Project Scope and Plan:

Lock-in: Schedule Generator & Course Buddy

Mohamad Baghdadi & Reind Ballout

April 3, 2025

Contents

1 High-Level Overview

- **Goal:** Develop a system that accepts free-form text describing a user's day or week, extracts tasks, meetings, and course identifiers (with explicit priority cues), and uses user feedback to generate an optimized, prioritized schedule.
- **Enhanced Features:**
  - **Hybrid Planning Approach:** Users provide a weekly overview (serving as the scheduling backbone) and can further add daily details.
  - **Fixed Commitments:** Users can choose to sync fixed events (e.g., course schedules from Outlook/Google/iCloud) or input them manually.
  - **Dynamic Modifications:** After schedule generation, users rate the schedule (e.g., on a 1–5 scale) and provide a free-form modification prompt. This prompt is processed by IEP1 to extract structured actions, fetch the most recent JSON schedule, and apply the changes. The modified JSON is then sent to IEP2 for re-generation of an optimized schedule.
  - **Calendar Syncing:** Users opting to sync will trigger a separate calendar syncing service (a dedicated Docker image) that connects to external calendar providers.
  - **Future Adaptation:** Collected user feedback (ratings and modification actions) will later support reinforcement learning (RL) to fine-tune scheduling parameters.

2 Project Scope & Requirements

Business Pitch

- Users provide an initial free-form weekly description, which serves as the baseline schedule.
- Fixed commitments (e.g., classes) are either synced from external calendars or added manually.
- The Schedule Generator (via EEP #1) calls IEP #1 to extract tasks, meetings, course codes, and priority cues.
- After generation, users are prompted to rate the schedule (e.g., 1–5) and provide a modification prompt if needed.
- The modification logic in IEP #1 processes the prompt into structured actions, retrieves the most recent JSON schedule, applies the changes, and then forwards the modified JSON to IEP #2 for re-generation.
- Calendar syncing is performed by a dedicated calendar syncing service, which connects to Outlook, Google, or iCloud.

Technical Requirements

- Version control using Git with comprehensive documentation.
- MLOps pipeline (e.g., MLflow or Weights & Biases) for model tracking and experimentation.
- Comprehensive testing: unit, integration, and end-to-end.
- Containerization using Docker (each component is deployed as a separate Docker image).
- Cloud deployment with publicly accessible endpoints.
- Monitoring and alerting via Prometheus, Grafana, and related configuration files.

3 Component Breakdown

User Interface (UI)

- **Function:** Provides a web interface for user input and feedback.
- **Deployment:** Runs as a separate Docker image.
- **Interaction:** Sends requests to EEP #1 with the necessary context (e.g., "generate schedule" or "modify schedule").

EEP #1: Schedule Generator API

- **Function:** Acts as the gateway for schedule generation and modification.
- **Primary Endpoints:**
  - `/parse-tasks` — For initial parsing (calls IEP #1).
  - `/compile-schedule` — For schedule compilation (calls IEP #2).
  - `/sync-calendar` — For triggering calendar syncing (calls Calendar Syncing Service).
- **Additional Logic:**
  - After schedule generation, prompts users to rate the schedule (1–5) and provide a modification prompt.
  - For modification requests, EEP #1 forwards the request (with a context flag) to IEP #1.

IEP #1: Task Parsing, Extraction, and Modification Processing

- **Function:** Processes free-form weekly input to extract tasks, meetings, and course codes; also processes modification prompts.
- **Details:**
  - Uses an LLM (e.g., GPT-3.5 Turbo) with context-aware prompt templates.
  - When called with context "parsing," extracts scheduling components.
  - When called with context "modification," extracts structured actions to update the existing schedule JSON.

IEP #2: Schedule Compilation and Prioritization

- **Function:** Generates an optimized, time-ordered schedule from the structured JSON.
- **Inputs:**
  - Baseline schedule JSON (from IEP #1) or a modified version after user feedback.
  - MCQ feedback (user ratings).
- **Technology:** Implements scheduling heuristics and optimization algorithms considering deadlines, durations, and priorities.

Calendar Syncing Service

- **Function:** Synchronizes the finalized schedule with external calendar systems (Outlook, Google, iCloud).
- **Deployment:** Runs as a separate Docker image.
- **Endpoint:** Provides an endpoint (e.g., `/sync-calendar`) that EEP #1 calls to perform calendar syncing.

4 Security Considerations

- **Input Validation & Sanitization:** Validate and sanitize free-form text inputs at all endpoints.
- **Content Filtering:** Implement filtering to ignore or flag irrelevant/malicious content.
- **Rate Limiting and API Gateways:** Use rate limiting and API gateways to prevent abuse.

5 Docker & Deployment

- **Docker Images (5 Total):**
  - `Dockerfile.UI` — For the User Interface.
  - `Dockerfile.iep1` — For IEP #1 (Task Parsing, Extraction, and Modification Processing).
  - `Dockerfile.iep2` — For IEP #2 (Schedule Compilation and Prioritization).
  - `Dockerfile.eep1` — For EEP #1 (Schedule Generator API: endpoints `/parse-tasks`, `/compile-schedule`).
  - `Dockerfile.calendar` — For the Calendar Syncing Service (endpoint `/sync-calendar`).
- **docker-compose.yml:** Defines services for `UI`, `iep1`, `iep2`, `eep1`, and the Calendar Syncing Service, and configures inter-container communication.
- **Cloud Deployment:** Images are pushed to a container registry and deployed on AWS, Azure, or GCP with publicly accessible endpoints.

