

HTML and CSS tutorial

Rui Couto

José C. Campos

Sistemas Interactivos
Mestrado Integrado em Engenharia Informática
DI/UMinho
March 9, 2020

Contents

1	Introduction	1
2	Base template	1
3	Semantic markup	2
4	Layout structure	4
5	Style	5
6	Responsive Web design	7
7	CSS preprocessors	9
A	GamesLibrary Prototype	10
A.1	Mockups	10
A.1.1	Home page	10
A.1.2	Login window	10
A.1.3	List of all games	11
A.1.4	List of my games	11
A.1.5	Add a game	12
A.1.6	Information on a game	12
A.2	Navigation map	13

1 Introduction

This is the first of a set of tutorials that propose the development of a Web application to manage a library of games: the GamesLibrary application. A prototype of the user interface to be developed is provided in Appendix A. This tutorial, in particular, guides you through the process of creating the List of all games Web page (see page 11), resorting to HTML and CSS.

During the tutorial it will help to use a text editor/IDE with support for Web programming (e.g. Visual Studio Code or Atom). Although good editors will have code completion, it is useful to have HTML and CSS references available. The following are recommended:

HTML reference: <https://www.w3schools.com/tags/>

CSS reference: <https://www.w3schools.com/cssref/>

2 Base template

A base template is provided with this Tutorial. It consists of the following files:

index.html The html page contains all the information to display. Its content is as follows.

```
1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <meta charset="utf-8">
5     <meta name="description" content="">
6     <meta name="author" content="">
7     <link rel="icon" href="favicon.ico">
8     <title>Template</title>
9     <link rel="stylesheet" type="text/css" href="style/reset.css">
10    <link rel="stylesheet" type="text/css" href="style/style.css">
11    <script src="https://code.jquery.com/jquery-3.1.1.min.js"
12      integrity="sha256-hVVnYaiADRTO2PzUGmuLJr8BLUSjGIZsDYGmIJLv2b8="
13      crossorigin="anonymous"></script>
14  </head>
15
16  <body>
17
18  </body>
19  <script src="script/script.js" ></script>
20 </html>
```

Line 1 provides the browser information regarding the file type (identified this as an HTML5 file). Line 2 starts the document itself. Lines 4 to 6 provide meta

information. Line 9 includes the `reset.css` file, and line 10 the custom `style.css` file. Line 11 includes the jQuery library (to be used later). Lines 16 to 18 is where your HTML contents should go, and line 19 includes the custom scripts for this application (again, to be used later). Line 20 closes the HTML content.

reset.css A common practice in web development is to include a `reset.css`¹ file. This file will clear default styling values set by the browsers (which differ from browser to browser), and allow the developers to start from an “empty canvas”. As result, the HTML will be rendered as specified by the developer. The downside is that we are taking control away from the user.

style.css Custom styles should be placed in this file. This includes all styles and page structure customisation. Typically, CSS files are imported in the `head` section.

script.js The definition of custom Javascript functions should be placed in this file. This includes not only auxiliary functions, but also callbacks. Since jQuery was imported **before** this file, it is possible to use jQuery functions in this file. **All custom scripts should be placed at the end of the page, in order to ensure that DOM elements have already been loaded when they are invoked.**

Needless to say, multiple scripts (js) and styles (css) files might be added as needed. It is recommended to keep a **well organised structure** for the files. In the provided template, Javascript files are placed inside the `scripts` folder, and CSS files inside the `style` folder.

Tasks:

1. Extract the template, and open `index.html` in the browser.
2. Add the table with games “Game 1” and “Game 2” (cf. List of all games mockup) to the page (inside the `<body>` tag), and refresh the browser page to see the results.

Lookup the `<table>`, `<thead>`, `<tr>`, `<th>` and `<td>` tags in the reference, if you are not familiar with them.

3 Semantic markup

The HTML pages should be free of any style and behaviour. Its objective is to declare the **content** to be shown. Structuring the HTML contents is a way that highlights its semantic value (what the elements *mean*) is relevant for later adding structure and presentation style. A typical approach is to have elements for:

¹See <https://github.com/jasonkarns/css-reset> for more informations on the used `reset.css`.

- Application name.
- Navigation bar.
- Main content.
- Side/additional content.
- Footer.

HTML5 encourages semantic markup and provides tags for the elements above. A possible structure for a page is then:

```

1 <!DOCTYPE html>
2 <html lang="en">
3   <head>...</head>
4   <body>
5     <header>
6       logo
7     </header>
8     <nav>
9       navbar
10    </nav>
11    <aside>
12      sidebar
13    </aside>
14    <main>
15      main contents
16    </main>
17    <footer>
18      footer
19    </footer>
20  </body>
21  <script src="script/script.js" />
22 </html>

```

As expected, opening the file in the browser presents a page which lists its contents sequentially (cf. Figure 1). The layout will be added later via CSS.

