

# **Institute Timetable Automation**

*A Project Report Submitted  
in Partial Fulfillment of the Requirements  
for the Degree of*

**Bachelor of Technology**

*by*

**Mayank Singla (111901030)**

**Rudr Tiwari (111901044)**



INDIAN INSTITUTE  
OF TECHNOLOGY  
**PALAKKAD**

**COMPUTER SCIENCE AND ENGINEERING  
INDIAN INSTITUTE OF TECHNOLOGY PALAKKAD**

# CERTIFICATE

*This is to certify that the work contained in the project entitled “**Institute Timetable Automation**” is a bonafide work of **Mayank Singla (Roll No. 111901030)** and **Rudr Tiwari (Roll No. 111901044)**, carried out in the Department of Computer Science and Engineering, Indian Institute of Technology Palakkad is under my guidance and it has not been submitted elsewhere for a degree.*

**Dr. Albert Sunny**

Assistant Professor

Department of Computer Science & Engineering

Indian Institute of Technology Palakkad

# Acknowledgement

We thank our mentor, Dr. Albert Sunny, for his immense efforts. He collaborated with us and helped us by explaining the complete problem statement and reviewing our ideas and algorithms for the project. We would also like to thank Shiva Santhosh and Sirpa Sahul for exposing us and giving some basic ideas of how we can continue on the foundation of the project they left last year.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	About the Project . . . . .	1
1.2	Challenges . . . . .	1
1.3	Constraints . . . . .	3
1.3.1	Hard Constraints . . . . .	3
1.3.2	Soft Constraints . . . . .	3
1.4	Organization of The Report . . . . .	4
<b>2</b>	<b>Review of Prior Works</b>	<b>5</b>
2.1	Survey . . . . .	5
2.2	Approaches in the last semester . . . . .	6
2.3	Conclusion . . . . .	6
<b>3</b>	<b>Project Workflow</b>	<b>7</b>
3.1	Overall Workflow of the Project . . . . .	7
3.2	Conclusion . . . . .	8
<b>4</b>	<b>Dataset of Class Schedule</b>	<b>9</b>
4.1	Main Classes . . . . .	9
4.2	Enums . . . . .	10
4.3	Example Dataset used in the Algorithms . . . . .	10

4.3.1	Randomly Generated Dataset . . . . .	11
4.3.2	Actual Timetable Dataset . . . . .	13
4.4	Conclusion . . . . .	14
<b>5</b>	<b>Genetic Algorithm</b>	<b>15</b>
5.1	What is Genetic Algorithm? . . . . .	15
5.1.1	Genetic Algorithm Workflow . . . . .	16
5.1.2	Initial Population . . . . .	17
5.1.3	Fitness Function . . . . .	17
5.1.4	Selection . . . . .	17
5.1.5	Crossover . . . . .	17
5.1.6	Mutation . . . . .	18
5.2	Using Genetic Algorithm for Timetabling Problem . . . . .	18
5.2.1	Randomised Approach . . . . .	18
5.2.2	Greedy and Maximal Approach . . . . .	25
5.3	Conclusion . . . . .	30
<b>6</b>	<b>Simulated Annealing</b>	<b>31</b>
6.1	What is Simulated Annealing? . . . . .	31
6.1.1	Hill Climbing Algorithm . . . . .	31
6.1.2	Fixing the Hill Climbing algorithm . . . . .	33
6.1.3	Simulated Annealing Workflow . . . . .	36
6.2	Using Simulated Annealing (SA) in Timetabling Problem . . . . .	37
6.2.1	Why can we use SA in the timetabling problem? . . . . .	37
6.2.2	Initializing the start state . . . . .	37
6.2.3	Neighborhood searching . . . . .	38
6.2.4	Defining the cost function . . . . .	39
6.2.5	Choosing the temperature and acceptance probability function . . . . .	40

6.2.6	Reheating . . . . .	41
6.3	Results and Observations . . . . .	41
6.4	Conclusion . . . . .	42
<b>7</b>	<b>Greedy Initialization for Simulated Annealing</b>	<b>43</b>
7.1	Why do we need a greedy initialization? . . . . .	43
7.2	Problem Formulation . . . . .	44
7.3	Algorithm . . . . .	44
7.3.1	Course Selection Heuristic . . . . .	46
7.3.2	Slot-Room Selection Heuristic . . . . .	47
7.4	Results and Observations . . . . .	48
7.5	Conclusion . . . . .	48
<b>8</b>	<b>Web Application</b>	<b>49</b>
8.1	Understanding the Repository Workspace . . . . .	49
8.1.1	Polyrepo vs Monorepo . . . . .	49
8.1.2	Understanding Nx (by Nrwl) . . . . .	50
8.2	Understanding the Backend . . . . .	52
8.2.1	Tech Stack . . . . .	52
8.2.2	Configuring MongoDB Atlas . . . . .	53
8.2.3	Configuring Google OAuth Consent Screen and Creating Web Client ID	55
8.2.4	REST API Endpoints . . . . .	57
8.2.5	Understanding the need for BullMQ . . . . .	60
8.2.6	Testing APIs with Postman . . . . .	61
8.3	Understanding the Frontend . . . . .	63
8.3.1	Tech Stack . . . . .	63
8.3.2	State Management with Redux Toolkit . . . . .	64
8.3.3	Google Authentication Workflow . . . . .	65

8.3.4	Web Interface Features . . . . .	66
8.4	Workspace Dependency Graph . . . . .	78
8.5	Conclusion . . . . .	78
<b>9</b>	<b>Future Work</b>	<b>79</b>
	<b>References</b>	<b>80</b>

# Chapter 1

## Introduction

### 1.1 About the Project

This project aims to develop an automated solution where faculty and academic section can enter the data about the courses offered in a semester, the list of faculties offering courses, the slot system, and the details of available rooms. It should also provide a workflow that allows the academic section to create an optimal timetable (room-slot-course allocation) by considering the various constraints.

The link to the entire project can be found here: [GitHub](#)

### 1.2 Challenges

There are multiple challenges involved in the project. We list below the challenges which we will be trying to address in our project as of now.

1. **Room allocation** - Since the timetable is accompanied by rooms in which the classes for that course will hold, we will try to allocate rooms for each course as well. We will have to keep in mind room capacity and course size in order to achieve this in timetable allocation



2. **Minimal clashes** - We expect to generate a slot system using our algorithms in such a way that the number of slot clashes is minimal for all the branches in the institute
3. **Common course allocation** - Needless to say, the project should handle the common courses as well in the timetable. Thus it should be made sure that slots for the common courses are the same across the streams
4. **Faculty preferences** - We want the allocation system to have an option for faculty to give their preferences before the allocation is done as to when and where they would like to have their courses held. The allocation system will then try to produce a timetable that best fits all the faculty preferences
5. **Multiple campuses** - Since, currently, our institute has two campuses, this produces further challenges for timetable allocation. It should be made sure that most of the classes for a batch should be held on their campus of residence. The allocation should try to minimize faculty and student travel. It should also be made sure that a faculty or student does not have classes on the alternative campus on consecutive slots
6. **Integration with MRBS** - We would want the room allocation generated by our project to be automatically updated on the MRBS servers of IIT Palakkad
7. **Credit-based system compatibility** - Since our college will be moving to a credit-based system shortly, the project should be either compatible or easily extensible to such a system whenever needed
8. **Adjustment by admins** - Since the timetable generated by the algorithm need not be perfect, the portal should allow admins to manually change some of the slots in both timetable and room allocation at a later time

## 1.3 Constraints

Based on the above challenges, we can reduce our problem into the following hard and soft constraints.

### 1.3.1 Hard Constraints

- **Faculty Conflict:** No faculty should have more than one class at a time
- **Student Conflict:** No student group should have more than one class at a time
- **Room Conflict:** A class should be placed only in a spare classroom
- **Room Capacity:** A classroom must have enough seats to accommodate all the students
- **Room Features:** A classroom must have laboratory equipment (ex. computers) if the class requires it
- **Distinct Periods:** All lectures of a course must be scheduled in distinct periods

### 1.3.2 Soft Constraints

- There should be enough time for traveling between the campuses for both faculty and students if they need to
- Maximum number of times a faculty or a student group needs to travel between the campuses should not be greater than 1
- All the lectures of a course should be in the same room
- All the lectures of a course should be evenly distributed across different weekdays
- The courses are given slots based on the slot preferences of faculties

## 1.4 Organization of The Report

This chapter provided a complete description of the problem statement, the various challenges that need to tackle in the project, and how we reduced our problem into different hard and soft constraints. Thus, it also briefly covers the expected features and deliverables that the project has to offer. In the further chapters, we will dive into the proposed workflow and implementation of the different algorithms for timetable automation. Chapter 3 will cover the expected workflow of our final project and chapter 4 will mention the required dataset for the timetabling problem. Chapter 5 and Chapter 6 will cover the genetic algorithm and simulated annealing approach for solving the timetabling problem respectively and the various results of these algorithms. Chapter 7 will cover the greedy initialization approach for providing input to the simulated annealing algorithm for solving the timetable problem. Chapter 8 will cover the technologies used for building the web application and their detailed configuration and usage. Finally, Chapter 9 covers the future work and scope of the project.

# Chapter 2

## Review of Prior Works

### 2.1 Survey

We read the report shared by our mentor that was created by the students working on this project earlier. That report tells us the data requirements for the project that the academic section will provide. We were earlier not clear on what is meant by the Slot Systems in the data requirements that they mentioned. This was cleared by our mentor in a discussion session. The intention for getting the slot system was that they were trying to fit the courses in a fixed slot system already made by the academic section.

The report next tells us the different roles that will be involved and the workflow for the whole timetable creation process. It describes mainly three roles the admin, the faculty, and the stream coordinator. Admin is the one who will have full access to the entire system and will be responsible for uploading all the data. Faculty will be responsible for giving their preferences for the slots. The stream coordinator is the one who will act as a stream in charge for each stream and will be executing the algorithm for their respective streams and will also solve any clashes manually that might occur in slot allotment.

Lastly, the report listed the various REST API endpoints and the database schema they constructed for their web interface.

## 2.2 Approaches in the last semester

To minimize the clashes in the constraints as much as possible, our first attempt was to create a slot system as well along with the timetable. To do this we opted for a combination of brute force and greedy approaches to get a slot system simultaneously with the timetable. We were able to successfully implement this approach, but based on several reviews, it was concluded that having a fixed slot system is more convenient for both the students and the faculty. We also constructed a basic web interface for quicker and faster visualization of the results of our algorithm.

## 2.3 Conclusion

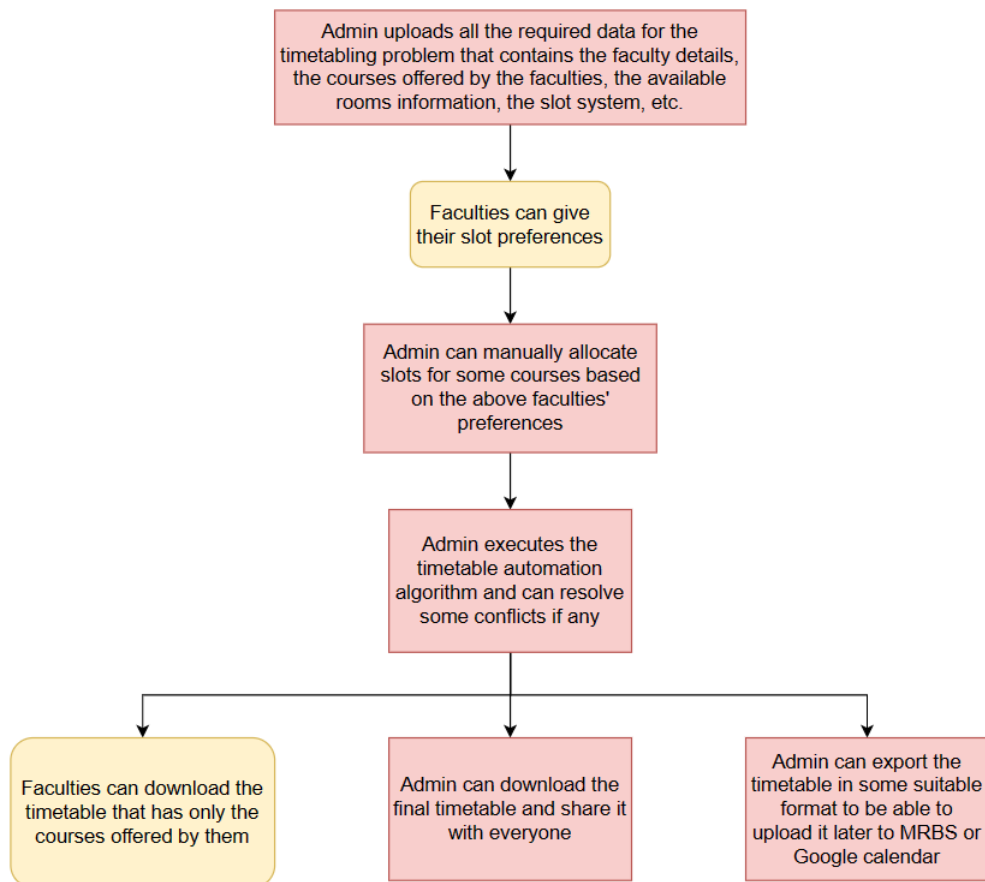
This chapter discusses the old report that clearly explains the data requirements, the overall workflow of the project, and the REST API endpoints that could potentially be made for the web interface. However, the actual algorithm for the slot and room allocation for the project was yet to be constructed. It then discussed the initial approaches that we tried during the last semester and the web interface for the visualization of the timetable.

In the upcoming chapters, we will first discuss the workflow of our project from the point of view of a user (who is anyone using the web interface). Then we will discuss and compare the main algorithms, the **Genetic Algorithm**, and the **Simulated Annealing Algorithm**, which we are going to use for the timetable scheduling problem.

# Chapter 3

## Project Workflow

### 3.1 Overall Workflow of the Project



**Fig. 3.1** Overall Workflow of the Project

## 3.2 Conclusion

In this chapter, we proposed the workflow for our project that involves various phases that could be done on the final web interface. In the next chapter, we will discuss the main classes and enumerated types required while scheduling a curriculum timetable along with some sample data.

# Chapter 4

## Dataset of Class Schedule

### 4.1 Main Classes

- **Faculty:** The Faculty class has an ID and the name of the faculty.
- **Department:** The Department class has an ID, the name of the department, and the list of faculties belonging to that department. IDs are generated internally and automatically.
- **Room:** The Room class has an ID, the name of the room, the type of lecture the room should be used, the seating capacity, and the campus in which the room is located. IDs are generated internally and automatically.
- **Course:** The Course class has an ID, the course code, the name of the course, the credits offered by the course in the *L-T-P-C* format, the type of the course, the lecture type of the course, the maximum number of students the course can have, the list of faculties that are offering the course, the department which is offering the course, and whether the course requires a slot or not. IDs are generated internally and automatically.
- **Slot:** The Slot class has an ID, the name used to represent the slot, the type of lecture



of the slot, the list of timings of the slot, and the total credits the slot is equivalent to. IDs are generated internally and automatically.

- **Class:** The Class class has an ID, a reference to the course to which the class belongs, a reference to the slot, and the room to which the class is allocated. IDs are generated internally and automatically.

## 4.2 Enums

- **Course Type:** The Course Type could either be Common, GCE, HSE, SME, OE, PMT, PMP, PME, or Project.
- **Lecture Type:** The Lecture Type could either be a normal lecture or a lab lecture.
- **Departments:** The Departments could either be CSE, EE, ME, CE, GCE, SME, HSE, or General.

## 4.3 Example Dataset used in the Algorithms

	08:00-08:50	09:00-09:50	10:00-10:50	11:00-11:50	12:00-12:50	13:00-13:50	14:00-15:15	15:30-16:45	17:00-17:50	
Mon	B1	A	H	D	E	Lunch Break	L	K	C	
			P1				P4			
Wed	C	A	B	H	E		C1	EML		D
			P2							
Fri	H	C	D	B	E		K	L	F	
			P3				P7			

	08:00-08:50	09:00-10:15	10:30-11:45	12:00-12:50	13:00-13:50	14:00-15:15	15:30-16:45	17:00-17:50	
Tue	D1	F	G	M	Lunch Break	I	J	A	
						P5			
Thu	A1	F	G	M		J	I	B	
					P6				

Fig. 4.1 Sample Slot System

#### 4.3.1 Randomly Generated Dataset

Room #	Campus	Capacity	LectureType
R1	Ahalia	60	Normal
R2	Nila	100	Normal
R3	Ahalia	45	Normal
R4	Nila	60	Normal
R5	Ahalia	100	Normal
R6	Nila	45	Normal
R7	Ahalia	60	Normal
R8	Nila	100	Normal
R9	Ahalia	100	Lab
R10	Nila	60	Normal
R11	Ahalia	100	Normal

**Fig. 4.2** Sample List of Rooms

Course	Department	Credits	Course Type	Lecture Type	Max # of Students	Faculties
C1	CSE	4	PMT	Normal	35	F01
C2	CSE	5	PMT	Normal	45	F02
C3	CSE	3	PMP	Lab	100	F01
C4	CSE	3	PMP	Lab	100	F02
C5	CSE	4	PME	Normal	35	F01
C6	EE	4	PMT	Normal	35	F11
C7	EE	5	PMT	Normal	45	F12
C8	EE	3	PMP	Lab	100	F11
C9	EE	3	PMP	Lab	100	F12
C10	EE	4	PME	Normal	35	F11

**Fig. 4.3** Sample List of Courses

#### 4.3.2 Actual Timetable Dataset

Room #	Campus	Capacity	LectureType
A 112	Ahalia	60	Normal
A 18	Ahalia	60	Normal
A 33	Ahalia	60	Normal
A Aud	Ahalia	60	Normal
A Lab1	Ahalia	100	Lab
N 204	Nila	60	Normal
N 301	Nila	60	Normal
N 303	Nila	60	Normal
N Lab4	Nila	100	Lab

Fig. 4.4 Sample List of Rooms

Code	Course	Department	Credits	Course Type	Lecture Type	Max # of Students	Faculties
ID1110	Introduction to Programming	GENERAL	4	COMMON	Lab	100	Albert Sunny, Srimanta Bhattacharya, Sovan Lal Das, Piyush P. Kurur
HS2000	Professional Ethics	GENERAL	2	COMMON	Normal	60	Anoop George
HS2031	Principles of Economics	HSE	3	HSE	Normal	60	Amrita Roy
MA2032	Numerical Analysis	MAE	3	MAE	Normal	60	HVR Mittal
HS4605	Foundations of Linguistics	GCE	3	GCE	Normal	60	Reenu Punnoose
CE2020	Structural Analysis	CE	4	PMT	Normal	60	Madhu Karthik M
CS2020	Discrete Mathematics for Computer Science	CSE	3	PMT	Normal	60	Jasine Babu
CS5016	Computational Methods and Applications	CSE	4	PME	Normal	60	Albert Sunny
EE2020	Electrical Machines	EE	4	PMT	Normal	60	Arun Rahul S
ME2020	Gas Dynamics	ME	3	PMT	Normal	60	Ganesh Natarajan

**Fig. 4.5** Sample List of Courses

## 4.4 Conclusion

In this chapter, we discussed the main classes and enumerated types required while scheduling a curriculum timetable. We also saw sample data that we will be using for testing the implemented algorithms for timetable automation. In the next chapter, we will discuss the **Genetic Algorithm** for solving the timetable automation problem

# Chapter 5

## Genetic Algorithm

### 5.1 What is Genetic Algorithm?

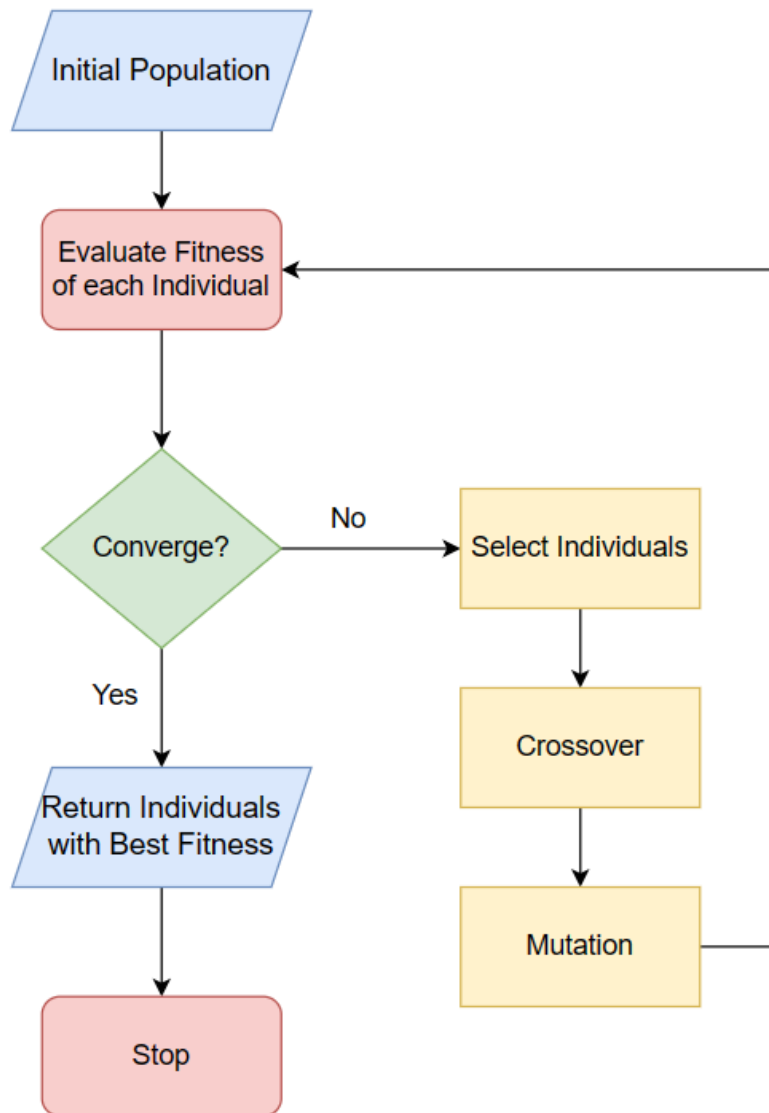
A genetic Algorithm is a kind of evolutionary, randomized, and adaptive heuristic search algorithm. It is **based on genetics and natural selection**. It is often used to generate a high-quality solution for an optimization problem.

