Software Engineering for WWW Dr. Vinod Dubey SWE642 George Mason University

JSON+AJAX+TypeScript

GEORGE MASON UNIVERSITY

OBJECTIVES

- **After completing this section, you should be able to**
  - Use JSON – JavaScript Object Notation in Ajax calls
  - Demonstrate an understanding of Ajax technology
  - Demonstrate the knowledge of XMLHttpRequest object - methods and properties
  - Create Ajax applications to retrieve data from server

JSON (JavaScript Object Notation)

- JSON is a lightweight **data interchange format** commonly used for transmitting data between a server and a web application, and it's easy for humans to read and write.
- It's a versatile and **widely used format**, especially in web development.
- **JSON Syntax**
  - JSON uses **key-value pairs** to represent data.
  - **Data** is enclosed in **curly braces {}**
  - **Keys and values** are **separated** by a **colon :**
  - **Key-value pairs** are separated by **commas ,**
  - **Keys** must be **strings**, enclosed in **double quotes "**
  - **Values** can be **strings**, **numbers**, **booleans**, **null**, **arrays**, or other JSON objects.

```
{
  "name": "John Doe",
  "age": 30,
  "isStudent": false
}
```

JSON (JavaScript Object Notation)

- **JSON Data Types**
  - **String**: A sequence of characters enclosed in double quotes.
  - **Number**: An integer or floating-point number.
  - **Boolean**: Either true or false.
  - **Null**: Represents the absence of a value.
  - **Array**: An ordered list of values, enclosed in square brackets [].
  - **Object**: A collection of key-value pairs, enclosed in curly braces {}.

```
{
  "name": "John Doe",
  "age": 30,
  "isStudent": false
}
```

- **Creating JSON object:** To create a JSON object, **define key-value pairs** within **curly braces {}**
- **JSON Object Array**
  - Represented in JSON with **square brackets** containing a **comma-separated list of objects**
  - Each value in a JSON array can be a string, a number, a JSON representation of an object, `true`, `false` or `null`

JSON - JavaScript Object Notation

- **Accessing JSON Data**: In JavaScript, you can access JSON data using **dot notation** or **square bracket** notation.

```
const data = {
  "name": "John Doe",
  "age": 30,
  "isStudent": false
};

console.log(data.name);  // Output: John Doe
console.log(data["age"]);  // Output: 30
```

- **Modifying JSON Data**: You can modify JSON data by **assigning** new **values** to the desired **key**.

```
data.name = "Jane Smith";
data["age"] = 35;
```

JSON - JavaScript Object Notation

- **Converting JSON to a String:** To transmit or store JSON data, you need to convert it to a string using **JSON.stringify()**

```
const jsonString = JSON.stringify(data);
```

- **Converting a String to JSON:** To work with JSON data received as a string, you can convert it to a JavaScript object using **JSON.parse()**

```
const jsonObject = JSON.parse(jsonString);
```

- **JavaScript `eval` function**
  - Can convert JSON strings into JavaScript objects
  - To evaluate a JSON string properly, a left parenthesis should be placed at the beginning of the string and a right parenthesis at the end of the string before the string is passed to the `eval` function
  - A secure way to process JSON is to use a JSON parser

JSON - JavaScript Object Notation

- **JSON and AJAX**
  - **JSON** is **often used** in combination **with AJAX** to **retrieve data** from a server asynchronously.
  - You can make an AJAX request to a server and receive the response in JSON format.
  - Example using the fetch API:

```
fetch('https://example.com/data.json')
  .then(response => response.json())
  .then(data => {
    // Work with the JSON data
  })
  .catch(error => {
    // Handle any errors
  });
```

AJAX – Asynchronous JavaScript and XML

- AJAX is a technique used **to send** and **retrieve data** from a server **asynchronously without reloading** the entire web page. It enables building **dynamic** and **interactive web applications**.
  - It enables **asynchronous communication** between a web browser and a server, allowing data to be retrieved, sent, and processed in the background, without disrupting the user experience.
  - It allows you to **update parts of a web page** without requiring a full page reload.
  - While the name implies XML, modern AJAX implementations often use **JSON as the preferred data format** due to its simplicity and efficiency.
  - Modern JavaScript frameworks, such as **jQuery, Axios**, and Fetch API, provide higher-level abstractions and helper functions to simplify the process of making AJAX requests and handling server responses.

AJAX – Asynchronous JavaScript and XML

- **Here's how AJAX typically works:**
  - **XMLHttpRequest**: The XMLHttpRequest object is a built-in browser object that allows you to send and receive data asynchronously from a server. It allows JavaScript to make HTTP requests to a server and handle the server's responses asynchronously.
  - **Asynchronous Requests**: With AJAX, you can send **asynchronous requests** to the server **without blocking** the execution of other JavaScript code or requiring a page reload. This means that users can continue interacting with the page while data is being fetched or processed in the background.
  - **Event-driven Programming**: AJAX relies on event-driven programming, where you define **callbacks** or **event handlers** that are triggered when certain events occur, such as the completion of an AJAX request or the receipt of a server response.
  - **Updating the DOM**: Once the server responds with data, you can use JavaScript to dynamically **update the web page's content (e.g., displaying the retrieved data) without refreshing the entire page**.

