

Database System Project Report

Classroom Booking System



Student: Xingyu Zhao (赵兴宇)

Campus ID: 22307130276

Email: 22307130276@m.fudan.edu.cn

Advisors: Prof. Shuigeng Zhou

Mr. Yi Guan, Mr. Haoran Zhou & Mr. Wenbo Liu

School of Computer Science, Fudan University

June, 2024

0. Preface

This is a report, including detailed description of the entire system design, implementation and frontend illustration, of the campus classroom booking system designed by the author as his course project for the 'Introduction to Database System' course in Spring 2024.

The project was mainly completed by the author under the guidance of the official Django tutorials. (Source: <https://docs.djangoproject.com/en/5.0/>). The author also received help from OpenAI's ChatGPT, especially during the frontend developing process, but in no way can be described as 'copying' or 'plagiarizing' without independent thinking. Besides, during the frontend development, the author had a lot of discussion with Miss Qilin Hu (the author's classmate in this class) and was grateful for her generous advice. As a final part of this project, this report was finished by the author independently, without any AI-generated writing.

It must be pointed out that, from the beginning to end, the database system design theory I learnt from Professor Shuigeng Zhou's class is of great help to my classroom booking system development. Moreover, I would like to sincerely thank all the TAs' help, including but not limited to for the tools (both the Django framework and the frontend tools) they introduced to me, and their overall guidance about the project design and the report writing.

1. Design Purpose and Background

Nowadays, more and more facilities at universities, have been accessible to faculty, students and staff. To better organize and manage these resources, many universities have adopted a variety of booking systems. Exactly out of this purpose, I designed this ‘classroom-booking’ system, which supports displaying future available time slots for reservation, their according available classrooms (as well as their locations, capacities and equipment).

The system allows users to make booking if the user has the permissions and the selected classroom is available at that particular time slot. Besides, I have also designed a user homepage to facilitate them to manage their personal information and lead to a page showing the bookings that they already made, where they can cancel them any time. (More additional and detailed features will be elaborated in later parts of this report.)

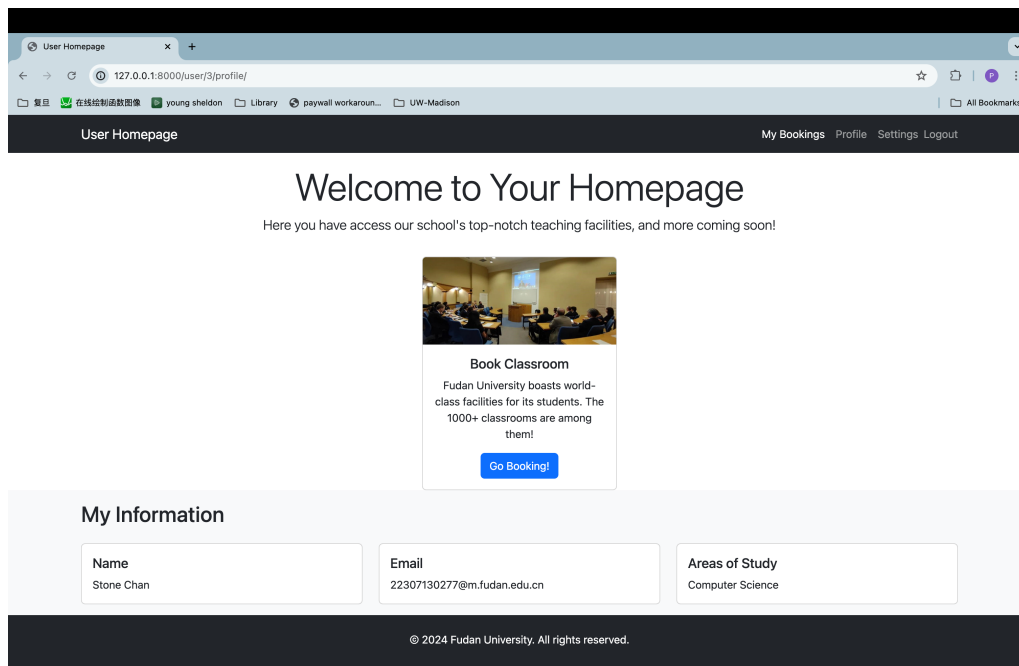


Figure 1. Overview of a user's homepage in my ‘classroom-booking system’

2. Database Design

2.1 Conceptual Structure Design

In my database, there are six major models, namely ‘Classroom’, ‘Time’, ‘Classroom info by time slot’, ‘Reservation’, ‘Student’, ‘User’. It’s worth noting that the relationship sets in conventional database design using E-R models has been replaced with the same ‘models’ as the entity set traditionally maps to, due to the considerations of unified abstraction and more flexibility in Django features. Now let’s have a closer look of the E-R diagram associated with my ‘classroom-booking’ system.

