



สัมมนาวิศวกรรมซอฟต์แวร์

การศึกษาวิธีการพัฒนา Web Applications

ด้วยDjango Framework

Study of Web Application Development

Using the Django Framework

โดย

6401260083 ภาณุพงศ์ อาซากิจ

รายงานฉบับนี้เป็นส่วนหนึ่งรายวิชา วช.492 สัมมนาวิศวกรรมซอฟต์แวร์

สาขาวิชาวิศวกรรมซอฟต์แวร์ คณะบริหารธุรกิจ มหาวิทยาลัยพายัพ

ภาคการศึกษาที่ 2 ปีการศึกษา2567

กิตติกรรมประกาศ

รายงานฉบับนี้จัดทำขึ้นเพื่อศึกษาเกี่ยวกับการศึกษาวิธีการพัฒนา Web Applications ด้วย Django Framework ได้สำเร็จลุล่วงแล้วขอขอบคุณ อ. ที่คอยให้คำแนะนำและข้อเสนอแนะอันเป็นประโยชน์ตลอดการทำรายงาน และขอขอบคุณเอกสารแหล่งความรู้และงานวิจัยต่างๆที่เกี่ยวข้องกับ Django Framework และขอบคุณเพื่อนๆ ที่คอยตั้งคำถามสำหรับหัวข้อต่างๆที่เป็นประโยชน์สำหรับการนำไปศึกษาต่อ

สุดท้ายนี้ข้าเจ้าหวังว่าการศึกษานี้จะสามารถเป็นประโยชน์สำหรับผู้สนใจจะศึกษาหัวข้อนี้ต่อไป

คำนำ

รายงานฉบับนี้จัดทำมาเพื่อศึกษาเกี่ยวกับหัวข้อการศึกษาวิธีการพัฒนา Web Applications ด้วย Django Framework เนื่องจาก Framework นี้เป็น Framework ภาษา Python ซึ่งได้รับความนิยมสูงในการเลือกใช้สำหรับการพัฒนา Web Applications โดยเนื้อหาในรายงานฉบับนี้ประกอบด้วย Introduction และหลักการทำงานของ Django Framework การติดตั้งและการเริ่มต้นใช้งาน Django Framework การเชื่อมต่อกับฐานข้อมูลใน Django การใช้งาน Django REST Framework (DRF) เปรียบเทียบ Django & Flask Built-in Features และการนำมาใช้งาน โดยได้ศึกษาจากเอกสารที่เกี่ยวข้องกับการพัฒนา Web Applications ด้วย Django จึงเป็นแหล่งที่มาที่มีความน่าเชื่อถือสูง

ข้าเจ้าหวังเป็นอย่างยิ่งว่ารายงานฉบับนี้จะเป็นประโยชน์ต่อผู้ที่สนใจจะศึกษาหัวข้อนี้ หากมีข้อผิดพลาดประการใด ขออภัยมา ณ ที่นี้

สารบัญ (และสารบัญภาพ สารบัญตารางถ้ามี)

1. กิตติกรรมประกาศ	หน้า 1
2. คำนำ	หน้า 2
3. บทคัดย่อ	หน้า 3
4. บทที่ 1 บทนำ	หน้า 4
○ 1.1 ความเป็นมาและความสำคัญของการศึกษา	หน้า 4
○ 1.2 วัตถุประสงค์ของการศึกษา	หน้า 5
○ 1.3 ขอบเขตของการศึกษา	หน้า 5
○ 1.4 วิธีการดำเนินการศึกษา	หน้า 6
○ 1.5 โครงสร้างรายงาน	หน้า 7
5. บทที่ 2 ทบทวนวรรณกรรม (Literature Review)	หน้า 8
○ 2.1 ความรู้พื้นฐานและแนวคิดหลักในหัวข้อ	หน้า 8
○ 2.2 กรณีสืบศึกษา	หน้า 9
○ 2.3 แนวทางการวิจัยและการพัฒนา	หน้า 10
○ 2.4 ข้อมูลเชิงสถิติและเชิงปริมาณ	หน้า 11
○ 2.5 การประยุกต์ใช้จริง	หน้า 12
6. บทที่ 3 รายละเอียดการศึกษา	หน้า 13
○ 3.1 Introduction และหลักการทำงานของ Django Framework	
▪ 3.1.1 Introduction to Django	หน้า 14
▪ 3.1.2 จุดเด่นของ Django	หน้า 15
▪ 3.1.3 Architecture Workflow	หน้า 16

▪	3.1.4 Key Features and Tools in Django	หน้า 17
○	3.2 การติดตั้งและการเริ่มต้นใช้งาน Django Framework	หน้า 18
○	3.3 การเชื่อมต่อกับฐานข้อมูลใน Django	หน้า 19
▪	3.3.1 ORM คืออะไร	หน้า 20
▪	3.3.2 Django Model และการจัดการข้อมูลใน Model ..	หน้า 21
○	3.4 การใช้งาน Django REST Framework (DRF) สำหรับการสร้าง API	หน้า 22
○	3.5 เปรียบเทียบ Django & Flask	หน้า 23
○	3.6 Built-in Features และการนำมาใช้งาน	หน้า 24
7.	บทที่ 4 สรุปผลการศึกษา	หน้า 25
○	4.1 องค์ความรู้ที่ได้	หน้า 26
▪	4.1.1 Introduction และหลักการทำงานของ Django Framework	หน้า 27
▪	4.1.2 การติดตั้งและการเริ่มต้นใช้งาน Django Framework	หน้า 28
▪	4.1.3 หลักการและการเชื่อมต่อกับฐานข้อมูลใน Django	หน้า 29
▪	4.1.4 Django REST Framework	หน้า 30
▪	4.1.5 เปรียบเทียบ Django & Flask	หน้า 31
▪	4.1.6 Built-in Features และการนำมาใช้งาน	หน้า 32
8.	บรรณานุกรม	หน้า 33

บทคัดย่อ

Django Framework เป็นหนึ่งใน Framework ที่ได้รับความนิยมสูงในการใช้สำหรับพัฒนา Web Applications ที่จะช่วยให้นักพัฒนาสามารถพัฒนา Web Applications ได้อย่างรวดเร็วและมีประสิทธิภาพที่ดี เนื่องจาก Django มีโครงสร้างที่เป็นระเบียบช่วยให้การทำงานเป็นไปได้อย่างราบรื่นและมีเครื่องมือต่างๆ มากมายที่ทำให้การพัฒนาสามารถสร้างระบบที่มีความยืดหยุ่นและปลอดภัยออกมาได้ โดยได้ทำการศึกษาจากเอกสารที่เกี่ยวข้องกับการพัฒนา Web Application ด้วย Django Framework และทดลองสร้าง Web Application โดยใช้ Django Framework ผลจากการศึกษาเอกสารที่เกี่ยวข้องจึงได้พัฒนาระบบบล็อก (Simple Blog) ซึ่งเป็นระบบที่ผู้ใช้สามารถโพสต์บทความและจัดการแก้ไขบทความได้จากการใช้ Django Framework

จากการศึกษาและการทดลองสร้างระบบนี้แสดงให้เห็นถึงประสิทธิภาพในการพัฒนาระบบของ Framework นี้ว่าสามารถทำได้อย่างรวดเร็วเพราะมีเครื่องมือที่ครบครันมาให้ในตัว จึงสามารถที่จะใช้ในงานที่แตกต่างกันได้อย่างมีประสิทธิภาพ

บทที่ 1

บทนำ

การศึกษาการพัฒนา Web Applications ด้วย Django Framework เพื่อให้เข้าใจคุณสมบัติหลัก และการทำงานของ Django Framework เพื่อจะได้นำ Django Framework มาประยุกต์ใช้ในการ พัฒนา Web Applications เพื่อจะได้แสดงถึงความสามารถและประสิทธิภาพของการใช้ Framework นี้ ในการพัฒนาระบบสำหรับจัดการ Blog

บทที่ 2

ทบทวนวรรณกรรม (Literature review)

2.1 ความรู้พื้นฐานและแนวคิดหลักในหัวข้อ

• คุณสมบัติหลักของ Django

☐ การพัฒนาอย่างรวดเร็ว: Django ส่งเสริมการเปลี่ยนแปลงโครงการอย่างรวดเร็ว ทำให้ นักพัฒนาสามารถเปิดตัวแอปพลิเคชันได้ในเวลาหลายชั่วโมงแทนที่จะเป็นวัน (Sufyan, 2022)

☐ ความปลอดภัยที่แข็งแกร่ง: ประกอบด้วยคุณสมบัติเช่นการตรวจสอบสิทธิ์ของผู้ใช้และการ ป้องกันช่องโหว่บนเว็บทั่วไป (Shubham Dhadil, 2024)

☐ ORM (Object-Relational Mapping): ลดความซับซ้อนในการโต้ตอบฐานข้อมูล ทำให้ นักพัฒนาสามารถทำงานกับวัตถุ Python แทนการสอบถาม SQL ที่ซับซ้อน (Songtao Chen, 2020)

☐ เทมเพลตและมุมมอง: ระบบเทมเพลตของ Django ช่วยให้ผู้พัฒนาสามารถแสดงผลเนื้อหาแบบไดนามิกช่วยเพิ่มประสบการณ์ของผู้ใช้ (M. Ramya, 2024)

2.2 กรณีศึกษา

☐ อินเทอร์เน็ตที่ใช้งานง่าย: การผสมรวมเฟรมเวิร์ก CSS และ JavaScript แบบโอเพนซอร์ส เช่น Bootstrap ช่วยให้ผู้พัฒนาสามารถสร้างเว็บไซต์ที่ตอบสนองและสวยงามน่าประทับใจ (Sanmukh Kaur, 2023) (Shubham Dhadil, 2024)

☐ การพัฒนาอย่างรวดเร็ว: การทำงานที่ซ้ำซ้ำโดยอัตโนมัติของ Django ช่วยให้นักพัฒนาสามารถมุ่งเน้นไปที่คุณสมบัติสร้างสรรค์ ซึ่งช่วยเร่งกระบวนการพัฒนาอย่างมีนัยสำคัญ (Bennett, 2008)

☐ คุณสมบัติความปลอดภัย: เฟรมเวิร์กประกอบด้วยมาตรการรักษาความปลอดภัยในตัว เช่น การเข้ารหัสข้อมูลและการรับรองความถูกต้องของผู้ใช้ ซึ่งมีความสำคัญต่อการปกป้องข้อมูลที่ละเอียดอ่อน (Manoj Kumar, 2024)

2.3 แนวทางการวิจัยและการพัฒนา

□ กระบวนการพัฒนาเว็บแอปพลิเคชันโดยใช้เฟรมเวิร์ก Django เกี่ยวข้องกับองค์ประกอบสำคัญหลายอย่างที่จะช่วยเพิ่มประสิทธิภาพความปลอดภัยและประสบการณ์ของผู้ใช้สถาปัตยกรรมของ Django ซึ่งอิงจากรูปแบบ Model-Template-View (MTV) อำนวยความสะดวกในแนวทางที่มีโครงสร้างในการพัฒนาเว็บ ช่วยให้สามารถแยกความกังวลได้อย่างชัดเจนและการจัดการโครงการที่คล่องตัว (Manoj Kumar, 2024)

2.4 ข้อมูลเชิงสถิติและเชิงปริมาณ

□ อัตราการใช้งาน Django ถูกนำมาใช้กันอย่างแพร่หลายในภาคส่วนต่าง ๆ รวมถึงการดูแลสุขภาพและการผลิตเพื่อความสามารถในการจัดการข้อมูลที่คล่องตัวและปรับปรุงปฏิสัมพันธ์กับผู้ใช้ (Shubham Dhadil, 2024) (Jenn-Yih, 2022)

2.5 การประยุกต์ใช้จริง

□ Django อำนวยความสะดวกในการพัฒนาแอปพลิเคชันที่ใช้งานง่าย เช่น แอปพลิเคชันสำหรับการลงทะเบียนผู้ใช้และการติดตามกิจกรรมซึ่งจำเป็นสำหรับความต้องการทางธุรกิจสมัยใหม่ (Sanmukh Kaur, 2023)

□ แอปพลิเคชันเว็บแบบทดสอบที่พัฒนาด้วย Django เป็นตัวอย่างของความสามารถในการสร้างแพลตฟอร์มแบบโต้ตอบ โดยมีการวิเคราะห์แบบเรียลไทม์และการตั้งค่าที่ปรับแต่งได้ตอบสนองความต้องการด้านการศึกษาและองค์กร (M. Ramya, 2024)

บทที่ 3

รายละเอียดการศึกษา

3.1 IntroductionและหลักการทำงานของDjango Framework

3.1.1Introduction to Django

Django เป็นเว็บเฟรมเวิร์กแบบ Full-Stack ที่พัฒนาด้วยภาษาPython

เปิดตัวครั้งแรกวันที่ 21 กรกฎาคม 2005 โดย Adrian Holovaty และ Simon Willison

3.1.2จุดเด่นของDjango

ความนิยมสูง Django คือเฟรมเวิร์กที่ได้รับความนิยมสูงสุดของ Python และในปีล่าสุดคือปี 2020 ครองอันดับ 1 back-end framework จาก stackshare.io

มีDocumentationที่ดี มีการอัปเดตและข้อมูลที่เกี่ยวข้องในการศึกษาอยู่ตลอดทำให้เป็นอีกหนึ่งจุดเด่นของDjango

มีแหล่งเรียนรู้เยอะ

เนื่องจากเป็นFrameworkที่ได้รับความนิยมสูงจึงทำให้มีการสอนการใช้งานต่าง ๆ มากมายบนแพลตฟอร์มต่าง ๆ

มีBuilt-in features

มีฟีเจอร์ Admin site และ Authenticationมาให้ทำให้ลดเวลาในการทำงานลงได้และเพิ่มความสะดวกสบายในการพัฒนาให้แก่ผู้พัฒนามากยิ่งขึ้น

ความปลอดภัยสูง

Djangoมีระบบที่ช่วยป้องกันการถูกโจมตีจากภายนอกเข้ามาในระบบ

3.1.3 Architecture Workflow

MVT Architecture

Model (M)

