



Escuela
Politécnica
Superior

Web Aumentada para destinos turísticos inteligentes



Grado en Ingeniería Multimedia

Trabajo Fin de Grado

Autor:
Francisco Javier García Fernández

Tutor/es:
José Norberto Mazón López
Irene Garrigos Fernández

julio 2021

 Universitat d'Alacant
Universidad de Alicante

Versión del documento

2021.07.11

Licencia

Se permite la reproducción, distribución y comunicación pública de la obra, incluso con fines comerciales siempre y cuando reconozca y cite la obra de la forma especificada por el autor o el licenciante.



Resumen

El sector del turismo digital está sufriendo grandes cambios en la actualidad debido a la aparición y auge de nuevas tecnologías de información y comunicación. Dichas variaciones se reflejan, sobre todo, en aplicaciones y sitios web orientados a este sector que realizan una muestra de información correspondiente al destino turístico siguiendo un esquema tradicional.

Dicho esquema no contempla las necesidades reales de los usuarios a la hora de adquirir los conocimientos necesarios para utilizar un servicio turístico de estas características. La aparición de nuevas metodologías de gestión turística como son la creación de destinos turísticos inteligentes ha generado una gran cantidad de nuevas posibilidades basadas en la experiencia de usuario.

Gracias a esto, es posible elaborar aplicaciones y funcionalidades nuevas que permitan la obtención rápida y efectiva de información por parte del turista. En dicho contexto, este proyecto se centra en la aumentación de las funcionalidades existentes de los sitios web orientados al turismo.

En concreto, se ha realizado un aumento de web en el sitio oficial del parque natural de Lagunas de la Mata y Torrevieja. Para realizar dicho incremento de funcionalidades se han empleado los denominados scripts de usuario o user-scripts que permiten la adición de nuevas características sin modificar el código fuente del sitio web oficial.

Siguiendo estas indicaciones, se han desarrollado scripts de usuario que permiten conocer la predicción de tiempo que puede hacer en una fecha futura o que hubo en fechas pasadas, elaborar un mapa interactivo que aloja los datos mostrados en la web de una manera simple e intuitiva y crear una tabla de avistamiento de aves en el parque natural.

La elaboración de estos user-scripts genera un incremento considerativo en la experiencia de usuario aportando información de interés a la hora de realizar la toma de decisión correspondiente a consumir dicho servicio turístico, o cualquier otro. Esta situación provoca un crecimiento del consumo referente al turismo digital que beneficia tanto a los destinos turísticos como a los propios turistas al aumentar la comunicación existente entre servicio y consumidor.

En definitiva, la aumentación web utilizada para mejorar la experiencia de usuario implica un mayor uso de estos sitios web, incrementando el consumo del turismo digital de cara al futuro.

Motivación, justificación y objetivo general

El sector del turismo en el ámbito de los sitios web presenta una falta de información relevante a la hora de planificar viajes en la actualidad. Esto otorga como consecuencia la necesidad de invertir una cantidad razonable y extensa de tiempo para poder informarse de características importantes a la hora de visitar un destino turístico como puede ser el tiempo atmosférico que hará a la hora de realizar el viaje.

El desarrollo de este proyecto puede facilitar esta toma de decisión por parte de los turistas, consistente en viajar a un determinado lugar aportando información suficiente como para tomar esa determinación. Además, beneficiaría al sector del turismo aumentando la experiencia de usuario mediante el incremento de la información que sus sitios webs oficiales poseen de manera general.

Gracias a un aumento de web de las funcionalidades que estos sitios webs presentan, se consigue un ahorro cuantitativo de tiempo al poseer toda la información necesaria para decidir si viajar a ese destino turístico o a otro.

Se ha decidido enfocar estas características a los sitios web de los parques naturales ya que en su mayoría presentan bastante escasez en la información que proporcionan para realizar el viaje. Por tanto, son sensibles al aumento de funcionalidades que mejorarán la experiencia de usuario en el momento de utilizar estos sitios web.

Además, el reto de modificar un servicio que ya está creado sin poder cambiar el sitio web original aumenta en gran medida las habilidades de ingeniería que se puedan poseer. Esto se justifica mediante el uso de alternativas que toca investigar y utilizar a la hora de generar funcionalidades nuevas en un entorno no desarrollado desde 0.

Dicha situación conlleva la adaptación de las soluciones al formato del servicio ya creado, dificultando, en el caso del desarrollo web, las cuestiones relativas al diseño CSS que puedan poseer los elementos generados, entre otros inconvenientes.

Mediante este proyecto se pueden adquirir las habilidades necesarias para enfrentar estas problemáticas que son comunes en el ámbito laboral ya que en muchas ocasiones no se desarrolla un software desde 0 sino que se mejora la calidad del existente empleando la creación de nuevas funcionalidades.

Agradecimientos

Quiero agradecer a mis familiares y amigos cercanos por apoyarme en mis momentos difíciles durante mi paso por la universidad. En concreto, quiero agradecer a mi madre, a mi abuela y a mi tía *Ana* por la paciencia que han tenido conmigo estos años y los valores que me han inculcado a lo largo de mi vida. También le dedico este trabajo a mi abuelo, a quien tengo presente día tras día.

Gracias *María del Rosario Fernández García* por demostrarme que la verdadera amistad existe. Gracias *Jenifer Boente Pereira* por ser una luz para mí en los peores momentos de mi vida. Gracias *Carlos de la Fuente Torres* por aconsejarme y escucharme siempre que lo he necesitado.

Además, también quiero dar las gracias a internet por existir y enseñarme todo lo que mis profesores no me han enseñado.

Gracias a la universidad por hacer de mí una persona pobre y mentalmente inestable y depresiva.

También le agradezco el desvalorizar mi opinión y destruir toda ilusión y motivación para sacarme esta titulación.

Gracias a todos esos profesores que me han enseñado que leer una diapositiva es ser docente.

Y, por último, gracias a mis compañeros de ABP por enseñarme que hay gente que ni el infierno acepta.

Citas

*Hay que amar lo que es digno de ser amado y odiar lo que es odioso,
mas hace falta buen criterio para distinguir entre lo uno y lo otro.*

Robert Frost

Sólo la inteligencia se examina a sí misma.

Jaime Luciano Balmes

El mundo no está en peligro por las malas personas sino por aquellas que permiten la maldad

Albert Einstein

Índice de contenidos

Resumen.....	3
Motivación, justificación y objetivo general	4
Agradecimientos	5
Citas.....	6
Índice de figuras	9
Índice de tablas	14
1. Introducción	15
2. Marco teórico.....	18
2.1. Estudio de extensiones	20
3. Objetivos	27
3.1. Objetivos principales	27
3.2. Objetivos secundarios	28
4. Metodología	30
5. Arquitectura del trabajo.....	32
6. Weather Prediction Calendar Script (WPCS).....	34
6.1. Tecnologías utilizadas.....	34
6.2. Obtención del API key de Aemet	35
6.3. Desarrollo	37
6.3.1. Datos de Torrevieja y de su estación Aemet más cercana.....	39
6.3.2. Creación de la interfaz de usuario.....	44
6.3.3. Creación de las peticiones de tiempo pasado y futuro a Aemet	47
6.3.4. Muestra de datos al usuario.....	52
6.3.5. Problemas encontrados	81
6.3.6. Visualización de cambios.....	84
6.3.7. Pruebas realizadas.....	85
7. Natural Park Interactive Map (NPIM).....	88

7.1.	Tecnologías utilizadas.....	88
7.2.	Desarrollo	89
7.2.1.	Creación del fichero JSON con los datos del sitio web.....	90
7.2.2.	Obtención del fichero JSON	94
7.2.3.	Elaboración del mapa interactivo	95
7.2.4.	Creación de la UI	113
7.2.5.	Generación de los eventos de visualización.....	118
7.2.6.	Problemas encontrados	122
7.2.7.	Visualización de cambios.....	124
7.2.8.	Pruebas realizadas.....	125
8.	Natural Park Bird Sighting (NPBS)	129
8.1.	Tecnologías Utilizadas	129
8.2.	Obtención del API key de eBird.....	130
8.3.	Desarrollo	134
8.3.1.	Obtención del código referente a la provincia de Alicante.....	134
8.3.2.	Creación de las peticiones de avistamiento de aves.....	136
8.3.3.	Elaboración de la interfaz de usuario.....	139
8.3.4.	Muestra de datos	141
8.3.5.	Problemas encontrados	152
8.3.6.	Visualización de cambios.....	155
8.3.7.	Pruebas realizadas.....	156
9.	Conclusiones y trabajo futuro	159
9.1.	Conclusiones.....	159
9.2.	Trabajo futuro	160
	Referencias.....	161
	Recursos utilizados.....	165

Índice de figuras

Figura 1. Interfaz de Tampermonkey	21
Figura 2. Interfaz de Greasemonkey	22
Figura 3. Interfaz de Violentmonkey	23
Figura 4. Resultado final según las 3 extensiones.....	24
Figura 5. Porcentaje de utilización de navegadores web en España	25
Figura 6. Porcentaje de utilización de navegadores web a nivel global	26
Figura 7. Objetivos principales	28
Figura 8. Objetivos secundarios	29
Figura 9. Metodología-Tablón de Trello general.....	31
Figura 10. Metodología-Tablón de Trello específico para las tareas	31
Figura 11. Arquitectura general de los user-scripts	33
Figura 12. WPCS-Logo e icono.....	34
Figura 13. WPCS-Formulario de obtención del API key de Aemet.....	36
Figura 14. WPCS-Correo enviado para la obtención del API key de Aemet.....	36
Figura 15. WPCS-Obtención del API key de Aemet.....	37
Figura 16. WPCS-Cabecera	38
Figura 17. WPCS-Arquitectura.....	39
Figura 18. WPCS-Clase Municipio y variable global	40
Figura 19. WPCS- Obtención de los datos del municipio de Torrevieja	41
Figura 20. WPCS-Obtención de todas las estaciones Aemet	42
Figura 21. WPCS-Obtención de la estación Aemet más cercana	42
Figura 22. WPCS-Calculo de distancias empleando coordenadas GPS	43
Figura 23. WPCS-Clase Position.....	44
Figura 24. WPCS-Obtención del año, mes y días presentes en el calendario del sitio web.....	45
Figura 25. WPCS-Inserción de las imágenes correspondientes a la interfaz de usuario	45
Figura 26. WPCS-Comparación de fechas	46
Figura 27. WPCS-Icono insertado para realizar la interfaz de usuario.....	46
Figura 28. WPCS-Interfaz de usuario.....	46
Figura 29. WPCS-Constantes utilizadas.....	47
Figura 30. WPCS-Conversión de día y mes a un formato de 2 dígitos	48
Figura 31. WPCS-Conversión de una fecha a formato UTC.....	48

Figura 32. WPCS-Petición para una fecha pasada.....	49
Figura 33. WPCS-Segunda petición para obtener los datos de una fecha pasada.....	49
Figura 34. WPCS-Petición para una fecha futura	51
Figura 35. WPCS-Segunda petición para obtener los datos de una fecha futura	51
Figura 36. WPCS-Filtrado de fecha futura para el día seleccionado por el usuario.....	52
Figura 37. WPCS-Mensaje modal de error en el servidor (código).....	53
Figura 38. WPCS-Mensaje modal de error en el servidor (web).....	54
Figura 39. WPCS-Mensaje modal de cargando datos (código)	55
Figura 40. WPCS-Mensaje modal de cargando datos (web)	55
Figura 41. WPCS-Mensaje modal de inexistencia de datos (código)	56
Figura 42. WPCS-Mensaje modal de inexistencia de datos (web)	56
Figura 43. WPCS-Cambio de formato a "dd-mm-aaaa"	57
Figura 44. WPCS-Constantes referentes a los distintos meses junto con sus horas de sol	59
Figura 45. WPCS-Muestra adecuada de campos en función de su existencia.....	60
Figura 46. WPCS-Obtención del tipo de día	63
Figura 47. WPCS-Obtención del tipo de precipitación.....	63
Figura 48. WPCS-Creación del copyright de Aemet	63
Figura 49. WPCS-Mensaje modal de tiempo pasado (código).....	63
Figura 50. WPCS-Imágenes identificativas del tipo de día	64
Figura 51. WPCS-Mensaje modal de tiempo pasado (web).....	64
Figura 52. WPCS-Inserción adecuada de datos en el array utilizado por el gráfico de líneas	67
Figura 53. WPCS-Extracción de datos para generar el gráfico de líneas.....	68
Figura 54. WPCS-Función puente para crear el gráfico de líneas	68
Figura 55. WPCS-Creación del gráfico de líneas.....	69
Figura 56. WPCS-Filtrado de datos para las tablas de predicción.....	70
Figura 57. WPCS-Creación de filas y columnas de la tabla de predicción.....	71
Figura 58. WPCS-Obtención de filas y columnas de la tabla de predicción	72
Figura 59. WPCS-Creación de la tabla de predicción	72
Figura 60. WPCS-Creación del mensaje modal de predicción de tiempo (gráfico-código).....	73
Figura 61. WPCS-Creación del mensaje modal de predicción de tiempo (gráfico-web)	73
Figura 62. WPCS-Extracción de todos los datos para la tabla de predicción 2	75
Figura 63. WPCS-Extracción de los datos del viento para la tabla de predicción	76
Figura 64. WPCS-Creación de filas y columnas de la tabla del viento.....	76
Figura 65. WPCS-Creación de la tabla de predicción del viento	76
Figura 66. WPCS-Creación de la tabla de predicción de valores mínimos y máximos.....	77

Figura 67. WPCS-Creación del mensaje modal de predicción de tiempo (tablas-código)	78
Figura 68. WPCS-Creación del mensaje modal de predicción de tiempo (tablas-web).....	78
Figura 69. WPCS-Creación del mensaje modal de predicción de tiempo (párrafos-código)	80
Figura 70. WPCS-Creación del mensaje modal de predicción de tiempo (párrafos-web)	80
Figura 71. WPCS-Problemas encontrados (gráfico de líneas)	84
Figura 72. WPCS-Comparativa de activación del user-script	85
Figura 73. WPCS-Pruebas en navegador Google Chrome	86
Figura 74. WPCS-Pruebas en navegador Mozilla Firefox	87
Figura 75. WPCS-Pruebas en navegador Opera.....	87
Figura 76. NPIM-Logo e icono	88
Figura 77. NPIM-Arquitectura	89
Figura 78. NPIM-Archivo JSON (datos generales)	91
Figura 79. NPIM-Archivo JSON (área).....	91
Figura 80. NPIM-Archivo JSON (rutas)	93
Figura 81. NPIM-Archivo JSON (sitios de información).....	94
Figura 82. NPIM-Carga del archivo JSON	95
Figura 83. NPIM-Cabecera	96
Figura 84. NPIM-Carga del CSS de Leaflet.....	97
Figura 85. NPIM-Clase Map.....	97
Figura 86. NPIM-Creación del mapa de Leaflet junto con el objeto Map.....	98
Figura 87. NPIM-Creación de la información a mostrar en el mapa interactivo.....	99
Figura 88. NPIM-Modificación del TileLayer	99
Figura 89. NPIM-TileLayer del mapa interactivo (web).....	100
Figura 90. NPIM-Establecimiento del área de visión del mapa	100
Figura 91. NPIM-Área de visión del mapa (web).....	100
Figura 92. NPIM-Mensaje modal Leaflet (área)	101
Figura 93. NPIM-Creación del evento hover para las líneas del mapa interactivo	102
Figura 94. NPIM-Creación del reborde que representa el área del parque natural	102
Figura 95. NPIM-Área del parque natural (web).....	103
Figura 96. NPIM-Clase Route	104
Figura 97. NPIM-Mensaje modal Leaflet (recorrido de ruta).....	105
Figura 98. NPIM-Creación del objeto Route	106
Figura 99. NPIM-Creación de las rutas del parque natural	106
Figura 100. NPIM-Iconos que representan las paradas del parque natural	107
Figura 101. NPIM-Constantes utilizadas para la creación de las paradas.....	107

Figura 102. NPIM-Creación del icono que identifica las paradas.....	108
Figura 103. NPIM-Creación del evento hover para los marcadores de las paradas	109
Figura 104. NPIM-Mensaje modal Leaflet (paradas)	109
Figura 105. NPIM-Creación de las paradas del parque natural	110
Figura 106. NPIM-Rutas del parque natural (web)	110
Figura 107. NPIM-Creación del icono de los sitios de información	111
Figura 108. NPIM-Icono de marcador para los centros de información.....	111
Figura 109. NPIM-Mensaje modal Leaflet (sitios de información)	112
Figura 110. NPIM-Creación de los sitios de información	112
Figura 111. NPIM-Mapa interactivo completo (web)	113
Figura 112. NPIM-Creación del formulario que contiene la interfaz de usuario	114
Figura 113. NPIM-Creación del estilo que posee el formulario de la interfaz de usuario	114
Figura 114. NPIM-Creación de los botones de interacción de la interfaz de usuario.....	115
Figura 115. NPIM-Creación del estilo CSS de los botones de interacción	116
Figura 116. NPIM-Botón de interacción (área)	116
Figura 117. NPIM-Botones de interacción (rutas).....	117
Figura 118. NPIM-Botón de interacción (sitios de información).....	118
Figura 119. NPIM-Determinación de la visibilidad de los elementos del mapa	119
Figura 120. NPIM-Recreación de los elementos del mapa	119
Figura 121. NPIM-Destrucción de los elementos del mapa	120
Figura 122. NPIM-Creación completa de la interfaz de usuario	120
Figura 123. NPIM-Interfaz de usuario (web).....	121
Figura 124. NPIM-Mapa interactivo junto con interfaz de usuario (web)	121
Figura 125. NPIM-Mensaje modal (paradas-web)	121
Figura 126. NPIM-Comparativa de activación del user-script.....	125
Figura 127. NPIM-Pruebas en navegador Google Chrome	126
Figura 128. NPIM-Pruebas en navegador Mozilla Firefox.....	127
Figura 129. NPIM-Pruebas en navegador Opera.....	127
Figura 130. NPBS-Logo e icono.....	129
Figura 131. NPBS-Obtención del API key de eBird (formulario de registro)	131
Figura 132. NPBS-Obtención del API key de eBird (correo de validación).....	131
Figura 133. NPBS-Obtención del API key de eBird (correo de bienvenida)	132
Figura 134. NPBS-Obtención del API key de eBird (formulario de solicitud)	133
Figura 135. NPBS-API key obtenido	133
Figura 136. NPBS-Arquitectura	134

Figura 137. NPBS-Obtención del código de la Comunidad Valenciana.....	135
Figura 138. NPBS-Obtención del código de la provincia de Alicante	136
Figura 139. NPBS-Conversión de fecha al formato "aaaa/mm/dd"	137
Figura 140. NPBS-Filtrado de resultados por nombre de localidad	137
Figura 141. NPBS-Petición de avistamiento de aves en una fecha específica	138
Figura 142. NPBS-Petición de avistamiento de aves en los últimos 30 días	139
Figura 143. NPBS-Creación del estilo CSS de la interfaz de usuario.....	140
Figura 144. NPBS-Creación de la interfaz de usuario.....	140
Figura 145. NPBS-Interfaz de usuario (web)	140
Figura 146. NPBS-Mensaje modal de no indicación de fecha (código).....	142
Figura 147. NPBS-Mensaje modal de no indicación de fecha (web).....	142
Figura 148. NPBS-Conversión de fecha al formato “dd-mm-aaaa”	143
Figura 149. NPBS-Mensaje modal de especificación de fecha futura (código).....	143
Figura 150. NPBS-Mensaje modal de especificación de fecha futura (web)	143
Figura 151. NPBS-Mensaje modal de inexistencia de resultados (código)	144
Figura 152. NPBS-Mensaje modal de inexistencia de resultados (web)	144
Figura 153. NPBS-Creación del elemento HTML "<table></table>"	146
Figura 154. NPBS-Creación del estilo CSS que poseerá la tabla.....	146
Figura 155. NPBS-Creación del DataTable.....	146
Figura 156. NPBS-Tabla de avistamiento de aves	147
Figura 157. NPBS-Icono de avistamiento de aves.....	148
Figura 158. NPBS-Creación del mapa de Leaflet	149
Figura 159. NPBS-Creación del icono de avistamiento de aves	149
Figura 160. NPBS-Creación del marcador correspondiente al ave avistada	149
Figura 161. NPBS-Mensaje modal Leaflet (ave avistada).....	149
Figura 162. NPBS-Creación completa del mapa de Leaflet	150
Figura 163. NPBS-Mensaje modal ave avistada (web).....	150
Figura 164. NPBS-Eliminación del DataTable en caso de su existencia	151
Figura 165. NPBS-Mensaje modal de insuficiencia de datos (código)	152
Figura 166. NPBS-Mensaje modal de insuficiencia de datos (web)	152
Figura 167. NPBS-Comparativa de activación del user-script	155
Figura 168. NPBS-Pruebas en navegador Google Chrome.....	156
Figura 169. NPBS-Pruebas en navegador Mozilla Firefox	157
Figura 170. NPBS-Pruebas en navegador Opera.....	157

Índice de tablas

Tabla 1. Compatibilidad entre extensiones.....	20
Tabla 2. Porcentaje de uso de navegadores web.....	26
Tabla 3. Etiquetas básicas presentes en la cabecera de un user-script	38
Tabla 4. WPCS-Cálculo de horas totales de sol en Torrevieja	59
Tabla 5. WPCS-Clasificación del tipo de precipitación	61
Tabla 6. WPCS-Clasificación del tipo de día	62

1. Introducción

En la actualidad, es difícil encontrar sitios web orientados al turismo que posean toda la información necesaria para poder utilizar sus funcionalidades como se desea. Esta situación es consecuencia de una gestión tradicional de los destinos turísticos, la cual no contempla las necesidades reales de los usuarios a la hora de adquirir el conocimiento necesario para poder emplear el servicio que se les proporciona. Debido a esto, se genera el requisito de buscar información en otros sitios web de interés para así tomar la decisión de consumir ese servicio o cualquier otro.

Dichas necesidades reales especificadas se corresponden principalmente con las 2 primeras etapas del ciclo de viaje. Este ciclo determina las fases por las que pasa un turista a la hora de realizar un viaje. Según el sitio web easystaytech.com [1], estas 2 primeras fases se identifican por el nombre de *dreaming* y *planning*.

El *dreaming* se corresponde con la fase de planteamiento de un viaje. Es en esta fase en la que el turista piensa a qué lugares ir y qué le motiva a elegir ese sitio en lugar de otro. También influyen notablemente la experiencia de amigos y familiares a la hora de elegir un destino turístico en concreto.

El *planning* se identifica como la etapa de planificación de un viaje. Es en esta fase donde se ha decidido viajar a un determinado destino turístico y se están planeando las cuestiones relativas a los lugares que se van a visitar en dicho destino y qué actividades van a realizarse, entre otras. La opinión de amigos y familiares es muy relevante en esta etapa como también lo es la información que pueda extraer el turista de los sitios web oficiales.

Pongamos como ejemplo el caso de una turista llamada Ana que desea viajar a un parque natural porque le apasiona la naturaleza. En su investigación para averiguar qué reserva natural visitar no encuentra, en ningún sitio web oficial, información que le pueda ayudar a conocer las aves que se puedan avistar en dichos parques naturales. También le gustaría conocer qué tiempo atmosférico puede hacer para cuando quiera realizar la reserva y qué rutas oferta el parque natural, además de los lugares por los que pasa, de manera en que pueda visualizar de una manera gráfica y sencilla el recorrido de dichas rutas.

Ana encuentra con dificultad y empleando una interfaz poco visual, información de las rutas ofertadas por el parque natural. Además, con respecto a la predicción de tiempo que puede hacer para cuando quiera realizar la reserva y las aves que se puedan avistar no encuentra

información en el sitio web oficial. Para conocer estos datos se ve obligada a visitar numerosos sitios web externos con tal de tomar una decisión que conlleve el viajar a este parque natural o a cualquier otro, tal y como se ha comentado anteriormente.

Según el esquema tradicional planteado para los sitios web orientados al turismo, esta situación no tendría ninguna clase de solución.

Sin embargo, la aparición de nuevas tecnologías emergentes de información y comunicación (ICTs) suponen un fuerte impacto en el desarrollo de la gestión turística provocando una evolución de los servicios ofertados siguiendo el esquema tradicional anteriormente mencionado. Este cambio posee una influencia notable en el presente y se estima que alcanzará un mayor grado de relevancia en el futuro.

Según el estudio *Smart destinations and the evolution of ICTs* [2] esta situación ha generado un nuevo escenario para la administración de destinos turísticos. Dicho escenario provoca la aparición de nuevos modelos de gestión que realicen esta labor de manera eficiente. Entre ellos destaca la aparición de los destinos turísticos inteligentes o *smart tourism destination* (STD).

El nuevo paradigma se enfoca en la transformación de la información obtenida mediante medios físicos y digitales, por parte del destino turístico, en experiencias y propuestas de valor basadas en la eficiencia, la sostenibilidad y el enriquecimiento de la experiencia de usuario empleando para ello tecnologías avanzadas. Dentro de dichas tecnologías se identifican algunas que presentan técnicas sofisticadas como las redes neuronales además de otras más convencionales correspondientes a aplicaciones móviles.

Su objetivo es generar un acercamiento entre la gestión del turismo con la investigación y los avances tecnológicos. Sin embargo, este acercamiento presenta numerosas dificultades en el sector del turismo como es la inexistencia de un estándar global para el manejo de los datos que emplea esta nueva tecnología. Otro inconveniente reside en que, en numerosas ocasiones, los datos almacenados correspondientes al servicio que ofrecen los destinos turísticos son locales y específicos al destino en cuestión, dificultando con ello la creación de soluciones generales.

Con independencia de los inconvenientes anteriores, la aplicación de este modelo posee un auge en aumento conforme se van desarrollando y refinando estas nuevas tecnologías. Ejemplos correspondientes a la aplicación de dicho procedimiento se pueden observar en la implementación de determinados servicios como pueden ser la creación de una visita guiada

empleando realidad virtual o la generación de aplicaciones móviles que engloben determinados requisitos necesarios para el usuario final a la hora de utilizar un servicio turístico en concreto.

En el artículo antes mencionado se identifican 5 campos referentes a la gestión del turismo que pueden ser mejorados mediante esta metodología, destacando entre ellos el denominado *Destination intelligence*. Dicho punto hace alusión a la utilización de información abierta u *open data* relativa al resto de variables influyentes en la realización del turismo dentro de una aplicación móvil o del propio sitio web oficial.

Su finalidad es proporcionar al turista toda aquella información que precisa a la hora de efectuar esta actividad combinando información de sitios externos junto con los proporcionados por el propio destino turístico. Esto provoca una mejora cuantitativa en la comunicación entre el destino turístico y los usuarios finales que deriva en un aumento directo de la experiencia de usuario.

Por tanto, el objetivo de este Trabajo de Fin de Grado o TFG se basa precisamente, en el planteamiento anterior. Se busca realizar un aumento de las funcionalidades que presenta la página web oficial del parque natural de *Lagunas de la Mata y Torrevieja* empleando los denominados scripts de usuario o *user-scripts* [3]. Dicho sitio web puede ser encontrado accediendo al siguiente enlace: <https://parquesnaturales.gva.es/es/web/pn-lagunas-de-la-mata-torrevieja>.

Se espera conseguir efectuar un incremento positivo en la experiencia de usuario de manera en que encuentre toda aquella información que precisa para la realización de una reserva dentro del propio sitio web oficial. Además, el contenido y desarrollo de este TFG puede utilizarse a modo de inspiración por los destinos turísticos para determinar qué servicios adicionales pueden ser necesarios a la hora de llevar a cabo una reserva en un parque natural atendiendo a las 2 primeras fases del ciclo de viaje antes mencionadas.

En los siguientes apartados se detallará el marco teórico sobre el que se basa este proyecto, la arquitectura que tendrá este aumento de funcionalidades junto con los resultados obtenidos tras su aplicación en el sitio web.

2. Marco teórico

El término user-script hace referencia a un programa escrito por el usuario, generalmente utilizando el lenguaje *JavaScript* [4], con la finalidad de modificar páginas web para así aumentar sus funcionalidades. Estos scripts representan las diferentes técnicas desarrolladas en el lado del cliente empleadas para la aumentación web.

Este último término se adjudica a todos aquellos procedimientos que se realizan en la capa de presentación del sitio web con la finalidad de incluir ciertos requisitos o aspectos que no fueron considerados o efectuados por los desarrolladores del servicio. Esta característica provoca una mejora cuantitativa en la experiencia de usuario a la hora de utilizar dicha web. Su justificación se basa en la posibilidad de obtener información relevante adicional, entre otras consideraciones necesarias para realizar las gestiones que desea.

El uso y almacenamiento de estos scripts se realiza empleando *extensiones* [5] para el navegador web. De esta manera, gestionamos sin problema el número de user-scripts en ejecución junto con la información relativa a cuáles de ellos se encuentran activos en ese momento.

Se conoce con el término de “extensión” a módulos de software conformados usualmente por código fuente que se utilizan con el fin de personalizar un navegador web. Estas modificaciones admiten una gran variedad de campos que van desde cambios en la interfaz de usuario y bloqueo de anuncios hasta la administración de “cookies”. Dicha característica la comparten con los denominados complementos o *plug-ins* [6] del navegador.

Los plug-ins se corresponden con un tipo de módulo independiente dentro del navegador conformado por “código objeto”, es decir, *ejecutables* [7]. En la época actual su uso ha sido desaprobado por los navegadores, manteniendo el uso de las extensiones al alza llegando a existir hasta más de 100000 extensiones para el navegador *Google Chrome* [8]. Es debido a esta situación por la que empleamos extensiones para poder desarrollar y utilizar los user-scripts.

Con respecto a las numerosas extensiones que existen en la actualidad para gestión y desarrollo de user-scripts cabe decir que destaca una de ellas conocida como *Greasemonkey* [9], por ser pionera en el desarrollo de dichos scripts. Greasemonkey, creada para el navegador *Mozilla Firefox* [10], se corresponde con la primera extensión desarrollada con este

fin y fue creada por *Aaron Boodman* en el año 2004 y lanzada al público el 28 de marzo de 2005.

En menos de 2 meses de su lanzamiento, concretamente, en mayo de ese mismo año ya se habían desarrollado 60 scripts de usuario generales y 115 específicos de sitios web distribuidos para Greasemonkey. Este fue el inicio del auge que obtuvo el desarrollo y la utilización de esta nueva tecnología. En consecuencia se desarrolló “userscripts.org” fundada por *Britt Selvitelle* y otros miembros de la comunidad de Greasemonkey a finales del 2005.

Este sitio web, catalogado originalmente de código abierto, pasó a convertirse en el repositorio principal de user-scripts y llegando a acumular miles por año. Sin embargo, este servicio acabó tachándose de “descuidado” debido a los errores de mantenimiento del servidor que se generaron conforme iban acumulándose los user-scripts a lo largo de los años. Como resultado, se generaron otras alternativas para el almacenamiento de estos scripts como son “openuserjs.org” y “greasyfork.org”. Al final, en agosto de 2014, “userscripts.org” acabó cerrado definitivamente.

Cabe indicar que, los user-scripts almacenados en esta plataforma no desaparecieron sino que se mantuvieron en su página espejo “userscripts-mirror.org”, desde donde es posible encontrarlos en la actualidad.

El elevado uso de estos user-scripts también propiciaron la creación de otros manejadores de scripts de usuario para distintos navegadores. Entre ellos, se indican *Tampermonkey* [11], *Violentmonkey*, *Greasemonkey for Pale Moon*, *Firemonkey* y *USI*.

Destacamos entre las extensiones anteriores a Tampermonkey creada por *Jan Biniok* en mayo de 2010. Originalmente, se construyó como un script de usuario de Greasemonkey empaquetado para ser compatible con Google Chrome, pero pasó a ser una extensión independiente de este navegador. Actualmente, es uno de los manejadores de scripts de usuario más utilizados y una de las 13 extensiones de la “Chrome Web Store” que tiene al menos 10 millones de usuarios.

Cabe indicar que, como todas estas extensiones surgieron a partir de Greasemonkey presentan, en la mayoría de los casos, una compatibilidad con user-scripts desarrollados para esta extensión en determinadas versiones de la misma. Esto provoca que scripts desarrollados según las especificaciones de Greasemonkey puedan presentar cierta portabilidad a otras extensiones creadas.

La existencia de numerosas extensiones utilizables para gestionar los user-scripts propició un estudio entre los manejadores conocidos como Tampermonkey, Greasemonkey, Violentmonkey, Greasemonkey for Pale Moon, Firemonkey y USI mencionados anteriormente. Su finalidad consiste en determinar qué manejador utilizar para el desarrollo de este proyecto.

Mediante las siguientes referencias es posible acceder a los sitios web oficiales de dichas extensiones: *Tampermonkey - sitio oficial [12]*, *Greasemonkey - sitio oficial [13]*, *Violentmonkey - sitio oficial [14]*, *Greasemonkey for Pale Moon - sitio oficial [15]*, *Firemonkey - sitio oficial [16]* y *USI - sitio oficial [17]*.

2.1. Estudio de extensiones

Esta investigación comienza con el análisis de compatibilidad entre navegadores y versiones de Greasemonkey soportadas por los manejadores de scripts de usuario mencionados con anterioridad. Para ello, se elaboró una tabla en la que se observa para qué navegadores son compatibles y qué versiones de scripts realizados para Greasemonkey soportan:

Tabla 1. Compatibilidad entre extensiones

Extensión	Navegadores Compatibles	GM soportado
Tampermonkey	Chrome, Opera, Safari, Microsoft Edge, Firefox (con soporte para el navegador para móviles Dolphin y para el navegador UC)	GM 3 y GM 4 user-scripts
Greasemonkey	Firefox	GM 4 user-scripts
Violentmonkey	Chrome, Firefox, Maxthon, Opera	GM 3 y GM 4 user-scripts
Greasemonkey for Pale Moon	Pale Moon	GM 3 user-scripts
Firemonkey	Firefox	GM 4 user-scripts y algunos GM 3
USI	Firefox	GM 3 user-scripts

(Fuente: Awesome Userscripts [18])

Se indica que para aquellos que utilicen el navegador *Internet Explorer* [19] no disponen de ningún plug-in o complemento en particular para poder interpretar los user-scripts, sin embargo, poseen la extensión llamada “*Adguard* [20]” que les permite lograr este mismo fin.

Tras comprobar la información mostrada en la tabla anterior junto con la popularidad de las extensiones analizadas, destacamos que las más famosas son Tampermonkey, Greasemonkey y Violentmonkey.

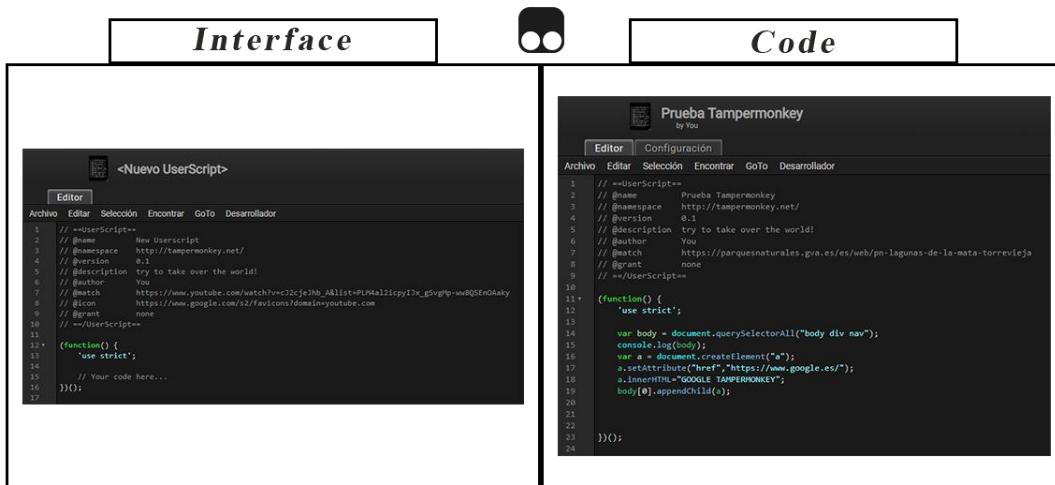
Por este motivo, el siguiente paso en nuestro estudio se basa en realizar un simple user-script en estas extensiones para comprobar el funcionamiento de estos manejadores de scripts de usuario. Su funcionalidad consiste en incluir un simple enlace a Google en la página web del parque natural.

Realización de los user-scripts iniciales

Estos scripts emplearán las mismas instrucciones para generar un enlace al navegador de Google pero siguiendo la interfaz proporcionada por cada extensión.

Comenzaremos mostrando la interfaz de Tampermonkey junto con el código generado y el resultado final en el sitio web.

Figura 1. Interfaz de Tampermonkey



(Fuente: Logotipo Tampermonkey [1])

En la figura anterior se observa la interfaz expuesta por Tampermonkey cuando seleccionamos la opción de crear un nuevo script desde el manejador de la extensión. Según puede observarse, incluye un gran número de cabeceras ya expuestas para definir las características

del user-script que se va a crear. Además, incluye también una sección denominada “function()” en la que se alojará todo el código a desarrollar.

También, cabe decir que posee un detector de errores a tiempo real en el que Tampermonkey te avisa de posibles errores en el código realizado. Evitando así una comprobación forzosa directamente desde su prueba directa en el sitio web. Gracias a esta característica, se puede optimizar el tiempo de desarrollo a la hora de realizar proyectos con esta extensión. Además, se destaca que se ha empleado el modo oscuro en esta extensión en particular. El resto emplean su modo de estilo por defecto.

Con respecto al código empleado para generar el enlace hacia el navegador de Google cabe indicar que, simplemente, se ha utilizado el *DOM* [21] junto con el lenguaje JavaScript para generar una etiqueta HTML “”. Dicha etiqueta, se muestra en la barra de navegación del sitio web con el estilo predefinido por la página para caracterizar estos elementos. El resto de extensiones siguen, tal y como se ha indicado anteriormente, estas mismas especificaciones en el código.

Ahora pasamos a analizar la interfaz de Greasemonkey una vez elegimos crear un nuevo script de usuario.

Figura 2. Interfaz de Greasemonkey



(Fuente: Logotipo Greasemonkey [2])

En este caso, la interfaz es mucho menos específica y más rudimentaria, tal vez debido a que fue la primera extensión en generarse para el desarrollo de user-scripts. En este caso, hay

muchas cabeceras que debes incluir de manera manual para poder ejecutar el script en los sitios web que deseas. Además, tampoco posee un detector de código, por lo que la única manera de comprobar errores en el mismo es ejecutándolo directamente en el sitio web en cuestión.

Una vez mostrada la interfaz de Greasemonkey pasamos a mostrar la de Violentmonkey.

Figura 3. Interfaz de Violentmonkey

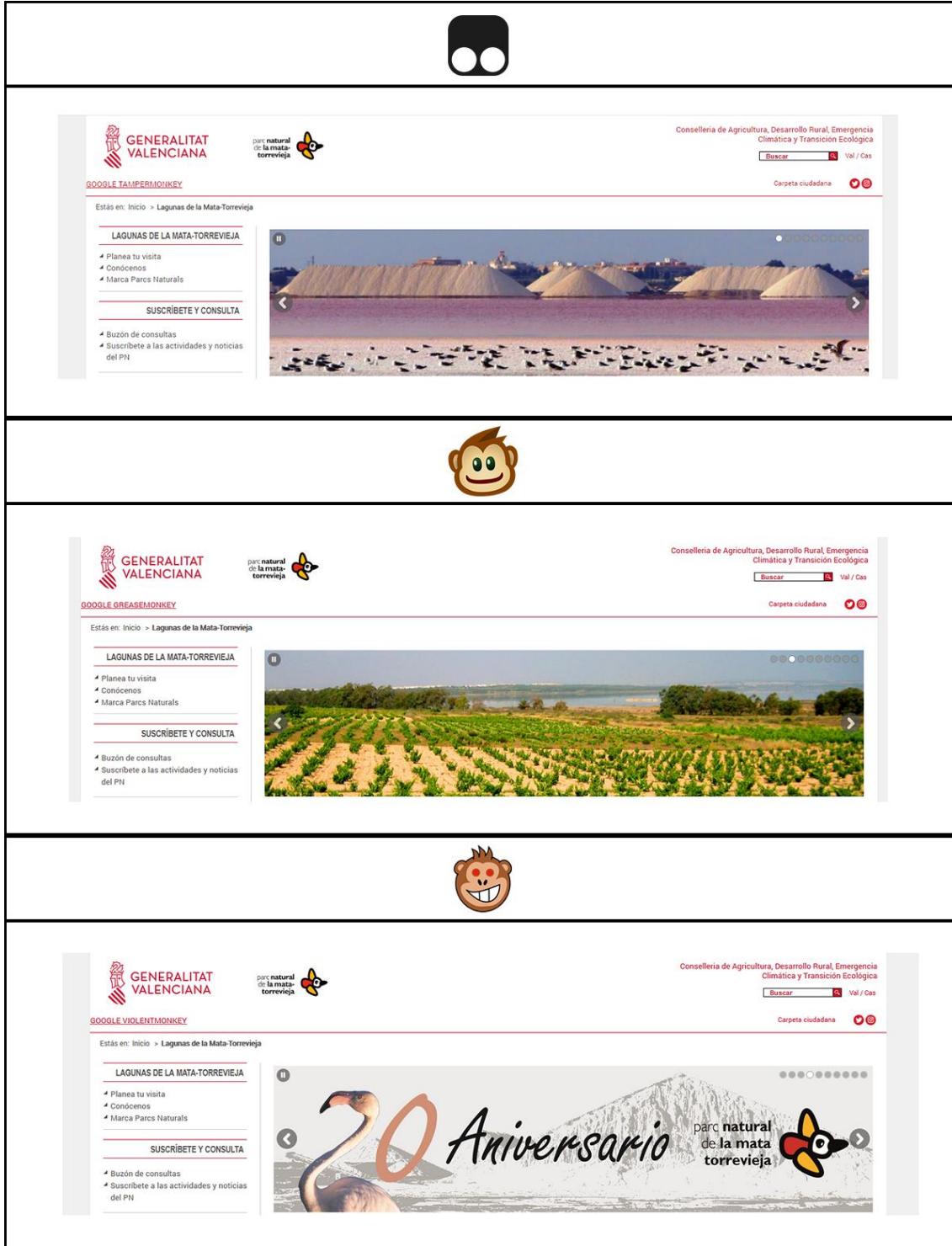


(Fuente: Logotipo Violentmonkey [3])

Esta interfaz presenta un aspecto un poco más elaborado que la extensión anterior, aunque sigue siendo menos específica que la de Tampermonkey. En este caso, sí que incluye las cabeceras generales necesarias para poder ejecutar el user-script en el sitio web en cuestión. Al igual que en Greasemonkey esta extensión tampoco posee un detector de errores en el código, dificultando con ello el desarrollo del mismo.

Tras realizar el análisis de las 3 interfaces expuestas por las extensiones pasamos a mostrar el resultado final de este user-script básico en el sitio web del parque natural.

Figura 4. Resultado final según las 3 extensiones



(Fuente: Logotipo Tampermonkey [1], Logotipo Greasemonkey [2], Logotipo Violentmonkey [3])

Tal y como puede observarse, el resultado final en el sitio web empleando las 3 extensiones es exactamente el mismo debido a que es simplemente código JavaScript ejecutado a nivel de usuario. La diferencia radica en la usabilidad de las extensiones a la hora de desarrollar el código necesario para llegar a dicho resultado.

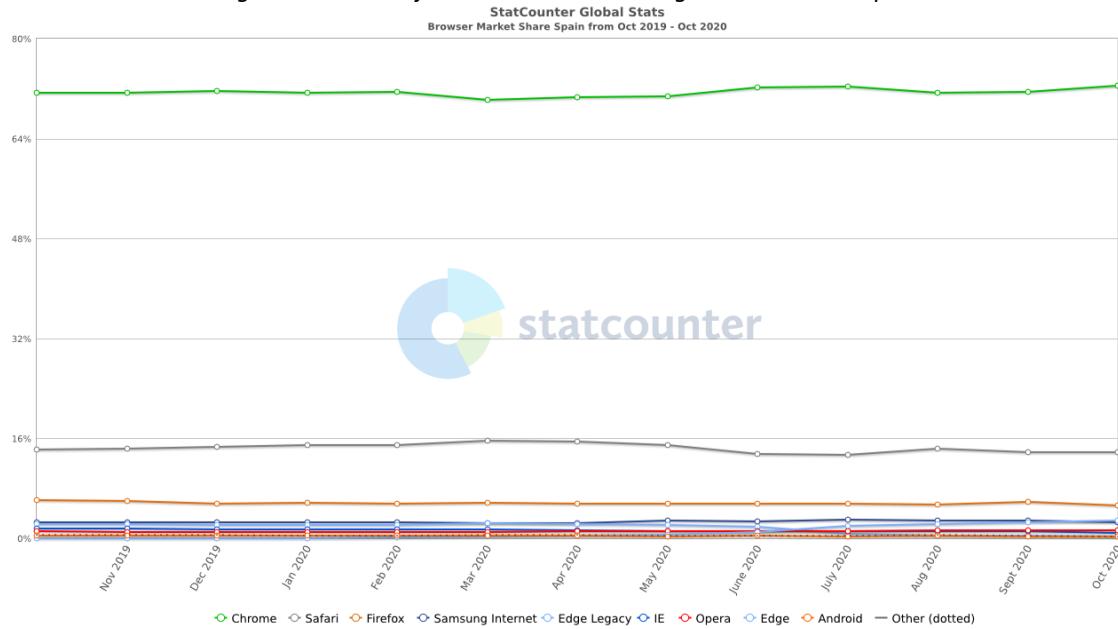
En consecuencia, se determina que Tampermonkey presenta la mejor interfaz al poseer un aspecto más ameno y una serie de funcionalidades añadidas como el detector de errores de código que provocan un aumento de rendimiento a la hora de alcanzar un determinado resultado. Al ser una extensión desarrollada para ser nativa de Google Chrome, ha tomado sus máximas de usabilidad creando esta interfaz con una mayor facilidad de uso comparada con la del resto de gestores de user-scripts.

Una vez estudiada la interfaz y el resultado final que poseen las extensiones anteriores, realizamos un análisis sobre el porcentaje de utilización de los navegadores web existentes más relevantes en la actualidad.

