Bioscience Research RESTful Web Service

Table of Contents

Project Documentation

Location	bio_project/README

The bio_project folder contains project documentation (README), which includes important project information on how to unpackage and run application, run unit tests and notes about project's state (**Figure 1**).

```
=====================================================

Operating System: Project was developed on MacOS Ventura 13.3.1 (a)
Source-code Editor: Visual Studio Code
Browser Testing: Project was tested in Chrome Browser
Python version: Python 3.10.9

Database: SQLite3
Configuration: os.path.join(BASE_DIR, 'db.sqlite3')
Path: relative  -> 'os.path.join' joins base directory with 'db.sqlite3' to
make the database location relative to the location of the Django project

=====================================================

Project Setup and Installation using Terminal:

1. Unzip the project folder

2. Navigate to the project directory
        cd [file name]

3. Create a virtual environment
        python3 -m venv biovenv

4. Activate the virtual environment
        source biovenv/bin/activate

5. Install the dependencies
        pip install -r requirements.txt

6. Navigate to the source folder
        cd src

7. Migrate the database
        python manage.py migrate

8. Load Bulk Data
        python scripts/load_data.py

9. Start the development server
        python manage.py runserver

10. Visit localhost to view the project
        http://127.0.0.1:8000/


If installation was unsuccessful
•       review installation errors in terminal
•       review documentation to solve errors

=====================================================
```

```
asgiref==3.6.0
beautifulsoup4==4.12.2
certifi==2023.5.7
charset-normalizer==3.1.0
coreapi==2.3.3
coreschema==0.0.4
Django==4.2
django-bootstrap4==23.1
django-extensions==3.2.1
djangorestframework==3.14.0
drf-yasg==1.21.5
factory-boy==3.2.1
Faker==18.7.0
idna==3.4
inflection==0.5.1
itypes==1.2.0
Jinja2==3.1.2
MarkupSafe==2.1.2
packaging==23.1
python-dateutil==2.8.2
pytz==2023.3
requests==2.30.0
ruamel.yaml==0.17.26
ruamel.yaml.clib==0.2.7
six==1.16.0
soupsieve==2.4.1
sqlparse==0.4.4
uritemplate==4.1.1
urllib3==2.0.2
```

Figure 1: Setup instructions and Packages (see readme.txt)

Application Overview

Basic Functionality

Functionality	Yes/No	Description
models	yes	models section described how models were created and database design
migrations	yes	• during the initial project setup to sets up the database tables as per models. • every time changes were made to models.py.
form	yes	• Search Result Forms • Create New Protein Form
validators	yes	forms, views, JS
serialization	yes	serialization is done in serializers.py to convert models into a format that HTTP understands
DRF	yes	serializers
URL routing	yes	urls.py
unit testing	yes	tests folder contains tests for forms, models, serializers, views

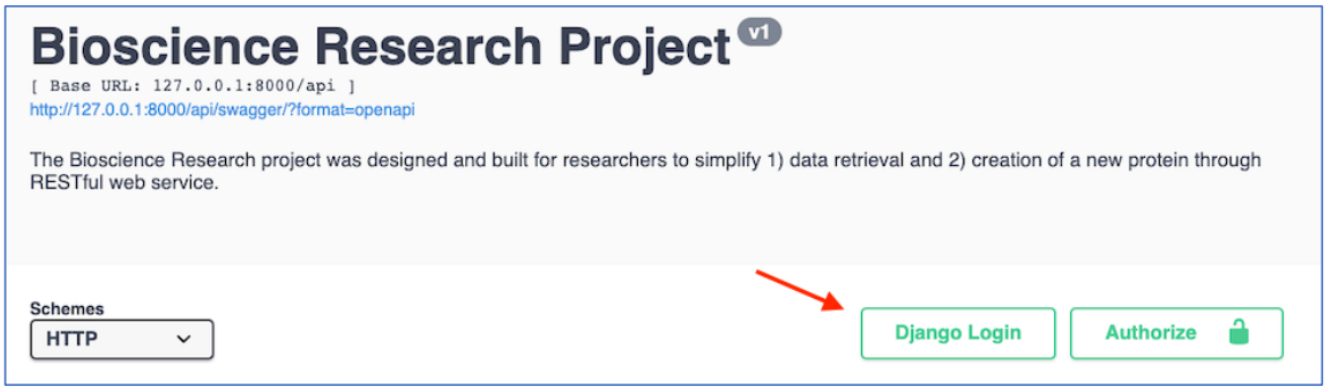
Home Page

Navigation links were created for grading purposes to showcase that requirements were met. Please see **Appendix** for details.

Django Admin

Django Admin Button on the navigation bar

Django Admin Button in the Swagger view




```
=====================================================

Django Admin Login

Username: XXXXX
Password: XXXXX

In the event of login problem, please create a new superuser and follow terminal instructions

        python manage.py createsuperuser

=====================================================
```

SPA Challenges

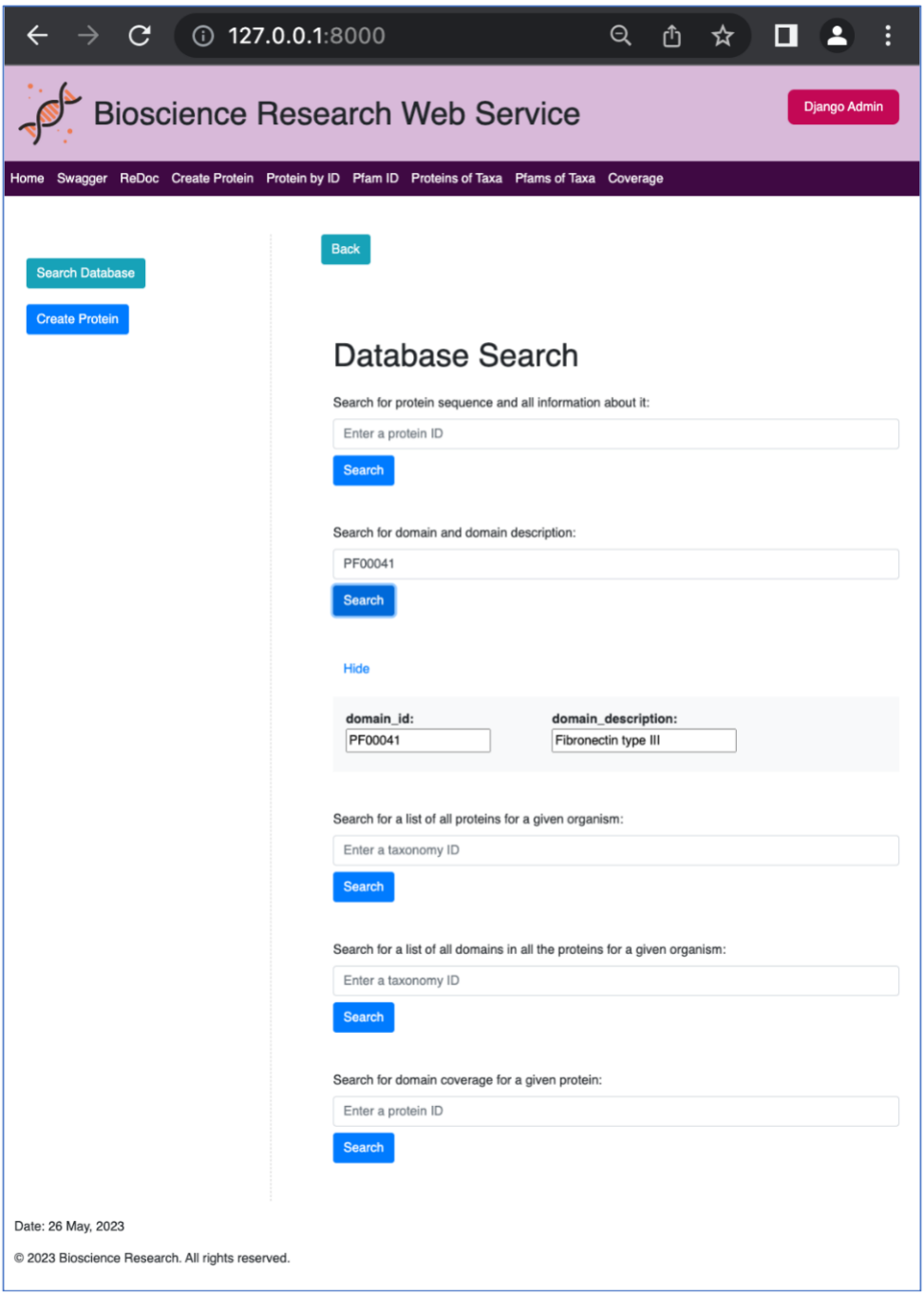
The initial idea was to make a Single Page Application (SPA), but unexpected behavior occurred during second half of the development phase while combining all previously developed elements into spa.html. Create New Protein Form was not showing in the SPA. My research indicated that if the form was created in the form.py, added to the spa.html template, used event listeners to toggle on/off the display of form and used Fetch API to handle form submission, it should have worked. (developer.mozilla.org, n.d.) (developer.mozilla.org, n.d.) (developer.mozilla.org, n.d.). However, the form only worked as expected when it was not part of the SPA, otherwise it was hidden from the user. All necessary code was showing in *Google Inspector*, but it was not displaying in the browser. Initially it was suspected that the issue was with the event listener, but debugging showed it was not the problem. It's suspected the issue is with some sort of conflict within the Django framework. After a week of failures, it was decided to move on with the project as a two-page application. The solution to the problem may become more apparent in the second part of the semester. The finished website has a two-sided view home page, on the left two buttons displayed and on the right search fields displayed when the first button is triggered.

Home page view

Search Database

Search Database Button – when the button is clicked the search fields appear on the right of home page where user can search database. The search result is displayed in a table format below "Search" button. The hide/unhide search result feature was implemented to give user control over the amount of information displayed.

SPA view when Search Database button is triggered and search result is displayed for Pfam ID



Search Database Implementation

The search database feature was implemented by using JavaScript and HTML.

- **createSearchResultForm()** checks the data and assigns label and an input field, in the event the data belongs to an object, the sub-form is created. In other words, this function handles the creation of HTML elements.

- **SearchSectionSetup()** uses **getElementById()** to find input and button elements and adds button.onclick event handler to the button. By using "try" method it validates user input when button is clicked and gets data from API and displays it; if there is an error it gets raised through "catch" method. In other words, this function sets up search boxes.

- **SearchSections()** calls **SearchSectionSetup()** for five searches and a callback function.

- **fetchDataURL()** sends an API request to the specified URL and throws an error if the there is an error, otherwise returns the response data.

- **showData()** gets data from the URL using **fetchDataURL()**, then creates and displays a search result form with data. It shows error message in the event of error during getting or displaying the form data.

- **spa.html** creates Single Page application that is split into two parts on the left *Search Database* and *Create Protein* buttons displayed and on the right search fields displayed when the first button is triggered.

Search Validation

Validation for white spaces and empty string is done in *search_database.js*. If input is invalid, error is thrown (alert box shown with error message). Data validation on the server-side (*views_api.py*) is done by API through filtering *querySet*, retrieving a record (CoverageView), using URL parameters and serialization. The validation on the server-side is not explicit. Unfortunately, due to timing constrains it was not possible to research on how to make validation implicit.

Create New Protein

New protein can be created in three ways:

New Protein Creation Option	URL
Custom Form	http://127.0.0.1:8000/protein/create_new_protein/
Django Framework view	http://127.0.0.1:8000/api/protein/
Swagger	http://127.0.0.1:8000/api/swagger/

Create New Protein Form (Custom Form)

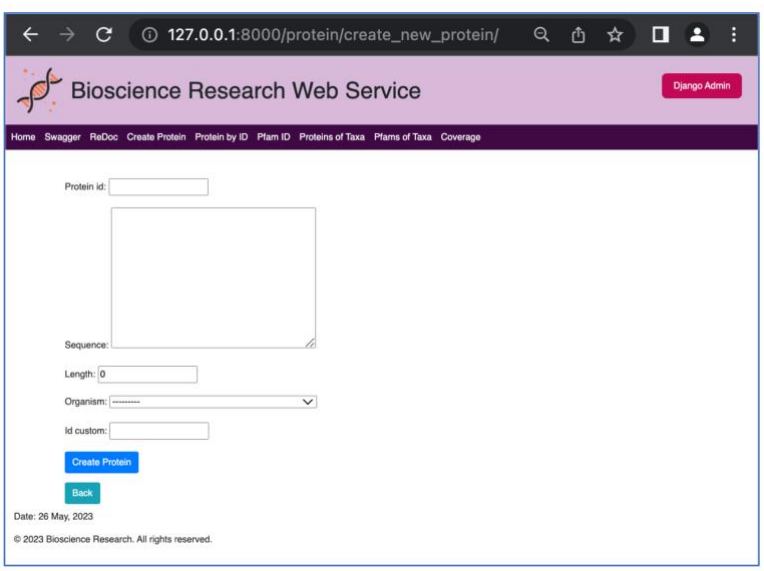
Create Protein Button – when button is clicked the user is redirected to a new page the *create_new_protein.html* where they can input data to add a new protein to database. The back button was added to allow user to return to the home page.

