A Project Report on

Angular module to access RESTful Web Services

Submitted in partial fulfilment of the requirements of 2^{nd} semester 2^{nd} year of

Master of Technology
In
Computer Science & Engineering

by

Vijay Jadhav

157506

Under the guidance of

Dr. K. Ramesh

Department of Computer Science & Engineering
NATIONAL INSTITUTE OF TECHNOLOGY WARANGAL
2016-17

Dissertation Approval for M.Tech

This Project Work entitled

"Angular module to access RESTful Web Services"

by

Vijay Jadhav

is approved for the degree of

Master of Technology

in

Computer Science and Engineering

Examiners

_____

_____

_____

Supervisor(s)

_____

_____

_____

Chairman

_____

Place : _____

Date : _____

Declaration

I declare that this written submission represents my ideas in my own words and where others' ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

_____

(Signature)

_____

(Name of the student)

_____

(Roll No.)

Date: _____

Department of Computer Science and Engineering
National Institute of Technology,Warangal

C E R T I F I C A T E

This is to certify that, the projcet entitled **"Angular module to access RESTful Web Services"** is a bonafide work carried out by **Vijay Jadhav** in partial fulfillment of the requirement for the award of the degree of **M.Tech in Computer Science and Engineering** and submitted to the Department of Computer Science and Engineering, National Institute of Technology,Warangal.

Place: Warangal

Date:

Dr. Ch. Sudhakar
Head of the Department
Department of CSE
NIT Warangal

Dr. K. Ramesh
Project Guide
Department of CSE
NIT Warangal

Abstract

Angular is a MVC like front-end web developement framework. Single Page Application (SPAs) are written using Angular that runs inside a browser. It is wriiten in Javascript and mainly uses Typescript for developing applications which are compatible across all platforms. Angular uses modules to broke applications into parts and each module does some specific job.

This project aims to implement a module for Angular to access RESTful Web Services. In an Angular application that need heavily relies on resources exposed via REST, a developer might need to write thousands of line of code to make it run smoothly and handle all possible errors. Using this module in Angular project, resources on the server can be fetched or updated in as few as possible functions calls. This module handles errors and notifies user of the errors.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 What is Angular?

Angular is a MVC like front-end web developement framework [1]. Angular uses modules to broke applications into parts and each module does some specific job. Angular aims to simplfy development and testing by providing a framework for client-side machines in client-server model. Single Page Application (SPAs) are written using Angular that runs inside a browser. It is written in Javascript and mainly uses Typescript for developing applications which are compatible across all platforms. Typescript is a superset of Javascript, it is transpiled into Javascript before execution. You can write Javascript code inside Typecsript code. Typescript extends Javascript by adding features such as Static Typings, Classes, Interfaces, Debugging etc. Typescript helps make Javascript code less prone to runtime errors.

1.1.1 Versions of Angular

Early version of Angular framework was called as AngularJS. Angular 2+ versions are simply called as Angular. It is a completely rewritten version of AngularJS [2]. In this work, Angular refers to the Angular 2+ versions, unless otherwise stated.

1.1.2 Differences in AngularJS and Angular 2+

Differences between AngularJS and Angular 2+ have stated below.

- **Speed :** Angular is much faster than AnguarJS. Boot time of Angular applications has been increased by at least 5 times [3].
- **Architecture :** Angular does not use $scope or controllers anymore, instead it uses a hierarchy of components.
- **Mobile development :** Angular mainly focuses on performance issues which are issential for running app on a Mobile Platform.
- **Modularity :** The core functionality of Angular has been moved to modules, producing a faster, lighter core.
- **Language :** Angular uses typescript to write applications, where as AngularJS uses javascript. TypeScript is a superset of JavaScript.
- **Improved dependency injection :** Angular used a new Improved Dependacy Injection techniques.

1.2 REST

REpresentational State Transfer (REST) [4] is a set of principles that describes how resources on a server machine are addressed and accessed by another machine on the same network. The resources (or also called as Web Services) exposed via REST are often called RESTful Web Services.

