**Міністерство освіти і науки України**

**Національний університет «Запорізька політехніка»**

Кафедра програмних засобів

**ЗВІТ**

Дисципліна «Розробка прикладних програм»

Робота №3

Тема «Розроблення вебдодатків та реалізація доступу до систем керування базами даних через програмні інтерфейси»

**Виконав варіант 19**

Студент КНТ-122 Онищенко О. А.

**Прийняли**

Викладач Дейнега Л. Ю.

2024

Мета роботи

Навчитися використовувати програмні інтерфейси для доступу до баз даних. Навчитися розробляти вебдодатки за допомогою фреймворку Django.

Індивідуальне завдання

Вебдодаток керування фінансами, який забезпечує виконання одноразових та періодичних платежів клієнтами. Початкова кількість коштів на рахунку кожного клієнта та його кредитні ліміти визначаються менеджером, якому доступна вся інформаія про клієнтів. Клієнт може переглядати всю інформацію про власний рахунок (разом з історією транзакцій), вносити кошти, сплачувати за послуги, призначати періодичні платежі.

Тексти файлів

Виділено жирним шрифтом місця з найважливішими змінами.

runner/settings.py

"""

Django settings for runner project.

Generated by 'django-admin startproject' using Django 5.1.3.

For more information on this file, see

https://docs.djangoproject.com/en/5.1/topics/settings/

For the full list of settings and their values, see

https://docs.djangoproject.com/en/5.1/ref/settings/

"""

from pathlib import Path

# Build paths inside the project like this: BASE\_DIR / 'subdir'.

BASE\_DIR = Path(\_\_file\_\_).resolve().parent.parent

# Quick-start development settings - unsuitable for production

# See https://docs.djangoproject.com/en/5.1/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!

SECRET\_KEY = 'django-insecure-14#l+(td8@3g#=m4)jw+%@jk!t6@&cjly6vyctboulo=f=^8p^'

# SECURITY WARNING: don't run with debug turned on in production!

DEBUG = True

ALLOWED\_HOSTS = []

# Application definition

INSTALLED\_APPS = [

'django.contrib.admin',

'django.contrib.auth',

'django.contrib.contenttypes',

'django.contrib.sessions',

'django.contrib.messages',

'django.contrib.staticfiles',

**'finance',**

]

MIDDLEWARE = [

'django.middleware.security.SecurityMiddleware',

'django.contrib.sessions.middleware.SessionMiddleware',

'django.middleware.common.CommonMiddleware',

'django.middleware.csrf.CsrfViewMiddleware',

'django.contrib.auth.middleware.AuthenticationMiddleware',

'django.contrib.messages.middleware.MessageMiddleware',

'django.middleware.clickjacking.XFrameOptionsMiddleware',

]

ROOT\_URLCONF = 'runner.urls'

TEMPLATES = [

{

'BACKEND': 'django.template.backends.django.DjangoTemplates',

'DIRS': [],

'APP\_DIRS': True,

'OPTIONS': {

'context\_processors': [

'django.template.context\_processors.debug',

'django.template.context\_processors.request',

'django.contrib.auth.context\_processors.auth',

'django.contrib.messages.context\_processors.messages',

],

},

},

]

WSGI\_APPLICATION = 'runner.wsgi.application'

# Database

# https://docs.djangoproject.com/en/5.1/ref/settings/#databases

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.**mysql'**,

**'USER': 'root',**

**'PASSWORD': '1313',**

**'HOST': 'localhost',**

**'PORT': '3306',**

**'NAME': 'data',**

}

}

# Password validation

# https://docs.djangoproject.com/en/5.1/ref/settings/#auth-password-validators

AUTH\_PASSWORD\_VALIDATORS = [

{

'NAME': 'django.contrib.auth.password\_validation.UserAttributeSimilarityValidator',

},

{

'NAME': 'django.contrib.auth.password\_validation.MinimumLengthValidator',

},

{

'NAME': 'django.contrib.auth.password\_validation.CommonPasswordValidator',

},

{

'NAME': 'django.contrib.auth.password\_validation.NumericPasswordValidator',

},

]