Create New Protein Form Implementation

The following files were created using separation of concerns principals to allow users to construct new proteins.

file name	purpose
models.py	responsible for creation of the database tables/models which are used by forms.py classes.
forms.py	responsible for handling the creation and validation of form fields using Django's forms.ModelForm
views_protein.py (Business Logic)	responsible for POST request and data validation
create_new_protein.html (User Interface Rendering)	displays the form to the user and corresponding messages when user submits the form to indicate if submission was successful or if it failed to the user

Create New Protein Form



Create New Protein Form Validation

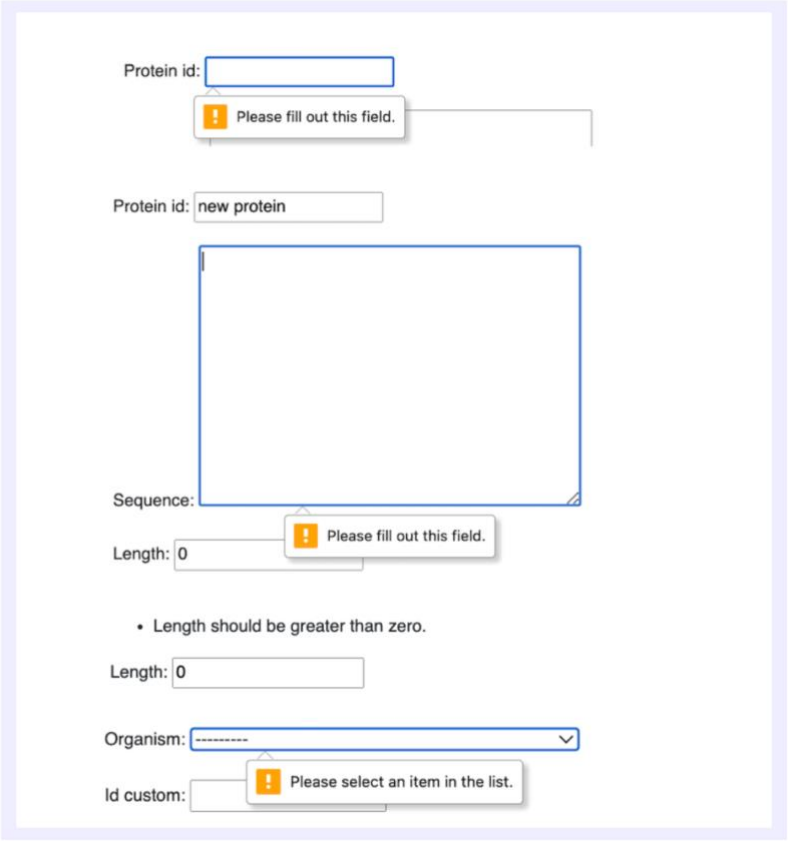
Location	views_protein.py, form.py

Create New Protein Form has three types of validation:

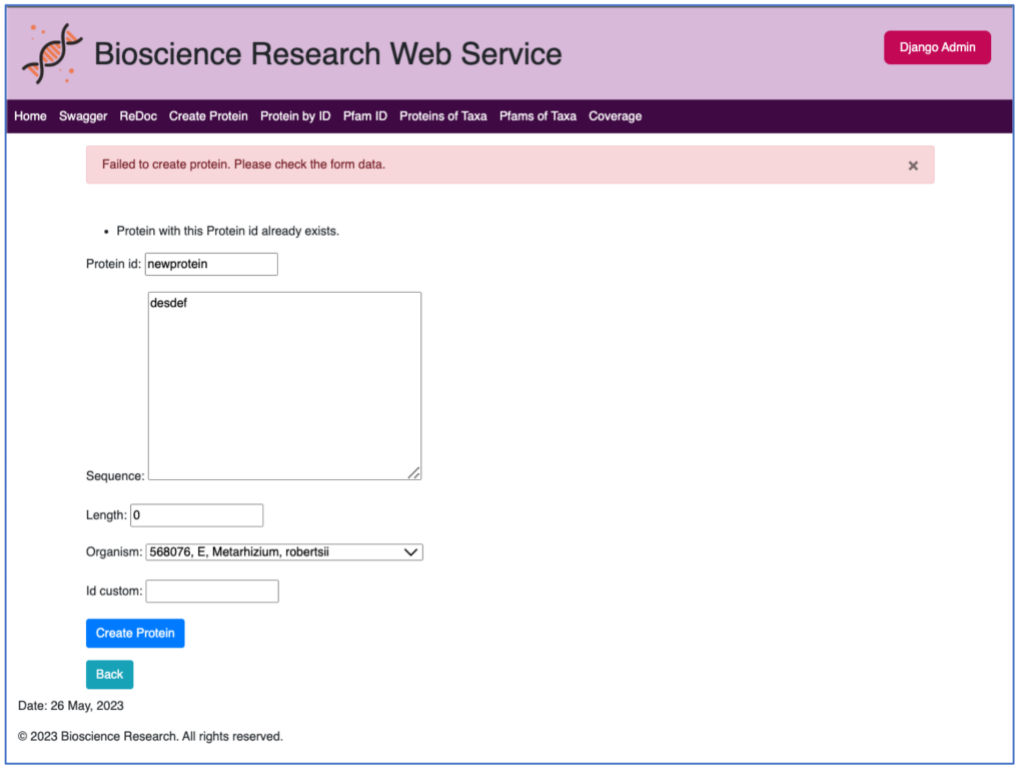
1. **Validation of Forms**: Each form is created with 'forms.ModelForm' this is Django's built-in form validation where each field is checked for data type (*string, int* etc.) and constraints specified in the model. (djangoproject.com, n.d.)
2. **Validation of Formset**: *DomainAssignment* form has many-to-many relationship between Protein and Domain. *DomainAssignmentFormSet* was created to validate this relationship. (djangoproject.com, n.d.)
3. **Custom Validation:** 1) *data_validation()* checks if all required fields are in the data, if not, ValidationError is raised. 2) *data_exists()* validates submitted data uniqueness by checking database for an object with the same 'id', if it exists, ValidationError is raised. (djangoproject.com, n.d.)

If submitted data doesn't meet any of the above validations, an error message will be shown to the user. Important to note that data will only be saved if submission is successful.

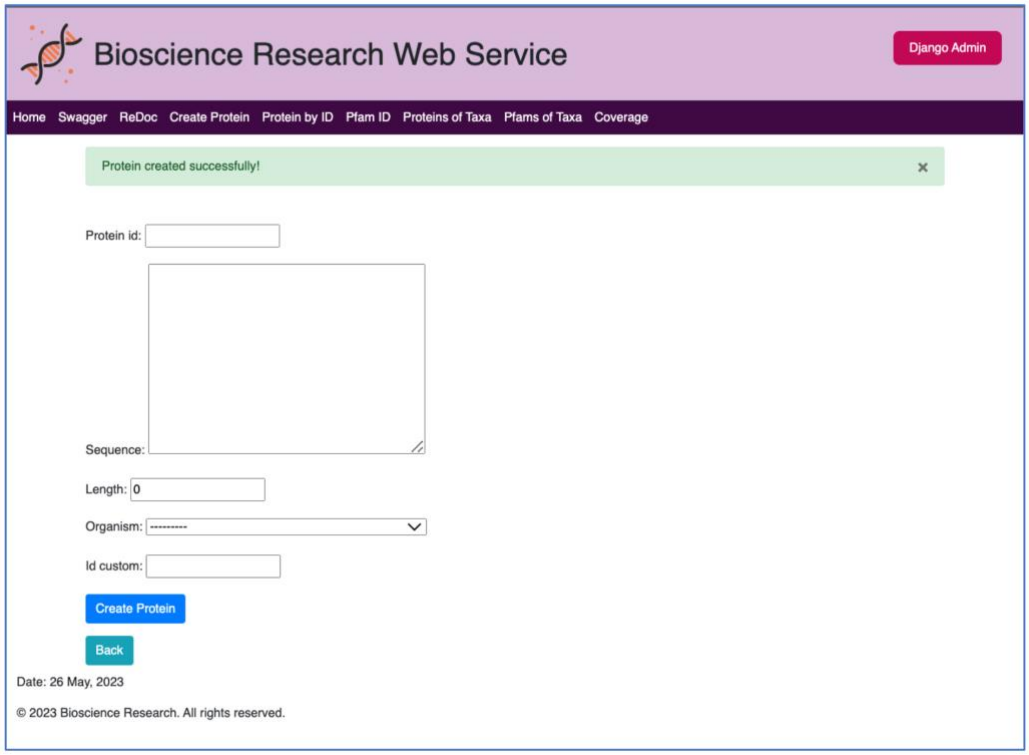
Missing Fields



Failure Message



Success Message



JavaScript (new_protein_form.js)

Code Logic

Front-End:
createNewProtein() called → object + protein data
createNewProtein() sends POST request to → '/api/protein/create/' →
→ new protein data sent (JSON Format)

Server Side:
Django receives request → SerializerForProtein validates data
If data is valid, create a new protein, save to DB, and return success response
If data is invalid, return a validation error response

Front-End:
createNewProtein() checks response status
If not ok, error is thrown
If ok, redirect user to refreshed page

Create New Protein and Validation (Django Framework)

The *SerializerForProtein* uses *ModelSerializer(DRF)* and it handles creation of serializers for model instances and *querySets*. The validation process is done by DRF's serializer using the field definitions which takes place before the *create()* or *update()* methods are called.

- **Fields:** When data submitted for serialization the *is_valid*() method is called which validates all fields (protein_id, sequence, taxonomy, length, domains). Error message is returned if validation is failed.

- **Objects:** *validate()* method handles object validation. Any errors are raised during validation process and the process of creating a new protein is stopped.

Important to note that the data is only valid if both fields and objects pass validation. Valid data is passed to *create()* or *update()*, depending on the case, and then used to create or update instances of models (Organism, Protein, Domain, Pfam, DomainAssignment).

