Open Water Journal – Volume 7, Issue 1, Article 3

Design and Development of a Tethys Framework Web Application to Elucidate the HydroShare.org Application Programmer Interface

Research Article

Abhishek Amalaraj[1], Daniel P. Ames[2]

[1] *Civil and Environmental Engineering, Brigham Young University, Provo, Utah 84602, abhishekamal18@gmail.com*
[2] *Civil and Environmental Engineering, Brigham Young University, Provo, Utah 84602, dan.ames@byu.edu*

Abstract

In recent years, data and file sharing have advanced significantly, opening doors for engineers from all over the world to stay connected with each other and share data, models, scripts and other information required for scientific and engineering purposes. HydroShare (www.hydroshare.org) was developed by a consortium of universities sponsored by the National Science Foundation (NSF) as a means for improving data and model sharing. Originally released in 2014, and continually updated since that time, HydroShare has proven to be a valuable resource for a growing number of active users in the field of water resources and environmental research. The graphical user interface is relatively simple and easy to understand and the system provides users with a large amount of free data storage, which makes it particularly useful for academics, researchers, and scientists as well as practicing engineers. This project report presents the design and development of a web-based application (web app) that demonstrates all core functions of HydroShare via a published application programmer interface (API). The resulting web app was developed using the Tethys Platform which is intended for creating web-based applications with database and mapping capabilities. This app demonstrates the use of all of the core functions of the HydroShare Python REST client and includes sample code and instructions for using these functions. The overarching goal of this work is to increase the use and usability of HydroShare via its API and to simplify using the API for student and other programmers developing their own web applications.

Keywords: Hydrology, Education, Web development, Tethys Platform, HydroShare

1.0 Introduction

Web based applications, web services, and online data and model sharing technologies are becoming increasingly available to support hydrologic research. This promises benefits in terms of collaboration, computer platform independence, and reproducibility of modeling workflows and results (Gan et al. 2020). New advances in cyberinfrastructure and semantic mediation technologies have provided the means for creating better tools supporting data discovery and access (Ames et al. 2012). The Consortium of Universities for the Advancement of Hydrologic Science Inc. (CUAHSI) hydrological information system (HIS) is a widely used platform that is service oriented to manage time series data (Sadler, Ames, and Livingston 2016). HydroShare is a web based hydrologic information system of CUAHSI and that was developed to allow users to share and publish data and models in a variety of flexible formats, and to make this information available in a citable, shareable and discoverable manner. The CUAHSI HIS is an internet-based system to support the sharing of hydrologic data. It is comprised of hydrologic databases and servers connected through web services as well as software for data publication, discovery and access. The CUAHSI Observations Data Model (ODM) provides community defined semantics needed to allow sharing of hydrologic information (Horsburgh et al. 2008). HydroShare enables users to collaborate and work as teams in a web based collaborative environment, thereby enhancing research, education and application of hydrologic knowledge. It includes tools (web apps) that can act on content in HydroShare providing users with a gateway to computing and analysis. Much like Google Drive, the capability to have cloud based applications that act on its data is a key part of HydroShare that advances its capability within general trend towards providing web-based software services (Crawley et al. 2017). HydroShare also includes support for hydrologic models and model input/output files which is expected to facilitate further use by water resources engineers and managers who rely more and more on hydrologic models (Roberts et al. 2018).

One example of a web-based application that uses HydroShare is HydroShare GIS. It functions by accessing the spatial metadata contained within the HydroShare resource data model and overlaying datasets as layers within the OpenLayers JavaScript library which is a web-mapping client library for rendering interactive maps on a web page (Hazzard 2011). Data are passed from the app's server to a GeoServer data server and shared as web mapping service layers. Thus, users can easily build map projects from data sources registered in HydroShare and save them back to HydroShare as map project resources, which can both be shared with others and re-opened in HydroShare GIS (Crawley et al. 2017). Shown below in Figure 1 is the HydroShare website home page.

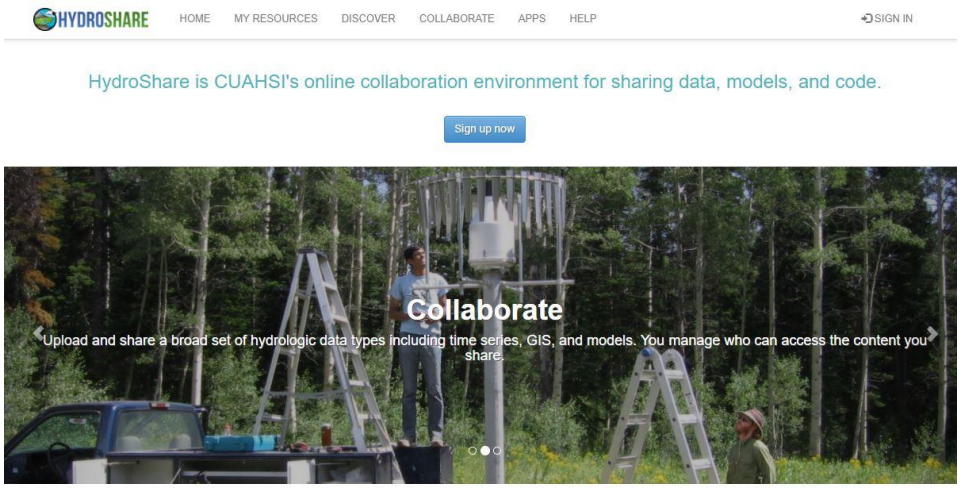

Figure 1 HydroShare website homepage.

HydroShare has a number of stated and intended uses as well as extended uses that researchers and water resources engineers may find in the future. Stated uses of HydroShare include:

- Share hydrologic data and models with research colleagues.
- Manage who has access to the content that is shared.
- Share, access, visualize and manipulate a broad set of hydrologic data types and models.
- Use the web services application programmer interface (API) to program automated and client access.
- Publish data and models to meet the requirements of a data management plan.
- Discover and access data and models published by others.
- Use web apps to visualize, analyze and run models on data in HydroShare.

Water resources data scientists often require access to cloud-based data storage systems and end user models, tools, and data analysis applications. In a service that is organized as a framework, interaction between end user tools and the online data storage system is facilitated with the help of APIs (Ong et al. 2015). Upload and download of large water related GIS data and hydrologic databases can cause a web application to perform poorly, however when this information can be accessed and provided in smaller chunks, as needed, and processed for specific purposes – i.e. through the use of API's -these limitations can be avoided (Michaelis and Ames 2012). HydroShare is a data and model sharing system and it comprises a systematic approach to file sharing using a distributed file sharing back end called IRODs (Rajasekar et al. 2010). Data stored within the IRODS system is primarily accessed through the HydroShare front-end web portal. However, there is also a built in capability to access HydroShare resources and data using an API built on Representational State Transfer (REST) (Khare and Taylor 2004). This API has been extended through a Python wrapper module called hs-restclient. The REST interface and the hs-restclient Python module have both been used by developers creating web applications for HydroShare but there is a lack of detailed demonstration code for this functionality, which limits use of these powerful means of interacting with HydroShare. According to Crawley et al. (2017) "The good news is that much advancement and innovation has been achieved in the field of spatial data cloud computing in recent years that could contribute to and greatly simplify the development of a cloudbased application for interacting with HydroShare's spatial data". The specific problem in HydroShare is that we have no tutorial features with the API that can be useful with a beginner or any intermediate level programmer to incorporate the found code in the API. The option to understand how it falls into place of any web app or website to use the same feature found in the HydroShare API is not found in it. Hence, the universality or opportunity to expand for the HydroShare API which being a forerunner in storing hydrologic data is restricted and also the resources and the option to store water and scientific data cannot be used at their best potential. CUAHSI provides JUPYTER notebooks that are notebooks to run Python code which help with running a block of code in the API but they are not self-explanatory. In a growing world of environmental engineers and water scientists, there is also a rise in developing technologies that make use of the web in a complete sense and having a cloud of such apps focused on helping this line of fellow scholars is one of many aspirations that we hope to achieve and will help with establishing it in the not so distant future. So, an app to provide some reference and education on this specific task will be greatly beneficial. Previously we have discussed that CUAHSI has Jupyter notebooks to help with the purpose of understanding the API and the blocks of code. These notebooks to practice script and functions are very smooth and fast. The idea behind making this active feature is to fill the gap and sort of form a bridge between using the features on the HydroShare website and learning how they work. But it does not go the extra mile in explaining step by step how each piece of the script helps with accomplishing different specific tasks found on the website.

"Sharing hydrological data across geographical boundaries can help alleviate damage from floods. Water rights can be managed on a need basis instead of greed, thus preventing droughts and dry spells in downstream regions" (Khattar and Ames 2020). The idea of the project began keeping in mind the beginner and intermediate level programmer and the limited but unique challenge they face at understanding how to implement the REST API of

one of the best environmental science and engineering-based data storage and modelling systems. With the rise in free services and literature online, one can use them to its full capacity but can also turn out to be a herculean task to achieve a simple function with minimum documentation. The research project was funded by HydroShare. Thorough research was done on the idea of implementing the REST API in a Tethys framework and enabling all the functions. The project is focused at filling the gap that medium level programmers come across with understanding APIs and database management.