Análisis sobre la utilización de navegadores web

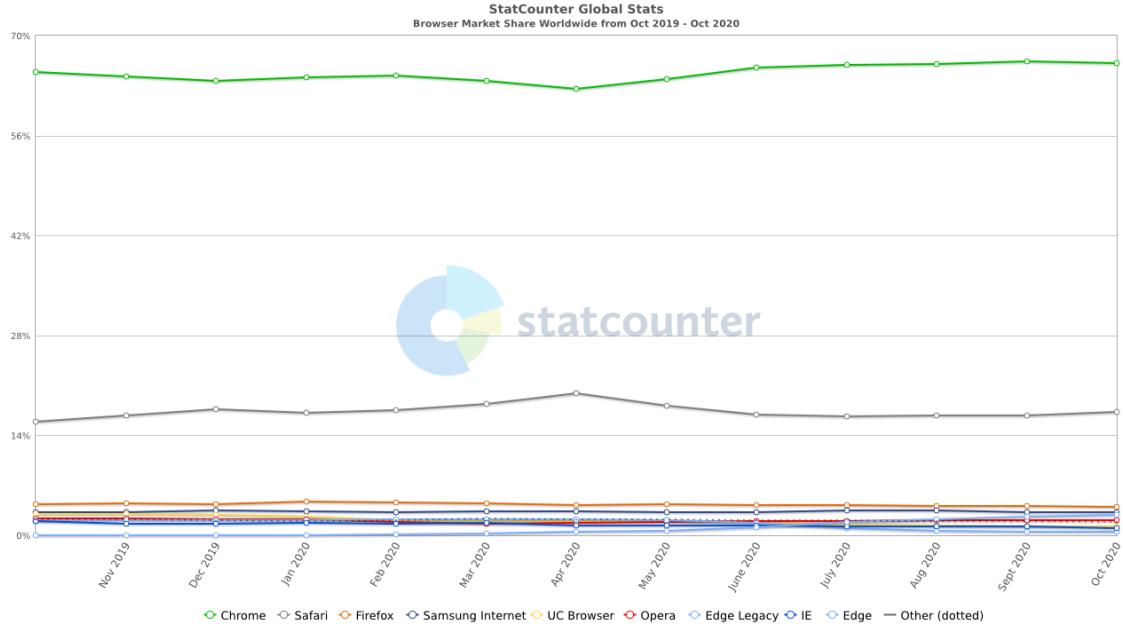
Para esta investigación, empleamos el sitio web *statcounter* [22] para poder determinar el porcentaje de utilización de los distintos navegadores web más conocidos. Este estudio se realizará tanto para el caso particular de España como globalmente.

Figura 5. Porcentaje de utilización de navegadores web en España



(Fuente: statcounter [22])

Figura 6. Porcentaje de utilización de navegadores web a nivel global



(Fuente: statcounter [22])

Tal y como se observa en las estadísticas sobre la utilización de los navegadores, los más utilizados en la actualidad son:

Tabla 2. Porcentaje de uso de navegadores web

%	Google Chrome	Safari	Firefox	Samsung Internet	Edge	Opera
España	72.48%	13.77%	5.23%	2.56%	2.87%	1.33%
Mundo	66.12%	17.24%	3.98%	3.18%	2.85%	2.08%

(Fuente: statcounter [22])

Por lo que se deduce que Google Chrome es el navegador por excelencia según los datos recopilados desde Octubre de 2019 hasta Octubre de 2020 tanto en España como en todo el mundo.

Elección de extensión

Estudiando la interfaz y los resultados presentados por cada uno de los manejadores de scripts de usuario junto con el análisis sobre el navegador web más utilizado, se determina como extensión elegida la denominada Tampermonkey. Dicho manejador es nativo de este mismo navegador como se ha comentado anteriormente y además, observando todas sus compatibilidades tanto con Greasemonkey como con el resto de navegadores, es la extensión más utilizable en una mayor variedad de navegadores de internet.

3. Objetivos

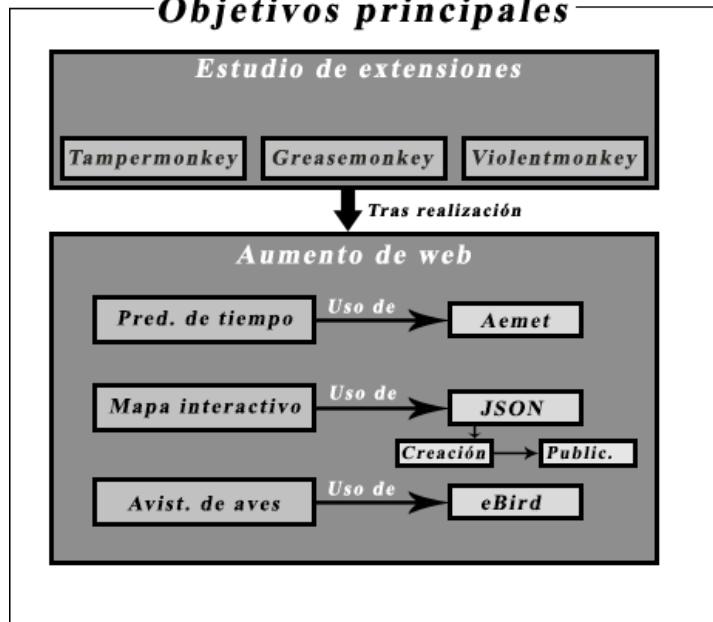
Para desarrollar este TFG se han estimado diferentes objetivos correspondientes a las características que posee el aumento de web realizado. Para ello, se ha tenido en cuenta la cantidad de horas estimadas para un TFG (150h) junto con la investigación referente a las tecnologías utilizadas con tal de seguir el modelo de creación de destinos turísticos inteligentes. Dichos objetivos se especifican de manera general atendiendo a las siguientes indicaciones.

3.1. Objetivos principales

En esta sección se detallan los objetivos esenciales para la elaboración de este proyecto.

- Estudio sobre las diferentes extensiones que existen para realizar aumentación web mediante scripts de usuario.
- Realización de un aumento de web correspondiente a la predicción de tiempo en el parque natural.
 - Empleo de *Aemet* [23] para conseguir estos datos de predicción de tiempo.
- Elaboración de un plugin que detalle toda la información relativa a las rutas y paradas del parque natural en un mapa interactivo.
 - Creación de un archivo *JSON* [24] propio que englobe toda esta información separando la capa de datos de la capa de implementación.
 - Publicación de este archivo en una plataforma de almacenamiento digital online con tal de emplear dicha información en la realización del mapa interactivo.
- Desarrollo de un user-script que especifica, en forma de tabla, las aves avistadas en el parque natural en una fecha concreta seleccionada por el usuario.
 - Utilización de *eBird* [25] para la obtención de datos sobre avistamiento de aves.

Figura 7. Objetivos principales



(Fuente: Elaboración propia)

3.2. Objetivos secundarios

En este apartado se especificarán aquellos objetivos que se consideran importantes para un buen acabado en la elaboración del aumento web pero que no son imprescindibles para su desarrollo funcional.

- Creación de interfaces de usuario que faciliten el uso de los diferentes plugins.
- Muestra de datos considerando las propiedades de usabilidad y accesibilidad.
- Adaptación de la información mostrada por los user-scripts a dispositivos móviles empleando directivas *responsive* [26].
- Empleo de librerías y recursos con licencias *Creative Commons* [27] o *MIT* [28] que no posean copyright para la posible publicación del proyecto.

Figura 8. Objetivos secundarios

Objetivos secundarios

Interfaz y muestra de datos

Sigue las directrices:

Usabilidad

Accesibilidad

Responsive

Facilidad de uso

Licencias libres

(Fuente: Elaboración propia)

4. Metodología

Con tal de desarrollar este proyecto se han empleado tecnologías que facilitan la división de tareas y la gestión del código generado. Una de dichas tecnologías es *Trello* [29], mediante el cual ha sido posible planificar cada una de las tareas que ha conllevado la elaboración de este TFG.

Se realizó un tablero en el que se dividieron, por orden de prioridad, las tareas a realizar. Dichas tareas pueden contener listas que representan objetivos a alcanzar antes de considerar dicha tarea como finalizada. La división de este proyecto se efectuó atendiendo a los siguientes apartados:

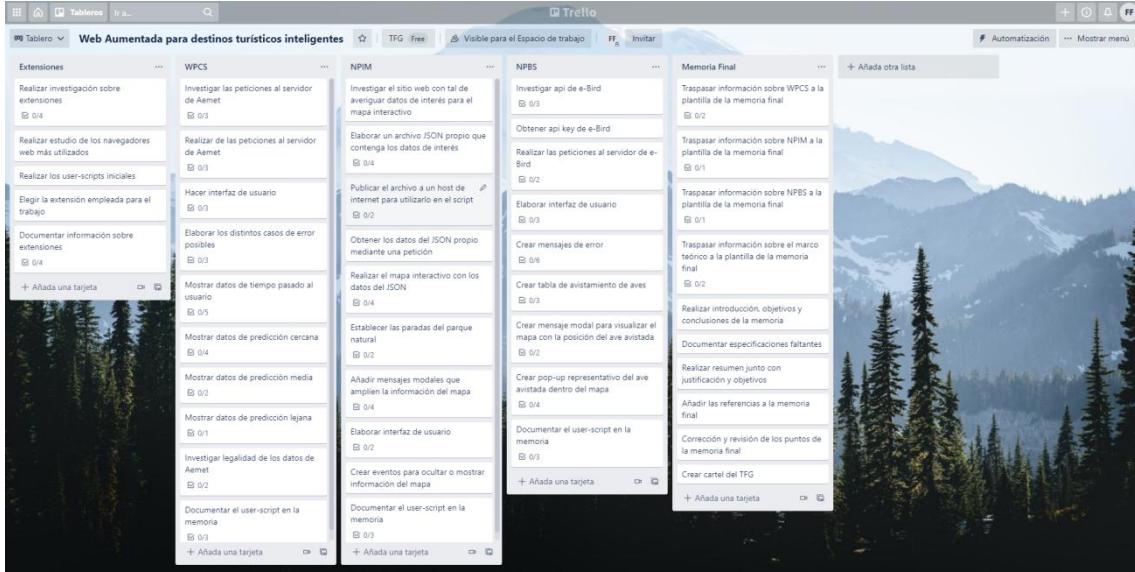
1. Realizar investigación sobre extensiones.
2. Realizar el script correspondiente a la predicción del tiempo.
3. Realizar el mapa interactivo.
4. Realizar el script que muestra la tabla de avistamiento de aves en el parque natural.
5. Realizar la memoria final.

Estos puntos se completaron en orden descendente hasta conseguir el resultado final de este proyecto. Como se ha comentado, los apartados anteriores poseen tareas con el fin de finalizar los objetivos propuestos para el desarrollo del TFG. Dichas tareas se efectúan también por orden descendente según el tablero de Trello de manera que, la última tarea en el desarrollo de cada punto (a excepción de realizar la memoria final), se corresponde con su documentación pertinente en dicha memoria.

Cuando se completa una tarea se archiva del tablero de Trello con el fin de poder evaluar de una manera visual la evolución del desarrollo correspondiente al proyecto.

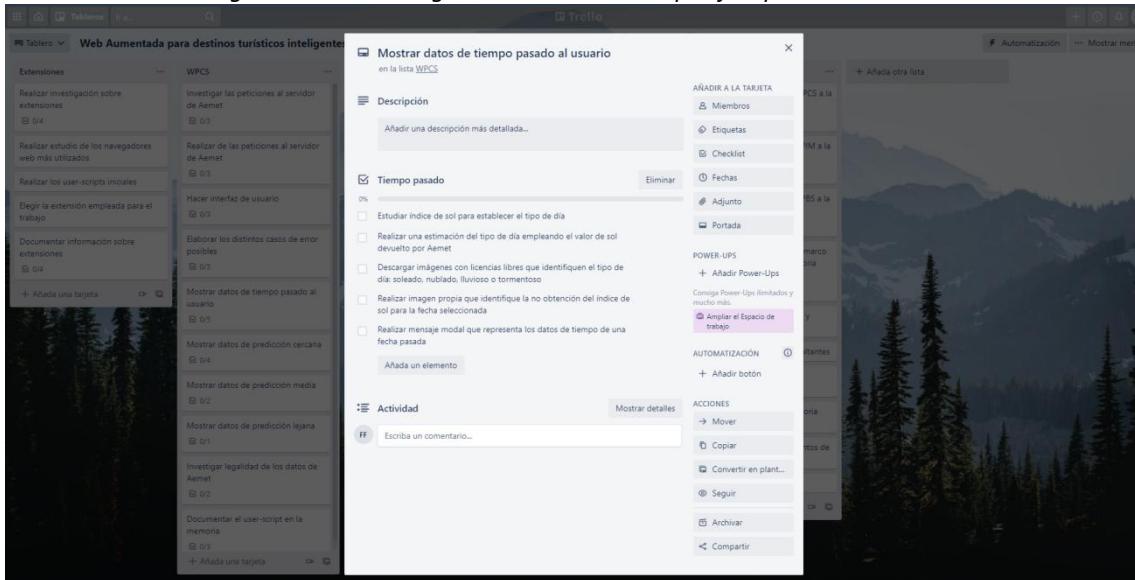
De esta manera, se evita la omisión de información relevante con respecto a cada uno de los apartados que conforman el desarrollo del trabajo. En las siguientes figuras se muestra el tablero de Trello empleado junto con un ejemplo de tareas necesarias para completar un apartado en concreto dentro de la elaboración de este TFG.

Figura 9. Metodología-Tablón de Trello general



(Fuente: Elaboración propia)

Figura 10. Metodología-Tablón de Trello específico para las tareas



(Fuente: Elaboración propia)

También se ha precisado del uso de una plataforma que realizara el control de versiones de los diferentes user-scripts creados para la elaboración del proyecto. Se decidió emplear la herramienta *GitHub* [30] con esta finalidad. Gracias a GitHub, se han podido establecer determinadas copias de seguridad además de otorgar un control del código elaborado durante el desarrollo del TFG.

Además, el código generado mediante el cual se ha creado cada user-script ha sido publicado en la plataforma bajo licencia Creative Commons para su libre descarga y utilización según las especificaciones de la licencia CC. Los scripts de usuario se pueden encontrar accediendo al siguiente enlace: <https://GitHub.com/FranciscoJavierGF/TFG>.

Una vez expuesta la metodología de desarrollo utilizada se detalla en el siguiente punto la arquitectura que seguirá la elaboración de este proyecto.

5. Arquitectura del trabajo

Para cumplir con todos los objetivos propuestos para la realización del aumento web se ha dividido su elaboración en módulos independientes. Es decir, el desarrollo de este TFG es el resultado de la creación de 3 user-scripts que realizan las funcionalidades marcadas por los objetivos detallados anteriormente.

Se ha decidido establecer esta separación con el objetivo de otorgar al usuario la posibilidad de elegir que funcionalidades quiere ejecutar en cada momento. Esto permite aligerar la sobrecarga del sitio web prescindiendo de aquellas operaciones que no se encuentran en los user-scripts activos. Además, también posibilita una mejor organización en el desarrollo del proyecto al dividir los objetivos específicos en scripts de usuario independientes.

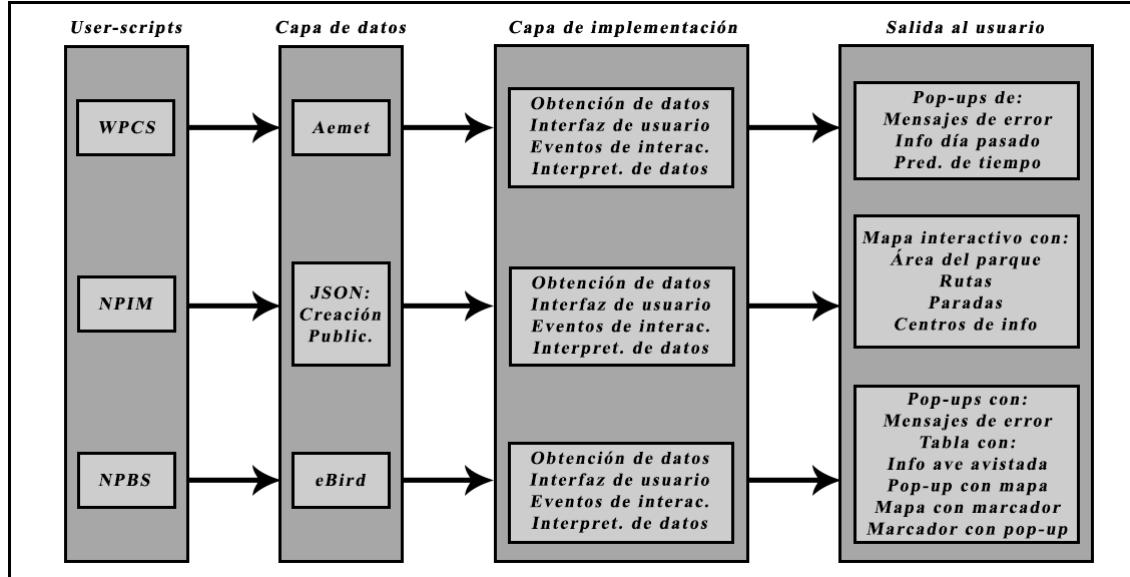
Los siguientes capítulos de la memoria explican la elaboración de cada módulo en particular junto con todas sus características semejantes e independientes. Dicha especificación se detalla según los siguientes apartados:

1. **Weather Prediction Calendar Script (WPCS):** Explica el desarrollo del user-script generado para realizar las peticiones de predicción de tiempo empleando Aemet. En sus apartados se detalla la obtención de la key necesaria para realizar las peticiones en esta tecnología junto con el desarrollo completo del script. Esta elaboración especifica las tecnologías utilizadas, la creación de dichas peticiones junto con su interpretación, la interfaz de usuario, la muestra de datos, los problemas encontrados, las comparaciones en el sitio web y las pruebas realizadas.
2. **Natural Park Interactive Map (NPIM):** En este capítulo se detalla la elaboración del mapa interactivo con los datos sobre las rutas, paradas, centros de información, entre otros, presentes en el parque natural. Al igual que en el caso anterior, se explica su desarrollo completo atendiendo a las mismas especificaciones con la particularidad de que, en este caso, se ha desarrollado un archivo JSON propio que aloja estos datos presentes en la web. Por tanto, se indica como se ha creado dicho archivo, su publicación y la petición realizada para obtenerlo dentro del user-script.
3. **Natural Park Bird Sighting (NPBS):** En este caso se especifica el script correspondiente a la generación de la tabla de avistamiento de aves en el parque natural utilizando eBird. Como en los apartados anteriores, se detalla dentro de este capítulo su desarrollo completo junto con la obtención de la key correspondiente a este api y las llamadas a realizar según dicha tecnología.

4. **Conclusiones y trabajo futuro:** Una vez expuesto detalladamente el desarrollo de todos los módulos independientes se explican las conclusiones obtenidas tras la elaboración de este proyecto junto con el trabajo futuro a realizar.

En la siguiente figura se muestra de manera general la arquitectura realizada.

Figura 11. Arquitectura general de los user-scripts



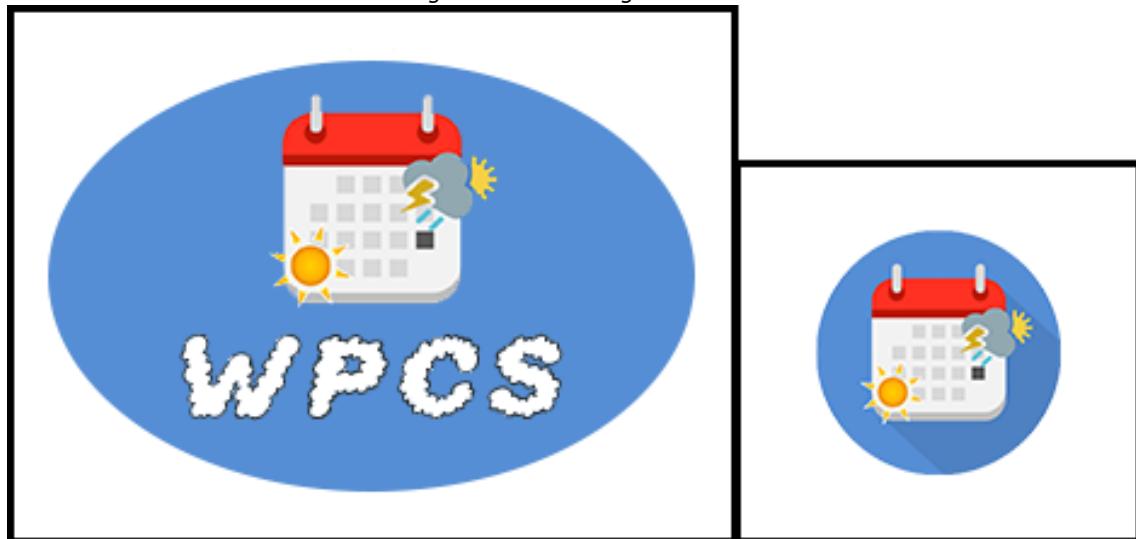
(Fuente: Elaboración propia)

6. Weather Prediction Calendar Script (WPCS)

Este script surge de la necesidad del usuario de conocer la predicción de tiempo que habrá cuando desee realizar una reserva en el parque natural. Dicha información no puede ser obtenida desde el sitio web de manera natural por lo que el usuario tendrá que navegar por diferentes servicios de internet para poder obtener esta información. Mediante el aumento de web, esta acción ya no es necesaria debido a que se emplea la información proporcionada por el *API de Aemet* [31] para mostrar esta información en el calendario de la página principal del parque natural.

En los siguientes apartados se detallará el desarrollo completo de este user-script.

Figura 12. WPCS-Logo e icono



(Fuente: AlphaClouds [4], Icono de Calendario [5], Icono de Parcialmente nublado [6], Icono de Sol [7])

6.1. Tecnologías utilizadas

En el desarrollo de este user-script se han empleado librerías JavaScript que permiten la realización de características necesarias para el correcto funcionamiento del mismo. Entre ellas, se indica una que permite crear mensajes modales o *pop-ups* [32] con un aspecto agradable para el usuario. También, se precisa de otra librería que permite el desarrollo de gráficos con el fin de mostrar los datos de predicción de tiempo de una manera óptima.

Además, es necesario el tratamiento de coordenadas geográficas por lo que se ha empleado otra librería para suplir este requerimiento. A continuación, se especifican dichas librerías:

SweetAlert2 [33]: Se trata de una librería javascript que permite el desarrollo de mensajes modales o pop-ups con un estilo agradable y accesible para el usuario. Posee bastantes opciones de personalización incluyendo la característica responsive que permite la adaptación de estos pop-ups a dispositivos móviles. Presenta licencia MIT, por lo que se la cataloga como de código abierto u *open source* [34].

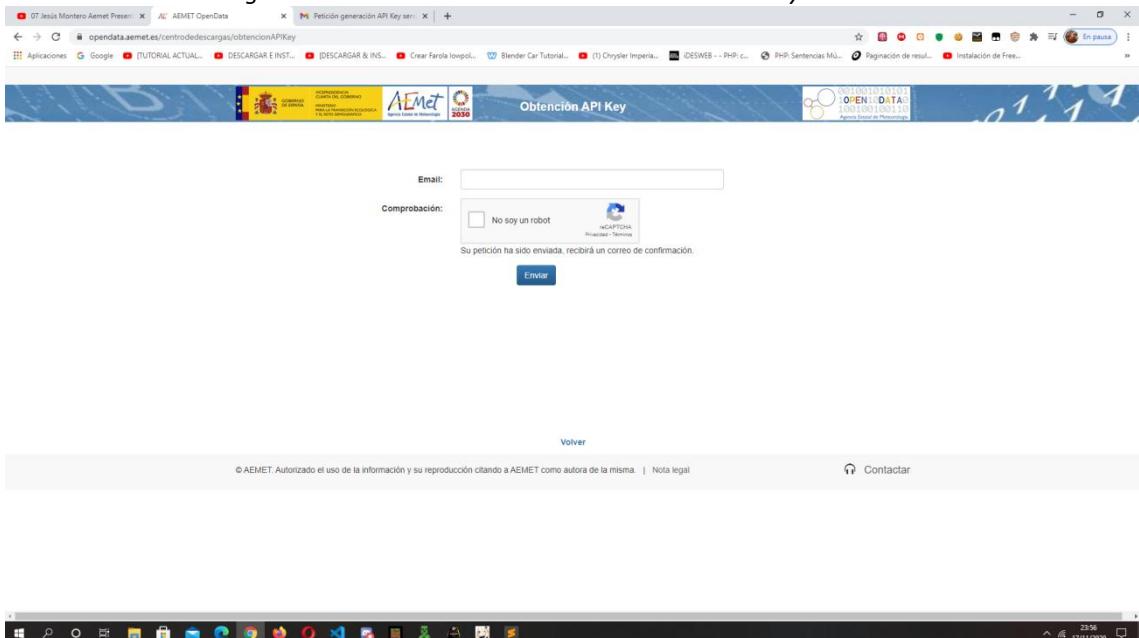
Chart.js [35]: Es una librería javascript que permite la creación de diversos tipos de gráficos para la muestra de datos empleando para ello la etiqueta “<canvas></canvas>” propia de HTML5. Se pueden elaborar hasta 8 tipos de gráficos diferentes, todos ellos animados, personalizables y responsive. Dichos tipos son: gráfico de líneas, de barras, circular, de área, de radar, de burbuja, polar y de dispersión. Alojada bajo licencia MIT, también se cataloga como open source.

Dms.js [36]: Se corresponde con un módulo dentro de la librería *Geodesy* [37]. Dicha librería dividida en bloques, se utiliza para realizar una gran variedad de cálculos geográficos, conversiones entre sistemas de coordenadas y funciones de mapeo. El módulo dms.js se encarga, en concreto, de proporcionar funciones para el análisis y la representación de grados, minutos y segundos. Además, dicho módulo presenta licencia MIT, por lo que es catalogado como open source.

6.2. Obtención del API key de Aemet

Como paso previo a la implementación de WPCS, se debe de obtener una key por parte del API de Aemet para poder acceder a sus datos de manera gratuita. Para ello, se accedió a la página de Aemet (<https://opendata.aemet.es/>) y se clicó sobre la opción de obtener key. Para obtener una key tendremos que poner nuestro correo electrónico y darle a “no soy un robot” para completar el capcha y pulsamos el botón de enviar.

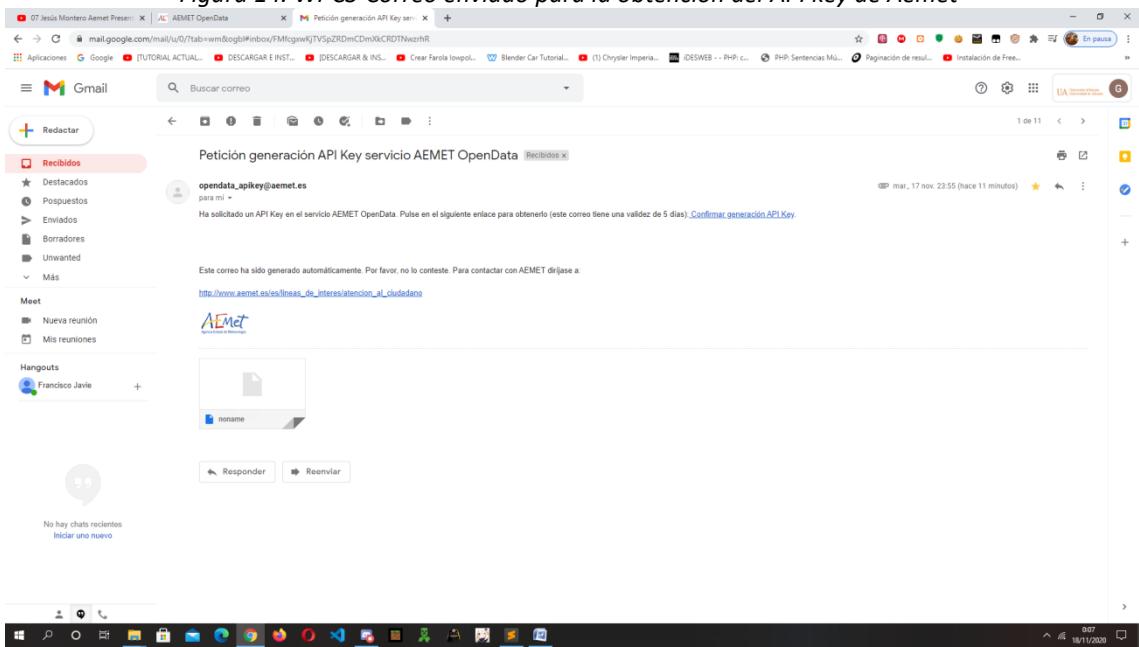
Figura 13. WPCS-Formulario de obtención del API key de Aemet



(Fuente: Elaboración propia)

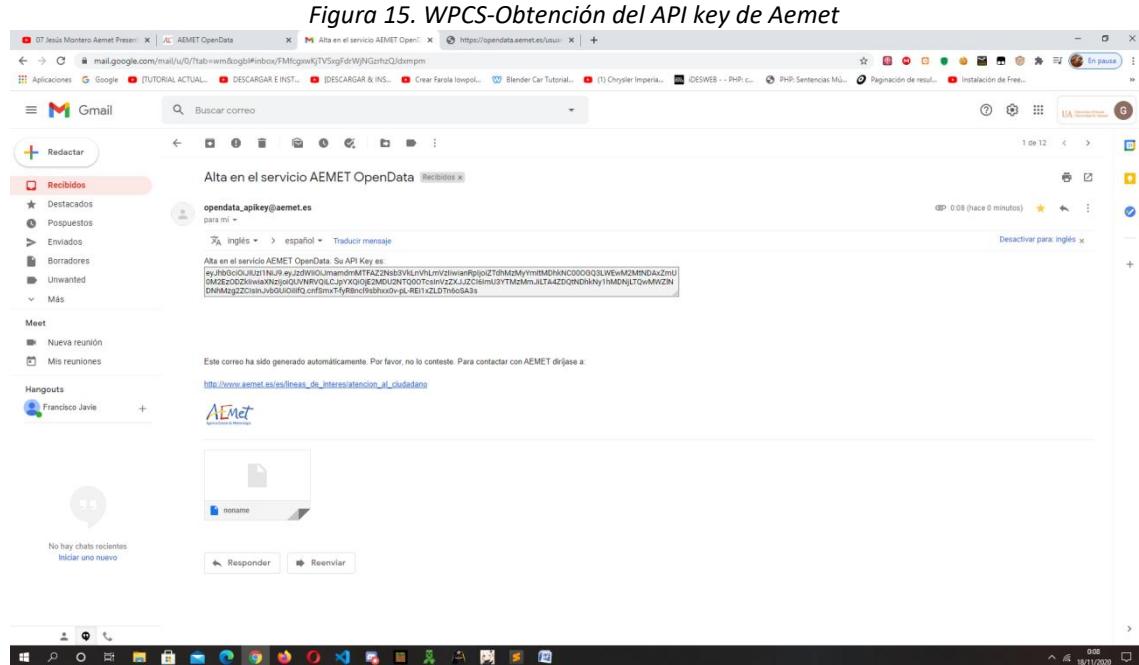
Nos llegará el siguiente correo a nuestro email:

Figura 14. WPCS-Correo enviado para la obtención del API key de Aemet



(Fuente: Elaboración propia)

Clicamos en obtención de API key y obtenemos así nuestra key, la cual la recibimos mediante otro correo electrónico con el API key que utilizaremos:



(Fuente: Elaboración propia)

Cabe destacar que tenemos hasta un plazo de 5 días para poder obtenerla, sin embargo, una vez obtengamos la key, esta la podremos utilizar siempre que queramos debido a que no tiene fecha de caducidad. Con esta key podremos hacer las llamadas al API de Aemet para poder obtener los datos de predicción de tiempo que necesitamos.

6.3. Desarrollo

Lo primero que hacemos a la hora de desarrollar un user-script es definir en su cabecera los recursos que va a necesitar, las URL en las que se va a ejecutar, el nombre que va a tener, su descripción, etc. A continuación, se muestra una tabla con los nombres de las etiquetas de Tampermonkey utilizadas en el script y con qué objetivo se utilizan.

Tabla 3. Etiquetas básicas presentes en la cabecera de un user-script

Descripción	
name	El nombre del script.
Namespace	El espacio de nombres al que pertenece el script.
Version	La versión actual del script. Utilizado para el control de actualizaciones.
Description	Descripción breve de lo que hace el script.
Author	El autor del script.
Icon	El icono que identificará al script y que aparecerá en la gestión de user-scripts de Tampermonkey.
Match	URL's en donde se ejecutará el script.
Require	Librerías javascript que se necesitan para que el script funcione. Se cargan antes de que se cargue el script, y si falta alguna, el script falla en su ejecución, por lo que con esto se definen dependencias.
Grant	Se utiliza para introducir determinadas funciones GM que proporciona Tampermonkey con el fin de añadir características potentes al script en la lista blanca, para que así dichas características se ejecuten correctamente. Si no se especifica, estas funciones GM se detectan automáticamente y si se especifica como none, ninguna de estas se podrá utilizar en el script ya que no tendrán permisos para ello.

(Fuente: Documentación de Tampermonkey [38])

A continuación, se muestra una imagen en donde se visualizan estas etiquetas en el header de WPCS.

Figura 16. WPCS-Cabecera

```

1 // ==UserScript==
2 // @name Weather Prediction Calendar Script
3 // @namespace http://tampermonkey.net/
4 // @version 1.0
5 // @description Script que recopila los datos de Aemet y muestra en el calendario situado en la página de index de lagunas de la mata en torrevieja
6 // @author Francisco Javier García Fernández
7 // @icon https://i.ibb.co/dtv3qf1/time-prediction-calendar-script-icon.png
8 // @match https://parquesnaturales.gva.es/es/web/pn-lagunas-de-la-mata-torrevieja
9 // @match https://parquesnaturales.gva.es/es/web/pn-lagunas-de-la-mata-torrevieja/lagunas-de-la-mata-torrevieja*
10 // @require https://cdn.jsdelivr.net/npm/geodesy@2/dms.js
11 // @require https://cdn.jsdelivr.net/npm/sweetalert2@10
12 // @require https://cdnjs.cloudflare.com/ajax/libs/Chart.js/2.9.4/Chart.js
13 // @grant none
14 // ==/UserScript==
```

(Fuente: Elaboración propia)

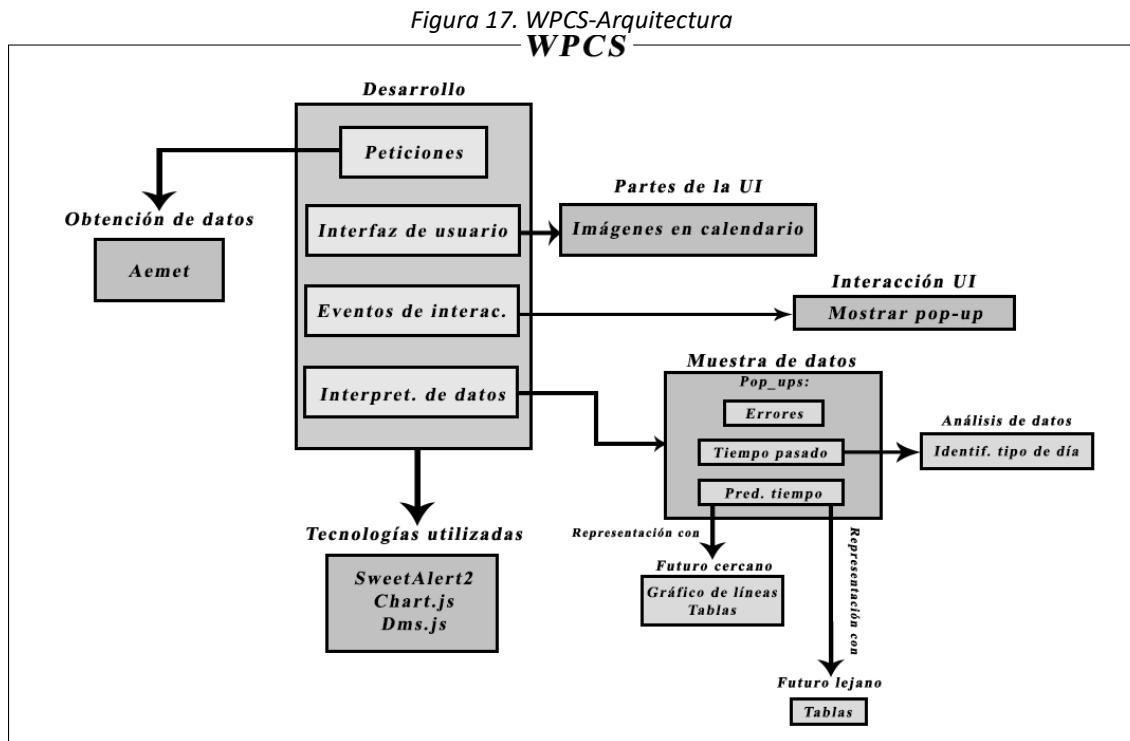
Una vez definida la cabecera del script pasamos a exponer su implementación. Para ello, se divide su desarrollo en 4 partes diferenciadas:

1. Obtención de los datos del municipio de Torrevieja y de su estación Aemet más cercana.
2. Creación de la interfaz de usuario.

3. Creación de las peticiones de tiempo pasado y futuro a Aemet
4. Muestra de datos al usuario.

En los siguientes apartados se detallarán cada uno de los puntos anteriores.

Además, se proporciona una figura que detalla la arquitectura mediante la cual se ha desarrollado este user-script. Dicha figura puede observarse a continuación.



(Fuente: Elaboración propia)

6.3.1. Datos de Torrevieja y de su estación más cercana

Con tal de extraer los datos de Torrevieja y de su estación más cercana empleando latitud y longitud obtenemos los datos del municipio realizando una consulta al API de Aemet. Dicha consulta devuelve la totalidad de municipios de los que Aemet recopila información.

Cabe destacar que esta es la petición que más tiempo tarda en realizarse debido al formato en el que se devuelven los datos. Aemet, en este caso particular, devuelve todos los datos directamente en formato JSON y no mediante una URL en la que se encuentran los datos (situación que sucede con el resto de peticiones realizadas a este API).

Como debe devolver muchos datos se precisa de un determinado número de segundos para obtener el resultado de esta consulta. Número que varía dependiendo de la conexión a

internet que se tenga y del número de peticiones que se estén realizando al API en ese momento.

Una vez obtenidos todos los municipios realizamos una búsqueda por nombre empleando la cadena de texto “torrevieja”. Tras ello, almacenamos todos los datos relativos a dicho municipio en una variable global. Cabe destacar que se ha realizado una *clase en javascript* [39] denominada “Municipio” la cual almacenará toda la información detallada en este punto de desarrollo del script y que se observa en las siguientes figuras.

Figura 18. WPCS-Clase Municipio y variable global

```
// Datos del municipio de Torrevieja obtenidos mediante Aemet
class Municipio{
    constructor(id, data){
        this.id = id;
        this.data = data;
        this.estMasCercana = null;
        this.distanceToEstMasCercana = null;
    }
}

var torrevieja = null; // Municipio sobre los que se basan los datos
```

(Fuente: Elaboración propia)

Figura 19. WPCS- Obtención de los datos del municipio de Torrevieja

```
function findMunTorreviejaData(municipios){
    var munData = null;
    for(var key in municipios){
        var actMun = (municipios[key].nombre).toLowerCase();
        if(actMun == "torrevieja"){
            munData = municipios[key];
            break;
        }
    }
    console.log(munData);
    return munData;
}

function getMunTorreviejaData(){
    var xhr = new XMLHttpRequest();
    xhr.withCredentials = true;
    xhr.addEventListener("readystatechange", function () {
        if (this.readyState === XMLHttpRequest.DONE) {
            if(this.status === 0 || (this.status >= 200 && this.status < 400)){
                var data = JSON.parse(this.responseText);
                var torrData = findMunTorreviejaData(data);
                var torrDataID = (torrData.id).slice(2);
                torrevieja = new Municipio(torrDataID, torrData);
                console.log(torrDataID);
                getEstacionMasCercanaData();
            }
            else{
                showServerErrorToUser();
            }
        }
    });
    xhr.open("GET", "https://opendata.aemet.es/opendata/api/maestro/municipios?api_key=" + aemetKey);
    xhr.setRequestHeader("cache-control", "no-cache");
    xhr.send();
}
```

(Fuente: Elaboración propia)

Una vez obtenidos los datos de Torrevieja, creamos el objeto municipio pasándole al constructor los datos del municipio y su ID, la cual obtenemos previamente a través de los datos de Torrevieja. Una vez creamos la variable global llamada torrevieja como un objeto de tipo Municipio, pasamos a conseguir los datos de su estación Aemet más cercana para así completar todos los datos de dicho objeto.

Para ello, empleamos las coordenadas GPS del municipio, determinadas por su latitud y longitud. Estas coordenadas vienen especificadas por un formato denominado DMS por lo que para realizar esta comparativa, empleamos la librería *Dms.js* antes comentada. Almacenaremos los datos de la estación que se encuentra a una menor distancia geográfica del municipio de Torrevieja.

En las siguientes figuras se representa tanto la petición que se tiene que realizar a Aemet para obtener todas las estaciones como la obtención de dicha estación más cercana.

Figura 20. WPCS-Obtención de todas las estaciones Aemet

```
function getEstacionMasCercanaUrl(urlData){  
    var xhr = new XMLHttpRequest();  
    xhr.withCredentials = true;  
    xhr.addEventListener("readystatechange", function () {  
        if (this.readyState === XMLHttpRequest.DONE) {  
            if(this.status === 0 || (this.status >= 200 && this.status < 400)){  
                var data = JSON.parse(this.responseText);  
                console.log(data);  
                findEstacionMasCercana(data);  
            }  
            else{  
                showServerErrorToUser();  
            }  
        }  
    });  
  
    xhr.open("GET", urlData);  
    xhr.setRequestHeader("cache-control", "no-cache");  
    xhr.send();  
}  
  
function getEstacionMasCercanaData(){  
    var xhr = new XMLHttpRequest();  
    xhr.withCredentials = true;  
    xhr.addEventListener("readystatechange", function () {  
        if (this.readyState === XMLHttpRequest.DONE) {  
            if(this.status === 0 || (this.status >= 200 && this.status < 400)){  
                console.log(this.responseText);  
                var url = JSON.parse(this.responseText).datos;  
                getEstacionMasCercanaUrl(url);  
            }  
            else{  
                showServerErrorToUser();  
            }  
        }  
    });  
  
    xhr.open("GET", "https://opendata.aemet.es/opendata/api/valores/climatologicos/inventarioestaciones/todasestaciones?api_key=" + aemetKey);  
    xhr.setRequestHeader("cache-control", "no-cache");  
    xhr.send();  
}
```

(Fuente: Elaboración propia)

Figura 21. WPCS-Obtención de la estación Aemet más cercana

```
function getMunEstDistance(munPosition, estData){  
    var [lat, long] = coordenadasGPS(estData.latitud, estData.longitud);  
    var estPosition = new Position(lat, long);  
    return munPosition.distanceTo(estPosition);  
}  
  
function findEstacionMasCercana(estaciones){  
    var munPosition = new Position(torrevieja.data.latitud_dec, torrevieja.data.longitud_dec);  
    var distance = getMunEstDistance(munPosition, estaciones[0]);  
    for(var key = 1; key < estaciones.length; key++){  
        var distAux = getMunEstDistance(munPosition, estaciones[key]);  
        if(distAux < distance){  
            distance = distAux;  
            torrevieja.estMasCercana = estaciones[key];  
            torrevieja.distancetoEstMasCercana = distance;  
        }  
    }  
    console.log("DISTANCE: " + torrevieja.distancetoEstMasCercana);  
    console.log("DISTANCE - KM: " + torrevieja.distancetoEstMasCercana/1000);  
    console.log("ESTACION: " + torrevieja.estMasCercana.nombre);  
}
```

(Fuente: Elaboración propia)

Figura 22. WPCS-Cálculo de distancias empleando coordenadas GPS

```
function coordenadasGPS(coordlat, coordlon) {  
  
    var lt = Array.from(coordlat);  
    var ln = Array.from(coordlon);  
    var gradlat = parseInt(lt[0] + lt[1]);  
    var minlat = parseInt(lt[2] + lt[3]);  
    var seglat = parseFloat(lt[4] + lt[5]);  
    var hemisflat = lt[6];  
    var gradlon = parseInt(ln[0] + ln[1]);  
    var minlon = parseInt(ln[2] + ln[3]);  
    var seglon = parseFloat(ln[4] + ln[5]);  
    var hemisflon = ln[6];  
  
    var latdms = gradlat + " " + minlat + " " + seglat + hemisflat;  
    var londms = gradlon + " " + minlon + " " + seglon + hemisflon;  
  
    var latdef = Dms.parse(latdms);  
    var londef = Dms.parse(londms);  
  
    return [latdef, londef];  
}  
}
```

(Fuente: Elaboración propia)

Cabe destacar que en esta ocasión, Aemet nos ha devuelto la respuesta al servidor mediante una URL en la que se aloja el JSON con los datos solicitados. Debido a esto, se ha precisado de una segunda petición a esta nueva URL para poder obtener los datos de todas las estaciones.

Con tal de realizar el cálculo de distancias entre el municipio de Torrevieja y las estaciones de Aemet se creó una clase denominada “Position”, la cual almacena internamente valores de latitud y longitud. Con esta clase creamos 2 objetos Position, uno para el municipio y otro para la estación Aemet que se esté comprobando en ese momento. Mediante el objeto Position del municipio empleamos el método “distanceTo”, que implementa la fórmula de “Haversine”, para ir calculando la distancia en metros desde Torrevieja hasta cada una de las estaciones de Aemet. Almacenaremos la de menor distancia, tal y como se ha comentado anteriormente.

Se indica que la implementación de dicha fórmula ha sido inspirada a partir del método “distanceTo” presente en la librería con licencia MIT denominada “latlon-spherical.js [40]”. En la siguiente figura se muestra la clase Position especificada.

Figura 23. WPCS-Clase Position

```

class Position{
    constructor(latitud, longitud){
        this.latitud = latitud;
        this.longitud = longitud;
    }
    degreeToRadians(degree){
        return (degree*(Math.PI))/180.0;
    }
    distanceTo(position, radio = 6371e3){ // Metodo basado en el metodo distanceTo de la libreria con licencia MIT llamada latlon-spherical.js con enlace --> https://cdn.jsdelivr.net/npm/geodesy@2/latlon-spherical.js
        // Formula de Haversine:
        // a = sin2(δθ/2) + cos(φ1)cos(φ2)sin2(λ/2)
        // δ = 2·atan2(√a), √(1-a))
        const R = radio;
        const F1i = this.degreeToRadians(this.latitud);
        const Lambda1i = this.degreeToRadians(this.longitud);
        const F1z = this.degreeToRadians(position.latitud);
        const Lambda2i = this.degreeToRadians(position.longitud);
        const DeltaPhi = F1z - F1i;
        const DeltaLambda = Lambda2i - Lambda1i;

        var a = Math.sin(DeltaPhi/2)*Math.sin(DeltaPhi/2) + Math.cos(F1i)*Math.cos(F1z) * Math.sin(DeltaLambda/2)*Math.sin(DeltaLambda/2);
        var c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1-a));
        var distance = R * c;
    }
    return distance;
}

```

Una vez obtenida la estación más cercana, se actualiza el objeto Municipio denominado torrevieja con estos datos. Se indica que la estación más cercana a Torrevieja es “San Javier Aeropuerto”, a una distancia de 23.5 km del municipio aproximadamente.

