Bioinformatician Engineer

Nuritas - Technical Assignment

Mohammad Eslami

1. Deployment

A full instruction through which the application can be deployed has been provided in the *README.md* file. The application has been dockerized and will be running on the docker containers.

A guide on how to populate the data is also provided alongside with other commands required for running different parts of the application.

2. Application

2.1 Database

MySQL database is used for this application. The main reason for this choice comes down to my experience with working with MySQL database. It is easy to setup and compatible with Django framework. There are also several documentation on how to use MySQL in Django.

An alternative choice would be to use NoSQL database and store the data as json objects which would make it easier to work with list fields.

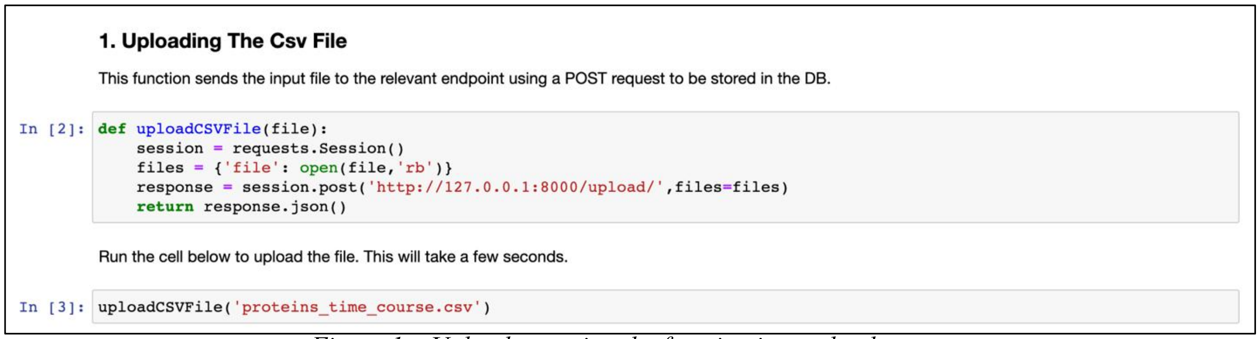
2.2 Protein Model

The protein ORM been created based on the schema given. The fields types shown in Table (1). Since some of the protein abundance values lie beyond the regular Integer limit, BigIntegerField are used for storing those values. Using of BigIntegers.

2.3 Populating Data

In this application, there are two ways to populate the data:

1) Use the function provided in the notebook, which basically sends a post request containing the csv file to the relevant endpoint in which the csv file is read and parsed, the protein objects will be created and the data will be stored in the database.



1. Uploading The Csv File

This function sends the input file to the relevant endpoint using a POST request to be stored in the DB.

```
In [2]: def uploadCSVFile(file):
            session = requests.Session()
            files = {'file': open(file,'rb')}
            response = session.post('http://127.0.0.1:8000/upload/',files=files)
            return response.json()
```

Run the cell below to upload the file. This will take a few seconds.

```
In [3]: uploadCSVFile('proteins_time_course.csv')
```

Figure 1 – Upload csv using the function in notebook

Column	Chosen Filed Type
ProteinID	IntegerField
AverageMass	TextField
Description	TextField
ZeroHrProteinAbundance	BigIntegerField
HalfHrProteinAbundance	BigIntegerField
TwoHrProteinAbundance	BigIntegerField
ThreeHrProteinAbundance	BigIntegerField
FourHrProteinAbundance	BigIntegerField
FiveHrProteinAbundance	BigIntegerField
SixHrProteinAbundance	BigIntegerField
NineHrProteinAbundance	BigIntegerField
TwelveHrProteinAbundance	BigIntegerField
TwentyFourHrProteinAbundance	BigIntegerField
CellularProcesses	TextField
ProteinFunctions	TextField
ReactomePathways	TextField

Table 1 - Fields of the protein model and their types

2) Or use the link provided in the admin profile under Protein folder in the core section. By clicking on the link, you will be asked to select and upload the csv file.

