Small music school business platform

- Github repository for this project: https://github.com/kaythep003/ICCS225-final-project
- Live database: render.com
    - Username: root
    - Password: CjPOYDeNLlFvKaLfLtiNgcZ12jUGJ6od

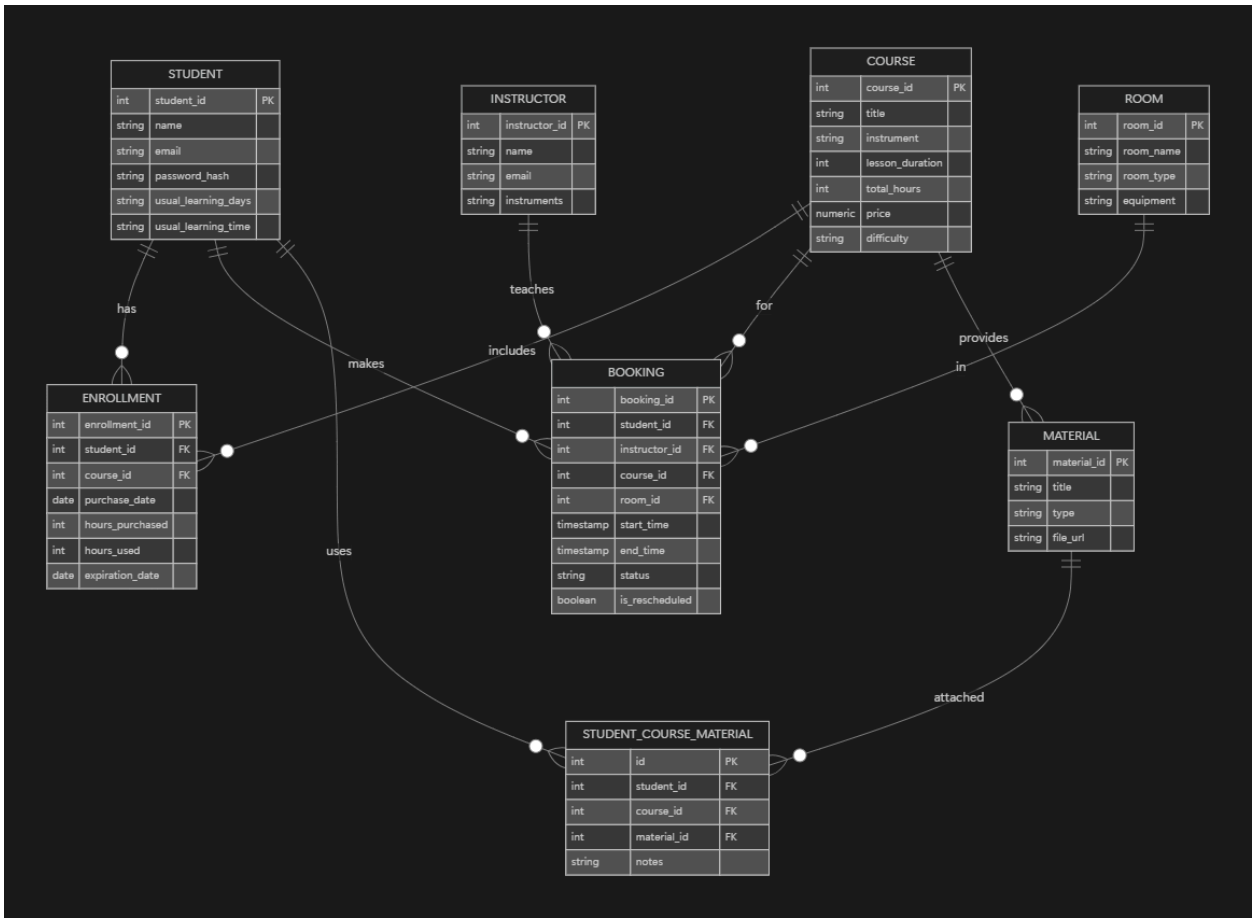
Business objective: Trying to solve 1) process pain point and 2) productivity pain point by minimizing human resources and increasing efficiency specifically in the booking process by developing database-driven platform

A key function of this application is to schedule music lessons without any help from staff, ensuring the consideration in conflict between student, instructor, and room availability.

