



Module 4 Day 5

Asynchronous Programming in JS

What makes an application?

- Program Data

- ✓ Variables & .NET Data Types
- ✓ Arrays
- ✓ More Collections (list, dictionary, stack, queue)
- ✓ Classes and objects (OOP)

- Program Logic

- ✓ Statements and expressions
- ✓ Conditional logic (if)
- ✓ Repeating logic (for, foreach, do, while)
- ✓ Methods (functions / procedures)
- ✓ Classes and objects (OOP)
- ✓ Frameworks (MVC)

- Input / Output

- User
 - ✓ Console read / write
 - ✓ HTML / CSS
 - ❖ Front-end frameworks (HTML / CSS / JavaScript)
- Storage
 - ✓ File I/O
 - ✓ Relational database
 - ❖ APIs

Definitions

Term	Definition
API	<p>Application Programming Interface. A set of functions and procedures allowing other applications to access the features or data of an operating system, application, or other service.</p> <p>An API may be for a web-based system, operating system, database system, computer hardware, or software library.</p>
Web API or Web Service	<p>A service (API) offered by an application or device to another application, which communicate with each other via the World Wide Web (typically HTTP).</p>
REST	<p>Representational State Transfer. One style of web service which utilizes HTTP features (statelessness, Request and Response, and GET, PUT, POST and DELETE methods) for accessing and updating data.</p>
Consumer	<p>A user of web services. This is software, not a person.</p>
Endpoint	<p>The "location" at which a service's features (methods / data) can be accessed. For a web service, this is usually a URL.</p>

Asynchronous Programming

- Start an operation, but don't wait for completion before moving on
- When we start the operation, specify "what to do" when the operation completes
 - The function returns us a "Promise"
 - The Promise is "pending"
- Move on to do more work while the operation is taking place
- When the operation finishes, the above "what to do" code is called
 - The Promise is "fulfilled"

fetch-ing Data

```
JS fetchtxt.js ▶ ...
1  // filename: fetchtxt.js
2  fetch('demo.txt')           // sends an HTTP request to the relative path 'demo.txt'
3  .then((response) => {       // get a Response object once this completes
4      response.text()         // Call async function to get the text from the response
5      .then( (txt) => {       // get a string once that completes
6          console.log(txt);   // log the string data
7          document.getElementById('results').innerText = txt
8      })
9  });
```

```
11 fetch('demo.txt')           // sends an HTTP request to the relative path 'demo.txt'
12 .then((response) => {       // get a Response object once this completes
13     return response.text(); // Call async function to get the text from the response
14 }).then((txt) => {          // get a string once that completes
15     console.log(txt);       // log the string data
16     document.getElementById('results').innerText = txt
17 });
```

Let's
Code

fetch and catch

- "catch" catches network errors, NOT bad http status codes.
- You must check the response code

```
fetch('demoxxx.txt')           // sends an HTTP request to the relative path 'demo.txt'
  .then((response) => {         // get a Response object once this completes
    if (response.ok) {
      response.text()           // Call async function to get the text from the response
      .then( (txt) => {         // get a string once that completes
        console.log(txt);      // log the string data
        document.getElementById('results').innerText = txt
      })
    }
    else
    {
      console.log(`BAD STATUS CODE: ${response.status} ${response.statusText}`)
    }
  }).catch ( (err) => {
    console.log(`There was an ERROR: ${err}`);
  });
```

Let's
Code

Fetches are not just for GETs

- You can POST, PUT and DELETE too
- You need to create a [Request](#) object for some commands (POST, PUT)

Method		URL	Req Body	Resp Body	Status
GET	Retrieve a list	/parks	-	Json []	200 ok
GET	Retrieve an object	/parks/{id}	-	Json { }	200 ok
POST	Insert	/parks	Json { }	(empty) or Json { }	201 created
PUT	Update	/parks/{id}	Json { }	(empty) or Json { }	204 no content Or 200 ok