RoWaterAPI, an open-source solution for accessing and managing hydrology data

Mihnea Cristian Popa[1], Daniel Constantin Diaconu[2], Emilia Avram[1,3]

Abstract: Hydrological measurements have been performed in various ways, both manually and automatically, especially in recent years. The role of hydrological measurements is a vital one in issuing hydrological forecasts but also in alerting the authorities and the population in case of potential devastating floods. The frequency of these measurements plays a crucial role in issuing these forecasts, so the need for near real-time measurement solutions is a genuine concern. The research proposes developing a RESTful API solution that provides access to historical and near-real-time water flow information, built using the Django/GeoDjango web framework and PostgreSQL, an open-source object-relational database system. The solution is intended to be proposed and implemented within the "Romanian Waters" National Administration. The API will provide the four basic operations of persistent storage, create, read, update, and delete (CRUD), which will allow the end-user to query the database and the RoWater staff to manage and update the data.

Keywords: API, hydrological measurements, open-source, water resource management

1. Introduction

The rapid advancement of technology has made it possible to develop and implement large-scale applications and systems previously closed-loop into an open system that can be easily shared. The development of API (Application Programming Interface) services has allowed both private and public entities to achieve this goal of information sharing, both in the web and mobile environment. This study proposes the development of an API product for the National Administration "Romanian Waters" to facilitate the sharing of hydrological information and forecasts. The idea of this project is one of pure initiative, noting the need to digitize the exchange of information within this administration and the need for such a product to develop an agricultural land management application, an application proposed in my doctoral thesis. The development of such a product thus leads to an increase in the digitalization of services and information provided by the Romanian national authorities, a sector that is still underdeveloped and requires serious investments.

The development of an API product, available in JSON and XML format with near real-time hydrological data, is an essential step in the application and the development of future applications for forecasting and monitoring the hydrological situation of rivers in Romania. This product can be used by other national institutions that have a role in

[1] PhD Candidate, "Simion Mehedinți-Nature and Sustainable Development" Doctoral School, University of Bucharest, 010041 Bucharest, Romania.
[2] Associate Professor, PhD, Faculty of Geography, University of Bucharest, Bucharest, Romania.
[3] PhD Candidate, "Simion Mehedinți-Nature and Sustainable Development" Doctoral School, University of Bucharest, 010041 Bucharest, Romania; National Institute of Hydrology and Water Management, 97E Sos. Bucuresti-Ploiesti, 1st District, 013686 Bucharest, Romania.

alerting the population in emergencies (DSU, ISU, etc.). Following the analysis of similar products, we have already been able to identify similar solutions proposed both in Romania and internationally. In Romania, such a product was developed by the National Meteorological Administration (ANM) and proposed an API product available in JSON and XML formats. The API provides weather information, nowcasting warnings, general warnings, and city weather forecasts. Internationally, similar solutions have been developed by the United Kingdom (Hydrology Data API), Norway (HydAPI), Finland (Hydrologiarajapinta), and the State of Texas in the United States (Water Data for Texas). The solutions provide information about the hydrological situation at the hydrometric station level and other hydrological parameters, and the results of the queries are presented in both JSON and GeoJSON format.

The proposed solution will have two environments, one for ordinary users where they will be able to return the available data, having access to historical and near real-time data for the Romanian hydrographic network, and for the lakes where there is information available. Users will be able to access this information through sixteen endpoints. The second environment is one for RoWater internal employees, an environment that also has a front-end component where they can manage the data in the database that communicates with the application. They will only be able to access this environment through a user that can only be created for the institution's email addresses. Access will be based on an access *token* valid for 90 minutes.

The objective of this research and solution is to contribute to a better understanding of these types of solutions and their importance and facilitate the internal processes of RoWater for both internal and external transmission of hydrological measurements and warnings.

2. Research Methodology

The term API is the application programming interface (Application Programming Interface), which is a set of development rules and protocols by which software can interact (access and use) with resources (data) provided by programs (RedHat).