# Internationalization

# https://docs.djangoproject.com/en/5.1/topics/i18n/

LANGUAGE\_CODE = 'en-us'

TIME\_ZONE = 'UTC'

USE\_I18N = True

USE\_TZ = True

# Static files (CSS, JavaScript, Images)

# https://docs.djangoproject.com/en/5.1/howto/static-files/

STATIC\_URL = 'static/'

# Default primary key field type

# https://docs.djangoproject.com/en/5.1/ref/settings/#default-auto-field

DEFAULT\_AUTO\_FIELD = 'django.db.models.BigAutoField'

runner/urls.py

"""

URL configuration for runner project.

The `urlpatterns` list routes URLs to views. For more information please see:

https://docs.djangoproject.com/en/5.1/topics/http/urls/

Examples:

Function views

1. Add an import: from my\_app import views

2. Add a URL to urlpatterns: path('', views.home, name='home')

Class-based views

1. Add an import: from other\_app.views import Home

2. Add a URL to urlpatterns: path('', Home.as\_view(), name='home')

Including another URLconf

1. Import the include() function: from django.urls import include, path

2. Add a URL to urlpatterns: path('blog/', include('blog.urls'))

"""

from django.contrib import admin

from django.urls import include, path

urlpatterns = [

path('admin/', admin.site.urls),

**path('', include('finance.urls')),**

]

