Zodiac Architecture Revamp Assessment (ZARA)

UI & UX Design and Implementation Guidelines

September 09, 2015
Prepared by
INTECH Creative Services Pvt Ltd

1 TECHNOLOGY SELECTION

Layers/Component	Technology
Language	HTML 5.0 HTML5 is a mark-up language used for structuring and presenting content on the World Wide Web. It was finalized, and published, on 28 October 2014 by the World Wide Web Consortium (W3C). This is the fifth revision of the HTML standard since the inception of the World Wide Web.
Stylesheet	CSS 3.0 CSS3 is the latest evolution of the Cascading Style Sheets language and aims at extending CSS2.1. It brings a lot of long-awaited novelties, like rounded corners, shadows, gradients, transitions or animations, as well as new layouts like multi-columns, flexible box or grid layouts.
Scripting Language	JQuery JQuery is a cross-platform JavaScript library designed to simplify the client-side scripting of HTML. Version: 1.11.3 or v2.1.4 Website: https://jquery.com/
Framework	Bootstrap Bootstrap, a sleek, intuitive, and powerful mobile first front-end framework for faster and easier web development. Version: v3.3.5 Website: http://getbootstrap.com/

Table: 1 Technology selection

2 BROWSER SELECTION

	Default Browser	Chrome	Firefox	IE	Safari
Windows (7,8,10)	2 (H)	3		1 (10+)	
Mac OS X (10.10)		2	3		1
Android (4.2 – 5.1)	2 (Samsung)	1			

iOS (7 & 8.4)		2			1

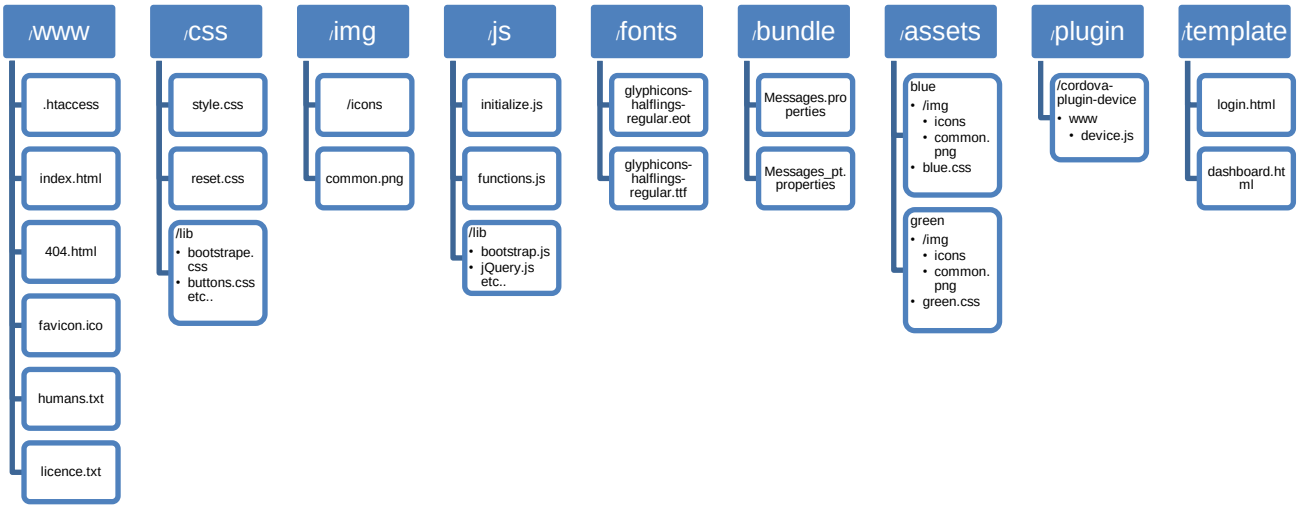
Table: 2 Browser Selection

Note: Certification of testing will be done for the priority 1.0 as in above live.

3 FOLDER STRUCTURE

Application will have specific organization of the files and folders of our web page. Clean, organized file tree for Zodiac application will not only make development easier, but improve the experience for our visitors.

Below is graphical user interface for managing files and folders to create our file tree.



3.1 Folder Structure

Location	File Name/Directory	Purpose
/www	index.html	The default file name for a Web site's home page
	.htaccess	Directory-level configuration file supported by several web servers, that allows for decentralized management of web server configuration.
	robots.txt	It's a TXT file that contains information about the different people who have contributed to building the website.
	favicon.ico	It's the little icon beside your site's name in the favourites list, before the URL in the address bar and bookmarks folder and as a bookmarked website on the desktop in some operating systems.
	humans.txt	It contains the human's responsibilities, roles and technologies specific information.
	licence.txt	License shell mean the terms and conditions for use
	404.html	Error page
/css	style.css	custom website CSS rules

	reset.css	often compressed (minified) set of CSS rules that *resets* the styling of all HTML elements to a consistent baseline
	/lib/ bootstrap.css	Bootstrap framework CSS rules
	/lib/buttons.css etc..	Bootstrap framework CSS rules related to buttons style and etc...
/img	/icons	All icons which are used in website
	common.png	Needed images with .png,.jpg and .gif
/js	initialize.js	Common function and operations
	functions.js	customized functions file related to operations perform on web page
	/lib/ bootstrap.js	Contain all plug-in in a single file
	/lib/jQuery.js etc..	Cross platform JavaScript library
/fonts	glyphicons-halflings-regular.eot	GLYPHICONS is a library of precisely prepared icons and symbols
	glyphicons-halflings-regular.ttf	GLYPHICONS's ttf file and other use file added
/bundle	Messages.properties	Language. Properties files from a hook, allowing you to change any of the messages displayed by Life ray to suit your needs
	Messages_pt.properties	This property returns the name of the action associated with the message, as a string.
/assets	/blue	Theme colour name Folder
	/blue/img/icons.png	All Icons split.png images
	/blue/img/common.png	Other useful split images name with common.png
	/green	Theme colour name Folder
	/green/img/icons.png	All Icons split.png images
	/green/img/common.png	Other use full split images name with common.png
/plugin	/cordova-plugin-device	Cordova plugin for device
	/www/device.js	In www folder those js files here ex. device.js
/template	login.html	Login page of website
	dashboard.html	Dashboard of website

Table: 3.1 Folder Structure

4 GENERIC DEVELOPMENT STANDARDS

A coding standard tells developers how they must write their code. Instead of each developer coding in their own preferred style, they will write all code to the standards outlined in the document.

This makes sure that a large project is coded in a consistent style parts are not written differently by different programmers. Not only does this solution make the code easier to understand, it also ensures that

any developer who looks at the code will know what to expect throughout the entire application. Coding standards are great but how do you decide which standards you want to apply, and how they will be defined?

