

**ADAJAR, MILVI M.**  
**BSCpE 2A2**  
**Software Design**

## **Laboratory Activity No. 2:**

**Topic belongs to: Software Design and Database Systems**

**Title:** *Designing the Database Schema for the Library Management System*

---

**Introduction:** In this activity, you will design the database schema for the Library Management System. The database will include tables for books, authors, users, and borrowing records. You will also learn how to use Django's ORM (Object-Relational Mapping) to define the models.

---

### **Objectives:**

- Design the database schema for the Library Management System.
  - Create Django models to represent the schema.
  - Use Django's ORM to interact with the database.
- 

**Theory and Detailed Discussion:** Django uses an ORM (Object-Relational Mapping) system to map Python objects to database tables. By defining models in Python code, Django automatically creates the corresponding database tables. We will start by designing the database schema with the necessary relationships between entities like books, authors, and users.

---

### **Materials, Software, and Libraries:**

- **Django** framework
  - **SQLite** database (default in Django)
- 

**Time Frame:** 2 Hours

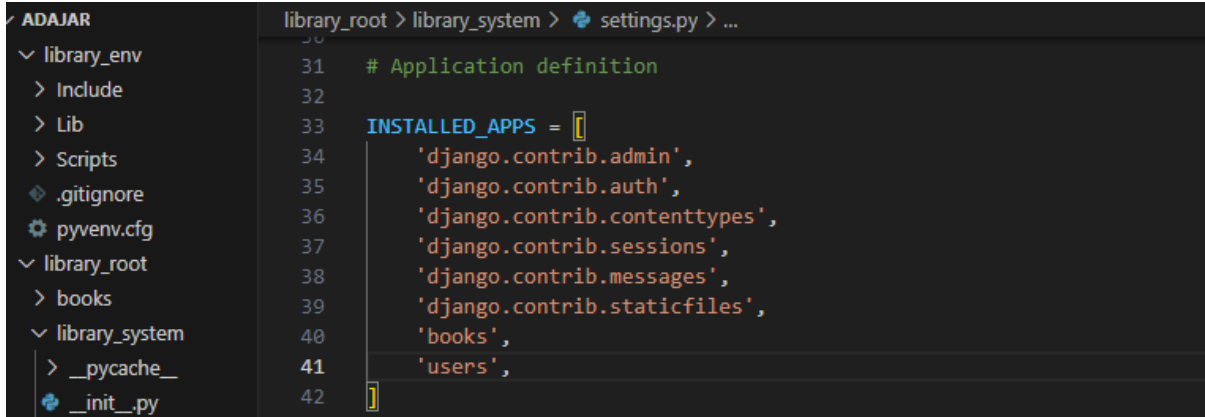
---

### **Procedure:**

1. **Create Django Apps:**
  - In Django, an app is a module that handles a specific functionality. To keep things modular, we will create two apps: one for managing books and another for managing users.

```
python manage.py startapp books
python manage.py startapp users
```

```
[05/Feb/2025 14:34:50] "GET / HTTP/1.1" 200 12068
[05/Feb/2025 14:37:04] "GET / HTTP/1.1" 200 12068
(library_env) PS C:\Users\EB204_U01\Desktop\ADAJAR\library_system> cd..
(library_env) PS C:\Users\EB204_U01\Desktop\ADAJAR> cd library_root
(library_env) PS C:\Users\EB204_U01\Desktop\ADAJAR\library_root> python manage.py startapp books
(library_env) PS C:\Users\EB204_U01\Desktop\ADAJAR\library_root> python manage.py startapp users
(library_env) PS C:\Users\EB204_U01\Desktop\ADAJAR\library_root> []
```



```
library_root > library_system > settings.py > ...
31 # Application definition
32
33 INSTALLED_APPS = [
34     'django.contrib.admin',
35     'django.contrib.auth',
36     'django.contrib.contenttypes',
37     'django.contrib.sessions',
38     'django.contrib.messages',
39     'django.contrib.staticfiles',
40     'books',
41     'users',
42 ]
```

## 2. Define Models for the Books App:

- Open the books/models.py file and define the following models:

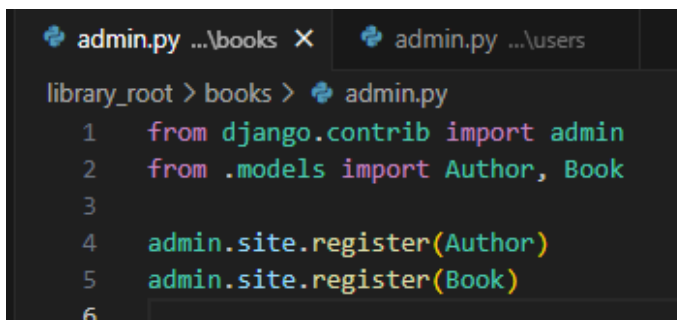
```
from django.db import models

class Author(models.Model):
    name = models.CharField(max_length=100)
    birth_date = models.DateField()

    def __str__(self):
        return self.name

class Book(models.Model):
    title = models.CharField(max_length=200)
    author = models.ForeignKey(Author, on_delete=models.CASCADE)
    isbn = models.CharField(max_length=13)
    publish_date = models.DateField()

    def __str__(self):
        return self.title
```



```
admin.py ...\books X admin.py ...\users
library_root > books > admin.py
1 from django.contrib import admin
2 from .models import Author, Book
3
4 admin.site.register(Author)
5 admin.site.register(Book)
6
```

## 3. Define Models for the Users App:

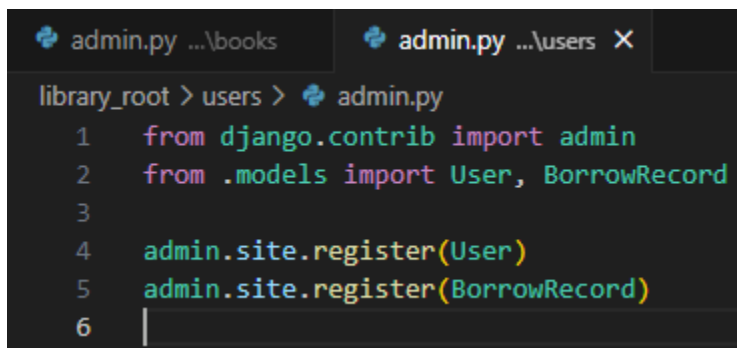
- Open the `users/models.py` file and define the following models:

```
from django.db import models
from books.models import Book

class User(models.Model):
    username = models.CharField(max_length=100)
    email = models.EmailField()

    def __str__(self):
        return self.username

class BorrowRecord(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    book = models.ForeignKey(Book, on_delete=models.CASCADE)
    borrow_date = models.DateField()
    return_date = models.DateField(null=True, blank=True)
```

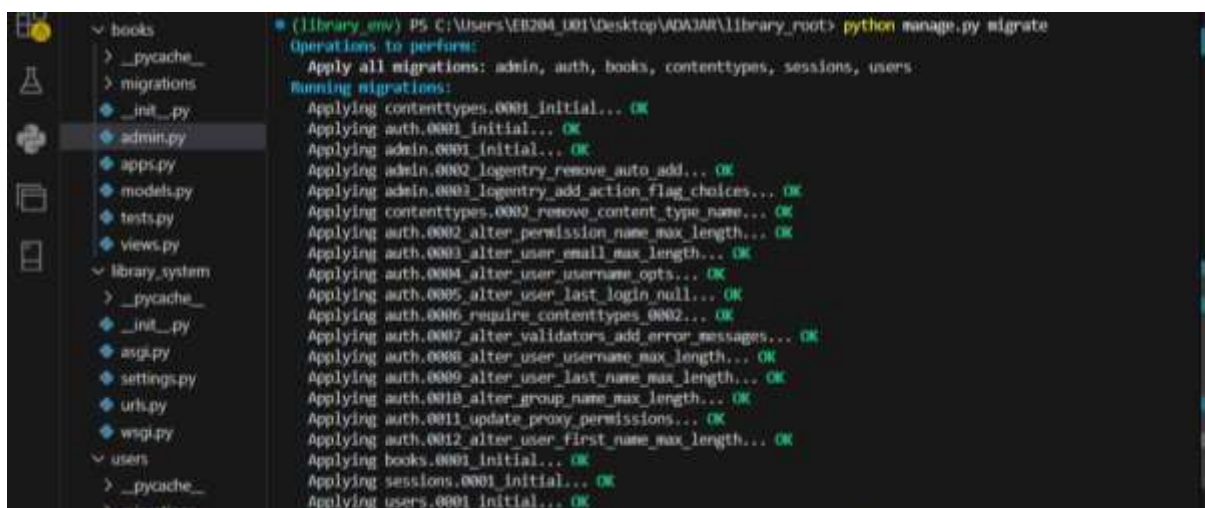


```
admin.py ...\books  admin.py ...\users X
library_root > users > admin.py
1  from django.contrib import admin
2  from .models import User, BorrowRecord
3
4  admin.site.register(User)
5  admin.site.register(BorrowRecord)
6  |
```

#### 4. Apply Migrations:

- To create the database tables based on the models, run the following commands:

```
python manage.py makemigrations
python manage.py migrate
```

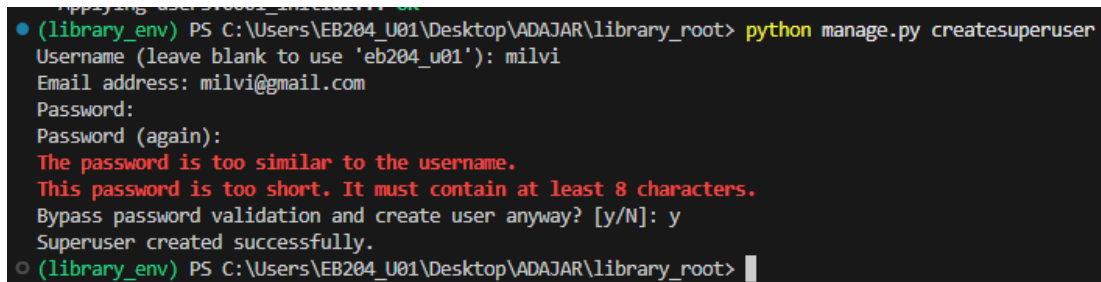


```
(library_env) PS C:\Users\EB204_001\Desktop\ADAJAR\library_root> python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, books, contenttypes, sessions, users
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying books.0001_initial... OK
  Applying sessions.0001_initial... OK
  Applying users.0001_initial... OK
```

## 5. Create Superuser for Admin Panel:

- Create a superuser to access the Django admin panel:

```
python manage.py createsuperuser
```



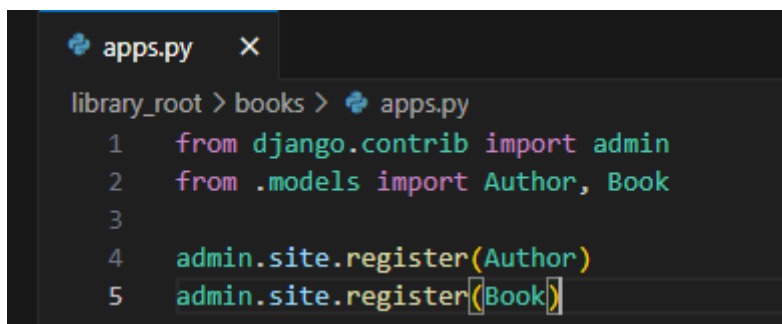
```
(library_env) PS C:\Users\EB204_U01\Desktop\ADAJAR\library_root> python manage.py createsuperuser
Username (leave blank to use 'eb204_u01'): milvi
Email address: milvi@gmail.com
Password:
Password (again):
The password is too similar to the username.
This password is too short. It must contain at least 8 characters.
Bypass password validation and create user anyway? [y/N]: y
Superuser created successfully.
(library_env) PS C:\Users\EB204_U01\Desktop\ADAJAR\library_root>
```

## 6. Register Models in Admin Panel:

- In books/admin.py, register the Author and Book models:

```
from django.contrib import admin
from .models import Author, Book

admin.site.register(Author)
admin.site.register(Book)
```

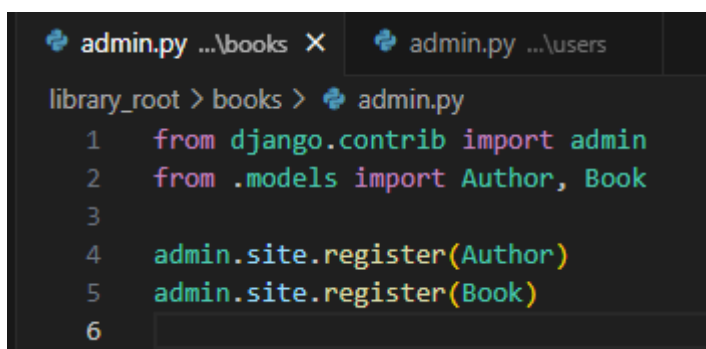


```
apps.py
library_root > books > apps.py
1  from django.contrib import admin
2  from .models import Author, Book
3
4  admin.site.register(Author)
5  admin.site.register(Book)
```

