HOSTEL MANAGEMENT SYSTEM

FINAL PROJECT REPORT

Description of the Chosen Business Scenario:

The business scenario involves the development of a "Hostel Management System" designed to efficiently manage hostel operations. The system caters to various types of hostels (Girls or Boys) with multiple wings and floors. Each floor contains multiple rooms, each with different occupancy types (Single, Double, Triple) and room types (AC or Non-AC), along with various room descriptions (e.g., Residential Room, Kitchen, Lounge). Administrative staff, categorized by types (e.g., Warden, Patron), oversee the hostel. Applicants can apply for hostel accommodation, and their applications are reviewed by administrators. If accepted, applicants may become residents, and the admins assign them rooms.

Business Rules:

1. **Hostel Type and Name**: Each hostel is categorized by its type (e.g., Girls, Boys) and has a unique name (e.g., Girls New Hostel, Girls Old Hostel, Boys Hostel).

2. **Wings**: Hostels consist of multiple wings, each with a unique name (e.g., West Wing, Capt. Haleem Wing).

3. **Floors:** Wings are composed of different floors, identified by floor numbers.

4. **Rooms:** Each floor has multiple rooms, each with a room number, occupancy type (Single, Double, Triple), room type (AC, Non-AC), and a description (e.g., Residential Room, Kitchen, Lounge).

5. **Applicant to Resident Transition**: Applicants can become residents upon approval by administrators.

Explanation of the Core Business Rules and Use Cases:

Applicant Use Cases:

1. Applicant Registration: Applicants can create an account with their personal details.

2. Hostel Application: Applicants can submit applications specifying preferences for hostels and rooms.

Admin Use Cases:

3. Admin Login: Administrative staff can log in with their credentials.

4. Administer Hostels: Administrators are responsible for overseeing and managing hostels.

5. Review Applications: Administrators review and approve or reject applicant applications, potentially assigning rooms.

6. Manage Room Details: Update room information, including occupancy type, room type, and room descriptions.

Resident Use Cases:

8. Resident Login: Residents can log in to access their information and perform actions like downloading fee vouchers or filing complaints/requests.

Entities, Attributes, and Relationships

1. Hostel

- Attributes:

- `id`: Primary key, numeric.
- `name`: Name of the hostel, text.
- `admin_id`: Foreign key referencing `Admin`, numeric.
- `address`: Address of the hostel, text.
- `type`: Type of the hostel, text.
- **Relationships:**
- One Hostel is managed by one Admin.
- One Hostel contains multiple Wings.

2. Wing

- Attributes:

- `id`: Primary key, numeric.
- `name`: Name of the wing, text.
- `hostel_id`: Foreign key referencing `Hostel`, numeric.
- **Relationships**:
- One Wing belongs to one Hostel.
- One Wing includes multiple Floors.

3. Floor

- Attributes:

- `id`: Primary key, numeric.
- `number`: Number of the floor, numeric.
- `wing_id`: Foreign key referencing `Wing`, numeric.
- **Relationships:**
- One Floor is part of one Wing.
- One Floor comprises multiple Rooms.

4. Room

- Attributes:

- `id`: Primary key, numeric.
- `number`: Room number, text.
- `room_type`: Type of room, text.
- `occupancy`: Occupancy status, text.
- `floor_id`: Foreign key referencing `Floor`, numeric.
- `student_id`: Foreign key referencing ‘Student’, numeric
- **Relationships:**
- One Room is situated on one Floor.

5. Complaint

- Attributes:

- `id`: Primary key, numeric.
- `description`: Description of the complaint, text (CLOB).
- `status`: Status of the complaint, text.
- `student_id`: Foreign key referencing `Student`, numeric.
- **Relationships:**
- One Complaint is filed by one Student.

6. Application

- Attributes:

- `id`: Primary key, numeric.
- `applicant_id`: Foreign key referencing `Student`, numeric.
- `room_type`: Desired room type, text.
- `occupancy`: Desired occupancy, text.
- `status`: Application status, numeric.
- **Relationships:**
- One Application is made by one Student.

7. User

- Attributes:

- `id`: Primary key, numeric.
- `username`: Username, text.
- `email`: Email address, text.
- `is_student`: Indicator if the user is a student, character.
- `is_admin`: Indicator if the user is an admin, character.
- **Relationships:**
- One User can be a Student or an Admin.

8. Student

- Attributes:

- `student_id`: Primary key, text.
- `name`: Student's name, text.
- `email`: Email address, text.
- `semester`: Current semester, numeric.
- `application_status`: Status of the hostel application, numeric.
- `user_id`: Foreign key referencing `User`, numeric.
- **Relationships:**
- One Student is a User.

