Міністерство освіти та науки України
Національний університет водного господарства та природокористування
Кафедра комп'ютерних наук та прикладної математики

04-01-62М

МЕТОДИЧНІ ВКАЗІВКИ
до виконання проектної роботи
з навчальної дисципліни

«Сучасні та спеціалізовані мови програмування»

для здобувачів вищої освіти першого (бакалаврського) рівня
за освітньо-професійною програмою
«Прикладна математика»
спеціальності 113 «Прикладна математика»

Рекомендовано науково-методичною
радою з якості ННІ АКОТ
Протокол № 1 від 11.11.2021 р.

Рівне – 2021

Методичні вказівки до виконання проектної роботи з навчальної дисципліни «Сучасні та спеціалізовані мови програмування» для здобувачів вищої освіти першого (бакалаврського) рівня за освітньо-професійною програмою «Прикладна математика» спеціальності 113 «Прикладна математика» денної та заочної форм навчання [Електронне видання] / Грицюк І. М. – Рівне : НУВГП, 2021. – 31 с.

Укладач:
Грицюк І. М., асистент кафедри комп'ютерних наук та прикладної математики.

Відповідальний за випуск:
Турбал Ю. В., д.т.н., професор, завідувач кафедри комп'ютерних наук та прикладної математики

Керівник групи забезпечення спеціальності 113 «Прикладна математика»: Прищепа О. В., к.ф.-м.н., доцент кафедри комп'ютерних наук та прикладної математики.

Зміст

Мета роботи

Ознайомитися з середовищем програмування Python. Навчитися встановлювати, налаштовувати середовище для комфортної та ефективної роботи. Ознайомитися з інтерфейсом та основними функціями. Створити першу, ознайомчу програму.

1. Теоретичні відомості

На сьогоднішній день є дуже популярним розробка web-додатків, а не додатків для персональних комп'ютерів (ПК), адже сьогодні інтернет є в кожного в телефоні та комп'ютері, що спрощує доступ до них та дає можливість спільно користуватись різними додатками без їх встановлення на ПК.

Для спрощення розробки таких додатків використовують різні фреймворки.

Фреймворк – це структура програмних рішень та наборів бібліотек, які полегшують розробку складних система. Також фреймворки часто оголошують правила побудови архітектури проекту.

В екосистемі Python для розробки веб-додатків одним із найбільш популярних фреймворків є Django.

Django - це веб-фреймворк Python, який стимулює швидкий розвиток та чистий, прагматичний дизайн архітектури та коду додатків. Вирішує велику частину задач при веб-розробці. Django безкоштовний та з відкритим кодом.

В якості шаблону проектування Django використовує патерн проектування MVC.

MVC – це патерн проектування, який передбачає поділ системи на три взаємопов'язані частини: модель даних, вигляд (інтерфейс користувача) та модуль керування. Застосовується для відокремлення даних (моделі) від інтерфейсу користувача (вигляду) так, щоб зміни інтерфейсу користувача мінімально впливали на роботу з даними, а зміни в моделі даних могли здійснюватися без змін інтерфейсу користувача.

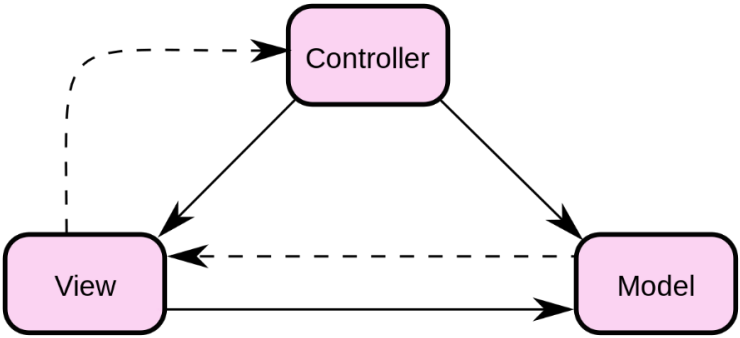

Рис. 1. Схема MVC

1.1. Встановлення Django та створення проекту

Так як Django це фреймворк для мови Python, тому для його роботи необхідно мати встановленим дану мову у вашій системі. У разі якщо Python встановлений тоді Django встановлюється за допомогою пакетного менеджера *pip* командою: *pip3 install Django.*

Після встановлення Django в систему створити проект Django можна за допомогою команди *django-admin startproject <project_name>* знаходячись в будь-якій папці на вашому ПК.

В Django для полегшення розробки проекту є локальний сервер який буде обробляти HTTP запити.

HTTP – протокол передачі даних, що використовується в комп'ютерних мережах. Назва скорочена від *HyperText Transfer Protocol*, протокол передачі гіпертекстових документів

HTTP належить до протоколів моделі OSI 7-го прикладного рівня. Основним призначенням протоколу HTTP є передача веб-сторінок (текстових файлів з розміткою HTML), хоча за допомогою нього успішно передаються як інші файли, які пов'язані з веб-сторінками (зображення та застосунки), так і не пов'язані з ними.

В протоколу HTTP існує кілька *основних методів відправки запитів*:

- *OPTIONS*. Повертає методи HTTP, які підтримуються сервером. Цей метод може служити для визначення можливостей веб-сервера.
- *GET*. Запитує вміст вказаного ресурсу. Запитаний ресурс може приймати параметри (наприклад, пошукова система може приймати як

параметр шуканий рядок). Вони передаються в рядку URI (наприклад: http://www.example.net/resource?param1=value1&param2=value2).

Згідно зі стандартом HTTP, запити типу *GET* вважаються *ідемпотентними* — багаторазове повторення одного і того ж запиту *GET* повинне приводити до однакових результатів (за умови, що сам ресурс не змінився за час між запитами). Це дозволяє кешувати відповіді на запити *GET*. Якщо назва ресурсу не вказана (у URI наявні лише схема та доменне ім'я), то вебсервер повертає індекс директорії вебсервера.

