Review For Midterm

Exam

- Posted in Moodle in Week Nine materials
- During Lecture (with exceptions)
- All exceptions must be cleared through me via email
- Bring a laptop
- Open book, open internet
- 50 minutes – Moodle will open and close the test
- About 30 questions
- 100 points or 10% of your final grade
- Password – either in lecture or your own copy

Review For Midterm

Linux Shell

- What is the Linux shell?
- Navigating the shell – using directories, files
  - pwd, mkdir, cd, ls, cp, rm, mv, rmdir
- Processing files
  - cat, more, head, tail, wc, grep, diff, sort, find
- Running programs
  - redirecting output, pipes, ps
- History of commands typed

Review For Midterm

Shell Scripting, Regex, Awk & Sed

- Shell Scripting
  - #! – shebang
  - Chmod
  - Variables, comments
- Regular Expressions
  - . [a-z] * ? + ^ $ \ [^] \< \> x\{m\}
- awk & sed
  - Processing files and records using regex
  - awk is a full-blown programming language
  - awk is good for searching input files/fields and reporting
  - sed is a less robust language
  - sed is good for modifying a streamed file

Version Control, git

- Version Control
  - Remote versus local repo
- Git
  - Init, status, diff
  - Working copy, staged file, commit
  - Checkout, head, merge

Project Management

- Project Management Principles
  - What makes a project a project?
  - What makes a project successful?
  - Triple constraint model
  - Managing scope
- Project Management Methodologies
  - Waterfall – its benefits
  - Agile, Scrum – its benefits
  - The agile manifesto – key principles
  - Daily Scrums: 15 minutes, 3 questions

Review For Midterm

- HTML
  - uses pre-defined tags
  - Markup language rendered by the browser

- CSS
  - Cascading Style Sheets
  - Hierarchy of applying style
  - 3 places to apply style
    - External
    - Internal in <style> in <head>
    - Internal in specific tags

Review For Midterm

Defining Requirements

- **What is a "Use Case"**
- **What are "functional" requirements versus "nonfunctional" requirements**

Full Stack Development

- **What does it mean?**
- **Elements of the stack**

Review For Midterm

- Database Fundamentals
  - Tables, Rows, Columns
  - Primary and Foreign keys
  - Data model
- SQL
  - Select statement
    - Where clause conditions (like, in)
    - Group functions, Group By, Having
    - SubQuery, Join, Outer Join
  - DDL – create, alter, drop
  - DML – insert, update, delete

Review For Midterm

- Web Services
  - http, xml, json, api
- REST & SOAP
  - REST is an architectural style
    - Stateless, layered, cacheable, scalable
    - Uniform interface methods (http), data format (json, xml)
    - ~70% of public APIs
  - SOAP is a protocol
    - Relies on XML
    - Requires WSDL
    - Less flexible than RESTful web services

Web Services

Agenda

- Web Services Overview
- Protocols
  - http
  - xml
  - json
  - api
  - REST
  - SOAP
- **Some code examples**

Protocols

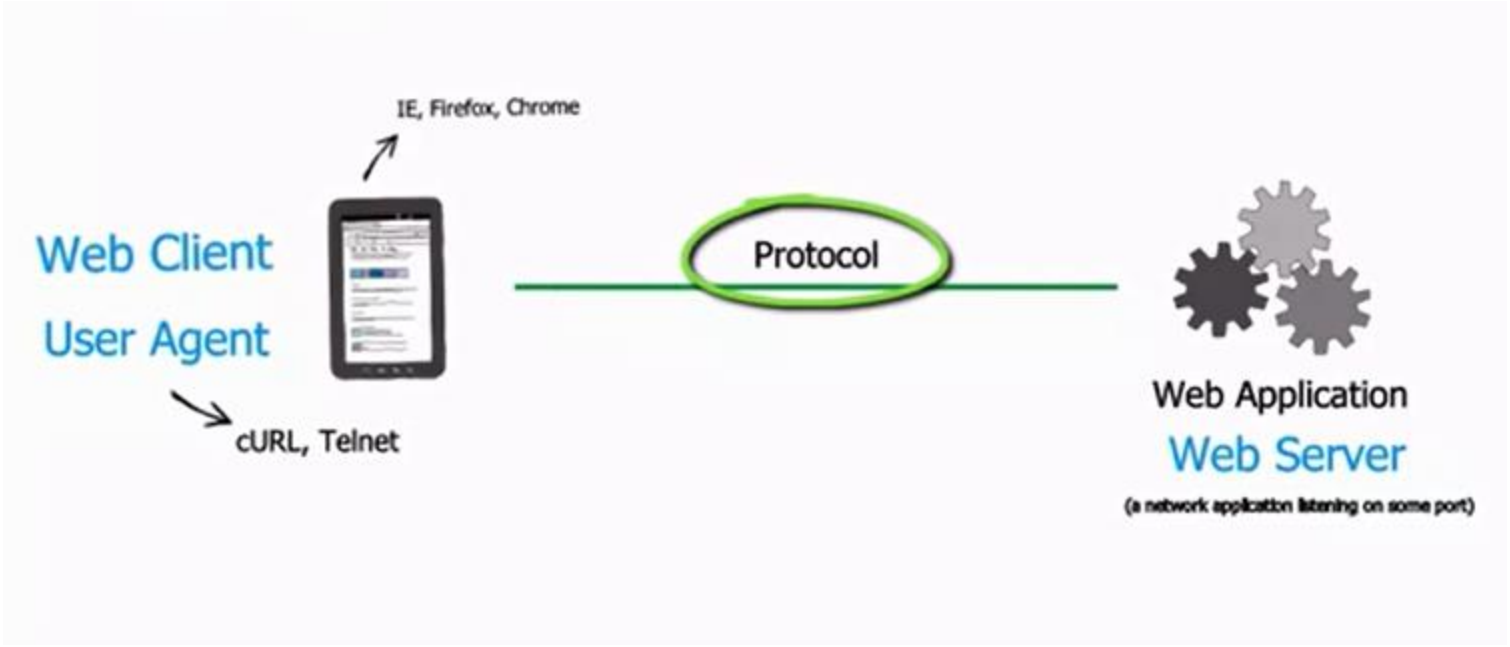
- How do we pass messages and requests from one layer to another?

  PROTOCOLS !

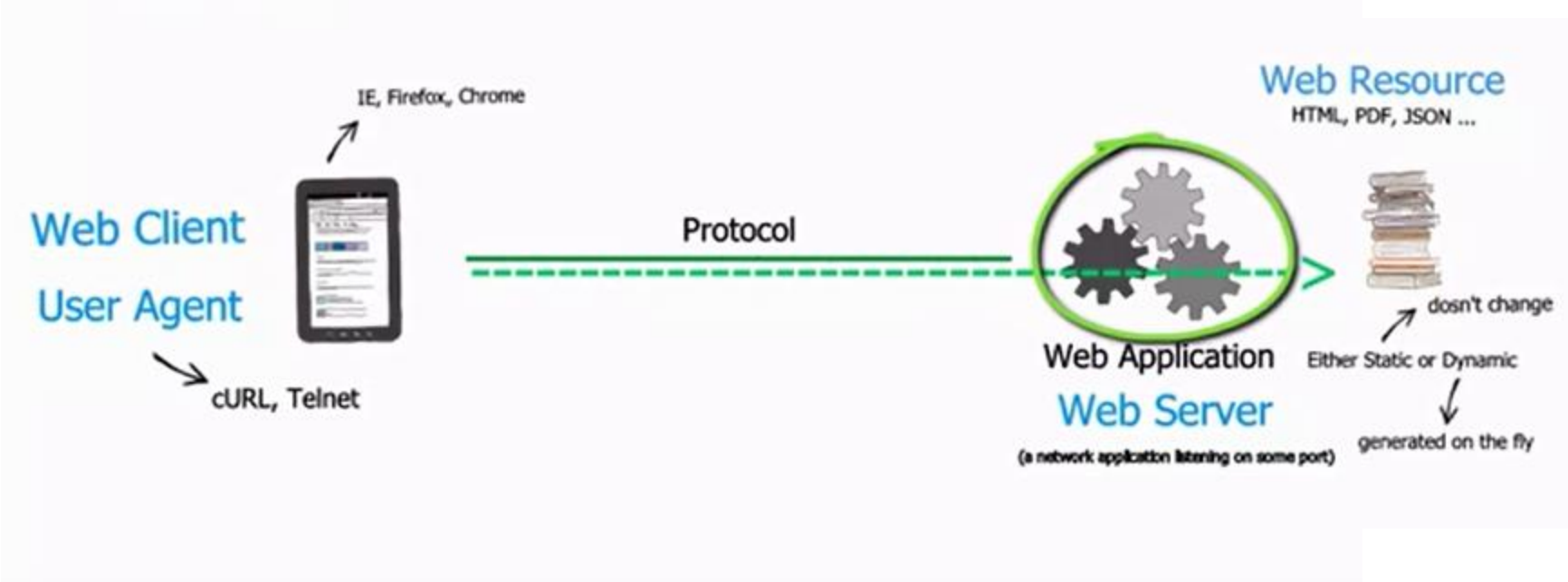
Protocols

- Internet Protocols
- What happens when you type a URL into a browser and press <ENTER>?
- What happens when you click on a hyperlink in a web page?

