

HTML and CSS tutorial

Rui Couto

José C. Campos

DI/UMinho
April 18, 2023

Contents

1	Introduction	1
2	Base template	1
3	Semantic markup	2
4	Layout structure – CSS Grid	5
5	Style	10
6	Responsive Web design	12
7	CSS preprocessors	14
A	GamesLibrary Prototype	16
A.1	Mockups	16
A.2	Navigation map	19

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 17), 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/style.css">
10    </head>
11    <body> </body>
12    <script src="script/script.js" ></script>
13 </html>
```

Line 1 provides the browser information regarding the file type (identifying this as an HTML5 file). Line 2 starts the document itself. Lines 4 to 6 provide meta information. Line 9 includes the custom `style.css` file (to be used later). Line 12 is where your HTML contents should go, and line 14 includes the custom scripts for this application (again, to be used later). Line 15 closes the HTML content.

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. All custom scripts should be placed at the end of the page. This ensures 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. The end result should be similar to Figure 1.

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:

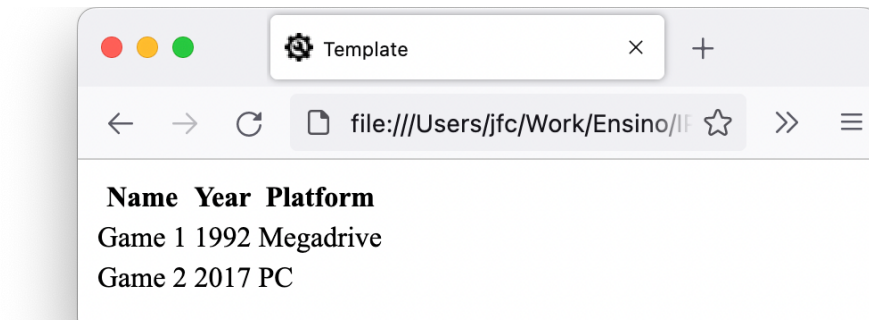


Figure 1: The HTML table being displayed.

- 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    <main>
12      main contents
13    </main>
14    <aside>
15      sidebar
16    </aside>
```

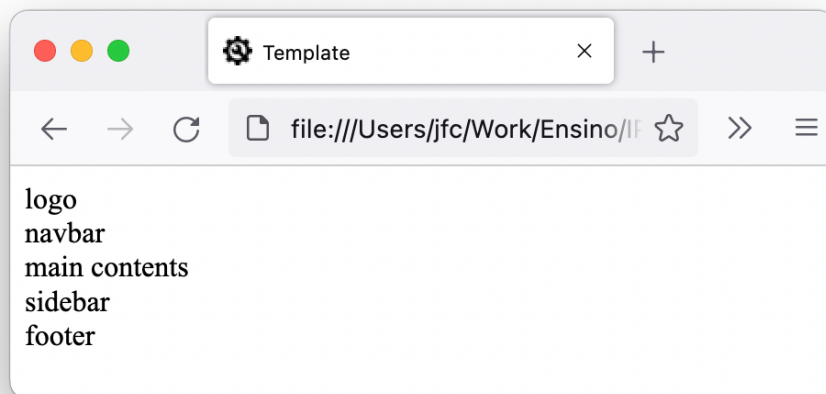


Figure 2: The base structure.

```
17         <footer>
18             footer
19         </footer>
20     </body>
21     <script src="script/script.js" />
22 </html>
```

As expected, opening a file with the above code in the browser presents a page which lists its contents sequentially (cf. Figure 2). The layout will be added later via CSS.

Tasks:

3. 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.

For now, use lists to represent the menu at the top, and the navigation at the bottom¹.

¹The use of lists guarantees some minimum formatting, even if no styling is applied.