9. Admin

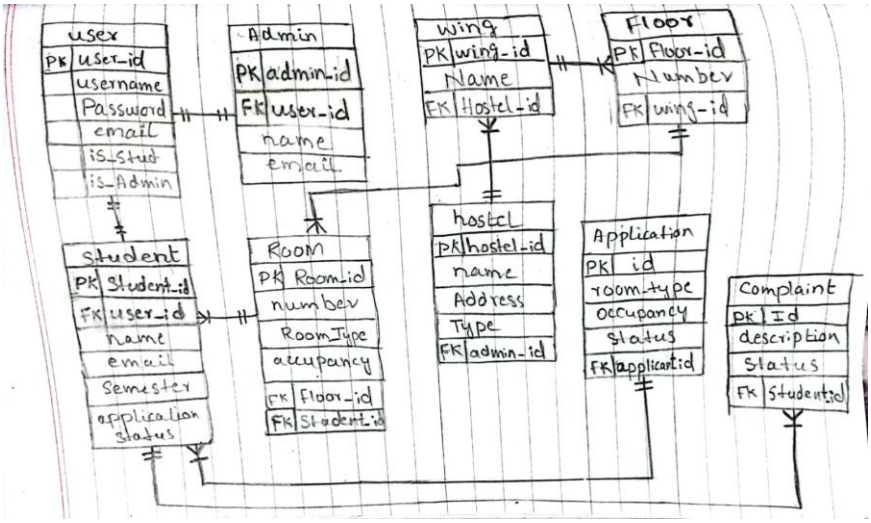
- Attributes:

- `admin_id`: Primary key, text.
- `name`: Admin's name, text.
- `email`: Email address, text.
- `user_id`: Foreign key referencing `User`, numeric.
- **Relationships:**
- One Admin is a User.

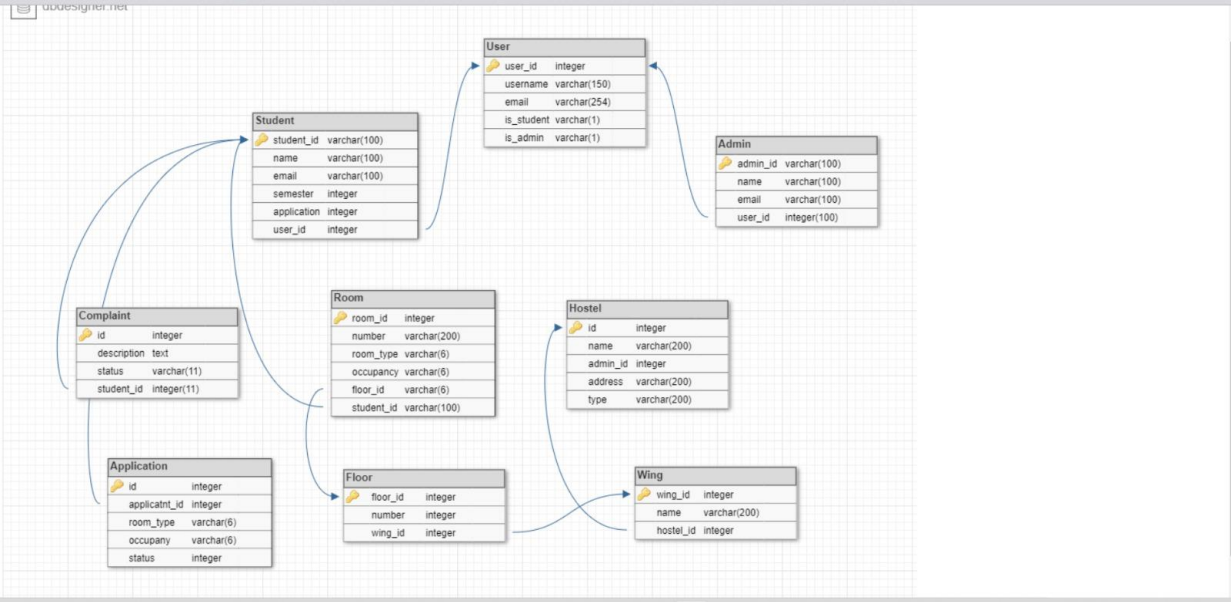
Multiplicity Constraints

- **One-to-Many:** Hostel to Wing, Wing to Floor, Floor to Room, Student to Complaint, Student to Application.
- **One-to-One**: Each Student and Admin is a User.
- **Foreign Key Constraints**: Each table with a foreign key (`admin_id`, `hostel_id`, `wing_id`, `floor_id`, `student_id`, `user_id`, `applicant_id`) references the primary key of the respective linked entity.

ERD:



Relational Schema:



TRIGGERS:

The following triggers are used in the application:

```
CREATE OR REPLACE TRIGGER set_student_id
BEFORE INSERT ON user_student
FOR EACH ROW
DECLARE
    username VARCHAR2(150);
BEGIN
    SELECT username INTO username FROM user_user WHERE id = :NEW.user_id;
    :NEW.student_id := username;
END;


CREATE OR REPLACE TRIGGER audit_student_changes
AFTER UPDATE OR DELETE ON user_student
FOR EACH ROW
BEGIN
    IF UPDATING THEN
        INSERT INTO audit_log(student_id, operation, change_time)
        VALUES (:OLD.student_id, 'UPDATE', SYSDATE);
    ELSIF DELETING THEN
        INSERT INTO audit_log(student_id, operation, change_time)
        VALUES (:OLD.student_id, 'DELETE', SYSDATE);
    END IF;
END;

CREATE OR REPLACE TRIGGER set_wing_name
BEFORE INSERT ON hostel_wing
FOR EACH ROW
DECLARE
    hostel_name VARCHAR2(200);
BEGIN
    SELECT name INTO hostel_name FROM hostel_hostel WHERE id = :NEW.hostel_id;
    :NEW.name := hostel_name || ' ' || :NEW.name;
END;


CREATE OR REPLACE TRIGGER update_student_application_status
AFTER UPDATE OF status ON hostel_application
FOR EACH ROW
BEGIN
    UPDATE user_student
    SET application_status = :NEW.status
    WHERE user_id = :NEW.applicant_id;
END
```

Trigger: audit_student_changes

This is used in areas involving updates or deletions of student information, which is the "Admin Dashboard" where admins manage student applications and complaints. This trigger would log changes in student records after administrators update or delete them.

