8. РОЗРОБКА WEB-ДОДАТКІВ.

Під веб-застосуванням (прикладною інтернет-програмою) розуміють програму, основний інтерфейс користувача якої працює в стандартному веб-браузері під керуванням html та XML документів. Для поліпшення якості інтерфейсу користувача часто використовують JavaScript, однак це дещо знижує універсальність інтерфейсу. Зазначимо, що інтерфейс можна побудувати на Java- або Flash-аплетах, однак, такі програми складно назвати веб-програмами, тому що Java або Flash можуть використовувати власні протоколи для спілкування із сервером, а не стандартний для www протокол HTTP.

При створенні веб-програм намагаються відокремити форму (зовнішній вигляд, стиль), зміст та логіку обробки даних. Сучасні технології побудови веб-сайтів дають можливість підійти досить близько до цього ідеалу. Проте, навіть не використовуючи багаторівневі програми, можна дотримуватися стилю, що дозволяє змінювати кожний із цих аспектів, не зачіпаючи (або майже не зачіпаючи) два інші.

8.1. CGI додатки

Додаток CGI (Common Gateway Interface, загальний шлюзовий інтерфейс) не на багато відрізняється від звичайної програми. Основні відмінності полягають в тому, що в цьому інтерфейсі засоби введення-виведення даних і взаємодія з користувачем реалізовані інакше, ніж в програмі, не призначеної для роботи в мережі. Після запуску сценарію CGI веб-сервер отримує дані, введені користувачем, але вони приходять від веб-клієнта, а не з локального комп'ютера або з файлу на диску. Цей процес називається запитом (request).

На відміну від звичайного програмного додатку, при виведенні даних CGI-додаток передає їх веб-клієнту, а не у вікно графічного інтерфейсу програми. Дані, що передаються сервером веб-клієнту, називаються відповіддю (response).

Нарешті, слід зазначити, що безпосередньо в самому сценарії взаємодія з користувачем не передбачено. Весь обмін даними відбувається між веб-клієнтом, що діє від імені користувача та веб-сервером і додатком CGI.

Класичний шлях створення програм для веб-додатків – написання CGI-сценаріїв (іноді кажуть – скриптів). CGI – це стандарт, що регламентує взаємодію сервера із зовнішніми програмами. У випадку з www веб-сервер може направити запит на генерацію сторінки визначеному сценарію. Цей сценарій, отримавши на вхід дані від веб-сервера (той, у свою чергу, міг отримати їх від користувача), генерує готовий об'єкт (зображення, аудіодані, таблицю стилів тощо).

При виклику сценарію веб-сервер передає йому інформацію через стандартний потік введення, змінні оточення і, для *ISINDEX*, через аргументи командного рядка

(вони доступні через *sys.argv*).

Два основні методи передавання даних із заповненої у браузері форми веб-сервера (і CGI-сценарію) – GET та POST. Залежно від методу дані передаються по-різному. У першому випадку вони кодуються та містяться прямо в URL, наприклад: http://host/cgi-bin/a.cgi?*a=1&b=3*. Сценарій отримує їх у змінній оточення з іменем *QUERY_STRING*. У випадку методу POST вони передаються на стандартний потік уведення. Для коректної роботи сценарії розміщують у призначеному для цього каталозі на веб-сервері (зазвичай він називається *cgi-bin*) або, якщо це дозволено конфігурацією сервера, у будь-якому місці серед документів html. Сценарій повинен бути виконуваним (у файловій системі). У системі Unix його можна встановити за допомогою команди chmod a+x.

Наступний найпростіший сценарій виводить значення зі словника *os.environ* та дозволяє побачити, що йому було передано:

```
#!/usr/bin/python
import os
print ("""Content-Type: text/plain
%s""" % os.environ)
```

За його допомогою можна побачити встановлені веб-сервером змінні оточення. CGI-сценарій, що видає веб-серверу файл, має заголовок, у якому містяться поля з метаінформацією (тип вмісту, час останнього відновлення документа, кодування тощо).

Основні змінні оточення:

QUERY_STRING – рядок запиту.

REMOTE_ADDR – IP-адреса клієнта.

REMOTE_USER – ім'я клієнта (якщо він був ідентифікований).

SCRIPT_NAME – ім'я сценарію.

SCRIPT_FILENAME – ім'я файла зі сценарієм.

SERVER_NAME – ім'я сервера.

HTTP_USER_AGENT – назва браузера клієнта.

REQUEST_URI – рядок запиту (URI).

HTTP_ACCEPT_LANGUAGE – бажана мова документа.

8.1.1. Модуль cgi

У Python є підтримка CGI у вигляді модуля *cgi*. В модулі *cgi* передбачено основний клас *FieldStorage*, який виконує дану роботу. Цей клас зчитує всю необхідну інформацію про користувача, передану веб-клієнтом (через веб-сервер), тому копія цього класу повинна бути створена відразу після запуску CGI-сценарію у

мові Python. Екземпляр вказаного класу включає об'єкт, що нагадує словник, який містить набір пар "ключ-значення". Ключовими є імена елементів вводу, переданих з заповненої форми. Значення містять відповідні дані.

