June 5, 2022

Chia yi quan

Student Number: 210220223

Advanced web development

report for coursework 1(rest api)

Contents

[Introduction 2](#_Toc107129860)

[Running Of The Projects 2](#_Toc107129861)

[Setting Up Of The Project 2](#_Toc107129862)

[Migration Of The Project 3](#_Toc107129863)

[Running Of The Project 5](#_Toc107129864)

[Running The Test Case 10](#_Toc107129865)

[Code Explanation 11](#_Toc107129866)

[Models 11](#_Toc107129867)

[Data Population Script 13](#_Toc107129868)

[API And Serializers 17](#_Toc107129869)

[Test Case 23](#_Toc107129870)

[Conclusion 28](#_Toc107129871)

# Introduction

The purpose of this report is to document down on the bioscience project. The report will be separated into 2 sections which is running of the projects and code explanation. The first section will cover setting up of the project, running migration and script, running of the project, running of test case and code explanation of the required api endpoint. The second section will explain the code in the scripts, models design, serializer and the api code.

# Running Of The Projects

## Setting Up Of The Project

Inside the folder that was submitted, there should be 2 items which is a requirements.txt and bioscience folder. Firstly, we have to setup the virtual environment before going to the migration. Open a terminal and navigate into the project folder(the folder where the requirements.txt and bioscience folder is in). You can create a new virtual environment with the following command.

**py -m venv env\_name** -> Replace the env\_name with the name you want.

Text

Description automatically generated

This will create a env folder in the directory.



In the terminal, for windows change the directory to the Scripts folder which in my case will be **cd bioscience\_env/Scripts** and type in the following command

**activate.bat**

For unix system, change the directory to the bin folder which in my case will be **cd bioscience\_env/bin** and type in the following command

**source activate**

If done successfully, the env name should appear at the front of the directory.

Text

Description automatically generated

Now we have to change directory back to the project folder which is 2 directory above the current directory we are in, so in the terminal we can execute **cd ..** for 2 time. After changing the working directory to the project folder, we can run the following command to install the dependencies.

**pip install -r ./requirements.txt**

After running the command above you should see this screen

Text

Description automatically generated

After installing the dependencies, we can continue to the next step but do not close the terminal yet.

## Migration Of The Project

In the terminal(with the env activated), change directory into bioscience folder. After changing the directory to the folder, we can run the following command in the terminal to do a migration. For the python command it depend on the system global variable which can be python3 py etc..

**python manage.py migrate**

All the models will be migrated into the default database which is the sqlite.

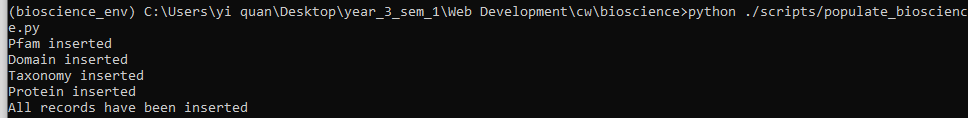
Text

Description automatically generated

Next, we will run the script to import all the csv data into the database, I will explain all the scripts codes at the later part of the report but for now we will run this command to execute the scripts.

**python ./scripts/populate\_bioscience.py**

It will take around 5 minutes if is on SSD and around 10 to 15 minutes for HDD for the scripts to execute finish. However, changing the database to postgres will improve the migration speed. If all data have been inserted in successfully, the screen will appear as the following screenshot.



## Running Of The Project

To start the project, simply just run the command below.

**python manage.py runserver**

Once the server started, the list of api is as listed:

**[POST]** <http://localhost:8000/api/protein/>

The structure to submit a new protein must be in this format

{

"protein\_id": "",

"sequence": "",

"taxonomy": {

"taxa\_id": null,

"clade": "",

"genus": "",

"species": ""

},

"length": null,

"domains": []

}

**Example:**

{

"protein\_id": "A0A016S8J7-duplicate",

"sequence": "MVIGVGFLLVLFSSSVLGILNAGVQLRIEELFDTPGHTNNWAVLVCTSRFWFNYRHVSNVLALYHTVKRLGIPDSNIILMLAEDVPCNPRNPRPEAAVLSA",

"taxonomy": {

"taxa\_id": 53326,

"clade": "E",

"genus": "Ancylostoma",

"species": "ceylanicum"

},

"length": 101,

"domains": [

{

"pfam\_id": {

"domain\_id": "PF01650",

"domain\_description": "PeptidaseC13family"

},

"description": "Peptidase C13 legumain",

"start": 40,

"stop": 94

},

{

"pfam\_id": {

"domain\_id": "PF02931",

"domain\_description": "Neurotransmitter-gatedion-channelligandbindingdomain"

},

"description": "Neurotransmitter-gated ion-channel ligand-binding domain",

"start": 23,

"stop": 39

}

]

}

The result of the post request will be like the following screenshot.

