Academic Section Management System

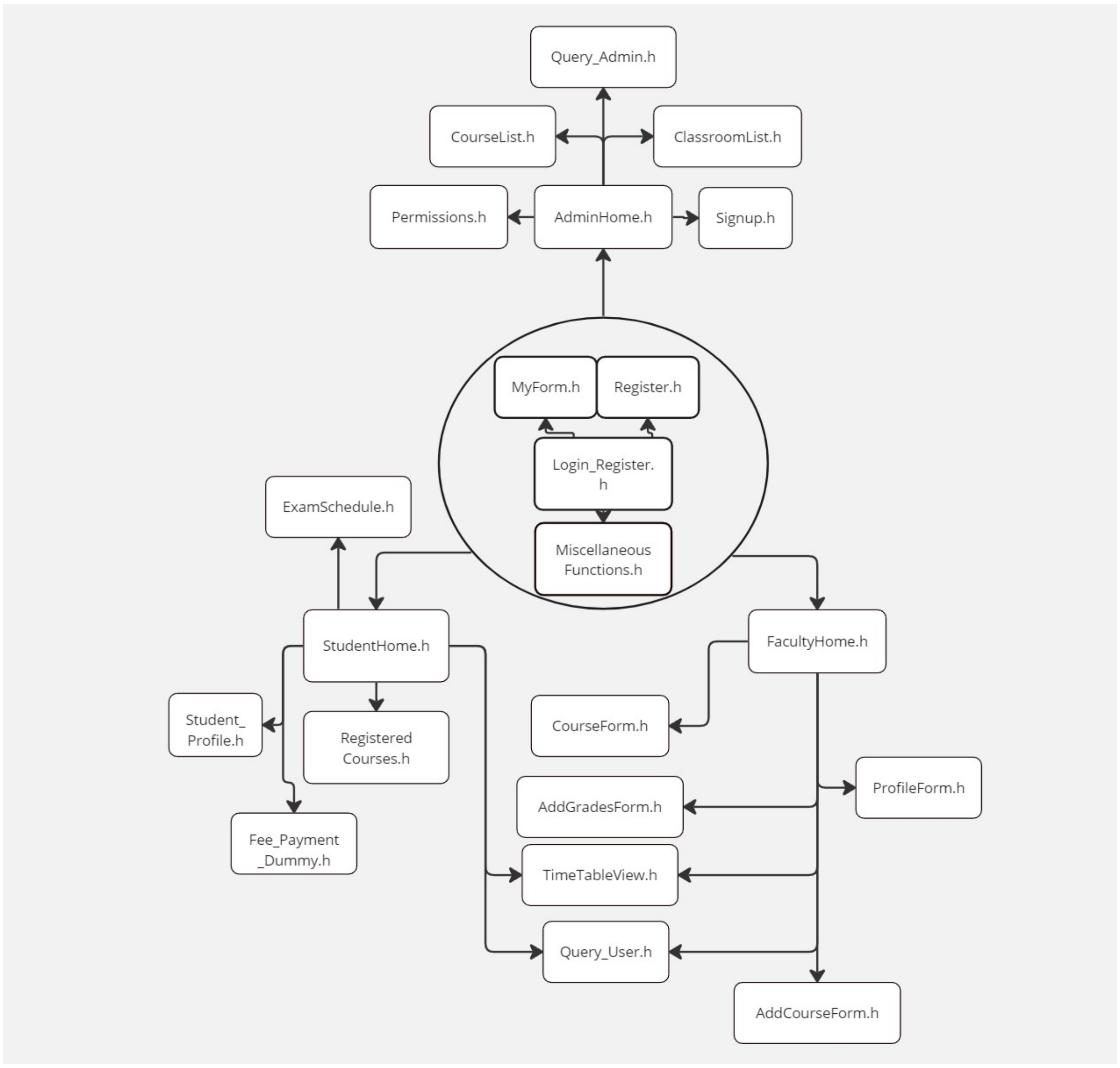
CS346 - Software Engineering Lab Assignment 2 - Group 10B

Technical Documentation

Introduction

This comprehensive document serves as a guide for developers, administrators, and users involved in the implementation and maintenance of Academic Section Management System Software. In this documentation, you will find detailed information about the architecture, functionalities, procedures and database schemas involved in the software. The Academic Section Management System follows a client-server architecture, utilizing Visual C++ Windows Form for the frontend and Azure SQL for the backend.

