УДК 004.056
В.М. Мельник, К.В. Мельник, Б.В. Шульга
Луцький національний технічний університет

ПОРІВНЯННЯ ТРЬОХ НАЙПОПУЛЯРНІШИХ WEB-КАРКАСІВ ДЛЯ РОЗРОБКИ ПРОЕКТІВ В PYTHON

Мельник В.М., Мельник К.В., Шульга Б.В. Порівняння трьох найпопулярніших web-каркасів для розробки проектів в Python. В даній статті наведено результати досліджень особливостей застосування трьох найпопулярніших web-каркасів для розробки проектів у Pyton: Pyramid, Django та Flask. Відмічається, що найбільш гнучкий каркас Pyramid може бути застосований лише для створення невеликих додатків з відкритим кодом. Однак, Django надає чудовий набір компонентів за замовчуванням і з цієї причини він набуває популярності у виборі для створення середніх та великих веб-програм. Каркас Flask відмінно підходить для розробників, які працюють з невеликими проектами і потребують швидкого створення web-додатків, простого веб-інтерфейсу і вимагає невеликої конфігурації.

Всі три каркаси задовольняють відповідний список вимог, який змінює дизайн продукту та швидкість роботи нових функцій та виправлень. Тут Flask показує себе з найкращого боку і як Django може відчувати себе незграбним у невеликому проектному масштабі. Гнучкість і розширюваність Pyramid не відіграє важливої ролі для розробників, але слід також враховувати нові вимоги до динамічного технічного забезпечення, що постійно висуваються розробниками.

Мельник В.М., Мельник Е.В., Шульга Б.В. Сравнение трех самых популярных web-каркасов для разработки проектов в Python. В данной статье приведены результаты исследований особенностей применения трех самых популярных web-каркасов для разработки проектов в Pyton: Pyramid, Django и Flask. Отмечается, что наиболее гибкий каркас Pyramid может быть применен только для создания небольших приложений с открытым кодом. Однако, Django предоставляет замечательный набор компонентов по умолчанию и по этой причине он приобретает популярность в выборе для создания средних и крупных веб-приложений. Каркас Flask отлично подходит для разработчиков, работающих с небольшими проектами которые требуют быстрого создания web-приложений, простого веб-интерфейса и небольшой конфигурации.

Все три каркасы удовлетворяют соответствующий список требований, который меняет дизайн продукта и скорость работы новых функций и исправлений. Здесь Flask показывает себя с наилучшей стороны и как Django может чувствовать себя неуклюжим в небольшом проектном масштабе. Гибкость и расширяемость Pyramid не играется важную роль для разработчиков, но следует также учитывать и новые требования к динамическому техническому обеспечению, которые постоянно выдвигаются разработчиками.

Melnyk V.M., Melnyk K.V., Shulga B.V. A comparison of the three most popular web-frameworks for Python project development. This article presents the research results on the peculiarities of using the three most popular web-frames for Pyton project development: Pyramid, Django and Flask. It is noted that the most flexible framework Pyramid can only be used to create small applications with open source code. However, Django provides a wonderful set of components by default and for this reason it is becoming popular in choice for development of medium and large web applications. The Flask framework is great for developers who work with small projects and require the rapid creation of web-based applications, a simple web-interface and requires a small configuration.

All three frameworks meet the relevant requirements list, which changes product design and the speed of new features and fixes. Here Flask shows itself on the best side and how Django can feel awkward on a small projective scale. Flexibility and extensibility of Pyramid does not play an important role for developers, but it should also take into account the new requirements for dynamic technical support that are every time being put forward by developers.

1. Постановка проблеми. Світ веб-каркасів Python повний вибору: Django, Flask, Pyramid, Tornado, Bottle, Diesel, Pecan, Falcon та багато інших, що конкурують за інтересами розробників. Розробник може скоротити варіанти вибору і це допоможе завершити йому проект і перейти до наступного. В даній статті ми зосередимося на Flask, Pyramid та Django [5,9,12]. Ці каркаси охоплюють рішення від мікропроекту до веб-служби розміром підприємства.

Щоб допомогти зробити вибір між трьома простими (або, принаймні більш обгрунтованими) каркасами, ми можемо створити один і той же додаток у кожній структурі та порівняємо код, виділяючи сильні та слабкі сторони кожного підходу.

Flask являє собою "мікрокаркас", перш за все, спрямований на малі програми з більш простими вимогами. Pyramid і Django спрямовані на програми більшого об'єму, однак використовують вони різні підходи до розширюваності та гнучкості [7]. Pyramid – націлений на гнучкість і дозволяє розробнику використовувати потрібні інструменти для свого проекту. Це означає, що розробник може вибрати базу даних, структуру URL-адреси, стиль шаблону тощо. Django прагне включити всі "батарейки" до веб-додатків, тому розробникам потрібно лише відкрити вікно і почати працювати, позичаючи багато модулів Django з коробки.

Django включає в себе ORM з коробки, а Pyramid та Flask залишають ці речі за розробником, щоб вибрати, як вони хочуть зберігати свої дані чи взагалі для збереження даних в цілому [1]. Найпопулярнішою програмою ORM для веб-додатків крім Django є SQLAlchemy [3,11]. Однак існує безліч інших варіантів від DynamoDB і MongoDB до простих локальних завдань, таких як LevelDB або простий SQLite. Все ж Pyramid призначений для використання будь-якого стійкого шару, навіть, можливо, ще не винайденого.

2. Аналіз останніх досліджень. Підхід з використанням "батарейок", включених в комплект Django дозволяє розробникам, які знають Python вже давно, занурюватися в веб-додатки швидко, без необхідності достроково приймати багато рішень про їх інфраструктуру. Django має власні інструменти для обробки шаблонів, форм, маршрутизації, автентифікації, адміністрування баз даних та інші. На відміну від Pyramid, який включає в себе маршрутизацію та автентифікацію, адміністрування шаблонів і баз даних вимагає зовнішніх бібліотек.

Вибір компонентів для додатків Flask та Pyramid дає більшу гнучкість для розробників, використання яких не відповідає стандартному ORM і потребує взаємодії з різними робочими процесами або шаблонами.

Flask – наймолодший з трьох каркасів, з'явився в середині 2010 року. Pyramid почав своє життя в проекті Pylons і отримав назву Pyramid. Все ж перший реліз був здійснений у 2005 році. Django випустив свій перший реліз у 2006 році, незабаром після початку проекту Pylons (Pyramid). Pyramid і Django – це надзвичайно досконалі каркаси, які мають багато плагінів та розширень для задоволення неймовірно великої кількості потреб.

Хоча Flask має більш коротку історію, але перехід від каркасів, що існували раніше, не буде великою складністю. Він міцно встановив свої пам'ятки на малих проектах. Найчастіше використовується він у невеликих проектах з лише однією або двома функціями. Один з таких проектів – це httpbin, простий, але надзвичайно потужний помічник для налагодження та тестування бібліотек HTTP.

Що ж спільного в них? Найбільш активною на StackOverflow є спільнота Django з 165873 питань, зафіксованих в Інтернет-дослідженні та здоровим набором блогів від розробників та користувачів. Спільнота Flask та Pyramid не настільки великі, але їх спільноти досить активні у своїх списках розсилки та в IRC. Зібравши лише 20723 питань StackOverflow, Flask є у 8 разів менший, ніж Django. На Github вони мають майже однакову кількість зірок: 32580 – Django, 34029 – Flask. Всі три каркаси доступні під ліцензіями BSD.

Метою даної статті послужило дослідження порівняння трьох найпопулярніших web-каркасів для розробки проектів в python. Зокрема, ставилося питання гнучкості у використанні, наповнення зручним інтерфейсом для розробників, використання ресурсів та інших програмних частин в ході розробки вищезгаданих додатків.

3. Основна частина. Що стосується завантаження, то Django та Pyramid поставляються з вбудованими інструментами для створення масивних програм-додатків. Flask – не містить нічого подібного, оскільки цільова аудиторія Flask не намагається створювати великі MVC-проекти.

Сама найпростіша програма "Hello World!" є базовою, і багато IDE, які дозволяють вибрати тип проекту, створюють щось подібне в якості шаблону. Наведемо простий приклад "Hello World!" з http://flask.pocoo.org/ tutorial:

```
from flask import Flask
app = Flask(__name__)
@app.route('/)
def hello_world():
   return "Hello World!"
if __name__ == '__main__':
   app.run()
```

Ось чому для Flask немає інструментів завантаження – на них не існує попиту. Це видно звичайному програмісту з наведеного вище прикладу Hello World, розміщеного на домашній сторінці Flask.

Для проектів, що потребують більшої відокремленості компонентів, Flask має у своєму функціоналі макети (blueprint). Наприклад, розробник може структурувати додаток Flask з усіма функціями, пов'язаними з користувачем, у users.py та функціями, пов'язаними з продажами, в ecommerce.py, а потім імпортувати їх і додавати їх у свій додаток на site.py.

