Facultad de Informática
UNIVERSIDADE DA CORUÑA

TRABAJO FIN DE GRADO
GRADO EN INGENIERÍA INFORMÁTICA
MENCIÓN EN COMPUTACIÓN

Funcionalidades para la publicación de mapas web interactivos georreferenciados con integración fluida de narrativa textual

Estudiante: Jaime Fuertes Agras

Dirección: Ángel Gómez García

Estefanía López Salas

A Coruña, septiembre de 2022.

A mi familia y amigos

Resumen

La integración gráfico-textual en contextos donde se busca la difusión y comprensión de un mapa interactivo acompañado de una narrativa de formato extenso resultado de una investigación no es trivial. Hasta la fecha, esta problemática ha sido resuelta por algunas aplicaciones de manera específica o *ad-hoc*, es decir, para dar respuesta a las necesidades de determinados proyectos. Lo que proponemos en este TFG es buscar una solución más generalizable a través del uso de la principal librería de Javascript para la creación de mapas web interactivos en la actualidad, *Leaflet*, que permite su extensión mediante *plugins*. Los *plugins* son una de las formas más sencillas de añadir a un proyecto funcionalidades extra.

Este TFG se ha desarrollado siguiendo una metodología ágil para obtener resultados parciales del producto final y poder exponerlo ante los directores del proyecto. El alcance del proyecto contempla desarrollar tres funcionalidades: enlazar elementos de una narrativa textual con las representaciones gráficas de las geometrías que componen un mapa en un contexto web, así como modificar el cómo se visualiza el mapa en relación directa con el texto cambiando, por ejemplo, la capa del mismo que es visible cuando el usuario se desplace por la narrativa y viceversa. El proyecto utilizará un entorno web real como base para el desarrollo y las pruebas de las funcionalidades.

Abstract

The graphic-textual integration in contexts where the dissemination and understanding of an interactive map accompanied by a long format narrative resulting from an investigation is sought is not trivial. To date, this problem has been resolved by some applications specifically or *ad-hoc*, that is, to respond to the needs of certain projects. What we propose in this end of degree project is to find a more generalizable solution through the use of the main Javascript library for the creation of interactive web maps today, *Leaflet*, which allows its extension through *plugins*. *Plugins* are one of the easiest ways to add extra functionality to a project.

This end of degree project has been developed following an agile methodology to obtain partial results of the final product and be able to expose it to the project directors. The scope of the project contemplates developing three functionalities: linking elements of a textual narrative with the graphic representations of the geometries that make up a map in a web context, as well as modifying how the map is displayed in direct relation to the text by changing, for example , the layer of it that is visible when the user scrolls through the narrative and

vice versa. The project will use a real web environment as the basis for the development and testing of the functionalities.

Palabras clave:

- Mapas interactivos
- Leaflet
- Plugin
- Aplicación web
- JavaScript
- Narrativa Textual
- Metodología ágil

Keywords:

- Interactive maps
- Leaflet
- Plugin
- Web application
- JavaScript
- Textual narrative
- Agile methodology

Índice general

Índice de figuras

Índice de tablas

Capítulo 1
Introducción

En este primer capítulo introductorio, se presentan, en primer lugar la motivación de este Trabajo Fin de Grado (TFG), es decir, los aspectos tras la elección del tema y el enfoque de los mismos. A continuación se explican los objetivos, es decir, el alcance del trabajo y lo que se pretende conseguir a su finalización.

1.1 Motivación

La combinación de imágenes, mapas y contenido textual en un entorno web, cuando los elementos gráficos no sólo ilustran lo publicado en forma textual, sino que forman una parte integral del proceso de comprensión y transmisión de la información, derivan normalmente en la publicación de dos tipos de entidades separadas, basadas en una división entre el texto (con su formato extenso), por un lado, y el mapa o mapas como imagen fija o bien con un cierto grado de interactividad, por otro, colocados en proximidad espacial entre ellos, pero desconectados visual y técnicamente.

La lectura que deriva de esa publicación web es fragmentada, por la falta de un marcado y conexión interactiva computacional y visible entre contenidos de distinta naturaleza (gráfica y textual). La mayoría de las soluciones propuestas y ya existentes sobre este campo son específicas para la plataforma que la desarrolla (*ad-hoc*). Estas soluciones nos dan una idea general de las funcionalidades que facilitarían la lectura para el usuario, pero no proporcionan un método de implementación para quien quiere desarrollar un diseño de narrativa gráfico-textual fluida. Es por eso que en este proyecto se pretende conseguir una solución genérica y adaptable aprovechando como base herramientas con funcionalidades básicas de gestión del mapa. Con este proyecto se pretende facilitar la incorporación de funcionalidades que permitan mejorar la lectura texto-mapa mediante el uso de un *plugin* creado sobre *Leaflet* [2]. Neatline [3] proporciona una aplicación de exhibición y mapeo de uso general diseñada para satisfacer las necesidades de presentación genéricas de una amplia variedad de proyectos de

humanidades digitales. Pero en el uso real, estos proyectos académicos a menudo han incluido algún tipo de narrativa de formato largo que pretendía acompañar al mapa. Por lo general, este texto adjunto suele ser un artículo de revista o un capítulo de libro y en otras ocasiones de carácter menos formal, como una entrada de *blog*. Pero rara vez, si es que alguna vez, se ha utilizado Neatline para publicar un mapa interactivo acompañado de narrativa extensa, tipo libro y en un contexto web. La combinación de imágenes, mapas y texto de formato más largo siempre genera cierta dificultad en el proceso de creación y en la lectura por parte del usuario. Los autores que se han enfrentado a esta problemática generalmente publican las dos entidades por separado, utilizando algún tipo de página de descripción general del proyecto que vincula, generalmente con hipervínculos un texto con ciertas imágenes o un mapa completo. Sin embargo, lo que realmente se necesita es muy simple. Se trata de encontrar una forma de "citar" ciertas partes de un mapa que sea similar a la que sí podemos hacer con ciertas palabras o frases dentro del texto. No obstante, cuando el objeto de publicación es un mapa no encontramos una manera sencilla de hacer lo mismo que sí podemos hacer en una narrativa textual. Esto es lo que se intentó desarrollar en el proyecto *The Chinese Deathscape*. La forma en la que *The Chinese Deathscape*[4] presenta las ideas sobre su base nos servirá de referencia para el *plugin* de *Leaflet*.

En este TFG se pretende conseguir enlazar narrativa textual con un mapa interactivo. Para ello se pueden poner en práctica conocimientos de procesamiento de lenguaje de cara a conseguir relacionar palabras de una narrativa textual de formato extenso con, en este caso, cada uno de los elementos gráficos que, por suma, definen un mapa interactivo. Las posibles aplicaciones de los resultados son infinitas debido al gran uso de mapas interactivos que se hace en la actualidad. Por ello, ampliar sus funcionalidades supone un reto con futura utilidad por diversos sectores, tanto relacionados con el campo de la informática, como la conexión directa que este Trabajo Fin de Grado tiene con la arquitectura. Con este proyecto se pretende conseguir una herramienta generalizable en cualquier ámbito o temática. Los mapas interactivos están presentes actualmente como recurso secundario en múltiples aplicaciones, por lo que una solución genérica añadiría valor a los mapas interactivos.

Para ilustrar el alcance que se pretende conseguir con este proyecto se presentan ejemplos de aplicación realistas en los que la presencia de las nuevas funcionalidades mejoraría la experiencia del usuario. Un caso cercano sería añadir a la web de estudios de la UDC un mapa que reflejase los campus de la universidad con las diferentes facultades, acompañado de una narrativa en la que se presentasen los grados con una breve descripción haciendo referencia a la facultad a la que pertenecen. Como resultado de este añadido obtendríamos una web donde el usuario puede navegar entre los diferentes grados de forma gráfica y textual, pero esta navegación no sería independiente, mejorando el entendimiento del usuario de la relación entre las facultades y los grados impartidos. Lo mismo ocurriría para, por ejemplo, una web que

permitiese visualizar el Camino de Santiago acompañado de una narrativa que explicase las diferentes etapas y caminos que existen. De nuevo en este caso sería interesante poder aplicar las funcionalidades que se pretenden conseguir con este proyecto. En otras palabras, lo que se busca es que a un usuario que accede a una web en la que el mapa es un elemento narrativo principal que también tiene un cuerpo textual que lo acompaña, o que forma parte de la lectura o comprensión del primero, pueda realmente leer y comprender de una manera ágil la relación que existe entre el camino y su representación gráfica y cualquier dato de carácter textual que hiciera referencia a ciertas partes de aquel. Por ejemplo, desconociendo por completo las diferentes rutas que existen del Camino de Santiago, un usuario podría estar leyendo sobre el camino inglés, pero visualizando secciones del camino francés y sería el usuario el que manualmente tendría que navegar por el mapa en busca de la relación con la sección de la narrativa que se encuentre leyendo en el momento y así durante toda la lectura.

Con estos ejemplos se pretende dejar clara la necesidad de unas funcionalidades que permitan a un desarrollador implementar en su web, cualquiera que sea el campo, las mejoras que a lo largo de este TFG se describen. El alcance de este proyecto es limitado y existen diversas opciones para conseguir lo que aquí planteamos en proyectos específicos de mapeado web interactivo. En este trabajo también desarrollaremos unas funcionalidades y las testearemos sobre un proyecto concreto, pero con la intención final de que sean aplicables a cualquier ámbito. Es decir, las funcionalidades que se implementarán en este proyecto se rigen por los requisitos propuestos por los directores del TFG, pero se desarrollan de forma genérica para su uso de manera independiente a la web del proyecto sobre la que las testeamos y que introduciremos más adelante.

1.2 Objetivos

El objetivo principal de este proyecto es lograr el desarrollo de funcionalidades genéricas para la inclusión en entornos donde se presenta un mapa interactivo acompañado de una narrativa textual. Como se ha comentado anteriormente, este proyecto se construirá con la librería de JavaScript llamada *Leaflet* [5]. Se ha escogido *Leaflet* ya que presenta una gran diversidad de funcionalidades básicas para la creación de mapas web y permite su expansión mediante *plugins*. En este caso realizaremos un *plugin* que facilite al desarrollador la integración de narrativa textual con un mapa interactivo sobre entornos *Leaflet*.

Concretamente el objetivo de este proyecto será el desarrollo de funcionalidades apropiadas para integrar de forma fluida el espacio bidimensional de un mapa interactivo web, y de sus diferentes componentes gráficos, con una narrativa textual de formato extenso que se presenta al mismo tiempo, de modo que las conexiones entre mapa y texto sean visibles en el proceso de lectura por parte del usuario final del recurso y se consiga una total integra-

ción entre determinadas partes del texto y el mapa o ciertas partes del mapa que desarrollan, demuestran o expanden el argumento escrito.

Dentro de los objetivos secundarios de este trabajo estará el desarrollo de funcionalidades de marcado, anotación e interactividad específicas para la publicación, en un contexto web, de mapas georreferenciados, creados con *Leaflet*, que no son meros acompañantes de un contenido textual paralelo de formato extenso, sino los verdaderos hilos conductores del argumento.

Se buscará conseguir que el lector de la publicación tenga libertad suficiente en el producto final para explorar con un contenido rico de datos espaciales publicados en un contexto web, conectados de forma clara, pero sutil con el contenido y la estructura de una narrativa textual larga que lo acompaña.

Se desarrollarán las funcionalidades necesarias para que el lector pueda seleccionar determinados párrafos, frases o palabras de texto, codificado en HTML, que directamente lo conecten con una determinada capa de un mapa interactivo, codificado con la librería de JavaScript de código abierto *Leaflet*, o bien con alguno de sus objetos, zonas o puntos del mapa.

Asimismo, se desarrollará una funcionalidad que haga visible esa conexión entre el texto y el mapa y otra que permita mover el mapa de forma automática (en escala o posición), de acuerdo al argumento.

Cada uno de estos objetivos será una funcionalidad independiente que permitirá al usuario desarrollador elegir el nivel de fluidez en la narrativa textual. Para lograr los objetivos nos basaremos en las ideas implementadas por *The Chinese Deathscape* [4], entre las que destaca el movimiento del mapa cuando el usuario hace clic, remarcar el texto cuando el usuario pasa el ratón por encima y señalar en el mapa la estructura a la que se hace referencia en el texto cuando el usuario pasa el ratón. La solución que presenta *The Chinese Deathscape* es *ad-hoc* y basado en su propio entorno de desarrollo.

Para poder ilustrar el funcionamiento y comprobar que se cumplen los objetivos del proyecto se utilizará un entorno web real. Este entorno es el proyecto web *Digital Samos* [6]. Para ello, se implementarán también las modificaciones necesarias en el código del entorno web para conseguir una integración completa del *plugin*, pero teniendo siempre en mente la generalidad de los resultados para su posible futura aplicación a cualquier ámbito.

1.3 Estructura de la memoria

El contenido de este Trabajo Fin de Grado se ha estructurado en capítulos, en los cuales se explicarán desde los fundamentos tecnológicos hasta cada una de las fases del desarrollo. La memoria inicia con un capítulo de **introducción [1]** en el que se exponen la motivación y objetivos de este proyecto, así como la estructura del TFG, seguido de un capítulo donde

se exponen los **fundamentos tecnológicos [2]**, presentando las herramientas y el estado del arte. A continuación se desarrolla la **metodología [3]**, por una parte en este capítulo se explica la metodología a seguir durante el desarrollo del Trabajo Fin de Grado y por otro la adaptación de la misma a la situación real del proyecto, así como la herramienta empleada para el seguimiento.

Entrando en la fase de **desarrollo [4]** puro del proyecto, nos encontramos una división en *sprints* propios de la metodología escogida. Esta sección comienza con una breve exposición del aprendizaje y experimentación que fueron realizados sobre la herramienta principal de este proyecto, *Leaflet*, para continuar exponiendo cada uno de los *sprints* del desarrollo. Para finalizar el desarrollo tendremos un apartado extra donde se especificará la forma en la que se consigue el producto final, el ***plugin***.

La memoria cuenta también con un capítulo de **conclusiones [5]** en el que se presentan por un lado cómo se han cumplido los objetivos, las lecciones que se han aprendido durante el desarrollo y las competencias del grado aplicadas al proyecto. En esta sección se presentan también posibles líneas de trabajo futuras a partir de lo logrado en este desarrollo. Además contará con un **anexo [A]** donde se explicará la guía de uso e instalación del *plugin*. Por último todas las referencias se podrán consultar en la **bibliografía**.

Capítulo 2

Fundamentos tecnológicos

Este segundo capítulo se focaliza en presentar, por un lado, las herramientas y sistemas que existen actualmente, recogidos en la sección del estado del arte, y por otro, las herramientas mediante las cuales se va a realizar este TFG. En esta última sección se introduce la herramienta principal para el proyecto denominada *Leaflet*.

2.1 Estado del arte

En base a los objetivos definidos para este proyecto se ha realizado, por un lado, una búsqueda y revisión en literatura relacionada con los objetivos planteados y sus proyectos web asociados, que compartan objetivos específicos con los propuestos para este proyecto, si bien aplicados a otras herramientas y contextos de representación, y por otro, sobre los *plugins* de la herramienta sobre la cual se va a realizar este Trabajo Fin de Grado, *Leaflet*. En cuanto a *plugins* creados, se ha comprobado que no existe ninguna aproximación similar a los objetivos de este proyecto dentro de la librería de *Leaflet*.

La web de *The chinese deathscape* [4] muestran ideas de integración de narrativa textual mediante líneas que conectan el mapa con las secciones de texto sobre las que el usuario pasa el cursor como se ve en la figura 2.1. Si este hace clic sobre alguna de estas palabras o frases remarcadas en el texto, el mapa se desplazará para mostrar dicha sección del mapa.

Otra de las herramientas interesantes es *Neatline* [3], que permite crear mapas complejos, anotaciones en imágenes y secuencias narrativas. Funciona sobre el *framework* de *Omeka* [7], que proporciona una plataforma de publicación y control de colecciones de datos. Podemos ver en la figura 2.3 un ejemplo de uso de Neatline, similar al objetivo final de este proyecto. En este podemos ver las secciones del mapa (izquierda) coloreadas igual que el texto (derecha) y cuando el usuario pasa el cursor sobre el texto, este cambia de color al igual que el elemento del mapa. Si el usuario hiciese clic, el mapa haría un zoom hace la zona concreta que corresponda. Durante el desarrollo se intentará replicar el comportamiento sobre *Leaflet*.

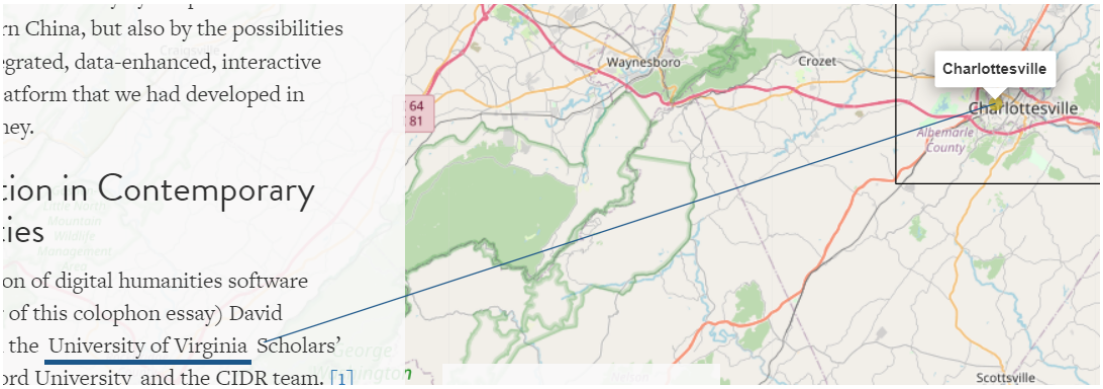

Figura 2.1: Integración de texto y mapa mediante líneas presente en The Chinese Deathscape

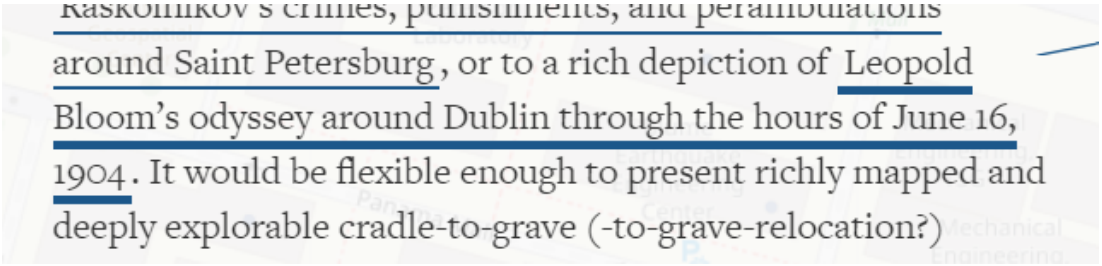

Figura 2.2: Ejemplo de marcado sobre el texto en The Chinese DeathScape

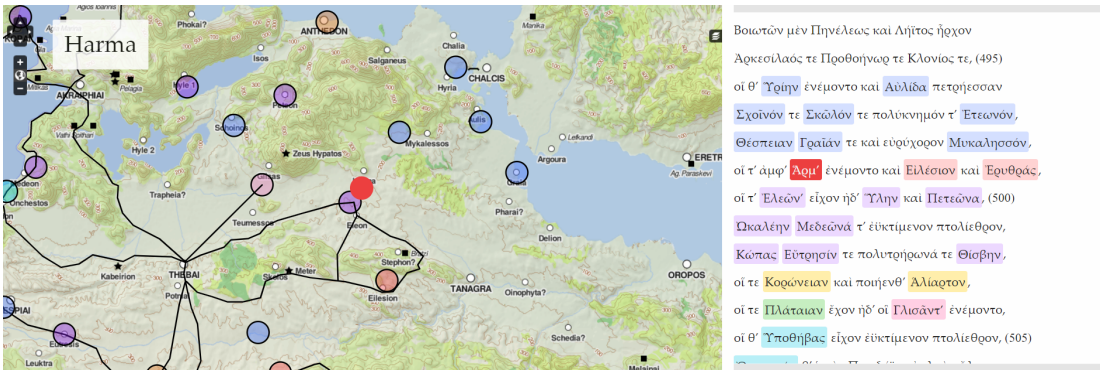

Figura 2.3: Ejemplo de uso de *Neatline* sobre *Omeka Classic* observable en Mapping the Catalogue of Ships [1]

Un sistema de información geográfica (GIS, de las siglas en inglés Geographic Information System) es un entorno para recopilar, gestionar y analizar datos. Arraigado en la ciencia de la geografía, el GIS integra una amplia variedad de tipos de datos. Analiza la ubicación espacial y organiza capas de información en visualizaciones usando mapas y escenas 3D. Con esta capacidad, los GIS revelan conocimientos más profundos de los datos, como patrones, relaciones y situaciones, ayudando a los usuarios a tomar decisiones más inteligentes. QGIS [8] es una aplicación profesional de GIS que está construida sobre Software Libre y de Códi-

go Abierto. Entre las muchas funcionalidades que ofrece, destaca la creación de mapas con elementos (normalmente polígonos) y su exportación con la información asociada a los elementos. Con QGIS es posible exportar en formato *JSON* cada capa de un mapa previamente creada dentro del entorno de este sistema de información geográfica. De hecho, QGIS fue el entorno utilizado para la creación de las múltiples capas de información que definen el mapa interactivo del proyecto *Digital Samos*.

