**Project**: Student Registrations System

**Student name:** Rohith

**Student surname:** Padakanti

**Student email address:** rpada003@fiu.edu

**Student panther ID:** 6390854

**Course code**: COP 5725

**1:** **Write all the entity names and their columns and other information similar to how I wrote in the Entities section (9 points).**

**Student -** This entity represents a student and includes attributes such as student ID, first name, last name, address, phone number, email address, and date of birth.

Student ID | First Name | Last Name | Address | Phone Number | Email | Date of Birth |

|---------------|----------------|----------------|------------|---------------------|---------|-------------------|

| 1 | David | John | 111 NW 4St. | (305)-305-1234 | johndavid123@gmail.com | 11/12/1998 |

**Course -** This entity represents a course and includes attributes such as course ID, course name, instructor name, start time, end time, and room number.

| Course ID | Course Name | Instructor Name | Start Time | End Time | Room Number |

|--------------|-------------------|------------------------|---------------|--------------|---------------------|

| 1 | Principles of DBMS | kiavash.bahreini | 9:00am | 10:30am | AB 205|

**Enrolment** - This entity represents a student's enrolment in a course and includes attributes such as enrolment ID, student ID, course ID, and grade.

| Enrolment ID | Student ID | Course ID | Grade |

|-------------------|---------------|---------------|---------|

| 1 | 1 | 1 | A |

**Schedule** - This entity represents a student's schedule and includes attributes such as student ID, course ID, start time, end time, and room number.

Student ID | Course ID | Start Time | End Time | Room Number |

|---------------|---------------|---------------|--------------|--------------------|

| 1 | 1 | 9:00am | 10:30am | Room 101 |

**Department -** This entity represents a department and includes attributes such as department ID and name.

| Department ID | Department Name |

|---------------------|--------------------------|

| 1 | Computer Science |

**Professor -** This entity represents a professor and includes attributes such as professor ID, name, address, phone number, and email address.

| Professor ID | First Name | Last Name | Address | Phone Number | Email |

|-----------------|----------------|----------------|------------|---------------------|---------|

| 1 | Alex | Doe | 369 Sun St. | (888) 333-6699 | [alexdoe@email.com](mailto:alexdoe@email.com) |

**Attendance:** This entity represents the attendance of students and includes attributes such as Attendance, Student\_Id, Course\_Id, submission Date(Due date), and Status.

