**Introduction to Project**

This is a project report on “ONLINE MUSIC PLAYER”. During the making/developing of this project we explored new ideas and functionality behind the working of a notepad.

This project is designed by JAVASCRIPT. JavaScript is a client-side scripting language, which means the [source code](https://techterms.com/definition/sourcecode) is processed by the client's [web browser](https://techterms.com/definition/web_browser) rather than on the [web server](https://techterms.com/definition/web_server). This means JavaScript [functions](https://techterms.com/definition/function) can run after a webpage has loaded without communicating with the server. For example, a JavaScript function may check a web form before it is submitted to make sure all the required [fields](https://techterms.com/definition/field) have been filled out. The JavaScript code can produce an error message before any information is actually transmitted to the server. Our project **“**ONLINE MUSIC PLAYER**”** is not only site for Music lovers. This is a web based application which helps people to find and listen latest songs with different languages on internet. It is useful in the way that it makes an easier way to search music online.

The customer have to open website for any enquiry related to Music.

The admin module contains the access of admin on the application. The admin can change everything in the application. He have the ability to add, delete, update any information regarding the music.

**Technologies Used**

**What is HTML?**

HTML is a language for designing web pages.

* HTML stands for Hyper Text Markup Language
* HTML is not a programming language, it is a markup language
* A markup language is a set of markup tags
* **HTML Tags**

HTML markup tags are usually called HTML tags

* HTML tags are keywords surrounded by angle brackets like <html>
* HTML tags normally come in pairs like <b> and </b>
* The first tag in a pair is the start tag, the second tag is the end tag
* Start and end tags are also called opening tags and closing tags.
* **HTML Headings**

HTML headings are defined with the <h1> to <h6> tags.

|  |  |  |  |
| --- | --- | --- | --- |
| Example   |  | | --- | | <h1>This is a heading</h1> <h2>This is a heading</h2> <h3>This is a heading</h3> | |  | |  | |

* **HTML Paragraphs**

HTML paragraphs are defined with the <p> tag.

|  |  |
| --- | --- |
| Example   |  | | --- | | <p>This is a paragraph</p> <p>This is another paragraph</p> | |

* **HTML Links**

HTML links are defined with the <a> tag.

|  |  |
| --- | --- |
| Example   |  | | --- | | <a href="http://www.w3schools.com">This is a link</a> | |

* **HTML Images**

HTML images are defined with the <img> tag.

|  |  |
| --- | --- |
| Example   |  | | --- | | <img src="constr4.gif" width="144" height="50" /> | |

* **HTML Paragraphs**

Paragraphs are defined with the <p> tag.

|  |  |
| --- | --- |
| Example   |  | | --- | | <p>This is a paragraph</p> <p>This is another paragraph</p> | |

* **HTML Elements**

An HTML element is everything from the start tag to the end tag:

|  |  |  |
| --- | --- | --- |
| Start tag | Element content | End tag |
| <p> | This is a paragraph | </p> |
| <a href="default.htm" > | This is a link | </a> |

* **HTML Element Syntax**
* An HTML element starts with a start tag
* An HTML element ends with an end tag
* The element content is everything between the start and end tag
* Some HTML elements have empty content
* Some HTML elements have a missing end tag
* **HTML Attributes**
* HTML elements can have attributes
* Attributes provide additional information about the element
* Attributes are always specified in the start tag
* **HTML Style Examples**
* style="background-color:yellow"
* style="font-size:10px"
* style="font-family:Times"
* style="text-align:center"
* **The Image Tag and the Src Attribute**
* In HTML, images are defined with the <img> tag.  The <img> tag is empty, which means that it contains attributes only and it has no closing tag.
* To display an image on a page, you need to use the src attribute. Src stands for "source". The value of the src attribute is the URL of the image you want to display on your page.
* The syntax of defining an image:

|  |
| --- |
| <imgsrc="url" /> |

* **The Alt Attribute**

The alt attribute is used to define an "alternate text" for an image if image is not loaded.

|  |
| --- |
| <imgsrc="boat.gif" alt="Big Boat" /> |

The "alt" attribute tells the reader what he or she is missing on a page if the browser can't load images.

**Tables**

Tables are defined with the <table> tag. A table is divided into rows (with the <tr> tag), and each row is divided into data cells (with the <td> tag). The letters td stands for "table data," which is the content of a data cell. A data cell can contain text, images, lists, paragraphs, forms, horizontal rules, tables, etc.

|  |
| --- |
| <table border="1">  <tr>  <td>row 1, cell 1</td>  <td>row 1, cell 2</td>  </tr>  <tr>  <td>row 2, cell 1</td>  <td>row 2, cell 2</td>  </tr>  </table> |

Browser Preview:

|  |  |
| --- | --- |
| row 1, cell 1 | row 1, cell 2 |
| row 2, cell 1 | row 2, cell 2 |

* **Headings in a Table**

Headings in a table are defined with the <th> tag.

|  |
| --- |
| <table border="1">  <tr>  <th>Heading</th>  <th>Another Heading</th>  </tr>  <tr>  <td>row 1, cell 1</td>  <td>row 1, cell 2</td>  </tr>  <tr>  <td>row 2, cell 1</td>  <td>row 2, cell 2</td>  </tr>  </table> |

Browser Preview:

|  |  |
| --- | --- |
| **Heading** | **Another Heading** |
| row 1, cell 1 | row 1, cell 2 |
| row 2, cell 1 | row 2, cell 2 |

* **Forms**

A form is an area that can contain form elements.

