52north exploring horizons

Google Summer of Code 2019

Angular WPS Client Library

Aditya Soni
Integrated M. Tech Geological Technology,
Department of Earth Sciences
Indian Institute of Technology, Roorkee
Roorkee, Uttrakhand, India

Personal Information

Name : Aditya Soni

Email : asoni@es.iitr.ac.in

IRC Nick : adityasoni

GitHub : aditya-soni

Resume : Link

LinkedIn : https://www.linkedin.com/in/aditya-soni/

Phone No : +91-7597298876

Location : Roorkee, Uttrakhand, India

Time Zone : IST (UTC +5:30)

University Information

University : Indian Institute of Technology, Roorkee

Major : Geological Technology

Current Year : 4th Year (2020 expected graduation)

Degree : Integrated Master of Technology (M. Tech)

Personal Background

I'm a 22 year old, fourth-year undergraduate student currently enrolled in Geological Technology (V Year Course) at IIT Roorkee. I'm passionate about web development and competitive programming and have

I have a professional experience of working in a fast-paced startup environment. In the past, I've successfully completed three internships, where the majority of my work was based on client-side and server-side programming. Through my past working experiences, I've developed proficiency in writing scalable & efficient code and developing digital products from ideation to production using . It also helped me grow my skills in team collaboration, verbal communication and project leadership.

Why I'm interested in Open Source ?

I've been using various open source libraries and software available since my initial days as a developer. They helped me ease my task as I can focus more on the important aspects of my applications. What I like about open source is you can tweak/remodel available resources to fit your own requirements which helps not only you but also others who are stuck in the same situation like you. From these habits of tweaking, I also learned the art of writing reusable code which can be used by the community. For me Open Source is a great way to learn and grow together as a community. Due to this I've always fascinated by the idea of contributing towards Open Source and that's what driving me to participate in Google Summer of Code.

Project Description

1. Overview

Link to Project Idea - Angular WPS Client Library

Web Processing Service (WPS) provides client access across a network to pre-programmed calculations and/or computation models that operate on spatially referenced data. These Calculations can be extremely simple or highly complex, with any number of data inputs and outputs. Generally, to interact with a WPS a user requires an API which standardize the inputs and outputs for geospatial processing services. [1]

The WPS JavaScript Library by 52°North provides this functionality of querying Web Processing Services of Version 1.0.0 and 2.0.0 for applications based on Vanilla JS. However, this library does not fully supports all the features and typings for TypeScript based frameworks such as Angular. This is a major concern as Angular is one of the most popular framework when it comes to web application development.

To resolve this problem we need to implement an Angular Library which provides the features and methods required for OGC WPS interaction.

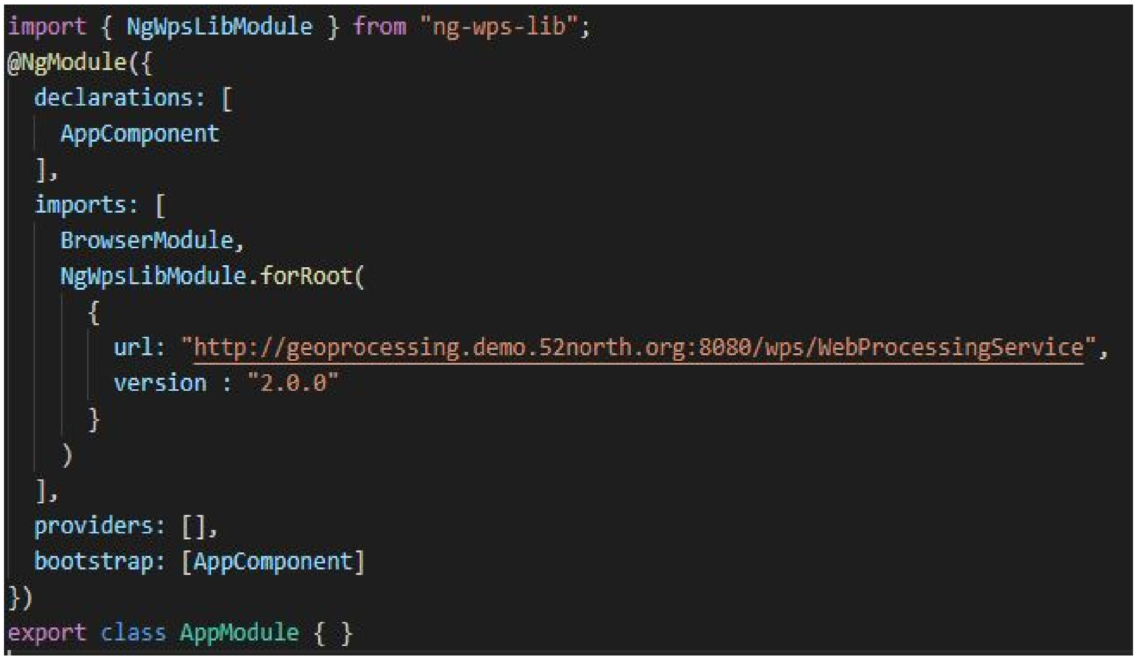
In this project, I shall primarily focus on :

