Front-End Web Development

Unit 13: Building a Web Application with JavaScript

Course Outline

Course Learning Outcomes

- Competently write HTML and CSS code
- Create web page layouts according to requirements using styles
- Add interactivity to a web page with JavaScript
- Access and display third-party data on the web page
- Leverage Bootstrap and Static Site Generator

Assessment

- Final Project - 100% of the grade

  - Design and Build functioning Website using HTML5, CSS (including Bootstrap), JavaScript (browser only)

  ✓ Code will be managed in GitHub
  ✓ Website will be deployed to GitHub Pages
  ✓ All code to follow best practice and be documented

  - Details and How-To-Guide are available on the course page under the section called Assessments

13. Building a Web Application with JavaScript

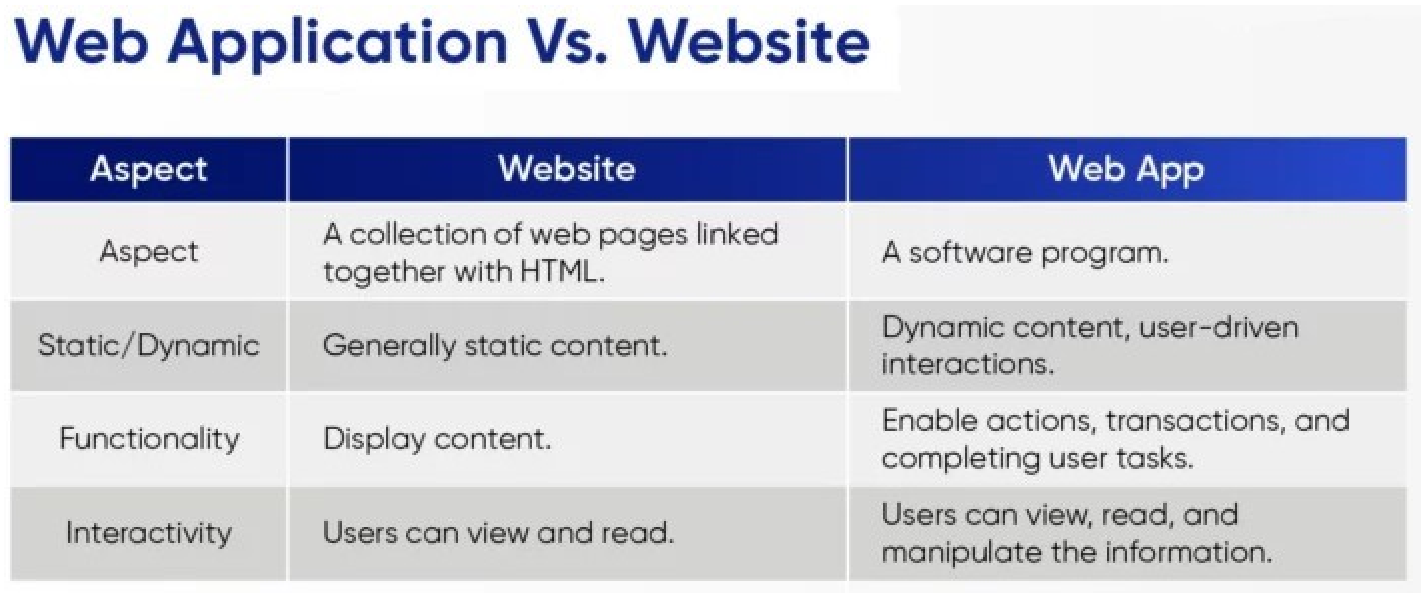
In This Unit

Title
Web Applications and REST
Building a Web Application using Fetch
JavaScript Modules

Web Applications and REST

What is a Web Application

A web application is software that runs in your web browser. The most common website features like shopping carts, product search and filtering, instant messaging, and social media newsfeeds are web applications in their design. They allow you to access complex functionality without installing or configuring software



Web Application Vs. Website

Aspect	Website	Web App
Aspect	A collection of web pages linked together with HTML.	A software program.
Static/Dynamic	Generally static content.	Dynamic content, user-driven interactions.
Functionality	Display content.	Enable actions, transactions, and completing user tasks.
Interactivity	Users can view and read.	Users can view, read, and manipulate the information.