Form elements are elements that allow the user to enter information (like text fields, textarea fields, drop-down menus, radio buttons, checkboxes, etc.) in a form.

A form is defined with the <form> tag.

|  |
| --- |
| <form>  <input>  <input>  </form> |

* **Input**

The most used form tag is the <input> tag. The type of input is specified with the type attribute. The most commonly used input types are explained below.

**Text Fields**

Text fields are used when you want the user to type letters, numbers, etc. in a form.

|  |
| --- |
| <form>  First name:  <input type="text" name="firstname">  <br>  Last name:  <input type="text" name="lastname">  </form> |

How it looks in a browser:

Top of Form

Firstname:  
Last name: 

* **Radio Buttons**

Radio Buttons are used when you want the user to select one of a limited number of choices.

|  |
| --- |
| <form>  <input type="radio" name="sex" value="male"> Male  <br>  <input type="radio" name="sex" value="female"> Female  </form> |

How it looks in a browser:

Top of Form

Male   
Female

Bottom of Form

Note that only one option can be chosen.

* **Checkboxes**

Checkboxes are used when you want the user to select one or more options of a limited number of choices.

|  |
| --- |
| <form>  I have a bike:  <input type="checkbox" name="vehicle" value="Bike">  <br>  I have a car:  <input type="checkbox" name="vehicle" value="Car">  <br>  I have an airplane:  <input type="checkbox" name="vehicle" value="Airplane">  </form> |

How it looks in a browser:

ComputerTop of Form

Computer:  
Laptop:  
Palmtop: 

Bottom of Form

* **The Form's Action Attribute and the Submit Button**

When the user clicks on the "Submit" button, the content of the form is sent to the server. The form's action attribute defines the name of the file to send the content to. The file defined in the action attribute usually does something with the received input.

|  |
| --- |
| <form name="input" action="html\_form\_submit.php"  method="get">  Username:  <input type="text" name="user">  <input type="submit" value="Submit">  </form> |

How it looks in a browser:

Top of Form

Username: 

Bottom of Form

If you type some characters in the text field above, and click the "Submit" button, the browser will send your input to a page called "html\_form\_submit.php". The page will show you the received input.

### CSS

CSS was first developed in 1997, as a way for Web developers to define the look and feel of their Web pages. It was intended to allow developers to separate content from design so that HTML could perform more of the function that it was originally based on - the markup of content, without worry about the design and layout.

CSS didn’t gain in popularity until around 2000, when Web browsers began using more than the basic font and color aspects of CSS. And now, all modern browsers support all of CSS Level 1, most of CSS Level 2, and some aspects of CSS Level 3.

Web Designers that don’t use CSS for their design and development of Web sites are rapidly becoming a thing of the past. And it is arguably as important to understand CSS as it is to know HTML - and some would say it was more important to know CSS.

### CSS is an Abbreviation

It stands for Cascading Style Sheet.

Style sheet refers to the document itself. Style sheets have been used for document design for years. They are the technical specifications for a layout, whether print or online. Print designers use style sheets to insure that their designs are printed exactly to specifications. A style sheet for a Web page serves the same purpose, but with the added functionality of also telling the viewing engine (the Web browser) how to render the document being viewed.

### Where is CSS Used?

CSS is used to style Web pages. But there is more to it than that. CSS is used to style XHTML and XML markup. This means that anywhere you have XML markup (including XHTML) you can use CSS to define how it will look.

CSS is also used to define how Web pages should look when viewed in other media than a Web browser. For example, you can create a print style sheet that will define how the Web page should print out and another style sheet to display the Web page on a projector for a slide show.

**What is WAMP?**

I am sure a lot of my friends who know WAMP very well must be thinking he is stupid who is writing about WAMP but I think it is a good idea to write about WAMP for a lot of young web software (webware) developers out there who don’t know what is WAMP and how it can benefit their software development.

So let’s start with the basics, WAMP is Free and Open Source Cross-Platform Web Server Solution Stack. It comes with Apache Web Server, MySQL Database, PHP adn Perl Programming Languages. Together with these there is an SMTP and FTP server included as well.

So what’s special?