Important Note

The listed guidelines incorporates the most of the standard implementations considering specific language of development. If the developer is using any 3rd party libraries which follows its own standard format, they don't have to re-organize the library with respect to 3rd party guidelines. But, in case if additional code is required with in such library, then they must follow Zodiac standard guidelines.

Indentation

Proper and consistent indentation is important in producing easy to read and maintainable programs.

- Emphasize the body of a control statement such as a loop or a select statement.
- Emphasize the body of a conditional statement.
- Emphasize a new scope block.

4.1 Coding Guidelines

General coding guidelines provide the programmer with a set of best practices which can be used to make programs easier to read and maintain.

Coding guidelines specify a common format for the source code and comments. This allows developers to easily share code, and the ideas expressed within the code and comments, between each other. It also specifies how comments (internal documentation) should be handled. More importantly, a well designed standard will also detail how certain code should be written, not just how it looks on screen.

4.1.1 General

Title	Description
General Localization	Check user interface, content files, text filters, hot keys, spelling rules, sorting rules and upper/lower case conversion rules.
Content Checking	Check video, static & dynamic content (e.g. catalogs, search results, metadata).
Characters	Use appropriate character sets for the chosen languages.
Language Direction	Left to right or right to left?
Postal Codes	Verify if the target countries support only number alphanumeric.
Tax Calculation Verification	Test the rules around the calculation of VAT, sales tax.
Currency Conversion	Test for currency conversion for internet retailers applications.
Translation Validation	Test on-demand translation validation and accuracy.
Folder Structure	Use Standard folder structure.
Inline Comment	Functioning of the subroutine or key aspects of the algorithm shall be frequently used.
Classes, Functions, and Methods	Keep functions, and methods reasonably sized. This depends upon the language being used.

Source Files	The name of the source file or script shall represent its function.
Variable Name	Variable shall have mnemonic or meaningful names that convey to a casual observer, the intent of its use.
Compiler Warning	Two types of messages: warnings and errors.

Table: 4.1.1 General

4.1.2 Layout / Design

Title	Description
Page Layout	Evaluate the overall effectiveness of the page layout.
Color Schemas	Background, text, links, icons, and buttons must go well together.
Findability	Key items like Help, About, Instructions & Search should be easy to find.
Localization	Consistent in terms of text, messages and symbols in all supported languages.
Touch screens	Layout, Navigation, button size, workflow, and other input methods if supported.
Hard Keys	Check that the hard keys work with the app: Start, Home, Menu, Back with the app in a similar way.
Short Cuts	If there are any expected shortcuts for the device, test their use within the app.

Table: 4.1.2 Layout/Design

4.1.3 Usability

Title	Description
Sign-up & Login	Make sure it is easy to use.
Menu Options	Test to make certain menu options are easy to find and navigate.
Keys	Check if the app will run well with a keyboard and/or touch-screen.
Image	Standard Guidelines • Images should have alt attribute. • Uniform GUI across the Application. • Images should be optimized so that site can load faster. • All images should have a preloaded. • Images display correctly in different browsers.
Navigation	Standard Guidelines • Main navigation links should be clear and consistent. • Buttons in whole site should be same in look & feel and should have nice hover effect. • Site logo linked to homepage.

	• Buttons should be of same size, shape, font type & font size.
Email	Email functionality of the system. Email should not be spammed. It should go in the Inbox.
Year	Standard Guidelines • Leap years are validated correctly. • Standard Date format.
Data Handling	Manage data handling and avoid data deletion.
Connection Speed / Carrier	Check across the most popular & likely carriers.
Operating System	Mobile /Web OS running the app to evaluate relative performance.
Connection Issue	Standard Guidelines • Loading. (Page must be loading as fast.) • Performance of memory. • Large amount of data accessed by user (at this time site must be work proper, not hang. whenever, used multiple user this site.)
Screen Size	Different Desktop, phones and tablets to identify screen size discrepancies.
Mandatory	Mandatory fields should be marked with "*" in red color.
Links	Standard Guidelines • All internal links should work & open in same window. • All external links should work & open in new window.
Sorting	Sorting should be work proper as per Ascending & Descending.
Error message	Standard Guidelines • Must be clear, concise and actionable. • No grammatical errors. • Error fields should be presented correctly and should be displayed in same • way in all pages. • Is all the error message text spelt correctly on this screen?
Consistent	Standard Guidelines • Consistent Menu in whole site. • Consistent Site logo/header. • All headings should have same font type & size & style.
Space	Spacing between Labels and Controls.
Look & Feel	Standard Guidelines • Assure that all windows have a consistent look and feel.

	• Assure that all dialog boxes have a consistent look and feel. • Hyperlinks use distinct visited, active and hover colors and css.
Screen Resolution	Check site on common variations of Screen Resolution 1024 x 768, 1280 x 1024.
Multi Media	Standard Guidelines • Audio/video/flash has clear purpose. • Files should be fast to download. • Flash should have preload. • Proper links should be available to download the files. • Should show a related image when flash or any other plug-in is disabled.
Web Standards	Standard Guidelines • Correct Doc Type. • Correct character encoding. • Valid CSS (W3c Validator).

Table: 4.1.3 Usability

4.1.4 Security

Title	Description
Web Applications	Check with Firefox web developer (or similar) and use a web proxy to intercept and monitor.
Browsers	Firefox, Chrome, Internet Explorer, Safari
Format	Watch for unencrypted password/usernames and make sure footprint & fingerprint analysis can be used to make hash files.
Local Market Compliance	Research permissions configuration to ensure that your application can get access to the device areas it needs in order to function properly. Provide useful error messages.
Levels & Permissions	Verify successful installation at all expected device security level settings and verify it's possible to change the security settings of your device or app permissions.

Table: 4.1.4 Security

5 CODING STANDARD IMPLEMENTATION

Dashboard will be the default view which will open first time after user logged into community portal. It will contain list of all companies, notifications, visitors, projects, quick links etc…

Things need to take care:

- List of companies by default will be based on latest added.
- By default latest 5 companies will get displayed.

