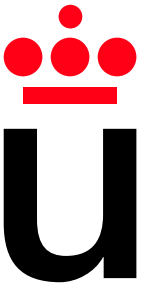

Universidad
Rey Juan Carlos

Escuela Técnica Superior
de Ingeniería Informática

Grado en Ingeniería Informática

Curso 2022-2023

Trabajo Fin de Grado

BACKEND PARA UNA APLICACIÓN DE GESTIÓN Y REPRESENTACIÓN DE DATOS OCEANOGRÁFICOS

Autor: Óscar Ballesteros Izquierdo
Tutor: Manuel Rubio Sánchez

Agradecimientos

A las primeras personas que quiero agradecer este trabajo es a mi familia, que son quienes más me han apoyado para que pueda estudiar y dedicarme a lo que me gusta desde que era pequeño, la informática.

También tengo que agradecer a todos mis compañeros de Medio Físico de Puertos del Estado, he estado cuatro años trabajando con ellos y desde un primer momento siempre me han dato total confianza en los proyectos que he planteado, gracias a ellos he crecido como profesional y siete años después sigo trabajando en aplicaciones relacionadas con la oceanografía, en concreto me gustaría mencionar a las personas que más me han ayudado explicándome los conceptos que necesito para desarrollar este trabajo: Enrique Álvarez, Susana Pérez, Begoña Pérez y José María García-Valdecasas.

Por último, agradecer a mi profesor del TFG, Manuel Rubio Sánchez, que me ha ayudado mucho a sacar unas memorias con la calidad necesaria, y a la Universidad Rey Juan Carlos por todos los conocimientos que he aprendido a lo largo de estos años, también, gracias al programa de prácticas encontré el puesto en Puertos del Estado que tanto me ha ayudado a encauzar mi carrera profesional.

Resumen

El fin del proyecto es desarrollar una interfaz web completamente interactiva y adaptable a cualquier dispositivo, a través de la cual los usuarios puedan ver varios tipos de representaciones de datos oceanográficos, tales como series de tiempo, mapas de calor o tablas especiales.

El primer paso es la recolección de los casos de uso y objetivos que el cliente espera de la aplicación, como pueden ser los distintos elementos interactivos, el tipo de interfaz esperada o los requisitos de carga y rendimiento. Después se estudian todas las posibles tecnologías a implementar y se justifican las elecciones tomadas. Para el frontend se buscan las librerías más importantes de representación gráfica. En la parte backend se investigan los frameworks para el desarrollo de servidores web que se ajustan mejor a las características de este proyecto. Por último, se aborda la forma de implementar estas tecnologías y la puesta en producción para los usuarios finales.

Este proyecto es público y los gráficos son accesibles para todo el mundo a través de la aplicación Portus [1]. Para facilitar el acceso a éstos, al final de este documento se añadirá un apéndice con varios enlaces a distintos gráficos, en los que se podrán ver de forma rápida todas las características más importantes de la aplicación desarrollada.

La descripción del desarrollo de la aplicación se ha dividido en dos trabajos de fin de grado (Ingeniería del Software e Ingeniería Informática). Esta memoria, relacionada con el Grado de Ingeniería del Informática, se centrará en la parte backend de PortusData.

Índice de contenidos

1 Introducción

Este TFG es fruto del trabajo realizado para Puertos del Estado [2], donde además realicé las prácticas externas, en concreto, llevé a cabo el mantenimiento, desarrollo y evolución de aplicaciones web enfocadas a predicciones oceanográficas, la mayoría de ellas se integran en el sistema Portus de acceso público.

El sistema Portus está compuesto de varios servicios desarrollados por distintas personas y empresas. Este trabajo va a tratar uno de los más importantes, PortusData, que ha sido desarrollado completamente por mí.

Tengo el permiso de Puertos del Estado para compartir el código y poder mostrarlo como trabajo final de carrera.

1.1. Alcance

El alcance de este proyecto es poder presentar los datos que tenemos de predicciones y tiempo real de forma gráfica, sencilla e interactiva para el público general, y que a su vez estos gráficos sean precisos, funcionales y tengan algunas herramientas para usuarios más avanzados, hay mucha gente que trabaja en los puertos de España y parte su trabajo depende de estos datos.

Para ello, partimos de una base de datos muy grande y un servidor REST ya existente que sirve los datos de esta, el objetivo es crear una aplicación web que sea capaz de representarlos de varias formas específicas y un pequeño backend que organice de manera eficiente los metadatos de estas y se encargue del routing de la aplicación.

Estos datos se pueden representar de varias formas: series temporales, tablas especiales, gráficos de área, mapas de calor, etc. Cada variable puede tener algunas características específicas para lograr la representación más óptima posible.

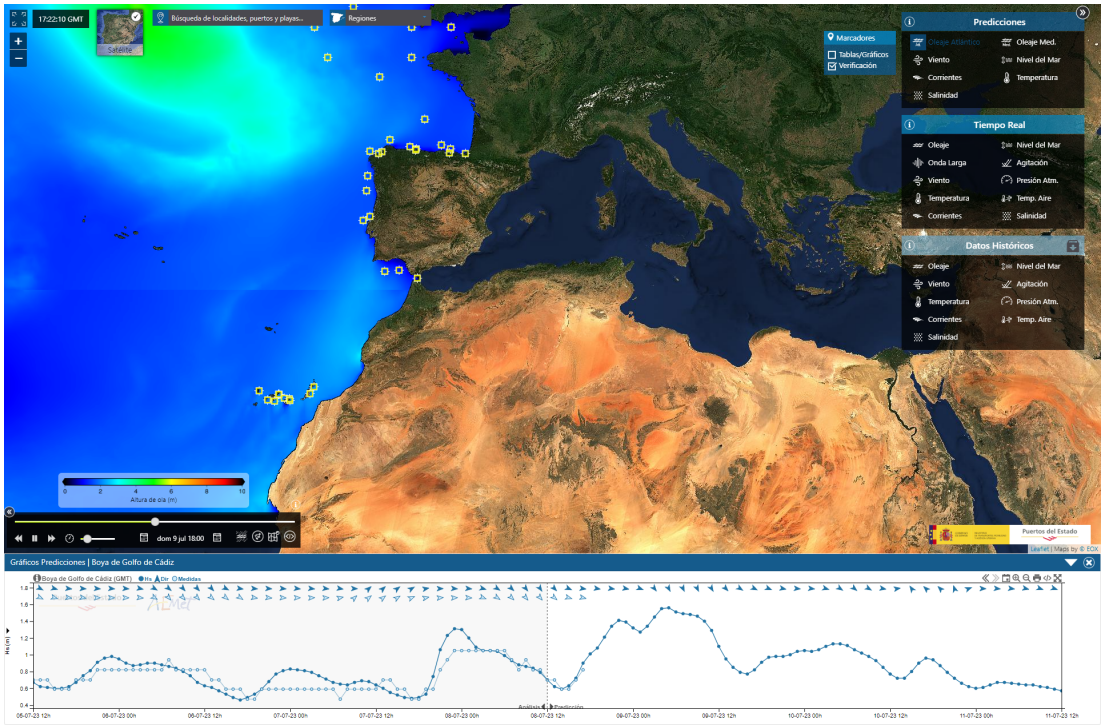

Figura 1.1: Chart de PortusData embebido en Portus.

1.2. Contexto

La idea de este proyecto surge de una necesidad de modernización de los servicios web de Puertos del Estado, partimos de un sistema Portus que se trataba de una sola aplicación que integraba todos los servicios en ella, tales como charts, animaciones, mapas, tablas, reportes en PDF y más. Este sistema tenía dos grandes problemas:

- **Tecnología anticuada:**
  Los dos frameworks principales eran una versión antigua de Spring MVC [3] (con Java 7) y GWT [4], que son perfectamente usables, pero cada vez más en desuso ya que la comunidad ha ido migrando a otras tecnologías más modernas. Esto afectaba mucho a los futuros evolutivos que se querían hacer ya que limitaba bastante las opciones en cuanto a librerías, actualizaciones y seguridad.
- **Complejidad del proyecto:**
  El tener tantos servicios distintos integrados en la misma aplicación generaba muchos problemas, por ejemplo, el generador de reportes en PDF a veces llenaba la memoria del servidor y esto hacía que tampoco se pudiera acceder ni a los gráficos ni a los mapas. También, por la aplicación han pasado distintas empresas y desarrolladores, y a lo largo del tiempo la estructura del proyecto se ha ido haciendo innecesariamente compleja, con mucho código antiguo o duplicado y con partes del proyecto que generaban problemas de compilación.

Figura 1.2: Portus antiguo (Tecnología anticuada, todos los servicios en una app).

Debido a estos problemas, se decidió que para los futuros evolutivos del pro-

yecto la mejor opción era reescribir todo el sistema Portus, pero esta vez usando librerías modernas y con buena proyección y siguiendo un esquema de microservicios en el que se separen las aplicaciones con distinto uso, con un web service central que se encargue de nutrirlas de los datos necesarios.

La integración de este nuevo esquema ha traído bastantes cosas beneficiosas para los servicios web:

- **Código con más funcionalidad y menos complejidad:**
  El reescribir de cero todo y la modernización de los frameworks nos ha permitido aplicar los nuevos estándares web que han ido surgiendo, y esto se refleja mucho en la calidad y menor complejidad del código final.
- **Los servicios se mantienen en pie:**
  Antes era un gran problema ya que los servicios se pasaban caídos una parte muy importante del tiempo, la modernización de librerías ha ayudado, pero lo que más ha influido en esto es la separación en microaplicaciones, ahora si surge un problema en una de poca importancia, esto no se propaga hacia las principales.
- **Mejora en el mantenimiento:**
  Al ser aplicaciones separadas, ahora distintas empresas o desarrolladores pueden trabajar en una de ellas solo y especializarse en esta, lo que ha hecho mucho más eficiente la forma de interactuar entre terceros.
- **Mayores posibilidades de evolución:**
  En los sistemas de información geográfica se trabaja con muchos estándares que evolucionan de forma muy rápida, el tener frameworks con una gran comunidad nos ayuda a poder implementarlos de forma mucho más rápida y sencilla que antes.