Figure 2: DRF Create New Protein

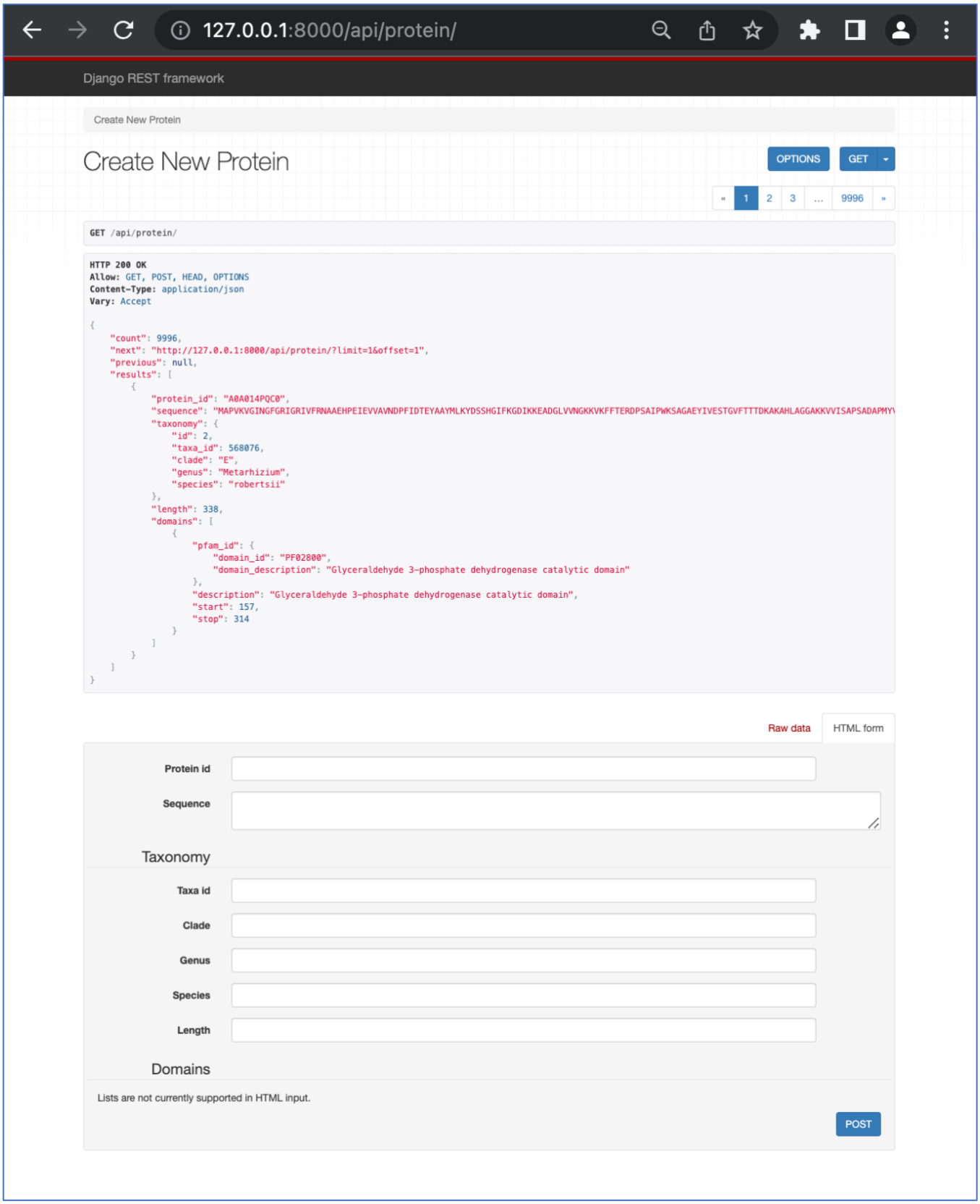


Figure 3: DRF Failed Protein Creation

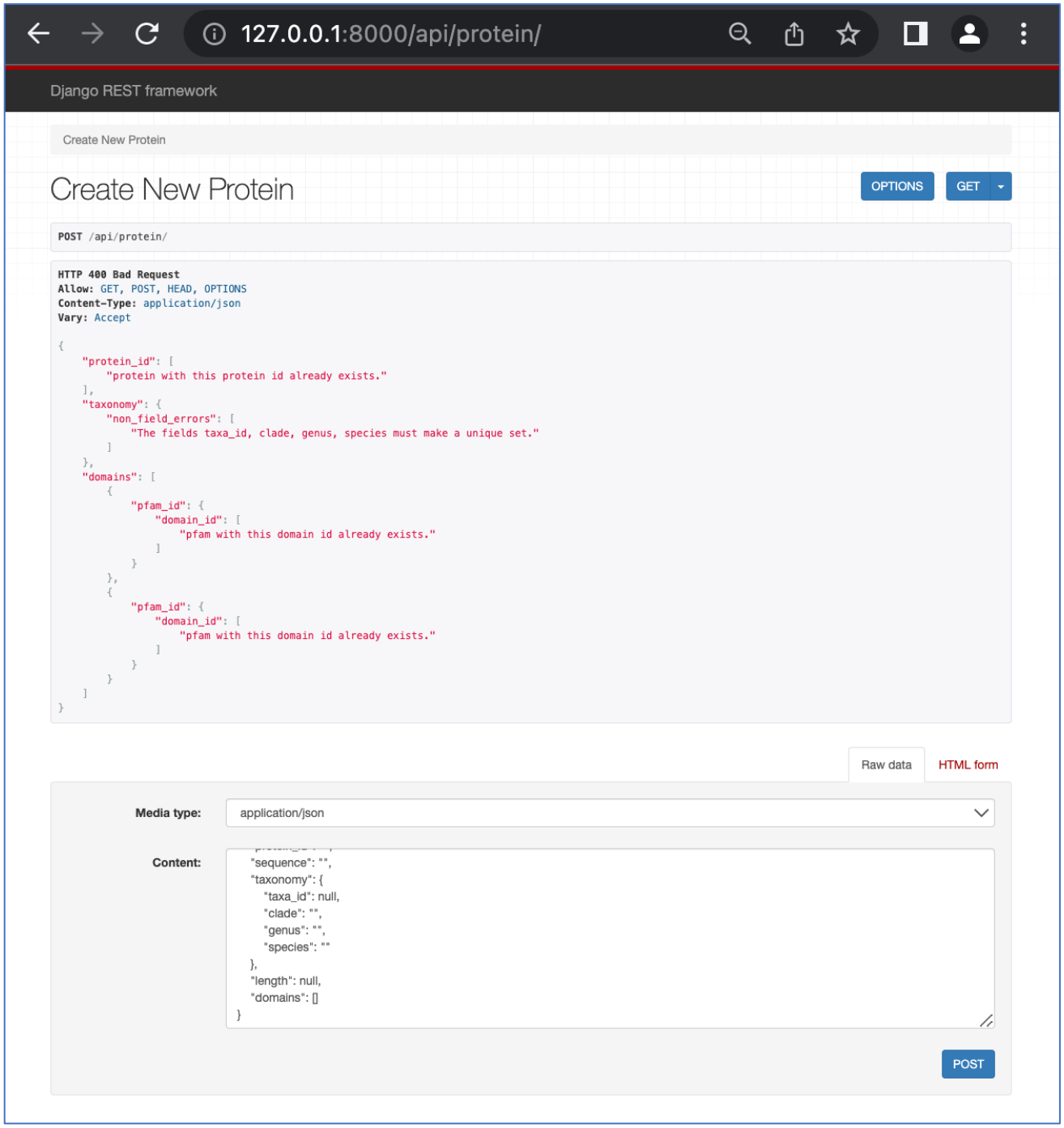
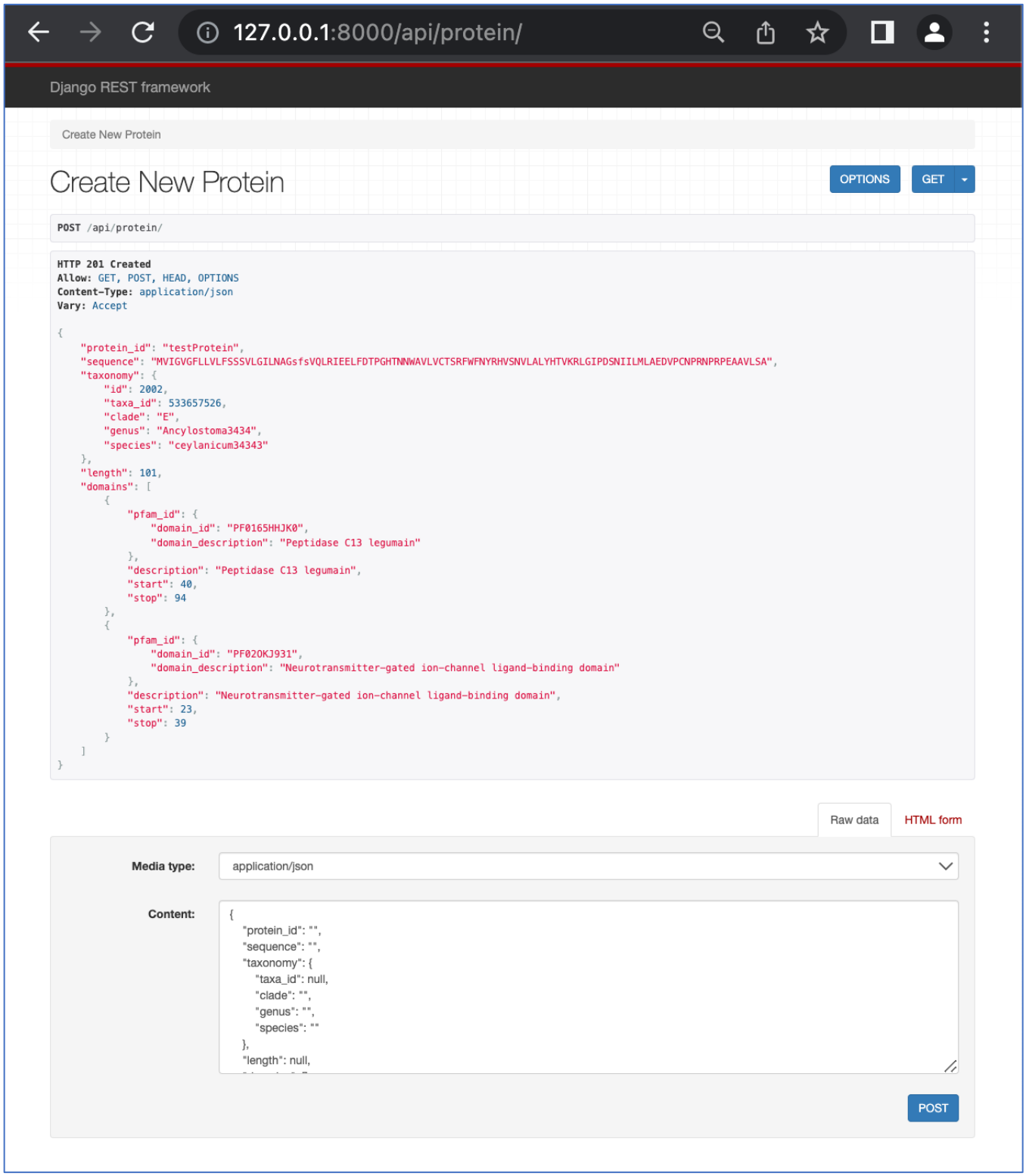


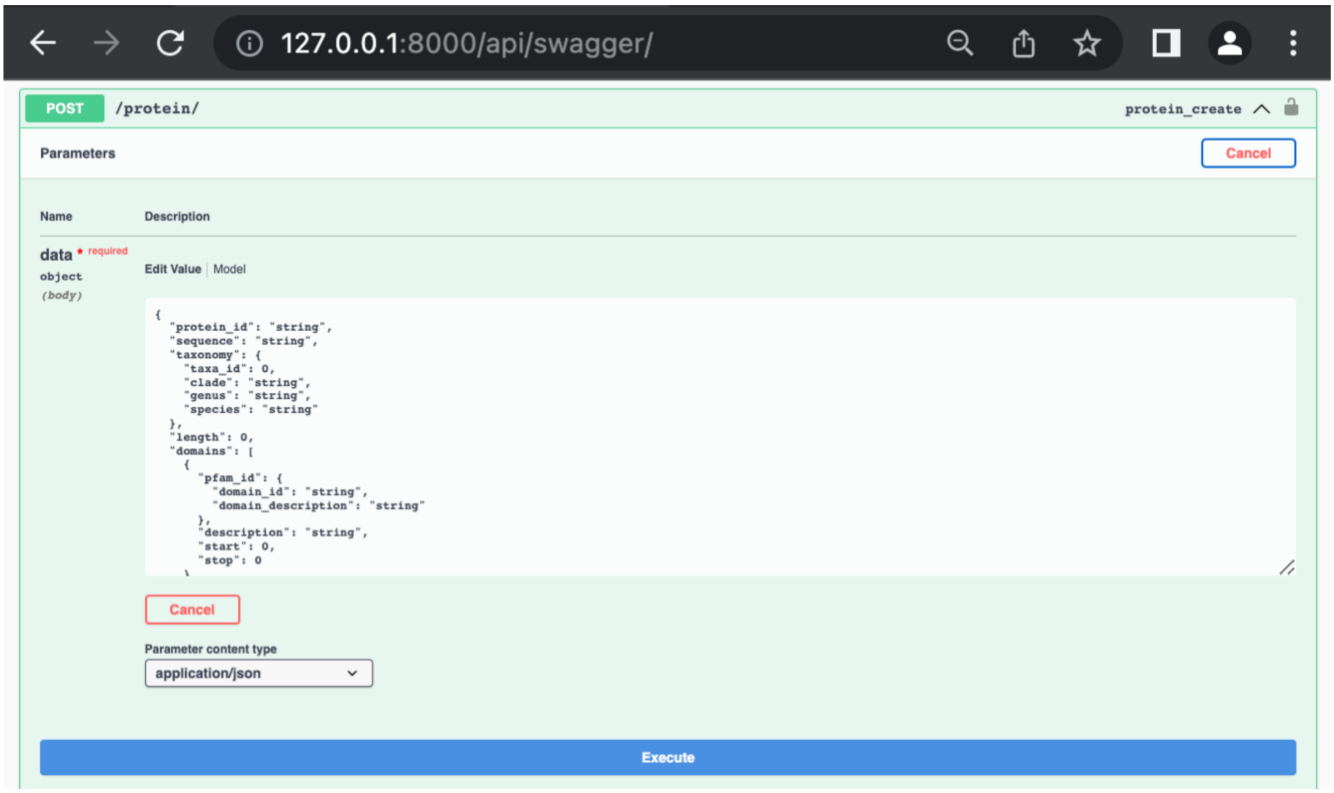
Figure 4: DRF Successful Protein Creation



Swagger and ReDoc

The DRF-YASG (Django Rest Framework Yet Another Swagger Generator) and ReDoc were discovered accidentally while performing research for another part of the project (swagger.io, n.d.), (openapis.org, n.d.), (redocly.com, n.d.). Swagger and ReDoc were implemented after completion of frontend Create New Protein Database functionality with JavaScript. In retrospect, it would have been great to have this helpful and easy to implement tool for creating new protein, creating schema and generating interactive API documentation earlier in the development as it would have saved a lot of time with the frontend functionality. Nonetheless, the Create New Protein page I created, has a minimalistic design, which offer a less distracting UX for end-user.

