Universidad de Valladolid

Escuela de Ingeniería Informática

Trabajo Fin de Grado

Grado en Ingeniería Informática

Mapache: Framework de soporte a sistemas de localización de personas y activos

Autor:
D. Sergio García Villanueva

Tutores:
D. Daniel Barba Gutiérrez
Dr. Diego R. Llanos Ferraris

Resumen

Este proyecto contiene el análisis, diseño e implementación de un framework para aplicaciones web del lado del cliente que puede comunicarse de manera asíncrona con el servidor utilizando la tecnología AJAX. El objetivo principal de este proyecto es la creación de un framework de apoyo al desarrollo de aplicaciones web basadas en el posicionamiento y la sensorización.

Este framework ofrece la posibilidad de ser utilizado por cualquier aplicación web independientemente del lenguaje mediante comunicaciones con objetos JSON. Además, se proporciona un sistema de creación automática de interfaz gráfica en el navegador acompañado de un fichero de estilos para simplificar la tarea de diseño por parte del programador que use este framework.

Abstract

This project contains the analysis, design, and implementation of a client-side web application framework, which transfers data with a backend server using the AJAX technology. The main point of this proyect is building a framework for web applications based on positioning and sensorization.

This framework can be used by any web application no matter the language used for the implementation by using JSON objects for the communication. Moreover, the framework offers a full frontend interface creation system with the proper stylesheet files in order to simplify the design process of the web developer using this framework.

Tabla de Contenidos

Lista de Figuras

Lista de Tablas

Capítulo 1

Introducción

El crecimiento de las aplicaciones web ha desencadenado la aparición de diferentes frameworks y librerías para facilitar el desarrollo de las mismas tanto en el lado del servidor, como en el del cliente. En el año 2006, el *World Wide Web Consortium* presentó la especificación de la interfaz *XMLHttpRequest* [16], creada para realizar peticiones *HTTP* hacia un servidor desde el lado del cliente.

Esta interfaz permite crear aplicaciones web en el lado del cliente utilizando la tecnología *AJAX* (Asynchronous JavaScript And XML) [6] para la creación de aplicaciones que se ejecutan en el navegador y mantienen una comunicación asíncrona con el servidor, permitiendo así el desarrollo de páginas web que no requieren recargar el navegador para obtener datos actualizados.

Con la aparición de *AJAX* surgieron frameworks y aplicaciones del lado del cliente que permiten la creación de mapas interactivos que obtienen la información geográfica del mapa de un servidor de forma asíncrona.

Además, en la actualidad existen tecnologías para la obtención de posicionamiento de personas y activos, y necesitan de una herramienta que les permita reflejar esta información en una aplicación web en tiempo real. Como respuesta a esta necesidad surge *Mapache*, un framework para aplicaciones web del lado del cliente de localización de personas y activos en mapas interiores y exteriores, diseñado con los principios de interactividad, velocidad, y usabilidad de las aplicaciones desarrolladas con la tecnología *AJAX*.

1.1. Objetivos

Este proyecto tiene como objetivo dar respuesta a las necesidades mencionadas anteriormente. Se desarrollará un framework que, mediante el uso de *AJAX*, se permita mostrar la localización de personas y activos en tiempo real sobre un mapa interactivo.

Además, el otro objetivo de *Mapache*, será el de mostrar información de sensorización actualizada sobre estos activos. De este modo, podrá ser utilizado por aplicaciones web que tengan distintos objetivos, pero que tengan en común la necesidad de posicionar activos sobre un mapa.

1.2. Estructura de la Memoria

Esta memoria contiene los siguientes capítulos:

Estado del Arte Estudio de las diferentes soluciones existentes al problema que se quiere resolver en este proyecto, y valorar si alguna de ellas va a ser utilizada en este.

Tecnologías para el Desarrollo de Frameworks Web Análisis de las herramientas y lenguajes para el desarrollo de frameworks en el lado del cliente de una aplicación web.

Análisis Especificación de los objetivos, requisitos, y casos de uso de *Mapache*.

Plan de Proyecto Plan de Desarrollo del proyecto. Incluye el plan de proceso, el ciclo de vida, la gestion del proceso, su planificación temporal, y el plan de gestión de riesgos.

Diseño Definición de la arquitectura, modelo de dominio, y los diagramas de secuencia del framework. Además, se desarrolla el modelo de comunicación con el servidor de la aplicación web.

Implementación y Pruebas Presentación de la solución obtenida tras los capítulos de análisis y diseño, con el modelo de implementación, y las pruebas realizadas para comprobar el correcto funcionamiento del framework.

Casos de Uso Reales *Mapache* está siendo utilizado actualmente por aplicaciones web reales, por lo que se hará una descripción de las mismas y cómo el framework ayuda al desarrollo de la aplicación.

Conclusiones Resumen del trabajo realizado y los objetivos cumplidos, y presentación de las líneas de trabajo futuro que surgen de este proyecto.

Capítulo 2

Estado del Arte

En este capítulo se analizarán las tecnologías actuales para la representación de mapas en aplicaciones web en el lado del cliente, y si estas proporcionan funcionalidad extra para localizar elementos en el mapa y mostrar información sobre ellos.

2.1. Servidores de Mapas

Primero se hará un estudio de las diferentes bases de datos que contienen información geográfica acerca del mapa del mundo, concretamente se estudiarán los servidores de mapas *Google Maps* y *OpenStreetMap*.

2.1.1. Google Maps

Google Maps [2] es un servidor de aplicaciones de mapas. Utiliza la información geográfica de sus servidores para la creación de aplicaciones que muestran mapas interactivos. Esta información geográfica no es accesible directamente por el desarrollador, son necesarias APIs que Google facilita para poder renderizar el mapa del mundo.

Las necesidades de *Mapache* hace que también sea necesario acceder a datos sobre mapas interiores, Google ofrece el servicio Google MapsIndoor CMS en el que se puede modificar el diseño de mapas interiores para tu aplicación, y ofrece servicios de localización indoor basada en GPS y WiFi. Toda esta información sobre la aplicación a desarrollar tiene que alojarse en los servidores de Google.

Ventajas

- La cobertura de sus mapas es total a nivel internacional en los aspectos de imágenes y geocodificación.
- Permite al usuario modificar el diseño del mapa de manera intuitiva, tanto en mapas exteriores como interiores.
- La API es fácil de utilizar y ofrece funcionalidades extra como clustering de nodos que pueden ser de utilidad en el desarrollo del proyecto.

- Ofrece sistemas de búsqueda de rutas más cercanas entre dos puntos.
- Ofrece sistemas de localización mediante GPS y WiFi.

Desventajas

- La información geográfica de los mapas a nivel de base de datos no es accesible.
- La API JavaScript no se puede modificar.
- Los marcadores pueden ser personalizables a nivel de icono, pero no se pueden añadir elementos decorativos dinámicamente para mostrar información en tiempo real sobre estos.
- Los mapas de interiores están completamente integrados con el mapa del mundo exterior, *Mapache* tiene que ofrecer esta funcionalidad, pero no siempre, hay aplicaciones que solo quieran mostrar mapas interiores.
- Toda la información sobre los mapas, marcadores, y localización indoor residen en el servidor de Google.
- El uso de estos servicios para fines empresariales conlleva un método de suscripción de pago para hacer uso de los mismos.

2.1.2. OpenStreetMap

OpenStreetMap [8] es un proyecto colaborativo de código abierto que ofrece información geográfica a nivel de base de datos sobre el mapa del mundo conocida como información vectorial. OpenStreetMap anima a los desarrolladores a alojar su propio servidor de mapas para usos comerciales, es por ello que ofrecen toda la información geográfica que poseen de forma gratuita.

OpenStreetMap no ofrece una API para el renderizado de mapas en aplicaciones. Al ser de código libre, existe una comunidad de desarrolladores dedicada a crear librerías y frameworks para esta tarea, es decir, OpenStreetMap sólo ofrece información geográfica. Esta información geográfica es almacenada en bases de datos PostGIS, un sistema de administración de bases de datos PostgreSQL que permite almacenar información sobre mapas y localización.

Para almacenar información sobre mapas de interiores OSM está desarrollando actualmente OpenStreetMap Indoor Mapping en el que añade a sus sistemas de información nuevos tags para representar mapas interiores.

Ventajas

- La cobertura de sus mapas es muy amplia aunque al ser suministrada por personas independientes como proyecto de software libre no está a la altura de Google Maps.
- La posibilidad de alojar un servidor de mapas privado es un punto fuerte de este sistema.

- Al ser un proyecto software libre, existe una comunidad muy grande detrás de este, ofreciendo muchas librerías distintas para elegir la que más se adapte a tu proyecto.

- Permite la posibilidad de sólo poblar la base de datos con la información geográfica del área que necesitas, reduciendo así el espacio necesario si tu objetivo no es el mapa del mundo entero.

Desventajas

- Necesidad de una máquina que actue de servidor.

- Tener que montar tu propio servidor es más complicado que te lo den hecho.

- No tiene una API o aplicación sobre la que trabajar directamente.

- Es necesario actualizar frecuentemente el servidor.

2.2. Librerías de creación de mapas interactivos

Aunque Google Maps viene con su propia librería JavaScript para la creación de mapas interactivos, es necesario hacer un estudio de las mismas opciones que ofrece OpenStreetMap. Para ello hay que buscar librerías que permitan el posicionamiento de elementos en el mapa. Estas son algunas de las librerías encontradas, con una valoración de cómo de útiles pueden llegar a ser para *Mapache*.

2.2.1. Cartagen

Cartagen es un framework de renderizado de mapas interactivos utilizando el elemento Canvas de HTML5. El estilo del mapa se crea con GSS (Geographic Style Sheets), parecido al CSS para mostrar información geoespacial. Este framework es bastante útil para mostrar datos a tiempo real sobre un mapa pero sin nigún estilo, suele utilizarse para cuando la interacción con el usuario no es una prioridad.

2.2.2. Kartograph

Es un framework ligero para crear mapas interactivos sin ningún tipo de servicio adicional más que mostrar el mapa.

2.2.3. Kothic JS

Usando el Canvas de HTML5, renderiza secciones del mapa que se le solicita, sin servicios adicionales. No es un mapa construido por tiles, por lo que no se puede navegar ni hacer zoom sobre el mismo.

2.2.4. Mapbox GL JS

Mapbox es una una suite de librerías para la creación de mapas de tiles, y la creación de herramientas del lado del servidor de renderizado de mapas. Utilizando su propia guía de estilos, la información de OSM, y OpenGL, Mapbox crea de manera dinámica los tiles del mapa y los sirve según los va necesitando el cliente.

Mapbox GL JS [13] es la librería JavaScript para el renderizado de estos mapas en una aplicación web. Tiene soporte para navegadores de dispositivos fijos y móviles. Aporta funcionalidades extra como la creación de marcadores o popups para los eventos de ratón.

Aunque OSM permite que el desarrollador aloje su propio servidor de mapas, Mapbox usa el suyo propio, con la condición de tener que suscribirte a sus planes para empresas. El problema de estos planes es que aumenta el precio según aumenta el número de renderizaciones del mapa.

2.2.5. OpenLayers

OpenLayers [15] es otra librería para crear mapas web interactivos. Es un proyecto con muchos años de desarrollo y provee muchas funcionalidades extra para personalizar los mapas.

Una de sus características más importantes es que permite renderizar mapas utilizando diferentes formatos de capas, puede usar tiles generadas por OSM, y también permite renderizar mapas utilizando datos de GeoJSON.

Posee las funcionalidades requeridas por *Mapache* como poder renderizar varias capas de un mapa, marcadores personalizables, botones para controlar el mapa personalizables.

El problema que tiene esta librería es que no tiene compatibilidad con dispositivos móviles, por lo que para integrarlo en una aplicación con diseño responsive es necesario capturar los eventos que cambian el estado del mapa y actuar en consecuencia.

2.2.6. Leaflet

La librería utilizada por el propio sitio de web de OSM. Leaflet es la librería moderna para el renderizado de mapas interactivos. Tiene como objetivo el renderizado de estos mapas con eficiencia, mediante una API [11] simple. Es una herramienta ligera y con soporte para todos los navegadores actuales, tanto móviles como fijos.

Utiliza la información de tiles generados con OSM para crear el mapa, estos tiles son solicitados al servidor de mapas de manera dinámica, permitiendo así mover el mapa, hacer zoom sobre éste de manera rápida y eficiente. Posee funciones para la creación de marcadores propios y personalizados por el desarrollador.

Una característica que hace a Leaflet muy versátil es el concepto de capas que utiliza, todo en Leaflet es una capa, y todas las capas pueden agruparse a su vez en nuevas capas. Las capas presentan una interfaz que permite realizar operaciones básicas como el renderizado sobre estas.

Al ser de código abierto, tiene guías realizadas por ellos mismos para ampliar o modificar la funcionalidad de Leaflet sin tener que modificar el código fuente (que también se puede).

El desarrollador de la librería es desarrollador también en Mapbox, por lo que utiliza elementos de este como la guía de estilos. Esto hace posible descargar los estilos de Mapbox y utilizarlos en Leaflet.

Es una librería moderna y utiliza el nuevo estándar ECMAScript 6, además tiene una documentación muy completa y bien definida por lo que aprender a utilizar Leaflet es muy sencillo si estás familiarizado con este nuevo estándar.

Capítulo 3

Tecnologías para el Desarrollo de Frameworks Web

En este capítulo se detalla el Estado del Arte de las diferentes herramientas y tecnologías que utiliza *Mapache*.

Primero se explicará qué es JavaScript y los diferentes estándares que se usan actualmente. A continuación se expondrán los diferentes frameworks para el desarrollo de aplicaciones web JavaScript y las arquitecturas que están detrás de los mismos. Posteriormente se analizarán los diferentes gestores de paquetes y librerías de JavaScript, y se expondrán los benificios y problemas de cada uno. Al ser un framework front-end que va a necesitar renderizar elementos en un navegador web, también se hará un estudio de las diferentes herramientas que se usan para el preprocesado de los archivos CSS.

3.1. JavaScript

JavaScript es el lenguaje de programación más utilizado en el desarrollo del lado del cliente de una aplicación web [12]. Es un lenguaje basado en prototipos, un estilo de programación orientado a objetos en el cual, el objeto no se define mediante la instanciación de una clase, sino mediante la escritura de código, o mediante la clonación de objetos previamente creados.

En la actualidad, JavaScript está tomando fuerza en el desarrollo de aplicaciones web en el lado del servidor, el ejemplo más conocido es *Node.js*. La ventaja de poder usar el mismo lenguaje tanto en el lado del servidor como en el del cliente, es que muchas veces se usan las mismas funciones en ambos lados, ahorrando gran cantidad de líneas de código y tiempo.

3.1.1. Estándar actual

Hasta 2009, sólo existía ECMAScript [9] como estándar de JavaScript. De este estándar surgieron lenguajes de programación que son superconjuntos de ECMAScript, es decir, a través de un compilador se traducen a lenguaje JavaScript original.

TypeScript

Desarrollado por Microsoft, TypeScript es un superconjunto de JavaScript que añade tipado estático y objetos basados en clases en vez de prototipos. En su día introdujo un concepto importantísimo para poder ver JavaScript al nivel que se encuentra actualmente, los módulos.

Los módulos permiten que no tengas que añadir un enlace en tu página HTML por cada script JS que tuvieses, y añade una conexión entre ellos para poder acceder a un módulo desde otro cualquiera

CoffeeScript

Inspirado en Ruby, CoffeeScript es un lenguaje de programación compilado sobre JavaScript. Introduce una sintaxis muy parecida a Ruby, eliminación del carácter ";" para finalizar una expresión, eliminación de paréntesis en los argumentos de las funciones y muchas más.

Una funcionalidad muy interesante que añadió CoffeScript es la gestión del ámbito de las variables, que en su momento no estaba muy bien gestionada por ECMAScript.

Es el lenguaje por defecto en las aplicaciones de Ruby on Rails cuando creas el proyecto, esto se debe a que una gran parte de los desarrolladores de Ruby prefieren este lenguaje por la similitud en la sintaxis.

ECMAScript

JavaScript original, define un lenguaje de tipos dinámicos ligeramente inspirado en Java (aunque el nombre no tiene nada que ver).

A continuación se muestran los cambios más importantes del estándar con las diferentes versiones.

ECMAScript 5 Se añade un modo para la corrección de posibles errores en la implementación: el modo estricto.

ECMAScript 6 El estándar actual. Fue una reestructuración de la sintaxis. Aceptó todas las nuevas funcionalidades que otros lenguajes iban añadiendo, como las clases y módulos de TypeScript o los iteradores de CoffeeScript.

ECMAScript 7 En desarrollo. Tiene como objetivo continuar la reforma empezada en la versión anterior. Añade las "promesas" para mejorar la concurrencia y así dejar a un lado la necesidad de utilizar la tecnología AJAX a través de jQuery obligatoriamente en todos los proyectos JavaScript.

A día de hoy no hay ninguna razón funcional por la que pueda suponer una ventaja utilizar otros lenguajes que no sean ECMAScript 6, aunque también puede ser determinante la sintaxis como ya se expuso en CoffeeScript para los desarrolladores Ruby.

3.1.2. Librerías

En el año 2006, con la especificación de la interfaz *XMLHttpRequest*, surgió la tecnología *AJAX* que permitía realizar peticiones HTTP hacía un servidor desde el lado del cliente sin la necesidad de recargar la página web, la librería más conocida sobre esta tecnología es *jQuery*.

Hasta la creación de AJAX, el uso principal de JavaScript era la validación de formularios en el lado del cliente, y modificar el DOM HTML [4]. Con la aparición de AJAX se pasó a un nuevo modelo de programación basado en servicios web y aplicaciones del lado del cliente puramente JavaScript. Al no existir ninguna arquitectura estándar para la creación de aplicaciones y frameworks front-end, surgieron muchas librerías y frameworks para intentar dar solución a este problema.

A continuación se exponen las dos arquitecturas más utilizadas a día de hoy para el desarrollo de aplicaciones front-end.

Angular

Desarrollado por Google, es un framework que no consiguió mucha aceptación entre la comunidad de desarrolladores en su primera versión. Es por ello que es muy común hablar de *Angular2* [7], que fue la segunda versión que Google decidió lanzar para resolver los problemas que presentaba, aunque sigue siendo Angular totalmente reformado.

Se trata de un framework que utiliza una arquitectura basada en "Módulos" y "Componentes".

Un *Componente* es un trozo de funcionalidad de la aplicación, normalmente asociado a un fragmento de vista del navegador. Contiene la lógica del dominio de ese fragmento de la aplicación, una plantilla HTML (*Template*) para renderizar el componente en el DOM, y metadatos que que hacen de controlador entre el Componente y el Template.

Por encima de los componentes se encuentran los *Servicios*, que sirven para compartir funcionalidad entre componentes.

Una aplicación Angular está construida con *Módulos*, que sirven para organizar la aplicación en bloques de funcionalidad. Existe un módulo raíz, representado con la notación *@NgModule*. Este módulo raíz contiene a su vez otros módulos y componentes para crear la aplicación.

La arquitectura de Angular se representa en la Figura 3.1.

Flux

Desarrollado por Facebook, es una arquitectura para la creación de aplicaciones front-end. Está pensada para ser utilizada junto a *React*, un framework creado por ellos mismos, aunque no es necesario.

Una aplicación Flux [3] está compuesta por cuatro componentes: el manejador de eventos, la store, las acciones, y las vistas (generadas con JavaScript, o componentes React).

La *Store* contiene el estado y la lógica de la aplicación. Se puede decir que es el modelo de un patrón MVC convencional, aunque es posible tener más de una Store

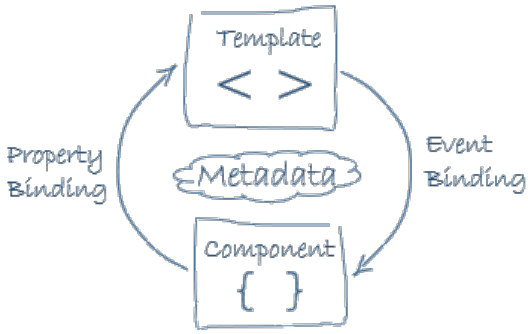


Figura 3.1: Arquitectura Angular

que manejan el estado de diferentes partes de la aplicación.

Las *Vistas* son fragmentos del HTML DOM que actualizan su estado cuando reciben una acción por parte del usuario o de la Store. Usando *React* estas vistas llevan asociado un controlador (no confundir con el controlador de un MVC) que se encarga de propagar el estado de la Store a sus hijos.

Una *Acción* es un objeto lanzado por el Manejador de Eventos con la intención de modificar el estado de la Store.

El *Manejador de Eventos* es único en las aplicaciones Flux, y es el encargado de distribuir la Acciones a las diferentes Stores.

La arquitectura de Flux se representa en la Figura 3.2.

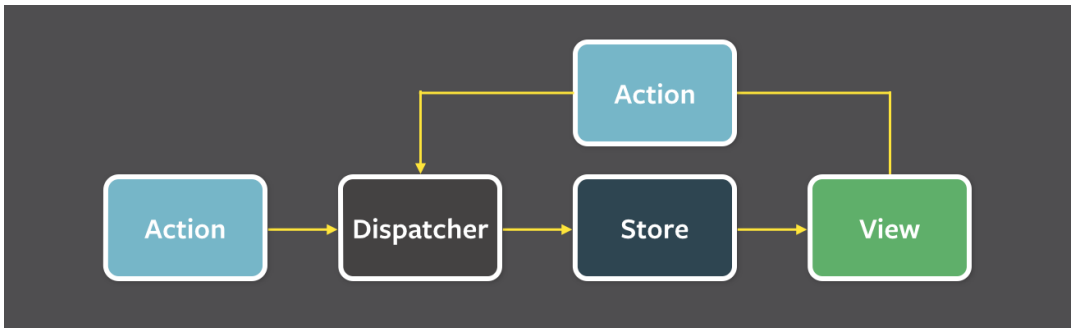


Figura 3.2: Arquitectura Flux

3.1.3. Module Bundlers

En la nueva especificación del estándar de ECMAScript que se está definiendo, se va a permitir crear módulos JS para poder ser reutilizados por otros scripts. Hasta entonces, es necesario utilizar librerías que permitan el uso de módulos, estas librerías son *Webpack* y *Browserify*. La primera está enfocada a la creación de aplicaciones front-end, con módulos para crear un servidor dedicado a esa aplicación [17]. La segunda opción es más ligera y sirve principalmente para importar scripts y librerías JS en el proyecto, lo que la hace perfecta para el desarrollo de frameworks.

3.2. Preprocesadores CSS

Un preprocesador CSS es un lenguaje escrito sobre CSS que compila y genera código CSS. A día de hoy, manejar un sólo fichero de CSS para todo un proyecto es un signo de malas prácticas. Se necesita dividir el código CSS en elementos más pequeños y reutilizables bajo demanda en otras secciones sin repetir porciones de código.

Es por ello que surgieron estos preprocesadores con nueva sintaxis más intuitiva y con funcionalidad extra para obtener mejores resultados. Hay una gran cantidad de preprocesadores, pero el más completo y utilizado es SASS. A continuación se expondrán las ventajas de usar SASS sobre CSS convencional.

SASS

Syntactically Awesome Stylesheets es el preprocesador más utilizado y conocido. Está programado en Ruby, por lo que se necesita de este para utilizarlo. Se define a sí mismo como "CSS con superpoderes", debido a que aporta muchas mejoras respecto al CSS convencional.

Variables Permite almacenar en variables estilos que se van a utilizar a menudo.

Anidamiento HTML tiene una jerarquía que se ve a simple vista, CSS no. SASS permite establecer jerarquías para definir estilos en elementos específicos.

Parciales SASS permite la creación de pequeñas hojas de estilos que van a ser reutilizadas por otras, evitando así la duplicación de código. Estos estilos se incluyen fácilmente mediante la directiva *import*.

Mixins Muchos estilos de CSS3 no están implementados en las versiones antiguas de navegadores y se necesita declarar un estilo por cada versión, estos son conocidos como vendor prefixes. Para no tener que reescribir una y otra vez estos prefijos, SASS da la opción de crear un estilo propio que gestione los prefijos y sólo lo tengas que escribir una vez y reutilizar donde quieras.

Herencia Parecida a la herencia de objetos, SASS permite heredar todos los estilos de un selector con el objetivo de reutilizar código.

Módulos CSS

Aunque ahora mismo SASS es el camino a seguir, parece ser que en un futuro no muy lejano esta tendencia va a cambiar a favor de los módulos CSS. Se trata de ficheros CSS en el que los selectores de clases tienen ámbito local por defecto.

La diferencia es que no se aplican directamente al documento HTML, sino que se utilizan en JavaScript, se importan usando librerías como Browserify, y se añade el estilo al elemento cuando lo creas. Como se ha descrito en el análisis de las librerías JavaScript, las nuevas arquitecturas hacen el renderizado de elementos HTML desde el lado del cliente y no desde el servidor, por lo que esta aproximación es la ideal para estas nuevas arquitecturas.

La gran ventaja es que se tiene control absoluto sobre el estilo que va a tener ese elemento concreto, y no hay que estar pendiente de si va a ser sobreescrito por un estilo definido en algún otro estilo del documento.

Capítulo 4

Análisis

Este capítulo detalla el Análisis de Requisitos de *Mapache*. Se trata del análisis de objetivos, requisitos y casos de uso del framework, según el estudio realizado en la fase previa, y recogido en el capítulo de Estado del Arte.

4.1. Requisitos

Los objetivos del sistema son requisitos de alto nivel que debe de cumplir el sistema.

4.1.1. Objetivos

Los objetivos con requisitos funcionales de usuario que se detallan a continuación.

OBJ-01: Localización de personas y activos sobre mapas exteriores e interiores en navegadores web

Mapache deberá mostrar sobre mapas de exteriores e interiores la localización en tiempo real de personas y activos.

OBJ-02: Mostrar información sobre personas y activos en navegadores web

Mapache deberá mostrar sobre un navegador web información relativa a personas y activos en tiempo real.

4.1.2. Requisitos Funcionales

Los requisitos funcionales describen el comportamiento del sistema.

RF-01: Renderizar mapa exterior

El sistema deberá permitir el renderizado del mapa del mundo exterior. Este renderizado se efectuará según RNF-02, y RNF-03.

RF-02: Renderizar mapa interior

El sistema deberá permitir el renderizado de mapas interiores. Este renderizado se efectuará según RNF-02, y RNF-03.

RF-03: Navegación entre plantas de mapas interiores

El sistema deberá permitir la navegación entre las distintas plantas de un mapa interior. Esta navegación se efectuará según RNF-04.

RF-03: Posicionamiento de nodos

El sistema deberá permitir el posicionamiento de nodos. Este posicionamiento se efectuará según RNF-01, RNF-05, RNF-06, y RNF-07.

RF-04: Información sobre nodos

El sistema deberá permitir al usuario consultar información sobre los nodos. Esta información se muestra según RNF-01, RNF-08, y RNF-09.

RF-05: Estado de los nodos

El sistema deberá permitir mostrar el estado de un nodo. Este estado se muestra según RNF-10

RF-05: Agrupación de nodos

El sistema deberá permitir la agrupación de nodos en clusters. Esta agrupación se efectuará según RNF-11.

RF-06: Histórico de información

El sistema deberá permitir al usuario consultar información histórica sobre nodos. Esta información se muestra según RNF-12.

RF-06: Visibilidad de nodos

El sistema deberá permitir al usuario cambiar la visibilidad de los nodos.

RF-07: Iconos personalizados

El sistema deberá permitir mostrar iconos personalizados para los nodos.

RF-08: Modificar el documento HTML

El sistema deberá permitir modificar el documento HTML para añadir o borrar elementos HTML, y cambiar el estilo de dichos elementos. Esta modificación se efectuará según RNF-13.

4.1.3. Requisitos No Funcionales

Los requisitos no funcionales definen las características de cómo debe funcionar el sistema.

RNF-01: Obtención de información del sistema

La información necesaria del servidor recogida en RI-01, RI-02, RI-03, RI-04, RI-05, RI-06, y RI-07 se recopila mediante llamadas asíncronas a la API de la aplicación cada segundo.

RNF-02: Uso de Leaflet para el renderizado de mapas

El renderizado de los mapas se efectuará utilizando la librería de JavaScript *Leaflet*.

RNF-03: Formato de los tiles de mapas

Los tiles que se usarán para el renderizado de mapas estarán en formato PNG.

RNF-04: Navegación entre plantas

La navegación entre plantas de un mapa interior deberá realizarse mediante seleccionables mostrados en un desplegable en el mapa.

RNF-05: Posicionamiento de nodos en tiempo real

El posicionamiento de los nodos deberá realizarse en tiempo real.

RNF-06: Posicionamiento de nodos sobre mapas

El posicionamiento de los nodos deberá realizarse sobre un mapa interactivo.

RNF-07: Sistemas de referencia para el posicionamiento de nodos

El posicionamiento de los nodos podrá realizarse usando el sistema de coordenadas geográficas, y el sistema de referencias relativas.

RNF-08: Visualización de información en menús

La información obtenida sobre los nodos se mostrará en un sistema de menús.

RNF-09: Visualización de información en tooltips

La información obtenida sobre los nodos se mostrará en un tooltip colocado en la parte superior del marcador del nodo en el mapa.

RNF-10: Visualización estado de nodos

El estado de los nodos se mostrará junto al marcador del nodo en el mapa con un sistema de colores.

RNF-11: Visualización de cluster de nodos

A partir de un nivel de zoom proporcionado por la API, se mostrarán los nodos que pertenecen al cluster en el mapa, en caso contrario se mostrará un marcador representando al conjunto.

RNF-12: Uso de gráficas para mostrar el histórico

La información sobre el histórico de los nodos se mostrará mediante el uso de gráficas interactivas.

RNF-13: Uso de DOM para modificar el documento HTML

Para la modificación del documento HTML se utilizará el estándar de W3C DOM.

RNF-14: Formato de los datos del servidor

La API del servidor deberá devolver los datos requeridos en formato JSON.

RF-15: Compatibilidad de navegadores

El sistema deberá funcionar correctamente en el 95 % de los navegadores del mercado, tanto los de dispositivos fijos, como móviles.

4.1.4. Requisitos de Información

Los requisitos de información recopilan los datos con los que se trabaja, en el caso de *Mapache*, representa los datos que va a utilizar de la API de la aplicación a la que se conecta.

