STREAMFORGE – Visual Data Pipeline Orchestration Platform

PROJECT REPORT

Submitted in the partial fulfillment of the requirements for the award of degree in

BACHLEOR OF COMPUTER APPLICATIONS

Submitted By,

Abhinav R Nair (230021079339)

SEMESTER VI

SWAMY SASWATHIKANANDA COLLEGE POOTHOTTA

(Affiliated to Mahatma Gandhi University)

2026

SWAMY SASWATHIKANANDA COLLEGE

POOTHOTTA

(Affiliated to Mahatma Gandhi University)

CERTIFICATE

This is to certify that the project entitled "**STREAMFORGE – Visual Data Pipeline Orchestration Platform**" submitted in partial fulfilment of the requirements for the award of the degree in **Bachleor of Computer Applications** is a bonafide report of the project done by **Abhinav R Nair (230021079339)** during the VIth semester in the academic year 2025-2026.

Internal Guide **Head of the department**

Examiners:

1……..

2……..

College Seal **Department Seal**

DECLARATION

I hereby declare that this project work entitled **"STREAMFORGE – Visual Data Pipeline Orchestration Platform"** is a record of the original work done by me under the guidance of Ms. Divya R**, Associate Professor**, Department of Computer Applications, **SWAMY SASWATHIKANANDA COLLEGE,** and the project work has not formed the basis for the award of any Degree/Diploma or similar title to any candidate of University.

Internal Guide:	**Name and Signature of the candidates**
DIVYA R	**Abhinav R Nair**

ACKNOWLEDGEMENT

At the outset, I thank God Almighty for making my endeavour a success.

I express my gratitude to **Prof. K. S. ULLAS,** the principal of **SWAMY SASWATHIKANANDA COLLEGE, POOTHOTTA** for providing me with adequate facilities, ways and means by which I was able to complete the project work.

I express my sincere thanks to **Assoc. Professor Ms. DIVYA R,** Head of the Department of Computer Applications, and my project coordinator **Ms. SHOBIMOL S** and my project guide **Ms. DIVYA R** who has been showing deep interest in my project and inspired me through development by valuable suggestions, and all the faculty members of the department of Computer Applications for their sincere help and support.

Last but not the least, I also express my profound gratitude to all other members of the faculty and well-wishers who assisted me in various occasions during the project work.

CONTENTS

1. SYNOPSIS

SYNOPSIS

StreamForge is a centralized "No-Code/Low-Code" platform designed to democratize data engineering by allowing users to build complex ETL (Extract, Transform, Load) workflows without writing code. The application features a visual drag-and-drop interface where users can design data flows using specialized nodes for tasks such as filtering, joining, and deduplicating datasets. This intuitive design enables both technical and non-technical users to process data efficiently while maintaining full traceability of datasets through a comprehensive data catalog.

To enhance productivity, the platform integrates a Google Gemini-powered AI chat assistant that helps users debug logic or generate pipeline structures directly from text prompts. Beneath the visual interface lies a powerful transformation engine capable of in-memory data processing using Pandas and NumPy, allowing for complex operations like aggregation and mathematical calculations. Additional collaboration features allow teammates to view and edit pipelines in real-time, while an automated scheduling system ensures tasks run at set intervals.

built on a robust modern tech stack utilizing React with Vite for the frontend and Python with Flask for the backend logic. The system leverages Socket.IO for real-time updates and SQLite for data persistence, ensuring a responsive user experience. It also includes administrative tools for user management and system-wide broadcasting, providing a secure and scalable environment for data orchestration.

2. INTRODUCTION

INTRODUCTION

StreamForge is a centralized digital platform designed to democratize data engineering by offering a "No-Code/Low-Code" solution for building complex ETL (Extract, Transform, Load) workflows. By removing the barrier of writing code, it allows both technical and non-technical users to design, manage, and monitor data pipelines effortlessly, making data processing accessible to a wider audience.

The platform features a user-friendly visual interface where pipelines are constructed using a drag-and-drop builder with a comprehensive catalog of specialized nodes. Users can easily ingest data from various sources like CSV or Excel and perform advanced transformations—such as filtering, joining, and deduplicating datasets—using a powerful in-memory engine powered by Pandas and NumPy.

Beyond standard data tasks, StreamForge integrates cutting-edge features like a Google Gemini-powered AI assistant to help users debug logic or generate pipelines from text prompts. It also supports real-time collaboration, enabling teams to view and edit workflows simultaneously, and includes automated scheduling capabilities to run tasks at specific intervals or times.

Developed by Abhinav R Nair, the system is built on a modern tech stack utilizing React (Vite) for the frontend and Python (Flask) for the backend logic. The application ensures a responsive experience with Socket.IO for real-time updates and includes robust administrative tools for user management and system-wide broadcasting.

3. SYSTEM ANALYSIS

SYSTEM ANALYSIS

3.1. EXISTING SYSTEM

Currently, data processing and ETL (Extract, Transform, Load) workflows are predominantly handled through manual coding using languages like Python and SQL, or by utilizing complex, developer-centric orchestration tools like Apache Airflow. In most organizations, business analysts and non-technical users lack the specialized skills to build these pipelines, forcing them to rely entirely on dedicated data engineering teams to access, clean, or move data.

However, this traditional approach creates significant operational bottlenecks. Writing custom scripts for every new data requirement is time-consuming and prone to human error, often resulting in unmanaged "spaghetti code" that is difficult to debug or scale. Furthermore, these code-heavy systems lack visual transparency; stakeholders cannot easily see how data is being transformed or trace its origin, turning the data pipeline into a "black box." This dependency on technical experts slows down decision-making, as even simple data requests can take days or weeks to fulfill while tickets sit in an engineering backlog.

Drawbacks of existing system:

- High Technical Barrier (Requires Coding)
- Dependency on Specialized Data Teams
- Lack of Process Visibility (Black Box)
- High Maintenance Overhead
- Slow Turnaround Time
- Error-Prone Manual Scripts

3.2. PROPOSED SYSTEM

The proposed system, **StreamForge**, is a next-generation Data Pipeline Orchestration Platform designed to democratize data engineering by eliminating the technical barriers of traditional ETL processes. Unlike existing manual methods that rely on complex code and specialized teams, StreamForge offers a "No-Code/Low-Code" visual interface where users can build sophisticated data workflows simply by dragging and dropping nodes.

This system addresses critical issues such as lack of process visibility and high maintenance overhead by introducing features like a comprehensive Data Catalog for full lineage traceability and an AI-powered Chat Assistant (Google Gemini) to automate logic generation and debugging. With a focus on accessibility and teamwork, the platform enables real-time collaboration, allowing technical and non-technical users alike to design, schedule, and monitor data pipelines in a secure, centralized environment.

Advantages:

- **Intuitive Visual Interface:** Drag-and-drop builder eliminates the need for coding.
- **AI-Driven Assistance:** Integrated Generative AI for debugging and workflow creation.
- **Real-Time Collaboration:** Team members can edit and view pipelines simultaneously.
- **Full Traceability:** Built-in Data Catalog and lineage tracking for compliance.
- **Automated Scheduling:** Cron-style automation for reliable task execution.

3.3. MODULE DISCRIPTION

A module is a small, self-contained part which is the building block in software development that does a specific job. They help in organizing complex programs by breaking them down into smaller parts.

3.3.1. Administration module

The administrator has exclusive access to a centralized dashboard for managing the platform's user base. They can monitor the total data processed per user and have the authority to suspend accounts if necessary. Additionally, the admin can send system-wide broadcasts to provide real-time notifications to all active users and queue email blasts for critical updates.

3.3.2. User management module

Users can utilize the visual pipeline builder to design complex ETL workflows using a drag-and-drop interface without writing code. They can collaborate with teammates in real-time to view or edit pipelines and schedule automated tasks using cron-style intervals. The module also allows users to interact with the AI chat assistant for debugging logic and instantly visualize processed data through generated charts.

3.4. FEASIBILITY

A feasibility study is undertaken to determine the possibility or probability of either improving the existing system or developing a completely new system. It helps to obtain an overview of the problem and to get a rough assessment of whether a feasible solution exists. This is essential to avoid committing large resources to a project.

3.4.1. Technical Feasibility:

- **Modern Stack:** The system is built on a robust, industry-standard technology stack (React, Python, Flask) that ensures stability and maintainability.
- **AI Integration:** The integration of Google Gemini API for the AI assistant is technically viable and provides scalable logic generation capabilities.
- **Performance:** The use of Pandas and NumPy for in-memory data processing allows for high-performance handling of complex transformation tasks.

3.4.2. Economic Feasibility:

- **Cost Reduction:** By enabling non-technical users to build pipelines without code, the organization reduces the need for expensive, specialized data engineering resources.
- **Time Efficiency:** The visual drag-and-drop interface significantly speeds up the creation of ETL workflows compared to writing manual scripts, leading to faster operational turnaround.
- **Low Overhead:** Utilizing open-source technologies (Python, SQLite) keeps initial licensing and infrastructure costs low.

3.4.3. Social Feasibility:

- **Democratization:** The platform lowers the barrier to entry for data engineering, empowering a wider range of users to access and manipulate data.
- **Collaboration:** Real-time sharing features foster better teamwork between technical and non-technical stakeholders.
- **User Adoption:** The intuitive "No-Code" interface is designed to be user-friendly, ensuring high adoption rates among users intimidated by traditional coding environments.

4. REQUIREMENT ANALYSIS

REQUIREMENT ANALYSIS

Requirement analysis task is a process of discovery, refinement, modelling and specification both the developers and customer take an activity role in requirement analysis can be divided into:

1. Problem recognition
2. Problem evaluation & synthesis
3. Modeling

4.1. Problem Recognition

The primary problem is that traditional data processing and ETL (Extract, Transform, Load) workflows currently rely heavily on manual coding (Python/SQL) or complex, developer-centric orchestration tools. This creates a high technical barrier that excludes non-technical users from managing their own data.

- **Dependency on Specialists:** Business analysts must rely on data engineering teams for even simple data requests, leading to bottlenecks.
- **Lack of Visibility:** Hand-coded scripts often become "black boxes" where stakeholders cannot visualize how data is being transformed or trace its lineage.
- **Inefficiency:** Writing custom scripts for every new data requirement is time-consuming and error-prone compared to automated solutions.

4.2. Problem Evaluation & Synthesis

The proposed **StreamForge** system is designed to directly address the limitations of existing manual methods by creating a centralized, "No-Code/Low-Code" platform for data orchestration.
The system aims to provide a superior user experience through several key advantages:

- **Democratized Access:** It offers a visual drag-and-drop interface that allows users to build complex pipelines without writing a single line of code, making data engineering accessible to everyone.
- **Intelligent Automation:** The platform integrates an AI Chat Assistant (Google Gemini) to help users generate logic and debug errors, significantly reducing the learning curve.
- **Collaboration & Traceability:** StreamForge introduces real-time collaboration features and a comprehensive Data Catalog, ensuring teams can work together securely while maintaining full visibility over data lineage.

4.3. Modeling

The system's architecture and data flow are defined through a detailed database schema using SQLAlchemy models to ensure data integrity.

Data Modeling (Database Tables):

The system's data is organized across several tables to handle user management, pipeline execution, and file storage:

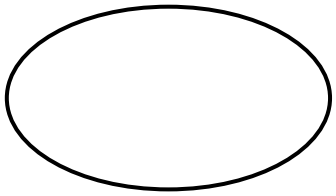
- **User and Notification:** The User table stores credentials, admin status, and usage metrics like total_processed_bytes. The Notification table manages system alerts and user preferences for success/failure updates.
- **Pipeline and SharedPipeline:** The Pipeline table defines the core logic, storing the visual structure as JSON and scheduling details. The SharedPipeline table manages collaboration, tracking which users have "viewer" or "editor" roles on specific pipelines.
- **PipelineRun:** This table tracks the execution history, recording the status (Running, Success, Failed), timestamps, and execution logs for auditing purposes.
- **DataSource and ProcessedFile:** These tables handle the input and output data. DataSource catalogues uploaded files and their metadata (row count, columns). ProcessedFile tracks the outputs generated by pipelines, establishing lineage by linking back to the source_pipeline.

DATA FLOW DIAGRAM(DFD)

A DFD also known as the bubble chart has the purpose of clarifying system requirements and identifying major information that will become programs in system design. A DFD is a pictorial representation of network that describes the flow of data through a system. The symbols used in Data Flow Diagram are:

- It represents a data source or destination

- It represents flow of data



- It represents a process that transforms data

- It represents data storage (e.g.: table)

5. SYSTEM SPECIFICATION

SYSTEM SPECIFICATION

5.1. HARDWARE REQUIREMENTS

Component	Requirement
CPU Type	Intel Core i3 / i5 or equivalent (Multi-core recommended)
RAM	4 GB or above (8 GB recommended for data processing)
Hard Disk Drive	40 GB or greater

5.2. SOFTWARE REQUIREMENTS

Component	Requirement
Operating System	Windows 10 / 11, Linux, or macOS
Front-End	React (Vite), HTML5, CSS3, JavaScript
Back-End	Python (Flask), Socket.IO, Pandas
Database	SQLite (via SQLAlchemy)
Runtime Environment	Node.js (v18+), Python (v3.10+)
Web Browser	Google Chrome, Mozilla Firefox, or Microsoft Edge

6. SYSTEM DESIGN

SYSTEM DESIGN

System design is a process of developing specification for candidate system that meet the criteria established in the system analysis. Major step in design is the preparation of the input forms output reports in a form application to the user.

The main objective of the system design is to use the package easily by any computer operation. System design is the creative act of invention, developing new inputs, a database, offline files, method procedure and output for processing business to meet an organization objective. System design builds information gathered during the system analysis.

6.1. DATA DESIGN

Data design creates a model of data or information that is represented at a higher level of abstraction. The structure of data has always been an important part of software design. The software design activities translate this requirements model into data structure at software component level.

In **StreamForge**, data design utilizes a relational database schema (SQLite via SQLAlchemy) to manage the complex relationships between users, workflows, and datasets. Key data entities include:

- **User:** Stores authentication details, administrative status, and usage metrics like total_processed_bytes.
- **Pipeline:** Defines the structure of the data workflow (stored as JSON), including scheduling configuration (cron expressions) and ownership.
- **PipelineRun:** Tracks the execution history, status (Running, Success, Failed), and logs for every process run.
- **DataSource & ProcessedFile:** Manages file metadata, establishing a lineage link between input files and the pipelines that transform them.

6.2. ARCHITECTURAL DESIGN

Architectural design is a comprehensive framework that describes its form and how they fit together. The properties of component interact with other components. Architectural design focuses on the representation of structure of the software.

StreamForge follows a modern Client-Server architecture:

- **Frontend (Client):** A Single Page Application (SPA) built with **React (Vite)** that handles the visual aspects of the pipeline builder using **React Flow** and manages state with **Framer Motion**.
- **Backend (Server):** A **Flask** (Python) application that serves as the central controller, handling API requests, database interactions, and AI integration.

- **Real-Time Communication Layer:** Uses **Socket.IO** to establish a bidirectional link between the client and server, enabling live log streaming and status updates during pipeline execution.

6.3. PROCEURAL DESIGN

Procedural design or component level design occurs after data, architectural and interface design must be translated into operational software. The procedural design for each component, represented in graphical, tabular or text-based notation, is primary work product produced during component level design.

For this system, procedural logic is encapsulated in the **Pipeline Engine**:

- **Execution Logic:** The backend parses the JSON structure of a pipeline, creating a Directed Acyclic Graph (DAG) where nodes represent operations (e.g., Filter, Join).
- **Data Transformation:** Python scripts using **Pandas** and **NumPy** are dynamically generated to execute the specific logic defined in each node (e.g., in-memory merging of two dataframes).
- **Scheduling: APScheduler** handles the timing logic, triggering pipeline execution functions based on user-defined intervals.

6.4. INTERFACE DESIGN

Interface design creates and effective communication medium between a human and computer. Design identifies objects and action then creates a screen layout that forms the basis for user interface. **StreamForge** focuses on a high-interactivity interface:

1. **Visual Pipeline Builder:** A "canvas" interface where users interact with nodes as visual blocks, connecting them via "edges" to define data flow. This bridges the gap between technical logic and visual representation.
2. **AI Chat Sidebar:** A dedicated panel for the Gemini AI assistant, allowing users to converse with the system using natural language to debug errors or generate ideas.
3. **Data Visualization:** Output nodes generate graphical charts (Bar, Line, Scatter) directly within the UI, allowing users to instantly verify their data processing results.

7. CODING

CODING

The coding phase involves the translation of the system design into a machine-readable format. For the StreamForge project, this involved writing server-side logic in **Python** using the **Flask** framework to handle API requests and orchestration tasks, while the frontend was constructed using **React (Vite)** to deliver a responsive Single Page Application. The user interface leverages **React Flow** for the visual drag-and-drop capabilities and **Socket.IO** for real-time bidirectional communication, ensuring dynamic updates such as live log streaming.

The code is organized into a modular architecture, strictly separating concerns to promote maintainability and scalability:

- **Backend Logic:** The server-side code is structured using the "Application Factory" pattern in __init__.py, allowing for easy scalability and testing. Core functionalities are divided into distinct modules: routes.py handles HTTP endpoints for user management and pipeline operations, while pipeline_engine.py contains the complex logic for parsing JSON workflows and dynamically executing **Pandas/NumPy** transformations in-memory.
- **Frontend Architecture:** The client-side code is organized into reusable functional components within the src/components/ directory. Key components include PipelineBuilder.jsx for the interactive canvas and ChatAssistant.jsx for the AI interface. The application utilizes modern React hooks for state management and framer-motion for fluid UI animations.
- **Data Persistence & Security:** Database interactions are managed through **SQLAlchemy** models defined in models.py, which map Python classes like User, Pipeline, and PipelineRun to SQLite tables. This ORM layer ensures secure data handling and prevents SQL injection vulnerabilities. Additionally, the system integrates google-generativeai to power the AI features, encapsulating the prompt logic within chatbot_context.py.

8. SYSTEM TESTING

SYSTEM TESTING

Testing is the process of executing a program with the intent of finding errors. System testing is the stage of implementation aimed at ensuring that the system works accurately and efficiently before the live operation commences. A series of tests were performed for the proposed system to verify its stability and performance.

8.1. Unit Testing

Unit testing focuses verification effort on the smallest unit of software design—the module. For StreamForge, individual Python functions and React components were tested in isolation. For example, specific transformation logic in pipeline_engine.py was tested to ensure that data processing functions (like filtering or joining Pandas DataFrames) produced correct results given a known input. On the front-end, individual React components such as FilterNode.jsx or SourceNode.jsx were tested to verify they correctly handle user input and update their local state before interacting with the main canvas.

8.2. Integration Testing

Integration testing is a systematic technique for constructing the program structure while at the same time conducting tests to uncover errors associated with interfacing. In this project, the interaction between the React frontend and the Flask backend was rigorously tested. For instance, tests ensured that when a user clicks "Run Pipeline" in the interface, the JSON structure is correctly serialized and sent to the routes.py endpoint, and that the backend correctly interprets this structure to trigger the PipelineRun database entry. Additionally, the Socket.IO integration was tested to verify that real-time logs generated during backend execution were successfully emitted and displayed in the frontend console without latency.

8.3. Validation Testing

Validation testing is done to ensure the complete assembly of an error-free software product. Validation is termed successful if the software functions in a manner that is reasonably expected by the customer. The fully integrated StreamForge platform was tested against user requirements to ensure end-to-end functionality. This included verifying that a non-technical user could successfully upload a dataset, build a valid workflow using the visual builder, and generate a visualization, and that the AI Chat Assistant correctly interpreted natural language prompts to debug or suggest pipeline configurations.

9. SYSTEM IMPLEMENTATION

SYSTEM IMPLEMENTATION

A crucial phase in the system's life cycle is the successful implementation of the new system design. This involves deploying the application to a production environment, configuring the necessary runtime dependencies, and making the platform accessible to users. The implementation of **StreamForge** requires a server environment with **Python (v3.10+)** for the backend and **Node.js (v18+)** for the frontend.

The system can be implemented only after thorough testing is done and if it is found to be working according to the specifications. Given its modular architecture, StreamForge can be deployed directly on a host machine or containerized using Docker for easier orchestration.

The implementation process includes:

- **Environment Configuration:** Creating a .env file in the backend directory to securely store sensitive credentials, specifically the GEMINI_API_KEY required for the AI Assistant.
- **Dependency Installation:** Installing necessary Python libraries (via pip install -r requirements.txt) and Node.js packages (via npm install) to support the application's logic and interface.
- **Database Initialization:** Setting up the SQLite database (pipelines.db) using SQLAlchemy to automatically generate the required tables for users, pipelines, and runs.
- **Directory Permissions:** Ensuring the server has write permissions for the backend/uploads/ and backend/processed/ directories to handle user data ingestion and file generation.

10. SOFTWARE MAINTENANCE

SOFTWARE MAINTENENCE

Software maintenence for StreamForge focuses on the deployment, maintenance, and security of the data pipeline orchestration tool. This phase ensures that the system remains reliable, scalable, and secure as it manages complex data workflows.

