Universidad Autónoma de Madrid

Departamento de Ingeniería Informática

Computer Systems Project
Assignment 1

The teaching staff of the subject

Changelog

Version[1]	**Date**	**Author**	**Description**
1.0	03.01.2023	JAMI	First version.
1.1	09.01.2023	AB	Translation to English.
1.2	26.01.2023	RM	Revised before upload to Moodle.
1.3	31.01.2023	JAMI	Changing Application name (rango vs catalog) in coverage.
1.4	03.02.2023	AB	Fixed inconsistency with the name of the tests folder.
1.5	11.02.2023	JAMI	Adding new information about the utilization of neon.tech as DBMS.

[1]Version control is made using 2 numbers $X.Y$. Changes in Y denote clarifications, more detailed descriptions of some aspect, or translations. Changes in X denote deeper modifications that either change the provided material or the content of the assignment.

Contents

1 Introduction

The goals of this assignment are, on the one hand, introduce the web application programming through the *Django* framework and, on the other, deploy the web application in a production environment through the *Render* platform. To achieve these goals, the student will need to closely follow, step by step, the tutorial available at: `https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django`.

Besides, the specifications provided in this document must be followed. In general, the following aspects will be considered:

- The code will be graded using the operating system *Ubuntu 22.04 LTS*, which is the one available in the EPS labs.
- Python version to use will be 3.9.
- *Django* version to use will be 3.2.
- It is recommended to use, as development environment, the IDE Visual Studio Code.
- A virtual environment (virtualenv) will be used to work and create the specific dependencies for the development of this assignment.
- `git` will be used, by creating a new private repository for this assignment that will be shared only between the pair of students who work together in the same team. The (hidden) folder `.git` must be mandatory included as part of the submission of this assignment. Additionally, a `.gitignore` file must be created to avoid uploading irrelevant files to the repository (temporary files, etc.). The team of students must discuss with their lab teacher the usefulness of sharing the `git` repository created for this assignment. The use of GitHub is recommended.
- The Python code implemented must satisfy the *PEP8* style guide, which must be verified through the `flake8` utility (which might be installed through the command `pip3 install flake8`).
- Even though the tutorial itself will indicate when to install each package with `pip3`, in *Moodle* you will find the final version of the `requirements.txt` file, where all the necessary dependencies to develop this assignment can be found.

- In *Moodle* you may find a set of test that the assignment shall pass obligatory. The general form to proceed will be to pass the test corresponding to each week. This can be done at the end of the required implementation, but the test can also be used as an implementation guide of the assignment, since they will be showing certain specific requirements that have to be satisfied. Any error that appears in the test will require to fix the code. Therefore, the provided test in *Moodle* cannot be modified, since they will be used to evaluate the assignment.

- Do not modify the names of the objects to create through the tutorial (classes, fields, databases, etc.), nor those provided in this document.

- The general way to work with this assignment will be to proceed working on the proposed tasks for each week, which implies to study and implement the required chapters from the tutorial while performing, mandatorily, the proposed exercises at the end of each part (*challenges*). At the end of each chapter, some links are provided too for the student to use to further extend or consolidate their knowledge.

- This assignment aims to encourage to study, step by step, the basic and needed knowledge for the development and deployment of web applications. It is convenient that, even you may work in teams, each student internalize in an individual way the necessary knowledge to advance and get over the following assignments and mandatory exams.

2 Description of the work to be done during the first week

As an introduction, you must read and put in practice the following chapters of the tutorial previously mentioned, especially if you also want a personalized installation of the environment in your own PC. In any case, the reference operating system would be the one mentioned previously (*Ubuntu 22.04 LTS*). The introductory chapters to address are the following (in this case, it is not necessary to submit what is implemented in these chapters):

- *Django introduction.*

- *Setting up a Django development environment.*

To start working with the library application (*locallibrary*), you must create a new *Git* repository for this assignment. Besides, you will create and use (activate) a virtual environment for the required Python version (3.9):

```
virtualenv p1_env --python=python3.9
source p1_env/bin/activate
```

With the virtual environment active, you must install the required version of Django:

```
pip3 install django~=3.2
```

Once you have a strong knowledge on what Django is, you must read and implement, step by step, what is described in the following chapters of the tutorial to start implementing the web application of the library (project `locallibrary`, application `catalog`) that will be submitted according to what is indicated in this assignment planning, by performing the exercises presented at the end of each chapter:

- *Django Tutorial: The Local Library website.*
- *Django Tutorial Part 2: Creating a skeleton website.*
- *Django Tutorial Part 3: Using models.*

Recall the created code must follow the *PEP8* directives. That is, if the code is checked with the `flake8` analyzer, the output must not contain errors nor warnings. You may ignore the warnings related to the provided code, or with code lines produced automatically by Django.