The best feature of WAMP is ease of installation. With WAMP you don’t need to install each of the individual components of a web server individually and hence the installation is easier and quicker. No need to worry about the PHP configuration or MySQL setup etc.

Also, it is portable and if you don’t want to run WAMP automatically every time your computer starts; you don’t need to install the services.

**Components**

WAMPP 2.1e-x3.2.exe, including:

Apache 2.2.21

MySQL 5.5.16

PHP 5.3.8

**What is JavaScript?**

JavaScript is a [programming language](https://techterms.com/definition/programming_language) commonly used in [web development](https://techterms.com/definition/web_development). It was originally developed by Netscape as a means to add dynamic and interactive elements to websites. While JavaScript is influenced by [Java](https://techterms.com/definition/java), the [syntax](https://techterms.com/definition/syntax) is more similar to [C](https://techterms.com/definition/cplusplus) and is based on ECMAScript, a scripting language developed by Sun Microsystems.

JavaScript is a client-side scripting language, which means the [source code](https://techterms.com/definition/sourcecode) is processed by the client's [web browser](https://techterms.com/definition/web_browser) rather than on the [web server](https://techterms.com/definition/web_server). This means JavaScript [functions](https://techterms.com/definition/function) can run after a webpage has loaded without communicating with the server. For example, a JavaScript function may check a web form before it is submitted to make sure all the required [fields](https://techterms.com/definition/field) have been filled out. The JavaScript code can produce an error message before any information is actually transmitted to the server.

Like server-side scripting languages, such as [PHP](https://techterms.com/definition/php) and [ASP](https://techterms.com/definition/asp), JavaScript code can be inserted anywhere within the [HTML](https://techterms.com/definition/html) of a [webpage](https://techterms.com/definition/webpage). However, only the [output](https://techterms.com/definition/output) of server-side code is displayed in the HTML, while JavaScript code remains fully visible in the source of the webpage. It can also be referenced in a separate [.JS](https://fileinfo.com/extension/js) file, which may also be viewed in a browser.

Below is an example of a basic JavaScript function that adds two numbers. The function is called with the parameters 7 and 11. If the code below were included in the HTML of a webpage, it would display the text "18" in an [alert box](https://techterms.com/definition/alertbox).

JavaScript can be implemented using JavaScript statements that are placed within the **<script>... </script>**.

You can place the **<script>** tags, containing your JavaScript, anywhere within your web page, but it is normally recommended that you should keep it within the **<head>** tags.

The <script> tag alerts the browser program to start interpreting all the text between these tags as a script. A simple syntax of your JavaScript will appear as follows.

<script ...>

JavaScript code

</script>

The script tag takes two important attributes −

* **Language** − This attribute specifies what scripting language you are using. Typically, its value will be javascript. Although recent versions of HTML (and XHTML, its successor) have phased out the use of this attribute.
* **Type** − This attribute is what is now recommended to indicate the scripting language in use and its value should be set to "text/javascript".

So your JavaScript segment will look like −

<script language="javascript" type="text/javascript">

JavaScript code

</script>

## Your First JavaScript Script

Let us take a sample example to print out "Hello World". We added an optional HTML comment that surrounds our JavaScript code. This is to save our code from a browser that does not support JavaScript. The comment ends with a "//-->". Here "//" signifies a comment in JavaScript, so we add that to prevent a browser from reading the end of the HTML comment as a piece of JavaScript code. Next, we call a function **document.write** which writes a string into our HTML document.

This function can be used to write text, HTML, or both. Take a look at the following code.

<html>

<body>

<script language="javascript" type="text/javascript">

<!--

document.write("Hello World!")

//-->

</script>

</body>

</html>

**Whitespace has no meaning outside of quotation marks**

**var** foo = 'hello world';

**Parentheses indicate precedence**

2 \* 3 + 5; *// returns 11; multiplication happens first*  
2 \* (3 + 5); *// returns 16; addition happens first*

**Tabs enhance readability, but have no special meaning**

**var** foo = **function**() {  
 **console**.log('hello');  
};

## [Operators](https://autotelicum.github.io/Smooth-CoffeeScript/literate/js-intro.html#TOC)

### [Basic Operators](https://autotelicum.github.io/Smooth-CoffeeScript/literate/js-intro.html#TOC)

Basic operators allow you to manipulate values.

**Concatenation**

**var** foo = 'hello';  
  
**var** bar = 'world';  
  
**console**.log(foo + ' ' + bar); *// 'hello world'*

**Multiplication and division**

2 \* 3;  
2 / 3;

**Incrementing and decrementing**

**Var** I = 1;  
  
**var** j = ++i; *// pre-increment: j equals 2; i equals 2*  
**var** k = i++; *// post-increment: k equals 2; i equals 3*

### [Operations on Numbers & Strings](https://autotelicum.github.io/Smooth-CoffeeScript/literate/js-intro.html#TOC)

In JavaScript, numbers and strings will occasionally behave in ways you might not expect.

