

# CMPE 220

## Class 13

### Servers (and client/server applications)

# The Client / Server Model

- A *server* is a program that runs on a computer, providing a specific service to other program(s), called *clients*
- In the POSIX world, a server is often called a *daemon* (demon)
  - a long-running background process that answers requests for services
- *The server program may launch multiple processes, as needed*
- The software that makes up a server may - or may not - have system dependencies
- Examples:
  - Database Management System
  - FTP Server
  - Mail Server
  - Web Server
  - Windowing System

# Basic Web Protocols

Web Browsers use **two** basic services

- Domain Name Server (DNS Server)
  - Looks up a domain name to get an IP address
  - User Datagram Protocol (UDP) – a low latency protocol - on port 53
- Web Server
  - Responds to web page requests
  - HyperText Transfer Protocol (HTTP) – over TCP - on port 80
  - HyperText Transfer Protocol - Secure (HTTPS) – over TCP - on port 443

# Layered Protocols (TCP/IP Model)

Network Model Layers	Domain Name Lookup	Web Page Fetch
Application Layer	DNS request	HTTP or HTTPS
Transport Layer	UDP – User Datagram Protocol	TCP - Transmission Control Protocol
Internet Layer	IP – Internet Protocol	IP – Internet Protocol
Network Layer	Ethernet, etc	Ethernet, etc

- UDP is a connectionless, stateless protocol designed for low latency
- TCP is a connection based, stateful protocol designed for reliability
  - Requires each transmission to be acknowledge by the peer

# Leveraging Protocols

- Once protocols (such as http and https) are defined, they can be used to expand services
- Additional protocols may be defined, either on top of existing protocols, or as complements
- Protocols are the glue that allows client-server application systems to be robust and extensible
- Protocols allow an *ecosystem* of services

# Web Services and APIs

# Web Services

- A piece of software that provides a function or service over the Internet, using a standard (usually XML-based) interface
- Client applications make requests from the service, and get back results
- A web server is a type of API, but not all APIs are web services

# Traditional Web Services (ca. 2007)

- Traditional web services are described by a service contract written in the Web Services Description Language (WSDL)
  - The WSDL document is an XML document that provides a machine-readable description of how the service can be called
  - The WSDL document and the request and response messages are transmitted over http or https
  - <https://www.w3.org/TR/2001/NOTE-wsdl-20010315>
  - W3 or W3C = World Wide Web Consortium



# Traditional Web Services

- Messages use the Service Oriented Architecture Protocol (SOAP)
  - SOAP is an XML-based format
  - [https://www.w3schools.com/xml/xml\\_soap.asp](https://www.w3schools.com/xml/xml_soap.asp)
  - <https://www.w3.org/TR/soap/>
- Runs on top of http/https
- SOAP allows developers to invoke processes running on different operating systems to authenticate, authorize, and communicate using XML data

# RESTful Web Services

- A simpler web services protocol is REST
  - Representational State Transfer
- A software *architecture style* consisting of guidelines and best practices for creating scalable web services
- REST is not an interface standard
  - <https://www.w3.org/2001/sw/wiki/REST>
  - <https://www.codecademy.com/article/what-is-rest>

# The REST Architecture

- REST systems are *stateless*, meaning that the server does not need to know anything about what state the client is in and vice versa
- Client and server implementations are independent; the only connect is via messages
- In the REST architecture, clients send requests to retrieve or modify resources, and servers send responses to these requests

# REST Requests

- REST requires that a client make a request to the server in order to retrieve or modify data on the server. A request generally consists of:
  - an HTTP verb, which defines what kind of operation to perform (GET, POST, PUT, DELETE)
  - a header, which allows the client to pass along information about the request
  - a path to a resource
  - an optional message body or *payload* containing data

# REST Responses

- A REST Response consists of a status code and an option *payload*

Status code	Meaning
200 (OK)	This is the standard response for successful HTTP requests.
201 (CREATED)	This is the standard response for an HTTP request that resulted in an item being successfully created.
204 (NO CONTENT)	This is the standard response for successful HTTP requests, where nothing is being returned in the response body.
400 (BAD REQUEST)	The request cannot be processed because of bad request syntax, excessive size, or another client error.
403 (FORBIDDEN)	The client does not have permission to access this resource.
404 (NOT FOUND)	The resource could not be found at this time. It is possible it was deleted, or does not exist yet.
500 (INTERNAL SERVER ERROR)	The generic answer for an unexpected failure if there is no more specific information available.