A REST API product (also known as the RESTful API) is also an application programming interface that tracks the constraints of the REST architectural style and allows interaction with RESTful web services. The term REST was proposed by the American computer scientist Roy Fielding and represented Representational State Transfer (Dorasamy, 2021).

The development of this API product proposes the use of technologies such as the Django and GeoDjango web framework to develop the back-end component and the Vue.js framework for the development of the front-end component. The development of this product will have two components, the back-end and the front end. The front-end component will allow you to enter data manually using a form to enter them and in an automatic way by uploading a .csv file that contains information about the measurements collected from the station. The database used by the solution is a PostgreSQL one to which the PostGIS library was added, a library that allows the use of geospatial data in this type of database.

Currently, the solution development process is developing the graphical interface and writing tests to ensure the proper functioning of the solution. The development of the solution was done using the integrated development environment PyCharm Professional 2021.3.3, developed by the Czech company JetBrains.

The database consists of 26 tables, tables generated by both the created models, and tables generated automatically after connecting the database with the Django and GeoDjango framework. An advantage of using Django is that when migrating the models created to the database, they create a series of tables containing information about the models that have been migrated and their migration history.

The API structure consists of a basic project in which several applications are installed. In our case, the basic project is called RoWater, while the applications that are part of the project are:

- API;
- Common;
- Hidro;
- Organizație;
- Unități_administrative.

Models, that is, those Python classes (Figure 1), are the blueprints for creating tables in the database. Creating templates is done by creating classes that inherit the Model class from the django.contrib.gis.db package in the GeoDjango framework, and these must contain objects that represent the columns that will be created in the database and their types.

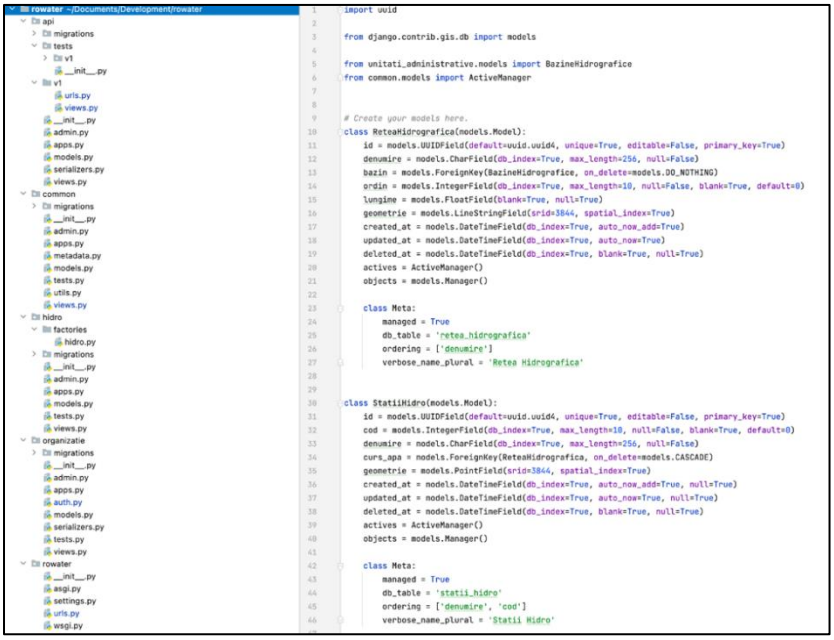

Figure 1 Example of a Python model

The resulting tables are both relational and independent. A relational table links to another table based on a Foreign Key; for example, in the case of river basin administration and river basin tables, the link between the basin table and the administrations is made through the administration_id column, a UUID column representing the unique identifier of a basin administration (Figure 2).

