List of Required Services
CSE445/598 Service Development Assignment
Dr. Yinong Chen

Fall 2023

This document lists service requirements that have a reasonable functionality and adequate level of complexity. These service requirements are used as the required services to be developed in Project 3 (Assignments 3 and 5). The services that you develop can be used in Project 5 (final project) as its components. Thus, you and your team should define a Web application in Project 3 and choose related services around your application. If your services cannot be used in your application development in Project 5, you will have to develop new services in Project 5, which can significantly increase your team's workload in Project 5.

Some of the service requirements listed are related. Combination of the related services can form a coherent application. Some of the service requirements are independent of each other.

Some of the service requirements listed refer to external services. These services **may no longer be free or no longer exist**. You need to test any services before using them. You can use Visual Studio WCF Client Test Tool for testing any WSDL service, or use web browser for testing any RESTful services, that you found before using the services in your application. You can search for alternative services or use different services.

Contents

Service 1. WebDownloading 2

Service 2. WordFilter 3

Service 3. Stemming 3

Service 4. WordCount 3

Service 5. Top10ContentWords 3

Service 6. Stock Build and Quote (It can be two services) 4

Service 7. WsdlAddress 4

Service 8. WsdlDiscovery 4

Service 9. WsOperations 5

Service 10. WsHashOperations 5

Service 11. WsTesting 5

Service 12. GroupTesting 6

Service 13. NewsFocus 6
Service 14. Storage Service 6
Service 15. Weather Service 6
Service 16. Solar Energy Service 7
Service 17. Wind Energy Service 7
Service 18. Crime Data Service 8
Service 19. Natural Hazards Service 8
Service 20. Number2Words 8
Service 21. Find the Nearest Store 9
Service 22. Wrap an API to create a new service 9
Service 23. The U.S. Government Data Services 10
Service 24. ChatGPT APIs 10
Service 25. VoiceCommand // Amazon Alexa Service 13
Service 26. VoiceInvocation // Amazon Alexa Service 13

Service 1. WebDownloading

Description: It takes a URL in string as input and returns the content at the given URL.

Operation: string WebDownload(string url)

Input: A webpage url in string.

Output: A string the content at the given URL. You can also save the string into a text file.

Note: You can follow lecture slides using the following commands to implement the service:

```
WebClient channel = new WebClient();            // create a channel
byte[] abc = channel.DownloadData(completeUri); // return byte array
Stream strm = new MemoryStream(abc); // convert to mem stream
DataContractSerializer obj = new DataContractSerializer(typeof(string));
string randString = obj.ReadObject(strm).ToString();//convent to string
```

The function `channel.DownloadData()` downloads the website as a byte array. A byte array contains not only printable characters, but also non printable characters. If you want to download printable characters only, you can simply use:

```
WebClient channel = new WebClient();            // create a channel
string randString = channel.DownloadString(completeUri); //return string
```

Service 2. WordFilter

Description: Analyze a string of words and filter out the function words (stop words) such as “a”, “an”, “in”, “on”, “the”, “is”, “are”, “am”, and any words that are not meaningful to be counted at the top words in search. You must also filter out the XML element tag names and attribute names.

Operation: string WordFilter(string str)

Input: A string.

Output: A string with the stop words removed.

Service 3. Stemming

Description: Analyze a text file or a string containing words and replace each of the inflected or derived words to their stem or root word. For example, “friend”, is the stem of “friends”, “friendship”, “friendships”. These derived words should be replaced by the stem word “friend”. This service can help finding useful keywords or index words in information processing and retrieval. https://en.wikipedia.org/wiki/Word_stem

Operation: string Stemming(string)

Input: A text file or a string words.

Output: The string of the inflected or derived words replaced by their stem words.

Service 4. WordCount

Description: Analyze a large text file and return a string of JSON data. The key is a word appear in the file and the value is the number of occurrences of the word in the file.

Operation: string WordCount(file)

Input: A large file containing string or words.

Output: A JSON data or a string of JSON data consisting of key-value pairs. The key is a word appearing in the file and the value is the number of occurrences of the word in the file.