5.1 HTML Implementation Guidelines

5.1.1 HTML Structure

Doc Type	Standard Guidelines HTML document must start with the Document Type Definition (DTD) • Common Doc type : <!DOCTYPEhtml> • Possible Versions : HTML5 EXAMPLE <!DOCTYPE html>
HEAD	Standard Guidelines It contains below elements and code • Title <title> - element is required. It should be meaningful as possible. EXAMPLE <title> Page title comes here </title> • Meta tags <meta> - To describe metadata within HTML Document. It is always passed as name/value pairs. Some examples • <meta name="description" content="Free Web tutorials"> • <meta name="keywords" content="HTML,CSS,XML,JavaScript"> • <meta name="viewport" content="width=device-width, initial-scale=1"> • CSS/Favicon <link> - Include external css file links and favicon icon. Example: <link href="css/bootstrap.min.css" rel="stylesheet"> <link rel="icon" type="image/x-icon" href="favicon.ico"> • Scripts <script> - Include scripting languages like JavaScript, • Example. • <script type="text/javascript" src="js/jquery.js" /> • <script type="text/javascript"> • //generate javascript code here.. • </script>

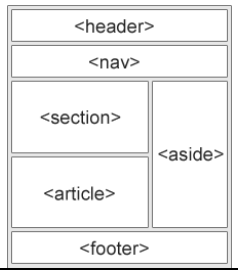
<table>
<tr><td rowspan="2">BODY</td><td colspan="2">It describes the viewable portion of the page.
<header>, <nav>, <section>, <article>, <aside>, , <footer></td></tr>
<tr><td>Diagram
 
</td><td>Code Example
 
Click here to view code
view-source:http://design.iprojectlab.com/zara/html_sample.html</td></tr>
</table>

Table: 5.1.1 Html Structure

Example code

view-source:http://design.iprojectlab.com/zara/html_sample.html
http://design.iprojectlab.com/zara/html_sample.html

5.1.2 HTML Standard Guideline

Title	Description
Document Type	<!DOCTYPE html> - must declare this doctype
HTML code	Html code structure follow in tree structure otherwise convert in beautifier code html : http://codebeautify.org/css-beautify-minify
Css file	Standard Guidelines • All Css are include in only head section • bootstrap css always put before your custom css ex.style.css
Tags	Standard Guidelines

	• All open tags must be close. • All tag name use lowercase only. • Use html5 tags – like <header>,<nav><section><aside><footer>
Class	Standard Guidelines • never use any custom class with .container class • We use bootstrap framework so must be use that predefine classes like .container , .row and grid related all class – col-md-4
Id	• Don't use same id in multiple time
Image	• All images have alt text

Stylesheet Naming Conventions in HTML	`/* Do not */`
	`<div class="box profile pro-user">`
	`<img class="avatar image" />`
	`<p class="bio">...</p>`
	`</div>`
	`/* Do */`
	`<div class="box profile profile-is-pro-user">`
	`<img class="avatar profile_image" />`
	`<p class="profile_bio">...</p>`
	`</div>`

Table: 5.2 HTML Guideline standard

5.1.3 Images rule in html

Title	Description
images to compress	Ask yourself: "Are all my graphics absolutely necessary?" If not, remove unneeded images. You can scan your site here for suggestions on which images to compress.
Preload images	This will not improve the cost on initial download, but can make other pages on the site that use the images load much faster
Responsive images	Create responsive images by adding an .img-responsive class to the <img> tag. The image will then scale nicely to the parent element. **Example** <img src="cinqueterre.jpg" class="img-responsive" alt="Cinque Terre">

<table>
<tr><td>Alt text</td><td colspan="2">An image with an alternate text specified:

Example

<img src="smiley.gif" alt="Smiley face"></td></tr>
<tr><td>Image ratio</td><td colspan="2">Give image width and height

Example

<img src="smiley.gif" alt="Smiley face" height="42" width="42"></td></tr>
<tr><td>Shape of images</td><td colspan="2">The classes below can be used to style any image:

.img-rounded - Adds rounded corners to an image (not available in IE8)
.img-circle -Shapes the image to a circle (not available in IE8)
.img-thumbnail -Shapes the image to a thumbnail</td></tr>
<tr><td rowspan="2">Sprites Image</td><td colspan="2">which are made up of combining small images into one larger image such as. All icons which are use in website made up in one image

Example</td></tr>
<tr><td>Sprite image

Making the Image Sprite

  </td><td>Css code

#home {
width: 46px;
height: 44px;
background: url(img_navsprites.gif) 0 0;
}
#next {
width: 43px;
height: 44px;
background-position: url(img_navsprites.gif) -91px 0;
}</td></tr>
</table>

Table: 5.1.3 Image rules in HTML

5.1.4 Don't use in html

Title	Description
Don't Writing Old-School HTML	Examples here are using <table> elements for layout, or elements when other semantic-specific tags would be more appropriate, or using tags that are not supported in current HTML standard, such as or <font>, or spacing items on a page with a large number of & nbsp; entities.
Bloated Responses	Don't adopt the mindset that access to the internet is getting faster and faster, thus allowing for bloated scenarios. Instead, consider everything going back and forth from the browser to your site as a cost. Images are a major offender in page bloat. To minimize the cost of images that slow down page loads.
Shape of image	Never cut images in any shape like not in circle, and ovel and other shap.
Individual Section	Don't merge all section in one section, specify own section with comment
Js in footer	Don't use Js files after </html>

Table: 5.1.4 Don't use in HTML

5.1.5 Font Specification guideline

Title	Description

Custom font	create or have created your custom font file.
	• Upload your font files to your server under your Web root.
	• Add code like the following to your CSS files or within the <style> tags of your HTML header:
	@font-face
	{
	font-family: myFancyFont;
	src: url('myFancyFont.ttf'),
	url('myFancyFont.eot');
	}
	• Put the new font in your style sheet as shown in the following example:
	h1 { font-family: 'myFancyFont', Times, serif; }

Google font	Help of this site : https://www.google.com/fonts/ • Select font which you use in website and the click on "use" button • Only select that font style which are you need like , light , regular, bold • Add in head like this: <link href='https://fonts.googleapis.com/css?family=Open+Sans:400,300,700' rel='stylesheet' type='text/css'>

	• In css file import font like this: @import url(https://fonts.googleapis.com/css?family=Open+Sans:400,300,700); And in css use font in font-family property ex: font-family: 'Open Sans', sans-serif;

Shorthand	Syntax
	
	font: font-style font-variant font-weight font-size/line-height font-family;
	
	Use
	
	body {
	font: italic small-caps normal 13px/150% Arial, Helvetica, sans-serif;
	}

Table: 5.1.5 Font specification guideline

5.2 CSS Implementation Guidelines

Table of Contents

A simple table of contents will—in order, naturally—simply provide the name of the section and a brief summary of what it is and does,