The project began in May 2020 and was completed over the course of a few months. The completed app as intended has all the demonstrations needed and a few improvisations which are successful and smooth. All the features in the web app were tried and tested and they are also provided with the step-by-step instruction.

1.1 REST Architecture

REST is an architecture style that can be used for designing networked applications (Wei Zhou et al. 2014). As "REST architectural style has gained more popularity in implementing loosely coupled systems, RESTful services are becoming the style of choice for northbound API and gaining increasingly importance in Software Defined Networking (SDN) architecture. Adopting REST for a SDN architecture has the following benefits:

1. Decentralized management of dynamic resources: REST does not use any centralized resource registry but relies on connections between resources to discover and manage them. REST allows network elements, such as routers, switches, middle boxes (e.g. NAT and DPI devices), to be dynamically deployed and changed in a distributed fashion.
2. Heterogeneous clients: because REST separates resource representations, identification, and interaction, it can adjust resource representations and network protocols based on SDN client capabilities and network conditions to optimize API performance.
3. Service composition: the current trend in SDN is to use programming composition to achieve functional flexibility, such as REST can provide service-oriented compositions that are independent of programming languages and can run on different platforms.
4. Localized migration: since the functions of SDN are fast evolving, the northbound APIs of SDN controllers will likely change accordingly. REST API supports backward-compatible service migration through localized migration by which a newly added resource only affects the resources that connect to it. Combined with uniform interface and hypertext-driven service discovery, it can ease the tension between the new service deployments and backward compatibility.
5. Scalability: REST achieves server scalability by keeping the server stateless and improves server performance through layered caches. This feature will become useful, when an SDN controller needs to support many concurrent host-based applications and to use network resources in an efficient way." (W. Zhou et al. 2014).

1.2 Application Programming Interfaces (APIs)

An API is a computing technique which establishes communication between many software intercessors. API can be completely customized, discretely made for an element in the app, or it can be planned out for an organization to ensure integrative usefulness. It sets the types of calls, requests or queries that can be made, their nature, the data types that have to be used and the agreements that should follow. These interfaces can also be prepared to furnish certain special mechanisms that offer capabilities in several ways and formats (Ofoeda, Boateng, and Effah 2019). In software applications, an API simplifies programming by abstracting the underlying implementation and only exposing objects or actions the developer needs. While a graphical interface for an email client might provide a user with a button that performs all the steps for fetching and highlighting new emails, an

API for file input/output might give the developer a function that copies a file from one location to another without requiring that the developer understand the file system operations occurring behind the scenes.

To understand the way an API works is to visualize a middle person capable of transferring communication between two separate parties and also provides the service as part of that communication. For example, as shown in Figure 2, imagine giving an order to a waiter in a drive through restaurant. He takes in the order over the microphone and then passes the information of your order to the kitchen where the food is processed. The kitchen processes the food and gives it to the person who took the order who then eventually delivers it. This sort of service sees that communication is performed effectively to get the end product delivered.

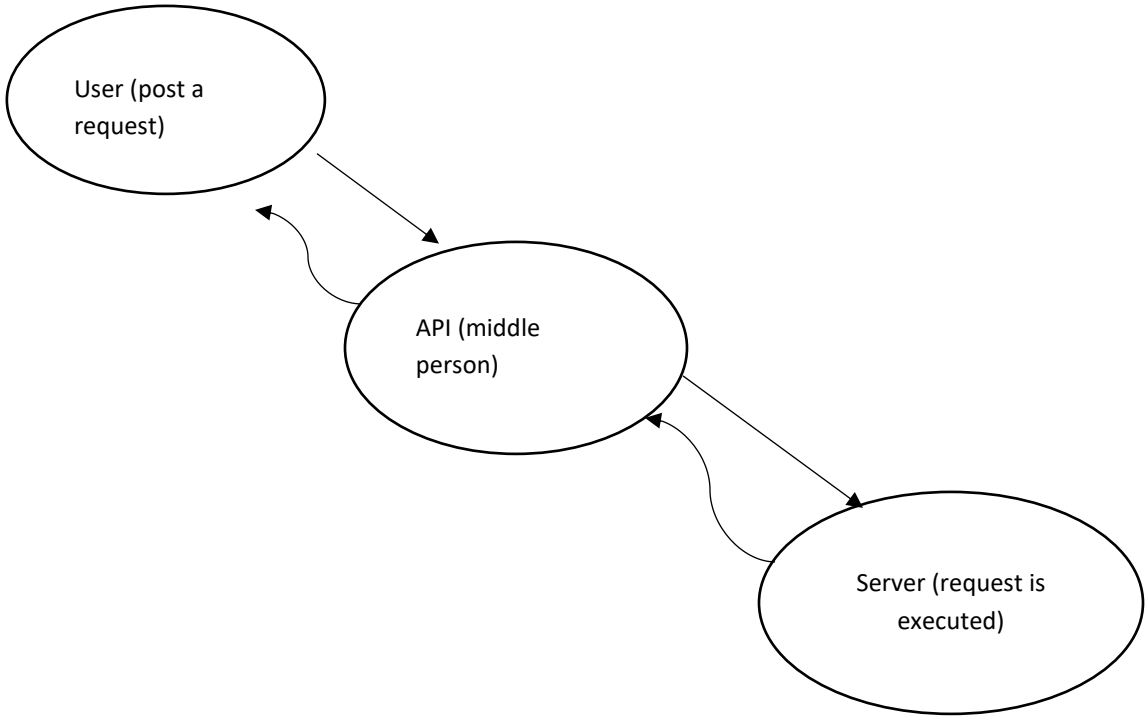

Figure 2 Understanding how an API works

An API works similar to the example described earlier and helps with transferring the command from the user to the server and the server then executes the command and transfers the end product or the query result to the user through the API which acts as the middleman. HydroShare provides a Python based API that can be used in our Tethys app as we use a Django framework for creating and developing an app. This project will help with using all the functions on the API provided to make the app as useful as possible and import the major functions in the HydroShare page. The Python API provided has good support and is updated annually. That way, the features of HydroShare not supported by the API now could have it added to the documentation sometime later in the future.

1.3 Tethys Platform

Programming can be a difficult skill to adapt (Gries 2012) and would require creative thinking and problem solving to accomplish simple to advanced steps. Web programming involves the use of understanding frameworks and how the elements in different file formats fall into place and depend on each other. “The interactive nature of

web applications or 'web apps' makes them a well-suited medium for conveying complex scientific concepts to lay audiences and creating decision support tools that harness cutting edge modeling techniques and promote the work of environmental scientists and engineers. Despite this potential, the technical expertise required to develop web apps represents a formidable barrier—even for scientists and engineers who are skilled programmers" (Swain et al. 2016). Tethys is a platform that makes web development a lot easier by supporting APIs from various sources and software packages and enabling to incorporate them very effectively into our apps. The platform synthesizes several free and open source software projects that are needed for the development of cloud-based hydrologic modeling applications and includes software that offers typical web development tools in the form of a web framework as well as web-based geographic information systems (GIS), high performance computing management utilities and interactive scientific visualization libraries (Jones et al. 2014).

Tethys Platform is updated periodically and is open source. The software package is written in such a way that even beginner and intermediate level programmers with low to minimal programming skills can catch up and build a fully functioning web app that does all that they intend to do with it. Tethys Platform provides a suite of free and open source software. Included in the Software Suite is PostgreSQL with the PostGIS extension - which is a spatial database add-on including support for all of advanced spatial processing and querying entirely at the SQL command-line (Ramsey and Columbia 2005), GeoServer for spatial data publishing (Youngblood 2013). Tethys also provides Gizmos for inserting OpenLayers (a dynamic web interactive service) (Farkas 2016) and Google Maps for interactive spatial data visualizations in web apps. The Software Suite also includes HTCondor for managing distributed computing resources and scheduling computing jobs (Fajardo et al. 2015).

Building a Tethys app is much easier than building a web app from scratch. The dependencies and the framework are set in stone right from the beginning. Changes can be made to the core structure and styling of the app with very few and simple steps(Swain et al. 2016). Tethys Platform requires the 'Conda' packaging system. Conda standardizes software installations across various language ecosystems by describing each specific software with a human readable recipe that defines meta-information and dependencies, and also the simple 'build script' that performs the tasks necessary to build and install the software (Grüning et al. 2018) and is an open source package management system that runs on Windows, macOS and Linux. Conda quickly installs, runs and updates packages and their dependencies. Tethys Platform includes a modern web portal built on Django (Forcier, Bissex, and Chun 2008) that is used to host web apps called Tethys Portal. It provides the core website functionality that is often taken for granted in modern web applications including a user account system with a password reset mechanism for forgotten passwords (Holovaty and Kaplan-Moss 2009).

The software development kit takes advantage of the Django template system to help with building dynamic pages for web apps while writing less HTML. It also provides a series of modular user interface elements called Gizmos. With only a few lines of code you can add range sliders, toggle switches, auto completes, interactive maps, and dynamic plots to your web app. The platform also includes Python modules that allow you to provision and run computing jobs in distributed computing environments. With CondorPy you can define your computing jobs and submit them to distributed computing environments provided by HTCondor. This technology has been used to build the app and makes full use of its sustainable functionality

The unique value of Tethys is that it has some shortcuts for setting up the PSQL databases you need in a Django app, for setting up the user management database, and also providing the admin interface. They are very useful especially to set up a production web app server. To understand it better, it is basically Django. The above-mentioned features, from Tethys actually do not. The other 'Tethys' features are really other Django extensions or other software which there are docker containers for. Web apps add huge value to research projects and can be useful tools for scientists, engineers or other users.