The main idea behind this algorithm is the process of natural selection where the fittest individuals are selected for reproduction to produce offspring for the next generation. The offspring produced inherits the characteristics from their parents. If the parents have better fitness, their offspring are expected to have better fitness than their parents. This process keeps on iterating until a generation with the fittest individuals is found.

The different phases that are considered in the genetic algorithm are:

1. Initial Population
2. Fitness Function
3. Selection
4. Crossover
5. Mutation

### 5.1.1 Genetic Algorithm Workflow



**Fig. 5.1** Genetic Algorithm Workflow

### 5.1.2 Initial Population

We initially have a set of individuals which is called a **population**. Each individual can be considered as a possible solution to the problem. Each individual can be treated as a **chromosome** and each chromosome has many **genes**. The representation of the genes will vary depending on the problem we are trying to solve.

### 5.1.3 Fitness Function

The **fitness function** determines how fit an individual is. It is used to compare an individual's ability to compete with other individuals. Each individual is given a fitness score and based on the problem, we will either try to maximize or minimize the fitness score of the fittest individual. Based on the fitness score, we will define some **convergence conditions** for the algorithm to terminate.

### 5.1.4 Selection

In this phase, we try to select the fittest individuals and let them pass their genes to the next generation. The pair of individuals are selected from their fitness score and the individuals with a better fitness score will have more chance to be selected for reproduction. There are multiple selection techniques like Tournament Selection, Roulette Wheel Selection, and Rank-Based Selection. We can also have a small set of **elite individuals** which are the fittest individuals and are passed to the next generation as it is.

### 5.1.5 Crossover

In this phase, the selected pair of parents(chromosomes) to be mated generate a new offspring using their genes. **Offspring** are created by using the genes of parents among themselves using multiple techniques like Single Point Crossover, Multi-point Crossover, and Uniform Crossover with Crossover Mask.



### 5.1.6 Mutation

In some of the offspring generated, some of the genes can **mutated** with a low probability. It occurs to maintain diversity within the population and prevent premature convergence.

## 5.2 Using Genetic Algorithm for Timetabling Problem

The following are the parameters (and their values used) involved in the algorithm.

- POPULATION\_SIZE (= 9)
- NUMBER\_ELITE\_SCHEDULES (= 1)
- TOURNAMENT\_SELECTION\_SIZE (= 3)
- Crossover\_PROBABILITY (= 0.5)
- MUTATION\_PROBABILITY (= 0.1)

These parameters can be varied and they will affect the rate of convergence and quality of the results from the genetic algorithm.

### 5.2.1 Randomised Approach

#### Creating Schedules

We can represent a schedule (**or a chromosome/individual**) as a list of **Class objects**. Each class in this list is allotted a random room with enough room capacity and which also satisfies the required room features. Also, each class in this list is allotted random slots such that they are enough to satisfy the credit requirements of the course associated with the class. This way all the classes in all the schedules are initialized randomly to create the initial population.

## Fitness Evaluation

For a given schedule in a population, we find the number of different conflicts present in that schedule based on the different hard constraints mentioned before. Then we calculate the fitness of a schedule as

$$fitness = \frac{1}{\#conflicts + 1}$$

Also, we create a set of multipliers, and based on the different soft constraints, we add some random value less than 1 to make a schedule lose fitness if it doesn't satisfy any soft constraint and add some random value greater than 1 to make a schedule gain fitness if it satisfies any soft constraint.

The algorithm will **terminate** when its fitness value has reached greater than or equal to 1.

## Selecting Schedules

We use **tournament selection** to select parents for the crossover to generate new offspring for the next generation. In each tournament selection round, we randomly select some schedules and pick the one with the highest fitness value as the parent. Hence, for each crossover, we perform two tournament selection rounds.

Also, there is a set of **elite schedules** which are the schedules with the best fitness, which are passed to the next generation as it is without any changes.

## Crossover of Schedules

To generate a new offspring schedule from the two parent schedules, a random number is generated, and based on the crossover probability defined, the class from the offspring is taken either from the first schedule or from the second schedule.

## Mutating Schedule

We first create and initialize a completely new schedule as before. Then, for each class in the schedule that we want to mutate, we generate a random number and based on the mutation probability, we alter the class in this schedule by replacing it with the corresponding class in the newly generated schedule.

## Results

Here are the results obtained from the algorithm for a small number of input courses. We can verify that there are no conflicts in the resultant timetable obtained and all the hard constraints are also satisfied.

```
Generation: 1      Conflicts: 29, Fitness : 0.0000039583333333333333
Generation: 2      Conflicts: 29, Fitness : 0.0000039583333333333333
Generation: 3      Conflicts: 29, Fitness : 0.0000039583333333333333
Generation: 4      Conflicts: 28, Fitness : 0.000016379310344827585
Generation: 5      Conflicts: 24, Fitness : 0.000019
Generation: 6      Conflicts: 24, Fitness : 0.000019
Generation: 7      Conflicts: 24, Fitness : 0.000019
Generation: 8      Conflicts: 26, Fitness : 0.00004398148148148147
Generation: 9      Conflicts: 26, Fitness : 0.00006597222222222222
Generation: 10     Conflicts: 25, Fitness : 0.00009134615384615384
```

**Fig. 5.2** Initial stage of the algorithm

```

Generation: 1404
                Conflicts: 1, Fitness : 0.00375
Generation: 1405
                Conflicts: 1, Fitness : 0.00375
Generation: 1406
                Conflicts: 1, Fitness : 0.00375
Generation: 1407
                Conflicts: 1, Fitness : 0.00375
Generation: 1408
                Conflicts: 1, Fitness : 0.00375
Generation: 1409
                Conflicts: 1, Fitness : 0.00375
Generation: 1410
                Conflicts: 1, Fitness : 0.00375
Generation: 1411
                Conflicts: 1, Fitness : 0.00375
End:
                Conflicts: 0, Fitness : 1

```

**Fig. 5.3** Final stage of the algorithm

Class #	Course (Department, Lecture Type, Credits) (Max # of Students) [Faculties]	Room (Campus, Lecture Type, Capacity)	Slot (Name, Day, Credits)
CLS1	<C1 (CSE, Normal, 4) (35) [F01]>	<R16 (Nila, Normal, 60)>	(B1, 1) (J, 3)
CLS2	<C2 (CSE, Normal, 5) (45) [F02]>	<R6 (Nila, Normal, 45)>	(K, 3) (G, 3)
CLS3	<C3 (CSE, Lab, 3) (100) [F01]>	<R9 (Ahalia, Lab, 100)>	(P1, 3)
CLS4	<C4 (CSE, Lab, 3) (100) [F02]>	<R29 (Ahalia, Lab, 100)>	(P2, 3)
CLS5	<C5 (CSE, Normal, 4) (35) [F01]>	<R8 (Nila, Normal, 100)>	(F, 4)
CLS6	<C6 (EE, Normal, 4) (35) [F11]>	<R17 (Ahalia, Normal, 100)>	(F, 4)
CLS7	<C7 (EE, Normal, 5) (45) [F12]>	<R4 (Nila, Normal, 60)>	(C, 3) (A, 3)

**Fig. 5.4** Sample resultant allocation w.r.t. Classes

Slot (Lecture Type, Credits)	Classes (Course, Department, Room, Faculties)
<B1 (Normal, 1)>	<C1,CSE,R16,[F01]>, <C12,ME,R24,[F22]>
<A (Normal, 3)>	<C7,EE,R4,[F12]>, <C16,CE,R5,[F31]>, <C30,SME,R12,[F55]>
<H (Normal, 3)>	<C17,CE,R3,[F32]>, <C23,GCE,R7,[F43]>
<D (Normal, 3)>	<C21,GCE,R27,[F41]>
<E (Normal, 3)>	<C35,HSE,R19,[F65]>, <C39,GENERAL,R2,[F32,F42,F52]>
<L (Normal, 3)>	<C17,CE,R3,[F32]>, <C26,SME,R13,[F51]>
<K (Normal, 3)>	<C2,CSE,R6,[F02]>, <C12,ME,R24,[F22]>, <C33,HSE,R11,[F63]>
<C (Normal, 3)>	<C7,EE,R4,[F12]>, <C38,GENERAL,R2,[F01,F11,F21]>
<P1 (Lab, 3)>	<C3,CSE,R9,[F01]>, <C8,EE,R29,[F11]>

**Fig. 5.5** Sample resultant allocation w.r.t. Slots

Faculty	Classes (Course, Department, Max # of Students, Room, Slots)
F01	<C1,CSE,35,R16,[B1,J]>, <C3,CSE,100,R9,[P1]>, <C5,CSE,35,R8,[F]>, <C38,GENERAL,100,R2,[C]>, <C40,GENERAL,100,R26,[G]>
F02	<C2,CSE,45,R6,[K,G]>, <C4,CSE,100,R29,[P2]>
F11	<C6,EE,35,R17,[F]>, <C8,EE,100,R29,[P1]>, <C10,EE,35,R20,[M,C1,D1]>, <C38,GENERAL,100,R2,[C]>, <C40,GENERAL,100,R26,[G]>
F12	<C7,EE,45,R4,[C,A]>, <C9,EE,100,R9,[P7]>
F21	<C11,ME,35,R6,[F]>, <C13,ME,100,R29,[P3]>, <C15,ME,35,R22,[C1,M,A1]>, <C38,GENERAL,100,R2,[C]>
F22	<C12,ME,45,R24,[D1,B1,K]>, <C14,ME,100,R19,[P6]>, <C41,GENERAL,100,R17,[G]>
F31	<C16,CE,35,R5,[A,C1]>, <C18,CE,100,R9,[P3]>, <C20,CE,35,R3,[F]>
F32	<C17,CE,45,R3,[H,L]>, <C19,CE,100,R19,[P5]>, <C39,GENERAL,100,R2,[E]>, <C41,GENERAL,100,R17,[G]>
F41	<C21,GCE,35,R27,[A1,D]>

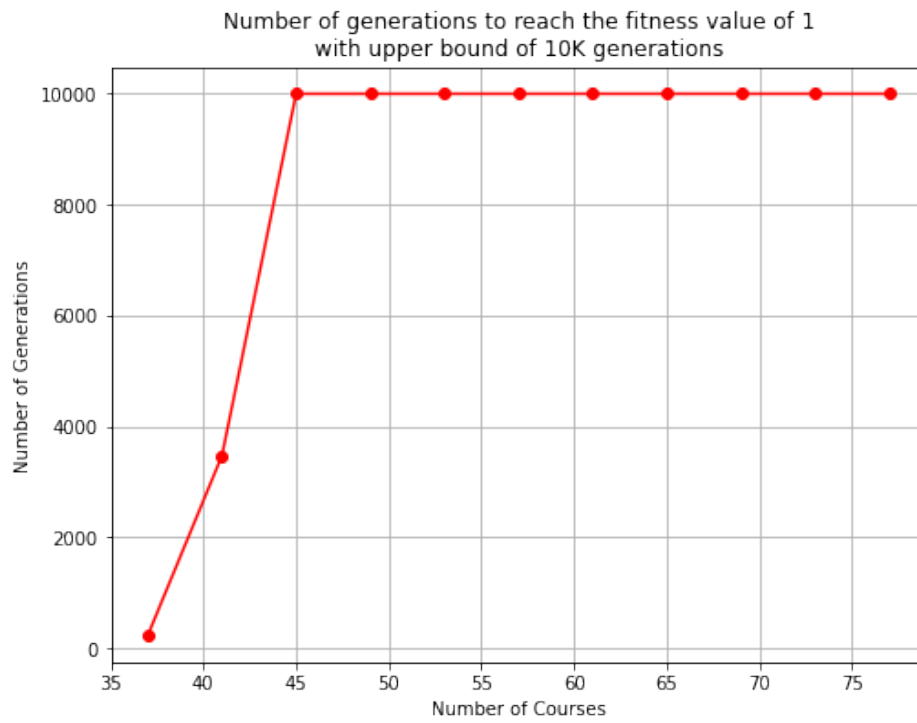
**Fig. 5.6** Sample resultant allocation w.r.t. Faculties

Room (Name, Campus, Lecture Type, Capacity)	Classes (Course, Department, Max # of Students, Faculties, Slots)
<R1 (Ahalia, Normal, 60)>	<C28,SME,60,[F53][G]>
<R3 (Ahalia, Normal, 45)>	<C17,CE,45,[F32][H,L]>, <C20,CE,35,[F31][F]>
<R4 (Nila, Normal, 60)>	<C7,EE,45,[F12][C,A]>
<R5 (Ahalia, Normal, 100)>	<C16,CE,35,[F31][A,C1]>
<R6 (Nila, Normal, 45)>	<C2,CSE,45,[F02][K,G]>, <C11,ME,35,[F21][F]>
<R7 (Ahalia, Normal, 60)>	<C23,GCE,60,[F43][H]>
<R8 (Nila, Normal, 100)>	<C5,CSE,35,[F01][F]>
<R9 (Ahalia, Lab, 100)>	<C3,CSE,100,[F01][P1]>, <C9,EE,100,[F12][P7]>, <C18,CE,100,[F31][P3]>
<R11 (Ahalia, Normal, 100)>	<C24,GCE,100,[F44][B]>, <C33,HSE,60,[F63][K]>
<R12 (Nila, Normal, 45)>	<C30,SME,25,[F55][M,A]>

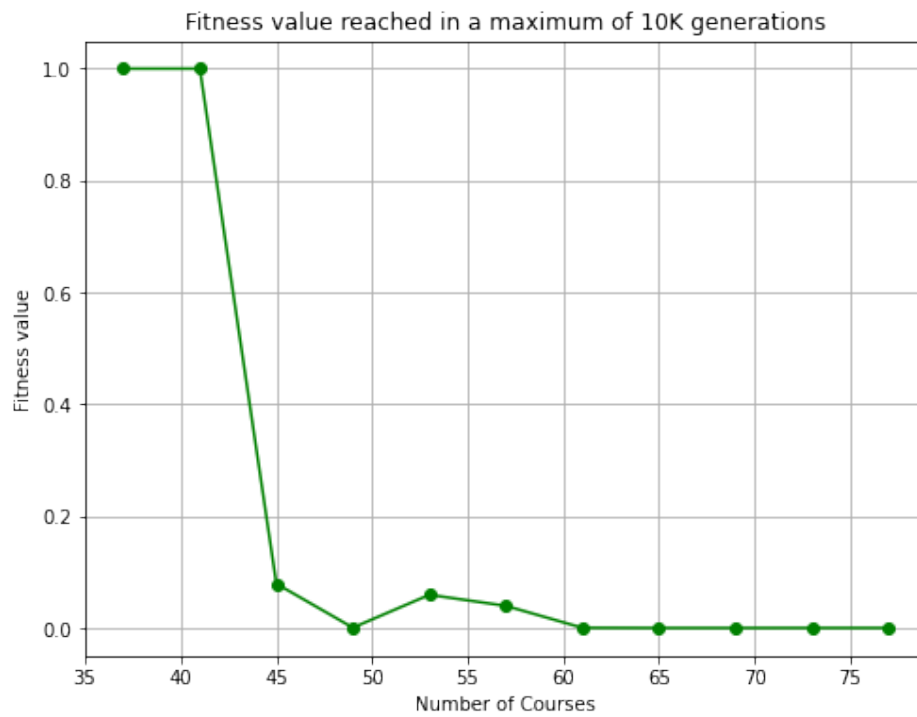
**Fig. 5.7** Sample resultant allocation w.r.t. Rooms

WeekDay	08:00 AM - 08:50 AM	09:00 AM - 09:50 AM	10:00 AM - 12:50 PM	10:30 AM - 11:45 AM	11:00 AM - 11:50 AM	12:00 PM - 12:50 PM	02:00 PM - 03:15 PM	03:30 PM - 04:45 PM	05:00 PM - 05:50 PM
Monday	C1,B1 C12,B1	C7,A C16,A C30,A	C3,P1 C8,P1 C17,H C23,H		C21,D	C35,E C39,E	C17,L C26,L	C2,K C12,K C33,K	C7,C C38,C
Tuesday	C10,D1 C12,D1	C5,F C6,F C11,F C20,F C25,F C27,F		C2,G C28,G C40,G C41,G		C10,M C15,M C22,M C29,M C30,M C32,M	C19,P5 C31,I C34,I	C1,J C22,J	C7,A C16,A C30,A
Wednesday	C7,C C38,C	C7,A C16,A C30,A	C4,P2 C24,B		C17,H C23,H	C35,E C39,E	C10,C1 C15,C1 C16,C1 C25,C1 C27,C1 C29,C1 C32,C1		C21,D
Thursday	C15,A1 C21,A1	C5,F C6,F C11,F C20,F C25,F C27,F		C2,G C28,G C40,G C41,G		C10,M C15,M C22,M C29,M C30,M C32,M	C1,J C14,P6 C22,J	C31,I C34,I	C24,B
Friday	C17,H C23,H	C7,C C38,C	C13,P3 C18,P3 C21,D		C24,B	C35,E C39,E	C2,K C9,P7 C12,K C33,K	C17,L C26,L	C5,F C6,F C11,F C20,F C25,F C27,F

**Fig. 5.8** Final Timetable Obtained



**Fig. 5.9** #Courses vs #Generations



**Fig. 5.10** #Courses vs Fitness Reached

## Observations

The final timetable obtained satisfies all the hard constraints and is well-suited to be used as a timetable for a university. We found out that the algorithm works well and converges quickly for a smaller set of courses. However, as the number of courses increases where faculties are offering many courses in a semester, the number of conflicts to be resolved also increases, and hence the algorithm converges very slowly.