(Note: The E-R diagram displayed below was automatically generated by ‘django-extensions’, a third-part package of Django, which draws on the ‘model.py’ file to generate the according E-R diagram. And this process was further simplified by Shaowen Chen’s code. Source:

<https://www.chenshaowen.com/blog/django-model-and-er-diagram.html>)

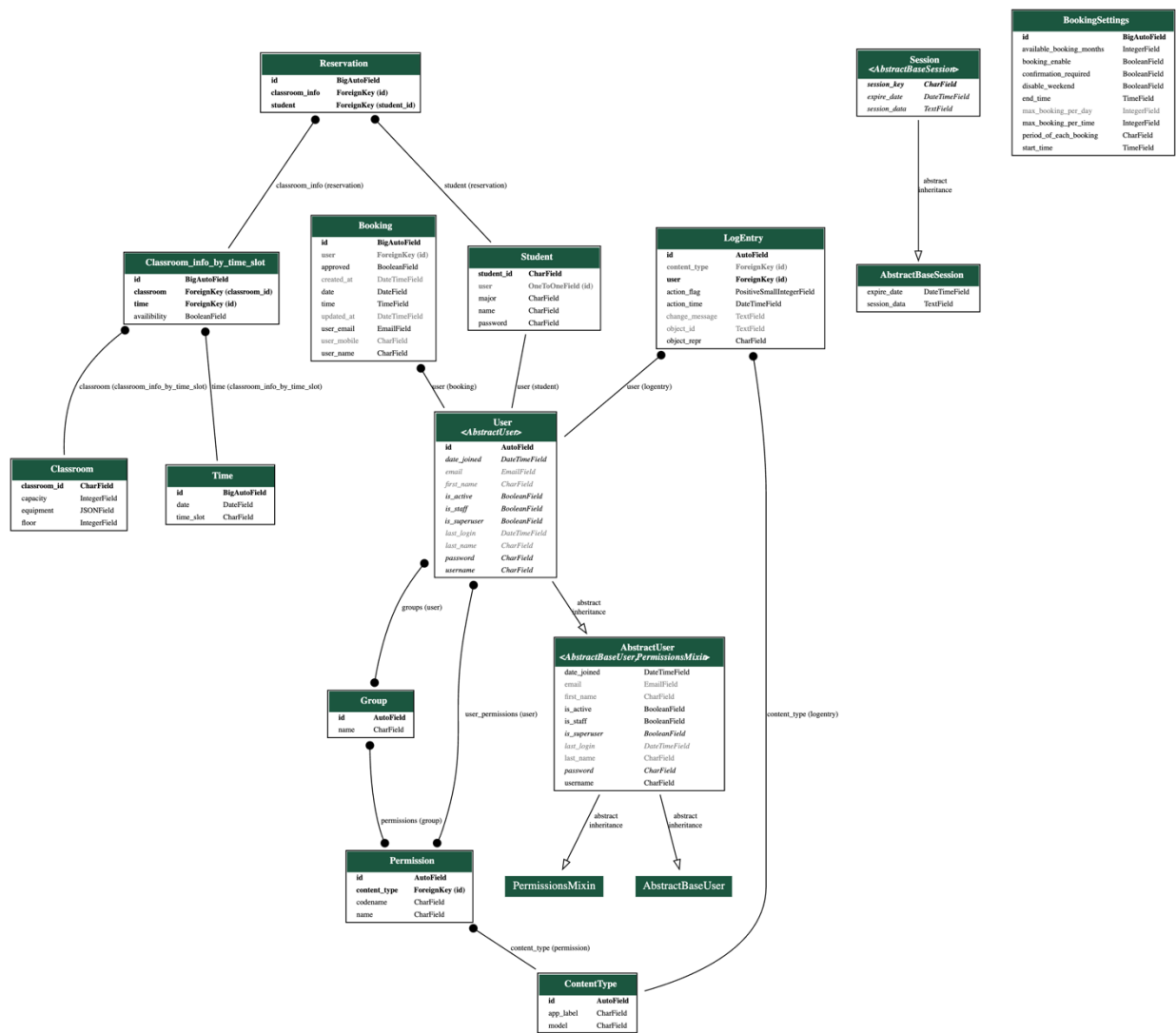


Figure 2. The E-R diagram associated with my ‘classroom-booking’ system

2.2 Logical Structure Design

Model: Classroom

Field Name	Field Type	Remark	Is Null
classroom_id	CharField(max_length=20, primary_key=True)	Classroom ID (Primary Key)	No
floor	IntegerField(default=0)	Floor number	No
capacity	IntegerField(default=0)	Classroom capacity	No
equipment	JSONField()	Classroom equipment details	No

Table 1. A detailed description of the 'Classroom' model

'__str__' Method:

This method returns the `classroom_id` as a string representation of the Classroom instance. This approach makes it convenient to identify different objects of Classroom in the admin site. All the following models have this method due to the same reason.

