**Report On**

**Assignment – Optimization**

**By**

**Riddhi Goswami**

**IDNO: 2021A7PS0017U**

****

**BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI**

**DUBAI CAMPUS, DUBAI UAE**

**May - 2024**

# Introduction

**Transportation Algorithm**

Before understanding what my project is bade on, we need to look at a special type of Linear Programming problem known as Transportation Algorithm. Usually, you will have a cost matrix with rows denoting suppliers, columns denoting demand. Each cell will have an associated cost. The aim of this problem is to allocate resources satisfying each supply, demand and ensuring that cost is minimised.

**Vogel’s Approximation Method**

Vogel’s Approximation Method (VAM) is one of the methods used to calculate the initial basic feasible solution to a transportation problem. VAM is an iterative procedure such that in each step, we should find the penalties for each available row and column by taking the least cost and second least cost.

# Methodology

**Plan of work**

Students at BPDC get an academic timetable from college that shows when courses occur, who teaches them, the various sections, electives to choose and so on in a PDF format.

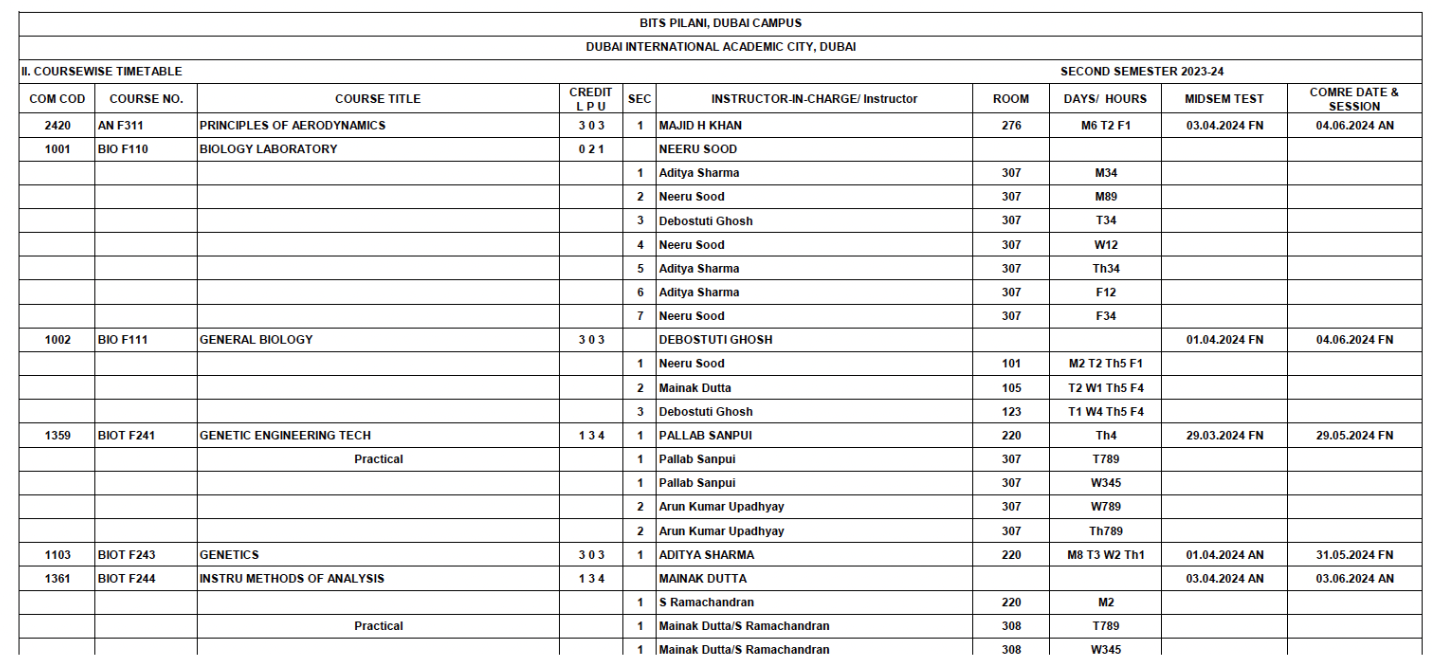


Fig 1: A Section of Timetable Provided

The aim is to create a webapp which can take in details of the student like discipline, year, semester, and this pdf – and output a timetable which can be subject to constraints like section, electives chosen etc. Here’s the code to this: <https://github.com/Secret-Ambush/AI-project/tree/main/Timetable%20generator>