10.1. Software Maintenance Plan

Maintenance is critical for StreamForge to ensure high availability and data integrity. The maintenance strategy is categorized into three distinct areas:

10.1.1. Corrective Maintenance

This involves reactive modification of the software to repair faults discovered after deployment.

- **Objective:** To ensure system stability by resolving processing failures or logic errors in pipeline execution.
- **Implementation for StreamForge:**
    - **Pipeline Recovery:** Fixing bugs where specific data pipelines fail to trigger or terminate unexpectedly.
    - **Error Logging:** Resolving issues where error logs are not correctly captured or displayed in the dashboard.
    - **Hotfixes:** Deploying immediate patches for critical failures, such as a memory leak causing the orchestration engine to crash.

10.1.2. Adaptive Maintenance

This involves modifying the system to cope with changes in the software environment.

- **Objective:** To keep StreamForge compatible with evolving technologies and external dependencies.
- **Implementation for StreamForge:**
    - **Dependency Updates:** Upgrading core libraries or the underlying framework (e.g., Python/Django or Node.js versions) to their latest stable releases.
    - **API Compatibility:** Updating connectors to ensure they remain functional when third-party services (like AWS S3, Google Sheets, or SQL databases) change their API endpoints or authentication methods.
    - **Environment Scaling:** Adjusting configuration settings to support deployment on new cloud infrastructure or container orchestration platforms like Kubernetes.

10.1.3. Perfective Maintenance

This focuses on evolving the software to improve performance or add attributes based on user feedback.

- **Objective:** To enhance the user experience and expand the system's capabilities.
- **Implementation for StreamForge:**
    - **Feature Expansion:** Adding new integration plugins, such as a connector for a new type of database or a cloud storage service.
    - **Performance Optimization:** Refactoring the scheduling algorithm to reduce latency in pipeline execution.
    - **UI Enhancements:** Implementing a drag-and-drop interface for easier pipeline construction or adding a dark mode based on user requests.

10.2. Security Implementation

To protect the integrity of the StreamForge platform and the sensitive data it processes, robust security measures are implemented at the application and network levels.

- **Access Control:**
    - **Role-Based Access Control (RBAC):** The system enforces strict permission levels, ensuring that only authorized administrators can modify pipeline configurations or view sensitive logs.
    - **Session Management:** Secure session handling is used to verify user identity. Sessions are monitored for suspicious activity, and idle sessions are automatically terminated to prevent unauthorized access.
- **Data Security:**
    - **Credential Encryption:** API keys, database passwords, and other sensitive credentials used in pipelines are encrypted at rest using industry-standard algorithms (e.g., AES-256).
    - **Secure Transmission:** All data transmitted between the client, the server, and external integrations is secured using SSL/TLS encryption.
    - **Input Validation:** Rigorous input validation is applied to all user-defined parameters to prevent injection attacks and ensure system stability.

11. CONCLUSION

CONCLUSION

The StreamForge project provides a robust framework that enables developers and data engineers to orchestrate complex data workflows effectively. It eliminates the inefficiencies and risks associated with managing disjointed scripts and manual data transfers. StreamForge allows users to design, schedule, and monitor data pipelines, creating a centralized control tower for data operations within a single, integrated environment.

The implementation of this system significantly reduces the operational overhead required to maintain data reliability and ensures that critical data is available when needed.

The goals achieved by this project are:

- **Efficient orchestration** of data pipelines and workflow configurations.
- **Automated reliability** through scheduled execution and error monitoring.
- **A user-friendly interface** for visualizing pipeline dependencies and status.
- **A scalable platform** that is flexible for integrating new data sources and future enhancements.

12.1. **APPENDIX A** (Tables)

12.1. APPENDIX A (Tables)

1. User

Field name	datatype	Constraints	description
id	INT	Primary Key,Auto Increment	Unique user ID
username	VARCHAR(64)	Unique, Index	User's login name
email	VARCHAR(120)	Unique, Index	User's email
password_hash	VARCHAR(128)	Not Null	Encrypted password
is_admin	BOOLEAN	Default False	Admin privileges
is_suspended	BOOLEAN	Default False	Ban/Suspension
total_processed_bytes	INT	Default 0	Data usage metric
notify_on_success	BOOLEAN	Default True	Notification
notify_on_failure	BOOLEAN	Default True	Notification

2. Pipeline

Field name	datatype	Constraints	description
id	INT	Primary Key, Auto Increment	Unique pipeline ID
name	VARCHAR(140)	Not Null	Name of the pipeline
structure	TEXT	Not Null	JSON string of nodes & edges
status	VARCHAR(20)	Default 'Ready'	Current pipeline state
created_at	DATETIME	Default current_timestamp	Creation date
user_id	INT	Foreign Key → User	Creator of the pipeline
schedule	VARCHAR(50)	Nullable	Cron for timing
next_run	DATETIME	Nullable	Next scheduled execution

3. PipelineRun

Field name	datatype	Constraints	description
id	INT	Primary Key, Auto Increment	Unique run ID
pipeline_id	INT	Not Null, Foreign Key → Pipeline	The pipeline being run
status	VARCHAR(20)	Default 'Running'	Execution status
start_time	DATETIME	Default current_timestamp	Start time of run
end_time	DATETIME	Nullable	End time of run
logs	TEXT	Nullable	JSON execution logs

4. DataSource

Field name	datatype	Constraints	description
id	INT	Primary Key, Auto Increment	Unique source ID
filename	VARCHAR(140)	Not Null	Original file name
file_type	VARCHAR(20)	Not Null	File extension (e.g., CSV)
file_size	VARCHAR(20)	Not Null	Human-readable size
filepath	VARCHAR(200)	Not Null	Storage path
upload_date	DATETIME	Default current_timestamp	When file was uploaded
user_id	INT	Foreign Key → User	Owner of the file
row_count	INT	Default 0	Number of rows
columns	TEXT	Nullable	JSON list of columns

5. ProcessedFile

Field name	datatype	Constraints	description
id	INT	Primary Key, Auto Increment	Unique output file ID
filename	VARCHAR(140)	Not Null	Output file name
file_type	VARCHAR(20)	Not Null	Output format
file_size_display	VARCHAR(20)	Not Null	Human-readable size
file_size_bytes	INT	Not Null	Size in bytes
filepath	VARCHAR(200)	Not Null	Storage path
created_at	DATETIME	Default current_timestamp	Creation timestamp
user_id	INT	Foreign Key → User	Owner of the file
row_count	INT	Default 0	Number of rows
columns	TEXT	Nullable	JSON list of columns
source_pipeline_id	INT	Nullable, Foreign Key → Pipeline	Pipeline that created it

6. SharedPipeline

Field name	datatype	Constraints	description
id	INT	Primary Key, Auto Increment	Unique share ID
pipeline_id	INT	Foreign Key → Pipeline	The shared pipeline
user_id	INT	Foreign Key → User	User shared with
role	VARCHAR(20)	Default 'viewer'	Access permission level

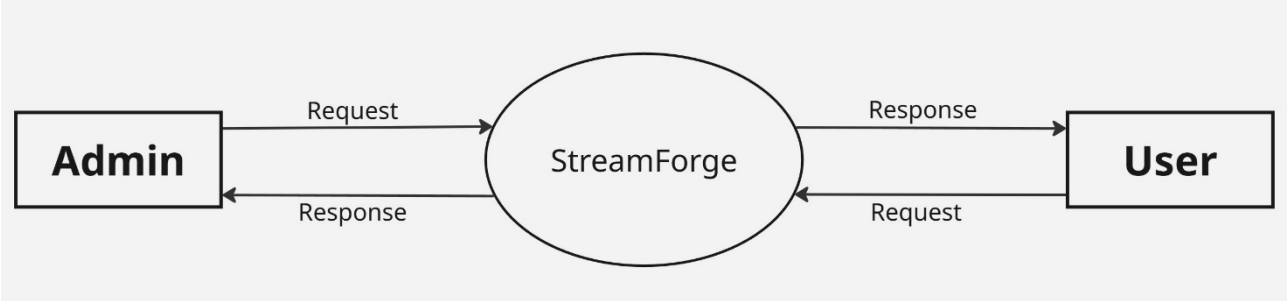
7. Announcements

Field name	datatype	Constraints	description
id	INT	Primary Key, Auto Increment	Unique notification ID
user_id	INT	Not Null, Foreign Key → User	Recipient user
message	VARCHAR(500)	Not Null	Notification content
type	VARCHAR(20)	Not Null	Type (success, error, info)
read	BOOLEAN	Default False	Read/Unread status
timestamp	DATETIME	Default current_timestamp	Time of notification

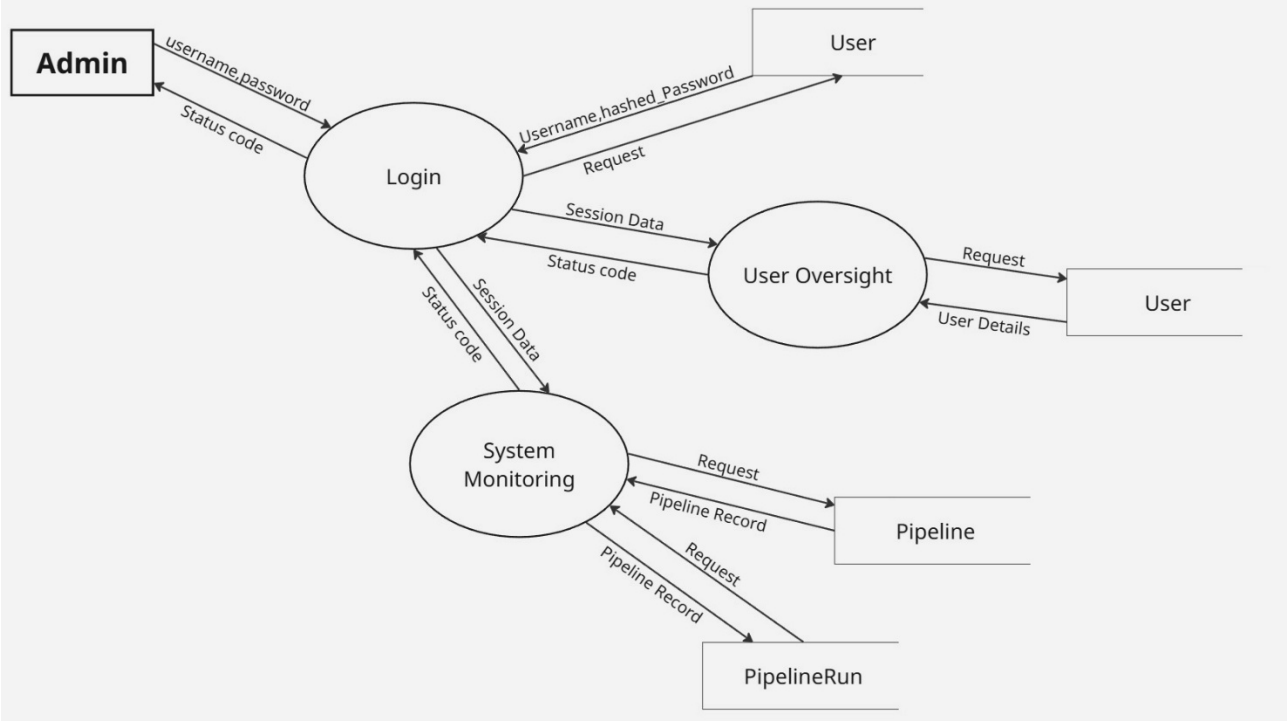
12.2. **APPENDIX B** (DFD)

12.2. APPENDIX B (DFD)

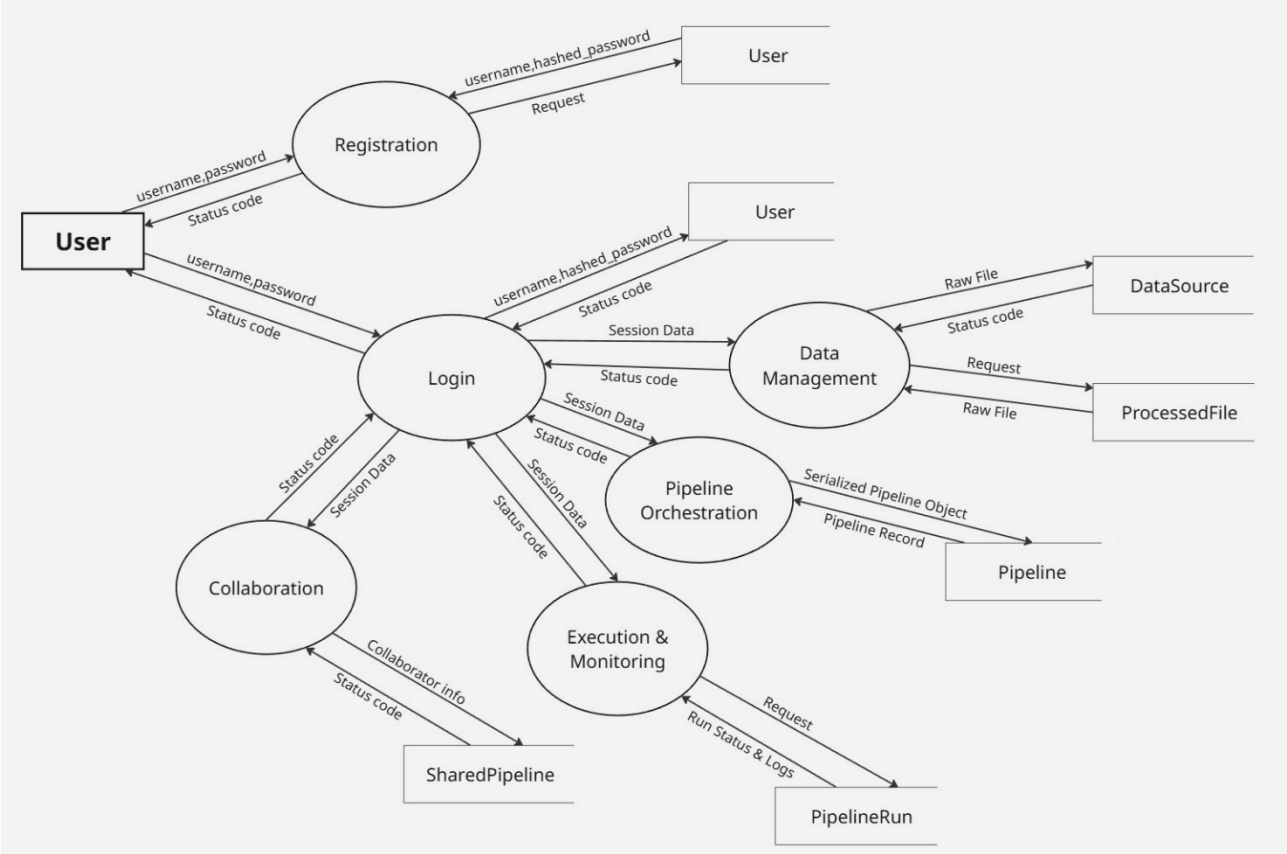
Context level DFD



Level 1 DFD for Admin



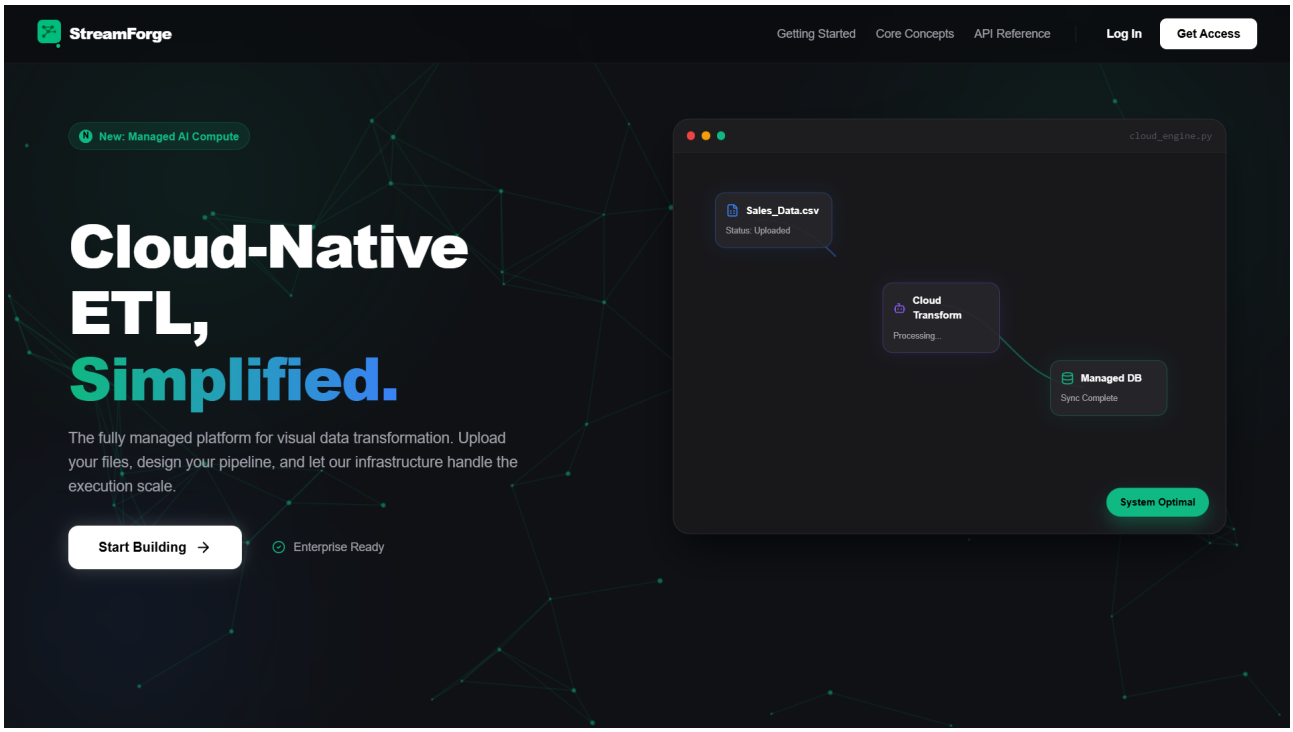
Level 1 DFD for User



12.3. APPENDIX C (INPUT AND OUTPUT FORMS)

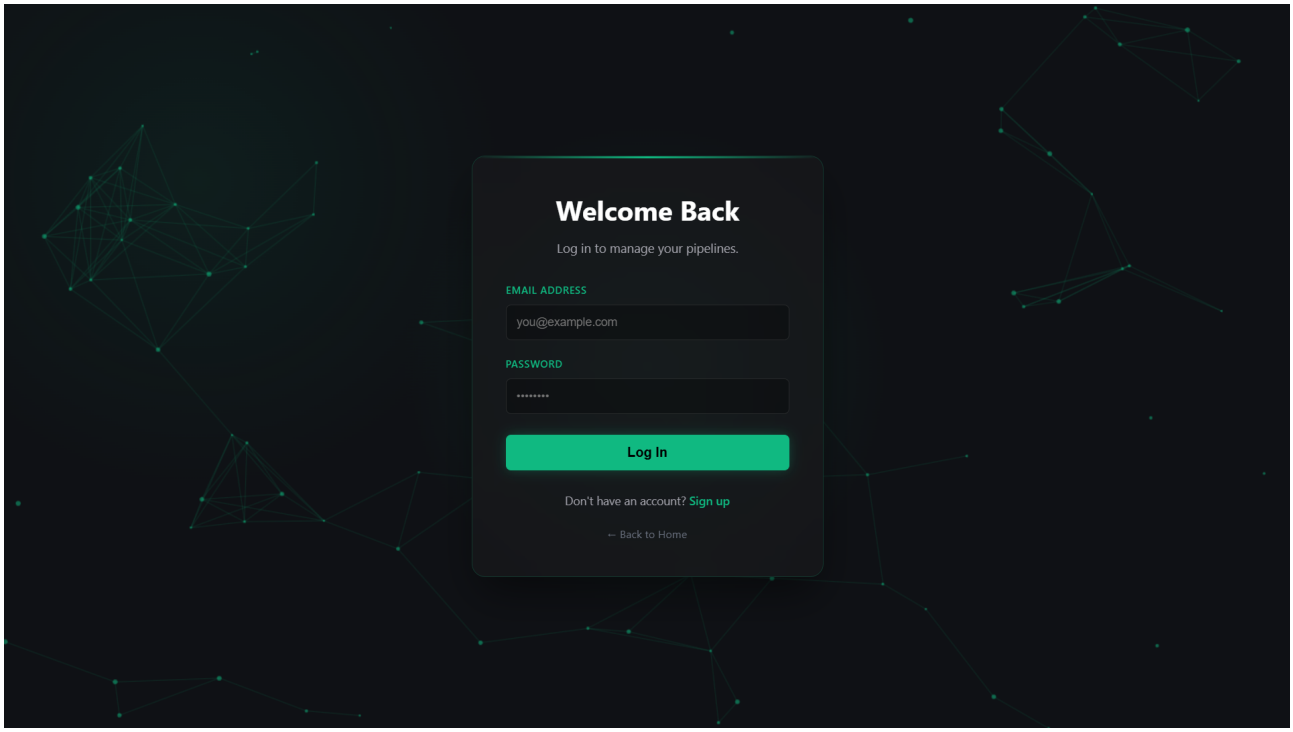
12.3. APPENDIX C (INPUT AND OUTPUT FORMS)