**Addition vs. concatenation**

**var** foo = 1;  
**var** bar = '2';  
  
**console**.log(foo + bar); *// 12. uh oh*

**Forcing a string to act as a number**

**var** foo = 1;  
**var** bar = '2';  
  
*// coerce the string to a number*  
**console**.log(foo + Number(bar));

The Number constructor, when called as a function (like above) will have the effect of casting its argument into a number. You could also use the unary plus operator, which does the same thing:

**Forcing a string to act as a number (using the unary-plus operator)**

**console**.log(foo + +bar);

### [Logical Operators](https://autotelicum.github.io/Smooth-CoffeeScript/literate/js-intro.html#TOC)

Logical operators allow you to evaluate a series of operands using AND and OR operations.

**Logical AND and OR operators**

**var** foo = 1;  
**var** bar = 0;  
**var** baz = 2;  
  
foo || bar; *// returns 1, which is true*  
bar || foo; *// returns 1, which is true*  
  
foo && bar; *// returns 0, which is false*  
foo && baz; *// returns 2, which is true*  
baz && foo; *// returns 1, which is true*

Though it may not be clear from the example, the || operator returns the value of the first truthy operand, or, in cases where neither operand is truthy, it’ll return the last of both operands. The && operator returns the value of the first false operand, or the value of the last operand if both operands are truthy.

Be sure to consult [the section called “Truthy and Falsy Things”](https://autotelicum.github.io/Smooth-CoffeeScript/literate/js-intro.html#truthy-and-falsy-things) for more details on which values evaluate to true and which evaluate to false.

### [Note](https://autotelicum.github.io/Smooth-CoffeeScript/literate/js-intro.html#TOC)

You’ll sometimes see developers use these logical operators for flow control instead of using if statements. For example:

*// do something with foo if foo is truthy*  
foo && doSomething(foo);  
  
*// set bar to baz if baz is truthy;*  
*// otherwise, set it to the return*  
*// value of createBar()*  
**var** bar = baz || createBar();

This style is quite elegant and pleasantly terse; that said, it can be really hard to read, especially for beginners. I bring it up here so you’ll recognize it in code you read, but I don’t recommend using it until you’re extremely comfortable with what it means and how you can expect it to behave.

### [Comparison Operators](https://autotelicum.github.io/Smooth-CoffeeScript/literate/js-intro.html#TOC)

Comparison operators allow you to test whether values are equivalent or whether values are identical.

**Comparison operators**

**var** foo = 1;  
**var** bar = 0;  
**var** baz = '1';  
**var** bim = 2;  
  
foo == bar; *// returns false*  
foo != bar; *// returns true*  
foo == baz; *// returns true; careful!*  
  
foo === baz; *// returns false*  
foo !== baz; *// returns true*  
foo === parseInt(baz); *// returns true*  
  
foo > bim; *// returns false*  
bim > baz; *// returns true*  
foo <= baz; *// returns true*

## [Conditional Code](https://autotelicum.github.io/Smooth-CoffeeScript/literate/js-intro.html#TOC)

Sometimes you only want to run a block of code under certain conditions. Flow control — via if and else blocks — lets you run code only under certain conditions.

**Flow control**

**Var** foo = **true**;  
**var** bar = **false**;  
  
**if** (bar) {  
 *// this code will never run*  
 **console**.log('hello!');  
}  
  
**if** (bar) {  
 *// this code won't run*  
} **else** {  
 **if** (foo) {  
 *// this code will run*  
 } **else** {  
 *// this code would run if foo and bar were both false*  
 }  
}

### [Note](https://autotelicum.github.io/Smooth-CoffeeScript/literate/js-intro.html#TOC)

While curly braces aren’t strictly required around single-line if statements, using them consistently, even when they aren’t strictly required, makes for vastly more readable code.

Be mindful not to define functions with the same name multiple times within separate if/else blocks, as doing so may not have the expected result.

### [Truthy and Falsy Things](https://autotelicum.github.io/Smooth-CoffeeScript/literate/js-intro.html#TOC)

In order to use flow control successfully, it’s important to understand which kinds of values are “truthy” and which kinds of values are “falsy.” Sometimes, values that seem like they should evaluate one way actually evaluate another.

**Values that evaluate to**true

'0';  
'any string';  
[]; *// an empty array*  
{}; *// an empty object*  
1; *// any non-zero number*

**Values that evaluate to false**

0;  
''; *// an empty string*  
NaN; *// JavaScript's "not-a-number" variable*  
null;  
undefined; *// be careful -- undefined can be redefined!*

### [Conditional Variable Assignment with The Ternary Operator](https://autotelicum.github.io/Smooth-CoffeeScript/literate/js-intro.html#TOC)

Sometimes you want to set a variable to a value depending on some condition. You could use an if/else statement, but in many cases the ternary operator is more convenient. [Definition: The ternary operator tests a condition; if the condition is true, it returns a certain value, otherwise it returns a different value.]