finance/template/client.html

<a href="/">Home</a>

<h1>{{client.name}}</h1>

<p>Balance: {{client.balance}}</p>

<p>Limit: {{client.credit}}</p>

<a href="{% url 'deposit' id=client.id%}">Deposit</a>

<a href="{% url 'withdraw' id=client.id%}">Withdraw</a>

{%if periodic\_payments%}

<h2>Periodic Payments</h2>

<table>

<tr>

<td>Amount</td>

<td>Purpose</td>

<td>Period</td>

<td>Next payment</td>

<td>Action</td>

</tr>

{%for row in periodic\_payments%}

<tr>

<td>{{row.amount}}</td>

<td>{{row.purpose}}</td>

<td>{{row.period}}</td>

<td>{{row.next\_date|date:'d.m.Y'}}</td>

<td><a href="{%url 'pay\_period' payment\_id=row.id%}">Pay</a></td>

</tr>

{%endfor%}

</table>

{%endif%}

<a href="{%url 'periodic' id=client.id%}">Periodic</a>

{%if client.manager%}

<h2>Clients Data</h2>

<table>

<tr>

<td>ID</td>

<td>Name</td>

<td>Balance</td>

<td>Credit</td>

<td>Manager?</td>

<td>Actions</td>

</tr>

{%for c in clients%}

<tr>

<td>{{c.id}}</td>

<td>{{c.name}}</td>

<td>{{c.balance}}</td>

<td>{{c.credit}}</td>

<td>{{c.manager}}</td>

<td>

<a href="{%url 'edit' id=c.id admin=client.id%}">Edit</a>

</td>

</tr>

{%endfor%}

</table>

{%endif%}

{%if payments%}

<h2>Payments</h2>

<table>

<tr>

<td>Amount</td>

<td>Purpose</td>

<td>Operation</td>

<td>Type</td>

<td>Time</td>

</tr>

{%for row in payments reversed%}

<tr>

<td>{{row.amount}}</td>

<td>{{row.purpose}}</td>

<td>{{row.operation}}</td>

<td>{{row.kind}}</td>

<td>{{row.timestamp|date:'d.m.Y H:i:s'}}</td>

</tr>

{%endfor%}

</table>

{%endif%}

finance/template/deposit.html

<a href="{%url 'client' id=client.id%}">Back</a>

<h1>Deposit for {{client.name}}</h1>

<form method="post">

{%csrf\_token%}

{{form}}

<button type="submit">Amen</button>

</form>

finance/template/edit.html

<a href="{%url 'client' id=admin%}">Back</a>

<h1>Edit {{client.name}}</h1>

<form method="post">

{%csrf\_token%}

{{form}}

<button type="submit">Amen</button>

</form>

finance/template/home.html

<h1>Welcome</h1>

<form method="post">

{%csrf\_token%}

{{form}}

<button type="submit">Amen</button>

</form>

finance/template/periodic.html

<a href="{%url 'client' id=client.id%}">Back</a>

<h1>Adding periodic payment for {{client.name}}</h1>

<form method="post">

{%csrf\_token%}

{{form}}

<button type="submit">Amen</button>

</form>

finance/template/withdraw.html

<a href="{%url 'client' id=client.id%}">Back</a>

<h1>Withdraw for {{client.name}}</h1>

<form method="post">

{%csrf\_token%}

{{form}}

<button type="submit">Amen</button>

</form>

finance/admin.py

from django.contrib import admin

from .models import Client, Payment, PeriodicPayment

class ClientAdmin(admin.ModelAdmin):

list\_display=['name','balance','credit','manager']

class PaymentAdmin(admin.ModelAdmin):

list\_display=['client','amount','purpose','operation','kind','timestamp',]

class PeriodicPaymentAdmin(admin.ModelAdmin):

list\_display=['client','amount','purpose','period','next\_date',]

admin.site.register(Client,ClientAdmin)

admin.site.register(Payment,PaymentAdmin)

admin.site.register(PeriodicPayment,PeriodicPaymentAdmin)

finance/forms.py

from django import forms

from .models import PeriodicPayment

class NameForm(forms.Form):

name = forms.CharField(label='Client name')

class DepositForm(forms.Form):

amount = forms.IntegerField(label='Amount to deposit')

purpose = forms.CharField()

class WithdrawForm(forms.Form):

amount = forms.IntegerField(label='Amount to withdraw')

purpose = forms.CharField()

class EditForm(forms.Form):

name=forms.CharField()

balance=forms.IntegerField()

credit=forms.IntegerField()

manager=forms.BooleanField(required=False)

class PeriodicForm(forms.Form):

amount = forms.IntegerField(label='Amount to pay')

period = forms.ChoiceField(choices=PeriodicPayment.PERIODS)

purpose = forms.CharField()

finance/models.py

from django.db import models

class Client(models.Model):

name=models.CharField(max\_length=127)

balance=models.PositiveIntegerField(default=0)

credit=models.PositiveIntegerField(default=0)

manager=models.BooleanField(default=False)

def \_\_str\_\_(self):

return self.name

class Payment(models.Model):

client=models.ForeignKey(Client,on\_delete=models.CASCADE)

timestamp=models.DateTimeField()

amount=models.PositiveIntegerField()

purpose=models.TextField()

OPERATIONS=[

('Withdrawal','Withdrawal'),

('Deposit','Deposit'),

]

operation=models.CharField(max\_length=12,choices=OPERATIONS)

KINDS=[

('Single','Single'),

('Periodic','Periodic'),

]

kind=models.CharField(max\_length=12,choices=KINDS)

def \_\_str\_\_(self):

return f'{self.purpose} on {self.timestamp.strftime('%d.%m.%Y at %H:%M:%S')} by {self.client.name} for {self.amount}'

class PeriodicPayment(models.Model):

client=models.ForeignKey(Client,on\_delete=models.CASCADE)

amount=models.PositiveIntegerField()

purpose=models.TextField()

PERIODS=[

('Day','Day'),

('Month','Month'),

('Year','Year'),

]

period=models.CharField(max\_length=12,choices=PERIODS)

next\_date=models.DateField()

def \_\_str\_\_(self):

return f'Every {self.period.lower()} by {self.client.name} for {self.amount} at {self.next\_date.strftime('%d.%m.%Y')}'

finance/urls.py

from django.urls import path

from . import views

urlpatterns = [

path('', views.home, name='home'),

path('client/<int:id>', views.client, name='client'),

path('deposit/<int:id>', views.deposit, name='deposit'),

path('withdraw/<int:id>', views.withdraw, name='withdraw'),

path('edit/<int:admin>/<int:id>', views.edit, name='edit'),

path('periodic/<int:id>', views.periodic, name='periodic'),

path('pay\_period/<int:payment\_id>', views.pay\_period, name='pay\_period'),

]