Index Page

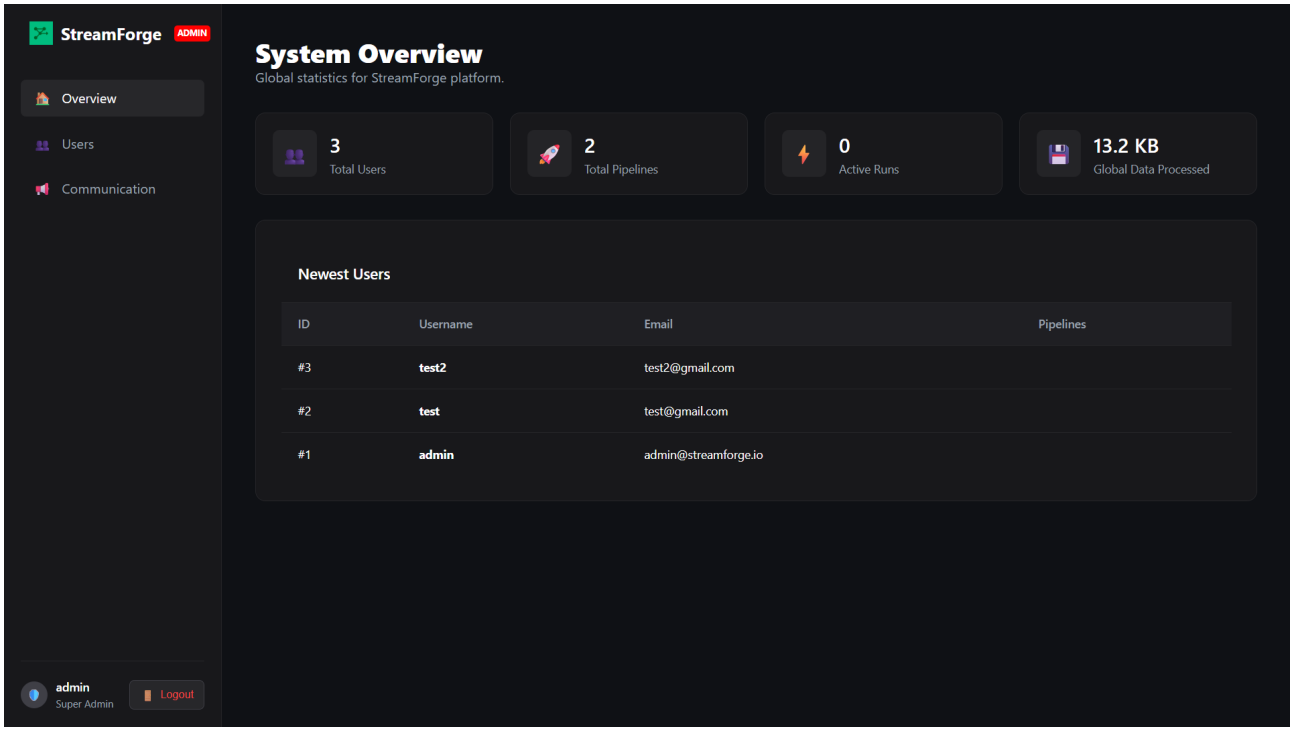


ADMIN

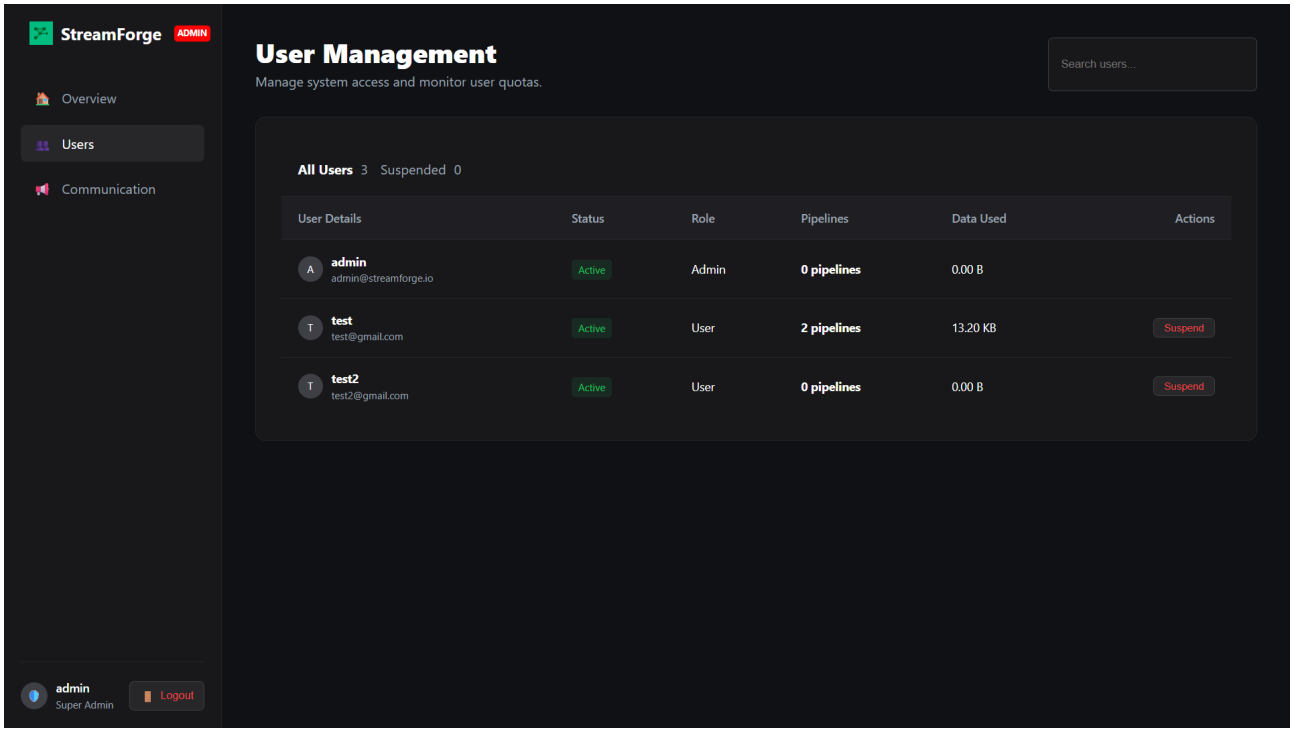
Admin Login



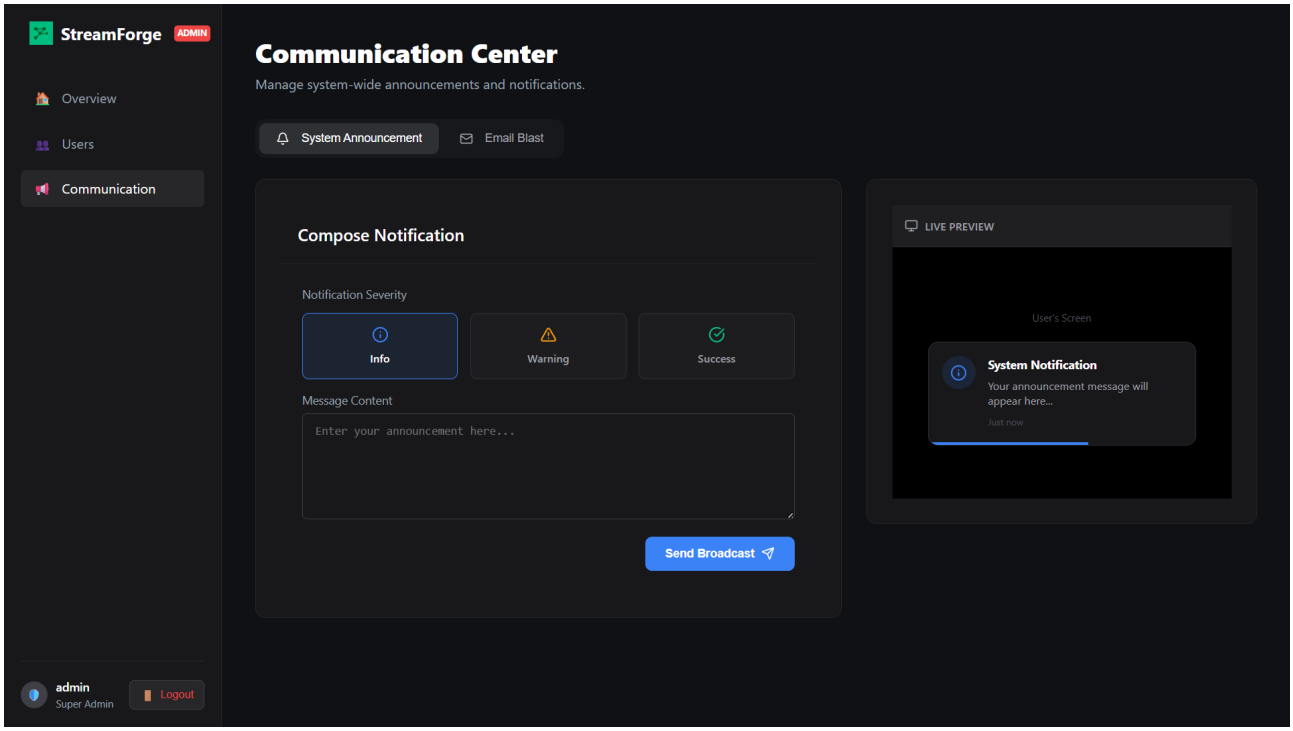
Dashboard



User Management

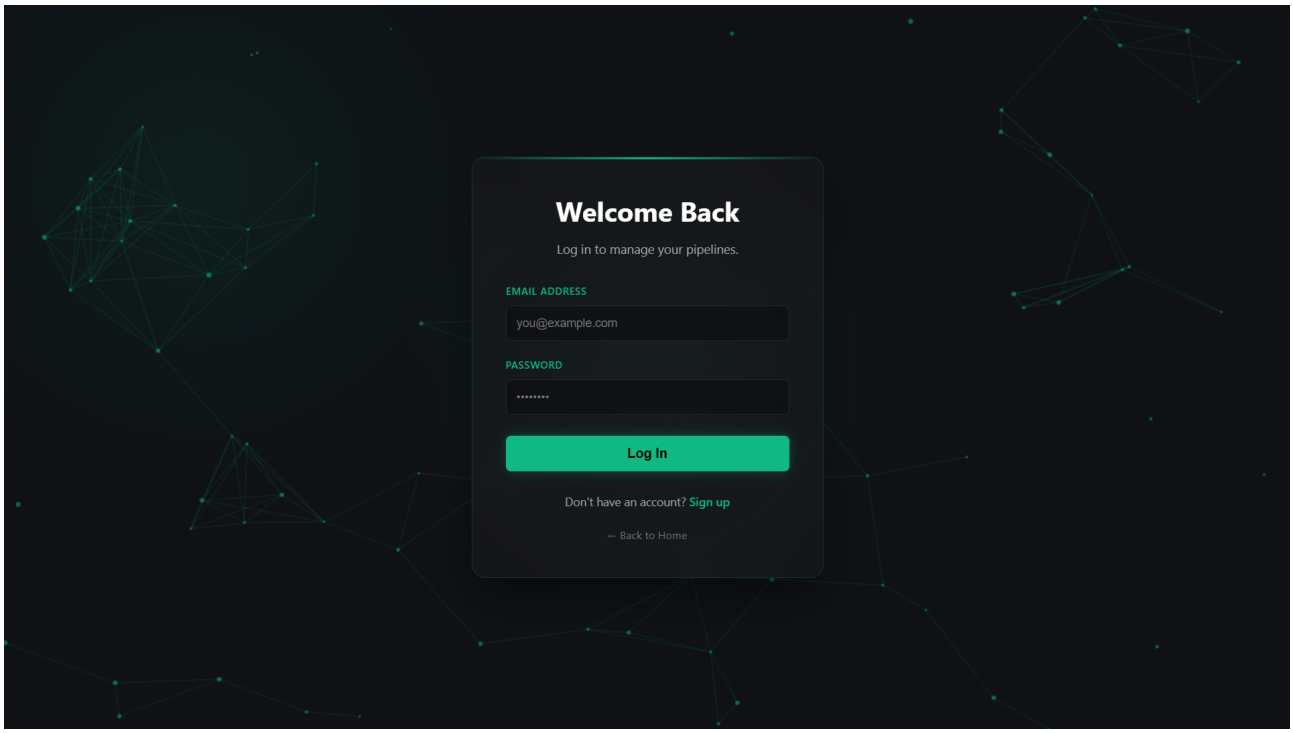


Communication Center

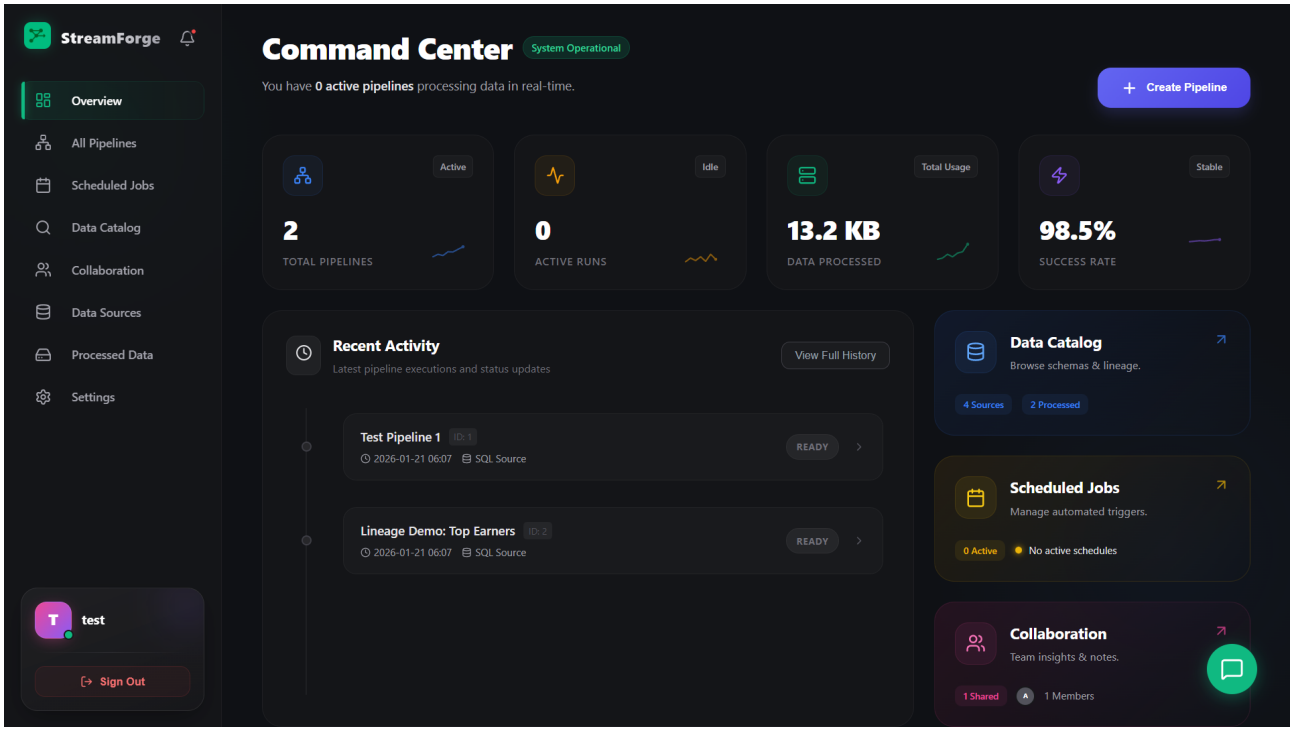


USER

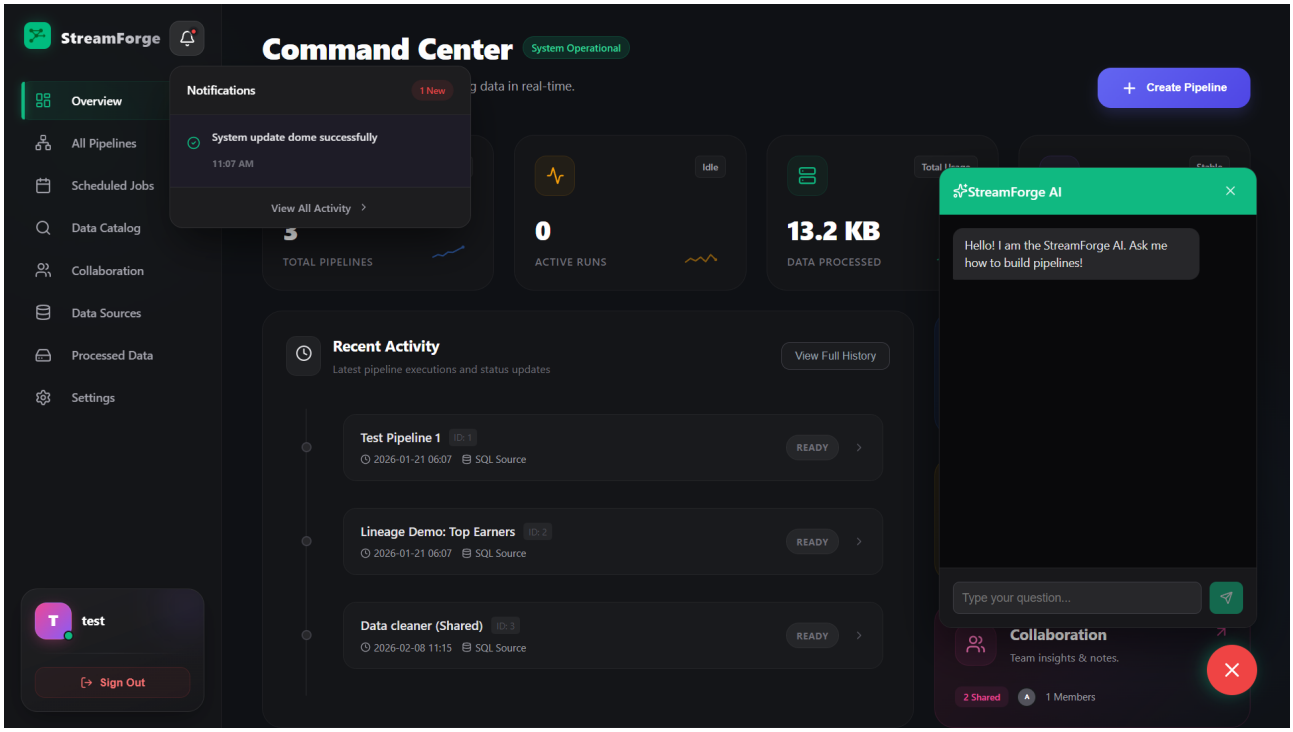
User Login



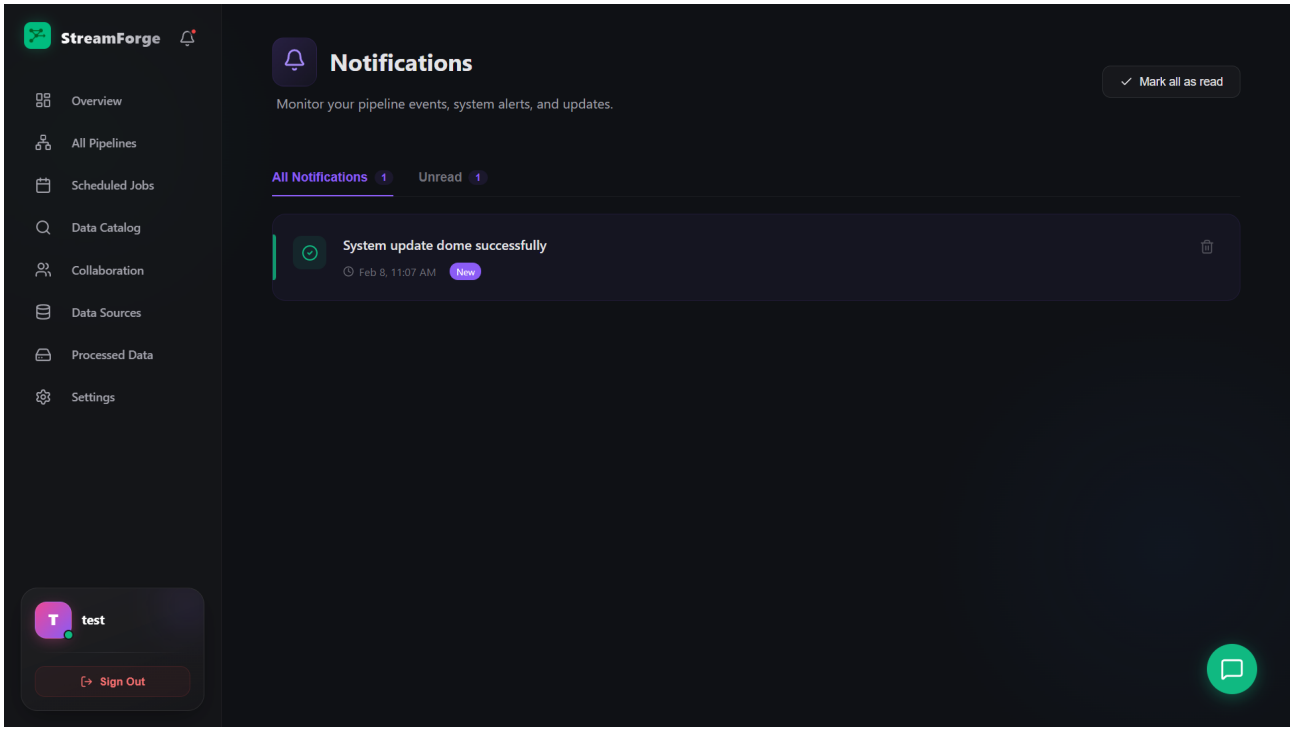
Dashboard



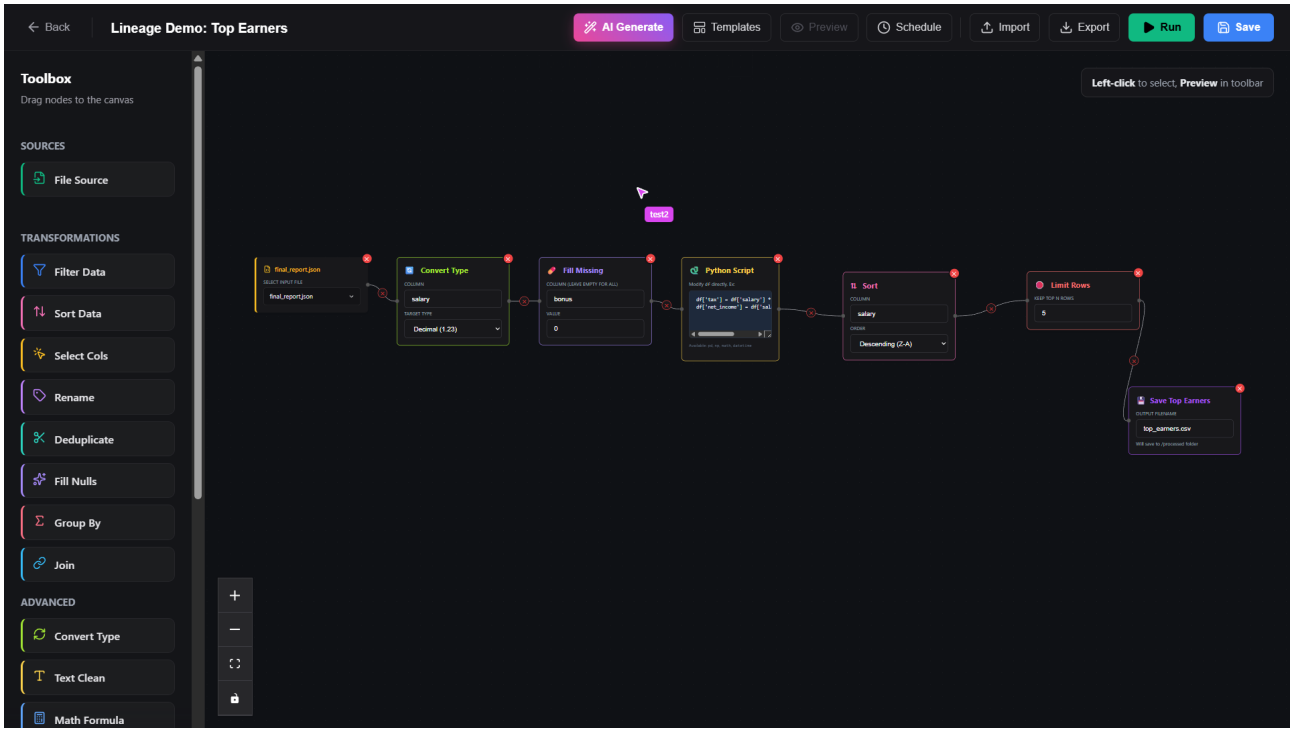
Notification and AI Assistant



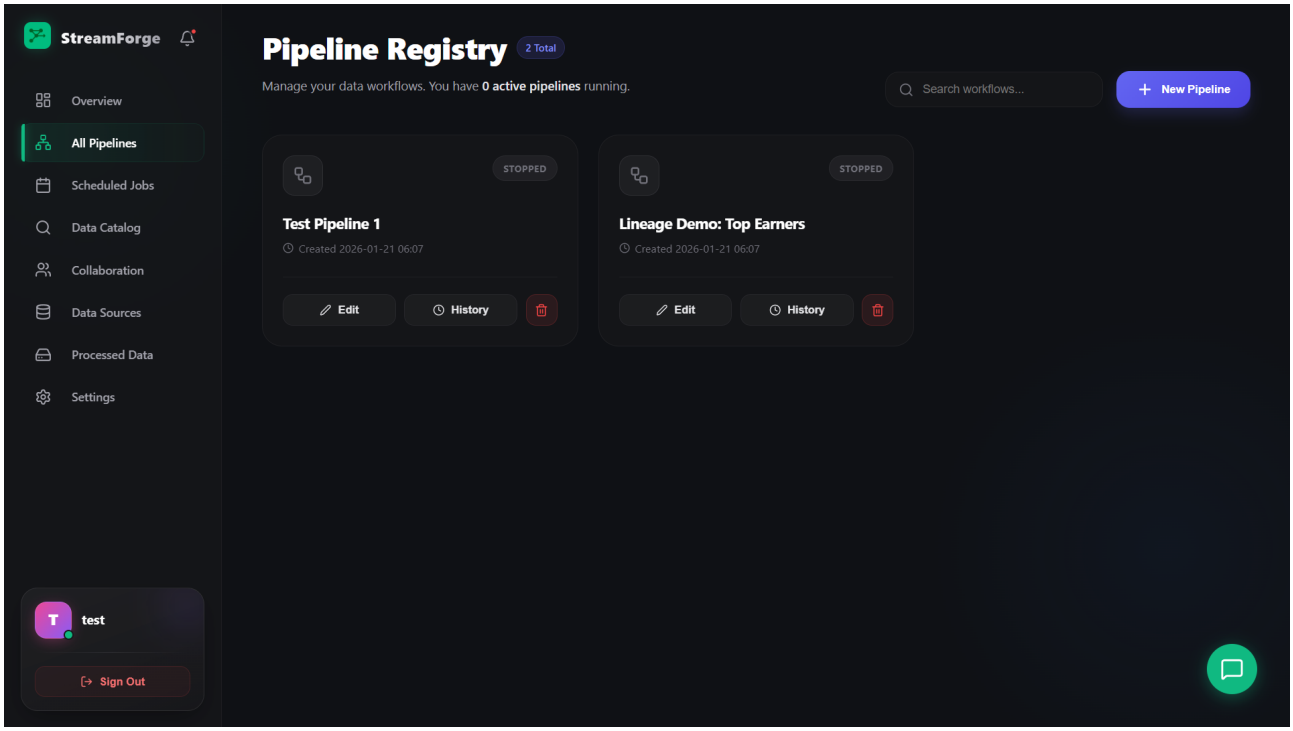
Notifications



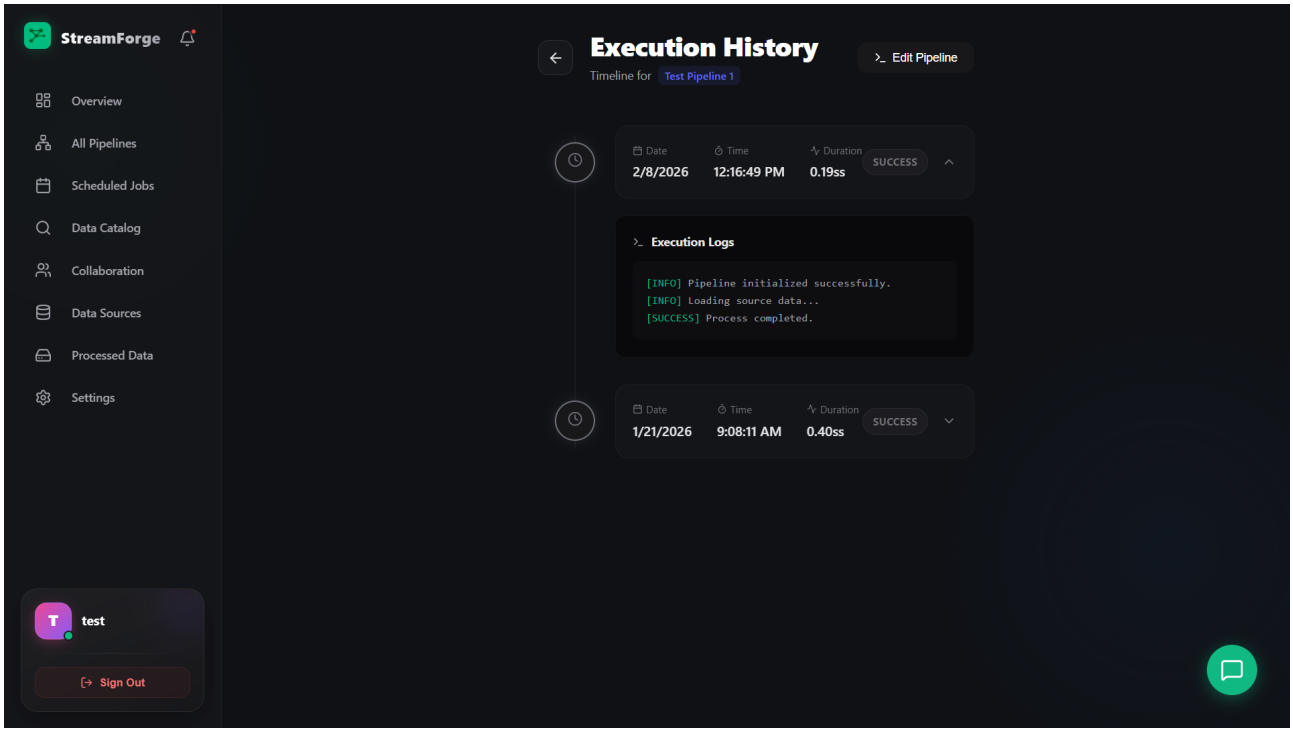
Pipelines Builder



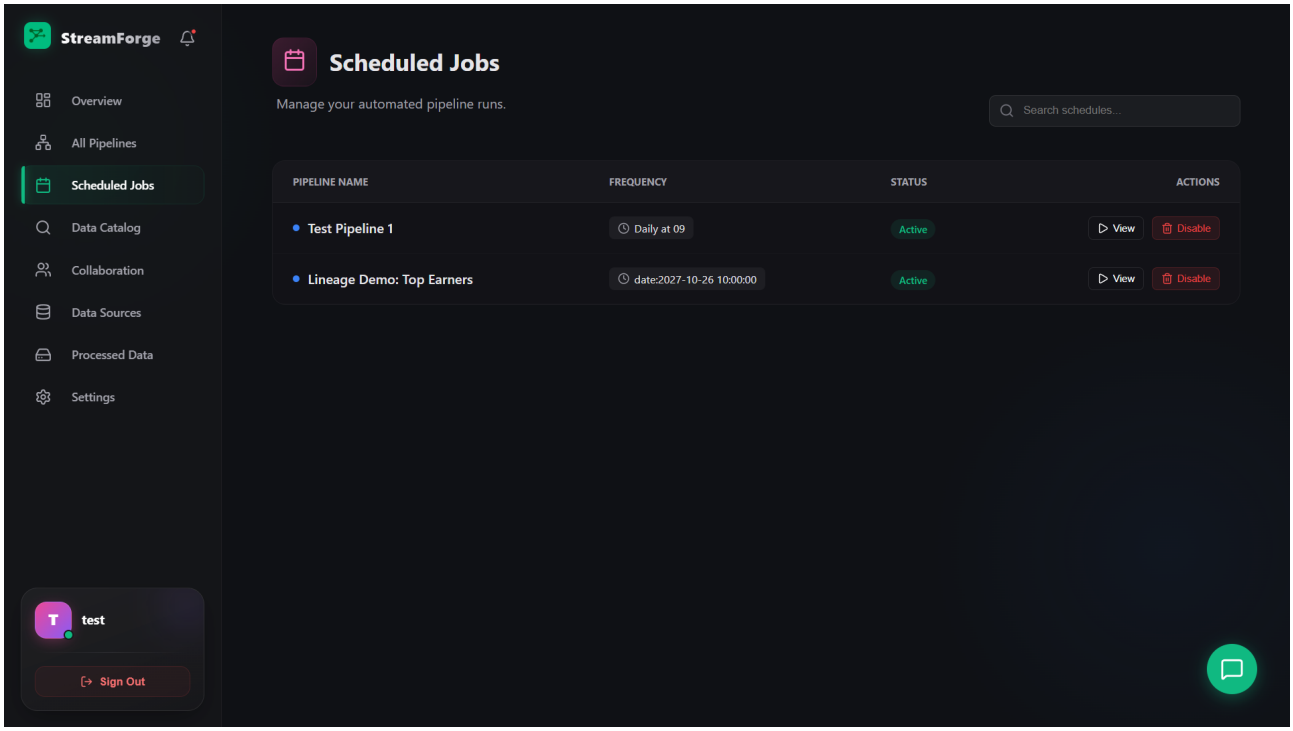
All Pipelines



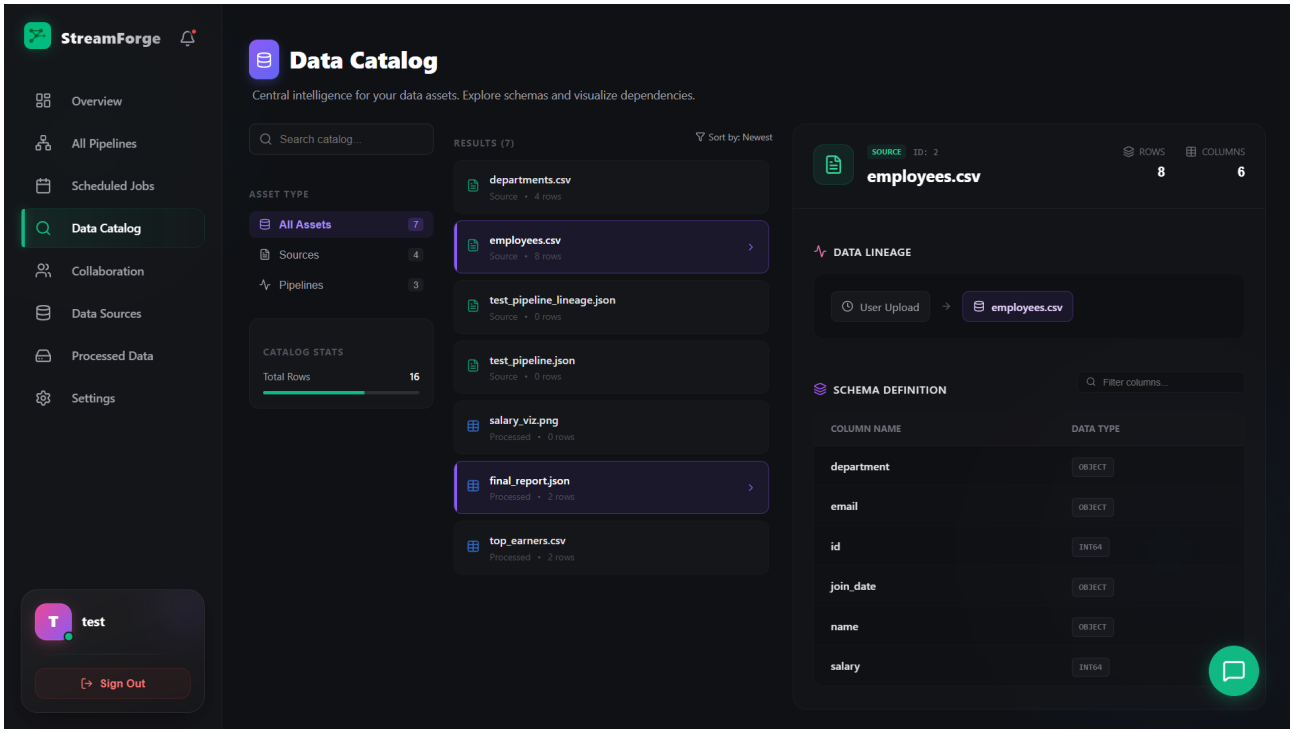
Execution History



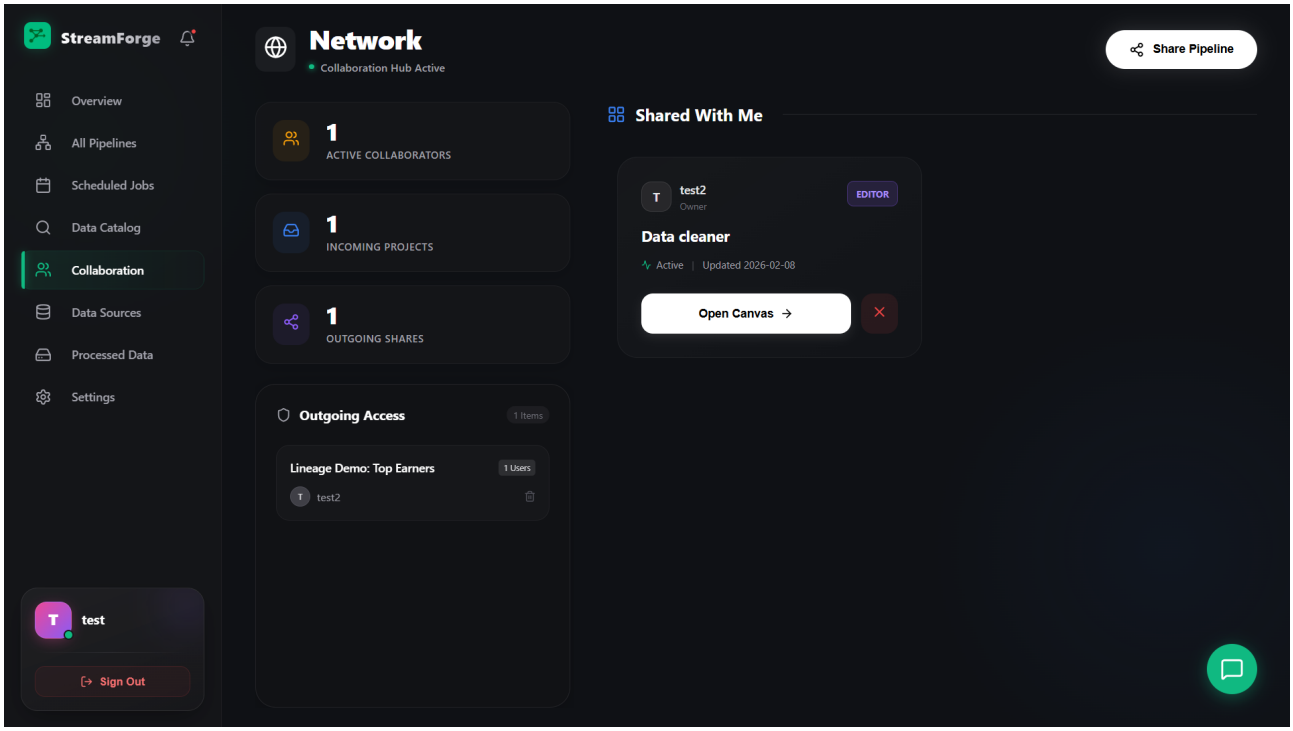
Scheduled Jobs



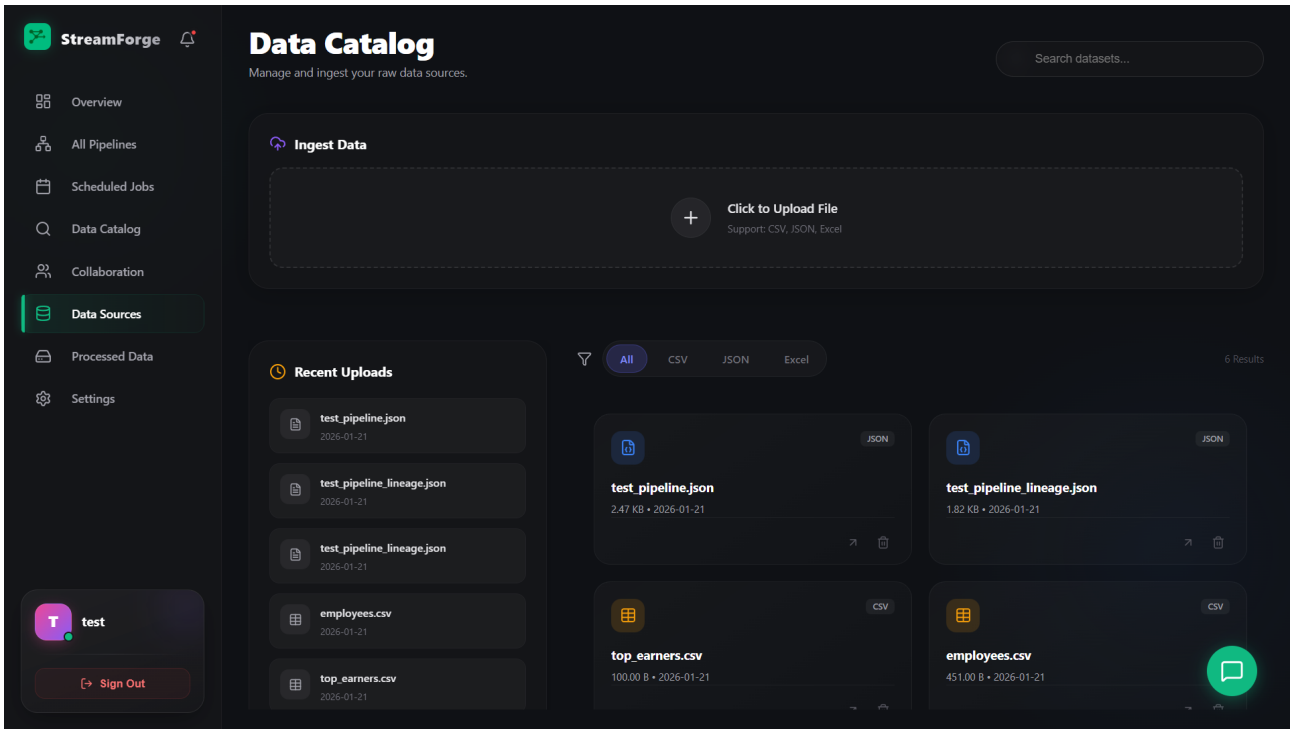
Data Catalog



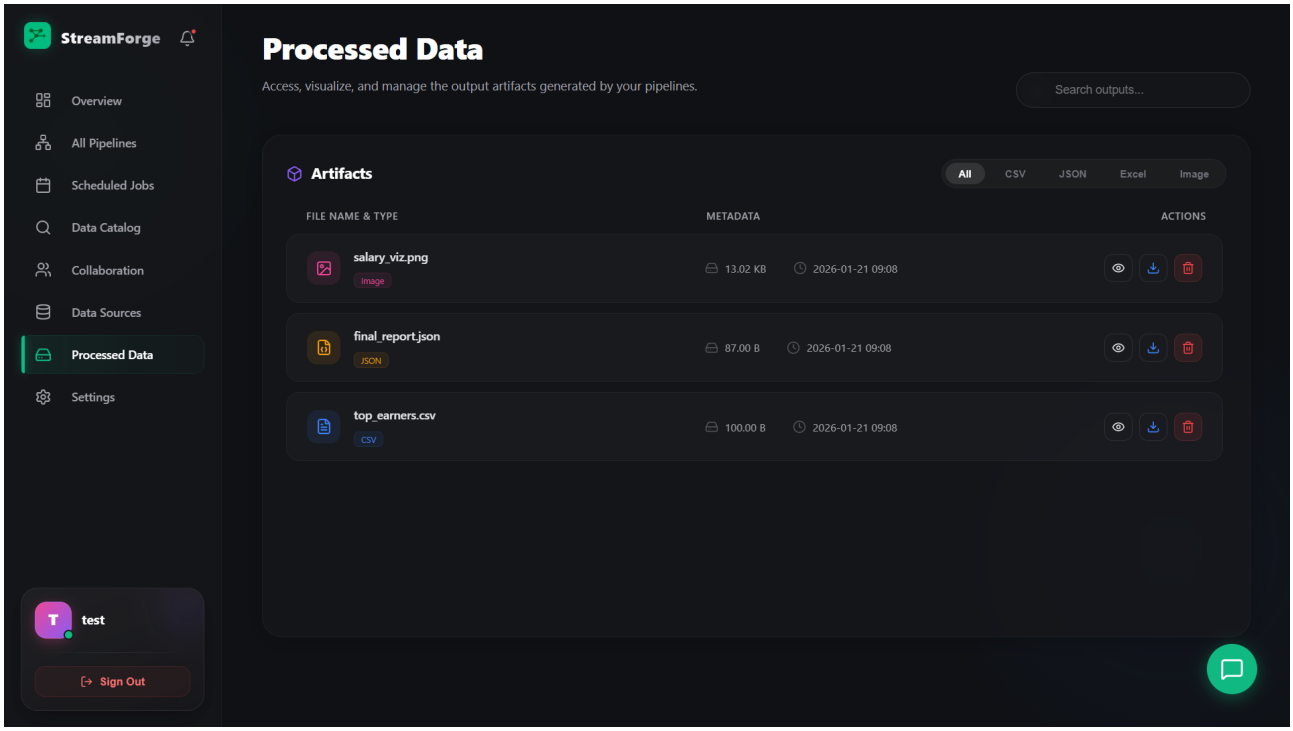
Collaboration



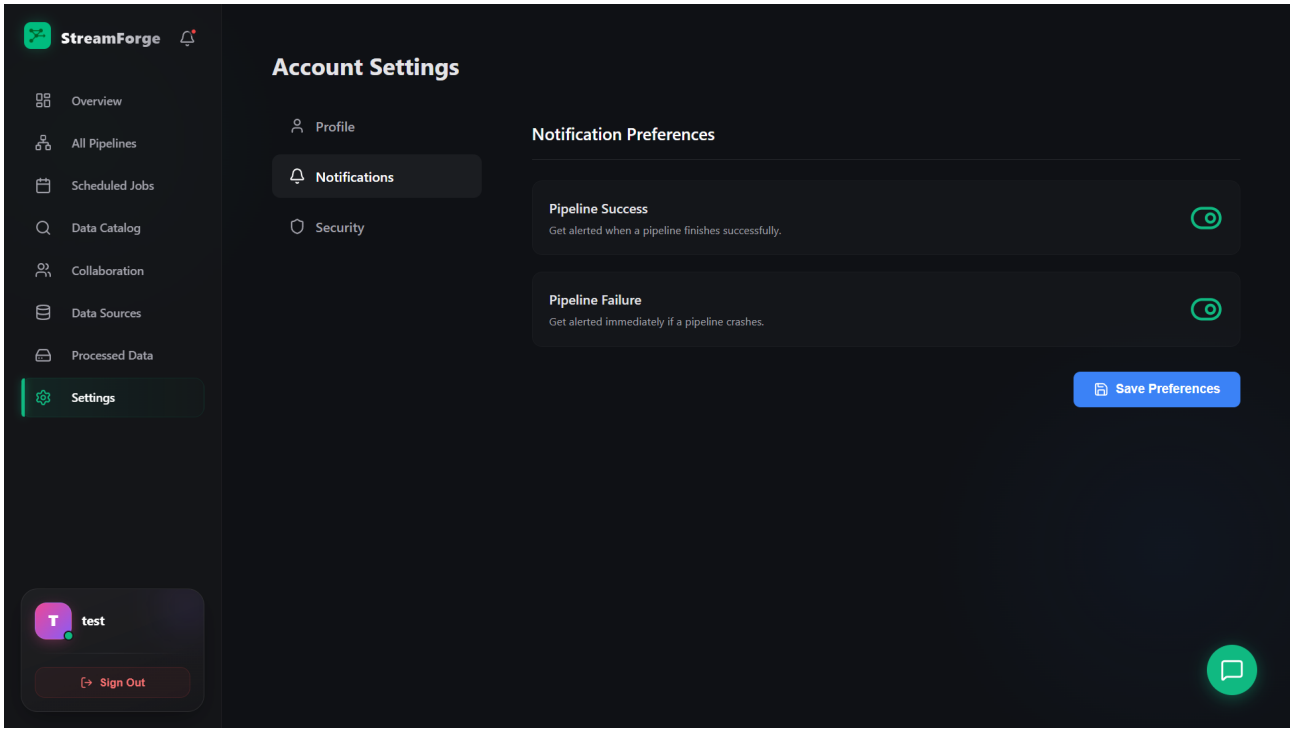
Data Sources



Processed Data



Settings



12.4. APPENDIX D (CODING)

12.4. APPENDIX D (CODING)

Admin

1. AdminDashboard.jsx

```
import React, { useEffect, useState } from 'react';
import { useNavigate } from 'react-router-dom';
import axios from 'axios';
import logo from '../assets/logo.png';
import '../App.css';
const formatBytes = (bytes, decimals = 2) => {
    if (!+bytes) return '0 B';
    const k = 1024;
    const dm = decimals < 0 ? 0 : decimals;
    const sizes = ['B', 'KB', 'MB', 'GB', 'TB', 'PB'];
    const i = Math.floor(Math.log(bytes) / Math.log(k));
    return `${parseFloat((bytes / Math.pow(k, i)).toFixed(dm))} ${sizes[i]}`;
}
const AdminDashboard = () => {
  const navigate = useNavigate();
  const [stats, setStats] = useState({
    total_users: 0,
    total_pipelines: 0,
    active_pipelines: 0,
    total_processed_bytes: 0,
    recent_users: []
  });
  const [user, setUser] = useState({ username: 'Admin' });
  useEffect(() => {
    const storedUser = localStorage.getItem('user');
    const token = localStorage.getItem('token');
```

```
  if (storedUser) {
    const parsedUser = JSON.parse(storedUser);
    if (!parsedUser.is_admin) {
      navigate('/dashboard'); // Kick non-admins out
      return;
    }
    setUser(parsedUser);
  } else {
    navigate('/login');
    return;
  }
  if (token) {
    axios.get('http://127.0.0.1:5000/admin/stats', {
      headers: { Authorization: `Bearer ${token}` }
    })
    .then(res => setStats(res.data))
    .catch(err => console.error("Error fetching admin stats:", err));
  }
 }, [navigate]);
 const handleLogout = () => {
  localStorage.clear();
  navigate('/login');
 };
 return (
  <div className="app-container">
   <aside className="sidebar">
    <div className="sidebar-logo">
     <img src={logo} alt="Logo" style={{ width: '28px', height: '28px' }} />
     StreamForge <span style={{fontSize:'10px', background:'red', padding:'2px 4px',
borderRadius:'4px', marginLeft:'5px'}}>ADMIN</span>
    </div>
```

```
<nav className="sidebar-nav">
  <div className="sidebar-item active"><span>🏠</span> Overview</div>
  <div className="sidebar-item" onClick={() => navigate('/admin/users')}><span>👥</span> Users</div>
  <div className="sidebar-item" onClick={() => navigate('/admin/communication')}><span>📢</span> Communication</div>
</nav>
<div className="sidebar-profile">
  <div className="flex items-center gap-10">
    <div className="profile-avatar">♡ </div>
    <div>
      <p style={{ margin: 0, fontSize: '14px', fontWeight: '500', color: 'white' }}>{user.username}</p>
      <p className="text-muted" style={{ margin: 0, fontSize: '12px' }}>Super Admin</p>
    </div>
  </div>
  <button onClick={handleLogout} className="btn-sidebar-logout"><span>🚪</span> Logout</button>
</div>
</aside>
<main className="main-content">
<div className="content-wrapper">
  <div style={{ marginBottom: '30px' }}>
    <h1 style={{ fontSize: '32px', marginBottom: '10px', margin: 0 }}>System Overview</h1>
    <p className="text-muted" style={{ margin: 0 }}>Global statistics for StreamForge platform.</p>
  </div>
  <div className="flex gap-20" style={{ marginBottom: '30px' }}>
    <StatCard label="Total Users" value={stats.total_users} icon="👥" />
    <StatCard label="Total Pipelines" value={stats.total_pipelines} icon="🚀" />