**The ternary operator**

*// set foo to 1 if bar is true;*  
*// otherwise, set foo to 0*  
**var** foo = bar ? 1 : 0;

While the ternary operator can be used without assigning the return value to a variable, this is generally discouraged.

### [Switch Statements](https://autotelicum.github.io/Smooth-CoffeeScript/literate/js-intro.html#TOC)

Rather than using a series of if/else if/else blocks, sometimes it can be useful to use a switch statement instead. [Definition: Switch statements look at the value of a variable or expression, and run different blocks of code depending on the value.]

**A switch statement**

**switch** (foo) {  
  
 **case** 'bar':  
 alert('the value was bar -- yay!');  
 **break**;  
  
 **case** 'baz':  
 alert('boo baz :(');  
 **break**;  
  
 default:  
 alert('everything else is just ok');  
 **break**;  
  
}

Switch statements have somewhat fallen out of favor in JavaScript, because often the same behavior can be accomplished by creating an object that has more potential for reuse, testing, etc. For example:

**var** stuffToDo = {  
 'bar' : **function**() {  
 alert('the value was bar -- yay!');  
 },  
  
 'baz' : **function**() {  
 alert('boo baz :(');  
 },  
  
 'default' : **function**() {  
 alert('everything else is just ok');  
 }  
};  
  
**if** (stuffToDo[foo]) {  
 stuffToDo[foo]();  
} **else** {  
 stuffToDo['default']();  
}

We’ll look at objects in greater depth later in this chapter.

## [Loops](https://autotelicum.github.io/Smooth-CoffeeScript/literate/js-intro.html#TOC)

Loops let you run a block of code a certain number of times.

**Loops**

*// logs 'try 0', 'try 1', ..., 'try 4'*  
**for** (**var** i=0; i<5; i++) {  
 **console**.log('try ' + i);  
}

Note that in Loops even though we use the keyword var before the variable name *i*, this does not “scope” the variable *i* to the loop block. We’ll discuss scope in depth later in this chapter.

### [The for loop](https://autotelicum.github.io/Smooth-CoffeeScript/literate/js-intro.html#TOC)

A for loop is made up of four statements and has the following structure:

**for** ([initialisation]; [conditional]; [iteration])  
 [loopBody]

The initialisation statement is executed only once, before the loop starts. It gives you an opportunity to prepare or declare any variables.

The conditional statement is executed before each iteration, and its return value decides whether or not the loop is to continue. If the conditional statement evaluates to a falsey value then the loop stops.

The iteration statement is executed at the end of each iteration and gives you an opportunity to change the state of important variables. Typically, this will involve incrementing or decrementing a counter and thus bringing the loop ever closer to its end.

The loopBody statement is what runs on every iteration. It can contain anything you want. You’ll typically have multiple statements that need to be executed and so will wrap them in a block ( {...}).

Here’s a typical for loop:

**A typical**for**loop**

**for** (**var** i = 0, limit = 100; i < limit; i++) {  
 *// This block will be executed 100 times*  
 **console**.log('Currently at ' + i);  
 *// Note: the last log will be "Currently at 99"*  
}

### [The while loop](https://autotelicum.github.io/Smooth-CoffeeScript/literate/js-intro.html#TOC)

A while loop is similar to an if statement, except that its body will keep executing until the condition evaluates to false.

**while** ([conditional]) [loopBody]

Here’s a typical while loop:

**A typical**while**loop**

**var** i = 0;  
**while** (i < 100) {  
  
 *// This block will be executed 100 times*  
 **console**.log('Currently at ' + i);  
  
 i++; *// increment i*  
  
}

You’ll notice that we’re having to increment the counter within the loop’s body. It is possible to combine the conditional and incrementer, like so:

**A while loop with a combined conditional and incrementer**

**var** i = -1;  
**while** (++i < 100) {  
 *// This block will be executed 100 times*  
 **console**.log('Currently at ' + i);  
}

Notice that we’re starting at -1 and using the prefix incrementer (++i).

### [The do-while loop](https://autotelicum.github.io/Smooth-CoffeeScript/literate/js-intro.html#TOC)

This is almost exactly the same as the while loop, except for the fact that the loop’s body is executed at least once before the condition is tested.

**do** [loopBody] **while** ([conditional])

Here’s a do-while loop:

**A do-while loop**

**do** {  
  
 *// Even though the condition evaluates to false*  
 *// this loop's body will still execute once.*  
  
 alert('Hi there!');  
  
} **while** (**false**);

These types of loops are quite rare since only few situations require a loop that blindly executes at least once. Regardless, it’s good to be aware of it.