History of Ajax

- The term Ajax was coined by Jesse James Garrett of Adaptive Path in February 2005
- **Ajax technologies**
  - HTML, JavaScript, CSS, the DOM, JSON, and XML have existed for many years
  - In 1998, Microsoft introduced the **`XMLHttpRequest`** object to create and manage asynchronous requests and responses.
  - The XMLHttpRequest object is now a built-in browser object.
- Popular applications that use the `XMLHttpRequest` to update pages dynamically
  - Flickr, Google's Gmail, and Google Maps
- Ajax has quickly become one of the hottest technologies in Web development

XMLHttpRequest object properties

Property	Description
`onreadystatechange`	Stores the callback function—the event handler that gets called when the server responds.
`readyState`	Keeps track of the request's progress. It is usually used in the callback function to determine when the code that processes the response should be launched. The `readyState` value `0` signifies that the request is uninitialized; `1` signifies that the request is loading; `2` signifies that the request has been loaded; `3` signifies that data is actively being sent from the server; and `4` signifies that the request has been completed.
`responseText`	Text that is returned to the client by the server.
`responseXML`	If the server's response is in XML format, this property contains the XML document; otherwise, it is empty. It can be used like a `document` object in JavaScript, which makes it useful for receiving complex data (e.g. populating a table).
`status`	HTTP status code of the request. A `status` of `200` means that request was successful. A `status` of `404` means that the requested resource was not found. A status of `500` denotes that there was an error while the server was proccessing the request.
`statusText`	Additional information on the request's status. It is often used to display the error to the user when the request fails.

`XMLHttpRequest` object methods

Method	Description
`open`	Initializes the request and has `two` mandatory parameters—method and URL. The method parameter specifies the purpose of the request—typically `GET` if the request is to take data from the server or `POST` if the request will contain a body in addition to the headers. The URL parameter specifies the address of the file on the server that will generate the response. A third optional boolean parameter specifies whether the request is asynchronous—it's set to `true` by default.
`send`	Sends the request to the sever. It has one optional parameter, `data`, which specifies the data to be POSTed to the server—it's set to `null` by default.

`XMLHttpRequest` object methods

Method	Description
`setRequestHeader`	Alters the header of the request. The two parameters specify the header and its new value. It is often used to set the `content-type` field.
`getResponseHeader`	Returns the header data that precedes the response body. It takes one parameter, the name of the header to retrieve. This call is often used to determine the response's type, to parse the response correctly.
`getAllResponseHeaders`	Returns an array that contains all the headers that precede the response body.
`abort`	Cancels the current request.

| **`XMLHttpRequest`** object methods. (Part 2 of 2.)

Creating Ajax Applications using the `XMLHttpRequest` object

- **Set Up the Development Environment:**
  - Create a new HTML file or use an existing one.
  - Include the necessary JavaScript code in your HTML file.
- **Create an XMLHttpRequest Object:**
  - Create an instance of the XMLHttpRequest object using the new XMLHttpRequest() constructor.
- **Make an Asynchronous Request:**
  - Use the XMLHttpRequest object to make an asynchronous HTTP request to the server.
  - Use the open() method to specify the HTTP method (GET, POST, etc.) and the URL of the server-side script or API endpoint.
  - Use the send() method to send the request to the server. For POST requests, include the request payload as an argument to the send() method.
- **Handle the Server Response:**
  - Register an event handler function to handle the response from the server.
  - Use the onreadystatechange event and check the readyState property of the XMLHttpRequest object to determine the current state of the request.
  - Inside the event handler function, check if the readyState is 4 (indicating that the request is complete) and the status is 200 (indicating a successful response).
  - Access the response data using the responseText property of the XMLHttpRequest object.

Creating Ajax Applications using the `XMLHttpRequest` object

- **Display the Response:**
  - Update the HTML content of your web page to display the retrieved data.
  - Use JavaScript to access and modify the HTML elements.
  - For example, you can use the innerHTML property to update the content of a specific element.
- **Handle Errors:**
  - Implement error handling to handle failed AJAX requests or server errors.
  - Check the status property of the XMLHttpRequest object to determine the status code of the response.
  - Display appropriate error messages or handle errors gracefully.
- **Send Data to the Server:**
  - AJAX requests can also send data to the server.
  - For POST requests, include the request payload in the send() method as a string, typically in the format of URL-encoded parameters or JSON data.
- **Enhance User Experience:**
  - Use AJAX to perform dynamic updates without page reloads.
  - Implement features like auto-complete, live search, or real-time data updates.
  - Update specific parts of the web page selectively, without reloading the entire page.