Create New Protein with Swagger

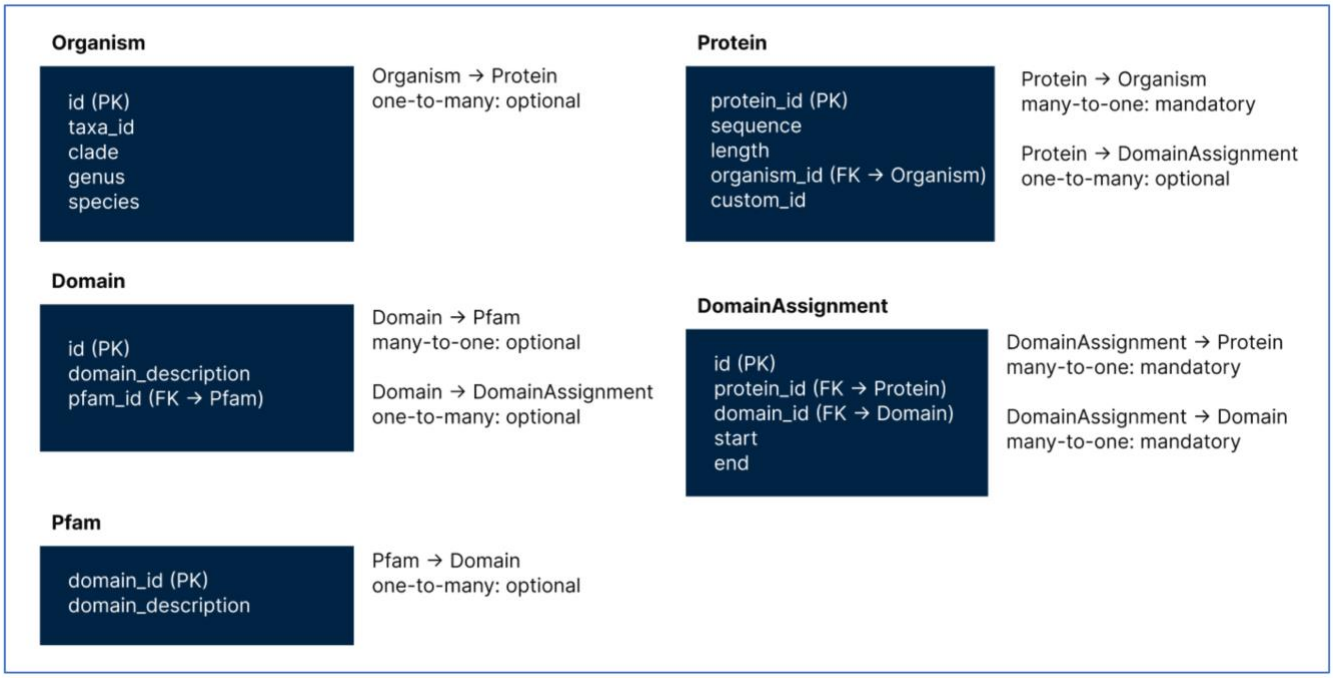


Data

Data Model

The database was designed based on understanding of the assignment. It was a confusing puzzle to solve, which resulted in the following relationship representation. This project is the most complicated database project done to date. Regretfully, I'm unsure that my solution is the right solution, although the project works as expected. The following resources were used to brush up on the relational database. (Peterson, 2023), (lucidchart.com, n.d.), (stanford.edu, n.d.). The database structure is located in the *models.py* file. ERD was created in Figma to add additional explanatory comments.

ERD (Entity Relationship Diagram)



Data Location

Location	bio_project/src/data_files

The project data was loaded from the following files:

1. "assignment_data_sequences.csv"
2. "assignment_data_set.csv"
3. "pfam_descriptions.csv"

Data Loading Script

Location	bio_project/src/scripts

The data files are imported into the SQLite database using the *load_data.py* script using the following approach.

Loading of Bulk Data

The loading of three CSV files has the following logic. This approach was chosen as it complies with separation of concerns principle.

1. **create loading functions**: *load_assignment_data_sequences()*, *load_assignment_data_set()*, *load_data_pfam_descriptions()*.
2. **load files**: "*assignment_data_sequences.csv*, *assignment_data_set.csv*, *pfam_descriptions.csv*"
3. **check for duplicates:** remove with *delete_duplicates()* if found
4. **add data to database**

Why get_or_create() was used:

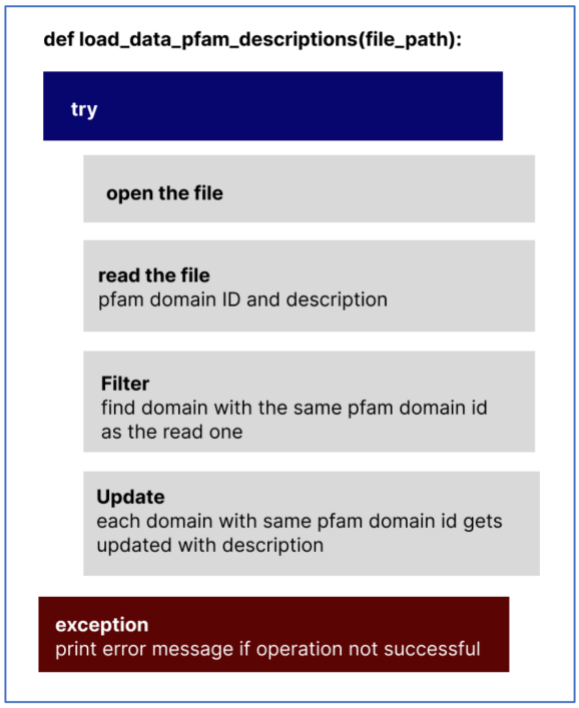
The get_or_create() is designed to check if existing database record exists and if it doesn't it gets created. Considering that the Researchers will be updating csv files in the future, it was decided to use this method to load data into the database. The Django documentation doesn't make reference to speed of this method (docs.djangoproject.com, n.d.), but if method checks for duplicates than it will go through every row and that can be time consuming, however, my objective was to make sure that data in the database has unique records. Method to remove duplicates was created to handle duplicate entries.

Why bulk_create() was not used:

- SQLite has a limitation on number of records created in one query *"The batch_size parameter controls how many objects are created in a single query. The default is to create all objects in one batch, except for SQLite where the default is such that at most 999 variables per query are used."* (djangoproject.com, n.d.), (sqlite.org, n.d.), (sqlite.org, n.d.) This is a big disadvantage as the midterm instructions specifically asked for SQLite database to be used.
- SQLite also has another limitation, if *bulk_create()* used with SQLite automatic incrementation of inserted record (ID) will not work. (djangoproject.com, n.d.)
- The *bulk_create()* only handle loading of large data at once, but it doesn't check for duplicate entries. (djangoproject.com, n.d.)

To summarize, the *get_or_create()* offers bigger advantage for recurrent data loading as it checks for duplicates. It was decided to use a possibly slower performing method to load data for this project as the combination of SQLite and *bulk_create()* method offered greater disadvantages then *get_or_create()* such as lack of auto incrementation of ID and lack of check for duplicate entries.

Loading Functions



def load_assignment_data_sequences(file_path)

try

open the file

read the file
protein id and sequence

create_or_get organism
if doesn't exist: use specified default values to create it
if exists: use DB values

create_or_get protein
if doesn't exist: create a new record
if exists: use DB values

assign
sequence to protein record

save to DB

exception
print error message if operation not successful

def load_assignment_data_set(file_path)

try

open the file

read the file
protein id, taxa id, clade, genus, species, domain description, pfam id, start, end, length

create_or_get organism
if doesn't exist: use specified default values to create it
if exists: use DB values

create_or_get protein
if doesn't exist: create a new record and connect it to organism
if exists: use DB values

update
length of protein and custom id

create_or_get pfam domain
if doesn't exist: create a new record
if exists: use DB values

create_or_get domain
if doesn't exist: create a new record and connect it to pfam domain
if exists: use DB values

domain to protein assignment
if doesn't exist: create a new assignment
if exists: use DB values

exception
print error message if operation not successful

Serializers

The serialization is done in **serializers.py** to convert models into a format that HTTP understands. Below is how it was achieved.

Name	Serializes	Fields/Notes
SerializerForOrganism	Organism model	all fields
erializerForPfam	Pfam model	all fields
SerializerForDomainByTaxa	Domain model	id , pfam_id
SerializerForDomainAssignment	DomainAssignment model	pfam_id, description, start and stop
SerializerForProtein	Protein model	has overriding create() and update() methods
SerializerForProteinByTaxa	Protein model	id, protein_id

Application Architecture

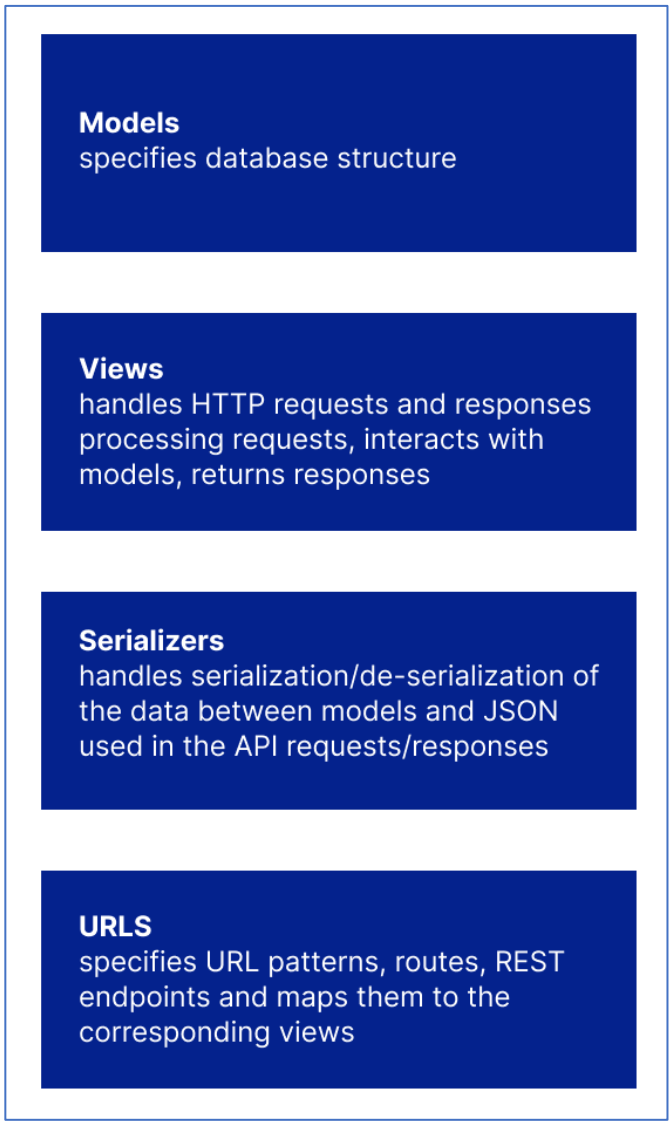
root directory	description
bioscience	created by Django
bioscience_app	application logic
data files	contains assignment_data_sequences.csv, assignment_data_set.csv, pfam_descriptions.csv and REST_specification_and_examples.txt
scripts	contains load_data.py which loads 3 csv files
db.sqlite3	SQLite database file
manage.py	the Django command-line utility

bioscience	description
settings.py	settings for the project
urls.py	urls for the project

bioscience_app	description
models.py	database schema and helper methods
views.py, views_api.py, and views_protein.py	specifies how the program reacts to user input
forms.py	form classes for user input
middleware.py	bad request middleware for forms
urls.py	URLs for bioscience_app
admin.py	configuration for the Django admin interface
apps.py	configuration for bioscience_app
tests folder	unittest cases
templates folder	HTML files
static folder	CSS, JavaScript and favicon
migrations	database migration files, used to create database schema.
templatetags	custom template tags and filters for templates

