# CPT_S 451: Final Project Report

# Room Booking System

Member Names:

Felix Zheng (011799245)

Jacob Madison (011737149)

Khushi Panchal (011829256)

Trisha Teredesai (011811221)

Instructor’s Name: Parteek Kumar

Submission Date: April 27, 2025

## Abstract

The objective of this project is to create a dorm management system in order to streamline the process of assigning rooms to students at a university. This application allows students to browse available residence halls and dorm rooms, and apply for the room they want based on their preferences. It also allows administrators to view incoming applications and manage room assignments & maintenance requests. The expected outcome of this project is to create an efficient system to manage large volumes of student room assignments, with a dynamically updating database which streamlines the process for all parties involved.

## Table of Contents

## Introduction

Our project ‘Room Booking System’ is a dorm finder management system that students can use to apply and select dorm rooms, which also incorporates preference-based roommate matching. With this project we hope to create an easy-to-use platform that students can use to find their desired dorm rooms and roommates. We also aim to make it simple for administrators to manage student applications as well as students currently residing in the dorms.

Some of our objectives are:

- Give students the ability to browse and apply to dorms based on their preferences.
- Give administrators the ability to manage dorm residents and room availability.
- Create a database to store student data such as personal information, application status and room assignments.
- Real-time updates on room availability.

This project could improve dorm room searching by allowing students to quickly browse available dorm rooms, while ensuring that administrators can efficiently manage room assignments. It would enhance productivity and optimize the dorm room assignment process.

## Functional Requirements

Here are the functional requirements of our system:

- Student Registration & Login: Ensures that only verified students are able to access the application, allowing students to secure their information.
- Room Database Management: Maintains a comprehensive database of all residence halls and available rooms, including details such as roomsize, occupancy and furnishings.
- Room Browsing: Allows students to explore the available rooms based on their preferences. Students can also filter the rooms based on features, making it easier to find what they are looking for.
- Room Booking: Allows students to make an application to reserve the room they would like to live in.
- Preference-based Roommate Matching: Students can provide details about their living habits, which is used to match potential roommates.
- Maintenance Requests: Students can place maintenance requests to report issues and request for repair services.

- Admin Room Booking Management: The administrator is able to view, manage and manipulate room booking applications. The administrator can also assign rooms to each student.
- Maintenance Request Management: The administrator can track and manage maintenance requests, ensuring quick resolution of reported issues.
- Room Status Updates: The room availability is automatically updated after booking & cancelling making sure that the data is consistent and reliable.
- Search/Filter Functionality: Students can filter the rooms based on specific criteria, saving time and making it easier for students to narrow down their options.
- User Roles: Establishes clear roles for students, administrators, and maintenance staff, making sure that each user has access to features pertaining to their responsibilities.
- User Permissions: Implants role-based access controls based on user roles to protect sensitive data.

**Non-Functional Requirements**

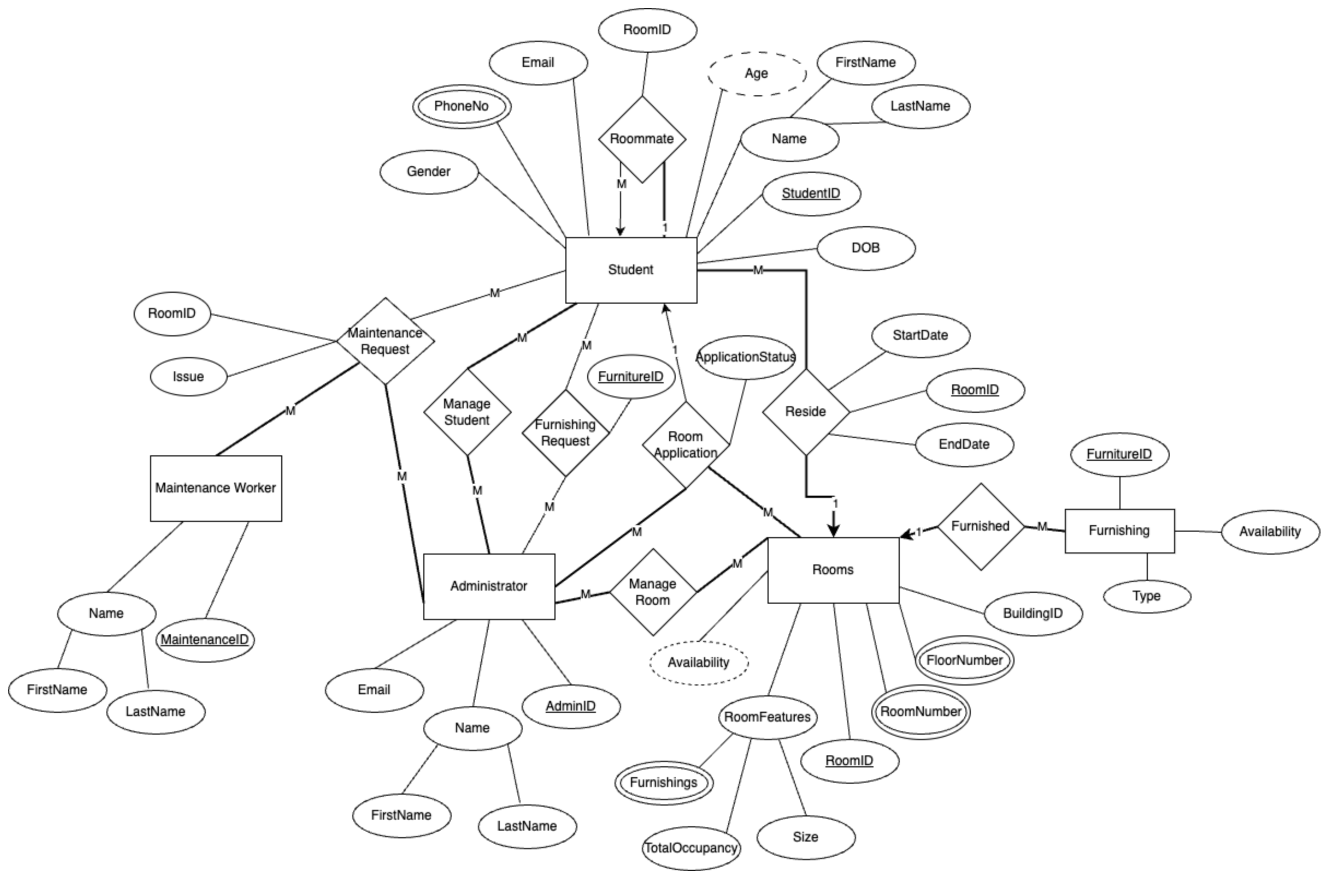
Here are the non-functional requirements of our system:

- Efficient Performance: The system should be able to handle multiple concurrent users and large computations without significant slowdown.
- High Security: The system should be sure with proper permissions protecting the user's sensitive information.
- Workplace Compliance: The system should be compliant with all policies that involve employee viewing privileges and client information, which maintains the users privacy.
- Scalability: The system should have the ability to scale with any future data growth, and be able to accommodate changing amounts of data.
- Accuracy and reliability: The system shall properly and accurately sort data at a reliable rate, never returning false or incorrect data.
- Easy accessibility: The system will have a user-friendly interface that those unfamiliar with database operations can easily manage.
- Regular backups & Recovery: The system will have an automated backup system to ensure recovery in the possible event of data loss. This is necessary since the business should be able to run as expected regardless of server issues.

- ■ Availability: The system must always be available for use. Our clients need to be able to access the server so they can book at their convenience.
- ■ Durability: The system will be able to keep up and be easily adapted with software evolution in the future.
- ■ Maintainability: The system will be maintained at all times. The system will be managed by database administrators, who will update the system as the customer's needs change.
- ■ Agility: The system will be able to be updated at reasonable intervals to keep up with changes in the bookings.