Example

```
/**
 * CONTENTS
 *
 * SETTINGS
 * Global.....Globally-available variables and config.
 *
 * TOOLS
 * Mixins.....Useful mixins.
 *
 * GENERIC
 * Normalize.css.....A level playing field.
 * Box-sizing.....Better default `box-sizing`.
 *
 * BASE
 * Headings.....H1–H6 styles.
 *
 * OBJECTS
 * Wrappers.....Wrapping and constraining elements.
 *
 * COMPONENTS
 * Page-head.....The main page header.
 * Page-foot.....The main page footer.
 * Buttons.....Button elements.
 *
 * TRUMPS
 * Text.....Text helpers.
 */
```

Anatomy of a Ruleset

```
[selector] {
    [property]: [value];
    [<--declaration--->]
}
```

5.2.1 Selector

Title	Description
Start / close	The opening curly brace is divided by a space from the selector and a line-break for the statements and close with closeing curly Brace.
generic classes	It is encouraged to use generic classes for common styling issues instead of "id".
multiple Selectors use	If you apply a style to multiple selectors, separate the selectors with a comma and a line-break. Example class1, .class2

Alignment	Attempt to align common and related identical strings in declarations, for Example
	
	.foo {
	-webkit-border-radius: 3px;
	-moz-border-radius: 3px;
	border-radius: 3px;
	}
	
	.bar {
	position: absolute;
	top: 0;
	right: 0;
	bottom: 0;
	left: 0;
	margin-right: -10px;
	margin-left: -10px;
	padding-right: 10px;
	padding-left: 10px;
	
	}

Whitespace	Standard Guidelines • One (1) empty line between closely related rule sets. • Two (2) empty lines between loosely related rule sets. • Five (5) empty lines between entirely new sections.

Table: 5.2.1 Selector

5.2.2 Property

Title	Description
Shorthand code	When using the short version for properties like "padding", "margin" or "border", instead of "padding-left", "padding-right"..., all four values are preferred (padding: 10px 0 10px 0). Two values are also allowed in this case (padding: 10px 0).
Null-values	Null-values should be abbreviated with "0" instead of "0em" or "0px".
Format rule	Every CSS command is **indented** with a tab, every property is followed by a colon and then a **white-space**, every statement is separated by a **line-break**.
Class / ID name	All CSS selectors (classes or IDs) are lowercase; multiple words are separated with a hyphen, underscore or camel-case.
Naming convention	Standard Guidelines • All strings in classes are delimited with a hyphen (-), like so: .page-head {} .sub-content {} Naming convention not use • Camel case and underscores are not used for regular classes; the following are incorrect: .pageHead {} .sub_content {}
code represents	• code represents its own Block: ex. .page {} .content {} .sub-content {} .footer {}

	.footer_copyright {} • Incorrect notation for this would be: Ex. .page {} .page_content {} .page_sub-content {} .page_footer {} .page_copyright {}

Multi-line CSS	• A reduced chance of merge conflicts, because each piece of functionality exists on its own line.
	• for example:
	
	.icon {
	display: inline-block;
	width: 16px;
	height: 16px;
	background-image: url(/img/sprite.svg);
	}
	
	.icon-home { background-position: 0 0 ; }
	.icon-person { background-position: -16px 0 ; }
	.icon-files { background-position: 0 -16px; }

Table: 5.2.2 Property

5.2.3 Value

Title	Description

Single Responsibility	/* Do Not */
	.error-message {
	display: block;
	padding: 10px;
	border-top: 1px solid #f00;
	border-bottom: 1px solid #f00;
	background-color: #fee;
	color: #f00;
	font-weight: bold;
	}
	
	.success-message {
	display: block;
	padding: 10px;

	border-top: 1px solid #0f0;
	border-bottom: 1px solid #0f0;
	background-color: #efe;
	color: #0f0;
	font-weight: bold;
	}
	
	Use style in individual css class
	
	/* Do */
	.box {
	display: block;
	padding: 10px;
	}
	
	.message {
	border-style: solid;
	border-width: 1px 0;
	font-weight: bold;
	}
	
	.message-error {
	background-color: #fee;
	color: #f00;
	}
	
	.message-success {
	background-color: #efe;
	color: #0f0;
	}

Prefixes	CSS features created specifically for a browser, be cautious as to how you will approach vendor prefixes such as -webkit-, -moz-, or -ms-.

Table: 5.2.3 Value

5.2.4 Comments in CSS

Title	Description
Section name Comment	The tile section is prefixed with a hash (#) symbol to allow us to perform more targeted searches. /*----------*\ #SECTION-TITLE *----------*/ .selector {}

Single Comment	Use /* */ for comment
	Signle line comment
	
	Example
	
	li {
	float:left;
	text-indent:0px;
	/* list-style-type:none; */
	}
	
	Multi line comment
	
	Example
	
	/* li {
	float:left;
	text-indent:0px;
	list-style-type:none;
	}
	*/

Remove comments	When css minify that time remove all comments

Table: 5.2.4 Comments in css

5.3 JavaScript Implementation Guidelines

JavaScript is most commonly used as a client side, interpreted, object oriented, high level scripting language. This means that JavaScript code is written into an HTML page. When a user requests and HTML page with JavaScript in it, the script is sent to the browser and it's up to the browser to do some-thing with it. JavaScript is easy to learn.

It is most popular programming language in the world. It can be included on web page to make them more interactive. We can use it to check or modify the contents of forms, change images, open new windows and write dynamic page content. We can even use it with CSS to make DHTML (Dynamic Hypertext Markup Language).

This style guide is a list of dos and don'ts for JavaScript programs.

5.3.1 Standard Guidelines

5.3.1.1 Embedded JavaScript

JavaScript code should not be embedded in HTML files unless the code is specific to a single session. Code in HTML adds significantly to page weight with no opportunity for mitigation by caching and compression.

Example of Embed JavaScript into HTML

Filename: js_example.html

```
1  <!DOCTYPE html>
2  <html>
3          <head>
```

```
4                   <title>JavaScript Embedded Example</title>
5           </head>
6           <body>
7
8                   <h1>Example</h1>
9                   <p id="demo">It is a JavaScript example embedded in HTML.</p>
10
11                  <button type="button" onclick="myFunction()">Try it</button>
12
13                  <script src="myscript.js"></script>
14
15          </body>
16  </html>
```

Filename: myscript.js

```
1   /* call my function on click of button event */
2
3   function myFunction(){
4           document.getElementById("demo").innerHTML = "Paragraph Changed";
5   }
```

5.3.1.2 No Inline Js

Never used inline JavaScript. Define it in separate file. So user can manage and read. It is easy to access

Example

<script src=filename.js> tags should be placed as late in the body as possible.

This reduces the effects of delays imposed by script loading on other page components. There is no need to use the language or type attributes. It is the server, not the script tag, which determines the MIME type.

