Pre-lab 1: Introduction and background concepts

1 Introduction/overview

Students taking SENG365 come from different degrees programmes, and have taken different 100-level and 200-level courses. To help ensure that students have a fairly consistent foundation for SENG365, this document provides background concepts for the SENG365 course.

For those that are new to web development, read through the document and visit the links provided to read more about each topic. For those of you that are well accustomed with the concepts discussed, use this document as a source of revision, going into as much detail as you feel necessary.

If anything doesn't make sense, or you want to learn more, email your tutor, or ask your tutor in the lab session/s.

The concepts covered in this document are:

1. A brief history of the Web
2. The client/server model
3. The HyperText Transfer Protocol (HTTP)
4. Application Programming Interfaces (APIs).
5. Source code version control and the GitHub code repository

2 The history of the Web

The World Wide Web (WWW, or “Web”) was invented by Tim Berners-Lee in 1989-1991. While working at CERN in Switzerland, Berners-Lee saw that there was a problem with the way employees shared information. Different computers ran many different programs that employees had to learn. With the information required to learn the programs scattered over many computers, it was often easier to just ask people for advice while they were drinking coffee.

Berners-Lee proposed a solution that made use of the internet and an emerging technology called ‘hypertext.’ By the end of 1990, he had invented many technologies that we still use today:

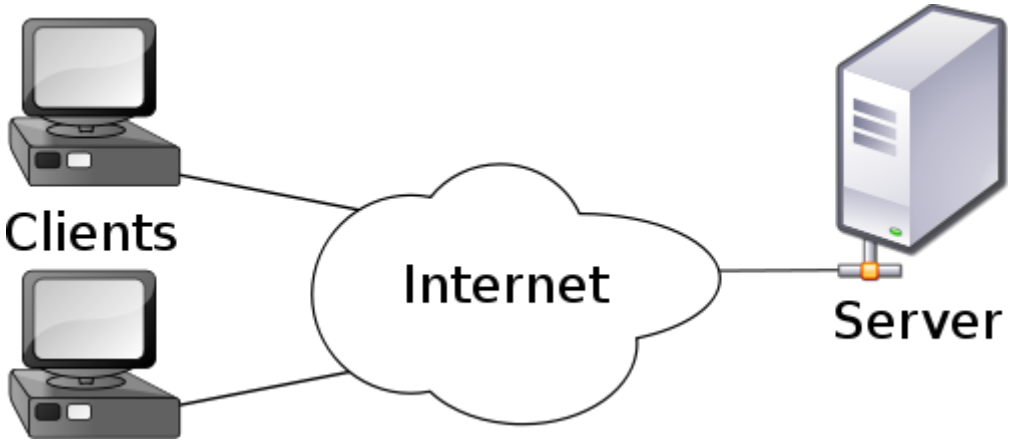
- HTML (Hypertext Markup Language): the formatting language for ‘pages’ of information on the Web.
- URI (Uniform Resource Identifier): An addressing system used to point to a specific resource on the Web (commonly known as a URL: uniform resource locator).
- HTTP (Hypertext Transfer Protocol): the protocol used for transferring content from one web resource to another. HTTP can be contrasted with other protocols used on the internet e.g. file transfer protocol (FTP).
- The first Web browser
- The first Web server

You may have noticed that the list of bullet points refers to pages, resources and content. The information that is shared via HTTP, and how that information is specified, is diverse and complex. The SENG365 course will help you to better appreciate this variety of information. You can read more at: http://webfoundation.org/about/vision/history-of-the-web/

A distinction should be made between the internet, as a global network of computers that communicate with each other using various protocols, and the World Wide Web, a global information system that ‘sits’ on top of the internet. HTTP is known as an application-level protocol, a high-level protocol for sharing information. This contrasts with lower-level computer-communication protocols such as TCP/IP (Transport Control Protocol / Internet Protocol) and UDP (User Datagram Protocol). Different communication protocols have an implication for the reliability of communication, and this impacts the efficacy of application protocols. You can read more at: https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/How_the_Web_works

3 The client/server model

3.1 How does the Web work?



Credit: http://girldevelopit.github.io/gdi-featured-web-concepts

3.2 The client-server model

The de facto standard model for web (HTTP) applications is the client-server model, where a web server communicates with multiple web clients. A typical web client would be a web browser. This means that on the internet exists servers (which store the web resources) and clients (that use the web resources). The servers exist to serve the client.

