Agentic AI Solution for Smart Classroom Management

Project Report

Names:

Chaturthi Naik (24P0620004)

Tanaya Chari (24P0620013)

Class: MSc. AI (II)

1. Objective of Work (Use Case Description)

The **Smart Classroom Agent** system is designed to automate and optimize the academic administration process within an educational institution.

Core Use Case: Automated scheduling, room allocation, and conflict resolution for university lectures.

Problem Solved	**Agentic Solution**
Scheduling Conflicts	Agent automatically checks and flags conflicts across faculty schedules and room bookings using time-overlap logic.
Resource Underutilization	Agent allocates the smallest sufficient room based on enrollment, saving large classrooms for higher-capacity needs (optimization).
Manual Data Retrieval	Provides instant, filtered access to current schedules and real-time faculty availability via dedicated finder tools.

The solution serves **Faculty, Students, and Academic Administrators**, ensuring operational efficiency and minimizing human error in complex timetable creation.

2. Why Agentic AI Solution Instead of Workflow Automation?

While simple Python scripts or basic workflow automation (Directed Acyclic Graphs, or DAGs) can handle linear tasks (e.g., send notification $\rightarrow$ update status), they fail at the complexity inherent in scheduling.

Feature	**Agentic System (LangGraph Model)**	**Simple Workflow (Automation)**
Logic Type	**Conditional and Cyclic (Non-Linear)**	Sequential ($\text{Step}_1 \rightarrow \text{Step}_2 \rightarrow \text{End}$)
Decision-Making	Context-aware decision nodes (e.g., "Check for overlap," "Select optimal room").	Fixed, pre-defined rules.
State Management	**Explicit, persistent State** (via Firestore) that is updated by each Node.	Implicit, often lost between steps.

The system requires an Agentic approach because scheduling demands **iterative problem-solving** (e.g., if Room A fails, try Room B, if Room B fails, flag 'No Room'). The **LangGraph** architecture provides the necessary structural primitives (Nodes and Conditional Edges) to model this dynamic, reflective decision process.

3. Functionality of the Agentic AI

This is currently implemented as a **Single Agent, Code-Based Simulation**.

- **Agent Name:** Classroom Allocation Agent
- **Type:** Single Agent (Text-Based Logic Simulation)
- **Key Functionalities (Nodes):**
  1. **Conflict Check Node:** Checks for conflicts involving the requesting faculty and the requested time slot using robust time-overlap calculations.
  2. **Allocation Node:** Iterates through available rooms, filtering by capacity, equipment, and current time occupancy, then selects the most appropriate room.
  3. **Tool/Finder Functionalities:** The system exposes two major Agent Tools: the **Quick Classroom Finder** and the **Faculty Availability Checker**, both of which use the agent's core conflict logic.

4. LLM Used

The current deployed system is a **simulation** using deterministic JavaScript code to represent the decision-making of an Agent. **No live LLM API calls are currently being made.**

Aspect	**Justification (for the Simulation)**
LLM Choice (Hypothetical)	**Gemini 2.5 Flash**
LLM Role	The LLM would replace the `checkConflictNode` and `allocateClassroomNode` logic, providing JSON output for allocation decisions.

5. Agentic AI Framework Used and Introduction

The architectural design of this project is based on **LangGraph**.

Framework	**Introduction & Role**
LangGraph (Simulation)	LangGraph extends the linear workflow of LangChain by allowing developers to represent agents as a **Stateful Graph** of Nodes and Edges. In this application, the core runAgentWorkflow function precisely mimics this state-based routing.
Strengths	1. **Explicit State:** The State object (represented by the pending schedule and external Firestore data) is central. 2. **Cycles & Branching:** Allows implementation of sophisticated conditional logic (e.g., IF conflict $\rightarrow$ LOG CONFLICT). 3. **Modularity:** Each step (Node) is a discrete, testable function.
Limitation	The current implementation relies on deterministic code rather than true LLM-based reasoning, making it a powerful framework demonstration but not a fully autonomous LLM Agent.

6. External Data Resources Used

Resource	Description and Role
Firebase Firestore	Acts as the **Memory** and **External Knowledge Base** for the Agent. All scheduling and room inventory data are stored in the cloud.
RAG Protocol / MCP	**Not Used.** RAG (Retrieval-Augmented Generation) is typically used when querying unstructured data with an LLM. Since the data is highly structured (JSON/tables) and the decision logic is code-based, these protocols were not necessary.
How Data is Used	Data is read via real-time listeners (onSnapshot) which provide the Agent with the current global schedule state—allowing it to perform real-time **lookups** for conflict and room availability checks.

7. If Multiple Agents, Agent-to-Agent Protocol Used

This system is designed as a **Single Agent** (the Classroom Allocation Agent).

- **Protocol: N/A.** The design prioritizes centralized control over distributed collaboration. A future multi-agent system (e.g., a "Faculty Agent" interacting with a "Resource Agent") would typically use a **Blackboard Pattern** for communication, where agents read and write to a central shared state (the Firestore database).

