Software Requirements Specification

for

Restaurant Management System

A Project Report

Prepared by

Srimathy Kumar

Guided by

Junie

Project Objectives

I want to make this project for the purpose of making a restaurant management system easier. We know that managing a restaurant in old tradition is very costly and time consuming. To make the overall system efficient we want to make this management system. Here we want to serve our customer getting very efficient management system which will provide all kind of management like as workforce management and inventory control, etc.

Features

- **On Place Features:** Add, delete, update food items, items price.
- **Online Features:** Place food order for both on place and home delivery.

Advantages

1. Decreased workload
2. Save time.
3. Customer can easily order food
4. Admin can add, delete, update food details

System Requirements

Hardware Requirements

• Processor	: Intel core i5 or i7.
• Hard Disk	: 10 GB.
• Ram	: 8Gb.
• Compact Disk	: 650 Mb.
• Input device	: Standard Keyboard and Mouse.
• Output device	: High Resolution Monitor.

Software Requirements

Operating System	: Windows Family, Linux, Mac.
Application	: Eclipse (JDK 19), Apache Tomcat, SQLYog
Data Bases	: MySQL 8
Front End	: HTML5, CSS3, Boostrap 5.2, JavaScript
Back End:	: Java, JSP, Servlet.

Software Description

1. Front-End

1.1. HTML

HTML (**H**yper**T**ext **M**arkup **L**anguage) is the code that is used to structure a web page and its content. For example, content could be structured within a set of paragraphs, a list of bulleted points, or using images and data tables. As the title suggests, this article will give you a basic understanding of HTML and its functions.

Wat is HTML?

HTML is a *markup language* that defines the structure of your content. HTML consists of a series of elements, which you use to enclose, or wrap, different parts of the content to make it appear a certain way, or act a certain way. The enclosing tags can make a word or image hyperlink to somewhere else, can italicize words, can make the font bigger or smaller, and so on. For example, take the following line of content:

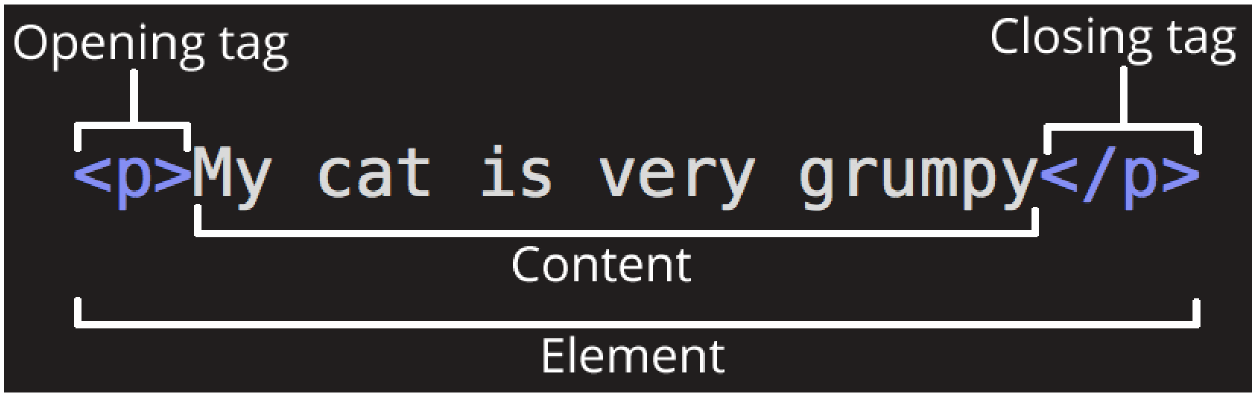
My cat is very grumpy

If we wanted the line to stand by itself, we could specify that it is a paragraph by enclosing it in paragraph tags:

<p>My cat is very grumpy</p>

Anatomy of an HTML element

Let's explore this paragraph element a bit further.

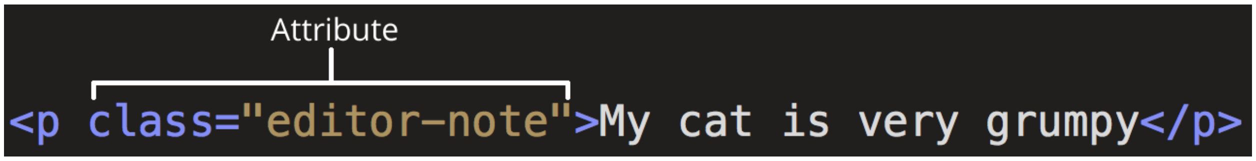


```
<p>My cat is very grumpy</p>
```

The main parts of our element are as follows:

1. **The opening tag:** This consists of the name of the element (in this case, p), wrapped in opening and closing angle brackets. This states where the element begins or starts to take effect — in this case where the paragraph begins.
2. **The closing tag:** This is the same as the opening tag, except that it includes a *forward slash* before the element name. This states where the element ends — in this case where the paragraph ends. Failing to add a closing tag is one of the standard beginner errors and can lead to strange results.
3. **The content:** This is the content of the element, which in this case, is just text.
4. **The element:** The opening tag, the closing tag, and the content together comprise the element.

Elements can also have attributes that look like the following:



```
<p class="editor-note">My cat is very grumpy</p>
```

Attributes contain extra information about the element that you don't want to appear in the actual content. Here, class is the attribute *name* and editor-note is the attribute *value*. The class attribute allows you to give the element a non-unique identifier that can be used to target it (and any other elements with the same class value) with style information and other things. Some attributes have no value, such as required.

Attributes that set a value always have:

1. A space between it and the element name (or the previous attribute, if the element already has one or more attributes).
2. The attribute name followed by an equal sign.
3. The attribute value wrapped by opening and closing quotation marks.

