

Universidad Internacional de La Rioja (UNIR)

ESIT

**Máster en Análisis y Visualización de Datos
Masivos**

RailML Dashboard

Trabajo Fin de Máster

presentado por: Sánchez Moreno, Adrián

Director/a: Martí, Julio Marcelo

Ciudad: Madrid

Fecha: 21 de Agosto de 2019

Resumen

Este trabajo recoge el desarrollo de una herramienta para la visualización de datos ferroviarios del tipo “RailML”. Una visualización que recoge un resumen del contenido de los datos para poder obtener una idea general del alcance del proyecto que se está visualizando. Se visualizarán datos sobre los elementos (agujas, señales, vías y pasos a nivel) del proyecto. Se ha optado por una herramienta de tipo web desarrollada con Python, en la que al usuario se le pide un archivo “RailML” y se genera automáticamente un “dashboard” con todas las visualizaciones. Gracias a esta herramienta se obtendrán visualizaciones sencillas con las que se podrán tomar mejores decisiones, además aventaja a otras herramientas debido a la sencillez de uso y la no necesidad de tener conocimientos ferroviarios.

This final Project encompasses the development of a visualization tool of railway data of the “RailML” type. The visualization covers a summary of the data content to obtain a general idea of the project scope that has been visualized. Data of elements (switches, signals, tracks and crossings levels) of the project will be visualized. The chosen technology has been a web tool developed in Python, in which the user is asked for a “RailML” file and automatically generates a “dashboard” with all the visualizations. Thanks to this tool it will be obtain simply visualizations to make better decisions, moreover the tool takes an advantage due to the simply usability and the non-necessary railway knowledge.

Palabras Clave: “dashboard”, “RailML”, sistemas ferroviarios, visualización de datos.

Índice de Contenido

Resumen.....	2
Introducción.....	5
Contexto y estado del arte.....	7
Objetivos	15
Metodología de trabajo.....	17
RailML.....	19
Requisitos	26
Requisitos no funcionales	27
Requisitos funcionales	27
Descripción de la herramienta	29
Despliegue de la herramienta.....	40
Control de configuración	41
Evaluación	45
Conclusiones.....	49
Líneas futuras	51
Referencias y enlaces	53
Anexos	56
Código de la aplicación web	56

Índice de Ilustraciones

Figura 1. Extracto de XML visualizado con Notepad++	7
Figura 2. Extracto de XML visualizado con Internet Explorer.....	8
Figura 3. Abrir XML con Excel.....	8
Figura 4. Extracto de XML visualizado con Microsoft Excel	8
Figura 5. Herramienta railVIVID.....	9
Figura 6. Herramienta OpenTrack.....	11
Figura 7. Herramienta railOscope.....	13
Figura 8. Teoría de las cinco estrellas.....	20
Figura 9 RailMI Dashboard App.....	29
Figura 10 Dialogo para elegir fichero.....	30
Figura 11 RailML Dashboard.....	31
Figura 12 Pop Up en el gráfico de barras.....	33
Figura 13 Signal Graph	33
Figura 14 Signals distribution graph	34
Figura 15 Tabla de señales	35
Figura 16 Tabla agujas.....	35
Figura 17 Limitaciones de velocidad de las agujas.....	36
Figura 18 Distribución de las agujas.....	37
Figura 19. Tabla de vías.....	37
Figura 20 Tipos de vías.....	38
Figura 21 Tamaño de las vías	38
Figura 22 Tabla paso a nivel	39
Figura 23 Distribución de los pasos a nivel	39
Figura 24 Leyenda Plotly.....	39
Figura 25. Ramas del proyecto GitHub.....	43
Figura 26. Ramas del repositorio.....	44

Introducción

En el mundo de la seguridad y la señalización ferroviaria se generan infinidad de datos en relación a los sistemas que entran en juego. Datos de señales, agujas, vías, conexiones, plataformas, balizas, redes, restricciones de velocidad, elementos de campo, paquetes de datos, aspectos de señales, etc. Se explicará a grandes rasgos el concepto de los elementos más importantes mencionados anteriormente en [“Descripción de la herramienta”](#).

Empresas como Thales, Siemens o Alstom tienen un formato de datos interno y diferente, lo que dificulta el intercambio de datos entre empresas, se pierde mucho tiempo y genera un gran coste el crear interfaces para que este intercambio sea posible.¹

Además estos datos son prácticamente ilegibles. Miles de líneas con información, siendo muy difícil su interpretación. Es complicado tener una idea general del proyecto simplemente echando un vistazo al fichero de datos, o imposible realizar tareas de validación.

En 2002 se funda “railML” (Railway Markup Language), cuyo objetivo es satisfacer las necesidades de los grupos de usuarios interesados en mantener un estándar internacional de la industria para el intercambio de datos ferroviarios. Es un formato abierto de intercambio de datos basado en XML (Lenguaje de Marcado Extensible). Gracias a este formato se soluciona el problema que se tenía en cuanto a la comunicación entre diferentes sistemas de distintas empresas.²

Este trabajo tiene como principal objetivo solucionar el problema de interpretación de los datos. Proponiendo una página web en la que cualquier usuario de este formato de datos pueda hacer uso de ella. Alimentando a la web con un fichero “railML” se mostrará un resumen de los datos, un “dashboard” en el que el usuario pueda hacerse una idea de cómo es el proyecto al que se enfrenta. Señales y sus aspectos, número de vías, agujas...información global del proyecto, que pueda ser útil para los usuarios de este formato de datos para que puedan tomar decisiones con echar un vistazo al “dashboard” generado.

Esta memoria constará de los siguientes capítulos, en los que se abordarán los siguientes temas:

¹ Background - railML.org (EN). (2019).

² Background - railML.org (EN). (2019).

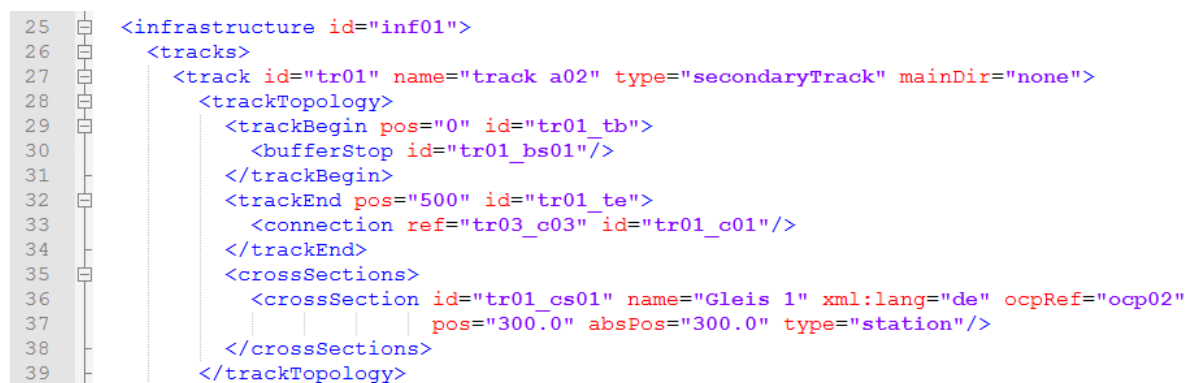
- El primer capítulo [“Contexto y estado del arte”](#) se estudiará el campo de los “dashboards” en el ámbito ferroviario, se hará un repaso a este tema durante toda su historia, se incluirán referencias, estudios, etcétera.
- El segundo capítulo [“Objetivos”](#) se centrará en los puntos a conseguir.
- El tercer capítulo [“Metodología de trabajo”](#) contemplará los pasos a seguir para conseguir los objetivos mencionados anteriormente.
- El cuarto capítulo [“RailML”](#) hará un repaso por este formato de datos, su historia, objetivos, licencias, etcétera.
- El quinto capítulo [“Requisitos”](#) fijará los requisitos software que cumplirá la herramienta a desarrollar.
- El sexto capítulo [“Descripción de la herramienta”](#) se detallará el desarrollo de la herramienta así como su funcionalidad e interfaz de usuario.
- El séptimo capítulo [“Despliegue de la herramienta”](#) explicará cómo y dónde se ha desplegado la aplicación.
- El octavo capítulo [“Control de configuración”](#) se centrará en cómo se ha gestionado el desarrollo de la herramienta.
- El noveno capítulo [“Evaluación”](#) validará la usabilidad de la herramienta y su aplicabilidad.
- El décimo capítulo [“Conclusiones”](#) englobará el alcance y la relevancia de la aportación.
- El undécimo capítulo [“Líneas futuras”](#) aportará ideas para continuar con este trabajo.
- El duodécimo capítulo [“Referencias y enlaces”](#) incluirá las pertinentes referencias y enlaces.
- El trigésimo capítulo [“Anexos”](#) incluirá el código o fragmentos del mismo.

Contexto y estado del arte

XML es un lenguaje de marcado similar a HTML (“HyperText Markup Language”). En inglés significa “Extensible Markup Language”, que quiere decir, “Lenguaje de marcado de extensible”. Es una especificación de W3C ³ (“World Wide Web Consortium”) como lenguaje de marcado de propósito general. Lo cual quiere decir que, a diferencia de otros lenguajes de marcado, XML no está predefinido, por lo que se tendrá que definir las etiquetas previamente. El propósito del lenguaje es compartir datos a través de diferentes sistemas. ⁴

Hay varias maneras de poder visualizar un fichero XML. Varias de ellas se detallan a continuación: ⁵

- Editor de texto: Haciendo “click” en el botón derecho en el fichero y pulsando “Abrir con” se listarán los programas con los que se puede abrir. Entre ellos están “Bloc de notas”, “Notepad++” o “Wordpad”. En la siguiente imagen podemos observar un extracto de un fichero XML abierto con un editor de texto.



```

25 <infrastructure id="inf01">
26   <tracks>
27     <track id="tr01" name="track a02" type="secondaryTrack" mainDir="none">
28       <trackTopology>
29         <trackBegin pos="0" id="tr01_tb">
30           <bufferStop id="tr01_bs01"/>
31         </trackBegin>
32         <trackEnd pos="500" id="tr01_te">
33           <connection ref="tr03_c03" id="tr01_c01"/>
34         </trackEnd>
35         <crossSections>
36           <crossSection id="tr01_cs01" name="Gleis 1" xml:lang="de" ocpRef="ocp02"
37             pos="300.0" absPos="300.0" type="station"/>
38         </crossSections>
39       </trackTopology>

```

Figura 1. Extracto de XML visualizado con Notepad++

- Navegador web: De la misma manera, botón derecho en el fichero y en “Abrir con” elegimos un Navegador web (“Internet Explorer”, “Google Chrome” o “Mozilla FireFox”). En la siguiente imagen podemos observar un fragmento de un fichero XML abierto con un navegador web.

³ World Wide Web Consortium (W3C). (2019).

⁴ Introducción a XML. (2019).

⁵ Cómo abrir un archivo XML. (2019).

```

- <infrastructure id="inf01">
- <tracks>
- <track id="tr01" mainDir="none" type="secondaryTrack" name="track a02">
- <trackTopology>
- <trackBegin id="tr01_tb" pos="0">
- <bufferStop id="tr01_bs01"/>
- </trackBegin>
- <trackEnd id="tr01_te" pos="500">
- <connection id="tr01_c01" ref="tr03_c03"/>
- </trackEnd>
- <crossSections>
- <crossSection xml:lang="de" id="tr01_cs01" type="station" name="Gleis 1" pos="300.0" absPos="300.0" ocpRef="ocp02"/>
- </crossSections>
- </trackTopology>

```

Figura 2. Extracto de XML visualizado con Internet Explorer

- Excel: Con Microsoft Excel abierto, seleccionamos “Abrir” y elegimos nuestro XML. Aparecerá una ventana emergente como la siguiente:

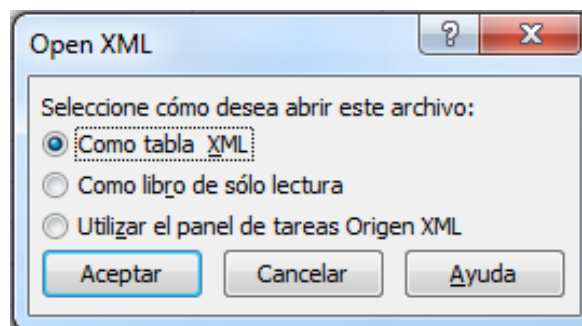


Figura 3. Abrir XML con Excel

Seleccionamos “Como tabla XML” y aceptamos. La siguiente imagen muestra cómo sería un fragmento un de un fichero XML abriéndolo con Excel.

	M	N	O	P	Q	R	S	T	U	V	W
1	id	code	id3	id4	name	type	mainDir	pos	id5	id6	ref
2	ima01	SZDC	inf01	tr01	track a02	secondaryTrack	none	0	tr01_tb	tr01_bs01	
3	ima01	SZDC	inf01	tr01	track a02	secondaryTrack	none				
4	ima01	SZDC	inf01	tr01	track a02	secondaryTrack	none				
5	ima01	SZDC	inf01	tr01	track a02	secondaryTrack	none				

Figura 4. Extracto de XML visualizado con Microsoft Excel

Además de estas maneras de visualización y edición de este tipo de ficheros, existen más programas específicos para su tratamiento. Por ejemplo, “XML Viewer” o “Altova”.

Sin embargo, las empresas que desarrollan sistemas y utilizan XMLs como archivos de datos de configuración, suelen crear sus propias herramientas para generar, editar, visualizar y validar dichos datos.

La aplicación que se desarrolla en este trabajo se utilizará para visualizar y conocer los datos que ya han sido generados previamente. Además también se podrá validar los datos que se visualicen. En futuros desarrollos de la aplicación se podrá llevar a cabo generación de informes para validar datos o funcionalidades específicas.

El propio “RailML” ya dispone de una propia herramienta para validación y visualización. Se llama “railVIVID” y su primera versión alfa fue presentada por TU Dresden (“Technische Universität Dresden”, Universidad Técnica de Dresde) y “railML.org” en la cuarta conferencia de “UIC RailTopoModel” en París el 29 de abril de 2015. Se puede descargar “railVIVID” en <https://www.railml.org/en/user/railvivid.html>, previamente hay que crearse un usuario en “railML” sin coste alguno.

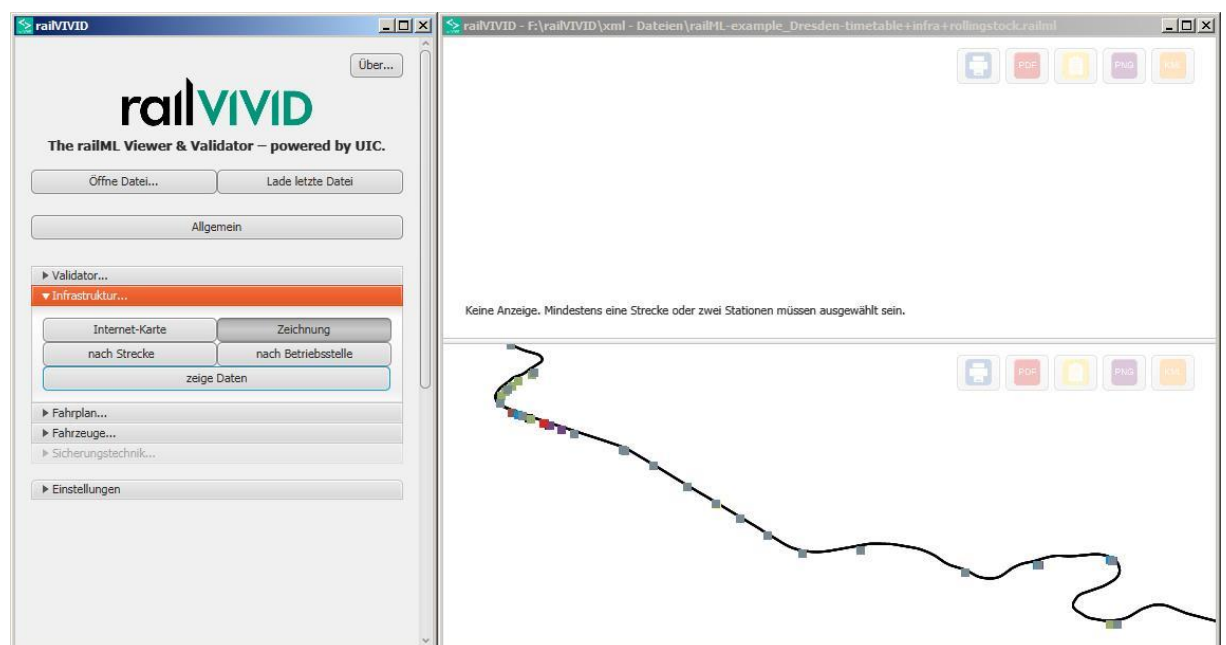


Figura 5. Herramienta railVIVID

Como podemos observar en la imagen anterior en la que se muestra una captura de la herramienta “railVIVID”, se visualiza la línea del proyecto, es decir la topología del proyecto ferroviario, con las estaciones que lo completan. Pero no hay ninguna opción disponible para tener una visión global y sintética del proyecto de los elementos que lo componen.