Значення можуть представляти собою об'єкти одного з трьох типів. По перше, екземпляри класу *FieldStorage*. По-друге, екземпляри аналогічного класу *MiniFieldStorage*, який використовується в тих випадках, якщо не потрібна передача (upload) файлів або ведеться обробка даних форми, яка не складається з кількох частин. Екземпляри *MiniFieldStorage* містять тільки пари "ключ-значення", що складаються з імені та даних. По-третє, передані значення можуть являти собою список зазначених об'єктів. Такі списки формуються, якщо форма містить кілька елементів введення з однаковим ім'ям поля.

Для простих веб-форм зазвичай достатньо застосовувати лише екземпляри *MiniFieldStorage*.

8.1.2. Модуль cgitb

Для відладки CGI-сценарію можна використовувати модуль *cgit*b. При виникненні помилки цей модуль видасть html-сторінку, де вкаже місце виникнення виняткової ситуації. На початку налагоджуваного сценарію потрібно включити наступні рядки:

```
import cgitb
cgitb.enable()
```

Або, якщо не потрібно показувати помилки у браузері:

```
import cgitb
cgitb.enable(0, logdir="/tmp")
```

Тільки слід пам'ятати, що варто видалити ці рядки, коли сценарій буде налагоджено, тому що він видає частини коду сценарію. Цим можуть скористатися зловмисники, для того, щоб знайти вразливості у CGI-сценарії або підглянути паролі (якщо такі є у сценарії).

Наступний приклад демонструє деякі з можливостей даних модулів:

1) В поточному каталозі створіть Python-скрипт start.py, який буде запускати сервер на 8000 порту:

```
from http.server import HTTPServer, CGIHTTPRequestHandler
server_address = ("", 8000)
httpd = HTTPServer(server_address, CGIHTTPRequestHandler)
httpd.serve_forever()
```

2) Створіть в поточному каталозі HTML-форму під назвою forms.html, яка буде приймати дані від клієнта і посилати їх на обробку:

```
<!DOCTYPE html>
<html>
<head>
  <title>A simple form demonstration</title>
</head>
<body>
  <div style="text-align:center;">
    <h1>User login</h1>
    <form action="/cgi-bin/retrieval.py" method="get">
      username : <input type="text" name="username" style="text-
    align:center;">
      <br><br>
      password : <input type="password" name="password" style="text-
    align:center;">
      <br><br><br>
      <input type="submit" value="Submit">
    </form>
  </div>
</body>
  </html>
```

3) В поточному каталозі створіть каталог cgi-bin в якому створіть Python-скрипт retrieval.py для обробки даних, що приходять з веб-сторінки:

```
#!/usr/bin/env python3

import cgi, cgitb
cgitb.enable()          ## дозволяє виправляти помилки з сценаріїв cgi в
браузері
form = cgi.FieldStorage()

## зчитування даних з відповідних полів
first = form.getvalue('username')
last = form.getvalue('password')
```

```
print("Content-type:text/html\r\n\r\n")
print("<html>")
print("<head><title>User entered</title></head>")
print("<body>")
print("<h1>User has entered</h1>")
print("<b>Firstname : </b>" + first + "<br>")
print("<br><b>Lastname : </b>" + last + "<br>")
print("")
print("</div>")
print("</body>")
print("</html>")
```

4) Зайдіть в браузері за посиланням http://localhost:8000/forms.html , заповніть форму та підтвердіть введення

8.2. Стандарт WSGI

CGI-технологія не має перспектив подальшого розвитку, оскільки не забезпечує масштабування; процеси CGI створюються для обробки кожного окремого запиту, а потім знищуються (це можна порівняти з тим, як виконуються сценарії інтерпретатором Python). Якщо в веб-додаток будуть приходити тисячі запитів і у відповідь на це буде запускатися така ж кількість інтерпретаторів, то буде створене така навантаження, з яким не впорається жоден сервер. У зв'язку з цим отримав широке поширення наступні WSGI.

WSGI (Web Server Gateway Interface) — стандарт взаємодії між Python-програмою, яка виконується на стороні сервера, і самим веб-сервером, наприклад, Apache.

В Python існує велика кількість різного роду веб-фреймворків, інструментаріїв і бібліотек. У кожного з них власний метод встановлення та налаштування, вони часто написані так, що не можуть взаємодіяти між собою. Це може стати проблемою, оскільки вибір фреймворку може обмежити вибір веб-сервера і навпаки.

WSGI надає простий і універсальний інтерфейс для взаємодії між більшістю веб-серверів і веб-додатками чи фреймворками.