6.3.2. Creación de la interfaz de usuario

Una vez obtenida toda la información necesaria para realizar las peticiones al API de Aemet, realizamos la inserción de las imágenes que se utilizarán como interfaz de usuario. Para ello, utilizamos un servicio web denominado “*imgbb [41]*” con el cual se pueden subir imágenes a internet con la opción de que estas no se borren con el paso del tiempo a no ser que lo especifiques manualmente. De este modo, es posible emplear imágenes externas al sitio web de lagunas de la mata en torrevieja. El objetivo de esto es generar una interfaz mucho más usable para el usuario empleando para ello un icono que representa el tiempo atmosférico.

Para realizar la inserción de dichas imágenes se recorrerá el DOM de la página para recoger aquella etiqueta HTML que se corresponda con el calendario de la página. Una vez se obtiene esta etiqueta, se puede acceder a la cabecera de dicho calendario en donde se podrá ver el mes y el año al que pertenece, junto con cada una de las celdas en las que se especifica el día del mes en cuestión.

Una vez tenemos estos datos empleamos la fecha actual y la comparamos con el año, el mes y con cada día presente en el calendario. Si el día se corresponde con un día anterior, se realiza una petición al API de Aemet para recoger el tiempo que hizo en un tiempo pasado. Si por el contrario el día es el día actual o superior, se emplea otra petición para mostrar la predicción de tiempo que habrá en los sucesivos días. Estas peticiones se realizarán ejecutando el evento “*onclick()*”sobre las distintas imágenes que conforman la interfaz de usuario.

Cabe destacar que, en el calendario del sitio web se pueden mostrar días que pertenezcan o a un mes anterior o a un mes siguiente. Debido a esto, se deben filtrar los días obtenidos del

DOM de manera que solo se ejecute esta inserción de imágenes en aquellos días que pertenezcan al mes, dejando vacíos aquellos que no lo sean.

También se indica que se ha empleado la numeración mensual que se encuentra en el sistema para comparar los meses ya que esta es la que se utiliza en la instrucción “new Date()” cuando obtenemos la fecha actual. Para conseguir esto, se ha creado la función denominada “processDaysInCalendar(days)” mostrada en las siguientes figuras.

Igualmente, se visualiza más adelante el ícono utilizado para generar la interfaz de usuario junto con el resultado final de dicha interfaz expuesta en la web.

Figura 24. WPCS-Obtención del año, mes y días presentes en el calendario del sitio web

```
function processDaysInCalendar(days){
    var newDays = new Array();
    for(var i = 0; i < days.length; i++){
        if(!days[i].classList.contains("calendar-inactive")){
            newDays.push(days[i]);
        }
    }
    return newDays;
}

function getMonthNumber(month){
    var monthNumber = 0;
    month = month.toLowerCase();
    for(var m in MONTHS_OF_YEAR){
        if(m == month){
            monthNumber = MONTHS_OF_YEAR[m];
            sunHoursOfTorri = SUN_HOURS_OF_TORRI_PER_MONTH[monthNumber]; // Actualizamos las horas de sol del municipio de torrevieja en función del mes mostrado en el calendario del sitio web
        }
    }
    return monthNumber;
}

function getMonthYear(){
    var monthYear = document.querySelectorAll("body .mini-calendar-mes .mini-calendar-header .mini-calendar-month")[0].innerHTML;
    monthYear = monthYear.split(" ");
    monthYear[0] = getMonthNumber(monthYear[0]);
    return [monthYear[0], monthYear[1]];
}
```

(Fuente: Elaboración propia)

Figura 25. WPCS-Inserción de las imágenes correspondientes a la interfaz de usuario

```
function createImageForCalendar(type, month, year){
    var imgCalendar = document.createElement('img');
    imgCalendar.src = "https://i.ibb.co/ZYm23T/pred-Tiempo.png";
    imgCalendar.alt = 'Imagen predicción de tiempo';
    imgCalendar.border = '0';
    imgCalendar.onmouseover=function(e){
        this.style="max-width: 100%; cursor: pointer; filter: sepia(125%)";
    }
    imgCalendar.onmouseleave=function(e){
        this.style="max-width: 100%; cursor: auto; filter: sepia(0%)";
    }
    if(type==PREDICTION_TYPE){
        imgCalendar.addEventListener("click", function(){getDayPrediction(this.parentNode.children[0].children[0].innerHTML, month, year)}, false);
    }
    else{
        imgCalendar.addEventListener("click", function(){getDayTimePast(this.parentNode.children[0].children[0].innerHTML, month, year)}, false);
    }
    return imgCalendar;
}

function insertImagesInCalendar(){
    var days = document.querySelectorAll("body .mini-calendar-mes div div table tbody tr td");
    var now = new Date();
    var [month, year] = getMonthYear();
    days = processDaysInCalendar(days); // Obtenemos los días que pertenecen al mes, excluyendo para ello a aquellos que pertenezcan o a un mes anterior o a un mes siguiente
    for(var i = 0; i < days.length; i++){
        var day = new Date(year, month, days[i].children[0].children[0].innerHTML);
        if(compareDatesWith(day, now) == DATES_EQUALS || compareDatesWith(day, now) == DATES_HIGHER){
            days[i].appendChild(createImageForCalendar(PREDICTION_TYPE, month, year));
        }
        else{
            days[i].appendChild(createImageForCalendar(TIME_PAST_TYPE, month, year));
        }
    }
}
```

(Fuente: Elaboración propia)

Figura 26. WPCS-Comparación de fechas

```
function compareDatesWH(date1, date2){  
    var comp = DATES_EQUALS; // Son iguales  
    if(date1.getFullYear() < date2.getFullYear()){  
        comp = DATES_LOWER; // Date 1 es menor que date 2  
    }  
    else if(date1.getFullYear() > date2.getFullYear()){  
        comp = DATES_HIGHER; // Date 1 es mayor que Date 2  
    }  
    else{  
        if(date1.getFullYear() == date2.getFullYear() && date1.getMonth() < date2.getMonth()){  
            comp = DATES_LOWER;  
        }  
        else if(date1.getFullYear() == date2.getFullYear() && date1.getMonth() > date2.getMonth()){  
            comp = DATES_HIGHER; // Date 1 es mayor que Date 2  
        }  
        else{  
            if(date1.getFullYear() == date2.getFullYear() && date1.getMonth() == date2.getMonth() && date1.getDate() < date2.getDate()){  
                comp = DATES_LOWER;  
            }  
            else{  
                if(date1.getFullYear() == date2.getFullYear() && date1.getMonth() == date2.getMonth() && date1.getDate() > date2.getDate()){  
                    comp = DATES_HIGHER; // Date 1 es mayor que Date 2  
                }  
            }  
        }  
    }  
    return comp;  
}
```

(Fuente: Elaboración propia)

Figura 27. WPCS-Icono insertado para realizar la interfaz de usuario



(Fuente: Icono de sol, nube y lluvia [8])

Figura 28. WPCS-Interfaz de usuario

NOVIEMBRE 2020						
L	M	X	J	V	S	D
26	27	28	29	30	31	1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	1	2	3	4	5	6

(Fuente: Elaboración propia)

Se han empleado constantes para determinar si las fechas son iguales, inferiores o superiores. Esta identificación se utiliza para determinar que petición realizar a Aemet para obtener la información de tiempo correspondiente. En la siguiente figura se muestra el valor de dichas constantes.

Figura 29. WPCS-Constantes utilizadas

```
const DATES_EQUALS = 0;
const DATES_LOWER = 1;
const DATES_HIGHER = 2;

const PREDICTION_TYPE = 1;
const TIME_PAST_TYPE = 2;
```

(Fuente: Elaboración propia)

6.3.3. Creación de las peticiones de tiempo pasado y futuro a Aemet

Una vez tenemos tanto los datos necesarios para realizar las peticiones a Aemet como las imágenes que se utilizarán como interfaz de usuario, pasamos a explicar los 2 tipos de peticiones que se realizarán al API.

Comenzamos explicando la petición de obtener el tiempo que hizo en un día pasado. Para realizar esta consulta es necesario utilizar la estación Aemet más cercana a Torrevieja. Esto es así debido a que, Aemet almacena datos de días anteriores en sus estaciones Aemet.

Como consecuencia, si se quiere saber qué tiempo hizo en un día concreto pasado se necesita hacer una estimación a través de la estación más cercana para así poder mostrar los datos pertinentes. Debido a esto, no podemos utilizar el municipio de Torrevieja para extraer esta información sino que necesitamos la estación más cercana a dicho municipio.

Para realizar la petición necesitamos especificarle el rango de fechas del que queremos obtener datos especificado en formato UTC. Como vamos a enviar la petición al API de Aemet por cada día clicado de manera individual, este rango de fechas se corresponderá con el día seleccionado en el calendario.

El día y el mes seleccionado deberán presentar un formato de 2 cifras por lo que se emplea la función “parseDateNumberToCorrectFormat(dateNumber)” para cumplir con este requisito. También se indica que el valor del mes pasado a esta función debe incrementarse en 1 debido

a que se ha empleado internamente la numeración del sistema (0-11) y en este caso debe de utilizarse la numeración natural (1-12).

Además de la fecha necesitamos la id de la estación Aemet de la que se quiera consultar estos datos. Esta id ya la habríamos obtenido al realizar la petición para obtener los datos de la estación más cercana a Torrevieja. Cabe indicar que antes de realizar la consulta se comprueba si se poseen dichos datos.

En caso de que la petición haya podido realizarse correctamente, se envía una respuesta en formato JSON con una URL en la que se aloja el resultado de la consulta. Esta URL se identifica con el nombre “datos”. Una vez se obtienen los datos en formato JSON, estos se envían a la función que le detalla dicha información al usuario. En las siguientes figuras se muestra el código que refleja esta petición.

Figura 30. WPCS-Conversión de día y mes a un formato de 2 dígitos

```
function parseDateNumberToCorrectFormat(dateNumber){  
    var correctNumber = String(dateNumber);  
    for(var i = correctNumber.length; i < 2; i++){  
        correctNumber = '0' + correctNumber;  
    }  
    return correctNumber;  
}
```

(Fuente: Elaboración propia)

Figura 31. WPCS-Conversión de una fecha a formato UTC

```
function convertDateToUTC(day, month, year){  
    return year + "-" + month + "-" + day + "T00:00:00";  
}
```

(Fuente: Elaboración propia)

Figura 32. WPCS-Petición para una fecha pasada

```

function getDayTimePast(day, month, year){
    console.log(day);
    console.log(month);
    console.log(year);
    var correctDay = parseDateNumberToCorrectFormat(day);
    var correctMonth = parseDateNumberToCorrectFormat(month + 1); // Le incrementamos 1 al mes ya que nos lo almacenamos como valor 0 para poder realizar la comparativa de fechas con now a la hora de insertar las imágenes en el calendario
    var date = convertDate(correctDay, correctMonth, year) + "UTC";
    if((date == null || estHasCercana == null) && estHasCercana != null && sunHoursOff == null){
        var xhr = new XMLHttpRequest();
        xhr.withCredentials = true;
        xhr.addEventListener("readystatechange", function () {
            if (this.readyState === 0 || (this.status >= 200 && this.status < 400)){
                var data = JSON.parse(this.responseText);
                console.log(data);
                if(data.estado == 404){ // Si no hay datos para este dia el responseText devolvera el codigo 404 NOT FOUND, el cual se encuentra almacenado en el campo JSON denominado estado dentro de la variable data
                    showServerErrorToUser(correctDay + "-" + correctMonth + "-" + year);
                }
                else{
                    processTimePast(data.datos);
                }
            } else{
                showServerErrorToUser();
            }
        });
        xhr.open("GET", "https://opendata.aemet.es/opendata/api/valores/climatologicos/diarios/datos/fechaini/" + date + "/fechafin/" + date + "/estacion/" + torrevieja.estHasCercana.indicativo + "?api_key=" + aemetKey);
        xhr.setRequestHeader("cache-control", "no-cache");
        xhr.send();
    }
    else{
        showNoDataLoadedToUser();
    }
}

```

(Fuente: Elaboración propia)

Figura 33. WPCS-Segunda petición para obtener los datos de una fecha pasada

```

function processTimePast(dataUrl){
    var xhr = new XMLHttpRequest();
    xhr.withCredentials = true;
    xhr.addEventListener("readystatechange", function () {
        if (this.readyState === XMLHttpRequest.DONE) {
            if(this.status === 0 || (this.status >= 200 && this.status < 400)){
                var data = JSON.parse(this.responseText);
                console.log(data);
                showTimePastToUser(data);
            }
            else{
                showServerErrorToUser();
            }
        }
    });
    xhr.open("GET", dataUrl);
    xhr.setRequestHeader("cache-control", "no-cache");
    xhr.send();
}

```

(Fuente: Elaboración propia)

Y con esto queda explicada la petición al API de Aemet para conocer el tiempo que hizo en un día pasado. A continuación, se explica la petición para obtener una predicción de tiempo para los sucesivos días.

Esta petición funciona de manera un poco distinta a la anterior, ya que en este caso se necesita únicamente la id del municipio del que se quiera obtener la predicción del tiempo. Esto es así porque la predicción de tiempo sí que se realiza por municipios, por lo que en este caso no se necesita la estación Aemet más cercana a Torrevieja.

El resultado de esta petición será devuelto con el mismo formato que el anterior, es decir, la respuesta en formato JSON contendrá una URL en la que se alojará el resultado de esta consulta. También se ha añadido para este caso la comprobación de la existencia de los datos del municipio de Torrevieja ya que sin estos datos no se puede enviar la solicitud al API de Aemet.

Cabe decir que los datos de predicción del tiempo no se pueden obtener para una fecha futura determinada, sino que la petición te devuelve la estimación para el rango de días de los que se han podido predecir datos desde que se elaboró dicha estimación. Esto quiere decir que se elaboran informes de predicción cada “x” días y se almacena en el servidor de Aemet esa información con un rango de 7 días incluyendo el día en la que se elaboró dicha predicción.

Por ejemplo, si el día 20 de un mes “y” se realizaron las estimaciones para obtener una predicción de tiempo para los sucesivos días. El servidor de Aemet devolverá como resultado de la consulta, la predicción para el intervalo de días del 20 al 26 del mismo mes. Conformando con ello, una predicción de tiempo para 1 semana completa desde el día en el que se elaboró dicha predicción.

Conforme pasa un número determinado de días los datos de predicción se actualizan en el servidor mediante la elaboración de un nuevo informe que presentará las características especificadas en el ejemplo anterior.

Es por esta razón por la que se debe filtrar por fecha el resultado obtenido en esta consulta para obtener los datos de predicción para el día seleccionado por el usuario. El resultado de esta consulta se corresponde con un array de tamaño 7 representando la predicción para cada día en cuestión.

Esta situación se ha contemplado en la función “processDayPrediction(dataURL, day, month, year)”. Cabe decir que, para poder realizar este filtrado se ha tenido que incrementar en 1 la variable “month” y convertirla junto a la variable “day” a un formato de 2 dígitos de la misma manera que en la petición anterior para obtener datos de una fecha pasada. Se ha necesitado la función “convertDateToUTC(day, month, year)” explicada anteriormente para poder obtener la fecha con el formato adecuado para realizar el filtrado.

Por último, se ha utilizado la función “getDayPredictionProcessed(data, date)” para obtener los datos de la predicción para el día seleccionado por el usuario. Realizando un filtrado por fecha en el que, si la fecha de la predicción consultada era la que se le había especificado por parámetro, se devuelven los datos de dicha predicción. En caso de que la fecha especificada no

sea ninguna de las que se encuentran en el array de predicción de tiempo, esta función devuelve “null”.

Toda esta explicación se puede observar en las siguientes figuras.

Figura 34. WPCS-Petición para una fecha futura

```
function getDayPrediction(day, month, year){
    console.log(day);
    console.log(month);
    console.log(year);
    var correctDay = parseDateNumberToCorrectFormat(day);
    var correctMonth = parseDateNumberToCorrectFormat(month + 1); // Le incrementamos 1 al mes porque los datos se obtienen con el índice habitual del mes (enero ahora tiene un valor de 1 y no de 0)
    if(torreblanca.id != null & sunhoursOfforr != null){
        var xhr = new XMLHttpRequest();
        xhr.withCredentials = true;
        xhr.addEventListener("readystatechange", function () {
            if (this.readyState === XMLHttpRequest.DONE) {
                if (this.status >= 200 && this.status < 400){
                    var data = JSON.parse(this.responseText);
                    console.log(data);
                    if(data.estado == 404){ // Si no hay datos para este día el responseText devolverá el código 404 NOT FOUND, el cual se encuentra almacenado en el campo JSON denominado estado dentro de la variable data
                        showNoDataPredictionToUser(correctDay + "-" + correctMonth + "-" + year);
                    }
                    else{
                        processDayPrediction(data.datos, correctDay, correctMonth, year);
                    }
                }
                else{
                    showServerErrorToUser();
                }
            }
        });
        xhr.open("GET", "https://opendata.aemet.es/opendata/api/prediccion/especifica/municipio/diaria/" + torreblanca.id + "?api_key=" + aemetkey);
        xhr.setRequestHeader("cache-control", "no-cache");
        xhr.send();
    }
    else{
        showNoDataLoadedToUser();
    }
}
```

(Fuente: Elaboración propia)

Figura 35. WPCS-Segunda petición para obtener los datos de una fecha futura

```
function processDayPrediction(dataUrl, day, month, year){
    var xhr = new XMLHttpRequest();
    xhr.withCredentials = true;
    xhr.addEventListener("readystatechange", function () {
        if (this.readyState === XMLHttpRequest.DONE) {
            if(this.status == 0 || (this.status >= 200 && this.status < 400)){
                var data = JSON.parse(this.responseText);
                console.log(data);
                var date = convertDateToUTC(day, month, year);
                var dayPred = getDayPredictionProcessed(data, date);
                if(dayPred != null && dayPred != undefined){
                    if(dayPred.probPrecipitacion.length == 7){
                        showDayPredictionGraphToUser(dayPred, day + "-" + month + "-" + year);
                    }
                    else if(dayPred.probPrecipitacion.length == 3){
                        showDayPredictionTableToUser(dayPred, day + "-" + month + "-" + year);
                    }
                    else{
                        showDayPredictionToUser(dayPred, day + "-" + month + "-" + year);
                    }
                }
                else{
                    showNoDataPredictionToUser(day + "-" + month + "-" + year);
                }
            }
            else{
                showServerErrorToUser();
            }
        }
    });
    xhr.open("GET", dataUrl);
    xhr.setRequestHeader("cache-control", "no-cache");
    xhr.send();
}
```

(Fuente: Elaboración propia)

Figura 36. WPCS-Filtrado de fecha futura para el día seleccionado por el usuario

```
function getDayPredictionProcessed(data, date){  
    var dayPredProc = null;  
    for(var i = 0; i < data[0].prediccion.dia.length; i++){  
        if(data[0].prediccion.dia[i].fecha == date){  
            dayPredProc = data[0].prediccion.dia[i];  
        }  
    }  
    return dayPredProc;  
}
```

(Fuente: Elaboración propia)

Y con esto ya quedan especificadas las funciones que se utilizan para obtener los datos de tiempo pasado y futuro con el API de Aemet. A continuación, se detalla cómo se muestran estos datos con un formato agradable y entendible al usuario.

6.3.4. Muestra de datos al usuario

Para comenzar con la muestra de datos al usuario cabe decir que se han contemplado numerosos casos en función de los datos obtenidos con el API de Aemet. Estos casos se encuentran divididos de la siguiente manera:

- Si ha habido un error en el servidor de Aemet a la hora de realizar las llamadas al API.
- Si aún no se han obtenido los datos del municipio de Torrevieja y de su estación más cercana.
- Si los datos que se piden no han podido ser devueltos por el API debido a su inexistencia.
- Si los datos que se solicitan pertenecen a un día pasado.
- Si los datos que se piden pertenecen al día actual o al siguiente.
- Si los datos que se solicitan pertenecen a un día futuro correspondiente a 2 o 3 siguientes al actual.
- Si los datos que se piden pertenecen a un día futuro correspondiente a 4, 5 o 6 días siguientes al actual.

Estas situaciones se especifican al usuario en forma de mensajes modales o pop-ups en el que se detalla la información correspondiente a cada caso en cuestión. Estos pop-ups se han realizado empleando la librería javascript SweetAlert2 compatible con el diseño responsive. A continuación, se detallan cada uno de los apartados anteriores.

Error en el servidor de aemet

Es posible que a la hora de realizar la petición al API de Aemet, el código de respuesta no sea el de petición HTTP correcta (200). Esto puede significar que el servidor puede estar atravesando algún problema como por ejemplo el exceso de peticiones simultáneas, un error de conexión, entre otros. Dicha posibilidad se ha contemplado en cada petición realizada al API, desde la extracción de los datos del municipio de Torrevieja hasta la obtención de los datos de tiempo para un día concreto pasado o futuro.

En caso de darse esta situación, se le indica al usuario que recargue la página con el fin de realizar otro intento de conexión para conseguir realizar las peticiones pertinentes de la manera en la que se espera.

Cabe decir que este caso se produce en muy escasas circunstancias ya que los servidores de Aemet están preparados para atender un número bastante elevado de peticiones. Exceptuando aquellas ocasiones en las que dicho servidor se encuentre en mantenimiento, sería una situación que por normas generales no se produciría en el funcionamiento normal de WPCS.

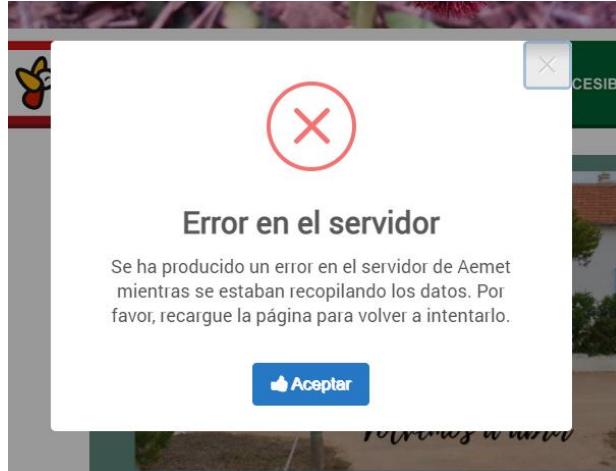
La función a la que se recurrirá cuando este caso suceda se denominará “showServerErrorToUser()”. En las siguientes figuras se muestra una captura del código empleado para crear el mensaje modal junto con su resultado final en la web.

Figura 37. WPCS-Mensaje modal de error en el servidor (código)

```
function showServerErrorToUser(){
  Swal.fire({
    title: '<strong>Error en el servidor</strong>',
    icon: 'error',
    html:
      '<p>Se ha producido un error en el servidor de Aemet mientras se estaban recopilando los datos. Por favor, recargue la página para volver a intentarlo.</p>',
    showCancelButton: true,
    showCancelButton: false,
    focusConfirm: false,
    confirmButtonText:
      '<i class="fa fa-thumbs-up"></i> Aceptar',
    confirmButtonAriaLabel: 'Thumbs up, great!',
  });
}
```

(Fuente: Elaboración propia)

Figura 38. WPCS-Mensaje modal de error en el servidor (web)



(Fuente: Elaboración propia)

Cargando datos de Torrevieja y de su estación aemet más cercana

En el caso en el que no se hayan obtenido los datos de Torrevieja o de su estación Aemet más cercana cuando el usuario interactúa con el script, se muestra un pop-up con el que se le indica que espere unos segundos antes de volver a intentar interactuar con el mismo. Esta situación puede ser producida por una respuesta tardía por parte del servidor a consecuencia de una conexión a internet aletargada.

La finalidad de esto es indicar al usuario que espere una cantidad corta de tiempo, dependiente de la velocidad de internet del mismo, para que así se puedan obtener estos datos que son necesarios para realizar las peticiones tanto para obtener datos de un tiempo pasado como de una predicción futura.

Este mensaje se muestra empleando la función “showNoDataLoadedToUser()” que se llama en aquellas funciones correspondientes a la interacción del usuario con el script. Es decir, se llama a esta función en la obtención de los datos de tiempo pasado o futuro. En el momento en el que dichos datos se hayan recopilado, esta situación dejará de contemplarse.

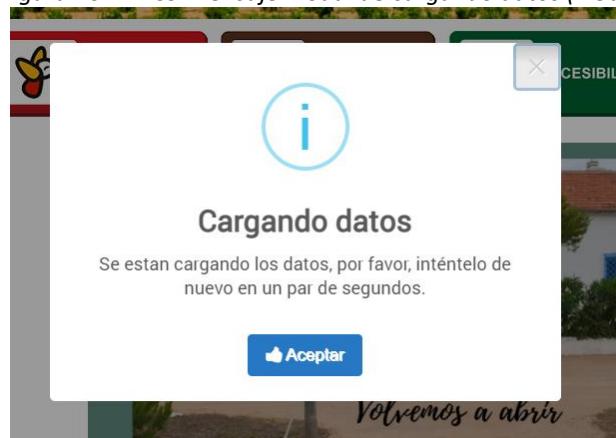
A continuación, se muestra en las siguientes figuras una captura del código empleado para generar este pop-up junto con el resultado observado en la web.

Figura 39. WPCS-Mensaje modal de cargando datos (código)

```
function showNoDataLoadedToUser(){
  Swal.fire({
    title: '<strong>Cargando datos</strong>',
    icon: 'info',
    html:
      '<p>Se estan cargando los datos, por favor, inténtelo de nuevo en un par de segundos.</p>',
    showCloseButton: true,
    showCancelButton: false,
    focusConfirm: false,
    confirmButtonText:
      '<i class="fa fa-thumbs-up"></i> Aceptar',
    confirmButtonAriaLabel: 'Thumbs up, great!',
  });
}
```

(Fuente: Elaboración propia)

Figura 40. WPCS-Mensaje modal de cargando datos (web)



(Fuente: Elaboración propia)

Inexistencia de datos

En caso de que se soliciten datos para una fecha pasada próxima a la actual de la cual Aemet no ha guardado datos o de una fecha futura muy lejana de la que no sea posible establecer ninguna clase de predicción, se informa al usuario de esta situación. Para ello se emplea la función “showNoDataPredictionToUser()”.

Esta función es llamada siguiendo el patrón descrito anteriormente, de manera que si es una fecha pasada o futura y la respuesta es un código 404 (Not Found) en el cuerpo de la respuesta devuelta por Aemet (teniendo en cuenta que la petición se haya podido realizar correctamente) entonces se mostraría este mensaje.

Si se obtienen datos de predicción para un rango de fechas y la fecha seleccionada se corresponde con una fecha futura, se comprueba la fecha seleccionada con la del rango de

fechas de las que se tiene predicción y si no es ninguna entonces también se le indica al usuario.

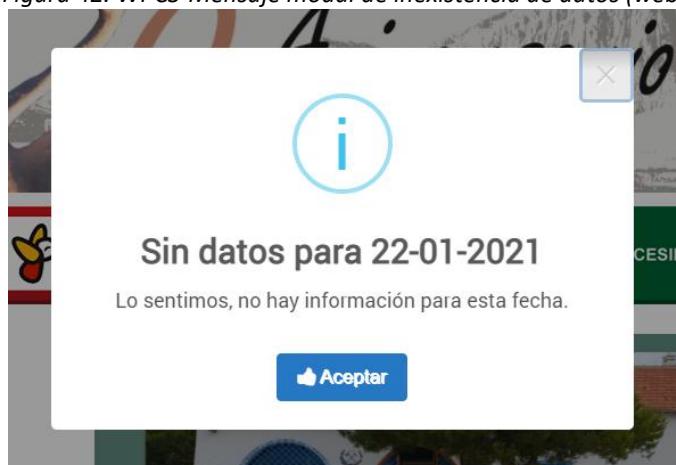
En las siguientes figuras se muestra el código empleado para mostrar este pop-up y su resultado final en la web.

Figura 41. WPCS-Mensaje modal de inexistencia de datos (código)

```
function showNoDataPredictionToUser(date){  
    Swal.fire({  
        title: '<strong>Sin datos para ' + date + '</strong>',  
        icon: 'info',  
        html:  
            '<p>Lo sentimos, no hay información para esta fecha.</p>',  
        showCloseButton: true,  
        showCancelButton: false,  
        focusConfirm: false,  
        confirmButtonText:  
            '<i class="fa fa-thumbs-up"></i> Aceptar',  
        confirmButtonAriaLabel: 'Thumbs up, great!',  
    });  
}
```

(Fuente: Elaboración propia)

Figura 42. WPCS-Mensaje modal de inexistencia de datos (web)



(Fuente: Elaboración propia)

Datos pertenecientes a un día pasado

Una vez expuestos los casos en los que no se hayan podido obtener datos de tiempo, pasamos a exponer la estructura del mensaje que recibirá el usuario cuando se haya obtenido la información para el día seleccionado. En esta situación, la fecha se corresponde con una fecha pasada. Dicha estructura tendrá la siguiente forma:

1. Fecha en formato natural de la información consultada.

2. Explicación del tipo de día que hizo junto con las horas de sol que hubo y su imagen identificativa.
3. Temperatura máxima, mínima y media en °C.
4. Índice de precipitación en mm de todo el día junto con el tipo de precipitación correspondiente.
5. Información de copyright de los datos junto con un enlace al sitio oficial de Aemet.

A continuación, se detallarán todos los apartados anteriores teniendo en cuenta la interpretación que se realiza de la información proporcionada por Aemet.

1. Fecha en formato natural

La fecha se mostrará como título del pop-up y se obtiene directamente desde los datos proporcionados por Aemet. Sin embargo, esta fecha presenta el formato de “año-mes-día” o “aaaa-mm-dd”, formato poco natural a la hora de referirse a una determinada fecha según la lengua española. Es por esto por lo que se cambiará este formato por el de “día-mes-año” o “dd-mm-aaaa” con tal de facilitar la lectura y comprensión de los datos recibidos. Para realizar este cambio de formato se emplea la función “getCorrectDateFormat(date)” mostrada en la siguiente figura.

Figura 43. WPCS-Cambio de formato a "dd-mm-aaaa"

```
function getCorrectDateFormat(date){
    var dateAux = date.split("-");
    return dateAux[2] + "-" + dateAux[1] + "-" + dateAux[0];
}
```

(Fuente: Elaboración propia)

2. Obtención del tipo de día

Para realizar esta interpretación se ha empleado el valor de sol correspondiente al *índice de insolación* [42] que hizo durante todo el día. Este índice se corresponde con el tiempo en horas en el que el sol ha estado efectuando radiación solar de manera en que se puedan producir sombras bien diferenciadas, por lo que nos sirve para poder estimar las horas de sol que hubo en el día según la estación Aemet en la que se han extraído los datos. También se empleará el índice de precipitación por día para conocer si hubo precipitaciones.

Para poder estimar si un día fue soleado o nublado, se emplea el índice de sol junto con las horas de sol que hay en el mes dentro del municipio de Torrevieja. Con estos datos se realiza

una aproximación más o menos fidedigna de cuando estimar si en el parque natural ha habido un día soleado o no empleando los datos de la estación Aemet más cercana a dicho municipio.

Con tal de averiguar el número de horas de sol que hay por mes en Torrevieja se empleó el sitio web “*meteogram.es [43]*” ya que en él se puede observar una tabla para cada mes del año en el que se refleja la hora de salida el sol junto con la hora en la que este se pone. Consultando esta tabla y observando su variación con respecto al paso de los años se considera que es una aproximación adecuada, ya que estas horas de salida y puesta de sol varían muy ligeramente de un año a otro y no en todos los meses, por lo que se puede asumir el error relativo correspondiente y emplear los datos de este sitio web.

En este caso se ha empleado el intervalo de horas de sol para el año 2020. Se ha decidido emplear este sistema de obtención de las horas de sol debido a que se asumiría un mayor error relativo si se tratara de calcular el intervalo de horas de sol que habría en Torrevieja por estaciones, empleando para ello el solsticio de verano y el de invierno. Debido a esto, la estimación horaria englobaría un total de 3 meses (correspondientes a cada estación), por lo que el desplazamiento producido por la variación de las horas de sol por meses sería mayor.

El cálculo de las horas totales de sol se realizó calculando el rango máximo de tiempo en horas, desde la hora mínima de salida del sol hasta la hora máxima de puesta del mismo de la siguiente manera:

$$H_{TOT_SOL} = |h_{min_salida} - h_{max_puesta}| = h_{max_puesta} - h_{min_salida}$$

A continuación, se muestra una tabla en la que se refleja este total de horas por mes para el municipio de Torrevieja. Cabe destacar que en aquellos meses en los que se produzca el cambio de hora se emplea la hora previa al cambio de forma que si la hora expuesta en la tabla es 20:25 se tomará como las 19:25 si es el cambio al horario de verano. Si es del horario de verano al horario de invierno se tomará la hora siguiente. Las horas expresadas en la tabla del sitio web están divididas por horarios, de manera que las horas redactadas en negro están en horario de invierno y las escritas en azul están en horario de verano.

Tabla 4. WPCS-Cálculo de horas totales de sol en Torrevieja

	H_{min_salida}	H_{max_puesta}	H_{TOT_SOL}
Enero	08:08	18:23	10h 15'
Febrero	07:34	18:55	11h 21'
Marzo	06:48	19:25	12h 37'
Abril	07:07	20:52	13h 45'
Mayo	06:41	21:19	14h 38'
Junio	06:39	21:29	14h 50'
Julio	06:44	21:29	14h 45'
Agosto	07:06	21:11	14h 05'
Septiembre	07:33	20:32	12h 59'
Octubre	07:58	19:45	11h 47'
Noviembre	07:28	18:03	10h 35'
Diciembre	08:00	17:53	9h 53'

(Fuente: Elaboración propia)

Estas horas se encuentran almacenadas en forma de array empleando para ello una constante de manera que, en el momento en el que se extrae el mes del calendario del sitio web, se actualiza el valor de la variable global “sunHoursOfTor” con el valor de las horas totales de sol para el mes en cuestión. Esta actualización se puede observar en la Figura 24, concretamente, en la función “getMonthNumber(month)”. Dicha función aprovecha la extracción de la identificación del mes para actualizar esta variable global a su valor correcto de horas totales de sol.

Figura 44. WPCS-Constantes referentes a los distintos meses junto con sus horas de sol

```
const MONTHS_OF_YEAR = {enero: 0, febrero: 1, marzo: 2, abril: 3, mayo: 4, junio: 5, julio: 6, agosto: 7, septiembre: 8, octubre: 9, noviembre: 10, diciembre: 11};
const SUN_HOURS_OF_TORR_PER_MONTH = new Array(10.15, 11.21, 12.37, 13.45, 14.38, 14.50, 14.45, 14.05, 12.59, 11.47, 10.35, 9.53); // Media de horas de sol por mes e
```

(Fuente: Elaboración propia)

Una vez detallado el proceso de obtención de las horas de sol totales por mes existentes en Torrevieja se especifica la clasificación del día como soleado o nublado. El proceso es simple, únicamente se trata de dividir estas horas totales entre 2, de manera que si el índice de insolación es mayor o igual a esta cantidad de horas es que el día ha sido mayoritariamente soleado por lo que se establece como tal. Si es menor, el día será nublado según la siguiente expresión:

$$\text{Soleado} \rightarrow \text{Sol} \geq H_{TOT_SOL} / 2$$

$$\text{Nublado} \rightarrow \text{Sol} < H_{TOT_SOL} / 2$$

Para saber si el día fue lluvioso o cualquiera de los derivados se emplea el índice de precipitación, de manera que si este índice no es “Ip” (Inapreciable, es decir, menor que 0.1 mm en todo el día) y mayor que 0, ha precipitado en el día por lo que se considera a ese día lluvioso. En el caso en el que la precipitación por horas sea mayor que 15 mm se considera que el día ha sido tormentoso, esta especificación se detallará más adelante en el apartado referente a la obtención del tipo de precipitación que hubo en el día.

Cabe destacar que existe la posibilidad de que no se proporcione el índice de insolación ya que no es un dato obligatorio a la hora de obtener los datos referentes al tiempo que hizo en un día pasado según los metadatos proporcionados por Aemet. Si se diera este caso simplemente se le especifica al usuario que no se poseen datos para poder determinar el tipo de día que hubo en la fecha seleccionada.

3. Especificación de los valores de temperatura

Para este caso lo único que se emplea es una función que corrige la salida que se le mostrará al usuario de manera que, si no se poseen datos sobre cualquiera de las temperaturas, se escribe en el pop-up el mensaje “Sin datos” según corresponda. Este mismo caso se generaliza a la muestra de datos de cualquier campo de forma que, la función empleada en esta situación, se seguirá empleando también en el pop-up correspondiente a los datos de predicción para una fecha futura. La función descrita se denomina “getCorrectData(data)” y se puede visualizar en la siguiente figura.

Figura 45. WPCS-Muestra adecuada de campos en función de su existencia

```
function getCorrectData(data){  
    var correctData = "Sin datos";  
    if(data != null && data != undefined && data.toString() != ""){  
        correctData = data.toString();  
    }  
    return correctData;  
}
```

(Fuente: Elaboración propia)

4. Índice y tipo de precipitación

Para determinar el tipo de precipitación que hubo en el día se realiza una estimación consistente en que si ha habido una precipitación en mm/h superior a una cifra en particular, se cataloga a dicha precipitación de una manera u de otra. Estos valores de intensidad de precipitación se corresponden con los siguientes:

Tabla 5. WPCS-Clasificación del tipo de precipitación

Intensidad	Acumulación por hora (mm)
Débil	$i \leq 2$
Moderada	$2 < i \leq 15$
Fuerte	$15 < i \leq 30$
Muy fuerte	$30 < i \leq 60$
Torrencial	$i > 60$

(Fuente: divulgameteo.es [44] y wikipedia [45])

El caso de “ $0.0 < i \leq 0.1$ ” se considera precipitación débil también debido a que el índice de precipitación obtenido mediante Aemet es diario y no por hora, por lo que si ese valor es superior a 0 y no es inapreciable se considera que ha habido precipitaciones. Esto quiere decir que si la estimación horaria del índice de precipitación es igual o inferior a esta cantidad se determina como precipitación débil al haber precipitado la cantidad suficiente como para que ese dato no se considere inapreciable para el índice de precipitación diario.

Como se ha mencionado anteriormente, hay que realizar una aproximación de la cantidad de milímetros o “mm” que han precipitado por hora en el día en cuestión. Esta estimación considera que ha precipitado de manera regular durante las 24 horas del día ya que no se posee información más detallada que determine a qué hora ha precipitado ni cuanta cantidad.

Debido a esto, se asume el error relativo cometido realizando esta estimación de manera que se considera que si en un día ha precipitado un índice muy elevado es que las precipitaciones han sido, como mínimo, fuertes aunque no se conozca si la mayoría de dicho índice de precipitación se ha recopilado en, por ejemplo, un par de horas en las que han habido precipitaciones torrenciales. Según este esquema, el índice de precipitación que utilizaremos se calcula empleando la siguiente metodología:

$$I_{PREC_H} = I_{PREC_DIARIA} / 24$$

Realizando este cálculo y estableciendo la clasificación pertinente se muestra el resultado obtenido al usuario. Cabe indicar que en la obtención del tipo de día se ha empleado también este cálculo para determinar si un día ha sido soleado con precipitaciones, lluvioso o tormentoso, de manera que se establece la siguiente clasificación para el tipo de día:

Tabla 6. WPCS-Clasificación del tipo de día

Tipo de día	Condiciones
Soleado con precipitaciones	$\text{Sol} \geq \text{H}_{\text{TOT_SOL}} / 2 \ \&\ \& \text{I}_{\text{PREC_DIARIA}} > 0.1 \ \&\ \& \text{I}_{\text{PREC_H}} > 0.0$
Lluvioso	$\text{Sol} < \text{H}_{\text{TOT_SOL}} / 2 \ \&\ \& \text{I}_{\text{PREC_DIARIA}} > 0.1 \ \&\ \& \text{I}_{\text{PREC_H}} \leq 15$
Tormentoso	$\text{Sol} < \text{H}_{\text{TOT_SOL}} / 2 \ \&\ \& \text{I}_{\text{PREC_DIARIA}} > 0.1 \ \&\ \& \text{I}_{\text{PREC_H}} > 15$

(Fuente: Elaboración propia)

5. Información de copyright y enlace a Aemet

De acuerdo con la *política de Aemet* [46] se ha de especificar explícitamente que los datos que se muestren empleando la información obtenida mediante su API pertenecen a la Agencia Estatal de Meteorología. Es por esto por lo que, en todos los pop-ups en los que se muestran datos de Aemet se redacta esta especificación para cumplir con su acuerdo de legalidad.

También cabe decir que para poder cumplir también con la normativa impuesta por Aemet hay que presentarle al usuario los datos de manera en que no se pueda manifestar una interpretación errónea de los mismos. Debido a ello, se han realizado las estimaciones anteriores según la información proporcionada por Aemet para cada dato que devuelve su API, con el objetivo de presentarle al usuario dicha información de manera fidedigna a lo que representa la Agencia Estatal de Meteorología. Dicha información puede ser encontrada en la sección de *ayuda de Aemet* [47].

Como ya se ha comentado anteriormente, esta información de copyright también aparecerá en los pop-ups correspondientes a la predicción de tiempo para una fecha futura de igual forma en que aparece para una fecha pasada. También se indica que se ha empleado la función “getCopyrightInfo()” para cumplir con esta especificación.

A continuación, se muestra en las siguientes figuras el código empleado para la estimación del tipo de día y de precipitación además de mostrar el código referente a la creación del pop-up en cuestión. También se muestran las imágenes que se emplearán para caracterizar cada tipo de día de manera separada y diferenciada junto con el resultado final en la web.

Figura 46. WPCS-Obtención del tipo de día

```

function getTypeOfDay(sunhours, prec){
    var typeOfDay = "<figure><img src='https://i.ibb.co/2vJ0Kz/no-weather-info-icon.png' alt='Icono de sin datos sobre el tipo de dia' width='250' height='250' border='0'></figure><br><p>sin datos sobre el tipo de dia</p>";
    if(sunhours > 0.0 && sunhours <= 2.0){
        var hoursMonth = sunhours*24/2.0;
        console.log("HORAS DE SOL TORREZUELA: " + sunhours*24/2.0);
        console.log("SUN HOURS: " + sunhours);
    }
    if(parseFloat(sunhours) < hoursMonth){
        if(prec <= "Ip" && parseFloat(prec) > 0.0 && parseFloat(prec) <= 24.0) {
            typeOfDay = "<figure><img src='https://i.ibb.co/mCryqBd/lluvia.png' alt='lluvia' width='250' height='250' border='0'></figure><br><p>lluvioso con horas de sol: " + sunhours + "h</p>";
        } else if(prec > "Ip" && parseFloat(prec) > 15.0) {
            typeOfDay = "<figure><img src='https://i.ibb.co/f8RrVx/tormenta.png' alt='tormenta' width='250' height='250' border='0'></figure><br><p>Tormentoso con horas de sol: " + sunhours + "h</p>";
        } else{
            typeOfDay = "<figure><img src='https://i.ibb.co/kK2b6rk/nublado.png' alt='nublado' width='250' height='250' border='0'></figure><br><p>nublado con horas de sol: " + sunhours + "h</p>";
        }
    } else{
        if(prec == "Ip" && parseFloat(prec) > 0.0) {
            typeOfDay = "<figure><img src='https://i.ibb.co/5sTYBF/sol-con-precipitaciones.png' alt='soleado con precipitaciones' width='250' height='250' border='0'></figure><br><p>despejado con precipitaciones y con horas de sol: " + sunhours + "h</p>";
        } else{
            typeOfDay = "<figure><img src='https://i.ibb.co/mgSLFv/soleando.png' alt='soleado' width='250' height='250' border='0'></figure><br><p>despejado con horas de sol: " + sunhours + "h</p>";
        }
    }
}
return typeOfDay;
}

```

(Fuente: Elaboración propia)

Figura 47. WPCS-Obtención del tipo de precipitación

```

function getTypeOfPrec(prec){
    var dayPrec = "<p>";
    if(prec == null || prec == undefined){
        dayPrec = dayPrec + "Sin datos sobre la precipitación que hubo en el dia</p>";
    }
    else if(prec == "Ip"){
        dayPrec = dayPrec + "Precipitación inapreciable: < 0.1mm en 24 horas</p>";
    }
    else{
        var precForHour = (parseFloat(prec)/24.0).toFixed(3);
        if(precForHour == 0.0){
            dayPrec = dayPrec + "Sin precipitaciones</p>";
        }
        else if(precForHour > 0.0 && precForHour <= 2.0){
            dayPrec = dayPrec + "Precipitaciones débiles: " + prec + "mm en 24 horas y " + precForHour + "mm por hora</p>";
        }
        else if(precForHour > 2.0 && precForHour <= 15.0){
            dayPrec = dayPrec + "Precipitaciones moderadas: " + prec + " en 24 horas y " + precForHour + "mm por hora</p>";
        }
        else if(precForHour > 15.0 && precForHour <= 30.0){
            dayPrec = dayPrec + "Precipitaciones fuertes: " + prec + " en 24 horas y " + precForHour + "mm por hora</p>";
        }
        else if(precForHour > 30.0 && precForHour <= 60.0){
            dayPrec = dayPrec + "Precipitaciones muy fuertes: " + prec + " en 24 horas y " + precForHour + "mm por hora</p>";
        }
        else{
            dayPrec = dayPrec + "Precipitaciones torrenciales: " + prec + " en 24 horas y " + precForHour + "mm por hora</p>";
        }
    }
    return dayPrec;
}

```

(Fuente: Elaboración propia)

Figura 48. WPCS-Creación del copyright de Aemet

```

function getCopyrightInfo(){
    return '<p>Información elaborada por la Agencia Estatal de Meteorología (@AEMET). Más información en su <a href="http://www.aemet.es/es/portada" style="color: #374DC3; target="_blank">sitio oficial</a></p>';
}

```

(Fuente: Elaboración propia)

Figura 49. WPCS-Mensaje modal de tiempo pasado (código)

```

function showTimePastToUser(data){
    Swal.fire({
        title: '<strong>TIEMPO DÍA: ' + getCorrectDateFormat(data[0].fecha) + '</strong>',
        icon: 'info',
        html:
            getTipoOfDay(data[0].sol, data[0].prec)
            + '<p>Temperatura máxima (°C): ' + getCorrectData(data[0].tmax) + '</p>'
            + '<p>Temperatura mínima (°C): ' + getCorrectData(data[0].tmin) + '</p>'
            + '<p>Temperatura media (°C): ' + getCorrectData(data[0].tméd) + '</p>'
            + getTypeOfPrec(data[0].prec) + '<br>' +
            getCopyrightInfo(),
        showCancelButton: true,
        showCancelButton: false,
        focusConfirm: false,
        confirmButtonText:
            '<i class="fa fa-thumbs-up"></i> Aceptar',
        confirmButtonAriaLabel: 'Thumbs up, great!',
    });
}

```

(Fuente: Elaboración propia)

Figura 50. WPCS-Imágenes identificativas del tipo de día



(Fuente: Soleado [9], Soleado con prec. [10], Nublado [11], Lluvioso [12], Tormentoso [13])

Figura 51. WPCS-Mensaje modal de tiempo pasado (web)



(Fuente: Icono de tiempo Soleado [9])

Una vez detallada la muestra de la información correspondiente a una fecha pasada pasamos a especificar el de la predicción para una fecha futura.