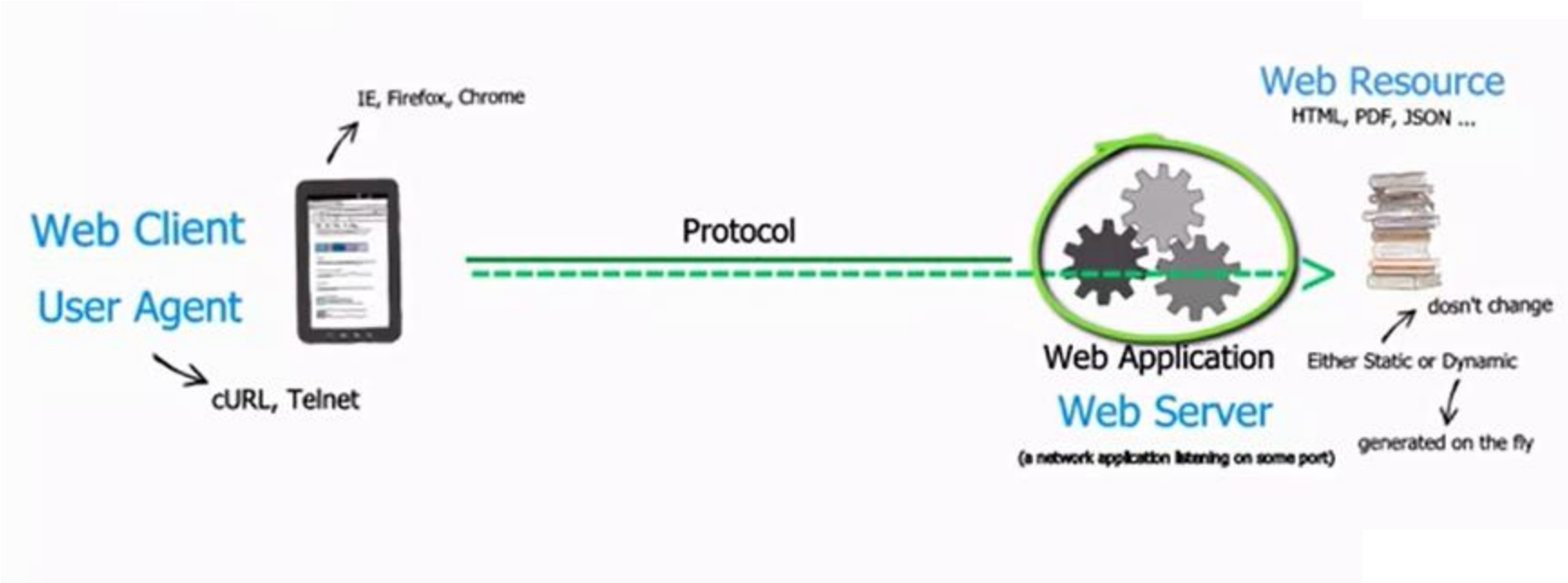
Protocols


IE, Firefox, Chrome
Web Client
User Agent
cURL, Telnet
Protocol
Web Application
Web Server
(a network application listening on some port)

Protocols


IE, Firefox, Chrome
Web Client
User Agent
cURL, Telnet
Protocol
Web Resource
HTML, PDF, JSON ...
dosn't change
Either Static or Dynamic
generated on the fly
Web Application
Web Server
(a network application listening on some port)

Protocols



Each web resource is identified by a URI

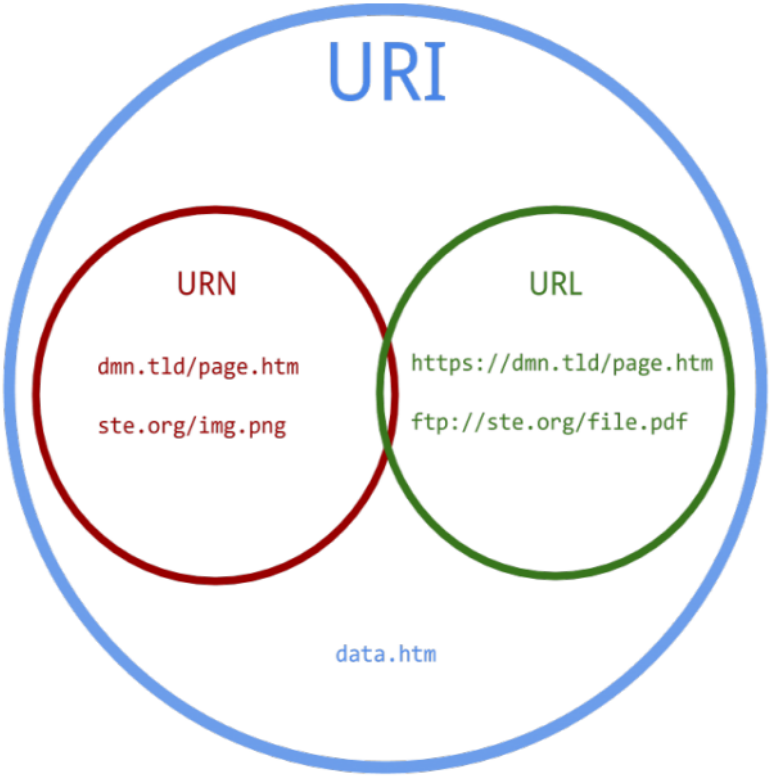
Protocols

- The URI
  - http://www.colorado.edu
  - URI, URL, URN ?

URL = locator
(where/how to find it)

URN = name
(what is its name)

URI = either one



Protocols

- **HTTP** – a request/response protocol
  - It is **STATELESS**
  - The client submits a request, HTTP responds with the requested resource and a return code
    - Resources may be static or dynamic
    - Resources may redirect, include other resources, etc.

- **HTTP Methods**
  - GET Retrieves the URI
  - POST Submits a resource to the URI
    Like submitting a FORM to be processed by a script
  - PUT Stores a resource under the URI
  - DELETE Deletes the URI

Protocols

- **Common HTTP return codes**
- 200 : OK
- 302 : Redirect
- 400 : Bad Request
- 401 : Unauthorized
- 403 : Forbidden
- 404 : Not Found
- 500 : Server Error

Protocols

- **Passing data to/from the web server**
- XML Extensible Markup Language
- JSON Java Script Object Notation

XML

XML Extensible Markup Language

- “Tag” based, like HTML
- Tags are user-defined
- XML is human readable AND machine readable
- Tags describe the data (XML tags do NOT display the data like HTML tags do)
- You can use a programming language like javascript or php or python to read, parse, modify and write XML documents sent/received to/from a Web Service
- The XML document structure is defined by a DOM – Document Object Model

```
<bookstore>
   <book category="cooking">
      <title lang="en">Everyday Italian</title>
      <author>Giada De Laurentiis</author>
      <year>2005</year>
      <price>30.00</price> </book>
   <book category="children">
      <title lang="en">Harry Potter</title>
      <author>J K. Rowling</author>
      <year>2005</year>
      <price>29.99</price> </book>
   <book category="web">
      <title lang="en">XQuery Kick Start</title>
      <author>James McGovern</author>
      <author>Per Bothner</author>
      <author>Kurt Cagle</author>
      <author>James Linn</author>
      <author>Vaidyanathan Nagarajan</author>
      <year>2003</year>
      <price>49.99</price> </book>
   <book category="web" cover="paperback">
      <title lang="en">Learning XML</title>
      <author>Erik T. Ray</author>
      <year>2003</year>
      <price>39.95</price> </book>
</bookstore>
```

JSON

- Represents data in key:value pair format.
- Many think JSON is easier to use than XML
- More compact than XML
- Like XML, JSON is easy for both humans & computers to understand

JSON:

```
{"employees":[
    { "firstName":"John", "lastName":"Doe" },
    { "firstName":"Anna", "lastName":"Smith" },
    { "firstName":"Peter", "lastName":"Jones" }
] }
```

XML:

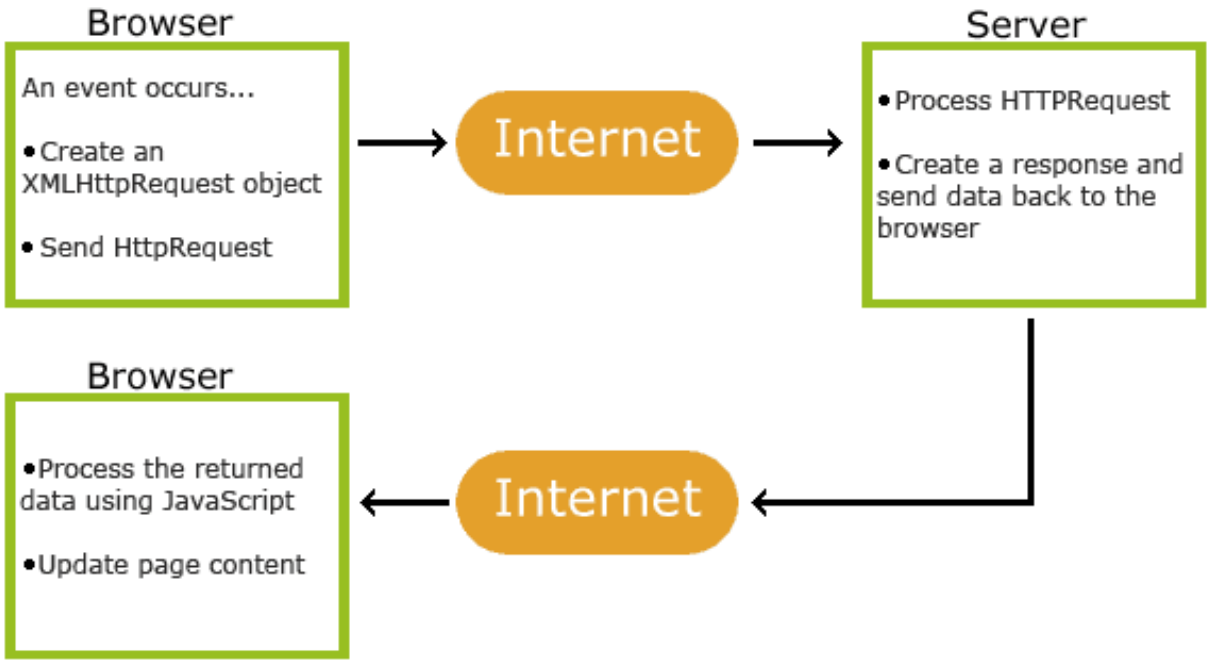
```
<employees>
    <employee>
        <firstName>John</firstName> <lastName>Doe</lastName>
    </employee>
    <employee>
        <firstName>Anna</firstName> <lastName>Smith</lastName>
    </employee>
    <employee>
        <firstName>Peter</firstName> <lastName>Jones</lastName>
    </employee>
</employees>
```

JSON is Like XML Because

- Both JSON and XML are "self describing" (human readable)
- Both JSON and XML are hierarchical (values nested within values)
- Both JSON and XML can be parsed and used by lots of programming languages
- Both JSON and XML can be fetched with an HttpRequest

JSON is Unlike XML Because

- JSON doesn't use end tag
- JSON is shorter
- JSON is quicker to read and write
- JSON can use arrays



1. An event occurs in a web page (the page is loaded, a button is clicked)
2. An XMLHttpRequest object is created by JavaScript
3. The XMLHttpRequest object sends a request to a web server
4. The server processes the request
5. The server sends a response back to the web page
6. The response is read by JavaScript
7. Proper action (like page update) is performed by JavaScript

JSON

For AJAX applications, JSON is faster and easier than XML:

Using XML

- Fetch an XML document
- Use the XML DOM to loop through the document
- Extract values and store in variables

Using JSON

- Fetch a JSON string
- JSON.Parse the JSON string

AJAX = Asynchronous JavaScript And XML

“AJAX is a developer's dream, because you can:

- **Update a web page without reloading the page**
- **Request data from a server - after the page has loaded**
- **Receive data from a server - after the page has loaded**
- **Send data to a server - in the background”**

(quote from w3schools.com)

AJAX is a technique for creating better, faster, and more interactive web applications with the help of XML, HTML, CSS, and Java Script.

Ajax uses XHTML for content, CSS for presentation, along with Document Object Model and JavaScript for dynamic content display. (JavaScript runs on the CLIENT)

Example AJAX versus CGI (Common Gateway Interface)

https://www.tutorialspoint.com/ajax/ajax_examples.htm

Web Services

Early Web (CGI) 1989

- hypertext / hyperlinks
- page by page

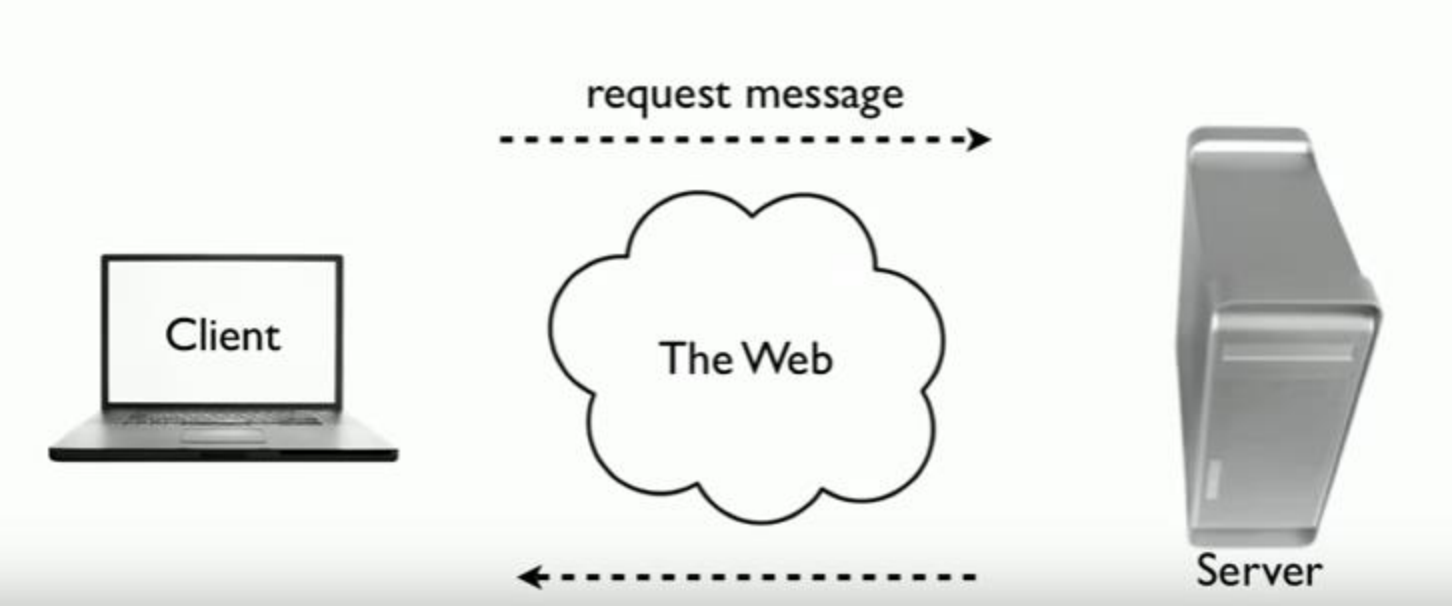
Web 2.0 (AJAX) 2004

- web page stays in place
- parts of the web page are updated

How are web 2.0 requests handled between client and server?

Web Services !

A framework for a conversation between computers over the web



Web Services

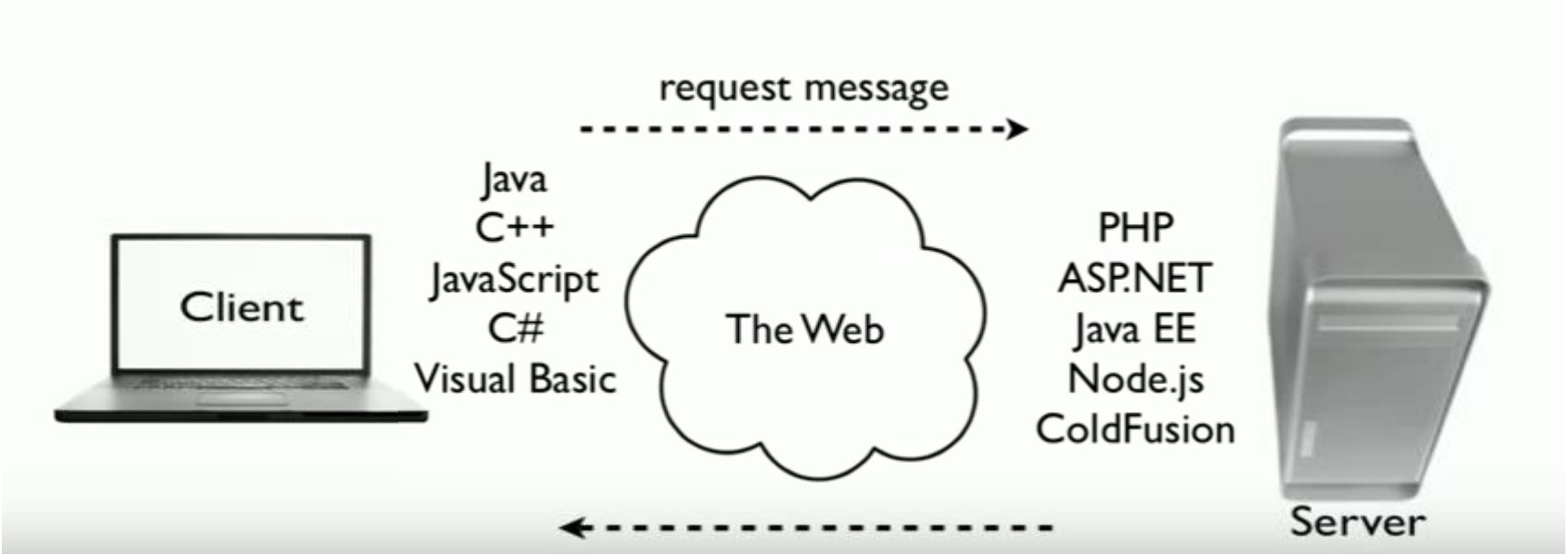
If you want to use a web service, you must use an API (application programming interface)

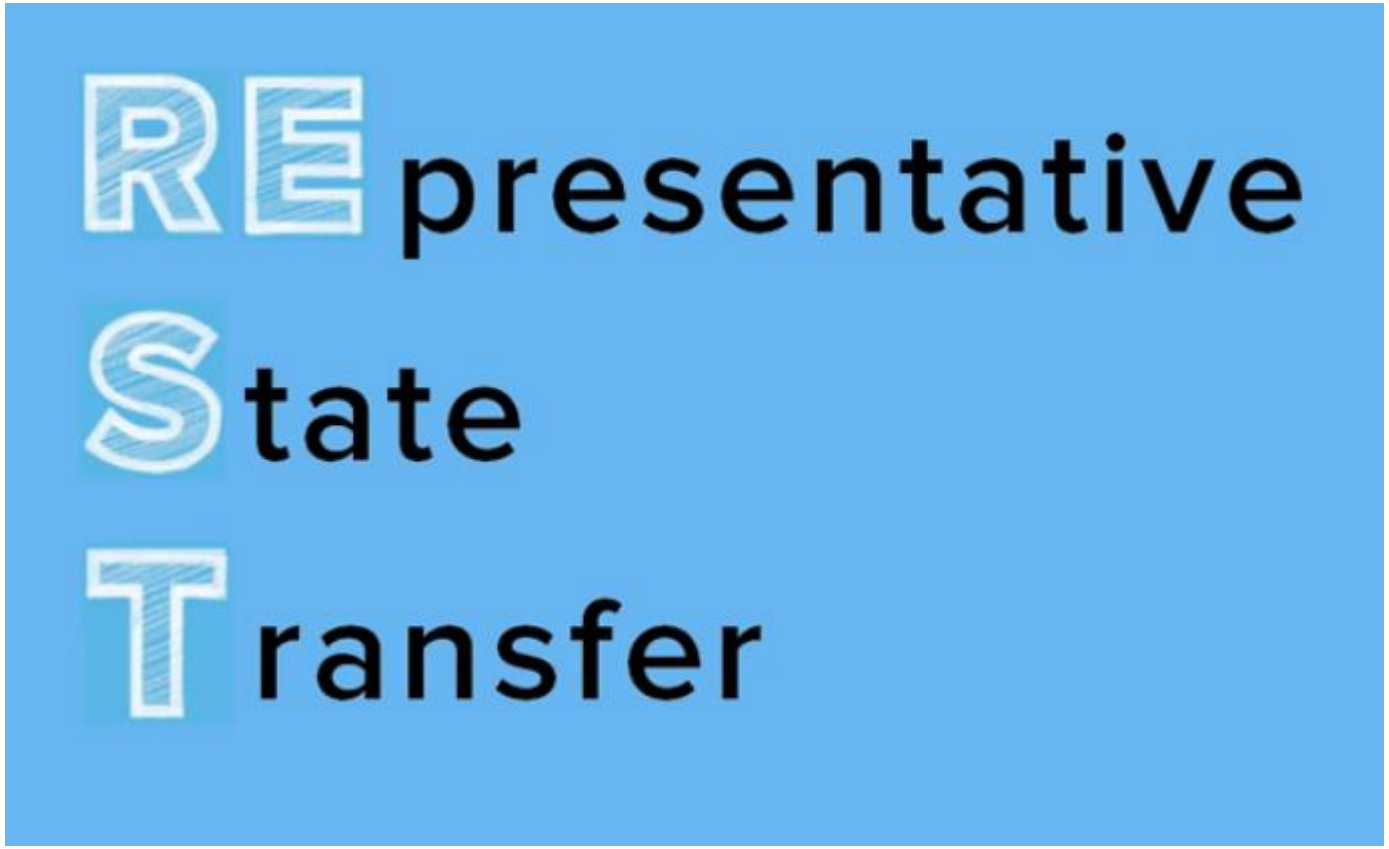
Defines everything you need to know to talk to a web service:

1. **Message format: SOAP, XML, JSON, etc.**
2. **Request syntax: URI, Parameters & Data types**
3. **Actions on the server: named methods, HTTP verbs**
4. **Security: authentication (username & password)**
5. **Response format: SOAP, XML, JSON, etc.**

The web service hides its complexity behind the API

The web service hides its complexity behind the API




REpresentative
State
Transfer

REST
is an *architectural style*

Modern Architectural Style:

Colonial Architectural Style:

The "architectural style" is an abstract concept
it defines the characteristics and features you would find in a house built according to that style

It is NOT the same as the house itself.

REST is an abstract concept that defines the characteristics and features you would find in a web service request built according to the REST style

REST is not really a protocol – it is a set of standards used to define Web Services

Web Services

- Everything in REST is considered as a resource.
  - Every resource is identified by an URI.
  - Uses uniform interfaces. Resources are handled using http POST, GET, PUT, DELETE operations
  - Stateless. Every request is an independent request. Each request from client to server must contain all the information necessary to understand the request.

Web Services

- RESTFul web services are based on HTTP methods
- a RESTFul web service typically defines the base URI for the services, the format/rules of the API, and the set of operations (POST, GET, PUT, DELETE) which are supported.

Characteristics of a request/response following the REST style

Resources follow the rules

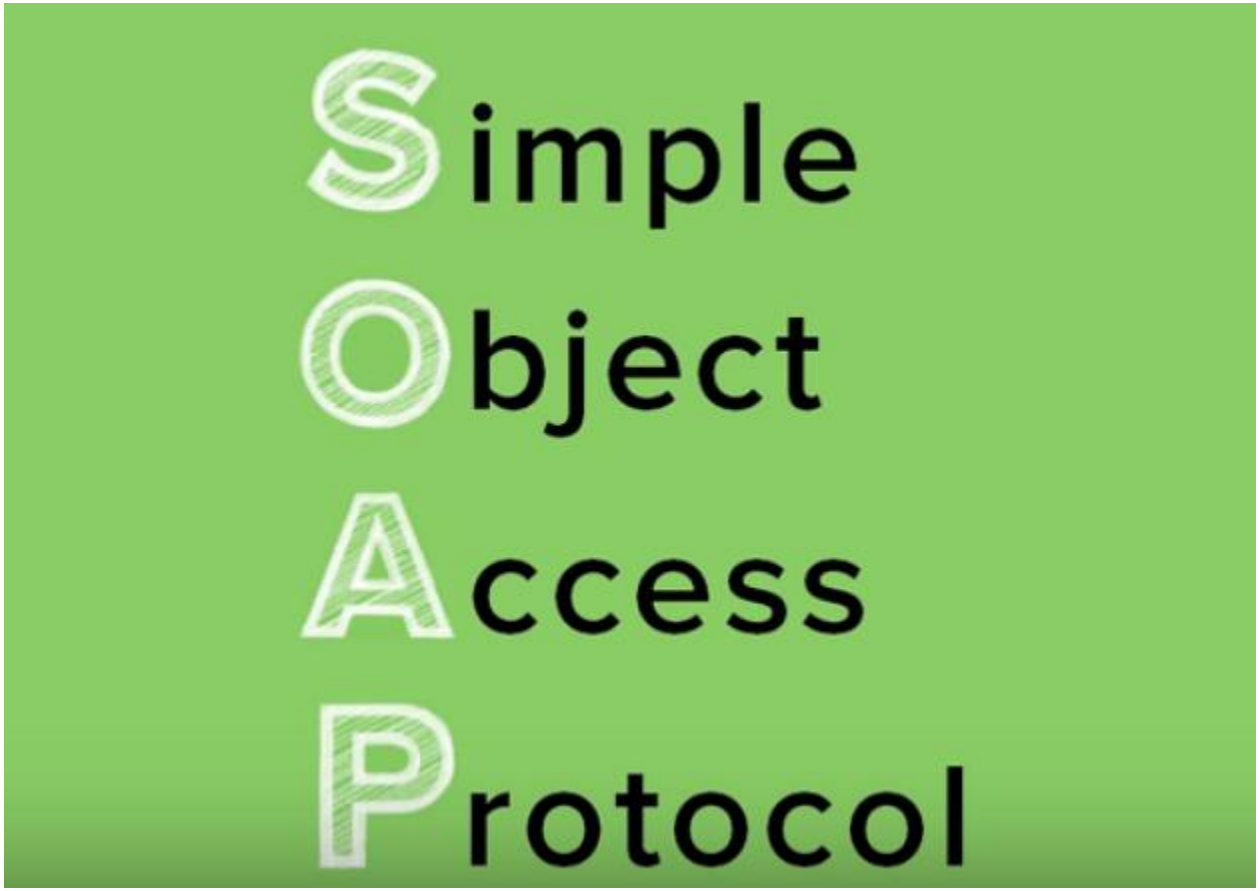
- **URI (**identifies the resource being requested**)**
- **Uniform Interface Methods (**GET, PUT, POST, etc.**)**
- **Uniform Interface Representation (**XML, JSON, HTML**)**

Protocols offer features

- **Client-Server (**like HTTP**)**
- **Stateless (**each request is independent**)**
- **Layered (**may pass through intermediaries**)**
- **Cacheable (**intermediaries may cache for performance**)**

Advantages of a request/response following the REST protocol

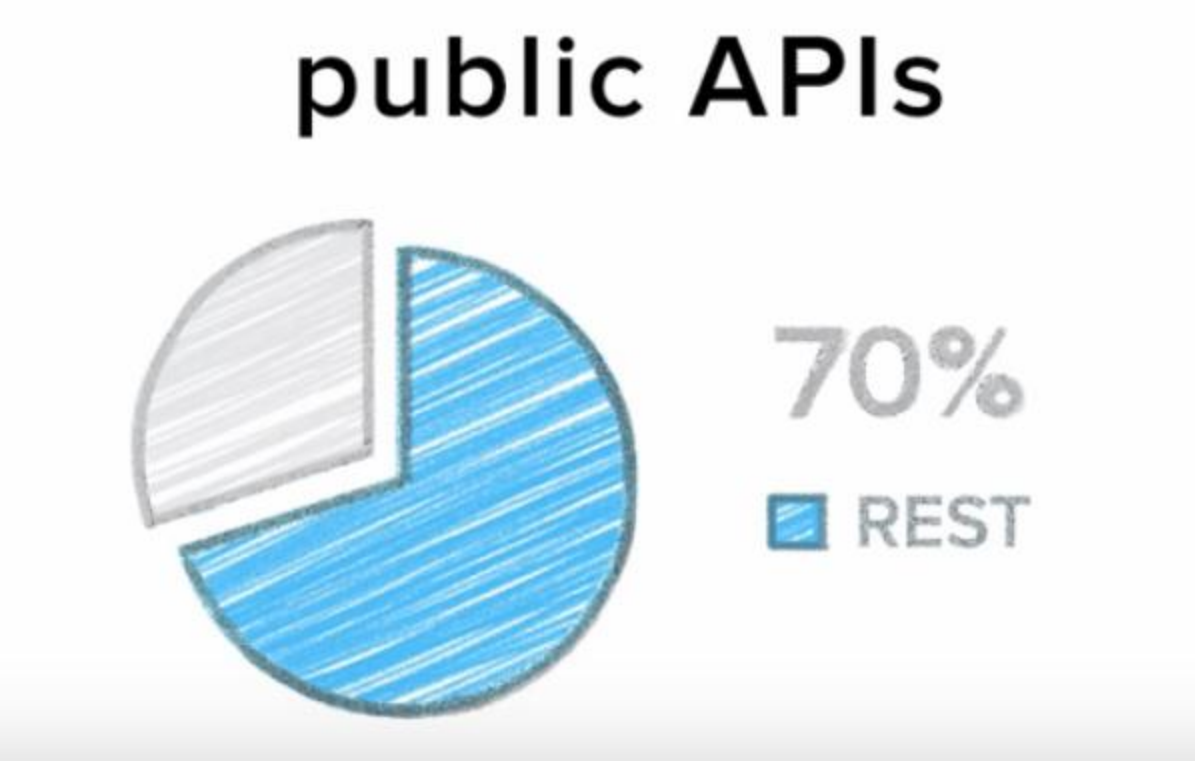
- **Efficiency**
  (through caching & compression)
- **Scalability**
  (gateways distribute traffic, caching, statelessness allows different intermediaries)
- **User Perceived Performance**
  (code on demand, client validation, caching)
- **Simplicity**


Simple
Object
Access
Protocol

SOAP -- Simple Object Access Protocol

REST	SOAP
Representational State Transfer	Simple Object Access Protocol
Architecture Style	An actual protocol
Uses simple HTTP	Uses SOAP envelope, then HTTP (or FTP, or other) to transfer the data
Uses many different data formats like JSON, XML, YAML*	Supports only XML format
Performance & Scalability & Caching	Slower performance. Scalability is limited and complex. Caching is not possible.
Widely and frequently used	Used where REST is not possible

*YAML: YAML Ain't Markup Language
What It Is: YAML is a human friendly data serialization standard for all programming languages.


public APIs
70%
REST

Web Services

- WSDL (Web Service Description Language) is an XML document that defines “contract” between client and service and is static by its nature.

- SOAP builds an XML based protocol on top of HTTP or some other protocol according to the rules described in the WSDL for that Web Service.

SOAP

- A SOAP message is an XML document containing the following elements:
  - An **Envelope** element that identifies the XML document as a SOAP message
  - A **Header** element that contains header information
  - A **Body** element that contains call and response information
  - A **Fault** element containing errors and status information

Web Services

http://www.w3schools.com/graphics/google_maps_basic.asp

Demo of an API for using a web service

```
center:new google.maps.LatLng(40.0150,-105.2705),
zoom:10,
```

http://apigee.com/providers/

A list of published web services and APIs

https://www.youtube.com/watch?v=7YcW25PHnAA

– This video shows a clear example of how we can use REST framework API for using a web service

Cascading Style Sheets

- **Web Pages**
  - Content (what's there: text, links, images, forms)
  - Style (how it looks)

Cascading Style Sheets

- **What is Style?**
  - Objects in an HTML document have “style” attributes
    - Font, font size, font color assigned to headings, paragraphs
    - Background color
    - Size and shape of images
    - Hyperlinks, colors, behaviors
    - Placement of objects on the page

Cascading Style Sheets

- **Applying style**
  - Tag-level: Style attributes within a tag on a page.
  - Page-level: Style defined within <head> of each page.
  - Site-Level: Within an external file, pulled into each page for an enitre website

Cascading Style Sheets

- **Applying style**
  - Tag-level: Style attributes within a tag on a page.
    - Very Granular. Difficult to maintain, No Consistency
  - Page-level: Style defined within <head> of each page.
    - Difficult to maintain across many pages, No consistency

Cascading Style Sheets

- **Applying style**
  - Site-Level: Within an external file, pulled into each page for an enitre website
    - Easy to maintain
    - Saves time for support & maintenance
    - Saves page load time
    - Enables consistency across the website

Cascading Style Sheets

- **What is a Style Sheet?**
  - The style sheet is an EXTERNAL document (.css) containing the rules of style to be applied to your document
  - You “link” to it to bring it into one or more pages
- **Why do we use them?**
  - Consistency from Page to Page
  - Faster, Easier Page Construction

- **Guidelines for a Style Sheet**
  - Should not contain any HTML tags
  - Saved as file with a type/extension of “.css”

Cascading Style Sheets

- **Style Sheet –**
  - Reference it and bring it into your document:

```
<head>
   < link rel="stylesheet"
     type="text/css"
     href="mystyle.css">
< /head>
```

Cascading Style Sheets

- **What does “cascading” mean?**
  - 3 ways to apply “style” to your document
    - External style sheet
    - Internal `<style>` tag in `<head>` section
    - Inline style attributes for a tag
- **Hierarchy of Applying Style**
  - Child tags Inherit Style from Parent tags
  - Detail-Level Overrides High-Level

Style Sheets - Hierarchy

<link rel="stylesheet" href="mainstyle.css" />

<style type="text/css">
Body {color:purple;}
</style>

Red Text

Style Sheets

- **Using the <style> tag**
  - Entered in <head> section
  - Applies to entire document
- **Style Rules are Not HTML**
- **Rule Syntax**
  1. selector – the tag that the rule applies to
  2. { } Braces to contain the rule
  3. Property:value pair
  4. Ends with semicolon

Style Sheets

- **Let's Look at Some Examples**
- **www.w3schools.com/css**

- **Examples**

```
<style type="text/css">
   body  {font-size: 16pt;
          color: blue;
          background-color: pink;}
   h1    {font-size: 24pt;
          color: black; }
   p     {margin-left: 10%;
          margin-right: 10%;}
</style>
```

Style Sheets

- **Multiple selectors are OK**

```
h1, h2        {color:purple;
               text-align:center;}
```

- **Nested selectors are OK**

```
div h1 {color:red;}
```

- **ID References are OK**

```
<h1 id="beginning">
<style> #beginning {color:red;} </style>
```

Style Sheets

- **Classes**
  - A class allows you to set different styles for the same tag
  - Example:

```
<style>
   h1.majorheading {font-family: arial;
                    color: blue;}
</style>
```

then you refer to the **class** in the tag:

```
<h1 class=majorheading>
```

- **Types of Style Rules**
  - element (tag)
  - # id
  - .class
  - Element.class
  - Group (multiple selectors)
  - Nested (selector within a selector)

- **“!important” override indicator**
  - For a rule that must NOT be overridden in cascade
  - “!important” must follow the rule

    <p style=“color: green !important;”>

- **Best Web Tutorials:**

  www.w3cschools.com/css

- **Some Style Properties**
  - Font Size
  - Font Family
  - Font-weight
  - Text-transform – capitalization
  - Word-spacing
  - Letter-spacing
  - Text-align
  - Text-indent

- **More Style Properties**
  - Color – for text
  - Background-color
  - Background-image
  - Box properties

Style Sheets

- **Bootstrap**
  - A style library
  - Easy formatting using templates
  - Easy transition from computer-based viewing to phone-based viewing

Style Sheets

- **Bootstrap**
  - A style library
  - Easy formatting using templates
  - Easy transition from computer-based viewing to phone-based viewing

https://www.w3schools.com/bootstrap/default.asp

- **Bringing Bootstrap into your pages**

1. **Include an external CSS sheet from a CDN (Content Delivery Network)**
2. **Include a jquery (Java Script) library (only needed for certain js plug-ins)**
3. **Include a client java script engine from a CDN**

- **Bringing Bootstrap into your pages**

```
<!-- Latest compiled and minified CSS -->
<link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/boo
tstrap.min.css">

<!-- jQuery library -->
<script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquer
y.min.js"></script>

<!-- Latest compiled JavaScript -->
<script
src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/boots
trap.min.js"></script>
```

Bootstrap Grid System

Bootstrap's grid system allows up to 12 columns across the page.

