Poeas Design Document

Project Name: Scheduling Engine

TNPG: Poeas

Roster: Andy Shyklo, Jady Lei, Mr. Dillon

TARGET SHIP DATE: 6/6/2025

Program Overview:

“In this project, developers will write an algorithm that places students into course sections (aka a scheduling engine) based on the students need to graduate (requirements) and want to take (electives). Students will work with Mr. Dillon using anonymized Stuyvesant data. You'll learn some of the intricacies of the Program Offices/Registrar. This is not a feature that Talos currently has and would be used to more effectively program students.” (Dillon, April 29th via DTM Piazza)

Program Components:

1. Organize a schedule request per student
2. Create a readable table of students and their scheduled classes and periods.
3. Allow elective requests to supplement impossible lists.
4. Factor class sizes and class selectiveness into sorting out impossible schedules.

Recursion Organization:

- We iterate through each students schedule, and find the first matching schedule where all classes fit together, using recursion to add or remove classes from that schedule
- Students with conflicting schedules can use recursive functions for multiple students at once, creating an adaptive schedule (unschedules one student in preference for another’s perfect schedule).

Site Map:

N/A

Component Map:

Read csv

-> find course availability

-> create schedule

-> edit class table

-> modify schedules (for impossible results)

-> store created schedules

Database Organization:

Student Requests: List of Dictionaries

Course List: List of Dictionary

Availability List: 2D list of periods for requested course

Created Schedule: List of period numbers

Schedules: List of Dictionaries

APIS: None

Modules:

- CSV - necessary as our primary input of data
- Math - used to floor and with decimals for A/B schedules

Functions:

- **main()**
  - The main initializing function
  - Input: None
  - Output: (csvTextOfSchedules -> String, csvTextOfRoster -> String, csvTextOfSeats -> String)

- **_return_list_of_availability(student_request, course_info)**
  - Returns sorted/filtered list of courses and their sections that need to be scheduled.
  - input: (student_request -> Dictionary {StudentID: String, Course1: String, ...}, course_info -> List of Dictionaries [{CourseCode: String, SectionID: String, ...}, {...}])
  - output: availability -> ~4D List [[CourseCode, (SectionID, PeriodID, ScheduleID, AvailableSeats), ...], [CourseCode, ((first_SectionID, second_SectionID), (first_PeriodID, second_PeriodID), (first_ScheduleID, second_ScheduleID), AvailableSeats)], ...]

- **_check_schedule(studentid, availability)**
  - Wrapper function for recursive function _check_schedule_r. Makes actual schedules
  - input: (student_request -> Dictionary {StudentID: String, Course1: String, ...}, availability -> ~4D List [[CourseCode, (SectionID, PeriodID, ScheduleID, AvailableSeats), ...], ...])
  - output: _check_schedule_r

- **create_schedules(student_requests, class_list_in, student_requests_dictionary_in)**
  - Wrapper function to sort requests based on difficulty to schedule for _create_schedule.
  - input: (student_requests -> List of Dictionaries [{StudentID: String, Course1: String, ...}, {...}], class_list_in -> List of Dictionaries [{CourseCode: String, SectionID: String, ...}, {...}], student_requests_dictionary -> Dictionary {'1': [{StudentID: String, Course1: String, ...}], ...})
  - output: _create_schedules_r

- **_create_schedules_r(student_requests)**
  - Function to sort requests based on difficulty to schedule. Recursive
  - input: (student_request -> Dictionary {StudentID: String, Course1: String, ...} availability -> ~4D List [[CourseCode, (SectionID, PeriodID, ScheduleID, AvailableSeats), ...], ...])
  - output: _check_schedule_r

Task Breakdown:

- Andy: Project Manager
  - Coding to specification of Mr. Dillon
  - Modifying existing code to accommodate for schedule inflexibility
  - Debugging and special cases
  - Formatting code output and writing to csv files
- Jady:
  - Coding to specification of Mr. Dillon
  - Documentation on the code
  - Write in doubles
  - Optimizing availability and scheduling/unscheduling

FLOW CHART: