Name: Nic Albert B. Seraño
Cours/S.Y: BSCpE-2B

Laboratory Activity No. 3:

Topic belongs to: Programming Constructs and Paradigms
Title: Implementing CRUD Operations for Books and Users in the Library Management System
Introduction:
This activity involves implementing CRUD (Create, Read, Update, Delete) operations for books and users in the Library Management System. You will create views and forms to handle these operations via a web interface.

Objectives:

- Create views for CRUD operations.
- Use Django forms to manage user input.
- Learn about Django's URL routing for view handling.

Theory and Detailed Discussion:
CRUD operations are the fundamental actions in any database-driven application. Django's views and forms make it easy to manage the interaction between the user and the database. Using Django's URL routing system, we can map different actions (like creating or updating data) to specific views.

Materials, Software, and Libraries:

- Django Framework
- SQLite (or any database you are using)

Time Frame:
2 hours

Step by Step Procedure:

1. Create views for handling CRUD operations:
    - Open books/views.py and add the following views:

```
from django.shortcuts import render, redirect
from .models import Book
from .forms import BookForm
```

```
def book_list(request):
    books = Book.objects.all()
    return render(request, 'books/book_list.html', {'books': books})

def book_create(request):
    if request.method == 'POST':
        form = BookForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('book_list')
    else:
        form = BookForm()
    return render(request, 'books/book_form.html', {'form': form})

def book_update(request, pk):
    book = Book.objects.get(pk=pk)
    if request.method == 'POST':
        form = BookForm(request.POST, instance=book)
        if form.is_valid():
            form.save()
            return redirect('book_list')
    else:
        form = BookForm(instance=book)
    return render(request, 'books/book_form.html', {'form': form})

def book_delete(request, pk):
    book = Book.objects.get(pk=pk)
    if request.method == 'POST':
        book.delete()
        return redirect('book_list')
```

```
   return render(request, 'books/book_confirm_delete.html', {'book':
book})
```

2. Create a `BookForm` in `books/forms.py`:

Create first the forms.py then inside of it past the code.

```
from django import forms
from .models import Book
class BookForm(forms.ModelForm):
   class Meta:
       model = Book
       fields = ['title', 'author', 'isbn', 'publish_date']
```

3. Define the URLs for the views in `books/urls.py`:

Under the urls.py

```
from django.urls import path
from . import views

urlpatterns = [
   path('', views.book_list, name='book_list'),
   path('create/', views.book_create, name='book_create'),
   path('update/<int:pk>/', views.book_update,
name='book_update'),
   path('delete/<int:pk>/', views.book_delete,
name='book_delete'),
]
```

4. Register the URLs in `library_system/urls.py`:

```
from django.contrib import admin
```

```
from django.urls import path, include


urlpatterns = [
    path('admin/', admin.site.urls),
    path('books/', include('books.urls')),
]
```

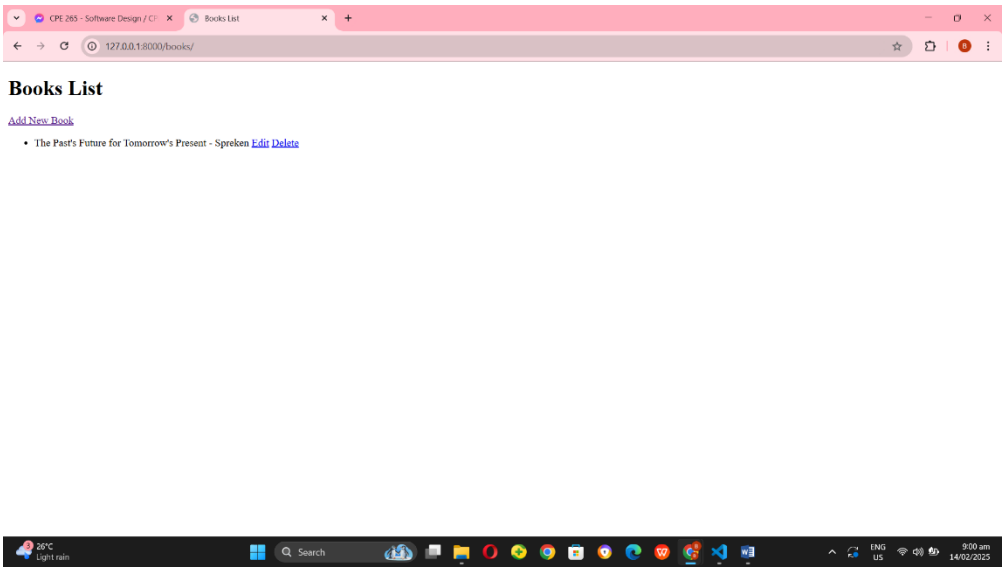
5. Create the HTML templates (`book_list.html`, `book_form.html`, `book_confirm_delete.html`) to render the CRUD forms.

Inside the books (1) create new folder named templates (3). Inside the templates(3) create a folder books (4).

Register the templates directory to the Settings template directory

Django Program or Code:
Code for views, forms, and URL patterns has been provided above.

Results:
The user can now view the list of books, add new books, update existing books, and delete books.



Github link: https://github.com/NicAlbert9/library_system

Follow-Up Questions:

1. How do Django forms work with models for CRUD operations?

   Ans:
   Django forms simplify handling user input for creating, updating, and deleting records. Using **ModelForms**, Django automatically generates form fields based on a model's attributes, making it easier to connect the form to the database. For example, `BookForm` linked to the `Book` model allows me to reuse the form for both creating and editing books, ensuring consistency and reducing redundant code. The form also handles validation before saving data to the database.

2. What is the role of the redirect() function in Django views?

   Ans:
   `redirect()` function is used to navigate users to another page after performing an action, such as adding, updating, or deleting a record. This prevents unnecessary form resubmissions. For example after a book is added successfully, I used `redirect('book_list')` to send users back to the book list page. This ensures a smooth workflow and avoids unnecessary reloading of the form.

Findings:
Through this activity, I was able to successfully implement CRUD operations for both **books** and **users** in the Library Management System. This includes:
Viewing a list of books and users
Adding new records
Updating existing records
Deleting records

Summary:
This activity helped me understand how to manage a database using Django's ORM. By implementing CRUD operations. I now have a functional system where books and users can be managed dynamically.

Conclusion:
CRUD functionality is an essential part of any web application, and this activity allowed me to gain hands-on experience in implementing it. By completing this, I now have a solid

understanding of how to structure a database and build a **fully functional Library Management System** using Django.