Benefits of a Web Application

- **Accessibility**
  Web apps can be accessed from all web browsers and across various personal and business devices
- **Efficient development**
  As detailed, the development process for web apps is relatively simple and cost-effective for businesses. Small teams can achieve short development cycles, making web applications an efficient and affordable method of building computer programs. Only a single version is required and will work across all modern browsers and devices
- **User simplicity**
  Web apps don't require users to download them, making them easy to access while eliminating the need for end-user maintenance and hard drive capacity. Web applications automatically receive software and security updates, meaning they are always up to date and less at risk of security breaches
- **Scalability**
  Businesses using web apps can add users as and when they need, without additional infrastructure or costly hardware. In addition, the vast majority of web applications operate on cloud infrastructure and additional resources (compute and data storage) can be added as and when required including automatically

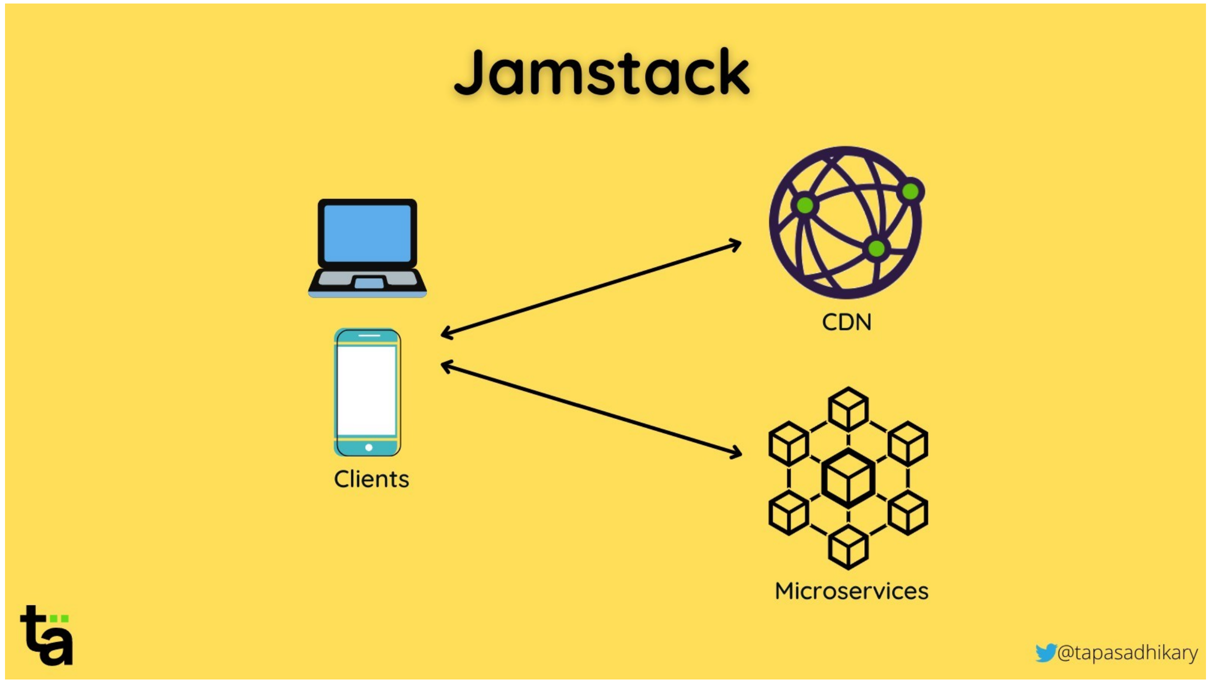
Types of Web Application

- **Static Web Application**
  Static web applications deliver content directly to the user without the need for server interaction. Content has to be modified at the source code level. E.g. small personal blog
- **Dynamic Web Application**
  Dynamic web applications showcase content that can change based on user interactions. They request and display real-time data, making them more interactive and user-friendly. E.g. Content Management System (Wordpress)
- **Single Page Application (SPA)**
  Single web page applications present all information seamlessly on a single HTML page. Users can access all content without navigating to different pages. E.g. GMail, React, Vue, Angular, Swelte
- **Multiple Page Application (MPA)**
  Multi-page web applications consist of interconnected pages accessible through navigation elements. Users can navigate through menus or links to access different sections. E.g. Astro
- **Portal Web Application**
  Portal web applications serve as gateways to various resources and services. They often require user authentication and offer personalized experiences. E.g. LinkedIn
- **E-commerce Web Application**
  E-commerce web applications facilitate online buying and selling of products or services. E.g. Amazon, Shopify