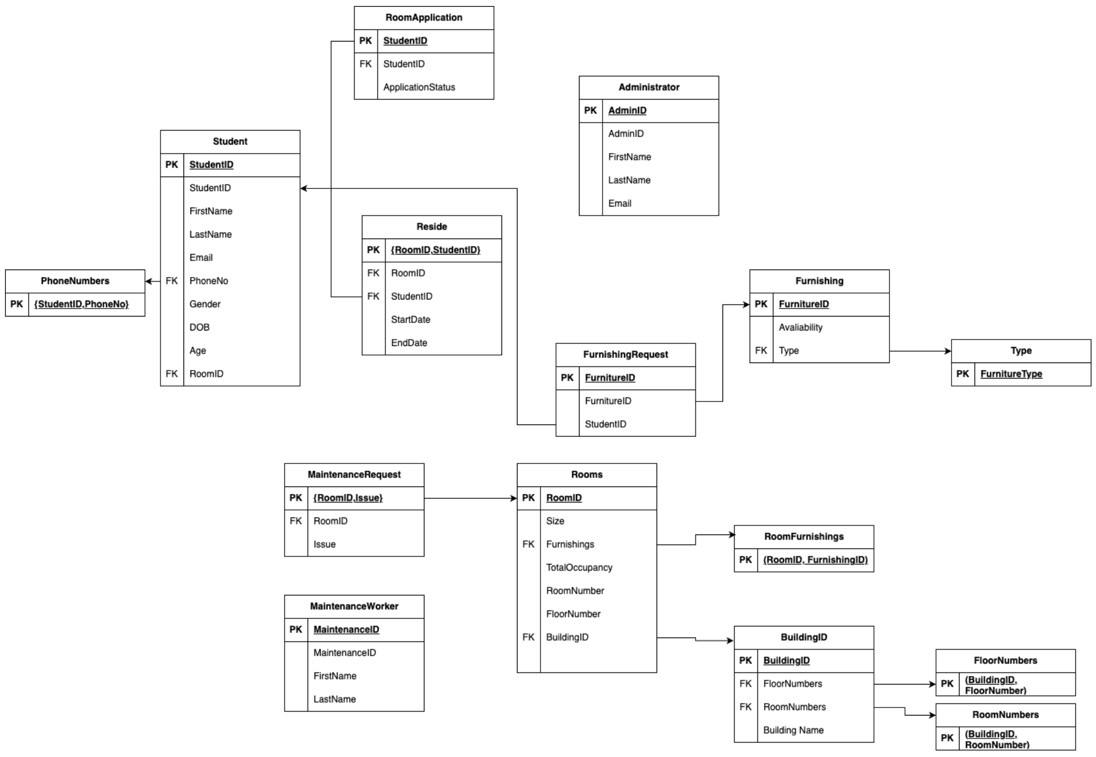
**Database Design**

ER Diagram:



As per our project requirements, we identified five entities, namely Student, Admin, Maintenance Worker, Room and Furnishing. To determine the relationships between the entities, we considered the actions each user would need to perform and how each entity interacts with one another.

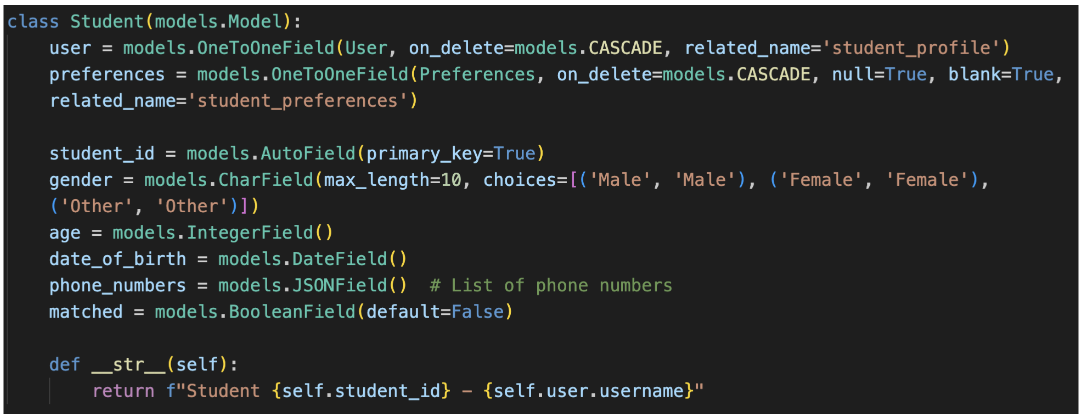
Conversion to Relational Tables:



To convert an ER diagram into relational tables, the entity sets, relationships, and attributes became tables, foreign keys, and table columns. Each table contains its respective attributes as indicated in the ER diagram. Relationships between tables is represented by a foreign key attribute which points to another table containing the primary keys of the involved entities and any relationship attributes. Multi-value attributes are also represented by a foreign key that references another table containing the primary key and value pairs.

## SQL Implementation

First, we started off by designing our tables in SQL, which can be seen in RoomBooking.sql Next, we created Django models using Django ORM based off of the tables. Here are some of the key tables in the application, which can be seen in RoomBookingWebsite > models.py:



```python
class Student(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE, related_name='student_profile')
    preferences = models.OneToOneField(Preferences, on_delete=models.CASCADE, null=True, blank=True,
    related_name='student_preferences')

    student_id = models.AutoField(primary_key=True)
    gender = models.CharField(max_length=10, choices=[('Male', 'Male'), ('Female', 'Female'),
    ('Other', 'Other')])
    age = models.IntegerField()
    date_of_birth = models.DateField()
    phone_numbers = models.JSONField()  # List of phone numbers
    matched = models.BooleanField(default=False)

    def __str__(self):
        return f"Student {self.student_id} - {self.user.username}"
```

*CREATE TABLE student (*

*student_id SERIAL PRIMARY KEY,*

*user_id INTEGER NOT NULL, -- Foreign key to User model*

*gender VARCHAR(10) CHECK (gender IN ('Male', 'Female', 'Other')),*

*age INTEGER,*

*date_of_birth DATE,*

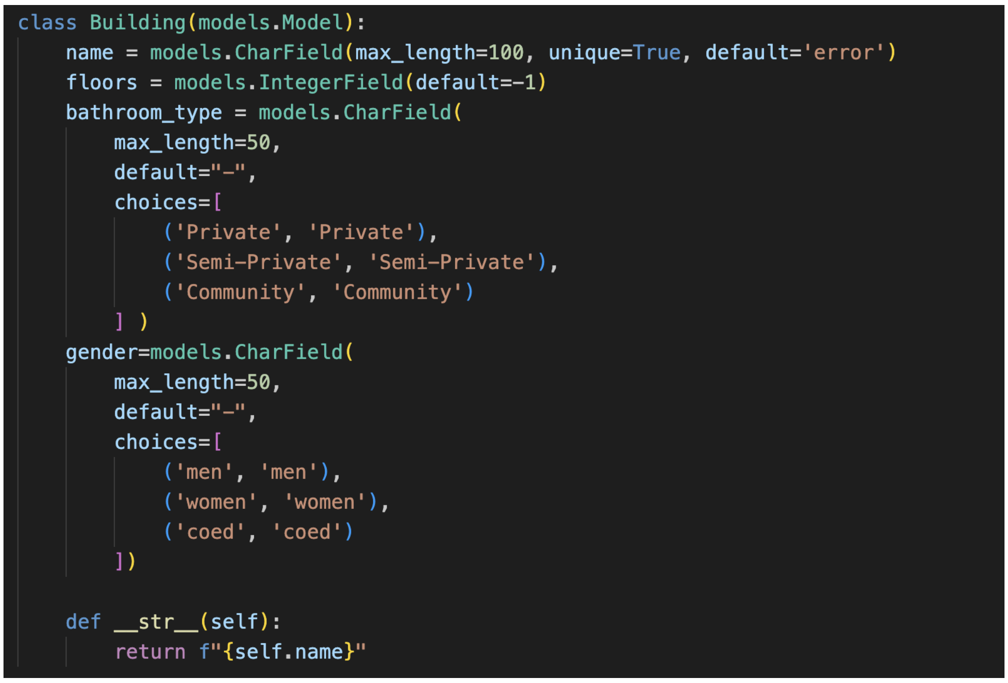
*phone_numbers JSONB,*

*matched BOOLEAN DEFAULT FALSE,*

*preferences_id INTEGER, -- Foreign key to Preferences model*

*CONSTRAINT fk_user FOREIGN KEY (user_id) REFERENCES auth_user(id),*

*CONSTRAINT fk_preferences FOREIGN KEY (preferences_id) REFERENCES preferences(id));*



```python
class Building(models.Model):
    name = models.CharField(max_length=100, unique=True, default='error')
    floors = models.IntegerField(default=-1)
    bathroom_type = models.CharField(
        max_length=50,
        default="-",
        choices=[
            ('Private', 'Private'),
            ('Semi-Private', 'Semi-Private'),
            ('Community', 'Community')
        ] )
    gender=models.CharField(
        max_length=50,
        default="-",
        choices=[
            ('men', 'men'),
            ('women', 'women'),
            ('coed', 'coed')
        ])

    def __str__(self):
        return f"{self.name}"
```