### 5.2.2 Greedy and Maximal Approach

#### Creating Schedules

Here for generating the initial population, we initialize each schedule greedily by allocating the first available slot combination and room which satisfies the required criteria to a class. Hence, we represent a schedule (**or a chromosome/individual**) using two lists, the first containing the list of courses that are successfully allocated a slot and a room without any conflicts between them, and the second containing the list of the courses that will lead to a conflict if added to the first list irrespective of any slot or room and hence are not allocated. This way the list of allocated courses is always maximal.

Because no randomization is involved in this approach, it is expected to produce better and quicker results than the previous approach.

#### Fitness Evaluation

For a given schedule in a population, we define its fitness as

$$fitness = \frac{\#Allocated\ Courses}{\#Total\ Courses}$$

The algorithm will **terminate** when its fitness value has become equal to 1.



## Selecting Schedules

We again use **tournament selection** to select parents for the crossover to generate new offspring for the next generation. In each tournament selection round, we randomly select some schedules and pick the one with the highest fitness value as the parent. Hence, for each crossover, we perform two tournament selection rounds.

Also, there is a set of **elite schedules** which are the schedules with the best fitness, which are passed to the next generation as it is without any changes.

## Crossover of Schedules

For producing a new offspring, we will try to combine the list of courses from both parents in different ways and try to allocate courses to them following that sequence. This way from each pair of parents, we can get multiple offspring and pick the ones with the best fitness for the next generation.

Here we tried four different sequences of the list of courses either by taking the sequence from the start or the end from the two parents, and iterating in the two lists alternatively.

## Mutating Schedule

We will iterate over each course in the allocated list and based on the mutation probability select it for mutation. We will remove this selected course from the allocated list, and try assigning the other courses from the unallocated courses to get a maximal allocation. This way for each selected course, we will get a new maximal allocation of the courses. Out of these new maximal allocations, we will select the one with the best fitness as the result of our mutation.

## Results

Here are the results obtained from the algorithm for a small number of input courses. We can verify that there are no conflicts in the resultant timetable obtained and all the hard constraints are also satisfied.

```
Generation: 1, Total: 45
              Allocated: 45, Fitness: 1
End:
              Allocated: 45, Fitness: 1
```

**Fig. 5.11** Execution of the algorithm

Class #	Course (Department, Lecture Type, Credits) (Max # of Students) [Faculties]	Room (Campus, Lecture Type, Capacity)	Slot (Name, Day, Credits)
CLS37	<C37 (HSE, Normal, 3) (60) [F63]>	(R1, Ahalia)	(F, 4)
CLS34	<C34 (SME, Normal, 3) (25) [F55]>	(R2, Nila)	(F, 4)
CLS29	<C29 (GCE, Normal, 3) (25) [F45]>	(R3, Ahalia)	(F, 4)
CLS16	<C16 (ME, Lab, 3) (100) [F24]>	(R9, Ahalia)	(P1, 3)
CLS26	<C26 (GCE, Normal, 3) (45) [F42]>	(R4, Nila)	(F, 4)
CLS8	<C8 (EE, Normal, 5) (45) [F12]>	(R5, Ahalia) (R5, Ahalia)	(F, 4) (A, 3)

**Fig. 5.12** Sample resultant allocation w.r.t. Classes

Slot (Lecture Type, Credits)	Classes (Course, Department, Room, Faculties)
<A (Normal, 3)>	<C8,EE,R5,[F12]>, <C45,GENERAL,R9,[F24,F34,F44]>, <C2,CSE,R13,[F02]>, <C23,CE,R2,[F35]>, <C44,GENERAL,R11,[F63,F03,F13]>, <C13,ME,R1,[F21]>, <C24,CE,R4,[F31]>, <C43,GENERAL,R8,[F32,F42,F52]>
<H (Normal, 3)>	<C23,CE,R2,[F35]>, <C11,EE,R1,[F15]>, <C24,CE,R4,[F31]>, <C12,EE,R3,[F11]>
<D (Normal, 3)>	<C11,EE,R1,[F15]>, <C12,EE,R3,[F11]>, <C20,CE,R2,[F32]>
<E (Normal, 3)>	<C7,EE,R5,[F11]>, <C20,CE,R2,[F32]>
<C (Normal, 3)>	<C13,ME,R1,[F21]>, <C5,CSE,R3,[F05]>, <C7,EE,R5,[F11]>, <C6,CSE,R7,[F01]>
<P1 (Lab, 3)>	<C16,ME,R9,[F24]>, <C3,CSE,R19,[F03]>

**Fig. 5.13** Sample resultant allocation w.r.t. Slots

Faculty	Classes (Course, Department, Max # of Students, Room, Slots)
F01	<C42,GENERAL,100,R17,[F]>, <C1,CSE,35,R1,R1,[I,J]>, <C6,CSE,45,R7,R7,[C,G]>
F02	<C2,CSE,45,R13,R13,[F,A]>
F03	<C3,CSE,100,R19,[P1]>, <C44,GENERAL,100,R11,[A]>
F04	<C4,CSE,100,R9,[P4]>
F05	<C5,CSE,35,R3,R3,[C,G]>
F11	<C42,GENERAL,100,R17,[F]>, <C7,EE,35,R5,R5,[E,C]>, <C12,EE,45,R3,R3,[H,D]>
F12	<C8,EE,45,R5,R5,[F,A]>
F13	<C44,GENERAL,100,R11,[A]>, <C9,EE,100,R9,[P5]>
F14	<C10,EE,100,R19,[P4]>
F15	<C11,EE,35,R1,R1,[H,D]>

**Fig. 5.14** Sample resultant allocation w.r.t. Faculties

Room (Name, Campus, Lecture Type, Capacity)	Classes (Course, Department, Max # of Students, Faculties, Slots)
<R1 (Ahalia, Normal, 60)>	<C37,HSE,60,[F63][F]>, <C11,EE,35,[F15][H,D]>, <C13,ME,35,[F21][A,C]>, <C1,CSE,35,[F01][I,J]>, <C14,ME,45,[F22][B,M]>
<R2 (Nila, Normal, 100)>	<C34,SME,25,[F55][F]>, <C23,CE,35,[F35][A,H]>, <C40,GENERAL,100,[F]>, <C41,GENERAL,100,[F]>, <C20,CE,45,[F32][D,E]>
<R3 (Ahalia, Normal, 45)>	<C29,GCE,25,[F45][F]>, <C5,CSE,35,[F05][C,G]>, <C18,ME,45,[F21][I,J]>, <C12,EE,45,[F11][H,D]>
<R4 (Nila, Normal, 60)>	<C26,GCE,45,[F42][F]>, <C24,CE,45,[F31][A,H]>
<R5 (Ahalia, Normal, 100)>	<C8,EE,45,[F12][F,A]>, <C7,EE,35,[F11][E,C]>
<R6 (Nila, Normal, 45)>	<C19,CE,35,[F31][F]>
<R7 (Ahalia, Normal, 60)>	<C25,GCE,35,[F41][F]>, <C6,CSE,45,[F01][C,G]>
<R8 (Nila, Normal, 100)>	<C28,GCE,100,[F44][F]>, <C43,GENERAL,100,[F32,F42,F52][A]>

**Fig. 5.15** Sample resultant allocation w.r.t. Rooms

WeekDay	08:00 AM - 08:50 AM	09:00 AM - 09:50 AM	10:00 AM - 12:50 PM	10:30 AM - 11:45 AM	11:00 AM - 11:50 AM	12:00 PM - 12:50 PM	02:00 PM - 04:45 PM	03:30 PM - 04:45 PM	05:00 PM - 05:50 PM	
Monday		C8,A C45,A C2,A C23,A C44,A C13,A C24,A C43,A	C16,P1 C3,P1 C23,H C11,H C24,H C12,H		C11,D C12,D C20,D	C7,E C20,E	C4,P4 C10,P4 C15,P4		C13,C C5,C C7,C C6,C	
Tuesday		C37,F C34,F C29,F C26,F C8,F C19,F C28,F C25,F C39,F C31,F C38,F	C33,F C42,F C27,F C17,F C30,F C36,F C40,F C32,F C41,F C35,F C2,F	C5,G C6,G		C14,M	C9,P5 C1,I C22,P5 C18,I	C1,J C18,J	C8,A C45,A C2,A C23,A C44,A C13,A C24,A C43,A	
Wednesday	C13,C C5,C C7,C C6,C	C8,A C45,A C2,A C23,A C44,A C13,A C24,A C43,A	C14,B		C23,H C11,H C24,H C12,H	C7,E C20,E			C11,D C12,D C20,D	
Thursday		C37,F C34,F C29,F C26,F C8,F C19,F C28,F C25,F C39,F C31,F C38,F	C2,F C33,F C42,F C27,F C17,F C30,F C36,F C40,F C32,F C41,F C35,F	C5,G C6,G		C14,M	C1,J C18,J C21,P6	C1,I C18,I	C14,B	
Friday	C23,H C11,H C24,H C12,H	C13,C C5,C C7,C C6,C	C11,D C12,D C20,D		C14,B	C7,E C20,E			C37,F C34,F C29,F C26,F C8,F C19,F C28,F C25,F C39,F C31,F C38,F	C2,F C33,F C42,F C27,F C17,F C30,F C36,F C40,F C32,F C41,F C35,F

**Fig. 5.16** Final Timetable Obtained

## Observations

The final timetable obtained satisfies all the hard constraints and is well-suited to be used as a timetable for a university. Because no randomization is involved in this approach, it produces better results than the previous approach. We found out that due to checking over all the possible combinations of slots and rooms, the algorithm becomes slow to even initialize the initial population, and all the other steps of crossover and mutation also become slow due to checking multiple combinations. Hence, it takes it a lot of time to even jump from one generation to the other.

Hence, for the small set of courses, it is able to find a solution on the very first generation itself, but for the large set of courses, it is slow to even go to the next generation.

## 5.3 Conclusion

This chapter introduces the genetic algorithm and how we can use it to solve the timetabling automation problem. We discussed two approaches to designing a chromosome for the genetic algorithm, the first using a randomized approach and the second using a greedy and maximal allocation approach. Though based on the observations it works well when there are a lesser number of courses having conflicts between them, and as the number of such types of courses increases, the algorithm converges very slowly. The results show that the timetable obtained satisfies all the hard constraints and is optimized for the soft constraints. The implementation of these approaches can be found in the [GitHub](#) repository. In the next chapter, we will discuss another algorithm, **Simulated Annealing**, for solving the timetable automation problem.

# Chapter 6

## Simulated Annealing

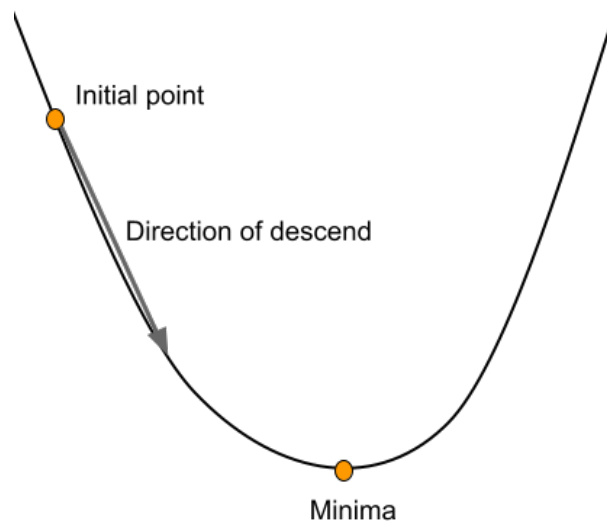
### 6.1 What is Simulated Annealing?

Simulated annealing is an approximation and probabilistic algorithm to find out the global minima/maxima for a given function. To understand what simulated annealing is, let's first understand a more straightforward algorithm, the hill climbing algorithm.

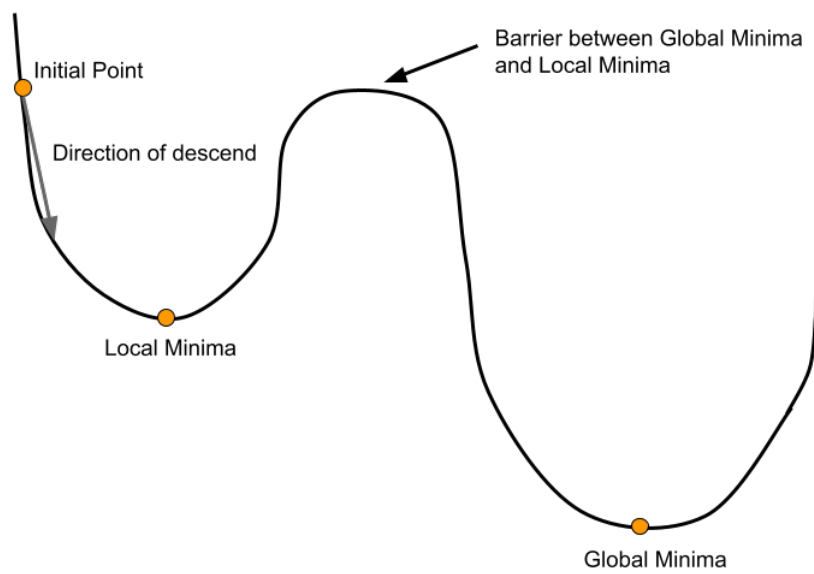
#### 6.1.1 Hill Climbing Algorithm

The hill climbing algorithm is used to find optimums for a function, which mostly has a discrete space. The main idea is to pick a random state as the initial current state. Then we change the current state to one of the neighbors (+1 neighbor and -1 neighbor) until both the neighbors are worse than the current state. This final state is chosen as the output for the hill climbing algorithm. Figure 6.1 below shows how the hill-climbing algorithm would work for any given function.

The problem with the hill climbing algorithm is that it often gets stuck at the local minima. This happens because when the algorithm reaches a minima, both its neighbors have a cost greater than the cost of the function at the current point. Hence the algorithm does not proceed to find the global minima. This can be seen in the function shown in Figure 6.2 shown below.



**Fig. 6.1** Successful Hill Climbing



**Fig. 6.2** Function for which Hill Climbing will fail

Hence, the hill climbing algorithm is not directly applicable if we want to find the global minima/maxima of a function.

### 6.1.2 Fixing the Hill Climbing algorithm

Simulated annealing is nothing but a way to modify the above-discussed algorithm in such a way that the probability for the algorithm to find a global minima is increased (Since finding the counterpart of minima, i.e., maxima is the same as finding minima, I will continue the entire discussion for finding the minima itself. We will also see later that to solve the timetabling problem, we would aim to minimize a function and hence talking about minima is more relevant in our case)

The way simulated annealing works is that it allows us to jump to a worse neighbor state with some probability. Thus it allows us to move towards the global minimum even if we have reached local minima. The probability with which the neighbor state is chosen depends on the temperature of the algorithm currently. If the temperature of the algorithm is high, then we choose to go to a worse state with more probability, and vice versa. As the algorithm descends, we start with a higher temperature and then move on to lower temperatures gradually. The algorithm finishes when we exhaust a pre-set time limit.

Thus, the major components of simulated annealing, given a function, are -

- **Finding Neighbors of a State** - The neighbors of a state are found by finding the next state by transforming a given current state as input. An essential property of the neighbor function should be that the new state should be fairly close to the old state. Thus the change in the answer based on transitioning to the new state should be small. This ensures that we do not skip the minima while transitioning to a new state.
- **Temperature Function** - The temperature function determines the willingness of the algorithm to move to a worse state from a given state, as discussed above. If the temperature of the algorithm is currently zero, then the simulated annealing reduces to



the greedy hill-climbing algorithm and tends to find the local minima. Thus the temperature of the algorithm must start from a big value and gradually move toward zero (hence the temperature function must be decreasing function). A commonly accepted temperature function is the geometric function  $\alpha \cdot T$ . We will be finding/choosing the best-suited temperature function and  $\alpha$  for the timetabling problem eventually. Other common temperature reduction functions are -

1.  $T = \alpha \cdot T$

2.  $T = T - \alpha$

3.  $T = \frac{T}{1+\alpha \cdot T}$

- **Acceptance Probability Function** - The acceptance probability function gives the probability with the algorithm can move to a worse solution based on the temperature of the current algorithm. The probability can be denoted by  $P(oldState, newState, temp)$  Below is a common acceptance probability function used in simulated annealing

$$P(oldState, newState, temp) = \begin{cases} 1, & \text{if } newState < oldState \\ e^{\frac{oldState - newState}{temp}}, & \text{otherwise} \end{cases} \quad (6.1)$$

*Note that the above probability is always less than or equal to one since  $oldState - newState \leq 0$  in the second condition.*

Using all the above functions, the pseudo-code of simulated annealing can be written down as -

---

**Algorithm 1:** Simulated Annealing Pseudo Code

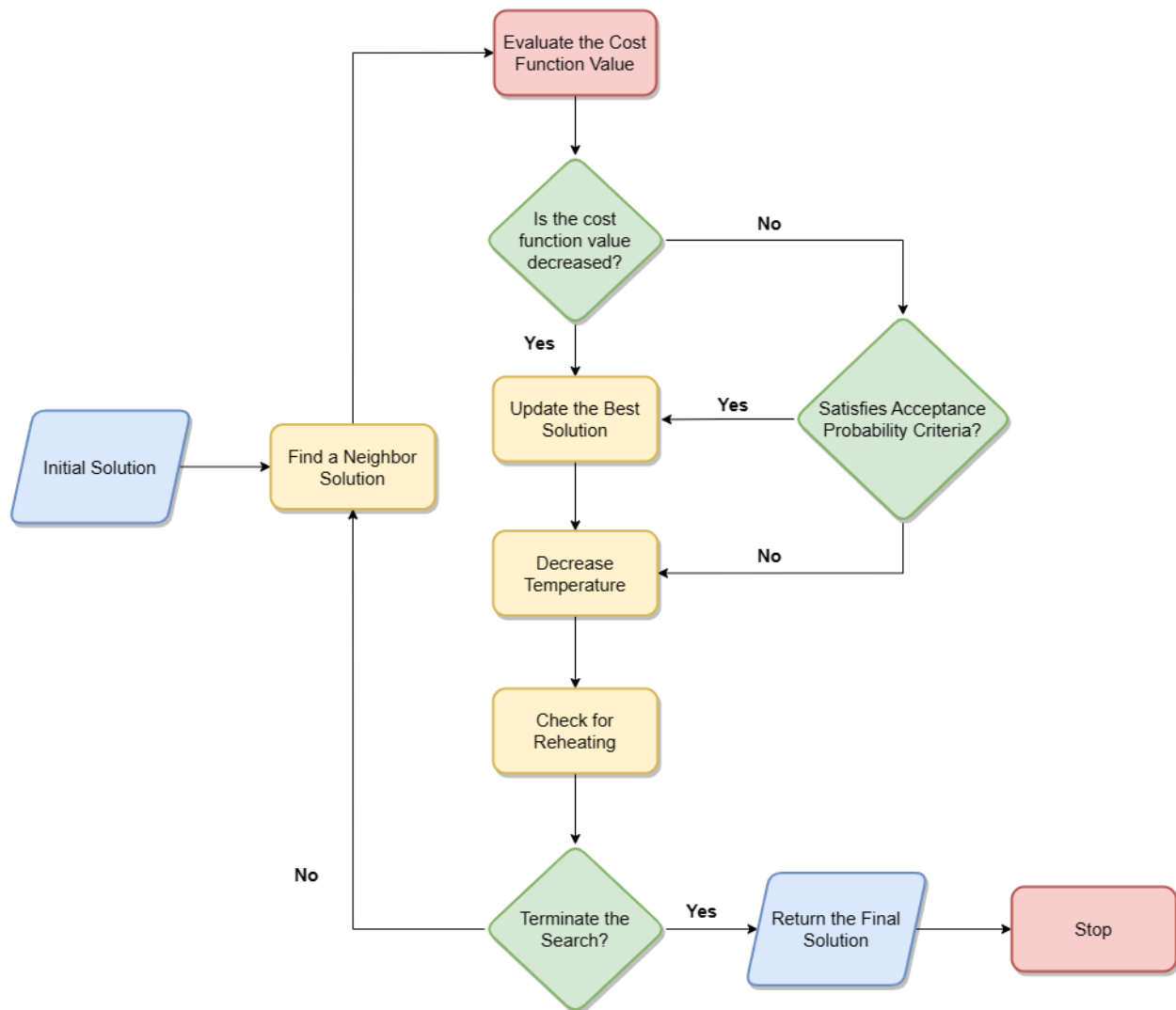
---

**Let:**  $s \leftarrow s_0$  be initial solution using constraint solver;  
**while** *stopping condition is false* **do**  
     $t \leftarrow \text{Temperature}(t);$   
     $s' \leftarrow \text{Neighbor}(s);$   
     $\delta \leftarrow F(s') - F(s);$     // where F is the cost function  
    **if**  $\delta \leq 0 \parallel P(\text{value}(s), \text{value}(next), t) \geq \text{random}(0, 1)$  **then**  
         $s \leftarrow s';$   
    **end**  
**end**

---

The **stopping condition** in the while loop can be anything depending on how much we want to refine the solution. In our case, we will break the loop on the basis of a set time limit.

