The CareFinder Client

CSCI 424 - Client/Server Development

Version: 2.0.0

Read this entire document before actually doing anything.

Prerequisite

If you have not read the project introduction document titled, “Introduction to the CareFinder Project”, please read that before reading this document.

Materials

To complete this part of the project, you will need:

1. ***A client-side build environment.*** This depends on your decision regarding the client-side platform you choose to develop. (See below.)
2. ***Postman.*** One of the nicest tools available for sending HTTP requests is called Postman (`https://www.getpostman.com/`). Postman is an extremely powerful application! You can save requests for later as well as group requests together to form test suites. The test suites can even include scripts that get run before/during/after the requests so you can determine if the requests returned the expected results. Think, Junit but for HTTP requests! *This project assignment assumes you will install and use Postman.*

Introduction

Do you have a smart phone? Do you have a computer with a web browser? Duh, right? Of course, we all do! Now for a slightly different but just as important set of questions… Have you seen a bad one? Have you seen a client that just does not look good or does not function as advertised? Conversely, have you used clients that look great and work great? I think the answer to all of these is, again, yes!

This points out how important clients actually are. In some respects (though we will see in the future why this is not the case), the client *IS* the program, at least from the perspective of the user. Most users have no idea of what goes on behind the scenes but expect is all to “just work”. Indeed, the client is important.

This part of the project focuses on the CareFinder client. Our client needs to make requests to the CareFinder server and display the data returned from the server.

But first, let us see how a CareFinder web service works…

Understanding How the Web Service Works

Our first step is to understand how the CareFinder web service should work. Web Services are data providers. You send it a request for data, it replies back with the data. The requests and responses are sent using HTTP through the use of a URI. The first part of this project helps you understand how to send requests to a RESTful web service.

Structure of the URI

Data is requested by sending the web service the proper URI. Your instructor has installed a working version of the CareFinder RESTful web service on a web server at `http://www.knautzfamilywi.com`. In addition, the web service is installed in a folder named `CareFinder-1.0.0`. Thus, the format of the URI's are as follows:

```
http://www.knautzfamilywi.com/CareFinder-1.0.0/api/endpoint/param1/param2
```

where ***`endpoint`*** is the name of an API endpoint that describes the type of the resource being requested and ***`param1`*** and ***`param2`*** are parameters that are passed to the endpoint to identify the resource. Some examples of URI's are shown below.

Key API Usage

The first step to using the CareFinder web service is to register for an API key. Many web services require a key be sent with each request to validate the fact that the requestor is a valid user. The key registration process works as follows:

1. **Request a CareFinder key.** The first step is to request a key from the CareFinder web service by sending a "key get" to the API endpoint using the proper URL. The URI for a key request is as follows:

   ```
   http://www.knautzfamilywi.com/CareFinder-1.0.0/api/key/get
   ```

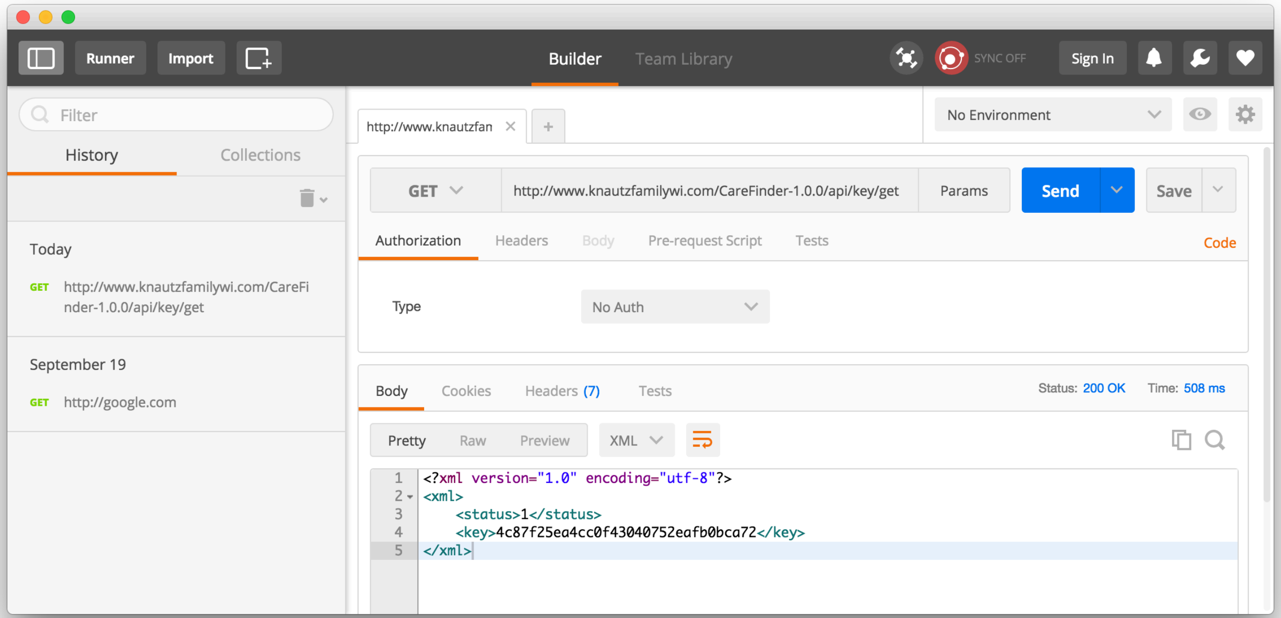
   FIGURE 1 below shows an HTTP GET request to the CareFinder key API at the URI shown above using Postman. To make this work, do the following:

   1. Make sure *GET* is selected as the method. If you click on the *GET* button, you will see that you can select any of the HTTP 1.1 methods. This allows you to send any HTTP 1.1 method you need!
   2. Type your URI in the URI box. FIGURE 1 shows that I used the URI shown above.
   3. If you have parameters for the request, click the *Params* button and fill the name/value pairs in the space provided.
   4. When you are ready to go, click the *Send* button.

   When the request is made, and the web service returns a response and Postman displays the returned result:

   ```
   <?xml version="1.0" encoding="utf-8"?>
   <xml>
       <status>1</status>
       <key>4c87f25ea4cc0f43040752eafb0bca72</key>
   </xml>
   ```

The `key` element contains the key you need to use in future requests.



GET http://www.knautzfamilywi.com/CareFinder-1.0.0/api/key/get

Type: No Auth

Status: 200 OK Time: 508 ms

```
<?xml version="1.0" encoding="utf-8"?>
<xml>
    <status>1</status>
    <key>4c87f25ea4cc0f43040752eafb0bca72</key>
</xml>
```

Figure 1: Example Postman request for an API key

As an aside, another easy way to send HTTP requests in general is to use a utility called **curl**. Curl is a *command line program* that transfers data between clients and servers using URI syntax. MacOS and Linux have curl installed as part of the operating system. Windows users can get curl from the curl web site and install it. The web site is `http://curl.haxx.se/`.

The command to use to send a request to the CareFinder key API looks something like this:

```
curl -i -H "Accept: application/xml" -X GET http://knautzfamilywi.com/ ↵
CareFinder-1.0.0/api/key/get
```

The result will look something like this:

```
HTTP/1.1 200 OK
Date: Wed, 18 Feb 2015 04:54:53 GMT
Server: Apache
Content-Length: 112
Vary: Accept-Encoding
Content-Type: application/xml; charset=utf-8

<?xml version="1.0" encoding="utf-8">
<xml><status>1</status><key>8cd8ddac1ebd1815fd695a9b863d90aa</key></xml>
```

2. **Save/Create the key.** Once you have obtained a key, the next step is to “save” the key with CareFinder. This is done by sending a “key create” to the API endpoint using the proper URL. Use Postman or curl to do this also. The URI would be:

   `.../api/key/create/8cd8ddac1ebd1815fd695a9b863d90aa/5/5`

   The ending `5/5` is used for permissions on the data. Do not worry about that for the moment. We will come back later to show how the two 5's are used.