*CREATE TABLE building (*

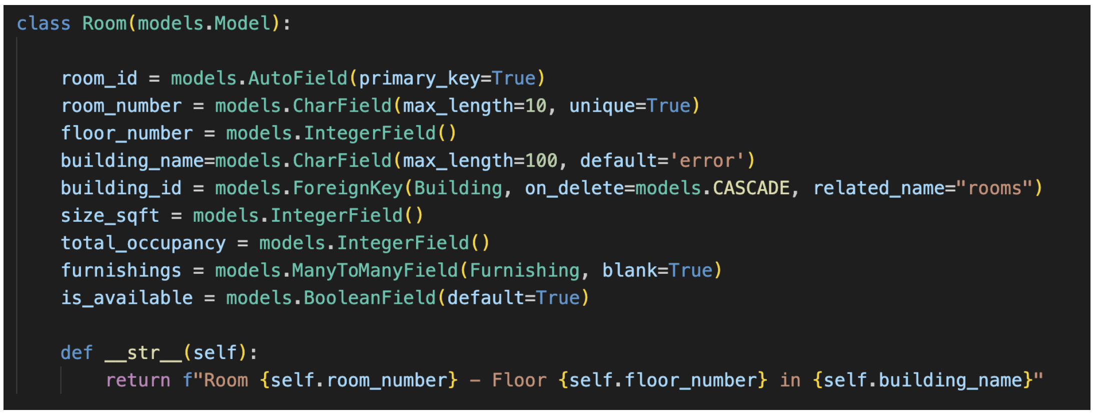
*id SERIAL PRIMARY KEY,*

*name VARCHAR(100) UNIQUE DEFAULT 'error',*

*floors INTEGER DEFAULT -1,*

*bathroom_type VARCHAR(50) CHECK (bathroom_type IN ('Private', 'Semi-Private', 'Community')),*

*gender VARCHAR(50) CHECK (gender IN ('men', 'women', 'coed')));*



```python
class Room(models.Model):

    room_id = models.AutoField(primary_key=True)
    room_number = models.CharField(max_length=10, unique=True)
    floor_number = models.IntegerField()
    building_name=models.CharField(max_length=100, default='error')
    building_id = models.ForeignKey(Building, on_delete=models.CASCADE, related_name="rooms")
    size_sqft = models.IntegerField()
    total_occupancy = models.IntegerField()
    furnishings = models.ManyToManyField(Furnishing, blank=True)
    is_available = models.BooleanField(default=True)

    def __str__(self):
        return f"Room {self.room_number} - Floor {self.floor_number} in {self.building_name}"
```

*CREATE TABLE room (*

*room_id SERIAL PRIMARY KEY,*

*room_number VARCHAR(10) UNIQUE,*

*floor_number INTEGER,*

*building_name VARCHAR(100) DEFAULT 'error',*

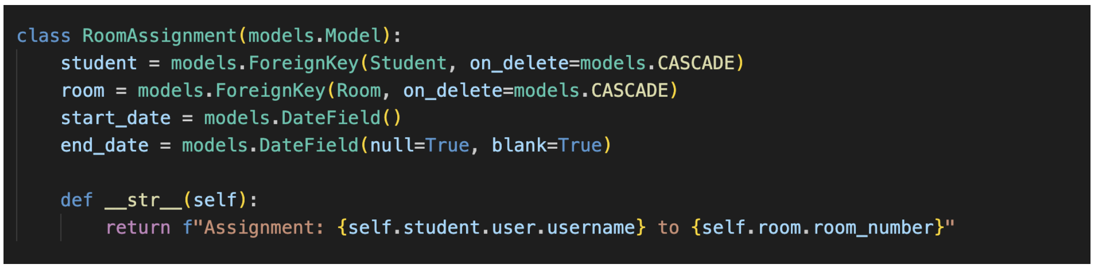
*building_id INTEGER,*

*size_sqft INTEGER,*

*total_occupancy INTEGER,*

*is_available BOOLEAN DEFAULT TRUE,*

*CONSTRAINT fk_building FOREIGN KEY (building_id) REFERENCES building(id));*



```
class RoomAssignment(models.Model):
    student = models.ForeignKey(Student, on_delete=models.CASCADE)
    room = models.ForeignKey(Room, on_delete=models.CASCADE)
    start_date = models.DateField()
    end_date = models.DateField(null=True, blank=True)

    def __str__(self):
        return f"Assignment: {self.student.user.username} to {self.room.room_number}"
```

*CREATE TABLE room_assignment (*

*id SERIAL PRIMARY KEY,*

*student_id INTEGER NOT NULL,*

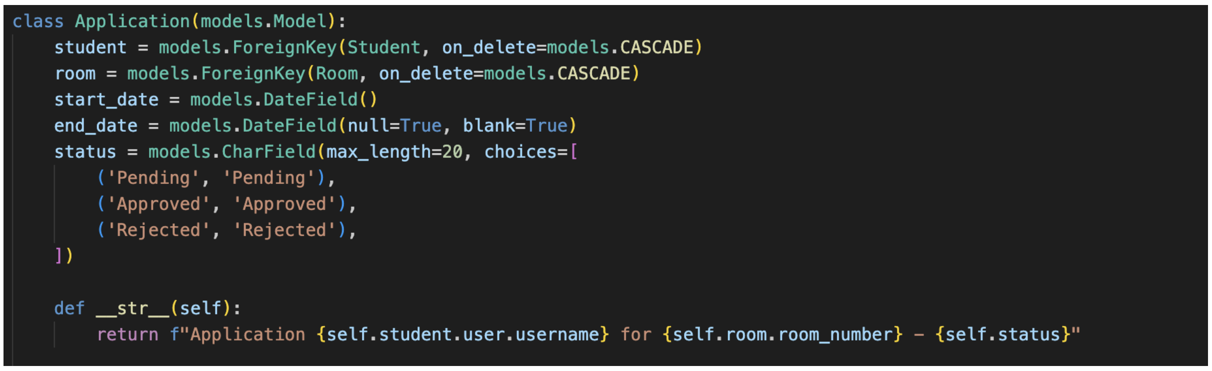
*room_id INTEGER NOT NULL,*

*start_date DATE,*

*end_date DATE,*

*CONSTRAINT fk_student FOREIGN KEY (student_id) REFERENCES student(student_id),*

*CONSTRAINT fk_room FOREIGN KEY (room_id) REFERENCES room(room_id));*



```
class Application(models.Model):
    student = models.ForeignKey(Student, on_delete=models.CASCADE)
    room = models.ForeignKey(Room, on_delete=models.CASCADE)
    start_date = models.DateField()
    end_date = models.DateField(null=True, blank=True)
    status = models.CharField(max_length=20, choices=[
        ('Pending', 'Pending'),
        ('Approved', 'Approved'),
        ('Rejected', 'Rejected'),
    ])

    def __str__(self):
        return f"Application {self.student.user.username} for {self.room.room_number} - {self.status}"
```

