# Семинарское задание: Создание нового приложения на Django

## Цель:

Изучить процесс создания нового приложения в Django, включая настройку моделей, представлений, маршрутов и форм, а также реализацию базового интерфейса с использованием HTML и CSS.

## Задание:

Создайте приложение, моделирующее систему управления библиотекой. Приложение должно поддерживать следующие функции:  
- Управление книгами.  
- Управление авторами.  
- Управление категориями.  
- Отображение списка книг и деталей каждой книги.

### Шаг 1: Создание проекта и приложения

1. Установите Django, если он еще не установлен:  
 ```bash  
 pip install django  
 ```  
2. Создайте новый проект Django:  
 ```bash  
 django-admin startproject library\_project  
 cd library\_project  
 ```  
3. Создайте приложение для управления библиотекой:  
 ```bash  
 python manage.py startapp library  
 ```  
4. Зарегистрируйте приложение `library` в настройках проекта:  
 В файле `settings.py` добавьте:  
 ```python  
 INSTALLED\_APPS = [  
 # Другие приложения  
 'library',  
 ]  
 ```

### Шаг 2: Настройка моделей

1. В файле `library/models.py` создайте следующие модели:  
 - `Author`: модель для хранения информации об авторах книг (имя, биография).  
 - `Category`: модель для категорий книг (например, "Наука", "Художественная литература").  
 - `Book`: модель для книг, содержащая следующие поля:  
 - `title` (название),  
 - `author` (связь с моделью `Author`),  
 - `category` (связь с моделью `Category`),  
 - `description` (описание),  
 - `published\_date` (дата публикации).  
  
2. Пример модели:  
 ```python  
 from django.db import models  
  
 class Author(models.Model):  
 name = models.CharField(max\_length=200)  
 biography = models.TextField()  
  
 def \_\_str\_\_(self):  
 return self.name  
  
 class Category(models.Model):  
 name = models.CharField(max\_length=100)  
  
 def \_\_str\_\_(self):  
 return self.name  
  
 class Book(models.Model):  
 title = models.CharField(max\_length=200)  
 author = models.ForeignKey(Author, on\_delete=models.CASCADE)  
 category = models.ForeignKey(Category, on\_delete=models.CASCADE)  
 description = models.TextField()  
 published\_date = models.DateField()  
  
 def \_\_str\_\_(self):  
 return self.title  
 ```  
  
3. Примените миграции:  
 ```bash  
 python manage.py makemigrations  
 python manage.py migrate  
 ```

### Шаг 3: Настройка панели администратора

1. Зарегистрируйте модели в `library/admin.py` для управления через панель администратора:  
 ```python  
 from django.contrib import admin  
 from .models import Author, Category, Book  
  
 admin.site.register(Author)  
 admin.site.register(Category)  
 admin.site.register(Book)  
 ```  
  
2. Создайте суперпользователя:  
 ```bash  
 python manage.py createsuperuser  
 ```  
  
3. Зайдите в панель администратора по адресу: `http://127.0.0.1:8000/admin` и добавьте несколько записей для авторов, категорий и книг.

### Шаг 4: Настройка представлений и маршрутов

1. В файле `library/views.py` создайте следующие представления:  
 - `book\_list`: отображение списка всех книг.  
 - `book\_detail`: отображение деталей конкретной книги.  
  
2. Пример кода:  
 ```python  
 from django.shortcuts import render, get\_object\_or\_404  
 from .models import Book  
  
 def book\_list(request):  
 books = Book.objects.all()  
 return render(request, 'library/book\_list.html', {'books': books})  
  
 def book\_detail(request, book\_id):  
 book = get\_object\_or\_404(Book, id=book\_id)  
 return render(request, 'library/book\_detail.html', {'book': book})  
 ```  
  
3. Настройте маршруты в `library/urls.py`:  
 ```python  
 from django.urls import path  
 from . import views  
  
 urlpatterns = [  
 path('', views.book\_list, name='book\_list'),  
 path('<int:book\_id>/', views.book\_detail, name='book\_detail'),  
 ]  
 ```  
  
4. Подключите маршруты приложения в файле `library\_project/urls.py`:  
 ```python  
 from django.contrib import admin  
 from django.urls import path, include  
  
 urlpatterns = [  
 path('admin/', admin.site.urls),  
 path('', include('library.urls')),  
 ]  
 ```

### Шаг 5: Создание шаблонов

1. Создайте папку `templates/library/` в приложении `library` и добавьте два файла:  
 - `book\_list.html`: шаблон для отображения списка книг.  
 - `book\_detail.html`: шаблон для отображения информации о книге.  
  
