WEB PROGRAMMING 0601632 2

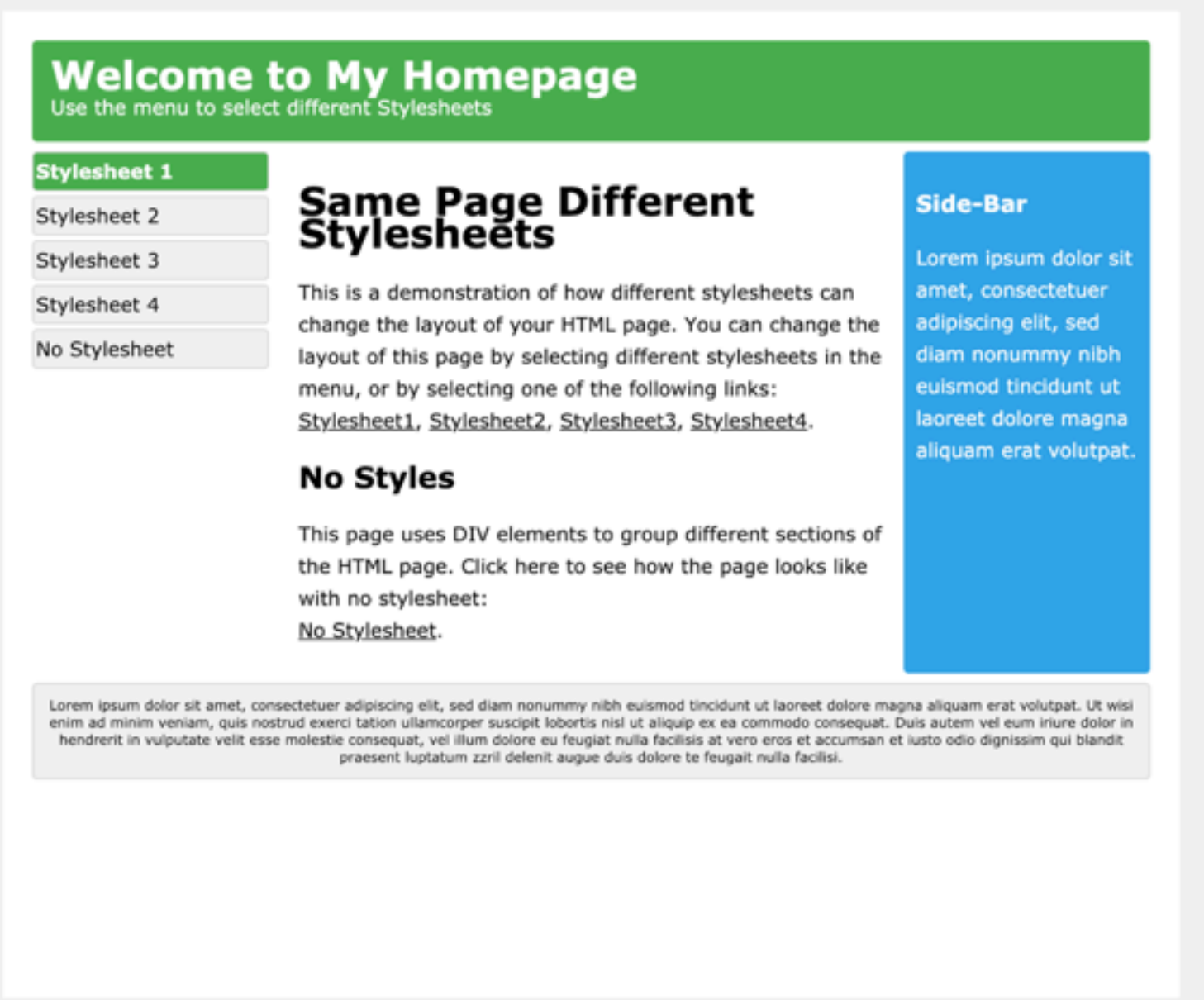
BY DR BUNDIT THANASOPON

CSS REVISION

Src: https://www.W3schools.Com/css/default.Asp

WHAT IS CSS?

- **CSS** stands for **C**ascading **S**tyle **S**heets
- CSS describes **how HTML elements are to be displayed on screen, paper, or in other media**
- CSS **saves a lot of work**. It can control the layout of multiple web pages all at once
- External stylesheets are stored in **CSS files**



GOTO: HTTPS://WWW.W3SCHOOLS.COM/CSS/CSS_INTRO.ASP

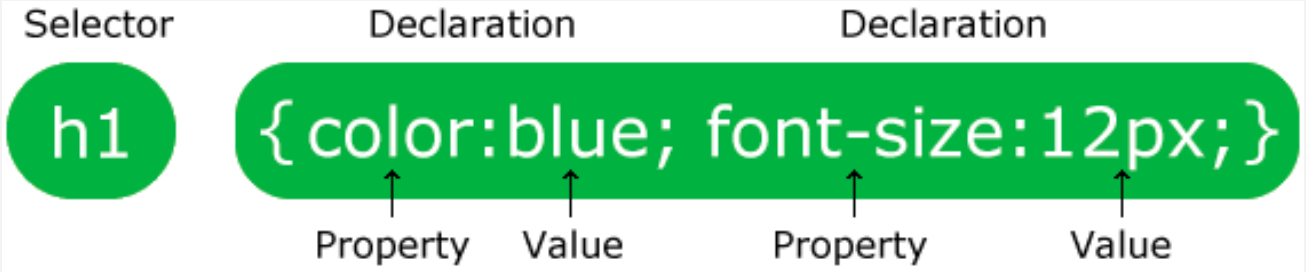
CSS Saves a Lot of Work!

The style definitions are normally saved in external .css files.

With an external stylesheet file, you can change the look of an entire website by changing just one file!

CSS SYNTAX

- CSS Syntax - a CSS rule-set consists of a selector and a declaration block:



- The **selector** points to the HTML element you want to style.
- The declaration block contains one or more **declarations** separated by semicolons.
- Each declaration includes a CSS property **name** and a **value**, separated by a colon.
- A CSS declaration always ends with a semicolon, and declaration blocks are surrounded by curly braces.

CSS SELECTORS

- CSS selectors are used to "find" (or select) HTML elements based on their element name, id, class, attribute, and more.

```css
p {
  text-align: center;
  color: red;
}
```

```css
#para1 {
  text-align: center;
  color: red;
}
/* This is a single-line comment */
```

```css
.center {
  text-align: center;
  color: red;
}
```

```css
p.center {
  text-align: center;
  color: red;
}
```

```css
h1, h2, p {
  text-align: center;
  color: red;
}
```

THREE WAYS TO INSERT CSS

- There are three ways of inserting a style sheet:
  - **External style sheet**

```
<head>
    <link rel="stylesheet" type="text/css" href="mystyle.css">
</head>
```

  - **Internal style sheet**

```
<head>
    <style>
    h1 {
            color: maroon;
            margin-left: 40px;
    }
    </style>
</head>
```

  - **Inline style:** `<h1 style="color:blue;margin-left:30px;">This is a heading</h1>`

BULMA

Https://bulma.lo/documentation/overview/start/

GETTING STARTED WITH BULMA

- There are several ways to **get started** with Bulma. You can either:
  1. use **npm** to install the Bulma package
  2. use the **jsDelivr CDN** to link to the Bulma stylesheet
  3. use the **GitHub repository** to get the latest development version

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Hello Bulma!</title>
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bulma@0.9.1/css/bulma.min.css">
  </head>
  <body>
  <section class="section">
    <div class="container">
      <h1 class="title">
        Hello World
      </h1>
      <p class="subtitle">
        My first website with <strong>Bulma</strong>!
      </p>
    </div>
  </section>
  </body>
</html>
```

MODIFIERS SYNTAX

- Most Bulma elements have alternative styles. To apply them, you only need to append one of the **modifier classes**. They all start with **is-** or **has-**.

Let's start with a simple **button** that uses the `"button"` CSS class:

Button

```
<a class="button">
  Button
</a>
```

By **adding** the `"is-primary"` CSS class, you can modify the **color**:

Button

```
<a class="button is-primary">
  Button
</a>
```

You can also alter the **size**:

- `is-small`
- `is-medium`
- `is-large`

Button

Button

Button

Button

```
<a class="button is-small">
  Button
</a>
<a class="button">
  Button
</a>
<a class="button is-medium">
  Button
</a>
<a class="button is-large">
  Button
</a>
```

RESPONSIVENESS

BULMA IS A **MOBILE-FIRST** FRAMEWORK

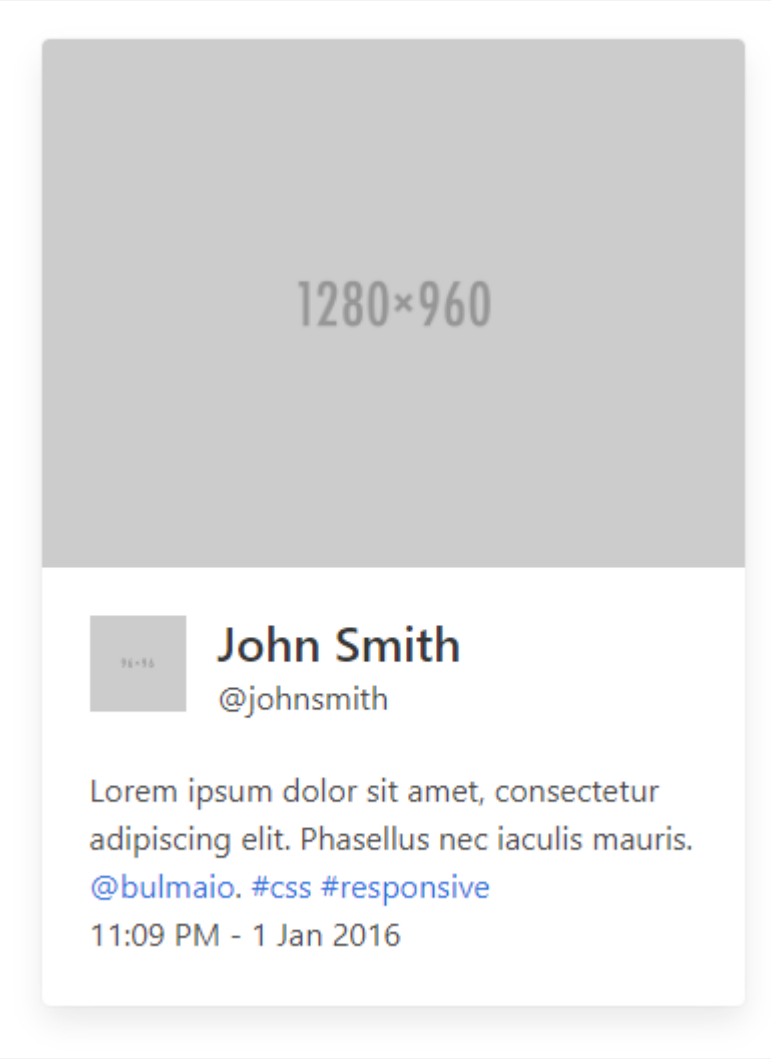
- **Vertical by default**
  - Every element in Bulma is **mobile-first** and optimizes **for vertical reading**, so by default on mobile:
    - **columns** are stacked vertically
    - the **level** component will show its children stacked vertically
    - the **nav** menu will be hidden
- **Breakpoints**
  - Bulma has 5 breakpoints:

- `mobile` : up to `768px`
- `tablet` : from `769px`
- `desktop` : from `1024px`
- `widescreen` : from `1216px`
- `fullhd` : from `1408px`

CONCEPTS YOU SHOULD KNOW

- Colors
- Spacing
- Typography
- Columns
- Layout
- Form
- Often used components

https://bulma.io/documentation/



COLORS

- Text color – 10 colors

Class	Color	Example
has-text-white	hsl(0, 0%, 100%)	Hello Bulma
has-text-black	hsl(0, 0%, 4%)	Hello Bulma
has-text-light	hsl(0, 0%, 96%)	Hello Bulma
has-text-dark	hsl(0, 0%, 21%)	Hello Bulma
has-text-primary	hsl(171, 100%, 41%)	Hello Bulma
has-text-link	hsl(217, 71%, 53%)	Hello Bulma
has-text-info	hsl(204, 86%, 53%)	Hello Bulma
has-text-success	hsl(141, 71%, 48%)	Hello Bulma
has-text-warning	hsl(48, 100%, 67%)	Hello Bulma
has-text-danger	hsl(348, 100%, 61%)	Hello Bulma

- Background color – 10 colors

Class	Background color
has-background-white	hsl(0, 0%, 100%)
has-background-black	hsl(0, 0%, 4%)
has-background-light	hsl(0, 0%, 96%)
has-background-dark	hsl(0, 0%, 21%)
has-background-primary	hsl(171, 100%, 41%)
has-background-link	hsl(217, 71%, 53%)
has-background-info	hsl(204, 86%, 53%)
has-background-success	hsl(141, 71%, 48%)
has-background-warning	hsl(48, 100%, 67%)
has-background-danger	hsl(348, 100%, 61%)

SPACING

- Bulma provides margin **m*** and padding **p*** helpers in all directions:
  - ***t** for top
  - ***r** for right
  - ***b** for bottom
  - ***l** for left
  - ***x** horizontally for both left and right
  - ***y** vertically for both top and bottom
- You need to combine a margin/padding prefix with a direction suffix. For example:
  - for a **margin-top**, use **mt-***
  - for a **padding-bottom**, use **pb-***
  - for both **margin-left and margin-right**, use **mx-***

Suffix	Value
*-0	0
*-1	0.25rem
*-2	0.5rem
*-3	0.75rem
*-4	1rem
*-5	1.5rem
*-6	3rem

TYPOGRAPHY

Size

Class	Font-size
is-size-1	3rem
is-size-2	2.5rem
is-size-3	2rem
is-size-4	1.5rem
is-size-5	1.25rem
is-size-6	1rem
is-size-7	0.75rem

Text weight

Class	Weight
has-text-weight-light	Transforms text weight to **light**
has-text-weight-normal	Transforms text weight to **normal**
has-text-weight-medium	Transforms text weight to **medium**
has-text-weight-semibold	Transforms text weight to **semi-bold**
has-text-weight-bold	Transforms text weight to **bold**

Alignment

Class	Alignment
has-text-centered	Makes the text **centered**
has-text-justified	Makes the text **justified**
has-text-left	Makes the text aligned to the **left**
has-text-right	Makes the text aligned to the **right**

TYPOGRAPHY - HEADINGS

- There are **2 types** of heading:

- There are **6 sizes** available:

```
<h1 class="title is-1">Title 1</h1>
<h2 class="title is-2">Title 2</h2>
<h3 class="title is-3">Title 3</h3>
<h4 class="title is-4">Title 4</h4>
<h5 class="title is-5">Title 5</h5>
<h6 class="title is-6">Title 6</h6>
```

```
<h1 class="subtitle is-1">Subtitle 1</h1>
<h2 class="subtitle is-2">Subtitle 2</h2>
<h3 class="subtitle is-3">Subtitle 3</h3>
<h4 class="subtitle is-4">Subtitle 4</h4>
<h5 class="subtitle is-5">Subtitle 5</h5>
<h6 class="subtitle is-6">Subtitle 6</h6>
```

COLUMNS

- Building a columns layout with Bulma is very simple:
  - Add a columns container
  - Add as many column elements as you want
  - Each column will have an equal width, no matter the number of columns.

Copy

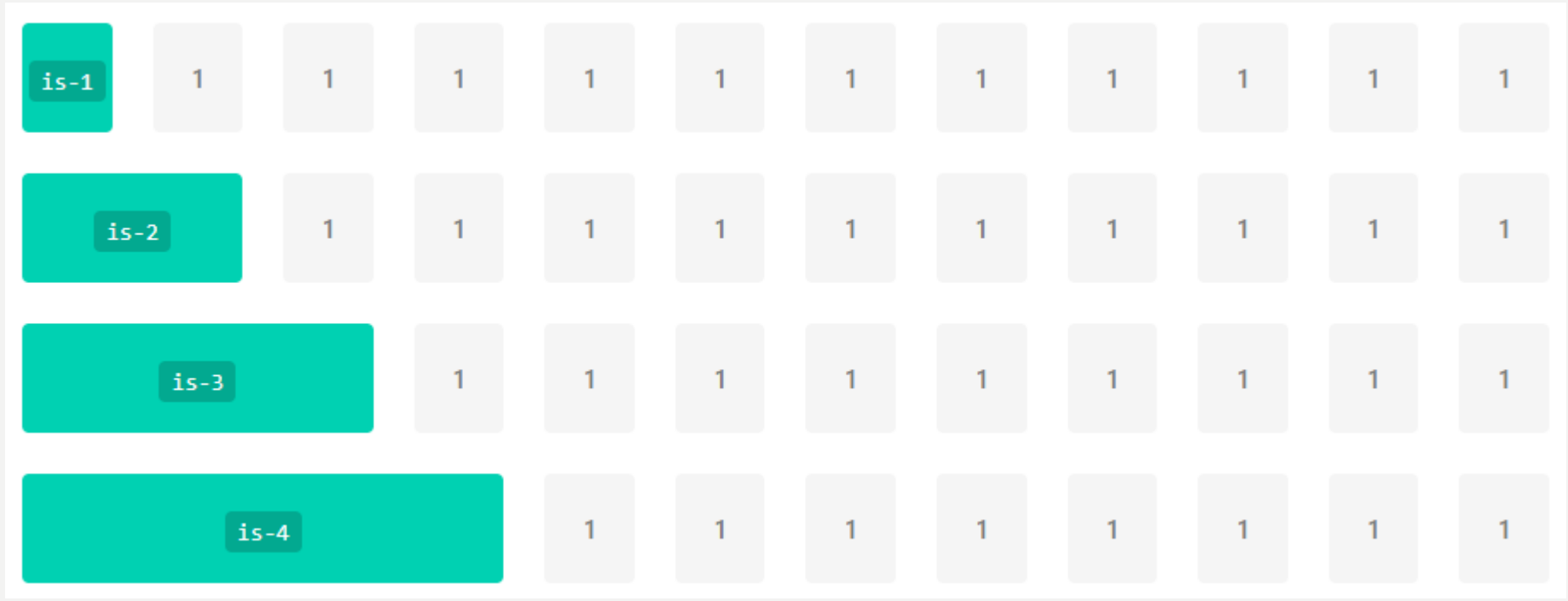
```
<div class="columns">
  <div class="column">
    First column
  </div>
  <div class="column">
    Second column
  </div>
  <div class="column">
    Third column
  </div>
  <div class="column">
    Fourth column
  </div>
</div>
```

COLUMNS - SIZE

- If you want to change the size of a single column, you can use one of the following classes:
  - **is-three-quarters**
  - **is-two-thirds**
  - **is-half**
  - **is-one-third**
  - **is-one-quarter**
  - **is-full**
- The other columns will fill up the **remaining** space automatically.
- You can also use the following multiples of 20% as well:
  - **is-four-fifths**
  - **is-three-fifths**
  - **is-two-fifths**
  - **is-one-fifth**

COLUMNS - 12 COLUMNS SYSTEM

- As the grid can be divided into 12 columns, there are size classes for each division:
  - is-1
  - is-2
  - is-3
  - is-4
  - is-5
  - is-6
  - is-7
  - is-8
  - is-9
  - is-10
  - is-11
  - is-12



LAYOUT - SECTION

- A simple container to divide your page into **sections.**

- Use sections as direct children of **body.**

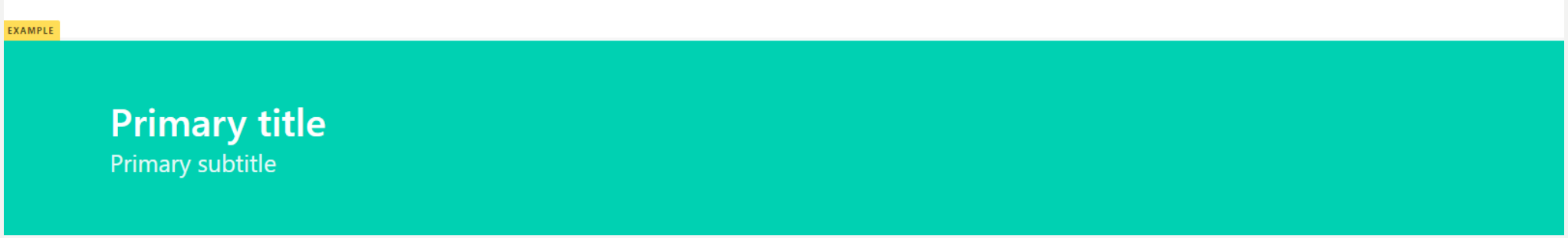
```
<body>
  <section class="section">
    <div class="container">
      <h1 class="title">Section</h1>
      <h2 class="subtitle">
        A simple container to divide your page into <strong>sections</strong>, like the one you're currently reading
      </h2>
    </div>
  </section>
</body>
```

- You can use the modifiers is-medium and is-large to change the **spacing**.

LAYOUT - HERO

- The hero component allows you to add a full width banner to your webpage, which can optionally cover the full height of the page as well.
- The basic requirement of this component are:
  - **hero** as the main container
  - **hero-body** as a direct child, in which you can put all your content

