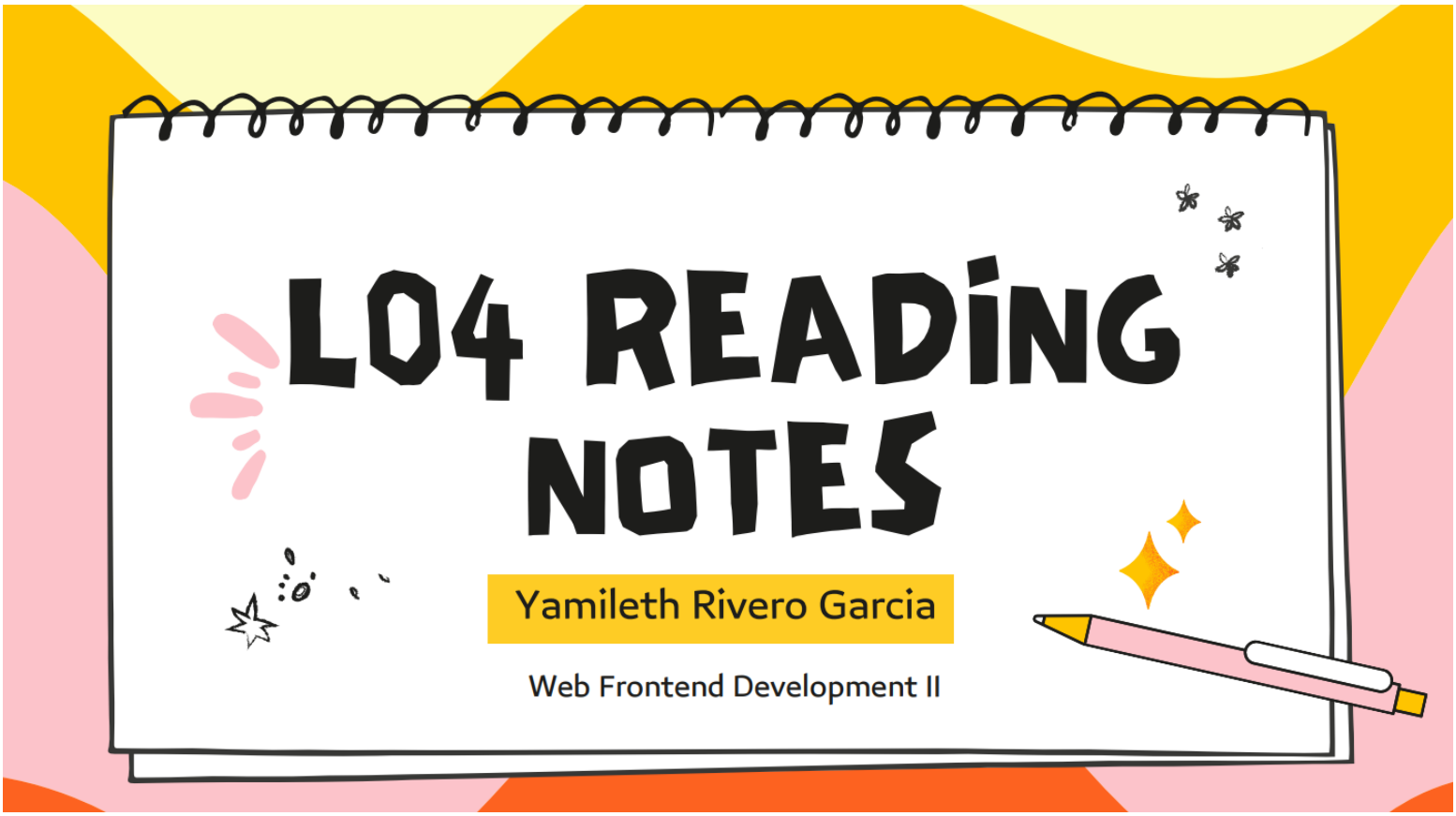



# 1.- Ch10: Testing and Debugging

Errors and bugs are a fact of life in programming.

## Errors

Errors are caused when something goes wrong in a program. They are usually caused by:

- **System error:** Problem with the system or external devices.
- **Programmer error:** Program contains incorrect syntax or faulty logic.
- **User error:** The user has entered data incorrectly

We must ensure they are minimized as much as possible and fixed promptly.

## Exceptions

An exception is an error that produces a **return value** that can then be used by the program to deal with the error.

**Example:** Trying to call a method that is nonexistent will result in a reference error that raises an exception.

```
unicorn();
<< ReferenceError: unicorn is not defined
```

### Stack Traces

An exception will also produce a stack trace.

This is a sequence of functions or method calls that lead to the point where the error occurred, which means that a stack trace will work backwards from the point at which the error occurred to identify the original function or method that started the sequence.