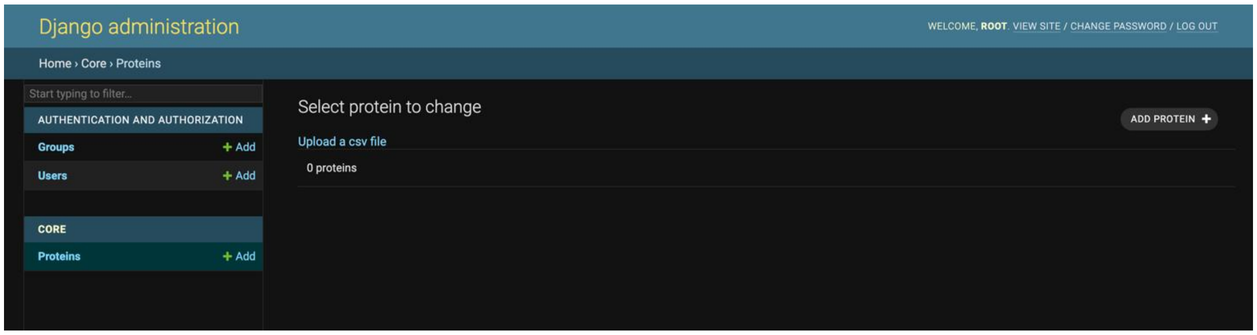


Figure 2 – Upload csv from the admin portal

2.4 Bulk Create

The bulk create function in Django ORM was used to store the protein objects in the database. This methods showed a significant drop in the time taken for the population process, compared to saving objects one-by-one or using transactions library.

2.5 Serializers

For the purpose of converting ORM models to json objects, a set of serializer classes are defined to serialize the results of the ORM queries, making it suitable to be returned by the views.

2.6 Rest Framework

For some of the views in which a list of records are to be returned, the ListAPIVIew from the rest framework has been used which provides a set methods which take care of returning a list of the records.

3. Notebook

A set of functions has been implemented in the Jupiter notebook which interacts with the REST API of the application and retrieve data based on the inputs.

3.1 Uploading Csv File:

Signature: *def uploadCSVFile(file)*

Description: This function sends the input file to the relevant endpoint using a POST request to be stored in the DB.

Endpoint: *http://127.0.0.1:8000/upload/*

3.2 Retrieve Protein Information Excluding Numeric Values:

Signature: *def getProteinInfo(proteinID)*

Description: This function takes a Protein ID as input, sends the ID to the relevant endpoint using a GET request. It then returns the protein information excluding the timepoint numeric values.

Endpoint: http://127.0.0.1:8000/info

3.3 Retrieve Protein Information, Only The Numeric Values

Signature: *def getProteinAllTimePointAbundance(proteinID)*

Description: This function takes a Protein ID as input, sends the ID to the relevant endpoint using a GET request. It then returns only the numeric timepoint values of the protein.

Endpoint: *http://127.0.0.1:8000/time-point*

3.4 Retrieve Protein Abundance On a Specific Timepoint

Signature: *def getProteinSingleTimePointAbundance(proteinID, timePoint)*

Description: This function takes a Protein ID and the required Time Point as inputs, sends them to the relevant endpoint using a GET request. It then returns only the abundance value measured on the given timepoint.

Endpoint: *http://127.0.0.1:8000/single-field-time-point*

3.5 Retrieve Proteins With Threshold

Signature: *def getProteinSingleTimePointAbundance(proteinID, timePoint)*

Description: This function takes a time point and a threshold. It then returns the proteins whose abundances are above the given threshold in the specified the timepoint.

Endpoint: *http://127.0.0.1:8000/list*

3.6 Retrieve Proteins With Specific Celluar Process

Signature: *def getProteinwithCellularProcess(cellularProcess)*

Description: This function takes a Celluar Proces (e.g. 'cell adhesion') returns the proteins which contain in this process in their Celluar Processes.

Endpoint: *http://127.0.0.1:8000/process*

There are also some other functions implemented in the notebook to create plots from the results of some of the functions. A sample of which is shown in the next section.

4.Visualization

Since the convention for visualizing the time series data is to use a line or a bar chart, to show the timeseries data which are the protein abundance measured on specific timepoints, A line and a bar chart are provided both for single and multiple records.