1. **Milestone #1** : Creating an Angular Based Library which works reliably with WPS version 1.0.0 and 2.0.0 using best practices and WPS Interface Standards.
2. **Milestone #2** : Implementing methods for :
    a. GetCapabilities
    b. DescribeProcess
    c. ExecuteProcess (sync/async)
    d. DismissProcess
    e. MonitorProcess
3. **Milestone #3** : Creating typescript classes and interface to :
    a. Parse WPS Responses and Properties.
    b. Generate Inputs and Outputs for various geospatial data types such as Literal Data, Complex Data and Bounding Box Data.
4. **Milestone #4** : Writing Test cases and unit testing for all the features and APIs.
5. **Milestone #5** : Developing a tester angular application which consumes the Angular WPS Library.
6. **Milestone #6** : Writing tutorials and documentation on how to import and utilize this library.

2. Detailed Description

2.1. Initializing the Angular WPS Module

Once the library is installed, the WPS Module will be available for importing in a Child/App Module. The WPS Module will be initialized by consuming a configuration object in the **forRoot()** or **forChild()** method. This configuration object will be of type “**InitialConfig”** containing parameters : **WPS_URL** and **Version**.



```
import { NgWpsLibModule } from "ng-wps-lib";
@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    NgWpsLibModule.forRoot(
      {
        url: "http://geoprocessing.demo.52north.org:8080/wps/WebProcessingService",
        version : "2.0.0"
      }
    )
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

After this a **WPSService, InputGeneratorService** & **OutputGeneratorService** will be available for importing in a component, which provides methods and features for interacting with Web Processing Service.

List of methods provided by WPSService -

1. GetCapabilities
2. DescribeProcess
3. Execute (async/sync)
4. DismissProcess
5. MonitorProcess
6. setVersion
7. setURL
8. getWPS_1_0_0Response // Only for async WPS 1.0.0 execution
9. getStatus // Only for async WPS 2.0.0 execution
10. getResult // Only for async WPS 2.0.0 execution

2.2. Retrieving WPS offered Processes (GetCapabilities)

#Use : The GetCapabilities method provides a list of all the available processes on the WPS along with details on individual processes. It comprises capabilities for process discovery and retrieval of process descriptions.

# Input Parameters : None

# Output Parameters : Callback containing Response:<Capabilities>

# Request : GetCapabilities method will execute a HTTP Get Request to **WPS_URL** with query parameters service**,** version and request**.**

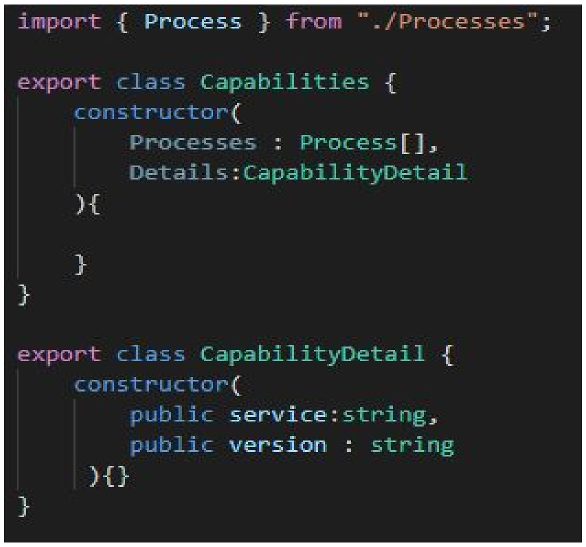
# Response : GetCapabilites will return a callback function containing Response. The structure of Response depends on these two cases :

a) Error

In case of an error no valid response object will be constructed and the response will include two properties “textStatus” & “errorThrown”.

b) Success

In case of successful request the response will be a “**Capabilities”** object containing the WPS **Details** and **Processes** array.