finance/views.py

from django.forms.models import model\_to\_dict

from django.shortcuts import redirect, render

from django.utils.timezone import make\_aware

import datetime

from .forms import \*

from .models import Client, Payment, PeriodicPayment

def home(request):

if request.method=='POST':

form=NameForm(request.POST)

if not form.is\_valid():

return redirect('home')

name=form.cleaned\_data['name']

client=Client.objects.filter(name=name).first()

if not client:

client=Client(name=name)

client.save()

return redirect('client', id=client.id)

if request.method=='GET':

form = NameForm()

return render(request, 'home.html', {'form':form})

def client(request, id):

client=Client.objects.get(pk=id)

clients=Client.objects.all().values() if client.manager else None

payments=Payment.objects.filter(client=client).values()

periodic\_payments=PeriodicPayment.objects.filter(client=client).values()

return render(request, 'client.html', {'client':client,'clients':clients,'payments':payments,'periodic\_payments':periodic\_payments})

def deposit(request, id):

client=Client.objects.get(pk=id)

if request.method=='POST':

form=DepositForm(request.POST)

if not form.is\_valid():

return redirect('client', id=client.id)

amount=form.cleaned\_data['amount']

if client.credit < amount:

return redirect('client', id=client.id)

client.credit-=amount

client.balance+=amount

client.save()

payment=Payment(

client=client,

timestamp=make\_aware(datetime.datetime.now()),

purpose=form.cleaned\_data['purpose'],

amount=amount,

operation=dict(Payment.OPERATIONS)['Deposit'],

kind=dict(Payment.KINDS)['Single']

)

payment.save()

return redirect('client', id=client.id)

if request.method=='GET':

form=DepositForm()

return render(request, 'deposit.html', {'form':form,'client':client})

def withdraw(request, id):

client=Client.objects.get(pk=id)

if request.method=='POST':

form=WithdrawForm(request.POST)

if not form.is\_valid():

return redirect('client', id=client.id)

amount=form.cleaned\_data['amount']

if client.balance < amount:

return redirect('client', id=client.id)

client.balance-=amount

client.credit+=amount

client.save()

payment=Payment(

client=client,

timestamp=make\_aware(datetime.datetime.now()),

purpose=form.cleaned\_data['purpose'],

amount=amount,

operation=dict(Payment.OPERATIONS)['Withdrawal'],

kind=dict(Payment.KINDS)['Single']

)

payment.save()

return redirect('client', id=client.id)

if request.method=='GET':

form=WithdrawForm()

return render(request, 'withdraw.html', {'form':form,'client':client})

def edit(request, id, admin):

client=Client.objects.get(pk=id)

if request.method=='POST':

form=EditForm(request.POST)

if not form.is\_valid():

return redirect('client', id=client.id)

data=form.cleaned\_data

edited\_client=Client.objects.filter(name=data['name']).first()

new\_name=data['name']

new\_balance=data['balance']

new\_credit=data['credit']

new\_status=data['manager']

edited\_client.name=new\_name

edited\_client.balance=new\_balance

edited\_client.credit=new\_credit

edited\_client.manager=new\_status

edited\_client.save()

return redirect('client', id=admin)

if request.method=='GET':

form=EditForm(initial=model\_to\_dict(client))

return render(request, 'edit.html', {'form':form,'client':client,'admin':admin})

def periodic(request, id):

client=Client.objects.get(pk=id)

if request.method=='POST':

form=PeriodicForm(request.POST)

if not form.is\_valid():

return redirect('client',id=client.id)

data=form.cleaned\_data

amount=data['amount']

period=data['period']

purpose=data['purpose']

periodic\_payment=PeriodicPayment(

client=client,

amount=amount,

purpose=purpose,

period=period,

next\_date=datetime.date.today()