Example 1: viewing the university's home page

1. I open my browser (Google Chrome) and type the university's URL in the address bar (canterbury.ac.nz)
2. My computer sends off a HTTP request to get the resource located at canterbury.ac.nz. It first uses a Domain Name Server (DNS) to translate this name into an IP address but that is a networking task and thus, beyond the scope of this class.
3. The university's Web server receives the request and responds by sending the resource at canterbury.ac.nz back to me via HTTP.
4. My computer receives the resource and passes it back to my browser.
5. The browser parses the resource and determines how to present it to me.
6. The browser presents me with the resource.

4 HyperText Transfer Protocol (HTTP)

4.1 The HyperText Transfer Protocol (HTTP)

HTTP is the protocol invented in 1989 by Berners-Lee for transferring resources between clients and servers on the internet. There are two types of HTTP messages, a request and a response. Typically, an HTTP client (such as a browser) will send an HTTP request message to an HTTP server for some resource, and then the server will respond with an HTTP response message that contains the requested resource.

Some points about HTTP:

1. HTTP/1.0 is connectionless. This means that after making a request, the client disconnects from the server. When the server is ready to send its response, it must first re-establish a connection to the client, send the response and then disconnect again. For HTTP/1.1, the connection is specified by headers in the HTTP message (explained below).
2. HTTP can deliver any sort of data as long as both computers are able to read it.
3. HTTP is stateless. This means that the client and the server only know about each other during the current interaction. If another resource is requested then the two must establish a new connection. (Note: The new HTTP/2 protocol is a stateful protocol meaning that a connection between the client and server remains open. This will be introduced later in the course.)
4. Almost all HTTP traffic on the internet today follows HTTP/1.1.

4.2 The HTTP message

A HTTP message contains three parts: a start line, one or more header lines, and an optional body (that contains multiple lines). Example 2 compares two examples of an HTTP message: on the left, an example of an HTTP request from an HTTP client; and on the right an example of an HTTP response from an HTTP server. In the example, there is no body to the HTTP request.

Example 2: a HTTP/1.1 request and response for retrieving the UC engineering homepage (http://www.canterbury.ac.nz/engineering/)

The HTTP request		The HTTP response
`GET /engineering/ HTTP/1.1`	Start line	`HTTP/1.1 200 OK`

`Host: www.canterbury.ac.nz`
`Connection: keep-alive`
`Accept: text/html`
	Headers	`Date: Sun, 14 May 2017 04:06:06 GMT`
		`Content-Length: 66900`
		`Access-Control-Allow-Origin: *`
		`Connection: close`
		`Content-Type: text/html; charset=utf-8`

	Body	`<!DOCTYPE html>`
		`<head>`
		`<title>College of Engin….`

The HTTP request start line has three parts; the HTTP method, the URI of the resource to be retrieved and the HTTP version.

```
GET /engineering/ HTTP/1.1
```

The response message then replies with a response line that includes the HTTP protocol version, the status code and a status message. A '200 OK' code means that the request was processed without problem. If the server was unable to find the resource then a '404 Not found' code would be returned. More information on response codes can be found at: https://en.wikipedia.org/wiki/List_of_HTTP_status_codes

```
HTTP/1.1 200 OK
```

Next, we have the HTTP headers. Headers provide a structure for extra information in the message. A list of possible headers with explanations of what they are for can be found at: https://en.wikipedia.org/wiki/List_of_HTTP_header_fields

Finally, we have the body of the message. For this request, no body is specified as there is no content to send to the server. The server replies with a HTTP message, the body of which contains the HTML of a page to be rendered by the browser and presented to the user.

4.3 Introduction to Chrome DevTools

The Chrome DevTools are a set of tools built into Google Chrome that are useful for debugging web applications. Similar tools exist in most of the main browsers. In this exercise, we are going to use DevTools to inspect the HTTP messages for a web page.

