- JavaScript Dynamic website development

Dynamic website development - Dynamically updating parts of the web page without refreshing the entire webpage. The entire webpage is not reloaded. Only some of the webpage is updated. Technologies used to create dynamic websites. Javascript DOM, Single Page Application (SPA) development ie: Ract, Angular, Vue, Other.
Some websites can be dynamic ie: 80%-90% dynamic - 10-20% static content, 30% dynamic, 70% static etc..
Examples of dynamic web pages include:
User interaction on webpage, mouse events ie: move, click, other, on page load update screen, display date/time, e-commerce websites, Responsive websites, navigation menu, Large number of webpages, large number of custom website, ie: shopify store, shopping cart, animation, dynamically displaying ads on a webpage from an api call, online games, online apps, other

Javascript DOM - Document object model -

programming interface that allows you to interact with and manipulate the elements of an HTML or XML document.
Think of it as a tree-like representation of the document, where each element, attribute, and text content is a node in the tree.
This structure allows you to access and modify these nodes using JavaScript

SPA - React -

It's a collection of pre-written JavaScript code that you can use to build your own applications.

User interfaces:
It's specifically designed for building the visual part of your application, the part that the user interacts with.
Component-based:

React encourages you to break down your UI into small, reusable components. This makes your code easier to manage and maintain.
Declarative:

You tell React what you want your UI to look like, and it takes care of updating the DOM (Document Object Model) efficiently.

JavaScript (often abbreviated as JS) is a programming language that is one of the core technologies of the World Wide Web, alongside HTML and CSS. It is used to make web pages dynamic and interactive.

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

https://www.w3schools.com/js/js_htmldom.asp

DOM introduciton
events - call code base on event: ie: onclick, onload ... etc..
select 1 or more elements in html document, dicplay info on screen
build new dom element dynamically
examples:
page validation - contact, login, regsiter, other
dynamically updating webpage
display date time, counter
shopping cart app
todo list app
drawing on screen
javascript games
other

With the object model, JavaScript gets all the power it needs to create dynamic HTML:

JavaScript can change all the HTML elements in the page
JavaScript can change all the HTML attributes in the page
JavaScript can change all the CSS styles in the page
JavaScript can remove existing HTML elements and attributes
JavaScript can add new HTML elements and attributes
JavaScript can react to all existing HTML events in the page
JavaScript can create new HTML events in the page

Here's a basic overview of JavaScript:

Purpose:
JavaScript allows you to implement complex features on web pages, such as:
Displaying timely content updates
Creating interactive maps
Animating 2D/3D graphics
Building scrolling video jukeboxes
Handling user interactions (e.g., button clicks, form submissions)
Making asynchronous requests to servers
Syntax:

JavaScript's syntax is similar to C and Java, making it relatively easy to learn for those familiar with these languages.
Features:

Object-oriented: JavaScript supports object-oriented programming (OOP) with object prototypes and classes.

Functional: JavaScript also supports functional programming, treating functions as first-class objects.
Dynamic: JavaScript is a dynamically typed language, meaning the type of a variable is determined at runtime.

How it works:
JavaScript code is executed by the browser's JavaScript engine, which is built into every modern web browser.
Example:
Here's a simple JavaScript code snippet that displays an alert message:
JavaScript

```
alert("Hello, World!");
```

Where to use it: JavaScript is primarily used for client-side web development, but it can also be used in other environments, such as:
Node.js: A JavaScript runtime environment for building server-side applications
React Native: A framework for building native mobile apps using JavaScript
Electron: A framework for building cross-platform desktop apps using JavaScript

REST API, or Representational State Transfer Application Programming Interface, is a type of API that uses HTTP requests to allow applications to communicate with each other. REST APIs are often used in web and mobile app development.
How does REST API work?
A client sends a request to a server
The server responds to the request
The client and server can only interact in this way
What are the principles of REST API?

Uniform interface: REST APIs have a consistent interface
Stateless: REST APIs are stateless, meaning they don't store information about previous requests

Cacheable: REST APIs can be cached
Client-server: REST APIs are client-server based, meaning the client initiates all interactions
Layered system: REST APIs are layered systems

What are some examples of REST APIs?

OpenWeatherMap is a REST API that provides weather data based on a user's location
What are some types of APIs?