Graphical user interface, text, application

Description automatically generated

**[GET]** [http://localhost:8000/api/protein/[PROTEIN\_ID](http://localhost:8000/api/protein/%5bPROTEIN_ID)]

**Example:**

<http://localhost:8000/api/protein/A0A016S8J7-duplicate/>

The result of the get request will be like the following screenshot.

Graphical user interface, text, application, email

Description automatically generated

**[GET]** [http://localhost:8000/api/pfam/[PFAM\_ID](http://localhost:8000/api/pfam/%5bPFAM_ID)]

**Example:**

<http://localhost:8000/api/pfam/PF00360/>

The result of the get request will be like the following screenshot.

Graphical user interface, text, application, email

Description automatically generated

**[GET]** [http://localhost:8000/api/proteins/[TAXA\_ID](http://localhost:8000/api/proteins/%5bTAXA_ID)]

**Example:**

<http://localhost:8000/api/proteins/55661/>

The result of the get request will be like the following screenshot.

Graphical user interface, application

Description automatically generated

**[GET]** [http://localhost:8000/api/pfams/[TAXA\_ID](http://localhost:8000/api/pfams/%5bTAXA_ID)]

**Example:**

<http://127.0.0.1:8000/api/pfams/55661/>

The result of the get request will be like the following screenshot.

Text

Description automatically generated with low confidence

**[GET]** [http://localhost:8000/api/coverage/[PROTEIN\_ID](http://localhost:8000/api/coverage/%5bPROTEIN_ID)]

**Example:**

<http://localhost:8000/api/coverage/A0A016S8J7/>

The result of the get request will be like the following screenshot.

Table

Description automatically generated

## Running The Test Case

To run the test case, simply run the command.

**python manage.py test**

There are a total of 31 test case which will be like the following screenshot

Text

Description automatically generated

# Code Explanation

## Models

Text

Description automatically generated

The taxonomy model consisted of taxa\_id(integer field), clade(char field with max length 256), genus(char field with max length 256) and species(char field with max length 256). The taxa\_id in this case is a “unique” key as there should not be 2 entry with same taxa\_id with different properties. However, with the create protein api the unique constraint is not added in here because there can be multiple protein having the same taxonomy and the data won’t be able to pass the validation check.

Text

Description automatically generated

The Pfam model consisted of domain\_id(char field with max length 256) and domain\_description(char field with max length 256). The domain\_id in this case is a “unique” key as there should not be 2 entry with same domain\_id with different domain description. However, with the create protein api the unique constraint is not added in here because there can be multiple domain having the same Pfam and the data won’t be able to pass the validation check.

Text

Description automatically generated

The Domain model consisted of pfam\_id(foreign key referencing to Pfam model), description(char field with max length 256), start(integer field), stop(integer field).

Text

Description automatically generated

The Protein model consisted of protein\_id(char field which is unique and have a max length 256), sequence(char field with max length 40000 and is nullable), taxonomy(foreign key referencing to taxonomy model), length(integer field) and domains(many to many field with Domain and is linked through ProteinDomainLink model with protein and domain field). The reason why protein\_id is unique because we only want to have 1 unique record from the dataset as the duplicated data in the csv have different taxanomy and domains. The reason for ManyToManyField on domains is because a protein can have multiple domains.

Text

Description automatically generated

The ProteinDomainLink model consisted of protein(foreign key referencing Protein model) and domain(foreign key referencing Domain model).

## Data Population Script

Text

Description automatically generated

Text

Description automatically generated

This portion of the code is to iterate through the assignment\_data\_sequence.csv and store each row of sequence into the protein\_sequence variable with protein\_id as the key and sequence as the value

Text

Description automatically generated

This portion of code iterate the assignment\_data\_set file. For each row, the organism\_name will store the Genus and Species by splitting them either by “ x “ or “. “ or “ “. The reason for different kind of split is because there are different data with different formatting, so it is inaccurate to split everything by empty spaces. Each row is then stored in 3 different variables which is taxnomy, domain and protein. The taxonomy variable will store the taxa\_id(second column in csv) as the key and add a set of unique set clade(third column in csv), genus and species. Reasoning for using a set is because there is a lot of duplicated data in the csv which can be normalized. The domain variable will store the domain\_id as the key and a list of tuple of description(fifth column in csv), domain\_id(sixth column in csv), start(seventh column in csv), stop(eighth column in csv). There is a lot of duplicated description and domain\_id in the data set however I did not normalize it because normalizing it the api result will be different from the rest\_specification file. Lastly the protein variable store a set of tuple of protein\_id(first column in csv), taxa\_id(second column in csv) and length(last column in csv). The reason of using a set so that duplicated data will be removed.

