Timetable Generation System using Genetic Algorithm

Abstract:

The "Timetable Generation System using Genetic Algorithm" is a web-based application designed to automate timetable creation for educational institutions. Utilizing genetic algorithms, the system optimizes course schedules, instructor assignments, and resource allocations while minimizing conflicts and adhering to specified constraints. The project incorporates a SQLite database backend, facilitating data storage and retrieval. Key features include user authentication, course and resource management, genetic algorithm-based timetable generation, and an interactive display of generated schedules. By leveraging SQLite for database management, the system offers scalability, reliability, and efficient data handling. Overall, the project provides a robust solution for improving operational efficiency in educational institutions through automated timetable generation.

Introduction:

The Timetable Generation System using Genetic Algorithm is a web-based application designed to automate the process of generating optimized timetables for educational institutions. This project addresses the complex task of scheduling classes, instructors, rooms, and meeting times, which can be challenging and time-consuming when done manually. By employing genetic algorithms, a type of optimization algorithm inspired by the process of natural selection, the system efficiently generates timetables that meet various constraints and preferences

Technologies used:

* **Django framework**

Django serves as the primary framework for developing the backend of your web application.

Functionality: It handles URL routing, database interactions with SQLite, user authentication, and provides a structured environment for building web applications.

* **DBsqlite3 database**

SQLite acts as the database backend for storing data related to users, courses, instructors, rooms, and meeting times.

Functionality: SQLite provides a lightweight and efficient relational database management system that integrates seamlessly with Django ORM, allowing for easy data manipulation and retrieval.

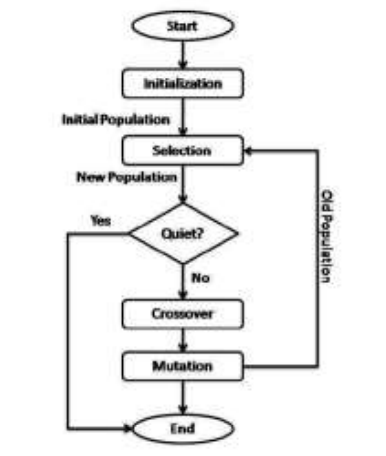
* **Web Technologies:**

1. HTML
2. CSS
3. Javascript
4. JQuery

HTML forms the structural backbone of web pages, defining elements like forms for inputting course details and tables for displaying timetables. CSS enhances the visual appeal by styling these elements, ensuring a consistent and attractive layout across different pages. JavaScript adds interactivity by handling form validation, user events, and dynamic updates to the timetable. jQuery further streamlines development, simplifying tasks like DOM manipulation and AJAX requests, and enhancing cross-browser compatibility. Together, these technologies contribute to a seamless and engaging user experience, making the timetable generation system efficient and easy to use.

**What is Genetic Algorithm?**

A genetic algorithm is a type of metaheuristic algorithm that is inspired by the process of natural selection and genetics. It is a search algorithm used in artificial intelligence and optimization problems to find an optimal solution to a problem. The genetic algorithm mimics the process of natural selection, where only the fittest individuals survive and reproduce, while the weaker ones die off. In a genetic algorithm, the problem is represented as a set of solutions, which are called chromosomes. Each chromosome is a candidate solution to the problem. The chromosomes are then evaluated based on a fitness function, which measures how well the chromosome solves the problem. The genetic algorithm then generates a new population of chromosomes by applying genetic operators of selection, crossover, and mutation. In the selection process, the fittest chromosomes are selected for reproduction. In the crossover process, two selected chromosomes are combined to create a new offspring chromosome. In the mutation process, the offspring chromosome is randomly altered to introduce new genetic information



**Chromosome Representation:** In a genetic algorithm, the genetic material or the set of parameters that define a potential solution to the problem being solved is represented as a simple string, called a chromosome. The fitness of a chromosome is determined by how effectively it addresses the problem at hand. In genetic algorithms, chromosomes are used to represent candidate solutions to a problem. Chromosome representation is an important aspect of the genetic algorithm as it determines how the problem is encoded and how the genetic operators are applied. The most used chromosome representation schemes are binary, integer, and real-valued.

**Initial Population**: The first step in a genetic algorithm is to create an initial population, where each member of the population represents a potential solution to the problem at hand. The fitness function evaluates each unit in the population and assigns a fitness value accordingly the success of the algorithm largely depends on the quality of the initial population. If the initial population is good, then the algorithm has a better chance of finding an optimal solution. On the other hand, if the initial pool of building blocks is insufficient or of poor quality, the algorithm may struggle to find a good solution.

**Selection:** The selection operator in a genetic algorithm chooses chromosomes from the population for reproduction. Chromosomes with higher fitness values are more likely to be selected for reproduction. In each successive generation, a portion of the population is selected to create a new generation. The selection process is based on fitness, where fitter solutions are more likely to be chosen. Selection is a genetic operator in genetic algorithms that is used to choose the fittest chromosomes for reproduction.

**Crossover:** Crossover is a genetic operator that is used to introduce variation in the programming of chromosomes from one generation to the next. This operator involves taking more than one parent solutions and producing a child solution. The crossover operator randomly exchanges subsequences before and after a specific locus between two chromosomes to create two children. This process is similar to natural recombination between two organisms with a single chromosome.

**Mutation:** Mutation is used in a genetic algorithm to maintain genetic diversity from one generation to the next, similar to natural mutation. This operator alters one or more gene values in a chromosome from its initial state. The result of mutation may be entirely different from the previous result, which can lead to an improved outcome. Mutation can occur at each bit position in a string with a small probability, usually less than 1%.

**Fitness Function:** The fitness function evaluates the genetic representation and measures the quality of the solution represented. This function is specific to the problem being solved and is always problem dependent. In genetic programming and genetic algorithms, each design represents a solution. After each round of testing, the goal is to remove the 'n' worst design solutions. Therefore, each solution is assigned a merit value to indicate how close it came to meeting the general requirements, and this value is generated by applying the fitness function to test results obtained from that solution. The performance of genetic algorithm is shown in figure 3.

Genetic Algorithm in this Application:

The views.py file in this Django project does contain code related to the genetic algorithm. Let's break down the relevant sections:

1. Classes Definitions:

* + **Data**: This class fetches data from the database related to rooms, meeting times, instructors, courses, and departments.
  + **Schedule**: Represents a timetable schedule. It initializes with data and generates classes.
  + **Population:** Represents a population of schedules.
  + **GeneticAlgorithm:** Implements methods for evolving the population using genetic algorithms.
  + **Class:** Represents a class in a schedule, including attributes like section ID, department, course, instructor, meeting time, and room.

2. Initialization and Fitness Calculation:

* + **Schedule.initialize():** Initializes a schedule by generating classes based on section details, courses, instructors, meeting times, and rooms.
  + **Schedule.calculate\_fitness():** Calculates the fitness of a schedule based on the number of conflicts.

3. Evolutionary Process:

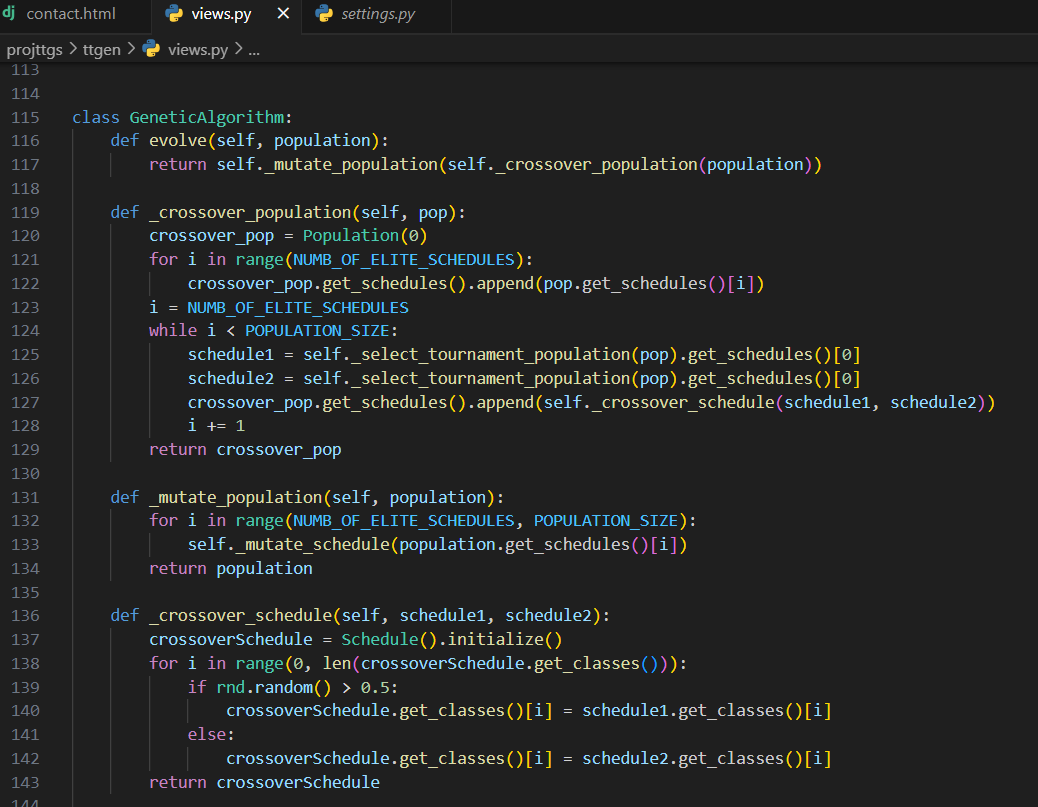
* + **GeneticAlgorithm.evolve(population):** Evolves the population using crossover and mutation.
  + **\_crossover\_population():** Performs crossover operation to create a new population.
  + **\_mutate\_population():** Performs mutation operation on the population.
  + **\_crossover\_schedule():** Performs crossover operation between two schedules.
  + **\_mutate\_schedule():** Performs mutation operation on a schedule.
  + **\_select\_tournament\_population():** Selects schedules for crossover based on tournament selection.

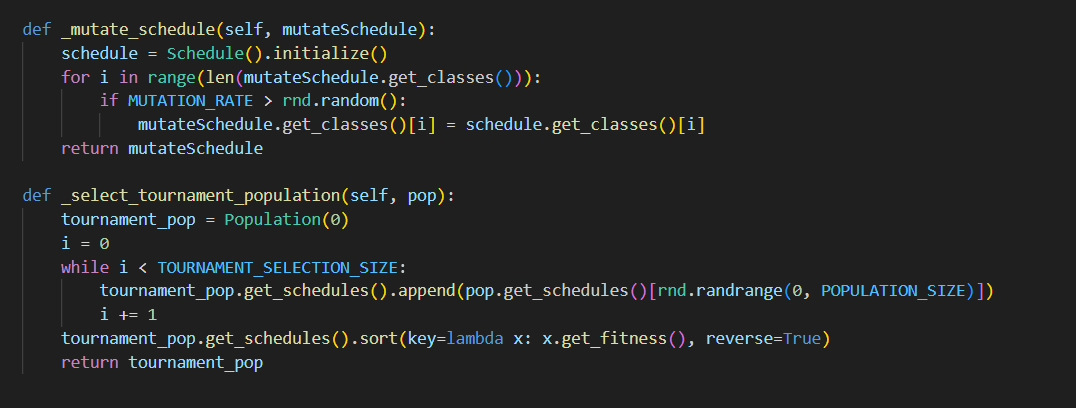
4. Generating Timetable:

* + **timetable(request):** Generates the timetable by evolving the population until a schedule with maximum fitness (no conflicts) is found.