Creating Ajax Applications using the `XMLHttpRequest` object

- **Cross-Origin Resource Sharing (CORS):**
  - Understand CORS if you're making AJAX requests to a different domain or port.
  - Ensure that the server supports and allows cross-origin requests.
  - Set appropriate CORS headers on the server-side to enable access from different origins.
- **Testing and Debugging:**
  - Use browser developer tools to inspect AJAX requests and responses.
  - Debug and troubleshoot any issues that may arise.
  - Validate and verify the data exchanged between the client and server.

An Ajax application example

- A user **interacts** with the page by **moving the mouse over** book-cover images
- The `onmouseover` and `onmouseout` events when the user moves the mouse over and out of an image.
- The `onmouseover` event calls function `getContent` with the URL of the document containing the book's description
- The function makes this request asynchronously using an `XMLHTTPRequest` object
- When the `XMLHTTPRequest` object receives the response, the book description is displayed below the book images
- When the user moves out of the image, the `onmouseout` event calls function `clearContent` to clear the display box
- These tasks are accomplished **without reloading the entire page** on the client

An Ajax application example

```
8  <head>
9     <style type="text/css">
10       .box { border: 1px solid black;
11              padding: 10px }
12    </style>
13    <title>Switch Content Asynchronously</title>
14    <script type = "text/javascript" language = "JavaScript">
15       <!--
16       var asyncRequest; // variable to hold XMLHttpRequest obj
17
18       // set up and send the asynchronous request
19       function getContent( url )
20       {
21          // attempt to create the XMLHttpRequest and make the
22          try
23          {
24             asyncRequest = new XMLHttpRequest(); // create req
25
26             // register event handler
27             asyncRequest.onreadystatechange = stateChange;
28             asyncRequest.open( 'GET', url, true ); // prepare
29             asyncRequest.send( null ); // send the request
30          } // end try
31          catch ( exception )
32          {
33             alert( 'Request failed.' );
34          } // end catch
35       } // end function getContent
36
37       // displays the response data on the page
38       function stateChange()
39       {
40          if ( asyncRequest.readyState == 4 && asyncRequest.status == 200 )
41          {
42             document.getElementById( 'contentArea' ).innerHTML =
43                asyncRequest.responseText; // places text in contentArea
44          } // end if
45       } // end function stateChange
46
47       // clear the content of the box
48       function clearContent()
49       {
50          document.getElementById( 'contentArea' ).innerHTML = '';
51       } // end function clearContent
52       // -->
```

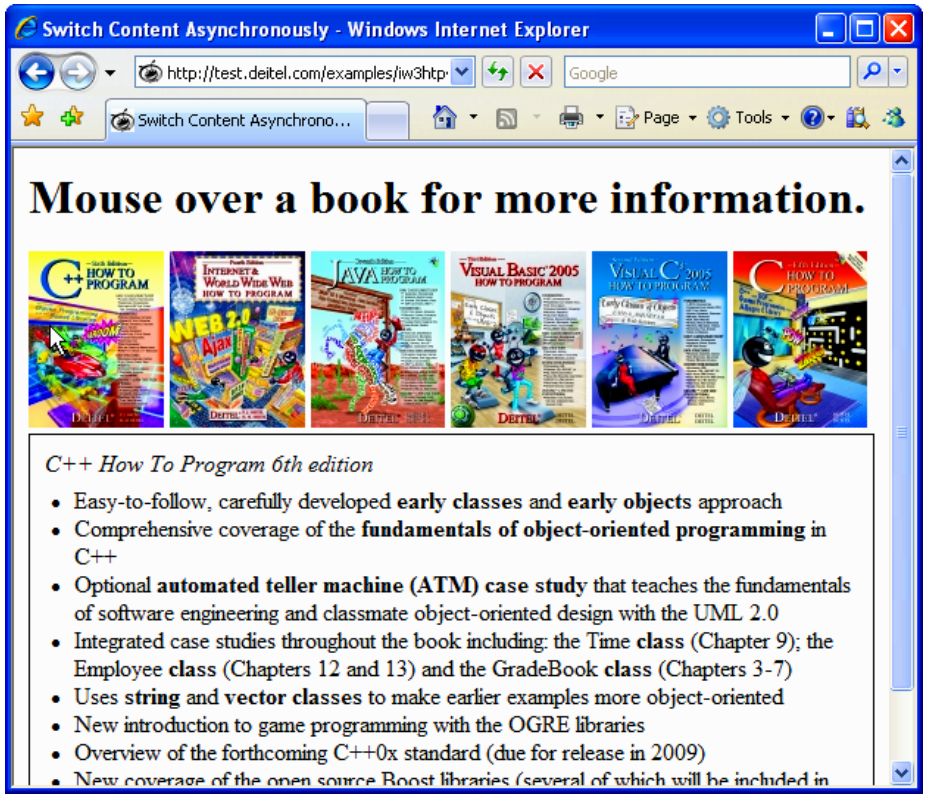
An Ajax application example

```
53    </script>
54 </head>
55 <body>
56    <h1>Mouse over a book for more information.</h1>
57    <img src =
58       "http://test.deitel.com/examples/iw3htp4/ajax/thumbs/cpphtp6.jpg"
59       onmouseover = 'getContent( "cpphtp6.html" )'
60       onmouseout = 'clearContent()'/>
61    <img src =
62       "http://test.deitel.com/examples/iw3htp4/ajax/thumbs/iw3htp4.jpg"
63       onmouseover = 'getContent( "iw3htp4.html" )'
64       onmouseout = 'clearContent()'/>
65    <img src =
66       "http://test.deitel.com/examples/iw3htp4/ajax/thumbs/jhtp7.jpg"
67       onmouseover = 'getContent( "jhtp7.html" )'
68       onmouseout = 'clearContent()'/>
69    <img src =
70       "http://test.deitel.com/examples/iw3htp4/ajax/thumbs/vbhtp3.jpg"
71       onmouseover = 'getContent( "vbhtp3.html" )'
72       onmouseout = 'clearContent()'/>
73    <img src =
74       "http://test.deitel.com/examples/iw3htp4/ajax/thumbs/vcsharphtp2.jpg"
75       onmouseover = 'getContent( "vcsharphtp2.html" )'
76       onmouseout = 'clearContent()'/>
```