### 6.1.3 Simulated Annealing Workflow



**Fig. 6.3** Simulated Annealing Workflow

## 6.2 Using Simulated Annealing (SA) in Timetabling Problem

### 6.2.1 Why can we use SA in the timetabling problem?

The major challenge as we have discussed in the timetabling problem is the vast amount of constraints that need to be satisfied. The hard constraints should always be satisfied whereas the soft constraints should be tried to satisfy as much as possible. Thus we can associate a cost with a presented solution for the timetabling problem. The cost will be more if more and more constraints are unsatisfied and will be less if most of the constraints are satisfied.

Thus we can see that on defining such a cost function for the timetabling problem, the problem of finding the best-fit timetable reduces to finding the minimum value for the specified cost function. Hence, as in the classical simulated annealing, we will break the timetabling problem into the following subsections.

### 6.2.2 Initializing the start state

Choosing the start state for simulated annealing plays a critical role in its performance. The start state of the simulated annealing should be chosen in a way that all or most of the hard constraints are satisfied. To do this, we will have to use any of the **constraint solver** or **SAT solver** available. The soft constraints should be satisfied as much as possible as well. One method to generate the start state for simulated annealing is that we take the final output from the **genetic algorithm** discussed in the previous chapter after many generations have been created.

Unfortunately, it turns out that the performance of the genetic algorithm is not enough for simulated annealing to work properly. Hence the best way is to use a constraint solver for the initialization, but finding a constraint solver which can work well with the exact requirements of our institute is difficult. Hence we decided to implement one of the greedy algorithms used in the ITC-2007 (International Timetabling Competition). The algorithm is discussed in detail in the next chapter.

### 6.2.3 Neighborhood searching

A good neighborhood searching function is needed for a probabilistic search algorithm like simulated annealing to work. Below are the three neighborhood functions which we implemented.

Some conditions that should be surely satisfied while finding neighbors are

- The credit requirement for any of the courses should not be violated in the new timetable since this condition is not included as a hard constraint
- The new timetable should not have a lab course in a non-lab slot

We use the following three ways to find the neighbor of the current timetable

- **Simple Searching Neighborhood** - In this method, we choose a random course and assign it to a random pair of slots and rooms.
- **Swapping Neighborhoods** - In this method, we choose two random sets of slot, room, course -  $\langle c_{i1}, s_{j1}, r_{k1} \rangle$  and  $\langle c_{i2}, s_{j2}, r_{k2} \rangle$ . Then we swap the courses in the chosen sets and make sure that the above conditions are satisfied. If they are not satisfied then we try to choose a new neighbor.
- **Simple searching and Swapping Neighborhoods** - Two lectures and two slots are randomly picked simultaneously and the lectures are allocated the slots and then swapped in the next iteration

Although the first swapping neighborhood method of choosing neighbors is enough to run SA, the most efficient way to do so is to run all the neighborhood functions one by one. This helps in achieving the states of the timetable which cannot be reached by just swapping and hence is faster than the simulated annealing with just one neighborhood function.

#### 6.2.4 Defining the cost function

We will now try to formulate the cost function  $F$  which we need to minimize using simulated annealing. Let's first calculate the cost associated with each of the constraints individually. The cost function associated with some of the major constraints is listed below.

$C1$  : No faculty should have more than one class at a time. Hence the cost associated with this constraint should be -

$$F_{C1} = \omega_1 \sum_{n=1}^{n-1} \sum_{j=i+1}^n T_{i,j}$$

where  $n$  is the total number of lectures and  $T_{i,j}$  is the total number of faculty who have to give lectures  $i$  and  $j$  at the same time.

$C2$  : No student group should have more than one class at a time. Hence the cost associated with this constraint should be -

$$F_{C2} = \omega_2 \sum_{n=1}^{n-1} \sum_{j=i+1}^n S_{i,j}$$

where  $n$  is the total number of lectures and  $S_{i,j}$  is the total number of students who have to attend lectures  $i$  and  $j$  at the same time. This can be adjusted to a soft constraint when the lecture conflict is an elective and a hard constraint when both lectures are compulsory for the student.

$C3$  : No room should have more than one lecture running at a time. Hence the cost associated with this constraint should be -

$$F_{C3} = \omega_3 \sum_{n=1}^{n-1} \sum_{j=i+1}^n R_{i,j}$$

where  $n$  is the total number of lectures and  $R_{i,j}$  is the total number of rooms that are associated with lectures  $i$  and  $j$  at the same time.

A similar cost equation can be associated with each of the constraints, which is nothing but the total number of conflicts for each constraint multiplied by a constant  $\omega_i$ . The value of the constant is chosen to be very large (treated as infinity) in the case of a hard constraint and is smaller for soft constraints.

The final cost function  $F$  will thus be finally given by -

$$F = F_{C1} + F_{C2} + F_{C3} + \dots \quad (6.2)$$

Implementation in code for the cost function is quite straightforward. We just iterate through all the courses allocated in a particular slot, room, etc, and check for conflicts between faculty or students. The cost for a hard conflict is set as 1000000 whereas for a soft conflict, it is set as 100.

### 6.2.5 Choosing the temperature and acceptance probability function

The temperature and the acceptance probability function used in the implementation initially would be the ones mentioned above since they are the most widely used functions in simulated annealing.

Below are the exact versions of temperature and acceptance probability functions which we use along with the values of the constant.

- Temperature Function -

$$t = t \cdot \left(1 - \frac{\ln t - \ln FT}{NMOVE}\right)$$

where  $FT$  is the final temperature of the algorithm.  $NMOVE$  is a parameter that is defined to slow down temperature reduction. Increasing the value of  $NMOVE$  slows

down the speed at which temperature reduces. The initial temperature is chosen to be 80000 and the final temperature as 0.00000001. *NMOVE* is chosen to be 500. These values best ensure that the temperature of the timetable is such that most of the states are covered and enough time is spent on each temperature.

- Acceptance Probability - The acceptance probability is chosen to be the same as equation 6.1

### 6.2.6 Reheating

As the temperature of the algorithm decreases, there are chances it gets stuck at local minima and hence does not find the more optimal global solution. Hence to avoid this, we keep a check on the current cost of the timetable over the iterations. If the cost does not change for a long time, then we set the temperature back to the initial (or another higher) temperature, thus allowing the algorithm to accept worse states than the current ones again. In our case, we found that reheating works best when there have been 25000 iterations without any change in cost, at which we set the current temperature back to the initial temperature (equal to 80000).

## 6.3 Results and Observations

No.of courses	Hard Conflicts Genetic Algorithm	Hard Conflicts Simulated Annealing	Soft Conflicts Simulated Annealing
37 (random data)	0	0	0
45 (random data)	13	0	0
57 (random data)	43	0	2
69 (random data)	90	1	3
97 (random data)	141	3	16
94 (real data)	222	11	62

**Table 6.1** Comparison between Genetic Algorithm and Simulated Annealing used on timetable initialized by Genetic Algorithm

As we can see in the above results that the simulated annealing drastically reduces



the conflicts of the genetic algorithm. But the starting conflicts being this huge is a big disadvantage for simulated annealing. Hence in the next chapter, we will be seeing how greedy performs with the simulated algorithm as compared to the genetic algorithm.

Note that our implementation considers a credit-based system for the students. Hence the actual slot system being used in the current semester is infeasible, thereby producing more hard conflict values.

## 6.4 Conclusion

This chapter introduces the simulated annealing algorithm and how we can use it to solve the timetabling automation problem. We saw how we modify the hill climbing algorithm into the simulated annealing algorithm. We also saw various components of the simulated annealing algorithm like the temperature function, the acceptance probability function, the cost function, the neighbor function, and various ways of implementing these. In the next chapter, we will discuss a greedy method for finding a better initial solution as an input for the Simulated Annealing problem and observe its performance.

# Chapter 7

## Greedy Initialization for Simulated Annealing

### 7.1 Why do we need a greedy initialization?

Simulated annealing is a probabilistic method for finding the best solution. Since the timetable changes on the swap of slot or room for any two courses, the number of states the simulated annealing algorithm can visit is huge. Hence although simulated annealing can initially reduce the number of conflicts very fast, we observed that for a real timetable, after some point in time (when there are about 40 hard conflicts left), the algorithm slows down drastically and the further decrease in a number of slots is rare, even with reheating. This method is hence very efficient in reducing the number of soft constraints (as they are always large in number), but since the number of hard constraints should be reduced as much as possible, we have to aim to start with a very few or none hard conflicts whatsoever at the beginning of simulated annealing itself. Hence we try to use a greedy allocation for the initialization of the timetable. The algorithm which we use here is an adaptation of the greedy algorithm presented in [1], which was proposed as the third track of the Second International Timetabling Competition (ITC-2007).

## 7.2 Problem Formulation

To run the greedy algorithm, we first need to formulate the problem for ourselves in the best way possible. Our problem consists of  $n$  courses,  $C = \{c_1, c_2, \dots, c_n\}$ . These courses should be allocated to  $s$  slots  $T = \{t_1, t_2, \dots, t_s\}$ . The slots here are the named slots in our timetable according to the given slot system. Each slot corresponds to a certain number of hours given to it. Each course will also be allocated rooms from the set of  $m$  rooms,  $R = \{r_1, r_2, \dots, r_m\}$ .

We choose a simple representation for our timetable solution. The timetable will be represented by an  $s \times m$  matrix  $M$ , where  $M_{ij}$  is either  $-1$  or the course id for the course which has been allocated to  $i_{th}$  slot from  $T$  and  $j_{th}$  room from  $R$ . This representation already ensures that the hard constraint of

“No two courses should be allocated a same room at any period of time”

is always satisfied.

## 7.3 Algorithm

To explain the algorithm, we first need to define some notions used in the algorithm.

A partially filled timetable is denoted by  $\widetilde{M}$ . This indicates the initial empty matrix  $M$  filled by courses after some time in the algorithm. The below notions are defined on this partially filled timetable  $\widetilde{M}$

- $con_{ij}$ : Denotes conflict between course  $i$  and  $j$ . Equal to 1 if there is a conflict otherwise zero.
- $l_i$ : Number of required lectures / credits for course  $c_i$
- $apd_i(\widetilde{M})$ : This denotes the available number of slots in  $\widetilde{M}$  for the course  $c_i$ . Hence it is the number of slots that do not have courses that conflict with  $c_i$  in terms of faculty

or students.

$$apd_i(\widetilde{M}) = \sum_{j=1}^s \phi_{i,j}(\widetilde{M})$$

where

$$\phi_{i,j}(\widetilde{M}) = \begin{cases} 0, & \exists m_{j,k} = c_u \in \widetilde{M}, con_{iu} = 1 \\ 1, & otherwise \end{cases}$$

- $aps_i(\widetilde{M})$ : Number of the available slot-room pairs (or positions in timetable) for course  $c_i$  in  $\widetilde{M}$ .

$$aps_i(\widetilde{M}) = \sum_{j=1, \dots, s, k=1, \dots, m} \chi\{\phi_{i,j}(\widetilde{M}) = 1 \ \& \ m_{jk} = -1\}$$

- $nl_i(\widetilde{M})$ : Number of unassigned lectures of course  $c_i$  in  $\widetilde{M}$

$$nl_i(\widetilde{M}) = \max(l_i - \sum_{j=1, \dots, s, k=1, \dots, m} \omega_{i,j,k}(\widetilde{M}), 0)$$

where

$$\omega_{i,j,k}(\widetilde{M}) = \begin{cases} 0, & x_{j,k} \neq c_i \\ sw_j, & otherwise \end{cases}$$

Here  $sw_j$  is the slot weight of the  $j_{th}$  slot, which is nothing but the number of credits of the slot.

- $uac_{i,j}(\widetilde{M})$ : The number of courses which become unavailable at slot  $s_j$  when course  $c_i$

is assigned to slot  $s_j$ .

$$uac_{i,j}(\widetilde{M}) = \sum_{u=1}^n \delta_{i,j,u}(\widetilde{M}) \cdot nl_u(\widetilde{M})$$

where

$$\delta_{i,j,u}(\widetilde{M}) = \chi\{con_{iu} = 1 \ \& \ (\forall x_{jk} = c_v \in \widetilde{M}, con_{uv} = 0)\}$$

### 7.3.1 Course Selection Heuristic

The course heuristic determines which course is needed to be picked for a given partially filled timetable in our greedy algorithm. The heuristic goes as follows

1. Choose a course with the smallest value of  $apd_i(\widetilde{M})/\sqrt{nl_i(\widetilde{M})}$
2. In case of ties after step 1, choose a course with the smallest value of  $aps_i(\widetilde{M}) \cdot \sqrt{nl_i(\widetilde{M})}$
3. In case of ties after step 2, any course can be chosen

The idea behind choosing this heuristic is simple.  $apd$  and  $aps$  tell about the number of slots and positions available for the given course.  $nl$  is the number of credits yet to be allocated for the given course. Since we would like to first allocate a slot to the course which has the least number of available slots left for it and which has most of the lectures still unallocated, the heuristic follows.

After we have chosen a lecture, we now want to choose a slot-room pair for the lecture to assign. For this, we use the following slot-room selection heuristic.

### 7.3.2 Slot-Room Selection Heuristic

If a course  $c_i$  is chosen after using the Course Selection Heuristic, then the slot  $s_{j'}$  and room  $r_{k'}$  is chosen such that

$$f(j', k') = \min_{j,k} \{f(j, k)\}, \quad f(j, k) = k1 \cdot uac_{i,j}(\widetilde{M}) + k1 \cdot \gamma + k2 \cdot \Delta h$$

$\Delta h$  is the number of soft conflict changes on insertion of  $c_i$  in slot  $s_j$  and room  $r_k$  and  $\gamma$  is equal to the number of hard conflicts which incur with other courses in slot  $s_j$  on insertion of  $c_i$ . The logic of this heuristic is also straightforward, we insert the course which has the least effect on other courses and is also affected least in this type of insertion. The values of  $k1 = 1$  and  $k2 = 0.1$  are seen to perform best for our timetable. Although there is no proof that this greed works, it worked for all the test cases in ITC-2007 (International Timetabling Competition). Even for our timetable, this heuristic seems to provide the best results among other tested greedy approaches. The pseudo-code of the greedy algorithm combining the above heuristics is given in Algorithm 2 below.

---

**Algorithm 2:** Greedy initialization for SA Pseudo Code

---

**Input:** Set of courses, slots, and rooms to be incorporated in the timetable;

**Output:** A near-feasible timetable;

Set  $\widetilde{M} = \phi$  and initialize the set  $LC$  of unassigned lectures;

**while**  $LC$  is not empty **do**

Choose an unassigned lecture  $c_i$  based on **Course Selection Heuristic**;

Choose a slot-room pair  $(s_j, r_k)$  based on **Slot-Room Selection Heuristic**;

Insert  $c_i$  at position  $(j, k)$  in the timetable;

Reduce credit requirement of  $c_i$  by credit weightage of slot  $s_j$ ;

Remove  $c_i$  from  $LC$  if it has met its credit requirements;

**end**

---

## 7.4 Results and Observations

No.of courses	Hard Conflicts Greedy Algorithm	Soft Conflicts Greedy Algorithm	Hard Conflicts SA	Soft Conflicts SA
37 (random data)	0	0	0	0
45 (random data)	0	0	0	0
57 (random data)	0	0	0	0
69 (random data)	0	0	0	0
97 (random data)	6	33	0	39
94 (real data)	18	34	8	33

**Table 7.1** Comparison between Greedy Algorithm and Simulated Annealing used on timetable initialized by Greedy Algorithm

The data used here is the same and corresponds to that of the previous chapter for easier comparisons.

We can see here that the initial solution given by the greedy algorithm produces far better results than the genetic algorithm for larger data sets. Hence the performance of simulated annealing also improves since it gets to start from a state with fewer conflicts.

But as we discussed before, our implementation considers a credit-based system for the students, because of which the actual slot system being used in the current semester is infeasible, thereby producing more hard conflict values.

## 7.5 Conclusion

In this chapter, we discussed a greedy initialization technique for simulated annealing. As the results show, the performance of this initialization is much better than the genetic algorithm and hence it defaults as our solution for the timetabling problem. In the next chapter, we will discuss the web application created for the project.

# Chapter 8

## Web Application

### 8.1 Understanding the Repository Workspace

#### 8.1.1 Polyrepo vs Monorepo

A **polyrepo** is the standard way of developing web applications in which we create a repository for each different application in the project. Each repository has its separate build pipeline and artifacts. The problems with the above approach are:

- It becomes difficult to share the same pieces of code between different applications
- It leads to a signification repetition of code across different applications
- The changes across a repository are very costly to shared library repositories
- It leads to inconsistent tooling and configuration across different projects

A **monorepo** is a single repository that contains many different projects with a well-defined relationship between them. It overcomes the problems associated with the polyrepo as:

- There is no overhead of creating new applications, as all the configuration and tooling is already done



- There are atomic commits across different projects due to which we get a straight-line commit graph
- It maintains one version of everything in a single repository
- It provides the developer mobility across projects
- It makes it easier to share different pieces of code across various projects
- It provides consistent tooling across various projects

### Why migrating from Create-React-App?

The **CRA (Create-React-App) workspace** can not be used to share the **TypeScript** classes and interfaces across the backend and the frontend application. Hence, we need to find an alternative that provides the tooling and configuration required for building the application following the best practices for an optimized production application.

#### 8.1.2 Understanding Nx (by Nrwl)

**Nx** is a monorepo tool that provides various features to maintain a monorepo easily. It provides local and distributed cache computing, local and distributed task execution, detecting affected libraries and applications, workspace analysis and dependency graph visualization, code sharing, consistent tooling, and project constraints.