If you do not want to use all 12 columns individually, you can group the columns together to create wider columns:

span 1	span 1	span 1	span 1	span 1	span 1	span 1	span 1	span 1	span 1	span 1	span 1
span 4				span 4				span 4			
span 4				span 8							
span 6						span 6					
span 12											

Bootstrap's grid system is responsive, and the columns will re-arrange automatically depending on the screen size.

Grid Classes

The Bootstrap grid system has four classes:

- `xs` (for phones - screens less than 768px wide)
- `sm` (for tablets - screens equal to or greater than 768px wide)
- `md` (for small laptops - screens equal to or greater than 992px wide)
- `lg` (for laptops and desktops - screens equal to or greater than 1200px wide)

The classes above can be combined to create more dynamic and flexible layouts.

Using HTML Forms

- **HTML forms**
- **How are they used?**
  - Use the browser's window as a data entry screen
  - Collect information from the user
  - Pass it to the web server via http
  - Invoke a server-side script
  - Passes ***form data*** as input to the script

Using HTML Forms

- **`<form>` tag has several attributes – two are required**
- **ACTION**
  - `<form action="http://URL">` name of a program on the web server
    - URL specifies the location of the executable file on the web server
  - `<form action="mailto:mailrecipient">` sends an email
- **METHOD**
  - `<form method="POST" >` or `<form method="GET">`
    - **POST** when you have large amount of data being sent, encryption available, a two-step process
    - **GET** for small amounts, no security – all in one step

      `<form enctype=`
      - multipart/form-data (default)
      - text/plain (used only for mailto)

Using HTML Forms

- **the <input> tag**
  - Specifies an input field on a form
- **type attribute – tells us what kind of control**
  - **text**
  - **radio**
  - **checkbox**
  - **submit button**
  - **reset button**

Using HTML Forms

- **<form> examples**
- **Text Box**

```
<input type="text" name="Name" size="20" maxlength="30">
```

- **Radio Button(s)**

```
<input type="radio" name="Gender" value="M" /> Male
<input type="radio" name="Gender" value="F" /> Female
```

- **Check Box(es)**

```
<input type="checkbox" name= "size" value="S"
   checked="checked" />Small
<input type="checkbox" name="size" value="M" />Medium
<input type="checkbox" name="size" value="L" />Large
<input type="checkbox" name="size" value="XL" />X-Large
```

- **List Box**

```
<select name="Grade" size="3">
        <option>A
        <option>B
        <option>C
        <option>D
        <option>F
</select>
```

- **List Box via <select> tag**
  - **Size** attribute
    - When absent: you get a "drop down list", first item selected by default
    - When present: indicates the number of items in the list
  - **Selected** attribute: specifies selected item
  - **Multiple** attribute: when "yes", can click > 1

Using Forms

```
<input type="submit" />
<input type="reset" />

<textarea name="comments" cols="40" rows="8">
```

HTML

- HyperText Markup Language
- The foundation of all web pages
- Marking up text with tags
- Tags look like **`<tag>text text text</tag>`**

```
<!DOCTYPE html>
<html>
      <head>
            <!-- head elements go here -->
      </head>
      <body>
            <!-- body elements go here -->
      </body>
</html>
```

Elements

- Paragraph

  `<p> text goes here </p>`

- Headings

  `<h1> big heading </h1>`

  `<h2> smaller heading </h2>`

  `<h3> even smaller heading </h3>`

- Divisions

  `<div> contains multiple paragraphs </div>`

- Lists

  `<ol> an ordered list </ol>` A,B,C or 1,2,3

  `<ul> an unordered list </ul>` bullets

More Elements

- Horizontal Lines

  <hr> size="10" width="xx" float="xxx" </hr>

- Line Break

  **
 </br>**

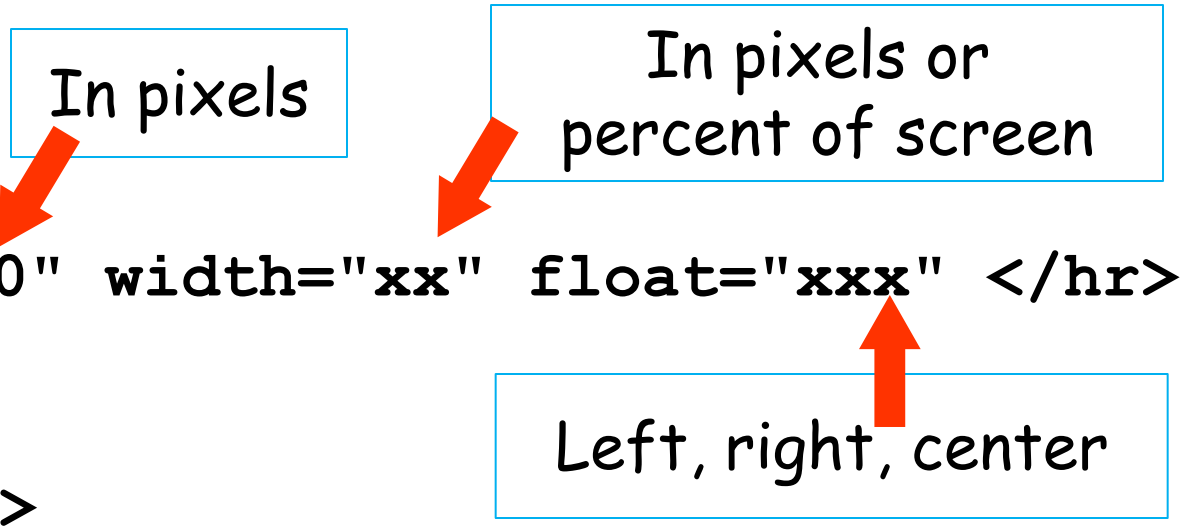
- Blanks

  ** **

- Comments

  <!-- xxxx -->

More Elements



- Horizontal Lines

```
<hr> size="10" width="xx" float="xxx" </hr>
```

- Line Break

```
<br>  </br>
```

- Blanks

```
 
```

- Comments

```
<!-- xxxx -->
```

- Let’s Play

  https://www.tutorialspoint.com/online_html_editor.php

Some basic guidelines

- HTML commands are NOT case sensitive.

  Tip: always enter tags in lower case.
- Anything in quotes *might be* case sensitive

  Such as attribute values:

```
<img src="picture.jpg">
```

- Spaces: Many spaces = one space.
- Always use end tags.
- Nesting elements: From <tag> to </endtag>
- Good habit: Quote all attribute values.

- **Cache**: Browser caches pages. Beware when reloading. SHIFT + Reload or CTRL + Reload will force.

- "**Deprecated**" = no longer supported by the standard.

- It's either a **TAG** or it's **TEXT**

- There are 4 basic **attributes** for every tag

  `id="xxxx"` – identifies it

  `class="classname"` – ties it into a group for style

  `style="xxxxx"` – where xxx is a list of style elements

  `title="xxxxx"` – adds misc info to the tag

Let’s look at an example

http://www.funto o.org/Linux_Fundamentals,_Part_1

Right click and view source, inspect elements

More Elements

- **Hyperlink**
  - Takes the web user to another document
  - Can be on the same site, or ANYWHERE
  - Implemented via **`<a>`** tag ("a" is for "anchor")
  - Uses the **`href="xxxxx"`** attribute
  - Browser identifies Link via underscore and color
    - Avoid **`<u>`** tag
  - Cursor changes shape on "mouseover"
  - Status Bar shows URL of the link on "mouseover"

Linking

- **Managing Hyperlink Colors**
  - **`<body link="xxxx", alink="xxxx", vlink="xxxx">`**
  - xxxx = valid color name
- **Absolute versus Relative URL**
  - Protocol://host.domain.tld/fullpath/file.htm ← absolute
  - file.htm ← relative
  - Relative location set by current page OR <base> tag in <head> section
  - <base href=http://host.domain.tld/path>
  - No file name ➔ uses "index.htm" or "default.htm"

Linking

- **URL**
  - Directory path – filename plus extension
- **Linking to a "marker"**
  - Defined by **id="xxxx"** attribute

Images

- **Two basic types**
  - .gif –Graphics Interchange Format
  - .jpg – Joint Photographic Experts Group
- **.gif**
  - For illustrations
  - Good with large areas of contiguous color
  - Compresses nicely. "lossless"
  - Limited to 256 colors
  - Uses dithering to simulate other colors
  - Allows transparency
  - Supports animation

Images

- **.jpg**
  - For photos
  - Good with large numbers of various colors
  - "lossy" compression.
  - Millions of colors
  - No transparency
  - No animation

Images

- **`<img>` tag attributes**
  - **`Alt="xxxxx"`** ➔ text to display when image is not available
  - **`Hspace="xx", Vspace="xx"`** ➔ puts a buffer of space around the image
  - **`Height="xx", Width="xx"`** ➔ resizes the image
    - Reserves the space to speed in page loading
- **`<img>` as "link" (I.e. a "button")**

```
<a href="week3_ex4.htm"><img src="Button1.gif" /></a>
```