RI-01: Información sobre posición

- Coordenada X
- Coordenada Y
- Coordenada Z (corresponde a la planta del mapa)

RI-02: Información sobre mapa

- CRS (Current Reference System)
- Tamaño del mapa
- Zoom mínimo soportado por el mapa
- Zoom máximo soportado por el mapa

RI-03: Información sobre plantas del mapa

- Nombre
- Nivel
- URL tiles de la planta del mapa

RI-04: Información sobre nodos

- Nombre del dato
- Valor
- Unidades

RI-05: Información sobre cluster

- Nombre
- Nodos hijo
- Nivel mínimo para expandir nodos

RI-06: Información sobre histórico del nodo

- Nombre del dato
- Valor
- Unidades
- Tiempos de las medidas

RI-07: Información sobre estado del nodo

- Estado
- Valor
- Color

4.2. Casos de Uso

A continuación se detalla el análisis de casos de uso de *Mapache*. Los casos de uso nos ayudarán a describir cómo se usa el sistema, y cómo los usuarios interactúan con el mismo.

4.2.1. Actores

Al tratarse de un framework front-end, tenemos el actor Usuario, que es el que interactúa directamente con el sistema. Por otro lado se encuentran los actores secundarios Servidor de Aplicación y Servidor de Mapas que son los encargados de suministrar la información de los nodos y de los mapas respectivamente.

ACT-01:	**Usuario**
Autores:	Sergio García Villanueva
Fuentes:	Sergio García Villanueva
Este actor representa al usuario final que va a utilizar los elementos de interacción proporcionados por *Mapache*.	

ACT-02:	**Servidor de Aplicación**
Autores:	Sergio García Villanueva
Fuentes:	Sergio García Villanueva
Este actor representa al servidor que va a proporcionar a *Mapache* la información relativa a los nodos.	

ACT-03:	**Servidor de Mapas**
Autores:	Sergio García Villanueva
Fuentes:	Sergio García Villanueva
Este actor representa al servidor que se encarga de proveer a *Mapache* las imagenes del mapa que se va a renderizar.	

4.2.2. Diagrama de Casos de Uso

El diagrama general de casos de uso se muestra en la figura 4.1.

4.2.3. Descripción de Casos de Uso

A continuación se detalla la descripción de los Casos de Uso de *Mapache*.

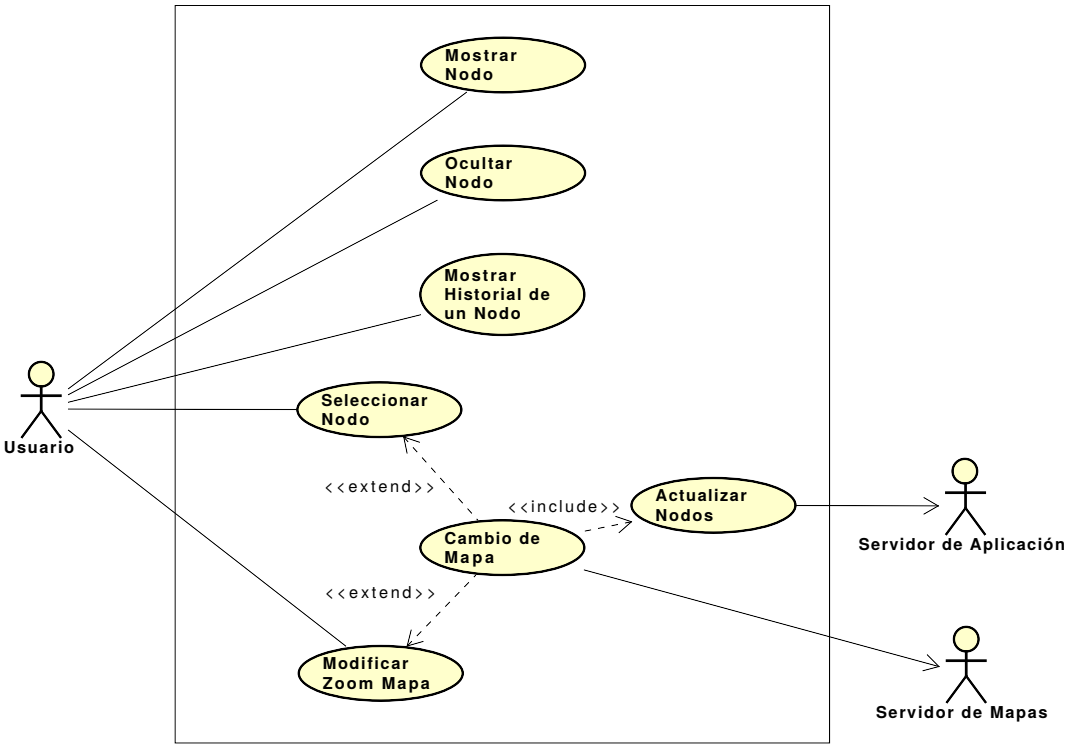


Figura 4.1: Diagrama general de Casos de Uso

UC-01:	**Mostrar Nodo**
Autores:	Sergio García Villanueva
Fuentes:	Sergio García Villanueva
Descripcion:	El sistema deberá comportarse tal y como se describe en el siguiente caso de uso cuando el usuario seleccione mostrar un nodo.
Precondición:	El estado del nodo debe de ser no visible.
	Secuencia normal
1.	El actor *Usuario* selecciona mostrar un nodo.
2.	El *Sistema* muestra el nodo en el mapa.
3.	El *Sistema* muestra la información sobre la visibilidad del nodo.
4.	El *Sistema* marca el nodo como visible.
	Flujo alternativo
2a.	Si se detecta que el nodo es un cluster, el *Sistema* muestra en el mapa todos los nodos que pertenecen al cluster.
Postcondición:	El estado del nodo es visible.

UC-02:	**Ocultar Nodo**
Autores:	Sergio García Villanueva
Fuentes:	Sergio García Villanueva
Descripcion:	El sistema deberá comportarse tal y como se describe en el siguiente caso de uso cuando el usuario seleccione ocultar un nodo.
Precondición:	El estado del nodo debe de ser visible.
	Secuencia normal
1.	El actor *Usuario* selecciona ocultar un nodo.
2.	El *Sistema* oculta el nodo en el mapa.
3.	El *Sistema* muestra la información sobre la invisibilidad del nodo.
4.	El *Sistema* marca el nodo como no visible.
	Flujo alternativo
2a.	Si se detecta que el nodo es un cluster, el *Sistema* oculta en el mapa todos los nodos que pertenecen al cluster.
Postcondición:	El estado del nodo es no visible.

UC-03:	**Mostrar Historial de un Nodo**
Autores:	Sergio García Villanueva
Fuentes:	Sergio García Villanueva
Descripcion:	El sistema deberá comportarse tal y como se describe en el siguiente caso de uso cuando el usuario solicite mostrar el historial de datos de un nodo.
	Secuencia normal
1.	El actor *Usuario* selecciona mostrar el historial de un dato concreto de un nodo.
2.	El *Sistema* comprueba que existe más de un dato de ese tipo para ese nodo.
3.	El *Sistema* crea la gráfica correspondiente a esos datos y renderiza la misma.
	Excepciones
3a.	Si no hay suficientes datos para crear una gráfica, el *Sistema* emite un mensaje de error y el caso de uso queda sin efecto.
Postcondición:	Los datos seleccionados se muestran en el navegador.

UC-04:	**Renderizar Mapa**
Autores:	Sergio García Villanueva
Fuentes:	Sergio García Villanueva
Descripcion:	El sistema deberá comportarse tal y como se describe en el siguiente caso de uso cuando se solicite cambiar de mapa.
	Secuencia normal
1.	El *Sistema* comprueba que el mapa renderizado no es el actual.
2.	El *Sistema* elimina el mapa actual.
3.	El *Sistema* solicita al *Servidor de Mapas* las imágenes del nuevo mapa.
4.	El actor *Servidor de Mapas* devuelve al *Sistema* las imágenes solicitadas.
5.	El *Sistema* guarda la información sobre el nuevo mapa.
6.	El *Sistema* renderiza el mapa.
7.	Se realiza el caso de uso Actualizar Nodos.
	Excepciones
1a.	Si el mapa seleccionado es el mismo que está renderizado el caso de uso queda sin efecto.
5a.	Si no hay comunicación con el *Servidor de Mapas* el *Sistema* emite un mensaje de error y el caso de uso queda sin efecto.
Postcondición:	El nuevo mapa esta seleccionado como renderizado.

UC-05:	**Actualizar Nodos**
Autores:	Sergio García Villanueva
Fuentes:	Sergio García Villanueva
Descripcion:	El sistema deberá comportarse tal y como se describe en el siguiente caso de uso cuando se solicite actualizar los nodos.
	Secuencia normal
1.	El *Sistema* solicita al *Servidor de Aplicación* información sobre los nodos del mapa actual.
2.	El actor *Servidor de Aplicación* devuelve al *Servidor de Aplicación* información sobre los nodos.
3.	El *Sistema* actualiza la información de los nodos.
	Excepciones
3a.	Si no hay comunicación con el *Servidor de Aplicación* el *Sistema* emite un mensaje de error y el caso de uso queda sin efecto.
Postcondición:	La información de los nodos está actualizada.
Frecuencia:	Este caso de uso se espera que se lleve a cabo una vez por segundo.

UC-06:	**Seleccionar Nodo**
Autores:	Sergio García Villanueva
Fuentes:	Sergio García Villanueva
Descripcion:	El sistema deberá comportarse tal y como se describe en el siguiente caso de uso cuando el usuario seleccione un nodo.
	Secuencia normal
1.	El actor *Usuario* selecciona un nodo.
2.	El *Sistema* desplaza el mapa hasta la localización del nodo y resalta el mismo.
3.	El *Sistema* muestra la información relativa al nodo.
4.	El *Sistema* marca el nodo como seleccionado.
	Flujo alternativo
2a.	Si se detecta que el nodo es un mapa, se realiza el caso de uso Renderizar Mapa y el caso de uso continua.
Postcondición:	El estado del nodo es seleccionado.

UC-07:	Modificar Zoom Mapa
Autores:	Sergio García Villanueva
Fuentes:	Sergio García Villanueva
Descripcion:	El sistema deberá comportarse tal y como se describe en el siguiente caso de uso cuando el usuario solicite modificar el zoom del mapa.
	Secuencia normal
1.	El actor *Usuario* solicita modificar el zoom del mapa.
2.	El *Sistema* comprueba el objetivo del zoom.
3.	El *Sistema* solicita al *Servidor de Mapas* las imágenes del nuevo zoom.
4.	El actor *Servidor de Mapas* devuelve al *Sistema* las imágenes solicitadas.
5.	El *Sistema* renderiza el nuevo zoom.
	Flujo alternativo
2a.	Si se detecta que el objetivo del zoom es un mapa, se realiza el caso de uso Renderizar Mapa y el caso de uso queda sin efecto.
Postcondición:	El nuevo zoom está renderizado sobre el mapa.

4.3. Diagrama de Clases

El Diagrama de Clases de la Figura 4.2 representa las clases definidas en la fase de análisis. La idea principal de *Mapache* es obtener un framework cuyo modelo de datos sea dependiente de la aplicación del servidor.

Para alcanzar este objetivo se define un elemento con la mínima información necesaria para el posicionamiento al que se le van añadiendo componentes que amplían su funcionalidad de forma dinámica, este elemento es el nodo. Un nodo es cualquier elemento que pueda ser posicionado en un mapa, desde una persona hasta otros mapas. Los componentes que amplían su funcionalidad son los siguientes:

Posición Este es el componente principal, contiene la información sobre la posición que va a tener el nodo en el mapa, todos los nodos tienen que utilizar este componente.

Mapa Añade al nodo la funcionalidad de mapa. Un mapa define métodos para cambiar entre mapas, ya sean interiores o exteriores, para ello necesita el sistema de referencia de coordenadas del mapa, el zoom mínimo y máximo del mapa, las dimensiones del mismo, y la información de los niveles o plantar del mapa.

Cluster Añade al nodo la funcionalidad de ser un cluster de nodos. Un cluster se encarga de agrupar nodos bajo una misma característica, y define métodos para ocultar y mostrar los nodos que pertenecen al cluster en función del nivel de zoom presente en el mapa.

Información Añade al nodo la funcionalidad de mostrar información detallada sobre el nodo, para ello necesita los datos que le proporciona el servidor, cada uno de estos datos deberá tener un nombre, valor, y la unidad de los mismos.

Estado Añade al nodo la funcionalidad de mostrar el estado del nodo. Para ello necesita conocer el nombre del estado, el color asociado al mismo, y si el estado se va a mostrar sobre un icono aparte o no.

TimeOut Este componente permite a los nodos gestionar su línea de vida si no reciben datos actualizados del servidor. Necesita información sobre los tiempos que tiene que esperar si no recibe información del servidor antes de eliminarse.

4.3.1. Estado del nodo

Se puede ver el estado del nodo en la Figura 4.3. *Mapache* mantiene dos estados concurrentes de un nodo, el propio del framework referente a la visibilidad del mismo que puede modificar el usuario, y el referente a la lógica del servidor de la aplicación.

La idea principal del estado que mantiene de la lógica del servidor reside en que si el servidor no envía información relativa a un nodo, no hay manera de identificar si ha sido por un fallo de la aplicación, de la red, o que la aplicación quiere que se elimine el nodo. Por ello se permite a la aplicación del servidor establecer unos timeouts para cada nodo, de modo que si en ese periodo de tiempo no se recibe información del servidor, se considera que el nodo tiene que ser eliminado.

Si la aplicación quiere eliminar ese nodo directamente, puede cambiar el timeout a cero y el nodo se eliminará en el momento.

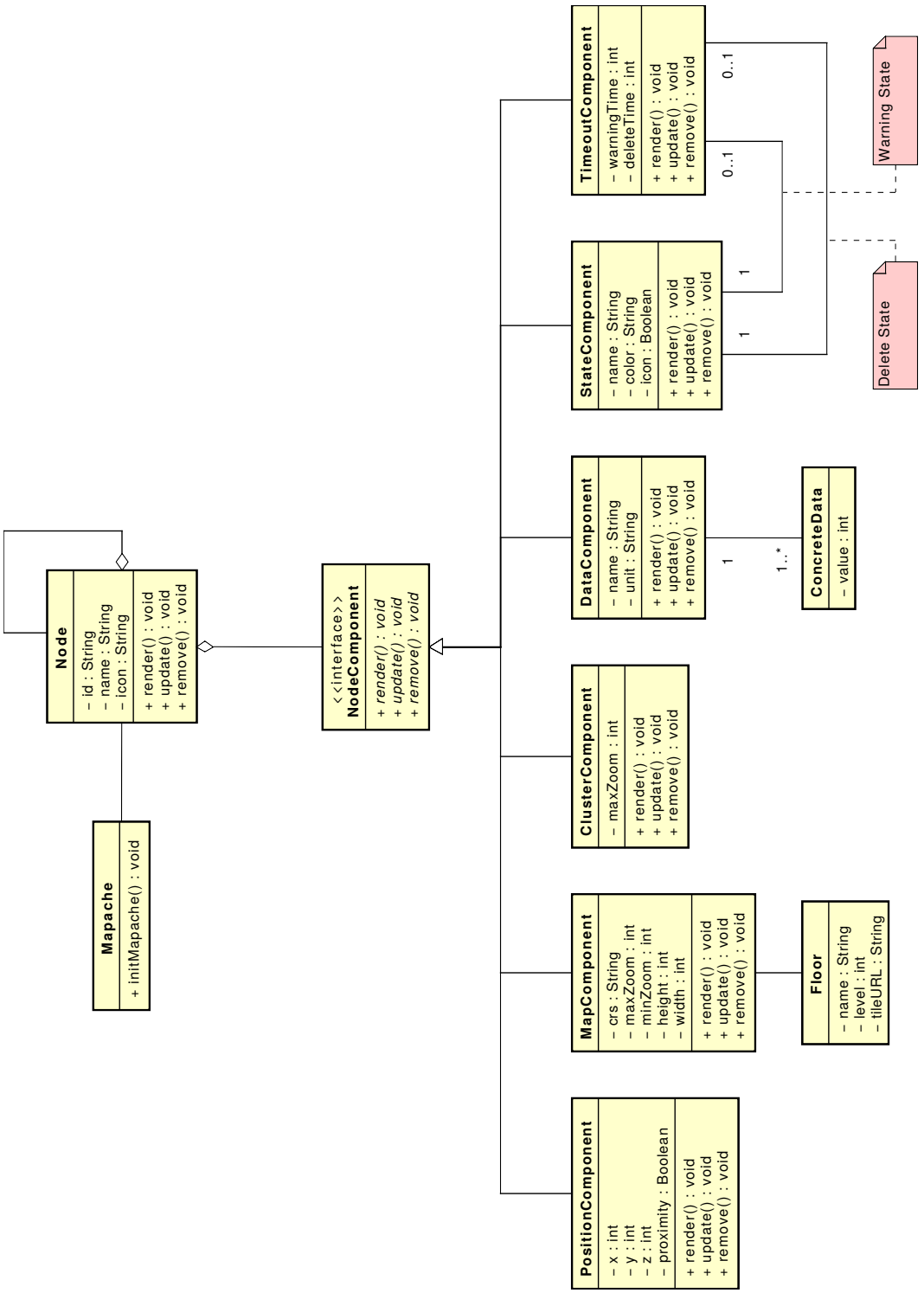


Figura 4.2: Modelo de Dominio de *Mapache*

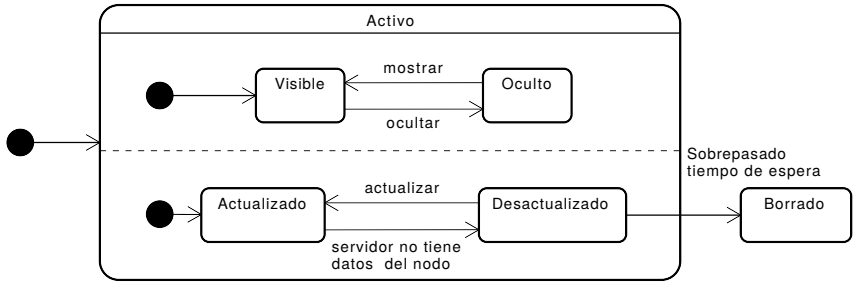


Figura 4.3: Máquina de estado de un Nodo

4.4. Diagramas de secuencia

Se utilizan Diagramas de Secuencia para describir cómo interactúan los actores con el sistema para realizar los Casos de Uso definidos anteriormente.

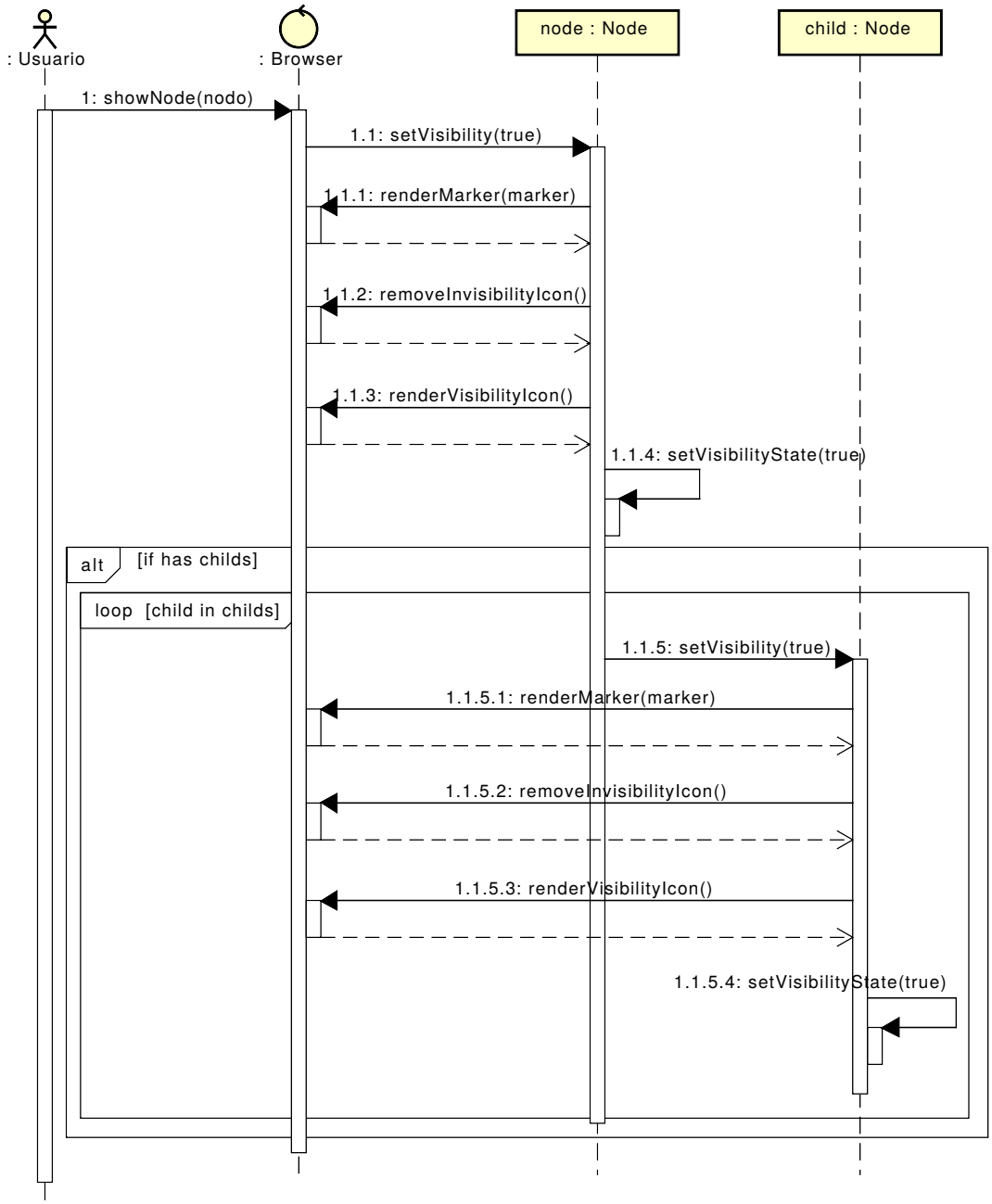


Figura 4.4: Diagrama de secuencia del caso de uso Mostrar Nodo

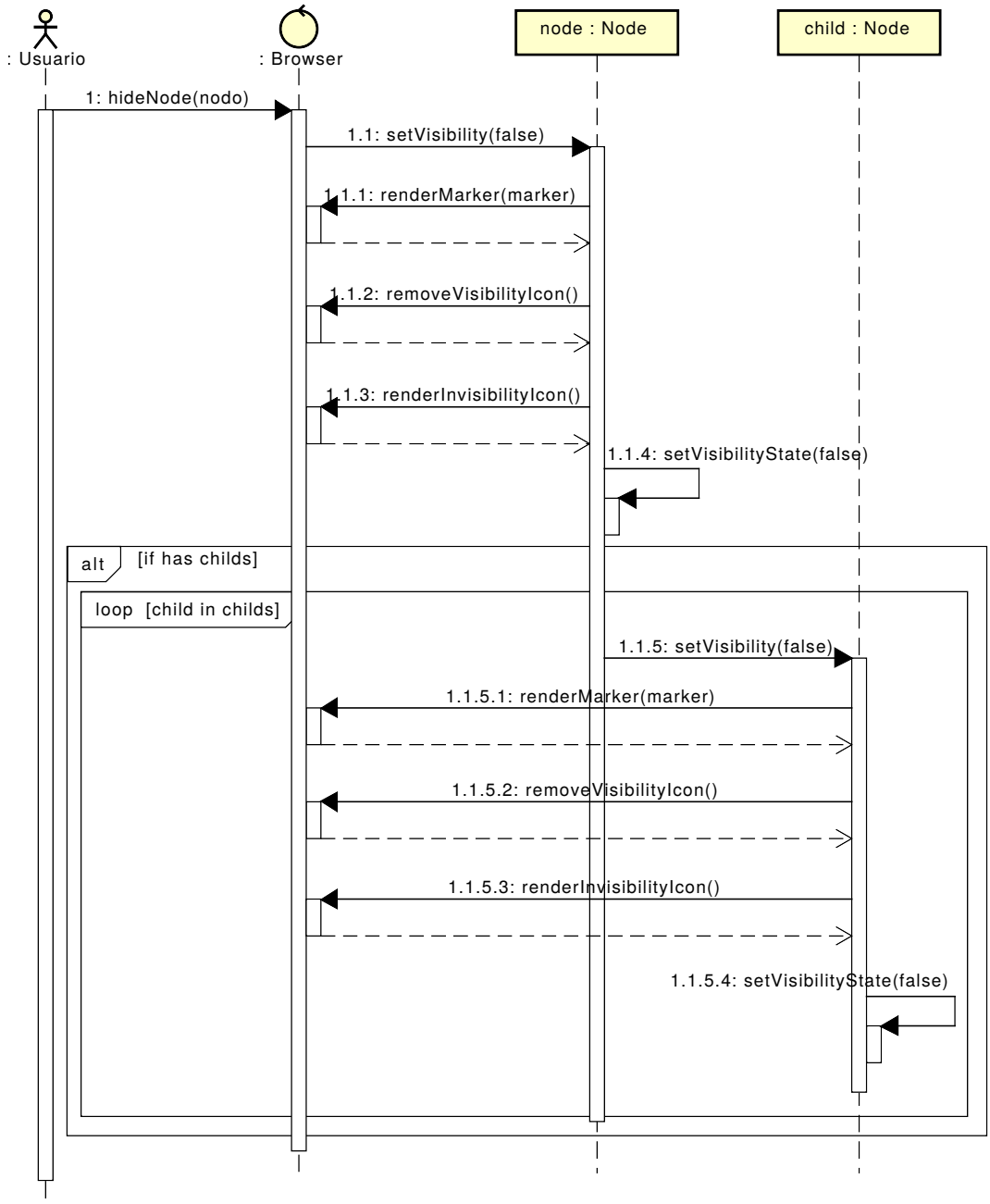


Figura 4.5: Diagrama de secuencia del caso de uso Ocultar Nodo

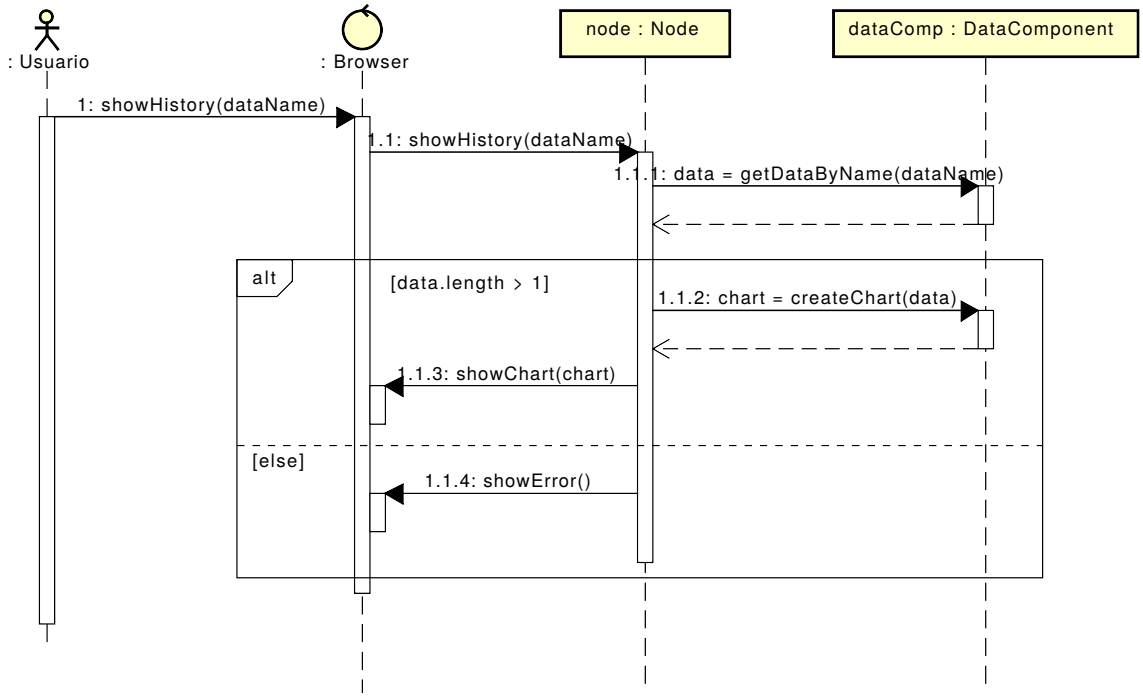


Figura 4.6: Diagrama de secuencia del caso de uso Mostrar Historial de un Nodo

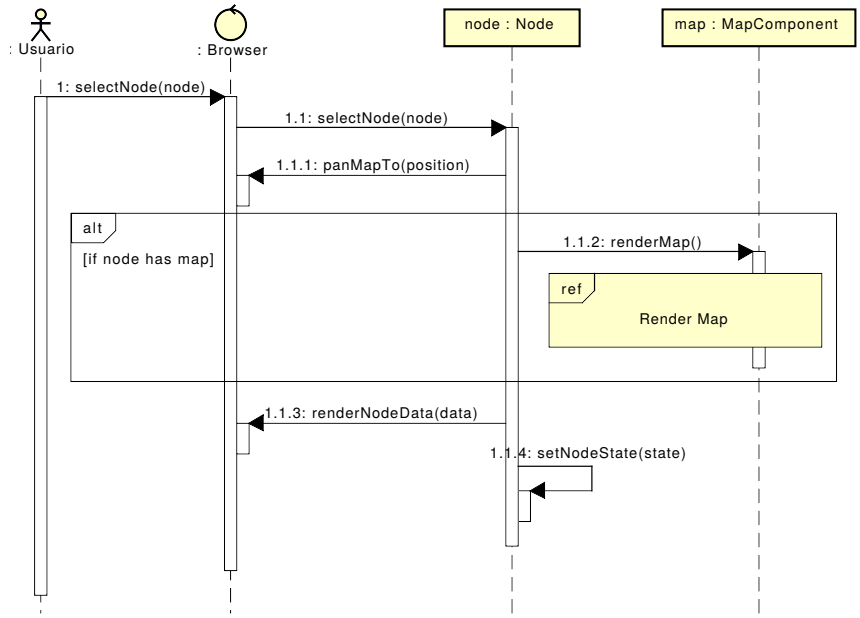


Figura 4.7: Diagrama de secuencia del caso de uso Seleccionar Nodo

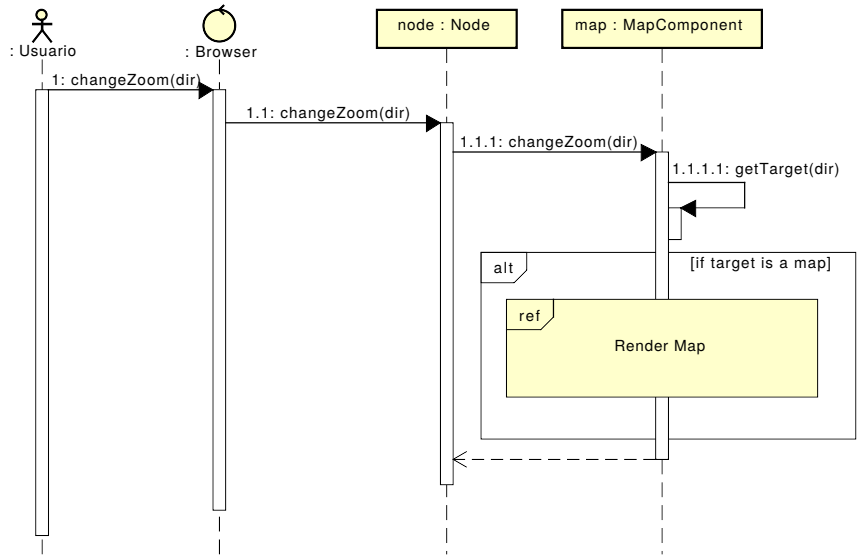


Figura 4.8: Diagrama de secuencia del caso de uso Modificar Zoom Mapa

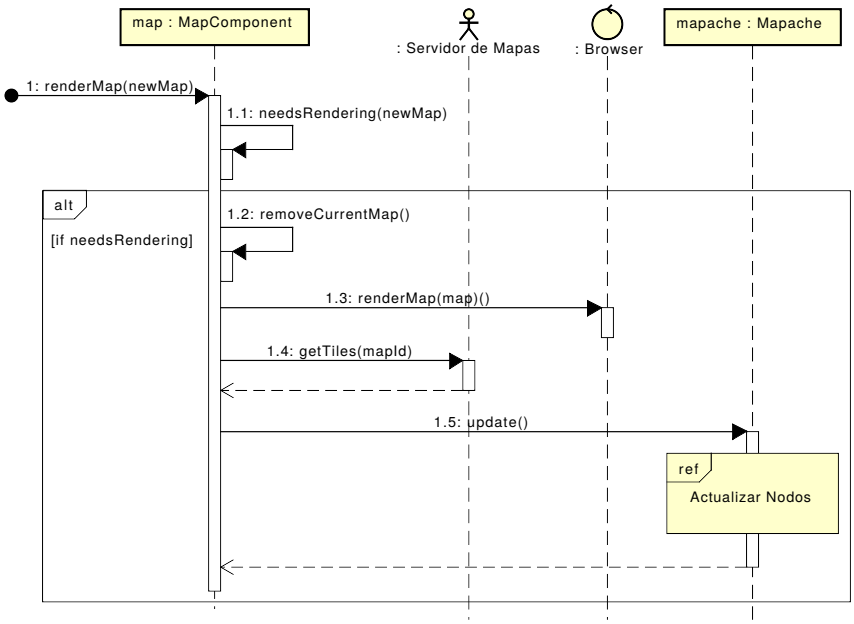


Figura 4.9: Diagrama de secuencia del caso de uso Renderizar Mapa

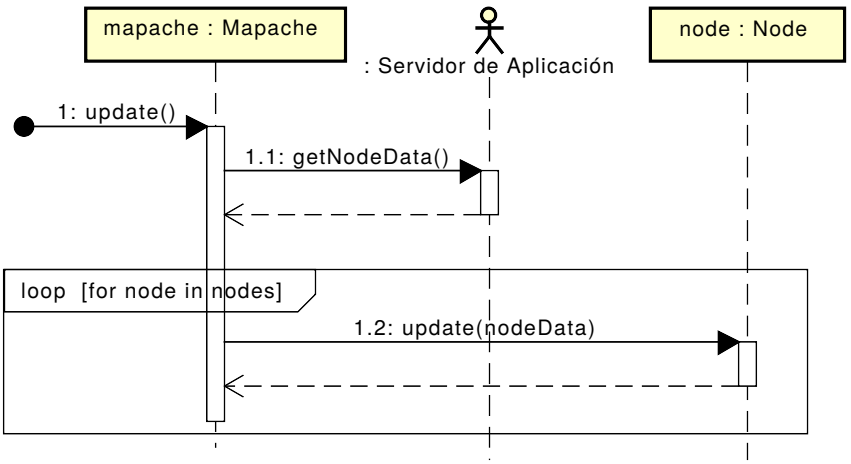


Figura 4.10: Diagrama de secuencia del caso de uso Actualizar Nodos

4.5. Diagramas de Actividad

Los Diagramas de Actividad de la Figura 4.11 y de la Figura 4.12 describen el comportamiento de *Mapache*, profundizando más en los mensajes y estados internos de *Mapache*.

4.5.1. Núcleo de *Mapache*

La parte más importante de *Mapache* es la actualización automática de sus nodos con la información proporcionada por la API del servidor de la aplicación que va a hacer uso del framework.

Una vez cada segundo, *Mapache* solicita al servidor la información actualizada de los nodos de la aplicación. Si el primer nodo recibido es diferente al primer nodo del que tiene información *Mapache*, quiere decir que se ha producido un cambio de mapa y hay que renderizarlo.

Una vez renderizado el mapa, se actualizan los nodos con los datos obtenidos de la respuesta del servidor de la aplicación.

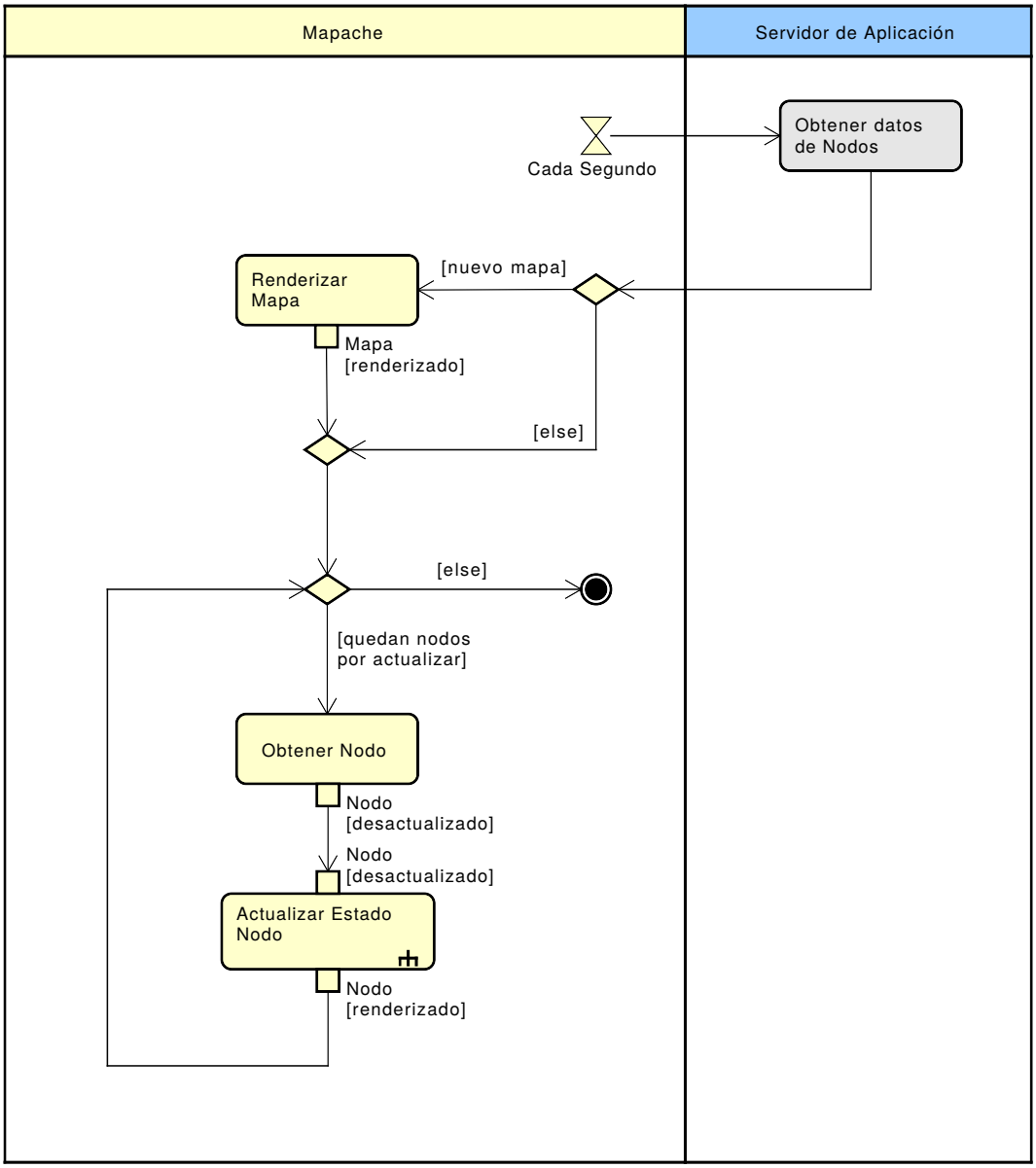


Figura 4.11: Diagrama de actividad del núcleo de *Mapache*

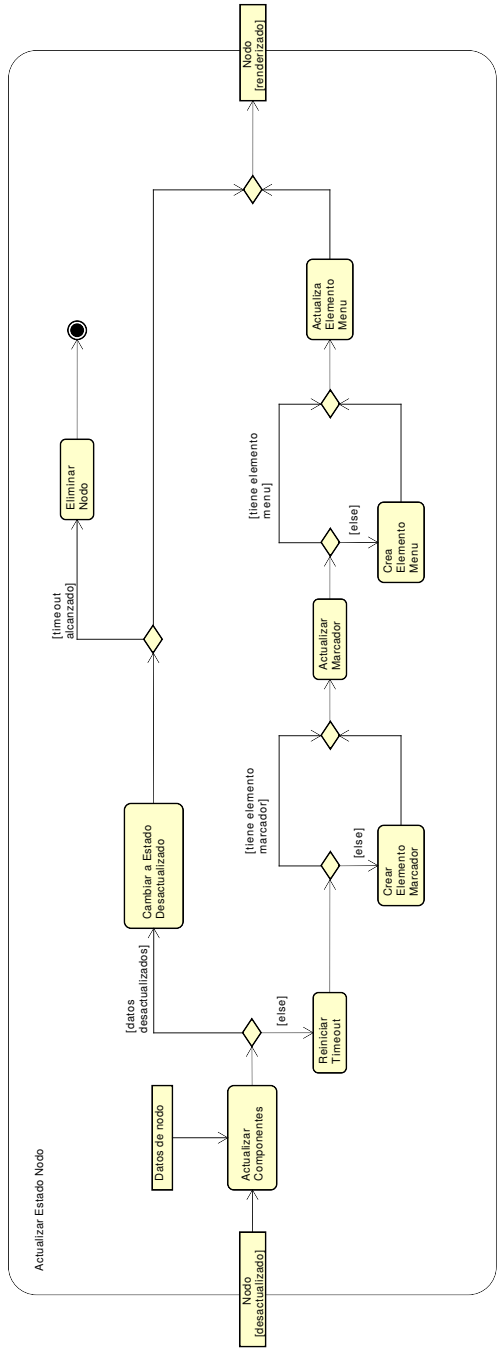


Figura 4.12: Diagrama de actividad de Actualizar Estado de un Nodo

Capítulo 5

Plan de Proyecto

5.1. Objetivos Generales

El objetivo de este proyecto consiste en realizar un framework que permita, mediante una aplicación web, posicionar personas y activos, así como mostrar información relativa a los mismos, sobre una cartografía propia.

El framework debe de dar respuesta a la necesidad de mostrar información sobre personas y activos proporcionada por distintas aplicaciones web.

5.2. Metodología Software

La elaboración del plan de proyecto va a realizarse siguiendo la metodología Rational Unified Process (RUP) [14].

5.2.1. Artefactos de Proyecto

Siguiendo esta metodología deben elaborarse los siguientes artefactos:

- Hito 1 (02/01/2017): Plan de Proyecto Software.
- Hito 2 (08/02/2017): Análisis del Framework.
- Hito 3 (23/03/2017): Diseño Software.
- Hito 4 (16/05/2017): Herramienta completa.

5.2.2. Evolución del Plan

El Plan de Desarrollo del Software se revisará y se modificará antes del comienzo de cada iteración.

5.2.3. Plan de Proceso

Para la elaboración del proyecto se ha decidido utilizar la metodología RUP. Las características de esta metodología son:

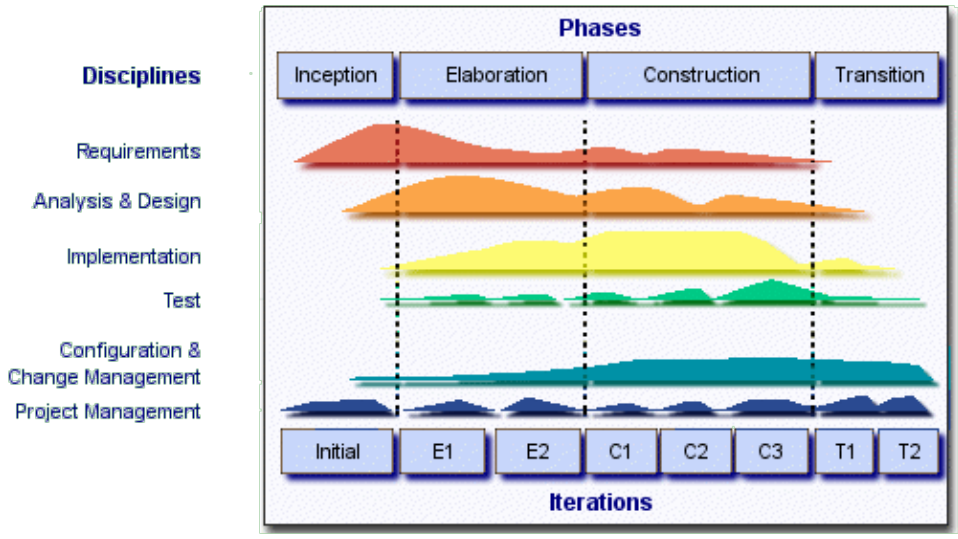


Figura 5.1: Fases de RUP

- **Dirigido por casos de uso:** Ajustándose a las necesidades reales del cliente.
- **Centrado en la arquitectura:** Ajustándose a un patrón que dirigirá la construcción del sistema en las primeras fases.
- **Iterativo e incremental:** Permitiendo ajustar objetivos concretos en cada paso.

Ciclo de Vida del Proyecto

El proceso iterativo e incremental de RUP está caracterizado por la realización en paralelo de todas las disciplinas de desarrollo a lo largo del proyecto como se puede ver en la figura 5.1, con lo cual la mayoría de los artefactos son generados en etapas tempranas y van evolucionando según avanza el proyecto.

El ciclo de vida en RUP se descompone en cuatro fases:

- **Inicio:** Elaborar un documento de visión general en el que se definen los requisitos generales del proyecto, las características principales y las restricciones.
- **Elaboración:** Analizar el el dominio del problema, estableciendo una arquitectura base sólida, y desarrollar un plan de proyecto ajustado para la fase de construcción.
- **Construcción:** Elaborar versiones del producto utilizables, y prueba de las mismas frente a los criterios de evaluación.
- **Transición:** Validación del sistema, y elaboración de la documentación necesaria para la entrega.

5.3. Gestión del Proceso

A continuación se detallan los diferentes planes de la Gestión del Proceso.

5.3.1. Plan de Puesta en Marcha

El desarrollo del proyecto es individual, por lo que la disponibilidad del personal es total.

Para la realización del proyecto es necesario tener las siguientes aptitudes:

- Manejo de sistemas Linux.
- Conocimiento de lenguajes JavaScript, HTML, y CSS3.
- Conocimiento del estándar ES6 de JavaScript.
- Conocimiento del gestor de paquetes JavaScript NPM.
- Conocimiento de librerías de Javascript como JQuery, Browserify, y Babel.
- Conocimiento del preprocesador de estilos SASS.
- Conocimiento de PostgreSQL y PostGIS.

5.3.2. Plan de Trabajo

A continuación se detallan las actividades en las que está dividido el proyecto, así como la calendarización de las mismas.

ID: 01	**Aprendizaje de JavaScript**
Predecesoras:	-
Duración:	20 días
Recopilar información del lenguaje JavaScript, en concreto leer la documentación sobre el estándar ES6 (ECMAscript v6).	

ID: 02	**Recopilación de Librerías**
Predecesoras:	01
Duración:	10 días
Investigar sobre las librerías disponibles para JavaScript, en concreto aquellas relacionadas con sistemas en tiempo real.	

ID: 03	Investigación sobre GIS
Predecesoras:	-
Duración:	10 días
Esta investigación tiene como objetivo recopilar información sobre los servidores de mapas más conocidos, en concreto información acerca de la cantidad de información cartográfica del servidor, el gestor de base de datos que utilizan, posibilidad de modificación de la cartografía, licencias, y costes.	

ID: 04	Investigación Servidores GIS
Predecesoras:	03
Duración:	5 días
Esta investigación tiene como objetivo recopilar información sobre los servidores de mapas más conocidos, en concreto datos acerca de la cantidad de información cartográfica del servidor, el gestor de base de datos que utilizan, posibilidad de modificación de la cartografía ,y costes.	

ID: 05	Investigación Mapas Interactivos
Predecesoras:	04
Duración:	3 días
Esta investigación tiene como objetivo recopilar información sobre librerías de creación de mapas interactivos en aplicaciones web.	

ID: 06	Control de Versiones
Predecesoras:	-
Duración:	1 día
Configuración de un repositorio web basado en Git para el mantenimiento de versiones de la aplicación.	

ID: 07	Experimentación
Predecesoras:	02 | 05
Duración:	5 días
Se crearán scripts para realizar pruebas sobre las librerías encontradas. El objetivo de la experimentación es obtener un análisis más detallado sobre las fortalezas y debilidades de la librerías en cuestión.	

ID: 08	Planificación de Riesgos
Predecesoras:	07
Duración:	5 días
Se definirán los riesgos del proyecto, describiendo los riesgos, el impacto, la probabilidad de suceso, y los planes de protección y contingencia.	

ID: 09	Definición de Actividades
Predecesoras:	08
Duración:	4 días
Se definirán las actividades necesarias para completar el proyecto. A continuación se realizará la secuenciación de las mismas.	

ID: 10	Calendarización
Predecesoras:	09
Duración:	2 días
Con la actividades definidas, se realizará una estimación de la duración de las mismas y posteriormente se eleborará el diagrama de Gantt.	

ID: 11	Plan de Desarrollo
Predecesoras:	10
Duración:	10 días
Redactar el Plan de Desarrollo del Proyecto.	

ID: 12	Elicitación de Requisitos
Predecesoras:	11
Duración:	5 días
Se detallarán los requisitos del proyecto en función de la información obtenida en las fases previas del desarrollo.	

ID: 13	Casos de Uso
Predecesoras:	12
Duración:	5 días
Se obtendrán los Casos de Uso en función de los Requisitos del proyecto. Posteriormente se elaborará el diagrama de casos de uso, y la descripción detallada de los mismos.	

ID: 14	Modelo de Dominio
Predecesoras:	13
Duración:	7 días
Utilizando el lenguaje UML, se definirá el modelo de dominio del framework.	

ID: 15	Diagramas de Secuencia
Predecesoras:	14
Duración:	10 días
Se obtendrán los diagramas de secuencia correspondientes a los Casos de Uso detallados anteriormente.	

ID: 16	Servidor de Mapas
Predecesoras:	15
Duración:	10 días
Utilizando la base de datos cartográfica de OSM, se montará el servidor de tiles que utilizará el framework para mostrar el mapa del mundo en una aplicación web.	

ID: 17	Renderizado Mapa Exterior
Predecesoras:	16
Duración:	5 días
Utilizando el servidor de mapas anteriormente puesto en marcha, se añadirá la funcionalidad al framework para el renderizado básico del mapa exterior sobre la aplicación web.	

ID: 18	Localización de Activos
Predecesoras:	17
Duración:	3 días
Mediante peticiones al servidor de la aplicación web, localizar sobre el mapa exterior los activos de los que se quiere hacer un seguimiento.	

ID: 19	Información en Tiempo Real
Predecesoras:	18
Duración:	3 días
Con los activos en el mapa, mostrar información detallada sobre los mismos en tiempo real, esta información es dependiente de la aplicación web, por lo que puede variar de una aplicación a otra.	

ID: 20	Soporte Coordenadas ETRS89
Predecesoras:	18
Duración:	2 días
Se modificará el framework para poder localizar activos utilizando el sistema de coordenadas ETRS89 (European Terrestrial Reference System).	

ID: 21	Servidor Mapas de Interiores
Predecesoras:	19 | 20
Duración:	5 días
Utilizando planos de edificios, se montará el servidor que utilizará el framework para mostrar planos de interiores en una aplicación web.	

ID: 22	Renderizado Mapas Interiores
Predecesoras:	21
Duración:	3 días
Utilizando el servidor de planos interiores, se modificará el framework para que de soporte a mapas que muestren el interior de edificios.	

ID: 23	Soporte Coordenadas Relativas
Predecesoras:	22
Duración:	2 días
Se modificara el framework para que de soporte a coordenadas relativas, necesarias para tener un sistema de coordenadas de planos interiores.	

ID: 21	Clustering de Nodos
Predecesoras:	23
Duración:	4 días
Añadir al framework funcionalidad para que, a partir de la información proporcionada por el servidor, los nodos puedan ser agrupados en conjuntos sobre un mapa.	

ID: 25	Modularizar Funcionalidades
Predecesoras:	24
Duración:	10 días
Creación de módulos en el framework con diferentes funcionalidades para que, dependiendo de la aplicación que utilice el framework, pueda seleccionar módulos en función de sus necesidades.	

ID: 26	Integracion Indoor/Outdoor
Predecesoras:	24
Duración:	8 días
Integración de los planos interiores con la cartografía exterior.	

ID: 27	Compatibilidad entre Navegadores
Predecesoras:	25 | 26
Duración:	2 días
Modificar el framework y utilizar librerías para que se pueda ejecutar sin errores en todos los navegadores web.	

ID: 28	Diseño Responsive
Predecesoras:	25 | 26
Duración:	3 días
Modificar el framework y sus librerías para obtener la mejor experiencia de usuario en diferentes dispositivos con diferentes resoluciones de pantalla.	

ID: 29	Internalización
Predecesoras:	25 | 26
Duración:	2 días
Se modificará el framework para que el contenido se muestre en el idioma preferido por el usuario final.	

ID: 30	Plan de Pruebas
Predecesoras:	27 | 28 | 29
Duración:	5 días
Implementar las pruebas necesarias para asegurar la ejecución correcta del framework.	

ID: 31	Verificación
Predecesoras:	30
Duración:	3 días
Comprobación de que el framework se ejecuta correctamente en distintos navegadores web.	

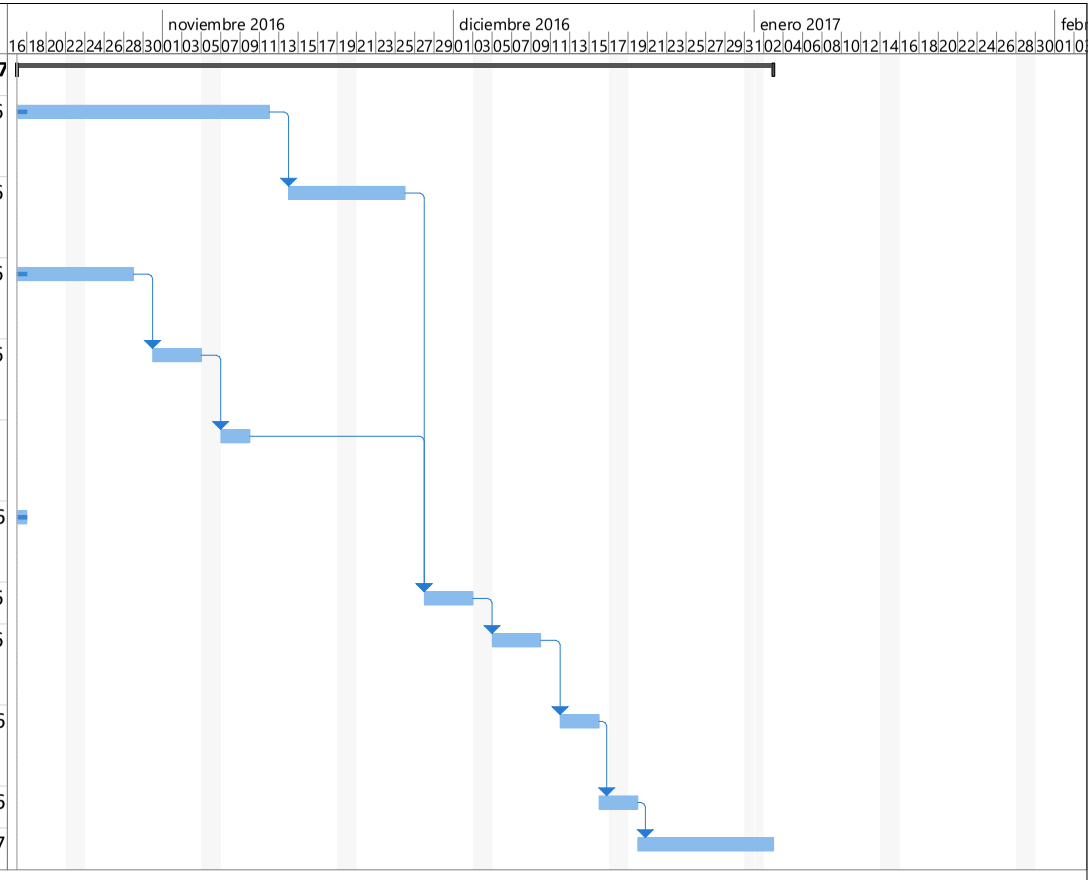
ID: 32	Manual de Instalación
Predecesoras:	31
Duración:	1 día
Elaboración del manual de instalación del framework.	

ID: 33	Manual de Usuario
Predecesoras:	31
Duración:	1 día
Elaboración del manual de usuario del framework.	

5.3.3. Planificación de las Actividades

A continuación se incluyen los diagramas de Gantt de las fases del proyecto.

