Перші кроки в програмуванні веб-сайтів з боку сервера

У цій темі, присвяченій програмуванню з боку сервера, ми відповімо на кілька фундаментальних питань про програмування серверної частини:

- *що це таке?*
- *чим воно відрізняється від програмування клієнтської частини?*
- *чому воно так корисно?*

Потім проведемо огляд деяких найпопулярніших веб-фреймворків для серверної частини і керівництво по вибору найбільш підходящого фреймворка для створення вашого першого сайту. Нарешті, завершимо тему розглядом безпеки веб-сервера.

2. Клиент-сервер

2.1. Web-сервери і HTTP

Веб-браузери взаємодіють з веб-серверами за допомогою **протоколу передачі гіпертексту** (HTTP). Коли ви натискаєте на посилання на сторінці, заповнюєте форму або запускаєте пошук, браузер відправляє на сервер HTTP-запит.

Цей запит включає або може включати:

- Шлях, який визначає цільової сервер і ресурс (наприклад, файл, певна точка даних на сервері, що запускається сервіс, і т.д.).
- Метод, який визначає необхідну дію (наприклад, отримати файл, зберегти або відновити деякі дані). Різні методи / команди і пов'язані з ними дії:
  - GET - отримати певний ресурс (наприклад, HTML-файл, що містить інформацію про товар або список товарів).
  - POST - створити новий ресурс (наприклад, нову статтю на вікі, додати новий контакт в базу даних).
  - HEAD - отримати метадані про певний ресурсі без отримання змісту, як робить GET. Ви, наприклад, можете використовувати запит HEAD, щоб дізнатися, коли в останній раз ресурс оновлювався і тільки потім використовувати (більш «витратний») запит GET, щоб завантажити ресурс, який був змінений.
  - PUT - оновити існуючий ресурс (або створити новий, якщо такої не існує).
  - DELETE - видалити вказаний ресурс.
  - TRACE, OPTIONS, CONNECT, PATCH - ці команди використовуються для менш популярних / просунутих завдань, і ми не будемо їх розглядати.
- Додаткова інформація може бути закодована в запиті (наприклад, дані форми). Інформація може бути закодована як:

- *Параметри URL*: GET запити зашифровують дані в URL-адресу, що відправляється на сервер, додаючи пари ім'я / значення в його кінець, наприклад, http://mysite.com?name=Fred&age=11. Завжди є знак питання (?), що відокремлює іншу частину URL-адреси від її параметрів, знак рівності (=), що відокремлює кожне ім'я від відповідного йому значення, і амперсанд (&), що розділяє пари. URL-параметри за своєю суттю «небезпечні», бо вони можуть бути змінені користувачами і потім відправлені заново. В результаті параметр / GET запиту не використовуються для запитів, які оновлюють дані на сервері (як ми це розглядали раніше).
- *POST дані*: POST запити додають нові ресурси, дані яких зашифровані в тілі запиту.
- *Кукі-файли клієнтської частини*: Кукі-файли містять дані сесій про клієнта, включаючи ключові слова, які сервер може використовувати для визначення його авторизаційного статусу і права доступу до ресурсів.

Веб-сервери очікують повідомлень із запитами від клієнтів, обробляють їх, коли вони надходять і відповідають веб-браузеру через повідомлення з HTTP-відповіддю. Відповідь містить Код статусу HTTP-відповіді, який показує, чи був запит успішним (наприклад, «200 OK» означає успіх, «404 Not Found» якщо ресурс не може бути знайдений, «403 Forbidden», якщо користувач не має права переглядати ресурс, і т.д.). Тіло успішної відповіді на запит GET буде містити запитуваний ресурс.

Після того, як HTML сторінка була повернута, вона обробляється браузером. Далі браузер може досліджувати посилання на інші ресурси (наприклад, HTML сторінка зазвичай використовує JavaScript і CSS файли), та надіслати окремий HTTP запит на завантаження цих файлів.

Як статичні, так і динамічні веб-сайти використовують такий самий протокол / шаблон зв'язку.