The app can be enhanced by incorporating the hs_restclient API in it. The REST client will be inserted in the controllers.py file of the framework created while creating the app. This Python file can run all the major functions and is responsible for the running and executing of the different features enabled with the app. A few dependent

JavaScript files will be added to the app to help with simple tasks like giving prompts and pop ups as well to execute certain complex controller option which would require the need of combining more than one API function in the REST client. Subsequently the app will also need updating in the app.py file to make sure that the function mentioned in the html document can be read from the controllers.py file. This would account to a major chunk in completing the app.

The source code is available for feedback and further improvement and is uploaded to the BYU Hydroinformatics group found in GitHub. As mentioned earlier, the API is updated from time to time which could be taken to advantage and the app can be improved and updated. The app will be hosted on the Brigham Young University (BYU) Tethys portal (https://tethys.byu.edu/apps) and is ready to install from another Tethys app called 'Warehouse' – a storehouse for similar Tethys web apps.

2.0 Design and Development of a HydroShare API Tethys App

The app is powered by the framework that Tethys provides and so it is provided with a base.html file that displays all the features found on the page. The features of the app are ordered neatly on the left pane of the page individually which makes it easier to fully understand how the functions in the REST API come into play. If one or more functions are used in the same page of the app, it could be a little difficult for the beginning user to follow. The HydroShare API Tethys app is made with an introduction to HydroShare and what exactly we will be trying to accomplish in terms of understanding how the API provided by HydroShare works. For that reason, there is a tutorial page that also talks about how to get this app running. This app is made keeping in mind that it is a tutorial app and so it is built with chunks of code inserted into each of the pages to display the function that the page will be trying to incorporate in the app.

2.1 Elements of the app

a) CSS File - The CSS file dedicated to making visible and static UI changes is named as the main.css file and we can modify the logo and other similar things like having a background color, image and changes to the font and other similar details. This Tethys app will be closely connected to the HydroShare website and so it is styled very similar to the website.

b) JavaScript - The role that JavaScript plays in this app is that it helps with a lot of simple basic UI features like displaying a pop up or showing a loading status for when the app fetches information. It also helps a lot to perform certain complicated functions like joining more than function in the API to perform a specific task which will be shown later in the document.

c) HTML Templates - The HTML templates are predefined as part of the package and they help with making the app display various types of information and are stitched to the base.html file which makes the pages accessible from there. This action is done by extending each page from the base.html file using a HTML function.

d) Controllers.py - As mentioned earlier in this paper, the app is powered by Tethys which is a modified and a robust Django framework which uses Python to assign tasks and implement functions. The powerhouse for the framework would be the 'controllers.py' file. Here, we have all the API functions from REST assigned to their individual pages and the gizmos assigned to do certain specific functions.

c) Template Gizmos API - Template Gizmos are building blocks that can be used to create beautiful interactive controls for web apps. Using the Template Gizmos API, developers can add date-pickers, plots, and maps to their app pages with minimal coding. These blocks are very efficient and cleans up the code by having them do an

assigned function from the controllers.py file without having to repeat them over and over again in the HTML templates.

2.2 Arrangement of the controllers.py File

The HydroShare API Tethys app is Django based and uses Python as its programming language. The app framework is designed to be of highest convenience, and it comes with a 'controllers.py' file. The file is the powerhouse of the app as all the functions that are run in the web app are entered in this file and are connected to the 'url_maps' in the app.py file. These functions are then made available to the template from which it is used on the front end. Overall, this file is what makes the app to use the REST API in its functions to import the functionality of the HydroShare web page.

a) Packages - The controllers.py file needs to be stacked with the packages that are needed for the app to run smoothly and effectively. These packages are listed at the top of the file as Python requires indentation in a specific format and shown below are all the packages we have used for the app.

```
from django.shortcuts import render
from tethys_sdk.permissions import login_required
from tethys_sdk.gizmos import Button
from tethys_sdk.gizmos import TextInput, DatePicker, SelectInput
from tethys_sdk.gizmos import DataTableView
from tethys_services.backends.hs_restclient_helper import get_oauth_hs
from django.shortcuts import redirect, reverse
from django.contrib import messages
from django.http import HttpResponse, JsonResponse
from hs_restclient import HydroShare, HydroShareAuthBasic
from django.utils.encoding import smart_str
from wsgiref.util import FileWrapper
import os
import tempfile
import zipfile
import json
from django.core import serializers
```

b) Classes - The controllers.py file is Python based and so its organized into classes and each class is defined with functions which use the API and helps with the execution of the function. The different functions are used by the HTML templates when it is connected to the url_maps in the app.py file. For an example, the class and function shown below is for adding a file to a resource in HydroShare and it could be used to add multiple files. From the structure of it we see a controller for the homepage.

```
@login_required()
def add_file(request):
    """
    Controller for the Add Dam page.
    """
    # Default Values
    username = ''
    password = ''
    resourcein = ''

    # Errors
```



```
username_error = ''
password_error = ''
resourcein_error = ''
loggedin = False
try:
        # pass in request object
    hs = get_oauth_hs(request)
    loggedin = True

except Exception as e:
    pass
    # handle exceptions

# Handle form submission
if request.POST and 'add-button' in request.POST:
    # Get values
    has_errors = False
    username = request.POST.get('username', None)
    password = request.POST.get('password', None)
    resourcein = request.POST.get('resourcein', None)
    print(dict(request.FILES))
    uploaded_file = request.FILES['addfile']

    with tempfile.TemporaryDirectory() as temp_dir:
        temp_zip_path = os.path.join(temp_dir, uploaded_file.name)
        print(temp_zip_path)

        # Use with statements to ensure opened files are closed when done
        with open(temp_zip_path, 'wb') as temp_zip:
            for chunk in uploaded_file.chunks():
                temp_zip.write(chunk)

        # Validate
        try:
        # pass in request object
            hs = get_oauth_hs(request)

            # your logic goes here. For example: list all HydroShare resources
            # for resource in hs.getResourceList():
            #     print(resource)

        except Exception as e:
        # handle exceptions
            if not username:
                has_errors = True
                username_error = 'Username is required.'

            elif not password:
                has_errors = True
                password_error = 'Password is required.'

            else:
                auth = HydroShareAuthBasic(username= username, password= password)
                hs = HydroShare(auth=auth)

        if not resourcein:
```



```
            has_errors = True
            resourcein_error = 'Resource is required.'

        if not has_errors:
            fpath = temp_zip_path
            resource_id = hs.addResourceFile(resourcein, fpath)
            messages.success(request, "File added successfully")
        if has_errors:
            messages.error(request, "Please fix errors.")

# Define form gizmos
resourcein_input = TextInput(
    display_text='Resource ID',
    name='resourcein',
    placeholder='Enter id here eg: 08c6e88adaa647cd9bb28e5d619178e0 '
)

username_input = TextInput(
    display_text='Username',
    name='username',
    placeholder='Enter your username'
)

password_input = TextInput(
    display_text='Password',
    name='password',
    attributes={"type":"password"},
    placeholder='Enter your password'
)

add_button = Button(
    display_text='Add',
    name='add-button',
    icon='glyphicon glyphicon-plus',
    style='success',
    attributes={'form': 'add-dam-form'},
    submit=True
)

cancel_button = Button(
    display_text='Cancel',
    name='cancel-button',
    href=reverse('hydroshare_python:home')
)

context = {
    'loggedin' : loggedin,
    'resourcein_input': resourcein_input,
    'username_input': username_input,
    'password_input': password_input,
    'add_button': add_button,
    'cancel_button': cancel_button,
}

return render(request, 'hydroshare_python/add_file.html', context)
```


c) App.py File - The url_maps methods is tightly related to the App Base Class API. The url_maps methods is tightly related to the App Base Class API. The app.py file has a url_maps function that has the duty to map or locate the functions in the controllers.py file and utilize them in on their respective urls. UrlMap objects must be created from a UrlMap class that is bound to the root_url of the app. Use the url_map_maker() function to create the bound UrlMap class. If an app is generated from the scaffold, this is done automatically. Starting in Tethys 3.0, the WebSocket protocol is supported along with the HTTP protocol. To create a WebSocket UrlMap, follow the same pattern used for the HTTP protocol. The URL should be typed with precision and could lead to typo errors or indent errors. A proper way to avoid that is to maintain the same name throughout the project for a function and its execution in the web app.

```
def url_maps(self):
        """
        Add controllers
        """
        UrlMap = url_map_maker(self.root_url)

        url_maps = (
            UrlMap(
                name='home',
                url='hydroshare-python',
                controller='hydroshare_python.controllers.home'
            ),
            UrlMap(
                name='get_file',
                url='hydroshare-python/get_file',
                controller='hydroshare_python.controllers.get_file'
            )
```

3.0 App Implementation

The HydroShare API Tethys app is by nature a well-documented tutorial app that is self-explanatory to the point of working the functions on the app and has a very simple GUI. Following is a discussion of the development of the code and how we made and connected all these elements with the API to bring it onto the front-end surface. The primary functions of our HydroShare API demonstration web app are as follows:

- Create a Resource
- Browse for a Resource
- View the boundaries of a Resource
- GeoServer - View a Resource
- Metadata
- Get science Metadata
- Add a File
- Download a File
- Delete a File
- Delete a Resource
- Download a Resource
- Change a Resource from Private to Public
- Create a Folder
- Delete a Folder

The simpler the process in executing a function the easier it is to grasp the concept of it. With that regard, the best function of the app that we can demonstrate is the 'Create Folder' function. The 'Create Folder' feature on the page looks like the image shown below. The source code for the Create Folder function follows.