5.3.2 Language Rules Guidelines

Title	Description
var	Standard Guidelines • Declaration with var: always • Reason : ○ When you fail to specify var, the variable gets placed in the global context, potentially clobbering existing values. Also, if there's no declaration, it's hard to tell in what scope a variable lives (e.g., it could be in the Document or Window just as easily as in the local scope).
Constants	Standard Guidelines • Use NAMES_LIKE_THIS for constant *values*. All constant value should be in capital letters.

	• Use @const to indicate a constant (non-overwritable) *pointer* (a variable or property). • Never use the const keyword as it's not supported in Internet Explorer.

Example :

```
/**
 * Request timeout in milliseconds.
 * @type {number}
 */
js.example.TIMEOUT_IN_MILLISECONDS = 60;
```

Semicolons	Standard Guidelines • Always use semicolons. • Reason : ○ JavaScript requires statements to end with a semicolon, except when it thinks it can safely infer their existence. Otherwise it will give you an error.

Semicolons should be included at the end of function expressions, but not at the end of function declarations.

The distinction is best illustrated with an example

```
var foo = function() {
  return true;
};  // semicolon here.

function foo() {
  return true;
}  // no semicolon here.
```

Nested Functions	Nested functions can be very useful to create continuations and for the task of hiding helper functions.
Exceptions	Do not avoid exceptions when using an application development framework, etc.
Custom Exceptions	Standard Guideline • Without custom exceptions, returning error information from a function that also returns a value can be tricky, not to mention inelegant. • Bad solutions include passing in a reference type to hold error information or always returning Objects with a potential error member. • These basically amount to a primitive exception handling hack. Feel free to use custom exceptions when appropriate.

Table: 5.3.2 Language rules Guidelines

5.3.3 JavaScript Styling Rules

5.3.3.1 Naming Convention

Title	Description

	Standard Guideline
	• In general use, function name, variable name, class name, method name and namespace should be like below :
	  ○ FunctionNamesLikeThis
	• File name should be like below :
	  ○ filenamelikethis.js
	• Constant values should be like below :
	  ○ CONSTANT_VALUES_LIKE_THIS

Properties and methods	Standard Guideline • Private properties and methods should be named with a trailing underscore. • Protected properties and methods should be named without a trailing underscore (like public ones).
Method and function parameter	Standard Guideline • Optional function arguments start with `opt_`. • Functions that take a variable number of arguments should have the last argument named `var_args`. • Optional and variable arguments can also be specified in `@param` annotations. Although either convention is acceptable to the compiler, using both together is preferred.
Accessor functions	Standard Guideline • Getters and setters methods are not required. However if we are used, then getters must be named *getFoo()* and setters must be named *setFoo(value)*. • For Boolean getters better to use *isFoo()*.

Table: 5.3.3.1 Naming Convention

5.3.3.2 Code formatting

Code format should be proper for function, methods and variable or array define in code.

Curly Braces

Because of implicit semicolon insertion, always start your curly braces on the same line as wherever they are opening.

Example

```
if (something) {
```

```
  // ...
} else {
  // ...
}
```

Array and Object Initializers

Single-line array and object initializers are allowed when they fit on a line:

Example

```
var arr = [1, 2, 3];  // No space after [ or before ].
var obj = {a: 1, b: 2, c: 3};  // No space after { or before }.
```

Multiline array initializers and object initializers are indented 2 spaces, with the braces on their own line, just like blocks.

Example

```
// Object initializer.
var inset = {
  top: 10,
  right: 20,
  bottom: 15,
  left: 12
};

// Array initializer.
this.rows_ = [
  '"Slartibartfast" <fjordmaster@magrathea.com>',
  '"Zaphod Beeblebrox" <theprez@universe.gov>',
  '"Ford Prefect" <ford@theguide.com>',
  '"Arthur Dent" <has.no.tea@gmail.com>',
  '"Marvin the Paranoid Android" <marv@googlemail.com>',
  'the.mice@magrathea.com'
];
```

5.3.3.3 Code Indentation

- Always use 4 spaces for indentation of code blocks:
- Do not use tabs (tabulators) for indentation. Different editors interpret tabs differently.

5.3.3.4 Line Length

Avoid lines longer than 80 characters.

- When a statement will not fit on a single line, it may be necessary to break it.
- Place the break after an operator, ideally after a comma.
- A break after an operator decreases the likelihood that a copy-paste error will be masked by semicolon insertion. The next line should be indented 8 spaces.

Example

5.3.3.5 Comments

Be generous with comments.

- It is useful to leave information that will be read at a later time by people (possibly yourself) who will need to understand what you have done.
- The comments should be well-written and clear, just like the code they are annotating. An occasional nugget of humour might be appreciated.

- It is important that comments be kept up-to-date. Erroneous comments can make programs even harder to read and understand.
- Make comments meaningful. Focus on what is not immediately visible. Don't waste the reader's time with stuff like
    - i = 0; // Set i to zero.

Generally use line comments. Save block comments for formal documentation.

Example

```
/**
 * A JSDoc comment should begin with a slash and 2 asterisks.
 * Inline tags should be enclosed in braces like {@code this}.
 * @desc Block tags should always start on their own line.
 */
```

Tags Specifications

@author: Define author of a file or an owner of test, only used to overview of file comments.
@desc : Define description of method, property and functions used.
@param: Define parameter used in functions and methods.

5.3.3.6 Whitespace

Blank lines improve readability by setting off sections of code that are logically related.

Blank spaces should be used in the following circumstances:

- A keyword followed by ((left parenthesis) should be separated by a space.
    - while (true) {
- A blank space should not be used between a function value and its ((left parenthesis). This helps to distinguish between keywords and function invocations.
- All binary operators except . (period) and ((left parenthesis) and [(left bracket) should be separated from their operands by a space.
- No space should separate a unary operator and its operand except when the operator is a word such as typeof.
- Each ; (semicolon) in the control part of a for statement should be followed with a space.
- Whitespace should follow every , (comma).

5.3.4 Coding Rules

5.3.4.1 Selectors

Selector allows you to select and manipulate HTML elements in JavaScript or jQuery

- Use ID selector whenever possible. It is faster in use because they are handled using *document.getElementById()*.
- When use Class selectors, don't use the element type in your selector.

    Example

    - var $products = $("div.products"); // SLOW
    - var $products = $(".products"); // FAST
- Be specific on the right side of your selector, and less specific on the left.

    Example

- // Unoptimized
  $("div.data .gonzalez");

- // Optimized
  $(".data td.gonzalez");

- Avoid Excessive Specificity

Example

- $(".data table.attendees td.gonzalez");

- // Better: Drop the middle if possible.
  $(".data td.gonzalez");

- Avoid Universal Selectors.