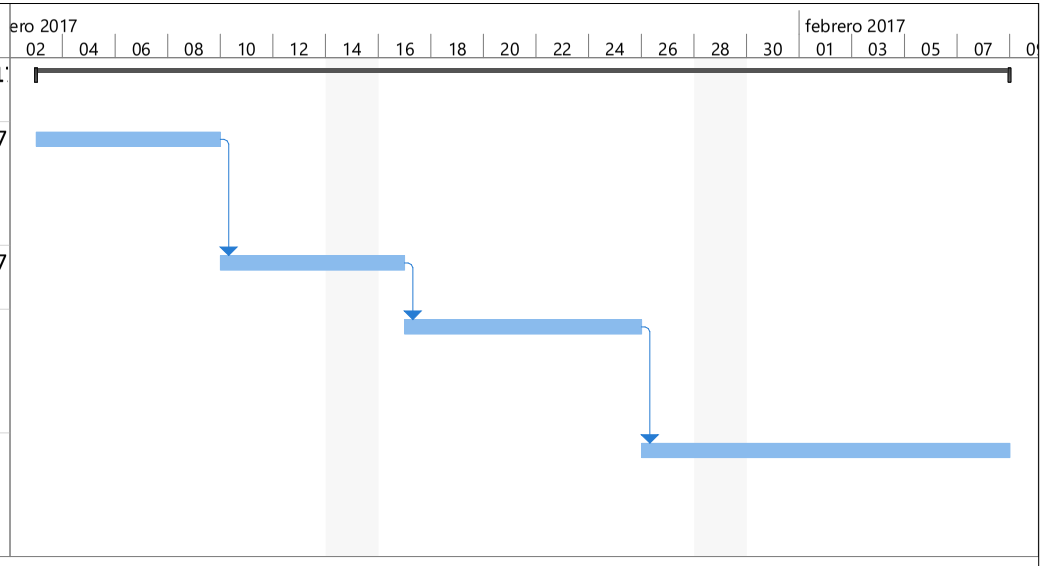
Id	Nombre de tarea	Duración	Comienzo	Fin
1	**Inicio**	**56 días**	**lun 17/10/16**	**lun 02/01/17**
2	Aprendizaje JavaScript	20 días	lun 17/10/16	vie 11/11/16
3	Recopilación de Librerías	10 días	lun 14/11/16	vie 25/11/16
4	Investigación sobre GIS	10 días	lun 17/10/16	vie 28/10/16
5	Investigación sobre Servidores GIS	5 días	lun 31/10/16	vie 04/11/16
6	Investigación Mapas Interactivos	3 días	lun 07/11/16	mié 09/11/16
7	Control de Versiones	1 día	lun 17/10/16	lun 17/10/16
8	Experimentación	5 días	lun 28/11/16	vie 02/12/16
9	Planificación de Riesgos	5 días	lun 05/12/16	vie 09/12/16
10	Definición de Actividades	4 días	lun 12/12/16	jue 15/12/16
11	Calendarización	2 días	vie 16/12/16	lun 19/12/16
12	Plan de Desarrollo	10 días	mar 20/12/16	lun 02/01/17



Proyecto: Mapache
Fecha: vie 07/04/17

Tarea	Resumen inactivo	Tareas externas	
División	Tarea manual	Hito externo	
Hito	solo duración	Fecha límite	
Resumen	Informe de resumen manual	Progreso	
Resumen del proyecto	Resumen manual	Progreso manual	
Tarea inactiva	solo el comienzo		
Hito inactivo	solo fin		

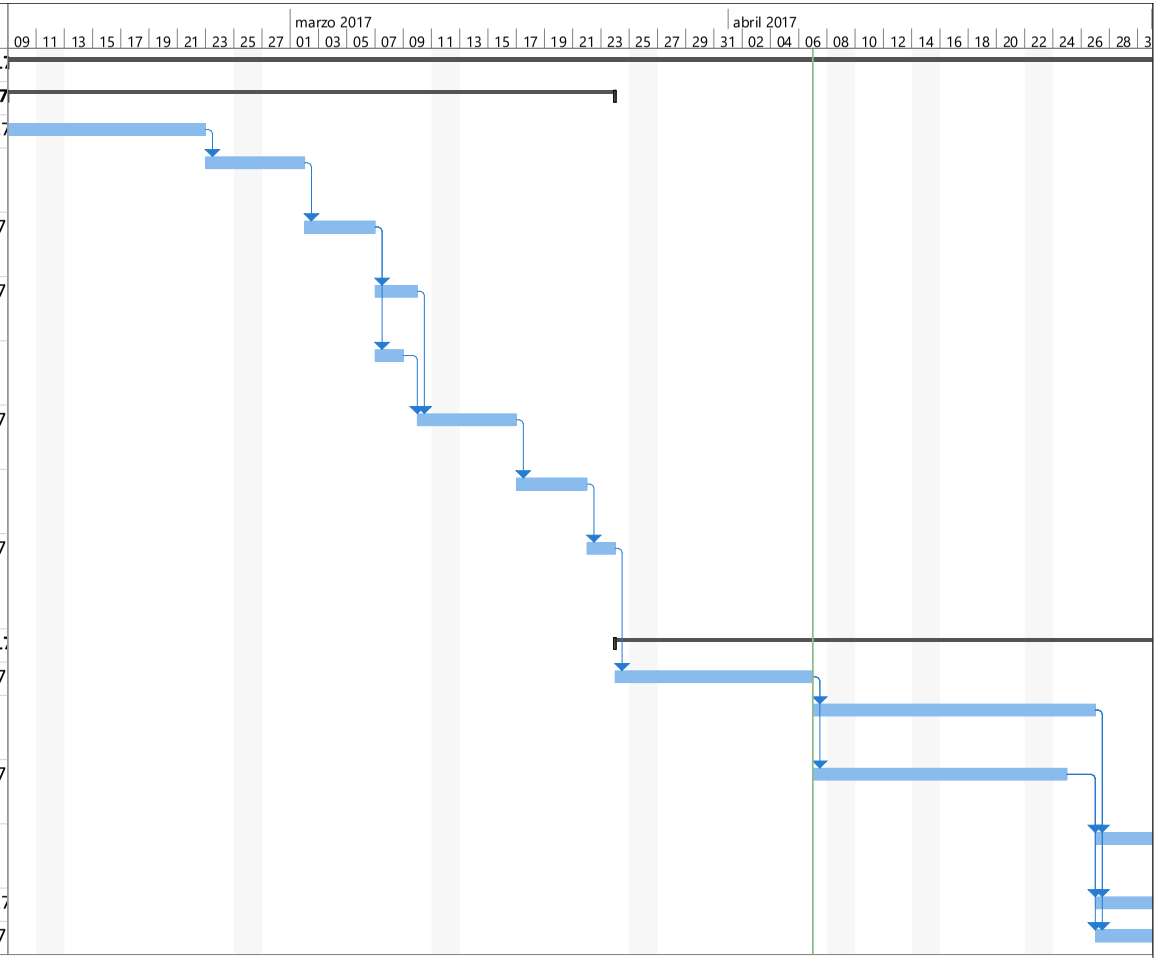
Id	Nombre de tarea	Duración	Comienzo	Fin
13	**Elaboración**	**27 días**	**mar 03/01/1**	**mié 08/02/1**
14	Elicitación de Requisitos	5 días	mar 03/01/17	lun 09/01/17
15	Casos de Uso	5 días	mar 10/01/1	lun 16/01/17
16	Modelo de Dominio	7 días	mar 17/01/17	mié 25/01/17
17	Diagramas de Secuencia	10 días	jue 26/01/17	mié 08/02/17



Proyecto: Mapache
Fecha: vie 07/04/17

Tarea	Resumen inactivo	Tareas externas
División	Tarea manual	Hito externo
Hito	solo duración	Fecha límite
Resumen	Informe de resumen manual	Progreso
Resumen del proyecto	Resumen manual	Progreso manual
Tarea inactiva	solo el comienzo	
Hito inactivo	solo fin	

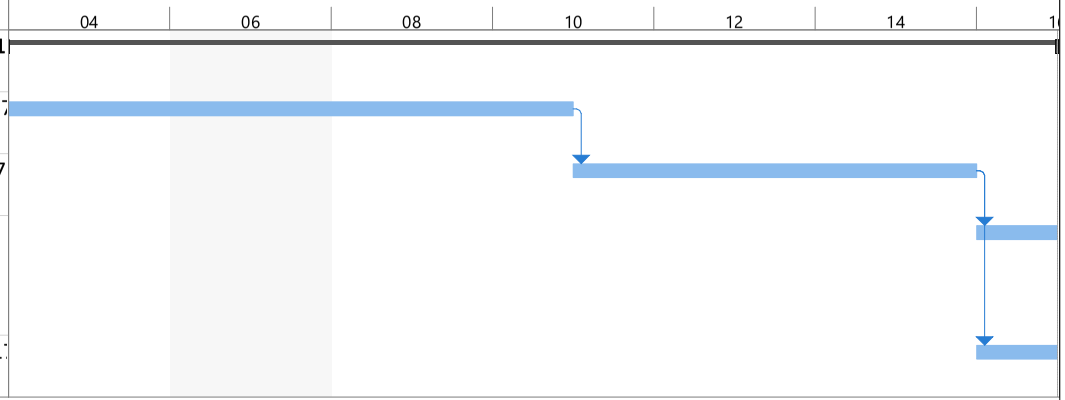
Id	Nombre de tarea	Duración	Comienzo	Fin
18	**Construcción**	**58 días**	**jue 09/02/17**	**mié 03/05/17**
19	**Iteración 1**	**31 días**	**jue 09/02/17**	**jue 23/03/17**
20	Servidor de Mapa	10 días	jue 09/02/17	mié 22/02/17
21	Renderizado Mapa Exterior	5 días	jue 23/02/17	mié 01/03/17
22	Localización de Activos	3 días	jue 02/03/17	lun 06/03/17
23	Información en Tiempo Real	3 días	mar 07/03/17	jue 09/03/17
24	Soporte Coordenadas ETRS89	2 días	mar 07/03/17	mié 08/03/17
25	Servidor de Mapas Interiores	5 días	vie 10/03/17	jue 16/03/17
26	Renderizado Mapa Interior	3 días	vie 17/03/17	mar 21/03/17
27	Soporte Coordenadas Relativas	2 días	mié 22/03/17	jue 23/03/17
28	**Iteración 2**	**27 días**	**vie 24/03/17**	**mié 03/05/17**
29	Clustering de No	10 días	vie 24/03/17	jue 06/04/17
30	Modularizar Funcionalidades	12 días	vie 07/04/17	mié 26/04/17
31	Integración Indoor/Outdoor	10 días	vie 07/04/17	lun 24/04/17
32	Compatibilidad entre Navegadores	5 días	jue 27/04/17	mié 03/05/17
33	Diseño Responsiv	5 días	jue 27/04/17	mié 03/05/17
34	Internacionalizac	3 días	jue 27/04/17	lun 01/05/17



Proyecto: Mapache
Fecha: vie 07/04/17

Tarea	Tarea inactiva	Informe de resumen manual	Hito externo
División	Hito inactivo	Resumen manual	Fecha límite
Hito	Resumen inactivo	solo el comienzo	Progreso
Resumen	Tarea manual	solo fin	Progreso manual
Resumen del proyecto	solo duración	Tareas externas	

Id	Nombre de tarea	Duración	Comienzo	Fin
35	**Transición**	**9 días**	**jue 04/05/17**	**mar 16/05/1**
36	Pruebas	5 días	jue 04/05/17	mié 10/05/1
37	Verificación	3 días	jue 11/05/17	lun 15/05/17
38	Manual de Instalación	1 día	mar 16/05/17	mar 16/05/17
39	Manual de Usuario	1 día	mar 16/05/1	mar 16/05/1



Proyecto: Mapache
Fecha: vie 07/04/17

Tarea	Resumen inactivo	Tareas externas
División	Tarea manual	Hito externo
Hito	solo duración	Fecha límite
Resumen	Informe de resumen manual	Progreso
Resumen del proyecto	Resumen manual	Progreso manual
Tarea inactiva	solo el comienzo	
Hito inactivo	solo fin	

5.3.4. Plan de Control

A continuación se detallan las acciones a realizar respecto al control en el desarrollo del proyecto.

Gestión de Requisitos

En caso de que existan modificaciones en los requisitos una vez el proyecto está en curso, se efectuarán las siguientes tareas:

1. Establecer una prioridad para los nuevos cambios.
2. Analizar las consecuencias derivadas de estos cambios.
3. Incorporar los cambios al plan.
4. Evaluar posibles riesgos derivados de los nuevos cambios.
5. Replanificar la calendarización en caso de ser necesario.

Control de Calendario

Para el control temporal de las actividades, se han establecido unos hitos para poder controlar desviaciones respecto a lo planificado, y así poder tomar las medidas correctivas correspondientes.

5.3.5. Plan de Gestión de Riesgos

A continuación se detallan los riesgos detectados, junto con un plan de protección y contingencia. Para obtener la exposición al riesgo haremos uso de la tabla 5.1.

R-01 Máquina personal averiada

El desarrollo del proyecto se realizará en la máquina personal del alumno.

Impacto Catastrófico

Probabilidad 5 %

Exposición Moderada

Plan de Protección Utilizar un sistema de control de versiones para tener una copia de seguridad en otra máquina.

Plan de Contingencia Al estar el trabajo guardado en otra máquina, solo habría que cambiar la máquina.

Tabla 5.1: Matriz Impacto/Probabilidad

Impacto\ Probabilidad	Muy Alto	Alto	Medio	Bajo	Muy Bajo
Catastrófico	Alto	Alto	Moderado	Moderado	Bajo
Crítico	Alto	Alto	Moderado	Bajo	Ninguno
Marginal	Moderado	Moderado	Bajo	Ninguno	Ninguno
Despreciable	Moderado	Bajo	Bajo	Ninguno	Ninguno

R-02 Pérdida de datos

Pérdida de los ficheros relacionados con el proyecto, desde documentación hasta los códigos fuente de la aplicación.

Impacto Catastrófico

Probabilidad 1 %

Exposición Baja

Plan de Protección Utilizar un sistema de control de versiones para tener una copia de seguridad en otra máquina.

Plan de Contingencia Si se perdiese la información del sistema de versiones, habría que evaluar el impacto de los ficheros perdidos sobre el proyecto y replanificar en consecuencia.

R-03 Enfermedad

Es posible que por motivos de enfermedad no se pueda trabajar las horas diarias establecidas.

Impacto Crítico

Probabilidad 1 %

Exposición Baja

Plan de Protección Ninguno.

Plan de Contingencia Evaluar el tiempo que se ha perdido y replanificar en consecuencia.

R-04 Cambios en librerías de terceros

Las librerías de JavaScript están en cambio constante, y un cambio en las especificaciones de las mismas puede provocar el mal funcionamiento de la aplicación.

Impacto Catastrófico

Probabilidad 15 %

Exposición Moderada

Plan de Protección Utilizar siempre versiones estables, sin dependencia de archivos en repositorios remotos que pueden cambiar con nuevas versiones.

Plan de Contingencia Evaluar si es mejor corregir el fallo, o cambiar de versión o librería.

R-05 Bugs en la aplicación

El framework puede presentar bugs durante el desarrollo.

Impacto Crítico

Probabilidad 10 %

Exposición Baja

Plan de Protección Escribir un código legible y comentado.

Plan de Contingencia Utilizar herramientas de debugging, arreglar los errores y replanificar si es necesario.

R-06 Fallo en el diseño

El diseño elaborado en las primeras etapas no es el ideal.

Impacto Crítico

Probabilidad 20 %

Exposición Moderada

Plan de Protección Prestar especial atención a la fase de análisis y recolectar correctamente los requisitos.

Plan de Contingencia De ser un fallo importante en las bases del diseño habría que replanificar en el momento, de tener gran importancia habría que usar la siguiente iteración del plan de proceso y ver si hay que hacer un replanificación.

5.3.6. Presupuesto

Con la planificación detalladaposteriormente, el proyecto se desarrolla en un plazo de 7 meses. El salario para un Ingeniero Informático es de aproximadamente 22 000 € al año, más 7 300 € de costes sociales, se estima un presupuesto de 12 210 € más IVA.

Capítulo 6

Diseño

El siguiente paso en el desarrollo de *Mapache* es definir la arquitectura, el modelo de dominio, diagramas de secuencia, patrones de diseño utilizados, y el modelo de comunicación con el servidor del sistema.

6.1. Arquitectura del Framework

La arquitectura de *Mapache*, mostrada en la Figura 6.1, está organizada siguiendo un modelo de capas en la que cada capa proporciona un conjunto de servicios a la capa inferior. Esta estructura de capas es la siguiente:

JavaScript Utilities Capa que aporta la funcionalidad básica de JavaScript como el manejo de arrays, manejadores de eventos del DOM, funciones de manipulación del DOM [10], y funciones matemáticas.

jQuery Aporta funcionalidad extra a las utilidades de JavaScript para la manipulación del DOM HTML, y la creación de llamadas HTTP asíncronas usando la tecnología *AJAX*.

d3 D3.js [1] es una librería para manejar documentos basados en información. Permite crear, a partir de un array de datos, tablas y gráficos utilizando los estándares web actuales, HTML5, SVG, CSS por lo que la compatibilidad con navegadores modernos es total.

Leaflet Contiene la funcionalidad para la creación de mapas y marcadores.

Lógica del Dominio Contiene los objetos de nuestro dominio del sistema.

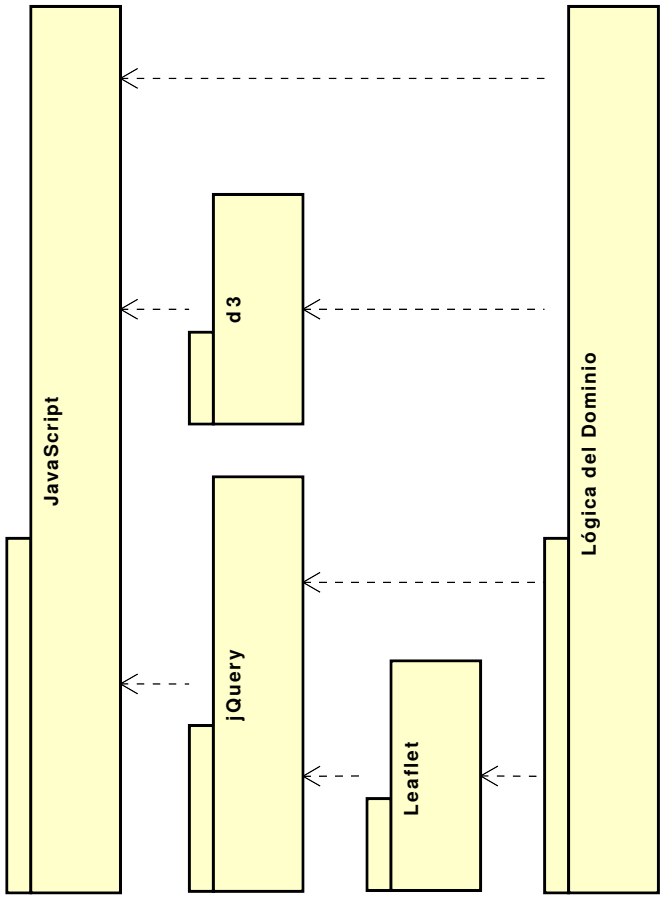


Figura 6.1: Arquitectura general de *Mapache*

6.2. Capa de Lógica del Dominio

En este apartado se describe la descomposición de la capa de lógica del dominio. Esta capa representa el núcleo de *Mapache*, contiene todos los objetos que participan en el dominio del problema que se quiere implementar, siguiendo la arquitectura Flux. La división en paquetes es la siguiente:

Store Contiene el estado del framework. Este estado sólo puede ser modificado a través de una *Acción*. Una Acción es un objeto plano que representa una intención de modificar el estado de la aplicación, en *Mapache* estas acciones corresponden con eventos de UI, y con la respuesta del servidor de la aplicación. Tras actualizar el estado, lo envía a las Vistas para que actualicen el DOM.

Views Son las encargadas de crear y actualizar los elementos del DOM HTML.

Dispatcher Se encarga de distribuir las Acciones hacia la Store para que éstas modifiquen su estado.

Utils Contiene clases que aportan operaciones que se repiten a lo largo del desarrollo del framework como funciones de modificación del DOM, u operaciones sobre arrays.

6.2.1. Store

Contiene el estado del framework, y lo utiliza para cambiar la vista de la aplicación. *Mapache* utiliza la vista a dos niveles para mostrar la información de los nodos, el primero es a nivel de marcador en el mapa, y el segundo es a nivel de menú que se mostrará en la parte izquierda de la vista. Por ello, los componentes de un nodo deben de definir métodos para modificar la vista del menú y del marcador (aunque algunos componentes sólo hacen uso de una de ellas).

Mapache Clase que sirve de interfaz entre *Mapache* y el desarrollador que va a utilizar el framework. A su vez, es la encargada de realizar de forma automática las llamadas al servidor para actualizar el estado de la aplicación.

Map Mantiene la referencia al mapa renderizado en la vista y define métodos para modificar el mismo.

Node Clase que contiene los métodos para crear, actualizar, y eliminar los nodos del mapa. También se encarga de realizar las mismas operaciones sobre sus nodos hijo.

NodeComponent Clase que representa un componente de un nodo, añade funcionalidad extra a este de manera dinámica.

Decorator Clase que mantiene la referencia al componente anterior y define los métodos de creación, actualización, y eliminación del componente.

PositionElement Clase que contiene los métodos para posicionar un elemento en un mapa, y crear el contenedor HTML principal del marcador, y del menú del nodo.

MapDecorator Clase que contiene los métodos para renderizar un mapa al seleccionar este nodo. También define métodos para cambiar entre plantas de el mapa que representa este nodo.

ClusterDecorator Clase que contiene los métodos para controlar si hay que mostrar el nodo cluster, o los hijos de este en el mapa en función del nivel de zoom que tenga el mapa.

DataDecorator Clase que mantiene la información que se quiere mostrar del nodo. Define métodos para mostrar esta información en el menú, y mostrar un historial de datos para un periodo de tiempo solicitado.

StateDecorator Clase que define métodos para cambiar el estado del nodo tanto en el marcador como en el menú.

TimeoutDecorator Clase que define métodos para actualizar el estado del nodo cuando no llegan datos del servidor sobre este.

6.2.2. Views

Las vistas son componentes del DOM HTML que son actualizados cuando se produce un cambio en el estado del framework almacenado en la Store. Estas vistas encapsulan toda la lógica de cómo el framework interactúa con el DOM. *Mapache* posee cuatro tipos de vistas que se describen a continuación:

Main Menu El componente que contiene los componentes del menú principal del framework, es la encargada de mostrar un menú lateral sobre el que *Mapache* va a renderizar todos los nodos almacenados en la Store.

Map Node Encargada de gestionar y crear los marcadores del mapa, apoyándose en los métodos que ofrece Leaflet para posicionar estos en el mapa.

Menu Item El objetivo de este componente es mostrar información básica de un nodo sobre el menú principal de *Mapache* en forma de lista.

Menu Data Est componente recibe los datos detallados de un nodo, y utiliza los métodos proporcionados por la librería d3 para tratar estos datos y mostrarlos en forma de gráficas y diagramas de barras en el menú principal de *Mapache*.

6.2.3. Utils

Contiene métodos auxiliares que se utilizan a lo largo del desarrollo de *Mapache* en las diferentes clases del dominio, y se encapsulan en este paquete para seguir el concepto de *Don't Repeat Yourself*.

Arrays JavaScript trae en su nuevo estándar métodos para el manejo de arrays. Esta clase se apoya en estos métodos para crear otros más avanzados que se van a utilizar en el desarrollo de *Mapache* de manera frecuente.

CRS Esta clase aporta métodos para la creación de las diferentes capas de *Leaflet* independientemente del sistema de referencia geográfico que se vaya a utilizar en *Mapache*.

DOMUtil Esta clase permite crear y modificar elementos del DOM HTML.

6.3. Diagrama de Clases de Diseño

A continuación se muestra en la Figura 6.2 el diagrama de clases de diseño. Para poder visualizarlo mejor, primero se muestra el diagrama, y posteriormente las clases detalladas.