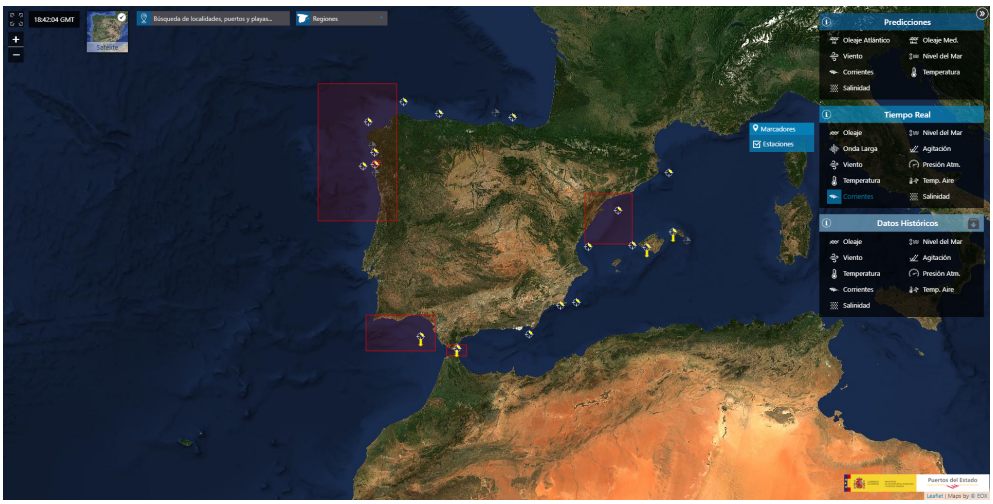

Figura 1.3: Portus nuevo (Tecnología moderna, arquitectura de microservicios).

2

Objetivos

Este proyecto trata de uno de los nuevos servicios más importantes: los charts. Partimos de una versión que estaba integrada con todo el sistema Portus antiguo, usaban una librería para GWT bastante antigua y con pocas capacidades y posibilidades de interacción, solo sirve para representar series temporales en formatos muy concretos.

Los nuevos charts queremos que sean un proyecto individual mucho más potente que las gráficas originales, que represente los datos de varios formatos distintos y no solo series de tiempo, que sean mucho más interactivos y fáciles de manejar para los usuarios y que puedan ser usados no solo en Portus, sino en otras aplicaciones que ofrecen e incluso el público pueda usarlos como widgets en su página web.

Tras la aprobación del nuevo proyecto, paso a recoger los datos del cliente sobre los charts actuales, qué cosas se quieren mejorar de ellos y qué nuevas características esperan de PortusData. Las características más importantes relacionadas con el backend que se recogen del proyecto son:

- Poder servir varios tipos de representaciones gráficas (series de tiempo, mapas de calor, bandas de confianza, etc).
- Gestión de los metadatos de modelos y estaciones.
- Posibilidad de elegir el idioma de los textos (español e inglés).
- Gestión de los logos de los proveedores de datos.
- Posibilidad de servir los gráficos como imagen.
- Tienen que poder compilarse para acceder a los datos de cualquier entorno (desarrollo, preproducción y producción).

3

Descripción Informática

3.1. Metodología

Para hacer este proyecto tenemos muy claros los requisitos, lo que queremos mostrar y el resultado final de la aplicación.

Desde el principio se ha planteado de tal forma que se pudiera hacer una primera entrega con el tipo de chart más simple: los de predicción, después se irían añadiendo el resto de gráficos de forma individual según las preferencias que marque el cliente. Según se han ido añadiendo más tipos, se han replanteado los anteriores ya creados con tal de buscar código y funcionalidad comunes, reduciendo así la complejidad total de la aplicación.

Como los tipos de chart se han ido desarrollando uno a uno, se vio conveniente usar la metodología de cascada para la creación de cada uno de ellos, haciendo varias iteraciones a las fases de esta por cada desarrolo de un nuevo gráfico:

- **Requisitos:**
  Cuando se va a crear un tipo de chart es muy importante tener claros los requisitos particulares de este y qué estamos representando.
  El primer paso siempre es un traspaso de los conocimientos oceanográficos básicos relativos al gráfico, de esto se ha encargado el cliente que es quien me ha explicado todos los conceptos necesarios para poder comenzar a representar las distintas variables.
  Por último, es necesario plantear todos los elementos que se van a mostrar y cómo el usuario interaccionará con ellos.

- **Diseño y construcción:**
  Una vez sabemos los requisitos, lo siguiente es comenzar a diseñar el chart, en esta fase usamos datos precargados y nos olvidamos de momento del backend.
  En el primer tipo de chart se tuvieron que definir algunas cosas como las distintas interfaces y los controles que en los siguientes no ha sido necesario.
  Por último, se construye el código del frontend del gráfico y pasa una primera revisión por mi parte.
- **Fase de prueba:**
  Cuando ya tenemos una primera demo del chart, se hace una primera iteración con el cliente y se le muestra todo el funcionamiento.
  Normalmente hay algún concepto técnico que no ha quedado claro y hay que modificar algún elemento del gráfico, por ejemplo, el punto de referencia de las flechas de dirección o el cálculo de algunas unidades deben ser revisados siempre por algún experto en el tema.
  Se corregirán los errores detectados y se harán todas las iteraciones de esta fase necesarias hasta que el cliente lo dé por aprobado.
- **Instalación / implantación:**
  El siguiente paso es crear el código del backend para que el chart pueda funcionar en todos los puntos de modelo o estaciones existentes.
  Para ello el primer paso es hacer las llamadas necesarias al web service para recuperar los metadatos y almacenarlos en memoria.
  Por último, se crea una clase Java que atenderá a un endpoint en concreto y manejará los parámetros que se manden para crear dinámicamente la plantilla html que será servida al usuario.
- **Soporte y mantenimiento:**
  Una vez ya tenemos creado tanto el frontend como el backend, la aplicación está lista para ser entregada al cliente.
  El cliente primero subirá esta primera versión a preproducción, donde hará una última revisión.
  Por último, si todo está bien, la aplicación estará lista para subir a producción y se vuelve a comenzar este proceso en cascada hasta hacer todos los tipos de gráficos requeridos en este proyecto.

3.2. Análisis

3.2.1. Requisitos funcionales

Los requisitos funcionales son declaraciones detalladas que describen las acciones y funciones que la aplicación debe ser capaz de realizar para satisfacer las necesidades del usuario o del cliente.

Los relativos al backend de PortusData son los siguientes:

- **Carga de metadatos:**
  El servidor deberá precargar en memoria una serie de datos al arrancar.
- **Resolución de URLs:**
  El backend escuchará llamadas http, analizará el endpoint y los parámetros enviados para resolver la petición.
- **Servir gráficos:**
  Una vez resuelta esta URL, se mandará al usuario el tipo de gráfico adecuado.
- **Exportar gráfico a imagen:**
  A través de phantomjs [5], se podrá sacar una captura de este gráfico y será mandada al usuario.
- **Internacionalización:**
  Se podrán servir los textos tanto en inglés como en español.

3.2.2. Requisitos no funcionales

Los requisitos no funcionales son las restricciones no impuestas al sistema, se centran en cómo se deben hacer las cosas y están relacionados con la calidad del software final. Será necesario cumplirlos todos para garantizar la satisfacción del cliente y los usuarios.

Para esta parte del proyecto son los siguientes:

- **Velocidad de carga:**
  Cuando se cambian los metadatos, es necesario reiniciar PortusData para actualizarlos, por ello, la aplicación deberá tardar unos pocos segundos en arrancar.
- **Tolerancia a fallos:**
  Los datos cambian constantemente y es fácil encontrarse variables que se dejan de medir, para ello es importante que el gráfico siempre intente mostrar algo de información aunque un parámetro no sea correcto o haya dejado de medirse.
- **Compatibilidad:**
  Se elige el modo de empaquetado de aplicaciones WAR ya que las máquinas del cliente están ya preparadas para desplegarlos.
- **Logging:**
  Los logs se sacan a un path de la máquina "/var/log/PortusData" que sigue el esquema de monitoreo de aplicaciones del cliente-
- **Rendimiento:**
  Las solicitudes se deben procesar lo más rápido posible, para ello se eligen las estructuras de datos que nos permiten recuperar la información necesaria de la forma más rápida posible.

3.2.3. Casos de uso

Los casos de uso sirven para capturar y describir las interacciones entre el sistema y los actores, en este caso el usuario.

Representan una forma de entender y documentar toda la funcionalidad que tendrá la aplicación, centrándose en las tareas específicas que serán llevadas a cabo por los actores.

El diagrama de casos de uso es una representación gráfica de todas estas interacciones y funcionalidades, de forma esquemática, sin entrar en los detalles internos de su representación:

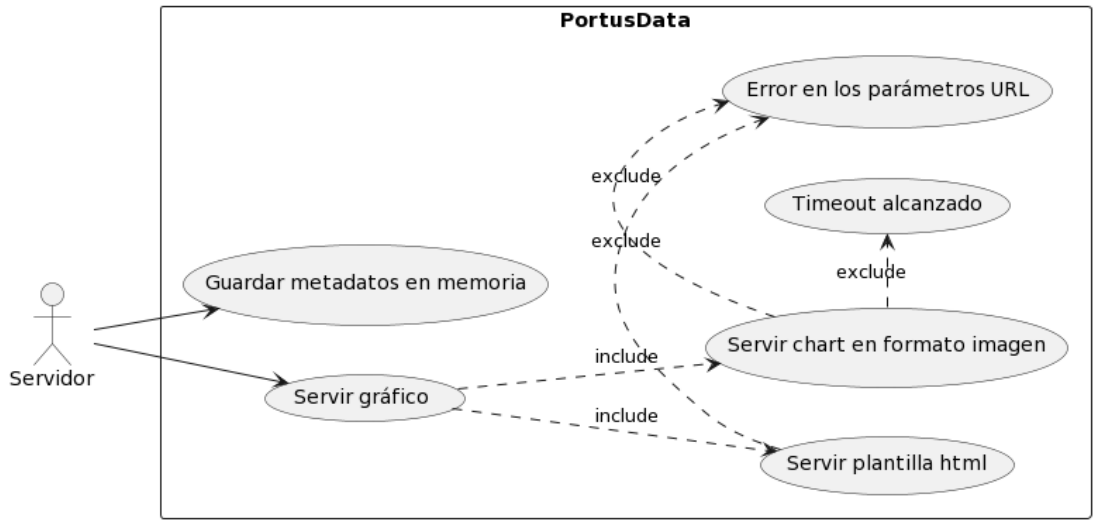

Figura 3.1: Diagrama de casos de uso del backend.

