Revise Final Design

FYP-Group 14

Project Title

“Automatic Course Scheduler”

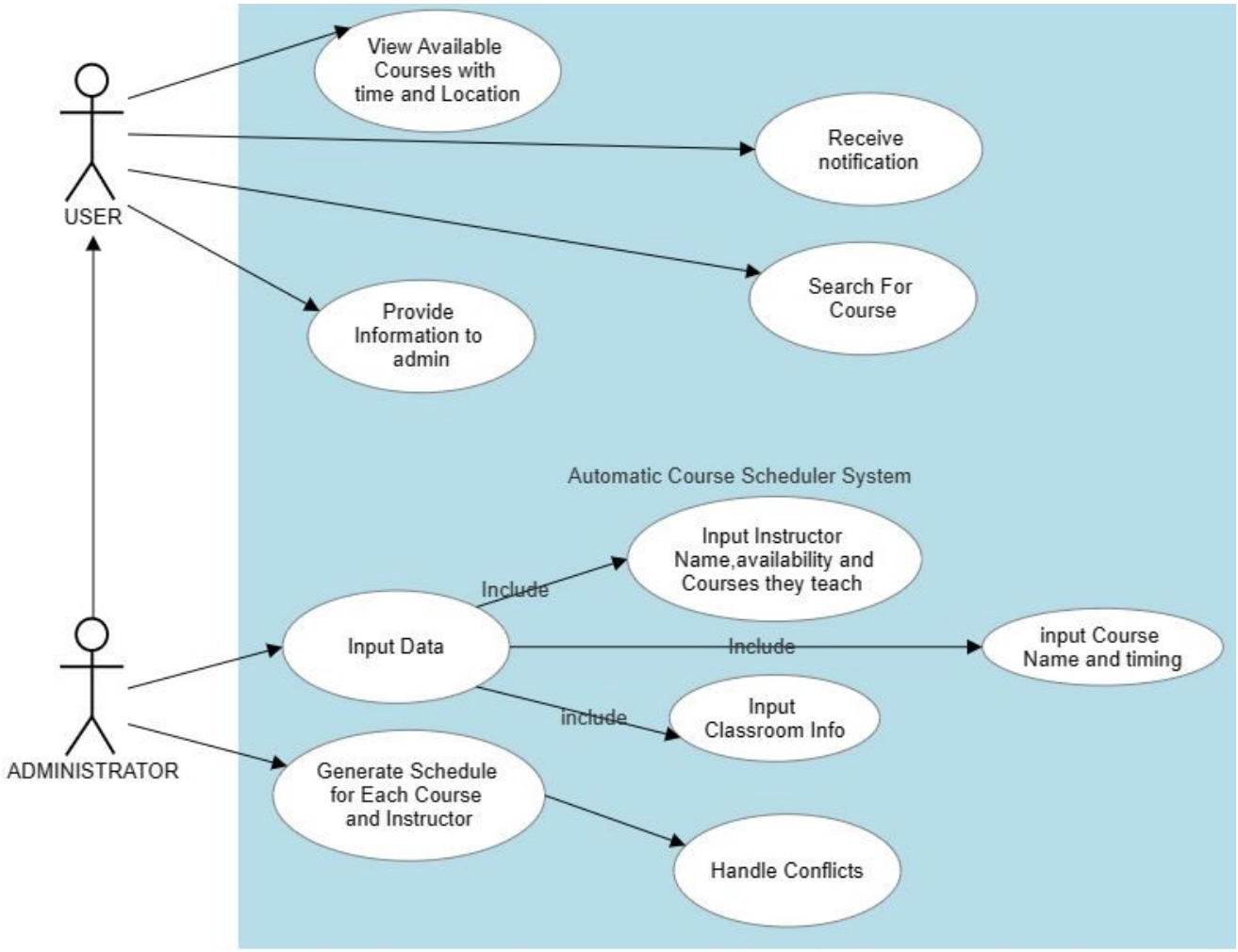
Instructor: DR. AARIJ MAHMOOD HUSSAAN

Group Members	Reg_ID	Email
Abdul Majid	51347	Abdul.51347@iqra.edu.pk
Muhammad	51228	Muhammad.51228@iqra.edu.pk
Taha Mir	50560	Taha.50560@iqra.edu.pk

Table of Contents

1st Design of Iteration:

UML USE CASE DIAGRAM:



Automated Course Scheduler:

METADATA:

Actors:

1) **USER**
2) **Administrator**

USER'S USE CASES:

1. **Use Case:** View Available Courses with Time and Location
    - **Goal:** To allow the user to view a list of available courses with their respective times and locations.
    - **Actor:** User
    - **Trigger:** The user initiates the action to view available courses.
    - **Pre-condition:** Available course data is stored in the system.
    - **Post-condition:** The user can see a list of available courses with their times and locations.