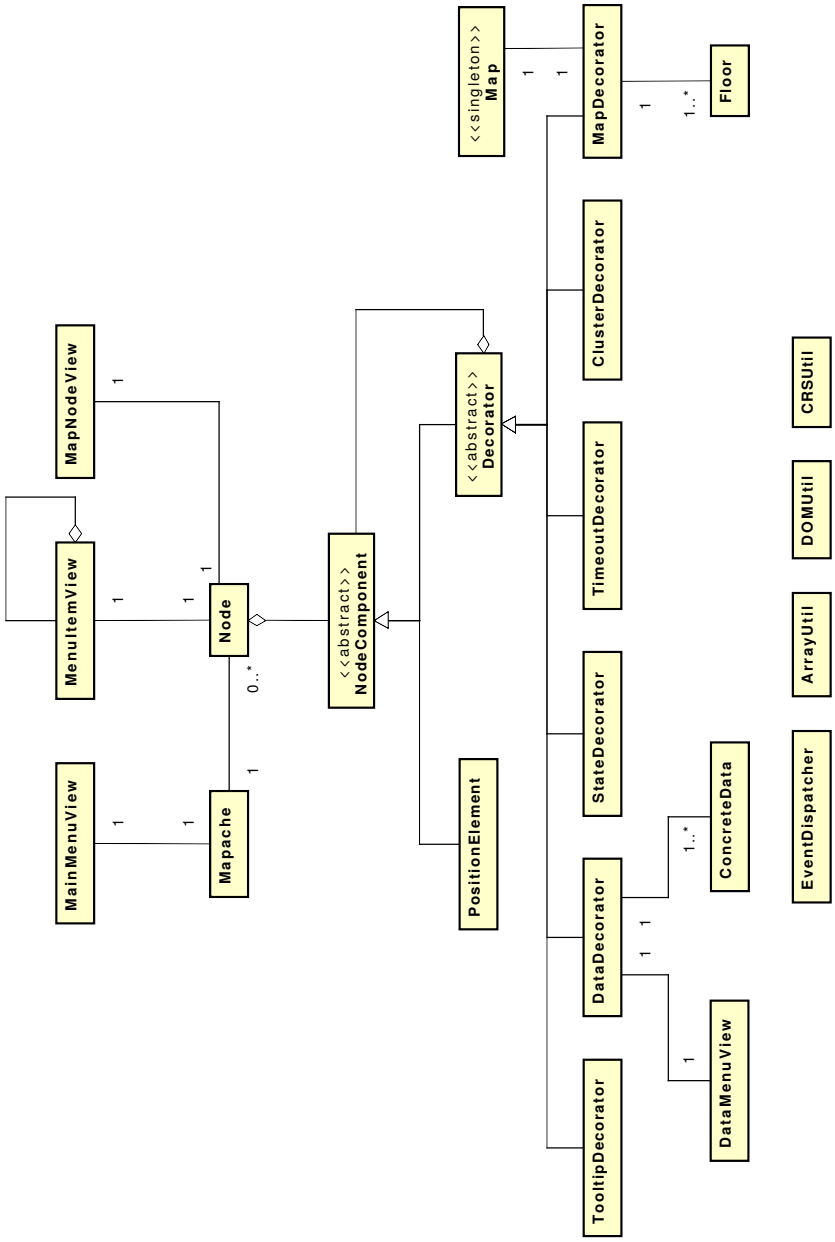


Figura 6.2: Diagrama General de Clases de Diseño

6.3.1. Clases Detalladas: Store

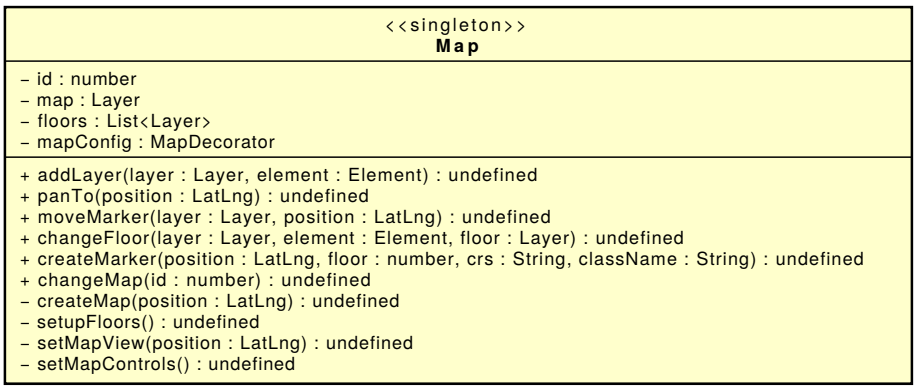


Figura 6.3: Clases detalladas: Map

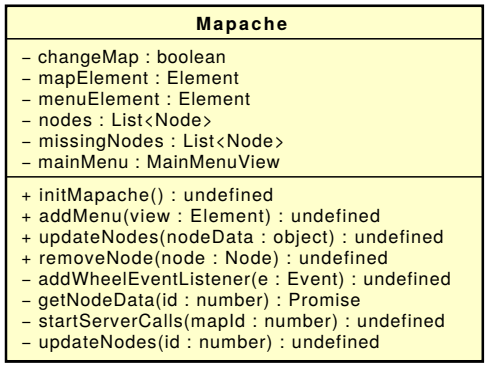


Figura 6.4: Clases detalladas: Mapache

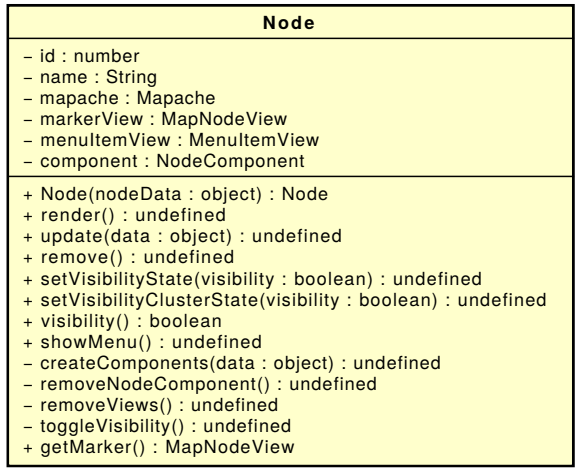


Figura 6.5: Clases detalladas: Node

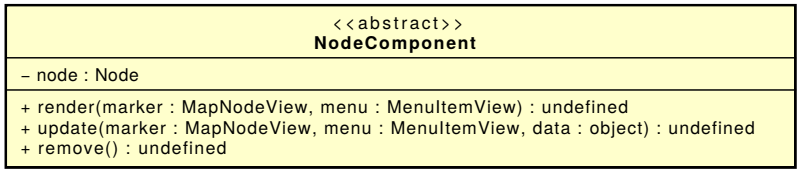


Figura 6.6: Clases detalladas: Node Component

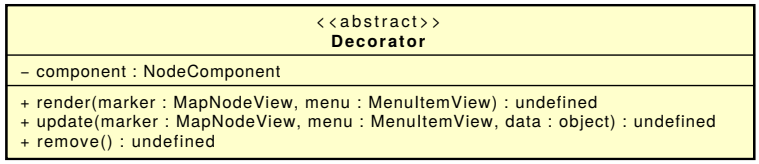


Figura 6.7: Clases detalladas: Decorator

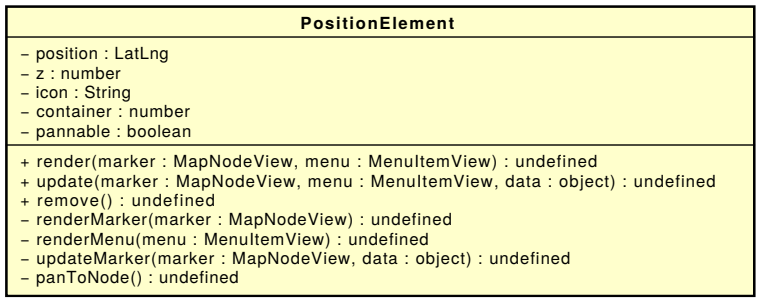


Figura 6.8: Clases detalladas: Position Element

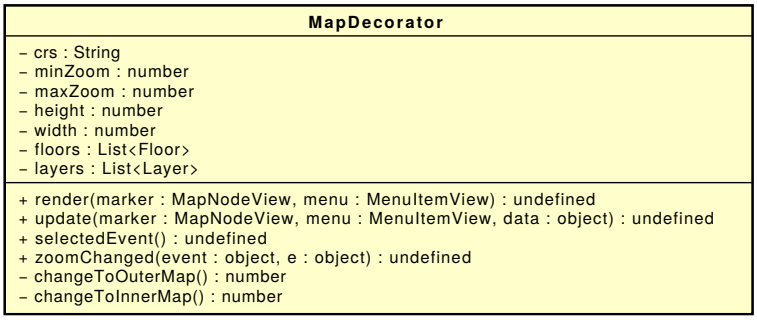


Figura 6.9: Clases detalladas: Map Decorator

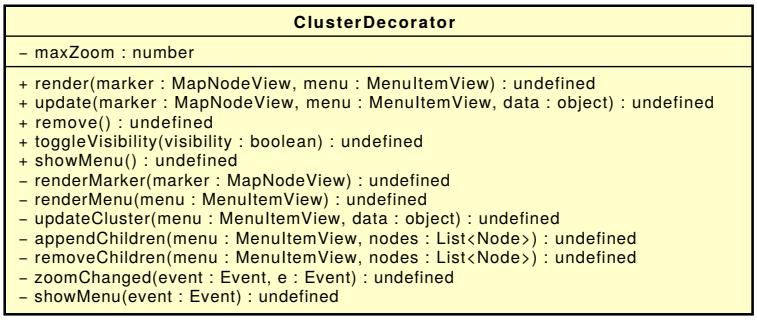


Figura 6.10: Clases detalladas: Cluster Decorator

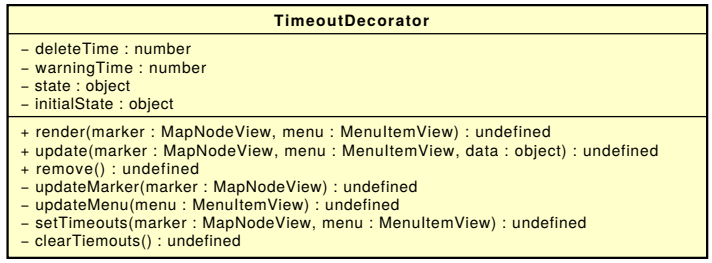


Figura 6.11: Clases detalladas: Timeout Decorator

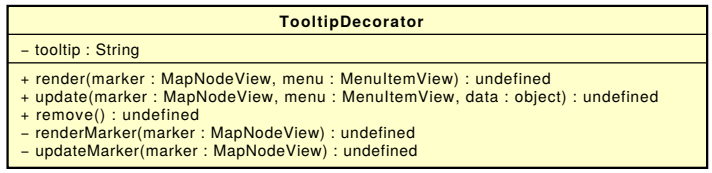


Figura 6.12: Clases detalladas: Tooltip Decorator

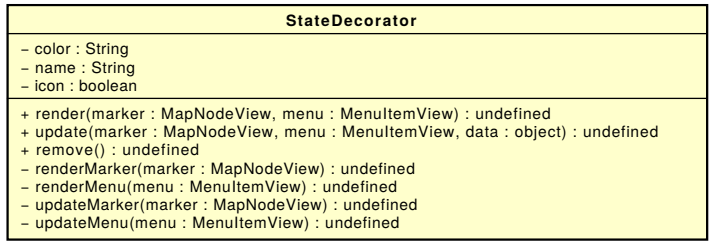


Figura 6.13: Clases detalladas: State Decorator

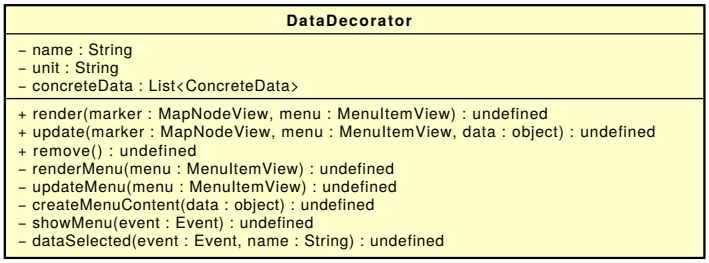


Figura 6.14: Clases detalladas: Data Decorator

Figura 6.15: Clases detalladas: Floor y Concrete Data

6.3.2. Clases Detalladas: Event Dispatcher

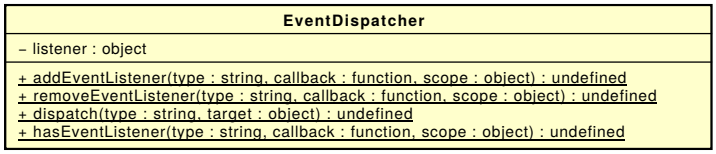


Figura 6.16: Clases detalladas: Event Dispatcher

6.3.3. Clases Detalladas: Views

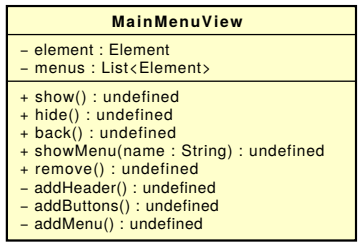


Figura 6.17: Clases detalladas: Main Menu View

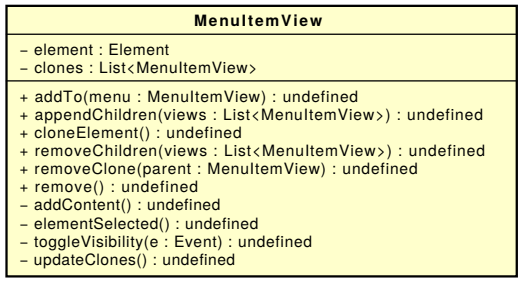


Figura 6.18: Clases detalladas: Menu Item View

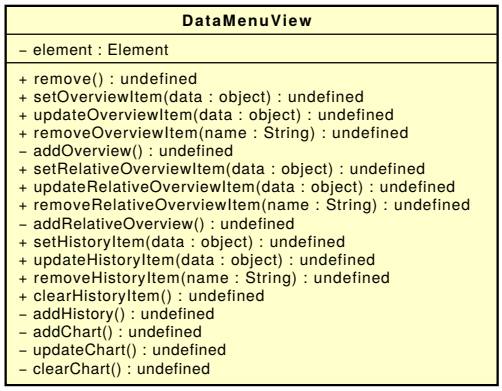


Figura 6.19: Clases detalladas: Data Menu View

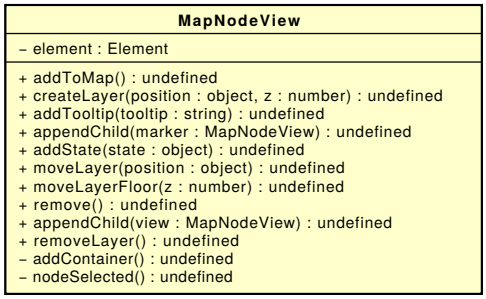


Figura 6.20: Clases detalladas: Map Node View

6.3.4. Clases Detalladas: Utils

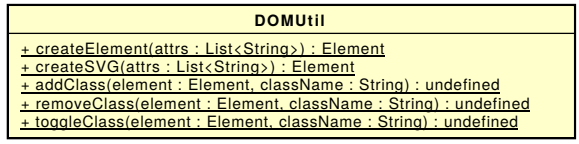


Figura 6.21: Clases detalladas: DOM utils

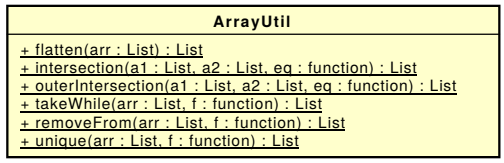


Figura 6.22: Clases detalladas: Array Utils

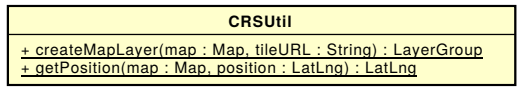


Figura 6.23: Clases detalladas: CRS Utils

6.4. Relaciones de Dependencia Lógica del Dominio

Una vez definida la capa de la lógica del dominio y el diagrama de clases de diseño, se procede a realizar el diagrama de dependencias de la lógica del dominio con las demás capas de *Mapache*. Se puede ver el diagrama en la Figura 6.24.