```
<section class="hero is-primary">
  <div class="hero-body">
    <div class="container">
      <h1 class="title">
        Primary title
      </h1>
      <h2 class="subtitle">
        Primary subtitle
      </h2>
    </div>
  </div>
</section>
```

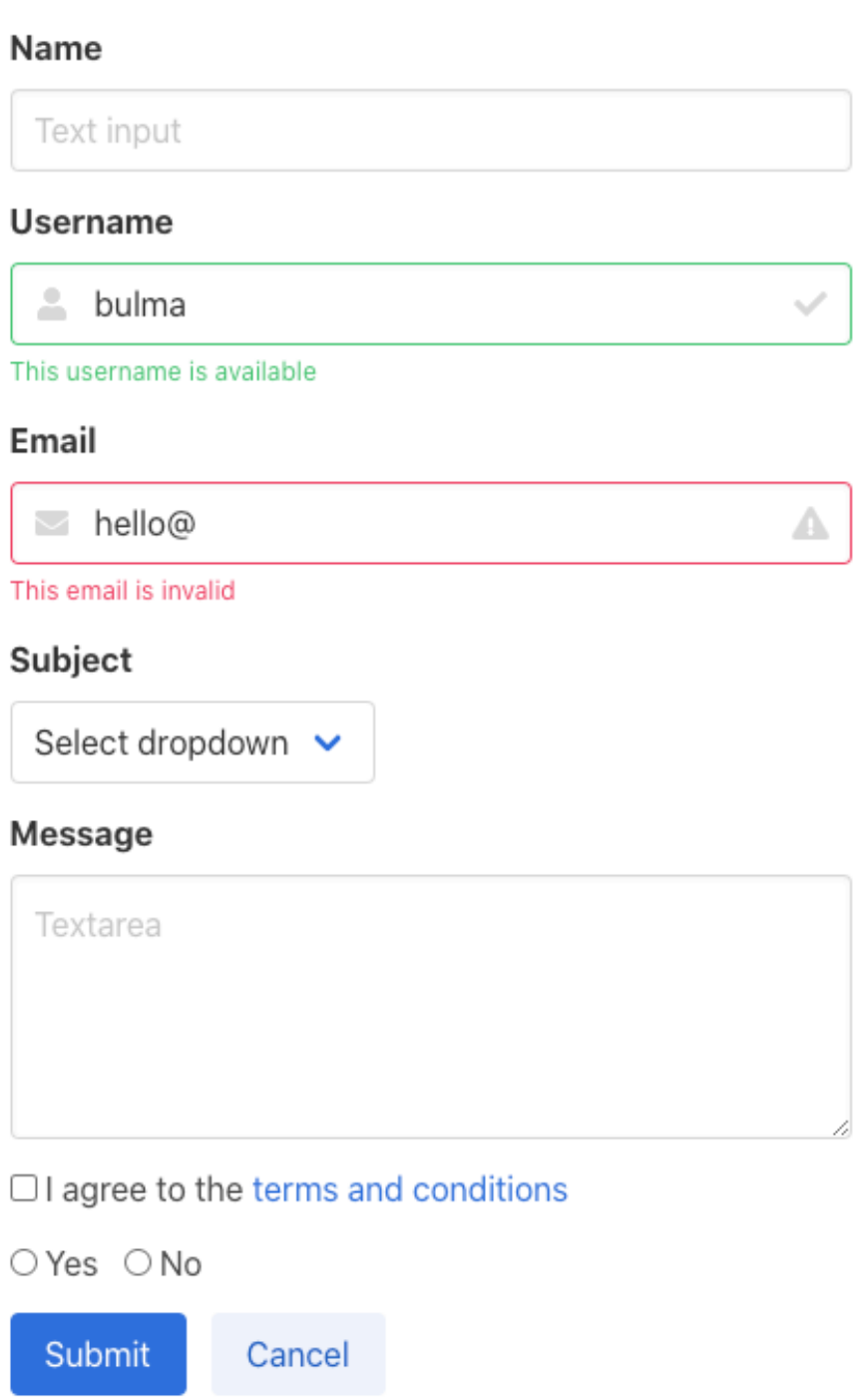


LAYOUT - CONTAINER

- The **container** is a simple utility element that allows you to center content on larger viewports. It can be used in any context, but mostly as a direct child of one of the following:
  - Navbar
  - Hero
  - Section
  - Footer
- By default, the container will only be activated from the **$desktop** breakpoint. It will increase its max-width after reaching the **$widescreen** and **$fullhd** breakpoints.

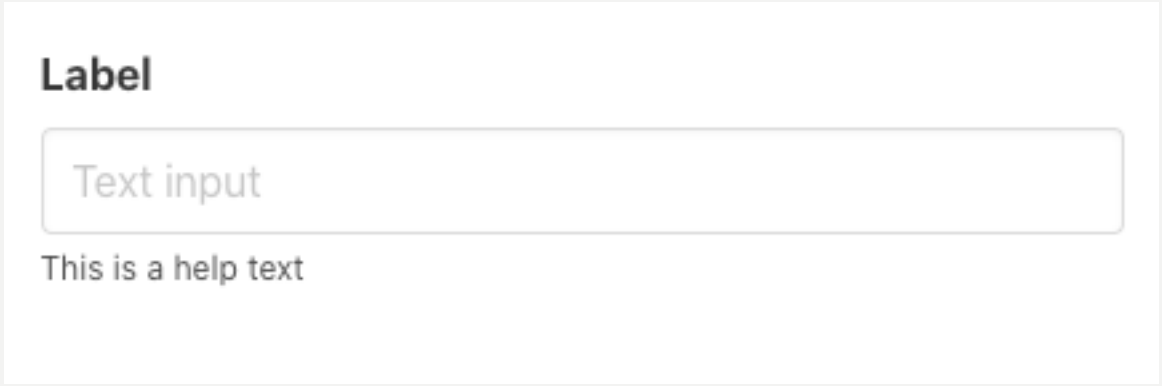
FORM CONTROLS

- Bulma supports the following native HTML form elements: **<form> <button> <input> <textarea>** and **<label>.**
- The following CSS classes are supported:
  - label
  - input
  - textarea
  - select
  - checkbox
  - radio
  - button
  - help



FORM FIELD

- The **field** container is a simple container for:
  - a text **label**
  - a form **control**
  - an optional **help** text




```
<div class="field">
  <label class="label">Label</label>
  <div class="control">
    <input class="input" type="text" placeholder="Text input">
  </div>
  <p class="help">This is a help text</p>
</div>
```

FORM CONTROL

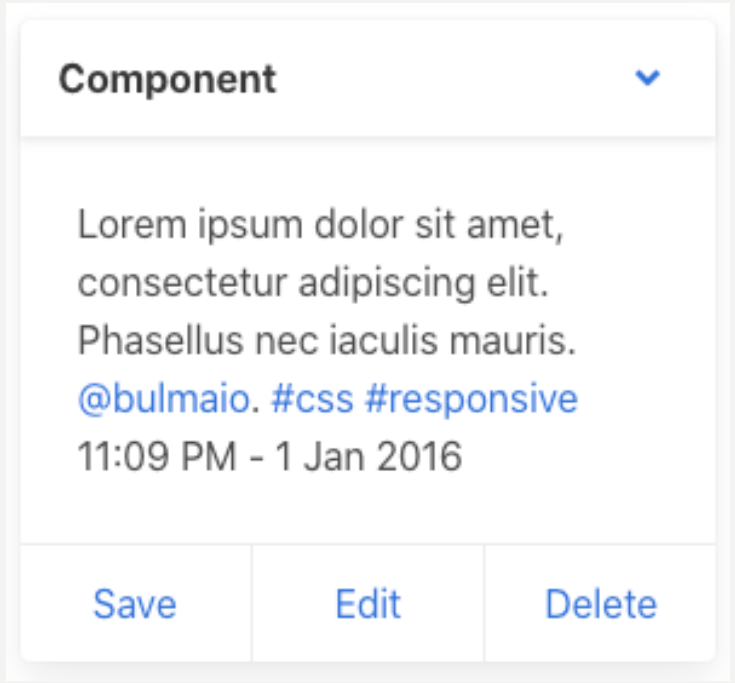
- The Bulma **control** is a versatile block container meant to enhance single form controls.
- This container gives the ability to:
  - keep the **spacing** consistent
  - combine form controls into a **group**
  - combine form controls into a **list**
  - append and prepend **icons** to a form control

```
<div class="control">
  <div class="select">
    <select>
      <option>Select dropdown</option>
      <option>With options</option>
    </select>
  </div>
</div>
```

COMPONENTS - CARD

- The card component comprises several elements that you can mix and match:
  - **card**: the main container
    - **card-header**: a horizontal bar with a shadow
      - **card-header-title**: a left-aligned bold text
      - **card-header-icon**: a placeholder for an icon
    - **card-image**: a fullwidth container for a responsive image
    - **card-content**: a multi-purpose container for any other element
    - **card-footer**: a horizontal list of controls
      - **card-footer-item**: a repeatable list item

- You can center the **card-header-title** by appending the **is-centered** modifier.



```
<div class="card">
  <header class="card-header">
    <p class="card-header-title">
      Component
    </p>
    <a href="#" class="card-header-icon" aria-label="more
options">
      <span class="icon">
        <i class="fas fa-angle-down" aria-hidden="true"></i>
      </span>
    </a>
  </header>
  <div class="card-content">
    <div class="content">
      Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Phasellus nec iaculis mauris.
      <a href="#">@bulmaio</a>. <a href="#">#css</a> <a
href="#">#responsive</a>
      <br>
      <time datetime="2016-1-1">11:09 PM - 1 Jan 2016</time>
    </div>
  </div>
  <footer class="card-footer">
    <a href="#" class="card-footer-item">Save</a>
    <a href="#" class="card-footer-item">Edit</a>
    <a href="#" class="card-footer-item">Delete</a>
  </footer>
</div>
```

COMPONENTS - MODAL

- The modal structure is very simple:
  - **modal**: the main container
    - **modal-background**: a transparent overlay that can act as a click target to close the modal
    - **modal-content**: a horizontally and vertically centered container, with a maximum width of 640px, in which you can include *any* content
    - **modal-close**: a simple cross located in the top right corner

- To **activate** the modal, just add the **is-active** modifier on the **.modal** container. You may also want to add **is-clipped** modifier to a containing element (usually html) to stop scroll overflow.

WEB PROGRAMMING 0601632 2

BY DR BUNDIT THANASOPON

HMTL REVISION

Src: https://www.W3schools.Com/html/html_intro.Asp

WHAT IS HTML?

- HTML is the standard markup language for creating Web pages.
  - HTML stands for "Hyper Text Markup Language"
  - HTML describes the structure of Web pages using markup
  - HTML elements are the building blocks of HTML pages
  - HTML elements are represented by tags
  - HTML tags label pieces of content such as "heading", "paragraph", "table", and so on
  - Browsers do not display the HTML tags, but use them to render the content of the page

```
<html>
  <head>
    <title>Page title</title>
  </head>
  <body>
    <h1>This is a heading</h1>
    <p>This is a paragraph.</p>
    <p>This is another paragraph.</p>
  </body>
</html>
```

EXAMPLE EXPLAINED

- The <!DOCTYPE html> declaration defines this document to be HTML5
- The <html> element is the root element of an HTML page
- The <head> element contains meta information about the document
- The <title> element specifies a title for the document
- The <body> element contains the visible page content
- The <h1> element defines a large heading
- The <p> element defines a paragraph

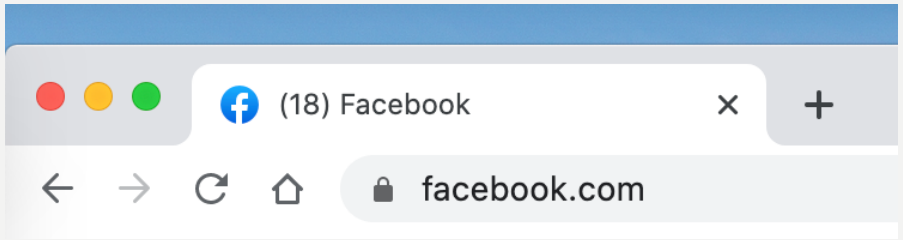
HTML HEAD

- The <head> element is a container for metadata (data about data) and is placed between the <html> tag and the <body> tag.
- HTML metadata is data about the HTML document. Metadata is not displayed.
- The following tags describe metadata: <title>, <style>, <meta>, <link>, <script>, and <base>.

METADATA TAGS

- The <title> element:
  - defines a title in the browser tab
  - provides a title for the page when it is added to favorites
  - displays a title for the page in search engine results

```
<head>
  <title>Page Title</title>
</head>
```



- The <style> element is used to define style information for a single HTML page:

```
<style>
  body {background-color: powderblue;}
  h1 {color: red;}
  p {color: blue;}
</style>
```

- The <link> element is used to link to external style sheets:

```
<link rel="stylesheet" href="mystyle.css">
```

METADATA TAGS

- The <meta> element is used by browsers (how to display content), by search engines (keywords), and other web services.

```
<meta charset="UTF-8">
<meta name="description" content="Free Web tutorials">
<meta name="keywords" content="HTML,CSS,XML,JavaScript">
<meta name="author" content="John Doe">
```

HTML TAGS

```
<tagname>Content goes here...</tagname>
```

- HTML headings are defined with the <h1> to <h6> tags.
- <h1> defines the most important heading. <h6> defines the least important heading

```
<h1>This is heading 1</h1>
<h2>This is heading 2</h2>
```

- HTML paragraphs are defined with the <p> tag

```
<p>This is a paragraph.</p>
```

HTML TAGS

- HTML links are defined with the <a> tag.

```
<a href="https://www.w3schools.com">This is a link</a>
```

- HTML images are defined with the <img> tag

```
<img src="w3schools.jpg" alt="W3Schools.com" width="104" height="142">
```

- HTML lists are defined with the <ul> (unordered/bullet list) or the <ol> (ordered/numbered list) tag, followed by <li> tags (list items):

```
<ul>
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ul>
```

HTML ATTRIBUTES

- All HTML elements can have **attributes**
- Attributes provide **additional information** about an element
- Attributes are always specified in **the start tag**
- Attributes usually come in name/value pairs like: **name="value"**

Attribute	Description
alt	Specifies an alternative text for an image, when the image cannot be displayed
disabled	Specifies that an input element should be disabled
href	Specifies the URL (web address) for a link
id	Specifies a unique id for an element
src	Specifies the URL (web address) for an image
style	Specifies an inline CSS style for an element
title	Specifies extra information about an element (displayed as a tool tip)

HTML STYLES

- Setting the style of an HTML element, can be done with the **style** attribute.

```
<tagname style="property:value;">
```

The ***property*** is a CSS property.

The ***value*** is a CSS value.

- Examples:
  - `<h1 style="font-family:verdana;">This is a heading</h1>`
  - `<p style="text-align:center;">Centered paragraph.</p>`
  - `<body style="background-color:powderblue;">`

HTML COMMENTS

```
<!-- Write your comments here -->
```

IMPORTANT HTML TAGS

HTML LINKS - HYPERLINKS

```
<a href="url">Link text</a>
```

```
<a href="https://www.w3schools.com/html/">Visit our HTML tutorial</a>
```

VS

```
<a href="html_images.asp">HTML Images</a>
```

```
<h2 id="C4">Chapter 4</h2>
<a href="#C4">Jump to Chapter 4</a>
```

HTML IMAGES

- In HTML, images are defined with the <img> tag.
- The <img> tag is empty, it contains attributes only, and does not have a closing tag.
- The **src** attribute specifies the URL (web address) of the image.
- The **alt** attribute provides an alternate text for an image, if the user for some reason cannot view it
  - `<img src="img_chania.jpg" alt="Flowers in Chania">`
  - `<img src="img_girl.jpg" alt="Girl in a jacket" style="width:500px;height:600px;">`

HTML TABLE

- An HTML table is defined with the <table> tag.
- Each table row is defined with the <tr> tag. A table header is defined with the <th> tag.
- By default, table headings are bold and centered. A table data/cell is defined with the <td> tag.

```
<table style="width:100%">
  <tr>
    <th>Firstname</th>
    <th>Lastname</th>
    <th>Age</th>
  </tr>
  <tr>
    <td>Jill</td>
    <td>Smith</td>
    <td>50</td>
  </tr>
  <tr>
    <td>Eve</td>
    <td>Jackson</td>
    <td>94</td>
  </tr>
</table>
```

HTML TABLE - CELLS THAT SPAN MANY COLUMNS

```
<table style="width:100%">
  <tr>
    <th>Name</th>
    <th colspan="2">Telephone</th>
  </tr>
  <tr>
    <td>Bill Gates</td>
    <td>55577854</td>
    <td>55577855</td>
  </tr>
</table>
```

- To make a cell span more than one column, use the colspan attribute:

Name	Telephone	
Bill Gates	55577854	55444778

HTML TABLE - CELLS THAT SPAN MANY ROWS

- To make a cell span more than one row, use the rowspan attribute:

Name:	Bill Gates
Telephone:	55577854
	55444778

```
<table style="width:100%">
  <tr>
    <th>Name:</th>
    <td>Bill Gates</td>
  </tr>
  <tr>
    <th rowspan="2">Telephone:</th>
    <td>55577854</td>
  </tr>
  <tr>
    <td>55577855</td>
  </tr>
</table>
```

EXERCISES

HTML LISTS

- Unordered HTML List

```
<ul>
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ul>
```

- Coffee
- Milk
- Tea

- Ordered HTML List

```
<ol>
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ol>
```

1. Coffee
2. Milk
3. Tea

EXERCISES

- Item 1
- Item 2
  - Item 2.1
  - Item 2.2
  - Item 2.3
    - Item 2.3.1
    - Item 2.3.2
- Item 3

1. Item 1
2. Item 2
3. Item 3
   - Item 3.1
   - Item 3.2
   - Item 3.3
4. Item 4

HTML BLOCK AND INLINE ELEMENTS

- Block-level Elements
  - A block-level element always starts on a new line and takes up the full width available (stretches out to the left and right as far as it can).

```
<div>Hello</div>
<div>World</div>
```

<address>
<article>
<aside>
<blockquote>
<canvas>
<dd>

<dl>
<dt>
<fieldset>
<figcaption>
<figure>
<footer>
<form>
<h1>-<h6>
<header>
<hr>

<main>
<nav>
<noscript>

<output>
<p>
<pre>
<section>
<table>
<tfoot>

<video>

HTML BLOCK AND INLINE ELEMENTS

- An inline element does not start on a new line and only takes up as much width as necessary.
- This is an inline element inside a paragraph:

```
<span>Hello</span>
<span>World</span>
```

<a>		<select>
<abbr>	<i>	<small>
<acronym>	<img>	
<b>	<input>	
<bdo>	<kbd>	<sub>
<big>	<label>	<sup>
 	<map>	<textarea>
<button>	<object>	<time>
<cite>	<q>	<tt>
<code>	<samp>	<var>
<dfn>	<script>	

THE <DIV> ELEMENT

- The element is often used as a container for other HTML elements.
- The element has no required attributes, but style, class and id are common.
- When used together with CSS, the element can be used to style blocks of content:

```
<div style="background-color:black;color:white;padding:20px;">
  <h2>London</h2>
  <p>London is the capital city of England. It is the most populous city in the United Kingdom, with a metropolitan area of over 13 million inhabitants.</p>
</div>
```

THE <SPAN> ELEMENT

- The element is often used as a container for some text.
- The element has no required attributes, but style, class and id are common.
- When used together with CSS, the element can be used to style parts of the text:

```
<h1>My <span style="color:red">Important</span> Heading</h1>
```

HTML FORM

HTML FORM

- The HTML <form> element defines a form that is used to collect user input:

```
<form>
.
form elements
.
</form>
```

- An HTML form contains **form elements**.
- Form elements are different types of input elements, like text fields, checkboxes, radio buttons, submit buttons, and more

KEY ATTRIBUTES

- The **action** attribute defines the action to be performed when the form is submitted.
  - Normally, the form data is sent to a web page on the server when the user clicks on the submit button.

- The **target** attribute specifies if the submitted result will open in a new browser tab, a frame, or in the current window.
  - The default value is "_self" which means the form will be submitted in the current window.

- The **method** attribute specifies the HTTP method (**GET** or **POST**) to be used when submitting the form data.