8. Prompts Used and Effectiveness

Since the application uses deterministic JavaScript logic in place of an LLM, **no external prompts were required or used.**

- **Prompt Engineering: N/A.** The Agent's behavior is dictated entirely by JavaScript conditional statements (`if/else`), ensuring 100% reliable and predictable outcomes (a necessary trade-off for a guaranteed-functional simulation).

9. Implementation Details and Code Explanation

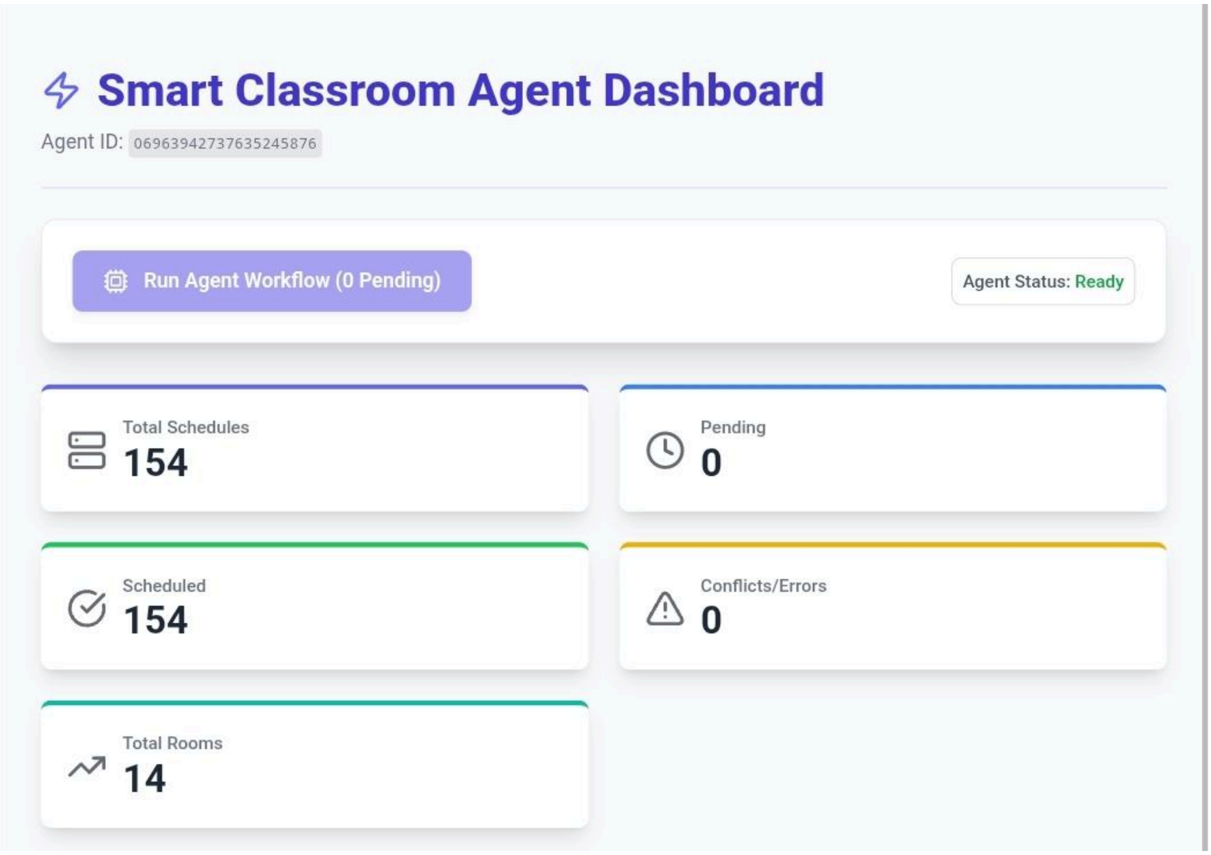
The solution is implemented using modern web technologies to create a single, efficient, and responsive application.

- **Technology Stack:** React (UI Framework), Tailwind CSS (Styling), Firebase (Persistence & Authentication).
- **Vibe Coding:** Yes, the project utilizes clean, modern, single-file component practices within React (JSX/TSX).
- **Code Explanation Highlights:**
    - **Data Structures:** Schedules and Classrooms are loaded from embedded JSON data (simulating a permanent configuration) into Firestore upon the first run.
    - **`checkOverlap` Utility:** This is a crucial function that translates complex time ranges (e.g., `10:30 - 12:30`) into minutes to correctly determine if two time blocks conflict, even if one is nested inside the other.
    - **Agent Logic (`runAgentWorkflow`):** This function implements the LangGraph logic, iterating through pending requests and using sequential **Node** execution (Conflict $\rightarrow$ Allocation) followed by a **Batch Write** to update Firestore, simulating a single, atomic, state-driven transaction.

10. Screenshots of Dashboard

The application provides a comprehensive dashboard view:

- **Dashboard Cards:** Real-time statistics on total, pending, scheduled, and conflict counts.
- **Agent Control:** A main button to trigger the runAgentWorkflow and a status line showing the last action.
- **Finder Tools:** Dedicated sections for **Quick Classroom Finder** and **Faculty Availability Check** (both using the core conflict logic).
- **Timetable Filter:** Allows users to filter schedules by **Program** and **Semester**.
- **Logs Panel:** A scrollable window showing a history of the Agent's decisions.