Core Features (for UI design) :

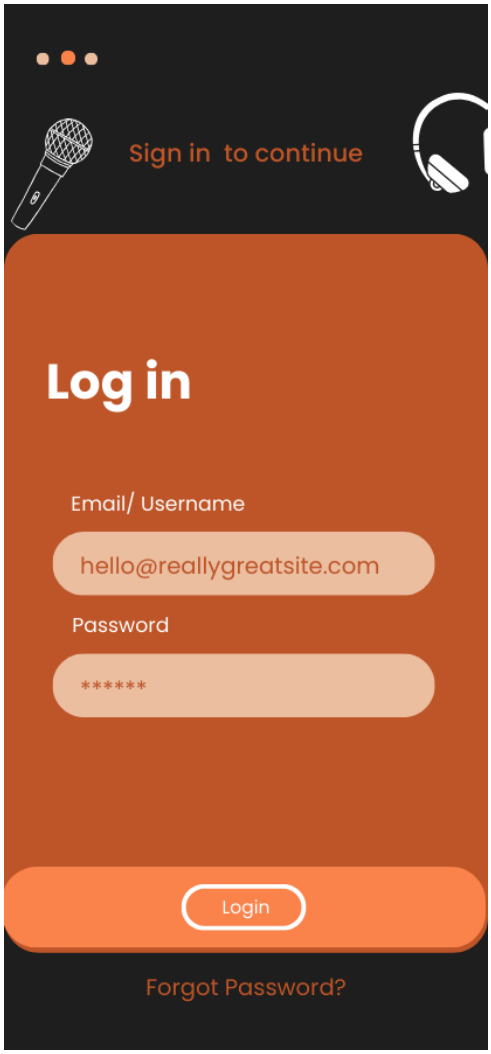
→ Login & authentication
→ Payment tracking
→ Booking & rescheduling
→ Lesson details and material provision
→ Student' s information

ER diagram :



Entities (4 main entities and 4 weaker entities):

1) **STUDENT**, primary key: student_id
2) **INSTRUCTOR**, primary key: instructor_id
3) **COURSE**, primary key: course_id
4) **ROOM,** primary key: room_id
5) **MATERIAL**, primary key: material_id
6) **BOOKING**, primary key: booking_id, foreign keys: student_id, instructor_id, course_id, room_id
7) **ENROLLMENT**, primary key: enrollment_id, foreign keys: student_id, course_id
8) **STUDENT_COURSE_MATERIAL**, primary key: id, foreign keys: student_id, course_id, material_id