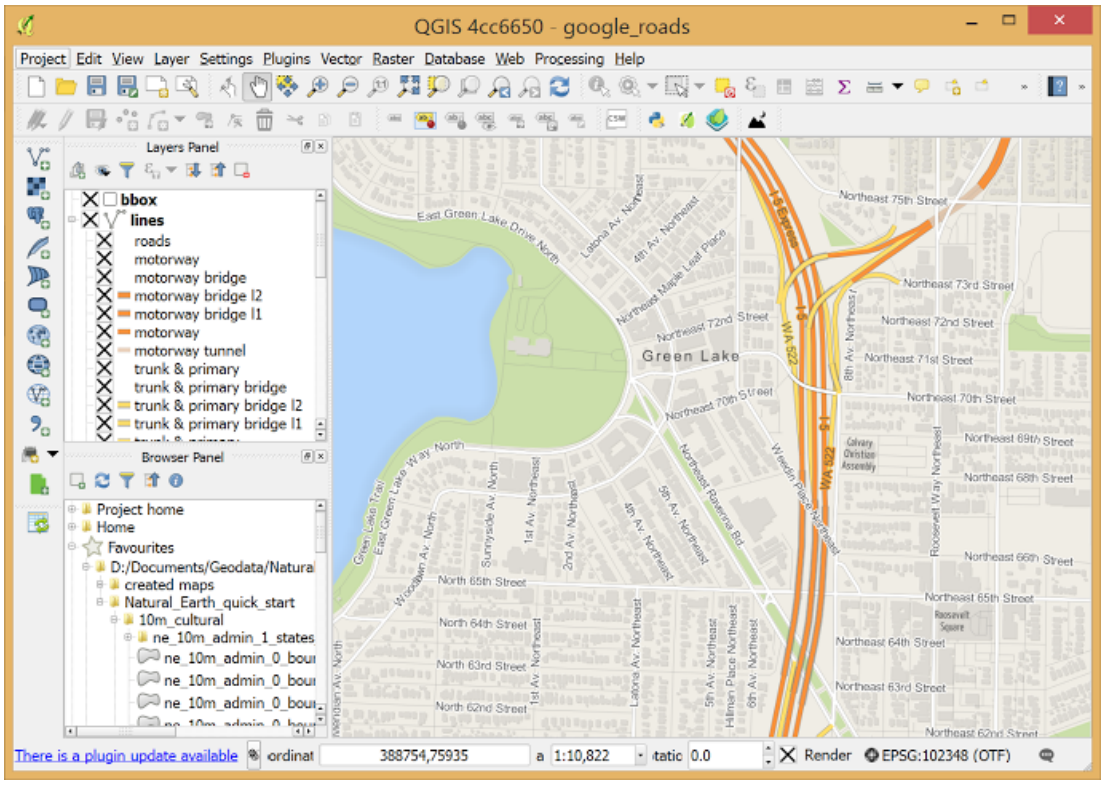

Figura 2.4: Aplicación de escritorio de QGIS

Como se ha mencionado anteriormente en esta memoria, el proyecto que se va a desarrollar se implementará en un entorno real ya existente, este entorno es la web del proyecto *Digital Samos* [6] (véase figura 2.5). Dentro de esta web nos centraremos en el mapa interactivo que representa la evolución espacio-temporal del Monasterio de San Julián de Samos, es decir, las diferentes fases evolutivas de su historia constructiva y la de su entorno inmediato. Este avance se representa en la web mediante la combinación de una narrativa marcada por los diferentes siglos y un mapa interactivo en el que se encuentran geometrías que representan elementos del monasterio en el plano. Como se ha comentado ya previamente, y es también el caso de *Digital Samos*, esta combinación provoca una lectura fragmentada para el usuario de la web. Es por eso que usando *Digital Samos* de base se intentará con el desarrollo de este proyecto mejorar el estado actual de la publicación y lectura de este tipo de proyectos

digitales en un entorno web. Todos los cambios que se realizasen directamente sobre la web, serían considerados cambios específicos (*ad-hoc*) y no mejorarían la experiencia de otros desarrolladores ya que no se podrían extrapolar a otros escenarios. Es por esto que los cambios se basarán en la web, pero de forma genérica, siendo la web un ejemplo de uso final de las funcionalidades que nuestro *plugin* aportará a este tipo de problemáticas.

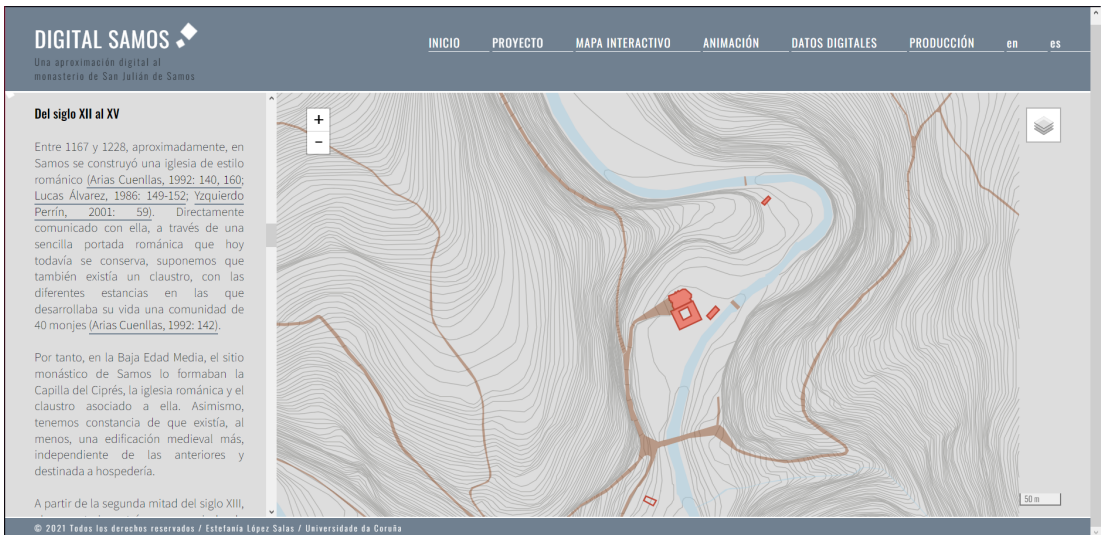

Figura 2.5: Web de *Digital Samos* en la actualidad.

2.2 Herramientas y tecnologías

Tal y como indicamos anteriormente, en este TFG desarrollaremos un *plugin* para *Leaflet*. Para ello haremos uso de las tecnologías de desarrollo web tales como *HTML*, *JavaScript* y *CSS*. El código del proyecto se realizará utilizando el IDE de desarrollo *Visual Studio Code* [9] y se llevará un control de versiones mediante *git*, en concreto se usará *Github* [10] como repositorio del código y para el seguimiento de la metodología se usará Jira [11]. En las siguientes secciones se describen brevemente estas tecnologías y herramientas.

2.2.1 HTML

Figura 2.6: HTML

HTML (Lenguaje de Marcas de Hipertexto, del inglés *HyperText Markup Language*) [12]

es el componente más básico de la Web. Define el significado y la estructura del contenido web. [13]

Hipertexto hace referencia a los enlaces que conectan páginas web entre sí, ya sea dentro de un único sitio web o entre sitios web. Los enlaces son un aspecto fundamental de la Web.

HTML utiliza marcas para etiquetar texto, imágenes y otro contenido para mostrarlo en un navegador Web. Las marcas HTML incluyen elementos especiales como <head>, <title>, <body>, <header>, <footer>, <article>, <section>, <p>, , , <img>, <aside>, <audio>, <canvas>, <datalist>, <details>, <embed>, <nav>, <output>, <progress>, <video>, <ul>, <ol>, <li> y muchos otros.

Un elemento HTML se distingue de otro texto en un documento mediante etiquetas, que consisten en el nombre del elemento rodeado por '<' y '>'. El nombre de un elemento dentro de una etiqueta no distingue entre mayúsculas y minúsculas.

2.2.2 JavaScript

Figura 2.7: JavaScript

JavaScript (JS) [14] es un lenguaje de programación ligero, interpretado, o compilado en tiempo de ejecución (*just-in-time*) con funciones de primera clase. Si bien es más conocido como un lenguaje de *scripting* (secuencias de comandos) para páginas web, y es usado en muchos entornos fuera del navegador, tal como Node.js, Apache CouchDB y Adobe Acrobat. JavaScript es un lenguaje de programación basada en prototipos, multiparadigma, de un solo hilo, dinámico, con soporte para programación orientada a objetos, imperativa y declarativa, por ejemplo programación funcional. [13]

Dentro de JavaScript haremos uso de las APIs, entre ellas destacar el uso de DOM. El DOM (*Document Object Model*, en español Modelo de Objetos del Documento) es una API definida para representar e interactuar con cualquier documento HTML o XML. El DOM es un modelo de documento que se carga en el navegador web y que representa el documento como un árbol de nodos, en donde cada nodo representa una parte del documento (puede tratarse de un elemento, una cadena de texto o un comentario).

2.2.3 CSS

Figura 2.8: CSS

CSS (Hojas de Estilo en Cascada en inglés *Cascading Style Sheets*) [15] es el lenguaje de estilos utilizado para describir la presentación de documentos HTML o XML, incluyendo varios lenguajes basados en XML como SVG, MathML o XHTML. CSS describe como debe ser renderizado el elemento estructurado para visualización, impresión, reproducción de voz o en otros medios. [13]

CSS es uno de los lenguajes base de la Open Web y posee una especificación estandarizada por parte del W3C. Anteriormente, el desarrollo de varias partes de las especificaciones de CSS era realizado de manera sincrónica, lo que permitía el versionado de las recomendaciones.

Desde CSS3, el alcance de las especificaciones se incrementó de forma significativa y el progreso de los diferentes módulos de CSS comenzó a mostrar varias diferencias, lo que hizo más efectivo desarrollar y publicar recomendaciones separadas por módulos.

2.2.4 Leaflet

Figura 2.9: *Leaflet*

Leaflet [5] es la librería *open source* líder sobre *JavaScript* para la creación de mapas web interactivos. Esta librería ha sido diseñada con la simpleza, el rendimiento y la usabilidad en mente. Funciona de manera eficiente entre las principales plataformas, tanto de escritorio como móviles. *Leaflet* puede ser extendido mediante *plugins*, cuenta con una *API* bien documentada y un código fuente cómodo de leer.

Como se puede apreciar en la figura 2.10, *Leaflet* cuenta con una gran variedad de características básicas, si bien la librería no se centra en abarcar todas las posibilidades, sino en que las básicas funcionen correctamente. Entre las funcionalidades presentes en *Leaflet* podemos destacar la diversidad de interacción que el usuario tiene con el mapa, ya sea el usuario final a través de la web, como el desarrollador a través de las funciones de *Leaflet*. Además de esto cuenta con una amplia compatibilidad de navegadores y un rendimiento mejorado con la

aceleración de hardware en dispositivos móviles, lo cual lo hace una gran herramienta para poder representar información en uno de los medios más utilizados actualmente que son los dispositivos móviles.

Características

Leaflet no intenta hacer todo para todos. En cambio, se enfoca en hacer que *las cosas básicas funcionen perfectamente* .

Capas fuera de la caja

- Capas de teselas, WMS
- Marcadores, ventanas emergentes
- Capas vectoriales : polilíneas, polígonos, círculos, rectángulos
- superposiciones de imágenes
- GeoJSON

Funciones de interacción

- Arrastre paneo con inercia
- Zoom de la rueda de desplazamiento
- Pinch-zoom en el móvil
- Ampliar con doble clic
- Zoom al área (shift-drag)
- Navegación por teclado
- Eventos : clic, mouseover, etc.
- Arrastrar marcador

Características visuales

- Animación de zoom y panorámica
- Animación de desvanecimiento de mosaicos y ventanas emergentes
- Diseño predeterminado muy agradable para marcadores, ventanas emergentes y controles

Funciones de personalización

- Ventanas emergentes y controles de CSS3 puro para cambiar el estilo fácilmente
- Marcadores basados en imágenes y HTML
- Una interfaz simple para capas y controles de mapas personalizados
- Proyecciones de mapas personalizados (EPSG:3857/4326/3395fuera de la caja)
- Potentes instalaciones de programación orientada a objetos para ampliar las clases existentes

Características de rendimiento

- La aceleración de hardware en dispositivos móviles lo hace sentir tan fluido como las aplicaciones nativas
- Utilizando las funciones de CSS3 para hacer que la panorámica y el zoom sean realmente fluidos
- La representación inteligente de polilíneas/polígonos con recorte

Controles de mapa

- Botones de acercamiento
- Atribución
- Conmutador de capa
- Escala

Compatibilidad con navegador

Escritorio

- Cromo
- Firefox
- Safari 5+
- Ópera 12+
- IE 7–11
- Borde

Móvil

- Safari para iOS 7+
- Chrome para móviles
- Firefox para móvil
- IE10+ para dispositivos Win8

Varios

- Extremadamente ligero
- Sin dependencias externas

Figura 2.10: Características de *Leaflet* (recogidas en la web)

Leaflet cuenta con una estructura de clases [16] como la que se puede apreciar en la figura 2.11. A continuación haré una breve mención de las clases más importantes de la herramienta así como las que nos interesan de cara al desarrollo del Trabajo Fin de Grado.

La primera y más importante es el mapa, representado en *Leaflet* mediante la clase *L.Map*. Esta clase tiene las funcionalidades básicas para crear el mapa, permite añadir capas (*L.Layer*) y control (*L.Control*). La primera de ellas contiene la información sobre cada una de las capas y está compuesta por diversas subclases que permiten añadir información sobre la capa, ya sea un *popup* (*L.Popup*), un polígono (*L.Polygon*) o una de las importantes para este trabajo un GeoJSON (*L.GeoJSON*).

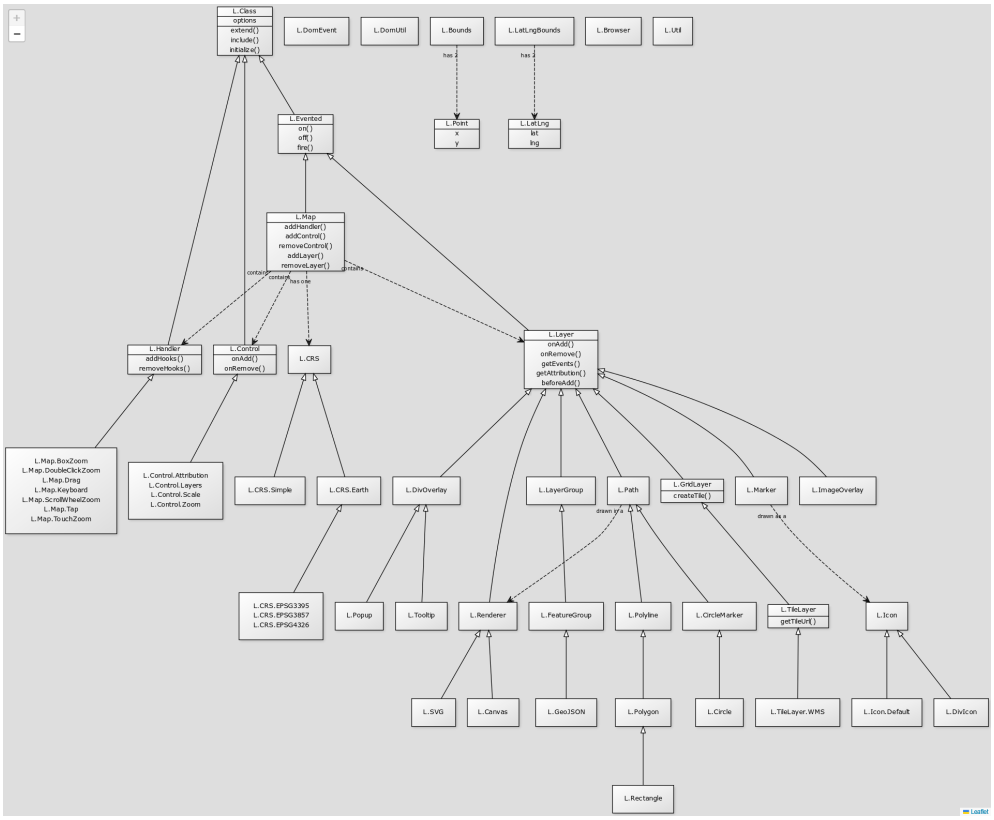

Figura 2.11: Diagrama de clases *Leaflet*

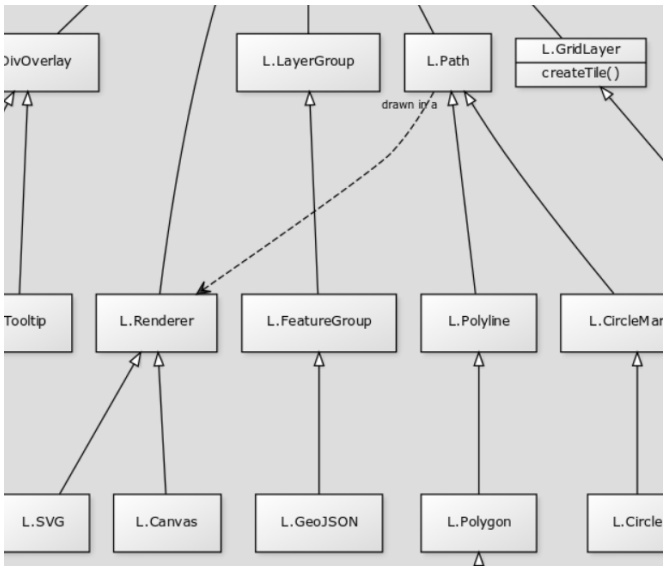

Figura 2.12: Diagrama de clases *Leaflet* centrado en L.GeoJSON

El *GeoJSON* 2.12 nos permite añadir información geográfica a través de archivos *JSON* (*JavaScript Object Notation*). Estos archivos *JSON* nos serán de utilidad de cara al desarrollo, ya que podremos extraer de ellos información que relacione de cierta manera la narrativa con el elemento representado por ese *JSON* en el mapa.

Como se puede observar en la figura 2.11, la estructura de clases base no es compleja ni completa, por ello *Leaflet* provee la posibilidad de expandirla mediante funciones como las siguientes:

- **L.Class.extend()**: permite crear subclases que deriven de las clases propias de *Leaflet*.
- **L.Class.include()**: permite añadir nuevos métodos a las clases propias de *Leaflet*, así como modificar aquellos ya presentes estas.
- **L.Class.initialize()**: es el método constructor de las clases de *Leaflet*.
- **Factories**: no es un método como tal, pero es necesario comentar que en *Leaflet* la mayoría de las clases cuentan con una función factoría. El nombrado de las factorías es el mismo que el de la clase pero en minúsculas (*lowerCamelCase* en lugar de *UpperCamelCase*).

2.2.5 Git

Figura 2.13: Git

Git [17] es un sistema de control de versiones de código abierto maduro y con un mantenimiento activo que desarrolló originalmente Linus Torvalds, el creador del kernel del sistema operativo Linux, en 2005. Un asombroso número de proyectos de software dependen de Git para el control de versiones, incluidos proyectos comerciales y de código abierto. Los desarrolladores que han trabajado con Git cuentan con una buena representación en la base de talentos disponibles para el desarrollo de software, y este sistema funciona a la perfección en una amplia variedad de sistemas operativos e IDE (entornos de desarrollo integrados). Git, que presenta una arquitectura distribuida, es un ejemplo de DVCS (sistema de control de versiones distribuido, por sus siglas en inglés). En lugar de tener un único espacio para todo el historial de versiones del software, como sucede de manera habitual en los sistemas de control de versiones antaño populares, como CVS o Subversion (también conocido como SVN), en Git, la copia de trabajo del código de cada desarrollador es también un repositorio que

puede albergar el historial completo de todos los cambios. Además de contar con una arquitectura distribuida, Git se ha diseñado teniendo en cuenta el rendimiento, la seguridad y la flexibilidad.