**Handling PDFs**

The first step is to understand how to read the input file, and process it. As given in Fig 1, the timetable is in tabular format. Python has two libraries *fitz* (also known as *PyMuPDF*), *tabula* which can help in processing the PDFs. Finally, analysing it into a dataframe requires some manipulations like removing the header rows etc.

**Creating a Sensible Dataframe**

As it is noticible from the PDF there are many blank spaces – for example the course name is mentioned once, and the next rows aren’t having it. This is understandable to us, but not to a dataframe. Linearly filling the dataframe is the answer. This helps in extending the course no, course title, credits down.

The next step is Practicals which come up under a course name. It’s implied that it belongs to the previously mentioned course, but this has to be ensured in the dataframe as well. The codes for all this are on my Github Repo which is mentioned at the beginning of this section.

Next step is mapping the Days/ Hours section. Here it is given as M4, T45 which mean Monday 4th hour, Tuesday 4th, 5th hours. This mapping has to be created.

Now, there are some rows with which there are no rooms are assigned – we can safely assume that there will be no conflict with the timetable – so these can be removed.

With this, the dataframe is ready.

**Creating a CSP**

Python’s library *constraint* is the best way to solve CSP. Now, it’s a matter of defining the variables, constraints and adding the no\_overlap function. Here, I have to add all the courses (lectures, practicals) and all the unique sections as variables. For constraints, ensure that the student is enrolled in at least one section of each complsory course (practicals of core courses if applicable) and ensure that course timings don’t clash.