“railVIVID” es el resultado de un proyecto tras la convocatoria de propuestas de la UIC (International union of railways) en noviembre de 2014, el cual fue financiado por varios gestores europeos de infraestructuras ferroviarias. TU Dresden, en concreto el Instituto Telemático de Tráfico, ganó este proyecto y empezó el desarrollo de la herramienta “railVIVID”. Más adelante, “railVIVID” se entregará a la comunidad de “railML” y será desarrollado por esta comunidad.⁶

Con “railVIVID” se tiene el control total sobre los archivos “railML”. Se puede visualizar la infraestructura (topología, elementos del proyectos y sus características, ubicación de los elementos...), el calendario (días operativos de los trenes, calendario de mantenimiento...) y los elementos del material rodante (vehículos ferroviarios y sus características, formaciones de los juegos de trenes...) incluidos en el archivo “railML” incluso sin comprender completamente la sintaxis XML y “railML”. El validador de “railVIVID” permite verificar el archivo “railML” con respecto a la exactitud de la sintaxis, así como varios aspectos semánticos. Por lo tanto, “railVIVID” brinda toda la comodidad necesaria para trabajar con éxito con las interfaces de importación y exportación de “railML” con el software.⁷

“railVIVID” incluye:

- Visor gráfico de los datos del calendario.
- Visor tabular para datos de horario con exportación de hoja de cálculo.
- Visor de los datos de material rodante.
- Vista topológica de los datos de infraestructura.
- Vista geográfica de los datos de infraestructura.
- Validador del esquema de “railML”.
- Software en inglés y alemán con manual de usuario.

Existen también herramientas muy potentes de simulación y generación de datos ferroviarios como “OpenTrack”. Esta herramienta comenzó a mediados de los 90 como un proyecto de investigación de los “Ferrocarriles Federales Suizos”. Este proyecto tenía como propósito conseguir un modelo de ferrocarril y a la vez una herramienta sencilla para los usuarios que pudiera resolver preguntas sobre el ferrocarril mediante simulaciones.⁸

⁶ railVIVID - railML.org (EN). (2019). Disponible en <https://www.railml.org/en/user/railvivid.html>

⁷ railVIVID - railML.org (EN). (2019). Disponible en <https://www.railml.org/en/user/railvivid.html>

⁸ OpenTrack Railway Technology - Railway Simulation. (2019). Disponible en http://www.opentrack.ch/opentrack/opentrack_s/opentrack_s.html

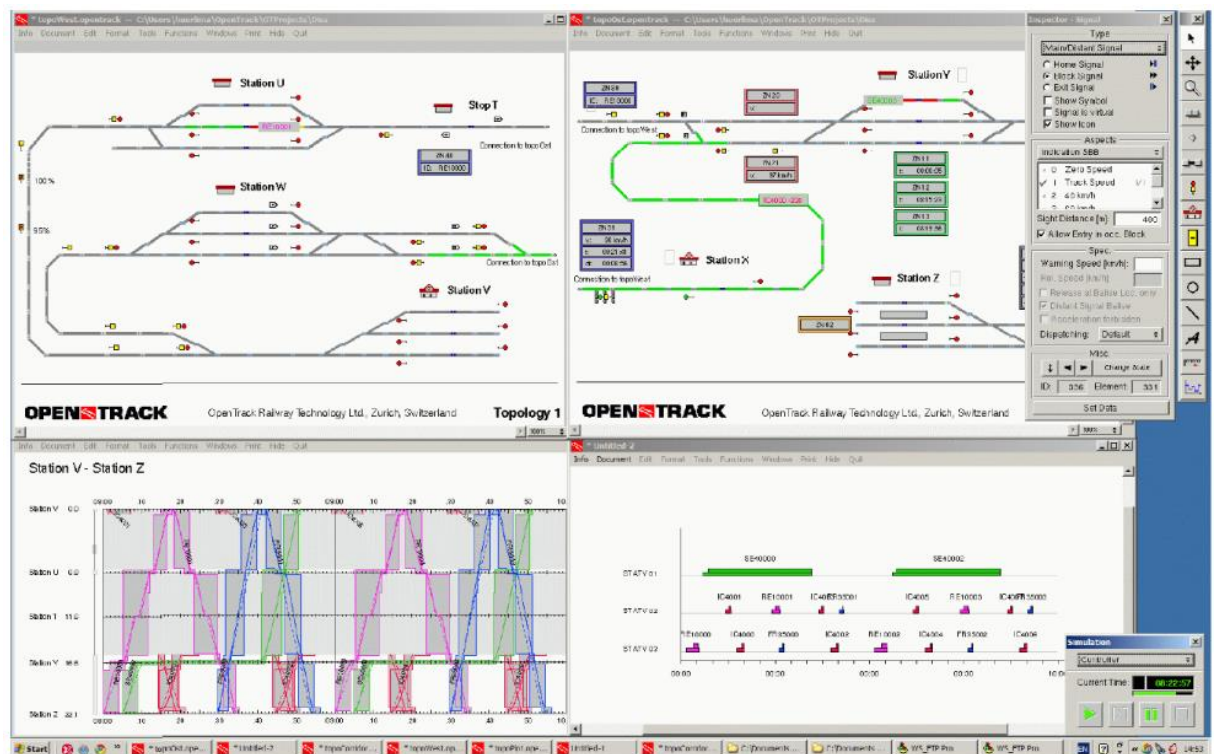


Figura 6. Herramienta OpenTrack.

La imagen anterior muestra una captura de la herramienta “OpenTrack”. Los dos cuadrantes superiores son estaciones y se muestran las vías, las señales y las agujas y como se relacionan entre sí. El cuadrante inferior izquierdo es una malla ferroviaria que sirve para la simulación de los recorridos de los trenes. El último cuadrante muestra las limitaciones temporales de velocidad. Como podemos observar no es una herramienta intuitiva ni fácil de usar. Se requiere un alto conocimiento ferroviario y además no muestra de manera sencilla y clara un resumen de los datos del proyecto.

En la actualidad, la herramienta “OpenTrack” es usada por administraciones ferroviarias como ADIF (Administración de Infraestructuras ferroviarias), SNCF (Société Nationale des Chemins de Fer Français); empresas ferroviarias como Siemens, Alstom, Thales o Bombardier; consultoras como Hatch o Sener y universidades de varios países como la Universidad Politécnica de Cataluña o la Universidad Pontificia de Comillas.

“Opentrack” es capaz de llevar a cabo las siguientes funcionalidades:

- Calcular los requisitos para una infraestructura ferroviaria.
- Estudiar la competencia de infraestructuras y estaciones.
- Investigación de reservas de material rodante (por ejemplo, futuras necesidades).

- Elaboración de horarios, estudiar la cohesión de los horarios (simulaciones simples o múltiples, Simulación “Monte – Carlo”.
- Estudio de varios sistemas de señales, como el sistema de cantones discretos, cantones cortos, LZB, ETCS, nivel 1, ETCS nivel 2, ETCS nivel 3 (cantones móviles) o el ERTMS.
- Considerar los efectos de fallos del sistema (como fallos de infraestructura o de trenes) y retrasos.
- Deducir el consumo de potencia imprescindible para los servicios ferroviarios.
- Simulación de sistemas de levitación magnética (como el “Transrapid”).

“OpenTrack” describe una red ferroviaria en unos gráficos específicos llamados “gráficos de doble vértice”. Se permite fijar la topología del sistema gráficamente. Los elementos del proyecto contienen varias particularidades. Un tramo, por ejemplo, está formado por su longitud, pendiente, velocidad máxima según los diferentes tipos de trenes. Se puede establecer y manipular objetos para tramos y desniveles, y además señales, agujas, estaciones y rutas.⁹

“OpenTrack” propone interfaces para los formatos de datos ordinarios, (como el ASCII, y XML), y para algunos de los formatos concretos de los ferrocarriles (incluyendo FBS, Protim, Simu VII, y otros). Además es compatible con archivos de tipo “railML”.

Otra herramienta digna de mencionar es “railOscope”. Esta herramienta es una plataforma basada en la nube para ver, editar y compartir datos ferroviarios (cronograma, infraestructura y material rodante).¹⁰

Esta herramienta te permite:

- Visualizar los datos.
- Publicar tus datos.
- Colaborar con la comunidad.

⁹ OpenTrack Railway Technology - Railway Simulation. (2019). Disponible en http://www.opentrack.ch/opentrack/opentrack_s/opentrack_s.html

¹⁰ railOscope - Features. (2019). Disponible en <https://railoscope.com/docs/features>

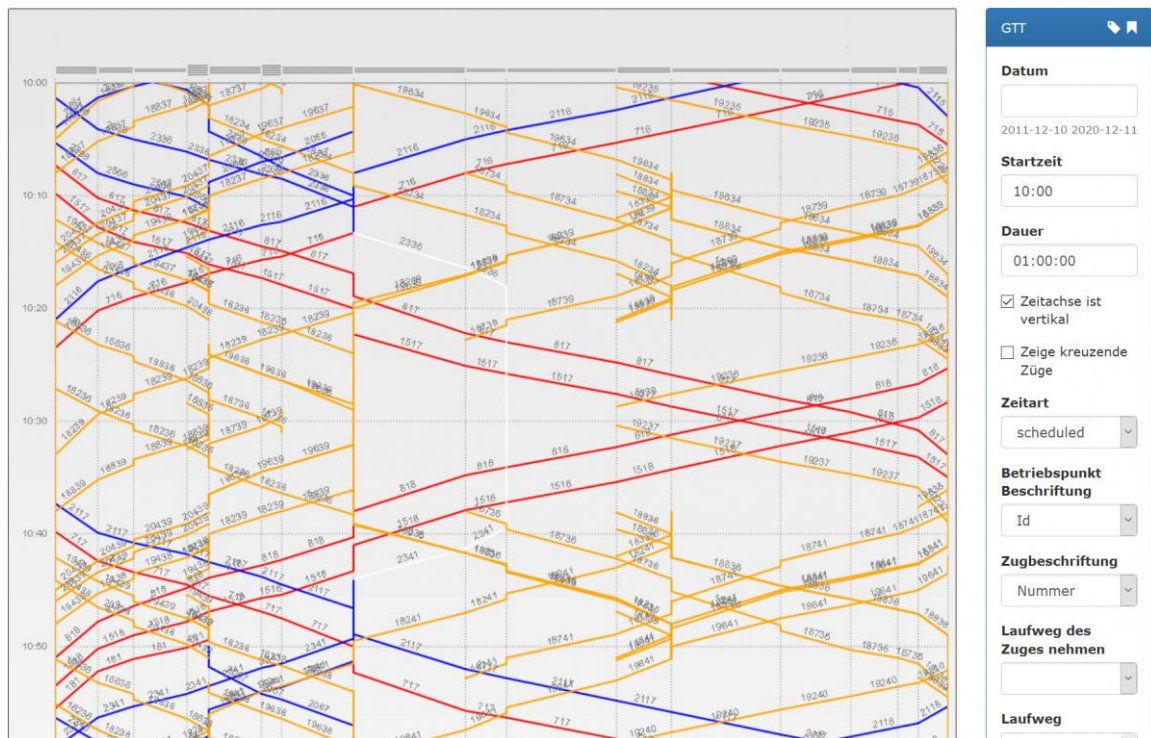


Figura 7. Herramienta railOscope

La imagen anterior muestra un extracto de la herramienta “railOscope” en la que se visualiza una estación de tren. En la que las líneas son las vías de la estación y cada color corresponde con un tipo de vía. Es un trabajo tedioso y poco legible el entender esta complicada visualización.

Además soporta formatos de datos “railML”; en todas sus versiones; datos en formato GTFS (“General Transit Feed Specification”, Especificación de alimentación de tránsito general), además de paquetes de datos “zip”, “bz”, “tar” o “gz”.¹¹

En la propia página de “railML.org” se menciona y se habla de esta herramienta. Se describen las compatibilidades con sus propios formatos de datos y explica la herramienta y sus funcionalidades, como que es capaz de analizar y validar datos “railML” o por ejemplo crear y editar la topología de las líneas ferroviarias con un simple mecanismo de “drag and drop”.¹²

Todas estas herramientas son útiles para generar, validar o visualizar todos los datos de los archivos “railML”, pero ninguna de ellas es útil para conseguir una visión general del

¹¹ railOscope - Features. (2019). Disponible en <https://railoscope.com/docs/features>

¹² railOscope - railML.org (EN). (2019). Disponible en <https://www.railml.org/en/introduction/applications/detail/railoscope.html>

proyecto. Lo que propone este trabajo es una herramienta para visualizar la información más general, conformando un “dashboard” que sirva de apoyo para la toma de decisiones.

Objetivos

El objetivo general de este trabajo es la generación de un “dashboard” en el que se recoja un resumen de la información guardada en un fichero de datos con formato “railML” para que los usuarios de este tipo de ficheros puedan de un vistazo saber qué es lo que esconden los datos.

Un “dashboard” es una sección de datos en el que los usuarios, empresas, consumidores o lectores contemplan la información relevante, en otras palabras, una representación esquemática de las trascendentales KPIs (“Key Performance Indicator”), permitiendo la mejora de la táctica de la empresa o usuario. El “dashboard” convierte los datos en conocimiento y facilita a los trabajadores la toma de decisiones.¹³

Otra definición de “dashboard” podría ser un instrumento para la administración de la información que controla, estudia y expone de un modo visual los indicadores básicos de desempeño (KPI), métricas y datos esenciales para llevar un control del estado de una empresa, un departamento, una maniobra o un proceso específico.¹⁴

Este trabajo intenta ayudar al problema de enfrentarse a un fichero de datos de gran volumen difícil de analizar sin herramientas o visualizadores. Miles de líneas tediosas de leer, en las que no se sacan conclusiones sólidas.

Este “dashboard” se desplegará al alimentar a una aplicación web con un fichero de datos de este tipo. Una vez que el usuario proporcione el fichero, se “parsearan” los datos, se analizarán, se hará una limpieza de los mismos, se harán cálculos, se determinará la información a mostrar, evaluando e identificando la información relevante.

Una vez hechas todas estas tareas previas, se desplegará el “dashboard” en la misma página web.