Datos que pertenecen al día actual o al siguiente

Aemet especifica los datos de predicción que tiene para los sucesivos días de manera que apila, por horas, las estimaciones de tiempo de cara al futuro. Esta especificación permite establecer diferentes representaciones para presentar los datos al usuario. Cuando la predicción temporal es para el día de hoy o para el siguiente, Aemet apila los datos de predicción correspondientes a su evolución climática para cada periodo de 6 horas existente en el día.

Es por esta razón por la que, datos como la probabilidad de precipitación o el estado del cielo devueltos por Aemet presentan un tamaño de 7 ya que se encuentran apilados los datos referentes a la evolución climática por cada 6 horas, por cada 12 y por las 24 horas del día, dando lugar al tamaño total antes mencionado.

Se indica también que datos como la temperatura o la sensación térmica presentarán un tamaño de 4 referente a la evolución cada 6 horas en esta situación y no de 7 como ocurre con el caso expuesto anteriormente. Sin embargo, esto no es incompatible con la información previa, ya que solo presentarán un tamaño de 4 cuando datos como la probabilidad de precipitación posean un tamaño de 7. A continuación, se indican que datos presentan un tamaño de 7 y de 4 respectivamente:

- **Tamaño 7:** Cota de nieve, estado del cielo, probabilidad de precipitación, viento y racha máxima del mismo.
- **Tamaño 4:** Humedad relativa, sensación térmica y temperatura.

Sabiendo esto, comenzamos a detallar como se presentan los datos recibidos por Aemet cuando tenemos una división horaria de 6 horas que nos permita observar la evolución del tiempo en el día. Debido a la obtención de una predicción bastante específica en este caso, la estructura del pop-up que se mostrará al usuario presentará la siguiente forma:

1. Gráfico de líneas con la evolución de la temperatura, la humedad relativa, la probabilidad de precipitación y el viento.
2. Tabla en la que se muestra la evolución del estado del cielo, la cota de nieve, la racha máxima de viento y la sensación térmica.
3. Índice de radiación ultravioleta máxima en el día.
4. Información de copyright y enlace a Aemet.

Al igual que en el caso anterior se detallará, punto por punto, la creación de cada una de las partes de este pop-up.

1. Gráfico de líneas

Con tal de mostrar la predicción del tiempo para datos generales como la temperatura, la humedad relativa, la probabilidad de precipitación y el viento se ha realizado un gráfico que refleje de manera clara y concisa la evolución de dichos datos cada 6 horas. Debido a que disponemos en esta situación de datos bastante específicos podemos realizar un gráfico de estas características. Para el resto de situaciones de predicción de tiempo no se empleará ningún gráfico debido a que no se poseerán los datos suficientes como para crear uno.

La librería *Chart.js* funciona a través de un “<canvas>” en el que dibuja el gráfico pertinente según los datos que se le especifiquen, indicando que dicho “<canvas>” debe estar previamente creado en la web. Por este motivo, se emplea una arquitectura de *promesa en javascript* [48], de manera en que se crea el gráfico cuando el mensaje modal ha sido mostrado con tal de asegurarnos de que dicho “<canvas>” ya ha sido generado en el momento en el que se llama a la librería. El “<canvas>” se encontrará en el pop-up que se le muestra al usuario y se recogerá en la función que carga el gráfico mediante el DOM.

También se recogen posibles excepciones a la promesa de manera que no se ejecute el código referente a la utilización de esta librería si se ha producido cualquier error que haya provocado, posiblemente, la no generación del “<canvas>” necesario. Esta arquitectura se podrá visualizar en las siguientes figuras, concretamente en aquella en la que se muestra la función “*showDayPredictionGraphToUser(dayPredData, date)*”.

Se indica que, para seguir esta arquitectura se ha empleado una función puente denominada “*loadGraphs(dayPredData, date)*” que será la encargada de llamar a la función “*createGraphs(dayPredData, date)*”, siendo esta última quien creará el gráfico. Esta función se podría omitir pero en este caso se ha decidido mantener por si en el futuro se tuviera que emplear algún procedimiento adicional a la hora de generar y mostrar el gráfico al usuario.

Una vez explicada la arquitectura utilizada para la generación del gráfico pasamos a detallar la extracción de los datos que se muestran en el mismo. Para ello, se extraen los datos referentes a la temperatura, la humedad relativa, la probabilidad de precipitación y el viento, de manera que se devuelve el valor concreto para la variación cada 6 horas expuesto hasta el momento.

Cabe destacar que se realizará un filtrado para aquellos datos pertenecientes al tamaño 7 de manera que solo se almacenen los correspondientes al periodo horario establecido. También se detalla que las etiquetas que se mostrarán por cada punto del gráfico de líneas se pueden especificar de manera manual o extrayéndolas de alguno de los datos procesados.

En este caso, se ha empleado la extracción de la información relativa a la probabilidad de precipitación para obtener estas etiquetas ya que, al ser uno de los datos de tamaño 7, el periodo horario está representado de una manera más cómoda y fácil de entender para el usuario. De esta forma se aprovechan aún más los datos proporcionados por Aemet.

También se especifica que, en dicho filtrado, se ha empleado una función denominada “pushDataToArray(dataArray, newData)” que se encargará de introducir la información con un formato adecuado en el array que se le especifica por parámetro de manera que, si el dato en cuestión es un “null”, un “undefined” o directamente una cadena vacía (“”), se introduce dentro del array el valor “Sin datos” reflejando esta situación.

Este caso no debería de suceder en ninguna instancia debido a que se realiza un filtrado previo a este para determinar el tipo de predicción, asegurándonos con ello de que cuando se llame a esta función se tengan todos los datos precisos para la elaboración del gráfico de líneas. Sin embargo, con el fin de hacer el código más robusto, se ha decidido emplear dicha función.

Figura 52. WPCS-Inserción adecuada de datos en el array utilizado por el gráfico de líneas

```
function pushDataToArray(dataArray, newData){  
    if(newData != null && newData != undefined && newData.toString() != ""){  
        dataArray.push(newData.toString());  
    }else{  
        dataArray.push("Sin datos");  
    }  
}
```

(Fuente: Elaboración propia)

Figura 53. WPCS-Extracción de datos para generar el gráfico de líneas

```
function getTempArrayDataForGraph(temp){
    var tempArrayData = new Array();
    for(var i = 0; i < temp.dato.length; i++){
        pushDataToArray(tempArrayData, temp.dato[i].value);
    }
    return tempArrayData;
}

function getHumRelArrayDataForGraph(humRel){
    var humRelArrayData = new Array();
    for(var i = 0; i < humRel.dato.length; i++){
        pushDataToArray(humRelArrayData, humRel.dato[i].value);
    }
    return humRelArrayData;
}

function getProbPrecArrayDataForGraph(probPrec){
    var graphLabels = new Array();
    var graphProbPrecArrayData = new Array();
    for(var i = 0; i < probPrec.length; i++){
        if(probPrec[i].periodo == "00-24" || probPrec[i].periodo == "00-12" || probPrec[i].periodo == "12-24"){
            continue;
        }
        graphLabels.push(probPrec[i].periodo + "h");
        pushDataToArray(graphProbPrecArrayData, probPrec[i].value);
    }
    return [graphLabels, graphProbPrecArrayData];
}

function getVientoArrayDataForGraph(viento){
    var graphVientoArrayDir = new Array();
    var graphVientoArrayData = new Array();
    for(var i = 0; i < viento.length; i++){
        if(viento[i].periodo == "00-24" || viento[i].periodo == "00-12" || viento[i].periodo == "12-24"){
            continue;
        }
        graphVientoArrayDir.push(viento[i].direccion);
        pushDataToArray(graphVientoArrayData, viento[i].velocidad);
    }
    return [graphVientoArrayDir, graphVientoArrayData];
}
```

(Fuente: Elaboración propia)

Por último, cabe decir que *Chart.js* funciona empleando una estructura de datos denominada “dataset”, que interpreta internamente, para generar cada uno de los puntos que conformarán el gráfico de líneas anteriormente nombrado. Se crea un gráfico por cada “dataset” especificado, de manera en que se puede crear más de un gráfico del mismo tipo en el canvas utilizado. En las siguientes figuras se muestran las funciones empleadas para crear y mostrar el gráfico en cuestión.

Figura 54. WPCS-Función puente para crear el gráfico de líneas

```
function loadGraphs(dayPredData, date){
    createGraphs(dayPredData, date);
}
```

(Fuente: Elaboración propia)

Figura 55. WPCS-Creación del gráfico de líneas

```

function createGraphs(dayPredData, date){
    var dataGraphNames = ['Temperatura (°C)', 'Humedad Relativa (%)', 'Prob Precipitación (%)', 'Viento (km/h)'];
    var tempValues = getTempArrayDataForGraph(dayPredData.temperatura);
    var humeRelValues = getHumeRelArrayDataForGraph(dayPredData.humedadRelativa);
    var [probValues, probRecValues] = getProbPrecipArrayDataForGraph(dayPredData.probPrecipitacion);
    var [vientoLabels, vientoValues] = getVientoArrayDataForGraph(dayPredData.viento);
    var canvas = document.getElementById("tempPredGraph").getContext('2d');
    var tempPredGraph = new Chart(canvas, {
        type: 'line',
        data: {
            labels: graphLabels,
            datasets: [
                {
                    label: dataGraphNames[0],
                    fill: false,
                    data: tempValues,
                    borderColor: '#FF7E0C',
                    pointBorderWidth: 3,
                    pointRadius: 5,
                    borderWidth: 1
                },
                {
                    label: dataGraphNames[1],
                    fill: false,
                    data: humeRelValues,
                    borderColor: '#D9A2FB',
                    pointBorderWidth: 3,
                    pointRadius: 5,
                    borderWidth: 1
                },
                {
                    label: dataGraphNames[2],
                    fill: false,
                    data: probRecValues,
                    borderColor: '#25ACFF',
                    pointBorderWidth: 3,
                    pointRadius: 5,
                    borderWidth: 1
                },
                {
                    label: dataGraphNames[3],
                    fill: false,
                    data: vientoValues,
                    borderColor: '#4B8C69B',
                    pointBorderWidth: 3,
                    pointRadius: 5,
                    borderWidth: 1
                }
            ],
            options: {
                responsive: true,
                maintainAspectRatio: false,
                scales: {
                    xAxes: [
                        {
                            scaleLabel: {
                                display: true,
                                labelString: 'Horas',
                                fontSize: 15
                            },
                            ticks: {
                                beginAtZero: true
                            }
                        }],
                    yAxes: [
                        {
                            ticks: {
                                beginAtZero: true
                            }
                        }]
                },
                title: {
                    display: true,
                    text: date,
                    fontSize: 25
                },
                tooltips: {
                    callbacks: [
                        label: function(tooltipItem, data) {
                            var label = data.datasets[tooltipItem.datasetIndex].label + ": " + tooltipItem.value;
                            if(data.datasets[tooltipItem.datasetIndex].label == dataGraphNames[3]){
                                label = data.datasets[tooltipItem.datasetIndex].label + " Dirección: " + vientoDir[vientoItem.datasetIndex] + ". Velocidad: " + tooltipItem.value + " km/h";
                            }
                            return label;
                        }
                    ]
                }
            });
        });
}

```

(Fuente: Elaboración propia)

Se puede consultar la información relativa a cada especificación empleada en la creación del gráfico mediante la función “new Chart(canvas, function)” en la *documentación oficial [49]* de la librería *Chart.js*.

2. Tabla

Una vez explicada la elaboración del gráfico pasamos a detallar la creación de la tabla que mostrará los datos relativos a la evolución cada 6 horas del estado el cielo, la cota de nieve, la racha máxima de viento y la sensación térmica. Como estos datos no se pueden expresar de una manera clara y concisa empleando un gráfico debido a la diversidad de información que representan, se ha decidido emplear una tabla.

Se indica que, la generación de esta tabla se ha realizado de manera dinámica con la finalidad de poder emplearla en el caso de que se posean menos datos de predicción. Esta tabla se ha generado mediante el DOM y, de la misma forma que en la obtención de los datos necesarios para la elaboración del gráfico, se ha empleado un filtrado con el objetivo de mostrar los datos de manera adecuada en la tabla. Este filtrado presenta las mismas características que en el caso anterior. A continuación, se muestran las funciones empleadas para realizarlo.

Figura 56. WPCS-Filtrado de datos para las tablas de predicción

```

function getEstadoCielo(estCielo){
    var tableColumns = new Array();
    var estCieloData = new Array();
    for(var i = 0; i < estCielo.length; i++){
        if(estCielo[i].periodo == "00-24" || estCielo[i].periodo == "00-12" || estCielo[i].periodo == "12-24"){
            continue;
        }
        tableColumns.push(estCielo[i].periodo + "h");
        pushDataToArray(estCieloData, estCielo[i].descripcion);
    }
    return [tableColumns, estCieloData];
}

function getCotaNieve(cotaNieve){
    var cotaNieveData = new Array();
    for(var i = 0; i < cotaNieve.length; i++){
        if(cotaNieve[i].periodo == "00-24" || cotaNieve[i].periodo == "00-12" || cotaNieve[i].periodo == "12-24"){
            continue;
        }
        pushDataToArray(cotaNieveData, cotaNieve[i].value);
    }
    return cotaNieveData;
}

function getRachaMax(rachaMax){
    var rachaMaxData = new Array();
    for(var i = 0; i < rachaMax.length; i++){
        if(rachaMax[i].periodo == "00-24" || rachaMax[i].periodo == "00-12" || rachaMax[i].periodo == "12-24"){
            continue;
        }
        pushDataToArray(rachaMaxData, rachaMax[i].value);
    }
    return rachaMaxData;
}

function getSensTermica(sensTerm){
    var sensTermData = new Array();
    for(var i = 0; i < sensTerm.datos.length; i++){
        pushDataToArray(sensTermData, sensTerm.datos[i].value);
    }
    return sensTermData;
}

```

(Fuente: Elaboración propia)

En sintonía con el caso anterior, también se ha extraído el título de cada columna de la tabla de uno de los datos que se va a mostrar en la misma. En este caso, se ha empleado el estado del cielo para extraer el periodo horario de 6 horas que se especificará como título de cada una de las columnas de la tabla.

Una vez indicada la extracción de los datos necesarios en el formato adecuado pasamos a detallar la creación de la tabla empleando el DOM. Para ello, simplemente debemos crear cada fila y columna de la tabla con los datos obtenidos empleando respectivamente, las etiquetas

HTML “<tr></tr>” y “<td></td>”. El título de cada columna se especificará mediante la etiqueta “<th></th>”.

Con tal de obtener estas filas y columnas, se ha creado una función que las devuelve en este formato distinguiendo entre el tipo de tabla que se va a crear. Es decir, esta función posee un parámetro que determina si la tabla que se va a crear se corresponde con la que se detallará debajo del gráfico o con la que se especificará cuando no se pueda crear el mismo por falta de datos. Esto es así ya que, dependiendo del caso, se llamará a unas funciones u otras según corresponda.

Una vez se han obtenido estas filas y columnas se crea la tabla según la estructura del DOM y se muestra en el pop-up. Cabe destacar que la tabla se coloca dentro de un contenedor “<div></div>” con el estilo en línea “overflow-x: auto” con la finalidad de mantener el diseño responsive en las tablas. De esta manera, si el contenido de la misma no se puede visualizar al completo, se muestra una barra de scroll lateral que permite visualizar el resto del contenido de una manera sencilla e intuitiva. Esta misma característica la comparten todas las tablas creadas y mostradas en WPCS.

En las siguientes figuras se muestra la creación de esta tabla según el formato especificado.

Figura 57. WPCS-Creación de filas y columnas de la tabla de predicción

```
function createTableRowsTD(tableColumnsValues, estCieloValues, cotaNieveValues, rachaMaxValues, sensTermOrProbPrecValues, type){  
    var tableColumns = "<th></th>";  
    var rowEstCielo = "<td><b>Estado cielo</b></td>";  
    var rowCotaNieve = "<td><b>Cota nieve (m)</b></td>";  
    var rowRachaMax = "<td><b>Racha máx (km/h)</b></td>";  
  
    var rowSensTermorProbPrec = "<td><b>Prob precipitación (%)</b></td>";  
    if(type == GRAPH_TABLE_TYPE){  
        rowSensTermorProbPrec = "<td><b>Sensación térmica (°C)</b></td>";  
    }  
  
    for(var i = 0; i < estCieloValues.length; i++){  
        tableColumns = tableColumns + "<th>" + tableColumnsValues[i] + "</th>"  
        rowEstCielo = rowEstCielo + "<td>" + estCieloValues[i] + "</td>"  
        rowCotaNieve = rowCotaNieve + "<td>" + cotaNieveValues[i] + "</td>"  
        rowRachaMax = rowRachaMax + "<td>" + rachaMaxValues[i] + "</td>"  
        rowSensTermorProbPrec = rowSensTermorProbPrec + "<td>" + sensTermOrProbPrecValues[i] + "</td>"  
    }  
  
    return [tableColumns, rowEstCielo, rowCotaNieve, rowRachaMax, rowSensTermorProbPrec];  
}
```

(Fuente: Elaboración propia)

Figura 58. WPCS-Obtención de filas y columnas de la tabla de predicción

```

function getTableRowsAndColumns(dayPredData, type){
    var [tableColumnsValues, estCieloValues, cotaNieveValues, rachaMaxValues, sensTermValues, probPrecValues] = "";
    var [tableColumns, tdEstCielo, tdCotaNieve, tdRachaMax, tdSensTermOrProbPrec] = "";
    if(type == GRAPH_TABLE_TYPE){
        [tableColumnsValues, estCieloValues] = getEstadoCielo(dayPredData.estadoCielo);
        cotaNieveValues = getCotaNieve(dayPredData.cotaNieveProv);
        rachaMaxValues = getRachaMax(dayPredData.rachaMax);
        sensTermValues = getSensTermica(dayPredData.sensTermica);
        [tableColumns, tdEstCielo, tdCotaNieve, tdRachaMax, tdSensTermOrProbPrec] = createTableRowsTD(tableColumnsValues, estCieloValues, cotaNieveValues, rachaMaxValues, sensTermValues, GRAPH_TABLE_TYPE);
    } else{
        [tableColumnsValues, estCieloValues, cotaNieveValues, rachaMaxValues, probPrecValues] = getTableData(dayPredData);
        [tableColumns, tdEstCielo, tdCotaNieve, tdRachaMax, tdSensTermOrProbPrec] = createTableRowsTD(tableColumnsValues, estCieloValues, cotaNieveValues, rachaMaxValues, probPrecValues, TABLE_TYPE);
    }
    return [tableColumns, tdEstCielo, tdCotaNieve, tdRachaMax, tdSensTermOrProbPrec];
}

```

(Fuente: Elaboración propia)

Figura 59. WPCS-Creación de la tabla de predicción

```

function createTable(dayPredData, type){
    var [tableColumns, tdEstCielo, tdCotaNieve, tdRachaMax, tdSensTermOrProbPrec] = getTableRowsAndColumns(dayPredData, type);
    var table = document.createElement("table");
    table.style="font-size: 15px; text-align: center";
    var thead = document.createElement("thead");
    var tbody = document.createElement("tbody");
    var firstTr = document.createElement("tr");
    var rowEstCielo = document.createElement("tr");
    var rowCotaNieve = document.createElement("tr");
    var rowRachaMax = document.createElement("tr");
    var rowSensTermOrProbPrec = document.createElement("tr");
    firstTr.innerHTML= tableColumns;
    rowEstCielo.innerHTML=tdEstCielo;
    rowCotaNieve.innerHTML=tdCotaNieve;
    rowRachaMax.innerHTML=tdRachaMax;
    rowSensTermOrProbPrec.innerHTML=tdSensTermOrProbPrec;
    thead.appendChild(firstTr);
    table.appendChild(thead);
    tbody.appendChild(rowEstCielo);
    tbody.appendChild(rowCotaNieve);
    tbody.appendChild(rowRachaMax);
    tbody.appendChild(rowSensTermOrProbPrec);
    table.appendChild(tbody);
    return table.outerHTML;
}

```

(Fuente: Elaboración propia)

3. Índice de radiación ultravioleta máxima

Una vez expuesta la tabla pasamos a exponer el índice de radiación ultravioleta máxima (UV) en el día. Para mostrarlo en el pop-up simplemente se llama a la función “getCorrectData(data)” expuesta en la Figura 45Figura 45 con el fin de obtener este dato siguiendo un formato adecuado en función de su existencia o no. Esta información, al corresponderse a un dato para las 24 horas del día, se muestra simplemente en un párrafo dentro del pop-up.

4. Información de copyright

Al igual que en la muestra al usuario de los datos de tiempo de una fecha pasada, se crea un párrafo con la información de copyright de Aemet junto con su respectivo enlace siguiendo el mismo formato indicado en la Figura 48.

Una vez explicadas cada una de las partes del pop-up que se especifica en esta situación, se muestra en las siguientes figuras el código empleado para generarlo y el aspecto visual del mismo en el sitio web.

Figura 60. WPCS-Creación del mensaje modal de predicción de tiempo (gráfico-código)

```
function showDayPredictionGraphToUser(dayPredData, date){
    console.log(dayPredData);
    Swal.fire({
        title: 'Predicción de tiempo',
        icon: 'info',
        html:
            '<div><canvas id="tempPredGraph" width="400" height="400"></canvas></div>' +
            '<div style="overflow-x: auto;">' +
            createTable(dayPredData, GRAPH_TABLE_TYPE) + '</div><br>' +
            '<p>Radiación ultravioleta máxima: ' + getCorrectData(dayPredData.uvMax) + '</p><br>' +
            getCopyrightInfo(),
        showCloseButton: true,
        showCancelButton: false,
        focusConfirm: false,
        confirmButtonText:
            '<i class="fa fa-thumbs-up"></i> Aceptar',
        confirmButtonAriaLabel: 'Thumbs up, great!',
        }).then(loadGraphs(dayPredData, date)).catch( (dismiss) => {}); // Empleamos una promesa para indicar que se cargaran los graficos cuando el mensaje modal se halle disparado
}
```

(Fuente: Elaboración propia)

Figura 61. WPCS-Creación del mensaje modal de predicción de tiempo (gráfico-web)



(Fuente: Elaboración propia)

Datos que pertenecen a 2 o 3 días siguientes al actual

En este caso no poseemos tanta información de predicción de tiempo como en el anterior. Es por esto por lo que los datos pasan a corresponderse a un periodo de 12 horas impidiéndonos, con ello, la interpretación de dicha información mediante mecanismos como los gráficos.

Para esta situación se ha empleado el uso de tablas que reflejen la evolución del tiempo en el día según el periodo horario establecido. Según esta especificación, la estructura de este pop-up presenta las siguientes características:

1. Tabla en la que se muestran los datos relativos al estado del cielo, la cota de nieve, la racha máxima de viento y la probabilidad de precipitación.
2. Tabla en la que se muestra la evolución en velocidad y dirección del viento.
3. Tabla en la que se muestran los valores máximos y mínimos de la temperatura, la sensación térmica y la humedad relativa.
4. Radiación ultravioleta máxima en el día.
5. Información de copyright y enlace a Aemet.

En similitud al resto de pop-ups, exponemos punto por punto la generación de cada parte del mismo.

1. Tabla correspondiente al estado cielo, cota nieve, racha máx. y prob. precip.

Esta tabla se genera de la misma manera que la expuesta en el caso anterior, de forma en que se utiliza el DOM para poder crearla. Debido a que se ha generado esta clase de tabla de manera dinámica, el número de columnas se adapta al periodo de 12 horas especificado en este caso sin ningún inconveniente. Como no se poseen datos para un intervalo de 12 horas para el caso de la sensación térmica, este dato es sustituido por el de la probabilidad de precipitación en esta tabla.

También se indica que aquellos datos que en el caso anterior presentaban una longitud de 7 hora poseen una longitud de 3 correspondientes a la información referente a cada periodo de 12 horas en el día y a las 24 horas del día. Los datos que poseían un tamaño de 4 hora presentan una longitud de 0, ya que la información que reflejan no se corresponde con un periodo de 12 horas sino con el de las 24 horas del día.

Siguiendo esta especificación se ha creado una única función que recopila directamente todos los datos de esta tabla al tener solamente un único periodo de tiempo que descartar. La generación de los títulos de las columnas se ha realizado siguiendo el mismo esquema empleando para ello el periodo detallado en la información relativa al estado del cielo.

Figura 62. WPCS-Extracción de todos los datos para la tabla de predicción 2

```
function getTableData(dayPredData){  
    var estadoCieloData = new Array();  
    var cotaNieveProvData = new Array();  
    var probPredData = new Array();  
    var rachaMaxData = new Array();  
    var tableColumns = new Array();  
    for(var i = 0; i < dayPredData.probPrecipitacion.length; i++){  
        if(dayPredData.estadoCielo[i].periodo == "00-24" || dayPredData.cotaNieveProv[i].periodo == "00-24" || dayPredData.probPrecipitacion[i].periodo == "00-24" || dayPredData.rachaMax[i].periodo == "00-24"){  
            continue;  
        }  
        tableColumns.push(dayPredData.estadoCielo[i].periodo + "h");  
        pushDataToArray(estadoCieloData, dayPredData.estadoCielo[i].descripcion);  
        pushDataToArray(cotaNieveProvData, dayPredData.cotaNieveProv[i].value);  
        pushDataToArray(rachaMaxData, dayPredData.rachaMax[i].value);  
        pushDataToArray(probPredData, dayPredData.probPrecipitacion[i].value);  
    }  
    return [tableColumns, estadoCieloData, cotaNieveProvData, rachaMaxData, probPredData];  
}
```

(Fuente: Elaboración propia)

Esta función en llamada en la *Figura 58* para generar todas las filas y las columnas de la tabla siguiendo el formato del DOM y empleando las etiquetas “<th></th>” y “<td></td>”. La generación de esta tabla se realiza siguiendo la misma estructura mostrada en la *Figura 59*.

2. Tabla del viento

Una vez detallada la creación de la tabla anterior se explica la generación de esta nueva tabla en la que se alojarán los datos relativos a la evolución del viento. Su formato es muy similar al de la tabla anterior, con la diferencia de que el viento tiene una característica adicional denominada dirección. Es por esto por lo que se ha extraído el viento en una tabla propia adicional con tal de mostrar su información de una manera clara y concisa, ya que añadir una nueva columna a la tabla anterior en la que solo tuviera valor para el viento podría ser el causante de una confusión por parte del usuario.

Para generar esta tabla se emplea el DOM y las filas pasan a representar exclusivamente la dirección del viento y su velocidad. Se mantiene el formato de columnas correspondiente al periodo horario de 12 horas, de manera que se extrae dicho periodo de la información relativa al viento en semejanza a la situación anterior.

En las siguientes figuras se muestran las funciones correspondientes a la extracción de los datos del viento y a la creación de esta tabla.

Figura 63. WPCS-Extracción de los datos del viento para la tabla de predicción

```
function getVientoDataForTable(viento){  
    var tableColumns = new Array();  
    var vientoDirData = new Array();  
    var vientoVelData = new Array();  
    for(var i = 0; i < viento.length; i++){  
        if(viento[i].periodo == "00-24"){  
            continue;  
        }  
        tableColumns.push(viento[i].periodo + "h");  
        pushDataToArray(vientoDirData, viento[i].direccion);  
        pushDataToArray(vientoVelData, viento[i].velocidad);  
    }  
    return [tableColumns, vientoDirData, vientoVelData];  
}
```

(Fuente: Elaboración propia)

Figura 64. WPCS-Creación de filas y columnas de la tabla del viento

```
function createVientoTableRows(viento){  
    var [tableColumnsValues, vientoDirValues, vientoVelValues] = getVientoDataForTable(viento);  
    var tableColumns = "<th></th>";  
    var vientoDir = "<td><b>Dirección</b></td>";  
    var vientoVel = "<td><b>Velocidad (km/h)</b></td>";  
    for(var i = 0; i < tableColumnsValues.length; i++){  
        tableColumns = "<th>" + tableColumnsValues[i] + "</th>";  
        vientoDir = vientoDir + "<td>" + vientoDirValues[i] + "</td>";  
        vientoVel = vientoVel + "<td>" + vientoVelValues[i] + "</td>";  
    }  
    return [tableColumns, vientoDir, vientoVel];  
}
```

(Fuente: Elaboración propia)

Figura 65. WPCS-Creación de la tabla de predicción del viento

```
function createVientoTable(viento){  
    var [tableColumns, rowVientoDirData, rowVientoVelData] = createVientoTableRows(viento);  
    var table = document.createElement("table");  
    table.style="font-size: 15px; text-align: center";  
    var legend = document.createElement("legend");  
    var thead = document.createElement("thead");  
    var tbody = document.createElement("tbody");  
    var firstTr = document.createElement("tr");  
    var rowVientoDir = document.createElement("tr");  
    var rowVientoVel = document.createElement("tr");  
    legend.innerHTML= "Viento";  
    firstTr.innerHTML= tableColumns;  
    rowVientoDir.innerHTML= rowVientoDirData;  
    rowVientoVel.innerHTML= rowVientoVelData;  
    table.appendChild(legend);  
    thead.appendChild(firstTr);  
    table.appendChild(thead);  
    tbody.appendChild(rowVientoDir);  
    tbody.appendChild(rowVientoVel);  
    table.appendChild(tbody);  
    return table.outerHTML;  
}
```

(Fuente: Elaboración propia)

3. Tabla de valores mínimos y máximos

Tras exponer la creación de la tabla correspondiente a la evolución del viento, pasamos a detallar la generación de esta nueva tabla. Esta tabla se crea para aquella información de las que solo se posean valores para las 24 horas del día, de forma que dichos valores no se ubican en ningún periodo en concreto sino que simplemente reflejan una estimación del valor mínimo y el valor máximo que pueden alcanzar a lo largo del día. Es por esta razón por la que esta es la

única tabla que se puede mantener para la situación en la que tengamos menos información sobre la predicción de tiempo que haga en el día.

Su elaboración mantiene la utilización del DOM, con la diferencia de que en este caso no formaremos las columnas extrayendo el periodo de tiempo al que pertenecen los datos, sino que estas columnas pasarán a ser directamente los valores mínimos y máximos al ser ahora valores unitarios. Con tal de representar estos datos siguiendo un formato adecuado se ha empleado la función “getCorrectData(data)” representada en la *Figura 45* dependiendo de la existencia de dicha información.

A continuación, se muestra la figura en la que se refleja la función empleada para la generación de esta tabla.

Figura 66. WPCS-Creación de la tabla de predicción de valores mínimos y máximos

```
function createMaxMinTable(dayPredData) {
    var table = document.createElement("table");
    table.style="font-size: 15px; text-align: center";
    var thead = document.createElement("thead");
    var tbody = document.createElement("tbody");
    var rowMin = document.createElement("tr");
    var rowTemp = document.createElement("tr");
    var rowsenTerm = document.createElement("tr");
    var rowHumRel = document.createElement("tr");
    var rowHumRel = document.createElement("tr");
    firstTr.innerHTML = "<th></th><th>Mínima</th><th>Máxima</th>";
    rowTemp.innerHTML = "<td><b>Temperatura ("</b></td><td>" + getCorrectData(dayPredData.temperatura.minima) + "</td><td>" + getCorrectData(dayPredData.temperatura.maxima) + "</td>"";
    rowsenTerm.innerHTML = "<td><b>Sensación térmica ("</b></td><td>" + getCorrectData(dayPredData.sensTermica.minima) + "</td><td>" + getCorrectData(dayPredData.sensTermica.maxima) + "</td>"";
    rowHumRel.innerHTML = "<td><b>Humedad relativa (%)</b></td><td>" + getCorrectData(dayPredData.humedadRelativa.minima) + "</td><td>" + getCorrectData(dayPredData.humedadRelativa.maxima) + "</td>"";
    thead.appendChild(firstTr);
    thead.appendChild(rowTemp);
    tbody.appendChild(rowsenTerm);
    tbody.appendChild(rowHumRel);
    table.appendChild(tbody);
    return table.outerHTML;
}
```

(Fuente: Elaboración propia)

4. Radiación UV máxima

De la misma manera que en el caso anterior, se emplea un párrafo en el pop-up para especificar esta información según el formato visualizado anteriormente.

5. Información de copyright

Se utiliza un párrafo para detallar esta información según la misma estructura de los casos anteriores y que se puede visualizar en la *Figura 48*.

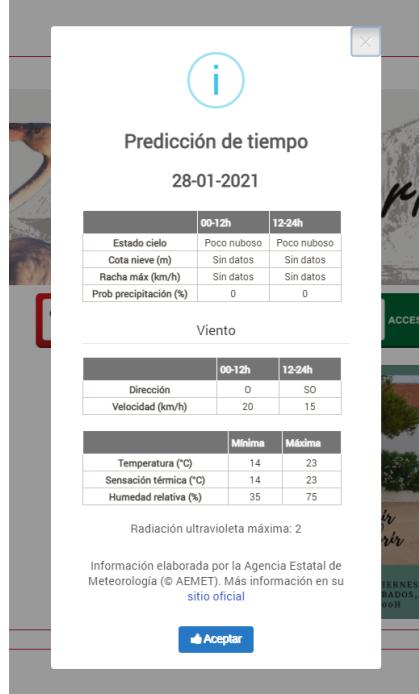
Una vez expuestos todos los puntos necesarios para generar este pop-up se muestra, en las siguientes figuras, el código empleado para generarlo y su resultado visual en la web.

Figura 67. WPCS-Creación del mensaje modal de predicción de tiempo (tablas-código)

```
function showDayPredictionTableToUser(dayPredData, date){
    console.log(dayPredData);
    Swal.fire({
        title: 'Predicción de tiempo',
        icon: 'info',
        html:
            '<h3 style="font-size: 1.5em;">' + date + '</h3><br>' +
            '<div style="overflow-x: auto;">' +
            createTable(dayPredData, TABLE_TYPE) + '</div><br>' +
            '<div style="overflow-x: auto;">' +
            createVientoTable(dayPredData.viento) + '</div><br>' +
            '<div style="overflow-x: auto;">' +
            createMaxMinTable(dayPredData) + '</div><br>' +
            '<p>Radiación ultravioleta máxima: ' + getCorrectData(dayPredData.uvMax) + '</p><br>' +
            getCopyrightInfo(),
        showCancelButton: true,
        showCloseButton: false,
        focusConfirm: false,
        confirmButtonText:
            '<i class="fa fa-thumbs-up"></i> Aceptar',
        confirmButtonAriaLabel: 'Thumbs up, great!',
    });
}
```

(Fuente: Elaboración propia)

Figura 68. WPCS-Creación del mensaje modal de predicción de tiempo (tablas-web)



(Fuente: Elaboración propia)

Datos que pertenecen a 4, 5 o 6 días siguientes al actual

Este es el caso en el que menos información de predicción de tiempo poseemos. La información de tiempo proporcionada por Aemet en esta situación corresponde al periodo de tiempo de las 24 horas del día, por lo que no es posible realizar ninguna clase interpretación mediante mecanismos como gráficos o tablas ya que solo se obtiene un valor unitario representativo de todo el día.

La única tabla que se puede mantener es la que refleja los valores máximos y mínimos debido a que representan a la totalidad del día tal y como se ha indicado anteriormente. Siguiendo estas características se muestra la estructura que poseerá este último pop-up:

1. Tabla con los valores máximos y mínimos de la temperatura, la sensación térmica y la humedad relativa.
2. Párrafos en los que se muestran el resto de datos de predicción de tiempo para las 24 horas del día.
3. Información de copyright y enlace a Aemet.

Una vez más detallamos, punto por punto, la generación de cada representación mostrada en el pop-up.

1. Tabla de valores mínimos y máximos

Esta tabla presentará el mismo formato expuesto para el caso anterior y se puede visualizar el código empleado para generarla en la *Figura 66*.

2. Párrafos con el resto de información

Para el resto de datos a especificar en el pop-up simplemente se emplea la función denominada “getCorrectData(data)” mostrada en la *Figura 45* con tal de representar estos datos con un formato adecuado atendiendo a su existencia. Este mismo procedimiento es el que se ha empleado en el resto de pop-ups para mostrar el valor “Sin datos” en caso de que no se conozca la estimación correspondiente a cada dato en cuestión. Se ubican en este punto los datos referentes al estado del cielo, la probabilidad de precipitación, la dirección, la velocidad y la racha máxima del viento, la cota de nieve y la radiación UV máxima en el día.

3. Información de copyright

Al igual que en los casos anteriores, se muestra un párrafo con la información del copyright de Aemet junto con su respectivo enlace siguiendo el formato especificado en la *Figura 48*.

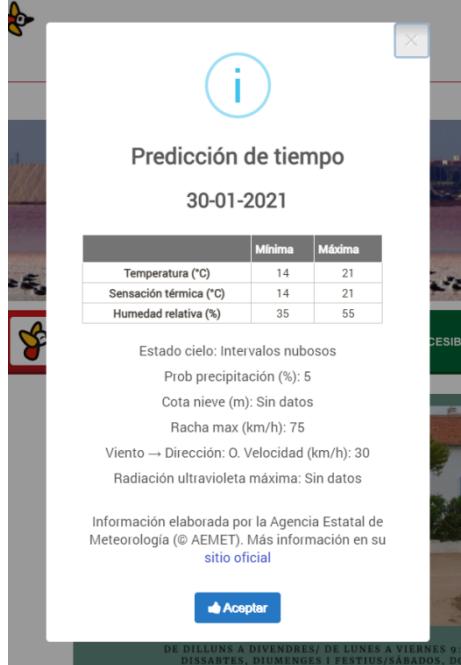
Tras detallar todos los puntos que conforman este último pop-up se muestra, en las siguientes figuras, el código empleado para generarlo y su resultado final en el sitio web.

Figura 69. WPCS-Creación del mensaje modal de predicción de tiempo (párrafos-código)

```
function showDayPredictionToUser(dayPredData, date){
    console.log(dayPredData);
    Swal.fire({
        title: '<strong>Predicción de tiempo</strong>',
        icon: 'info',
        html:
        '<h3 style="font-size: 1.5em;">' + date + '</h3><br>' +
        '<div style="overflow-x: auto;">' +
        createDayMinTable(dayPredData) + '</div><br>' +
        '<p>Estado cielo: ' + getCorrectData(dayPredData.estadoCielo[0].descripcion) + '</p>' +
        '<p>Prob precipitación (%): ' + getCorrectData(dayPredData.probPrecipitacion[0].value) + '</p>' +
        '<p>Cota nieve (m): ' + getCorrectData(dayPredData.cotaNieveProv[0].value) + '</p>' +
        '<p>Racha max (km/h): ' + getCorrectData(dayPredData.rachaMax[0].value) + '</p>' +
        '<p>Viento → Dirección: ' + getCorrectData(dayPredData.viento[0].direccion) + ". Velocidad (km/h): " + getCorrectData(dayPredData.viento[0].velocidad) +
        '<p>Radiación ultravioleta máxima: ' + getCorrectData(dayPredData.uvMax) + '</p><br>' +
        getCopyrightInfo(),
        showCloseButton: true,
        showCancelButton: false,
        focusConfirm: false,
        confirmButtonText:
        '<i class="fa fa-thumbs-up"></i> Aceptar',
        confirmButtonAriaLabel: 'Thumbs up, great!',
    });
}
```

(Fuente: Elaboración propia)

Figura 70. WPCS-Creación del mensaje modal de predicción de tiempo (párrafos-web)



(Fuente: Elaboración propia)

Y con esto concluye la explicación detallada sobre la creación del user-script denominado WPCS.

En el siguiente punto se detallan los problemas encontrados a la hora de desarrollar el susodicho script y la resolución que se ha adoptado en cada caso.

6.3.5. Problemas encontrados

A lo largo del desarrollo de WPCS han ido apareciendo una serie de problemas que han dificultado su realización. A continuación, se detalla una lista con los problemas más relevantes:

1. Enlazado de la librería “dms.js”.
2. Muestra de los datos de viento en el gráfico de líneas.
3. Cambios en la URL del sitio web.

Una vez detallada la lista pasamos a explicar, punto por punto, el problema junto con la solución empleada.

1. Enlazado de la librería “dms.js”

A la hora de enlazar determinadas librerías javascript podemos encontrarnos con la problemática de su compatibilidad con el código que se va a desarrollar. Con tal de enlazar librerías en un user-script de manera en que estas se carguen antes que el código desarrollado en cuestión, se debe de especificar su cargado mediante la etiqueta “require” en la cabecera del mismo.

Sin embargo, importar esta librería siguiendo este esquema generaba un error de compilación en WPCS debido a que “dms.js” sigue una estructura modular según el estándar “ECMAScript 6” o “ES6 [50]”. El error era producido por la sentencia “*export* [51]” ubicada en la línea 332 de la librería que permite la exportación del código de la misma en forma de módulos que se pueden cargar en otro archivo javascript. El problema se encuentra en que, para poder cargar estos módulos en otro script, dicho script debe de ser también de tipo modular para que acepte esta clase de sintaxis, característica que no se podía especificar para un user-script.

Según “ES6”, la sentencia “*export*” no puede ser utilizada en scripts embebidos, es decir, no se puede emplear un código javascript que contenga esta sentencia directamente en otro script sin especificar previamente en el HTML que este script es de tipo modular. Dicha especificación se realizaría indicando en la etiqueta “`<script></script>`” correspondiente al código que va a importar esta librería, el atributo `type="module"`. De esta manera, el navegador comprendería que son módulos o partes de código y no un script completo tradicional que no emplea esta sintaxis.

Esta podría ser la solución para cuando se está creando una página web desde 0 ya que se poseería el control total sobre el HTML del sitio web por lo que esta solución sería fácilmente

aplicable. Sin embargo, en el caso de los user-scripts esta solución no es válida ya que son fragmentos de código que se ejecutan a través de un manejador de user-scripts que hay que instalar en el navegador. Es este motivo por el cual se ejecuta en otro ámbito dentro del procesamiento realizado por el navegador.

Esto quiere decir que, aunque se importara este script de manera dinámica en el HTML a través de nuestro user-script no serviría de nada, ya que los enlaces referentes al código de la librería no serían accesibles desde el user-script al haber importado esta librería en el ámbito en el que se encuentra el código nativo de la página web. Además, dicha importación debería realizarse antes de que se ejecute el código del mismo porque es un requisito para su funcionamiento. Este último detalle ya se ha solventado empleando la etiqueta “require” antes mencionada.

Expuesta toda esta problemática sólo queda indicar la solución que se ha adoptado en este caso. Tampermonkey posee una interfaz que permite realizar modificaciones locales a los archivos que se importen en los scripts creados mediante esta herramienta. Esto nos permite comentar o eliminar ciertas partes de código que nos puedan generar problemas sin alterar el código original de la librería enlazado mediante su URL. Sabiendo esto, y con el fin de alterar lo menos posible el código de la librería original, simplemente comentamos de manera local la línea 332 de esta librería.

Para realizar este cambio hay que acceder a la categoría “Externals” de la interfaz de Tampermonkey y seleccionar el botón de “Editar” de la librería que se quiera modificar. Cabe destacar que es posible realizar esta acción debido a que la licencia se aloja bajo una licencia de tipo MIT que permite realizar modificaciones al código de esta librería sin necesidad de notificárselo a sus creadores. Si tuviera otra licencia ya habría que tomar otra serie de medidas con tal de respetar el acuerdo de legalidad al que está sujeta la misma.

Tras realizar esta acción, el código se enlaza correctamente y se puede utilizar debido a que el navegador ya no identifica este script como un módulo sino como un script tradicional que se puede importar normalmente sin necesidad de ninguna especificación adicional. Se indica que esta solución es válida en el posible caso de publicación del script, ya que cuando se genere su archivo “.zip” necesario para su instalación en otros navegadores, Tampermonkey realiza una copia de las librerías y los recursos que se requieran para el correcto funcionamiento del script.

Por este motivo, obtendríamos una copia de la librería en formato javascript con la línea 332 ya comentada.

Otra solución hubiera sido directamente copiar y pegar el código necesario de la librería en WPCS ya que la licencia permitiría esta acción también. Sin embargo, se ha optado por la solución de importar la librería con la finalidad de evitar realizar más código del estrictamente necesario para la correcta elaboración del script.

2. Muestra de los datos de viento en el gráfico de líneas

El gráfico realizado mediante la librería “Chart.js” posee una serie de eventos que permiten visualizar los datos del gráfico de una manera más detallada. En el caso de un gráfico de líneas, “Chart.js” implementa el evento “onhover” y “onclick” para cada uno de los puntos que conforman el gráfico en cuestión. Para el caso de mostrar datos como la temperatura no es necesaria hacer ninguna clase modificación, debido a que, internamente, ya se muestra en esos eventos el nombre del gráfico que se está visualizando junto con el valor que se está consultando para el “eje x”, quien representa su variación en el tiempo.

Sin embargo, para el caso del viento solo mostraba su valor de velocidad, ya que es con el único valor con el que se podía establecer un gráfico de este tipo al ser un valor numérico. Repasando los datos de predicción obtenidos para el viento averiguamos que también se detalla la dirección en la que este sopla en ese periodo horario. Es por esta razón por la que, si se deseaba mostrar esta información al usuario había que realizar modificaciones en las opciones del gráfico remarcadas por esta librería.

La solución empleada para mostrar una información personalizada cuando se ejecutan esta clase de eventos sobre el gráfico ha sido la de generar una función *callback de Chart.js* [52] a la cual se llame cuando se disparen estos eventos. En esta función se recogerían los datos relativos al punto del gráfico que ha provocado la llamada a esta función callback de manera que si se corresponde con un punto correspondiente al gráfico del viento, se realicen las modificaciones pertinentes a la salida de dicho callback.