ใช้จัดการข้อมูลและการโต้ตอบกับฐานข้อมูลรับผิดชอบการกำหนดโครงสร้างข้อมูล เช่น ตารางในฐานข้อมูล

View (V)

จัดการและประมวลผลคำขอ (Request)

ดึงข้อมูลจาก Model และส่งข้อมูลไปยัง Template View ไม่ได้แสดงผลโดยตรงแต่ทำหน้าที่เป็น"ตัวกลาง"

Template (T)

ใช้สำหรับการแสดงผลข้อมูลในรูปแบบ HTML

Template Engine ของ Django รองรับการใช้ตัวแปรและฟังก์ชันที่เกี่ยวข้อง เช่น Loops, Conditions แยกส่วนการแสดงผลออกจากใน View

3.1.4 Key Features and Tools in Django

Django Admin Interface

ระบบจัดการหลังบ้านที่สร้างขึ้นอัตโนมัติจาก Model ใช้สำหรับจัดการข้อมูลในฐานข้อมูล เช่น เพิ่ม, ลบ, แก้ไข

Object-Relational Mapping (ORM)

ช่วยให้จัดการฐานข้อมูลโดยใช้ Python แทน SQL ช่วยลดความซับซ้อนของการจัดการข้อมูลในฐานข้อมูล และป้องกัน SQL Injection

Authentication and Authorization

ระบบจัดการผู้ใช้และการตรวจสอบสิทธิ์ (User Authentication) รองรับการสร้างระบบล็อกอิน, สัมผัสสมาชิก, และเปลี่ยนรหัสผ่าน

Template Engine

รองรับการเขียน Template ด้วย HTML + Django Tags ใช้ตัวแปรและโครงสร้างควบคุม (เช่น Loops, Conditions) ในไฟล์ HTML ได้

URL Routing

ระบบแมป URL กับ View Function เพื่อกำหนดเส้นทางที่รองรับคำขอ รองรับการใช้

Regular Expressions หรือ Path Converters

3.2 การติดตั้งและการเริ่มต้นใช้งาน Django Framework

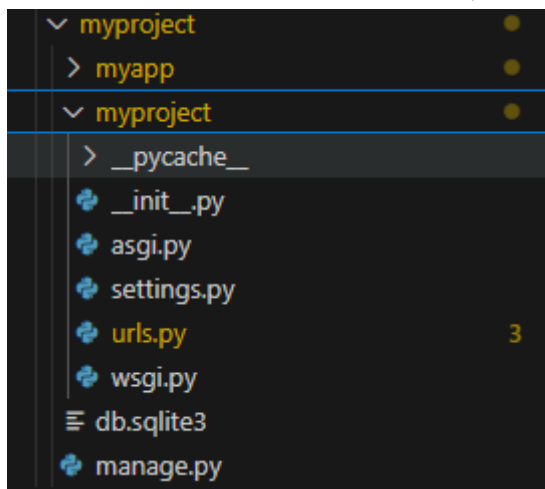
สร้าง Project

สร้างโปรเจกต์ `django-admin startproject` ตามด้วยชื่อโปรเจกต์คือ `myproject`

Django ใช้โครงสร้างไดเรกทอรีเพื่อจัดระเบียบส่วนต่าง ๆ ของเว็บแอปพลิเคชัน โดยจะสร้างโฟลเดอร์โปรเจกต์และแอปสำหรับการจัดการส่วนประกอบเหล่านี้

เมื่อเราสร้างโปรเจกต์ Django ตัว Django จะสร้างไดเรกทอรีหลักของโปรเจกต์โดยใช้ชื่อโปรเจกต์ที่เรากำหนดไว้ ภายในไดเรกทอรีนี้จะมีไฟล์ที่จำเป็นสำหรับการให้ฟังก์ชันพื้นฐานแก่เว็บแอปพลิเคชันของเรา

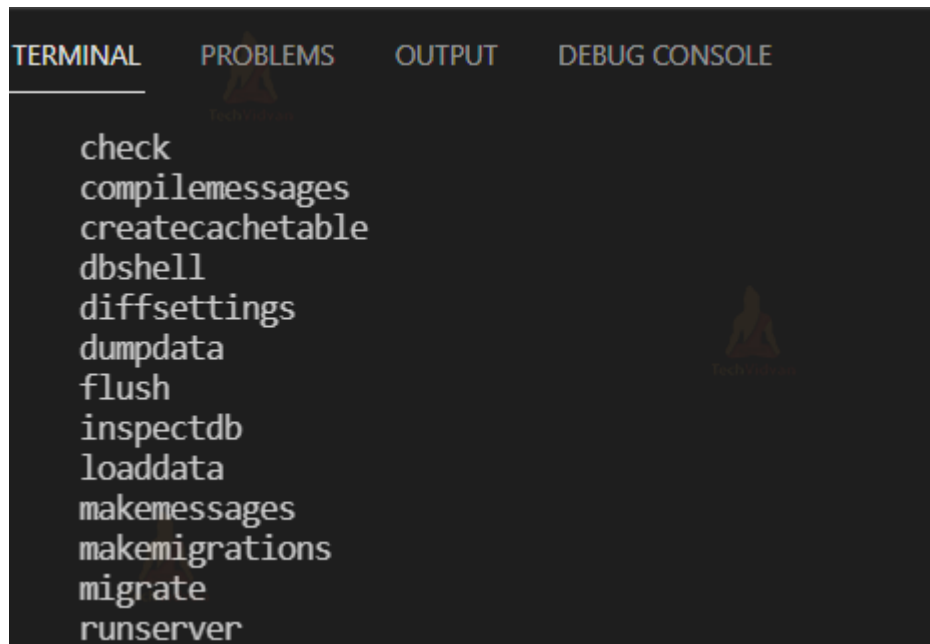
โดยหลังจากที่สร้าง project จะมีไฟล์ต่าง ๆ ดังนี้



ภาพที่ 3.2.1 สร้าง Project

`manage.py`

ไฟล์นี้ทำหน้าที่เป็นเครื่องมือสำหรับการจัดการโปรเจกต์ Django ของเราโดยใช้คำสั่งผ่านบรรทัดคำสั่ง เช่น การดีบั๊ก (debugging) การดีพลอย (deploying) และการรัน (running) เว็บแอปพลิเคชัน

A screenshot of a terminal window with a dark background. At the top, there are four tabs: 'TERMINAL', 'PROBLEMS', 'OUTPUT', and 'DEBUG CONSOLE'. The 'TERMINAL' tab is active. Below the tabs, a list of Django management commands is displayed in a light-colored font. The commands are: check, compilemessages, createcachetable, dbshell, diffsettings, dumpdata, flush, inspectdb, loaddata, makemessages, makemigrations, migrate, and runserver. A small, faint logo is visible in the background of the terminal window.

```
check
compilemessages
createcachetable
dbshell
diffsettings
dumpdata
flush
inspectdb
loaddata
makemessages
makemigrations
migrate
runserver
```

ภาพที่3.2.2 คำสั่งที่สามารถใช้ได้ด้วยmanage.py

คำสั่งสำคัญใน manage.py

Runserverคำสั่งนี้ใช้สำหรับเริ่มต้นเซิร์ฟเวอร์ทดสอบที่ Django มีให้สำหรับเว็บแอปพลิเคชัน Makemigrationsใช้สำหรับสร้าง migrations ใหม่ในโปรเจกต์และแอป เพื่อรองรับการเปลี่ยนแปลงในฐานข้อมูล Migrateเป็นขั้นตอนที่ตามมาหลังจากคำสั่ง makemigrations ใช้สำหรับปรับเปลี่ยนโมดูลต่าง ๆ ในฐานข้อมูลให้ตรงกับ migrations

ไฟล์ต่อมาในprojectจะมี

init.py

ไฟล์ว่างที่บอก Python ว่าไดเรกทอรีนี้เป็นแพ็คเกจ Python

asgi.py

ใช้สำหรับการทำงานร่วมกับ ASGI (Asynchronous(เอซิงโครนัส) Server Gateway Interface) เพื่อรองรับโปรโตคอลแบบ asynchronous

settings.py

ไฟล์ที่ใช้สำหรับการกำหนดค่าโปรเจกต์ เช่น การตั้งค่าแอป การเชื่อมต่อฐานข้อมูล และ static files

urls.py

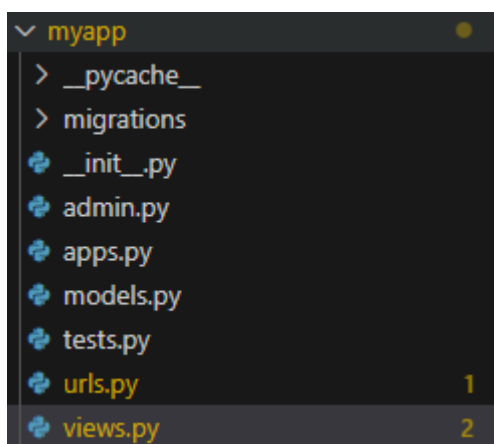
ใช้สำหรับกำหนดเส้นทาง URL (URL routing) ไปยัง views ต่าง ๆ ในโปรเจกต์

wsgi.py

ใช้สำหรับการทำงานร่วมกับ WSGI (Web Server Gateway Interface) ซึ่งเป็นมาตรฐานในการรันเว็บแอปพลิเคชัน Python

settings.py และ urls.py มักจะถูกใช้งานบ่อยที่สุดในระหว่างการพัฒนา
wsgi.py และ asgi.py จะมีบทบาทในขั้นตอนการโฮสต์เว็บแอปพลิเคชัน

สร้างแอปโดยใช้คำสั่ง `python manage.py startapp` ตามด้วยชื่อแอป `myapp`



ภาพที่3.2.3 ไฟล์ต่างๆในapp

__init__.py

ไฟล์นี้ทำหน้าที่เหมือนกับไฟล์ `__init__.py` ในโครงสร้างของโปรเจกต์ Django

admin.py

หน้าที่หลักในการกำหนดและปรับแต่งการจัดการข้อมูลผ่าน Django Admin Interface ซึ่งเป็นเครื่องมือจัดการข้อมูลที่ Django สร้างมาให้พร้อมใช้งานโดยอัตโนมัติสำหรับแอดมินหรือผู้พัฒนาระบบ

apps.py

ไฟล์นี้ช่วยให้ผู้ใช้สามารถกำหนดค่าคอนฟิกของแอปได้

ผู้ใช้สามารถปรับแต่งแอตทริบิวต์ของแอปพลิเคชันด้วยไฟล์ `apps.py`

models.py

ไฟล์นี้ใช้ในการกำหนด โมเดล ของเว็บแอปพลิเคชันในรูปแบบคลาส

ถือเป็นส่วนที่สำคัญที่สุดในโครงสร้างไฟล์แอป

โมเดล กำหนดโครงสร้างของฐานข้อมูลและความสัมพันธ์ระหว่างข้อมูลต่าง ๆ รวมถึง

ข้อกำหนดของแอตทริบิวต์

views.py

มีหน้าที่หลักในการจัดการการร้องขอ (request) จากผู้ใช้และส่งผลลัพธ์ (response) กลับไป

ยังผู้ใช้ โดยมีบทบาทเป็นตัวกลางที่เชื่อมระหว่าง โมเดล (Model) และ เทมเพลต (Template)

หรือให้บริการ API หากเป็นระบบ backend

urls.py

ไฟล์ urls.py ทำงานคล้ายกับไฟล์ urls.py ในโครงสร้างของโปรเจกต์

หน้าที่หลักคือการเชื่อมโยง URL ที่ผู้ใช้ร้องขอกับหน้าหรือทรัพยากรที่ต้องการ

ไฟล์นี้จะไม่ได้อยู่ในไฟล์แอปโดยตรง แต่จะต้องสร้างใหม่ด้วยการคลิกที่ New file หลังจาก

โปรเจกต์ชื่อ

tests.py

ไฟล์ tests.py ใช้สำหรับเขียน เทส (tests) เพื่อทดสอบการทำงานของแอปพลิเคชัน

ใน Django โครงการ (Project) และแอปพลิเคชัน (Application) เป็นแนวคิดสำคัญทั้งคู่ แต่มีวัตถุประสงค์ที่แตกต่างกัน

Django Project

Project คือ โครงสร้างหลักของโปรเจกต์ทั้งหมด ซึ่งครอบคลุมทุกองค์ประกอบที่เกี่ยวข้องกับเว็บแอปพลิเคชัน เช่น การตั้งค่า, การจัดการ URL, การกำหนดฐานข้อมูล และการรวมแอปพลิเคชันต่าง ๆ เข้าไว้ด้วยกัน

Django Application

Application คือ ส่วนประกอบย่อย ที่มีฟังก์ชันเฉพาะเจาะจงในโปรเจกต์มีลักษณะสำคัญคือแต่ละแอปพลิเคชันมีความเป็นอิสระโดยมีไฟล์และฟังก์ชันการทำงานของตัวเองและสามารถนำกลับมาใช้ซ้ำในโปรเจกต์อื่นได้

ยกตัวอย่างการใช้งานเบื้องต้นการURL Routing

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include("myapp.urls"))
]
```

ภาพที่3.2.4 urls.py/myproject

```
from django.contrib import admin
from django.urls import path, include

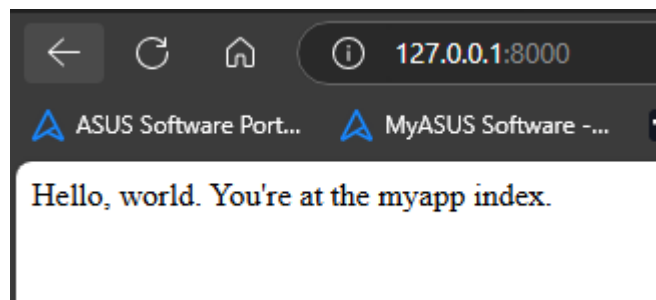
urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include("myapp.urls"))
]
```

ภาพที่3.2.5 urls.py/myapp

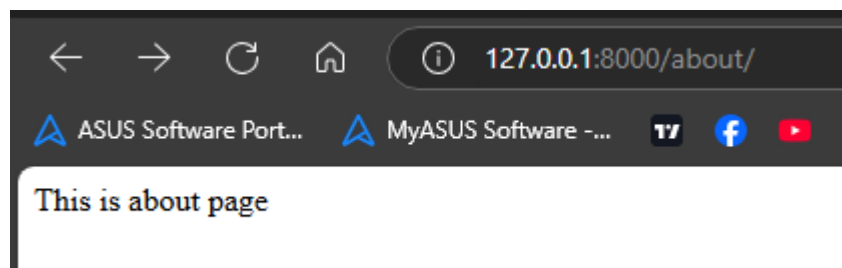
```
from django.shortcuts import render
from django.http import HttpResponse
# Create your views here.
def index(request):
    return HttpResponse("Hello, world. You're at the myapp index.")