System Architecture



Database Schema

TABLE_NAME	COLUMN_NAME	DATA_TYPE	max_length	is_nullable	is_identity
auth	UserID	int	4	False	False
auth	Password_hash	char	32	False	False
auth	UserType	nvarchar	100	False	False
auth	UserType	sysname	100	False	False
auth	Role	nvarchar	100	False	False
auth	Role	sysname	100	False	False
auth	Email	varchar	32	False	False
classroom	Room_ID	nvarchar	100	False	False
classroom	Room_ID	sysname	100	False	False
classroom	Capacity	int	4	False	False
classroom	Room_type	nvarchar	100	False	False
classroom	Room_type	sysname	100	False	False
Complaints	Email	varchar	50	True	False
Complaints	Complaint	varchar	500	True	False
Complaints	Status	varchar	20	True	False
Complaints	Comments	varchar	100	True	False
Complaints	complaint_id	int	4	False	False
course	Course_Code	char	5	False	False
course	Name	varchar	50	False	False
course	Description	varchar	200	True	False
course	L	int	4	False	False
course	T	int	4	False	False
course	P	int	4	False	False
course	C	int	4	False	False
course	Faculty_ID	int	4	True	False
course	Intake	int	4	True	False
course	Semester	varchar	50	False	False
course	ElectiveOrCompulsory	bit	1	False	False

TABLE_NAME	COLUMN_NAME	DATA_TYPE	max_length	is_nullable	is_identity
exam	Course_ID	char	5	False	False
exam	Date	date	3	False	False
exam	Time_Slot	varchar	20	False	False
exam	Room_ID	varchar	10	False	False
exam	Student_ID	int	4	False	False
exam	ExamType	bit	1	False	False
faculty	User_ID	varchar	10	False	False
faculty	Name	varchar	50	False	False
faculty	DOB	date	3	True	False
faculty	Contact	varchar	20	False	False
faculty	Email	varchar	100	False	False
faculty	Office_Room	varchar	5	True	False
faculty	Joining_Year	int	4	False	False
faculty	Research_Interests	varchar	268	True	False
faculty	Designation	varchar	50	False	False
Fee_Payment	User_ID	int	4	False	False
Fee_Payment	Fee_Amount	int	4	False	False
Fee_Payment	Paid_on	datetime	8	False	False
Fee_Payment	Semester	int	4	False	False
grade	User_ID	int	4	False	False
grade	CourseCode	char	5	False	False
grade	Grade	char	2	True	False
grade	Approval_Status	nvarchar	40	False	False
grade	Approval_Status	sysname	40	False	False
grade	Type	nvarchar	20	False	False
grade	Type	sysname	20	False	False
grade	Year	int	4	True	False
Lecture_Slots	slot	varchar	5	True	False
Lecture_Slots	weekday	varchar	10	True	False
Lecture_Slots	time_slot	varchar	20	True	False

TABLE_NAME	COLUMN_NAME	DATA_TYPE	max_length	is_nullable	is_identity
permissions	Course_Add	bit	1	False	False
permissions	Course_Enroll	bit	1	False	False
permissions	Grades_Add	bit	1	False	False
permissions	Grades_View	bit	1	False	False
permissions	Current_Year	int	4	False	False
permissions	Current_Sem	nvarchar	40	False	False
permissions	Current_Sem	sysname	40	False	False
permissions	Midsem_Start_Date	date	3	True	False
permissions	Endsem_Start_Date	date	3	True	False
Profile_photos	User_ID	int	4	False	False
Profile_photos	Photo	varbinary	-1	True	False
Sem_Fee	Semester	int	4	False	False
Sem_Fee	Fee_Amount	int	4	False	False
signup	Name	varchar	50	False	False
signup	DOB	date	3	True	False
signup	Contact	char	10	False	False
signup	Address	varchar	200	False	False
signup	Email	char	100	False	False
signup	Approval_status	nvarchar	40	False	False
signup	Approval_status	sysname	40	False	False
signup	UserType	nvarchar	40	False	False
signup	UserType	sysname	40	False	False
signup	Password_hash	char	32	False	False
signup	enrollment_year	varchar	4	True	False
student	User_ID	int	4	False	False
student	Name	varchar	50	False	False
student	DOB	date	3	False	False
student	Contact	char	10	False	False
student	Enrollment_Year	int	4	False	False
student	Address	varchar	200	False	False

TABLE_NAME	COLUMN_NAME	DATA_TYPE	max_length	is_nullable	is_identity
student	Current_Semester	int	4	False	False
student	Email	nvarchar	100	False	False
student	Email	sysname	100	False	False
timetable	Course_ID	nvarchar	20	True	False
timetable	Course_ID	sysname	20	True	False
timetable	Room_ID	nvarchar	510	True	False
timetable	Room_ID	sysname	510	True	False
timetable	Slot	varchar	10	False	False