- In users/admin.py, register the User and BorrowRecord models:

```
from django.contrib import admin
from .models import User, BorrowRecord

admin.site.register(User)
admin.site.register(BorrowRecord)
```



```
admin.py ...\books  admin.py ...\users
library_root > books > admin.py
1  from django.contrib import admin
2  from .models import Author, Book
3
4  admin.site.register(Author)
5  admin.site.register(Book)
6
```

## 7. Run the Development Server:

- Start the server again to access the Django admin panel:

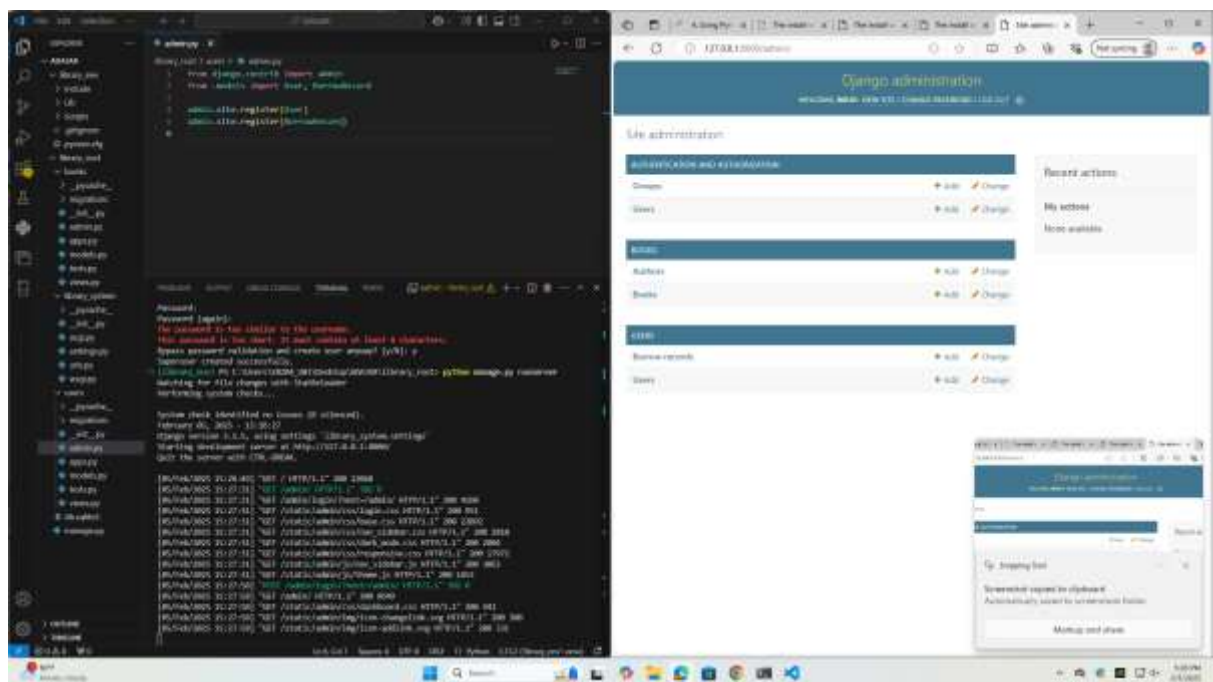
```
python manage.py runserver
```

```
superuser created successfully.
(library_env) PS C:\Users\EB204_U01\Desktop\ADAJAR\library_root> python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
February 05, 2025 - 15:26:27
Django version 5.1.5, using settings 'library_root.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

## 8. Access Admin Panel:

- Go to <http://127.0.0.1:8000/admin> and log in using the superuser credentials. You should see the Author, Book, User, and BorrowRecord models.

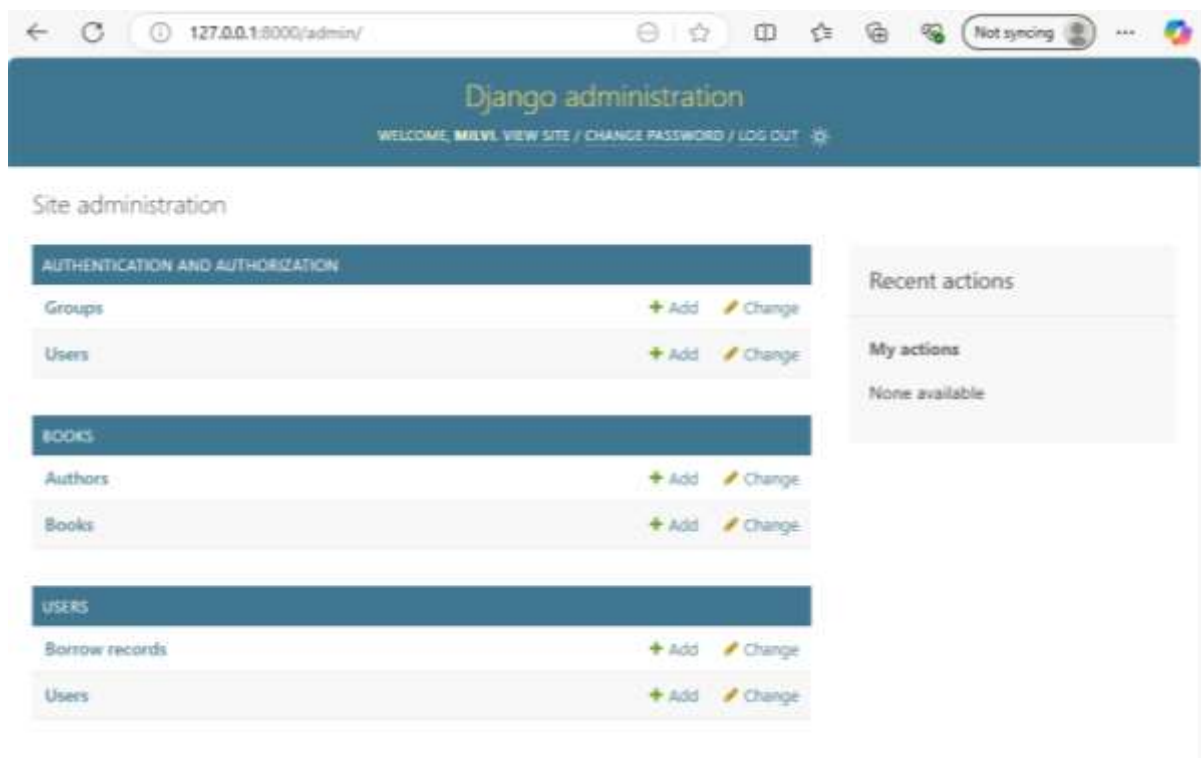


**Django Program or Code:** Write down the summary of the code for models that has been provided in this activity.

- The Django models define a **Library Management System** with two apps: **Books** and **Users**. The **Books** app includes an Author model to store author details and a Book model that stores book information while linking to an Author via a **ForeignKey**. The **Users** app contains a User model for user details and a BorrowRecord model to track book loans, linking User and Book using **ForeignKey** relationships. The system utilizes Django's **ORM (Object-Relational Mapping)** for database interactions and registers models in the **admin panel** for easy management.

---