El “dashboard” recogerá la información mínima para poder tener una visión global del proyecto al que se enfrenta. Visualizando gráficas, tablas, nombre del proyecto, estaciones, conexiones, elementos del proyecto, etcétera. Con toda esta información sintetizada el usuario podrá tomar decisiones y conocer de manera sencilla cómo es el proyecto o a qué tipo de proyecto se está enfrentando.

¹³ Para qué sirve un 'dashboard'. (2019).

¹⁴ Ortiz, D. (2019). ¿Qué es un dashboard?.

Además una vez que se muestra el “dashboard”, el usuario podrá volver a cargar otro fichero si lo que desea es analizar otro fichero distinto.

Metodología de trabajo

A continuación se detallarán una serie de pasos a seguir para conseguir los objetivos descritos anteriormente.

Una de las primeras cuestiones a analizar es los datos que se quieren visualizar. El tipo de fichero, su procedencia, extensión, cómo se van a leer esos datos, etcétera.

Después de elegir el fichero con el que trabajar, se hará un estudio profundo de lo que abarca dicho fichero, eligiendo la información más importante del mismo para su posterior visualización.

Cuando ya conocemos los datos que queremos visualizar hay que pensar cual sería la mejor visualización para dichos datos, teniendo en cuenta si los datos a visualizar son variables cualitativas o cuantitativas, además de pensar en el nivel de conocimiento del usuario final para elegir una visualización u otra, ya sean gráficos de barras, de líneas, de dispersión o puntos, etcétera.

Una vez elegidos los datos a analizar y con los que se va a centrar la aplicación, se decidirán las herramientas más convenientes para llevarlo a cabo, lenguajes de programación, librerías, bases de datos, tecnologías...

Cuando ya se hayan tomado estas decisiones, se empezará a pensar en la codificación y en cómo se va a estructurar el código. Para ello se trabajará en los [requisitos](#), en los que se detallará todas las funcionalidades de la aplicación, así como especificaciones técnicas necesarias.

Para la codificación se diferenciarán los siguientes módulos a llevar a cabo:

- Lectura de los datos.
- Comprobación del esquema del fichero de datos.
- División de los datos en pequeños “data frames” para analizarlos por separados.
- Limpieza de los “data frames”.
- Cálculos necesarios.
- Visualización.

Una vez terminada la codificación se pasará a la fase de verificación, en la que se determinará si se cumplen los objetivos y los requisitos definidos. La verificación se

analizará con unos test funcionales, en los que se detallará cada prueba que se haga. Todo lo pertinente a la fase de verificación se detallará en el capítulo de [“Evaluación”](#).

RailML

El objetivo de “[railML.org](https://railml.org)” satisfacer las necesidades de los grupos interesados en un estándar industrial europeo para el intercambio de datos ferroviarios.

Esta iniciativa fue fundada en 2002 debido a la dificultad que se presentada a la hora de conectar diferentes aplicaciones informáticas ferroviarias. El primer esquema de datos “railML” fue desarrollado en 2005. Después de las primeras experiencias con el desarrollo y uso de railML 1.x, railML2.0 fue publicado en 2009. En 2014 comenzó una cooperación con [UIC](https://www.uic.org/) (International unión of railways), siguiendo a muchas empresas ferroviarias estatales (ÖBB, DB, SNFC, etcétera) para convertirse en socios de “railML”.

El desarrollo de “railML” tiene lugar en grupos de trabajos de expertos en el área del software ferroviario, modelización de datos e informáticos. Los grupos de trabajo están definidos por los cuatro diferentes esquemas de “railML”.

Cada grupo de trabajo se reúne por separado una vez cada dos o tres meses para discutir sobre diferentes problemas y futuros pasos a seguir en el desarrollo. La comunidad de “railML” se reúne dos veces al año en conferencias oficiales de “railML.org”.

La teoría de las cinco estrellas para datos abiertos, definida por Tim Berners-Lee, inventor de la Web e iniciador de los Datos Enlazados (Linked Data), define cinco niveles para datos abiertos en su teoría de las cinco estrellas (<https://5stardata.info/es/>), así como el coste y beneficio para consumidores y editores.¹⁵

La primera estrella define que como consumidor se pueden ver, imprimir y guardar localmente. Asociar los datos en cualquier otro sistema, transformar los datos como se quiera y divulgar los datos con cualquiera que se quiera. Siendo editor es fácil compartirlos, y no hace falta clarificar a los demás que pueden usar tus datos.

¹⁵ Home - railML.org (EN). (2019).

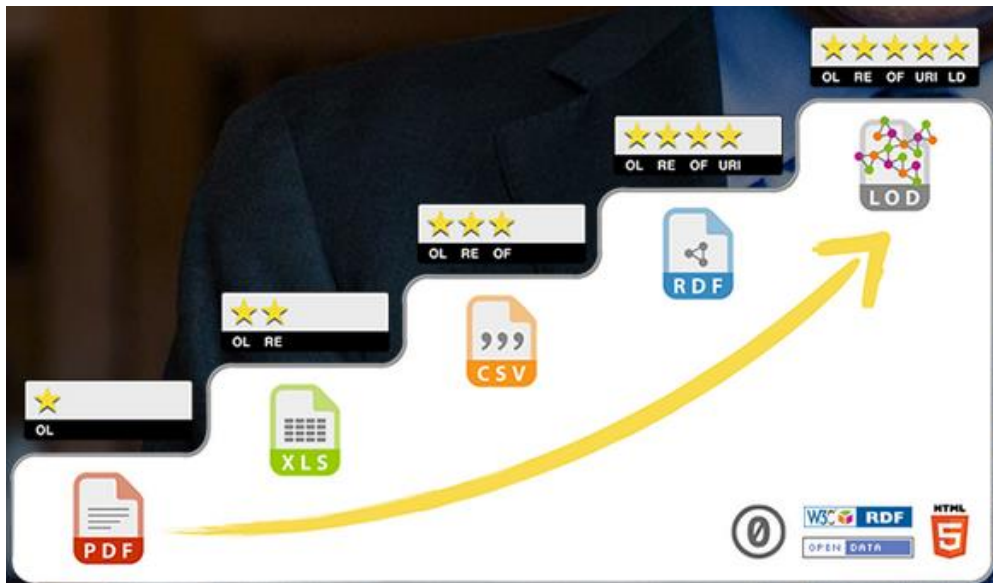


Figura 8. Teoría de las cinco estrellas

La imagen anterior muestra la evolución según se va aumentando de estrella en la “Teoría de las cinco estrellas”.

La segunda estrella, permite para el consumidor favorecerse de los beneficios de la primera estrella, además de poder gestionarlos abiertamente con software propio para incorporarlos, hacer operaciones, visualizaciones, etcétera. Como editor sigue siendo simple publicarlos.

Como consumidor en la tercera estrella, se sigue englobando todo lo explicado en la segunda estrella, además de poder manejar los datos de cualquier manera que se quiera, sin restricción de características o de uso de algún tipo de software específico. Como editor se podría precisar convertidores o “plug-ins” para cargar los datos del formato origen. Sigue siendo fácil compartirlos.¹⁶

La cuarta estrella sigue manteniendo las ventajas del nivel tres y además, se pueden enlazar a partir de otro sitio (Web o local), se pueden marcar como preferidos, se pueden volver a utilizar fragmentos de los datos, se puede sacar ventaja de las herramientas y librerías disponibles, se pueden mezclar unos datos con otros sin ninguna complicación. Las URI (Identificador de recursos uniforme) son una representación global por lo que si dos cosas tienen la misma URI entonces fue intencionado, y en tal caso se estará en proceso de alcanzar el nivel de cinco estrellas de datos. Como editor se tendrá el control sobre estos datos y se podrá mejorar su acceso, otros editores de datos pueden ahora enlazarse a tus datos, se tendrá que determinar URIs a los datos y deliberar en cómo representarlos.

¹⁶ 5 estrellas de Datos Abiertos. (2019).

El nivel cinco, como los demás niveles, gozará de las ventajas del nivel cuatro y además, se podrán revelar más datos (relacionados) al tiempo que se consumen los datos, se podrá ilustrar claramente sobre la estructura de los datos, ahora se tiene que manipular enlaces rotos de datos, como los errores 404 en las páginas web. Como editor se los datos se pueden hacer descubiertos, se podrá aumentar el valor de los datos y se podrán ganar las mismas ventajas de los enlaces como los consumidores.¹⁷

De acuerdo con esta teoría los esquemas de “railML” están disponibles como datos abiertos y no propietarios (“tres estrellas”) en la página web de “railML.org”. Lo que significa que los usuarios pueden manipular los datos libremente, sin la necesidad de poseer un paquete software.

“RailML” es un formato de intercambio de datos ferroviarios de código abierto. Esto significa que está disponible para cualquier persona de manera gratuita. La organización coordina el trabajo de “railML” y no es una empresa comercial con la intención de obtener beneficio.

Aunque varias empresas relacionadas con el negocio ferroviario, desarrolladores de software y otras organizaciones apoyan la idea de un modelo de intercambio de datos común, único y gratuito para el sector ferroviario, algunos servicios no se pueden proporcionar de manera gratuita o no pueden ser proporcionados a la comunidad.

A continuación se detalla una lista con los servicios gratuitos, los servicios de pago y los servicios que no ofrece “railML.org”.¹⁸

Los servicios gratuitos son:

- Acceso y descarga de los esquemas “railML”.
- Uso de los esquemas “railML” como parte de la interfaz del software o servicio.
- Acceso a la información de la comunidad y datos, con una cuenta de usuario en [“www.railML.org”](http://www.railML.org).
- Miembro de “railML.org”.
- Acceso y uso para propósitos de pruebas a ficheros de datos de ejemplo, los cuales se utilizan en el desarrollo de este trabajo.
- Acceso al fórum de “railML”.
- Participación en los grupos de desarrollo.
- Acceso a la wiki de “railML”.
- Presentación de propuestas de ampliación de esquemas “railML”.
- Participación en las conferencias y eventos de “railML”.

¹⁷ 5 estrellas de Datos Abiertos. (2019).

¹⁸ Home - railML.org (EN). (2019).

- Descarga y uso de “railVIVID”: The “railML viewer and validator”.
- Información esencial sobre “railML”, “railML.org” y el uso de “railML”.

Los servicios de pago son:

- Reuniones de presentación de productos, así como talleres generales y específicos sobre “railML”.
- Consultoría por expertos de “railML.org” sobre las mejores prácticas de gestión de datos / estrategia de intercambio.
- Adquisición de la certificación obligatoria (tarifa única) para el software o la interfaz “railML” antes de su uso.
- Costos por cesar y desistir de advertencias y honorarios legales causados por el uso indebido de la propiedad intelectual de “railML.org” y la violación de los términos de la licencia.

Los servicios que no son ofrecidos son:

- Programación para la implementación de interfaces de exportación o importación de “railML” en un software.
- Acceso a bases de datos de cronograma, infraestructura, enclavamiento o material rodante en formato de archivo “railML”.

“railML.org” es una asociación de desarrollo de empresas e instituciones independientes. Sus sugerencias, comentarios y críticas son las bases del desarrollo y consolidación de los esquemas de “railML”. Los socios se dividen en tres grupos: usuarios, desarrolladores y colaboradores. Algunos de los socios más importantes y conocidos son Adif, CAF, Siemens, Thales, Alstom o Bombardier.

La especificación de “railML” contiene esquemas para cuatro grandes áreas: infraestructura, cronograma, material rodante y enclavamiento. Estos cuatro subesquemas están divididos a su vez en más subesquemas que abordan áreas más específicas. Se están planificando y discutiendo subesquemas de “railML” adicionales como la lista de tripulantes, la gestión de activos o los datos en tiempo real.¹⁹

Un fichero “railML” puede contener otros subesquemas. La división en subesquemas permite que cada aplicación individual compatible con “railML” aplique las partes del esquema completo, por ejemplo, Los subesquemas - en varias combinaciones.

¹⁹ Home - railML.org (EN). (2019).

El subesquema de cronograma está centrado en la descripción del calendario ferroviario, incluidas todas las diversas facetas que necesitan las aplicaciones de intercambio de datos. En concreto, el esquema de cronograma contiene la siguiente información:

- Períodos de operación: Los días de operación para servicios de trenes o de clasificación.
- Partes del tren: las partes básicas de un tren como una secuencia de operación o puntos de control con las mismas características, como la formación y el período de operación. La parte del tren incluye la información real con respecto a la trayectoria del tren, así como la información del horario correspondiente.
- Trenes: Una o más partes del tren conforman un tren y representan la vista operativa o comercial de la carrera del tren.
- Lista: Las partes del tren se pueden vincular para formar las circulaciones necesarias para la clasificación (horarios de material rodante).

El subesquema de material rodante “railML” se centra en la descripción del material rodante de los ferrocarriles, incluidas todas sus diversas facetas que las aplicaciones de intercambio de datos consideran necesarias. En particular, el esquema de material rodante “railML” contiene la siguiente información:²⁰

- Vehículos: las características de los vehículos ferroviarios individuales o las familias de vehículos se describen en esta parte del esquema. La descripción de los vehículos considera algunos datos generales utilizados para organizar los activos, como la denominación, la clasificación o los números de los vehículos, tal como los da su operador. La mayoría del esquema se centra en la estructura para almacenar los diversos aspectos técnicos de los vehículos ferroviarios con respecto a su sistema de propulsión, características de la carrocería, frenos o servicios instalados dentro del vehículo.
- Formaciones: las características de los juegos de trenes o partes de los mismos formados por varios vehículos diferentes o similares se describen en esta parte. Esta combinación de vehículos se utiliza para describir las características del tren según sea necesario. Sin embargo, la consistencia lógica entre la formación y los vehículos de los que está hecha no se aplica por el esquema. Debe ser asegurado por la aplicación que produce los datos.

²⁰ Home - railML.org (EN). (2019).

El subesquema de infraestructura “railML” se centra en la descripción de la infraestructura de la red ferroviaria que incluye todas sus diversas facetas que son necesarias para las aplicaciones de intercambio de datos. Concretamente este esquema contiene la siguiente información:

- Topología: La red de seguimiento se describe como un modelo de borde de nodo topológico.
- Coordenadas: todos los elementos de la infraestructura ferroviaria pueden ubicarse en un sistema de coordenadas de dos o tres dimensiones arbitrario.
- Geometría: La geometría de la vía puede ser descrita en términos de radio y gradiente.
- Los elementos de infraestructura ferroviaria: incluyen una variedad de activos relevantes para ferrocarriles que se pueden encontrar debajo, sobre o cerca de la vía férrea.
- Otros elementos ubicados: engloban elementos que están estrechamente relacionados con la infraestructura ferroviaria.

El esquema de enclavamiento “railML” se centra en la información que los administradores de infraestructura suelen mantener en los planes de señal y las tablas de bloqueo de ruta. Los principales usuarios de este esquema son proveedores interconectados y simuladores.²¹

En este subesquema se recoge la siguiente información:

- Preparación de datos: es el proceso de adaptar un sistema de enclavamiento y señalización ferroviaria a un proyecto específico. Los errores en los datos afectan la seguridad; es demasiado obvio que un aspecto incorrecto de la señal puede causar accidentes terribles. Por eso se invierten mucho tiempo y esfuerzo en las pruebas. Un formato de intercambio de datos estándar permitirá la automatización de la transferencia de datos y reducirá el número de errores al eliminar el factor humano. Esto creará mayores niveles de seguridad a un costo sustancialmente menor.
- Los programas de simulación: calculan el impacto del enclavamiento y la configuración de la seña. Cosas como cambiar una señal, usar un impulso de puntos más rápido o acortar bloques pueden tener un impacto significativo. Los algoritmos de simulación son cada vez más potentes y alcanzan un nivel de precisión donde los segundos son importantes. En la actualidad, el comportamiento en tiempo real del enclavamiento y la señalización a menudo es desconocido. El esquema railML IL

²¹ Home - railML.org (EN). (2019).

permite a los modelistas absorber rápidamente información sobre los sistemas de enclavamiento, como el comportamiento de la sincronización y las rutas, y analizar el impacto en la capacidad del ferrocarril.²²

El subesquema de infraestructura es en el que se centrará este trabajo, al ser de gran importancia los datos de configuración del enclavamiento y los elementos de campos con los que se interactúa.