Nesting elements

You can put elements inside other elements too — this is called **nesting**. If we wanted to state that our cat is **very** grumpy, we could wrap the word "very" in a <strong> element, which means that the word is to be strongly emphasized:

```
<p>My cat is <strong>very</strong> grumpy.</p>
```

The elements have to open and close correctly so that they are clearly inside or outside one another. If they overlap as shown above, then your web browser will try to make the best guess at what you were trying to say, which can lead to unexpected results. So don't do it!

Void elements

Some elements have no content and are called void elements. Take the <img> element that we already have in our HTML page:

```
<img src="images/firefox-icon.png" alt="My test image" />
```

This contains two attributes, but there is no closing </img> tag and no inner content. This is because an image element doesn't wrap content to affect it. Its purpose is to embed an image in the HTML page in the place it appears.

Anatomy of an HTML document

That wraps up the basics of individual HTML elements, but they aren't handy on their own. Now we'll look at how individual elements are combined to form an entire HTML page.

```
<!DOCTYPE html>
<html lang="en-US">
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width" />
    <title>My test page</title>
  </head>
  <body>
    <img src="images/firefox-icon.png" alt="My test image" />
  </body>
</html>
```

Here, we have the following:

- <!DOCTYPE html> — doctype. It is a required preamble. In the mists of time, when HTML was young (around 1991/92), doctypes were meant to act as links to a set of rules that the HTML page had to follow to be considered good HTML, which could mean automatic error checking and other useful things. However, these days, they don't do much and are basically just needed to make sure your document behaves correctly. That's all you need to know for now.

- <html></html> — the <html> element. This element wraps all the content on the entire page and is sometimes known as the root element. It also includes the lang attribute, setting the primary language of the document.
- <head></head> — the <head> element. This element acts as a container for all the stuff you want to include on the HTML page that *isn't* the content you are showing to your page's viewers. This includes things like keywords and a page description that you want to appear in search results, CSS to style our content, character set declarations, and more.
- <meta charset="utf-8"> — This element sets the character set your document should use to UTF-8 which includes most characters from the vast majority of written languages. Essentially, it can now handle any textual content you might put on it. There is no reason not to set this, and it can help avoid some problems later on.
- <meta name = "viewport" content = "width = device-width"> —This viewport element ensures the page renders at the width of viewport, preventing mobile browsers from rendering pages wider than the viewport and then shrinking them down.
- <title></title> — the <title> element. This sets the title of your page, which is the title that appears in the browser tab the page is loaded in. It is also used to describe the page when you bookmark/favorite it.
- <body></body> — the <body> element. This contains *all* the content that you want to show to web users when they visit your page, whether that's text, images, videos, games, playable audio tracks, or whatever else.

Images

Let's turn our attention to the <img> element again:

```
<img src="images/firefox-icon.png" alt="My test image" />
```

We have also included an alt (alternative) attribute. In the alt attribute, you specify descriptive text for users who cannot see the image, possibly because of the following reasons:

1. They are visually impaired. Users with significant visual impairments often use tools called screen readers to read out the alt text to them.
2. Something has gone wrong causing the image not to display. For example, try deliberately changing the path inside your src attribute to make it incorrect. If you save and reload the page, you should see something like this in place of the image:

Marking up text

Headings

Heading elements allow you to specify that certain parts of your content are headings — or subheadings. In the same way that a book has the main title, chapter titles, and subtitles, an HTML document can too. HTML contains 6 heading levels, <h1> - <h6>, although you'll commonly only use 3 to 4 at most:

4 Heading levels:

```
<h1>My main title</h1>
<h2>My top level heading</h2>
<h3>My subheading</h3>
<h4>My sub-subheading</h4>
```

Paragraphs

As explained above, <p> elements are for containing paragraphs of text; you'll use these frequently when marking up regular text content:

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

Lists

A lot of the web's content is lists and HTML has special elements for these. Marking up lists always consists of at least 2 elements. The most common list types are ordered and unordered lists:

1. **Unordered lists** are for lists where the order of the items doesn't matter, such as a shopping list. These are wrapped in a <ul> element.
2. **Ordered lists** are for lists where the order of the items does matter, such as a recipe. These are wrapped in an <ol> element.

Each item inside the lists is put inside an <li> (list item) element.

For example, if we wanted to turn the part of the following paragraph fragment into a list

```
<p>
  At Mozilla, we're a global community of technologists, thinkers, and builders
  working together…
</p>

<p>At Mozilla, we're a global community of</p>

<ul>
  <li>technologists</li>
  <li>thinkers</li>
  <li>builders</li>
</ul>

<p>working together…</p>
```

Links

Links are very important — they are what makes the web a web! To add a link, we need to use a simple element <a> — "a" being the short form for "anchor". To make text within your paragraph into a link, follow these steps:

1. Choose some text. We chose the text "Mozilla Manifesto".
2. Wrap the text in an <a> element, as shown below:
3. <a>Mozilla Manifesto</a>
4. Give the <a> element an href attribute, as shown below:
5. <a href="">Mozilla Manifesto</a>
6. Fill in the value of this attribute with the web address that you want the link to:
7. <a href="https://www.mozilla.org/en-US/about/manifesto/">Mozilla Manifesto</a>

1.2. CSS

CSS (Cascading Style Sheets) is the code that styles web content. *CSS basics* walks through what you need to get started. We'll answer questions like: How do I make text red? How do I make content display at a certain location in the (webpage) layout? How do I decorate my webpage with background images and colors?

What is CSS?

Like HTML, CSS is not a programming language. It's not a markup language either**.** CSS is a style sheet language. CSS is what you use to selectively style HTML elements. For example, this CSS selects paragraph text, setting the color to red:

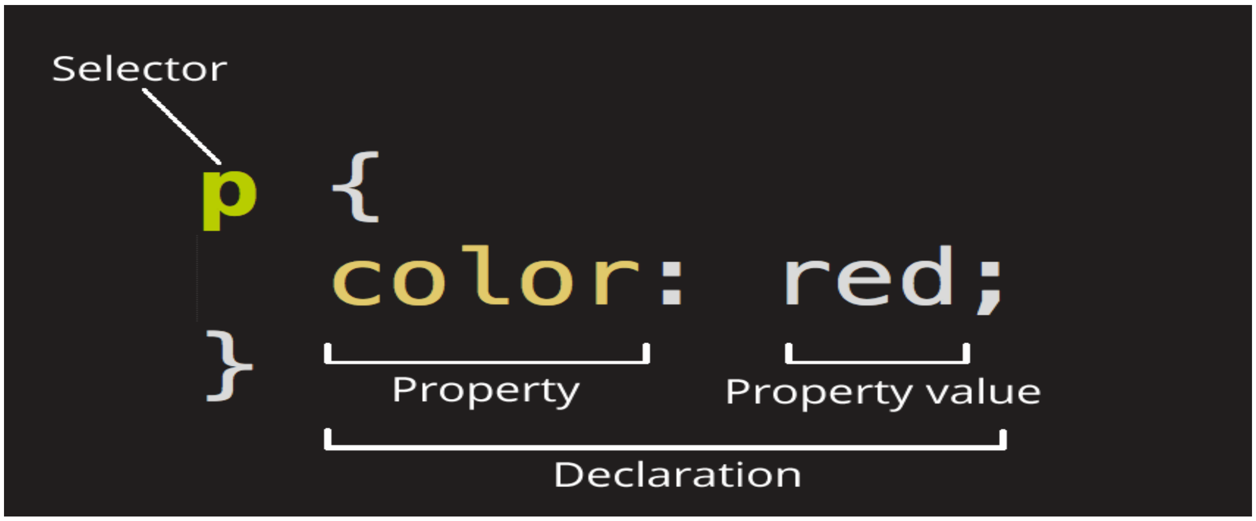
```
p {
  color: red;
}
```

To make the code work, we still need to apply this CSS (above) to your HTML document. Otherwise, the styling won't change the appearance of the HTML.

1. Open your index.html file. Paste the following line in the head (between the <head> and </head> tags):
2. <link href="styles/style.css" rel="stylesheet" />
3. Save index.html and load it in your browser. You should see something like this:

Anatomy of a CSS ruleset

Let's dissect the CSS code for red paragraph text to understand how it works:



The whole structure is called a **ruleset**. (The term *ruleset* is often referred to as just *rule*.) Note the names of the individual parts:

- Selector - This is the HTML element name at the start of the ruleset. It defines the element(s) to be styled (in this example, <p> elements). To style a different element, change the selector.
- Declaration - This is a single rule like color: red;. It specifies which of the element's **properties** you want to style.
- Properties - These are ways in which you can style an HTML element. In CSS, you choose which properties you want to affect in the rule.
- Property value - To the right of the property—after the colon—there is the **property value**. This chooses one out of many possible appearances for a given property.

Note the other important parts of the syntax:

- Apart from the selector, each ruleset must be wrapped in curly braces. ({})
- Within each declaration, you must use a colon (:) to separate the property from its value or values.
- Within each ruleset, you must use a semicolon (;) to separate each declaration from the next one.

To modify multiple property values in one ruleset, write them separated by semicolons, like this:

```
p {
  color: red;
  width: 500px;
  border: 1px solid black;
}
```

Selecting multiple elements

You can also select multiple elements and apply a single ruleset to all of them. Separate multiple selectors by commas. For example:

```
p,
li,
h1 {
  color: red;
}
```

Different types of selectors

Fonts and text

Now that we've explored some CSS fundamentals, let's improve the appearance of the example by adding more rules and information to the style.css file.

1. First, find the output from Google Fonts that you previously saved from What will your website look like?. Add the <link> element somewhere inside your index.html's head (anywhere between the <head> and </head> tags). It looks something like this:
2. <link href="https://fonts.googleapis.com/css?family=Open+Sans" rel="stylesheet" />
3. Next, delete the existing rule you have in your style.css file. It was a good test, but let's not continue with lots of red text.
4. Add the following lines (shown below), replacing the font-family assignment with your font-family selection from What will your website look like?. The property font-family refers to the font(s) you want to use for text. This rule defines a global base font and font size for the whole page. Since <html> is the parent element of the whole page, all elements inside it inherit the same font-size and font-family.
5. html {
6. font-size: 10px; /* px means "pixels": the base font size is now 10 pixels high */
7. font-family: "Open Sans", sans-serif; /* this should be the rest of the output you got from Google Fonts */
8. }
9. Now let's set font sizes for elements that will have text inside the HTML body (h1, <li>, and <p>). We'll also center the heading. Finally, let's expand the second ruleset (below) with settings for line height and letter spacing to make body content more readable.
10. h1 {
11. font-size: 60px;
12. text-align: center;
13. }
14. p,
15. li {
16. font-size: 16px;
17. line-height: 2;
18. letter-spacing: 1px;
19. }

Changing the page color

```
html {
  background-color: #00539f;
}
```

This rule sets a background color for the entire page.

Styling the body

```
body {
  width: 600px;
  margin: 0 auto;
  background-color: #ff9500;
  padding: 0 20px 20px 20px;
  border: 5px solid black;
}
```

There are several declarations for the <body> element:

- width: 600px; This forces the body to always be 600 pixels wide.
- margin: 0 auto; When you set two values on a property like margin or padding, the first value affects the element's top and bottom side (setting it to 0 in this case); the second value affects the left and right side. (Here, auto is a special value that divides the available horizontal space evenly between left and right). You can also use one, two, three, or four values, as documented in Margin Syntax.
- background-color: #FF9500; This sets the element's background color. This project uses a reddish orange for the body background color, as opposed to dark blue for the <html> element.
- padding: 0 20px 20px 20px; This sets four values for padding. The goal is to put some space around the content. In this example, there is no padding on the top of the body, and 20 pixels on the right, bottom and left. The values set top, right, bottom, left, in that order. As with margin, you can use one, two, three, or four values, as documented in Padding Syntax.
- border: 5px solid black; This sets values for the width, style and color of the border. In this case, it's a five-pixel–wide, solid black border, on all sides of the body.