**Example:** We can use the stack trace to work backwards and see that this error was caused by invoking the **function one()** in the first place.

```
function three(){ unicorn(); }
function two(){ three(); }
function one(){ two(); }
one();

<< index.html:13 Uncaught ReferenceError: unicorn is not defined
    at three (index.html:13)
    at two (index.html:17)
    at one (index.html:21)
    at index.html:24`
```

## Warnings

> Can occur if there’s an error in the code that isn't enough to cause the program to crash.

NOTE: **The program will continue to run after a warning, it might sound good, but the program continues running incorrectly.**

**Example:** Assigning a value to a variable that’s undeclared

```
pi = 3.142;
<< JavaScript Warning: assignment to undeclared variable
```

NOTE: Sometimes you would need to open the console to see any warnings or exceptions.

## The Importance of Testing and Debugging

Sometimes instead of alerting a user to an error in a program, it just failed silently in the background.

I should ensure that the code they write fails loudly in development so any errors can be identified and fixed quickly.

## Strict Mode

> Strict mode considers coding practices that were previously accepted as just being 'poor style' as actual errors.

Increasing the chance of errors spot errors early on, rather than have them cause problems later.

Strict mode simply requires the following string to be added to the first line of a JavaScript file:

```
'use strict';
```

**Example:** If you try to assign a value to a variable that is undeclared in strict mode, you'll get an exception, instead of a warning

```
'use strict';

e = 2.718;
<< ReferenceError: e is not defined
```

```
function strictly(){
'use strict';
// function code goes here
```

You can even use strict mode on a per-function basis by adding the line inside a function. Strict mode will then only be applied to anything inside that function

The recommended way to invoke **strict mode** is to place all your code into a self-invoking function:

```
(function() {
    'use strict';

    // All your code would go inside this function

}());
```

Placing **'use strict'** at the beginning of a file will enforce strict mode on all the JavaScript in the file.

## Feature Detection

Some browsers will support certain features and others won't.

You can't always rely on users having the most up-to-date browser, either.

The recommended way to determine browser support for a feature is to use feature detection.

- Use an if statement to check whether an object or method exists before trying to call the method

Feature detection guarantees that the method is only called if it actually exists and **fails gracefully** if the method doesn't exist.

- The 'old-school' way of checking for browser support was known as **browser sniffing.**

## Debugging in the Browser

Debugging is the process of finding out where bugs occur in the code and then dealing with them.

You'll need to run through the program to see what's happening at different stages of its execution.

Is useful to create what are known as breakpoints.

- Breakpoints: Halt the progress of the code and allow us to view the value of different variables at that point in the program.

There are a number of options for debugging JavaScript code in the browser.

## 1) The Trusty Alert

The most basic form of debugging is to use the alert() method to show a dialog at certain points in the code.

Allows us to effectively put breakpoints in the code that let us check the value of variables at that point to see if they're what we expect them to be.

**Example:** The following code checks to see if a person's age is appropriate.

```
function amIOldEnough(age){
    if (age = 12) {
        alert(age);
        return 'No, sorry.';
    } else if (age < 18) {
        return 'Only if you are accompanied by an adult.';
    }
    else {
        return 'Yep, come on in!';
    }
}
```

The alert method inside the if block will allow us to see the value of the age variable at that point.

If you try the example above, you will find that there is a bug in the code:

```
amIOldEnough(21)
<< 'No, sorry.'
```

**Answer:** Instead of checking if the value of age is equal to 12, we have inadvertently assigned it the value of 12!

To check for equality, we should use === instead of = which assigns a value to a variable (even inside an if block)

In actual fact, we want to return the message 'No, sorry.' for all values of age that are **less** than 12, so we could update the code to the following:

```
function amIOldEnough(age){
    if (age < 12) {
        alert(age);
        return 'No, sorry.';
    } else if (age < 18) {
        return 'Only if you are accompanied by an adult.';
    }
    else {
        return 'Yep, come on in!';
    }
}
```

Try this again and it works as expected:

```
amIOldEnough(21)
<< 'Yep, come on in!'
```

## 2) Using the Console

Most modern JavaScript environments have a console object that provides a number of methods for logging information and debugging.

We've already seen and used the console.log() method.

This can be used to log the value of variables at different stages of the program, although it will not actually stop the execution of the program in the same way as alert() does.

## 3) Debugging Tools

Most modern browsers also have a debugging tool that allows you to set breakpoints in your code that will pause it at certain points.