def about(request):
    return HttpResponse("This is about page")
```

ภาพที่3.2.6 views.py



ภาพที่3.2.7 127.0.0.1:8000



ภาพที่3.2.8 127.0.0.1:8000/about/

3.3การเชื่อมต่อกับฐานข้อมูลในDjango

ORMคืออะไร

ORM (Object Relational Mapping) คือ การเขียนPythonเพื่อติดต่อกับฐานข้อมูลโดยไม่จำเป็นต้องเขียนภาษา SQL แบบปกติทั่วไป ซึ่งเราสามารถเขียนRaw SQL ใน Djangoได้ แต่การใช้ORMเป็นมาตรฐานที่นิยมใช้กัน ส่วนRaw SQLจะใช้กันในตอนที่Method ของ Django มีข้อจำกัด ในกรณีที่ต้องการcustom queryที่มีความซับซ้อน

ID	Title	Body
1	Django 101	Django is
2	Basic Python	Learning Python ...
3	Python OOP	OOP is a core concept ...

ภาพที่ 3.3.1 ตารางฐานข้อมูลตัวอย่าง

```
CREATE TABLE "post" (  
    "id" integer NOT NULL PRIMARY KEY AUTOINCREMENT,  
    "title" varchar(80) NOT NULL,  
    "body" text NOT NULL,  
)
```

ภาพที่ 3.3.2 ภาษาSQLสำหรับการสร้างฐานข้อมูล

```
class Post(models.Model):  
    title = models.CharField(max_length=80)  
    body = models.TextField()
```

ภาพที่ 3.3.3 Django Model สำหรับการสร้างฐานข้อมูล

ข้อดีข้อเสียของการใช้ORM

ข้อดี

เพิ่มประสิทธิภาพในการพัฒนา: ช่วยในการทำงานและลดข้อผิดพลาดในการเขียนSQL

ความยืดหยุ่นในการเปลี่ยนฐานข้อมูล: การเปลี่ยนแปลงฐานข้อมูล (เช่น จาก MySQL ไป PostgreSQL) ทำได้ง่ายขึ้น เนื่องจากไม่ต้องเปลี่ยนคำสั่ง SQL

การจัดการ Transaction: ORM มีระบบจัดการ transaction ที่ช่วยให้สามารถ commit หรือ rollback การแก้ไขข้อมูลได้สะดวก

ข้อเสีย

ประสิทธิภาพต่ำกว่า SQL ที่เขียนเอง: ในบางกรณี การใช้ ORM อาจมีประสิทธิภาพต่ำกว่าเมื่อเปรียบเทียบกับ การเขียน SQL โดยตรง

ไม่สามารถเข้าถึงฐานข้อมูลโดยตรง: การใช้ ORM อาจทำให้ไม่สามารถเข้าถึงฐานข้อมูลได้โดยตรง ซึ่งอาจเป็นปัญหาสำหรับผู้ที่ต้องการควบคุมหรือปรับแต่ง query อย่างละเอียด

Django Modelคืออะไร

Django Model คือส่วนที่ติดต่อกับฐานข้อมูลหรือส่วนหลังบ้าน โดยตามหลัก Django MVT Design Patterns โดย M

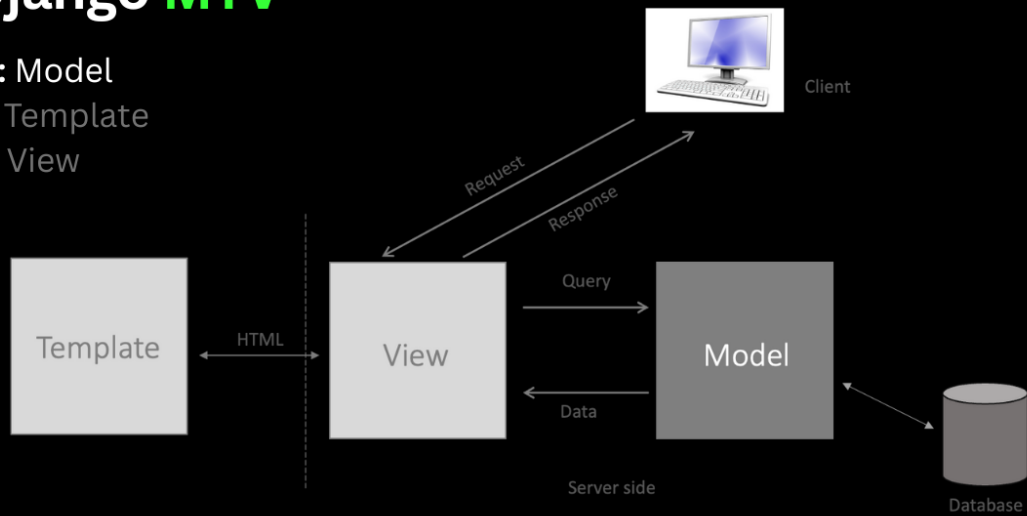
หรือ Model จะเป็นส่วนประสานกับDatabaseจะมีview เป็นตัวกลางสำหรับการ query ข้อมูลจาก Model ไป response ที่ Client

Django MTV

M: Model

T: Template

V: View



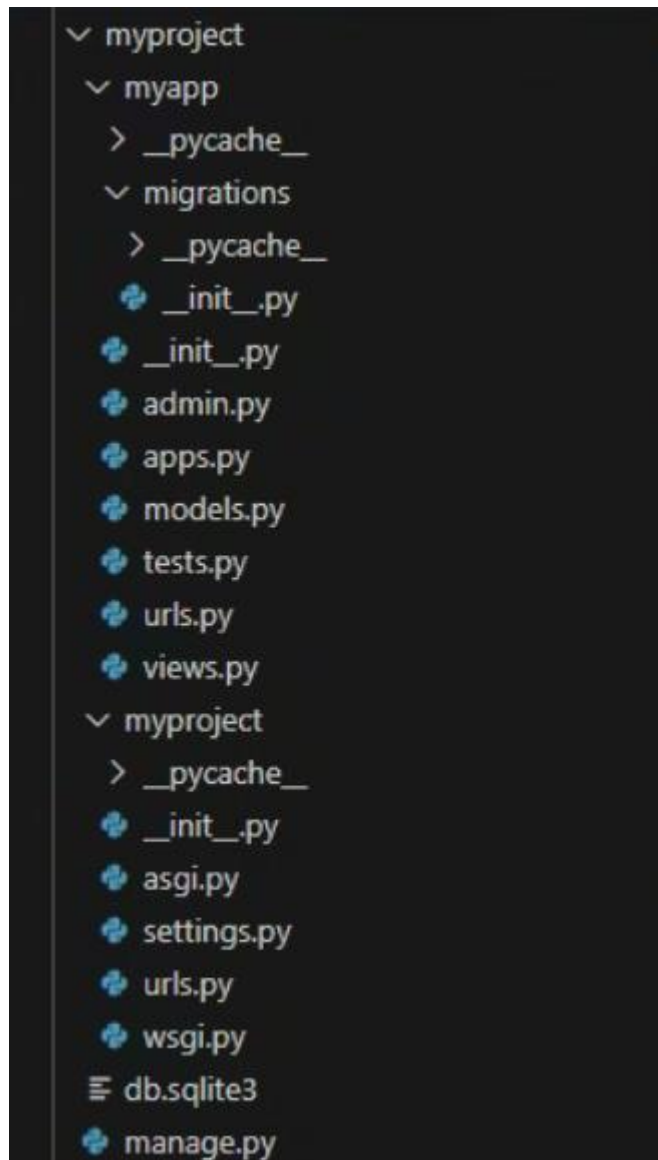
django

DE

ภาพที่3.3.4 Django MTV M:Model

การสร้างModelและการจัดการข้อมูลในModel

เริ่มจากสร้างโปรเจ็คและappก็จะได้ไฟล์settings.py models.py



ภาพที่3.3.5 โครงสร้างของDjango Project

```
DATABASES = {  
    'default': {  
        'ENGINE': 'django.db.backends.sqlite3',  
        'NAME': BASE_DIR / 'db.sqlite3',  
    }  
}
```

ภาพที่3.3.6 ทำการตั้งค่าการเชื่อมต่อกับฐานข้อมูลในไฟล์Settings.py โดยในส่วนนี้จะใช้ SQLite

ในไฟล์Models.pyจะเป็นการกำหนดโครงสร้างตารางที่จะเก็บในฐานข้อมูลโดยการสร้าง Class

```
1  from django.db import models
2
3  # Create your models here.
4  class MyModel(models.Model):
5      name = models.CharField(max_length=100)
6      description = models.TextField()
7      created_at = models.DateTimeField(auto_now_add=True)
```

ภาพที่3.3.7 โครงสร้างในไฟล์ Model.py

หลังจากนั้นทำการMigrationsโดยใช้คำสั่ง python manage.py makemigrations เป็นการสร้างmigrationที่เก็บโครงสร้างตารางmodelเมื่อสร้างเสร็จในไฟล์เตอร์migrationsจะมีการสร้างไฟล์ที่นำตัวmodelที่เราสร้างมาใช้งาน

```
▼ migrations
  > __pycache__
  • __init__.py
  • 0001_initial.py
```

ภาพที่3.3.8 ไฟล์ที่ได้จากการ makemigrations

```
from django.db import migrations, models

▼ class Migration(migrations.Migration):
    initial = True

    ▼ dependencies = [
    ]

    ▼ operations = [
    ▼     migrations.CreateModel(
    ▼         name='MyModel',
    ▼         fields=[
    ▼             ('id', models.BigAutoField(auto_created=True, primary_key=True, serialize=False, verbose_name='id')),
    ▼             ('name', models.CharField(max_length=100)),
    ▼             ('description', models.TextField()),
    ▼             ('created_at', models.DateTimeField(auto_now_add=True)),
    ▼         ],
    ▼     ),
    ]
```

ภาพที่3.3.9 โครงสร้างภายในของไฟล์ที่ได้จากการ makemigrations

ต่อมาใช้คำสั่ง `python manage.py migrate` เพื่อทำการนำตารางหรือไฟล์migrationเข้าไปในฐานข้อมูล

การจะจัดการกับmodelในฐานข้อมูลจะใช้Admin Panelในการจัดการโดยเริ่มจากการสร้างsuperuserในการที่จะเข้าไปสู่หน้าAdmin Panel

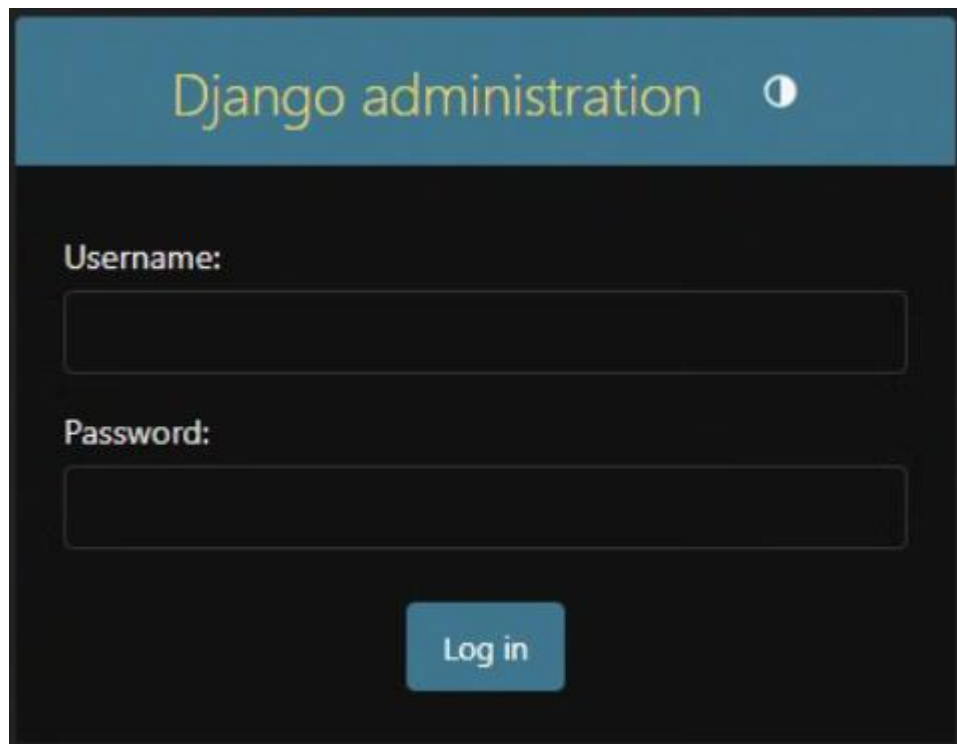
```
C:\Users\ASUS\Desktop\SE492\myproject>python manage.py createsuperuser
Username (leave blank to use 'asus'): admin
Email address: admin@gmail.com
Password:
Password (again):
This password is too short. It must contain at least 8 characters.
This password is too common.
This password is entirely numeric.
Bypass password validation and create user anyway? [y/N]: y
Superuser created successfully.
```

ภาพ3.3.10 การสร้าง Superuser

หลังจากสร้างเสร็จทำการrunserverและเข้าสู่หน้าadmin