Once the models indicated in Part 3 are created, you must populate the database using the file `populate_catalog.py` that could be downloaded from *Moodle*. Place the file at the root of the project, and it must be called as follows:

```
python3 populate_catalog.py
```

2.1 Test

In the *Moodle* page of the subject you may find the files `test_xxxx_week.py`, which conform the test collection to be passed during the implementation of the assignment. These files must be located in a folder called `tests` inside the application folder (`catalog`) created for the project (`catalog/tests`).

For this first week, you must pass the corresponding test (`test_first_week.py`). If any error arises, you must modify your code to satisfy the test, never the code of the provided test. The test of this first week will be run in the following way:

```
python3 manage.py test catalog.tests.test_first_week --verbosity 2
```

2.2 Test coverage

The application `coverage` will be used to measure the coverage of the test, that is, the percentage of code that is accessed from the test, which will allow knowing how much code we are really testing. For this, it is convenient to have the version of `coverage` 5.5 or higher installed. When installed inside a virtual environment, it is convenient to make sure the `coverage` application that is being executed is the one from the virtual environment, and not an external one (one that is in the *path*).

To run `coverage`, you should proceed as follows (more information in: `https://coverage.readthedocs.io`):

```
coverage erase
coverage run --omit="*/test*" --source=catalog manage.py test catalog.tests
coverage report -m -i
```

It is important to indicate that `coverage` will run the test that are available in the previously mentioned folder, hence it is convenient to gradually add the test so they do not raise errors because of a lack of (not implemented yet) code. To make an initial check, it is necessary to have in folder `catalog/test` only the test that correspond to the first week.

After the execution, you will obtain a result similar to the one shown in Listing 1. This coverage will increase as long as you advance in the tutorial and test for successive weeks are incorporated. Moreover, the percentage will increase when, later on, the required test in Part 10 from the tutorial had been implemented. Ideally, at the end the coverage should be near 100%.

3 Description of the work to be done during the second week

During this week, you will configure the persistence of data through the database manager system `PostgreSQL`. Besides, you will continue with the proposed tutorial, following step by step what is indicated in every chapter and performing the proposed exercises.

To ensure the persistence of the data from the application, you will work with `PostgreSQL`, instead of `SQLite` which is the default database manager. For this, you must install two packages (`dj-database-url` and `psycopg2-binary`), whose versions

Listing 1: Output of `coverage` command.

```
Name                                  Stmts   Miss  Cover   Missing
-------------------------------------------------------------------
catalog/__init__.py                       0      0   100%
catalog/admin.py                          1      0   100%
catalog/apps.py                           3      3     0%   1-5
catalog/migrations/__init__.py            0      0   100%
catalog/models.py                         1      0   100%
catalog/urls.py                           3      0   100%
catalog/views.py                          8      0   100%
-------------------------------------------------------------------
TOTAL                                    18      5    80%
```

are specified in the `requirements.txt` file. This is also explained in Part 11 of the tutorial (*Django Tutorial Part 11: Deploying Django to production*), although in this case the task will be done during this week.

Once the dependencies are installed, the variable `DATABASES` must be modified in the `settings.py` file in the following way:

```
#The following environment variable, called DATABASE_URL, has to be defined
#at the o.s. level: export DATABASE_URL =
#        'postgres://alumnodb:alumnodb@localhost:5432/psi'

import dj_database_url

db_from_env =
  dj_database_url.config(default='postgres://alumnodb:alumnodb@localhost:5432/psi',
    conn_max_age=500)

DATABASES['default'].update(db_from_env)
```

You must always use `psi` as name of the database associated to the project, and `alumnodb` as the corresponding user name and password.

The databases created using `PostgreSQL` are not deleted when the lab computer is shutdown. Therefore, it is possible there is already an existing database in the computer called `psi` with a structure different from the required one. To avoid conflicts, it is recommendable that, at the beginning of every session, the `psi` database is deleted. To delete the database you may use the command `dropdb -U alumnodb -h localhost psi`, then, you may create it again with `createdb -U alumnodb -h` `localhost psi`. When deleting the database, it is necessary to populate it again using the `populate_catalog.py` script.

Once all these tasks are done, you will continue with the next chapters of the tutorial, going on with the implementation of the library application, and performing the exercises (challenges) proposed at the end of every chapter:

- *Django Tutorial Part 4: Django admin site.*
- *Django Tutorial Part 5: Creating our home page.*
- *Django Tutorial Part 6: Generic list and detail views.*

It is important to note that in the tutorial the development Django server uses port 8000. However, it is possible this port is not available in the lab computers, so you may use the port 8001 instead by running the command:

```
python3 manage.py runserver 8001
```

As discussed in the tutorial, to have access to the admin interface you must create a *superuser*, which will also allow to access the database generated from the models and to manipulate the information it contains. For this purpose the command `python manage.py createsuperuser` will be used. Both the user and password to be used should be `alumnodb`.

