Time Table Automation And Integration with MRBS

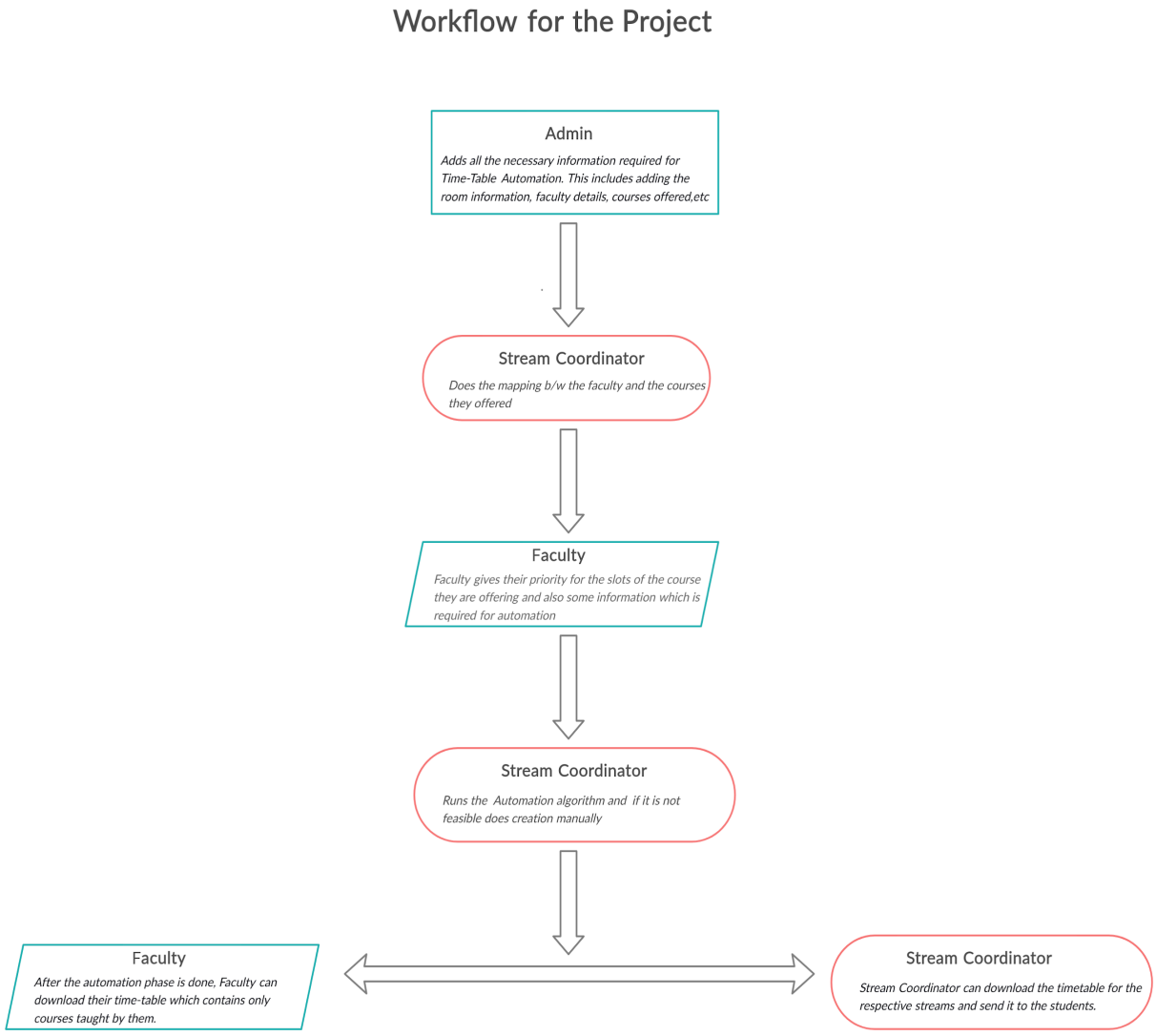
Data Requirements:

1. List of All the Faculty (FacultyID, Name, Department, Email, Designation, .. etc)
2. List of All courses. (Credits, CourseID, Course Type, [eligible departments and the year], ….etc:)
3. Courses offered/taught by the respective Faculty that year (CourseID, FaultyID).
4. All Class Rooms Information (Room Size, Room Number, Room Location (Transit/Ahalia), Room Capacity…. etc)
5. Slot Systems.