A continuación, se muestran las tablas de casos de uso, que es otra forma de representarlos en la que las interacciones se describen con más detalle:

Caso de uso	Guardar metadatos en memoria
Actores	Servidor
Descripción	El backend almacenará cierta cantidad de información para hacer más rápido el proceso de carga del chart
Precondición	Arrancar PortusData
Escenario	Nada más lanzar la aplicación, se harán varias llamadas a SIMO para descargar la información necesaria, posteriormente se almacenarán en la estructura de datos más adecuada para su posterior recuperación
Postcondición	Una vez se han guardado los metadatos en memoria, continúa la carga de Spring Boot y el servidor comienza a escuchar llamadas
Extensión	-
Inclusión	-

Caso de uso	Servir gráfico
Actores	Servidor
Descripción	La función de PortusData es mostrar al usuario varios tipos de charts
Precondición	La aplicación deberá haber arrancado correctamente
Escenario	El servidor analizará el endpoint y los parámetros introducidos en la URL y devolverá al usuario el tipo de gráfico adecuado
Postcondición	El backend seguirá en espera hasta recibir una nueva llamada
Extensión	-
Inclusión	▪ Servir plantilla html ▪ Servir chart en formato imagen

Caso de uso	Servir plantilla html
Actores	Servidor
Descripción	El usuario recibirá un gráfico interactivo
Precondición	La aplicación deberá haber arrancado correctamente
Escenario	Se generará una plantilla html dinámica y se inyectarán variables a esta para posteriormente servirse al usuario.
Postcondición	-
Extensión	▪ Error en los parámetros URL
Inclusión	-

Caso de uso	Servir chart en formato imagen
Actores	Servidor
Descripción	El usuario recibirá un chart en formato imagen
Precondición	La máquina deberá tener instalado phantomjs
Escenario	El servidor cargará el chart en un navegador headless y sacará una captura del gráfico en un archivo temporal, que será mandado al usuario como respuesta
Postcondición	Se borran los archivos temporales generados
Extensión	▪ Error en los parámetros URL ▪ Timeout alcanzado
Inclusión	

Caso de uso	Error en los parámetros URL
Actores	Servidor
Descripción	Ocurre cuando se construye mal una URL
Precondición	Hacer una llamada con parámetros incorrectos
Escenario	Si el backend no reconoce alguno de los elementos indispensables para que funcione el gráfico, se servirá una plantilla con un mensaje de error
Postcondición	-
Extensión	-
Inclusión	-

Caso de uso	Timeout alcanzado
Actores	Servidor
Descripción	Al cargar un chart en formato imagen, hay un tiempo máximo en el que se espera al servidor para responder con esta.
Precondición	Hacer una petición al servidor de un chart con la interfaz “img“
Escenario	Si pasa demasiado tiempo y el phantomjs no recibe la señal del chart indicando que ha terminado de cargar, se termina el proceso de phantomjs y se manda una imagen vacía.
Postcondición	-
Extensión	-
Inclusión	-

3.3. Especificación

La versión de los gráficos de la que partimos y queremos renovar estaba integrada en otra aplicación junto a muchos otros servicios, el backend servía tanto los datos como los metadatos del gráfico, previamente formateados para que sean compatibles con la librería de charting que se usaba.

Problemas o carencias de esta versión:

- Alta complejidad y muchas duplicidades en el código que hace las llamadas a la BD.
- Hacer un cambio en la forma de servir los datos de los gráficos podría romper la funcionalidad de otro servicio como las tablas.
- Se hacían una gran cantidad de llamadas para gestionar los metadatos cada vez que se carga un gráfico.
- Solo podían servir datos para hacer gráficos de series temporales.
- No había límite de datos, una llamada mal hecha podía bloquear temporalmente la BD de producción.
- El título del gráfico se puede configurar por un parámetro en la URL y estos están preparados para que la gente los comparta en su web, cualquiera podría cambiar el título por un mensaje ofensivo y esto saldría con el logo del cliente.
- Parámetros web muy complejos, que se codificaban en Base64 y complicaba mucho construir las URLs de estos.
- Usaban Java 7, con bastantes problemas de seguridad encontrados.

La mayoría de estos problemas surgen de la concepción inicial del sistema Portus, al principio estaba pensado para ser solo una aplicación sencilla que muestre unos pocos modelos y gráficos, y según ha ido evolucionando, se han hecho muchos parches en el código para ir cumpliendo con las nuevas funcionalidades que se han ido requiriendo, esto, junto a que las librerías usadas se estaban en desuso por la comunidad, motivó la reescritura de todos los servicios, y en concreto, la separación de estos gráficos de la aplicación principal, creando un nuevo proyecto mucho más potente y reutilizable.

Debido a que ya no tenemos el backend de Portus para servir los datos y metadatos de los gráficos, necesitamos crear uno nuevo cuya función sea únicamente la gestión de estos, lo primero que se requiere de este backend es abandonar la alta complejidad en el código y empezar a hacer aplicaciones sencillas de usar por cualquier desarrollador.

Portus era una aplicación bastante antigua que gestionaba a su manera el acceso a los datos de la BD, pero más tarde se creó en Puertos del Estado un web service para el acceso a toda esa información llamado SIMO, que puede servir todos los datos que necesitamos para este proyecto, gracias a esto, nuestro backend no va a tener que gestionar ningún tipo de acceso a bases de datos,

haciendo un código mucho más simple y evitando los problemas de seguridad que pueden surgir a la hora de hacer las conexiones.

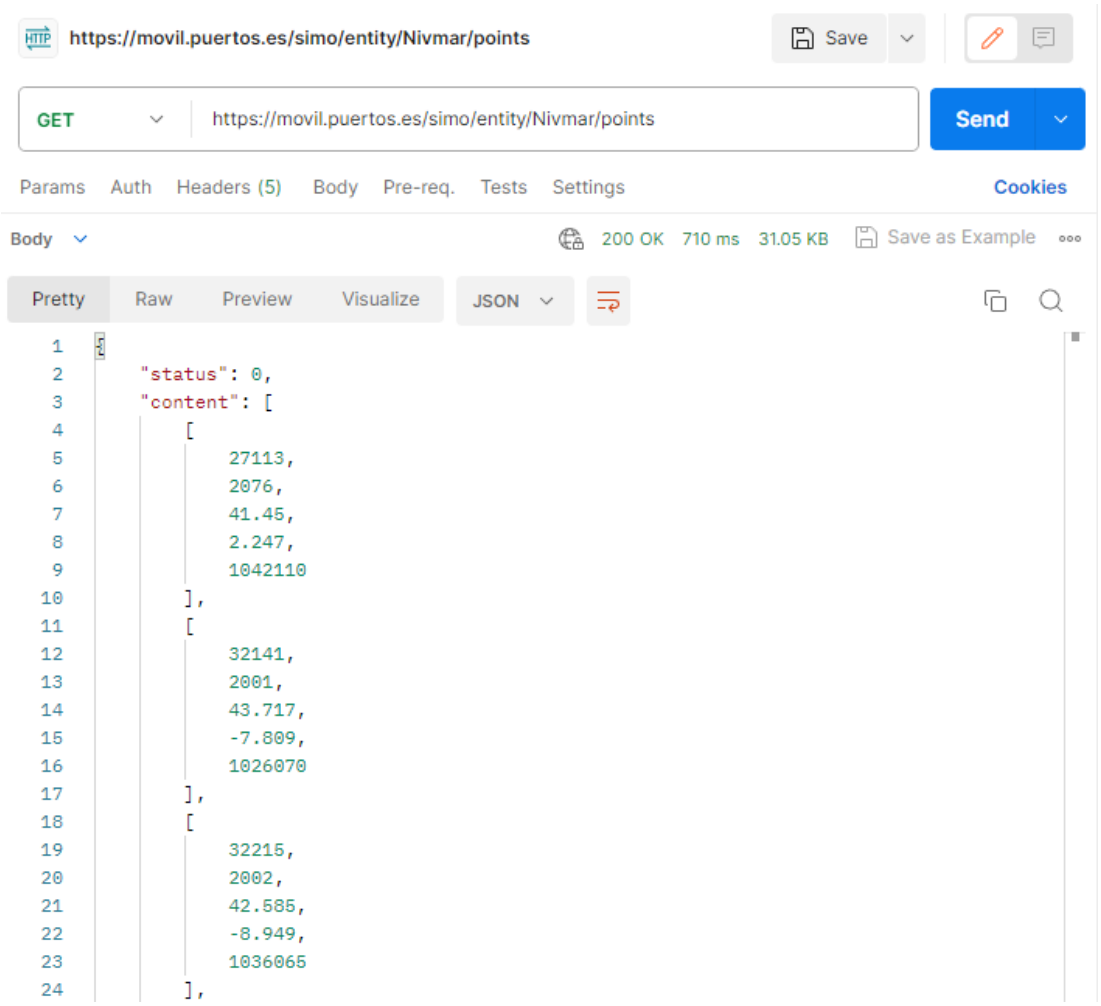

Figura 3.2: Ejemplo de llamada a SIMO que devuelve metadatos de puntos de nivel del mar.

Entonces, SIMO servirá tanto los datos directamente al frontend, como los metadatos al backend, por lo que la función de este va a ser prácticamente solo la gestión de esta información y del routing y parámetros de las URLs. Esto resultará en un servidor bastante sencillo y la mayoría de la complejidad de PortusData se encuentra en la parte frontend.