²² Home - railML.org (EN). (2019).

Requisitos

Los requisitos en el desarrollo de aplicaciones informáticas son un punto clave en el proyecto. Muchos de estos proyectos fracasan ya que los requisitos se definen, se especifican o se administran de manera incorrecta. Requisitos incompletos o una mala administración de los cambios de los mismos son motivos más que suficientes para fracasar.

El Glosario de Terminología Estándar de Ingeniería de Software (IEEE: Standard Glossary of Software Engineering Terminology) define al requisito como:

1. Condición o capacidad que necesita un usuario para resolver un problema o lograr un objetivo.
2. Condición o capacidad que tiene que ser alcanzada o poseída por un sistema o componente de un sistema para satisfacer un contrato, estándar, u otro documento impuesto formalmente.
3. Una representación en forma de documento de una condición o capacidad como las expresadas en 1 o en 2.

Puesto que los requisitos son las mínimas exigencias del producto a llevar a cabo en cualquier proyecto de software, es significativo tener en cuenta que un requisito tiene que ser definido por escrito como cualquier contrato o convenio entre dos partes; se tiene que poder examinar o verificar para concluir que se satisface el requisito; además no se deben contradecir con otros requisitos y deben ser concisos, es decir, cómodo de leer y razonar. Asimismo, un requisito tiene que estar completo, o sea, que garantice la información necesaria para su entendimiento. Finalmente no debe ser ambiguo ya que se puede confundir al lector.²³

En el documento ERS (Especificación de requisitos software) es donde se deben registrar las necesidades del negocio (requisitos del cliente y necesidades del usuario) y se declaran los requerimientos que tiene cumplir el producto, sistema o software a llevar a cabo. Por lo que, es el medio de comunicación entre los clientes, usuarios y desarrolladores.

Los requisitos no funcionales o atributos de calidad son propiedades o condiciones que el producto debe cumplir, es decir, limitaciones que el producto que está siendo desarrollado

²³ Requisitos de Software - EcuRed. (2019).

debe tener. Por ejemplo, restricciones de sistema operativo, de ambiente, rapidez, seguridad, usabilidad, etc.²⁴

Los requisitos funcionales son las necesidades que el software debe cumplir de manera satisfactoria, es decir, las funciones que el software será capaz de realizar. Estos requerimientos son cálculos, detalles técnicos, manipulaciones de datos (entrada y salida) y otras funciones específicas que se supone, que el software debe cumplir.²⁵

A continuación se detallan los requisitos funcionales y no funcionales.

Requisitos no funcionales

- La aplicación se desarrollará en Windows 10.
- La aplicación será una aplicación web.
- La aplicación será compatible con la mayoría de navegadores (mínimo Google Chrome y Mozilla Firefox).
- La aplicación se desarrollará en Python 3.
- Se utilizará la librería “*dash*”, para la creación de “dashboards”.
- Se utilizará la librería *pandas*, para limpieza de datos y data frames.
- Se utilizará la librería “*element tree xml*” para la lectura de datos del fichero XML (Lenguaje de Marcado Extensible).

Requisitos funcionales

- El usuario será capaz de proporcionar el fichero de datos que desee, siempre y cuando el fichero cumple el esquema “railML”, y contenga el subesquema de infraestructura.
- El usuario será capaz de volver a proporcionar un nuevo fichero.
- El “dashboard” se generará automáticamente, una vez que el usuario proporcione el archivo, sin necesidad de más interacción por parte del usuario.
- El “dashboard” tendrá una cabecera con el nombre del archivo, del proyecto y datos generales.

²⁴ ¿Cómo escribir un buen documento de especificación de requisitos de software? - Medium. (2019).

²⁵ ¿Cómo escribir un buen documento de especificación de requisitos de software? - Medium. (2019).

- El “dashboard” contendrá información del subesquema de infraestructura.
- El “dashboard” contendrá tablas y gráficos de los elementos del enclavamiento tales como agujas, señales, vías, etcétera.
- Dichas tablas y gráficos deberán resumir y recabar la información sobre los elementos de campo.

Descripción de la herramienta

En este capítulo se desarrollará una detallada descripción de cómo funciona la herramienta. Además de describir que significa cada gráfico y tabla que contienen el “dashboard”.

El código de la herramienta se puede ver en [anexos](#).

Cuando abrimos la aplicación web visualizamos lo siguiente:

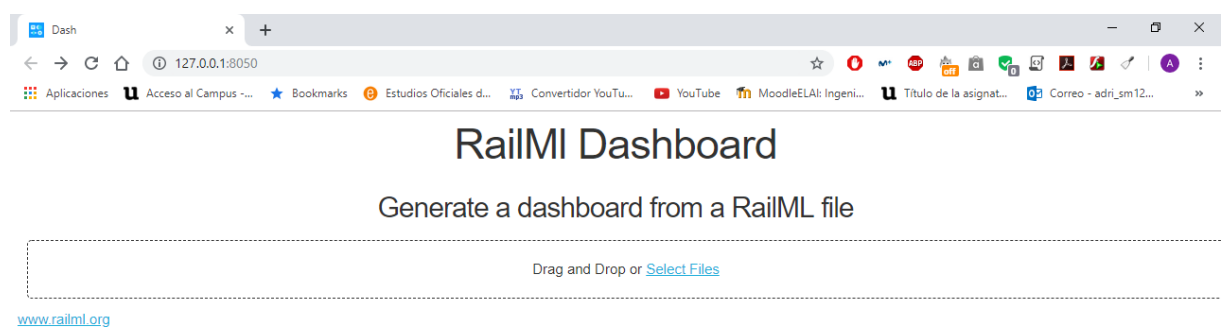


Figura 9 RailMI Dashboard App

La Figura 9 corresponde desde la referencia en el código [Ref 1.0](#) hasta la referencia [Ref 1.1](#).

Como podemos observar lo primero que visualizamos son dos títulos “RailMI Dashboard” y “Generate a dashboard from a RailML file”. El primer título corresponde con un elemento de cabecera de HTML H1 (cabecera de mayor importancia) y el segundo título corresponde a una cabecera H2.

Las cabeceras HTML son etiquetas HTML que se utilizan para concretar la cabecera o título de una página. Las cabeceras se definen con <Hn> donde n es un número entre 1 y 6 que establece el orden de la cabecera.²⁶

Lo siguiente que nos encontramos en la página es un espacio en el que se puede soltar un fichero o pulsar “Select Files” lo que abrirá un diálogo para seleccionar nuestro fichero. Este elemento se ha conseguido con un componente del tipo “Upload” de la librería “Dash”.

²⁶ ¿Qué son las Cabeceras HTML? - Ryte Digital Marketing Wiki. (2019).

El componente “Upload” de “Dash” permite a los espectadores de su aplicación cargar archivos, como hojas de cálculo de Excel o imágenes, en su aplicación.²⁷

El siguiente elemento de la página es una referencia hacia la página oficial de “RailML” donde el usuario podrá pulsar y acceder directamente dicha página y descargar archivos “RailML” para poder probar la aplicación o documentación para poder entender “RailML” y todo lo que conlleva.

Una vez que el usuario pulsa en “Select Files” se abrirá el diálogo que se muestra en la Figura 10:

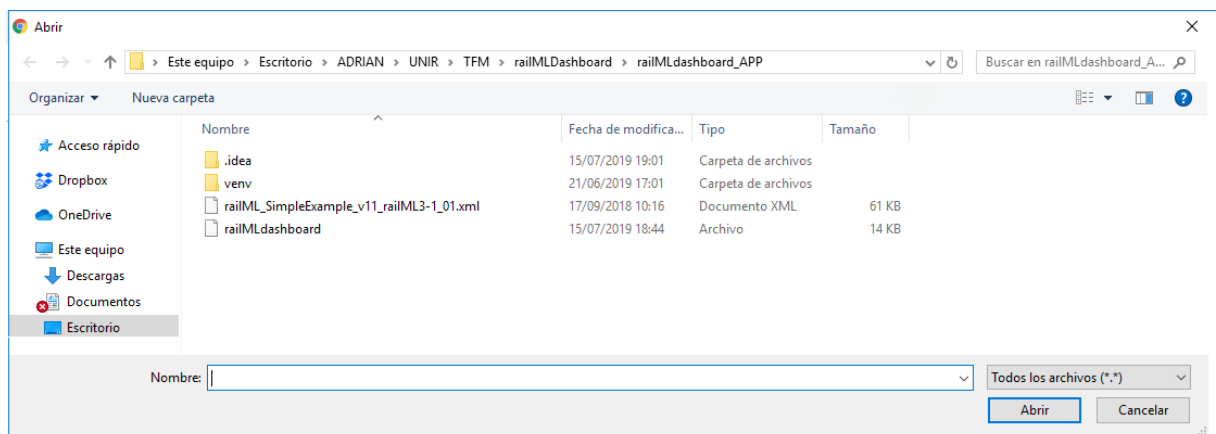


Figura 10 Diálogo para elegir fichero.

Una vez que el usuario elige el archivo “RailML” se llama a la función [“update-output”](#) que a su vez llama a la función [“parse-content”](#), donde se recoge la información que se encuentra en el fichero que facilita el usuario y luego se definen las tablas y gráficos a desplegar utilizando los datos recogidos, dando como resultado lo siguiente:

²⁷ Dash User Guide and Documentation - Dash by Plotly. (2019).

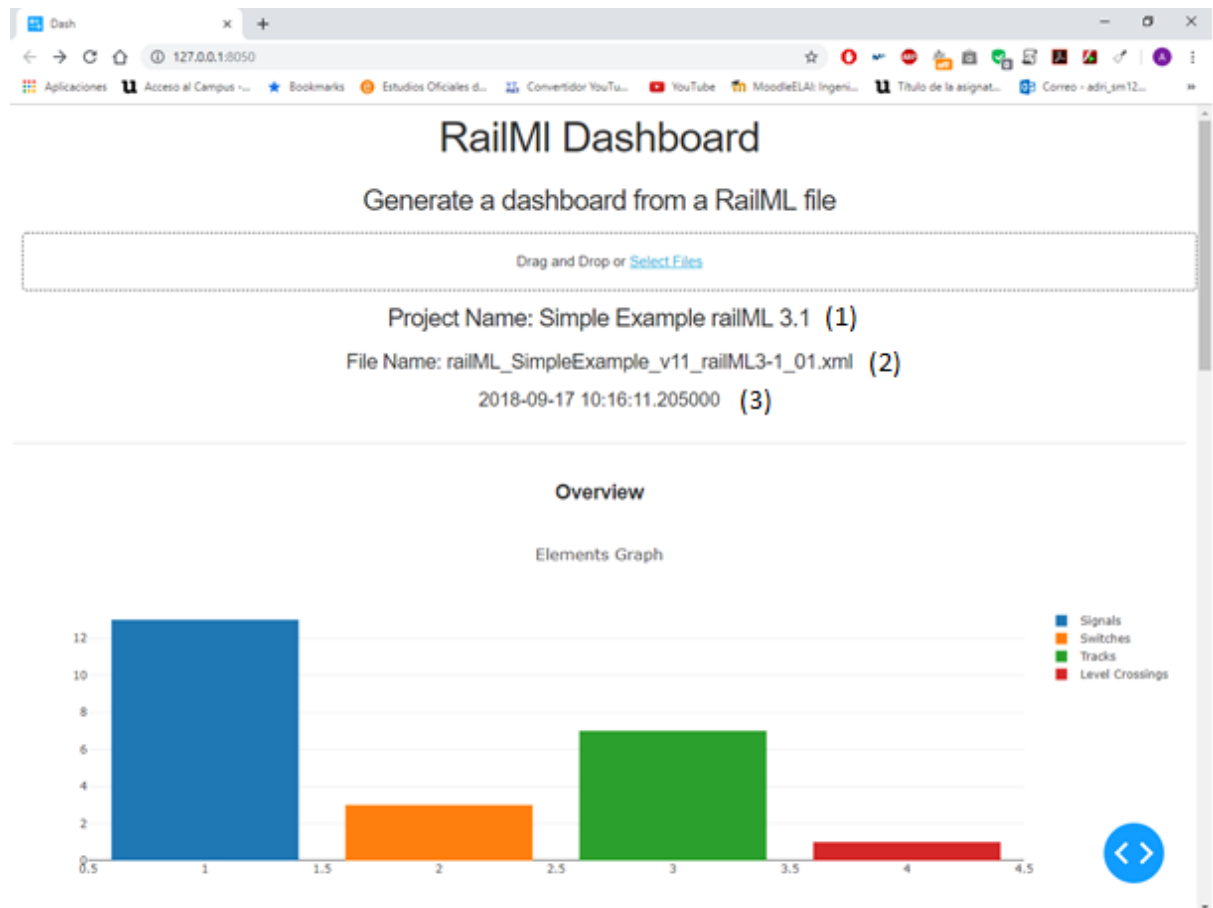


Figura 11 RailML Dashboard

Se despliega una serie de información, gráficos y tablas que analizaremos a continuación.

La Figura 11 corresponde en el código con el comentario "[Overview](#)".

Lo primero que el usuario se encuentra es el nombre del proyecto (1). Esta información se encuentra alojada en el fichero "RailML" bajo la etiqueta "metadata/title". Para crear este elemento se ha utilizado una etiqueta de cabecera H4.

La siguiente línea hace referencia al nombre del fichero (2) con una cabecera H5. A continuación observaremos la fecha y hora (3) en que fue creado el documento, para tener una referencia de la antigüedad del fichero. Este elemento es una cabecera H6.

Seguidamente encontramos un título "Overview", cabecera H5, que indica que a continuación habrá un resumen del contenido.

El primer elemento gráfico que aparece es un gráfico de barras llamado "Elements Graph". Es un resumen de los elementos que nos encontraremos en el "dashboard" y el número que hay de cada elemento. Este gráfico se consigue con un objeto de la librería "Dash" de tipo

“Graph”. En el que luego se indica el tipo de gráfico ('type': 'bar'), los datos que irán en cada eje, títulos, leyendas, etc.

El eje “y” se refiere a la cantidad de elementos que hay, mientras que cada barra del eje “x” es un tipo de elemento.

A continuación se describen los tipos de elementos que se muestran en el gráfico.

Una señal (“Signal”) es cualquier indicativo mostrado por los componentes integrados a lo largo del trazado ferroviario, los trenes o aquéllas que los agentes puedan recurrir con el propósito de notificar a los maquinistas órdenes definidas en la circulación de los trenes.²⁸

Llamamos desvío o aguja (“Switch”) al elemento de vía que posibilita la ramificación de una vía, permitiendo el paso de los movimientos de una vía a otra, cuyo eje se acuerda tangencialmente con el de la primera o formando un ángulo muy pequeño con él.