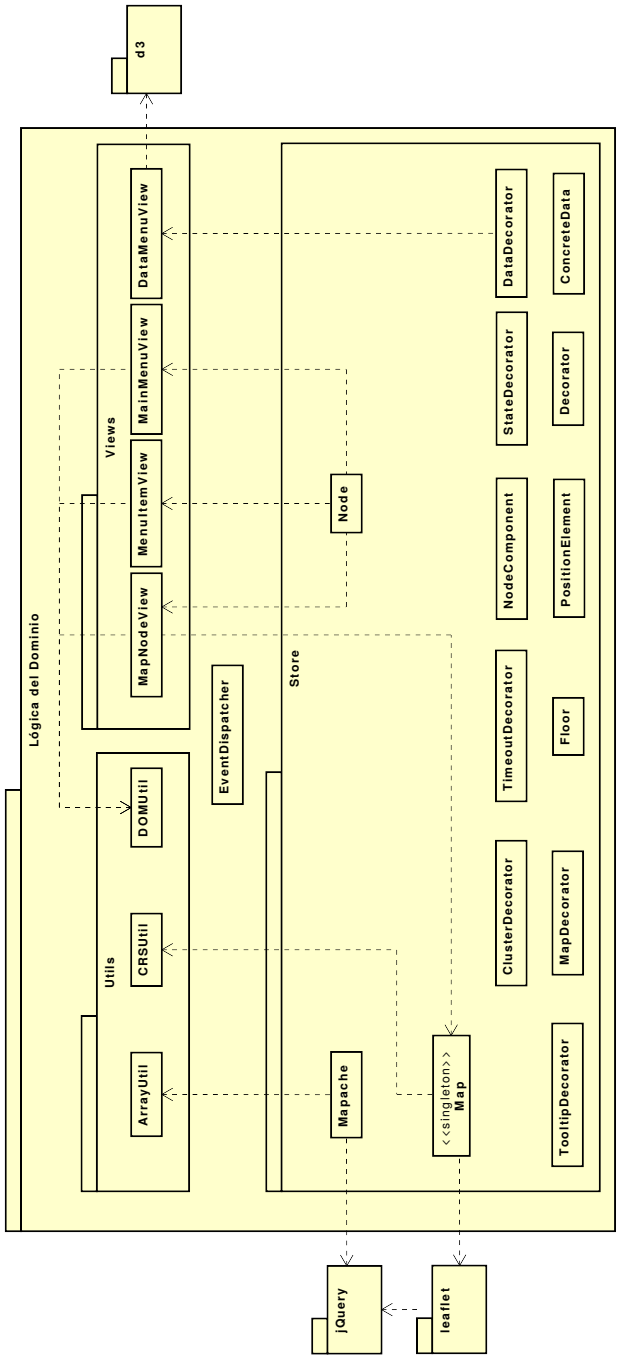


Figura 6.24: Relaciones de dependencia de la lógica del dominio

6.5. Diagramas de Secuencia de Diseño

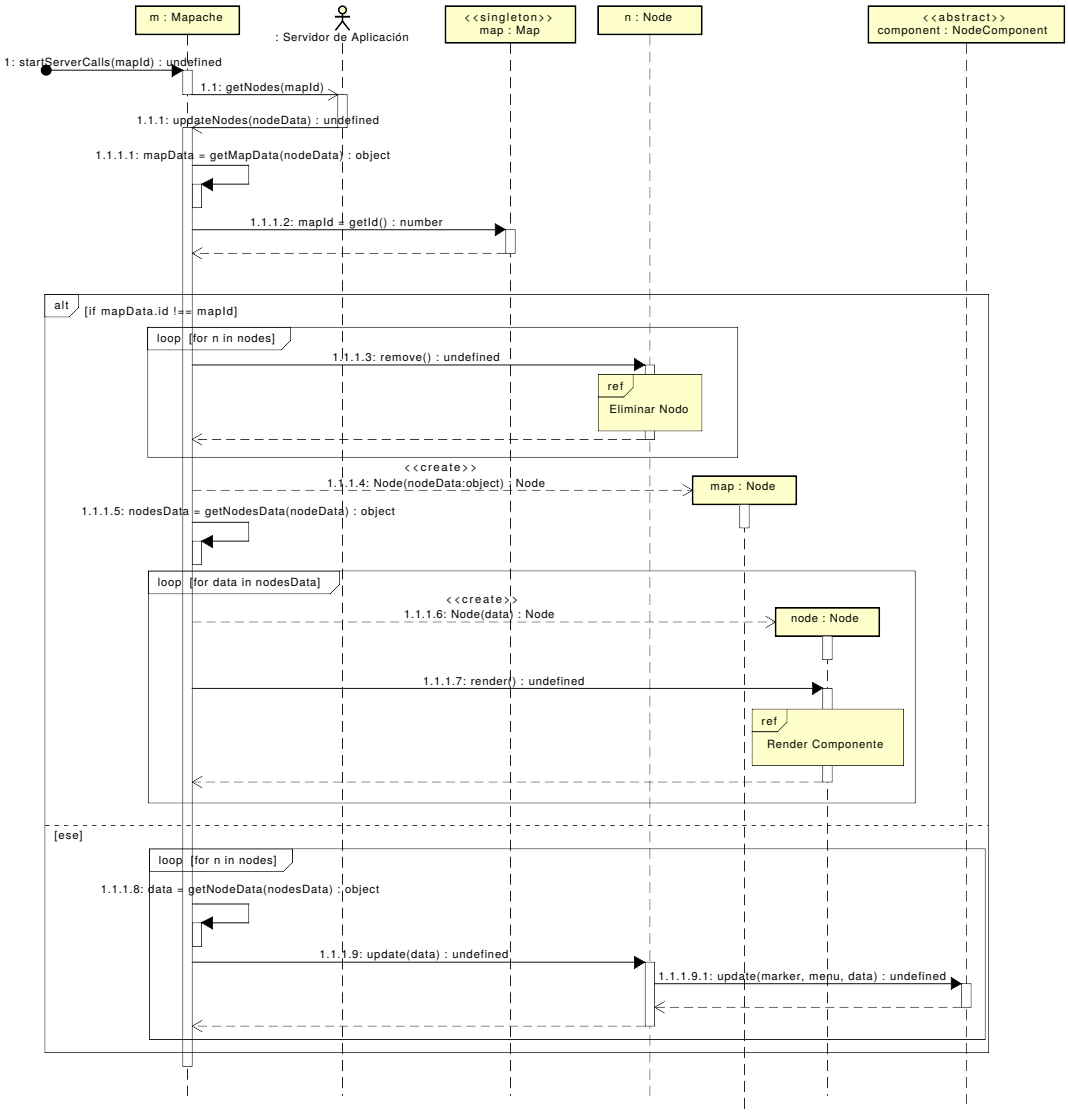


Figura 6.25: Diagrama de secuencia Actualizar Nodos

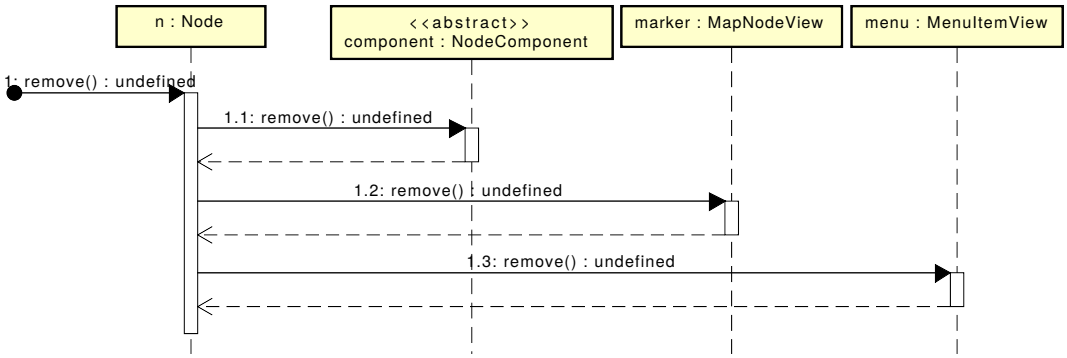


Figura 6.26: Diagrama de secuencia Eliminar Nodo

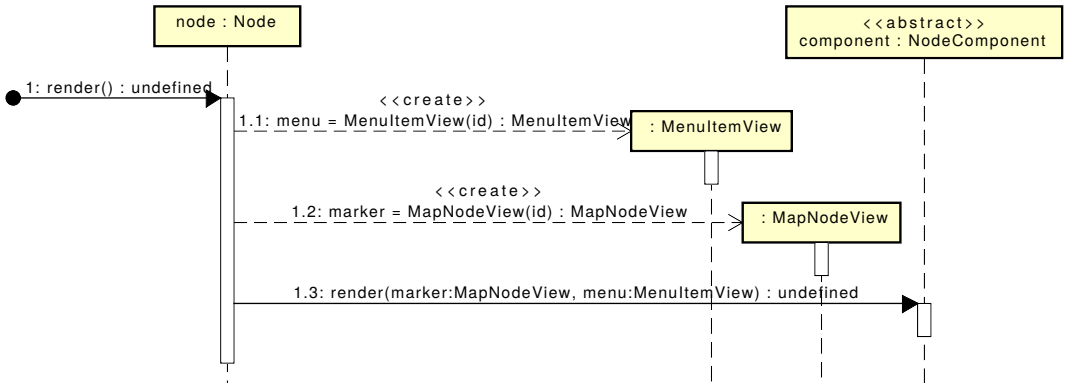


Figura 6.27: Diagrama de secuencia Render Componente

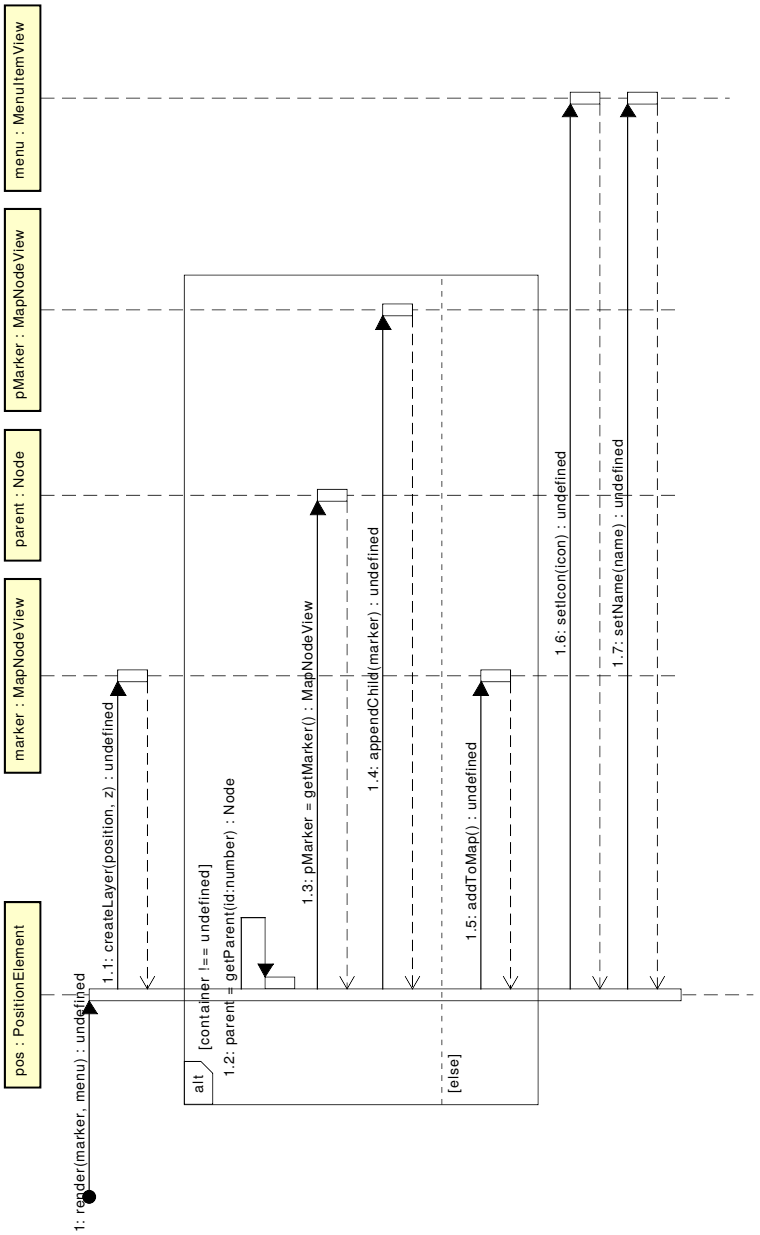


Figura 6.28: Diagrama de secuencia Render Componente Posición

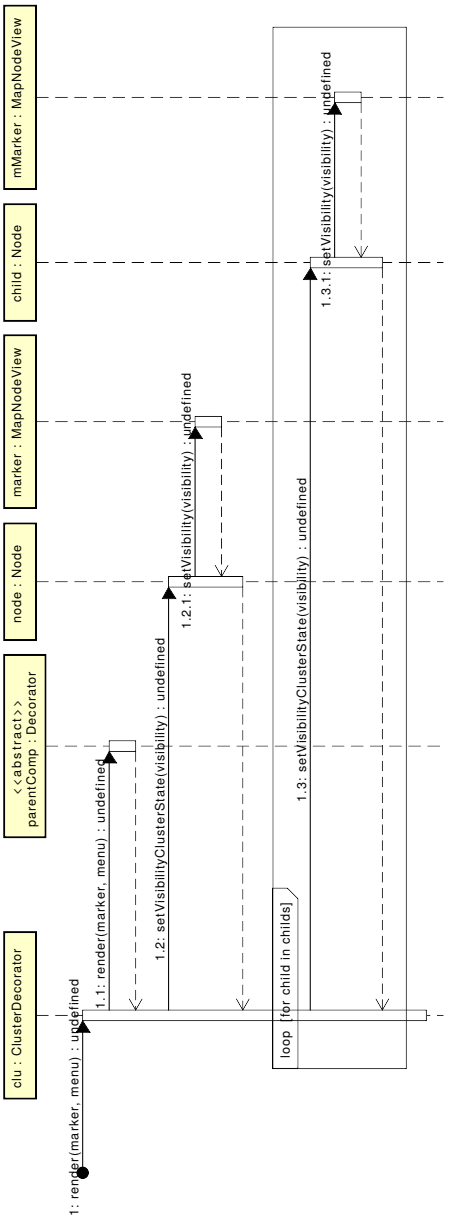


Figura 6.29: Diagrama de secuencia Render Componente Cluster

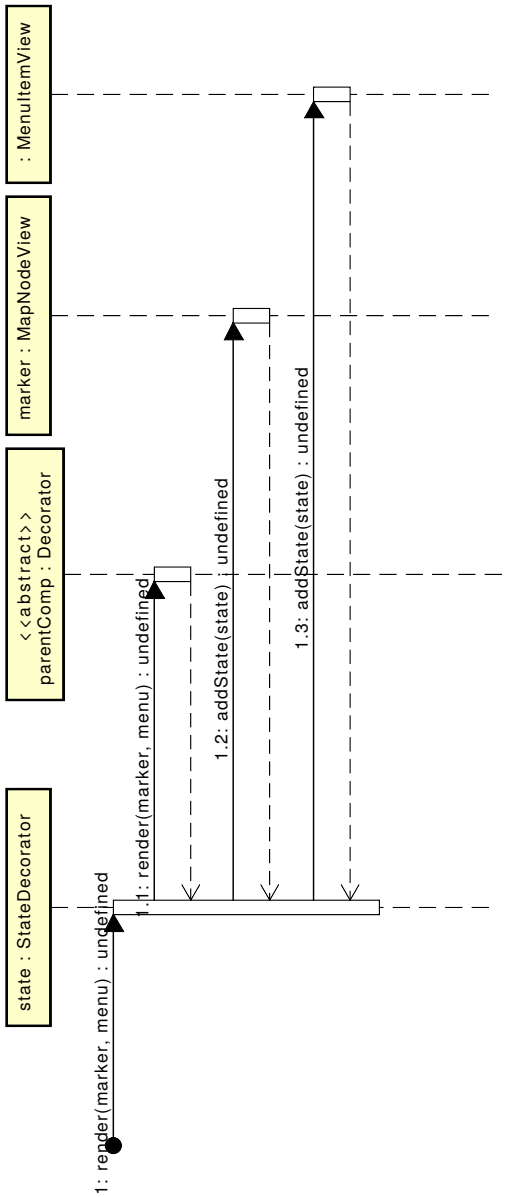


Figura 6.30: Diagrama de secuencia Render Componente State

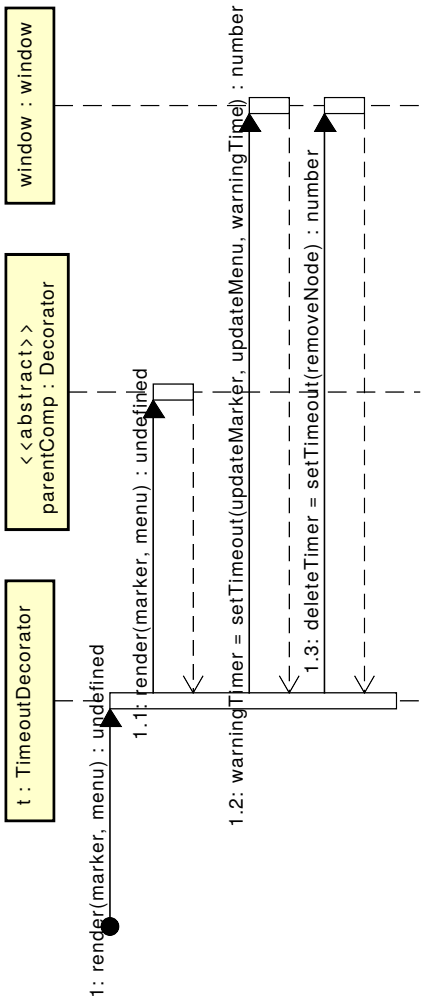


Figura 6.31: Diagrama de secuencia Render Componente Timeout

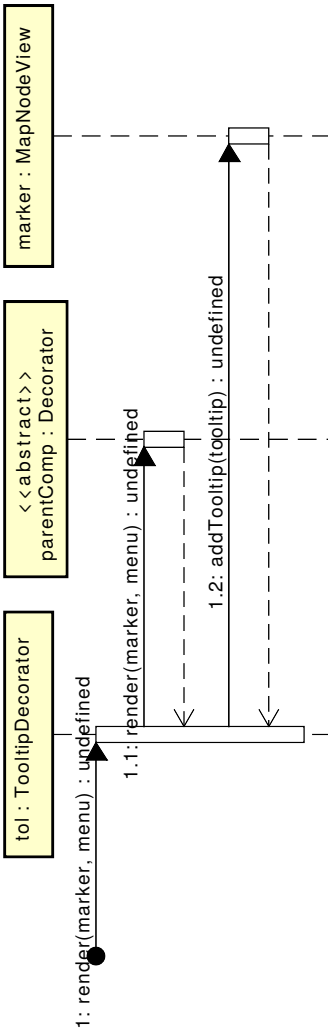


Figura 6.32: Diagrama de secuencia Render Componente Tooltip

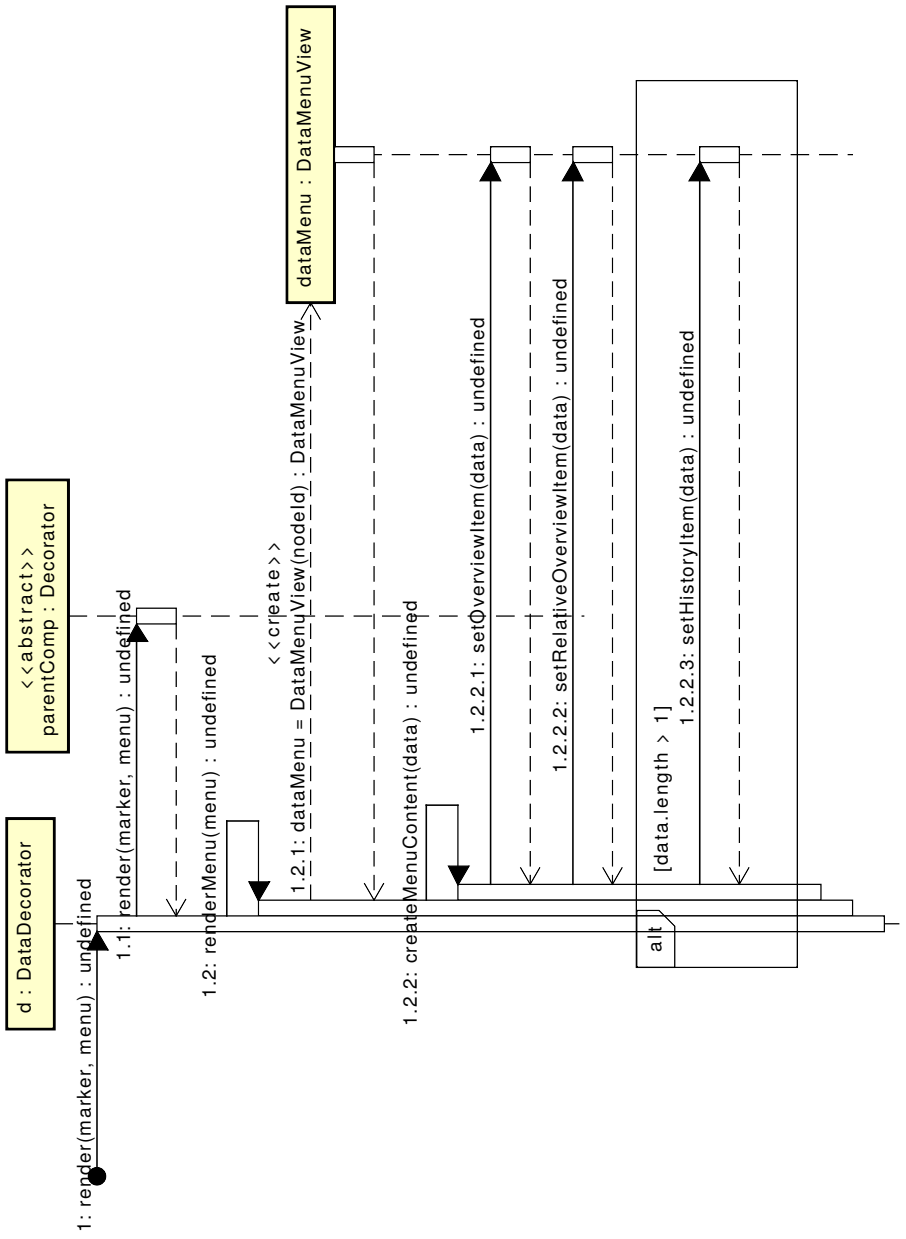


Figura 6.33: Diagrama de secuencia Render Componente Data

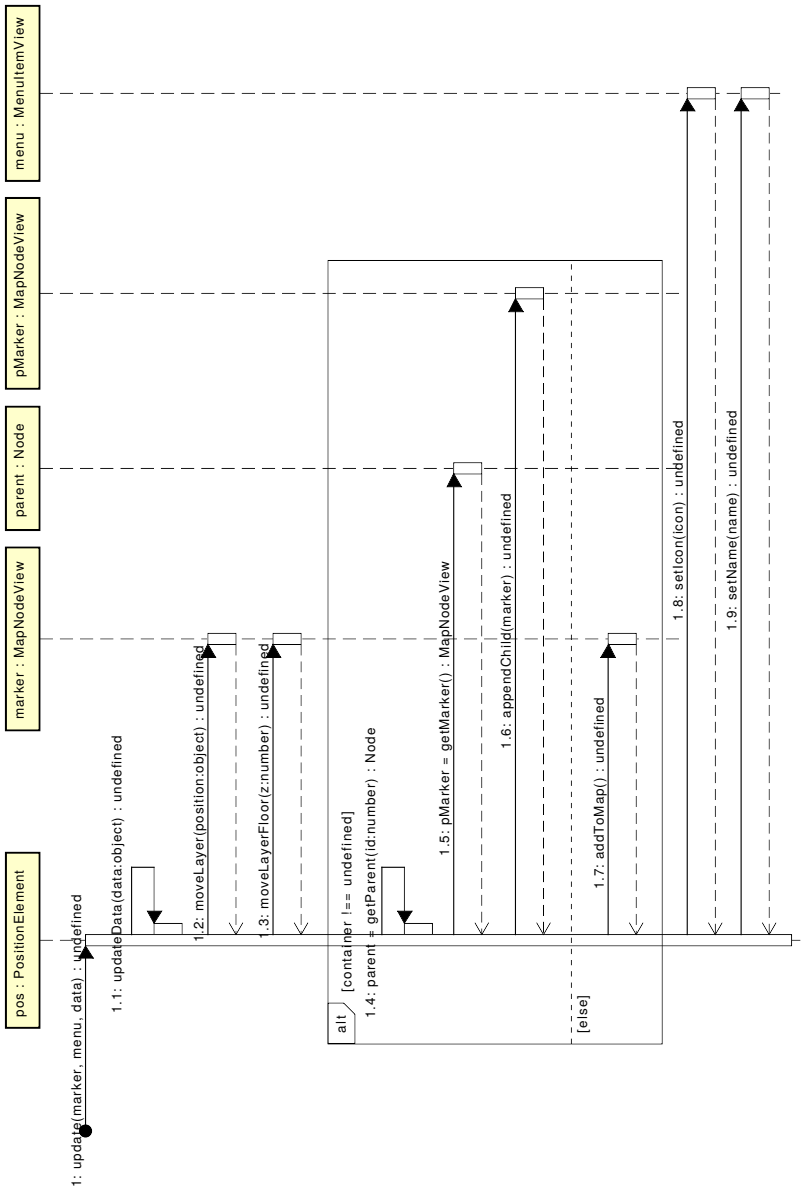


Figura 6.34: Diagrama de secuencia Actualizar Componente Posición

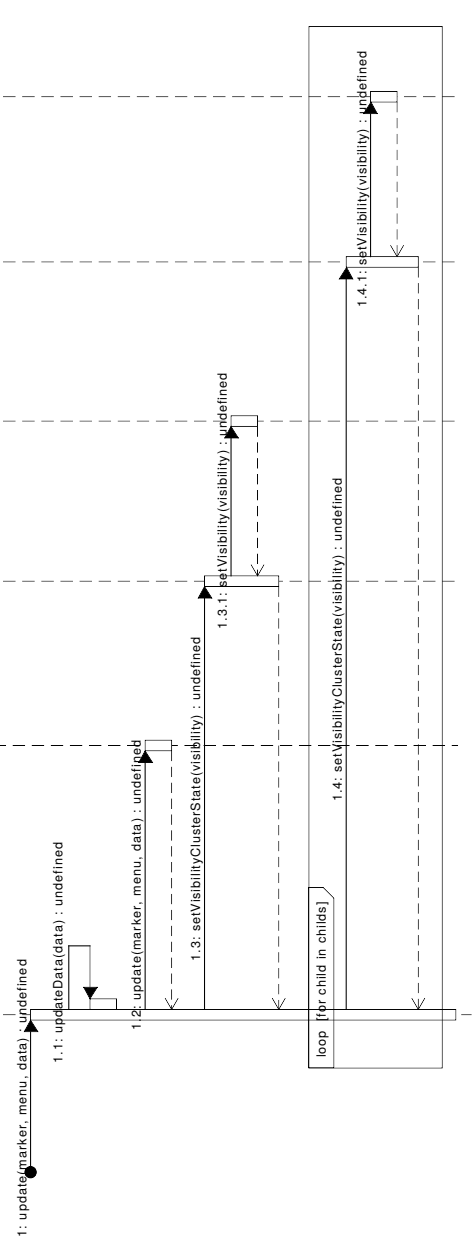


Figura 6.35: Diagrama de secuencia Actualizar Componente Cluster

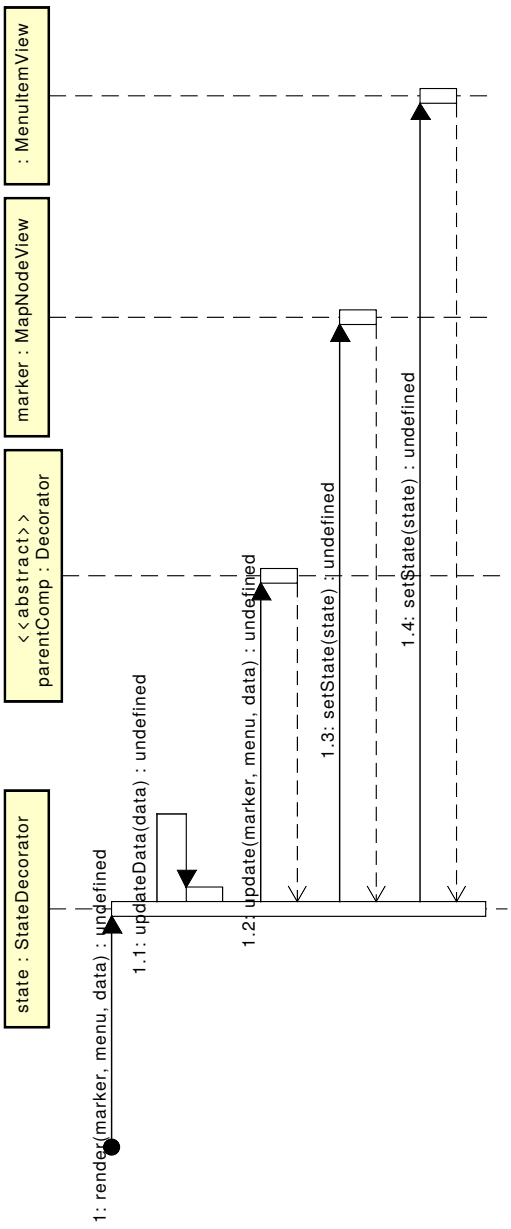


Figura 6.36: Diagrama de secuencia Actualizar Componente State

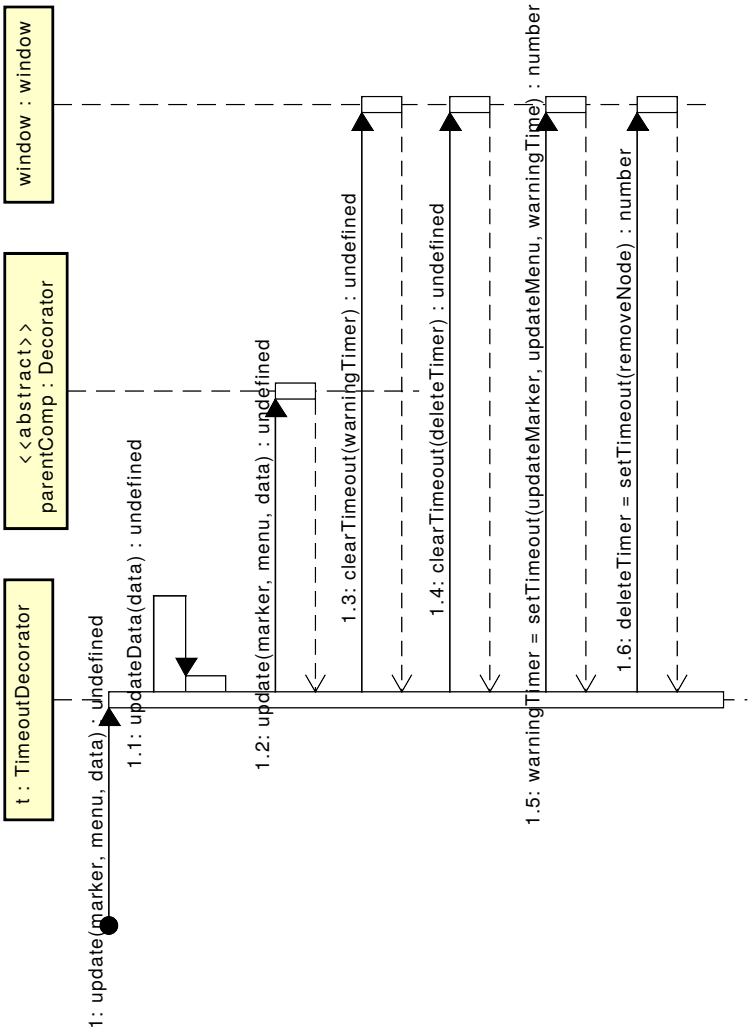


Figura 6.37: Diagrama de secuencia Actualizar Componente Timeout

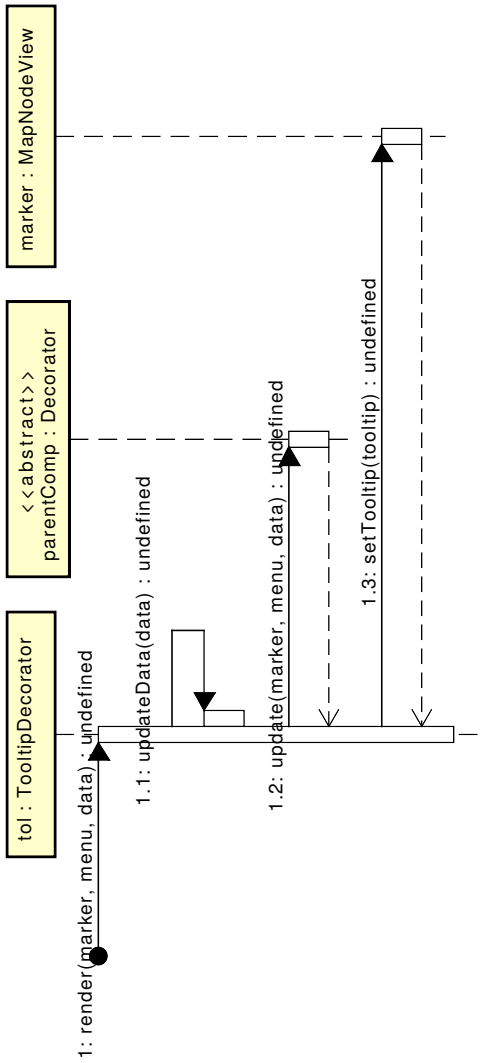


Figura 6.38: Diagrama de secuencia Actualizar Componente Tooltip

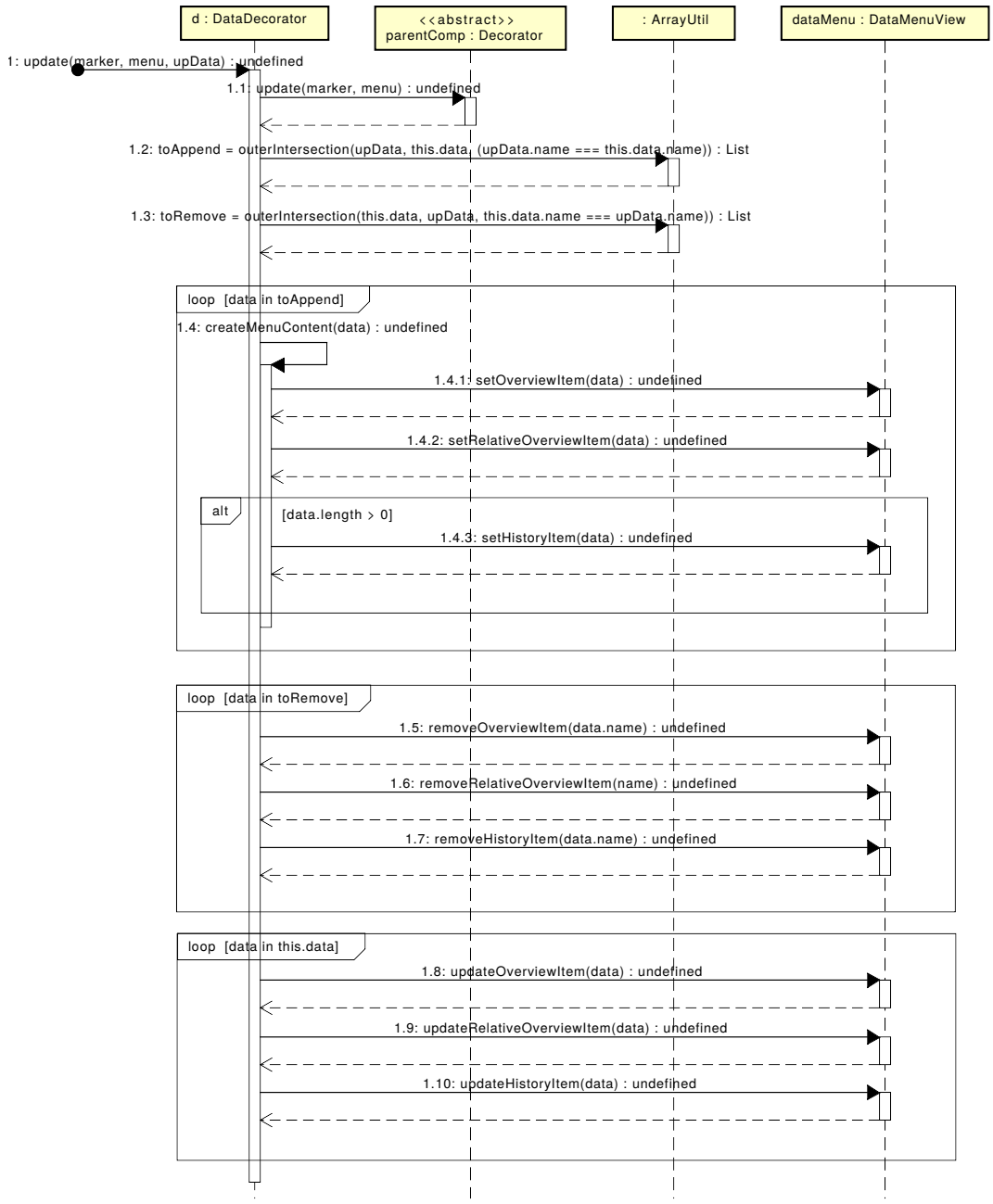


Figura 6.39: Diagrama de secuencia Update Componente Data

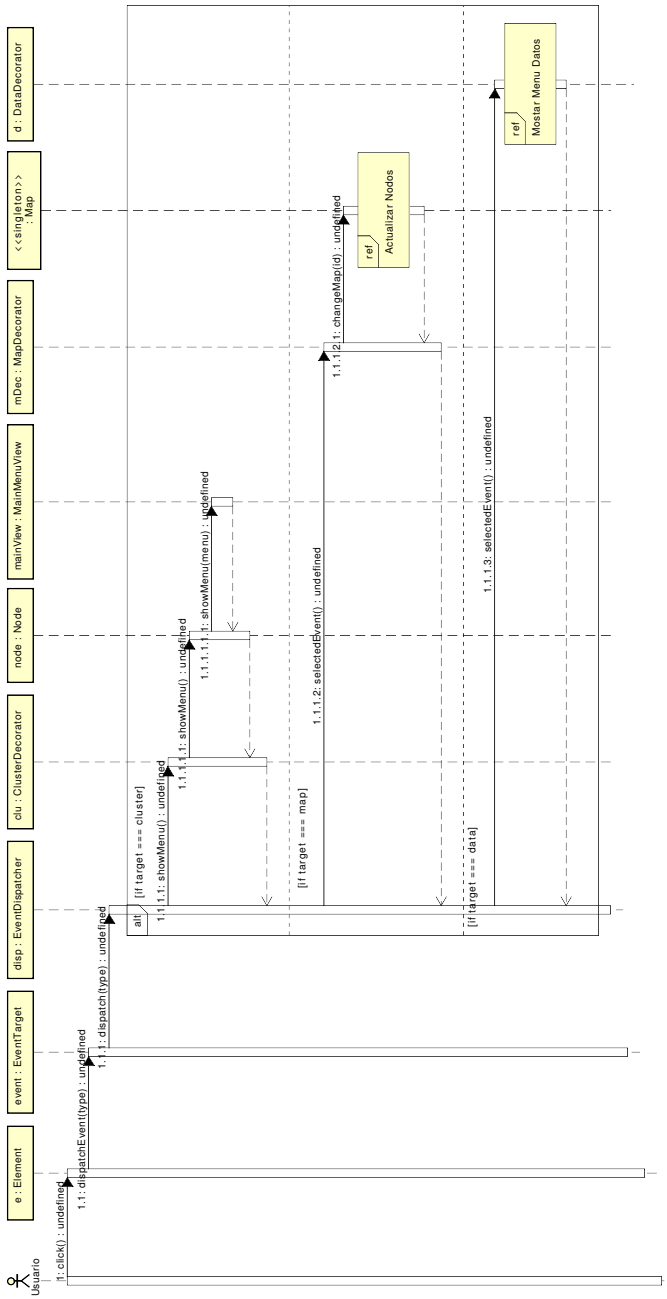


Figura 6.40: Diagrama de secuencia Seleccionar Nodo

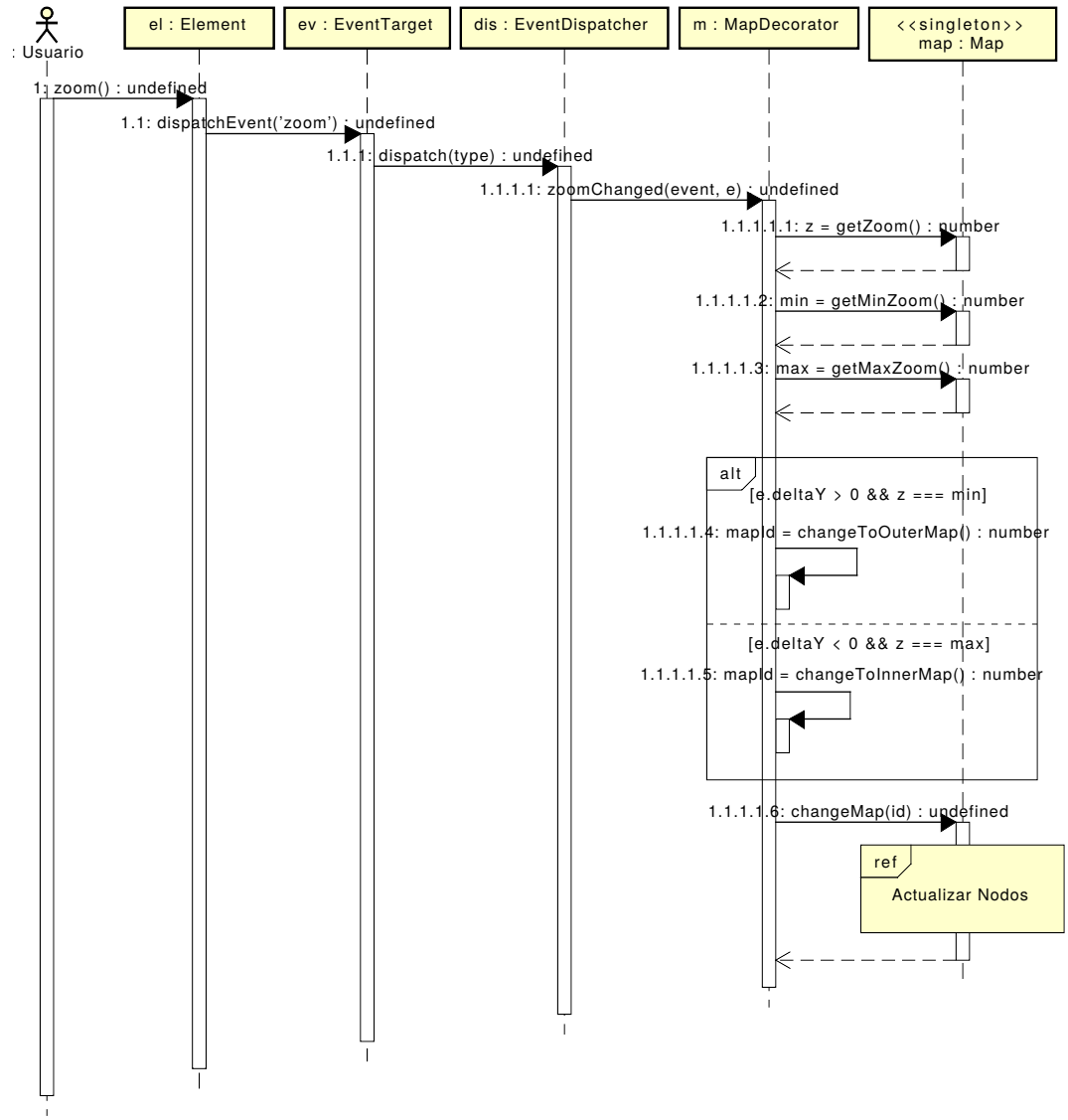


Figura 6.41: Diagrama de secuencia Modificar Zoom

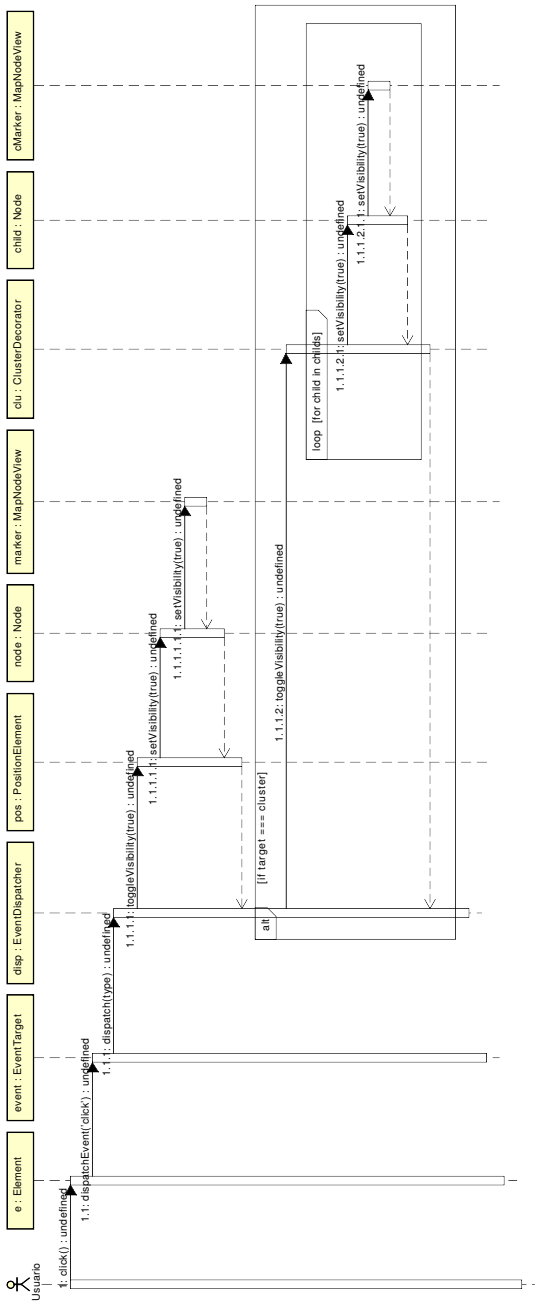


Figura 6.42: Diagrama de secuencia Mostrar Nodo

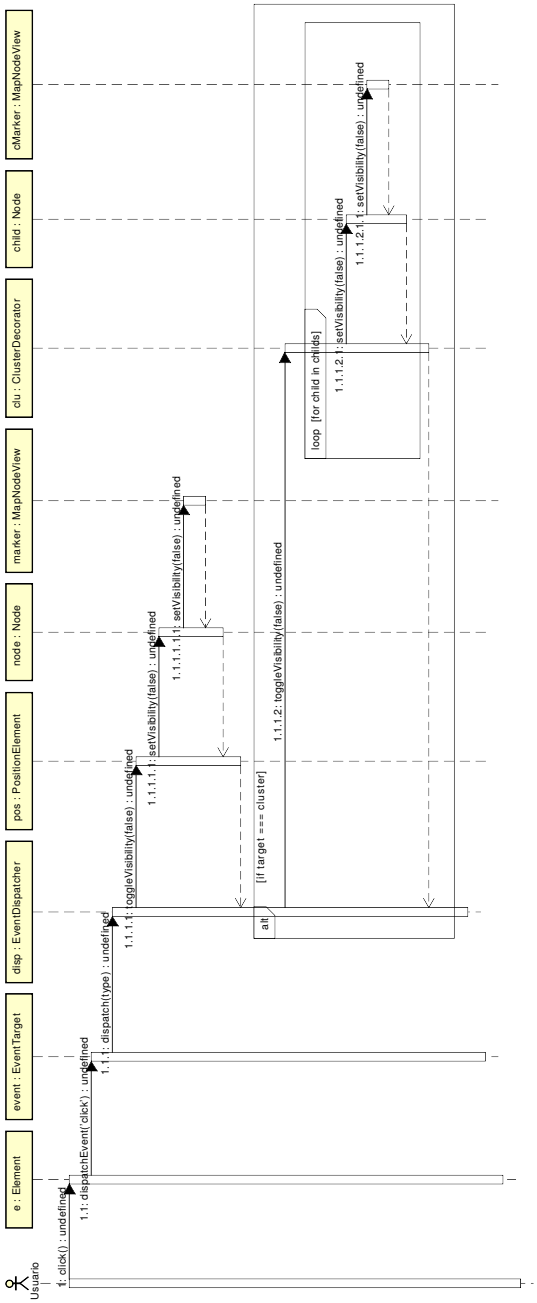


Figura 6.43: Diagrama de secuencia Ocultar Nodo

6.6. Análisis de Patrones de Diseño

A continuación se describen los patrones de diseño [5] que han sido utilizados en la fase de modelado.

6.6.1. Singleton

Mapache necesita tener almacenada información sobre el mapa y toda la lógica de comunicación con Leaflet en un objeto. Además, la información que este almacena debe de ser accesible desde cualquier objeto del dominio.

Este problema se solventa con el patrón *Singleton* proporcionando una instancia y sólo una de un objeto que tiene un punto de acceso conocido por todos los demás objetos. Este patrón y su uso en *Mapache* se puede ver en la Figura 6.44.

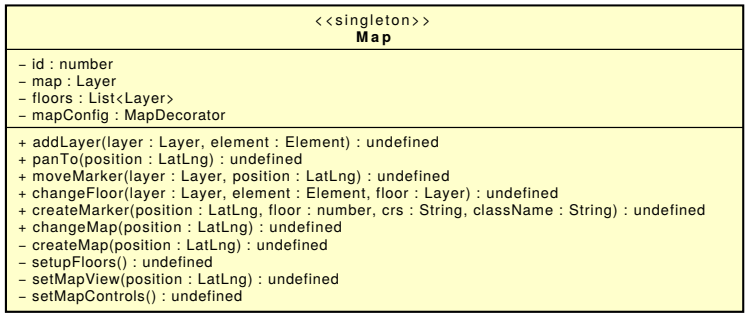


Figura 6.44: Patrón Singleton

Participantes

Singleton (*Map*) En el caso de JavaScript no es necesario declarar un método que devuelva la instancia del objeto, tan solo hay que exportar del módulo la instancia en vez de la clase, y cada vez que se accede a ella, es la misma instancia.

6.6.2. Observer

JavaScript trae por defecto un manejador de eventos del HTML DOM, pero no permite añadir eventos personalizados a este manejador. Es por ello que es necesario implementar un manejador de eventos propio de *Mapache* utilizando el patrón observador.

Este patrón ofrece la posibilidad de crear un modelo de subscripción y notificación de eventos. Los objetos suscritos son notificados cuando el evento ocurre. Este comportamiento se ve reflejado en la figura 6.45.

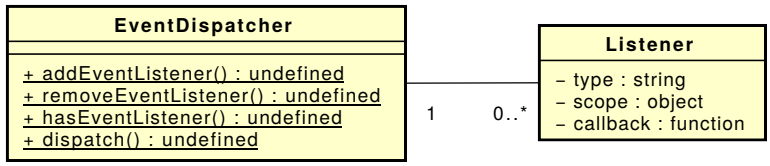


Figura 6.45: Patrón Observer

Participantes

Sujeto (*EventDispatcher*) Mantiene la lista de Observadores e implementa una interfaz que permite a los observadores subscribirse o darse de baja a un evento.

Observador (*Listener*) Implementa la función que ha de llamarse cuando el evento ocurre.

6.6.3. Decorator

El patrón Decorator resuelve el problema de añadir dinámicamente funcionalidad a un objeto.

En *Mapache*, el componente básico es el nodo. El Decorator permite dar funcionalidad extra a estos nodos en función de que representen, por ejemplo, si un nodo representa un cluster, tendrá la responsabilidad de renderizar a los nodos dentro de ese cluster cuando el nivel de zoom sea suficientemente grande. Este comportamiento se representa en la figura 6.46.

Participantes

Componente (*NodeComponent*) Define los métodos de los objetos a los que se les puede añadir funcionalidad extra.

Componente Concreto (*PositionElement*) Define el objeto al que se le puede añadir funcionalidad extra.

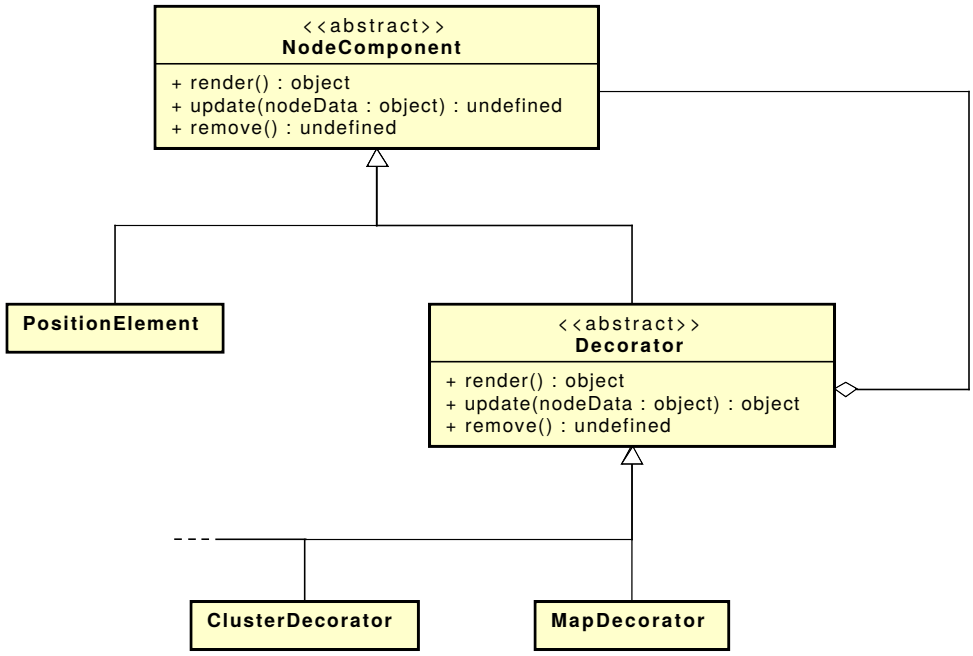


Figura 6.46: Patrón Decorator

Decorador (*Decorator*) Mantiene la referencia al componente y define la interfaz que usan los componentes.

Decorador Particular (*MapComponent...*) Implementa la funcionalidad de su responsabilidad.