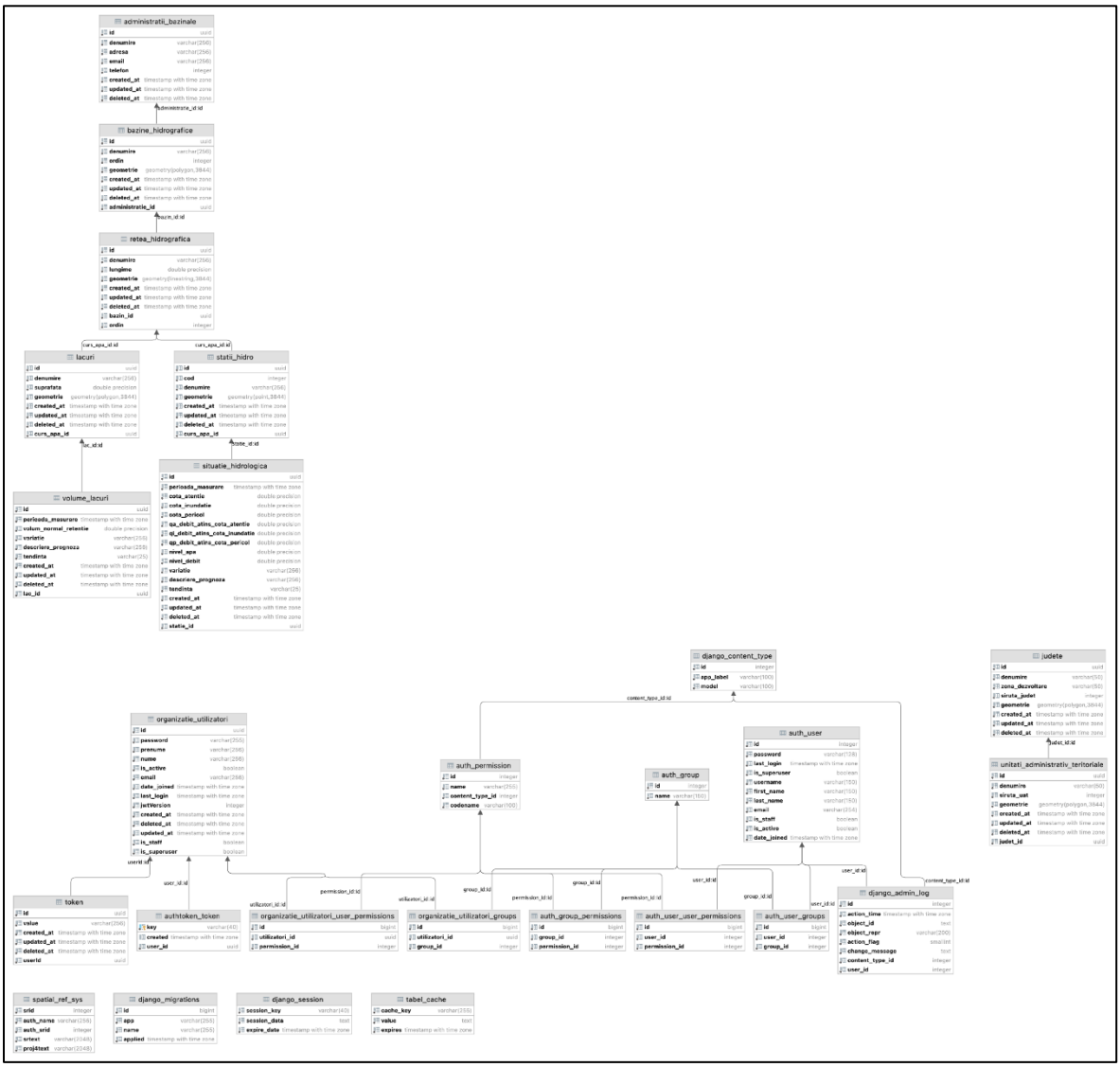

Figure 2 Database structure

Once the models are made and migrated to the database, the next step is to define Views, being a Python class that has the role of controllers in the architectural style MVC (Model-View-Controller), i.e., allows interaction with the database and definition HTTP request methods that can be used for that view. The solution has twenty-three endpoints and uses the HTTP verbs GET (data return), POST (data insertion), PUT (update data), and DELETE (data deletion) (Figure 3).

```python
class StatiiHidrologiceDupaDenumireStatieView(CachedView):
    @extend_schema(
        responses={200: StatiiHidrologiceSerializer},
        parameters=[
            OpenApiParameter(name='Informatii stații hidrologice',
                             description='Returnare stații hidrologice după denumire',
                             required=False, type=str),
        ],
    )
    def get(self, requests, denumire_statie):
        statie = StatiiHidro
        if statie.objects.filter(denumire=denumire_statie, deleted_at__isnull=True).exists():
            statie = StatiiHidro.objects.filter(denumire=denumire_statie, deleted_at__isnull=True).first()
        else:
            return Response({'mesaj': f'Nu au fost găsite date pentru selecția dvs.!'}, status=400)
        ret = {
            'meta': display_metadata(self),
            'data': StatiiHidrologiceSerializer(statie, many=False).data
        }
        return Response(ret)
```

Figure 3 Example of GET method

Once the views have been defined, for users to use them and perform database operations, their path must be defined in the URLs class (Figure 4).

```python
from rest_framework.routers import DefaultRouter

from django.urls import path, include

from api import v1
from api.v1.views import StatiiHidrologiceView

router = DefaultRouter()
urlpatterns = [
    path('', include(router.urls)),
    path('retea_hidro/', v1.views.ReteaHidroView.as_view(), name='retea-hidro'),
    path('retea_hidro/<str:denumire>', v1.views.ReteaHidroDupaDenumireView.as_view(), name='retea-hidro-dupa-denumire'),
    path('retea_hidro/bazin/<str:denumire>', v1.views.ReteaHidroDupaDenumireBazinHidrograficView.as_view(), name='retea-hidro-dupa-denumire-bazin'),
    path('retea_hidro/uat/<int:siruta>', v1.views.ReteaHidroDupaSirutaUATView.as_view(), name='retea-hidro-dupa-siruta-uat'),