2.1.1. Приклад запиту / відповіді GET

Ви можете сформувати простий GET запит клікнувши на посилання або через пошук по сайту (наприклад, сторінка механізму пошуку). Але пам’ятайте, що частини повідомлення залежать від Вашого браузера \ налаштувань.

Запит.

Кожен рядок запиту містить інформацію про запит. Перша частина (заголовок) містить важливу інформацію про запит, точно так само як HTML head містить важливу інформацію про HTML документі (але не вміст документа, що розташований в body):

```
1. GET https://developer.mozilla.org/en-
   US/search?q=client+server+overview&topic=apps&topic=html&topic=css&topic=js&topic
   =api&topic=webdev HTTP/1.1
2. Host: developer.mozilla.org
3. Connection: keep-alive
4. Pragma: no-cache
5. Cache-Control: no-cache
6. Upgrade-Insecure-Requests: 1
7. User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like
   Gecko) Chrome/52.0.2743.116 Safari/537.36
8. Accept:
   text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
9. Referer: https://developer.mozilla.org/en-US/
10. Accept-Encoding: gzip, deflate, sdch, br
11. Accept-Language: en-US,en;q=0.8,es;q=0.6
12. Cookie: sessionid=6ynxs23n521lu21b1t136rhbv7ezngie;
    csrftoken=zIPUJsAZv6pcgCBJSCj1zU6pQZbfMUAT; dwf_section_edit=False;
    dwf_sg_task_completion=False; _gat=1; _ga=GA1.2.1688886003.1471911953; ffo=true
```

Перший і другий рядки містять значну частину інформації:

- Тип запиту (GET).
- URL цільового ресурсу (/ en-US / search).
- Параметри URL (q = client% 2Bserver% 2Boverview & topic = apps & topic = html & topic = css & topic = js & topic = api & topic = webdev).
- Цільовий вебсайт (developer.mozilla.org).

Кінець першого рядка так само містить короткий рядок, що ідентифікує версію протоколу (HTTP / 1.1).

Останній рядок містить інформацію про клієнтських куки - в даному випадку можна побачити куки, які включають id для керування сесіями (Cookie: sessionid = 6ynxs23n521lu21b1t136rhbv7ezngie; ...).

Решта рядків містять інформацію про браузер, який використовується, і про його деякі можливості. Наприклад, тут Ви можете побачити:

- Мій браузер (User-Agent) Mozilla Firefox (Mozilla / 5.0).
- Він може приймати інформацію упаковану gzip (Accept-Encoding: gzip).
- Він може приймати зазначені кодування (Accept-Charset: ISO-8859-1, UTF-8; q = 0.7, *; q = 0.7) і мов (Accept-Language: de, en; q = 0.7, en-us; q = 0.3).
- Рядок Referer ідентифікує адресу веб сторінки, що містить посилання на даний ресурс (тобто джерело оригінального запиту, https://developer.mozilla.org/en-US/).

HTTP запит так само може містити body (в даному випадку воно порожнє).

Відповідь

Перша частина відповіді на запит можна переглянути нижче. Заголовок містить наступну інформацію:

- перший рядок містить код відповіді 200 OK, який свідчить про те, що запит виконано успішно.
- видно, що відповідь має text / html формат (Content-Type).
- також видно, що відповідь використовує кодування UTF-8 (Content-Type: text / html; charset = utf-8).
- заголовок так само містить розмір відповіді (Content-Length: 41823).