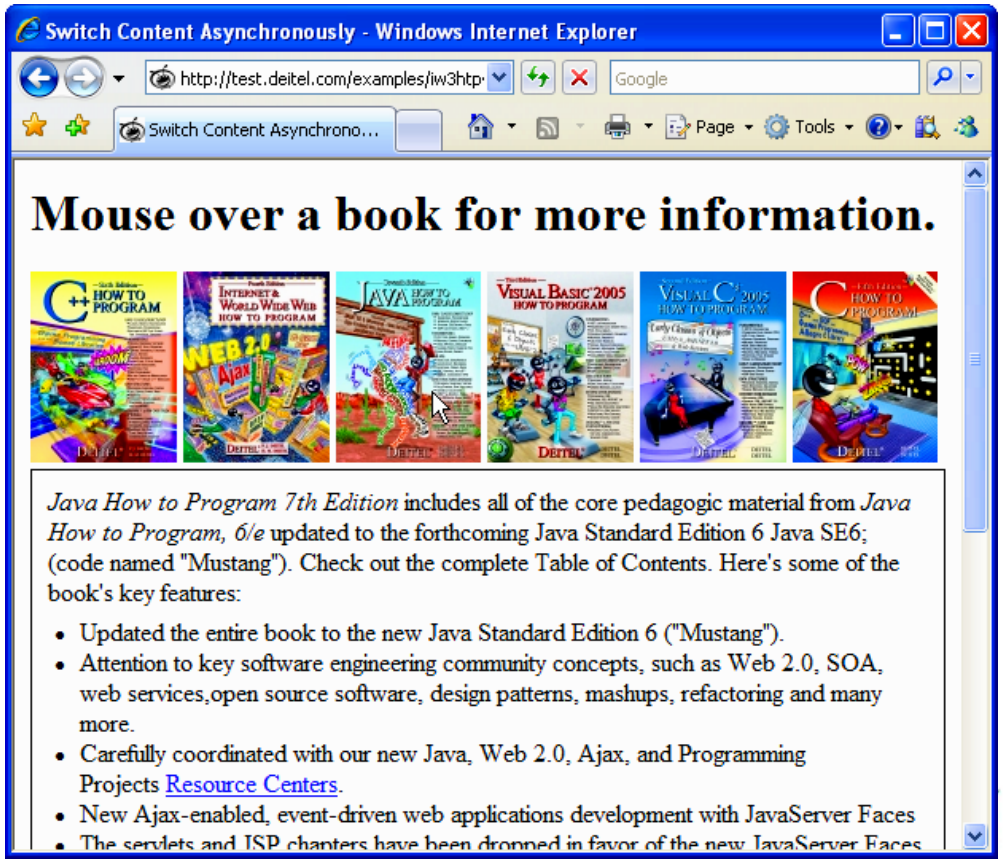
```
77    <img src =
78       "http://test.deitel.com/examples/iw3htp4/ajax/thumbs/chtp5.jpg"
79       onmouseover = 'getContent( "chtp5.html" )'
80       onmouseout = 'clearContent()'/>
81    <div class = "box" id = "contentArea"> </div>
82 </body>
83 </html>
```

An Ajax application example

a) User hovers over *C++ How to Program* book cover image, causing an asynchronous request to the server to obtain the book's description. When the response is received, the application performs a partial page update to display the description.



b) User hovers over *Java How to Program* book cover image, causing the process to repeat.