Jamstack

Recall from unit 6, the core principles of the Jamstack architecture are:

- Static Site Generation, e.g. Astro
- Client-side JavaScript
- Reusable APIs, i.e. REST api's
- CDN Delivery



REST

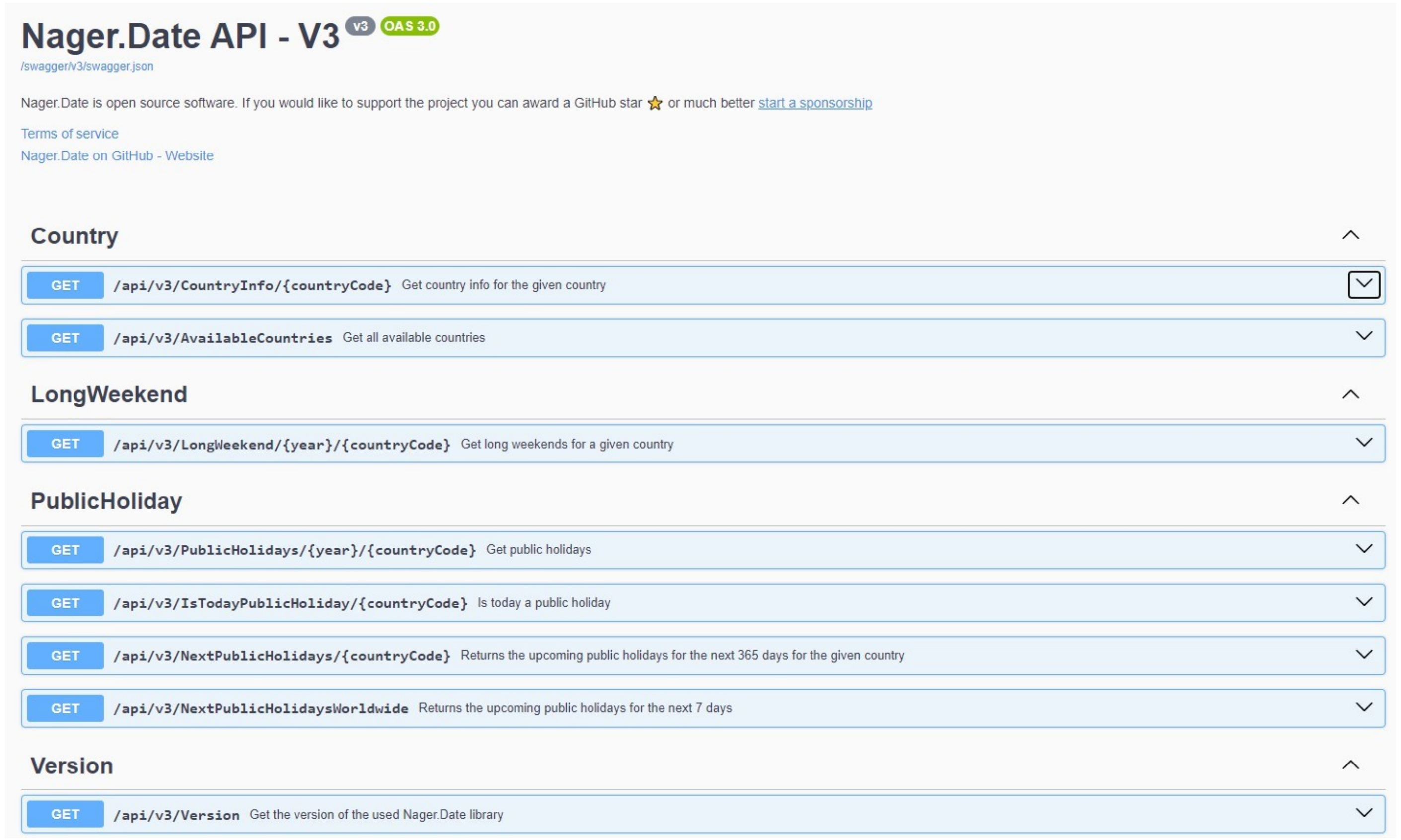
Technically, REST is an acronym for “REpresentational State Transfer” which is simply an architectural style originally written about by Roy Fielding in his doctoral dissertation (https://ics.uci.edu/~fielding/pubs/dissertation/top.htm)