По стандарту WSGI, веб-застосунок має задовольняти наступним вимогам:

- має бути викличним (callable) об'єктом;
- приймати два параметри:
  - словник змінних оточення (environ);
  - обробник запиту(start_response)
- викликати обробник запиту з кодом HTTP-відповіді та HTTP-заголовками;

- повертати ітератор з тілом відповіді.

Простим прикладом WSGI-застосунку може служити така функція:

```
def simple_wsgi_app (environ, start_response) :
   status = ' 200 OK '
   headers = [( 'Content-type' , 'text/plain')]
   start_response ( status, headers)
   return [ 'Hello world !' ]
```

Додаток WSGI визначається як викликаний код, який завжди приймає такі параметри: словник змінних оточення сервера і ще один викликаний фрагмент коду, який ініціює підготовку відповіді з кодом статусу HTTP і заголовками HTTP для повернення клієнту. Цей код повинен повертати ітеруємий об'єкт, який становить корисні дані.

У наведеному вище додатку "Hello World" на основі WSGI ці змінні іменуються відповідно *environ* і *start_response()*

Змінна *environ* включає в себе змінні оточення, такі як *HTTP_HOS*T, *HTTP_USER_AGENT*, *SERVER_PROTOCOL* і т.д. Функція *start_response()*, яка повинна бути виконана в додатку, готує відповідь, що відправляється в кінцевому підсумку назад клієнту. Відповідь має включати код повернення HTTP (200, 300 і т.д.), а також відповіді HTTP- заголовки.

8.3. Введеня у фреймворк Django

Django – високорівневий відкритий Python-фреймворк для розробки веб-систем.

Сайт на Django будується з однієї або декількох частин, які рекомендується робити модульними..

Архітектура Django подібна на «Модель-Вид-Контролер» (MVC). Однак, те що називається «контролером» в класичній моделі MVC, в Django називається «вид», а те, що мало б бути «видом», називається «шаблон». Таким чином, MVC розробники Django називають MTV («Модель-Шаблон-Вид»).

Початкова розробка Django, як засобу для роботи новинних ресурсів, досить сильно позначилася на його архітектурі: він надає ряд засобів, які допомагають у швидкій розробці веб-сайтів інформаційного характеру. Так, наприклад, розробнику не потрібно створювати контролери та сторінки для адміністративної частини сайту, в Django є вбудований модуль для керування вмістом, який можна включити в будь-який сайт, зроблений на Django, і який може керувати відразу декількома сайтами на одному сервері. Адміністративний модуль дозволяє створювати, змінювати і вилучати будь-які об'єкти наповнення сайту, протоколюючі всі дії, а також надає інтерфейс для управління користувачами і групами (з призначенням прав).

У дистрибутив Django також включені програми для системи коментарів, синдикації RSS і Atom, «статичних сторінок» (якими можна управляти без необхідності писати контролери та відображення), перенаправлення URL і т.д.

До основних можливостей фреймворку Django слід віднести:

1) Об'єктно-реляційне відображення (ORM)

Django підтримує парадигму ООП. Об'єкти БД в термінології Django іменуються «моделями». Фреймворк надає у розпорядження розробникові розвинутий прикладний програмний інтерфейс для високорівневого доступу до даних. В більшості випадків немає потреби писати SQL-запити.

Для прикладу, для проекту обліку учнів можна створити таку модель:

```
class Student(models.Model):
    name = models.CharField("Ім'я", max_length="100")
    surname = models.CharField("Прізвище", max_length="100")
    birth_date = models.DateField()
```

При виконанні синхронізації проекту з БД автоматично буде створена таблиця БД з полями, які відповідають полям (*properties*) моделі.

Вибірка всіх студентів:

```
students = Student.objects.all()
```

Вибірка з фільтром по прізвищу, по частині прізвища, по даті народження:

```
students = Student.objects.filter(surname="Іванов")
students = Student.objects.filter(surname__iexact="нов") # LIKE-фільтр
students = Student.objects.filter(birth_date__gte=datetime.date('1982', '4', '5')) #старші за дану дату
```

2) Автоматична побудова інтерфейсу для адміністрування

Django автоматично створює CRUD (create read update delete – 4 базові функції управління даними: створення, зчитування, зміна і видалення)-інтерфейс ('адмінку').

3) Елегантні URL

Парсинг URL-адрес побудований на регулярних виразах. Розробник не обмежений у використанні певної схеми посилань.

4) Зручна система шаблонів

В Django є окрема мова для опису шаблонів. Вона є дуже простою для початківців. В ній присутні оператори циклу, умови, форматування даних.

5) Гнучка підсистема кешування даних

Django-проект може бути налаштований на роботу з Memcached чи будь-яким іншим фреймворком зберігання даних в оперативній пам’яті. Інструменти Django дозволяють кешувати SQL-вибірки, шаблони та їх частини і просто окремі змінні.

Детальну інформацію з використання фреймворка Django можна знайти за посиланням https://www.djangoproject.com/ .