Model: Time

Field Name	Field Type	Remark	Is Null
date	DateField()	Date of the time slot	No
time_slot	CharField(max_length=50)	Time slot description	No

Table 2. A detailed description of the 'Time' model

'__str__' Method:

This method returns a string combining the `date` and `time_slot` fields, formatted as "YYYY-MM-DD - Time Slot".

Model: Classroom_info_by_time_slot

Field Name	Field Type	Remark	Is Null
classroom	ForeignKey(Classroom, on_delete=models.CASCADE)	Linked classroom	No
time	ForeignKey(Time, on_delete=models.CASCADE)	Linked time slot	No
availability	BooleanField(default=True)	Availability status	No

Table 3. A detailed description of the 'Classroom_info_by_time_slot' model

Meta Options:

- `unique_together`: ('classroom', 'time')

Classroom-Booking System Project Report

This ensures that in the model ‘Time’ there will be only one unique tuple for each combination for date and time_slot values.

‘__str__’ Method:

Directly display whether a classroom is available at a particular time

Model: Student

Field Name	Field Type	Remark	Is Null
student_id	CharField(max_length=20, primary_key=True)	Student ID (Primary Key)	No
name	CharField(max_length=50)	Student's name	No
user	OneToOneField(User, on_delete=models.CASCADE, null=True, blank=True)	Linked user account	Yes
password	CharField(max_length=128, default=12345678)	Password	No
major	CharField(max_length=100, null=True)	Major	Yes

Table 4. A detailed description of the ‘Student model

Customized Save Method:

```
def save(self, *args, **kwargs):
    if not self.user:
        username = self.student_id
        email = f"{username}@m.fudan.edu.cn"
        self.user = User.objects.create_user(username=username, password=self.password, email=email)
        self.user.first_name = self.name.split(' ')[0]
        self.user.last_name = ' '.join(self.name.split(' ')[1:])
        self.user.save()
    else:
        self.user.set_password(self.password)
        self.user.save()
    super().save(*args, **kwargs)
```

This customized save method ensures that every ‘Student’ instance has a corresponding ‘User’ object in Django’s authentication system. It handles both the creation of new ‘User’ objects and the updating of existing ones.

Model: Reservation

Field Name	Field Type	Remark	Is Null
student	ForeignKey(Student, on_delete=models.CASCADE)	Linked student	No
classroom_info	ForeignKey(Classroom_info_by_time_slot, on_delete=models.CASCADE, null=True)	Linked classroom and time slot	Yes

Table 5. A detailed description of the ‘Reservation model

Customized Save Method:

It sets the 'availability' of the related 'Classroom_info_by_time_slot' to 'False' before saving the reservation, which is exactly what we mean by saying 'one cannot make a reservation for a classroom at a particular time slot if someone else has already booked it'.

2.3 Physical Structure Design

Database Selection

For this classroom booking system, I used the 'SQLite', which is a relational database that supports complex queries, indexing, JSON fields. It is suitable for this application.

Indexes associated with each model

1. Classroom Model:

- Primary Key Index on classroom_id.

2. Time Model:

- Composite Unique Index on (date, time_slot) to ensure no duplicate time slots for a single day.

3. Classroom_info_by_time_slot Model:

- Composite Unique Index on (classroom, time) to ensure no duplicate entries for the same classroom and time slot.
- Foreign Key Indexes on classroom and time for efficient lookups.

4. Student Model:

- Primary Key Index on student_id.
- Foreign Key Index on user for efficient lookups.

5. Reservation Model:

- Foreign Key Indexes on student and classroom_info for efficient lookups.

3. System Design and Implementation

3.1 Unique Constraint

Since Django does not support composite primary keys, I used unique constraints to achieve our desired effect in the Classroom_info_by_time_slot model. Specifically, this is implemented using the unique_together option in the model's Meta class:

```
class Classroom_info_by_time_slot(models.Model):
    classroom = models.ForeignKey(Classroom, on_delete=models.CASCADE)
    time = models.ForeignKey(Time, on_delete=models.CASCADE)
    availability = models.BooleanField(default=True)

    class Meta:
        unique_together = ('classroom', 'time')
```

The unique_together option ensures that each combination of classroom and time is unique, which effectively creates a composite unique constraint, ensuring that no two records can have the same combination of classroom and time. This approach is used to emulate the functionality of composite primary keys, which are not directly supported in Django.

3.2 Use of Signals

To implement more complex integrity constraints, my classroom booking system make use of 'signals' in Django to achieve the effect that 'whenever one tuple is inserted in a model, another model will be automatically updated'.

Here are the details of how this is implemented.

1. Updating Classroom_info_by_time_slot when a Classroom is created:

```
@receiver(post_save, sender=Classroom)
def add_classroom_in_Classroom_info_by_time_slot(sender, instance, created, **kwargs):
    if created:
        times = Time.objects.all()
        for time in times:
            Classroom_info_by_time_slot.objects.create(classroom=instance, time=time, availability=True)
```

This signal listens for the post_save event on the Classroom model. When a new classroom is created, it automatically creates entries in the Classroom_info_by_time_slot table for all existing time slots.