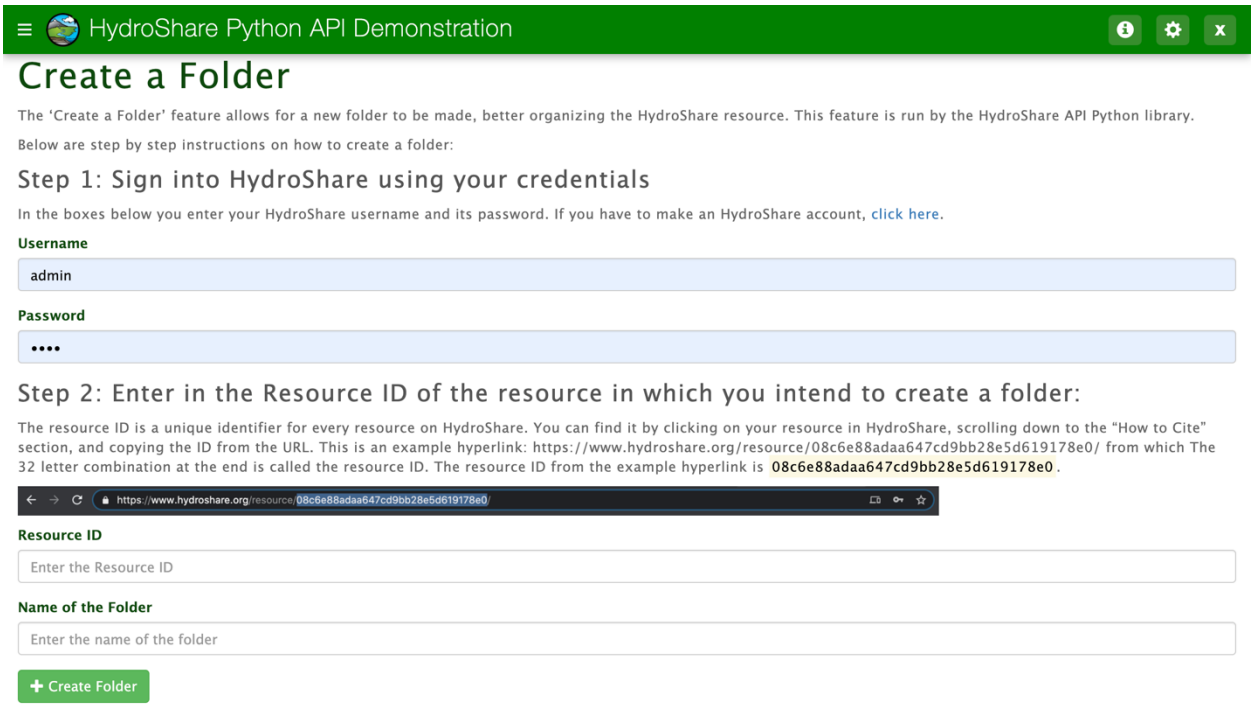

Figure 3 'Create Folder' User Interface

```
@login_required()
def create_folder(request):
    """
    Controller for the Add Dam page.
    """
    # Default Values
    username = ''
    password = ''
    # river = ''
    resourcein = ''
    foldername = ''


    # Errors
    username_error = ''
    password_error = ''
    resourcein_error = ''
    foldername_error = ''
    loggedin = False
    try:
            # pass in request object
        hs = get_oauth_hs(request)
        loggedin = True
```



```
except Exception as e:
    pass

# Handle form submission
if request.POST and 'create-button' in request.POST:
    # Get values
    has_errors = False
    username = request.POST.get('username', None)
    password = request.POST.get('password', None)
    resourcein = request.POST.get('resourcein', None)
    foldername = request.POST.get('foldername', None)


    # Validate
    if not resourcein:
        has_errors = True
        resourcein_error = 'resourcein is required.'

    try:
        # pass in request object
        hs = get_oauth_hs(request)

    # your logic goes here. For example: list all HydroShare resources

    except Exception as e:
# handle exceptions

        if not username:
            has_errors = True
            username_error = 'Username is required.'

        elif not password:
            has_errors = True
            password_error = 'Password is required.'

        else:
            auth = HydroShareAuthBasic(username=username, password=password)
            hs = HydroShare(auth=auth)


    if not foldername:
        has_errors = True
        foldername_error = 'Folder name is required.'

    if not has_errors:
        folder_to_create = foldername
        response_json = hs.createResourceFolder(resourcein, folder_to_create)
        messages.success(request, "Folder created successfully")
    if has_errors:
        messages.error(request, "Please fix errors.")

# Define form gizmos
username_input = TextInput(
    display_text='Username',
    name='username',
```



```
        placeholder='Enter your username'
    )

    password_input = TextInput(
        display_text='Password',
        name='password',
        attributes={"type":"password"},
        placeholder='Enter your password'
    )


    foldername_input = TextInput(
        display_text='Name of the Folder',
        name='foldername',
        placeholder='Enter the name of the folder'
    )

    resourcein_input = TextInput(
        display_text='Resource ID',
        name='resourcein',
        placeholder='Enter the Resource ID'
    )

    create_button = Button(
        display_text='Create Folder',
        name='create-button',
        icon='glyphicon glyphicon-plus',
        style='success',
        attributes={'form': 'add-dam-form'},
        submit=True
    )

    cancel_button = Button(
        display_text='Cancel',
        name='cancel-button',
        href=reverse('hydroshare_python:home')
    )

    context = {
        'loggedin': loggedin,
        'username_input': username_input,
        'password_input': password_input,
        'resourcein_input': resourcein_input,
        'create_button': create_button,
        'cancel_button': cancel_button,
        'foldername_input': foldername_input,
    }
   return render(request, 'hydroshare_python/create_folder.html', context)
```

The add file feature is found on the HydroShare website which as the name suggests, enables the end user to create a folder inside a resource that you own. The feature proves to be very useful as it helps with organizing the resource very effectively. The first step in the process of creating a function is creating default values which are basically arrays that store the values that are manually entered by the user.

a) The values necessary for this function are the username, password, resource ID and the name of the folder that we want to create.

```
# Default Values
username = ''
password = ''
resourcein = ''
foldername = ''
```

b) Error defaults – Just as above, here we are setting a set of defaults for the errors and create errors for the functions in case there are any. There is also a try object that is used for signing into HydroShare.

```
# Errors
username_error = ''
password_error = ''
# river_error = ''
resourcein_error = ''
foldername_error = ''
loggedin = False
try:
# pass in request object
hs = get_oauth_hs(request)
loggedin = True

except Exception as e:
pass
```

c) Handle form submission – The requests and posts are handled here where information is received and then sent for further processing when we hit the 'create' button which in other cases are buttons named after their specific functions.

```
# Handle form submission
if request.POST and 'create-button' in request.POST:
# Get values
has_errors = False
username = request.POST.get('username', None)
password = request.POST.get('password', None)
resourcein = request.POST.get('resourcein', None)
foldername = request.POST.get('foldername', None)
```

d) Validating – We further go on to validate the requests to see if they have a value and if they do, we fill them up in the array that we create for default values and if there is an invalid value like null or invalid type, then we enter it in the error defaults and proceed. Each condition is checked to be complete and ready for the submission and the task to be performed is entered in the 'if not has_errors' part where we use the API of HydroShare which is as shown below:

```
auth = HydroShareAuthBasic(username='myusername', password='mypassword')
hs = HydroShare(auth=auth)
folder_to_create = "folder_1/folder_2"
response_json = hs.createResourceFolder('ID OF RESOURCE', folder_to_create)
```


The code is modified a little as the order in which the elements are placed depends on our specific individual app as follows:

```
 # Validate
     if not resourcein:
         has_errors = True
         resourcein_error = 'resourcein is required.'

     try:
         # pass in request object
         hs = get_oauth_hs(request)

     # your logic goes here. For example: list all HydroShare resources

     except Exception as e:
 # handle exceptions

   if not username:
has_errors = True
 username_error = 'Username is required.'

 elif not password:
     has_errors = True
     password_error = 'Password is required.'

    else:
     auth = HydroShareAuthBasic(username= username, password= password)
             hs = HydroShare(auth=auth)

     if not foldername:
         has_errors = True
         foldername_error = 'Folder name is required.'

     if not has_errors:
         folder_to_create = foldername
         response_json = hs.createResourceFolder(resourcein, folder_to_create)
         messages.success(request, "Folder created successfully")
     if has_errors:
         messages.error(request, "Please fix errors.")
```