2. Пример `book\_list.html`:  
 ```html  
 <!DOCTYPE html>  
 <html>  
 <head>  
 <title>Список книг</title>  
 </head>  
 <body>  
 <h1>Список книг</h1>  
 <ul>  
 {% for book in books %}  
 <li>  
 <a href="{{ book.id }}/">{{ book.title }}</a>  
 - {{ book.author.name }}  
 </li>  
 {% endfor %}  
 </ul>  
 </body>  
 </html>  
 ```  
  
3. Пример `book\_detail.html`:  
 ```html  
 <!DOCTYPE html>  
 <html>  
 <head>  
 <title>{{ book.title }}</title>  
 </head>  
 <body>  
 <h1>{{ book.title }}</h1>  
 <p><strong>Автор:</strong> {{ book.author.name }}</p>  
 <p><strong>Категория:</strong> {{ book.category.name }}</p>  
 <p><strong>Описание:</strong> {{ book.description }}</p>  
 <p><strong>Дата публикации:</strong> {{ book.published\_date }}</p>  
 <a href="/">Вернуться к списку книг</a>  
 </body>  
 </html>  
 ```

### Дополнительные задания:

1. Реализуйте поиск книг по названию или автору.  
2. Создайте форму для добавления новых книг в базу данных.  
3. Реализуйте функционал редактирования и удаления книг.

### Оценка выполнения:

1. \*\*Базовый уровень\*\*: Приложение работает, настроены модели, маршруты и шаблоны.  
2. \*\*Средний уровень\*\*: Добавлены формы для добавления и редактирования данных.  
3. \*\*Высокий уровень\*\*: Приложение полностью стилизовано, реализован поиск и другие дополнительные функции.

### Шаг 6: Создание переходов между страницами в приложении

Для улучшения навигации между страницами добавьте ссылки и переходы в шаблонах приложения.  
1. В шаблоне `book\_list.html` добавьте ссылку на каждую книгу для перехода к её деталям.  
2. В шаблоне `book\_detail.html` добавьте кнопку для возврата к списку всех книг.

### Пример реализации в шаблонах:

Обновленный код `book\_list.html`:  
```html  
<!DOCTYPE html>  
<html>  
<head>  
 <title>Список книг</title>  
</head>  
<body>  
 <h1>Список книг</h1>  
 <ul>  
 {% for book in books %}  
 <li>  
 <a href="{{ book.id }}/">{{ book.title }}</a>  
 - {{ book.author.name }}  
 </li>  
 {% endfor %}  
 </ul>  
</body>  
</html>  
```

Обновленный код `book\_detail.html`:  
```html  
<!DOCTYPE html>  
<html>  
<head>  
 <title>{{ book.title }}</title>  
</head>  
<body>  
 <h1>{{ book.title }}</h1>  
 <p><strong>Автор:</strong> {{ book.author.name }}</p>  
 <p><strong>Категория:</strong> {{ book.category.name }}</p>  
 <p><strong>Описание:</strong> {{ book.description }}</p>  
 <p><strong>Дата публикации:</strong> {{ book.published\_date }}</p>  
 <a href="/">Вернуться к списку книг</a>  
</body>  
</html>  
```

Убедитесь, что маршруты для переходов между страницами уже настроены в файле `library/urls.py`.  
Пример:  
```python  
from django.urls import path  
from . import views  
  
urlpatterns = [  
 path('', views.book\_list, name='book\_list'),  
 path('<int:book\_id>/', views.book\_detail, name='book\_detail'),  
]  
```

### Шаг 7: Создание базового шаблона и его расширение для разных страниц

Для унификации стилей и упрощения работы с HTML-кодом создайте базовый шаблон, который будет использоваться всеми страницами приложения.  
1. Создайте базовый шаблон `base.html`, содержащий структуру HTML-документа и общий контент (например, шапка и подвал).  
2. Настройте страницы `book\_list.html` и `book\_detail.html`, чтобы они расширяли базовый шаблон.

### Пример базового шаблона (`base.html`):

Создайте файл `base.html` в папке `templates/library/`:  
```html  
<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1.0">  
 <title>{% block title %}Зоопарк{% endblock %}</title>  
 <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/css/bootstrap.min.css" rel="stylesheet">  
</head>  
<body>  
 <header class="bg-primary text-white text-center py-3">  
 <h1>Система управления библиотекой</h1>  
 </header>  
 <div class="container mt-4">  
 {% block content %}  
 {% endblock %}  
 </div>  
 <footer class="bg-light text-center py-3 mt-4">  
 <p>&copy; 2024 Библиотека. Все права защищены.</p>  
 </footer>  
</body>  
</html>  
```

### Пример расширения в `book\_list.html`:

Обновите файл `book\_list.html`:  
```html  
{% extends "library/base.html" %}  
  
{% block title %}Список книг{% endblock %}  
  
{% block content %}  
<h2>Список книг</h2>  
<ul class="list-group">  
 {% for book in books %}  
 <li class="list-group-item">  
 <a href="{{ book.id }}/">{{ book.title }}</a> - {{ book.author.name }}  
 </li>  
 {% endfor %}  
</ul>  
{% endblock %}  
```

### Пример расширения в `book\_detail.html`:

Обновите файл `book\_detail.html`:  
```html  
{% extends "library/base.html" %}  
  
{% block title %}{{ book.title }}{% endblock %}  
  
{% block content %}  
<h2>{{ book.title }}</h2>  
<p><strong>Автор:</strong> {{ book.author.name }}</p>  
<p><strong>Категория:</strong> {{ book.category.name }}</p>  
<p><strong>Описание:</strong> {{ book.description }}</p>  
<p><strong>Дата публикации:</strong> {{ book.published\_date }}</p>  
<a href="/" class="btn btn-primary mt-3">Вернуться к списку книг</a>  
{% endblock %}  
```

### Шаг 8: Создание базы данных и её добавление в Django

Для работы приложения необходимо настроить базу данных. Django по умолчанию использует SQLite, но вы можете подключить другую базу данных, например, PostgreSQL или MySQL.  
В этом шаге вы научитесь:  
1. Создавать базу данных.  
2. Настраивать соединение базы данных в Django.  
3. Применять миграции для создания таблиц в базе данных.

#### 1. Настройка базы данных в Django:

По умолчанию в Django используется SQLite. Если вы хотите использовать другую базу данных, выполните следующие шаги:  
  
1. Установите драйвер для выбранной базы данных:  
 - Для PostgreSQL:  
 ```bash  
 pip install psycopg2  
 ```  
 - Для MySQL:  
 ```bash  
 pip install mysqlclient  
 ```  
  
2. Откройте файл `settings.py` вашего проекта и обновите секцию `DATABASES`:  
 - Пример для PostgreSQL:  
 ```python  
 DATABASES = {  
 'default': {  
 'ENGINE': 'django.db.backends.postgresql',  
 'NAME': 'library\_db',  
 'USER': 'your\_username',  
 'PASSWORD': 'your\_password',  
 'HOST': 'localhost',  
 'PORT': '5432',  
 }  
 }  
 ```  
 - Пример для MySQL:  
 ```python  
 DATABASES = {  
 'default': {  
 'ENGINE': 'django.db.backends.mysql',  
 'NAME': 'library\_db',  
 'USER': 'your\_username',  
 'PASSWORD': 'your\_password',  
 'HOST': 'localhost',  
 'PORT': '3306',  
 }  
 }  
 ```

#### 2. Применение миграций:

Миграции используются для создания таблиц базы данных на основе ваших моделей.  
  
1. Создайте миграции для вашего приложения:  
 ```bash  
 python manage.py makemigrations  
 ```  
  
2. Примените миграции:  
 ```bash  
 python manage.py migrate  
 ```  
  
После этого таблицы для моделей `Author`, `Category` и `Book` будут созданы в базе данных.

