เอกสารประกอบการสอน

รายวิชา

คพ.459 หัวข้อเลือกสรรด้านระบบสารสนเทศ

Django Web Framework

ผู้ช่วยศาสตราจารย์ ดร. วสิศ ลิ้มประเสริฐ

สาขาวิทยาการคอมพิวเตอร์

คณะวิทยาศาสตร์และเทคโนโลยี

มหาวิทยาลัยธรรมศาสตร์

2562

Django Web Framework

เอกสารประกอบการสอน
คพ.459 หัวข้อเลือกสรรด้านระบบสารสนเทศ
อาจารย์ ดร. วสิศ ลิ้มประเสริฐ
สาขาวิชาวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์และเทคโนโลยี มหาวิทยาลัยธรรมศาสตร์

Lecture note
CS459 Selected Topics in Information Systems
Wasit Limprasert, PhD, wasit_l@sci.tu.ac.th
Department of Computer Science, Faculty of Science and Technology, Thammasat University

Django Web Framework เป็นชุดเครื่องมือช่วยสร้างระบบเว็บแอปพลิเคชัน ซึ่งสามารถเชื่อมต่อฐานข้อมูลด้วยเทคนิคที่เรียกว่า Object Relational Mapping (ORM) จึงทำให้จัดการข้อมูลที่ซับซ้อนของฐานข้อมูลได้โดยใช้คำสั่งที่สั้นเมื่อเทียบกับการใช้ SQL จึงถูกนำมาประยุกต์ใช้เป็นเครื่องมือในการเก็บข้อมูลผ่าน REST API ซึ่งมีความสำคัญต่อการเชื่อมโยงระบบแอปพลิเคชันในปัจจุบัน นอกจากนี้ Django พัฒนาโดยใช้ภาษา Python ที่มีเครื่องมือแวดล้อมสนับสนุนหลากหลายเช่น Machine Learning, ETL, Data Analytics, Security, NLP และ Computer Vision จึงทำให้สามารถพัฒนาระบบข้อมูลได้รวดเร็วโดยไม่ต้องเขียนโปรแกรมส่วนพื้นฐาน เช่น ระบบยืนยันบุคคลด้วยรหัสผ่าน การจัดการฐานข้อมูล และการป้องกันการโจมตีเว็บแอปพลิเคชัน จึงเหมาะสมในการใช้เป็นเครื่องมือในการสอนผู้เรียนเบื้องต้นให้เข้าใจระบบข้อมูลที่มีการจัดการข้อมูลอย่างมีประสิทธิภาพและปลอดภัย

ในส่วนแรกของเอกสารนี้จะอธิบายโดยใช้ภาษาไทยเพื่อให้ผู้อ่านเข้าใจสิ่งแวดล้อมคอมพิวเตอร์และเครื่องมือที่จะถูกใช้ หลังจากนั้นจะใช้คำอธิบายเป็นภาษาอังกฤษเพื่อให้ผู้เรียนคุ้นชินกับรูปแบบคู่มือการใช้งานมาตราฐานซึ่งทั่วไปเอกสารคู่มือการใช้งานเชิงเทคนิคจะถูกเขียนเป็นภาษาอังกฤษทั้งหมด จึงสนับสนุนให้ผู้อ่านใช้ภาษาอังกฤษในการเขียนบันทึกคอมเม้นภายในโปรแกรมและนำไปเขียนคู่มือเชิงเทคนิคของระบบที่จะพัฒนาขึ้นในอนาคต โดยผู้สอนได้เตรียมโปรแกรมตัวอย่างที่ใช้ในการเรียนการสอนและสามารถเข้าถึงได้จาก https://github.com/wasit7/cs459_django2018

```
git clone https://github.com/wasit7/cs459_django2018.git
```

Contents

Week 01 Introduction to Django Web Framework

ปัจจุบันมีเครื่องมือและแพลตฟอร์มสำหรับทำเว็บโมบายแอปพลิเคชันอยู่หลากหลาย ในวิชานี้เลือกนำเสนอ Django ด้วยเหตุผลสำคัญ 3 ประการได้แก่

1.ความเร็วในการพัฒนาแอบพลิเคชันโดยใช้โครงสร้างและภาษาที่เข้าใจง่าย

2.ซอฟต์แวร์เครื่องมือให้เลือกจำนวนมากกว่า 2000 ชิ้น

3.ความปลอดภัยของข้อมูลเนื่องจากเป็นแพลตฟอร์มที่ได้รับการทดสอบมามากกว่า 20ปี จึงได้เลือกเทคโนโลยีนี้ไว้สำหรับเป็นต้นแบบในนักศึกษาและผู้สนใจได้ทดลองใช้งานเพื่อนำไปใช้และยกระบบอุตสาหกรรมซอฟต์แวร์ไทย

First Run Server

ในสัปดาห์แรกนี้จะเป็นการแนะนำให้รู้จักเครื่องมือที่จำเป็นซึ่งจะใช้ต่อเนื่องหลังจากนี้ ได้แก่ basic command line, git และ Django command line interface โดยแนะนำให้ผู้เรียนหาหนังสือ [1-2] และคู่มือการใช้งานเชิงเทคนิค [3-4] เพื่อขยายแหล่งอ้างอิงเพราะเนื้อหาในวิชานี้เป็นเพียงส่วนพื้นฐานที่จำเป็น โดยยังมีเทคโนโลยีขั้นสูงที่ไม่ได้กล่าวถึงอีกเช่น deployment, load balancing, time zone และ server push จึงจำเป็นต้องหาข้อมูลเพิ่ม

Setup

การเริ่มต้นพัฒนาระบบเว็บแอปพลิเคชันด้วย Djago จะเริ่มจากการสร้างโฟลเดอร์และเข้าโฟลเดอร์ไปในโฟลเดอร์ cs459_django2018 ตามคำสั่งด้านล่าง หลังจากนั้นสร้างโฟลเดอร์ week1 และเข้าโฟลเดอร์ไปในโฟลเดอร์เพื่อเป็น root directory ของ project

```
mkdir cs459_django2018
cd cs459_django2018
mkdir week01
cd week01
```

จากนั้นทำการสร้างโปรเจค โดยใช้คำสั่ง django-admin startproject <project name>

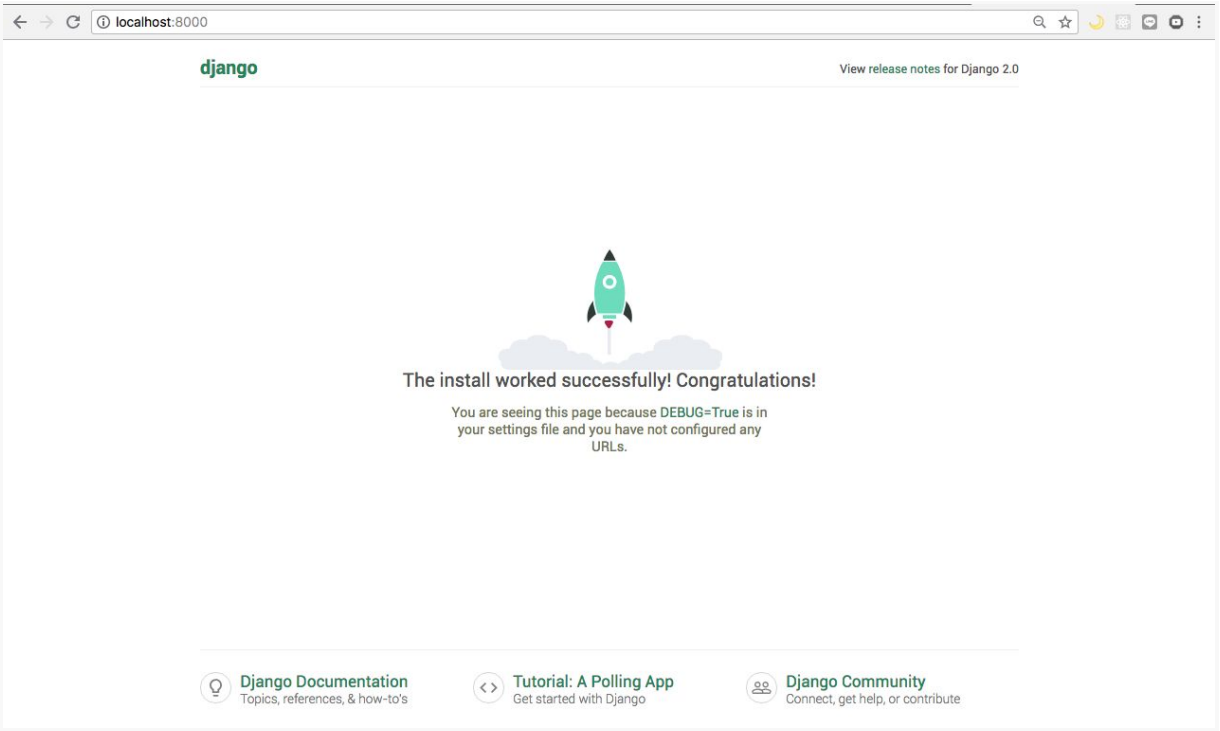
```
django-admin startproject project01
cd project01
```

เราสามารถทดสอบการติดตั้ง ด้วยการเริ่มต้นระบบด้วยคำสั่ง runserver

```
python manage.py runserver
```

ซึ่งควรได้ผลตามรูปด้านล่างนี้

จากนั้นเราสามารถเข้าไปทดสอบโดยการเปิดเว็บเบราว์เซอร์ เข้าไปที่ http://localhost:8000 หรือ http://127.0.0.1:8000 จะได้ดังนี้



เราสามารถนำโปรแกรมที่เขียนไว้เข้าไปเก็บไว้ที่ local repository ได้ด้วยคำสั่ง git add ซึ่งต้องการการเริ่มต้น git ซึ่งขึ้นอยู่กับ repository ที่จะใช้งาน

```
git add -A
git commit -m "first run server"
```

Add Superuser

หลังจากเริ่มการทำงานของระบบ server เราสามารถสร้างฐานข้อมูลโดยเริ่มด้วยการใช้คำสั่ง makemigrations เป็นการสังเคราะห์ SQL เพื่อเปลี่ยนแปลง database schema ของฐานข้อมูล หลังจากนั้นใช้คำสั่ง migrate เพื่อทำการสั่งฐานข้อมูลปฏิบัติการตาม SQL ที่สังเคราะห์ไว้

```
python manage.py makemigrations
python manage.py migrate
```

ทุกฐานข้อมูลควรจะมี admin ที่ดูแลข้อมูลในระบบ เราสามารถสร้าง admin account โดยการสร้าง superuser ด้วยคำสั่ง createsuperuser

```
python manage.py createsuperuser
```