In other words, the caller (or client):

1. Makes an HTTP request to an Endpoint/URL…
   - Using one of the standard HTTP methods (GET, PUT, POST, PATCH, DELETE, etc.)…
   - With some content (usually JSON) in the body…
2. And waits for a response, which:
   - Indicates status via an HTTP response code
   - And usually has more JSON in the body

https://blog.postman.com/rest-api-examples/

Example of REST API - Public Holidays



Nager.Date API - V3 v3 OAS 3.0

/swagger/v3/swagger.json

Nager.Date is open source software. If you would like to support the project you can award a GitHub star ⭐ or much better start a sponsorship

Terms of service

Nager.Date on GitHub - Website

Country

GET `/api/v3/CountryInfo/{countryCode}` Get country info for the given country

GET `/api/v3/AvailableCountries` Get all available countries

LongWeekend

GET `/api/v3/LongWeekend/{year}/{countryCode}` Get long weekends for a given country

PublicHoliday

GET `/api/v3/PublicHolidays/{year}/{countryCode}` Get public holidays

GET `/api/v3/IsTodayPublicHoliday/{countryCode}` Is today a public holiday

GET `/api/v3/NextPublicHolidays/{countryCode}` Returns the upcoming public holidays for the next 365 days for the given country

GET `/api/v3/NextPublicHolidaysWorldwide` Returns the upcoming public holidays for the next 7 days

Version

GET `/api/v3/Version` Get the version of the used Nager.Date library

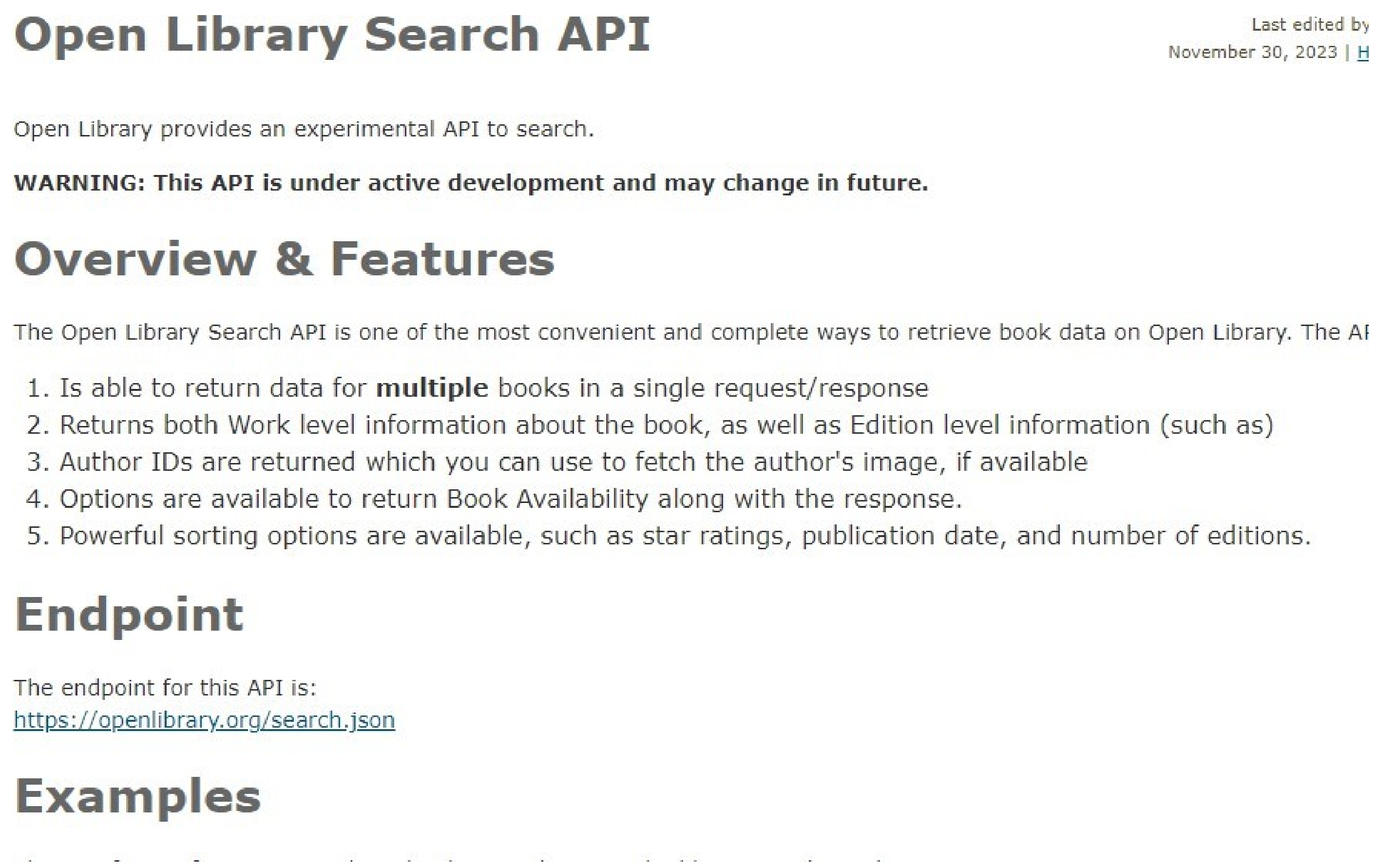
https://date.nager.at/swagger/index.html