#### 3. Наполнение базы данных:

Вы можете наполнить базу данных через админ-панель или написать скрипт.  
  
1. Для работы через админ-панель:  
 - Создайте суперпользователя:  
 ```bash  
 python manage.py createsuperuser  
 ```  
 - Зайдите по адресу `http://127.0.0.1:8000/admin` и добавьте записи для авторов, категорий и книг.  
  
2. Для заполнения через скрипт:  
 - Откройте файл `library/management/commands/populate\_db.py` (создайте нужные папки, если их нет):  
 ```python  
 from django.core.management.base import BaseCommand  
 from library.models import Author, Category, Book  
  
 class Command(BaseCommand):  
 def handle(self, \*args, \*\*kwargs):  
 author = Author.objects.create(name="Автор 1", biography="Описание автора 1")  
 category = Category.objects.create(name="Категория 1")  
 Book.objects.create(  
 title="Книга 1",  
 author=author,  
 category=category,  
 description="Описание книги 1",  
 published\_date="2024-01-01"  
 )  
 self.stdout.write("База данных успешно заполнена!")  
 ```  
  
 - Выполните команду:  
 ```bash  
 python manage.py populate\_db  
 ```

#### 4. Проверка соединения:

Убедитесь, что ваше приложение подключено к базе данных:  
1. Запустите сервер разработки:  
 ```bash  
 python manage.py runserver  
 ```  
2. Проверьте работу приложения, убедитесь, что данные отображаются корректно.

### Шаг 9: Добавление пользователей в административной панели Django

Django предоставляет мощную встроенную систему для управления пользователями через административную панель.   
В этом шаге вы научитесь:  
1. Создавать пользователей через административную панель.  
2. Настраивать группы пользователей и их права.  
3. Добавлять авторизацию для доступа к определённым страницам.

#### 1. Создание суперпользователя:

Чтобы управлять пользователями и их правами, сначала создайте суперпользователя.  
1. Выполните команду в терминале:  
 ```bash  
 python manage.py createsuperuser  
 ```  
2. Введите имя пользователя, email и пароль.  
3. После успешного создания суперпользователя войдите в административную панель по адресу `http://127.0.0.1:8000/admin`.

#### 2. Управление пользователями через административную панель:

1. В административной панели перейдите в раздел \*\*Пользователи\*\*.  
2. Нажмите \*\*Добавить пользователя\*\* и заполните необходимые поля:  
 - Имя пользователя,  
 - Email,  
 - Пароль.  
3. Установите статус суперпользователя, если необходимо, или добавьте пользователя в определённые группы.

#### 3. Создание групп и настройка прав:

1. В административной панели перейдите в раздел \*\*Группы\*\*.  
2. Нажмите \*\*Добавить группу\*\*.  
3. Укажите имя группы (например, "Библиотекари") и добавьте права, связанные с моделями `Author`, `Category` и `Book`:  
 - Добавление (`add`),  
 - Изменение (`change`),  
 - Удаление (`delete`),  
 - Просмотр (`view`).  
4. Добавьте пользователей в группу для применения настроенных прав.

#### 4. Ограничение доступа к страницам:

1. Чтобы ограничить доступ к определённым страницам, используйте декораторы `login\_required` или права пользователей.  
2. Пример использования `login\_required` в представлениях (`views.py`):  
 ```python  
 from django.contrib.auth.decorators import login\_required  
  
 @login\_required  
 def book\_list(request):  
 books = Book.objects.all()  
 return render(request, 'library/book\_list.html', {'books': books})  
 ```  
3. Пример использования проверки прав:  
 ```python  
 from django.contrib.auth.decorators import permission\_required  
  
 @permission\_required('library.add\_book', raise\_exception=True)  
 def add\_book(request):  
 # Логика добавления книги  
 pass  
 ```

#### 5. Настройка перенаправления после входа:

1. Добавьте в `settings.py` параметр для перенаправления после успешного входа:  
 ```python  
 LOGIN\_REDIRECT\_URL = '/'  
 ```  
2. Если пользователь не авторизован и пытается открыть защищённую страницу, его перенаправит на страницу входа. Настройте путь к странице входа в `settings.py`:  
 ```python  
 LOGIN\_URL = '/accounts/login/'  
 ```