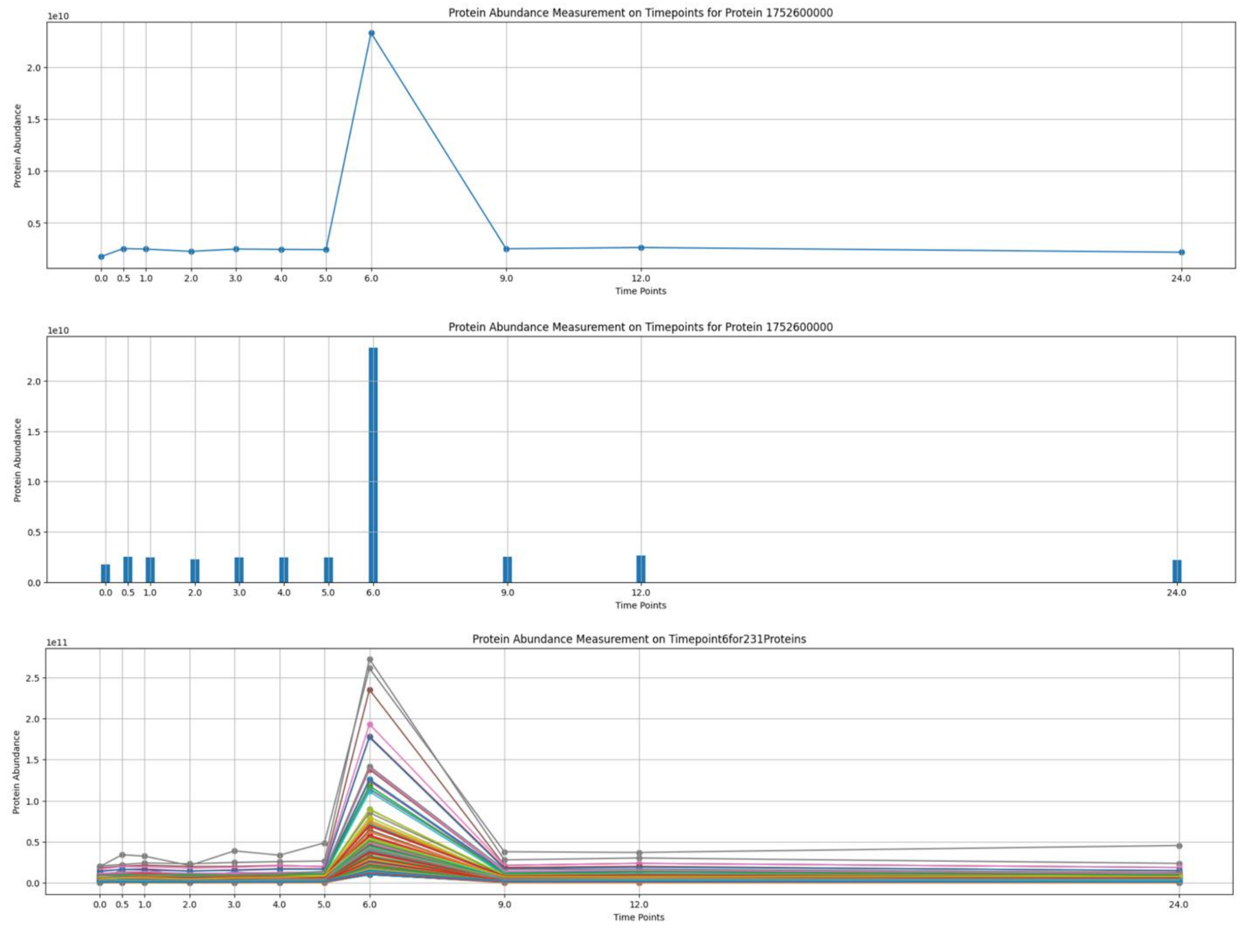


Figure 3 – Protein Abundance Visualization

5. Testcases

A set of 21 testcases have been written to test the functionality of the views. The instruction on how to run the test cases has been provided in the *README.md* file.

6. Env Files

The database credentials are placed in env files. The env files are present in the main folder for this version.

7. Application Structure

```
BioinformaticianTechnicalAssignment
|
core
|
|_ _ _ _ views.py : The Class-based REST APIs
|
|_ _ _ _ settings.py: Application settings including Database settings
|
|_ _ _ _ urls.py: url paths for the endpoints
|
|_ _ _ _ models
|        |
|        |_ _ Protein.py: Protein ORM
|        |
|        |_ _ Serializer.py: Serializer Classes
|
|_ _ _ _ migrations
|
|_ _ _ _ Dockerfile: Application dockerfile
|
|_ _ _ _ docker-compose.yml
|
|_ _ _ manage.py
|
|_ _ _ DB-SQL
|       |
|       |_ _ Dockerfile: MySQL Database dockerfile
|
|_ _ Notebook.ipynb
```