### Creating an Nx Workspace

```
npx create-nx-workspace@latest
```

- Follow the interactive instructions to set up the app. Nx will set end-to-end (E2E) tests automatically using **Cypress**
- The **apps** folder will contain all our applications

- The **libs** folder will contain all our shared libraries that can be used across different projects for code sharing
  - **80% of the code logic should reside in libraries and 20% in apps**
  - We have typically four kinds of libraries: **Feature** libraries for business use-cases, **UI** libraries for presentational components, **Data-access** libraries for interacting with external data services and **Utility** libraries for common utilities that are shared by many projects
- The configuration files for each project can be found in the **project.json** file present in each project folder

## Serving an Nx Application

To build the application and start serving the application on a **localhost** port using the configured bundler, run

```
npx nx serve app-name
```

## Building an Nx Application for Production

```
npx nx build app-name --prod
```

## Running Unit Tests for an Nx Application

```
npx nx test app-name
```

## Running Linter for an Nx Application

```
npx nx lint app-name
```

## Running Formatter for an Nx Application

```
npx nx format:check
```

```
npx nx format:write
```

## 8.2 Understanding the Backend

### 8.2.1 Tech Stack

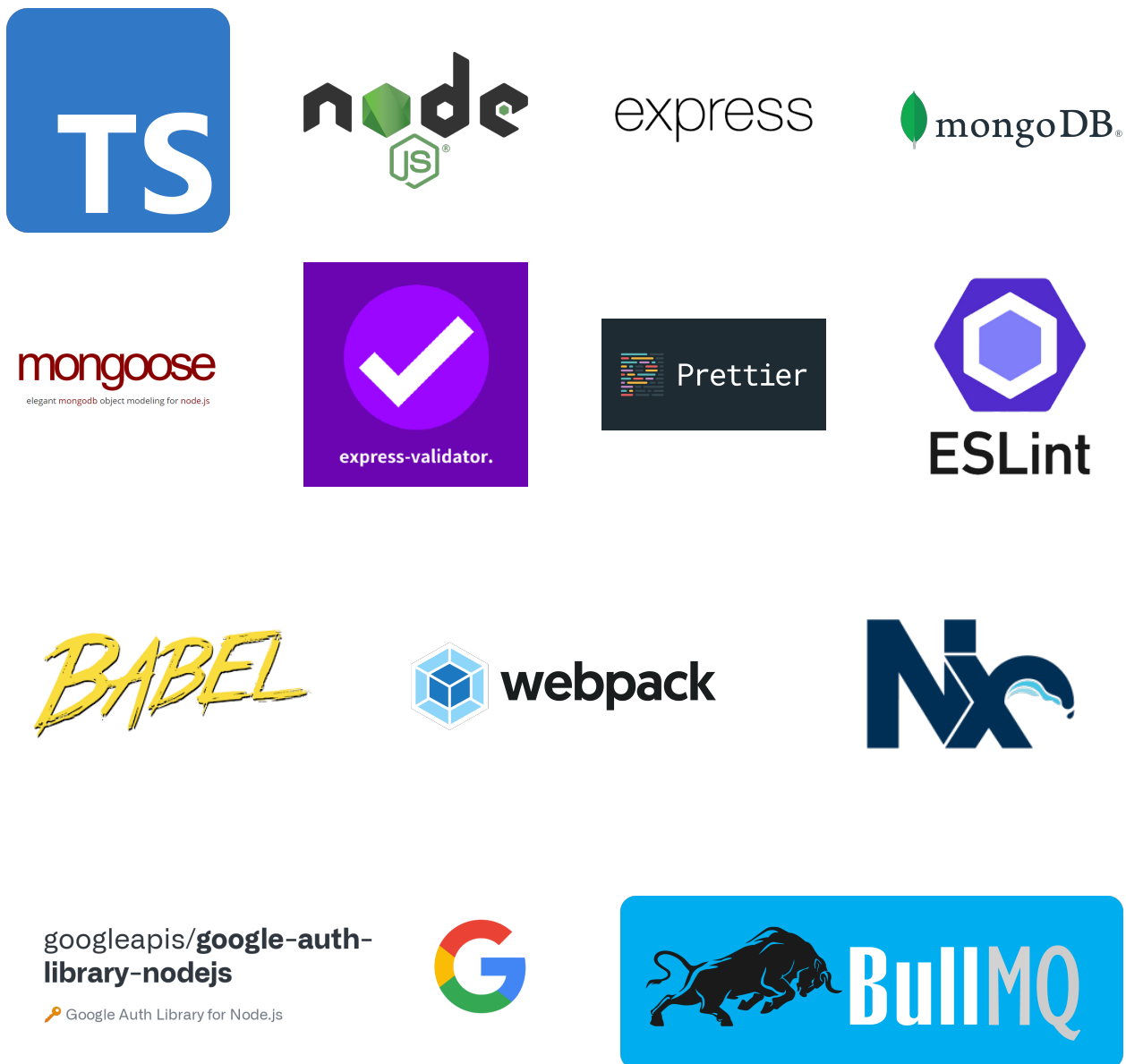
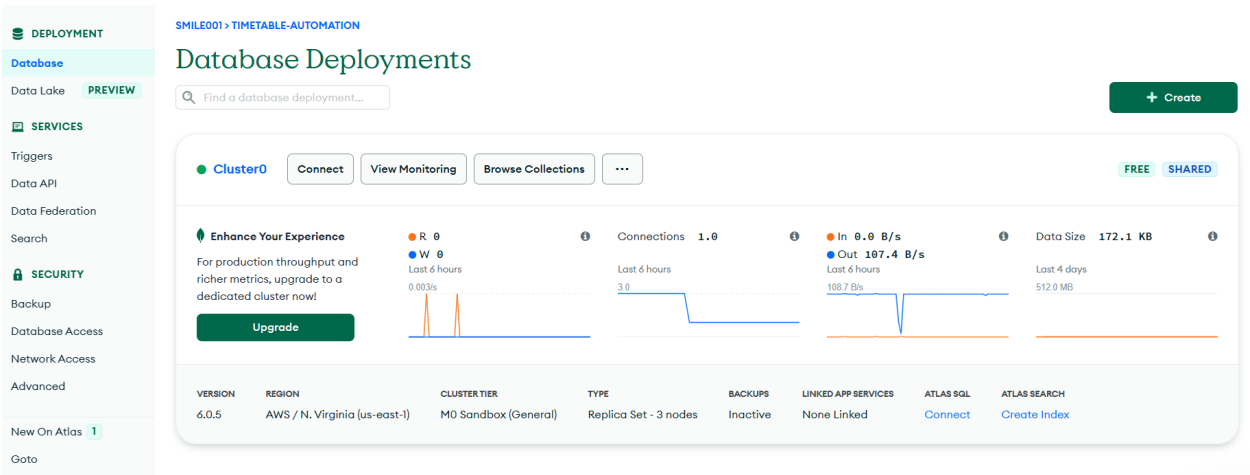


Fig. 8.1 Backend Tech Stack

## 8.2.2 Configuring MongoDB Atlas

### Create a MongoDB Atlas Free Cluster



### Create a User with Read and Write Privileges

Database Users Custom Roles

[+ ADD NEW DATABASE USER](#)

User Name	Authentication Method	MongoDB Roles	Resources	Actions
mayank	SCRAM	readWriteAnyDatabase@admin	All Resources	<a href="#">EDIT</a> <a href="#">DELETE</a>

### Set the Allowed List of IP Addresses

Network Access

IP Access List Peering Private Endpoint

[+ ADD IP ADDRESS](#)

You will only be able to connect to your cluster from the following list of IP Addresses:

IP Address	Comment	Status	Actions
157.46.209.45/32	My IP Address	Active	<a href="#">EDIT</a> <a href="#">DELETE</a>
0.0.0.0/0 (includes your current IP address)		Active	<a href="#">EDIT</a> <a href="#">DELETE</a>

## Get the Connection URI

### Connect to Cluster0



### Connecting with MongoDB Driver

#### 1. Select your driver and version

We recommend installing and using the latest driver version.

Driver	Version
Node.js ▼	4.1 or later ▼

#### 2. Install your driver

Run the following on the command line

```
npm install mongodb
```

[View MongoDB Node.js Driver installation instructions.](#)

#### 3. Add your connection string into your application code

☐ View full code sample

```
mongodb+srv://mayank:<password>@cluster0.tnvtifb.mongodb.net/?  
retryWrites=true&w=majority
```

Replace **<password>** with the password for the **mayank** user. Ensure any option params are [URL encoded](#).

## Create and Manage Collections via Server app using Mongoose

The screenshot shows the MongoDB Atlas web interface. On the left sidebar, under 'DATABASES: 1 COLLECTIONS: 4', the 'timetable-automation' database is expanded, showing a list of collections: 'courses', 'rooms', 'slots', and 'users'. The 'courses' collection is selected. The main panel displays the 'timetable-automation.courses' collection details, including storage size (62KB), logical data size (31.92KB), total documents (109), and indexes total size (62KB). Below this, there are tabs for 'Find', 'Indexes', 'Schema Anti-Patterns', 'Aggregation', and 'Search Indexes'. The 'Find' tab is active, showing a query filter input field with the placeholder 'Type a query: { field: 'value' }'. Below the filter, it says 'QUERY RESULTS: 1-20 OF MANY'. A sample document is displayed in a light blue box with the following fields: `_id` (ObjectId), `code` (PHY1), `name` (Physics), `credits` (Array), `courseType` (COMMON), `lectureType` (Normal), `maxNumberOfStudents` (60), `faculties` (Array), `department` (GENERAL), `createdAt` (2023-05-11T02:21:17.972+00:00), and `updatedAt` (2023-05-11T02:28:53.433+00:00).

### 8.2.3 Configuring Google OAuth Consent Screen and Creating Web Client ID

#### Set up Google OAuth Consent Screen on Google Cloud Console

The screenshot shows the Google Cloud Console interface. At the top, there's a navigation bar with the Google Cloud logo, a dropdown menu showing 'timetable-automation', and a search bar. Below the navigation bar, the left sidebar shows the 'APIs and services' section, with 'OAuth consent screen' selected. The main panel is titled 'Edit app registration' and shows a progress bar with four steps: 1. OAuth consent screen (active), 2. Scopes, 3. Test users, and 4. Summary. Under the 'App information' section, there's a description: 'This shows in the consent screen, and helps end users know who you are and contact you'. Below this, there are two input fields: 'App name \*' with the value 'Timetable Generator' and 'User support email \*' with the value 'singlamayank001@gmail.com'. The 'App name' field has a hint: 'The name of the app asking for consent'. The 'User support email' field has a hint: 'For users to contact you with questions about their consent'.

Add profile info in the scopes to express the required permissions to the user

**APIs and services**

- Enabled APIs and services
- Library
- Credentials
- OAuth consent screen**
- Page usage agreements

**Edit app registration**

1 OAuth consent screen — **2 Scopes** — 3 Test users — 4 Summary

Scopes express the permissions that you request users to authorise for your app and allow your project to access specific types of private user data from their Google Account. [Learn more](#)

[ADD OR REMOVE SCOPES](#)

**Your non-sensitive scopes**

API	Scope	User-facing description
..	./auth/userinfo.email	See your primary Google Account email address
..	./auth/userinfo.profile	See your personal info, including any personal info you've made publicly available

Create Credential for the Web Application to get the Client ID and Client Secret

**APIs and services**

- Enabled APIs and services
- Library
- Credentials**
- OAuth consent screen
- Page usage agreements

**Client ID for Web application** [DELETE](#)

**Name \***  
timetable-automation

The name of your OAuth 2.0 client. This name is only used to identify the client in the console and will not be shown to end users.

**The domains of the URIs you add below will be automatically added to your OAuth consent screen as authorised domains.**

**Authorised JavaScript origins**

For use with requests from a browser

**URIs 1 \***  
http://localhost

**URIs 2 \***  
http://localhost:4200

**URIs 3 \***  
http://localhost:3000

[+ ADD URI](#)

Add the required Authorized JavaScript origins. Save the **Google Client ID** and **Google Client Secret** in `.env` environment variable files and use them within our apps.

## 8.2.4 REST API Endpoints

### Public Endpoints

They don't require any **Authorization Header** to be sent with the request.

Endpoint	Request (JSON)	Response (JSON)	Description
<b>POST</b> /api/auth/createAdmin	{ email: string; name: string }	{ admin: object; msg: string }	Create a new application admin with the given email and name.
<b>POST</b> /api/auth/signIn	{ credential: string }	{ msg: string; email: string; name: string; image: string; roles: UserRoles[]; token: string; tokenExpiration: number }	Signs the user in with the Google credential received and gets the user profile info from the Google servers and sends back the token used for accessing the private routes.


### Private Endpoints

They require the **authentication token** to be sent as **Authorization Header** with every request.

<b>GET</b> /api/algorithm/timetableData		{ msg: string; algorithmStatus: string; classes: Class[]; data: Data }	Returns the algorithm status and the algorithm results from the database once the algorithm is finished. <b>Authorized Users:</b> [Admin, Coordinator, Guest]
<b>POST</b> /api/algorithm/generateTime table	{ algorithmId: string }	{ msg: string; algorithmStatus: string; classes: Class[]; data: Data }	Starts execution of the algorithm in the background and return the status as PENDING. <b>Authorized Users:</b> [Admin, Coordinator]

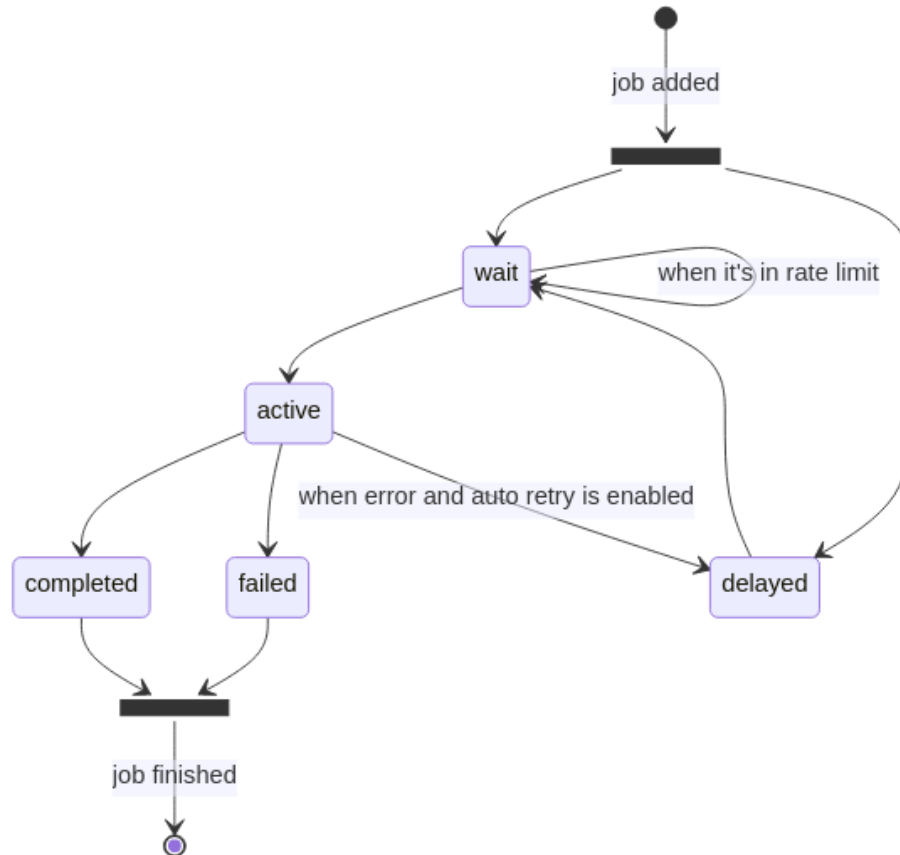


Endpoint	Request (JSON)	Response (JSON)	Description
<b>POST</b> /api/admin/createUser	<pre>{ email: string;   name: string;   roles: UserRole[] }</pre>	<pre>{ newUser: object;   msg: string }</pre>	Create a new application user in the database. <b>Authorized Users:</b> [Admin]
<b>DELETE</b> /api/admin/deleteUser	<pre>{ email: string }</pre>	<pre>{ msg: string }</pre>	Deletes an application user from the database. <b>Authorized Users:</b> [Admin]
<b>GET</b> /api/courses/get		<pre>{courses: Course[];   msg: string }</pre>	Returns the list of courses uploaded on the database. <b>Authorized Users:</b> [Admin, Coordinator]
<b>POST</b> /api/courses/upload	<pre>{ courses:   Course[] }</pre>	<pre>{ msg: string }</pre>	Replaces the list of old courses in the database with the new list of courses. <b>Authorized Users:</b> [Admin, Coordinator]

<b>GET</b> /api/rooms/get		{ rooms: Room[]; msg: string }	Returns the list of rooms uploaded on the database. <b>Authorized Users:</b> [Admin, Coordinator]
<b>POST</b> /api/rooms/upload	{ rooms: Room[] }	{ msg: string }	Replaces the list of old rooms in the database with the new list of rooms. <b>Authorized Users:</b> [Admin, Coordinator]
<b>GET</b> /api/slots/get		{ slots: Slot[]; msg: string }	Returns the list of slots uploaded on the database. <b>Authorized Users:</b> [Admin, Coordinator]
<b>POST</b> /api/slots/upload	{ slots: Slot[] }	{ msg: string }  	Replaces the list of old slots in the database with the new list of slots. <b>Authorized Users:</b> [Admin, Coordinator]

### 8.2.5 Understanding the need for BullMQ

BullMQ is a library that provides a fast queue system for scheduling jobs in a micro-services architecture. It is built on top of **Redis** and is highly performant for scheduling CPU-intensive tasks.



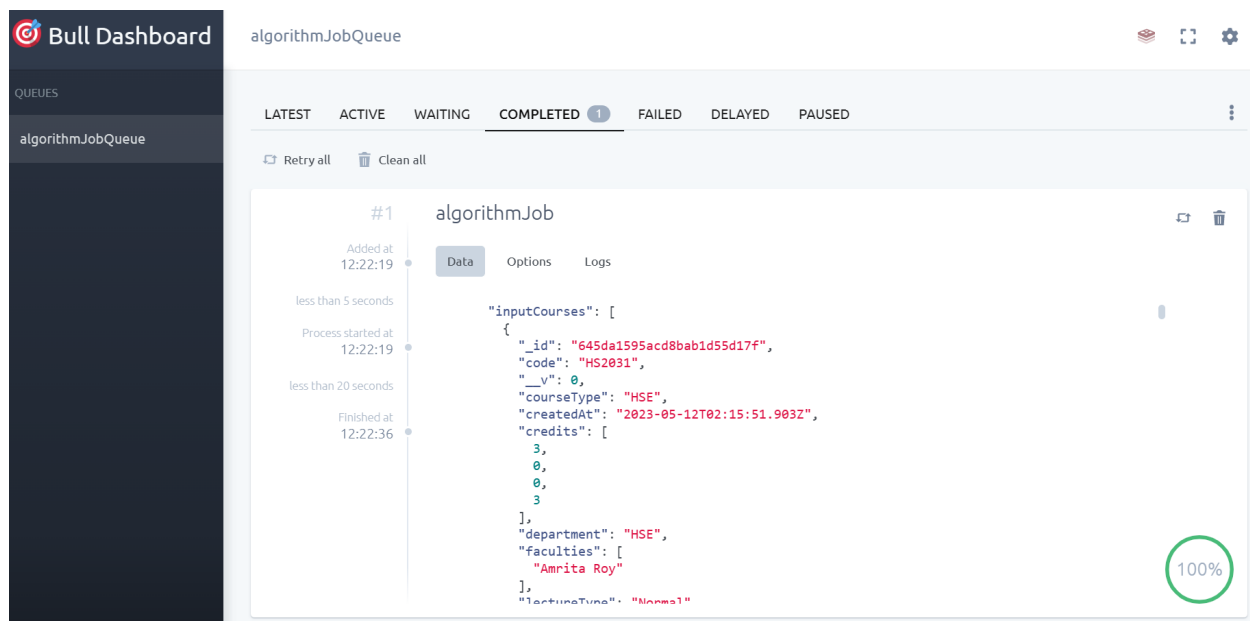
**Fig. 8.2** Lifecycle of a job in a Queue [\[Source\]](#)

Since the **timetable generation algorithm is quite CPU-intensive**, it makes sense to run the algorithm on a separate thread instead of the main thread, so that the event loop of the JavaScript is not blocked. This helps the server to be responsive at all times and it is able to serve other requests simultaneously from the frontend.