Pyramid – це інструмент завантаження піраміди, що називається pcreate і є частиною Pyramid. Раніше набір інструментів "Вставити" передбачав завантаження, але потім це було замінено спеціальним інструментом для Pyramid, наприклад:

```
$ pcreate -s starter hello_pyramid
```

Пірамід, порівняно з Flask, призначений для великих і складних додатків. Через це його інструмент завантаження створює великий скелет проекту. Він також містить основні файли конфігурації. Приклад шаблону дерева та файли для додавання програми розробника для завантаження в індекс-пакета Python наведемо нижче.

```
hello_pyramid
├── CHANGES.txt
├── development.ini
├── MANIFEST.in
├── production.ini
├── hello_pyramid
│   ├── __init__.py
│   ├── статичний
│   │   ├── піраміди-16x16.png
│   │   ├── пірамід.png
│   │   ├── theme.css
│   │   └── theme.min.css
│   ├── шаблони
│   │   └── mytemplate.pt
│   ├── tests.py
│   └── views.py
├── README.txt
└── setup.py
```

Як і в інших каркасах, завантажувач Pyramid є неймовірно гнучким. Він не обмежується однією програмою за замовчуванням; pcreate може використовувати будь-яку кількість шаблонів проектів. Включений у створення шаблон "starter" можна використовувати разом із проектами, які підтримують SQLAlchemy і ZODB. На PyPi можна знайти готові розроблені додатки для Google App Engine, jQuery Mobile, Jinja2, сучасних frontend фреймворків та багато іншого [2,10].

Django також має свій власний інструмент завантаження, побудований як частина django-admin.

```
django-admin startproject hello_django
django-admin startapp howdy
```

Ми вже можемо бачити один із способів, який відрізняє Django від Pyramid. Видно, що Django розділяє проект на індивідуальні програми, тоді як Pyramid і Flask очікують, що проект буде "єдиним додатком" з кількома видами (функції відображення views) або моделями. Тут можна відтворити структуру проекту на Flask та Pyramid, але це рішення не існує за замовчуванням.

```
hello_django
├── hello_django
│   ├── __init__.py
│   ├── settings.py
│   ├── urls.py
```

```
    └── wsgi.py
├── вонді
│   ├── admin.py
│   ├── __init__.py
│   ├── migrations
│   │   └── __init__.py
│   ├── models.py
│   ├── tests.py
│   └── views.py
└── manage.py
```

За замовчуванням Django включає лише порожні моделі та файли шаблонів, так що новий користувач бачить трохи менше коду для прикладу. Це також (на жаль) залишає вибір за розробником, як розповсюджувати свій проект.

Недоліком цього інструменту і його завантаження є складність для розуміння новачкам. Якщо розробник раніше не упаковував додаток, то процедура розгортання буде здаватись їм затратною в часі справою. Проекти з великою спільнотою, наприклад, django-oscar, упаковані та доступні на PyPi, але менші проекти на Github часто не мають єдиної упаковки.

І ще одна перспектива роботи з шаблонізаторами. Слід зазначити, що запуск додатка Python, який може обробляти HTTP-запити, є чудовим початком, але більшість активних користувачів не будуть зацікавлені в використанні Curl для взаємодії з веб-програмою розробника. На щастя, всі три суперники надають простий спосіб заповнити HTML-коди з користувальницькою інформацією, а також дозволяють людям насолоджуватися розробленим чудовим інтерфейсом Bootstrap.

Шаблон дозволяє вставляти динамічну інформацію безпосередньо на сторінку без використання запитів AJAX. Це приємно з точки зору досвіду роботи з користувачем, оскільки розробнику потрібно лише зробити один запит, щоб отримати всю сторінку та всі її динамічні дані. Це особливо важливо на мобільних сайтах, де запити можуть тривати кілька секунд.

Всі параметри шаблонів, які можна побачити, покладаються на "контекст", який забезпечує динамічну інформацію для шаблону, в ході перетворення в HTML. Найпростіший спосіб використання шаблону полягає в тому, щоб заповнити ім'я входу в систему і привітати користувача належним чином. Можна було б використовувати AJAX, щоб отримати подібну динамічну інформацію, але для повного виклику, щоб просто заповнити ім'я користувача, даний підхід буде трохи надмірним у порівнянні з шаблонами.