В кінці повідомлення видно вміст body, що містить HTML-код відповіді, що повертається.

```
HTTP/1.1 200 OK
Server: Apache
X-Backend-Server: developer1.webapp.scl3.mozilla.com
Vary: Accept,Cookie, Accept-Encoding
Content-Type: text/html; charset=utf-8
Date: Wed, 07 Sep 2016 00:11:31 GMT
Keep-Alive: timeout=5, max=999
Connection: Keep-Alive
X-Frame-Options: DENY
Allow: GET
X-Cache-Info: caching
Content-Length: 41823


<!DOCTYPE html>
<html lang="en-US" dir="ltr" class="redesign no-js"  data-ffo-opensanslight=false data-ffo-opensans=false >
<head prefix="og: http://ogp.me/ns#">
  <meta charset="utf-8">
  <meta http-equiv="X-UA-Compatible" content="IE=Edge">
  <script>(function(d) { d.className = d.className.replace(/\bno-js/, '');
})(document.documentElement);</script>
  ...
```

Інша частина заголовка відповіді містить інформацію про саму відповідь (наприклад, коли її було згенеровано), про сервер і про те, як він очікує, що браузер обробляє сторінку (наприклад, рядок X-Frame-Options: DENY говорить браузеру не допускати впровадження цієї сторінки, якщо вона буде впроваджена в <iframe> на іншому сайті).

2.1.2. Приклад запиту / відповіді POST

HTTP POST створюється, коли ви відправляєте форму, яка містить інформацію, що повинна бути збережена на сервері.

Запит

У наведеному нижче тексті показаний HTTP-запит, зроблений тоді, коли користувач представляє нові дані профілю на цьому сайті. Формат запиту майже такий же, як приклад запиту GET, показаний раніше, хоча перший рядок ідентифікує цей запит як POST.

```
POST https://developer.mozilla.org/en-US/profiles/hamishwillee/edit HTTP/1.1
Host: developer.mozilla.org
Connection: keep-alive
Content-Length: 432
Pragma: no-cache
Cache-Control: no-cache
```

```
    Origin: https://developer.mozilla.org
    Upgrade-Insecure-Requests: 1
    User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
    Chrome/52.0.2743.116 Safari/537.36
    Content-Type: application/x-www-form-urlencoded
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
    Referer: https://developer.mozilla.org/en-US/profiles/hamishwillee/edit
    Accept-Encoding: gzip, deflate, br
    Accept-Language: en-US,en;q=0.8,es;q=0.6
    Cookie: sessionid=6ynxs23n521lu21b1t136rhbv7ezngie; _gat=1;
    csrftoken=zIPUJsAZv6pcgCBJSCj1zU6pQZbfMUAT; dwf_section_edit=False;
    dwf_sg_task_completion=False; _ga=GA1.2.1688886003.1471911953; ffo=true

    csrfmiddlewaretoken=zIPUJsAZv6pcgCBJSCj1zU6pQZbfMUAT&user-username=hamishwillee&user-
    fullname=Hamish+Willee&user-title=&user-organization=&user-location=Australia&user-
    locale=en-US&user-timezone=Australia%2FMelbourne&user-irc_nickname=&user-
    interests=&user-expertise=&user-twitter_url=&user-stackoverflow_url=&user-
    linkedin_url=&user-mozillians_url=&user-facebook_url=
```

Основна відмінність полягає в тому, що *URL-адреса не має параметрів*. Інформація з форми закодована в тілі запиту (наприклад, нове повне ім'я користувача встановлюється з використанням: & user-fullname = Hamish + Willee).

Відповідь

Розглянемо відповідь запиту, що показана нижче. Код стану «302 Found» повідомляє браузеру, що повідомлення вдалося, і що він повинен видати другий HTTP-запит для завантаження сторінки, яка зазначена в поле «Місце». В іншому випадку інформація аналогічна інформації для відповіді на запит GET.

```
    HTTP/1.1 302 FOUND
    Server: Apache
    X-Backend-Server: developer3.webapp.scl3.mozilla.com
    Vary: Cookie
    Vary: Accept-Encoding
    Content-Type: text/html; charset=utf-8
    Date: Wed, 07 Sep 2016 00:38:13 GMT
    Location: https://developer.mozilla.org/en-US/profiles/hamishwillee
    Keep-Alive: timeout=5, max=1000
    Connection: Keep-Alive
    X-Frame-Options: DENY
```

```
X-Cache-Info: not cacheable; request wasn't a GET or HEAD
Content-Length: 0
```

2.2. Веб-структури спрощують веб-програмування з боку сервера

Веб-фреймворки з боку сервера роблять код запису значно простіше.