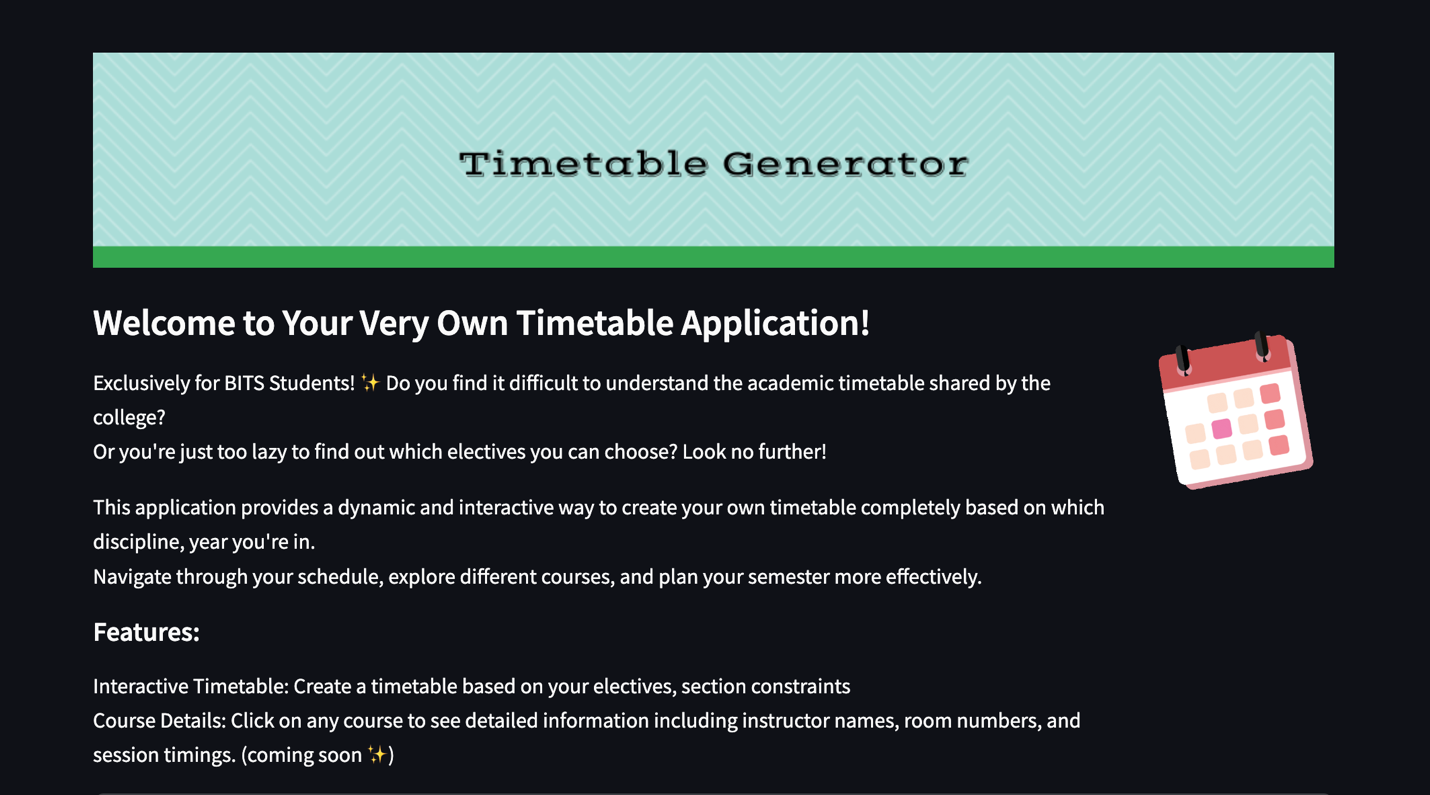
**Building the timetable**

The solution is presented in the form of a dictionary. Now the course title, section details, time slots have to be extracted and converted into a dataframe. I have a timeslots dictonary which has the number of slots/periods per day. Now, I map the solution to this and whenever there is a match and ensure the course name and section are placed. Finally, the dataframe is ready. Using *Pandas Styler*, I can style the dataframe such that if there is an elective or practical – you can highlight it.