หลังจากนั้นทำการ runserver อีกครั้งเพื่อดูการเปลี่ยนแปลง ซึ่งควรจะได้ผลลัพธ์ตามรูปด้านล่าง

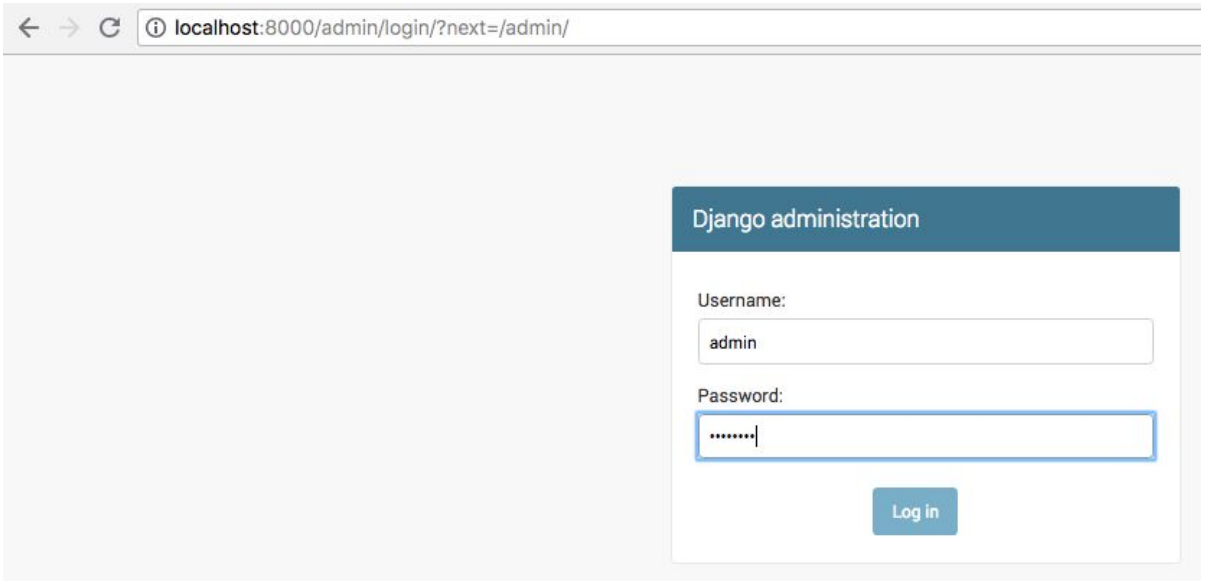
```
python manage.py runserver
```

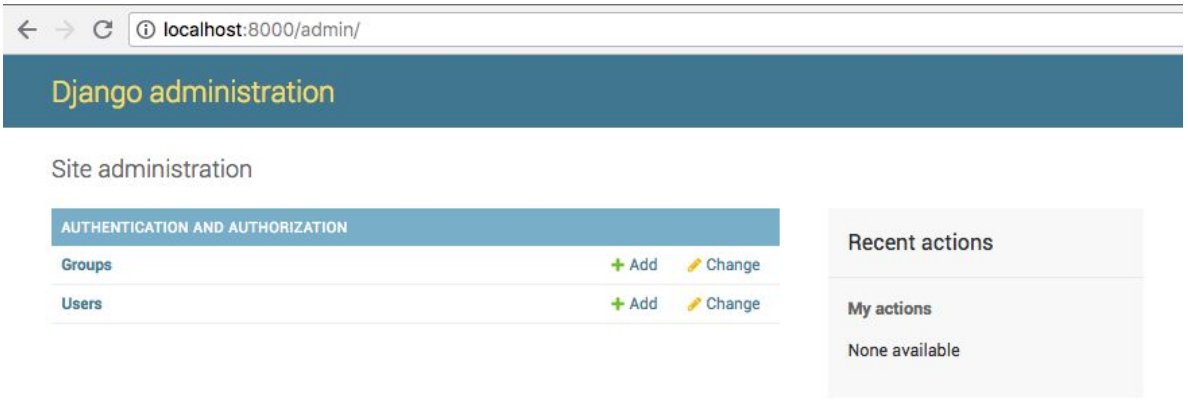
```
(cs459) Sothana-MacBook-Pro:project01 naii$ python manage.py makemigrations
No changes detected
(cs459) Sothana-MacBook-Pro:project01 naii$ python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying sessions.0001_initial... OK
(cs459) Sothana-MacBook-Pro:project01 naii$ python manage.py createsuperuser
Username (leave blank to use 'naii'): admin
Email address:
Password:
Password (again):
Superuser created successfully.
(cs459) Sothana-MacBook-Pro:project01 naii$ python manage.py runserver
Performing system checks...

System check identified no issues (0 silenced).
July 28, 2018 - 06:54:56
Django version 2.0.7, using settings 'project01.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

เราสามารถเข้าไปดูข้อมูลในระบบจัดการฐานข้อมูลที่มาพร้อมกับ Django Web Framework โดยการเปิดเบราว์เซอร์และเข้าไปที่ http://localhost:8000/admin หรือ http://127.0.0.1:8000/admin ซึ่งจะได้หน้าจอตามรูปด้านล่าง โดยระบบจะสร้างตารางข้อมูล user และ group ให้

ตาราง user เก็บข้อมูลผู้ใช้และ hashed password ซึ่งรหัสที่ใช้ถอดข้อมูลถูกระบุไว้ในไฟล์ setting.py โดยในการนำเว็บเผยแพร่สู่สาธารณะจำเป็นต้องปิดรหัสนี้ไว้เพื่อป้องกันการสูญเสียข้อมูลในฐานข้อมูล

ตาราง group เก็บข้อมูลประเภทของผู้ใช้และสิทธิการเข้าถึงส่วนต่างๆของฐานข้อมูล โดยการกำหนดสิทธิสามารถทำได้ภายในระบบจัดการนี้ Django Admin หรือสามารถระบุลงไปในโปรแกรมของส่วนที่ต้องการให้ผู้ใช้เข้าถึง





ถ้าได้หน้าจอผลลัพธ์ตามรูปภาพถือว่าสำเร็จในการสร้างเว็บแอปพลิเคชันตั้งต้น โดยจะมีหน้ายืนยันตัวตนด้วยรหัส หน้าจัดการฐานข้อมูล ตามที่เห็นด้านบน ถือเสร็จสิ้นการสร้างเว็บพื้นฐานในสัปดาห์แรกนี้

reference:

Reference:

[1] Greenfeld, Daniel, and Greenfeld, Audrey. Two Scoops of Django: Best Practices for Django 1.11. Two Scoops Press, 2017.

[2] Tarantino, Quentin. Django unchained. Giunti, 2014.

[3] Django Software Foundation, Django Document, https://docs.djangoproject.com/en/2.1/, retrived 2018

[4] Wasit Limprasert, CS459 Django 2018 week01, https://github.com/wasit7/cs459_django2018/tree/master/week01

Week 02 Generic View

Install python packages

เนื่องจากสัปดาห์นี้ต้องใช้โมดูลไพทอนหลายตัวที่นักเรียนยังไม่ได้ติดตั้งจึงแนะนำให้ติดตั้งโดยการดาวน์โหลด requirement.txt ตามลิงค์นี้ แล้วใช้คำสั่งด้านล่าง เพื่อลง python package ที่จำเป็นสำหรับสัปดาห์นี้

```
pip install -r requirement.txt
```

create app

เราเริ่มทวนความรู้เดิมในการสร้างโปรเจคโดยการสร้าง app โดยใช้คำสั่ง python manage.py startapp <app name>

```
python manage.py startapp myapp
```

install app /myproject/setting.py

เราสามารถเพิ่ม app ที่ได้สร้างขึ้นใหม่หรือที่มีอยู่แล้วและนำเข้าระบบ โดยการนำโฟล์เดอร์ app ไปไว้ในโปรเจค และทำการแก้ไขไฟล์ project01/settings.py โดยเพิ่มราบละเอียดด้านล่างนี้

```
INSTALLED_APPS = [
     ...
    'myapp',
    'crispy_forms',
    'django_extensions',
]
```

setup media storage and crispy form

ทำการเพิ่มรายละเอียดใน settings.py เพื่อใช้งาน crispy form และตั้งค่า media directory crispy form ใช้เพื่อสร้างเว็บฟอร์มด้วยเทคนิค generic form ซึ่งสามารถสร้างหน้าเว็บฟอร์มได้โดยใช้จำนวนบรรทัดของโปรแกรมน้อยมากจึงเหมาะกับการสอนเพื่อให้นักเรียนเห็นระบบในภาพรวมก่อนลงรายละเอียด

media directory คือตำแหน่งที่เก็บไฟล์ที่เกิดจากการ upload file ที่ได้รับจากเว็บเบราว์เซอร์

```
CRISPY_TEMPLATE_PACK = 'bootstrap3'
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
MEDIA_URL = '/media/'
```

edit model.py

ทุกระบบฐานข้อมูลจำเป็นต้องสร้างโครงสร้างฐานข้อมูล database schema ซึ่งในตัวอย่างนี้เราจะเริ่มจากการสร้างฐานข้อมูลตารางบุคคลและตารางรูปภาพโดยเขียนใน myapp/model.py โดยสองตารางนี้ยังไม่มีความเกี่ยวข้องกัน เพื่อให้นักเรียนเรียนรู้ประเภทข้อ attribute or field ของ class or table

ถ้าสังเกตุให้ดีจะเห็นการเทียบเคียงความหมายระหว่าง class กับ table และระหว่าง attribute กับ field ซึ่งเทคนิคนี้มีชื่อเรียกว่า Object Relational Mapping (ORM) โดยการใช้เทคนิคนี้เราสามารถเขียนอ่านและถามฐานข้อมูลโดยใช้การเขียนโปรแกรมเชิงวัตถุแทนการเขียนภาษา SQL

```
from __future__ import unicode_literals
from django.db import models

class Person(models.Model):
	name=models.CharField(max_length=100)
	dob=models.DateField(blank=True,null=True)
	def __str__(self):
		return self.name

class Image(models.Model):
	image=models.ImageField(upload_to='images')
	description=models.CharField(max_length=100,blank=True,null=True)
```

view.py

เพื่อตอบ request จาก web browser เราต้องสร้าง view.py ที่กำหนดการตอบกลับ โดยทำการสร้างหน้า view ซึ่งเป็นหน้าที่ทำการส่งให้ html templates ตามตัวอย่างด้านล่าง

