[PORTADA]

Índice

[1. Resumen 4](#_Toc47429055)

[2. Introducción 4](#_Toc47429056)

[3. Gestión del proyecto 4](#_Toc47429057)

[3.1. Alcance 4](#_Toc47429058)

[3.2. Objetivos 7](#_Toc47429059)

[3.3. Planificación 7](#_Toc47429060)

[3.4. Metodología 11](#_Toc47429061)

[3.5. Herramientas 12](#_Toc47429062)

[3.6. Relación con las asignaturas del Grado 13](#_Toc47429063)

[4. Realización del proyecto 13](#_Toc47429064)

[4.1. Backend 13](#_Toc47429065)

[4.1.1. Análisis 13](#_Toc47429066)

[4.1.2. Diseño 14](#_Toc47429067)

[4.1.3. Implementación 14](#_Toc47429068)

[4.1.4. Pruebas 17](#_Toc47429069)

[4.2. Frontend 18](#_Toc47429070)

[4.2.1. 18](#_Toc47429071)

[4.2.2. Diseño 18](#_Toc47429072)

[4.2.3. Implementación 18](#_Toc47429073)

[4.2.4. Pruebas 19](#_Toc47429074)

[5. Conclusión 19](#_Toc47429075)

[I. Anexo I – Glosario de términos. 19](#_Toc47429076)

[II. Anexo II – Control de cambios. 19](#_Toc47429077)

# Resumen

Este proyecto se centra en la extracción de datos de varias fuentes para almacenar y mostrar la información de diferentes maneras. Para ello, se realizará una aplicación con arquitectura desacoplada diferenciando entre Front-end y Back-end.

Las características más importantes de este proyecto son los distintos Scrappers personalizados que han de extraer los datos de las distintas páginas web junto a él Front-end, que deberá mostrar los datos de forma gráfica.

A lo largo de este documento se detallan las distintas tecnologías usadas para la realización del proyecto, de la cual se destaca Svelte, un framework web muy joven con buenas cualidades que puede llegar a dar mucho rendimiento.

Además, en este documento se encuentra toda la información acerca del alcance y la planificación del proyecto, las metodologías que se usan durante el desarrollo, se detalla la implementación del front-end y back-end por separado y se explican todos los problemas encontrados.

# Introducción

Se quiere conseguir comparar series de datos de distintas fuentes, tanto de forma visual como los datos numéricos. No obstante, este objetivo se puede fragmentar en distintos apartados, todos ellos centrados en las series de datos: Extracción y almacenamiento, recuperación y entrega de datos, y, por último, visualización y exportación. Estos tres apartados se corresponderían, respectivamente, a los Scrappers, el back-end y el front-end.

Por lo tanto, para llegar a la solución, se ha de conseguir una aplicación que cumpla con los tres apartados anteriores. Además, al optar por una arquitectura desacoplada, el back-end puede ser usado como una API con la que aplicaciones de terceros puedan interactuar.

La finalidad es tener un sistema completo que extraiga toda la información relevante y la muestre en el mismo formato independientemente de la fuente. Como se va a extraer información de distintas fuentes, es probable que los datos vengan en distintos formatos y/o estructurados de distinta forma, o que directamente falte información. Por ello, al recolectar la información hay que realizar un tratamiento para almacenar los datos con la misma estructura.

# Gestión del proyecto

## Alcance

Objetivo del proyecto

El objetivo del proyecto es desarrollar una aplicación que permita comparar series de datos.

La aplicación contará con distintos Scrappers que se encarguen de extraer y almacenar las series de datos de las distintas fuentes especificadas por el cliente. Una vez los datos están almacenados, la aplicación deberá mandar las series de datos al front-end. Allí, los usuarios tendrán a su disposición el conjunto de series provenientes de distintas fuentes y serán capaces de elegir cuales comparar tanto gráficamente como numéricamente. Además, los usuarios tendrán la opción de exportar las comparaciones que realicen con solo pulsar un botón.

