Prompt #4 - 23Nov2025 @ 4:00 PM

CONTEXT & GOAL

You have access to the PRD “Semester Volunteer Matching System (V1.0)” and a live Google Sheet that contains all data for Learners and Learning Peers.

The front-end UI for the TutorMatchScheduler is already built.

Your task is to design and implement the backend (API + matching logic + Sheets integration) so that it fully satisfies the PRD and the additional requirements below.

Tech stack & environment

- Choose the backend stack (language + framework) you are strongest with on Replit (e.g., Node/Express, Python/FastAPI, etc.).
- Constraints:
  - It must expose HTTP REST endpoints for the existing front-end to call.
  - It must integrate securely with the **Google Sheets API** for the live data source.
- Before coding, briefly state which stack you are choosing and why it fits this project.

Data source: Google Sheets (primary source of truth)

- Primary data source: a single live Google Sheet: `https://docs.google.com/spreadsheets/d/1pNHzIYH58a7zPTjexSQW6eGibYJGQYk5Yfv1pdNYdW8`
- This spreadsheet contains four worksheets:
  - `Requests`
  - `Learning Peers`
  - `Volunteer Class Schedule`
  - `Learner Class Schedule`
- The structure and required fields for each sheet are as follows (only use these fields in your models):
  - **Requests**: `Timestamp`, `Email Address`, `First name`, `Last name`, `Which specific course would you like help in?`, `Who is your instructor?`, `Instructor Match Required?`, `What is your section number?`
  - **Learning Peers**: `Email`, `Preferred Name`, `Last Name`, `Groups`, `Course Code 1`, `Instructor 1`, `Course Code 2`, `Instructor 2`, `Course Code 3`, `Instructor 3`, `Other Courses`

    - **Volunteer Class Schedule**: `MRU_EMAIL`, `First`, `MONDAY`, `TUESDAY`, `WEDNESDAY`, `THURSDAY`, `FRIDAY`
    - **Learner Class Schedule**: same 7 fields as Volunteer Class Schedule.

Google Sheets integration & models

- Implement a dedicated data access module (e.g., `googleSheetsData.*`) that exposes:
    - `loadRequests()`
    - `loadLearningPeers()`
    - `loadVolunteerSchedules()`
    - `loadLearnerSchedules()`
- Each function should:
    - Use the Google Sheets API to read from the correct worksheet in the spreadsheet above.
    - Use credentials and spreadsheet ID stored in Replit Secrets (environment variables, not hard-coded).
    - Return typed objects for each row, using only the required fields listed above.
- Ignore any rows where the email address = `chickey@test`.
- Parse and clean schedule fields (MONDAY–FRIDAY) as lists of time slots like `0830-0950; 1000-1120; 1300-1420`, representing times the participant **cannot** meet.
- Parse `Learning Peers.Other Courses` as a list of `CourseCode + Instructor` combinations, stripping letter grades like `(A)`, `(A-)`, `(B+)`, etc.

Data cleaning & instructor normalization

- Trim whitespace from all strings.
- Implement a normalization / grouping function for instructor names per course code, e.g.:
    - For a given course code, treat `Marina Elliot`, `Marina Elliott`, and `Dr. Elliott` as the same instructor cluster.
    - Treat clearly different names (e.g., `Professor Emeritus John Robertson`) as separate instructors even for the same course code.
- The field `Instructor Match Required?` is either `'Y'` or blank. If `'Y'`, the Learner must only be grouped with other Learners and a Learning Peer with the **same instructor** for that course.

Matching logic & constraints (from PRD – treat as hard rules)

- Implement the matching algorithm as described in PRD section 4.2 (hard constraints and optimization priorities).
  Refined PRD Based on Refined Pr…

- Hard constraints include at minimum:
    - Group size: 1–4 Learners per group, all in the same course code.
    - Volunteer load: ≤2 groups total per Learning Peer.
    - Availability window: 1-hour session between 08:00 and 20:00.
    - Travel buffer: at least 5 minutes before and after any class for all participants.
    - Instructor match: if `Instructor Match Required = 'Y'`, the Learner's instructor must match the Peer's instructor, and all group Learners must share that instructor.
- Optimization priorities:
    - Priority 1 (hard): maximize the total number of matched Learners.
    - Priority 2 (soft): where multiple options exist, prefer time slots within ~2 hours before or after existing classes for on-campus convenience.

API endpoints

- Implement REST endpoints consistent with the PRD user stories, including:
    - `POST /api/matching-runs`: ingest live data from Google Sheets, run the algorithm, store and return a summary (run ID, counts, timestamp).
    - `GET /api/matching-runs/:id/groups`: return all proposed groups (group info + Peer + Learners + 1-hour slot).
    - `POST /api/groups/:id/approve`: finalize a group and trigger a notification function that (for now) simulates sending a single email to all participants with the details.
    - `POST /api/groups/:id/reject`: reject and re-queue participants for future runs.
    - `GET /api/unmatched`: list unmatched Learners and Peers with constraint failure reasons, plus suggested alternative time/group when failure is due to a single constraint.

Authentication (build last)

- The PRD includes secure Admin login (US-001), but implement this **after** the core backend, matching, and Google Sheets integration are working.
- Until then, allow unauthenticated access to the Admin endpoints to simplify development, but structure routing so authentication middleware can be added later.

Development process

Before writing code:

1. Summarize your understanding of the data model, Google Sheets integration, constraints, and endpoints.
2. Propose a file/module structure and indicate which stack you will use.
   Then implement in small, testable steps: data access → schedule parsing → matching algorithm → API endpoints → finally, Admin authentication.