Demo

Information on a wide range of public REST api's -> https://www.postman.com/cs-demo/workspace/public-rest-apis

For example:

- https://openlibrary.org/dev/docs/api/search



Open Library Search API

Last edited by
November 30, 2023 | H

Open Library provides an experimental API to search.

WARNING: This API is under active development and may change in future.

Overview & Features

The Open Library Search API is one of the most convenient and complete ways to retrieve book data on Open Library. The AI

1. Is able to return data for **multiple** books in a single request/response
2. Returns both Work level information about the book, as well as Edition level information (such as)
3. Author IDs are returned which you can use to fetch the author's image, if available
4. Options are available to return Book Availability along with the response.
5. Powerful sorting options are available, such as star ratings, publication date, and number of editions.

Endpoint

The endpoint for this API is:
https://openlibrary.org/search.json

Examples

Activity

Breakout

- Join a breakout room
- Download the unit 13 exercises from Moodle
- Follow the instructions and complete the exercises
- You have 35 minutes
- Lecturer will visit each room in turn, etc...
- Will start next topic on the hour

Building a Web Application using Fetch

Building a Simple Web Application

- Demo

Additional factors to consider when building a Web Application

1. Event handling in Component/Class
2. Managing component configuration
    i. passing a JavaScript object to constructor (JS approach)
    ii. configure using data attributes in html (html approach)
3. Multiple component instances
4. State Management

Event Handling in Components

Managing events on a Component class can get very disorganised as the number of events and handlers increases. Adding a `handleEvent()` method to the component/class can simplify matters

The `handleEvent()` method is part of the EventListener API, and has been around for decades

If you listen for an event with the addEventListener() method, you can pass in an object instead of a callback function as the second argument

As long as that object has a handleEvents() method, the event will be passed into it, but `this` will maintain it’s binding to the object

Event Handling in Components

Demo - handle-event-in-class-factorial-using-slider

```
class FactorialSlider {
  componentRoot;
  factSlider;
  outputContainer;

  constructor(componentRoot) {
    this.componentRoot = document.querySelector(componentRoot);
    this.factSlider = this.componentRoot.querySelector('input[name="fact"]');
    this.outputContainer = this.componentRoot.querySelector('.factorial-output-container');
    // event handling for component
    this.componentRoot.addEventListener("change", this);
  }

  handleEvent(event) {
    this.factorial(parseInt(event.target.value));
  }
```

Managing component configuration

Components typically have some initial defaults which the web developer is allowed to specify, e.g. initial country and year for the public holiday component