Service 5. Top10ContentWords

Description: Analyze a text file a webpage at a given url and return the ten most-frequently occurred “content” words in the webpage. Return the words in descending order of their appearing frequencies. Content words do not include the stop words and the element tag names and attribute names quoted in angle brackets < … >, if the string represents an XML page or HTML source page, as well as other words that do not represent the content of the webpage. The ranking of the words should be used based on the string that have been processed by stemming service.

Operation: string[] Top10ContentWords(string url)

Input: A webpage url in string.

Output: An array of strings that contains the ten most-frequently occurred words in descending order of their frequencies. The required words must not be element tag name or attribute name of XML page or HTML source page.

Alternative input and output: The input can be a text file or string of words. The output can be a string of words separated by a separator.

Service 6. Stock Build and Quote (It can be two services)

Description: Download a stock price page, for example:

http://nasdaq.com, https://polygon.io/, http://iexcloud.io

http://www.wsj.com/mdc/public/page/2_3024-NYSE.html

https://www.wsj.com/market-data/stocks

Operation1 calls a stock service or API and process the data to provide the required output. Or, it analyzes the online data to obtain the open price of each stock symbol. Save the (symbol price) pairs into a file.

Note, this operation should, but does not have to work for all stock pages. For workload purpose, it is fine for this operation to work for the given NYSE page only.

Operation2 (optional) reads the file generated from operation1 and returns the stock price of the given stock symbol

Operation1: string Stockbuild(string sourceURL); // return file name to the (symbol price) pairs.

Operation2: string Stockquote(string symbol); // return the stock price of the given stock symbol in string or float.

Service 7. WsdlAddress

Description: Analyze a webpage and return all WSDL addresses in that webpage in an array of strings. The WSDL address can have these formats: xxx.wsdl (e.g. developed Java), or ?wsdl (e.g., .php?wsdl, .svc?wsdl and .asmx?wsdl).

Operation: string[] getWsdlAddress(string url)

Input: A webpage url.

Output: An array of strings that contains WSDL addresses in the given webpage.

Service 8. WsdlDiscovery

Description: Call Google or Bing search engine APIs, using keywords such as .wsdl, .php?wsdl, .svc?wsdl and .asmx?wsdl, to discover the Web pages that contain WSDL addresses, and follow the address to discover WSDL files. You can find Google search APIs in Google code, and you can find Bing search APIs in MSDN library: Bing SOAP Services: http://msdn.microsoft.com/en-us/library/cc966738.aspx or http://msdn.microsoft.com/en-us/library/dd251056.aspx. For example, when I use these ".wsdl", ".php?wsdl", ".svc?wsdl" ".asmx?wsdl", as search keywords, I find pages with the required wsdl files, including the one below, which has a long list of WSDL files:
http://data.serviceplatform.org/servicefinder/sf-wsdls.list.sorted
Notice that you need to analyze the addresses returned from the search engine and remove those that are not wsdl addresses. The service should return only those addresses that are wsdl addresses. If you read service 7, you can see that the addresses will be given to another service to discover the operations in the wsdl file.

Operation: string[] WsdlDiscovery(string keyWords)

Input: A sting of keywords related wsdl address format

Output: An array of strings that contains WSDL addresses discovered through the keywords search engines.

Service 9. WsOperations

Description: Analyze a WSDL file in XML format and return operation names and their corresponding input parameter and return types.

Operation: string[] getWsOperations(string url)

Input: The url address that points to an WSDL file

Output: An array of strings. Each of the array elements contains return the type, operation name, and parameter types

Service 10. WsHashOperations

Description: Analyze a WSDL file in XML format and return operation names and their corresponding input parameter types and return type in a Hashtable or a similar data structure, for example, Dictionary<object, object>. The format should look like: return type, Input parameter1, parameter2, etc.

Operation: Dictionary<object, object> WsHashOpertions(string wsdlUrl)

Input: The wsdlUrl address that points to an WSDL file

Output: An object of Dictionary<object, object> that contains operation names, input and output parameter types.

Service 11. WsTesting

Description: Given a WSDL address, call all the service operations in the service. The service should first obtain the list of operations and generate their input/output values, for example, calling the service "WsHashOperations", and then, invoke each operation. The return values must be encoded into an aggregate type, for example, an XML, string, and array types.

Operation: UserType WsTesting(string wsdlUrl)

Input: The wsdlUrl address that points to an WSDL file.

Output: UserType that wraps the return result.