```
<form action="/action_page.php" method=”POST">
```

WHEN TO USE GET?

- The default method when submitting form data is GET.
- However, when GET is used, the submitted form data will be **visible in the page address field**: `/action_page.php?firstname=Mickey&lastname=Mouse`

- NOTE:
  - Appends form-data into the URL in name/value pairs
  - The length of a URL is limited (about 3000 characters)
  - Never use GET to send sensitive data! (will be visible in the URL)
  - Useful for form submissions where a user wants to bookmark the result
  - GET is better for non-secure data, like query strings in Google

WHEN TO USE POST?

- Always use POST if the form data contains sensitive or personal information.
- The POST method does not display the submitted form data in the page address field.

- NOTE:
  - POST has no size limitations and can be used to send large amounts of data.
  - Form submissions with POST cannot be bookmarked

THE <INPUT> ELEMENT

Type	Description
<input type="text">	Defines a one-line text input field
<input type="radio">	Defines a radio button (for selecting one of many choices)
<input type="submit">	Defines a submit button (for submitting the form)

```
<form action="/action_page.php">
  First name:<br>
  <input type="text" value="Mickey"><br>
  Last name:<br>
  <input type="text" name="lastname" value="Mouse"><br><br>
  <input type="submit" value="Submit">
</form>
```

THE NAME ATTRIBUTE

- Each input field must have a name attribute to be submitted.
- **If the name attribute is omitted, the data of that input field will not be sent at all.**
- This example will only submit the "Last name" input field:

```
<form action="/action_page.php">
  First name:<br>
  <input type="text" value="Mickey"><br>
  Last name:<br>
  <input type="text" name="lastname" value="Mouse"><br><br>
  <input type="submit" value="Submit">
</form>
```

HTML INPUT TYPES

- The <input> element can be displayed in several ways, depending on the type attribute.
  - Text
  - Password
  - Submit
  - Reset
  - Radio
  - Checkbox
  - Button
- HTML5 added several new input types:
  - Color
  - Date
  - Datetime-local
  - Email
  - Month
  - Number
  - Range
  - Search
  - Tel
  - Time
  - Url
  - Week

HTML INPUT ATTRIBUTES

- The <input> element can have attributes that specify its behavior
  - Value
  - Readonly
  - Disabled
  - Size
  - Maxlength
- HTML5 added the following attributes for <input>:
- Autocomplete
- Autofocus
- Height and width
- List
- Min and max
- Multiple
- Pattern (regexp)
- Placeholder
- Required
- Step

OTHER FORM ELEMENTS

- The <select> element defines a **drop-down list**:

```
<select name="cars">
  <option value="volvo">Volvo</option>
  <option value="saab">Saab</option>
  <option value="fiat">Fiat</option>
  <option value="audi">Audi</option>
</select>
```

- The <textarea> element defines a multi-line input field (**a text area**):

```
<textarea name="message" rows="10" cols
="30">
The cat was playing in the garden.
</textarea>
```

OTHER FORM ELEMENTS

- The <button> element defines a clickable **button**:

- ```
  <button type="button" onclick="
  alert('Hello World!')">Click
  Me!</button>
  ```

- HTML5 added the following form elements:
  - <datalist>
  - <output>
  ```

TRY TO SEND POST AND GET REQUEST TO THIS URL

https://www.w3schools.com/action_page.php

WEB PROGRAMMING 0601632 2

BY DR BUNDIT THANASOPON

JAVASCRIPT REVISION

SRC: HTTPS://WWW.W3SCHOOLS.COM/JS/DEFAULT.ASP

EXAMPLES OF WHAT JAVASCRIPT CAN DO

Change	JavaScript Can Change HTML Content • document.getElementById("demo").innerHTML = "Hello JavaScript";
Change	JavaScript Can Change HTML Attribute Values • document.getElementById(”myImage").src = "hello.jpg";
Change	JavaScript Can Change HTML Styles (CSS) • document.getElementById("demo").style.fontSize = "35px";
Create	JavaScript Can Create HTML Elements
Remove	Javascript Can Remove HTML Elements

THE <SCRIPT> TAG

- Scripts can be placed in the **<body>,** or in the **<head>** section of an HTML page, or in both.

```
<script>
function myFunction() {
 document.getElementById("demo").innerHTML = "Paragraph changed.";
}
</script>
```

- External JavaScript

```
<script src="myScript.js"></script>
```

EXTERNAL JAVASCRIPT ADVANTAGES

- Placing scripts in external files has some advantages:
  - It separates HTML and code
  - It makes HTML and JavaScript easier to read and maintain
  - Cached JavaScript files can speed up page loads

- To add several script files to one page - use several script tags:

```
<script src="myScript1.js"></script>
<script src="myScript2.js"></script>
```

JAVASCRIPT OUTPUT

- JavaScript can "display" data in different ways:
  - Writing into an HTML element, using innerHTML.
  - Writing into the HTML output using document.write().
  - Writing into an alert box, using window.alert().
  - Writing into the browser console, using console.log().

```
<h1>My First Web Page</h1>
<p>My first paragraph.</p>

<script>
document.write(5 + 6);
</script>
```

JAVASCRIPT KEYWORDS

Keyword	Description
break	Terminates a switch or a loop
continue	Jumps out of a loop and starts at the top
debugger	Stops the execution of JavaScript, and calls (if available) the debugging function
do ... while	Executes a block of statements, and repeats the block, while a condition is true
for	Marks a block of statements to be executed, as long as a condition is true
function	Declares a function
if ... else	Marks a block of statements to be executed, depending on a condition
return	Exits a function
switch	Marks a block of statements to be executed, depending on different cases
try ... catch	Implements error handling to a block of statements
var	Declares a variable

JAVASCRIPT SYNTAX

- JavaScript Identifiers
  - Identifiers are names.
  - In JavaScript, identifiers are used to name variables (and keywords, and functions, and labels).
  - In JavaScript, the first character must be a letter, or an underscore (_), or a dollar sign ($).

- JavaScript is Case Sensitive
  - All JavaScript identifiers are **case sensitive**.
  - The variables lastName and lastname, are two different variables
- JavaScript Comments
  - Code after double slashes // or between /* and */ is treated as a **comment**.

JAVASCRIPT OPERATORS

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
**	Exponentiation (ES6)
/	Division
%	Modulus (Division Remainder)
++	Increment
--	Decrement

Operator	Description
==	Equal to
===	Equal value and equal type
!=	Not equal
!==	Not equal value or not equal type
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
?	Ternary operator
&&	Logical and
||	Logical or
!	Logical not

JAVASCRIPT DATA TYPES – PRIMITIVE DATA

- JavaScript Types are Dynamic - this means that the same variable can be used to hold different data types:

```
let x;       // Now x is undefined
x = 5;       // Now x is a Number
x = "John"; // Now x is a String
```

- A **primitive data** value is a single simple data value with no additional properties and methods.
- The typeof operator can return one of these primitive types:
  - string
  - number
  - boolean (true or false)
  - undefined

JAVASCRIPT DATA TYPES COMPLEX DATA

- **Complex Data**
- The typeof operator can return one of two complex types:
  - function
  - object
- The typeof operator returns object for both objects, arrays, and null.
- The typeof operator return " function" for functions.

```
typeof {name:'John', age:34}
// Returns "object"
typeof [1,2,3,4]
// Returns "object"
typeof null
// Returns "object"
typeof function myFunc(){}
// Returns "function"
```

The **typeof** operator returns **"object"** for arrays because in JavaScript arrays are objects.

JAVASCRIPT FUNCTIONS

- A JavaScript function is a block of code designed to perform a particular task.
- A JavaScript function is executed when "something" invokes it (calls it).

```
function myFunction(p1, p2) {
  return p1 * p2;   // The function returns the product of p1 and p2
}
```

- The () Operator Invokes the Function
  - toCelsius refers to the function object, and toCelsius() refers to the function result.
  - Accessing a function without () will return the function definition instead of the function result:

```
function toCelsius(fahrenheit) {
  return (5/9) * (fahrenheit-32);
}
document.getElementById("demo").innerHTML = toCelsius;
```

JAVASCRIPT OBJECTS

- What is an object? In real life, a car is an **object**. A car has **properties** like weight and color, and **methods** like start and stop:

Object	Properties	Methods
	car.name = Fiat	car.start()
	car.model = 500	car.drive()
	car.weight = 850kg	car.brake()
	car.color = white	car.stop()

- All cars have the same **properties**, but the property **values** differ from car to car.
- All cars have the same **methods**, but the methods are performed **at different times**.

JAVASCRIPT OBJECTS

```
var person = {
  firstName: "John",
  lastName : "Doe",
  id       : 5566,
  fullName : function() {
    return this.firstName + " " + this.lastName;
  }
};
```

- In a function definition, this refers to the "owner" of the function.
- In the example above, this is the **person object** that "owns" the fullName function.
- In other words, this.firstName means the firstName property of **this object**.

JAVASCRIPT EVENTS

- **HTML events** are **"things"** that happen to HTML elements.
- When JavaScript is used in HTML pages, JavaScript can **"react"** on these events.

- An HTML event can be something the browser does, or something a user does.
- Here are some examples of HTML events:
  - An HTML web page has finished loading
  - An HTML input field was changed
  - An HTML button was clicked
- Often, when events happen, you may want to do something. JavaScript lets you execute code when events are detected.

COMMON HTML EVENTS

Event	Description
onchange	An HTML element has been changed
onclick	The user clicks an HTML element
onmouseover	The user moves the mouse over an HTML element
onmouseout	The user moves the mouse away from an HTML element
onkeydown	The user pushes a keyboard key
onload	The browser has finished loading the page

ADDING EVENT LISTENERS

```
<button id="btn1">Click me</button>
<button id="btn2">Act-once button</button>
<script>
    let button1 = document.getElementById(“btn1");
    button1.addEventListener("click", () => {
            console.log("Button clicked.");
    });

    let button2 = document.getElementById(“btn2");
    function once() {
            console.log("Done.");
            button2.removeEventListener("click", once);
    }
    button2.addEventListener("click", once);
</script>
```

JAVASCRIPT ARRAY

JAVASCRIPT ARRAYS

- An array is a special variable, which can hold more than one value at a time.

```
var cars = ["Saab", "Volvo", "BMW"];
```

- You access an array element by referring to the **index number**.

```
var name = cars[0];
```

- This statement changes the value of the first element in cars:

```
cars[0] = "Opel";
```

JAVASCRIPT ARRAY ITERATION METHODS - FOREACH

- The forEach() method calls a function (a callback function) once for each array element.

```
var txt = "";
var numbers = [45, 4, 9, 16, 25];
numbers.forEach(myFunction);

function myFunction(value, index, array) {
  txt = txt + value + "<br>";
}
```

- Note that the function takes 3 arguments:
  - The item value
  - The item index
  - The array itself

JAVASCRIPT ARRAY ITERATION METHODS - MAP

- The map() method creates a new array by performing a function on each array element.
  - The map() method **does not** execute the function for array elements without values.
  - The map() method **does not** change the original array.
- This example multiplies each array value by 2:

```
var numbers1 = [45, 4, 9, 16, 25];
var numbers2 = numbers1.map(myFunction);

function myFunction(value, index, array) {
  return value * 2;
}
```

JAVASCRIPT ARRAY ITERATION METHODS - FILTER

- The filter() method creates a new array with array elements that passes a test.
- This example creates a new array from elements with a value larger than 18:

```
var numbers = [45, 4, 9, 16, 25];
var over18 = numbers.filter(myFunction);

function myFunction(value, index, array) {
  return value > 18;
}
```

JAVASCRIPT ARRAY ITERATION METHODS - REDUCE

- The reduce() method runs a function on each array element to produce (reduce it to) a single value.
- The reduce() method works from left-to-right and does not reduce the original array.
- This example finds the sum of all numbers in an array:

```
var numbers1 = [45, 4, 9, 16, 25];
var sum = numbers1.reduce(myFunction);

function myFunction(total, value, index, array) {
  return total + value;
}
```

Note that the function takes 4 arguments:

- The total (the initial value / previously returned value)
- The item value
- The item index
- The array itself

```
var sum = numbers1.reduce(myFunction);
```

Loop 2
total = 49
value = 9

Loop 4
total = 74
value = 25

[45 , 4 , 9 , 16 , 25]

99

Loop 1
total = 45
value = 4

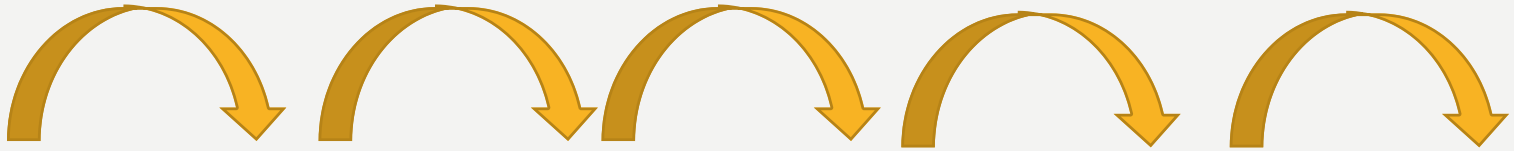
Loop 3
total = 58
value = 16

```
var sum = numbers1.reduce(myFunction, 5);
```

Loop 1
total = 5
value = 45

Loop 3
total = 54
value = 9

Loop 5
total = 79
value = 25



[45 , 4 , 9 , 16 , 25]

104

Loop 2
total = 50
value = 4

Loop 4
total = 63
value = 16

JAVASCRIPT ARRAY ITERATION METHODS - INDEXOF

- The indexOf() method searches an array for an element value and returns its position.
- **Note:** The first item has position 0, the second item has position 1, and so on.

```
var fruits = ["Apple", "Orange", "Apple", "Mango"];
var a = fruits.indexOf("Apple");
```

- Array.lastIndexOf() is the same as Array.indexOf(), but searches from the end of the array.

SPLICE

- The splice() method adds/removes items to/from an array, and returns the removed item(s).

```
array.splice(index, howmany, item1, ....., itemX)
```

Parameter	Description
index	Required. An integer that specifies at what position to add/remove items, Use negative values to specify the position from the end of the array
howmany	Optional. The number of items to be removed. If set to 0, no items will be removed
item 1, ..., itemX	Optional. The new item(s) to be added to the array

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.splice(2, 1, "Lemon", "Kiwi");
```

JAVASCRIPT COOKIES

- Cookies are data, stored in small text files, on your computer.
  - When a web server has sent a web page to a browser, the connection is shut down, and the server forgets everything about the user - **stateless**
- Cookies were invented to solve the problem "how to remember information about the user":
  - When a user visits a web page, his name can be stored in a cookie.
  - Next time the user visits the page, the cookie "remembers" his name.

- When a browser requests a web page from a server, cookies belonging to the page is added to the request. This way the server gets the necessary data to "remember" information about users.

CREATE A COOKIE WITH JAVASCRIPT

- JavaScript can create, read, and delete cookies with the ”document.cookie” property.
  - Create

```
document.cookie = "username=John Doe; expires=Thu, 18 Dec 2013 12:00:00 UTC";
```

  - Read

```
var x = document.cookie;
```

  - Delete

```
document.cookie = "username=; expires=Thu, 01 Jan 1970 00:00:00 UTC; path=/;";
```

TRY this js library: https://github.com/js-cookie/js-cookie

WINDOW LOCALSTORAGE PROPERTY

- The localStorage and sessionStorage properties allow to save key/value pairs in a web browser.
  - The localStorage object stores data with no expiration date. The data will not be deleted when the browser is closed, and will be available the next day, week, or year.
  - The localStorage property is read-only.

```
localStorage.setItem("key", "value");
```

```
var lastname = localStorage.getItem("key");
```

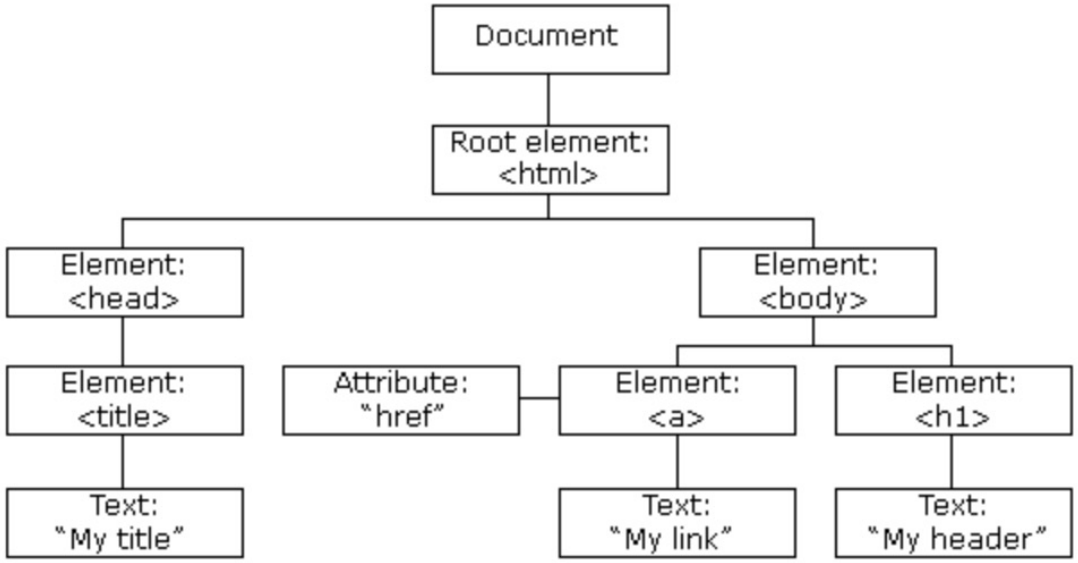
```
localStorage.removeItem("key");
```

THE HTML DOM

(DOCUMENT OBJECT MODEL)

THE HTML DOM

- With the HTML DOM, JavaScript can access and change all the elements of an HTML document.
- When a web page is loaded, the browser creates a **D**ocument **O**bject **M**odel of the page.
- The **HTML DOM** model is constructed as a tree of **Objects**:



WHAT CAN JAVASCRIPT DO WITH THE HTML DOM?

- 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

THE DOM PROGRAMMING INTERFACE

- The HTML DOM can be accessed with JavaScript (and with other programming languages).
- In the DOM, all HTML elements are defined as **objects**.
  - A **property** is a value that you can get or set (like changing the content of an HTML element).
  - A **method** is an action you can do (like add or deleting an HTML element).