Однією з найбільш важливих операцій, яку вони виконують, є надання простих механізмів для зіставлення URL-адрес для різних ресурсів / сторінок з конкретними функціями обробника. Це спрощує збереження окремого коду, пов'язаного з кожним типом ресурсу. Він також має переваги в плані обслуговування, оскільки надає змогу змінити URL-адресу, що використовується для доставки певної функції в одному місці, без необхідності змінювати функцію обробника.

Наприклад, розглянемо наступний код Django (Python), який відображає два шаблони URL для двох функцій перегляду. Перший шаблон гарантує, що HTTP-запит з URL-адресою ресурсу / best буде передано функції з ім'ям index () в модулі views. Запит, який має шаблон «/ best / junior», відповідно буде передано функції перегляду junior ().

```
# file: best/urls.py
#

from django.conf.urls import url

from . import views

urlpatterns = [
    # example: /best/
    url(r'^$', views.index),
    # example: /best/junior/
    url(r'^junior/$', views.junior),
]
```

Веб-інфраструктура також спрощує процедуру отримання інформації з бази даних. Структура даних (в прикладі) визначається в моделях, що є класами Python для визначення полів, які повинні зберігатися в основній базі даних. Наприклад, якщо є модель з ім'ям Team з полем «team_type», то можна використати простий синтаксис запиту, щоб повернути всі команди, які мають певний тип.

У наведеному нижче прикладі подано список всіх команд, у яких є точний (з урахуванням регістру) team_type (тип_команди) «junior». Зверніть увагу на формат: ім'я поля (team_type), за яким слідує подвійний знак підкреслення, а потім тип відповідності для використання (в цьому випадку точний). Існує багато інших типів збігів, і їх можна об'єднати. Також можна контролювати порядок і кількість результатів, які повертаються.

```
#best/views.py

from django.shortcuts import render

from .models import Team


def junior(request):
    list_teams = Team.objects.filter(team_type__exact="junior")
    context = {'list': list_teams}
    return render(request, 'best/index.html', context)
```

Після того, як функція junior () отримує список молодших команд, вона викликає функцію render (), передаючи вихідний HttpRequest, HTML-шаблон і об'єкт «context», що визначає інформацію, яка повинна бути включена в шаблон. Функція render () - це функція зручності, яка генерує HTML з використанням контексту і HTML-шаблону і повертає його в об'єкт HttpResponse.

3. Фреймворки серверної частини

Серверні веб-фреймворки (або «фреймворки веб-додатків») - *це програмні середовища, які спрощують створення, підтримку і масштабування веб-додатків*. Вони надають інструменти та бібліотеки, які спрощують спільні завдання веб-розробки, включаючи *маршрутизацію* URL-адрес для відповідних обробників, *взаємодію* з базами даних, *підтримку* сеансів і авторизацію користувачів, *форматування* виводу (наприклад, HTML, JSON, XML) і поліпшення *захисту* від веб -атаки.

3.1. Що може зробити веб-фреймворк для вас?

Веб-фреймворки надають інструменти та бібліотеки для спрощення загальних операцій веб-розробки. Ви не зобов'язані використовувати веб-фреймворк на стороні сервера, але це настійно рекомендується – бо зробить ваше життя значно простіше.

3.1.1. Працюйте безпосередньо з HTTP-запитами і відповідями

Веб-сервери і браузери обмінюються даними за протоколом HTTP - сервери очікують HTTP-запити від браузера, а потім повертають інформацію у HTTP-відповідях. Веб-фреймворки дозволяють використовувати спрощений синтаксис, який буде генерувати серверний код для роботи з цими запитами і відповідями. І це означає, що треба буде взаємодіяти з більш простим кодом більш високого рівня, а не з мережевими примітивами нижчого рівня.