REST endpoints

The following logic was followed to implement the REST endpoints:



Unit Testing

Location	bio_project/ src/bioscience_app/tests

Tests described in **Figure 5** were implemented to test functionality of project. Initially the project started as TDD, but after enormous work load for this project and 2 other Level 6 projects, it was decided to stop using TDD. Note that *test_models.py* and *test_serializers.py* include some failing tests to illustrate that TDD was used. Furthermore, *test_models.py* and *test_serializers.py* include tests which:

a. directly creating objects in the test methods using Django's built-in ORM

b. using factory boy with factories created in *factories.py*

```
=====================================================

Unit Tests Instructions:

1. test_forms.py: 18 passing tests
        command: python manage.py test bioscience_app.tests.test_forms

2. test_models.py: 25 passing tests, 8 failing tests (commented out)

        passing tests: tests can be run with the following command
                command: python manage.py test bioscience_app.tests.test_models
        failing tests: uncomment the code to run the following command
                command: python manage.py test bioscience_app.tests.test_models

3. test_serializers.py: 11 passing tests, 7 failing tests (commented out)

        passing tests: tests can be run with the following command
                command: python manage.py test bioscience_app.tests.test_serializers
        failing tests: uncomment the code to run the following command
                command: python manage.py test bioscience_app.tests.test_serializers

4. test_views_api.py: 36 passing tests
        command: python manage.py test bioscience_app.tests.test_views_api

5. test_views_protein.py: 5 passing tests
        command: python manage.py test bioscience_app.tests.test_views_protein

6. test_views.py: 4 passing tests
        command: python manage.py test bioscience_app.tests.test_views

7. run all passing tests: 99 tests
        command: python manage.py test

8. run failing tests, please uncomment the code in test_models.py and test_serializers.py then run:
        python manage.py test

==================================
```

Figure 5: Unit Testing Instructions (see readme.txt)

According to my research the "*tearDown*" method used in the module videos is unnecessary as the Django testing framework creates a separate environment in which test database is deleted after each test (djangoproject.com, n.d.), (djangoproject.com, n.d.), (djangoproject.com, n.d.), (python.org, n.d.). Nonetheless, I've included "*tearDown*" method in some tests as that is how it was taught in the course.

```
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
..
----------------------------------------------------------------------
Ran 2 tests in 0.016s

OK
Destroying test database for alias 'default'...
```

Summary

The project was extremely labor-intensive and demanded extensive independent learning (see **Bibliography**). The implementation of data loading and model development required considerable time. Since I have limited experience with Database and Python, this project presented a significant challenge. New Protein Creation was also difficult because I began implementation with JavaScript. Later, I concentrated on creating a new protein using DRF. I struggled with nested fields during the DRF implementation until I discovered my logic flaw by inspecting the Swagger view. This user-friendly tool revealed that serializers had unnecessary constraints which were preventing correct protein creation. Although Swagger was discovered late in the project's development, it was the most important tool I was able to implement which enabled to troubleshoot various issues. In retrospect, if I prioritized creation of new protein with DRF, secondary form wouldn’t have been necessary.

Appendix

Homepage links and their purpose

Page	URL	Purpose
Home	http://127.0.0.1:8000/	allows user to get back to home page at anytime
Swagger	http://127.0.0.1:8000/api/swagger/	displays project schema
ReDoc	http://127.0.0.1:8000/api/redoc/	displays interactive API documentation
Create New Protein	http://127.0.0.1:8000/api/protein/	Creates new protein. The page will display one protein and Raw Data section for new protein creation.
Protein by ID	http://127.0.0.1:8000/api/protein/A0A016S8J7/	"return the protein sequence and all we know about it" (Module, 2023)
Pfam ID	http://127.0.0.1:8000/api/pfam/PF00360/	"return the domain and it's deacription" (Module, 2023)
Proteins of Taxa	http://127.0.0.1:8000/api/proteins/55661/	"return a list of all proteins for a given" (Module, 2023)
Pfams of Taxa	http://127.0.0.1:8000/api/pfams/55661/	"return a list of all domains in all the proteins for a given organism." (Module, 2023)
Coverage	http://127.0.0.1:8000/api/coverage/A0A016S8J7/	"return the domain coverage for a given protein" (Module, 2023)