Two approaches:

1. Specify the configuration using a set of `data-*` attributes in the component HTML
2. Pass a JavaScript object which describes the configuration to the class constructor

Component Configuration - `data-*` attributes

`data-*` attributes allow us to store extra information on standard, semantic HTML elements without other hacks such as non-standard attributes, or extra properties on DOM

Note: not suitable for large number of properties and so may be used in conjunction with the second approach

```
<article
  id="electric-cars"
  data-columns="3"
  data-index-number="12314"
  data-parent="cars">
  …
</article>
```

```
  const article = document.querySelector("#electric-cars");
  // The following would also work:
  // const article = document.getElementById("electric-cars")

  article.dataset.columns; // "3"
  article.dataset.indexNumber; // "12314"
  article.getAttribute("data-parent"); // "cars"
```

Component Configuration - Pass JS object to Class constructor

- Create a JavaScript configuration object with the specific configuration properties for the component
- Pass the configuration object to the class constructor for the component
- Store the configuration object as a property in the component class
- Access the configuration object in the constructor or other class methods to setup the component

```
const comp1 = new ComponentOne(
  { property1: "value1",
    property2: "value2"
  }
)

Class ComponentOne {
  #config;

  Constructor(configuration){
    this.#config = configuration;
  }

  render() { const prop1 = this.#config.property1;}
}
```

Multiple instances of Component

Certain types of component (e.g. carousels) may be defined multiple times on the same web page

It's important to ensure that the component code and property data is self-contained and avoids the use of global variables and objects

State management

State management refers to the practice of handling, storing, and reacting to changes in the data that drives an application’s behavior and UI

https://medium.com/@sudheer.gowrigari/state-management-in-web-components-crafting-cohesive-and-scalable-solutions-f4bbeb6c74d2

Component Local State

Each Component can maintain its internal state, which influences its behavior and rendered output

- Use Class Properties: Utilize class properties to store local state and react to changes
- Leverage Getters and Setters: Implement getters and setters to observe and react to state changes, triggering re-renders or side effects

State Management (Advanced)

Shared State Across Components

When multiple components need to access or modify the same state, a shared state mechanism becomes essential

- Use Custom Events: Propagate state changes using custom events to inform parent components or other listeners
- Implement a Global Store: For larger applications, consider implementing a global state management system or store

Integrating with State Management Libraries

Several state management libraries can be integrated with Components to streamline state handling

- Consider Lightweight Libraries: Libraries like MobX or Zustand can be integrated with Web Components for efficient state management
- Integrate with Redux: While traditionally used with React, Redux can also be employed with Web Components, ensuring a predictable state container

Demo

Activity

Breakout

- Join a breakout room
- Continue working on the unit 13 exercises
- You have 35 minutes
- Lecturer will visit each room in turn, etc...
- Will start next topic on the hour

JavaScript Modules

JavaScript Modules

As JavaScript programs have got larger and more complicated, it made sense to provide a mechanism for splitting JavaScript programs up into separate modules that can be imported when needed

- A module is just a file. One script is one module
- Introduced in ES6(2015)
- Modules can load each other and use special directives `export` and `import` to interchange functionality, call functions of one module from another one:
  - `export` keyword labels variables and functions that should be accessible from outside the current module
  - `import` allows the import of functionality from other modules
- Module file normally use '.js' extension but 'mjs' is also used. GitHub Pages supports '.mjs'

Modules Example (demos/javascript-modules-basic)

```
<!-- index.html -->
<script type="module" src="js/main.js"></script>
```

```
// main.js
import { create, createReportList } from './modules/canvas.js';
import { name, draw, reportArea, reportPerimeter } from './modules/square.js';
import randomSquare from './modules/square.js';

let myCanvas = create('myCanvas', document.body, 480, 320);
let reportList = createReportList(myCanvas.id);

let square1 = draw(myCanvas.ctx, 50, 50, 100, 'blue');
reportArea(square1.length,  reportList);
reportPerimeter(square1.length,  reportList);

// Use the default
let square2 = randomSquare(myCanvas.ctx);
```

```
export { create, createReportList }; // end of canvas.js
export { name, draw, reportArea, reportPerimeter }; // end of square.js
export default randomSquare;
```

export and import

- `export`
  - Can export functions, var, let, const, and classes
  - Needs to be top-level items: for example, you can't use export inside a function
  - Convenient way of exporting all the items you want to export is to use a single export statement at the end of the module file, followed by a comma-separated list of the features you want to export wrapped in curly braces

```
export { create, createReportList }; // end of canvas.js
```