Trigger: set_wing_name

This trigger would be active in the admin functionalities, when an admin adds new wings to a hostel. When a new wing is inserted in the database via the admin interface, this trigger would automatically set its name based on the related hostel.

Trigger: update_student_application_status

Used in the application review process by the admin, as described under "Administer Hostels" and "Review Applications". When an admin updates the status of a student's hostel application, this trigger would automatically update the application status in the student's record.

Trigger: set_student_id

The "set_student_id" trigger is used on the "Sign Up" screen of the application, where new students register. It sets the student's ID to their username as part of the account creation process in the `user_student` table.

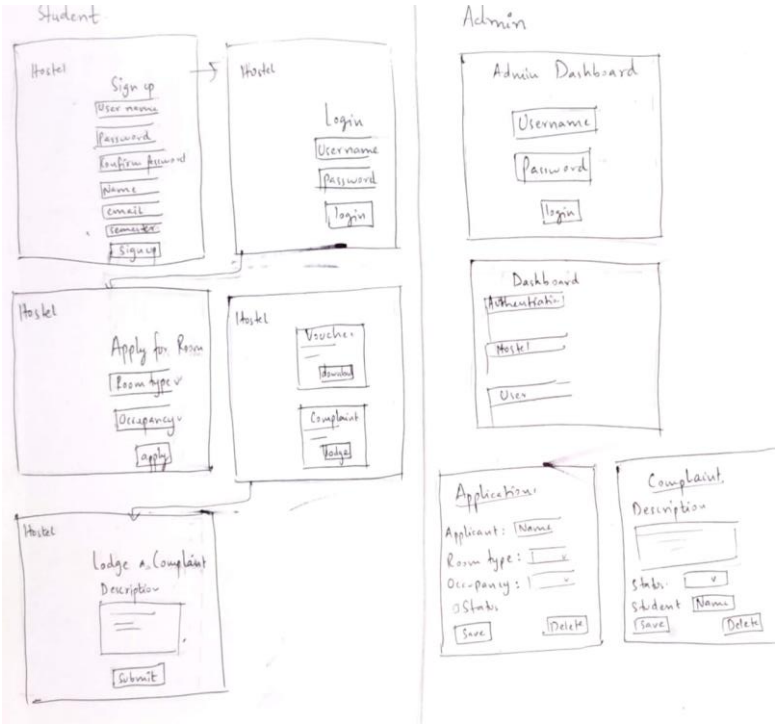
DDL FOR TABLE CREATION:

```
CREATE TABLE Hostel (
    id NUMBER(11) PRIMARY KEY,
    name VARCHAR2(200) NOT NULL,
    admin_id NUMBER(11) NOT NULL,
    address VARCHAR2(200) NOT NULL,
    type VARCHAR2(6) NOT NULL,
    CONSTRAINT fk_admin FOREIGN KEY (admin_id) REFERENCES Admin(id)
);

CREATE TABLE Wing (
    id NUMBER(11) PRIMARY KEY,
    name VARCHAR2(200) NOT NULL,
    hostel_id NUMBER(11) NOT NULL,
    CONSTRAINT fk_hostel FOREIGN KEY (hostel_id) REFERENCES Hostel(id)
);

CREATE TABLE Floor (
    id NUMBER(11) PRIMARY KEY,
    number NUMBER(11) NOT NULL,
    wing_id NUMBER(11) NOT NULL,
    CONSTRAINT fk_wing FOREIGN KEY (wing_id) REFERENCES Wing(id)
);

CREATE TABLE Room (
    id NUMBER(11) PRIMARY KEY,
    number VARCHAR2(200) NOT NULL,
    room_type VARCHAR2(6) NOT NULL,
    occupancy VARCHAR2(6) NOT NULL,
    floor_id NUMBER(11) NOT NULL,
    CONSTRAINT fk_floor FOREIGN KEY (floor_id) REFERENCES Floor(id)
    CONSTRAINT stu_id FOREIGN KEY (student_id) REFERENCES Student(student_id)
);

CREATE TABLE Complaint (
    id NUMBER(11) PRIMARY KEY,
    description CLOB NOT NULL,
    status VARCHAR2(11) NOT NULL,
    student_id NUMBER(11) NOT NULL,
    CONSTRAINT fk_student FOREIGN KEY (student_id) REFERENCES Student(id)
);
```

```sql
CREATE TABLE Application (
    id NUMBER(11) PRIMARY KEY,
    applicant_id NUMBER(11) NOT NULL,
    room_type VARCHAR2(6) NOT NULL,
    occupancy VARCHAR2(6) NOT NULL,
    status NUMBER(1) NOT NULL,
    CONSTRAINT fk_applicant FOREIGN KEY (applicant_id) REFERENCES Student(id)
);

CREATE TABLE User (
    id NUMBER GENERATED BY DEFAULT ON NULL AS IDENTITY,
    username VARCHAR2(150) NOT NULL,

    email VARCHAR2(254) NOT NULL,

    is_student CHAR(1) DEFAULT '0' NOT NULL,
    is_admin CHAR(1) DEFAULT '0' NOT NULL,
    PRIMARY KEY (id)
);

CREATE TABLE Student (
    student_id VARCHAR2(100) PRIMARY KEY,
    name VARCHAR2(100) NOT NULL,
    email VARCHAR2(100) NOT NULL,
    semester NUMBER(11) NOT NULL,
    application_status NUMBER(1) NOT NULL,
    user_id NUMBER(11) NOT NULL,
    CONSTRAINT fk_user FOREIGN KEY (user_id) REFERENCES User(id)
);

CREATE TABLE Admin (
    admin_id VARCHAR2(100) PRIMARY KEY,
    name VARCHAR2(100) NOT NULL,
    email VARCHAR2(100) NOT NULL,
    user_id NUMBER(11) NOT NULL,
    CONSTRAINT fk_user FOREIGN KEY (user_id) REFERENCES User(id)
);
```