URI Samples

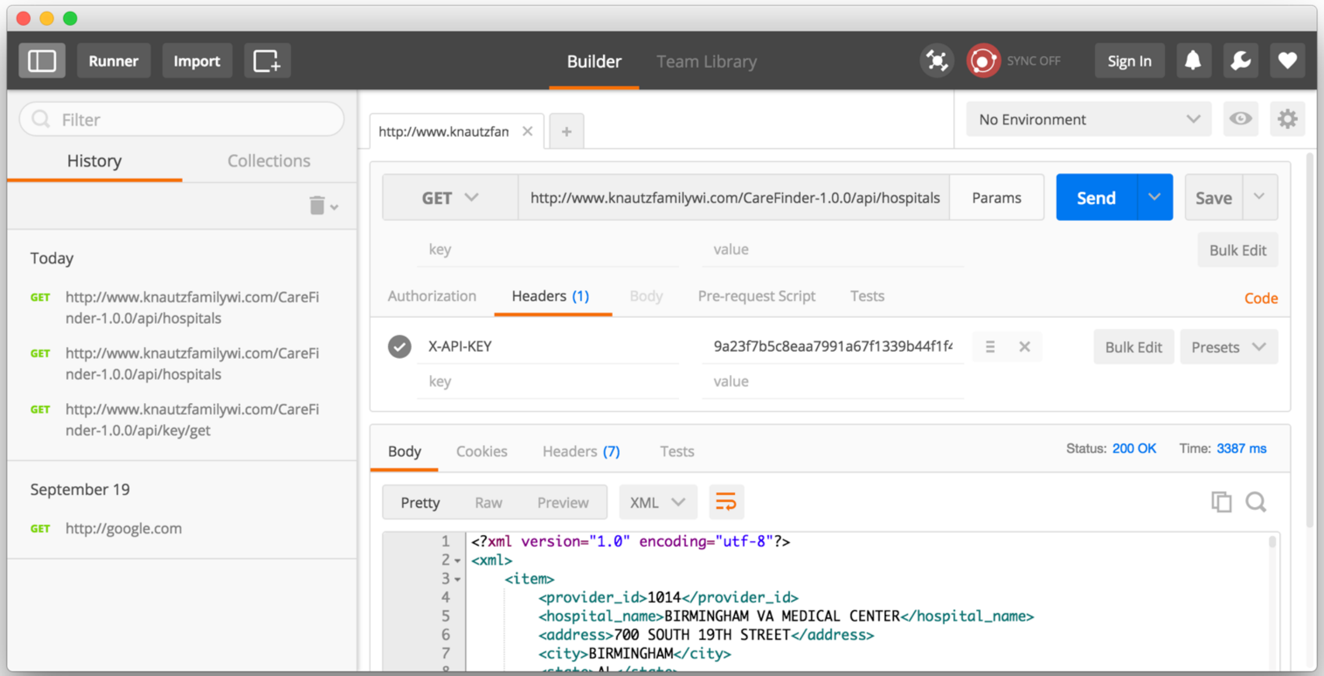
Now that you have your key, you can send requests to the CareFinder web service. Note that the API key is sent in the HTTP header using a user defined header field. In HTTP, all user-defined header fields start with `X-`. The name of the field is `X-API-KEY`.

The URI structure for the web service follows the typical structure but each requests a different type of data. The table below shows the URI's and their corresponding description.

URI	Description
`…/api/hospitals`	Get all hospitals
`…/api/hospitals/id/`*`number`*	Get a hospital based on its unique identifier
`…/api/hospitals/city/`*`name`*	Get a hospital based on city name
`…/api/hospitals/state/`*`name`*	Get a hospital based on state name
`…/api/hospitals/county/`*`name`*	Get a hospital based on county name
`…/api/hospitals/citystate/`*`name`*`/`*`name`*	Get a hospital based on city/state name combination
`…/api/hospitals/name/`*`name`*	Get a hospital based on the hospital's name

Again, to get a feel for the web service, the easiest way to do this is via Postman or curl with Postman being the preferred metod.