REST Specification

Figure 6: Protein List

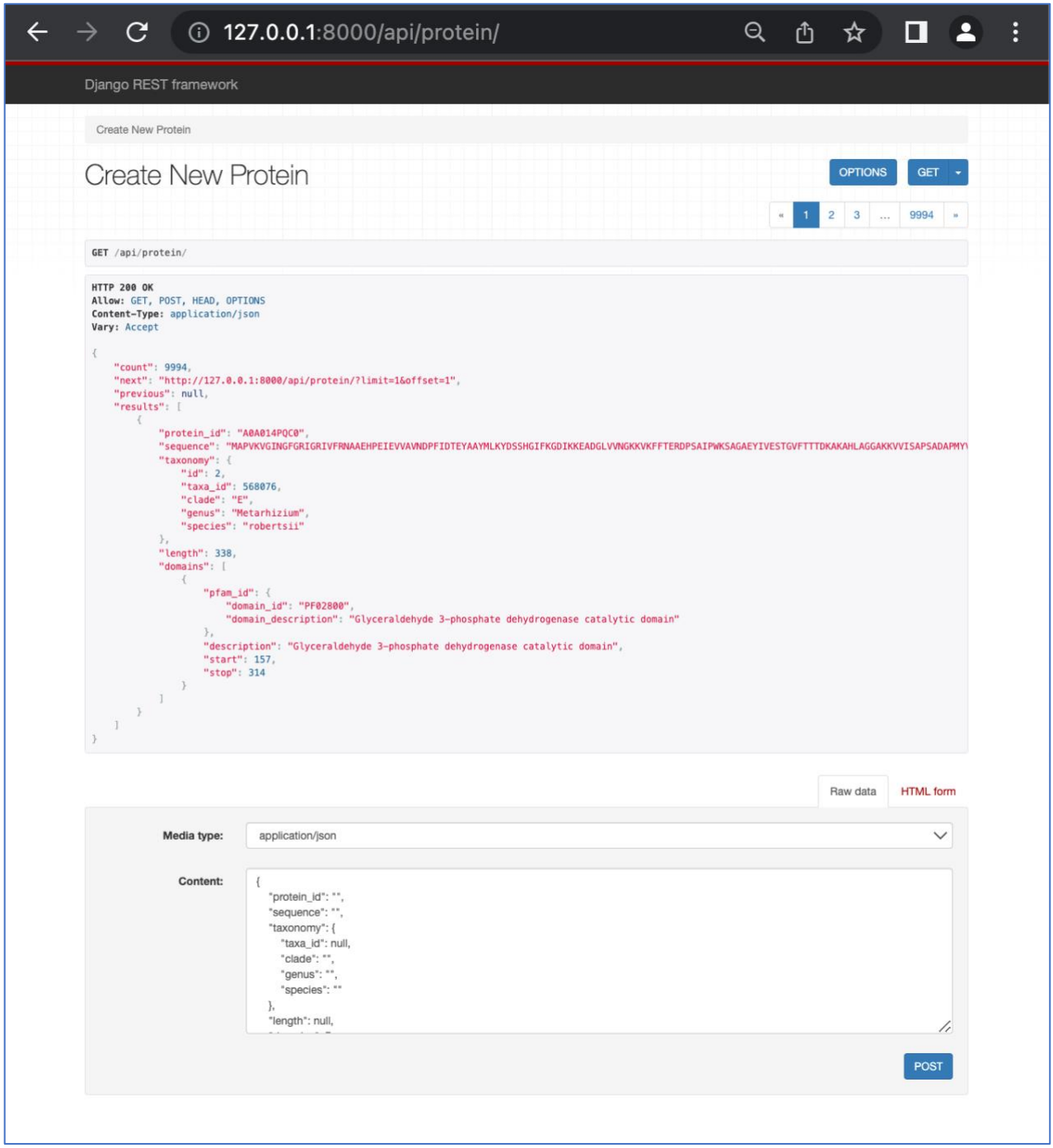


Figure 7: Protein by ID

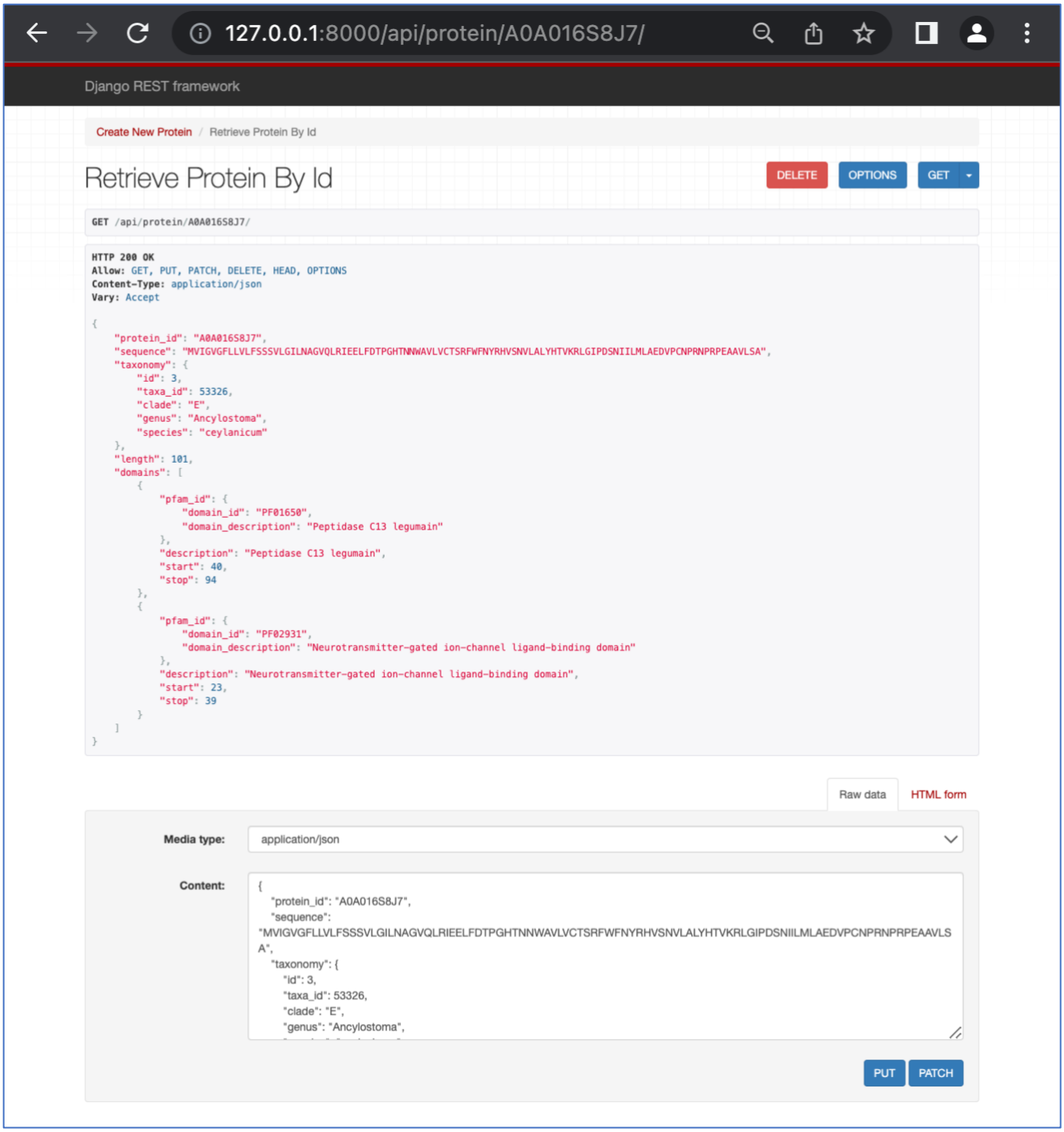


Figure 8: Pfam ID

Figure 9: Proteins of Taxa

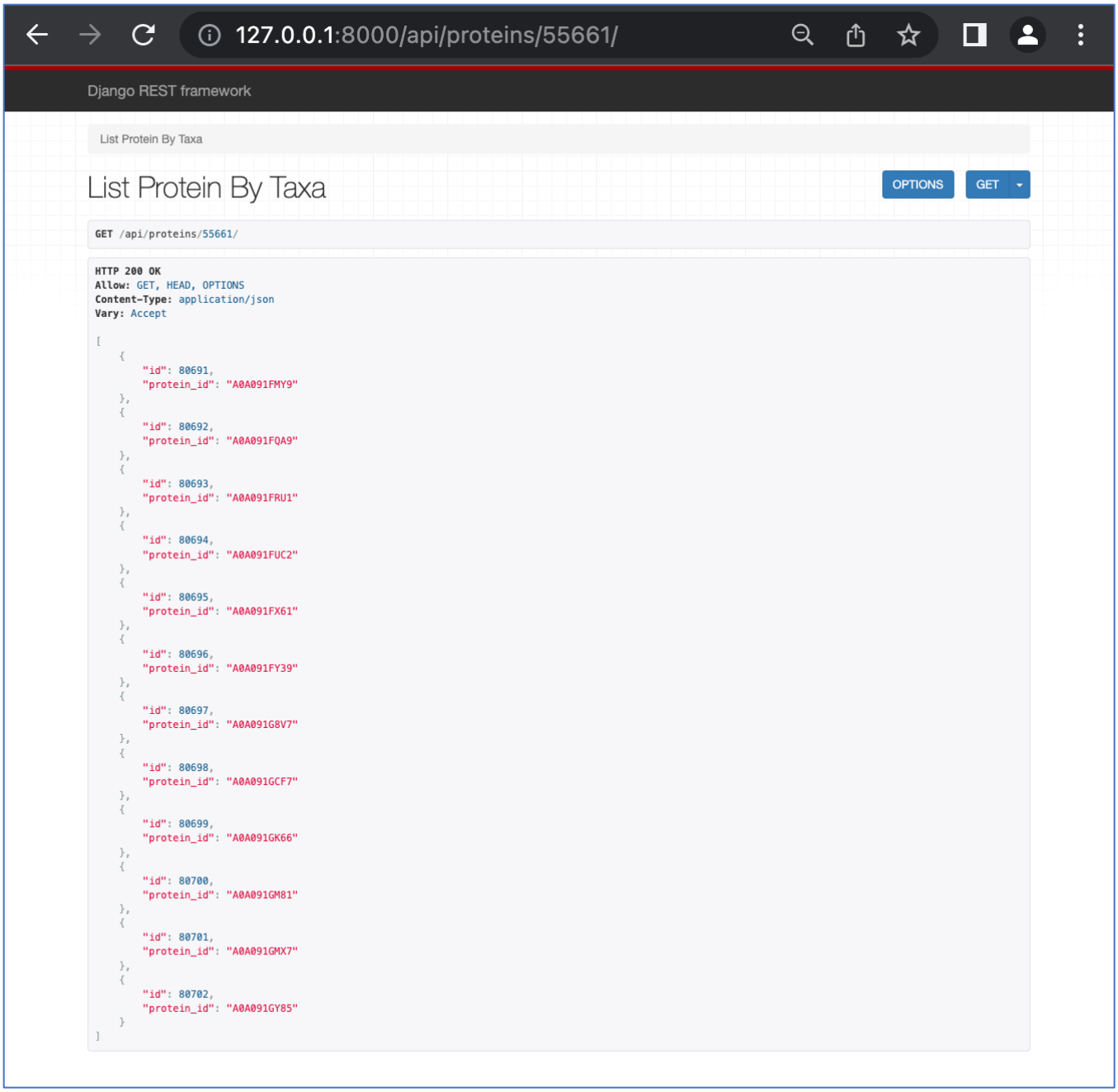


Figure 10: Pfams of Taxa

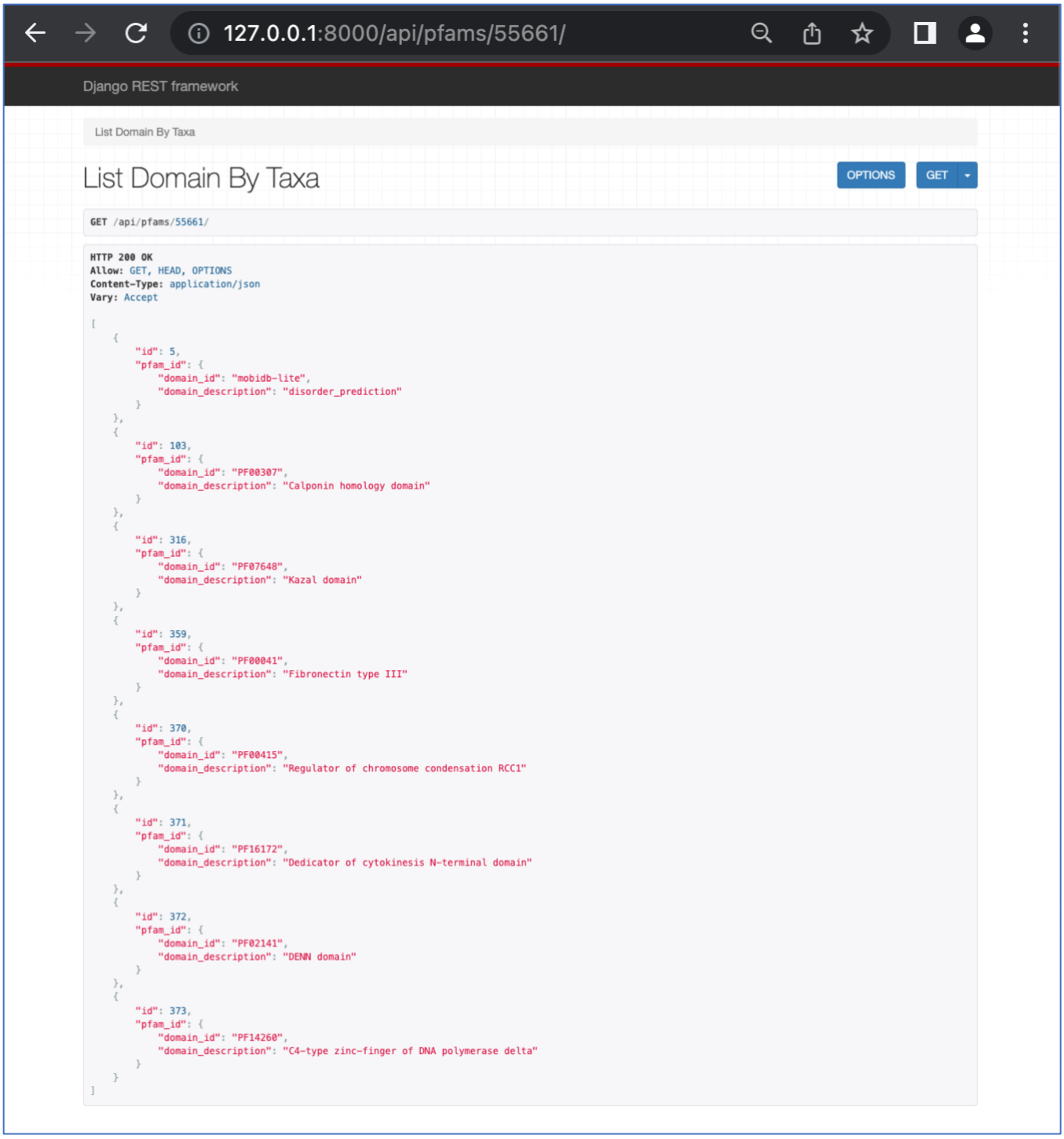
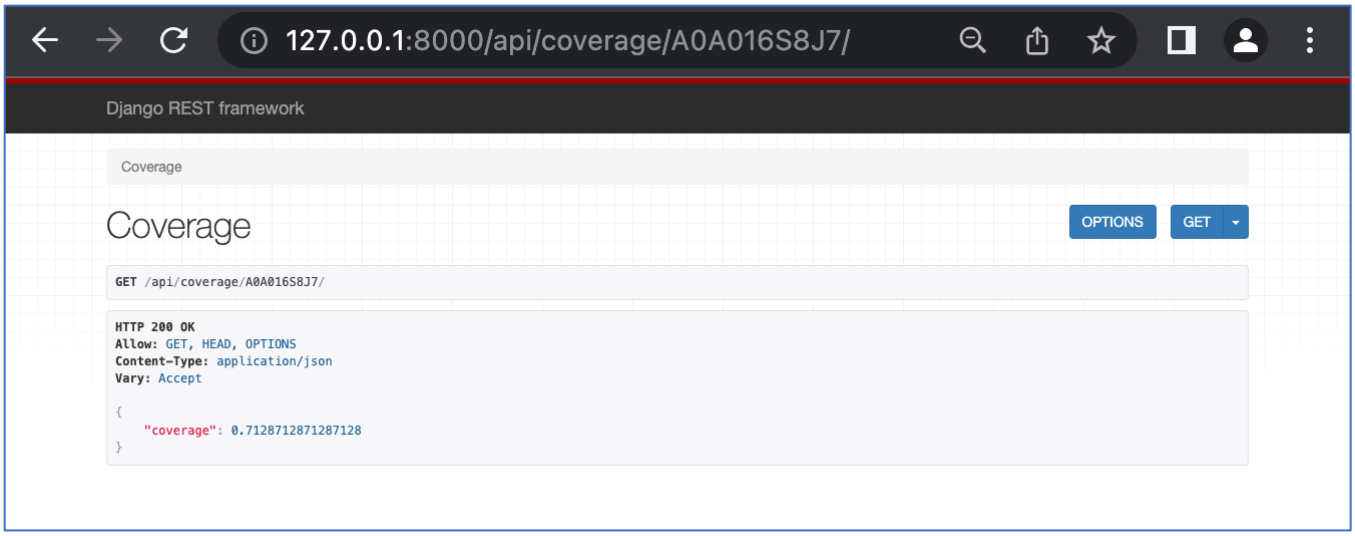