    path('retea_hidro/judet/<str:denumire>', v1.views.ReteaHidroDupaDenumireJudetView.as_view(), name='retea-hidro-dupa-denumire-judet'),


    path('statii_hidrologice/', v1.views.StatiiHidrologiceView.as_view(), name='statii-hidrologice'),
    path('statii_hidrologice/<str:denumire_statie>', v1.views.StatiiHidrologiceDupaDenumireStatieView.as_view(), name='statii-hidrologice-dupa-denumire-statie'),

    path('situatie_hidro/<str:denumire_statie>', v1.views.SituatieHidrologicaDupaDenumireStatieHidroView.as_view(), name='situatie-hidro-dupa-denumire-statie'),
    path('situatie_hidro/perioada/<str:denumire_statie>', v1.views.SituatieHidrologicaDupaPerioadaMasurareDupaDenumireStatieView.as_view(), name='situatie-hidro-dupa-perioada-masurare-dupa-denumire-statie'),
    path('situatie_hidro/', v1.views.AdaugareSituatieHidrologicaView.as_view(), name='adaugare-situatie-hidro'),
    path('situatie_hidro/delete/<str:pk>', v1.views.StergereSituatieHidrologicaDupaPkView.as_view(), name='stergere-situatie-hidro'),
    path('situatie_hidro/edit/<str:pk>', v1.views.EditareSituatieHidrologicaDupaPkView.as_view(), name='editare-situatie-hidro'),

    path('lacuri/', v1.views.LacuriView.as_view(), name='lacuri'),
    path('lacuri/<str:denumire_lac>', v1.views.LacuriDupaDenumireLacView.as_view(), name='lacuri-dupa-denumire-lac'),
    path('lacuri_curs/<str:denumire_curs_apa>', v1.views.LacuriDupaDenumireCursApaView.as_view(), name='lacuri-dupa-denumire-curs-apa'),

    path('volume_lacuri/<str:denumire_lac>', v1.views.VolumeLacuriDupaDenumireLacView.as_view(), name='volume-lacuri-dupa-denumire-lac'),
    path('volume_lacuri/perioada/<str:denumire_lac>', v1.views.VolumeLacuriDupaPerioadaMasurareDupaDenumireLacView.as_view(), name='volume-lacuri-dupa-perioada-masurare-dupa-denumire-lac'),
    path('volume_lacuri/', v1.views.AdaugareVolumeLacuriView.as_view(), name='adaugare-volume-lacuri'),
    path('volume_lacuri/delete/<str:pk>', v1.views.StergereVolumLacDupaPkView.as_view(), name='stergere-volum-lac'),
    path('volume_lacuri/edit/<str:pk>', v1.views.EditareVolumLacDupaPkView.as_view(), name='editare-volum-lac'),

    path('inregistrare/', v1.views.InregistrareUtilizatorView.as_view(), name='inregistrare-utilizator'),
    path('login/', v1.views.LoginView.as_view(), name='login-utilizator'),
    path('utilizator/<str:tokenValue>', v1.views.DetaliiUtilizatorDupaIDTokenView.as_view(), name='utilizator'),
]
```