For this, we will maintain the algorithm status in the Redis and mark it as PENDING at the start of the algorithm execution. The algorithm will keep on running on a separate worker thread in the background and at the end of it, it will upload the algorithm results

to the database and mark the algorithm status as COMPLETED. When the frontend will request the generated timetable, the server will check the algorithm status key in the Redis and if it is COMPLETED, will fetch the algorithm result from the database and send it to the frontend.

We can also see the status of the jobs in the queues on the Bull Dashboard, which can be accessed at <http://localhost:3000/bullmq>



**Fig. 8.3** Bull Dashboard

### 8.2.6 Testing APIs with Postman

To test the backend APIs separately from the frontend, we can use the Postman tool to send the request by providing the appropriate authorization headers and the request body in the expected format.

```

└─ npm run start:server

> timetable_automation@0.1.0 start:server
> redis-cli FLUSHDB && nx serve server

OK

> nx run server:serve:development

chunk (runtime: algorithm) algorithm.js (algorithm) 152 KiB [entry] [rendered]
chunk (runtime: main) main.js (main) 177 KiB [entry] [rendered]
webpack compiled successfully (3a39ff41effcc701)
Debugger listening on ws://localhost:9229/b528a6c3-d73f-43dc-989e-b3fed3b0246f
For help, see: https://nodejs.org/en/docs/inspector
Algorithm Worker Started!
Connected to MongoDB!
Listening at http://localhost:3000

```

Fig. 8.4 Start the backend server

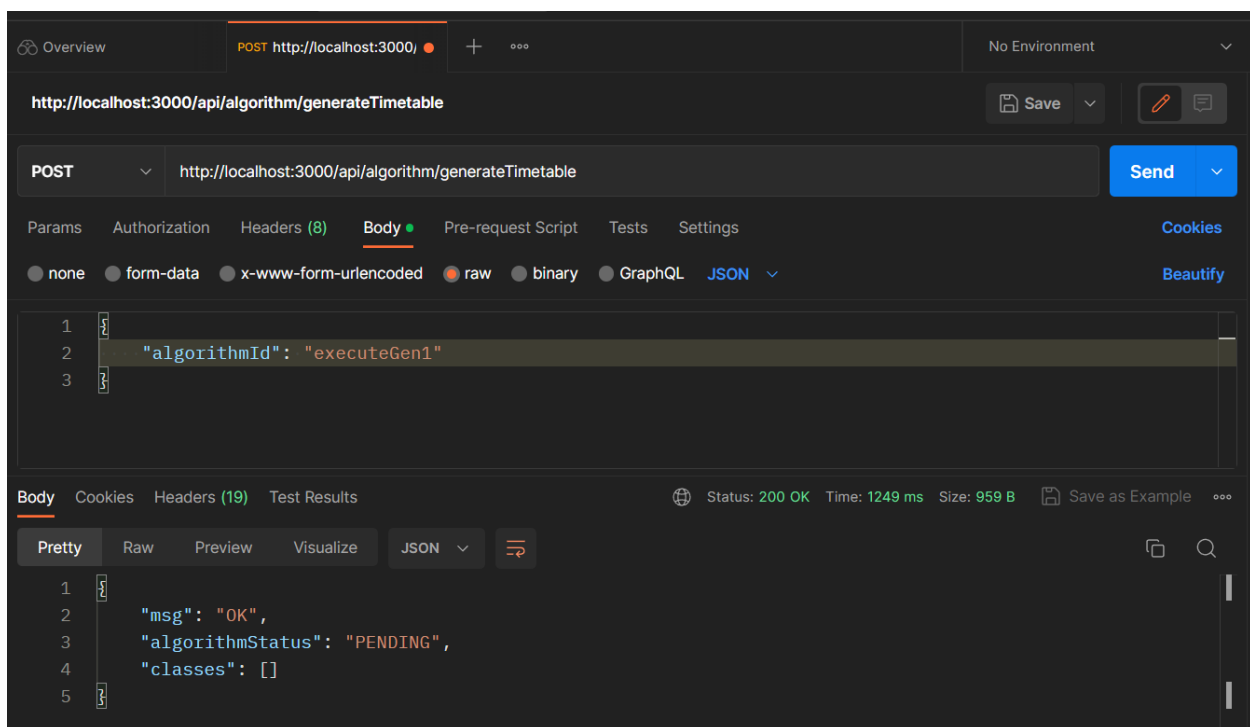


Fig. 8.5 Send the request from the Postman

## 8.3 Understanding the Frontend

### 8.3.1 Tech Stack

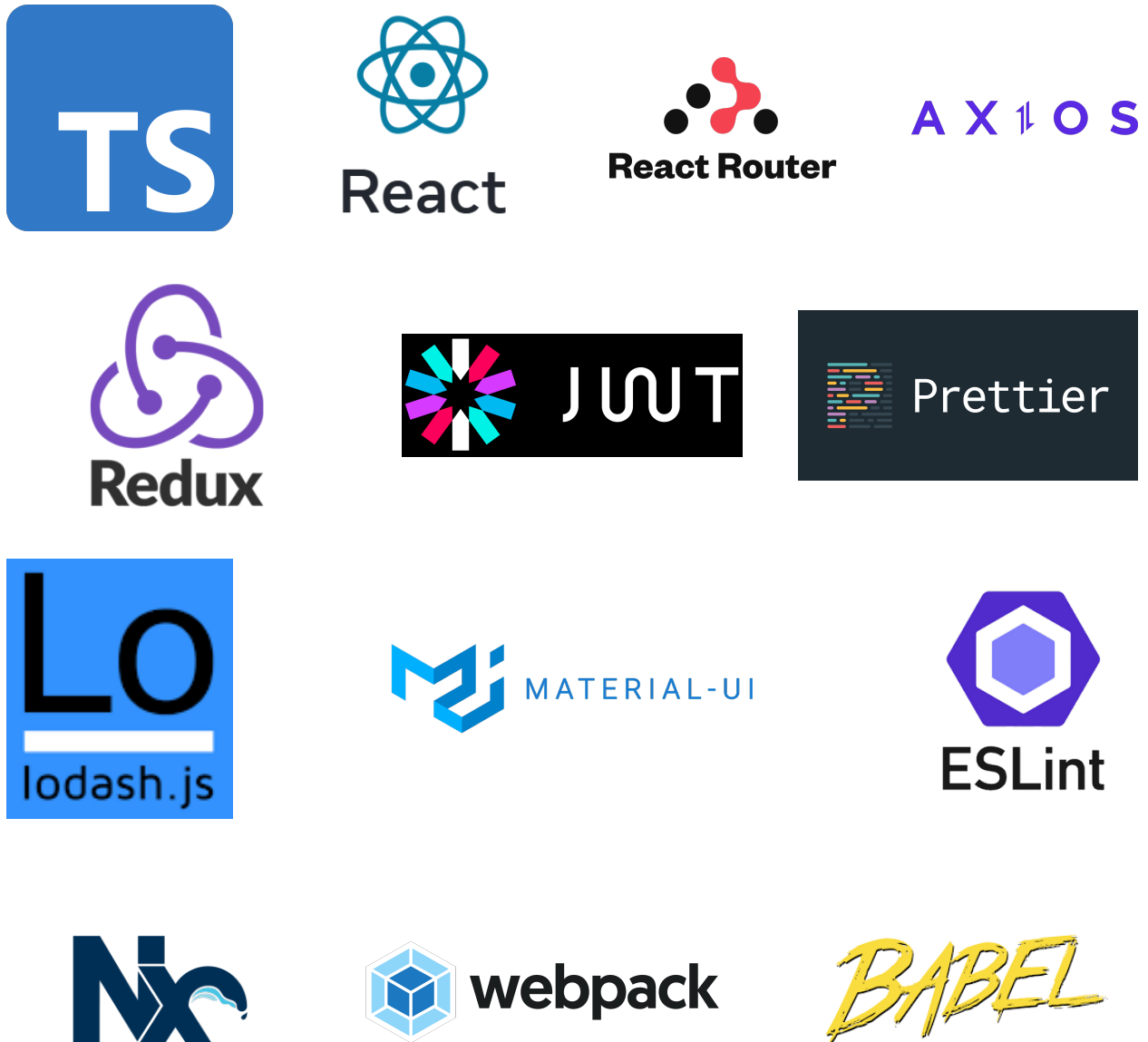


Fig. 8.6 Frontend Tech Stack

### 8.3.2 State Management with Redux Toolkit

We are using **Redux** which is a predictable state container for JavaScript applications. It helps us to manage the application state in a centralized store that is accessible to all the parts of an application. It is more helpful for applications that involve multiple pages and we need persistent state storage to share state across different pages. **Redux Toolkit** allows us to follow the best practices for writing redux code and managing state with ease and fewer errors and preventing common mistakes.

We can install the Redux DevTools Extension in the browser and can view our application state there. We can also start a redux session there and do time traveling to go to the previous state of an application to manually debug it.

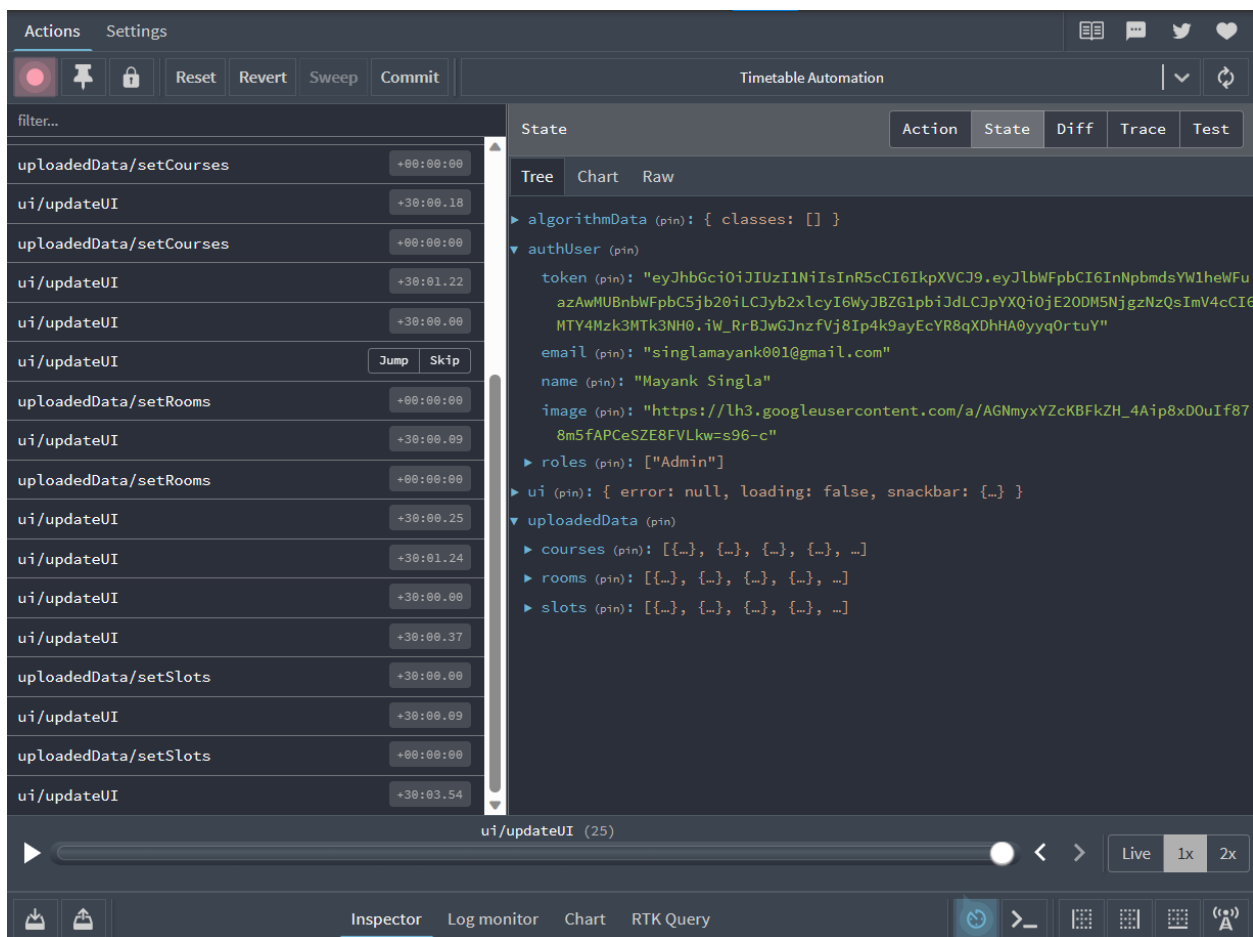
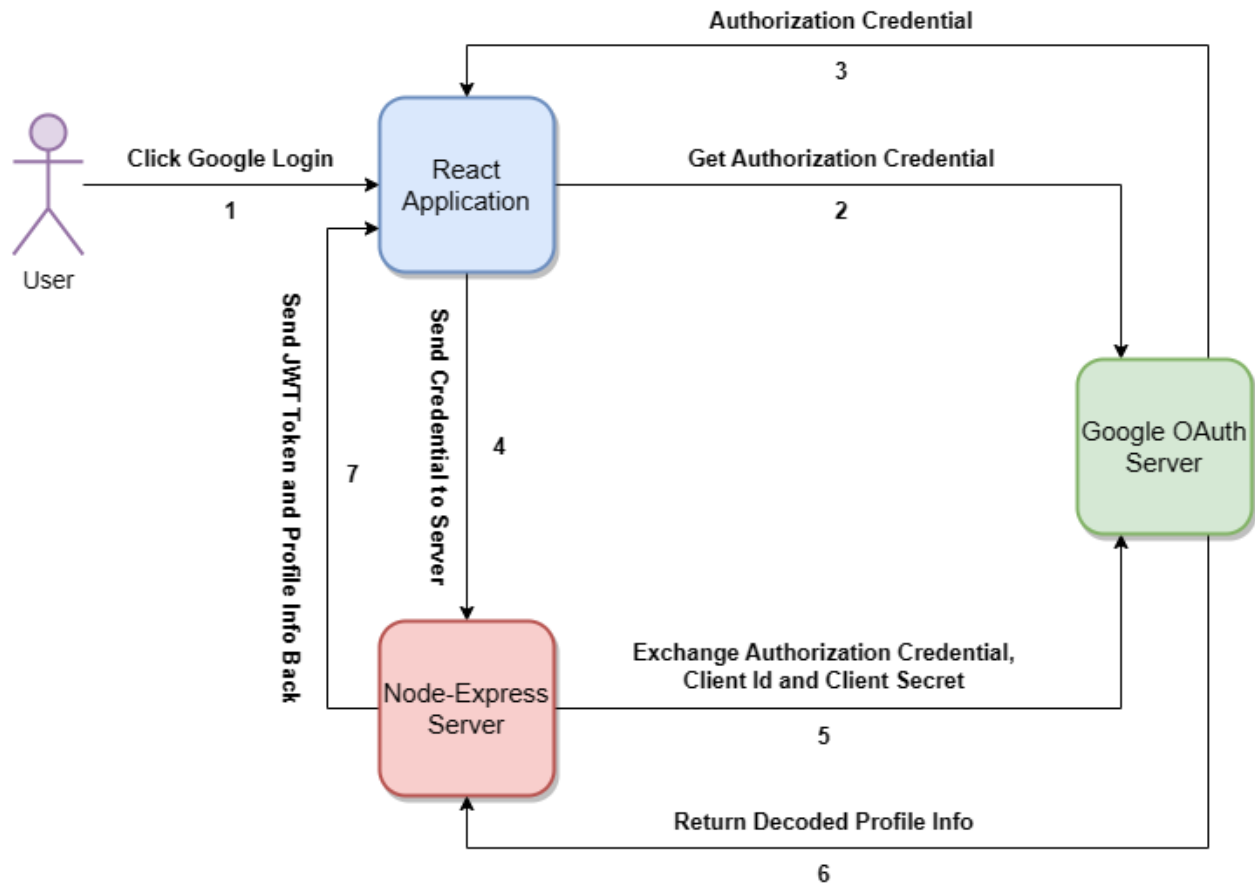


Fig. 8.7 Redux DevTools Debugging Session

### 8.3.3 Google Authentication Workflow



In addition to the above steps, the server also authenticates the user e-mail received from the Google OAuth Server from the database and set the user roles accordingly. User Roles are decided as:

- If the user's e-mail is present in the database, it is given the same roles as stored in the database
- If the user's e-mail is in the list of the tester's e-mails, it is given the role of **Admin**
- If the user's e-mail is neither present in the database nor in the list of the tester's e-mails, but it is from the institute domain, then it is given the role of **Guest**
- If none of the above cases matches, the user is **NOT authenticated**



### 8.3.4 Web Interface Features

#### Google Sign In

The user needs to log in using Google Authentication in order to access the web application. The **Authentication token** received from the server is stored in the local storage of the web browser, which will be required as an authorization header for every request to private endpoints on the server.

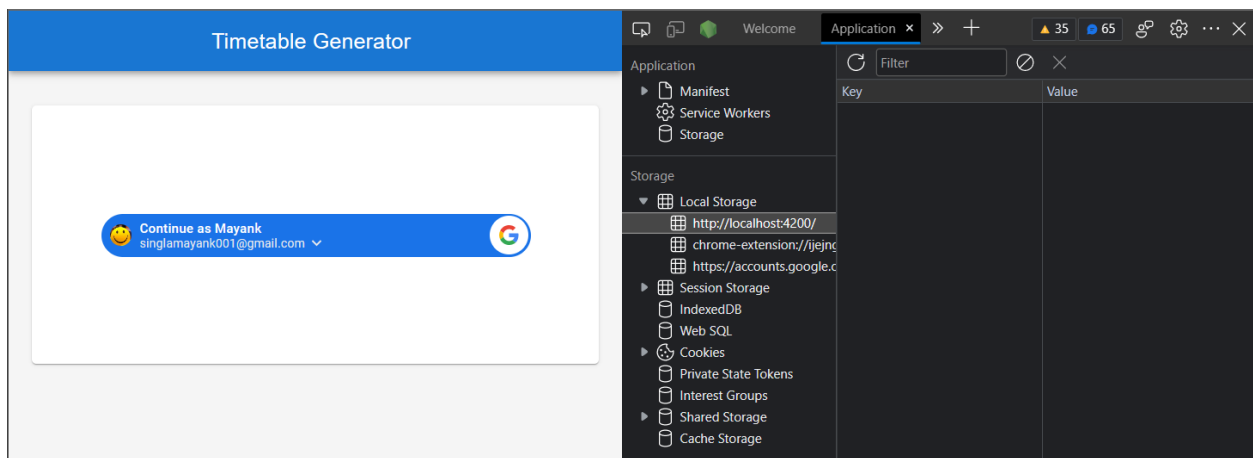


Fig. 8.8 Before Login

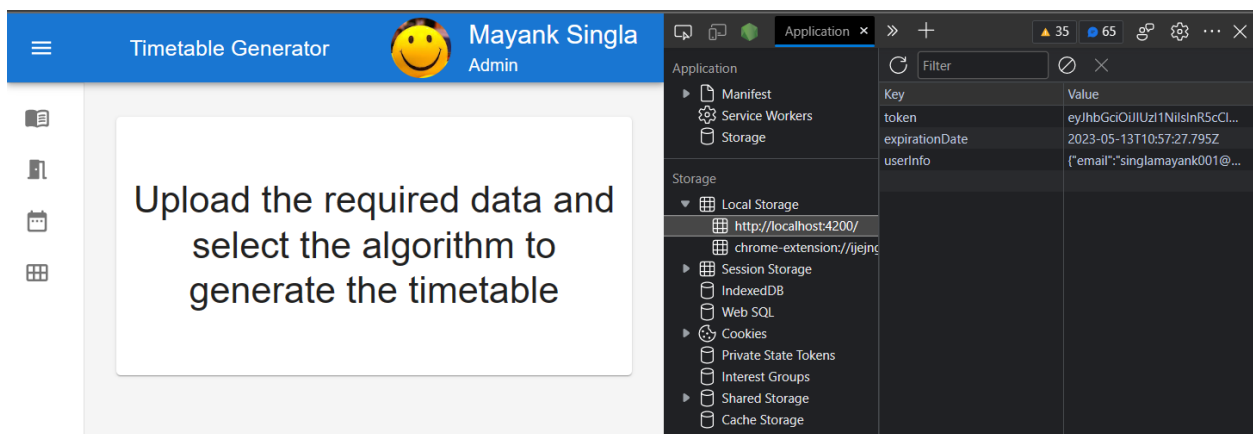


Fig. 8.9 After Google Login

## Automatic Logout

The web interface implements automatic logout of the client after a fixed duration, currently set as 1 hour. The token and other info will be removed from the local storage and the client will be redirected back to the login screen to login again. It is a good practice to implement an automatic logout feature, as the credential shared by the Google OAuth Server is also valid for a fixed duration, hence one should get a new credential after that duration.