```
import { Process } from "./Processes";

export class Capabilities {
    constructor(
        Processes : Process[],
        Details:CapabilityDetail
    ){

    }
}

export class CapabilityDetail {
    constructor(
        public service:string,
        public version : string
     ){}
}
```

Capabilities TypeScript Class

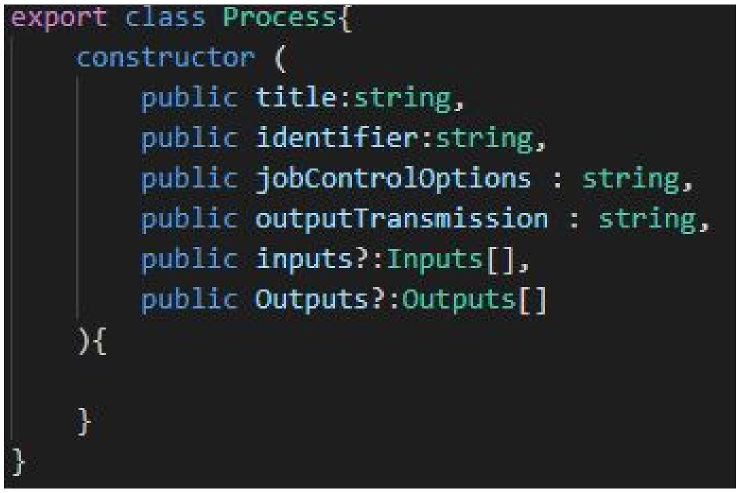
# Sample Response :

```
{
  Details: {service: "WPS", version: "2.0.0"}
  Processes: (221) [Process, Process, Process, Process, …]
}
```

Where, a single process will be an object of TypeScript Class **“Process”**

```
{
  identifier: "org.n52.wps.server.algorithm.r.AnnotationValidation"
  jobControlOptions: "sync-execute"
  outputTransmission: "value reference"
  title: "R Annotation Validation"
}
```

And the **Process** Class looks like,



```
export class Process{
    constructor (
        public title:string,
        public identifier:string,
        public jobControlOptions : string,
        public outputTransmission : string,
        public inputs?:Inputs[],
        public Outputs?:Outputs[]
    ){

    }
}
```

Process TypeScript Class

2.3. Retrieving details on a single process (DescribeProcess)

#Use : The **DescribeProcess** Method provides details about a single process, it’s input-output parameters, data types, responseFormat etc.

#Input Parameters : processIdentifier:<String>

#Output Parameters : Callback containing Response:<processOfferings>.

# Request : DescribeProcess method will execute a HTTP Get Request to **WPS_URL** with query parameters service, version, request and identifiers.

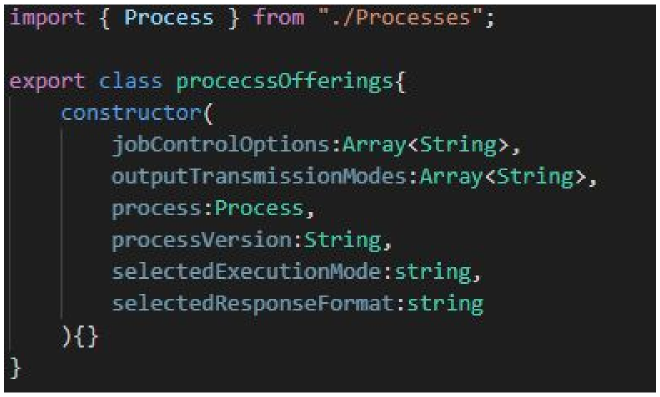
#Response : DescribeProcess will return a callback function containing Response. The structure of Response depends on these two cases :

a) Error

In case of an error no valid response object will be constructed and the response will include two properties "textStatus" & "errorThrown".

b) Success

In case of successful request the response will be a "**processOfferings"** object.