2. **Use Case:** Provide Information to Admin
    - **Goal:** To enable the user to provide relevant information or feedback to the administrator.
    - **Actor:** User
    - **Trigger:** User decides to provide information or feedback.
    - **Pre-condition:** User has relevant information or feedback to share.
    - **Post-condition:** Information or feedback is submitted to the administrator.

3. **Use Case:** Search for Course.
    - **Goal:** To allow the user to search for a specific course based on different criteria.
    - **Actor:** User
    - **Trigger:** User initiates a search for a course.
    - **Pre-condition:** Search criteria and course data are available in the system.
    - **Post-condition:** User receives search results matching the specified criteria.

4. **Use Case:** Receive Notifications.
    - **Goal:** To enable the user to receive notifications about course updates, schedule changes, or other relevant information.
    - **Actor:** User
    - **Trigger:** User receives a notification.
    - **Pre-condition:** Notifications are generated and sent by the system.
    - **Post-condition:** User is informed about course updates, schedule changes, or other relevant information.

ADMINISTRATOR USE CASES:

1. **Use Case:** Input Course, Teacher, Classroom Information
    - **Goal:** To allow the administrator to input course, teacher, and classroom information.
    - **Actor:** Administrator
    - **Trigger:** Administrator initiates the input process.
    - **Pre-condition:** The system is ready to receive data input.
    - **Post-condition:** Course, teacher, and classroom information is stored in the system.

2. **Use Case:** Generate Schedule
    - **Goal:** To generate a schedule for each course, teacher, and classroom.
    - **Actor:** Administrator
    - **Trigger:** Administrator initiates the schedule generation process.
    - **Pre-condition:** Course, teacher, and classroom information is available in the system.
    - **Post-condition:** A schedule is generated for each course, teacher, and classroom.

3. **Use Case:** Handle Conflicts and Adjust Schedules
    - **Goal:** To handle conflicts and adjust schedules automatically.
    - **Actor:** Administrator
    - **Trigger:** Conflicts are detected during the schedule generation process.
    - **Pre-condition:** A schedule with conflicts exists.
    - **Post-condition:** Schedules are adjusted, conflicts are resolved, and an updated schedule is generated.

4. **Use Case:** Display the Schedule in User-Friendly Interface
    - **Goal:** To display the generated schedule in a user-friendly interface.
    - **Actor:** Administrator
    - **Trigger:** The administrator selects the option to display the schedule.
    - **Pre-condition:** A schedule is available in the system.
    - **Post-condition:** The schedule is displayed in a user-friendly interface.

5. **Use Case:** Accommodate Instructor Preferences and Availability
    - **Goal:** To consider instructor preferences and availability during schedule generation.
    - **Actor:** Administrator

- **Trigger:** The administrator initiates the schedule generation process.
- **Pre-condition:** Instructor preferences and availability data is available.
- **Post-condition:** The schedule is generated, ensuring no instructor has more than one class at a time and accommodating preferred class times.

6. **Use Case:** Notify Users of Schedule Changes
    - **Goal:** To send notifications to users when changes are made to the schedule.
    - **Actor:** Administrator
    - **Trigger:** Schedule changes are made.
    - **Pre-condition:** Schedule changes are made in the system.
    - **Post-condition:** Users receive notifications about the schedule changes.

SYSTEM'S USE CASES:

1. **Use Case:** Manage Course Data
    - **Goal:** To allow the system to manage course-related data, including course names, schedules, and locations.
    - **Actor:** System
    - **Trigger:** The system receives course data from the administrator.
    - **Pre-condition:** Course data is provided by the administrator.
    - **Post-condition:** Course data is stored and managed by the system.

2. **Use Case:** Manage Instructor Data
    - **Goal:** To enable the system to manage instructor-related data, including instructor names, availability, and courses they teach.
    - **Actor:** System
    - **Trigger:** The system receives instructor data from the administrator.
    - **Pre-condition:** Instructor data is provided by the administrator.
    - **Post-condition:** Instructor data is stored and managed by the system.

3. **Use Case:** Manage Classroom Data
    - **Goal:** To facilitate the system in managing classroom-related data, including classroom availability and capacity.
    - **Actor:** System

- **Trigger:** The system receives classroom data from the administrator.
- **Pre-condition:** Classroom data is provided by the administrator.
- **Post-condition:** Classroom data is stored and managed by the system.

