Project Aura

Part I: Strategic Blueprint

1.0 Executive Summary: The Winning Vision

Project Aura (Automated Review & Assurance) is the proposed solution to the "Automated Balance Sheet Assurance" challenge. The core vision is to create an intelligent AI agent that serves as a partner to the Adani Group's finance teams.1 Aura's primary mission is to absorb the repetitive, manual data validation and consolidation tasks, thereby transforming the role of human analysts from "data janitors" to strategic advisors.1 This frees them to focus on high-value activities like anomaly investigation and strategic interpretation.1
The hackathon prototype will deliver a compelling, end-to-end demonstration of this vision. The user journey will begin with the simple upload of a trial balance and end with a manager asking natural language questions to an interactive, AI-powered dashboard.1 This unified plan is the definitive roadmap to building this prototype by the November 8th deadline.

2.0 Deconstructing the Core Problem

The problem statement highlights a critical business bottleneck: finance teams managing over 1,000 entities are mired in manual, repetitive review processes.1 This creates operational risk, delays decision-making, and limits the strategic impact of the finance function. Aura is designed to directly address the distinct pain points of its key stakeholders:
Finance & Accounts Teams: Overburdened by manual GL account assignment, hygiene checks, and verification of supporting documents.1 Aura automates these tasks, elevating their role.
Leadership & Business Financial Controllers (FCs): Hindered by the latency of static, historical reports.1 Aura provides a real-time, conversational insight engine for dynamic, forward-looking analysis.1

3.0 The Unified & Validated Technology Stack

Both independent analyses converged on an identical technology stack, providing strong validation for this approach. The core principle is a Unified Python Stack to maximize development velocity by eliminating integration friction and cognitive overhead.2 This allows a single data object to flow seamlessly from ingestion to visualization.

Architectural Layer	Recommended Tool	Rationale & Justification
Data Ingestion & Consolidation	Pandas	The industry standard for programmatic data manipulation in Python. Superior speed and flexibility for the hackathon's scope compared to GUI-based tools like Power Query.[4, 5, 6]
Validation & Compliance Checks	Great Expectations (GX)	Provides "unit tests for data," a professional and robust method to implement required checks, such as ensuring the trial balance totals to nil.[7, 8, 1] This demonstrates a mature approach to data quality that will differentiate the project.[9, 10]
Continuous Learning Loop		

Scikit-learn
Ideal for the straightforward classification task of learning from user corrections. Its simple, consistent API makes it easy to train and retrain a model like a DecisionTreeClassifier to demonstrate the learning capability.[11, 12, 13, 1, 14]
Agentic Behaviour & Conversational Interface
LangChain
Explicitly designed for orchestrating workflows and building AI agents that use tools to perform actions.[15, 16] This directly implements the "agentic behaviour" requirement of a reporting assistant and is superior to alternatives like LlamaIndex for this specific, tool-based task.[17, 1]
Interactive Visualization & UI
Streamlit
The fastest way to build interactive data applications purely in Python.[2] It integrates seamlessly with the rest of the stack and is ideal for the rapid prototyping of dashboards and charts required by the hackathon, offering more flexibility than Power BI or Tableau.[3, 18, 1]

Part II: The Definitive 15-Day Execution Roadmap

This granular, day-by-day plan integrates the phased approach from the initial analysis with the agile sprint structure of the second. It provides a single, unified timeline with clear daily goals, task ownership, and deliverables.

Phase 1: Foundation & Data Pipeline (Days 1-3)

Goal: Establish a stable development environment and build the core data ingestion and consolidation pipeline.

Objectives

- Define canonical schemas for `trial_balance.csv` and `responsibility_matrix.csv`.
- Implement Pandas-based ingestion and consolidation (join on GL_Account).
- Stand up an initial Streamlit UI with uploaders and raw views.

Scope

- CSV-based ingestion (SAP extraction simulated via CSV for hackathon).
- Schema validation at load; basic type enforcement.

Deliverables

- Ingestion module (`src/data_ingestion.py`) and schemas documented.
- Sample datasets under `data/raw` and processed data artifact.
- Basic Streamlit app (`app.py`) with file uploaders and data preview.

