Front-end Development Guidelines

The guidelines will cover standard guidelines that should be considered during production of front-end templates. These will also cover testing procedures that should be taken to validate the technologies in the likes of HTML, CSS, JavaScript used and some guidelines that should be considered for performance optimization and coding standards for front-end builds.

Table of Contents

[Front-end Development Guidelines 1](#_Toc326484329)

[Separation of Concerns 2](#_Toc326484330)

[Validation 5](#_Toc326484331)

[Code Quality & Standards 6](#_Toc326484332)

[Markup (HTML) 6](#_Toc326484333)

[Presentation (CSS) 9](#_Toc326484334)

[Behavior (JavaScript) 11](#_Toc326484335)

Directory Structure

The directory structure for the templates should adhere to a standard in such a way those assets especially the types of images i.e. background images, content images, generic & sprites should be in a specific folder structure. A good example would be to have type folders in the images folder below is proposed directory structure and if the development team has one that supersedes this one should use that and keep Gulf News Development team in the loop and understanding of that structure.

|  |  |
| --- | --- |
|  | Parent Directory |
| Cascading Style |
| @font-face fonts |
| Images Parent Directory |
| All background images go here including sprites |
| All content images |
| All generic images for complete site |
| All icons not part of sprite |
| All JavaScript goes here libraries go into lib whereas if using a build process we can have each object in a separate file. |

Separation of Concerns

Separation of Content (Markup), Presentation (CSS) & Behavior (JavaScript) should be imposed there should not be any overlapping of any of the three technologies at any point of time into each other’s layer.

So no CSS code should be mixed in HTML neither any JavaScript code should be written inside HTML layer. Each should sit in its own layer with loosely coupled selector based linking.

*Below tags references are all written in uppercase for better word formatting for the purpose of this document not for actual HTML Markup. All elements in the actual document should be in lower case.*

Validation

All layers involved will be validated against their propriety validators as per W3C standards:

* 1. HTML & XHTML   
     All files markup should validate against a valid type and DTD either any of XHTML Strict or Transitional 1.0, HTML5 or any custom based on the requirements. The markup should also comply with Accessibility standards under WCAG 1.0 guidelines with a minimum Priority 2(AA) checklist.
  2. CSS  
     All CSS files should validate against CSS 2.1 specifications. CSS3 specifications can be taken into account which has wide support with browsers and a failsafe fallback is provided to help gecko or non-supporting browsers to degrade gracefully.
  3. JavaScript  
     All JavaScript files should be validate against JSLINT or JSHINT for code quality and defined patterns for performance and anti-patterns that needed to be removed or replaced with a better code that preserve for better performance.

Code Quality & Standards

Markup (HTML)

Markup should only define the structure and outline & content of the document and offer a structured content. Markup should not be mixed with Inline CSS or JavaScript to define the look and feel or behavior of the content on the page beyond rudimentary concepts such as headers, paragraphs, and lists. The presentation attributes of HTML have all been deprecated and style should be contained in style sheets.

[HTML5](http://www.ibm.com/developerworks/library/wa-webstandards/index.html) is the version of HTML & XHTML we tend to follow for all our new developments. [HTML5 draft](http://www.w3.org/TR/html5/) defines a single language that can be written in HTML & XML.

HTML5 DOCTYPE would be used and any cross-browser and gracefully degradable feature would be used where ever possible.

Markup would be validated as per the validation section for mostly well-formed code. 100% valid code is not required if valid code is not aligned with user experience we will prefer maintaining user experience over valid code.

Page should be readable without CSS & content hierarchy should be maintained.

Template

* [HTML5BOILERPLATE](http://html5boilerplate.com) or [INITIALIZR](http://www.initializr.com/) would be used as a starting point for all our future developments

Guidelines

All markup should be delivered as UTF-8 as it is most friendly for internationalization.   
<META CHARSET=”UTF-8”> charset attribute in <META> tags can be used.