Приклад демонструє, як це працює в веб-фреймворку Django (Python). Кожна функція «view» (обробник запиту) отримує об'єкт HttpRequest, що містить інформацію про запит, і повинна повернути об'єкт HttpResponse з форматованим висновком (в даному випадку рядок).

```
# Django view function
from django.http import HttpResponse

def index(request):
    # Get an HttpRequest (request)
    # perform operations using information from the request.
    # Return HttpResponse
    return HttpResponse('Output string to return')
```

3.1.2. Запити маршруту до відповідного обробника

Більшість сайтів надають можливість працювати з кількома різними ресурсами, доступними через окремі URL-адреси. Організувати і підтримувати роботу з цими функціями в одній буде важко, тому веб-фреймворки надають прості механізми для зіставлення шаблонів URL-адрес з конкретними функціями обробника. Цей підхід також має переваги з точки зору обслуговування, тому що надає змогу змінити URL-адресу, яка використовується для отримання певної функції, без

зміни базового коду. Різні фреймворки використовують різні механізми для зіставлення. Наприклад, веб-фреймворк Flask (Python) додає маршрути для перегляду функцій використовуючи декоратори.

3.1.3. Спростіть доступ до даних в запиті

Дані можуть бути закодовані в HTTP-запиті у різний спосіб. Для отримання файлів або даних з сервера, HTTP-запит GET може кодувати які саме дані необхідні в URL параметрах або в структурі URL. HTTP-запит POST для поновлення ресурсу на сервері замість цього буде включати оновлену інформацію як "POST дані" всередині свого тіла. HTTP-запит може також включати інформацію про поточну сесії або користувача в cookie з боку клієнта.

Веб-фреймворки надають відповідні до мови програмування механізми доступу до інформації. Наприклад, об'єкт HttpRequest, який Django передає кожній функції уявлення, містить методи і властивості для доступу до цільового URL, типу запиту (наприклад, HTTP GET), параметрам GET або POST, файлам cookie і даними сеансу і т.д. Django також може передавати інформацію, закодовану в структурі URL шляхом визначення «шаблонів захоплення» в перетворювачі URL (див. останній фрагмент коду в розділі вище).

3.1.4. Абстрагується і спростите доступ до бази даних

Веб-сайти використовують бази даних для зберігання інформації як ДЛЯ користувачів, так і ПРО користувачів. Веб-платформи часто представляють шар бази даних, який абстрагує операції читання, запису, запиту і видалення бази даних. Цей рівень абстракції називається **Object-Relational Mapper** (ORM).

Використання ORM має дві переваги:

- можна замінити базу даних без необхідності змінювати код, в якому вона використовується, що дозволяє оптимізувати характеристики різних баз даних в залежності від їх використання.
- можна реалізувати базову перевірку даних. Це дозволяє легше і безпечніше перевірити, чи дані зберігаються в вірному типу поля бази даних, мають правильний формат (наприклад, адреса електронної пошти) і не є шкідливими (зломщики можуть використовувати певні шаблони коду, щоб зробити такі речі, як видалення записів бази даних) .

Наприклад, веб-інфраструктура Django надає ORM і посилається на об'єкт, який використовується для визначення структури записи в якості моделі. Модель задає типи полів для збереження, що дозволяє забезпечити перевірку на рівні поля щодо того, яка інформація може бути збережена (наприклад, поле електронної пошти буде вирішувати тільки дійсні адреси електронної пошти). У налаштуваннях полів також можна вказати їх максимальний розмір, значення за замовчуванням, параметри списку вибору, текст довідки для документації, текст мітки для

форм і т. д. Модель не містить ніякої інформації про базу даних, оскільки це конфігураційний параметр, який може бути змінений, окремо від коду.