Development Environment

- Platform: Visual Studio Ultimate 2013
- Frontend : Visual C++ Windows Form
- Backend : Azure SQL

Code Structure

Student

Profile page (Student_Profile.h):

1. Get Profile:

- **Function invoked**: `get_profile`
- **Description**: Populates student data into Student class
- **Input**: User instance
- **Output**: User details loaded in the form
- **Effect**: Student data loaded from database table(student)
- **SQL Query**

```
SELECT * FROM student where User_ID = @user_ID_fetch
```

2. Get Profile Photo

- **Function invoked**: `profile_photo_fetch`
- **Description**: Populates student data into Student class
- **Input**: User instance
- **Output**: Profile Photo loaded
- **Effect**: Photo retrieved from database(profile_photos) or default photo loaded into Photo(pictureBox3)

- **SQL Query**

```
SELECT Photo FROM profile_photos WHERE User_ID = @UserID
```

3. Update Profile:

- **Function invoked**: `button1_Click` which further invokes `get_profile` and `update_profile`
- **Description**: Updates the the student's data in from the form displayed in profile page to the database
- **Input**: Updated profile information date of birth, address, contact from the textfields, password(if entered)
- **Output**: Updated successfully or not pops up
- **Effect**: Updates the user's profile information in the database(student and auth(if password changed))
- **SQL Query**:

```
UPDATE student SET DOB = @DOB, Contact = @Contact, Address = @Address WHERE
User_ID = @User_ID
UPDATE auth SET Password_Hash = @Password WHERE UserID = @User_ID
```

4. Upload Photo

- **Function invoked**: `button2_Click` which further invokes `UploadImageToDatabase`
- **Description**: Updates the student's profile photo
- **Input**: Input is given by choosing an image file from the file dialog box. The following types can be taken as input: `.bmp;.jpg;.jpeg;.gif;.png;.tif.`
- **Output**: Updated successfully or not pops up.
- **Effect**: Updates or inserts(If not present) the user's profile photo in the database(profile_photos)
- **SQL Query**:

```
INSERT INTO profile_photos values(@UserID, @Photo)
UPDATE profile_photos SET Photo = @Photo WHERE User_ID = @UserID
```

Registered Courses(Registered Courses.h)

1. Fetch Registered courses

- **Function invoked**: `fill_Registered_table`
- **Description**: Updates the table of registered courses
- **Input**: Instance of the student user class
- **Output**: Registered courses table filled with query result
- **Effect**: Fetches the data from the grade table to get status, and grades also, then joins with the course table to get the name and other details of the course
- **SQL Query**:

```
SELECT [CourseCode],[Name],[L],[T],[P],[C],[Semester],[Type],[Approval_Status],
[Grade] FROM [dbo].[grade],[dbo].[course] where User_ID = @user_ID_fetch and
CourseCode = Course_Code
```

2. Open Course Enrollment page

- **Function invoked**: `button1_Click` which depends on `get_reg_status`
- **Description**: Opens the page for enrolling in courses
- **Input**: Value of the reg_status from the permissions table set by the admin
- **Output**: Opens the course enroll page if allowed other wise no procedure/action taken
- **Effect**: Fetches the reg_status from the permissions table and then changes behviour by showing "Closed" or "Open" registration.
- **SQL Query**:

```
SELECT top(1) Course_Enroll FROM permissions
```

Course Enroll Page(Course_EnrollPage.h)

1. Display Compulsory Courses:

- **Function invoked**: `disp_comp_courses`
- **Description**: Displays the list of compulsory courses that the student must register for in that semester.
- **Input**: The current semester of the student
- **Output**: The page displays the list of compulsory courses along with an option to view the details of each course and to register/de-register.
- **Effect**: None
- **SQL Query**:

```
SELECT [L],[T],[P],[C],[Course_Code],[Name],[Slot] FROM [dbo].[course_Structure],
[dbo].[timetable] where [Semester] = @sem and [Elective] = 0 and [Course_Code] =
[Course_Id];
```

2. Display Audit Courses:

- **Function invoked**: `display_audit_courses`
- **Description**: Displays the list of audit courses that the student can register for in that semester.
- **Input**: None
- **Output**: The page displays the list of audit courses along with an option to view the details of each course and to register/de-register.
- **Effect**: None
- **SQL Query**:

```
SELECT [Course_ID],[Name]FROM[dbo].[course], [dbo].[timetable] WHERE [Course_ID] =
[Course_Code];
```