3.1 Test

The test related to the work of this second week are located in the file `test_second_weeky.py`. Tests from first and second week must pass. To run the test, you will proceed as follows:

```
python3 manage.py test catalog.tests.test_first_week --verbosity 2
python3 manage.py test catalog.tests.test_second_week --verbosity 2
```

4 Description of the work to be done during the third week

In this week, you will study the last chapters of the tutorial. You will, on top of that, deploy in *Render* the created application with *Django*, and proceed with the final submission of the assignment, complying with everything required in this document.

You must read and implement, step by step, what is described in the following chapters of the tutorial to continue implementing the library web application, while performing the required exercises at the end of each part:

- *Django Tutorial Part 7: Sessions framework.*
- *Django Tutorial Part 8: User authentication and permissions.*
- *Django Tutorial Part 9: Working with forms.*
- *Django Tutorial Part 10: Testing a Django web application.*
- *Django web application security.*
- *Django Tutorial Part 11: Deploying Django to production.*

During the study of chapter 11, you will take into account the general aspects related to the deployment of an application in a real production environment, even though the specific instructions will be provided in Section *4.2. Additional considerations about Render* below.

4.1 Test

Your code should pass the test from the first, second, and third week. As usual, when facing any error, you must modify the implemented code to satisfy the test, and never the provided code in the test:

```
python3 manage.py test catalog.tests.test_first_week --verbosity 2
python3 manage.py test catalog.tests.test_second_week --verbosity 2
python3 manage.py test catalog.tests.test_third_week --verbosity 2
```

Additionally, you must pass all the test to be implemented in Part 10 of the tutorial (*Django Tutorial Part 10: Testing a Django web application*), together with those test implemented based on the proposed *challenge* at the end of this chapter. All the test must be located in the folder `/catalog/test`, and might be executed in the following way:

```
python3 manage.py test catalog.tests --verbosity 2
```

It is possible that, when running the implemented test in chapter 10, the system returns the error mentioned in the tutorial (*Missing staticfiles manifest entry*), which can be fixed according to the provided solutions in the same chapter. By default, the IDs assigned to the objects persisted in the database are created automatically. Usually the object created first will have id=1, the second id=2, etc. Nevertheless, *Django* does not guarantee the values chosen for the IDs and therefore, it is possible to have an error in those test that perform an explicit register selection through `.objects.get(id=1)`. The way to fix this error is to add, when the object is created, a field called `id`, for example: `Author.objects.create(id=1, first_name='Big', last_name='Bob')`.

4.2 Additional considerations about Render

Render is a *plataform as a service* for computing in the cloud (PaaS) that allows to deploy the created application in a production environment.

To work with Render, the first step is to create an individual (free) account, which can be done in the following link: `https://www.render.com`. The step of production deployment must be done once the application is finished and free of errors. It is a time consuming process, thus it is recommended to finish, fix all errors, and pass all the test first in the development environment and prepare finally the application, as a last step, to be deployed in the production environment in Render.

It is important the `requirements.txt` file is in the root of the project. The file `requirements.txt` is the one provided in *Moodle*, but in any case it can be automatically created with the command:

```
pip3 freeze > requirements.txt
```

The content of the resulting file must be similar to what is shown in Listing 2.

Listing 2: File `requirements.txt`.

```
asgiref==3.6.0
coverage==6.4.3
dj-database-url==0.5.0
Django==3.2.1
gunicorn==20.0.4
Pillow==9.4.0
psycopg2-binary==2.9.5
PyJWT==2.6.0
pytz==2022.7
six==1.15.0
sqlparse==0.4.3
```

```
text-unidecode==1.2
urllib3==1.26.9
whitenoise==5.2.0
python-dotenv==0.21.0
Brotli==1.0.9
```

To perform the deployment, you must create a (new) web service (*New Web Service*). It is not necessary to create a PostgreSQL service with Render since, even though it is possible, the created database would have important limitations; the connection with the database manager will be explained later.The deployment can be done automatically by linking the GitHub repository with the Render project, by being properly identified and providing credentials. It is even possible to perform the deployment automatically after every *commit* in the GitHub repository, although it is more recommendable to do it manually (*Manual Deploy - Deploy Latest Commit*).

It is convenient to create or modify the hidden `.gitignore` file, by indicating those files to be ignored in the deployment. Such file must contain, at least, the constraint to not upload compiled files (`*.pyc`).

