Working with Web Services - Week 1

Summary

- HTTP (Hypertext Transfer Protocol)
- SOAP (Simple Object Access Protocol) vs. REST (Representational State Transfer)
- Web Services
- API Usage Exercise
- Homework (due in second week)

HTTP (Hypertext Transfer Protocol)

The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems.

HTTP is a TCP/IP based communication protocol, that is used to deliver data (HTML files, image files, query results, etc.) on the World Wide Web. The default port is TCP 80, but other ports can be used as well. It provides a standardized way for computers to communicate with each other. HTTP specification specifies how clients' request data will be constructed and sent to the server, and how the servers respond to these requests.

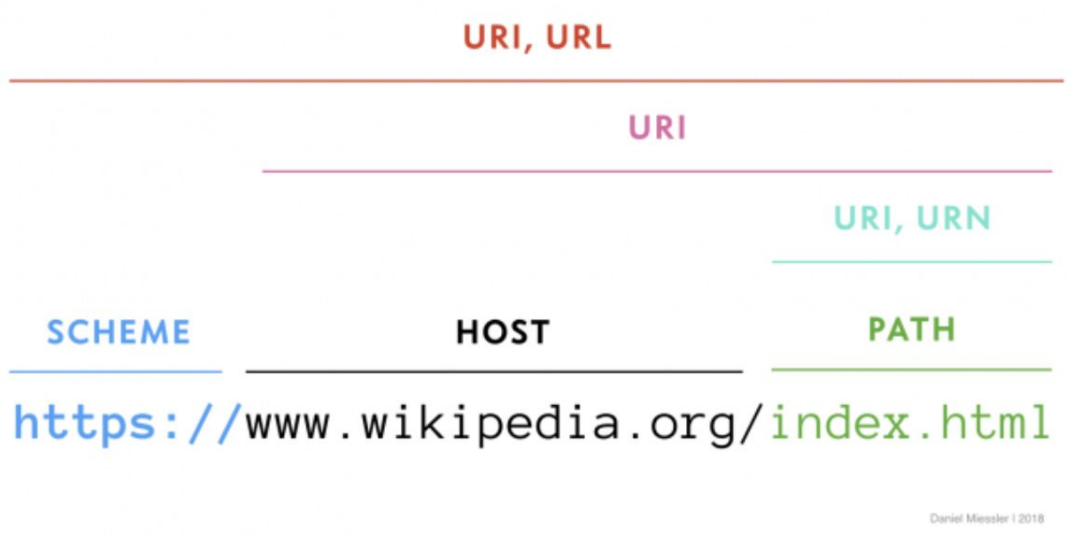
The HTTP protocol is a request/response protocol based on the ***client/server based*** architecture where web browsers, robots and search engines, etc. act like HTTP clients, and the Web server acts as a server.

URI, URL & URN disambiguation:

- **URI:** Uniform Resource Identifier
- **URL:** Uniform Resource Location
- **URN:** Uniform Resource Name

Based on the RFC 3986, *a URI can be further classified as a locator, a name, or both*. This basically means that a URI can be both a URL or a URN.

A ***URN*** is the name of the resource without a way to locate it (e.g. *index.html* from below, without the server name)
A ***URL*** is the name of a resource with the way to locate it (e.g. the entire URI from below).



HTTP Methods (“Verbs”):

GET	used to **receive information** about a resource or collection
POST	used to **create** a resource
PUT	used to **update** a resource
PATCH	used to **partially update** a resource
DELETE	used to **delete** a resource or a collection

Do it yourself:
Open a terminal and play around with the curl commands from here: https://docs.postman-echo.com/ . *Note: add -v to any command to see the headers*

Status Codes:
Status codes are issued by a server in response to a client's request made to the server. All HTTP response status codes are separated into five classes (or categories). The first digit of the status code defines the class of response.

Status Class	**Description**	**Examples**
1xx	***Informational*** - The request was received, continuing process	**100** Continue

2xx	***Successful*** - The request was successfully received, understood, and accepted	**200** OK **201** Created **204** No Content
3xx	***Redirection*** - Further action needs to be taken in order to complete the request	**301** Moved Permanently **304** Not Modified
4xx	***Client Error*** - The request contains bad syntax or cannot be fulfilled	**400** Bad Request **401** Unauthorized **403** Forbidden **404** Not Found **409** Conflict
5xx	***Server Error*** - The server failed to fulfill an apparently valid request	**500** Internal Server Error **501** Not Implemented **503** Service Unavailable

Resources

An overview of HTTP,
https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview

SOAP (Simple Object Access Protocol) vs. REST (Representational State Transfer)

1. SOAP

- relies exclusively on XML to provide messaging services and was designed to replace binary messaging technologies; the XML messaging that SOAP employs works better over the Internet;