**Results:** By the end of this activity, you will have successfully defined the database schema using Django models, created the corresponding database tables, and registered the models in the admin panel. (print screen the result and provide the github link of your work)



**Github Link:**

---

### **Follow-Up Questions:**

1. What is the purpose of using ForeignKey in Django models?
    - A virtual environment isolates the dependencies of different projects, ensuring that each project has its own version of libraries without conflicting with global installations. This makes development more manageable and prevents version-related issues.
  2. How does Django's ORM simplify database interaction?
    - Django enables rapid development with built-in features like authentication and ORM, ensures security against common threats, supports scalability for large applications, and benefits from strong community support and documentation.
- 

### **Findings:**

During the setup process, it was observed that installing and configuring Django was straightforward, provided that dependencies were correctly installed. Some common issues included virtual environment activation errors in Windows, which were resolved by adjusting the execution policy.

---

### **Summary:**

The key steps in this activity included installing Python, setting up a virtual environment, installing Django, creating a project, and running the development server. The activity provided an understanding of the importance of an isolated environment and the basic setup required for Django projects.

---

### **Conclusion:**

Setting up the Django development environment is a crucial first step in web application development. This activity demonstrated the importance of using a virtual environment to manage dependencies, ensuring a smooth and organized development workflow. By following the outlined steps, developers can create and manage Django projects efficiently.