*CREATE TABLE application (*

*id SERIAL PRIMARY KEY,*

*student_id INTEGER NOT NULL,*

*room_id INTEGER NOT NULL,*

*start_date DATE,*

*end_date DATE,*

*status VARCHAR(20) CHECK (status IN ('Pending', 'Approved', 'Rejected')),*

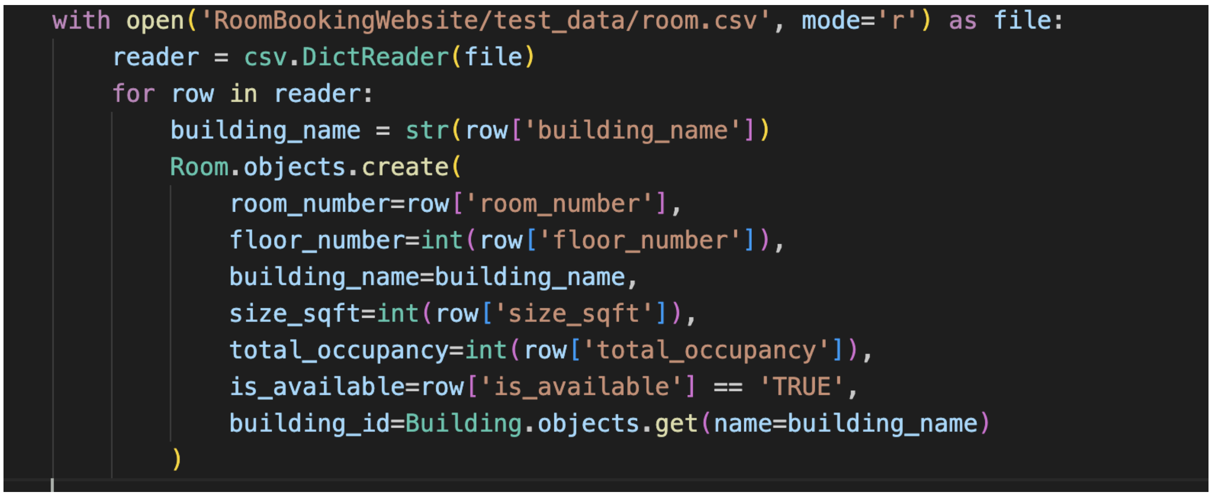
*CONSTRAINT fk_student_application FOREIGN KEY (student_id) REFERENCES student(student_id),* *CONSTRAINT fk_room_application FOREIGN KEY (room_id) REFERENCES room(room_id));*

Sample data was created in RoomBookingWebsite > testdata as .csv files.

DML scripts:

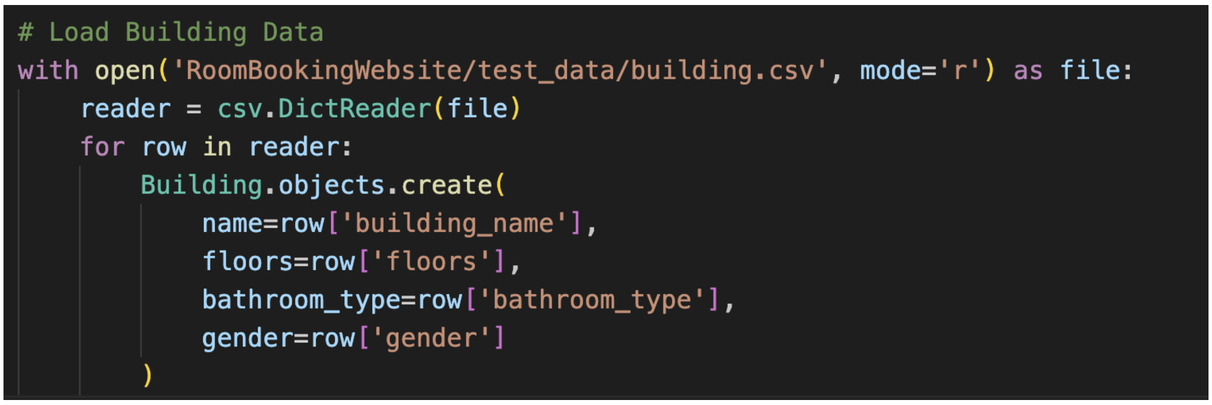
Can be seen at RoomBookingWebsite > management > commands > test_data_from_csv.py

An example can be seen below, the function reads from the csv files and then inserts into the table.



```
with open('RoomBookingWebsite/test_data/room.csv', mode='r') as file:
    reader = csv.DictReader(file)
    for row in reader:
        building_name = str(row['building_name'])
        Room.objects.create(
            room_number=row['room_number'],
            floor_number=int(row['floor_number']),
            building_name=building_name,
            size_sqft=int(row['size_sqft']),
            total_occupancy=int(row['total_occupancy']),
            is_available=row['is_available'] == 'TRUE',
            building_id=Building.objects.get(name=building_name)
        )
```

*INSERT INTO Room(room_number,floor_number,building_name,size_sqft,total_occupancy,is_available,building_id)*

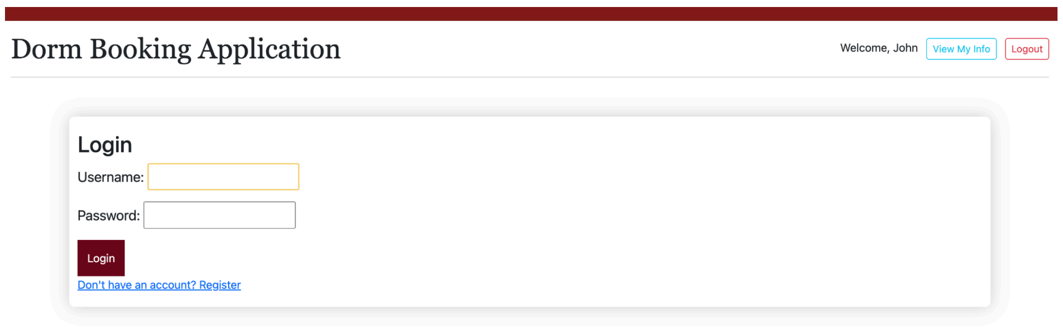


```
# Load Building Data
with open('RoomBookingWebsite/test_data/building.csv', mode='r') as file:
    reader = csv.DictReader(file)
    for row in reader:
        Building.objects.create(
            name=row['building_name'],
            floors=row['floors'],
            bathroom_type=row['bathroom_type'],
            gender=row['gender']
        )
```

*INSERT INTO Building VALUES(building_name, floors, bathroom_type, gender)*

## Feature Implementations

Login/Registration:



Dorm Booking Application

Welcome, John View My Info Logout

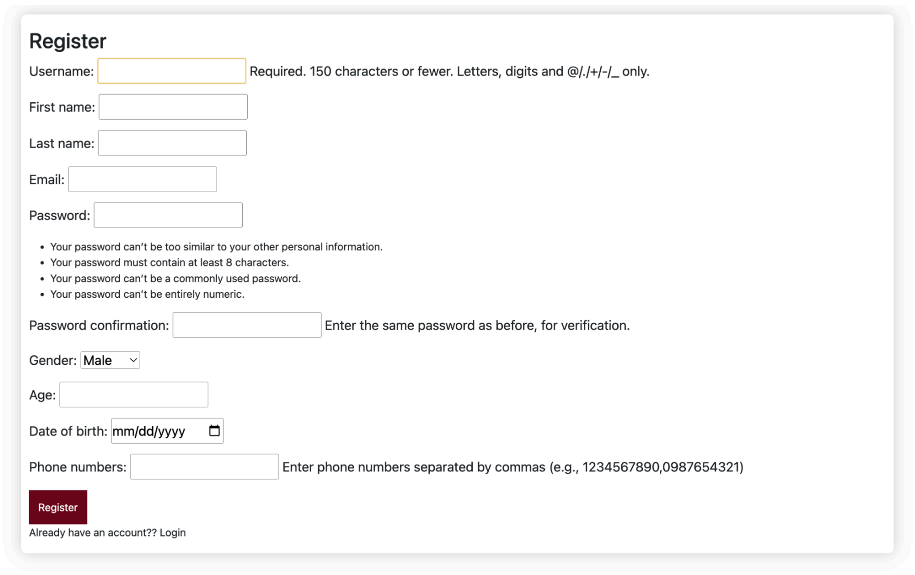
Login

Username:

Password:

Login

Don't have an account? Register



Register

Username: Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only.

