Final Project Roadmap.

Project Roadmap: University Timetable Scheduling System

Stakeholders (The Actors)

- **System Admin:** Sets up the "physical world" (Blocks, Rooms), manages system health, runs the schedule generation, and handles data imports (including who teaches what).
- **HOD (Head of Department):** Approves faculty constraints, defines "Elective Buckets," and reviews the schedule for workload fairness (Auditor Role).
- **Faculty:** Inputs availability constraints (subject to approval) and views their personal schedule.

Epic 1: User Management & Access Control

Goal: Secure the system so only the right people can see or change data.

Story 1.1: Role-Based Access Control (RBAC)

Story: As a System Admin, I want to assign strict roles (Admin, HOD, Faculty), so that HODs only modify their own department's data and Faculty cannot change the master schedule.

Tasks:

- **Configure User Permissions:** Set up the system so that each role (HOD, Faculty, Admin) has access only to the specific buttons and data they need for their job. **(Developer)**
- **Secure Data Entry Points:** Build "checkpoints" that verify a user's role every time they try to view or change sensitive schedule information. **(Developer)**
- **Protect Admin Accounts:** Add a safety lock to prevent the accidental deletion of the main Administrator account. **(Developer)**

Story 1.2: Secure Authentication

Story: As a User, I want to log in securely using my institutional email, so that unauthorized users cannot access sensitive schedule data.

Tasks:

- **Enable School Email Login:** Connect the login page to the institution's existing email system (like Google or Microsoft) so users don't need new passwords. **(Developer)**
- **Keep Users Logged In:** Ensure the system "remembers" the user securely so they don't have to log in again every time they refresh the page. **(Developer)**
- **Prevent Session Timeouts:** Implement a feature that keeps the user's session active while they are working on long tasks, preventing data loss. **(DevOps)**

Story 1.3: Activity Audit Logging

Story: As an Admin, I want to view a log of critical modifications, so that I can maintain accountability for schedule changes.

Tasks:

- **Build a History Log:** Create a digital "History Book" that records important actions, such as who changed a room or who deleted a course. **(Developer)**
- **Automate Event Tracking:** Ensure the system automatically writes to this log the moment any critical data is changed. **(Developer)**
- **Verify Log Accuracy:** Test by making a change and checking if the "History Book" correctly shows who did it and when. **(Tester)**

Epic 2: Institutional Modeling & Data Ingestion

Goal: Digitize the college structure, student hierarchy, and constraints.

Story 2.1: Infrastructure & Spatial Locality

Story: As a System Admin, I want to define Rooms and assign them to specific Academic Blocks, so that the AI knows where classes are located for travel calculations.

Tasks:

- **Catalog All Rooms:** Create a complete list of every classroom and lab in the system, specifying which building (Block) they are in and exactly how many students they can seat. **(Developer)**
- **Define Walking Times:** Input the walking time (in minutes) between different academic blocks so the system understands travel constraints. **(Developer)**
- **Verify Room Details:** Review the room list to ensure every room is correctly linked to its building and has the correct capacity listed. **(Tester)**

Story 2.2: Transactional Bulk Data Import

Story: As an Admin, I want to upload CSV files for Faculty, Courses, and Teaching Assignments, so that I can map professors to courses in bulk without manual entry.

Tasks:

- **Provide Downloadable Templates:** Create standard CSV templates for "Faculty," "Courses," and **"Teaching Allocations"** (which maps Course_ID to Professor_ID) for users to download. **(Developer)**
- **Implement "Safe Upload" Check:** Build a safeguard that rejects the entire uploaded file if even a single row contains an error (e.g., assigning a non-existent professor), ensuring database integrity. **(Developer)**
- **Test Error Reporting:** Upload a file with a known error (typo in Course ID) to ensure the system rejects it and identifies the specific row. **(Tester)**

Story 2.3: Student Section & Batch Management

Story: As an Admin, I want to define Student Sections (e.g., "3rd Year CSE-A") and map courses to them, so that the scheduler knows which "Batch" of students must attend which set of classes.