Figure 11: Coverage



Bibliography

developer.mozilla.org. (n.d.). *addEventListener*. Retrieved May 2023, from developer.mozilla.org: https://developer.mozilla.org/en-US/docs/Web/API/Document/addEventListener

developer.mozilla.org. (n.d.). *Alert*. Retrieved May 2023, from developer.mozilla.org: https://developer.mozilla.org/en-US/docs/Web/API/Window/alert

developer.mozilla.org. (n.d.). *appendChild*. Retrieved May 2023, from developer.mozilla.org: https://developer.mozilla.org/en-US/docs/Web/API/Node/appendChild

developer.mozilla.org. (n.d.). *Callback function*. Retrieved May 2023, from developer.mozilla.org: https://developer.mozilla.org/en-US/docs/Glossary/Callback_function

developer.mozilla.org. (n.d.). *classList*. Retrieved May 2023, from developer.mozilla.org: https://developer.mozilla.org/en-US/docs/Web/API/Element/classList

developer.mozilla.org. (n.d.). *Conditional Operator*. Retrieved May 2023, from developer.mozilla.org: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Conditional_Operator

developer.mozilla.org. (n.d.). *createElement*. Retrieved May 2023, from developer.mozilla.org: https://developer.mozilla.org/en-US/docs/Web/API/Document/createElement

developer.mozilla.org. (n.d.). *Django Tutorial Part 10: Testing a Django web application*. Retrieved May 2023, from developer.mozilla.org: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Testing

developer.mozilla.org. (n.d.). *DOMContentLoaded event*. Retrieved May 2023, from developer.mozilla.org: https://developer.mozilla.org/en-US/docs/Web/API/Window/DOMContentLoaded_event

developer.mozilla.org. (n.d.). *EventListener*. Retrieved May 2023, from developer.mozilla.org: https://developer.mozilla.org/en-US/docs/Web/API/EventListener

developer.mozilla.org. (n.d.). *EventTarget: addEventListener() method*. Retrieved June 2023, from developer.mozilla.org: https://developer.mozilla.org/en-US/docs/Web/API/EventTarget/addEventListener

developer.mozilla.org. (n.d.). *Fetch API*. Retrieved June 2023, from developer.mozilla.org: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API

developer.mozilla.org. (n.d.). *Fetch API*. Retrieved May 2023, from developer.mozilla.org: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API

developer.mozilla.org. (n.d.). *function*. Retrieved May 2023, from developer.mozilla.org: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/function

developer.mozilla.org. (n.d.). *getElementById*. Retrieved May 2023, from developer.mozilla.org: https://developer.mozilla.org/en-US/docs/Web/API/Document/getElementById

developer.mozilla.org. (n.d.). *Getting started*. Retrieved June 2023, from developer.mozilla.org: https://developer.mozilla.org/en-US/docs/Web/Guide/AJAX/Getting_Started

developer.mozilla.org. (n.d.). *Headers*. Retrieved May 2023, from developer.mozilla.org: https://developer.mozilla.org/en-US/docs/Web/API/Headers

developer.mozilla.org. (n.d.). *HTMLInputElement*. Retrieved May 2023, from developer.mozilla.org: https://developer.mozilla.org/en-US/docs/Web/API/HTMLInputElement

developer.mozilla.org. (n.d.). *Input*. Retrieved May 2023, from developer.mozilla.org: https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input

developer.mozilla.org. (n.d.). *Introduction to the DOM*. Retrieved June 2023, from developer.mozilla.org: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction

developer.mozilla.org. (n.d.). *isArray*. Retrieved May 2023, from developer.mozilla.org: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/isArray
developer.mozilla.org. (n.d.). *location*. Retrieved May 2023, from developer.mozilla.org: https://developer.mozilla.org/en-US/docs/Web/API/Window/location
developer.mozilla.org. (n.d.). *offsetHeight*. Retrieved May 2023, from developer.mozilla.org: https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement/offsetHeight
developer.mozilla.org. (n.d.). *Parse*. Retrieved May 2023, from developer.mozilla.org: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON/parse
developer.mozilla.org. (n.d.). *preventDefault*. Retrieved May 2023, from developer.mozilla.org: https://developer.mozilla.org/en-US/docs/Web/API/Event/preventDefault
developer.mozilla.org. (n.d.). *Promise*. Retrieved May 2023, from developer.mozilla.org: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise
developer.mozilla.org. (n.d.). *querySelector*. Retrieved May 2023, from developer.mozilla.org: https://developer.mozilla.org/en-US/docs/Web/API/Document/querySelector
developer.mozilla.org. (n.d.). *Recursion*. Retrieved May 2023, from developer.mozilla.org: https://developer.mozilla.org/en-US/docs/Glossary/Recursion
developer.mozilla.org. (n.d.). *removeChild*. Retrieved May 2023, from developer.mozilla.org: https://developer.mozilla.org/en-US/docs/Web/API/Node/removeChild
developer.mozilla.org. (n.d.). *Response*. Retrieved May 2023, from developer.mozilla.org: https://developer.mozilla.org/en-US/docs/Web/API/Response
developer.mozilla.org. (n.d.). *setAttribute*. Retrieved May 2023, from developer.mozilla.org: https://developer.mozilla.org/en-US/docs/Web/API/Element/setAttribute
developer.mozilla.org. (n.d.). *stringify*. Retrieved May 2023, from developer.mozilla.org: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON/stringify
developer.mozilla.org. (n.d.). *style*. Retrieved May 2023, from developer.mozilla.org: https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement/style
developer.mozilla.org. (n.d.). *Trim*. Retrieved May 2023, from developer.mozilla.org: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/Trim
developer.mozilla.org. (n.d.). *Using Fetch*. Retrieved May 2023, from developer.mozilla.org: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch
djangoproject.com. (n.d.). *bulk_create()*. Retrieved May 2023, from djangoproject.com: https://docs.djangoproject.com/en/3.2/ref/models/querysets/#bulk-create
djangoproject.com. (n.d.). *bulk_create()*. Retrieved May 2023, from djangoproject.com: https://docs.djangoproject.com/en/3.2/ref/models/querysets/#bulk-create
djangoproject.com. (n.d.). *Form and field validation*¶. Retrieved May 2023, from docs.djangoproject.com: https://docs.djangoproject.com/en/3.2/ref/forms/validation/
djangoproject.com. (n.d.). *Formsets*¶. Retrieved May 2023, from docs.djangoproject.com: https://docs.djangoproject.com/en/3.2/topics/forms/formsets/
djangoproject.com. (n.d.). *Raising ValidationError*. Retrieved May 2023, from docs.djangoproject.com: https://docs.djangoproject.com/en/3.2/ref/forms/validation/#raising-validationerror
djangoproject.com. (n.d.). *Test Case*. Retrieved June 2023, from docs.djangoproject.com: https://docs.djangoproject.com/en/3.2/topics/testing/tools/#testcase

djangoproject.com. (n.d.). *Testing tools*. Retrieved June 2023, from docs.djangoproject.com: https://docs.djangoproject.com/en/3.2/topics/testing/tools/#django.test.TestCase.tearDown
djangoproject.com. (n.d.). *Writing and running tests*. Retrieved 2023 June, from docs.djangoproject.com: https://docs.djangoproject.com/en/3.2/topics/testing/overview/
django-rest-framework.org. (n.d.). *API References*. Retrieved May 2023, from django-rest-framework.org: https://www.django-rest-framework.org/api-guide/views/#api-reference
django-rest-framework.org. (n.d.). *Generic API view*. Retrieved May 2023, from django-rest-framework.org: https://www.django-rest-framework.org/api-guide/generic-views/#genericapiview
django-rest-framework.org. (n.d.). *Generic Views*. Retrieved May 2023, from django-rest-framework.org: https://www.django-rest-framework.org/api-guide/generic-views/
django-rest-framework.org. (n.d.). *Limit offset pagination*. Retrieved May 2023, from django-rest-framework.org: https://www.django-rest-framework.org/api-guide/pagination/#limitoffsetpagination
django-rest-framework.org. (n.d.). *List API view*. Retrieved May 2023, from django-rest-framework.org: https://www.django-rest-framework.org/api-guide/generic-views/#listapiview
django-rest-framework.org. (n.d.). *List Create API view*. Retrieved May 2023, from django-rest-framework.org: https://www.django-rest-framework.org/api-guide/generic-views/#listcreateapiview
django-rest-framework.org. (n.d.). *Lookup Field*. Retrieved May 2023, from django-rest-framework.org: https://www.django-rest-framework.org/api-guide/generic-views/#lookup-field
django-rest-framework.org. (n.d.). *Model Serializer*. Retrieved May 2023, from django-rest-framework.org: https://www.django-rest-framework.org/api-guide/serializers/#modelserializer
django-rest-framework.org. (n.d.). *Nested relationships*. Retrieved May 2023, from django-rest-framework.org: https://www.django-rest-framework.org/api-guide/relations/#nested-relationships
django-rest-framework.org. (n.d.). *Pagination*. Retrieved May 2023, from django-rest-framework.org: https://www.django-rest-framework.org/api-guide/pagination/
django-rest-framework.org. (n.d.). *Response*. Retrieved May 2023, from django-rest-framework.org: https://www.django-rest-framework.org/api-guide/responses/#response
django-rest-framework.org. (n.d.). *Retrieve API view*. Retrieved May 2023, from django-rest-framework.org: https://www.django-rest-framework.org/api-guide/generic-views/#retrieveapiview
django-rest-framework.org. (n.d.). *Retrieve Update Destroy API view*. Retrieved May 2023, from django-rest-framework.org: https://www.django-rest-framework.org/api-guide/generic-views/#retrieveupdatedestroyapiview
django-rest-framework.org. (n.d.). *Saving instances*. Retrieved May 2023, from django-rest-framework.org: https://www.django-rest-framework.org/api-guide/serializers/#saving-instances
django-rest-framework.org. (n.d.). *Serializer class*. Retrieved May 2023, from django-rest-framework.org: https://www.django-rest-framework.org/api-guide/serializers/#serializer-class
django-rest-framework.org. (n.d.). *Serializer Method Field*. Retrieved May 2023, from django-rest-framework.org: https://www.django-rest-framework.org/api-guide/fields/#serializermethodfield
django-rest-framework.org. (n.d.). *Serializers*. Retrieved May 2023, from django-rest-framework.org: https://www.django-rest-framework.org/api-guide/serializers/
django-rest-framework.org. (n.d.). *Source*. Retrieved May 2023, from django-rest-framework.org: https://www.django-rest-framework.org/api-guide/fields/#source

django-rest-framework.org. (n.d.). *Specifying which fields to include*. Retrieved May 2023, from django-rest-framework.org: https://www.django-rest-framework.org/api-guide/serializers/#specifying-which-fields-to-include

django-rest-framework.org. (n.d.). *Validation Error*. Retrieved May 2023, from django-rest-framework.org: https://www.django-rest-framework.org/api-guide/exceptions/#validationerror

django-rest-framework.org. (n.d.). *Views*. Retrieved May 2023, from django-rest-framework.org: https://www.django-rest-framework.org/api-guide/views/

docs.djangoproject.com. (n.d.). *Create*. Retrieved May 2023, from docs.djangoproject.com: https://docs.djangoproject.com/en/3.2/ref/models/querysets/#create

docs.djangoproject.com. (n.d.). *custom template tags*. Retrieved May 2023, from docs.djangoproject.com: https://docs.djangoproject.com/en/3.2/howto/custom-template-tags/

docs.djangoproject.com. (n.d.). *Deleting objects*. Retrieved May 2023, from docs.djangoproject.com: https://docs.djangoproject.com/en/3.2/topics/db/queries/#deleting-objects

docs.djangoproject.com. (n.d.). *django core wsgi get wsgi application*. Retrieved May 2023, from docs.djangoproject.com: https://docs.djangoproject.com/en/3.2/howto/deployment/wsgi/#django.core.wsgi.get_wsgi_applicationhttps://docs.djangoproject.com/en/3.2/howto/deployment/wsgi/#django.core.wsgi.get_wsgi_application

docs.djangoproject.com. (n.d.). *django.contrib.messages.error*. Retrieved May 2023, from docs.djangoproject.com: https://docs.djangoproject.com/en/3.2/ref/contrib/messages/#django.contrib.messages.error

docs.djangoproject.com. (n.d.). *Filter*. Retrieved May 2023, from docs.djangoproject.com: https://docs.djangoproject.com/en/3.2/ref/models/querysets/#filter

docs.djangoproject.com. (n.d.). *Form is valid*. Retrieved May 2023, from docs.djangoproject.com: https://docs.djangoproject.com/en/3.2/ref/forms/api/#django.forms.Form.is_valid

docs.djangoproject.com. (n.d.). *Forms*. Retrieved May 2023, from docs.djangoproject.com: https://docs.djangoproject.com/en/3.2/topics/forms/

docs.djangoproject.com. (n.d.). *Forms*. Retrieved May 2023, from docs.djangoproject.com: https://docs.djangoproject.com/en/3.2/topics/forms/

docs.djangoproject.com. (n.d.). *get object or 404*. Retrieved May 2023, from docs.djangoproject.com: https://docs.djangoproject.com/en/3.2/topics/http/shortcuts/#get-object-or-404

docs.djangoproject.com. (n.d.). *get or create*. Retrieved May 2023, from docs.djangoproject.com: https://docs.djangoproject.com/en/3.2/ref/models/querysets/#get-or-create

docs.djangoproject.com. (n.d.). *get_or_create()*. Retrieved May 2023, from docs.djangoproject.com: https://docs.djangoproject.com/en/3.2/ref/models/querysets/#get-or-create

docs.djangoproject.com. (n.d.). *HttpRequest Method*. Retrieved May 2023, from docs.djangoproject.com: https://docs.djangoproject.com/en/3.2/ref/request-response/#django.http.HttpRequest.method

docs.djangoproject.com. (n.d.). *Inline formsets*. Retrieved May 2023, from docs.djangoproject.com: https://docs.djangoproject.com/en/3.2/topics/forms/modelforms/#inline-formsets