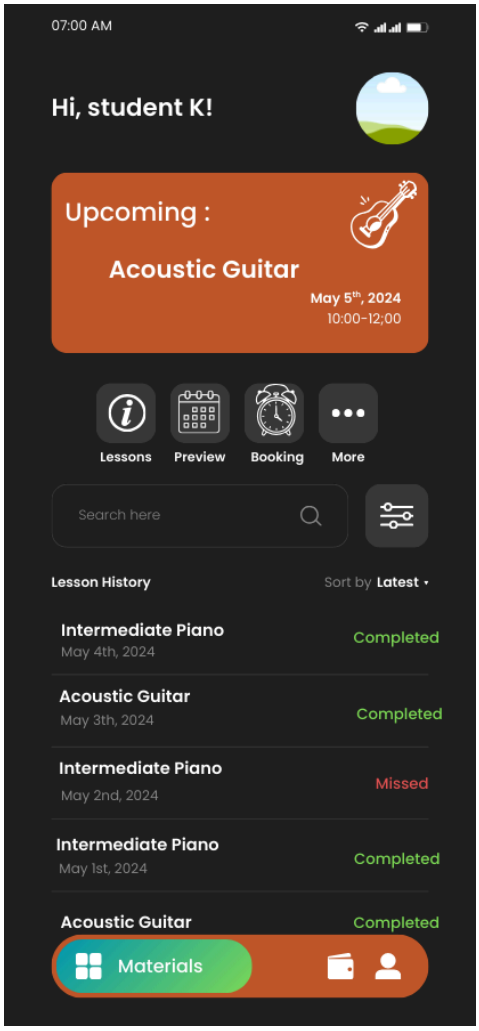
A) Login & authentication

- Ensuring only valid users access stored student data, booking information, and course resources
- **Main function:**

```
None
CREATE OR REPLACE FUNCTION login(_email TEXT, _pw TEXT)
RETURNS TABLE(user_id INT, role TEXT, username TEXT) AS $$
BEGIN
    IF NOT password_check(_email, _pw) THEN
        RAISE EXCEPTION 'Invalid credentials';
    END IF;
```

```
    RETURN QUERY
    SELECT id, role, name FROM users WHERE email = _email;
END;
$$ LANGUAGE plpgsql;
```

- **Expected Output :**
    - UI receives a **t**oken / session flag
    - Grants access to dashboard and personalized data



B) Student dashboard

- Provide a quick overview of the user's learning progress and schedule including the past record checking the student has missed the class
    - Show upcoming class time

```
SELECT * FROM booking
WHERE student_id=$1 AND start_time > NOW()
ORDER BY start_time;
```

- **Main function:**

```
None
CREATE OR REPLACE FUNCTION get_upcoming_classes(_user INT)
RETURNS TABLE(
```

```
    lesson_id INT,
    start_time TIMESTAMP,
    end_time TIMESTAMP,
    room VARCHAR,
    teacher VARCHAR
) AS $$
BEGIN
    RETURN QUERY
    SELECT l.id, s.start_time, s.end_time, r.name, u.name
    FROM bookings b
    JOIN schedule s ON b.schedule_id = s.id
    JOIN lessons l ON s.lesson_id = l.id
    JOIN rooms r ON s.room_id = r.id
    JOIN users u ON l.teacher_id = u.id
    WHERE b.user_id = _user
      AND s.start_time > NOW()
    ORDER BY s.start_time ASC;
END;
$$ LANGUAGE plpgsql;
```

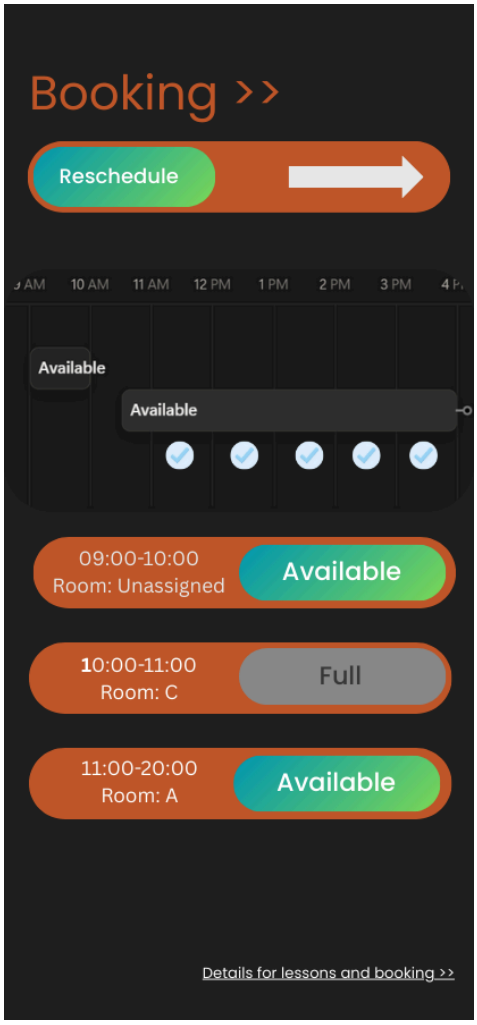
- Checking lesson history

```
SELECT * FROM booking
WHERE student_id=$1 AND start_time < NOW()
ORDER BY start_time DESC;
```