Tasks:

- **Create Student Groups:** Set up a feature to define distinct student groups (like "Section A" or "Batch 1") and link them to the specific list of courses they need to take. **(Developer)**
- **Build Batch Assignment Tool:** Develop a simple "Batch Manager" screen where Admins can easily click to assign courses to specific student sections. **(Developer)**
- **Verify Group Schedules:** Check that assigning a course to "Section A" correctly makes it appear in that group's requirements list. **(Tester)**

Story 2.4: Room Attribute Tagging

Story: As an Admin, I want to tag rooms with specific attributes (e.g., "Computer Lab"), so that the scheduler doesn't assign a lab course to a standard lecture hall.

Tasks:

- **Create Room Labels:** Implement a system of "tags" (like "Has Computers," "Chemistry Lab," or "Projector") that Admins can attach to room profiles. **(Developer)**
- **Enforce Room Matching:** Set up a rule that strictly prevents the system from placing a "Lab Course" into any room that doesn't have the matching "Lab" tag. **(Developer)**
- **Test Misassignment Prevention:** Try to force a lab class into a standard classroom to verify the system blocks it. **(Tester)**

Epic 3: The Core Scheduling Engine (The "Brain")

Goal: The smart system that figures out the best schedule.

Story 3.1: Fundamental Hard Constraints (Resource Clashes)

Story: As the System, I want to prevent any double-booking of resources, so that the generated schedule is physically possible to execute.

Tasks:

- **Prevent Faculty Clashes:** Program a strict rule that makes it impossible for the system to assign one professor to two classes at the same time. **(Developer)**
- **Prevent Room Double-Booking:** Program a rule ensuring that no single room can host two different classes simultaneously. **(Developer)**
- **Prevent Student Overlap:** Program a rule ensuring that a specific student group is never expected to attend two different lectures at once. **(Developer)**

Story 3.2: Complex Hard Constraints (Academic Rules)

Story: As the System, I want to enforce academic structural rules for Labs and Electives, so that classes run as intended.

Tasks:

- **Link Lab Hours:** Program the system to treat lab sessions as "indivisible blocks" (e.g., if a lab starts at 10 AM, the 11 AM slot must also be reserved for it). **(Developer)**
- **Synchronize Electives:** Program the system to force all elective courses in a specific "Bucket" to be scheduled in the exact same time slot. **(Developer)**
- **Test Academic Rules:** Feed the system specific lab and elective scenarios to verify it respects these groupings. **(Tester)**

Story 3.3: Soft Constraints & Optimization

Story: As the System, I want to score schedules based on "Quality of Life" metrics, so that the AI prefers schedules that are convenient for humans.

Tasks:

- **Minimize Travel:** Program the AI to penalize schedules that force faculty or students to walk between far-apart buildings in back-to-back hours. **(Developer)**
- **Reduce Awkward Gaps:** Program the AI to avoid creating useless 1-hour gaps between classes, preferring compact schedules instead. **(Developer)**
- **Even Out Distribution:** Program the AI to spread classes out over the week, avoiding situations like having 3 Math classes on Monday and none on Tuesday. **(Developer)**

Story 3.4: Handling Unsolvable Schedules (Conflict Reporting)

Story: As an Admin, I want to see a "Conflict Report" if the Hard Constraints cannot be satisfied, so that I know why the schedule failed.

Tasks:

- **Enable Partial Solutions:** Ensure the system calculates the "best possible" schedule even if it can't find a perfect one, rather than just crashing. **(Developer)**
- **Show Unscheduled List:** Create a clear report that lists exactly which courses couldn't be scheduled and explains why (e.g., "Professor Smith has no open slots"). **(Developer)**
- **Verify Error Clarity:** Check that the error messages are easy for a non-technical Admin to understand. **(Tester)**

Epic 4: Advanced Workload Management & Policy

Goal: Ensure fairness and follow university rules.

Story 4.1: Role-Based Max Load (Compliance)

Story: As an Admin, I want to define strict maximum teaching hours per week, so that we comply with university contracts.