### [Breaking and continuing](https://autotelicum.github.io/Smooth-CoffeeScript/literate/js-intro.html#TOC)

Usually, a loop’s termination will result from the conditional statement not evaluating to true, but it is possible to stop a loop in its tracks from within the loop’s body with the break statement.

**Stopping a loop**

**for** (**var** i = 0; i < 10; i++) {  
 **if** (something) {  
 **break**;  
 }  
}

You may also want to continue the loop without executing more of the loop’s body. This is done using the continue statement.

**Skipping to the next iteration of a loop**

**for** (**var** i = 0; i < 10; i++) {  
  
 **if** (something) {  
 **continue**;  
 }  
  
 *// The following statement will only be executed*  
 *// if the conditional 'something' has not been met*  
 **console**.log('I have been reached');  
  
}

## [Reserved Words](https://autotelicum.github.io/Smooth-CoffeeScript/literate/js-intro.html#TOC)

JavaScript has a number of “reserved words,” or words that have special meaning in the language. You should avoid using these words in your code except when using them with their intended meaning.

|  |  |  |  |
| --- | --- | --- | --- |
| Abstract | boolean | break | byte |
| case | Catch | char | class |
| const | continue | debugger | default |
| delete | Do | double | else |
| enum | Export | extends | final |
| finally | Float | for | function |
| goto | If | implements | import |
| in | instanceof | int | interface |
| long | Native | new | package |
| private | protected | public | return |
| short | Static | super | switch |
| synchronized | This | throw | throws |
| transient | Try | typeof | var |
| void | volatile | while | with |

## [Arrays](https://autotelicum.github.io/Smooth-CoffeeScript/literate/js-intro.html#TOC)

Arrays are zero-indexed lists of values. They are a handy way to store a set of related items of the same type (such as strings), though in reality, an array can include multiple types of items, including other arrays.

**A simple array**

**var** myArray = [ 'hello', 'world' ];

**Accessing array items by index**

**var** myArray = [ 'hello', 'world', 'foo', 'bar' ];  
**console**.log(myArray[3]); *// logs 'bar'*

**Testing the size of an array**

**var** myArray = [ 'hello', 'world' ];  
**console**.log(**myArray**.length); *// logs 2*

**Changing the value of an array item**

**var** myArray = [ 'hello', 'world' ];  
myArray[1] = 'changed';

While it’s possible to change the value of an array item as shown in “Changing the value of an array item”, it’s generally not advised.

**Adding elements to an array**

**var** myArray = [ 'hello', 'world' ];  
**myArray**.push('new');

**Working with arrays**

**var** myArray = [ 'h', 'e', 'l', 'l', 'o' ];  
**var** myString = **myArray**.join(''); *// 'hello'*  
**var** mySplit = **myString**.split(''); *// [ 'h', 'e', 'l', 'l', 'o' ]*

## [Objects](https://autotelicum.github.io/Smooth-CoffeeScript/literate/js-intro.html#TOC)

Objects contain one or more key-value pairs. The key portion can be any string. The value portion can be any type of value: a number, a string, an array, a function, or even another object.

[Definition: When one of these values is a function, it’s called a method of the object.] Otherwise, they are called properties.

As it turns out, nearly everything in JavaScript is an object — arrays, functions, numbers, even strings — and they all have properties and methods.

**Creating an “object literal”**

**var** myObject = {  
 sayHello : **function**() {  
 **console**.log('hello');  
 },  
 myName : 'Rebecca'  
};  
**myObject**.sayHello(); *// logs 'hello'*  
**console**.log(**myObject**.myName); *// logs 'Rebecca'*

#### [Note](https://autotelicum.github.io/Smooth-CoffeeScript/literate/js-intro.html#TOC)

When creating object literals, you should note that the key portion of each key-value pair can be written as any valid JavaScript identifier, a string (wrapped in quotes) or a number:

**var** myObject = {  
 validIdentifier: 123,  
 'some string': 456,  
 99999: 789  
};

Object literals can be extremely useful for code organization; for more information, read [Using Objects to Organize Your Code](http://blog.rebeccamurphey.com/2009/10/15/using-objects-to-organize-your-code/) by Rebecca Murphey.

## [Functions](https://autotelicum.github.io/Smooth-CoffeeScript/literate/js-intro.html#TOC)

Functions contain blocks of code that need to be executed repeatedly. Functions can take zero or more arguments, and can optionally return a value.

Functions can be created in a variety of ways:

**Function Declaration**

**function** foo() { */\* do something \*/* }

**Named Function Expression**

**var** foo = **function**() { */\* do something \*/* }

I prefer the named function expression method of setting a function’s name, for some rather[*in-depth and technical reasons*](http://kangax.github.com/nfe/). You are likely to see both methods used in others’ JavaScript code.