2.2.6 Github

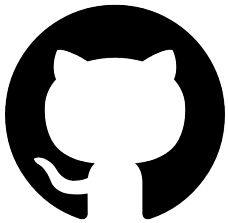

Figura 2.14: Github

Para la posterior publicación del *plugin*, y como repositorio de control de versiones se utilizará *Github* [10].

Github es un portal creado para alojar el código de las aplicaciones de cualquier desarrollador, y que fue comprada por *Microsoft* en junio del 2018. La plataforma está creada para que los desarrolladores suban el código de sus aplicaciones y herramientas, y que como usuario no sólo puedas descargarte la aplicación, sino también entrar a su perfil para leer sobre ella o colaborar con su desarrollo. Como su nombre indica, la web utiliza el sistema de control de versiones *Git* diseñado por Linus Torvalds. Un sistema de gestión de versiones es ese con el que los desarrolladores pueden administrar su proyecto, ordenando el código de cada una de las nuevas versiones que sacan de sus aplicaciones para evitar confusiones. Así, al tener copias de cada una de las versiones de su aplicación, no se perderán los estados anteriores cuando se va a actualizar.

Git es uno de estos sistemas de control, que permite comparar el código de un archivo para ver las diferencias entre las versiones, restaurar versiones antiguas si algo sale mal, y fusionar los cambios de distintas versiones. También permite trabajar con distintas ramas de un proyecto, como la de desarrollo para meter nuevas funciones al programa o la de producción para depurar los *bugs*. Las principales características de la plataforma es que ofrece las mejores características de este tipo de servicios sin perder la simplicidad, y es una de las más utilizadas del mundo por los desarrolladores. Es multiplataforma, y tiene multitud de interfaces de usuario.

Github es un portal para gestionar las aplicaciones que utilizan el sistema *Git*. Además de permitirte mirar el código y descargarte las diferentes versiones de una aplicación, la plataforma también hace las veces de red social conectando desarrolladores con usuarios para que estos puedan colaborar mejorando la aplicación.

2.2.7 NPM

Figura 2.15: NPM

NPM [18] (*Node Package Manager*) es un sistema gestor de paquetes para *JavaScript*y es el registro de software más grande del mundo. Desarrolladores de código abierto de todos los continentes usan *npm* para compartir y tomar prestados paquetes de código. Muchas organizaciones usan *npm* de manera privada para organizar el desarrollo. Este software está compuesto por tres componentes distintos: la web del proyecto, la interfaz de línea de comandos (*CLI* en ingles *Command Line Interface*) y el registro.

Utiliza la web para descubrir paquetes, crear perfiles, y manejar otros aspectos de tu experiencia *npm*. Por ejemplo, puedes crear una organización para manejar el acceso a paquetes públicos y/o privados. La CLI se ejecuta en un terminal y es la forma en la que la mayoría de los desarrolladores interactúan con *npm*. El registro es una gran base de datos pública de software en JavaScript y la meta-información que la rodea.

Capítulo 3
Metodología

En este capítulo se describe la metodología aplicada en el desarrollo del proyecto, así como la herramienta de seguimiento del mismo. Para ello se divide en dos secciones, la primera de ellas describe la metodología y la adaptación de la misma a este proyecto, y la segunda, se centra en la herramienta escogida para el seguimiento y la organización del desarrollo.

3.1 Descripción

Para este proyecto se ha optado por una metodología ágil dirigida por las funcionalidades del sistema.

Como describe el manifiesto ágil [19] en estas metodologías se valora:

- Al individuo y las interacciones del equipo de desarrollo sobre el proceso y las herramientas.
- Desarrollar software que funciona más que conseguir una buena documentación.
- La colaboración con el cliente más que la negociación de un contrato.
- Responder a los cambios más que seguir estrictamente un plan.

La metodología *Scrum* [20] [21] es un proceso para llevar a cabo un conjunto de tareas de forma regular con el objetivo principal de trabajar de manera colaborativa. El desarrollo de software se realiza mediante iteraciones, denominadas *sprints* de una duración media de 30 días. En el alcance de este proyecto la duración de los sprints serán de aproximadamente 1 mes. El resultado de cada sprint es un incremento ejecutable que se muestra al cliente. En este caso el rol de cliente lo desempeñarán los directores del proyecto.

Scrum es una metodología útil cuando se requieren resultados a corto plazo y en los que existen tareas poco definidas, por ello funciona perfectamente de cara a este proyecto, en el que, si bien no cuenta con un equipo de desarrollo, el tiempo del mismo es limitado.

3.1.1 Fases *Scrum*

La metodología *Scrum* se divide en 5 fases: inicio, planificación, implementación o ejecución, revisión y lanzamiento. En esta sección comentaremos las 3 más importantes y las centrales de la metodología. Las fases de inicio y lanzamiento se consideran triviales.

- **Planificación: Product Backlog**
  El Product Backlog es la fase en la que se establecen las tareas prioritarias y donde se obtiene información breve y detallada sobre el proyecto que se va a desarrollar.

- **Ejecución: Sprint**
  Dentro del método Scrum, el Sprint o iteración es el corazón, un intervalo de tiempo que como máximo tiene una duración de un mes y en donde se produce el desarrollo de un producto que es entregable potencialmente. Se puede definir el Sprint como un pequeño proyecto en donde el equipo de trabajo se focaliza en el desarrollo de tareas para alcanzar el objetivo.

- **Control: Burn Down**
  El Burn Down es la fase en la que se mide el progreso de un determinado proyecto Scrum. En ella, el Scrum Master será el encargado de actualizar los gráficos cuando se finalice cada uno de los Sprint.

3.1.2 Roles Scrum

Los roles típicos de la metodología *Scrum* son el *product owner*, *scrum master* y el *team*.

- **Product Owner (Dueño del Producto)**
  Tiene la responsabilidad de decidir qué trabajo debe hacerse y maximizar el valor del producto, para ello debe **gestionar prioridades**, siendo estas los presupuestos o la contratación del equipo de desarrollo entre otras, ser el **representante de negocio** e **intraemprendedor**. Con estas dos últimas el *product owner* demuestra una capacidad de decisión y aporta valor al negocio midiendo el valor generado en cada sprint.

- **Scrum Master**
  Actúa como un líder servicial, ayudando al equipo y a la organización a usar lo mejor posible la metodología Scrum. Se focaliza en la parte de negocio y es responsable del ROI del proyecto. Traslada la visión del proyecto al equipo, formaliza las prestaciones en historias a incorporar en el Product Backlog y las prioriza de forma regular.

- **Team (Equipo de desarrollo)**
  Grupo de profesionales con los conocimientos técnicos necesarios y que desarrollan el

proyecto de manera conjunta llevando a cabo las historias a las que se comprometen al inicio de cada sprint. En cada Sprint, el equipo construye y entrega un incremento del producto.

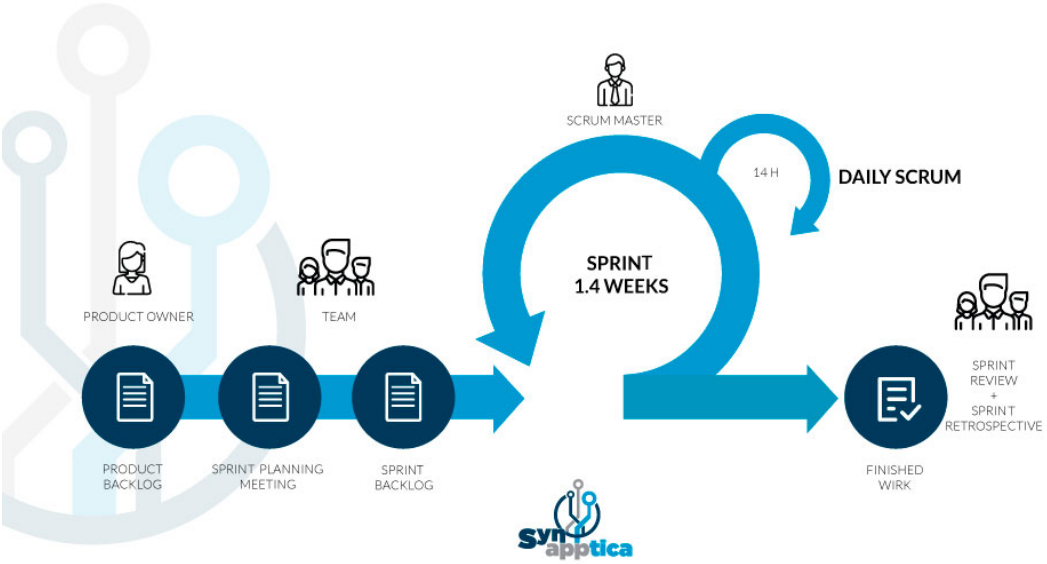

Figura 3.1: Ciclo de vida *Scrum*

Como se puede apreciar en la figura 3.1 el ciclo de vida de *Scrum* es semejante a un ciclo de vida en espiral, partiendo de una backlog y una planificación de los *sprints*. La parte en espiral se corresponde al desarrollo de cada *sprint* ya que se pretende tener un producto parcial entregable al final del *sprint*. Como se puede apreciar también dentro del ciclo del *sprint* encontramos las reuniones diarias (*daily scrum*). Al finalizar todos los *sprints* obtendremos el producto final.

3.1.3 Adaptaciones de la metodología al proyecto

En este proyecto se ha realizado una adaptación de la metodología ya que el equipo de desarrollo consta de una única persona, en este caso el estudiante, que desempeñará todos los roles propios de *Scrum*. Así, las funciones del *scrum master* y del *team* recaen en el estudiante al ser el único integrante del equipo de desarrollo. Por este mismo motivo las reuniones del tipo *Daily Scrum* no se realizan ya que no hay un equipo que reunir. Así mismo el *Sprint Review* se realiza con los *product owners* que en este caso son los directores de este TFG.

3.2 Herramienta de control y seguimiento

En esta sección se presenta la herramienta utilizada para el control de la metodología y cómo se ha realizado el seguimiento de las tareas.

3.2.1 Jira Software

Jira [11] es una familia de soluciones para la gestión del trabajo de metodología ágil que impulsa la colaboración entre todos los equipos, desde el diseño hasta el cliente final. Jira ofrece varias opciones de implementación y productos diseñadas específicamente para el software, las TI, la empresa, los equipos de operaciones y mucho más.

El menú principal de Jira Software cuenta con dos secciones que se ven representadas en la figura 3.2. La primera de ellas es el tablero del *sprint*, en el cual se muestran las incidencias asignadas al mismo, así como el estado de estas y el usuario asignado. La segunda sección es el *backlog*, una lista de todas las tareas del proyecto que puede ir variando a medida que aparezcan nuevos requisitos. Las incidencias representadas en Jira pueden ser de tres clases: **tarea**, **historia** o **error**.

Otra de las secciones con las que cuenta Jira es el tablero de incidencias (figura 3.3) que proporciona otra forma de visualización del sprint actual dividido en columnas por la situación de la tarea.

Para realizar el seguimiento se llevan a cabo reuniones semanales con los directores del Trabajo Fin de Grado donde se comentan los avances en el *sprint* y se crea el siguiente.

Dentro de la metodología *Scrum* se realizan reuniones de seguimiento diarias con el equipo de desarrollo. Como ya se ha comentado estas reuniones no son necesarias en este proyecto debido a que el equipo de desarrollo está compuesto por una única persona.

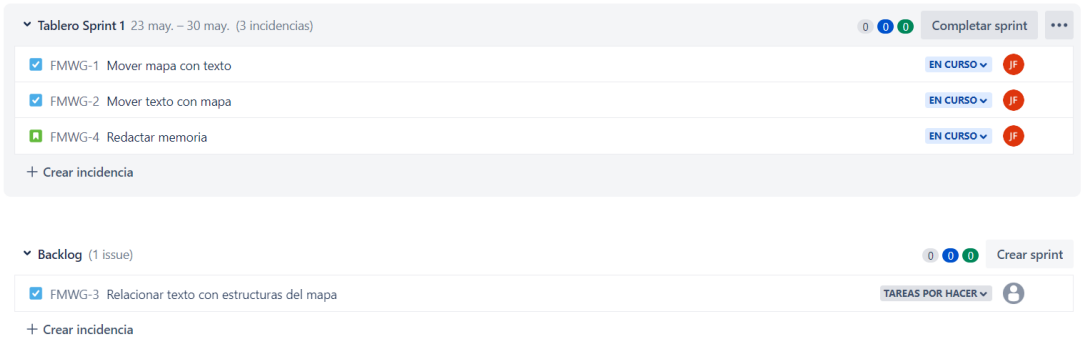

Figura 3.2: Detalle del *backlog* del proyecto en Jira

Figura 3.3: Detalle del tablero del *sprint* en Jira

Capítulo 4
Desarrollo

Este capítulo presenta las fases de desarrollo del proyecto siguiendo la metodología *Scrum*. Comienza presentando la planificación y estimación de costes del proyecto para continuar exponiendo el aprendizaje realizado sobre la herramienta de *Leaflet* y luego abordar el desarrollo puro del proyecto, dividido en iteraciones (*sprints*) con sus correspondientes fases de análisis, diseño, implementación y pruebas. En la última iteración se terminará con la generación del *plugin*, dando por concluido el desarrollo.

4.1 Planificación

Este proyecto se ha llevado a cabo en un período de 4 meses y medio a tiempo parcial (Mayo 2022 - Septiembre 2022). En las reuniones iniciales del proyecto se identificaron tres funcionalidades independientes con los requisitos que ya se han mencionado en el capítulo introductorio 1 de este TFG. Para cada una de las funcionalidades se estimó una duración de 1 mes, es decir, un *sprint* por cada funcionalidad.

Para poder desarrollar las funcionalidades es necesario pasar por un proceso de diseño y una posterior implementación. Se tendrán en cuenta dos perfiles que en este proyecto desempeñará el estudiante, siendo a la vez diseñador y desarrollador. Como se ha comentado la duración del proyecto se ha planificado para durar algo más de cuatro *sprints* (4 meses y medio). Cada *sprint* tiene una duración de 30 días, de los cuales se estima en torno a 2 horas de trabajo diarias, siendo en total unas 60 horas de trabajo al mes. De esas 60 horas el 20% (12 horas) se dedicarán al diseño de la funcionalidad y las otras 48 restantes al desarrollo en código.

En las figuras 4.1 y 4.2 podemos apreciar la diferencia del desarrollo con la planificación de tiempos en cada uno de los *sprints*. Como podemos observar se planificó un *sprint* para cada funcionalidad (en la figura los *sprints* 1, 2 y 3), pero a la hora del desarrollo las dos primeras funcionalidades, al tener mucho en común, se desarrollaron en paralelo. Como se comentará

a continuación el *sprint* que se planificó para mejoras de las funcionalidades, el 4 en la figura, en el momento de desarrollo se comenzó como segundo *sprint* y durando hasta el final del desarrollo, siendo constantes estas mejoras. Debido a problemas externos involucrados con plazos de la propia Universidad los plazos de los *sprints* no coinciden con la planificación ya que no se han podido dedicar las horas estimadas en el plazo estimado.

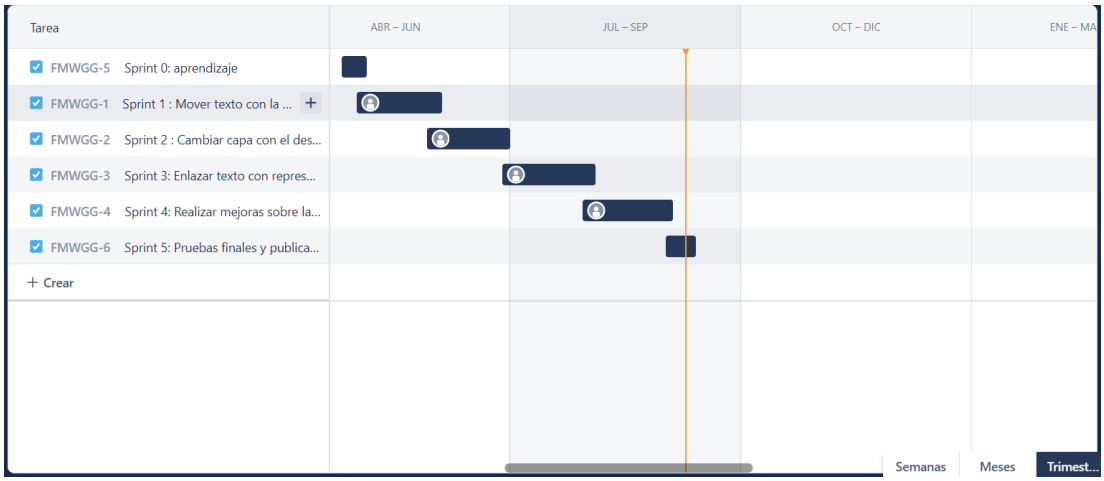

Figura 4.1: Representación de la Diagrama de Gantt según la planificación estimada en Jira

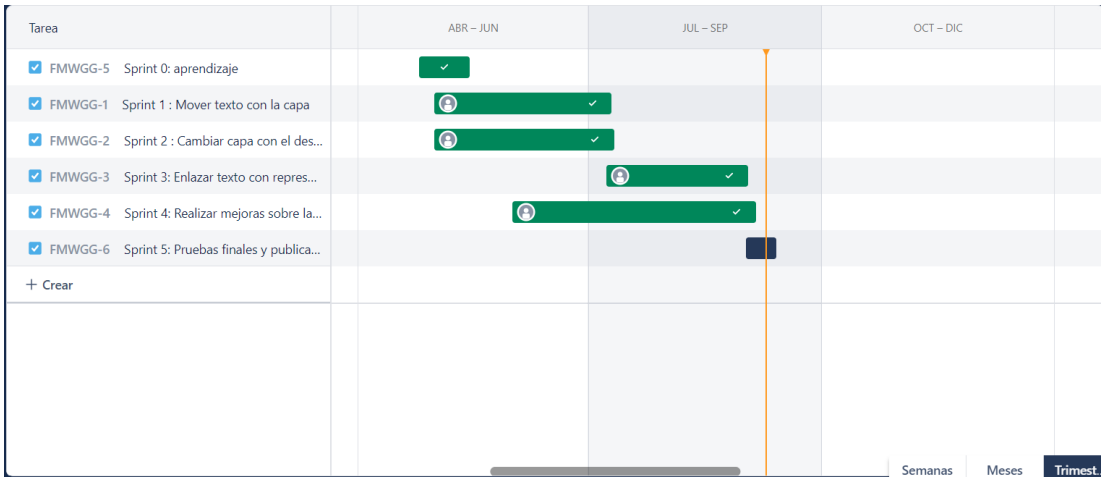

Figura 4.2: Representación de la Diagrama de Gantt según la implementación real en Jira

4.2 Estimación de costes

Una vez planificado y organizado el proyecto se calcula la estimación de costes según se indica en la tabla 4.1. Para obtener los datos se ha consultado en el BOE [22], aunque el coste/hora indicado es probablemente inferior al coste actual del mercado laboral

Recurso	Horas	Coste (€/hora)	Coste total
Diseñador	48	22€	1056€
Desarrollador	192	22€	4224€
Ordenador	-	-	0€
Suma costes totales			5280€

Tabla 4.1: Estimación de costes

4.3 *Sprint* 0: Experimentación y aprendizaje

Previo al inicio del desarrollo de este Trabajo Fin de Grado, ha sido necesario familiarizarse con la herramienta *Leaflet* para la publicación de mapas web interactivos. La web de *Leaflet* cuenta con tutoriales detallados sobre las funciones básicas de la herramienta como se puede observar en la figura 4.3.