|Attendance | Student\_Id | Course\_Id | Date | Status |

| ---------------| --------------- | ------------- | ------ | -------- |

| 100% | 1 | 1 | 20-12-24 | Present |

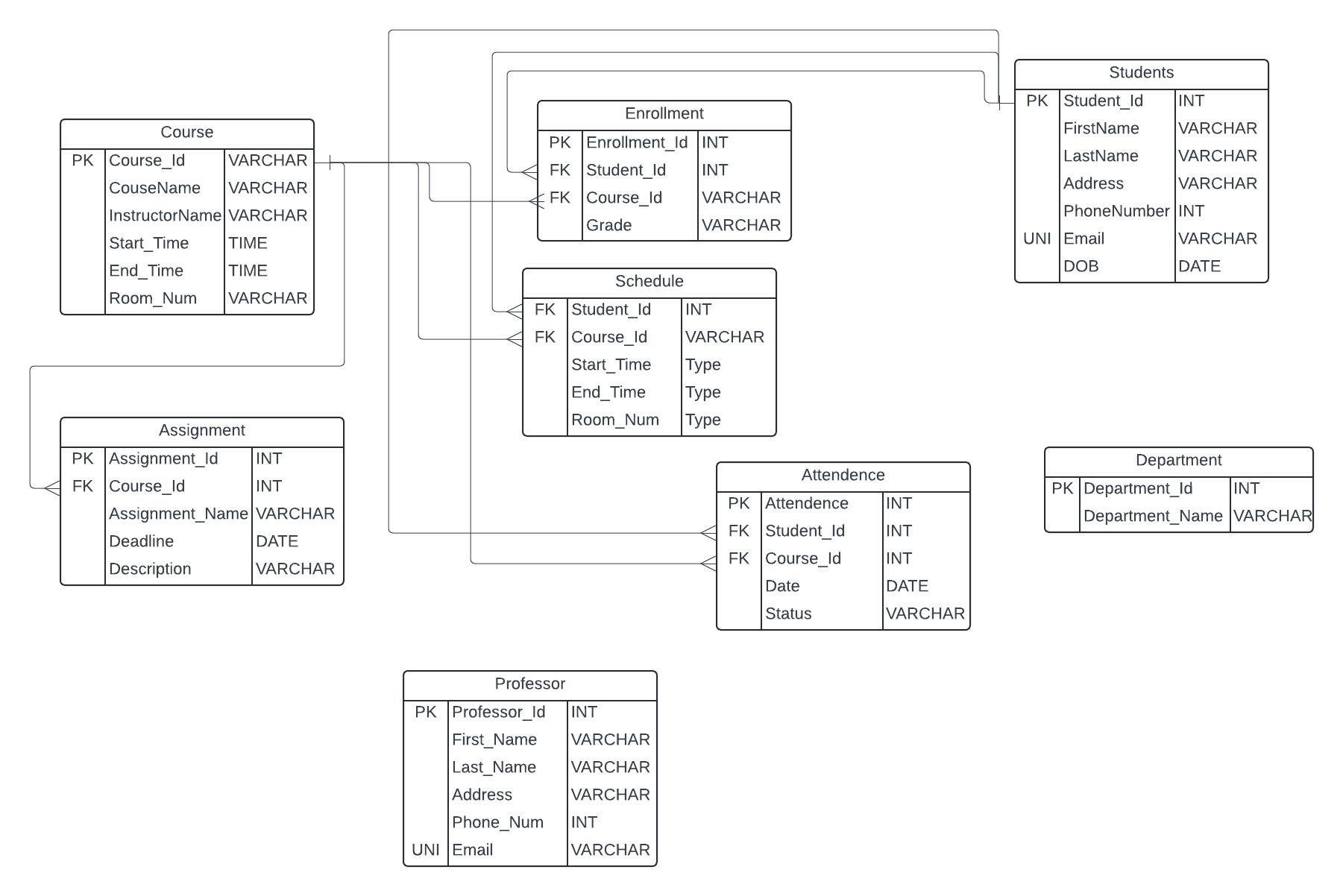
**Assignment:** This entity represents the assignment of students and includes attributes such as assignment, Course\_Id, Assign\_Name, Deadline, and Description of the assignment.

| Assignment\_Id | Course\_Id | Assignment\_Name | Deadline | Description |

| ------------------- | -------------- | ------------------------ | ----------- | --------------- |

| 1 | 1 | Project\_1 | 15-04-24 | Student Registrations System |

**2:** **Perform task 1, capture an image of the Lucidchart's conceptual E-R model for the database, and add it to this section. You may use any other tools for this task (9 points).**