### [Using Functions](https://autotelicum.github.io/Smooth-CoffeeScript/literate/js-intro.html#TOC)

**A simple function**

**var** greet = **function**(person, greeting) {  
 **var** text = greeting + ', ' + person;  
 **console**.log(text);  
};  
greet('Rebecca', 'Hello');

**A function that returns a value**

**var** greet = **function**(person, greeting) {  
 **var** text = greeting + ', ' + person;  
 **return** text;  
};  
  
**console**.log(greet('Rebecca','hello'));

**A function that returns another function**

**var** greet = **function**(person, greeting) {  
 **var** text = greeting + ', ' + person;  
 **return** **function**() { **console**.log(text); };  
};  
  
  
**var** greeting = greet('Rebecca', 'Hello');  
greeting();

### [Self-Executing Anonymous Functions](https://autotelicum.github.io/Smooth-CoffeeScript/literate/js-intro.html#TOC)

A common pattern in JavaScript is the self-executing anonymous function. This pattern creates a function expression and then immediately executes the function. This pattern is extremely useful for cases where you want to avoid polluting the global namespace with your code — no variables declared inside of the function are visible outside of it.

**A self-executing anonymous function**

(**function**(){  
 **var** foo = 'Hello world';  
})();  
  
  
**console**.log(foo); *// undefined!*

### [Functions as Arguments](https://autotelicum.github.io/Smooth-CoffeeScript/literate/js-intro.html#TOC)

In JavaScript, functions are “first-class citizens” — they can be assigned to variables or passed to other functions as arguments. Passing functions as arguments is an extremely common idiom in jQuery.

**Passing an anonymous function as an argument**

**var** myFn = **function**(fn) {  
 **var** result = fn();  
 **console**.log(result);  
};  
  
myFn(**function**() { **return** 'hello world'; }); *// logs 'hello world'*

**Passing a named function as an argument**

**var** myFn = **function**(fn) {  
 **var** result = fn();  
 **console**.log(result);  
};  
  
**var** myOtherFn = **function**() {  
 **return** 'hello world';  
};  
myFn(myOtherFn); *// logs 'hello world'*

# DataTables CDN

The DataTables [Content Delivery Network (CDN)](http://en.wikipedia.org/wiki/Content_delivery_network) is a permanent store of the software released as part of the DataTables project for you to use on your site without needing to host it yourself.

Furthermore, the CDN can serve the files for the DataTables components and dependencies that you require as a single concatenated and minified file, or as individual files through use of the [DataTables download builder](https://datatables.net/download).

All you then need to do is include the files you want on your page! This can help improve performance on your site and make prototyping much easier.

The DataTables CDN is powered by the [CloudFlare](http://cloudflare.com/) network.

**OBJECTIVE**

The system helps in buying of goods, products and services online by choosing the listed products from website.

**SCOPE OF THE PROJECT:**

Most generic consumer to consumer website,which covers almost all possible categories,with 2 level listing -

1. Maximize benefits and minimize the disadvantages of a common e-commerce website.

2. User friendly, Vendor friendly environment.

**Requirement Analysis**

## HARDWARE AND SOFTWARE USED:

## Hardware:

|  |  |
| --- | --- |
| Hardware Environment (Deployment Server) | |
| PC | Linux Server or windows |
| Processor | Intel i5 or i3 |
| RAM | 800 GB RAM |
| HDD | 1 TB |
| Monitor | Color Monitor |
| Network | Any network supporting TCP/IP |
|  |  |

|  |  |
| --- | --- |
| Hardware Environment (End user’s PC) | |
| PC | PC (Windows), Apple, Linux |
| Processor | Intel i5 or i3 |
| RAM | 8 GB RAM |
| HDD | 10GB of free HDD space for Internet Cache |
| Monitor | Color Monitor |
| Network | Any network supporting TCP/IP |

## Software:

|  |  |
| --- | --- |
| Software Environment (Deployment Server) | |
| Operating System | Linux and windows 8 or higher |
| Application Server | WampServer or XAMPP Server |
| Web Browser: | Google Chrome or Mozilla Firefox |
| Other software | HTML/Ajax/JavaScript/CSS/Bootstrap/Data Table API/Visualizer API/Animation  flash 11 |

|  |  |
| --- | --- |
| Software Environment (End User) | |
| Operating System | Window 8 or higher, Apple, Linux |
| Browser | Google Chrome or Mozilla Firefox |
| ActiveX/Plug-in | Flash Player 11 |

**System Design**

**Introduction:**

Design is the abstraction of a solution; it is a general description of the solution to a problem without the details. Design is view patterns seen in the analysis phase to be a pattern in a design phase. After design phase we can reduce the time required to create the implementation. In this chapter we are introduce context diagram, models, system architecture, principal system object, design model and object interface.