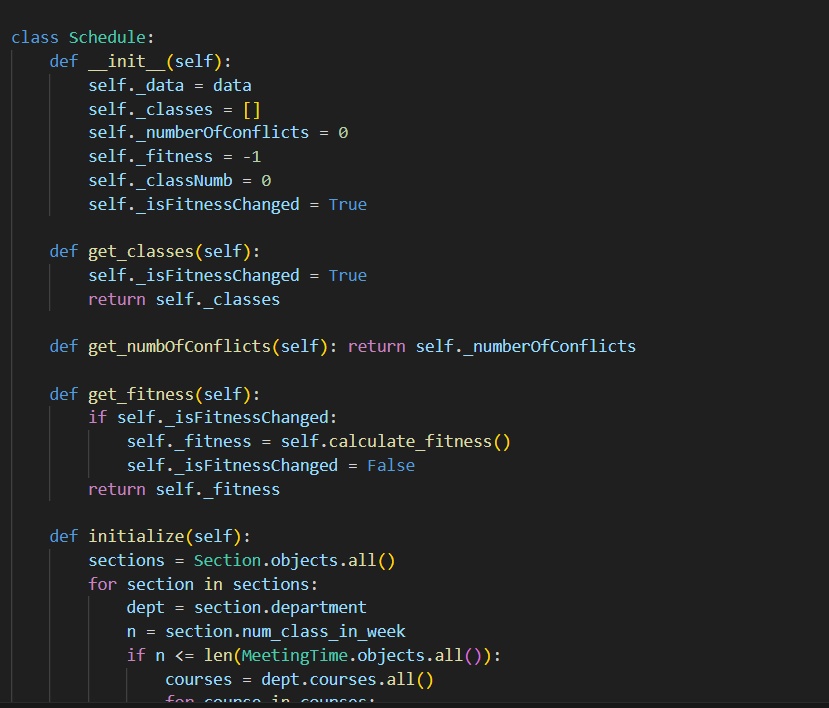
* + It initializes a population of schedules, evolves them through crossover and mutation, and selects the fittest schedules for the next generation. Finally, it presents the generated timetable on the web interface.

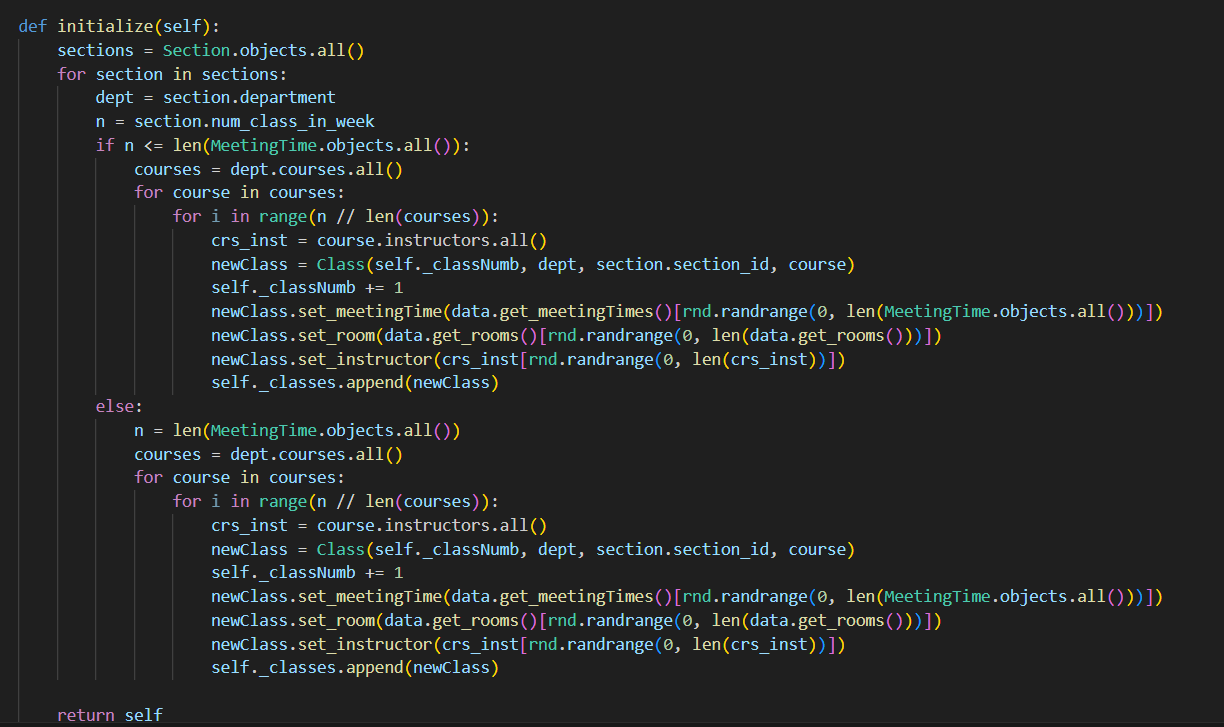
**Class Genetic Algorithm:**

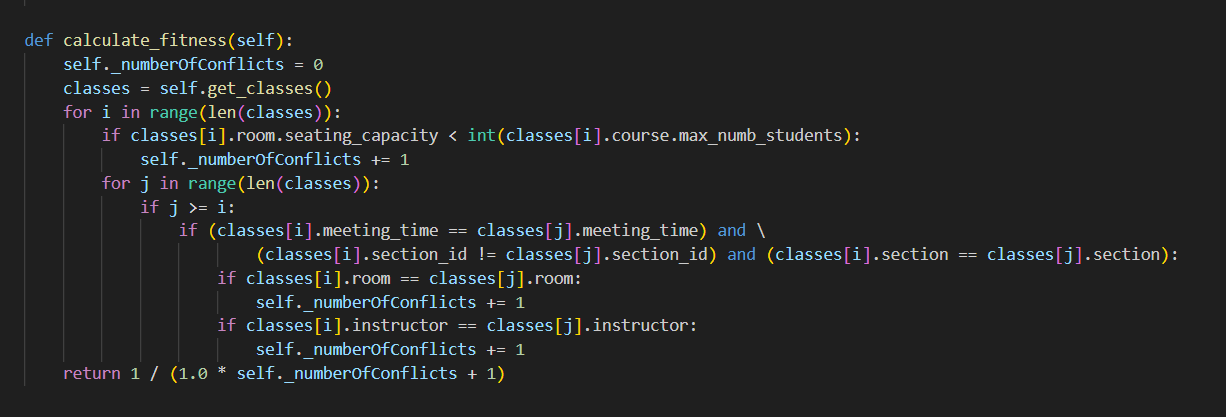




Fitness Calculation:

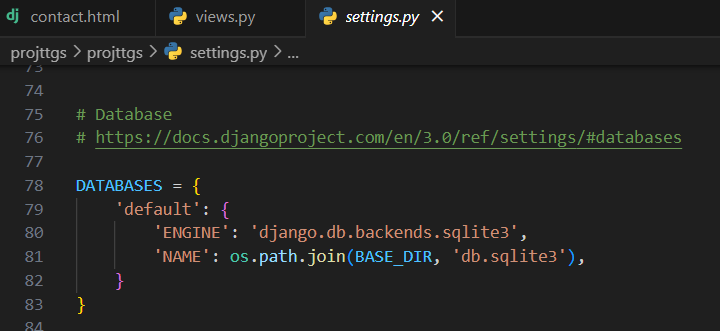




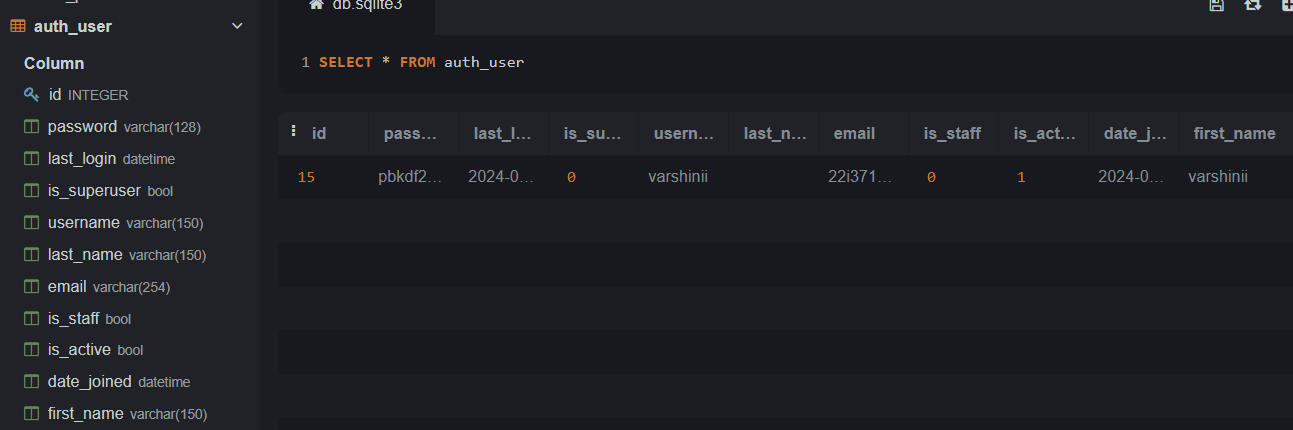


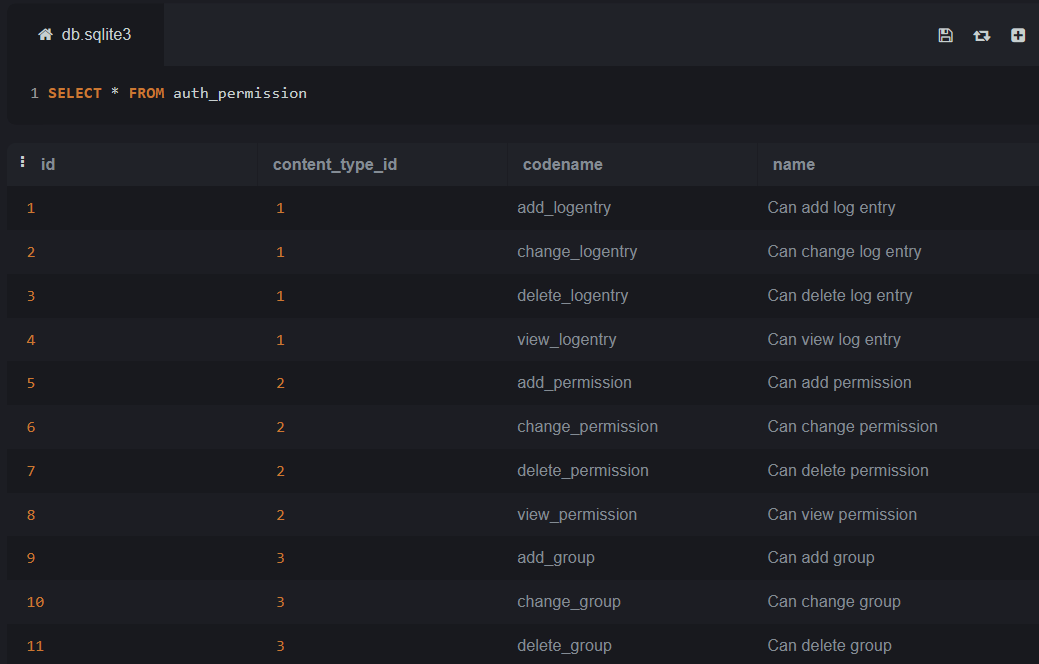
Database and Schemas:

Connection to the default database in project file:

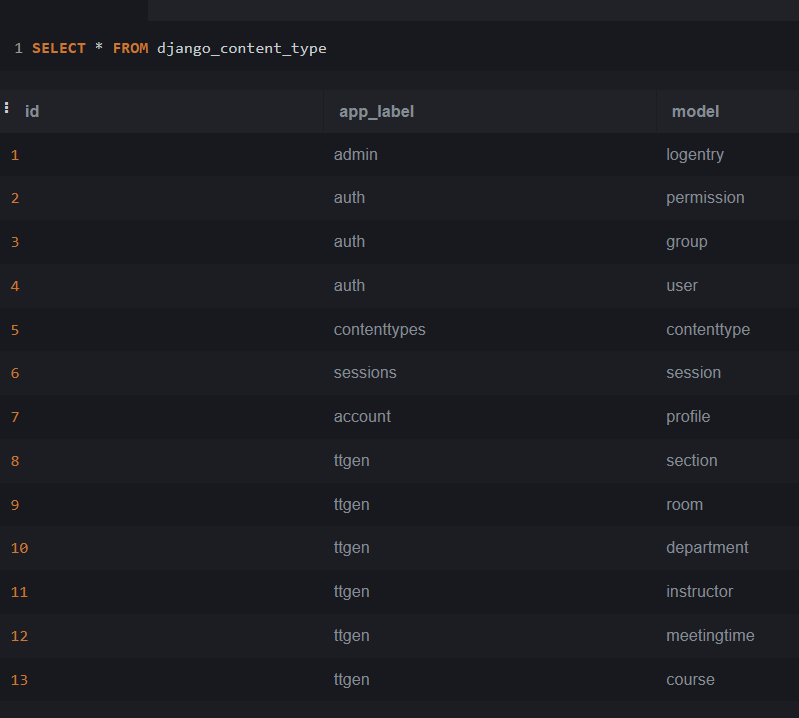


Auth\_user:

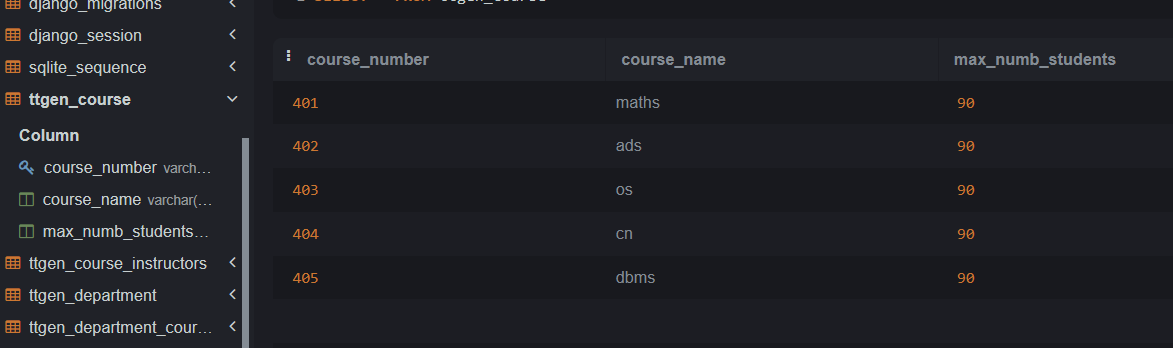


Auth\_permission

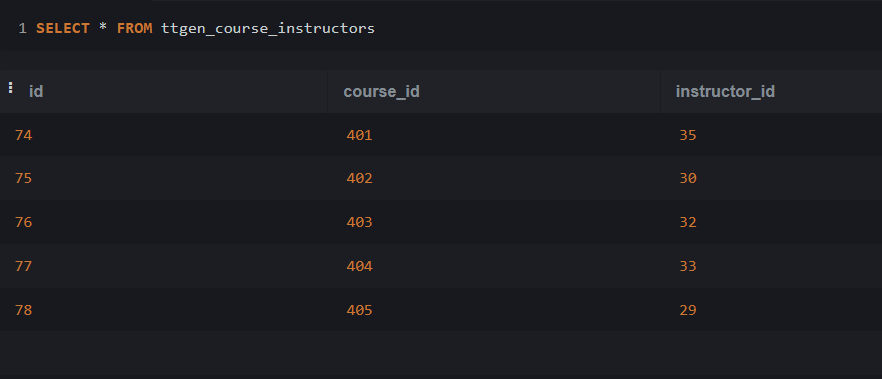
Django\_content\_type:



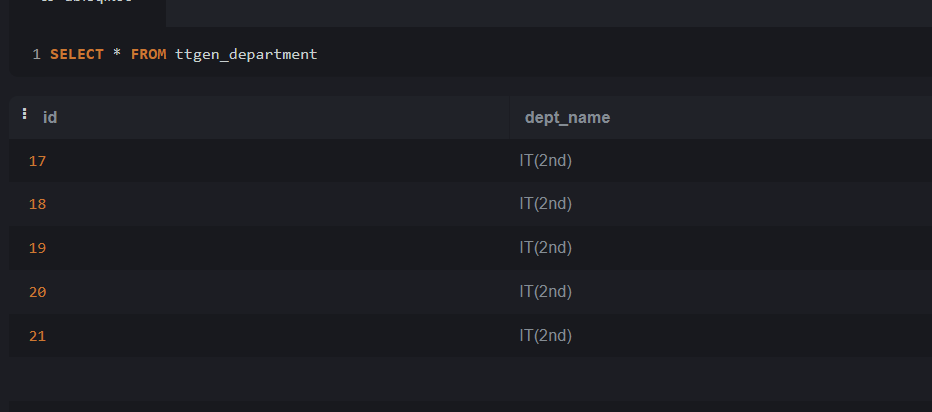
Ttgen\_course:



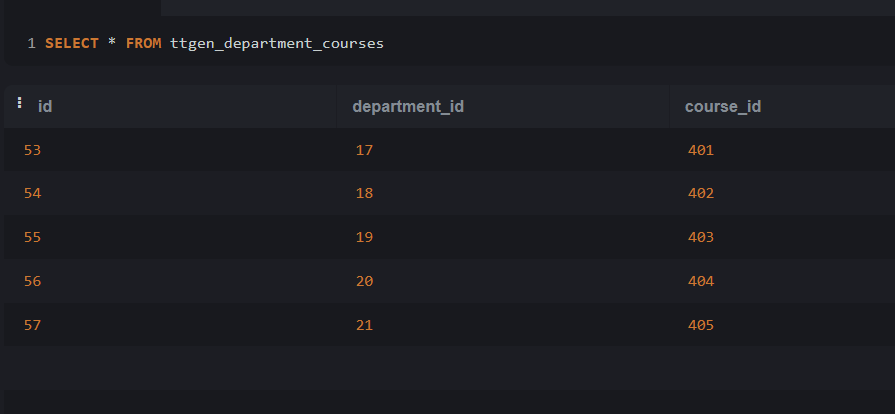
Ttgen\_course\_instructors:



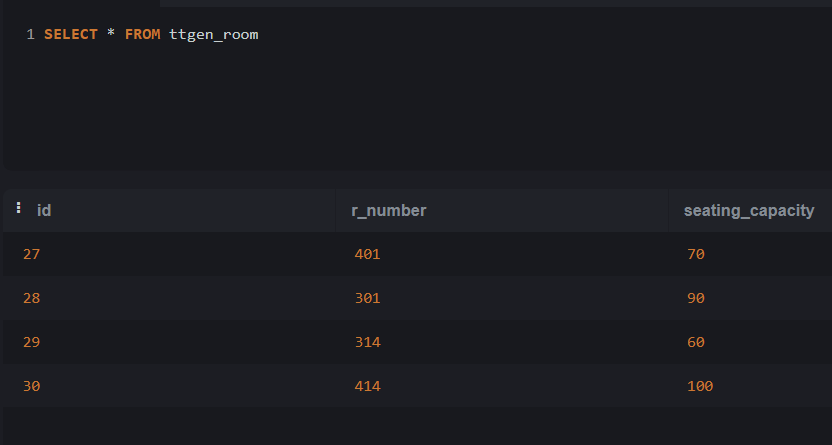
Ttgen\_department:



Ttgen\_department\_courses:



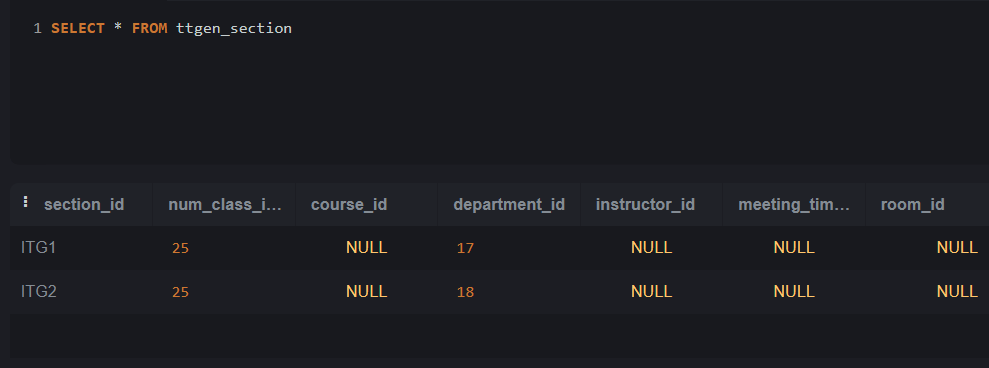
Ttgen\_room:



Ttgen\_meetingtime:

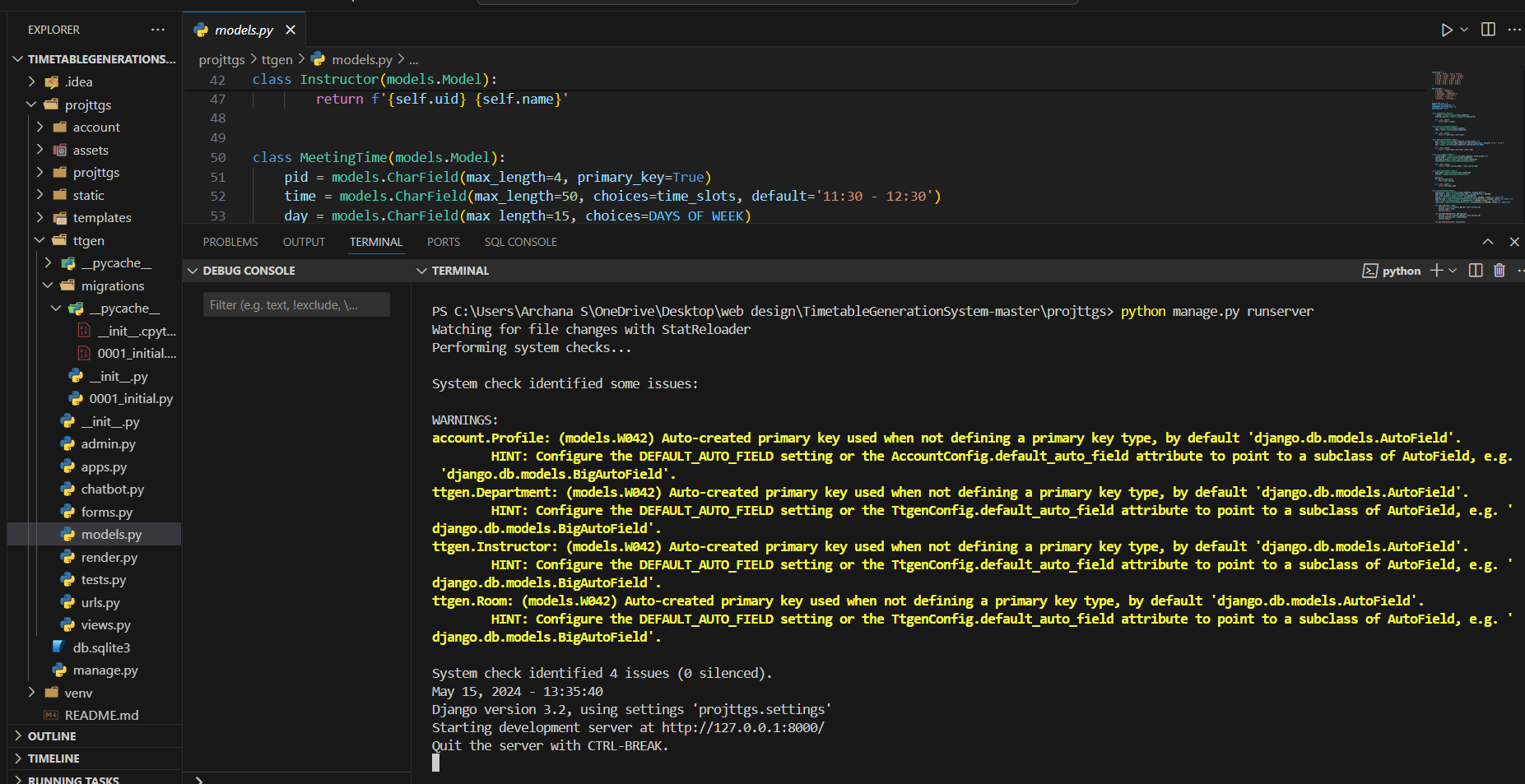


Ttgen\_section:

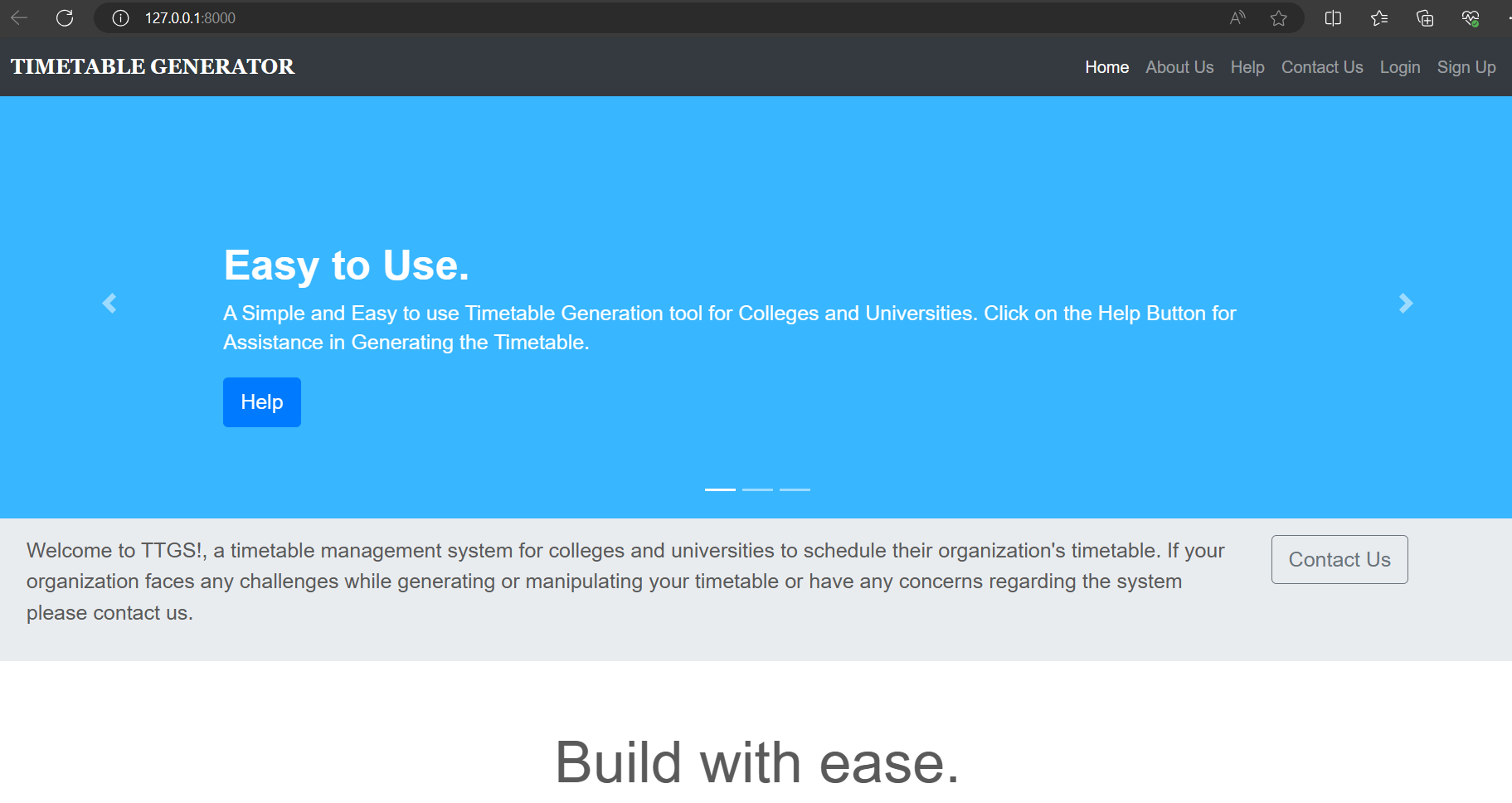


**IMPLEMENTATION**

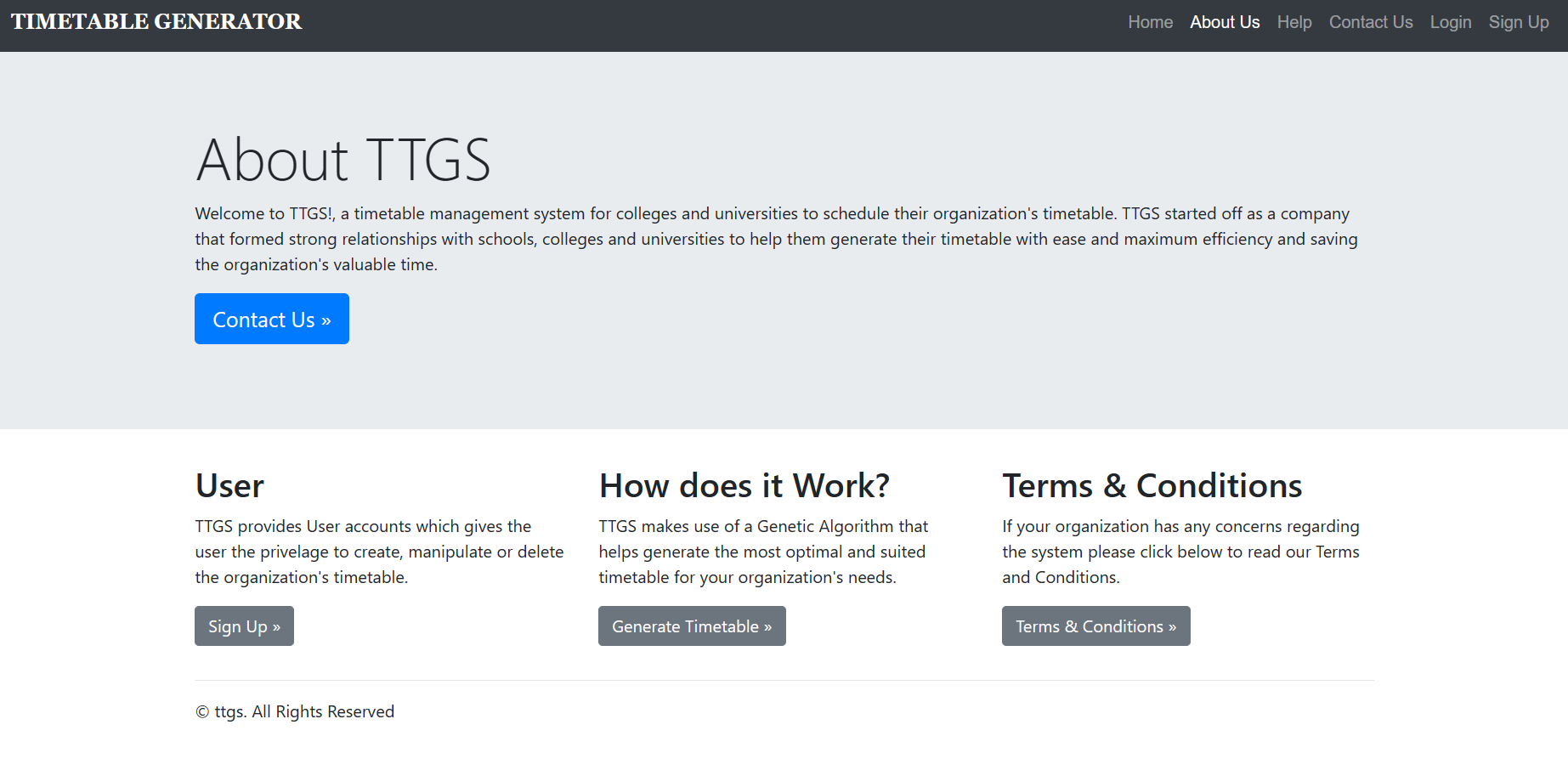
Running the project file using the Command: python manage.py runserver



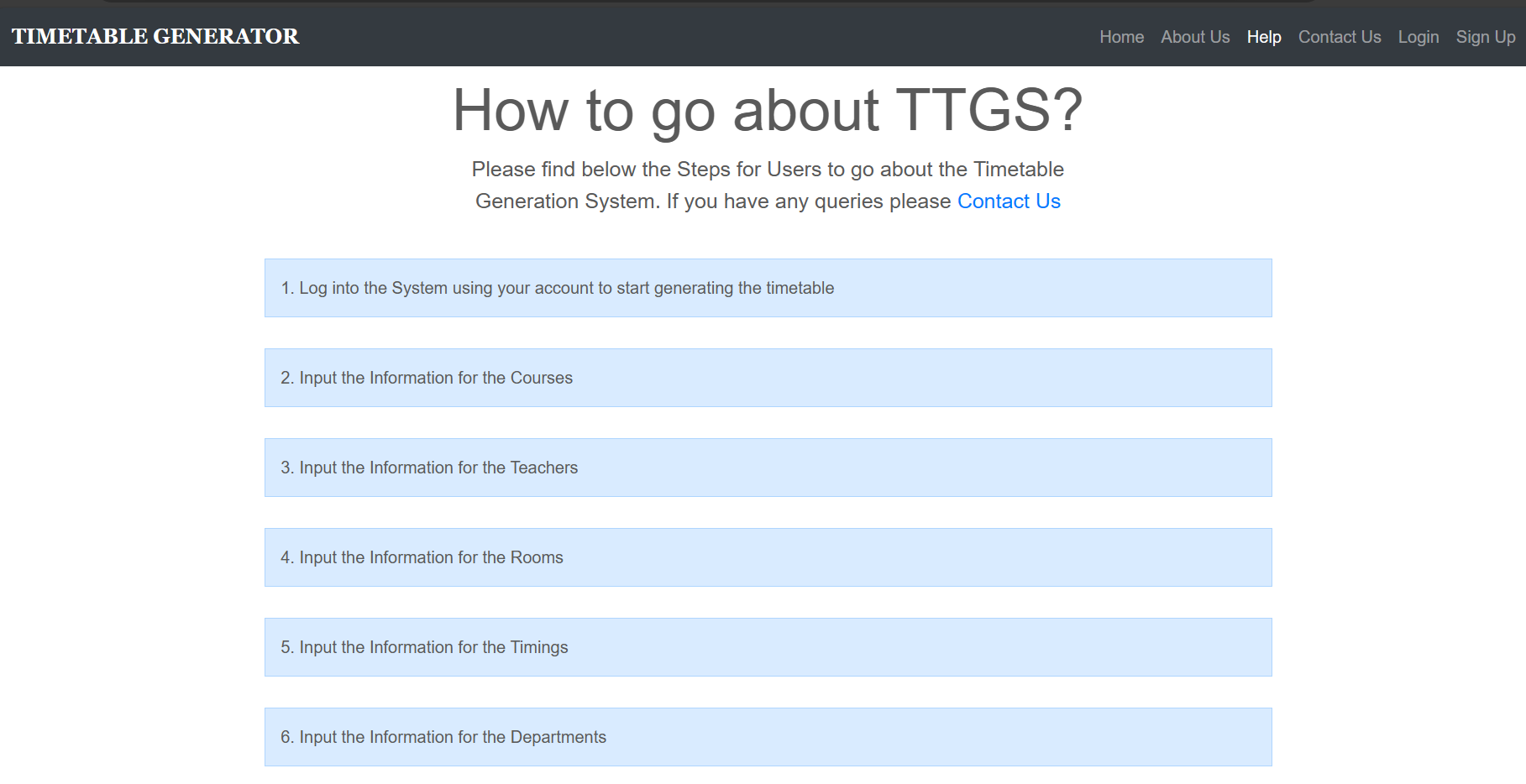
**Home page**:



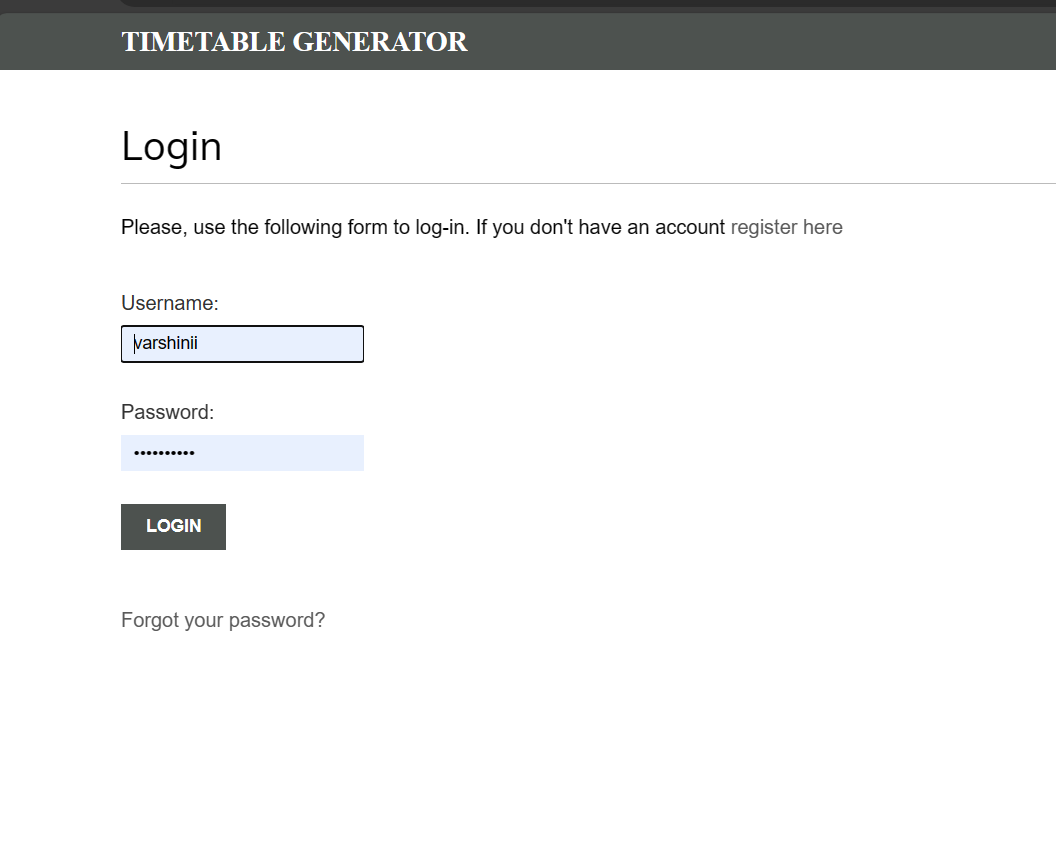
**About Us:**

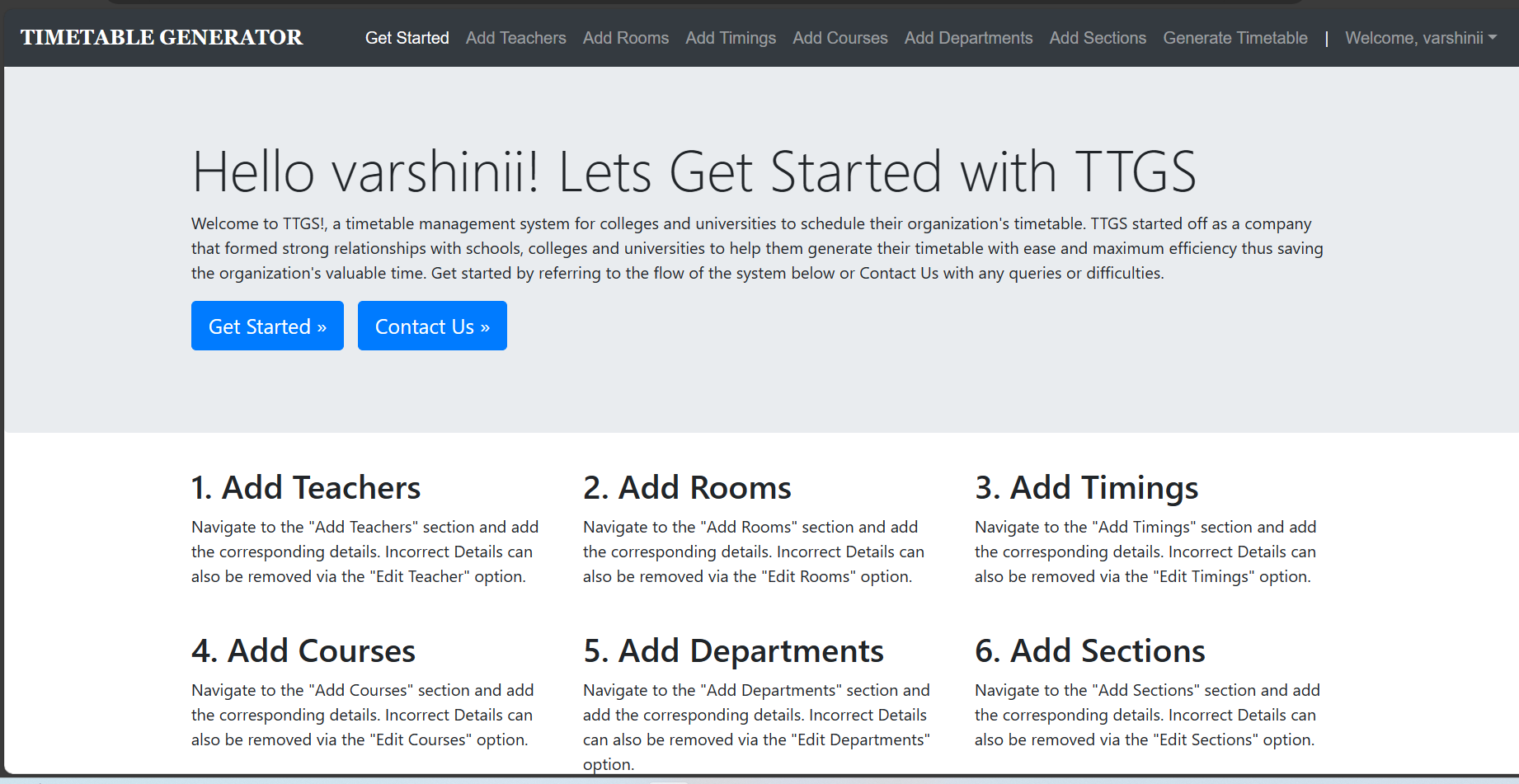


**Help:** This includes the steps how to generate timetable by this website.

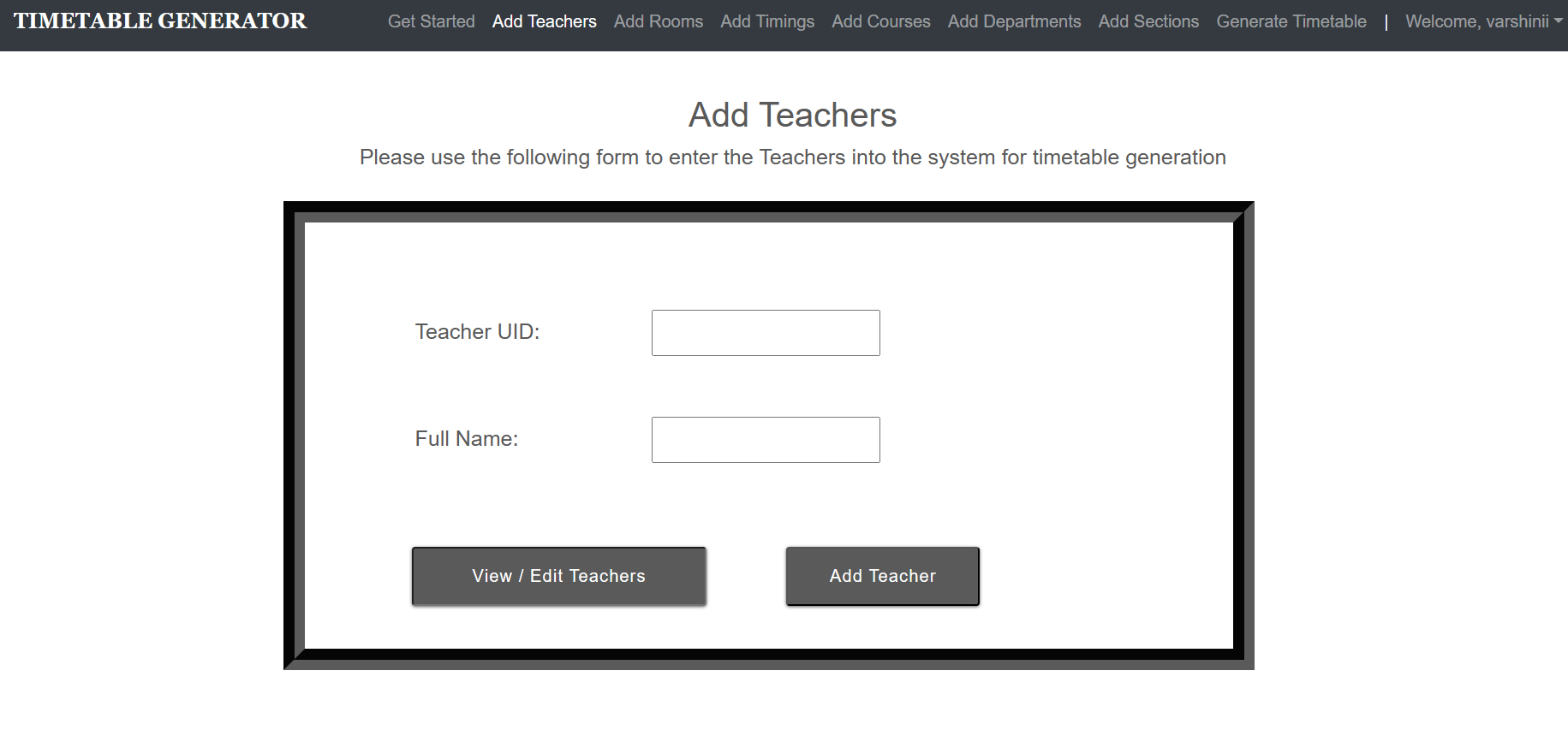


**Login:** Authenticated log in page.

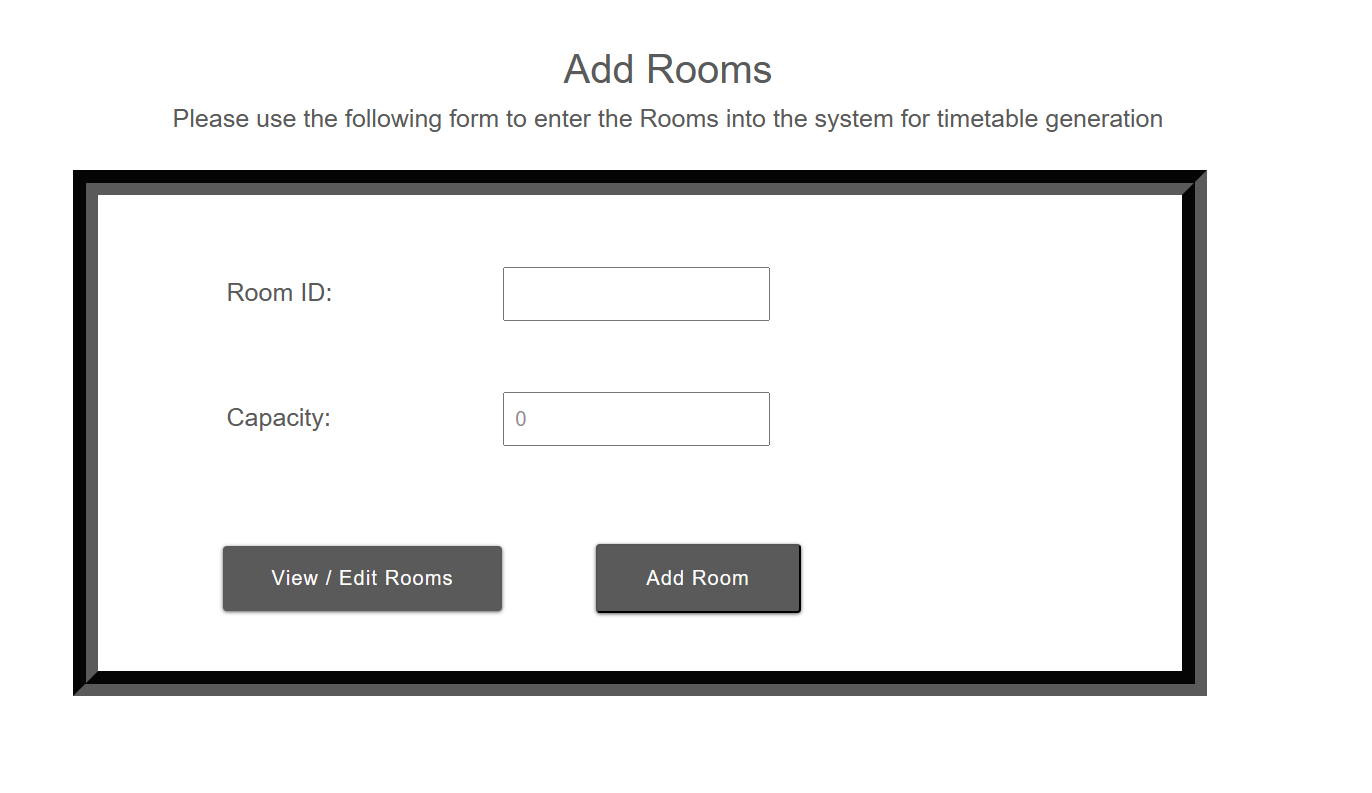
**Get Started:** After login this page will open.