Що стосується Django, то в прикладному використанні він дуже простий. Припустимо, що у нас є об'єкт користувача, який має властивість повного імені, який також містить ім'я користувача. У Python ми передали поточному користувачеві шаблон наступним чином [3,4]:

```
def a_view(request):
   return render_to_response(
      "view.html",
      {"user": cur_user})
```

Заповнення контексту шаблону настільки ж просте, як передавання словника об'єктів Python та структур даних, які повинен використовувати шаблон. Тепер нам потрібно відображати назву на сторінці, лише якщо вони забудуть, хто вони.

```
<!-- view.html -->
<div class="top-bar row">
 <div class="col-md-10">
 <!-- more top bar things go here -->
 </div>
 {% if user %}
 <div class="col-md-2 whoami">
  You are logged in as {{ user.fullname }}
```

```
  </div>
  {% endif %}
</div>
```

По-перше, ви помітите використання {% if user %}. У шаблонах Django {% використовується для керуючих тверджень, таких як цикли та умовні оператори. Якщо команда 'if user' використовується, щоб запобігти випадкам, коли немає користувача. Анонімні користувачі не повинні бачити залогінення на сайті.

У блоці if можна побачити, що включення імені є настільки ж простим, як обертання властивості, яку потрібно вставити в {{}}. "{{" – використовується для вставки фактичних значень у шаблон, наприклад {{user.fullname}}.

Ще одним загальним використанням шаблонів є відображення груп предметів, таких як сторінка інвентаризації для комерційного сайту.

```
def browse_shop(request):
   return render_to_response(
      "browse.html",
      {"inventory": all_items})
```

У шаблоні ми також можемо використовувати оператор "{%", щоб обійти цикл по всіх елементах інвентаря та заповнити URL-адресу своєї окремої сторінки.

```
{% for widget in inventory %}
   <li><a href="/widget/{{ widget.slug }}/">{{ widget.displayname }}</a></li>
{% endfor %}
```

Для виконання найбільш поширених завдань з шаблонізації Django може досягти мети, використовуючи лише декілька конструкцій, що і значно полегшує початок роботи.

Flask за замовчуванням використовує шаблонізатор Jinja2, але може бути налаштованим на використання іншого шаблонізатора. Обидва приклади DjangoTemplates, згадані вище, працюють і у Jinja2. Замість того, щоб перебирати ті ж приклади, можна розглянути ті місця, які відрізняють Jinja2 від DjangoTemplates.

Як Jinja2, так і DjangoTemplates надають функцію, яка називається фільтром, де список може бути переданий через функцію перед тим, як відображатись [10]. Блог, в якому доступні категорії публікацій, можуть використовувати фільтри [1], щоб відображати їх категорії у списках, розділених комами.

```
<!-- DjangoTemplates -->
<div class="categories">Categories: {{ post.categories|join:", " }}</div>

<!--  Jinja2 -->
<div class="categories">Categories: {{ post.categories|join(", ") }}</div>
```

У шаблонізаторі Jinja можна передавати у фільтр будь-яку кількість аргументів, оскільки Jinja розглядає його як виклик функції Python в круглих дужках навколо аргументів. DjangoTemplates використовує двокрапку як роздільник між іменем фільтра та аргументом фільтра, який обмежує кількість аргументів до одного.

Виклик циклів виглядає подібним чином як у Jinja так і в DjangoTemplates. Подивимося, чим вони відрізняються. У Jinja2 конструкція for-else-endfor дозволяє розробнику прокручувати список, а також керувати випадком, коли немає елементів.

```
{% for item in inventory %}
<div class="display-item">{{ item.render() }}</div>
{% else %}
<div class="display-warn">
```

```
<h3>No items found</h3>
<p>Try another search, maybe?</p>
</div>
{% endfor %}
```

Версія Django цієї функції однакова, але використовує for-empty-endfor, а не for-else-endfor.

```
{% for item in inventory %}
<div class="display-item">{{ item.render }}</div>
{% empty %}
<div class="display-warn">
<h3>No items found</h3>
<p>Try another search, maybe?</p>
</div>
{% endfor %}
```

Окрім синтаксичних розбіжностей показаних вище, Jinja2 забезпечує більший контроль над середовищем виконання та розширеними функціями. Наприклад, можна відключити потенційно небезпечні функції, щоб безпечно виконувати ненадійні шаблони, або скомпілювати шаблони заздалегідь, щоб забезпечити їх дійсність (протидія ін'єкціям).