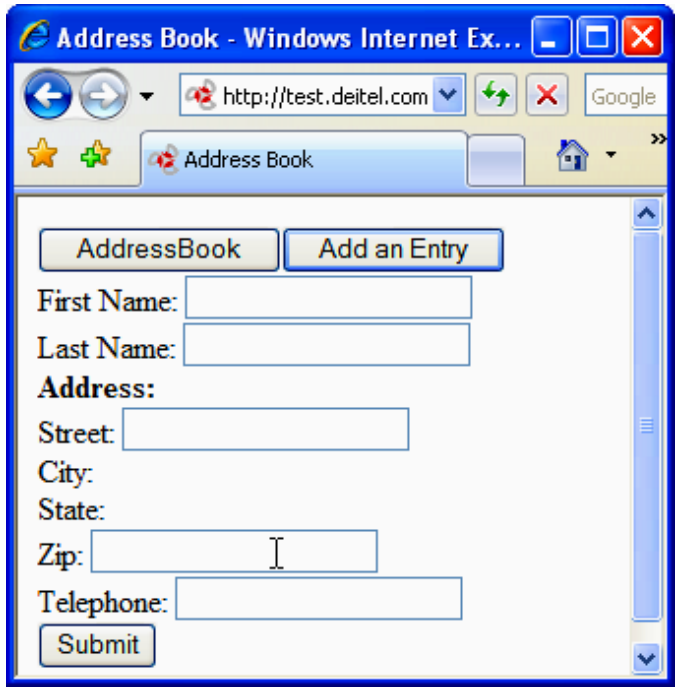


Another Ajax example: **Automated retrieval of city and state based on zip**

- This Web application **interacts with a web service** to **obtain data** to implement **automated retrieval** of **city** and **state** based on **zip**
- This uses an element's **onblur** event handler to make asynchronous requests
- The web application and server communicate using JSON data format
- When a request is sent using the GET method
  - Parameters are concatenated to the URL
  - URL parameter strings start with a ? symbol and have a list of *parameter-value* bindings, each separated by an &

Another Ajax example: **Automated retrieval** of **city** and **state** based on **zip**

```
353  <div id = "addEntry" style = "display : none">
354     First Name: <input id = 'first'/>
355     <br/>
356     Last Name: <input id = 'last'/>
357     <br/>
358     <strong> Address: </strong>
359     <br/>
360     Street: <input id = 'street'/>
361     <br/>
362     City: <span id = "city" class = "validator"></span>
363     <br/>
364     State: <span id = "state" class = "validator"></span>
365     <br/>
366     Zip: <input id = 'zip' onblur = 'validateZip( this.value )'/>
367     <span id = "validateZip" class = "validator">
368     </span>
369     <br/>
370     Telephone:<input id = 'phone'
371        onblur = 'validatePhone( this.value )'/>
372     <span id = "validatePhone" class = "validator">
373     </span>
374     <br/>
375     <input type = "button" value = "Submit"
376        onclick = "saveForm()" />
377     <br/>
378     <div id = "success" class = "validator">
379     </div>
380  </div>
```




```
206  // send the zip code to be validated and to generate city and state
207  function validateZip( zip )
208  {
209     // build parameter array
210     var params = '[{"param": "zip", "value": "' + zip + '"}]';
211     callWebService ( "validateZip", params, showCityState );
212  } // end function validateZip
213
```

Another Ajax example: **Automated retrieval** of **city** and **state** based on **zip**

```
14      // URL of the web service
15      var webServiceUrl = '/AddressBookWebService/AddressService.asmx';
```

```
31      // send the asynchronous request to the web service
32      function callWebService( method, paramString, callBack )
33      {
34         // build request URL string
35         var requestUrl = webServiceUrl + "/" + method;
36         var params = paramString.parseJSON();
37
38         // build the parameter string to add to the url
39         for ( var i = 0; i < params.length; i++ )
40         {
41            // checks whether it is the first parameter and builds
42            // the parameter string accordingly
43            if ( i == 0 )
44               requestUrl = requestUrl + "?" + params[ i ].param +
45                  "=" + params[ i ].value; // add first parameter to url
46            else
47               requestUrl = requestUrl + "&" + params[ i ].param +
48                  "=" + params[ i ].value; // add other parameters to url
49         } // end for
50
```

```
51         // attempt to send the asynchronous request
52         try
53         {
54            var asyncRequest = new XMLHttpRequest(); // create request
55
56            // set up callback function and store it
57            asyncRequest.onreadystatechange = function()
58            {
59               callBack( asyncRequest );
60            }; // end anonymous function
61
62            // send the asynchronous request
63            asyncRequest.open( 'GET', requestUrl, true );
64            asyncRequest.setRequestHeader("Accept",
65               "application/json; charset=utf-8" );
66            asyncRequest.send(); // send request
67         } // end try
68         catch ( exception )
69         {
70            alert ( 'Request Failed' );
71         } // end catch
72      } // end function callWebService
```