```
from django.views.generic.edit import CreateView, UpdateView
from django.views.generic.list import ListView
from .forms import PersonForm
from .models import Person, Image

from django.shortcuts import render

def home(request):
       return render(request, 'home.html', {'key': "value" })

class CreatePersonView(CreateView):
       queryset = Person()
       template_name='person.html'
       form_class = PersonForm
       success_url = '/'

class UpdatePersonView(UpdateView):
       queryset = Person.objects.all()
       template_name='person.html'
       form_class = PersonForm
       success_url = '/'

class ListPersonView(ListView):
    model = Person
    template_name='person_list.html'
```

forms.py

เพื่อลดภาระการเขียนโปรแกรมในตัวอย่างนี้เลือกใช้ crispy form เพื่อการสร้าง html form โดยใช้โครงสร้างข้อมูลที่กำหนดใน model.py โดยให้สร้างฟอร์มเพื่อเก็บข้อมูล Person ตามตัวอย่างด้านล่าง

```
from django import forms
from django.forms import ModelForm
from crispy_forms.helper import FormHelper
from crispy_forms.layout import Submit
#from django.forms.extras.widgets import SelectDateWidget
from django.contrib.admin import widgets
import datetime
from .models import Person
```

```
class PersonForm(ModelForm):
      class Meta:
            model =  Person
            exclude=[]

            widgets = {
                  'dob': forms.DateInput(
                        attrs={
                        'type': 'date',
                        'value': datetime.datetime.now().strftime("%Y-%m-%d")
                        }, format="%Y-%m-%d"
                  ),
            }

      def __init__(self, *args, **kwargs):
            super(PersonForm, self).__init__(*args, **kwargs)
            self.helper = FormHelper()
            self.helper.add_input(Submit('submit', 'Submit'))
```

urls.py

เพื่อกำหนด URL และการตอบสนองเราจำเป็นต้องเชื่อมโยงและกำหนดการทำงานของแต่ละ URL ของระบบโดยสามารถกำหนด ซึ่งรูปแบบการกำหนดนี้เป็นของ Django version 1.x โดยรูปแบบจะเปลี่ยนใน Django verion 2.x

- เมื่อ web browser เรียกหา URL localhst/admin/ ระบบจะตอบกลับโดยอ้างอิง admin.site.urls
- เมื่อเรียก localhost/ จะเข้าไปเรียก views.home ในไฟล์ views.py
- เมื่อเรียก localhost/person/ จะสร้างข้อมูล Person ในฐานข้อมูล

```
urlpatterns = [
    url(r'^admin/', admin.site.urls),
    url(r'^$', views.home, name='home'),
    url(r'^person/$', views.CreatePersonView.as_view(), name='person'),
    url(r'^person/(?P<pk>[0-9]+)$', views.UpdatePersonView.as_view()),
    url(r'^person_list/$', views.ListPersonView.as_view(), name='person_list'),
]
```

admin.py

เพื่อจัดการฐานข้อมูลของตาราง Person และ Image ให้กำหนดรูปแบบหน้าจัดการข้อมูลสำหรับ admin

```
from django.contrib import admin
from myapp.models import Person, Image

class PersonAdmin(admin.ModelAdmin):
      list_display=[f.name for f in Person._meta.fields]

admin.site.register(Person,PersonAdmin)

class ImageAdmin(admin.ModelAdmin):
      list_display=[f.name for f in Image._meta.fields]

admin.site.register(Image,ImageAdmin)
```

migrate and run

หลังจากนั้นทำการเปลี่ยนโครงสร้างฐานข้อมูลโดยการ migrate ซึ่งระบบจะทำการเปลี่ยนโครงสร้างตารางในฐานข้อมูล และเริ่มการทำงานของ server ตามชุดคำสั่งด้านล่าง

```
python manage.py makemigrations
python manage.py migrate
python manage.py runserver
```

หลังจากนั้นนักเรียนควรสามารถใช้ web broswer เพื่อเข้าไปหน้าฐานข้อมูล โดยระบบยังไม่เสร็จในสัปดาห์หน้าจะมาพูดถึงการกำหนด URL ในขั้นถัดไป

Reference:

[1] Greenfeld, Daniel, and Greenfeld, Audrey. Two Scoops of Django: Best Practices for Django 1.611. Two Scoops Press, 2017.

[2] Tarantino, Quentin. Django unchained. Giunti, 2014.

[3] Django Software Foundation, Django Document, https://docs.djangoproject.com/en/2.1/, retrived 2018

[4] Wasit Limprasert, CS459 Django 2018, https://github.com/wasit7/cs459_django2018/tree/master/week02/week02_project

Week 03 Basic Web View-URL

เพื่อให้นักศึกษาเข้าใจการทำงานโดยสรุปของระบบในสัปดาห์นี้จึงขอทบทวนและเริ่มสร้าง view function ที่แสดงการทำงานแบบละเอียดของ Framework

views.py

views.py กำหนดการถามตอบระหว่าง web browser และ server โดยในตัวอย่างนี้ server จะตอบเวลาของเครื่องกลับไปที่ web browser โดยทำการเขียนวันเวลาทับ `%s` และตอบกลับในรูปแบบ html และ http meta data

```
from django.shortcuts import render
from django.http import HttpResponse
import datetime

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

urls.py

เราสามารถจัดการการวางเส้นทาง URL และ view ได้โดยการจัดการในไฟล์ urls.py ซึ่งเป็นการจัดการกำหนดการตอบกลับเมื่อมีการเรียก URL ที่กำหนด ในกรณีนี้เมื่อ web browser เรียก localhost/current_datetime ระบบจะทำการเรียก current_datetime() ใน views.py ซึ่งชื่อสามารถกำหนดเป็นชื่อใดๆได้

```
from django.conf.urls import url, include
from django.contrib import admin
from . import views

urlpatterns = [
    url(r'^current_datetime/', views.current_datetime ),
]
```

Week 04 Basic Model-Admin

model.py

เราสามารถกำหนดโครงสร้างฐานข้อมูลโดยในตัวอย่างนี้มีการใช้ Foreignkey ในการเชื่อมโยงข้อมูล โดยเราสามารถนำ User ซึ่งถูกสร้างขึ้นตอนสร้างโปรเจค เราจะเห็นว่าตาราง Rent มีการใช้ Foreignkey จาก Car และ User

```
from django.db import models
from django.contrib.auth.models import User
# Create your models here.
class Car(models.Model):
	model=models.CharField(max_length=20)
	detail=models.CharField(max_length=100)
	price=models.DecimalField(max_digits=10,decimal_places=2)

class Rent(models.Model):
	user=models.ForeignKey(User, on_delete=models.CASCADE)
	car=models.ForeignKey(Car, on_delete=models.CASCADE)
	start=models.DateTimeField()
	stop=models.DateTimeField()
	fee=models.DecimalField(max_digits=10,decimal_places=2)
```

admin.py

เราสามารถเพิ่มตาราง Rent และ Car ลงไปที่หน้า Django Admin เพื่อการจัดการข้อมูลได้สะดวก

```
from django.contrib import admin
from rent.models import Rent, Car

class RentAdmin(admin.ModelAdmin):
	list_display=[f.name for f in Rent._meta.fields]
admin.site.register(Rent,RentAdmin)

class CarAdmin(admin.ModelAdmin):
	list_display=[f.name for f in Car._meta.fields]
admin.site.register(Car,CarAdmin)
```

Week 05 Web Server Core Concepts

Flask

Flask เป็น python micro web framework ที่ทำหน้าที่เป็นเว็บเซิฟเวอร์ตัวหนึ่ง ซึ่งมีข้อดีที่ใช้งานง่ายเขียนง่าย แต่มีข้อเสียคือทำงานใหญ่ค่อนข้างลำบากเมื่อต้องใช้ฐานข้อมูลเพราะต้องใช้ SQL ในการเข้าถึงข้อมูล

```
from flask import Flask
from flask import render_template
app = Flask(__name__)

@app.route('/')
def home():
    return render_template('home.html', name='Wasit Limprsert')

# 'ipconfig' to check your pubic ip
# you have to disable firewall or allow incomming connection to the server
if __name__ == '__main__':
    app.run(debug=True, host='0.0.0.0')
```

Django

Django เป็น python web framework ถูกพัฒนาขึ้นในปี 2005 และผ่านการปรับปรุงอย่างต่อเนื่อง จึงทำให้สามารถเข้าถึงฐานข้อมูลได้สะดวกและปลอดภัย โดยแบ่งโปรแกรมที่จัดการส่วนต่างๆของระบบดังนี้

views.py

สร้าง ฟังก์ชั่นแสดงชื่อ Albert Einstein ในหน้า home.html ในไฟล์ views.py

```
from django.shortcuts import render

# Create your views here.
def home(request):
      return render(request, 'home.html', {'name': 'Albert Einstein'})
```

urls.py

สร้าง path ที่จะแสดงหน้า home.html โดย import มาจากไฟล์ views.py

```
from django.contrib import admin
from django.urls import path
from myapp import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', views.home, name='home')
]
```

Work with template

แสดงข้อมูลจากฐานข้อมูลผ่าน template ที่กำหนด โดยการแสดงเป็นแบบ Listview คือการแสดงข้อมูลทั้งหมดที่มีในฐานข้อมูล และสร้าง Class เพื่อแสดงข้อมูลจากฐานข้อมูล ในที่นี้คือแสดงข้อมูลรถในหน้า list.html

```
from django.http import HttpResponse
from django.views.generic.list import ListView
from .models import Car


# Create your views here.
def home(request):
       print(HttpResponse('Hello World'))
       return HttpResponse('Hello World')



class CarListView(ListView):
   model = Car
   template_name='list.html'
```

Project file structure

โครงสร้างไฟล์ในสัปดาห์นี้

```
wasitVision/week05_2: tree -F
.
└── myproject/
    ├── db.sqlite3
    ├── manage.py*
    ├── myproject/
    │   ├── __init__.py
    │   ├── __pycache__/
    │   │   ├── __init__.cpython-35.pyc
    │   │   ├── settings.cpython-35.pyc
    │   │   ├── urls.cpython-35.pyc
    │   │   └── wsgi.cpython-35.pyc
    │   ├── settings.py
    │   ├── urls.py
    │   └── wsgi.py
    └── rent/
        ├── admin.py
        ├── apps.py
        ├── __init__.py
        ├── migrations/
        │   ├── 0001_initial.py
        │   ├── __init__.py
        │   └── __pycache__/
        │       ├── 0001_initial.cpython-35.pyc
        │       └── __init__.cpython-35.pyc
        ├── models.py
        ├── __pycache__/
        │   ├── admin.cpython-35.pyc
        │   ├── __init__.cpython-35.pyc
        │   ├── models.cpython-35.pyc
        │   └── views.cpython-35.pyc
        ├── templates/
        │   ├── list.html
        │   └── login.html
        ├── tests.py
        └── views.py
```

Week 06 Static Files

Server static files

ในเว็บแอปพลิเคชันจำเป็นต้องเปิดบริการไฟล์เพื่อให้ผู้ใช้สามารถเข้าถึงไฟล์ที่ไม่มีการเปลี่ยนแปลง เช่น ภาพโลโก้ ไฟล์ JavaScript Library และ HTML โดยวิธีเปิดการเข้าถึงไฟล์เหล่านี้จะแตกต่างจากที่ทำเราทำใน view.py เราจำเป็นต้องจัดเก็บไฟล์ไว้ที่ไดเร็กทอรีที่อนุญาตให้ผู้ใช้ทั่วไปเข้าถึง และทำการกำหนด STATIC_URL เพื่อให้ผู้ใช้เข้าถึงไฟล์ image, css, js และไฟล์ที่ไม่เปลี่ยนแปลงประเภทอื่นๆ ส่วนไฟล์ที่เกิดจากการเขียนโดยผู้ใช้เช่นการอัพโหลดไฟล์เข้าเว็บบอร์ดหรือแอปพลิเคชันอื่นเราจะจัดเก็บไฟล์ไว้ที่ /media โดยสามารถเข้าถึงได้ที่ MEDIA_URL

Static file เป็นไฟล์แบบคงที่ เช่น รูปภาพ, css, javascript เป็นต้น ในบทนี้จะแสดงวิธีที่ง่ายที่สุด คือ การเพิ่ม URL ที่ใช้เก็บข้อมูลไฟล์ static และ เพิ่ม url ของ media สำหรับไฟล์ที่มีการอัพโหลด หรือลบไฟล์ลงในไฟล์ url.py ตาม รูปด้านล่าง

urls.py

```
from django.urls import path

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

เราเพิ่ม code ส่วนนี้ลงในไฟล์ urls.py เพื่อให้ผู้ใช้สามารถเข้าถึง static file และ media file

```
from django.conf import settings
from django.conf.urls.static import static
urlpatterns += static(settings.STATIC_URL, document_root=settings.STATIC_ROOT)
urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

File structure

โครงสร้างการจัดเก็บ static และ media file โดยมี myproject/static และ myproject/media เป็นที่จัดเก็บไฟล์สองประเภทนี้

```
wasitVision/static_demo: tree -F
.
└── myproject/
    ├── db.sqlite3
    ├── manage.py*
    ├── media/
    │   ├── cars/
    │   │   └── Screenshot_from_2018-02-12_12-36-29.png
    │   └── media.gif
    ├── myapp/
    │   ├── admin.py
    │   ├── apps.py
    │   ├── __init__.py
    │   ├── migrations/
    │   │   ├── 0001_initial.py
    │   │   ├── 0002_auto_20180228_0741.py
    │   │   ├── __init__.py
    │   │   └── __pycache__/
    │   │       ├── 0001_initial.cpython-35.pyc
    │   │       ├── 0002_auto_20180228_0741.cpython-35.pyc
    │   │       └── __init__.cpython-35.pyc
    │   ├── models.py
    │   ├── __pycache__/
    │   │   ├── admin.cpython-35.pyc
    │   │   ├── __init__.cpython-35.pyc
    │   │   └── models.cpython-35.pyc
    │   ├── tests.py
    │   └── views.py
    ├── myproject/
    │   ├── __init__.py
    │   ├── __pycache__/
    │   │   ├── __init__.cpython-35.pyc
    │   │   ├── settings.cpython-35.pyc
    │   │   ├── urls.cpython-35.pyc
    │   │   └── wsgi.cpython-35.pyc
    │   ├── settings.py
    │   ├── urls.py
    │   └── wsgi.py
    └── static/
        └── hello.txt

10 directories, 28 files
```

Week 07 Notebook

เนื่องจากเว็บแอปพลิเคชันของระบบมีความซับซ้อนของข้อมูลการแก้ไขไฟล์ในระบบ server เพื่อทดสอบผลลพทธ์จึงทำได้ลำบาก วิธีที่สะดวกกว่าคือการทดลองส่วนที่สนใจใน Jupyter notebook ซึ่งเป็น Interactive evironment สามาถแก้ไขโค๊ดและทดลองการทำงานก่อนที่จะใช้โค๊ดนั้นใน Server ซึ่ง Jupyter notebook เป็นเครื่องมือที่ช่วยในการตรวจสอบ debug ฐานข้อมูล

model.py

```
from django.db import models

# Create your models here.
class Car(models.Model):
	model=models.CharField(max_length=100)
	detail=models.CharField(max_length=100)
	price=models.DecimalField(max_digits=10, decimal_places=2)
	def __str__(self):
		return "id: %s, model: %s, price: %s"%(self.id, self.model,
self.price)

class Customer(models.Model):
	first_name=models.CharField(max_length=100)
	last_name=models.CharField(max_length=100)
	phone=models.CharField(max_length=20)
	def __str__(self):
		return "id: {}, {}".format(self.id, self.first_name)

class Rent(models.Model):
	car=models.ForeignKey('Car',on_delete=models.CASCADE)
	customer=models.ForeignKey('Customer',on_delete=models.CASCADE)
	start=models.DateTimeField(null=True)
	stop=models.DateTimeField(null=True)
```

To run

gเพื่อเปิดการทำงานของ Jupyter notebook gikใช้คำสั่งด้านล่างนี้ในการเรียก interactive notebook

```
python manage.py shell_plus --notebook
```

Create a Class diagram

ในระบบที่มีขนาดใหญ่จำนวนตารางในฐานข้อมูลมากกว่า 20 ตาราง เราจำเป็นที่จะต้องมีเครื่องมือในการเข้าใจโครงสร้างการเชื่อมโยงของข้อมูล โดย django-extension มีเครื่องมือในการสร้าง Class Diagram ของฐานข้อมูลปัจจุบันของระบบ

To create a diagram please read a documents of django-extensions here.

```
python ./manage.py graph_models --pydot -a -g -o others/classdiagram.png
```

Week 08 Basic ORM

เนื่องจากการพัฒนาแอปพลิเคชันจะมีความซับซ้อนในการถามและดึงข้อมูลจากฐานข้อมูล ซึ่งในกรณีเทคโลโลยีดังเดิมจะใช้ Query Language เพื่อการถามข้อมูลจากฐานข้อมูล อย่างไรก็ตามเมื่อแอปลเคชันมีขนาดใหญและมีความซับซ้อนมากขึ้น การใช้ Query Language จะเป็นอุปสรรคในการพัฒนาระบบและดูแลแอปพลิเคชัน Django webframework จึงพัฒนา Object-relational mapping (ORM) ซึ่งเป็นการใช้การเขียนโปรแกรมแบบ Object Oriented Programming เพื่อการถามและดึงข้อมูลจากฐานข้อมูล โดย table in database คือ class in OOP และ field in table คือ attribute in class การใช้ ORM จะมีความยากในการเริ่มใช้แต่เมื่อชำนวญและเข้าใจเครื่องมือต่างๆที่ ORM จัดเตรียมไว้ให้จะสามารถเขียน Query จากหลายสิบบรรทัดเหลือเพียงโค๊ด ORM เพื่อไม่กี่บรรทัด

ORM

Select all cars

ตัวอย่างด้านล่างเป็นการใช้ ORM ในการเลือกรายการข้อมูลรถทั้งหมด โดยการเลือกแสดงข้อมูลรถทั้งหมดที่มีในฐานข้อมูลสามารถทำได้โดยคำสั้ง

SQL: SELECT * FROM "myapp_car" หรือใช้

ORM: Car.objects.all()

```
In [1]:
Car.objects.all()
#ORM: Object Relational mapping
Out[1]:
<QuerySet [<Car: id: 1, model: Vios, price: 500000>, <Car: id: 2, model:
Camry, price: 800000>, <Car: id: 3, model: Jazz, price: 400000>]>
In [2]:
print(Car.objects.all().query)
SELECT "myapp_car"."id", "myapp_car"."model", "myapp_car"."detail",
"myapp_car"."price" FROM "myapp_car"
In [3]:
for i in Car.objects.all():
   print(i.model, i.detail, i.price)
Vios medium price 500000
Camry High price 800000
Jazz low price entry car 400000
```

get a car by id

การเลือกแสดงข้อมูลผู้ใช้ตาม id ที่กำหนด

ORM: Customer.objects.get(id=1)

```
In [4]:
Customer.objects.get(id=1)
Out[4]:
<Customer: id: 1, Albert>
```

get cars by filter

การเลือกแสดงข้อมูลรถจากเงื่อนไขที่กำหนด

SQL: SELECT * FROM "myapp_car" WHERE "myapp_car"."price" >= 500000

ORM: Car.objects.filter(price__gt=500000)

```
In [5]:
Car.objects.filter(price__lte=500000)
Out[5]:
<QuerySet [<Car: id: 1, model: Vios, price: 500000>, <Car: id: 3, model:
Jazz, price: 400000>]>
In [6]:
Car.objects.filter(price=500000)
Out[6]:
<QuerySet [<Car: id: 1, model: Vios, price: 500000>]>
In [7]:
Car.objects.filter(price__gt=500000)
Out[7]:
<QuerySet [<Car: id: 2, model: Camry, price: 800000>]>
In [8]:
print( Car.objects.filter(price__gte=500000).query )
SELECT "myapp_car"."id", "myapp_car"."model", "myapp_car"."detail",
"myapp_car"."price" FROM "myapp_car" WHERE "myapp_car"."price" >= 500000
```

Relation

ORM สามาถสืบค้นแบบกราฟขอข้อมูลที่จัดเก็บไว้ใน model ที่มีความสัมพันธ์กันได้

ORM can resolve forward and reward relation

Car

ID	MODEL	DETAIL	PRICE
3	Jazz	low price entry car	400000.00
2	Camry	High price	800000.00
1	Vios	medium price	500000.00

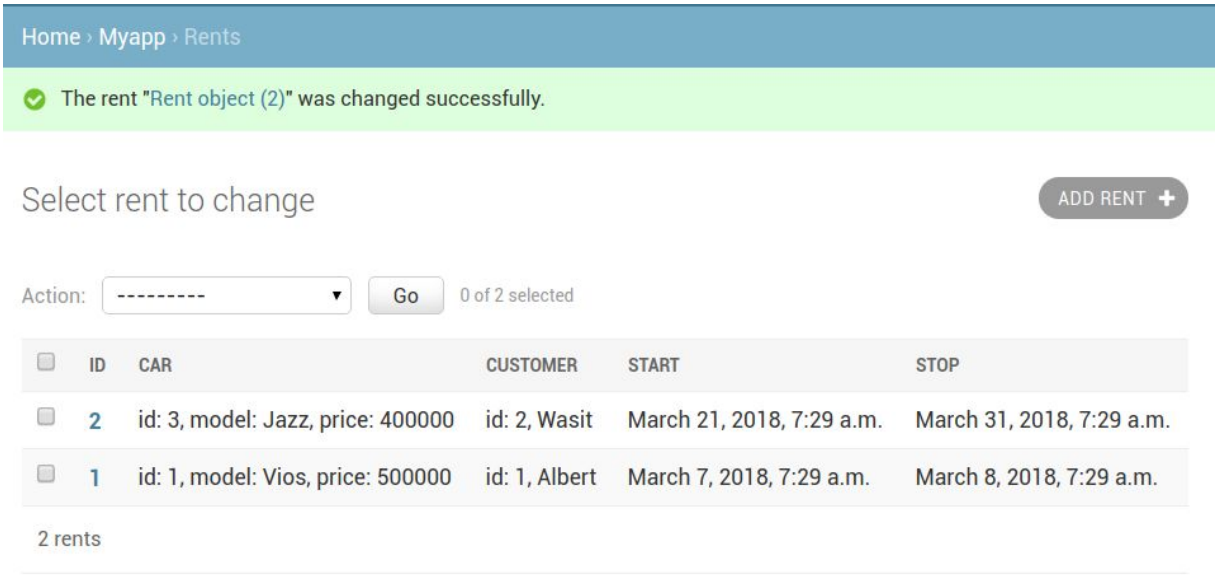
3 cars

Customer

ID	FIRST NAME	LAST NAME	PHONE
2	Wasit	Limprasert	0822222222
1	Albert	Einstein	0888888888

2 customers

Rent



Model Rent มีความสัมพันธ์ระหว่าง Car และ Customer มาเชื่อมกัน สามารถใช้ ORM ในการแสดงข้อมูลผ่านตารางที่มีความสัมพันธ์นี้ได้

```
In [9]:
Rent.objects.get(id=2).car
Out[9]:
<Car: id: 3, model: Jazz, price: 400000>
In [10]:
Rent.objects.get(id=2).customer
Out[10]:
<Customer: id: 2, Wasit>
In [11]:
Rent.objects.filter(car__price__lte=400000)
Out[11]:
<QuerySet [<Rent: Rent object (2)>]>
In [12]:
Rent.objects.filter(car__price__lte=500000)
Out[12]:
<QuerySet [<Rent: Rent object (1)>, <Rent: Rent object (2)>]>
In [13]:
Rent.objects.filter(car__price__lte=500000, customer__first_name='Albert')
#AND conjunction
Out[13]:
<QuerySet [<Rent: Rent object (1)>]>
```

เช่น ถ้าต้องการหาบันทึกการเช่าทีรถมีราคาน้อยกว่า 500,000 บาท และผู้เช้ามีชื่อต้นว่า Albert เราสามารถใช้ ORM: Rent.objects.filter(car__price__lte=500000, customer__first_name='Albert')

การบ้าน: จงหา SQL ที่ทำการถามข้อมูลจากฐานข้อมูลที่เทียบเท่ากับคำสั่ง ORM ด้านบน

Week 09 REST API

setting.py

นำเข้า django rest framework เข้าสู่โปรเจคโดย django rest framework จะทำหน้าที่สร้าง rest framework อย่างง่ายให้กับโมเดลนั้น ๆ ทั้งสร้าง อ่าน แก้ไข และลบข้อมูล

```
INSTALLED_APPS = [
    ...,
    'rest_framework',
]
```

routers.py

สร้างไฟล์ router เพื่อระบุจุดเชื่อมต่อกับส่วนต่อประสานแอปพลิเคชัน เช่น router.register(r'car', CarViewSet) ระบุว่า สามารถเชื่อมต่อส่วนประสานแอปพลิเคชัน CarViewSet ผ่าน url ที่ /car ได้

```
from rest_framework import routers, serializers, viewsets
from myapp.viewsets import CustomerViewSet, CarViewSet, RentViewSet
router = routers.DefaultRouter()
router.register(r'customer', CustomerViewSet)
router.register(r'car', CarViewSet)
router.register(r'rent', RentViewSet)
```

serializers.py

Serializer ทำหน้าแปลงข้อมูลที่ซับซ้อนเช่น QuerySet ให้เป็นประเภทข้อมูลทั่วไปของ python ส่งผลทำให้สามารถส่งผ่าน json หรือ xml ได้ง่ายขึ้น และ Serializer เองสามารถทำให้ข้อมูลที่รับเข้าที่เป็นประเภททั่วไปของ python ให้กลายเป็นข้อมูลที่ซับซ้อนขึ้นได้ด้วย

```
from myapp.models import Customer, Car, Rent
from rest_framework import routers, serializers, viewsets


class CustomerSerializer(serializers.ModelSerializer):
    class Meta:
        model = Customer
        fields = '__all__'

class CarSerializer(serializers.ModelSerializer):
    class Meta:
        model = Car
        fields = '__all__'
```

```
class RentSerializer(serializers.ModelSerializer):
    class Meta:
        model = Rent
        fields = '__all__'
```

viewsets.py

Viewset เป็นตัวกำหนดการเข้าถึงข้อมูลเช่น viewsets.ModelViewSet จะให้ผู้ใช้เข้าถึงข้อมูลด้วยวิธีตาม HTTP ทั้งหมดคือ GET, POST, PUT, PATCH และ DELETE เป็นต้น Viewset จำเป็นต้องกำหนดข้อมูลที่จะใช้อ้างอิงจาก Model ไหนและ Serializer ของ Model นั้น ๆ

```
from rest_framework import routers, serializers, viewsets
from myapp.models import Customer, Car, Rent
from myapp.serializers import CustomerSerializer, CarSerializer, RentSerializer

class CustomerViewSet(viewsets.ModelViewSet):
    queryset = Customer.objects.all()
    serializer_class = CustomerSerializer

class CarViewSet(viewsets.ModelViewSet):
    queryset = Car.objects.all()
    serializer_class = CarSerializer

class RentViewSet(viewsets.ModelViewSet):
    queryset = Rent.objects.all()
    serializer_class = RentSerializer
```

Week 10 Agile Management

Agile Software Development คือวิธีการจัดการโปรเจคยุคใหม่ที่มีสาระคำคัญดังนี้

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

Scrum

การจัดการโปรเจคแบบ Agile มีหลายรูปแบบโดยในคลาสนี้หยิบเทคนิคการใช้บอร์ดและการจัดการโปรเจคที่เรียกว่า Scrum โดยการดำเนินการโปรเจคมีแนวทางดังนี้

1. Product owner มีไอเดียที่จะผลิต product
2. Product owner เขียนลักษณะการใช้งาน product ในรูปแบบของ User story
3. ทีมที่ประกอบด้วย Scrum master, Developer และ Product owner ประชุมเพื่อจัดเรียงลำดับความสำคัญของแต่ละรายการของ User story โดยมีการประมาณเวลาจำนวนวันที่จะใช้ในการพัฒนา (manday) โดยการพัฒนาโปรเจคจะแบ่งเป็นส่วนย่อยๆเรียกว่า Splint โดยระยะเวลาควรอยู่ในช่วง 2-3 สัปดาห์
4. ระหว่างที่ดำเนินการพัฒนางานใน Splint จะมีการประชุมทุกวันในเวลาประจำโดยการประชุมจะใช้เวลาไม่เกิน 15 นาที โดยมีสาระสำคัญของแต่ละคนในทีมดังนี้
    a. เมื่อวานทำอะไรไปบ้าง
    b. ติดปัญหาอะไรไหม
    c. วันนี้วางแผนจะทำอะไร
5. เมื่อสิ้นสุดระยะเวลา Splint ทุกคนในทีมจะเข้าประชุมกับหุ้นส่วนและผู้ลงทุนเพื่อพิจารณาว่าโปรดักพร้อมเข้าสู่ตลาดหรือไม่ และมีอะไรต้องพัฒนาอีกบ้าง
6. ถ้ามีส่วนที่ต้องพัฒนาเพิ่ม ทีมจะกลับมาคุยถึงเรื่องที่ประสบความสำเร็จและปัญหาที่เกิดขึ้น ก่อนที่จะวางแผน Splint ถัดไป โดยทำซ้ำข้อที่ 3

User Story

มีลักษณะคล้าย use-case ของ UML โดยแตกต่างตรงที่ User Story จะเขียนเป็นประโยคภาษาอังกฤษโดยมีองค์ประกอบของโปรโยคดังนี้

[Who] can [do something] at [some place] for [some reason]

ตัวอย่าง User story เช่น

- Customer can select a car from list for renting
- Customer can review the total cost before payment
- Shopkeeper can confirm rent request
- Shopkeeper can add a new car with details
- Shopkeeper can see a dialy report and maonthly report for planing

โดยแต่ละ User Story จะถูกนำไปแยกออกเป็น Task ย่อยๆสำหรับให้ Developer พัฒนาเป็นส่วนย่อย โดยแต่ละ Task จะมีการประมาณเวลาที่ใช้และทำการบันทึกเวลาที่ใช้ในการพัฒนา ข้อมูลที่ได้นำไปวาดเป็น Burn Down Chart

Scrum Board

วิธีการกระจายงานของ Scrum จะใช้ Scrum Board โดยงานแบ่งเป็น To-do Doing และ Done งานที่เข้ามาใหม่จะถูกเขียนลงบนกระดาษโน๊ต(การ์ด) และแปะอยู่ที่ To-do โดยงานที่สำคัญอยู่ด้านบน

เมื่อ Developer เสร็จงานเดิมจะรับงานใหม่เขาสามารถมาหยิบการ์ดโดยควรหยิบงานที่สำคัญด้านบนก่อน และเขียนชื่อตัวเองลงบนการ์ดก่อนที่จะย้ายไปอยู่ที่ Doing และเมื่อเสร็จจะย้ายไป Done โดยระหว่าง Spint ถ้าทีมเห็นการ์ดหยุดค้างอยู่นานทีมสามารถเข้ามาช่วยเพื่อให้งานเสร็จทันเวลา

Week 11 Data Analytics with Pandas

01 Import Data

Setting some useful variables

```
In [1]:
import os
import re
APP_NAME="myapp"
ROOT_PATH=os.path.abspath(".")
print "ROOT_PATH: %s"%ROOT_PATH

input_filename=os.path.join(ROOT_PATH,"rent_input.xls")
print input_filename
import datetime
Output [1]:
ROOT_PATH: C:\Users\Wasit\Documents\GitHub\cs459_final\myproject
C:\Users\Wasit\Documents\GitHub\cs459_final\myproject\rent_input.xls
```

Loading Car detail from the 1st sheet from the input excel

```
In [2]:
import pandas as pd
cvt={
        0:int,
        1:unicode,
        2:unicode,
        3:float,
        4:unicode,
        5:int,
    }
df_car=pd.read_excel(io=input_filename,sheetname=0,converters=cvt)
In [3]:
print df_car
Output: [3]
  ID    CarMaker CarModel  CarPrice   CarColor  CarYear
0   1  Mitsubishi     L200    9995.0        red     2001
1   2        Mini   Cooper   12500.0        red     2005
2   3         TVR   Tuscan   18000.0       blue     2003
3   4         BMW       Z3   13995.0     silver     2002
4   5      Toyota   Celica    4665.0  dark blue     1997
5   6        Audi       TT   21995.0     silver     2005
6   7    Mercedes     E320   15495.0      green     2004
```