La aguja que permite la desviación se separa en tres secciones importantes: el cambio, una zona intermedia y el denominado cruzamiento, que es el más primordial para conseguir el correcto funcionamiento.²⁹

Un Paso a nivel (“Level Crossing”) es una intersección física entre la línea ferroviaria y una calzada o una senda a la misma altura de nivel, autorizando en ese determinado punto la circulación de vehículos y de personas sobre la vía férrea. Emplazamiento de seguridad sobre el corte entre la línea ferroviaria y una calzada o pasaje para establecer el movimiento de vehículos y personas por ese lugar, se puede completar con barreras, señalización vertical, luminosa y/o acústica.³⁰

Se designa vía férrea (“Track”) a la parte de la instalaciones ferroviarias, constituido por el conjunto de componentes que modelan el lugar por el cual se deslizan los trenes. Las vías férreas son la pieza primordial de la construcción ferroviaria y constan, fundamentalmente, de carriles sustentados sobre vigas que se colocan dentro de una capa de balasto. Se necesita para la construcción efectuar movimiento de suelos y obras de fábrica (puentes, alcantarillas, muros de contención, drenajes, etcétera).³¹

Además cuando el usuario pasa por encima de las barras con el ratón aparece un “pop up” indicando el número del elemento correspondiente, de igual manera que se ilustra en la siguiente imagen.

²⁸ Señal -Ferropedia. (2019).

²⁹ Aparatos de vía: los desvíos ferroviarios | MÁS QUE INGENIERÍA. (2019).

³⁰ Paso a nivel - Ferropedia. (2019).

³¹ Vía férrea. (2019).

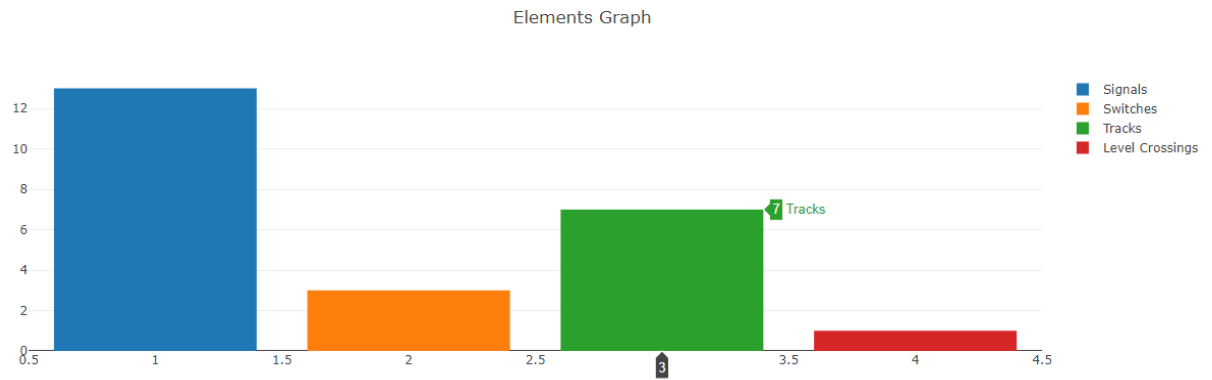


Figura 12 Pop Up en el gráfico de barras.

También se puede observar la leyenda con los distintos elementos y sus códigos de colores.

Cuando el usuario hace “scroll” con el ratón hacia abajo continuará visualizando el “dashboard”.

Lo siguiente que se encontrará es el apartado de “[Signals](#)”. El primer gráfico de este apartado se muestra a continuación.

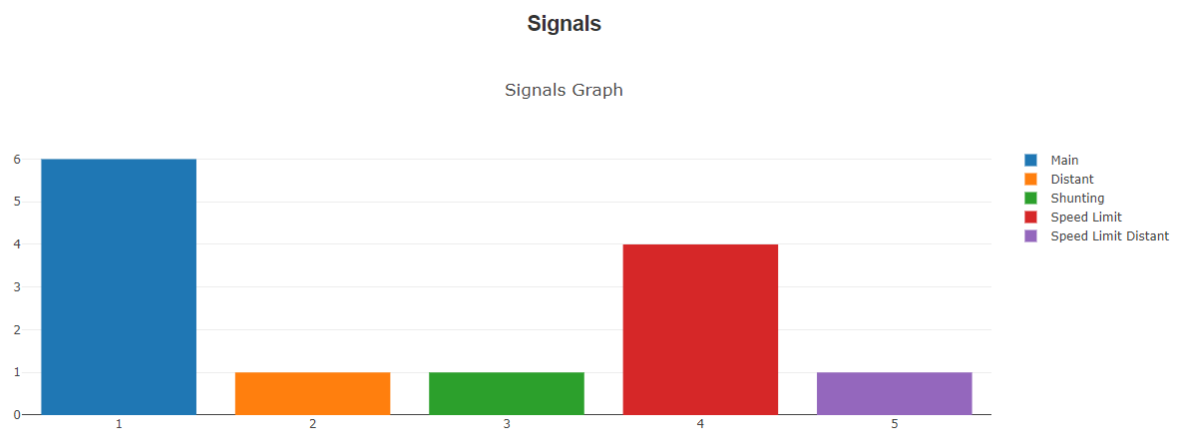


Figura 13 Signal Graph

Es un gráfico de barras que muestra los distintos tipos de señales que hay en el proyecto y el número de señales de cada tipo.

Las señales de tipo “Main” son las señales principales y las más usadas, hacen referencia a indicaciones de cualquier tipo.

Las señales de tipo “Distant” son señales colocadas a una distancia que permita una adecuada advertencia anticipada del ajuste de una señal de inicio en la cual el tren debe detenerse.³²

Las señales de tipo “Shunting” hacen referencia a señales que indican maniobras. No indican que la línea por delante esté despejada, pero que los movimientos pueden avanzar hasta donde la línea esté libre.

Las señales de “Speed Limit” y “Speed Limit Distant” son señales que indican restricciones de velocidad con la diferencia de que la segunda además también es de tipo “Distant”.

También tiene la funcionalidad de “pop up” como hemos visto en el gráfico de “Overview”.

El siguiente gráfico cuando el usuario sigue haciendo “scroll” es el siguiente:

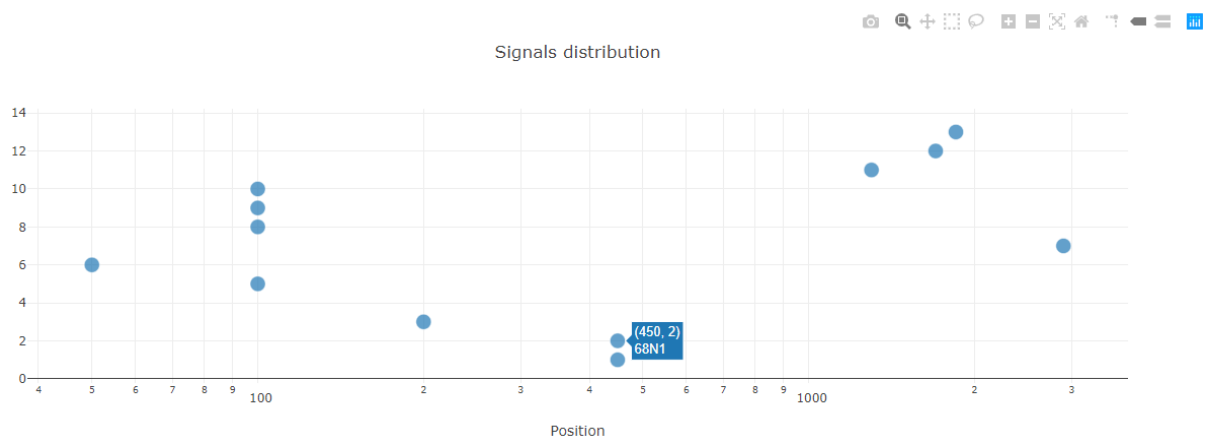


Figura 14 Signals distribution graph

El siguiente gráfico de puntos representa las señales a lo largo de la línea. De esta manera el usuario puede hacerse una idea de la distribución de las señales. El eje “x” representa la posición de las señales y en el eje “y” distribuye las señales por cada unidad.

Este gráfico también es un objeto de tipo “Graph” pero de tipo “Scatter”.

El “pop up” en este gráfico podemos ver que ilustra la posición en la que se encuentra la señal y el nombre de dicha señal.

³² Definition of DISTANT SIGNAL. (2019).

A continuación del gráfico de puntos ya descrito nos encontramos una tabla con toda la información referente a las señales:

ID	Type	Name	Position
sig01	main	68N2	450.0
sig02	main	68N1	450.0
sig03	main	68F	200.0
sig04	main	69A	0.0
sig05	main	69P2	100.0
sig06	main	69P1	50.0
sig07	distant	69Va	2900.0
sig08	shunting	69W04Y	100.0
sig09	speed_limit	-	100.0
sig10	speed_limit	-	100.0
sig11	speed_limit_distant	-	1300.0
sig12	speed_limit	-	1700.0
sig13	speed_limit	-	1850.0

Figura 15 Tabla de señales

Esta tabla es otro objeto de la librería “Dash” que se llama “DataTable”, donde se identifican las cabeceras, el grosor de las líneas, etc.

El siguiente elemento después de las señales son las [agujas](#). Lo primero respecto a las agujas que nos encontramos es una tabla resumen.

ID	Name	Default Course	Type	Position	Left Branch Speed	Right Branch Speed
swi01	68W02	right	ordinarySwitch	0.0	60	80
swi02	69W03	left	ordinarySwitch	200.0	80	40
swi03	69W04	left	ordinarySwitch	0.0	60	40

Figura 16 Tabla agujas

En esta tabla se encuentran parámetros tales como “Default Course” que indica la rama por defecto o “Left Branch Speed” y “Right Branch Speed” que indica las limitaciones de velocidad en la rama de la izquierda y en la rama de la derecha.

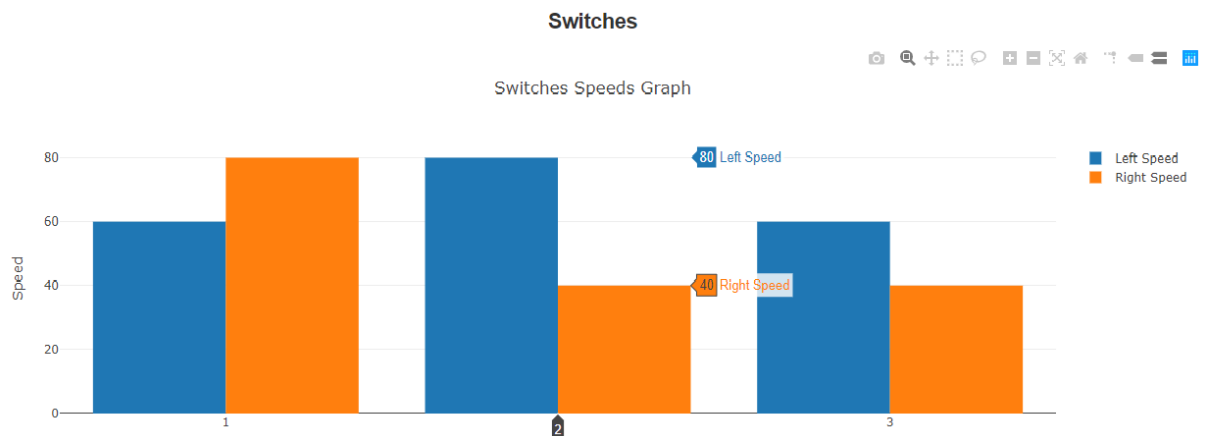


Figura 17 Limitaciones de velocidad de las agujas

Con respecto a las limitaciones de velocidad, mencionadas anteriormente, tenemos la Figura 17 que representa las limitaciones de cada aguja comparando la rama de la izquierda con la rama de la derecha. Se puede observar que siempre la rama por defecto tiene un valor mayor que la rama desviada.

En este gráfico el “pop up” que aparece es los limites tanto para la rama de la izquierda como el de la rama de la derecha.

De igual manera que para las agujas, el siguiente gráfico es un gráfico de puntos que representa la distribución a lo largo de la línea ferroviaria de las agujas.

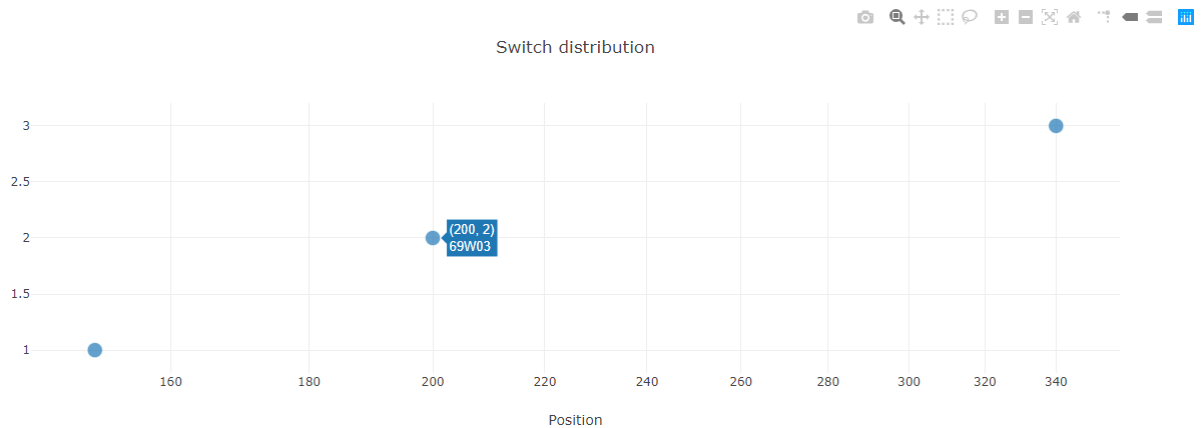


Figura 18 Distribución de las agujas

El siguiente elemento ferroviario que aparece en la página web son las [vías](#).

ID	Type	Direction	From Position	To Position	Length
trc01	mainTrack	both	0.0	500.0	500.0
trc02	secondaryTrack	both	0.0	500.0	500.0
trc03	mainTrack	both	0.0	200.0	4000.0
trc04	mainTrack	both	0.0	500.0	500.0
trc05	secondaryTrack	both	0.0	450.0	450.0
trc06	sidingTrack	both	0.0	200.0	200.0
trc07	connectingTrack	both	0.0	50.0	50.0

Figura 19. Tabla de vías

Además de los campos “ID” y “Type”, los cuales son un identificador de la vía y el tipo de la misma respectivamente se han añadido “Direction” que indica si la vía está diseñada para que el tren vaya en una u otra dirección o ambas. Los atributos “From Position” y “To Position” indican los kilometrajes de las vías desde donde empieza hasta donde acaba. El último atributo “Length” indica el tamaño de la vía.

Resulta curioso observar que una de las vías, la “trc03” no coincide el tamaño con la resta entre los kilómetros de inicio y fin. Esto es debido a que entre el kilómetro de inicio y el de fin hay un salto kilométrico.

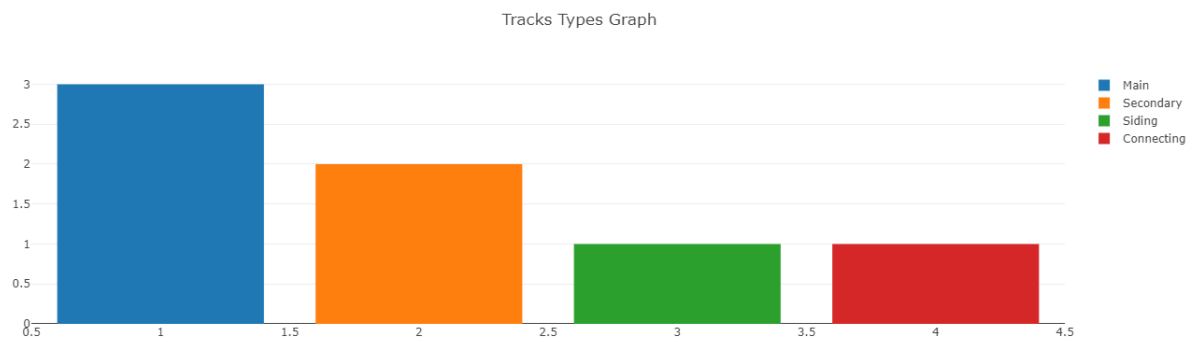


Figura 20 Tipos de vías

Este último gráfico es para ver de un vistazo los tipos de vías y cuantas vías de cada tipo existen en el proyecto. El eje “y” cuenta el número de vías, mientras que cada barra del eje “x” representa cada tipo de vía.