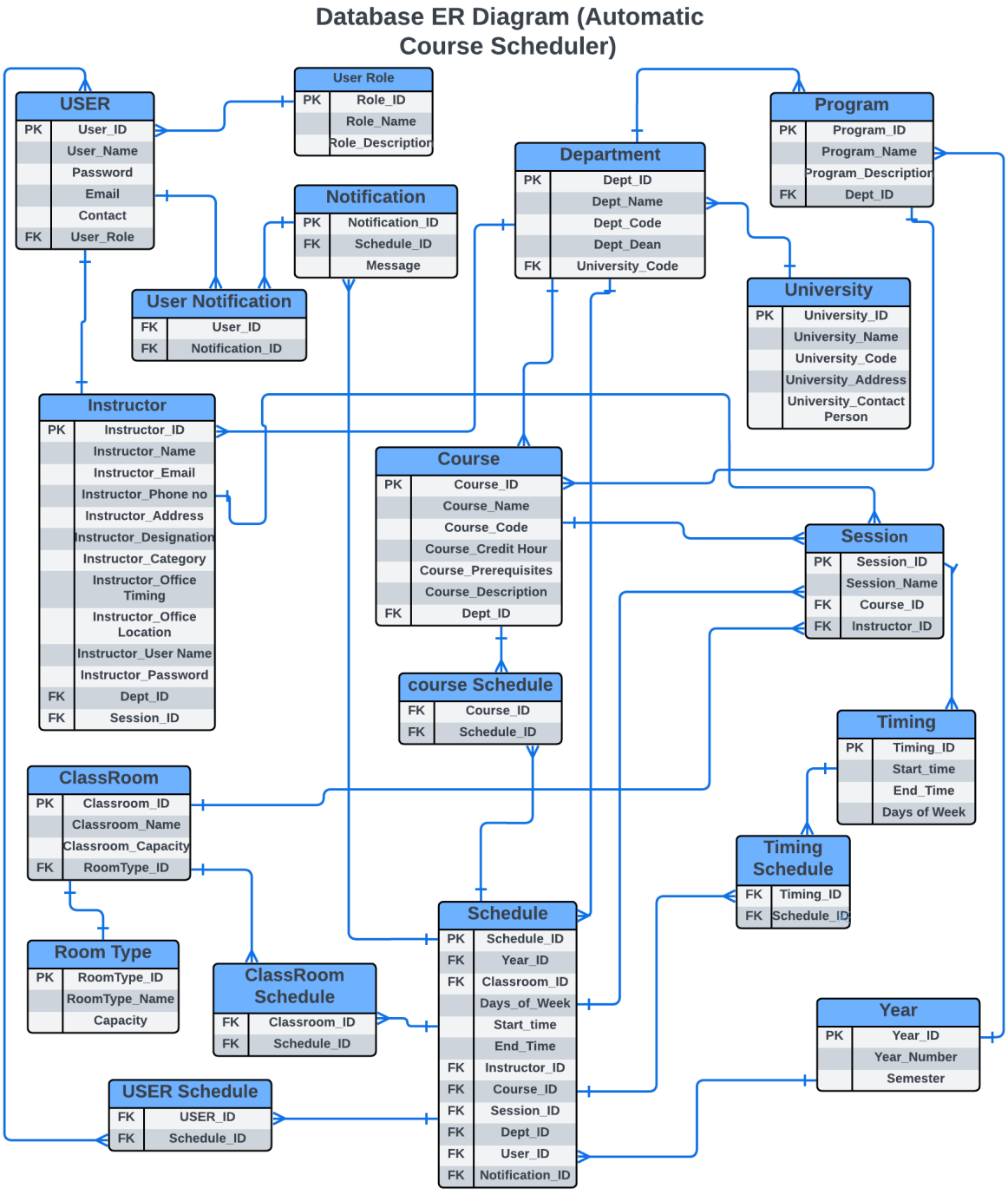
4. **Use Case:** Handle Schedule Generation
    - **Goal:** To enable the system to handle the generation of schedules for courses, instructors, and classrooms.
    - **Actor:** System
    - **Trigger:** The system receives a request to generate schedules from the administrator.
    - **Pre-condition:** Course, instructor, and classroom data are available in the system.
    - **Post-condition:** Schedules are generated and stored by the system.

5. **Use Case:** Manage Schedule Conflict Resolution
    - **Goal:** To allow the system to handle conflicts that may arise during schedule generation and make necessary adjustments.
    - **Actor:** System
    - **Trigger:** Schedule conflicts are detected during the schedule generation process.
    - **Pre-condition:** Conflicting schedules exist.
    - **Post-condition:** Conflicts are resolved, and an updated schedule is stored by the system.

6. **Use Case:** Send Schedule Change Notifications
    - **Goal:** To enable the system to send notifications to users when changes are made to the schedule.
    - **Actor:** System
    - **Trigger:** The system detects changes in the schedule.
    - **Pre-condition:** Schedule changes are made in the system.
    - **Post-condition:** Users receive notifications about the schedule changes.

2nd Design of iteration:

Entity Relationship Diagram:



Relationships Information:

Course and Sessions:

Relationship: One-to-Many

Explanation: A course can have multiple sessions, but a session belongs to only one course. Therefore, there is a one-to-many relationship between the Course and Sessions entities.

Sessions and Timing:

Relationship: One-to-Many

Explanation: A session can have multiple meeting timings, but a meeting timing belongs to only one session. Thus, there is a one-to-many relationship between the Sessions and Timing entities.

Sessions and Classroom:

Relationship: Many-to-Many

Explanation: One Session is assigned to One Classroom, and a Classroom can host Multiple Sessions. (One-to-Many relationship) - Each session is scheduled in a specific classroom, while a classroom can host multiple sessions.