Application Flow:

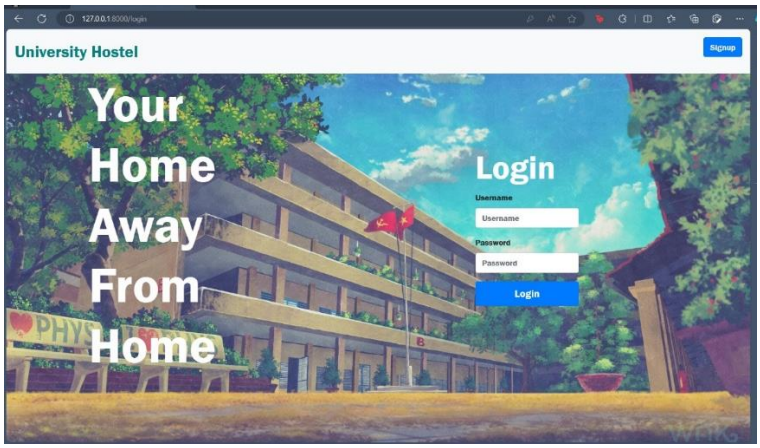
Wireframes:



Page-by-Page Navigation and SQL Queries:

Home page:

The first screen is the home page with login and signup options as shown in the picture below.

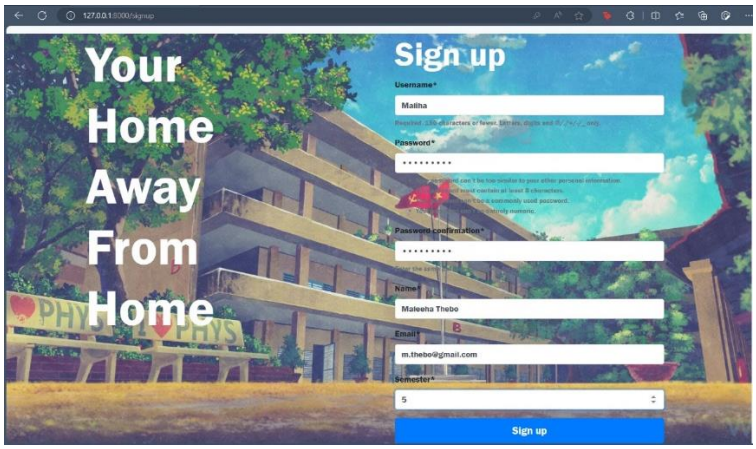


Sign Up page:

Next if the user/ student navigates to sign up, the following sign-up form shows up. Here the trigger **set_student_id** is used.

```
-- INSERT
INSERT INTO user_Student (user_id, student_id, name, email, semester, application_status)
VALUES (1, 'Student ID', 'Name', 'Email', 1, '0');
```

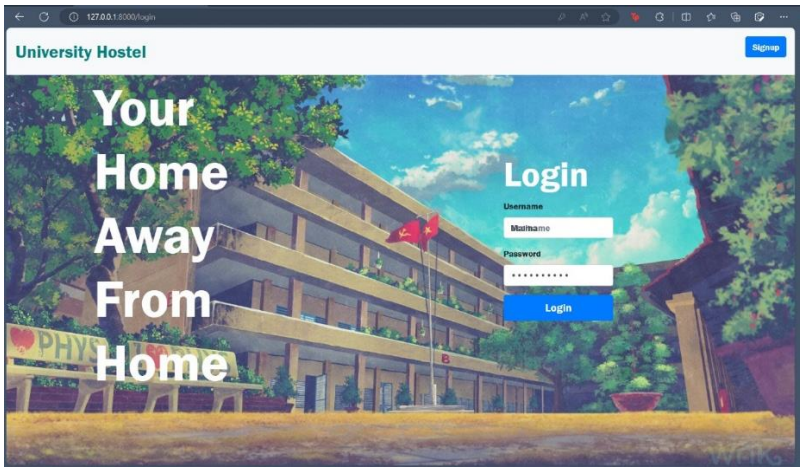
Above query used in the "Student Sign-up" screen for new student account creation.