Positioning and styling the main page title

```
h1 {
  margin: 0;
  padding: 20px 0;
  color: #00539f;
  text-shadow: 3px 3px 1px black;
}
```

There's a horrible gap at the top of the body. That happens because browsers apply default styling to the h1 element (among others). That might seem like a bad idea, but the intent is to provide basic readability for unstyled pages. To eliminate the gap, we overwrite the browser's default styling with the setting margin: 0;.

Following that, we set the heading text to be the same color as the HTML background color.

Finally, text-shadow applies a shadow to the text content of the element. Its four values are:

- The first pixel value sets the horizontal offset of the shadow from the text: how far it moves across.
- The second pixel value sets the vertical offset of the shadow from the text: how far it moves down.
- The third pixel value sets the blur radius of the shadow. A larger value produces a more fuzzy-looking shadow.
- The fourth value sets the base color of the shadow.

Centering the image

```
img {
  display: block;
  margin: 0 auto;
}
```

Next, we center the image to make it look better. We could use the margin: 0 auto trick again as we did for the body. But there are differences that require an additional setting to make the CSS work.

The <body> is a **block** element, meaning it takes up space on the page. The margin applied to a block element will be respected by other elements on the page. In contrast, images are **inline** elements, for the auto margin trick to work on this image, we must give it block-level behavior using display: block;.

1.3 Bootstrap

Today website development has become easy with the help of themes available. There are many options, but you should pick bootstrap themes. The themes made with bootstrap are feature-packed, mobile-friendly, and make your website easy to navigate for users.

However, do you know there is a new release in bootstrap v5 known as Bootstrap v5.2.0-beta1? Today, in this blog we are going to read about the features of bootstrap 5.2 and how it has enhanced the working and will help you in your project.

What's new in Bootstrap 5.2?

Bootstrap 5.2 is the new version of Bootstrap v5 having all the new features to make things easier for open-source developers. This is one of the longest releases among all the bootstrap releases. So, without any delay, let's move ahead and look at the features it holds!

What are the new features of Bootstrap 5.2?

The Bootstrap 5.2 is a longing release and is updating a new beta version, the latest bootstrap is v5.2, and even the stable is also released. So, what are the new features that will help developers make the website design an easy task? Here are the most decorated features of v 5.2:

- **Redesigned docs**

The Bootstrap docs are redesigned and made more navigable. Confused? Now, you can see the navbar and subnav bar on the page side of the page, it will show the every page link. In short, you can discover everything right there. They have also updated and refreshed the quick start guide to using Bootstrap via CDN.

If you see the navbar, now you can also choose versions from the new version picker in the upper right corner. The new doc is powered with the latest version of Algolia's DocSearch. So, you can see the most recent searches there as well.

- **Design tweaks**

With the change in docs, the design tweaks in the buttons and inputs were necessary. There's not much of a change but the button radius and box size have been adjusted and made aligned with the new version of the doc theme. They are more pleasing to look at!

- **Component CSS variables**

In this new bootstrap 5.2.0 release, they have added CSS variables to all the Components. It will only ease up things for the developers, whether experienced or newbies. Each component page now includes a reference guide to the relevant CSS variables.

Also, the values of each CSS variable are assigned via the Sass variable. So, now customization is also available in both types of variables be it CSS or Sass. Even some of the components of CSS variables are also customizable.

- **New _maps.scss**

The new bootstrap v5.2.0 has also fixed the issue of original map updates not being applied to the secondary map. This is not the ideal solution but will help developers while working with customized maps. The new version has added a new Sass file _maps.scss.

Sass variable and CSS variable have the same shortcoming, if the default variable or map has been used, it cannot be updated. To solve the error you have to override the defaults before they get used. Therefore, variable customization must follow @import "functions"; but before @import "variables"; and the rest of the import stack.

- **New helpers and utilities**

Apart from the new map Sass file, there's been an addition of new helpers and utilities in the new version of Bootstrap 5.2. These new additions will help in quick customization and building of components. Which are those new helpers and utilities?

- With the new .text-bg-{color}, you can now update the new background color in contrast with the foreground color.
- You can now have semibold fonts in font-weight utilities with .fw-semibold.
- There's an extension in border-radius utilities of .rounded-4 and .rounded-5.
- **Responsive offcanvas**

The Offcanvas component of Bootstrap now has responsive variations. It hides content across all viewports with the original .offcanvas class. The .offcanvas class can be changed to any .offcanvas-[sm|md|lg|xl|xxl] class to make it more responsive.

Dark Mode – An Upcoming Feature

We've all seen the update and are working on the dark mode in Bootstrap v5.2.0-beta 1. It is soon going to be rolled out in Bootstrap's next minor release. Apart from that, many changes have been worked upon to make the website theme designing easier and quicker for developers.

What can you expect in Bootstrap v5.3.0?

Yes, there's going to be a new version update soon in Bootstrap. There's no specific date given but all we know are about the few features that the team is working on and might be rolled out in the next release. So, which are they?

- Dark mode
- Attribute toggle plugin to toggle classes and attributes by writing HTML only.
- CSS Variables for forms
- Sticky headers for tables
- Mixins and functions for modifying the utility API
- An option for "always floating" floating forms. To know more about the release of Bootstrap v5.3.0, stay tuned to https://getbootstrap.com/.

Wrapping Up

So, overall Bootstrap 5.2.0 has new features that will help developers in building the theme quickly and effortlessly. Don't worry you won't have to do it still. Opt for pxdraft's Bootstrap templates and themes for your website for quick setup. All our themes and templates are developed with the latest version of bootstrap to get you the website with ultimate speed and mobile-friendliness.

1.4. JavaScript

JavaScript basics

JavaScript is a programming language that adds interactivity to your website. This happens in games, in the behavior of responses when buttons are pressed or with data entry on forms; with dynamic styling; with animation, etc. This article helps you get started with JavaScript and furthers your understanding of what is possible.

What is JavaScript?

JavaScript is a powerful programming language that can add interactivity to a website. It was invented by Brendan Eich.