Below are some guidelines to follow for structuring of markup. Markup should be semantically correct and well-formed. All IDs and CLASS names should be in lower case

*Headings:*

* All pages must use heading elements to define structure of document
* Heading elements should be ordered hierarchically i.e. if <H2></H2> tag is defined somewhere in document it should have a preceding <H1></H1> tag
* Headings should be followed by further content <H3>Title</H3><P>Content Text</P>
* Headings should not have a consecutive series of same level in hierarchy <H3>Title</H3><H3>Sub Title</H3>
* All headings and title should be title cased in the markup. If required can be transformed using CSS text-transform property

*Lists:*

* <UL></UL> should only be used when order of the list is not editorially significant
* <OL></OL> should be used when order of the list is editorially significant
* <UL></UL>, <OL></OL> must have at least 1 <LI></<LI>
* Nested list may not be more than 3 levels deep

*Tags must not be used:*

* <B>, <I>, <BIG>, <BLINK>, <MARQUEE>, <S>, <SMALL>, <STRIKE>, <TT>, <U>, <CENTER>, <NOBR>, <FONT>

*Tags must be used when content match their description:*

* <P>, <STRONG>, <EM>

*Tags should be used when content match their description:*

* <BLOCKQUOTE>, <Q>, <CITE>, <ABBR>, <DFN>, <CODE>, <SAMP>, <KBD>, <VAR>, <INS>, <DEL>, <ADDRESS>, <PRE>, <BR>, <SUP>, <SUB>

*Tag attributes:*

* HEIGHT & WIDTH maybe used for images and embedded media
* BORDER attribute should not be used

*Tables:*

* Tables should only be used to represent tabular data unless a valid must
* Make use of <THEAD></THEAD>, <TBODY></TBODY> & <TH></TH> where appropriate
* SUMMARY attribute should be used to describe editorial intent of data

*MICROFORMATS:*

* Use [MICROFORMATS](http://microformats.org/wiki/Main_Page#Specifications) and/or MICRODATA where possible or appropriate especially HCARD and ADR

*Forms:*

* Use <LABEL></LABEL> for all form fields with FOR Attribute to pair with ID attribute of the form element. Also labels can be styled to have pointer as cursor
* Size attribute on input fields should not be used
* <FIELDSET> element should be used to wrap <FORM> markup

*Comments:*

* All containers providing document structure must have an open and close comment indicated where a container has started and where it is ending
* All modules should be placed inside opening and closing comments indicating
* Descriptor comments should be used describing the use of a specific markup element

*HTML5 Semantic Elements:*

* HTML5 semantic elements like <NAV>, <FIGURE> & <FIGCAPTION>, <HEADER>, <FOOTER>, <SECTION>, <ASIDE>, <ARTICLE> should be used with a fallback JavaScript library for enabling these elements for IE8 and below browsers like [HTML5SHIV](http://code.google.com/p/html5shiv/) or [MODRENIZR](http://modernizr.com/)

Presentation (CSS)

Presentation should be separated from markup. This allows decrease in redundancy of presentational aspects of page design such as typography, layout, possible interaction & design features.

Template

* If HTML5BOILERPLATE is not used [NORMALIZE](http://necolas.github.com/normalize.css/) can be used for general style resets and a starting template for CSS for consistent browser rendering

Guidelines

*Placement:*

* All CSS should be added using external files
* For the purpose of source code providing files can be divided into generic.css (containing resets and generic classes & element styles) and main.css for rest of the styles
* External CSS files should be included using <LINK> tag rather than @IMPORT
* No styles should exist inside the HTML document neither in the <HEAD> section nor inline the element STYLE attribute

