Week 8 - JQuery

jQuery is a **JavaScript library** designed to simplify common tasks like **DOM manipulation, event handling, AJAX, and animations**.

Its main goal is to let you *write less code and do more*, especially for browser inconsistencies.

Core Idea of jQuery

- Wraps native DOM APIs with a **simpler, chainable interface**
- Handles **cross-browser differences** internally
- Uses **CSS-style selectors** to find elements
- Works on collections of elements, not single nodes

```
$(".box").addClass("active").hide();
```

Basics

`$()` Function

`$()` is a **factory function**, not a constructor.

What it can do:

- Select DOM elements
- Create new elements
- Wrap existing DOM nodes
- Run code when DOM is ready

```
$(selector)
$(htmlString)
```

```
$(DOMElement)
$(function () { /* DOM ready */ })
```

DOM Ready

`$(function () { ... })` runs when the **DOM is fully loaded**, similar to `DOMContentLoaded`.

- Ensures elements exist before manipulation
- Preferred shorthand in jQuery

Content Manipulation

Used to **read or update element content**.

- `text()` → works with plain text (safe, no HTML parsing)
- `html()` → works with HTML (can insert tags)
- `val()` → used for form inputs, selects, textareas

Same method is used for **get** (no argument) and **set** (with argument).

Attribute Handling

`attr()` is used to **get or set HTML attributes**.

- `attr(name)` → get attribute
- `attr(name, value)` → set attribute
- Works for attributes like `href`, `src`, `id`, `title`

DOM Insertion Methods

These methods **insert elements relative to existing elements**.

- `append()` → adds as **last child**
- `prepend()` → adds as **first child**
- `before()` → inserts **before the element**
- `after()` → inserts **after the element**

Inserted content can be:

- HTML string
- DOM element
- jQuery object

Removing Content

- `remove()` → removes the element itself + children
- `empty()` → removes only children, keeps the element

CSS & Class Manipulation

Class-based styling:

- `addClass()`
- `removeClass()`
- `toggleClass()`

Inline styling:

- `css(property)` → get value
- `css(property, value)` → set value
- `css({})` → set multiple properties

Dimension Methods

Used to measure element size.

- `width()` → content only
- `innerWidth()` → content + padding
- `outerWidth()` → content + padding + border
- `outerWidth(true)` → content + padding + border + margin

All values are in **pixels**.

Traversal: Moving Around the DOM

Upwards (Ancestors)

- `parent()` → immediate parent

- `parents()` → all ancestors
- `parentsUntil(selector)` → ancestors until matched element

Downwards (Descendants)

- `children()` → direct children
- `children(selector)` → filtered direct children
- `find(selector)` → all matching descendants

Key difference:

- `children()` → one level deep
- `find()` → any depth

Sideways (Siblings)

- `siblings()` → all siblings
- `next()` / `prev()` → immediate sibling
- `nextAll()` / `prevAll()` → all siblings in one direction
- `nextUntil()` / `prevUntil()` → siblings until matched element

Filtering Matched Elements

Used to **narrow down jQuery collections**.

- `first()` → first element
- `last()` → last element
- `eq(index)` → element at index
- `filter(fn | selector)` → keep matching elements
- `not(fn | selector)` → exclude matching elements

Events

- jQuery passes a **wrapped event object** to handlers
- Most native properties are copied or proxied

- The original browser event is available via `event.originalEvent`
- This wrapper ensures **cross-browser compatibility**

Event Binding Methods

- `on()` → attach event handler (preferred)
- `off()` → remove event handler
- `one()` → attach handler that runs only once
- `trigger()` → fire event programmatically

Event Properties

Property	Description
`type`	Type of event (e.g. `click`)
`target`	Actual element that triggered the event
`currentTarget`	Element where handler is attached
`delegateTarget`	Element handling delegated event
`timeStamp`	Time when event occurred
`namespace`	Event namespace (if used)
`originalEvent`	Native browser event object
`pageX`	Mouse X position relative to page
`pageY`	Mouse Y position relative to page
`which`	Key or mouse button pressed
`metaKey`	Whether Meta/Command key was pressed
`relatedTarget`	Secondary target (mouseenter/leave)
`result`	Last handler return value
`data`	Data passed during event binding