Tasks:

1. Update the page you created to reflect the structure above, and fill it with content according to the mockup (i.e. names, links, etc.). Use a `<form>` with `<select>` elements for the dropdown menus.
Lookup the `<form>` tag in the reference, for information on HTML forms.
2. Open the page in the browser to ensure that all content is being shown (see Figure 2 for the expected result – note that the pagination links have not been added yet).

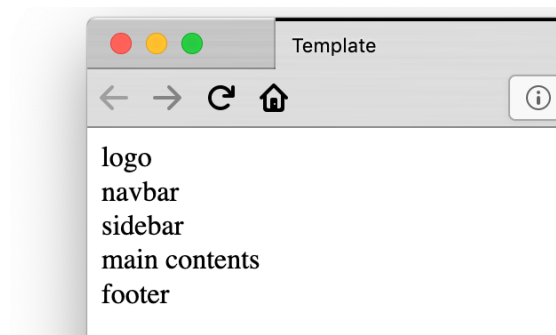


Figure 1: Version 1 of the page.

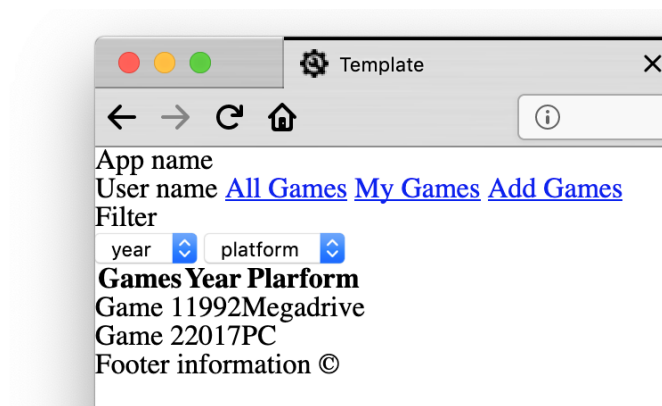


Figure 2: Version 2 of the page.

4 Layout structure

After adding all the page content, it is now possible to proceed and add style and layout structure to the page. A common practice is to highlight each page component with a different color, in order to clearly see them when defining layout. In the `style.css` file, add a different color for each relevant component. For instance, the header can be marked with the red color:

```
1 header {background-color: red;}
```

Tasks:

1. Add colors to the remaining main elements (`nav`, `aside`, `main` and `footer`)

The final result should be a page in which each element is clearly identified by a color (see Figure 3).

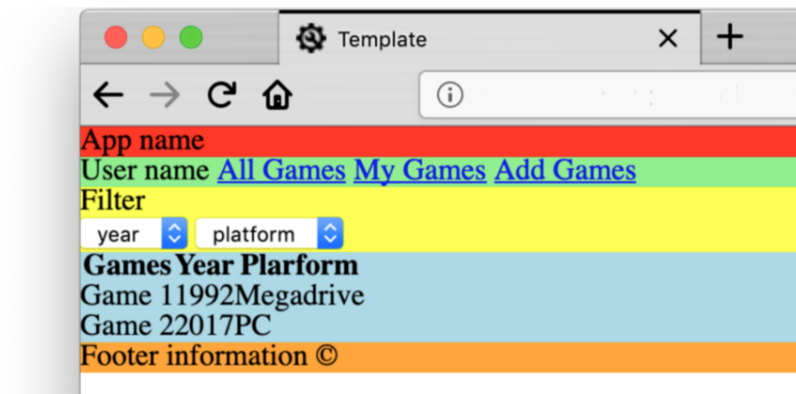


Figure 3: Version 3 of the page.

Now, layout structure can be added. In this example, the sidebar should be placed at the right side. That can be achieved using the `float` property.

Two notes to take into account:

- For `float` to work as desired, the sidebar should be defined (in the HTML) **before** the main content;
- Increasing the height of the sidebar (e.g. `height: 10em2`) will make the sidebar overlap the footer. In order to avoid this, `clear: both` should be used in the footer.

Tasks:

1. In the CSS, move the sidebar to the right, resorting to the `float` property. Also, specify the widths for the table container and for the sidebar (e.g. 80%, 20%).
2. Specify the sizes for the main content and for the sidebar.
3. Increase the height of the side bar to 10em and fix the footer overlap issue.

The expected result is a page similar to Figure 4.