The messages.success and the messages.error help with displaying the messages on execution of the function. The ‘if not has_errors’ is the block that implements the API at the request of the create button.

a. The ‘gizmos’ – The gizmos are building blocks that accomplish simple tasks in Tethys apps for certain assigned functions. Though limited to their predefined nature, these blocks cover a wide range of useful functions. These blocks have to be activated from the controllers.py using their special gizmo tag.

```
# Define form gizmos
    username_input = TextInput(
        display_text='Username',
        name='username',
        placeholder='Enter your username'
```



```
    )

    password_input = TextInput(
        display_text='Password',
        name='password',
        attributes={"type":"password"},
        placeholder='Enter your password'
    )


    foldername_input = TextInput(
        display_text='Name of the Folder',
        name='foldername',
        placeholder='Enter the name of the folder'
    )

    resourcein_input = TextInput(
        display_text='Resource ID',
        name='resourcein',
        placeholder='Enter the Resource ID'
    )

    create_button = Button(
        display_text='Create Folder',
        name='create-button',
        icon='glyphicon glyphicon-plus',
        style='success',
        attributes={'form': 'add-dam-form'},
        submit=True
    )

    cancel_button = Button(
        display_text='Cancel',
        name='cancel-button',
        href=reverse('hydroshare_python:home')
)
```

Context – The context tag transmits the functionality of all the objects and variables to the template when the tag is placed there in the html file. It can be understood as importing it there.

```
context = {
        'loggedin' : loggedin,
        'username_input': username_input,
        'password_input': password_input,
        'resourcein_input': resourcein_input,
        'create_button': create_button,
        'cancel_button': cancel_button,
        'foldername_input': foldername_input,
    }
```

The gizmo tags like the username and password are shown below:


```
  <p>In the boxes below you enter your HydroShare username and its password. If you hav
e to make an HydroShare account, <a href="https://www.hydroshare.org/sign-
up/?next=">click here</a>.</p>
  {% if not loggedin %}
{% gizmo username_input %}
```


3.1 Graphical User Interface

Shown below is the 'Create a Resource' feature with the different stages of its execution. The screenshots are numbered to avoid confusion and to maintain the order of flow.

a) The first stage of the 'Create Resource' feature where we enter the information that is needed to create a resource in database and avoid recurring repetitions.

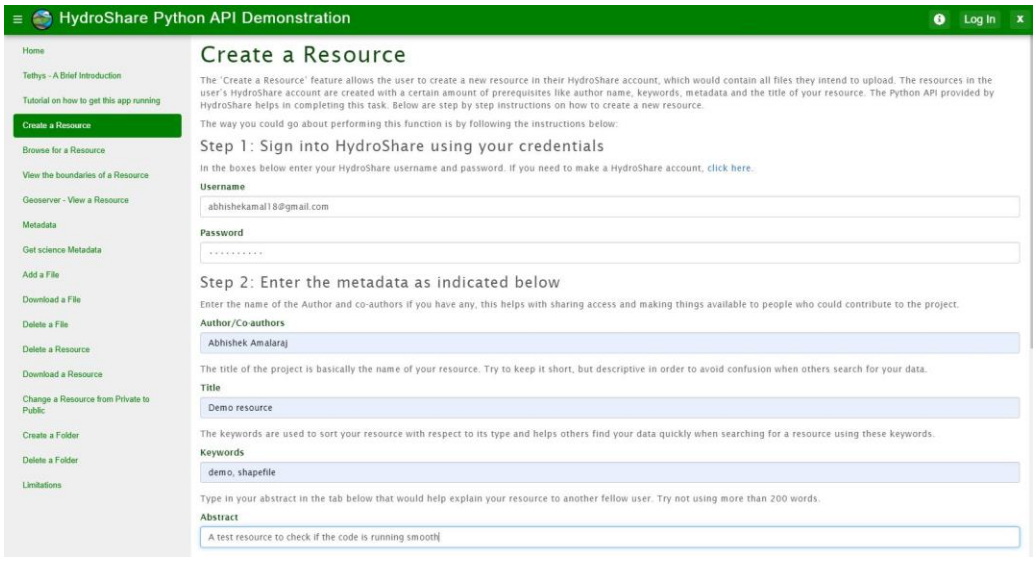

Figure 4 'Create Resource' User Interface

b) A feature has been added to the app that enables the user to add a file to the fresh resource that is being created.

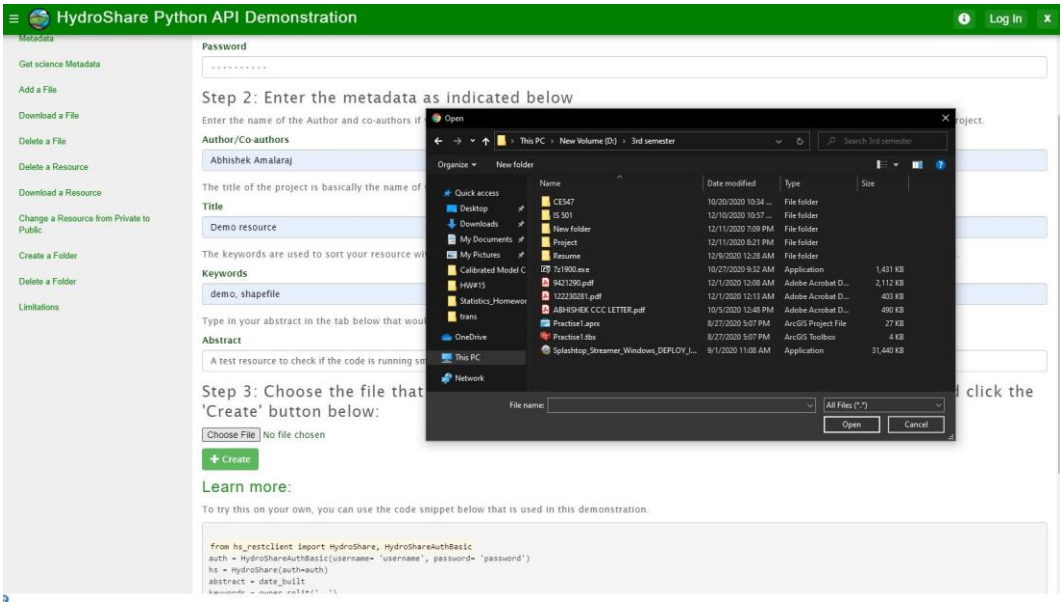

Figure 5 Adding a file to a resource

c) The ‘Create’ button executes the code present in the function.

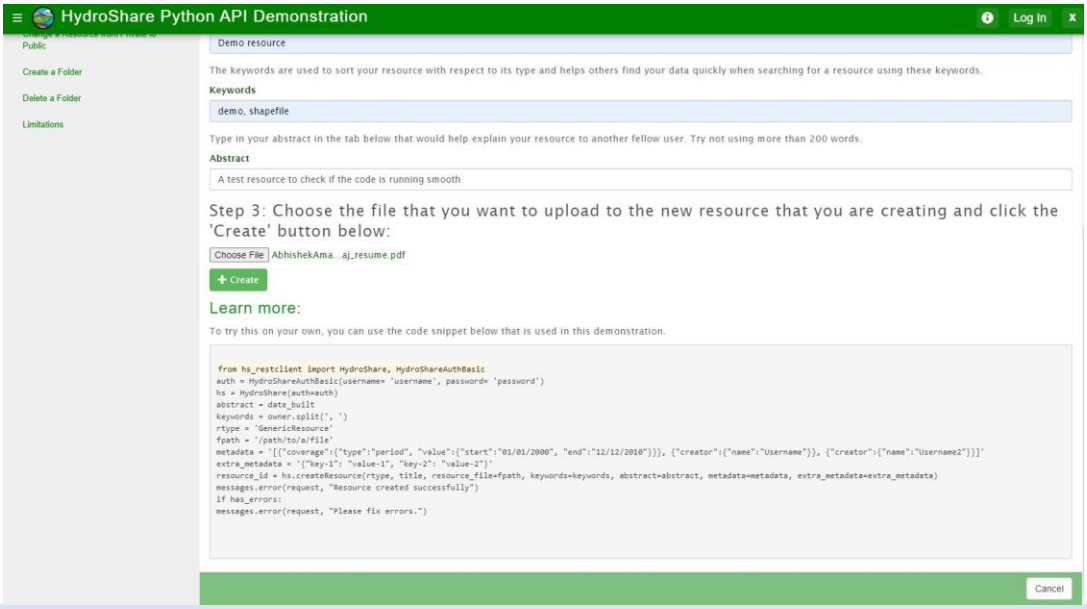

Figure 6 Emphasis on the 'Create' button

d) The pop-up message that asks us if we want to finalize our function.