Roles:

- **Admin** (Academic section will act as admin and he will have full access to the entire system).
    - The admin will add all the Faculty details, the Courses, Class Rooms Information.
    - As of now to avoid clashes he will only fix the slots for common courses (like GCE, basically the courses which are offered across the disciplines and years)
    - For each stream, the admin will make one of the faculty as stream coordinator who will be involved in making the time table and also the does mapping b/w the faculty and the courses.(For example **UGCS5** (undergraduate Computer Science 5th semester) will be assigned a stream coordinator he/she will be in charge of generating the time table for that stream).
- **Faculty (Each faulty will can give their preferences to the courses)**
    - As faculty teach the courses they are allowed to give their preferences for the courses they are teaching.
    - And once the time table generation is done then they can download their time table.
- **Stream Coordinator (For each stream there will be a faculty in charge who will act as the stream in charge)**
    - The faculty in charge will be running the algorithm for the stream assigned to him and if any glitches occur he may resolve the problem manually.
    - As the courses are taught by the faculty, the admin will add the courses that are offered by the faculty.(Mapping b/w faculties and courses)

Workflow of the project:



- The admin will add all the data required for the time table generation.(courses, faculty, rooms, slot system ..etc)
- The Admin will make some of the faculty as Stream Coordinators and admin will create slot systems and admin will block some slots/assign the slots for the common courses.
- The faculty will give their preference for the courses they are teaching.
- Once the above steps are done then the stream coordinator will run the algorithm for time table generation and he can resolve the issues manually if any occur.
- Now the faculty's and streamwise time table can be downloaded.

- API

//post request, status
Sample post request

Path: api/course
request
{
	Cmd: get_courses
}
Reponses
{
	Status: true/false
	Msg: string
	Data: array of objects.// each object is a course.
}

Path: api/course
Request
{
	Cmd: modify_course
	Data: Course object
}
Reponses
{
	Status: true/false
	Msg: string //status
}

PS: The stated APIs are the main apis, there may be other apis used for other purposes as the project goes on.
Response msg
Status: true/false
Msg: information like error/status

ENDPOINT: **Courses**

END POINT	Request (JSONType) Sample API requests	Response (JSONType) Sample API responses	Description

api/courses/	[{
	Action: POST,
	CourseID: CS4010,
	Credits: 3,
	Name: Algebra,
	CourseType: PME,
	..
	}]
		{
		status: 200 OK
		msg {}
		}
			Will add the courses

api/courses/courseID	{
	Action: GET
	CourseID: CS4010
	}
		{ Status : 200 OK
		msg{
		CourseID: CS4010
		Credits: 3
		Name: Algebra
		CourseType: PME
		}
		}
			Will get info of the particular course

api/courses/courseID	{
	Action: Delete
	ID : CS4010
	}
		{
		Status: 200 OK
		}
			Will delete the course/ will also provide option to delete multiple also

api/courses/courseID	{
	Action: PUT
	CourseID: CS4010
	Credits: 3
	Name: Algebra
	CourseType: PME,
	...
	}
		{
		Status: 200 OK
		}
			Will update that particular course info.

ENDPOINT: Faculty

END POINT	Request (JSONType)	Response (JSONType)	Description

api/faculty	{
	Action: POST,
	facultyID: SS,
	name: jason crouse,
	dept : CS,
	mail: jason@gmail.com
	}
		{ status: 200 ok
		msg:{
		}
		}
			Will add the faculty info and will also provide api to add multiple entries too

api/faculty/facultyID	{
	Action : GET
	ID: SS
	}
		{ status: 200 ok
		msg{
		facultyID: SS,
		name: jason crouse,
		dept : CS,
		mail: jason@gmail.com
		}
		}
			Will get the information regarding the faculty with facultyID

api/course/courseID	{
	Action : Delete
	ID: SS
	}
		{
		status : OK
		}
			Will delete the faculty information.

api/course/courseID	{
	Action:PUT,
	facultyID: SS,
	name: jason crouse,
	dept : CS,
	mail: jason@gmail.com
	}
		{
		status : OK
		}
			Will update the faculty info and will also provide api to add multiple entries too