2. Updating Classroom_info_by_time_slot when a Time is created:


```
@receiver(post_save, sender=Time)
def add_time_in_Classroom_info_by_time_slot(sender, instance, created, **kwargs):
    if created:
        classrooms = Classroom.objects.all()
        for classroom in classrooms:
            Classroom_info_by_time_slot.objects.create(classroom=classroom, time=instance, availability=True)
```

Similarly, this signal listens for the `post_save` event on the `Time` model. When a new time slot is created, it automatically creates entries in the `Classroom_info_by_time_slot` table for all existing classrooms.

These signals ensure that the `Classroom_info_by_time_slot` table is kept up-to-date whenever new classrooms or time slots are added, maintaining the integrity and consistency of the data across related tables. This reflects the concept of using signals in Django to automatically update related content in other tables when a new record is inserted.

3.3 Model-level Views

Django doesn't directly support SQL views, but we can simulate them using model managers. In my classroom booking system, I created two custom managers to encapsulate frequent queries.

1. Classroom Availability View

I added a custom manager to the '`Classroom_info_by_time_slot`' model to get all available classrooms for a specific time slot.

```
class ClassroomManager(models.Manager):
    def available_classrooms(self, time_slot):
        return self.filter(time=time_slot, availability=True)

class Classroom_info_by_time_slot(models.Model):
    classroom = models.ForeignKey(Classroom, on_delete=models.CASCADE)
    time = models.ForeignKey(Time, on_delete=models.CASCADE)
    availability = models.BooleanField(default=True)

    objects = ClassroomManager()

    class Meta:
        unique_together = ('classroom', 'time')

    def __str__(self):
        return f"{self.classroom} - {self.time} - {'Available' if self.availability else 'Unavailable'}
```

2. Student Reservations View

I added a custom manager to the '`Reservation`' model to get all reservations made by a specific student.

```
class ReservationManager(models.Manager):
    def student_reservations(self, student):
        return self.filter(student=student)

class Reservation(models.Model):
    student = models.ForeignKey(Student, on_delete=models.CASCADE)
    classroom_info = models.ForeignKey(Classroom_info_by_time_slot, on_delete=models.CASCADE, null=True)

    objects = ReservationManager()

    def save(self, *args, **kwargs):
        self.classroom_info.availability = False
        self.classroom_info.save()
        super().save(*args, **kwargs)

    def __str__(self):
        if self.classroom_info:
            return f"{self.student.name} reserved {self.classroom_info.classroom} at {self.classroom_info.time}"
        else:
            return f"{self.student.name} reservation information unavailable"
```

With two model managers added to our Classroom_info_by_time_slot and Reservation models, we can handle frequent queries like ‘the available classrooms at a particular time slot’ and ‘a particular student’s reservation’ efficiently and support reusing.

3.4 Nested Queries

My classroom booking system supports a filtering mechanism to find classrooms that meet users’ demand. Specifically, the system makes use of nested queries to filter classrooms based on their availability and equipment. The attributes belong to the Classroom_info_by_time_slot model and Classroom model respectively. Thus, a nested query would support our desired functions.

[Home](#) [My Reservations](#) [View Time Slot](#) [Dropdown](#) [Disabled](#) [Logout](#)

Available Classrooms

You are looking at time 2024-07-01 - 10:00-12:00

Enter required equipment (comma-separated):

Search

#	Classroom	Floor	Capacity	Equipment	Actions
1	H2104B	1	38	['Projector', 'Blackboard']	<button>Make Reservation</button>
2	H2106A	1	30	['Blackboard']	<button>Make Reservation</button>
3	H3109	1	120	['Blackboard', 'Multimedia']	<button>Make Reservation</button>

Figure 3. The classroom filtering functionality, supported by nested queries

The following is the part of code which draws on nested query to implement this function.

```
@staticmethod
def find_available_classrooms_with_equipment(time_id, required_equipment):
    available_classrooms = Classroom_info_by_time_slot.objects.filter([
        time_id=time_id,
        availability=True,
        classroom__equipment__contains=required_equipment
    ]).values_list('classroom_id', flat=True)

    classrooms_with_required_equipment = Classroom.objects.filter(
        classroom_id__in=available_classrooms
    ).distinct()

    return classrooms_with_required_equipment
```

3.5 Role-Based Access Control

My classroom booking system supports three different roles--‘administrator’, ‘faculty’, and ‘students’. They have different permissions and access levels within the system. Specifically, the ‘administrator’ has the highest permissions including organizing (inserting, deleting, modifying) all resources and users. The ‘students’ are only allowed to book classrooms and manage their own personal information (like password), while the ‘faculty’ can book faculty-only meeting rooms.

Administrator can set a user’s role/type at the admin site as follows. (The example is setting the type of User ‘17002’ to ‘faculty’.)