5 Style

At this stage, the layout structure is defined. Now, the style can be added, according to the mockup. Note that you might need to add additional tags (e.g. `<div>` or `` tags) to group relevant elements together in order to style them.

²em is a measure unit relative to the character size.

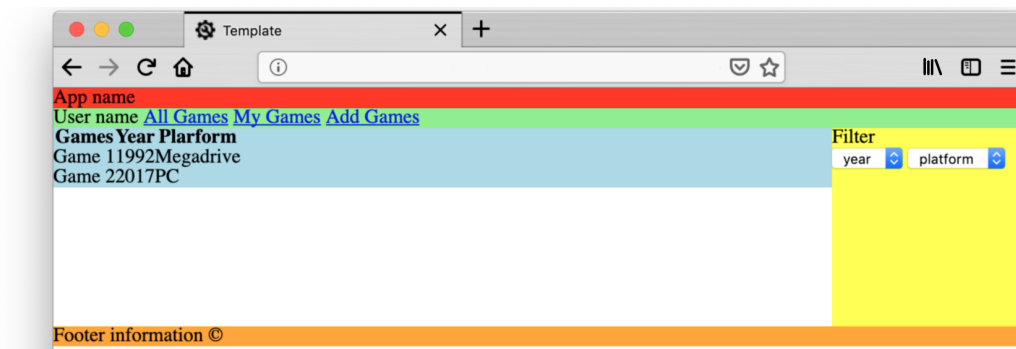


Figure 4: Version 4 of the page.

As much as possible this grouping should reflect the semantics of the elements in the page. For example, you can use `` tags to group the lists of navigation links on the top right and/or the pagination buttons (1 2 3) at the bottom of the table. When tag names are not enough, you should use IDs and classes to allow referring to different HTML elements in the CSS (e.g. to display the list of navigation links horizontally). You can also make use of predefined classes to identify, for instance, active elements (in this case, with `:hover`).

See the list of properties available at <https://www.w3schools.com/cssref/>, and remember the selector syntax:

Selecting elements by tag name To select all elements with a given tag, simply specify the tag name. Example for buttons:

HTML `<button>Click me</button>`

CSS `button {background-color: red; }`

Selecting an element by its ID To select an element by its id use the hash (“#”) symbol. Example for the id `myId`:

HTML `<div id="myId">Hello</div>`

CSS `#myId {color:blue;} or div#myId {color:blue;}`

Selecting all the elements of a given class Use the dot (“.”) symbol in order to identify all the elements belonging to a class. Example for the class `myClass`:

HTML `<div class="myClass">Hello</div> World`

CSS `.myClass {color:green;} or div.myClass, span.myClass {color:green;}`

Nested attributes In order to specify an element, which is nested inside others, the attribute should be placed sequentially. Example:

HTML

`<div id="myDiv"> <button>Click me</button></div>`

CSS For any button, inside a span, inside a div, the corresponding CSS is `div span button {border:2px solid black;}`

For any button inside a span with the class `myClass`, inside a div with the id `myDiv`:

```
div#myDiv span.myClass button {border:2px solid black;}
```

Tasks:

1. Change the font and size of relevant elements to make it similar to the proposed prototype — the "Verdana" and "Helvetica Neue" font families are fairly similar to the mockup.
2. Add the correct **padding, margins, borders and alignment** formatting to make the layout closer to the prototype. Use flexbox layouts to make the name and the links appear at opposite sides of the header. Do the same to make the pagination buttons appear at the bottom.
3. Add the correct colours to the elements.

The expected result is a page similar to the mockup, as presented in Figure 5.

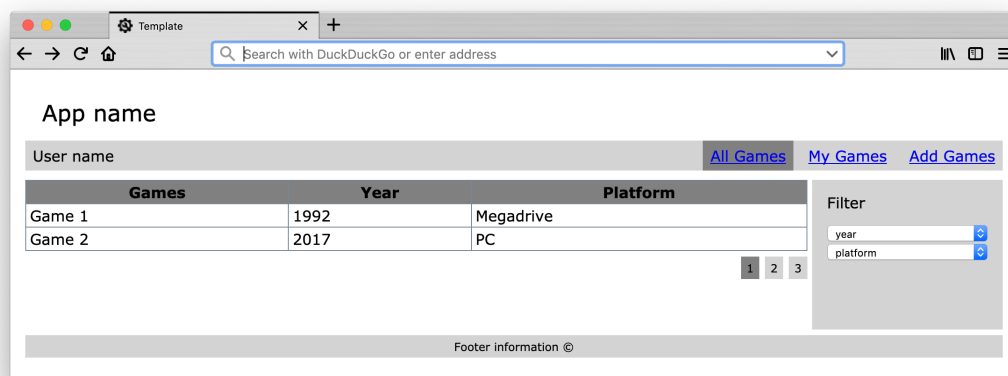


Figure 5: Version 5 of the page.