Text

Description automatically generated

This portion of code iterate the pfam\_descriptions file. For each row, the pfam variable will store the domain\_id(first column in csv) and the domain\_description(second column in csv) as the value.

Text

Description automatically generated

This portion of code is to delete all the entry in the table everytime the script runs.

Graphical user interface, text

Description automatically generated

Text

Description automatically generated

This portion of code iterate each key in the pfam variable that was created earlier on, once we have the domain\_id, it will create the Pfam object and save it to the database. After the insertion is successful, the pfam\_rows variable will store the domain\_id as the key and the row inserted into the database as the value. The pfam\_rows data will be used as foreign key in domain insertion.

Text

Description automatically generated

This portion of the code is to iterate each key in the domain variable and iterate each tuple in the list. Each domain will get the id of the pfam by using the domain\_id and access the entry in pfam\_rows variable. After the data is stored in the database, the row will be stored in domain\_rows variable which will be used in protein insertion.

Text

Description automatically generated

This portion of the code is to iterate each key in taxonomy and iterate each set. Once the data is stored in the database, the row will be stored in taxonomy\_rows variable which will be used in protein insertion.

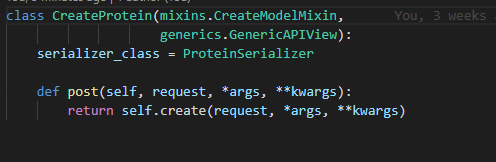
Text

Description automatically generated

This portion of code will iterate the data in the protein variable. Because a protein can have no sequence so there is a check to see if the protein\_id is inside the protein\_sequence if is not then None will be assigned to the sequence variable. Once the protein object is created, we must iterate through the domain\_rows with the protein\_id key as a protein can have multiple domains, after adding all the domains to the protein object, we save it to database and the populating of the csv have finished.

## API And Serializers

**[POST] api/protein/**

****

This is the code that can be found in the api.py file. The CreateProtein class uses the ProteinSerializer to create a new protein.

**Text

Description automatically generated**

This are the 3 serializers which can be found in the serializers.py file that is used in the ProteinSerializer. The TaxonomySerializer uses the Taxonomy model, PfamSerializer uses the Pfam model and DomainSerializer uses the Domain model.

**Text

Description automatically generated**

This is the ProteinSerializer which can be found in the serializers.py file. Firstly, in the create function, the taxonomy data and domains data are being extracted from the post request body. After that we check if the existing taxonomy id is in the database if it is we will use the record that is in the database else create a new taxonomy in the database. The same check goes for pfam if there is existing pfam it will use that pfam else it will create a new pfam record. After the pfam check, a new domain will be created with the pfam as the foreign key and it is appended to the domain list. After that a new protein object is created and the domain list is being looped to add each domain into the protein.

**api/protein/<str:protein\_id>/**

Text

Description automatically generated

This is the code that can be found in the api.py file. The ProteinDetail class uses the ProteinSerializer to get the detail of a protein. The lookup\_field is to search the result based on the protein\_id field in the database and the url param.

**Text

Description automatically generated**

This are the 3 serializers which can be found in the serializers.py file that is used in the ProteinSerializer. The TaxonomySerializer uses the Taxonomy model, PfamSerializer uses the Pfam model and DomainSerializer uses the Domain model.

Text

Description automatically generated

This is the ProteinSerializer that can be found in serializers.py file. It calls the TaxonomySerializer and DomainSerializer with the many=True parameter as it is expected that domains will return a list of result. It returns the protein\_id, sequence, taxonomy, length and domains field from the Protein model.

**api/pfam/<str:** **pfam\_id >/**

Text

Description automatically generated

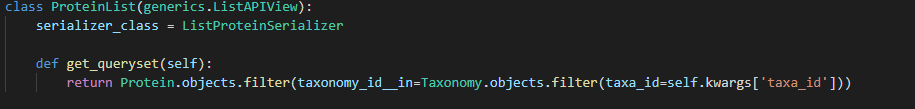
This is the code that can be found in the api.py file. The PFamDetail class uses the PfamSerializer to find the pfam details with pfam\_id. The lookup\_field is the field to search in the database and the lookup\_url\_kwarg is the field to search in the url param.

Text

Description automatically generated

This is PfamSerializer which can be found in the serializers.py file. It returns the domain\_id and domain\_description field from the Pfam model.