- *HEAD*. Аналогічний методу *GET*, за винятком того, що у відповіді сервера відсутнє тіло. Це корисно для витягання метаінформації, заданої в заголовках відповіді, без пересилання всього вмісту. Зокрема, клієнт чи проксі, перевіривши заголовок Last-Modified: (останній час модифікації), таким чином може переконатися, що сторінка на сервері не змінилася від часу попереднього запиту.
- *POST*. Передає призначені для користувача дані (наприклад, з HTML-форми) заданому ресурсу. Наприклад, в блогах відвідувачі зазвичай можуть вводити свої коментарі до записів в HTML-форму, після чого вони передаються серверу методом *POST*, і він поміщає їх на сторінку. При цьому передані дані (у прикладі з блогами — текст коментаря) включаються в тіло запиту (*Request body*). На відміну від методу *GET*, метод *POST* не вважається ідемпотентним, тобто багаторазове повторення одних і тих же запитів *POST* може повертати різні результати (наприклад, після кожного відправлення коментаря з'являтиметься одна копія цього коментаря).
- *PUT*. Завантажує вказаний ресурс на сервер.
- *PATCH*. Завантажує певну частину ресурсу на сервер.
- *DELETE***.** Видаляє вказаний ресурс.

1.2. Додатки Django

Додаткам в Django відповідають розділи сайту. Таких додатків може бути велика кількість. Кожен з цих додатків має три основних частини:

- пакет в якому реалізовані моделі даних, які взаємодіють з базою та інші структури даних;

- контролери для обробки запитів клієнтів;
- шаблони, які відповідають за відображення даних користувачу.

Для створення нового додатку необхідно виконати команду *python manage.py startapp <app_name>*, де *app_name* – це назва додатку, яка задається розробником.

1.3. Моделі (Models)

Моделі відображають інформацію про дані, з якими ви працюєте. Вони містять поля і поведінку ваших даних. Зазвичай одна модель представляє одну таблицю в базі даних.

Основи:

- Кожна модель – це клас успадкований від *django.db.models.Model*;
- Атрибут моделі представляє поле в базі даних;
- Django надає автоматично створене API для доступу до даних;

Приклад 1: модель, яка визначає гіпотетичного людини (Person), з ім'ям (*first_name*) і прізвищем (*last_name*): *first_name* і *last_name* поля моделі. Кожне поле визначено як атрибут класу, і кожен атрибут відповідає полю таблиці в базі даних.

```
from django.db import models


class Person(models.Model):
    first_name = models.CharField(max_length=30)
    last_name = models.CharField(max_length=30)
```

Приклад 2: Модель Person створить в базі даних таблицю.

```
CREATE TABLE app_name_person
(
    "id"         serial      NOT NULL PRIMARY KEY,
    "first_name" varchar(30) NOT NULL,
    "last_name"  varchar(30) NOT NULL
);
```

Технічні зауваження:

- Назва таблиці, *app_name_person*, автоматично згенерована з метаданих моделі і може бути перевизначена.
- Поле *id* додано автоматично, але його також можна перевизначити.
- *CREATE TABLE SQL* в цьому прикладі відповідає синтаксису MySQL, але варто врахувати, що Django використовує синтаксис SQL відповідно налаштувань бази даних у файлі налаштувань.

1.4. Контролери відображення (view)

Функція клас *відображення* - це функція або клас Python, який приймає веб-запит і повертає веб-відповідь. Ця відповідь може бути вмістом HTML веб-сторінки, або переспрямуванням, або помилкою 404, або XML-документом, або зображенням. Саме відображення містить будь-яку довільну логіку, необхідну для повернення такої відповіді. Відображення розміщуються у файлі *views.py*.

Приклад 3: функції відображення.

```
from django.http import HttpResponse
import datetime


def current_datetime(request):
    now = datetime.datetime.now()
    html = "<html><body>It is now %s.</body></html>" % now
    return HttpResponse(html)
```

Проаналізуємо код рядок за рядком:

Спочатку ми імпортували клас *HttpResponse* з модуля *django.http* і бібліотеку Python *datetime*.

Тепер визначимо функцію *current_datetime*. Це функція уявлення. Кожна функція уявлення приймає об'єкт *HttpRequest* першим аргументом, який зазвичай називають *request*.

Назва функції може бути якою завгодно, немає ніяких конкретних правил для іменування. Ми назвали функцію *current_datetime*.

Подання повертає об'єкт *HttpResponse*, який містить згенеровану відповідь. Кожна функція уявлення повинна повертати об'єкт *HttpResponse*.

1.5. Шаблони (Templates)

Шаблон Django - це текстовий документ або рядок Python, розмічений за допомогою мови шаблонів Django.

Деякі конструкції розпізнаються та інтерпретуються *механізмом шаблонів*. Основні з них – змінні та теги.

Шаблон рендерится з контекстом. Рендерінг замінює змінні значення за їх значеннями, які шукає в контексті та виконує теги. Все останнє виводиться як є.

Синтаксис мовних шаблонів Django використовує *чотири конструкції*:

1. *Змінні* виводять значення з контексту, який є словником. Змінні виділяються *{{* та *}}*, наприклад:

*My first name is **{{** first_name **}}**. My last name is **{{** last_name **}}**.*

2. *Теги* дозволяють додавати виробничу логіку в шаблон. Наприклад, теги можуть вивести текст, додати логічні оператори, такі як «якщо» або «за», отримати вміст із баз даних або запропонувати доступ до інших тем. Теги виділяються *{% и%},* наприклад:

{% csrf_token %}

Велика кількість тегів приймають аргументи приклад:

{% cycle 'odd' 'even' %}

Деякі теги вимагають закриває тег:

{% if user.is_authenticated %}Hello, {{ user.username }}.{% endif %}

3. *Фільтри* перетворюють змінні та аргументи тегів. Фільтри часто виглядають наступним чином *{{ model.title | title }}*

4. *Коментарі* використовуються для коментування коду в шаблону виділяється *{#* та *#}*.