```
127.0.0.1:8000/admin/
```

ภาพ3.3.11 URLของหน้าadmin



ภาพที่3.3.12 หน้าของAdmin page

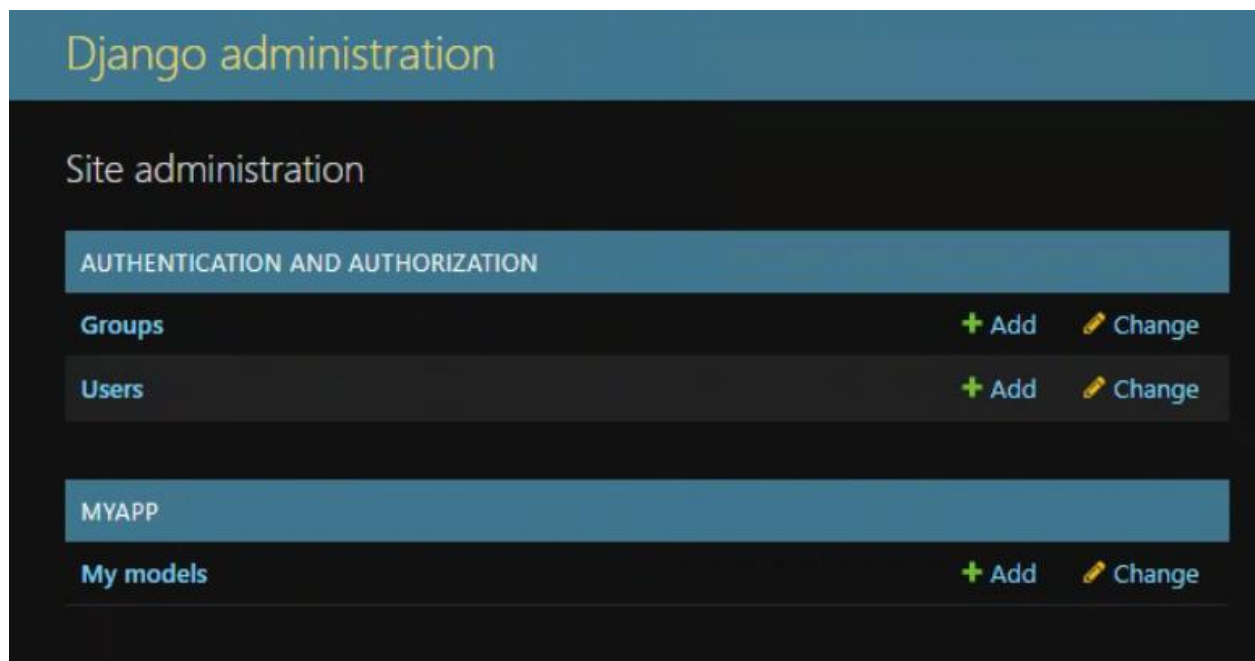
```

SE492 > myproject > myapp > admin.py
1  from django.contrib import admin
2  from myapp.models import MyModel
3
4  # Register your models here.
5  admin.site.register(MyModel)

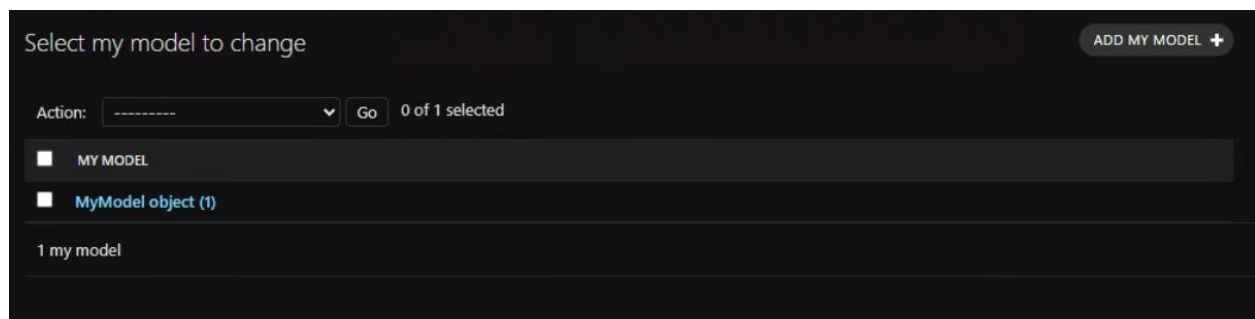
```

ภาพที่3.3.13 ภายในไฟล์admin.py

นำmodel mymodel มาใส่ในหน้าadminเพื่อให้สามารถจัดการในหน้าdashboardได้



ภาพที่3.3.14 UIหลังจากloginเสร็จของadminpage



ภาพที่3.3.15 ข้อมูลภายในของTable Mymodels

Add my model

Name:

Description:

ภาพที่3.3.16การเพิ่มข้อมูลภายในฐานข้อมูล

Select my model to change

Action: 0 of 2 selected

- ☐ MY MODEL
- ☐ MyModel object (2)
- ☐ MyModel object (1)

2 my models

ภาพที่3.3.17ข้อมูลที่ถูกเพิ่ม

Change my model

MyModel object (1) HISTORY

Name:

Description:

Hello world

ภาพที่3.3.18 การจัดการข้อมูลในฐานข้อมูล

3.4การใช้งาน Django REST Framework (DRF) สำหรับการสร้าง API

แนะนำ Django REST Framework

Django Rest Frameworkคืออะไร

Django REST Framework (DRF)คือToolkitหรือไลบรารีของPythonในการสร้างRestful APIsได้อย่างรวดเร็วและสมบูรณ์โดยDRF นั้นอยู่บนพื้นฐานของ Django Framework รูปแบบการเขียน การขึ้นโปรเจกต์ในช่วงแรกนั้นมีความคล้ายคลึงกันค่อนข้างมาก

Rest และ Restful API คืออะไร

Representational State Transfer (REST) เป็นสถาปัตยกรรมซอฟต์แวร์ที่กำหนดเงื่อนไขว่า API ควรทำงานอย่างไร โดย API ที่เป็นไปตามรูปแบบสถาปัตยกรรม REST เรียกว่า REST API หรือ RESTful API

RESTful API (Representational State Transfer API) คืออินเทอร์เฟซที่ช่วยให้ระบบคอมพิวเตอร์สองระบบสามารถแลกเปลี่ยนข้อมูลกันผ่านโปรโตคอล HTTP ได้อย่างมีประสิทธิภาพและยืดหยุ่น โดย RESTful API เป็นการออกแบบ API ตามหลักการของสถาปัตยกรรม REST ซึ่งเน้นความเรียบง่ายและความสามารถในการปรับขยาย.

ข้อดีของ Restful API

1.ความเรียบง่ายและใช้งานง่าย (Simplicity and Ease of Use)

REST APIs เข้าใจและใช้งานได้ง่าย เพราะใช้ HTTP Methods มาตรฐาน

ใช้รูปแบบการแสดงผลข้อมูลมาตรฐาน เช่น JSON หรือ XML ซึ่งช่วยให้นักพัฒนาทำงานได้ง่ายขึ้น

2.การปรับขนาดที่ดี (Scalability)

RESTful APIs สามารถ ขยายขนาดแนวนอน (Horizontally Scalable)[การเพิ่มจำนวนเซิร์ฟเวอร์หลายเครื่องเพื่อรองรับปริมาณงานที่เพิ่มขึ้น แทนที่จะเพิ่มพลังประมวลผลของเซิร์ฟเวอร์เครื่องเดียว] ได้ง่าย เนื่องจากเป็น Stateless คำขอแต่ละครั้งมีข้อมูลที่จำเป็นทั้งหมด ทำให้สามารถกระจายโหลดและรองรับคำขอจำนวนมากได้ดี

3.ความยืดหยุ่น (Flexibility)

REST รองรับหลายรูปแบบของข้อมูล แต่ JSON เป็นที่นิยมที่สุดเพราะอ่านง่ายและประมวลผลได้รวดเร็ว ความยืดหยุ่นนี้ช่วยให้ REST APIs ใช้งานได้กับแอปพลิเคชันและอุปกรณ์ต่าง ๆ

4.ไม่มีสถานะ (Statelessness)

เซิร์ฟเวอร์ไม่ต้องเก็บข้อมูลเกี่ยวกับสถานะของลูกค้า ช่วยให้การออกแบบและพัฒนาระบบง่ายขึ้น

5.ความสามารถในการทำงานร่วมกัน (Interoperability)

REST APIs ทำงานได้บนทุกแพลตฟอร์ม และสามารถพัฒนาได้ด้วยภาษาโปรแกรมใด ๆ สามารถเรียกใช้ REST APIs จากเทคโนโลยีที่แตกต่างกัน ซึ่งช่วยเพิ่มความสามารถในการทำงานร่วมกันของระบบ

6.รองรับการแคช (Cacheability)

REST รองรับ กลไกการแคชซึ่งช่วยให้ลูกค้าสามารถเก็บข้อมูลที่ร้องขอบ่อย ๆ ไว้ในแคช การแคชช่วย ลดภาระของเซิร์ฟเวอร์ และปรับปรุงประสิทธิภาพโดยเฉพาะกับข้อมูลที่ไม่เปลี่ยนแปลงบ่อย

7.อินเทอร์เฟซที่เป็นมาตรฐาน (Uniform Interface)

RESTful APIs มีโครงสร้างที่เป็นมาตรฐาน เช่น รูปแบบ URL, HTTP Methods และ Data Representationช่วยให้นักพัฒนาเรียนรู้และทำงานกับ REST APIs ได้ง่ายขึ้น

8.ลดความหน่วงเวลา (Reduced Latency)

เนื่องจาก REST API เป็น Stateless เซิร์ฟเวอร์ไม่ต้องเก็บสถานะของลูกค้า ทำให้ลดเวลาในการประมวลผลคำขอ ลูกค้าสามารถส่งข้อมูลทั้งหมดที่จำเป็นมาในคำขอเดียว และเซิร์ฟเวอร์สามารถตอบกลับได้อย่างรวดเร็ว

9.ง่ายต่อการรวมระบบ (Ease of Integration)

RESTful APIs สามารถเชื่อมต่อกับระบบอื่น ๆ ได้ง่าย กระบวนการพัฒนาระบบร่วมกับ REST APIs ค่อนข้างตรงไปตรงมา และสามารถใช้ได้กับแอปพลิเคชันที่มีอยู่แล้ว

10.ความปลอดภัย (Security)

REST APIs สามารถเข้ารหัสด้วย HTTPS เพื่อสร้างช่องทางการสื่อสารที่ปลอดภัยระหว่างลูกค้าและเซิร์ฟเวอร์

เปรียบเทียบDjango และ DRF

วัตถุประสงค์หลัก

Django: เป็นเว็บเฟรมเวิร์กที่ใช้สำหรับการพัฒนาแอปพลิเคชันเว็บทั่วไป โดยมีฟีเจอร์ครบถ้วนสำหรับการจัดการฐานข้อมูล, การสร้างฟอร์ม, การจัดการผู้ใช้, และการสร้างหน้าเว็บ.

Django REST Framework (DRF): เป็นไลบรารีที่สร้างขึ้นบน Django เพื่อช่วยในการพัฒนา RESTful APIs โดยเฉพาะ DRF ทำให้การสร้าง API ง่ายขึ้นและมีประสิทธิภาพมากขึ้น.

การใช้งาน

Django: สามารถใช้สร้างทั้งส่วนหน้า (frontend) และส่วนหลัง (backend) ของเว็บแอปพลิเคชัน แต่เมื่อสร้าง API ด้วย Django จะต้องเขียนโค้ดมากขึ้นและทำการออกแบบเองมากกว่า.

DRF: ช่วยให้สามารถสร้าง API ได้ง่ายและรวดเร็ว โดยมีเครื่องมือและฟีเจอร์เฉพาะสำหรับ API เช่น การ serialization, authentication, และ viewsets ที่ช่วยลดปริมาณโค้ดที่ต้องเขียน.

ฟีเจอร์

Django: มีฟีเจอร์ที่หลากหลายสำหรับการพัฒนาเว็บแอปพลิเคชัน เช่น Django ORM, ระบบจัดการผู้ใช้, และ admin interface.

DRF: มีฟีเจอร์เฉพาะสำหรับ API เช่น serializers ที่ช่วยในการแปลงข้อมูลระหว่าง Python objects กับ JSON, ระบบ authentication ที่หลากหลาย, browsable API.

ความยืดหยุ่นและความซับซ้อน

Django: มีความยืดหยุ่นสูงในการพัฒนาเว็บแอปพลิเคชัน แต่บางครั้งอาจต้องใช้โค้ดจำนวนมากในการสร้าง API.

DRF: แม้จะมีฟีเจอร์ที่หลากหลาย แต่ก็มีความซับซ้อนในการตั้งค่าและเรียนรู้มากกว่า Django เนื่องจากต้องเข้าใจแนวทาง RESTful.

ประสิทธิภาพ

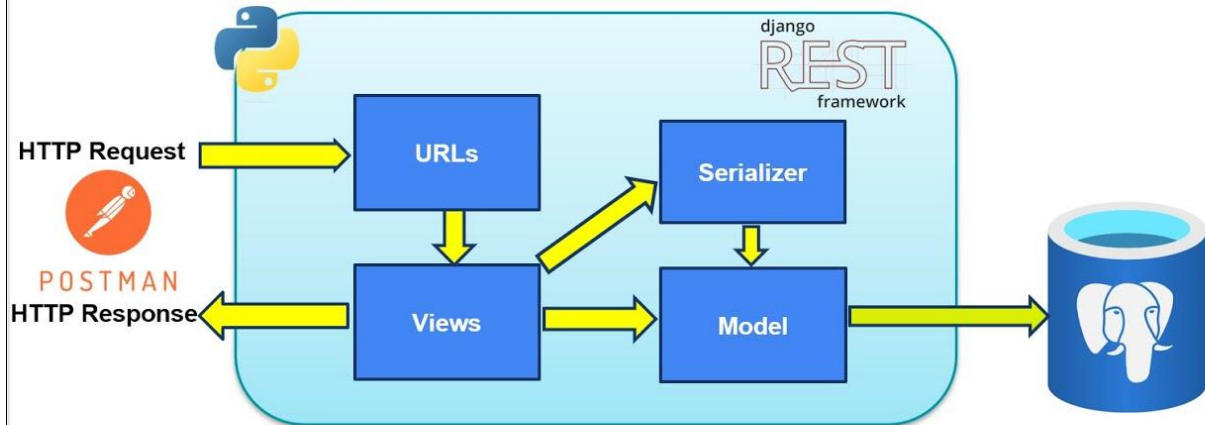
Django: ประสิทธิภาพดีสำหรับการพัฒนาเว็บแอปพลิเคชันทั่วไป แต่เมื่อเปรียบเทียบกับ DRF อาจไม่เหมาะสมที่สุดสำหรับการสร้าง API.

DRF: ออกแบบมาเพื่อให้มีประสิทธิภาพสูงในการจัดการกับคำขอ API โดยเฉพาะเมื่อทำงานกับข้อมูลขนาดใหญ่หรือซับซ้อน.

การทำงานของ Django Rest Framework

Django REST Framework (DRF) ทำงานโดยใช้หลักการของ RESTful API เพื่อสร้างและจัดการ API ที่สามารถสื่อสารกับไคลเอนต์

CRUD operations APIs with Django REST Framework



ภาพที่3.4.1การทำงานของAPIใน Django Rest Framework

การติดตั้งและใช้งานDjango REST Framework

ติดตั้ง Django REST Framework

`pip install django djangorestframework`

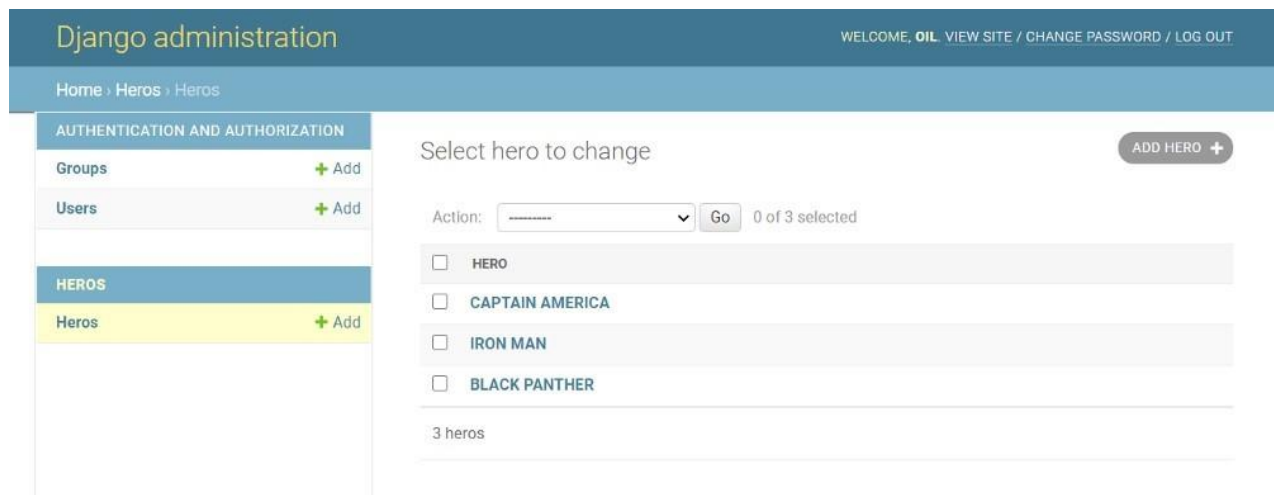
เพิ่ม `rest_framework` ลงใน `INSTALLED_APPS` ในไฟล์ `settings.py`

