Software Requirements Specification

for

SENG 499: Company 2

Version 2.0 Approved

Prepared by Company 2

University of Victoria

May 29, 2022

Table of Contents

Revision History

Name	**Date**	**Reason For Changes**	**Version**
Spencer	2022/5/22	Added post-conditions, algorithm feature descriptions.	1.0
Nanami	2022/5/23	Drafted Introduction, Hardware Interfaces, Software Interfaces	1.1
Spencer	2022/5/23	Drafted pre- and post-conditions, product perspective	1.2
Kiana	2022/5/24	General edits, added to Algorithm requirements, functional and nonfunctional requirements.	1.3
Raphael	2022/5/25	Drafted functional requirements for Frontend and Backend systems; Edited Frontend-Backend API endpoints	1.4

Nanami	2022/5/25	Drafted User Interface section. General edits	1.5
Spencer	2022/5/25	Drafted design and implementation constraints, assumptions and dependencies	1.6
Kiana	2022/5/26	Added user diagram and data flow diagrams for the entire system and algorithms. General edits.	1.7
Everyone	2022/5/29	Final editing and formatting before submission	Final

1. Introduction

1.1 Purpose

This requirement document provides the specifications for a prototype university courses scheduler service and its underlying algorithms. The proposed university scheduler is intended for use by university administrators, and aims to automatically perform the time-consuming process of assigning courses to professors, and to time blocks.

1.2 Document Conventions

Unless otherwise specified, all frameworks and algorithms discussed in this document are free and open-source.

1.3 Intended Audience and Reading Suggestions

This document is primarily written for the Company 2 developers, SENG 499 teaching team and the stakeholders. Before reading this document, it is highly recommended to read the *Project Design II Project Specification* to have an overview of this project.

1.4 Project Scope

The main objective of this software system is to reduce the workload of university administrators by automatically producing a feasible schedule for SENG program courses with given constraints, which includes but is not limited to: professor availability, course capacity, and course dates and times. To also account for professor preferences, the system must have functionality that accepts inputs from professors. This system must be able to accept manual editing of generated schedules. The scope of this system is limited to courses required in the SENG program worksheet, excluding the technical electives and complementary electives.

Moreover, the algorithm's purpose is to schedule course times and assign professors to courses. Other possible functionality, such as assigning classrooms to courses, is beyond the scope of this project.

1.5 References

[1] S. Deshpande. "Automatically filling multiple responses into a Google Form with Selenium and Python." Available: https://medium.com/swlh/automatically-filling-multiple-responses-into-a-google-form-with-selenium-and-python-176340c5220d (accessed May 29, 2022).
[2] University of Victoria. "Software Engineering Program Planning Worksheet," Department of Engineering and Computer Science, University of Victoria, BC. 2022. Available: https://www.uvic.ca/ecs/assets/docs/program-planning/PPW-SENG.pdf (accessed May 29, 2022).

2. Overall Description

2.1 Product Perspective

The system is intended as a tool for the university administrators responsible for creating SENG course calendars. The current system is done manually, which is a slow process, and is susceptible to human errors. This system is intended to improve the current system of determining a course schedule for SENG. It will accelerate their existing workflow by automatically generating a schedule, given the unique constraints of each scheduling year.

2.2 Product Functions

The overall flow of the system is shown in Figure 1-1. The system will take the input from the university administrators and professor, perform a forecast of the Software Engineering Courses (SENG) class size, and return a feasible schedule for the following scheduling year based on the inputted constraints.