Service 12. GroupTesting

Description: Given an array of WSDL addresses, call the service operations of each service. These services are supposed to implement the same function. The testing service will validate whether the opeartions indeed generate the same output for the same input.

Operation: boolean GroupTesting(string[] wsdlUrl,)

Input: The wsdlUrl addresses that point to an WSDL files.

Output: a true or false value indicating whether the services give the same output or not.

Service 13. NewsFocus

Description: Find news about specific topics, for example, find all (as many as possible) news articles about ASU (Arizona State University).

Operation: string[] NewsFocus(string[] topics)

Input: a list of topics or key words

Output: A list of URLs in which the given topics are reported.

Service 14. Storage Service

Description: Create a service that can load a local file or a file at a URL (implement one of these two options) and store the file into the server. A URL will be returned. The returned URL can be used for retrieving the file from the server.

Operation: string Storefile(string fileNameOrUrl)

Input: file name with local path or a URL

Output: URL of the file in the server.

Hint: read: https://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.fileupload.saveas(v=vs.110).aspx

Service 15. Weather Service

Description: Create a 5-day weather forecast service of zipcode location based on the Search for free weather services or APIs, such as:
national weather service at:
http://graphical.weather.gov/xml/SOAP_server/ndfdXMLserver.php?wsdl.

Operation: string[] Weather5day(string zipcode) // You may return JSON format.

Input: a U.S. zipcode

Output: An array (or list) of strings, storing 5-day weather forecast for the given zipcode location.

Hint, the given national weather service has an operation: `LatLonListZipCode(zipcode);` It returns an XML file containing the latitude and longitude of the zipcode location.
For example, `LatLonListZipCode(85281)` will return

```
"<?xml version='1.0'?>
<dwml version='1.0' xmlns:xsd='http://www.w3.org/2001/XMLSchema'
xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
xsi:noNamespaceSchemaLocation='http://graphical.weather.gov/xml/DWMLgen/schema/DWM
L.xsd'><latLonList>33.4357,-111.917</latLonList></dwml>"
```

Then, you can use latitude and longitude to call the operation to obtain the 5-day forecast. The available operations in this service is listed and explained at:
http://graphical.weather.gov/xml/SOAP_server/ndfdXMLserver.php

You must discover what is available and how the service can be used. You can also use other weather services for creating your service.

Service 16. Solar Energy Service

Description: Create a service that returns the annual average sunshine index of a given position (latitude, longitude). This service can be used for deciding if installing solar energy device is effective at the location.

Operation: decimal SolarIntensity(decimal latitude, decimal longitude)

Input: latitude and longitude

Output: a number reflecting the annual average solar intensity at the location.

You must discover what is available and how the discovered service can be used.

Service 17. Wind Energy Service

Description: Create a service that returns the annual average wind index of a given position (latitude, longitude). This service can be used for deciding if installing windmill device is effective at the location.

Operation: decimal WindIntensity(decimal latitude, decimal longitude)

Input: latitude and longitude

Output: a number reflecting the annual average wind intensity at the location.

You must discover what is available and how the discovered service can be used.

Hint: you may find solar and wind data from sources like:
https://eosweb.larc.nasa.gov/cgi-bin/sse/global.cgi?email=skip@larc.nasa.gov

https://eosweb.larc.nasa.gov/sse/global/text/10yr_wspd50m

Service 18. Crime Data Service

Description: Create a service that returns certain crime data for a given location. The service can use data published by police crime reports and statistics.

Operation: int crimedata (decimal latitude, decimal longitude) // or zipcode

Input: latitude and longitude

Output: a number or stream reflecting the number of a given crime in the area of the given location.

You must discover what data are available and how the data can be used.

Hint: For example, get data from: http://geodataservice.net/DemographicsAPI.aspx

or retrieve the data from web APIs, such as https://azure.geodataservice.net/GeoDataService.svc/GetUSDemographics?includecrimedata=true&zipcode=33444 using web request classes

Service 19. Natural Hazards Service

Description: Create a service that returns the natural hazards (Tsunamis, earthquake, volcanoes) index of a given position (latitude, longitude). This service can be used for building decision and insurance premium.

Operation: decimal NaturalHazards(decimal latitude, decimal longitude)

Input: latitude and longitude

Output: a number reflecting the natural hazards at the location.