Як Flask, так і *Pyramid* підтримує багато шаблонізаторів, у тому числі Jinja2 і Mako, але за замовчуванням Pyramid використовує Chameleon – реалізацію шаблону ZPT (Zope Page Template). Давайте розглянемо наведений перший приклад, додавши ім'я користувача до верхньої панелі розробленого сайту. Код на Python виглядає так само, крім того, нам не потрібно явно називати функцію render_template [14].

```
@view_config(renderer='templates/home.pt')
def my_view(request):
   # TODO something
   return {'user': user}
```

Розроблений власний шаблон виглядає інакше. ZPT – це шаблонний стандарт на основі XML, тому ми використовуємо оператори, подібні до XSLT, для обробки даних.

```
<div class="top-bar row">
  <div class="col-md-10">
  <!-- more top bar things go here -->
  </div>
  <div tal:condition="user"
      tal:content="string:You are logged in as ${user.fullname}"
      class="col-md-2 whoami">
  </div>
</div>
```

Chameleon насправді має три різні області простору імен для дій шаблонів. TAL (Template Attribute Language) забезпечує такі основні інструменти, як умовні оператори, засоби форматування рядків та заповнення вмісту тегів [15]. Наведений вище приклад використовував TAL (Template Attribute Language) для завершення своєї роботи. Для більш складних завдань необхідні TALES та METAL. TALES (Template Attribute Language Expression Syntax) містить такі вирази, як розширене форматування рядків, оцінка виразів Python та імпорт виразів і шаблонів.

Однак METAL (Macro Expansion Template Attribute Language) є найпотужнішою (і складною) частиною шаблону Chameleon. Дані макроси легко розширюються, і їх можна визначити як слоти, які заповнюються при виклику макросу.

4. Висновки і перспективи. З проведеного дослідження можна зробити висновок, що Pyramid є найбільш гнучким каркасом з трьох, які бралися на увагу для дослідження. Він може бути використаний для невеликих додатків, а також в складних проектах, наприклад – Dropbox. Спільності з відкритим кодом, такі як Fedora, вибирають його для таких програм, як система значків спільноти, яка отримує інформацію про події з багатьох інструментів проекту, щоб призначати користувачам значки у стилі досягнення. Одна з найбільш поширених скарг на Pyramid полягає в тому, що він містить багато варіантів, що можуть завадити почати новий проект.

Однак, найбільш популярним каркасом є Django, і список сайтів, які використовують його – вражаючий. Bitbucket, Pinterest, Instagram та The Onion використовують Django для всіх або частини своїх сайтів. Для сайтів, із загальними потребами Django надає чудовий набір компонентів за замовчуванням і з цієї причини він став популярним вибором для створення середніх та великих веб-програм.

Каркас Flask відмінно підходить для розробників, які працюють з невеликими проектами, що потребують швидкого створення простого веб-сайту з підтримкою Python. Це дозволяє завантажувати невеликі одноразові інструменти або прості веб-інтерфейси, побудовані за існуючими API. Вбудовані проекти, які потребують простого веб-інтерфейсу, що швидко розвивається і вимагає невеликої конфігурації, часто користуються Flask на зовнішній панелі. Наприклад, jitviewer, який надає веб-інтерфейс для перевірки журналів компіляторів PyPy в режимі реального часу.

Всі три каркаси змогли задовільнити невеликий список вимог, і можна побачити, чим вони відрізняються. Ці розбіжності не просто косметичні, вони також змінять дизайн продукту та швидкість відправлення нових функцій та виправлень. Оскільки наведений приклад невеликий, то видно, де Flask показує себе з найкращого боку і як Django може відчувати себе незграбним у невеликому масштабі. Гнучкість і розширюваність Pyramid не стала важливим фактором, але слід враховувати, що в реальному світі постійно висуваються нові вимоги до "динамічного технічного забезпечення".

1. Мігель Грінберг – Flask. Створення Web-додатків, 2006, - 272 с.
2. Джефф Форс'є – Розробка Web лолатків на Django, 2009, - 456 с.
3. Адріан Головатий - Django. Детальний довідник, 2010, - 550 с.
4. Володимир Дронов. Django: практика створення Web-сайтів, 2016, - 528 с.
5. https://github.com/Pylons/pyramid
6. https://github.com/miguelgrinberg/Flask-SocketIO
7. https://www.djangoproject.com/
8. https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-i-hello-world-legacy
9. https://github.com/django/django
10. http://jinja.pocoo.org/
11. http://docs.sqlalchemy.org/en/latest/