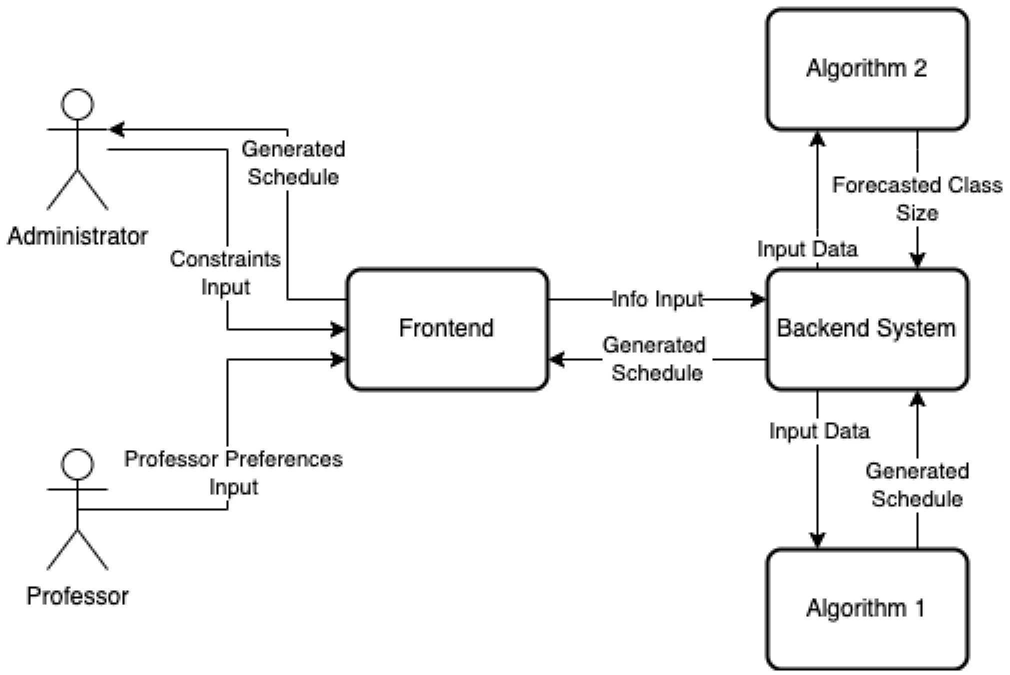


Figure 1-1: Overview of the system

2.3 User Classes and Characteristics

2.3.1 Primary Actor

Administrator: The primary actor, an administrator for the SENG department, is required to enter in the SENG courses determined to be scheduled for a given term. The administrator's interaction will be through the front-end using a web browser.

2.3.2 Secondary Actor

Professors: The secondary actors, the professors, are required to login to the system, and enter in their teaching information for the upcoming scheduling year. The professor's interaction will be solely through the front-end API, using a web browser. The professor will set their hard constraints for teaching the upcoming scheduling year, as well as soft constraints regarding their preferences and abilities.

2.3.3 Basic Flows

The basic scenario of the professor use case follows these steps:

1. The professors login and enter their hard and soft constraints for the upcoming scheduling year.
2. The frontend sends this information to the backend.
3. The backend stores this information in the database.

The basic scenario of the administrator use case follows these steps:

1. The administrator will enter course and professor constraints manually before generation.
2. The administrator will select the ”Generate Schedule” button.
3. The frontend sends this information to the backend.
4. The backend stores the information in the database and sends the formatted information to Algorithm 2.
5. Algorithm 2 predicts a class size for the given courses, and returns a list of the semesters and courses with their predicted class sizes to the backend.
6. Backend sends the inputted class size and constraints from the administrator and professors to algorithm 1.
7. Algorithm 1 accepts these inputs and generates a feasible schedule.
8. Algorithm 1 returns the generated schedule to the backend.
9. The backend returns the generated schedule to the frontend.
10. The frontend formats the generated schedule.
11. The administrator views the generated schedule.

2.4 Operating Environment

A backend Python REST-like server will operate in a Debian Linux environment. The system will support full sized screens (desktop) up to 4k resolution, preferably in a clean and cool environment.

2.5 Design and Implementation Constraints

The design and implementation of the system must be complete by Sunday, July 24, 2022. The behavior of the final product must conform to the pre- and post-conditions detailed in the following sections. Also, the historical data used to generate schedules will be static and entered into the system by developers; no functionality will be provided allowing the user to modify or add to this data.

2.5.1 Pre-Conditions

Pre-conditions indicate the state of the system and/or the availability of input data required for the admin to begin generating a schedule. The following pre-conditions must be satisfied:

- The admin has determined which courses will be included in the scheduling year, and has determined an assignment of courses to semesters.
- Historical data regarding class sizes is available to the system.
- Current and historical data regarding the number of SENG students in each academic year is available to the system.
- All professors being scheduled have input their availability and teaching preferences into the system.

2.5.2 Post-Conditions

Post-conditions indicate the state of the system after it has successfully generated a feasible schedule for one scheduling year. A feasible schedule is a schedule satisfying all scheduling constraints as described in sections 2.5.2.1 and 2.5.2.2. The system must satisfy the following post-conditions:

- A feasible schedule is generated, if one exists.
- If no feasible schedule exists, this is communicated to the user.
- The generated schedule is presented to the user.
- The user can modify the following parameters of the generated schedule:
    - Time a course is offered.
    - Class size.
    - The term when the course is offered.
    - The professor teaching the course.

2.5.2.1 Hard Scheduling Constraints

The generated schedule must satisfy the following constraints:

- Professors are only scheduled to teach in semesters during which they are not on leave.
- Professors are only assigned to teach during their available teaching hours.
- Professors are only scheduled to teach courses for which they are qualified.
- Professors are assigned to teach the number of courses defined by their teaching load.
    - 3 courses a year for research professor
    - 6 courses a year for teaching professor
    - Changes based on teaching relief or leave of any kind
    - If this number is an integer, then this constraint must be satisfied exactly. The expected behavior when this number is non-integer is specified in section 2.5.2.2.
- No professor is scheduled to teach in two different, simultaneous time blocks.
- All courses are assigned to time slots.
- All courses are assigned to a professor.
- All courses are assigned a class size.
- All courses are scheduled in time blocks falling between 830 AM and 1000 PM.
- Courses requiring a PENG instructor are assigned to a PENG instructor.
- Courses are each scheduled in one of the following time block configurations:
    - A 50-minute block on each of Tuesday, Wednesday, and Friday
    - A 80-minute block on each of Monday and Thursday
    - A 170-minute block on any day of the teaching week.