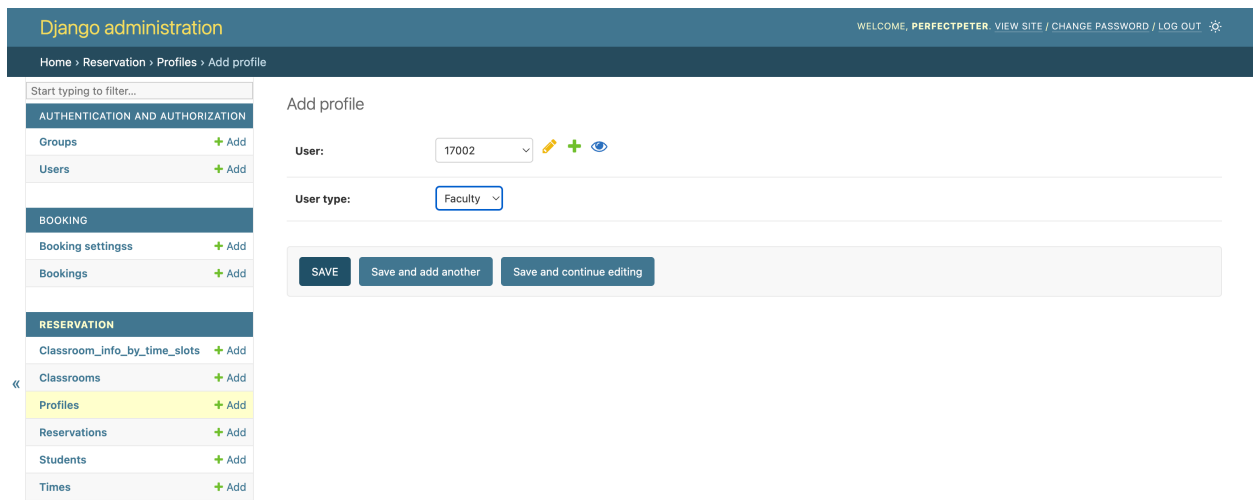
The screenshot shows the Django administration interface. The top navigation bar includes the 'Django administration' logo and a welcome message for 'PERFECTPETER'. The breadcrumb trail indicates the path: Home > Reservation > Profiles > Add profile. On the left, a sidebar menu lists various system components under three main categories: AUTHENTICATION AND AUTHORIZATION (Groups, Users), BOOKING (Booking settings, Bookings), and RESERVATION (Classroom_info_by_time_slots, Classrooms, Profiles, Reservations, Students, Times). The 'Profiles' item is currently selected and highlighted. The main content area is titled 'Add profile' and contains a form with two fields: 'User:' with a dropdown menu showing '17002' and 'User type:' with a dropdown menu showing 'Faculty'. Below the form are three buttons: 'SAVE', 'Save and add another', and 'Save and continue editing'.

Figure 4. An admin interface that supports role-based access control

The following two figures show the difference of the access levels between ‘faculty’ users and ‘students’ users. They will get different information while they are viewing the available rooms at the same time slot.

Classroom-Booking System Project Report

[Home](#) [My Reservations](#) [View Time Slot](#) [Dropdown](#) [Disabled](#) [Logout](#)

Available Classrooms

You are looking at time 2024-07-02 - 10:00-12:00

Enter required equipment (comma-separated):

Search

#	Classroom	Floor	Capacity	Equipment	Actions
1	H2104B	1	38	Projector, Blackboard	Make Reservation
2	H2106A	1	30	Blackboard	Make Reservation
3	H3108	1	120	Book, Table, Chair	Make Reservation
4	H3109	1	120	Blackboard, Multimedia	Make Reservation
5	HGD201	2	40	Whiteboard, Projector, Multimedia	Make Reservation
6	HGD202	2	40	Blackboard, Table, Microphone	Make Reservation
7	HGD301	3	50	Screen, Computer, Whiteboard	Make Reservation

Figure 5. The search result that a ‘faculty’ user would see
(In comparison with Figure 6.)

[Home](#) [My Reservations](#) [View Time Slot](#) [Dropdown](#) [Disabled](#) [Logout](#)

Available Classrooms

You are looking at time 2024-07-02 - 10:00-12:00

Enter required equipment (comma-separated):

Search

#	Classroom	Floor	Capacity	Equipment	Actions
1	H2104B	1	38	Projector, Blackboard	Make Reservation
2	H2106A	1	30	Blackboard	Make Reservation
3	H3108	1	120	Book, Table, Chair	Make Reservation
4	H3109	1	120	Blackboard, Multimedia	Make Reservation

Figure 6. The search result that a ‘Student’ user would see
(In comparison with Figure 5.)