3. Display Department Elective Courses:

- **Function invoked**: `Add_DE`
- **Description**: Displays the list of department electives that student can register for in that semester.
- **Input**: The current semester of the student, the number of DE
- **Output**: The page displays the list of department electives available along with an option to view the details of each course and to register/de-register.
- **Effect**: None
- **SQL Query**:

```
SELECT [Course_Code],[Name] FROM [dbo].[course]WHERE [Course_Code] LIKE '%CS%' AND
[ElectiveOrCompulsory] = 1 AND [Semester] LIKE @sem;
```

4. Display Open Elective Courses:

- **Function invoked**: `Add_OE`
- **Description**: Displays the list of open electives that student can register for in that semester.
- **Input**: The current semester of the student, the number of OE
- **Output**: The page displays the list of open electives available along with an option to view the details of each course and to register/de-register.
- **Effect**: None
- **SQL Query**:

```
SELECT [Course_Code],[Name] FROM [dbo].[course]WHERE [Course_Code] LIKE '%OE%' AND
[ElectiveOrCompulsory] = 1 AND [Semester] LIKE @sem;
```

5. View Course Description

- **Function invoked**: `DescButtonClicked`
- **Description**: Displays information regarding the selected course.
- **Input**: The course code of the course whose information is required
- **Output**: A message box is displayed with the the course description
- **Effect**: None
- **SQL Query**:

```
SELECT[Description] FROM[dbo].[course] WHERE[Course_Code] = @code
```

6. Register/De-Register from a Course

- **Function invoked**: `RegButtonClicked` which invokes `check_slot_clash`
- **Description**: Allows user to register/de-register when they click on the button corresponding to the course. It first checks for any slot clashes before registering.
- **Input**: Index of the row and column of the course selected in the DataGridView, Instance of the DataGridView, Course code, Time slot of the course, Flag bit for audit course.
- **Output**: A message box is displayed reporting either a succesful registration/de-registration or an error message regarding a slot clash.
- **Effect**: Updates the grade database by inserting/removing an entry containing the user_id and course_id.
- **SQL Query**:

```
INSERT INTO [dbo].[grade] VALUES (@id , @code, 'NA', 'Pending' , 'Audit',@year);
INSERT INTO [dbo].[grade] VALUES (@id , @code, 'NA', 'Pending' , 'Credit', @year);
DELETE FROM [dbo].[grade] WHERE [User_ID] =@id  and [CourseCode] = @code";
```

7. Check Course Slot Clash in the Timetable

- **Function invoked**: `check_slot_clash`
- **Description**: Checks if the slot of the course the user would like to register for has any clash with the slots of the already registered courses
- **Input**: Time slot of the course
- **Output**: Returns true if slot clash exists and false otherwise.
- **Effect**: None
- **SQL Query**:

```
SELECT[Course_ID], [Slot] FROM [dbo].[grade], [dbo].[timetable] WHERE[Course_ID] =
[CourseCode] AND[User_ID] = @id AND [Slot] = @slot;
```

Fee Payment Page

1. Get Fee Details

- **Function invoked**: `get_fee_data`
- **Description**: Gets the fee details for the Student
- **Input**: Instance of Student class
- **Output**: Fills fee details data in the respective fields
- **Effect**: Checks fee details table(Fee_Payment) to check if user has already paid. If not then get fee details from the Sem_Fee table. If not paid, then Pay fee button gets activated. Otherwise show succesfull payment message.
- **SQL Query**:

```
SELECT * FROM [dbo].[Sem_Fee] where Semester = @Semes
SELECT * FROM [dbo].[Fee_Payment] where User_ID = @user_ID_fetch and Semester =
@Semes
```

Time Table Page(TimeTableView.h)

- **Function invoked:** `label6_Click`
- **Description:** The page for viewing the time table shows up
- **Input:** The day for which the user wants to view the time table is choosen from the combo box.
- **Output:** In a list view, the time table for the day choosen comes up in the form course name, slot,time and venue.
- **Effect:** Time table fetched from database.
- **SQL Query:**

```
select Course_ID,Name,timetable.Room_ID,timetable.Slot,time_slot,Room_type from
course INNER JOIN ((timetable INNER JOIN Lecture_Slots ON timetable.Slot =
Lecture_Slots.slot) INNER JOIN classroom ON classroom.Room_ID = timetable.Room_ID)
ON Course_Code = Course_ID where Course_ID in (select CourseCode from grade where
User_ID = @user_ID_fetch) and weekday = @day
```