## Routes Protection

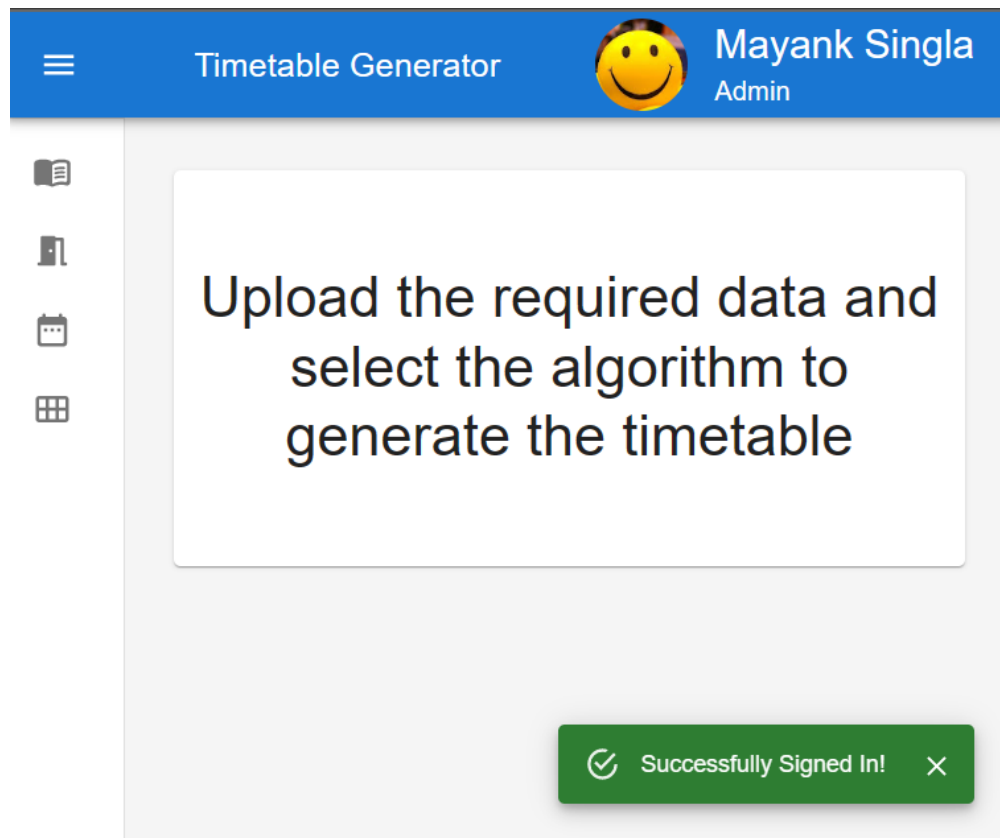
The web application has different routes to access different pages of the application. However, each page is associated with a list of user roles that are allowed to access those pages. If an unauthenticated user tries to access those pages, it will be redirected back to the login screen. If a logged-in user tries to access a route for which he is not authorized, he will be redirected to the page to which he has access. For now, the following criteria are followed for routes protection:

- Unauthenticated users can't access any page other than the login page
- Admin and Coordinator can access all the pages of the application
- A Guest can only access the timetable page where he can only interact with the generated timetable

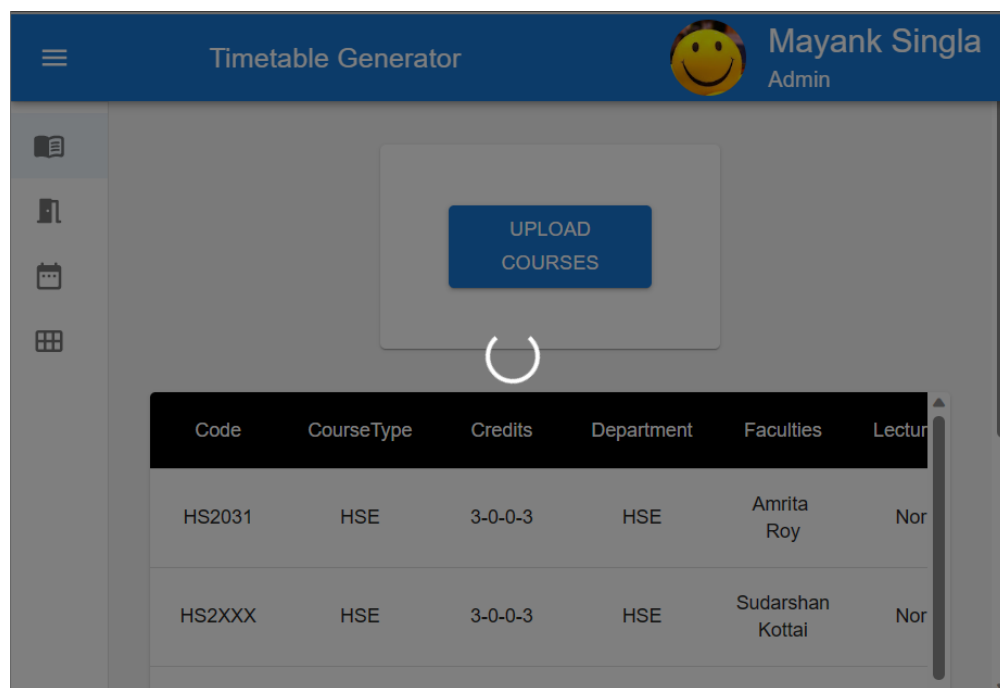
## Toast and Loading Spinner

A **Toast** is a brief notification to inform the users of a process that an application has performed. They appear for a small duration at the bottom of the screen.

A loading spinner is a circular progress indicator used to indicate that a process is currently in progress. It covers the whole application with a modal on top.



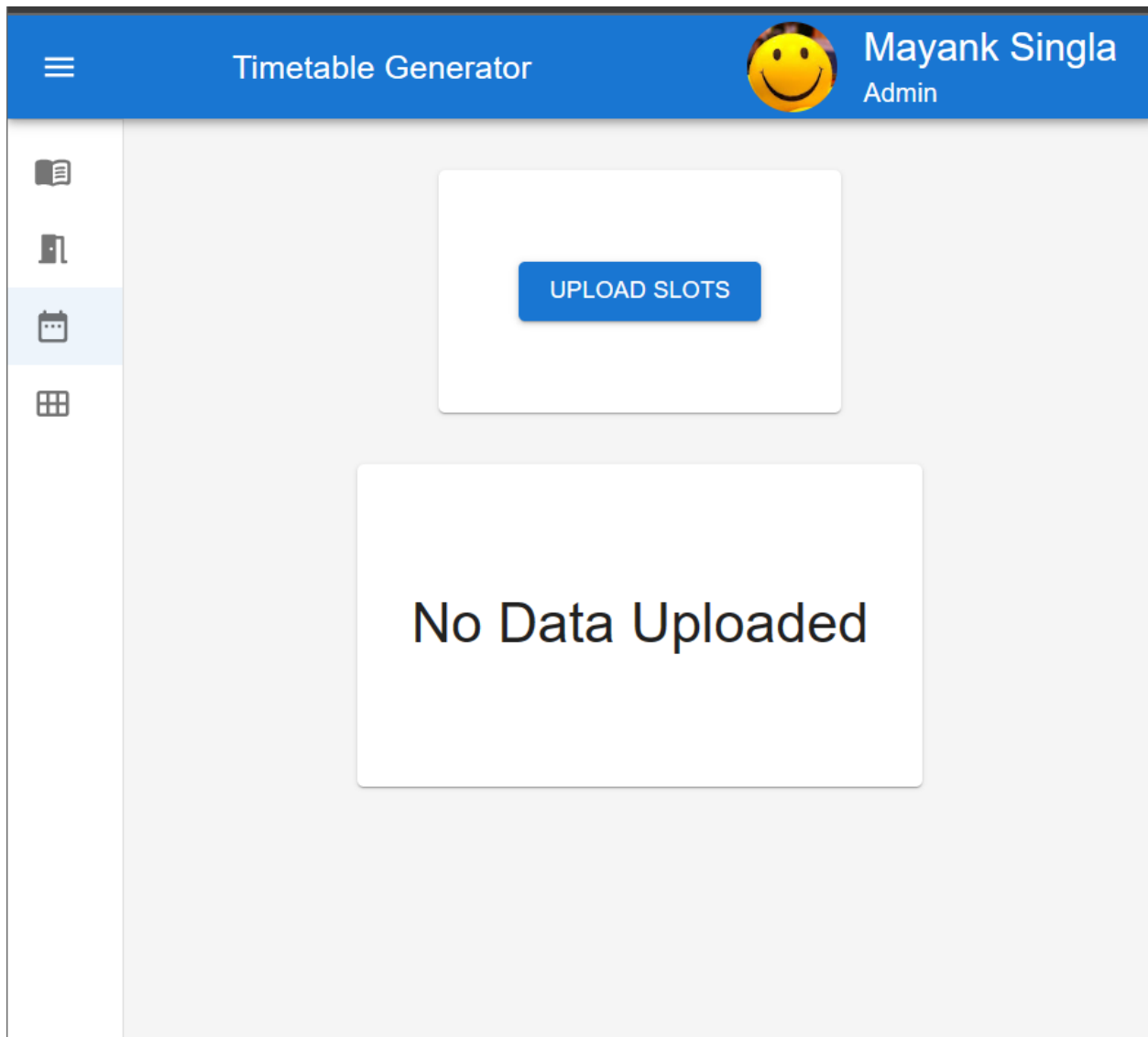
**Fig. 8.10** Toast Notification



**Fig. 8.11** Loading Spinner with Modal

## Uploading Data to the Server

The user can upload all course information, room information, and slot system information from the web application in the JSON format which will be required for executing the timetable generation algorithm later on the server.



**Fig. 8.12** When No Data is Uploaded

Timetable Generator

Mayank Singla  
Admin

UPLOAD SLOTS

Name	LectureType	DayTime
B1	Normal	Monday   08:00 AM - 08:50 AM
A	Normal	Monday   09:00 AM - 09:50 AM Tuesday   05:00 PM - 05:50 PM Wednesday   09:00 AM - 09:50 AM
H	Normal	Monday   10:00 AM - 10:50 AM Wednesday   11:00 AM - 11:50 AM Friday   08:00 AM - 08:50 AM
D	Normal	Wednesday   05:00 PM - 05:50 PM

✔ Successfully Uploaded Slots! ✕


**Fig. 8.13** After Data is Uploaded

## Automatic Fetching Data from the Server


If any data was previously uploaded to the server, it will automatically be fetched from the server again and displayed on the web application as soon as the user navigates to the corresponding web page.



## Pagination to Display the Long List of Data


The user can select the number of rows to display at once in the data table and navigate across different pages created for that table.




Timetable Generator

 Mayank Singla  
Admin







Name	LectureType	DayTime
B	Normal	Wednesday   10:00 AM - 10:50 AM Thursday   05:00 PM - 05:50 PM Friday   11:00 AM - 11:50 AM
P3	Lab	Wednesday   02:00 PM - 04:45 PM
A1	Normal	Thursday   08:00 AM - 08:50 AM
P4	Lab	Thursday   02:00 PM - 04:45 PM
P5	Lab	Friday   02:00 PM - 04:45 PM

Rows per page:

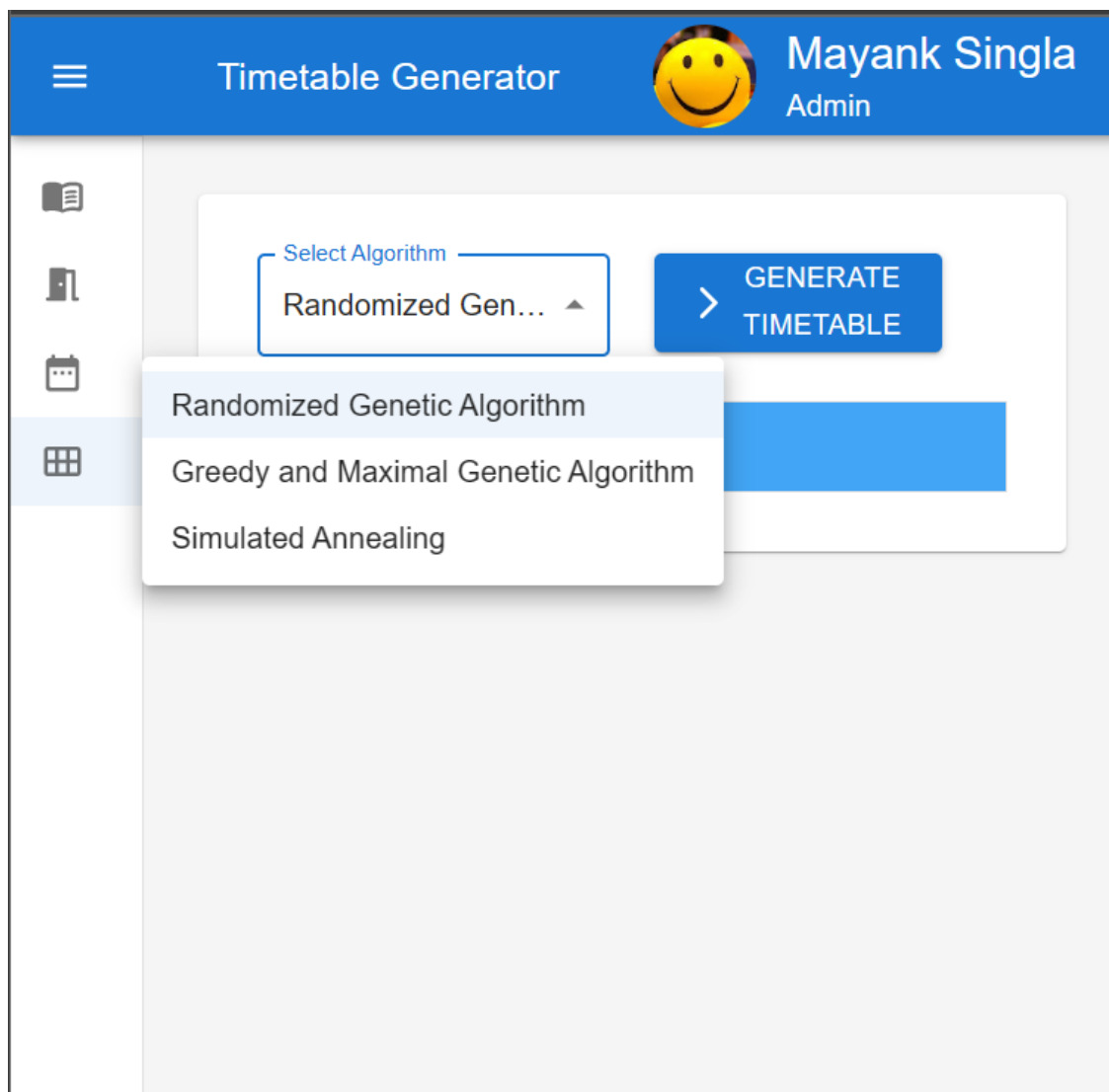
1025100



11-17 of 17<>





## Select Algorithm to Use for Timetable Generation

As we tried various different algorithms for timetable automation, we provide the user to select among those algorithms for generating the timetable. It will send the request to the server to execute the algorithm with the uploaded data in a background process on the server.

The frontend will then keep asking the server about the algorithm status at regular intervals, and once the algorithm execution is finished on the server, the frontend will display the button to see the generated timetable.