4. Open the page in the browser to ensure that all content is being shown (see Figure 3 for the expected result²).

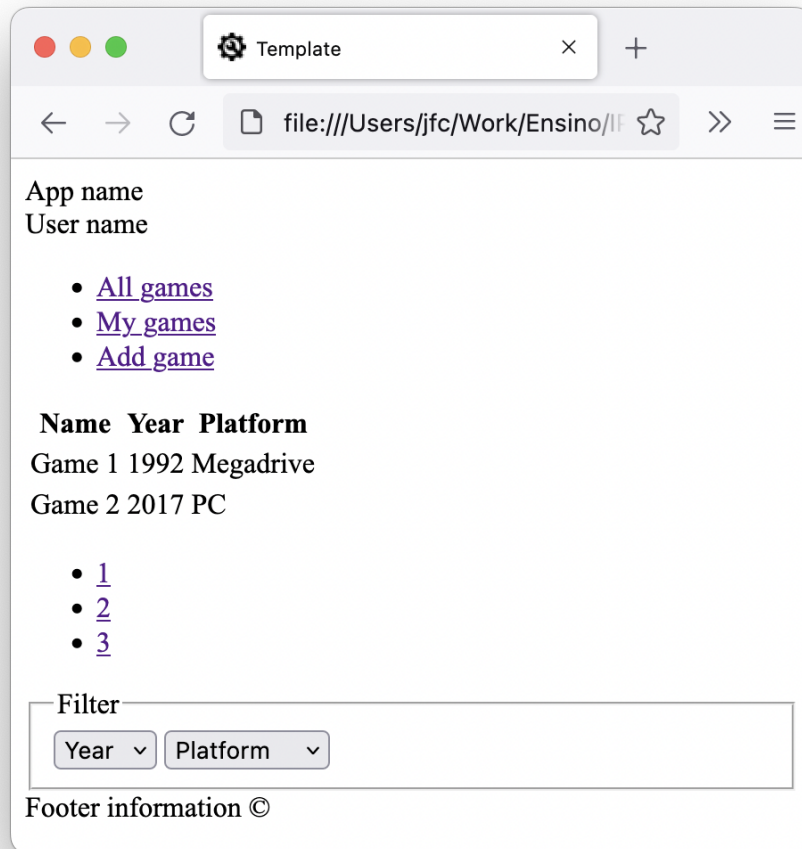


Figure 3: Version 1 of the page.

4 Layout structure – CSS Grid

After adding all the page content, it is now possible to proceed and add style and layout structure to the page. A useful practice is to highlight each page component with a different color, in order to clearly see them when defining

²Your actual result might vary from the example shown, depending on the exact tags you used.

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;}
```

This example illustrates the basic syntax of a CSS expression. A selector, used to identify which elements will be affected, followed by one or more property/-value pairs. In this case the selector is the `header` tag, the property the background colour (`background-color`) and the value `red`. Other selectors include the universal selector: `*` (selects all elements); class selectors: `.c1` (selects all elements with class `c1`); and identifier selectors: `#id` (selects the element with identifier `id`). These selectors can be combined in different ways³.

Tasks:

5. 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 4).

Now, layout structure can be added. We will be using CSS Grid⁴ to achieve the desired layout. To do this, we must have a *container* with the CSS `display` property set to `grid`⁵. We can add a `<div>` element inside `<body>` to act as the grid container, or use `<body>` directly. To be flexible, we add a class `grid-container` to the `<div>/<body>` and we add the following to the CSS:

```
1 .grid-container {  
2     display: grid;  
3 }
```

This will make the contents of page to be organised in a single column, so (except for some spacing) the page will not change from what is displayed in Figure 4.

Next we need to look at the intended layout and define a grid to fit our design. Looking at the mockup we can imagine it as being structured in a 4 line by

³See https://www.w3schools.com/cssref/css_selectors.php.

⁴For an introduction to CSS Grid see: https://www.w3schools.com/css/css_grid.asp

⁵Look also at *grid-inline*

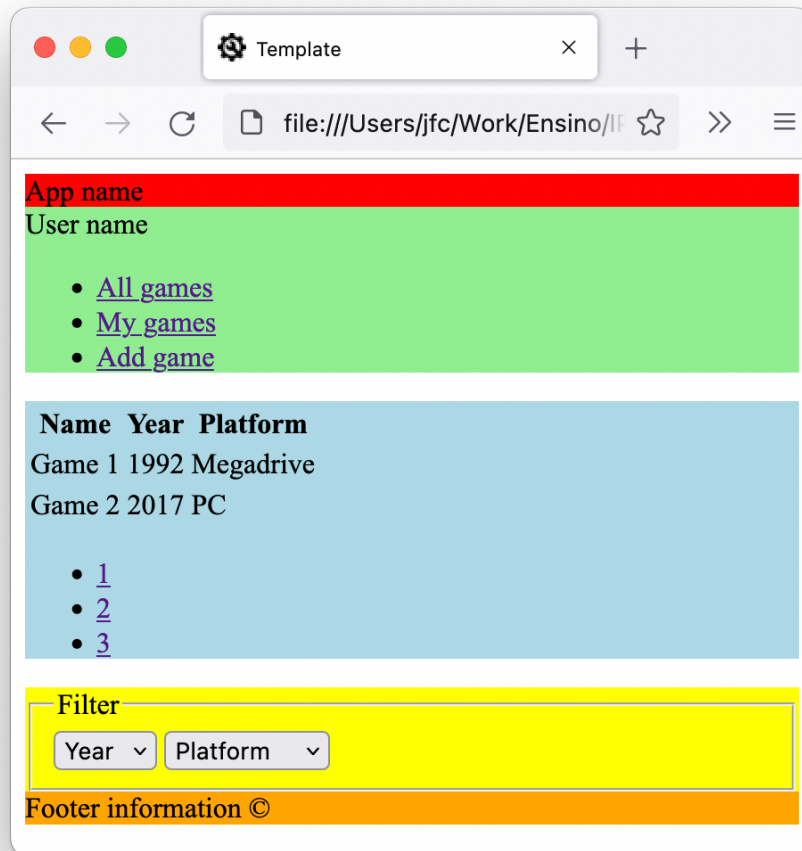


Figure 4: Version 2 of the page.

2 row grid (see Figure 5). Rows 1, 2 and 4 have a single cell, extending over the two columns. Row 3 has two cells, one for each column.

We can start implementing the intended design by using the `grid-template-columns` CSS property:

```

1 .grid-container {
2     display: grid;
3     grid-template-columns: auto 20%;
4 }

```

Which means that the second column will take 20% of the grid width and the

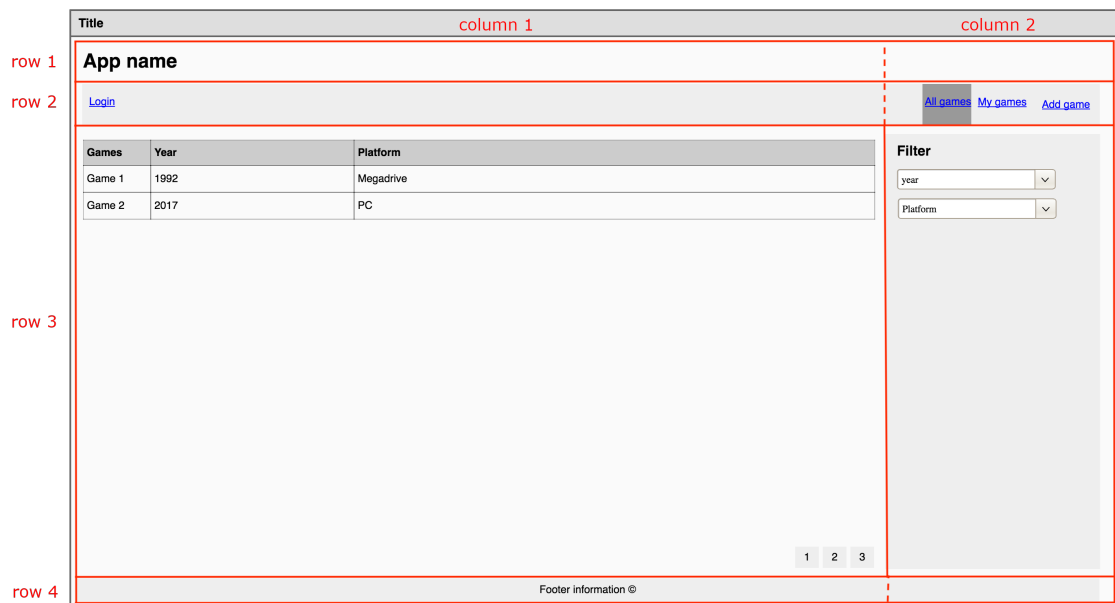


Figure 5: The grid (highlighted in red) to use for the page.

first all the remaining space.

The above, however, will organise the elements in a 4x2 grid (you can try it!). We need to make the `<header>`, `<nav>` and `<footer>` extend the whole row. For that we must define the start and end rows for them. In the case of the header, this can be achieved with the following CSS:

```
1 header {
2   grid-column-start: 1;
3   grid-column-end: 3;
4 }
```

Which means that the header should start at column 1 and end at (before) column 3⁶.

⁶There are alternative, more compact, ways of achieving the same effect: `grid-column: 1 / 3;` defines the start and end column in a single line; `grid-column: 1 / span 2;` defines the start column and how many columns should be used.

Tasks:

6. In the CSS, make the changes described above.
7. Configure, also, the navbar and the footer.

The expected result is a page similar to Figure 6.

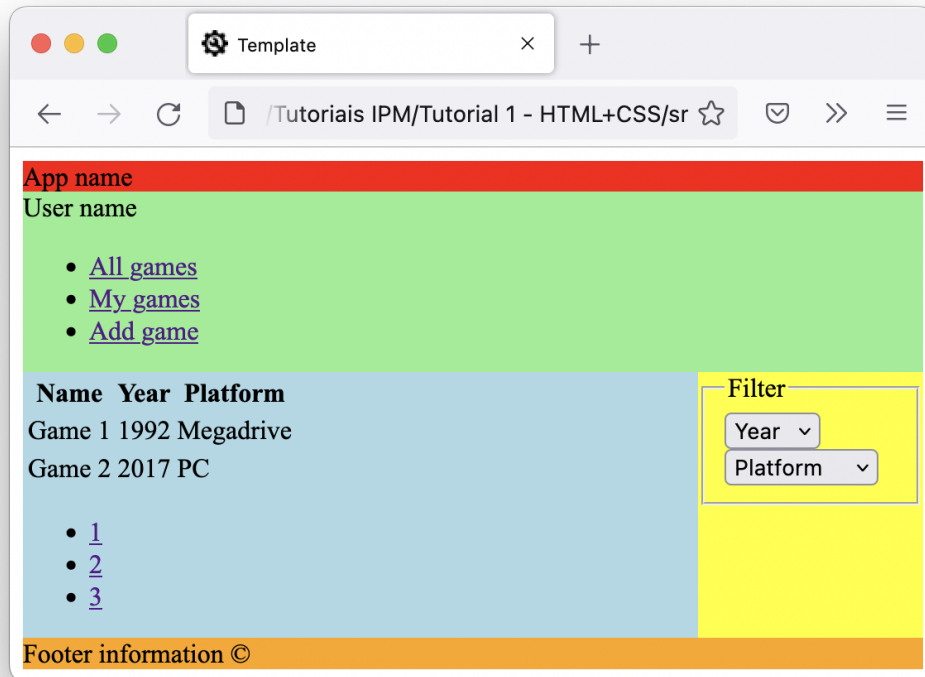


Figure 6: Version 3 of the page.

Note: Explicitly defining the positioning of each element can become hard to maintain for complex layouts. An alternative is to use named areas. In this case, we would define the grid container as:

```
1 .grid-container {  
2     display: grid;  
3     grid-template-columns: auto 25%;  
4     grid-template-areas:
```

```
5     "header header"
6     "nav nav"
7     "content aside"
8     "footer footer";
9 }
```

We have added the `grid-template-areas` to define that the grid element named “header” takes the two columns of the first row, the grid element named navbar the second row, etc.

Now we need to name the elements. In the cases of the navbar and main contents this is achieved thus:

```
1 nav {
2     grid-area: nav;
3 }
4
5 main {
6     grid-area: content;
7 }
```

If you make the equivalent changes to the remaining elements the end result will be the same as before (cf. Figure 6), but we no longer need to refer to concrete columns. This makes the layout more maintainable.

Tasks:

8. Change your CSS to use area names.

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. As much as possible this grouping should reflect the semantics of the elements in the page.

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> <span class="myClass">World</span>
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"> <span class="myClass"><button>Click me</button></span></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:

9. 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.
10. 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 header⁷. Do the same to make the pagination buttons appear at the bottom.
11. Use flex layout, also, to get the menus to display horizontally, and remove the bullet symbols⁸.
12. Add the correct colours to the elements.

The expected result is a page similar to the mockup, as presented in Figure 7.

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/>⁹. 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) {
```

⁷Look for the `flex-direction` and `justify-content` properties.