**One-to-Many Relationships:**

students to Enrolment

Course to Enrolment

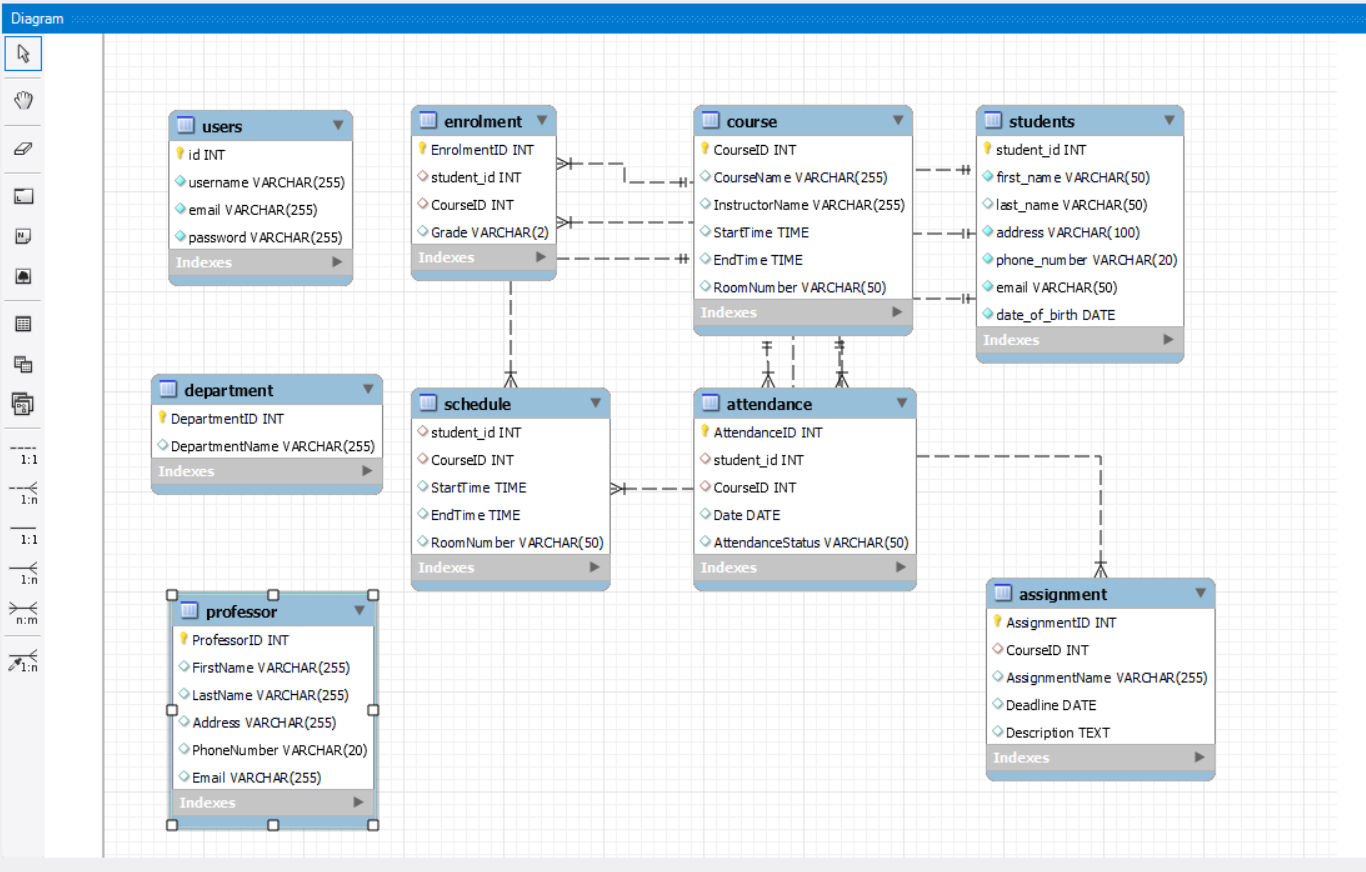
Course to Assignment

Course to Attendance

**Many-to-Many Relationships:**

students to Course through Enrolment

**3: Perform task 2, add a database diagram for your tables and their relationships in MySQL (or another DBMS of your choice), and add it below. Then, add all the SQL codes (CREATE TABLE,ALTER TABLE, CREATE PRIMARY KEY, FOREIGN KEY, etc.) related to creating the relational model for the database in MySQL in this section (9 points).**



CREATE TABLE students (

student\_id INT AUTO\_INCREMENT PRIMARY KEY, ( Auto-incremented and primary key)

first\_name VARCHAR(50) NOT NULL,

last\_name VARCHAR(50),

address VARCHAR(100) NOT NULL,

phone\_number VARCHAR(20) NOT NULL,

email VARCHAR(50) UNIQUE NOT NULL, ( Unique and not null constraint added)

date\_of\_birth DATE NOT NULL

);

CREATE TABLE IF NOT EXISTS users (

id INT AUTO\_INCREMENT PRIMARY KEY, ( Auto-incremented and primary key)

username VARCHAR(255) NOT NULL,

email VARCHAR(255) NOT NULL UNIQUE, ( Unique and not null constraint added)

password VARCHAR(255) NOT NULL

);