Exam table Page

1. Get Exam Seating Details

- **Function invoked**: `get_examschedule`
- **Description**: Gets the fee details for the Student
- **Input**: Instance of Student class and the type of exam midsem or endsem
- **Output**: Fills exam venue, date, time details in the list items
- **Effect**: Checks the Exam table in the database, joins with the course table and grade table, and for every course a student is enrolled in finds the exam details.
- **SQL Query**:

```
SELECT Course_ID,Name,Date,Time_Slot,exam.Room_ID,classroom.Room_type from ((exam
INNER JOIN course ON Course_ID = Course_Code) INNER JOIN classroom ON
classroom.Room_ID = exam.Room_ID) where Student_ID = @user_ID_fetch and ExamType =
@type= @Semes
```

Faculty

1. Profile page (ProfileForm.h):

Upload Photo:

- **Function invoked**: `btnPhoto_Click` which further invokes UploadImageToDatabase.
- **Description**: An open file dialog opens where one can choose a photo to upload
- **Input**: Input is given by choosing an image file from the file dialog box. The following types can be taken as input: `.bmp;.jpg;.jpeg;.gif;.png;.tif`.

- **Output**: The uploaded photo would come up as profile photo.
- **Effect**: The photo uploaded is stored in the online azure database
- **SQL Query**

```
UPDATE profile_photos SET Photo = @Photo WHERE User_ID = @UserID
```

Upload Profile:

- **Function invoked**: `btnProfile_Click`
- **Description**: Updates the user's profile information such as date of birth, contact, office room, designation, and research interests.
- **Input**: Updated profile information including date of birth, contact, office room, designation, and research interests.
- **Output**: Updated successfully or not pops up.
- **Effect**: Updates the user's profile information in the database.
- **SQL Query**:

```
UPDATE faculty SET DOB = @DOB, Contact = @Contact, Office_room = @Room,
Designation = @Designation, Research_Interests = @Research WHERE User_ID = @UserID
```

Fetch User Data:

- **Function invoked:** `FetchUserData`
- **Description:** Retrieves user data from the database and populates the profile form with the fetched data.
- **Input:** User ID
- **Output:** None
- **Effect:** Populates the profile form with user data fetched from the database.
- **SQL Query:**

```
SELECT Name, DOB, Contact, Email, Office_room, Joining_Year, Research_Interests,
Designation FROM faculty WHERE User_ID = @UserID
```

2. Add Courses/Electives (AddCourseForm.h):

- **Name to call in code:** `AddCourseToDatabase`
- **Description:** Adds the course details to the database after validating and checking for duplicate course codes.
- **Input:** `int userid` - User ID of the faculty adding the course.
- **Output:** None
- **Effect:** Checks if the course code already exists in the database. If not, inserts the course details into the database.
- **SQL Query:**

```
INSERT INTO course (Name, Description, L, T, P, C, Faculty_ID, Intake, Semester,
Course_Code, ElectiveorCompulsory)
VALUES (@Name, @Description, @L, @T, @P, @C, @FacultyID, @Intake, @Semester,
@CourseCode, @ElectiveorCompulsory);
```

Submit Course:

- **Name to call in code:** `button2_Click`
- **Description:** Event handler for the submit button click. Validates input data and adds the course to the database if all validations pass and calls `AddCourseToDatabase` function.
- **Input:** `System::Object^ sender, System::EventArgs^ e`
- **Output:** None
- **Effect:** Validates input data from text boxes such as course code, name, description, lecture hours (L), tutorial hours (T), practical hours (P), intake, and semester. If the data is valid, it adds the course to the database through function `AddCourseToDatabase`.
- **SQL Query:** None

3. View courses and approve students (CourseForm.h):

View Courses:

- **Name to call in code:** `CourseForm_Click`
- **Description:** Event handler for clicking on a course button.
- **Input:** `int _userid`
- **Output:** List of courses along with course details and approve/reject
- **Effect:** Retrieves course details and related grades from the database based on the course button clicked.
- **SQL Query:** Query 1: Retrieve course information along with grades and student names

```
SELECT c.Name AS CourseName, c.Description, c.L, c.T, c.P, c.C,
       c.Intake, c.Semester, c.Course_Code, g.User_ID, g.Grade,
       g.Approval_Status, s.Name AS StudentName
FROM course c LEFT JOIN
     grade g ON c.Course_Code = g.CourseCode LEFT JOIN
     student s ON g.User_ID = s.User_ID
WHERE c.Course_Code = courseCode;
```

Query 2: Retrieve course name and code for courses taught by a specific faculty

```
SELECT Name, Course_Code
FROM course
WHERE Faculty_ID = userid;
```