Characterization of RESTful Web Services can be given as :

- RESTful Web Services should be stateless [5].
- Every resource should be uniquly addressable.
- Access resources over HTTP commands of GET, PUT, DELETE, POST etc.
- Returned response should in XML or JSON.

In RESTful Web Services, every resource is given a URI (Uniform Resource Identifier). A resource can be a file, single record in the database, set of records, a table or the database itself etc. When this resource is requested by a client machine, it is converted into a uniform format (XML or JSON) before sending which can be understood

by both machines. RESTful Web Services are stateless [5]. All the session information is stored only in the client machine. Each request from any client carries all the neccessary information to successfully complete the request.

SOAP [6] is another standard as an alternavice for REST but with some differences. SOAP-based Web Services have an official standard, but REST-based Web Services don't. SOAP is a protocol, but REST is an architectural style. REST is not really a standard but it makes use of other standards such as HTTP, XML, URI, JSON etc.

1.2.1 REST Semantics

A RESTful API (Application Programming Interface) is simply a collection of URIs (Uniform Resource Indentifiers), all HTTP requests to these URIs and some JSON (XML is also preferred as much as JSON) representations of resources. These URIs should follow a principal that adheres to REST. Each resource on the server has its own URI or we can also call it an address.

For resources residing on the server, nouns should be used as opposed to verbs which describes actions. A RESTful URI should refer to a resource that is a thing instead of referring to an action[7].

Some example of the server resources are given as follows:

- Users of the bulletin board system.
- A list of courses taught by a professor.
- A user's posts over a period of time.
- An article about global warming.

Each resource on the server exposing RESTful Web Services will have at least one URI to identify that resource. URIs should follow a predictable, hierarchical structure to enhance understandability and, therefore, usability. This is not a REST rule or constraint, but it keeps the API look better and easier to follow[7].

Below some examples explained how one should define URIs for server resources:

To create a new student on the server:

```
1       POST http://www.example.com/students\\
```

To read a student with Student ID 33:
GET http://www.example.com/students/33
The same URI can be used for PUT and DELETE, to update and delete, respectively.

Here is an example URIs for courses:
POST http://www.example.com/courses for creating a new course.

GET—PUT—DELETE http://www.example.com/courses/62
for reading, updating, deleting course 62, respectively.

What if there is a need to apply to new course for a student? One option can be:
POST http://www.example.com/courses
And that could work to add a course, but it looks like it is outside the context of a student.

Because we want to add a new course for a student, this URI doesnt look , as it should be. It could be argued that the following URI would offer better clarity:
POST http://www.example.com/students/33/courses
Now we know we're adding a new course for Student ID 33.

Now what would the following return?
GET http://www.example.com/students/33/courses
Probably a list of courses for student id 33 has has enrolled. Note: we may choose to not support DELETE or PUT for that url since it's operating on a collection.

1.3 HTTP

HTTP is very popular Request-Response protocol in the clientserver model in a network. A browser software, for example, could be the client in this model and an application running on a other machine having a website may be the server in this model. The client makes an HTTP request to the server. The server provides resources such as HTML files and other many types of content, or it performs other tasks on behalf of the client machine, returns a response message to the client. The response received from server contains status information about the sent request and possibly could also contain requested content in its message body.

1.3.1 HTTP Methods

HTTP protocol defines 'methods' (sometimes also called as verbs) to indicate the required action that needs to be performed on the identified resource. This resource represents, whether existing data or data that is generated on the fly, relies on the implementation of the server.

1. **GET :** The HTTP GET method is used to read a resource residing on a server. In a successful request, GET returns a response in XML/JSON and an HTTP response code of 200. In case if an error occures , it returns code 4** or 5**, but mostly returns a 404 (404 is a NOT FOUND) or 400 (400 is a BAD REQUEST).

