Web Programming Languages Assignment #2

Name: Linghuan Hu

UTD-ID: 2021179513

Email: hulinghuan@gmail.com

1. Web Page Layout

1.1. Web page layout design tools

Tool: Axure RP Pro
Version: 6.5.0.3051

1.2. Before design web page layout

1.2.1. What is the website about

This website is a lightweight online finance management tools. After sign up, user can create their own individual and group finance management plan.
Users can add, edit, and delete what they purchased to the plan, which is the basic function of individual plan. At any time, users can generate a finance analysis report of their finance behavior between a period.
Users can also add, edit, delete the future purchase to control and manage their purchase budget.
By using the group finance management plan, user can collaborate with others to build a group finance project.

1.2.2. Use case diagram

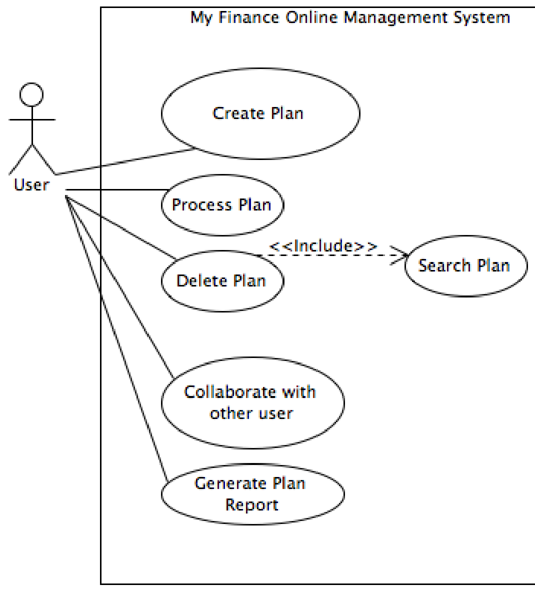


Figure 1.1 Use case diagram

1.2.3. What is the functional requirements

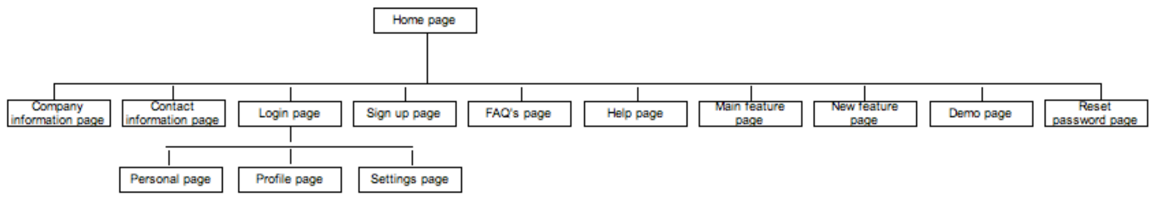
- User can login their own account.
- User can create, rename, and delete multiple individual plan and group plan.
- User can add, edit, delete purchased item information to each individual or group plan.
- User can specify the buyer and payer in a group plan.
- User can browse the finance analysis report of individual or group plan.
- Individual finance analysis report include the total amount, the total amount of every category of a time period.
- Group finance analysis report include the total amount, total amount of every each member of group, the total amount of every category of a time period.

1.2.4. What web pages we have to build

The web pages we have to build are:

- Home page
- Login in page
- Sign up page
- FAQ’s page
- Contact information page
- Company info page
- Help center page
- Main feature page
- New feature page
- Demo page
- Reset password page
- Personal page
- Settings page
- Profile page

The structure of this website is showing as following:



Picture 1.2: Website structure

1.3. Process of designing layout

1.3.1. Guides and size

The width web page size is 960 pixels.

1.3.2. Basic content of web page

- Home page

Home page has navigation bar, the login/register section, main feature section, new feature section, demo section, information section.

- Personal page

Personal page has profile section, navigation bar, individual and group finance functional section.

