections (called “Handles”), but there is also a significant amount of shared code between nodes.

Currently, you could create a new node by copying an existing node into a new file and making modifications, but this approach ends up rewriting significant amounts of code. While this approach is tractable for a small number of nodes, it becomes difficult to maintain as the number of nodes increases.

Your task is to create an abstraction for these nodes that speeds up your ability to create new nodes and apply styles across nodes in the future.

Once you have created your abstraction, make five new nodes of your choosing to demonstrate how it works. Don’t spend too long worrying about what the nodes actually do; you should use this as an opportunity to showcase the flexibility/efficiency of your node abstraction.

### Part 2: Styling

The frontend files you receive do not apply any significant styling. Your task is to style the various components into an appealing, unified design. You can use the VectorShift’s existing styles as inspiration if you’d like, but you are also free to create your own design from the ground-up. You can use whatever React packages/libraries that you would like.

## Part 3: Text Node Logic

The Text node included in the `/frontend/src/nodes` has a field for text input. We want to improve the functionality of this text input in two ways.

First, we want the width and height of the Text node to change as the user enters more text into the text input, improving visibility for what the user types in.

Second, we want to allow users to define variables in their text input. When a user enters a valid JavaScript variable name surrounded by double curly brackets (e.g., "{{ input }}"), we want to create a new Handle on the left side of the Text node that corresponds to the variable. For examples of what this should look like, you can try using a VectorShift Text node or watching Tutorials using the VectorShift Text node.

## Part 4: Backend Integration

In the `/backend` folder, you will find a very simple Python/FastAPI backend. Your task is to build an integration between the frontend you completed and this simple backend.

On the frontend, you should update `/frontend/src/submit.js` to send the nodes and edges of the pipeline to the `/pipelines/parse` endpoint in the backend when the button is clicked.

On the backend, you should update the `/pipelines/parse` endpoint in `/backend/main.py` to calculate the number of nodes and edges in the pipeline. You should also check whether the nodes and edges in the pipeline form a directed acyclic graph (DAG). The response from this endpoint should be in the following format: `{num_nodes: int, num_edges: int, is_dag: bool}`.

Once you have updated the button and the endpoint, you should create an alert that triggers when the frontend receives a response from the backend. This alert should display the values of `num_nodes`, `num_edges`, and `is_dag` in a user-friendly manner.

The final result should allow a user to create a pipeline, click submit, and then receive an alert with the number of nodes/edges as well as whether the pipeline is a DAG.