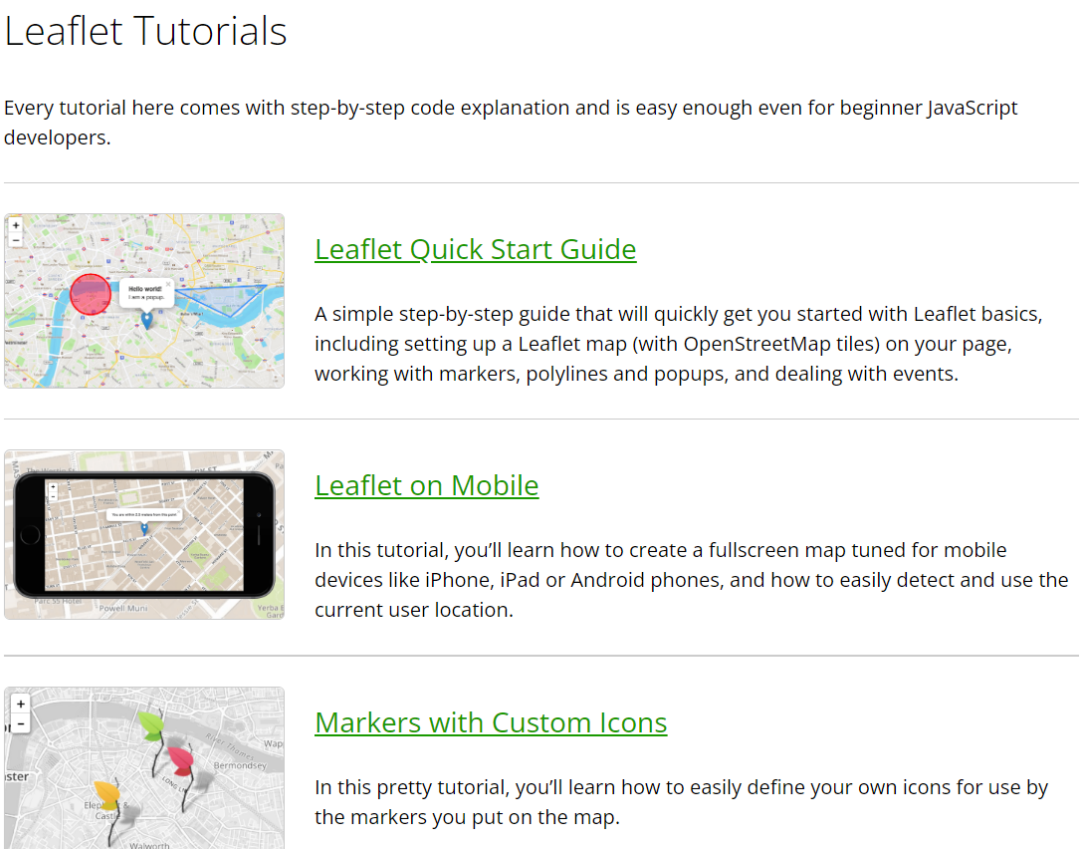

Figura 4.3: Ejemplo de tutoriales básicos de *Leaflet*

Una vez realizados los tutoriales, el siguiente paso fue crear una primera prueba desde cero con lo aprendido en los tutoriales y comenzar a probar formas de utilizar las funciones definidas por *Leaflet* con la intención de lograr modificar información del mapa a través de las funciones de *JavaScript*.

Con las primeras pruebas completadas se dio el paso al desarrollo de la base inicial del contenido principal de este TFG, la creación de funcionalidades para la publicación de mapas web interactivos georreferenciados con integración fluida de narrativa textual, llevada a cabo extendiendo la librería de JavaScript *Leaflet* mediante un *plugin* que permita dar respuesta a las necesidades del problema que pretendemos resolver y que se pueda integrar en cualquier ámbito. El siguiente paso es analizar la web de *Digital Samos* para entender la forma en la que hace uso de *Leaflet* que, como se comentó en los objetivos (ver sección 1.2), será la web sobre la que se realizarán las pruebas del *plugin* para comprobar el alcance de la solución.

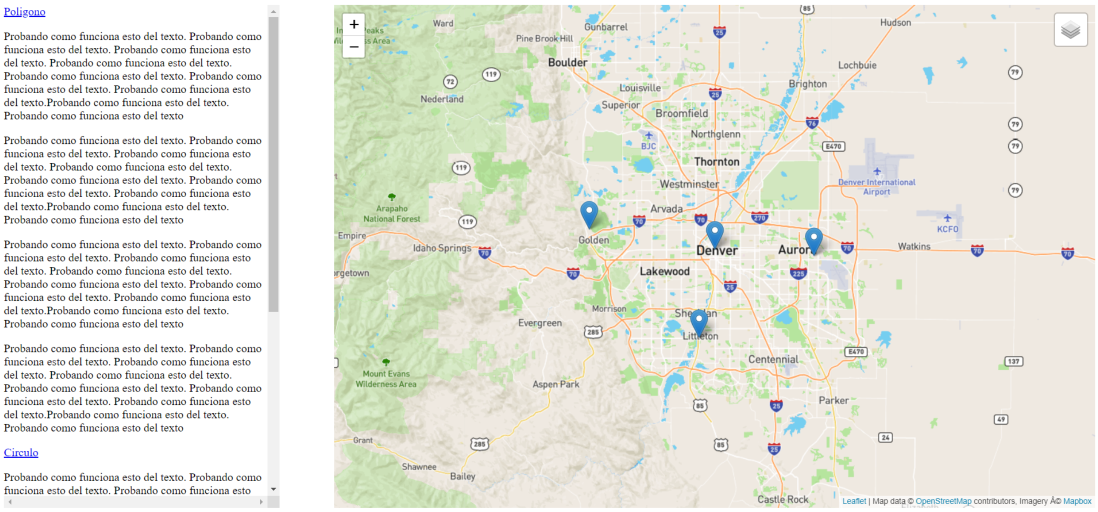

Figura 4.4: Ejemplo creado siguiendo los tutoriales de *Leaflet*

Digital Samos [6] es una web centrada en aumentar el conocimiento de la evolución espacial arquitectónica experimentada por el complejo de San Julián de Samos. Para ello, como se puede apreciar en la figura 4.5, cuenta con un mapa interactivo acompañado de una breve narrativa textual a su izquierda. En la esquina superior derecha del mapa se puede apreciar el selector de capas propio de *Leaflet*, el cual permite al usuario moverse entre diferentes etapas evolutivas del complejo monástico. Por el momento esta navegación es independiente del texto y esta unión se pretende conseguir con las funcionalidades que será implementadas en este proyecto.

En cuanto a cómo está estructurada la página a nivel de código, el *HTML* contiene el texto envuelto con una *tag* <div> y cada sección del texto contiene un icono en la parte superior con un *id* propio que nos será útil para identificar las secciones del texto. En la parte de *JS*

podemos ver el uso de la librería *Leaflet* tanto para la creación del mapa en sí, como con el amplio uso de la función *L.geoJSON()*. Esta función permite leer un archivo con extensión *.js* que contiene las coordenadas geográficas de cada entidad representada así como otros datos de muy diverso tipo (nombre de la entidad, tipo de la entidad, características, época, estilo,...).

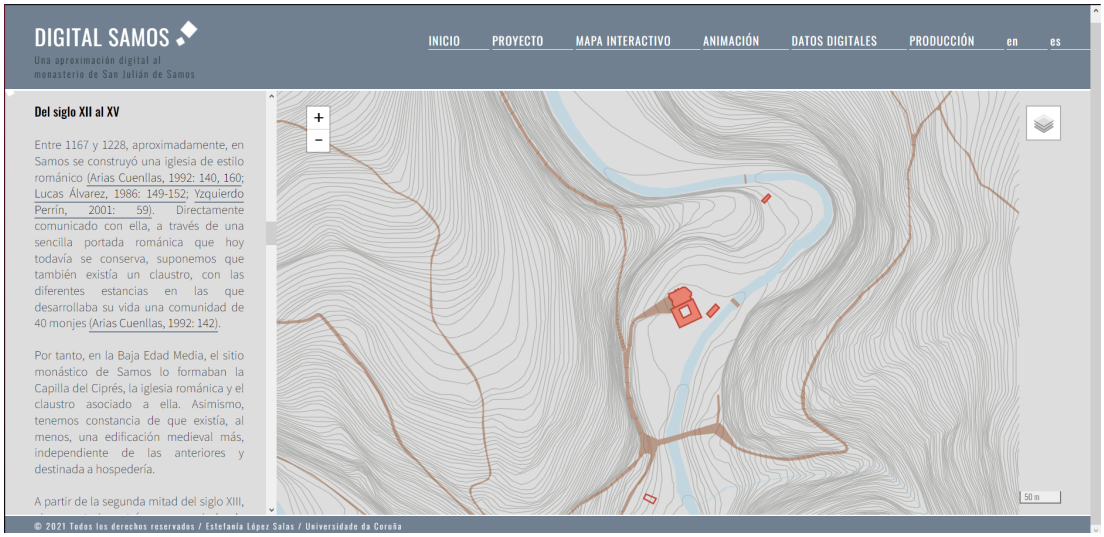

Figura 4.5: Web de *Digital Samos* sobre la que se realizará el Trabajo Fin de Grado

Todas las construcciones del monasterio de Samos tienen su representación en el mapa y se denominarán geometrías a lo largo de este TFG. Los archivos con los datos de las geometrías en formato *.js* fueron creados manualmente a partir de la exportación en formato *JSON* de las geometrías previamente representadas en el entorno de un QGIS. Aunque QGIS [8] es una herramienta que se sale del alcance de este proyecto, podría ser de gran utilidad en aproximaciones futuras del *plugin*. En el proyecto que nos ocupa los *JSON* creados en QGIS contienen información acerca de las coordenadas de las geometrías (ver figura 4.6), así como los datos que se mostrarán en los *popups* (ventanas emergentes en el mapa) cuando se haga clic sobre las geometrías. La información que contiene este archivo nos será útil para poder relacionar elementos en el texto (palabras) con geometrías del mapa (polígonos).

Este primer *sprint* de aprendizaje se considera el 0 ya que no ha tenido la duración establecida para los *sprints* de desarrollo, sin embargo ha servido para conocer más a fondo la herramienta sobre la que se va a trabajar y aclarar ideas acerca de las funcionalidades que se pretenden conseguir con este proyecto. Estas ideas se comenzarán a desarrollar en el siguiente *sprint*.

```
{
"type": "FeatureCollection",
"name": "romanesque_church_7",
"crs": { "type": "name", "properties": { "name": "urn:ogc:def:crs:OGC:1.3:CRS84" } },
"features": [
{ "type": "Feature", "properties": { "fid": 1, "Layer": "church", "Name": "Romanesque Church",
"Period": "Medieval", "Construction": "From around 1167 to 1228", "Event": "-",
"Extant": "No, demolished in 1746, only partial remains", "Url": "under_construction.html"},
"geometry": { "type": "MultiPolygon",
"coordinates": [ [ [ [ -7.325974322706277, 42.732236352565664 ],
[ -7.325899634718406, 42.732120919117776 ], [ -7.32599455531136, 42.73208778286476 ],
[ -7.326015179190977, 42.73208058318032 ], [ -7.326016381843612, 42.732080163340726 ],
[ -7.326014756038687, 42.732077650586319 ], [ -7.326017588953856, 42.732076661630906 ],
[ -7.326015556697686, 42.732073520687678 ], [ -7.326027617396209, 42.732069310362427 ],
[ -7.326029649652376, 42.732072451305839 ], [ -7.326032482567549, 42.732071462350326 ],
[ -7.326034108372492, 42.732073975104996 ], [ -7.326081949201825, 42.732057274126106 ],
[ -7.326080323396906, 42.732054761370783 ], [ -7.326083156312073, 42.732053772414993 ],
[ -7.326081124055905, 42.732050631470621 ], [ -7.32609318475443, 42.732046421143799 ],
[ -7.326095217010593, 42.732049562088392 ], [ -7.326098049925776, 42.732048573132516 ],
[ -7.326099675730705, 42.732051085888102 ], [ -7.326122686364478, 42.732043052995877 ],
[ -7.32612106055955, 42.732040540239986 ], [ -7.326123893474724, 42.732039551283968 ],
[ -7.326121857580431, 42.732036411608931 ], [ -7.3261212482895, 42.732035541385606 ],
[ -7.326121286008445, 42.732035528218127 ], [ -7.326120856141732, 42.732034981298291 ],
[ -7.326132270666869, 42.732030029698521 ], [ -7.326137322925065, 42.732035601359712 ],
[ -7.326139358204945, 42.732034801855811 ], [ -7.326140769497862, 42.732036740267688 ],
[ -7.326164258103208, 42.732028540516218 ], [ -7.32616259518175, 42.732026256492034 ],
```


Figura 4.6: Ejemplo de *JSON* utilizado en *Digital Samos* para una de las geometrías

4.4 *Sprint* 1

4.4.1 Implementación

En este primer *sprint* se desarrollaron las funcionalidades básicas de interacción texto-mapa.

Primeramente, implementamos una función llamada *changeLayerWithText()* en *JS* que permite el cambio automático de la capa que se muestra en el mapa a medida que el usuario avanza la lectura por el texto (ver figura 4.7). Para poder lograrlo el *plugin* necesita la información de las capas entre las que cambiar y las diferentes secciones del texto que involucran un cambio en el mapa. Esta información la tendrá que proporcionar el desarrollador que utilice el *plugin* como variables de entrada a la función. Para ello deberá crear dos listas, la primera contendrá los *id* de las secciones del texto y la segunda los nombres de las capas en el control del mapa. El orden en el que aparecen los elementos de la lista deberá ser el mismo como forma de enlace, es decir, la capa que aparezca como elemento 3 de la lista se relacionará con la sección del texto que ocupe la misma posición en su lista correspondiente.

Esta primera función implementada accede a la lista de capas cuando la sección del texto cambia, mediante la propiedad *scrollTop* de los elementos. Es necesario que en el *HTML* exista un elemento con un *id* asignado en el que se incluya toda la narrativa y definir una variable de tipo *string* cuyo valor sea el mismo que el *id* de la narrativa, ya que del elemento que

contenga el texto relacionaremos la propiedad *.onscroll* con la llamada a la función que se encarga de modificar el mapa. Una vez obtenida la capa que debería de verse en pantalla en ese momento, se procede a hacer uso de las funciones de *Leaflet* para cambiar las capas, estas son *removeLayer* y *addLayer* que se llaman sobre el mapa.

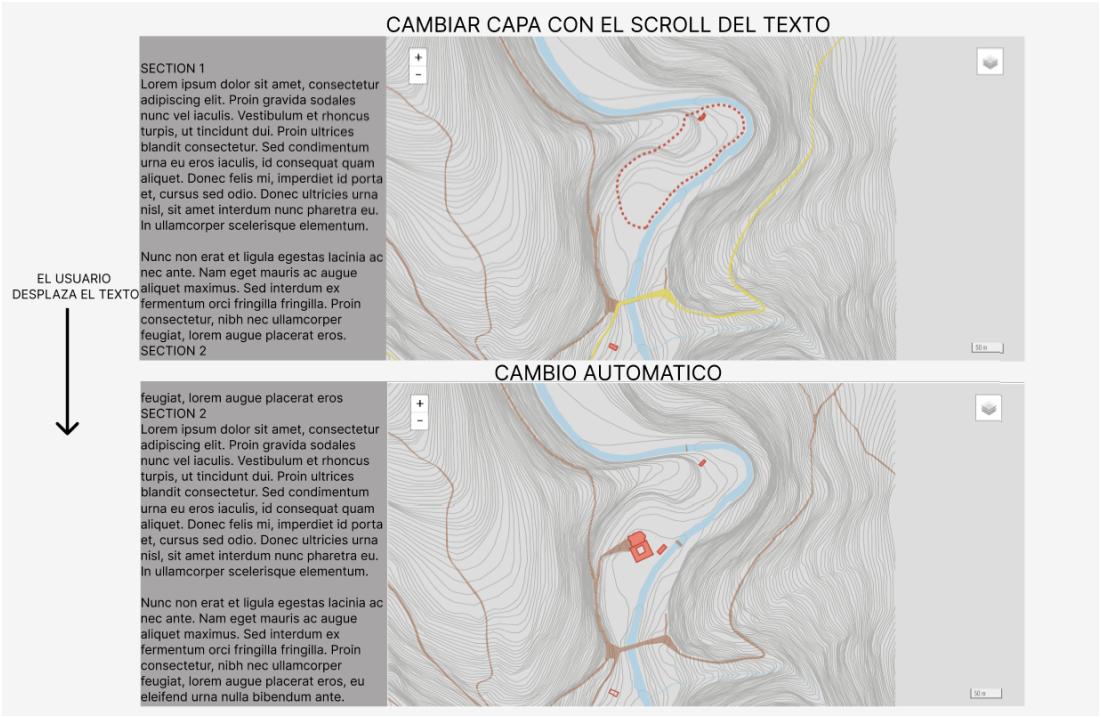

Figura 4.7: *Mockup* del funcionamiento de la primera funcionalidad

Asimismo, hemos desarrollado la función complementaria *changeTextWithLayer()*. Con ella, cuando se modifica la capa actual el texto se ajusta para mostrar correctamente la sección que corresponde a esa capa (ver figura 4.8). Para ello hemos extendido las funciones básicas de *Leaflet* para implementar sobre la clase de control un método que permite obtener la capa actual. Una vez contamos con esta extensión de la herramienta, la función puede recoger el valor actual y acceder a la lista de secciones del texto para obtener de ella su valor de la propiedad *scrollTop* y modificar el texto de manera que se mueva a la posición correcta, es decir, la sección del texto que corresponde a la capa seleccionada por el usuario. Para esta función solo una de las listas mencionadas en la primera función es necesaria, siendo esta la que contiene la información de las secciones del texto, ya que de las capas se encarga el elemento Control de *Leaflet*. Esto se consigue mediante la función *map.on("baselayerchange", function())*, que permite asociar el cambio de una capa con una función.

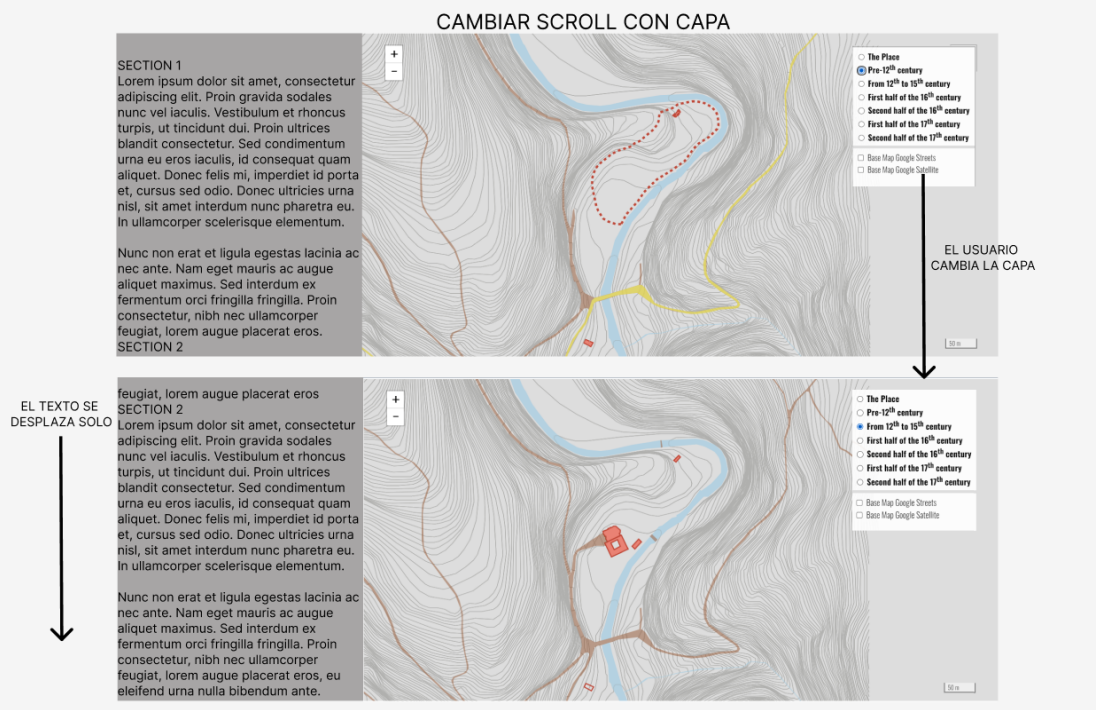

Figura 4.8: *Mockup* del funcionamiento de la segunda funcionalidad

```
L.Control.Layers.include({
    getCurrentLayer: function() {
      // create hash to hold all layers
      var control, layers;
      layers = {};
      control = this;

      let i = 0, result = 0;
      // loop thru all layers in control
      control._layers.forEach(function(obj) {
        // check if layer is an overlay

        if (!obj.overlay) {
            i++;
            if(control._map.hasLayer(obj.layer)){
                return result = i;
            }
        }
      });
      return result;
    }
});
```

Función para obtener capa actual añadida sobre Leaflet

En esta sección del código podemos apreciar la función que se ha desarrollado para ampliar las funcionalidades básicas del control de *Leaflet*. En ella podemos ver como se ha obtenido la capa actual de la siguiente forma: primero se obtienen todas las capas utilizando el control y accediendo a *control._layers* que contiene una lista con todas las capas. Posteriormente recorremos esa lista comprobando si la capa es un *overlay* (capa superpuesta). Hasta el momento no hemos hablado de las diferentes capas que *Leaflet* proporciona. Existen dos tipos de capas, las capas base, que son las que nos interesan en este desarrollo porque contienen la información principal de la narrativa, y las capas de superposición, las cuales pueden estar activas por encima de otra de las capas base. En este código descartamos todos los *overlays* ya que entendemos que la información de los *overlays* es extra para mejorar la experiencia del usuario o la aclaración de cierta información. En el caso de no ser un *overlay*, es decir, cuando la capa sea una capa base, comprobaremos mediante la función *control._map.hasLayer(layer)* si esta se encuentra presente en el mapa. En caso afirmativo guardamos el índice de la capa activa. Al terminar de recorrer todas las capas devolvemos este índice. En *Leaflet* existe un *plugin* que implementa una función parecida a esta disponible en el repositorio Leaflet.ActiveLayer. La función del *plugin* devuelve el nombre, la capa y un valor si es *overlay* o no, pero en nuestro caso, nos interesa conocer el índice de la capa ya que pretendemos enlazarlo con el mismo índice en las secciones del texto.