Classroom and Classroom type:

Relationship: One-to-One

Explanation: Each classroom belongs to a specific classroom type, and a classroom type can be associated with only one classroom. Therefore, there is a one-to-one relationship between the Classroom and Classroom type entities.

University and Department:

Relationship: One-to-Many

Explanation: A university can have multiple departments, but a department belongs to only one university. Thus, there is a one-to-many relationship between the University and Department entities.

Program and Department:

Relationship: One-to-Many

Explanation: A program can belong to only one department, but a department can have multiple programs. Hence, there is a one-to-many relationship between the Program and Department entities.

Year and Program:

Relationship: One-to-Many

Explanation: A year can have multiple programs, but a program belongs to only one year. Therefore, there is a one-to-many relationship between the Year and Program entities.

Schedule and Year:

Relationship: One-to-Many

Explanation: A schedule is created for a specific year, but a year can have multiple schedules. Hence, there is a one-to-many relationship between the Schedule and Year entities.

User and User Role:

Relationship: Many-to-One

Explanation: Multiple users can have the same user role, but a user can have only one user role. Therefore, there is a many-to-one relationship between the User and User Role entities.

User and Notification:

Relationship: Many-to-Many

Explanation: The relationship between "Notification" and "User" is a many-to-many relationship facilitated by an intermediate table called "User_Notification."

Schedule and User:

Relationship: Many-to-Many

Explanation: A schedule can involve multiple users, and a user can participate in multiple schedules. The relationship between "User" and "Schedule" is a many-to-many relationship facilitated by an intermediate table called "User_Schedule."

Schedule and Classroom:

Relationship: Many-to-Many

Explanation: Many schedules can be assigned to a many classroom. and, each schedule is allocated to many classroom. The relationship between "Classroom" and "Schedule" is a many-to-many relationship facilitated by an intermediate table called "Classroom_Schedule.

Schedule and Notification:

Relationship: One-to-Many

Explanation: A single schedule can have multiple notifications associated with it. However, each notification is specific to one particular schedule.

Department and University:

Relationship: One-to-Many

Explanation: A department belongs to only one university, but a university can have multiple departments. This one-to-many relationship signifies that departments are associated with a specific university.

Year and Schedule:

Relationships: One-to-Many

Explanation: A year can have multiple schedules, but a schedule is created for a specific year. This one-to-many relationship indicates that schedules are associated with a particular year.

Program and Course:

Relationship: One-to-Many

Explanation: A program can offer multiple courses, but a course belongs to only one program. This one-to-many relationship denotes that courses are associated with a specific program.

Schedule and Timing:

Relationship: Many to Many

Explanation: Each timing can have multiple schedule and each schedule can have multiple timing. This relationship is many to many. The relationship between "Timing" and "Schedule" is a many-to-many relationship facilitated by an intermediate table called "Timing_Schedule”.

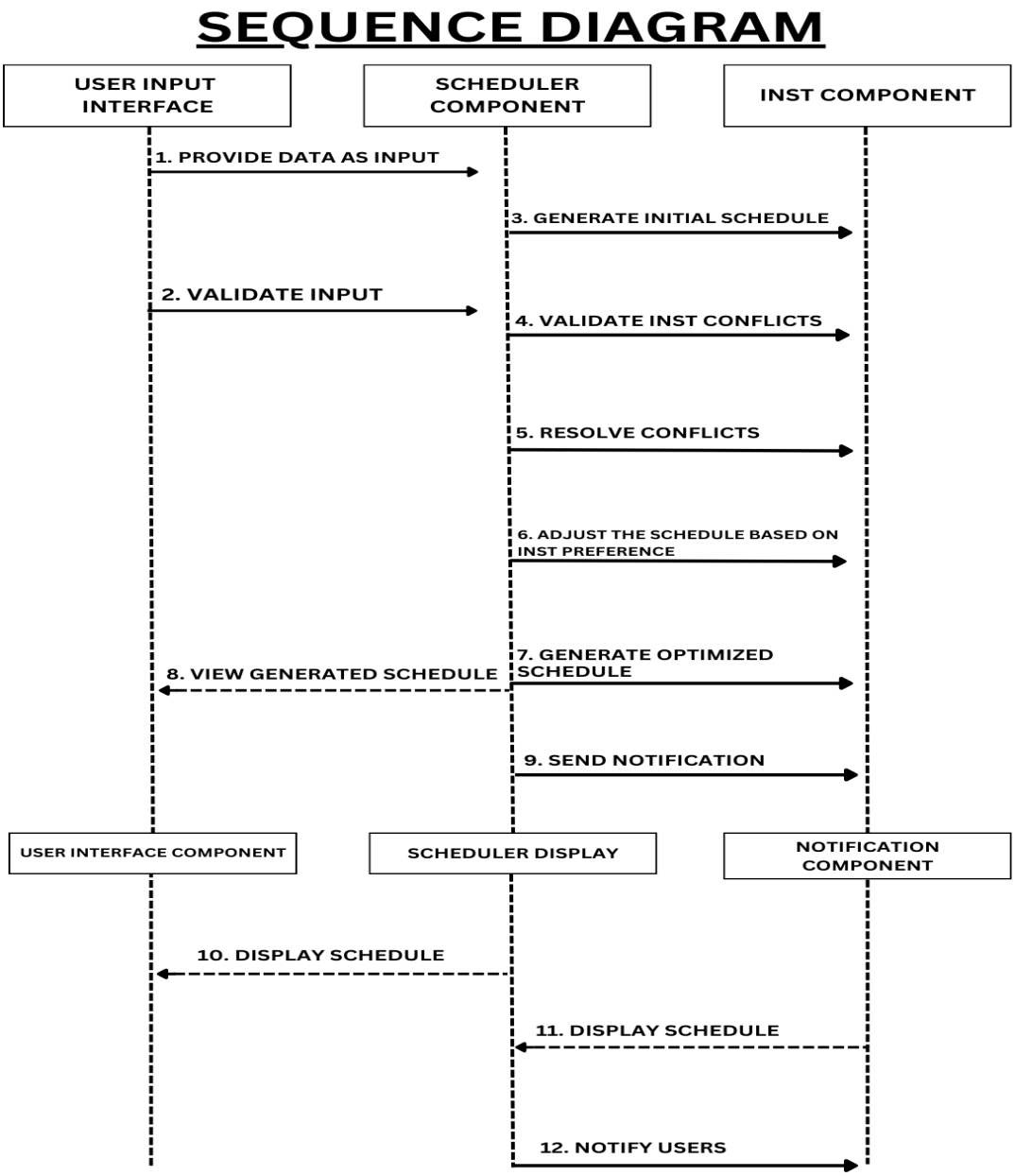
Course and Schedule:

Relationship: Many to Many

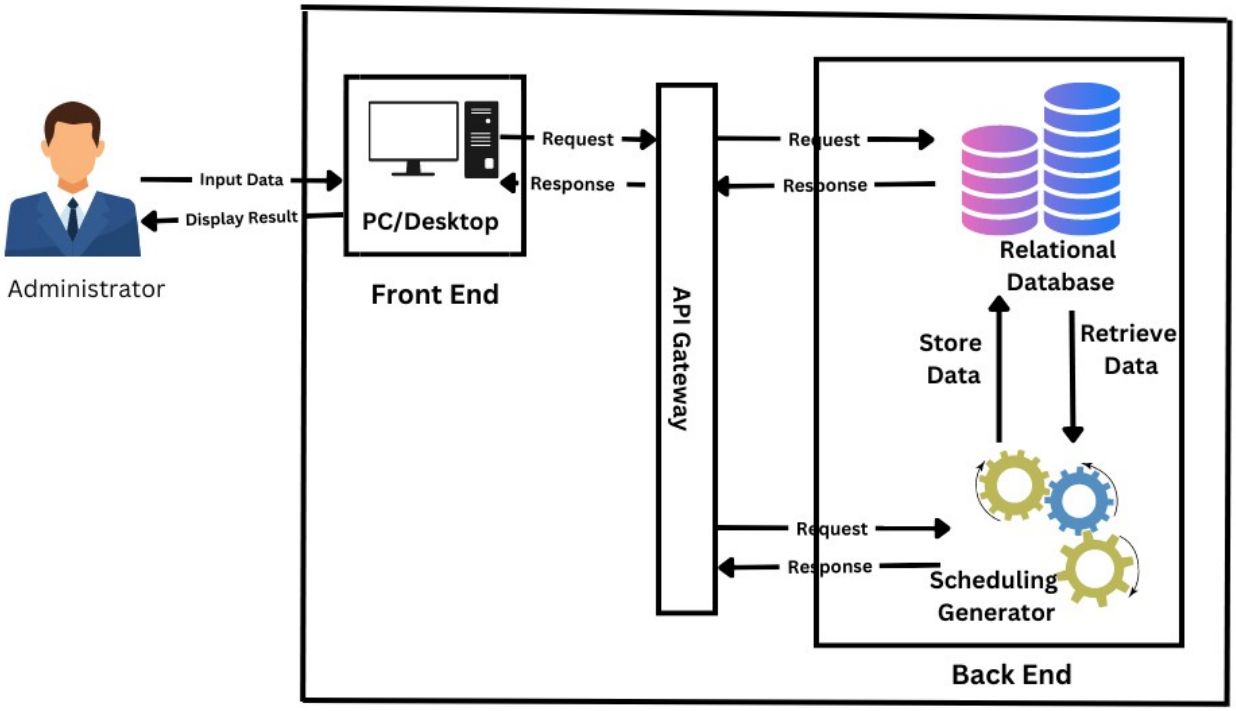
One Course can be part of Multiple Schedules, and a Schedule can include Multiple Courses. The relationship between "Course" and "Schedule" is a many-to-many relationship facilitated by an intermediate table called "Course_Schedule."

