COMP3011 Web Services and Web Data Coursework 2

Django Project Implementation Report

Project Initialisation

This project was edited using Visual Studio Code. Upon opening a new folder with VS Code I initialised the project by using pip to install the ‘django’, ‘requests’ and ‘pipreqs’ modules. This was done through the terminal using the following commands using Python’s package manager. ‘Pipreqs’ creates the requirements.txt folder.

```
pip install django
pip install requests
pip install pipreqs
pipreqs
```

Next, I created the Django project by running this command:

```
django-admin startproject bank
```

This creates a new directory called ‘bank’ in the current directory. Essential Python files provided by Django are installed. The setup is shown below.

```
myproject/
    manage.py
    bank/
        __init__.py
        settings.py
        urls.py
        asgi.py
        wsgi.py
```

The web application will be empty however, to verify it is running correctly the following command and output are expected.

```
python manage.py runserver
```

```
Starting development server at http://127.0.0.1:8000/
```

Model Definition

Firstly, the models for the database are created. This is achieved by creating a file called ‘models.py’ in the ‘bank’ directory i.e. in the same directory as ‘settings.py’. To utilise Django’s database handler the following line of code is used in ‘models.py’.

```
from django.db import models
```


```
from django.db import models
```


The structure of these databases was predefined in Coursework 1 however there have been minor adjustments since then. I am acting as a bank web application, an example of one of the tables in the database is as follows.

```
class Transaction(models.Model):

    transaction_id = models.AutoField(primary_key=True)

    account_id = models.ForeignKey(Account,
related_name='transactions', on_delete=models.CASCADE)

    booking_id = models.CharField(max_length=255)

    transaction_date = models.DateTimeField(auto_now_add=True)

    transaction_amount = models.FloatField()

    transaction_currency = models.ForeignKey('Currency',
on_delete=models.CASCADE)
```

The name of this table is ‘Transaction’. The attributes in the table include: ‘transaction_id', ‘account_id’, ‘booking_id’, ‘transaction_date’, ‘transaction_amount’ and ‘transaction_currency’. They each have different fields, defining what type their instances intend to hold. For example, ‘transaction_amount’ expects a float. ‘Transaction_id’ is the primary key meaning this value is what’s used to uniquely identify each row in the table. Foreign keys are used to link one table to another, for example, ‘account_id’ references the ‘Account’ table.

From the command line, two commands are needed to migrate the models into the project utilising ‘manage.py’.

```
py manage.py makemigrations
```

```
python manage.py migrate
```

Admin Interface

In terms of providing an admin interface, the following commands are put into the local terminal. The information for gaining access to the admin page is in the ‘readme.txt’ file.

```
python manage.py shell
```

```
In [1]: from django.contrib.auth.models import User

In [2]: User.objects.create_superuser('ammar',
'ammar@example.com', 'password')
```

URL Configuration

Utilising the already existing ‘urls.py’ Python file allows a requested URL to find the correct view. Functions from the views page must be imported, using:

```
from . import views
```

Inside the ‘urlpatterns’ array, a list of pages is defined. The order is <appended url>, <view name>, <name>. This can be seen below.

```
urlpatterns = [

    path('admin', admin.site.urls),

    path('bank/pay', views.pay, name='pay'),

    path('bank/refund', views.refund, name='refund'),

    path('bank/exchange/<str:from_currency>/<str:amount>',
views.exchange, name='currency exchange'),

]
```

‘<str:from_currency>/<str:amount>/’ can be seen in the final path. This is because this path is expecting a GET request so data is not sent via POST to this class. A currency code, for example, EUR or USD is expected along with an amount of money.

Note: the path django function must also be loaded.

```
from django.urls import path
```

To access these URLs there is a base URL followed by the directories shown above, for example, accessing the admin path is as follows:
http://127.0.0.1:8000/admin
https://sc20wrpf.pythonanywhere.com/admin/

Views

In a similar fashion to ‘urls.py’, create a Python file called ‘views.py’. This file contains the functionality of the website i.e. the backend. Each view (class/function) within ‘views.py’ accepts a web request and responds with a web response. The following imports are necessary for this project.

```
from django.http import JsonResponse
```

```
from bank.models import Account, Currency, Transaction

import json

import requests
```

Django provides a package allowing responses from web requests to be in JSON through the use of ‘JsonResponse’ meaning information can be sent from different web applications. The models as mentioned before are imported from the ‘models.py’ file so they can be modified from this ‘views.py’. Requests are also another important component of this project as they allow information in the form of JSON to be posted to other web applications and provide a status code showing whether a request was a success or not.

Populating the database locally allows for testing the modification of the database as it would be when connected to the other web applications. I created a simple function that adds a variety of instances to each of the three tables in the database, a snippet can be seen below.

```
def populateDatabase():

    Currency.objects.create(code='GBP', symbol='£',
exchange_rate=1.00)

      gbp = Currency.objects.get(code='GBP')

    Account.objects.create(company_name='KevAir', balance=10000.00,
currency_id=gbp)

    account_a = Account.objects.get(account_id=1)

    Transaction.objects.create(account_id=account_a,
booking_id='001', transaction_date=timezone.now(),
transaction_amount=format(random.uniform(100, 1000), '.2f'),
transaction_currency=gbp)
```

The logic behind the web application is contained within three important classes. These are paying the bank, refunds from the bank and a currency converter. The pay and refund classes expect a POST whereas the currency converter expects a GET. In the case of POST, data is received in the form of JSON to the classes which are used to determine what airline bank account is responsible for making or receiving a payment. The currency convertor uses information from the URL to compose JSON data to return back to the caller.

Deployment

PythonAnywhere allows users to upload Python projects for use online. The implementation of this project followed along with PythonAnywhere’s documentation for the deployment of a Django project found here:
https://help.pythonanywhere.com/pages/DeployExistingDjangoProject/

In summary:

1. Create PythonAnywhere account
2. Open Bash
3. Git clone project

```
git clone <link to git project>
```

4. Open virtual environment

```
mkvirtualenv --python=/usr/bin/python3.10 bank-virtualenv
```

5. Move to directory with requirements.txt and install dependencies

```
pip install -r requirements.txt
```

6. From the web tab select: manual configuration > python3.10
7. Input virtual environment name i.e. bank-virtualenv
8. Edit ‘settings.py’

```
ALLOWED_HOSTS = ['sc20wrpf.pythonanywhere.com']
```

9. Edit WSGI to contain only Django section with the correct information

```
import os
import sys

# assuming your django settings file is at '/home/willfleming/mysite/mysite/settings.py'
# and your manage.py is is at '/home/sc20wrpf/mysite/manage.py'
path = '/home/sc20wrpf/cwk2/bank'
if path not in sys.path:
    sys.path.append(path)

os.environ['DJANGO_SETTINGS_MODULE'] = 'bank.settings'

# then:
from django.core.wsgi import get_wsgi_application
application = get_wsgi_application()
```

10. Migrate database as before

```
./manage.py migrate
```

This project can be found at: https://sc20wrpf.pythonanywhere.com/