Cada una de estas funciones se puede utilizar sin depender de la otra, pudiendo contar con solo una de las funcionalidades en la página o ambas.

4.4.2 Pruebas

Para comprobar el funcionamiento de las funcionalidades, estas se añadieron a la web de *Digital Samos* [6]. Se comprobó que cada una de las funcionalidades modificaban correctamente la capa y el texto según las especificaciones. Se probaron primero de forma individual y el funcionamiento era el esperado. Pero surgió un problema cuando se probaron ambas simultáneamente ya que el movimiento del texto de forma suave provocaba cambios en el texto que llamaban a la actualización de las capas, haciendo un bucle infinito de actualizaciones que. en el mejor de los casos, terminaba al llegar el texto al desplazamiento correcto, pero en otras ocasiones, siendo estas la mayoría, el bucle no terminaba nunca y, tanto el texto como el mapa cambiaban constantemente entre dos secciones consecutivas. El problema resulta ser, como se ha mencionado, del comportamiento del texto ante el desplazamiento. Comprobando las opciones, se descubre que existen dos formas de desplazar el texto. En el siguiente *sprint* se tratará dar solución a este problema.

4.5 *Sprint* 2

4.5.1 Implementación

A raíz de la reunión de cierre del primer *sprint* surgen nuevas mejoras para las funciones desarrolladas en ese *sprint*. En la reunión se propone incluir opciones como argumento de las funciones para ampliar su variabilidad de uso y permitir la personalización por parte del desarrollador usuario del comportamiento final. Por lo tanto este segundo *sprint* se centró en desarrollar dichas mejoras. Se añadieron opciones a ambas funciones y se insertaron dentro de una clase llamada *NarrativeIntegration* la cual se sitúa dentro de las clases de *Leaflet* como subclase de *L.Class*. En *Leaflet* es común que todas las clases implementen el método factoría con el mismo nombre en minúsculas (*lowercase*).

```
L.narrativeIntegration = function (){
    return new L.NarrativeIntegration();
}
```
Función factoría

A la primera de las funciones, la que actualiza la capa del mapa en función del *scroll* del texto, se le añadió la opción de poder decidir cómo se actualizará la posición y el zoom del mapa cada vez que se cambie la capa. Por defecto ambas opciones, centrado y zoom, hacen uso de las funciones de *map.getCenter()* y *map.getZoom()* que devuelven las coordenadas del centro y el zoom inicial del mapa. Estas funciones son propias de la librería y tiene su contraparte para modificar el estado del mapa. La modificación de la función *changeLayerWithText()* añade como argumento de entrada unas opciones de zoom y centro. Y en la función se toman estos valores y se llama a las funciones de *Leaflet*, *setZoom(<Number> zoom, <Zoom/pan options> options?)* y *flyTo(<LatLng> latlng, <Number> zoom?, <Zoom/pan options> options?)* para desplazar el mapa.

```
changeTextWithLayer : function (texts, {textScrollBehavior="auto"}={}){
    map.on('baselayerchange',
    function(e) {
        let i = controlLayers.getCurrentLayer() - 1;
        if(!isCLWTactive){
                scrollBehavior = textScrollBehavior;
            }
        document.getElementById(text).scroll({
            top: document.getElementById(texts[i]).offsetTop - 100,
            behavior: scrollBehavior
        });
        section = i;
```

```
    });
},
```

Función changeTextWithLayer()

En el código de la función *changeTextWithLayer()* podemos ver cómo se hace uso de la función de *Leaflet map.on()* que funciona como un evento de JavaScript y realiza una llamada a la función asignada cuando se cumple la condición del evento. En este caso el evento es *baselayerchange* que se llama cada vez que se cambia la capa del mapa. Es importante mencionar que el evento de cambio de la capa no distingue cuando este cambio se realiza por parte del usuario o por parte del código, lo cual genera problemas a la hora de compatibilizar las dos funciones de cambio de mapa y texto. En este caso se obtiene, mediante la función que desarrollamos previamente, el índice de la capa actual. Con este índice accedemos a la lista de los marcadores de las secciones del texto, representadas mediante la variable *texts* para obtener, utilizando las funciones de JavaScript, su posición en el texto mediante la propiedad *offsetTop* que indica la distancia con respecto al inicio del texto. Con este valor desplazamos el texto acorde al *scrollBehavior* (por defecto en *auto*). Utilizamos una variable de control llamada *section* que permite conocer y mantener por separado el valor del índice de la capa.

A la función *changeTextWithLayer()* se le añadió también la posibilidad de incluir opciones como parámetro. En este caso la opción es única, y permite decidir de qué forma se desplazará el texto al cambiar la capa. Esto se realiza mediante la propiedad *scrollBehaviour* que puede llevar dos valores, *auto*, que desplaza instantáneamente, o *smooth* que desplaza el texto de manera suave. Este añadido creó una serie de problemas a la hora de tener ambas funciones activas en nuestra web. El principal problema deriva de la manera en la que las funciones reaccionan a las entradas del usuario.

Cuando un usuario cambia la capa del mapa, si el texto se desplaza de forma *smooth*, el sistema está recibiendo cambios en el texto, por lo que intentará cambiar la capa a la que encaje. Esto provoca la desincronización del mapa y el texto e incluso que pueda llegar a haber movimiento constante en el texto entre dos secciones. En este momento se plantean dos posibles soluciones, dejar al usuario del *plugin* ser conocedor de este problema, para que no se dé el caso, o forzar que en el momento de incluir las dos funciones sobre una misma web, el *scrollBehaviour* del texto sea *auto*. Este comportamiento se logra haciendo que la primera función, en caso de estar presente, modifique el valor del *scrollBehaviour* sobrescribiendo el valor que se haya introducido como opción en la segunda función. De todas formas, aún implementando la segunda, se considera necesario incluir en el *plugin* el aviso de esta problemática para dar a entender que las opciones de la segunda función, en caso de estar las dos presentes, no se tendrán en cuenta.

```
changeLayerWithText: function (texts, layers,
                    {center=map.getCenter(),
                    zoom=map.getZoom()}={}){
    isCLWTactive = true;
    document.getElementById(text).onscroll = function() {
        //leaves section from top, changes to previous section
        if (document.getElementById(text).scrollTop <=
        (document.getElementById(texts[section]).offsetTop - 250)){
            if((section - 1) >= 0){
                map.removeLayer(layers[controlLayers.getCurrentLayer() - 1]);
                section = section - 1;
                scrollBehavior = "smooth"
                map.addLayer(layers[section]);
                map.setView(center, zoom);
            }
        }
        //leaves section from bottom, changes to next section
        else if(document.getElementById(text).scrollTop >=
        (document.getElementById(texts[section + 1]).offsetTop - 80)){
            map.removeLayer(layers[controlLayers.getCurrentLayer() - 1]);
            section = section + 1;
            scrollBehavior = "smooth"
            map.addLayer(layers[section]);
            map.setView(center, zoom);
        }
    };
}
```

Función changeLayerWithText()

En el código de esta función se enlaza un comportamiento al *scroll* del texto. Este comportamiento se divide en dos comprobaciones, saber si al desplazar el texto hemos salido de la sección por la parte superior o por la parte inferior. En el primer caso, debemos cambiar la capa por la capa anterior y en el segundo caso a la capa siguiente. Para ello utilizamos la variable que se comentó en la función anterior de *section* que nos indica en que sección del texto se encuentra el usuario. Para la primera comprobación calculamos si el valor actual del *scroll* del texto es inferior a la posición global del marcador de la sección actual menos un cierto umbral. Para saber si el usuario abandona la sección por la parte inferior, utilizamos como referencia la posición absoluta de la sección siguiente. En ambos casos una vez se cumple la condición se cambia el valor de la sección, se añade la capa nueva y se elimina la capa anterior. Esto provoca que se actualice la capa actual. Cuando el usuario abandona el texto por la parte superior, se entiende que quiere leer la sección anterior y por lo tanto se desplaza el texto completo hasta el inicio de la sección además de modificar la capa.

4.5.2 Pruebas

Al igual que en el *sprint* anterior se han realizado una serie de pruebas para comprobar el funcionamiento de los cambios implementados. Lo primero que se ha realizado es cambiar las llamadas que se hacen desde el *JavaScript* de la web ya que se ha implementado una factoría. En un principio este cambio solo precisa la comprobación de que las llamadas se realizan correctamente. A ambas funciones se les añadió la posibilidad de incluir opciones pasadas como argumentos. Como resultado de las mejoras de las funcionalidades se ha comprobado que ya no existe el problema que presentaba el uso del desplazamiento del texto *smooth* por lo que se consideran completas las funcionalidades. Según las pruebas, en caso de usar la opción *smooth* para la funcionalidad que permite el cambio del texto con el mapa, si la otra funcionalidad está activa, la comprobación que se muestra en el código de abajo será falso y por lo tanto no tendrá en cuenta el valor de la variable *textScrollBehavior* que define la opción del usuario. Y tomará el valor de *scrollBehavior* que es *auto*. Si la otra función no está presente, entonces la condición será cierta y se tomará el valor de la opción.

```
//variable global
let isCLWTactive = false;

//incluido en la función de cambio de texto con el mapa
if(!isCLWTactive){
    scrollBehavior = textScrollBehavior;
}

//incluido en la función de cambio de mapa con el texto
isCLWTactive = true;
```

Solució al problema del desplazamientosmooth

4.6 *Sprint* 3

4.6.1 Implementación

En este tercer *sprint*, se comienza el desarrollo de una nueva función que profundiza más la conexión de la narrativa textual con los elementos (geometrías) del mapa. La idea de esta función es que analice el texto y como resultado, seleccione palabras o frases clave que puedan coincidir con las geometrías. Para ello, disponemos de los archivos *JSON* que podemos obtener de los datos espaciales previamente representados en *QGIS*, ya citados anteriormente, con la información de los elementos que se representan en el mapa. Dentro de uno de estos *JSON* podemos encontrar información relevante como el nombre de la geometría como se puede ver en la figura 4.6.

La función comienza recorriendo el *DOM* (Document Object Model, Modelo de Objetos del Documento en español) y aplicando una expresión regular busca las palabras del texto que coincidan con lo especificado a la entrada de esa función. Como se ha comentado la entrada sería el nombre de la estructura que queremos enlazar. Una vez encontrada en el texto la palabra (o frase) que se pretende enlazar, se envuelve en un *tag* y se le asigna una función cuando el usuario de la página haga clic sobre el texto remarcado. La forma en la que el texto sobresale para indicar que es seleccionable no está definida por el momento, se definirá en el siguiente *sprint*. Por el momento se barajan ideas sobre la visualización mediante estilos, cambiando el color del fondo del texto como en este ejemplo de Neatline (ver figura 2.3) y modificando el tono cuando el usuario sitúe el ratón encima del elemento en cuestión para indicar que se puede hacer clic.

Una vez que el usuario hace clic, se hace una llamada a una función que modifica el estado del mapa para centrarlo en la estructura que el usuario ha seleccionado. Además se abre el *popup* con la información de la estructura y se acerca con un nivel de zoom superior. Esta función está oculta al desarrollador ya que es interna del *plugin* y los datos necesarios para el ajuste del mapa se obtienen de la misma forma que el nombre, a través del *JSON*.

Para poder hacer uso de esta función el desarrollador usuario del *plugin* deberá especificar la ruta de la carpeta que contenga los *JSON* con la información de las geometrías. La función visible para el desarrollador se llama *readJSONData()*.

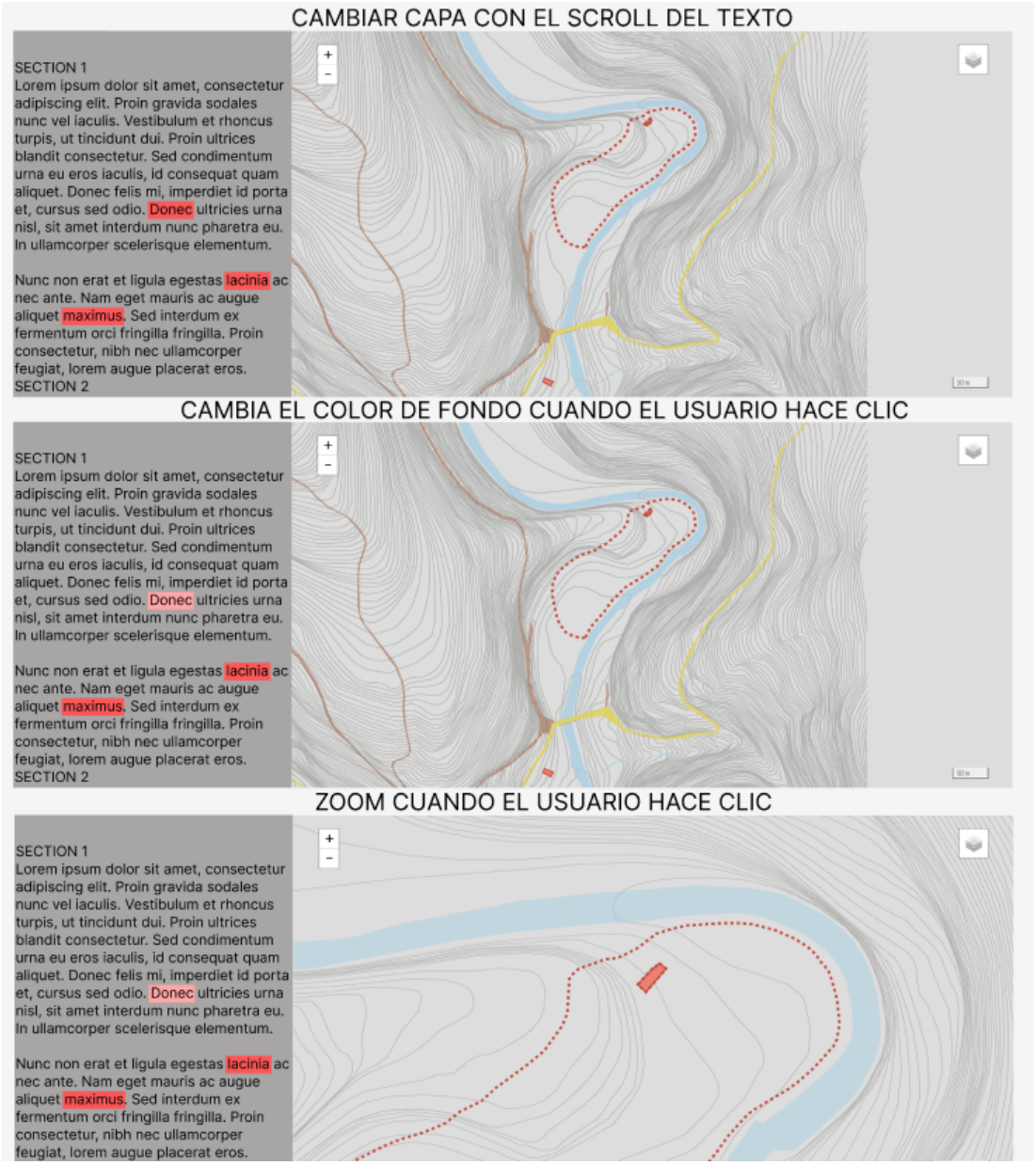

Figura 4.9: *Mockup* del funcionamiento de la tercera funcionalidad

```
function readJsonData(id, dataPath){
    fetch(dataPath)
        .then(res => res.json())
        .then(data => {
            data = data["es"];
            for(i = 0; i< Object.keys(data).length;i++){
                for(j = 0; j< Object.keys(data[i]).length; j++){
```

```
                    searchInText(id, data[i][j])
                }
            }
    })
}
```

Función readJsonData() para lectura de JSON

En el código podemos ver cómo se realiza una petición asíncrona con *fetch* para poder leer el archivo donde se encuentran especificados los nombres de los *JSON* con la información de las geometrías. A continuación se accede a los campos del *JSON* mediante el idioma, en este caso español (es). En futuros *sprints* este valor será una variable que el desarrollador podrá modificar. Por cómo está estructurado el fichero de datos tenemos que recorrer una lista que hace referencia a las capas, y en cada capa, recorremos la lista de las geometrías. Para cada uno de los nombres se llama a la función *searchInText* explicada abajo que realizará la búsqueda en el texto de correspondencias a la geometría.

```
function showInMap(lat, lng){
    map.flyTo([lat, lng], 20);
}
```

Función para mover el mapa al hacer clic en el texto

```
function searchInText (id, path) {
    fetch("/data_json/" + path + ".json")
        .then(res => res.json())
        .then(data => {
            let name = data.features[0].properties.Name;

            let lng = data.features[0].geometry.coordinates[0][0][0][0];
            let lat = data.features[0].geometry.coordinates[0][0][0][1];
            //name = name.normalize("NFD").replace(/[^a-zA-Z ]/g, "");
            name = name.toLowerCase();
            console.log(name);
            html = document.getElementById(id).innerHTML;
            //---Eliminar los spans
            html = html.replace(/<span class="finded">(.*?)<\/span>/g, "$1");

            //---Crear la expresión regular que buscará la palabra
            var reg = new RegExp(name.replace(/[\[\]\(\)\{\}\.\-\?\*\+]/,
            "\\$&"), "gi");
            var htmlreg =
            /<\/?(?:a|b|br|em|font|img|p|span|strong)[^>]*?\/?>/g;

            //---Añadir los spans var array;
            var htmlarray;
            var len = 0;
```

```
            var sum = 0;
            var pad = 28 + name.length;
            let nameClass = name.normalize("NFD").replace(/[^a-zA-Z]/g, "");

            while ((array = reg.exec(html)) != null) {

                htmlarray = htmlreg.exec(html);

                //---Verificar si la búsqueda coincide con una etiqueta html
                if (htmlarray != null && htmlarray.index < array.index &&
                htmlarray.index + htmlarray[0].length > array.index +
                name.length) {
                    reg.lastIndex = htmlarray.index + htmlarray[0].length;
                    continue;
                }

                len = array.index + name.length;
                //html = html.slice(0, array.index) + "<span class='" +
                nameClass + "'>" + html.slice(array.index, len) + "</span>" +
                html.slice(len, html.length);
                //html = html.slice(0, array.index) + "<button type='button'>"
                + html.slice(array.index, len) + "</button>" + html.slice(len,
                html.length);
                html = html.slice(0, array.index) + "<a>" +
                html.slice(array.index, len) + "</a>" + html.slice(len,
                html.length);


                reg.lastIndex += pad;

                if (htmlarray != null) htmlreg.lastIndex = reg.lastIndex;
                sum++;

            }

            document.getElementById(id).innerHTML = html;


            return { total: sum, html: html };

        }
    );

}
```

Función (en progreso) para enlazar el texto con las geometrías

La función descrita en el código anterior corresponde con la estructura principal del funcio-

namiento de la funcionalidad de enlace entre el texto y las geometrías. Como se ha explicado la funcionalidad se divide en tres funciones que leen los datos, modifican el mapa y la que nos atañe en este código, la función que se encarga de obtener la expresión regular y utilizarla para encontrar coincidencias en la narrativa. Para ello lo primero que obtiene es el nombre de los *JSON* que se encuentra bajo *data.features[0].properties.Name*. A continuación, obtiene también los datos que reflejan la posición en el mapa de la geometría, es decir, las coordenadas. El nombre pasa por un proceso de normalización, en este caso se convierte a minúsculas. La función obtiene el *HTML* correspondiente al texto a partir del *id* proporcionado por el desarrollador. Luego se eliminan los posibles *tags* de *span* y se crea la expresión regular. Para poder manejar las clases que se van a crear la función obtiene un nombre de clase a partir del nombre de la geometría. A continuación procede a recorrer el texto con la expresión regular para encontrar similitudes y marcarlas con un *tag* <a>. Al terminar modifica el *HTML* actual con el nuevo que contiene las clases identificadas.