    <StatCard label="Active Runs" value={stats.active_pipelines} icon="⚡" />
```

```
          <StatCard label="Global Data Processed" value={formatBytes(stats.total_processed_bytes)}
icon="💾" />
        </div>
        <div className="card">
          <div style={{ padding: '20px', borderBottom: '1px solid var(--border-color)' }}>
            <h3 style={{ margin: 0, fontSize: '18px' }}>Newest Users</h3>
          </div>
          <table className="data-table">
            <thead>
              <tr>
                <th>ID</th>
                <th>Username</th>
                <th>Email</th>
                <th>Pipelines</th>
              </tr>
            </thead>
            <tbody>
              {stats.recent_users.map(u => (
                <tr key={u.id}>
                  <td>#{u.id}</td>
                  <td className="font-bold">{u.username}</td>
                  <td className="text-muted">{u.email}</td>
                  <td>{u.pipelines}</td>
                </tr>
              ))}
            </tbody>
          </table>
        </div>
      </div>
    </main>
  </div>
```

```
  );
};
const StatCard = ({ label, value, icon }) => (
  <div className="card" style={{ flex: 1, padding: '20px', display: 'flex', alignItems: 'center', gap: '15px' }}>
    <div style={{ fontSize: '24px', background: 'rgba(255,255,255,0.05)', padding: '10px', borderRadius: '8px' }}>
      {icon}
    </div>
    <div>
      <h3 style={{ margin: 0, fontSize: '24px' }}>{value}</h3>
      <p className="text-muted" style={{ margin: 0, fontSize: '14px' }}>{label}</p>
    </div>
  </div>
);
export default AdminDashboard;
```

2. AdminUsers.jsx

```
import React, { useEffect, useState } from 'react';
import { useNavigate } from 'react-router-dom';
import axios from 'axios';
import logo from '../assets/logo.png';
import '../App.css';
const AdminUsers = () => {
  const navigate = useNavigate();
  const [users, setUsers] = useState([]);
  const [user, setUser] = useState({ username: 'Admin' });
  const [searchTerm, setSearchTerm] = useState('');
  useEffect(() => {
    const storedUser = localStorage.getItem('user');
```

```
  const token = localStorage.getItem('token');
  if (storedUser) {
     const parsedUser = JSON.parse(storedUser);
     if (!parsedUser.is_admin) {
        navigate('/dashboard');
        return;
     }
     setUser(parsedUser);
  } else {
     navigate('/login');
     return;
  }
  if (token) {
     fetchUsers(token);
  }
}, [navigate]);
const fetchUsers = async (token) => {
   try {
      const res = await axios.get('http://127.0.0.1:5000/admin/users', {
         headers: { Authorization: `Bearer ${token}` }
      });
      setUsers(res.data);
   } catch (err) {
      console.error("Failed to fetch users", err);
   }
};
const toggleSuspend = async (userId, currentStatus) => {
   const action = currentStatus ? "ACTIVATE" : "SUSPEND";
   if (!window.confirm(`Are you sure you want to ${action} this user?`)) return;
   const token = localStorage.getItem('token');
```

```
    try {
      await axios.put(`http://127.0.0.1:5000/admin/users/${userId}/suspend`,
        { is_suspended: !currentStatus },
        { headers: { Authorization: `Bearer ${token}` } }
      );
      setUsers(users.map(u =>
        u.id === userId ? { ...u, is_suspended: !currentStatus } : u
      ));
    } catch (err) {
      alert("Action failed: " + (err.response?.data?.error || err.message));
    }
  };
  const handleLogout = () => {
   localStorage.clear();
   navigate('/login');
  };
  const filteredUsers = users.filter(u =>
    u.username.toLowerCase().includes(searchTerm.toLowerCase()) ||
    u.email.toLowerCase().includes(searchTerm.toLowerCase())
  );
  return (
   <div className="app-container">
    <aside className="sidebar">
     <div className="sidebar-logo">
      <img src={logo} alt="Logo" style={{ width: '28px', height: '28px' }} />
      StreamForge <span style={{fontSize:'10px', background:'red', padding:'2px 4px',
borderRadius:'4px', marginLeft:'5px'}}>ADMIN</span>
     </div>
     <nav className="sidebar-nav">
      <div className="sidebar-item" onClick={() => navigate('/admin')}><span>🏠</span>
Overview</div>
```

```
        <div className="sidebar-item active"><span> 👥 </span> Users</div>