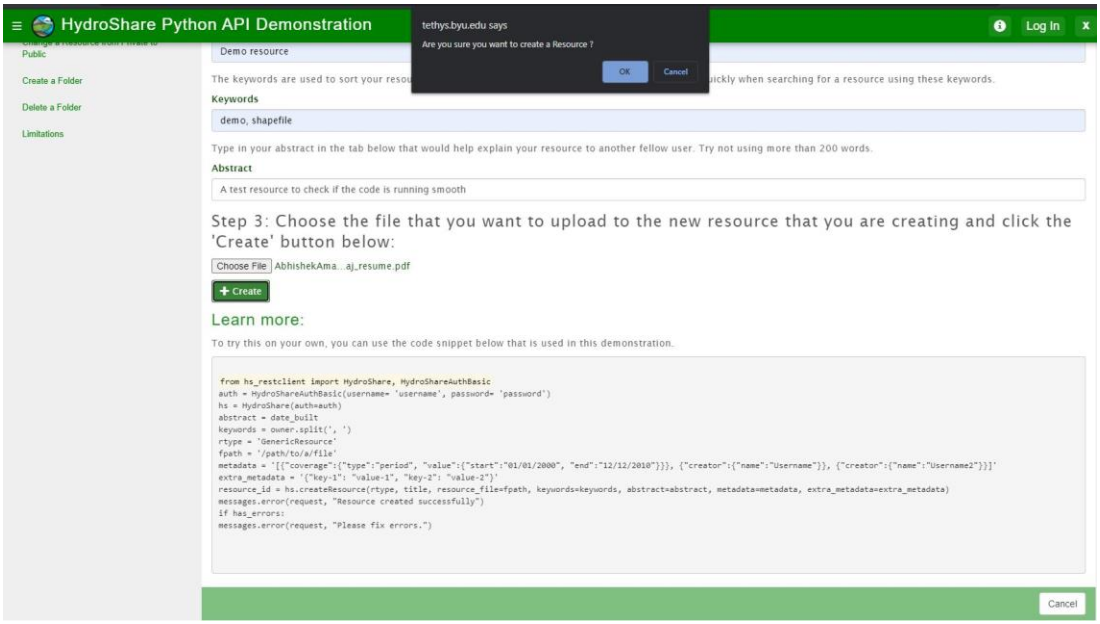

Figure 7 Pop up input prompt message box

e) Tethys has a success message and an error message that is displayed depending on the success of the operation. In our case, we get a message saying that the operation was successful.

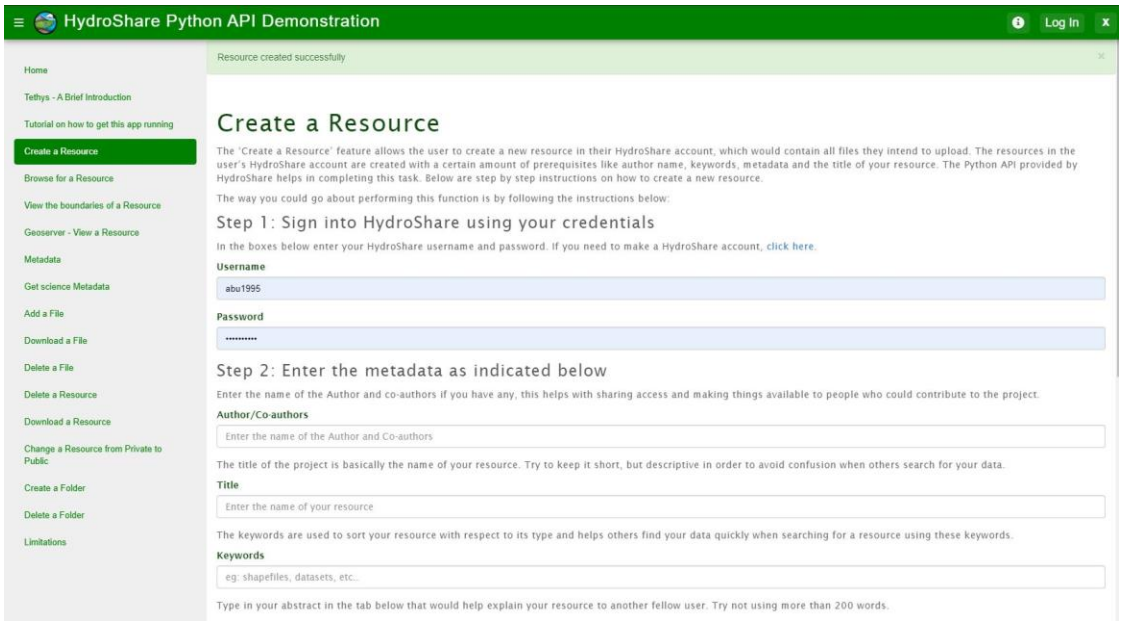

Figure 8 Success message - a default Tethys function

f) Shown below is the HydroShare website with the resource created successfully and is added to the resources that the author owns.

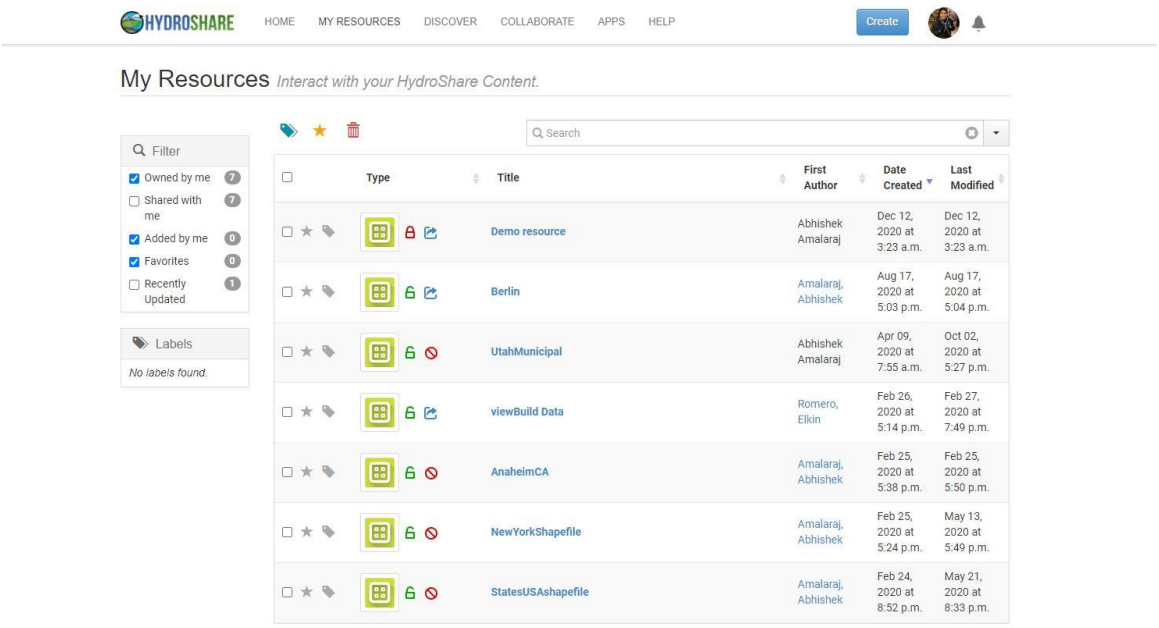

Figure 9 Resource added successfully to the HydroShare database

g) Below, we can see the information we entered in our app.

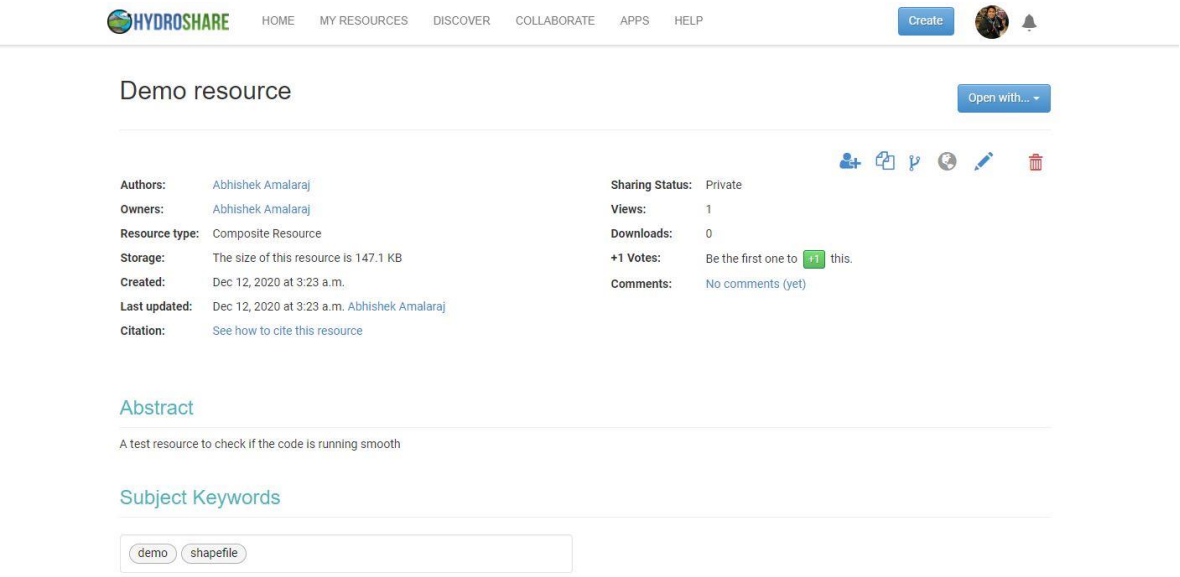

Figure 10 Resource created with the information entered in our app

h) The GUI of the 'Browse for a Resource' function which searches and displays the name and resource ID of a function based on the parameter 'subject'