4. Frontend Development Process

4.1 Overview

Tools and Technologies Used

- **Bootstrap:** We utilized Bootstrap for creating a responsive and visually appealing layout. It simplifies the development of complex web pages with its comprehensive grid system and pre-designed components. My frontend development process was greatly simplified thanks to it. (Source: <https://getbootstrap.com/>)
- **Font Awesome:** Font Awesome was used for integrating icons into the user interface, enhancing visual clarity and user experience.
- **jQuery:** jQuery was used to enhance the interactivity of the page, particularly for handling dynamic content and user interactions.
- **Django Admin:** As Django's built-in admin interface, it provides ample functionalities for administrative tasks, such as managing users, classrooms, and reservations.

4.2 User Homepage

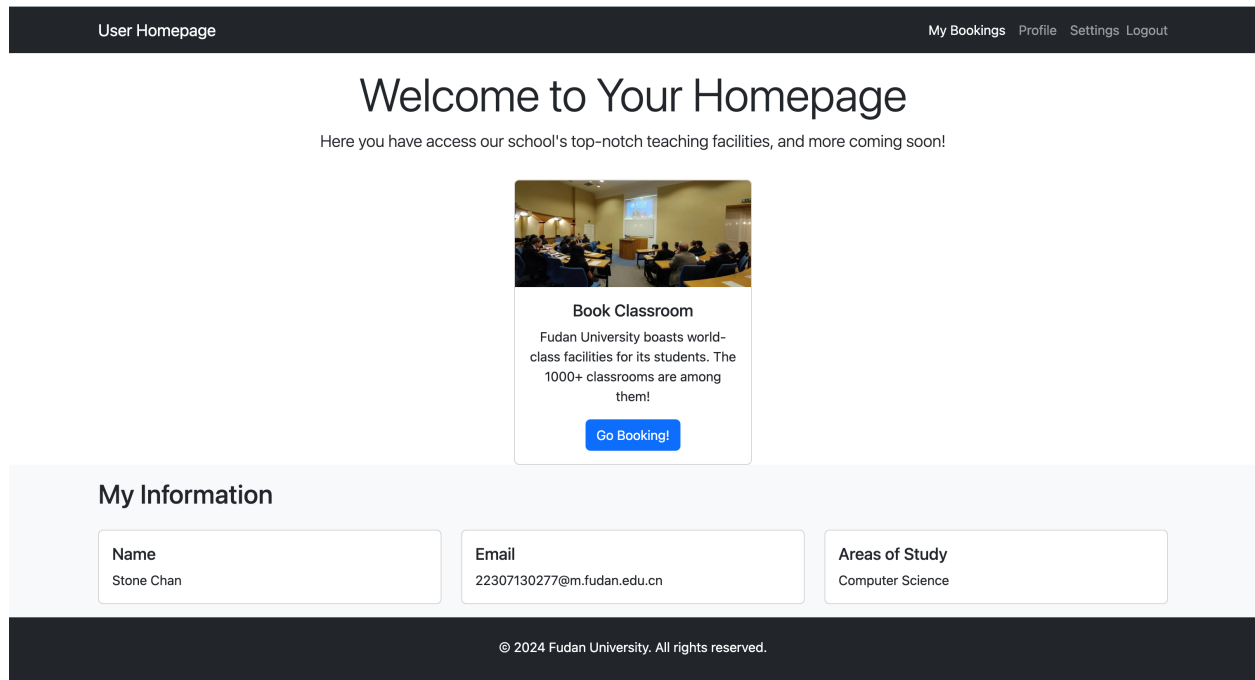


Figure 7. A user homepage

Navigation Bar

Tools:

Bootstrap for layout and responsiveness.

Features:

It has links to different pages. ‘My Bookings’ will lead to a page displaying all the bookings made by the user (We will explain its functionality in detail later). ‘Profile’ will lead to a page that the user can edit their personal information, such as email and password. ‘Settings’ has no functions so far ☺. ‘Logout’, as its name suggests, will log the current user out and automatically redirects to the log in page like follows.

**Welcome to Classroom Booking
System!**

Username

Password

Login

Figure 8. The logout interface

Welcome Section

Features:

The section includes a welcome message and a brief introduction to the booking system’s capabilities, emphasizing the state-of-the-art facilities available at Fudan University.

Booking Card

Features:

It displays a fancy classroom image at the School of Management at Fudan University ☺. Below, it includes a brief description of FDU’s classrooms and a call-to-action button linking to the booking page (which will be explained in detail later).

User Information Section

Features:

This section displays key user information in a card layout, including the user’s name, email and areas of study.

Footer

Features:

The footer includes a copyright notice ‘© 2024 Fudan University. All rights reserved.’ (You know, just to make it look more formal☺).

4.3 Time Slot Overview Page

Navbar Home My Bookings Dropdown Disabled Logout		
Available Time Slots		
Day	Time Slot	Actions
July 1, 2024	08:00-10:00	View Details
July 1, 2024	10:00-12:00	View Details
July 1, 2024	14:00-16:00	View Details
July 1, 2024	16:00-18:00	View Details
July 2, 2024	08:00-10:00	View Details
July 2, 2024	10:00-12:00	View Details

Figure 9. The time slot listing page

Navigation Bar

Features:

The navigation bar includes links to the user's homepage (as described before), their own bookings, as well as a logout function. (The dropdown menu is just for aesthetics purpose. It has no real functions ☺).

