Explanation for the Task

1. Hierarchy of the Project:

In the folder data_platform1, Application folder is an app under data_platform.
data_platform contains the following files:

1. __init__.py
2. Asgi.py
3. Settings.py - set up the database, installed apps.
4. Urls.py - included the urls.py file from application for individual urls.
5. Wsgi.py

application contains .py files like:

1. Resource folder - Directory for saving the uploaded database files.
2. Models.py - **Created 2 database models called Dataset and Resource**.
3. Urls.py - Set up urls for every API created in the app.
4. Views.py - Defined API for tasks mentioned in the readme file in github.
5. Query.py - Code for the defined APIs in views.py file.
6. Serializers.py - Used serializers from rest_framework for uploading dataset files.
7. __init__.py
8. Admin.py
9. Apps.py
10. Tests.py

2. Explanation of application:

First off, I've defined API to

- upload the dataset and resource and save metadata in db and resource file in file system
- list all datasets/ show a dataset detail given its name/
- api to fetch the resource data from db using resource id

Urls.py:

- **path("allDataset/", views.all_dataset),**
  # to get the details of all the Datasets present in the database.

- **path("uploadDataset/", views.UploadFileView.as_view(), name='upload_dataset')**,
  # to upload a dataset

- **path("fetchDatasetDetail/<str:dataset_name>", views.fetch_dataset_detail)**,
  # to get the metadata of a particular dataset

- **path("getResource/<str:dataset_name>", views.get_resource_data)**
  # to download the resource files associated with a particular dataset

2.1 Upload a Dataset:

For uploading a dataset, I've used Serializers in Django framework for this task. I've created a file called **serializers.py** file in the same directory which includes a custom class '**FileUploadSerializer**' which has dataset_name, file and description as attributes.

An object of this class is created in '**UploadFileView'** class in views.py and a function '**post**' is defined under the same class. After serializing the atributes, the post function is used for saving the instances and upload the data to database using the '**upload_dataset**' function from **query.py.** In **upload_dataset** function, it takes the inputs which were dataset_name, resource_file and description.
From the problem statement it is understood that a **single dataset may have multiple Resource files** for which I've created two variables count and res_count.
Count takes the count of the number of records in the dataset table with a given dataset_name, **res_count** takes the out of the number of records in the resource table with a given dataset_name.
So, when we are uploading a resource file with a new dataset_name, the **count** value will be zero and it goes through a 'if' loop where in the entries to the dataset table and resource table are done and the resource file is stored in the file system with the count value in its naming.

When uploading a resource file with an existing dataset_name, the **count** has the value of 1 with which it goes through the 'else' loop in which the **res_count** already 1 as there is already a resource file with the same dataset_name, there is no change in the dataset table, the resource table is updated with the name, location and dataset_name and the resource file is saved in the filesystem with the res_count value in its naming.

A function called **make_csv** in query.py is used for the writing of resource files and naming of these files too. It renames the uploaded csv file with a format like '**dataset_name+(count/ res_count)+_resource-file.csv**'

2.2 Viewing names of Datasets:

The function **all_dataset** from views.py is used for viewing the details of what datasets are present in the dataset table. Another function also called **all_dataset** from query.py takes all the entries from the dataset table from the database using raw SQL queries and the function all_dataset from views.py makes a list of it and prints them.

2.3 Viewing metadata of a particular Dataset:

The function **fetch_dataset_detail** from views.py is used for viewing the metadata of a particular dataset given its dataset_name. In this, the function **fetch_dataset_detail** from **query.py** is called where in the details of the dataset are extracted from the dataset table from the database using raw SQL queries.

2.4 Downloading the Resource Files:

The function **get_resource_data** from views.py is used for getting the resource files associated with a particular dataset given its dataset_name. In this function, a function called **getfiles** from query.py is called where in the locations of the csv files saved in the filesystem associated with a particular dataset are extracted from the Resource table from the Database.

After making a list of all the locations of the csv files for a particular dataset, the files are made into zip file with the name same as that of the dataset and is downloaded.