Project Scope and Plan:
Lock-in: Schedule Generator & Course Buddy

Mohamad Baghdadi & Reind Ballout

March 27, 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 additional user feedback to generate an optimized, prioritized schedule.
- **Enhanced Features:**
  - **Hybrid Planning Approach:** Users provide a weekly overview (as a scheduling backbone) and can further add daily details.
  - **Fixed Commitments:** On first use, users choose to sync fixed events (e.g., course schedules from Outlook/Google Calendar) or manually input them.
  - **Dynamic Modifications:** Mid-week modifications are handled via a dedicated external endpoint (EEP #3: Modification Engine) as well as through a manual UI for direct schedule editing.
  - **Calendar Syncing:** Finalized schedules are synchronized with external calendars.
  - **Course Buddy Integration:** Detected course codes are automatically forwarded to a Course Buddy module for study support.

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 manually added.
- The Schedule Generator (EEP #1) uses IEP #1 to extract tasks, meetings, course codes, and priority cues.
- MCQ feedback refines scheduling in IEP #2, generating an optimized, time-ordered schedule.
- Mid-week, users can modify their schedule using a text-based modification prompt via a new external endpoint (EEP #3) or via manual adjustments.
- The updated schedule is synchronized with external calendar systems (Outlook/Google Calendar).
- Course Buddy (EEP #2) receives course identifiers to provide study support through summarization (IEP #3) and diagnostic quiz generation (IEP #4).

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 (now 7 images in total, with a new image for the Modification Engine).
- Cloud deployment with publicly accessible endpoints.
- Monitoring and alerting via Prometheus, Grafana, and related configuration files.

3 Component Breakdown

EEP #1: Daily Schedule Generator

- **Function:** Accepts free-form text input (weekly overview plus daily details) and generates a structured baseline schedule.
- **Primary Endpoints:**
  - `/parse-tasks` — Invokes IEP #1 to extract tasks, meetings, course codes, and explicit priorities.
  - `/compile-schedule` — Invokes IEP #2 to generate an optimized schedule based on parsed data and MCQ feedback.
  - `/sync-calendar` — Synchronizes the finalized schedule with external calendars.
- **User Interface:**
  - A text box for entering the weekly overview.
  - Options to either sync fixed commitments (e.g., course schedules) from external calendars or manually add them.
  - MCQ prompts to capture productivity/focus feedback.
  - A visual display of the generated schedule.

IEP #1: Task Parsing and Extraction

- **Function:** Processes the free-form input to extract:
  - Tasks (e.g., "Contract Review", "PHYS201 Study").
  - Meetings (e.g., "Q1 Financial Report" meeting).
  - Course codes (e.g., MATH201, PHYS201).
  - Explicit priority cues.
- **Technology:**
  - Uses an LLM (e.g., GPT-3.5 Turbo) or similar NLP models to parse free-form text.
  - Applies rule-based heuristics and post-processing to refine output.

IEP #2: Schedule Compilation and Prioritization

- **Function:** Generates an optimized, time-ordered schedule from the structured output of IEP #1 and user MCQ feedback.
- **Technology:**
  - Uses scheduling heuristics and optimization algorithms that consider deadlines, durations, and priorities.

EEP #3: Modification Engine

- **Function:** Processes mid-week modification prompts to update the existing schedule.
- **Primary Endpoints:**
  - `/modify-schedule` — Accepts natural language modification prompts (e.g., "Cancel my Thursday meeting and reschedule it for Friday").
  - `/update-schedule` — Supports manual schedule adjustments via a user interface.
- **Internal Workflow:**
  - An LLM parses modification prompts to produce structured change instructions.
  - The changes update a shared schedule store.
  - The scheduling engine (IEP #2) is re-triggered (via event-driven or periodic mechanisms) to recompile the schedule.

