TECHNICAL DOCUMENTATION

VectorShift Frontend Technical Assessment

System Architecture & Implementation Report

VectorShift – Frontend Technical Assessment Documentation

Project Overview

This project is a sophisticated Pipeline Builder that allows users to create, visualize, and validate workflows. It features a modern, drag-and-drop interface with dynamic node logic and backend-integrated validation.

I added new nodes to the project: - fileNode - noteNode - logicNode - integrationNode - promptNode

Technical Implementation

1. Node Abstraction Layer

A robust abstraction layer was introduced to streamline node creation and ensure UI consistency. - **BaseNode.js**: A reusable component that handles common node features such as: - **Header Actions**: Minimize/Maximize and Delete functionality. - **Dynamic Handles**: Automated rendering of input and output handles. - **Visual Indicators**: Highlight effects for connected nodes. - **Styling Hooks**: Built-in support for theme-aware styling.

2. New Custom Nodes

Beyond the initial requirements, 5 new specialized nodes were added using the new `BaseNode` abstraction: 1. **fileNode.js**: Handles file uploads and input signals. 2. **noteNode.js**: A stylized "sticky note" for user documentation within the canvas. 3. **logicNode.js**: Implements conditional logic and flow control. 4. **integrationNode.js**: For connecting to external APIs and services. 5. **promptNode.js**: Specifically designed for prompt engineering and LLM instructions.

3. Dynamic Text Node Logic

The **TextNode.js** was enhanced with advanced interactive features: - **Auto-resizing**: Text areas dynamically adjust height as the user types, ensuring content is always visible. - **Variable Detection**: Uses regex to identify `{{variable_name}}` patterns in real-time. - **Dynamic Handle Generation**: Automatically adds target handles on the left side for every unique variable detected in the text.

4. UI/UX & Styling Enhancements

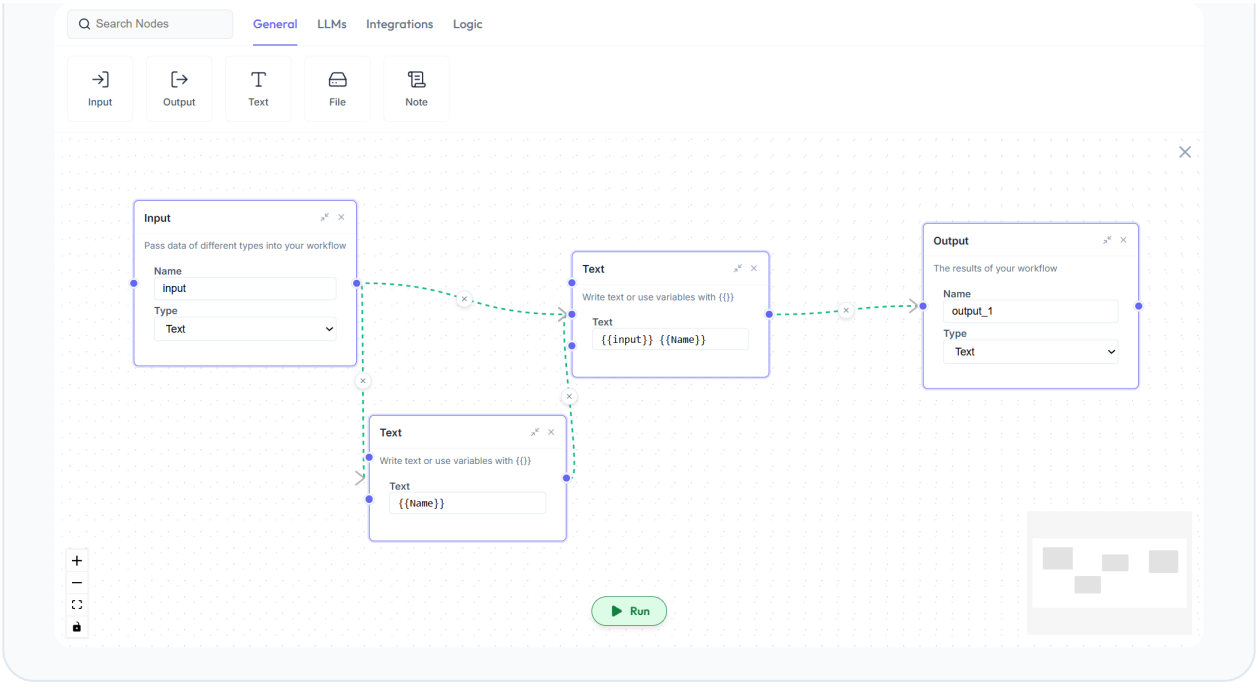
The application underwent a significant visual overhaul to provide a premium feel: - **Custom CSS Engine**: Implemented in **ModifiedStyle.css** using CSS variables for high-performance styling. - **Interactive Toolbar**: A scrollable, category-based navbar and toolbar for easy node access. - **"Run" Workflow UX**: for Instead of submit button i perfromed changes in button name as Run Workflow {Run Button}. (Because Run is more relavant to user and That the user can understand easily and if we trigger the workflow we get the output Like: DAG is valid or not, if not then we get the error message) As per better UX. - **ResultModal.js**: Instead of basic alerts, results are displayed in a clean, modern card featuring: - **SVG Mini-map**: A live visualization of the submitted pipeline. - **Detailed Stats**: Node count, edge count, and DAG status. - **Intuitive Feedback**: Color-coded results (success/error) and helpful descriptions.

