

PEC 1: Desarrollo de una web

Herramientas HTML y CSS: Universitat Oberta de Catalunya

Ignacio Casares Ruiz

Noviembre de 2022

En este documento se describirán los pasos que se siguieron en la elaboración del proyecto. Se han descrito estos de forma concisa y objetiva, intentando respetar el orden cronológico en el que sucedieron las distintas tomas de decisión y posteriores ejecuciones de las mismas, aunque dando siempre prioridad al orden temático.

La página final es accesible de forma pública a través del siguiente enlace:

COMPLETAR

El proyecto se realizó en un ordenador con un SO Linux Ubuntu 22.04.1 LTS. Para la redacción de la parte teórica se creó un documento en formato [markdown](#) que sería posteriormente transformado en un documento PDF a través de LaTeX con la herramienta de conversión libre [Pandoc](#).

Aspectos generales

Antes de comenzar el aspecto programático del diseño de la web, se consideró el tema en torno al cual sería construido el contenido. El enunciado de la PEC pedía «un pequeño portal biográfico». Para este fin, se eligió al personaje histórico [Tarrare](#).

Con el objetivo de conseguir cierta coherencia visual en cuanto a colores, se decidió utilizar la herramienta [Coolors](#) para establecer una paleta de colores la cual se seguiría a lo largo del proyecto. Se decidió, igualmente, utilizar un estilo sobrio y minimalista, sin excesivo abarrotamiento de la página con el objetivo de evitar tiempos de carga excesivos; se tomaron algunas de las ideas mostradas en [brutalistwebsites.com](#) como inspiración.

El enunciado de la PEC solicitaba que el contenido fuera libre, por lo que se decidió en este punto la utilización de una licencia libre Creative Commons que solo incluyera restricciones a la reproducción de la página para el uso comercial y ante una modificación sustancial. Para este fin, se empleó la licencia [Creative Commons Attribution-NonCommercial-NoDerivs 2.0 Generic](#) (CC BY-NC-ND 2.0).

Posteriormente, se creó un repositorio en git de forma local y [se subió](#) a GitHub. Esto permitiría tener un sistema de control de cambios y un *back-up* almacenado en línea en caso de que se produjera algún problema con los archivos en el curso de la realización del proyecto. Este cambio fue reflejado en el archivo `json`:

```
"repository": {  
  "type": "git",  
  "url": "https://github.com/nacaru-w/PEC1-Herramientas.git"  
},
```

Editor de código

Como gestor de código se decidió utilizar Visual Studio Code (VSCode).

- **Justificación:** la razón por la que se eligió esta aplicación respecto a las otras es porque ya estaba familiarizado con su uso, porque posee una terminal integrada a través de la cual pueden ejecutarse comandos y porque ya había instalado algunas herramientas para optimizar código en el pasado que me ayudarían en el proceso: GitLens y multiples *build tasks*.
- **Resultados:** VSCode fue una herramienta muy útil para desarrollar el código. Todos los comandos de npm y pandoc, además de las diversas interacciones con git fueron ejecutados a través de la consola integrada. El programa incluye un servicio de diagnóstico local que te avisa de posibles problemas en el código, y a través de Gitlens se exploró el historial de ediciones en varias ocasiones para decidir o revertir múltiples cambios.