JavaScript is versatile and beginner-friendly. With more experience, you'll be able to create games, animated 2D and 3D graphics, comprehensive database-driven apps, and much more!

JavaScript itself is relatively compact, yet very flexible. Developers have written a variety of tools on top of the core JavaScript language, unlocking a vast amount of functionality with minimum effort. These include:

- Browser Application Programming Interfaces (APIs) built into web browsers, providing functionality such as dynamically creating HTML and setting CSS styles; collecting and manipulating a video stream from a user's webcam, or generating 3D graphics and audio samples.
- Third-party APIs that allow developers to incorporate functionality in sites from other content providers, such as Twitter or Facebook.
- Third-party frameworks and libraries that you can apply to HTML to accelerate the work of building sites and applications.

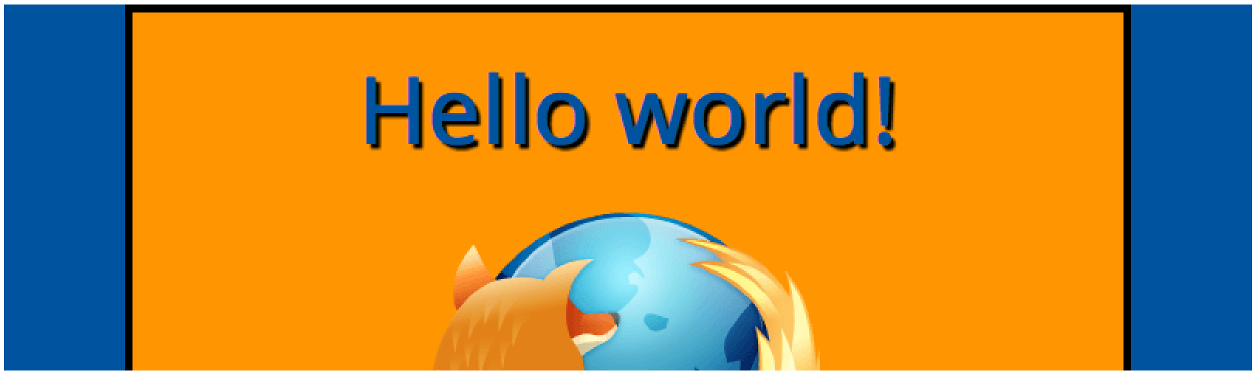
It's outside the scope of this article—as a light introduction to JavaScript—to present the details of how the core JavaScript language is different from the tools listed above. You can learn more in MDN's JavaScript learning area, as well as in other parts of MDN.

A "Hello world!" example

JavaScript is one of the most popular modern web technologies! As your JavaScript skills grow, your websites will enter a new dimension of power and creativity.

However, getting comfortable with JavaScript is more challenging than getting comfortable with HTML and CSS. You may have to start small, and progress gradually. To begin, let's examine how to add JavaScript to your page for creating a *Hello world!*

1. Go to your test site and create a new folder named scripts. Within the scripts folder, create a new text document called main.js, and save it.
2. In your index.html file, enter this code on a new line, just before the closing </body> tag:
3. <script src="scripts/main.js"></script>
4. This is doing the same job as the <link> element for CSS. It applies the JavaScript to the page, so it can have an effect on the HTML (along with the CSS, and anything else on the page).
5. Add this code to the main.js file:
6. const myHeading = document.querySelector("h1");
7. myHeading.textContent = "Hello world!";
8. Make sure the HTML and JavaScript files are saved. Then load index.html in your browser. You should see something like this:



Following that, the code set the value of the myHeading variable's textContent property (which represents the content of the heading) to *Hello world!*.

Language basics crash course

To give you a better understanding of how JavaScript works, let's explain some of the core features of the language. It's worth noting that these features are common to all programming languages. If you master these fundamentals, you have a head start on coding in other languages too!

Variables

Variables are containers that store values. You start by declaring a variable with the let keyword, followed by the name you give to the variable:

let myVariable;

A semicolon at the end of a line indicates where a statement ends. It is only required when you need to separate statements on a single line. However, some people believe it's good practice to have semicolons at the end of each statement. There are other rules for when you should and shouldn't use semicolons.

JavaScript is case sensitive. This means myVariable is not the same as myvariable. If you have problems in your code, check the case!

After declaring a variable, you can give it a value:

```
myVariable = "Bob";
```

Also, you can do both these operations on the same line:

```
let myVariable = "Bob";
```

You retrieve the value by calling the variable name:

```
myVariable;
```

After assigning a value to a variable, you can change it later in the code:

```
let myVariable = "Bob";
myVariable = "Steve";
```

Variables may hold values that have different data types:

- String
- Number
- Boolean
- Array
- Object

Comments

Comments are snippets of text that can be added along with code. The browser ignores text marked as comments. You can write comments in JavaScript just as you can in CSS:

```
/*
Everything in between is a comment.
*/
```

If your comment contains no line breaks, it's an option to put it behind two slashes like this:

```
// This is a comment
```

Operators

An operator is a mathematical symbol that produces a result based on two values (or variables). In the following table, you can see some of the simplest operators, along with some examples to try in the JavaScript console.

Mixing data types can lead to some strange results when performing calculations. Be careful that you are referring to your variables correctly, and getting the results you expect. For example, enter '35' + '25' into your console. Why don't you get the result you expected? Because the quote marks turn the numbers into strings, so you've ended up concatenating strings rather than adding numbers. If you enter 35 + 25 you'll get the total of the two numbers.

Conditionals

Conditionals are code structures used to test if an expression returns true or not. A very common form of conditionals is the if...else statement. For example:

```
let iceCream = "chocolate";
if (iceCream === "chocolate") {
  alert("Yay, I love chocolate ice cream!");
} else {
  alert("Awwww, but chocolate is my favorite…");
}
```

The expression inside the if () is the test. This uses the strict equality operator (as described above) to compare the variable iceCream with the string chocolate to see if the two are equal. If this comparison returns true, the first block of code runs. If the comparison is not true, the second block of code—after the else statement—runs instead.