```
document.getElementById("demo").innerHTML = "Hello World!";
```

- In the example above, getElementById is a **method**, while innerHTML is a **property**.

THE HTML DOM DOCUMENT OBJECT

- The document object represents your web page.
- If you want to access any element in an HTML page, you always start with accessing the document object.
- **Finding HTML Elements**

Method	Description
document.getElementById(*id*)	Find an element by element id
document.getElementsByTagName(*name*)	Find elements by tag name
document.getElementsByClassName(*name*)	Find elements by class name
document.querySelector(*selectors*)	Find an element by css selector
document.querySelectorAll(*selectors*)	Find elements by css selector

CHANGING HTML ELEMENTS

Method	Description
element.innerHTML = *new html content*	Change the inner HTML of an element
element.attribute = *new value*	Change the attribute value of an HTML element
element.setAttribute*(attribute, value)*	Change the attribute value of an HTML element
element.style.*property* = *new style*	Change the style of an HTML element

ADDING AND DELETING ELEMENTS

Method	Description
document.createElement(*element*)	Create an HTML element
document.removeChild(*element*)	Remove an HTML element
document.appendChild(*element*)	Add an HTML element
document.replaceChild(*element*)	Replace an HTML element
document.write(*text*)	Write into the HTML output stream

JAVASCRIPT HTML DOM EVENTS

- A JavaScript can be executed when an event occurs, like when a user clicks on an HTML element.
- Examples of HTML events:
  - When a user clicks the mouse
  - When a web page has loaded
  - When an image has been loaded
  - When the mouse moves over an element
  - When an input field is changed
  - When an HTML form is submitted
  - When a user strokes a key

```
<h1 onclick="changeText(this)">
Click on this text!</h1>
```

JAVASCRIPT HTML DOM EVENTLISTENER

```
document.getElementById("myBtn").addEventListener("click", displayDate);
```

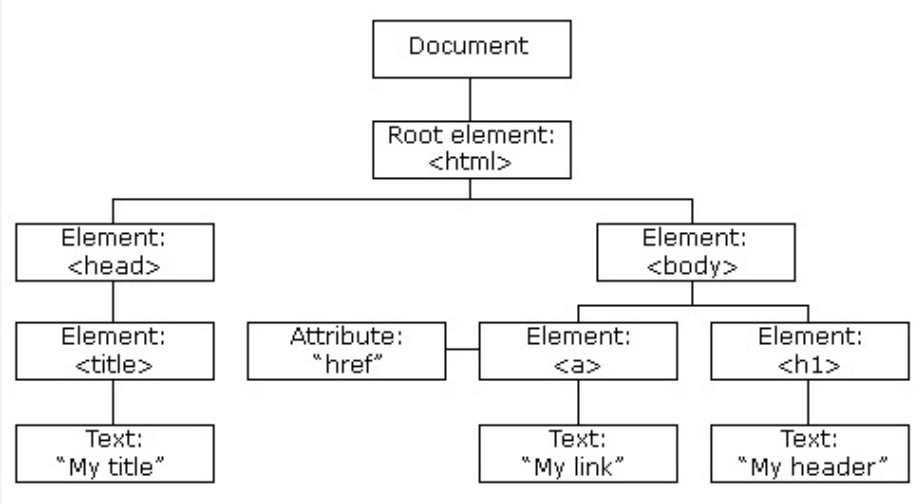
- The addEventListener() method attaches an event handler to the specified element.
- The addEventListener() method allows you to add event listeners on any HTML DOM object such as HTML elements, the HTML document, the window object.

- The removeEventListener() method removes event handlers that have been attached with the addEventListener() method:

```
element.removeEventListener("mousemove", myFunction);
```

JAVASCRIPT HTML DOM NAVIGATION

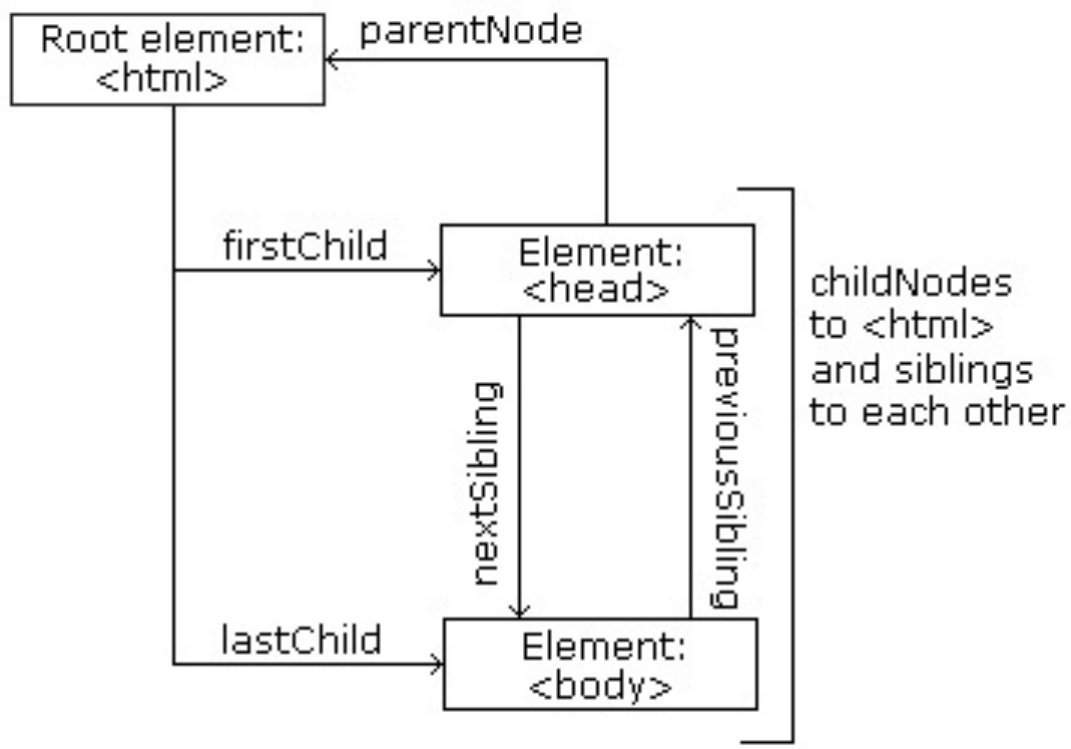
- According to the W3C HTML DOM standard, everything in an HTML document is a node:
  - The entire document is a document node
  - Every HTML element is an element node
  - The text inside HTML elements are text nodes
  - All comments are comment nodes



- With the HTML DOM, all nodes in the node tree can be accessed by JavaScript.
- New nodes can be created, and all nodes can be modified or deleted.

NODE RELATIONSHIPS

- The nodes in the node tree have a hierarchical relationship to each other.
  - The terms parent, child, and sibling are used to describe the relationships.
  - In a node tree, the top node is called the root (or root node)
  - Every node has exactly one parent, except the root (which has no parent)
  - A node can have a number of children
  - Siblings (brothers or sisters) are nodes with the same parent



NODE RELATIONSHIPS

```
<html>

  <head>
    <title>DOM Tutorial</title>
  </head>

  <body>
    <h1>DOM Lesson one</h1>
    <p>Hello world!</p>
  </body>

</html>
```

<html> is the root node

<html> has no parents

<html> is the parent of <head> and <body>

<head> is the first child of <html>

<body> is the last child of <html>

<head> has one child: <title>

<title> has one child (a text node): "DOM Tutorial"

<body> has two children: <h1> and <p>

<h1> has one child: "DOM Lesson one"

<p> has one child: "Hello world!"

<h1> and <p> are siblings

NAVIGATING BETWEEN NODES

- You can use the following node properties to navigate between nodes with JavaScript:
  - parentNode
  - childNodes[*nodenumber*]
  - firstChild
  - lastChild
  - nextSibling
  - previousSibling

- The nodeName property specifies the name of a node.
- The nodeValue property specifies the value of a node.
- The nodeType property is read only. It returns the type of a node.

ASYNCHRONOUS JAVASCRIPT

WEB PROGRAMMING

TODAY'S TOPICS

- The "function" expression
- Higher order functions
- The "arrow" function
- Event loop
  - The Callstack
  - WebAPIs
- Callback hell!
- Promises
- The "async" keyword
- The "await" keyword

THE “FUNCTION” EXPRESSION

FUNCTION EXPRESSION

- A function expression is very similar to and has almost the same syntax as a function declaration.

- The main difference between a function expression and a function declaration:
  - Function expressions are used to create *anonymous* functions (functions without function name).
  - Function expressions in JavaScript are not **hoisted**, unlike function declarations. You can't use function expressions before you create them.

LET'S SEE SOME CODE!

THE FUNCTION EXPRESSION

HIGHER ORDER FUNCTIONS

HIGHER ORDER FUNCTIONS

- Higher order functions are functions that accept other functions as arguments
  - Any function that is passed as an argument is called a "**callback**" function.
  - Functions that return a function

LET'S SEE SOME CODE!

HIGHER ORDER FUNCTIONS

THE ARROW FUNCTIONS

SYNTACTICALLY COMPACT ALTERNATIVE TO A REGULAR "FUNCTION" EXPRESSION

ARROW FUNCTIONS, THE BASICS

- There's another very simple and concise syntax for creating functions, that's often better than Function Expressions.

- **Arrow functions** are handy for one-liners. They come in two flavors:
  - Without curly braces: `(...args) => expression` – the right side is an expression: the function evaluates it and returns the result.
  - With curly braces: `(...args) => { body }` – brackets allow us to write multiple statements inside the function, but we need an explicit return to return something.

LET’S SEE SOME CODE!

THE ARROW FUNCTIONS

setTimeout() & setInterval()

- The setTimeout() method calls a function or evaluates an expression after a specified number of milliseconds.

- The setInterval() method calls a function or evaluates an expression at specified intervals (in milliseconds).
  - The setInterval() method will continue calling the function until clearInterval() is called, or the window is closed.
  - The ID value returned by setInterval() is used as the parameter for the clearInterval() method.

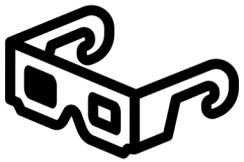
```
var myVar = setInterval(myTimer, 1000);

function myTimer() {
  var d = new Date();
  var t = d.toLocaleTimeString();
  document.getElementById("demo").innerHTML = t;
}

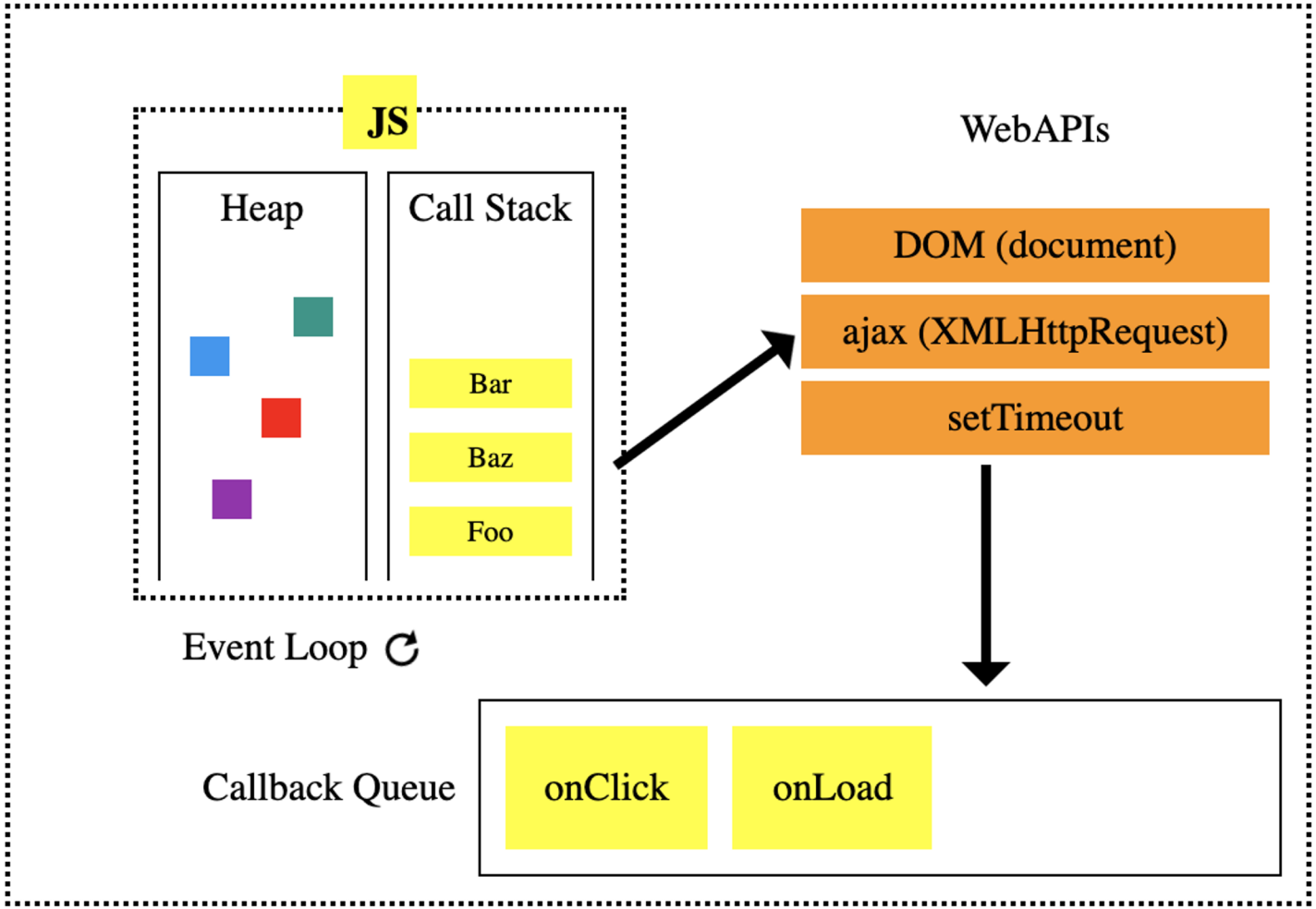
function myStopFunction() {
  clearInterval(myVar);
}
```

THE EVENT LOOP

Event Loop



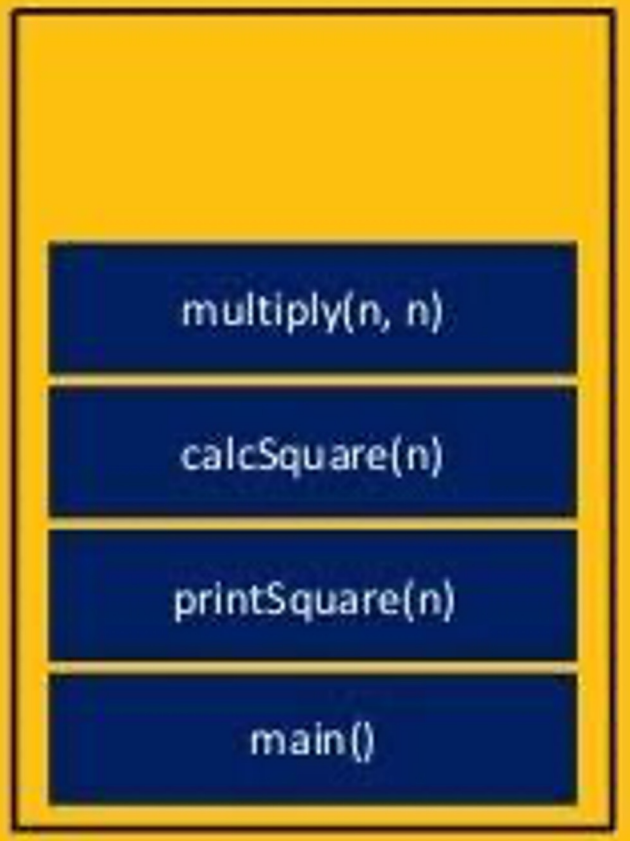
Let's see how it works



THE WEB APIS

- Browsers come with Web APIs (ref) that are able to handle certain tasks in the backgound
  - setTimeout()
  - Making AJAX requests
  - Manipulating DOM
- The JS call stack passes these tasks off to the browser to take care of.
- When the browser finishes those tasks, they return to "**Callback Queue**" and are later pushed to the **Call Stack** as a **Callback.**

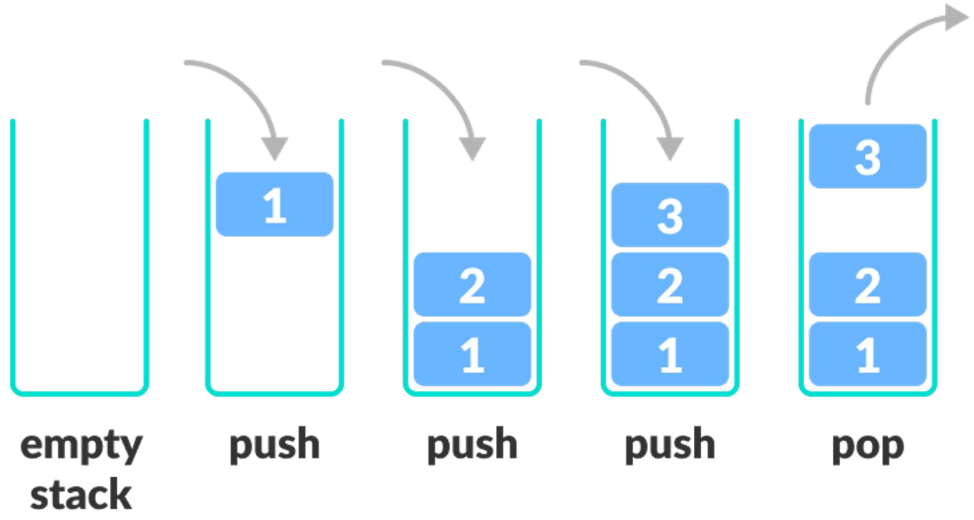
THE CALLSTACK

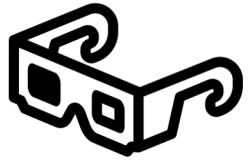


One thread == One call stack == One thing at a time

WHAT IS CALL STACK?

- The call stack is a Last-In, First-Out (LIFO) data structure containing the address at which execution will resume and often local variables and parameters from each call.
- Let's understand the call stack by a simple example.





Let's see how it works

CALLBACK HELL!!!

- **Seasoned JS developers must have heard tha term “Callback hell”. What does it mean?**

LET’S SEE SOME CODE!

PROMISES

WHAT IS A PROMISE?

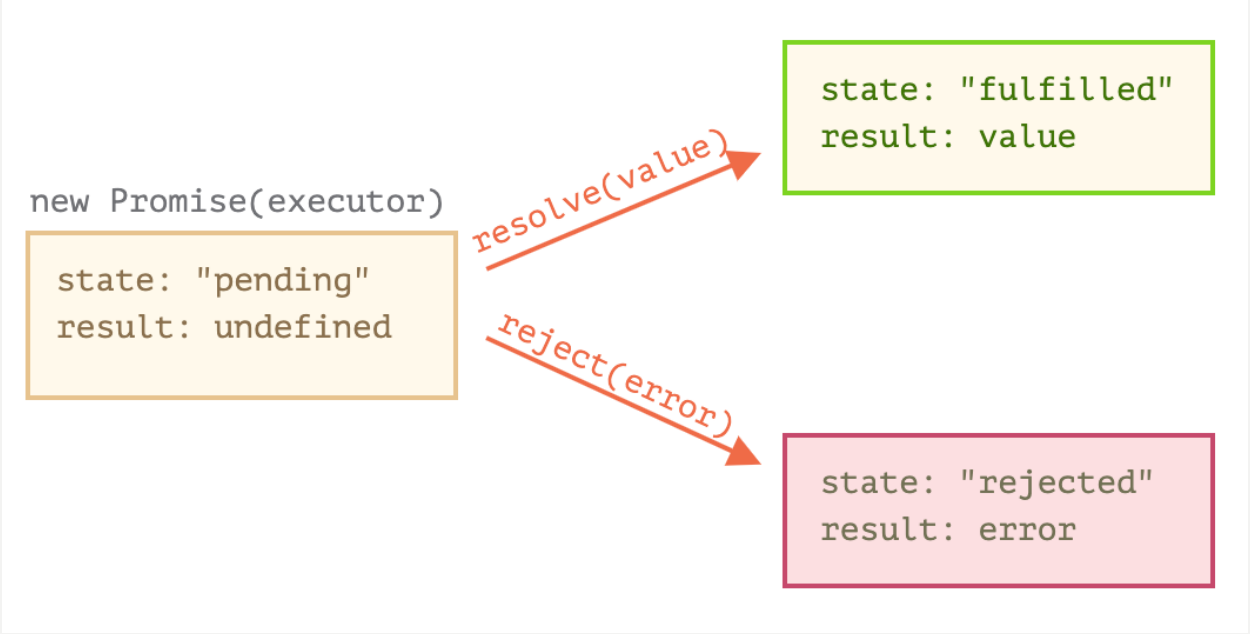
- A Promise in short:

"Imagine you are a **kid**. Your mom **promises** you that she'll get you a **new phone** next week."

- You *don't know* if you will get that phone until next week.
- Your mom can either *really buy* you a new phone, or *she doesn't*, because she is not happy :(

3 STATES OF PROMISE

- That is a **promise**. A promise has 3 states. They are:
  - **Pending**: You *don't know* if you will get that phone
  - **Fulfilled**: Mom is *happy*, she buys you a brand new phone
  - **Rejected**: Mom is *unhappy*, she doesn't buy you a phone




```
new Promise(function (resolve, reject) {  } );
```

Consumers: then, catch and finally

- promise.**then**()
  - The first argument of .**then** is a function that runs when the promise is resolved, and receives the result.
  - The second argument of .**then** is a function that runs when the promise is rejected, and receives the error.
- promise.**catch**()
  - The call .**catch**(func) is a complete analog of .**then**(null, func), it's just a shorthand.
- promise.**finally**()
  - .**finally**() is always run when the promise is settled: be it resolve or reject.

```
let promise = new Promise(function(resolve, reject) {
  setTimeout(() => resolve("done!"), 1000);
});

// resolve runs the first function in .then
promise.then(
  result => alert(result), // shows "done!" after 1 second
  error => alert(error) // doesn't run
);
```

```
let promise = new Promise((resolve, reject) => {
  setTimeout(() => reject(new Error("Whoops!")), 1000);
});

// .catch(f) is the same as promise.then(null, f)
promise.catch(alert); // shows "Error: Whoops!" after 1 second
```

Let’s make some promises

ASYNC & AWAIT

THERE'S A SPECIAL SYNTAX TO WORK WITH PROMISES IN A MORE COMFORTABLE FASHION, CALLED "ASYNC/AWAIT".

ASYNC

- Let's start with the **async** keyword. It can be placed before a function, like this:

```
async function f() {
  return 1;
}
```

- The word "**async**" before a function means one simple thing: a function *always* returns a **promise**. Other values are wrapped in a resolved promise automatically.

```
async function f() {
  return 1;
}

f().then(alert); // 1
```

```
async function f() {
  return Promise.resolve(1);
}

f().then(alert); // 1
```

AWAIT

```
// works only inside async functions
let value = await promise;
```

- The keyword **await** makes JavaScript wait until that promise settles and returns its result.
- Here's an example with a promise that resolves in 1 second:

```
async function f() {

  let promise = new Promise((resolve, reject) => {
    setTimeout(() => resolve("done!"), 1000)
  });

  let result = await promise; // wait until the promise resolves (*)

  alert(result); // "done!"
}

f();
```

The function execution "pauses" at the line (*) and resumes when the promise settles, with result becoming its result.

So the code above shows "done!" in one second.

ERROR HANDLING

```
async function f() {
  throw new Error("Whoops!");
}
```

- If a promise resolves normally, then await promise returns the result. But in the case of a rejection, it throws the error, just as if there were a throw statement at that line.