Restricciones

A continuación, se definen las restricciones del proyecto a desarrollar, estas restricciones limitan:

* Tiempo: es imprescindible que se cumpla la fecha de entrega.
* Calidad: el producto desarrollado tiene que cumplir con un mínimo de calidad que satisfaga tanto al cliente como al tutor de este.
* Riesgos: los riesgos contemplados en el proyecto deben tenerse en cuenta para su correcto desarrollo.
* Costes: al ser un proyecto sin presupuesto inicial, las herramientas y tecnologías usadas han de ser gratuitas. El único gasto que se contempla es el del hosting y dominio si se desplegase la aplicación y no fuese un coste alto.

Riesgos

Lista de riesgos posibles durante la vida del proyecto, cada riesgo ha de tenerse en cuenta para evitar contratiempos:

* Planificación: si no se estima y planifica adecuadamente, es probable que no se cumplan los objetivos para la fecha acordada.
* Tecnología: al usar un framework desconocido, es posible que el proyecto no se pueda desarrollar acorde a los requisitos establecidos o que las tareas de investigación se alarguen en exceso.
* COVID-19: Si se contrae el virus esto provocaría inevitablemente un retraso en el proyecto como mínimo.
* Pérdida de datos: Existe la posibilidad de que se pierda información debido al malfuncionamiento de algún software o el fallo en el disco duro. Además, incidentes naturales o de caída de energía pueden provocar la pérdida de datos.

|  |  |  |  |
| --- | --- | --- | --- |
| Riesgo | Impacto | Probabilidad | Nivel de riesgo |
| Planificación | 4 | 4 | 8 |
| Tecnología | 3 | 7 | 10 |
| COVID-19 | 10 | 1 | 11 |
| Pérdida de datos | 10 | 2 | 12 |

Medidas de control correspondientes:

* Planificación: se realizará una planificación pesimista, dejando margen suficiente a las tareas hasta adquirir la capacidad de realizar estimaciones más aproximadas.
* Tecnología: Se realizarán tareas de investigación previas al desarrollo con las tecnologías desconocidas.
* COVID-19: Se tomarán las medidas de seguridad propuestas por el estado, minimizando así la posibilidad de contraer el virus.
* Pérdida de datos: se usará un repositorio privado en GitHub para almacenar todos los documentos y proyectos del producto, actualizándose en cada incremento. Esto reduce el impacto al mínimo, ya que únicamente se podrá perder las modificaciones realizadas al ocurrir la incidencia.

Criterios de aceptación

El proyecto será aceptado únicamente si se cumplen los siguientes criterios de aceptación:

* Cumple los requisitos establecidos.
* Es entregado dentro del plazo correspondiente.
* Se extraen datos desde dos fuentes como mínimo.

## Objetivos

El Trabajo de fin de grado tiene como objetivo proporcionar al alumno una visión general y completa de la planificación, gestión y metodologías aplicables a un proyecto informático. Debe ser un trabajo con una complejidad considerable, al tener una carga de 12 créditos ECTS. Por lo tanto, uno de los objetivos principales de este proyecto es desarrollar un software completo de principio a fin, resumiendo todo lo aprendido a lo largo del grado.

Al ser un proyecto individual, entre los objetivos también se incluyen las metas personales del alumno, destacando un mayor interés por aprender tecnologías nuevas desconocidas por el mismo. Esto se tendrá en cuenta a la hora de elegir las tecnologías para el producto, a pesar de que usar tecnologías más novedosas puede acarrear inconvenientes.

Por último, pero no menos importante, este es un trabajo desarrollado para un cliente, el cual quiere un producto útil y de una calidad mínima. Por tanto, es importante desarrollar un buen producto con el que impresionar al cliente, ya que puede llegar a ser un factor determinante para el comienzo de la vida laboral del alumno.

Al ser la última asignatura del grado, esta asignatura, junto con las prácticas, son el enlace entre la vida universitaria y la laboral, por lo que elaborar un producto de calidad puede ser un factor determinante a la hora de encontrar un buen puesto y dar a conocer las habilidades del alumno.

## Planificación

A pesar de no seguir una metodología ágil, el desarrollo de este proyecto se divide en tres fases principales. Cada una de las fases tiene un propósito distinto y está bien definida en el tiempo. En la primera fase se realizarán las tareas de investigación, el propósito de esta fase es determinar los frameworks y herramientas que se van a usar (especialmente en el frontend) y adquirir una base para poder comenzar a trabajar con ellos. El objetivo de la segunda fase es construir un prototipo con las funcionalidades clave, este prototipo debe ser completo para poder hacer una demostración al cliente y recibir buen feedback. El objetivo de la tercera y útima fase es completar el producto, para ello se tendrá en cuenta el feedback recibido al final de la segunda fase. Además, se implementarán las funcionalidades extras y se mejorará tanto el apartado visual del producto como la API.