You may follow the instructions included in the next link to correctly configure the project to be deployed in Render:

`https://testdriven.io/blog/django-render`

Besides, you must take into account the following considerations:

- The web application must be started, for this case, with `gunicorn locallibrary.wsgi`. This parameter must be indicated in the service configuration in Render.
- In the `build.sh` script, which is found in the root of the project, you may add all the necessary commands to populate the database, etc. On the other side, in the `createsu` command that is executed in the script file, you must indicate the user name and password already described (`alumnodb`).
- You must pay special attention to the management of static files. This is, besides, especially convenient towards future assignments. It is recommended to create a `static` folder in the root of the project.
- The PostgreSQL database we have locally will not work in Render, hence we need to use the proper mechanism to be able to access a database remotely and persist the information in it. For this, we will use *neon.tech*, a PostgreSQL service in the cloud that allow us to create and use a database remotely, by providing a URL to access it (to be used through the variable `dj_database_url`

in *Django*, and also in Render. To obtain this service, you need to register in `https://www.`*`neon.tech`*. A free account should be enough. Recall to update the obtained URL to access the database in the previous variable, both in the `settings.py` file, and in *Render* (DATABASE_URL).

However, the *neon.tech* does not allow to create new databases to execute the tests. This way, it is necessary to create a new environment variable to control whether we run tests (using local PostgreSQL) or populate the application data (in *neon.tech*). To do that, it is necessary to modify the file `settings.py`:

```
# To run the tests: export TESTING=1, or to use the app: unset TESTING
# To see the current value just type echo $TESTING

if 'TESTING' in os.environ:
   db_from_env = dj_database_url.config(default=POSTGRESQL_URL', conn_max_age=50
else:
   db_from_env = dj_database_url.config(default=NEON_URL, conn_max_age=500)

DATABASES['default'].update(db_from_env)
```

Where `POSTGRESQL_URL` is the local URL to access PostgreSQL, and `NEON_URL` is the URL provided by *neon.tech* (copied from the *neon.tech* web interface).

Make sure to run all the test and check they are working correctly. You must submit, besides, a text file with the result of running the `coverage` utility on all the implemented test, which should report an approximated coverage of 100%.

5 Material to be submitted at the end of the assignment

1. Include in the root of the project a file called `authors.txt`, with the name of the authors and the number of the team (pair).
2. Calculate the test coverage through the `coverage` utility, by generating a file in text format (`coverage.txt`) that must be included in the root of the project. The result must be a table similar to the one that can be seen in Listing 1.
3. You must clearly indicate, in the material submitted to *Moodle*, the URL where the project is deployed in *Render*. This information must appear correctly configured in the `settings.py` file submitted with the project (variable `ALLOWED_HOSTS`). In *neon.tech*, the database must be populated with the content of the `populate_catalog.py` file, and the admin interface must be accessible through the required user name and password (`alumnodb`).
4. Make sure **ALL** the test, both those from *Moodle* as those developed in the tutorial, are satisfied by the implemented code. It is not acceptable to modify the code of the provided test.
5. Upload to *Moodle*, in a single ZIP file, the project with all the required and necessary files to run the developed application in this assignment. In particular, you must upload to *Moodle* the file obtained after running the command `"zip -r ../assign1.zip ."` or similar, making sure to include all the project files, including the hidden `.git` folder. Do NOT submit the virtual environment used or the `*.pyc` files.

6 Evaluation criteria

The grade of this assignment will be *Pass* or *No Pass*. To pass the assignment it is necessary to satisfy ALL the following criteria:

- The code created for this assignment must: (a) run correctly using Python 3.9 and *Django* 3.2, and (b) satisfy all the checks made by `flake8`, except those relative to the code provided by the teachers or automatically generated by *Django*.

- The application data must be persisted in a PostgreSQL database, according to the given specifications.

- The complete application, developed throughout the tutorial, must be deployed in *Render*, and must be accessible through the provided URL. In *Render*, the database must be populated in *neon.tech*. On the other side, the admin interface must be complete, allowing to visualize and change the data using the created superuser (`alumnodb`).

- The created code in this assignment must be stored in a private repository through `git`. There must be written evidence, therefore, of the work done in `git` through the `.git` folder, which must be mandatorily submitted.

- The necessary code to run the application must be submitted in *Moodle*, and it must be possible to run it locally.

- The submitted code must satisfy ALL the test.

- The result of running the `coverage` utility must be included (see Listing 1) in a file in text format called `coverage.txt`, located in the root of the project.

NOTE: The code used to evaluate this assignment will be the one uploaded to *Moodle*. In no case the existing code in the *git* repository or in *Render* will be used.