```
async function f() {

  try {
    let response = await fetch('http://no-such-url');
  } catch(err) {
    alert(err); // TypeError: failed to fetch
  }
}

f();
```

- We can catch that error using **try..catch**, the same way as a regular throw:

SUMMARY

- The **async** keyword before a function has two effects:
  - Makes it always return a promise.
  - Allows await to be used in it.
- The **await** keyword before a promise makes JavaScript wait until that promise settles, and then:
  - If it's an error, the exception is generated — same as if throw error were called at that very place.
  - Otherwise, it returns the result.

LET'S SEE SOME CODE!

Async & Await

WEB PROGRAMMING 0601632 2

BY DR BUNDIT THANASOPON

WHAT IS VUE.JS?

https://vuejs.org/v2/guide/index.html#what-is-vue-js

- Vue (pronounced /vjuː/, like **view**) is a **progressive framework** for building user interfaces.
- The core library is focused on the **view** layer only, and is easy to pick up and integrate with other libraries or existing projects.
- **Free videos:** https://www.vuemastery.com/courses/intro-to-vue-js/vue-instance/

LET'S GET STARTED

- CDN
  - Include this in the header section of your html file
    - `<script src="https://cdn.jsdelivr.net/npm/vue@2.6.12/dist/vue.js"></script>`
- NPM
  - NPM is the recommended installation method when building large scale applications with Vue.
    - $ npm install vue
- CLI
  - Vue provides an **official CLI** for quickly scaffolding ambitious Single Page Applications.
  - **Website:** https://cli.vuejs.org/guide/

THE VUE INSTANCE

CREATING A VUE INSTANCE

- Every Vue application starts by creating a new **Vue instance** with the **Vue** function:

```
var vm = new Vue({
  // options
})
```

- When you create a Vue instance, you pass in an "**options object**".
- For reference, you can also browse the full list of options in the **API reference**.

DATA

- When a Vue instance is created, it adds all the properties found in its **data** object to Vue's **reactivity system**. When the values of those properties change, the view will "react", updating to match the new values.

- When this data changes, the view will re-render.

```
data: {
  newTodoText: '',
  visitCount: 0,
  hideCompletedTodos: false,
  todos: [],
  error: null
}
```

DECLARATIVE RENDERING

- At the core of Vue.js is a system that enables us to declaratively render data to the DOM using straightforward template syntax:

The data and the DOM are now linked, and everything is now **reactive**.

- !!! It should be noted that properties in data are only **reactive** if they existed when the instance was created.

```
<div id="app">
  {{ message }}
</div>
```

```
var app = new Vue({
  el: '#app',
  data: {
    message: 'Hello Vue!'
  }
})
```

INTERPOLATIONS

INTERPOLATIONS

- Text
  - The most basic form of data binding is text interpolation using the "Mustache" syntax (double curly braces):

```
<span>Message: {{ msg }}</span>
```

  - You can also perform one-time interpolations that do not update on data change by using the **v-once directive**

```
<span v-once>This will never change: {{ msg }}</span>
```

- Raw HTML

```
HTML
<p>Using mustaches: {{ rawHtml }}</p>
<p>Using v-html directive: <span v-html="rawHtml"></span></p>
```

Using mustaches: This should be red.
Using v-html directive: This should be red.

```
rawHtml = '<span style="color: red;">Hello</span>'
```

INTERPOLATIONS

- Attributes
  - Mustaches cannot be used inside HTML attributes. Instead, use a **v-bind directive**:

```
<div v-bind:id="dynamicId"></div>

<button v-bind:disabled="isButtonDisabled">Button</button>
```

- Using JavaScript Expressions
  - Vue.js actually supports the full power of JavaScript expressions inside all data bindings:

```
{{ number + 1 }}

{{ message.split('').reverse().join('') }}

<div v-bind:id="'list-' + id"></div>
```

V-BIND DIRECTIVE

```
<div id="app-2">
  <span v-bind:title="message">
    Hover your mouse over me for a few seconds
    to see my dynamically bound title!
  </span>
</div>
```

```
var app2 = new Vue({
  el: '#app-2',
  data: {
    message: 'You loaded this page on ' + new Date().toLocaleString()
  }
})
```

- **Directives** are prefixed with v- to indicate that they are special attributes provided by Vue.
- They apply special reactive behavior to the rendered DOM.

SHORTHANDS

- **v-bind Shorthand**

```
<!-- full syntax -->
<a v-bind:href="url"> ... </a>

<!-- shorthand -->
<a :href="url"> ... </a>
```

- **v-on Shorthand**

```
<!-- full syntax -->
<a v-on:click="doSomething"> ... </a>

<!-- shorthand -->
<a @click="doSomething"> ... </a>
```

CONDITIONAL RENDERING

CONDITIONALS

- **v-if**
  - Conditionally render the element based on the truthy-ness of the expression value.
- **v-else**
  - **Restriction:** previous sibling element must have v-if or v-else-if.
- **v-else-if**
  - **Restriction:** previous sibling element must have v-if or v-else-if.
- **v-show**
  - Toggles the element's display CSS property based on the truthy-ness of the expression value.

```
<div v-if="type === 'A'">
  A
</div>
<div v-else-if="type === 'B'">
  B
</div>
<div v-else-if="type === 'C'">
  C
</div>
<div v-else>
  Not A/B/C
</div>
```

CONDITIONAL GROUPS WITH V-IF ON <TEMPLATE>

- Because v-if is a directive, it has to be attached to a single element.
- But what if we want to toggle more than one element? In this case we can use **v-if** on a <template> element, which serves as an invisible wrapper. The final rendered result will not include the <template> element.

```
<template v-if="ok">
  <h1>Title</h1>
  <p>Paragraph 1</p>
  <p>Paragraph 2</p>
</template>
```

CONTROLLING REUSABLE ELEMENTS WITH KEY

- Vue tries to render elements as efficiently as possible, often re-using them instead of rendering from scratch. Beyond helping make Vue very fast, this can have some useful advantages. For example, if you allow users to toggle between multiple login types:

Then switching the **loginType** in the code above will not erase what the user has already entered.

Since both templates use the same elements, the **<input>** is not replaced - just its placeholder.

```
<template v-if="loginType === 'username'">
  <label>Username</label>
  <input placeholder="Enter your username">
</template>
<template v-else>
  <label>Email</label>
  <input placeholder="Enter your email address">
</template>
```

CONTROLLING REUSABLE ELEMENTS WITH KEY

- This isn't always desirable though, so Vue offers a way for you to say, "These two elements are completely separate - don't re-use them."
- Add a **key** attribute with unique values:

```
<template v-if="loginType === 'username'">
  <label>Username</label>
  <input placeholder="Enter your username" key="username-input">
</template>
<template v-else>
  <label>Email</label>
  <input placeholder="Enter your email address" key="email-input">
</template>
```

LIST RENDERING

MAPPING AN ARRAY TO ELEMENTS WITH V-FOR

- We can use the **v-for** directive to render a list of items based on an array.
- The **v-for** directive requires a special syntax in the form of **item in items**, where **items** is the source data array and **item** is an **alias** for the array element being iterated on:

```
<ul id="example-1">
  <li v-for="item in items" :key="item.message">
    {{ item.message }}
  </li>
</ul>
```

```
var example1 = new Vue({
  el: '#example-1',
  data: {
    items: [
      { message: 'Foo' },
      { message: 'Bar' }
    ]
  }
})
```

V-FOR WITH AN OBJECT

- You can also use v-for to iterate through the properties of an object.

```
<ul id="v-for-object" class="demo">
  <li v-for="value in object">
    {{ value }}
  </li>
</ul>
```

```
new Vue({
  el: '#v-for-object',
  data: {
    object: {
      title: 'How to do lists in Vue',
      author: 'Jane Doe',
      publishedAt: '2016-04-10'
    }
  }
})
```

LIMITATIONS - ARRAY

- Due to limitations in JavaScript, Vue **cannot** detect the following changes to an array:
  - When you directly set an item with the index,
    - e.g. vm.items[indexOfItem] = newValue
  - When you modify the length of the array,
    - e.g. vm.items.length = newLength

- To deal with limitation 1:
  - vm.items.splice(indexOfItem, 1, newValue)
- To deal with limitation 2:
  - vm.items.splice(newLength)

ARRAY CHANGE DETECTION

- Mutation Methods - Vue wraps an observed array's mutation methods so they will also trigger view updates. The wrapped methods are:
  - **push()**
  - **pop()**
  - **shift()**
  - **unshift()**
  - **splice()**
  - **sort()**
  - **reverse()**

LIMITATIONS - OBJECT

- **Vue cannot detect property addition or deletion!**
- Vue does not allow dynamically adding new root-level reactive properties to an already created instance. For example:

```
var vm = new Vue({
  data: {
    a: 1
  }
})
// `vm.a` is now reactive

vm.b = 2
// `vm.b` is NOT reactive
```

However, it's possible to add reactive properties to a nested object using the Vue.set(object, propertyName, value) method.

```
Vue.set(vm.someObject, 'b', 2)
```

MORE WITH V-FOR

- **v-for** with a Range – **v-for** can also take an integer. In this case it will repeat the template that many times.

```
<div>
  <span v-for="n in 10">{{ n }} </span>
</div>
```

Result:

1 2 3 4 5 6 7 8 9 10

- **v-for** on a <template> – Similar to template **v-if**, you can also use a <template> tag with **v-for** to render a block of multiple elements. For example:

```
<ul>
  <template v-for="item in items">
    <li>{{ item.msg }}</li>
    <li class="divider" role="presentation"></li>
  </template>
</ul>
```

V-FOR WITH V-IF

Note!!!
it's **not** recommended to use v-if and v-for together.

- When they exist on the same node, **v-for** has a higher priority than **v-if**. That means the v-if will be run on each iteration of the loop separately.

```
<li v-for="todo in todos" v-if="!todo.isComplete">
  {{ todo }}
</li>
```

Bad

```
<ul v-if="todos.length">
  <li v-for="todo in todos">
    {{ todo }}
  </li>
</ul>
<p v-else>No todos left!</p>
```

Good

- The example above only renders the todos

that are not complete.

CLASS & STYLE BINDINGS

BINDING HTML CLASSES OBJECT SYNTAX

- We can pass an object to **v-bind:class** to dynamically toggle classes:

```
<div v-bind:class="{ active: isActive }"></div>
```

- You can have multiple classes toggled by having more fields in the object.
- In addition, the **v-bind:class** directive can also co-exist with the plain class attribute. So given the following template:

```
<div
  class="static"
  v-bind:class="{ active: isActive, 'text-danger': hasError }"
></div>
```

```
data: {
  isActive: true,
  hasError: false
}
```

BINDING HTML CLASSES ARRAY SYNTAX

- We can pass an **array** to **v-bind:class** to apply a list of classes:

```
<div v-bind:class="[activeClass, errorClass]"></div>
```

```
data: {
  activeClass: 'active',
  errorClass: 'text-danger'
}
```

- If you would like to also toggle a class in the list conditionally, you can do it with a ternary expression:

```
<div v-bind:class="[isActive ? activeClass : '', errorClass]"></div>
```

```
<div v-bind:class="[{ active: isActive }, errorClass]"></div>
```

BINDING INLINE STYLES - OBJECT SYNTAX

- The object syntax for **v-bind:style** is pretty straightforward - it looks almost like CSS, except it's a JavaScript object. You can use either camelCase or kebab-case (use quotes with kebab-case) for the CSS property names:

```
<div v-bind:style="styleObject"></div>
```

```
data: {
  styleObject: {
    color: 'red',
    fontSize: '13px'
  }
}
```

```
<div v-bind:style="{ color: activeColor, fontSize: fontSize + 'px' }"></div>
```

```
data: {
  activeColor: 'red',
  fontSize: 30
}
```

BINDING INLINE STYLES - ARRAY SYNTAX

- The array syntax for **v-bind:style** allows you to apply multiple style objects to the same element

```
<div v-bind:style="[baseStyles, overridingStyles]"></div>
```

WEB PROGRAMMING 0601632 2

BY DR BUNDIT THANASOPON

FORM INPUT BINDINGS

V-MODEL MAGIC

You can use the v-model directive to create **two-way data bindings** on form input, textarea, and select elements. It automatically picks the correct way to update the element based on the input type.

```
<input v-model="message" placeholder="edit me">
<p>Message is: {{ message }}</p>
```

```
var app = new Vue({
    el: '#app',
    data: {
        message: 'Hello Vue!'
    }
})
```

v-model will ignore the initial **value**, **checked**, or **selected** attributes found on any form elements.

V-MODEL EXAMPLE

- <input type=”text” />
- <textarea>
- <select>
- <checkbox>
- <radio>
- <input type=“date” />
- <input type=“time” />
- <input type=“color” />

VUE EVENT HANDLING

JAVASCRIPT EVENTS

- **HTML events** are **"things"** that happen to HTML elements.
- When JavaScript is used in HTML pages, JavaScript can **"react"** on these events.

- An HTML event can be something the browser does, or something a user does.
- Here are some examples of HTML events:
  - An HTML web page has finished loading
  - An HTML input field was changed
  - An HTML button was clicked
- Often, when events happen, you may want to do something. JavaScript lets you execute code when events are detected.

JAVASCRIPT EVENTS

```
<button onclick="document.getElementById('demo').innerHTML =
Date()">The time is?</button>

<button onclick="this.innerHTML = Date()">The time is?</button>

<button onclick="displayDate()">The time is?</button>
```

HANDLE EVENT IN VUEJS

```
// inline
<button onclick="this.innerHTML = Date()">The time is?</button>
<button v-on:click=”now = Date()">The time is {{ now }}</button>
<button @click=”now = Date()">The time is {{ now }}</button>

// method
<button onclick="displayDate()">The time is?</button>
<button v-on:click="displayDate()">The time is?</button>
<button @click="displayDate()">The time is?</button>
```

Handling event in Vue is very similar to vanilla JavaScript, just change `onclick` to `@click`

HANDLE EVENT WITH METHODS

```
<button @click="displayDate()">The time is?</button>
```

Normally handling event is more complex than one-line JavaScript, for that we can use "methods"

```
var example2 = new Vue({
    el: '#app',
    data: {
        now: ''
    },
    methods: {
        displayDate() {
            // this is a function
            // write any js you want!
            this.now = Date()
        }
    }
})
```

METHODS ARGUMENTS

```
<button @click=”say(‘hi’)">Hi</button>
<button @click=”say(‘hello’)">Hello</button>
```

You can also call methods with arguments

```
var example2 = new Vue({
    el: ‘#app’,
    methods: {
        say(msg) {
            alert(msg)
        }
    }
})
```

THE $EVENT OBJECT

```
<button @click=”say($event)">Hi</button>
<button @click=”say($event)">Hello</button>
```

You can also access DOM Event by using the $event special variable

```
var app = new Vue({
    el: ‘#app’,
    methods: {
        say(e) {
            alert(e.target.innerHTML)
        }
    }
})
```

EVENT PROPAGATION

- For most event types, handlers registered on nodes with children will also receive events that happen in the children.
- If a button inside a paragraph is clicked, event handlers on the paragraph will also see the click event.

- At any point, an event handler can call the **stopPropagation** method on the event object to prevent handlers further up from receiving the event

```
<p>A paragraph with a <button>button</button>.</p>

<script>
  let para = document.querySelector("p");
  let button = document.querySelector("button");
  para.addEventListener("mousedown", () => {
    console.log("Handler for paragraph.");
  });
  button.addEventListener("mousedown", event => {
    console.log("Handler for button.");
    if (event.button == 2) event.stopPropagation();
  });
</script>
```

DEFAULT ACTIONS

- Many events have a default action associated with them.
  - If you click a link, you will be taken to the link's target.
  - If you press the down arrow, the browser will scroll the page down.
  - If you right-click, you'll get a context menu.
- You can use **preventDefault** to prevent the default action from happening.

```
<a href="https://developer.mozilla.org/">MDN</a>
<script>
  let link = document.querySelector("a");
  link.addEventListener("click", event => {
    console.log("Nope.");
    event.preventDefault();
  });
</script>
```

EVENT MODIFIER

To make handling event easier. Vuejs provided Event Modifier to do common task like stopPropagation() or preventDefault()

- .stop
- .prevent
- .capture
- .self
- .once
- .passive

```
<!-- the click event's propagation will be stopped -->
<a v-on:click.stop="doThis"></a>

<!-- the submit event will no longer reload the page -->
<form v-on:submit.prevent="onSubmit"></form>

<!-- modifiers can be chained -->
<a v-on:click.stop.prevent="doThat"></a>

<!-- just the modifier -->
<form v-on:submit.prevent></form>
```

COMPUTED PROPERTIES AND WATCHERS

COMPUTED PROPERTIES

- In-template expressions are very convenient, but they are meant for simple operations.
- Putting too much logic in your templates can make them bloated and hard to maintain.

```
<div id="example">
  {{ message.split('').reverse().join('') }}
</div>
```

- The problem is made worse when you want to include the reversed message in your template more than once.
- That's why for any complex logic, you should use a **computed property**.

BASIC EXAMPLES

```
<div id="example">
  <p>Original message: "{{ msg }}"</p>
  <p>Computed reversed message: "{{ reversedMessage }}"</p>
</div>
```

```
var vm = new Vue({
    el: '#example’,
    data: {
        msg: 'Hello’
    },
    computed: {
                reversedMessage() {
                    return this.msg.split('').reverse().join(’’)
                }
    }
})
```

Original message: "Hello"

Computed reversed message: "olleH"

COMPUTED CACHING VS METHODS

- You may have noticed we can achieve the same result by invoking a method in the expression.
- For the end result, the two approaches are indeed exactly the same. However, the difference is that **computed properties are cached based on their dependencies.**

- A computed property will only re-evaluate when some of its dependencies have changed.

```
methods: {
        reverseMessage: function () {
                return this.message.split('').reverse().join('')
        }
}
```

WATCHERS

- Watchers are most useful when you want to perform asynchronous or expensive operations in response to changing data.

```
watch: {
        // whenever question changes, this function will run
        question: function (newQuestion, oldQuestion) {
                this.answer = 'Waiting for you to stop typing...'
                this.debouncedGetAnswer()
        }
},
```

COMPUTED VS WATCHED PROPERTY

- When you have some data that needs to change based on some other data, it is tempting to overuse watch.
- However, it is often a better idea to use a computed property rather than an imperative watch callback.

- Try to do with computed!

```
var vm = new Vue({
        el: '#demo',
        data: {
            firstName: 'Foo',
            lastName: 'Bar',
            fullName: 'Foo Bar'
        },
        watch: {
            firstName: function (val) {
            this.fullName = val + ' ' + this.lastName
            },
            lastName: function (val) {
            this.fullName = this.firstName + ' ' + val
            }
        }
})
```

VUE INSTANCE LIFE CYCLE

LIFE CYCLE HOOK

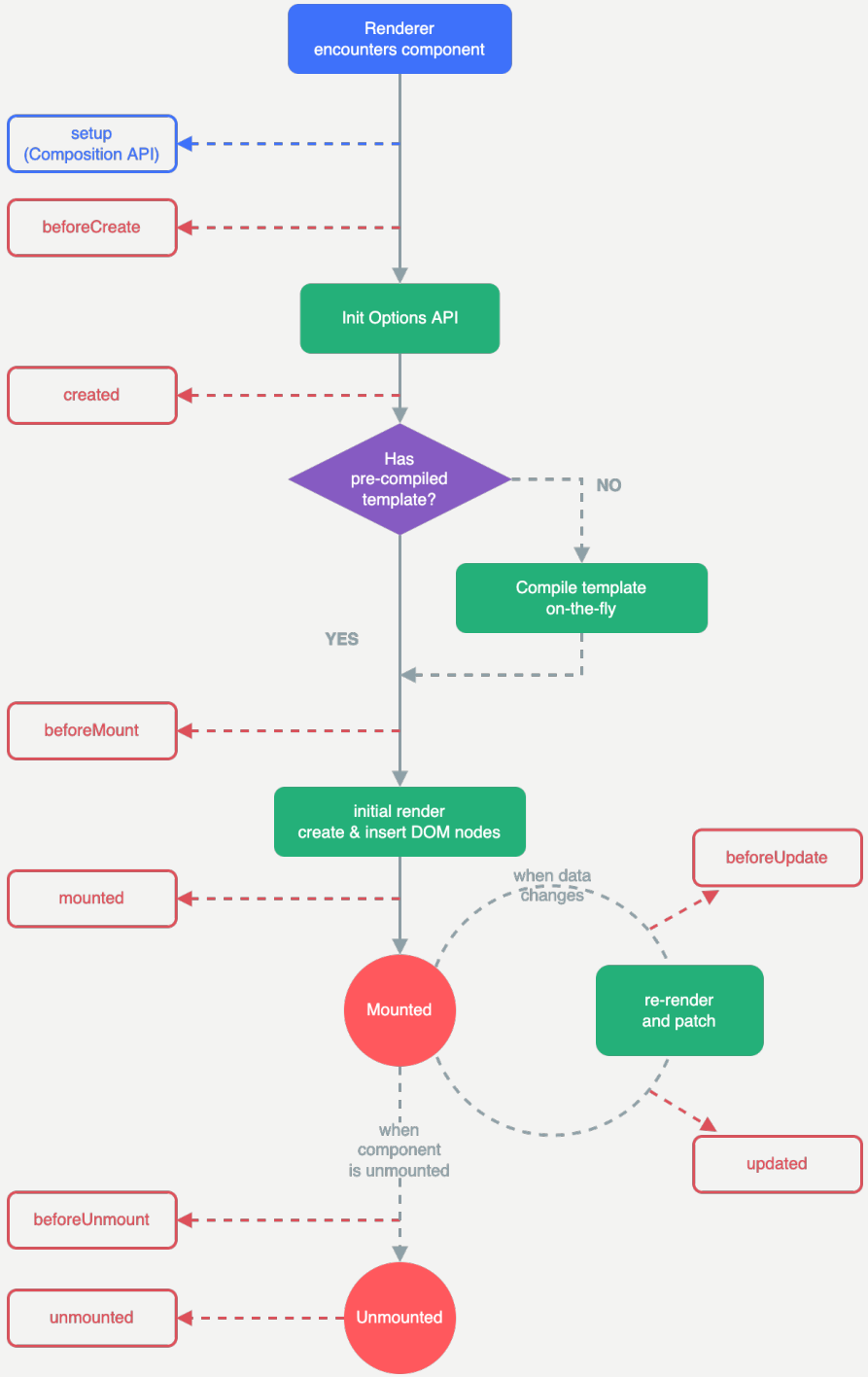
- Each Vue instance goes through a series of initialization steps when it's created
- it needs to set up data observation
- compile the template
- mount the instance to the DOM
- update the DOM when data changes.

Along the way, it also runs functions called **lifecycle hooks**, giving users the opportunity to add their own code at specific stages.