Primera fase: Inicio

Como se indica en el párrafo anterior, la finalidad de esta fase es determinar las tecnologías que se van a usar y adquirir cierto conocimiento básico en aquellas que sean desconocidas. Para ello se ha de realizar un análisis de las herramientas y framework propuestos y valorar cuáles son las mejores opciones. En concreto, estas son las alternativas:

Base de datos: Oracle (sql) / MongoDB (nosql).

Frontend: Svelte / React.

Backend: Spring Boot / Otra.

Segunda fase: Prototipo

El objetivo de esta fase es implementar las funcionalidades principales. Para ello, se creará un scrapper para una de las fuentes de datos, INDEA, y se implementará toda la funcionalidad desde el almacenamiento en base de datos hasta la visualización de las series de datos en el frontend.

Tercera fase: Finalización

A lo largo de esta fase, se completará el producto con las tareas detectadas en el feedback obtenido del cliente, además de cualquier funcionalidad extra propuesta. En el caso de que no diese tiempo a implementar alguna funcionalidad en la fase anterior, esta pasaría como tarea de esta fase con una prioridad superior a las demás.

Tareas

A continuación, se encuentra el listado de tareas. Aquí se definen los tiempos de inicio y finalización de las fases, las descripciones de las tareas y sus prioridades. Las tareas que se encuentran en este backlog son tareas generales que se dividirán en subtareas antes de ser desarrolladas.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Primera fase: Inicio** | | | | |
| **Fecha inicio: 01/07/2020** | | | **Fecha fin: 07/07/2020** | |
| **TI.01** | Determinar herramientas y frameworks a usar. | | | A |
| **TI.02** | Obtener conocimientos básicos sobre las herramientas elegidas desconocidas. | | | A |
| * Svelte | | | | 3h |
| * MongoDB | | | | 1h |
| **Segunda fase: Prototipo** | | | | |
| **Fecha Inicio: 08/07/2020** | | | **Fecha fin: 28/07/2020** | |
| **TP.01** | Inicialización del proyecto backend | | | A |
| * Crear proyecto base con Spring initializr. | | | | 2h |
| * Instalar dependencias (POM). | | | | 1h |
| **TP.02** | Tabla de series. | | | A |
| * Componente para mostrar las series en una tabla en Svelte. | | | | 5h |
| * Paginación en la tabla. | | | | 5h |
| * Endpoint con paginación en la API. | | | | 1h |
| **TP.03** | Almacenamiento de datos del sistema sobre series y sus datos en MongoDB. | | | A |
| * Dominio java para Serie con la siguiente información: código, periodo, fuente y colección de datos. | | | | 2h |
| * Dominio java para Datos con la siguiente información: momento y valor. | | | | 2h |
| * Repositorios y servicios para Series y Datos | | | | 2h |
| * BDD MongoDB y conectar con los repositorios anteriores | | | | 2h |
| **TP.04** | Scrapper INDEA | | | A |
| * Scrapper que recolecte series de INDEA e inserte las series en el sistema. | | | | 8h |
| **TP.05** | API backend | | | A |
| * Endpoints básicos para enviar las series de datos al frontend. | | | | 3h |
| * CORS. | | | | 2h |
| **TP.06** | Visualización de las series en forma de tabla. | | | A |
| * Componente tabla de datos Svelte. | | | | 5h |
| **TP.07** | Implementación de e-chart para la visualización de series. | | | A |
| * Componente e-chart en Svelte. | | | | 5h |
| * Visualización de varias series en el mismo gráfico. | | | | 5h |
| **TP.08** | Calcular la tasa de variación y visualizarla. | | | M |
| * Calcular tasa de variación en el backend. | | | | 2h |
| * Serie DTO para enviar la tasa con las series al frontend | | | | 3h |
| **TP.09** | Botón de descarga del gráfico. | | | M |
| * Botón para descargar el gráfico que se esté visualizando en el momento. | | | | 2h |
| **TP.10** | Mejorar apartado visual. | | | M |
| * Añadir estilo al frontend | | | | 6h |
| **TP.11** | Filtrado de series | | | A |
| * Formulario de búsqueda que debe incluir los siguientes criterios: palabra clave, fuente, origen, tipo de datos y periodicidad | | | | 10h |
| * Endpoint en la API que devuelva un listado de series filtrado según los criterios descritos anteriormente. | | | | 3h |
| **TP.12** | Scrapper MINECO | | | A |
| * Scrapper que recolecte series del Ministerio de economía e inserte las series en el sistema. | | | | 6h |
| * Datos de las fuentes coherentes y mostrados correctamente. | | | | 3h |
| Tercera fase: Finalización | | | | |
| Fecha Inicio: 29/07/2020 | | Fecha fin: 29/08/2020 | | |
| **TF.01** | Scrapper INE, EUROSTAT. | | | A |
| * Scrapper para fuente: INE. | | | | 6h |
| * Scrapper para fuente: EUROSTAT. | | | | 6h |
| **TF.02** | Mejorar frontend – tareas extra frontend | | | B |
| * Menú principal. | | | | 5h |
| * Mejorar estilo, apartado visual del sistema. | | | | 5h |
| * Control de errores. | | | | 3h |
| **TF.03** | Mejorar backend – tareas extra backend | | | B |
| * Control de excepciones. | | | | 8h |
| * Documentación Swagger. | | | | 3h |
| **TF.04** | Avisos del sistema | | | M |
| * Aviso (pop up) al visualizar series con unidades o periodicidad distinta. | | | | 4h |