Перший фрагмент коду нижче демонструє просту модель Django для об'єкта Team. Який зберігає назву команди та її рівень як символьні поля і визначає максимальну кількість символів для кожного запису. Team_level - це поле вибору, тому ми також надаємо відображення між варіантами відображення і даними для зберігання разом зі значенням за замовчуванням.

```
#best/models.py

from django.db import models

class Team(models.Model):
    team_name = models.CharField(max_length=40)

    TEAM_LEVELS = (
        ('U09', 'Under 09s'),
        ('U10', 'Under 10s'),
        ('U11, 'Under 11s'),
        ...  #list our other teams
    )
    team_level = models.CharField(max_length=3,choices=TEAM_LEVELS,default='U11')
```

Модель Django надає простий API запит для пошуку в базі даних. За декількома полями одночасно, використовуючи різні критерії (наприклад, точний, без урахування регістру, більше ніж і т.д.), І може підтримувати складні оператори (наприклад, ви можете вказати пошук для команд U11, в яких є команда ім'я, яке починається з «Fr» або закінчується на «al»).

Другий фрагмент коду демонструє функцію подання (обробник ресурсів) для відображення всіх команд U09. В цьому випадку треба вказати, що відфільтрувати для всіх записів, де поле team_level має в точності текст «U09» (зверніть увагу нижче, як цей критерій передається функції filter () як аргумент з ім'ям поля і типом відповідності, розділеними подвійним підкреслення : team_level__exact).

```
#best/views.py

from django.shortcuts import render
from .models import Team

def youngest(request):
    list_teams = Team.objects.filter(team_level__exact="U09")
    context = {'youngest_teams': list_teams}
    return render(request, 'best/index.html', context)
```

3.1.5. Рендерінг (Візуалізація) даних

Веб-фреймворки часто надають системи шаблонів. Вони дозволяють вказати структуру вихідного документа, використовуючи наповнювачі для даних, які будуть додані при створенні сторінки. Шаблони часто використовуються для створення HTML, хоча дозволяють створювати і інші типи документів.

Веб-фреймворки часто надають механізм, що дозволяє легко створювати інші формати з збережених даних, включаючи JSON і XML.

Наприклад, система шаблонів Django дозволяє задавати змінні з використанням синтаксису «подвійних рулів» (наприклад, {{variable_name}}), який буде замінений значеннями, переданими з функції уявлення при візуалізації сторінки. Система шаблонів також забезпечує підтримку виразів (з синтаксисом: {% expression%}), які дозволяють шаблонами виконувати прості операції, такі як ітерація значень списку, переданих в шаблон. Багато інших систем шаблонів використовують аналогічний синтаксис, наприклад: Jinja2 (Python), handlebars (JavaScript), moustache (JavaScript) і т. д.

На прикладі нижче розглянемо, як це працює. Продовжимо приклад «наймолодшої команди» - шаблон HTML передає поданню змінну списку youngest_teams. Всередині тіла HTML у нас є вираз, який спочатку перевіряє, чи існує змінна youngest_teams, а потім повторює її в циклі for. На кожній ітерації шаблон відображає значення team_name команди в елементі списку.

```
#best/templates/best/index.html

<!DOCTYPE html>
<html lang="en">
<body>

 {% if youngest_teams %}
    <ul>
    {% for team in youngest_teams %}
        <li>{{ team.team_name }}</li>
    {% endfor %}
    </ul>
{% else %}
    <p>No teams are available.</p>
{% endif %}

</body>
</html>
```

3.2. Як вибрати веб-фреймворк

Численні веб-фреймворки існують практично для кожної мови програмування, тут перерахуємо кілька найбільш популярних фреймворків. При такій великій кількості варіантів іноді складно визначити, яка структура забезпечує кращу відправну точку для конкретного нового веб-додатки.

Ось деякі з факторів, які можуть вплинути на це рішення:

Зусилля з вивчення: зусилля з вивчення веб-середовища залежать від того, наскільки ви знайомі з базовою мовою програмування, послідовністю його API, якістю документації, а також розміром і активністю його спільноти. Якщо ви починаєте з абсолютно ніякого досвіду програмування, подумайте про Django (це

один з найпростіших способів вивчення на основі вищевказаних критеріїв). Якщо ви є частиною команди розробників, яка вже має значний досвід роботи з певною веб-інфраструктурою або мовою програмування, то має сенс дотримуватися її.