El último gráfico de las vías es referente al tamaño de las mismas.

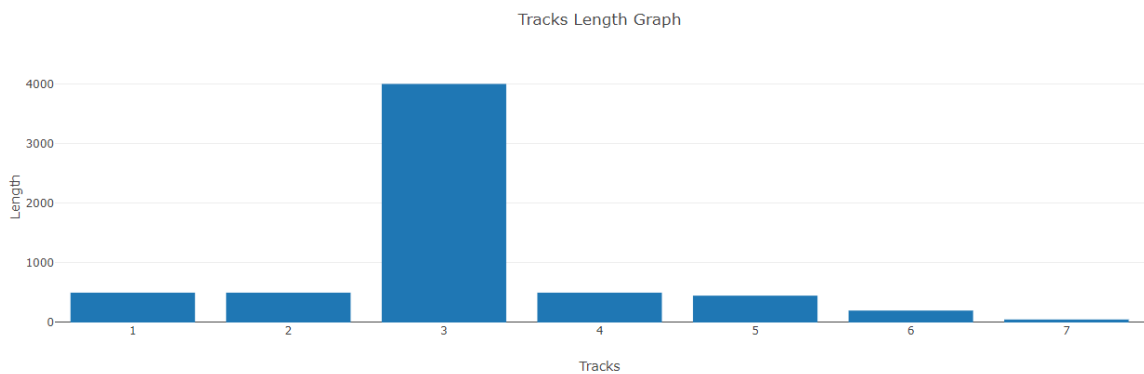


Figura 21 Tamaño de las vías

El último elemento que aparece en el “dashboard” es el [paso a nivel](#).

ID	Name	Activation	Direction	Position
1cr01	LX Arnau Cstadt	infrastructureAutomatic	both	1800.0
1cr02	LX Polatli Cstadt	infrastructureAutomatic	both	3500.0

Figura 22 Tabla paso a nivel

Del que también tenemos un gráfico que representa su distribución a lo largo de la línea.

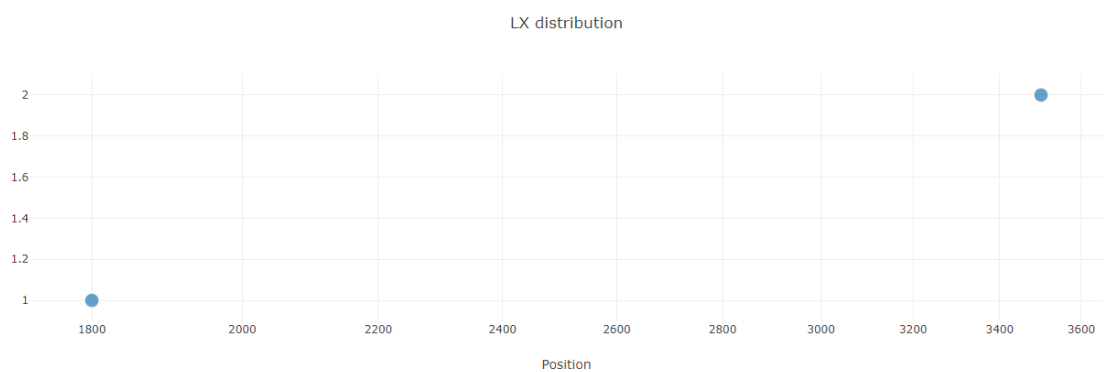


Figura 23 Distribución de los pasos a nivel

Todas las gráficas del “dashboard” pueden descargarse, hacer zoom, etcétera, como podemos ver en la siguiente imagen.



Figura 24 Leyenda Plotly

Es una funcionalidad que ofrece la librería “Plotly” para poder inspeccionar mejor las gráficas.

Despliegue de la herramienta

El código desarrollado para conseguir la aplicación web se ha desplegado en un servidor web gratuito llamado “Heroku”. “Heroku” es una plataforma en la nube como servicio. Esta plataforma ofrece de una manera sencilla el despliegue de aplicaciones web sin tener que centrarnos en la infraestructura.³³

Tan sencillo como tener un repositorio local de “Git” con los ficheros necesarios para que funcione la aplicación, después volcar el repositorio en el servidor de “Heroku” y ya tendríamos la aplicación web en un dominio.

En este caso la dirección web para acceder a la herramienta es:

<https://railmldashboard.herokuapp.com/>

³³ ¿Qué es Heroku? y mi experiencia con su servicio. (2019).

Control de configuración

La gestión de configuraciones del software (GCS) consiste en una colección de actividades realizadas para administrar las modificaciones a lo largo del ciclo de vida. La GCS es una labor para asegurar la calidad del software que se emplea en todas y cada una de las etapas del proceso de ingeniería software.³⁴

Existen muchas definiciones acerca de esta especialidad. Cualquier experto que ha intentado formular una definición ha contribuido con distintos puntos de vista, así como nuevas labores a contemplar. A menudo, chocan tantos puntos de vista, sobre todo en cuanto a nombres o preferencias entre quehaceres a ejecutar, no obstante, concuerdan siempre en la relevancia de esta disciplina.³⁵ Roger S. Pressman la definió de la siguiente manera:

“El arte de coordinar el desarrollo de software para minimizar la confusión se denomina gestión de configuración. La gestión de configuración es el arte de identificar, organizar y controlar las modificaciones que sufre el software que construye un equipo de programación. La meta es maximizar la productividad minimizando los errores”.

Otros autores la han definido como:

“La GCS es una disciplina de la ingeniería de software que comprende herramientas y técnicas (procesos o metodología) que una compañía utiliza para manejar los cambios en sus software activos”.

En 1972 apareció SCCS (Source Code Control System), fue uno de los primeros programas de control de versiones. Fue desarrollado por IBM y después fue adaptado por UNIX.³⁶

A SCCS le suceden programas como RCS (Revision Control System, 1982), PVCS (Polytron Version Control System, 1958), CVS (Concurrent Version System, 1990), Clercase (1992), VSS (Visual Source Safe, 1995), Perforce (1995), Accurev (1999), Subversion (2000), Bitkeeper (2000) y finalmente Git (2005).

Git es un sistema de control de versiones libre y “open source” diseñado para manejar todos los proyectos, desde pequeños a grandes proyectos con rapidez y eficiencia.

³⁴ Anónimo (2019).

³⁵ Morejón, M. (2019). ¿Qué es la Gestión de Configuración?.

³⁶ Git. (2019).

Para desarrollar este trabajo se ha utilizado Git como sistema de control de versiones.

Además se utilizará GitHub para alojar el proyecto. GitHub es un sitio web y un servicio en la nube que ayuda a los desarrolladores a almacenar y administrar su código, al igual que llevar un registro y control de cualquier cambio sobre este código.³⁷

Hay algunos conceptos de Git que son necesarios para entender los siguientes párrafos. A continuación se definen algunos de los conceptos:

- **Repositorio:** Un repositorio, depósito o archivo es un lugar centralizado donde se acumula y conserva información digital, normalmente se almacenan bases de datos o archivos informáticos.³⁸
- **Commit:** Un "commit" es el ejercicio de archivar o subir los archivos modificados o nuevos a tu repositorio remoto (una renovación de los cambios) así mismo puede hacerse localmente (depende donde hayas creado tu repositorio).³⁹
- **Rama:** Las ramas son las vías por las que el progreso de un software puede moverse, suele ocurrir ciertamente para solucionar problemas o implementar nuevas funcionalidades. En la práctica posibilitan que el proyecto sea capaz tener varias etapas y que los desarrolladores puedan moverse de uno a otro de un modo ágil.⁴⁰

El link del repositorio GitHub para acceder a su contenido es el siguiente:

<https://github.com/adriSM07/railMLdashboard>

³⁷ ¿Qué es y para qué sirve Github? | Deusto Formación. (2019).

³⁸ "¿Qué es un repositorio? - Base de Conocimientos - ICTEA", 2019

³⁹ Commits - Administrar tu repositorio. (2019).

⁴⁰ Alvarez, M. (2019). Trabajar con ramas en Git: git branch.

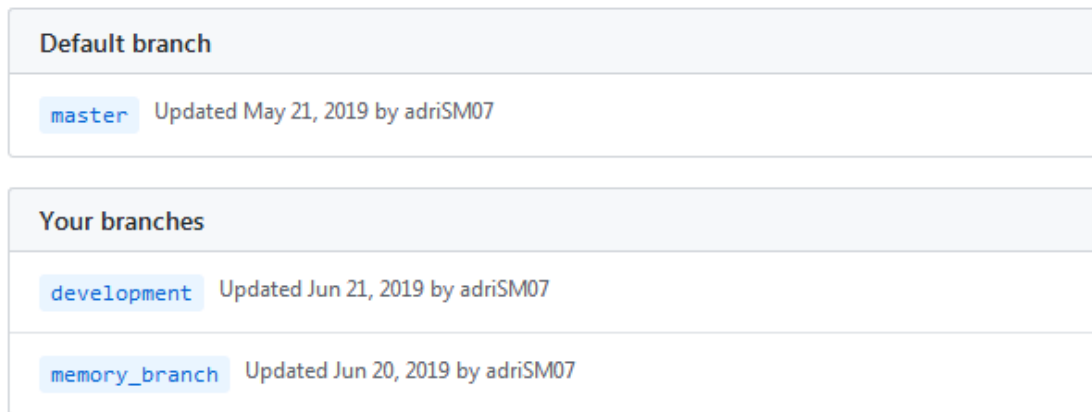


Figura 25. Ramas del proyecto GitHub

Como muestra la imagen anterior en el repositorio se encontrarán tres ramas. Una rama “master” en la que se encontrarán versiones finales etiquetadas de la aplicación. Una rama “memory-branch” en la que se han ido haciendo “commits” del documento de la memoria. Y una última rama “development” en la que se ha ido desarrollando la aplicación.

La rama “master” tendrá una versión final del “software” etiquetada cuando de la rama “development” se haga un “merge” a la rama “master” y además se aplique una etiqueta de versión.

Un “merge” es una unión de dos o más historias de desarrollo juntas. Incorpora cambios de los “commits” nombrados (desde el momento en que sus historias se desviaron de la rama actual) a la rama actual.⁴¹

Una etiqueta en control de configuración sirve para especificar puntos de la historia como importantes. Principalmente se usa el etiquetado para marcar puntos donde se ha lanzado alguna versión.⁴²

⁴¹ Git - git-merge Documentation. (2019).

⁴² Git - Etiquetado. (2019).

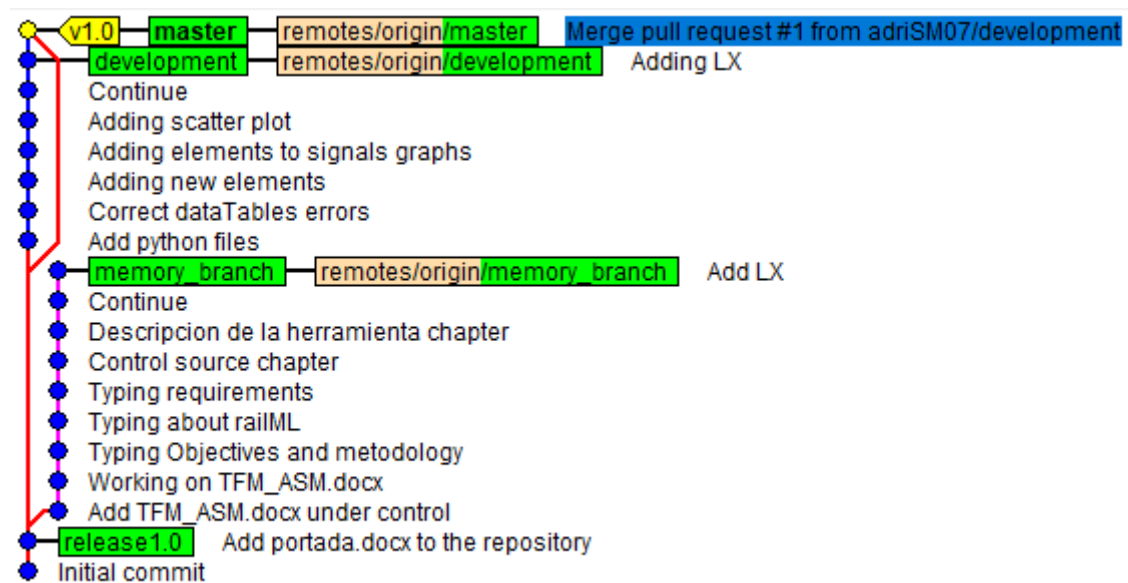


Figura 26. Ramas del repositorio

Como podemos ver en la imagen anterior se ha hecho un “merge” de la rama “development” a la rama “master”, debido a que ya se ha formalizado una versión y además esta versión ha sido etiquetada como “v1.0”.

Por ejemplo si quisiéramos añadir una nueva funcionalidad, se podría seguir desarrollando en la rama “development” y una vez finalizada la funcionalidad se volvería a hacer un “merge” a la rama “master” y etiquetarlo como “v1.1”.

Evaluación

A continuación se detallarán una serie de test de usabilidad y aplicabilidad para definir si la herramienta funciona correctamente y documentar el resultado esperado.

- **Test 1**

Arrancar la página web y observar que la página se abre correctamente, además de que aparecen los títulos correctos y esperados.

Resultado: OK.

- **Test 2**

Comprobar que el enlace a que existe al final de la página hacia la página oficial de “railML” es el correcto.

Resultado: OK.

- **Test 3**

Comprobar que cuando se pulsa “Select file” se abre un diálogo en el que se puede elegir un fichero.

Resultado: OK.

- **Test 4**

Comprobar que al seleccionar un fichero correcto se despliega el “dashboard” esperado.

Resultado: OK.

- **Test 5**

Comprobar que al soltar un fichero en la zona que pone “Drag and Drop” se genera el “dashboard” esperado.

Resultado: OK.

- **Test 6**

Comprobar que si se selecciona o se suelta un fichero que no es “railML” aparece un mensaje de error.

Resultado: OK.

- **Test 7**

Comprobar que si el fichero “railML” no está formalizado, es decir correctamente definido, aparece un mensaje de error.

Resultado: OK.

- **Test 8**

Comprobar que lo primero en aparecer en el “dashboard” es el nombre del proyecto, el nombre del fichero y la fecha de creación del fichero.

Resultado: OK.

- **Test 9**

Comprobar que lo siguiente en aparecer es un gráfico de barras con los diferentes elementos ferroviarios justo debajo del título “Overview”.

Resultado: OK.

- **Test 10**

Seguidamente deberá estar el título “Signals” y los gráficos relacionados con las señales. Una tabla, un gráfico de barras, indicando los distintos tipos de señales y el número de cada una y un gráfico de dispersión indicando la posición de cada señal.

Resultado: OK.

- **Test 11**

Después de las señales, nos deberemos encontrar con el título de “Point switches” haciendo referencia a las agujas. Este elemento tiene una tabla con información de las agujas del proyecto, además de un gráfico de barras que refleja las restricciones de velocidad tanto por una rama como por otra de la aguja y un gráfico de dispersión que refleja la posición de cada aguja.

Resultado: OK.