Esta función modificaría la salida para cada punto del gráfico del viento de manera en que se muestra la velocidad y la dirección del mismo en ese intervalo de tiempo. De este modo, se le muestra al usuario toda la información obtenida sobre la predicción de tiempo para el día seleccionado. A continuación se adjunta una figura en la que se muestra la generación concreta de esta función callback. Cabe destacar que esta función se puede visualizar también en la *Figura 55*.

Figura 71. WPCS-Problemas encontrados (gráfico de líneas)

```
tooltips: {
    callbacks: [
        label: function(tooltipItem, data) {
            var label = data.datasets[tooltipItem.datasetIndex].label + ": " + tooltipItem.value;
            if(data.datasets[tooltipItem.datasetIndex].label == dataGraphNames[3]){
                label = data.datasets[tooltipItem.datasetIndex].label + " + Dirección: " + vientoDirs[tooltipItem.datasetIndex] + ". Velocidad: " + tooltipItem.value + " km/h";
            }
            return label;
        }
    ],
    title: false
};
```

(Fuente: Elaboración propia)

3. Cambios en la URL del sitio web

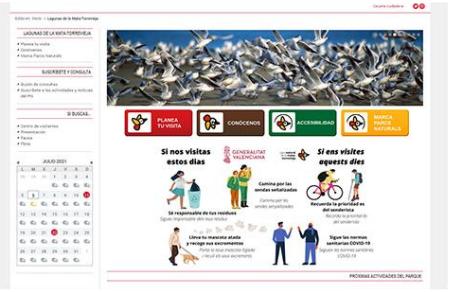
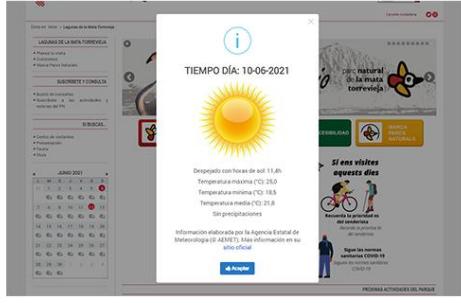
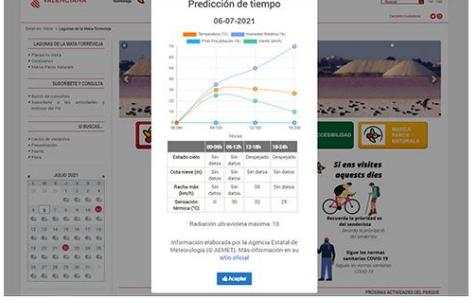
Este problema puede surgir a lo largo del tiempo y consiste en que el sitio web original modifica los componentes de su URL. En este caso, la única solución consiste en corregir manualmente las URL's especificadas en la etiqueta “match” de la cabecera del user-script. Esto sucede debido a que esta etiqueta identifica las URL's en donde se ejecutará el script en cuestión.

6.3.6. Visualización de cambios

Tras exponer todas las características y problemas encontrados a la hora de desarrollar WPCS se detalla la evolución mostrada en el sitio web cuando se utiliza este user-script.

En este caso, los cambios se especifican tal y como se ha comentado anteriormente, en el calendario del sitio web. En la siguiente figura se visualiza el aspecto original de la web comparada con la nueva muestra de datos observada tras la activación de WPCS.

Figura 72. WPCS-Comparativa de activación del user-script
WPCS

Sin activar	Activado
	
En ejecución 1	En ejecución 2
	

(Fuente: Elaboración propia)

Como se pueda observar, hay un incremento notable de la información que puede extraer el usuario incluyendo información sobre el tiempo atmosférico que se predice para una fecha futura. Dicha información se acrecienta al incluir también la opción de adquirir datos relativos al tiempo que hizo en una fecha pasada.

En el siguiente punto se detallarán las pruebas realizadas para comprobar el correcto funcionamiento de WPCS.

6.3.7. Pruebas realizadas

WPCS ha sido probado en diferentes navegadores con el fin de comprobar su correcto funcionamiento. Para ello, se han comprobado una gran diversidad de situaciones para así observar el comportamiento del script. Dichas pruebas son:

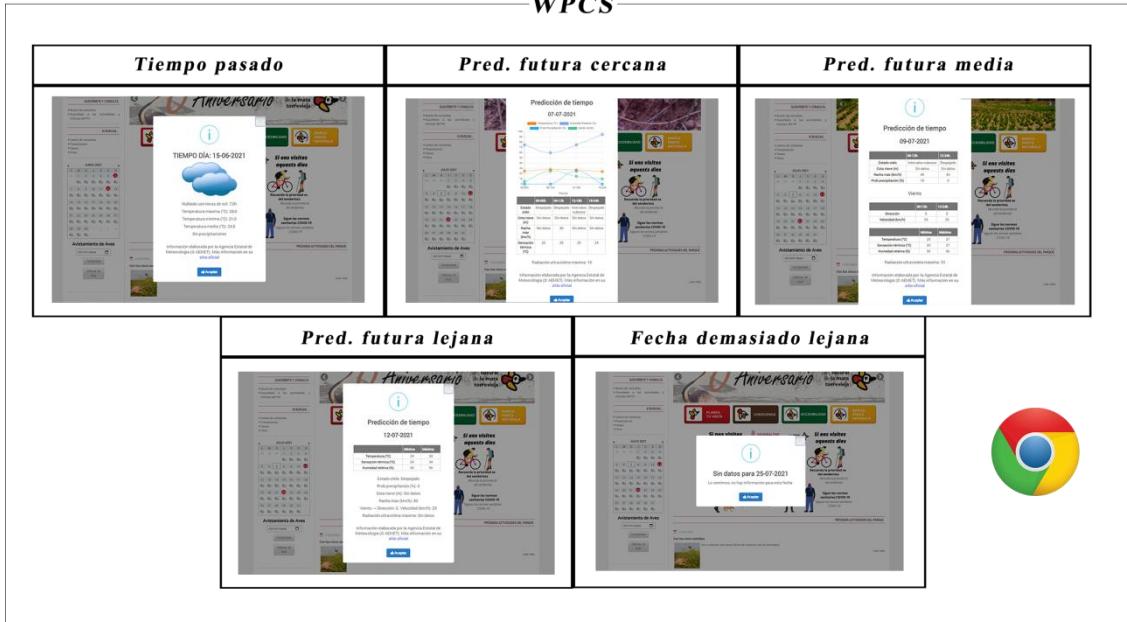
- Obtener información de una fecha pasada.

- Obtener predicción de una fecha futura cercana.
- Obtener predicción de una fecha futura media.
- Obtener predicción de una fecha futura lejana.
- Obtener el mensaje modal de error ante la consulta de una predicción en una fecha muy lejana.

Debido a que el lenguaje empleado para la realización de este proyecto es javascript se determina que el comportamiento de estos scripts no debe variar en los navegadores que posean la extensión Tampermonkey disponible para ser instalada.

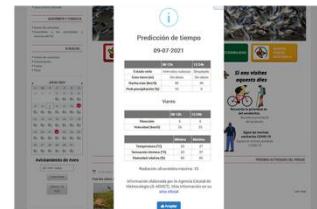
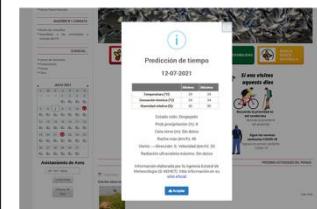
Estas comprobaciones se han realizado en los navegadores más conocidos: Google Chrome, Mozilla Firefox y Opera. Y como se puede observar en las siguientes figuras, la comprobación ha resultado en un éxito con respecto al funcionamiento de WPCS.

*Figura 73. WPCS-Pruebas en navegador Google Chrome
WPCS*



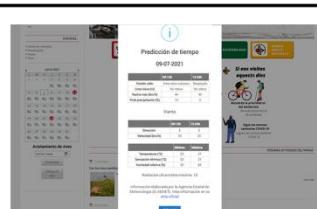
(Fuente: Logotipo de Google Chrome [14])

Figura 74. WPCS-Pruebas en navegador Mozilla Firefox
WPCS

Tiempo pasado	Pred. futura cercana	Pred. futura media
		
Pred. futura lejana	Fecha demasiado lejana	
		

(Fuente: Logotipo de Mozilla Firefox [15])

Figura 75. WPCS-Pruebas en navegador Opera
WPCS

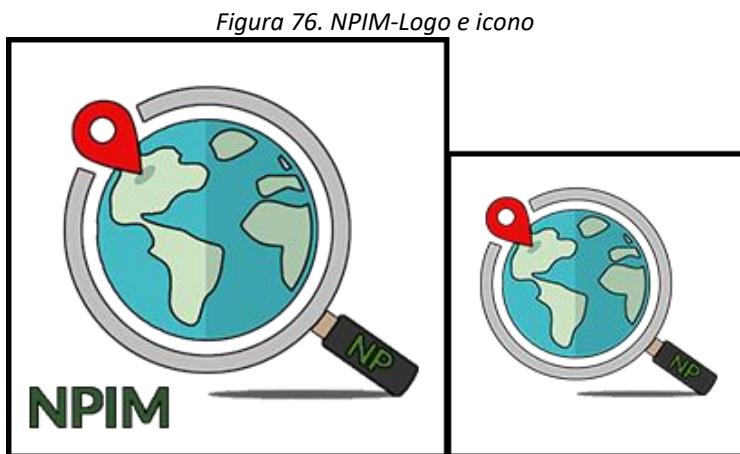
Tiempo pasado	Pred. futura cercana	Pred. futura media
		
Pred. futura lejana	Fecha demasiado lejana	
		

(Fuente: Logotipo de Opera [16])

Todas las funcionalidades comentadas operan correctamente en los 3 navegadores utilizados y muestran los resultados con un estilo casi idéntico debido a la especificación CSS de las librerías empleadas tal y como se observa en las figuras anteriores.

7. Natural Park Interactive Map (NPIM)

El propósito de este script consiste en la realización de un mapa interactivo que sirva para englobar toda la información relativa a las rutas del parque natural mostrada en la web. En los siguientes puntos se detalla la recopilación de datos realizada junto con la implementación de dicho mapa interactivo.



(Fuente: Icono de lupa con ubicación [17])

7.1. Tecnologías utilizadas

Para la realización de este user-script se ha empleado la librería *Leaflet* [53] encargada del tratamiento de mapas junto con *jQuery* [54]. A continuación, se realiza una breve descripción de ambas tecnologías:

Leaflet: Es también una librería de tratamiento de mapas open source basada en la simplicidad, el rendimiento y la usabilidad. Pretende mantenerse lo más ligera posible (33 KB) centrándose para ello en el desarrollo de un conjunto básico de características. Presenta una amplia documentación y es utilizada por otras tecnologías en sus desarrollos como por ejemplo “*MapBox* [55]” o “*Carto.js* [56]”.

jQuery: Es una biblioteca javascript creada inicialmente por *John Resig* y que permite simplificar la manipulación de documentos HTML junto con la tecnología AJAX. Además, permite optimizar el código empleado para generar funciones javascript de manera en que

estas, presentan una semántica más simple y reducida que si se implementan en su versión nativa. Su versión más reciente es la 3.6.0 y es una de las tecnologías javascript más utilizada dentro de la industria y el desarrollo web.

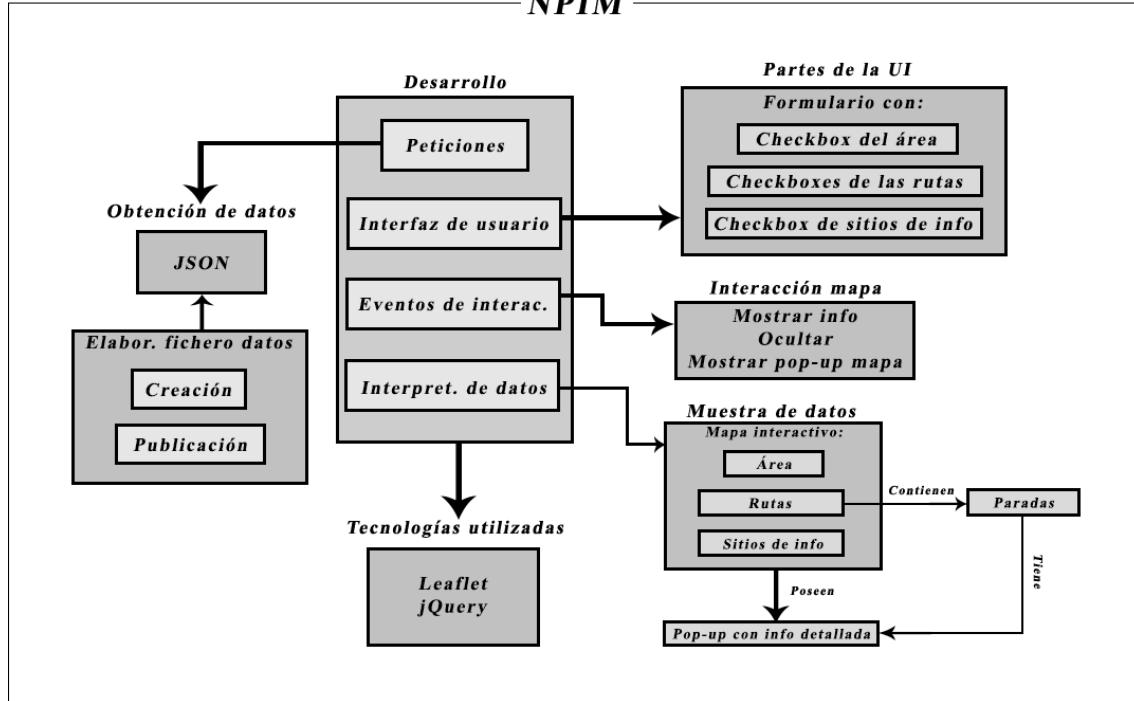
7.2. Desarrollo

El desarrollo de este user-script se divide en 4 secciones o apartados:

1. Obtención de los datos del sitio web y almacenamiento de los mismos en un fichero JSON.
2. Obtención de dicho JSON mediante una petición GET a GitHub.
3. Elaboración del mapa interactivo según los datos existentes en el JSON obtenido.
4. Creación de la UI que le permitirá al usuario interactuar con dicho mapa interactivo.

Realizaremos la exposición de NPIM especificando detalladamente cada uno de los puntos anteriores. Con tal de indicar de una mejor manera el desarrollo mediante el cual se ha efectuado el user-script NPIM, se muestra en la siguiente figura su arquitectura.

Figura 77. NPIM-Arquitectura



(Fuente: Elaboración propia)

7.2.1. Creación del fichero JSON con los datos del sitio web

Con tal de separar la capa de datos del script de la capa de implementación, se elaboró un fichero JSON con los datos existentes en el sitio web. Se extrajo la información obtenida en el apartado de “Planea tu Visita/Rutas” y “Planea tu Visita/Centro de Visitantes” de dicho sitio.

Además, se utilizó el visor *Cartográfico del Instituto Cartográfico de Valencia* [57] enlazado en el apartado de “Planea tu Visita/Rutas” para la extracción de las coordenadas que se necesitaron para la creación de cada elemento en el mapa interactivo. Cabe destacar que este fichero cumple la especificación JSON RFC 8259.

A continuación, detallamos la estructura de la propuesta de fichero JSON creada según la información obtenida:

1. Información general del archivo JSON como nombre, descripción, nombre del parque natural y licencia de los datos recopilados.
2. Área del parque natural, tanto en coordenadas como en valor en hectáreas o “ha”.
3. Rutas del parque natural con toda la información asociada a las mismas, desde su descripción hasta las paradas que estas posean.
4. Lugares de información ubicados en el parque natural junto con los detalles que ofrece el sitio web sobre ellos.

Una vez especificada la estructura que tendrá el fichero JSON pasamos a detallar cada uno de los metadatos que poseerá cada parte de dicha estructura.

1. Información general del archivo JSON

Esta sección del archivo JSON posee los siguientes metadatos:

- **name:** Nombre del archivo JSON. Dato de tipo string.
- **description:** Descripción general de los datos que recopila este archivo. Dato de tipo string.
- **natPark:** Nombre del parque natural al que hacen referencia los datos recopilados. Dato de tipo string.
- **mun:** Municipio en el que se aloja el parque natural. Dato de tipo string.
- **author:** Autor del archivo JSON. Dato de tipo string.
- **license:** Licencia y copyright de los datos recopilados en el archivo JSON. Dato de tipo string.
- **code:** Codificación de la información recopilada en el JSON. Dato de tipo string.
- **posGPS:** Posición geográfica del parque natural. Dato de tipo array de floats de tamaño 2, correspondientes a latitud y longitud.

A continuación, se muestra una figura en la que se visualizan estos metadatos recopilados para el parque natural Lagunas de la Mata en Torrevieja.

Figura 78. NPIM-Archivo JSON (datos generales)

```

"name": "Natural Park Interactive Map Data",
"description": "Este archivo 'JSON' recopila los datos relativos al área que cubre el parque natural junto con sus lugares de información. También se detallan las coordenadas GPS del mismo.",
"natPark": "Lagunas de la Mata y Torrevieja",
"mun": "Torrevieja",
"author": "Francisco Javier García Fernández",
"license": "@ Institut Cartogràfic Valencià, © 2015, Generalitat Parques Naturales de la Comunitat Valenciana - Lagunas de la Mata-Torrevieja Conselleria",
"code": "utf-8",
"posGPS": [38.0122, -0.709444],

```

(Fuente: Elaboración propia)

2. Área del parque natural

En este apartado, los metadatos son:

- **area:** Array de datos con toda la información relativa al área del parque natural.
 - ◊ **value:** Valor en hectáreas (ha) del área. Dato de tipo float.
 - ◊ **edge:** Conjunto de coordenadas correspondientes a la totalidad del área del parque natural. Utilizado para definir un reborde que indique las limitaciones del mismo. Se corresponde con un array de objetos JSON conformado a su vez por un array de floats de tamaño 2 especificando latitud y longitud por cada punto perteneciente a dicho reborde.

A continuación, se muestra la sección del JSON correspondiente a este apartado.

Figura 79. NPIM-Archivo JSON (área)

```

"area": {
  "value": 3717.3,
  "edge": [
    [38.020538, -0.736028], [38.030815, -0.717660], [38.031220, -0.718175], [38.031896, -0.715885], [38.040549, -0.719833], [38.045687, -0.713540], [38.044605, -0.709877], [38.045363, -0.706330], [38.048120, -0.700322], [38.046633, -0.698949], [38.046633, -0.696891], [38.048796, -0.691739], [38.047579, -0.682469], [38.049877, -0.680753], [38.050013, -0.679208], [38.049666, -0.676289], [38.050824, -0.675431], [38.051094, -0.671981], [38.052581, -0.670968], [38.053040, -0.666848], [38.046656, -0.666676], [38.042940, -0.669571], [38.057467, -0.666761], [38.051554, -0.655391], [38.078449, -0.658301], [38.036691, -0.656591], [38.025948, -0.657106], [38.024054, -0.659381], [38.023311, -0.659424], [38.022296, -0.661312], [38.021417, -0.662943], [38.020538, -0.664960], [37.964124, -0.711486], [37.965207, -0.713712], [37.965613, -0.715600], [37.968596, -0.722981], [37.972108, -0.727618], [37.973191, -0.729389], [37.977155, -0.739813], [37.980295, -0.739117], [37.981445, -0.738431], [37.983069, -0.741349], [37.983474, -0.742894], [37.985233, -0.742722], [37.986180, -0.744439], [37.988616, -0.743066], [37.992539, -0.750796], [37.994298, -0.751305], [37.995651, -0.752679], [38.005931, -0.749589], [38.011341, -0.751134], [38.011071, -0.749932], [38.020538, -0.737573]
  ]
}

```

(Fuente: Elaboración propia)

3. Rutas del parque natural

En este apartado se muestra toda la información recopilada de las posibles rutas del parque natural. A continuación, se especifica la estructura con la que se recopilará dicha información:

- **routes:** Array de objetos JSON correspondientes a toda la información recopilada para las diferentes rutas del parque natural.
 - ◊ **name:** Nombre de la ruta. Dato de tipo string.
 - ◊ **description:** Descripción general de la ruta. Se especifica información relativa a en qué consiste la misma. Dato de tipo string.
 - ◊ **dist:** Distancia del recorrido total de la ruta. Dato de tipo float.
 - ◊ **dur:** Duración en minutos del recorrido. Dato de tipo float.
 - ◊ **diff:** Dificultad de la ruta valorada en 3 posibilidades -> Fácil, Normal o Difícil. Dato de tipo float.
 - ◊ **recom:** Recomendaciones a la hora de realizar el recorrido. Dato de tipo float.
 - ◊ **details:** Detalles relativos a la ruta como puede ser una especificación extra sobre las paradas que se detallan en la misma entre otra serie de características. Dato de tipo string.
 - ◊ **color:** Color con el que se identifica la ruta siguiendo el formato hexadecimal. Dato de tipo string.
 - ◊ **stops:** Array de datos correspondientes a la totalidad de paradas que posea la ruta en cuestión.
 - ◆ **num:** Número de la parada. Dato de tipo float.
 - ◆ **name:** Nombre de la parada. Dato de tipo string.
 - ◆ **description:** Descripción relativa a la parada en cuestión en el que se explique qué se ve y donde se está, entre otras características.
 - ◆ **pos:** Posición GPS de la parada en cuestión. Dato de tipo array de floats de tamaño 2 correspondientes a latitud y longitud.
 - ◆ **details:** Detalles asociados a esa parada en concreto. En esta sección se pueden especificar información relativa a la siguiente parada a la que se irá a continuación o si a partir de esa parada ya se retorna al principio debido a la finalización de la ruta, entre otras características.
 - ◊ **points:** Conjunto de coordenadas correspondientes a la ruta del parque natural. Utilizado para remarcar un reborde que indique el recorrido de la misma. Dato de tipo array de objetos JSON conformado a su vez por un array de floats de tamaño 2 indicando latitud y longitud por cada punto perteneciente a dicho reborde.

A continuación, se muestra una figura en la que se observa esta estructura con la información relativa a la “Ruta del Vino” del parque natural.

Figura 80. NPIM-Archivo JSON (rutas)

```

routes: [
    {
        "name": "Ruta del vino",
        "description": "La ruta del vino se puso en marcha en mayo del 2010. Con esta ruta se pretende poner en valor :",
        "dist": 1.5,
        "diff": "Baja",
        "recom": "Llevar ropa, calzado cómodo y agua. No salirse de la ruta. Respetar el cultivo privado.",
        "details": "",
        "color": "#209718",
        "stops": [
            {
                "num": 0,
                "name": "Comienzo de ruta",
                "description": "Desde este punto comienza la ruta y se realiza el recorrido hacia la primera parada:",
                "pos": [38.024941, -0.659429],
                "details": ""
            },
            {
                "num": 1,
                "name": "El Altillo",
                "description": "Esta parada ofrece una vista panorámica del entorno de la Laguna de la Mata, en la que",
                "pos": [38.024015, -0.659438],
                "details": ""
            },
            {
                "num": 2,
                "name": "El cruce",
                "description": "Se encuentra situada en un cruce de caminos, entre parcelas de viñedos. Aquí encontrarás",
                "pos": [38.023288, -0.661627],
                "details": "Desde la segunda parada, seguiremos en dirección norte hacia la Laguna de la Mata."
            },
            {
                "num": 3,
                "name": "El Acequión",
                "description": "En las proximidades del canal, que comunica el mar con la Laguna de la Mata, conocido",
                "pos": [38.026204, -0.661122],
                "details": "Desde aquí, caminando hacia el Este, en dirección al pueblo de la Mata, iremos bordeando 'l"
            },
            {
                "num": 4,
                "name": "Villa laguna",
                "description": "Es un área accesible en la que dispones de sombra, mesas y bancos donde podrás reposar",
                "pos": [38.027531, -0.657571],
                "details": ""
            },
            {
                "num": 5,
                "name": "Fin de ruta",
                "description": "Por último, la quinta parada se encuentra situada entre viñedos donde se descubre la Población de la Mata",
                "pos": [38.025781, -0.656799],
                "details": ""
            }
        ],
        "points": [
            [38.024945, -0.659429],
            [38.024387, -0.659377],
            [38.024201, -0.659408],
            [38.024015, -0.659438],
            [38.023905, -0.659459],
            [38.023893, -0.659459],
            [38.023518, -0.660141],
            [38.023652, -0.660521],
            [38.023559, -0.660876],
            [38.023356, -0.661305],
            [38.023288, -0.661627],
            [38.023252, -0.661755],
            [38.024609, -0.661895],
            [38.024683, -0.662034],
            [38.024683, -0.662034],
            [38.025503, -0.661873],
            [38.025748, -0.661873],
            [38.026077, -0.661308],
            [38.026204, -0.661122],
            [38.026204, -0.661122],
            [38.026373, -0.660779],
            [38.026424, -0.660672],
            [38.026737, -0.659867],
            [38.026954, -0.659771],
            [38.027049, -0.659805],
            [38.027683, -0.658804],
            [38.027531, -0.657571],
            [38.027497, -0.657453],
            [38.027497, -0.657453],
            [38.027497, -0.657453],
            [38.027497, -0.657453],
            [38.027497, -0.657453],
            [38.027497, -0.657453],
            [38.027497, -0.657453],
            [38.027497, -0.657453],
            [38.027497, -0.657453],
            [38.027497, -0.657453],
            [38.027497, -0.657453],
            [38.027497, -0.657453],
            [38.027497, -0.657453]
        ]
    }
],

```

(Fuente: Elaboración propia)

4. Lugares de información

En esta última sección de detallará toda la información relativa a los centros de información de los que disponga el parque natural. Estos centros se utilizan para informar a los visitantes del parque de las actividades que se pueden realizar en él, las rutas que hay disponibles, lo que está o no permitido hacer, etc. A continuación, se detalla la estructura que seguirá este apartado del archivo JSON:

- **infoSites:** Array de objetos JSON que especifican toda la información relativa a los centros de información del parque natural.
 - ◊ **name:** Nombre del centro de información. Dato de tipo string.
 - ◊ **description:** Descripción general del centro de información en el que se indica que clase de información se puede encontrar en él junto con otras características como

puede ser la explicación redactado de cómo llegar a dicho centro y cómo identificarlo. Dato de tipo string.

- ◊ **details:** Detalles relativos al centro de información como pueden ser los días en los que permanecerá cerrado por festividades u otras circunstancias. Dato de tipo string.
- ◊ **staff:** Personal que se puede encontrar en el centro. Dato de tipo string.
- ◊ **time:** Horario de apertura y cierre del centro de información. Dato de tipo string.
- ◊ **phone:** Teléfono mediante el cual se puede contactar con el centro de información. Dato de tipo string.
- ◊ **email:** Dirección de correo electrónico o email con el que se puede contactar con el centro. Dato de tipo string.
- ◊ **pos:** Posición geográfica en la que se ubica el centro. Dato de tipo array de floats de tamaño 2 correspondientes a latitud y longitud.

Una vez especificada esta parte de la estructura, se muestra en la siguiente figura la información recopilada del centro de información del parque natural.

Figura 81. NPI-M-Archiivo JSON (sitios de información)

```
"infoSites": [
    {
        "name": "Centro de Información",
        "description": "Lugar en donde se puede obtener información relativa a las rutas ofertadas en el parque natural junto con sus diversas actividades",
        "details": "Días que el centro permanecerá cerrado: 1 y 6 de enero y 24, 25 y 31 de diciembre.",
        "staff": "2 Educadores medioambientales, 1 Técnico del Servicio de Gestión de Espacios Naturales Protegidos CITMA",
        "time": "De 9:30h a 14:00h (de lunes a viernes). De 9:00h a 13:00h (sábados, domingos y festivos)",
        "phone": "96 572 16 50",
        "email": "parque_lamata@gva.es",
        "pos": [38.025129, -0.658424]
    }
]
```

(Fuente: Elaboración propia)

Cabe destacar que esta es una propuesta de fichero JSON que sigue una estructura rígida. Es decir, si no se conoce información sobre algún campo en concreto, se envía una cadena vacía como respuesta en ese campo. Otra posibilidad sería la de no enviar el dato en cuestión, volviéndola con ello una estructura más dinámica, pero se ha decidido emplear esta rigidez para preservar de una manera más clara y concisa el significado de cada dato del archivo JSON creado.

7.2.2. Obtención del fichero JSON

Una vez creado el fichero que contendrá todos los datos del parque natural que mostraremos en el mapa interactivo, pasamos a detallar su obtención vía petición GET mediante nuestro user-script.

Antes de comenzar con esta explicación se indica que el archivo se publicó en GitHub en un repositorio público con tal de poder obtenerlo mediante petición GET. Se escogió esta

plataforma debido a que el borrado de sus archivos no se realiza de manera automática como puede suceder con otros servidores de almacenamiento digital online. Además, es muy utilizado por desarrolladores para almacenar sus proyectos y controlar su desarrollo, entre otros motivos.

Sin embargo, cabe destacar que empleamos esta plataforma como servidor improvisado, ya que estos datos no han sido extraídos de una base de datos ni de ningún servicio que los proporcione siguiendo el formato JSON. Por este motivo se decidió emplear jQuery para poder enviar las cabeceras correspondientes a la petición GET de manera adecuada con el fin de obtener el objeto JSON en NPIM.

Detallado esto, pasamos a explicar el cómo se obtiene en nuestro user-script.

Su obtención es simple, únicamente empleamos la función jQuery “\$.getJSON(URL, function(response){})” para poder obtener el objeto JSON asociado a la URL especificada como primer parámetro. De esta manera, conseguimos los datos del fichero JSON subido a GitHub.

A continuación se muestra la función “loadMapData(leafletMap)” la cual carga el objeto JSON y lo almacena como variable global para su posterior utilización. En los siguientes apartados se detallarán el resto de funciones a las que llama, ya que se corresponden con la creación del mapa interactivo y con la interfaz de usuario.

Figura 82. NPIM-Carga del archivo JSON

```
function loadMapData(leafletMap){
    $.getJSON("https://raw.githubusercontent.com/FranciscoJavierGF/TFG/main/NPIM_Data/NPIM_Script_Data.json").done(function(data) {
        console.log(data);
        mapData = data;
        createMapInfo(leafletMap);
        createUserInterface();
    });
}
```

(Fuente: Elaboración propia)

7.2.3. Elaboración del mapa interactivo

La elaboración del mapa interactivo se encuentra dividida en los siguientes apartados:

1. Carga de la librería Leaflet.
2. Creación del objeto “Map” utilizado en NPIM.
3. Lectura del archivo JSON.
4. Creación del “TileLayer” específico del mapa interactivo.
5. Establecimiento del área de visión del mapa.
6. Creación del área que abarca el parque natural.
7. Creación de las distintas rutas junto con sus diferentes paradas.

8. Creación de los sitios de información de la reserva natural.

A continuación, se expondrá detalladamente la elaboración de cada sección.

1. Carga de Leaflet

Con tal de elaborar el mapa interactivo se cargó la librería javascript “Leaflet” en Tampermonkey como requerimiento para el correcto funcionamiento de NPIM. En este caso fue necesario cargar además de su fichero “.js” asociado, un fichero “.css” en el que se especifica la hoja de estilos básica de los mapas de dicha librería. La carga de este archivo CSS se observa en la función denominada “loadCssRequirements()” llamada en el inicio de ejecución del script.

Para cargar este archivo, se necesitó hacer uso de la etiqueta “@grant” ubicada en la cabecera del “user-script” con el fin de otorgar permisos de lectura de archivos CSS externos. También se necesitó la etiqueta “@resource” para poder cargar recursos externos al script como pueden ser archivos CSS.

Se empleó el valor “GM_getResourceText” de la etiqueta “@grant” para poder obtener el recurso en forma de texto especificado mediante la etiqueta “@resource” dentro de la cabecera del “user-script”. Además, se usó “GM_addStyle” para añadir y activar el o los archivos CSS cargados mediante “GM_getResourceText” a los archivos CSS utilizados por el script. Este procedimiento está basado en el asunto respondido de GitHub: “*how to load css file by GM method? [58]*”.

En las siguientes figuras se muestra el valor de estas etiquetas en la cabecera de NPIM junto a la función “loadCssRequirements()” antes mencionada.

Figura 83. NPIM-Cabecera

```
// @resource    LEAFLET_LIBRARY_CSS  https://unpkg.com/leaflet@1.7.1/dist/leaflet.css
// @require     https://unpkg.com/leaflet@1.7.1/dist/leaflet.js
// @require     https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js
// @grant       GM_getResourceText
// @grant       GM_addStyle
```

(Fuente: Elaboración propia)

Figura 84. NPIM-Carga del CSS de Leaflet

```
function loadCssRequirements(){
    const LEAFLET_LIB_CSS = GM_getResourceText("LEAFLET_LIBRARY_CSS");
    GM_addStyle(LEAFLET_LIB_CSS);
}
```

(Fuente: Elaboración propia)

2. Creación del objeto “Map”

La creación del mapa interactivo precisó de la implementación de un objeto propio que almacenara todas las características utilizadas en Leaflet. De esta manera, se mantiene un registro de los datos mostrados en el mapa interactivo a tiempo real de forma que, si alguno de estos deja de mostrarse, su valor dentro de este objeto desaparece.

Este objeto Map contiene el mapa de Leaflet junto con los objetos creados por esta librería utilizados hasta el momento. En la siguiente figura se visualiza la estructura de la clase creada con este fin.

Figura 85. NPIM-Clase Map

```
class Map{
    constructor(leafletMap, tileLayer, area, routes, infoSites){
        this.leafletMap = leafletMap;
        this.tileLayer = tileLayer;
        this.area = area;
        this.routes = routes;
        this.infoSites = infoSites;
    }
};
```

(Fuente: Elaboración propia)

Cada una de las variables de instancia de esta clase se corresponde con un objeto de Leaflet que almacena y muestra los datos del mapa interactivo en tiempo de ejecución. El valor de estos objetos se determinará conforme se realice el procesamiento del archivo JSON con los datos del mapa interactivo.

En primera instancia, creamos una variable global llamada “map” que se corresponderá con un objeto de esta clase. A continuación, creamos el mapa de Leaflet y lo almacenamos dentro de esta variable para así poder realizar cambios sobre él en el futuro. La creación de nuestro objeto Map propio junto con la del mapa de Leaflet se puede observar en la función “createMap()” la cual llama internamente a la función “createMapContainer()” que crea el

contenedor en donde se colocará el mapa en el sitio web para su visualización. Dicho contenedor se corresponderá con un elemento HTML “<div></div>” con un estilo CSS en línea que permita la correcta visualización del mapa interactivo.

En la siguiente figura se observa la implementación de dichas funciones.

Figura 86. NPIM-Creación del mapa de Leaflet junto con el objeto Map

```
function createMapContainer(){
    var mapLoc = document.querySelectorAll("body .nav-menu-style-images")[0];
    var mapContainer = document.createElement("div");
    mapContainer.id = "map";
    mapContainer.style = "width: 60em; height: 65em";
    mapLoc.appendChild(mapContainer);
}

function createMap(){
    createMapContainer();
    var leafletMap = L.map('map');
    var tileLayerMap = L.tileLayer('https://s.tile.openstreetmap.org/{z}/{x}/{y}.png', {
        attribution: '&copy; <a href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'
    }).addTo(leafletMap);
    map = new Map(leafletMap);
    map.tileLayer = tileLayerMap;
}
```

(Fuente: Elaboración propia)

3. Lectura del archivo JSON

Una vez creado el mapa de Leaflet pasamos a llenar los datos que se mostrarán en este. Para ello, leemos el archivo JSON tal y como se especifica en el punto denominado *Obtención del fichero JSON* y almacenamos sus datos en una variable global denominada “mapData” para su posterior utilización. La función empleada para ello es llamada desde la función “initScript()” que remarca el inicio de ejecución de NPIM y se le pasa como parámetro, tal y como se ve en la *Figura 82*, el mapa de Leaflet con el fin de procesar los datos mediante esta librería.

Además, se observa la llamada a la función “createMapInfo(leafletMap)”, función con la que crearemos cada objeto de Leaflet que se mostrará en nuestro mapa interactivo. Mediante esta última función realizaremos las llamadas al resto de funciones necesarias para la creación de cada dato a mostrar en el mapa. En la siguiente figura se muestra esta misma función.

Figura 87. NPIM-Creación de la información a mostrar en el mapa interactivo

```
function createMapInfo(leafletMap){
    setTileLayer(leafletMap, mapData.license);
    setMapView(leafletMap, mapData.posGPS, LEAFLET_MAP_DEFAULT_ZOOM);
    createNatParkArea(leafletMap, mapData.natPark, mapData.mun, mapData.area);
    createRoutes(leafletMap, mapData.routes);
    createInfoSites(leafletMap, mapData.infoSites);
    console.log(map);
}
```

(Fuente: Elaboración propia)

4. Creación del “Tilelayer”

El tilelayer es una característica de los mapas de Leaflet que permite remarcar características como la licencia del mapa junto con la de los datos mostrados en él. En el momento de la creación del mapa, se estableció como tilelayer el del copyright de Leaflet tal y como puede observarse en la *Figura 86*. Sin embargo, falta remarcar la licencia de los datos mostrados en él por lo que, para ello, se empleó la etiqueta “license” del archivo JSON leído utilizando para ello la variable global mapData.

Se realiza una sustitución sobre el tilelayer del mapa por la de uno nuevo que contenga los datos de copyright de Leaflet junto con los del atributo license de la variable mapData. Esta sustitución se observa en la función denominada “setTileLayer(leafletMap, license)” llamada desde la función “createMAPInfo(leafletMap)”. Cabe destacar que se ha tenido en cuenta la no existencia de información con respecto al atributo license del archivo JSON, de manera que, si no hay datos sobre la licencia de los datos, únicamente se muestra el copyright de dicha librería.

Figura 88. NPIM-Modificación del TileLayer

```
function setTileLayer(leafletMap, license){
    var newTileLayer = map.tileLayer.options.attribution;
    if(license != null && license != undefined && license != ""){
        newTileLayer = newTileLayer + ", " + license;
    }
    map.tileLayer = L.tileLayer(map.tileLayer._url, {attribution: newTileLayer}).addTo(leafletMap);
}
```

(Fuente: Elaboración propia)



(Fuente: Elaboración propia)

5. Establecimiento del área de visión del mapa

Para visualizar el parque natural de Lagunas de la Mata y Torrevieja utilizando Leaflet es necesario especificarle, mediante coordenadas de latitud y longitud, la posición GPS desde la que se quiere observar el mapa. Además, se le debe de indicar también el zoom.

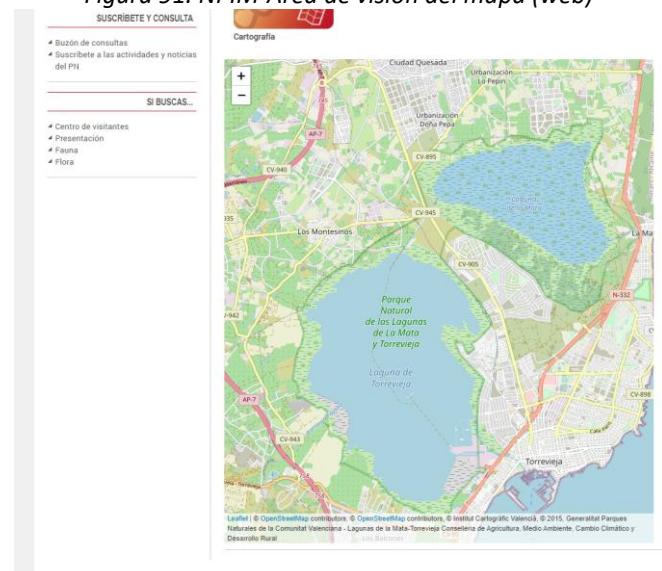
Con tal de establecer este punto inicial de visualización se empleó el atributo “posGPS” de la variable global `mapData` junto con una constante denominada “LEAFLET_MAP_DEFAULT_ZOOM”, cuyo valor es 13. Con este propósito se creó la función llamada “`setMapView(leafletMap, view, zoom)`” gracias a la cual podremos visualizar el parque natural.

Figura 90. NPIM-Establecimiento del área de visión del mapa

```
function setMapView(leafletMap, view, zoom){
    leafletMap.setView(view, zoom);
}
```

(Fuente: Elaboración propia)

Figura 91. NPIM-Área de visión del mapa (web)



(Fuente: Elaboración propia)

6. Creación del área que abarca el parque natural

Una vez tenemos el punto de visión colocado sobre el parque natural pasamos a delimitar el área que este abarca. Para ello, empleamos la información ubicada en el atributo “area” de la variable global mapData. Utilizando el subatributo “edge” realizaremos un reborde que delimita la extensión del parque natural empleando el objeto “poliline” de Leaflet.

Este objeto permite crear una línea poligonal especificando las coordenadas que la conforman. Al ser una línea, si se desea que esta sea cerrada debe especificarse como último punto la primera coordenada con la que comienza. Este último aspecto ha sido contemplado en la muestra de datos dentro del archivo JSON por lo que el valor de edge ya tiene en cuenta esta situación.

La fabricación de este objeto es simple, únicamente hay que crearlo mediante la función “L.poliline(points)” y añadirlo al mapa empleando la instrucción “addTo(map)”. Una vez creado, pasamos a detallar el estilo que posee este reborde y la información que especifica cuando el usuario interactúa con él.

Primero se comenta la información que muestra este reborde cuando se interactúa con él. Esta información aparece cuando se produce el evento “onclick()” y muestra un pop-up de Leaflet con la información relativa al valor del área del parque natural en “ha” o hectáreas junto con el nombre del parque natural y el municipio al que pertenece. Este pop-up posee un ancho máximo definido mediante una constante denominada “MAX_POPUP_WIDTH_LEAFLET” y que tiene por valor un ancho de 300. En la siguiente figura se observa la función empleada para crear este mensaje modal.

Figura 92. NPIM-Mensaje modal Leaflet (área)

```
function createAreaPopUp(natPark, mun, areaValue){  
    var popUp = document.createElement("div");  
    var title = "<h2>" + natPark + "</h2>";  
    var munic = "<br><strong>Municipio: </strong>" + mun + "<br><br>";  
    var area = "<strong>Área: </strong>" + areaValue + " ha";  
    popUp.innerHTML = title + munic + area;  
    return popUp;  
}
```

(Fuente: Elaboración propia)

Con tal de conseguir un efecto de interacción que incite al usuario a que, si pulsa sobre el reborde, recibirá más información, se implementó el evento CSS “hover”. Este evento consistirá en que, si el ratón se encuentra sobre el reborde, su ancho cambiará haciéndose más grueso y volverá a su grosor original cuando deje de estar sobre él. Su implementación puede observarse en la función llamada “setBorderHover(poliline)” mostrada en la *Figura 93*.

Una vez creadas todas las características que poseerá el área del parque natural, almacenamos el objeto poliline de Leaflet con todas estas especificaciones en nuestra variable global map, concretamente, en su atributo area. Esto se puede observar en la función denominada “createNatParkArea(leafletMap, natPark, mun, area)” indicada en la *Figura 94*. Cabe destacar que los parámetros que recibe esta función se corresponden con el mapa de Leaflet y con la información que necesita tanto para generar el pop-up como para crear el reborde del área.

Figura 93. NPIM-Creación del evento hover para las líneas del mapa interactivo

```
function setBorderHover(polyline){  
    polyline.on('mouseover', function(e){  
        var layer = e.target;  
        layer.setStyle({  
            weight: 6  
        });  
    });  
    polyline.on('mouseout', function(e){  
        var layer = e.target;  
        layer.setStyle({  
            weight: 3  
        });  
    });  
}
```

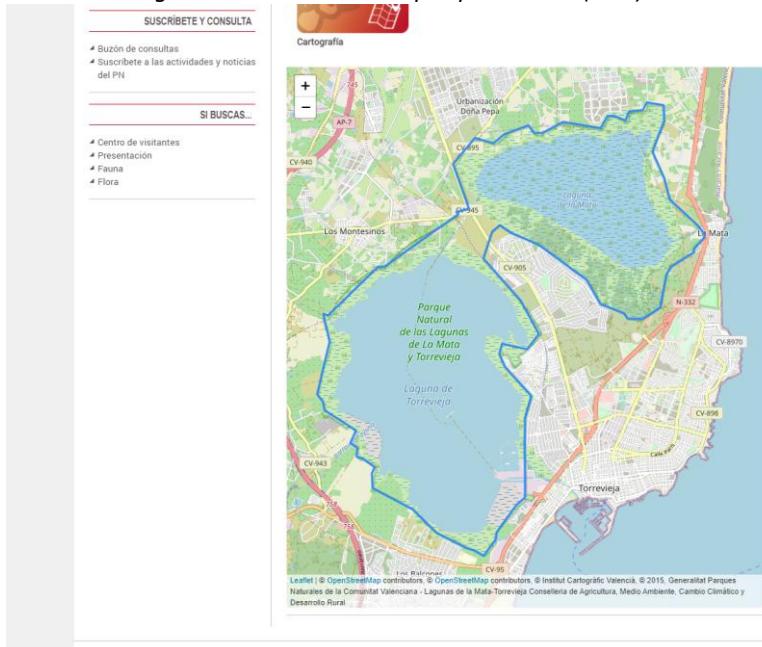
(Fuente: Elaboración propia)

Figura 94. NPIM-Creación del reborde que representa el área del parque natural

```
function createNatParkArea(leafletMap, natPark, mun, area){  
    var polyline = L.polyline(area.edge).addTo(leafletMap);  
    polyline.bindPopup(createAreaPopUp(natPark, mun, area.value), {maxWidth: MAX_POPUP_WIDTH_LEAFLET});  
    setBorderHover(polyline);  
    map.area = polyline;  
}
```

(Fuente: Elaboración propia)

Figura 95. NPIM-Área del parque natural (web)



(Fuente: Elaboración propia)

7. Creación de las rutas del parque natural

Una vez tenemos remarcada el área correspondiente al parque natural pasamos a especificar cada una de las rutas que se ofertan en el mismo. Esta información la obtenemos a partir del atributo “routes” de la variable global `mapData`. La explicación referente a la creación de las rutas se dividirá según los siguientes apartados:

1. Creación de la clase “Route”.
2. Creación del recorrido de la ruta.
3. Creación de las paradas.

A continuación, se detallará cada punto de la lista anterior.

1. Creación de la clase “Route”

Con el fin de poder almacenar todos los datos de las rutas incluyendo paradas, entre otras características, se creó una clase denominada “Route” que contuviera todos los datos necesarios para poder tener este registro en nuestra variable global `map`.

Esta clase tendrá como atributos:

- Un identificador de tipo string para la ruta conformado por la cadena “route” + la posición que ocupe dentro del atributo “routes” perteneciente a la variable global mapData.
- El nombre de la ruta.
- Un objeto poliline de Leaflet con el que se remarca la línea que sigue la ruta en cuestión.
- Un array con todos los objetos de Leaflet empleados para crear cada uno de los marcadores que servirán para identificar las paradas.