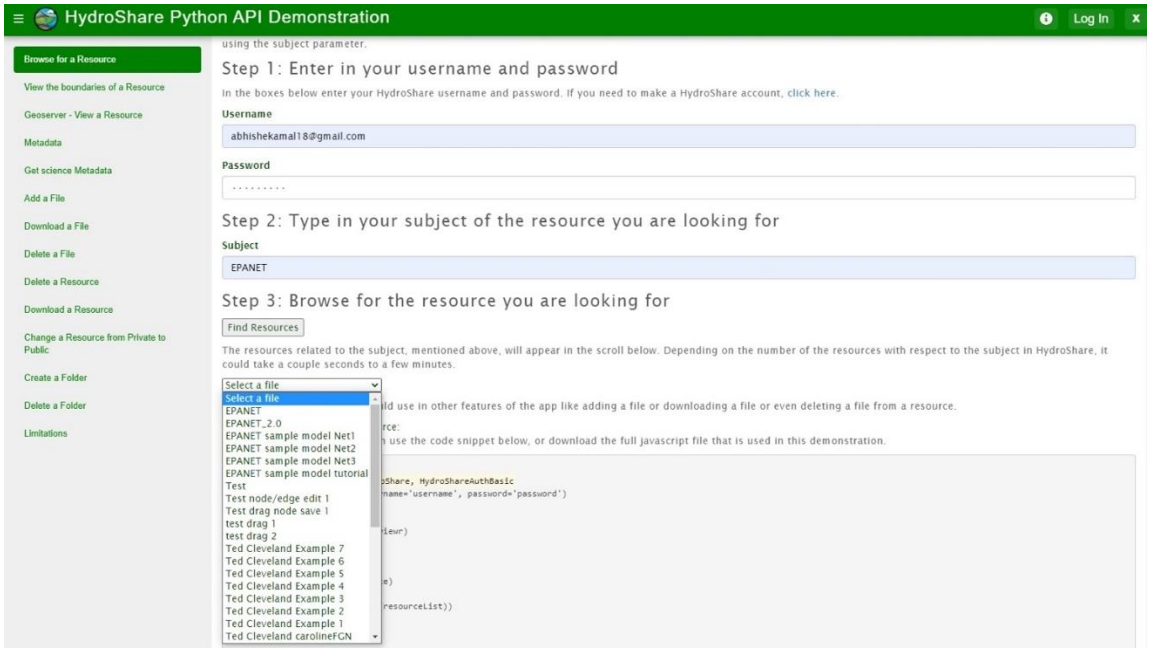

Figure 11 'Browse for a Resource' User Interface

i) The GUI of the 'View the boundaries of a resource' function which shows the geographic extent of a resource.

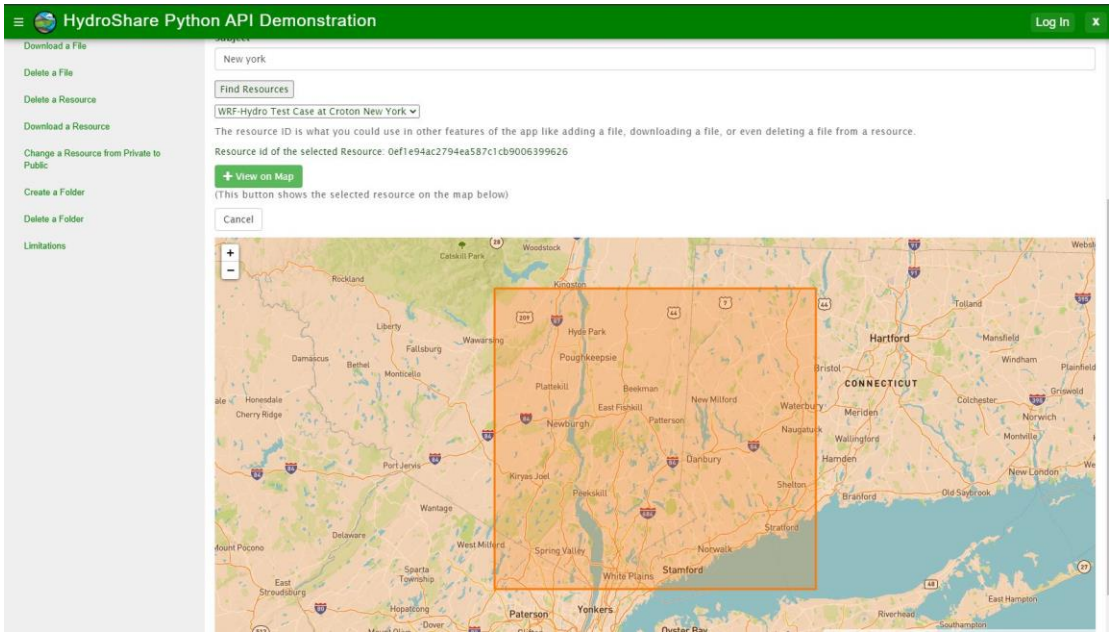

Figure 12 Boundary of the Resource is displayed on the map

j) The feature shown below fetches the metadata of a resource and display it on the map.

Figure 13 'Get science Metadata' User Interface

k) A resource created in the app is of type ‘Private’. The feature below exists to change the type to ‘Public’ by entering the resource ID of the resource that was created.

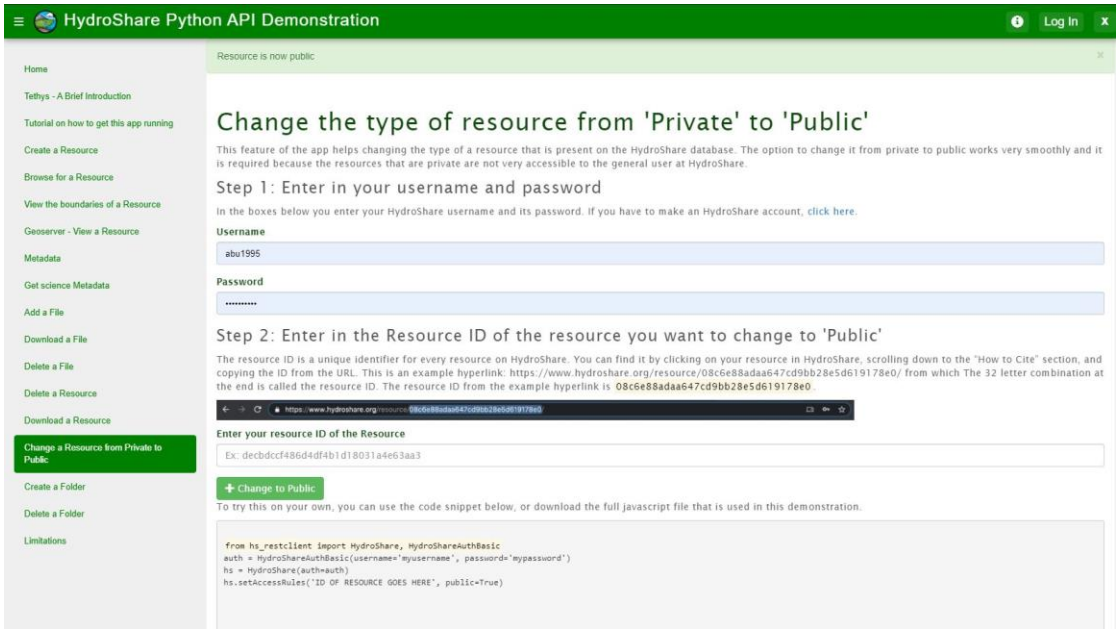

Figure 14 Private to public feature in the app

The ‘GeoServer – View a Resource’ feature that was created to show the resource shapefile in a separate window on the app.