First name:

Last name:

Email:

Password:

- Your password can't be too similar to your other personal information.
- Your password must contain at least 8 characters.
- Your password can't be a commonly used password.
- Your password can't be entirely numeric.

Password confirmation: Enter the same password as before, for verification.

Gender: Male

Age:

Date of birth: mm/dd/yyyy

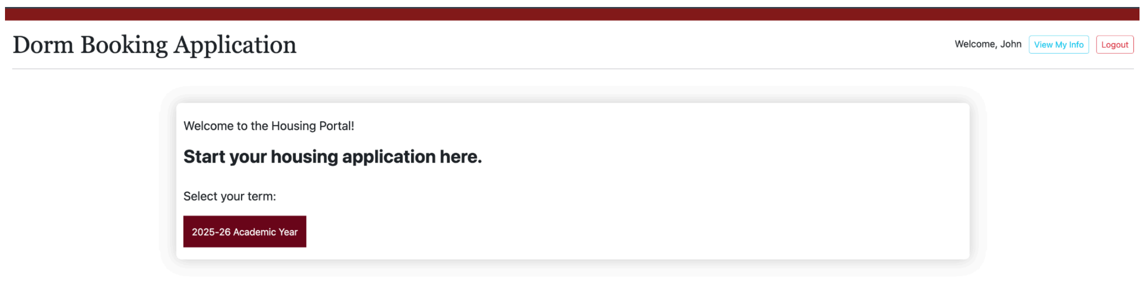
Phone numbers: Enter phone numbers separated by commas (e.g., 1234567890,0987654321)

Register

Already have an account?? Login

- Purpose: Allows students to login using their credentials to access the application
- Flow from UI to database: When the student enters in their login information, it uses django's built in authentication to check the User table and stores the user information in the session. Once authenticated, the user is redirected to the home page.
- User stories: Student Registration & Login, User Roles, User Permissions
- Schema: User, Student, Admin

Application Portal:



Dorm Booking Application

Welcome, John View My Info Logout

Welcome to the Housing Portal!

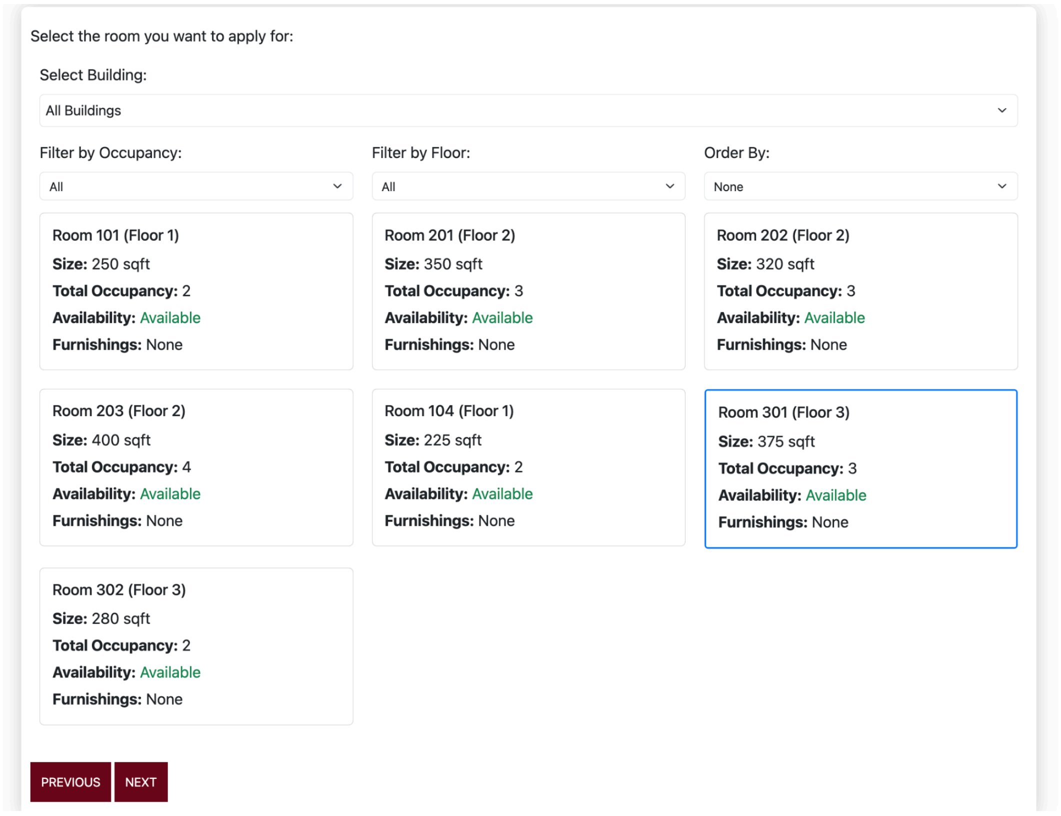
Start your housing application here.

Select your term:

2025-26 Academic Year

- Purpose: Allows students to start an application and select a dorm room.
- Flow from UI to database: Once the user is logged in, they can start an application. Here they will enter their information and select a room. Finally the application is created and saved to the database for the administrator to view.
- User stories: Room Booking, Room Database Management
- Schema: Student, Room, Building, Application

Room Browsing:



Select the room you want to apply for:

Select Building: All Buildings

Filter by Occupancy: All

Filter by Floor: All

Order By: None

Room 101 (Floor 1)
**Size:** 250 sqft
**Total Occupancy:** 2
**Availability:** Available
**Furnishings:** None

Room 201 (Floor 2)
**Size:** 350 sqft
**Total Occupancy:** 3
**Availability:** Available
**Furnishings:** None

Room 202 (Floor 2)
**Size:** 320 sqft
**Total Occupancy:** 3
**Availability:** Available
**Furnishings:** None

Room 203 (Floor 2)
**Size:** 400 sqft
**Total Occupancy:** 4
**Availability:** Available
**Furnishings:** None

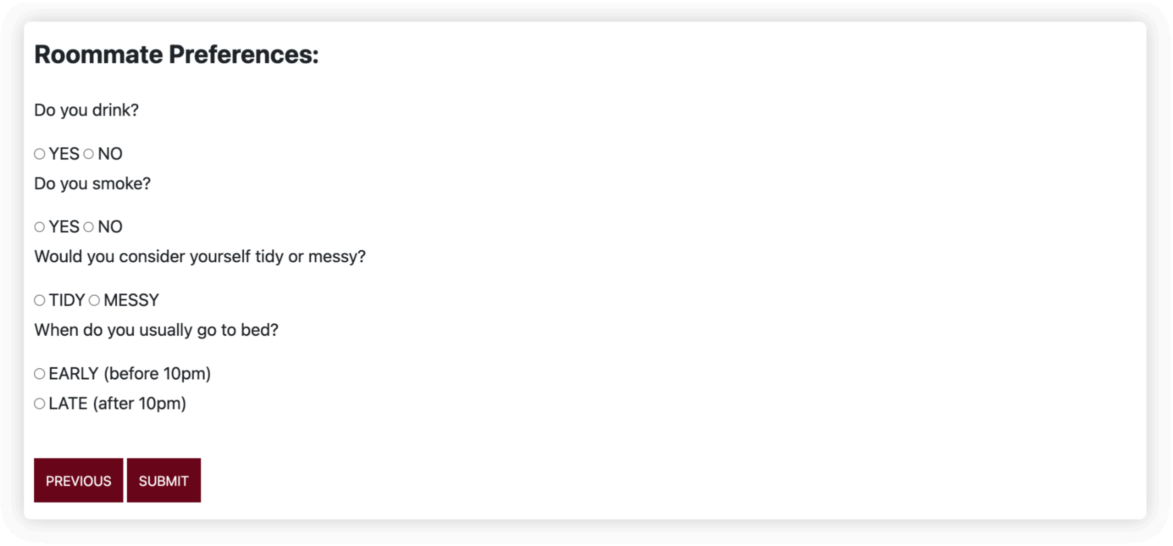
Room 104 (Floor 1)
**Size:** 225 sqft
**Total Occupancy:** 2
**Availability:** Available
**Furnishings:** None