   GET requests are used only to read resources and not to change them. So, when used this way (i.e without modifiying anything on the server), they are considered safe. They (requests) can be called without any risk of data modification (or data corruption)calling it once has the similar effect as calling it 100 times, or none at all. Also, GET is a idempotent request, which means that making multiple similar requests will have the same result as a single very first request.

   Examples:
   GET http://www.example.com/students/123
   GET http://www.example.com/students/123/courses

2. **HEAD :** The HEAD method in the HTTP protocol desires for a response similar to that of a GET request, but no response body should be there. This is very useful

for fetching metadata specified in response headers, without having to transport the full content that comes with that request.

3. **POST :** The POST method in HTTP is mostly used to create new resources on the server. On successful creation of a resourse, it return HTTP response status 201, returning a Location header with a link to the just created resource with the 201 response status.

   POST method is neither safe nor idempotent. It is desirable only for non-idempotent resource requests. Making two similar POST requests will mostly result in two resources containing the same type data.

   Examples:
   POST http://www.example.com/students
   POST http://www.example.com/students/123/courses

4. **PUT :** PUT is used to update the resources residing on server, PUT-ing to a known resource URI with the request body containing the newly-updated resource. PUT can also be used to create new resources, but it is not recommanded. Also, use POST to create new resources and provide the client-defined ID in the body.

   On successful update operation, it should return 200 status code (or 204 if not returning any content in the body) from a PUT. If PUT used to create, should return HTTP status code 201 on successful creation of the resource. A body in the response is optional. It is optional to return a link in the location header in the creation case since the client already set the resource ID.

   PUT method is not safe, means it modifies a resource (or creates) state on the server, but it is idempotent. If you create or update a resource residing on the server using PUT method and then make that same request again, the resource is still there and also has the same state as it did with the very first request. If calling PUT on a resource increments a counter value within the resource residing on the server, the call is no longer said to be idempotent. Many times it happens and it may be enough to document that the call is not idempotent. However, it's recommended to keep PUT requests idempotent.

   Examples:
   PUT http://www.example.com/students/123

PUT http://www.example.com/students/123/courses/987

5. **DELETE :** DELETE method in the HTTP protocol is used to delete a resource which is indentified by a URI address. On successful deletion of the resource, it returns HTTP status code 200 (also called as OK response) with a response body, with a possible wrapped response. Either that or return HTTP status code 204 (called as NO CONTENT) without any response body in the reply. A 204 status code with no body, or the JSON-style response and HTTP status 200 are the recommended responses in this case.

   DELETE operations are idempotent operations. If you DELETE a resource residing on the server, it's will be removed, if access is granted. Calling DELETE multiple times on that resource ends up the same one: the resource is deleted. If calling DELETE , decrements a counter value, the DELETE call is no longer remains idempotent. Using POST for non-idempotent resource requests is desired and adheres to quality.

   Calling DELETE method on a resource, more than once will often return a 404 (also called NOT FOUND) since it was already deleted and so it is no longer available to delete. It just no longer exists. This makes DELETE operations no longer idempotent, but the end-state of the resource remains the same. Returning a 404 is ok and communicates well the status code of the request.

   Examples:
   DELETE http://www.example.com/students/123
   DELETE http://www.example.com/students/123/courses

6. **PATCH :** PATCH is mostly be used for partially modifying resources residing on the server. The PATCH request only needs to contain the changes to the resource, it does not need the complete resource. This is similar to PUT, but the body of the request contains a set of operation that describe how a resource currently residing on the server should be modified to produce a new version of the resource.

   PATCH method is neither safe nor idempotent operation. But a PATCH method request can be made in such a way that it looks idempotent, it also helps to prevent bad outputs that come from collisions between two PATCH requests on the same resource at the same time. Collisions from many PATCH requests may be a lot more dangerous than PUT collisions because some patch formats need to operate

from a known base-point. Clients using this kind of patch application hould be using a conditional request such that the request will be failed completely if the resource has been updated since the time that client last accessed the resource. For example, the client can make a use of ETag in an If-Match header on the PATCH request made to the server.