Smart Classroom Agent Dashboard

Agent ID: 06963942737635245876

Agent Status: Ready

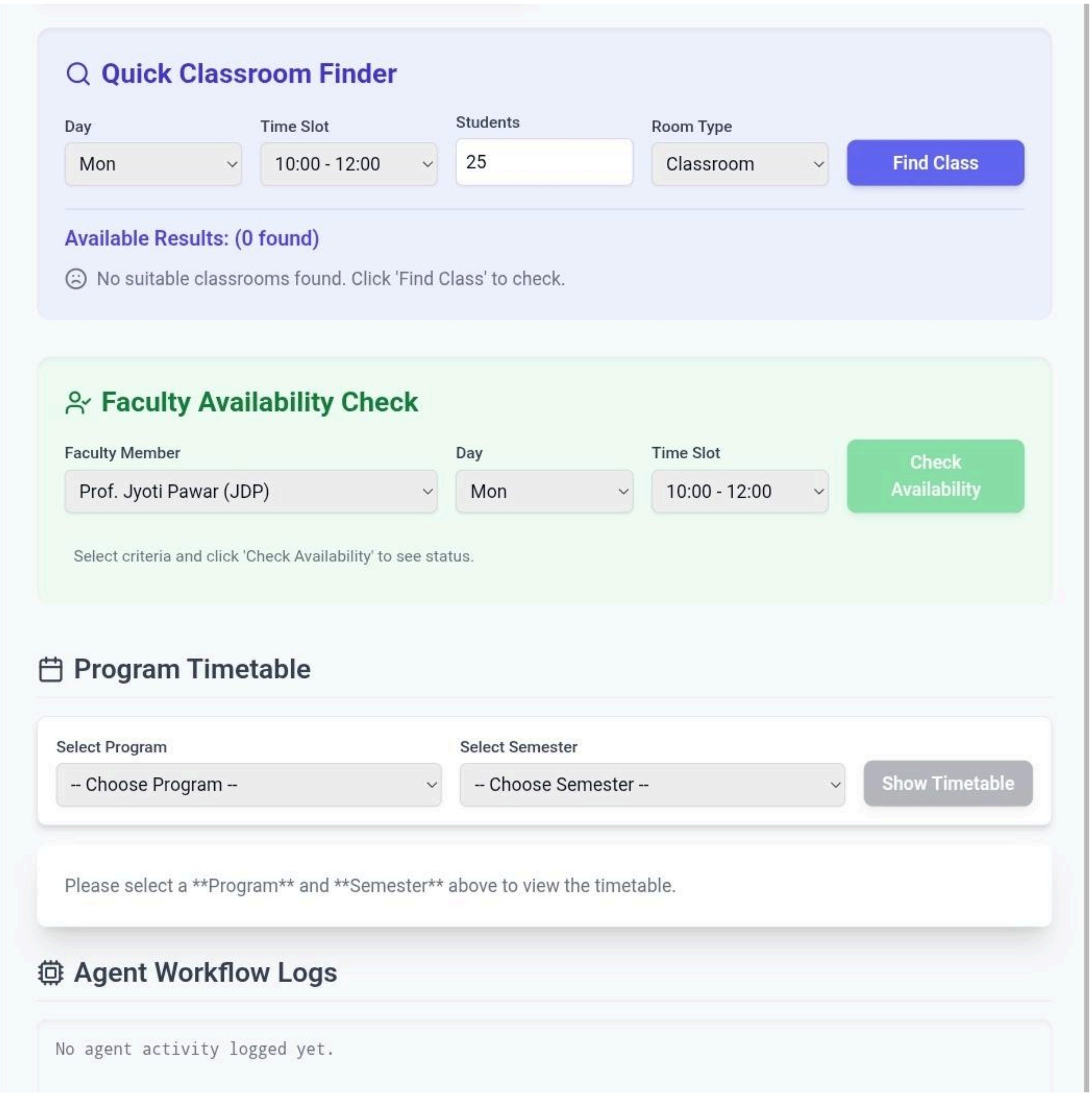
Total Schedules
154

Pending
0

Scheduled
154

Conflicts/Errors
0

Total Rooms
14



Day
Mon

Time Slot
10:00 - 12:00

Students
25

Room Type
Classroom

Available Results: (0 found)

No suitable classrooms found. Click 'Find Class' to check.

Faculty Member
Prof. Jyoti Pawar (JDP)

Day
Mon

Time Slot
10:00 - 12:00

Select criteria and click 'Check Availability' to see status.

Select Program
-- Choose Program --

Select Semester
-- Choose Semester --

Please select a **Program** and **Semester** above to view the timetable.

```
No agent activity logged yet.
```

11. References

1. LangChain Documentation (Conceptual basis for component modularity).
2. LangGraph Documentation (Architectural model for stateful, cyclic workflows).
3. Firebase Firestore Documentation (Implementation of persistence, memory, and real-time listeners).
4. Modern web development practices: React functional components and Tailwind CSS.