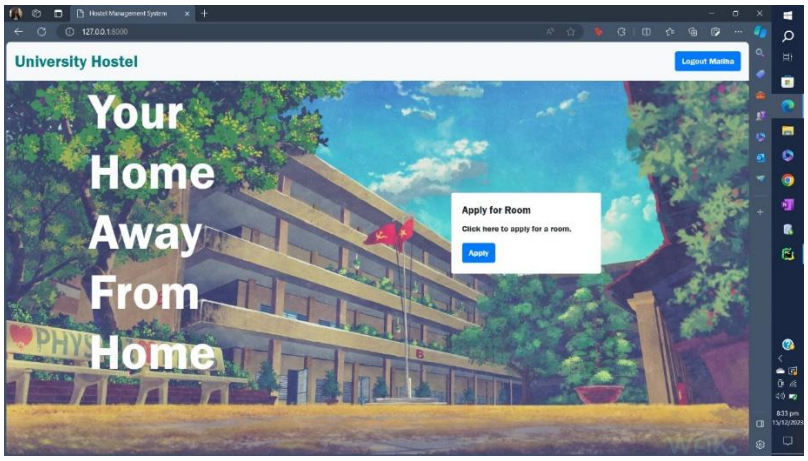


Filling in details, and submitting the form, the account is created.

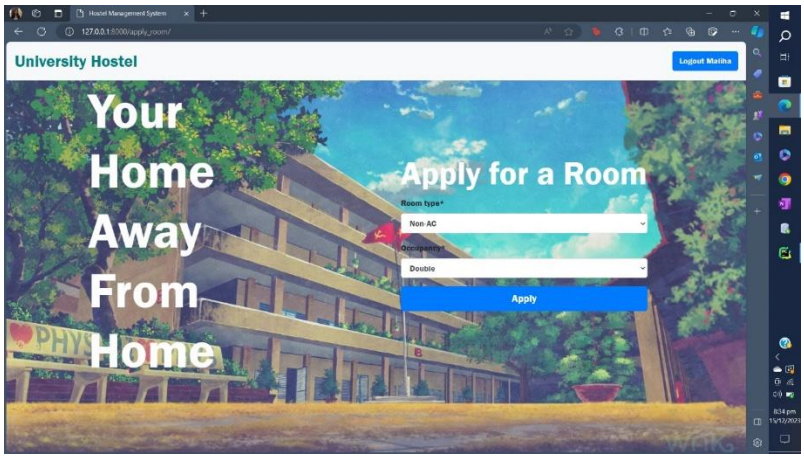
Login Page:

Navigating back to the login page, now if the valid credentials are entered the student can apply for hostel (if new applicant, who just created an account and is not a resident yet).





Filling the application with preferences and submitting.



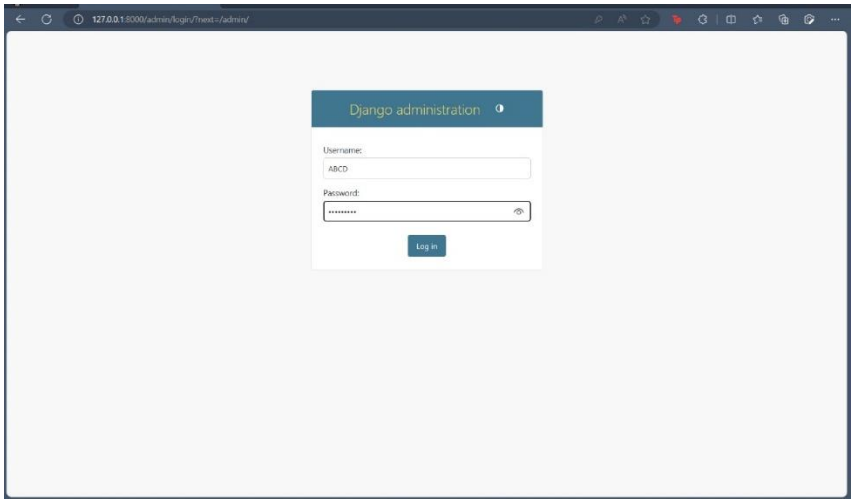

```
-- INSERT
INSERT INTO Hostel_Application (applicant_id, room_type, occupancy, status)
VALUES (1, 'AC', 'Single', '0');
```

Used on the above screen for students to apply for hostel accommodation.

Admin Login Page:

NOTE: A new admin is created initially using “Create superuser” command in terminal.

For existing admin, valid credentials can be added, which then navigate to the dashboard.



Admin Dashboard:

```
set define off;

  CREATE OR REPLACE NONEDITIONABLE PROCEDURE "SYSTEM"."FETCH_APPLICATIONS" (p_applications OUT SYS_REFCURSOR)
IS
BEGIN
    OPEN p_applications FOR SELECT * FROM hostel_application;
END;

/
--------------------------------------------------------
--  DDL for Procedure GET_ALL_COMPLAINTS
--------------------------------------------------------
set define off;

  CREATE OR REPLACE NONEDITIONABLE PROCEDURE "SYSTEM"."GET_ALL_COMPLAINTS" (complaints_cursor OUT SYS_REFCURSOR) AS
BEGIN
  OPEN complaints_cursor FOR
  SELECT c.id, c.description, c.status, s.name
  FROM hostel_complaint c
  JOIN user_student s ON c.student_id = s.user_id;
END get_all_complaints;
```

Here, the admin can see all the applications and complaints and for that purpose the following procedures are used to retrieve applications and complaints from respective tables.

And admin can approve or reject those applications and complaints.