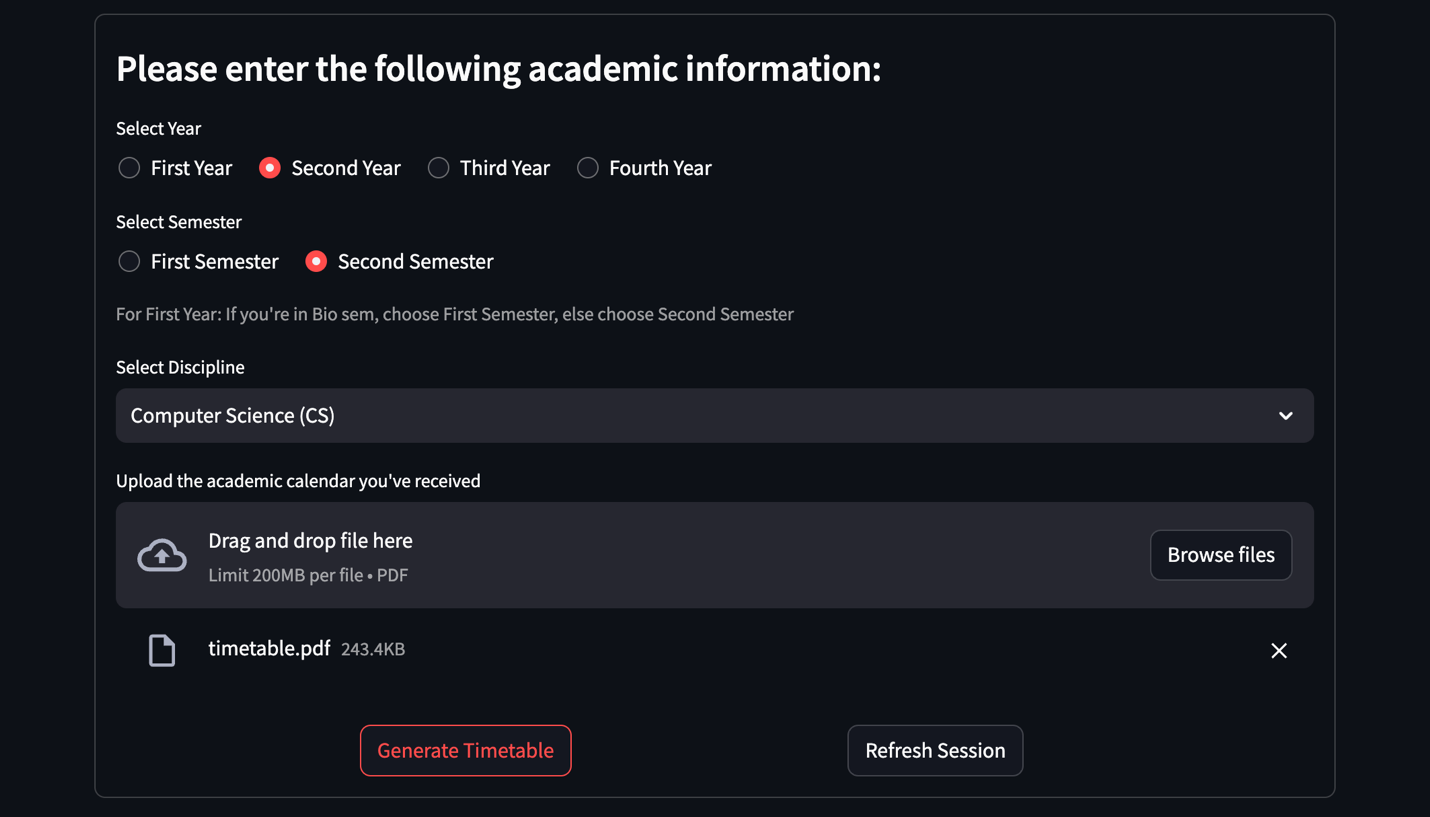
**Building an interface**

Now, comes the tricky part – how to ensure that the user is able to add dynamically constraints based on their year, semester. I built a streamlit interface to collect this data. In streamlit, using buttons and other utils refreshes the session – which will be an issue as we want the constraints to be retained. For this, *session\_state* is used to retain variables like year, semester, section, timetable generated, constraint, electives. Handling this and updating based on the year is quite complicated but I was able to manage it. My code illustrates how this was done. There is an option to download the final timetable as a csv as well.