        <div className="sidebar-item" onClick={() => navigate('/admin/communication')}><span>
📢 </span> Communication</div>
      </nav>
      <div className="sidebar-profile">
        <div className="flex items-center gap-10">
          <div className="profile-avatar">♡ </div>
          <div>
            <p style={{ margin: 0, fontSize: '14px', fontWeight: '500', color: 'white'
}}>{user.username}</p>
            <p className="text-muted" style={{ margin: 0, fontSize: '12px' }}>Super Admin</p>
          </div>
        </div>
        <button onClick={handleLogout} className="btn-sidebar-logout"><span> 🔲 </span>
Logout</button>
      </div>
    </aside>
    <main className="main-content">
      <div className="content-wrapper">
        <div className="flex justify-between items-end" style={{ marginBottom: '30px' }}>
          <div>
            <h1 style={{ fontSize: '32px', marginBottom: '10px', margin: 0 }}>User
Management</h1>
            <p className="text-muted" style={{ margin: '5px 0 0 0' }}>Manage system access and
monitor user quotas.</p>
          </div>
          <div style={{ display: 'flex', gap: '10px' }}>
            <input
              className="search-input"
              type="text"
              placeholder="Search users..."
              value={searchTerm}
```

```
            onChange={(e) => setSearchTerm(e.target.value)}
          />
        </div>
      </div>
      <div className="card">
        <div style={{ padding: '20px', borderBottom: '1px solid var(--border-color)', display:'flex', gap:'15px' }}>
          <span className="text-white font-bold">All Users <span className="text-muted" style={{fontWeight:'normal', marginLeft:'5px'}}>{users.length}</span></span>
          <span className="text-muted">Suspended <span className="text-white" style={{marginLeft:'5px'}}>{users.filter(u => u.is_suspended).length}</span></span>
        </div>
        <table className="data-table">
          <thead>
            <tr>
              <th>User Details</th>
              <th>Status</th>
              <th>Role</th>
              <th>Pipelines</th>
              <th>Data Used</th>
              <th style={{ textAlign: 'right' }}>Actions</th>
            </tr>
          </thead>
          <tbody>
            {filteredUsers.map(u => (
              <tr key={u.id}>
                <td>
                  <div className="flex items-center gap-10">
                    <div className="profile-avatar" style={{width:'30px', height:'30px', fontSize:'12px'}}>
                      {u.username.charAt(0).toUpperCase()}
                    </div>
```

```
                    <div>
                      <div className="font-bold">{u.username}</div>
                      <div className="text-muted" style={{fontSize:'12px'}}>{u.email}</div>
                    </div>
                  </div>
                </td>
                <td>
                  <span
                    className="status-badge"
                    style={{
                      background: u.is_suspended ? 'rgba(239, 68, 68, 0.1)' : 'rgba(34, 197, 94,
0.1)',
                      color: u.is_suspended ? '#ef4444' : '#22c55e'
                    }}
                  >
                    {u.is_suspended ? 'Suspended' : 'Active'}
                  </span>
                </td>
                <td className="text-muted">{u.is_admin ? 'Admin' : 'User'}</td>
                <td className="text-muted font-bold">{u.pipelines_count} pipelines</td>
                <td className="text-muted">{u.processed_bytes}</td>
                <td style={{ textAlign: 'right' }}>
                  {!u.is_admin && (
                    <button
                      className={`btn ${u.is_suspended ? 'btn-success' : 'btn-ghost-danger'}`}
                      style={{ padding: '4px 12px', fontSize: '12px' }}
                      onClick={() => toggleSuspend(u.id, u.is_suspended)} >
                      {u.is_suspended ? 'Activate' : 'Suspend'}
                    </button>
                  )}
                </td>
```

```
                </tr>
              ))}
            </tbody>
          </table>
        </div>
      </div>
    </main>
  </div>
 );
};
export default AdminUsers;
```