Tasks:

- **Create Workload Settings:** Build a settings menu where Admins can type in the maximum allowable teaching hours for different faculty roles. **(Developer)**

- **Trigger Overload Errors:** Program the system to show a clear error message if a generated schedule tries to assign a professor more hours than the limit. **(Developer)**
- **Test Compliance:** Intentionally try to over-schedule a professor to ensure the system catches the violation. **(Tester)**

Story 4.2: Workload Analytics Dashboard

Story: As a HOD, I want to view charts showing "Teaching Hours per Faculty," so that I can visually verify if the load is distributed fairly based on the imported assignments.

Tasks:

- **Calculate Total Hours:** Program the system to automatically add up the total teaching hours assigned to each faculty member. **(Developer)**
- **Visualize Data:** Create easy-to-read Bar Charts and Pie Charts that display these totals, allowing HODs to spot unfair workloads at a glance. **(Developer)**
- **Verify Chart Accuracy:** Compare the charts against the raw schedule data to ensure the numbers are correct. **(Tester)**

Story 4.3: Fatigue Management

Story: As the Scheduler Agent, I want to avoid scheduling more than 4 continuous hours of lectures for any single faculty, so that we prevent faculty burnout.

Tasks:

- **Program Burnout Prevention:** Teach the AI to recognize "bad" schedules where a professor works too many hours in a row and apply a penalty score to them. **(Developer)**
- **Add Priority Controls:** Allow the Admin to choose how important this rule is (High, Medium, or Low) depending on the semester's needs. **(Developer)**
- **Verify Break Insertion:** Run a test schedule to confirm that the system automatically inserts breaks for faculty with heavy loads. **(Tester)**

Epic 5: Visualization & Export

Goal: See the schedule clearly and share it.

Story 5.1: Master Interactive Timetable

Story: As an Admin, I want to view a "Master View" of the entire institution's schedule with filtering, so that I can audit the global schedule.

Tasks:

- **Build Master Grid:** Construct the main calendar view that displays the entire schedule in a grid format. **(Developer)**
- **Add Smart Filters:** Implement dropdown menus that allow users to hide everything except the specific "Block," "Department," or "Semester" they want to see. **(Developer)**

- **Optimize Performance:** Ensure the grid loads quickly and scrolls smoothly even when displaying thousands of classes. **(Tester)**

Story 5.2: Drag-and-Drop Manual Override

Story: As an Admin, I want to manually move a class with real-time validation, so that I can resolve specific edge cases.

Tasks:

- **Enable Drag-and-Drop:** Allow users to simply click, hold, and drag a class block to move it to a different time or room. **(Developer)**
- **Visual Conflict Indicators:** Program the system to turn the block "Red" or "Green" while dragging, instantly showing if the new spot is valid or invalid. **(Developer)**
- **Prevent Move Conflicts:** Ensure that if two admins are editing at the same time, one cannot overwrite a move the other just made. **(Developer)**

Story 5.3: Public Schedule Export (PDF)

Story: As an Admin, I want to export class schedules as high-quality PDFs, so that I can print them for notice boards.

Tasks:

- **Generate PDF Tool:** Develop a feature that converts the current schedule view into a downloadable PDF file. **(Developer)**
- **Format for Print:** Design the PDF layout to ensure it fits perfectly on standard paper and is easy to read. **(Developer)**
- **Verify Output:** Check the generated PDFs to ensure no classes are cut off or missing details. **(Tester)**

Story 5.4: Explainability Tooltips

Story: As an Admin, I want to hover over a scheduled class to see why the AI placed it there, so that I can understand the decision-making logic behind the schedule.

Tasks:

- **Store Decision Logic:** Save the AI's "thought process" (constraint score) for every class it places, creating a transparent audit trail. **(Developer)**
- **Create "Hover-to-Explain" UI:** Build a simple popup that appears on mouse-over, explaining "Placed here to avoid travel penalty," helping the Admin trust the system. **(Developer)**

Epic 6: System Resilience, Versioning & Rollover

Goal: Keep data safe and manage semester changes.

Story 6.1: Pre-Publish Integrity Check

Story: As an Admin, I want to click a "Verify Schedule" button that scans for double-bookings, so that I can be 100% sure the schedule is valid before publishing.