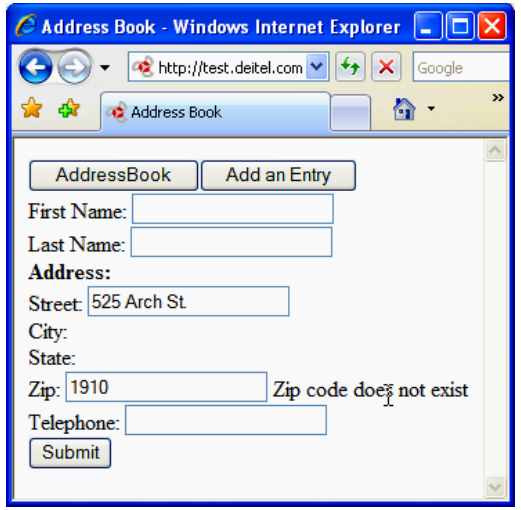

MASON

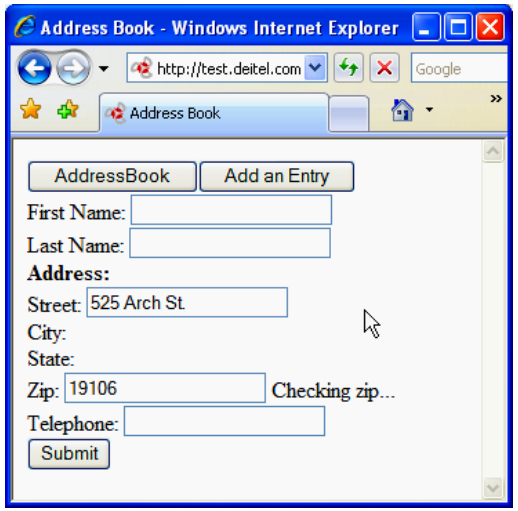
Another Ajax example: **Automated retrieval** of **city** and **state** based on **zip**

```
214     // get city and state that were generated using the zip code
215     // and display them on the page
216     function showCityState( asyncRequest )
217     {
218        // display message while request is being processed
219        document.getElementById( 'validateZip' ).
220           innerHTML = "Checking zip...";
221
222        // if request has completed successfully, process the response
223        if ( asyncRequest.readyState == 4 )
224        {
225           if ( asyncRequest.status == 200 )
226           {
227              // convert the JSON string to an object
228              var data = asyncRequest.responseText.parseJSON();
229
230              // update zip code validity tracker and show city and sta
231              if ( data.Validity == 'Valid' )
232              {
233                 zipValid = true; // update validity tracker
234
235                 // display city and state
236                 document.getElementById( 'validateZip' ).innerHTML = '';
237                 document.getElementById( 'city' ).innerHTML = data.City;
238                 document.getElementById( 'state' ).
239                    innerHTML = data.State;
240              } // end if
241              else
242              {
243                 zipValid = false; // update validity tracker
244                 document.getElementById( 'validateZip' ).
245                    innerHTML = data.ErrorText; // display the error
246
247                 // clear city and state values if they exist
248                 document.getElementById( 'city' ).innerHTML = '';
249                 document.getElementById( 'state' ).innerHTML = '';
250              } // end else
251           } // end if
252           else if ( asyncRequest.status == 500 )
253           {
254              document.getElementById( 'validateZip' ).
255                 innerHTML = 'Zip validation service not avaliable';
256           } // end else if
257        } // end if
258     } // end function showCityState
```

Another Ajax example: Automated retrieval of city and state based on zip

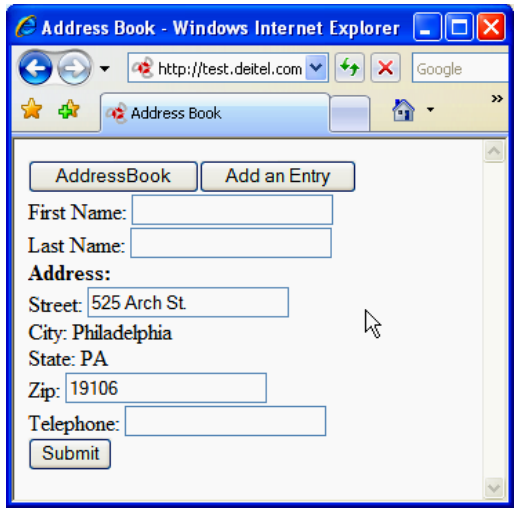
g) User types in a nonexistent zip code. An error is displayed.

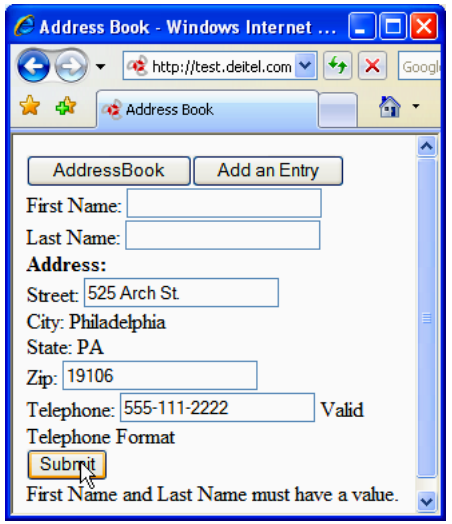




h) User enters a valid zip code. While the server processes it, **Checking Zip...** is displayed on the page.

i) The server finds the city and state associated with the zip code entered and displays them on the page.





j) The user enters a telephone number and tries to submit the data. The application does not allow this, because the First Name and Last Name are empty.