Example for iterate over all row in the sheet

```
In [4]:
for k,i in df_car.iterrows():
    print "k: %s, i:%s\n%s"%(k, i, "-"*30)
Output [4]:
k: 0, i:ID                  1
CarMaker    Mitsubishi
CarModel          L200
CarPrice          9995
CarColor           red
CarYear           2001
Name: 0, dtype: object
------------------------------
k: 1, i:ID              2
CarMaker      Mini
CarModel    Cooper
CarPrice     12500
CarColor       red
CarYear       2005
Name: 1, dtype: object
------------------------------
k: 2, i:ID              3
CarMaker       TVR
CarModel    Tuscan
CarPrice     18000
CarColor      blue
CarYear       2003
Name: 2, dtype: object
------------------------------
k: 3, i:ID              4
CarMaker       BMW
CarModel        Z3
CarPrice     13995
CarColor    silver
CarYear       2002
Name: 3, dtype: object
------------------------------
k: 4, i:ID                  5
CarMaker       Toyota
CarModel       Celica
CarPrice         4665
CarColor    dark blue
CarYear          1997
Name: 4, dtype: object
------------------------------
k: 5, i:ID              6
CarMaker      Audi
CarModel        TT
CarPrice     21995
CarColor    silver
```

```
CarYear       2005
Name: 5, dtype: object
------------------------------
k: 6, i:ID                 7
CarMaker    Mercedes
CarModel         E320
CarPrice        15495
CarColor        green
CarYear          2004
Name: 6, dtype: object
------------------------------
```

Each row contains 6 columns as following

```
In [5]:
i
Out[5]:
ID                  7
CarMaker    Mercedes
CarModel         E320
CarPrice        15495
CarColor        green
CarYear          2004
Name: 6, dtype: object
```

Uploading the row i into database

```
In [6]:
import pytz
year=datetime.datetime(year=i['CarYear'], month=1, day=1, tzinfo=pytz.UTC)
kargs={
    'maker':i['CarMaker'],
    'price':i['CarPrice'],
    'model':i['CarModel'],
    'year': year
}
car, created = Car.objects.update_or_create(
                    id=i['ID'],
                    defaults=kargs
                )
```

Now uploading every rows

```
In [7]:
for k,i in df_car.iterrows():
    year=datetime.datetime(year=i['CarYear'], month=1,day=1, tzinfo=pytz.UTC)
    kargs={
        'maker':i['CarMaker'],
        'price':i['CarPrice'],
        'model':i['CarModel'],
        'year': year
    }
    car, created = Car.objects.update_or_create(
                        id=i['ID'],
                        defaults=kargs
                    )
```

Then loading customer from the 2nd sheet

```
In [8]:
import pandas as pd
cvt={
        0:unicode,
        1:unicode,
        2:unicode,
        3:unicode,
        4:unicode,
        5:unicode,
        6:unicode
    }
df_customer=pd.read_excel(io=input_filename,sheetname=1,converters=cvt)
keys=df_customer.keys()
print keys
out [8]:
Index([u'ID', u'ClientFirstName', u'ClientLastName', u'ClientAddress',
       u'Postcode', u'Tel', u'Email'],
      dtype='object')
```

And uploading the customer

```
In [9]:
for k,i in df_customer.iterrows():
    kargs={
        'first_name':i[keys[1]],
        'last_name':i[keys[2]],
        'Address':i[keys[3]],
        'postcode':i[keys[4]],
        'telephone':i[keys[5]],
        'email':i[keys[6]]
    }
    customer, created = Customer.objects.update_or_create(
                        id=i['ID'],
                        defaults=kargs
                    )
```

Finally Loading the 3rd sheet

```
In [10]:
import pandas as pd
df_rent=pd.read_excel(io=input_filename,sheetname=2)
keys=df_rent.keys()
print keys
Index([u'ID', u'RentDate', u'ServiceCost', u'ReturnDate', u'ClientID',
       u'CarID'],
      dtype='object')
In [11]:
df_rent
```

Out[11]:

	ID	RentDate	ServiceCost	ReturnDate	ClientID	CarID
0	1	2014-03-12	549.75	2014-03-17	1	1
1	2	2014-03-12	1050.00	2014-03-20	2	2
2	3	2014-03-13	1310.00	2014-03-20	3	3
3	4	2014-03-17	425.00	2014-03-20	1	2
4	5	2014-03-20	189.95	2014-03-21	4	4
5	6	2014-03-20	50.00	2014-03-20	2	5
6	7	2014-03-20	269.95	2014-03-21	2	6
7	8	2014-03-21	514.85	2014-03-24	5	7

8	9	2014-03-24	549.75	2014-03-29	6	1
9	10	2014-03-29	50.00	2014-03-29	1	2
10	11	2014-03-29	500.00	2014-03-29	5	3
11	12	2014-03-29	500.00	2014-03-29	7	4
12	13	2014-03-30	430.00	2014-03-29	2	1
13	14	2014-03-30	430.00	2014-03-29	6	3
14	15	2014-03-30	430.00	2014-03-29	1	4
15	16	2014-03-30	430.00	2014-03-29	5	5
16	17	2014-03-30	430.00	2014-03-29	6	6

And uploading Rent records to the database

```
In [12]:
for k,i in df_rent.iterrows():
    utc=pytz.timezone('UTC')
    kargs={
        'rent_date': utc.localize(i['RentDate']),
        'return_date':utc.localize(i['ReturnDate']),
        'cost':i['ServiceCost'],
        'car': Car.objects.get(id=i['CarID']),
        'customer': Customer.objects.get(id=i['ClientID']),
    }
    customer, created = Rent.objects.update_or_create(
                        id=i['ID'],
                        defaults=kargs
                    )
```

Code explain

To convert from naive-datetime to time-zone-aware-datetime

```
import pytz
utc=pytz.timezone('UTC')
utc.localize( your_datetime )
```

Week 12 ORM Query

Query Pattern

- What is total rental cost between 13/03/2014-24/03/2014?
- How much money collected from the car id=2?

Getting a record by id

```
In [2]:
c=Customer.objects.get(id=2)
print(c)

Customer object (2)
```

Getting all records from table Customer

```
In [3]:
Customer.objects.all()
Out[3]:
<QuerySet [<Customer: Customer object (1)>, <Customer: Customer object (2)>,
<Customer: Customer object (3)>, <Customer: Customer object (4)>, <Customer:
Customer object (5)>, <Customer: Customer object (6)>, <Customer: Customer
object (7)>, <Customer: Customer object (8)>, <Customer: Customer object (9)>,
<Customer: Customer object (10)>]>
```

แปลง ORM เป็น SQL

```
In [3]:
# SQL command
print Customer.objects.all().query
SELECT "myapp_customer"."id", "myapp_customer"."first_name",
"myapp_customer"."last_name", "myapp_customer"."Address",
"myapp_customer"."postcode", "myapp_customer"."telephone",
"myapp_customer"."email" FROM "myapp_customer"
```

Filter records within range

```
In [4]:
from datetime import datetime
import pytz
utc=pytz.timezone('UTC')
start_date = utc.localize( datetime.strptime('2014-03-13','%Y-%m-%d') )
stop_date = utc.localize( datetime.strptime('2014-03-24','%Y-%m-%d') )
```

```
In [5]:
Rent.objects.filter(rent_date__range=[start_date, stop_date])
Out[5]:
[<Rent: id: 3>, <Rent: id: 4>, <Rent: id: 5>, <Rent: id: 6>, <Rent: id: 7>,
<Rent: id: 8>, <Rent: id: 9>]
```

```
In [6]:
# SQL command
print Rent.objects.filter(rent_date__range=[start_date, stop_date ]).query
Out[6]:
SELECT "myapp_rent"."id", "myapp_rent"."rent_date", "myapp_rent"."return_date",
"myapp_rent"."cost", "myapp_rent"."car_id", "myapp_rent"."customer_id" FROM
"myapp_rent" WHERE "myapp_rent"."rent_date" BETWEEN 2014-03-13 00:00:00 AND
2014-03-24 00:00:00
```

Filter less_than_or_equal (__lte)

```
In [7]:
# rent that happended before or equal 13 March 2014
Rent.objects.filter(rent_date__lte=start_date)
Out[7]:
[<Rent: id: 1>, <Rent: id: 2>, <Rent: id: 3>]
```

```
In [8]:
# SQL command
print Rent.objects.filter(rent_date__lte=start_date).query
SELECT "myapp_rent"."id", "myapp_rent"."rent_date", "myapp_rent"."return_date",
"myapp_rent"."cost", "myapp_rent"."car_id", "myapp_rent"."customer_id" FROM
"myapp_rent" WHERE "myapp_rent"."rent_date" <= 2014-03-13 00:00:00
```

Filter greater than (__gt)

```
In [9]:
# rent that happended after 13 March 2014
Rent.objects.filter(rent_date__gt=start_date)
Out[9]:
[<Rent: id: 4>, <Rent: id: 5>, <Rent: id: 6>, <Rent: id: 7>, <Rent: id: 8>,
<Rent: id: 9>, <Rent: id: 10>, <Rent: id: 11>, <Rent: id: 12>, <Rent: id: 13>,
<Rent: id: 14>, <Rent: id: 15>, <Rent: id: 16>, <Rent: id: 17>]
```

```
In [10]:# SQL command
```

```
print Rent.objects.filter(rent_date__gt=start_date).query
Out [10]:
SELECT "myapp_rent"."id", "myapp_rent"."rent_date", "myapp_rent"."return_date",
"myapp_rent"."cost", "myapp_rent"."car_id", "myapp_rent"."customer_id" FROM
"myapp_rent" WHERE "myapp_rent"."rent_date" > 2014-03-13 00:00:00
```

What is total rental cost between 13/03/2014-24/03/2014?