Approve Button Click:

- **Name to call in code:** `ApproveButton_Click`
- **Description:** Event handler for clicking the "Approve" button.
- **Input:** Checked items from listview
- **Output:** `Approved` is displayed
- **Effect:** Approves selected grades by updating the approval status in the database.
- **SQL Query:**

```sql
 --Updates the approval status in the grade table for selected grades.
UPDATE grade SET Approval_Status = 'Approved' WHERE User_ID = '" + item->Text + "'
AND CourseCode = '" + course_id + "'
```

Reject Button Click:

- **Name to call in code:** `RejectButton_Click`
- **Description:** Event handler for clicking the "Reject" button.
- **Input:** Checked items from listview
- **Output:** `Rejected` is displayed, deleting the item.
- **Effect:** Rejects selected grades by deleting the row.
- **SQL Query:**

```sql
DELETE FROM grade WHERE User_ID = '" + item->Text + "' AND CourseCode = '" +
course_id + "'
```

4. Grades page (AddGradesForm.h):

Add Grades:

- **Name to call in code:** `label5_Click`
- **Description:** Inserts grade status and grade of approved students into the database and faculty can modify grades.
- **Input:** Checked items from listview.
- **Output:** Grade of student updated.
- **Effect:** Inserts grades for students into the database.
- **SQL Query:**

```sql
UPDATE grade SET Grade = '" + newGrade + "' WHERE User_ID = '" + userId + "' AND
CourseCode = '" + course_id + "'
```

5. View Time Table(TimeTableView.h):

- **Function invoked:** `label6_Click`
- **Description:** The page for viewing the time table shows up
- **Input:** The day for which the user wants to view the time table is choosen from the combo box.

- **Output:** In a list view, the time table for the day choosen comes up in the form course name, slot,time and venue.
- **Effect:** Time table fetched from database.
- **SQL Query:**

```
select Course_ID,Name,timetable.Room_ID,timetable.Slot,time_slot,Room_type from
course INNER JOIN ((timetable INNER JOIN Lecture_Slots ON timetable.Slot =
Lecture_Slots.slot) INNER JOIN classroom ON classroom.Room_ID = timetable.Room_ID)
ON Course_Code = Course_ID where Course_ID in (select Course_Code from course
where  Faculty_ID = @user_ID_fetch) and weekday = @day
```

Admin

Signup Page (Signup.h)

1. Fetch All Students:

- **Description**: Populates all the students/faculty whose approval_status is pending, approval or rejected.
- **Effect**: Signup data loaded from signup database table
- **SQL Query**

```
select * from [dbo].[signup]
```

2. Reject Student/Faculty

- **Description**: Clicking on the 'Reject' button deletes the record of the student/faculty from the signup table.
- **SQL Query**

```
delete from [dbo].[signup] where Email= @Email
```

3. Approve Student/Faculty

- **Description**: Clicking on the 'Approve' button after checking in the box of a student/faculty approves him/her.
- **Effect**: Data added to Auth, Student/Faculty database.
- **SQL Query**

```
update [dbo].[signup] set Approval_status = '" + approve + "' where Email = @Email
insert into [dbo].[auth] values(@UserId,@PassHash,@UserType,@UserType,@Email)
insert into [dbo].[student]
```

```
values(@UserId,@Name,@dob,@contact,@enrollment_year,@address,@current_Sem,@email)
insert into [dbo].[faculty]
values(@UserId,@Name,@dob,@contact,@email,@room_num,@enrollment_year,@research_int
erest,@designation)
```

Permissions Page (permissions.h)

1. Fetch all permissions:

- **Description**: Populates the permissions required for the functioning of the academic system management such as Course_Add, Course_Enroll, Grades_Add, midsem_start_date, endsem_start_date, current year and current semester. This data will be used by other entities like student/faculty for smooth functioning.
- **Effect**: Permissions loaded from permissions database
- **SQL Query**

```
select * from [dbo].[permissions]
```

2. Enable/Disable permissions:

- **Description**: Used to toggle values present in the database of the following buttons:
- **Enable/Disable Course_Add** (For Faculty)
- **Enable/Disable Course_Enroll** (For Student)
- **Enable/Disable Grades_Add** (For Faculty)
- **Effect**: Toggles the value in the corresponding column in the permissions database
- **SQL Query**

```
UPDATE [dbo].[permissions] SET course_add = @course_add
UPDATE [dbo].[permissions] SET course_enroll = @course_enroll
UPDATE [dbo].[permissions] SET grades_add = @grades_add
```