GMU642_M7_LU1_Screencast1_NodeJS_v1

TypeScript Primer

https://www.typescriptlang.org/docs/
https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html

GEORGE MASON UNIVERSITY

OBJECTIVES

- **After completing this section, you should be able to**
  - Understand **key features of TypeScript**
  - Create TypeScript functions
  - Use TypeScript Interfaces and Classes

TypeScript

- TypeScript is an **open-source programming language** developed and maintained by **Microsoft**.
- It is a **superset of JavaScript**, meaning that any valid JavaScript code is also valid TypeScript code.
- **TypeScript adds** optional **static typing** and **additional features** to JavaScript, providing developers with tools for building large-scale, maintainable applications.
- **Angular 2** and later versions **use TypeScript**

MASON

TypeScript

- **Key features of TypeScript include:**
  - **Static Typing**: TypeScript introduces static typing, allowing you to specify types for variables, function parameters, and return values.
    - This helps catch errors during development and provides better tooling support, such as code completion and refactoring.
  - **Type Inference**: TypeScript can **infer the types of variables** and expressions based on their usage. This means that **you don't always have to explicitly annotate types**, as TypeScript can often determine them automatically.
  - **Object-oriented Programming**: TypeScript supports object-oriented programming concepts like **classes, interfaces, inheritance, and access modifiers**.
  - **ECMAScript Compatibility**: TypeScript is designed to align with the evolving ECMAScript (JavaScript) standards.
  - **Advanced Tooling**: TypeScript comes with a **rich set of developer tools**, including a **compiler** that translates TypeScript code to JavaScript, code editors with intelligent autocompletion and code navigation.

TypeScript

- **TypeScript is a superset of JavaScript.**
  - It supports all JavaScript syntax and **adds additional features** for **type checking** and **static types**.
  - Here's a simple TypeScript code snippet:

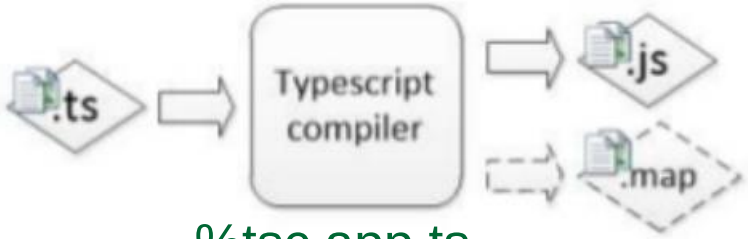
```
function greet(name: string) {
  console.log("Hello, " + name + "!");
}

greet("John");
```

  - In the above example,
    - we define a function greet that takes a parameter name of type string.
    - The function then logs a greeting message to the console.
  - TypeScript was developed by Microsoft.

TypeScript

- **TypeScript is written in .ts file**
  - Create a new TypeScript file with a .ts extension, e.g., app.ts
  - TypeScript files contain TypeScript code that will be compiled to JavaScript.
  - You can compile a TypeScript file (app.ts) using TypeScript compiler **tsc**, using %**tsc app.ts**
  - This command will compile the TypeScript file into JavaScript and generate an app.js file.



%tsc app.ts

  - You can run the compiled JavaScript file with Node.js (%node app.js)
    - You should see the output "Hello, John!" in the consol.
  - You can also include it in an HTML file and open it in a web browser.

TypeScript

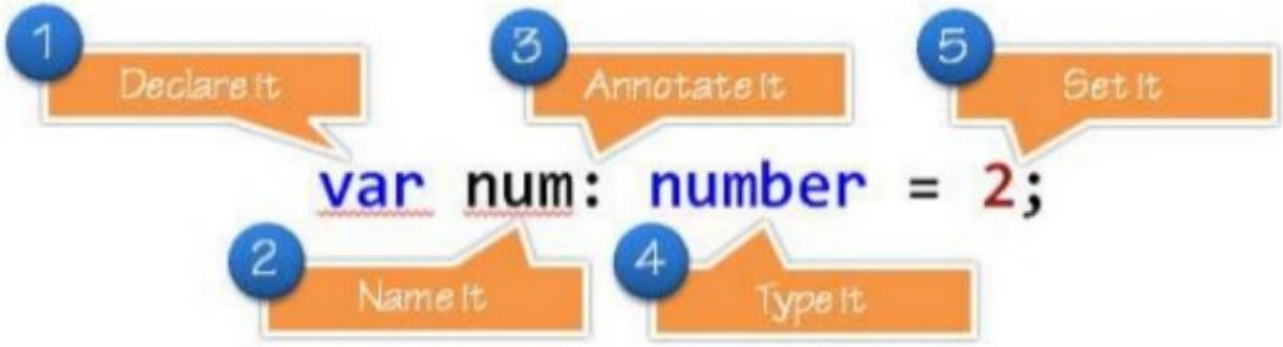
- **TypeScript Features**

  - Type annotations
  - Type inference
  - Compile time type checking
  - Optional, default and rest parameters
  - Classes
  - Interfaces
  - Structural typing
  - Arrow function expressions
  - Enums
  - Generics
  - Modules
  - Tuple types
  - Union types and type guards