CREATE TABLE Course (

CourseID INT PRIMARY KEY,( Primary Key)

CourseName VARCHAR(255),

InstructorName VARCHAR(255),

StartTime TIME,

EndTime TIME,

RoomNumber VARCHAR(50)

);

CREATE TABLE Enrolment (

EnrolmentID INT PRIMARY KEY, (Primary Key)

student\_id INT, (Foreign Key)

CourseID INT, (Foreign Key)

Grade VARCHAR(2),

FOREIGN KEY (student\_id) REFERENCES students(student\_id),( Foreign Key)

FOREIGN KEY (CourseID) REFERENCES Course(CourseID) (Foreign Key)

);

CREATE TABLE Schedule (

student\_id INT, (Foreign Key)

CourseID INT, (Foreign Key)

StartTime TIME,

EndTime TIME,

RoomNumber VARCHAR(50),

FOREIGN KEY (student\_id) REFERENCES students(student\_id), (Foreign Key)

FOREIGN KEY (CourseID) REFERENCES Course(CourseID) (Foreign Key)

);

CREATE TABLE Department (

DepartmentID INT PRIMARY KEY, (Primary Key)

DepartmentName VARCHAR(255)

);

CREATE TABLE Professor (

ProfessorID INT PRIMARY KEY, (Primary Key)

FirstName VARCHAR(255),

LastName VARCHAR(255),

Address VARCHAR(255),

PhoneNumber VARCHAR(20),

Email VARCHAR(255)

);

CREATE TABLE Assignment (

AssignmentID INT PRIMARY KEY, (Primary Key)

CourseID INT, (Foreign Key)

AssignmentName VARCHAR(255),

Deadline DATE,

Description TEXT,

FOREIGN KEY (CourseID) REFERENCES Course(CourseID) (Foreign Key)

);

CREATE TABLE Attendance (

AttendanceID INT PRIMARY KEY,

student\_id INT,

CourseID INT,

Date DATE,

AttendanceStatus VARCHAR(50),

FOREIGN KEY (student\_id) REFERENCES students(student\_id), (Foreign Key)

FOREIGN KEY (CourseID) REFERENCES Course(CourseID) (Foreign Key)

);

**4**: **Perform task 3 and write all the SQL queries for SELECT, INSERT, UPDATE, and DELETE statements of the system's register and login pages (15 points).**

**Insert:**

INSERT INTO students (first\_name, last\_name, address, phone\_number, email, date\_of\_birth)

VALUES (?, ?, ?,?, ?,?);

**Update:**

UPDATE students

SET first\_name = ?,

last\_name = ?,

address = ?,

phone\_number = ?,

email = ?,

date\_of\_birth = ?

WHERE student\_id = ?;

**Select:**

SELECT \* FROM students;

SELECT \* FROM students WHERE first\_name = 'John';

SELECT \* FROM students WHERE address LIKE '%New York%';

SELECT \* FROM students WHERE date\_of\_birth > '2000-01-01';

SELECT \* FROM students WHERE phone\_number = '123-456-7890';

SELECT \* FROM students WHERE email LIKE '%@example.com';

**DELETE:**

DELETE FROM students WHERE student\_id = ?;

**5:** **Perform task 4 and write what language you used. What connectivity API did you use? What is your connection string using the chosen programming language? Why did you use this programming language? Capture the screen of the register and login page of the software application you designed (15 points).**

* I employed React for frontend development and Node.js with Express for backend development, leveraging the MySQL package for connectivity to the MySQL database.
* The mysql module is a choice for connecting Node.js applications to MySQL databases
* The connection string in Node.js typically includes localhost(Host), username(MySQL), password, and database name of MySQL.

const db = mysql.createConnection({

host: 'localhost',

user: 'root',

password: '#########',

database: 'Project'

