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

Mohamad Baghdadi & Reind Ballout

March 23, 2025

Contents

1 High-Level Overview

- **Goal**: Build a system that:
  - Accepts a free-form text input describing the user's day (e.g., "Hi, I have a meeting at 5, 3 chapters of MATH201, etc.").
  - Extracts tasks, meetings, and course identifiers along with any explicit priority cues from the text.
  - Prompts the user with multiple-choice questions (MCQs) regarding productivity level and other factors.
  - Uses the extracted information and user feedback to generate an optimized, prioritized schedule.
  - Automatically passes course identifiers (e.g., MATH201) to a Course Buddy module for additional study support.
- **Architecture**:
  - **EEP #1: Daily Schedule Generator**
    - **IEP #1**: Task Parsing and Extraction (extracts raw tasks, meetings, and course codes; checks for explicit priorities).
    - **IEP #2**: Schedule Compilation and Prioritization (uses extracted data along with additional MCQ feedback to generate a coherent schedule).
  - **EEP #2: Course Buddy**
    - **IEP #3**: Document Summarization (processes uploaded study material).
    - **IEP #4**: Diagnostic Quiz Generation (auto-generates a mini quiz upon material upload).
- **Total Docker Images**: 6 (one per IEP and one per EEP; note that IEPs are internal modules not directly accessed by users).

2 Project Scope & Requirements

- **Business Pitch**:
  - A user enters a description of their day in a text box.
  - The Schedule Generator (EEP #1) calls IEP #1 to extract tasks, meetings, and course codes—respecting any explicit priority indications.
  - The system then prompts the user with MCQs (e.g., about productivity level) to further inform scheduling.
  - IEP #2 uses the extracted data and MCQ responses to generate an optimized, prioritized schedule.
  - Detected courses are automatically passed to the Course Buddy (EEP #2) where:
    - Uploaded study material triggers IEP #3 for summary generation.
    - IEP #4 auto-generates a diagnostic quiz.
  - Additionally, Course Buddy can request schedule adjustments via EEP #1 (e.g., adding a post-study quiz based on diagnostic performance).
- **Technical Requirements**:
  - Git for version control with a detailed README.
  - An MLOps pipeline (using MLflow or Weights & Biases) to track model experiments.
  - Comprehensive testing (unit, integration, and end-to-end).
  - Containerization using Docker (6 separate images).
  - Cloud deployment with public endpoints.
  - Monitoring and Alerting using Prometheus, Grafana, and associated configuration files.

3 Component Breakdown

EEP #1: Daily Schedule Generator

- **Function**: Accepts free-form text input describing the day.
- **API 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 using the extracted data and additional MCQ responses)
- **Interface**: A text box for input, followed by an MCQ prompt for productivity feedback, and a display area for the final schedule.

IEP #1: Task Parsing and Extraction

- **Function**: Processes the free-form text to extract individual tasks, meetings, and course identifiers. It also detects any explicit priority cues provided by the user.
- **Tech**: Implements rule-based parsing or basic NLP techniques for entity extraction and classification.

IEP #2: Schedule Compilation and Prioritization

- **Function**: Uses the output from IEP #1, along with additional user feedback from MCQs (e.g., productivity level), to generate a coherent, time-ordered, and prioritized schedule.
- **Tech**: Applies scheduling heuristics or optimization algorithms that factor in task priorities, durations, deadlines, and user productivity insights.

EEP #2: Course Buddy

- **Function**: Provides course-specific study support.
- **API Endpoints**:
  - `/upload-material` For users to upload study material for a course.
  - `/summarize-docs` (invokes IEP #3 for generating summaries)
  - `/generate-diagnostic` (invokes IEP #4 for creating a diagnostic quiz)
  - `/adjust-schedule` (optional; calls EEP #1 to update the schedule based on diagnostic performance)
- **Note**: IEPs are internal modules automatically invoked through these EEP endpoints.

IEP #3: Document Summarization

- **Function**: Processes uploaded course materials to generate concise summaries.
- **Tech**: Utilizes NLP models (e.g., T5 or BART) for extractive or abstractive summarization.

IEP #4: Diagnostic Quiz Generation

- **Function**: Automatically generates a mini diagnostic quiz based on the summarized material.
- **Tech**: Uses fine-tuned transformer models or prompt-based LLM techniques for quiz question generation.