Room 301 (Floor 3)
**Size:** 375 sqft
**Total Occupancy:** 3
**Availability:** Available
**Furnishings:** None

Room 302 (Floor 3)
**Size:** 280 sqft
**Total Occupancy:** 2
**Availability:** Available
**Furnishings:** None

- Purpose: This is one of the pages under the application. Allows the student to browse available rooms and select the one they would like to apply for. The student can also sort and filter the rooms according to its features.
- Flow from UI to database: All rooms in the database are displayed. Based on the criteria entered by the user, the database items are filtered and displayed dynamically on the page. Once the user selects the room, the roomID is stored in the session for future use.
- User stories: Room Browsing, Search/Filter functionality, Room Status Updates
- Schema: Room, Building

Roommate Preferences:



**Roommate Preferences:**

Do you drink?

○ YES ○ NO

Do you smoke?

○ YES ○ NO

Would you consider yourself tidy or messy?

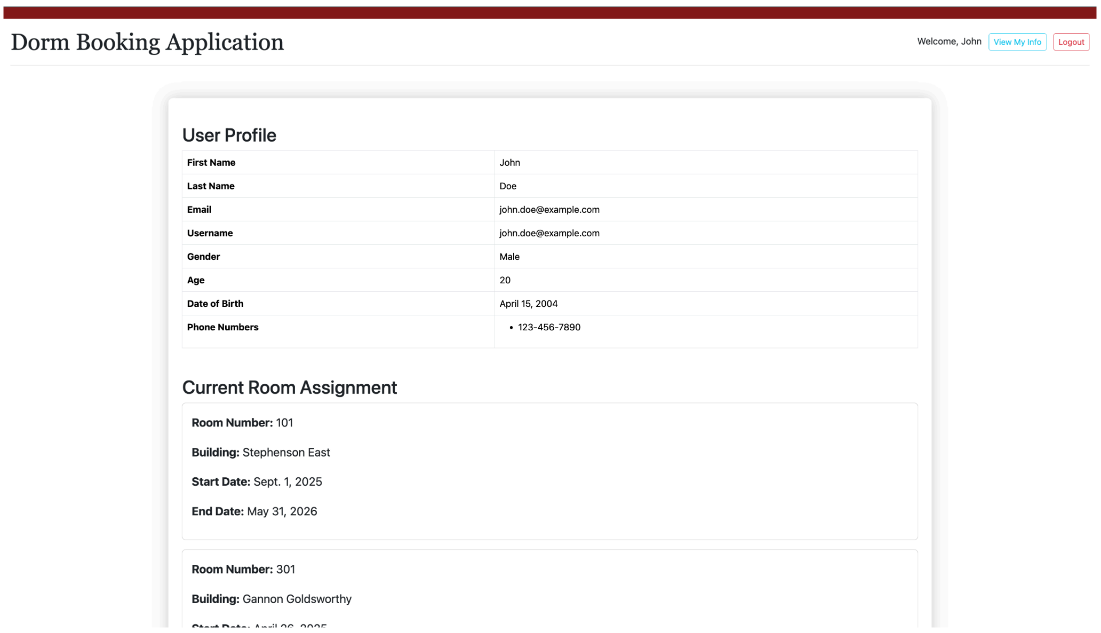
○ TIDY ○ MESSY

When do you usually go to bed?

○ EARLY (before 10pm)
○ LATE (after 10pm)

- Purpose: This is one of the pages under the application. Allows the student to enter their living habits, which will be taken into consideration by admin when assigning roommates.
- Flow from UI to database: When the student answers the questions, and their answers are stored in the Preferences table. Each student has their own instance of the preferences model.
- User stories: Preference-based Roommate Matching
- Schema: Student, Preferences

Room Assignment:



Dorm Booking Application

User Profile

| | |
|---|---|
| **First Name** | John |
| **Last Name** | Doe |
| **Email** | john.doe@example.com |
| **Username** | john.doe@example.com |
| **Gender** | Male |
| **Age** | 20 |
| **Date of Birth** | April 15, 2004 |
| **Phone Numbers** | • 123-456-7890 |