```
var vm = new Vue({
    el: '#example',
    data: {
        msg: 'Hello'
    },
    created () {
        // code to run when vue is created
    },
    mounted () {
        // code to run when vue is mounted
    }
})
```

LIFE CYCLE DIAGRAM

- https://vuejs.org/guide/essentials/lifecycle.html#lifecycle-diagram



INTRODUCTION TO NODE + EXPRESS

Web Programming

TODAY'S TOPICS

- What is Node.js
- The Node REPL + process
- NPM + package.json
- Intro to Express + installation
- Hello world app
- Express basics
  - The Request and Response objects
  - Basic routing
  - Path parameters
  - Query string
  - Serving static files
- Templating basics

WHAT IS NODE.JS

- Node.js is an asynchronous event-driven JavaScript runtime.
- What Can Node.js Do?
  - Can generate dynamic page content
  - Can create, open, read, write, delete, and close files on the server
  - Can collect form data
  - Can add, delete, modify data in your database

WHY NODE.JS

Node.js uses asynchronous programming! -> JavaScript

A common task for a web server can be to open a file on the server and return the content to the client.

HOW PHP OR ASP HANDLES A FILE REQUEST:

1. Sends the task to the computer's file system.
2. Waits while the file system opens and reads the file.
3. Returns the content to the client.
4. Ready to handle the next request.

HOW NODE.JS HANDLES A FILE REQUEST:

1. Sends the task to the computer's file system.
2. Ready to handle the next request.
3. When the file system has opened and read the file, the server returns the content to the client.

LET'S TRY NODE.JS

The node REPL = Chrome's console

WRITING YOUR FIRST NODE.JS FILE

- Create a .js file and write some JavaScript code.
- In Terminal/Powershell use this command
  - > node filename.js

NODE.JS MODULES

- What is a Module in Node.js?
  - Consider modules to be the same as JavaScript libraries.
  - A set of functions you want to include in your application.

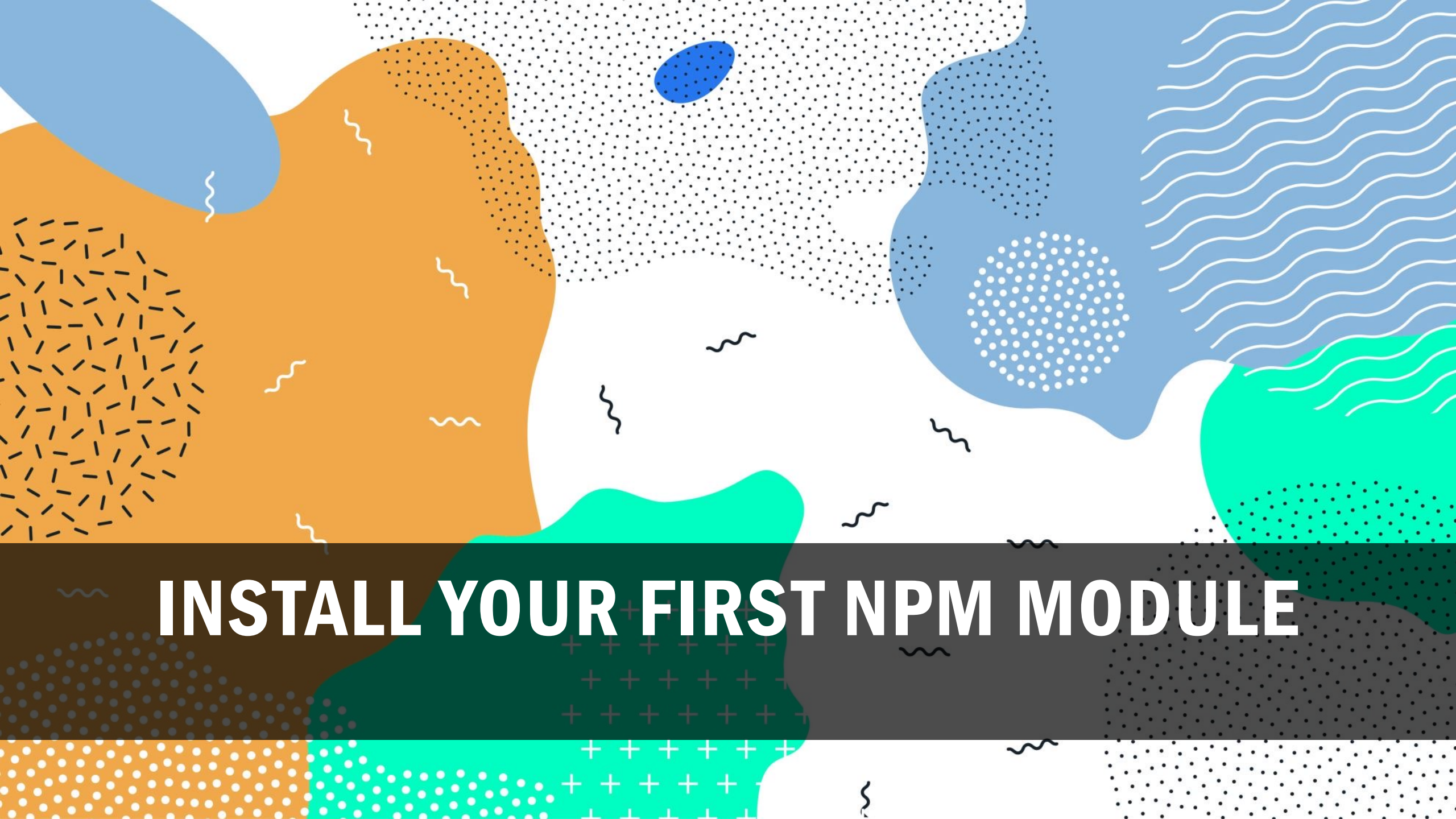
- Built-in Modules
  - Node.js has a set of built-in modules which you can use without any further installation. (ref)
- Custom Modules
  - You can also create your own modules or use others' modules

LET'S CREATE A NODE.JS MODULE

- require
- module.exports

WHAT IS NPM?

- NPM is a package manager for Node.js packages, or modules if you like.
- www.npmjs.com hosts thousands of free packages to download and use.
- The NPM program is installed on your computer when you install Node.js
- What is a Package?
  - A package in Node.js contains all the files you need for a module.
  - Modules are JavaScript libraries you can include in your project.


INSTALL YOUR FIRST NPM MODULE

UPPER-CASE

- In Terminal/Powershell, go to your project directory
- Type this command:

> npm install upper-case

- NPM creates a folder named "node_modules", where the package will be placed.
- All packages you install in the future will be placed in this folder.

upper-case TS

2.0.2 • Public • Published 3 months ago

Readme | Explore BETA | 1 Dependency

Upper Case

npm v2.0.2 | downloads 29M/month | minzipped size 371 B

> Transforms the string to upper case.

Installation

```
npm install upper-case --save
```

Usage

```
import { upperCase, localeUpperCase } from "upper-case";

upperCase("string"); //=> "STRING"

localeUpperCase("string", "tr"); //=> "STRİNG"
```

PACKAGE.JSON

- The package.json file is kind of a manifest for your project.
- It's a central repository of configuration for tools, for example. It's also where npm stores the names and versions for all the installed packages.
- Let’s try:
  - Create a folder
  - Type command: npm init


express

INTRODUCTION

Express (https://expressjs.com/) is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications.

What is a framework?

INSTALLATION

Node.js must be installed first!

1. Open the Terminal/Powershell
2. Create your project folder: mkdir myproject
3. Go to the folder: cd myproject
4. Create package.json file: npm init
5. Install Express: npm install express


HELLO WORLD APP
Let’s create your first Express application

THE REQUEST AND RESPONSE OBJECTS

- The req object represents the HTTP request and has properties for the request query string, parameters, body, HTTP headers, and so on. (https://expressjs.com/en/4x/api.html#req)

- The res object represents the HTTP response that an Express app sends when it gets an HTTP request. (https://expressjs.com/en/4x/api.html#res)

EXPRESS ROUTING BASICS

Routing refers to determining how an application responds to a client request to a particular endpoint, which is a URI (or path) and a specific HTTP request method (GET, POST, and so on).

- Each route can have one or more handler functions, which are executed when the route is matched.

```
app.METHOD(PATH, HANDLER)
```

```
app.get('/', function (req, res) {
  res.send('Hello World!')
})
```

PATH PARAMETERS

Documentation:
https://expressjs.com/en/guide/routing.html

Sometimes we want to pass parameters with the URL.

- Route parameters are named URL segments that are used to capture the values specified at their position in the URL.
- The captured values are populated in the req.params object.

```
app.get('/users/:userId/books/:bookId', function (req, res) {
  res.send(req.params)
})
```

```
Route path: /users/:userId/books/:bookId
Request URL: http://localhost:3000/users/34/books/8989
req.params: { "userId": "34", "bookId": "8989" }
```

QUERY STRING

```
// GET /search?q=tobi+ferret
console.dir(req.query.q)
// => 'tobi ferret'

// GET /shoes?order=desc&shoe[color]=blue&shoe[type]=converse
console.dir(req.query.order)
// => 'desc'

console.dir(req.query.shoe.color)
// => 'blue'

console.dir(req.query.shoe.type)
// => 'converse'

// GET /shoes?color[]=blue&color[]=black&color[]=red
console.dir(req.query.color)
// => ['blue', 'black', 'red']
```

SERVING STATIC FILES

To serve static files such as images, CSS files, and JavaScript files, use the express.static built-in middleware function in Express.

```
express.static(root, [options])
```

- The root argument specifies the root directory from which to serve static assets.
- For more information on the options argument, see express.static.

TEMPLATING BASICS

Templating allows us to define a preset pattern for a webpage, that we can dynamically modify.

CREATE YOUR FIRST TEMPLATE

Steps:

1. Call app.set() in your index.js file
2. Create “views” folder in your project directory
3. Create views/home.ejs and write some HTML code
4. Call res.render(‘home.ejs’) in the controller

```
app.set('view engine', 'ejs');
```

```
app.get("/", (req, res) => {
  res.render('home.ejs')
});
```

EJS SYNTAX

Interpolation

- <% 'Scriptlet' tag, for control-flow, no output
- <%= Outputs the value into the template (HTML escaped)
- <%- Outputs the unescaped value into the template
- <%# Comment tag, no execution, no output
- %> Ending tag

PASSING DATA TO TEMPLATE

```
app.get("/", (req, res) => {
  res.render('home.ejs',
    {
      title: "This is my home",
      message: "Welcome to Web Programming!"
    }
  )
});
```

LET'S BEUTIFY OUR HOME PAGE WITH BULMA

EXPRESS + MYSQL

Web Programming

TODAY'S TOPICS

- Installation MySQL driver for Node.js
- Establishing connections
- Performing queries
  - Escaping values
  - Preparing queries
  - Getting inserted id
  - Getting number of affected rows vs. changed rows
- Transactions
- Error handling

INSTALLATION

Once you have MySQL up and running on your computer, you can access it by using Node.js.

- To access a MySQL database with Node.js, you need a MySQL driver.
- We are going to use “mysql2” package (https://www.npmjs.com/package/mysql2) which can be downloaded from “npm”
  - Go to your project folder
  - Type command: npm install mysql2

ESTABLISHING CONNECTIONS

Connection options

- **host**: The hostname of the database you are connecting to. (Default: localhost)
- **port**: The port number to connect to. (Default: 3306)
- **user**: The MySQL user to authenticate as.
- **password**: The password of that MySQL user.
- **database**: Name of the database to use for this connection.

```
1   const mysql = require('mysql2/promise');
2
3   const conn = mysql.createConnection({
4     host: 'localhost',
5     user: 'root',
6     password: '...',
7     database: 'webpro',
8   })
```

PERFORMING QUERIES

The most basic way to perform a query is to call the .query() method on a Connection object

```
try{
    const [rows, fields] = await conn.query('SELECT * FROM blogs')
    console.log('rows:', rows)
}catch(err){
    console.log(err)
}
```

ESCAPING QUERY VALUES

Alternatively, you can use ? characters as placeholders for values you would like to have escaped like this:

```
try{
    const [rows, fields] = await conn.query(
        'INSERT INTO `blogs`(title, content, status, pinned) VALUES(?, ?, ?, ?);',
        ['My new blog', 'TEST CONTENT', '01', 0]
    )
    console.log('rows:', rows)
}catch(err){
    console.log(err)
}
```

ESCAPING QUERY VALUES (2)

Multiple placeholders are mapped to values in the same order as passed.

```
try{
    const [rows, fields] = await conn.query(
        'UPDATE `blogs` SET title = ?, content = ? WHERE id = ?;',
        ['My new blog 3', 'TEST CONTENT 2', 13]
    )
    console.log('rows:', rows)
}catch(err){
    console.log(err)
}
```

GETTING THE ID OF AN INSERTED ROW

If you are inserting a row into a table with an auto increment primary key, you can retrieve the insert id like this:

```
try{
    const [rows, fields] = await conn.query(
        'INSERT INTO `blogs`(title, content, status, pinned) VALUES(?, ?, ?, ?);',
        ['My new blog', 'TEST CONTENT', '01', 0]
    )
    console.log('insertId:', rows.insertId)
}catch(err){
    console.log(err)
}
```

GETTING THE NUMBER OF AFFECTED ROWS

You can get the number of affected rows from an insert, update or delete statement.

```
try{
    const [rows, fields] = await conn.query(
        'INSERT INTO `blogs`(title, content, status, pinned) VALUES(?, ?, ?, ?);',
        ['My new blog', 'TEST CONTENT', '01', 0]
    )
    console.log('affectedRows:', rows.affectedRows)
}catch(err){
    console.log(err)
}
```

GETTING THE NUMBER OF CHANGED ROWS

You can get the number of changed rows from an update statement.

- "changedRows" differs from "affectedRows" in that it does not count updated rows whose values were not changed.

```
try{
    const [rows, fields] = await conn.query(
        'UPDATE `blogs` SET title = ?, content = ? WHERE id = ?;',
        ['My new blog 3', 'TEST CONTENT 2', 13]
    )
    console.log('changedRows:', rows.changedRows)
}catch(err){
    console.log(err)
}
```

TRANSACTIONS

A transaction can be defined as a group of tasks. A single task is the minimum processing unit which cannot be divided further.

```
const conn = await pool.getConnection()
// Begin transaction
await conn.beginTransaction();

try{

  let results = await conn.query(
    "INSERT INTO blogs(title, content, status) VALUES(?, ?, ?);",
    [title, content, status]
  )

  const blogId = results[0].insertId;

  results = await conn.query(
    "INSERT INTO images(blog_id, file_path) VALUES(?, ?);",
    [blogId, file.path])