2. Приклад виконання

В якості прикладу використання Django розглянуто розробку простого додатку для керування завданнями.

2.1. Встановлення Django та створення додатку

Для початку потрібно встановити фреймворк Django.

Щоб встановити фреймворк потрібно створити віртуальне оточення. Віртуальне оточення необхідне в Python проектах для того щоб різні версії бібліотек та фреймворків не конфліктували один з одним та для того щоб на “продакшені” та у розробників програмного забезпечення були однакові версії бібліотек та не виникало додаткових помилок по програмному забезпеченні через різні версії бібліотек.

Щоб створити віртуальне оточення необхідно в систему встановити модуль *virtualenv* це можна зробити за допомогою команди в терміналі:

pip install virtualenv

Далі необхідно створити віртуальне оточення виконавши команду:

python3 -m venv myenv та активувати його за допомогою команди для Linux систем там MacOS *source myenv/bin/activate* та *myenv\Scripts\activate.bat* для Windows.

Після створення та віртуального оточення можна приступити до встановлення фреймворку Django. Для того щоб встановити цей фреймворк можна використати пакетний менеджер *pip*, виконавши команду *pip install Django.* Буде встановлено останню стабільну версію фреймворку.

Приклад 4: У разі успішного виконання команди в терміналі буде наступний вивід.

```
Collecting Django
  Downloading Django-3.2.3-py3-none-any.whl (7.9 MB)
     |████████████████████████████████| 7.9 MB 693 kB/s
Collecting sqlparse>=0.2.2
  Using cached sqlparse-0.4.1-py3-none-any.whl (42 kB)
Collecting pytz
  Downloading pytz-2021.1-py2.py3-none-any.whl (510 kB)
     |████████████████████████████████| 510 kB 7.7 MB/s
Collecting asgiref<4,>=3.3.2
  Downloading asgiref-3.3.4-py3-none-any.whl (22 kB)
Installing collected packages: sqlparse, pytz, asgiref, Django
Successfully installed Django-3.2.3 asgiref-3.3.4 pytz-2021.1 sqlparse-0.4.1
```

Далі створимо проект Django з використанням команди *django-admin startproject todos*, де *todos* це назва проекту. Після виконання

даної команди в папці має бути дві папки *myenv* та *todos*. Для перевірки роботи Django запустимо локальний сервер для розробки додатку Django, для цього необхідно виконати команду *python manage.py runserver*. Якщо все вірно встановлено, то в терміналі буде наступний вивід.