4.6.2 Pruebas

Como se ha comentado la funcionalidad aún no está completa, pero podemos apreciar los primeros indicios de funcionamiento. En este caso se ha comprobado que realiza de forma correctas las búsquedas modificando el *CSS* manualmente con uno de los nombres de las clases para que le cambiase el color de fondo. Como resultado obtenemos algunas palabras en el texto remarcadas con un color de fondo (ver figura 4.10).

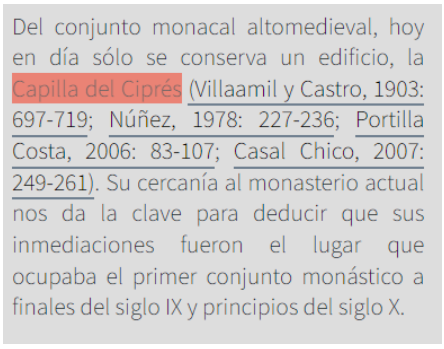

Figura 4.10: Resultado de la prueba de uso de la tercera funcionalidad.

En el siguiente *sprint* se decidirá la forma en la que se visualizarán los elementos resaltados del texto mediante el código, ya que por el momento coexisten tres posibles soluciones de visualización.

4.7 *Sprint* 4

4.7.1 Implementación

El cuarto y último *sprint* se centra en corregir y mejorar la función que permite al desarrollador enlazar el texto con las geometrías del mapa. Esta función ha sido la más compleja debido a la dependencia que todas las subtareas tenían entre sí.

Primero se desarrolló una función que permite analizar el *DOM* mediante el uso de expresiones regulares. La función obtiene el valor del HTML y realiza un *parsing* de la expresión regular correspondiente. Esta expresión regular viene dada por la estructura que se está intentado enlazar con el texto. La obtención de este valor plantea el primero de los problemas de la función, ya que no existe una manera de leer el directorio actual en busca de archivos con un formato sin conocer la ruta exacta de ese archivo. Además en una primera instancia, la información de las geometrías no estaba contenida en *JSON* como se ha comentado anteriormente, sino que este cambio se realizó pensando en esta función.

Como solución se plantea modificar los datos de las geometrías a formato *JSON* y guardarlas bajo un directorio común. Esto nos dará acceso a la información mediante la función asíncrona *fetch* de la cual nos quedaremos con el nombre de la estructura, que será la expresión regular y con las primeras coordenadas que aparezcan, que corresponderán con alguno de los vértices de los polígonos. El *plugin* necesita que el directorio común tenga de nombre *data_json* para poder leer los archivos. Para ello es necesario que el desarrollador cree un *JSON* extra que contenga los nombre de los archivos que haya en el directorio y se quieran enlazar, pero sin la extensión *.json*, el *plugin* ya asume dicha extensión al hacer la búsqueda. Para poder leer este nuevo archivo se desarrolló una función llamada *readJSONData()*, que recibe como parámetros el *id* del *HTML* sobre el que se va a buscar y la ruta del nuevo archivo *JSON*.

Con el estado actual de la función podemos leer mediante un par de bucles todos los archivos *JSON* especificados y extraer de ellos la información de *parsing*. También en estos instantes se puede realizar el *parsing* y modificar así el formato del HTML. Aquí surgen varias ideas a futuro ya que no solo queremos remarcar ese texto, sino que la intención es que o bien cuando el usuario mueva el ratón por encima o bien haga clic sobre el texto, el mapa se vea afectado en un movimiento hacia la estructura correspondiente.

Para ello se plantearon tres posibles cambios sobre el texto seleccionado, el primero consiste en convertir el texto a un botón y asignar la función auxiliar de movimiento del mapa a la opción *onclick* del botón, la segunda consiste en envolver el texto en una *tag* del tipo y asignarle una clase con el nombre que se obtuvo del *JSON* para poder modificar en el CSS el formato de esa clase y añadirle un evento de *JS* (*eventListener)* asignando la función auxiliar al evento *click* y por último y la opción implementada, envolver el texto en una *tag*

<a> al cual se le aplicaría una clase como la explicada en la opción 2 y se le asignaría, también de la misma forma un evento del tipo *click*.

```
fetch(dataPath)
    .then(res => res.json())
    .then(data => {
        //resto del código
    })
```

Cabecera función fetch

```
html = html.slice(0, array.index) + "<span class='" + nameClass + "'>" +
html.slice(array.index, len) + "</span>" + html.slice(len, html.length);

html = html.slice(0, array.index) + "<button type='button'>" +
html.slice(array.index,len) + "</button>" + html.slice(len, html.length);

html = html.slice(0, array.index) + "<a class='" + nameClass + "'>" +
html.slice(array.index, len) + "</a>" + html.slice(len, html.length);
```

Opciones de modificación del texto para enlazar

Por la manera en la que la función analiza el texto surge un segundo problema al añadir el evento a cada una de las apariciones del texto, ya que en *HTML* no se puede añadir directamente el código *onclick=foo(a, b)* ya que los parámetros (a y b en este ejemplo) no pueden ser variables. Por lo tanto se plantea hacer una búsqueda sobre las clases una vez modificado el *DOM* y añadir en este instante el evento. En una primera instancia se añade esta sección del código al terminar de analizar y encontrar en el texto todas las coincidencias. Al probar esta idea nos encontramos con que sólo la última de las clases buscadas tiene enlazada la función al evento. Según un breve análisis del código, se llega a la conclusión de que es debido a que cada una de las búsquedas lee todo el *HTML*, y lo que no sea una coincidencia lo escribe como se encuentre en el *DOM*. Esto lleva a que las funciones enlazadas, al no estar presentes en el *DOM* no se mantengan entre iteraciones de la función.

Como primera solución se plantea el mover la búsqueda de las clases a otra parte de la función en la que ya se haya modificado todo el *DOM* de manera correcta, pero siendo la función asíncrona, no hay una parte dentro de la función segura donde ya haya acabado de realizar el *parsing* de todos los elementos. Para tratar de solucionar este problema se realizan unas búsquedas de información sobre funciones asíncronas encontrando que *JS* permite el uso de promesas. Una *Promise* (promesa en castellano) es un objeto que representa la terminación o el fracaso de una operación asíncrona [23]. Al realizar múltiples llamadas a la función asíncrona, necesitaremos múltiples promesas. Cada promesa esperará el valor de nombre de la clase, longitud y latitud, necesarios para el evento *click*. Como se puede observar en el código

adjunto a continuación se utiliza la función *Promise.all(promises)* que espera a que todas se completen o alguna falle, luego obtiene los valores de todas las promesas en una lista y se accede a ella mediante un bucle *for*. Para poder modificar cada una de las clases, obtenemos el elemento mediante la función *document.getElementsByClassName(className)* que devuelve todos los elementos que pertenezcan a la clase *className*. Luego recorremos todos los elementos y modificamos su estilo, cambiando el fondo y añadiendo el evento. La función es imprecisa en la búsqueda ya que intenta encontrar la pareja perfecta en el texto con el nombre exacto de la estructura.

```
Promise.all(promises).then(values =>{
    for(let i= 0; i < values.length;i++){
        console.log(values);
        classes = document.getElementsByClassName(values[i][0]);
        for (let j = 0; j < classes.length; j++) {
            classes[j].style.background = "#EF5B47";
            classes[j].style.opacity = 0.7;
            classes[j].addEventListener("click",
                function(){map.flyTo([values[i][1], values[i][2]], 20);});
        }
    }

});
```

Código realizado al cumplirse las promesas

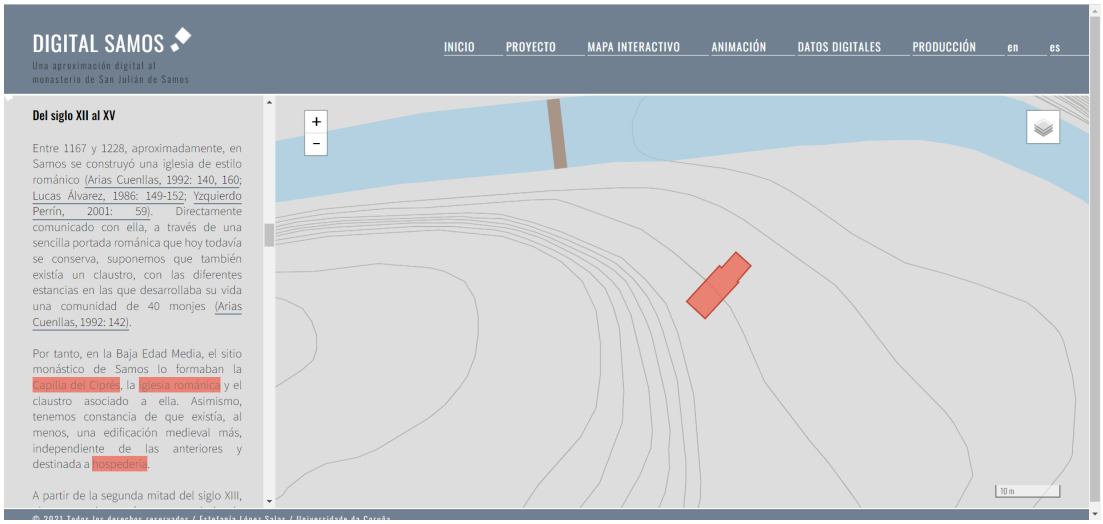

Figura 4.11: Web de *Digital Samos* tras los cambios realizados con el *plugin*.

Por último en este *sprint* se ha creado un archivo *variables.js* que contiene las variables modificables del *plugin* para dar más facilidades al desarrollador a personalizar algunos as-

pectos de las funcionalidades. Entre ellas se encuentran las que se muestran en la figura 4.12 que son:

- **scrollBehaviour**: permite configurar el comportamiento por defecto del desplazamiento del texto cuando el usuario cambia la capa.
- **textBgColor**: modifica el color de fondo de las palabras o frases seleccionadas por la función de emparejamiento.
- **textBgHighlightColor**: modifica el color de fondo de las palabras o frases seleccionadas por la función de emparejamiento cuando el usuario pasa el cursor por encima.

```
1
2  //scrollBehavior for both active functions
3  let scrollBehavior = "auto";
4
5  //Text elements found in readJSONData function that are connected to map elements
6  let textBgColor = "#EF5B47";
7  let textBgHightlightColor = "#ed978c";
8
```

Figura 4.12: Variables incluidas en *variables.js* para personalizar las funciones.

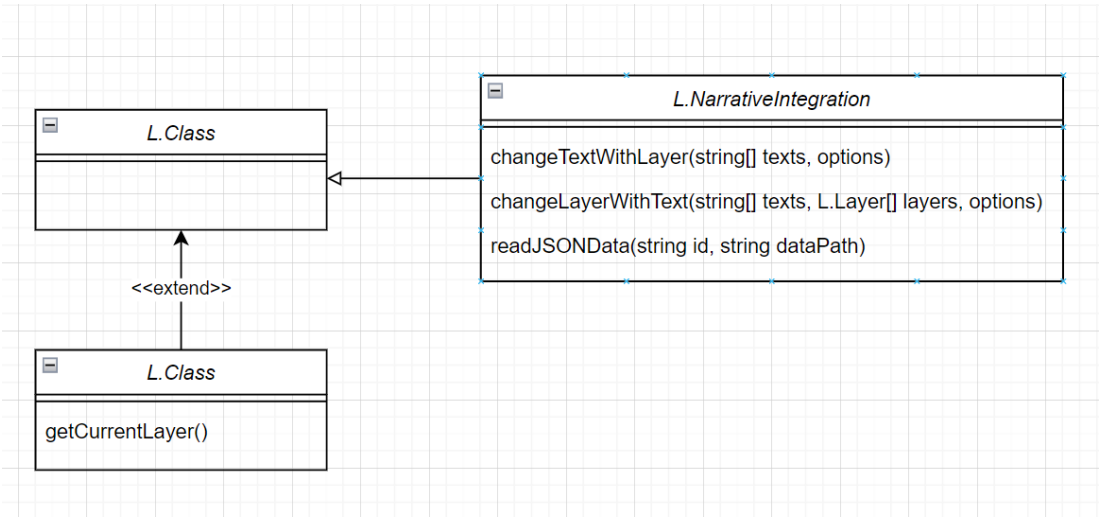

Figura 4.13: Clase desarrollada del *plugin* añadida a *Leaflet* en el *plugin*

4.7.2 Pruebas

En este *sprint* se realizaron las pruebas finales para cerrar el desarrollo del proyecto y el *plugin* implementado. Se ha comenzado probando sobre una web real (*Digital Samos*) que

funcionan todas las funcionalidades correctamente de manera independiente y de manera conjunta. En especial las pruebas se centraron en la tercera de las funcionalidades ya que las otras dos ya habían sido probadas en anteriores *sprints*. La tercera de las funcionalidades es capaz de reconocer y leer todos los archivos que se presentan en el *data.json* y se ha comprobado mediante la consola. Podemos apreciar que en el caso concreto que nos atañe algunos de los nombres de las geometrías son muy específicos y difícilmente se encontrarán en el texto de manera exacta. Esto es algo con lo que ya se contaba durante el desarrollo del proyecto y que será presentado como una posible mejora de cara al trabajo futuro. Por otro lado, durante el desarrollo se detectó que las búsquedas que realizaba la función modificaban los resultados de las anteriores, quedando útiles las últimas búsquedas. Esto se solventó usando las *Promesas* que se han implementado en este *sprint*. Esta funcionalidad ha sido la más compleja del desarrollo y, por lo tanto, la que ha conllevado más errores y fallos de funcionamiento. Una vez solucionados los problemas encontrados, al cierre de este *sprint* se considera que la funcionalidad base es completa y correcta, siendo validada por los clientes finales (en este caso los directores del proyecto).

4.8 *Sprint* 5: Generación *plugin*

Al igual que la primera sección de la parte de desarrollo (el *sprint* 0 4.3) esta sección corresponde a un quinto *sprint* de cierre de proyecto de duración inferior al resto de *sprints*.

La herramienta de *Leaflet* está abierta para modificación mediante *plugins* y para ello ofrecen una guía sobre cómo desarrollar tu propio *plugin* [24]. En esta última sección se comentarán los cambios que se han tenido que realizar al *plugin* implementado en el *sprint* anterior para que entre dentro de los estándares de *Leaflet* y que pueda formar parte de la librería.

En primer lugar deberemos crear un repositorio para el *plugin* y para ello utilizaremos la herramienta de Github [10]. Una vez subidos los archivos base del *plugin* el siguiente paso era crear un *README.md* para explicar al usuario las funcionalidades que aporta el *plugin* y cómo usarlo.

En la guía de publicación de *plugins* se presentan una serie de *code conventions* (convenciones de código) para la mejora del *plugin* dentro de los estándares de *Leaflet*.

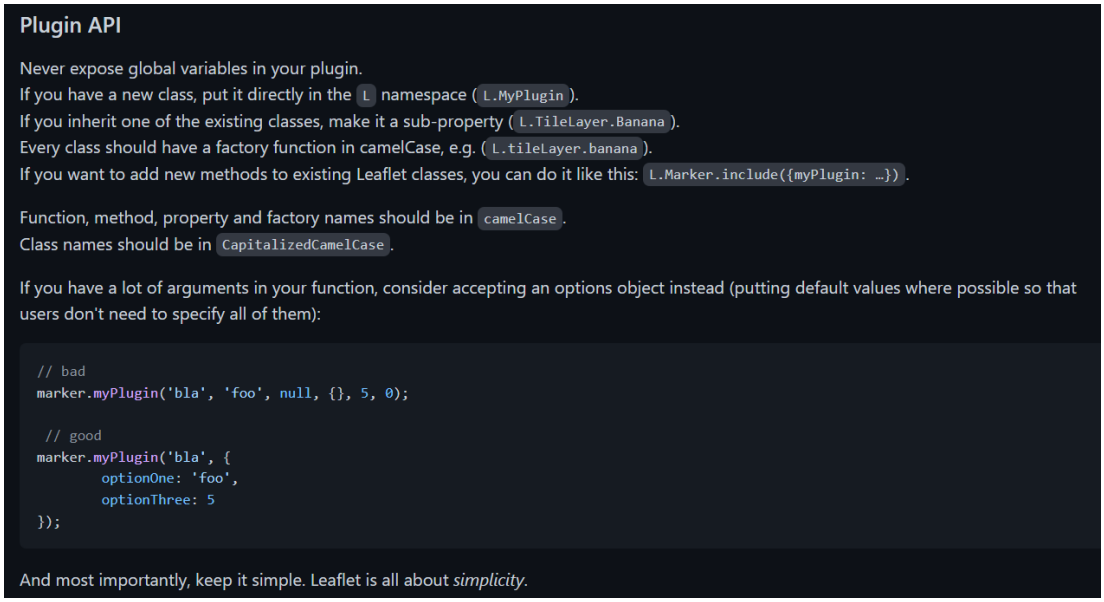

Figura 4.14: Convenciones de código propuestas por *Leaflet* para la creación del *plugin*

Como se puede apreciar en la figura 4.14 entre las convenciones encontramos las siguientes:

- Nunca exponer variables globales en el *plugin*.
- Si tienes una clase nueva, añádela directamente bajo el *namespace* (espacio de nombre) L (ej. L.MiPlugin).
- Si heredas de una de las clases existentes, conviértelo en una sub-propiedad.

- Toda clase deberá tener una factoría en *camelCase* (ej. L.miPlugin).
- Si quieres añadir nuevos métodos a clases de *Leaflet* ya existentes, puedes hacerlo de la siguiente forma L.Class.include(miPlugin: ...).
- Los nombres de funciones, métodos, propiedades y factorías deberán ir en *camelCase*.
- Los nombres de las clases deberán ir en *CapitalizedCamelCase*.
- Si tienes muchos argumentos en una clase, considera aceptar opciones en su lugar (poniendo valores por defecto para reducir el número de variables que tiene que especificar el usuario)
- Y lo más importante, mantenlo simple. *Leaflet* se basa en la simplicidad.

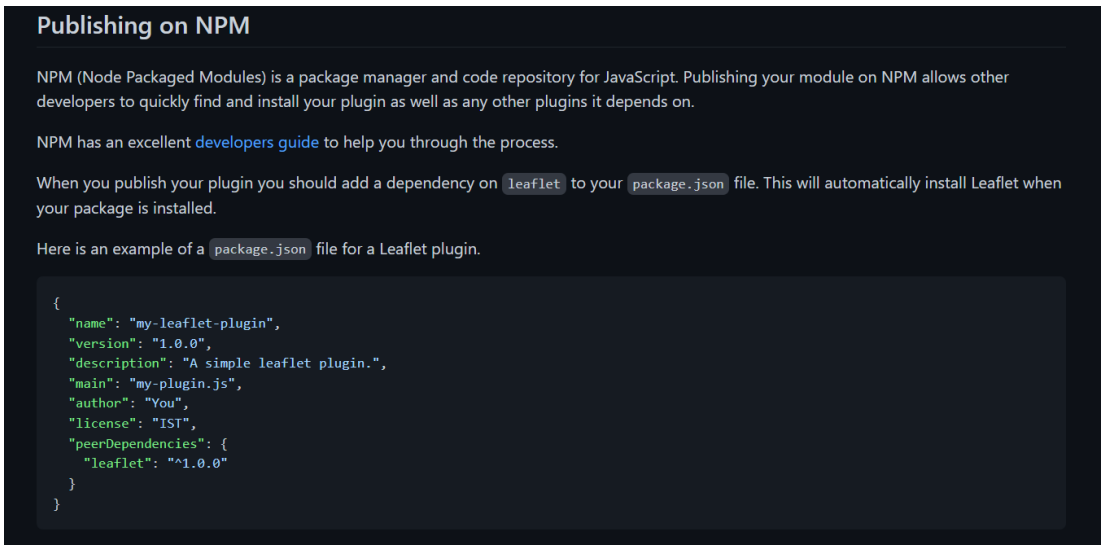

Figura 4.15: Normas de publicación del *plugin* mediante NPM

A continuación vendrá la publicación del *plugin* mediante *npm* [18] como se puede ver en la figura 4.15. NPM es un administrador de paquetes y repositorio de código para JavaScript. Publicar tu módulo en *npm* permite a otros desarrolladores encontrar e instalar rápidamente tanto tu *plugin* como las dependencias. El administrador de paquetes *npm* cuenta con una guía de desarrolladores para ayudarte en el proceso. En el caso que nos refiere, debemos añadir la dependencia de *Leaflet* al archivo *package.json*. Esta dependencia instalará de manera automática *Leaflet* cuando se instale tu paquete.