4.3.1 Exercise

1. Open Chrome
2. Open DevTools by going to the Chrome menu and selecting ‘More Tools’ > ‘Developer Tools.’ Alternatively, you can open DevTools using the shortcut: CTRL + SHIFT + I .
3. Chrome will open DevTools. This may be at the bottom of your browser, or in a separate window. In DevTools, open the ‘Network’ tab.
4. In the browser window, navigate to https://www.wikipedia.org/.
5. You should see the HTTP requests for all the different resources used on the Wikipedia home page appear in DevTools.
6. Open the HTML document in the list of names; this will be named ‘www.wikipedia.org.’ We look at HTML documents later in the course so don’t worry if you don’t know what a HTML document is.
7. Use DevTools to inspect the request and response headers. Can you identify the start line, headers and body?

5 Application Programming Interfaces (APIs).

5.1 What is an API?

An Application Programming Interface (API) is not a web-only concept. In general, an API describes an interface that can be used to programmatically interact with an application. For example, a Java interface defines an API that software engineers can use to access the methods and subroutines in an application. On the web, the term API is commonly used to describe an interface provided by a server or service for the use of web clients. HTTP requests are used to interact with API’s across the web. This will be discussed further in the lectures.

Kevin Stanton, API chapter manager at Sprout Social defines APIs as “a precise specification written by providers of a service that programmers must follow when using that service. … It [the API] describes what functionality is available, how it must be used and what formats it will accept as input or return as output. In recent years, the term API colloquially is used to describe both the specification and service itself, e.g., the Facebook Graph API.”

When writing applications for the web, it is usual to keep the application's core functionality and processing on the server. An API can be used as the interface to this application. The web client uses the API to access the application services provided through the server.

5.2 Why separate the client and server?

There are many reasons why we may want to structure our web application in a way that keeps the core functionality out of the client via use of an API. Here are two reasons why:

1. **Multiple flexible clients**: Structuring the client as a frontend interface whose only purpose is to interact with the API allows for clients to be interchangeable. Each type

of device can be given its own interface and all interact with the same API. For example, a desktop client, a mobile phone client, and a Smart TV can all access the same web service using different device-specific clients.
2. **Independent scaling**: Client/server separation allows for each to scale independently. This saves money and allows you more flexibility in modifying your application as your user base grows.

5.3 API-first development

Many practitioners now advocate 'API-first development.' This is where software engineers begin writing an application by designing and creating the API. This allows the developers to focus from the start on how the overall application will be structured and what functionality it will contain. Some other key reasons for API first development are:

1. Defining the API early means that both frontend and backend developers can begin to work on the project with a clear idea of how the server and client interact with each other.
2. The security of the API is planned and integrated into the development from the outset.
3. Defining the API means knowing how the application's processes will work together. It becomes a tool for finding design flaws early in the development process.
4. Similarly, API-first development also becomes a way to more easily document how the application will work.
5. API-first development helps to design the backend database by helping to visualise the structure of the required data.
6. API-first development fits naturally with agile development. (This is relevant to other software engineering courses, such as SENG302.) Once the API is defined, developers can work iteratively on features in order of priority.

6 Version control and GitLab

Version control systems track changes and provide control over changes to files. They are commonly used to control the source code of software projects. Read more about them here: https://en.wikipedia.org/wiki/Version_control

For this course, we will be using GitLab. GitLab is based on the widely used 'Git' version control system, and so it is important that you are comfortable with the basics of git prior to starting the course. There are a lot of resources online for learning git, here is a simple guide that should give you an understanding of the key concepts: http://rogerdudler.github.io/git-guide/

Git and Gitlab also have their own dedicated Pre-lab. Make sure you understand the concepts covered before starting the course.

GitLab also includes a markdown-based wiki system that we will be using during this course (as it is a lot nicer than learn). Here is a tutorial on the basics of how to write in markdown: http://www.markdowntutorial.com/

7 Databases

Many software applications use databases to persist their data to permanent storage on the server. In this class, we will be using the MariaDB database for our applications, which is a “drop-in replacement” for MySQL. This means that a MySQL client of the same version should be compatible with a MariaDB server.

The MySQL website provides many in-depth tutorials on MySQL which can be used as a reference guide throughout this course:
https://dev.mysql.com/doc/refman/5.7/en/tutorial.html

Another important aspect of databases is their conceptual design and internal structure. There are many resources online that introduce conceptual database design. This course assumes background equivalent to the material covered in COSC 265.