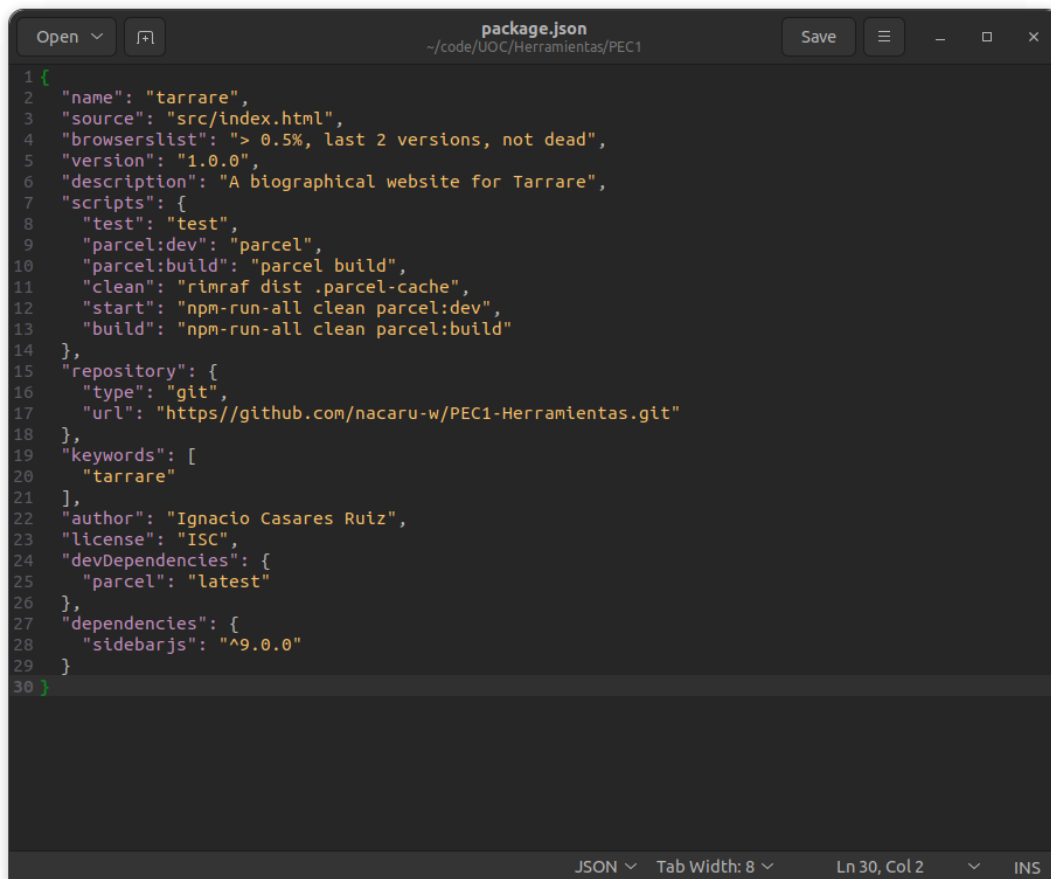
Gestión de paquetes

A continuación se tomó la decisión de con qué gestor de paquetes trabajar el proyecto. Para este fin, aunque se consideró emplear yarn, se decidió finalmente utilizar el gestor de paquetes [npm](#).

- **Justificación:** las principales razones por las que se eligió npm fueron: es más común entre los desarrolladores *frontend*, incluye herramientas útiles por defecto como NodeJS, y posee un [buscador](#) sencillo de utilizar a la hora de incorporar nuevas dependencias que puedan formar parte del proyecto. Además, npm ya se encontraba instalado en mi dispositivo y estaba familiarizado con algunos de sus comandos, por lo que solo hubo que realizar una comprobación para confirmar que se estaba usando una versión adecuada.

Tras comprobar que se estaba utilizando la versión de **npm** adecuada, a través de la consola se ejecutó el comando **init** del mismo gestor de paquetes, creando un archivo **json** a partir del cual construir la página web.

- **Resultados:** npm es una herramienta sencilla de utilizar. Se encuentra ya instalado en muchos dispositivos y su catálogo de dependencias es útil a la hora de añadir nuevos elementos a la página. Instala por defecto diversos módulos de node necesarios e incluye una serie de dependencias por defecto que pueden ser de utilidad en proyectos similares.

A screenshot of a code editor window titled "package.json" with the file path "~/code/UOC/Herramientas/PEC1". The editor shows a JSON configuration for a project named "tarrare". The configuration includes source files, a browserslist, a description, scripts for test, build, and clean, a repository URL, keywords, author, license, and dependencies. The dependencies section lists "parcel" as a devDependency and "sidebarjs" as a dependency. The status bar at the bottom indicates the file is in JSON format, with a tab width of 8, and the cursor is at line 30, column 2.

```
1 {
2   "name": "tarrare",
3   "source": "src/index.html",
4   "browserslist": "> 0.5%, last 2 versions, not dead",
5   "version": "1.0.0",
6   "description": "A biographical website for Tarrare",
7   "scripts": {
8     "test": "test",
9     "parcel:dev": "parcel",
10    "parcel:build": "parcel build",
11    "clean": "rimraf dist .parcel-cache",
12    "start": "npm-run-all clean parcel:dev",
13    "build": "npm-run-all clean parcel:build"
14  },
15  "repository": {
16    "type": "git",
17    "url": "https://github.com/nacaru-w/PEC1-Herramientas.git"
18  },
19  "keywords": [
20    "tarrare"
21  ],
22  "author": "Ignacio Casares Ruiz",
23  "license": "ISC",
24  "devDependencies": {
25    "parcel": "latest"
26  },
27  "dependencies": {
28    "sidebarjs": "^9.0.0"
29  }
30 }
```

Imagen 1: Archivo `json`. Algunas de las dependencias que aparecen en la imagen serían instaladas más tarde.

Utilización de un *module bundler*

A través de npm se decidió utilizar un module bundler a través del cual construir un *boilerplate* sobre el cual trabajar en el proyecto.

- **Justificación:** aunque ya había utilizado Webpack [en proyectos anteriores](#), decidí utilizar Parcel para afianzar lo aprendido en el módulo M2 de la asignatura. Además, parcel imprime en la consola posibles problemas y otros detalles de diagnóstico (avisa de archivos incompatibles o demasiado pesados).

Se instaló Parcel de forma local y se procedió a ejecutar el comando `npx parcel src/index.html` para construir un servidor local que permitiría ver los cambios realizados a nuestra página web en tiempo real. A continuación, se creó una hoja de estilo CSS y un archivo `.js` como dependencia al archivo `index.html` principal.