6 Workflow & MLOps

- **Data & Preprocessing:** Organize sample data (weekly/daily descriptions) in a `data/` folder.
- **Model Training:** Develop and fine-tune NLP/LLM models, tracking experiments with MLflow or Weights & Biases.
- **CI/CD:** Utilize GitHub Actions (or similar) for automated building, testing, and deployment.
- **Dynamic Schedule Updates:** IEP #2 is triggered by the initial output from IEP #1 and re-invoked after user modification feedback.

7 Monitoring, Alerting & Quality Assurance (QA)

- **Monitoring and Alerting:**
  - Deploy Prometheus (using a `prometheus.yml` file) to collect metrics such as CPU usage, memory, and API response times.
  - Use Grafana for dashboard visualization and alert rule configuration.
- **Quality Assurance (QA):**
  - Unit tests for each module in IEP #1 and IEP #2.
  - Integration tests to ensure EEP #1 correctly invokes IEP #1, IEP #2, and the Calendar Syncing Service.
  - End-to-end tests simulating:
    - Weekly input parsing (`/parse-tasks`).
    - Schedule generation with MCQ feedback and modification prompt processing.
    - Modification processing that fetches and updates the latest JSON schedule.
    - Schedule re-generation (`/compile-schedule`).

    * Calendar synchronization (`/sync-calendar`).

8 Testing Strategy

- **Unit Tests:** Validate core functionalities within IEP #1 and IEP #2.
- **Integration Tests:** Ensure that EEP #1 properly interacts with IEP #1, IEP #2, and the Calendar Syncing Service.
- **End-to-End Tests:** Simulate the complete workflow:
  - Initial weekly input parsing (`/parse-tasks`).
  - Schedule generation with user feedback (rating and modification prompt).
  - Modification processing to update the latest JSON schedule.
  - Schedule re-generation (`/compile-schedule`).
  - Calendar synchronization (`/sync-calendar`).

9 Implementation Timeline

1. **Week 1:**
   - Set up the Git repository, define data structures, and create initial Dockerfiles.
   - Develop the UI as a Docker image.
2. **Week 2:**
   - Develop EEP #1 along with IEP #1 and IEP #2.
   - Implement parsing of free-form weekly input, MCQ prompting, and schedule generation.
3. **Week 3:**
   - Implement modification processing in IEP #1: process user ratings and modification prompts, fetch and update the latest JSON schedule.
   - Develop the Calendar Syncing Service with an endpoint for connecting to external calendars.
4. **Week 4:**
   - Finalize integration, enhance testing, monitoring, and load testing.
   - Prepare for the final demo/presentation.

10 Project Directory Structure

Below is a sample directory tree outlining the organization of the revised project:

```
.
        README.md
        docker-compose.yml
        data/
              sample_day.txt        % Free-form weekly/daily
↪ descriptions
        monitoring/
              prometheus.yml        % Prometheus configuration file
              alert_rules.yml       % Alert rules for Prometheus
        UI/
              app.py                % User Interface application
              Dockerfile.UI
              requirements.txt
        EEP1/
```

```
            app.py                % Schedule Generator API (
↪ endpoints: /parse-tasks, /compile-schedule, /sync-calendar)
            Dockerfile.eep1
            requirements.txt
        IEP1/
            parser.py             % Extracts tasks, meetings, course
↪  codes; processes modification prompts
            Dockerfile.iep1
            tests/
                test_parser.py
        IEP2/
            scheduler.py          % Generates an optimized, time-
↪ ordered schedule using parsed JSON and MCQ feedback
            Dockerfile.iep2
            tests/
                test_scheduler.py
        CalendarSync/
            sync.py               % Syncs finalized schedules with
↪ external calendar systems (Outlook, Google, iCloud)
            Dockerfile.calendar
            tests/
                test_calendar_sync.py
```

11 Final Note

This document outlines a refined microservices architecture for the Lock-in project, featuring:

- **Schedule Generation (EEP #1):**
  - **IEP #1:** Task Parsing, Extraction, and Modification Prompt Processing.
  - **IEP #2:** Schedule Compilation and Prioritization (incorporating MCQ feedback and modified schedule JSON).
- **Calendar Syncing Service:** A dedicated service that provides an endpoint (`/sync-calendar`) for connecting to external calendar systems (Outlook, Google, iCloud).
- **User Interface (UI):** A separate Docker image that handles user interaction.
- **User Feedback Flow:** After each schedule generation, the user rates the schedule (1–5) and provides a modification prompt. IEP #1 processes the prompt into structured actions, retrieves and updates the latest JSON schedule accordingly, and forwards the modified JSON to IEP #2 for re-generation.
- **Security:** Comprehensive input validation, sanitization, filtering, and rate limiting.
- **Deployment & MLOps:** Containerization with Docker (5 images total), CI/CD pipelines, cloud deployment, and monitoring via Prometheus/Grafana.

End of Document.