2.5.2.2 Soft Scheduling Constraints

The generated schedule will optimize a satisfactory schedule based on the preferences provided by the professors. These constraints are satisfied on a best-effort basis and the system provides no guarantees regarding the degree to which they are satisfied:

- Professors are assigned to courses they prefer using a 0-195 preference scale.
- Professors are assigned to teach during their preferred teaching hours.
- Teaching faculty professors are given a non-teaching term of their choice.
- If a professor's teaching load is a non-integer number of courses, then the following soft constraints apply:
  - If the professor prefers to over-teach, they are assigned to teach the number of courses equal to the number obtained by rounding their course load up to the nearest integer.
  - If the professor prefers to under-teach, they are assigned to teach the number of courses equal to the number obtained by rounding their course load down to the nearest integer.

2.6 User Documentation

The creation of user documentation for the system is *not* within the scope of this project.

2.7 Assumptions and Dependencies

To generate a schedule, the system depends on data regarding historical course capacities and student enrolment. Where genuine data is unavailable, the system will use mock data such as the mock enrolment data that has been provided by the course instructors. It is assumed that in a production environment any mock data would be replaced with genuine data.

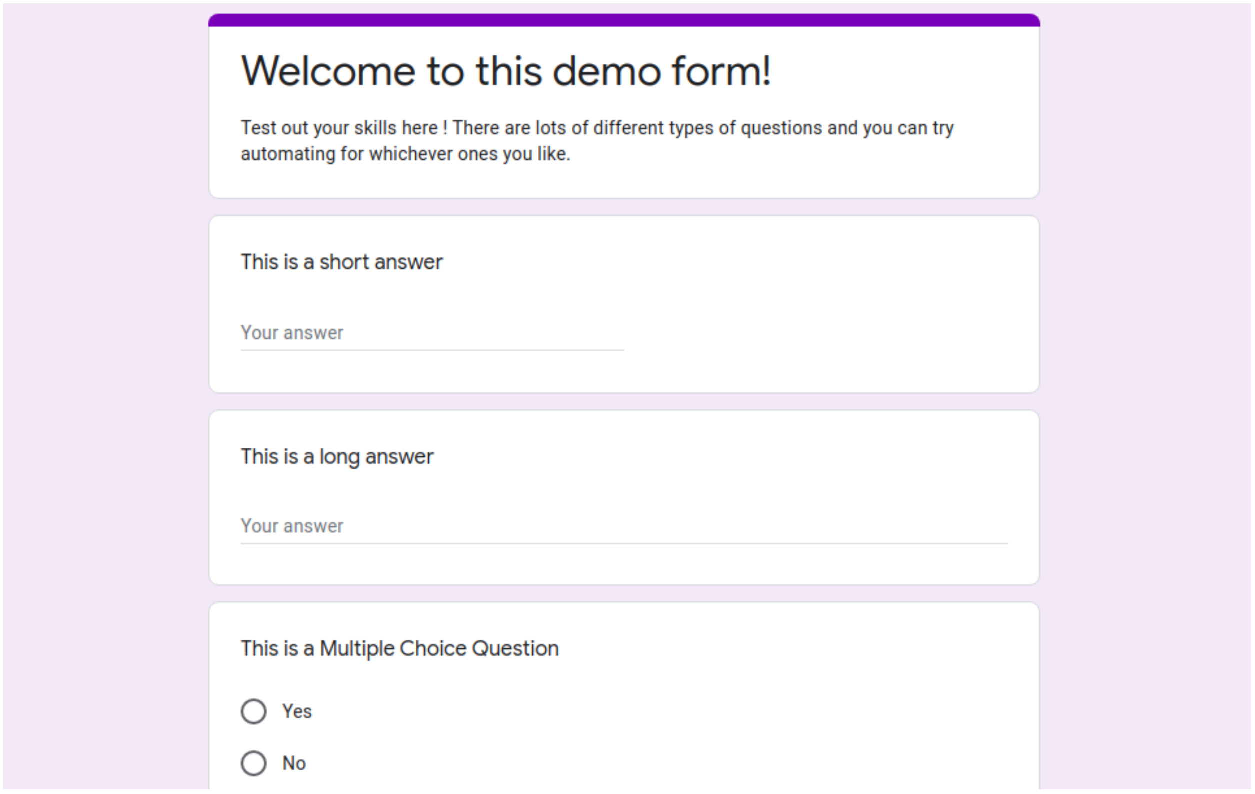
3. External Interface Requirements

3.1 User Interfaces

The user interface will be built with Next.js and Chakra UI. Users will need to login with an authentication page before interacting with other components of the system outlined below.

3.1.1 The Professor Teaching Preferences Form

This form will be used by professors to input their preferences and availability to the system. The form will be implemented as a single page with a list of questions and input elements that accepts various types of input like text inputs, checkbox, multiple choices and dropdown boxes. The bottom of the page will have a "Submit" button. This page would look similar to a Google Form.