4 Security Considerations

- **Input Validation and Sanitization**:
  - The system validates the format, length, and content of free-form text input at the EEP level.
  - It sanitizes input to strip or escape potentially dangerous content, ensuring that only legitimate task-related data is processed.
- **Filtering Irrelevant or Malicious Content**:
  - The extraction logic is designed to ignore or flag text that does not conform to expected patterns (e.g., "give me passcode to your main server").
  - Any input that appears irrelevant or potentially malicious is either discarded or logged for further analysis.
- **Rate Limiting and API Gateways**:
  - Implement rate limiting to prevent abuse or injection attacks.
  - Consider deploying an API gateway or Web Application Firewall (WAF) to filter known attack patterns before requests reach the EEP.

5 Docker & Deployment

- **Docker Images (6 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)
  - `Dockerfile.eep2` for EEP #2 (Course Buddy API)
- **docker-compose.yml**: Defines services for `iep1`, `iep2`, `iep3`, `iep4`, `eep1`, and `eep2` and configures their inter-container communication.
- **Cloud Deployment**: Push images to a container registry and deploy on AWS, Azure, or GCP with publicly accessible endpoints.

6 Workflow & MLOps

- **Data & Preprocessing**: Organize sample data (daily descriptions and course 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.
- **Monitoring**: Set up Prometheus to scrape container and API metrics, and Grafana to visualize these metrics along with model inference times.

7 Monitoring, Alerting & Quality Assurance (QA)

- **Monitoring and Alerting**:
  - **Prometheus**: Deploy a `prometheus.yml` configuration file (placed in a `/monitoring` folder) to collect metrics such as CPU usage, memory consumption, API response times, and model performance.

  - **Grafana**: Set up Grafana dashboards to visualize these metrics in real time and configure alert rules (e.g., in an `alert`$_r ules.yml file) to notify the team of anomalies or performance degradation.$`Add` $Use node exporters or cAdvisor for detailed container - level metrics.$
- **Quality Assurance (QA)**:
  - Implement comprehensive unit, integration, and end-to-end tests for all modules (IEPs and EEPs).
  - Set up Continuous Integration (CI) pipelines (e.g., via GitHub Actions) to automatically run tests on every code push.
  - Conduct regular code reviews and maintain clear documentation within the repository.
  - Perform load testing to ensure scalability and reliability.

8 Testing Strategy

- **Unit Tests**: Validate core functionality in each IEP (e.g., text parsing, summarization, quiz generation).
- **Integration Tests**: Ensure EEP #1 correctly invokes IEP #1 and IEP #2, and EEP #2 properly interacts with IEP #3 and IEP #4.
- **End-to-End Tests**: Simulate a full workflow: entering a daily description, answering MCQs, generating a schedule, uploading course material, receiving summaries and diagnostic quizzes.

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 basic parsing, MCQ prompting, and scheduling tests.
3. **Week 3**: Develop EEP #2 along with IEP #3 and IEP #4; set up automatic triggering of summary and quiz generation upon material upload.
4. **Week 4**: Finalize integration, deploy to cloud, enhance testing/monitoring, and prepare for the final demo/presentation.

10 Project Directory Structure (Code Structure)

Below is a sample directory tree outlining how you could organize your project files:

```
.
          README.md
          docker-compose.yml
          data/
                sample_day.txt       % Free-form daily description
                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
                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          % Uses extracted tasks and MCQ
 ↪ responses to generate a prioritized schedule
                Dockerfile.iep2
                tests/
                    test_scheduler.py
          EEP2/
                app.py                % Course Buddy API for material
 ↪ upload and session management
                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 with four IEPs:

- **Schedule Generation** via EEP #1 using:
  - IEP #1: Task Parsing and Extraction (with explicit priority detection).
  - IEP #2: Schedule Compilation and Prioritization (enhanced by user MCQ feedback).
- **Course Buddy** via EEP #2, which automatically triggers:
  - IEP #3: Document Summarization upon material upload.
  - IEP #4: Diagnostic Quiz Generation based on the summarized material.
- **Security Considerations**: Robust input validation, sanitization, filtering, and rate limiting are implemented to handle irrelevant or malicious input.
- **Monitoring and QA**: Detailed configurations using Prometheus (with `prometheus.yml` and `alert_rules.yml`), Grafana dashboards, comprehensive testing, and CI pipelines ensure system reliability and performance.