6 Responsive Web design

Media queries support the specification of different styles according to the details of the presentation medium (e.g. depending on the size of the screen of browser window).

See the description of the `@media` rule available at <https://www.w3schools.com/cssref/>. In short, CSS rules can be aggregated inside `@media` rules, to restrict their application to defined media types (e.g. print, screen, tv) and sizes, according to the following syntax:

```
1 @media mediatypedef and (mediafeaturedef) {  
2     CSS rules  
3 }
```

For example, the CSS code:

```
1 @media (max-width: 800px) {  
2     aside {  
3         background-color: red;  
4     }  
5 }
```

makes the background of the `<aside>` tag red, for screen sizes smaller than 800 pixels in all media types (no media type is being specified, which defaults to `all`).

Tasks:

1. Change the stylesheet you created to change to a fully vertical layout for screens less than 800 pixels wide. The result should be similar to Figure 6
2. Further adapt your stylesheet/HTML so that for screens less than 600 pixels the user name and the year column are removed, and the select-boxes are laid out vertically.

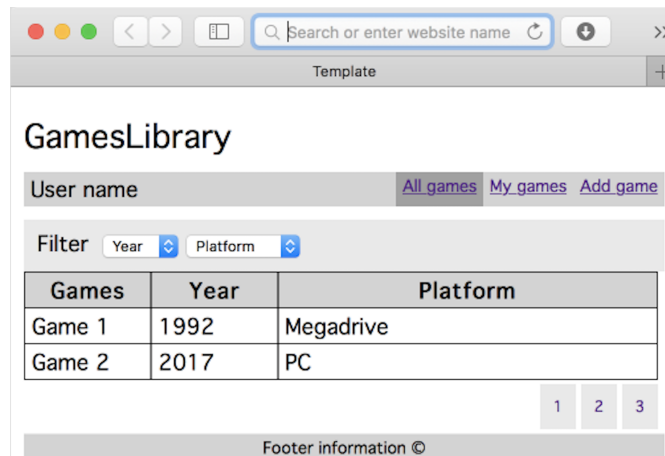


Figure 6: Version 6 of the page.

7 CSS preprocessors

One problem with the CSS code you just wrote is that it is static and not easily reusable. For example, you probably used the same margin width in several places and it would be nice to be able to define a variable so that changing the margin could be done in one single place.

CSS preprocessors allow for that (and more!). We suggest you give SASS a try (<http://sass-lang.com>). SASS supports defining not only variables but also functions (`@mixins`), the nesting of rules and inheritance between rules. CSS files become more structured and compact, hence easier to maintain.

Tasks:

1. Start by installing SASS (see the Web site).
2. Create a `sass` folder and move your `style.css` file into it changing the extension to `.scss`. You will now edit the `.scss` file, not the `.css` (which will be generated by Sass). In order for the `.css` file to be generated run the command:

```
$sass --watch sass:style
```

now, the `.scss` files in the `scss` folder will be automatically compiled into the `style` folder.

3. In the `.scss` file, define variables for values that are used repeatedly in the CCS rules. E.g, you are probably using the same margin/padding value in several places, so you might right:

```
1 $normal-padding: 1em;  
2 $mid-padding: 0.5em;  
3 $small-paddding: 0.25em;
```

and use those variables in the CSS rules.

4. Organise your rules (including media queries) through nesting.
5. Look for sets of properties that are equally defined across multiple rules (for example, setting and configuring flex layout) and use a `@mixin` to encapsulate those properties. Use parameters to make the `@mixin` more generic (e.g. the layout direction can be a parameter).
6. Look for rules that can be considered as extending other rules (i.e. equal to them but with a few more properties defined) and use `@extend` to define them (e.g. is the difference between the rules for the `<td>` and `<th>` tags in the table the fact that the latter defines a background color?).