6.7. Modelo de Comunicación con el Servidor

A la hora de implementar *Mapache* en un aplicación web, es necesario que esta aplicación proporcione una API que sirva a *Mapache* de herramienta para obtener la información que quiere mostrarse en la aplicación. Esta API debe de implementar un método que devuelva la información de los nodos en formato JSON con el siguiente formato:

Tabla 6.1: Modelo implementación API servidor: Main

Atributo	**Tipo**	**Opcional**	**Descripción**
map	Node	No	Nodo mapa que hay que renderizar
nodes	Node [ ]	No	Nodos que pertenecen al mapa

Tabla 6.2: Modelo implementación API servidor: Nodo

Atributo	**Tipo**	**Opcional**	**Descripción**
id	string	No	Identificador único del nodo
name	string	No	Nombre del nodo, utilizado en el menú
position	Position	No	Posición del nodo en el mapa actualmente renderizado
timeout	Timeout	Sí	Da al nodo la funcionalidad de controlar su linea de vida
tooltip	string	Sí	Nombre identificativo para que se muestre justo bajo el marcador
map	Map	Sí	Da al nodo la funcionalidad de mapa
cluster	Cluster	Sí	Da al nodo la funcionalidad de cluster
state	State	Sí	Da al nodo la funcionalidad de tener un estado
data	Data [ ]	Sí	Da al nodo la funcionalidad de mostrar información detallada sobre sí mismo

Tabla 6.3: Modelo implementación API servidor: Posición

Atributo	**Tipo**	**Opcional**	**Descripción**
x	number	No	Coordenada X de la posición
y	number	No	Coordenada Y de la posición
z	number	No	Coordenada Z de la posición, se corresponde con la planta de un mapa interior, para el mapa del mundo su valor es cero
container	number	Sí	Identificador del Nodo en el que se tiene que mostrar contenido este Nodo

Tabla 6.4: Modelo implementación API servidor: Timeout

Atributo	**Tipo**	**Opcional**	**Descripción**
delete Time	number	No	Tiempo transcurrido sin recibir información del servidor para eliminar el nodo del mapa
warning Time	number	Sí	Tiempo transcurrido en segundos sin recibir información del servidor para mostrar el estado de alerta
warning State	State	Sí	Estado que sobreescribe al estado del nodo cuando se sobrepasa el warningTime

Tabla 6.5: Modelo implementación API servidor: Map

Atributo	Tipo	Opcional	Descripción
crs	string	No	Sistema de referencia del mapa actual. **Valores:** <'indoor', 'outdoor'>
maxZoom	number	No	Máximo zoom que permite el mapa
minZoom	number	No	Mínimo zoom que permite el mapa
width	number	Sólo indoor	Anchura en pixels de la imagen del mapa interior
height	number	Sólo indoor	Altura en pixels de la imagen del mapa interior
floors	Floor [ ]	No	Las plantas del mapa

Tabla 6.6: Modelo implementación API servidor: Floor

Atributo	Tipo	Opcional	Descripción
main	boolean	Sí	Indica si la planta es la primera que se tiene que renderizar en el mapa tras ser cargado
name	string	No	Nombre de la planta, se usa para identificar la planta en los controles de cambio de planta del mapa
tileURL	string	No	Dirección URL de la que se puede obtener los tiles del mapa

Tabla 6.7: Modelo implementación API servidor: Cluster

Atributo	Tipo	Opcional	Descripción
maxZoom	number	No	Nivel de zoom de mapa en el cual se deja de renderizar el nodo cluster y se renderizan sus hijos
children	string [ ]	Sí	Lista con los identificadores de los nodos que pertenecen a este cluster

Tabla 6.8: Modelo implementación API servidor: State

Atributo	Tipo	Opcional	Descripción
name	string	No	Nombre del estado para mostrar en el menú
color	string	No	Color del icono cuando el estado esta activo
icon	boolean	No	Indica si el estado debe mostrarse en un icono o en el marcador

Tabla 6.9: Modelo implementación API servidor: Data

Atributo	**Tipo**	**Opcional**	**Descripción**
types	ConcreteData [ ]	No	Todos los datos diferentes de los que se quiere mostrar información

Tabla 6.10: Modelo implementación API servidor: ConcreteData

Atributo	**Tipo**	**Opcional**	**Descripción**
name	string	No	Nombre del dato a mostrar
unit	string	No	Nombre de la unidad de medida
values	Value [ ]	No	Historial con los datos a mostrar

Tabla 6.11: Modelo implementación API servidor: Value

Atributo	**Tipo**	**Opcional**	**Descripción**
value	string	No	Valor del dato
timestamp	number	No	Momento en el que se realizó la medida

Capítulo 7

Implementación y Pruebas

En este capítulo se presenta la solución obtenida presentada en los capítulos de análisis y diseño. Primero se detalla el modelo de implementación, y finalmente se muestran las pruebas realizadas para el correcto funcionamiento de *Mapache*.

7.1. Modelo de Implementación

El desarrollo de *Mapache* se llevó a cabo en dos iteraciones. La primera de ellas tuvo como objetivo la obtención de una versión inicial del framework en la que no había conexión entre la localización exterior e interior, se trataba de dos módulos independientes en el que existían dos elementos de localización, nodos y boundaries.

Tras un cambio en los requisitos de la aplicación, se llegó a la conclusión de que era necesario obtener un framework más general en el que sólo existiese un elemento de localización (el nodo) al que se le añadiese funcionalidad extra con la información suministrada por el servidor.

7.1.1. Iteración 1

En esta versión, *Mapache* es responsable de construir la jerarquía de los elementos de localización. Esta jerarquía no permitía más de un nivel de profundidad, las boundaries eran los elementos de mayor nivel y debajo de estas estaban los nodos.

A continuación se muestra el modelo de implementación de esta solución.

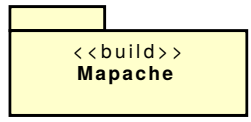


Figura 7.1: Iteración 1

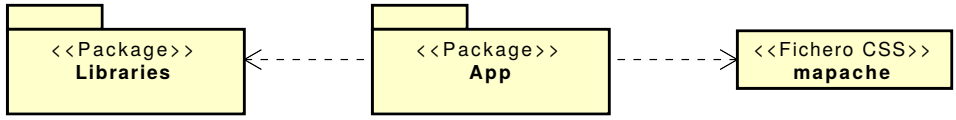


Figura 7.2: Iteración 1 - Paquete Mapache

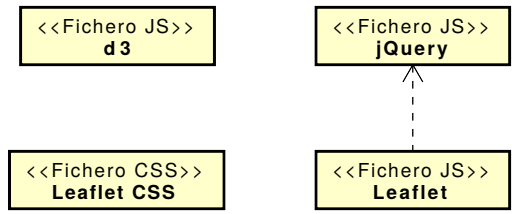


Figura 7.3: Iteración 1 - Paquete Libraries

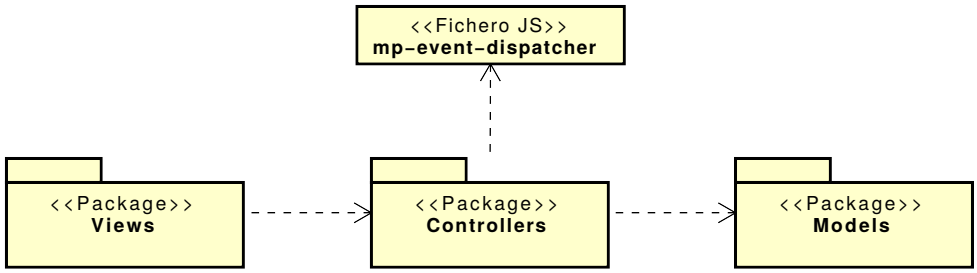


Figura 7.4: Iteración 1 - Paquete App

Figura 7.5: Iteración 1 - Paquete Models

Figura 7.6: Iteración 1 - Paquete Controllers

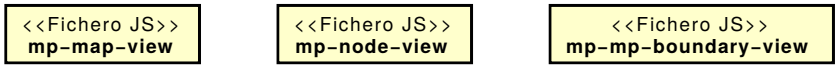


Figura 7.7: Iteración 1 - Paquete Views

7.1.2. Iteración 2

Tras un cambio en los requisitos, surgió la necesidad de poder integrar mapas interiores y exteriores en una misma aplicación, y poder tener más niveles de profundidad en la jerarquía de los elementos. La solución fue encapsular todos los elementos bajo Nodos, y poder añadir a estos funcionalidad extra dinámicamente. Para integrar los distintos mapas se optó por hacer que los propios mapas también fuesen nodos.

El modelo de implementación de esta solución es el siguiente.

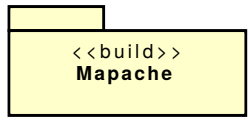


Figura 7.8: Iteración 2

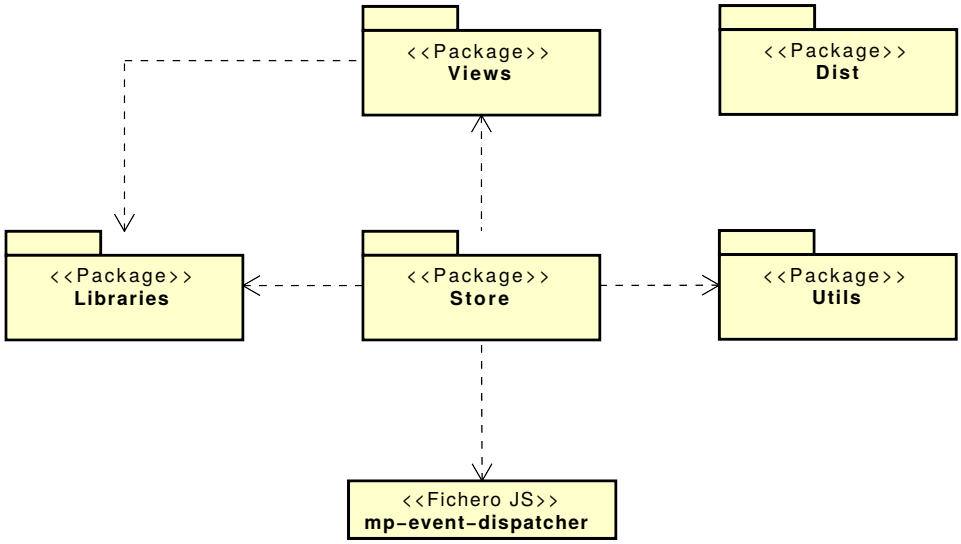


Figura 7.9: Iteración 2 - Paquete Mapache

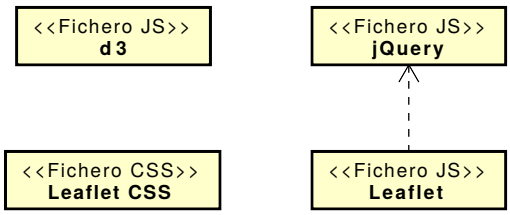


Figura 7.10: Iteración 2 - Paquete Libraries

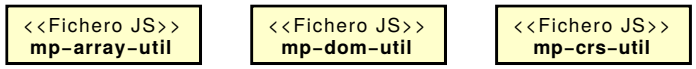


Figura 7.11: Iteración 2 - Paquete Utils

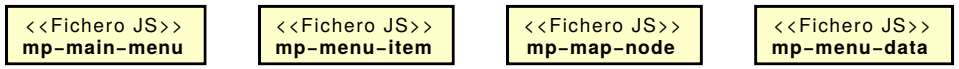


Figura 7.12: Iteración 2 - Paquete Views

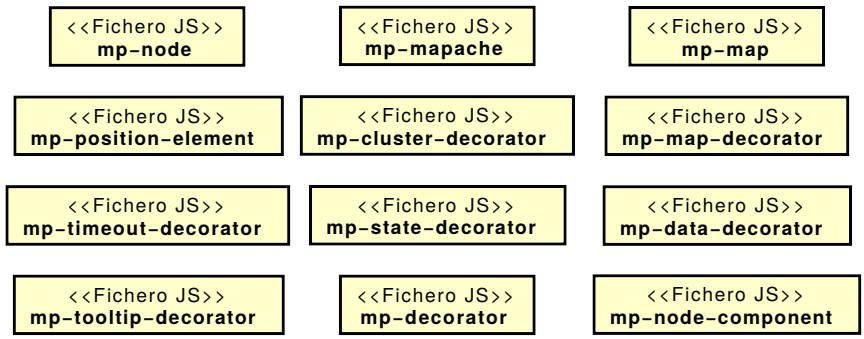


Figura 7.13: Iteración 2 - Paquete Store

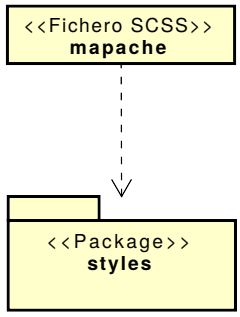


Figura 7.14: Iteración 2 - Paquete Dist

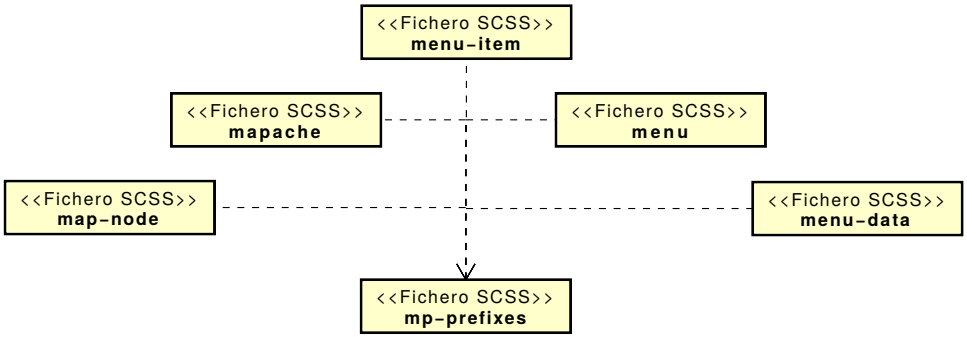


Figura 7.15: Iteración 2 - Paquete Styles

7.2. Pruebas

Para el apartado de pruebas se detallan las pruebas de caja negra, es decir, aquellas que dependen de las entradas y salidas del framework, sin tener en cuenta la implementación intermedia. Se ha decidido hacer estas pruebas de manera más exhaustiva debido a que al tratarse de un framework, es común que el desarrollador no tenga conocimiento del funcionamiento interno de este. Se han realizado dos tipos de pruebas, unas introduciendo como entradas JSON prefijados de los que se espera un comportamiento, y otras utilizando la aplicación en el servidor y manipulando directamente los dispositivos de localización y sensorización.

Con esta última aproximación, no solo se tiene en cuenta posibles errores en el framework, sino que también se pueden encontrar pruebas relacionadas con el fallo de la API del servidor, problemas con la estabilidad o velocidad de la red (vitales en un sistema en tiempo real). Además, con estas pruebas es más sencillo simular escenarios complejos.

Las pruebas de caja blanca, aunque no se detallan en la documentación, se han realizado a lo largo del proceso de desarrollo, parte de ellas antes de la implementación de las diferentes funcionalidades, y otra parte después debido a la naturaleza impredecible de los datos que recibe *Mapache* por parte de las APIs de los servidores.

Tabla 7.1: Pruebas: Inicialización de *Mapache*

Inicialización de *Mapache*	
Acción	Lanzar el script de incialización del framework pasando como argumento los elementos del DOM sobre los que se tiene que construir.
Salida Esperada	*Mapache* almacena los elementos del DOM en variables y lanza la primera petición al servidor.
Salida Obtenida	*Mapache* almacena los elementos del DOM en variables y lanza la primera petición al servidor.

Tabla 7.2: Pruebas: Inicialización de *Mapache* con parámetros erróneos

Inicialización de *Mapache* con parámetros erroneos	
Acción	Lanzar el script de inicialización del framework pasando como argumento los elementos que no forman parte del DOM o son nulos.
Salida Esperada	*Mapache* informa por la consola del navegador que alguno de los elementos no existe o es nulo y termina su ejecución.
Salida Obtenida	*Mapache* informa por la consola del navegador que alguno de los elementos no existe o es nulo y termina su ejecución.

Tabla 7.3: Pruebas: Creación del Mapa Exterior

Creación del Mapa Exterior	
Entrada	Un nodo con el componente mapa indicando el sistema de referencia outdoor.
Salida Esperada	Leaflet crea el mapa y lo introduce en el DOM.
Salida Obtenida	Leaflet crea el mapa y lo introduce en el DOM.

Tabla 7.4: Pruebas: Creación del Mapa Interior con una sola planta

Creación del Mapa Exterior con una sola planta	
Entrada	Un nodo con el componente mapa indicando el sistema de referencia indoor y que contiene una planta.
Salida Esperada	Leaflet crea el mapa y lo introduce en el DOM sin un controlador para el manejo de plantas.
Salida Obtenida	Leaflet crea el mapa y lo introduce en el DOM con un controlador para el manejo de plantas.
Observaciones	Leaflet crea por defecto un controlador de plantas al pasarle como argumento las capas de estas, la solución es crear el controlador con las herramientas proporcionadas por leaflet, de modo que si solo hay una planta, no se cree.

Tabla 7.5: Pruebas: Creación del Mapa Interior con dos plantas

Creación del Mapa Exterior con dos plantas	
Entrada	Un nodo con el componente mapa indicando el sistema de referencia indoor y que contiene dos plantas.
Salida Esperada	Leaflet crea el mapa y lo introduce en el DOM con un controlador para el manejo de plantas.
Salida Obtenida	Leaflet crea el mapa y lo introduce en el DOM con un controlador para el manejo de plantas.

Tabla 7.6: Pruebas: Creación del Mapa sin especificar maxZoom

Creación del Mapa sin especificar maxZoom	
Entrada	Un nodo con el componente mapa sin indicar el parámetro maxZoom.
Salida Esperada	*Mapache* informa de que en la creación del mapa falta el atributo maxZoom y finaliza la ejecución del framework.
Salida Obtenida	*Mapache* informa de que en la creación del mapa falta el atributo maxZoom y finaliza la ejecución del framework.

Tabla 7.7: Pruebas: Creación del Mapa sin especificar minZoom

Creación del Mapa sin especificar minZoom	
Entrada	Un nodo con el componente mapa sin indicar el parámetro minZoom.
Salida Esperada	*Mapache* informa de que en la creación del mapa falta el atributo minZoom y finaliza la ejecución del framework.
Salida Obtenida	*Mapache* informa de que en la creación del mapa falta el atributo minZoom y finaliza la ejecución del framework.

Tabla 7.8: Pruebas: Creación del Mapa Interior sin especificar width

Creación del Mapa Interior sin especificar width	
Entrada	Un nodo con el componente mapa especificando el sistema de referencia indoor y sin indicar el parámetro width.
Salida Esperada	*Mapache* informa de que en la creación del mapa falta el atributo width y finaliza la ejecución del framework.
Salida Obtenida	*Mapache* informa de que en la creación del mapa falta el atributo width y finaliza la ejecución del framework.

Tabla 7.9: Pruebas: Creación del Mapa Interior sin especificar height

Creación del Mapa Interior sin especificar height	
Entrada	Un nodo con el componente mapa especificando el sistema de referencia indoor y sin indicar el parámetro height.
Salida Esperada	*Mapache* informa de que en la creación del mapa falta el atributo height y finaliza la ejecución del framework.
Salida Obtenida	*Mapache* informa de que en la creación del mapa falta el atributo height y finaliza la ejecución del framework.

Tabla 7.10: Pruebas: Creación del Mapa Interior sin especificar plantas

Creación del Mapa Interior sin especificar plantas	
Entrada	Un nodo con el componente mapa con cero plantas.
Salida Esperada	*Mapache* informa de que en la creación del mapa faltan plantas para renderizar y finaliza la ejecución.
Salida Obtenida	*Mapache* informa de que en la creación del mapa faltan plantas para renderizar y finaliza la ejecución.

Tabla 7.11: Pruebas: Creación de un Nodo sin componentes

Creación de un Nodo sin componentes	
Entrada	El mapa principal y un nodo con id, nombre, y posición.
Salida Esperada	El nodo se muestra en el mapa en la posición especificada, y en el menú.
Salida Obtenida	El nodo se muestra en el mapa en la posición especificada, y en el menú.

Tabla 7.12: Pruebas: Creación de un Nodo sin identificador

Creación de un Nodo sin identificador	
Entrada	El mapa principal y un nodo sin identificador.
Salida Esperada	*Mapache* informa de que el nodo a crear necesita un identificador, no se muestra en el mapa ni en el menú.
Salida Obtenida	*Mapache* termina su ejecución.
Observaciones	No se estaba capturando correctamente la excepción, por lo que la ejecución de *Mapache* se terminaba. Capturando la excepción de manera adecuada arreglaba el problema.

Tabla 7.13: Pruebas: Creación de un Nodo sin nombre

Creación de un Nodo sin nombre	
Entrada	El mapa principal y un nodo sin nombre.
Salida Esperada	*Mapache* informa de que el nodo a crear necesita un nombre, no se muestra en el mapa ni en el menú.
Salida Obtenida	*Mapache* informa de que el nodo a crear necesita un nombre, no se muestra en el mapa ni en el menú.

Tabla 7.14: Pruebas: Creación de un Nodo sin componente posición

Creación de un Nodo sin componente posición	
Entrada	El mapa principal y un nodo sin posición.
Salida Esperada	*Mapache* informa de que el nodo a crear necesita una posición, no se muestra en el mapa ni en el menú.
Salida Obtenida	*Mapache* informa de que el nodo a crear necesita una posición, no se muestra en el mapa ni en el menú.

Tabla 7.15: Pruebas: Añadir posición a un nodo sin coordenada X

Añadir posición a un nodo sin coordenada X	
Entrada	El mapa principal y un nodo con posición sin coordenada X.
Salida Esperada	*Mapache* informa de que el componente posición necesita la coordenada X, no se muestra en el mapa ni en el menú.
Salida Obtenida	*Mapache* informa de que el componente posición necesita la coordenada X, no se muestra en el mapa ni en el menú.

Tabla 7.16: Pruebas: Añadir posición a un nodo sin coordenada Y

Añadir posición a un nodo sin coordenada Y	
Entrada	El mapa principal y un nodo con posición sin coordenada Y.
Salida Esperada	*Mapache* informa de que el componente posición necesita la coordenada Y, no se muestra en el mapa ni en el menú.
Salida Obtenida	*Mapache* informa de que el componente posición necesita la coordenada Y, no se muestra en el mapa ni en el menú.

Tabla 7.17: Pruebas: Añadir posición a un nodo sin coordenada Z

Añadir posición a un nodo sin coordenada Z	
Entrada	El mapa principal y un nodo con posición sin coordenada Z.
Salida Esperada	*Mapache* informa de que el componente posición necesita la coordenada Z, no se muestra en el mapa ni en el menú.
Salida Obtenida	*Mapache* informa de que el componente posición necesita la coordenada Z, no se muestra en el mapa ni en el menú.

Tabla 7.18: Pruebas: Añadir posición a un nodo coordenada Z errónea

Añadir posición a un nodo coordenada Z errónea	
Entrada	El mapa principal y un nodo con coordenada Z no existente en el mapa.
Salida Esperada	Leaflet informa de que la capa solicitada no se encuentra disponible, no se muestra en el mapa ni en el menú.
Salida Obtenida	Leaflet informa de que la capa solicitada no se encuentra disponible, no se muestra en el mapa ni en el menú.

Tabla 7.19: Pruebas: Añadir posición a un nodo bajo un contenedor

Añadir posición a un nodo bajo un contenedor	
Entrada	El mapa principal, un nodo que va a ser el contenedor, y un nodo con el atributo contenedor referenciando al primer nodo en el componente posición.
Salida Esperada	El nodo se muestra en el elemento del DOM contenedor del primer nodo.
Salida Obtenida	El nodo se muestra en el elemento del DOM contenedor del primer nodo, pero la posición del elemento está encima del nodo en Firefox.
Observaciones	El estilo flex de CSS no está estandarizado y por lo tanto la especificación en cada navegador no es la misma, el problema se encuentra en que el elemento contenedor no posee la propiedad de width y height, haciendo que el comportamiento en Firefox sea errático, la solución es añadir esas propiedades al contenedor del nodo.

Tabla 7.20: Pruebas: Añadir cluster a un nodo

Añadir cluster a un nodo	
Entrada	El mapa principal, un nodo que va a ser hijo del cluster, y un nodo con el componente cluster con el nodo anterior como hijo del cluster.
Salida Esperada	El nodo hijo se muestra en un desplegable bajo el cluster en el menú.
Salida Obtenida	El nodo hijo se muestra en un desplegable bajo el cluster en el menú.

Tabla 7.21: Pruebas: Añadir cluster a un nodo sin maxZoom

Añadir cluster a un nodo sin maxZoom	
Entrada	El mapa principal, y un nodo con el componente cluster sin el atributo maxZoom.
Salida Esperada	*Mapache* informa de que un cluster necesita el atributo maxZoom, el nodo no se muestra en el mapa ni en el menú.
Salida Obtenida	*Mapache* informa de que un cluster necesita el atributo maxZoom, el nodo no se muestra en el mapa ni en el menú.

Tabla 7.22: Pruebas: Añadir estado a un nodo

Añadir estado a un nodo	
Entrada	El mapa principal, y un nodo con el componente estado.
Salida Esperada	Se muestra el icono con el color del estado en el menú y en el mapa, también se muestra el nombre del estado en el menú.
Salida Obtenida	Se muestra el icono con el color del estado en el menú y en el mapa, también se muestra el nombre del estado en el menú.

Tabla 7.23: Pruebas: Añadir estado sin nombre a un nodo

Añadir estado sin nombre a un nodo	
Entrada	El mapa principal, y un nodo con el componente estado sin nombre.
Salida Esperada	*Mapache* informa que un componente estado necesita un nombre, el nodo no se muestra en el mapa ni en el menú.
Salida Obtenida	*Mapache* informa que un componente estado necesita un nombre, el nodo no se muestra en el mapa ni en el menú.

Tabla 7.24: Pruebas: Añadir estado sin color a un nodo

Añadir estado sin color a un nodo	
Entrada	El mapa principal, y un nodo con el componente estado sin color.
Salida Esperada	*Mapache* informa que un componente estado necesita un color, el nodo no se muestra en el mapa ni en el menú.
Salida Obtenida	*Mapache* informa que un componente estado necesita un color, el nodo no se muestra en el mapa ni en el menú.

Tabla 7.25: Pruebas: Añadir estado sin icono a un nodo

Añadir estado sin icono a un nodo	
Entrada	El mapa principal, y un nodo con el componente estado sin icono.
Salida Esperada	*Mapache* informa que un componente estado necesita un icono, el nodo no se muestra en el mapa ni en el menú.
Salida Obtenida	*Mapache* informa que un componente estado necesita un icono, el nodo no se muestra en el mapa ni en el menú.

Tabla 7.26: Pruebas: Añadir estado con icono "true" a un nodo

Añadir estado con icono "true" a un nodo	
Entrada	El mapa principal, y un nodo con el componente estado y el atributo icono a "true".
Salida Esperada	Se muestra el estado del nodo en un icono circular en la esquina inferior del marcador, tanto en el mapa como en el menú.
Salida Obtenida	No se muestra el icono con el estado.
Observaciones	Para añadir un SVG al DOM desde JavaScript es necesario crear el elemento con el espacio de nombres URI y un nombre cualificado mediante el método "createElementNS()", en el caso de los SVG la URI es "http://www.w3.org/2000/svg".

Tabla 7.27: Pruebas: Añadir estado con icono "false" a un nodo

Añadir estado con icono "false" a un nodo	
Entrada	El mapa principal, y un nodo con el componente estado y el atributo icono a "false".
Salida Esperada	Se muestra el estado del nodo cambiando el color del marcador tanto en el menú como en el mapa.
Salida Obtenida	Se muestra el estado del nodo cambiando el color del marcador tanto en el menú como en el mapa.

Tabla 7.28: Pruebas: Añadir timeout a un nodo

Añadir timeout a un nodo	
Entrada	El mapa principal, y un nodo con el componente timeout con un deleteTimeout de cinco segundos.
Salida Esperada	El nodo se elimina del mapa y del menú pasados cinco segundos sin recibir datos.
Salida Obtenida	El nodo se elimina del mapa y del menú pasados cinco segundos sin recibir datos.

Tabla 7.29: Pruebas: Añadir timeout con warningTime a un nodo

Añadir timeout con warningTime a un nodo	
Entrada	El mapa principal, y un nodo con el componente timeout con un warningTime de dos segundos y su estado asociado, y con un deleteTimeout de cinco segundos.
Salida Esperada	El nodo muestra el estado de warning pasados dos segundos sin recibir información, y se elimina pasado cinco segundos.
Salida Obtenida	El nodo muestra el estado de warning pasados dos segundos sin recibir información, y se elimina pasado cinco segundos.

Tabla 7.30: Pruebas: Añadir timeout a un nodo y actualizarlo cada segundo

Añadir timeout a un nodo y actualizarlo cada segundo	
Entrada	El mapa principal, y un nodo con el componente timeout con un deleteTimeout de cinco segundos.
Salida Esperada	En cada actualización el nodo restablece el timeout a cero, y por ello nunca se elimina.
Salida Obtenida	El nodo se elimina pasados cinco segundos.
Observaciones	No se estaba reiniciando el timeout correctamente, por lo que se elimina aunque recibiese información actualizada, para reiniciarlo hay que usar el método “clearInterval()”.

Tabla 7.31: Pruebas: Eliminar un nodo

Eliminar un nodo	
Entrada	El mapa principal, y un nodo con el componente timeout con un deleteTimeout de cero segundos.
Salida Esperada	El nodo se elimina del mapa y del menú.
Salida Obtenida	Se informa de una excepción de que el objeto nodo ya no existe y se intenta acceder al mismo.
Observaciones	Al eliminar el nodo con el timeout, la ejecución de la actualización de ese nodo seguía adelante por los demás componentes, y al intentar acceder al mismo, saltaba la excepción, la solución es detener la actualización del nodo antes de borrarlo.

Tabla 7.32: Pruebas: Añadir datos a un nodo

Añadir datos a un nodo	
Entrada	El mapa principal, y un nodo con el componente data.
Salida Esperada	Se crea un menú con los datos incluidos.
Salida Obtenida	Se crea un menú con los datos incluidos.

Tabla 7.33: Pruebas: Añadir datos sin nombre a un nodo

Añadir datos sin nombre a un nodo	
Entrada	El mapa principal, y un nodo con el componente data sin nombre del dato.
Salida Esperada	*Mapache* informa de que el componente data necesita datos con nombre.
Salida Obtenida	*Mapache* informa de que el componente data necesita datos con nombre.

Tabla 7.34: Pruebas: Añadir datos sin unidad a un nodo

Añadir datos sin unidad a un nodo	
Entrada	El mapa principal, y un nodo con el componente data sin la unidad del dato.
Salida Esperada	*Mapache* informa de que el componente data necesita datos con unidades.
Salida Obtenida	*Mapache* informa de que el componente data necesita datos con unidades.

Tabla 7.35: Pruebas: Añadir datos sin valores a un nodo

Añadir datos sin valores a un nodo	
Entrada	El mapa principal, y un nodo con el componente data sin valores de los datos.
Salida Esperada	*Mapache* informa de que el componente data necesita datos con valores.
Salida Obtenida	*Mapache* informa de que el componente data necesita datos con valores.

Tabla 7.36: Pruebas: Añadir datos con un solo valor a un nodo

Añadir datos con un solo valor a un nodo	
Entrada	El mapa principal, y un nodo con el componente data con un único valor para un dato.
Salida Esperada	Se crea un menú con los datos incluidos sin la gráfica con el historial de datos.
Salida Obtenida	Se crea un menú con los datos incluidos sin la gráfica con el historial de datos.