EEP #2: Course Buddy

- **Function:** Provides course-specific study support.
- **Primary Endpoints:**
  - `/upload-material` — For uploading course study material.
  - `/summarize-docs` — Invokes IEP #3 to generate concise summaries.
  - `/generate-diagnostic` — Invokes IEP #4 to generate diagnostic quizzes.
  - `/adjust-schedule` — (Optional) Allows Course Buddy to request schedule adjustments based on diagnostic performance.

IEP #3: Document Summarization

- **Function:** Processes uploaded study materials to produce concise summaries.
- **Technology:** Uses extractive or abstractive summarization models (e.g., T5 or BART).

IEP #4: Diagnostic Quiz Generation

- **Function:** Automatically generates diagnostic quizzes based on document summaries.
- **Technology:** Utilizes fine-tuned transformer models or prompt-based LLM techniques.

4 Security Considerations

- **Input Validation & Sanitization:**
  - Validate the format, length, and content of free-form text inputs at all external endpoints.
  - Sanitize inputs to remove or escape potentially dangerous content.
- **Content Filtering:**
  - The extraction logic is designed to ignore or flag irrelevant or malicious content.
  - Suspicious input is logged and/or discarded.
- **Rate Limiting and API Gateways:**
  - Implement rate limiting on all external endpoints.
  - Use an API gateway or WAF to filter known attack patterns.

5 Docker & Deployment

- **Docker Images (7 Total):**
  - `Dockerfile.iep1` — For IEP #1 (Task Parsing and Extraction)
  - `Dockerfile.iep2` — For IEP #2 (Schedule Compilation and Prioritization)
  - `Dockerfile.iep3` — For IEP #3 (Document Summarization)
  - `Dockerfile.iep4` — For IEP #4 (Diagnostic Quiz Generation)
  - `Dockerfile.eep1` — For EEP #1 (Schedule Generator API: endpoints `/parse-tasks`, `/compile-schedule`, `/sync-calendar`)
  - `Dockerfile.modification` — For the Modification Engine (EEP #3: endpoints `/modify-schedule` and `/update-schedule`)
  - `Dockerfile.eep2` — For EEP #2 (Course Buddy API: endpoints `/upload-material`, `/summarize-docs`, `/generate-diagnostic`, `/adjust-schedule`)
- **docker-compose.yml:** Defines services for `iep1`, `iep2`, `iep3`, `iep4`, `eep1`, `modification`, and `eep2` 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, study materials) 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 of Docker images.
- **Dynamic Schedule Updates:** The scheduling engine (IEP #2) is invoked initially via the parsing process and is re-triggered upon modification events from the Modification Engine (EEP #3) or manual updates via `/update-schedule`.

7 Monitoring, Alerting & Quality Assurance (QA)

- **Monitoring and Alerting:**
  - **Prometheus:** Deploy a `prometheus.yml` configuration file (in a `/monitoring` folder) to collect metrics such as CPU usage, memory consumption, API response times, and model inference times.
  - **Grafana:** Set up dashboards to visualize these metrics and configure alert rules (via an `alert_rules.yml` file) for anomaly detection.
  - **Additional Tools:** Use node exporters or cAdvisor for detailed container-level metrics.
- **Quality Assurance (QA):**
  - Comprehensive unit tests for each IEP (e.g., parsing, summarization, quiz generation).
  - Integration tests to verify that EEP #1 correctly invokes IEP #1 and IEP #2, and that EEP #2 interacts properly with IEP #3 and IEP #4, as well as ensuring the Modification Engine (EEP #3) correctly updates the schedule.
  - End-to-end tests simulating a complete workflow: initial weekly input, MCQ feedback, schedule generation, mid-week modifications (via both text-based and manual endpoints), and calendar syncing.
  - CI/CD pipelines (e.g., GitHub Actions) to run tests on every commit.