USER

1. UserDashboard.jsx

```
import React, { useEffect, useState } from 'react';
import { useNavigate } from 'react-router-dom';
import axios from 'axios';
import { motion } from 'framer-motion';
import {
  Network, Activity, Server, Plus, Zap, ArrowUpRight, Clock,
  MoreHorizontal, Database, Calendar, Users, ChevronRight
} from 'lucide-react';
import AppLayout from './layout/AppLayout';
import '../App.css';
const formatBytes = (bytes, decimals = 2) => {
  if (!+bytes) return '0 B';
  const k = 1024;
  const dm = decimals < 0 ? 0 : decimals;
  const sizes = ['B', 'KB', 'MB', 'GB', 'TB'];
```

```
    const i = Math.floor(Math.log(bytes) / Math.log(k));
    return `${parseFloat((bytes / Math.pow(k, i)).toFixed(dm))} ${sizes[i]}`;
}
const Dashboard = () => {
  const navigate = useNavigate();
  const [pipelines, setPipelines] = useState([]);
  const [recentPipelines, setRecentPipelines] = useState([]);
  const [totalDataSize, setTotalDataSize] = useState('0 B');
  const [activeRuns, setActiveRuns] = useState(0);
  const [loading, setLoading] = useState(true);
  const [catalogStats, setCatalogStats] = useState({ sources: 0, processed: 0 });
  const [scheduledStats, setScheduledStats] = useState({ count: 0, next: null });
  const [collabStats, setCollabStats] = useState({ members: 0, shared: 0 });
  useEffect(() => {
    const token = localStorage.getItem('token');
    if (token) {
      setLoading(true);
      const headers = { Authorization: `Bearer ${token}` };
      const reqPipelines = axios.get('http://127.0.0.1:5000/pipelines', { headers });
      const reqUserStats = axios.get('http://127.0.0.1:5000/user-stats', { headers });
      const reqDataSources = axios.get('http://127.0.0.1:5000/datasources', { headers });
      const reqCollab = axios.get('http://127.0.0.1:5000/collaboration/stats', { headers });
      Promise.all([reqPipelines, reqUserStats, reqDataSources, reqCollab])
      .then(([resPipelines, resStats, resData, resCollab]) => {
        const allPipelines = resPipelines.data;
        setPipelines(allPipelines);
        setRecentPipelines(allPipelines.slice(0, 6));
        setActiveRuns(allPipelines.filter(p => p.status === 'Active').length);
        const scheduled = allPipelines.filter(p => p.schedule);
        setScheduledStats({
```

```
            count: scheduled.length,
            next: scheduled.length > 0 ? scheduled[0].name : null
          });
          setTotalDataSize(formatBytes(resStats.data.total_processed_bytes));
          const sources = resData.data.filter(d => d.category === 'upload').length;
          const processed = resData.data.filter(d => d.category === 'processed').length;
          setCatalogStats({ sources, processed });
          setCollabStats({
            members: resCollab.data.team_members || 1, // Default to 1 (self) if 0
            shared: resCollab.data.shared_with_me + resCollab.data.my_shared_pipelines
          });
        })
        .catch(err => console.error("Error loading dashboard:", err))
        .finally(() => setLoading(false));
    }
  }, []);
  const containerVariants = {
    hidden: { opacity: 0 },
    visible: { opacity: 1, transition: { staggerChildren: 0.08 } }
  };
  const itemVariants = {
    hidden: { y: 15, opacity: 0 },
    visible: { y: 0, opacity: 1, transition: { type: "spring", stiffness: 300, damping: 24 } }
  };
  return (
    <AppLayout>
      <style>
        {`
          .main-content::-webkit-scrollbar {
            display: none;
```

```
            }
            .main-content {
               -ms-overflow-style: none;
               scrollbar-width: none;
            }
          `}
      </style>
      <div style={{ padding: '32px 48px', maxWidth: '1800px', margin: '0 auto', width: '100%', boxSizing: 'border-box' }}>
         <motion.div variants={containerVariants} initial="hidden" animate="visible">
            <div style={{ display: 'flex', justifyContent: 'space-between', alignItems: 'flex-end', marginBottom: '32px' }}>
               <div>
                  <div style={{ display: 'flex', alignItems: 'center', gap: '12px', marginBottom: '8px' }}>
                     <h1 style={{ fontSize: '36px', fontWeight: '800', color: 'white', margin: 0, letterSpacing: '-0.5px' }}>
                        Command Center
                     </h1>
                     <span style={{ fontSize: '12px', padding: '4px 10px', background: 'rgba(16, 185, 129, 0.15)', color: '#34d399', borderRadius: '20px', border: '1px solid rgba(16, 185, 129, 0.2)', fontWeight: '600' }}>
                        System Operational
                     </span>
                  </div>
                  <p style={{ color: '#a1a1aa', fontSize: '15px', maxWidth: '600px', lineHeight: '1.5' }}>
                     You have <span style={{ color: '#e4e4e7', fontWeight: '600' }}>{activeRuns} active pipelines</span> processing data in real-time.
                  </p>
               </div>
               <motion.button
                  whileHover={{ scale: 1.02, boxShadow: '0 0 20px rgba(99, 102, 241, 0.4)' }}
```

```
            whileTap={{ scale: 0.98 }}
            onClick={() => navigate('/builder')}
            style={{
              background: 'linear-gradient(135deg, #6366f1 0%, #4f46e5 100%)', color: 'white',
border: 'none',
              padding: '14px 28px', borderRadius: '16px', fontWeight: '600', cursor: 'pointer',
              display: 'flex', alignItems: 'center', gap: '10px', boxShadow: '0 8px 16px rgba(99,
102, 241, 0.2)'
            }}
          >
            <Plus size={20} strokeWidth={2.5} /> Create Pipeline
          </motion.button>
        </div>
        <div className="bento-grid" style={{
          display: 'grid',
          gridTemplateColumns: 'repeat(12, 1fr)',
          gridAutoRows: 'minmax(150px, auto)',
          gap: '24px'
        }}>
          <div style={{ gridColumn: 'span 3' }}>
            <StatCard
              label="Total Pipelines" value={pipelines.length}
              icon={<Network size={22} />} color="#3b82f6"
              trend={pipelines.length > 0 ? "Active" : "No pipelines"}
              chartData={[30, 40, 35, 50, 49, 60, 70]}
            />
          </div>
          <div style={{ gridColumn: 'span 3' }}>
            <StatCard
              label="Active Runs" value={activeRuns}
              icon={<Activity size={22} />} color="#f59e0b"
```

```
            trend={activeRuns > 0 ? "Processing" : "Idle"}
            chartData={[10, 25, 15, 30, 12, 40, 20]}
          />
        </div>
        <div style={{ gridColumn: 'span 3' }}>
          <StatCard
            label="Data Processed" value={totalDataSize}
            icon={<Server size={22} />} color="#10b981"
            trend="Total Usage"
            chartData={[20, 25, 40, 30, 45, 50, 80]}
          />
        </div>
        <div style={{ gridColumn: 'span 3' }}>
          <StatCard
            label="Success Rate" value="98.5%"
            icon={<Zap size={22} />} color="#8b5cf6"
            trend="Stable"
            chartData={[90, 92, 94, 93, 95, 98, 98.5]}
          />
        </div>
        <motion.div variants={itemVariants} style={{
          gridColumn: 'span 8', gridRow: 'span 3',
          background: 'rgba(24, 24, 27, 0.6)', borderRadius: '24px',
          border: '1px solid rgba(255,255,255,0.05)', padding: '28px',
          display: 'flex', flexDirection: 'column', position: 'relative', overflow: 'hidden'
        }}>
          <div style={{ display: 'flex', justifyContent: 'space-between', alignItems: 'center',
marginBottom: '28px' }}>
            <div style={{ display: 'flex', alignItems: 'center', gap: '14px' }}>
              <div style={{ padding: '10px', background: 'rgba(255,255,255,0.05)',
borderRadius: '12px', border: '1px solid rgba(255,255,255,0.05)' }}>
```

```
                        <Clock size={20} color="#e4e4e7" />
                      </div>
                      <div>
                        <h3 style={{ fontSize: '18px', fontWeight: '700', color: 'white', margin: 0
}}>Recent Activity</h3>
                        <p style={{ margin: '4px 0 0', fontSize: '13px', color: '#71717a' }}>Latest
pipeline executions and status updates</p>
                      </div>
                    </div>
                    <button className="btn-ghost" style={{ fontSize: '13px', padding: '8px 16px',
background: 'rgba(255,255,255,0.03)', borderRadius: '10px' }}>
                      View Full History
                    </button>
                  </div>
                  <div className="custom-scrollbar" style={{ flex: 1, overflowY: 'auto',
paddingRight: '8px', display: 'flex', flexDirection: 'column', position: 'relative' }}>
                    <div style={{ position: 'absolute', left: '23px', top: '10px', bottom: '10px', width:
'2px', background: 'rgba(255,255,255,0.05)', zIndex: 0 }} />
                    {loading ? (
                      Array.from({ length: 4 }).map((_, i) => <SkeletonRow key={i} />)
                    ) : recentPipelines.length === 0 ? (
                      <EmptyState />
                    ) : (
                      recentPipelines.map((pipe, i) => (
                        <PipelineTimelineRow key={pipe.id} pipe={pipe} index={i}
navigate={navigate} />
                      ))
                    )}
                  </div>
                </motion.div>
                <FunctionalCard
                  title="Data Catalog"
```

```
                description="Browse schemas & lineage."
                icon={<Database size={24} color="#60a5fa" />}
                color="#3b82f6"
                bgGradient="linear-gradient(135deg, rgba(37, 99, 235, 0.1), rgba(24, 24, 27, 0))"
                onClick={() => navigate('/catalog')}
                tags={[`${catalogStats.sources} Sources`, `${catalogStats.processed} Processed`]}
              />
              <FunctionalCard
                title="Scheduled Jobs"
                description="Manage automated triggers."
                icon={<Calendar size={24} color="#facc15" />}
                color="#eab308"
                bgGradient="linear-gradient(135deg, rgba(202, 138, 4, 0.1), rgba(24, 24, 27, 0))"
                onClick={() => navigate('/schedules')}
                alert={scheduledStats.next ? `Active: ${scheduledStats.next}` : "No active
schedules"}
                tags={[`${scheduledStats.count} Active`]}
              />
              <FunctionalCard
                title="Collaboration"
                description="Team insights & notes."
                icon={<Users size={24} color="#f472b6" />}
                color="#ec4899"
                bgGradient="linear-gradient(135deg, rgba(219, 39, 119, 0.1), rgba(24, 24, 27, 0))"
                onClick={() => navigate('/collaboration')}
                activeUsers={collabStats.members}
                tags={[`${collabStats.shared} Shared`]}
              />
            </div>
          </motion.div>
        </div>
```

```
    </AppLayout>
  );
};
const FunctionalCard = ({ title, description, icon, color, bgGradient, onClick, tags, alert,
activeUsers }) => (
    <motion.div
      variants={{ hidden: { opacity: 0, x: 20 }, visible: { opacity: 1, x: 0 } }}
      onClick={onClick}
      whileHover={{ scale: 1.01, y: -2, borderColor: `${color}60` }}
      style={{
        gridColumn: 'span 4', gridRow: 'span 1',
        background: bgGradient || 'rgba(24, 24, 27, 0.6)',
        borderRadius: '24px', padding: '24px',
        border: `1px solid ${color}20`,
        cursor: 'pointer', display: 'flex', flexDirection: 'column', justifyContent: 'space-between'
      }}
    >
      <div style={{ display: 'flex', justifyContent: 'space-between', alignItems: 'flex-start' }}>
        <div style={{ display: 'flex', gap: '16px' }}>
          <div style={{
            padding: '12px', borderRadius: '16px',
            background: `${color}15`, border: `1px solid ${color}20`,
            display: 'flex', alignItems: 'center', justifyContent: 'center'
          }}>
            {icon}
          </div>
          <div>
            <h3 style={{ fontSize: '18px', fontWeight: '700', color: 'white', margin: '0 0 4px 0'
}}>{title}</h3>
            <p style={{ color: '#a1a1aa', fontSize: '13px', margin: 0 }}>{description}</p>
          </div>
```

```
        </div>
        <ArrowUpRight size={20} color={color} style={{ opacity: 0.7 }} />
      </div>
      <div style={{ display: 'flex', alignItems: 'center', marginTop: '16px', gap: '12px' }}>
        {tags && tags.map((tag, i) => (
          <span key={i} style={{
            fontSize: '11px', background: `${color}15`, color: color,
            padding: '4px 10px', borderRadius: '8px', fontWeight: '600'
          }}>
            {tag}
          </span>
        ))}
        {alert && (
          <div style={{ display: 'flex', alignItems: 'center', gap: '8px', fontSize: '12px', color:
'#e4e4e7' }}>
            <div style={{ width: '8px', height: '8px', borderRadius: '50%', background: color,
boxShadow: `0 0 8px ${color}` }} />
            <span style={{ maxWidth: '120px', whiteSpace: 'nowrap', overflow: 'hidden',
textOverflow: 'ellipsis' }}>{alert}</span>
          </div>
        )}
        {activeUsers > 0 && (
          <div style={{ display: 'flex', alignItems: 'center', gap: '10px' }}>
            <div style={{ display: 'flex', paddingLeft: '8px' }}>
              {[...Array(Math.min(activeUsers, 3))].map((_, i) => (
                <div key={i} style={{
                  width: '24px', height: '24px', borderRadius: '50%', background: '#52525b',
                  border: '2px solid #18181b', marginLeft: '-10px',
                  display: 'flex', alignItems: 'center', justifyContent: 'center', fontSize: '8px',
fontWeight: 'bold'
                }}>
                  {String.fromCharCode(65 + i)}
```

```
                    </div>
                  ))}
                </div>
                <span style={{ fontSize: '12px', color: '#a1a1aa' }}>{activeUsers} Members</span>
              </div>
            )}
          </div>
        </motion.div>
      );
      const StatCard = ({ label, value, icon, color, trend, chartData }) => (
        <motion.div
          variants={{ hidden: { opacity: 0, y: 10 }, visible: { opacity: 1, y: 0 } }}
          whileHover={{ y: -4, boxShadow: `0 10px 30px -10px ${color}15` }}
          style={{
            padding: '24px', height: '100%', display: 'flex', flexDirection: 'column', justifyContent: 'space-
      between',
            borderRadius: '24px', background: 'rgba(24, 24, 27, 0.6)',
            border: '1px solid rgba(255, 255, 255, 0.05)', position: 'relative', overflow: 'hidden',
            boxSizing: 'border-box'
          }}
        >
          <div style={{ display: 'flex', justifyContent: 'space-between', alignItems: 'flex-start' }}>
            <div style={{
              width: '48px', height: '48px', borderRadius: '14px',
              background: `${color}10`, color: color,
              display: 'flex', alignItems: 'center', justifyContent: 'center',
              border: `1px solid ${color}20`
            }}>
              {icon}
            </div>
            <span style={{
```

```
            fontSize: '11px', color: '#a1a1aa', fontWeight: '600',
            background: 'rgba(255,255,255,0.03)', padding: '4px 8px', borderRadius: '6px',
            border: '1px solid rgba(255,255,255,0.05)'
          }}>
            {trend}
          </span>
        </div>

        <div style={{ marginTop: '16px' }}>
          <div style={{ fontSize: '32px', fontWeight: '800', color: 'white', marginBottom: '4px', letterSpacing: '-1px' }}>{value}</div>
          <div style={{ fontSize: '13px', color: '#71717a', fontWeight: '600', textTransform: 'uppercase', letterSpacing: '0.5px' }}>{label}</div>
        </div>
        <div style={{ position: 'absolute', bottom: '24px', right: '24px', opacity: 0.5, pointerEvents: 'none' }}>
          <Sparkline data={chartData} color={color} />
        </div>
      </motion.div>
    );
    const PipelineTimelineRow = ({ pipe, index, navigate }) => (
      <motion.div
        initial={{ opacity: 0, x: -10 }}
        animate={{ opacity: 1, x: 0 }}
        transition={{ delay: index * 0.1 }}
        onClick={() => navigate(`/builder/${pipe.id}`)}
        style={{
          display: 'flex', alignItems: 'center', gap: '20px', padding: '16px 0',
          cursor: 'pointer', zIndex: 1, position: 'relative'
        }}
      >
```

```
{/* Timeline Dot */}
<div style={{
  width: '48px', display: 'flex', justifyContent: 'center', flexShrink: 0
}}>
  <div style={{
    width: '12px', height: '12px', borderRadius: '50%',
    background: pipe.status === 'Active' ? '#10b981' : '#27272a',
    border: pipe.status === 'Active' ? '2px solid rgba(16, 185, 129, 0.3)' : '2px solid #3f3f46',
    boxShadow: pipe.status === 'Active' ? '0 0 12px rgba(16, 185, 129, 0.4)' : 'none',
    zIndex: 2
  }} />
</div>
<motion.div
  style={{
    flex: 1, display: 'flex', alignItems: 'center', justifyContent: 'space-between',
    background: 'rgba(255,255,255,0.02)', padding: '16px 20px', borderRadius: '16px',
    border: '1px solid rgba(255,255,255,0.05)'
  }}
  whileHover={{ backgroundColor: 'rgba(255,255,255,0.04)', x: 4, borderColor:
'rgba(255,255,255,0.1)' }}
>
  <div>
    <div style={{ display: 'flex', alignItems: 'center', gap: '10px', marginBottom: '6px' }}>
      <span style={{ fontSize: '15px', fontWeight: '600', color: '#e4e4e7'
}}>{pipe.name}</span>
      <span style={{ fontSize: '11px', color: '#52525b', background:
'rgba(255,255,255,0.05)', padding: '2px 6px', borderRadius: '4px' }}>
        ID: {pipe.id}
      </span>
    </div>
    <div style={{ display: 'flex', gap: '12px', fontSize: '12px', color: '#a1a1aa' }}>
```

```
            <span style={{ display: 'flex', alignItems: 'center', gap: '4px' }}>
              <Clock size={12} /> {pipe.created_at || 'Just now'}
            </span>
            <span style={{ display: 'flex', alignItems: 'center', gap: '4px' }}>
              <Database size={12} /> SQL Source
            </span>
          </div>
        </div>
        <div style={{ display: 'flex', alignItems: 'center', gap: '16px' }}>
          {pipe.status === 'Active' ? (
            <span style={{
              fontSize: '11px', fontWeight: '700', textTransform: 'uppercase', letterSpacing: '0.5px',
              padding: '6px 12px', borderRadius: '20px',
              background: 'rgba(16, 185, 129, 0.1)', color: '#10b981',
              border: '1px solid rgba(16, 185, 129, 0.2)',
              display: 'flex', alignItems: 'center', gap: '6px'
            }}>
              <Activity size={12} className="spin" /> Running
            </span>
          ) : (
            <span style={{
              fontSize: '11px', fontWeight: '700', textTransform: 'uppercase', letterSpacing: '0.5px',
              padding: '6px 12px', borderRadius: '20px',
              background: 'rgba(255,255,255,0.05)', color: '#71717a',
              border: '1px solid rgba(255,255,255,0.05)'
            }}>
              Ready
            </span>
          )}
          <ChevronRight size={16} color="#52525b" />
```

```
          </div>
        </motion.div>
    </motion.div>
);
const Sparkline = ({ data, color }) => {
    const points = data.map((d, i) => `${i * 10},${50 - d / 2}`).join(' ');
    return (
        <svg width="60" height="30" viewBox="0 0 60 50" style={{ overflow: 'visible' }}>
            <path d={`M ${points}`} fill="none" stroke={color} strokeWidth="3"
strokeLinecap="round" strokeLinejoin="round" />
            <circle cx={data.length * 10 - 10} cy={50 - data[data.length-1] / 2} r="3" fill={color} />
        </svg>
    );
};
const SkeletonRow = () => (
    <div style={{ height: '76px', margin: '12px 0 12px 48px', background: 'rgba(255,255,255,0.02)',
borderRadius: '16px' }} />
);
const EmptyState = () => (
    <div style={{ textAlign: 'center', padding: '60px', marginLeft: '48px', border: '1px dashed
rgba(255,255,255,0.1)', borderRadius: '16px' }}>
        <p style={{ color: '#71717a' }}>No recent activity found.</p>
    </div>
);
export default Dashboard;
```