Hint: getting data from NOAA (National Geophysical Data Center): http://www.nesdis.noaa.gov/, http://maps.ngdc.noaa.gov/, and http://gosic.org/

In the site, you can search keyword API. You must discover what is available and how the service can be used.

Service 20. Number2Words

Description: Convert a number, e.g., a phone number, into an easy-to-remember character/digit string. All valid words, e.g., helloClass are easy to remember. Other commonly used character/digit strings, such as CSE100, CSE445, and SCIDSE, are easy to remember strings too.

Operation: string Number2Words(string number)

Input: a string of digits (a string consisting of digits).

Output: a character/digit string, which are easy-to-remember and each character corresponds to the input digits by the definition of a standard phone touch key system:

0 = none
1 = none
2 = ABC
3 = DEF
4 = GHI
5 = JKL
6 = MNO
7 = P(Q)RS
8 = TUV
9 = WXYZ

If the input contains 0 or 1, these two characters will remain 0 and 1. All other digits must be translated into letter, unless the digits make better sense, e.g., CSE445.

In order to implement the service, you must have a set of easy to remember words of your own, such as CSE445598, ASU, SCIDSE. You also need a dictionary. You could use the Web2String service to read a set of words from a public dictionary service. An example of the application is, you can publish your number like this: 1800-FLOWERS

Service 21. Find the Nearest Store

Description: Use an existing online service or API to find the provided storeName closest to the zipcode and return the address. If no store is found, return an error message. (Optional: if the store is further than 20 miles, from the zipcode, return a "no stores within 20 miles" message). You may find APIs that return store list from site such as Foursquare.com, http://www.programmableweb.com, or Yelp.com (doc: https://www.yelp.com/developers/documentation/v3/business_search).

Operation: Operation: string findNearestStore(string zipcode, string storeName)

Input: two strings

Output: string message

Service 22. Wrap an API to create a new service

Explore Useful Services and APIs that you can call in your services, for example:

Google's APIs

Remote methods

APIs from http://www.programmableweb.com
http://www.programmableweb.com/search/top%20mashups

Service 23. The U.S. Government Data Services

Explore the government service repository and find useful services to build your services.

http://www.usda.gov//wps/portal/usda/usdahome?navid=USDA_DEVELOPER

Service 24. ChatGPT APIs

This option should be a set of APIs (services) to be taking by the entire team to develop a chatbot application.

Description: Explore the APIs offered by OpenAI for developing your questioning and answering services and TryIt page. The resources available can be found from OpenAI related websites. For example:

https://platform.openai.com/docs/guides/gpt

https://www.makeuseof.com/chatgpt-api-complete-guide/

https://openai.com/blog/introducing-chatgpt-and-whisper-apis

https://medium.com/apis-with-valentine/getting-started-with-the-chatgpt-api-4d884b20f6d0

Having obtained basic understanding through reading, you can watch this overall video: https://www.youtube.com/watch?v=LX_DXLlaymg&ab_channel=AdrianTwarog

Operation: string getAnswer(string question)

Input: A string representing a question or a list of keywords

Output: A string that answers the question or present the text related to the list of keywords.

Sample code calling ChatGPT RESTful APIs:

(1) Get answer to a question:

```
<script>
  async function callCHATGPT() {
    function printMessage(message) {
      var responseText = document.getElementById("chatgpt-response");
      var index = 0;
      // Create a timer
      var interval = setInterval(function() {
          responseText.innerHTML += message[index];
          index++;
          // Clear timer after printing
          if (index >= message.length) {
            clearInterval(interval);
          }
        },
            150);  // Print a word with 150ms interval
        }
        var xhr = new XMLHttpRequest();
        var url = "https://api.openai.com/v1/completions";
        xhr.open("POST", url, true);
        xhr.setRequestHeader("Content-Type", "application/json");
        xhr.setRequestHeader("Authorization", "Bearer keykeykeykeykeykeykeykey…");
                        // you need to register as a developer to obtain your key
        xhr.onreadystatechange = function() {
          if (xhr.readyState === 4 && xhr.status === 200) {
            var json = JSON.parse(xhr.responseText);
            var response = json.choices[0].text;
            // Print the return text from ChatGPT into the textbox
            var responseText = document.getElementById("chatgpt-response");
            var index = 0;
            // Create a timer and print words with 50ms interval.
            var interval = setInterval(function() {
                responseText.innerHTML += response[index];
                index++;
                // Clear timer after printing
                if (index >= response.length) {
                  clearInterval(interval);
                }
              },
              50); // Print a word with 50ms interval
          }
        };

        var data = JSON.stringify({
          "prompt": document.getElementById("chat-gpt-input").value,
          "max_tokens": 2048,
          "temperature": 0.5,
          "top_p": 1,
          "frequency_penalty": 0,
          "presence_penalty": 0,
          "model": "text-davinci-003"
        });
        console.log(data);
        await printMessage('Thinking, please wait ......');
        await xhr.send(data);
      }
</script>
```