3rd Iteration of Design:

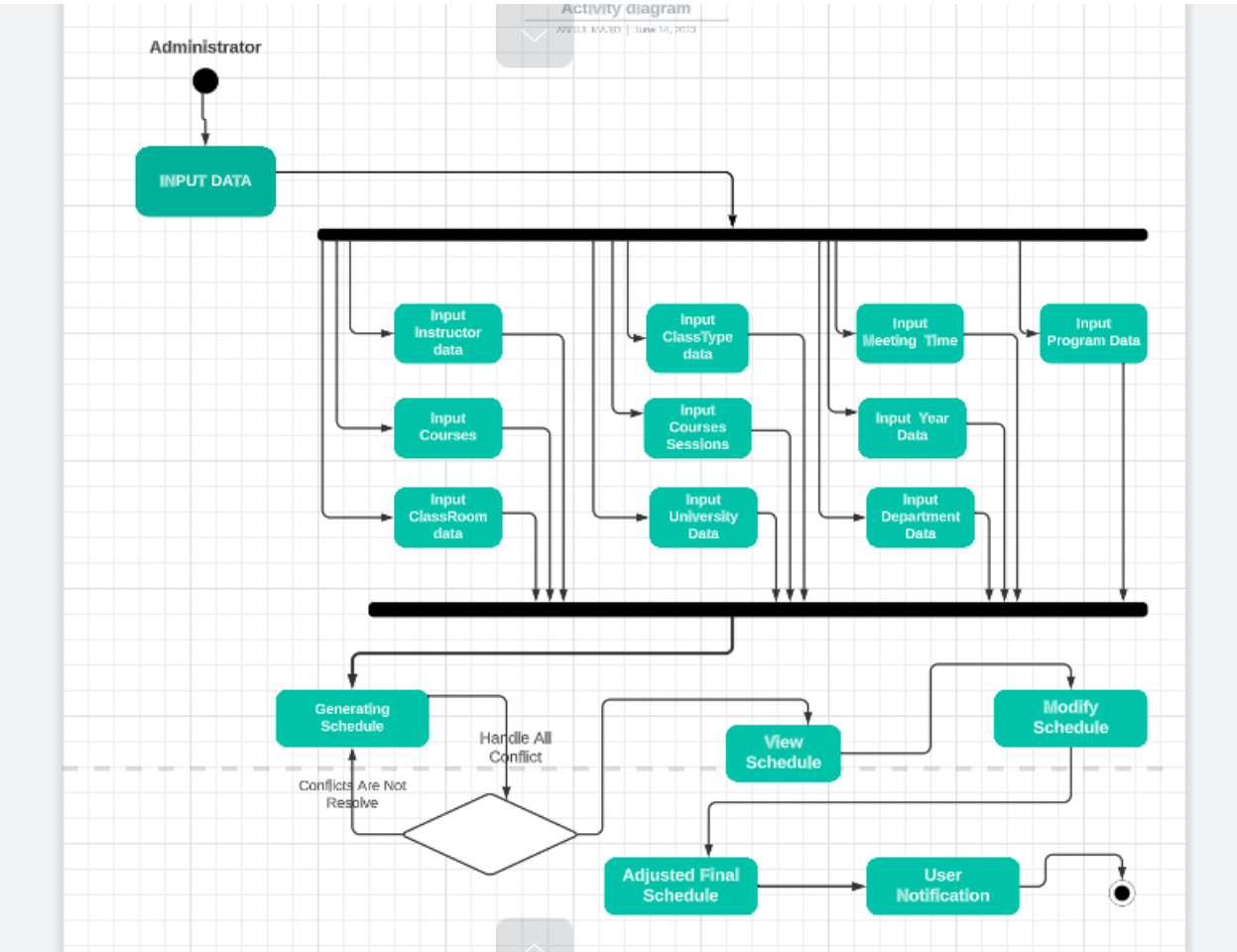
Sequence Diagram:



System Architecture Diagram:



Activity Diagram:



Formulate Chromosome:

PHENOTYPE TO GENOTYPE; Date

How to make a chromosome?

For Example we have following requirements.

Course	= 50	⟶	6 bits
Instructor	= 15	⟶	4 bits
Sessions	= 90	⟶	7 bits
Classroom	= 20	⟶	5 bits
Classroom Type	= 2	⟶	1 bits
Days	= 6	⟶	3 bits
Slots	= 4	⟶	2 bits

$2^0 = 1$
$2^1 = 2$
$2^2 = 4$
$2^3 = 8$
$2^4 = 16$
$2^5 = 32$
$2^6 = 64$
$2^7 = 128$

So, we Assign a unique binary string to each instructor. For Example, Dr. Aanj can be represented by "0000", Sir Ishan can be represented by "0001" and so on.