```
import { Process } from "./Processes";

export class procecssOfferings{
    constructor(
        jobControlOptions:Array<String>,
        outputTransmissionModes:Array<String>,
        process:Process,
        processVersion:String,
        selectedExecutionMode:string,
        selectedResponseFormat:string
    ){}
}
```

Process Offerings TypeScript Class

# Sample Response :

```
{
jobControlOptions: (2) ["sync-execute", "async-execute"]
outputTransmissionModes: (2) ["value", "reference"]
process: {

        title: "QuakeMLProcess for BoundingBox input",

        identifier: "org.n52.wps.python.algorithm.QuakeMLProcessBBox",

        inputs: [Input, Input, Input, …],

        outputs: [Output, Output, …]

       }

processVersion: "1.0.0"
selectedExecutionMode: "sync-execute"
selectedResponseFormat: "document"
}
```

2.4. Executing a Process on WPS (Execute)

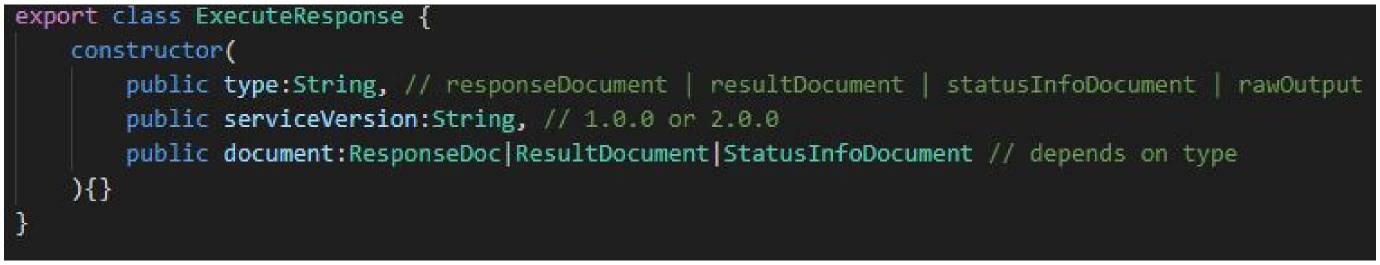
# Use : The Execute Method will be used to execute a process on the WPS. This may be a synchronous or asynchronous process.

# Input Parameters : processIdentifier:<String>, responseFormat:<String>,executionMode:<String>,inputs:Input[], Output[]

# Output Parameters : Callback containing Response:<ExecuteResponse>.

# Request : Execute method will execute a HTTP POST Request to **WPS_URL** Along with a xml payload containing inputs and outputs.

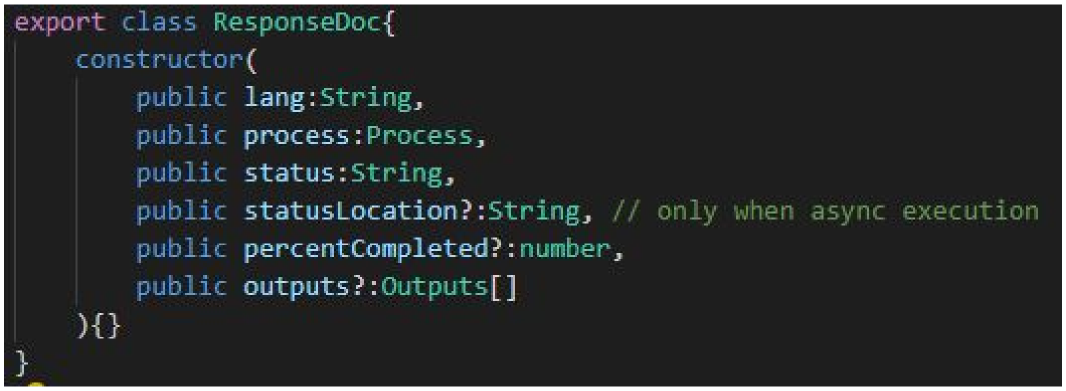
# Response : The response of Execute Method will be different in case of version 1.0.0 and 2.0.0