⁸Look for the `text-decoration` property.

⁹https://www.w3schools.com/cssref/css3_pr_mediaquery.asp

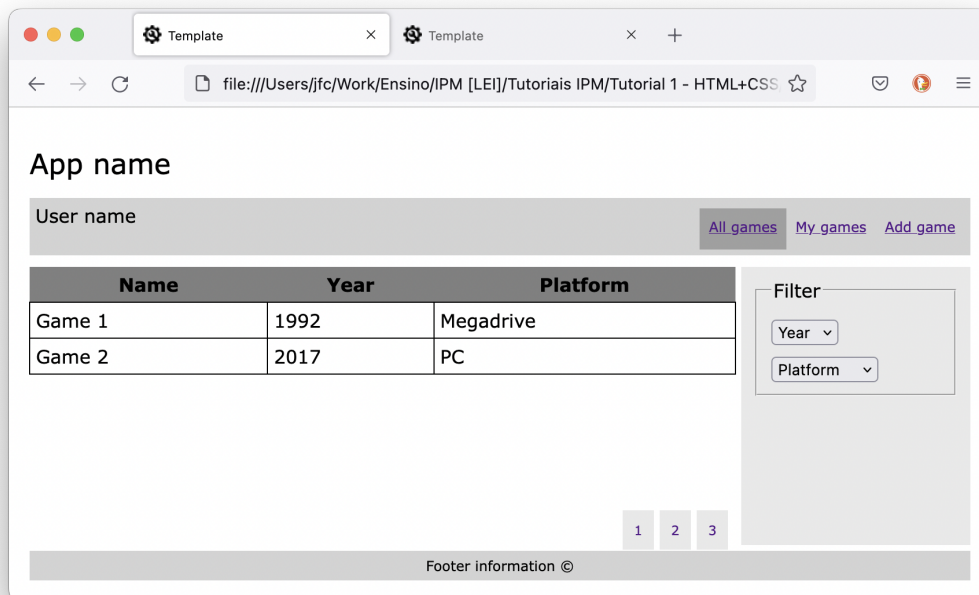


Figure 7: Version 4 of the page.

```

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:

13. 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 8.
14. 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.

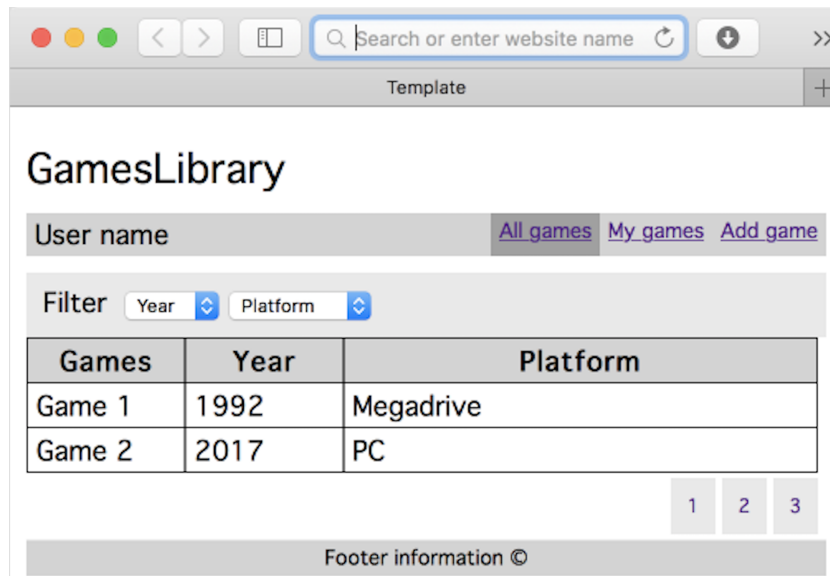


Figure 8: Version 6 of the page.

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:

15. Start by installing SASS (see the Web site).
16. 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.

17. In the `.scss` file, define variables for values that are used repeatedly in the CSS 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.

18. Organise your rules (including media queries) through nesting.
19. 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).
20. 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

Home page

Title

App name

[Login](#)

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 ©

Login window

Title

App name

[Login](#)

All games

My games

Add game

Games	Year
Game 1	1992
Game 2	2017

1