ENDPOINT: Rooms

END POINT	Request (JSONType)	Response (JSONType)	Description

api/rooms	{
	Action: POST,
	roomID: 123,
	capacity: 50,
	location : "Transit"
	}
		{ status: 200 ok
		msg : {}
		}
			Will add the rooms info and will also provide api to add multiple entries too

api/rooms/roomID	{
	Action: GET
	roomID : 123
	}
		{ status: 200 ok
		msg{
		roomID: 123,
		capacity: 50,
		location : "Transit"
		}
		}
			Will get info of the particular room

api/rooms/roomID	{
	Action: Delete
	roomID : 123
	}
		{ status: 200 ok
		msg : {}
		}
			Will delete the particular room

api/rooms	{
	Action: PUT,
	roomID: 123,
	capacity: 50,
	location : "Transit"
	}
		{ status: 200 ok
		msg : {}
		}
			Will update that particular rooms info.

ENDPOINT: CouseFaculty

END POINT	Request (JSONType)	Response (JSONType)	Description

api/courseFaculty	{
	Action : POST
	courseID: CS4010
	facultyID : SS
	}
		{ status: 200 ok
		msg : {}
		}
			Will add the courseFaculty info and will also provide api to add multiple entries too

api/courseFaculty/courseID	{
	Action: GET
	roomID : 123
	}
		{ status: 200 ok
		msg{
		roomID: 123,
		capacity: 50,
		location : "Transit"
		}
		}
			Will get info of the particular room

Slot System.

The slot system should be fixed initially. And each slot is for 1-hr.
As general the slot should accommodate the three credit courses and 4 credit courses and labs.

Labs Timings can be from 9-12 or 2-5 (ideally).
So whenever the Admin is making a slot he will choose whether it's 3 credit course or 4-credit course or it's lab course and depending upon the slot he has to fix 3 slots for 3credits course and lab course, 4 slots for 4 credit course.

Example Slot System.

For example The slot A is for 3 credit courses and slot B is 4 credit courses and lab1 is course.

Day	8 - 9	9 - 10	10 - 11	11 - 12	12 - 13	14 - 15	15 - 16	16 - 17
Monday	A					Lab1	Lab1	Lab1
Tuesday	B	A						
WednesDay		B	A					
ThursDay			B					
Friday				B				

EndPoint: slot

{slotno, day, timings} will uniquely determine the slot. So we have only get and post operation on that.

END POINT	Request (JSONType)	Response (JSONType)	Description

api/slot	{
	Action : POST
	SlotNumber: 1
	Day: Monday
	timings: 8:00 - 9:00
	}
		{ status: 200 ok
		msg : {}
		}
			Will add the corresponding slot into the slot System.

api/slot	{
	Action: GET
	SlotNumber: 1
	}
		{ status: 200 ok
		msg{
		Will contain all the details related to the slot number.
		}
		}
			Will get info of the particular slot system

Database Schema

//wherever possible use integers instead of strings.

Faculty.{
name: string
stream: string
facultyID: string // which will determine the faculty uniquely
mail: string
isStreamCordinator: boolean.
}

Course.
{
courseID: string
name: string
Type: string (restricted to PME, GCE , .. etc)
lecture: int
tutorial:int
lab: int
strength: int
Credit: int
Students: Array of String(Ex: [cse_s5, cse_s6, … etc])
facultyID: Id to faculty.
}

Students
{
Name: String
RollNo: String
Semester: String
Branch: String (select option)

ListOfSubjects: Array of CourseID's.
ListOfBacklogCourses: Array of CourseID's.
GroupID: string.
}

//representative element.
Group
{
GroupName: String
ListOfSubjects: Array of CourseID's
Strength: int
}

ROOM
{
capacity: int
roomID: string // typically room number can be an roomID
Location: LocationID// to represent whether it is an ahalia or Transit or MBA.
}
Location
{
Name: String
TravelTimes: Array of (int, locationID) // Distances to other locations
}
SlotSytem
{
Name: string
ID: string
Array of SlotSchema IDS
}
SlotSchema
{
ID: int // A, B,C
Array of (Day, Start_time, End_time) // start_time , end_time represent in seconds (integers).
}

// not required.
SlotSchema
{

ID: int // which will tell the slot number
Day: string // which will represent the day ex: monday, tuesday
Timings: Date/ String // for example it can be 8:00-9:00 AM or 9:00-10:00AM
Slot: string // which slot is there i.e (whether A, B, P1, ..etc)
Type: string // lab or lecture
}

streamCoordinator
{
Stream: string // the name of the stream
FacultyID: string // the faculty incharge ID
}