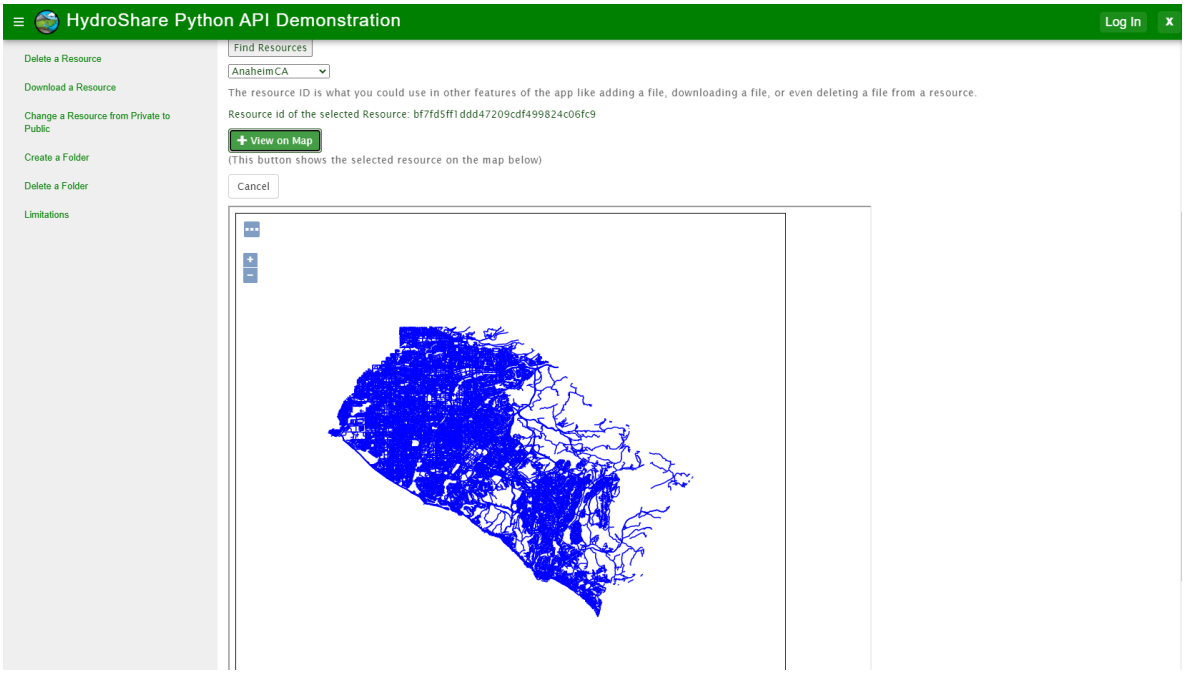

Figure 15 GeoServer - an improvised function in the app

This function is executed fetching the information of a resource that contains the geo-spatial content of a shapefile and displays it in a window (Iacovella 2017).

4.0 Discussion and Conclusions

Building a web application from scratch is a task that is becoming more common from day to day. However, in the world of civil and environmental engineering a beginner level programmer always has a need to understand the concept of building a web app even better. A web app that explains itself to a user on how it integrates an API in a web app and smoothly performs functions like interacting with a database can be very beneficial. Cloud computing and availability of apps online has always been a goal for BYU Hydroinformatics and to achieve this goal and expand on it, more web programmers are that not only understand programming online but also know how water data and shapefiles work and visualize.

This project helps accelerate that process and brings the user closer to the middle ground where he is close to both the back-end programming and understand what are the elements that are being put together for function and execution of tasks and also the front-end user experience and customizing the interface to be simpler to understand. This project was built in that trajectory as the person behind the project had very little programming experience and had to learn from the very start all the skills demonstrated in this paper.

Tethys is a product of BYU Hydroinformatics and is motivated to bring more engineers to collectively develop cloud computing and data management. This project complements Tethys on its mission by educating a programmer new to the field on the above stated purposes. Since most of the programming is done in Python it follows along with the framework that Tethys uses, which is basically a Django platform. So, there is a factor of convenience as well.

The project also helps the general user community as any help offered to understand web programming can only help improve the situation of any company. The concepts discussed and explained in the paper focuses on not just helping the BYU Civil and Environmental Engineering team but the whole web app development ecosystem. A little work is done in this project in explaining the different ways we can modify the API and perform advance functions that are not primarily covered in the API itself. The project also helps with spreading the range of HydroShare and making it more ubiquitous. The more the API is explored and used in different applications, the more it helps HydroShare with making hydrology and water data available to engineers and scientists around the world.

Acknowledgements

This work was supported by the National Science Foundation under collaborative grants ACI 1148453 and 1148090 for the development of HydroShare (**http://www.hydroshare.org**). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

Software Availability

HydroShare API Tethys App, Version 1.0 code is open source and available online as of January 1, 2021, retrievable at https://github.com/BYU-Hydroinformatics/hydroshare_api_tethysapp.

References

Ames, Daniel P., Jeffery S. Horsburgh, Yang Cao, Jiří Kadlec, Timothy Whiteaker, and David Valentine. 2012. "HydroDesktop: Web Services-Based Software for Hydrologic Data Discovery, Download, Visualization, and Analysis." *Environmental Modelling & Software* 37 (November): 146–56. https://doi.org/10.1016/j.envsoft.2012.03.013.

Crawley, Shawn, Daniel Ames, Zhiyu Li, and David Tarboton. 2017. "HydroShare GIS: Visualizing Spatial Data in the Cloud." *Open Water Journal* 4 (1): 3–20.

Fajardo, E. M., J. M. Dost, B. Holzman, T. Tannenbaum, J. Letts, A. Tiradani, B. Bockelman, J. Frey, and D. Mason. 2015. "How Much Higher Can HTCondor Fly?" *Journal of Physics: Conference Series* 664 (6): 062014. https://doi.org/10.1088/1742-6596/664/6/062014.

Farkas, Gabor. 2016. *Mastering OpenLayers 3*. Packt Publishing Ltd.

Forcier, Jeff, Paul Bissex, and Wesley J. Chun. 2008. *Python Web Development with Django*. Addison-Wesley Professional.

Gan, Tian, David G. Tarboton, Pabitra Dash, Tseganeh Z. Gichamo, and Jeffery S. Horsburgh. 2020. "Integrating Hydrologic Modeling Web Services with Online Data Sharing to Prepare, Store, and Execute Hydrologic Models." *Environmental Modelling & Software* 130 (August): 104731. https://doi.org/10.1016/j.envsoft.2020.104731.

Gries, David. 2012. *The Science of Programming*. Springer Science & Business Media.

Hazzard, Erik. 2011. *OpenLayers 2.10 Beginner's Guide*. Packt Publishing Ltd.

Holovaty, Adrian, and Jacob Kaplan-Moss. 2009. *The Definitive Guide to Django: Web Development Done Right*. Apress.

Horsburgh, Jeffery S., David G. Tarboton, David R. Maidment, and Ilya Zaslavsky. 2008. "A Relational Model for Environmental and Water Resources Data." *Water Resources Research* 44 (5).

Jones, Norm, Jim Nelson, Nathan Swain, Scott Christensen, David Tarboton, and Pabitra Dash. 2014. "Tethys: A Software Framework for Web-Based Modeling and Decision Support Applications."

Khare, R., and R. N. Taylor. 2004. "Extending the Representational State Transfer (REST) Architectural Style for Decentralized Systems." In *Proceedings. 26th International Conference on Software Engineering*, 428–37. https://doi.org/10.1109/ICSE.2004.1317465.

Khattar, Rohit, and Daniel P. Ames. 2020. "A Web Services Based Water Data Sharing Approach Using Open Geospatial Consortium Standards." *Open Water Journal* 6 (1): 2.

Michaelis, Christopher D., and Daniel P. Ames. 2012. "Considerations for Implementing OGC WMS and WFS Specifications in a Desktop GIS" 2012 (April). https://doi.org/10.4236/jgis.2012.42021.

Ofoeda, Joshua, Richard Boateng, and John Effah. 2019. "Application Programming Interface (API) Research: A Review of the Past to Inform the Future." *International Journal of Enterprise Information Systems (IJEIS)* 15 (3): 76–95.

Ong, Shyue Ping, Shreyas Cholia, Anubhav Jain, Miriam Brafman, Dan Gunter, Gerbrand Ceder, and Kristin A. Persson. 2015. "The Materials Application Programming Interface (API): A Simple, Flexible and Efficient API for Materials Data Based on REpresentational State Transfer (REST) Principles." *Computational Materials Science* 97 (February): 209–15. https://doi.org/10.1016/j.commatsci.2014.10.037.

Rajasekar, Arcot, Reagan Moore, Chien-Yi Hou, Christopher A. Lee, Richard Marciano, Antoine de Torcy, Michael Wan, et al. 2010. "IRODS Primer: Integrated Rule-Oriented Data System." *Synthesis Lectures on Information Concepts, Retrieval, and Services* 2 (1): 1–143. https://doi.org/10.2200/S00233ED1V01Y200912ICR012.

Ramsey, Paul, and Victoria-British Columbia. 2005. "Introduction to Postgis." *Refractions Research Inc*, 34–35.

Roberts, Wade, Gustavious P. Williams, Elise Jackson, E. James Nelson, and Daniel P. Ames. 2018. "Hydrostats: A Python Package for Characterizing Errors between Observed and Predicted Time Series." *Hydrology* 5 (4): 66. https://doi.org/10.3390/hydrology5040066.

Sadler, Jeffrey M., Daniel P. Ames, and Shaun J. Livingston. 2016. "Extending HydroShare to Enable Hydrologic Time Series Data as Social Media." *Journal of Hydroinformatics* 18 (2): 198–209. https://doi.org/10.2166/hydro.2015.331.

Swain, Nathan R., Scott D. Christensen, Alan D. Snow, Herman Dolder, Gonzalo Espinoza-Dávalos, Erfan Goharian, Norman L. Jones, E. James Nelson, Daniel P. Ames, and Steven J. Burian. 2016. "A New Open Source Platform for Lowering the Barrier for Environmental Web App Development." *Environmental Modelling & Software* 85 (November): 11–26. https://doi.org/10.1016/j.envsoft.2016.08.003.

Youngblood, Brian. 2013. *GeoServer Beginner's Guide*. Packt Publishing Ltd.

Zhou, Wei, Li Li, Min Luo, and Wu Chou. 2014. "REST API Design Patterns for SDN Northbound API." In *2014 28th International Conference on Advanced Information Networking and Applications Workshops*, 358–65. https://doi.org/10.1109/WAINA.2014.153.