Una vez tenemos clara la funcionalidad que va a tener nuestro backend, lo siguiente es elegir la tecnología más adecuada para desarrollarlo, en este caso, no hay total libertad de elección, se restringe a solo poder usar Java o Python como lenguaje, también, el framework que seleccionemos tiene que poder integrarse en las máquinas actuales sin tener que hacer grandes cambios.

Dadas estas restricciones, lo mejor es buscar una tecnología o framework que sea ya usado en alguna de las aplicaciones existentes o que sea compatible con estos, destacan tres grandes opciones:

- **Spring MVC:**
  Spring es uno de los mayores frameworks para desarrollo de aplicaciones web en Java. Se basa en el patrón Modelo-Vista-Controlador y cuenta con una amplia variedad de características y funcionalidades como la gestión de la seguridad, inyección de dependencias, acceso a datos y muchos más componentes. También cuenta con una amplia comunidad de desarrollo. Portus está escrito usando Spring MVC, por lo que es un buen candidato para este proyecto, aunque este se desarrollaría sobre Java 8.

- **CherryPy:**
  CherryPy es un framework de desarrollo web en lenguaje Python, se enfoca en ser una herramienta simple y minimalista para la creación de web services. Proporciona una base de funcionalidades necesarias para esto, pero no cuenta con todas las características más complejas que requiere un proyecto grande. SIMO está escrito en CherryPy usando python 2.7, si se elige esta tecnología, se podría hacer con python3.

- **Spring Boot:**
  Spring Boot es un subproyecto de Spring que se centra en simplificar y agilizar el proceso de configuración y desarrollo de aplicaciones basadas en Java. Su función es la de crear aplicaciones web listas para producción usando una configuración mínima, lo que hace que carezca de muchas de

las funcionalidades más complejas de Spring MVC. Es ideal para proyectos pequeños y cuenta también con una gran comunidad de desarrollo. En este caso no hay ninguna aplicación existente que use este framework, pero el ser compatible con Spring MVC lo hace un buen candidato para nuestro backend.

Dado que buscamos un servidor que sea sencillo y con poca complejidad en el código, la primera opción que se descarta es Spring MVC ya que está pensado para proyectos mucho más complejos y grandes que este, tanto CherryPy como Spring Boot cuadran más en un proyecto de estas características, finalmente se elige Spring Boot por los siguientes motivos:

- CherryPy es un framework más orientado a crear servicios REST completos, el backend de PortusData solo gestionará los parámetros URL y servirá el código html del gráfico.
- La integración de Spring Boot con la tecnología de plantillas Thymeleaf [6] ayudará a gestionar el frontend de forma fácil.
- La mayor parte de las aplicaciones del cliente están desarrolladas en Java.
- Las máquinas en las que desplegará la aplicación ya están preparadas para usar la tecnología de empaquetado de aplicaciones Java en archivos WAR.

3.4. Diseño

Una vez elegido Spring Boot como framework para desarrollar nuestro servidor, lo siguiente es tener clara toda la funcionalidad a implementar en nuestra aplicación:

- El punto de entrada será vía URL, donde cada tipo de chart tendrá un endpoint diferente, cada uno de ello contará con varios parámetros tales como código, variables a mostrar, idioma, etc.
- La aplicación tendrá que elegir la plantilla adecuada para servir el frontend.
- A partir de los parámetros dados, el servidor elegirá los metadatos asociados al código de modelo o estación aportado.
- Por último, estos metadatos se inyectarán en la plantilla en forma de variables y se servirá el código html generado al usuario.

3.4.1. Parámetros URL

El routing y los parámetros que se indiquen en la URL de acceso al chart son muy importantes ya que decidirán el contenido final del gráfico, una mala gestión de los parámetros, como indicar mal el nombre de una variable, también podrá hacer que el gráfico falle y no pueda mostrar ningún dato.

Cada tipo de chart puede tener sus propios parámetros, pero la aplicación se ha diseñado para que sean lo más homogéneos posible entre ellos.

Ya que se requiere que sean fáciles de integrar en las distintas aplicaciones, es importante que los parámetros sean lo más sencillos y entendibles que se pueda, por ejemplo usando String IDs de variables que sean reconocibles en vez de IDs numéricas, así podemos construir las urls de estos de fácilmente sin tener que consultar ningún manual complejo para esta función.

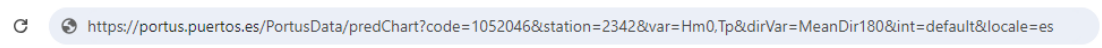

Figura 3.3: Ejemplo de url para una petición de un chart de predicción.

Los parámetros más importantes que se pueden encontrar son:

- **code, station:**
  Son los parámetros más importante de la aplicación, si no hay al menos uno de ellos configurado el chart fallará seguro ya que no podrá rescatar ningún dato.
  - "code" se usa para indicar el código del punto de modelo en caso de predicciones.
  - "station" indica el código de la estación que dará las medidas de tiempo real.
- **var, param:**
  Tienen la misma funcionalidad, pero "param" se usa en los charts de tiempo real y "var" en predicciones, indican la o las variables que se quieren mostrar en el gráfico, si no se indica ninguno de estos tampoco podrá mostrarse ningún dato.
  Las IDs se ponen en formato texto y es el cliente quien las decide ya que se usan nombres técnicos abreviados y tienen que ser reconocibles por ellos.
  Si se indica alguna ID incorrecta, el chart no la mostrará en vez de dar error, así evitamos que falle una gráfica entera por solo un dato mal puesto, pero como mínimo debe haber una variable correcta para poder cargar algún dato.

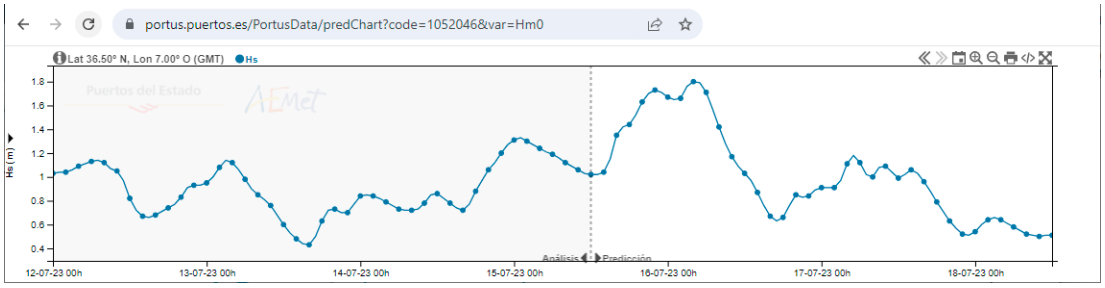

Figura 3.4: Parámetros url mínimos para llamar a un chart.

- **dirVar:**
  Sirve para que una variable se pinte en arriba en formato flecha de dirección en vez de en uno de los ejes, si se pone una variable que no sea de dirección los resultados serán extraños, por lo que la aplicación que incluya el gráfico deberá de encargarse de colocarlas donde corresponda.

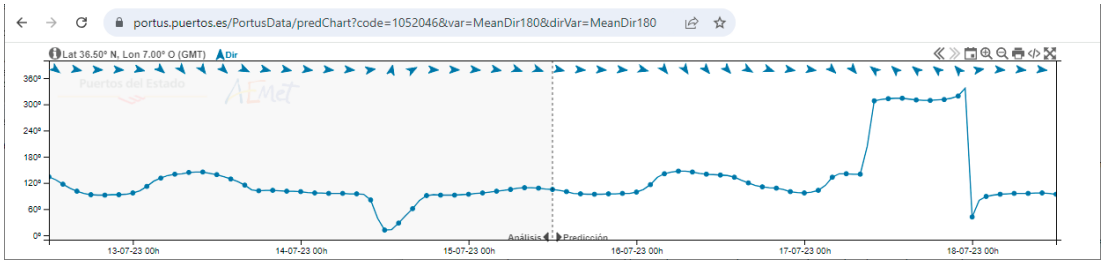

Figura 3.5: Las variables de dirección se pueden mostrar tanto en el eje Y como en formato de flecha de dirección, aunque su uso habitual es el segundo.

- **verVar, dirVerVar:**
  Sirven para los charts de predicción, indican las variables de las que se quieren poner medidas para comparar, necesitan que antes se indique el parámetro "station" para poder hacer la petición de los datos.
- **int:**
  Indica la interfaz en la que se va a mostrar el gráfico, si no se indica, el valor por predentermindao será "default"
  - **default:** Es la interfaz más completa.
  - **min:** Se reducen los botones a los esenciales y se ajustan los márgenes.
  - **prev:** Se eliminan todos los botones e interactividad en el gráfico.
  - **img:** El backend responderá con el chart en formato imagen en vez de html interactivo.
- **unit0, unit1:**
  Pueden cambiar la unidad en la que se carga por primera vez el chart, si no se indica, carga en la unidad por defecto definida en la configuración de la variable.

Las ids son definidas por el cliente y en formato texto, ejemplos: “METER“ , “KNOTS“...

- “unit0“ cambiará la unidad del eje izquierdo.
- “unit1“ la del eje derecho, si existe.

- **locale:**
  Permite seleccionar el idioma de los textos que se ven en el chart, si no se indica, por defecto cargará en español.
  - “es“ para indicar idioma español.
  - “en“ para cambiar el idioma a inglés.

Figura 3.6: Chart de tiempo real en inglés.