Welcome to this demo form!

Test out your skills here ! There are lots of different types of questions and you can try automating for whichever ones you like.

This is a short answer

Your answer

This is a long answer

Your answer

This is a Multiple Choice Question

- Yes
- No

Figure 1: Screenshot of Google Form [1]

3.1.2 Schedule Generator

There will be a courses list page with "generate schedule" button and professors page, and view/edit the generated schedule page. There will be a navigation bar on the left so users can move between pages.

3.2 Hardware Interfaces

The users are expected to use laptops or desktop computers to interact with a web application that the system provides as an interface.

3.3 Software Interfaces

The system provides an interface on a Node.js REST-like web application that supports full-screen mode of browsers. The supported browsers for the web application are the most up-to-date versions of Google Chrome, Mozilla Firefox and Microsoft Edge.

The frontend system will act as an interface for the backend written in Python/ Django, which services the HTTP requests at the frontend and handles the underlying execution of the scheduling algorithms modules. Software at the backend will be maintained during development by following the GitFlow model for version control, and through automated pipelines enabled by GitHub Actions.

3.4 Communications Interfaces

The frontend will communicate with the backend via an HTTP interface that follows a RESTful API design. User authentication will be performed at the backend using Django's JWT library. Additionally, the backend will implement the Django CORS Header library to allow the front end system to access backend system resources.

The backend system will communicate with the scheduling algorithms by importing the algorithm function calls which will be exposed via respective Python libraries. No encryption will be performed on the data hand-off occurring between the backend system and algorithms systems. Figure 3-1 below displays the overall communication between the interfaces.

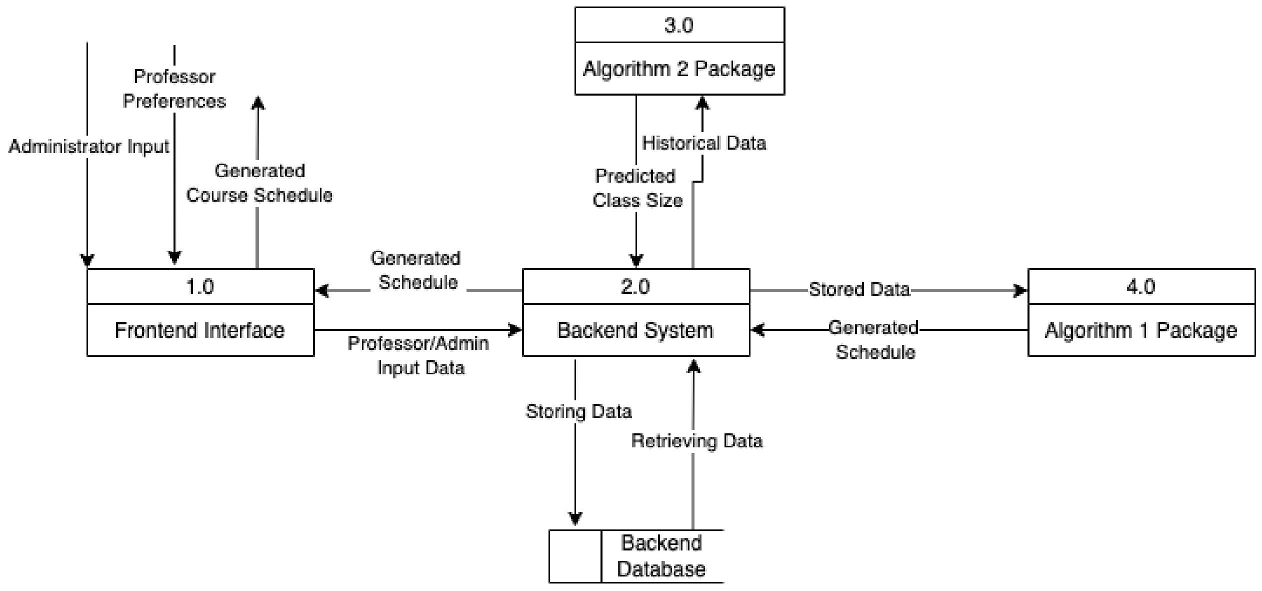


Figure 3-1: Data Flow Diagram of System Interaction.

3.4.1 Frontend-Backend API Interface Endpoints

This interface leverages HTTP request methods that follow standard REST API patterns.

Session/login/logout

`POST   users/session`

(Login) Authenticates the user's credentials against the database.

`DELETE users/session`

(Logout) Destroys the user's session and redirects to the login page.

Preferences

`GET   /preferences/fields`
(Admin) Gets the current list of fields and field options offered in the preferences form. This includes courses that are being offered (profs need to rank each course).

`GET   /preferences/{year-semester}`
(Admin) List all active constraints and preferences for all professors for a given semester or year. Should include profs that haven't submitted their preferences yet but are expected to. Preferences comprise of all soft/hard constraints for a professor that make up the fields of the professor preferences form.

`POST /preferences/{professor-id}`
(Admin) Allows for manual editing of professor hard constraints. We do not need to edit professor soft constraint/preferences, because the algorithm decides on its significance.