En la siguiente figura se observa la estructura detallada anteriormente.

Figura 96. NPIM-Clase Route

```
class Route{
    constructor(id, name, edge, stops){
        this.id = id;
        this.name = name;
        this.edge = edge;
        this.stops = stops;
    }
};
```

(Fuente: Elaboración propia)

Gracias a esta clase, podremos generar objetos Route que posean toda la información que necesitamos. Lo que haremos para crear cada ruta del parque natural simplemente será obtener un objeto Route asociado a cada ruta. Una vez conseguidos estos objetos, los almacenamos en un array con la finalidad de actualizar el valor del atributo routes de la variable global map.

2. Creación del recorrido de la ruta

Para crear cada ruta empleamos un objeto poliline de la misma manera que en el caso anterior con el fin de poder determinar el recorrido de la ruta. Utilizamos el atributo “points” de la ruta que se esté procesando en ese momento con tal de obtener todos los puntos necesarios para definir dicho recorrido. Una vez creado y añadido el objeto poliline al mapa de Leaflet, realizamos el evento hover empleando la función visualizada en la *Figura 93* y siguiendo el mismo esquema que el expuesto para el área del parque natural.

Es posible que se especifique un color que identifique a la ruta en cuestión según el formato propuesto para el archivo JSON. Si este color no es cadena vacía y se ha especificado, se realiza un “setStyle(cssStyle)” sobre el poliline con el fin de que el color del recorrido sea el mismo que el existente en el atributo “color” de la ruta que se está creando en ese momento.

Una vez especificado el estilo que posee el recorrido de la ruta pasamos a indicar que información muestra si el usuario realiza un evento “onclick()” sobre él. Esta información se mostrará empleando un pop-up de la misma manera que en el caso anterior. La estructura que poseerá con la información obligatoria se detalla a continuación:

1. Como título tendrá el nombre de la ruta.
2. Después aparecerá un párrafo con una descripción general de la misma.

Estos son los datos que se muestran obligatoriamente en el pop-up ya que es información que debe poseer la ruta. A partir de este punto se detalla, en orden, toda aquella información que aparece si se poseen datos para ello:

1. Distancia del recorrido en km.
2. Dificultad de la ruta catalogada en: Fácil, Normal o Difícil.
3. Duración en minutos.
4. Detalles extras de la ruta como puede ser información referente a las paradas de la misma. Un ejemplo sería indicar que las paradas de la ruta son virtuales y, por tanto, orientativas, de manera que no se realizan paradas en el recorrido como tal.
5. Recomendaciones a la hora de realizar la ruta en cuestión.

En función de los datos que se posean sobre la ruta se generará un elemento HTML “<div></div>” con toda esta información. La generación de este pop-up se puede observar en la función denominada “createRoutePopUp(routeData)”. Las rutas se crearán empleando la función “createRoutes(leafletMap, routes)” y cada objeto Route se especificará mediante la función “createRoute(leafletMap, id, route)”. Todas estas funciones se muestran en las siguientes figuras.

Figura 97. NPIM-Mensaje modal Leaflet (recorrido de ruta)

```
function createRoutePopUp(routeData){
    var popUp = document.createElement("div");
    var title = "<h2>Ruta: " + routeData.name + "</h2>";
    var desc = "<p>" + routeData.description + "</p>";
    var dist = "", diff = "", dur = "", details = "", recom = "";
    if(routeData.dist != null && routeData.dist != undefined && routeData.dist != ""){
        dist = "<strong>Distancia:</strong><p>" + routeData.dist + " km</p>";
    }
    if(routeData.diff != null && routeData.diff != undefined && routeData.diff != ""){
        diff = "<strong>Dificultad:</strong><p>" + routeData.diff + "</p>";
    }
    if(routeData.dur != null && routeData.dur != undefined && routeData.dur != ""){
        dur = "<strong>Duración:</strong><p>" + routeData.dur + " minutos</p>";
    }
    if(routeData.details != null && routeData.details != undefined && routeData.details != ""){
        details = "<strong>Detalles:</strong><p>" + routeData.details + "</p>";
    }
    if(routeData.recom != null && routeData.recom != undefined && routeData.recom != ""){
        recom = "<strong>Recomendaciones:</strong><p>" + routeData.recom + "</p>";
    }
    popUp.innerHTML = title + desc + dist + diff + dur + details + recom;
    return popUp;
}
```

(Fuente: Elaboración propia)

Figura 98. NPIM-Creación del objeto Route

```
function createRoute(leafletMap, id, route){  
    var polyline = L.polyline(route.points).addTo(leafletMap);  
    var stopsArray = createStops(leafletMap, route.stops, route.color, route.name);  
    polyline.bindPopup(createRoutePopUp(route), {maxWidth: MAX_POPUP_WIDTH_LEAFLET});  
    setBorderHover(polyline);  
    if(route.color != null && route.color != undefined && route.color != ""){  
        polyline.setStyle({  
            color: route.color  
        });  
    }  
    return new Route(id, route.name, polyline, stopsArray);  
}
```

(Fuente: Elaboración propia)

Figura 99. NPIM-Creación de las rutas del parque natural

```
function createRoutes(leafletMap, routes){  
    var routesArray = new Array();  
    for(var i = 0; i < routes.length; i++){  
        routesArray.push(createRoute(leafletMap, "route" + i, routes[i]));  
    }  
    map.routes = routesArray;  
}
```

(Fuente: Elaboración propia)

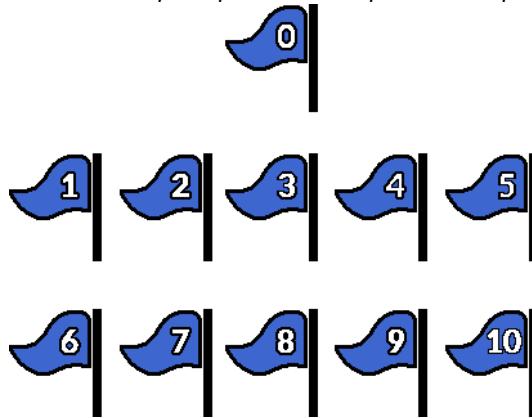
3. Creación de las paradas

Una vez analizada la creación del objeto poliline que se corresponde con el recorrido de la ruta, pasamos a detallar la elaboración de los distintos marcadores que poseen la información correspondiente a cada parada.

Las paradas se indican dentro de la ruta empleando una serie de marcadores colocados en la posición en la que deben realizarse. Se han creado una serie de marcadores con los mismos colores que los de las diferentes rutas del parque natural, con el fin de mejorar la inteligibilidad de los mismos. También se han creado con un color por defecto con el fin de considerar la posibilidad de que no se detalle un color específico para la ruta.

Estos marcadores se corresponden con una serie de banderas junto con un índice que detalla el número de la parada en cuestión. En la siguiente figura se muestra el estilo por defecto de estos iconos desarrollados para NPIM.

Figura 100. NPIM-Iconos que representan las paradas del parque natural



(Fuente: Icono de bandera [18])

Se destaca que el ícono con índice 0 se utiliza para remarcar la parada inicial de una ruta, es decir, identifica el inicio del recorrido.

Para poder emplear estos íconos en NPIM, se precisó de “imgbb” para poder subir todas las variantes del mismo a internet. Después, se realizó un filtrado por código hexadecimal de color con el fin de obtener la bandera correspondiente a la parada en cuestión con el color adecuado. Este filtrado ha sido realizado mediante una serie de constantes que almacenan en forma de array las rutas de las diferentes imágenes subidas a la plataforma junto con un “mapping” que identifica el código de color para así poder seleccionar un array u otro según corresponda. Estas constantes se visualizan en la siguiente figura.

Figura 101. NPIM-Constantes utilizadas para la creación de las paradas

```
const DEFAULT_STOP_ICONS = new Array("https://i.ibb.co/yYPGtMv/stop-Flag-0.png", "https://i.ibb.co/r4vtzS/stop-Flag-1.png", "https://i.ibb.co/kMpZtdg/stop-Flag-2.png",
    "https://i.ibb.co/pdBfZMr/stop-Flag-3.png", "https://i.ibb.co/c0BK1Bz/stop-Flag-4.png", "https://i.ibb.co/FsFn8Tz/stop-Flag-5.png",
    "https://i.ibb.co/2f4HyRw/stop-Flag-6.png", "https://i.ibb.co/YpfKsD/stop-Flag-7.png", "https://i.ibb.co/Hs9Qzrc/stop-Flag-8.png",
    "https://i.ibb.co/TpVp0/stop-Flag-9.png", "https://i.ibb.co/WnWBZTh/stop-Flag-10.png");

const GREEN_STOP_ICONS = new Array("https://i.ibb.co/n9CrDn/stop-Flag-Green-0.png", "https://i.ibb.co/StYq7/stop-Flag-Green-1.png", "https://i.ibb.co/LiyFj75/stop-Flag-Green-2.png",
    "https://i.ibb.co/k53BtkP/stop-Flag-Green-3.png", "https://i.ibb.co/z7dYRq/stop-Flag-Green-4.png", "https://i.ibb.co/pL4BR28/stop-Flag-Green-5.png",
    "https://i.ibb.co/ntSpjPf/stop-Flag-Green-6.png", "https://i.ibb.co/C7vRHc/stop-Flag-Green-7.png", "https://i.ibb.co/tCKLs2q/stop-Flag-Green-8.png",
    "https://i.ibb.co/2kytbf/stop-Flag-Green-9.png", "https://i.ibb.co/DbgvDn/stop-Flag-Green-10.png");

const YELLOW_STOP_ICONS = new Array("https://i.ibb.co/vDrrbbq/stop-Flag-Yellow-0.png", "https://i.ibb.co/fkM481/stop-Flag-Yellow-1.png", "https://i.ibb.co/hC5qZZ/stop-Flag-Yellow-2.png",
    "https://i.ibb.co/BfPQch/stop-Flag-Yellow-3.png", "https://i.ibb.co/q9cz5z1/stop-Flag-Yellow-4.png", "https://i.ibb.co/QmygBb5/stop-Flag-Yellow-5.png",
    "https://i.ibb.co/wLndkV/stop-Flag-Yellow-6.png", "https://i.ibb.co/f9gy87/stop-Flag-Yellow-7.png", "https://i.ibb.co/TkCmrXn/stop-Flag-Yellow-8.png",
    "https://i.ibb.co/mqYtJ6q/stop-Flag-Yellow-9.png", "https://i.ibb.co/gwsuDr/stop-Flag-Yellow-10.png");

const RED_STOP_ICONS = new Array("https://i.ibb.co/TbzJmp/stop-Flag-Red-0.png", "https://i.ibb.co/dmk4n1/stop-Flag-Red-1.png", "https://i.ibb.co/P64BzVP/stop-Flag-Red-2.png",
    "https://i.ibb.co/rwt5tw/stop-Flag-Red-3.png", "https://i.ibb.co/5vNFBV/stop-Flag-Red-4.png", "https://i.ibb.co/wS5fqgr/stop-Flag-Red-5.png",
    "https://i.ibb.co/ltrGln/stop-Flag-Red-6.png", "https://i.ibb.co/Jrn790w/stop-Flag-Red-7.png", "https://i.ibb.co/w88klick/stop-Flag-Red-8.png",
    "https://i.ibb.co/w6J2Ys/stop-Flag-Red-9.png", "https://i.ibb.co/XDmK1W/stop-Flag-Red-10.png");

const STOP_ICONS_COLOR = {"#2B9718": "green", "#FAE10A": "yellow", "#E90C0C": "red"};
const STOP_ICONS = {"green": GREEN_STOP_ICONS, "yellow": YELLOW_STOP_ICONS, "red": RED_STOP_ICONS, "default": DEFAULT_STOP_ICONS};
```

(Fuente: Elaboración propia)

Con el fin de obtener la imagen adecuada, se selecciona el array de íconos por defecto y después se selecciona el array de íconos adecuado atendiendo a la codificación del color de la ruta en caso de que exista. Esta selección se realiza obteniendo primero el nombre del color al que corresponde dicha codificación para así poder emplear dicho nombre con la finalidad de conseguir el array de íconos pertinente. Como los íconos se encuentran almacenados en orden

en el array, simplemente se especifica el índice de la parada que se está creando en ese momento como índice para acceder a dicho array, consiguiendo con ello el ícono pertinente.

Tras detallar la obtención del ícono que representará la parada en cuestión se detalla su carga mediante Leaflet. Estos íconos se cargarán mediante la función “L.icon” y se colocarán dentro de la especificación “L.marker” para poder crearlos como marcadores dentro del recorrido. Los marcadores se crean indicando la posición en la que se desea colocar junto con un objeto JSON que indique las características del marcador como son el ícono empleado y su etiqueta “alt” asociada.

Esta etiqueta se emplea para mejorar la accesibilidad del script y se corresponde con una cadena conformada por la palabra “Marker” + número de parada + un guión (“-”) + nombre de ruta. En la siguiente figura se muestra la creación del ícono correspondiente a las paradas del parque natural.

Figura 102. NPIM-Creación del ícono que identifica las paradas

```
function createStopIcon(index, markerIcons){  
    var stopIcon = null;  
    if(index < markerIcons.length){  
        stopIcon = L.icon({  
            iconUrl: markerIcons[index],  
            iconSize: [36, 42],  
            iconAnchor: [36, 42],  
            popupAnchor: [-3, -42]  
        });  
    }  
    return stopIcon;  
}
```

(Fuente: Elaboración propia)

Una vez creado el marcador pasamos a detallar el estilo que poseerá y que información mostrará al usuario en función de su interacción con él. El estilo estará remarcado por el evento hover al igual que en el recorrido de la ruta, la diferencia está en que en este caso, se mostrará el pop-up directamente cuando el ratón se encuentre sobre el marcador en cuestión y desaparecerá cuando ya no esté sobre el mismo. También se indica que el evento “onclick()” es eliminado para los marcadores, de manera que la única interacción con estos es mediante el evento hover.

Esta especificación puede observarse en la función “setMarkerHover(marker)” visualizada en la siguiente figura.

Figura 103. NPIM-Creación del evento hover para los marcadores de las paradas

```
function setMarkerHover(marker){  
    marker.on("mouseover", function(e){  
        this.openPopup();  
    });  
    marker.on("mouseout", function(e){  
        this.closePopup();  
    });  
    marker.off("click");  
}
```

(Fuente: Elaboración propia)

Con respecto a la información a mostrar cuando se realiza este evento hover cabe decir que, se realiza mediante un pop-up de la misma manera que en el caso del recorrido. La estructura de este mensaje modal tendrá la siguiente forma:

1. Título principal donde se remarca el nombre de la ruta a la que pertenece la parada.
2. Título secundario donde se detalla el número de la parada junto con su nombre.
3. Descripción general de la parada realizada.
4. En caso de existir, se muestran también detalles adicionales.

El único punto que es opcional en la muestra del pop-up es el último, el resto de datos se indicarán obligatoriamente para cada parada. En la siguiente figura se muestra la función empleada para generar este mensaje modal.

Figura 104. NPIM-Mensaje modal Leaflet (paradas)

```
function createStopPopUp(stopData, routeName){  
    var popUp = document.createElement("div");  
    var route = "<h2>" + routeName + "</h2>";  
    var title = "<h3>Parada: " + stopData.num + " - " + stopData.name + "</h3>";  
    var desc = "<p>" + stopData.description + "</p>";  
    var details = "";  
    if(stopData.details != null && stopData.details != undefined && stopData.details != ""){  
        details = "<strong>Detalles:</strong><p>" + stopData.details + "</p>";  
    }  
    popUp.innerHTML = route + title + desc + details;  
    return popUp;  
}
```

(Fuente: Elaboración propia)

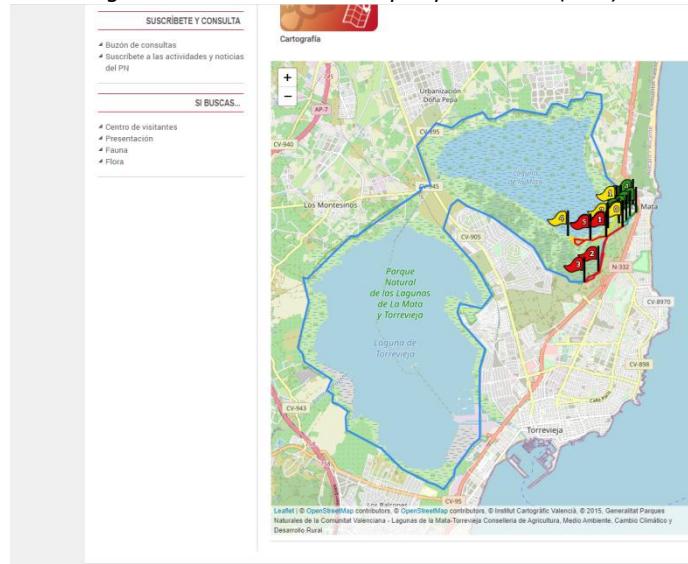
Una vez creado el objeto marker de Leaflet con todas estas características se almacena en un array con tal de devolver todas las paradas creadas de esta misma forma. El array devuelto se guardará a su vez en el atributo “stops” del objeto Route que se está creando en ese momento. En la siguiente figura se observa la función “createStops(leafletMap, stops, routeColor, routeName)” utilizada para crear todas las paradas de una ruta del parque natural.

Figura 105. NPIM-Creación de las paradas del parque natural

```
function createStops(leafletMap, stops, routeColor, routeName){  
    var stopsArray = new Array();  
    var markerIcons = STOP_ICONS["default"];  
    if(routeColor in STOP_ICONS_COLOR){  
        markerIcons = STOP_ICONS[STOP_ICONS_COLOR[routeColor]];  
    }  
    for(var i = 0; i < stops.length; i++){  
        var marker = L.marker([stops[i].pos, {icon: createStopIcon(i, markerIcons), alt: "Marker " + i + " - " + routeName}]).addTo(leafletMap);  
        marker.bindPopup(createStopPopUp(stops[i], routeName), {maxWidth: MAX_POPUP_WIDTH_LEAFLET});  
        setMarkerHover(marker);  
        stopsArray.push(marker);  
    }  
    return stopsArray;  
}
```

(Fuente: Elaboración propia)

Figura 106. NPIM-Rutas del parque natural (web)



(Fuente: Elaboración propia)

Con esto quedan creadas y almacenadas las rutas con todas las especificaciones necesarias para su futura utilización.

8. Creación de los sitios de información del parque natural

El último punto en la elaboración del mapa interactivo se corresponde con la muestra de aquellos lugares de información a los que un visitante del parque natural puede acceder. Estos datos los obtenemos a través del atributo “infoSites” de la variable global mapData.

La creación de estos lugares de información se basa en especificar un marcador que señale su ubicación en el mapa. Se creará por tanto uno por cada sitio de información existente en el archivo JSON. Estos marcadores presentarán las mismas características que los utilizados para las paradas. Esto quiere decir que, cuando se ejecute únicamente el evento “hover”, se mostrará un pop-up con la información correspondiente al centro de información.

El ícono empleado para identificar estos centros se corresponderá con el marcador típico que se puede visualizar en aplicaciones como Google Maps. Al igual que con las paradas, se ha elaborado este ícono y se ha subido a la plataforma “imgbb”. Para poder emplear dicho ícono en nuestro user-script es necesario remarcar explícitamente su URL dentro de la creación del objeto “L.icon”. También se ha especificado una sombra para este ícono con tal de otorgar una mayor sensación de volumen al marcador. Esto se detalla en la función denominada “createDefaultIcon()” mostrada en la *Figura 107*.

Figura 107. NPIM-Creación del ícono de los sitios de información

```
function createDefaultIcon(){
    var defaultIcon = L.icon({
        iconUrl: 'https://i.ibb.co/gRQzdPc/marcador-Info-Sites.png',
        shadowUrl: 'https://i.ibb.co/bWH440H/marcador-Info-Sites-Shadow.png',
        iconSize: [25, 41],
        shadowSize: [35, 41],
        iconAnchor: [12.5, 41],
        shadowAnchor: [0, 41],
        popupAnchor: [-3, -41]
    });
    return defaultIcon;
}
```

(Fuente: Elaboración propia)

Cabe destacar que el atributo “alt” también se ha tenido en cuenta para este ícono, de manera en que su valor ahora estará conformado por una cadena creada mediante la siguiente estructura: “Marker” + número del centro de información actual + “- InfoSites”. En la siguiente figura se observa el ícono creado para los centros de información antes mencionado junto con su sombra.

Figura 108. NPIM-Ícono de marcador para los centros de información



(Fuente: Ícono de marcador [19])

Con respecto al pop-up mostrado como información al usuario, cabe decir que posee la siguiente estructura según los datos especificados obligatoriamente:

1. Un título con el nombre del sitio de información.
2. Una breve descripción de dicho centro indicando características básicas como puede ser el cómo llegar a él.

Esta es la información mínima que se muestra de un centro de información. A continuación, se especifica en orden, todos aquellos campos que se pueden añadir en caso de existir en el archivo JSON:

1. Un párrafo con el personal que se puede encontrar trabajando en ese centro de información.
2. El teléfono mediante el cual se puede contactar con el centro.
3. El email del centro de información.
4. El horario de apertura y cierre de dicho centro.
5. Detalles correspondientes a días en los que el centro puede permanecer cerrado, entre otros.

Con esto queda detallada la información que podremos visualizar referente a los centros de información del parque natural. En la siguiente figura se observa la función utilizada para la creación de este mensaje modal.

Figura 109. NPIM-Mensaje modal Leaflet (sitios de información)

```
function createInfoSitePopUp(infoSite){
    var popUp = document.createElement("div");
    var title = "<h2>" + infoSite.name + "</h2>";
    var desc = "<p>" + infoSite.description + "<p>";
    var staff = "", phone = "", email = "", time = "", details = "";
    if(infoSite.staff != null && infoSite.staff != undefined && infoSite.staff != ""){
        staff = "<strong>Personal:</strong><p>" + infoSite.staff + "</p>";
    }
    if(infoSite.phone != null && infoSite.phone != undefined && infoSite.phone != ""){
        phone = "<strong>Teléfono:</strong><p>" + infoSite.phone + "</p>";
    }
    if(infoSite.email != null && infoSite.email != undefined && infoSite.email != ""){
        email = "<strong>Email:</strong><p>" + infoSite.email + "</p>";
    }
    if(infoSite.time != null && infoSite.time != undefined && infoSite.time != ""){
        time = "<strong>Horario:</strong><p>" + infoSite.time + "</p>";
    }
    if(infoSite.details != null && infoSite.details != undefined && infoSite.details != ""){
        details = "<strong>Detalles:</strong><p>" + infoSite.details + "</p>";
    }
    popUp.innerHTML = title + desc + staff + phone + email + time + details;
    return popUp;
}
```

(Fuente: Elaboración propia)

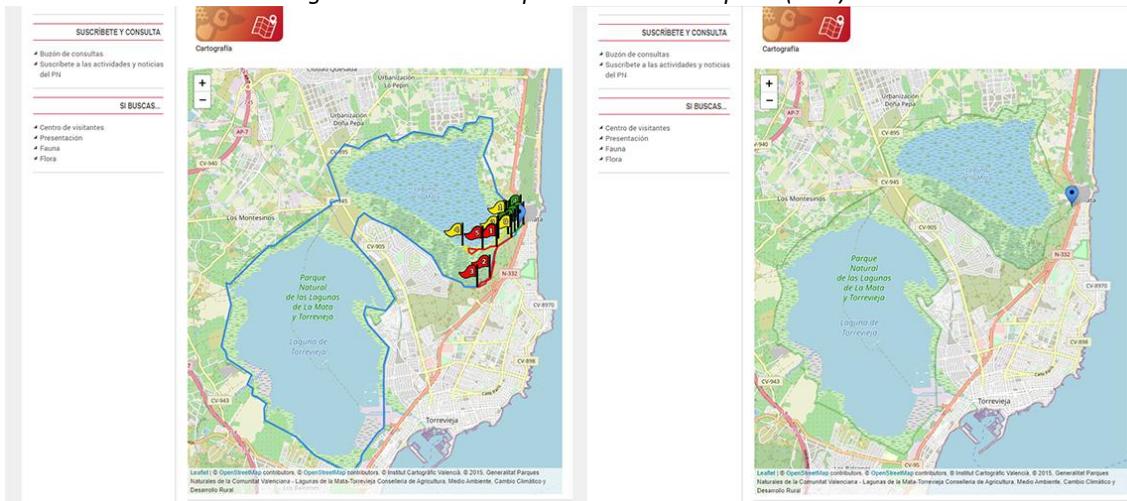
La función empleada para generar cada marcador correspondiente a los centros de información se denomina “createInfoSites(leafletMap, infoSites)” y se puede visualizar en la *Figura 110*.

Figura 110. NPIM-Creación de los sitios de información

```
function createInfoSites(leafletMap, infoSites){
    var infoSitesArray = new Array();
    for(var i = 0; i < infoSites.length; i++){
        var infoMarker = L.marker(infoSites[i].pos, {icon: createDefaultIcon(), alt: "Marker " + i + " - InfoSites"}).addTo(leafletMap);
        infoMarker.bindPopup(createInfoSitePopUp(infoSites[i]), {maxWidth: MAX_POPUP_WIDTH_LEAFLET});
        setMarkerHover(infoMarker);
        infoSitesArray.push(infoMarker);
    }
    map.infoSites = infoSitesArray;
}
```

(Fuente: Elaboración propia)

Figura 111. NPIM-Mapa interactivo completo (web)



(Fuente: Elaboración propia)

Y con esto queda detallada la creación del mapa interactivo. En el siguiente punto se especificará la interfaz de usuario generada para NPIM.

7.2.4. Creación de la UI

La interfaz de usuario estará conformada por una serie de “checkbox” que remarcarán los datos a mostrar en el mapa interactivo. Estos checkbox se colocarán en un “`<form></form>`” de HTML justo al lado derecho del mapa de Leaflet para facilitar su uso y visibilidad.

La creación de la interfaz de usuario se compone de los siguientes pasos:

1. Creación del espacio reservado para la interfaz de usuario.
2. Creación de los checkbox genéricos junto con su label asociado.
3. Creación del botón correspondiente al área del parque natural.
4. Creación de los botones de las diferentes rutas.
5. Creación del botón que representa los sitios de información.
6. Generación de eventos correspondientes a la visibilidad de los elementos en el mapa interactivo.

A continuación, se detallará cada punto indicado en los pasos anteriores.

1. Creación del espacio reservado para la interfaz de usuario

Para crear la UI es necesario reservar un espacio en el que se puedan colocar los diferentes botones de interacción con el mapa. Este espacio estará conformado por un elemento HTML “`<div></div>`” que contendrá el mapa de Leaflet junto con el formulario propio de la interfaz

de usuario. Se ha empleado esta estructura con el fin de colocar la UI a la derecha del propio mapa para así aumentar su usabilidad tal y como se ha comentado anteriormente.

El formulario poseerá de título el texto “NPIM Interface” y estará conformado por colores de fondo verdes debido a la relación directa de este color con la naturaleza y, por ende, con los parques naturales. En las siguientes figuras se muestra la función empleada para la creación del formulario junto con las utilizadas con el fin de otorgarle un estilo agradable al usuario.

Figura 112. NPIM-Creación del formulario que contiene la interfaz de usuario

```
function createUIButtonContainer(){
    var mapDom = document.getElementById("map");
    var mapParent = mapDom.parentNode;
    var uiFormTitle = document.createElement("h2");
    uiFormTitle.innerHTML = "NPIM Interface";
    setUIFormTitleStyle(uiFormTitle);
    var scriptContainer = document.createElement("div");
    setScriptContainerStyle(scriptContainer);
    var uiButtonContainer = document.createElement("form");
    uiButtonContainer.id = "uiCont";
    setUIButtonContainerStyle(uiButtonContainer);
    uiButtonContainer.appendChild(uiFormTitle);
    scriptContainer.appendChild(mapDom);
    scriptContainer.appendChild(uiButtonContainer);
    mapParent.appendChild(scriptContainer);
}
```

(Fuente: Elaboración propia)

Figura 113. NPIM-Creación del estilo que posee el formulario de la interfaz de usuario

```
function setUIButtonContainerStyle(uiButtonContainer){
    uiButtonContainer.style.display = "table";
    uiButtonContainer.style.backgroundColor = "#D7FADD";
    uiButtonContainer.style.height = "100%";
    uiButtonContainer.style.border = "0.3em solid #72BD22";
    uiButtonContainer.style.borderCollapse = "collapse";
    uiButtonContainer.style.marginLeft = "3%";
}

function setUIFormTitleStyle(uiFormTitle){
    uiFormTitle.style.color = "#000000";
    uiFormTitle.style.backgroundColor = "#7AD780";
    uiFormTitle.style.fontSize = "1.3em";
    uiFormTitle.style.textAlign = "center";
    uiFormTitle.style.margin = "0px";
    uiFormTitle.style.marginBottom = "1em";
    uiFormTitle.style.paddingTop = "0.5em";
    uiFormTitle.style.paddingBottom = "0.5em";
    uiFormTitle.style.borderBottom = "0.3em solid #72BD22";
}

function setScriptContainerStyle(scriptContainer){
    scriptContainer.style.display = "flex";
    scriptContainer.style.justifyContent = "space-between";
}
```

(Fuente: Elaboración propia)

2. Creación de los checkbox genéricos junto con su label asociado

Con el objetivo de crear una estructura de botones fácilmente usable por el usuario, se realizó un modelo de checkbox genérico que fuera reutilizable para todas las interacciones posibles del mapa interactivo. Este modelo está conformado por un elemento HTML “

</p>” que engloba a una etiqueta “<label></label>” junto con el propio checkbox. Gracias a esta

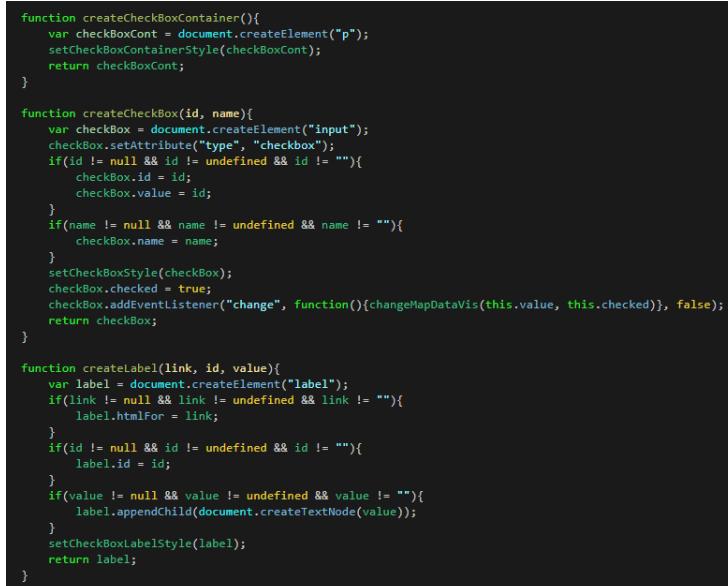
estructura se pueden incluir de manera lógica, ordenada y sencilla tantos botones de interacción como sean necesarios en nuestro user-script.

Estos checkbox presentan un estilo definido compuesto por la realización del evento “hover”. Cuando el ratón se encuentre encima del checkbox o de su label asociada se subrayará el botón de interacción y el estilo de letra pasará a ser “negrita” con la finalidad de recalcar el botón sobre el que se está actualmente. En el momento en el que el ratón deje de estar sobre dicho botón, el estilo CSS volverá a su estado anterior.

También se indica que se ha realizado el evento “onchange()” para los botones de interacción de manera que, en el momento en el que los checkbox cambien su estado a activado o desactivado, se llame a una función que muestre o elimine contenido del mapa interactivo. Esta función se denomina “changeMapDataVis(value, isActive)” y será detallada en los siguientes apartados.

Cabe destacar que estos checkbox se encontrarán activos por defecto ya que originalmente se mostrará toda la información disponible en el mapa interactivo. En las siguientes figuras se muestran las funciones empleadas para generar estos botones de interacción junto con las utilizadas con el fin de otorgarle un estilo CSS adecuado según corresponda.

Figura 114. NPIM-Creación de los botones de interacción de la interfaz de usuario



```
function createCheckBoxContainer(){
    var checkBoxCont = document.createElement("div");
    setCheckBoxContainerStyle(checkBoxCont);
    return checkBoxCont;
}

function createCheckBox(id, name){
    var checkBox = document.createElement("input");
    checkBox.setAttribute("type", "checkbox");
    if(id != null && id != undefined && id != ""){
        checkBox.id = id;
        checkBox.value = id;
    }
    if(name != null && name != undefined && name != ""){
        checkBox.name = name;
    }
    setCheckBoxStyle(checkBox);
    checkBox.checked = true;
    checkBox.addEventListener("change", function(){changeMapDataVis(this.value, this.checked)}, false);
    return checkBox;
}

function createLabel(link, id, value){
    var label = document.createElement("label");
    if(link != null && link != undefined && link != ""){
        label.htmlFor = link;
    }
    if(id != null && id != undefined && id != ""){
        label.id = id;
    }
    if(value != null && value != undefined && value != ""){
        label.appendChild(document.createTextNode(value));
    }
    setCheckBoxLabelStyle(label);
    return label;
}
```

(Fuente: Elaboración propia)

Figura 115. NPIM-Creación del estilo CSS de los botones de interacción

```
function setCheckBoxContainerStyle(checkBoxCont){
    checkBoxCont.style.display = "flex";
    checkBoxCont.style.margin = "0.5em";
    checkBoxCont.style.color = "#000000";
}

function setCheckBoxStyle(checkBox){
    checkBox.style.marginRight = "0.5em";
    checkBox.onmouseover = function(e){
        var label = this.nextSibling;
        label.style.textDecoration = "underline";
        label.style.fontWeight = "bold";
    };
    checkBox.onmouseout = function(e){
        var label = this.nextSibling;
        label.style.textDecoration = "none";
        label.style.fontWeight = "normal";
    }
}

function setCheckBoxLabelStyle(checkBoxLabel){
    checkBoxLabel.onmouseover = function(e){
        this.style.textDecoration = "underline";
        this.style.fontWeight = "bold";
    };
    checkBoxLabel.onmouseout = function(e){
        this.style.textDecoration = "none";
        this.style.fontWeight = "normal";
    }
}
```

(Fuente: Elaboración propia)

3. Creación del botón correspondiente al área del parque natural

Este botón representará la visibilidad del reborde correspondiente al área del parque natural. La forma de crearlo es sencilla, simplemente se utilizan las funciones de creación de los checkbox genéricos pasándoles los parámetros adecuados. En este caso, se le especificará como valor del label la cadena “Área” + el nombre del parque natural. Dicho nombre se extraerá del atributo “natPark” de la variable global mapData. Una vez creado el botón en cuestión, se introduce dentro del formulario correspondiente a la UI.

En la siguiente figura se observa la función empleada para crear este botón de interacción.

Figura 116. NPIM-Botón de interacción (área)

```
function createAreaButton(natPark){
    var uiCont = document.getElementById("uiCont");
    var areaCont = createCheckBoxContainer();
    var areaCheck = createCheckBox("area", "natParkArea");
    var areaLabel = createLabel("area", "arealabel", "Área de " + natPark);
    areaCont.appendChild(areaCheck);
    areaCont.appendChild(areaLabel);
    uiCont.appendChild(areaCont);
}
```

(Fuente: Elaboración propia)

4. Creación de los botones de las diferentes rutas

Estos botones representarán las diferentes rutas ofertadas por el parque natural. Debido a esto, habrá un botón por cada ruta existente en el archivo JSON. Se ha empleado esta

mecánica con el fin de poder ocultar rutas del parque natural en concreto para así otorgarle al usuario un mayor control de la información que desea visualizar.

La creación de estos botones sigue la misma mecánica que en el caso anterior solo que ahora, el nombre del botón se corresponderá con el nombre de la ruta de la que se estén obteniendo datos en ese momento. El nombre de la ruta se extrae del subatributo “name” perteneciente al atributo routes de la variable global mapData.

La ID de este checkbox presentará el mismo formato que el que poseen los objetos Route creados originalmente en el mapa interactivo. Dicho formato ha sido expuesto anteriormente en la definición de la clase *Creación de la clase “Route”*. De esta manera, la ID presentará un índice diferente por cada ruta detallada en el archivo JSON. Mediante este mecanismo, se van creando los diferentes botones para las rutas mientras se añaden a la UI.

En la siguiente figura se muestra la función empleada para la creación de estos botones.

Figura 117. NPIM-Botones de interacción (rutas)

```
function createRoutesButtons(routes){  
    var uiCont = document.getElementById("uiCont");  
    for(var i = 0; i < routes.length; i++){  
        var routeCont = createCheckBoxContainer();  
        var routeCheck = createCheckBox("route" + i, routes[i].name);  
        var routeLabel = createLabel("route" + i, "route" + i + "Label", routes[i].name);  
        routeCont.appendChild(routeCheck);  
        routeCont.appendChild(routeLabel);  
        uiCont.appendChild(routeCont);  
    }  
}
```

(Fuente: Elaboración propia)

5. Creación del botón que representa los sitios de información

Este botón representa todos los centros de información existentes en el parque natural. Es decir, existe un único botón que oculta o muestra todos los lugares de información de la reserva natural. Se ha empleado esta mecánica debido a la consideración de que un usuario general no quiere visualizar centros de información en concreto, sino que desea verlos todos o ninguno en contraposición.

Al igual que en los casos anteriores, se emplea el mismo mecanismo de creación. El nombre de este botón se corresponde con la cadena “Centros de Información”. En la siguiente figura se muestra la función empleada para la generación de este botón.

Figura 118. NPIM-Botón de interacción (sitios de información)

```
function createInfoSitesButton(){
    var uiCont = document.getElementById("uiCont");
    var infoSitesCont = createCheckBoxContainer();
    var infoSitesCheck = createCheckBox("infoSites", "natParkInfoSites");
    var infoSitesLabel = createLabel("infoSites", "infoSitesLabel", "Centros de Información");
    infoSitesCont.appendChild(infoSitesCheck);
    infoSitesCont.appendChild(infoSitesLabel);
    uiCont.appendChild(infoSitesCont);
}
```

(Fuente: Elaboración propia)

7.2.5. Generación de los eventos de visualización

Una vez creada la interfaz de usuario pasamos a detallar el funcionamiento de la visibilidad de los elementos del mapa en función de los checkbox que estén activos. Tal y como se ha comentado anteriormente, se hace uso de la función “changeMapDataVis(value, isActive)” para evaluar si el contenido debe mostrarse u ocultarse según corresponda.

Esta función utiliza el valor booleano que se le especifica como segundo parámetro para determinar si el contenido detallado como primer parámetro debe desaparecer o no. Si dicho booleano presenta un valor de “true” el contenido debe mostrarse, mientras que si posee un valor de “false” está obligado a ocultarse.

Con tal de realizar estas funcionalidades se hizo uso de otras 2 funciones auxiliares denominadas “recreateForm(id)” y “destroyMapInfo(id)”. La manera de ocultar información en un mapa de Leaflet de forma que no aparezcan ni siquiera los pop-ups asociados al elemento del mapa, es destruyéndola. Es por este motivo por el que las funciones presentan la nomenclatura de “destruir” y “recrear” la información.

Estas funciones emplean la ID o identificador del checkbox que se le pasa por parámetro para determinar qué información recrear o destruir según corresponda. Dicho identificador se corresponde con el del checkbox que ha sufrido el evento “onchange()” en cuestión.

Si la información debe recrearse, simplemente se hace uso de las funciones de creación del mapa interactivo antes comentadas. En caso de querer recrear una ruta será necesario obtener el número que ocupa esta en el array de objetos JSON ubicado en la variable global mapData bajo la nomenclatura de routes. Este índice se extrae de la última posición de la ID pasada por parámetro. Gracias a este valor, es posible acceder a la ruta adecuada dentro del array para así poder enviar su información pertinente a la función “createRoute(leafletMap, id, route)”.

Si por el contrario, la información debe destruirse, se emplea la función “removeLayers(leafletObject)” propia de Leaflet para poder destruir los elementos que se encuentran en el mapa interactivo. Cabe destacar que esta destrucción afecta a nuestro objeto Map propio. De manera en que, se elimina de la variable global map toda aquella información que ya no se muestre en el mapa, ya sea sustituyendo su valor por “null” como eliminándola del array a la que pertenezca.

Para las rutas no solo hay que eliminar el recorrido de la misma, sino también sus paradas. Es por esto por lo que se elimina primero cada marcador correspondiente a las paradas de la ruta antes de eliminar el recorrido en sí. Como los pop-ups van asociados al objeto de Leaflet con el que aparecen, eliminar dicho objeto mediante “removeLayers(leafletObject)” basta para destruir también su pop-up asociado.

En las siguientes figuras se muestran las funciones empleadas para efectuar estos cambios en el mapa interactivo.

Figura 119. NPIM-Determinación de la visibilidad de los elementos del mapa

```
function changeMapDataVis(value, isActive){
    console.log(value);
    console.log(isActive);
    if(isActive){
        recreateMapInfo(value);
    }
    else{
        destroyMapInfo(value);
    }
}
```

(Fuente: Elaboración propia)

Figura 120. NPIM-Recreación de los elementos del mapa

```
function recreateMapInfo(id){
    if(id == "area"){
        createNatParkArea(map.leafletMap, mapData.natPark, mapData.mun, mapData.area);
    }
    else if(id == "infoSites"){
        createInfoSites(map.leafletMap, mapData.infoSites);
    }
    else if(id.includes("route")){
        var routeNum = id.charAt(id.length - 1);
        map.routes.push(createRoute(map.leafletMap, id, mapData.routes[routeNum]));
    }
    console.log(map);
}
```

(Fuente: Elaboración propia)

Figura 121. NPIM-Destrucción de los elementos del mapa

```
function removeRouteStops(routeStops){
    for(var i = 0; i < routeStops.length; i++){
        map.leafletMap.removeLayer(routeStops[i]);
    }
}

function removeInfoSites(){
    for(var i = 0; i < map.infoSites.length; i++){
        map.leafletMap.removeLayer(map.infoSites[i]);
    }
    map.infoSites = null;
}

function destroyMapInfo(id){
    if(id == "area"){
        map.leafletMap.removeLayer(map.area);
        map.area = null;
    }
    else if(id == "infoSites"){
        removeInfoSites();
    }
    else if(id.includes("route")){
        for(var i = 0; i < map.routes.length; i++){
            if(map.routes[i].id == id){
                removeRouteStops(map.routes[i].stops);
                map.leafletMap.removeLayer(map.routes[i].edge);
                map.routes.splice(i, 1);
                break;
            }
        }
        console.log(map);
    }
}
```

(Fuente: Elaboración propia)

Por último, cabe destacar que la llamada a todas estas funciones ha sido englobada en otra denominada “createUserInterface()”. Esta función es la última llamada desde la función “loadMapData(leafletMap)” en la que se cargan los datos del mapa interactivo a través del archivo JSON tal y como puede visualizarse en la *Figura 82*. A continuación, se muestra esta función propiamente dicha.

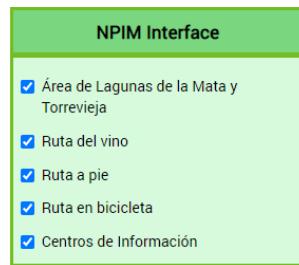
Figura 122. NPIM-Creación completa de la interfaz de usuario

```
function createUserInterface(){
    createUIButtonContainer();
    createAreaButton(mapData.natPark);
    createRoutesButtons(mapData.routes);
    createInfoSitesButton();
}
```

(Fuente: Elaboración propia)

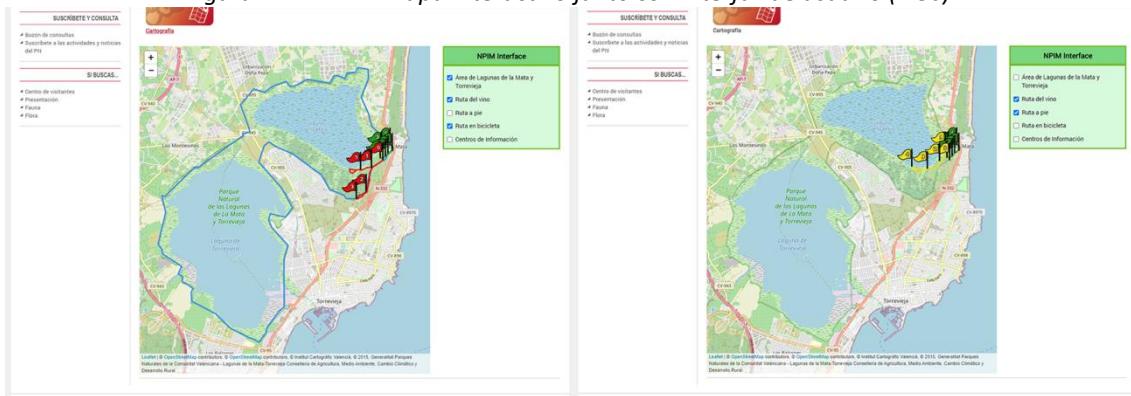
Para finalizar con la exposición de NPIM, se muestra en las siguientes figuras el aspecto que poseerá la interfaz de usuario junto con el resultado final visualizado en el sitio web.

Figura 123. NPIM-Interfaz de usuario (web)



(Fuente: Elaboración propia)

Figura 124. NPIM-Mapa interactivo junto con interfaz de usuario (web)



(Fuente: Elaboración propia)

Figura 125. NPIM-Mensaje modal (paradas-web)



(Fuente: Elaboración propia)

Y con esto queda detallada la elaboración del user-script NPIM. A continuación, se especificarán los problemas encontrados en su desarrollo.

7.2.6. Problemas encontrados

A lo largo del desarrollo de NPIM han ido apareciendo numerosas dificultades que han dificultado su elaboración. Las más destacadas son:

1. Obtención del fichero JSON procedente de GitHub.
2. Inserción correcta de iconos en el mapa de “Leaflet”.

A continuación, se detallará la totalidad de los puntos anteriores.

1. Obtención del fichero JSON procedente de GitHub

Debido a que GitHub no es una plataforma preparada para satisfacer peticiones GET de servidores externos a su propia API, obtener el fichero JSON a través de este servicio supuso realizar ciertas técnicas con tal de superar la capa de legalidad que existe tras peticiones de este tipo.

Para comenzar, fue necesario el uso de la librería jQuery para sortear la política de *CORS* [59] existente en los navegadores de internet. Cuando se realizaba la petición GET con el objeto XHR tradicional de javascript, daba como resultado un error de esta política. El error se calificaba como “Allow-Access-Control-Origin”, lo que viene a significar que no se podía recibir el objeto JSON en NPIM debido a que no se había especificado claramente cuál era el origen de la petición. Es decir, no estaba clarificado el servidor que estaba realizando esta consulta a GitHub.