1.4. Wire Frame Sketch

- Home page

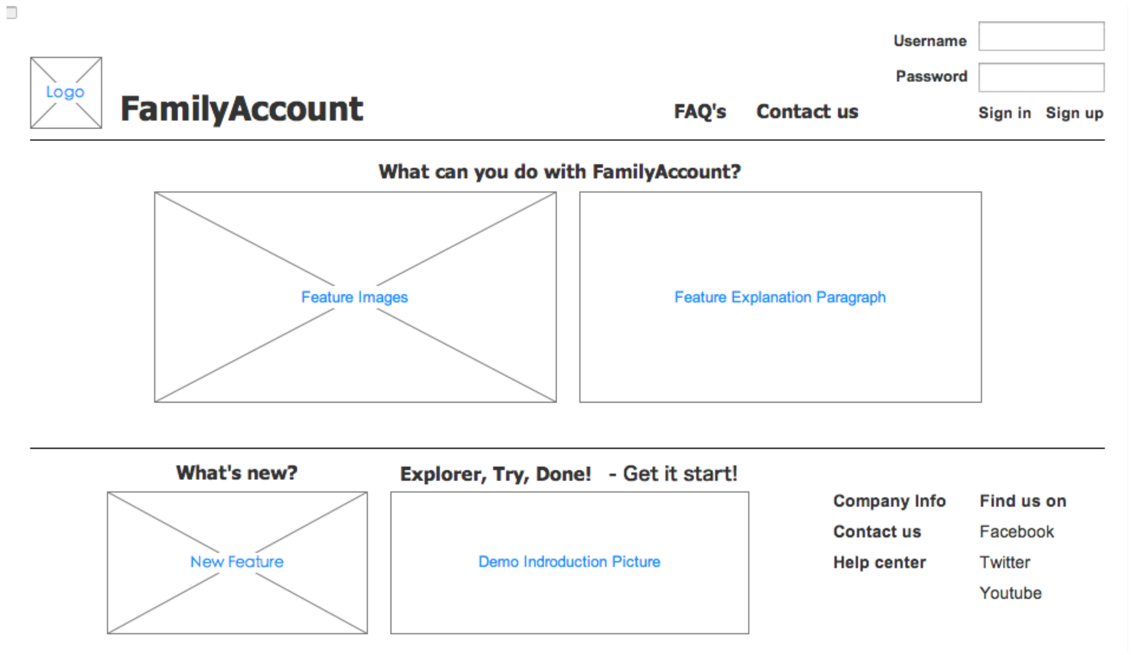
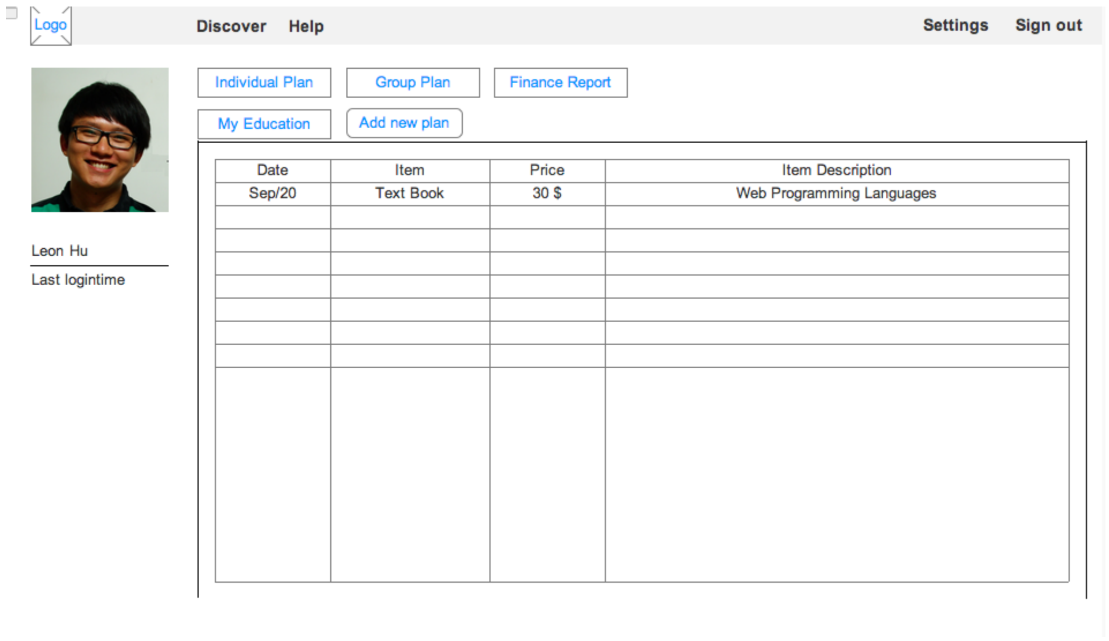