Other Formatting Tags

- **<body>** attributes
  - Bgcolor="xxxx"
  - Text="xxxx"
  - Link, alink, vlink
  - Background="image.gif"
  - Topmargin="xx", leftmargin="xx"

Tables

- **Used Primarily for LAYOUT options**
- **Rows and Columns, "cells"**
- **The browser will SIZE the table large enough to hold the cells' contents**
- **Every row gets the same number of "cell positions", whether or not you define or use them**
- **Table Tags:**
  - <table> defines the table
  - <tr> defines a table row
  - <th> defines the table's header cell content
  - <td> defines the table's data cell content
  - <caption> puts some text above, outside the table

Tables

- **<table> attributes**
  - bgcolor – just like <body>
  - border="xx" size in pixels. Default = no border
  - cellspacing="xx" size of cell space in pixels (space between cells)
  - cellpadding=size of cell pad in pixels (space around contents within cell)
  - width="nn" (pixels or percent) Size of table

Tables

- **<tr> defines a table ROW**
- **<tr> attributes**
  - bgcolor – just like <body>
- **<td> or <th> defines a table cell**
- **<td> <th> attributes**
  - bgcolor – just like <body>
  - Colspan="nn" – number of columns cell spans
  - Rowspan="nn" – number of rows cell spans

Linking

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
<p>In this example, the anchor tag creates a link to another
document.</p><br><br>

<p><a href="http://www.colorado.edu"> CUBOULDER </a></p>

</body>
</html>
```

Linking

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
<p>In this example, the image is a link to another
document.</p><br><br>

<p><a href="http://www.colorado.edu">
<img src="CU.png"></img></a></p>

</body>
</html>
```

Float

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
<p>In this example, the image will float to the right in the
paragraph,
and the text in the paragraph will wrap around the image.</p>

<p><img src="CU.png" style="float:right";>
Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Phasellus imperdiet, nulla et dictum interdum, nisi lorem
egestas odio, Cras ac leo purus. Mauris quis diam velit.
Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Phasellus imperdiet, nulla et dictum interdum, nisi lorem
egestas odio, Cras ac leo purus. Mauris quis diam velit.</p>