3. Setting Midsem-Endsem Dates:

- **Description**: Clicking the buttons below once the date is picked from the DateTimePicker results in setting new midsem and endsem start dates.
  Buttons contained are: ** Set Midsem Date ** Set Endsem Date
- **Input**: The date that can be selected as per the DateTimePicker
- **Effect**: Sets the midsem, endsem start date as per the input from the DateTimePicker.
- **SQL Query**

```
UPDATE [dbo].[permissions] SET Midsem_Start_Date = @midsem_start
UPDATE [dbo].[permissions] SET Endsem_Start_Date = @endsem_start
```

4. Generation of Seating Arrangement:

- **Description**: One click of this button generates both midsem and endsem seating plan for the students that can be accessed by them in their respective pages.
  Buttons contained are: ** Generate Seating Arrangement

- **Effect**: Populates the 'exam' database table with exam room details for each student.

- **Algorithm**: The algorithm assigns compulsory courses to lecture halls and department electives to Core 5. According to the assumptions, on first day, courses with slots A and A1 are scheduled first, followed by courses with slots B and B1 on the next day, and so forth.

  Suppose there are 4 compulsory courses to be mapped to lecture halls on same day. We will cyclically assign two lecture halls to each course, like this:
  Course 1 -> L1 L2
  Course 2 -> L2 L3
  Course 3 -> L3 L4
  Course 4 -> L4 L1
  The algorithm ensures an even distribution of students across each allocated room for each course while ensuring the room capacity constraints hold well

- More intricacies are presented along with the SQL Queries that is given below. Here we show Midsem Seating Arrangement Generation. The only difference b/w Midsem and Endsem is the start date and the Room type for storing data (0-Midsem, 1-Endsem).

1. Get the start date from the permissions table

- **SQL Query**

```
SELECT @Exam_Date = Midsem_Start_Date FROM permissions;
```

2. Fetch information about courses, their corresponding slots from the timetable, and whether the courses are elective or compulsory. The result is ordered by the Slot column.

- **SQL Query**

```
SELECT t.Course_ID, t.Slot, c.ElectiveOrCompulsory
FROM timetable t
JOIN course c ON t.Course_ID = c.Course_Code
ORDER BY Slot
```

3. Get the number of Rooms available. Room_Type 0(LH), 1(Classroom)

- **SQL Query**

```sql
SELECT @n = COUNT(*) FROM classroom WHERE Room_type = @Room_Type;
```

4. Get the first Room_ID from the classroom table where Room_type is determined by Elective Or Compulsory

- **SQL Query**

```sql
SELECT @Room_ID1 = Room_ID
FROM(
        SELECT Room_ID, ROW_NUMBER() OVER (ORDER BY Room_ID) AS rn
        FROM classroom
        WHERE Room_type = @Room_Type
      ) AS subquery
WHERE rn = @i % @n + 1;
```

5. Get the total number of students from the grade database whosoever's courses have been approved.

- **SQL Query**

```sql
SELECT @Student_Count = COUNT(*) FROM grade WHERE CourseCode = @Course_ID AND
Approval_Status = 'Approved';
```

6. Get the User_ID from the grade table where CourseCode is the same as Course_ID and Approval_Status is 'Approved'

- **SQL QUERY**

```sql
DECLARE student_cursor CURSOR FOR
SELECT User_ID FROM grade
WHERE CourseCode = @Course_ID AND Approval_Status = 'Approved'
ORDER BY User_ID;
```

7. Allocation of the date and time of the exam is done based on the slot allocated to the course during timetable generation.

- **SQL Query**

```sql
SET @Time_Slot = CASE WHEN SUBSTRING(@Slot, 2, 1) = '1' THEN '9AM-11AM' ELSE '2PM-
4PM' END;
```

```
SET @Exam_Date_New = DATEADD(day,
  CASE
  WHEN SUBSTRING(@Slot, 1, 1) = 'A' THEN 0
  WHEN SUBSTRING(@Slot, 1, 1) = 'B' THEN 1
  WHEN SUBSTRING(@Slot, 1, 1) = 'C' THEN 2
  WHEN SUBSTRING(@Slot, 1, 1) = 'D' THEN 3
  WHEN SUBSTRING(@Slot, 1, 1) = 'E' THEN 4
ELSE 0
END, @Exam_Date);
```

8. Students of each course have been split into two rooms equally, hence the below SQL Query is executed by comparing User_ID to the number of students. If it is less than half, then the student is assigned to first room else the second room.