Figure 1.3 Home page wire frame

- Personal page



Date	Item	Price	Item Description
Sep/20	Text Book	30 $	Web Programming Languages

Figure 1.4 Personal page wire frame

2. Source control and repository policy

2.1. Repository tool and server

- Tool Name: Git
- Tool Version: 1.8.4 or later
- Remote Git Server: https://github.com/hulinghuan/Web-Programming-Languages.git

2.2. Repository commit strategy

Each commit should contain clear and helpful commit message.
Each commit message should have:

1) commit version, include:
   a. main version number
   b. function version number
2) specific commit description, include:
   c. commit task type: working on/add/delete/modify file/function
   d. object name

   e. task description: the specific description about what’s the goal or purpose of the task and the abstract description about what have been done in this commit.
3) name of the person who did the commit

2.3. Repository branch strategy

- **Branch name format:**

branch type + task name.

- **Branch type:**

the type of each branch should be one of following:
1) master
2) function
3) hotfix

- **Task naming strategy**

The name of tasks should be the one of the subsystem name.

- **Branch merge strategy**

Master branch can be merged with other branches when and only when the other branch pass the black box test and has been proved it is stable.

3. Coding standard

3.1. C#

3.1.1. Clarity and consistency

Do ensure that clarity, readability and transparency are paramount. These coding standards strive to ensure that the resultant code is easy to understand and maintain, but nothing beats fundamentally clear, concise, self-documenting code.

3.1.2. Formatting and style

- **Tabs**

Do not use tabs. It's generally accepted across Microsoft that tabs shouldn't be used in source files - different text editors use different spacing to render tabs, and this causes formatting confusion. All code should be written using four spaces for indentation. Visual Studio text editor can be configured to insert spaces for tabs.

- **Length of line**

The length of lines of code should be 86 column.

- **Font**

Use a fixed-width font, typically Courier New, in your code editor.

3.1.3. Libraries

Do not reference unnecessary libraries, include unnecessary header files, or reference unnecessary assemblies.

3.1.4. Variable Declarations and Initializations

Do declare local variables in the minimum scope block that can contain them, typically just before use if the language allows; otherwise, at the top of that scope block.
Initialize variables when they are declared.
Do not declare multiple variables in a single line. One declaration per line is recommended since it encourages commenting, and could avoid confusion.

3.1.5. Statements

Use an enum to strongly type parameters, properties, and return values that represent sets of values.

3.1.6. Whitespace

Use blank lines to separate groups of related statements. Omit extra blank lines that do not make the code easier to read. For example, you can have a blank line between variable declarations and code.