Приклад 5: Перед вводом команди потрібно перейти в папку *todos*.

```
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).

You have 18 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): admin, auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.
May 30, 2021 - 13:25:38
Django version 3.2.3, using settings 'todos.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

Перейшовши за адресою http://127.0.0.1:8000/ у браузері відкриється стартова сторінка Django:

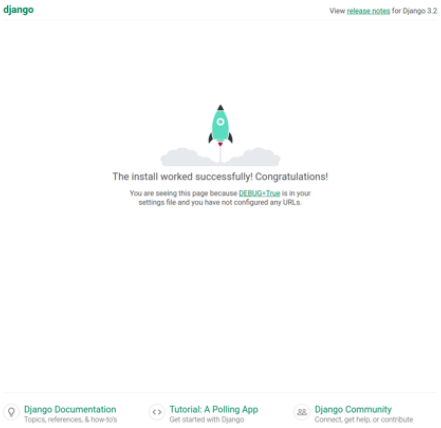

Рис. 2. Стартова сторінка Django

У випадку виникнення помилки під час запуску веб-сервера, потрібно застосувати міграції. Це можна виконати за допомогою команди *python manage.py migrate*, під час застосування буде створено базову структуру бази даних.

Відкриємо наш проект у редакторі та розглянемо його структуру детальніше.

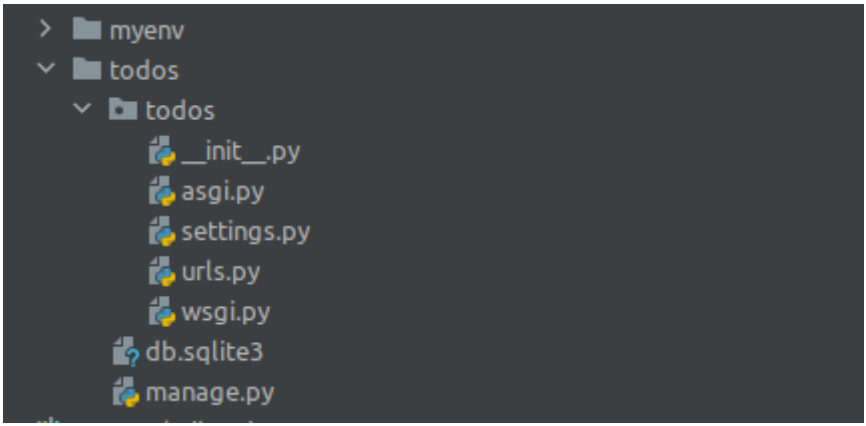

Рис. 3. Структура проекту Django

Папка *myenv* – це папка, в якій знаходяться файли конфігурації, виконувані файли та бібліотеки встановлені у віртуальне оточення.

Папка *todos* – це папка створеного проекту, яка містить в собі папку з такою ж назвою файл *db.sqlite3* та файл *manage.py*.

В папка *todos* містить в собі файл конфігурацій додатку *settings.py*, файли для запуску веб сервера на бойовому сервері *asgi.py* та *wsgi.py*, файл *urls.py* для опису шляхів які будуть на сайті.

Файл *manage.py* - це файл який використовується для запуску різних консольних утиліт Django, а також написаних самостійно.

db.sqlite3 - це файл реляційної бази даних для додатку.

2.2. Конфігурування додатку.

Розглянемо основні блоки файлу конфігурації:

- *DEBUG* – змінна, яка вмикає та вимикає режим відлагодження додатку Django. Значенням після створення є *True*, його рекомендовано використовувати лише при відлагодженні додатку та *False* на сервері.
- *INSTALLED_APPS* – змінна, в якій описані всі підключенні додатки. Код проекту Django розділений на менші додатки, які можна використовувати для логічного розділення коду на Django проекті. Та вмикаючи або вимикаючи їх будуть доступні різні частини сайту розробленого на Django. До прикладу в Django є вмонтована панель

адміністратора, яку можна модернізувати та заточувати під свої потреби вона може бути підключена до додатку в дану секцію конфігураційного файлу.

- *MIDDLEWARE* – список проміжних дій підчас обробки HTTP запиту. Тобто додаючи різні класи можна змінити вхідні дані або дані відповіді від сервера. Використовуючи цей механізм можна створити систему перевірки прав користувачів або перевірку чи авторизований користувач та відповідно відривати або забороняти доступи до різних частин розробленого додатку.
- *DATABASES* - блок конфігурування бази даних. Django підтримує роботу із великою кількістю різних систем керування базами даних. Не змінюючи програмного коду можна змінювати різні системи керування базами даних, необхідно лише змінити налаштування в цьому блоці.
- *LANGUAGE_CODE* та *TIME_ZONE* відповідають за мову, яка буде відображатись на сайті та тайм зону для сайту.

2.3. Додатки в Django та їх структура.

Створимо додаток в проекті Django, який буде реалізовувати логіку роботи із завданнями. Для створення додатку виконаємо команду *python manage.py startapp tasks* . Після виконання даної команди в проекті з'явиться ще одна папка *tasks.*

Розглянемо детальніше структуру згенерованого проекту. Згенерований додаток складається із папки зі міграціями та декількох фалів:

admin.py – файл, в якому описується взаємодія даного додатку із адміністративною панеллю Django.

apps.py - опис додатку для Django.

models.py - у цьому файлі описуються моделі робити.

tests.py - знаходяться тести для коду

views.py - описуються код, який виступає проміжним між моделями, тобто, даними та їх відображеннями в графічному інтерфейсі, контролер архітектури MVC.

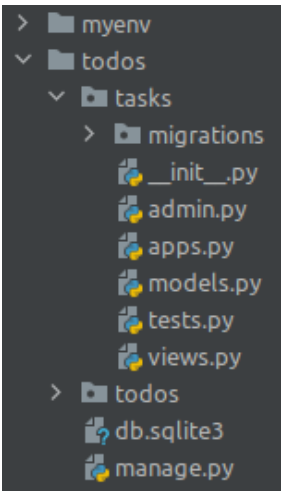

Рис. 4. Структура проекту з урахуванням папки tasks

Весь код в Django проектах поділений на додатки, це дає можливість розділяти на менші та простіші модулі, що спрощує код та в подальшому спрощує роботу із ним. Створимо додаток, який буде організовувати в собі код по роботі із завданнями в додатку для організації завдань. Щоб створити такий додаток необхідно виконати команду в терміналі python *manage.py startapp tasks* у разі успішного виконання команди в консолі не буде ніякого виводу, лише у папці із проектом з'явиться папка під назвою *tasks.*

Згенерований додаток має наступну структуру:

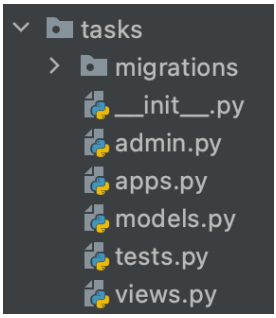

Рис. 5. Структура згенерованого додатку

Розглянемо структуру додатку детальніше.

- *migrations* – папка, в якій лежать міграції. Міграції – це код, який виконується один раз для зміни структури бази даних, створення різних папок файлів та інших дій, які потрібно виконувати один раз при оновленні додатку;
- *admin* – файл, в якому описуються моделі та структура вбудованої в Django адміністративної панелі;
- *apps* – описуються параметри згенерованого додатку;
- *models* – створюються класи, які описують моделі цього додатку.
- *tests* – записуються тести для додатку.
- *views* – створюються обробники HTTP запитів, які поєднують між собою дані (моделі) та їх відображення.

Для того щоб додаток буд під'єднаний до Django проекту необхідно його підключити у файлі конфігурацій:

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'tasks',
]
```

Рис. 6. Приклад під'єднання до Django

2.4. Створення моделей та міграцій

В Django із базою даних можна взаємодіяти використовуючи ORM.

ORM - об'єктно реляційне відображення даних, принцип, який дає можливість поєднати дані із таблиці бази даних до структурами даних мов програмування (у випадку Python та Django – це класи).

Такі об'єкти в термінології MVC та Django звуться *моделями*. Для моделі необхідно створити клас моделі та міграцію.

Всі моделі Django знаходяться у файлі *models.py*. Створимо в цьому класі *Task*. Цей клас буде відповідати завданням, якими буде оперувати користувач, та матиме поля *title* – заголовок, *description* – опис завдання, *is_done* – статус чи виконане завдання.

Приклад 6:

```python
from django.db import models


class Task(models.Model):
    title = models.CharField(max_length=255, verbose_name='Title')
    description = models.TextField(verbose_name='Description')
    is_done = models.BooleanField(verbose_name='Is Done', default=False)
```

Після оголошення класу моделі можна згенерувати міграцію, яка буде створювати таблицю для завдань в базі даних.