2. PipelineBuilder.jsx

```
import axios from 'axios';
import React, { useState, useCallback, useRef, useEffect, useMemo } from 'react';
import { useNavigate, useParams, useBeforeUnload } from 'react-router-dom';
import ReactFlow, {
```

```
  Controls,
  Background,
  applyNodeChanges,
  applyEdgeChanges,
  addEdge,
  ReactFlowProvider,
  useReactFlow,
  useViewport
} from 'reactflow';
import 'reactflow/dist/style.css';
import { motion, AnimatePresence } from 'framer-motion';
import { io } from 'socket.io-client';
import {
  ArrowLeft, LayoutTemplate, Upload, Download, Play, Save,
  Loader2, AlertCircle, CheckCircle2, X, Info, MousePointer2, Clock, Menu, Eye, Wand2, Calendar
} from 'lucide-react';
import Sidebar from './Sidebar';
import DataPreviewPanel from './DataPreviewPanel';
import FilterNode from './nodes/FilterNode';
import SourceNode from './nodes/SourceNode';
import DestinationNode from './nodes/DestinationNode';
import DeletableEdge from './edges/DeletableEdge';
import {
   SortNode, SelectNode, RenameNode, DedupeNode, FillNaNode, GroupByNode, JoinNode,
   CastNode, StringNode, CalcNode, LimitNode, ConstantNode, ChartNode, PythonNode
} from './nodes/Transformations';
import { TEMPLATES } from '../data/templates';
const getUserColor = (username) => {
   const safeName = username || 'Anonymous';
   const colors = ['#ef4444', '#f97316', '#f59e0b', '#84cc16', '#10b981', '#06b6d4', '#3b82f6',
'#8b5cf6', '#d946ef', '#f43f5e'];
```

```
    let hash = 0;
    for (let i = 0; i < safeName.length; i++) {
      hash = safeName.charCodeAt(i) + ((hash << 5) - hash);
    }
    return colors[Math.abs(hash) % colors.length];
};
const RemoteCursor = ({ x, y, name, color }) => (
    <div style={{
      position: 'absolute',
      left: x,
      top: y,
      pointerEvents: 'none',
      zIndex: 1000,
      transition: 'all 0.1s ease'
    }}>
      <MousePointer2 size={18} fill={color} color="white" />
      <div style={{
        backgroundColor: color,
        color: 'white',
        padding: '2px 6px',
        borderRadius: '4px',
        fontSize: '10px',
        whiteSpace: 'nowrap',
        marginLeft: '12px',
        marginTop: '2px',
        fontWeight: '600',
        boxShadow: '0 2px 4px rgba(0,0,0,0.2)'
      }}>
        {name}
      </div>
```

```
    </div>
  );
  const CursorsRenderer = ({ cursors }) => {
    const { x, y, zoom } = useViewport();
    return (
      <>
        {Object.entries(cursors).map(([userId, cursor]) => (
          <RemoteCursor
            key={userId}
            x={cursor.x * zoom + x}
            y={cursor.y * zoom + y}
            name={cursor.name}
            color={cursor.color}
          />
        ))}
      </>
    );
  };
  const initialNodes = [];
  const ScheduleModal = ({ isOpen, onClose, onSave, currentSchedule }) => {
   if (!isOpen) return null;
   const [mode, setMode] = useState('daily'); // 'daily' or 'date'
   const [time, setTime] = useState('09:00');
   const [date, setDate] = useState('');
   useEffect(() => {
    if (currentSchedule) {
     if (currentSchedule.startsWith('cron:')) {
      setMode('daily');
      const [_, val] = currentSchedule.split(':', 1); // Split only first colon? No, wait. cron:HH:MM
      // Correct splitting for cron:HH:MM
```

```
      const parts = currentSchedule.split(':');
      if (parts.length >= 3) {
          setTime(`${parts[1]}:${parts[2]}`);
      }
    } else if (currentSchedule.startsWith('date:')) {
      setMode('date');
      // date:YYYY-MM-DD HH:MM:SS
      // Remove "date:" prefix
      const raw = currentSchedule.substring(5);
      const [d, t] = raw.split(' ');
      setDate(d);
      if(t) setTime(t.slice(0, 5));
    }
  }
 }, [currentSchedule]);
 const handleSave = () => {
  if (mode === 'daily') {
    onSave('cron', time);
  } else {
    if (!date || !time) return alert("Please select both date and time");
    const formattedDate = `${date} ${time}:00`;
    onSave('date', formattedDate);
  }
 };
 return (
  <div style={{ position: 'fixed', inset: 0, background: 'rgba(0,0,0,0.7)', display: 'flex', alignItems: 'center', justifyContent: 'center', zIndex: 1000 }}>
    <div style={{ background: '#18181b', padding: '30px', borderRadius: '16px', width: '400px', border: '1px solid rgba(255,255,255,0.1)' }}>
      <div style={{ display: 'flex', justifyContent: 'space-between', alignItems: 'center', marginBottom: '20px' }}>
```

```
        <h2 style={{ fontSize: '20px', fontWeight: 'bold', margin: 0, display: 'flex', alignItems:
'center', gap: '10px', color: 'white' }}>
          <Clock size={20} color="#a78bfa" /> Schedule Pipeline
        </h2>
        <button onClick={onClose} style={{ background: 'transparent', border: 'none', color:
'#71717a', cursor: 'pointer' }}><X size={20} /></button>
      </div>
      <div style={{ display: 'flex', background: 'rgba(255,255,255,0.05)', padding: '4px',
borderRadius: '8px', marginBottom: '20px' }}>
        <button
          onClick={() => setMode('daily')}
          style={{
            flex: 1, padding: '8px', borderRadius: '6px', border: 'none', cursor: 'pointer', fontSize: '13px',
fontWeight: '600',
            background: mode === 'daily' ? '#27272a' : 'transparent',
            color: mode === 'daily' ? '#fff' : '#a1a1aa',
            boxShadow: mode === 'daily' ? '0 2px 5px rgba(0,0,0,0.2)' : 'none'
          }}
        >
          Run Daily
        </button>
        <button
          onClick={() => setMode('date')}
          style={{
            flex: 1, padding: '8px', borderRadius: '6px', border: 'none', cursor: 'pointer', fontSize: '13px',
fontWeight: '600',
            background: mode === 'date' ? '#27272a' : 'transparent',
            color: mode === 'date' ? '#fff' : '#a1a1aa',
            boxShadow: mode === 'date' ? '0 2px 5px rgba(0,0,0,0.2)' : 'none'
          }}
        >
          Specific Date
```

```
        </button>
      </div>
      <p style={{ color: '#a1a1aa', fontSize: '13px', marginBottom: '20px' }}>
        {mode === 'daily' ? 'Run this pipeline automatically every day at a specific time.' : 'Run this pipeline exactly once on a specific date.'}
      </p>
      <div style={{ display: 'flex', flexDirection: 'column', gap: '15px' }}>
        {mode === 'date' && (
          <div>
            <label style={{ display: 'block', fontSize: '12px', color: '#a1a1aa', marginBottom: '8px' }}>Date</label>
            <div style={{ position: 'relative' }}>
              <Calendar size={16} style={{ position: 'absolute', left: '12px', top: '50%', transform: 'translateY(-50%)', color: '#71717a' }} />
              <input
                type="date"
                value={date}
                onChange={(e) => setDate(e.target.value)}
                style={{
                  width: '100%', background: '#27272a', border: '1px solid rgba(255,255,255,0.1)',
                  padding: '12px 12px 12px 40px', borderRadius: '8px', color: 'white', outline: 'none'
                }}
              />
            </div>
          </div>
        )}
        <div>
          <label style={{ display: 'block', fontSize: '12px', color: '#a1a1aa', marginBottom: '8px' }}>Time (Server Time)</label>
          <div style={{ position: 'relative' }}>
            <Clock size={16} style={{ position: 'absolute', left: '12px', top: '50%', transform: 'translateY(-50%)', color: '#71717a' }} />
```

```
            <input
              type="time"
              value={time}
              onChange={(e) => setTime(e.target.value)}
              style={{
                width: '100%', background: '#27272a', border: '1px solid rgba(255,255,255,0.1)',
                padding: '12px 12px 12px 40px', borderRadius: '8px', color: 'white', outline: 'none'
              }}
            />
          </div>
        </div>
      </div>
      <div style={{ display: 'flex', justifyContent: 'flex-end', gap: '10px', marginTop: '30px' }}>
        <button onClick={onClose} style={{ background: 'transparent', border: '1px solid #3f3f46',
color: '#e4e4e7', padding: '10px 16px', borderRadius: '6px', cursor: 'pointer' }}>Cancel</button>
        <button
          onClick={handleSave}
          style={{
            background: '#3b82f6', color: 'white', border: 'none',
            padding: '10px 20px', borderRadius: '8px', cursor: 'pointer', fontWeight: '600'
          }}
        >
          Save Schedule
        </button>
      </div>
    </div>
  </div>
 );
};
const PipelineBuilderContent = () => {
   const navigate = useNavigate();
```

```
    const { id } = useParams();
    const reactFlowWrapper = useRef(null);
    const fileInputRef = useRef(null);
    const socketRef = useRef(null);
    const { getNodes, getEdges, screenToFlowPosition } = useReactFlow();
    const [nodes, setNodes] = useState(initialNodes);
    const [edges, setEdges] = useState([]);
    const [reactFlowInstance, setReactFlowInstance] = useState(null);
    const [pipelineName, setPipelineName] = useState("My New Pipeline");
    const [isRunning, setIsRunning] = useState(false);
    const [showTemplateModal, setShowTemplateModal] = useState(false);
    const [selectedNode, setSelectedNode] = useState(null);
    const [isMobile, setIsMobile] = useState(window.innerWidth < 768);
    const [isToolboxOpen, setIsToolboxOpen] = useState(!isMobile);
    const [showScheduleModal, setShowScheduleModal] = useState(false);
    const [currentScheduleStr, setCurrentScheduleStr] = useState(''); // Store the string from DB to
pass to modal
    const [showAIModal, setShowAIModal] = useState(false);
    const [aiPrompt, setAiPrompt] = useState("");
    const [isGenerating, setIsGenerating] = useState(false);
    const [isDirty, setIsDirty] = useState(false);
    const isLoadedRef = useRef(false);
    const [remoteCursors, setRemoteCursors] = useState({});
    const lastCursorUpdate = useRef(0);
    useEffect(() => {
      const handleResize = () => {
        const mobile = window.innerWidth < 768;
        setIsMobile(mobile);
        if (!mobile) setIsToolboxOpen(true);
        else if (mobile) setIsToolboxOpen(false);
      };
```

```
    window.addEventListener('resize', handleResize);
    return () => window.removeEventListener('resize', handleResize);
}, []);
const getUserInfo = () => {
    const storedUser = localStorage.getItem('user');
    if (storedUser) {
        try {
            const parsed = JSON.parse(storedUser);
            return {
                name: parsed.username || 'Anonymous',
                id: parsed.id ? String(parsed.id) : `user_${Date.now()}`
            };
        } catch (e) {
            console.error("Failed to parse user info", e);
        }
    }
    return { name: 'Anonymous', id: `user_${Date.now()}` };
};
const currentUser = useRef(getUserInfo());
const myColor = useMemo(() => getUserColor(currentUser.current.name), []);
const [previewPanel, setPreviewPanel] = useState({
    isOpen: false, loading: false, data: [], columns: [], error: null, nodeLabel: ''
});
const [notification, setNotification] = useState(null);
const showToast = (message, type = 'success') => {
    setNotification({ message, type });
    setTimeout(() => setNotification(null), 4000);
};
const updateNodeData = useCallback((nodeId, newData) => {
    setNodes((nds) => nds.map((node) => {
```

```
            if (node.id === nodeId) {
                return { ...node, data: { ...node.data, ...newData } };
            }
            return node;
        }));
        setIsDirty(true);
    }, []);
    const nodeTypes = useMemo(() => ({
        filterNode: FilterNode,
        source_unified: SourceNode,
        source_csv: SourceNode, source_json: SourceNode, source_excel: SourceNode, sourceNode: SourceNode,
        dest_db: DestinationNode, dest_csv: DestinationNode, dest_json: DestinationNode, dest_excel: DestinationNode, destinationNode: DestinationNode,
        trans_sort: SortNode, trans_select: SelectNode, trans_rename: RenameNode, trans_dedupe: DedupeNode,
        trans_fillna: FillNaNode, trans_group: GroupByNode, trans_join: JoinNode,
        trans_cast: CastNode, trans_string: StringNode, trans_calc: CalcNode, trans_limit: LimitNode,
        trans_constant: ConstantNode, vis_chart: ChartNode, trans_python: PythonNode
    }), []);
    const edgeTypes = useMemo(() => ({ deletableEdge: DeletableEdge }), []);
    useEffect(() => {
        if (!id) return;
        socketRef.current = io('http://127.0.0.1:5000');
        socketRef.current.emit('join_pipeline', { pipeline_id: id });
        socketRef.current.on('pipeline_updated', (data) => {
            if (data.type === 'node_change') {
                setNodes((nds) => applyNodeChanges(data.changes, nds));
            } else if (data.type === 'edge_change') {
                setEdges((eds) => applyEdgeChanges(data.changes, eds));
            } else if (data.type === 'connection') {
```

```
            setEdges((eds) => addEdge({ ...data.changes, type: 'deletableEdge' }, eds));
        }
    });
    socketRef.current.on('cursor_moved', (data) => {
        setRemoteCursors((prev) => ({
            ...prev,
            [data.userId]: { x: data.x, y: data.y, name: data.userName, color: data.color }
        }));
    });
    return () => {
        if (socketRef.current) {
            socketRef.current.emit('leave', { pipeline_id: id });
            socketRef.current.disconnect();
        }
    };
}, [id]);
const onMouseMove = useCallback((event) => {
    if (!reactFlowInstance || !socketRef.current) return;
    const now = Date.now();
    if (now - lastCursorUpdate.current > 50) {
        const flowPos = screenToFlowPosition({ x: event.clientX, y: event.clientY });
        socketRef.current.emit('cursor_move', {
            pipelineId: id,
            userId: currentUser.current.id,
            userName: currentUser.current.name,
            color: myColor,
            x: flowPos.x,
            y: flowPos.y
        });
        lastCursorUpdate.current = now;
```

```
    }
}, [reactFlowInstance, id, screenToFlowPosition, myColor]);
useEffect(() => {
    if (id) {
        isLoadedRef.current = false;
        const token = localStorage.getItem('token');
        axios.get(`http://127.0.0.1:5000/pipelines/${id}`, {
            headers: { Authorization: `Bearer ${token}` }
        })
        .then(response => {
            const { name, flow, schedule } = response.data;
            setPipelineName(name);
            setCurrentScheduleStr(schedule); // Store raw schedule string

            if (flow) {
                const loadedNodes = (flow.nodes || []).map(n => ({
                    ...n,
                    data: { ...n.data, onUpdate: updateNodeData }
                }));
                setNodes(loadedNodes);
                const dbEdges = (flow.edges || []).map(edge => ({
                    ...edge,
                    type: 'deletableEdge'
                }));
                setEdges(dbEdges);
            }
            setTimeout(() => { setIsDirty(false); isLoadedRef.current = true; }, 500);
        })
        .catch(err => {
            console.error("Error loading pipeline:", err);
```

```
            showToast("Could not load pipeline.", "error");
            isLoadedRef.current = true;
        });
    } else {
        isLoadedRef.current = true;
    }
}, [id, updateNodeData]);

useBeforeUnload(
    React.useCallback((e) => {
        if (isDirty) { e.preventDefault(); e.returnValue = ''; }
    }, [isDirty])
);
const handleBack = () => {
    if (isDirty) {
        if (window.confirm("You have unsaved changes. Are you sure you want to leave?")) {
            navigate('/dashboard');
        }
    } else {
        navigate('/dashboard');
    }
};
const onNodesChange = useCallback((changes) => {
    setNodes((nds) => applyNodeChanges(changes, nds));
    const selectionChange = changes.find(c => c.type === 'select');
    if (selectionChange && !selectionChange.selected) {
        if (selectedNode && selectedNode.id === selectionChange.id) {
            setSelectedNode(null);
        }
    }
```

```
    if (isLoadedRef.current) setIsDirty(true);

    if (socketRef.current && id) {
      socketRef.current.emit('pipeline_update', {
        pipelineId: id, type: 'node_change', changes: changes
      });
    }
  }, [id, selectedNode]);
  const onEdgesChange = useCallback((changes) => {
    setEdges((eds) => applyEdgeChanges(changes, eds));
    if (isLoadedRef.current) setIsDirty(true);

    if (socketRef.current && id) {
      socketRef.current.emit('pipeline_update', {
        pipelineId: id, type: 'edge_change', changes: changes
      });
    }
  }, [id]);
  const onConnect = useCallback((params) => {
    setEdges((eds) => addEdge({ ...params, type: 'deletableEdge' }, eds));
    if (isLoadedRef.current) setIsDirty(true);
    if (socketRef.current && id) {
      socketRef.current.emit('pipeline_update', {
        pipelineId: id, type: 'connection', changes: params
      });
    }
  }, [id]);
  const fetchPreview = async (node) => {
    if (!node) return;
    setPreviewPanel({
```

```
    isOpen: true,
    loading: true,
    data: [],
    columns: [],
    error: null,
    nodeLabel: node.data.label || node.type
});
const token = localStorage.getItem('token');
const currentNodes = getNodes();
const currentEdges = getEdges();
try {
    const payload = {
        targetNodeId: node.id,
        nodes: currentNodes.map(n => ({ id: n.id, type: n.type, data: n.data })),
        edges: currentEdges.map(e => ({ source: e.source, target: e.target }))
    };
    const res = await axios.post('http://127.0.0.1:5000/preview-node', payload, {
        headers: { Authorization: `Bearer ${token}` }
    });
    let previewData = res.data.data || [];
    let previewColumns = res.data.columns || [];
    if (previewData.length === 0) {
         let msg = "No data returned from server.";
         if (node.type.includes('source')) {
             msg = "Source preview empty (Lazy Loaded). Add a transformation to see data.";
             showToast(msg, "info");
         } else {
             showToast("No data returned for this node.", "info");
         }
         setPreviewPanel(prev => ({
```

```
            ...prev,
            loading: false,
            data: [],
            columns: [],
            error: msg
          }));
        } else {
          setPreviewPanel(prev => ({
            ...prev,
            loading: false,
            data: previewData,
            columns: previewColumns,
            error: null
          }));
        }
      } catch (err) {
        console.error("Preview error:", err);
        setPreviewPanel(prev => ({
          ...prev,
          loading: false,
          error: err.response?.data?.error || "Failed to fetch preview."
        }));
      }
    };
    const onNodeClick = useCallback((event, node) => {
      setSelectedNode(node);
      setPreviewPanel(prev => ({ ...prev, isOpen: false }));
    }, []);
    const onPaneClick = useCallback(() => {
      setSelectedNode(null);
```

```
    }, []);
    const onNodeContextMenu = useCallback((event, node) => {
        event.preventDefault();
        setSelectedNode(node);
        fetchPreview(node);
    }, [getNodes, getEdges]);
    const handleToolbarPreview = () => {
        if (selectedNode) {
            fetchPreview(selectedNode);
        } else {
            showToast("Please select a node to preview", "info");
        }
    };
    const onDragOver = useCallback((event) => { event.preventDefault();
event.dataTransfer.dropEffect = 'move'; }, []);
    const onDrop = useCallback(
        (event) => {
            event.preventDefault();
            const type = event.dataTransfer.getData('application/reactflow');
            const label = event.dataTransfer.getData('application/label');
            if (typeof type === 'undefined' || !type) return;
            const position = reactFlowInstance.screenToFlowPosition({
                x: event.clientX,
                y: event.clientY,
            });
            let defaultData = { label: label };
            if (type.includes('source') || type === 'sourceNode' || type === 'source_unified') {
                defaultData.fileType = 'UNIFIED';
            }
            if (type.includes('dest')) defaultData.destinationType = type.split('_')[1]?.toUpperCase() ||
'DB';
```

```
        if (type === 'filterNode') { defaultData.column = ''; defaultData.condition = '>';
defaultData.value = ''; }
        if (type === 'vis_chart') { defaultData.chartType = 'bar'; defaultData.x_col = '';
defaultData.y_col = ''; defaultData.outputName = 'my_chart'; }
        if (type === 'trans_python') { defaultData.code = "df['new_col'] = df['old_col'] * 2"; }
        defaultData.onUpdate = updateNodeData;
        const newNode = {
          id: `${type}_${Date.now()}`,
          type: type,
          position,
          data: defaultData,
        };
        setNodes((nds) => nds.concat(newNode));
        setIsDirty(true);
        setSelectedNode(newNode);
        if (isMobile) setIsToolboxOpen(false);
      },
      [reactFlowInstance, updateNodeData, isMobile]
    );
    const savePipeline = async () => {
      const currentNodes = getNodes();
      const currentEdges = getEdges();
      const flow = { nodes: currentNodes, edges: currentEdges };
      const token = localStorage.getItem('token');
      try {
        const payload = { name: pipelineName, flow: flow };
        const url = id ? `http://127.0.0.1:5000/pipelines/${id}` : 'http://127.0.0.1:5000/pipelines';
        const method = id ? axios.put : axios.post;
        const res = await method(url, payload, { headers: { Authorization: `Bearer ${token}` } });
        showToast(id ? 'Pipeline Updated Successfully!' : 'Pipeline Created Successfully!', 'success');
        setIsDirty(false);
```

```
            if (!id) navigate(`/builder/${res.data.id}`);
        } catch (error) {
            showToast('Error saving pipeline: ' + (error.response?.data?.msg || error.message), 'error');
        }
    };
    const handleAIGenerate = async () => {
        if (!aiPrompt.trim()) return;
        setIsGenerating(true);
        try {
            const token = localStorage.getItem('token');
            const res = await axios.post('http://127.0.0.1:5000/api/generate_pipeline',
                { prompt: aiPrompt },
                { headers: token ? { Authorization: `Bearer ${token}` } : {} }
            );
            const data = res.data;
            if (data.nodes && data.edges) {
                const newNodes = data.nodes.map(n => ({
                    ...n,
                    data: { ...n.data, onUpdate: updateNodeData }
                }));
                const newEdges = data.edges.map(e => ({ ...e, type: 'deletableEdge', animated: true }));
                setNodes(newNodes);
                setEdges(newEdges);
                setShowAIModal(false);
                setAiPrompt("");
                setIsDirty(true);
                showToast("Pipeline generated from text!", "success");
            } else if (data.error) {
                showToast(`AI Error: ${data.error}`, "error");
            } else {
```

```
            showToast("AI could not generate a valid pipeline. Try a more specific prompt.", "error");
        }
    } catch (error) {