)

periodic\_payment.save()

return redirect('client',id=client.id)

if request.method=='GET':

form=PeriodicForm()

return render(request, 'periodic.html', {'form':form,'client':client})

def pay\_period(request, payment\_id):

payment=PeriodicPayment.objects.get(pk=payment\_id)

client=payment.client

if client.balance < payment.amount:

return redirect('client',id=client.id)

client.balance-=payment.amount

client.save()

payment\_log=Payment(

client=client,

timestamp=make\_aware(datetime.datetime.now()),

purpose=payment.purpose,

amount=payment.amount,

operation=dict(Payment.OPERATIONS)['Withdrawal'],

kind=dict(Payment.KINDS)['Periodic']

)

payment\_log.save()

next\_year,next\_month,next\_day=payment.next\_date.year,payment.next\_date.month,payment.next\_date.day

if payment.period=='Day':

next\_day=next\_day+1

print(next\_day)

if next\_day>=28:

next\_day=1

next\_month=next\_month+1

if next\_month>12:

next\_month=1

next\_year=next\_year+1

if payment.period=='Month':

next\_month=next\_month+1

if next\_month>12:

next\_month=1

next\_year=next\_year+1

if payment.period=='Year': next\_year+=1

payment.next\_date=datetime.date(next\_year,next\_month,next\_day)

print(payment.next\_date)

payment.save()

return redirect('client',id=client.id)

Результати виконання

На вході до програми відкривається сторінка входу:

Зображення, що містить текст, знімок екрана, Шрифт, ряд

Автоматично згенерований опис

Рисунок 1.1 – Сторінка входу

При вході до існуючого акаунту відкривається сторінка з інформацією про користувача, періодичні виплати та історію усіх виплат:

Зображення, що містить текст, знімок екрана, Шрифт, документ

Автоматично згенерований опис

Рисунок 1.2 – Сторінка існуючого користувача

Якщо введено ім’я неіснуючого користувача на сторінці входу, ствроюється і відкривається сторінка нового користувача:

Зображення, що містить текст, знімок екрана, Шрифт, білий

Автоматично згенерований опис

Рисунок 1.3 – Сторінка новоствореного користувача

При натисканні кнопки «Ноme» відкривається сторінка входу. При натисканні кнопки «Deposit» відкривається сторінка додавання грошей до балансу:

Зображення, що містить текст, знімок екрана, Шрифт, ряд

Автоматично згенерований опис

Рисунок 1.4 – Сторінка додавання грошей

При натисканні кнопки «Back» відкривається сторінка того ж користувача. Гроші мають братися з кредитного ліміту. Для додавання грошей можна або зробити це через акаунт менеджера, або через адмін панель.

Для зміни кредитного ліміту через адмін панель потрібен адмін акаунт у застосунку. Після входу до нього через *localhost/admin* потрібно обрати таблицю для редагування (в цьому випадку Clients):

Зображення, що містить текст, знімок екрана, Шрифт, програмне забезпечення

Автоматично згенерований опис

Рисунок 1.5 – Вигляд адмін панелі

Тоді потрібно обрати необхідного користувача для редагування інформації:

Зображення, що містить текст, знімок екрана, програмне забезпечення, число

Автоматично згенерований опис

Рисунок 1.6 – Обрання потрібного користувача зі списку

Після зміни значення кредитного ліміту потрібно зберегти зміни через кнопку «SAVE»:

Зображення, що містить текст, знімок екрана, Шрифт, число

Автоматично згенерований опис

Рисунок 1.7 – Збереження змін до кредитного ліміту нового користувача

Тепер при переході на сторінку користувача на основному сайті можна побачити застосовані зміни:

Зображення, що містить текст, знімок екрана, Шрифт, білий

Автоматично згенерований опис

Рисунок 1.8 – Змінений кредитний ліміт нового користувача

Для зміни кредитного ліміту через панель менеджера потрібно зайти у акаунт зі статусом менеджера (*client.manager: bool*):

Зображення, що містить текст, знімок екрана, Шрифт, число

Автоматично згенерований опис

Рисунок 1.9 – Сторінка користувача менеджера

На сторінці менеджера має бути інформація про усіх користувачів з можливістю редагувати дані кожного. При виборі необхідного користувача і натисканні кнопки «Edit» має відкритися сторінка редагування даних:

Зображення, що містить текст, знімок екрана, Шрифт, ряд

Автоматично згенерований опис

Рисунок 1.10 – Сторінка редагування даних нового користувача зі зміненим кредитним лімітом

Після редагування даних і натискання кнопки «Амінь» має відкритися сторінка менеджера:

Зображення, що містить текст, знімок екрана, Шрифт, число

Автоматично згенерований опис

Рисунок 1.11 – Таблиця клієнтів на сторінці менеджера зі зміненим кредитним лімітом нового користувача

Після додавання грошей до кредитного ліміту можна додати їх до рахунку. Для цього потрібно знову перейти до сторінки нового користувача і натиснути кнопку «Deposit»:

Зображення, що містить текст, знімок екрана, Шрифт, ряд

Автоматично згенерований опис

Рисунок 1.12 – Створення транзакції поповнення рахунку нового користувача

При створенні нової транзакції поповнення або зняття грошей завжди потрібно вказувати мету (purpose) транзакції. Після натискання кнопки «Амінь» має відкритися сторінка нового користувача з оновленими даними рахунку та здійсненою транзакцією у історії:

Зображення, що містить текст, знімок екрана, Шрифт

Автоматично згенерований опис

Рисунок 1.13 – Сторінка користувача з проведеною транзакцією поповнення рахунку на 12

При натисканні на кнопку «Withdraw» має вікритися сторінка зняття грошей з рахунку:

Зображення, що містить текст, знімок екрана, Шрифт, ряд

Автоматично згенерований опис

Рисунок 1.14 – Сторінка зняття коштів зі вказаною причиною

Після вказання суми та причини операції користувач має знову повернутися на сторінку профіля з оновленою інформацією:

Зображення, що містить текст, знімок екрана, Шрифт

Автоматично згенерований опис

Рисунок 1.15 – Сторінка користувача з оновленими даними після зняття коштів з рахунку

При натисненні на кнопку «Periodic» має відкритися сторінка створення періодичної транзакції:

Зображення, що містить текст, знімок екрана, Шрифт

Автоматично згенерований опис

Рисунок 1.16 – Сторінка створення періодичної транзакції

Тут можна вказати суму транзакції, періодичність та мету. Після створення нової періодичної транзакції користувача має перенести до сторінки профілю з доданою періодичною транзакцією:

Зображення, що містить текст, знімок екрана, Шрифт

Автоматично згенерований опис

Рисунок 1.17 – Сторінка профілю нового користувача з оновленими даними

При натисканні кнопки «Pay» біля необхідної періодичної транзакції має здійснитися зняття коштів та додавання нової транзакції до історії:

Зображення, що містить текст, знімок екрана, Шрифт

Автоматично згенерований опис

Рисунок 1.18 – Сторінка профіля після здійснення періодичної транзакції

Контрольні питання

Які існують способи збереження даних в програмах написаних мовою Python?

Кілька способів:

* Як локальні змінні: тоді воні зберігатимуться лише на час роботи програми;
* Як файли: тоді їх потрібно вручну читати та записувати при змінах. Поширені формати: JSON, CSV, TXT;
* У базі даних: для цього зазвичай використовують Систему керування базою даних. Поширені СКБД: MySQL, PostgreSQL, MongoDB.

Для чого призначена бібліотека MySQLdb?

Для підключення та взаємодії з СКБД MySQL.

Які існують високорівневі функції бібліотеки MySQLdb?

Найпоширеніші:

* connect: створює підключення з базою даних. Приймає як аргументи назву хоста, ім’я користувача, пароль, опціонально базу даних з якою працювати;
* query: формує та виконує запит до бази даних. Текст запиту пишеться мовою SQL для MySQL;
* store\_result: завантажує результати запиту та зберігає їх локально в повному обсязі;
* use\_result: завантажує результати запиту та зберігає їх на сервері, подає рядок за рядком;
* fetch\_row: показує рядок результатів запиту.

[Джерело](https://mysqlclient.readthedocs.io/user_guide.html#some-mysql-examples)

Яким чином виконати запит до бази даних та яким чином переглянути результати?