Se trató de especificar dicho servidor mediante la instrucción “setRequestHeader()” del objeto XHR de numerosas maneras, pero ninguna con éxito. Al emplear la función “\$.getJSON()” propia de jQuery se pudo observar que estas cabeceras eran enviadas correctamente ya que se obtenía el archivo JSON deseado. Debido a este motivo, se decidió utilizar la librería para este propósito mientras se aprovechaban las capacidades de robustez y simplicidad propias de este framework de javascript en el desarrollo de NPIM.

Otro de los detalles a tener en cuenta es la obtención de la URL que se utilizará como fuente de la petición GET a GitHub. Para obtener este enlace, fue necesario acceder a la opción “Raw” existente en el visualizador de archivos de GitHub. Esta se aloja en la misma línea en la que se

encuentran los iconos de editar y eliminar el archivo en cuestión, identificados mediante un lápiz y una papelera. A la izquierda de estos iconos, encontramos el botón denominado “Blame”. La opción que buscamos, se encuentra justo a la izquierda de este último botón.

La opción Raw permite generar una URL con la que se puede obtener el contenido que aloja empleando tecnologías como jQuery. Es por esto por lo que, el último paso a realizar, simplemente consiste en copiar dicha URL y pegarla dentro de la definición propia de la función “\$.getJSON()” con la finalidad de obtener el archivo JSON utilizado en el user-script NPIM.

2. Inserción correcta de iconos en el mapa de “Leaflet”

La inserción de iconos en un mapa presenta algunas dificultades a tener en cuenta. Si bien con Leaflet esta inserción es relativamente sencilla, presenta algunos problemas con respecto a su colocación en el mapa. También se debe tener en cuenta la aparición de pop-ups a través del uso de estos iconos.

El principal problema viene derivado del tamaño del ícono en cuestión. Es decir, se emplean estas medidas de ancho y alto en píxeles para determinar su colocación en el mapa. Este tamaño debe de especificarse en la opción “iconSize” presente en la interfaz de creación del objeto “L.icon”. Se indica que también existe una propiedad denominada “shadowSize” que determina el tamaño de la sombra aplicada al ícono. Sin embargo, esto no es suficiente.

Con dicha opción podremos visualizar el ícono con el ancho y el alto que deseemos, pero es posible el deseo de que su colocación sea realizada desde el centro del propio ícono. Esto ocurre con el ícono clásico de ubicación empleado usualmente, el cual presenta una forma puntiaguda en la zona media del mismo, indicando que ese es exactamente el punto al que hace referencia.

Si se desea que dicha forma puntiaguda se coloque justo en la coordenada a la que corresponde el ícono, se debe de emplear la opción “iconAnchor” que determina los píxeles de ancho y alto por los que se quiere colocar el ícono. Por defecto, presentan el valor del ancho y el alto total del ícono. Para lograr colocar el ícono según esta especificación, simplemente habría que indicar en el valor correspondiente al ancho, la mitad del ancho del ícono, consiguiendo así el resultado esperado. Cabe destacar la existencia de la propiedad “shadowAnchor” que hace lo propio pero para la sombra aplicada al ícono.

Esta solución está basada en una consulta resuelta efectuada en el sitio web stackexchange.com [60].

En la *Figura 107* se puede observar la aplicación de este caso en concreto en la creación del ícono correspondiente a los sitios de información. Cabe indicar que, es posible emplear la opción “iconAnchor” para conseguir diferentes resultados según se desee.

Mediante la opción “popupAnchor” se especifica el rango de aparición del pop-up asociado al ícono. Esto permite especificar la aparición del mismo en el caso en el que, por ejemplo, el ratón se encuentre en cualquier punto sobre la imagen correspondiente al propio ícono. De esta manera, obtenemos un resultado mucho más realista y cómodo en la aparición y desaparición de estos pop-ups, ya que se mostrarán según se esté sobre la imagen y desaparecerán cuando ya no te encuentres situado sobre ella.

Se suelen emplear valores negativos en esta opción con la finalidad de otorgar un rango un poco superior al propio ícono. Al ser valores negativos, Leaflet toma toda la imagen como fuente para poder mostrar el pop-up. En NPIM, se ha empleado para el ancho el valor de “-3” con la finalidad de conseguir que el pop-up se muestre en la totalidad del ancho del ícono.

Con respecto al pop-up, se debe de especificar el rango con el que aparecerá. Si este valor no se especifica, toma por defecto el punto exacto en el que se coloca el ícono, sin ningún tipo de margen. Esto provoca resultados no deseados ya que solo se mostrará el pop-up si por ejemplo, el ratón se encontrase justo en esa posición en un evento hover. El resultado obtenido consistiría en una aparición tardía y complicada del pop-up acompañada de una desaparición rápida y sensible debido a la falta de rango de aparición del mensaje modal.

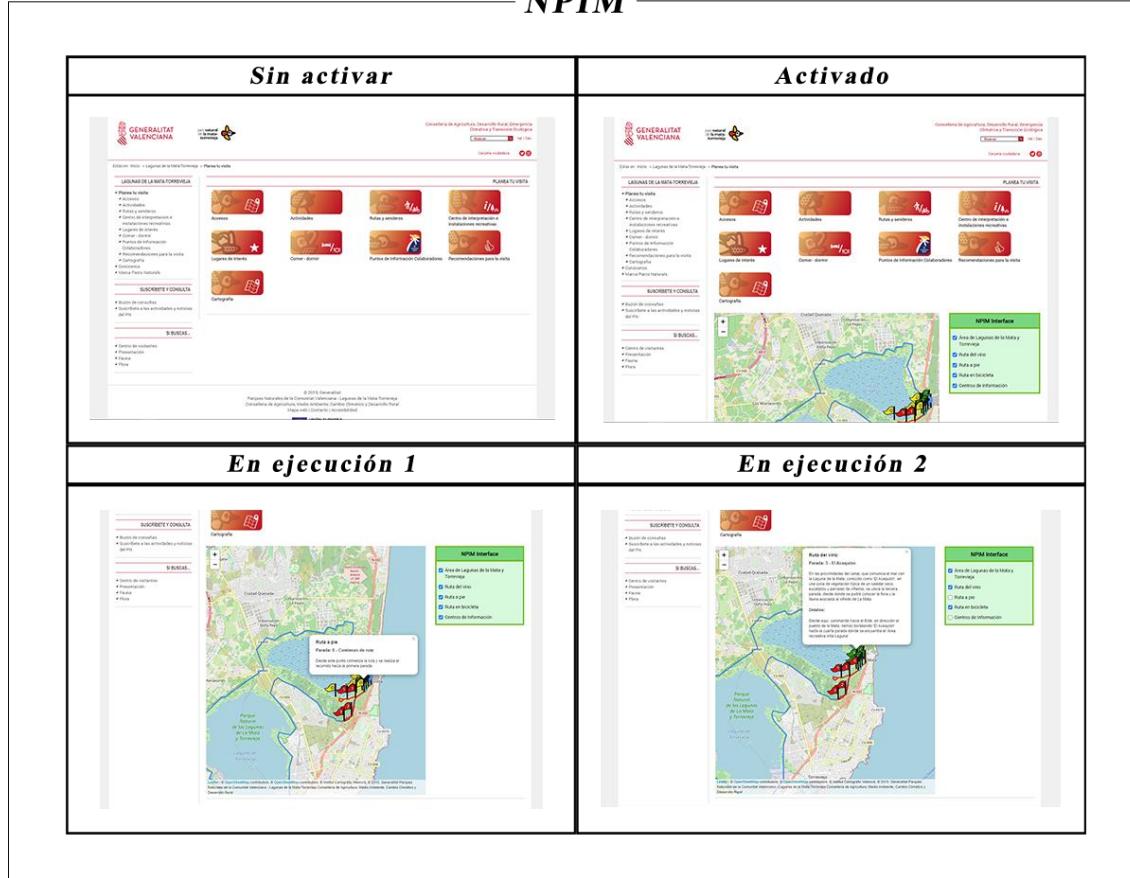
Para el alto, se ha utilizado el valor negativo de la altura original de la imagen con tal de conseguir el mismo efecto pero con respecto a la altura del ícono.

En la *Figura 102*, se muestra esta situación a la hora de crear los iconos correspondientes a las paradas de las rutas ofertadas por el parque natural. Gracias a los mecanismos proporcionados por Leaflet es posible solventar estas particularidades para así obtener un resultado con una mayor usabilidad y practicidad.

7.2.7. Visualización de cambios

Una vez expuestos todos los detalles correspondientes al desarrollo del mapa interactivo se muestra, al igual que en el script anterior, los cambios significativos de activar NPIM en el sitio web. En la siguiente figura se visualizan dichos cambios.

Figura 126. NPIM-Comparativa de activación del user-script
NPIM



(Fuente: Elaboración propia)

En este caso la diferencia radica en la aparición de un mapa dentro del apartado “Planea tu visita” en el que aparece toda la información relativa al área, las rutas, las paradas y los centros de información del parque natural. Este nuevo mapa interactivo permite la interacción con dicha información de una manera más sencilla y eficiente, facilitando al usuario la obtención de estos datos.

En el siguiente punto se detallarán las pruebas realizadas para comprobar el correcto funcionamiento de NPIM.

7.2.8. Pruebas realizadas

Con tal de comprobar el correcto funcionamiento de NPIM en una variedad de navegadores web, se realizaron diferentes pruebas relativas a su desempeño en distintas situaciones. Se contemplaron los siguientes ensayos para comprobar la correcta actuación del script:

- Muestra de todos los datos del mapa interactivo.
- Ocultación de 2 rutas del parque natural.
- Muestra del pop-up correspondiente a una ruta.
- Muestra del mensaje modal correspondiente a una parada.
- Muestra del pop-up de un centro de información.

En las siguientes figuras se observan estos ensayos para los navegadores Google Chrome, Mozilla Firefox y Opera.

Figura 127. NPIM-Pruebas en navegador Google Chrome

NPIM



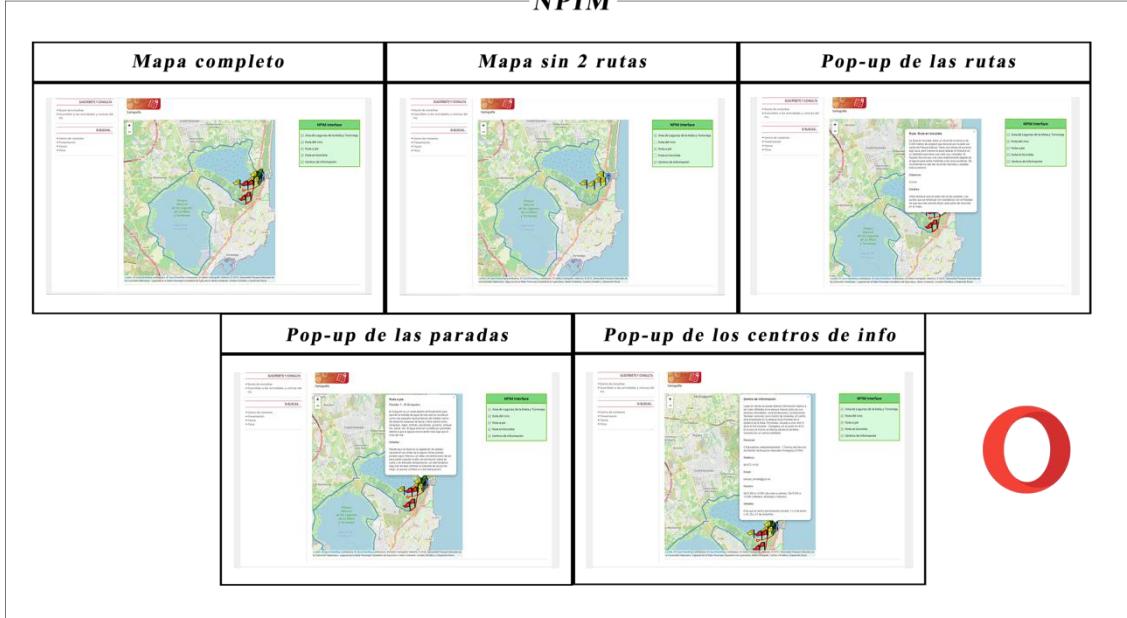
(Fuente: Logotipo Google Chrome [14])

Figura 128. NPIM-Pruebas en navegador Mozilla Firefox
NPIM



(Fuente: Logotipo de Mozilla Firefox [15])

Figura 129. NPIM-Pruebas en navegador Opera
NPIM



(Fuente: Logotipo de Opera [16])

Tal y como se observa en las figuras anteriores el resultado en los 3 navegadores es muy similar. En los 3 se ejecuta el script correctamente sin mucho problema y con el mismo aspecto visual. Solamente se diferencian un poco los checkboxes del navegador Opera, ya que especifica su estilo con otras características.

También cabe destacar que en el navegador Mozilla Firefox se produce un error en la carga del user-script NPIM. Dicho error consiste en que no carga las características del script a pesar de estar activado de forma en que no aparece ni el mapa interactivo ni la interfaz de usuario en el sitio web.

Este error puede ser producido debido a que Tampermonkey no se encuentra del todo adaptado a las características del navegador o por la utilización de librerías como jQuery enlazadas mediante esta extensión que dificultan su carga en el ciclo normal de Mozilla Firefox. En WPCS, este error no se aprecia debido a, posiblemente, la utilización de peticiones AJAX empleando el objeto XHR tradicional.

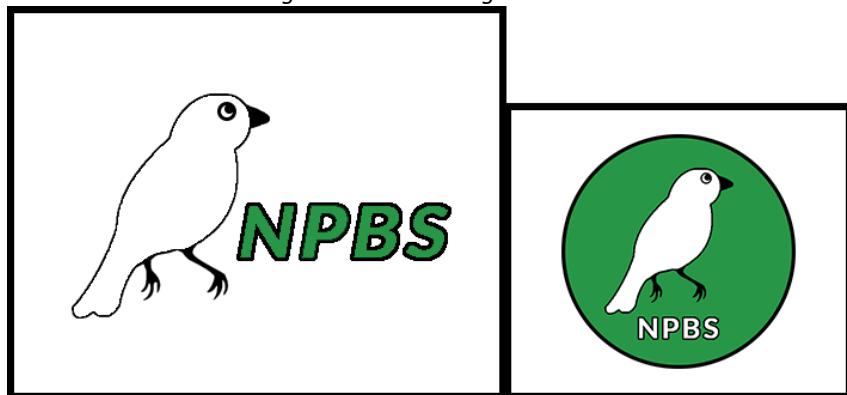
Sin embargo, se destaca que cuando el script logra cargar y ejecutarse correctamente en el navegador, no presenta ningún problema en su funcionamiento. Trabajando de la misma manera que en los otros 2 navegadores, tal y como se puede observar en las figuras anteriores.

8. Natural Park Bird Sighting (NPBS)

El último script que se ha realizado consiste en la elaboración de una tabla que mostrara, según la fecha seleccionada por el usuario, información sobre las aves avistadas en el parque natural. Además, se incluye un mapa en el que se visualiza gráficamente la posición exacta en la que se avistó cualquier ave en concreto dentro de dicho parque.

Este script utiliza el API sobre avistamiento de aves desarrollada en Estados Unidos conocida como “eBird”. Dicho API devuelve información correspondiente a las aves avistadas en regiones del mundo tales como países, estados, provincias, etc. de manera que aporta datos relativos a la posición GPS en la que se ha visualizado, el nombre común del ave, el nombre científico, la fecha en la que se ha avistado, el número de aves de la misma especie vistas en ese momento, entre otros. Su información se detallará sobre la página de index del sitio web al igual que el script WPCS.

Figura 130. NPBS-Logo e icono



(Fuente: Ave icono y logotipo [20])

8.1. Tecnologías Utilizadas

Para el desarrollo de este user-script ha sido necesaria la utilización de librerías javascript que permiten el tratamiento de mapas y mensajes modales junto con el desarrollo de tablas que siguen un diseño responsive. A continuación, se detallan estas librerías:

SweetAlert2: Detallada en la sección *Tecnologías utilizadas* del script “WPCS”.

JQuery: Explicada en el apartado *Tecnologías utilizadas* del script “NPIM”.

Leaflet: Indicada en *Tecnologías utilizadas* del script “NPIM”.

DataTables [61]: Se corresponde con un complemento o plug-in para la librería jQuery. Es una herramienta muy flexible construida según las especificaciones de dicha librería. Gracias a este plug-in, se pueden añadir las siguientes características avanzadas a cualquier tabla HTML: paginación, búsqueda avanzada y ordenación de varias columnas. Además, presenta propiedades que facilitan la utilización de este complemento según las distintas necesidades que se posean. Cabe destacar que utiliza la licencia MIT de código abierto y que empresas como Adobe, NASA, Amazon o Sony emplean este plug-in en la actualidad.

8.2. Obtención del API key de eBird

Como paso previo al desarrollo de NPBS se debe obtener una “key” por parte de eBird para poder acceder a sus datos de manera gratuita. Esta clave debe solicitarse en su página oficial (<https://ebird.org/spain/home>) y es necesario registrarse en ella para que dicha key solicitada se asocie a tu perfil de eBird. Es por eso que, el primer paso a realizar es llenar el formulario de registro de este sitio web. Para ello, simplemente hay que especificar el nombre, el o los apellidos, un nombre de usuario, una contraseña y una dirección de correo electrónico.

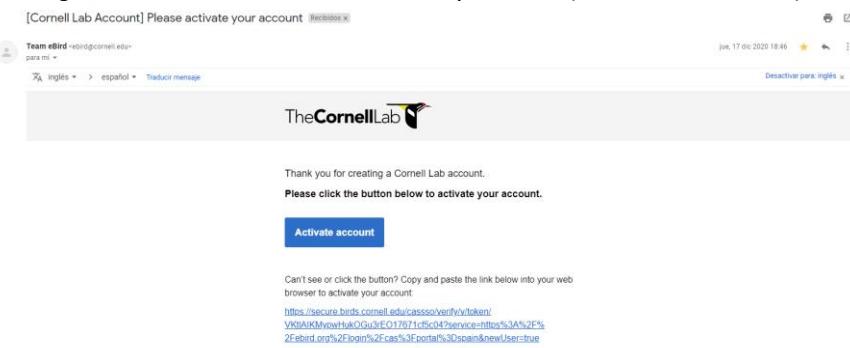
Una vez completados estos campos se enviará un correo de confirmación al correo electrónico especificado completando con ello el registro en el *laboratorio de ornitología de Cornell* [62]. Tras confirmar el email indicado se te enviará un nuevo correo electrónico en el que se ha confirmado tu registro y en el que se te da la bienvenida al sitio web. En las siguientes figuras se muestra el formulario de registro junto con el correo de confirmación y el de bienvenida enviados al email una vez realizado el registro.

Figura 131. NPBS-Obtención del API key de eBird (formulario de registro)

The screenshot shows the 'Create a Cornell Lab account' form. On the left, there's a sidebar with links to various projects: Bird Academy, Birds of the World, Celebrate Urban Birds, eBird, Great Backyard Bird Count, Macaulay Library, NestWatch, and Project FeederWatch. Below this is a note about existing accounts and a 'Sign in' link. The main form has fields for First name, Last name, Choose a username, Choose a password (with a note to enter at least 8 characters), and Email address. A large blue 'Create account' button is at the bottom. At the very bottom, there are links for Privacy Policy and Terms of Use, along with 'Already have an account?' and a 'Sign in' button.

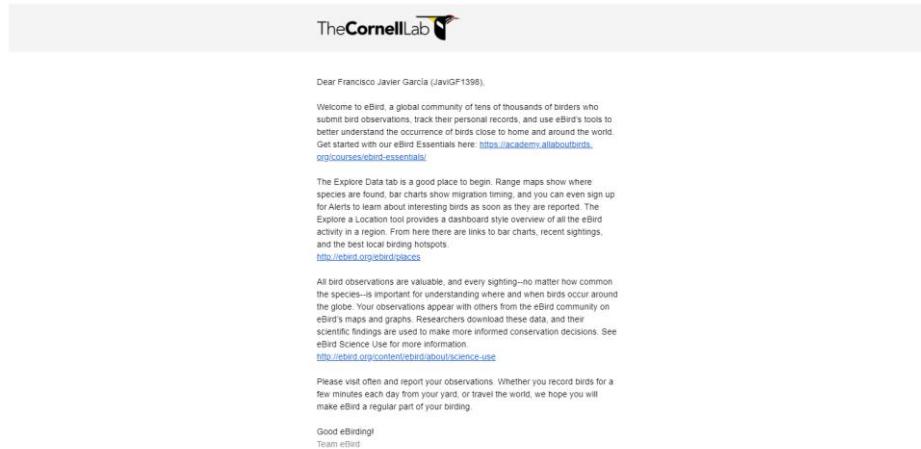
(Fuente: Elaboración propia)

Figura 132. NPBS-Obtención del API key de eBird (correo de validación)



(Fuente: Elaboración propia)

Figura 133. NPBS-Obtención del API key de eBird (correo de bienvenida)



(Fuente: Elaboración propia)

Una vez realizado el registro y con la sesión iniciada nos dirigimos a la URL (<https://ebird.org/data/download>), accesible navegando por la web desde la página principal. Se puede llegar a este apartado clicando en la opción “Solicitar datos” de la categoría “Ciencia” ubicada en el menú presente en el “footer” del sitio web. Desde esta sección, podemos solicitar la descarga de datos estáticos procedentes de la base de datos de eBird o podemos pedir la key para obtener acceso a su API.

El propósito de uso de estos datos se debe especificar en el formulario de petición tanto para el API como para los datos estáticos, de forma en que “The Cornell Lab of Ornithology” otorga el permiso de acceso y utilización de los datos de eBird empleando esta información para establecer un contrato de compromiso a modo de licencia. En caso de querer cambiar este uso, hay que ponerse obligatoriamente en contacto con eBird según su contrato legal, el cual se encuentra indicado en el apartado “Condiciones de uso” ubicado en la sección de “Ayuda”, dentro del susodicho “footer”.

Con el fin de conseguir el API key que se utilizará en NPBS, rellenamos el formulario de petición de acceso a este API. Para ello, hay que especificar la organización que lo va a utilizar (en este caso se indica como Universidad de Alicante), el título del proyecto, el tipo (elegimos académico) y el resumen en donde se aclara el propósito de uso de eBird. El nombre y el correo electrónico del usuario que realiza la petición son llenados automáticamente por el sitio web. Tras marcar la casilla de “Acepto Términos de Uso”, clicamos sobre el botón “Enviar solicitud” para así poder obtener el API key deseado.

Una vez hecho esto, se desbloquea el acceso a nuestra clave de manera en que esta se podrá visualizar automáticamente justo después de enviar la solicitud. A partir de ese momento, este

formulario ya no estará accesible para nuestro perfil debido a que ya se ha solicitado acceso al API y no es necesario volver a enviar dicha solicitud.

En caso de que se desieran descargar datos de eBird se debe llenar el mismo formulario y esperar a que llegue un correo al email especificado indicando si el acceso ha sido concedido o no. En caso de que sea concedido, se desbloquea acceso a la descarga de datos por un periodo de un año. A partir de ese momento, se pueden descargar datos de un determinado periodo de tiempo especificable de manera que, en cuanto eBird procese esta solicitud, se envía un correo electrónico con la URL de descarga de los datos solicitados. En las siguientes figuras se muestra el formulario de solicitud del API de eBird junto con el API key conseguido.

Figura 134. NPBS-Obtención del API key de eBird (formulario de solicitud)

Solicitud API Acceso

Nombre y correo electrónico: [REDACTED] | [Editar](#)

Organización: [REDACTED]

Título del Proyecto: [REDACTED]

Tipo de Proyecto: Academic/Student

Resumen: Please describe your project and how you plan to use the data

Términos de Uso: Acepto [Términos de Uso](#)

Enviar solicitud

(Fuente: Elaboración propia)

Figura 135. NPBS-API key obtenido

Request Submitted

Your eBird API Key is: dm3m200i26fm.

© Cornell Lab of Ornithology | [Contact](#) | [FAQ](#)

(Fuente: Elaboración propia)

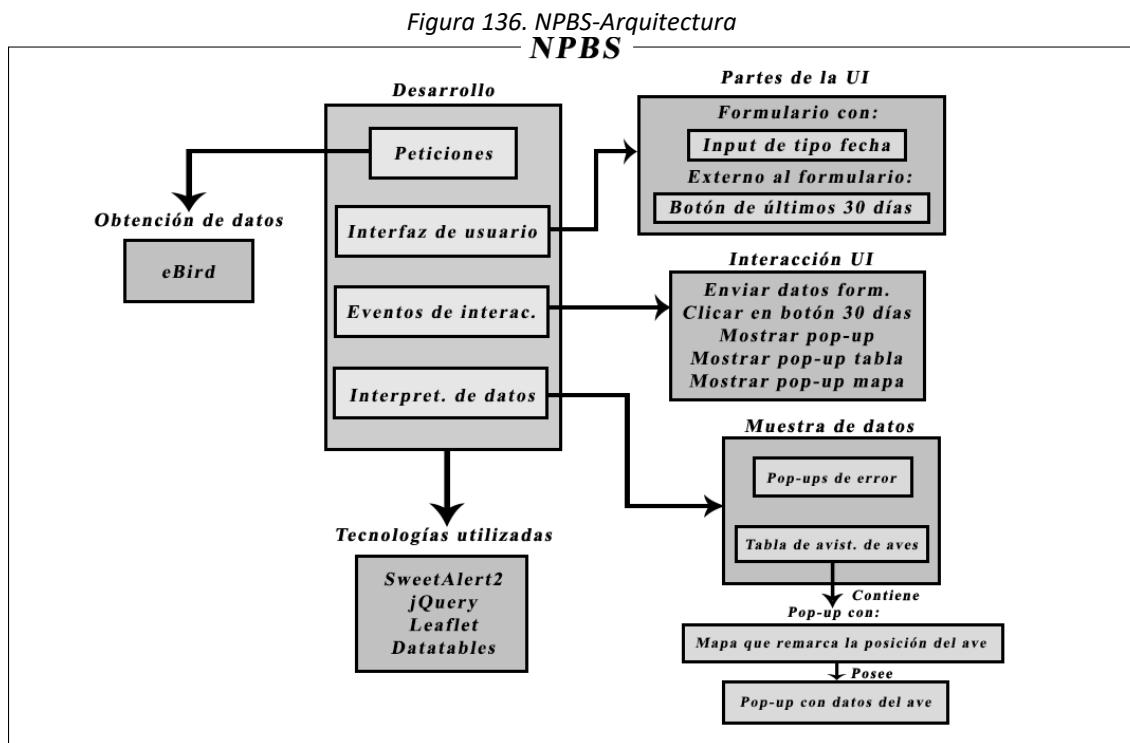
Cabe destacar que la traducción española del sitio web presenta un error a la hora de mostrar el API key obtenido. Ese error consiste en que, en dicho mensaje, se muestra que la clave expira en un día. Información errónea ya que dicha key se asocia a tu cuenta de eBird, por lo que se mantiene activa y utilizable hasta que la cuenta desaparezca.

8.3. Desarrollo

La elaboración de este script consta de los siguientes apartados:

1. Obtención del código referente a la provincia de Alicante.
2. Creación de las peticiones de avistamiento de aves al servidor de “eBird”.
3. Elaboración de la interfaz de usuario.
4. Muestra de datos.

En los siguientes puntos se detallarán, por orden, todos los apartados expuestos en la lista anterior. Además, se proporciona una figura en la que se visualiza de una manera gráfica la arquitectura que seguirá este user-script. Dicha figura se puede observar a continuación.



(Fuente: Elaboración propia)

8.3.1. Obtención del código referente a la provincia de Alicante

EBird es una aplicación internacional que funciona mediante la identificación de países, comunidades y municipios. Debido a esta característica, no se pueden extraer datos de avistamiento de aves directamente desde el parque natural sino que esta información debe obtenerse a través del municipio en el que se encuentre dicho parque.

Para ello, realizamos peticiones al servidor de eBird partiendo del código “ES” referente a España según la codificación “ISO 3166-1 alpha-2 [63]” propia del estándar “ISO 3166-1 [64]”. Este es el estándar que utiliza eBird para codificar las peticiones efectuadas referentes a los distintos países del mundo.

Empleando este código, realizamos la petición para conseguir los códigos referentes a las comunidades autónomas de España. Una vez obtenidos dichos códigos, seleccionamos el que se corresponde con la Comunidad Valenciana. Para realizar esta selección, utilizamos un filtro por nombre en el que, empleando la cadena mínima “valencia”, obtenemos dicho código empleando la menor cantidad de caracteres posible. En la siguiente figura se observan las funciones empleadas para conseguir este código.

Figura 137. NPBS-Obtención del código de la Comunidad Valenciana

```
function getComAutoCode(allComAuto){
    var comAutoCode = null;
    for(var i = 0; i < allComAuto.length; i++){
        if((allComAuto[i].name).toLowerCase().search("valencia") != -1){
            comAutoCode = allComAuto[i].code;
            break;
        }
    }
    return comAutoCode;
}

function getProvData(){
    var xhr = new XMLHttpRequest();
    xhr.withCredentials = false;

    xhr.addEventListener("readystatechange", function() {
        if(this.readyState === XMLHttpRequest.DONE) {
            if(this.status === 0 || (this.status >= 200 && this.status < 400)){
                var comAutos = JSON.parse(this.responseText);
                var comAutoCode = getComAutoCode(comAutos);
                getAllProvsOfComAuto(comAutoCode);
            } else{
                showServerErrorToUser();
            }
        }
    });
    xhr.open("GET", "https://api.ebird.org/v2/ref/region/list/subnational1/ES.json");
    xhr.setRequestHeader("X-eBirdApiToken", "dm3m200i26fm");

    xhr.send();
}
```

(Fuente: Elaboración propia)

Cabe destacar que eBird almacena los datos de codificación de regiones en archivos “.json” y, por ello, se debe concatenar esta extensión a cualquier código del que se deseen extraer sus diferentes estados, comunidades o provincias, entre otros.

Una vez obtenida la codificación empleada para la Comunidad Valenciana, realizamos otra petición para obtener las diferentes provincias que en dicha comunidad se encuentran. Al igual que en el caso anterior, usamos otro filtro por nombre para extraer el código referente a la provincia de Alicante. La cadena utilizada para este filtro es la palabra “alicante”. Dicho código, se almacenará en una variable global denominada “provData” con el fin de utilizarlo en futuras peticiones. En la siguiente figura, se observan las funciones empleadas para la extracción del código de Alicante.

Figura 138. NPBS-Obtención del código de la provincia de Alicante

```
function obtainProvData(provsOfComAuto){
    var prov = null;
    for(var i = 0; i < provsOfComAuto.length; i++){
        if((provsOfComAuto[i].name).toLowerCase() == "alicante"){
            prov = provsOfComAuto[i];
            break;
        }
    }
    return prov;
}

function getAllProvsOfComAuto(comAutoCode){
    var request = comAutoCode + ".json";
    var xhr = new XMLHttpRequest();
    xhr.withCredentials = false;

    xhr.addEventListener("readystatechange", function() {
        if(this.readyState === XMLHttpRequest.DONE) {
            if(this.status === 0 || (this.status >= 200 && this.status < 400)){
                var provs = JSON.parse(this.responseText);
                provData = obtainProvData(provs);
                console.log(provData);
            } else{
                showServerErrorToUser();
            }
        }
    });
    xhr.open("GET", "https://api.ebird.org/v2/ref/region/list/subnational2/" + request);
    xhr.setRequestHeader("X-eBirdApiToken", "dm3m200i26fm");

    xhr.send();
}
```

(Fuente: Elaboración propia)

Y con esto ya tenemos el código necesario para poder realizar las distintas peticiones de avistamiento de aves pertenecientes al parque natural.

8.3.2. Creación de las peticiones de avistamiento de aves

Las peticiones sobre avistamiento de aves realizadas al API de eBird se dividen en 2 categorías:

1. Avistamiento de aves en una fecha específica.
2. Avistamiento de aves en los últimos 30 días.

A continuación, se detallarán las funciones empleadas para los 2 puntos anteriores.

1. Avistamiento de aves en una fecha específica

Esta petición se realiza cuando un usuario indica la fecha concreta para conocer las aves que se han avistado en caso de que se haya divisado alguna. Para ello, se realiza un filtrado por fecha en el que se comprueba si la fecha seleccionada no se corresponde con una fecha futura. Dicho filtrado se realiza empleando la función “compareDatesWH(date1, date2)” perteneciente a WPCS y mostrada en la *Figura 26*.

Además, debe presentar el formato “aaaa/mm/dd” cuya obtención se puede observar en la función “getDateRequest(date)” presente en la *Figura 139*. En caso de que la fecha sea

correcta, se emplea dicha fecha junto con el código de la provincia de Alicante extraído según el punto anterior con el fin de obtener la información buscada.

Si se han obtenido resultados, se utiliza un filtrado por nombre con el fin de conseguir, únicamente, las aves que se han avistado en el parque natural. Este filtro por nombre se realiza en la función denominada “getNatParkBirdSightData(birdSightData)” que se muestra en la *Figura 140*.

Dicho filtrado consiste en emplear la cadena “pnat” para identificar los parques naturales de entre todas las localizaciones presentes en los resultados. Una vez se identifican dichos parques, se emplea la cadena mínima “la mata” para identificar el parque natural en cuestión. Las aves que se encuentren en una localización que presente estas características se almacenarán en un array con el fin de mostrar esta información al usuario.

Si hay aves avistadas en la reserva natural, se exponen en una tabla junto con sus datos identificativos, y si no hay resultados, también se le indica al usuario. Los mensajes de error que se pueden producir a la hora de realizar esta petición junto con la muestra de datos final al usuario se detallan en el punto *Muestra de datos* de la memoria. En las siguientes figuras se indican las funciones empleadas para realizar esta petición al servidor de eBird.

Figura 139. NPBS-Conversión de fecha al formato "aaaa/mm/dd"

```
function getDateRequest(date){  
    var dateSplit = date.split("-");  
    return [dateSplit[0] + "/" + dateSplit[1] + "/" + dateSplit[2]];  
}
```

(Fuente: Elaboración propia)

Figura 140. NPBS-Filtrado de resultados por nombre de localidad

```
function getNatParkBirdSightData(birdSightData){  
    var natParkBirdSight = new Array();  
    if(birdSightData != null && birdSightData != undefined && birdSightData != "" && birdSightData.length > 0){  
        for(var i = 0; i < birdSightData.length; i++){  
            if(birdSightData[i].locName != null && birdSightData[i].locName != undefined && birdSightData[i].locName != ""){  
                if((birdSightData[i].locName).toLowerCase() .search("pnat") != -1){  
                    if((birdSightData[i].locName).toLowerCase().search("la mata") != -1){  
                        natParkBirdSight.push(birdSightData[i]);  
                    }  
                }  
            }  
        }  
        console.log(natParkBirdSight);  
        return natParkBirdSight;  
    }  
}
```

(Fuente: Elaboración propia)

Figura 141. NPBS-Petición de avistamiento de aves en una fecha específica

```
function getBirdSightData(date){  
    console.log(date);  
    if(provData != null){  
        if(date != null && date != undefined && date != ""){  
            var dateObj = new Date(date);  
            var now = new Date();  
            var correctDate = putDateInCorrectFormat(date);  
            if(compareDatesWith(dateObj, now) != DATES_HIGHER){  
                var xhr = new XMLHttpRequest();  
                xhr.withCredentials = false;  
  
                xhr.addEventListener("readystatechange", function() {  
                    if(this.readyState === XMLHttpRequest.DONE) {  
                        if(this.status == 0 || (this.status >= 200 && this.status < 400)){  
                            var data = JSON.parse(this.responseText);  
                            console.log(data);  
                            var birdSightData = getNatParkBirdSightData(data);  
                            if(birdSightData.length > 0){  
                                showBirdSightDataToUser(birdSightData);  
                            }  
                            else{  
                                showNoDataToUser(correctDate);  
                            }  
                        }  
                        else{  
                            showServerErrorToUser();  
                        }  
                    }  
                });  
  
                xhr.open("GET", "https://api.ebird.org/v2/data/obs/" + provData.code + "/historic/" + getDateRequest(date) + "?sppLocale=es");  
                xhr.setRequestHeader("X-eBirdApiToken", "dm3m200i26fm");  
  
                xhr.send();  
            }  
            else{  
                showIncorrectDateErrorToUser(correctDate);  
            }  
        }  
        else{  
            showNoDateSpecifiedToUser();  
        }  
    }  
    else{  
        showNoDataLoadedToUser();  
    }  
}
```

(Fuente: Elaboración propia)

2. Avistamiento de aves en los últimos 30 días

Esta petición se realiza cuando un usuario emplea el botón “Últimos 30 días” presente en la interfaz de usuario detallada más adelante. Esta interacción le permite al usuario conocer si se ha avistado en el parque natural algún ave en los últimos 30 días. La solicitud de esta información precisa, únicamente, del código de la provincia de Alicante para poder efectuarse.

En caso de obtener resultados, se emplea la función “getNatParkBirdSightData(birdSightData)” presente en la *Figura 140*, con el fin de obtener las aves que se han divisado en el parque natural siguiendo las mismas especificaciones que en el caso anterior. En la siguiente figura se observa la función empleada para la elaboración de esta petición.

Figura 142. NPBS-Petición de avistamiento de aves en los últimos 30 días

```
function getBirdSightDataLast30Days(){
    if(provData != null){
        var xhr = new XMLHttpRequest();
        xhr.withCredentials = false;

        xhr.addEventListener("readystatechange", function() {
            if(this.readyState === XMLHttpRequest.DONE) {
                if(this.status === 0 || (this.status >= 200 && this.status < 400)){
                    var data = JSON.parse(this.responseText);
                    console.log(data);
                    var birdSightData = getNatParkBirdSightData(data);
                    if(birdSightData.length > 0){
                        showBirdSightDataToUser(birdSightData);
                    } else{
                        showNoDataToUser("Últimos 30 días");
                    }
                } else{
                    showServerErrorToUser();
                }
            }
        });
        xhr.open("GET", "https://api.ebird.org/v2/data/obs/" + provData.code + "/recent/notable?back=30&sppLocale=es");
        xhr.setRequestHeader("X-eBirdApiToken", "dm3m200i26fm");
        xhr.send();
    } else{
        showNoDataLoadedToUser();
    }
}
```

(Fuente: Elaboración propia)

8.3.3. Elaboración de la interfaz de usuario

La interfaz de usuario o “UI” se corresponde con un formulario que presenta la siguiente estructura:

1. Un input de tipo fecha junto con un botón para enviar la información a la función de petición pertinente.
2. Un botón que detalla las aves que se han avistado en los últimos 30 días.

Esta interfaz se inserta justo debajo del calendario presente en la página de index del parque natural. El botón de los últimos 30 días presenta una funcionalidad externa y ajena al envío de datos por parte del formulario, ya que no precisa de ningún input con el que enviar información a la petición que se realiza.

La interfaz presenta un estilo estándar siguiendo el marcado por la página de index del parque natural y se ha colocado en esa posición con la finalidad de aumentar la usabilidad de la misma. Su justificación radica en que, al encontrarse justo debajo del calendario, es fácilmente visible y utilizable otorgando una visión de conjunto al aumento de web realizado. Esto es debido a que se ha empleado el calendario como interfaz de usuario para el primer user-script.

En las siguientes figuras se muestran las funciones empleadas para la elaboración de esta interfaz de usuario junto con su resultado final en la web.

Figura 143. NPBS-Creación del estilo CSS de la interfaz de usuario

```
function setCalendarInputStyle(calInput){  
    calInput.style.alignSelf = "center";  
    calInput.style.width = "60%";  
}  
  
function setButtonStyle(button){  
    button.style.alignSelf = "center";  
    button.style.width = "40%";  
    button.style.marginBottom = "1em";  
}  
  
function setFormTitleStyle(formTitle){  
    formTitle.style.fontSize = "1.5em";  
    formTitle.style.margin = "0.5em";  
    formTitle.style.textAlign = "center";  
    formTitle.style.color = "#000000";  
}  
  
function setFormStyle(form){  
    form.style.display = "flex";  
    form.style.flexDirection = "column";  
    form.style.justifyContent = "center";  
}
```

(Fuente: Elaboración propia)

Figura 144. NPBS-Creación de la interfaz de usuario

```
function createCalendarInput(){  
    var calInput = document.createElement("input");  
    calInput.setAttribute("type", "date");  
    setCalendarInputStyle(calInput);  
    return calInput;  
}  
  
function createButton(name){  
    var button = document.createElement("button");  
    button.innerHTML = name;  
    setButtonStyle(button);  
    return button;  
}  
  
function createLast30DaysButton(name){  
    var button = createButton(name);  
    button.addEventListener("click", function(){event.preventDefault(); getBirdSightDataLast30Days()}, false);  
    return button;  
}  
  
function createFormTitle(){  
    var formTitle = document.createElement("h3");  
    formTitle.innerHTML = "Avistamiento de Aves";  
    setFormTitleStyle(formTitle);  
    return formTitle;  
}  
  
function createForm(){  
    var form = document.createElement("form");  
    form.setAttribute("id", "birdSight");  
    form.addEventListener("submit", function(){event.preventDefault(); getBirdSightData(this.firstChild.nextSibling.value)}, false);  
    form.appendChild(createFormTitle());  
    form.appendChild(createCalendarInput());  
    form.appendChild(createButton("Comprobar"));  
    form.appendChild(createLast30DaysButton("Últimos 30 días"));  
    setFormStyle(form);  
    return form;  
}  
  
function createUserInterface(){  
    var form = createForm();  
    var parent = document.querySelectorAll("body .mini-calendar-mes ")[0];  
    parent.appendChild(form);  
}
```

(Fuente: Elaboración propia)

Figura 145. NPBS-Interfaz de usuario (web)

Avistamiento de Aves

The user interface consists of a date input field containing "dd/mm/aaaa" with a small calendar icon to its right. Below it is a grey rectangular button labeled "Comprobar". At the bottom is another grey rectangular button labeled "Últimos 30 días".

(Fuente: Elaboración propia)

8.3.4. Muestra de datos

La muestra de datos al usuario se compone de las siguientes situaciones:

- Error en el servidor de eBird.
- Error por la falta de obtención de datos.
- Error por la no indicación de fecha para la petición de avistamiento de aves en una fecha concreta.
- Error por especificar una fecha futura.
- Inexistencia de resultados.
- Exposición de las aves avistadas en el parque natural.
- Error por falta de datos geográficos.

En los siguientes apartados se detallará cada uno de los puntos anteriores.

Error en el servidor de eBird

En este caso se ha producido algún error en el API de eBird a la hora de realizar las diferentes peticiones de obtención de datos. Esta situación es indicada al usuario mediante un mensaje modal empleando la librería *SweetAlert2* al igual que en WPCS. Su estructura presenta, por tanto, las mismas características que la del primer user-script expuesto. La única diferencia existente consiste en la identificación de eBird como el servidor que ha provocado el error. En la *Figura 38* del apartado *Muestra de datos al usuario* del script WPCS se puede observar el formato de este pop-up.

Error por la falta de obtención de datos

Este error consiste en la no obtención del código correspondiente al municipio de Alicante necesario para realizar las diferentes peticiones a eBird. Este caso se muestra al usuario mediante un pop-up con el mensaje de “Cargando datos” bajo las mismas características que en WPCS. En la *Figura 40* se puede observar el formato de este mensaje modal.

Error por la no indicación de fecha

Este error es producido cuando un usuario trata de comprobar el avistamiento de aves en una fecha determinada sin especificar dicha fecha. Esta acción determina el envío de una cadena vacía como fecha para realizar la petición, dando lugar a resultados inesperados. La forma de indicar al usuario que debe especificar la fecha por la que quiere buscar es mediante un pop-

up con el mensaje: “No has indicado la fecha por la que quieras comprobar el avistamiento de aves”.

Cabe destacar que este pop-up contiene una función denominada “cleanTable()” la cual se encarga de borrar la tabla de avistamiento de aves mostrada en el sitio web en caso de que se haya creado. En los siguientes puntos se detallarán las características de esta función.

En las siguientes figuras se muestra la función empleada para la generación de este pop-up junto con su resultado final en el sitio web.

Figura 146. NPBS-Mensaje modal de no indicación de fecha (código)

```
function showNoDateSpecifiedToUser(){
    cleanTable();
    Swal.fire({
        title: '<strong>Fecha no especificada</strong>',
        icon: 'info',
        html:
            '<p>No has indicado la fecha por la que quieras comprobar el avistamiento de aves.</p>',
        showCancelButton: true,
        showCancelButton: false,
        focusConfirm: false,
        confirmButtonText:
            '<i class="fa fa-thumbs-up"></i> Aceptar',
        confirmButtonAriaLabel: 'Thumbs up, great!',
    });
}
```

(Fuente: Elaboración propia)

Figura 147. NPBS-Mensaje modal de no indicación de fecha (web)



(Fuente: Elaboración propia)

Error por especificar una fecha futura

Este error es producido cuando el usuario inserta una fecha futura para determinar el avistamiento de aves para una fecha concreta. Esta acción provoca un error a la hora de realizar la petición al servidor de eBird por lo que se realiza un filtrado para evitar esta situación, tal y como se ha comentado anteriormente.

En caso de producirse esta situación, se le indica al usuario mediante un pop-up que no se puede determinar el avistamiento de aves en una fecha futura. Además, se le especifica la fecha que ha seleccionado empleando el formato “dd-mm-aaaa”. Este formato se consigue empleando la función “putDateInCorrectFormat(date)” mostrada en la *Figura 148*.

En las siguientes figuras se muestran las funciones utilizadas para generar este pop-up junto con su resultado final en la web.