A GamesLibrary Prototype

A.1 Mockups

A.1.1 Home page

Title

App name

[Login](#)

All games

My games

Add game

Games	Year	Platform
Game 1	1992	Megadrive
Game 2	2017	PC

123

Filter

year

Platform

Footer information ©

A.1.2 Login window

Title

App name

[Login](#)

All games

My games

Add game

Games	Year
Game 1	1992
Game 2	2017

123

Filter

year

Platform

Login

username

Password

Login

Footer information ©

A.1.3 List of all games

Title

App name

User name

All games

My games

Add game

Games	Year	Platform
Game 1	1992	Megadrive
Game 2	2017	PC

1

2

3

Filter

year

Platform

Footer information ©

A.1.4 List of my games

Title

App name

User name

All games

My games

Add game

Games	Year	Platform
Game 2	2017	PC

Filter

year

Platform

Add new

Footer information ©

A.1.5 Add a game

Title

App name

User name

All games My games Add game

Game added to collection

Title

Text box

Year

Text box

Platform

Combo Box

Picture URL

Text box

Description

Save

Footer information ©

A.1.6 Information on a game

Title

App name

User name

All games My games Add game

Game 1


Year

1991

Description

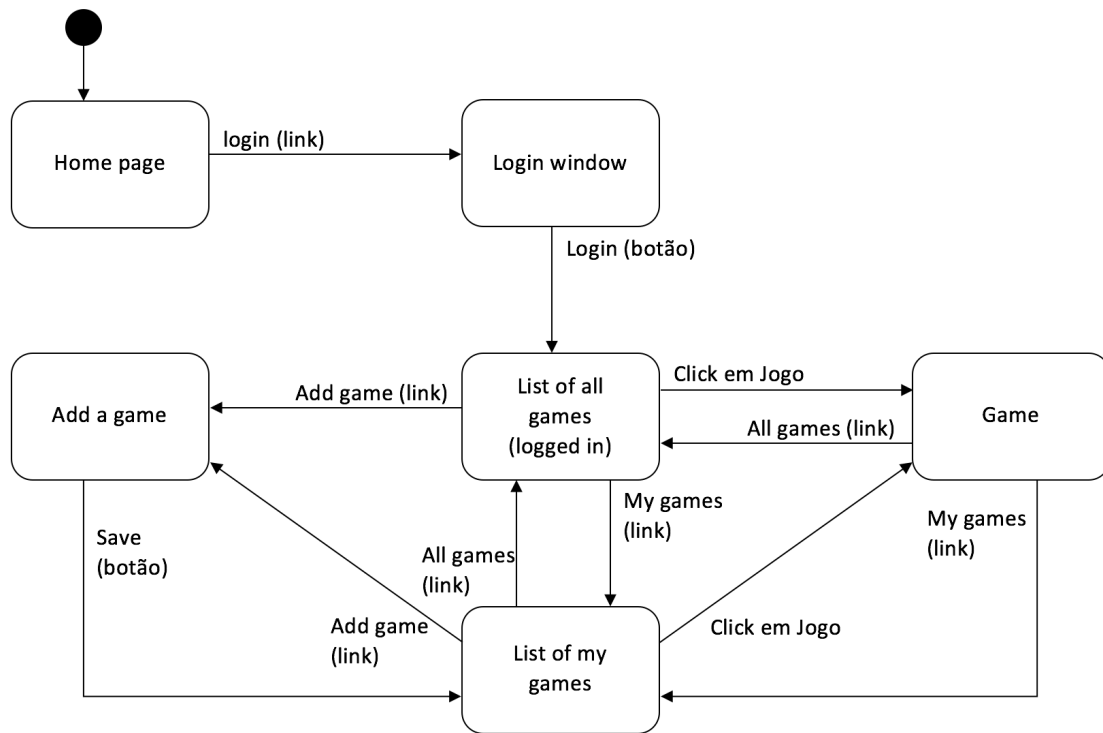
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Quisque bibendum tempus arcu. Morbi convallis lectus feugiat felis fermentum semper. Proin quam nisi, posuere a ante vitae, molestie ornare velit. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Donec aliquam tincidunt ipsum non tincidunt. Sed tincidunt risus facilisis, eleifend nisi quis, euismod tortor. Aenean ac lectus congue, molestie sem eu, faucibus velit. Sed sed erat nec justo dignissim vulputate vel feugiat tortor.

Nunc tincidunt mi ante, id varius neque tempus quis. Vestibulum vel orci malesuada, aliquam dolor non, consequat eros. Suspendisse mi diam, viverra id justo sit amet, suscipit tincidunt tellus. Nunc mollis, mi sit amet pellentesque posuere, elit nulla vulputate dui, sed mollis ipsum nisi in orci. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Cras tempus lorem vitae turpis laculis dictum. Morbi et pellentesque augue. Praesent scelerisque arcu at leo bibendum, eu pharetra justo venenatis. Aenean a vehicula elit, ac laoreet lacus. Phasellus aliquam at elit vitae sollicitudin. Integer id nisi vel dui efficitur molestie in ut tortor. Donec vitae metus turpis. Morbi sed leo erat.



Footer information ©

A.2 Navigation map



Exercise: Complete this navigation map and update the mockups accordingly.