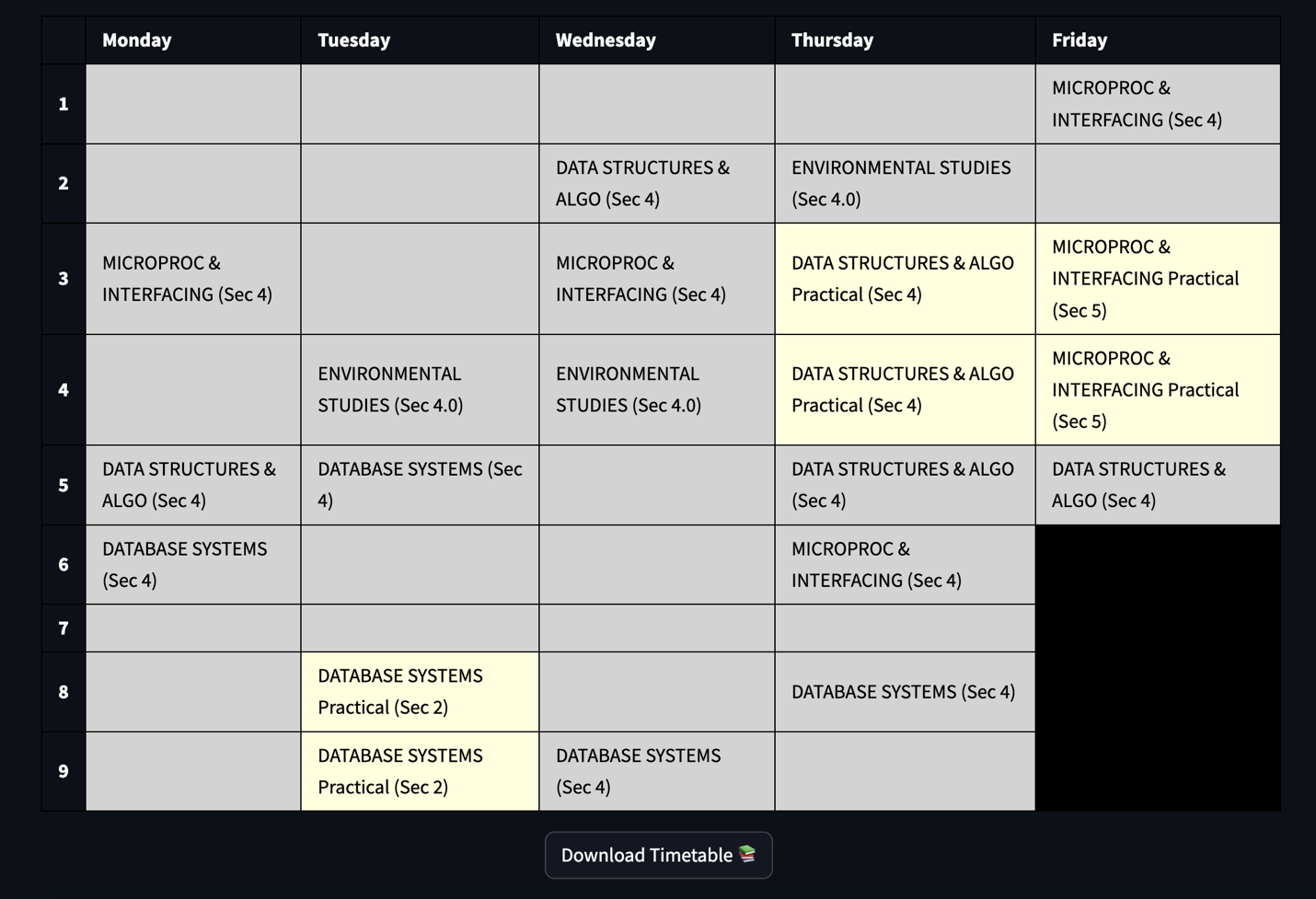
**Images of the interface**



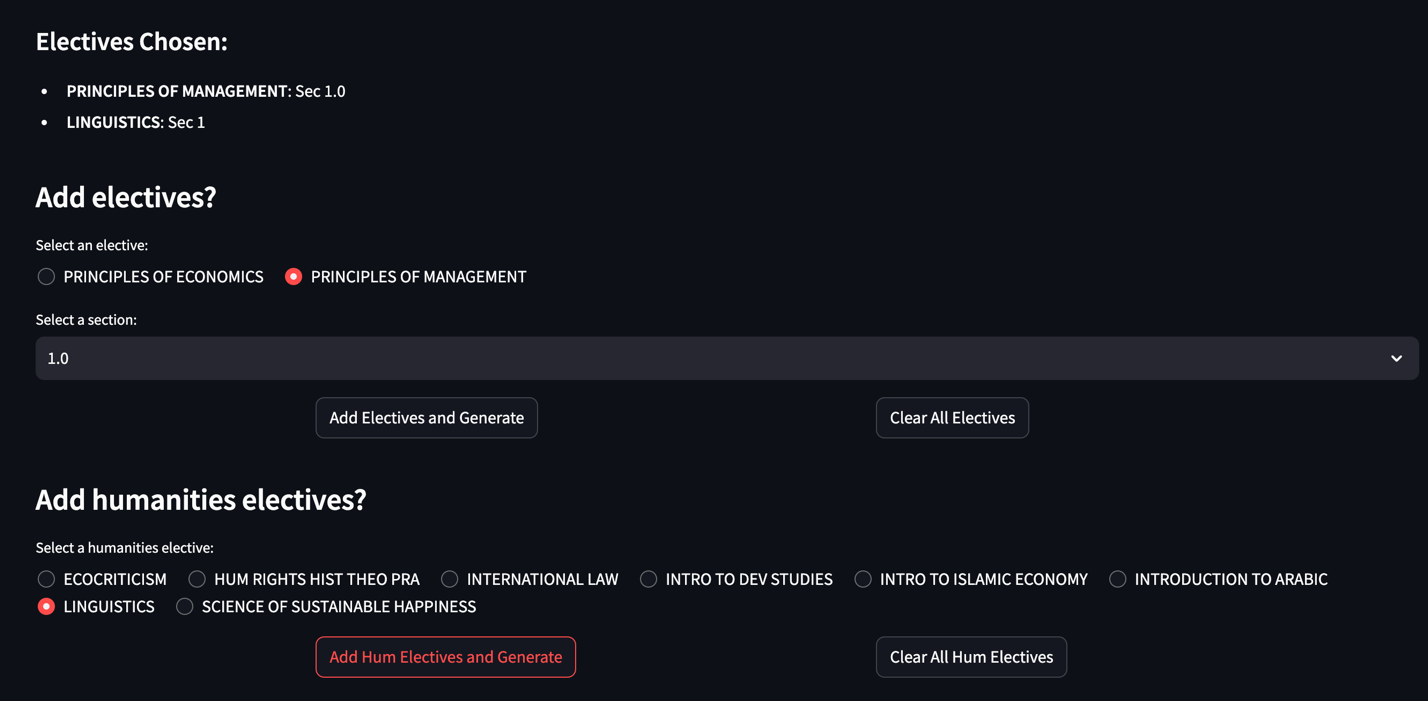