Example

- $('div.container > *'); // BAD
- $('div.container').children(); // BETTER

5.3.4.2 Events

Event methods trigger or attach a function to an *event* handler for the selected elements.

- Use only one Document Ready handler per page. It makes it easier to debug or keep track of the behaviour view.
- DO NOT use anonymous function to attach events. Anonymous functions are difficult to debug, test, reuse or maintain.

Example

//DO NOT USE

- $("#myLink").on("click", function(){...});

//USE

- function myLinkClickHandler(){...}
  $("#myLink").on("click", myLinkClickHandler);

- When possible, use custom namespace for events. It's easier to unbind the exact event that you attached without affecting other events bound to the DOM element.

Example

- $("#myLink").on("click.mySpecialClick", myEventHandler); // GOOD
- // Later on, it's easier to unbind just your click event
  $("#myLink").unbind("click.mySpecialClick");

5.3.4.3 Ajax

Ajax is the art of exchanging data with a server, and update parts of a web page – without reloading the whole page.

- Avid using *.getJson()* or *.get()*, Simply use the *$.ajax()* as that's what gets called internally.
- DO NOT put request parameters in the URL, send them using data object setting.

Example

- // Less readable...

```
$.ajax({
  url: "something.php?param1=test1&param2=test2",
  ....
});
```

```
// More readable...
$.ajax({
  url: "something.php",
  data: { param1: test1, param2: test2 }
});
```

- Try to specify the data Type setting so it's easier to know what kind of data you are working with.

5.3.4.4 Chaining

JQuery or $ returns itself (an object) which allows the chaining.

- Use chaining as an alternative to variable caching and multiple selector calls.

  Example

  ```
  $("#myDiv").addClass("error").show();
  ```

- Whenever the chain grows over 3 links or gets complicated because of event assignment, use appropriate line breaks and indentation to make the code readbale.

  Example

  ```
  $("#myLink")
      .addClass("bold")
      .on("click", myClickHandler)
      .on("mouseover", myMouseOverHandler)
      .show();
  ```

5.3.4.5 Miscellaneous

Some common things need to be take care while create JavaScript or jQuery program.

- Use Object literals for parameters.

  Example

  ```
  $myLink.attr("href", "#").attr("title", "my link").attr("rel", "external"); // BAD, 3 calls to attr()
  // GOOD, only 1 call to attr()
  $myLink.attr({
      href: "#",
      title: "my link",
      rel: "external"
  });
  ```

- Do not mix css with jQuery.

  Example

  ```
  $("#mydiv").css({'color':red, 'font-weight':'bold'}); // BAD
  .error { color: red; font-weight: bold; } // GOOD
  $("#mydiv").addClass("error"); // GOOD
  ```

- Do not use Deprecated Methods. It is always important to keep an eye on deprecated methods for each new version and try to avoid using them

```
/*
* jQuery custom datagrid plugin
* @author author name
* @desc short decription about the plugin,functions,methods and purpose of it what is used in files
* @defaults define default parameters used in file
*/

(function ($) {

    $.fn.dataGrid = function(options){

        // public helper vars

        var self = $(this);
        var checkBox;
        var actionBtn;
        var pagination;
        var pageHeader;
        var sortClass = "";
        var columnOrder;
        var RowOrder;
        var setClass;
        var triggerPrompt = 0;
        var defaults = $.extend({
            url : false,
            datatype:'json',
            method : 'POST',
            columns: [],
            paging: false,
            pageno : 1,
            total : 0,
        },options);
        var methods = {

            /* ------------------------------------------------------
                @method addCheckBox is used to add checkbox in file
                @param rowId on which row checkbox will be added
                @param recordIdString store the id of record
                @output - function will return checkbox output with defined
                necessary parameters.
            ------------------------------------------------------ */
            addCheckBox : function(rowId,recordIdString){
                checkBox = $(checkBoxTDContainer);
                $(checkBox).find("input").attr({"data-row":rowId,
                                                "id":"box"+rowId,
                                                "data-row-string":recordIdString
                                            });
                $(checkBox).find("label").attr({"for":"box"+rowId});
                return $(checkBox);
            },
            addAction : function(rowId){
                actionBtn = $(plusBtnTDContainer);
                actionBtn.find(".btn-more-view").attr("id",rowId);
                return $(actionBtn);
            },
            init : function(){
                $(".reset-data").html("");
                $(".reset-data_pagination").html("");
                $(".page-header").html("");
                if($("#side-menu .folderItemChild a").hasClass("mobile_view_toggle")){
                    $("#side-menu").slideUp("slow");
                }

                // function call to load file

                loadScript(setWebUrl+"/view/gridview.js", function(){
                    methods.addGrid(defaults.columns);
                });
            }
        };
        return methods.init();
    };
})(jQuery);
```

Example code

http://design.iprojectlab.com/zara/javascript_sample.js

6 BOOTSTRAP

- Download Source code website on http://getbootstrap.com
- Must be include bootstrap.min.css and bootstrap.min.css and jquery.min.js
- Must be include meta for responsive in device –
- <meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=1, user-scalable=no">
- <!--[if lt IE 9]>
  <scriptsrc="https://oss.maxcdn.com/html5shiv/3.7.2/html5shiv.min.js"></script><script src="https://oss.maxcdn.com/respond/1.4.2/respond.min.js"></script>
  <![endif]-->
- Rows must be placed within a **.container** (fixed-width) or **.container-fluid** (full-width) for proper alignment and padding.
- Use .**rows** to create horizontal groups of columns.
- Content should be placed within columns, and only columns may be immediate children of rows.
- Predefined grid classes like **.row** and **.col-xs-4** are available for quickly making grid layouts. Less mixings can also be used for more semantic layouts.
- Columns create gutters (gaps between column content) via padding. That padding is offset in rows for the first and last column via negative margin on **.rows**.
- The negative margin is why the examples below are outdented. It's so that content within grid columns is lined upmwith non-grid content.
- Grid columns are created by specifying the number of twelve available columns you wish to span. For example, three equal columns would use three .col-xs-4.
- If more than 12 columns are placed within a single row, each group of extra columns will, as one unit, wrap onto a new line.
- Grid classes apply to devices with screen widths greater than or equal to the breakpoint sizes, and override grid classes targeted at smaller devices. Therefore, e.g. applying any .col-md-* class to an element will not only affect its styling on medium devices but also on large devices if a .col-lg-* class is not present.

6.1 Version migration

Follow version update specification guidelines as per site: http://getbootstrap.com/migration/
CSS migration

- Check major classes changes shows the style changes between last version and update version
- Whats new in updated version - added new elements and changed some existing ones
- Whats removed- elements have been dropped or changed in last version and update version

JS Migration

Just replace js files.