En la figura 4.15 podemos apreciar también un ejemplo de *package.json* para un *plugin* de *Leaflet*.

En la figura 4.16 podemos apreciar las instrucciones para añadir nuestro *plugin* a la librería *Leaflet*. Para ello debemos:

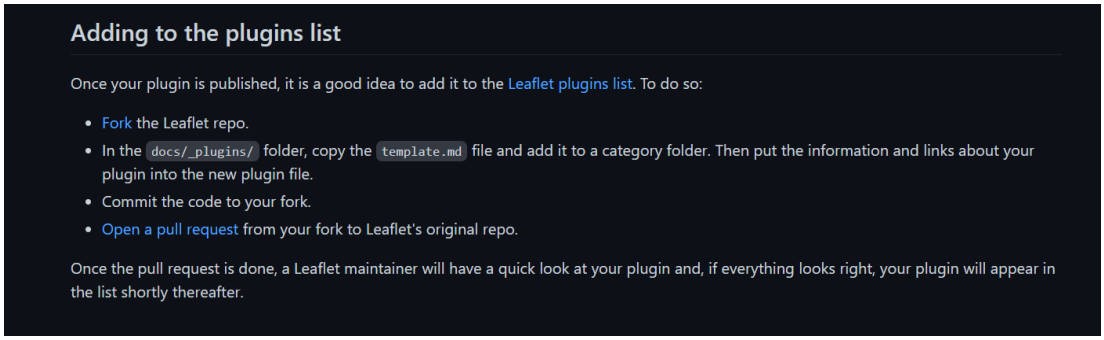

Figura 4.16: Instrucciones de cómo añadir el *plugin* a *Leaflet*

- Hacer una copia del repositorio de *Leaflet* (*fork*).
- En el directorio *docs/_plugins/*, copiar el archivo *template.md* y añadirlo a una carpeta de categoría. Luego añadir la información y links de tu *plugin* en el nuevo fichero de *plugin*.
- Subir los cambios a tu copia del repositorio (el *fork*).
- Crear una petición de extracción (*pull request*) de tu copia del repositorio al repositorio original.

Si todos los pasos se han seguido correctamente, un mantenedor de *Leaflet* mirará el *plugin* y aceptará la petición. Cuando se termine el proceso tu *plugin* formará parte de *Leaflet*.

4.9 Entorno de pruebas y resultados

En esta sección se presentan los resultados obtenidos al aplicar las funcionalidades desarrolladas en el *plugin* en el entorno de pruebas real de la web de *Digital Samos*.

4.9.1 Cambio de texto automático con la capa

Como se puede apreciar en la figura 4.18 el contenido del texto presente en la figura 4.17 se modifica para ajustarse a la capa seleccionada. Esto se corresponde a la primera de las funcionalidades que actualiza el valor del desplazamiento del texto en función de la capa seleccionada. En este caso el usuario ha seleccionado la capa de la primera mitad del siglo XVI y se corresponde al texto mostrado en la parte izquierda de la pantalla.

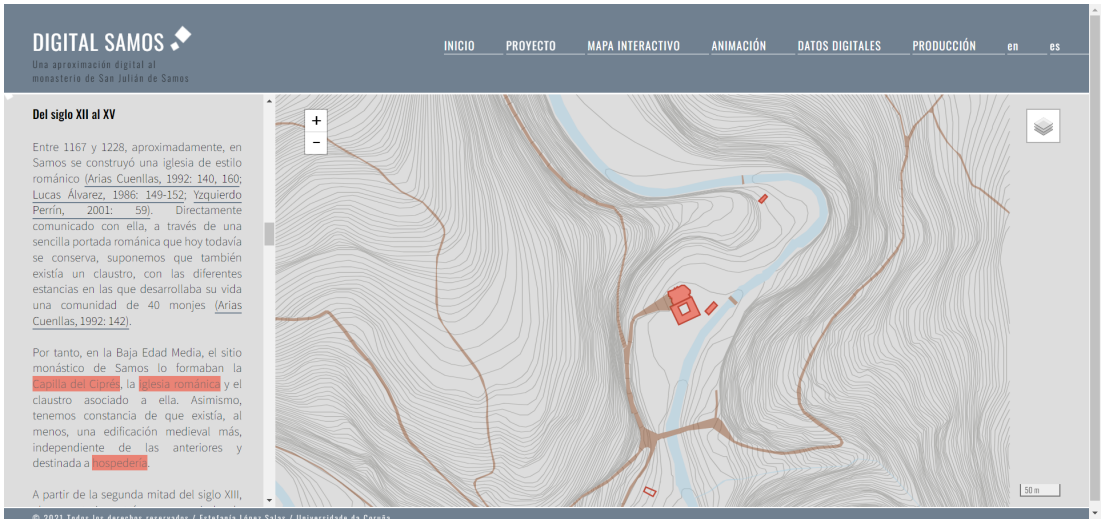

Figura 4.17: Web *Digital Samos* tras la incorporación de las funcionalidades del *plugin*

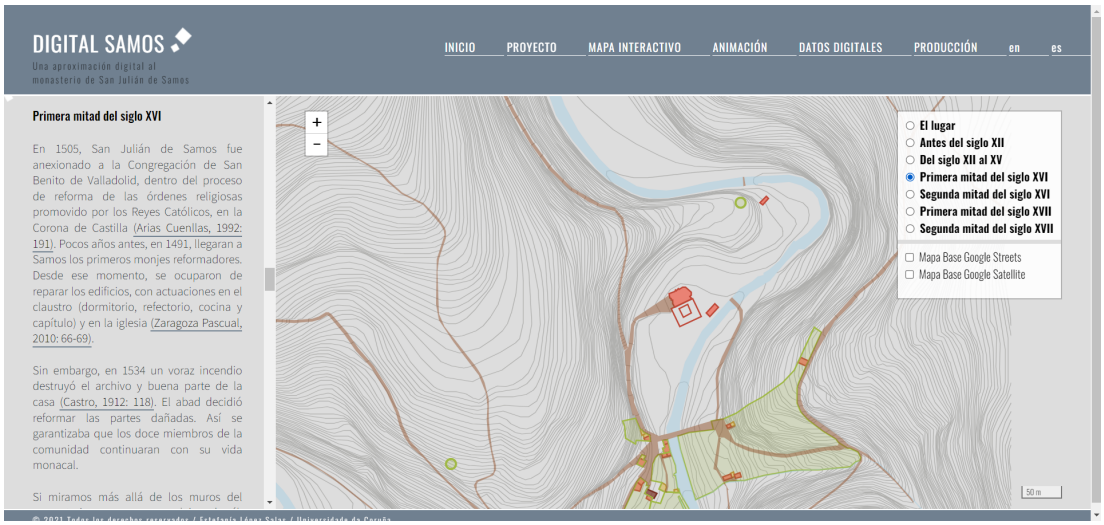

Figura 4.18: Ejemplo de la funcionalidad que cambia el texto automáticamente cuando se cambia la capa

4.9.2 Cambiar el mapa con el desplazamiento del texto

En esta sección se puede apreciar como al desplazar el texto como en la figura 4.19 y superar el umbral de texto inferior, en este caso la primera mitad del siglo XVI la capa que se muestra en la figura 4.20 coincide con la sección del texto. Podemos comprobar que el cambio de la capa es correcto viendo la figura 4.18 donde el usuario seleccionó la misma capa. En la figura 4.21 el usuario desplaza el texto hacia arriba y la capa se modifica de acorde a este desplazamiento.

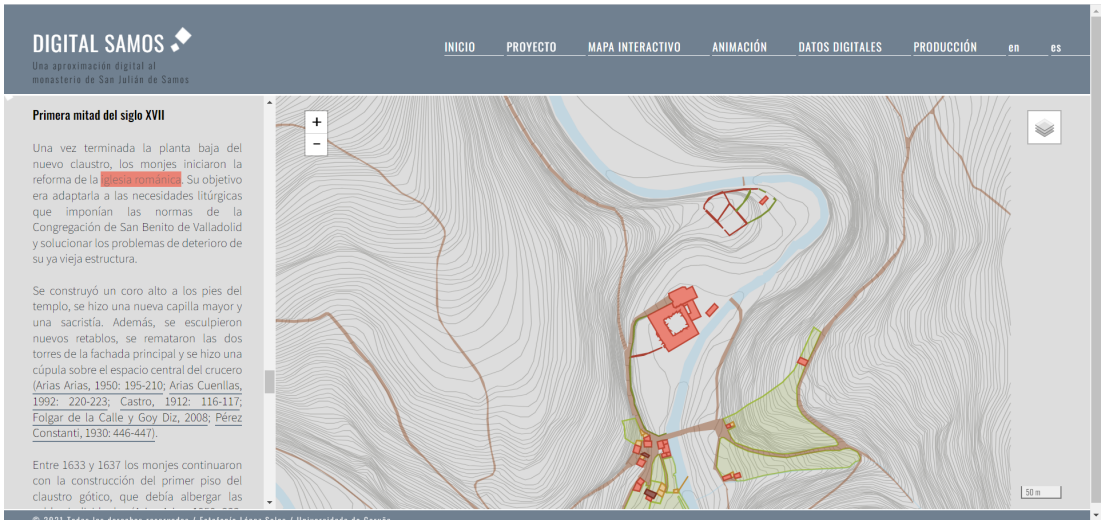

Figura 4.19: Ejemplo de la funcionalidad que cambia el mapa cuando el usuario abandona la narrativa por la parte inferior

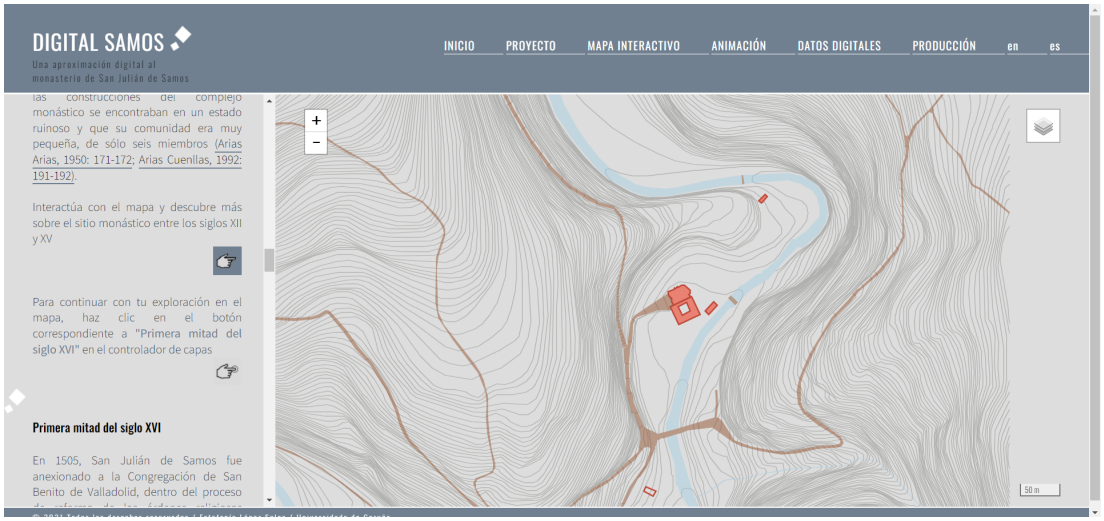

Figura 4.20: Ejemplo de la funcionalidad que cambia el mapa cuando el usuario abandona la narrativa por la parte superior

4.9.3 Enlace texto-mapa vía expresiones regulares

En esta sección podemos ver cómo ha resultado la página tras aplicar la tercera funcionalidad. En el texto de la izquierda de la figura 4.22 aparecen palabras y frases remarcadas en el color de las geometrías que indican que esos elementos tienen una correspondencia en el mapa. Cuando el usuario pasa el ratón por encima cambia el color de fondo y si hace clic el mapa hace zoom en la geometría como ha sido el caso de la figura.

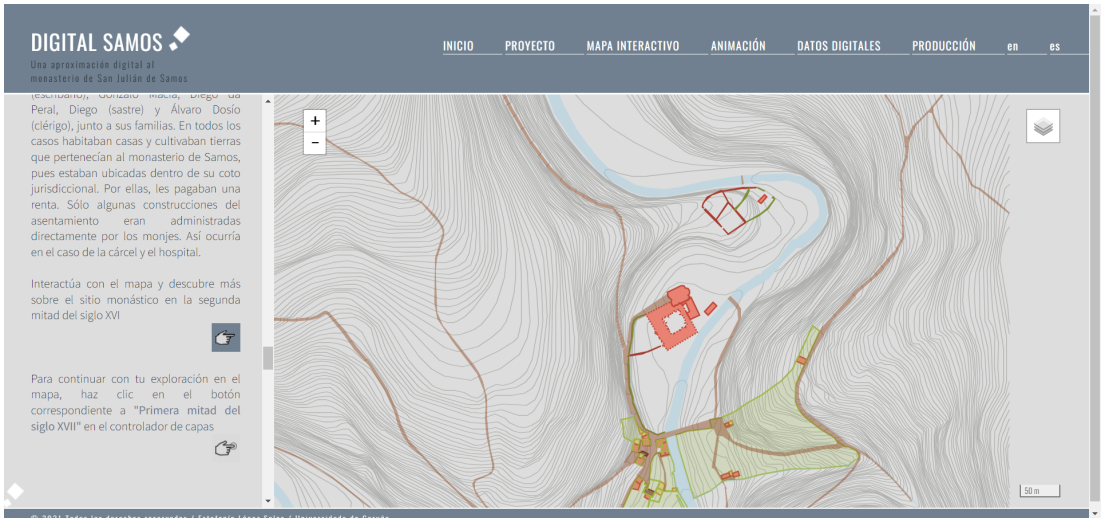

Figura 4.21: En este ejemplo el texto se está desplazando a la posición correcta mientras que la capa ya es la correcta

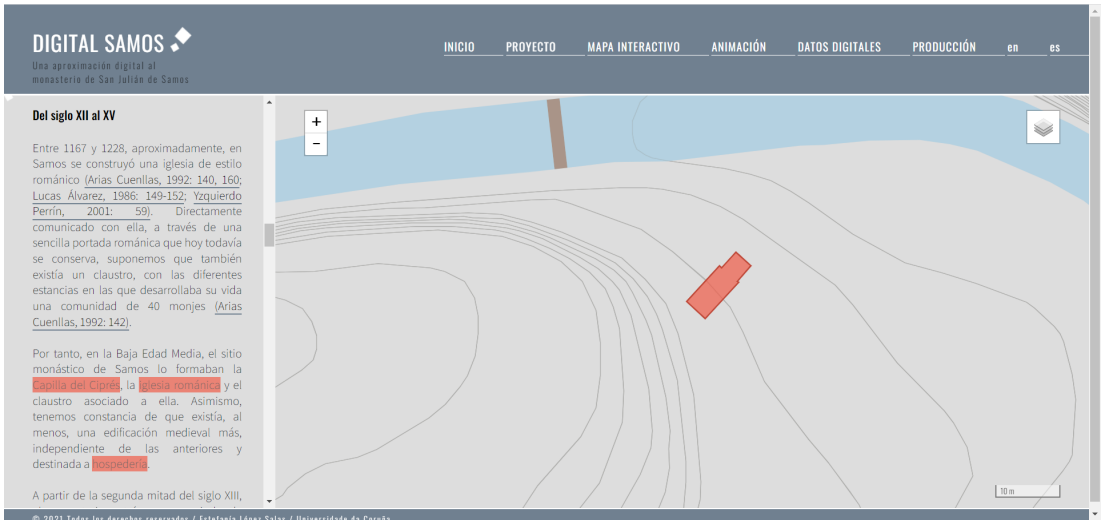

Figura 4.22: Web de *Digital Samos* tras los cambios realizados con el *plugin*.

Capítulo 5

Conclusiones y líneas de trabajo futuro

En este último capítulo de la memoria se presentará la situación final del TFG, las lecciones aprendidas, las relaciones del proyecto con las competencias adquiridas durante la titulación en general y la mención en particular y posibles ampliaciones o líneas futuras de trabajo sobre el proyecto.

5.1 Conclusiones

A lo largo del desarrollo del proyecto se han logrado cumplir los objetivos propuestos. El objetivo principal del proyecto era obtener una herramienta que permitiese al desarrollador mejorar la integración de la narrativa con el mapa. Esto se ha logrado mediante un *plugin* para la librería *Leaflet*. El *plugin* final ha resultado en una herramienta abierta, generalizable y fácilmente adaptable a múltiples entornos, siendo prueba de ello su uso en una web real como *Digital Samos*. A continuación se detallan las funcionalidades, marcadas como objetivos secundarios, que se han conseguido con la implementación de este proyecto:

- Se ha desarrollado una funcionalidad que modifica la capa del mapa que se está visualizando actualmente mediante las funciones de *Leaflet* cuando el usuario desplaza el texto, usando de referencia, en el proyecto concreto, los títulos de las secciones. Las referencias son asignadas por el desarrollador en la llamada a la función.
- Se ha desarrollado una funcionalidad que desplaza el texto cuando un usuario cambia manualmente la capa que se muestra en el mapa mediante las opciones del control de capas de *Leaflet*. Al igual que la función anterior toma referencias en el texto que son propuestas por el desarrollador en la llamada a la función.

- Se ha desarrollado una funcionalidad que permite resaltar elementos en el texto que coincidan con geometrías (o elementos) en el mapa y que al hacer clic sobre ellos se modifique el estado del mapa (zoom y centro) para ajustarse al elemento al que hace referencia el texto.

Estas funcionalidades pueden ser aplicadas de forma individual generando una integración parcial o de manera conjunta. Todas las funcionalidades son ampliables y adaptables mediante variables de control. Esto hace que el *plugin* sea genérico y utilizable en diversos ámbitos.

5.2 Lecciones aprendidas

El desarrollo de este TFG me ha permitido conocer la complejidad de un proyecto real, tanto por la planificación y cumplimiento de los plazos asignados como por el propio desarrollo del proyecto. En especial la aplicación de una metodología ágil como *Scrum* ha facilitado el desarrollo y la organización del proyecto.

He aprendido el funcionamiento de la librería *Leaflet* para la creación y modificación de mapas web interactivos. *Leaflet* ofrece diversas características para la creación de mapas y es por ello que se usa en la web sobre la que se empezó a desarrollar el proyecto. Esto me ha llevado a tener que aprender el funcionamiento de la herramienta antes de comenzar con el desarrollo. La idea de generalizar la herramienta surge al conocer las facilidades que *Leaflet* ofrece para su ampliación mediante funcionalidades en forma de *plugin*. Eso ha derivado en el aprendizaje de cómo está estructurada la librería y cómo ampliar las funcionalidades ya existentes.

He mejorado mi conocimiento básico sobre desarrollo web adquirido durante el grado, tanto en *HTML* como en *JS* para poder generar una interfaz de visualización avanzada capaz de vincular, a lo ojos del usuario, la información textual y gráfica.

5.3 Competencias

En el desarrollo del proyecto se han tocado diversos temas entre los cuales destacar las competencias que figuran a continuación:

- Capacidad para diseñar, desarrollar, seleccionar y evaluar aplicaciones y sistemas informáticos, asegurando su fiabilidad, seguridad y calidad, conforme a principios éticos y a la legislación y normativa vigente.
- Capacidad para planificar, concebir, desplegar y dirigir proyectos, servicios y sistemas informáticos en todos los ámbitos, liderando su puesta en marcha y su mejora continua y valorando su impacto económico y social.