5. Backend Logic (FastAPI)

The backend in **main.py** was updated to handle complex pipeline parsing: - **Endpoint**: `/pipelines/parse` accepts the full node and edge state. - **DAG Validation**: Implements **Kahn's Algorithm (Topological Sort)** to robustly detect cycles and ensure the pipeline is a Directed Acyclic Graph (DAG). - **Graceful Error Handling**: Includes detailed logging and error reporting to the frontend.

Fig 1.0: Real-time Pipeline Logic and Node Architecture

(By Brahmarishendra)



Deployment & Setup

- [x] Fronten*: `npm start` (Runs on port 3000)

Our Design Philosophy & UX Rationale

Beyond the technical requirements, I focused on making this pipeline builder feel like a polished, user-centric product. Here's the "how" and "why" behind some of my key design decisions:

Why the "Run" Button?

Instead of a generic "Submit" button, I updated it to **"Run Workflow"**. I feel that "Run" is much more relevant for this kind of tool—it tells you that you're actually triggering a process and getting an immediate result, rather than just sending a form. It's about making the workflow feel more interactive and purposeful.

Custom Result Modal vs. Browser Alerts

Let's be honest—standard browser alerts are disruptive and break the flow. I built a custom **Result Modal** with an illustrative UI that feels like a natural part of the product. It even includes a live SVG visualization of your pipeline, allowing you to see exactly what you built in a simple,

visual way. It helps you understand "what happened" based on your input without any guesswork.

Organized Navbar & Categorization

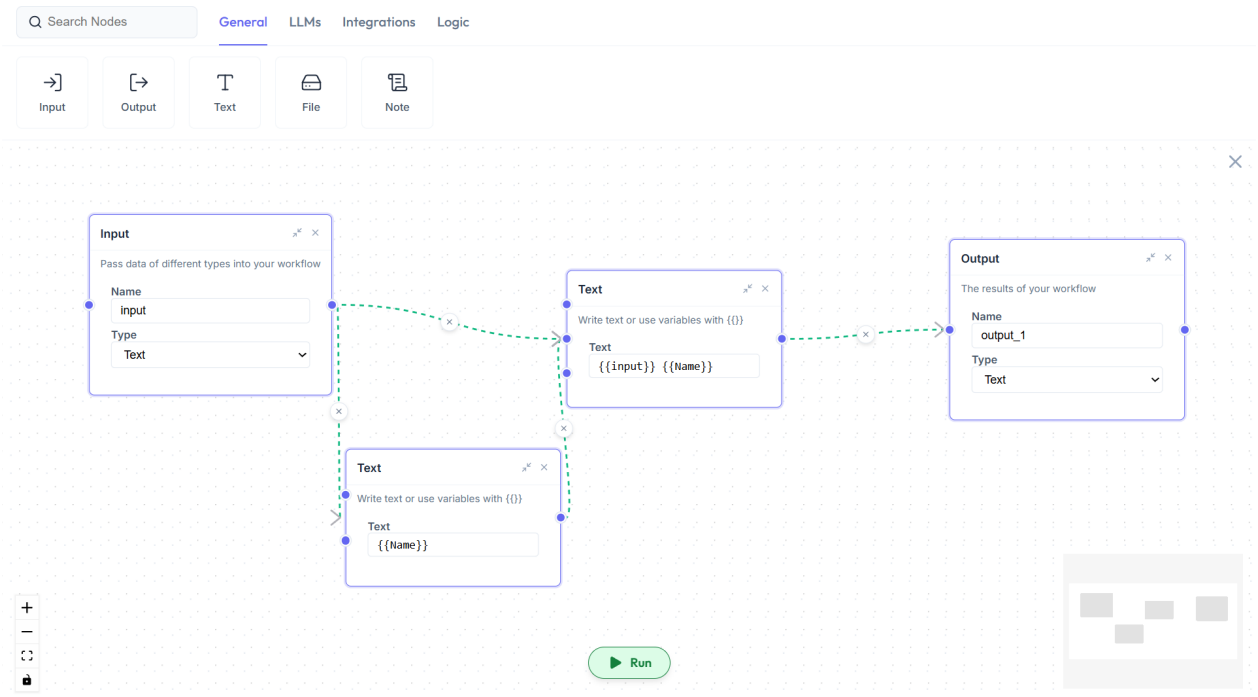
As the number of nodes grows, it's easy to get lost. So, I added a **Pipeline Navbar** with categories like General, LLMs, Integrations, and Logic. This organized approach keeps the experience smooth and ensures you can find exactly what you need without being overwhelmed. I also added a search bar for even faster access.

My Approach to UI/UX

My goal was to create a tool that stays out of your way and lets you focus on building. From the subtle hover effects to the smooth SVG connection lines and the ability to minimize complex nodes, every detail was carefully chosen to make the building process as intuitive and simple as possible. - **Backend**: `uvicorn main:app --reload` (Runs on port 8000)

Final System Visualization

Comprehensive overview of the developed component ecosystem and workflow canvas.



End of Report

Confidential - Internal Engineering Review