- **Main function:**

```
None
CREATE OR REPLACE FUNCTION get_learning_history(_user INT)
RETURNS TABLE(
    lesson_name TEXT,
    attended TIMESTAMP,
    teacher TEXT
) AS $$
BEGIN
    RETURN QUERY
```

```
    SELECT l.title, s.start_time, u.name
    FROM bookings b
    JOIN schedule s ON b.schedule_id = s.id
    JOIN lessons l ON s.lesson_id = l.id
    JOIN users u ON l.teacher_id = u.id
    WHERE b.user_id = _user
      AND s.end_time < NOW()
    ORDER BY s.end_time DESC;
END;
$$ LANGUAGE plpgsql;
```



C) **Booking**

- Assigning lesson slots by date, time, instructor, and room.
- **Main function;**

```
None
CREATE OR REPLACE FUNCTION book_lesson(_user INT, _schedule INT)
RETURNS TEXT AS $$
BEGIN
    IF check_time_conflict(_schedule, _user) THEN
        RETURN 'Time conflict';
    END IF;

    INSERT INTO bookings(user_id, schedule_id, booked_at)
    VALUES(_user, _schedule, NOW());
```

```
    RETURN 'Booking successful';
END;
$$ LANGUAGE plpgsql;
```

D) **Reschedule**

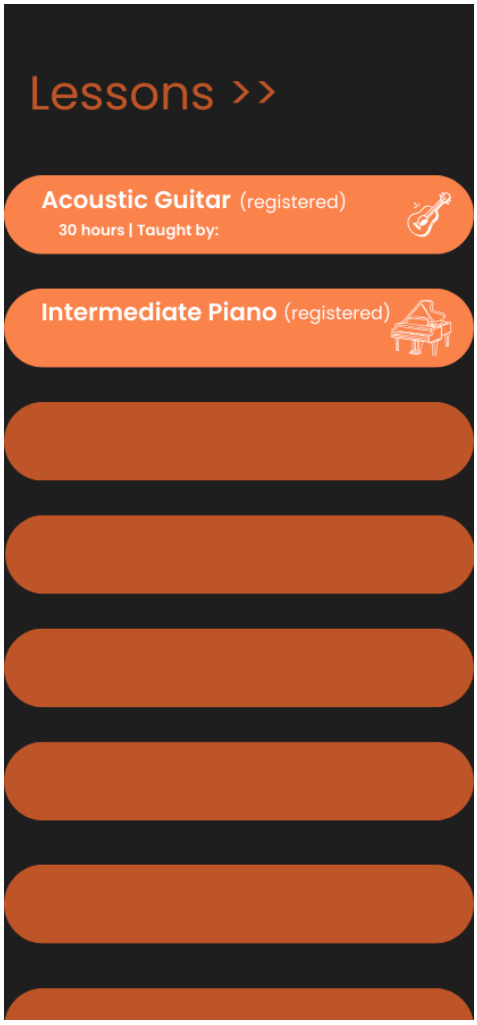
- Students have a chance to reschedule their class if they could not come to the class. However, they must reschedule at least 2 hours in advance of the class start time.
- **Main function:**

```
None
CREATE OR REPLACE FUNCTION reschedule_booking(_booking INT, _new_schedule INT)
RETURNS TEXT AS $$
DECLARE _user INT;
BEGIN
    SELECT user_id INTO _user FROM bookings WHERE id=_booking;

    IF check_time_conflict(_new_schedule, _user) THEN
        RETURN 'Time conflict';
    END IF;

    UPDATE bookings
    SET schedule_id = _new_schedule
    WHERE id=_booking;

    RETURN 'Rescheduled';
END;
$$ LANGUAGE plpgsql;
```



E)_**Lesson details and material distribution**

- Checking further information for registered course (Course name, difficulty, total hours, responsible instructor)
- Exploring the list of other courses
- Check the materials that have been used in the course