```
from django.db import models

class Hero(models.Model):
    name = models.CharField(max_length=100)
    description = models.TextField()
```

ภาพที่3.4.2 สร้างmodel

ทำการMigrateข้อมูลเข้าสู่Database



ภาพที่3.4.3 เพิ่มข้อมูลลงสู่ฐานข้อมูล

```
from rest_framework import serializers
from .models import Hero

class HeroSerializer(serializers.ModelSerializer):
    class Meta:
        model = Hero
        fields = ('id', 'name', 'description')
```

ภาพที่3.4.4 สร้างSerializer สำหรับแปลงข้อมูลจากPython object ไปเป็น JSON

```
# heros/views.py
from rest_framework import generics
from .models import Hero
from .serializers import HeroSerializer

class ListHero(generics.ListCreateAPIView):
    queryset = Hero.objects.all()
    serializer_class = HeroSerializer

class DetailHero(generics.RetrieveUpdateDestroyAPIView):
    queryset = Hero.objects.all()
    serializer_class = HeroSerializer
```

ภาพที่3.4.5 สร้างviewที่จะแสดงข้อมูล serializers

```
# heros/views.py
from rest_framework import generics
from .models import Hero
from .serializers import HeroSerializer

class ListHero(generics.ListCreateAPIView):
    queryset = Hero.objects.all()
    serializer_class = HeroSerializer

class DetailHero(generics.RetrieveUpdateDestroyAPIView):
    queryset = Hero.objects.all()
    serializer_class = HeroSerializer
```

ภาพที่ 3.4.6 กำหนด path ในไฟล์ urls.py ในไฟล์เตอร์ project และ app

```
from django.contrib import admin
from django.urls import include, path
urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/', include('heros.urls')),
]
```

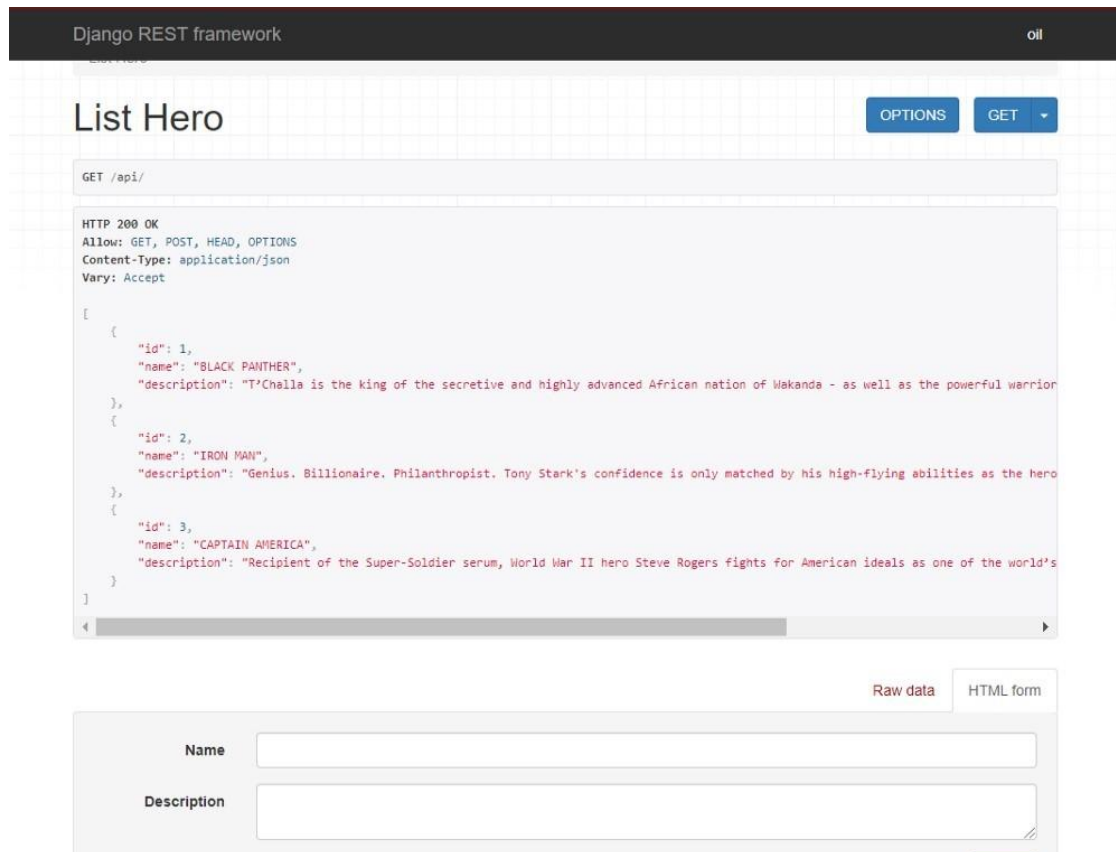
ภาพที่ 3.4.7 Urls.py (Project)