docs.djangoproject.com. (n.d.). *Language*. Retrieved May 2023, from docs.djangoproject.com: https://docs.djangoproject.com/en/3.2/ref/templates/language/

docs.djangoproject.com. (n.d.). *messages*. Retrieved May 2023, from docs.djangoproject.com: https://docs.djangoproject.com/en/3.2/ref/contrib/messages/

docs.djangoproject.com. (n.d.). *Messages*. Retrieved May 2023, from docs.djangoproject.com: https://docs.djangoproject.com/en/3.2/ref/contrib/messages/

docs.djangoproject.com. (n.d.). *Model Does Not Exist*. Retrieved May 2023, from docs.djangoproject.com: https://docs.djangoproject.com/en/3.2/ref/models/instances/#django.db.models.Model.DoesNotExist

docs.djangoproject.com. (n.d.). *Model forms*. Retrieved May 2023, from docs.djangoproject.com: https://docs.djangoproject.com/en/3.2/topics/forms/modelforms/

docs.djangoproject.com. (n.d.). *Model Multiple Choice Field*. Retrieved May 2023, from docs.djangoproject.com: https://docs.djangoproject.com/en/3.2/ref/forms/fields/#modelmultiplechoicefield

docs.djangoproject.com. (n.d.). *Model Save*. Retrieved May 2023, from docs.djangoproject.com: https://docs.djangoproject.com/en/3.2/ref/models/instances/#django.db.models.Model.save

docs.djangoproject.com. (n.d.). *Model Save*. Retrieved May 2023, from docs.djangoproject.com: https://docs.djangoproject.com/en/3.2/ref/models/instances/#django.db.models.Model.save

docs.djangoproject.com. (n.d.). *Models*. Retrieved May 2023, from docs.djangoproject.com: https://docs.djangoproject.com/en/3.2/topics/db/models/

docs.djangoproject.com. (n.d.). *Options*. Retrieved May 2023, from docs.djangoproject.com: https://docs.djangoproject.com/en/3.2/ref/models/options/

docs.djangoproject.com. (n.d.). *QuerySet all*. Retrieved May 2023, from docs.djangoproject.com: https://docs.djangoproject.com/en/3.2/ref/models/querysets/#django.db.models.query.QuerySet.all

docs.djangoproject.com. (n.d.). *querysets*. Retrieved May 2023, from docs.djangoproject.com: https://docs.djangoproject.com/en/4.1/ref/models/querysets/

docs.djangoproject.com. (n.d.). *querysets*. Retrieved May 2023, from docs.djangoproject.com: https://docs.djangoproject.com/en/3.2/ref/models/querysets/

docs.djangoproject.com. (n.d.). *Redirect*. Retrieved May 2023, from docs.djangoproject.com: https://docs.djangoproject.com/en/3.2/topics/http/shortcuts/#redirect

docs.djangoproject.com. (n.d.). *registering custom filters and tags*. Retrieved May 2023, from docs.djangoproject.com: https://docs.djangoproject.com/en/3.2/howto/custom-template-tags/#registering-custom-filters-and-tags

docs.djangoproject.com. (n.d.). *Retrieving objects*. Retrieved May 2023, from docs.djangoproject.com: https://docs.djangoproject.com/en/3.2/topics/db/queries/#retrieving-objects

docs.djangoproject.com. (n.d.). *Serialization*. Retrieved May 2023, from docs.djangoproject.com: https://docs.djangoproject.com/en/3.2/topics/serialization/

docs.djangoproject.com. (n.d.). *String*. Retrieved May 2023, from docs.djangoproject.com: https://docs.djangoproject.com/en/3.2/ref/models/instances/#str

docs.djangoproject.com. (n.d.). *template inheritance*. Retrieved May 2023, from docs.djangoproject.com: https://docs.djangoproject.com/en/3.2/topics/templates/#template-inheritance

docs.djangoproject.com. (n.d.). *Templateview*. Retrieved May 2023, from docs.djangoproject.com: https://docs.djangoproject.com/en/3.2/ref/class-based-views/base/#templateview

docs.djangoproject.com. (n.d.). *The save method*. Retrieved May 2023, from docs.djangoproject.com: https://docs.djangoproject.com/en/3.2/topics/forms/#the-save-method

docs.djangoproject.com. (n.d.). *Validation*. Retrieved May 2023, from docs.djangoproject.com: https://docs.djangoproject.com/en/3.2/ref/forms/validation/

docs.djangoproject.com/. (n.d.). *django.contrib.messages.success*. Retrieved May 2023, from docs.djangoproject.com/: https://docs.djangoproject.com/en/3.2/ref/contrib/messages/#django.contrib.messages.success

docs.djangoproject.com/. (n.d.). *http.require POST*. Retrieved May 2023, from docs.djangoproject.com/: https://docs.djangoproject.com/en/3.2/ref/csrf/#django.views.decorators.http.require_POST

docs.djangoproject.com/. (n.d.). *JSONresponse objects*. Retrieved May 2023, from docs.djangoproject.com/: https://docs.djangoproject.com/en/3.2/ref/request-response/#jsonresponse-objects

docs.djangoproject.com/. (n.d.). *Module django.core serializers*. Retrieved May 2023, from docs.djangoproject.com/: https://docs.djangoproject.com/en/3.2/topics/serialization/#module-django.core.serializers

docs.djangoproject.com/. (n.d.). *Relationships*. Retrieved May 2023, from docs.djangoproject.com/: https://docs.djangoproject.com/en/3.2/topics/db/models/#relationships

docs.djangoproject.com/. (n.d.). *render*. Retrieved May 2023, from docs.djangoproject.com/: https://docs.djangoproject.com/en/3.2/topics/http/shortcuts/#render

docs.djangoproject.com/. (n.d.). *simple tags*. Retrieved May 2023, from docs.djangoproject.com: https://docs.djangoproject.com/en/3.2/howto/custom-template-tags/#simple-tags

docs.python.org. (n.d.). *Binary Arithmetic Operations*. Retrieved May 2023, from docs.python.org: https://docs.python.org/3/reference/expressions.html#binary-arithmetic-operations

docs.python.org. (n.d.). *CSV*. Retrieved May 2023, from docs.python.org: https://docs.python.org/3/library/csv.html

docs.python.org. (n.d.). *CSV Reader*. Retrieved May 2023, from docs.python.org: https://docs.python.org/3/library/csv.html#csv.reader

docs.python.org. (n.d.). *Datetime*. Retrieved May 2023, from docs.python.org: https://docs.python.org/3/library/datetime.html

docs.python.org. (n.d.). *Exceptions*. Retrieved May 2023, from docs.python.org: https://docs.python.org/3/library/exceptions.html

docs.python.org. (n.d.). *JSON loads*. Retrieved May 2023, from docs.python.org: https://docs.python.org/3/library/json.html#json.loads

docs.python.org. (n.d.). *List Comprehensions*. Retrieved May 2023, from docs.python.org: https://docs.python.org/3/tutorial/datastructures.html#list-comprehensions

docs.python.org. (n.d.). *Open*. Retrieved May 2023, from docs.python.org: https://docs.python.org/3/library/functions.html#open

docs.python.org. (n.d.). *OS Enviroment*. Retrieved May 2023, from docs.python.org: https://docs.python.org/3/library/os.html#os.environ

docs.python.org. (n.d.). *OS Path*. Retrieved May 2023, from docs.python.org: https://docs.python.org/3/library/os.path.html

docs.python.org. (n.d.). *OS Path Absolute Path*. Retrieved May 2023, from docs.python.org: https://docs.python.org/3/library/os.path.html#os.path.abspath

docs.python.org. (n.d.). *OS Path Dirname*. Retrieved May 2023, from docs.python.org: https://docs.python.org/3/library/os.path.html#os.path.dirname

docs.python.org. (n.d.). *OS Path Join*. Retrieved May 2023, from docs.python.org: https://docs.python.org/3/library/os.path.html#os.path.join

docs.python.org. (n.d.). *Set*. Retrieved May 2023, from docs.python.org: https://docs.python.org/3/library/stdtypes.html#set

docs.python.org. (n.d.). *Sum*. Retrieved May 2023, from docs.python.org: https://docs.python.org/3/library/functions.html#sum

docs.python.org. (n.d.). *SYS Path*. Retrieved May 2023, from docs.python.org: https://docs.python.org/3/library/sys.html#sys.path

factoryboy. (n.d.). *Docs*. Retrieved May 2023, from factoryboy: https://factoryboy.readthedocs.io/

FactoryBoy. (n.d.). *Factory Boy*. Retrieved May 2023, from github.com: https://github.com/FactoryBoy/factory_boy

faker.readthedocs.io. (n.d.). *Docs*. Retrieved May 2023, from faker.readthedocs.io: https://faker.readthedocs.io/

favicon.io. (n.d.). *favicon.io*. Retrieved June 2023, from favicon.io: https://favicon.io/favicon-generator/

Fields. (n.d.). Retrieved May 2023, from docs.djangoproject.com: https://docs.djangoproject.com/en/3.2/ref/models/fields/

getbootstrap.com. (n.d.). *Alerts*. Retrieved May 2023, from getbootstrap.com: https://getbootstrap.com/docs/5.1/components/alerts/

https://developer.mozilla.org/en-US/docs/Web/API/HTMLLabelElement. (n.d.). *HTMLLabelElement*. Retrieved May 2023, from https://developer.mozilla.org/en-US/docs/Web/API/HTMLLabelElement: https://https://developer.mozilla.org/en-US/docs/Web/API/HTMLLabelElement/en-US/docs/Web/API/HTMLLabelElement

joke2k. (n.d.). *Faker*. Retrieved May 2023, from github.com: https://github.com/joke2k/faker

JSON response objects. (n.d.). Retrieved 2023 May, from https://docs.djangoproject.com/en/3.2/ref/request-response/#jsonresponse-objects

lucidchart.com. (n.d.). *Database Structure and Design Tutorial*. Retrieved May 2023, from lucidchart.com: https://www.lucidchart.com/pages/database-diagram/database-design

Malan, D. (n.d.). *Getting Started with Amazon EC2*. Retrieved June 2023, from harvard.edu: https://cs.harvard.edu/malan/publications/ec2.pdf

Module, U. o. (2023). rest_specification_and_examples.txt. *rest_specification_and_examples.txt*.

Note, A. (2022). *How to Connect to an EC2 Instance from your Mac Computer (For Mac Users Only)*. Retrieved June 2023, from youtube.com: https://www.youtube.com/watch?v=fsjs8XTi8JI

openapis.org. (n.d.). *OpenAPI Initiative Registry*. Retrieved May 2023, from openapis.org: https://spec.openapis.org

Peterson, R. (2023, May 6). Retrieved 2023 May, from guru99.com: https://www.guru99.com/er-diagram-tutorial-dbms.html

python.org. (n.d.). *tearDown()*. Retrieved June 2023, from docs.python.org: https://docs.python.org/3/library/unittest.html#unittest.TestCase.tearDown

redocly.com. (n.d.). *Getting started with Redocly*. Retrieved May 2023, from redocly.com: https://redocly.com/docs/

Request Response. (n.d.). Retrieved May 2023, from docs.djangoproject.com: https://docs.djangoproject.com/en/3.2/ref/request-response/

Song, J. D. (2021, May 2026). *Enhancing data science environments with Vim, tmux, and Zsh on Amazon EC2*. Retrieved June 2023, from aws.amazon.com: https://aws.amazon.com/blogs/opensource/enhancing-data-science-environments-with-vim-tmux-and-zsh-on-amazon-ec2/

sqlite.org. (n.d.). *Appropriate Uses For SQLite*. Retrieved May 2023, from sqlite.org: https://www.sqlite.org/whentouse.html

sqlite.org. (n.d.). *Limits In SQLite*. Retrieved May 2023, from sqlite.org: https://www.sqlite.org/limits.html#max_variable_number

stanford.edu. (n.d.). *Databases: Modeling and Theory*. Retrieved May 2023, from stanford.edu/: https://online.stanford.edu/courses/soe-ydatabases0003-databases-modeling-and-theory

swagger.io. (n.d.). *Documentation*. Retrieved May 2023, from swagger.io: https://swagger.io/docs/

www.programink.com. (n.d.). *How To Deploy Django Web Application*. Retrieved June 2023, from www.programink.com: https://www.programink.com/django-tutorial/django-deployment.html