Functions

Functions are a way of packaging functionality that you wish to reuse. It's possible to define a body of code as a function that executes when you call the function name in your code. This is a good alternative to repeatedly writing the same code. You have already seen some uses of functions. For example:

```
let myVariable = document.querySelector("h1");
```

```
alert("hello!");
```

These functions, document.querySelector and alert, are built into the browser.

If you see something which looks like a variable name, but it's followed by parentheses— () —it is likely a function. Functions often take arguments: bits of data they need to do their job. Arguments go inside the parentheses, separated by commas if there is more than one argument.

For example, the alert() function makes a pop-up box appear inside the browser window, but we need to give it a string as an argument to tell the function what message to display.

You can also define your own functions. In the next example, we create a simple function which takes two numbers as arguments and multiplies them:

```
function multiply(num1, num2) {
  let result = num1 * num2;
  return result;
}
```

Adding a personalized welcome message

Next, let's change the page title to a personalized welcome message when the user first visits the site. This welcome message will persist. Should the user leave the site and return later, we will save the message using the Web Storage API. We will also include an option to change the user, and therefore, the welcome message.

1. In index.html, add the following line just before the <script> element:
2. <button>Change user</button>
3. In main.js, place the following code at the bottom of the file, exactly as it is written. This takes references to the new button and the heading, storing each inside variables:
4. let myButton = document.querySelector("button");
5. let myHeading = document.querySelector("h1");
6. Add the following function to set the personalized greeting. This won't do anything yet, but this will change soon.
7. function setUserName() {
8. const myName = prompt("Please enter your name.");
9. localStorage.setItem("name", myName);
10. myHeading.textContent = `Mozilla is cool, ${myName}`;
11. }

    The setUserName() function contains a prompt() function, which displays a dialog box, similar to alert(). This prompt() function does more than alert(), asking the user to enter data, and storing it in a variable after the user clicks *OK.* In this case, we are asking the user to enter a name. Next, the code calls on an API localStorage, which allows us to store data in the browser and retrieve it later. We use localStorage's setItem() function to create and store a data item called 'name', setting its value to the myName variable which contains the user's entry for the name. Finally, we set the textContent of the heading to a string, plus the user's newly stored name.

12. Add the following condition block. We could call this initialization code, as it structures the app when it first loads.
13. if (!localStorage.getItem("name")) {
14.   setUserName();
15. } else {
16.   const storedName = localStorage.getItem("name");
17.   myHeading.textContent = `Mozilla is cool, ${storedName}`;
18. }

    This first line of this block uses the negation operator (logical NOT, represented by the !) to check whether the name data exists. If not, the setUserName() function runs to create it. If it exists (that is, the user set a user name during a previous visit), we retrieve the stored name using getItem() and set the textContent of the heading to a string, plus the user's name, as we did inside setUserName().

19. Put this onclick event handler (below) on the button. When clicked, setUserName() runs. This allows the user to enter a different name by pressing the button.
20. myButton.onclick = () => {
21.   setUserName();
22. };

A user name of null?

When you run the example and get the dialog box that prompts you to enter your user name, try pressing the *Cancel* button. You should end up with a title that reads *Mozilla is cool, null*. This happens because—when you cancel the prompt—the value is set as null. *Null* is a special value in JavaScript that refers to the absence of a value.

Also, try clicking *OK* without entering a name. You should end up with a title that reads *Mozilla is cool,* for fairly obvious reasons.

To avoid these problems, you could check that the user hasn't entered a blank name. Update your setUserName() function to this:

```
function setUserName() {
  const myName = prompt("Please enter your name.");
  if (!myName) {
    setUserName();
  } else {
    localStorage.setItem("name", myName);
    myHeading.textContent = `Mozilla is cool, ${myName}`;
  }
}
```

If myName has no value, run setUserName() again from the start. If it does have a value (if the above statement is not true), then store the value in localStorage and set it as the heading's text.

2. Back-End

2.1. Java Technology

Java technology is both a programming language and a Platform.

The Java Programming Language

The Java programming language is a high-level language that can be characterized by all of the following buzzwords:

- Simple
- Architecture neutral
- Object oriented
- Portable
- Distributed
- High performance
- Interpreted
- Multithreaded
- Robust
- Dynamic
- Secure

With most programming languages, you either compile or interpret a program so that you can run it on your computer. The Java programming language is unusual in that a program is both compiled and interpreted. With the compiler, first you translate a program into an intermediate language called Java byte codes—the platform-independent codes interpreted by the interpreter on the Java platform. The interpreter parses and runs each Java byte code instruction on the computer. Compilation happens just once; interpretation occurs each time the program is executed.

The Java Platform

A platform is the hardware or software environment in which a program runs. The Java platform differs from most other platforms in that it's a software-only platform that runs on top of other hardware-based platforms.

The Java platform has two components:

- The Java Virtual Machine (Java VM)
- The Java Application Programming Interface (Java API)

You've already been introduced to the Java VM. It's the base for the Java platform and is ported onto various hardware-based platforms.

The Java API is a large collection of ready-made software components that provide many useful capabilities, such as graphical user interface (GUI) widgets. The Java API is grouped into libraries of related classes and interfaces; these libraries are known as packages

Native code is code that after you compile it, the compiled code runs on a specific hardware platform. As a platform-independent environment, the Java platform can be a bit slower than native code. However, smart compilers, well-tuned interpreters, and just-in-time byte code compilers can bring performance close to that of native code without threatening portability.

JDBC

In an effort to set an independent database standard API for Java, Sun Microsystems developed Java Database Connectivity, or JDBC. JDBC offers a generic SQL database access mechanism that provides a consistent interface to a variety of RDBMSs. This consistent interface is achieved through the use of "plug-in" database connectivity modules, or *drivers*. If a database vendor wishes to have JDBC support, he or she must provide the driver for each platform that the database and Java run on.