**Choosing Second Year – Second Semester**



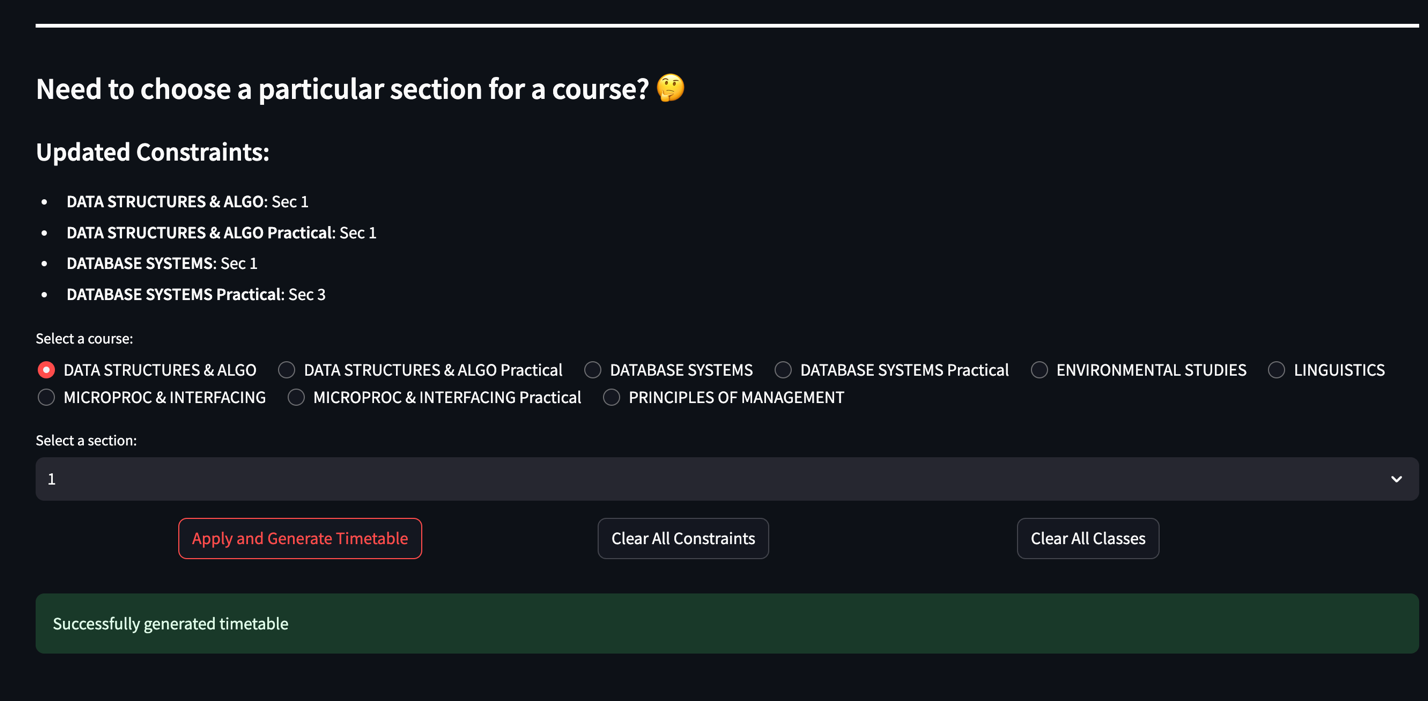
Generating General Timetable without Constraints