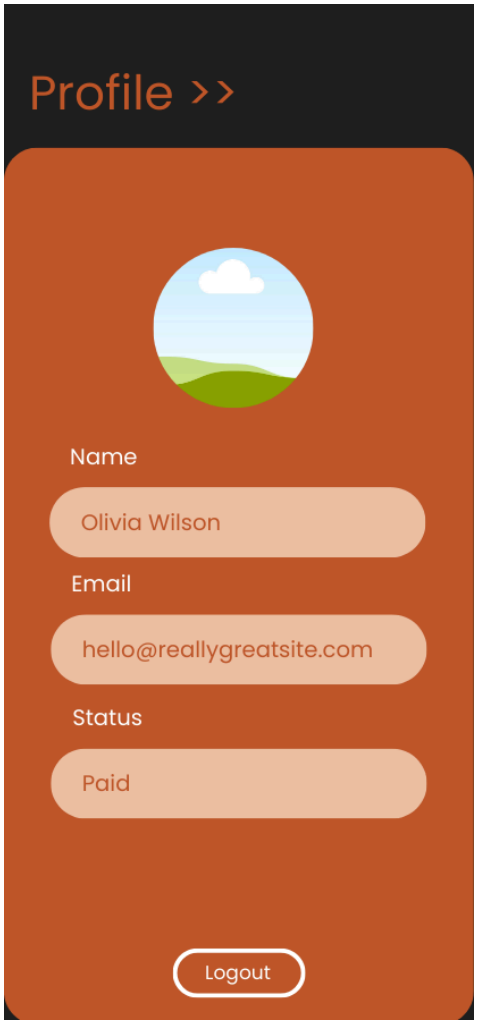
```
SELECT file_url FROM material
JOIN course_material USING(material_id)
WHERE student_id = $1;
```

  - **<u>Main function (lesson details):</u>**

```
None
CREATE OR REPLACE FUNCTION get_lesson_details(_lesson INT)
RETURNS TABLE(
    title TEXT,
    description TEXT,
    teacher TEXT,
    duration INT
) AS $$
BEGIN
    RETURN QUERY
    SELECT l.title, l.description, u.name, l.duration_minutes
    FROM lessons l
    JOIN users u ON l.teacher_id = u.id
    WHERE l.id = _lesson;
END;
$$ LANGUAGE plpgsql;
```

- **Main function (material information):**

```
None
CREATE OR REPLACE FUNCTION get_materials(_lesson INT)
RETURNS TABLE(
    filename TEXT,
    file_type TEXT,
    url TEXT
) AS $$
BEGIN
    RETURN QUERY
    SELECT m.filename, m.filetype, m.file_url
    FROM materials m
    WHERE m.lesson_id = _lesson;
END;
$$ LANGUAGE plpgsql;
```



F) **Student/ Instructor profile**

- Showing full name, email address, and status (paid, expired, instructor)
- **Main function (payment status):**

```
None
CREATE OR REPLACE FUNCTION get_payment_status(_user INT)
RETURNS TABLE(
    booking_id INT,
    status TEXT,
    amount NUMERIC
) AS $$
BEGIN
    RETURN QUERY
    SELECT p.booking_id, p.status, p.amount
```

```
    FROM payments p
    WHERE p.user_id = _user;
END;
$$ LANGUAGE plpgsql;
```

Data Normalization

- 1st Normal Form
  - There is no comma-separated list of lists (For example, room name lists and lessons are separated)
  - Times is stored as timestamps

- 2nd Normal Form → No partial dependency
  - Composite keys are avoided to use in this project. Every table has their own corresponding primary key

- 3rd Normal Form → No transitive dependency
  - Paid_status function is stored in a different table (STUDENT) instead of LESSON table.
  - Duration function is stored COURSE instead of LESSON.

Data Integrity and security

- Each password required for the login process is stored as a hash.
- Emails are enforced to be unique
- Access to the main page is different depending on the role for the user (student and instructor)
- Triggers are used to eliminate invalid writes before committing. One example is the feature for time slots that are fixed for choosing, it does not allow the end user to write their own input.
- Room availability and student availability for time slots collision are prevented at the database level.
- Lesson status get auto - updated as either ‘completed’ or ‘missed’