8 Testing Strategy

- **Unit Tests:** Validate core functionalities in each internal module.
- **Integration Tests:** Ensure external endpoints correctly invoke the internal modules (IEP #1, IEP #2, and the Modification Engine in EEP #3) and that EEP #2 interacts properly with IEP #3/IEP #4.
- **End-to-End Tests:** Simulate the complete workflow:
  - Initial weekly input (via `/parse-tasks`).
  - MCQ feedback and schedule compilation (via `/compile-schedule`).
  - Mid-week modifications using text-based (`/modify-schedule`) and manual updates (`/update-schedule`).
  - Calendar synchronization (`/sync-calendar`).
  - Course Buddy functionalities (`/upload-material`, `/summarize-docs`, `/generate-diagnostic`, `/adjust-schedule`).

9 Implementation Timeline

1. **Week 1:**
   - Set up the Git repository, define data structures, and create initial Dockerfiles.
2. **Week 2:**
   - Develop EEP #1 along with IEP #1 and IEP #2.
   - Implement parsing of free-form weekly input, MCQ prompting, and schedule compilation.
3. **Week 3:**
   - Develop the new Modification Engine (EEP #3):
     - `/modify-schedule` for text-based modifications.
     - `/update-schedule` for manual schedule adjustments.
   - Develop EEP #2 along with IEP #3 and IEP #4 for Course Buddy functionalities.
4. **Week 4:**
   - Finalize integration, including calendar syncing (`/sync-calendar`).
   - 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 project:

```
.
          README.md
          docker-compose.yml
          data/
                sample_day.txt        % Free-form weekly/daily
   ↪ descriptions
                sample_material.pdf   % Course study material
          monitoring/
                prometheus.yml        % Prometheus configuration file
                alert_rules.yml       % Alert rules for Prometheus
          EEP1/
                app.py                % Schedule Generator API (
   ↪ endpoints: /parse-tasks, /compile-schedule, /sync-calendar)
                Dockerfile.eep1
                requirements.txt
          IEP1/
                parser.py             % Extracts tasks, meetings, and
   ↪ course codes; checks for explicit priorities
                Dockerfile.iep1
                tests/
                    test_parser.py
          IEP2/
                scheduler.py          % Generates an optimized schedule
   ↪ using parsed data and MCQ feedback
                Dockerfile.iep2
                tests/
                    test_scheduler.py
          Modification/
                modify.py             % Processes text-based
   ↪ modification prompts using an LLM
```

```
               update.py           % Handles manual schedule
 ↪ adjustments from the UI
               Dockerfile.modification
               tests/
                   test_modification.py
         EEP2/
               app.py              % Course Buddy API (endpoints: /
 ↪ upload-material, /summarize-docs, /generate-diagnostic, /adjust-
 ↪ schedule)
               Dockerfile.eep2
               requirements.txt
         IEP3/
               summarizer.py       % Generates summaries from
 ↪ uploaded course material
               Dockerfile.iep3
               tests/
                   test_summarization.py
         IEP4/
             quiz_generator.py   % Creates diagnostic quizzes from
     ↪ summaries
             Dockerfile.iep4
             tests/
                 test_quiz.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 and Extraction (using an LLM for free-form weekly/daily input).
  - **IEP #2:** Schedule Compilation and Prioritization (incorporating MCQ feedback).
- **Modification Engine (EEP #3):**
  - `/modify-schedule`: For processing text-based modification prompts.
  - `/update-schedule`: For manual schedule adjustments via a UI.
- **Course Buddy (EEP #2):**
  - **IEP #3:** Document Summarization for uploaded course material.
  - **IEP #4:** Diagnostic Quiz Generation based on summaries.
- **Calendar Syncing:** External endpoint `/sync-calendar` to integrate with Outlook/-Google Calendar.
- **Security:** Comprehensive input validation, sanitization, filtering, and rate limiting.
- **Deployment & MLOps:** Containerization with Docker (7 images total), CI/CD pipelines, cloud deployment, and monitoring via Prometheus/Grafana.

End of Document.