 Timetable Generator  Mayank Singla  
Admin



Select Algorithm  
Randomized Gen... ▼

> GENERATE  
TIMETABLE

▼ Algorithm Logs

→ SEE  
TIMETABLE



Visualizing Timetable Interactively

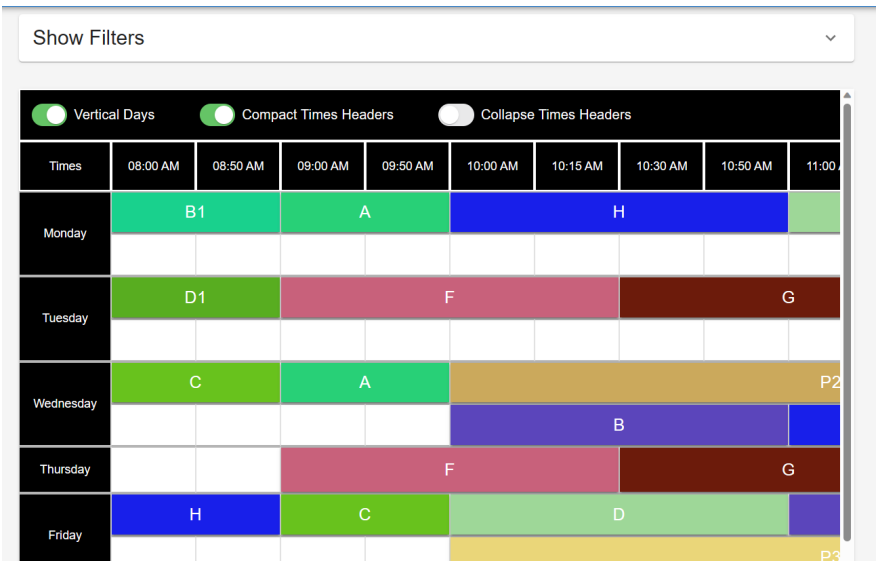


Fig. 8.14 Vertical Days Timetable

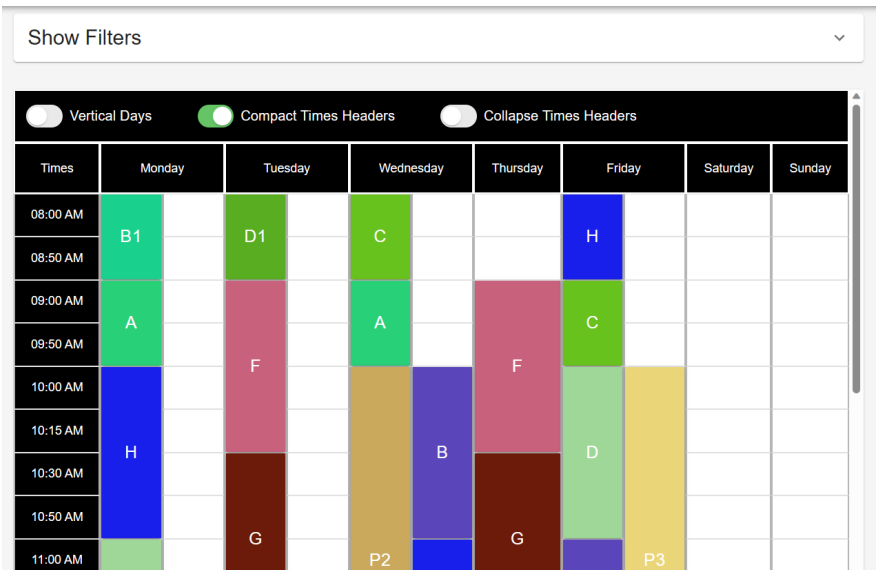


Fig. 8.15 Horizontal Days Timetable

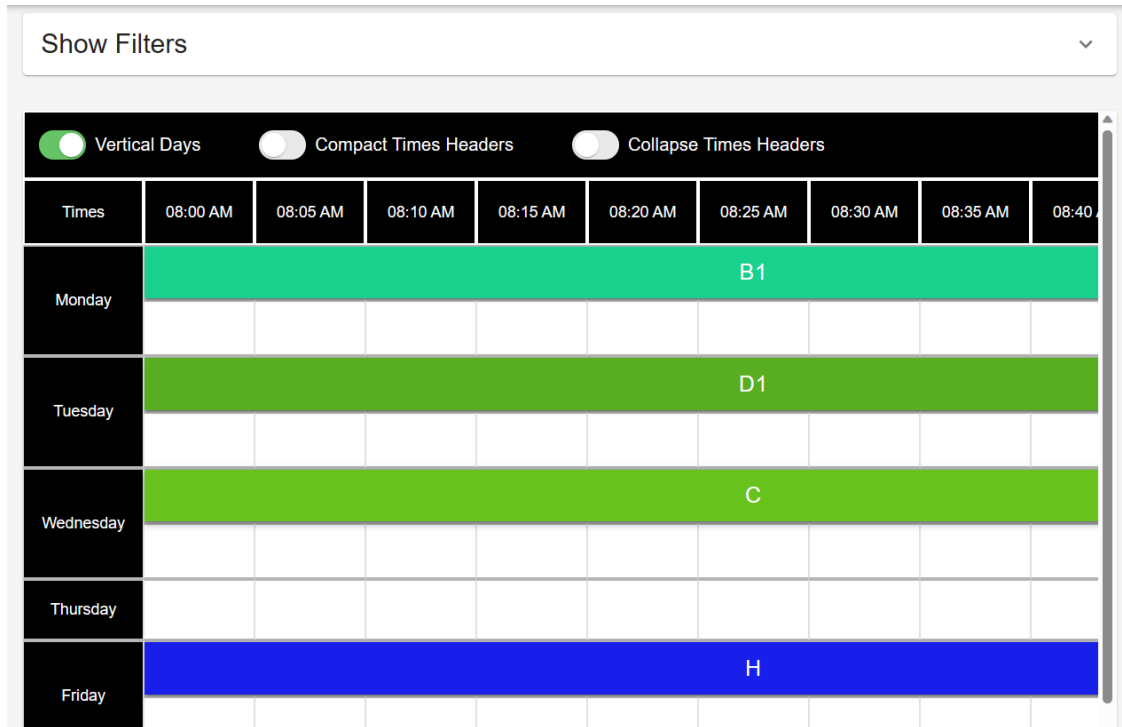


Fig. 8.16 Elongated Times Timetable with 5 minutes Interval

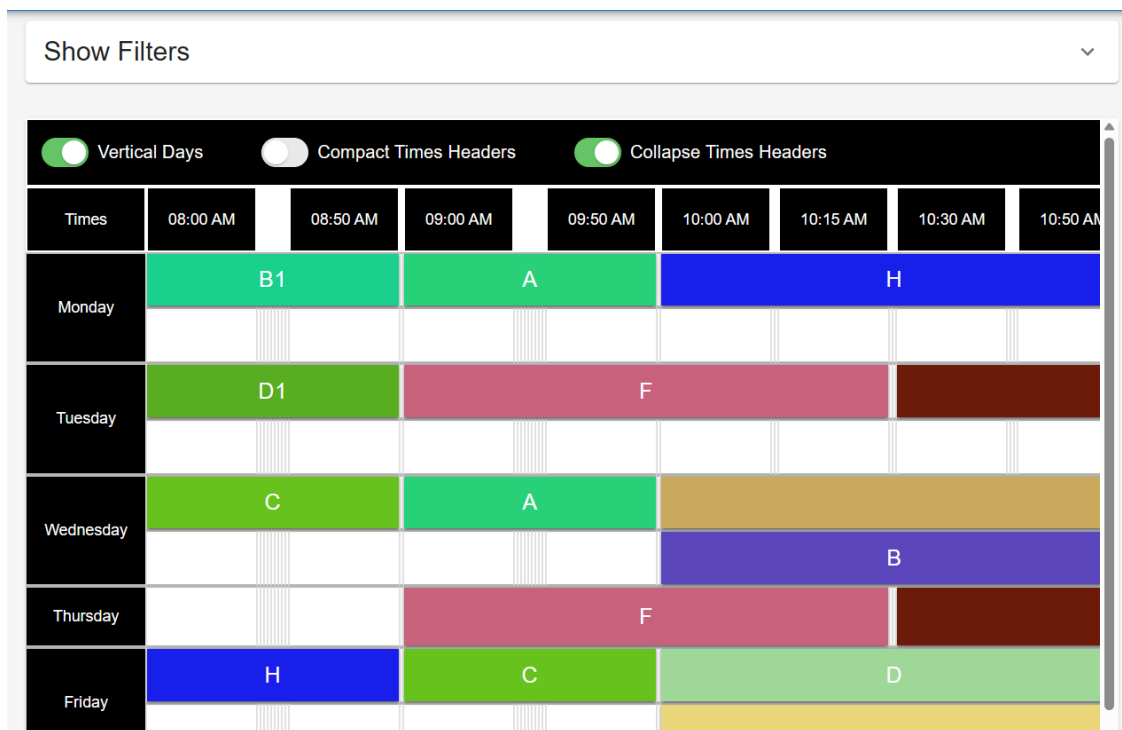
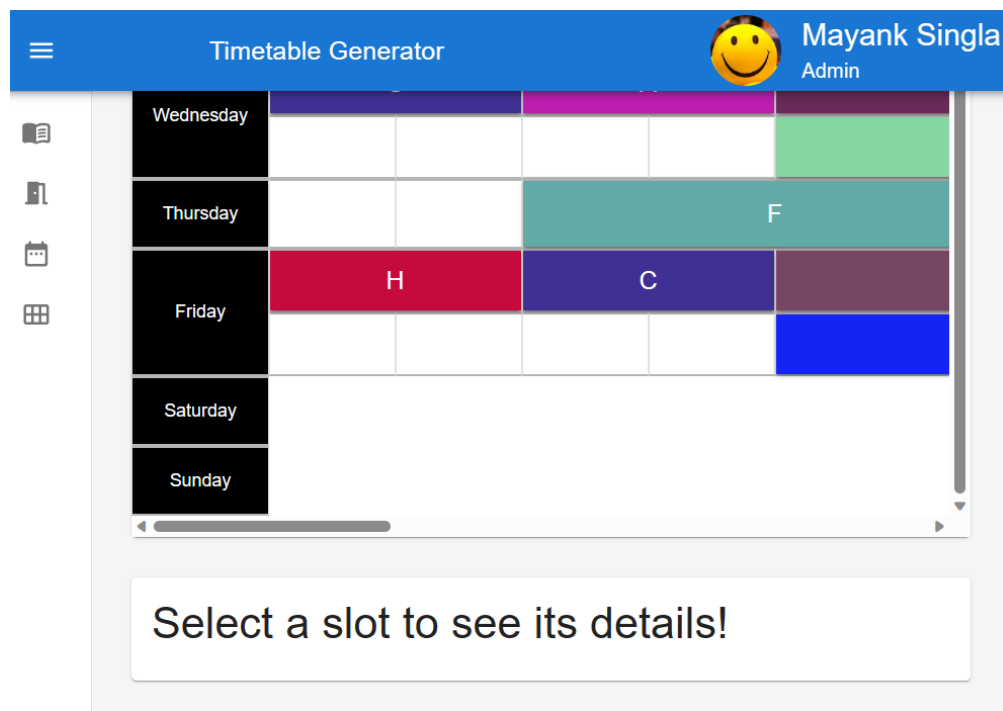
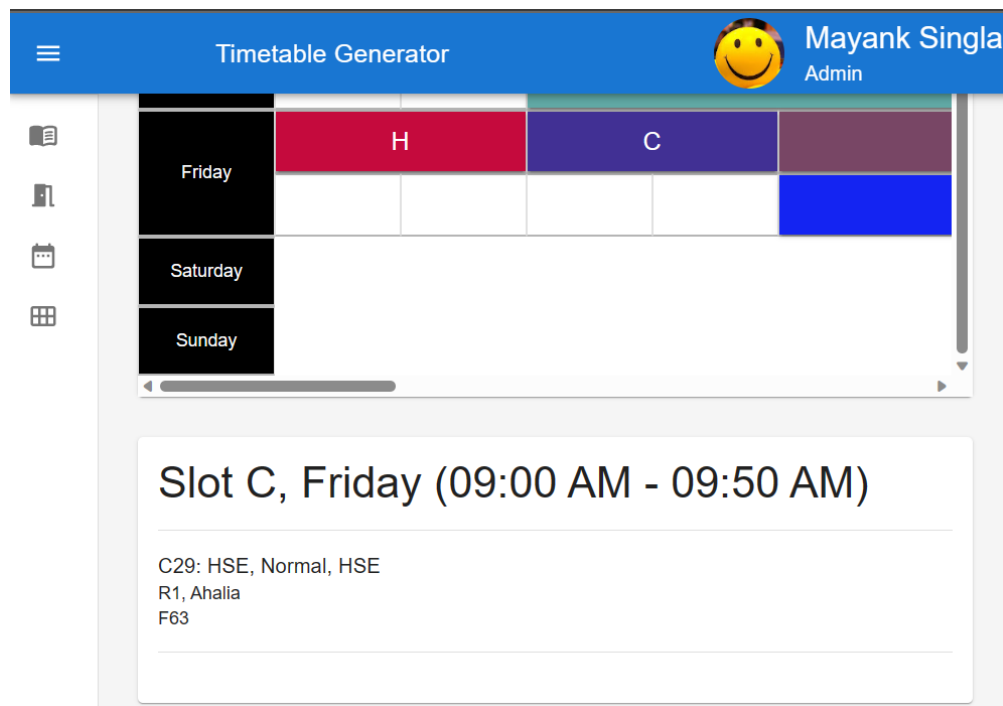


Fig. 8.17 Collapsed Times Timetable when Compact Headers is OFF



**Fig. 8.18** User Can Select a Slot from a Cell to See its Details



**Fig. 8.19** Selected Slot Displayed Information

## Applying Filters to Timetable

The 'Hide Filters' panel displays several filter categories with their current selections:

- Courses:** (Dropdown menu)
- Rooms:** R1, Ahalia; R4, Nila (Multi-select)
- Faculties:** (Dropdown menu)
- Slots:** (Dropdown menu)
- Departments:** (Dropdown menu)
- CourseTypes:** (Dropdown menu)
- LectureTypes:** (Dropdown menu)
- Campuses:** Ahalia; Nila (Multi-select, currently open)

**Fig. 8.20** User Can Multi-Select Various Filters

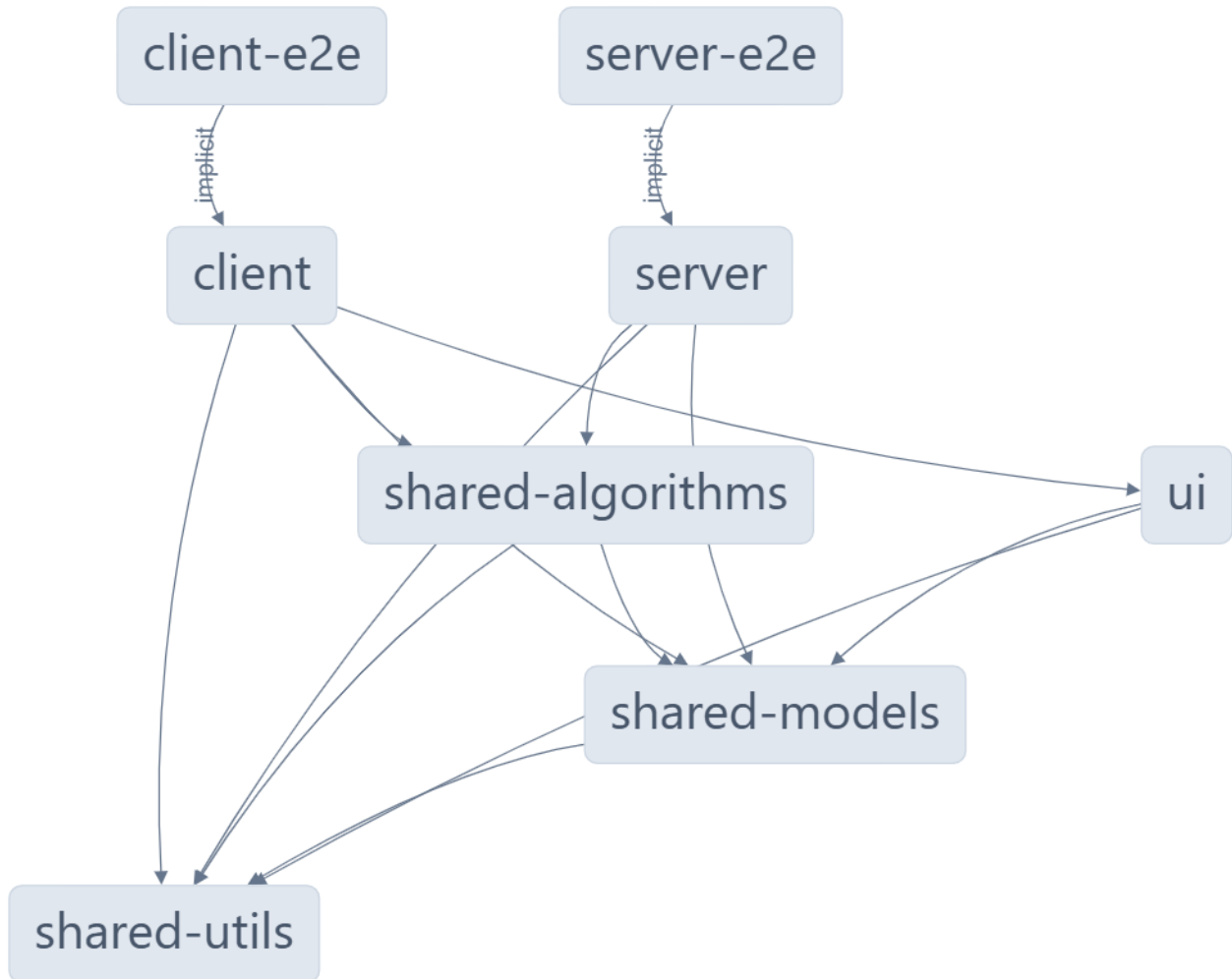
The Timetable view includes three toggle switches at the top: 'Vertical Days' (off), 'Compact Times Headers' (on), and 'Collapse Times Headers' (off). The table below shows the filtered data.

Times	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
08:00 AM	B1		C				
08:50 AM	B1		C				
09:00 AM		F		F	C		
09:50 AM		F		F	C		
10:15 AM		F		F			
05:00 PM	C				F		
05:50 PM	C				F		

**Fig. 8.21** Timetable with Filtered Data

## 8.4 Workspace Dependency Graph

Here is the workspace dependency graph through which we can visualize how different applications and libraries are dependent on other shared libraries.



## 8.5 Conclusion

In this chapter, we discussed the various web technologies used for building the web application. We discussed the need for using certain kinds of tools and how to configure them for our application usage. We also discussed the detailed usage of both the frontend and backend applications and the various features implemented in them. In the next chapter, we will discuss the future work and the scope of the project.

# Chapter 9

## Future Work

- Attempt to implement **Kempe chain** for a more faster Simulated Annealing algorithm
- Allow the admin or the coordinator to provide an initial allocation for the timetable, and execute the timetable allocation algorithm on top of that initial allocation
- Allow the admin or the coordinator to see the conflicts produced from the timetable allocation and provide a drag and drop interface to make it an **interactive** timetabling having user input
- Make the frontend for the admin to add/edit/delete users and allow the admin to not only replace the whole algorithm data at once but also edit/add to the existing data
- Better display of error messages from the server on the frontend
- Integrate the web application with MRBS to upload the generated timetable directly to the MRBS servers
- Allow the user to download the generated timetable in different supported formats like PDF, JSON, .ics file
- Explore the UniTime Software

# References

- [1] J.-K. H. Zhipeng Lü, “Adaptive tabu search for course timetabling,” 2010. [Online]. Available: [https://www.researchgate.net/publication/222418829\\_Adaptive\\_Tabu\\_Search\\_for\\_course\\_timetabling](https://www.researchgate.net/publication/222418829_Adaptive_Tabu_Search_for_course_timetabling)
- [2] S. Sahul and S. Santhosh, “Time Table Automation And Integration with MRBS,” Indian Institute of Technology Palakkad, Tech. Rep., 2021.
- [3] P. Guo, J. xin Chen, and L. Zhu, “The design and implementation of timetable system based on genetic algorithm,” 2010. [Online]. Available: <https://ieeexplore.ieee.org/document/6025756>
- [4] T. Muller, “Itc2007 solver description: A hybrid approach.” [Online]. Available: <https://www.unitime.org/papers/itc2007.pdf>
- [5] M. Jankovic, “Making a class schedule using a genetic algorithm.” [Online]. Available: <https://www.codeproject.com/Articles/23111/Making-a-Class-Schedule-Using-a-Genetic-Algorithm#Algorithm13>
- [6] V. Mallawaarachchi, “Using genetic algorithms to schedule timetables.” [Online]. Available: <https://towardsdatascience.com/using-genetic-algorithms-to-schedule-timetables-27f132c9e280>
- [7] “The slot assignment problem.” [Online]. Available: [https://www.iitg.ac.in/engfac/narayan/public\\_html/intro-to-simulated-annealing.htm](https://www.iitg.ac.in/engfac/narayan/public_html/intro-to-simulated-annealing.htm)

- [8] A. Rjoub, “Courses timetabling based on hill climbing algorithm.” [Online]. Available: <https://core.ac.uk/download/pdf/329119057.pdf>
- [9] N. Leite, F. Melicio, and A. C. Rosa, “A fast simulated annealing algorithm for the examination timetabling problem.” [Online]. Available: <https://www.researchgate.net/publication/329997648>
- [10] T. CURA, “Timetabling of faculty lectures using simulated annealing algorithm.” [Online]. Available: <https://www.ticaret.edu.tr/uploads/yayin/dergi/f12/M00196.pdf>
- [11] S. Abdennadher and M. Marte, “Universtiy course timetabling using constraint handling rules.” [Online]. Available: <https://www.en.pms.iflmu.de/publications/PMS-FB/PMS-FB-2000-2/PMS-FB-2000-2.pdf>
- [12] R. Bai, E. K. Burke, G. Kendall, and B. Mccollum, “A simulated annealing hyper-heuristic for university course timetabling.” [Online]. Available: <https://www.researchgate.net/publication/228918795>
- [13] E. Gashi and K. Sylejmani, “Simulated annealing with penalization for university course timetabling.” [Online]. Available: <https://www.itc2019.org/papers/itc2019-gashi.pdf>
- [14] K. Sylejmani, E. Gashi, and A. Ymeri, “Simulated annealing with penalization for university course timetabling.” [Online]. Available: <https://link.springer.com/article/10.1007/s10951-022-00747-5>
- [15] N. Basir, W. Ismail, and N. M. Norwawi, “A simulated annealing for tahmidi course timetabling.” [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S2212017313003678>
- [16] E. Aycaan and T. Ayav, “Solving the course scheduling problem using simulated annealing.” [Online]. Available: <https://www.researchgate.net/publication/224398617>



- [17] R. Garg, “Simulated annealing in competitive programming.” [Online]. Available: <https://codeforces.com/blog/entry/94437>