**‘Project Name’ - Backend & Database Implementation Report (Phase 1)**

**📅 Status: Student Module Completed**

## Technologies Used:

* **Backend:** Node.js with Express.js
* **Database:** MySQL
* **API Testing:** Postman
* **Environment Config:** dotenv (.env)
* **Architecture:** Modular routing and DB abstraction

## Modules To Be Done

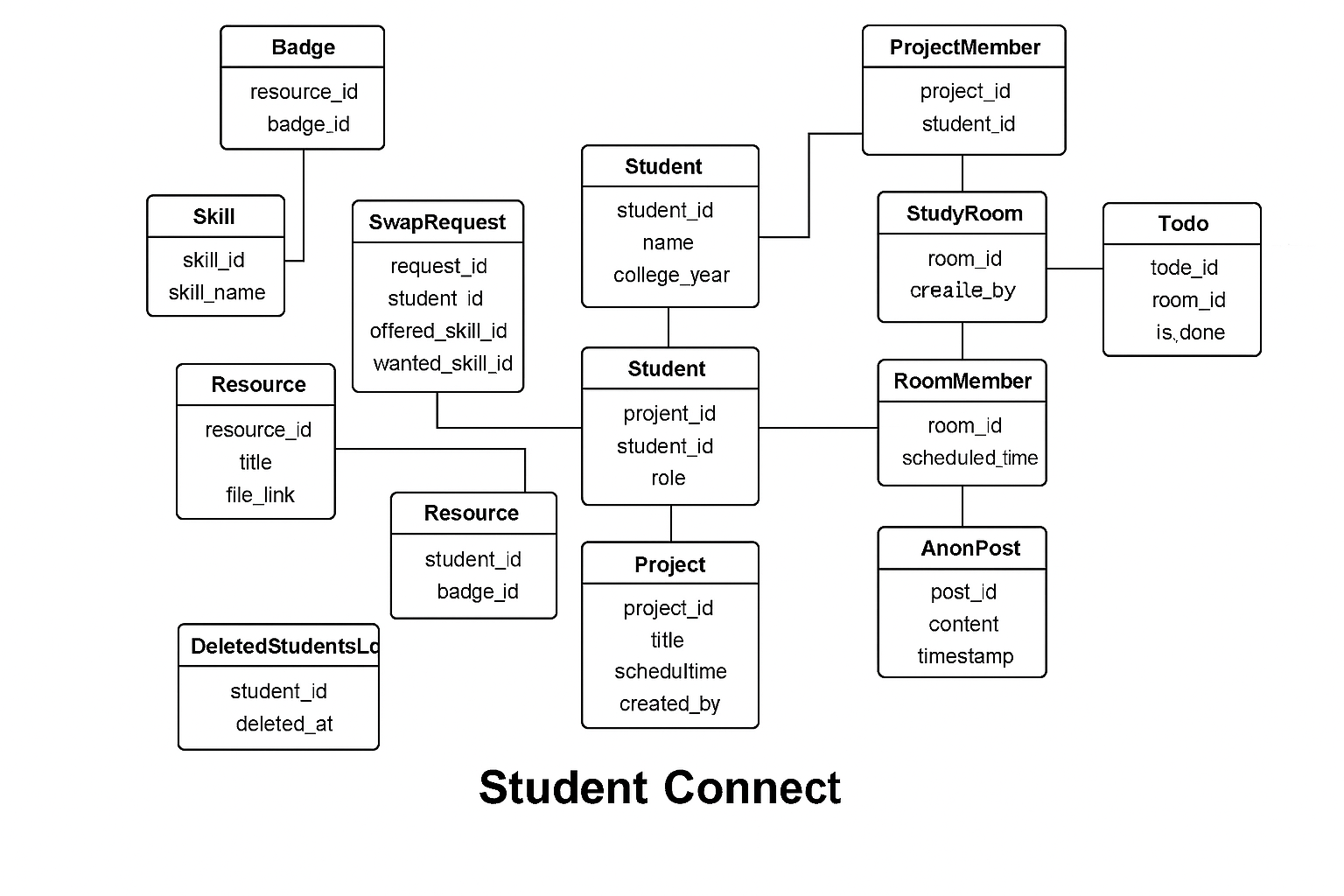
Here’s what we’ll build next (based on ER diagram):

1. **Skill**
2. **StudentSkill** (Many-to-Many join table)
3. **Project**
4. **StudentProject** (Many-to-Many)
5. **Post**
6. **Comment**
7. **College** (if needed for filtering)
8. **Authentication/Login Module**