# Browser-Based Web Service Clients

- You can invoke many web services today from a web browser using AJAX
- You download JavaScript code from the web service provider that makes the AJAX calls.
  - Therefore, you don't worry about what protocol the web service provider uses
- Popular web service providers include Google, Facebook, Amazon, etc.

# Web APIs

- APIs (Application Program Interfaces) include Javascript function calls as well as web service interfaces
- Function calls may hide the details of web services
- A web service is an API, but not all API are web services
- Additional differences
  - Web Services are network based (by definition)
  - APIs are protocol agnostic

# A few FREE web services

- Map zip codes to city & state: <https://www.zipcodeapi.com/API>
- Look up movie info: <https://www.omdbapi.com/>
- Google Maps: <https://developers.google.com/maps/>
- Google Translate: <https://cloud.google.com/translate/docs>
- Weather forecast info: <https://openweathermap.org/>
- Dictionary lookup: <https://dictionaryapi.com/>
- <https://www.freecodecamp.org/news/public-apis-for-developers/>

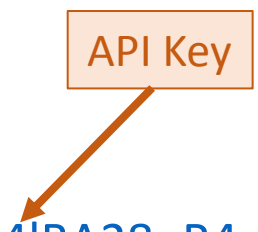


# Example Web API: Google Maps

- Google provides a large number of web-based web services, such as Google Maps.
  - See <https://developers.google.com/maps/documentation/javascript>
  - Load and incorporate the JavaScript API
- More Google Services:
  - <https://developers.google.com/maps/documentation>

# Google Maps Example: html

```
<html>
  <head>
    <title>Example 01 - Google Map Demo</title>
    <link rel="stylesheet" type="text/css" href="./Gmaps.css" />
    <script type="module" src="./Gmaps.js"></script>
  </head>
  <body>
    <h3>Google Maps Demo</h3>
    <!--The div element for the map -->
    <div id="map"></div>
    <script
src="https://maps.googleapis.com/maps/api/js?key=AlzaSyDrRNPIF1ug4lBA28qR4xP8NkLPAZrZQrk&callback=initMap&v=weekly" defer></script>
  </body>
</html>
```



API Key

Gmaps.html

# Google Maps Example: css

```
/* Set the size of the div element that contains the map */  
#map {  
    height: 400px; /* The height is 400 pixels */  
    width: 100%; /* The width is the width of the web page */  
}
```

Gmaps.css

# Google Maps Example: js

```
// Initialize and add the map
```

```
function initMap() {
```

```
  // The location of Uluru
```

```
  const uluru = { lat: -25.344, lng: 131.031 };
```

```
  // The map, centered at Uluru, Australia
```

```
  const map = new google.maps.Map(document.getElementById("map"), {
```

```
    zoom: 4,
```

```
    center: uluru,
```

```
  });
```

```
  // The marker, positioned at Uluru
```

```
  const marker = new google.maps.Marker({
```

```
    position: uluru,
```

```
    map: map,
```

```
  });
```

```
}
```

```
window.initMap = initMap;
```

Latitude & longitude



Initial Zoom Factor



Gmaps.js

# Examples

- Gmap example:
  - <http://cos-cs106.science.sjsu.edu/~012755158/Class-11/Gmaps.html>
  - <https://robertnicholson.info/cs174/Class-11/Gmaps.html>
- Animated map example:
  - <http://californiamissionguide.com/california-mission-guide/california-mission-map/>
- Locations example:
  - <https://pavbhajihut.com/locations/>