Open API: A public API that anyone can access
Partner API: A restricted API for business partners and clients
Private API: An internal API used within a company
Composite API: An API that combines data and services to streamline tasks

Node.js is an open-source JavaScript runtime environment that allows developers to run JavaScript code outside of a web browser. It's used for server-side programming, and is often used to build back-end services.

https://www.w3schools.com/nodejs/

Features of Node.js
Cross-platform
Node.js can run on many operating systems, including macOS, Linux, and Windows

Lightweight
Node.js is efficient and scalable, and can handle thousands of concurrent connections
Non-blocking
Node.js uses asynchronous I/O primitives to prevent JavaScript code from blocking

Event-driven
Node.js is built on event-driven programming, which allows it to handle requests without blocking the thread

How Node.js works
Node.js runs the V8 JavaScript engine, which is also used in Google Chrome
Node.js apps run in a single process, without creating a new thread for each request
Node.js uses an event loop to handle requests, allowing it to move on to the next task while waiting for a response

Uses of Node.js
Building web applications
Building back-end services, such as APIs
Powering client applications, such as web apps and mobile apps
Who uses Node.js?
Many big businesses, including Amazon, Netflix, eBay, Reddit, and PayPal

Javascript API Summary

REST API, or Representational State Transfer Application Programming Interface, is a type of API that uses HTTP requests to allow applications to communicate with each other. REST APIs are often used in web and mobile app development.

common http verbs used to maintain data from another website
- GET - retrieve data
- POST - send data
- PUT - update data
- DELETE - delete data

API Data format
- Json - Javascript Object Notation
- {key:, value:}
- example: {id:1, name:"item1"}
- nested json data
- example: [{id:1, name:"item1",["item1","item2"]}]

- Other data format

  - xml, text, binary ie: images, video

Browser libraries used to manage Api data from another website

Preview api - Browser Devtools - Network `

- Ajax -
  - https://www.w3schools.com/Js/js_ajax_intro.asp
  - legacy way to manage api data from another website
  - flexable, intermediate code
  - does not support promise, async/await
  - custom code to implement promise, async/await
- Jquery
  - https://www.w3schools.com/jquery/

  - legacy plugin (via url reference), way to manage api data from another website
  - flexable, minimal code
  - does not support promise, async/await
  - custom code to implement promise, async/await
- Fetch
  - https://www.w3schools.com/js/js_api_fetch.asp
  - https://medium.com/@turingvang/fetch-api-example-86ddf525fa69
  - modern library manage api data from another website
  - supported by all browsers ie: es5+ - 2014+
  - flexable, minimal code
  - supports, promise, async/await

Other api libraries, axios

```
#Mobile applications use api for send/receive, update data

#Api producer websites

- produce api data
  - get, post, put, delete
  - most websites has an api producer feature
  - json data returned is determined by the producer website
  - documents which describes and provides examples of json api data
  - require signup and apikey to get, post, update, delete
  - used for product/website data integration

  - producer api website example:
    - jssonplaceholder.org -> get, test: post, put, delete
    - giphy.com -> get
    - many api producer website
    - ie: facebook, amazon, google, other
    - sports, gmail

- Create your own producer api using technologies
          -  nodejs/express, python, other

- consumer websites
  - any website ca consume api
  - get, post, put, delet
```

callback functions or ananymous functions - week 7-8-9

- learn more from w3school.com
- function getdata(data, statue) -- developer call function
    - some function use callback/ananymous function -- javascript calls callback ananymous

```
//callback functions or anonymous functions - week 7-8-9
//learn more from w3school.com
//example:
// function getdata(data, statue) -- developer call function
// or
// callback or anonymous
// some function use callback/anonymous function -- javascript calls callback/anonymous
// -- when the data is available javascript call the callback/anonymous function
//
//function my_callback_function(data, status)
//{
//}
//
//$.get(_api_url_endpoint_users, my_callback_function)
//
//$.getJSON(_api_url_endpoint_users, function(data, status)
```

Json object introduction

- https://www.w3schools.com/js/js_json.asp
- https://www.w3schools.com/js/js_json_intro.asp

API Data format

- Json - Javascript Object Notation
- {key:, value:}
- example: {id:1, name:"item1"}
- valid, invalid json documents
    - array json documents []
    - object json documents {}
    - nested array, object json documents
    - example: [{},{}]
    - example: {[],[]}
- parsing json document,
    - json string to object
    - json object to string
- nested json documents