Prioridades: A – Alta, M – Media, B - Baja

Costes

El coste más importante y el único coste directo que presenta este proyecto es el salario, como este proyecto se está desarrollando en una asignatura, para el cálculo del sueldo se han usado las 300 horas determinadas por la asignatura. Para este cálculo, también se ha usado [Glassdoor](https://www.glassdoor.es/Sueldos/sevilla-programador-junior-sueldo-SRCH_IL.0,7_IM1014_KO8,26.htm) para determinar el sueldo anual de un programador Junior en Sevilla.

La fórmula usada para calcular el coste es:

->

S: Salario anual. H: Horas de la asignatura. JA: Jornada anual

Además del salario, se tiene en cuenta el coste de amortización del equipo usado durante el desarrollo. Se ha calculado aplicando un porcentaje de pérdida de valor anual del 20% al precio original sin impuestos del equipo, lo que supone la siguiente fórmula:

->

Por lo que el coste total del proyecto seria **2.871,68 €.**

## Metodología

Existen varias metodologías que se pueden usar a la hora de desarrollar un proyecto software. Estas metodologías se suelen separar entre lineales e iterativos. Los modelos lineales, llamados también modelos en cascada, se caracterizan por ser secuenciales, es decir, el proyecto pasa por las siguientes fases de manera sistemática: análisis, diseño, desarrollo, pruebas, mantenimiento. Por el contrario, los modelos iterativos se caracterizan por fragmentar los requisitos en distintos grupos y realizar una iteración para implementar cada uno de dichos grupos. Es decir, un desarrollo iterativo es similar a realizar varios desarrollos en cascada con menos carga de trabajo.

Por otro lado, el backend y el frontend van a ser desarrollados de manera individual, lo que se conoce como arquitectura desacoplada. Posiblemente, la ventaja principal de esta arquitectura es la fluidez que adquiere el sistema, ya que el intercambio de información entre backend y frontend es extremadamente ligero, en cada petición y respuesta entre el cliente y servidor solo se intercambian archivos *Json*. Además, el backend puede ser utilizado por otras interfaces web o aplicaciones, lo que añade valor al producto.

Imagen que contiene tabla

Descripción generada automáticamente

## Herramientas

Github

Github es una plataforma de desarrollo colaborativo en la que se pueden almacenar proyectos usando Git. En este proyecto, a pesar de que no sea colaborativo al ser desarrollado por una persona, se va a usar Github para el control de versiones y la gestión de las tareas.