Examples:
PATCH http://www.example.com/students/123
PATCH http://www.example.com/students/123/courses/987

1.4 Observables

Observable are a very integral part of Angular. It is not just specific to Angular, but it is a popular standard used to manage asyncronous data. Observables allow a continuous stream of communication in which multiple data values can be sent over time. We get a pattern of processing the data by using operations similar to array operations to parse, modify. Angular framework makes use of observables broadly. Mostly used in the HTTP service and the event system of Angular.

In an ordinary method call (which are synchronous), something like this happens:

1. Call a method.
2. Wait for the method to finish.
3. Store the return value from that method in a variable.
4. Use that variable and its new value to do something useful.

In the asynchronous model the flow goes more like this:

1. Define a method that does some work and returns a value from the asynchronous call
2. Define this asynchronous call as an Observable.
3. Subscribe to this Obervable and provide a method to be called.

4. Continue processing other things; whenever the call returns, the provided method will begin to operate on its return value or values – the items emitted by the Observable.

In many programming tasks, you more or less expect that the instructions (programs) you write and implement will execute and complete incrementally, one by one, in the order as you have written them. But in some cases, many instructions might execute in parallel and their results are captured afterwards, in random order, by observers. Rather than calling a function, you define a technique for fetching and transforming the data, in the form of an Observable, and then subscribe to it.

An advantage of this technique is when you have a lot of tasks that are independent of each other, you can start them all at the same time rather than waiting for each one to complete before starting the another one. This way your entire all of your tasks only takes as long to complete as the longest task among all tasks.

1.5 Data Formats used

JSON, or JavaScript Object Notation, is a minimal, readable format for structuring data. It is used mainly to transmit data between a server and a client, as an alternative to XML.

It is a very common data format used for asynchronous client/server communication, including as a replacement for XML in some AJAX-style systems. JSON is a language-independent data format. It was derived from JavaScript, but as of year 2017 many programming languages include code to generate and parse JSON-format data. The official Internet media type for JSON is 'application/json'. JSON filenames use the extension '.json'.

1.5.1 JSON

What is JSON?

JSON, or JavaScript Object Notation, is a minimal, readable format used for structuring data. It is used primarily to send data between a server and web application(or client), as an alternative to XML.

Keys and Values

The two primary properties that are part of JSON are keys and values.

Key: A key is always a string specified in quotation marks. Value: A value can be a string, number, boolean value (true or false), array, or object. Key/Value Pair: A key value pair has a specific syntax, with the key then a colon followed by the value. Key/value pairs are separated by using a comma. Let's take one line example of JSON and identify each part of the code.

"foo" : "bar" This example is a key/value pair. The key is "foo" and the value is "bar".

Types of Values

Array: An associative array of values. Boolean: True or false. Number: An integer. Object: An associative array of key/value pairs. String: Zero or more text characters which mostly form a word.

Arrays

Array can be used as value in key/value pairs if JSON object. Array is represented by a square brackets, within it all values of array separated by comma.

"foo" : "bar" : "Hello", "baz" : ["quuz", "norf"]

Objects

An object is specified by curly brackets. Everything inside of that curly brackets is part of that object. So that means "foo" and the corresponding object are a key/value pair.

"foo" : "bar" : "Hello"
The key/value pair "bar" : "Hello" is nested inside the key/value pair "foo" : ... .
That's an example of a hierarchy in JSON data.

Chapter 2

Literature Review

Angular Web Development framework is becoming popular among Web developers everyday. Hundreds of third party modules (also called as libraries or packages) exists for Angular framework. All Angular modules are available as open–source modules on www.npmjs.org. Angular itself is dependent on NodeJS and Node Package Manager (npm), that is, Angular is distributed as a npm package. Users can install angular by following npm command (npm install angular). Similiar way Angular modules are hosted on www.npmjs.org for others to install and use, and also these packages are installed in the same way (npm install 'packagename'). This chapter takes survey at similar modules to the one implented by this thesis work, and examines the differences and drawbacks here.