Se instaló Rimraf a través de npm para tener una herramienta que limpiase la caché creada por parcel en el procesamiento de los archivos y la construcción del servidor local.

Asimismo se crearon diversos scripts que facilitarían el trabajo en el proyecto y se añadieron al archivo `json` (véase imagen 1):

- `npm run clean` limpia la caché creada por Parcel a través de Rifraf (`rimraf dist .parcel-cache`)
- `npm run build` se asoció a `npm-run-all clean parcel:build` permite a parcel construir la aplicación para producción y optimizar los archivos del proyecto. También ejecuta `clean` que limpia la caché a través de Rifraf.
- `npm run start` se asoció a `npm-run-all clean parcel:dev`, esto permitiría igualmente limpiar la caché cada vez que se ejecutase el servidor local.

En general, los resultados de utilizar Parcel fueron los siguientes:

- **Resultados:** Parcel optimizó en una sola línea de código lo hallado en los archivos `.html`, `.css` y `.js`. Además, construyó un servidor local a través del cual se visualizaría el trabajo en tiempo real. En la consola, además, mostraba información útil como posibles incompatibilidades o el añadido de archivos con demasiado peso al proyecto.

Implementación de dependencia externa

Como requisito en la PAC, debía de realizarse la implementación de algún recurso externo en forma de dependencia.

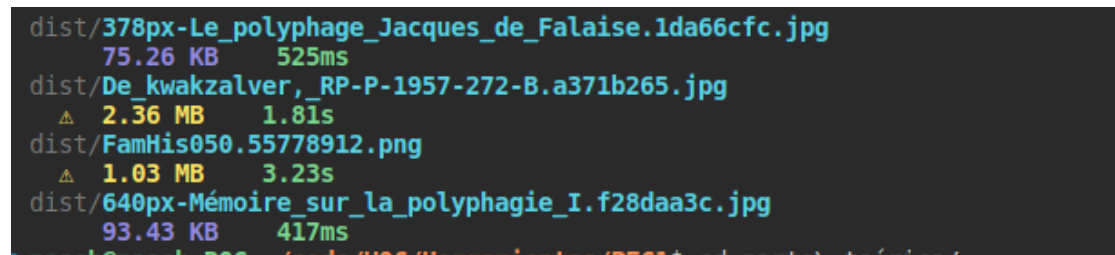


Imagen 2: Captura de pantalla que muestra el diagnóstico de Parcel a través de la consola

- Justificación: para este fin, se buscaron varias opciones a través del catálogo de npm. Se descartaron algunas de ellas por estar posiblemente obsoletas y se decidió utilizar `sidebarjs`. Se empleó esta dependencia por ser ligera, porque permitía añadir un menú de navegación lateral adicional, por estar adaptada a diferentes navegadores y a dispositivos móviles y tablets; y porque es gestionable a partir del archivo `app.js`, por lo que nos permitiría paralelamente realizar la implementación por javascript que también pedía el enunciado de la PEC.

Se instaló mediante la consola a través del código `npm install sidebarjs`. Se escribió el código HTML necesario (la dependencia requiere de la utilización de ciertos atributos), se realizó el `import` del código en el archivo `app.js` a través de las líneas:

```
import { SidebarElement, SidebarService } from 'sidebarjs';  
import 'sidebarjs/lib/sidebarjs.css';
```

Y se creó una variable que generara la *sidebar*:

```
const sidebarjs = new SidebarElement({  
});
```

La dependencia generó un fichero `sidebarjs` en la carpeta `node_modules` con el código necesario para su implementación.

Tras comprobar que la dependencia funcionaba correctamente y que no generaba lentitud o problemas de compatibilidad, se decidió mantener su uso en la página web.

Código HTML

Código CSS

Postprocesador CSS

Parte de los requisitos era la instalación de un post-procesador CSS/JS. Para esta finalidad se decidió utilizar PostCSS y Autoprefixer.

- **Justificación:** se utilizaron estos dos al estar incluidos en el compilador de Parcel. Se sopesaron otras opciones como [Sass](#) o [LESS](#), pero al no conocer las curvas de aprendizaje de estos preprocesadores, se decidió utilizar los que ya estaban incluidos en la compilación de Parcel.

Se configuró el archivo `json` del `root` para que el código se adaptase a navegadores a través de la siguiente línea:

```
"browserslist": "> 0.5%, last 2 versions, not dead",
```

- Resultados: como parte de la acción de estas aplicaciones, se optimizó el código CSS, de forma que algunas secciones de código quedaron modificadas para mayor facilidad de procesamiento. Uno de los ejemplos es el siguiente:

Este código:

```
main p {  
  font-size: 1.15em;  
}  
  
main ul {  
  font-size: 1.15em;  
}
```

Pasó a visualizarse de la siguiente forma:

```
main p, main ul {  
  font-size: 1.15em;  
}
```