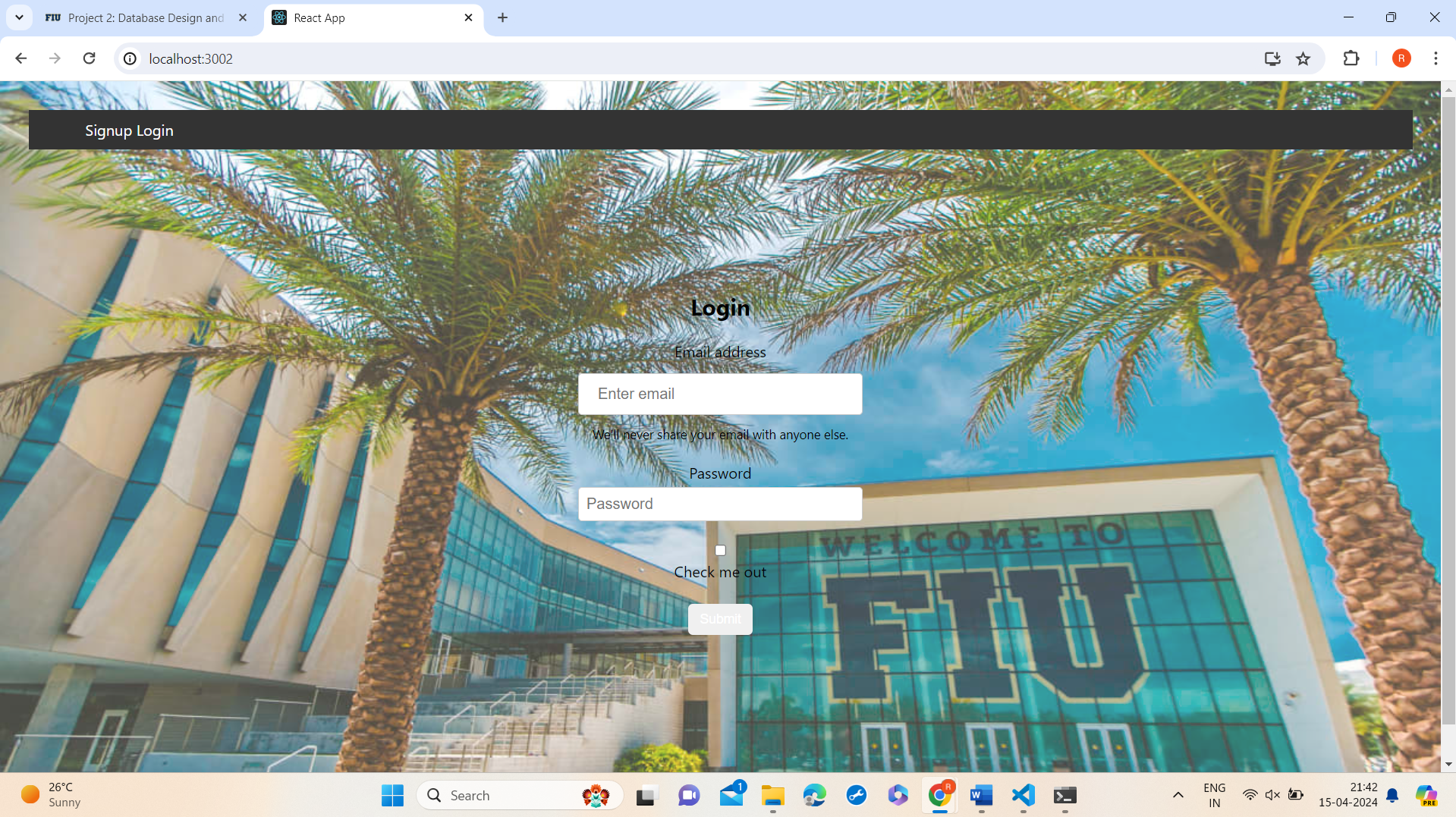
});

* Node.js is efficient and has a vast ecosystem of libraries and frameworks making it easy to build and find solutions for a beginner. Node.js is known for its scalability, making it suitable for both small-scale and large-scale applications, making it easier to server-side development.