(2) Ask ChatGPT about weather, which calls a dummy weather service

```
import openai
import json
# Example dummy function hard coded to return the same weather
# In production, this could be your backend API or an external API
def get_current_weather(location, unit="fahrenheit"):
    """Get the current weather in a given location"""
    weather_info = {
        "location": location,
        "temperature": "72",
        "unit": unit,
        "forecast": ["sunny", "windy"],
    }
    return json.dumps(weather_info)

def run_conversation():
    # Step 1: send the conversation and available functions to GPT
    messages = [{"role": "user", "content": "What's the weather like in Boston?"}]
    functions = [
        {
            "name": "get_current_weather",
            "description": "Get the current weather in a given location",
            "parameters": {
                "type": "object",
                "properties": {
                    "location": {
                        "type": "string",
                        "description": "The city and state, e.g., San Francisco, CA",
                    },
                    "unit": {"type": "string", "enum": ["celsius", "fahrenheit"]},
                },
                "required": ["location"],
            },
        }
    ]
    response = openai.ChatCompletion.create(
        model="gpt-3.5-turbo-0613",
        messages=messages,
        functions=functions,
        function_call="auto",  # auto is default, but we'll be explicit
    )
    response_message = response["choices"][0]["message"]

    # Step 2: check if GPT wanted to call a function
    if response_message.get("function_call"):
        # Step 3: call the function
        # Note: the JSON response may not always be valid; be sure to handle errors
        available_functions = {
            "get_current_weather": get_current_weather,
        }  # only one function in this example, but you can have multiple
        function_name = response_message["function_call"]["name"]
        fuction_to_call = available_functions[function_name]
        function_args = json.loads(response_message["function_call"]["arguments"])
        function_response = fuction_to_call(
            location=function_args.get("location"),
            unit=function_args.get("unit"),
        )

        # Step 4: send the info on the function call and function response to GPT
        messages.append(response_message)  # extend conversation with assistant's reply
        messages.append(
            {
                "role": "function",
                "name": function_name,
                "content": function_response,
            }
        )  # extend conversation with function response
        second_response = openai.ChatCompletion.create(
            model="gpt-3.5-turbo-0613",
            messages=messages,
        )  # get a new response from GPT where it can see the function response
        return second_response
print(run_conversation())
```

The following two services are related to Amazon Alexa Service (AVS). You can learn AWS from the following these simple steps:

1. Create an AWS account. The [AWS free tier]covers a wide majority of services before any charges are incurred.
2. Create a [developer.amazon.com] account
3. Create your first very easy AVS skill: https://github.com/alexa/skill-sample-nodejs-fact

You can see an Alex service example (video) here: http://venus.sod.asu.edu/VIPLE/Videos/HumanoidVoiceCommand.mp4

Service 25. VoiceCommand // Amazon Alexa Service

Description: Assume that you have a list commands (at least 20 different words), you read (voice input) one of the given words and your service will return the word in string. You can use Alexa skill to implement your service.

Operation: string VoiceCommand(voice);

Note: You need to have AWS Alexa prior knowledge for implementing this service.

Service 26. VoiceInvocation // Amazon Alexa Service

Description: For your TryIt page, you replace the button click by voice activation. For example, if your TryIt page have two buttons (Add to Cart and Continue Shopping) to click, you read the text on the button to the page that button links to. You can use Alexa skill to implement your service.

Operation: ClickLink VoiceInvocation(voice);

Note: You need to have AWS Alexa prior knowledge for implementing this service.