Tabla 7.37: Pruebas: Añadir datos con más de un valor a un nodo

Añadir datos con más de un valor a un nodo	
Entrada	El mapa principal, y un nodo con el componente data con más de un valor para un dato.
Salida Esperada	Se crea un menú con los datos incluidos con la gráfica del historial de datos.
Salida Obtenida	Se crea un menú con los datos incluidos con la gráfica del historial de datos.

Tabla 7.38: Pruebas: Seleccionar un mapa

Seleccionar un mapa	
Acción	Se selecciona un mapa en el mapa o en el menú.
Salida Esperada	Se eliminan los nodos presentes y se muestra el nuevo mapa con los nodos asociado a este.
Salida Obtenida	Se eliminan los nodos presentes y se muestra el nuevo mapa con los nodos asociado a este.

Tabla 7.39: Pruebas: Zoom máximo sobre un mapa

Zoom máximo sobre un mapa	
Acción	Se aumenta el zoom sobre un nodo mapa con nivel máximo de zoom.
Salida Esperada	Se eliminan los nodos presentes y se muestra el nuevo mapa con los nodos asociado a este.
Salida Obtenida	Se eliminan los nodos presentes y se muestra el nuevo mapa con los nodos asociado a este.

Tabla 7.40: Pruebas: Zoom mínimo sobre un mapa

Zoom mínimo sobre un mapa	
Acción	Se reduce el zoom sobre un nodo mapa indoor con nivel mínimo de zoom.
Salida Esperada	Se eliminan los nodos presentes y se muestra el mapa del mundo con sus nodos asociados.
Salida Obtenida	Se eliminan los nodos presentes y se muestra el mapa del mundo con sus nodos asociados.

Tabla 7.41: Pruebas: Desplazar el mapa seleccionando un nodo

Desplazar el mapa seleccionando un nodo	
Acción	Se selecciona un nodo en el mapa o en el menú.
Salida Esperada	El mapa se desplaza hasta que el viewport esté centrado en el nodo.
Salida Obtenida	El mapa se desplaza hasta que el viewport esté centrado en el nodo.

Tabla 7.42: Pruebas: Mostrar nodos de un cluster

Mostrar nodos de un cluster	
Acción	Se aumenta el zoom del mapa hasta alcanzar el máximo zoom del cluster.
Salida Esperada	El nodo cluster se oculta y sus hijos se muestran.
Salida Obtenida	El nodo cluster se oculta y sus hijos se muestran.

Tabla 7.43: Pruebas: Ocultar nodos de un cluster

Ocultar nodos de un cluster	
Acción	Se disminuye el zoom del mapa hasta sobrepasa el máximo zoom del cluster.
Salida Esperada	El nodo cluster se muestra y sus hijos se ocultan.
Salida Obtenida	El nodo cluster se muestra y sus hijos se ocultan.

Tabla 7.44: Pruebas: Ocultar un nodo

Ocultar un nodo	
Acción	Se pulsa el botón de visibilidad del menú del nodo.
Salida Esperada	El nodo se oculta.
Salida Obtenida	El nodo se oculta.

Tabla 7.45: Pruebas: Mostrar un nodo

Mostrar un nodo	
Acción	Se pulsa el botón de visibilidad del menú del nodo.
Salida Esperada	El nodo se muestra.
Salida Obtenida	El nodo se muestra.

Tabla 7.46: Pruebas: Ocultar un nodo cluster

Ocultar un nodo cluster	
Acción	Se pulsa el botón de visibilidad del menú del cluster.
Salida Esperada	El nodo y todos sus hijos se ocultan.
Salida Obtenida	El nodo y todos sus hijos se ocultan.

Tabla 7.47: Pruebas: Mostrar un nodo cluster

Mostrar un nodo cluster	
Acción	Se pulsa el botón de visibilidad del menú del cluster.
Salida Esperada	El nodo y todos sus hijos se muestran.
Salida Obtenida	El nodo y todos sus hijos se muestran.

Capítulo 8

Casos de Uso Reales

El framework desarrollado en este TFG está siendo utilizado actualmente en dos aplicaciones web que necesitan del apoyo de *Mapache* para la localización de sus activos.

8.1. XtremeOil

XtremeOil es una aplicación web cuyo objetivo es mostrar información en tiempo real de los contenedores en ciudades mediante el uso de sistemas de localización y sensorización.

Esta aplicación se apoya en *Mapache* para localizar estos contenedores, y para mostrar la información obtenida de estos dispositivos de sensorización, como por ejemplo el porcentaje de almacenamiento, la temperatura, y el nivel de humedad del mismo. Para ello, hace uso del módulo cluster de *Mapache* para agrupar estos contenedores en grupos, ya sean por categoría, o para establecer rutas de recogida. Para mostrar la información obtenida de los dispositivos de localización hace uso del módulo de datos que permite mostrar la información en tiempo real del contenedor y mostrar gráficos con el historial de los mismos a lo largo de un periodo de tiempo.

Con esta aplicación se puede ver que el objetivo de *Mapache* no es solo mostrar la posición de activos, sino que también muestra la información de estos para obtener un sistema de visualización en dos niveles en el que puedes ver la localización de activos mientras se visualiza información de los mismos en la misma vista.

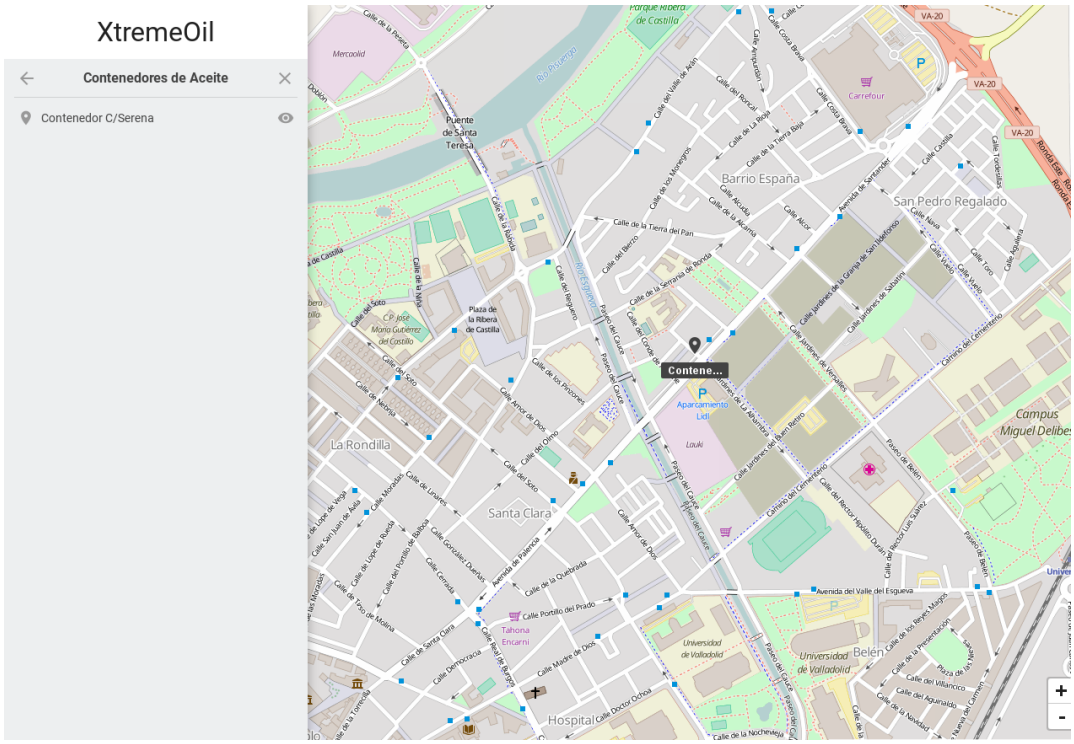


Figura 8.1: Aplicación Real: XtremeOil nodo sin seleccionar

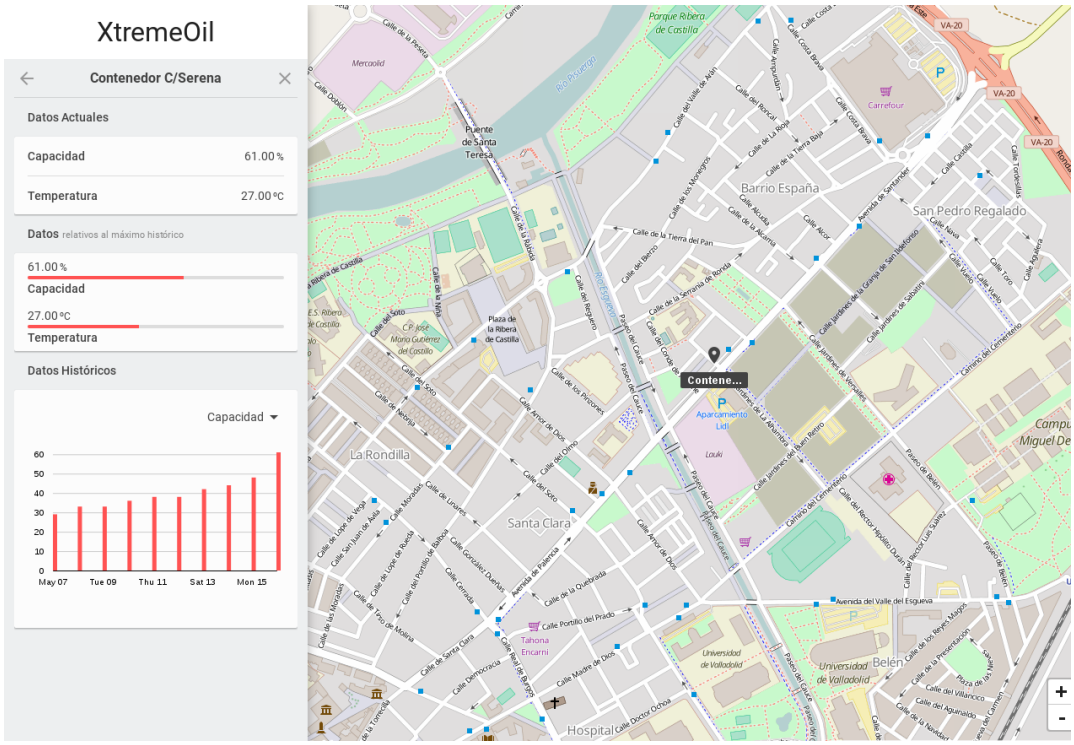


Figura 8.2: Aplicación Real: XtremeOil nodo seleccionado

8.2. XtremeLoc

XtremeLoc es una solución de localización de personas y activos sobre mapas exteriores e interiores bajo la misma aplicación utilizando dispositivos de localización basada en GPS y Bluetooth.

Para transicionar entre mapas interiores y exteriores, XtremeLoc utiliza el módulo de mapas. El módulo de timeout se utiliza para diferenciar entre activos cuya señal se ha perdido, de aquellos que están localizados. El módulo cluster en esta aplicación es utilizado para agrupar personas bajo un mismo subgrupo, por ejemplo agrupar a doctores por un lado y pacientes por el otro en un hospital, o materiales y obreros en una obra.

Figura 8.3: Aplicación Real: XtremeLoc mapa explotaciones

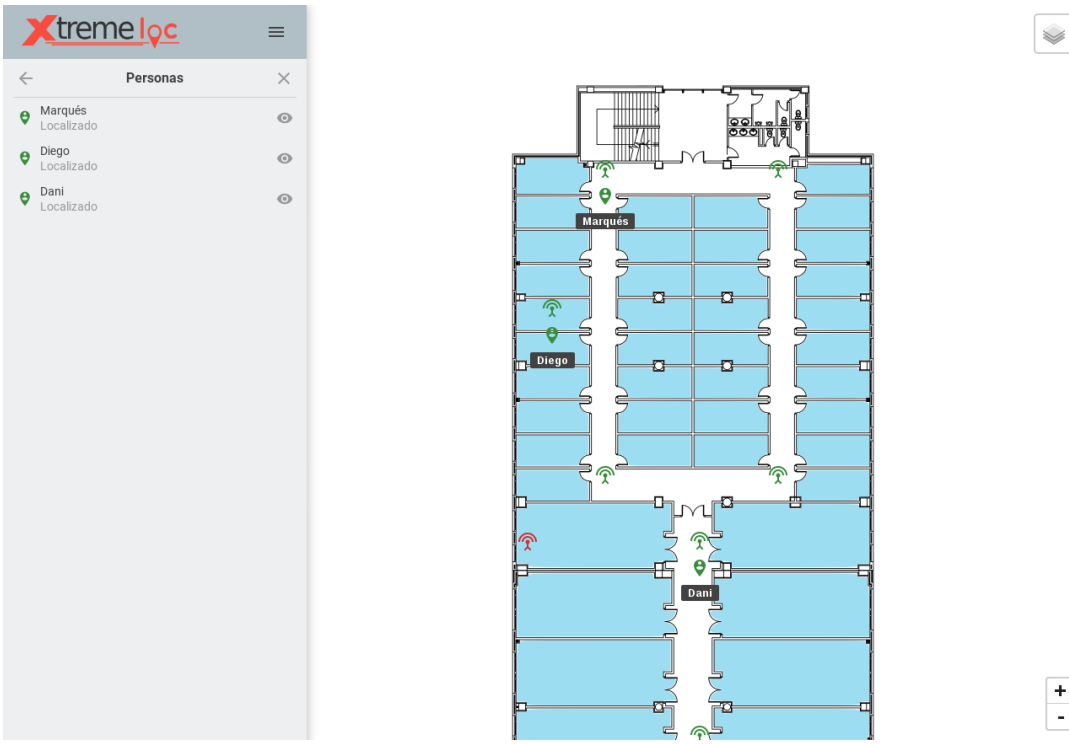


Figura 8.4: Aplicación Real: XtremeLoc mapa interior

Figura 8.5: Aplicación Real: XtremeLoc mapa datos persona

Capítulo 9

Conclusiones y Trabajo Futuro

9.1. Conclusiones

Mapache es una herramienta creada específicamente para dar solución al problema de localización de personas y activos. *Mapache* desarrolla un framework diseñado para todos los navegadores web del mercado, y provee una experiencia de usuario satisfactoria sin importar el tamaño de dispositivo que utilicen. A continuación se detallan las aportaciones de este framework:

- Se ha analizado el estándar ECMAScript 6 para poder sacar el máximo rendimiento al lenguaje y a las nuevas características introducidas en la nueva revisión del estándar.
- Se han analizado y utilizado las diferentes herramientas y frameworks disponibles para JavaScript. Estas herramientas han servido para crear un framework robusto y escalable, y han aportado ideas sobre cómo estructurar el proyecto.
- Se han analizado las diferentes herramientas para la creación de mapas con el objetivo de utilizar aquella que más se ajusta a los requisitos del problema.
- Se ha creado un nuevo framework de posicionamiento de personas y activos llamado *Mapache* que da respuesta a las necesidades encontradas en los requisitos.
- Dentro de Mapache, se ha desarrollado un sistema que permite mostrar personas y activos a la vez sobre el mismo mapa.
- Se ha diseñado asimismo una solución que permite la integración de mapas exteriores e interiores sobre la misma vista.
- Se ha creado un sistema de modularización que permite utilizar el mismo framework para aplicaciones con diferentes necesidades.
- Se ha diseñado el framework de forma tal que funcione correctamente en dispositivos de diferentes resoluciones y que utilizan navegadores distintos.

9.2. Trabajo Futuro

La idea principal de *Mapache* de dar soluciones de posicionamiento a varias aplicaciones con objetivos distintos, hace que la herramienta tenga que crecer a medida que surjan nuevas aplicaciones con requisitos totalmente diferentes. Con el modelo de modularización integrado en *Mapache*, las ampliaciones del mismo pasan por la creación de nuevos módulos que aporten funcionalidad extra a los ya existentes. Además, existen ampliaciones que no tienen que ver con añadir más módulos, sino que son para mejorar la experiencia de usuario al utilizar la aplicación. A continuación se detallan los posibles desarrollos futuros de *Mapache*:

- Se propone el desarrollo de un sistema para que el servidor sólo envíe información sobre lo que está renderizado en la vista actual del navegador, mejorando así la eficiencia del sistema tanto de la parte front-end como del back-end. Para ello se modificar tanto el framework como las APIs que utiliza.
- Se propone el desarrollo de una herramienta de búsqueda para acceder a nodos concretos de manera más directa, en vez de tener que navegar por los menús hasta llegar al nodo deseado.
- Se propone el desarrollo de un conjunto de componentes que ofrezcan funcionalidades adicionales que se consideran deseables, como un componente Popup, que permitiría mostrar avisos urgentes sobre nodos en el mapa; el componente Rutas, que permitiría obtener y mostrar el camino más corto para recorrer todos los nodos que pertenecen al mismo cluster; modificar el módulo Posición para que, cuando un activo se desplace, el nodo se desplace gradualmente a la nueva posición, en vez de actualizar su posición de manera brusca e instantánea, como se hace actualmente.

Referencias

[1] Mike Bostock. Data-driven documents documentation. https://github.com/d3/d3/wiki. [Online; accessed 20-Dic-2016].

[2] Google Developers. Biblioteca javascript de la google places api. https://developers.google.com/places/javascript/?hl=es-419. [Online; accessed 01-Dic-2017].

[3] Facebook. Flux application architecture for building user interfaces. https://facebook.github.io/flux, 2015. [Online; accessed 01-Mar-2017].

[4] César González Ferreras. Javascript. https://aulas.inf.uva.es/pluginfile.php/31470/mod_resource/content/1/javascript.pdf, 2017.

[5] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns Elements of Reusable Object-Oriented Software*. Addison Wesley, 2016.

[6] Jesse James Garrett et al. Ajax: A new approach to web applications. 2005.

[7] Google. Angular - architecture overview. https://angular.io/guide/architecture, 2016. [Online; accessed 01-Mar-2017].

[8] Mordechai Haklay and Patrick Weber. Openstreetmap: User-generated street maps. *IEEE Pervasive Computing*, 7(4):12–18, 2008.

[9] Ecma International. Ecmascript 2015 language specification. http://www.ecma-international.org/ecma-262/6.0/. [Online; accessed 12-Dic-2016].

[10] Philippe Le Hégaret, Ray Whitmer, and Lauren Wood. Document object model (dom). w3c recommendation, 2005.

[11] Leaflet. Leaflet api. http://leafletjs.com/reference-1.0.3.html. [Online; accessed 8-Dic-2016].

[12] Hongki Lee, Sooncheol Won, Joonho Jin, Junhee Cho, and Sukyoung Ryu. Safe: Formal specification and implementation of a scalable analysis framework for ecmascript. In *International Workshop on Foundations of Object-Oriented Languages (FOOL)*, volume 10, 2012.

[13] Mapbox. Mapbox gl js api. https://www.mapbox.com/mapbox-gl-js/api/. [Online; accessed 4-Dic-2016].

[14] Polytechnique Montreal. Unified process for education. http://www.upedu.org/. [Online; accessed 03-Dic-2017].

[15] OpenLayers. Openlayers api. https://openlayers.org/en/latest/apidoc/. [Online; accessed 6-Dic-2016].

[16] Anne Van Kesteren and Dean Jackson. The xmlhttprequest object. *World Wide Web Consortium, Working Draft WD-XMLHttpRequest-20070618*, 2007.

[17] webpack. Webpack dev server. https://webpack.github.io/docs/webpack-dev-server.html, 2017.

Anexos

Anexo I

Manual del programador

Instalación

El framework *Mapache* necesita de librerías JavaScript para su correcto funcionamiento, es por ello que el proyecto está construido utilizando el gestor de paquetes NPM. NPM es el gestor de paquetes que utilizan las aplicaciones Node.js, por lo que la forma de obtenerlo es instalando Node.js.

```
1 $ npm install npm@latest -g
```

Para comprobar que la instalación es correcta, comprobamos la versión de NPM con:

```
1 $ npm -v
```

Una vez tenemos instalado npm tenemos que obtener las dependencias de *Mapache*.

Dependencias

Las librerías en las que se apoya *Mapache* son las siguientes:

Browserify Permite el uso de módulos JavaScript al estilo Node.js.

Exorcist Complemento de Browserify para poder debuggear código una vez Browserify compacta el código en un único script.

Babelify Babel es un preprocesador de código ES6 que transforma el código a ES5 para obtener el máximo nivel de compatibilidad con todos los navegadores. Babelify es el transformador de Babel para Browserify, de modo que Babel se ejecuta sobre el módulo final generado por Browserify.

babel-preset-es2015 Plugin de Babel capaz de transformar código ES6 en ES5.

Para instalar estas dependencias se necesita crear un proyecto NPM, en este caso *Mapache*. En el fichero *package.json* se encuentran estas librerías de las que depende el framework. Para instalar estas dependencias lanzamos el siguiente comando desde el directorio donde se encuentra *Mapache*:

```
$ npm install
```

Este comando instala el paquete *Mapache*, y todas las dependencias de este. Si se quieren añadir más dependencias para un desarrollo futuro, se necesita añadirla desde desde el repositorio npm con el siguiente comando:

```
$ npm install <nombre-del-paquete> --save-dev
```

Ejecución

NPM permite ejecutar scripts definidos en *package.json*. Existen dos scripts de generación del código fuente de *Mapache*:

bundle Ejecuta Browserify con el transformador de babel usando ES5 obtener el archivo *mapache.js* para ser usado en producción.

debug Ejecuta Browserify con exorcist genera el archivo *mapache-debug.js* junto con el archivo de mapeado *mapache-debug.map.js* para ser usado en un entorno de desarrollo.

Estos scripts se lanzan con el siguiente comando:

```
$ npm run <nombre-del-script>
```

Tras este paso, se habrán creado los ficheros necesarios para su utilización en el directorio ./bin/.

La hoja de estilos de *Mapache* se encuentra en el directorio dist dentro del proyecto. Para poder utilizar SASS es necesario instalar Ruby primero, es aconsejable instalar ruby con RVM (Ruby Version Manager) ya que gestiona y actualiza las diferentes versiones de Ruby y las gemas instaladas. Con Ruby instalado hay que instalar la gema de SASS:

```
$ gem install sass
```

Con SASS instalado sólo queda preprocesar los archivos SASS en CSS estándar, para ello se ejecuta el siguiente comando:

```
$ sass _mapache.scss mapache.css
```

Este comando genera el archivo *mapache.css* final junto con el source map *mapache.css.map*. Durante el proceso de desarrollo se recomienda que SASS procese los ficheros automáticamente si detecta un cambio en los fuentes. Para ello se ejecuta el siguiente comando:

```
$ sass --watch _mapache.scss:mapache.css
```

Anexo II

Manual de usuario

Lado del Cliente

Para utilizar *Mapache* es necesario disponer de los siguientes ficheros:

d3.js

jquery.js

jquery-ui.js

leaflet.js

leaflet.css

mapache.css

mapache.js

Es necesario meter estos ficheros en la aplicación web que va a hacer uso de *Mapache*. A la hora de utilizar el framework es necesario que en la vista existan por lo menos dos elementos HTML, uno para el mapa, y otro para el menú principal de *Mapache*. El menú se recomienda que tenga 360px de ancho para obtener la mejor experiencia de usuario. El contenedor del mapa puede tener el tamaño que se desee, pero de nuevo, habrá que analizar las necesidades de la aplicación.

Además, es importante el orden en el que se importan los scripts de JavaScript. El orden es, primero jQuery, después d3, seguido de leaflet, y finalmente *Mapache*. Si no se sigue este orden, se cargan primero los scripts que requieren dependencias que aún no han sido cargadas. La página HTML podría ser como la siguiente:

```
1 <!DOCTYPE html>
2 <html lang="es">
3     <head>
4         <meta http-equiv="Content-Type" content="text/html;
5             charset=UTF-8">
6         <link rel="stylesheet" type="text/css" href="mapache.css">
7         <link rel="stylesheet" href="leaflet.css" />
8         <script src="jquery.min.js"></script>
```

```
9           <script src="jquery-ui.min.js"></script>
10          <script src="d3.v4.min.js"></script>
11          <script src="leaflet.js"></script>
12          <script src="mapache.js"></script>
13          <meta name="viewport" content="width=device-width,
14              initial-scale=1.0, maximum-scale=1.0, user-scalable=no" />
15          <title>Prueba Mapache</title>
16      </head>
17      <body>
18          <div id="app" class="container left-menu-open">
19              <div id="left-pane" class="left-pane-container">
20                  <div id="main-menu" class="left-pane-menu">
21                      <div class="sidebar-header">
22                          Mapache
23                      </div>
24                      <ul id="sidenav-menu" class="sidebar-list">
25                      </ul>
26                  </div>
27              </div>
28              <div id="main-container">
29                  <div id="map"></div>
30              </div>
31          </div>
32          <script src="mapache-test.js"></script>
33      </body>
34  </html>
```

El último paso es crear un script que llame a la función de inicialización de *Mapache*, y le pase por argumento los identificadores de los elementos HTML a los que si tiene que acoplar *Mapache*. Un script sencillo de ejemplo sería el siguiente:

```
1       (function () {
2           Mapache.initMapache({
3               mapHook: 'map',
4               menuHook: 'sidenav-menu'
5           });
6       })();
```

Lado del Servidor

A la hora de implementar *Mapache* en una aplicación, es necesario que esta implemente un método de comunicación HTTPS con *Mapache* para enviarle la información que tiene que mostrar. Este controlador devuelve un objeto JSON con la estructura solicitada por *Mapache*, esta estructura es la siguiente:

Atributo	**Tipo**	**Opcional**	**Descripción**
map	Node	No	Nodo mapa que hay que renderizar
nodes	Node [ ]	No	Nodos que pertenecen al mapa

Node

Atributo	Tipo	Opcional	Descripción
id	string	No	Identificador único del nodo
name	string	No	Nombre del nodo, utilizado en el menú
position	Position	No	Posición del nodo en el mapa actualmente renderizado
timeout	Timeout	Sí	Da al nodo la funcionalidad de controlar su linea de vida
tooltip	string	Sí	Nombre identificativo para que se muestre justo bajo el marcador
map	Map	Sí	Da al nodo la funcionalidad de mapa
cluster	Cluster	Sí	Da al nodo la funcionalidad de cluster
state	State	Sí	Da al nodo la funcionalidad de tener un estado
data	Data [ ]	Sí	Da al nodo la funcionalidad de mostrar información detallada sobre sí mismo

Ejemplo

```
{
    id: 'nodo-0',
    name: 'Juan Palacios',
    position: { x: 0, y: 0, z: 0 }
}
```

Position

Atributo	Tipo	Opcional	Descripción
x	number	No	Coordenada X de la posición
y	number	No	Coordenada Y de la posición
z	number	No	Coordenada Z de la posición, se corresponde con la planta de un mapa interior, para el mapa del mundo su valor es cero
container	number	Sí	Identificador del Nodo en el que se tiene que mostrar contenido este Nodo

Ejemplo

```
{
    x: 0.32,
    y: 0.5,
    z: 3
}
```

Timeout

Atributo	Tipo	Opcional	Descripción
delete Time	number	No	Tiempo transcurrido sin recibir información del servidor para eliminar el nodo del mapa
warning Time	number	Sí	Tiempo transcurrido en segundos sin recibir información del servidor para mostrar el estado de alerta
warning State	State	Sí	Estado que sobreescribe al estado del nodo cuando se sobrepasa el warningTime

Ejemplo

```
{
    warningTime: 2,
    deleteTime: 5,
    warningState: {
        name: 'Desactualizado',
        color: '#FF9800'
    }
}
```

Map

Atributo	Tipo	Opcional	Descripción
crs	string	No	Sistema de referencia del mapa actual. **Valores:** <'indoor', 'outdoor'>
maxZoom	number	No	Máximo zoom que permite el mapa
minZoom	number	No	Mínimo zoom que permite el mapa
width	number	Sólo indoor	Anchura en pixels de la imagen del mapa interior
height	number	Sólo indoor	Altura en pixels de la imagen del mapa interior
floors	Floor [ ]	No	Las plantas del mapa

Ejemplo

```
{
    crs: 'outdoor',
    minZoom: 0,
    maxZoom: 18,
    floors: [
        {
            main: true,
            name: 'world_map',
            tileURL: 'http://<url-servidor-mapas>/{z}/{x}/{y}.png'
```

```
        }
    ]
}
```

Floor

Atributo	Tipo	Opcional	Descripción
main	boolean	Sí	Indica si la planta es la primera que se tiene que renderizar en el mapa tras ser cargado
name	string	No	Nombre de la planta, se usa para identificar la planta en los controles de cambio de planta del mapa
tileURL	string	No	Dirección URL de la que se puede obtener los tiles del mapa

Ejemplo

```
{
    name: 'world_map',
    tileURL: 'http://<url-servidor-mapas>/{z}/{x}/{y}.png'
}
```

Cluster

Atributo	Tipo	Opcional	Descripción
maxZoom	number	No	Nivel de zoom de mapa en el cual se deja de renderizar el nodo cluster y se renderizan sus hijos
children	string [ ]	Sí	Lista con los identificadores de los nodos que pertenecen a este cluster

Ejemplo

```
{
    maxZoom: 16,
    children: ['receiver:0', 'receiver:1', 'receiver:2']
}
```

State

Atributo	Tipo	Opcional	Descripción
name	string	No	Nombre del estado para mostrar en el menú
color	string	No	Color del icono cuando el estado esta activo
icon	boolean	No	Indica si el estado debe mostrarse en un icono o en el marcador

Ejemplo

```
{
    name: 'Lleno',
    color: '#FF9800',
    icon: true
}
```

Data

Atributo	Tipo	Opcional	Descripción
types	ConcreteData [ ]	No	Todos los datos diferentes de los que se quiere mostrar información

Ejemplo

```
[
    {
        name: 'Temperatura',
        unit: 'Celsius',
        values: [
            {
                value: 25,
                timestamp: 1494435849
            },
            {
                value: 25,
                timestamp: 1494435848
            }
        ]
    }
]
```

ConcreteData

Atributo	Tipo	Opcional	Descripción
name	string	No	Nombre del dato a mostrar
unit	string	No	Nombre de la unidad de medida
values	Value [ ]	No	Historial con los datos a mostrar

Ejemplo

```
{
    name: 'Temperatura',
    unit: 'Celsius',
    values: [
        {
            value: 25,
            timestamp: 1494435849
        },
        {
            value: 25,
            timestamp: 1494435848
        }
    ]
}
```

Value

Atributo	Tipo	Opcional	Descripción
value	string	No	Valor del dato
timestamp	number	No	Momento en el que se realizó la medida

Ejemplo

```
{
    value: 25,
    timestamp: 1494435849
}
```