To gain a wider acceptance of JDBC, Sun based JDBC's framework on ODBC. As you discovered earlier in this chapter, ODBC has widespread support on a variety of platforms. Basing JDBC on ODBC will allow vendors to bring JDBC drivers to market much faster than developing a completely new connectivity solution.

JDBC was announced in March of 1996. It was released for a 90 day public review that ended June 8, 1996. Because of user input, the final JDBC v1.0 specification was released soon after.

The remainder of this section will cover enough information about JDBC for you to know what it is about and how to use it effectively. This is by no means a complete overview of JDBC. That would fill an entire book.

JDBC Goals

Few software packages are designed without goals in mind. JDBC is one that, because of its many goals, drove the development of the API. These goals, in conjunction with early reviewer feedback, have finalized the JDBC class library into a solid framework for building database applications in Java.

The goals that were set for JDBC are important. They will give you some insight as to why certain classes and functionalities behave the way they do. The eight design goals for JDBC are as follows:

1. SQL Level API - The designers felt that their main goal was to define a SQL interface for Java. Although not the lowest database interface level possible, it is at a low enough level for higher-level tools and APIs to be created. Conversely, it is at a high enough level for application programmers to use it confidently. Attaining this goal allows for future tool vendors to "generate" JDBC code and to hide many of JDBC's complexities from the end user.
2. SQL Conformance - SQL syntax varies as you move from database vendor to database vendor. In an effort to support a wide variety of vendors, JDBC will allow any query statement to be passed through it to the underlying database driver. This allows the connectivity module to handle non-standard functionality in a manner that is suitable for its users.
3. JDBC must be implemental on top of common database interfaces - The JDBC SQL API must "sit" on top of other common SQL level APIs. This goal allows JDBC to use existing ODBC level drivers by the use of a software interface. This interface would translate JDBC calls to ODBC and vice versa.
4. Provide a Java interface that is consistent with the rest of the Java system - Because of Java's acceptance in the user community thus far, the designers feel that they should not stray from the current design of the core Java system.
5. Keep it simple - This goal probably appears in all software design goal listings. JDBC is no exception. Sun felt that the design of JDBC should be very simple, allowing for only one method of completing a task per mechanism. Allowing duplicate functionality only serves to confuse the users of the API.
6. Use strong, static typing wherever possible - Strong typing allows for more error checking to be done at compile time; also, less errors appear at runtime.
7. Keep the common cases simple - Because more often than not, the usual SQL calls used by the programmer are simple SELECT's, INSERT's, DELETE's and UPDATE's, these queries should be simple to perform with JDBC. However, more complex SQL statements should also be possible.

2.3. MySQL

What is SQL?

SQL is the standard language for dealing with Relational Databases. SQL is used to insert, search, update, and delete database records.

Who Uses MySQL?

- Huge websites like Facebook, Twitter, Airbnb, Booking.com, Uber, GitHub, YouTube, etc.
- Content Management Systems like WordPress, Drupal, Joomla!, Contao, etc.
- A very large number of web developers around the world
-

Show Data On Your Web Site

To build a web site that shows data from a database, you will need:

- An RDBMS database program (like MySQL)
- A server-side scripting language, like PHP
- To use SQL to get the data you want
- To use HTML / CSS to style the page
-

How to Use SQL

The following SQL statement selects all the records in the "Customers" table:

SELECT * FROM Customers;

SQL keywords are NOT case sensitive: select is the same as SELECT

Semicolon after SQL Statements?

Some database systems require a semicolon at the end of each SQL statement.

Semicolon is the standard way to separate each SQL statement in database systems that allow more than one SQL statement to be executed in the same call to the server.

Some of The Most Important SQL Commands

- SELECT - extracts data from a database
- UPDATE - updates data in a database
- DELETE - deletes data from a database
- INSERT INTO - inserts new data into a database

- CREATE DATABASE - creates a new database
- ALTER DATABASE - modifies a database
- CREATE TABLE - creates a new table
- ALTER TABLE - modifies a table
- DROP TABLE - deletes a table
- CREATE INDEX - creates an index (search key)
- DROP INDEX - deletes an index

Modules

The main modules in the Restaurant Management System are as follows:

- Items (Food)
- Customer
- Admin

1. Items (Food)

- id
- itemName
- price

2. Customer

- Allow Customers to scroll through the menu and select the dishes he/she wants.
- Allow the Customers to edit the order any time before its prepared.
- Allow Customers to provide feedback regarding the food and overall service of the restaurant.

3. Admin

- Allow admin to perform CRUD (create, retrieve, update and delete) operations on Menu Items and Inventory.

Functionality of Modules

- Login – this module is used for admin and customer separately
- Logout – this module is also used for admin and customer separately
- Items – this module helps in selecting the food items
- Dashboard – Admin dashboard related all Food, Food details, Food listing
- Menu Management Module – adding new food details, editing the existing food, listing all the foods.

Functional Requirements

REQ-1: The system will show a list of cards (UI element) of dishes. Each card will have a picture of the dish. Below the dish it shows the price in Rupees per serving.

REQ-2: The system must show all available dishes to the Customer.

REQ-3: Unused for particular time, it will automatically entered into session expired, then need to sign in again.

REQ-4: Input will allow the user to click on the checkbox to select the order.

REQ-5: Incorrect password or username, entered into wrong username or password page.

REQ-6: Without sign in into Application, Menu is not available for users.

Nonfunctional Requirements

1. Performance Requirements

The system must be interactive, and the delays involved must be less. So, in every action, response of the system, there are no immediate delays. In case of scrolling through the menu there should be a delay of no more than 2 second before the next page of menu items is displayed otherwise our people's dining experience is affected. The order should be placed in pending orders and be visible to the head chef/chefs in less than 1 second to start the preparation. Cancel Order/ updates must be made with little delay to avoid delivery delay. Also, when connecting to the server the delay to make a successful connection should be less for effective real time communication.

2. Safety Requirements

The software is completely environmentally friendly and does not cause any safety violations. The menu will have a flexible font that can be zoomed so as to not over constrain the eyes.