Event Helper Methods

Method	Purpose
`preventDefault()`	Prevent default browser action
`stopPropagation()`	Stop event bubbling
`stopImmediatePropagation()`	Stop bubbling + other handlers

Method	Purpose
`isDefaultPrevented()`	Check if default was prevented
`isPropagationStopped()`	Check if bubbling was stopped
`isImmediatePropagationStopped()`	Check if immediate stop occurred

Programmatic Events

- `trigger()` fires the event **manually**

```
$("#btn").trigger("click");
```

Custom Events

Creating & Listening to a Custom Event

Custom events are attached using `on()` just like normal events.

```
$("#btn").on("customEvent", function (e, param1, param2) {
  // handle custom logic
});
```

Key points:

- First argument is the **custom event name**
- First parameter is always the **jQuery event object**
- Additional parameters are **custom data**

Triggering a Custom Event

Use `trigger()` to fire a custom event.

```
$("#btn").trigger("customEvent", ["value1", "value2"]);
```

JQuery validation plugin

The **jQuery Validation plugin** provides a declarative way to validate form fields on the client side.

It integrates tightly with jQuery's event system and DOM manipulation.

How Validation Works

- You call `.validate()` on a `<form>`
- Rules are attached to form fields (by `name` attribute)
- Validation runs automatically on:
  - `submit`
  - `keyup`
  - `focusout`
- Errors are shown/hidden dynamically
- Form submission is blocked until the form is valid

Required Libraries

- `jquery.min.js` → core dependency
- `jquery.validate.min.js` → main validation engine
- `additional-methods.min.js` → extra built-in rules (optional)

Initializing Validation

```
$("#registerForm").validate({ ... });
```

- Returns a **validator instance**
- This instance controls validation state, errors, and helpers

Validation Rules

Rules are defined using field **name attributes**, not IDs.

```
rules: {
  username: {
    required: true,
```

```
    minlength: 3
  }
}
```

Built-in rules:

Custom Validation Method

Custom rules are added using `$.validator.addMethod()`.

```
$.validator.addMethod("emailRegex", fn, message);
```

Key points:

- `value` → current input value
- `element` → DOM element
- `param` → rule parameter (regex here)
- `this.optional(element)` allows empty values if not required

Custom methods integrate exactly like built-in rules.

Error Messages

Messages can be:

- Global (default)
- Rule-specific
- Field-specific

```
messages: {
  username: {
    required: "Please enter a username"
  }
}
```

If not provided, default messages are used.

Dynamic Rules (Runtime Changes)

Rules can be added or removed after initialization.

```
$("#email").rules("add", { required: true });
$("#username").rules("remove", "minlength");
```

Useful for:

- Conditional fields
- Multi-step forms
- Dynamic UI changes

Submit & Invalid Handlers

`submitHandler(form)`

- Runs when form is **valid**
- Gives full control over submission

```
submitHandler: function (form) {
  form.submit();
}
```

`invalidHandler(event, validator)`

- Runs when form is **invalid**
- Useful for summaries or alerts

Validator Object (Important Methods)

Method	Purpose
`form()`	Validate entire form
`element(selector)`	Validate single field
`valid()`	Check if form is valid

Method	Purpose
`resetForm()`	Clear all errors
`showErrors(errors)`	Show custom errors
`numberOfInvalids()`	Count invalid fields
`destroy()`	Remove validation entirely

Class-based Rules

Rules can be applied to all elements with a class.

```
$.validator.addClassRules("special", {
  required: true,
  minlength: 5
});
```

Useful for reusable validation logic.

Styling Errors

The plugin automatically adds:

- `.error` class to inputs
- `<label class="error">` for messages

Styling is handled purely via CSS.

Debug Mode

```
jQuery.validator.setDefaults({
  debug: true
});
```

- Prevents actual form submission
- Useful during development

Important Notes

- Validation is **client-side only**
- Never trust it without server-side validation
- Rules bind to `name`, not `id`
- Plugin uses jQuery event delegation internally

Summary

The jQuery Validation plugin provides a powerful, extensible, and declarative way to validate forms. It supports built-in rules, custom logic, dynamic rule management, and full programmatic control through the validator instance—making it ideal for complex form workflows in jQuery-based applications.