Sign Up Page:



Login Page:



**6:** **Include all the source codes related to performing SELECT, INSERT, UPDATE, and DELETE Statements for the register and login page. The source should show the SQL commands in MySQL (or another DBMS) and the commands used in your programming language to call these SQL statements (9 points).**

**SELECT:**

app.get("/api/get/:student\_id",(req,res)=>{

    const { student\_id } = req.params;

    const sqlGet = "SELECT \* FROM students WHERE student\_id=?";

    db.query(sqlGet, student\_id, (error, result) => {

        if (error) {

            console.log(error);

            res.status(500).send("Error fetching student data");

        } else {

            res.status(200).send(result);

        }

    });

});

Update:

// Update

app.put("/api/update/:student\_id",(req,res)=>{

    const { student\_id } = req.params;

    const { first\_name, last\_name, address, phone\_number, email, date\_of\_birth } = req.body;

    const sqlUpdate = "UPDATE students SET first\_name =?, last\_name =?, address =?, phone\_number =?, email =?, date\_of\_birth =? WHERE student\_id=?";

    db.query(sqlUpdate, [first\_name, last\_name, address, phone\_number, email, date\_of\_birth, student\_id], (error, result) => {

        if (error) {

            console.log(error);

            res.status(500).send("Error updating student");

        } else {

            res.status(200).send("Student updated successfully");

        }

    });

});

Insert:

app.post("/api/post",(req, res) =>{

    const {first\_name, last\_name, address, phone\_number, email, date\_of\_birth} = req.body;

    const sqlInsert = "INSERT INTO students (first\_name, last\_name, address, phone\_number, email, date\_of\_birth) VALUES (?, ?, ?, ?, ?, ?)";

    db.query(sqlInsert, [first\_name, last\_name, address, phone\_number, email, date\_of\_birth], (error, result) =>{

        if(error){

            console.log(error);

        } else {

            res.status(200).send("Student added successfully");

        }

    });

});

Delete:

// Delete

app.delete("/api/remove/:student\_id", (req, res) => {

    const { student\_id } = req.params;

    const sqlRemove = "DELETE FROM students WHERE student\_id = ?";

    db.query(sqlRemove, student\_id, (error, result) => {

        if (error) {

            console.log(error);

            res.status(500).send("Error deleting student");

        } else {

            res.status(200).send("Student deleted successfully");