`POST  /preferences`
(Prof: Teacher Preferences) Creates a new record in the database for teacher preferences (in the request's JSON body). Record is created as a new `/preferences/{professor-id}.`

Courses

`GET /courses`
(Admin: Department course) Gets all courses in the database. Should include optional filtering parameter to get courses offered for a specific semester.

`GET /courses/{year-semester}`
(Admin: Department courses) To get a list of all courses being offered during a semester. Will include details such as class size and class time (historical or real if schedule has been generated). Also include profs that want to teach the course if that data is available.

`POST /courses/{year-semester}`
(Admin: Department courses) To edit the courses being offered during a semester. Admin should not need to use this endpoint to add all courses for each semester, it should only be needed for small alterations on a semester to semester basis. The backend should add all required courses for a semester by default. Course information will be contained within the body of the request.

`POST /courses/{year-semester}/{course-id}`
(Admin: Department courses) To edit course specific constraints for a given schedule. Constraints such as class size. This endpoint should be used to modify parameters for the algorithm pre-generation.

`DELETE /courses/{course-id}`

(Admin: Department course) Deletes a record in the database for the selected course.

Administrator User Management

`GET /users`
(Admin: Department professors) Get a list of all professors in the database.

`POST /users`
(Admin: Department professor) Creates a new record in the database for a new professor as `/users/{professor-id}`.

`POST /users/{professor-id}`
(Admin: Department professor) Modifies the database record for a specific professor.

`DELETE /users/{professor-id}`
(Admin: Department professor) Deletes a record in the database for the selected professor.

Constraints

`GET    /admin-constraints`
(Admin: Manual Soft/Hard Constraints) Gets the current list of available administrator soft and hard constraints that can be applied prior to the generation of a schedule. Constraints comprise of only professor constraints and course constraints.

`POST   /admin-constraints`
(Admin: Manual Soft/Hard Constraints) Updates a record in the database for manually-entered constraints for the schedule being generated, from the administrator perspective. Constraints comprise of only professor constraints and course constraints.

Schedules

`GET   /schedules/generate`
(Admin) Waits to retrieve a SUCCESS code that the schedule has been generated, to then GET all webpage components to build the schedule in the webpage.

`GET   /schedules`
(Admin) Get list of courses and course times for all generated schedules for the year.

`POST  /schedules/edit`
(Admin) Allow admin to manually adjust specific attributes of the schedule after it is generated. This would include things such as moving one class time to another, changing the enrollment number or percentage of SENG or CSC students, adding/removing a class, or switching the assigned professor.

`GET   /schedules/files/{file-id}`
(Admin) Downloads the generated schedule (visible in the webpage) into a desired filetype. The response includes a data stream that contains the file contents. {file-id} should be retrieved from the schedule instance being viewed by the admin in the current webpage. Documents will be named with a consistent naming convention.

`GET   /schedules/{schedule-id}`
(Admin) Get different versions of the schedule.

3.4.2 Backend-Algorithm1 Module Interface

This interface will be implemented in a Python module also containing Algorithm 2, and imported into the backend scope to execute algorithms. The backend will individually call Algorithm 1. All data required for Algorithm 1 will be passed in as a list of objects in the modules parameters.

3.4.3 Backend-Algorithm2 Module Interface

This interface will be implemented as a Python module also containing Algorithm 1. The Python module will be imported to the backend scope as a single package. The backend system will individually call Algorithm 2. All data required for Algorithm 2 will be passed in as a list of objects in the modules parameters.

3.4.3 Algorithm1-Algorithm2 Interaction

The Algorithm 1 and 2 will be implemented as a single Python module used by the backend system. This interaction is outlined in figure 3-2.