2

3

Filter

year

Platform

Login

username

Password

Login

Footer information ©

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

Filter

year

Platform

1

2

3

Footer information ©

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 ©

Add a game

Title

App name

User name

All gamesMy gamesAdd game

Game added to collection

Title

Text box

Year

Text box

Platform

Combo Box

Picture URL

Text box

Description

Save

Footer information ©

Information on a game

Title

App name

User name

All gamesMy gamesAdd game

Game 1


Year

1991

Description

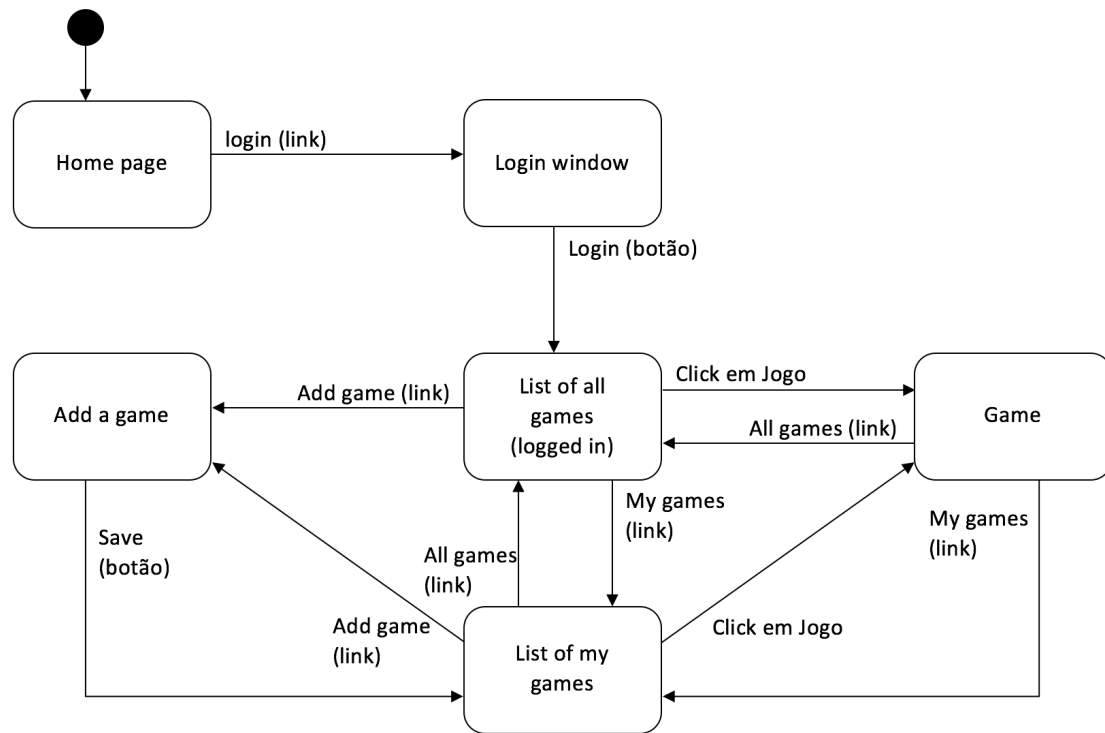
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Quisque bibendum tempus arcu. Morbi convallis lectus feugiat felis fermentum semper. Proin quam nisi, posuere a ante vitae, molestie ornare velit. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Donec 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 iaculis dictum. Morbi et pellentesque augue. Praesent scelerisque arcu at leo bibendum, eu pharetra justo venenatis. Aenean a vehicula elit, ac laoreet iacus. 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.