The triggers “**audit_student_changes**” and “**set_wing_name**” are used here and are described above.

```
-- DELETE
DELETE FROM Hostel_Application
WHERE id = 1;
```

Utilized here in the "Admin Dashboard" for administrators to remove specific hostel applications.

```
-- UPDATE
UPDATE Hostel_Application
SET room_type = 'Non-AC', occupancy = 'Double', status = '1'
WHERE id = 1;
```

Used in the "Admin Dashboard" to modify existing hostel applications by the administrator.

```
-- DELETE
DELETE FROM user_Student
WHERE user_id = 1;
```

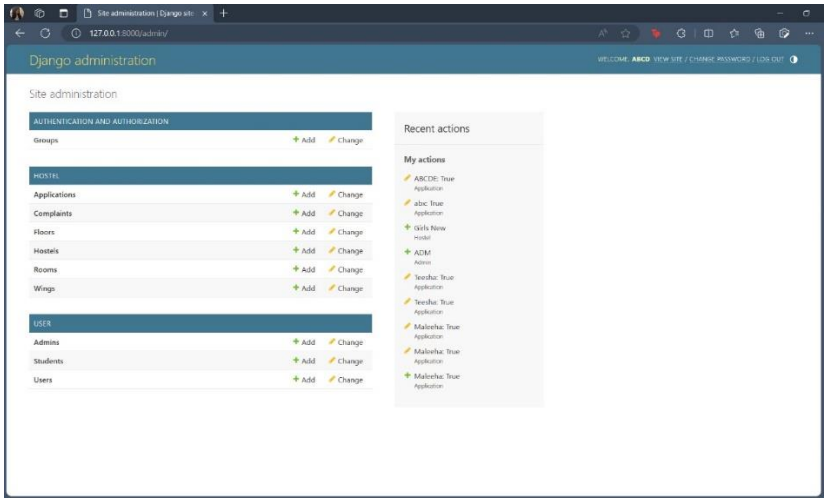
Used by administrators in the "Admin Dashboard" to delete student accounts.

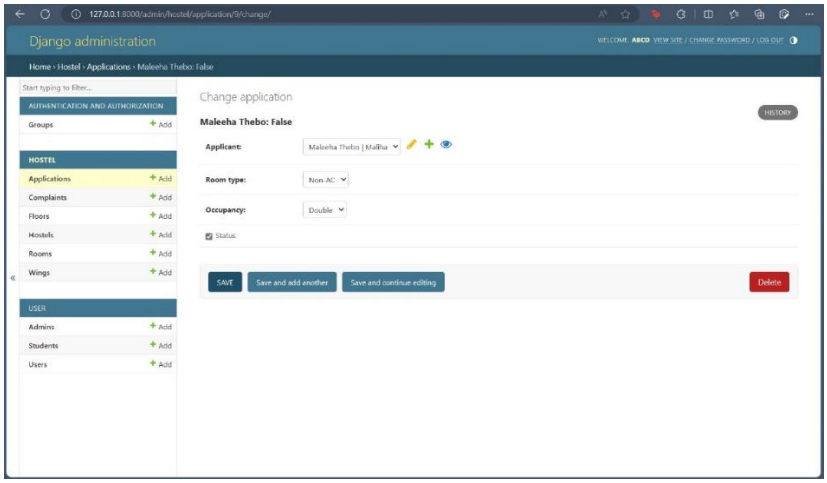
INSERT INTO user_Admin (user_id, admin_id, name, email) VALUES (1, 'Admin ID', 'Name', 'Email'); Used in the "Admin Dashboard" to create new admin.

Similarly, DELETE FROM user_Admin WHERE user_id = 1; and UPDATE user_Admin SET admin_id = 'New Admin ID', name = 'New Name', email = 'New Email' WHERE user_id = 1; are used delete and update any admin record, in the admin dashboard.

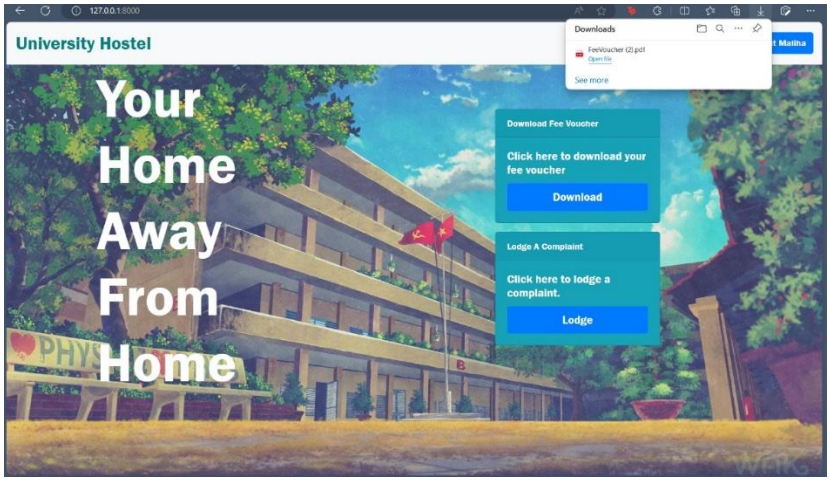
The following other queries are used in the "Dashboard", for respective components:

- INSERT INTO "HOSTEL_HOSTEL" (name, admin_id, address, type) VALUES ('"HOSTEL_HOSTEL" Name', 1, 'Hostel Address', 'Boys'); Used in the "Hostel Setup" by admins.
- DELETE FROM "HOSTEL_HOSTEL" WHERE id = 1; On the "Admin Hostel Management" screen for removing hostels.
- UPDATE "HOSTEL_HOSTEL" SET name = 'New "HOSTEL_HOSTEL" Name', address = 'New Hostel Address', type = 'Girls' WHERE id = 1; In the "Hostel Details Update" section.
- INSERT INTO Hostel_Wing (name, hostel_id) VALUES ('Wing Name', 1); Used in the "Wing Addition" interface.
- DELETE FROM Hostel_Wing WHERE id = 1; On the "Wing Management" page for deleting wings.
- UPDATE Hostel_Wing SET name = 'New Wing Name', hostel_id = 2 WHERE id = 1; In the "Wing Update" section.
- INSERT INTO Hostel_Floor (number, wing_id) VALUES (1, 1); Utilized in "Floor Addition" by admins.
- DELETE FROM Hostel_Floor WHERE id = 1; On the "Floor Management" interface for removing floors.
- UPDATE Hostel_Floor SET number = 2, wing_id = 2 WHERE id = 1; Employed in the "Floor Update" section for altering floor details.