#### 6. Добавление информации о пользователе в шаблоны:

1. В базовый шаблон добавьте ссылку на вход и выход пользователя.  
2. Пример:  
 ```html  
 <nav>  
 {% if user.is\_authenticated %}  
 <p>Вы вошли как {{ user.username }}</p>  
 <a href="{% url 'logout' %}">Выйти</a>  
 {% else %}  
 <a href="{% url 'login' %}">Войти</a>  
 {% endif %}  
 </nav>  
 ```

### Шаг 10: Создание моделей машинного и глубокого обучения

В этом шаге вы научитесь интегрировать модели машинного и глубокого обучения в Django.   
Мы создадим пример предсказательной модели, которая анализирует текстовые данные и возвращает предсказания.

#### 1. Установка необходимых библиотек:

Для работы с моделями установите библиотеки:  
```bash  
pip install numpy pandas scikit-learn tensorflow  
```

#### 2. Создание обученной модели:

1. Используйте Python для создания обученной модели машинного обучения.  
2. Пример создания модели для классификации текста с использованием `scikit-learn`:  
```python  
import pandas as pd  
from sklearn.feature\_extraction.text import CountVectorizer  
from sklearn.naive\_bayes import MultinomialNB  
from sklearn.pipeline import Pipeline  
import pickle  
  
# Данные для обучения  
data = {'text': ['Книга интересная', 'Книга скучная', 'Отличная книга', 'Не рекомендую'],  
 'label': [1, 0, 1, 0]} # 1 - позитивный, 0 - негативный отзыв  
df = pd.DataFrame(data)  
  
# Создание модели  
pipeline = Pipeline([  
 ('vectorizer', CountVectorizer()),  
 ('classifier', MultinomialNB())  
])  
  
# Обучение  
pipeline.fit(df['text'], df['label'])  
  
# Сохранение модели  
with open('text\_model.pkl', 'wb') as f:  
 pickle.dump(pipeline, f)  
```

#### 3. Интеграция модели в Django:

1. Поместите сохранённый файл `text\_model.pkl` в папку вашего приложения (например, `library/ml\_models/`).  
2. В представлениях загрузите модель и добавьте функцию для обработки запросов.  
Пример:  
```python  
import pickle  
from django.shortcuts import render  
  
# Загрузка модели  
with open('library/ml\_models/text\_model.pkl', 'rb') as f:  
 model = pickle.load(f)  
  
def predict\_sentiment(request):  
 if request.method == 'POST':  
 user\_input = request.POST.get('text')  
 prediction = model.predict([user\_input])  
 sentiment = 'Позитивный' if prediction[0] == 1 else 'Негативный'  
 return render(request, 'library/predict.html', {'sentiment': sentiment})  
 return render(request, 'library/predict.html')  
```

#### 4. Настройка маршрутов:

Добавьте маршрут для функции `predict\_sentiment` в `library/urls.py`:  
```python  
from django.urls import path  
from . import views  
  
urlpatterns = [  
 # Другие маршруты  
 path('predict/', views.predict\_sentiment, name='predict\_sentiment'),  
]  
```

#### 5. Создание шаблона:

Создайте файл `predict.html` в папке `templates/library/` для ввода текста и отображения результата:  
```html  
<!DOCTYPE html>  
<html>  
<head>  
 <title>Анализ текста</title>  
</head>  
<body>  
 <h1>Введите текст для анализа:</h1>  
 <form method="POST">  
 {% csrf\_token %}  
 <textarea name="text" rows="4" cols="50"></textarea><br>  
 <button type="submit">Анализировать</button>  
 </form>  
  
 {% if sentiment %}  
 <h2>Результат: {{ sentiment }}</h2>  
 {% endif %}  
</body>  
</html>  
```

#### 6. Расширение для моделей глубокого обучения:

Вы можете интегрировать модели глубокого обучения, используя библиотеку TensorFlow или PyTorch.   
Например, для анализа текста создайте модель с использованием TensorFlow и сохраните её с помощью формата `SavedModel`.   
Затем загрузите её в представлениях и используйте для предсказаний.