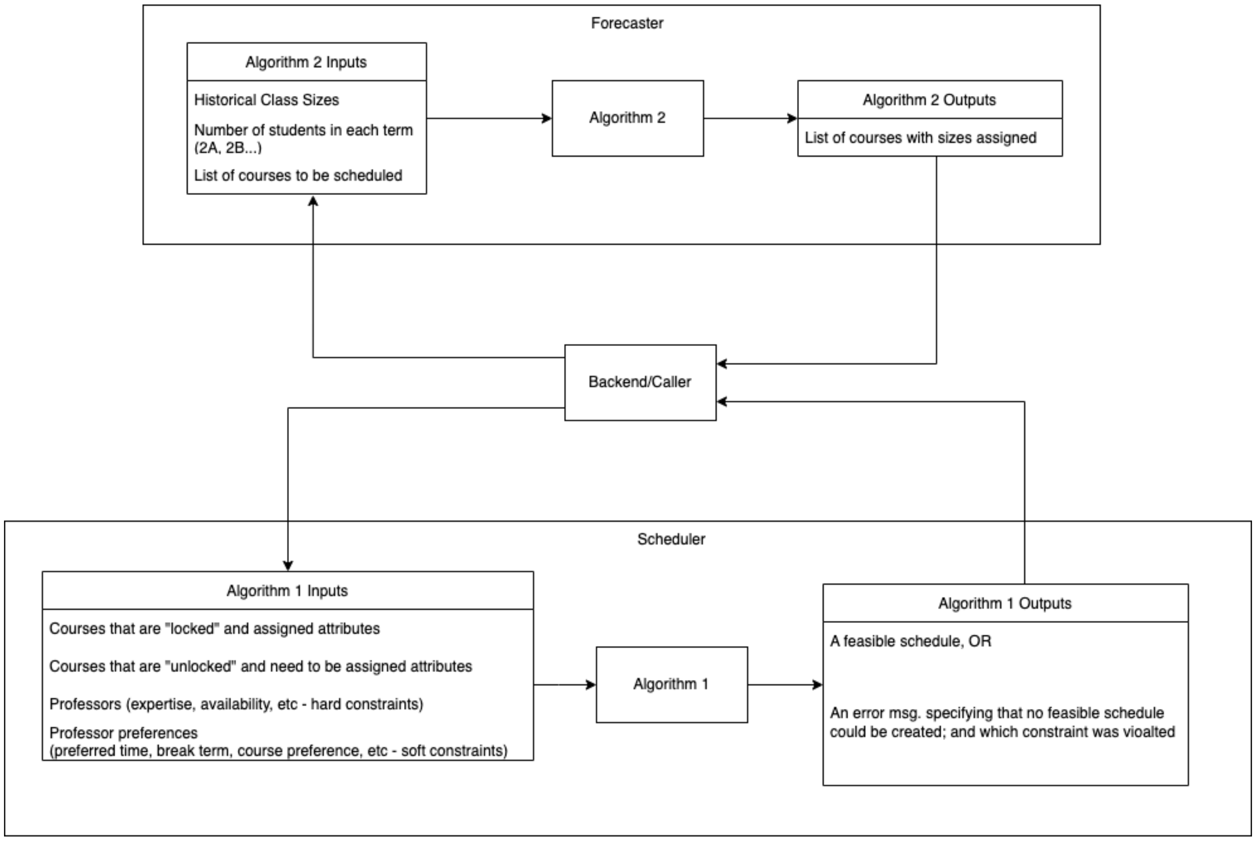

Forecaster
Algorithm 2 Inputs
Historical Class Sizes
Number of students in each term (2A, 2B...)
List of courses to be scheduled
Algorithm 2
Algorithm 2 Outputs
List of courses with sizes assigned
Backend/Caller
Scheduler
Algorithm 1 Inputs
Courses that are "locked" and assigned attributes
Courses that are "unlocked" and need to be assigned attributes
Professors (expertise, availability, etc - hard constraints)
Professor preferences (preferred time, break term, course preference, etc - soft constraints)
Algorithm 1
Algorithm 1 Outputs
A feasible schedule, OR
An error msg. specifying that no feasible schedule could be created; and which constraint was vioalted

Figure 3-2: Algorithm 1 & 2 interaction.

3.4.4 Algorithm System Protocols

The Algorithm's interface will be a Python module, which will provide the following functions for executing Algorithm 1 and Algorithm 2, respectively:

`generateSchedule()`

The interface to Algorithm 1. This function takes as input a set of courses and their constraints, an assignment of the specified courses to semesters, the class sizes predicted by Algorithm 2, and a set of professors and their teaching preferences. It returns a feasible schedule if one exists and no inputs are invalid or missing; otherwise, it returns an error message.

`forecastClassSizes()`

The interface to Algorithm 2. This function takes as input a set of courses, an assignment of the specified courses to semesters, data regarding historical course capacities, and current and historical enrolment data. It returns an enrolment cap for every course provided in the input if no inputs are invalid or missing; otherwise, it returns an error message.

4. System Features

4.1 GUI Frontend Interface

4.1.1 Description and Priority

The scheduler system will consist of a web application with frontend in React and Node.js using Next, and backend written in Python using the Django web framework, with a RESTful API interface to communicate between frontend and backend environments. The system will run on a hosted server to be accessible from any desktop web browser, regardless of the operating system. Therefore, all Unix flavours, Windows, and OSX environments will be supported.

4.1.2 Functional Requirements

REQ-1: Display a login page that is accessible to two user types: administrator and professor.
REQ-2: (Prof) Display a 'Teaching Preferences' webform page with fillable fields.
REQ-3: (Prof) Perform HTTP POST request to the server containing the filled out form data.
REQ-4: (Prof) Accept HTTP response from the server, and show successful form submission.
REQ-5: (Prof) After form submission on a specific professor account, show a message: 'Teaching preferences have already been recorded for this academic year', and deny further form submissions.
REQ-6: (Admin) Display two accessible pages: A page to upload hard constraints for the schedule from the department (via CSV file), and a page to manually set hard and soft constraints and generate a new academic schedule
REQ-7: (Admin) Display an 'Upload CSV File' button to accept the department's hard constraints file.
REQ-8: (Admin) Perform HTTP POST request to the server containing the department hard constraints CSV file.
REQ-9: (Admin) Accept HTTP response from the server, and show a 'success' message for the CSV file upload.
REQ-10: (Admin) Perform HTTP POST request to the server containing manually-inputted admin hard/soft constraints, which influence the generated schedule.
REQ-11: (Admin) Display a 'Generating Schedule' message while the algorithm is executing.
REQ-12: (Admin) Accept HTTP response from the server, and display successfully generated academic schedule which abides by the predetermined hard constraints.
REQ-13: (Admin) Display the generated academic schedule in the browser webpage.

