Graphic Era
Hill University
DEHRADUN • BHIMTAL • HALDWANI

PROJECT AND TEAM INFORMATION

Project Title

OS Algorithm Visualizer: A Web-Based Interactive Platform for Understanding Operating System Algorithms

Student/Team Information

Team Name:	CodeMasters
Team member 1 (Team Lead)	Chandra, Vinod – 22011515 itsvinod14@gmail.com

Team member 2	Rana, Rahul – 220113260 rahulrana89546@gmail.com
Team member 3	Chauhan, Divyansh – 22011456 divyanshchauhan349@gmail.com

PROJECT PROGRESS DESCRIPTION

Project Abstract

The OS Algorithm Visualizer aims to provide an intuitive and dynamic interface to understand and simulate key Operating System algorithms. From CPU scheduling to memory and disk management, the project will let users choose an algorithm type, select specific algorithms like FCFS, SJF, LRU, etc., and view their operations through visual outputs like Gantt charts and memory maps. This tool serves as a learning aid for students and educators by bridging the gap between theoretical OS concepts and their real-time execution. Built using a Flask backend and React frontend, the tool emphasizes simplicity, interactivity, and modularity.

Updated Project Approach and Architecture

The system is built using modular architecture:

- Frontend: React.js with plain CSS (organized in a style folder). Font Awesome is used for icons. The UI includes a responsive navbar, algorithm category selection, algorithm list, and input form toggling between default and custom input.
- Backend: Flask (Python) handles algorithm logic for process, disk, and memory management. Backend receives JSON input, computes the logic, and sends the output back to the frontend for visualization.
- Communication: REST API via Flask using the /visualize endpoint.
- Libraries Used: Axios (frontend), Flask, and standard Python libraries for logic.

This separation of concerns allows independent scalability and testing of UI and backend.

Tasks Completed

Task Completed	Team Member
Frontend navbar and routing setup	Member 2 (Rahul)
Algorithm category and list selection component	Member 2 (Rahul)
Custom/default input toggle UI	Member 1 (Vinod)
Basic Flask setup and routing	Member 1 (Vinod)
FCFS, SJF (NP/P) backend logic	Member 3 + Member 1 (Divyansh + Vinod)
Disk scheduling input form and logic	Member 3 (Divyansh)
Memory Management Algo	Member 2 + Member 3(Divyansh + Rahul)
Deployment	Member 1 (Vinod)

Challenges/Roadblocks

One major challenge has been synchronizing input forms between the frontend and the backend for different algorithm types due to their diverse input structures. To solve this, we're creating a unified data format sent to the backend regardless of the algorithm.

Another challenge is visualizing backend output meaningfully, especially for memory and disk scheduling algorithms. We're addressing this by integrating libraries like Chart.js or manually rendering Gantt chart bars using styled components.

Deploying and testing both parts locally without a database also creates session handling issues, so we are considering storing input temporarily in memory or using local Storage on the client side.

Project Outcome/Deliverables

- Fully working React frontend with OS algorithm input forms and UI feedback
- Flask backend with logic for Process Scheduling, Disk Scheduling, Memory Management
- REST API integration between frontend and backend
- Visualization output for selected algorithms
- Optional downloadable result (JSON or image)
- Educational documentation to assist future users

Progress Overview

The project is now 100% complete. The frontend is fully functional with all features implemented, and the backend logic for process, disk, and memory management is successfully integrated. Visualizations for all algorithms are also implemented and working as expected.

Codebase Information

GitHub Repository: https://github.com/VinodPandey14/OS-Algorithm-Visualizer
Live Link : https://os-algo.netlify.app/
Branch: *main*
Important Commits:

- fd2c1e2: UI input form toggle logic
- a15e3bb: FCFS backend logic with API connection
- c901fe1: Disk scheduling logic added

Testing and Validation Status

Test Type	Status (Pass/Fail)	Notes
FCFS Input-Output Test	Pass	Works for both custom and default input
SJF Preemptive Test	Pass	Accurate Gantt chart and timing output
Memory Management Logic	Pass	UI built but logic under implementation
API Input Validation	Pass	Handles malformed input gracefully
Algorithm Visualization	Pass	Checking the visualization working
Deployment Bugs	Pass	Web app working after deployment

Deliverables Progress

Deliverable	Status
Frontend UI with form toggling	Completed
Backend logic for scheduling	Completed
Disk and memory management	Completed
Visualization charts	Completed
API integration	Completed
Final documentation	Completed
Deployment	Completed