Через підключення до бази даних викликом методу connect, формування запиту викликом методу query з текстом запиту мовою SQL, збереження результатів віддалено викликом методу use\_result, та виведення результатів рядок за рядком викликом метду fetch\_row. АЛИЛУЯ

Що таке шоблон проєктування MVC?

Поділяє архітектуру програми на три рівні:

1. *model* (модель): містить всі інформаційні частини додатку;
2. *view* (представлення): описує логіку інтерфейсу та показує дані користувачеві;
3. *controller* (контроллер): інтерфейс між моделлю та представленням. Обробляє запити, модифікує дані, надсилає їх представленню.

[Джерело](https://www.tutorialspoint.com/mvc_framework/mvc_framework_introduction.htm)

За допомогою яких команд виконується розроблення додатків у бібліотеці Django?

Важливі команди:

1. Додати теку де проєкт буде жити

mkdir server

1. Ініціалізувати сам проєкт

django-admin startproject runner server

1. Перейти у теку з проєктом

cd server

1. Зробити новий застосунок де будуть всі файли основні

py manage.py startapp app\_name

1. Коли зробили зміни до моделі реєструємо міграцію

py manage.py makemigrations app\_name

1. Тоді здійснюємо міграцію

py manage.py migrate

1. Коли треба до адмін панелі зайти, створити користувача

py manage.py createsuperuser

Вказати всі необхідні дані:

* імя користувача
* електронна пошта
* пароль

1. Запустити сервер аби побачити всі зміни на сайті

py manage.py runserver

З яких файлів складається проєкт Django?

При створенні містить таку структуру файлів:

project/

default\_app/

\_\_init\_\_.py

asgi.py

settings.py

urls.py

wsgi.py

new\_app/

migrations/

\_\_init\_\_.py

0001\_initial.py

\_\_init\_\_.py

admin.py

apps.py

models.py

tests.py

urls.py

views.py

manage.py

Яку структуру мають додатки Django?

Типова структура:

назва\_проєкту/

назва\_додатку/

назва\_додатку/

manage.py

Для чого необіхдні та яким чином реалізуються моделі Django?

Модель це таблиця даних для бази даних. Реалізується створенням класів моделей у файлі models.py потрібного додатку:

from django.db import models

class BibleBook(models.Model):

canonical\_order\_number=models.PositiveIntegerField()

name=models.CharField(max\_length=77)

number\_of\_chapters=models.PositiveIntegerField()

Яким чином визначаються представлення Django?

У теці додатку має міститися файл *views.py*. До нього можна додати функцію, яка прийматиме аргумент *request* і повертатиме відповідь типу *HttpResponse*. Приклад такої функції нижче:

from django.shortcuts import render

form django.http import HttpResponse

def get\_response(request):

return HttpResponse("**ІСУС ХРИСТОС – ГОСПОДЬ**")

[Джерело](https://www.w3schools.com/django/django_views.php)

Для чого необіхдні та яким чином визначаються і підключаються шаблони?

Для створення шаблону потрібно створити нову теку у теці додатку під назвою *templates*. До неї можна вносити файли шаблонів типу HTML. Сам файл шаблону може містити HTML код, а також спеціальні вирази для відображення динамічного вмісту.

Для підключення шаблону до представлення необхідно модифікувати функцію у файлі *views.py* таким чином:

from django.shortcuts import render

form django.template import loader

def get\_response(request):

template=loader.get\_template("template\_name.html")

return HttpResponse(template.render())

[Джерело](https://www.w3schools.com/django/django_templates.php)

Яким чином можна використати статичні ресури в Django?

Для додавання статичних файлів до проєкту Django потрібно створити теку static у теці додатку. До теки static можна додавати статичні файли, як от зображення чи таблиці стилів.

Після додавання необхідних статичних файлів для їх відображення у шаблоні можна модифікувати його так:

{% load static %}

<!DOCTYPE html>

<html>

<link rel="stylesheet" href="{% static 'file\_name.css' %}">

<body>

**ІСУС ХРИСТОС – ГОСПОДЬ**

</body>

</html>

[Джерело](https://www.w3schools.com/django/django_add_static_files.php)