MASON

TypeScript

- **TypeScript Grammar**



- **let** and **const** are two **new types** of variable declarations in JavaScript.
- **let** is similar to **var** in some respects
- **const** is an augmentation of **let** in that it **prevents re-assignment** to a variable

Typescript

- **There are two main ways to get the TypeScript tools:**
  - Via **npm** (the Node.js package manager)
  - By installing TypeScript's Visual Studio plugins

- **<u>Visual Studio Code includes TypeScript by default.</u>**

- **For NPM users:**
  - npm install -g typescript

MASON

Building your first TypeScript file

- In your editor, type the following JavaScript code in `greeter.ts`

```
function greeter(person) {
    return "Hello, " + person;
}

let user = "Jane User";

document.body.textContent = greeter(user);
```

- **We used a .ts extension, but this code is just JavaScript.**

MASON

Type Annotations

- We can take advantage of some of the new tools TypeScript offers.
- Let's add a `: string` **type annotation to the 'person'** function **argument**

```
function greeter(person: string) {
    return "Hello, " + person;
}

let user = "Jane User";

document.body.textContent = greeter(user);
```

- Type annotations is used to record the **intended contract** of the function or variable.

MASON

Type Annotations

- In this case, we intend the `greeter` function to be called with a single string parameter.
- Try changing the call to greeter to pass an array instead

```
function greeter(person: string) {
    return "Hello, " + person;
}

let user = [0, 1, 2];

document.body.textContent = greeter(user);
```

- Re-compiling, you'll now see an error:

```
error TS2345: Argument of type 'number[]' is not assignable to parameter of type 'strin
g'.
```

- Similarly, removing all the arguments to the greeter call gives error
  - that the function called with an unexpected number of parameters.

TypeScript Functions Params

- TypeScript functions allow **optional** and **default parameters**

Functions

optional param

```
function buildName(firstName: string, lastName?: string)
{
    if (lastName)
        return firstName + " " + lastName;
    else
        return firstName;
}
```

default param

```
function buildName(firstName: string, lastName = "Doe")
{
    return firstName + " " + lastName;
}
```

TypeScript Types

- **Built-In types**
  - string
  - number
  - boolean
  - Date
  - Array
  - any
- **Custom types**

- **TypeScript Types Annotations**

```
name: string;
age: number;
isEnabled: boolean;
pets: string[];
accessories: string | string[];
```

MASON

TypeScript Types

- **TypeScript Types enforces compile time errors**

JavaScript

```
var a = 54
a.trim()
```

TypeError: undefined is not a function

runtime…

TypeScript

```
var a: string = 54
a.trim()
```

Cannot convert 'number' to 'string'

compile-time!

TypeScript Interfaces

- **TypeScript interfaces provide a code contract**

```
interface Person {
    firstName: string;
    lastName: string;
}
```

- **An example of a valid satisfied contract**

```
let user = { firstName: "Jane", lastName: "User" };
```

MASON

TypeScript Interfaces

- **An example of using the interface in function**

```
interface Person {
    firstName: string;
    lastName: string;
}
```

```
function greeter(person: Person) {
    return "Hello, " + person.firstName + " " + person.lastName;
}

let user = { firstName: "Jane", lastName: "User" };

document.body.textContent = greeter(user);
```

TypeScript Class

- **TypeScript supports class-based object-oriented programming.**
- **Let's create a Student class with a constructor and a few public fields.**

```
class Student {
    fullName: string;
    constructor(public firstName: string, public middleInitial: string, public lastNam
e: string) {
        this.fullName = firstName + " " + middleInitial + " " + lastName;
    }
}
```

  – Note, the use of public on arguments to the constructor is a shorthand that allows us to automatically create properties with that name.

TypeScript Interface and Class

- **In TypeScript, the two types (i.e., Interface and Class) are compatible if their internal structure is compatible.**
  - This allows us to implement an interface just by having the shape the interface requires, without an explicit implements clause.

```
interface Person {
    firstName: string;
    lastName: string;
}
```

```
class Student {
    fullName: string;
    constructor(public firstName: string, public middleInitial: string, public lastNam
e: string) {
        this.fullName = firstName + " " + middleInitial + " " + lastName;
    }
}
```

TypeScript Interface and Class

- **In TypeScript, Interface and Class are compatible if their internal structure is compatible**

```
class Student {
    fullName: string;
    constructor(public firstName: string, public middleInitial: string, public lastNam
e: string) {
        this.fullName = firstName + " " + middleInitial + " " + lastName;
    }
}

interface Person {
    firstName: string;
    lastName: string;
}

function greeter(person: Person) {
    return "Hello, " + person.firstName + " " + person.lastName;
}

let user = new Student("Jane", "M.", "User");

document.body.textContent = greeter(user);
```

Backups