- **Test 12**

El siguiente título que debe aparecer es el de “Tracks”. Este apartado tiene una tabla y dos gráficos de barras. El primero indica los tipos de vías y el número de cada tipo, y el segundo gráfico indica el largo de cada vía.

Resultado: OK.

- **Test 13**

El último título que tiene que aparecer es el de “Level Crossing”. Este apartado deberá tener una tabla y un gráfico de dispersión indicando la ubicación de cada paso a nivel.

Resultado: OK.

- **Test 14**

Comprobar que una vez que se ha desplegado el “dashboard” el link referenciando a la página oficial de “railML” se sigue encontrando al final de la página, después de todas las visualizaciones.

Resultado: OK.

- **Test 15**

Comprobar que se puede interactuar con todas las visualizaciones (gráficos), seleccionar barras, puntos; zoom; etcétera.

Resultado: OK.

- **Test 16**

Comprobar que todas las visualizaciones pueden volver a su estado inicial después de haber seleccionado una barra o un punto o haber hecho zoom.

Resultado: OK.

- **Test 17**

Comprobar que todas las visualizaciones pueden ser guardadas.

Resultado: OK.

- **Test 18**

Comprobar que una vez desplegado el “dashboard” siguen apareciendo los botones de “Drag and Drop or Select File”.

Resultado: OK.

- **Test 19**

Comprobar que una vez desplegado el “dashboard” la página web sigue permitiendo seleccionar un nuevo archivo.

Resultado: OK.

- **Test 20**

Comprobar que una vez desplegado el “dashboard” y seccionado un nuevo archivo la página web genera un nuevo “dashboard” correspondiente a los nuevos datos introducidos.

Resultado: OK.

Conclusiones

En este Trabajo de Fin de Máster se ha tratado de abordar el problema de visualizar, entender y sintetizar un fichero de datos de gran tamaño relacionado con el ámbito ferroviario.

Para ello se ha realizado una página web con Python y la ayuda de la librería “dash”, la cual es muy útil y sencilla para generar “dashboard” en entornos web. La página web pide al usuario un fichero de datos específico del sector ferroviario y automáticamente se genera un “dashboard” con variedad de gráficas y tablas.

El hecho de ser una aplicación web tiene como ventaja que se puede utilizar la herramienta en cualquier lugar y desde cualquier dispositivo sin necesidad de tener que instalarlo, simplemente con conexión a internet.

Cada fichero de datos corresponde con un proyecto diferente, de tal manera que cada “dashboard” hará referencia a dicho proyecto.

Además el fichero de datos elegido es un “railML”, como ya se ha explicado anteriormente. Esto significa que si en algún momento la iniciativa que propone “railML” prospera y las empresas empiezan a utilizar todavía más las estructuras “railML” para los datos de sus sistemas y proyectos, esta página web podría ser usada por cualquier empresa y para cualquier proyecto que desarrollen

El usuario será capaz de obtener del “dashboard” una síntesis de los elementos que conforman el proyecto, siendo capaz a su vez de conocer el alcance del proyecto, que tipos de elementos se manejan, donde están situados, etcétera.

Esto incluye una ventaja competitiva con respecto a las demás herramientas de visualización y análisis de datos ferroviarios, debido a su simplicidad y fácil manejo. Más aún ya que no se necesitan conocimientos ferroviarios para poder trabajar con la herramienta. Adicionalmente las herramientas disponibles suelen centrarse en los enclavamientos y no en los elementos que lo conforman. Un enclavamiento es una infraestructura de seguridad la cual se usa para manejar los elementos del proyecto.

Así mismo el usuario puede interactuar con la página web y sus gráficos, haciendo zoom o seleccionando varias barras o puntos de los gráficos. Incluso tiene la opción de exportar dichos gráficos.

Analizando la página web, cada gráfica y cada tabla se puede concluir que se han conseguido los objetivos iniciales.

Para decidir qué información se mostraría en la página web y cómo se iba a mostrar, se ha tenido en cuenta el público al que está dirigido. Este público objetivo se ha centrado en mandos intermedios con suficiente conocimiento del sector ferroviario pero que no tienen necesidad ni tiempo de bucear en los datos para extraer información de cada proyecto que manejan o en el que están envueltos.

Gracias a esta aplicación web, fácil e intuitiva de usar, dichos mandos intermedios están a un “click” de obtener todo ese conocimiento residente en los datos, el cual es imposible de obtener con tan solo un vistazo al fichero de datos.

Líneas futuras

Este Trabajo de Fin de Máster tiene numerosas funcionales que se pueden seguir implementando para mejorar o completar la función u objetivos de los que inicialmente parte este trabajo.

A continuación, se mencionarán algunas de las posibilidades para continuar con el desarrollo de este trabajo y obtener así valor añadido adicional a la herramienta desarrollada.

Una primera tarea que se podría realizar es la de compactar las gráficas y las tablas, agrupándolas por cada elemento (señales, agujas, pasos a nivel o vías). De esta manera se reducirá el espacio empleado en la página web, consiguiendo una mayor disponibilidad de la información en una sola página.

Otra tarea sencilla sería la de completar la información de los elementos ya visualizados. Desde nuevos gráficos, nuevos atributos en las tablas, mejorar los gráficos ya existentes. De igual manera que se podrían añadir elementos adicionales tales como (limitaciones de velocidad, secciones, estaciones, plataformas, etcétera)

Otra funcionalidad muy interesante podría ser la de añadir a la página web en un nuevo “dashboard” aparte, como un informe o un botón extra; información acerca del enclavamiento y sobre como los elementos se relacionan entre sí. Con esta información se puede tener un mapa esquemático de la línea ferroviaria el cual se podría visualizar mediante líneas y uniones entre elementos.

Si se consiguiera procesar todo lo anterior descrito respecto al enclavamiento, se podrían añadir funcionalidades de verificación de los datos. Esto no implica comprobar si un dato concreto está en su formato adecuado, implica realizar comprobaciones para ver que se cumplen reglas de ingeniería definidas para cada proyecto. Comprobaciones como por ejemplo que después de una vía ferroviaria de tipo principal no puede ser precedida de una vía de tipo maniobra. Otro ejemplo podría ser que siempre que nos encontremos con una señal del tipo avanzada, el siguiente elemento que debería aparecer es una señal de entrada.

Este tipo de comprobaciones pueden llegar a ser muy útiles para departamentos que se dedican a la validación de sistemas de enclavamientos y validación de los datos de sistema que van asociados a dicho enclavamiento.

Estas son algunas de las funcionalidades que podrían mejorar y completar el trabajo realizado, pero debido al gran volumen de información que reside en los datos, las posibles mejoras o nuevas tareas que se podrían realizar en la herramienta ascienden a infinitas.

Referencias y enlaces

- 5 estrellas de Datos Abiertos. (2019). Disponible en <https://5stardata.info/es/4>
- Alvarez, M. (2019). Trabajar con ramas en Git: git branch. Disponible en <https://desarrolloweb.com/articulos/trabajar-ramas-git.html>
- Anónimo (2019). Disponible en <https://w3.ual.es/~rguirado/posi/Tema5-Apartado5.pdf>
- Aparatos de vía: los desvíos ferroviarios | MÁS QUE INGENIERÍA. (2019). Disponible en <https://masqueingenieria.com/blog/aparatos-de-via-los-desvios/>
- Background - railML.org (EN). (2019). Disponible en <https://www.railml.org/en/introduction/background.html>
- Commits - Administrar tu repositorio. (2019). Disponible en <https://codigofacilito.com/articulos/commits-administrar-tu-repositorio>
- Cómo abrir un archivo XML. (2019). Disponible en <https://es.wikihow.com/abrir-un-archivo-XML>
- ¿Cómo escribir un buen documento de especificación de requisitos de software? - Medium. (2019). Disponible en <https://medium.com/grupo-carricay/c%C3%B3mo-escribir-un-buen-documento-de-especificaci%C3%B3n-de-requisitos-de-software-fd8bb3b5a39a>
- Dash User Guide and Documentation - Dash by Plotly. (2019). Disponible en <https://dash.plot.ly/dash-core-components/upload>
- Definition of DISTANT SIGNAL. (2019). Disponible en <https://www.merriam-webster.com/dictionary/distant%20signal>
- Git. (2019). Disponible en <https://git-scm.com/>
- Git - Etiquetado. (2019). Disponible en <https://git-scm.com/book/es/v2/Fundamentos-de-Git-Etiquetado>

- Git - git-merge Documentation. (2019). Disponible en <https://git-scm.com/docs/git-merge>
- Home - railML.org (EN). (2019). Disponible en <https://www.railml.org>
- Introducción a XML. (2019). Disponible en https://developer.mozilla.org/es/docs/Web/XML/Introducci%C3%B3n_a_XML
- Morejón, M. (2019). ¿Qué es la Gestión de Configuración?. Disponible en <http://mmorejon.github.io/blog/que-es-la-gestion-de-configuracion/>
- OpenTrack Railway Technology - Railway Simulation. (2019). Disponible en http://www.opentrack.ch/opentrack/opentrack_s/opentrack_s.html
- Ortiz, D. (2019). ¿Qué es un dashboard?. Disponible en <https://www.cyberclick.es/numerical-blog/que-es-un-dashboard>
- Para qué sirve un 'dashboard'. (2019). Disponible en <http://www.expansion.com/economia-digital/protagonistas/2016/11/12/5824c400e5fdea752d8b45d3.html>
- Paso a nivel - Ferropedia. (2019). Disponible en http://ferropedia.es/mediawiki/index.php/Paso_a_nivel
- ¿Qué es Heroku? y mi experiencia con su servicio. (2019). Disponible en <https://forobeta.com/temas/que-es-heroku-y-mi-experiencia-con-su-servicio.485051/>
- ¿Qué es y para qué sirve Github? | Deusto Formación. (2019). Disponible en <https://www.deustoformacion.com/blog/programacion-diseno-web/que-es-para-que-sirve-github>
- ¿Qué es un repositorio? - Base de Conocimientos - ICTEA. (2019). Disponible en <http://www.ictea.com/cs/index.php?rp=/knowledgebase/3481/iQue-es-un-repositorio.html>

- ¿Qué es y para qué sirve Github? | Deusto Formación. (2019). Disponible en <https://www.deustoformacion.com/blog/programacion-diseno-web/que-es-para-que-sirve-github>
- ¿Qué son las Cabeceras HTML? - Ryte Digital Marketing Wiki. (2019). Disponible en https://es.ryte.com/wiki/Cabeceras_HTML
- railOscope - Features. (2019). Disponible en <https://railoscope.com/docs/features>
- railOscope - railML.org (EN). (2019). Retrieved 23 July 2019, from <https://www.railml.org/en/introduction/applications/detail/railoscope.html>
- railVIVID - railML.org (EN). (2019). Disponible en <https://www.railml.org/en/user/railvivid.html>
- Requisitos de Software - EcuRed. (2019). Disponible en https://www.ecured.cu/Requisitos_de_Software
- Señal - Ferropedia. (2019). Disponible en <http://www.ferropedia.es/wiki/Se%C3%B1al>
- Vía férrea. (2019). Retrieved 16 July 2019, from https://ferrocarriles.fandom.com/wiki/V%C3%ADa_f%C3%A9rrea
- World Wide Web Consortium (W3C). (2019). Disponible en <https://www.w3.org/>

Anexos

Código de la aplicación web

```

1. import base64
2. import datetime
3. import io
4.
5. import dash
6. from dash.dependencies import Input, Output, State
7. import dash_core_components as dcc
8. import dash_html_components as html
9. import dash_table
10. import xml.etree.cElementTree as et
11. import plotly.graph_objs as go
12.
13. import pandas as pd
14. from pandas import DataFrame
15.
16. external_stylesheets = ['https://codepen.io/chriddyp/pen/bWLwgP.css']
17.
18. app = dash.Dash(__name__, external_stylesheets=external_stylesheets)
19. server = app.server
20.
21. #Ref 1.0
22. app.layout = html.Div([
23.     #Etiqueta H1
24.     html.H1(children='RailML Dashboard', style={'textAlign': 'center'}),
25.     #Etiqueta H3
26.     html.H3(children='Generate a dashboard from a RailML file', style={'textAlign': 'center'}),
27.     #Upload --> pide un archivo
28.     dcc.Upload(
29.         id='upload-data',
30.         children=html.Div([
31.             'Drag and Drop or ',
32.             html.A('Select Files')
33.         ]),
34.         style={
35.             'width': '100%',
36.             'height': '60px',
37.             'lineHeight': '60px',
38.             'borderWidth': '1px',
39.             'borderStyle': 'dashed',
40.             'borderRadius': '5px',
41.             'textAlign': 'center',
42.             'margin': '10px'
43.         },
44.         # Allow multiple files to be uploaded
45.         multiple=False
46.     ),
47.     # 'output-data-upload' --> donde se desplegará el dashboard
48.     html.Div(id='output-data-upload'),
49.     #pagina web de railML
50.     html.A(href='http://www.railml.org/en/', children='www.railml.org', style={'text-align': 'center'}),
51.
52. ])
53.
54. #Ref 1.1
55. #funcion para parsear el fichero
56.
57. def parse_contents(contents, filename, date):
58.     # content_type, content_string = contents.split(',')

```



```

59. #
60. # decoded = base64.b64decode(content_string)
61.
62. try:
63.     if 'xml' in filename:
64.         # Assume that the user uploaded a XML file
65.         # tree_2 = et.parse(io.StringIO(decoded.decode('utf-8')))
66.         tree_2 = et.parse(filename)
67.         root_2 = tree_2.getroot()
68.         projectName = "
69.         num = 0
70.         id_signal = []
71.         type_signal = []
72.         name_signal = []
73.         pos_signal = []
74.         pos_signal_Y = []
75.         pos_y = 0
76.         dic_signal = {}
77.         id_switch = []
78.         defaultCourse_switch = []
79.         type_switch = []
80.         name_switch = []
81.         pos_switch = []
82.         num_switch = []
83.         left_SpeedBranch = []
84.         right_SpeedBranch = []
85.         id_track = []
86.         type_track = []
87.         fromPos_tracks = []
88.         toPos_tracks = []
89.         length_tracks = []
90.         applicationDirection_track = []
91.         id_LX = []
92.         activation_LX = []
93.         name_LX = []
94.         direction_LX = []
95.         posLX = []
96.
97. #seleccion de datos del fichero que pasa el usuario
98.     for child_2 in root_2:
99.         if '{http://www.railml.org/schemas/3.1}infrastructure' == child_2.tag:
100.             for secondChild_2 in child_2:
101.                 if '{http://www.railml.org/schemas/3.1}functionalInfrastructure' == secondChild_2.tag:
102.                     for thirdChild_2 in secondChild_2:
103.                         if '{http://www.railml.org/schemas/3.1}signals' == thirdChild_2.tag:
104.                             for fourthChild_2 in thirdChild_2:
105.                                 id_signal.append(fourthChild_2.get('id'))
106.                                 type_signal.append(fourthChild_2.get('type'))
107.                                 for sixChild in fourthChild_2:
108.                                     if '{http://www.railml.org/schemas/3.1}name' == sixChild.tag:
109.                                         dic_signal.update({fourthChild_2.get('id'): sixChild.get('name')})
110.                                         name_signal.append(sixChild.get('name'))
111.                                     elif '{http://www.railml.org/schemas/3.1}spotLocation' != sixChild.tag and
112.                                         '{http://www.railml.org/schemas/3.1}speedSignal' != sixChild.tag and
113.                                         '{http://www.railml.org/schemas/3.1}etcsSignal' != sixChild.tag:
114.                                         name_signal.append("-")
115.                                         num = num + 1
116.                                         if len(name_signal) != num:
117.                                             name_signal.append("-")
118.                                         for sevenChild in fourthChild_2:
119.                                             if '{http://www.railml.org/schemas/3.1}spotLocation' == sevenChild.tag:
120.                                                 pos_signal.append(sevenChild.get('pos'))
121.                                     elif '{http://www.railml.org/schemas/3.1}switches' == thirdChild_2.tag:
122.                                         for fourthChild_3 in thirdChild_2:
123.                                             id_switch.append(fourthChild_3.get('id'))
124.                                             defaultCourse_switch.append(fourthChild_3.get('defaultCourse'))
125.                                             type_switch.append(fourthChild_3.get('type'))