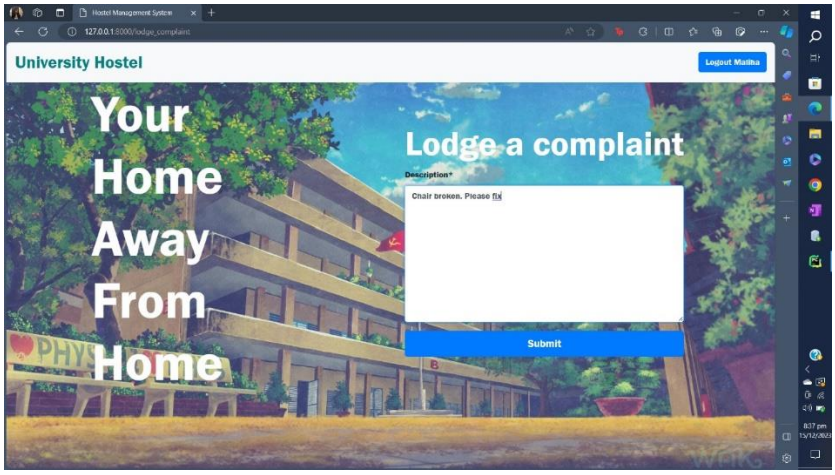




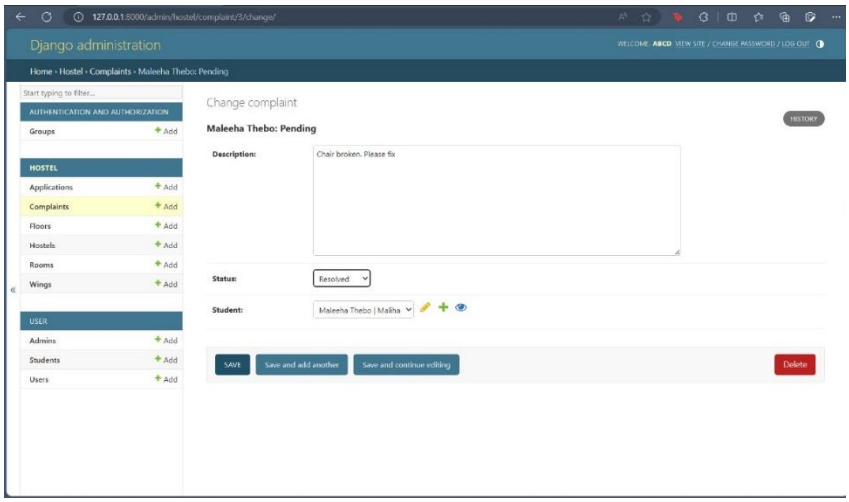
Once the application is accepted. One can login again as a student and see two options to download the fee voucher or to file a complaint. Here the "**update_student_application_status**" trigger is used, which is defined above.



INSERT INTO "HOSTEL_COMPLAINT" ("DESCRIPTION", "STATUS", "STUDENT_ID") VALUES ('Dummy Complaint', Pending, 1); Used on the "Student Complaint Submission" interface.

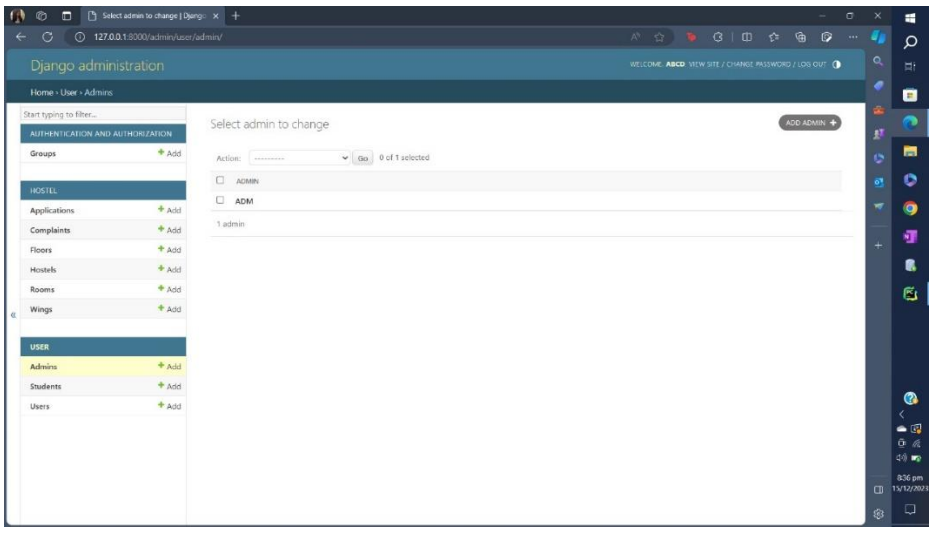


Once this complaint is submitted, it can be reflected on the admin dashboard, where its status can change.