- designed to support expansion, so it has all sorts of other acronyms and abbreviations associated with it, such as WS-Addressing, WS-Policy etc.; you can find a list of these standards on Web Services Standards;

- highly extensible, but you only use the pieces you need for a particular task; for example, when using a public Web service that's freely available to everyone, you really don't have much need for WS-Security;

- using XML to make requests and receive responses in SOAP can become extremely complex; in some programming languages, you need to build those requests manually;

- building requests manually is problematic because SOAP needs a specific format (and accepts no other variation of it);

- Web Services Description Language (WSDL) - SOAP file that provides a definition of how the Web service works, so that when you create a reference to it, the IDE can completely automate the process;

- the difficulty of using SOAP depends on the language you use;

- built-in error handling - the response contains error information that you can use to fix the problem; error reporting provides standardized codes so that it's possible to automate some error handling tasks in your code.

2. REST

- a more lightweight alternative: instead of using XML to make a request, REST relies on a simple URL in many cases. In some situations you must provide additional information in special ways, but most Web services using REST rely exclusively on obtaining the needed information using the URL approach. REST can use four different HTTP 1.1 verbs (GET, POST, PUT, and DELETE) to perform tasks;

- unlike SOAP, REST doesn't have to use XML to provide the response; you can find REST-based Web services that output the data in Command Separated Value (CSV), JavaScript Object Notation (JSON) and Really Simple Syndication (RSS). The point is that you can obtain the output you need in a form that's easy to parse within the language you need for your application;

3. SOAP vs. REST

SOAP is definitely the heavyweight choice for Web service access. It provides the following advantages when compared to REST:

- language, platform and transport independent (REST requires use of HTTP);
- works well in distributed enterprise environments (REST assumes direct point-to-point communication);
- standardized;
- provides significant pre-build extensibility in the form of the WS* standards;
- built-in error handling;
- automation when used with certain language products;

REST is easier to use for the most part and is more flexible. It has the following advantages when compared to SOAP:

- no expensive tools require to interact with the Web service;
- smaller learning curve;
- efficient (SOAP uses XML for all messages, REST can use smaller message formats);
- fast (no extensive processing required);
- closer to other Web technologies in Design Philosophy;

Some people try to say that one process is better than the other, but this statement is incorrect. Each protocol has definite advantages and equally problematic disadvantages. You need to select between SOAP and REST based on the programming language you use, the environment in which you use it, and the requirements of the application. Sometimes SOAP is a better choice and other times REST is a better choice. In order to avoid problems, you really do need to chart the advantages and disadvantages of a particular solution in your specific situation.

Resources

Understanding SOAP and REST Basics And Differences, https://smartbear.com/blog/test-and-monitor/understanding-soap-and-rest-basics/

Web Services

Web services are self-contained, modular, distributed, dynamic applications that can be described, published, located, or invoked over the network to create products, processes, and supply chains. These applications can be local, distributed, or web-based. Web services are built on top of open standards such as TCP/IP, HTTP, Java, HTML, and XML.

A web service is, therefore, any service that:

- Is available over the Internet or private (intranet) networks
- Uses a standardized XML messaging system
- Is not tied to any operating system or programming language
- Is self-describing via a common XML grammar
- Is discoverable via a simple find mechanism

Resources

Web Services Explained, https://www.service-architecture.com/articles/web-services/web_services_explained.html

API Usage Exercise

Do it yourself:
Create a program (Python, Node Js or any language you like) that sends HTTP requests to one of the following APIs (or any other you would like):

- https://www.virustotal.com/en/documentation/public-api/
- http://www.splashbase.co/api#images_random
- https://api.random.org/api-keys

Homework (due in second week)

Create an application that provides results from at least three different web services. Use or implement a system that allows (at least): monitoring of running web services, concurrent number of requests, logging requests/responses.
Observation*: The third web service must use the results from previous two.*

Additional Information:

- the application should have both a client side and a server side, which means that a *minimal* web interface is required*;*
- the web interface will be provider by the server and it will be used to send the requests
- the communication logic between web services will be implemented in the web server component;
- ***logging requests/responses***: each call will log at least the following information: request, response, latency (response time)
- ***monitoring of running web services***: a ***/metrics*** API route should be provided by the web server, which aggregates data obtained through logging the api calls and the requests received in the web server;
- ***concurrent number of requests*****:** create a script that sends a number of parallel requests in batches (e.g. 500 requests in batches of 50 parallel requests) and draws out some metrics about the behaviour of your API;

General observations:

- Any solution that matches the requirements is accepted (i.e. the logic or parts of it can be implemented on client side);
- In the evaluation process, the complexity of the used APIs will be taken into consideration;
- At least one of the used APIs should require an API key;
- Any form of secret (e.g. email password, secret key) should not be hardcoded in code. Try to at least store it in a configuration file;
- The homework can be implemented in any programming language.