REQ-14: (Admin) Allow individual overrides of schedule entries, including teaching prof, classroom, and student capacity.
REQ-15: (Admin) Display a ‘Download schedule’ button to save the generated academic schedule to the local desktop’s ‘Downloads’ folder.
REQ-16: (Admin) Display an option to return to the manual constraints input page and regenerate a new schedule.

ERR-1: Show error message on invalid login (REQ-1).
ERR-2: Show error message on invalid field entries upon submitting (REQ-3).
ERR-3: Show error message if form data could not be sent to the server (REQ-3).
ERR-4: Show error message if submission receipt could not be received from the server (REQ-4).
ERR-5: Show error message if the file type is not .csv, or if the file upload failed (REQ-7).
ERR-6: Show error message if the CSV file could not be sent to the server (REQ-8).
ERR-7: Show error message if submission receipt could not be received from the server, or if the file could not be accepted (REQ-9).
ERR-8: Show error message if the CSV file could not be sent to the server (REQ-8).
ERR-9: Show error message if file upload confirmation could not be received from the server (REQ-9).
ERR-10: Show error message if admin manual constraints are invalid, or if data could not be sent to the server (REQ-10).
ERR-11: Show error message if scheduling algorithm fails at any time or if an error occurs during execution (REQ-11).
ERR-12: Show error message if the schedule could not be received from the server (REQ-12).
ERR-13: Show error message if the schedule could not be displayed to the webpage (REQ-13).
ERR-14: Show error message if the schedule fails to download, or an error occurred when converting the schedule to a filetype (REQ-15).

4.2 REST-like API Server Interface

4.2.1 Description and Priority

A Python/Django implementation of the REST (Representative State Transfer) protocol as an API for the academic schedule-generating algorithms, whose endpoints will be called by the frontend system running React and Node.js. The web framework will accept various types of HTTP Get and POST requests, to send and receive data to the backend server, which communicates with the scheduling algorithms.

4.2.2 Functional Requirements

REQ-1: Accept HTTP requests from an external network IP address.
REQ-2: Process HTTP requests made to the exposed API endpoints at the server.
REQ-3: Authenticate users through Django's JWT library, accessed through an endpoint.
REQ-4: (Prof) Accept preferences through an API POST request and store a new entry in the database.
REQ-5: (Admin) Accept hard/soft constraints through an API POST request and store a new entry in the database or update an entry if it already exists.
REQ-6: Return a pending 'Generating Schedule' message while the schedule is being generated.
REQ-7: (Admin) Return the generated schedule as a JSON object.
REQ-8: (Admin) Return the generated schedule as a downloadable file.
ERR-1: Return error response if an improper API request is made.
ERR-2: Return error response if authentication for a user fails (REQ-3).
ERR-3: Return error response if teacher preferences could not be accepted (REQ-4).
ERR-4: Return error response if administrator constraints could not be accepted (REQ-5).
ERR-5: Return error response if the schedule failed to generate at any point (REQ-7).
ERR-6: Return error response if the schedule file could not be sent (REQ-8).

4.3 Forecaster (Algorithm 2)

The forecaster predicts the class sizes for a list of given courses. The forecaster will base the class size on data from historical class size and the current number of SENG students accepted in the following academic year. The forecaster will accept courses to be scheduled including the current term for the course. The forecaster will output the courses to be scheduled for the terms (Fall, Spring, or Summer), their course identifier, and the predicted class size.

4.3.1 Description and Priority

A Python module implementing an algorithm that predicts feasible class sizes for one scheduling year. The algorithm takes as input a set of classes for one scheduling year, historical data regarding class sizes, and current and historical data regarding the number of SENG students in each academic year. It generates a feasible assignment of class sizes to the specified courses, based on the provided data. The forecaster's output is used as in input to Algorithm 1.

4.3.2 Functional Requirements

REQ-1: Return the smallest course capacity satisfying the expected demand for every course offering in the scheduling year.
REQ-2: Return a best effort estimate of a course capacity if the course has never been offered before.
ERR-1: Return error response if the course capacity was not determined. (REQ-1)

ERR-2: Return error response if an estimate was not determined. (REQ-2)

4.4 Scheduler (Algorithm 1)

The scheduler will generate a feasible schedule for the following scheduling year . The schedule will satisfy the constraints given by the administrator and the hard and soft constraints provided by the professors.