- **from,to:**
  Es una funcionalidad poco usada, permiten hacer la carga inicial del gráfico en un intervalo de tiempo personalizado, el formato que se usa para indicar las fechas es el mismo que el del web service al que se piden los datos. Ejemplo: 20230715@0000
- **umbrales:**
  Este parámetro solo se usa en aplicaciones privadas en las que los usuarios pueden configurarse sus propios umbrales en los puntos de modelo o estaciones.
  Solo puede haber umbrales en uno de los parámetros no direccionales que se midan, el formato para indicarlo es “IDVariable:umbral1;umbral2“

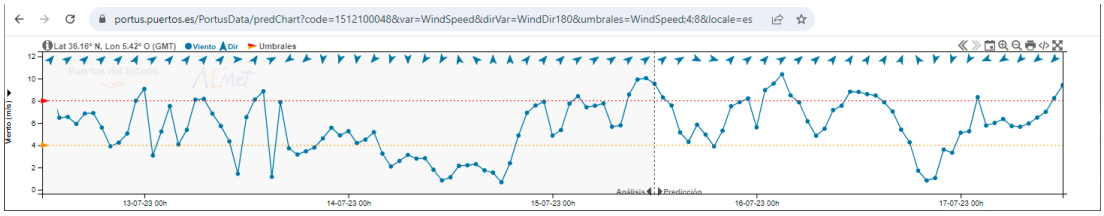

Figura 3.7: Chart con umbrales configurados.

3.4.2. Plantillas

Este servidor no responderá con datos, sino que devolverá el código necesario para cargar el gráfico que se desea. Para hacer este trabajo es conveniente usar un sistema de plantillas que puedan generar html dinámicos y mandarlos al usuario.

Thymeleaf [6] es el motor de plantillas más ampliamente usado en el framework de Spring, por lo que será la librería elegida para este proyecto. Sus características más importantes son:

- Sintaxis intuitiva y sencilla que se asemeja al html estándar.
- Fácil integración con Spring.
- Permite crear contenido dinámico de lado del servidor
- Permite vincular un contexto de datos que nos ayuda a pasar las variables necesarias.

Cada endpoint del servidor equivale a un tipo de chart y cada uno de ellos tiene su propia plantilla html, que será modificada dinámicamente dependiendo de los parámetros que se manden en la URL de la petición.

3.4.3. Gestión de los metadatos

Los metadatos son una parte esencial de este proyecto, ya que aportan información muy valiosa para el usuario y que puede ayudar a comprender mejor los datos.

Esta información es servida por SIMO, por lo que este backend tendrá que hacer peticiones REST para rescatar esta información.

El frontend también hace llamadas al mismo servidor, entonces, ¿Por qué no se piden estos metadatos también directamente desde el navegador?

La respuesta es que esta información se encuentra dispersa en la BD y para acceder a todo lo que necesitamos mostrar se necesita hacer varias llamadas a SIMO, pudiendo llegar a ser cuatro o cinco peticiones para obtener toda la información necesaria relativa a un punto de modelo o estación.

Si el backend tuviera que hacer estas llamadas cada vez que se pide un chart, también sería ineficiente, por lo que para resolver esto, se precargan absolutamente todos estos datos en memoria al arranque de la aplicación.

Los datos de las series temporales son muy grandes, se cuentan por Terabytes, sin embargo, estos metadatos, son muchos, pero ocupan muy poco en memoria y se descargan muy rápido a través de SIMO.

Gracias a esto se consigue que la carga inicial del chart sea lo más rápida posible, ya que el backend ya dispone toda la información necesaria para mandarla al frontend, el ahorrar estas llamadas también evita que el servidor REST pueda colapsar en momentos de picos de carga altos.

Por último, este sistema de precarga de la información también tiene dos inconvenientes:

- El arranque de la aplicación es más lento, tarda unos diez segundos en descargar toda la información necesaria.
- Si se actualiza alguno de estos datos en la BD, no se vería reflejado en los charts hasta que se reinicie la aplicación.

Se presentan estas ventajas y desventajas al cliente, que es quien finalmente ha decidido que lo más óptimo es precargar estos datos en vez de hacer estas peticiones en cada carga de chart.

3.5. Implementación

3.5.1. Gestor de proyectos

Antes de empezar con el código de la aplicación es importante elegir un buen gestor de proyectos que nos permita importar las librerías necesarias y gestionar el proceso de montaje final de la aplicación.

Para este tipo de proyectos la herramienta más usada es Maven [7], desarrollada por Apache Software Foundation, que será la elegida para integrar con esta aplicación. Sus características más importantes son:

- Gestión automática de dependencias.
- Permite configurar el ciclo de vida del proyecto, tanto para desarrollo como para el entorno de producción.
- Sistema de plugins que permite ampliar la funcionalidad del proceso de montaje.
- Se basa en el concepto de "Project Object Model"(POM).

Una vez integrado Maven en el proyecto, el siguiente paso es añadir todas las dependencias necesarias, serán descargadas e importadas de manera automática.

Por último hay que configurar todo el proceso de montaje de la aplicación final.

Los comando para la gestión del proyecto más importantes de Maven son:

- **clean:**
  Nos permite limpiar las build anteriormente creadas en el proyecto para dejarlo limpio.
- **compile:**
  Sirve para montar el proyecto y mostrar errores si surgen en tiempo de compilación.
- **package:**
  A través de los archivos compilados anteriormente, los organiza y empaqueta en formato WAR listo para su despliegue final.

3.5.2. Descarga de metadatos

Como ya se ha comentado en el apartado de diseño, la información de estos metadatos se descarga al iniciar el servidor y se queda guardada en memoria.

Para ello, tenemos que modificar el punto de entrada de la aplicación y meter código Java antes de que Spring Boot comience a escuchar las peticiones que se hagan al backend, en este caso se encuentra en el archivo PortusDataApplication.java.

Estos datos son de miles de puntos de modelo y cientos de estaciones, pero no son muy pesados en memoria, se hacen unas 20 distintas llamadas para recibir toda la información que necesitamos y los almacena cada uno en la estructura de datos más óptima para su posterior recuperación, generalmente son mapas que usan algunas clases Java auxiliares creadas para facilitar esta función. Estas clases auxiliares se encuentran en los siguientes archivos:

- **Datum.java:**
  Guarda información relativa a mareógrafos que miden el nivel del mar.
- **Cero.java:**
  Sencilla clase que almacena los distintos puntos de referencia que pueden tener los mareógrafos mencionados anteriormente.
- **Details.java:**
  Contiene información relativa a los puntos de modelo, tales como nombre, latitud, longitud y fecha de inicio de datos.
- **Param.java:**
  Contiene información importante sobre la variable de tiempo real que se quiere medir, por ejemplo, la id, el nombre, descripción, unidad de referencia...
- **Station.java:**
  Son los datos relevantes a la estación de medidas de tiempo real que se solicita, tales como latitud, longitud, nombre, ubicación, estado...
- **Radar.java:**
  Los radares son un tipo de estación que no solo mide un punto, sino una malla, a parte de incluir los datos de la clase Station, tiene información relativa al punto de la malla que se quiere ver.
- **StationParam.java:**
  Relaciona estaciones con los parámetros que puede medir y la cadencia de esta medida.
- **Unidad.java:**
  Guarda la información necesaria para hacer los cambios de unidad, contiene datos importantes como el símbolo, referencia y los cálculos que hay que hacer para transformar los valores.
- **Variable.java:**
  Contiene la información de las variables de predicción, los datos de tiempo

real están almacenados en BD y se pueden recuperar con SIMO, en este caso, nunca se ha llegado a crear la estructura necesaria para guardar los de predicción, por lo que se usan unos archivos de configuración en el proyecto que contienen toda esta información, estos archivos se llaman varConf.json y varConf.en.json
Los datos más importantes que se almacenan en esta clase son: nombre de la variable, abreviación, descripción, unidad base, color de la serie temporal, ID de la variable para hacer la llamada al web service o el modelo de predicción al que pertenecen.

Gracias a estas clases, podemos almacenar de forma muy eficiente la gran cantidad de datos que hay que descargar del web service, luego se recupera la pieza de esta información relativa al código o parámetros solicitados y se inyecta en la plantilla de la forma más rápida posible.

3.5.3. Plantillas

Como ya se ha indicado en la sección de diseño, por cada tipo de chart hay una plantilla distinta que se genera dinámicamente en función de los datos que requiera inyectar ese gráfico.

La gestión de las plantillas comienza desde el momento que se hace la petición al endpoint con el ID de esta, por cada ID de plantilla hay un archivo .java que maneja los parámetros mandados vía url e inyecta las dependencias necesarias. Estos archivos son:

- ADCPChart.java para los perfiladores de corrientes.
- astroChart.java para las series temporales de marea astronómica.
- nivmarChart.java para las predicciones de nivel del mar.
- predChart.java para los charts de predicciones generales.
- rtChart.java para las series de tiempo real.
- tablasMareas.java para las tablas de mareas.

La función de estas clases Java es primero leer los parámetros que se han mandado en la petición, como ya tiene precargada toda la información posible, lo siguiente que hace es comprobar que estos parámetros se hayan mandado correctamente, por ejemplo, que el código de modelo aportado exista, que los parámetros que se mandan sean medidos por la estación indicada, que la unidad esté bien, etc.

Una vez se ha verificado que esta información es correcta, se pueden empezar a generar los objetos y variables que se van a inyectar en las plantillas, cada una de ellas tendrá sus propias características pero se han homogeneizado todo lo posible, el nombre de estas plantilla será del tipo "idTipoChart.html". Ejemplos de datos que se inyectan en las plantillas:

- Detalles de la estación o punto de modelo.
- Distintas unidades entre las que se podrán transformar los valores.
- Idioma de la aplicación.
- Tipo de interfaz a cargar.
- Logos a pintar.
- URLs necesarias para pedir los datos de las series temporales.
- Objeto con la configuración de los ejes izquierdo, derecho y/o direccional.

La función de estas plantillas es, a través de estos datos inyectados, generar las variables globales que almacenen la información que necesitan, también incluirán todos los archivos con código JavaScript común para la funcionalidad del frontend. Dependiendo de la información que reciben, también eligen de forma dinámica otros tipos de scripts a insertar, tales como la interfaz, los controles o los estilos css.

Por último, se inserta el script principal con la funcionalidad de cada tipo de

gráfico, que también tendrá un nombre “idTipoChart.js“.

Una vez se ha insertado este archivo, el frontend ya puede arrancar y comenzar a descargar los datos de las series temporales para posteriormente comenzar a pintar todos los elementos en pantalla.

3.5.4. Entornos

Estos gráficos son de mucha utilidad para el propio cliente. Es una herramienta que sirve para la comprobación y validación de los nuevos modelos de predicción que se van lanzando.