3. Security Requirements

There is a need for a proper and encrypted login authentication for head chef and admin as employee sensitive information as well as inventory should be protected from hacking. Information transmission should be securely transmitted to Firebase without any changes in information to avoid disturbances in orders and billing.

4. Software Quality Attributes

1. Adaptability: There can be a change in the menu and information stored in the database about employees and inventory.

2. Availability: The system is up and running for most of the time and server is not down for more than a few minutes to avoid inconvenience of the customers.

3. Correctness: The bill generated by the application must be accurate and the orders placed should exactly be the same which the user has selected. Software Requirements Specification.

4. Flexibility: If need arises in the future, software can be modified to change the requirements.

5. Interoperability: The data is transferred from the customer's end to the kitchen and then head chef assigns orders to each chef. This way data is transferred from one part of the system to another.

6. Maintainability: Software can be easily repaired if a fault occurs.

7. Portability: Software can be easily installed on devices and would run smoothly according to the requirement.

8. Reliability: No matter how many orders are placed, system must give the correct results.

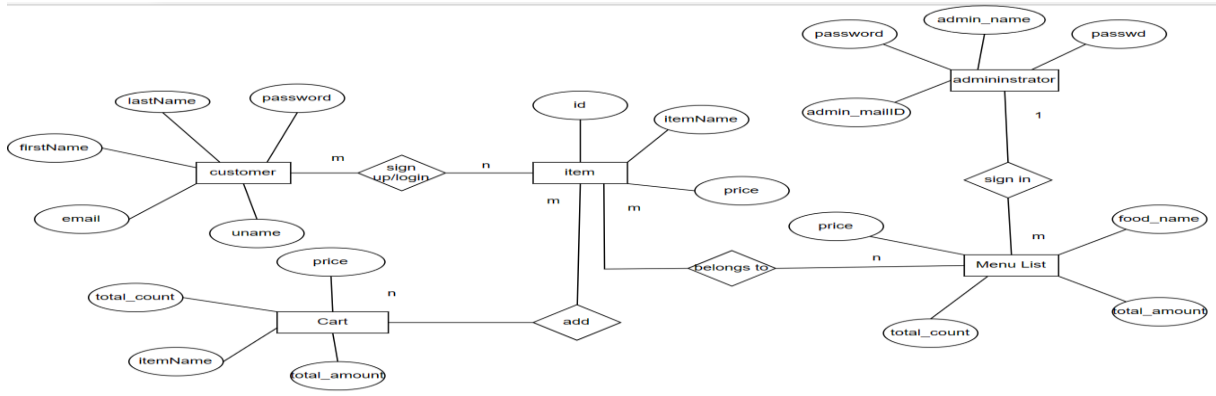
9. Reusability: Current version can be used in the future versions with more functionality added.

10. Robustness: Software must have checks to ensure that the items that are not available in the menu cannot be selected and the emails, phone numbers added are all valid.

11. Testability: All the requirements are fulfilled, response time is low, and all functions are working perfectly.

12. Usability: Interface of the software must be easy to use. It would not be complex since managers, chefs have a view, so interface should be simple.

E-R Diagram



Static pages with GUI

These static pages will be available in Restaurant Management System project.

- Home Page with good UI
- Home Page contain sign in and sign up for both Customer and Admin
- About us page has Restaurant and Contact Details

Testing

Process

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub-assemblies, assemblies and/or a finished product It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

Types of Testing

Unit Testing

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program input produces valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

Integration Testing

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

Functional Testing

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation and user manuals.

Functional testing is centered on the following items:

Valid Input	: identified classes of valid input must be accepted.
Invalid Input	: identified classes of invalid input must be rejected.
Functions	: identified functions must be exercised.
Output	: identified classes of application outputs must be exercised.
Systems/Procedures	: interfacing systems or procedures must be invoked.

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows; data fields, predefined processes, and successive processes must be considered for testing. Before functional testing is complete, additional tests are identified and the effective value of current tests is determined.

System Testing

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration-oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

White Box Testing

White Box Testing is a testing in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It is used to test areas that cannot be reached from a black box level.

Black Box Testing

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document. It is a testing in which the software under test is treated, as a black box .you cannot “see” into it. The test provides inputs and responds to outputs without considering how the software works.

Unit Testing

Unit testing is usually conducted as part of a combined code and unit test phase of the software lifecycle, although it is not uncommon for coding and unit testing to be conducted as two distinct phases.

Test Strategy and Approach

Field testing will be performed manually and functional tests will be written in detail.

Test objectives

- All field entries must work properly.
- Pages must be activated from the identified link.
- The entry screen, messages and responses must not be delayed.
- Features to be tested
- Verify that the entries are of the correct format
- No duplicate entries should be allowed
- All links should take the user to the correct page.

Integration Testing

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects.

The task of the integration test is to check that components or software applications, e.g. components in a software system or – one step up – software applications at the company level – interact without error.

Acceptance Testing

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

Test Results

All the test cases mentioned above passed successfully. No defects encountered.

System Implementation

System Implementation is the stage in the project where the theoretical design is turned into a working system. The most critical stage is achieving a successful system and in giving confidence on the new system for the user that it will work efficiently and effectively.

The existing system was long time process. The proposed system was developed using JSP and servlet. The existing system caused long time transmission process but the system developed now has a very good user-friendly tool, which has a menu-based interface, graphical interface for the end user.

After coding and testing, the project is to be installed on the necessary system. The executable file is to be created and loaded in the system. Again the code is tested in the installed system. Installing the developed code in system in the form of executable file is implementation.

Conclusion

A competent restaurant management system web project may make or break the success of your establishment. While there are numerous aspects to consider, it is worthwhile to spend time determining how the system should benefit of the company and what I hope to achieve from it.

Future Work

In future I would like solve all draw backs and I want to add some features like storing customer details with order details, uploading the images along with food item entry. Also I will try to add payment options and discount facilities.