Для створення міграції необхідно запустити команду *python manage.py makemigrations* у разі успішного створення міграції вивід команди буде наступного виду.

Приклад 7:

```
Migrations for 'tasks':
  tasks/migrations/0001_initial.py
    - Create model Task
```

Та з’явиться файл із міграцією в папці *migrations*, який буде виглядати наступним чином.

Приклад 8:

```python
from django.db import migrations, models


class Migration(migrations.Migration):
    initial = True

    dependencies = [
    ]

    operations = [
        migrations.CreateModel(
            name='Task',
            fields=[
                ('id', models.BigAutoField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID')),
                ('title', models.CharField(max_length=255)),
                ('description', models.TextField()),
                ('is_done', models.BooleanField()),
            ],
        ),
    ]
```

Далі необхідно застосувати міграцію за допомогою команди *python manage.py migrate*. Ця команда застосує всі міграції, які є в підключених додатках, якщо вони ще не були застосовані.

Приклад 9:

```
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions, tasks
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 sessions.0001_initial... OK
  Applying tasks.0001_initial... OK
```

2.5. Actions та Views.

Після створення моделі завдань створимо форми додавання та редагування завдань.

Для початку створимо клас, який буде обробляти дані з форми. Такі класи прийнято розміщувати у файлі *forms.py*, тому створимо такий файл та опишемо клас.

Приклад 10:

```python
class TaskAddForm(ModelForm):
    class Meta:
        model = Task
        fields = ['title', 'description']
        widgets = {
            'title': TextInput(attrs={
                'placeholder': 'Tasks title',
                'class': 'form-control',
            }),
            'description': Textarea(attrs={
                'placeholder': 'Tasks description',
                'class': 'form-control',
            }),
        }
```

Щоб описати форму створимо клас *TaskAddForm*, а в ньому мета клас. Для того щоб зв'язати модель *Task* із формою потрібно створити атрибут *model* та присвоїти їм клас *Task.* Для того щоб задати поля для форми необхідно оголосити проперті *fields* та передати туди список полів модулів, які повинні бути в цій формі. У проперті *widgets* було оголошено які поля використовувати для тієї чи іншої проперті під час рендеригу форми.

Створимо відображення цієї форми.

У фреймвоці Django, для створення шаблонів відображень використовується шаблонізатор *Django template language* (DTL).

Для початку створимо базовий шаблон від якого будемо наслідувати всі інші шаблони. Створимо файл в папці *templates* із назвою *layout.html* та опишемо в ньому контейнер для всіх сторінок, оголосимо блоки, які будуть перевизначатись в інших шаблонах та підключимо бібліотеку Bootstrap 5. Для оголошення блоків, які будуть перевизначені в іншому шаблоні використовується директива *block* <block name> обгорнута в *{% %}*, а в цьому шаблонізаторі це означає виклик функції або методу класу. Також існує запис *{{ }}* – це означає вивести вміст змінної або результат виконання функції.

Приклад 11:

```
{% load static %}

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>{% block title %}{% endblock %}</title>
    <link href='{% static "css/bootstrap.min.css" %}' rel="stylesheet">
</head>
<body>
<div class="container">
    {% block content %}
    {% endblock %}
</div>

<script src='{% static "js/bootstrap.js" %}'></script>
</body>
</html>
```

За допомогою директиви *load* підключаються різні функції для роботи в шаблонах. Таким чином було підключено функцію *static*, яка

будує маршрути до статичних файлів, таких як стилі та JavaScript скриптів. Для коректної роботи даної функції необхідно в конфіг файлі *settiings.py* налаштувати розміщення статичних файлів. Це можна зробити наступним чином:

Приклад 12:

```
STATIC_URL = '/static/'
STATICFILES_DIRS = (  # додано до стандартного конфігу, та вказує на розміщення папки із статичними файлами
    os.path.join(BASE_DIR, 'static'),
)
```

Також для того щоб додати завантажену бібліотеку необхідно створити папку *static* в папці проекту, та додати туди завантажену бібліотеку.

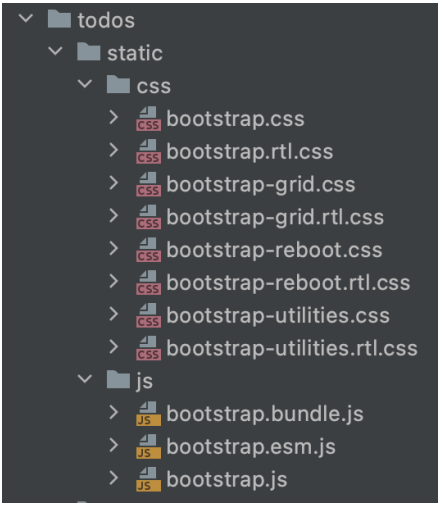

Рис. 7. Проект після додавання папки *static* та потрібних бібліотек

Після створення базового шаблону від нього унаслідувано всі інші шаблони додатку. До прикладу шаблон додавання завдання.

Приклад 13:

```
{% extends 'layout.html' %}

{% block title %}Add tasks{% endblock %}

{% block content %}
    <div class="text-center">
        <h4>Add new task</h4>
    </div>
    {% if form.errors %}
        <div class="alert alert-danger">
            {{ form.errors }}
        </div>
    {% endif %}
    <form method="POST">
        {% csrf_token %}
        {{ form.title }}
        {{ form.description }}

        <button class="btn btn-outline-primary submit-button">Create task</button>
    </form>

    <style>
        .form-control {
            margin-top: 10px;
        }

        .submit-button {
            margin-top: 10px;
            width: 100%;
        }
    </style>
{% endblock %}
```

Розберемо синтаксис даного шаблону детальніше.