For Postman, you must add the key in the header by using the *Headers* tab just below the URI input area. Click the *Headers* tab and fill in the key (`X-API-KEY`) and value (your API key) in the provided fields. FIGURE 2 shows this.



Today
GET http://www.knautzfamilywi.com/CareFinder-1.0.0/api/hospitals
GET http://www.knautzfamilywi.com/CareFinder-1.0.0/api/hospitals
GET http://www.knautzfamilywi.com/CareFinder-1.0.0/api/key/get
September 19
GET http://google.com

GET http://www.knautzfamilywi.com/CareFinder-1.0.0/api/hospitals

X-API-KEY 9a23f7b5c8eaa7991a67f1339b44f1f4

Status: 200 OK Time: 3387 ms

```
<?xml version="1.0" encoding="utf-8"?>
<xml>
    <item>
        <provider_id>1014</provider_id>
        <hospital_name>BIRMINGHAM VA MEDICAL CENTER</hospital_name>
        <address>700 SOUTH 19TH STREET</address>
        <city>BIRMINGHAM</city>
```

Figure 2 - Postman with X-API-KEY header

The following is a curl command that retrieves hospitals based on the name of the state in which they reside:

```
curl -i -H "Accept: application/xml" -H "X-API-KEY: 9a23f7b5c8eaa7991a67f1339b44f1f4" -X
GET http://knautzfamilywi.com/CareFinder-1.0.0/api/hospitals/state/wi
```

3. **Try the URI's.** Try each of the URI's in the table above using curl. Look at the data that is returned. Note that the data is in XML. How does the CareFinder web service know to return the data in XML?

The Assignment

Your assignment is to develop your CareFinder client. You are free to choose the type of client you want to write. However, your client must have a way to send each of the GET requests shown in the table above using the URI's shown in the table above. In addition, your client must display the returned XML in a user-friendly manner.

Do not try to write the entire client all at once! Start small and work up from there. Start by making a client that has a hard-coded GET request and simply displayed the returned data as text. Work up from there.

Also, you must take into account how a user is going to use this client. This is not necessarily a trivial task. How is the user going to request the information? How are you going to display the information? These factors will be taken into account in the assessment of the client.

Client Demo

You must demo your client for your instructor. The demo will be used to assess the "Maturity" component of the grading rubric (see below).

You can make a video of your client that shows all of the features working or you can demo the client in-person. If you choose to do an in-person demo, schedule the demo with your instructor within the week following the due date. An in-person demo will be done virtually via one of the video conference tools (Zoom, WebEx, Teams, etc.)

Submission

The following are the submission guidelines for this project:

1. (Required) Whichever platform you used, submit the entire project folder so that I can load the project on my system. Submit the project folder as a zip file.
2. (Required) Submit an "implementation document". This document is a "summary document" that describes exactly what you implemented. Describe your "trials & tribulations" that you encountered as you developed your client. Also describe any bugs that remain.
3. (Optional) If you chose to do a demo video, submit the video.

Grading

The following rubric will be used to assess the client:

	Exemplary	**Proficient**	**Limited**	**Unsatisfactory**	**Did not complete**
Maturity **4 pts. possible**	4 points	3 points	2 points	1 point	0 points
	The client is well designed, looks nice and functions smoothly from a user experience point of view.	The client is mostly well designed, looks nice and mostly functions smoothly from a user experience point of view.	The client design is mostly underdeveloped, look somewhat nice and does not functions smoothly from a user experience point of view.	The client design is significantly underdeveloped, does not look nice and does not functions smoothly from a user experience point of view.	No submission.
Implemented endpoints **7 pts. possible**	7 points	5 or 6 points	3 or 4 points	1 or 2 points	0 points
	All seven endpoints implemented.	Five to six endpoints implemented.	Three to four endpoints implemented.	One to two endpoints implemented.	No submission.

Ω