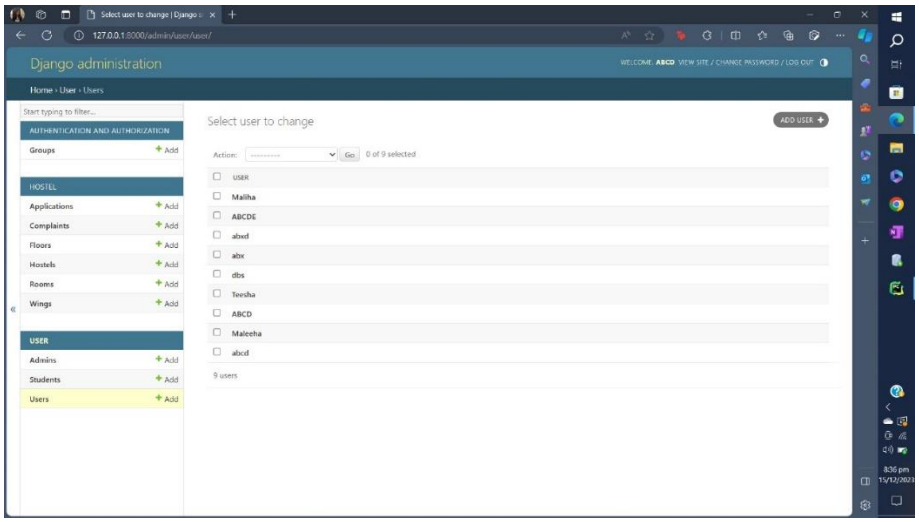


Following are the users that have been created.

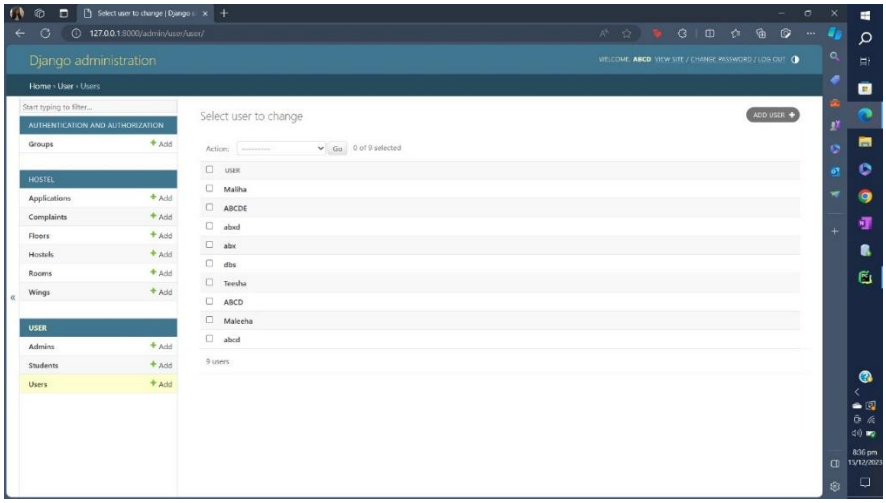
Admins:



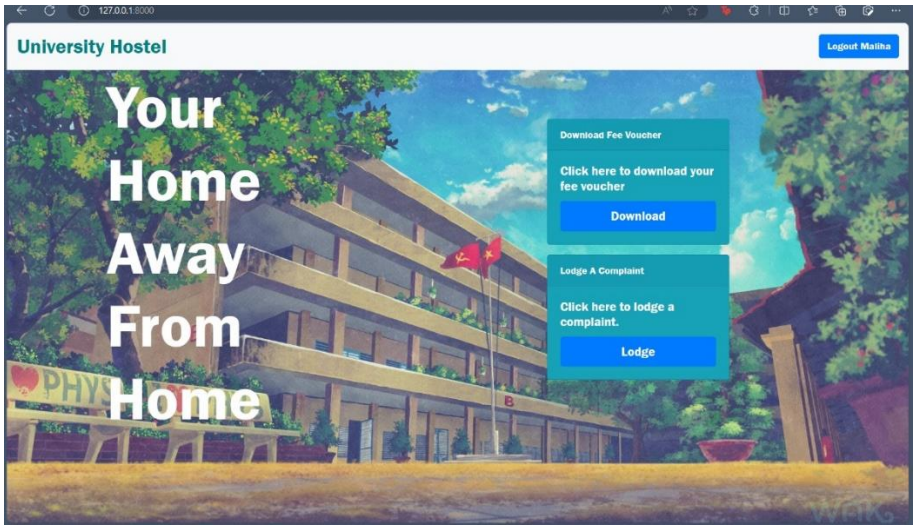
All users:



Students:



Lastly, we can logout using the logout button on top, like:



Framework:

The Django framework, as used in this project, is a high-level Python web framework that encourages rapid development and clean, pragmatic design. It is known for its simplicity, flexibility, and reliability. In this Hostel Management System, Django likely serves as the backbone, handling HTTP requests, rendering pages, managing data models and interactions with the database, and ensuring security and session management. Its use in this project would streamline the development process, facilitate the creation of dynamic web pages, and manage data effectively through its ORM (Object-Relational Mapping) capabilities. Django's modular architecture also makes it easier to maintain and upgrade the system over time.