```
# heros/urls.py
from django.urls import path
from . import views
urlpatterns = [
    path('', views.ListHero.as_view()),
    path('<int:pk>/', views.DetailHero.as_view()),
]
```

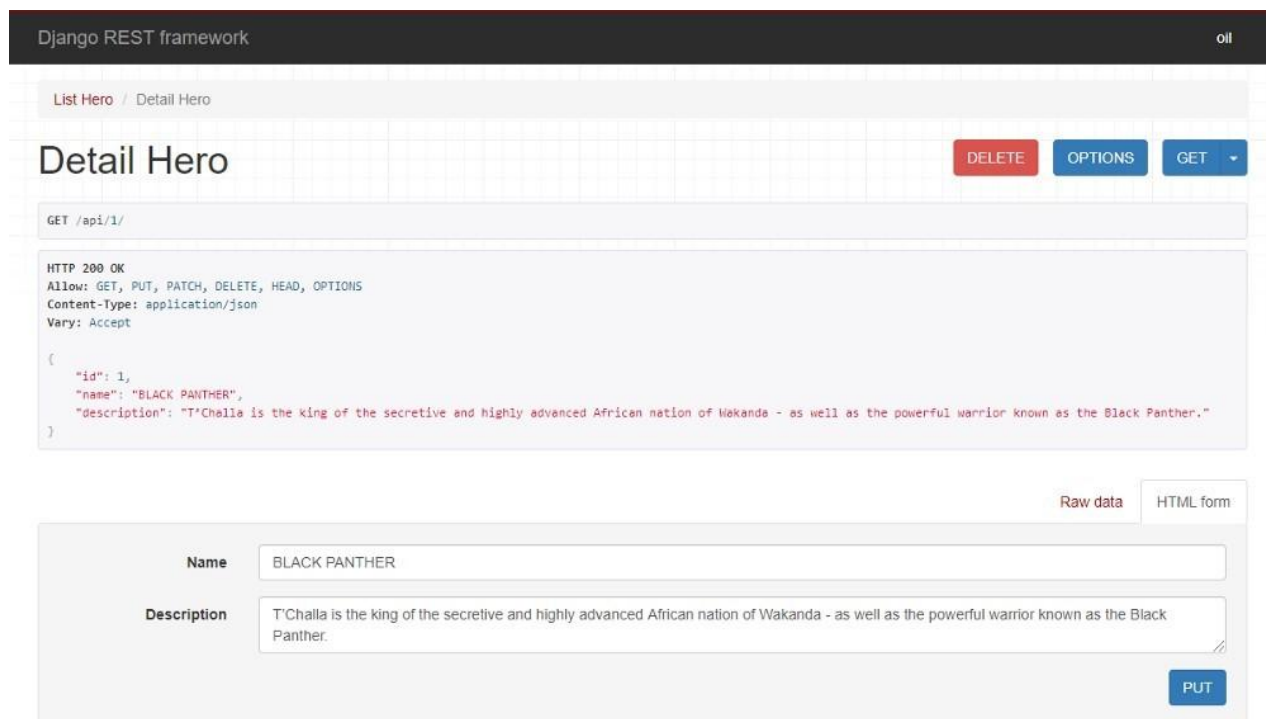
ภาพที่ 3.4.8 Urls.py (App)

Int:pk เป็นการกำหนด primary key ของข้อมูลในฐานข้อมูล ซึ่งเป็นตัวระบุเฉพาะของแต่ละ object ในฐานข้อมูล Django

เมื่อตั้งค่าทุกอย่างเสร็จแล้วทำการทดสอบโดยใช้ Browsable API คือ อินเทอร์เฟซที่ DRF ให้มาโดยอัตโนมัติเมื่อเราสร้าง API ช่วยให้เราสามารถ ทดสอบ API ได้ง่ายผ่านเว็บเบราว์เซอร์ โดยไม่ต้องใช้เครื่องมือภายนอก



ภาพที่3.4.9การรันserverและไปที่ลิ้งค์ <http://127.0.0.1:8000/api/>เพื่อดูผลลัพธ์

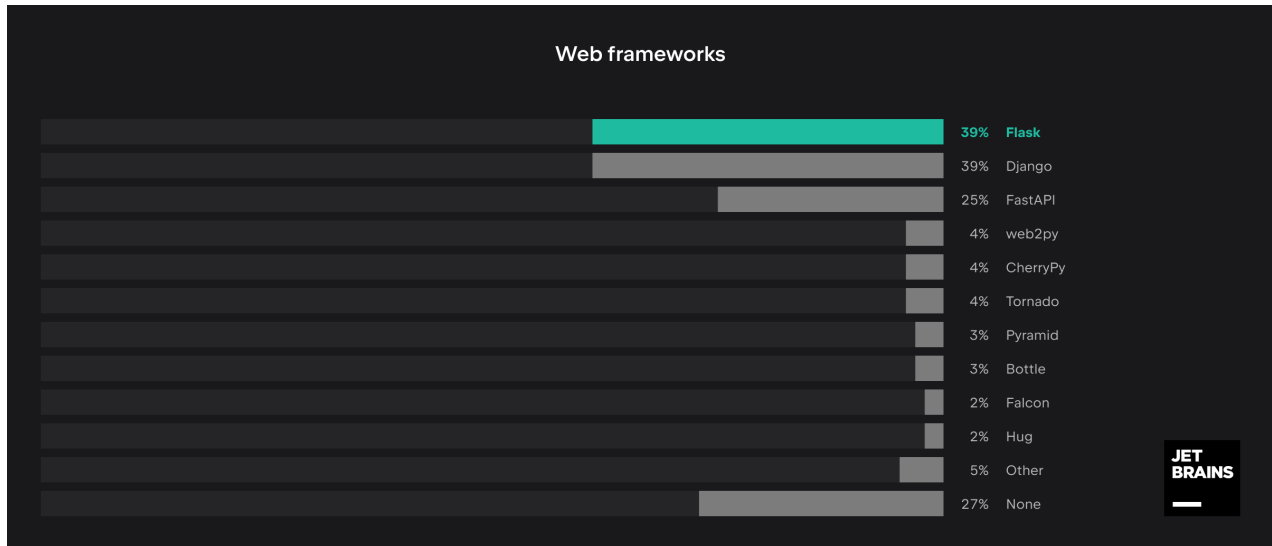


ภาพที่3.4.10 ถ้าเปลี่ยนลิ้งค์เป็น <http://127.0.0.1:8000/api/1>จะแสดงผลดังนี้

3.5เปรียบเทียบ Django & Flask

Django&Flask

DjangoและFlaskเป็นFrameworkในภาษาpythonที่ได้รับความนิยมมากที่สุดจากเว็บ JetBrainsในหัวข้อPython Developers Survey 2022 นักพัฒนา39%มีการใช้งาน FrameworkใดFrameworkหนึ่งหรือทั้ง2Framework



ภาพที่3.5.1 เปรียบเทียบผู้ใช้Django และ Flask

ภาพจาก(<https://blog.jetbrains.com/pycharm/2023/11/django-vs-flask-which-is-the-best-python-web-framework/#what-is-django>)

จุดเด่นและด้อยของDjangoและFlask

Django

จุดเด่น

โครงสร้างของprojectมีระเบียบและมาตรฐาน

มีBuilt-inFunctionต่าง ๆ มาให้เช่นAdminsite, Authentication

มีORMที่ใช้ติดต่อกับฐานข้อมูลมาให้ภายในตัว(Django ORM)

มีไลบรารีที่มากมายเพื่อนำมาประยุกต์ใช้ในการพัฒนา

จุดด้อย

มีอิสระในการเขียนน้อยกว่าFlaskเพราะต้องเขียนตามโครงสร้างและแพทเทิร์นของDjango

ไม่เหมาะกับProjectขนาดเล็ก

มีการตั้งค่าส่วนต่างๆที่เยอะทำให้ยุ่งยาก

Flask

จุดเด่น

ง่ายต่อการเรียนรู้เพราะมีโครงสร้างที่ไม่ซับซ้อน

สามารถใช้RawSQLหรือORM(SQLAlchemy)ในการติดต่อฐานข้อมูลได้ซึ่งยืดหยุ่นกว่า

Djangoที่ใช้DjangoORMเกือบทั้งหมด

ได้รับความนิยมสูงคู่กับDjangoจึงทำให้มีResourcesให้ศึกษาเยอะ

จุดด้อย

ไม่ได้มีToolมาให้ครบเหมือนDjango

เหมาะกับProjectขนาดเล็กเพราะถ้าProjectเริ่มมีขนาดใหญ่ขึ้นจะทำให้จัดการกับโครงสร้างต่างๆได้ยากถ้าไม่ออกแบบมาให้ดี

ความต่างกันของDjango&Flask

Attributes	Django	Flask
Type Of Framework	Django is a full-stack web framework that enables ready to use solutions with its batteries-included approach	Flask is a lightweight framework that gives abundant features without external libraries and minimalist features
Architecture	It follows an MVT architecture, which is split up into three parts, model, view, and template	It is a minimalistic framework that provides developers flexibility on application structuring
Project Layout	Django is suitable for multiple page applications	Flask is suitable for only single-page applications
Maturity	Launched in 2005, it is a very mature framework with extensive community support	Launched in 2010, Flask is a younger framework but has a large community
Database Support	Django supports the most popular relational database management systems like MySQL, Oracle etc	Flask does not support the basic database management system and uses SQL Alchemy for database requirements

ภาพที่3.5.2 เปรียบเทียบความต่างระหว่างDjangoและFlask

ภาพจาก(<https://www.simplilearn.com/flask-vs-django-article>)

เปรียบเทียบDjangoและFlask

Cost-Effectiveness (ความคุ้มค่า)

DjangoและFlask เป็นframework open sourceที่ไม่มีค่าใช้จ่ายในการใช้งาน โดย Django มีเครื่องมือในตัวมากกว่าทำให้ลดต้นทุนในการพัฒนาแอปขนาดใหญ่ได้ ในขณะที่ Flask มีความยืดหยุ่นแต่ต้องใช้ไลบรารีเสริมซึ่งอาจเพิ่มค่าใช้จ่ายด้านการพัฒนา

Development Time (ระยะเวลาในการพัฒนา)

Django มาพร้อมกับฟีเจอร์มากมายช่วยให้พัฒนาแอปพลิเคชันได้รวดเร็วยิ่งขึ้น ในขณะที่ Flask ต้องสร้างโครงสร้างและติดตั้งไลบรารีเสริมเองทำให้ใช้เวลามากขึ้นในการตั้งค่าแอป

Ease of Use/Learning (ความง่ายในการใช้งาน/เรียนรู้)

Flask มีโครงสร้างที่เรียบง่ายและใช้งานง่ายกว่าเหมาะสำหรับผู้เริ่มต้น ส่วนDjangoมาในรูปแบบ"batteries included"(เป็นสำนวนที่หมายถึง"มีทุกอย่างที่จำเป็นมาให้พร้อมใช้งาน") ซึ่งแม้จะมีฟีเจอร์มากมาย แต่ต้องใช้เวลาศึกษาแนวทางการทำงานที่เข้มงวดมากกว่า

Employment Opportunities (โอกาสในการทำงาน)

Django เป็นที่นิยมในองค์กรหรือโครงการขนาดใหญ่และมีโอกาสงานมากกว่าFlaskเนื่องจากใช้พัฒนาเว็บแอปพลิเคชันแบบครบวงจร ส่วนFlaskมักถูกใช้ในสตาร์ทอัพหรือโปรเจกต์ที่ต้องการความยืดหยุ่น

Features (ฟีเจอร์)

Django มีฟีเจอร์ในตัวมากมาย เช่น ORM, authentication, admin panelทำให้เหมาะกับโปรเจกต์ที่ต้องการความพร้อมในการใช้งานส่วนFlaskมีเพียงโครงสร้างพื้นฐานและต้องใช้ไลบรารีเสริมสำหรับฟีเจอร์เหล่านี้

Performance and Speed (ประสิทธิภาพและความเร็ว)

Flaskมีประสิทธิภาพสูงและเร็วเนื่องจากเป็นเฟรมเวิร์กที่เบาและไม่มีความซับซ้อน ส่วน Django อาจช้ากว่า Flask เนื่องจากมีฟีเจอร์มากมาย แต่ก็ยังถือว่ามีประสิทธิภาพสูงและเพียงพอสำหรับโครงการส่วนใหญ่

Scalability (ความสามารถในการขยายตัว)

Flask มีความยืดหยุ่นสูงและเหมาะสำหรับแอปพลิเคชันที่ต้องการการขยายตัวแบบโมดูลาร์ ส่วน Django มีโครงสร้างที่แข็งแกร่งและเหมาะกับแอปขนาดใหญ่ที่ต้องการรองรับการเติบโตในระยะยาว

Community Support (ชุมชนผู้ใช้)

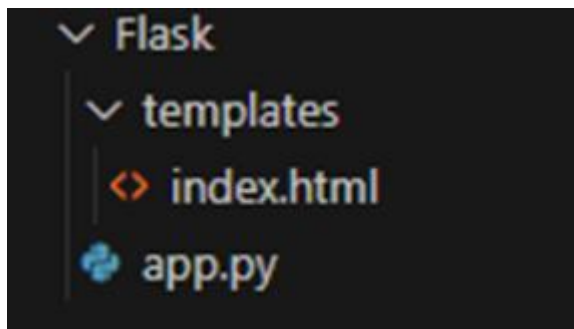
Django มีชุมชนขนาดใหญ่และเอกสารที่ครอบคลุม ทำให้การแก้ปัญหาและหาข้อมูลทำได้ง่าย ส่วน Flask ก็มีชุมชนที่แข็งแกร่งเช่นกัน แต่เล็กกว่า Django เล็กน้อย

Security (ความปลอดภัย)

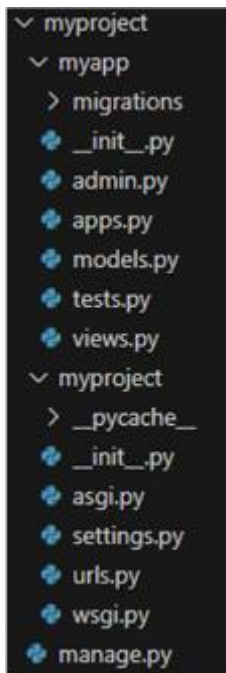
Django มีระบบความปลอดภัยในตัวทำให้ปลอดภัยกว่าโดยไม่ต้องตั้งค่าเพิ่มเติม ขณะที่ Flask ต้องจัดการความปลอดภัยด้วยตัวเองซึ่งอาจเสี่ยงหากไม่มีความรู้เพียงพอ

ความแตกต่างที่เจอในการทดลองใช้งาน

โครงสร้างของตัวProject



ภาพที่3.5.3 โครงสร้างของFlask



ภาพที่3.5.4โครงสร้างของDjango

URL Routing

```
1 from flask import Flask,render_template
2
3 app = Flask(__name__)
4
5 @app.route('/')
6 def home():
7     return render_template('index.html')
8
9 if __name__ == '__main__':
10     app.run(debug=True)
```

ภาพที่3.5.5 URL Routing ใน Flask

```

from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('myapp.urls'))
]

```

Urls.py(Project)

```

from django.urls import path
from myapp import views

urlpatterns = [
    path('', views.index, name='index'),
]

```

Urls.py(Project)

```

from django.shortcuts import render

# Create your views here.
def index(request):
    return render(request, "index.html")

```

View.py

ภาพที่3.5.6 URL Routing ใน Django

การสร้างตารางฐานข้อมูล

```

from flask import Flask
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///db.sqlite3'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False

db = SQLAlchemy(app)

class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(50))
    email = db.Column(db.String(100), unique=True)

```

```

Type "help", "copyright", "credits" or "license" for more information.
>>> from app import db, User
>>> db.create_all()
>>>

```

ภาพที่3.5.7 การสร้างตารางฐานข้อมูลในFlask

```
from django.db import models

# Create your models here.
class User(models.Model):
    name = models.CharField(max_length=100)
    email = models.CharField(max_length=100)
```

```
C:\Users\ASUS\Desktop\SE492\myproject>python manage.py makemigrations
Migrations for 'myapp':
  myapp\migrations\0001_initial.py
    - Create model User
```

```
operations = [
    migrations.CreateModel(
        name='User',
        fields=[
            ('id', models.AutoField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID')),
            ('name', models.CharField(max_length=100)),
            ('email', models.CharField(max_length=100)),
        ],
    ),
]
```