NOTE: Here are the links to the debugger documentation for each of the major browsers:

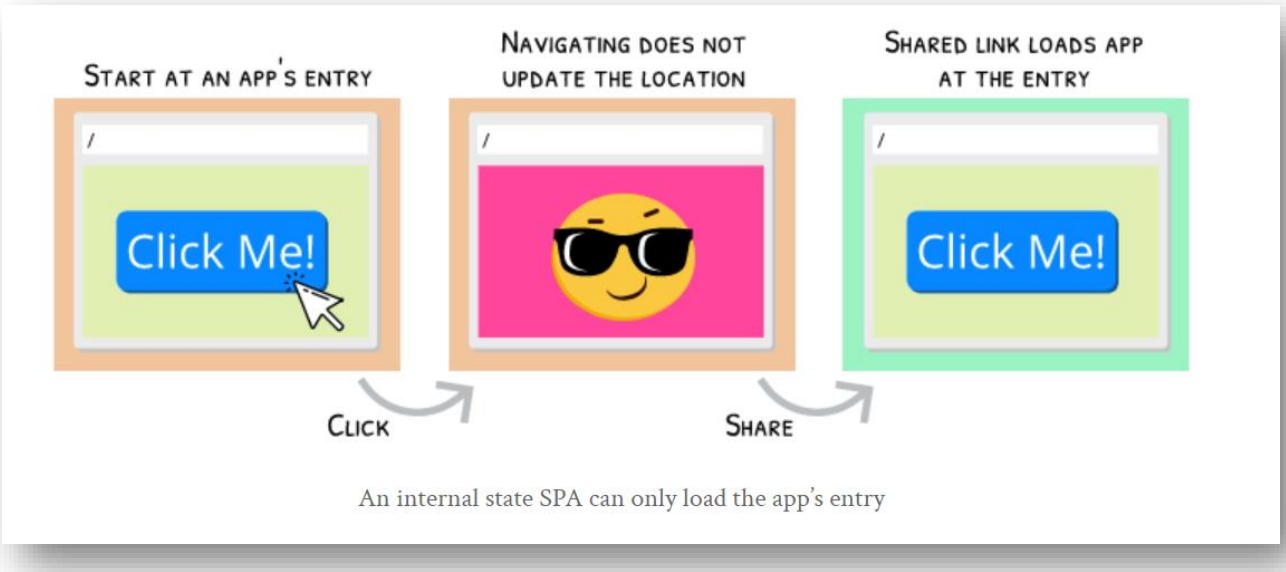
- FIREFOX: https://firefox-source-docs.mozilla.org/devtools-user/debugger/index.html
- EDGE: https://learn.microsoft.com/en-us/archive/microsoft-edge/legacy/developer/
- CHROME: https://developer.chrome.com/docs/devtools/
- SAFARI: https://developer.apple.com/library/archive/documentation/AppleApplications/Conceptual/Safari_Developer_Guide/Debugger/Debugger.html




# 2.- How Single – Page Applications Work

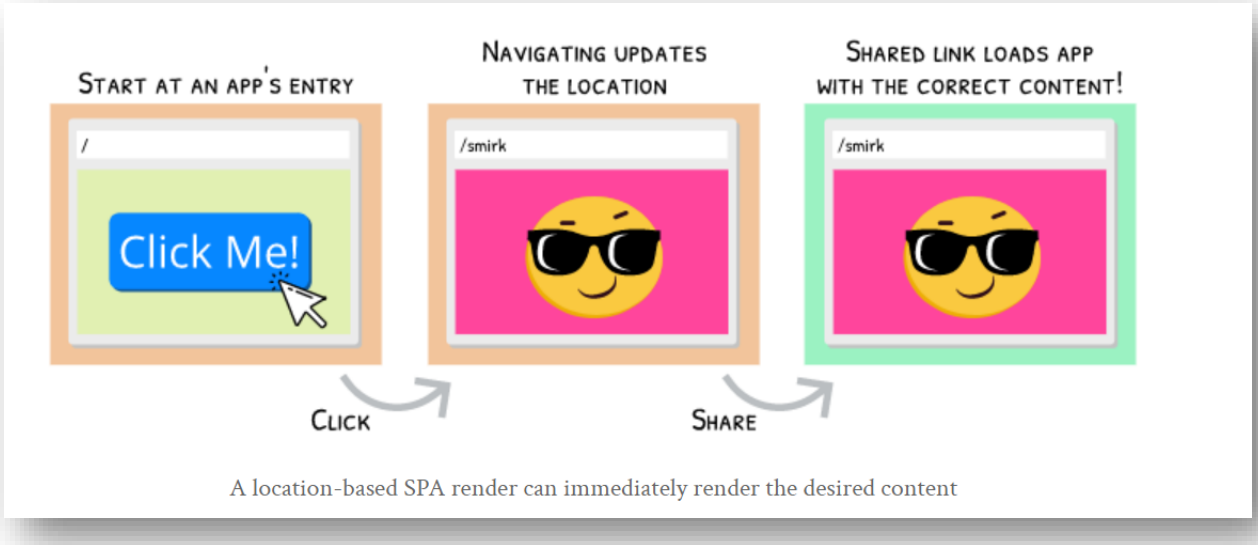
A single-page application (SPA) is a website that re-renders its content in response to navigation actions (e.g. clicking a link) without making a request to the server to fetch new HTML.