Figure 4 Example of Urls Python class

Once the access paths to the views have been defined, by accessing them either using Postman, the browser, or by using Swagger (the platform used to create the documentation), users will be able to return the requested data (Figure 5).

```
{
    "meta": {
        "status": 200,
        "autor": "Administrația Națională Apele Române",
        "licenta": "https://data.gov.ro/base/images/logoinst/OGL-ROU-1.0.pdf",
        "tip_licenta": "OGL-ROU 1.0",
        "versiune": "1.0"
    },
    "data": [
        {
            "bazin": "Olt",
            "ordin_rau": "1",
            "denumire": "Olt",
            "lungime": 140.55,
            "geometrie": {
                "type": "LineString",
                "coordinates": [
                    [
                        -71.160281,
                        42.258729
                    ],
                    [
                        -71.160837,
                        42.259113
                    ],
                    [
                        -71.161144,
                        42.25932
                    ]
                ]
            }
        }
    ]
}
```


Figure 5 Example of result

The API will be able to return data on the hydrological status and lake volumes at all hydrometric stations in the country or station level, as well as values recorded at a specific date, using query parameters that accept the reference year and/or month. . The solution will be able to return the geometries of the hydrological network, the lakes but also the locations of the hydrometric stations and the endpoints capable of these things will have the Caching functionality, i.e., once called they will be stored in temporary memory to be accessed quickly once they are called again (Developer Mozilla – HTTP caching).

To ensure a good quality of the application, Rollbar will be integrated. Rollbar allows monitoring errors produced in both the back-end and front-end environment. The following steps in developing the solution are to complete the front-end component, test, and refactor the application following the API development guidelines proposed by the German eCommerce platform Zalando (https://opensource.zalando.com/restful-api-guidelines/), being a landmark of guidelines in the development of API solutions. The last step is to implement the application on a production server.

3. Conclusion

The importance and benefits of such a solution are numerous. The data provided through the API can be used for the development of new GIS solutions, such as web maps with the hydrological status of rivers that can be integrated into the system of services and products offered by RoWater but also in population prevention applications, to example the RoAlert application, which can provide warnings in case of a potential hydrological hazard.

The development of such a solution could also be a step in developing similar solutions by other government agencies or the improvement/updating of existing solutions by adding new features or integrating them with other related solutions.

Implementing such a solution would lead to the modernization and facilitation of the exchange of information internally in RoWater and the exchange of information with the population or other local or national institutions.

References

Dorasamy, R. API Development. 2021. *API Development, API Marketplace Engineering*, Apress, Berkeley, CA, 173-198

Preibisch, S. 2018. *API Development: A Practical Guide for Business Implementation Success, API Development,* Apress, Berkeley, CA

Developer Mozilla – HTTP https://developer.mozilla.org/en-US/docs/Web/HTTP/Caching

RedHat - https://www.redhat.com/en/topics/api/what-are-application-programming-interfaces / https://www.redhat.com/en/topics/api/what-is-a-rest-api