Same with Courses, Session, classroom, classroom type, Days and Slot that we also assign unique binary string to each of them.

Course	Instructor	Sessions	C.R	C.R Type	Days	Slot	
Gene 1	Gene 2	Gene 3	Gene 4	Gene 5	Gene 6	Gene 7	
000000	0000	0000000	00000	0	000	00	Chromosome
000001	0001	0000001	00001	1	001	01	Chromosome
100110	0101	0001110	10110	1	101	10	Chromosome
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
111111	1111	1111111	11111	1	111	11	Chromosome

Signature UNIQUE No.

Date

SAMPLE TABLE

		Monday	Tuesday	wednesday	Thursday
Room No 101	S1	0000000000	0000010000	0000010111	0000000001
	S2	0000010001	0000110001	0000110000	0000110111
	S3	0000110011	0001110011	0001110001	0001110000
	S4	0001110111	0000000111	0000000011	0000010011

For Example, 6 bits in table represent Course and 4 bits represent Instructor.

So we Suppose "Dr. Aarij "0000" ", "Sir Israr "0001" ", "Dr. Aliya "0011" and "Dr. Mansoor "0111" ". and four Courses we Suppose "000000 as FYP-1", "000001 as AI", "000011 as DAA" and "000111 as OS".

So, By this information now in the upper given table we known that in Room 101 on monday Dr. Aarij will teaches FYP-1 on first Slot.

Same with the other day's and rooms as well that by binary String we will know which teaher teaching which Course on that day in which classroom and also which slot.

Each Chromosome represents a potential Solution or Schedule. In our project "Automatic Course Scheduler", the Chromosome can be Composed of binary String representing the entities Such as Instructor, Course, Session, Classroom, C.V type and days.

It will randomly generate binary Strings for each entity and Combine them to form Chromosome.

Signature _____ UNIQUE No. _____

Explanation:

In an automatic course scheduling problem, we need to convert the entities from their original form (phenotype) into a suitable representation for the genetic algorithm (genotype). Here's how we can do that:

1. Phenotype to Genotype Conversion:

Identify the entities: We have several entities, such as courses, instructors, sessions, classrooms, classroom types, days, and slots.

Assign binary strings: We assign a unique binary string to each entity. To ensure uniqueness, we determine the number of bits required based on the total number of unique values for each entity.

For example, if we have 50 courses, we can assign 6 bits (2^6 = 64 unique combinations) to each course to uniquely identify it using a binary string.

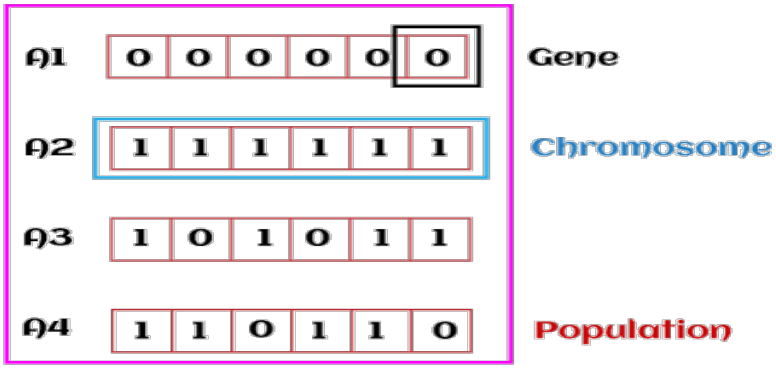
2. Chromosome Formation:

Combine the entities: To create a chromosome representing a potential schedule, we combine the binary strings for each entity.

For example, a chromosome can be formed by combining the binary strings representing a course, instructor, session, classroom, classroom type, day, and slot.

3. Random Chromosome Generation:

Initialize the population: We start by creating an initial population of chromosomes. To do this, we randomly generate combinations of binary strings for each entity.



4. Genetic Algorithm Optimization:

Fitness evaluation: We define a fitness function to assess the quality of each chromosome/schedule. The fitness function takes into account constraints like avoiding conflicts, maximizing room utilization, and minimizing gaps.

Selection: Based on their fitness scores, we select the fittest chromosomes from the population for reproduction. This selection process ensures that the better schedules have a higher chance of being chosen.

Crossover: We perform crossover between selected parent chromosomes. Crossover involves exchanging genetic material (binary strings) between parents, leading to the creation of new chromosomes that inherit characteristics from both parents. There are three type of crossovers.