Main Content Area

Features:

It displays the date and specific time slot offered for booking. It also has an action column which contains buttons that link to detailed views for each time slot, where they can actually view each available classroom and make a booking.

4.4 Time Slot Detail Page

[Home](#) [My Reservations](#) [View Time Slot](#) [Dropdown](#) [Disabled](#) [Logout](#)

Available Classrooms

You are looking at time 2024-07-02 - 10:00-12:00

Enter required equipment (comma-separated):

Search

#	Classroom	Floor	Capacity	Equipment	Actions
1	H2104B	1	38	Projector, Blackboard	<div>Make Reservation</div>
2	H2106A	1	30	Blackboard	<div>Make Reservation</div>
3	H3108	1	120	Book, Table, Chair	<div>Make Reservation</div>
4	H3109	1	120	Blackboard, Multimedia	<div>Make Reservation</div>

Figure 10. A time slot detail page, displaying all the available classrooms then.
(The results would differ depending on the user type, e.g. a faculty user or a student user.)

Navigation Bar

The top navigation bar provides easy access to the user's homepage (through 'Home'), the user's reservations list (through 'My Reservations'), the time slot list offered for booking (through 'View Time Slot'), as well as a logout functionality. (As before, the dropdown menu is just for aesthetics 😊)

Main Content Section

First, it includes a page header, displaying a title 'Available Classrooms' and a brief description indicating the current time slot that the user is browsing. Below, it has a search form that allows users to filter classrooms based on required equipment. The user can enter their requirements and click the 'Search' button directly below it to filter the results. Finally, a table displaying all the available classrooms at that particular time slot contains some key information about each classroom, such as the classroom number, the floor, its capacity, and the equipment it owns. At the right, there are buttons to make a reservation for the selected classroom at that particular time slot.

4.5 Success Notification Page

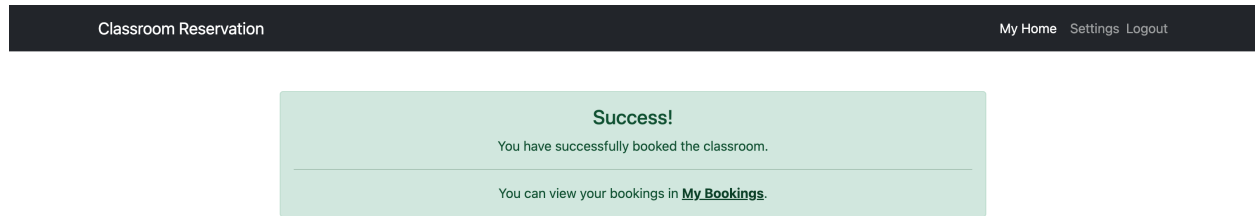


Figure 11. The success notification page

Navigation Bar

The top navigation bar includes a link to the user's homepage, as well as a logout functionality.

Main Content Section

The overall structure is a Bootstrap alert component that displays a success message. Besides, it also provides a link to view the user's bookings under 'My Bookings'.