Директива *extends* використовується для оголошення базового шаблону, від якого унаслідувався даний. Для перевизначення вмісту блоків із базового шаблону використовується така ж директива як і для їх оголошення, блок та його назва. Директива *csrf_token* – це директива, яка генерує приховане поле із токеном, який задається та обробляється сервером, цей токен використовується для запобіганню CSRF атак. Змінна *form* та її атрибути *errors, title ma description* – це віджети та атрибути класу *TaskAddForm*, екземпляр якого передається у відображення за допомогою контролера шаблону проектування MVC, який в термінології Django зветься *view*.

В Django є два способи опису контролерів шаблону MVC, функціональний та об’єктно орієнтований. Розглянемо детальніше об’єктно орієнтований спосіб. Контролер описаний об’єктно орієнтованим способом представляє із себе клас, який має декілька основних методів: *GET, POST, PUT, DELETE, OPTION, HEAD*, які повторюють назви методів HTTP протоколу. На рисунку зображено

приклад реалізації класу контролера, який рендерить відображення форми додавання завдання та обробляє цю форму.

Приклад 14:

```python
class AddTask(TemplateView):
    template_name = "add.html"

    def get_context_data(self, *args, **kwargs):
        context = super(AddTask, self).get_context_data(**kwargs)
        context['form'] = TaskAddForm()
        return context

    def get(self, request, *args, **kwargs):
        return render(request, self.get_template_names(), self.get_context_data())

    def post(self, request):
        form = TaskAddForm(request.POST)

        if not form.is_valid() or not form.save():
            context_data = self.get_context_data()
            context_data['form'] = form
            return render(request, self.get_template_names(), context_data)

        return redirect('/tasks')
```

Для відкриття сторінки браузер використовує метод *get* для отримання даних від серверу. Для відправки форм найчастіше використовується метод *post*. Тому, було оголошено методи для обробки таких запитів. Метод *get* відрендерить теплейт, який вказаний у атрибуті класу *template_name* та передасть в нього параметри, які поверне метод *get_context_data*, з допомогою функції *render*, яка підключена із пакету *django.shortcuts*. Функція *render* приймає 3 аргументи за об’єкт запиту, назву шаблона та змінні, які мають бути використані в шаблоні. У методі *post* відбувається обробка запиту від форми. В ній створюється об’єкт модулі форми та в конструктор передаються значення, які були передані із форми в тілі *POST* запиту. Далі значення валідуються та зберігаються в базу даних, у випадку, якщо якісь із цих етапів повернув *False*, то користувачу буде повернуто сторінку із формою та помилками, які він допустив у формі, а у випадку успішного завершення всіх операцій користувача буде перенаправлено на сторінку із списком завдань.

Також *view* необхідно підключити у файл *url* для того, щоб зв'язати його із маршрутом.

Приклад 15:

```
urlpatterns = [
    path('add/', AddTask.as_view(), name='add'),
]
```

Відповідно маршрут до цього контролера буде виглядати наступним чином: якщо перейти за цим шляхом відкриється форма додавання завдання.

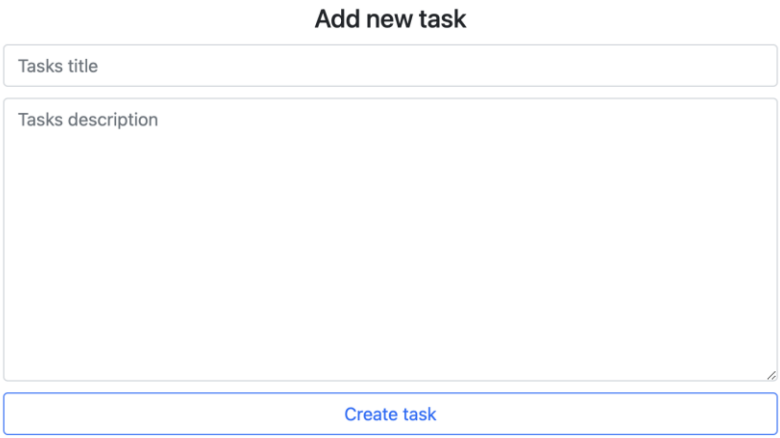

Рис. 8. Вигляд форми додавання завдань

Аналогічно, можна створити форму видалення та форму до редагування. Шаблон форми буде дуже подібним, а от класи цих форм та обробники запитів будуть відрізнятися, тому розглянемо їх детальніше.

У форму редагування було додано поле *is_done*, яке є булевим, тому в якості поля вводу у форму було використано *checkbox*.

Приклад 16:

```python
class TaskEditForm(ModelForm):
    class Meta:
        model = Task
        fields = ['title', 'description', 'is_done']
        widgets = {
            'title': TextInput(attrs={
                'placeholder': 'Tasks title',
                'class': 'form-control',
            }),
            'description': Textarea(attrs={
                'placeholder': 'Tasks description',
                'class': 'form-control',
            }),
            'is_done': CheckboxInput(),
        }
```

В обробник запиту було додано обробку *id* завдання, за допомогою якого знаходиться запис в базі даних. Передаючи у конструктор класу форми параметра *instance* можна передати існуючий об’єкт і відповідні поля із форми будуть заповнені даними із знайденого в базі даних об’єкту.

Приклад 17:

```python
class EditTask(TemplateView):
    template_name = "edit.html"

    def get(self, request, *args, **kwargs):
        context_data = self.get_context_data()
        task = self.__get_task_instance(self.kwargs['task_id'])
        context_data['form'] = TaskEditForm(instance=task)
        context_data['task'] = task
        return render(request, self.get_template_names(), context_data)

    def post(self, request, task_id):
        task = self.__get_task_instance(task_id)
        form = TaskEditForm(request.POST, instance=task)

        if not form.is_valid() or not form.save():
            context_data = self.get_context_data()
            context_data['task'] = task
            context_data['form'] = form
            return render(request, self.get_template_names(), context_data)

        return redirect('/tasks')

    def __get_task_instance(self, task_id):
        return Task.objects.get(pk=task_id)
```

Зареєструвавши шлях наступного виду *path('edit/<int:task_id>', EditTask.as_view(), name='edit')* можна буде отримати доступ до форми, яка матиме наступний вигляд.