6.2 Grid

.col-*-12 { width: 100%; }
.col-*-11 { width: 91.66666667%; }
.col-*-10 { width: 83.33333333%; }
.col-*-9 { width: 75%; }
.col-*-8 { width: 66.66666667%; }
.col-*-7 { width: 58.33333333%; }

.col-*-6 { width: 50%; }
.col-*-5 { width: 41.66666667%; }
.col-*-4 { width: 33.33333333%; }
.col-*-3 { width: 25%; }
.col-*-2 { width: 16.66666667%; }
.col-*-1 { width: 8.33333333%; }

6.2.1 Three equal columns

Get three equal-width columns **starting at desktops and scaling to large desktops**. On mobile devices, tablets and below, the columns will automatically stack.

.col-md-4	.col-md-4	.col-md-4

6.2.2 Three unequal columns

Get three columns **starting at desktops and scaling to large desktops** of various widths. Remember, grid columns should add up to twelve for a single horizontal block. More than that, and columns start stacking no matter the viewport.

.col-md-3	.col-md-6	.col-md-3

6.2.3 Two columns

Get two columns **starting at desktops and scaling to large desktops**.

.col-md-8	.col-md-4

6.2.4 Full width, single column

No grid classes are necessary for full-width elements.

	Extra small devices Phones (<768px)	**Small devices Tablets (≥768px)**	**Medium devices Desktops(≥992px)**	**Large devices Desktops(≥1200 px)**
Grid behavior	Horizontal at all times	Collapsed to start, horizontal above breakpoints		
Container width	None (auto)	750px	970px	1170px
Class prefix	.col-xs-	.col-sm-	.col-md-	.col-lg-
# of columns	12			
Column width	Auto	~62px	~81px	~97px
Gutter width	30px (15px on each side of a column)			
Nestable	Yes			
Offsets	Yes			
Column ordering	Yes			

Table: 6.2.4 Full width, single column

6.2.5 Grid – Mixed: mobile, tablet, and desktop

Build on the previous example by creating even more dynamic and powerful layouts with tablet .col-sm-* classes.

.col-xs-12 .col-sm-6 .col-md-8		.col-xs-6 .col-md-4
.col-xs-6 .col-sm-4	.col-xs-6 .col-sm-4	.col-xs-6 .col-sm-4

```
<div class="row">
        <div class="col-xs-12 col-sm-6 col-md-8">.col-xs-12 .col-sm-6 .col-md-8</div>
        <div class="col-xs-6 col-md-4">.col-xs-6 .col-md-4</div>
</div>
<div class="row">
        <div class="col-xs-6 col-sm-4">.col-xs-6 .col-sm-4</div>
        <div class="col-xs-6 col-sm-4">.col-xs-6 .col-sm-4</div>
        <!-- Optional: clear the XS cols if their content doesn't match in height -->
        <div class="clearfix visible-xs-block"></div>
        <div class="col-xs-6 col-sm-4">.col-xs-6 .col-sm-4</div>
</div>
```

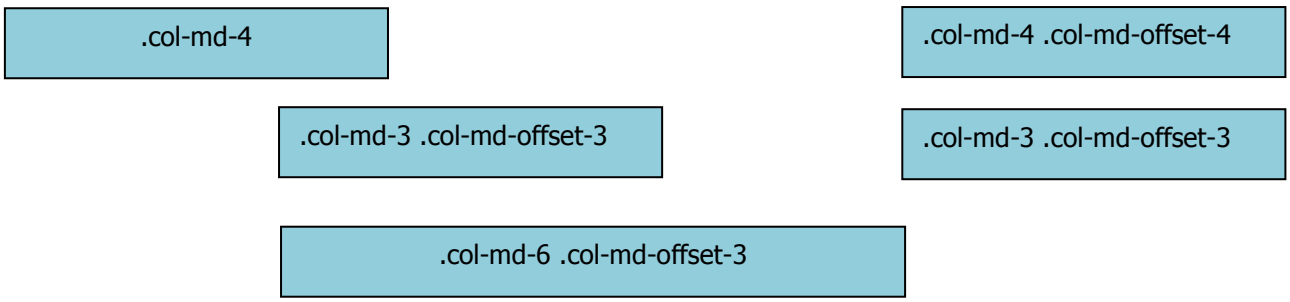
6.2.6 Helper Classes

6.2.6.1 Clear fix

Normally used to clear floats, adding this class to any column will make it shift to a new row automatically, to help you correct problems that occur with uneven column heights.

6.2.6.2 Offsetting columns

You don’t have to occupy all 12 of the virtual columns. You can use offset classes like .col-xs-offset-* or .col-md-offset-* to leave a particular number of virtual Bootstrap columns to the left of any column (kind of like invisible place holders).




```
<div class="row">
        <div class="col-md-4">.col-md-4</div>
        <div class="col-md-4 col-md-offset-4">.col-md-4 .col-md-offset-4</div>
</div>
<div class="row">
        <div class="col-md-3 col-md-offset-3">.col-md-3 .col-md-offset-3</div>
        <div class="col-md-3 col-md-offset-3">.col-md-3 .col-md-offset-3</div>
</div>
<div class="row">
        <div class="col-md-6 col-md-offset-3">.col-md-6 .col-md-offset-3</div>
</div>
```

6.2.6.3 Reordering

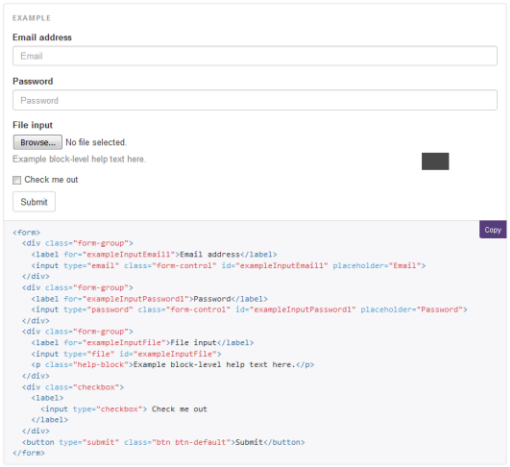
Use classes like .col-md-push-* and .col-md-pull-* to shift a column to the right or left, respectively
.