Choosing Electives



Applying Section Constraints

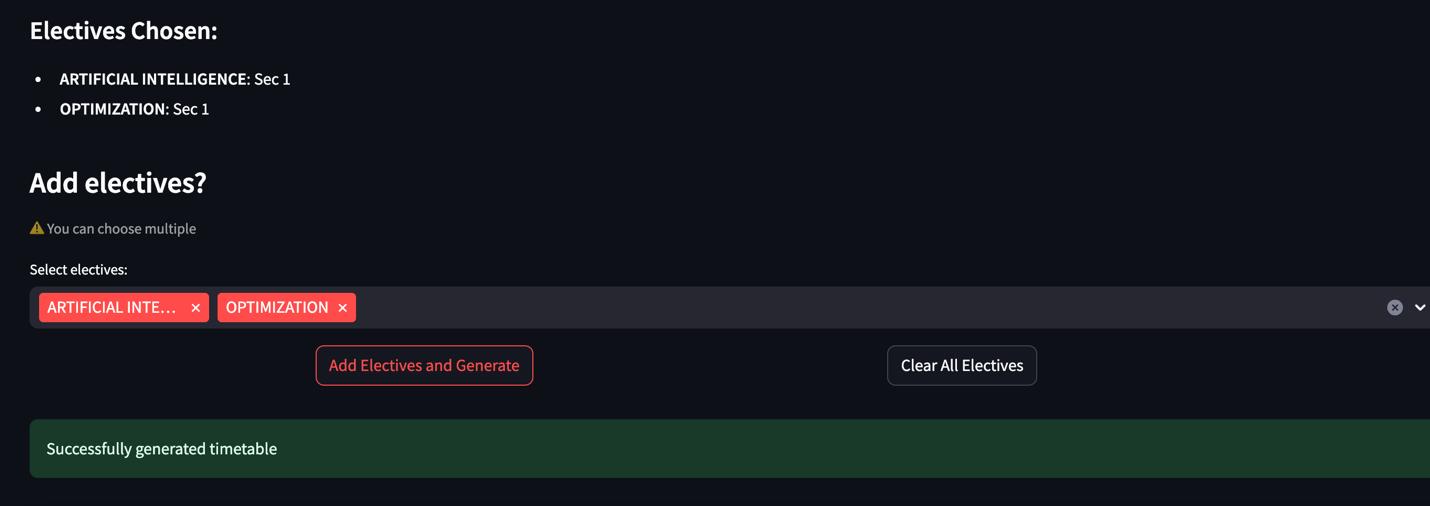


Generating Table Based on Electives and Constraints

A screenshot of a computer

Description automatically generated

**Choosing Third Year – Second Semester**

****