**api/proteins/<int:taxa\_id>/**

****

This is the code that can be found in the api.py file. The ProteinList class uses the ListProteinSerializer to list protein\_id with taxa\_id. Because generics.ListAPIView is unlike generics.GenericAPIView as it won’t take in lookup\_field so there is a need create a get\_queryset function to filter the taxa\_id.

Text

Description automatically generated

This is ListProteinSerializer which can be found in the serializers.py file. It returns the id(database generated) and protein\_id field from the Protein model.

**api/pfams/<int:taxa\_id>/**

Text

Description automatically generated

This is the code that can be found in the api.py file. The PfamList class uses the ListPfamSerializer to list domain\_id and domain\_description with taxa\_id. Because generics.ListAPIView is unlike generics.GenericAPIView as it won’t take in lookup\_field so there is a need to create a get\_queryset function to filter the taxa\_id.

Text

Description automatically generated

This is ListPfamsSerializer which can be found in the serializers.py file. It returns the id(database generated in domain table) and pfam\_id(domain\_id and domain\_description from Pfam) field from the Domain model.

**api/coverage/<str:protein\_id>/**

Text

Description automatically generated

This is the code that can be found in the api.py file. The coverage function does not use generics to generate the response instead I use api\_view as there is no need to create a model for this api. This function only accepts GET request and a protein\_id. Firstly, it will check whether the protein\_id exist in the protein table or not if it doesn’t exist then 404 will be returned with a status of protein not found. Next using the protein found which is guaranteed to be 1 record as explained in the [models](#_Models) section and search it in the domain table. After retrieving the list of domains, it will iterate through the list and sum up the start and stop value. Lastly the response will be returned with a field coverage and the value which is derived by using the (stopTotal-startTotal)/the protein length.

## Test Case

For test case I won’t cover on model\_factories.py but most of the code there are mock data and some data that will be used in tests.py file to easier compare the result.

**tests.py**

A screenshot of a computer screen

Description automatically generated with medium confidence

This few function are helper function which will be used in most of the test case.

Text

Description automatically generated

Text

Description automatically generated

This is the test case for create protein api. The first test case test\_create\_protein\_success test whether creating of a new protein is successful by checking the database after the request. The second test case test\_create\_protein\_without\_sequence\_success test whether if a protein without sequence can be created. The third test case test\_create\_protein\_duplicate\_protein\_id\_failure is to test if duplicated protein\_id can be inserted. The fourth test case test\_create\_protein\_missing\_protein\_id\_field\_failure is to test if the api will return 400 if protein\_id is not provided. The fifth test case test\_create\_protein\_missing\_taxonomy\_field\_failure is to test if the api will return 400 if taxonomy is missing. The sixth test case test\_create\_protein\_missing\_length\_field\_failure is to test if the api return 400 if length is not provided. The last test case test\_create\_protein\_missing\_domains\_field\_failure is to test if the api return 400 if domains is not provided.

Text

Description automatically generated

This is the test for protein detail api. The first test case test\_protein\_detail\_return\_success will test if the api return the data when it is found. The last test case test\_protein\_detail\_return\_failure will test if the api return 404 if protein\_id is not found.

Text

Description automatically generated

This is the test for pfam detail api. The first test case test\_pfam\_detail\_return\_success which will test the if the pfam is being returned. The last test case test\_pfam\_detail\_return\_failure will test if the api return 404 if the pfam is not found.

Text

Description automatically generated

This is the test for list of protein api. The first test case test\_protein\_list\_return\_success will return the list of protein\_id if the taxa\_id can be found. The last test case test\_protein\_list\_empty\_result\_return\_success will test if the api return an empty list if it cannot find any protein\_id related to the taxa\_id.

Text

Description automatically generated

This is the test for list of pfam api. The first case test\_pfam\_list\_return\_success will return the list of pfam if taxa\_id is found. The last test case test\_pfam\_list\_empty\_result\_return\_success will test if the api return an empty list if the taxa\_id is not found.

Text

Description automatically generated

This is the test for coverage api. The first test case test\_coverage\_return\_success will return a response with a calculated coverage. The last test case test\_coverage\_return\_failure will test the api in returning 404 if the protein\_id is not found.

The remaining test case in the file is serializer test case which I won’t go through as this report is getting too lengthy but the serializer test case basically test the column it returned.

# Conclusion

To summarize this coursework, I have learnt a lot from Django and the thinking in normalizing the dataset provided. It was a fun journey for this coursework, and I am sorry for my bad English and some of the thing I might not explained it clearly. Lastly, thank you for your time for reading this lengthy report.