Current Room Assignment

**Room Number:** 101

**Building:** Stephenson East

**Start Date:** Sept. 1, 2025

**End Date:** May 31, 2026

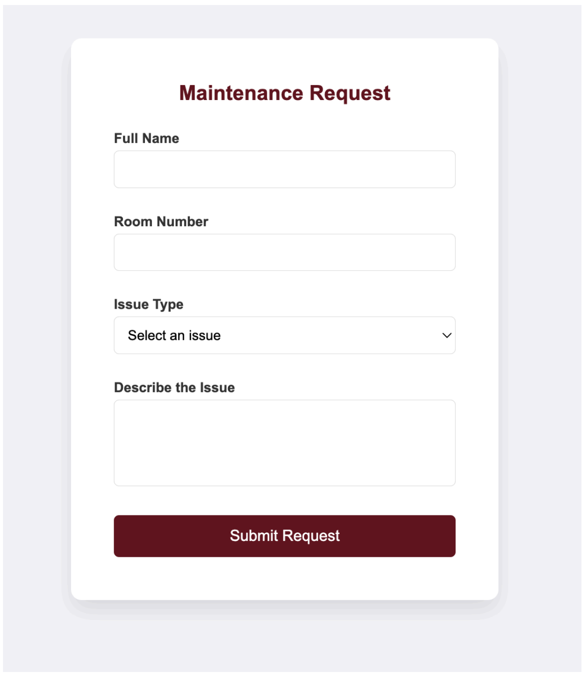
**Room Number:** 301

**Building:** Gannon Goldsworthy

**Start Date:** April 26, 2025

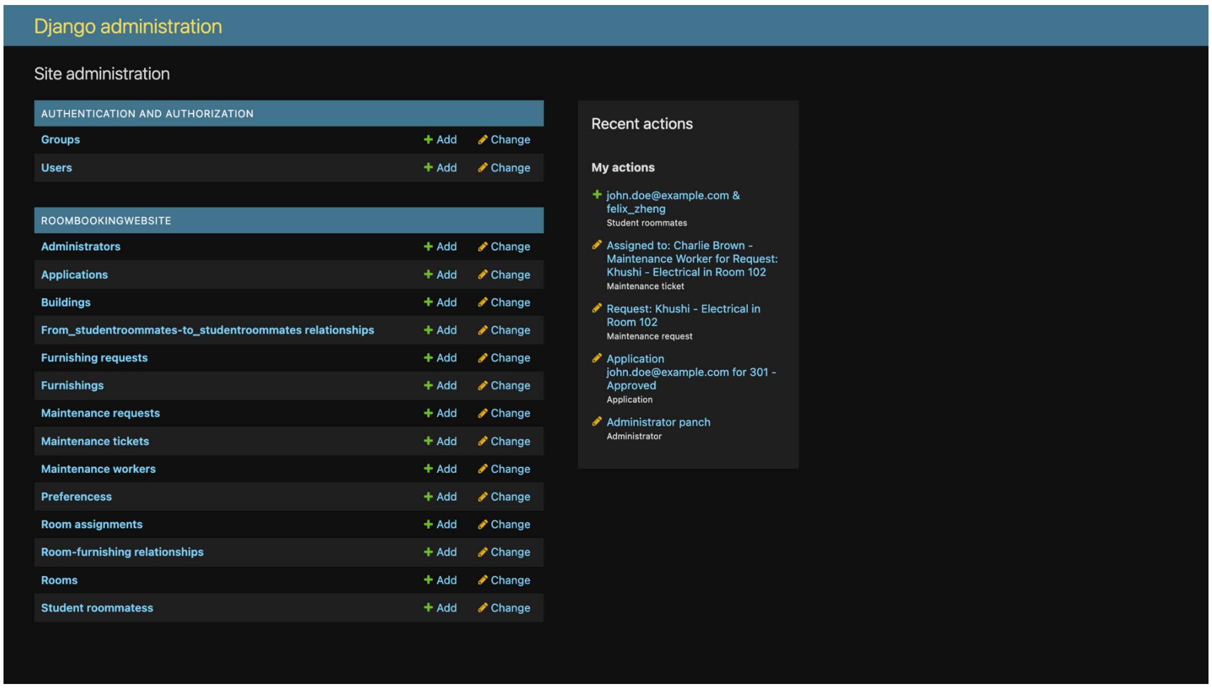
- Purpose: Allows the student to view their personal information as well as the current room assignment/application status. The students can also submit maintenance requests and furnishing requests from this page.
- Flow from UI to database: All the information displayed is from the database.
- User stories: This page is not connected to a user story, but it provides relevant information to the user and allows them to be able to navigate to the maintenance request page.
- Schema: Student, Room, Building, Application

Maintenance Request:



- Purpose: Allows the student to submit a maintenance request if they find any issues in the Room.
- Flow from UI to database: The user enters in the relevant information. Once they submit the request, it is added to the database to be viewed by the administrator for further action.
- User stories: Maintenance Requests
- Schema: Maintenance Request

Administrator View:



Django administration

Site administration

| AUTHENTICATION AND AUTHORIZATION | | |
|---|---|---|
| Groups | + Add | Change |
| Users | + Add | Change |

| ROOMBOOKINGWEBSITE | | |
|---|---|---|
| Administrators | + Add | Change |
| Applications | + Add | Change |
| Buildings | + Add | Change |
| From_studentroommates-to_studentroommates relationships | + Add | Change |
| Furnishing requests | + Add | Change |
| Furnishings | + Add | Change |
| Maintenance requests | + Add | Change |
| Maintenance tickets | + Add | Change |
| Maintenance workers | + Add | Change |
| Preferencess | + Add | Change |
| Room assignments | + Add | Change |
| Room-furnishing relationships | + Add | Change |
| Rooms | + Add | Change |
| Student roommatess | + Add | Change |

Recent actions

My actions

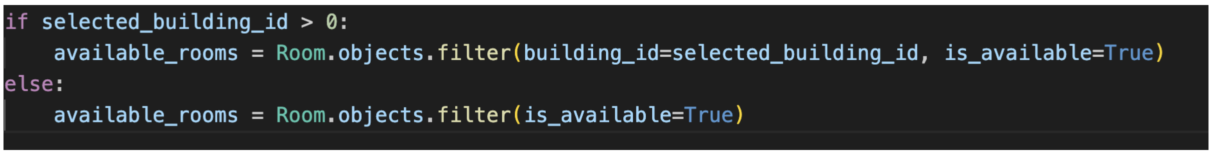
- john.doe@example.com & felix_zheng
  Student roommates
- Assigned to: Charlie Brown - Maintenance Worker for Request: Khushi - Electrical in Room 102
  Maintenance ticket
- Request: Khushi - Electrical in Room 102
  Maintenance request
- Application john.doe@example.com for 301 - Approved
  Application
- Administrator panch
  Administrator

- Purpose: Allows the administrator to navigate to and perform all relevant actions.
- Flow from UI to database: The administrator has access to all information in the database and they are able to perform any actions relating to their responsibilities. All data is displayed from the database, and any changes made by the administrator are reflected in the database.
- User stories: Admin Room Booking Management, Maintenance Request Management, Room Status Updates
- Schema: Room, Building, Student, Roommates, Furnishing, Application

**Query Design**

- SQL query to find available rooms,

applicationPortal > views.py > pg2



```
if selected_building_id > 0:
    available_rooms = Room.objects.filter(building_id=selected_building_id, is_available=True)
else:
    available_rooms = Room.objects.filter(is_available=True)
```

*SELECT * FROM Room*

*WHERE building_id = selected_building_id*

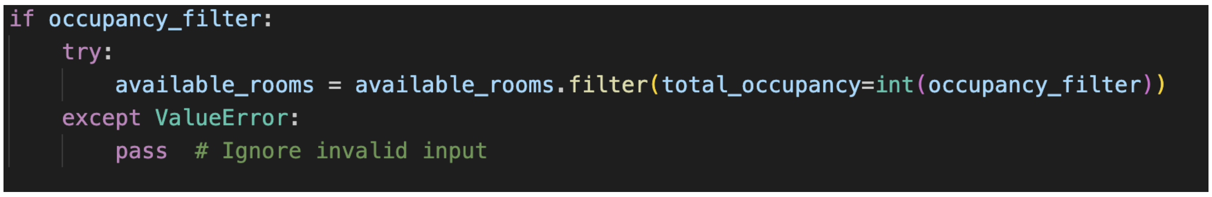
*AND is_available = TRUE;*

Purpose: To display all available rooms to the student so that they can choose among those options.

Expected Results: Shows all rooms where the status is available.

- SQL query to find available rooms with a certain occupancy,

applicationPortal > views.py > pg2



```
if occupancy_filter:
    try:
        available_rooms = available_rooms.filter(total_occupancy=int(occupancy_filter))
    except ValueError:
        pass  # Ignore invalid input
```

*SELECT * FROM room*

*WHERE building_id = <selected_building_id>*

*AND is_available = TRUE*

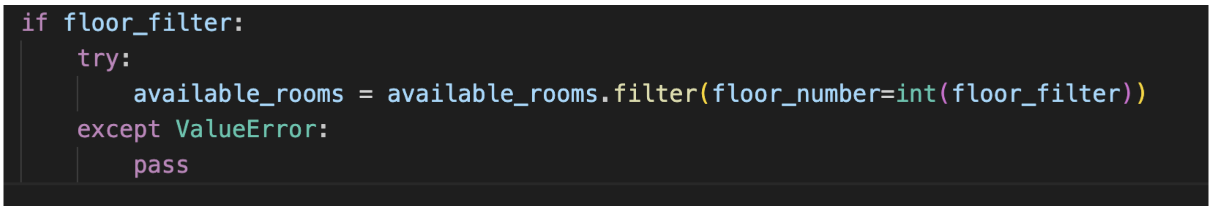
*AND total_occupancy = <selected_occupancy>;*

Purpose: To display all available rooms to the student according to the occupancy they want to search for.

Expected Results: Shows all rooms where the status is available and the occupancy is equal to the number selected by the user.

- SQL query to find available rooms on a certain floor,

applicationPortal > views.py > pg2



```
if floor_filter:
    try:
        available_rooms = available_rooms.filter(floor_number=int(floor_filter))
    except ValueError:
        pass
```

*SELECT * FROM room*

*WHERE building_id = <selected_building_id>*

*AND is_available = TRUE*

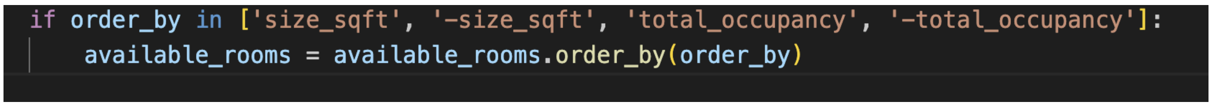
*AND floor_number = <selected_floor>;*

Purpose: To display all available rooms to the student according to the floor number they want to search for.

Expected Results: Shows all rooms where the status is available and the floor is equal to the number selected by the user.