## Tables Created:

| **Table Name** | **Description** |
| --- | --- |
| student | Stores basic information of registered students |
| deletedstudentslog | Trigger-based log table that records student deletions |
| skill | Master list of technical or soft skills |
| studentskill | Junction table linking students to their respective skills |
| project | Holds information about projects created or joined by students |
| projectmember | Links students to projects they are participating in |
| badge | Contains all possible badges students can earn |
| studentbadge | Tracks badges earned by individual students |
| studyroom | Represents virtual study rooms students can join |
| roommember | Lists which students are members of which study rooms |
| resource | Contains shared resources like links or files related to study rooms |
| swaprequest | Used to handle room swap requests between students |
| todo | Personal to-do list entries created by students |
| anonpost | Stores anonymous posts submitted by students |

## ER Diagram:



## Modules Completed:

**1. Student Module**

**Features:**

| **Feature** | **Endpoint** | **Method** | **DB Concept Used** |
| --- | --- | --- | --- |
| Register Student (via stored procedure) | /api/student/register | POST | Stored Procedure |
| Get All Students | /api/student | GET | CRUD |
| Get Student by ID | /api/student/:id | GET | CRUD |
| Update Student Info | /api/student/:id | PUT | CRUD |
| Delete Student | /api/student/:id | DELETE | Trigger (logging) |

**2. Stored Procedure: RegisterStudent**

DELIMITER $$

CREATE PROCEDURE RegisterStudent(

IN p\_name VARCHAR(100),

IN p\_email VARCHAR(100),

IN p\_college VARCHAR(100),

IN p\_year INT

)

BEGIN

INSERT INTO Student (name, email, college, year)

VALUES (p\_name, p\_email, p\_college, p\_year);

SELECT LAST\_INSERT\_ID() AS student\_id;

END$$

DELIMITER ;

* Tested in Postman – Returns studentId after successful insert

**3. Trigger: LogDeletedStudent**

CREATE TABLE DeletedStudentsLog (

log\_id INT AUTO\_INCREMENT PRIMARY KEY,

student\_id INT,

name VARCHAR(100),

email VARCHAR(100),

college VARCHAR(100),

year INT,

deleted\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

DELIMITER $$

CREATE TRIGGER LogDeletedStudent

BEFORE DELETE ON Student

FOR EACH ROW

BEGIN

INSERT INTO DeletedStudentsLog (student\_id, name, email, college, year)

VALUES (OLD.student\_id, OLD.name, OLD.email, OLD.college, OLD.year);

END$$

DELIMITER ;

* Tested – Deleting a student logs their data in DeletedStudentsLog

## Testing Status:

| **Route** | **Tested** | **Status** |
| --- | --- | --- |
| POST /api/student/register | ✅ | Working |
| GET /api/student | ✅ | Working |
| GET /api/student/:id | ✅ | Working |
| PUT /api/student/:id | ✅ | Working |
| DELETE /api/student/:id | ✅ | Working |
| SELECT \* FROM DeletedStudentsLog | ✅ | Logs student on delete |

## DBMS Concepts Implemented:

* Stored Procedure
* Trigger
* Modular API with Express
* Structured API Testing with Postman
* Database Connection Management

## Environment Setup:

.env file:

DB\_HOST=localhost

DB\_USER=root

DB\_PASSWORD=yourpassword

DB\_NAME=unicon

## Folder Structure So Far:

backend/

|

├── db/

│ └── index.js # MySQL connection setup

├── routes/

│ └── studentRoutes.js # All Student APIs

├── controller/

│ └── studentController.js

├── .env

├── app.js # Express setup

└── package.json

## Next Step:

Proceeding to **Skill & StudentSkill Module** with:

* Many-to-Many Relationships
* Inner Joins
* Possibly a Cursor if applicable
* Additional DBMS concepts like Views and Transactions (if needed)

### 🔧 1. **Stored Procedure to Create a Study Room**

This procedure inserts a new room into StudyRoom.

sql

CopyEdit

DELIMITER $$

CREATE PROCEDURE CreateRoom (

IN p\_topic VARCHAR(255),

IN p\_max\_members INT,

IN p\_scheduled\_time DATETIME,

IN p\_created\_by INT

)

BEGIN

INSERT INTO StudyRoom (topic, max\_members, scheduled\_time, created\_by)

VALUES (p\_topic, p\_max\_members, p\_scheduled\_time, p\_created\_by);

END $$

DELIMITER ;

### 🔧 2. **Transaction to Join a Room (with max limit check)**

This uses a **transaction** and **join** to ensure max member limit isn't exceeded.

sql

CopyEdit

DELIMITER $$

CREATE PROCEDURE JoinRoom (

IN p\_room\_id INT,

IN p\_student\_id INT

)

BEGIN

DECLARE current\_count INT;

DECLARE max\_allowed INT;

START TRANSACTION;

SELECT COUNT(\*) INTO current\_count FROM RoomMembership WHERE room\_id = p\_room\_id;