Acceptance Criteria

- Team can upload both files and see a consolidated dataframe with responsibilities.
- Missing/invalid schema fields are clearly flagged in the UI.
  Day
  Date
  Key Tasks

Primary Owner(s)
Deliverable / Daily Goal
1
Oct 25
Setup Git repository, Python virtual environment. Collaboratively define and finalize data schemas for trial_balance.csv and responsibility_matrix.csv.
P4, All
A shared, working Git repository. Finalized schemas documented.
2
Oct 26
Develop the core data ingestion script using Pandas to read, join, and consolidate the two sample CSV files based on the 'GL Account' key.
P1
A Python script that successfully reads and consolidates the sample data into a single dataframe.
3
Oct 27
Build the initial Streamlit application. Implement file uploader components for the two input files and display the raw and consolidated dataframes in a simple table.
P3
A running Streamlit app where a user can upload files and see the resulting data table.

Phase 2: Validation & Automated Reporting (Days 4-6)

Goal: Integrate a professional data validation pipeline and generate the core analytical dashboard with visual reports.

Objectives

- Initialize Great Expectations (GX) with an Expectation Suite enforcing Debits = Credits and non-null critical columns.
- Implement core analytics: variance analysis, review status tracking, hygiene metrics.
- Render tabular and visual summaries (Plotly) in Streamlit.

Scope

- GX context and checkpoint wired into ingestion pipeline.
- Variance vs previous period (month/quarter) and SLA deviation metrics.

Deliverables

- Validation module (`src/data_validation.py`) with GX suite and checkpoint.
- Analytics module (`src/analytics.py`) for variance/status/hygiene.
- Dashboard section with at least bar, pie, and gauge components.

Acceptance Criteria

- Uploaded data automatically validated; PASS/FAIL clearly shown with details.
- Variance and review status charts render with accurate aggregations.

Day	Date	Key Tasks	Primary Owner(s)	Deliverable / Daily Goal
4	Oct 28	Initialize a Great Expectations (GX) Data Context. Create the first Expectation Suite to validate the trial balance, including a check that debits equal credits.1	P1	GX is integrated into the project; great_expectations directory exists and is configured.
5	Oct 29	Integrate the GX validation checkpoint directly into the data pipeline. Display the clear "Pass/Fail" validation results in the Streamlit UI.	P1, P3	The app now automatically runs data quality checks on uploaded data and shows the status.
6	Oct 30	Develop backend functions for variance analysis and review status tracking. Add at least two interactive charts (e.g., bar chart, pie chart) to the Streamlit dashboard using Plotly.	P1, P3	The dashboard displays key analytical charts visualizing the financial data as required.1

Phase 3: The Intelligence Layer - Learning & Agency (Days 7-9)

Goal: Implement the core AI features: the continuous learning loop and the agentic framework.

Objectives

- Train a baseline GL classification model and persist it.
- Build a user correction flow and feedback store.
- Prepare backend functions as LangChain Tools.

Scope

- Scikit-learn model (e.g., DecisionTreeClassifier) for classification demo.
- Feedback-driven retraining (incremental or full fit).

Deliverables

- ML module (`src/ml_model.py`) and feedback handler (`src/feedback_handler.py`).
- Tool wrappers (`src/langchain_tools.py`) for load/validate/analyze.

Acceptance Criteria

- Correction submitted in UI changes the next prediction after retraining.
- Tools callable independently with clear I/O contracts.
  Day

	Date	Key Tasks	Primary Owner(s)	Deliverable / Daily Goal
7	Oct 31	Train an initial Scikit-learn model for a simple GL classification task. Build the UI feature that allows a user to correct a misclassification.	P2	A saved model file (.joblib) and a UI element for submitting corrections.
8	Nov 1	Implement the re-training logic for the continuous learning loop. Refactor the backend functions (data loading, validation, reporting) into modular "Tools" for LangChain.	P2	The model can be retrained based on user input. A script successfully runs a LangChain agent to call at least one tool.
9	Nov 2	Instantiate the main LangChain agent. Define the orchestration logic that allows the agent to sequentially call the necessary tools to perform an end-to-end analysis.	P2	The agent can autonomously run the data loading, validation, and reporting workflow based on a simple prompt.