Tasks:

- **Build Final Scanner:** Develop a tool that scans every single slot in the finished schedule to catch any remaining conflicts. **(Developer)**
- **Create Visual Status:** Design a simple summary screen that shows a big "Green Check" if the schedule is safe, or a "Red Alert" if problems are found. **(Developer)**
- **Stress Test:** Intentionally create a conflict and run the scanner to ensure it catches the error. **(Tester)**

Story 6.2: Automated Backups

Story: As an Admin, I want to have the system automatically backup the database daily, so that we can recover from server failure.

Tasks:

- **Schedule Daily Saves:** Configure the server to automatically save a complete copy of the database every 24 hours without human intervention. **(DevOps)**
- **Create Restore Button:** Build a simple "Restore" interface that allows the Admin to reload a previous day's backup with one click. **(Developer)**
- **Drill Recovery:** Perform a test where data is deleted and then successfully restored from a backup. **(Tester)**

Story 6.3: Schedule Snapshots (Versioning)

Story: As an Admin, I want to save multiple versions of a generated schedule, so that I can compare them before publishing.

Tasks:

- **Enable Multiple Drafts:** Update the system to allow saving "Draft 1," "Draft 2," etc., simultaneously without overwriting each other. **(Developer)**
- **Build Version Switcher:** Add a dropdown menu to the main screen that lets users instantly switch between viewing different drafts. **(Developer)**
- **Verify Draft Independence:** Ensure that making changes to "Draft 1" does not accidentally change "Draft 2." **(Tester)**

Story 6.4: Asynchronous Job Management

Story: As an Admin, I want to see a progress bar after starting the scheduler, so that the browser doesn't freeze during the calculation.

Tasks:

- **Background Processing:** Move the heavy schedule calculations to a background process so the user can still use the interface while it runs. **(Developer)**
- **Graceful Failure:** Ensure that if the calculation crashes, the system updates the status to "Failed" instead of leaving the screen frozen forever. **(Developer)**

- **Load Testing:** Run the scheduler with a large dataset to ensure the progress bar updates smoothly and accurately. **(Tester)**

Story 6.5: End-of-Semester Rollover

Story: As an Admin, I want to "Archive" the current semester's data and reset course assignments, so that I can start the new semester with a clean slate.

Tasks:

- **Build "Archive & Reset" Workflow:** Create a specific tool that moves old data to a "History" folder and wipes the board clean, making the system ready for new students instantly. **(Developer)**
- **Add Safety Lock:** Force the user to type "CONFIRM" before deleting data, protecting the university from a catastrophic accidental click. **(Developer)**

Epic 7: Notifications

Goal: Keep everyone informed.

Story 7.1: Schedule Publication Alert

Story: As a Faculty, I want to receive an email when the final timetable is published, so that I know when and where I am teaching.

Tasks:

- **Connect Email Service:** Integrate the system with an email provider (like SendGrid) to enable sending messages. **(Developer)**
- **Automate Announcement:** Set up a trigger so that clicking the "Publish Schedule" button automatically sends an email blast to all faculty. **(Developer)**
- **Test Delivery:** Verify that emails are successfully received by faculty members. **(Tester)**

Story 7.2: Constraint Violation Warnings

Story: As an HOD, I want to receive an immediate warning if I manually drag a class into a slot that causes a conflict, so that I don't accidentally break the schedule.

Tasks:

- **Create Warning Pop-up:** Design a pop-up alert that appears instantly on the screen if a manual move creates a problem. **(Developer)**
- **Explain the Error:** Ensure the warning clearly states why the move is bad (e.g., "Warning: Travel time too short"). **(Developer)**
- **Test Real-time Alerts:** Drag a class into a conflict zone to confirm the warning appears immediately. **(Tester)**

Story 7.3: Deadline Reminders

Story: As an Admin, I want to send automated reminders to Faculty to submit their availability constraints, so that the schedule generation isn't delayed.

Tasks:

- **Add "Remind All" Button:** Create a button on the Admin dashboard that triggers reminder emails to faculty. **(Developer)**
- **Draft Reminder Email:** Write a clear email template asking faculty to log in and submit their preferences before the deadline. **(Developer)**
- **Targeted Sending:** Ensure the system only sends reminders to faculty who have not yet submitted their constraints. **(Tester)**