3.1.7. Spaces

Spaces improve readability by decreasing code density. Here are some guidelines for the use of space characters within code:

```
CreateFoo(); // No space between function name and parenthesis
Method(myChar, 0, 1); // Single space after a comma
x = array[index]; // No spaces inside brackets
while (x == y) // Single space before flow control statements
if (x == y) // Single space separates operators
' VB.NET sample:
CreateFoo() ' No space between function name and parenthesis
Method(myChar, 0, 1) ' Single space after a comma
x = array(index) ' No spaces inside brackets
While (x = y) ' Single space before flow control statements
If (x = y) Then ' Single space separates operators
```

3.1.8. Braces

Use the brace rule as following format:

```
if (x > 5) {
y = 0;
}
```

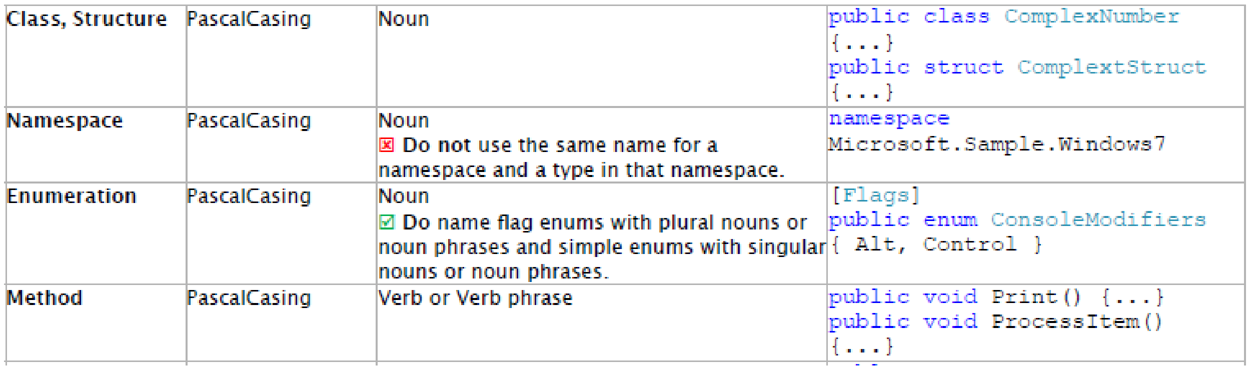
3.1.9. Comments

Use comments that sumCmarize what a piece of code is designed to do and why.
Do not use comments to repeat the code.
Use '//' comments instead of '/* */' for comments for C++ and C# code comments.
The single-line syntax (// …) is preferred even when a comment spans multiple lines.
Indent comments at the same level as the code they describe.
Use full sentences with initial caps, a terminating period and proper punctuation and spelling in comments.

3.1.10. Inline Code Comments

Inline comments should be included on their own line and should be indented at the same level as the code they are commenting on, with a blank line before, but none after. Comments describing a block of code should appear on a line by themselves, indented as the code they describe, with one blank line before it and one blank line after it.

3.1.11. Capitalization Naming Rules for Identifiers



Class, Structure	PascalCasing	Noun	`public class ComplexNumber {...}` `public struct ComplextStruct {...}`
Namespace	PascalCasing	Noun ✗ Do not use the same name for a namespace and a type in that namespace.	`namespace Microsoft.Sample.Windows7`
Enumeration	PascalCasing	Noun ✓ Do name flag enums with plural nouns or noun phrases and simple enums with singular nouns or noun phrases.	`[Flags]` `public enum ConsoleModifiers { Alt, Control }`
Method	PascalCasing	Verb or Verb phrase	`public void Print() {...}` `public void ProcessItem() {...}`

Public Property	PascalCasing	Noun or Adjective
		☑ Do name collection proprieties with a plural phrase describing the items in the collection, as opposed to a singular phrase followed by “List” or “Collection”.
		☑ Do name Boolean proprieties with an affirmative phrase (CanSeek instead of CantSeek). Optionally, you can also prefix Boolean properties with “Is,” “Can,” or “Has” but only where it adds value.
			`public string CustomerName`
			`public ItemCollection Items`
			`public bool CanRead`

Non-public Field	camelCasing or _camelCasing	Noun or Adjective.
		☑ Do be consistent in a code sample when you use the '_' prefix.
			`private string name;`
			`private string _name;`

Event	PascalCasing	Verb or Verb phrase
		☑ Do give events names with a concept of before and after, using the present and past tense.
		☒ Do not use “Before” or “After” prefixes or postfixes to indicate pre and post events.
			`// A close event that is raised after the window is closed.`
			`public event WindowClosed`
			