2.1 Related Work

angular-nested-resource [8] is an angular module that helps working with RESTful models. It does not have any major dependencies and does not make use of ladash library like other many other libraries do. This implementation focuses on the very first version of Angular, that is, AngularJS. Modules implemented for AngularJS and Angular are completly incompatible with each other, as they have a very different architectural styles of their implementation. angular-nested-resource have implementation based in nested objects. It uses Promises to handle the asynchronus data.

Another popular implementation to RESTful Web Services for Angular is ngx-restangular [9]. This project is the follow-up of the original Restangular project.ngx-restangular does not support AngularJS, It only supports Angular 2+ versions. This module simplifies HTTP's common methods such as GET, POST, DELETE, and UPDATE requests. This module can be used for RESTful apps. This module changed its name from ng2-restangular to ngx-restangular because of implementation of Semantic Versioning by Angular's Core Team. NPM (Node Package Manager) name has also changed, and you can install latest version of this module by execuing npm install ngx-restangular in a command window.

angular2-rest[10] is another Angular 2 HTTP client to access the RESTful Web Services. It is implemented in Typescript. This is yet production ready, and it is still in experimental phase (apha phase).

ng2-rest-api[11] is a HTTP client to consume RESTful Web Services implemented for Angular 2+ versions. It is built on Angular2/http module in the Angular library with TypeScript. It is a REST API template for all api consumption in an angular application. It's an Angular2 rest template for all CRUD operations (Create, Read, Update and Delete operations). This module has not been published to npm (node package manager), so its only available to download on githhub, after downloding include it in your service folder. This module does not supports nesting of object, thus have only API of only few functions. Functions available are get(), create(), update(), delete() for HTTP's GET, POST, PUT, DELETE operations respectively. This module has a drawback that you can not configure application wide settings. HTTP headers field has to be set every time a request is made. angular-rest-service (proposed by this thesis work) overcomes this problem by providing application wide settings.

Chapter 3

Proposed Work

3.1 Problem Statement

This project aims to design and implement a module for Angular to consume RESTful Web Services. It makes use of Angular's http module to make request over HTTP to a RESTful Web Service and uses HTTP's standard methods such as GET, PUT, POST etc. This module, called hereafter '**angular-rest-service**', is made available as a npm package on www.npmjs.org.

3.2 Setting Up

Angular applications and Angular core itself are dependent on functionalities provided by third parties packages. To install Angular, NodeJS and Node Package Manager (npm) are required. Once NodeJS and npm are installed, Angular can be installed by using npm command line tools. 'npm install' requires an active Internet connection. Please set proxy using 'npm config' if you are behind proxy.

To install angular, execute the following command.

```
1       npm install angular
```

3.2.1 Installing angular-rest-service

Once npm is installed, This module can be installed by executing the following command from the command line (or Terminal in case of Linux). Before executing the following command, first go to the root directory of your Angular project.

```
1       npm install angular-rest-service
```

After successful execution of the above command, This module should be available into the node module directory. Above command only makes the module accessible to the current project. To make it accessible from anywhere, use the -g or –global flag.

```
1       npm install angular-rest-service -g
```

The -g parameter indicates the this module should be installed into global modules directory and should be accessible to all projects.

3.2.2 Importing angular-rest-service to your project

Import AngularRestServiceModule, AngularRestService and AngularRestServiceSettings into your root module. Once imported, add the AngularRestServiceModule to the import's array of the @NgModule metadata. This way all the features from the AngularRestService will accessible to the entire root module of the Angular Project. Add AngularRestService and AngularRestServiceSettings to the providers array of @NgModule metadata.