        }

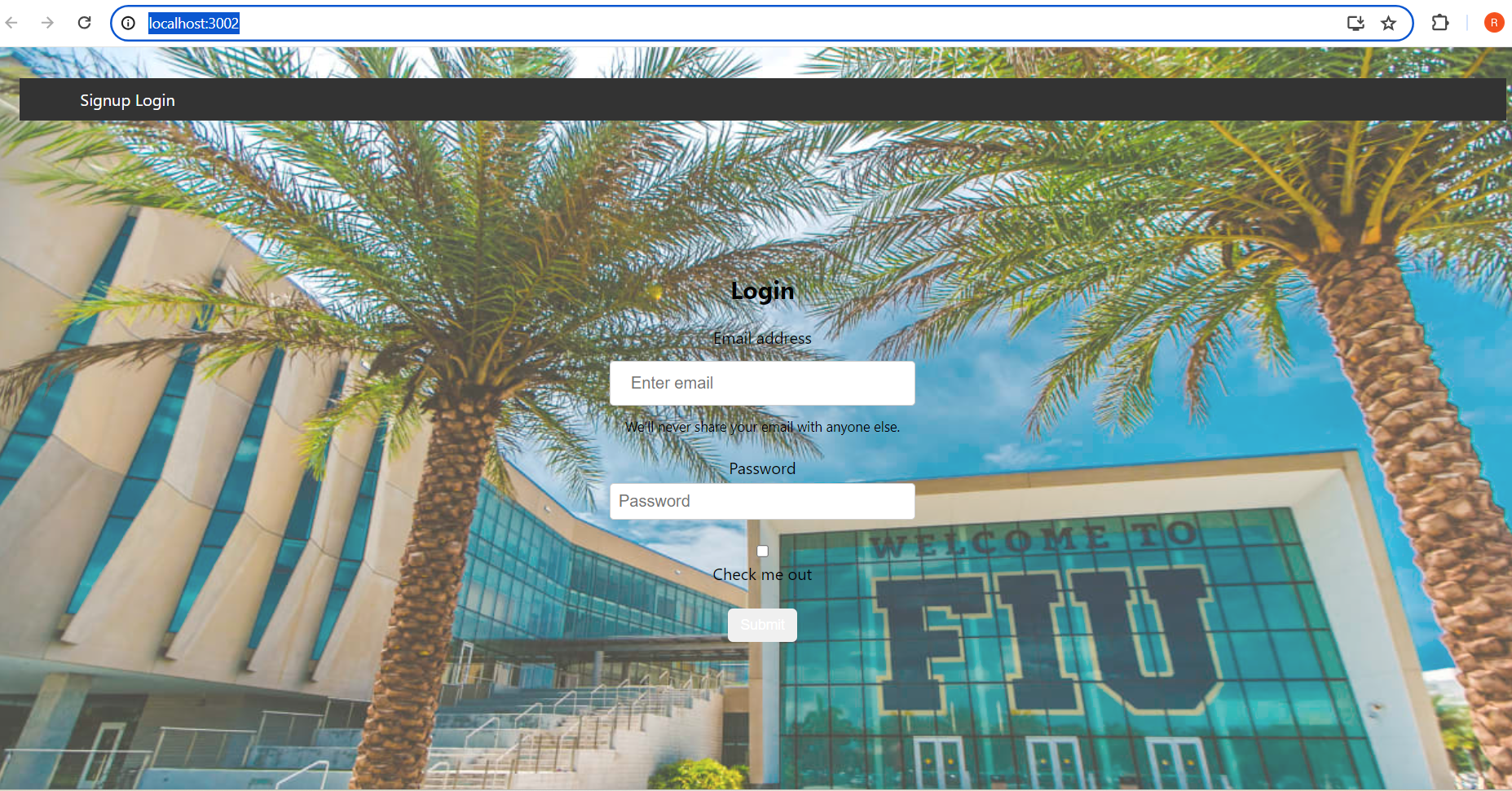
    });

});

**7:** **Write any limitations and bugs your system includes. You can add a capture screen of your issues in the following section (6 points).**

**Browser Compatibility:** Websites or web applications may not display or function consistently across different web browsers due to variations in rendering engines, CSS support, or JavaScript execution. Ensuring cross-browser compatibility can be challenging and may require additional testing and code adjustments.

**Concurrency Issues:** Managing concurrent access to resources, such as database connections or shared data, can lead to race conditions, deadlocks, or resource contention, affecting system performance and reliability.

**Performance on Low-End Devices:** Complex or resource-intensive frontend components, such as large images, animations, or JavaScript frameworks, can lead to slow performance or lag on low-end devices with limited processing power or memory. Optimizing front-end code and assets for performance can help improve user experience across various devices and network conditions.  


As you can see the background image we can make it more attractive with live images or animations but all the systems with basic processing power can not support those things and will not load the cite as normal systems do.

**8**. **Write any obstacles, issues, problems, difficulties, etc., you encountered during the design, development, implementation, and demonstration of the Student Registrations System at ABC University (6 points).**

Requirements Gathering: Ensuring a clear and comprehensive understanding of system requirements can be challenging, leading to miscommunication or ambiguity in system expectations.

Design Complexity: Designing a system that meets all functional and non-functional requirements while maintaining scalability, reliability, and usability can be complex, requiring careful consideration of architectural decisions and trade-offs.

Data Security and Privacy: Ensuring compliance with data protection regulations and safeguarding sensitive student information against unauthorized access, data breaches, or privacy violations is critical but can be challenging to implement effectively.

Testing and Quality Assurance: Thorough testing, including unit testing, integration testing, and user acceptance testing, is essential to ensure the system functions as intended and meets user expectations. However, testing can be time-consuming and resource-intensive.

Maintenance and Support: Providing ongoing maintenance, support, and updates for the Student Registrations System to address bugs, performance issues, or evolving requirements requires dedicated resources and processes.