```
export class ExecuteResponse {
    constructor(
        public type:String, // responseDocument | resultDocument | statusInfoDocument | rawOutput
        public serviceVersion:String, // 1.0.0 or 2.0.0
        public document:ResponseDoc|ResultDocument|StatusInfoDocument // depends on type
    ){}
}
```

ExecuteResponse TypeScript Class

a) ResponseDoc - WPS 1.0.0 response document



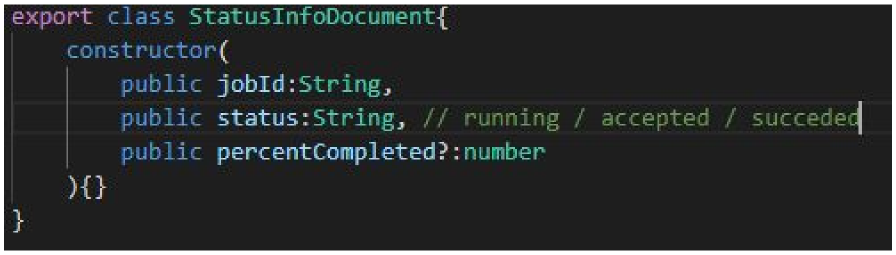
```
export class ResponseDoc{
    constructor(
        public lang:String,
        public process:Process,
        public status:String,
        public statusLocation?:String, // only when async execution
        public percentCompleted?:number,
        public outputs?:Outputs[]
    ){}
}
```

ResponseDoc TypeScript Class

Sample Response :

```
{
type: "responseDocument",
serviceVersion: "2.0.0"
document: {
             lang : "en-us",
             process : {
                     identifier:
             "org.n52.wps.python.algorithm.QuakeMLProcess",
                     title: "QuakeMLProcess",
                     outputs: []
                     }
              status : "ProcessAccepted",
              statusLocation :
             "https://riesgos.52north.org/wps/RetrieveResultServlet?id=74a2d6ae
             -479d-4626-9ff1-b652d1941d06"
            }
 }
```

b) StatusInfoDocument - WPS 2.0.0 async execute



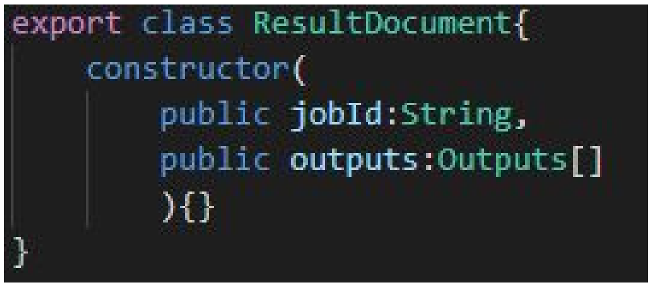
```
export class StatusInfoDocument{
    constructor(
        public jobId:String,
        public status:String, // running / accepted / succeded
        public percentCompleted?:number
    ){}
}
```

StatusInfoDocument TypeScript Model Class

Sample Response :

```
{
type: "statusInfoDocument",
serviceVersion: "2.0.0"
document:{
                jobId: "e5714294-616f-4057-88ca-5f741b0d5e06"
                percentCompleted: 0
                status: "Accepted" or "Running" or "Succeeded"
           }
}
```

c) ResultDocument - WPS 2.0.0 sync execute



```
export class ResultDocument{
    constructor(
        public jobId:String,
        public outputs:Outputs[]
        ){}
}
```

ResultDocument TypeScript Model Class

Sample Response :

```
{
type: "resultDocument",
serviceVersion: "2.0.0"
document: {
                jobId: "2158aab5-96ea-4220-a910-c3459c5e9d02",
                outputs: [Output, Output, …]
              }
 }
```

2.5. Retrieving results for WPS 1.0.0 Process (getWPS_1_0_0Response)

#Use : This method retrieves the response document stored on WPS server resulting due to Async Execute for WPS 1.0.0

# Input Parameters : statusLocation<String>

# Output Parameters : Response:<ResponseDoc>

# Request : This method will execute a HTTP GET Request to **statusLocation.**

# Response: In case of successful request the response will be a "**ResponseDoc"** object.

Sample Response :