Estos modelos primero se lanzan en los entornos de desarrollo ya que necesitan validarse antes de ser puestos en producción.

Los entornos con los que cuenta el cliente son:

- **Desarrollo:**
  Es el entorno que se usa para hacer cualquier tipo de pruebas, puede ser inestable y tener diferencias con las máquinas de producción.
- **Preproducción:**
  Sirve para probar las cosas en un entorno estable antes de ir a producción, a estos dos primeros entornos solo se puede acceder desde la red interna de Puertos del Estado.
- **Producción:**
  Es el entorno final donde se desplegarán las aplicaciones y los usuarios podrán acceder a ellas, por ello es de acceso público.

Cada entorno tiene su propia base de datos, un SIMO que sirve estos datos y un PortusData desplegado que apunta a las URLs del web service de ese entorno.

Para gestionar los entornos se usa el sistema más sencillo posible, un archivo de configuración con las distintas variables que se necesitan. Este archivo de configuración tiene el nombre de "application.properties" y a parte de URLs, también contiene alguna configuración más, como la ruta en la que guardar los logs de la aplicación.

Para facilitar el cambio de entorno, se ha creado un archivo de configuración para cada uno de ellos:

- "application.properties.dev" para desarrollo.
- "application.properties.pre" para preproducción.
- "application.properties.pro" para producción.

Antes de compilar el proyecto habrá que renombrar de forma manual o automática el archivo correspondiente.

application.properties.pre

```
1 spring.thymeleaf.mode=LEGACYHTML5
2 spring.thymeleaf.cache: false
3 spring.devtools.restart.enabled=true
4 server.error.whitelabel.enabled=false
5 logging.file = /var/log/portusdata/portusdata.log
6 url.simo=https://premovil.puertos.es/simo
7 url.cma=https://premovil.puertos.es/cma2
```

Figura 3.8: Ejemplo de archivo de entorno con las URLs del web service de pre-producción.

3.5.5. Estructura del proyecto

El proyecto PortusData contiene tanto los archivos del backend, como los del frontend, al ser un proyecto sencillo con pocas clases no necesita una organización de archivos muy compleja.

La mayor parte de la estructura está adaptada a los directorios estándar que usan los proyectos Java que implementan el framework de Spring Boot.

Lo primero que destaca son dos directorios importantes:

- **src/main/java:**
  Contiene los archivos de configuración de variables de predicción en ambos idiomas.
  También tiene el directorio "es/puertos/PortusData" que contiene todas las clases Java.
  A lo largo del documento ya se ha explicado la función de todas estas clases, excepto la de una, "Resources.java", es un simple archivo con funciones auxiliares que son comunes para las otras clases, esto nos permite reducir bastante la duplicidad y complejidad del código en toda la aplicación.

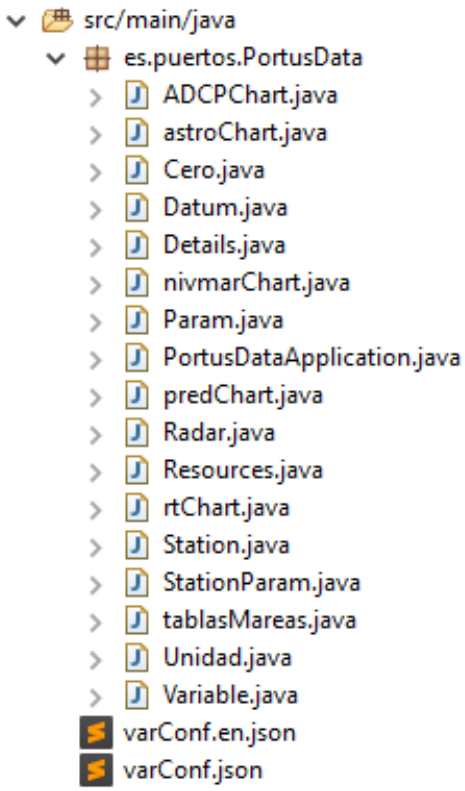

Figura 3.9: Las clases .java contienen toda la funcionalidad del backend.

- **src/main/resources:**

En esta carpeta lo primero que se aprecia son los diferentes archivos de configuración de los entornos, el resto de directorios contienen todo el frontend de la aplicación.
La carpeta "templates" es el directorio por defecto donde Thymeleaf buscará los archivos html para después modificarlos y finalmente servirlos. A parte de un template por cada tipo de chart, contiene el template "error.html" que se sirve cada vez que hay alguna URL mal formada.

Figura 3.10: Mensaje de error por llamar a un endpoint que no existe.

La carpeta "static" contiene el resto de archivos estáticos, tales como los .js, imágenes y otros recursos:

- **botones:**
  Contiene varias imágenes en formato .svg y .png que sirven para pintar los distintos botones de la interfaz.
- **codigo:**
  Es la carpeta más importante, tiene todos los archivos .js con toda la funcionalidad del frontend de la aplicación.
  El directorio "common" tiene recursos compartidos en todos los tipos de chart, como los controles y las distintas interfaces.
  En "main" se almacena la funcionalidad principal de cada tipo de chart, hay un archivo .js por cada uno de ellos.
  La carpeta "resources" contiene los archivos con los textos en diferentes idiomas y algunos archivos con funciones auxiliares que se han separado para hacer más fácil de entender el código en general del frontend.
- **css:**
  Es donde se encuentran todos los diferentes archivos de estilos .css que pueden importar las plantillas según sea necesario.
- **img:**
  Son otros recursos de imagen que no se corresponden con botones de la interfaz.
- **logos:**
  Contiene todos los posibles logos que luego se pintan en el fondo del gráfico, el nombre de estos archivos está preparado para coincidir con las IDs que inyecta el backend como parámetros a las plantillas de Thymeleaf.

- **resources:**
  Tiene las librerías principales en formato minimizado que implementa el frontend, todas en la última versión para la que han sido testeados los gráficos. También tiene otro tipo de recursos de menor importancia.
- **js:**
  Esta carpeta se genera cuando se compila el proyecto, tiene los mismos archivos que la carpeta "codigo" pero tras ser procesados por Babel [8]. Está incluida en el .gitignore ya que es necesario generarla siempre a partir de los fuentes.

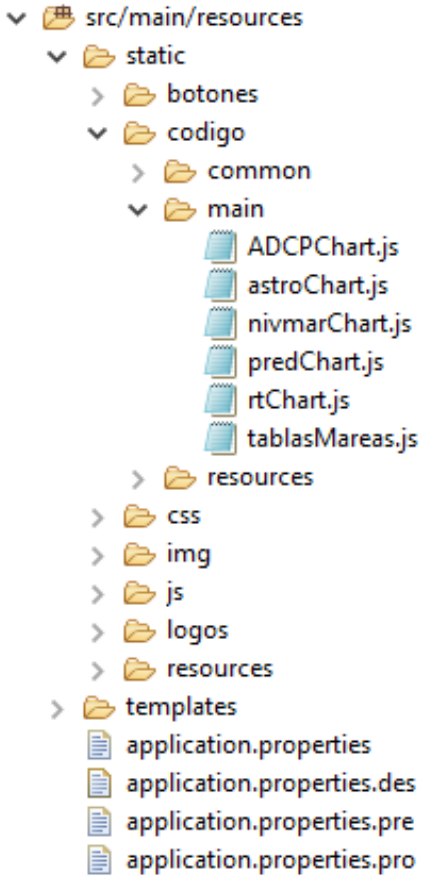

Figura 3.11: El mismo proyecto también almacena los archivos del frontend.

3.5.6. Desarrollo y montaje

Probar este proyecto mientras se está desarrollando es algo fácil gracias a integrar el framework de Spring Boot, que está preparado para poder arrancarlo sin tener que generar una build completa de la app.

Lo primero que se necesita es compilar por primera vez la aplicación usando el comando "maven compile" para que pueda generar los archivos del front necesarios, después simplemente hay que arrancar el proyecto como cualquier aplicación Java, siendo el punto de entrada la clase "PortusDataApplication".

Cuando se está en modo desarrollo, los cambios en las clases Java harán que el proyecto recargue y se puedan ver automáticamente, pero cualquier cambio en el frontend necesitará que se vuelva a ejecutar el comando de compilación.

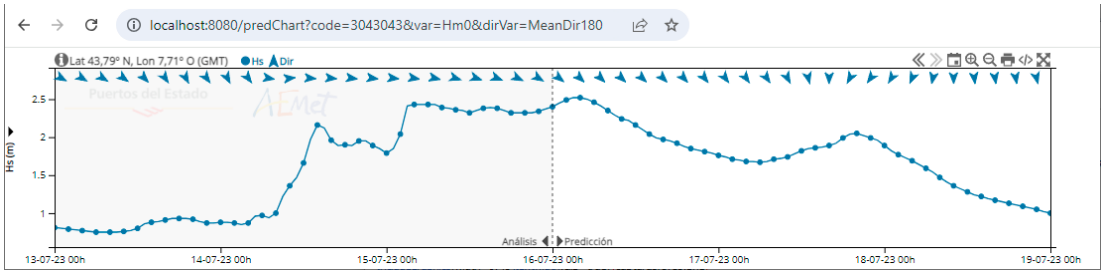

Figura 3.12: Charts arrancados en local en modo desarrollo.

Como ya se ha explicado en la parte de diseño, Maven es la librería encargada del proceso de compilación y montaje de la aplicación, pero en el proceso de compilación también interviene otra librería importante, llamada Babel [8].

Babel es una de las herramientas más utilizadas en el desarrollo web para transpilar código JavaScript. Su función principal es la de permitir utilizar las características más modernas del lenguaje a la hora de programar y luego esta librería se encargará de transformar el código a una versión que sea compatible con los navegadores más antiguos. También cuenta con una buena cantidad de plugins que pueden ampliar la funcionalidad de este proceso de transpilación.

En este proyecto se ha implementado de tal forma que antes de iniciar la compilación, desde Maven se llama a un archivo ejecutable incluido en el root del proyecto, este archivo se llama "babel-transpile.sh", aunque los charts siempre se compilan en una máquina Linux, se aporta también el "babel-transpile.bat"para que se pueda desarrollar en Windows si se desea.