- **SQL Query**

```
INSERT INTO exam (Course_ID, Date, Time_Slot, Room_ID, Student_ID, ExamType)
```

5. Generation of TimeTable Generation (TimeTable.h):

- **Description**: One click of this button generates the TimeTable that will be followed by all students and faculty for the semester. This functionality is implemented in the permissions.h page and uses a helper function that is contained in TimeTable.h. Furthur details of the exact functions that are called for generating slots for compulsory and electives are mentioned in the inline documentation. Buttons contained are: ** Generate TimeTable

- **Effect**: Populates the 'timetable' database with every offered course mapped to a unique slot.

- There are multiple steps in the underlying algorithm that generates the TimeTable. We will show the classes allocation here. The Lab generation is very similar. Compulsory courses are allotted to the Lecture Hall. Morning slots are named from 'A-E' and the evening ones 'A1-E1'. 2nd and 6th semester students are given evening slots and the 4th and 8th semester students are given morning slots.

1. We first fetch all courses that are not labs(i.e Lectures will be non zero) and the ones which are compulsory. Allot courses for semester 2 in LH1, semester 4 in LH2, semester 6 in LH3 and semester 8 in LH4. Since at max a semester can have 5 compulsory courses, they are filled in slots A-E iteratively. The code for compulsory course is as follows:

- **SQL Query**

```
Select * from course where L <> 0 and ElectiveOrCompulsory = 0 and semester =
@sem"
insert into timetable (Course_ID, Room_ID, Slot) values (@Course_ID, @Room_ID,
@Slot)
```

- The query for elective is as follows. Since electives can be offered in multiple sems, type2 captures it. The Room_ID is for now fed as X.
- **SQL Query**

```
Select * from course where L <> 0 and ElectiveOrCompulsory = 1 and ( semester =
@sem or semester = @type2)
```

```
insert into timetable (Course_ID, Room_ID, Slot) values (@Course_ID, @Room_ID,
@Slot)
```

2. The Labs have the same SQL Queries but with different Room_ids and Slot(ML/AL)

3. Now to allot the room for the electives that was earlier filled with 'X', we use the following query. The slot is iteratively given as morning(A-E) and evenings(A1-E1) and the Room_ID is iteratively chosen from the available rooms in Core 5.

- **SQL Query**

```
Select * from timetable where Room_ID = @Room_ID and Slot = @Slot
```

```
update timetable set Room_ID = @Room_ID where Course_ID = @Course_ID
```

Classrooms List Page (classroomlist.h):

1. Display Classroom List

- **Function invoked**: `display_classroom_list`
- **Description**: Retrieves the room id, capacity and room type of all the classrooms and displays it.
- **Input**: None
- **Output**: Displays list of classrooms in a DataGridView.
- **Effect**: None
- **SQL Query**:

```
SELECT [Room_ID],[Capacity],[Room_Type] FROM [dbo].[classroom];
```

2. Addition of a New Classroom

- **Function invoked**: `insert_into_db`
- **Description**: Checks if the data entered for the new classroom is valid and then inserts it into the database.

- **Input**: None
- **Output**: None
- **Effect**: Inserts a new entry with classroom information in classroom database.
- **SQL Query**:

```
SELECT [Room_ID],[Capacity],[Room_Type] FROM [dbo].[classroom];
```

Course Page (CourseList.h)

1. Display Course List along with the faculty according to semester:

- **Function invoked**: `get_course_table`
- **Description**: Retrieves the list of all courses offered in a particular semester along with offering professor.
- **Input**: Sem number chosen from the combo box.
- **Output**: Displays list of courses and faculty offering that course in a DataGridView.
- **SQL Query**:

```
SELECT c.Course_Code, c.Name AS CourseName, c.Semester, f.Name AS FacultyName "
"FROM [dbo].[course] c "
"LEFT JOIN [dbo].[faculty] f ON c.Faculty_ID = f.User_ID "
"WHERE c.Semester LIKE @semester
```

2. Assigning a course to a professor

- **Function invoked**: `btn_assign_Click`
- **Description**: On clicking the button 'assign', the checked course is assigned to the selected professor.
- **Input**: Check box and professor name selected from the combo box.
- **SQL Query**:

```
SELECT User_ID FROM [dbo].[faculty] WHERE Name = '" + selectedFacultyName + "'";
```

```
UPDATE [dbo].[course] SET Faculty_ID = " + userID + " WHERE Course_Code = '" +
courseCode + "'";
```

Integration

The integration of our academic section management system application, developed by a team of seven individuals, involved a systematic approach to merging distinct modules and functionalities into a cohesive final release version. Each team member was assigned specific responsibilities, ensuring a smooth integration process. Our collaborative work was made possible through github.