```
C:\Users\ASUS\Desktop\SE492\myproject>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, myapp, sessions
```

ภาพที่3.5.8 การสร้างตารางฐานข้อมูลในDjango

3.6 Built in Featuresและการนำมาใช้งาน

Django Template Language(DTL)เป็นเครื่องมือสำหรับสร้างหน้าเว็บแบบDynamic โดยโครงสร้างและไวยากรณ์พื้นฐานของDLT จะเป็นการผสมระหว่าง HTML และ DTL โดยคุณสมบัติของDTLที่ใช้ในการทำงานจะมี variables , filters , tags

Variablesจะเป็นการแทรกตัวแปรเพื่อแสดงบนหน้าเว็บ รูปแบบ{{ variable_name }}

Filters เป็นการแก้ไขและจัดรูปแบบค่าของตัวแปรก่อนนำไปแสดงผล

รูปแบบ{{ variable_name|upper }}

Tags เป็นตัวควบคุมLogic และ Flow ของTemplates รูปแบบ {% %}

โดยTagsมีอยู่หลายแบบแต่นี้อาจเป็นTagsที่จะเห็นบ่อยๆ

{% if %}กำหนดเงื่อนไข

{% for %}ทำซ้ำ(Loop)

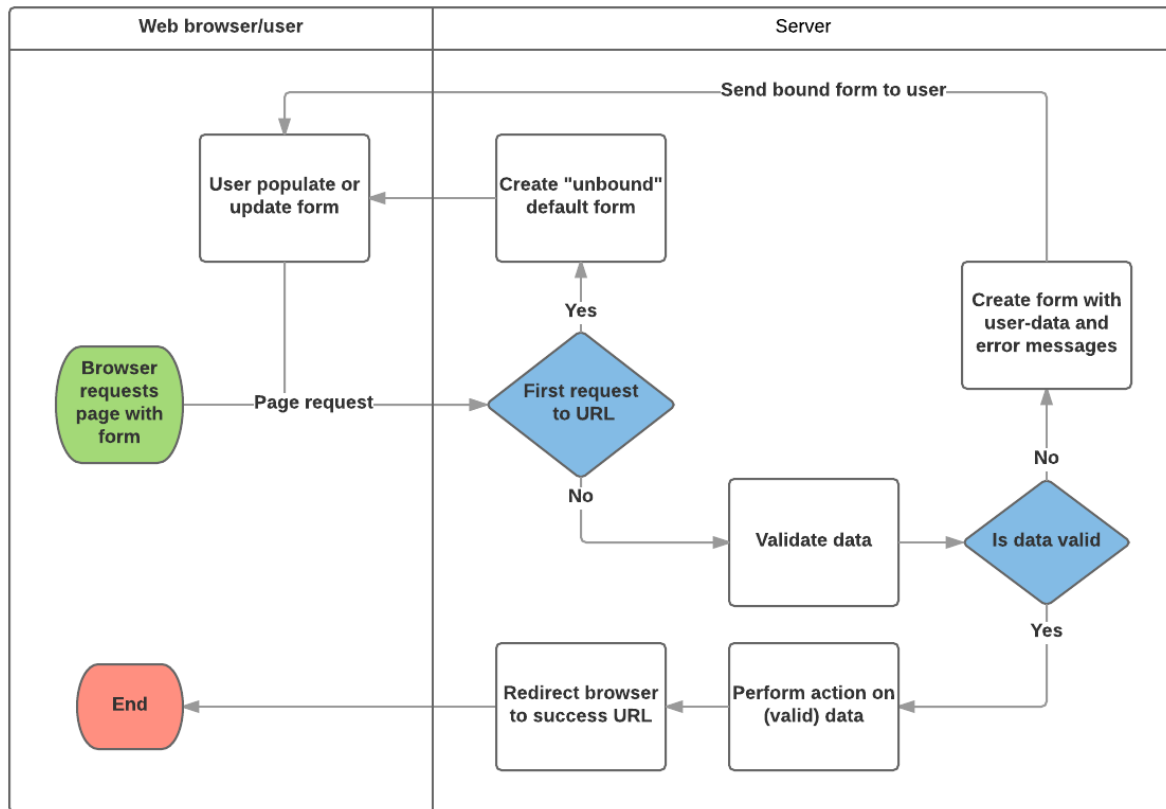
{% block %}เป็นการกำหนดblockของเนื้อหา

{% extends %}เป็นการระบุtemplateที่เป็นbaseเพื่อสืบทอด

Django Formเป็นแบบฟอร์มที่ใช้สำหรับรับข้อมูลจากผู้ใช้งานในรูปแบบต่างๆ และใช้ข้อมูลนั้นดำเนินการกับฐานข้อมูล

โดยDjango แมปฟิลด์ที่กำหนดไว้ในformลงในอินพุตฟิลด์ของHTML และจัดการงาน3ส่วนที่ต่างกัน

1. จัดเตรียมและปรับโครงสร้างข้อมูลให้พร้อมสำหรับการเรนเดอร์
2. สร้างแบบฟอร์ม HTML
3. รับและประมวลผลแบบฟอร์มและข้อมูลที่ส่งมาจากลูกค้า



ภาพที่3.6.1 การทำงานของDjangoForm

ภาพจาก

([https://developer.mozilla.org/en-](https://developer.mozilla.org/en-US/docs/Learn_web_development/Extensions/Server-side/Django/Forms)

[US/docs/Learn_web_development/Extensions/Server-side/Django/Forms](https://developer.mozilla.org/en-US/docs/Learn_web_development/Extensions/Server-side/Django/Forms))

1. user request form ไปยังเซิร์ฟเวอร์
2. มีการตรวจเช็ค ว่า request นี้เป็นครั้งแรกหรือไม่ ถ้าใช่จากการส่งformเปล่ากลับไปให้user กรอกและ อัปเดตform ถ้าไม่ใช่จะไปยังขั้นตอนถัดไป
3. เมื่อตรวจสอบว่าไม่ใช่ครั้งแรกทำการเซิร์ฟเวอร์จะรับข้อมูลจากformและไปตรวจสอบ ถ้าข้อมูลผิดจะทำการส่งฟอร์มกลับไปให้userพร้อมกับบอกข้อความแจ้งเตือนข้อผิดพลาด
4. ในกรณีที่ข้อมูลถูกต้องจะดำเนินการตามข้อมูลที่ได้รับ และ นำผู้ใช้ไปยัง URLที่กำหนดไว้ จะเป็นการเสร็จสิ้นกระบวนการทำงาน

Django Authentication คือระบบการยืนยันตัวตนที่ช่วยในการตรวจสอบและกำหนดสิทธิ์การเข้าถึงของผู้โดยจะตรวจสอบว่าผู้ใช้เป็นใครและกำหนดว่าผู้ใช้ที่ได้รับการตรวจสอบสิทธิ์จะได้รับอนุญาตให้ทำอะไรได้บ้าง

ในระบบนี้จะประกอบด้วย

ผู้ใช้ Users

สิทธิ์ Permissions ที่กำหนดว่าผู้ใช้สามารถทำอะไรได้บ้าง

กลุ่ม Groups การกำหนดสิทธิ์ผู้ใช้ที่มีมากกว่า1คน

การhashing password ที่สามารถกำหนดค่าได้

Form และ view สำหรับเข้าสู่ระบบหรือจำกัดเนื้อหาของผู้ใช้

Authentication ของDjangoถูกออกแบบมาให้มีความยืดหยุ่นที่สูงจึงจะไม่มีฟีเจอร์บางอย่างที่มักจะพบในระบบยืนยันตัวตนของเว็บทั่วไป แต่อย่างไรก็ตาม มีการใช้แพ็คเกจจากนักพัฒนาภายนอกมาช่วยเสริมฟีเจอร์เหล่านี้เช่น

การตรวจสอบความแข็งแรงของรหัสผ่าน

การจำกัดจำนวนครั้งในการพยายามlogin

การloginผ่านบริการอื่น(OAuth)

การกำหนดสิทธิ์การเข้าถึงในระบบวัตถุ(Object-level permissions)

Django admin siteเป็นระบบที่จะทำการอ่านข้อมูลจากโมเดลเพื่อสร้างinterfaceที่รวดเร็วเพื่อจัดการกับเนื้อหาบนเว็บของคุณโดยจะมีการให้สร้างsuperuserเพื่อเป็นคนจัดการกับข้อมูลเหล่านั้น

การนำมาใช้งาน

ทำฟีเจอร์loginและregister


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

{% block content %}
<h2>Login</h2>
<form method="post">
    {% csrf_token %}
    {{ form.as_p }} <!-- แสดงฟอร์ม -->
    <button type="submit" class="btn btn-primary">Login</button>
</form>

<!-- แสดงข้อความแจ้งเตือน -->
{% if messages %}
<ul class="messages">
    {% for message in messages %}
    <li {% if message.tags %} class="{{ message.tags }}" {% endif %}>{{ message }}</li>
    {% endfor %}
</ul>
{% endif %}
{% endblock %}
```

ภาพที่ 3.6.2 login.html

```
{% extends 'base_generic.html' %}
{% block content %}
<h2>สมัครสมาชิก</h2>
<form method="POST">
    {% csrf_token %}
    {{ form.as_p }}
    <button type="submit">สมัคร</button>
</form>
{% endblock %}
```

ภาพที่ 3.6.3 Sign_up.html

```
from django import forms
from django.contrib.auth.forms import AuthenticationForm, UserCreationForm
from django.contrib.auth.models import User

# SignUp Form
class SignUpForm(UserCreationForm):
    class Meta:
        model = User
        fields = ['username', 'password1', 'password2']

# Login Form
class LoginForm(AuthenticationForm):
    class Meta:
        model = User
        fields = ['username', 'password']
```

ภาพที่ 3.6.4 Forms.py

```

from django.shortcuts import render, redirect
from django.contrib.auth import login, authenticate
from django.contrib import messages
from .forms import SignUpForm, LoginForm
from django.contrib.auth.decorators import login_required # ใช้ decorator เพื่อตรวจสอบการล็อกอิน
from .models import CustomUser

# SignUp View
def sign_up(request):
    if request.method == 'POST':
        form = SignUpForm(request.POST)
        if form.is_valid():
            form.save()
            messages.success(request, 'สมัครสมาชิกสำเร็จ!')
            return redirect('login')
    else:
        form = SignUpForm()
    return render(request, 'accounts/sign_up.html', {'form': form})

```

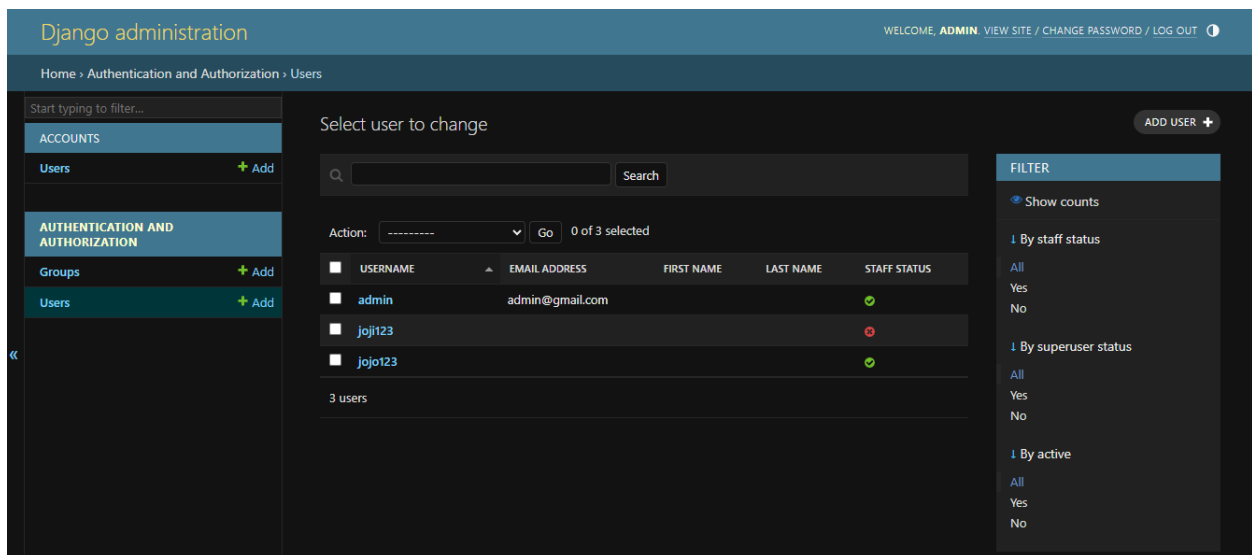
ภาพที่ 3.6.5 View.py(Sign_up&Library)

```

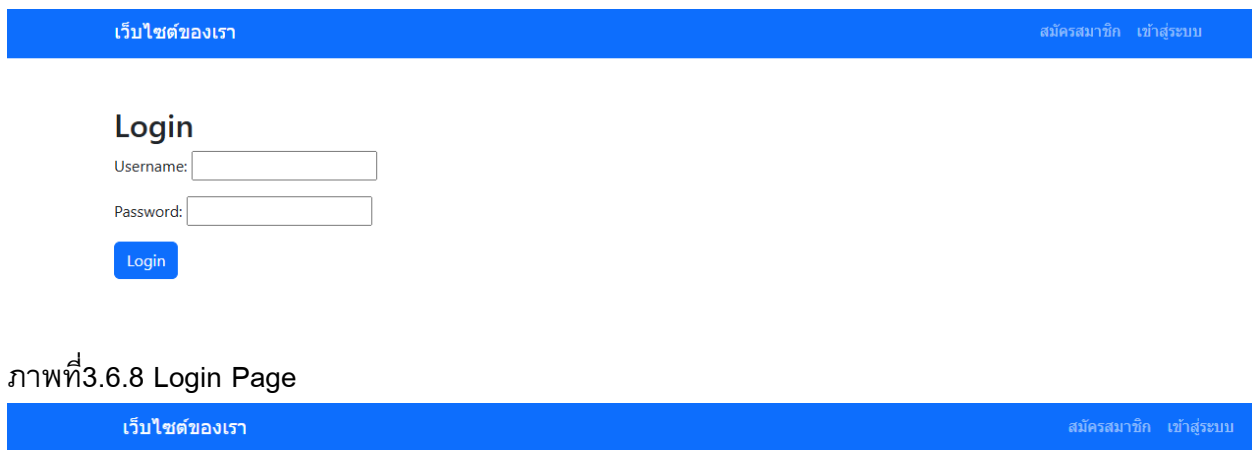
def login_view(request):
    if request.method == 'POST':
        form = LoginForm(request, data=request.POST) # ใช้ data=request.POST
        if form.is_valid():
            username = form.cleaned_data['username']
            password = form.cleaned_data['password']
            user = authenticate(request, username=username, password=password)
            if user is not None:
                login(request, user)
                messages.success(request, "เข้าสู่ระบบสำเร็จ!") # แสดงข้อความเมื่อเข้าสู่ระบบสำเร็จ
            else:
                messages.error(request, "ชื่อผู้ใช้หรือรหัสผ่านไม่ถูกต้อง") # แสดงข้อความเมื่อข้อมูลไม่ถูกต้อง
        else:
            messages.error(request, "กรุณากรอกข้อมูลให้ถูกต้อง") # แสดงข้อความเมื่อฟอร์มไม่ถูกต้อง
    else:
        form = LoginForm() # สร้างฟอร์มเปล่าเมื่อ request.method เป็น GET
    return render(request, 'accounts/login.html', {'form': form})

```

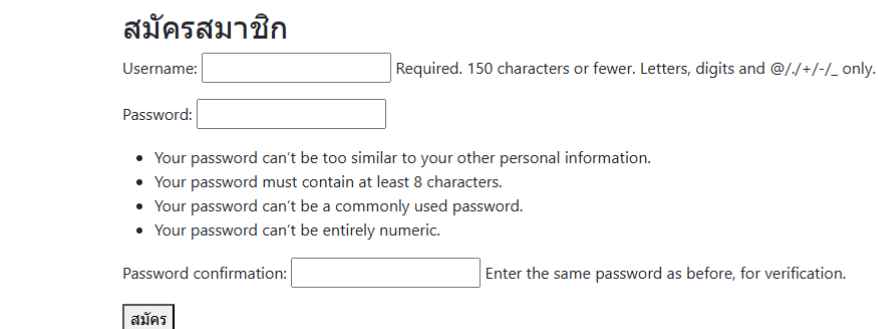
ภาพที่ 3.6.6 View.py(Login)



ภาพที่3.6.7 Admin page



ภาพที่3.6.8 Login Page



ภาพที่3.6.9 RegisterPage

บทที่ 4

สรุปผลการศึกษา

(ขนาดตัวอักษร TH Sarabun New ขนาด 16 Points)

4.1 องค์ความรู้ที่ได้

4.1.1 Introduction และหลักการทำงานของ Django Framework

ได้ทราบถึงความเป็นมาข้อดีข้อเสียของDjangoรวมถึงรู้ถึงสถาปัตยกรรมการทำงานของDjango และเครื่องมือต่าง ๆ ที่ติดตั้งมาพร้อมกับDjango

4.1.2 การติดตั้งและการเริ่มต้นใช้งาน Django Framework

ได้รู้ถึงการเริ่มสร้างโปรเจกต์ของDjangoและโครงสร้างและไฟล์ต่าง ๆ ที่ถูกสร้างขึ้นโดยอัตโนมัติ และได้รู้ถึงการใช้งานURL Routingเบื้องต้น