Naive solution (but slow)

```
In [11]:
%%timeit -n10
total=0
q=Rent.objects.filter(rent_date__range=[start_date, stop_date])
for i in q:
   total=total + i.cost
Out [11]:
10 loops, best of 3: 2.33 ms per loop
```

Better by Using "aggregration()"

```
In [12]:
%%timeit -n10
from django.db.models import Sum, Max, Min, Avg
Rent.objects.filter(rent_date__range=[start_date,
stop_date]).aggregate(Sum('cost'))
Out [12]:
10 loops, best of 3: 879 µs per loop
```

```
In [13]:
q=Rent.objects.filter(rent_date__range=[start_date, stop_date])
r=q.aggregate(Sum('cost'))
r
Out[13]:
{'cost__sum': Decimal('3309.50')}
```

```
In [14]:
Rent.objects.filter(rent_date__range=[start_date,
stop_date]).aggregate(Max('cost'))
Out[14]:
{'cost__max': Decimal('1310.00')}
```

Annotate Count

```
In [15]:
from django.db.models import Count
In [16]:
```

```
q=Car.objects.annotate(Count("rent"))
In [17]:
q[0].rent__count
Out[17]:
3
In [18]:
for i in q:
    print "rent__count:%s car:%s"%(i.rent__count, i)
Out [18]:
rent__count:3 car:id: 1, Mitsubishi L200
rent__count:3 car:id: 2, Mini Cooper
rent__count:3 car:id: 3, TVR Tuscan
rent__count:3 car:id: 4, BMW Z3
rent__count:2 car:id: 5, Toyota Celica
rent__count:2 car:id: 6, Audi TT
rent__count:1 car:id: 7, Mercedes E320
```

```
In [19]:
print Car.objects.annotate(Count("rent")).query
Out [19]:
SELECT "myapp_car"."id", "myapp_car"."maker", "myapp_car"."price",
"myapp_car"."model", "myapp_car"."year", COUNT("myapp_rent"."id") AS
"rent__count" FROM "myapp_car" LEFT OUTER JOIN "myapp_rent" ON ("myapp_car"."id"
= "myapp_rent"."car_id") GROUP BY "myapp_car"."id", "myapp_car"."maker",
"myapp_car"."price", "myapp_car"."model", "myapp_car"."year"
```

Reverse relation

```
In [20]:
Car.objects.get(id=2)
Out[20]:
<Car: id: 2, Mini Cooper>
In [21]:
Car.objects.get(id=2).rent_set.all()
Out[21]:
[<Rent: id: 2>, <Rent: id: 4>, <Rent: id: 10>]
In [22]:
# SQL command
print Car.objects.get(id=2).rent_set.all().query
Out [22]:
SELECT "myapp_rent"."id", "myapp_rent"."rent_date", "myapp_rent"."return_date",
"myapp_rent"."cost", "myapp_rent"."car_id", "myapp_rent"."customer_id" FROM
"myapp_rent" WHERE "myapp_rent"."car_id" = 2
```

How much money collected from the car id=2?

Reverse relation (slow)

```
In [23]:
%%timeit -n1
sum_cost=Car.objects.get(id=2).rent_set.all().aggregate(Sum('cost'))
print sum_cost
Out [23]:
{'cost__sum': Decimal('1525.00')}
{'cost__sum': Decimal('1525.00')}
{'cost__sum': Decimal('1525.00')}
1 loop, best of 3: 2.03 ms per loop
In [24]:
print Car.objects.get(id=2).rent_set.all().query
Out [24]:
SELECT "myapp_rent"."id", "myapp_rent"."rent_date", "myapp_rent"."return_date",
"myapp_rent"."cost", "myapp_rent"."car_id", "myapp_rent"."customer_id" FROM
"myapp_rent" WHERE "myapp_rent"."car_id" = 2
```

Forward relation

```
In [25]:
%%timeit -n1
sum_cost=Rent.objects.filter(car__id=2).aggregate(Sum('cost'))
print sum_cost
Out [25]:
{'cost__sum': Decimal('1525.00')}
{'cost__sum': Decimal('1525.00')}
{'cost__sum': Decimal('1525.00')}
1 loop, best of 3: 2.27 ms per loop
In [26]:
print Rent.objects.filter(car__id=2).query
Out [26]:
SELECT "myapp_rent"."id", "myapp_rent"."rent_date", "myapp_rent"."return_date",
"myapp_rent"."cost", "myapp_rent"."car_id", "myapp_rent"."customer_id" FROM
"myapp_rent" WHERE "myapp_rent"."car_id" = 2
```

Find total income for each car

```
In [27]:
q=Car.objects.annotate(Sum("rent__cost"))
for i in q:
    print "income:%s car:%s"%(i.rent__cost__sum,i)
```

```
Out [27]:
income:1529.50 car:id: 1, Mitsubishi L200
income:1525.00 car:id: 2, Mini Cooper
income:2240.00 car:id: 3, TVR Tuscan
income:1119.95 car:id: 4, BMW Z3
income:480.00 car:id: 5, Toyota Celica
income:699.95 car:id: 6, Audi TT
income:514.85 car:id: 7, Mercedes E320
```

Q: Why do we need to use revese relation?

A: Sometimes we need to iterate over all cars to get total cost of each car.

```
In [28]:
%%timeit -n1
for i in Car.objects.all():
    print "%s\n    %s"%( i, i.rent_set.all().aggregate(Sum('cost')) )
Out [28]:
id: 1, Mitsubishi L200
    {'cost__sum': Decimal('1529.50')}
id: 2, Mini Cooper
    {'cost__sum': Decimal('1525.00')}
id: 3, TVR Tuscan
    {'cost__sum': Decimal('2240.00')}
id: 4, BMW Z3
    {'cost__sum': Decimal('1119.95')}
id: 5, Toyota Celica
    {'cost__sum': Decimal('480.00')}
id: 6, Audi TT
    {'cost__sum': Decimal('699.95')}
id: 7, Mercedes E320
    {'cost__sum': Decimal('514.85')}
id: 1, Mitsubishi L200
    {'cost__sum': Decimal('1529.50')}
id: 2, Mini Cooper
    {'cost__sum': Decimal('1525.00')}
id: 3, TVR Tuscan
    {'cost__sum': Decimal('2240.00')}
id: 4, BMW Z3
    {'cost__sum': Decimal('1119.95')}
id: 5, Toyota Celica
    {'cost__sum': Decimal('480.00')}
id: 6, Audi TT
    {'cost__sum': Decimal('699.95')}
id: 7, Mercedes E320
    {'cost__sum': Decimal('514.85')}
id: 1, Mitsubishi L200
    {'cost__sum': Decimal('1529.50')}
id: 2, Mini Cooper
    {'cost__sum': Decimal('1525.00')}
id: 3, TVR Tuscan
    {'cost__sum': Decimal('2240.00')}
```

```
id: 4, BMW Z3
    {'cost__sum': Decimal('1119.95')}
id: 5, Toyota Celica
    {'cost__sum': Decimal('480.00')}
id: 6, Audi TT
    {'cost__sum': Decimal('699.95')}
id: 7, Mercedes E320
    {'cost__sum': Decimal('514.85')}
1 loop, best of 3: 8.94 ms per loop
```

Better Solution by using "annotation()"

```
In [29]:
%%timeit -n1
cars=Car.objects.all().annotate(Sum('rent__cost'))
for i in cars:
    print "%s\n    %s"%( i, i.rent__cost__sum )
Out [29]:
id: 1, Mitsubishi L200
    1529.50
id: 2, Mini Cooper
    1525.00
id: 3, TVR Tuscan
    2240.00
id: 4, BMW Z3
    1119.95
id: 5, Toyota Celica
    480.00
id: 6, Audi TT
    699.95
id: 7, Mercedes E320
    514.85
id: 1, Mitsubishi L200
    1529.50
id: 2, Mini Cooper
    1525.00
id: 3, TVR Tuscan
    2240.00
id: 4, BMW Z3
    1119.95
id: 5, Toyota Celica
    480.00
id: 6, Audi TT
    699.95
id: 7, Mercedes E320
    514.85
id: 1, Mitsubishi L200
    1529.50
id: 2, Mini Cooper
    1525.00
id: 3, TVR Tuscan
```

```
    2240.00
id: 4, BMW Z3
    1119.95
id: 5, Toyota Celica
    480.00
id: 6, Audi TT
    699.95
id: 7, Mercedes E320
    514.85
1 loop, best of 3: 6.45 ms per loop
```

```
In [30]:
print Car.objects.all().annotate(Sum('rent__cost')).query
Out [30]:
SELECT "myapp_car"."id", "myapp_car"."maker", "myapp_car"."price",
"myapp_car"."model", "myapp_car"."year", CAST(SUM("myapp_rent"."cost") AS
NUMERIC) AS "rent__cost__sum" FROM "myapp_car" LEFT OUTER JOIN "myapp_rent" ON
("myapp_car"."id" = "myapp_rent"."car_id") GROUP BY "myapp_car"."id",
"myapp_car"."maker", "myapp_car"."price", "myapp_car"."model",
"myapp_car"."year"
```

Week 13 Complex Query

จากตัวอย่างโค๊ดใน repository นี้ https://github.com/wasit7/cs459_django2018/tree/master/exam จงตอบคำถามเหล่านี้

Questions

- Q1 Who did spend most money for renting?
- Q2 Which room does make the most amount of income?
- Q3 How many time Jack Jones rent the room?
- Q4 What are the roomss rented by Claire Taylor?
- Q5 what is the total income of ALL rooms in June? Between 1st June 2018(inclusive) and 30th June 2018(inclusive), (use return date).
- Q6 What are the total incomes of EACH room in June? Between 1st June 2018(inclusive) and 30th June 2018(inclusive), (use return date)
- Q7 Find members that have total rent time exactly 8 hours in June?
- Q8 Find members that have total rent time greater than or equal than 9 hours in June?
- Q9 Find the total rent duration of EACH room in June?
- Q10 Find the total income of EACH room in June?