Json document ie: NOSQL , or table and rows in a database

- example:
- array [] rows
- {} 1 row of columns or fields
- Root or starting element can be a [] or {}

Users json document example ie: contact us or register data

-1 document

{"id":1,"name":"ann","email":"ann@gmail.com"}

-or-

- multiple documents

```
[
{"id":1,"name":"ann","email":"ann@gmail.com"},
{"id":2,"name":"sam","email":"sam@yahoo.com"}
]
```

-or- sami-structured document - null or missing data

- custom code to validate null and missing data

```
[
{"id":1,"email":"ann@gmail.com"},
{"id":2,"name":"sam","email":null}
]
```

-or- nested document example 1##

```
[
{"id":1,"name":"ann","email":"ann@gmail.com","address":[""street":"...","city":"..."]},
{"id":2,"name":"sam","email":"sam@yahoo.com"}
]
```

-or- nested document example 2##

```
[
{"id":1,"name":"ann","email":"ann@gmail.com","address":[""street":"...","city":"..."]},
{"id":2,"name":"sam","email":"sam@yahoo.com","address":{"street":"..."}}]
```

-- access fields/keys/elements in a json document

example #1

```
const user = {"id":1,"name":"ann","email":"ann@gmail.com"}

console.log(user.name)

//display json data as tring
```

```
console.log(JSON.stringify(user))

example #2

users = [
{"id":1,"name":"ann","email":"ann@gmail.com"},
{"id":2,"name":"sam","email":"sam@yahoo.com"}
]

//display json data as tring
console.log(JSON.stringify(users))

console.log(users[0].id)
console.log(users[0].name)
console.log(users[1].id)
console.log(users[1].name)
```

--- for loop

– json document errors: null, missing data, incomplete - [], {}, fields, “” commas,

example json document files

- comments, posts, users
- giphy result document

example api website

- https://www.jsonplaceholder.org/

Giphy website

- https://giphy.com/

demo access giphy api data

- https://developers.giphy.com/
- https://developers.giphy.com/explorer/

Choose an app / API Key
default dashboard: .....api.key.....

Choose a resource
default
GIPHY PUBLIC API

Choose an endpoint
search

Request URL

https://api.giphy.com/v1/gifs/search?api_key=.........&q=cars&limit=25&offset=0&rating=g&lang=en
&bundle=messaging_non_clips

Result
... json document ...

api data get summary demo #1

use browser dev tools to preview api request/response

example:
 single request
   request api data from website
   website return api data

 multiple api request

```
    request api data from website
    website return api data
    website return api data
    website return api data
```

```
multiple api request
    request api data from website
    request api data from website
        website return api data
    request api data from website
    website return api data
    website return api data
```

* best practice: return nested data in 1 request instead of multiple request/response, requires less api calls

* nested loops to display nested data returned

```
 - for
   - for
```

errors
- timeout when sending, receiving request

- send/receive data from different systems ie: distributed systems

api request demo websites , low to high api requests
- websites with ads
- streaming websites
- gaming websites
- ecommerce website ie: retrieve product and pricing information
- other

https://www.w3schools.com/js/default.asp
- preview api request using browser devtools network tab
- filter fetch xhr
- preview data returned and where data is stored ie: localstorage

##

api data returned example #1

result1
  [.....]

result2 - nested result
  [...[..]..]

result2 - deeply nested result
  [...[..]..[]...]

promise, async/await programming introduction

- Learn more at W3schools

```
# browser devetools - simulate network traffic and offline
  - from browser dev tools - select network
  - throttle network ie: 3g, 46, offline
```

some javasccript functions use
callback <- called by js, or developer
- ie: nested callback can be complex

parameter in function

$.get(praram1,function(){})

----
promise
----

modern js uses instead of callback
js promise - non blocking code
nested function chanining
- ie: nested callback can be complex

```
%.get().then()
{
}
.then()
{
}
.then()
{
}
.error()
{
}
```

----
async/await
----

async = define non blocking function
await - wait until data is avaliable

code can continue until awaited data is available

```
async getdata()
{
  result = await %.get()
}
```

See also:

Promise.all
Promise.allsettled