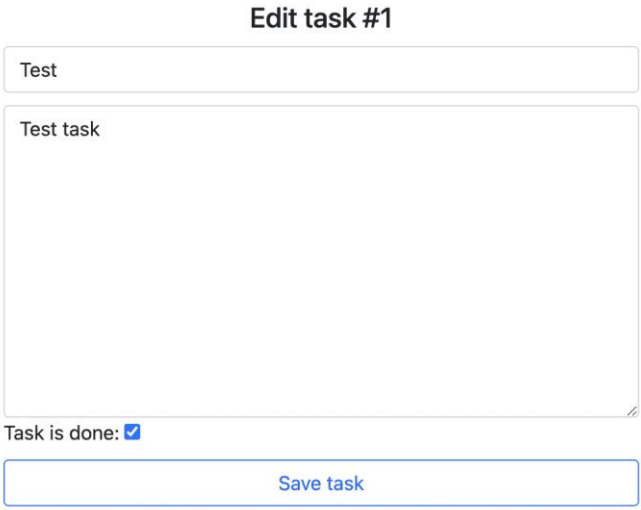

Рис. 9. Вигляд форми редагування завдань

Також Django має набір вже готових *views*, тобто контролерів, для обробки форм для реалізації типових дій із моделями таких як: додавання, редагування записів. Одним із тиках контролерів є *DeleteView*, використовуючи його створимо форуму видалення завдання, для цього унаслідуємось від цього класу та задамо значення атрибутів де model це клас моделі з якою буде працювати контролер, *template_name* шлях до шаблону, *success_url*

Приклад 18:

```
class DeleteTask(DeleteView):
    model = Task
    template_name = "delete.html"
    success_url = '/tasks'
```

та шаблон, він матиме наступний вигляд:

Приклад 19:

```
{% extends 'layout.html' %}

{% block title %}Delete task {{ object.id }}{% endblock %}

{% block content %}
    <div class="delete-alert alert alert-warning">
        <form method="post">{% csrf_token %}
            <p>Are you sure you want to delete "{{ object.title }} ({{ object.id }})"?</p>
            <input type="submit" value="Confirm" class="btn btn-danger delete-button">
        </form>
    </div>

    <style>
        .delete-alert {
            margin-top: 10px;
        }

        .delete-button {
            width: 100%;
        }
    </style>
{% endblock %}
```

Для роботи цієї *view* необхідно зареєструвати шлях наступного виду *<pk>/delete/*, де *pk* – це *id* запису. Перейшовши за цим шляхом відкриється розроблена форма, яка буде мати вигляд.

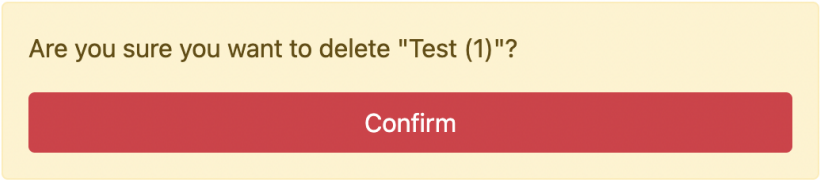

Рис. 10. Вигляд форми видалення завдання

Також створимо список завдань на якому будуть відображатись всі завдання для цього потрібно створити *view*, шаблон та зареєструвати шлях. Для відображення завдань потрібно створити шаблон, в який

буде передано список завдань та з допомогою циклу *for* він буде відображений.

Приклад 20:

```
{% block title %}Task list{% endblock %}

{% block content %}
    <h1 class="text-center">Tasks list</h1>
    <table class="table table-bordered table-striped">
        <thead>
        <tr>
            <td>Title</td>
            <td>Description</td>
            <td>Is done</td>
            <td>Actions</td>
        </tr>
        </thead>
        {% for item in data %}
            <tr>
                <td><a href="/tasks/edit/{{ item.id }}">{{ item.title }}</a></td>
                <td>{{ item.description }}</td>
                <td>{{ item.is_done }}</td>
                <td>
                    <a href="/tasks/edit/{{ item.id }}">Edit</a>
                    <a href="/tasks/{{ item.id }}/delete">Delete</a>
                </td>
            </tr>
        {% endfor %}
    </table>

    <a href="/tasks/add" class="btn btn-primary add-button">Add new task</a>

    <style>
        .table {
            width: 100%;
        }

        .add-button {
            width: 100%;
        }
    </style>
{% endblock %}
```

Список завдань був переданий із контролера у змінну *data*.

Приклад 21:

```python
class TasksList(TemplateView):
    template_name = "index.html"

    def get_context_data(self, *args, **kwargs):
        context = super(TasksList, self).get_context_data(**kwargs)
        context['data'] = Task.objects.all()
        return context

    def get(self, request, *args, **kwargs):
        return render(request, self.get_template_names(), self.get_context_data())
```

В результаті виконання даного коду отримаємо (див. рис. 11).

