PROJECT AND TEAM INFORMATION

Project Title:

Interactive Web-Based OS Simulator: CPU Scheduling and Deadlock Detection using RAG

Student / Team Information

Team Name:	
Team member 1 (Team Lead) • **Name:** — Kunal Malik • **Student ID:** —220211174 • **Email:** — Kunall0880@gmail.com	
Team member 2 • **Name:** — Akshat Bajpai • **Student ID:** —22011505 • **Email:** — akshatvajpai41@gmail.com	

Team member 3

- **Name:** — Arun Jethi
- **Student ID:** — 22021365
- **Email:** — arunjethi38@gmail.com

PROPOSAL DESCRIPTION

Motivation :

CPU scheduling and deadlock detection are foundational concepts in Operating Systems. However, students often struggle with understanding how these work in practice due to the abstract nature of textbook diagrams and static examples. Traditional teaching methods do not offer the dynamic, visual feedback necessary to intuitively understand the behavior of these mechanisms.

This project is inspired by the need to make OS concepts more interactive and engaging. By building a web-based simulator that not only visualizes CPU scheduling algorithms like FCFS, SJF, Round Robin, and Priority Scheduling but also simulates deadlock detection using a Resource Allocation Graph (RAG)

We aim to provide a comprehensive, hands-on learning tool. The visualizations will help users grasp the nuances of process execution and system deadlock detection step-by-step.

State of the Art / Current solution :

Currently, most educational tools for OS scheduling are either outdated desktop applications or limited in scope. They do not support real-time input, lack customization, or fail to provide performance metrics. Most deadlock detection tools are theoretical, often requiring manual cycle detection in RAGs, making it hard to relate the concept to actual system behavior.

Modern web technologies such as Flask for the backend and Chart.js for frontend visualization can now be used to simulate system behaviors dynamically. Our simulator aims to bridge the gap between theory and practice using interactive Gantt charts, input forms, performance tables, and real-time RAG construction with deadlock checks.

Project Goals and Milestones :

General Goals:

1. Develop an interactive web-based simulator to visualize CPU scheduling and deadlock detection.
2. Allow users to enter custom process and resource data.
3. Implement major CPU scheduling algorithms.
4. Dynamically generate Gantt charts and RAGs.
5. Detect cycles in the RAG and identify deadlocked processes visually.
6. Present turnaround time, waiting time, and response time metrics.

Milestones:

Week	Milestone
1–2	Setup frontend UI and forms for process and resource input
3	Add dropdown to select scheduling algorithm and input validation
4	Implement FCFS and SJF scheduling algorithms
5	Add Round Robin (with quantum input) and Priority Scheduling (non-pre-emptive) logic.
6	Integrate Chart.js for rendering dynamic Gantt chart based on response from backend.
7	Implement RAG visualization and deadlock detection logic
8	Test edge cases, refine user interface, and deploy final version on Vercel.

Project Approach :

Our solution will be implemented as a **web-based application** using a **modular architecture** that separates frontend (UI/UX), backend (scheduling logic), and visualization logic of scheduling and deadlock detection.

Frontend:

- **Tech Stack: HTML, CSS, JavaScript**
- **Features:**
  - **Input forms for process ID, arrival time, burst time, and priority**
  - **Dropdown to choose scheduling algorithm**
  - **Interactive RAG creation with Request and Allocate buttons**
  - **Buttons for "Start Simulation" and "Check Deadlock"**
  - **Chart.js for Gantt chart**

Backend:

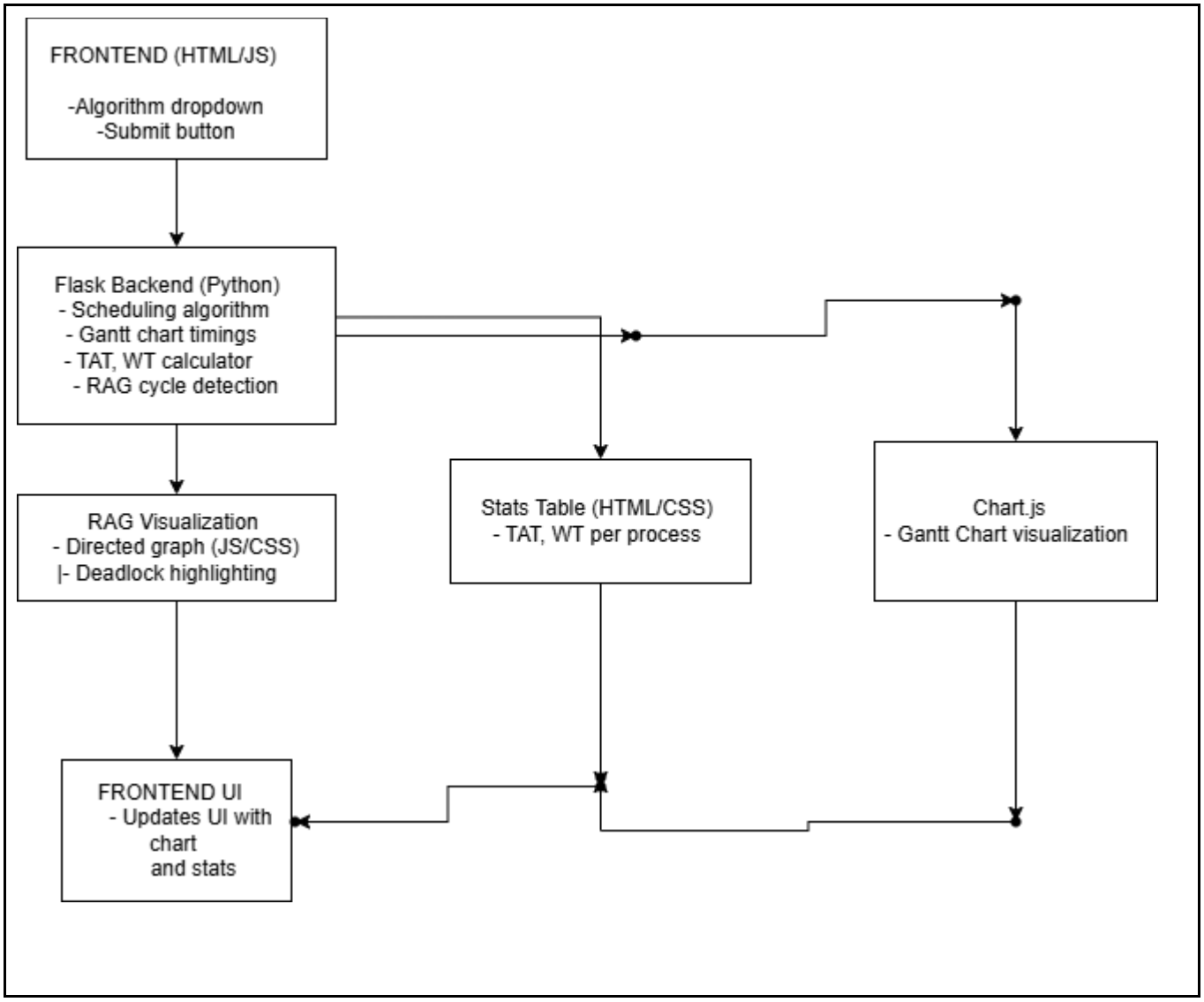
- **Tech Stack: Flask (Python)**
- **Responsibilities:**
  - **Handle scheduling algorithm logic based on user input**
  - **Return process execution order and time metrics**
  - **Analyze RAG structure to detect cycles (deadlocks)**

Visualization:

- **CPU Scheduling:**
  - **Gantt chart using Chart.js showing start and end time for each process**
  - **Table for TAT, WT, RT**

- **Deadlock Detection:**
  - **Render RAG using directed graphs**
  - **Visual indication (red) for processes involved in a deadlock**

System Architecture (High Level Diagram):



Project Outcome / Deliverables :

The final outcome of this project will be a polished, deployed web application that:

- Allows users to simulate CPU scheduling using custom inputs.
- Supports four key scheduling algorithms.
- Provides real-time visual feedback with Gantt charts.
- Lets users model RAGs and detect deadlocks visually.
- Calculates and displays performance metrics.

- Offers a responsive and clean user interface usable on desktop and mobile browsers.

The project will be deployed on a cloud platform like Vercel making it accessible without installation or setup.

Assumptions :

We assume that:

- All process and resource inputs are provided in valid formats (non-negative integers).
- The number of processes and resources remains within a manageable range (e.g., up to 20 each) for optimal UI performance.
- Priority scheduling will initially be implemented as non-preemptive. Preemptive logic may be added later.
- Round Robin quantum time is a fixed value provided by the user.
- There is negligible context switching time, and it is ignored in calculations.
- Deadlock detection assumes a well-formed RAG (no missing or invalid edges).
- The frontend will be used on modern web browsers with support for JavaScript and HTML5.
- Users understand basic scheduling terminology (e.g., TAT, WT, RT).

References:

- CPU Scheduling: https://www.geeksforgeeks.org/cpu-scheduling-in-operating-systems/
- Gantt Chart with Chart.js: https://www.chartjs.org/docs/latest/
- Resource Allocation Graph: https://www.geeksforgeeks.org/resource-allocation-graph-rag-in-deadlock/
- Flask Documentation: https://flask.palletsprojects.com/
- Deadlock Detection Algorithms: https://www.studytonight.com/operating-system/deadlock-detection
- FCFS Scheduling: https://www.programiz.com/os/fcfs-scheduling
- Round Robin: https://www.tutorialspoint.com/round-robin-scheduling-in-operating-system
- Chart.js Gantt Implementation Example: https://codepen.io/
- Operating System Concepts Book by Silberschatz, Galvin