- Capacidad para comprender la importancia de la negociación, los hábitos de trabajo efectivos, el liderazgo y las habilidades de comunicación en todos los entornos de desarrollo de software.
- Conocimiento, administración y mantenimiento de sistemas, servicios y aplicaciones informáticas.
- Conocimiento, diseño y utilización de forma eficiente de los tipos y estructuras de datos más adecuados a la resolución de un problema.
- Conocimiento y aplicación de los procedimientos algorítmicos básicos de las tecnologías informáticas para diseñar soluciones a problemas, analizando la idoneidad y complejidad de los algoritmos propuestos.
- Conocimiento y aplicación de los principios fundamentales y técnicas básicas de la programación paralela, concurrente, distribuida y de tiempo real.
- Conocimiento y aplicación de los principios, metodologías y ciclos de vida de la ingeniería de software.
- Capacidad para diseñar y evaluar interfaces persona computador que garanticen la accesibilidad y usabilidad a los sistemas, servicios y aplicaciones informáticas.
- Capacidad para desarrollar, mantener y evaluar servicios y sistemas software que satisfagan todos los requisitos del usuario y se comporten de forma fiable y eficiente, sean asequibles de desarrollar y mantener, y cumplan normas de calidad, aplicando las teorías, principios, métodos y prácticas de la ingeniería del software.
- Capacidad para conocer los fundamentos teóricos de los lenguajes de programación y las técnicas de procesamiento léxico, sintáctico y semántico asociadas, y saber aplicarlas para la creación, diseño y procesamiento de lenguajes.

5.4 Trabajo futuro

Una vez finalizado el proyecto, aunque se considera que los objetivos planteados se han cumplido satisfactoriamente, sería interesante continuar ampliando las funcionalidades implementadas. Así, a continuación se proponen las siguientes líneas de trabajo futuro:

- En primer lugar la función que se encarga de enlazar los elementos del texto con el mapa, hace las búsquedas de manera exacta, por lo que algunas de las geometrías no se ven representadas en el texto con el mismo nombre. Se podrían modificar las expresiones regulares de forma que las búsquedas hiciesen un *ranking* por similitud y si supera

cierto umbral considerar que ese elemento del texto se puede enlazar con la geometría. Sobre esta misma función se podrían mejorar los enlaces texto-mapa añadiendo una línea que una los elementos cuando el usuario pase el ratón por encima del texto. Por el momento el evento de pasar el ratón está asignado a modificar el color de fondo y dar a entender al usuario que se puede hacer clic sobre el elemento. Para poder unir las líneas se podría utilizar la librería *LeaderLine* [25].

- Otro de los puntos donde se podría continuar trabajando sería en la lectura de los datos de las geometrías a partir de los *JSON*. En el estado actual del proyecto, el *plugin* permite una lectura organizada por capas. Sin embargo, la búsqueda se realiza sobre todo el texto sin tener en cuenta la sección ni la capa a la que pertenece la geometría. Esto da lugar a dos problemas. El primero de ellos la ineficacia de las búsquedas en aquellos casos en los que ciertas geometrías que interese referenciar están presentes en una capa y no en otra. El segundo es el caso contrario, es decir, la posibilidad de que se produzcan coincidencias en otra sección del texto entre referencias como consecuencia de la variabilidad de palabras y expresiones que se pueden utilizar en una narrativa. Ambos problemas se solucionarían restringiendo la búsqueda a una única capa.

Apéndices

Apéndice A

Guía del usuario

Este apéndice presenta la guía del *plugin* para el desarrollador usuario de *Leaflet*.

A.1 Instalación

El plugin estará disponible para descarga bajo la página de *Github* del proyecto [26] además de la página oficial de *Leaflet* [5].

Una vez descargado bastará con incluir los archivos *narrativeIntegration.js* y *variables.js* en el directorio de la página web y añadirlos a los *HTML* que vayan a hacer uso de las funciones mediante el siguiente código:

```
<script type="text/javascript" src="variables.js"></script>
<script type="text/javascript" src="narrativeIntegration.js"></script>
```

Código de inclusión de scripts a HTML

Es importante remarcar que el *script* de las *variables* se debe declarar antes que el *script* de *narrativeIntegration* para el correcto funcionamiento del *plugin*.

A.2 API

En esta sección se muestran las funcionalidades del *plugin* a modo de *API* en el cuadro A.1. En él podremos ver la firma de la función, el valor de retorno y la descripción tal y como aparece en el repositorio de Github.

Method	Returns	Description
changeTextWithLayer(<String array> texts, <String> textScrollBehaviour)	void	Allows users to change the text shown on screen when the layer is changed on the map control
changeLayerWithText(<String array> texts, <Layer array> layers, <Number/Number options> center/zoom)	void	Allows users to change the layer when the text is scrolled while reading.
readJSONData(<String> id, <String> dataPath)	void	Allows users to link text data with map geometries via regular expressions

Tabla A.1: API methods

A.3 Ejemplo de uso

En esta sección utilizaré la web de *Digital Samos* (ver figura 4.5) sobre la que fue desarrollado inicialmente el *plugin* como ejemplo de uso. Se mostrarán los detalles del proyecto que serán necesarios en cualquier otra aplicación del *plugin* para su correcto funcionamiento, basándonos en los cambios realizados sobre Digital Samos.

Como podemos apreciar en la figura A.1 se han añadido bajo el directorio principal los dos archivos marcados en amarillo, correspondientes al *plugin*. Estos contienen, como ya se ha explicado anteriormente, las funciones de integración de narrativa y las variables de control de funcionamiento.

En las figuras A.2 y A.3 podemos apreciar como están estructuradas las variables de las que hará uso el *plugin* para poder lograr la integración de la narrativa. En la primera de ellas vemos la organización de las capas del mapa. La parte de la izquierda representa el formato que tendrá en el desplegable de control de *Leaflet* y la parte de la derecha es el nombre identificador de la capa. En la segunda imagen podemos ver las variables que representan las secciones del texto. La primera indica el *div* en el que está situado el texto y las otras son los puntos de referencia. En este caso las referencias son unos pequeños marcadores que hay antes de cada sección, pero típicamente será el id del título de la sección.

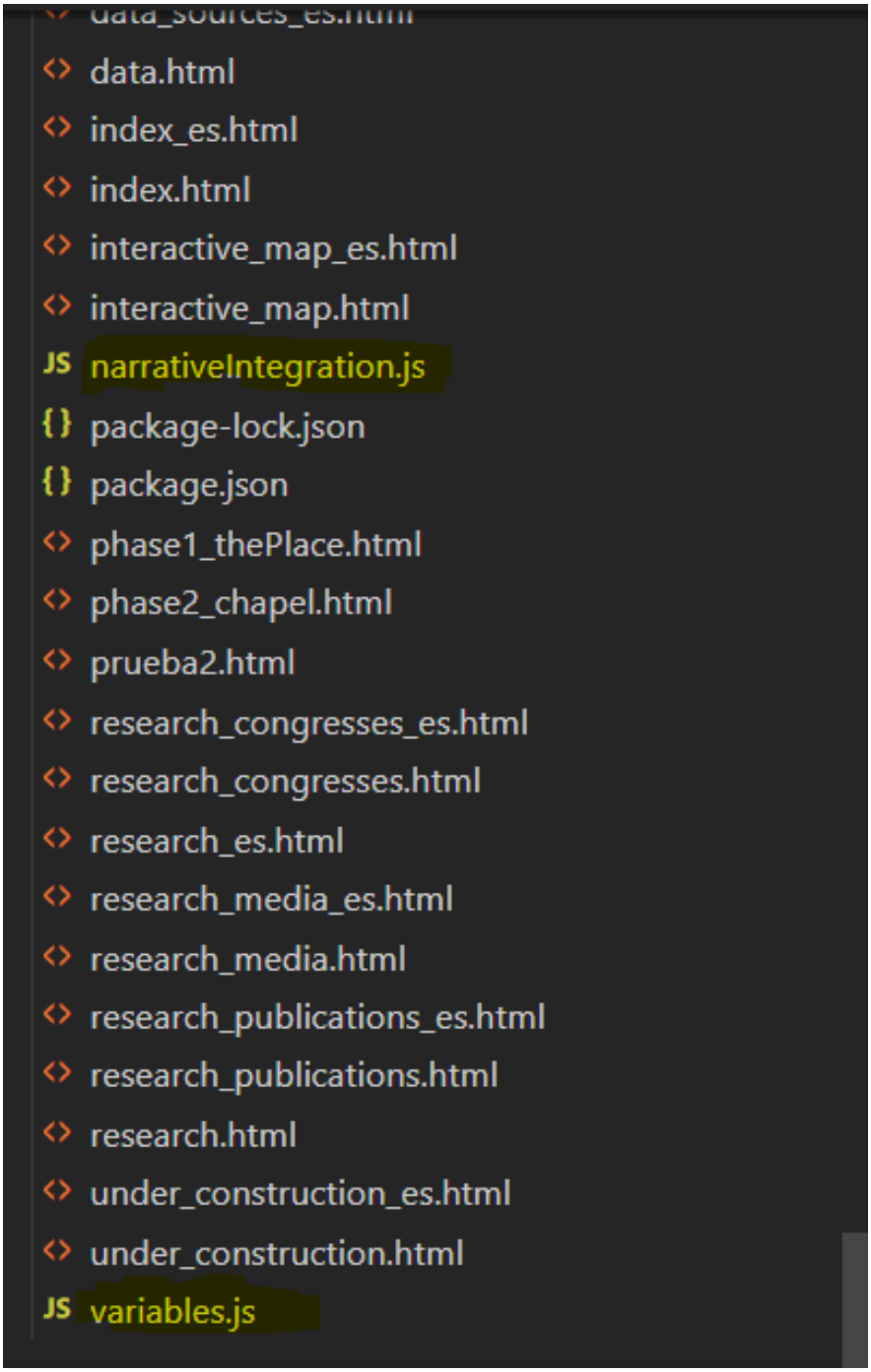

Figura A.1: Directorio web con los archivos del *plugin* añadidos (marcados en amarillo)

Una vez tenemos las variables procedemos a crear las listas de la figura A.4. Por un lado tenemos todas las variables de referencia de los textos y por el otro las capas. El orden de estas dos listas marca qué capa estará "enlazada" con qué sección del texto, siendo este, el que se encuentre en el mismo índice de la lista.

```
/**Creando la asociación de fases**/
var evolutionStages = {
"<strong>El lugar</strong>": thePlace,
"<strong>Antes del siglo XII</strong>": prerromanesqueSpaces,
"<strong>Del siglo XII al XV</strong>": romanesqueSpaces,
"<strong>Primera mitad del siglo XVI</strong>": firstHalf16thCentury,
"<strong>Segunda mitad del siglo XVI</strong>": secondHalf16thCentury,
"<strong>Primera mitad del siglo XVII</strong>": firstHalf17thCentury,
"<strong>Segunda mitad del siglo XVII</strong>": secondHalf17thCentury,
};
```

Figura A.2: Capas del mapa interactivo de *Leaflet* con sus nombres.

```
//funcionalidad para cambiar el texto en funcion del mapa selecionado
var text = "left_column",
    place = "marker01",
    preroman = "marker02",
    roman = "marker03",
    first16 = "marker04",
    second16 = "marker05",
    first17 = "marker06",
    second17 = "marker07";
```

Figura A.3: *Id* de los elementos del texto usados de referencia para las secciones.

```
texts = [place, preroman, roman, first16, second16, first17, second17];
layers = [thePlace, prerromanesqueSpaces, romanesqueSpaces, firstHalf16thCentury, secondHalf16thCentury, firstHalf17thCentury, secondHalf17thCentury];
```

Figura A.4: Listas que contienen los elementos de referencia del texto y las capas.

A continuación se muestra en la figura A.5 como se ha estructurado la información en formato *JSON* procedente, en este caso, de *QGIS*. Se ha creado dentro del directorio principal un subdirectorio con el nombre *data_json* ya que el *plugin* busca automáticamente sobre este directorio. En el ejemplo de *Digital Samos* los archivos están duplicados ya que la web cuenta con versión en inglés y castellano. Estos archivos debemos especificarlos en un *JSON* como el que se muestra en la figura A.6. En este ejemplo tenemos como lenguajes español (es) e inglés (en). El valor del idioma se obtiene de la variable *languaje* de *varibles.js* del *plugin*. Dentro del idioma está organizado por capas, y dentro de cada capa no hay un orden específico. Como se comentó anteriormente, el ordenar este archivo por capas fue pensado para poder hacer búsquedas más precisas y eficientes.

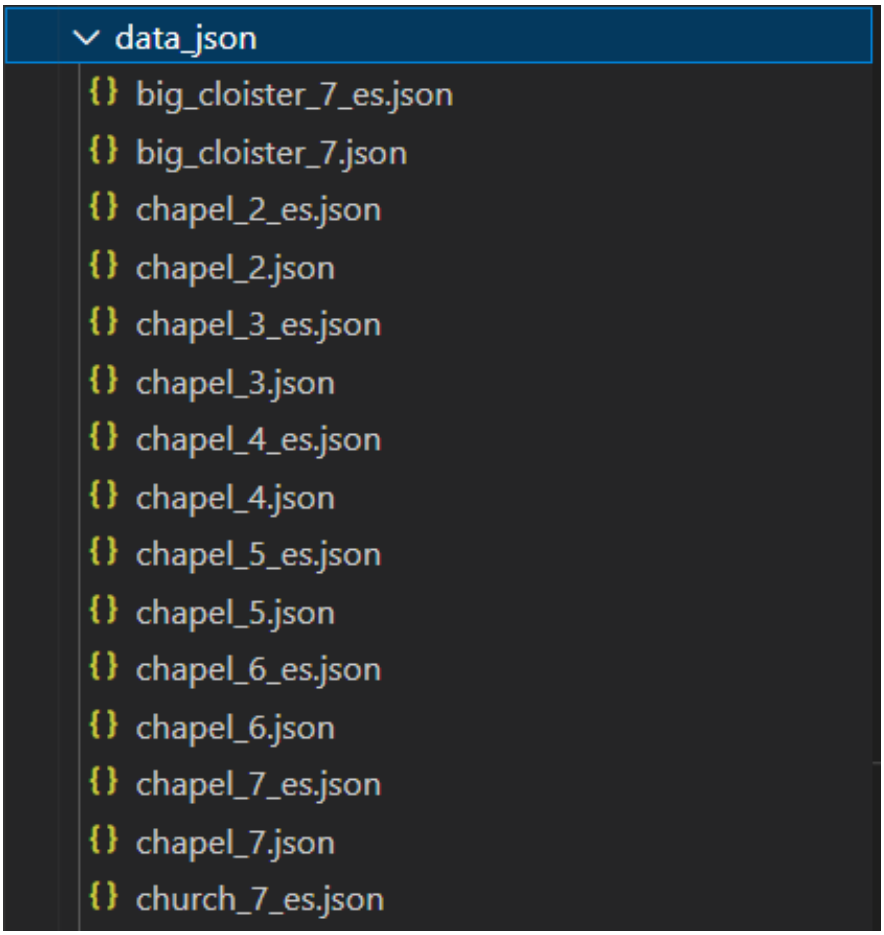

Figura A.5: Directorio que contiene los archivos *JSON* con la información de los elementos del mapa.

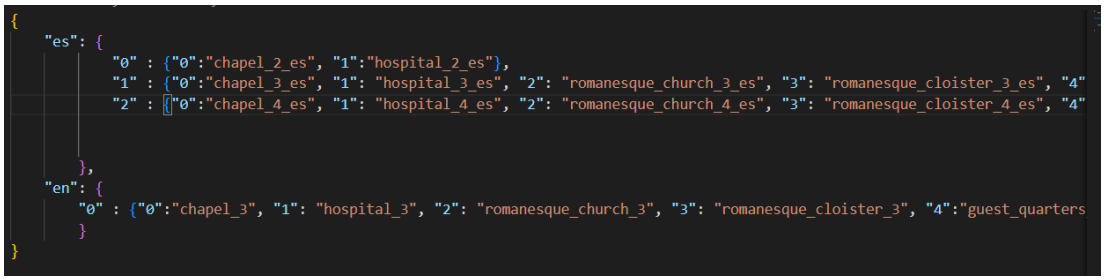

Figura A.6: Archivo de ejemplo *data.json* con la información de los nombres de JSON.

Por último, para poder hacer uso de las funciones debemos crear un objeto utilizando la factoría incluida en el *plugin* como se aprecia en la figura A.7. Una vez creado el objeto podremos llamar a cada una de las funciones de forma independiente. Recordar que las funciones tienen opciones que se pueden incluir o no como en el caso de la figura.

```
let demo = L.narrativeIntegration();
demo.readJSONData("left_column", "./data_json/data.json");
demo.changeTextWithLayer(texts);
demo.changeLayerWithText(texts, layers);
```

Figura A.7: Ejemplo de inicialización y llamada a las funciones.

Bibliografía

[1] "Mapping the catalogue of ships | the iliad, book 2," consultado el 09/09/2022. [En línea]. Disponible en: https://ships.lib.virginia.edu/neatline/show/iliad-book-2#records/269

[2] P. C. Iii, *Leaflet.js Essentials (Illustrated).* Packt Publishing., 2014.

[3] "What is neatline?" consultado el 09/09/2022. [En línea]. Disponible en: https://neatline.org/about/

[4] D. McClure and G. Worthey, "Colophon: Grapl, the graves "platform". The Chinese Deathscape: Grave reform in Modern China," consultado el 09/09/2022. [En línea]. Disponible en: https://chinesedeathscape.supdigital.org/read/colophon

[5] "Leaflet," consultado el 09/09/2022. [En línea]. Disponible en: https://leafletjs.com/index.html

[6] E. López Salas, "Digital samos: Interactive map," consultado el 09/09/2022. [En línea]. Disponible en: https://digitalsamos.udc.es/interactive_map.html

[7] "Omeka," consultado el 09/09/2022. [En línea]. Disponible en: https://omeka.org/

[8] "Descubre QGIS," consultado el 09/09/2022. [En línea]. Disponible en: https://www.qgis.org/es/site/about/index.html

[9] "Visualstudio," consultado el 09/09/2022. [En línea]. Disponible en: https://code.visualstudio.com/

[10] "Github: Where the world builds software," consultado el 09/09/2022. [En línea]. Disponible en: https://github.com/

[11] "Jira software," consultado el 09/09/2022. [En línea]. Disponible en: https://www.atlassian.com/es/software/jira

[12] "Html: Lenguaje de etiquetas de hipertexto," consultado el 09/09/2022. [En línea]. Disponible en: https://developer.mozilla.org/es/docs/Web/HTML

[13] I. Fajfar, *Start Programming Using HTML, CSS, and JavaScript (Chapman & Hall/CRC Textbooks in Computing)*. Chapman and Hall/CRC, 2015.

[14] "Javascript," consultado el 09/09/2022. [En línea]. Disponible en: https://developer.mozilla.org/es/docs/Web/JavaScript

[15] "CSS," consultado el 09/09/2022. [En línea]. Disponible en: https://developer.mozilla.org/es/docs/Web/CSS

[16] "Extending Leaflet, class theory - Leaflet - a javascript library for interactive maps," consultado el 09/09/2022. [En línea]. Disponible en: https://leafletjs.com/examples/extending/extending-1-classes.html

[17] J. Loeliger and M. McCullough, *Version control with Git*. O'Reilly, 2012.

[18] "Npm docs: Developers," consultado el 09/09/2022. [En línea]. Disponible en: https://docs.npmjs.com/cli/v8/using-npm/developers/

[19] P. Letelier, J. H. Canós, and C. Penadés, "Metodologías Ágiles en el desarrollo de software," *VIII Jornadas de Ingeniería del Software y Bases de Datos JISBD*, vol. 1, pp. 1–8, 2003.

[20] "Metodología Scrum: cómo aplicar el método Scrum," consultado el 09/09/2022. [En línea]. Disponible en: https://www.apd.es/metodologia-scrum-que-es/

[21] J. Subra, *Scrum Un Metodo Agil Para Sus Proyectos. (2.a ed.).*, 2020.

[22] "BOE-A-2019-14977," consultado el 09/09/2022. [En línea]. Disponible en: https://www.boe.es/eli/es/res/2019/10/07/(8)

[23] M. Madsen, O. Lhoták, and F. Tip, "Promises," *Proceedings of the ACM on Programming Languages*, vol. 1, pp. 1–24, 2017. [En línea]. Disponible en: https://dl.acm.org/doi/10.1145/3133910

[24] "Leaflet/plugin-guide.md at main · leaflet/leaflet. github," consultado el 09/09/2022. [En línea]. Disponible en: https://github.com/Leaflet/Leaflet/blob/main/PLUGIN-GUIDE.md

[25] "Leaderline," consultado el 09/09/2022. [En línea]. Disponible en: https://anseki.github.io/leader-line/

[26] J. Fuertes Agras, “Github - jaimefuertes/leaflet.narrativeintegration: Leaflet plugin for narrative integration. github,” consultado el 09/09/2022. [En línea]. Disponible en: https://github.com/jaimefuertes/Leaflet.NarrativeIntegration