.col-md-3 .col-md-pull-9	.col-md-9 .col-md-push-3

```
<div class="row">
        <div class="col-md-9 col-md-push-3">.col-md-9 .col-md-push-3</div>
        <div class="col-md-3 col-md-pull-9">.col-md-3 .col-md-pull-9</div>
</div>
```

6.3 Forms

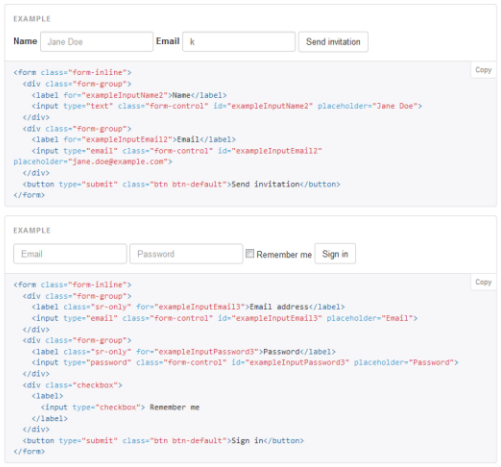
6.3.1 Basic example



Individual form controls automatically receive some global styling. All textual <input>, <textarea>, and <select> elements with .form-control are set to width: 100%; by default. Wrap labels and controls in .form-group for optimum spacing.
Ex .

<label for="exampleInputEmail1">Email address</label>
<input type="email" class="form-control" id="exampleInputEmail1" placeholder="Email">

6.3.2 Inline form



Add .form-inline to your form (which doesn't have to be a <form>) for left-aligned and inline-block controls. **This only applies to forms within viewports that are at least 768px wide.**
Always add labels
Screen readers will have trouble with your forms if you don't include a label for every input. For these inline forms, you can hide the labels using the .sr-only class. There are further alternative methods of providing a label for assistive technologies, such as the aria-label, aria-labelledby or title attribute. If none of these is present, screen readers may resort to using the placeholder attribute, if present, but note that use of placeholder as a replacement for other labelling methods is not advised.
Ex .
<form class="form-inline">

<label for="exampleInputName2">Name</label>
<input type="text" class="form-control" id="exampleInputName2" placeholder="Jane Doe">

<button type="submit" class="btnbtn-default">Send invitation</button>
</form>

6.4 Supported controls

Most common form control, text-based input fields. Includes support for all HTML5 types: text, password, datetime, datetime-local, date, month, time, week, number, email, url, search, tel, and color.

6.5 Bootstrap media queries

```
/* Large Devices, Wide Screens */
@media only screen and (max-width : 1200px) {…}
/* Medium Devices, Desktops */
@media only screen and (max-width : 992px) {…}
/* Small Devices, Tablets */
@media only screen and (max-width : 768px) {…}
/* Extra Small Devices, Phones */
@media only screen and (max-width : 480px) {…}
/* Custom, iPhone Retina */
@media only screen and (max-width : 320px) {…}
```

6.6 Things don't in bootstrap

- Do **not** modify the bootstrap.css file.
- Bootstrap scale up to 12 columns so never take column out of 12 columns.
- Bootstrap offer everything so don't use other element like responsive menu , slider
- Don't take header, navigation in div use that's html5 tags like <header>,<nav> etc..

7 BUSINESS CASES

A business case is an argument, usually documented, that is intended to convince a decision maker to approve some kind of action. The document itself sometimes referred to as a business case.

For more complex issues, a business case should be presented in a carefully constructed document. A business case document should examine benefits and risks involved with both taking the action and, conversely, not taking the action. The conclusions should be compelling arguments for implementation.

A well-crafted business case explores all feasible approaches to a given problem and enables business owners to select the option that best serves the organizations.

7.1 UI Validation

Validation is the process of checking data against a standard or requirement.

There will be two types of validations. Custom JavaScript library will be used to validate both the validations.

7.1.1.1 Validation Types

i) Client side validation

- This validation is performed by a web browser, before input is sent to a web server. In this, validation we can provide better user experience by responding quickly at the browser level. Client side validation does not require a round trip to the server, so network traffic which will help our server perform better.

  For example,

  - If user leaved mandatory field blank then it will show an error like "Please fill required field." Another example is about email id if user fill wrong email id format then it will give an error message like "Please enter valid email address". Same for other input field as per standard validation rules.

ii) Server Side Validation

- In Server side validation, the input submitted by the user is being sent to the server and validated using one of server side scripting such as PHP, Java, etc. After the validation process on server side, the feedback is sent to the client back by a new dynamically generated web page.

  For example,

  - If user stored email address in database like abc@gmail.com and he is trying to login with wrong email address like abcc@gmail.com then input value go to the server using ajax and check whether entered details are correct or not. In this case, it will return false and gives a valid error message like "Email Address does not exist".

7.2 Form Controls

HTML forms are used to control user input. It contains form elements. Form elements are different types of input elements, checkbox and radio buttons, submit and reset buttons and more. User will input data with respect of form elements. We can submit data to the server using ajax or form submit events to the server.

7.2.1.1 Form Types

There are two ways we can manage forms.

i) Static HTML Forms

- Form fields will be written with in HTML itself and access of fields will be specific on user roles. We can use numerous types of HTML form fields like input, textarea, dropdown, buttons etc. Each and every form input contains text, values and other necessary form attributes.

ii) Dynamic Form Control Management

- Form data will be managed dynamically in form of JSON array. Dynamic Form will be generated using jQuery ajax success method as per form input field types in HTML.

  For Example,

```
{
   "data":[
      {
         "name":"Container",
         "type":"inputbox",
         "validataion":"",
         "custom":"",
         "values":""
      },
      {
         "name":"Consult Person",
         "type":"inputbox",
         "validataion":"",
         "custom":"",
         "values":""
      }
   ]
}
```

7.3 Wizard Process

Good Design is not just about making your website beautiful; it's also about being user-friendly. One of the most common tools for a user-friendly website is the Wizard. Using Wizard user can be able to update information as per his convenient time and way. He just need to start fill data step by step and click on save and next. In case browser will be closed his data will be stored into server and when he comes back he will directly able to jump on that wizard window where he left to enter the details.

Wizards are easy for both user and the developer. We can also add validations, images, graphical buttons and any number of other finishing touches to make our wizard even better.

There are four general makeup to create any type of wizard.

Title	Description
Getting Started	This steps gives the basic info on what wizard will do and what information user may need in order to complete the wizard. If wizard is lengthy then it can also provide an estimate on how long it will take to complete.
Individual Steps	This is your main part of wizard. We basically break down all of the data that we want from user into logical steps.
Final Reviews	This part is also very important. Once user fills all required information and before going to submit it to the server he can review all together at this place.
Submission Confirmation	It is a final step. Once click on Submit button user's data will be stored and submitted to server. Once this step is being completed by user he/she will not be able to change or recover data as he fills in form.

7.4 Logging and Error Handling

7.5 Locale and Internationalization