Add Teachers: This is part is used to add the teachers with their ID.



**Add rooms:** We can add the classrooms with its capacity.

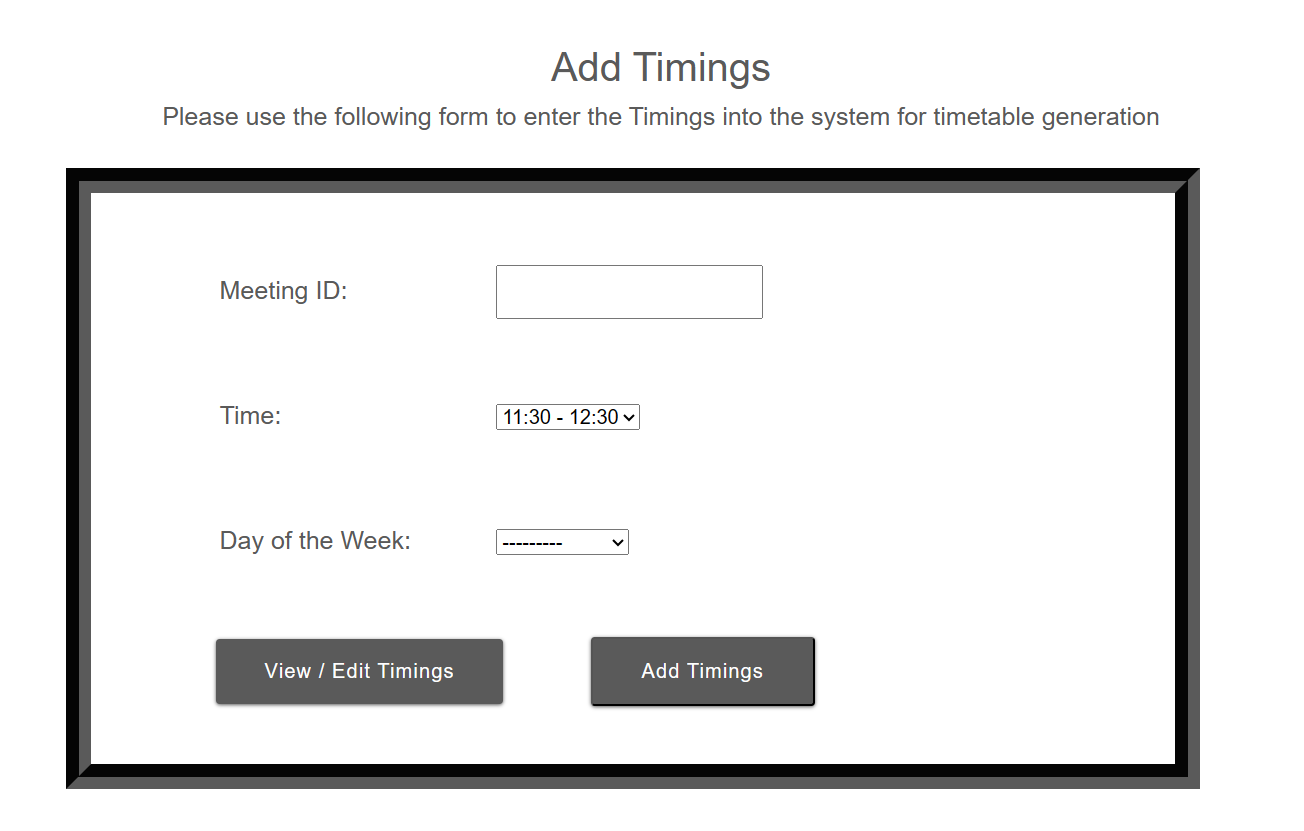


**Add Timings:** In this we have to include the meeting id with timing and day of the week.

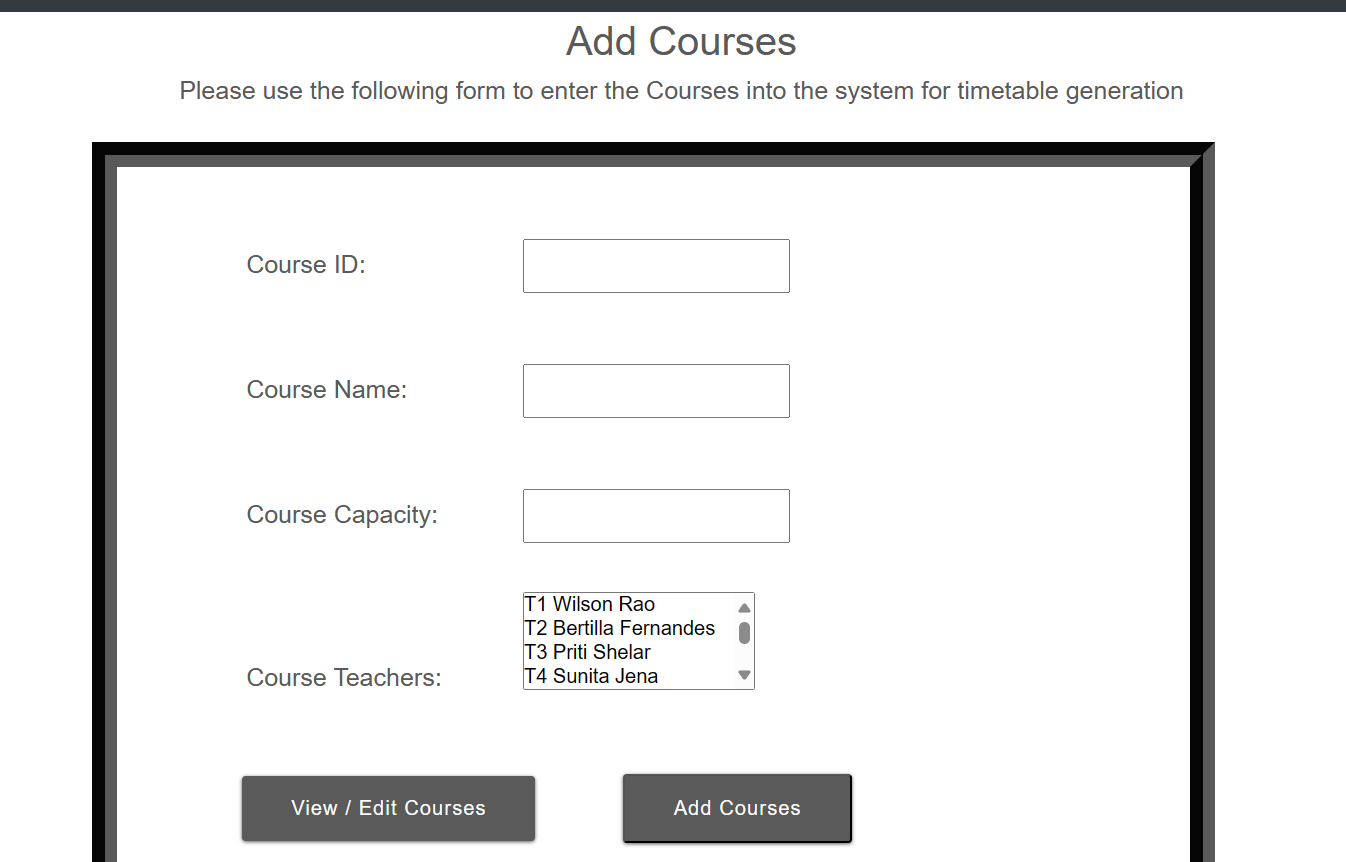
So, for each day each hour we have to give the unique meeting id.