  await conn.commit()
  res.send("success!");
}catch(err){
  await conn.rollback();
  next(err);
}finally{
  console.log('finally')
  conn.release();
}
```

LETS' CREATE A NEW BLOG

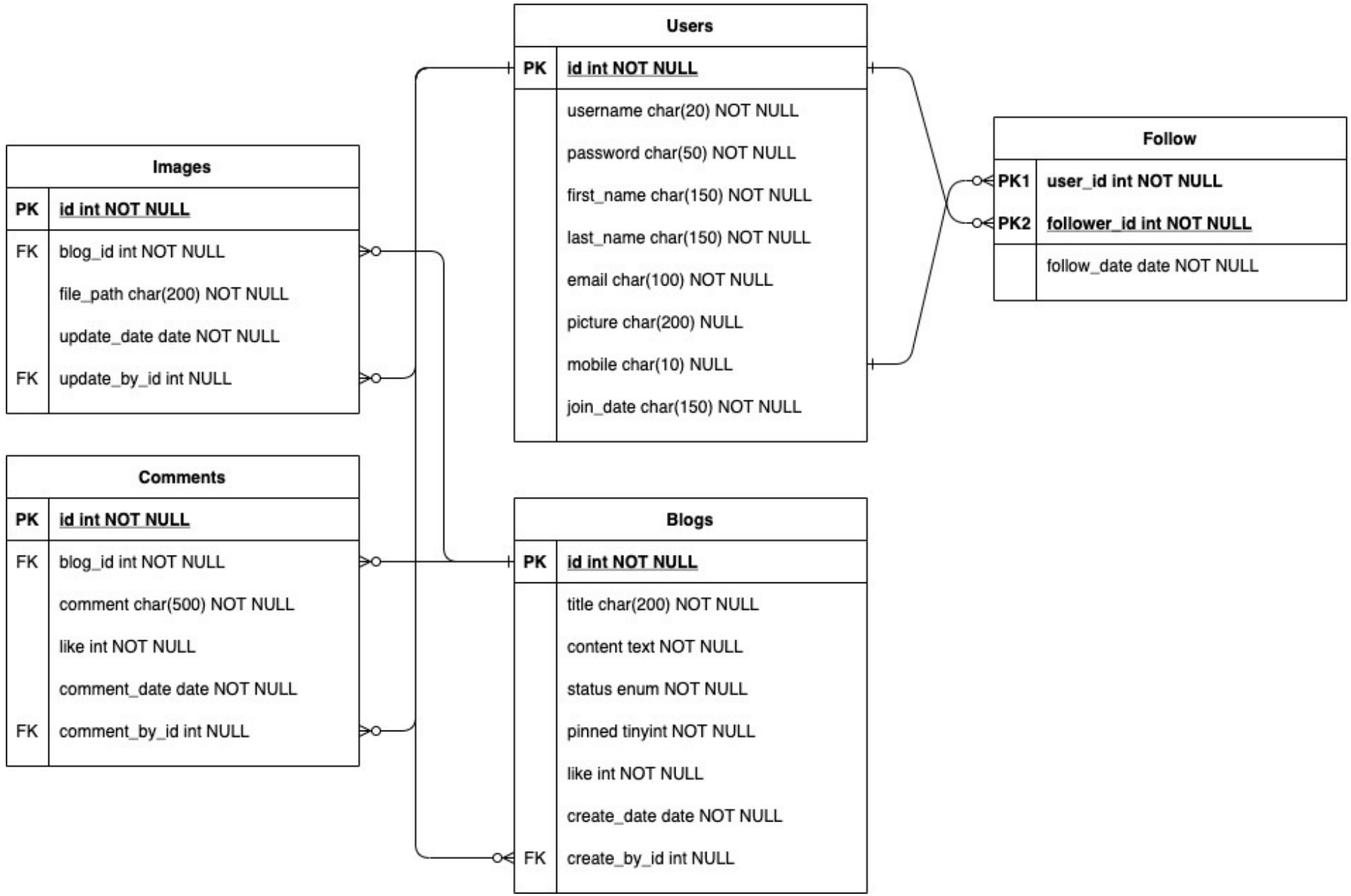
Steps:

1. Install "multer": > npm install multer
2. Create a controller
3. Get data from the POST request
4. Insert into blogs table
5. Insert successfully uploaded file paths in to images table

If something go wrong rollback everything!

YOUBLOG ER DIAGRAM

Let’s migrate the database and do some queries


Users
PK id int NOT NULL
username char(20) NOT NULL
password char(50) NOT NULL
first_name char(150) NOT NULL
last_name char(150) NOT NULL
email char(100) NOT NULL
picture char(200) NULL
mobile char(10) NULL
join_date char(150) NOT NULL
Follow
PK1 user_id int NOT NULL
PK2 follower_id int NOT NULL
follow_date date NOT NULL
Images
PK id int NOT NULL
FK blog_id int NOT NULL
file_path char(200) NOT NULL
update_date date NOT NULL
FK update_by_id int NULL
Comments
PK id int NOT NULL
FK blog_id int NOT NULL
comment char(500) NOT NULL
like int NOT NULL
comment_date date NOT NULL
FK comment_by_id int NULL
Blogs
PK id int NOT NULL
title char(200) NOT NULL
content text NOT NULL
status enum NOT NULL
pinned tinyint NOT NULL
like int NOT NULL
create_date date NOT NULL
FK create_by_id int NULL

INTRO TO RESTFUL API

Web Programming

WHAT IS REST?

RESTful API

GET PUT POST DELETE

REST is acronym for **RE**presentational **S**tate **T**ransfer. It is architectural style for **distributed hypermedia systems** and was first presented by Roy Fielding in 2000 in his famous dissertation.

REST is essentially a guideline!

RESOURCE

The key abstraction of information in REST is a **resource**.

- Any information that can be named can be a resource: a document or image, a temporal service, a collection of other resources, a non-virtual object (e.g. a person), and so on.
- REST uses a **resource identifier** to identify the particular resource involved in an interaction between components.
- The data format of a representation is known as a media type. The media type identifies a specification that defines how a representation is to be processed.

RESOURCE METHODS

Another important thing associated with REST is **resource methods** to be used to perform the desired transition.

- **GET – retrieving resource**
- **PUT/PATCH – updating**
- **POST – creating**
- **DELETE - deleting**

CRUD

LET'S DESIGN OUR BLOG APP RESTFUL

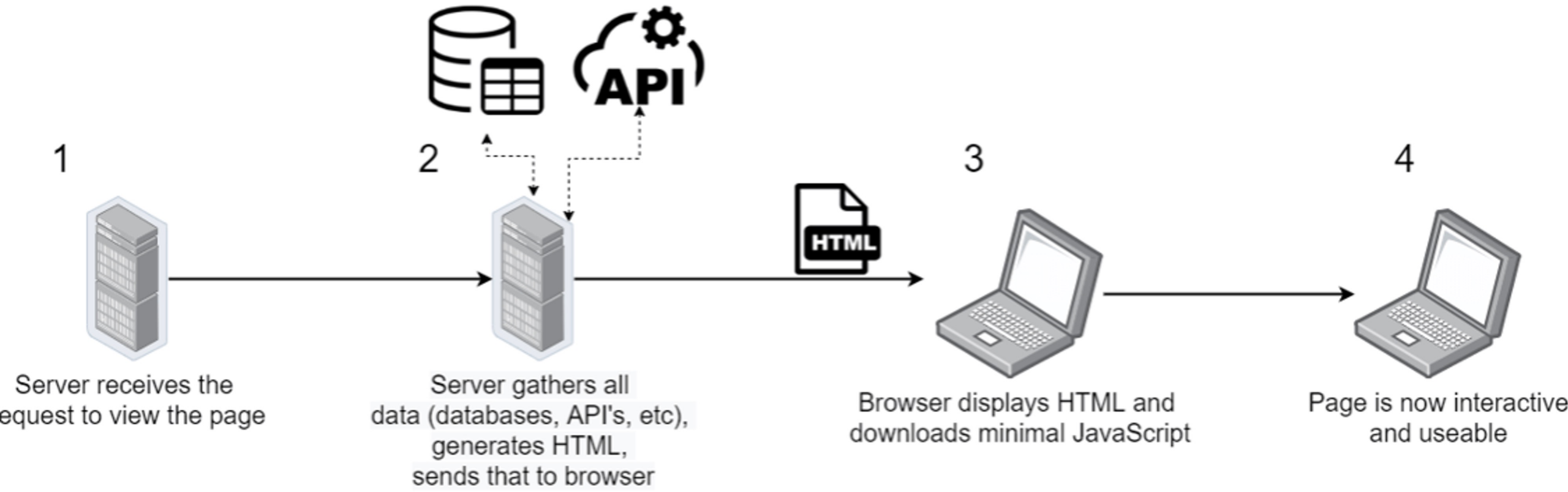
- GET "/blogs"
- POST "/blogs"
- PUT "/blogs/:id"
- GET "/blogs/:id"
- DELETE "/blogs/:id"

- GET "/comments/:blog_id"
- POST "/comments/:blog_id"
- PUT "/comments/:blog_id/:id"
- GET "/comments/:blog_id/:id"
- DELETE "/comments/:blog_id/:id"

AJAX + AXIOS

Web Programming

SSR



res.render('index')

Loading

Loading

Visible but not
interactive

Complete!

CSR

Front-end Server

Back-end Server

API

1

Server sends response to browser with minimal HTML

2

Browser downloads the JavaScript

3

Browser executes JavaScript starting frontend framework and gathers required data

4

Page is now interactive and ready for visitor

Loading

Loading

Loading

Complete!

TODAY'S TOPICS

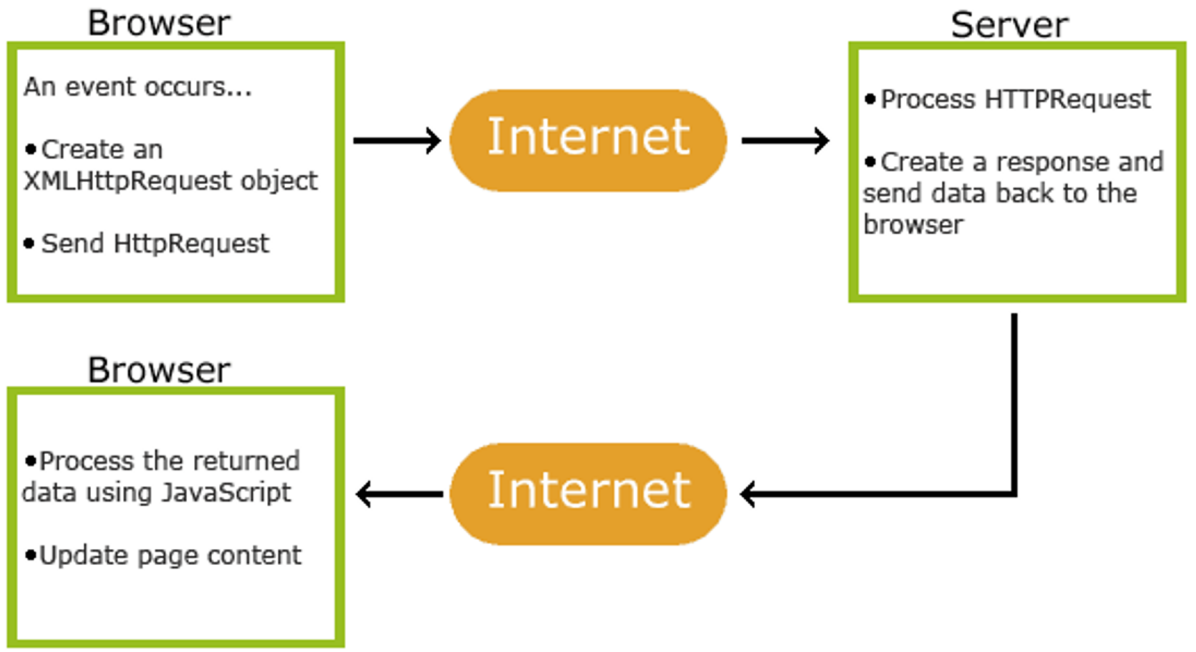
- AJAX
  - What is AJAX
  - Fetch API
- Axios
  - Consuming REST APIs
- Linking front-end with back-end
  - Vue CLI
  - Vue Router
  - Let's update the "youblog" website

WHAT IS AJAX?

- AJAX = **A**synchronous **J**avaScript **A**nd **X**ML.
- AJAX is not a programming language.
- AJAX is a technique that uses a combination of:
  - A browser built-in XMLHttpRequest object (to request data from a web server)
  - JavaScript and HTML DOM (to display or use the data)

HOW AJAX WORKS

1. An event occurs ... (clicking a button)
2. An XMLHttpRequest object is created by JavaScript
3. The XMLHttpRequest object sends a request to a web server
4. The server processes the request
5. The server sends a response back to the web page
6. The response is read by JavaScript
7. Proper action (like page update) is performed by JavaScript



AJAX

AJAX is a developer's dream, because you can:

- Read data from a web server - after the page has loaded
- Update a web page without reloading the page
- Send data to a web server - in the background

Let’s see some example

```
fetch('http://example.com/movies.json')
  .then(response => response.json())
  .then(data => console.log(data));
```

FETCH API

- The Fetch API provides an interface for fetching resources (including across the network).
- Fetch provides a generic definition of Request and Response objects.
  - The fetch() method takes one mandatory argument, the path to the resource you want to fetch.
  - It returns a Promise that resolves to the Response to that request, whether it is successful or not.

AXIOS

Promise based HTTP client for the browser and node.js

FEATURES

- Make XMLHttpRequests from the browser
- Make http requests from node.js
- Supports the Promise API
- Intercept request and response
- Transform request and response data
- Cancel requests
- Automatic transforms for JSON data
- Client side support for protecting against XSRF

GET REQUESTS

```
const axios = require('axios');

// Make a request for a user with a given ID
axios.get('/user?ID=12345')
  .then(function (response) {
    // handle success
    console.log(response);
  })
  .catch(function (error) {
    // handle error
    console.log(error);
  })
  .then(function () {
    // always executed
  });
```

Async & Await

```
async function getUser() {
  try {
    const response = await axios.get('/user?ID=12345');
    console.log(response);
  } catch (error) {
    console.error(error);
  }
}
```

POST REQUESTS

```
axios.post('/user', {
    firstName: 'Fred',
    lastName: 'Flintstone'
  })
  .then(function (response) {
    console.log(response);
  })
  .catch(function (error) {
    console.log(error);
  });
```

REQUEST METHOD ALIASES

- axios.request(config)
- axios.get(url[, config])
- axios.delete(url[, config])
- axios.head(url[, config])
- axios.options(url[, config])
- axios.post(url[, data[, config]])
- axios.put(url[, data[, config]])
- axios.patch(url[, data[, config]])

For more detail:
https://www.npmjs.com/package/axios

Vue CLI

🛠️ Standard Tooling for Vue.js Development

VUE CLI

Vue Command-line Interface

VUE CLI

Vue CLI is a full system for rapid Vue.js development, providing:

1. Interactive project scaffolding via @vue/cli.
2. Zero config rapid prototyping via @vue/cli + @vue/cli-service-global.
3. A runtime dependency (@vue/cli-service) that is:
   - Upgradeable;
   - Built on top of webpack, with sensible defaults;
   - Configurable via in-project config file;
   - Extensible via plugins
4. A rich collection of official plugins integrating the best tools in the frontend ecosystem.
5. A full graphical user interface to create and manage Vue.js projects.

VUE ROUTER

https://router.vuejs.org/

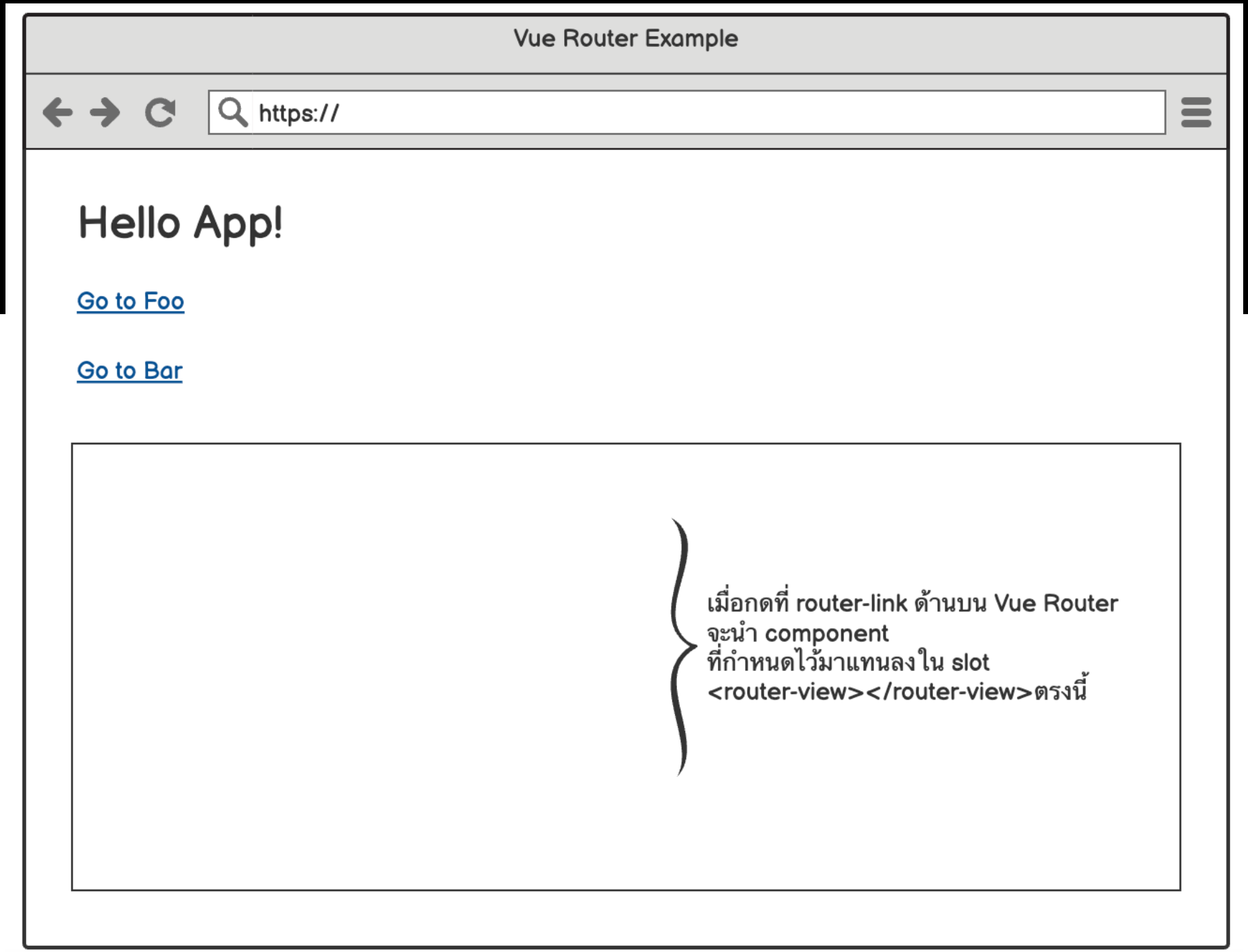
VUE ROUTER

Vue Router is the official router for Vue.js. It deeply integrates with Vue.js core to make building Single Page Applications with Vue.js a breeze.

SPA EXAMPLE

HTML

```html
<script src="https://unpkg.com/vue/dist/vue.js"></script>
<script src="https://unpkg.com/vue-router/dist/vue-router.js"></script>

<div id="app">
  <h1>Hello App!</h1>
  <p>
    <!-- use router-link component for navigation. -->
    <!-- specify the link by passing the `to` prop. -->
    <!-- `<router-link>` will be rendered as an `<a>` tag by default -->
    <router-link to="/foo">Go to Foo</router-link>
    <router-link to="/bar">Go to Bar</router-link>
  </p>
  <!-- route outlet -->
  <!-- component matched by the route will render here -->
  <router-view></router-view>
</div>
```


Vue Router Example
https://
Hello App!
Go to Foo
Go to Bar
เมื่อกดที่ router-link ด้านบน Vue Router
จะนำ component
ที่กำหนดไว้มาแทนลงใน slot
<router-view></router-view>ตรงนี้

JavaScript

```js
// 0. If using a module system (e.g. via vue-cli), import Vue and VueRouter
// and then call `Vue.use(VueRouter)`.

// 1. Define route components.
// These can be imported from other files
const Foo = { template: '<div>foo</div>' }
const Bar = { template: '<div>bar</div>' }

// 2. Define some routes
// Each route should map to a component. The "component" can
// either be an actual component constructor created via
// `Vue.extend()`, or just a component options object.
// We'll talk about nested routes later.
const routes = [
  { path: '/foo', component: Foo },
  { path: '/bar', component: Bar }
]

// 3. Create the router instance and pass the `routes` option
// You can pass in additional options here, but let's
// keep it simple for now.
const router = new VueRouter({
  routes // short for `routes: routes`
})

// 4. Create and mount the root instance.
// Make sure to inject the router with the router option to make the
// whole app router-aware.
const app = new Vue({
  router
}).$mount('#app')

// Now the app has started!
```

DYNAMIC ROUTE MATCHING – PATH PARAMS

In vue-router we can use a dynamic segment in the path to achieve that:

Now URLs
like /user/foo and /user/bar
will both map to the same route.

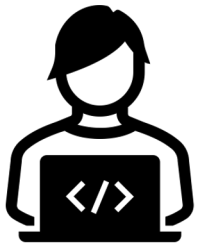
```
const User = {
  template: '<div>User</div>'
}

const router = new VueRouter({
  routes: [
    // dynamic segments start with a colon
    { path: '/user/:id', component: User }
  ]
})
```

$ROUTE.PARAMS

When a route is matched, the value of the dynamic segments will be exposed as this.$route.params in every component.

```
const User = {
  template: '<div>User {{ $route.params.id }}</div>'
}
```

$route.query:
An object that contains key/value pairs of the query string.
For example, for a path /foo?user=1, we get $route.query.user = 1.



Code Example Click Here!

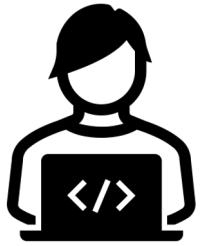
NESTED ROUTES

```
const router = new VueRouter({
  routes: [
    {
      path: '/user/:id',
      component: User,
      children: [
        {
          // UserProfile will be rendered inside User's <router-view>
          // when /user/:id/profile is matched
          path: 'profile',
          component: UserProfile
        },
        {
          // UserPosts will be rendered inside User's <router-view>
          // when /user/:id/posts is matched
          path: 'posts',
          component: UserPosts
        }
      ]
    }
  ]
})
```

Root path

Root component

Children



Code Example Click Here!

NAMED ROUTES

Sometimes it is more convenient to identify a route with a name, especially when linking to a route or performing navigations. You can give a route a name in the routes options while creating the Router instance:

```
const router = new VueRouter({
  routes: [
    {
      path: '/user/:userId',
      name: 'user',
      component: User
    }
  ]
})
```

```
<router-link :to="{ name: 'user', params: { userId: 123 }}">User</router-link>
```

PROGRAMMATIC NAVIGATION

Aside from using <router-link> to create anchor tags for declarative navigation, we can do this programmatically using the router's instance methods.

```
this.$router.push(location, onComplete?, onAbort?)
```

To navigate to a different URL, use this.$router.push. This method pushes a new entry into the history stack, so when the user clicks the browser back button they will be taken to the previous URL.