The updated root module file should look like this:

```
1 import { BrowserModule } from '@angular/platform-browser';
2 import { NgModule } from '@angular/core';
3 import { AppComponent } from './app.component';
4
5 import {AngularRestServiceModule, AngularRestService,
    AngularRestServiceSettings} from 'angular-rest-service';
6
```

```
@NgModule({
  declarations: [   AppComponent  ],
  imports: [
    BrowserModule,
    AngularRestServiceModule
  ],
  providers: [AngularRestService, AngularRestServiceSettings],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Before any request is made to a resource on the server machine, the base URL of the server machine need to be set. This Base URL is used for all requests except where custom base URL for the resource is specified. To set the base URL , import the AngularRestServiceSettings into the main component of root module. Add the AngularRestServiceSettings to the constructor of the component.

The updated main component should look like this:

```
import { Component, OnInit } from '@angular/core';
import { AngularRestServiceSettings } from 'angular-rest-service';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent implements OnInit {

      constructor( private settings: AngularRestServiceSettings) { }

      ngOnInit() {
            this.settings.setBaseUrl("http://base/url/api");
      }
}
```

3.3 angular-rest-service API

angular-rest-service module defines a simple inteface, to perform CRUD operations (Create, Read, Update, Delete) on the resources which are exposed via REST. This functions can be accessed by a singleton object which is available to all components that imports AngularRestServiceModule.

1. **list**(name: string, params?: any): any
   This function takes a resource name (eg. users, books, cources, categories etc.) and some optional parameters. This functions return an object (a list of users or list or books etc.) on which you can perform CRUD operations. To perform this operations, functions such as doGet() or doPost() made available, which uses corresponding HTTP method to perform neccessary operations on these resources.

```
1 this.rest.list('users').doGet().subscribe(result=>
2       /* process the result here */ );
```

2. **filterById**(id: number): any
   filterById() can be used after list(), which is useful when whole list needs to be filtered by given id. E.g. If among all users, a single user with id 23 needs to fetched, following single line code will do the job.

```
1 this.rest.list('users').filterById(23).doGet().subscribe(result=>
2       /* process the result here */ );
```

3. **filterById**(ids: Array): any
   Sometimes fetching several users is required. Overloaded filterById() also takes an array of numbers as input, which are a list of ids of users that needs to be fetched. E.g. If among all users, users with ids 23, 45, 12 needs to fetched, following single line code will do the job.

```
1 this.rest.list('users').filterById( [23, 45, 12] )
2       .doGet().subscribe(result=> /* process the result here */);
```

4. **doGet**(): Observable <any>
   The doGet() function performs a GET request and returns an Observable. As fetching data on a network can take time, these methods needs to asynchronous. Observable are a popular way to handle asynchronous events in Javascript. Observable have a subscribe() method with user defined function as parameter, this function is called as soon as a response received.

5. **doPost**(data : string): Observable <any>
   The doPost() function performs a POST request and returns an Observable.For doPost(), a single parameter is required which is a JSON object. This JSON object's format should adhere to the server's RESTful Web Service specification. Once this function is succesfully executed, a new record will be created on the server.

6. **doPut**(data : string): Observable <any>
   The doPut() function performs a PUT request and returns an Observable. For doPut(), a single parameter is required which is a JSON object. An id property with a valid value is mandatory to indentify a resource on the server. This JSON object's format should adhere to the server's RESTful Web Service specification. Once this function is succesfully executed, a record with specified id will be upadated on the server.

7. **doDelete**(): Observable <any>
   The doDelete() function performs a DELETE request and returns an Observable. The URL should locate a valid resource. After the successful execution of this function, returned Observable contains informations about whether the resource successfully deleted or not.

8. **doPatch**(data : string): Observable <any>

9. **addHttpHeader**(name : string, value : string)
   Use this function, when there is need to add a http header field in the current request. It should be a valid header field, and the user must have access the set a perticular header field. Some header fields (e.g. 'Accept-Encoding') are not allowed to be set. Only browsers can set these fields. A list of headers fields is maintained for each request. This function adds this header field to the list of already existing header fields. If the header field with same name already exists, its value is overwritten, else a new key–value pair is created. Apart from

this, there a global headers list stored in the AngularRestServiceSettings's object. These headers fields are included in each request to the server. If this function specifies a header field that already exists in the global headers, this local value is used instead for the current request.

10. **removeHttpHeader**(name : string, value : string)
    This function removes a header field from the list of headers fields, if it already exists. A separate copy of header fields is maintained for each request.

11. **setParameter**(name : string, value : any)
    The setParamter() is used to add a key–value pair in the request url. This function sets the parameter only for the current request. For example:

```
let users = this.rest.list("users");
users.setParameter("transform", 1);

//makes a /GET request to baseurl/users?transform=1
users.doGet().subscribe(
     response => {
        this.userlist = response.json().users;
     }
  );
```

If user want a parameter to be included in every request, another function is defined in the AngularRestServiceSettings.

3.4 Module Configuration

Module configuraton focuses on changing the behavior of the angular-rest-service. Sometimes the module needs to to congured, e.g. If every request should contaian authentication information in the HTTP header, such information can be set in the AngularRestServiceSettings. Module Configuration contains some functions that used to set application wide settings. AngularRestServiceSettings contains functions that are required to change the behavior of other functions, e.g setting the base url to which all calls should be made or a url paramter that every should made should contain.

- **setBaseUrl**(base : string)
  All request made to a RESTful Web Service have a unique URL or can also be called as Base URL which is common prefix for all resource addresses. The setBaselUrl() function can be used to set the base url which is used in all calls (except a few where Base URL is specified itself as a function parameter) to a RESTful Web Service.
  eg. http://abc.xyz/rest/api/ is valid base url.

- **getBaseUrl**() : string
  This function returns the currently set Base url. If the Base URL is not set, it simply returns 'undefined'.

- **addHttpHeader**(name : string, value : string)
  Use this function, when there is need to add a http header field for all requests. It should be a valid header field, and the user must have access the set a perticular header field. Some header fields (e.g. 'Accept-Encoding') are not allowed to be set. Only browsers can set these fields. A list of headers fields is maintained which are included in all requests. This function adds this header field (key–value pair) to the list of already existing global header fields. If the header field with same name already exists, its value is overwritten, else a new key–value pair is created. Apart from this, there a local headers list for each request. These headers fields are included in each request to the server. If this function specifies a header field that already exists in the local headers, this local value is used instead for the current request.

- **removeHttpHeader**(name : string, value : string)
  This function removes a header field from the list of global headers fields, if it already exists. Once this key–value pair is removed from this list, request made after this will not contain this key-value.

- **setGlobalParameter**(name : string, value : string)
  Global parameters are a way to maintain a list of key–value pairs, which are required in every request. For example, If request to the server needs to have an api–key to authenticate the user, it will cumbersome to independently add a parameter spearatly for every request. To solve this issue, just set the parameter using this function. Once parameter is successfully set, all request will include the this parameter.

3.5 Example application

Assume we have a server machine exposing RESTful Web Services. We want to fetch a list of all users. The following example shows how to get list of all users and set it to a local variable inside the Angular component. Import the angular-rest-service module and complete the basic set up (as explained in previous sections) to start using this module in your Angular project.

This example makes a GET request to /users expects a response in json format. This response is converted into a json object by using json() method.

```
import { Component, OnInit } from '@angular/core';
import { AngularRestService } from 'angular-rest-service';

@Component({
  selector: 'app-user-list',
  templateUrl: './user-list.component.html',
  styleUrls: ['./user-list.component.css']
})
export class UserListComponent implements OnInit {
  userlist;

  constructor(private rest: AngularRestService) { }

  ngOnInit() {
    let users = this.rest.list("users").doGet().subscribe(
      response => {
        this.userlist = response.json();
      }
    );
  }
}
```

Following example makes a GET request to /users/23.