- SQL query to order the available rooms based on size,

applicationPortal > views.py > pg2



```
if order_by in ['size_sqft', '-size_sqft', 'total_occupancy', '-total_occupancy']:
    available_rooms = available_rooms.order_by(order_by)
```

*SELECT * FROM room*

*WHERE building_id = <selected_building_id>*

*AND is_available = TRUE*

*ORDER BY size_sqft ASC;*

*SELECT * FROM room*

*WHERE building_id = <selected_building_id>*

*AND is_available = TRUE*

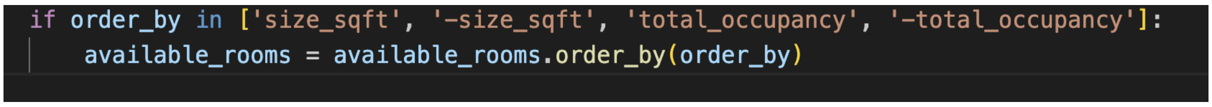
*ORDER BY size_sqft DESC;*

Purpose: To display all available rooms to the student in ascending or descending order according to the size of the room in square feet.

Expected Results: Shows all rooms where the status is available in the order that the user wishes, based on room size.

- SQL query to order the available rooms based on occupancy,

applicationPortal > views.py > pg2



```python
if order_by in ['size_sqft', '-size_sqft', 'total_occupancy', '-total_occupancy']:
    available_rooms = available_rooms.order_by(order_by)
```

*AND is_available = TRUE*

*ORDER BY total_occupancy ASC;*

*SELECT * FROM room*

*WHERE building_id = <selected_building_id>*

*AND is_available = TRUE*

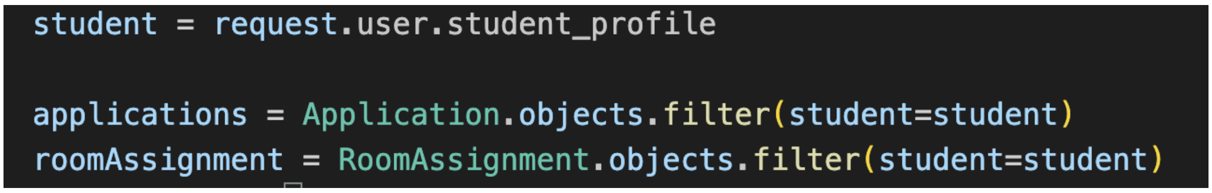
*ORDER BY total_occupancy DESC;*

Purpose: To display all available rooms to the student in ascending or descending order according to the total occupancy.

Expected Results: Shows all rooms where the status is available in the order that the user wishes, based on occupancy.

- SQL query to find room number from the room assignment,

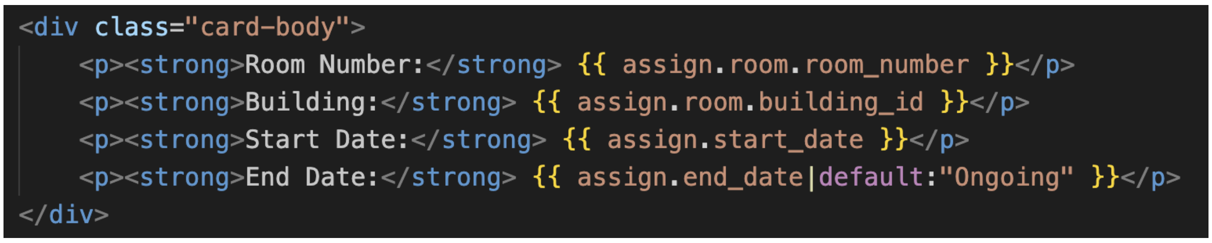
applicationPortal > views.py > userinfo



```python
student = request.user.student_profile

applications = Application.objects.filter(student=student)
roomAssignment = RoomAssignment.objects.filter(student=student)
```

applicationPortal > templates > viewuserinfo.html



```
<div class="card-body">
    <p><strong>Room Number:</strong> {{ assign.room.room_number }}</p>
    <p><strong>Building:</strong> {{ assign.room.building_id }}</p>
    <p><strong>Start Date:</strong> {{ assign.start_date }}</p>
    <p><strong>End Date:</strong> {{ assign.end_date|default:"Ongoing" }}</p>
</div>
```

*SELECT r.room_number*

*FROM room_assignment ra*

*JOIN room r ON ra.room_id = r.room_id*

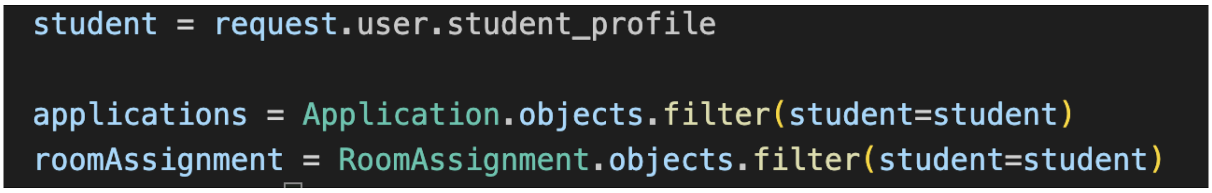
*WHERE ra.student_id = <student_id>;*

Purpose: To display room number which is assigned to the student, using the foreign key room_id.

Expected Results: Show the information corresponding to the roomID which the student has been assigned to from the Room table.

- SQL query to find room number from the room assignment,

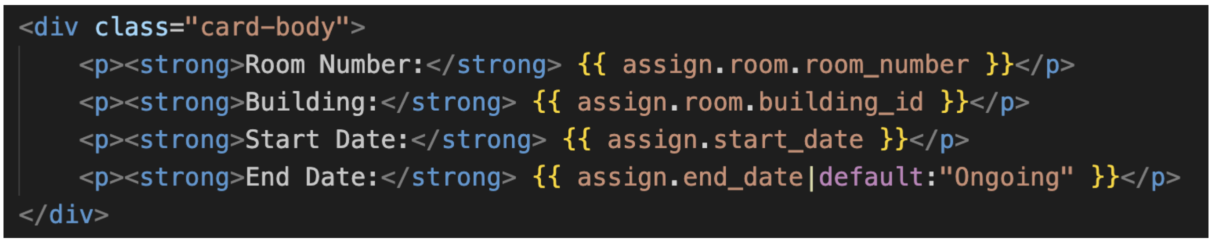
applicationPortal > views.py > userinfo



```
student = request.user.student_profile

applications = Application.objects.filter(student=student)
roomAssignment = RoomAssignment.objects.filter(student=student)
```

applicationPortal > templates > viewuserinfo.html



```
<div class="card-body">
    <p><strong>Room Number:</strong> {{ assign.room.room_number }}</p>
    <p><strong>Building:</strong> {{ assign.room.building_id }}</p>
    <p><strong>Start Date:</strong> {{ assign.start_date }}</p>
    <p><strong>End Date:</strong> {{ assign.end_date|default:"Ongoing" }}</p>
</div>
```

*SELECT r.building_id*

*FROM room_assignment ra*

*JOIN room r ON ra.room_id = r.room_id*

*WHERE ra.student_id = <student_id>;*

Purpose: To display the building number of the room which is assigned to the student, using the foreign key room_id.

Expected Results: Show the information corresponding to the roomID which the student has been assigned to from the Room table.

## Limitations & Future Scope

Since this was a student project designed and developed in the span of a few months, we were limited in time and the ability to implement more advanced features, such as in-app messaging which was one of our original ideas. Future improvements to this application may include: booking history, and a view for maintenance workers.

## Conclusion

Through this project, our team gained valuable technical experience as well collaborative skills. We realized the importance of database design when building an application, and gained hands-on experience on how to build a system on the database,while incorporating user's needs. Now at the end, we feel a sense of accomplishment with what we were able to achieve, along with a strong understanding of DBMS concepts.

## Resources

Github Link: https://github.com/Jacawb/CPTS_451-Project