For example: Monday first hour 9.30- 10.30

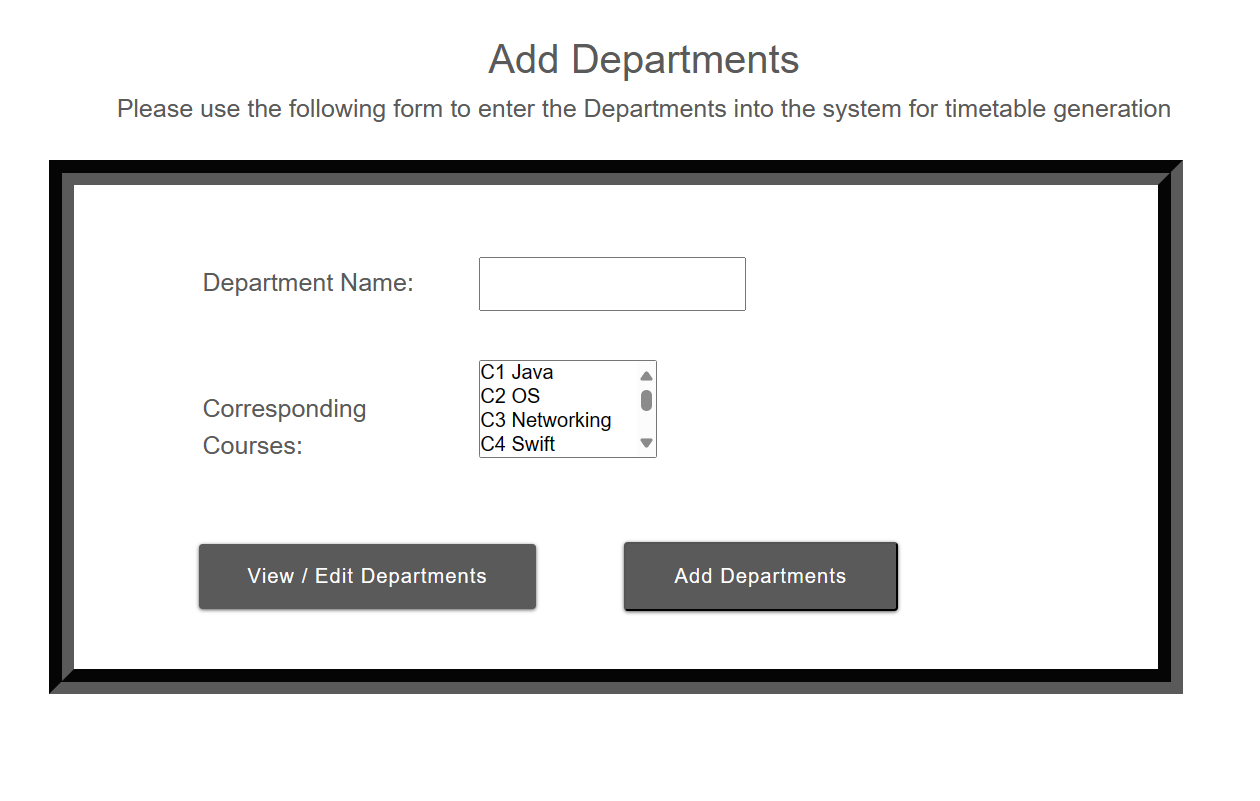
Give like, M1 9.30-10.30 Monday



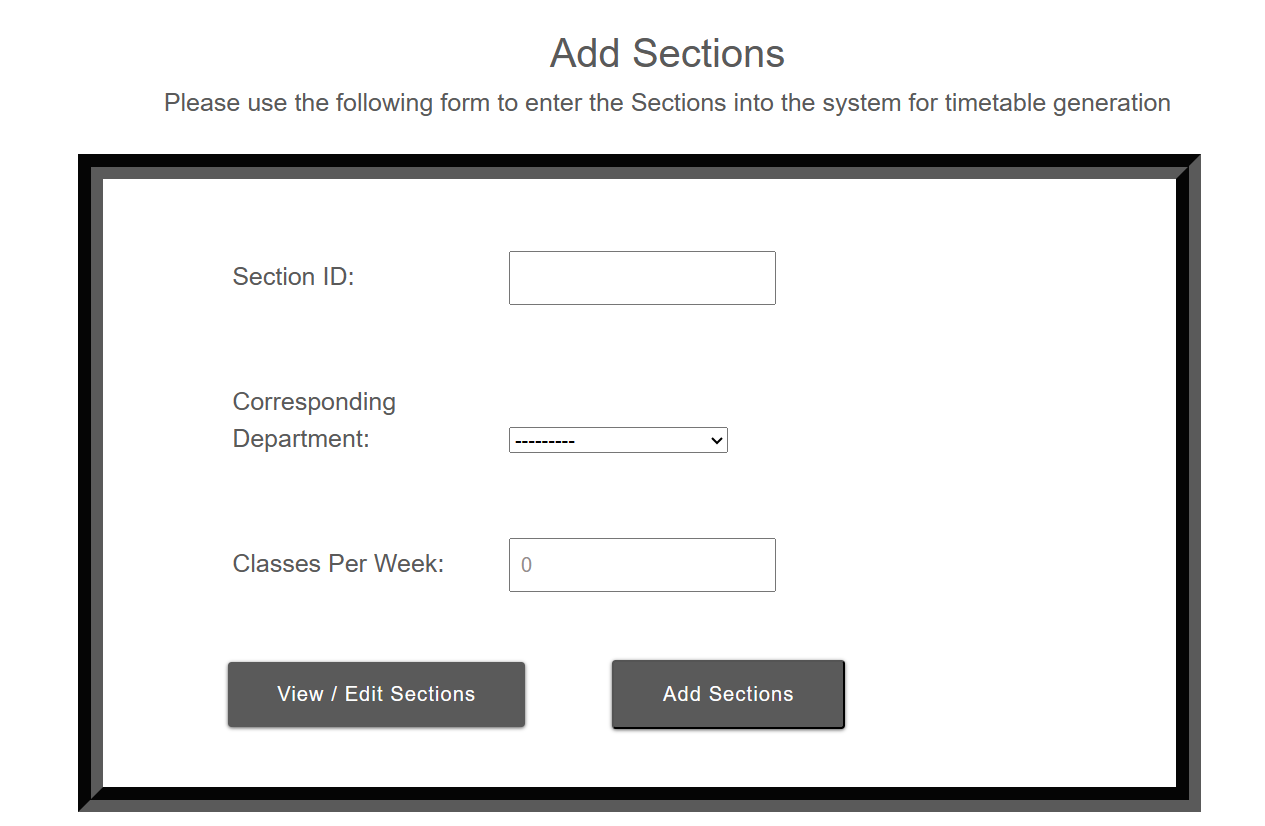
**Add courses:** We can add the courses by giving below informations.



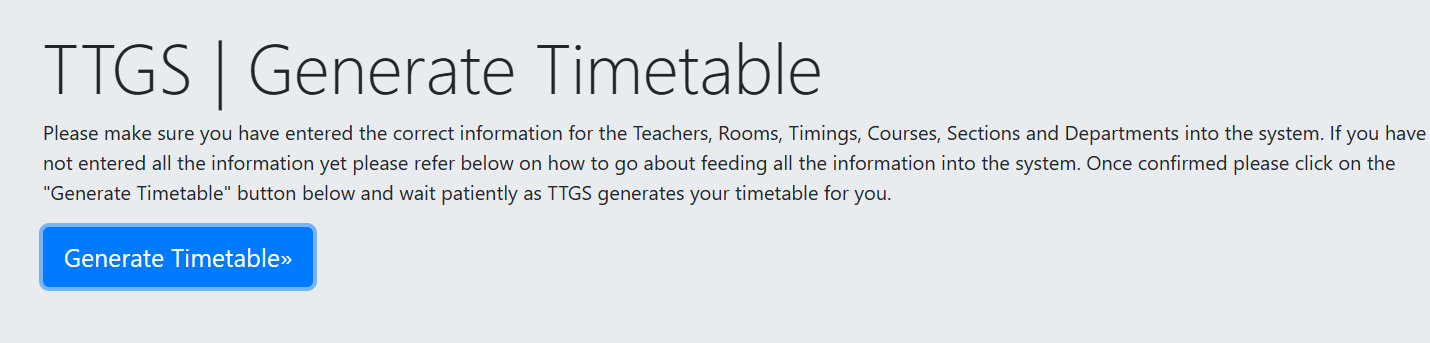
**Add Departments:** Add departments with its corresponding subjects.



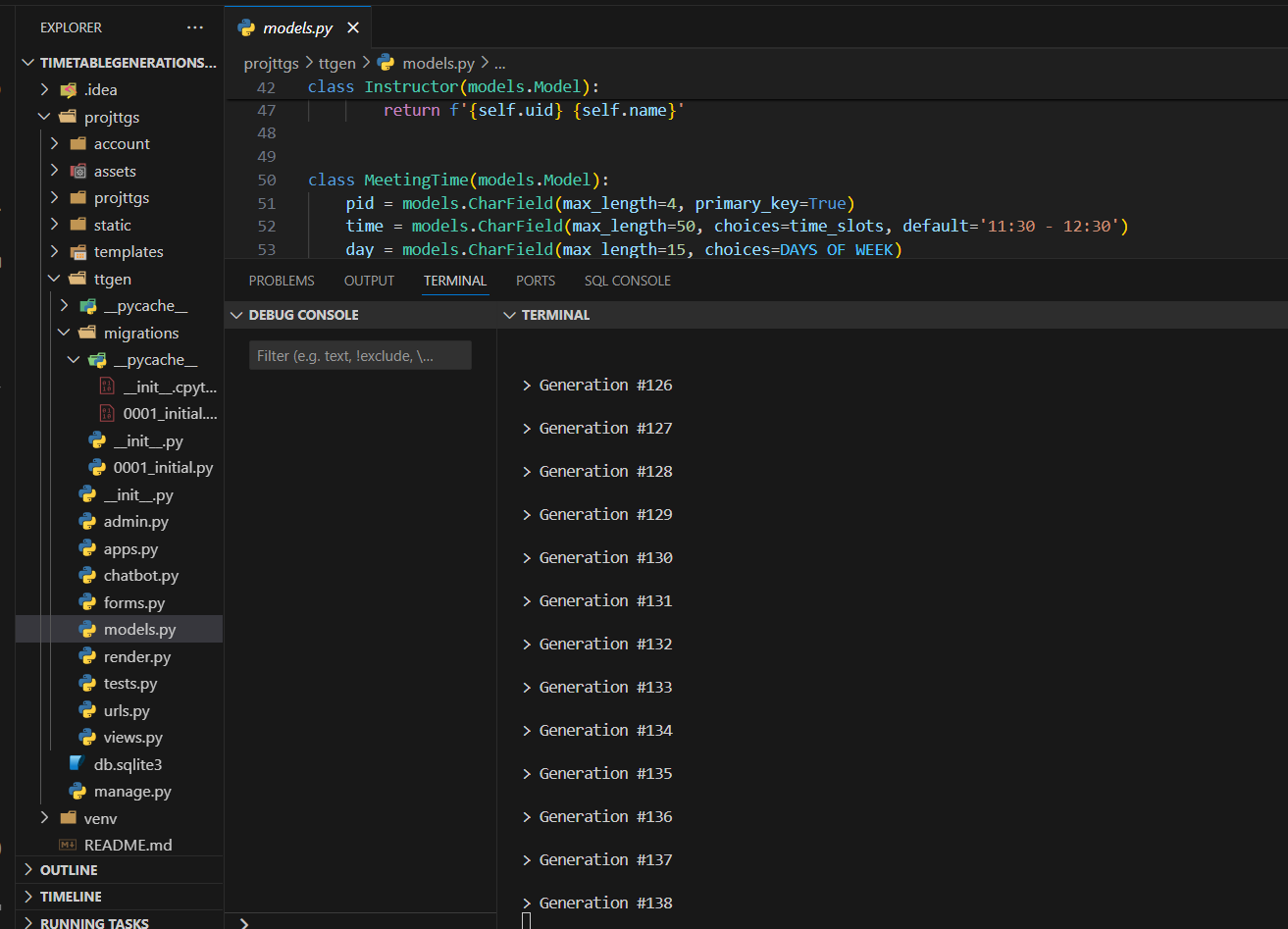
**Add Sections:**

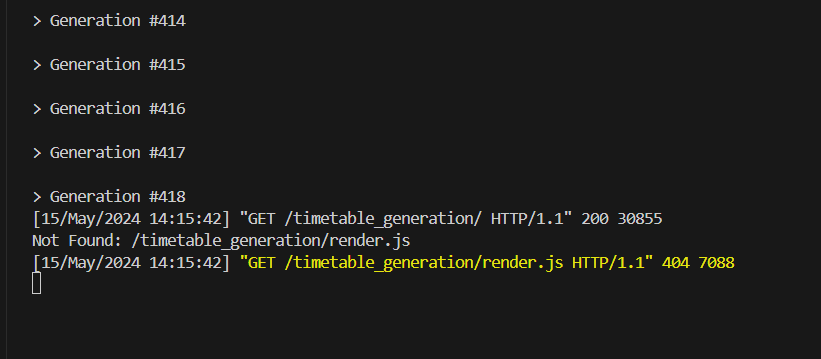


After added all the necessary informations, give generate timetable.



It will generate the combinations of more schedules like this finally it gives the optimized schedules for every classes.