			`// A close event that is raised before a window is closed.`
			`public event WindowClosing`

Delegate	PascalCasing	☑ Do add the suffix ‘EventHandler’ to names of delegates that are used in events. ☑ Do add the suffix ‘Callback’ to names of delegates other than those used as event handlers. ☒ Do not add the suffix “Delegate” to a delegate.	`public delegate WindowClosedEventHandler`
Interface	PascalCasing ‘I’ prefix	Noun	`public interface IDictionary`

Constant	PascalCasing for publicly visible;
	camelCasing for internally visible;
	All capital only for abbreviation of one or two chars long.
		Noun	`public const string MessageText = "A";`
			`private const string messageText = "B";`
			`public const double PI = 3.14159...;`

Parameter, Variable	camelCasing	Noun	`int customerID;`

Generic Type Parameter	PascalCasing ‘T’ prefix	Noun ☑ Do name generic type parameters with descriptive names, unless a single-letter name is completely self-explanatory and a descriptive name would not add value. ☑ Do prefix descriptive type parameter names with T. ☑ **You should** using T as the type parameter name for types with one single-letter type parameter.	`T, TItem, TPolicy`
Resource	PascalCasing	Noun ☑ Do provide descriptive rather than short identifiers. Keep them concise where possible, but do not sacrifice readability for space. ☑ Do use only alphanumeric characters and underscores in naming resources.	`ArgumentExceptionInvalidName`

3.2. HTML

3.2.1. Formatting

All HTML documents must use two spaces for indentation and there should be no trailing whitespace. XHTML syntax must be used (this is more a Genshi requirement) and all attributes must use double quotes around attributes.

```
<!-- XHTML boolean attributes must still have values and self closing tags must have a closing / -->

<video autoplay="autoplay" poster="poster_image.jpg">

  <source src="foo.ogg" type="video/ogg" />

</video>
```

HTML5 elements should be used where appropriate
reserving <div> and <span> elements for situations where there is no semantic value (such as wrapping elements to provide styling hooks).

3.2.2. Doctype and layout

All documents shall use the HTML 4.01 doctype and the <html> element should have the "lang" attribute. The <head> should also at a minimum include "viewport" and "charset" meta tags.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"

"http://www.w3.org/TR/html4/loose.dtd">
```

3.2.3. Forms

Form fields mCust always include a <label> element with a "for" attribute matching the "id" on the input. This helps accessibility by focusing the input when the label is clicked, it also helps screen readers match labels to their respective inputs.

```
<label for="field-email">email</label>

<input type="email" id="field-email" name="email" value="" />
```

Each <input> should have an "id" that is unique to the page. It does not have to match the "name" attribute.

3.2.4. Including meta data

Ideally, classes should only be used as styling hooks. If you need to include additional data in the html document, for example to pass data to JavaScript, then the HTML5 data- attributes should be used.

```
<a class="btn" data-format="csv">Download CSV</a>
```

3.2.5. Line breaks

Do not include line breaks within <p> blocks.

Do like this:

```
<p>Blah foo blah</p>

<p>New paragraph, blah</p>
```

Do not do like this:

```
<p>Blah foo blah

   New paragraph, blah</p>
```

3.3. CSS

3.3.1. Structure

There are plenty of different methods for structuring a stylesheet. With the CSS in core, it is important to retain a high degree of legibility. This enables subsequent contributors to have a clear understanding of the flow of the document.

- Use tabs, not spaces, to indent each property.
- Add two blank lines between sections and one blank line between blocks in a section.
- Each selector should be on its own line, ending in either a comma or an opening curly brace. Property-value pairs should be on their own line, with one tab of indentation and an ending semicolon. The closing brace should be flush left, using the same level of indentation as the opening selector.

Correct:

```
#selector-1,
#selector-2,
#selector-3 {
    background: #fff;
    color: #000;
}
```

Incorrect:

```
#selector-1, #selector-2, #selector-3 {
    background: #fff;
    color: #000;
    }

#selector-1 { background: #fff; color: #000; }
```

3.3.2. Selectors

With specificity, comes great responsibility. Broad selectors allow us to be efficient, yet can have adverse consequences if not tested. Location-specific selectors can save us time, but will quickly lead to a cluttered stylesheet. Exercise your best judgement to create selectors that find the right balance between contributing to the overall style and layout of the DOM.

- Similar to the WordPress Coding Standards for file names, use lowercase and separate words with hyphens when naming selectors. Avoid camelcase and underscores.
- Use human readable selectors that describe what element(s) they style.
- Attribute selectors should use double quotes around values
- Refrain from using over-qualified selectors, div.container can simply be stated as .container

Correct:

```
#comment-form {
    margin: 1em 0;
}

input[type="text"] {
    line-height: 1.1;
}
```

Incorrect:

```
#commentForm { /* Avoid camelcase. */
    margin: 0;
}

#comment_form { /* Avoid underscores. */
    margin: 0;
}
```

```
div#comment_form { /* Avoid over-qualification. */
    margin: 0;
}

#c1-xr { /* What is a c1-xr?! Use a better name. */
    margin: 0;
}

input[type=text] { /* Should be [type="text"] */
    line-height: 110% /* Also doubly incorrect */
}
```

3.3.3. Properties

Similar to selectors, properties that are too specific will hinder the flexibility of the design. Less is more. Make sure you are not repeating styling or introducing fixed dimensions (when a fluid solution is more acceptable).

- Properties should be followed by a colon and a space.
- All properties and values should be lowercase, except for font names and vendor-specific properties.
- Use hex code for colors, or rgba() if opacity is needed. Avoid RGB format and uppercase, and shorten values when possible: #fff instead of #FFFFFF.
- Use shorthand (except when overriding styles) for background, border, font, list-style, margin, and padding values as much as possible. (For a shorthand reference, see CSS Shorthand.)

3.3.4. Property Ordering

Above all else, choose something that is meaningful to you and semantic in some way. Random ordering is chaos, not poetry. In WordPress Core, our choice is logical or grouped ordering, wherein properties are grouped by meaning and ordered specifically within those groups. The properties within groups are also strategically ordered to create transitions between sections, such as background directly before color. The baseline for ordering is:

- Display
- Positioning
- Box model
- Colors and Typography
- Other

Things that are not yet used in core itself, such as CSS3 animations, may not have a prescribed place above but likely would fit into one of the above in a logical manner. Just as CSS is evolving, so our standards will evolve with it.

Top/Right/Bottom/Left (TRBL/trouble) should be the order for any relevant properties (e.g. margin), much as the order goes in values. Corner specifiers (e.g. border-radius-*-*) should be top-left, top-right, bottom-right, bottom-left. This is derived from how shorthand values would be ordered.

Example:

```
#overlay {
    position: absolute;
    z-index: 1;
    padding: 10px;
    background: #fff;
    color: #777;
}
```

Another method that is often used, including by the Automattic/WordPress.com Themes Team, is to order properties alphabetically, with or without certain exceptions.

Example:

```
#overlay {
    background: #fff;
    color: #777;
    padding: 10px;
    position: absolute;
    z-index: 1;
}
```

3.3.5. Vendor Prefixes

Vendor prefixes should go longest (-webkit-) to shortest (unprefixed). Values should be left aligned with spaces after the colon provided that all the values are the same across all prefixes.

Preferred method:

```
.koop-shiny {
    -webkit-box-shadow: inset 0 0 1px 1px #eee;
    -moz-box-shadow:    inset 0 0 1px 1px #eee;
    box-shadow:         inset 0 0 1px 1px #eee;
    -webkit-transition: border-color 0.1s;
    -moz-transition:    border-color 0.1s;
    -ms-transition:     border-color 0.1s;
    -o-transition:      border-color 0.1s;
    transition:         border-color 0.1s;
}
```

Not preferred:

```
.okay {
    -webkit-box-shadow: inset 0 0 1px 1px #eee;
       -moz-box-shadow: inset 0 0 1px 1px #eee;
            box-shadow: inset 0 0 1px 1px #eee;
}

.bad {
    -webkit-box-shadow: inset 0 0 1px 1px #eee;
    -moz-box-shadow: inset 0 0 1px 1px #eee;
    box-shadow: inset 0 0 1px 1px #eee;
}
```

Special case for CSS gradients:

```
.gradient {
    background: #fff;
    background-image: -webkit-gradient(linear, left bottom, left top, from(#fff), 
to(#000));
    background-image: -webkit-linear-gradient(bottom, #fff, #000);
    background-image:    -moz-linear-gradient(bottom, #fff, #000);
    background-image:      -o-linear-gradient(bottom, #fff, #000);
    background-image: linear-gradient(to top, #fff, #000);
}
```

3.3.6. ValCues

There are numerous ways to input values for properties. Follow the guidelines below to help us retain a high degree of consistency.

- Space before the value, after the colon
- Do not pad parentheses with spaces
- Always end in a semicolon
- Use double quotes rather than single quotes, and only when needed, such as when a font name has a space.
- 0 values should not have units unless necessary, such as with transition-duration.
- Line height should also be unit-less, unless necessary to be defined as a specific pixel value. This is more than just a style convention, but is worth mentioning here. More information: http://meyerweb.com/eric/thoughts/2006/02/08/unitless-line-heights/
- Use a leading zero for decimal values, including in rgba().
- Multiple comma-separated values for one property should be separated by either a space or a newline, including within rgba(). Newlines should be used for lengthier multi-part values such as those for shorthand properties like box-shadow and text-shadow. Each subsequent value after the first should then be on a new line, indented to the same level as the selector and then spaced over to left-align with the previous value.

Correct:

```
.class { /* Correct usage of quotes */
    background-image: url(images/bg.png);
    font-family: "Helvetica Neue", sans-serif;
}

.class { /* Correct usage of zero values */
    font-family: Georgia, serif;
    text-shadow: 0 -1px 0 rgba(0, 0, 0, 0.5),
                       0 1px 0 #fff;
}
```

Incorrect:

```
.class { /* Avoid missing space and semicolon */
    background:#fff
}

.class { /* Avoid adding a unit on a zero value */
    margin: 0px 0px 20px 0px;
}
```

3.3.7. Media Query

Media queries allow us to gracefully degrade the DOM for different screen sizes. If you are adding any, be sure to test above and below the break-point you are targeting.

It is generally advisable to keep media queries grouped by media at the bottom of the stylesheet.

- An exception is made for the wp-admin.css file in core, as it is very large and each section essentially represents a stylesheet of its own. Media queries are therefore added at the bottom of sections as applicable.

Rule sets for media queries should be indented one level in.

Example:

```
@media all and (max-width: 699px) and (min-width: 520px) {

        /* Your selectors */
}
```

3.3.8. Commenting

Comment, and comment liberally. If there are concerns about file size, utilize minified files and the SCRIPT_DEBUG constant. Long comments should manually break the line length at 80 characters.

A table of contents should be utilized for longer stylesheets, especially those that are highly sectioned. Using an index number (1.0, 1.1, 2.0, etc.) aids in searching and jumping to a location.

Comments should be formatted much as PHPDoc is. The CSSDoc standard is not necessarily widely accepted or used but some aspects of it may be adopted over time. Section/subsection headers should have newlines before and after. Inline comments should not have empty newlines separating the comment from the item to which it relates.

For sections and subsections:

```
/**
* #.# Section title
*
* Description of section, whether or not it has media queries, etc.
*/

.selector {
    float: left;
}
```

For inline:

```
/* This is a comment about this selector */

.another-selector {
    position: absolute;
    top: 0 !important; /* I should explain why this is so !important */
}
```