4.1.3 หลักการและการเชื่อมต่อกับฐานข้อมูลในDjango

ORMมีส่วนช่วยในการพัฒนาโปรเจกต์ของเราให้ประสิทธิภาพเพิ่มขึ้นได้แต่ก็ยังมีข้อจำกัดในการqueryที่บางกรณีจะมีประสิทธิภาพต่ำกว่าภาษาSQL

Model ใน Django มีไว้เพื่อทำงานในส่วนของหลังบ้านได้รู้ถึงการสร้างmodelในตัวโปรเจกต์และทำให้modelเข้าสู่ฐานข้อมูลและสามารถถูกการจัดการข้อมูลได้ผ่านadmin panel

4.1.4 Django Rest Framework

ได้รู้ว่าDjango REST FRAMEWORKคือToolkitหรือไลบรารีของPythonในการสร้างRestful APIs และ หลักการทำงานของมันจะยึดหลักของRestful api และได้นำมาเปรียบเทียบกับDjango ว่ามันแตกต่างกันตรงส่วนไหนและเหมาะกับงานประเภทไหน

4.1.5 เปรียบเทียบDjango & Flask

DjangoและFlaskเป็นFrameworkภาษาPythonที่ได้รับความนิยมสูงทั้งคู่ โดยการใช้ทั้งคู่ให้เหมาะสมขึ้นอยู่กับงานที่ได้มาเราจะเลือกใช้ตัวไหนในการพัฒนาเช่น Flaskเหมาะกับงานที่มีขนาดเล็กและสามารถพัฒนาได้อย่างรวดเร็วเป็นต้นDjangoเหมาะกับงานขนาดใหญ่ที่มีฟีเจอร์การทำงานที่ครบและมีความซับซ้อนของงานมากกว่าเป็นต้น

4.1.6 Built in Featuresและการนำมาใช้งาน

ได้ทราบถึงวิธีการนำBuilt in Featuresมาใช้งานและกระบวนการในการทำงานของแต่ละตัว และนำมาปรับใช้ในการพัฒนาระบบ ซึ่งจะการนำBuilt in Featuresมาใช้จะช่วยทำให้การพัฒนาระบบ สามารถทำได้รวดเร็วมากขึ้น

บรรณานุกรม

1. การอ้างอิง

- 9mza. (ม.ป.ป.). *ORM คืออะไร มาทำความรู้จักกันเถอะ*. เข้าถึงได้จาก 9mza.net: <https://9mza.net/post/what-is-orm>
- Ahmed Bahgat. (5 6 2023). *Flask vs Django: Let's Choose Your Next Python Framework*. เข้าถึงได้จาก kinsta.com: <https://kinsta.com/blog/flask-vs-django/#development-time>
- Anshul Srivastava,Vidya Shinde,Vinayak Walhekar,Ashwini Patil,Ankit Ganeshpurkar,M Karthikeyan,Ravindra Kulkarni Shubham Dhadil. (01 Jan 2024). *SCISPACE*. เรียกใช้เมื่อ 1 12 2024 จาก <https://typeset.io/papers/django-unleashed-a-deep-dive-into-the-features-and-284spzclohex>
- Anuranjana,Yannick Roy Sanmukh Kaur. (05 Sep 2023). *SCISPACE*. เรียกใช้เมื่อ 1 12 2024 จาก <https://typeset.io/papers/website-development-with-django-web-framework-2aefaz0qlc>
- AppMaster. (28 8 2023). *AppMaster*. เข้าถึงได้จาก REST API ทำงานอย่างไร: <https://appmaster.io/th/blog/rest-api-thamngaan-yaangair>
- asw. (ม.ป.ป.). *RESTful API คืออะไร*. เข้าถึงได้จาก awsamazon: <https://aws.amazon.com/th/what-is/restful-api/>
- bin, Uzayr. Sufyan. (03 Oct 2022). *SCISPACE*. เรียกใช้เมื่อ 1 12 2024 จาก <https://typeset.io/papers/mastering-django-2iknh00>
- Carvalhar, Fernandes., Juliane, Andressa, Camatti., Sandro, Jessé, Ferreira, Tabor., Luiz, Gustavo, Romanel., Liam, Brown., Milton, Borsato Ederson. (1 Jan 2023). *Value Stream Mapping Application Enabled by Django Web Framework*. เข้าถึงได้จาก SCISPACE: <https://typeset.io/papers/value-stream-mapping-application-enabled-by-django-web-3qpz072ukp>
- Chen., Yi-Ling, Lin., Bean, Yin, Lee Jenn-Yih. (28 Oct 2022). *Development of Tool Management System based on Django Web Framework*. เข้าถึงได้จาก SCISPACE: <https://typeset.io/papers/development-of-tool-management-system-based-on-django-web-1gxbqxbp>
- Codesanook. (14 11 2019). *ข้อดีของ Object-Relational mapping หรือ ORM มีอะไรบ้าง*. เข้าถึงได้จาก codesanook.com: <https://www.codesanook.com/advantages-of-object-relational-mapping-orm>
- Denis Mashutin. (9 9 2024). *Django vs. Flask: Which Is the Best Python Web Framework?* เข้าถึงได้จาก blog.jetbrains.com: <https://blog.jetbrains.com/pycharm/2023/11/django-vs-flask-which-is-the-best-python-web-framework/#what-is-django>
- DevSumitG. (18 02 2023). *Difference Between App And Project In Django*. เข้าถึงได้จาก medium: <https://medium.com/@devsumitg/difference-between-app-and-project-in-django-a40a40e7e038>

DH Team. (24 11 2024). *Django vs Flask ต่างกันอย่างไร ควรเลือกตัวไหนดี ?* เข้าถึงได้จาก devhub.in.th:

<https://devhub.in.th/blog/django-vs-flask>

Django. (ม.ป.ป.). *The Django admin site*. เข้าถึงได้จาก docs.djangoproject.com:

<https://docs.djangoproject.com/en/5.1/ref/contrib/admin/>

Django. (ม.ป.ป.). *User authentication in Django*. เข้าถึงได้จาก docs.djangoproject.com:

<https://docs.djangoproject.com/en/5.1/topics/auth/>

Dr Rainu Nandal Manoj Kumar. (29 Jun 2024). *SCISPACE*. เรียกใช้เมื่อ 1 12 2024 จาก

<https://typeset.io/papers/python-s-role-in-accelerating-web-application-development-4fvnuf6wlv>

Evgenia Verbina. (5 2 2025). *The Ultimate Guide to Django Templates*. เข้าถึงได้จาก blog.jetbrains.com:

<https://blog.jetbrains.com/pycharm/2025/02/the-ultimate-guide-to-django-templates/>

geeksforgeeks. (21 7 2024). *Django Forms*. เข้าถึงได้จาก geeksforgeeks.org:

<https://www.geeksforgeeks.org/django-forms/>

Islam., Haralambos, Mouratidis., Jan, Jürjens Shareeful. (1 Jul 2011). *A framework to support*

alignment of secure software engineering with legal regulations. เข้าถึงได้จาก SCISPACE:

<https://typeset.io/papers/a-framework-to-support-alignment-of-secure-software-4k1djmeoio>

James Bennett. (1 Nov 2008). *SCISPACE*. เรียกใช้เมื่อ 1 12 2024 จาก [https://typeset.io/papers/practical-](https://typeset.io/papers/practical-django-projects-591hjhr2ea)

[django-projects-591hjhr2ea](https://typeset.io/papers/practical-django-projects-591hjhr2ea)

KongRuksiam Official. (24 12 2022). *พัฒนาเว็บด้วย Django Framework / สำหรับผู้เริ่มต้น [FULL*

COURSE]. เข้าถึงได้จาก Youtube:

<https://www.youtube.com/watch?v=XLMLveR2BYo&t=7833s>

KongRuksiam Official. (22 12 2022). *พัฒนาเว็บด้วย Django Framework / สำหรับผู้เริ่มต้น [FULL*

COURSE]. เข้าถึงได้จาก Youtube:

<https://www.youtube.com/watch?v=XLMLveR2BYo&t=2741s>

Littikrai. (8 9 2020). *เรียนรู้การใช้งาน Django Rest Framework with React.js [EP.1]*. เข้าถึงได้จาก

STACKPYTHON: <https://stackpython.co/tutorial/django-rest-framework-with-reactjs-ep1>

Mariko Sasakura,Kinari Nishiura,Hiroki Inayoshi,Akito Monden Hikaru Tomita. (30 May 2024).

SCISPACE. เรียกใช้เมื่อ 1 12 2024 จาก <https://typeset.io/papers/porting-a-python-application-to-the-web-using-django-a-case-7hm6vkuo9lge>

ntprintf. (ม.ป.ป.). *ORM (Object-Relational Mapping)คืออะไร มีประโยชน์อย่างไร ใช้งานตอนไหน อธิบายแบบง่าย*

ที่สุด แบบเด็ก 8 ปีก็เข้าใจ. เข้าถึงได้จาก expert-programming-tutor.com: https://expert-programming-tutor.com/tutorial/article/KE001156_ORM_Object-Relational_What_is_mapping_How_useful_is_it_When_you_use_it_the_easiest_explanation.php

O. V. Bratkovskyy. (1 Jan 2024). *Enhancing data protection in a web application using Django*. เข้าถึง

ได้จาก SCISPACE: <https://typeset.io/papers/enhancing-data-protection-in-a-web-application-using-django-4xo04lrtb70a>

- Parfonov., O., Kolgatin. Yu. (24 May 2022). *Choosing a web hosting service for applications based on Django framework*. เข้าถึงได้จาก SCISPACE: <https://typeset.io/papers/choosing-a-web-hosting-service-for-applications-based-on-2anfyuc8>
- Pooja Thakur Rajiv Khandelwal. (05 Apr 2024). *SCISPACE*. เรียกใช้เมื่อ 1 12 2024 จาก <https://typeset.io/papers/digital-transformation-in-b2b-destination-management-4jp7grbhj>
- S. Prakash,D. Tamilselvam M. Ramya. (26 Jul 2024). *SCISPACE*. เรียกใช้เมื่อ 1 12 2024 จาก <https://typeset.io/papers/creating-a-web-application-and-elevate-learning-quiz-3vkodmt29b>
- Satyam Singh,Ashish Sharma Satish Singh. (2023 Dec 15). *Real-Time Secure Web-Based Chat Application using Django*. เข้าถึงได้จาก SCISPACE: <https://typeset.io/papers/real-time-secure-web-based-chat-application-using-django-54kdqf9bb8>
- Shahed Ahmmmed,Karu Lal,Chunhua Deming Songtao Chen. (31 Dec 2020). *SCISPACE*. เรียกใช้เมื่อ 1 12 2024 จาก <https://typeset.io/papers/django-web-development-framework-powering-the-modern-web-1rxej4n866>
- Simplilearn. (13 8 2024). *Django Vs. Flask: Understanding The Major Differences*. เข้าถึงได้จาก [simplilearn.com: https://www.simplilearn.com/flask-vs-django-article](https://www.simplilearn.com/flask-vs-django-article)
- Sonny. (6 Oct 2021). *ทำไมต้องใช้ Django*. เข้าถึงได้จาก STACKPYTHON: <https://stackpython.co/tutorial/django>
- stackpython. (4 7 2020). *สร้าง API แบบติดจรวดด้วย Django REST Framework*. เข้าถึงได้จาก [stackpython.medium.com: https://stackpython.medium.com/%E0%B8%AA%E0%B8%A3%E0%B9%89%E0%B8%B2%E0%B8%87-api-%E0%B9%81%E0%B8%9A%E0%B8%9A%E0%B8%95%E0%B8%B4%E0%B8%94%E0%B8%88%E0%B8%A3%E0%B8%A7%E0%B8%94%E0%B8%94%E0%B9%89%E0%B8%A7%E0%B8%A2-django-rest-framework-3f7172313bac](https://stackpython.medium.com/%E0%B8%AA%E0%B8%A3%E0%B9%89%E0%B8%B2%E0%B8%87-api-%E0%B9%81%E0%B8%9A%E0%B8%9A%E0%B8%95%E0%B8%B4%E0%B8%94%E0%B8%88%E0%B8%A3%E0%B8%A7%E0%B8%94%E0%B8%94%E0%B9%89%E0%B8%A7%E0%B8%A2-django-rest-framework-3f7172313bac)
- StackShare Team. (28 Jan 2021). *The Top 100+ Developer Tools 2020*. เข้าถึงได้จาก Stackshare: <https://stackshare.io/posts/top-developer-tools-2020#backend-full-stack-frameworks>
- TechVidvan Team. (02 06 2021). *Django Project Structure and File Structure*. เข้าถึงได้จาก techvidvan: <https://techvidvan.com/tutorials/django-project-structure-layout/>
- Tehreem Naeem. (3 9 2024). *REST API Definition: What are REST APIs (RESTful APIs)?* เข้าถึงได้จาก [astera.com: https://www.astera.com/type/blog/rest-api-definition/](https://www.astera.com/type/blog/rest-api-definition/)
- V., Venkatesh, P., Ravi, Kiran, K., Sadhana, P., Osman, Khan., Ch., Nanda, Krishna Puneet. (29 Mar 2022). *A Django Web Application to Promote Local Service Providers*. เข้าถึงได้จาก SCISPACE: <https://typeset.io/papers/a-django-web-application-to-promote-local-service-providers-196a1aqd>
- Withoutcoffee Icantbedev. (23 2 2023). *Django Model & ORM ครบ จบในบทความเดียว*. เข้าถึงได้จาก [devhub.in.th: https://devhub.in.th/blog/django-model-and-orm](https://devhub.in.th/blog/django-model-and-orm)

Withoutcoffee Icantbedev. (24 11 2024). *Django REST Framework 101 พัฒนา API ด้วยภาษา Python*. เข้าถึงได้จาก devhub.in.th: <https://devhub.in.th/blog/django-rest-framework>

ศวทช. (3 5 2024). แนะนำ *Object-Relational Mapping(ORM)*. เข้าถึงได้จาก www.nstda.or.th: https://www.nstda.or.th/home/knowledge_post/object-relational-mapping-orm/