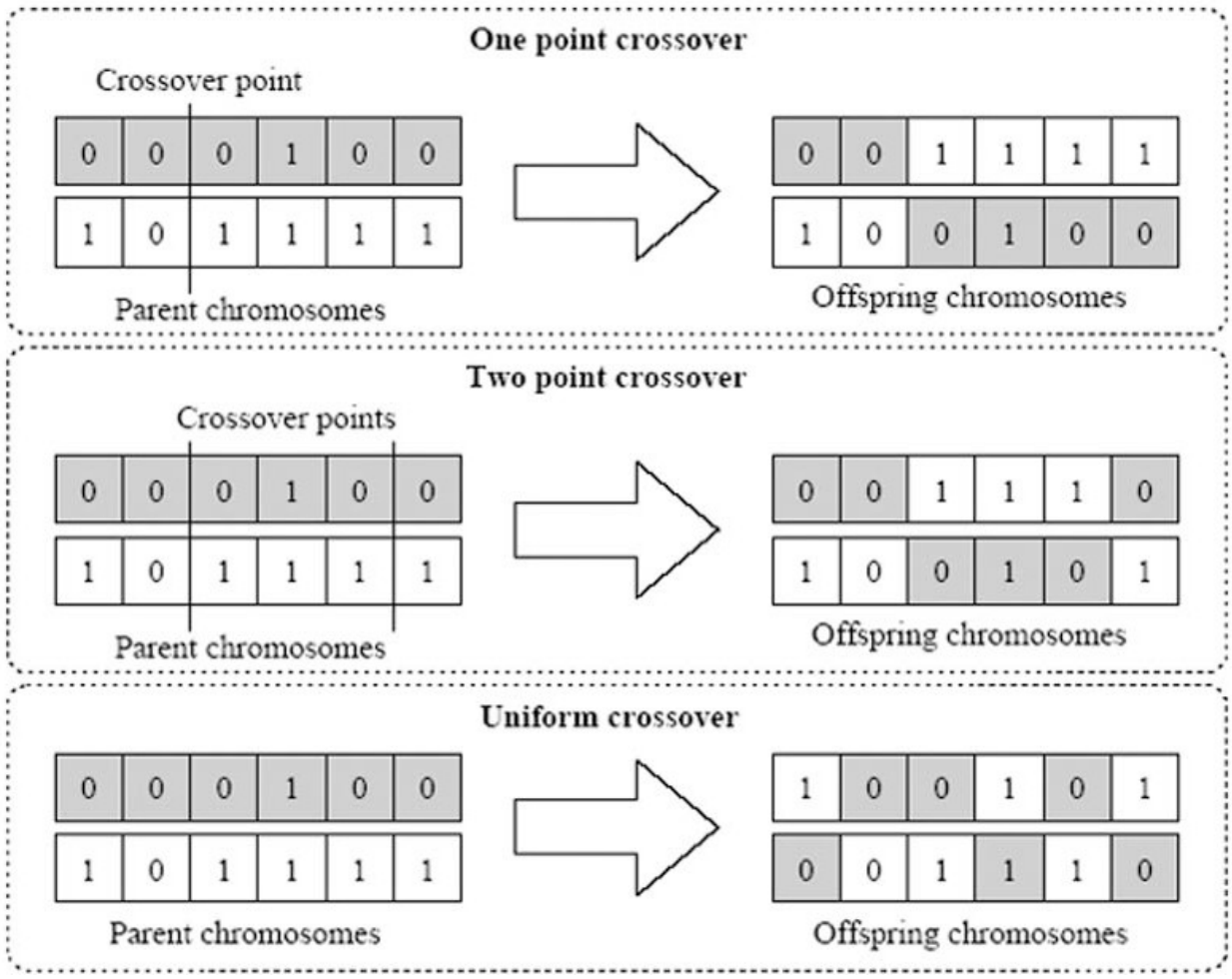


Figure 1 Crossovers functions

Mutation: To introduce diversity and prevent the population from converging too quickly, we randomly alter the binary strings within chromosomes. This random alteration is known as mutation and helps explore new solutions.

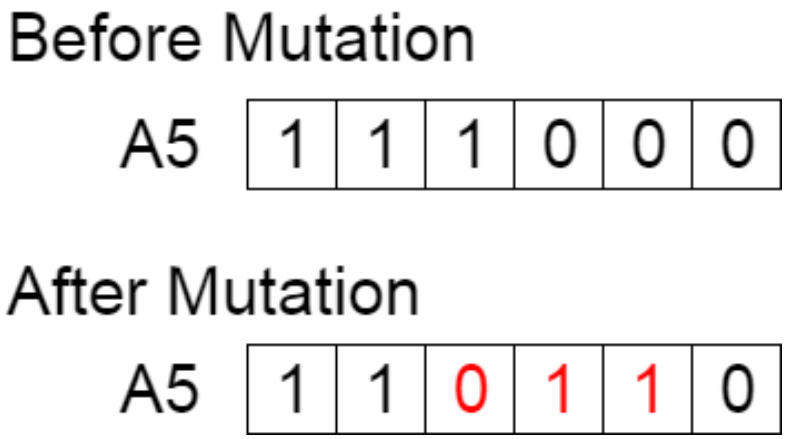


Figure 2 Mutation functions

Iteration and optimization: We repeat the selection, crossover, and mutation steps for multiple generations. Each iteration creates a new population of chromosomes, gradually improving the schedules based on the fitness function.