</body>
</html>
```

Stack

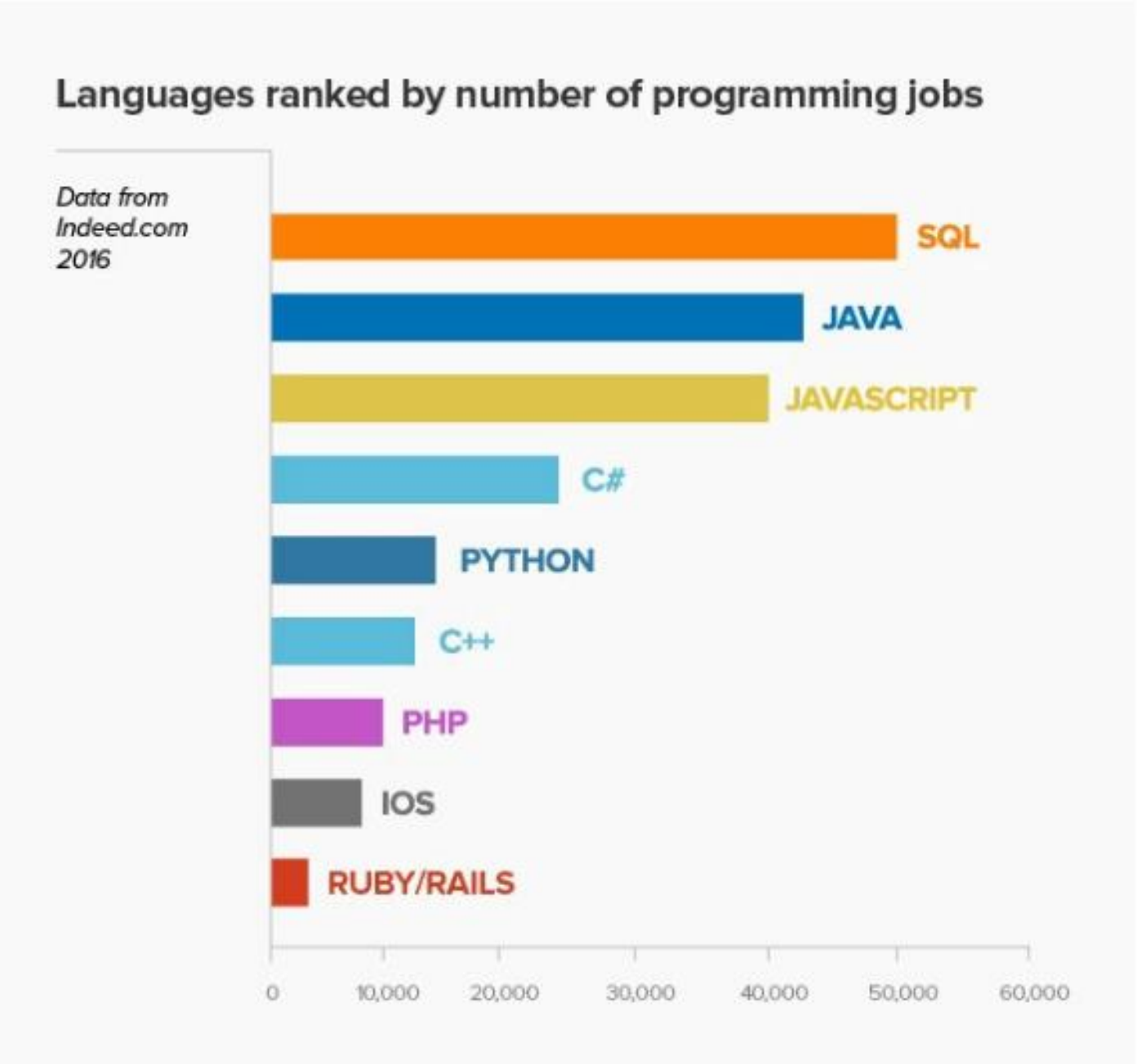
- **Front-End**
  - The presentation layer
  - What the user sees and interacts with
  - What software tools do I use to build it and present it to the users
- **Back-End**
  - Where I code for the underlying business logic
  - The underlying database and code to interact with it
  - Authentication, User Management
- **Full Stack Developer**

  "A Full-Stack Web Developer is someone who is able to work on both the front-end and back-end portions of an application."

Stack

- **Web Pages**
  - HTML and CSS are the building blocks of every web site
  - HTML/CSS = “Front End”
  - Java Script
    - Popular language
    - Runs in browser on client side – Active Web Pages
    - Can also run on the server side

  “JavaScript is the most popular language in both Full-Stack, Front-end, and Back-end Development. It’s the only language that runs natively in the browser, and can double up as a server-side language as well.”
    - NodeJS is an implementation of Java Script


Languages ranked by number of programming jobs
Data from Indeed.com 2016
SQL
JAVA
JAVASCRIPT
C#
PYTHON
C++
PHP
IOS
RUBY/RAILS
0
10,000
20,000
30,000
40,000
50,000
60,000

Stack

Stack

“Full Stack Developer”

- **If you want to become a Full-Stack Web Developer in 2017 and land your first job, below is a list of things you should learn.**
  - **HTML/CSS**
  - **JavaSript (node.js)**
  - **Backend Language**
    - **SQL, Java, pHp, c#, python, c++, php, IOS, ruby**
  - **Database – SQL, NoSQL**
  - **HTTP & REST**
  - **Git**
  - **Basic Algorithms & Data Structures**

Requirements

- Scope
  - Easily understood as a sum of REQUIREMENTS

"If it can't do……., I don't want it."

"It must be able to ……."

"I want a system that will ……."

Requirements

- Two types of Requirements:
  - Functional Requirements ➔ “what” – (Users’ view)
  - Non-Functional Requirements ➔ “how” – (Developers’ view)

Requirements

Often stated as User Stories

Agile User Story Template

User Story ID	As a *<type of user>*	I want to *<perform some task>*	so that I can *<achieve some goal>*
1	Project manager	View a status report from each team member	Ensure the project stays on track.
2	Employee	Be reminded of upcoming deadlines	Complete my tasks on time.
3	Director	See the big picture view of department work	Stay in the loop.

Requirements

Functional Requirements

- User Requirements, Business Requirements
- Documented by **Use Case**
- The sum of the Use Cases = “User Requirements”
- These requirements describe the users’ expectations for **what** the system does for them
- “**WHAT** I want the system to do for our organization”
- A functional requirement for an everyday object like a cup would be: “ability to contain tea or coffee without leaking”.

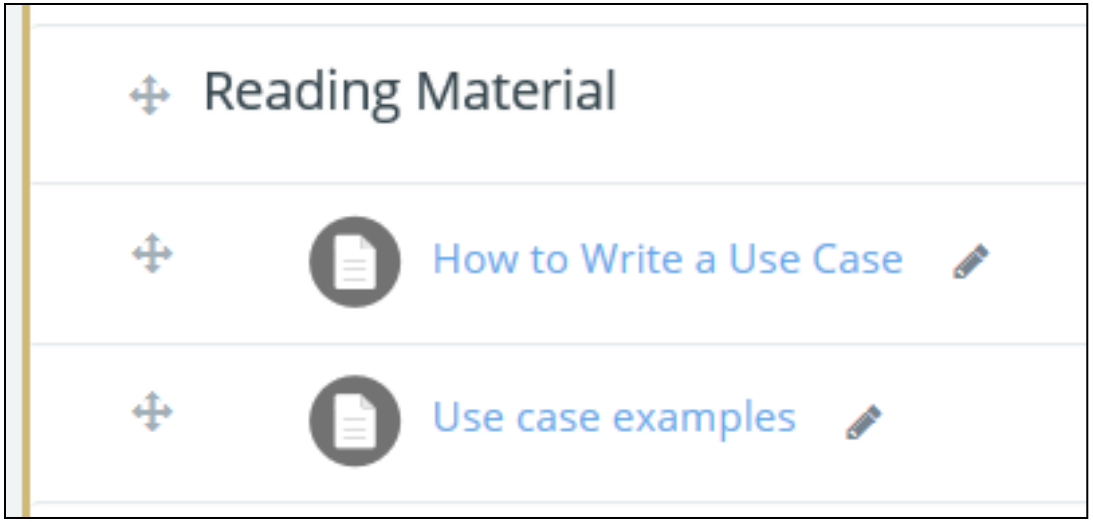
- The definition of a functional requirement is:

  “Any requirement which specifies **what** the system should do.”

Use Cases

- **Use Cases document your users' requirements**

- **Review this week's reading assignments**



- **The format of your Requirements Document can be**
  - **A table (rows, columns, cells)**
  - **A text document with paragraphs/sentences or bullet points**
  - **A picture showing "actors" and "flows"**
  - EXAMPLE: https://en.wikipedia.org/wiki/Use_case

Use Cases

- Martin Fowler (leading industry expert) states:

"There is no standard way to write the content of a use case, and different formats work well in different cases.“

He describes "a common style to use" as follows:

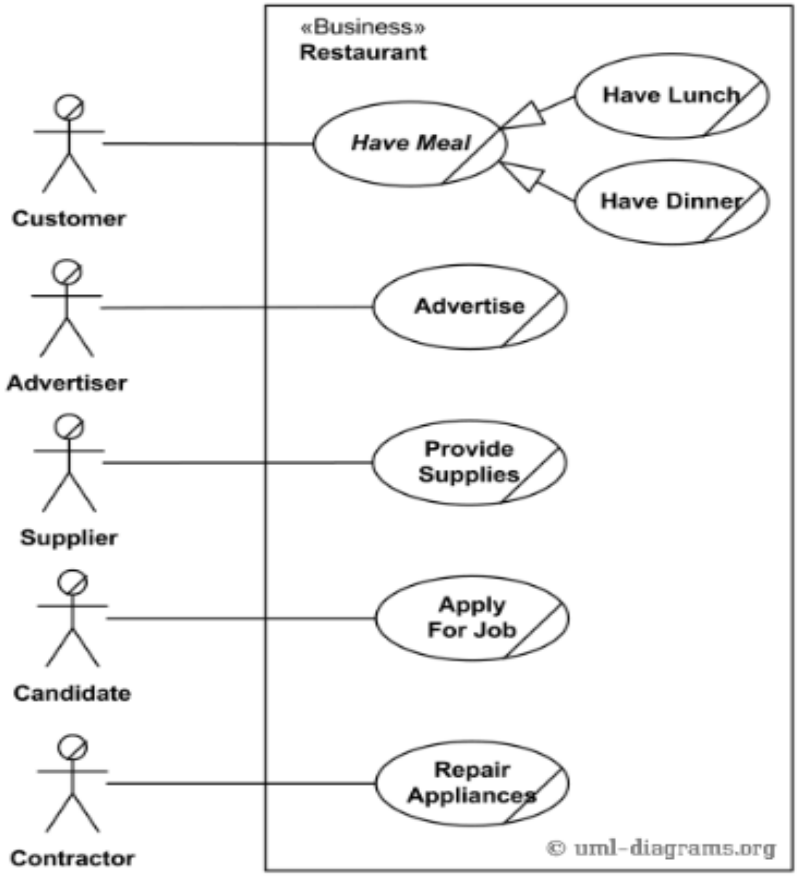
- Title: "goal the use case is trying to satisfy"
- Main Success Scenario: numbered list of steps
  - Step: "a simple statement of the interaction between the actor and a system"

Restaurant

UML Use Case Diagram Example

Here we provide two alternative examples of a **business use case** diagram for a Restaurant, rendered in a notation used by **Rational Unified Process** (RUP).

First example shows **external business view** of a restaurant. We can see several **business actors** having some needs and goals as related to the restaurant and **business use cases** expressing expectations of the actors from the business.



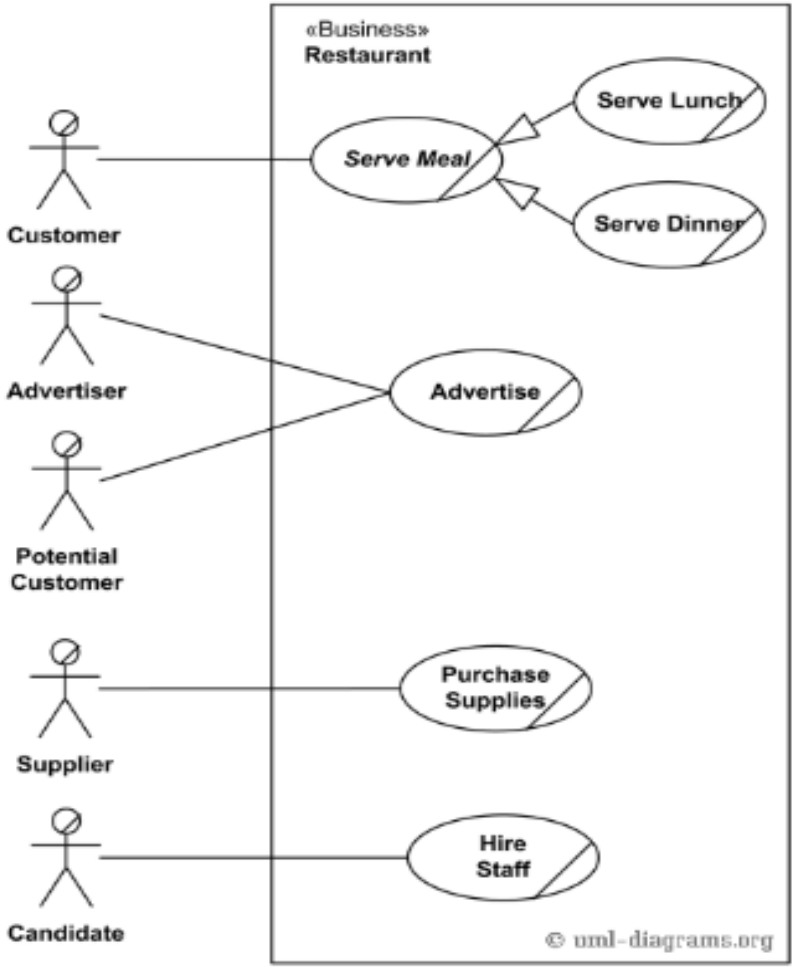
Business use case diagram for Restaurant - External view

For example, **Customer** wants to **Have Meal**, **Candidate** - to **Apply for Job**, and **Contractor** - to fix some appliances. Note, that we don't have such actors as **Chef** or **Waiter**. They are not external roles but part of the business we model - the Restaurant, thus - they are not actors. In terms of RUP Chef and Waiter are **business workers**.

Second example shows **internal business view** of a restaurant. In this case we can see that restaurant has several business processes represented by **business use cases** which provide some services to external **business actors**. As in the previous example, actors have some needs and goals as related to the restaurant.

This approach could be more useful to model services that the business provides to different types of customers, but reading this kind of business use case diagrams could be confusing.

For example, **Customer** is now connected to **Serve Meal** use case, **Supplier** - to **Purchase Supplies**. We have now new actor **Potential Customer** participating in **Advertise** use case by reading ads and getting some information about restaurant. At the same time, **Contractor** actor is gone because **Repair Appliances** is not a service usually provided by restaurants.



Business use case diagram for Restaurant - Internal view

Still, in this example we don't have actors as **Chef** or **Waiter** for the same reasons as before - they both are not external roles but part of the business we model.

Requirements

- Non- Functional Requirements
  - Ties Use Cases to Technical Implementation
  - Describes **HOW** the system should behave
  - A non-functional requirement for the cup mentioned previously would be:

    “contain hot liquid without heating up to more than 45 ° C”.
  - These requirements impact the users’ experience
  - For example, “The home page must load within 1.4 seconds.”
  - Or, “If a document is flagged as private, only the user who created it can see it.”

- The definition of a non-functional requirement is:

  “ Any requirement which specifies **how** the system performs a certain function.”

Requirements

Examples of non-functional requirements

- **Interface requirements**
  - Field 1 accepts numeric data entry.
  - Field 2 only accepts dates before the current date.
  - Screen 1 can print on-screen data to the printer.

- **Regulatory/Compliance Requirements**
  - The database will have a functional audit trail.
  - The system will limit access to authorized users.