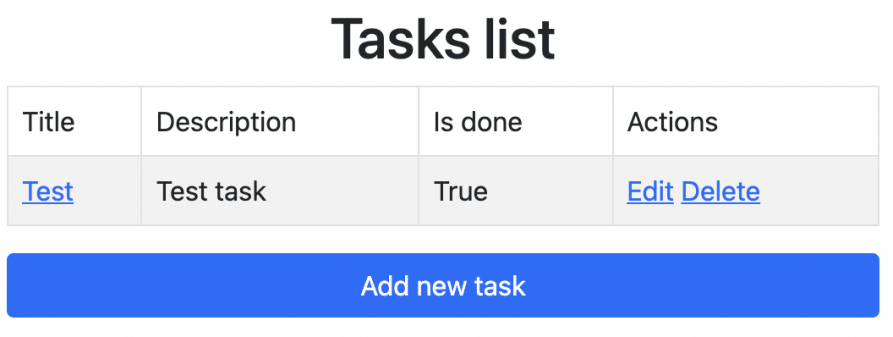

Рис. 11. Вигляд форми завдань

3. Порядок виконання роботи

1. Ознайомитись з теоретичними відомостями та програмою роботи.
2. Ознайомитись із правилами розробки програмного забезпечення.
3. Ознайомитись з мовою програмування Python 3 та Django.
4. Встановити Python 3 на робочий ПК.
5. Встановити Django та створити проект відповідно до теоретичних відомостей.
6. Розробити додаток відповідно до завдання із свого варіанту. Варіант обирається за останньою цифрою в номері залікової книжки.
7. Оформити звіт.

4. Варіанти завдань

Варіант 0. Створити мікроблог, в якому будуть базові функції: створення статті, її збереження, редагування, видалення, та перегляд користувачами. Кожна стаття в блозі має містити заголовок, текст, дату публікації та дату останньої зміни.

Варіант 1. Створити додаток, який буде реалізовувати сайт новин із можливістю публікувати, редагувати переглядати, та видаляти новини. Кожна новина має містити заголовок, текст, дату публікації та дату останньої зміни.

Варіант 2. Написати сайт курсу валют, в якого буде можливість додавати нові курси співвідношення валюти до гривні. Для кожної валюти має бути два значення курсу купівлі та курсу продажу. В результаті отримуємо таблицю із валютами, яка буде мати 3 колонки: назву, курс купівлі та курс продажу. Заповнятись дані в таблицю мають із окремої таблиці, також курс кожної із валюти повинен мати можливість редагувати, видаляти та змінювати кожне значення.

Варіант 3. Створити додаток, який допоможе керувати персоналом. Додаток повинен виводити таблицю, де присутні наступні дані: ПІБ, дата народження, робочий відділ, посаду та дату з якої найманий працівник був прийнятий на посаду.

Варіант 4. Створити програму обліку товару на складі. В програмі має бути можливість додавати нові, редагувати додані, видаляти та переглядати товари. Також має бути можливість змінювати кількість одиниць товару.

Варіант 5. Написати систему для збереження даних про клієнтів компанії, потрібно мати можливість додавання, редагування, перегляду та видалення клієнтів. Необхідно зберігати наступні поля: ПІБ, номер телефону та нотатку.

Варіант 6. Написати програму, яка буде зберігати список студентів. Зберігатись мають: ПІБ, група, та оцінки з декількох предметів за вибором. Також додаток повинен мати функції додавати нові, редагувати додані, видаляти та переглядати інформацію про студента.

Варіант 7. Написати додаток, який буде працювати із транзакціями. Додаток повинен мати можливість додавання, редагування, перегляду та транзакцій. Транзакція повинна мати 3 поля: опис, суму та тип дебіт або кредіт.

Варіант 8. Написати додаток для обліку пального у транспортній компанії. Розроблений додаток повинен реалізовувати можливість додавати поставки пального до сховища та можливістю видачі пального автомобілям. Також повинна бути можливість редагування та видалення кожної поставки та видачі пального.

Варіант 9. Написати додаток для консьєржа. Розроблений додаток повинен мати можливість введення людей, які зайшли у приміщення та вийшли з нього. Також додаток повинен мати можливість редагування всіх записів.

Контрольні питання

1. Що таке фреймворк?
2. Що таке Django?
3. Який патерн проектування використовується в Django?
4. Охарактеризуйте фреймворк Django та його патерн проектування.
5. Що таке патерн MVC, описати його принципи.
6. Що таке HTTP та які є основні методи відправки запитів?
7. Що таке додатки Django?
8. Що таке моделі?
9. Що таке функція клас відображення?
10. Що таке шаблон Django?
11. Охарактеризуйте конструкції мовних шаблонів Django.
12. Основні блоки файлу конфігурації.

Список літературних джерел

1. Welcome to Python : веб сайт. URL: https://www.python.org/
2. Документація Python : веб сайт. URL: https://docs.python.org/2/
3. The Python Tutorial : веб сайт. URL: https://docs.python.org/3/tutorial/
4. Т. Гедис. Починаємо програмувати на Python: підручник. БХВ-Петербург, 4-е видання, 2019. 768 с. URL: https://fileskachat.com/getfile/69426_d9e0b302eb1cdcc520c14037fef1682b
5. М. Лутц. Вивчаємо Python, том 1: підручник. Діалектика, 5-е вид, 2020. 832 с. URL: https://balka-book.com/files/2019/09_02/10_41/u_files_store_3_1985456.zip
6. Е. Метиз. Пришвидшений курс Python: підручник. Видавництво Старого Лева, 2021. 600 с.
7. О. Швець. Занурення в патерни проектування : підручник. 2021. 393 с. URL: https://refactoring.guru/files/design-patterns-uk-demo.pdf
8. Присяжнюк, О. В. Методичні вказівки до виконання лабораторних робіт з навчальної дисципліни «Програмування. Частина 2. Програмування мовою Python» для здобувачів вищої освіти першого (бакалаврського) рівня за освітньо-професійною програмою «Автоматизація та комп'ютерно-інтегровані технології» спеціальностей 151 «Автоматизація та комп'ютерноінтегровані технології», 141 «Електроенергетика, електротехніка та електромеханіка» денної та заочної форм навчання: методичне забезпечення. Рівне : НУВГП. 2020, 165 с. URL: http://ep3.nuwm.edu.ua/17989/