Este ejecutable simplemente contiene un comando de Node que llama a la librería Babel con dos argumentos:

- El primero es la carpeta donde se encuentran los fuentes a transpilar.
- Por último se indica el directorio de salida de este proceso.

Una vez se ejecuta este comando, Babel leerá el archivo ".babelrc", también incluído en la raíz del proyecto, este archivo se encarga de configurar cómo se realizará el proceso de transpilación, hay dos versiones de este archivo:

- **.babelrcPRO**
  Es la versión para producción, incluye el preset "@babel/preset-env" que indica que tiene que transformar el código aportado en los argumentos a otro compatible con navegadores más antiguos.
  También usa el plugin "minify"que se encargará de reducir el tamaño del código todo lo posible y así mejorar los tiempos de transferencia de estos, optimizando todo el proceso de carga en general de los charts.
- **.babelrc** Es el archivo por defecto, se usa para desarrollo y preproducción, se diferencia del anterior en que este no integra el plugin "minify" ya que en estos entornos no es tan preocupante el tiempo de transferencia, al minimizar un archivo también se hace más complicado el seguimiento cuando se hace debug, por eso esto no se usa mientras se están haciendo pruebas de la aplicación.

Cuando se genera una compilación para producción habrá que renombrar el archivo ".babelrcPRO" a ".babelrc"

Una vez Babel ha terminado de hacer su trabajo, comienza la compilación de las clases Java, si hay algún error en el código, no se continuará y se indicará la parte del código que ha generado este problema.

Cuando ya tenemos todo compilado sin problemas, ya podemos arrancar la aplicación en local o comenzar el proceso de montaje.

Para generar la build del proyecto habrá que ejecutar el comando "maven package", su función es la de previamente mover y eliminar algunos directorios necesarios y por último empaquetar todo en una archivo .WAR que ya estará listo para ser desplegado en las máquinas del cliente.

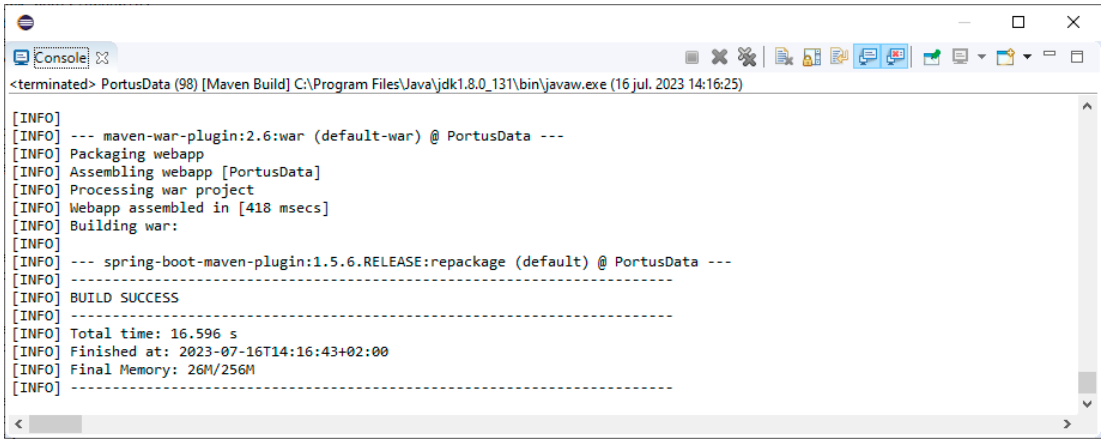

Figura 3.13: Output de la consola cuando el proceso de montaje ha sido exitoso.

3.5.7. Interfaz imagen

Uno de los requisitos más importantes para el cliente es que los charts puedan descargarse como imagen también, así pueden ser compartidos no solo en formato de aplicación interactiva, sino en cualquier sistema multimedia.

La primera idea fue hacer esto desde el frontend, una vez el chart estuviera cargado, se pondría un botón de descarga para exportarlo como imagen.

Para hacer esto se investigaron todas las librerías compatibles con JavaScript que ofrecían esta funcionalidad, pero se encontraban tres grandes problemas en ellas:

- No son compatibles con todos los navegadores.
- Funcionan bien para exportar código html, pero con svg no son fiables.
- Solo permiten descargar la imagen una vez el chart haya cargado.

Una vez reconocidos estos problemas, se concluye que esta funcionalidad no se puede cumplir desde el frontend, habrá que buscar otra alternativa.

La opción que finalmente se elige es usar algún software que nos permita abrir un navegador headless desde el servidor, cargar el gráfico y poder exportar toda la ventana como imagen. Para esto se investigan tres grandes librerías posibles:

- Selenium.
- Puppeteer.
- PhantomJS.

Estas tres opciones son herramientas enfocadas para la automatización de tareas de testing en aplicaciones web, pero en este caso les daremos otro uso ya que lo único que interesa es que puedan cargar el gráfico y devolver esta imagen a través del backend.

Esto da la ventaja de que no solo se puede tener el gráfico en formato imagen, sino que al servirla directamente del backend, se puede generar de forma dinámica siempre e incrustarla en correos electrónicos y otros sistemas multimedia que no aceptan código interactivo.

Selenium y Puppeteer son las herramientas más modernas y mejor mantenidas de las tres, pero en este caso vamos a usar phantomjs [5] porque esta librería ya se usa en otra de las aplicaciones del cliente y las máquinas están ya preparadas para usarla, aunque esté desactualizada y ya no se recomienda su uso. Tras varias pruebas, se concluye que sirve perfectamente para este cometido.

La implementación de esta tecnología se hace de la siguiente forma:

- Primero se llama a la URL de chart con el parámetro ´´int=img´´
- Cuando el backend de PortusData detecta esta interfaz, en vez de responder con la plantilla, se pone en espera.
- Se modifican algunos parámetros en la URL inicial: se vuelve a poner la interfaz default y se deshabilitan los botones y animaciones añadiendo "buttons=false" y "anim=false".
- Se ejecuta phantomjs desde el servidor, como argumentos se le mandan esta nueva URL y el path del script que se ejecutará, se encuentra en "src/main/resources/static/resources/phantom.js"
- Este script lo único que hace es cargar el chart con unas proporciones adecuadas y esperar a recibir una señal de ellos, si no la recibe, tras un timeout, responderá con un error.
- El chart cuando carga es capaz de detectar si está siendo ejecutado desde un navegador headless con phantomjs, si este es el caso, cuando termina de pintarse, manda una señal ejecutando la función "window.callPhantom()"
- El phantomjs detecta la señal mandada por el chart, saca una captura de toda la ventana y guarda el archivo de forma temporal, termina su ejecución.
- El backend de PortusData detecta que phantomjs ha terminado y sirve la imagen temporal creada.
- Finalmente, se borran los archivos temporales para evitar problemas de espacio en las máquinas.

Aunque este proceso parezca bastante largo, en la práctica suele tardar menos de un segundo, por lo que cumple a la perfección con los requerimientos de carga del cliente.

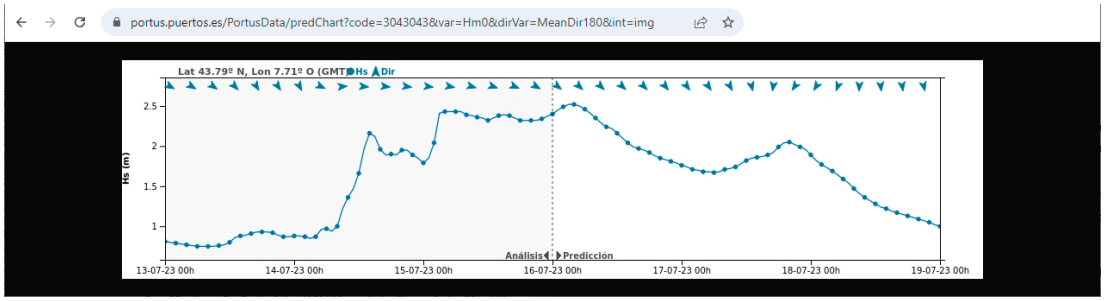

Figura 3.14: Chart servido como imagen por el backend.

4

Conclusiones

Este backend no necesita implementar ningún acceso a datos ni mucha seguridad, por lo que la parte más crítica es el rendimiento que se espera de él.

Como PortusData lleva funcionando varios años desde su primera versión, se ha podido comprobar el correcto funcionamiento del servidor hasta en picos de carga altos, que suelen ocurrir cuando hay temporales.

Gracias a las estructuras de datos elegidas para almacenar la metainformación, el procesado de las URLs y generación de plantillas es muy rápido, normalmente tarda menos de 80ms en responder, por lo que la mayor parte del tiempo de carga de un gráfico ocurre en el frontend cuando se hacen las llamdas a SIMO para rescatar las series temporales.

Cada día en Puertos del Estado se generan más de 100 reportes en PDF a la vez, cada uno de estos muestra varios charts usando la interfaz imagen. Esto hace que en cada día haya varios picos de carga en los que se tiene que llamar al phantomjs constantemente, y aunque esta sea ya una librería en desuso, rara vez se han detectado fallos en estos gráficos, por lo que se puede concluir que este sistema cumple con su objetivo perfectamente.

Actualmente hay cuatro versiones de esta aplicación desplegadas:

- Dos de ellas están en producción, con acceso público. Se encuentran en dos máquinas que se balancean para evitar caídas del servicio y aliviar la carga cuando es necesario.
- Otra se despliega en preproducción, sirve para poder testear correctamente

todo antes su puesta en producción.

- La última versión se encuentra en las máquinas de desarrollo, es usada por el cliente para probar los resultados de nuevos modelos de predicción o estaciones de medida.

