**Table of Contents**

1. Introduction To Code JS
2. Working With Code JS Template

* The AIR

1. Understanding Data Binding & Storage With Code JS Frame
2. Code JS Functions

* ROM Components
* JS Builder
* JS Manipulator
* Events Handling
* Objects

1. Code JS App

**INTRODUCTION TO CODE JS**

**History**

In the year 2022, I upgraded my ability to solve web frontend problems and even wrote my first web game (https://spellingbee1.vercel .app). Then I made few other games, because making websites was a bit boring. Then making more apps and games no longer became a major objective. I was still unsatisfied and wanted to do more than just writing apps and games. I thought of what I could create as a programmer that would help/benefit other programmers, then I thought of creating a very little library with some code snippets. I started out crazily and the end result was. After a few months, I decided to remake it and make it more complex, consisting of functions and some extra features that lets you bind and call application data and even external variables whenever and wherever you need them. I also added some special code to help solve some special problems, especially in the area of design and functionality. Now, the former (nano-sized) library, now a (micro) framework is finally ready.

**What is Code JS?**

Code JS is a JavaScript Frontend Development Framework that focuses on HTML DOM and lets one manipulate it easier and faster. With Code JS, you can;

* Build & manipulate web contents.
* Bind application and Meta data.
* Wrap new styles for objects in a single JS function.
* Directly insert variables from an external JS file.
* Import your stylesheets and scripts without directly insert them in your index file.
* Create single page applications.
* Manipulate contents faster with its functions.
* Build your web apps & contents with JS easier, if you do not wish to write HTML directly in your markups & lots more...

**Requirements**

To use Code JS, you must have a basic knowledge and understanding in;

* **HTML5** - Standard Markup Language for structuring, and defining the layout of a webpage/web app.
* **CSS 3** – Used for styling a webpage/web app.
* **JavaScript** - A programming language, basically for programming the behaviour of a webpage/web app.

**More Info on Code JS**

Code JS is a framework that tests & proves how smart & capable people are in solving web frontend problems in design and functionality, while providing them with tools and functions to help them code faster.

**WORKING WITH CODE JS TEMPLATE**

**THE AIR**

The AIR is simply an acronym for:

**app.js** – This is the main script which will hold every JavaScript code for developing the web app, and will also be used to import/add other external JS scripts to the page. It can be done using any of the import functions in Code JS framework and will be talked about later. This file is and must always be in the JS folder.

**Index.html** – This is the main index/homepage file.

**root.css** – This is the main file that will contain all our CSS code which we’ll use to add/import other external stylesheets to our page. This importation can simply be done using our CSS @import rule. This file is and must always be in the CSS folder.

**Note:** Others are folders that hold files, pages etc.

**The A**

**(app.js)**

**UNDERSTANDING DATA INTERCHANGE & BINDING WITH CODE JS FRAME**

Code JS lets you bind your major web app data using the <**application />** element. This element binds our application data and contain the three main elements for our app development.

The following are attributes of the **<application />** element:

**name** –This holds the name of the application/website and sends it to the head element under the title element and application-name attribute in the meta tag/element.

**logo** – This stores your application/website logo and places it where it’s needed. As we continue the journey, you’ll understand fully.

**author –** Thisstores the name of the author/company that owns the app.

**theme –** This is used to specify a theme (major colour) for your application.

**mode –** This is used to specify a theme between (light or dark) for the entire app.

Sample:

<application name=”” author=”” logo=”” theme=”” mode=””></application>

There are three sections in the application element that contains our entire (visible) content. We have the;

**<data></data>** section -- For containing our meta data.

**<layout></layout>** section -- For containing our HTML code.

**<engine></engine>** section -- For containing our external scripts (Code JS.cjs & app.js).

As you can see, all meta tags are contained with the **<data>** element, and are binded in three lines. One for the **<meta charset>**, then the **<meta-name>** (instead of **<meta name=”” content=””>**, and the **<meta-http-equiv>** (instead of **<meta http-equiv>**), as shown below;

<data>

<meta charset="charset">

<meta-name name="content">

<meta-http-equiv name="content">

</data>

The **<layout />** element contains our markup as shown below;

<layout>

<header></header>

<nav></nav>

<main>

<section>

<article></article>

</section>

</main>

<aside></aside>

<footer></footer>

</layout>

And comments in HTML can also be written as;

</\_Comments goes here...>

You can also add **<footnote />** inside the **<footer></footer>** element and see the result.