4.6 Admin Site Page

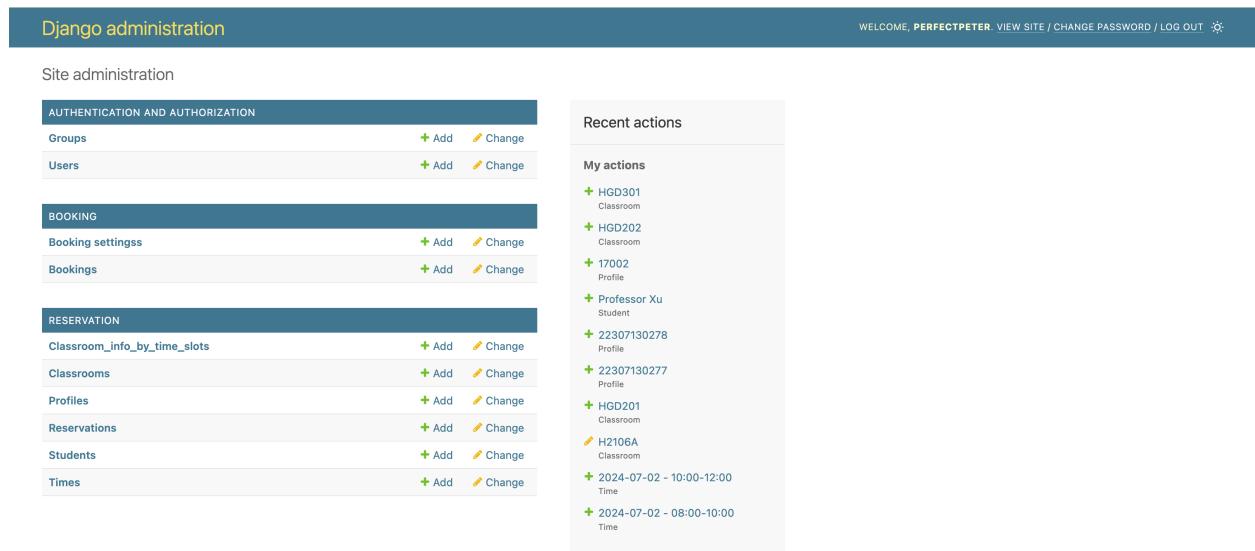


Figure 12. The admin site overview

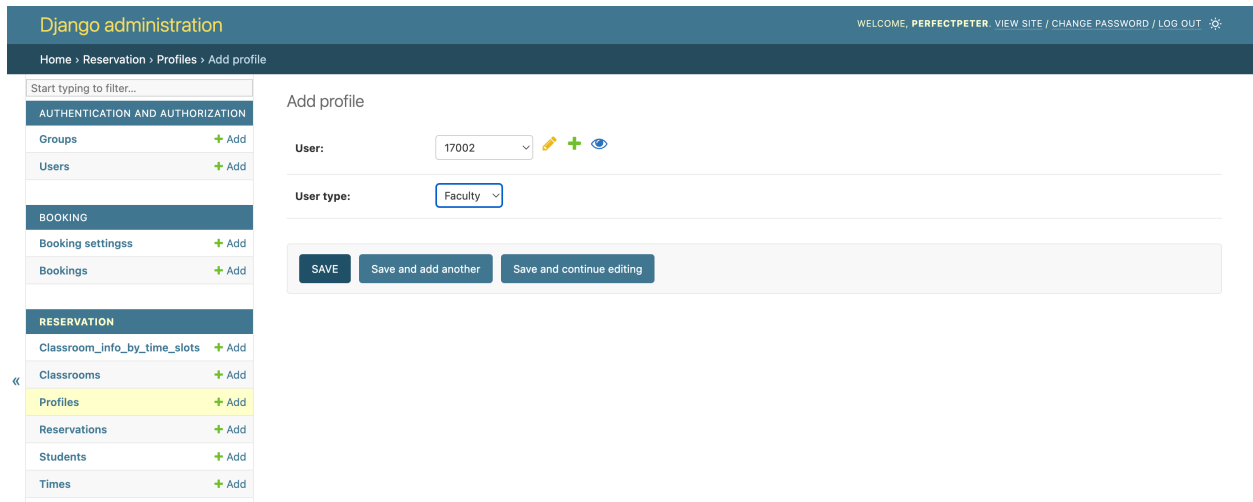
Navigation Bar

The top navigation bar here provides links to change the password of the current user (as an administrator), and to log out as before. Besides, it also provides an additional functionality to change between day and night mode.

Student Administration

There is a custom form for managing student records, including fields for ID, name, password, and major. The form ensures that passwords are correctly entered. Besides, I customized the display list, which provides a quick overview of all student users.

Profile Administration



The screenshot shows the Django administration interface for adding a user profile. The top navigation bar includes the Django logo, the title 'Django administration', and user information: 'WELCOME, PERFECTPETER. VIEW SITE / CHANGE PASSWORD / LOG OUT'. The breadcrumb trail is 'Home > Reservation > Profiles > Add profile'. On the left, a sidebar menu lists various models under categories: AUTHENTICATION AND AUTHORIZATION (Groups, Users), BOOKING (Booking settings, Bookings), and RESERVATION (Classroom_info_by_time_slots, Classrooms, Profiles, Reservations, Students, Times). The 'Profiles' model is highlighted. The main content area is titled 'Add profile' and contains a form with a 'User' dropdown menu (showing '17002') and a 'User type' dropdown menu (showing 'Faculty'). Below the form are three buttons: 'SAVE', 'Save and add another', and 'Save and continue editing'.

Figure 13. The user profile managing page, which supports designating different roles to users

There is a custom for managing user profiles. By selecting the user and the type (currently including student and faculty), the profile administration makes it easy to manage different types of users.

Classroom_info_by_time_slot

As we might expect, it manages the availability of classrooms based on time slots. I customized the list display to include the classroom, time, and availability, with filters to sort availability status.

Classroom, Times & Reservation

For the administrative management of these three models, the functionalities provided by Django's default admin are enough. Therefore, I did not add anything else.

5. Conclusion

This classroom booking system was developed under Django framework, incorporating both SQLite-based database and frontend tools like Bootstrap. The system is suitable for campus-scale facilities managing. Apart from the core functions of looking up and booking vacant classrooms, it also supports filtering rooms based on users' individual requirements. Besides, it is also a perfect platform for institutions that have varied types of users (like undergraduates, graduates, post-docs, faculty and administrative staff), since it allows assigning different permissions to different roles.