Figura 148. NPBS-Conversión de fecha al formato “dd-mm-aaaa”

```
function putDateInCorrectFormat(date){  
    var newDate = date.split("-");  
    newDate = newDate[2] + "-" + newDate[1] + "-" + newDate[0];  
    return newDate;  
}
```

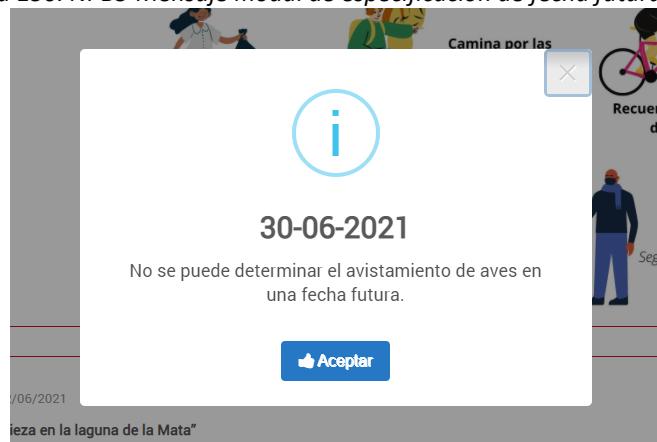
(Fuente: Elaboración propia)

Figura 149. NPBS-Mensaje modal de especificación de fecha futura (código)

```
function showIncorrectDateErrorToUser(date){  
    cleanTable();  
    Swal.fire({  
        title: '<strong>' + date + '</strong>',  
        icon: 'info',  
        html:  
            '<p>No se puede determinar el avistamiento de aves en una fecha futura.</p>',  
        showCancelButton: true,  
        showConfirmButton: false,  
        focusConfirm: false,  
        confirmButtonText:  
            '<i class="fa fa-thumbs-up"></i> Aceptar',  
        confirmButtonAriaLabel: 'Thumbs up, great!',  
    });  
}
```

(Fuente: Elaboración propia)

Figura 150. NPBS-Mensaje modal de especificación de fecha futura (web)



(Fuente: Elaboración propia)

Inexistencia de resultados

Esta situación se produce cuando se determina que no se ha avistado ningún tipo de ave en el parque natural. Esta información es indicada al usuario mediante un pop-up exponiendo la fecha que ha seleccionado el usuario o una cadena de texto correspondiente a los “Últimos 30 días” en caso de haber seleccionado esa opción en la interfaz de usuario. Dicha fecha o cadena de texto es especificada por parámetro en la función empleada para generar este mensaje modal.

En las siguientes figuras se muestra la función empleada para generar este pop-up junto con su resultado final en el sitio web.

Figura 151. NPBS-Mensaje modal de inexistencia de resultados (código)

```
function showNoDataToUser(date){  
    cleanTable();  
    Swal.fire({  
        title: '<strong>' + date + '</strong>',  
        icon: 'info',  
        html:  
            '<p>No se ha avistado ningún tipo de ave.</p>',  
        showCancelButton: true,  
        showConfirmButton: false,  
        focusConfirm: false,  
        confirmButtonText:  
            '<i class="fa fa-thumbs-up"></i> Aceptar',  
        confirmButtonAriaLabel: 'Thumbs up, great!',  
    });  
}
```

(Fuente: Elaboración propia)

Figura 152. NPBS-Mensaje modal de inexistencia de resultados (web)



(Fuente: Elaboración propia)

Exposición de las aves avistadas en el parque natural

En caso de que se hayan avistado aves en el parque natural se muestra dicha información atendiendo a las siguientes especificaciones:

1. Tabla en la que se muestran todos los datos relativos a cada ave divisada en el parque natural.
2. Pop-up en el que se visualiza un mapa de Leaflet junto con un marcador en la posición en la que se ha divisado el ave.
3. Limpieza de la tabla de datos en función de su previa existencia en el sitio web.

En los siguientes apartados se detallarán todos los puntos anteriores.

1. Tabla de avistamiento de aves

En esta tabla aparece toda la información relevante de las aves avistadas. Por ello, presenta las siguientes columnas:

1. Nombre común del ave.
2. Nombre científico del ave.
3. Número de aves divisadas de la misma raza.
4. Nombre de la localización en donde se han avistado.
5. Coordenada GPS de Latitud
6. Coordenada GPS de Longitud.
7. Fecha de avistamiento.
8. Un botón de visualización denominado “Mapa” con el que se mostrará el pop-up correspondiente al mapa de Leaflet.

Todos los campos, a excepción del botón de visualización, presentan un valor por defecto correspondiente a la cadena “Sin datos”. El motivo de esto es prevenir la falta de información de alguno de los campos de la tabla, mostrándole al usuario esta cadena en caso de que esto suceda. Con respecto al botón de visualización cabe decir que es un botón de HTML que se genera de manera automática para todas las filas de la tabla.

La tabla se genera mediante la librería jQuery denominada *DataTables* a partir de un elemento HTML “`<table></table>`”, por lo que el relleno de datos se genera automáticamente en función de los datos correspondientes al avistamiento de aves. Las columnas se especifican mediante el atributo “columns” propio de la librería y dentro de este atributo se indica la totalidad de columnas que se quiere mostrar junto con la información a exponer correspondiente a cada una de ellas. La inserción del botón denominado “Mapa” se realizó atendiendo al ejemplo de la documentación de *DataTables* denominado *Generated content for a column [65]*.

El diseño responsive de esta tabla mejora la usabilidad y la adaptabilidad de la información mostrada tanto para ordenadores como para dispositivos móviles. Este diseño se basa en ocultar aquellas columnas que no se pueden mostrar por falta de espacio, incluyendo un botón cuya función es ampliar hacia abajo la fila de la tabla con la finalidad de mostrar la información no visible.

En las siguientes figuras se indican las funciones utilizadas para generar esta tabla junto con su resultado final en la web.

Figura 153. NPBS-Creación del elemento HTML "<table></table>"

```
function createBirdSightTable(){
    var parent = document.getElementById("article_10155_80302856_170097689_1.1");
    var birdSightTable = document.createElement("table");
    birdSightTable.setAttribute("id", "birdSightTable");
    setBirdSightTableStyle(birdSightTable);
    parent.appendChild(birdSightTable);
}
```

(Fuente: Elaboración propia)

Figura 154. NPBS-Creación del estilo CSS que poseerá la tabla

```
function setBirdSightTableStyle(birdSightTable){
    birdSightTable.classList.add("display", "responsive", "nowrap");
    birdSightTable.style.width = "100%";
}
```

(Fuente: Elaboración propia)

Figura 155. NPBS-Creación del DataTable

```
function completeTable(birdsSightData){
    var birdSightTable = $("#birdSightTable").DataTable( {
        responsive: true,
        "data": birdsSightData,
        "columnDefs": [ {
            "targets": -1,
            "data": null,
            "defaultContent": "<button>Mapa</button>"
        }],
        "columns": [
            {title: "Nom Común", name: "comName", data: "comName", defaultContent: 'Sin Datos'},
            {title: "Nom Client", name: "sciName", data: "sciName", defaultContent: 'Sin Datos'},
            {title: "Núm Vistos", name: "howMany", data: "howMany", defaultContent: 'Sin Datos'},
            {title: "Nom Lo", name: "locName", data: "locName", defaultContent: 'Sin Datos'},
            {title: "Latitud", name: "lat", data: "lat", defaultContent: 'Sin Datos'},
            {title: "Longitud", name: "lng", data: "lng", defaultContent: 'Sin Datos'},
            {title: "Fecha", name: "obsDt", data: "obsDt", defaultContent: 'Sin Datos'},
            {title: "Visualización", name: "visualizar"}
        ]
    });
    $("#birdSightTable tbody").on("click", "button", function(){
        var parentRow = $(this).parents("tr").prev()[0];
        var rowData = birdSightTable.row(parentRow).data();
        if(rowData.lat != null && rowData.lat != undefined && rowData.lat != "" && rowData.lng != null && rowData.lng != undefined && rowData.lng != ""){
            showMapPopUpToUser(rowData);
        } else{
            showInsufDataErrorToUser();
        }
    });
}
```

(Fuente: Elaboración propia)

Figura 156. NPBS-Tabla de avistamiento de aves

Nom Común	Nom Cient	Núm Vistos	Nom Loc	Latitud
Aguilucho Lagunero Occidental	Circus aeruginosus	1	Lagunas de la Mata y Torrevieja PNat-Observatorio Fumarel	38.0444453
Avión Roquero	Ptyonoprogne rupestris	1	Lagunas de la Mata y Torrevieja PNat-Observatorio Fumarel	38.0444453
Bisbita Pratense	Anthus pratensis	5	Lagunas de la Mata y Torrevieja PNat-Observatorio Fumarel	38.0444453
Cernicalo Vulgar	Falco tinnunculus	1	Lagunas de la Mata y Torrevieja PNat-Observatorio Fumarel	38.0444453
Cisticola Buitrón	Cisticola juncidis	1	Lagunas de la Mata y Torrevieja PNat-Observatorio Fumarel	38.0444453
Cormorán Grande	Phalacrocorax carbo	14	Lagunas de la Mata y Torrevieja PNat-Observatorio Fumarel	38.0444453
Curruca Cabecinegra	Sylvia melanocephala	2	Lagunas de la Mata y Torrevieja PNat-Observatorio Fumarel	38.0444453
Gaviota Patiamarilla	Larus michahellis	1	Lagunas de la Mata y Torrevieja PNat-Observatorio Fumarel	38.0444453
Gaviota Picofina	Chroicocephalus genei	38	Lagunas de la Mata y Torrevieja PNat-Observatorio Fumarel	38.0444453
Jilguero Europeo	Carduelis carduelis	18	Lagunas de la Mata y Torrevieja PNat-Observatorio Fumarel	38.0444453

(Fuente: Elaboración propia)

2. Pop-up en el que se visualiza el mapa de Leaflet

Este pop-up se genera cuando el usuario clica en el botón denominado “Mapa” presente en cada una de las filas de la tabla. Tras suceder esta acción, se genera un elemento HTML “<div></div>” ubicado dentro del propio mensaje modal que contiene el mapa en cuestión. Se determina una altura y una anchura para este elemento con el fin de lograr la correcta visualización del mapa.

Dicho mapa se genera de la misma manera que en el script NPIM con la diferencia de que el tilelayer se origina ahora de manera estática indicando el copyright de Leaflet junto con el del laboratorio de ornitología de Cornell (<https://www.birds.cornell.edu/home>). Además, el zoom del mapa expresado en la constante “MAP_ZOOM” perteneciente al anterior user-script pasa ahora a tener un valor de 12. La creación del mapa se realiza en la función “createMap()” mostrada en la *Figura 158*.

La posición GPS desde donde se establece la vista del mapa posee el mismo valor que el indicado en el fichero JSON de NPIM, siendo [38.0122, -0.709444] correspondientes a unidades de latitud y longitud. Dicha posición se establece empleando la ya expuesta función “setMapView(leafletMap, view, zoom)” indicada en la *Figura 90* y utilizando una constante denominada “MAP_POS_GPS” con el valor indicado anteriormente.

Tras generar el mapa de Leaflet se origina el marcador con el que se especifica la posición del ave en el parque natural. Para elaborarlo, se ha creado un ícono propio que identifica de manera abstracta a cualquier tipo de ave. Este ícono posee forma de pájaro y presenta un

color verde propio de la naturaleza, aludiendo con ello a los parques naturales. Dicho icono se puede visualizar en la siguiente figura.

Figura 157. NPBS-Icono de avistamiento de aves



(Fuente: Icono de paloma [21])

Su inserción en el mapa de Leaflet posee la misma forma que en NPIM por lo que, para ello, se utiliza la función “createBirdIcon()” expuesta en la *Figura 159* con el fin de crear el icono que representará al marcador. Las únicas diferencias que presenta esta función con respecto a las de NPIM son la URL de la imagen a utilizar junto con el tamaño expuesto del propio ícono.

Este marcador presenta, además, otro pop-up propio que muestra los datos generales del ave que se pueden visualizar en la tabla. La información especificada presenta la siguiente estructura:

1. Título en el que aparece el nombre común del ave.
2. Párrafo en el que se muestra el nombre científico.
3. Si existe, párrafo en el que se indica el número de aves divisadas.
4. Si existe, párrafo en el que aparece la fecha de avistamiento del ave.

En función de la existencia de datos se mostrarán desde 2 puntos hasta la totalidad de los mismos, siendo obligatorios los 2 primeros. Este pop-up de Leaflet contiene las mismas indicaciones que en NPIM por lo que su ancho máximo será de 300. Este valor se encuentra definido en la constante “MAX_POPUP_WIDTH_LEAFLET”.

Con respecto a los eventos que harán aparecer este pop-up cabe decir que, al igual que en el user-script anterior, el evento “onclick()” queda eliminado utilizando únicamente el evento hover para la muestra de este mensaje modal. Para ello, se emplea la función “setMarkerHover(marker)” expuesta con anterioridad en la *Figura 103*.

La función para crear el marcador se denomina “createBirdMarker(leafletMap, birdData)” indicada en la *Figura 160* y la que establece su “pop-up” asociado se corresponde con “createBirdMarkerPopUp(birdData)” expuesta en la *Figura 161*. La función que engloba la generación de la totalidad del mapa de Leaflet junto con todas sus características se denomina “loadMap(birdData)” y se presenta en la *Figura 162*.

En las siguientes figuras se muestra el código necesario para la elaboración del pop-up que contiene el mapa de Leaflet junto con todas sus características. También se muestra el resultado final de dicho pop-up en el sitio web.

Figura 158. NPBS-Creación del mapa de Leaflet

```
function createMap(){
    var leafletMap = L.map('map');
    var tileLayerMap = L.tileLayer('https://s.tile.openstreetmap.org/{z}/{x}/{y}.png', {
        attribution: '&copy; <a href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors, &copy;';
    }).addTo(leafletMap);
    setMapView(leafletMap, MAP_POS_GPS, MAP_ZOOM);
    return leafletMap;
}
```

(Fuente: Elaboración propia)

Figura 159. NPBS-Creación del ícono de avistamiento de aves

```
function createBirdIcon(){
    var birdIcon = null;
    birdIcon = L.icon({
        iconUrl: "https://i.ibb.co/gdVH9k8/birdIcon.png",
        iconSize: [50, 50],
        iconAnchor: [25, 50],
        popupAnchor: [-3, -50]
    });
    return birdIcon;
}
```

(Fuente: Elaboración propia)

Figura 160. NPBS-Creación del marcador correspondiente al ave avistada

```
function createBirdMarker(leafletMap, birdData){
    var birdPos = JSON.parse("[ " + birdData.lat + ", " + birdData.lng + "]");
    console.log(birdPos);
    var marker = L.marker(birdPos, {icon: createBirdIcon(), alt: "Bird position marker"}).addTo(leafletMap);
    marker.bindPopup(createBirdMarkerPopUp(birdData), {maxWidth: MAX_POPUP_WIDTH_LEAFLET});
    setMarkerHover(marker);
}
```

(Fuente: Elaboración propia)

Figura 161. NPBS-Mensaje modal Leaflet (ave avistada)

```
function createBirdMarkerPopUp(birdData){
    var popUp = document.createElement("div");
    var title = "<h2>" + birdData.comName + "</h2>";
    var sciName = "<p><strong>Nombre Científico: </strong>" + birdData.sciName + "</p>";
    var num = "", date = "";
    if(birdData.howMany != null && birdData.howMany != undefined && birdData.howMany != ""){
        num = "<p><strong>Número de avistamientos: </strong>" + birdData.howMany + "</p>";
    }
    if(birdData.obsDt != null && birdData.obsDt != undefined && birdData.obsDt != ""){
        date = "<p><strong>Fecha y hora de observación: </strong>" + birdData.obsDt + "</p>";
    }
    popUp.innerHTML = title + sciName + num + date;
    return popUp;
}
```

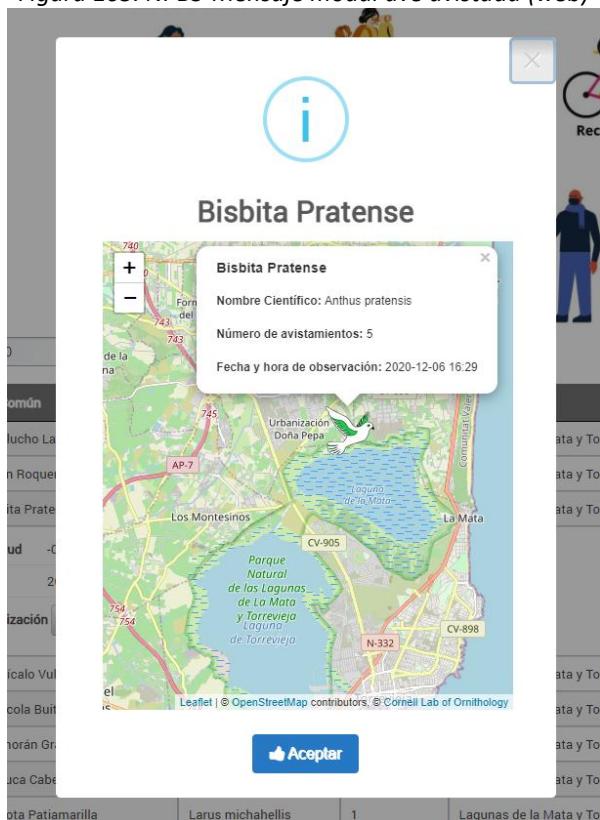
(Fuente: Elaboración propia)

Figura 162. NPBS-Creación completa del mapa de Leaflet

```
function loadMap(birdData){  
    var leafletMap = createMap();  
    createBirdMarker(leafletMap, birdData);  
}
```

(Fuente: Elaboración propia)

Figura 163. NPBS-Mensaje modal ave avistada (web)



(Fuente: Elaboración propia)

3. Limpieza de la tabla de avistamiento de aves

En el caso en el que se vaya a presentar un nuevo resultado de avistamiento de aves y ya haya una tabla que se esté mostrando en el sitio web, esta última debe eliminarse con el fin de especificar siempre los resultados correctamente. Esta limpieza consiste en destruir el “datatable” creado por la librería para después eliminar el elemento HTML “<table></table>” del DOM.

La función encargada de realizar esta tarea se denomina “cleanTable()” quien recurre internamente a la función “destroyTable(birdSightTableDom)”. Dicha función es llamada desde las siguientes situaciones con tal de reflejar siempre los resultados de la manera esperada:

1. Obtención de una nueva tabla de avistamiento de aves.
2. Inexistencia de resultados en una nueva búsqueda.
3. Error por la no inserción de fecha en una nueva búsqueda.
4. Error por la inserción de una fecha futura en una nueva búsqueda.

En la siguiente figura se muestra la creación de las funciones que eliminan la tabla de avistamiento de aves en caso de que exista.

Figura 164. NPBS-Eliminación del DataTable en caso de su existencia

```
function destroyTable(birdSightTableDom){
    $("#birdSightTable").DataTable().destroy();
    var parent = birdSightTableDom.parentNode;
    parent.removeChild(birdSightTableDom);
}

function cleanTable(){
    var birdSightTableDom = document.getElementById("birdSightTable");
    if(birdSightTableDom != undefined && birdSightTableDom != null){
        destroyTable(birdSightTableDom);
    }
}
```

(Fuente: Elaboración propia)

Error por falta de datos geográficos

Esta situación consiste en la falta de información correspondiente a la posición geográfica del ave que se ha avistado en el parque natural. Es decir, este error se produce en el caso de que la latitud o la longitud del ave no hayan sido proporcionadas por eBird mostrando en la tabla el resultado “Sin datos”.

Se destaca que, este caso es bastante excepcional ya que en la gran mayoría de los ocasiones esta información se obtiene con seguridad. Sin embargo, se ha decidido tener en cuenta esta situación con el fin de lograr una mayor robustez en el código elaborado para NPBS.

Si este error se produce, en lugar de mostrar al usuario un pop-up con el mapa de Leaflet se presenta un mensaje modal de error cuando haga clic sobre el botón “Mapa”. Esto es así debido a que no se posee la suficiente información como para poder poner el marcador del ave en el mapa, por lo que carece de sentido su exposición.

En las siguientes figuras se muestra la creación de este pop-up junto con su resultado final en el sitio web.

Figura 165. NPBS-Mensaje modal de insuficiencia de datos (código)

```
function showInsufDataErrorToUser(){
    Swal.fire({
        title: '<strong>Información insuficiente</strong>',
        icon: 'info',
        html:
            '<p>No se dispone de los datos de latitud o longitud necesarios para poder situar el ave en el mapa.</p>',
        showCloseButton: true,
        showCancelButton: false,
        focusConfirm: false,
        confirmButtonText:
            '<i class="fa fa-thumbs-up"></i> Aceptar',
        confirmButtonAriaLabel: 'Thumbs up, great!',
    });
}
```

(Fuente: Elaboración propia)

Figura 166. NPBS-Mensaje modal de insuficiencia de datos (web)



(Fuente: Elaboración propia)

Y con esto queda detallada la elaboración del script NPBS. En el siguiente apartado se expondrán los problemas encontrados en su desarrollo.

8.3.5. Problemas encontrados

A la hora de desarrollar NPBS han aparecido una serie de obstáculos que han dificultado su elaboración. Estas dificultades son:

1. Elaboración de los filtros para identificar las aves avistadas en el parque natural.
2. Inserción de las flechas de dirección mostradas en la tabla que sirven para reorganizar alfabéticamente los resultados.
3. Inclusión de 2 user-scripts en la misma página web.

A continuación, se expondrá detalladamente cada uno de los puntos anteriores.

1. Elaboración de los filtros para identificar las aves avistadas

En el desarrollo de NPBS se han tenido que utilizar numerosos filtros por nombre para identificar las aves avistadas en el parque natural. Se han tenido que emplear a pesar de tener un mayor porcentaje de error debido al propio funcionamiento del API de eBird.

EBird funciona mediante el almacenamiento de avistamientos de aves en distintas zonas geográficas. Pero dado que es un API internacional no puede trabajar en el orden de magnitud de los municipios. La identificación de municipios como Torrevieja se realiza mediante la utilización de determinadas id's que se corresponden con unas coordenadas de latitud y longitud determinadas.

Esto provoca que, a la más mínima variación en dichas coordenadas, se crea una nueva identificación para esa región del municipio. Generando con ello la imposibilidad de utilizar estas id's para la identificación de aves en zonas geográficas concretas.

Esta situación es perfectamente aplicable al parque natural de “Lagunas de la Mata y Torrevieja” puesto que experimentalmente se encontraron un total de 3 identificadores distintos para diferentes zonas de la reserva natural. Si lo que se busca es mostrar la totalidad de aves avistadas en toda el área geográfica de dicho parque, un filtro por id no era una solución viable debido a la gran cantidad de zonas geográficas que podrían desarrollarse en el mismo.

Debido a esto, se obtuvo por los filtros de nombre empleando las cadenas de texto identificadas a lo largo del desarrollo de este user-script.

2. Inserción de las flechas de dirección mostradas en la tabla

La tabla de avistamiento de aves presenta unas flechas de dirección que contienen la funcionalidad de reordenar alfabéticamente, en sentido ascendente o descendente, el contenido de cada columna. Sin embargo, estas flechas son imágenes pertenecientes a la propia librería que, en caso de enlazarla dinámicamente empleando Tampermonkey, no son accesibles por errores en la URL.

Debido a esto, la única manera de solucionar este inconveniente es modificar la propia librería sustituyendo las URL's relativas que se encuentran por defecto por URL's absolutas. De esta manera, las imágenes son encontradas por la librería a la hora de realizar la petición GET

pertinente, mostrándose correctamente en la tabla en cuestión. Esta solución es viable a causa de la licencia MIT de código abierto que posee la librería *DataTables* propia de jQuery.

3. Inclusión de 2 user-scripts en la misma página web

A la hora de ejecutar 2 user-scripts en la misma página web es posible encontrarse con la problemática de que solo se active 1 de los 2. Esta situación es debida al manejo de eventos propio de javascript.

En el caso en el que se emplee el manejo de eventos antiguo del tipo “window.onload = function()” no se podrá ejecutar más de un user-script a la vez. Su funcionamiento consiste en que la función “onload()” determinada para la carga de la ventana, en su versión antigua, es unitaria. Es decir, si se produce la situación de que 2 funciones “onload()” colapsan entre sí, únicamente se ejecuta 1 de las 2, siendo esta la última que se carga en el ciclo del navegador web. Es por esto por lo que se concluye que este manejo de eventos es sustitutivo en cuanto al número de eventos del mismo tipo que pueden producirse.

La manera de solucionar este inconveniente es empleando el manejo moderno de eventos de javascript identificado como los “addEventListener()”. Este manejo no es sustitutivo como el anterior sino que, en caso de duplicarse funciones de cualquier tipo para un determinado objeto javascript, dichas funciones se añaden a una lista o tabla virtual con el fin de ejecutarlas todas según su orden de aparición.

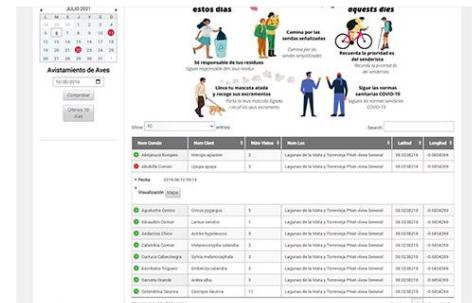
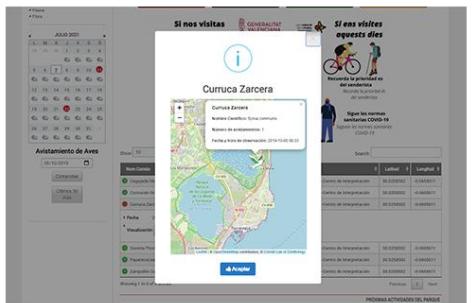
Esta tabla virtual es manejada por el navegador web y gracias a ella pueden colapsar eventos como el “window.onload” sin necesidad de que uno de los eventos no se ejecute. Debido a esto, se concluye con que la solución a este problema es sustituir el evento de llamada de la función que inicia el user-script por un “addEventListener()” para que así, en caso de tener más user-scripts ejecutándose, no colapsen entre sí dando lugar a errores de funcionamiento.

En el siguiente punto se detallan los cambios que se pueden visualizar en el sitio web tras la activación de este user-script.

8.3.6. Visualización de cambios

Como se ha comentado anteriormente, este user-script añade una interfaz con un input de tipo fecha que permite la obtención de las aves avistadas en el parque natural. Si se han enviado los datos correctamente y hay aves avistadas, se muestra una tabla con los resultados obtenidos. En la siguiente figura se observan los cambios con respecto al sitio web tras la activación de NPBS.

*Figura 167. NPBS-Comparativa de activación del user-script
NPBS*

<i>Sin activar</i>	<i>Activado</i>
	
<i>En ejecución 1</i>	<i>En ejecución 2</i>
	

(Fuente: Elaboración propia)

Como puede observarse esta tabla añade una mayor cantidad de información relativa a las aves avistadas que pueden encontrarse en el parque natural. Esto permite que aquellos usuarios que sientan cierto gusto por las aves se animen a visitar el parque natural si en sus registros se ha visualizado alguna en concreto que sea de su agrado.

En el siguiente punto se detallarán las pruebas realizadas para comprobar el correcto funcionamiento de NPBS.

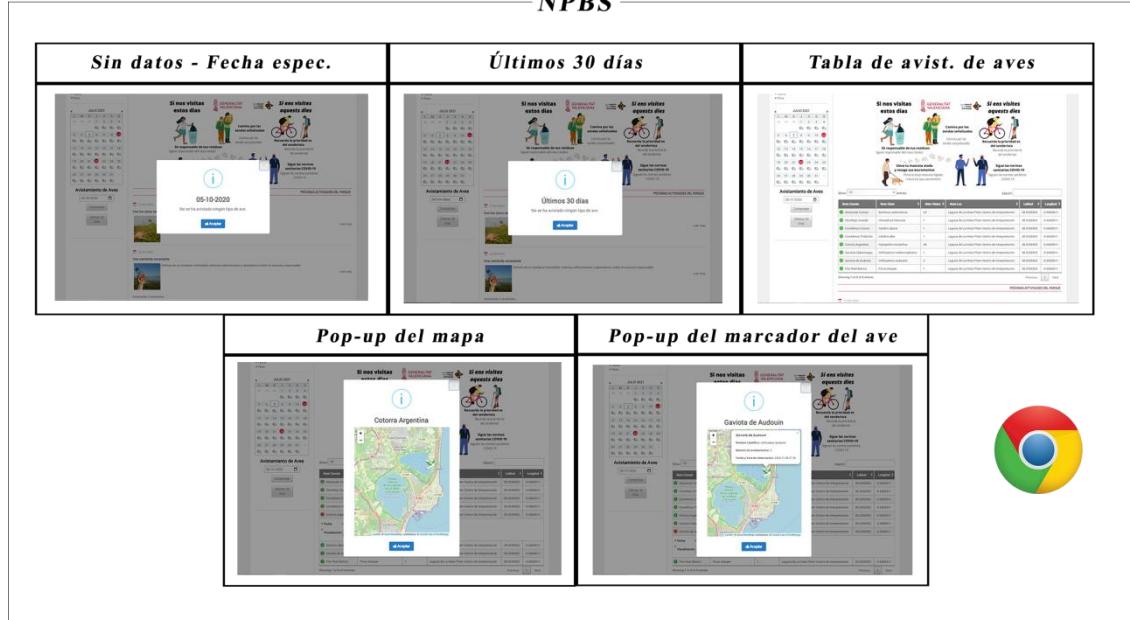
8.3.7. Pruebas realizadas

Con la finalidad de comprobar el correcto funcionamiento de NPBS, se realizaron una serie de pruebas en los navegadores: Google Chrome, Mozilla Firefox y Opera. Dichas pruebas se detallan a continuación:

- Muestra del mensaje modal correspondiente a la inexistencia de datos de una fecha seleccionada por el usuario.
- Muestra de los resultados correspondientes a las aves avistadas en los últimos 30 días.
- Muestra de una tabla de avistamiento de aves.
- Muestra del pop-up en el que aparezca el marcador del ave.
- Muestra del pop-up propio del marcador del ave.

En las siguientes figuras se visualizan las pruebas anteriores en los susodichos navegadores.

Figura 168. NPBS-Pruebas en navegador Google Chrome
NPBS



(Fuente: Logotipo de Google Chrome [14])

*Figura 169. NPBS-Pruebas en navegador Mozilla Firefox
NPBS*

Sin datos -Fecha espec.	Últimos 30 días	Tabla de avist. de aves
Pop-up del mapa	Pop-up del marcador del ave	

(Fuente: Logotipo de Mozilla Firefox [15])

*Figura 170. NPBS-Pruebas en navegador Opera
NPBS*

Sin datos - Fecha espec.	Últimos 30 días	Tabla de avist. de aves
Pop-up del mapa	Pop-up del marcador del ave	

(Fuente: Logotipo de Opera [16])

Como puede observarse, NPBS funciona sin mayores problemas según la experimentación realizada en los 3 navegadores. Las únicas diferencias visuales mínimas que se visualizan se producen en el navegador Mozilla Firefox, pero la representación de la información es idéntica en los 3 navegadores web.

También se destaca que, al igual que en el user-script NPIM, NPBS presenta problemas de carga en Mozilla Firefox. Esta situación puede ser debida a la falta de adaptación de Tampermonkey al navegador web o por la utilización de tecnologías como jQuery dentro de dicha extensión tal y como se ha comentado en NPIM. Sin embargo, una vez se carga NPBS en Mozilla Firefox funciona con las mismas prestaciones que en los navegadores Google Chrome y Opera, según se visualiza en las figuras anteriores.

Y con esto finaliza la exposición completa sobre la elaboración del script NPBS. En el siguiente punto se detallará el trabajo futuro que se puede derivar del desarrollo de estos user-scripts.

9. Conclusiones y trabajo futuro

Tras exponer el desarrollo completo del aumento de web realizado se detallan las conclusiones obtenidas con respecto al resultado final junto con el trabajo futuro a efectuar a raíz de esta base conseguida.

9.1. Conclusiones

Evaluando los objetivos propuestos para lograr el desarrollo adecuado de este TFG concluimos con que se han podido cumplir, en términos generales, con todos los planteados en dicho apartado.

Se han podido realizar los user-scripts que incorporan las funcionalidades correspondientes a la predicción de tiempo, al mapa interactivo y al avistamiento de aves. Además, se han podido utilizar las tecnologías planteadas en un inicio para su elaboración.

Se indica que también han podido alcanzarse los objetivos secundarios caracterizados por la creación de una interfaz amigable para el usuario. Se ha realizado una muestra de datos y una interfaz de usuario que contemplan las especificaciones de usabilidad y accesibilidad, todo ello adaptado a un diseño responsive que permite la ejecución de estos plugins en dispositivos móviles.

Además, para lograr una mayor facilidad de uso, se ha empleado en la medida de lo posible un diseño CSS compatible con el que posee el sitio web del parque natural.

Cabe destacar que, con respecto a las licencias correspondientes a las librerías y los recursos utilizados, se han empleado aquellas que se correspondan con el tipo “libre” tales como creative commons y MIT. Para los recursos, se ha investigado la legalidad correspondiente a su utilización y distribución con la finalidad de emplear aquellos que pudieran ser incluidos en el ámbito de este trabajo. Se ha tratado también que dichas licencias contengan la especificación de “utilizables bajo uso comercial” en la mayoría de imágenes incluidas.

También se indica que la elaboración de este proyecto ha servido para incrementar aún más los conocimientos sobre el lenguaje javascript elaborando estructuras con un mayor nivel de detalle tales como las clases. Además, el enfrentarse a una situación en la que no se podía

modificar el código fuente del sitio web ha servido para acrecentar los conocimientos relativos a la modificación de servicios web ya existentes.

Por último, se especifica que este TFG también ha servido para poseer mayores conocimientos correspondientes a la creación de aplicaciones web en forma de user-scripts empleados con el objetivo de aumentar las funcionalidades de los sitios web de Internet.

9.2. Trabajo futuro

El trabajo futuro a realizar consta de los siguientes apartados:

- Adaptación de la solución alcanzada a los sitios web de todos los parques naturales presentes en la Generalitat Valenciana.
- Corrección de los errores de carga de los user-scripts producidos en el navegador web Mozilla Firefox.
- Inclusión de un mayor número de navegadores en el que este aumento de web pueda ejecutarse correctamente.
- Creación de nuevas funcionalidades tanto generales como específicas que puedan servir para aumentar todavía más la experiencia del usuario a la hora de realizar una reserva en el parque natural.
- Publicación de los user-scripts en una plataforma como “openuserjs.org” para su libre descarga y utilización bajo las mismas condiciones que la licencia creative commons.

Realizando las directrices anteriores se puede obtener un producto mucho más elaborado para el sector del turismo digital en el campo de la aumentación web, siguiendo las especificaciones de los destinos turísticos inteligentes.

Referencias

1. Sitio web *easystaytech.com* - Las 5 etapas del viajero. Fuente: <http://easystaytech.com/las-5-etapas-del-viajero/>
2. Ivars-Baidal, J. A., Celdrán-Bernabeu, M. A., Mazón, J. N., & Perles-Ivars, Á. F. (2019). *Smart destinations and the evolution of ICTs: a new scenario for destination management?*. Current Issues in Tourism, 22(13), 1581-1600.
3. Definición *user-script*. Fuente: <https://en.wikipedia.org/wiki/Userscript>
4. Definición de *JavaScript* - Tutoriales y guías. Fuente: <https://developer.mozilla.org/es/docs/Web/JavaScript>
5. *Browser extensión* - Definition and history. Fuente: https://en.wikipedia.org/wiki/Browser_extension
6. *Plug-in (Computing)* - Definition. Fuente: [https://en.wikipedia.org/wiki/Plug-in_\(computing\)](https://en.wikipedia.org/wiki/Plug-in_(computing))
7. *Executable - Definition*. Fuente: <https://en.wikipedia.org/wiki/Executable>
8. Navegador *Google Chrome* - Definición e historia. Fuente: https://es.wikipedia.org/wiki/Google_Chrome
9. *Greasemonkey* - Definition and history. Fuente: <https://en.wikipedia.org/wiki/Greasemonkey>
10. Navegador *Mozilla Firefox* - Definición e historia. Fuente: https://es.wikipedia.org/wiki/Mozilla_Firefox
11. *Tampermonkey* - Definition and history. Fuente: <https://en.wikipedia.org/wiki/Tampermonkey>
12. *Tampermonkey - sitio oficial*. Fuente: <https://www.tampermonkey.net/>
13. *Greasemonkey - sitio oficial*. Fuente: <https://www.greasespot.net/>
14. *Violentmonkey - sitio oficial*. Fuente: <https://violentmonkey.GitHub.io/>
15. *Greasemonkey for Pale Moon* - sitio oficial. Fuente: <https://GitHub.com/janekptacijarabaci/greasemonkey/releases>
16. *Firemonkey* - sitio oficial. Fuente: <https://addons.mozilla.org/es/firefox/addon/firemonkey/>
17. *USI - sitio oficial*. Fuente: <https://addons.mozilla.org/es/firefox/addon/userunified-script-injector/>
18. *Awesome Userscripts - A curated list of Awesome Userscripts*. Fuente: <https://project-awesome.org/brunocvcunha/awesome-userscripts>

19. Navegador *Internet Explorer* - definición e historia. Fuente:
https://es.wikipedia.org/wiki/Internet_Explorer
20. *Adguard* - *Userscripts* information. Fuente:
<https://kb.adguard.com/en/general/userscripts>
21. *Document Object Model* - definición e historia. Fuente:
https://es.wikipedia.org/wiki/Document_Object_Model
22. *Statcounter - Browser Market Share*. Fuente: <https://gs.statcounter.com/browser-market-share>
23. Agencia Estatal de Meteorología (Aemet) - sitio oficial. Fuente:
<http://www.aemet.es/es/portada>
24. Formato JSON - definición y características. Fuente: <https://es.wikipedia.org/wiki/JSON>
25. *EBird - sitio web oficial*. Fuente: <https://ebird.org/spain/home>
26. Diseño Responsive - ¿Qué es el diseño responsive? Fuente:
<https://www.aeuroweb.com/que-es-diseno-responsive/>
27. Licencia Creative Commons (CC) - definición y características. Fuente:
https://es.wikipedia.org/wiki/Licencias_Creative_Commons
28. Licencia MIT - definición y características. Fuente:
https://es.wikipedia.org/wiki/Licencia_MIT
29. *Trello - sitio web oficial*. Fuente: <https://trello.com/es>
30. *GitHub - sitio web oficial*. Fuente: <https://GitHub.com/>
31. *AEMET OpenData - Sistema para la difusión y reutilización de la información de AEMET*. Fuente: <https://opendata.aemet.es/centrodedescargas/inicio>
32. Mensaje modal o pop-up - Concepto de Pop-up. Fuente:
<https://neoattack.com/neowiki/pop-up/>
33. *SweetAlert2 Library - A beautiful, responsive, customizable, accesible (WAI-ARIA) replacement for javascript's pop-up boxes*. Fuente: <https://sweetalert2.GitHub.io/>
34. Open source o código abierto - definición e historia. Fuente:
https://es.wikipedia.org/wiki/C%C3%B3digo_abierto
35. *Chart.js Library - Simple yet flexible JavaScript charting for designers & developers*. Fuente: <https://www.chartjs.org/>
36. *Dms.js Library - Geodesy representation conversion functions. (c) Chris Veness 2002-2019*. Fuente: <https://cdn.jsdelivr.net/npm/geodesy@2/dms.js>
37. *Geodesy Library - Movable Type Scripts*. Fuente: <https://www.movable-type.co.uk/scripts/geodesy-library.html>

38. Documentación oficial de Tampermonkey. Fuente:
<https://www.tampermonkey.net/documentation.php>
39. JavaScript Classes - syntax and examples. Fuente:
https://www.w3schools.com/js/js_classes.asp
40. Latlon-spherical.js Library - Latitude/longitude spherical geodesy tools. (c) Chris Veness 2002-2019. Fuente: <https://cdn.jsdelivr.net/npm/geodesy@2/latlon-spherical.js>
41. Imgbb - Servicio web para subir imágenes. Fuente: <https://es.imgur.com/>
42. Término insolación - Aemet. Fuente:
https://meteoglosario.aemet.es/es/termino/435_insolacion
43. Meteogram.es - Sitio web para averiguar la hora de salida y puesta de sol en Torrevieja. Fuente: <https://meteogram.es/sol/espana/torrevieja/>
44. Divulgometeo.es - El índice n de la precipitación intensa. Robert Monjo i Agut Departament de Física de la Terra i Termodinàmica. Universitat de Valencia. Fundació per la Investigació del Clima. Fuente: <https://www.divulgometeo.es/uploads/Indice-n.pdf>
45. Wikipedia - Lluvia. Diferenciar precipitación como débil, moderada, fuerte y muy fuerte. Fuente: <https://es.wikipedia.org/wiki/Lluvia>
46. Aemet - Acuerdo de legalidad. Fuente: http://www.aemet.es/es/nota_legal
47. Aemet - sección de ayuda. Interpretación: Predicción por municipios. Fuente:
<http://www.aemet.es/es/eltiempo/prediccion/municipios/ayuda>
48. Utilización de promesas en JavaScript. Fuente:
https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Using_promises
49. Librería Chart.js - Documentación oficial. Fuente: <https://www.chartjs.org/docs/latest/>
50. JavaScript ECMAScript (ES6). Fuente: https://www.w3schools.com/js/js_es6.asp
51. Declaración export - JavaScript. Sintaxis y descripción. Fuente:
<https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Sentencias/export>
52. Chart.js - Callback documentation. Fuente:
<https://www.chartjs.org/docs/latest/configuration/tooltip.html#labelCallback>
53. Librería Leaflet - sitio oficial. Fuente: <https://leafletjs.com/reference-1.7.1.html>
54. Librería jQuery - sitio oficial. Fuente: <https://jquery.com/>
55. Librería MapBox - sitio oficial. Fuente: <https://www.mapbox.com/mapbox-gl-js/>
56. Librería Carto.js - sitio oficial. Fuente: <https://carto.com/developers/carto-js/v3/>
57. Visor cartográfico del Institut Cartogràfic Valencià. Generalitat Valenciana. Fuente:
https://visor.gva.es/visor/?capasids=Espacios_Protegidos;9,6,8,7&extension=703636,4209987,706006,4211683&idioma=es

58. *GitHub Tampermonkey issue - how to load css file by GM method?* Fuente:
<https://GitHub.com/Tampermonkey/tampermonkey/issues/835>
59. *CORS - Control de acceso HTTP.* Fuente:
<https://developer.mozilla.org/es/docs/Web/HTTP/CORS>
60. *Question in StackExchange - Why my markers move when I resize my map on Leaflet?*
Fuente: <https://gis.stackexchange.com/questions/352480/why-my-markers-move-when-i-resize-my-map-on-leaflet>
61. *DataTables jQuery - sitio oficial.* Fuente: <https://datatables.net/>
62. *The Cornell Lab of Ornithology - Official site.* Fuente:
<https://www.birds.cornell.edu/home>
63. *Codificación ISO 3166-1 alfa-2.* Fuente: https://es.wikipedia.org/wiki/ISO_3166-1_alfa-2
64. *Estándar internacional de normalización ISO 3166-1.* Fuente:
https://es.wikipedia.org/wiki/ISO_3166-1
65. *DataTables Example - Generated content for a column.* Fuente:
https://datatables.net/examples/ajax/null_data_source.html

Recursos utilizados

1. Logotipo Tampermonkey. Wikimedia Commons. Fuente: https://commons.wikimedia.org/wiki/File:Tampermonkey_logo.svg. Licencia: GNU General Public License.
2. Logotipo Greasemonkey. Wikimedia Commons. Fuente: <https://commons.wikimedia.org/wiki/File:Greasemonkey.svg>. Licencia: MIT.
3. Logotipo Violentmonkey. GitHub. Fuente: <https://github.com/violentmonkey/violentmonkey/blob/master/src/resources/icon.png>. Licencia: MIT.
4. AlphaClouds Font by *Beeline*. Fuente: <https://www.fontspace.com/alphaclouds-font-f10179>. Licencia: Freeware
5. Icono de calendario por *jamesbhl*. Pixabay. Fuente: <https://pixabay.com/es/illustrations/calendario-icono-de-calendario-icono-3906791/>. Licencia: Pixabay License.
6. Icono de parcialmente nublado por *ArtsyBee*. Pixabay. Fuente: <https://pixabay.com/es/illustrations/nube-parcialmente-nublado-sol-2786100/>. Licencia: Pixabay License.
7. Icono de sol por *Clsru-Free-Vector-Images*. Pixabay. Fuente: <https://pixabay.com/es/vectors/sol-solares-energ%C3%ADa-la-luz-del-sol-41191/>. Licencia: Pixabay License.
8. Icono de sol, nube y lluvia por *raphaelsilva*. Pixabay. Fuente: <https://pixabay.com/es/vectors/sol-nube-la-lluvia-icono-verano-3000986/>. Licencia: Pixabay License.
9. Icono de tiempo soleado por *OpenClipart-Vectors*. Pixabay. Fuente: <https://pixabay.com/es/vectors/sol-clima-previs%C3%ADn-del-tiempo-157126/>. Licencia: Pixabay License.
10. Icono de tiempo soleado con precipitaciones por *OpenClipart-Vectors*. Pixabay. Fuente: <https://pixabay.com/es/vectors/clima-sol-nubes-157114/>. Licencia: Pixabay License.
11. Icono de tiempo nublado por *Clsru-Free-Vector-Images*. Pixabay. Fuente: <https://pixabay.com/es/vectors/nube-d%C3%ADa-oscuro-cludy-clima-37010/>. Licencia: Pixabay License.

12. Icono de tiempo lluvioso por *Clker-Free-Vector-Images*. Pixabay. Fuente: <https://pixabay.com/es/vectors/nube-clima-la-lluvia-lluvias-37011/>. Licencia: Pixabay License.
13. Icono de tiempo tormentoso por *TheUjulala*. Pixabay. Fuente: <https://pixabay.com/es/vectors/tormenta-flash-nubes-lluvia-1265161/>. Licencia: Pixabay License.
14. Logotipo de Google Chrome por *Wikimedialimages*. Pixabay. Fuente: <https://pixabay.com/es/vectors/google-chrome-logotipo-explorador-1326908/>. Licencia: Pixabay License.
15. Logotipo de Mozilla Firefox por *Clker-Free-Vector-Images*. Fuente: <https://pixabay.com/es/vectors/firefox-explorador-logotipo-fox-24921/>. Licencia: Pixabay License.
16. Logotipo de Opera por *Pixel Icons*. Iconscout. Fuente: <https://iconscout.com/icon/opera-53>. Licencia: Creative Commons 4 Attribution.
17. Icono de lupa con ubicación por *Megan_Rexazin*. Pixabay. Fuente: <https://pixabay.com/es/vectors/ubicaci%c3%b3n-la-tierra-mapa-mundo-4496459/>. Licencia: Pixabay License.
18. Icono de bandera por *benjsperry*. Icon-icons. Fuente: <https://icon-icons.com/es/icono/bandera/50393>. Licencia: Free for commercial use.
19. Icono de marcador sobre mapa por *Clker-Free-Vector-Images*. Pixabay. Fuente: <https://pixabay.com/es/vectors/mapa-pasador-illustrator-titular-42871/>. Licencia: Pixabay License.
20. Icono de ave por *mohamed_hassan*. Pixabay. Fuente: <https://pixabay.com/es/illustrations/ave-criaturas-la-vida-icono-2682039/>. Licencia: Pixabay License.
21. Icono de paloma por *OpenClipart-Vectors*. Pixabay. Fuente: <https://pixabay.com/es/illustrations/ave-criaturas-la-vida-icono-2682039/>. Licencia: Pixabay License.