*Writing Guidelines:*

* Standard naming conventions should be used for id and class names with all lower case and if a name has more than 1 words should be separated by hyphen e.g. #top-nav, .content
* While writing CSS keep in mind the Cascading & Specificity of selectors
* Specificity can be overridden by !important but do understand that !important also overrides JavaScript
* Use shorthand rules as much as possible but be sure not the omit shorthand value keywords
* Use selective approach to override shorthand rules e.g.   
  padding: 0 0 10px 5px;  
  padding-left:20px;
* While using universal selector (\*) consider Specificity of selectors
* While writing selectors use relative approach to minimize the elements in selector  
  e.g. #nav ul li a can be written as #nav a
* All module styles should be independent of their containers so that modules can be replaced anywhere in the page
* Element selectors can be used to define default styling for page elements
* class selectors can be used for repeated elements or modules
* id selectors can be used to provide a structural layout to the document and hence use ID with class in conjugation for a better maintainability & well-formed code
* Do not qualify id or class selector e.g.  
  div#myid or div.myclass
* Use multi-classing in a way the next class in the order overrides/adds functionality to the one before
* Simple attribute selectors can be used but should gracefully degrade for browsers like IE6 that do not support i.e. a[href], a[href=”http://www.gulfnews.com/”], div[class~=”panel”]  
  Ideas:   
  a[href^=”https”] – Secure server links  
  a[href^=”mailto”] – Email links  
  a[href$=”.pdf”] – PDF document links
* outline should be used instead of border where borders are not required to participate in layout
* Site navigation can use CSS Menu using child selector for drop-downs etc.
* Media Queries can be used to address any cross-device incompatibilities
* Modular approach should be used for the CSS for various widgets for easy abstraction for widgets being used common to all magazines
* Avoid CSS expression
* Use of CSS Sprites is highly recommended

*Comments:*

* Styles should be detailed commented in terms of reset, generic element styles, generic classes, layout styles, typography.
* Also strong comments should be provided for all layout based wrappers
* All elements like navigation, search box etc. should also me commented
* All overrides, hacks & workarounds should be mentioned in comments
* The comments should allow mapping of selectors and properties to element on markup
* Top of the CSS an index or glossary of type, color and any other helpful information should be provided

Behavior (JavaScript)

All behavior should be placed in its own layer. For most of our projects we would be using [jQuery](http://www.jquery.com). Use a CDN to load the library with a fallback to load the library locally.

Template

* Basic structural template will be provided with the guidelines to follow using a single variable pattern
* All modules would be define in the same namespace using object literal notation for better tracing & mapping
* All configuration scripts should be placed in separate extend function
* All initialization scripts should be placed in separate extend function
* All modules should be self-contained and loosely coupled in a way that changes made to one should not affect any other

Guidelines

* 99% of code should be housed in external JavaScript files being included at the end of JavaScript
* Don’t rely on user-agent string. Do proper feature detection [MODRENIZR](http://modernizr.com/) offers that & also [HTML5BOILERPLATE](http://html5boilerplate.com) does reflects on feature detection classes in <HTML>
* Variables and functions should be named logically using camel casing like hCarousel, popUpWindow, handleTopNavigation etc.
* All Boolean variable should be prepended with is e.g. isValidObject etc.
* Documentation should follow [NaturalDocs](http://www.naturaldocs.org/documenting.html) structure
* Code should be organized in one global singleton object e.g. SCENE = {} || SCENE;
* All application functionality should be extended to the object e.g.   
  SCENE = {  
   init:function(){ //all initialization code goes here },  
   config: function(){ //all configuration objects go here }  
  };
* Then we can instantiate as if using jQuery  
  $(function(){ SCENE.init(); });
* For objects initiating only once [module pattern](http://www.uipress.com/patterns-in-javascript-the-module-pattern/) can be used and for objects needed to be initialized more than once [jQuery Plugins Pattern](http://coding.smashingmagazine.com/2011/10/11/essential-jquery-plugin-patterns/) can be used
* All site specific code should be placed in **script.js**
* All plugins should be placed in **plugins.js** file with a table of contents section at the top with plugin name, author site summary of what it does licensing either BSD, MIT or GPL etc.
* For event delegations use jQuery BIND or DELEGATE
* 2 sets of files should be provided one in the production folder and other in the source folder. The production files should be minified & combined for better performance weather the source files should be for developers to read through and implementation details should be commented
* Naming conventions: all variables camel case e.g. count, found also all functions camel case with a verb in front e.g. getName, setName, handleNavigation and all constructor functions as Pascal case e.g. Person, GroupAccount etc. Whereas constants written will all upper case with words separated by underscore(\_) e.g. MAX\_COUNT, PI, AVG etc.