Then we have the <engine></engine> element for containing our scripts, including adding from other sources.

# Note:

All existing files in the template have been completely linked/wired together, so do not link again to avoid unnecessary errors. Goto css/stylesheets.css to import css stylesheets or libraries, even without adding <link /> to the index.html, but in case, you can use the <link /> in <data> element, after the <meta-http-equiv /> element. And Goto js/scripts.js and add your external scripts sources to the $myScripts array and you're done. Only the functions(use imp() function to import a new script, or importAll to import an array of scripts) maybe with conditionals(if...else statements) or within functions.

You can insert JavaScript variable/constant data from an internal or external source using "compile" attribute, and <js> element. For example;

Create a new variable called data or greetings in app.js (or any external script)...

var data = "Hello World";

In the html file;

<button compile> data </button>

<h1>

<js> data </js> (My greetings)

</h1>

Or....

</\_Using "backtick">

<!-- Using "backtick" -->

<js>`

<h2> ${data} </h2>

(My greetings)

`</js>

Then Check the result! You can also check your console to see that new comment tag (</\_Comment>).

# Topic 3

# Building With JavaScript In Code JS

In Code JS, HTML elements are treated as widgets/objects and are given the class "widget" when created using Code JS JavaScript functions.

To create an element, using the widgets.construct() function.

# Remember:

All newly created HTML elements for your website or web app must go into the <layout> tag/element.

Open your app.js file in js folder or create a new one and add to $myScripts array in scripts.js file in the same folder.

# widgets.construct() render()

After opening your js file, on a new line, add;

widgets.construct("div");

// Use HTML function to write HTML into it.

html(widget, "<h1> Hello World! </h1>");

// Then place it in the <layout> using render() function, and use the selector function $(element|id|single-class);

render($('layout'), widget);

Preview your index.html and you'll see the result.

# widgets.constructMultiple() renderAll

To create a single element or widget, use widgets.construct(element), and to create multiple elements, use widgets.constructMultiple(elements, number of elements), for instance;

widgets.constructMultiple("div", 5);

// You can write same data to all at the same time, they're contained in the same variable "widget".

html(widget, "<h1> Hello World! </h1>"); // Or widget.innerHTML

// Using renderAll() function to render all multiple elements

renderAll($('layout'), widget);

# Moving widgets with render(target\_element, widget\_to\_move)

You can use render() to move an already existing element in the <layout> from its current parent element into another element. For instance;

index.html:

<layout>

<article id="one">

<h1 id="head1"> Hello World </h1>

</article>

<article id="two"></article>

</layout>

app.js:

render($('#two'), $('#head1'));

# renderBefore(target\_element, widget\_to\_render)

Use this to render or place a widget before another widget.

renderBefore($('#two'), $('#head1'));

# renderAfter(target\_element, widget\_to\_render)

Use this to render or place a widget after another widget.

renderAfter($('#two'), $('#head1'));

# renderFirstPlace(target\_element, widget\_to\_render)

Use this to render or place a widget inside another widget/element, as its first child.

renderFirstPlace($('#two'), $('#head1'));

# renderLastPlace(target\_element, widget\_to\_render)

Use this to render or place a widget inside another widget/element, as its last child.

renderLastPlace($('#two'), $('#head1'));

# swap(first\_element, second\_element)

Use this to swap or rearrange two already existing elements or widgets.

swap($('#one'), $('#two'));

# Working With Code JS Functions (DOM Manipulation)