```

```

124.         for eightChild in fourthChild_3:
125.             if '{http://www.railml.org/schemas/3.1}name' == eightChild.tag:
126.                 name_switch.append((eightChild.get('name')))
127.             elif '{http://www.railml.org/schemas/3.1}spotLocation' == eightChild.tag:
128.                 pos_switch.append(eightChild.get('pos'))
129.             elif '{http://www.railml.org/schemas/3.1}leftBranch' == eightChild.tag:
130.                 left_SpeedBranch.append(eightChild.get('speedBranching'))
131.             elif '{http://www.railml.org/schemas/3.1}rightBranch' == eightChild.tag:
132.                 right_SpeedBranch.append(eightChild.get('speedBranching'))
133.         elif '{http://www.railml.org/schemas/3.1}tracks' == thirdChild_2.tag:
134.             for fourthChild_4 in thirdChild_2:
135.                 id_track.append(fourthChild_4.get('id'))
136.                 type_track.append(fourthChild_4.get('type'))
137.             for nineChild in fourthChild_4:
138.                 if '{http://www.railml.org/schemas/3.1}linearLocation' == nineChild.tag:
139.                     applicationDirection_track.append(nineChild.get('applicationDirection'))
140.                 for tenChild in nineChild:
141.                     fromPos_tracks.append(tenChild.get('fromPos'))
142.                     toPos_tracks.append(tenChild.get('toPos'))
143.                 elif '{http://www.railml.org/schemas/3.1}length' == nineChild.tag:
144.                     length_tracks.append(nineChild.get('value'))
145.         elif '{http://www.railml.org/schemas/3.1}levelCrossings' == thirdChild_2.tag:
146.             for fourthChild_5 in thirdChild_2:
147.                 id_LX.append(fourthChild_5.get('id'))
148.                 activation_LX.append(fourthChild_5.get('activation'))
149.             for fithChild in fourthChild_5:
150.                 if '{http://www.railml.org/schemas/3.1}name' == fithChild.tag:
151.                     name_LX.append(fithChild.get('name'))
152.                 elif '{http://www.railml.org/schemas/3.1}spotLocation' == fithChild.tag:
153.                     direction_LX.append(fithChild.get('applicationDirection'))
154.                     posLX.append(fithChild.get('pos'))
155.         elif '{http://www.railml.org/schemas/3.1}metadata' == child_2.tag:
156.             for secondChild_3 in child_2:
157.                 if '{http://purl.org/dc/elements/1.1}/title' == secondChild_3.tag:
158.                     projectName = secondChild_3.text
159.
160.     # SIGNALS
161.     signals_2 = {"ID": id_signal, "Type": type_signal, "Name": name_signal, "Position": pos_signal}
162.     signalsFrm_2: DataFrame = pd.DataFrame(signals_2)
163.     for i in pos_signal:
164.         pos_y = pos_y + 1
165.         pos_signal_Y.append(pos_y)
166.     main_signals = 0
167.     distant_signal = 0
168.     shunting_signal = 0
169.     speed_limit_signal = 0
170.     speed_limit_distant_signal = 0
171.     for i in type_signal:
172.         if i == 'main':
173.             main_signals = main_signals + 1
174.         elif i == 'distant':
175.             distant_signal = distant_signal + 1
176.         elif i == 'shunting':
177.             shunting_signal = shunting_signal + 1
178.         elif i == 'speed_limit':
179.             speed_limit_signal = speed_limit_signal + 1
180.         elif i == 'speed_limit_distant':
181.             speed_limit_distant_signal = speed_limit_distant_signal + 1
182.     # numSignal = go.Bar(x=1, y=len(id_signal), name='Signals')
183.     # SWITCH
184.     n = 0
185.     for i in id_switch:
186.         n = n + 1
187.         num_switch.append(n)
188.     switch = {"ID": id_switch, "Name": name_switch, "Default Course": defaultCourse_switch, "Type":
type_switch,
189.               "Position": pos_switch, "Left Branch Speed": left_SpeedBranch,

```

```

190.         "Right Branch Speed": right_SpeedBranch}
191.     switchFrm: DataFrame = pd.DataFrame(switch)
192.     # numSwitch = go.Bar(x=2, y=len(id_switch), name='Switches')
193.     numElements = [len(id_signal), len(id_switch)]
194.     # TRACKS
195.     track = {"ID": id_track, "Type": type_track, "Direction": applicationDirection_track,
196.             "From Position": fromPos_tracks, "To Position": toPos_tracks, "Length": length_tracks}
197.     trackFrm: DataFrame = pd.DataFrame(track)
198.     track_y = []
199.     for i in range(len(fromPos_tracks)):
200.         track_y.append(i+1)
201.     main_track = 0
202.     secondary_track = 0
203.     sidingTrack = 0
204.     connectingTrack = 0
205.     for i in type_track:
206.         if i == 'mainTrack':
207.             main_track = main_track + 1
208.         elif i == 'secondaryTrack':
209.             secondary_track = secondary_track + 1
210.         elif i == 'sidingTrack':
211.             sidingTrack = sidingTrack + 1
212.         elif i == 'connectingTrack':
213.             connectingTrack = connectingTrack + 1
214.     # LX
215.     levelCrossing = {"ID": id_LX, "Name": name_LX, "Activation": activation_LX,
216.                     "Direction": direction_LX, "Position": posLX}
217.     levelCrossingFrm: DataFrame = pd.DataFrame(levelCrossing)
218.     posLX_y = []
219.     for i in range(len(posLX)):
220.         posLX_y.append(i+1)
221.
222. except Exception as e:
223.     print(e)
224.     return html.Div([
225.         'There was an error processing this file.'
226.     ])
227.
228. #generacion del dashboard a partir de los datos recogidos
229. plot = html.Div([
230.     #project name, file name, date, OVERVIEW
231.     html.H4(children='Project Name: ' + projectName, style={'textAlign': 'center'}),
232.     html.H5(children='File Name: ' + filename, style={'textAlign': 'center'}),
233.     html.H6(children=datetime.datetime.fromtimestamp(date), style={'textAlign': 'center'}),
234.     html.Hr(), # horizontal line
235.     html.H5(children='Overview', style={'textAlign': 'center', 'font-weight': 'bold'}),
236.     dcc.Graph(
237.         id='elements-graph',
238.         figure={
239.             'data': [
240.                 {'x': [1], 'y': [len(id_signal)], 'type': 'bar', 'name': 'Signals'},
241.                 {'x': [2], 'y': [len(id_switch)], 'type': 'bar', 'name': 'Switches'},
242.                 {'x': [3], 'y': [len(id_track)], 'type': 'bar', 'name': 'Tracks'},
243.                 {'x': [4], 'y': [len(id_LX)], 'type': 'bar', 'name': 'Level Crossings'},
244.             ],
245.             'layout': {
246.                 'title': 'Elements Graph'
247.             }
248.         }
249.     ),
250.
251.     html.Hr(), # horizontal line
252.
253.     #SIGNALS
254.     html.H5(children='Signals', style={'textAlign': 'center', 'font-weight': 'bold'}),
255.     dash_table.DataTable(
256.         id="dataTable2",

```

```

257.     style_data={'whiteSpace': 'normal'},
258.     css=[{
259.         'selector': '.dash-cell div.dash-cell-value',
260.         'rule': 'display: inline; white-space: inherit; overflow: inherit; text-overflow: inherit;']],
261.     data=signalsFrm_2.to_dict('records'),
262.     columns=[{'id': i, 'name': i} for i in signalsFrm_2.columns],
263.     # style_as_list_view=True,
264.     style_header={
265.         'backgroundColor': 'white',
266.         'fontWeight': 'bold'
267.     },
268. ),
269. dcc.Graph(
270.     id='signals-graph',
271.     figure={
272.         'data': [
273.             {'x': [1], 'y': [main_signals], 'type': 'bar', 'name': 'Main'},
274.             {'x': [2], 'y': [distant_signal], 'type': 'bar', 'name': 'Distant'},
275.             {'x': [3], 'y': [shunting_signal], 'type': 'bar', 'name': 'Shunting'},
276.             {'x': [4], 'y': [speed_limit_signal], 'type': 'bar', 'name': 'Speed Limit'},
277.             {'x': [5], 'y': [speed_limit_distant_signal], 'type': 'bar', 'name': 'Speed Limit Distant'},
278.         ],
279.         'layout': {
280.             'title': 'Signals Graph'
281.         }
282.     },
283. ),
284. dcc.Graph(
285.     id='signal-scatter',
286.     figure={
287.         'data': [
288.             go.Scatter(
289.                 x=pos_signal,
290.                 y=pos_signal_Y,
291.                 text=name_signal,
292.                 mode='markers',
293.                 opacity=0.7,
294.                 marker={
295.                     'size': 15,
296.                     'line': {'width': 0.5, 'color': 'white'}
297.                 },
298.             )
299.         ],
300.         'layout': go.Layout(
301.             title='Signals distribution',
302.             xaxis={'type': 'log', 'title': 'Position'},
303.             yaxis={'title': ""},
304.             # margin={'l': 40, 'b': 40, 't': 10, 'r': 10},
305.             legend={'x': 0, 'y': 1},
306.             hovermode='closest'
307.         )
308.     },
309. ),
310.
311.
312.     html.Hr(), # horizontal line
313.
314. #SWITCHES
315.     html.H5(children='Switches', style={'textAlign': 'center', 'font-weight': 'bold'}),
316.     dash_table.DataTable(
317.         id="dataTableSwitch",
318.         style_data={'whiteSpace': 'normal'},
319.         css=[{
320.             'selector': '.dash-cell div.dash-cell-value',
321.             'rule': 'display: inline; white-space: inherit; overflow: inherit; text-overflow: inherit;']],
322.         data=switchFrm.to_dict('records'),
323.         columns=[{'id': i, 'name': i} for i in switchFrm.columns]

```

```

324.     ),
325.     dcc.Graph(
326.         id='Switches-speeds-graph',
327.         figure={
328.             'data': [
329.                 {'x': num_switch, 'y': left_SpeedBranch, 'type': 'bar', 'name': 'Left Speed'},
330.                 {'x': num_switch, 'y': right_SpeedBranch, 'type': 'bar', 'name': 'Right Speed'},
331.             ],
332.             'layout': go.Layout(
333.                 title='Switches Speeds Graph',
334.                 xaxis={'title': 'Switch'},
335.                 yaxis={'title': 'Speed'},
336.             )
337.         }
338.     ),
339.     dcc.Graph(
340.         id='switch-scatter',
341.         figure={
342.             'data': [
343.                 go.Scatter(
344.                     x=pos_switch,
345.                     y=num_switch,
346.                     text=name_switch,
347.                     mode='markers',
348.                     opacity=0.7,
349.                     marker={
350.                         'size': 15,
351.                         'line': {'width': 0.5, 'color': 'white'}
352.                     },
353.                 )
354.             ],
355.             'layout': go.Layout(
356.                 title='Switch distribution',
357.                 xaxis={'type': 'log', 'title': 'Position'},
358.                 yaxis={'title': ""},
359.                 # margin={'l': 40, 'b': 40, 't': 10, 'r': 10},
360.                 legend={'x': 0, 'y': 1},
361.                 hovermode='closest'
362.             )
363.         }
364.     ),
365.     html.Hr(), # horizontal line
366.
367. #TRACKS
368.     html.H5(children='Tracks', style={'text-align': 'center', 'font-weight': 'bold'}),
369.     dash_table.DataTable(
370.         id="dataTableTracks",
371.         style_data={'whiteSpace': 'normal'},
372.         css=[{
373.             'selector': '.dash-cell div.dash-cell-value',
374.             'rule': 'display: inline; white-space: inherit; overflow: inherit; text-overflow: inherit;'}],
375.         data=trackFrm.to_dict('records'),
376.         columns=[{'id': i, 'name': i} for i in trackFrm.columns]
377.     ),
378.     dcc.Graph(
379.         id='Tracks-graph',
380.         figure={
381.             'data': [
382.                 {'x': track_y, 'y': length_tracks, 'text': id_track, 'type': 'bar', 'name': 'Tracks Length'},
383.             ],
384.             'layout': go.Layout(
385.                 title='Tracks Length Graph',
386.                 xaxis={'title': 'Tracks'},
387.                 yaxis={'title': 'Length'},
388.             )
389.         }
390.     )

```

```

391.     ),
392.     dcc.Graph(
393.         id='tracks_type_graph',
394.         figure={
395.             'data': [
396.                 {'x': [1], 'y': [main_track], 'type': 'bar', 'name': 'Main'},
397.                 {'x': [2], 'y': [secondary_track], 'type': 'bar', 'name': 'Secondary'},
398.                 {'x': [3], 'y': [sidingTrack], 'type': 'bar', 'name': 'Siding'},
399.                 {'x': [4], 'y': [connectingTrack], 'type': 'bar', 'name': 'Connecting'},
400.             ],
401.             'layout': {
402.                 'title': 'Tracks Types Graph'
403.             }
404.         }
405.     ),
406.
407.     html.Hr(), # horizontal line
408.
409. #LEVEL CROSSINGS
410.     html.H5(children='Level Crossings', style={'text-align': 'center', 'font-weight': 'bold'}),
411.     dash_table.DataTable(
412.         id="dataTableLX",
413.         style_data={'whiteSpace': 'normal'},
414.         css=[{
415.             'selector': '.dash-cell div.dash-cell-value',
416.             'rule': 'display: inline; white-space: inherit; overflow: inherit; text-overflow: inherit;'}],
417.         data=levelCrossingFrm.to_dict('records'),
418.         columns=[{'id': i, 'name': i} for i in levelCrossingFrm.columns]
419.     ),
420.     dcc.Graph(
421.         id='LX-scatter',
422.         figure={
423.             'data': [
424.                 go.Scatter(
425.                     x=posLX,
426.                     y=posLX_y,
427.                     text=name_LX,
428.                     mode='markers',
429.                     opacity=0.7,
430.                     marker={
431.                         'size': 15,
432.                         'line': {'width': 0.5, 'color': 'white'}
433.                     },
434.                 )
435.             ],
436.             'layout': go.Layout(
437.                 title='LX distribution',
438.                 xaxis={'type': 'log', 'title': 'Position'},
439.                 yaxis={'title': ""},
440.                 # margin={'l': 40, 'b': 40, 't': 10, 'r': 10},
441.                 legend={'x': 0, 'y': 1},
442.                 hovermode='closest'
443.             )
444.         }
445.     ),
446.
447.     html.Hr(), # horizontal line
448. ])
449. return plot
450.
451.
452. @app.callback(Output('output-data-upload', 'children'),
453.               [Input('upload-data', 'contents')],
454.               [State('upload-data', 'filename'),
455.                State('upload-data', 'last_modified')])
456.
457. #funcion update output

```

```
458.def update_output(list_of_contents, list_of_names, list_of_dates):
459.    if list_of_contents is not None:
460.        # children = [
461.        #     parse_contents(c, n, d) for c, n, d in
462.        #     zip(list_of_contents, list_of_names, list_of_dates)]
463.        children = parse_contents(list_of_contents, list_of_names, list_of_dates)
464.        return children
465.
466.
467.if __name__ == '__main__':
468.    app.run_server(debug=True)
469.
```