SELECT max\_members INTO max\_allowed FROM StudyRoom WHERE room\_id = p\_room\_id;

IF current\_count < max\_allowed THEN

INSERT INTO RoomMembership (room\_id, student\_id)

VALUES (p\_room\_id, p\_student\_id);

INSERT INTO RoomLog (room\_id, action)

VALUES (p\_room\_id, CONCAT('Student ', p\_student\_id, ' joined'));

END IF;

COMMIT;

END $$

DELIMITER ;

### ⚡ 3. **Trigger to Log Leave Action**

Whenever a student is removed from a room, we log it in RoomLog.

sql

CopyEdit

DELIMITER $$

CREATE TRIGGER AfterLeaveRoom

AFTER DELETE ON RoomMembership

FOR EACH ROW

BEGIN

INSERT INTO RoomLog (room\_id, action)

VALUES (OLD.room\_id, CONCAT('Student ', OLD.student\_id, ' left'));

END $$

DELIMITER ;

### 🔁 4. **Cursor to Fetch All Members of a Room (for attendance)**

We’ll loop through members of a room and return their IDs.

sql

CopyEdit

DELIMITER $$

CREATE PROCEDURE GetRoomMembers (

IN p\_room\_id INT

)

BEGIN

DECLARE done INT DEFAULT FALSE;

DECLARE sid INT;

DECLARE cur CURSOR FOR

SELECT student\_id FROM RoomMembership WHERE room\_id = p\_room\_id;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

OPEN cur;

read\_loop: LOOP

FETCH cur INTO sid;

IF done THEN

LEAVE read\_loop;

END IF;

-- For now just SELECT, can be extended to do logic

SELECT sid AS student\_in\_room;

END LOOP;

CLOSE cur;

END $$

DELIMITER ;

 **Get All Study Rooms with Members (GET /rooms)**:

* You're correctly joining the StudyRoom and RoomMembership tables using a LEFT JOIN and grouping the rooms by room\_id. This will give you a good structure of rooms and their members.
* The grouping logic where you're aggregating members is a good approach to ensuring you get the desired result in one query.



RESOURCE HUB

**🧠 3. Use of DBMS Concepts**

**✅ Joins**

Later if you store uploader in a users table, use:

sql

CopyEdit

SELECT r.\*, u.name FROM resources r

JOIN users u ON r.uploader\_id = u.user\_id;

**✅ Triggers**

Track uploads:

sql

CopyEdit

CREATE TABLE upload\_log (

log\_id INT AUTO\_INCREMENT PRIMARY KEY,

resource\_id INT,

action VARCHAR(50),

timestamp DATETIME DEFAULT CURRENT\_TIMESTAMP

);

CREATE TRIGGER log\_upload

AFTER INSERT ON resources

FOR EACH ROW

INSERT INTO upload\_log (resource\_id, action)

VALUES (NEW.resource\_id, 'UPLOAD');

**✅ Stored Procedure**

Get recent N resources:

sql

CopyEdit

DELIMITER //

CREATE PROCEDURE GetRecentResources(IN n INT)

BEGIN

SELECT \* FROM resources ORDER BY upload\_date DESC LIMIT n;

END //

DELIMITER ;

Project

**1. Projects Table**

Stores the details about each project.

sql

CopyEdit

CREATE TABLE Projects (

project\_id INT AUTO\_INCREMENT PRIMARY KEY,

title VARCHAR(255) NOT NULL,

description TEXT,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

updated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP

);

**2. Roles Table**

Stores the roles for each project.

sql

CopyEdit

CREATE TABLE Roles (

role\_id INT AUTO\_INCREMENT PRIMARY KEY,

project\_id INT,

role\_name VARCHAR(100) NOT NULL,

FOREIGN KEY (project\_id) REFERENCES Projects(project\_id) ON DELETE CASCADE

);

**3. TeamMembers Table**

Tracks the team members for each project and their associated roles.

sql

CopyEdit

CREATE TABLE TeamMembers (

member\_id INT AUTO\_INCREMENT PRIMARY KEY,

project\_id INT,

member\_name VARCHAR(255) NOT NULL,

role\_id INT,

FOREIGN KEY (project\_id) REFERENCES Projects(project\_id) ON DELETE CASCADE,

FOREIGN KEY (role\_id) REFERENCES Roles(role\_id) ON DELETE SET NULL

);

**4. Skills Table**

Stores the skills associated with each project.

sql

CopyEdit

CREATE TABLE Skills (

skill\_id INT AUTO\_INCREMENT PRIMARY KEY,

project\_id INT,

skill\_name VARCHAR(100) NOT NULL,

FOREIGN KEY (project\_id) REFERENCES Projects(project\_id) ON DELETE CASCADE

);