4.1. Alcance final

La estructura de código común del proyecto hace que sea bastante fácil crear nuevos tipos de gráficos y por eso el proyecto ha ido creciendo hasta su estado actual. Gracias a todos los tipos de interfaz y modos de control que se han creado, han permitido que tenga un uso mucho más allá de la aplicación para la que fueron creados, podemos encontrarlos en varios proyectos de Puertos del Estado, tanto públicos como privados:

- **Portus** [1]**:** Es la aplicación principal para la que fueron pensados estos charts, sirven toda la información que se encuentra en los apartados de "Predicciones" y "Tiempo real". Cuenta con decenas de parámetros distintos, cientos de estaciones y miles de puntos de modelo, por lo que las distintas series temporales que pueden mostrar estos gráficos son prácticamente infinitas.
- **iMar** [9]**:** Es una app para Android e iOS que muestra los datos de Portus en una forma más manejable para dispositivos táctiles. En esta aplicación primero se presentan los charts en pequeño usando la interfaz de previsualización, pero permite expandirlo a pantalla completa, donde el usuario ya podrá interactuar con el chart de manera táctil.
- **CMA** [10]**:** En este caso es una aplicación de acceso privado a usuarios registrados. Se muestran prácticamente los mismos gráficos que en Portus, pero esta aplicación hace uso de los umbrales. También, gracias a la interfaz "img", permite insertar estos gráficos en los correos de alertas que se mandan.

Figura 4.1: Charts en CMA mostrando umbrales.

- **Reportes en PDF:** Gracias a poder devolver el gráfico en formato de imagen se pueden insertar en cualquier elemento multimedia, otro uso que se le da es para mostrar datos en reportes PDF que se generan diariamente de forma automática y se mandan a usuarios seleccionados.

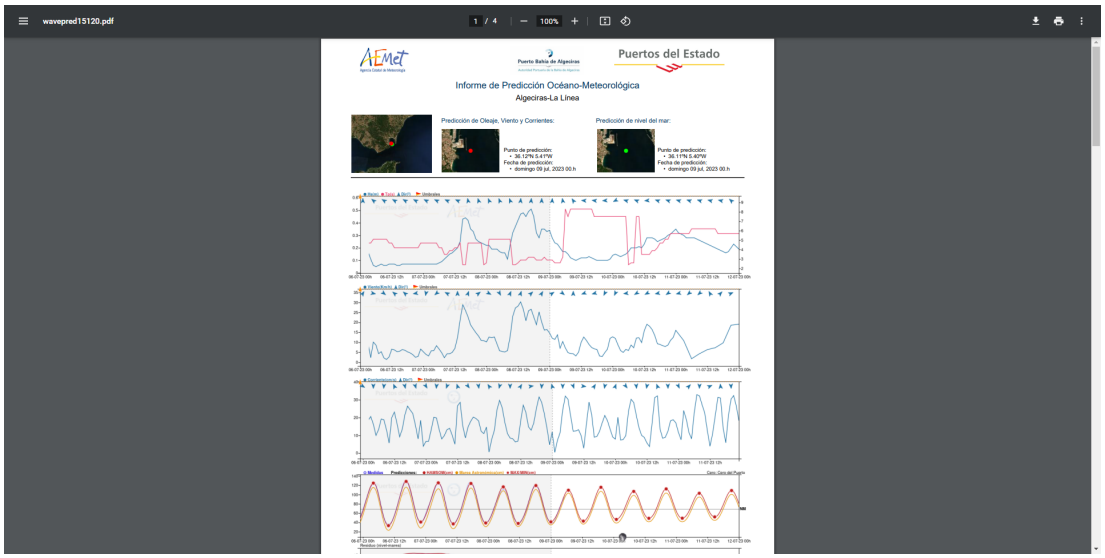

Figura 4.2: Charts integrados en reportes en PDF.

También se han usado en algunas aplicaciones o proyectos de menor importancia, gracias a que el cliente permite que los gráficos puedan ser compartidos como widgets por cualquier persona, se pueden encontrar estos charts en muchas páginas externas a Puertos del Estado, la mayoría de estas webs que comparten los gráficos están dedicadas al surf o tratan temáticas de meteorología.

Bibliografía

[1] P. del Estado, "Portus." [Online]. Available: https://portus.puertos.es/

[2] "Puertos del estado." [Online]. Available: https://www.puertos.es/

[3] "Spring." [Online]. Available: https://spring.io/

[4] "Google web toolkit." [Online]. Available: https://www.gwtproject.org/

[5] "Phantomjs." [Online]. Available: https://phantomjs.org/

[6] "Thymeleaf." [Online]. Available: https://www.thymeleaf.org/

[7] Maven. [Online]. Available: https://maven.apache.org/

[8] "Babel." [Online]. Available: https://babeljs.io/

[9] P. del Estado, "imar." [Online]. Available: https://play.google.com/store/apps/details?id=es.nologin.imar.android

[10] ——, "Cuadro de mando ambiental (cma)." [Online]. Available: https://cma.puertos.es/

Apéndices

A

Prueba PortusData

Como ya se ha indicado, PortusData es un proyecto público al que cualquier usuario puede tener acceso, la forma más fácil de ver el proyecto es accediendo a la aplicación Portus y navegar por las secciones de Predicción y Tiempo Real para seleccionar las distintas variables y puntos de modelo que pueden mostrar estos gráficos.

Para facilitar aún más el acceso, voy a recopilar una lista de URLs de acceso directo a cada uno de los tipos distintos de charts que existen y que muestren las características más importantes del proyecto.

Los gráficos están configurados para ocupar el máximo ancho y alto de su contenedor, la aplicación que los incluye es la que maneja el tamaño y relación de aspecto de estos contenedores para lograr la visualización más óptima posible, en este caso, como accedemos directamente por URL, el contenedor es el propio navegador, por lo que se recomienda cambiar el tamaño de la ventana a uno en el que el ancho sea el doble del alto.

Tanto los datos de predicciones como los de tiempo real se renuevan cada día, por lo que es posible que alguno de los modelos falle o una estación se estropee y alguna de las URLs que se presentarán a continuación falle debido a que no tenga datos disponibles en el momento de acceso.

Charts de Predicción

Todas las variables posibles de oleaje combinadas en un solo gráfico: `https://portus.puertos.es/PortusData/predChart?code=3120048&var=Hm0,Tp,Tm02&dirVar=MeanDir180`

Comparación del modelo de predicción de viento con los datos medidos de la estación más cercana: `https://portus.puertos.es/PortusData/predChart?code=2056079&station=2548&var=WindSpeed&dirVar=WindDir180`

Charts de Tiempo Real

Medidas de oleaje de una estación: `https://portus.puertos.es/PortusData/rtChart?`

station=2630&params=Hm0,Hmax&dirParams=MeanDir,MeanDirPeak

Observaciones de Nivel del Mar con cadencia 1 minuto: https://portus.puertos.es/PortusData/rtChart?station=3210&params=SeaLevel

Charts de Nivel del Mar

Los distintos modelos se van añadiendo a lo largo del día según se ejecutan, los modelos con bandas de confianza suelen estar disponibles a partir de las 14:00 cada día.

Nivel del Mar en el Mediterráneo: https://portus.puertos.es/PortusData/nivmarChart?code=16210&station=3656&var=SeaLevel,SeaSea,Residual

Nivel del Mar en el Atlántico: https://portus.puertos.es/PortusData/nivmarChart?code=11210&station=3108&var=SeaLevel,SeaSea,Residual

Charts y tablas de Mareas

Chart de mareas: https://portus.puertos.es/PortusData/astroChart?code=3219

Tablas de mareas: https://portus.puertos.es/PortusData/tablasMareas?code=3219

Charts de Perfiladores

Actualmente todos los perfiladores que hay están sin transmisión o tienen algún fallo en los datos, solo se puede acceder a uno de ellos y para ver datos buenos hay que seleccionar el calendario y cambiar las fechas a alguna pasada, por ejemplo, en Abril de 2023 los datos son correctos.

Perfilador de Cádiz: https://portus.puertos.es/PortusData/ADCPChart.html?station=5342&params=CurrentSpeed

Interfaces

Para una visualización óptima de las interfaces mínima y de previsualización, se recomienda un tamaño de ventana pequeño.

Interfaz default: https://portus.puertos.es/PortusData/predChart?code=3043043&var=Hm0&dirVar=MeanDir180&int=default

Interfaz minima: https://portus.puertos.es/PortusData/predChart?code=3043043&var=Hm0&dirVar=MeanDir180&int=min

Interdaz de previsualización:https://portus.puertos.es/PortusData/predChart?code=3043043&var=Hm0&dirVar=MeanDir180&int=prev

Interfaz imagen: https://portus.puertos.es/PortusData/predChart?code=3043043&var=Hm0&dirVar=MeanDir180&int=img

B

Instalar proyecto en local

Para poder arrancar el proyecto en local lo primero que se necesita es tener Java 8 instalado en el sistema. También será necesario tener NodeJS instalado y añadir la librería Babel de forma global:

"npm i -g @babel/cli @babel/core"

Por último, si se quiere probar el exportar gráficos a imagen, se tiene que tener instalado phantomjs y asegurarse de que los binarios están incluidos en el PATH.

Si se está haciendo esto desde un Linux es necesario modificar la línea 95 del archivo "pom.xml" para cambiar la extensión del script a ".sh"

El entorno de desarrollo usado es Eclipse, pero no debería haber problema en usar otros.

El primer paso es importar el directorio entero como proyecto, detectará automáticamente que es un proyecto Maven y comenzará a instalar las librerías necesarias.

Para que funcione el frontend es necesario compilar primero, para ello se hace click derecho en el proyecto - Run As.. - Maven Build.. Se abrirá un popup con varias opciones, en el campo Goals se pone "compile" y se pulsa en "Run"

Si todo ha salido bien ya se puede arrancar, para ello simplemente hay que hacerlo como cualquier aplicación Java.

Click derecho en el proyecto - Run As.. - Java Application

Se abrirá un nuevo popup para seleccionar la clase de entrada, hay que buscar "PortusDataApplication" y darle al OK, con esto ya debería inicializarse.

Para empaquetar la aplicación y generar el WAR hay que volver a ejecutar un comando Maven:

Click derecho en el proyecto - Run As.. - Maven Build.. Y en este caso se pone "package"

como goal.

Si todo ha salido bien, el .war se encontrará en la carpeta “target“.