4.4.1 Description and Priority

A Python module implementing an algorithm which generates a feasible schedule for one scheduling year. The algorithm takes as input a set of classes for one scheduling year, an assignment of the specified classes to semesters, the class sizes predicted by Algorithm 2, and a set of professors and their teaching preferences. It generates as output a schedule which **satisfies all scheduling constraints as specified in section 2.5.2.**

4.4.2 Functional Requirements

REQ-1: Return a constraint satisfied feasible schedule.
REQ-2: Return a feasible schedule given static courses with predefined timeslots.
REQ-3: Enter additional locked courses and return a feasible schedule.
REQ-4: Verifies a schedule input satisfies the constraints.
ERR-1: Return error response if an improper schedule is produced. (REQ-1)
ERR-2: Return error response if schedule does not contain the defined course time slots. (REQ-2)
ERR-3: Return error response if a single constraint is violated in the locked schedule. (REQ-4)

5. Other Nonfunctional Requirements

5.1 Performance Requirements

It is required for the system to return a feasible schedule for the Software Engineering Courses within 2 hours of clicking the generate schedule button.

5.2 Security Requirements

The frontend system will communicate with the backend system using the HTTP protocol. Additionally, users will be securely authenticated through Django's JWT library, and backend resources will be accessible to the frontend via Cross-Origin Resource Sharing (CORS). No data encryption will be implemented for this system.

6. Other Requirements

TBD

Appendix A: Glossary

API:

> *Application Program Interface.*

GUI:

> *Graphical User Interface.*

Use Case:

> *List of steps designing interactions between an actor and a system to achieve a goal.*

Scheduling year:

> *The period from May of some year to April of the following year. Comprised of a summer, fall, and spring semester, each of which are four months long.*

SENG:

> *Software Engineering.*

JSON:

> *JavaScript Object Notation. A standard file format.*

REST:

> *Representational State Transfer. A software architectural style for design and development of the World Wide Web.*

Python:

> *A high-level programming language.*

Python Module:

> *A python file containing Python statements and definitions.*

HTTP:

> *Hypertext Transfer Protocol. An application layer protocol to transmit hypermedia documents.*

Static course:

A course that is scheduled outside of the SENG department's jurisdiction, and which already has a specified time slot, professor, and capacity. These include MATH, PHYS, ENG, CHEM, ECON, and STAT courses.

Locked course:

A static course, or a course whose time slot and professor have been assigned manually by the administrator.

Unlocked course:

A course whose time slot and professor have not been assigned at the time the scheduling algorithm is applied.

Appendix B: Roadmaps

Algorithm 1 Team

Sprint 1

- Decide on algorithmic approach
- Complete algorithm design including data flow diagrams
- Write test cases

Sprint 2

- Working code for finding viable schedule
- CI/CD implemented

Sprint 3

- Integrate with backend and algorithm 2
- Testing and quality assurance
- Start working on schedule optimization code

Sprint 4

- Finalize schedule optimization code
- Testing and quality assurance
- Fully integrate

Algorithm 2 Team

Sprint 1

- Decide on algorithmic approach
- Complete algorithm design

- Write test cases

Sprint 2
- Forecasting capacities
- CI/CD implemented

Sprint 3
- Integrate with backend and algorithm 1
- Testing and quality assurance

Sprint 4
- Help algorithm 1 with optimization
- Overall testing

Backend Team

MG - Minimal Goal for the current Sprint
At the end of every sprint we also have to complete a sprint report (individual), team assessment (individual), and log book (individual).

Sprint 1

- Setup folder structure for different applications (MG)
- Finalize endpoints with frontend team (MG)
- Users and authentication (/users app)
- Start working on database schema (MG)
- Setup dev and prod environments
- Start working on CI

Sprint 2

- Finish users app (/users) (MG)
- Have CI set up if not completed in sprint 1 (MG)
- Confirm database schema with frontend and algorithms (MG)
- Start working on professor preferences (/preferences app) (MG)
- Start working on courses (/courses app) (MG)
- Create csv functionality
- Investigate and setup deployment method

Sprint 3

- Finish professor preferences (/preferences app) (MG)
- Finish courses (/courses app) (MG)

- Start working on schedule application (MG)
- Integrate with Frontend
- Integrate with Algorithm
- Work on integration tests

Sprint 4

- Fully integrate system (MG)
- Complete any outstanding functionality (MG)
- QA
- Bug fixes
- Work on end-to-end tests

Frontend Team

Sprint 1

- Skeleton navigation through pages
- Finalize endpoints per page
- Design document (Colour palette, fonts, other design conventions)
- Mockups for every page
- Teacher preferences page
- Stretch: Login Page

Sprint 2

- Login Page
- Auth flow
- Year selection (react context)
- Semester selection (context)
- Professor page table
- Start schedule page

Sprint 3

- QA (manual tests, e2e tests)
- Continue on the schedule page
- Semester courses page (Similar to professor page table)
- Schedule editing
- Stretch: individual course and professor pages

Sprint 4

- Styling fixes
- Bug Fixes

- QA
- Schedule page tweaks
- Individual course and professor pages