- ♥ Internal state SPAs are limited because there is only one "entry".
  - A **single entry** means that you always start at the root when you enter the application.
  - If you want to share some content, the other person loading the app will start at the **root**, so you'll have to explain how to get to desired content.




An internal state SPA can only load the app's entry

- ♥ Location-based SPAs are better.
    - You can share a link and be confident that anyone opening that link will see the same thing as you because the location is always updating as you navigate.




A location-based SPA render can immediately render the desired content

## Location Primer

- ♥ While the URL in the address bar is what users see and interact with, SPAs use window.location.
- ♥ Only three of the location object's properties are important for an SPA: pathname, hash, and search (commonly called a query string).

## Route Matching

- ♥ Single-page application generally rely on a router.
- ♥ Routers are made up of **routes**, which describe the location that they should match

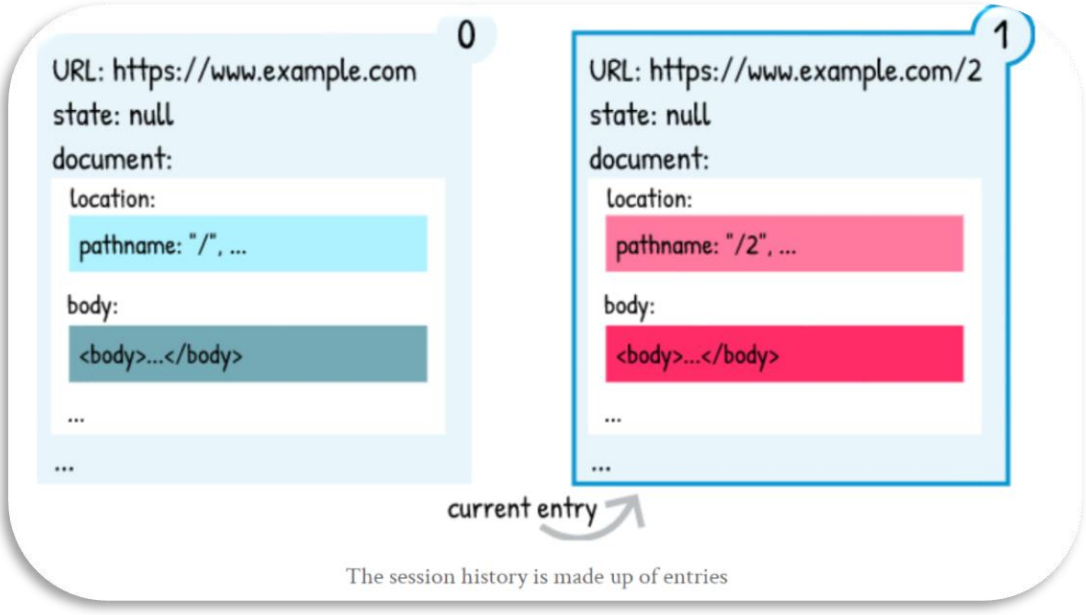
These can be static (/about) or dynamic (/album/:id, where the value of :id can be any number of possibilities) paths.

## How Browsers Handle Locations

Each browser tab has a "browsing context".

The browsing context maintains a "session history", which is essentially an array of location entries.

The browsing context also keeps track of which entry is currently active:




The session history is made up of entries

# 3.- The Problem With Single Page Apps

Single-page apps (or SPAs as they're sometimes called) serve all of the code for an entire multi-UI app from a single **index.html** file.

## Then…what is the issue with them?

You end up recreating with JavaScript a lot of the features the browser gives you out-of-the-box!

Which means:

- More code to maintain
- More complexity to manage
- More things to break
- Makes the whole app more fragile
- Makes the whole app bug-prone

## Are there alternatives then?

Of course!

- Build a multi-page app.
- Users can have a settings page
- Include a unique selector in the markup that describes the current view

### Why making this is better than SPAs?

Because…

- ♥ Support for the browser's forward and backward buttons are baked in
- ♥ You don't need to intercept clicks and determine if the clicked link points to an internal link or an external one
- ♥ You don't need to use complex regex patterns or another library to parse the URL and determine which view or UI to render.
- ♥ Simpler and easier to implement

### Does making this have a disadvantage?

Maybe, using JavaScript routing results in **faster apps** because **you avoid a page reload**.

# 4.- HTML5 Template Element

### What exactly is the <template> element?

It provides an easy way to define a **reusable** fragment of HTML that can be manipulated.