```
// literal string path
router.push('home')

// object
router.push({ path: 'home' })

// named route
router.push({ name: 'user', params: { userId: '123' } })

// with query, resulting in /register?plan=private
router.push({ path: 'register', query: { plan: 'private' } })
```

LET'S do the tutorial

https://github.com/it-web-pro/WEEK11-1-TUTORIAL

Authentication & Authorization

Today's topics

Authentication vs Authorization

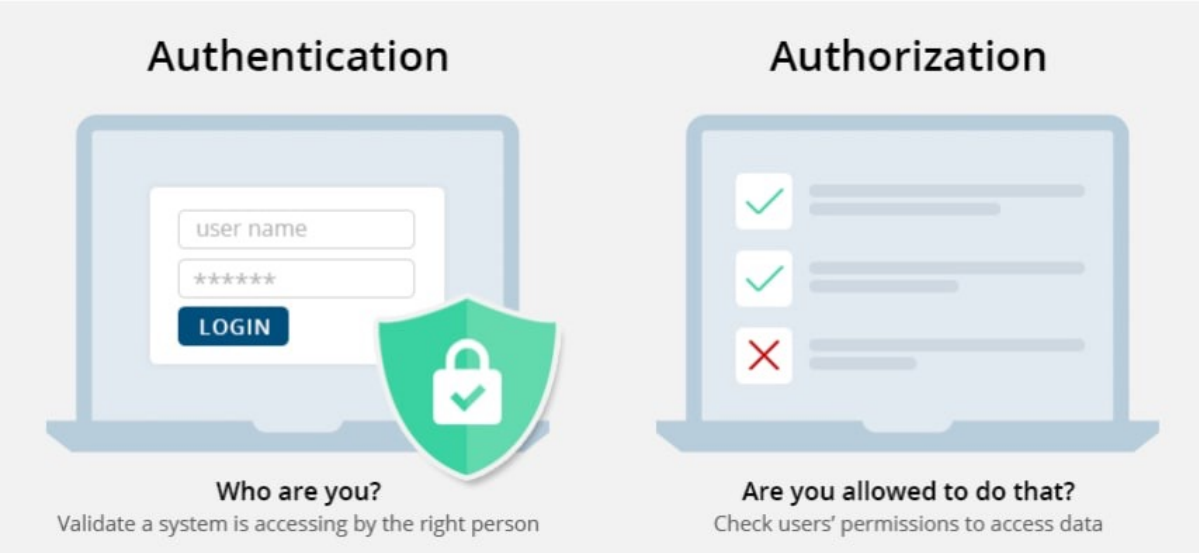
Express Middleware

Tutorial

- Logout/login
- isLoggedIn() middleware
- Vue router – navigation guard
- Roles & permissions - example

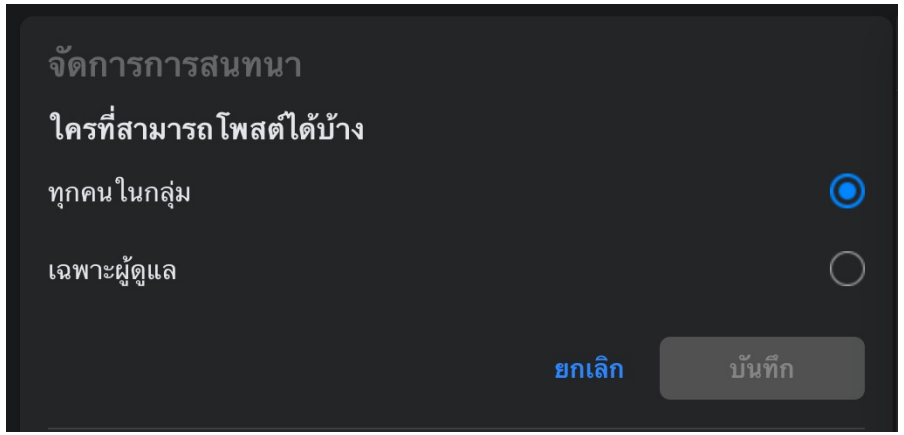
Authentication vs Authorization

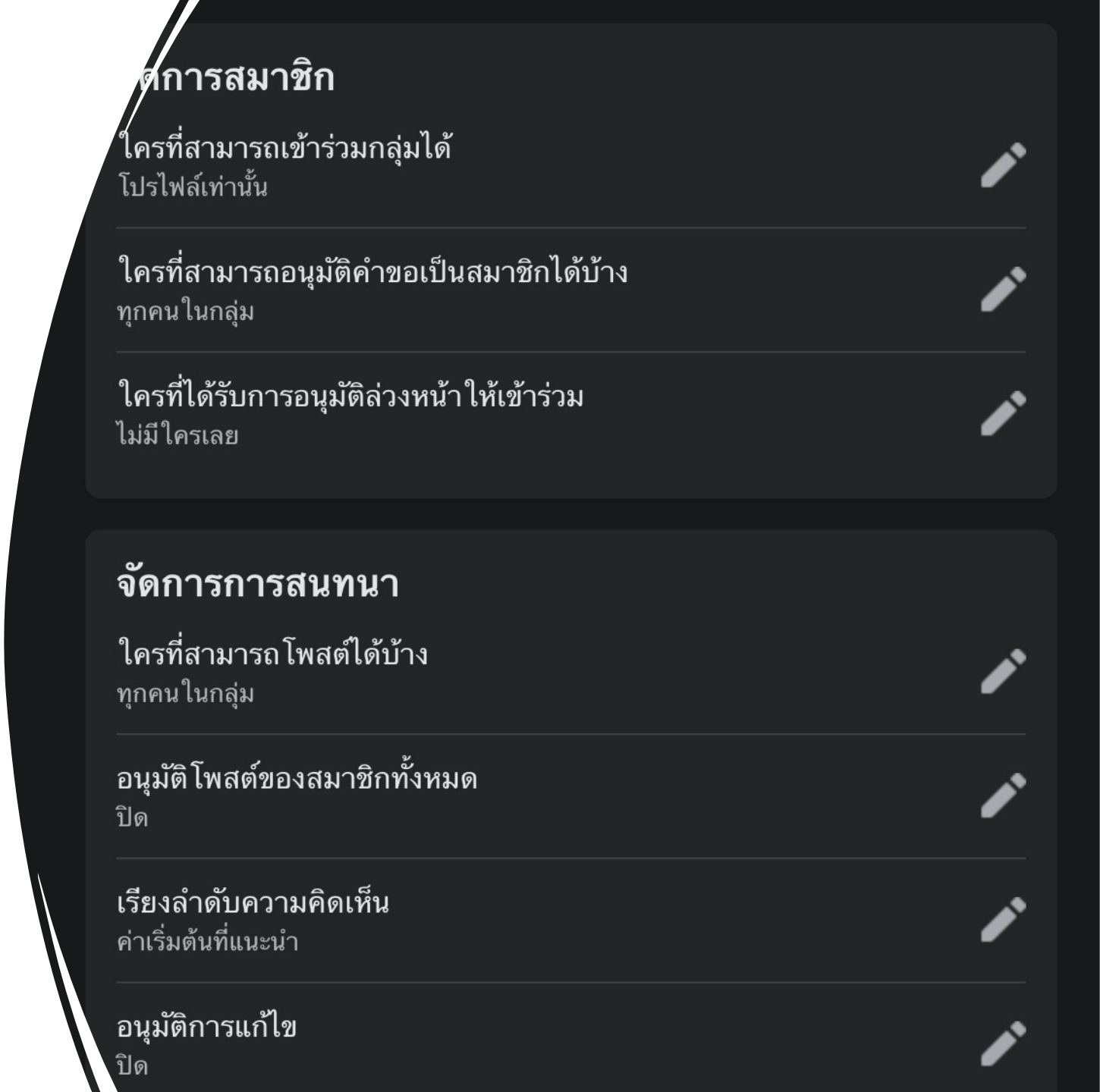
- **Authentication** confirms that users are who they say they are.
- **Authorization** Verifying user's permissions to access resources.



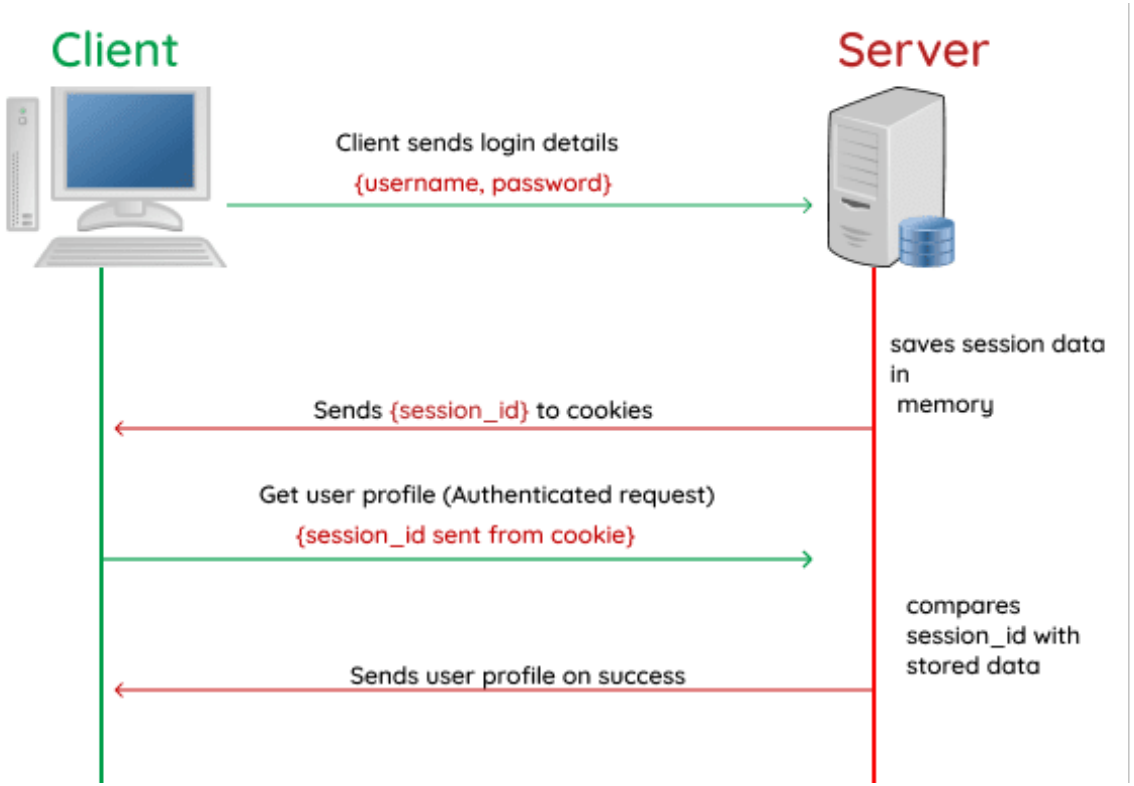
https://www.ssl2buy.com/wiki/authentication-vs-authorization-whats-the-difference

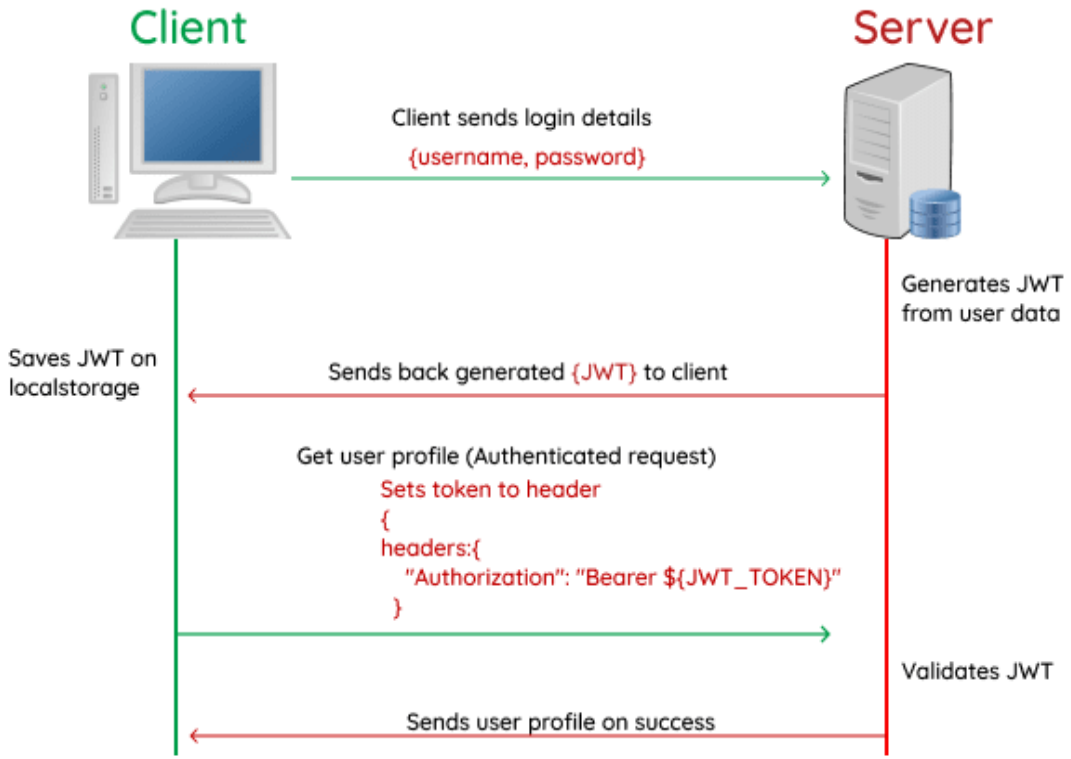
Facebook Group Authorization





Session Based vs Token Based Authentication





Session Based vs Token Based Authentication

Session Based

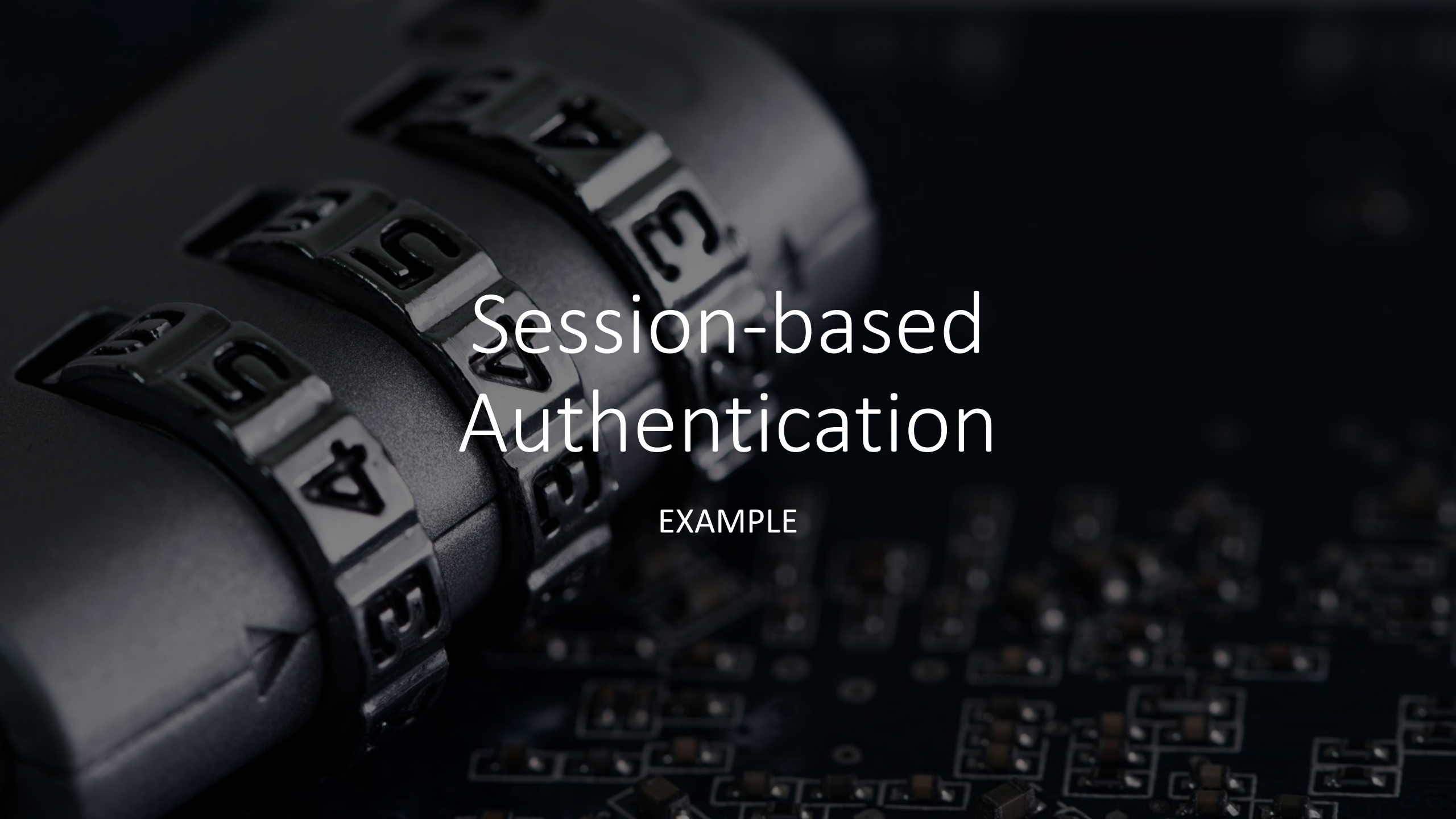
- User state is store in server (memory/database)
- Use Cookie to store session-id

Token Based

- User state is stored on client (localStorage)
- Attach token inside request header

NPM **express-session**
https://www.npmjs.com/package/express-session

NPM **express-jwt**
https://www.npmjs.com/package/express-jwt



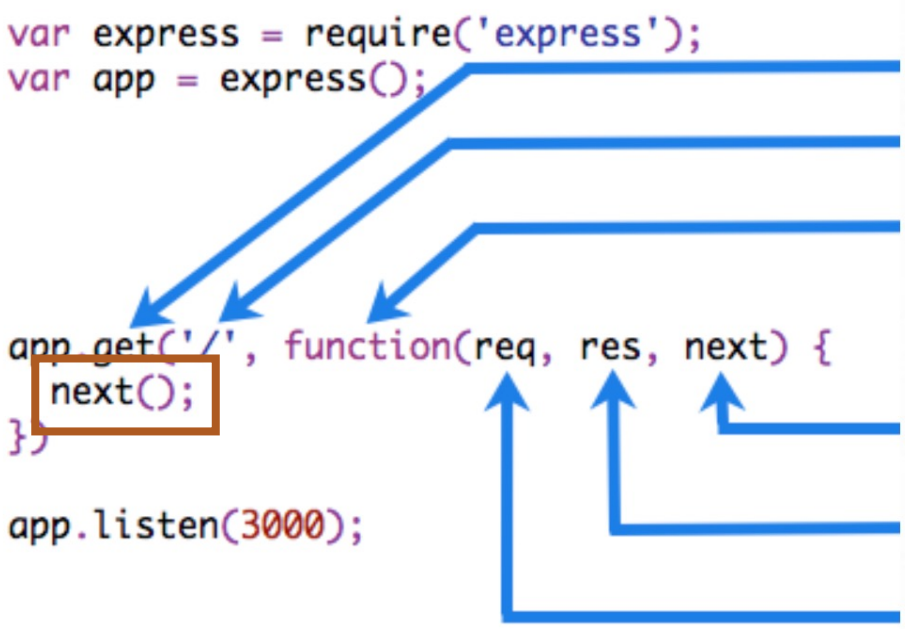
Session-based Authentication

EXAMPLE

Express Middleware

Middleware in Express JS

- **Middleware** functions are functions that have access to the request object (req), the response object (res), and the ***next*** function in the application's request-response cycle.
- The ***next*** function is a function in the Express router which, when invoked, executes the middleware succeeding the current middleware.
- **Middleware** functions can perform the following tasks:
  - Execute any code.
  - Make changes to the request and the response objects.
  - End the request-response cycle.
  - Call the next middleware in the stack.



HTTP method for which the middleware function applies.

Path (route) for which the middleware function applies.

The middleware function.

Callback argument to the middleware function, called "next" by convention.

HTTP response argument to the middleware function, called "res" by convention.

HTTP request argument to the middleware function, called "req" by convention.

Let’s see some examples

Express Middleware

Types of middleware

- An Express application can use the following types of middleware:
  - Application-level middleware
  - Router-level middleware
  - Error-handling middleware
  - Built-in middleware
  - Third-party middleware

Application-level middleware

- Bind application-level middleware to an instance of the app object by using the **app.use()** and **app.METHOD()** functions, where METHOD is the HTTP method of the request that the middleware function handles (such as GET, PUT, or POST) in lowercase.

```
const express = require('express')
const app = express()

app.use((req, res, next) => {
  console.log('Time:', Date.now())
  next()
})
```

```
app.get('/user/:id', (req, res, next) => {
  res.send('USER')
})
```

Router-level middleware

- Router-level middleware works in the same way as application-level middleware, except it is bound to an instance of express.Router().

- Let’s see some code

Error-handling middleware

Error-handling middleware always takes ***four*** arguments. You must provide four arguments to identify it as an error-handling middleware function. Even if you don't need to use the next object, you must specify it to maintain the signature. Otherwise, the next object will be interpreted as regular middleware and will fail to handle errors.

```
app.use((err, req, res, next) => {
  console.error(err.stack)
  res.status(500).send('Something broke!')
})
```

Error Handling

- ***Error Handling*** refers to how Express catches and processes errors that occur both synchronously and asynchronously. Express comes with a default error handler so you don't need to write your own to get started.
  - If synchronous code throws an error, then Express will catch and process it. For example:

```
app.get('/', (req, res) => {
  throw new Error('BROKEN') // Express will catch this on its own.
})
```

- For errors returned from asynchronous functions invoked by route handlers and middleware, you must pass them to the next() function, where Express will catch and process them. For example:

```
app.get('/', (req, res, next) => {
  fs.readFile('/file-does-not-exist', (err, data) => {
    if (err) {
      next(err) // Pass errors to Express.
    } else {
      res.send(data)
    }
  })
})
```

Built-in middleware

- Express has the following built-in middleware functions:
  - express.static serves static assets such as HTML files, images, and so on.
  - express.json parses incoming requests with JSON payloads. **NOTE: Available with Express 4.16.0+**
  - express.urlencoded parses incoming requests with URL-encoded payloads. **NOTE: Available with Express 4.16.0+**

```
app.use(express.static(path.join(__dirname, 'static')))
app.use(express.json()) // for parsing application/json
app.use(express.urlencoded({ extended: true })) // for parsing application/x-www-form-urlencoded
```

Third-party middleware

- Use third-party middleware to add functionality to Express apps.
- Install the Node.js module for the required functionality, then load it in your app at the application level or at the router level.

```
const express = require('express')
const app = express()
const cookieParser = require('cookie-parser')

// load the cookie-parsing middleware
app.use(cookieParser())
```


Simple Token Authentication
Let’s re-invent the wheel

Example (Simple Token Authentication)

Tutorial: https://github.com/it-web-pro/WEEK12-TUTORIAL-EXERCISE