# Mission Map Animation

```
function setlocations()
{
location_count = 0;
locations[location_count] = new Array ( "1", "San Diego de Alcalá",
"10818 San Diego Mission Road", "San Diego, CA 92108", "July 16,
1769", 32.790738, -117.106018 ); location_count++;
locations[location_count] = new Array ( "2", "San Carlos Borromeo de
Carmelo", "3080 Rio Road", "Carmel, CA 93923", "June 3, 1770",
36.550741, -121.92009 ); location_count++;
locations[location_count] = new Array ( "3", "San Antonio de Padua",
"End of Mission Road", "Ft. Hunter-Liggert Reservation<br />Jolon, CA
93928", "July 14, 1771", 36.016615, -121.249666 ); location_count++;
```

mission\_map.js

# Getting Latitude & Longitude

- Enter a location in Google Maps to find out its latitude & longitude
  - <https://www.google.com/maps>

# Google API Keys

- Google requires you to get an API key in order to use their services
- You need to provide a credit card number
  - Google will charge if you exceed a usage threshold
  - If you don't enable charges, they will simply block the service when you exceed the usage threshold
- Go to the Google Cloud Console



# Getting a Google API Key

- Go to the Google Cloud Console:
  - <https://console.cloud.google.com>
- Create or select a project
- Click **Continue** to enable the API and any related services
- On the **Credentials** page, get an **API key** (and set the API key restrictions). Note: If you have an existing unrestricted API key, or a key with browser restrictions, you may use that key

# How Do Server Apps Use Web Services?

- Web services are invoked from the browser
- JavaScript code can communicate with the server app to get parameters that are then used to connect with the service
- EXAMPLE:
  - A server application stores my comic book collection
  - I can pull up a comic record from my collection into the browser
  - JavaScript code can call a “Comic Price Guide” service to get the current price

# Email Protocols

# Email Protocols

- Simple Mail Transfer Protocol (SMTP)
  - Post Office Protocol (POP)
  - Internet Message Access Protocol (IMAP).
- 
- All three use TCP, and the last two are used for accessing electronic mailboxes.
- 
- Special records stored in DNS servers play a role as well, using UDP.

# Mail and DNS

- DNS servers hold several record types
- A DNS 'mail exchange' (MX) record directs email to a mail server
- Web servers (A records) can be independent from mail servers (MX records)

# Primary DNS Record Types

- **A Record:** The Address Mapping record (or DNS host record) stores a hostname and its corresponding IPv4 address
- **AAAA Record:** The IP Version 6 Address record also stores a hostname but points the domain to its corresponding IPv6 address
- **DNS CNAME Record:** The Canonical Name record can be used as a hostname alias that points to another domain or subdomain but not to an IP address
- **MX Record:** The Mail Exchanger record indicates an SMTP email server for the domain
- **TXT Record:** A text (TXT) record can store any type of descriptive information in text format. Often used for authentication

# Protocols and Data Types - MIME

- Although not a protocol, there is a series of Multipurpose Internet Mail Extensions (just MIME, never “MIMEs”) for various types of email attachments (not just simple text).
- MIME types tell the receiver how to handle the attachment
- MIME types are also used in web services
- <https://www.w3docs.com/learn-html/mime-types.html>

# SMTP

- SMTP stands for Simple Mail Transfer Protocol, and it is responsible for sending email messages
- This protocol is used by email clients and mail servers to exchange emails between computers
- A mail client and the SMTP server communicate with each other over a connection established through a particular email port (port 25)
  - Ports 587 & 465 => encrypted
- Allows any mail client to be used with any mail server



# POP / POP3

- Post Office Protocol
- Allows clients to retrieve and download emails from a server
  - Server port = 110
- Optional leave messages on the server for some period of time
- An early, basic protocol

# POP Deficiencies

- The ability to mark a message as read on multiple devices
- The ability to synchronize sent items from multiple devices
- The ability for emails to be pushed to your device as they arrive
- The ability to create folders in your POP account

# IMAP

- Internet Messaging Access Protocol
  - A replacement for POP
- With IMAP accounts, messages are stored in a remote server
- Users can log in via multiple email clients on computers or mobile device and read the same messages
- All changes made in the mailbox will be synced across multiple devices and messages will only be removed from the server if the user deletes the email