Phase 4: UI & Conversational Interface (Days 10-12)

Goal: Build a polished, interactive user interface, including the conversational chatbot.

Objectives

- Polish dashboard layout (tabs, columns, expanders) and add drill-down.
- Implement chat interface (`st.chat_input`) and connect to the agent.
- Add proactive insight generation and display.

Scope

- Streamlit UI; Plotly interactions; role-based demo login.
- LangChain agent orchestration of tools to answer queries.

Deliverables

- Agent module (`src/agent.py`) and insights engine (`src/insights.py`).
- Chat UI wired to agent; suggested prompts.

Acceptance Criteria

- User can ask NL questions and receive correct tabular/visual responses.
- Proactive insight (e.g., expense spike) shown on dashboard.

Day	Date	Key Tasks	Primary Owner(s)	Deliverable / Daily Goal
10	Nov 3	Refine the Streamlit dashboard layout using columns, tabs, and expanders for a clean UX. Implement drill-down functionality on at least one dashboard chart.[1]	P3	A polished, multi-section dashboard with enhanced interactivity.
11	Nov 4	Build the conversational chat interface in Streamlit (st.chat_input) and connect it directly to the LangChain agent developed in Phase 3.	P3, P2	A user can type a question in the UI, and the agent's text response is successfully displayed.
12	Nov 5	Expand the agent's capabilities to handle 3-4 distinct, pre-defined queries and return data or charts to the chat UI. Implement the "proactive insight" feature on the dashboard.[1]	P2, P3	The chatbot is fully functional for the demo scope. A proactive insight is shown on the dashboard.

Phase 5: Integration, Polish, & Pitch Prep (Days 13-15)

Goal: Ensure the prototype is stable and bug-free, and prepare a compelling final presentation.

Objectives

- End-to-end integration testing across multiple datasets.
- Performance, error handling, and UX polish.
- Pitch deck and demo rehearsal.

Scope

- Non-functional refinements; SMTP/ERP integration noted as roadmap.

Deliverables

- Tested, stable prototype; demo datasets; pitch deck and script.

Acceptance Criteria

- All P0/P1 issues closed; demo runs reliably in rehearsal conditions.

Day	Date	Key Tasks	Primary Owner(s)

Deliverable / Daily Goal
13
Nov 6
Conduct full end-to-end integration testing with multiple sample datasets. Create a bug list and prioritize critical fixes.
P4, All
A comprehensive list of bugs and inconsistencies is created and triaged.
14
Nov 7
Final bug fixing and UX polish (e.g., add loading spinners, improve text). Prepare clean, compelling demo data. Draft the pitch deck and presentation script.
All
A stable, near-final prototype. A first draft of the presentation is complete.
15
Nov 8
Conduct multiple full-team demo rehearsals to ensure timing and flow. Finalize the pitch deck and presentation script. Final Submission.
P4, All
A smooth, timed demo is ready. The pitch deck is finalized and the team is fully prepared for judging.

Team Roles and Responsibilities

The optimal four-person team structure, validated by both analyses, is as follows:
P1 - Backend & Data Lead: Owns the entire data pipeline, from the Pandas ingestion scripts to the Great Expectations validation integration and the analytical reporting functions.
P2 - AI & Agent Lead: Owns the intelligence layer. Responsible for the Scikit-learn continuous learning loop and the architecture of the LangChain agent, its tools, and its orchestration logic.
P3 - Frontend & UI Lead: Owns the complete user experience. Responsible for building and polishing the Streamlit application, including all layouts, visualizations, interactive elements, and the conversational chat interface.
P4 - Project Manager & Integrator: Owns the overall project plan and timeline. Manages the Git repository, leads the integration of backend, AI, and frontend components, and spearheads the final pitch preparation and demonstration.

Works cited
Problem2 (1).pdf
Introduction to Streamlit Dashboards | Prescience Decision Solutions, a Movate company, accessed November 1, 2025, https://prescienceds.com/introduction-to-streamlit-dashboards/
Data Analysis Efficiency: Power BI vs. Streamlit - ProCogia, accessed November 1, 2025, https://procogia.com/when-to-use-power-bi-vs-streamlit/