**Use Case Diagram:**

Administrator

User





**ER- Diagram:**

user

Customer

Adminstrator

**Data Flow Diagram Symbols: -**

**-source& destination**

**-Data Flow**

**-Process**

**-Storage**

**ZERO LEVEL DFD**

Administrator

user

**ONE-LEVEL Diagram:**

Admin

New User

Registered Customer

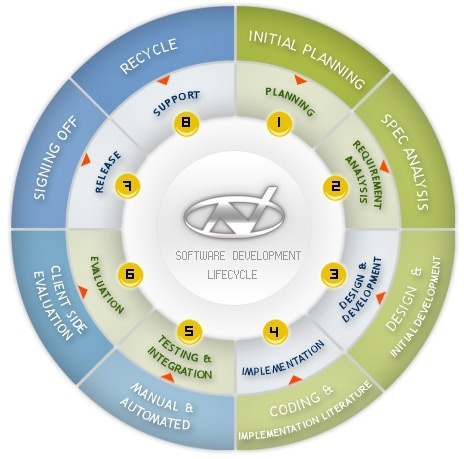
**DFD FOR User**

User

**Implementation:**

**Software Development Life Cycle**

The Systems Development Life Cycle (SDLC) is a conceptual model used in project management that describes the stages involved in an information system development project from an initial feasibility study through maintenance of the completed application. Various SDLC methodologies have been developed to guide the processes involved including the waterfall model (the original SDLC method), rapid application development(RAD), joint application development (JAD), the fountain model and the spiral model. Mostly, several models are combined into some sort of hybrid methodology. Documentation is crucial regardless of the type of model chosen or devised for any application, and is usually done in parallel with the development process. Some methods work better for specific types of projects, but in the final analysis, the most important factor for the success of a project may be how closely particular plan was followed. The following figure shows a general life cycle Process in software development:

****

**Figure4.1-Software development Lifecycle**

The most common steps in all the development methodologies are as follows:

* **System/Information Engineering and Modelling:**

As software is always of a large system (or business), work begins by establishing the requirements for all system elements and then allocating some subset of these requirements to software. This system view is essential when the software must interface with other elements such as hardware, people and other resources. System is the basic and very critical requirement for the existence of software in any entity. So if the system is not in place, the system should be engineered and put in place. In some cases, to extract the maximum output, the system should be re-engineered and spruced up. Once the ideal system is engineered or tuned, the development team studies the software requirement for the system.

* **Software Requirement Analysis:**

This process is also known as feasibility study. In this phase, the development team visits the customer and studies their system. They investigate the need for possible software automation in the given system. By the end of the feasibility study, the team furnishes a document that holds the different specific recommendations for the candidate system. To understand the nature of the program(s) to be built, the system engineer or "Analyst" must understand the information domain for the software, as well as required function, behavior, performance and interfacing. The essential purpose of this phase is to find the need and to define the problem that needs to be solved.

* **System Analysis and Design:**

In this phase, the software development process, the software's overall structure and its nuances are defined. A software development model is thus created. Analysis and Design are very crucial in the whole development cycle. Any glitch in the design phase could be very expensive to solve in the later stage of the software development. Much care is taken during this phase. The logical system of the product is developed in this phase.

* **Code Generation:**

The design must be translated into a machine-readable form. The code generation step performs this task. If the design is performed in a detailed manner, code generation can be accomplished without much complication. Programming tools like compilers, interpreters, debuggersetc are used to generate the code. Different high levelprogramming languages are used for coding. With respect to the type of application, the right programming language is chosen.

**Front-End:**

FRONT END:

1. Html
2. CSS
3. JavaScript
4. Bootstrap
5. jQuery
6. Data Table API
7. Visualizer API
8. Animation
9. Font Awesome

**Screen Shots:**

* Front-Page:

A picture containing object

Description generated with high confidence

* View Music Playlist:

A screenshot of a cell phone screen with text

Description generated with very high confidence

* Visualizer Page:

A screenshot of a computer

Description generated with high confidence

A screen shot of a computer

Description generated with high confidence

**BIBLIOGRAPHY**

I have referred to some books and have also sought help from various websites during the course of the project. The books and websites referred are as follows:

**BOOKS**

1. Web Development

2.The Web Book

**WEB REFERENCES**

Stack Overflow

[www.w3schools.com](http://www.w3schools.com)

**BACHELOR OT TECHNOLOGY**

**MAJOR PROJECT**

**ONLINE MUSIC PLAYER**

**SUBMITTED BY**

ASHRAFI SHAHEEN

1420085

**SuBMITTED TO**

**MR ASHISH OBEROI**



**RIMT INSTITUTE OF ENGINEERING & TECHNOLOGY**

Mandi Gobindgarh

**TABLE OF CONTENTS**

|  |  |
| --- | --- |
| **S.NO.** | **TITLE** |
| 1. | Introduction |
| 2. | Technologies Used |
| 3. | Objective |
| 4. | Scope of The Project |
| 5. | Requirement Analysis |
| 6. | System Design |
| 7. | Implementation |
| 8. | Screenshots |
| 9. | Bibliography |