Diploma Web Application Development: Introduction

ICT50220 Diploma of Information Technology(Front-End Web Development)

Code	Title
ICTWEB517	Create web-based programs
ICTWEB546	Validate application design against specifications

Sessions

- A session is a component of study
- Sessions may include:
  - Notes
  - Demonstrations
  - Challenges
  - Out of class activities

Modules

Modularised Code & UX/UI

Maintainability

- manage and update code.
- isolate bugs efficiently.

Reusability

- reuse components across the application.

Organisation

- separate concerns for better clarity.
- makes codebase easier to understand.

Code as reusable components

```
// Example of a reusable button component
function createButton(label, onClick) {
      const button = document.createElement('button');
      button.textContent = label;
      button.addEventListener('click', onClick);
      return button;
}
```

North Metropolitan

Modularised Code & UX/UI

Scalability

- add new features without disrupting existing code.

Consistent UX

- uniform UI elements across the application.
- update styles & behaviour in one place.

Collaboration

- teams can work on modules simultaneously.
- clear boundaries between parts of the application.

Code as reusable components

```
// Example of a modularised DOM update function
export function updateDOM(element, content) {
    element.innerHTML = content;
}
```

Types of Modules?

- **ES6 Modules** (import/export)
- **CommonJS** (require/module.exports)
- **AMD** (Asynchronous Module Definition)

```
// export from one file
export const add = (a, b) => a + b;

// import to another
import { add } from './math.js';
```

Creating & Exporting Modules

Use the export keyword

- **named** exports:
  1. multiple values
  2. curly braces to import.

- **default** exports:
  1. single value
  2. no curly braces.

```
// named
export const add = (a, b) => a + b;
export const subtract = (a, b) => a - b;

// default
export default function greet(name) {
  return `Hello, ${name}!`;
}
```

Importing Modules

Use the import keyword

- **named** exports
  1. use curly braces.
  2. import specific values.
- **default** exports
  1. no curly braces.
  2. can rename during import.
- **combining** imports
  1. import both named & default use aliases if needed.

```
import { add, subtract as myAlias } from
"math.js";

console.log("ES6 Module - Add:", add(2, 3));
// 5
console.log("ES6 Module - Subtract:",
subtract(5, 2)); // 3
```

Activity 1

Debugging Modules

Debugging Modules

We will need to debug & watch the sequence of our JavaScript code when we use modules in both JS & React…

Complete the steps using the *activity_.html*

Why We Use type="module" in our HTML

- tells the browser the script is an ES6 module.
- strict mode is used by default, variables are scoped to that module.
- deferred by default ensuring that the DOM is fully loaded
- we can use import statements to include other module
- we can use await at the top level

Why Refactor

Plan

- identify reusable code.
- create separate files.

Divide

- separation of concerns.
- easier to manage & more scalable

Simplicity & Collaboration

- simplify the main application code.
- improve readability & therefore maintainability.

Common Mistakes

1. each module should have a clear purpose
2. use pneumonic/meaningful names
3. use a consistent naming pattern.
4. ensure modules do not depend on each other in a circular manner.
5. document module functionality.

```
// Module documentation example
/**
* Updates the inner HTML of a given element.
* @param {HTMLElement} element-The element to update.
* @param {string} content - The content to  set.
*/
export function updateDOM(element, content) {
       element.innerHTML = content;
}
```

North Metropolitan

Modularising by Functionality

```
// domManipulation.js
export function updateDOM(element, content) {
        element.innerHTML = content;
}

// eventHandlers.js
export function handleClick(button, callback) {
        button.addEventListener('click', callback);
}

// main.js
import { updateDOM } from './domManipulation.js';
import { handleClick } from './eventHandlers.js’;

const button = document.getElementById('myButton’);
        handleClick(button, () =>
        updateDOM(document.getElementById('content'
), 'Button Clicked!'));
```

Modularising by Functionality

```
// Using existing modules project
import { updateDOM } from './domManipulation.js';
import { handleClick } from './eventHandlers.js';

// New feature
function showAlert(message) {
        alert(message);
}

handleClick(button, () => {
        updateDOM(document.getElementById('content’),'Button Clicked!’);
        showAlert('Button was clicked!');
});
```

TAFE WA North Metropolitan

ICT40120 Certificate IV in Programming

Modularisation Activity

JS

North Metropolitan

Modularisation Activity

In this activity we will:

1. Connect to GitHub classroom
2. Complete the modularisation of the code

Open *modules_exercise_starter*