```
  lang : "en-us",
  process : {
          identifier: "org.n52.wps.python.algorithm.QuakeMLProcess",
          title: "QuakeMLProcess",
          outputs: [Output, Output, …]
          }
   status : "ProcessAccepted",
   statusLocation :
  "https://riesgos.52north.org/wps/RetrieveResultServlet?id=74a2d6ae-479d-4626-9f
  f1-b652d1941d06"
}
```

2.6. Retrieving information on a running process (getStatus)

# Use : The getStatus methods will be implemented to retrieve the status of an async WPS 2.0.0 process.

#Input Parameters : jobId:<String>

#Output Parameters : Callback containing Response:<ExecuteResponse>

# Request : This method will execute a HTTP GET Request to **WPS_URL** along with request & jobId as query parameters**.**

# Response : In case of successful request the response will be a “**ExecuteResponse”** object.

2.7. Retrieving result for a Asynchronous process (getResult)

# Use : The getResult methods will be implemented to retrieve the result of an async WPS 2.0.0 process.

#Input Parameters : jobId:<String>

#Output Parameters : Callback containing Response:<ExecuteResponse>

# Request : This method will execute a HTTP GET Request to **WPS_URL** along with request & jobId as query parameters**.**

# Response : In case of successful request the response will be a “**ExecuteResponse”** object.

2.8. Dismissing/Deleting a Process (dismissProcess)

# Use : The dismissProcess method allows a client to communicate to the server that he is no longer interested in the results of a job.

#Input Parameters : jobId:<String>

#Output Parameters : Callback containing Response:<StatusInfoDocument>

# Request : This method will execute a HTTP GET Request to **WPS_URL** along with request & jobId as query parameters**.**

# Response : In case of successful request the response will be a “**StatusInfoDocument”** object containing status = dismissed and the former job id of the process.

2.9. Monitoring an asynchronous Process (monitorProcess)

This method will provide information about a currently running asynchronous process on the WPS Server.

# Input Parameters : jobID:<String>

# Output : Callback function containing Response : <StatusInfoDocument>

2.10. Changing current WPS Version (setVersion)

This method will be implemented to change the version being used by the WPS Library.

# Input Parameters : Version :<String> (“1.0.0” or “2.0.0”)

2.11. Changing current WPS URL (setURL)

This method will be implemented to change the URL being used by the WPS Library.

# Input Parameters : URL:<String> (“1.0.0” or “2.0.0”)

Preliminary Schedule

Community Bonding Period (May 6th to May 27th)

- I will utilise this time to get more familiarized with WPS & WPS-JS Library, dive deeper into concepts of OGC WPS, read the documentation and get to know my mentors and other fellow community members so that we can work together with an amplified efficiency.
- Some of the important things to study up on will be to learn more about unit tests along with planning and strategizing about the changes that would need to be done with the mentor.
- Setting up the required development environment and git repository.

Phase 1 (May 27th to June 24th)

Time Period	Work Plan
Week 1 (27th May - 2nd June)	Set up initializing the Angular WPS Module
Week 2-3 (2nd June - 16th June)	Implementation of setVersion and setURL. Implementing TypeScript classes for ComplexData, Literal Data and BoundingBoxData.
Week 4 (16th June - 22nd June)	Setting up validations. Writing unit testing and work on improvements and bug fixes. Create a Weekly status report.

****First Evaluation****

Phase 2 (June 27th to July 22nd)

Time Period	Work Plan
Week 5-6 (27th June - 9th July)	Make the changes and bug fixes as suggested in feedback. Discuss with mentors on additional functionalities/features to be added. Work on implementing the GetCapabilities, DescribeProcess. (**Milestone #2**)
Week 7 (9th June - 16th July)	Work on creating Typescript classes for Response properties. **(Milestone #3)** Work on Input and Output Generator Classes. **(Milestone #3)**

	Develop Async/Sync Execute feature for WPS 1.0.0 & 2.0.0
Week 8 (16th July - 22nd July)	Writing unit testing and work on improvements and bug fixes. Create a Weekly status report regarding status, problems & next tasks. Create a Weekly status report.

****Second Evaluation****

Phase 3 (July 26th to August 19th)

Time Period	Work Plan
Week 9 (July 26th - 3rd August)	Make the changes and bug fixes as suggested in feedback. Discuss with mentors on additional functionalities/features to be added. Implement the DismissProcess and Monitor Process. **(Milestone #2)** Work on the the tester GUI application which consumes Angular WPS Library **(Milestone #5)**