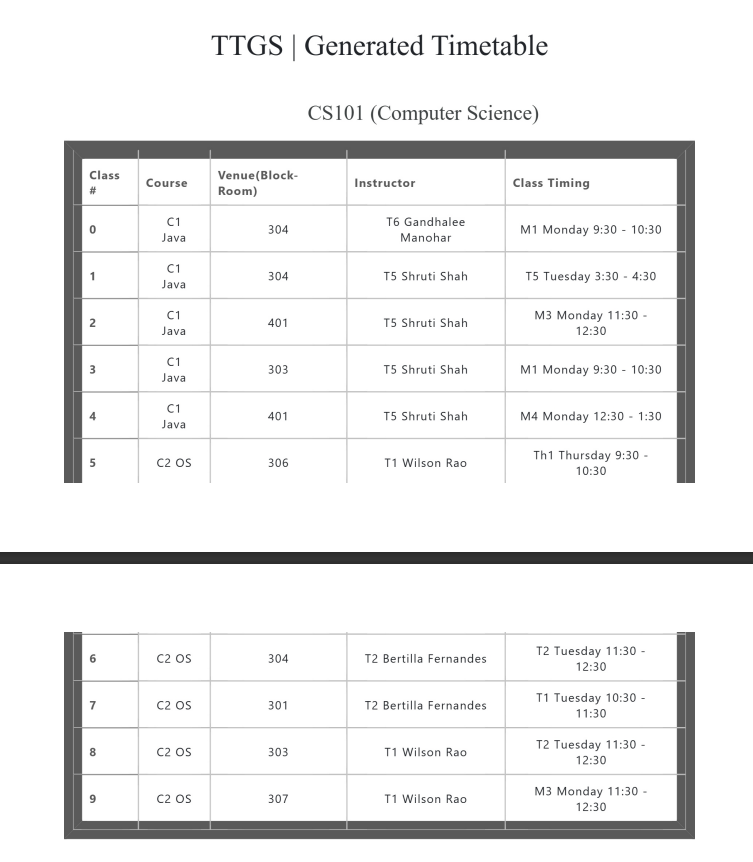


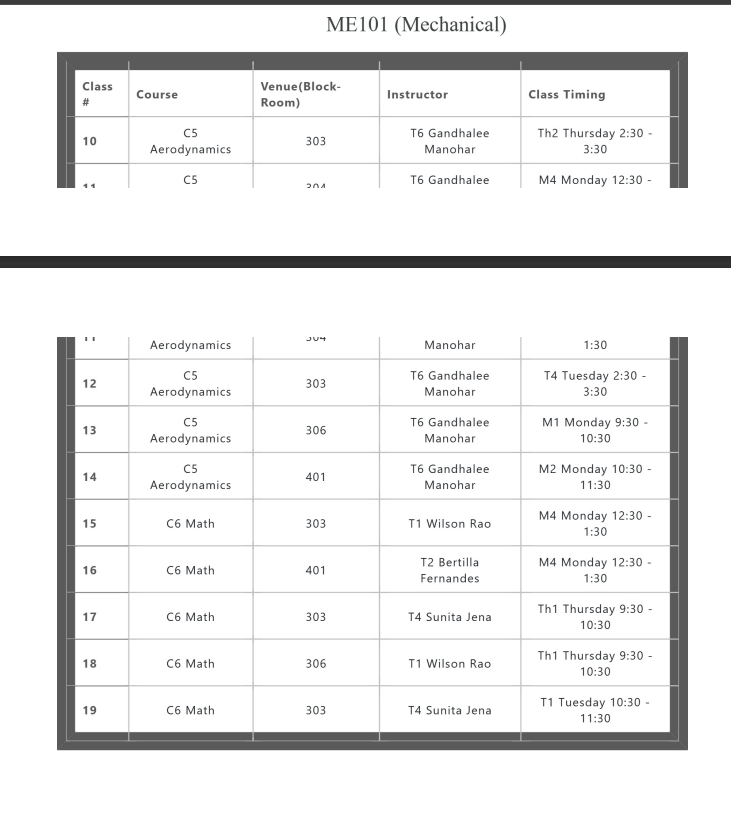


After the crossover,mutation and fitness checked among this generations it will return the optimized schedule for the class. But it takes some time to generate nearly 10 – 15 mins for generating all the below time tables.

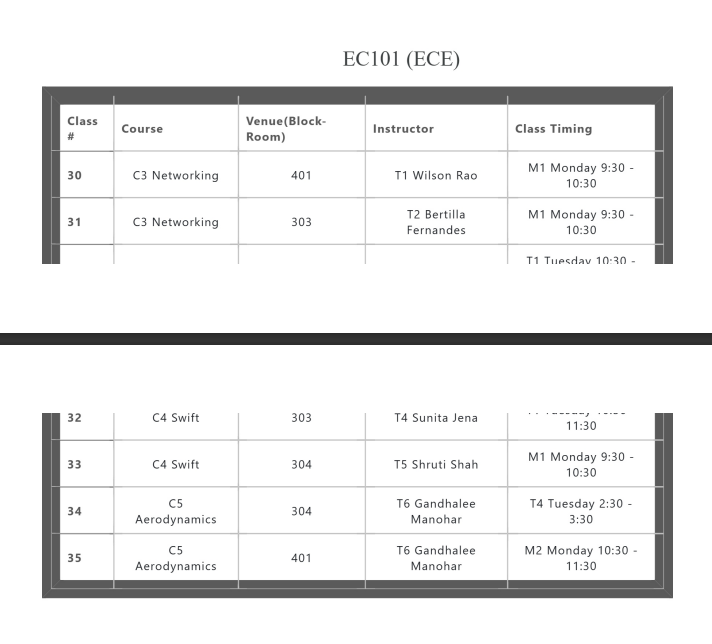
Here, we provide a output by give the sample data, you can see that the in the below time tables there is no collsions between teachers, classrooms.

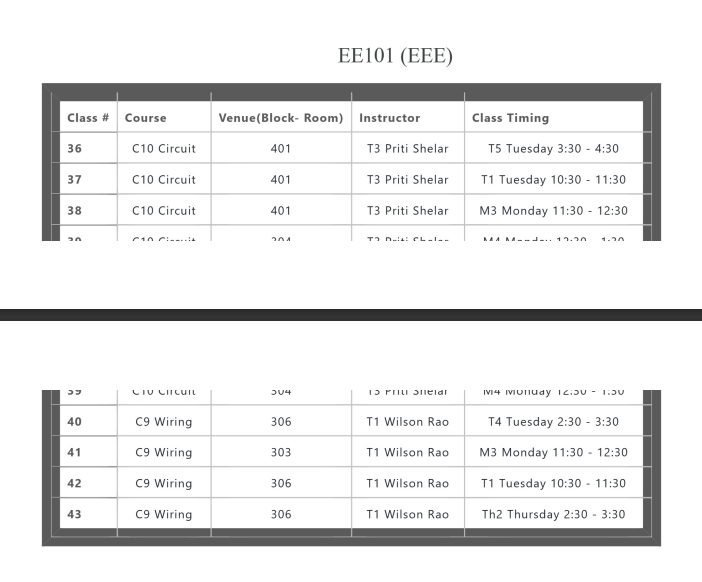
**Output:**



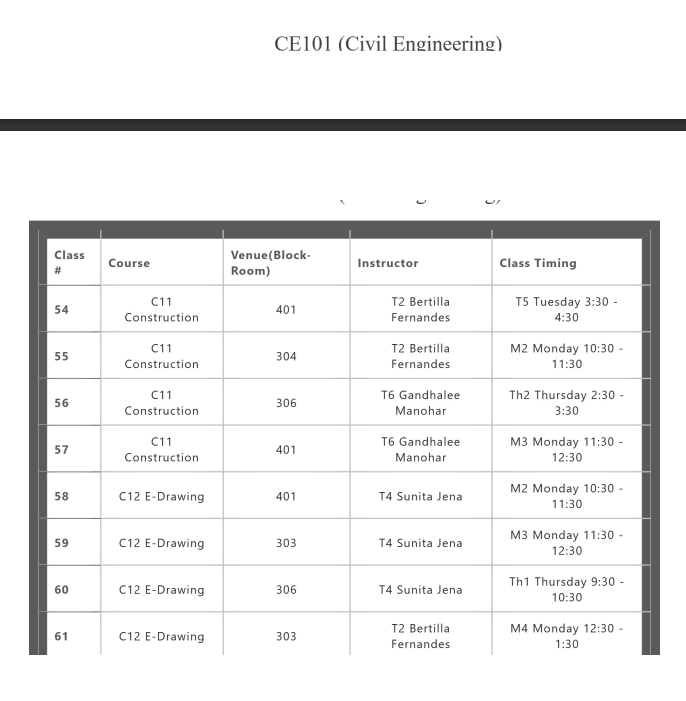


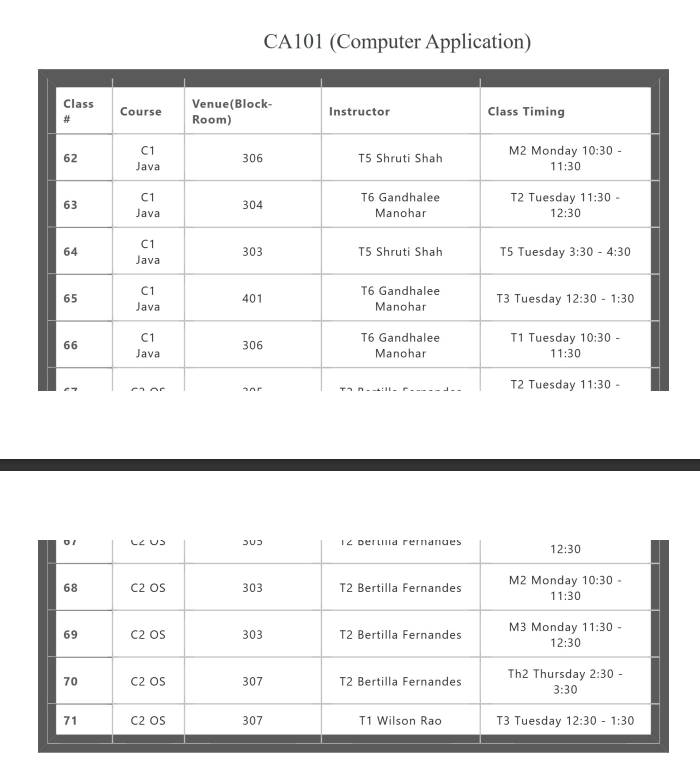




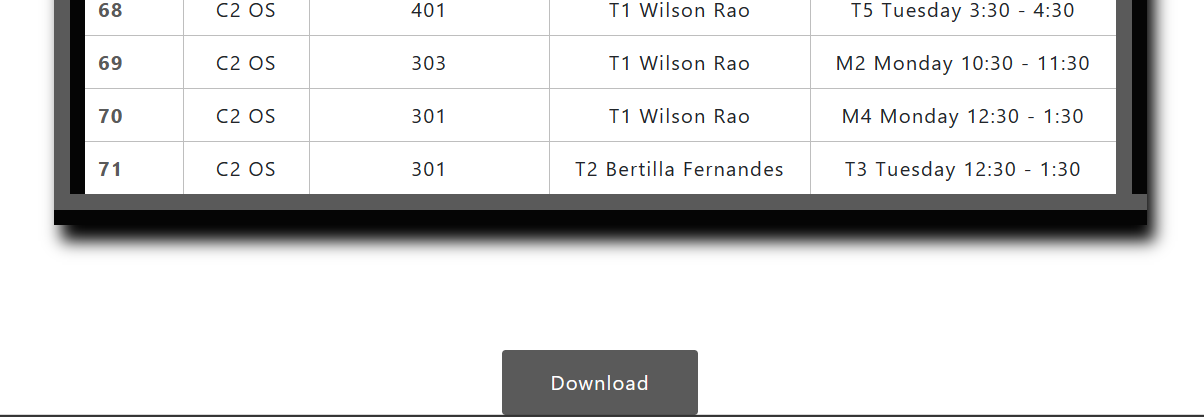






****

**We can download this in pdf format:**

****

In this application we provide a solution for optimized timetable with example by successfully implementing the genetic algorithm.

**Future Aspects:**

Our future aspect is changing the interface that exactly matches to our College by providing free periods,libraries etc, and generating the schedules using this algorithm.