- `import`
  - Once features have been exported out of the module, they then need to import them into the main script to be able to use them. The simplest way to do this is as follows:

```
import { create, createReportList } from './modules/canvas.js'; // main.js
```

Differences between modules and standard scripts

- Can't load the HTML file locally (i.e. with a file:// URL), you'll run into CORS errors due to JavaScript module security requirements. You need to do your testing through a server, e.g. VS Code Preview server
- Modules always work in strict mode. E.g. assigning to an undeclared variable will give an error
- There is no need to use the defer attribute (see <script> attributes) when loading a module script; modules are deferred automatically
- Modules are only executed once, even if they have been referenced in multiple <script> tags
- Module features are imported into the scope of a single script — they aren't available in the global scope. Therefore, you will only be able to access imported features in the script they are imported into, and you won't be able to access them from the JavaScript console, for example. You'll still get syntax errors shown in the DevTools, but you'll not be able to use some of the debugging techniques you might have expected to use
- Due to scoping rules, no need to use IIFE (https://developer.mozilla.org/en-US/docs/Glossary/IIFE)
- Event though `imports` are `hoisted`, it is considered good practice to put all your imports at the top of the code, which makes it easier to analyze dependencies

Default exports versus named exports

- **Named export**

```
export { create, createReportList }; // end of canvas.js
```

- **Default export** - only one default export allowed per module

```
export default randomSquare;
export default function (ctx) { // turns this into an anonymous function
  // …
}

import randomSquare from "./modules/square.js";
```

Renaming imports and exports

Names can be changes using the `as` keyword

```
// inside module.js
export { function1 as newFunctionName, function2 as anotherNewFunctionName };

// inside main.js
import { newFunctionName, anotherNewFunctionName } from "./modules/module.js";
```

```
  // inside module.js
export { function1, function2 };

// inside main.js
import {
  function1 as newFunctionName,
  function2 as anotherNewFunctionName,
} from "./modules/module.js";
```

Creating a module object

An even better solution is to import each module's features inside a module object. The following syntax form does that:

```
import * as Module from "./modules/module.js";
```

This grabs all the exports available inside module.js, and makes them available as members of an object Module, effectively giving it its own namespace. So for example:

```
Module.function1();
Module.function2();
```

Modules and classes

You can also export and import classes; this is another option for avoiding conflicts in your code, and is especially useful if you've already got your module code written in an object-oriented style

```
class Square {
  constructor(ctx, listId, length, x, y, color) { // … }
  draw() { // … }
}
export { Square };
```

Over in main.js, we import it like this:

```
import { Square } from "./modules/square.js";
```

And then use the class to draw our square:

```
const square1 = new Square(myCanvas.ctx, myCanvas.listId, 50, 50, 100, "blue");
square1.draw();
square1.reportArea();
square1.reportPerimeter();
```

<script type="importmap">

- The importmap value of the type attribute of the <script> element indicates that the body of the element contains an import map
- An import map is a JSON object that allows developers to control how the browser resolves module specifiers when importing JavaScript modules
- It provides a mapping between the text used as the module specifier in an import statement or import() operator, and the corresponding value that will replace the text when resolving the specifier
- https://developer.mozilla.org/en-US/docs/Web/HTML/Element/script/type/importmap

```
<!-- html file -->
<script type="importmap">
  {
    "imports": {
      "square": "./module/shapes/square.js",
      "circle": "https://example.com/shapes/circle.js"
    }
  }
</script>
<script> .... </script>
```

Demo

Activity

Breakout

- Join a breakout room
- Continue working on the unit 13 exercises
- You have 35 minutes
- Lecturer will visit each room in turn, etc...

Summary

Completed this Week

- Web Applications and REST
- Building a Web Application using Fetch
- JavaScript Modules

For Next Week

- Complete the remaining exercises for unit 13 before next class
- Review the slides and examples for unit 14