```
let user = this.rest.list("users").filterById(23).doGet().subscribe(
      response => {
        //response.json() hold the user information with id 23;
```

```
4       });
```

POST method is used to create new resources on the server.
Following example makes a POST request to /users. The doPost() fuction expects a single parameter in json format.

```
1 let user= {
2       "username" : "Vijay",
3       "age" : 23,
4       "gender" : ''Male''
5 };
6
7 this.rest.list("users").doPost(user).subscribe(res =>
8       //res.json() contains info whether a new user created or not
9 );
```

To update a user information :
This will to PUT to /users
Note that id property is neccessary and should be a number.

```
1
2 let user = {
3       "id" : 23,
4       "username" : ''Mayur''
5       ''gender'' : ''Male''
6 };
7 this.rest.list("users").doPut(user).subscribe(res =>
8       //res.json() contains info whether a user udpated or not
9 );
```

To delete a user :
It makes DELETE request to /users/23.

```
1 this.rest.list("users").filterById(23).doDelete().subscribe(res =>
2       //res.json() contains info whether the user with id deleted or not
```

```
3 );
```

Chapter 4

Conclusion

This project designed and implemented an Angular module to access RESTful Web Services. This module makes communicating with a REST-based Web Service simpler and in fewer lines of code. It takes care of the authentication, erros, URL construction. It makes use of Angular's http module to make request over HTTP to a RESTful Web Service and uses HTTP's standard methods such as GET, PUT, POST etc. This module '**angular-rest-service**' is made available as a npm package on www.npmjs.org. This module can be installed in an Angular by executing '**npm install angular-rest-service**' from command line, which will download the latest version into the projects' nodule-modules directory. angular-rest-service suppports Basic (or Bearer) authentication, OAuth2 authentication methods. In the future work, additional authentication techniques can be added.

Bibliography

[1] Angular Team. Angular framework for single page applications. *Retrieved from https://angular.io*, Mar, 2015.

[2] Shahbaz Sherif. Key differnces in angularjs and angular 2. *Retrieved from https://www.technicaldiary.com/difference-angular-1-vs-angular-2*, July, 2016.

[3] Angular University. Angularjs vs angular - an in-depth comparison. *Retrieved from http://blog.angular-university.io/introduction-to-angular2-the-main-goals*, May, 2015.

[4] World Wide Web Consortium. Relationship to the world wide web and rest architectures. *Retrieved from https://www.w3.org/TR/2004/NOTE-ws-arch-20040211/#relwwwrest*, 11 February 2004.

[5] Tech.groups.yahoo.com. *Retrieved from http://tech.groups.yahoo.com/group/rest-discuss/message/5841*, 2013-02-07.

[6] SOAP Simple Object Access Protocol. Key differnces in angularjs and angular 2. *Retrieved from https://tools.ietf.org/html/draft-box-http-soap-00*, September 1999.

[7] RestApiTutorial.com. Rest api - resource naming. *Retrieved from http://www.restapitutorial.com/lessons/restfulresourcenaming.html*, Jan, 2010.

[8] Roy Peled. Angular module which helps with restful models. *Retrieved from http://ngmodules.org/modules/angular-nested-resource*, Mar, 2014.

[9] 2muchcoffeecom. Angular 2+ service that simplifies common get, post, delete, and update. *Retrieved from https://github.com/2muchcoffeecom/ngx-restangular*, Aug, 2016.

[10] Paldom. Angular2 http client to consume restful services. built with typescript. *Retrieved from https://github.com/Paldom/angular2-rest*, Apr, 2015.

[11] Ranjithprabhu K. A rest api template for all api consumption. *Retrieved from https://github.com/ranjithprabhuk/ng2-rest-api*, Nov, 2016.

[12] James; Mogul Jeffrey C.; Nielsen Henrik Frystyk; Masinter Larry; Leach Paul J.; Berners-Lee Tim Fielding, Roy T.; Gettys. Hypertext transfer protocol http/1.1. *RFC 2616*, July, 2016.