        console.error(error);
        const errMsg = error.response?.data?.error || error.message || "Connection failed";
        showToast(`Server Error: ${errMsg}`, "error");
    } finally {
        setIsGenerating(false);
    }
};
const saveSchedule = async (type, value) => {
    try {
        const token = localStorage.getItem('token');
        if (!id) {
            showToast("Please save the pipeline first.", "error");
            return;
        }
        await axios.post(`http://127.0.0.1:5000/pipelines/${id}/schedule`, {
            type: type,
            value: value
        }, { headers: { Authorization: `Bearer ${token}` }});
        setCurrentScheduleStr(`${type}:${value}`); // Update local state
        showToast('Schedule Saved!', 'success');
        setShowScheduleModal(false);
    } catch (err) {
        console.error(err);
        showToast('Failed to save schedule', 'error');
    }
};
const handleExport = () => {
```

```
        const currentNodes = getNodes();
        const currentEdges = getEdges();
        const exportData = { name: pipelineName, nodes: currentNodes, edges: currentEdges };
        const dataStr = "data:text/json;charset=utf-8," +
encodeURIComponent(JSON.stringify(exportData, null, 2));
        const downloadAnchorNode = document.createElement('a');
        downloadAnchorNode.setAttribute("href", dataStr);
        downloadAnchorNode.setAttribute("download", `${pipelineName.replace(/\s+/g,
'_')}_pipeline.json`);
        document.body.appendChild(downloadAnchorNode);
        downloadAnchorNode.click();
        downloadAnchorNode.remove();
        showToast("Pipeline exported to JSON", 'info');
    };
    const handleImportClick = () => { fileInputRef.current.click(); };
    const handleImportFile = (event) => {
        const file = event.target.files[0];
        if (!file) return;
        const reader = new FileReader();
        reader.onload = (e) => {
            try {
                const json = JSON.parse(e.target.result);
                if (json.nodes && Array.isArray(json.nodes)) {
                    const importedNodes = json.nodes.map(n => ({
                        ...n,
                        data: { ...n.data, onUpdate: updateNodeData }
                    }));
                    setNodes(importedNodes);
                    const importedEdges = (json.edges || []).map(edge => ({ ...edge, type: 'deletableEdge'
}));
                    setEdges(importedEdges);
```

```
                if (json.name) setPipelineName(json.name);
                setIsDirty(true);
                showToast("Pipeline imported successfully!", 'success');
            } else {
                showToast("Invalid JSON format", 'error');
            }
        } catch (err) { showToast("Failed to parse JSON file.", 'error'); }
    };
    reader.readAsText(file);
    event.target.value = null;
};
const handleLoadTemplate = (template) => {
    if (nodes.length > 0 && !window.confirm("Loading a template will replace your current canvas. Continue?")) return;
    const idMap = {};
    const timestamp = Date.now();
    const newNodes = template.nodes.map(n => {
        const newId = `${n.id}_${timestamp}`;
        idMap[n.id] = newId;
        return {
            ...n,
            id: newId,
            data: { ...n.data, onUpdate: updateNodeData }
        };
    });
    const newEdges = template.edges.map(e => ({
        ...e,
        id: `e_${e.source}_${e.target}_${timestamp}`,
        source: idMap[e.source],
        target: idMap[e.target],
        type: 'deletableEdge'
```

```
    }));
    setNodes(newNodes);
    setEdges(newEdges);
    setPipelineName(template.name);
    setIsDirty(true);
    setShowTemplateModal(false);
    showToast(`Template '${template.name}' loaded`, 'success');
};
const handleRun = async () => {
    const currentNodes = getNodes();
    const currentEdges = getEdges();
    if (currentNodes.length === 0) { showToast("Canvas is empty.", 'error'); return; }
    setIsRunning(true);
    try {
        const payload = {
            nodes: currentNodes.map(n => ({ id: n.id, type: n.type, data: n.data })),
            edges: currentEdges.map(e => ({ source: e.source, target: e.target })),
            pipelineId: id
        };
        const token = localStorage.getItem('token');
        const res = await axios.post('http://127.0.0.1:5000/run-pipeline', payload, {
            headers: token ? { Authorization: `Bearer ${token}` } : {}
        });
        console.log(res.data.logs);
        showToast("Pipeline executed successfully!", 'success');
    } catch (error) {
        let errMsg = error.response?.data?.error || error.message;
        showToast(`Execution Failed: ${errMsg}`, 'error');
    } finally {
        setIsRunning(false);
```

```
    }
  };
  const ActionButton = ({ icon, label, onClick, disabled = false, special = false }) => (
    <motion.button
      onClick={onClick}
      disabled={disabled}
      style={{
        background: special ? 'linear-gradient(45deg, #ec4899, #8b5cf6)' : 'transparent',
        border: special ? 'none' : '1px solid rgba(255,255,255,0.1)',
        color: disabled ? '#52525b' : (special ? 'white' : '#d4d4d8'),
        padding: '8px 12px', fontSize: '13px', borderRadius: '6px',
        cursor: disabled ? 'not-allowed' : 'pointer',
        display: 'flex', alignItems: 'center', gap: '6px',
        boxShadow: special ? '0 4px 15px rgba(236, 72, 153, 0.3)' : 'none',
        fontWeight: special ? '600' : 'normal'
      }}
      whileHover={!disabled ? {
        background: special ? 'linear-gradient(45deg, #db2777, #7c3aed)' :
'rgba(255,255,255,0.05)',
        borderColor: special ? 'none' : 'rgba(255,255,255,0.2)',
        color: 'white'
      } : {}}
      whileTap={!disabled ? { scale: 0.95 } : {}}
    >
      {icon} {label}
    </motion.button>
  );
  const ToastNotification = () => (
    <AnimatePresence>
      {notification && (
        <motion.div
```

```
            initial={{ opacity: 0, y: -50, x: '-50%' }}
            animate={{ opacity: 1, y: 20, x: '-50%' }}
            exit={{ opacity: 0, y: -20, x: '-50%' }}
            style={{
              position: 'fixed', left: '50%', top: 0, zIndex: 2000,
              background: notification.type === 'error' ? 'rgba(239, 68, 68, 0.9)' :
                          notification.type === 'info' ? 'rgba(59, 130, 246, 0.9)' :
                          'rgba(16, 185, 129, 0.9)',
              color: 'white', padding: '12px 24px', borderRadius: '50px',
              display: 'flex', alignItems: 'center', gap: '10px',
              backdropFilter: 'blur(10px)', boxShadow: '0 10px 30px rgba(0,0,0,0.3)',
              border: '1px solid rgba(255,255,255,0.2)', minWidth: '300px', justifyContent: 'center'
            }}
          >
            {notification.type === 'error' ? <AlertCircle size={20} /> :
             notification.type === 'info' ? <Info size={20} /> :
             <CheckCircle2 size={20} />}
            <span style={{ fontSize: '14px', fontWeight: '500' }}>{notification.message}</span>
            <button
              onClick={() => setNotification(null)}
              style={{ background: 'transparent', border: 'none', color: 'white', marginLeft: 'auto',
cursor: 'pointer', display: 'flex' }}
            >
              <X size={16} />
            </button>
          </motion.div>
        )}
      </AnimatePresence>
    );
    return (
```

```
      <div style={{ height: '100vh', display: 'flex', flexDirection: 'column', backgroundColor: '#0f1115', position: 'relative' }}>
        <ToastNotification />
        <input type="file" accept=".json" ref={fileInputRef} style={{ display: 'none' }} onChange={handleImportFile} />
        <motion.header
          style={{
            height: '60px',
            background: 'rgba(24, 24, 27, 0.8)',
            backdropFilter: 'blur(10px)',
            borderBottom: '1px solid rgba(255, 255, 255, 0.05)',
            display: 'flex', alignItems: 'center', justifyContent: 'space-between', padding: '0 20px',
            zIndex: 10,
            overflowX: 'auto',
            whiteSpace: 'nowrap'
          }}
          initial={{ y: -50 }} animate={{ y: 0 }}
        >
          <div style={{ display: 'flex', alignItems: 'center', gap: '20px' }}>
            <motion.button
              onClick={handleBack}
              style={{ background: 'transparent', border: 'none', color: '#a1a1aa', cursor: 'pointer', display: 'flex', alignItems: 'center', gap: '5px' }}
              whileHover={{ color: 'white', x: -3 }}
            >
              <ArrowLeft size={18} /> Back
            </motion.button>
            <div style={{ width: '1px', height: '24px', background: 'rgba(255,255,255,0.1)' }}></div>
            <div style={{ position: 'relative' }}>
              <input
                type="text"
```

```
                value={pipelineName}
                onChange={(e) => { setPipelineName(e.target.value); setIsDirty(true); }}
                style={{
                  background: 'transparent',
                  border: 'none',
                  color: 'white',
                  fontSize: '16px',
                  fontWeight: '600',
                  outline: 'none',
                  width: '300px'
                }}
              />
              {isDirty && (
                <span style={{
                  position: 'absolute', right: '-80px', top: '50%', transform: 'translateY(-50%)',
                  fontSize: '11px', color: '#fbbf24', display: 'flex', alignItems: 'center', gap: '4px'
                }}>
                  <AlertCircle size={10} /> Unsaved
                </span>
              )}
            </div>
          </div>
          <div style={{ display: 'flex', gap: '10px' }}>
            <ActionButton
              icon={<Wand2 size={16}/>}
              label="AI Generate"
              onClick={() => setShowAIModal(true)}
              special={true}
            />
            <ActionButton icon={<LayoutTemplate size={16}/>} label="Templates" onClick={()
=> setShowTemplateModal(true)} />
```

```
            <ActionButton
               icon={<Eye size={16}/>}
               label="Preview"
               onClick={handleToolbarPreview}
               disabled={!selectedNode}
            />
            <ActionButton icon={<Clock size={16}/>} label="Schedule" onClick={() =>
setShowScheduleModal(true)} />


            <div style={{ width: '1px', height: '24px', background: 'rgba(255,255,255,0.1)',
alignSelf: 'center' }}></div>
            <ActionButton icon={<Upload size={16}/>} label="Import"
onClick={handleImportClick} />
            <ActionButton icon={<Download size={16}/>} label="Export"
onClick={handleExport} />
            <motion.button
               onClick={handleRun}
               disabled={isRunning}
               className="btn"
               style={{
                  padding: '8px 16px', fontSize: '13px', display: 'flex', alignItems: 'center', gap: '6px',
                  background: isRunning ? 'rgba(16, 185, 129, 0.2)' : '#10b981',
                  color: isRunning ? '#a1a1aa' : 'black', fontWeight: '600',
                  border: 'none', cursor: isRunning ? 'not-allowed' : 'pointer', borderRadius: '6px'
               }}
               whileHover={!isRunning ? { scale: 1.05 } : {}}
               whileTap={!isRunning ? { scale: 0.95 } : {}}
            >
               {isRunning ? <Loader2 size={16} className="spin" /> : <Play size={16}
fill="black" />}
               {isRunning ? 'Running...' : 'Run'}
            </motion.button>
```

```
        <motion.button
          className="btn"
          onClick={savePipeline}
          style={{
            padding: '8px 16px', fontSize: '13px', display: 'flex', alignItems: 'center', gap: '6px',
            background: '#3b82f6', color: 'white', fontWeight: '600',
            border: 'none', cursor: 'pointer', borderRadius: '6px'
          }}
          whileHover={{ scale: 1.05 }} whileTap={{ scale: 0.95 }}
        >
          <Save size={16} /> Save
        </motion.button>
      </div>
    </motion.header>
    <div style={{ display: 'flex', flexGrow: 1, overflow: 'hidden', position: 'relative' }}>
      {isMobile && (
        <button
          onClick={() => setIsToolboxOpen(!isToolboxOpen)}
          style={{
            position: 'absolute', top: '10px', left: '10px', zIndex: 50,
            background: '#27272a', border: '1px solid #3f3f46', color: 'white',
            padding: '8px', borderRadius: '6px', cursor: 'pointer',
            display: 'flex', alignItems: 'center', justifyContent: 'center'
          }}
        >
          {isToolboxOpen ? <X size={20} /> : <Menu size={20} />}
        </button>
      )}
      <div style={{
        display: isToolboxOpen ? 'block' : 'none',
```

```
            position: isMobile ? 'absolute' : 'relative',
            zIndex: 40,
            height: '100%',
            backgroundColor: isMobile ? '#0f1115' : 'transparent',
            boxShadow: isMobile ? '5px 0 15px rgba(0,0,0,0.5)' : 'none'
          }}>
            <Sidebar />
          </div>
          <div
            className="reactflow-wrapper"
            ref={reactFlowWrapper}
            style={{ width: '100%', height: '100%', backgroundColor: '#0f1115' }}
            onMouseMove={onMouseMove}
          >
            <div style={{ position: 'absolute', top: '20px', right: '20px', pointerEvents: 'none', zIndex: 10, background: 'rgba(24, 24, 27, 0.8)', padding: '8px 12px', borderRadius: '8px', border: '1px solid rgba(255,255,255,0.1)', backdropFilter: 'blur(4px)' }}>
              <p style={{ margin: 0, fontSize: '12px', color: '#9ca3af' }}>
              <span style={{ fontWeight: 'bold', color: '#e5e7eb' }}>Left-click</span> to select, <span style={{ fontWeight: 'bold', color: '#e5e7eb' }}>Preview</span> in toolbar
              </p>
            </div>
            <ReactFlow
              nodes={nodes} edges={edges} onNodesChange={onNodesChange} onEdgesChange={onEdgesChange} onConnect={onConnect}
              nodeTypes={nodeTypes} edgeTypes={edgeTypes} onInit={setReactFlowInstance} onDrop={onDrop} onDragOver={onDragOver}
              onNodeClick={onNodeClick}
              onPaneClick={onPaneClick}
              onNodeContextMenu={onNodeContextMenu}
              fitView deleteKeyCode={['Backspace', 'Delete']}
              proOptions={{ hideAttribution: true }}
```

```
            >
                <CursorsRenderer cursors={remoteCursors} />
                <Background color="#27272a" gap={25} size={1} />
                <Controls style={{ background: '#27272a', border: '1px solid rgba(255,255,255,0.1)',
fill: 'white' }} />
            </ReactFlow>
          </div>
        </div>

        <DataPreviewPanel
          isOpen={previewPanel.isOpen}
          onClose={() => setPreviewPanel(prev => ({ ...prev, isOpen: false }))}
          data={previewPanel.data}
          columns={previewPanel.columns}
          loading={previewPanel.loading}
          error={previewPanel.error}
          nodeLabel={previewPanel.nodeLabel}
        />
        <AnimatePresence>
        {showTemplateModal && (
          <motion.div
            style={{
              position: 'fixed', top: 0, left: 0, width: '100%', height: '100%',
              backgroundColor: 'rgba(0,0,0,0.7)', backdropFilter: 'blur(8px)',
              zIndex: 1000, display: 'flex', justifyContent: 'center', alignItems: 'center'
            }}
            initial={{ opacity: 0 }} animate={{ opacity: 1 }} exit={{ opacity: 0 }}
          >
            <motion.div
              style={{
                width: '800px', maxHeight: '80vh', backgroundColor: '#18181b',
```

```
                    border: '1px solid rgba(255,255,255,0.1)', borderRadius: '16px', padding: '30px',
                    display: 'flex', flexDirection: 'column', overflow: 'hidden',
                    boxShadow: '0 25px 50px -12px rgba(0, 0, 0, 0.5)'
                  }}
                  initial={{ scale: 0.9, y: 20 }} animate={{ scale: 1, y: 0 }} exit={{ scale: 0.9, y: 20 }}
                >
                  <div style={{ display: 'flex', justifyContent: 'space-between', alignItems: 'center', marginBottom: '25px' }}>
                    <h2 style={{ margin: 0, color: 'white', fontSize: '22px' }}>Pipeline Templates</h2>
                    <button onClick={() => setShowTemplateModal(false)} style={{ background: 'transparent', border: 'none', color: '#a1a1aa', cursor: 'pointer' }}><X size={24} /></button>
                  </div>
                  <div style={{ overflowY: 'auto', display: 'grid', gridTemplateColumns: 'repeat(2, 1fr)', gap: '20px', padding: '5px' }}>
                    {TEMPLATES.map(tpl => (
                      <motion.div
                        key={tpl.id}
                        onClick={() => handleLoadTemplate(tpl)}
                        style={{
                          backgroundColor: 'rgba(255,255,255,0.03)',
                          border: '1px solid rgba(255,255,255,0.05)',
                          borderRadius: '12px', padding: '20px',
                          cursor: 'pointer'
                        }}
                        whileHover={{ scale: 1.02, backgroundColor: 'rgba(255,255,255,0.06)', borderColor: '#3b82f6' }}
                        whileTap={{ scale: 0.98 }}
                      >
                        <h3 style={{ margin: '0 0 8px 0', color: 'white', fontSize: '16px', display: 'flex', alignItems: 'center', gap: '8px' }}>
                          {tpl.name}
```

```
                        </h3>
                        <p style={{ margin: 0, color: '#a1a1aa', fontSize: '13px', lineHeight: '1.5'
}}>{tpl.description}</p>
                        <div style={{ marginTop: '15px', display: 'flex', gap: '6px', flexWrap: 'wrap'
}}>
                            {tpl.nodes.slice(0, 4).map((n, i) => (
                                <span key={i} style={{ fontSize: '10px', background:
'rgba(255,255,255,0.1)', padding: '2px 8px', borderRadius: '4px', color: '#e4e4e7' }}>
                                    {n.type.split('_')[1] || n.type}
                                </span>
                            ))}
                        </div>
                    </motion.div>
                ))}
            </div>
        </motion.div>
    </motion.div>
)}
</AnimatePresence>
<AnimatePresence>
{showAIModal && (
    <motion.div
        style={{
            position: 'fixed', top: 0, left: 0, width: '100%', height: '100%',
            backgroundColor: 'rgba(0,0,0,0.7)', backdropFilter: 'blur(8px)',
            zIndex: 1000, display: 'flex', justifyContent: 'center', alignItems: 'center'
        }}
        initial={{ opacity: 0 }} animate={{ opacity: 1 }} exit={{ opacity: 0 }}
    >
        <motion.div
            style={{
```

```
                    width: '500px', backgroundColor: '#18181b',
                    border: '1px solid rgba(255,255,255,0.1)', borderRadius: '16px', padding: '30px',
                    display: 'flex', flexDirection: 'column',
                    boxShadow: '0 25px 50px -12px rgba(0, 0, 0, 0.5)'
                }}
                initial={{ scale: 0.9, y: 20 }} animate={{ scale: 1, y: 0 }} exit={{ scale: 0.9, y: 20 }}
            >
                <div style={{ display: 'flex', justifyContent: 'space-between', alignItems: 'center', marginBottom: '20px' }}>
                    <h2 style={{ margin: 0, color: 'white', fontSize: '20px', display: 'flex', gap: '10px', alignItems: 'center' }}>
                        <Wand2 size={20} color="#d946ef" /> AI Pipeline Generator
                    </h2>
                    <button onClick={() => setShowAIModal(false)} style={{ background: 'transparent', border: 'none', color: '#a1a1aa', cursor: 'pointer' }}><X size={20} /></button>
                </div>
                <p style={{ color: '#a1a1aa', fontSize: '14px', marginTop: 0, marginBottom: '20px' }}>
                    Describe your pipeline in plain English. For example: <br/>
                    <i style={{ color: '#94a3b8' }}>"Load sales.csv, filter rows where Amount = 500, and save as high_value.json"</i>
                </p>
                <textarea
                    value={aiPrompt}
                    onChange={(e) => setAiPrompt(e.target.value)}
                    placeholder="Type your request here..."
                    style={{
                        width: '100%', height: '100px', background: '#27272a',
                        border: '1px solid #3f3f46', borderRadius: '8px',
                        color: 'white', padding: '12px', marginBottom: '20px',
                        resize: 'none', fontSize: '14px', outline: 'none', boxSizing: 'border-box'
                    }}
```

```
                />
                <div style={{ display: 'flex', gap: '10px', justifyContent: 'flex-end' }}>
                  <button
                    onClick={() => setShowAIModal(false)}
                    style={{
                      background: 'transparent', border: '1px solid #3f3f46', color: '#e4e4e7',
                      padding: '10px 16px', borderRadius: '6px', cursor: 'pointer', fontWeight: '500'
                    }}
                  >
                    Cancel
                  </button>
                  <button
                    onClick={handleAIGenerate}
                    disabled={isGenerating}
                    style={{
                      background: 'linear-gradient(45deg, #ec4899, #8b5cf6)',
                      border: 'none', color: 'white',
                      padding: '10px 20px', borderRadius: '6px', cursor: isGenerating ? 'not-
allowed' : 'pointer',
                      fontWeight: '600', display: 'flex', gap: '8px', alignItems: 'center',
                      opacity: isGenerating ? 0.7 : 1
                    }}
                  >
                    {isGenerating ? <Loader2 size={16} className="spin"/> : <Wand2
size={16}/>}
                    {isGenerating ? 'Generating...' : 'Generate Pipeline'}
                  </button>
                </div>
              </motion.div>
            </motion.div>
          )}
```

```
            </AnimatePresence>
            <ScheduleModal
               isOpen={showScheduleModal}
               onClose={() => setShowScheduleModal(false)}
               onSave={saveSchedule}
               currentSchedule={currentScheduleStr}
            />
         </div>
      );
};
const PipelineBuilder = () => (<ReactFlowProvider><PipelineBuilderContent
/></ReactFlowProvider>);
export default PipelineBuilder;
```

13. BIBLIOGRAPHY

BIBLIOGRAPHY

Books Used:

- *Software Engineering: A Practitioner's Approach* – Roger S. Pressman
- *Flask Web Development: Developing Web Applications with Python* – Miguel Grinberg
- *Python for Data Analysis* – Wes McKinney (Creator of Pandas)
- *Learning React: Modern Patterns for Developing React Apps* – Alex Banks & Eve Porcello

Website References:

- https://reactflow.dev (For the pipeline visualization interface)
- https://flask.palletsprojects.com (For the backend framework)
- https://pandas.pydata.org (For data processing logic)
- https://socket.io (For real-time pipeline updates)
- https://ai.google.dev (For the Gemini AI integration)