Продуктивність: продуктивність - це показник того, наскільки швидко ви можете створювати нові функції, коли ви знайомі з платформою, і включає в себе зусилля з написання і обслуговування коду (оскільки ви не можете писати нові функції, поки старі не працюють). Багато з факторів, що впливають на продуктивність, аналогічні тим, які використовуються для «Зусилля з навчання» - наприклад, документація, співтовариство, досвід програмування і т.д. - інші чинники включають в себе:

Призначення / походження фреймворка: Деякі веб-фреймворки спочатку створювалися для вирішення певних типів проблем і залишаються кращими при створенні веб-додатків з аналогічними обмеженнями. Наприклад, Django був створений для підтримки розробки газетного веб-сайту, тому він хороший для блогів і сайтів, пов'язаних з публікацією матеріалів. Flask, навпаки, є більш легким середовищем і доре підходить для створення веб-додатків, що працюють на вбудованих пристроях.

Думка проти невпізнаного. Фракція, заснована на думці, - це та, в якій рекомендуються «кращі» шляхи вирішення конкретної проблеми. Багато фреймворків, як правило, більш продуктивні, коли ви намагаєтеся вирішити спільні проблеми, тому що вони ведуть вас в правильному напрямку, однак іноді вони менш гнучкі.

Все включено проти розбирайтеся самі: деякі веб-платформи включають в себе інструменти / бібліотеки, які вирішують кожну проблему, яку їх розробники можуть вважати «за замовчуванням», в той час як інші, більш легкі платформи очікують, що веб-розробники будуть вибирати рішення проблем з окремих бібліотек (наприклад, Django з перших, в той час як Flask є прикладом дуже легкого каркаса). З фреймворків, які включають в себе все, часто легше почати, тому що у них є все, що вам потрібно, і є ймовірність, що такі фреймворки добре інтегровані і документовані.

Незалежно від того, чи заохочує платформа гарні практики розробки: наприклад, структура, яка заохочує архітектуру Model-View-Controller для поділу коду на логічні функції, призведе до більш підтримуваного коду. Так само дизайн фреймворка може вплинути на те, наскільки легко тестувати і повторно використовувати код.

Продуктивність фреймворка / мови програмування: зазвичай «швидкість» не є найважливішим фактором при виборі, тому що навіть відносно повільні середовища виконання, такі як Python, більш ніж «досить добрі» для сайтів середнього розміру, що працюють на середньому обладнанні.

Підтримка кешування. Зі зростанням успішності сайту можна помітити, що він більше не може впоратися з усією кількістю запитів, які отримує від користувачів. На цьому етапі можна розглянути можливість додавання підтримки кешування. Кешування - це оптимізація, при якій зберігається весь або частина веб-запиту, на випадок повторних запитів. Повернення кешованого запиту набагато швидше, ніж його обчислення. Кешування може бути реалізовано в вашому коді або на сервері. Веб-фреймворки мають різні рівні підтримки для визначення того, який контент можна кешувати.

Масштабованість. Як тільки веб-сайт стане фантастично успішним, вичерпаєте усі переваги кешування і навіть досягнете меж вертикального масштабування (запуску веб-додатків на більш потужному обладнанні). На цьому етапі може знадобитися масштабувати горизонтально (розподілити навантаження, яке отримує сайт між декількома веб-серверами і базами даних) або масштабувати «географічно», тому що деякі з клієнтів знаходяться далеко від сервера. Обраний веб-фреймворк може істотно вплинути на те, наскільки легко буде масштабувати сайт.

Веб-безпека. Деякі веб-платформи надають кращу підтримку для обробки поширених веб-атак. Наприклад, Django очищає весь призначений для користувача введення від шаблонів HTML, щоб користувальницький JavaScript не міг бути запущений. Інші платформи надають аналогічну захист, але вона не завжди включена за замовчуванням.