Week 10-11 (3th August - 16th August)	Complete the tester application by writing unit tests and validations. Work on the Tutorials and Documentation on how to implement and use the library. **(Milestone #6)**
Week 12 (16th August - 22nd August)	This week will be to catch up in case anything is left due to some unforeseen circumstances. If we are well on track and everything is completed, then I'll use this week to test all the changes and finalizing the

	GSoC work and Fix bugs (if any).

****Final Evaluation****

Availability, Working Hours & Other Commitments

My vacations start from 5 May and end on 15 July. The official GSoC period is from 6th May to 26th August. I can easily devote 40-50 hours a week till my college reopens and 30-40 hours per week after that. I'm free on weekends and mostly free on Wednesdays. I intend to complete most of the work before my college reopens.

Apart from this project I do not have any commitments/vacations planned for the summer. I shall keep my status posted to all the relevant community members on a weekly basis and maintain transparency in the project.

Motivation

I'm a student of Earth Sciences and have worked on GeoSpatial Data and am familiar Geographic Information Systems. 52°North is working in advancement of spatial information and GIS Concepts. This is quite intriguing for me as a programmer and as an earth scientist. I've worked extensively Angular in the past and love building my applications on them. I've come across various such scenarios when a library which is perfect for a job is not available for frameworks like angular and react. I used to tweak some open-source angular modules according to my need but I'm yet to work on a creating full-fledged library. This project is a great opportunity for me to learn and apply.

Report on Code Challenge

I've submitted the code challenge to the project mentor Mr. Benjamin Pross and got a positive feedback. I had also shared the challenge on 52°North GSoC mailing list. You can find the repository at [3].

Experiences

Programming Level

I've worked extensively on JavaScript, Angular & NodeJS and have developed various application on them. I'm also familiar with Java & C++. Along with that I've experience of working on back client-side and server-side programming.

Development Experience

#Ingenium Education Portal

Online portal for education institute to manage student data, conduct regular online test, create test and analysis features. I developed the front-end of the application using CKEditor, Angular, Material Design and Bootstrap.

# Edbird.In

Edbird.in is an online learning and educational platform based on Angular 4, Bootstrap, HTML & CSS.

# ThingsAI.IO

A user-friendly, highly customizable data visualization, device management and analytical platform. Tech Stack : NodeJS, MongoDB, Firebase Real Time DB, SocketJS & Angular 5.

# ViZi

ViZi is an ML-based platform which solves the problem of understocking and overstocking of medical inventory. Tech Stack : Java, MySQL, Tomcat Server & JSP

# KV Store

KV-Store is a Key-value storing web service. It is based on NodeJS, Redis, Docker and uses microservice architecture.

Other Projects

Please find all my projects hosted on my GitHub Profile (link) .

Academic Experience

Institute Name : Indian Institute of Technology, Roorkee [4]

Degree : Integrated M. Tech in Geological Technology.

Expected Graduation : 2020

Academic Performance : I've completed courses on Geographical Information System : Introduction, Concepts and Methodologies, Digital Elevation Models and Applications. I've also had courses on basic programming and numerical methods.

Work Experience

Product Developer Internship

Medikabazaar - Mumbai, Maharashta, India

Full-Stack Developer Internship

ThingsCloud Technologies Private Limited - Bangalore, Karnataka, India

Front-End Developer Internship

Edbird.in - Jaipur, Rajasthan, India

After GSoC

- I plan on continuing my contributions to this organization, by adding to my past projects and working on open issues.
- If I cover everything early, I'll try to make bindings for more functions and add them to library and accordingly add tests and documentation and example usages.

References

[1] OpenGIS® Web Processing Service : https://www.opengeospatial.org/standards/wps

[2] WPS-Ng-Client : https://github.com/52North/wps-ng-client

[3] Code Challenge : https://github.com/aditya-soni/ng-wps-lib

[4] WPS-JS : https://github.com/52North/wps-js

[5] General Information : https://wiki.52north.org/Projects/GSoCForStudents