คำถาม

- Q1 ใครจ่ายเงินค่าเช่ามากที่สุด
- Q2 ห้องเช่าใดทำรายได้มากที่สุด
- Q3 Jack Jones เช่าห้องไปกี่ครั้ง
- Q4 Claire Taylor เช่าห้องไปกี่ครั้ง
- Q5 ในเดือนมิถุนายนมีรายรับรวมเท่าใด
- Q6 ในเดือนมิถุนายนแต่ละห้องมีรายรับรวมเท่าใด
- Q7 หาชื่อสมาชิกที่เช่าห้อง 8 ชั่วโมงในเดือนมิถุนายน
- Q8 หาชื่อสมาชิกที่เช่าห้องมากกว่าหรือเท่ากับ 9 ชั่วโมงในเดือนมิถุนายน
- Q9 หาระยะเวลาเช่ารวมของแต่ละห้องในเดือนมิถุนายน
- Q10 หารายได้รวมของแต่ละห้องในเดือนมิถุนายน

```
from datetime import timedelta, datetime
from django.db.models import Avg, Count, Min, Sum, F, ExpressionWrapper, fields
from myapp.models import Member, Room, Rent
```

```
# Q1 ใครจ่ายเงินค่าเช่ามากที่สุด
Member.objects.values('name').annotate(total_sum=Sum('rent__cost')).order_by('-total_sum').first()
Out [1]: {'name': 'Oscar Smith', 'total_sum': Decimal('15116.00')}
```

```
# Q2 ห้องเช่าใดทำรายได้มากที่สุด
Room.objects.values('id').annotate(total_sum=Sum('rent__cost')).order_by('-total_sum').first()
Out [2]: {'id': 10, 'total_sum': Decimal('269645.00')}
```

```
# Q3 Jack Jones เช่าห้องไปกี่ครั้ง
Member.objects.get(name="Jack Jones").rent_set.count()
Out [3]: 35
```

```
# Q4 Claire Taylor เช่าห้องไปกี่ครั้ง
Member.objects.get(name="Claire Taylor").rent_set.count()
Out [4]: 35
```

```
# Q5 ในเดือนมิถุนายนมีรายรับรวมเท่าใด
Rent.objects.filter(stop__month=6).aggregate(total_sum=Sum('cost'))
Out [5]: {'total_sum': Decimal('196236.00')}
```

```
# Q6 ในเดือนมิถุนายนแต่ละห้องมีรายรับรวมเท่าใด
Rent.objects.filter(stop__month=6).values('room_id').annotate(total_sum=Sum('cost'))
Out [6]: <QuerySet [{'room_id': 6, 'total_sum': Decimal('30135.00')},
{'room_id': 7, 'total_sum': Decimal('30400.00')}, {'room_id': 8, 'total_sum':
Decimal('48450.00')}, {'room_id': 9, 'total_sum': Decimal('37700.00')},
{'room_id': 10, 'total_sum': Decimal('49551.00')}]>
```

```
# Q7 หาชื่อสมาชิกที่เช่าห้อง 8 ชั่วโมงในเดือนมิถุนายน
Rent.objects.exclude(stop__isnull=True)\
    .filter(stop__month=6)\
    .values('member__name', )\
    .annotate(total_rent=Sum(ExpressionWrapper(F('stop')-F('start'),
output_field=fields.DurationField())))\
    .filter(total_rent=timedelta(seconds=60*60*8))\
    .values('member__name')
Out [7]: <QuerySet [{'member__name': 'James Jones'}, {'member__name': 'Oscar
Johnson'}, {'member__name': 'William Thomas'}]>
```

```
# Q8 หาชื่อสมาชิกที่เช่าห้องมากกว่าหรือเท่ากับ 9 ชั่วโมงในเดือนมิถุนายน
Rent.objects.exclude(stop__isnull=True)\
    .filter(stop__month=6)\
    .values('member__name', )\
    .annotate(total_rent=Sum(ExpressionWrapper(F('stop')-F('start'),
output_field=fields.DurationField())))\
    .filter(total_rent__gte=timedelta(seconds=60*60*9))\
    .values('member__name')
Out [8]: <QuerySet [{'member__name': 'Charlie Brown'}, {'member__name': 'Charlie
Evans'}, {'member__name': 'Charlie Johnson'}, {'member__name': 'Charlie
Thomas'}, {'member__name': 'Charlie Wilson'}, {'member__name': 'Claire Brown'},
{'member__name': 'Claire Davies'}, {'member__name': 'Claire Evans'},
{'member__name': 'Claire Johnson'}, {'member__name': 'Claire Jones'},
{'member__name': 'Claire Taylor'}, {'member__name': 'Claire Thomas'},
{'member__name': 'Claire Williams'}, {'member__name': 'Claire Wilson'},
{'member__name': 'George Davies'}, {'member__name': 'George Evans'},
{'member__name': 'George Jones'}, {'member__name': 'George Smith'},
{'member__name': 'George Taylor'}, {'member__name': 'George Williams'},
'...(remaining elements truncated)...']>
```

```
# Q9 หาระยะเวลาเช่ารวมของแต่ละห้องในเดือนมิถุนายน
Rent.objects.exclude(stop__isnull=True)\
    .filter(stop__month=6)\
    .values('room__id')\
    .annotate(total_rent=Sum(ExpressionWrapper(F('stop')-F('start'),
output_field=fields.DurationField())))
Out [9]: <QuerySet [{'room__id': 6, 'total_rent': datetime.timedelta(11,
82800)}, {'room__id': 7, 'total_rent': datetime.timedelta(12, 57600)},
{'room__id': 8, 'total_rent': datetime.timedelta(11, 75600)}, {'room__id': 9,
'total_rent': datetime.timedelta(12, 7200)}, {'room__id': 10, 'total_rent':
datetime.timedelta(10, 32400)}]>
```

```
# Q10 หารายได้รวมของแต่ละห้องในเดือนมิถุนายน
Rent.objects.exclude(stop__isnull=True)\
    .filter(stop__month=6)\
    .values('room__id')\
    .annotate(total_cost=Sum('cost'))
Out [10]: <QuerySet [{'room__id': 6, 'total_cost': Decimal('30135.00')},
{'room__id': 7, 'total_cost': Decimal('30400.00')}, {'room__id': 8,
'total_cost': Decimal('48450.00')}, {'room__id': 9, 'total_cost':
Decimal('37700.00')}, {'room__id': 10, 'total_cost': Decimal('49551.00')}]>
```

Week 14 Authentication

views.py basic authentication

```
from django.shortcuts import render
from django.http import HttpResponse
from django.contrib.auth import authenticate, login, logout, update_session_auth_hash
from django.shortcuts import redirect
from django.contrib.auth.decorators import login_required
from django.conf import settings
import sys
from django.contrib.auth.forms import PasswordChangeForm
def signin(request):
    if request.method == 'POST' and 'username' in request.POST:
        username = request.POST['username']
        password = request.POST['password']
        user = authenticate(username=username, password=password)
            if user is not None:
                if user.is_active:
                    if 'remember' in request.POST:
                        if request.POST['remember']=='1':
                            request.session.set_expiry(604800) #remember keep session for a week
                        else:
                            request.session.set_expiry(14400) #not remember keep session for 4hrs
                        login(request, user)
                        if 'username' in request.session:
                            pass
                        request.session['username'] = user.username
                        return redirect('http://localhost:8000/admin/')
                else:
                    msg="Disabled account"
             else:
                 msg="Invalid username or password"
                 return render(request,'login.html',{'msg': msg})
    return render(request,'login.html',{'msg': ""})
def signout(request):
    print("signout")
    if 'username' in request.session:
        del request.session['username']
    logout(request)
    return redirect('wl_auth:signin')
@login_required(login_url='wl_auth:signin')
def change_password(request):
    form = PasswordChangeForm(user=request.user)
    if request.method == 'POST':
        form = PasswordChangeForm(user=request.user, data=request.POST)
        if form.is_valid():
            form.save()
            update_session_auth_hash(request, form.user)
            return redirect('main:home')
    return render(request, 'change_password.html', {
        'form': form,
    })
```

ตัวอย่างการทำขั้นตอนการยี่ยยันตัวบุคคลอย่างง่าย

Week 15 Deployment with Docker

ในขั้นสุดท้ายโปรเจคที่พัฒนามาต้องนำขึ้นไปรันบนเครื่อง server ซึ่งในปัจจุบันเทคโนโลยี Docker เป็นที่นิยมอย่างมาเพราะทำให้สามารถจัดการติดตั้งได้ภายในชุดคำสั่งไม่กี่บรรทัด โดยรายละเอียดได้บรรยายไว้ในวิดีโอด้านล่านนี้

Please watch a video tutorial before the class

https://www.youtube.com/watch?v=gQe2txpV4eA

โดยรายละเอียดของการ deplyment ประกอบด้วยฐานข้อมูลที่กำหนด Docker image ชื่อ postgres_testcompose และอยู่ที่ ./postgres

และมี web server ที่กำหนด Docker image เป็น web1_testcompose โดยใช้คำสั่ง sh /code/run.sh ในการเริ่มการทำงาน และทำจากกหนดพอร์ตจาก host ไป container จาก 8000 ไป 8000 และทำการเชื่อมตำแหน่ง volume จาก ./myproject ของ host ไปที่ ./code ของ container

docker-compose.yml

```
version: '3.5'
services:
  db:
    container_name: postgres_testcompose
    build: ./postgres
    restart: always

  web1:
    container_name: web1_testcompose
    build: ./myproject
    command: sh /code/run.sh
    ports:
      - 8000:8000
    volumes:
      - ./myproject:/code
    depends_on:
      - db
```

/postgres/Dockerfile

```
FROM postgres
```

/myproject/Dockerfile

```
FROM python:3
ENV PYTHONUNBUFFERED 1
ADD . /code
WORKDIR /code
RUN pip3 install -r requirements.txt
```

Docker file ของ webserver ทำงานโดยการคัดลอก Docker Image จาก python:3 โดยตั้งค่า system environment variable เป็น PYTHONUNBUFFERED 1 และทำการสร้าง /code ภายใน container ก่อนที่จะย้าย working directory ไปที่ /code และสั่ง pip3 install -r requirements.tx ภายใน container

nginx

จากตัวอย่างใน /docker_compose ยังไม่มีประสิทธิภาพในการตอบสนอง เราจำเป็นต้องใช้ nginx เพื่อทำหน้าที่เป็นตัวจัดการ load balancing โดยสามารถเพื่อ web server ได้โดยแก้ไขไฟล์ yml

```
version: '3.5'

services:
  db:
    container_name: postgres
    build: ./postgres
    restart: always

  web1:
    container_name: web1
    build: ./myproject
    command: sh /code/run.sh
    volumes:
      - ./myproject:/code
    depends_on:
      - db

  nginx:
      container_name: nginx
      image: nginx:latest
      ports:
        - "80:80"
      volumes:
        - ./nginx/:/etc/nginx/conf.d/
        - ./myproject:/code
      depends_on:
        - web1
```

File structure

โครงสร้างไฟล์ของตัวอย่างในสัปดาห์นี้

- docker_file/ เป็นตัวอย่างการเขียน Dockerfile เพื่อจะสร้าง process or container ตามที่กำหนด
- docker_compose/ เป็นการกำหนดการทำงานของหลาย container ให้ทำงานร่วมกัน
- deployment/ เป็นตัวอย่างในการนำเว็บแอปพลิเคชันที่พัฒนาด้วย Django ให้พร้อมทำงานในเครื่อง server

```
wasitVision/week_14: tree -d
.
├── deployment
│   ├── myproject
│   │   ├── myapp
│   │   │   ├── migrations
│   │   │   │   └── __pycache__
│   │   │   └── __pycache__
│   │   ├── myproject
│   │   │   └── __pycache__
│   │   └── www
│   │       ├── media
│   │       │   └── cars
│   │       └── static
│   │           └── admin
│   │               ├── css
│   │               │   └── vendor
│   │               │       └── select2
│   │               ├── fonts
│   │               ├── img
│   │               │   └── gis
│   │               └── js
│   │                   ├── admin
│   │                   └── vendor
│   │                       ├── jquery
│   │                       ├── select2
│   │                       │   └── i18n
│   │                       └── xregexp
│   ├── nginx
│   └── postgres
├── docker_compose
│   ├── myproject
│   │   └── myproject
│   │       └── __pycache__
│   └── postgres
└── docker_file

34 directories
```