Además, al usar Github para almacenar el proyecto y para la gestión de tareas, se van a usar funcionalidades más avanzadas, como asociar cada subida de código o de archivos a una tarea concreta.

Herramientas de desarrollo

Se van a usar dos entornos de desarrollo distintos: IntelliJ para el backend y Visual Studio Code para el frontend.

IntelliJ es un buen IDE que ha sido seleccionado entre otros, como por ejemplo Eclipse, debido a que ofrece muchas funcionalidades útiles para desarrollar en Spring Boot. En concreto, se puede conectar directamente con Spring Initializr, lo cual es muy eficiente a la hora de crear proyectos nuevos. Además, es un IDE usado habitualmente por el desarrollador del proyecto, por lo que no hay alternativa mejor.

Se ha elegido Visual Studio Code para el frontend por ser usado habitualmente por el desarrollador del proyecto, al igual que IntelliJ, además de ser bastante popular para el desarrollo web.

Spring Boot

Spring Boot es un framework de desarrollo de proyectos en Java muy conocido tanto por la comunidad de desarrolladores como por el desarrollador del proyecto. Al ser tan conocido, se minimiza el riesgo del desarrollo del backend, y permite que se comience a trabajar lo antes posible.

Svelte

Respecto al frontend, la tecnología decidida finalmente para el desarrollo ha sido Svelte. Este es un framework de interfaces de usuario muy reciente que tiene varias características interesantes. Por el contrario, aunque ofrece ventajas sobre otros frameworks gracias a la reactividad o que se necesita escribir menos código, al ser nuevo, hay muchísima menos documentación, dudas resueltas y componentes ya hechos en internet.

MongoDB

Para la base de datos se va a usar un sistema NoSQL, es decir, en vez de guardar los datos en tablas, como se hace tradicionalmente, se guardan en un esquema dinámico sin relaciones. MongoDB es capaz de devolver los datos de manera más fácil y rápida que las bases de datos relacionales, por lo que, al no tener un sistema complejo de relaciones, es el sistema más adecuado para este proyecto.

## Relación con las asignaturas del Grado

DP: Spring, aplicaciones web.

ISPP: Desarrollo de producto completo.

PGPI: Documentación.

CBD: Bases de datos, MongoDB nosql

AII: Scrapper

SSII: certificados

IR: Elaboración de requisitos

Practicas?, …

# Realización del proyecto

## Backend

### Análisis

Requisitos de información

RI.01 - Se quiere guardar la siguiente información acerca de cada serie:

* Nombre
* Periodo (fecha de inicio y fin de datos disponibles)
* Fuente (IECA/INE/EUROSTAR)
* Periodicidad (Mensual/Trimestral/Anual)

En el caso de encontrar más información útil mediante los scrappers, se podrán añadir campos adicionales.

RI.02 - Para cada serie, hay que guardar los datos correspondientes a cada momento en el tiempo, los cuales constan de:

* Año
* Valor
* Periodo

Requisitos funcionales

RFB.01 – El backend deberá poder conectarse con la base de datos para almacenar, modificar o eliminar los datos.

RFB.02 – El backend deberá disponer de scrappers para las distintas fuentes especificadas para poder extraer los datos e insertarlos en la base de datos.

RFB.03 – El backend deberá de disponer de una API con la que poder comunicarse con el frontend. Esta API debe contener suficientes endpoints para las distintas operaciones que se realizan desde el frontend.

### Diseño

### Implementación

Scrapper IECA

El Instituto de Estadística y Cartografía de Andalucía (IECA) tiene una web en la que permite seleccionar series para ser descargadas en distintos formatos (Excel, CSV, Json) o guardar la selección para usarla más adelante.

El primer acercamiento para crear el Scrapper fue intentar crear un Crawler que recorriese las paginas de las series de datos extrayendo los datos del propio código fuente. Pero esto resultó imposible ya que la página esta construida con muchas funciones JavaScript y los datos no aparecen en el código fuente.

Afortunadamente, la página permite guardar una cesta con mucha información sobre las series seleccionadas. Finalmente, el Scrapper funciona recorriendo esta cesta y realizando peticiones a la web con los datos de la propia cesta. A continuación, se expone un ejemplo básico:

En la información de la cesta de la Serie X se indica que su código es: 1234, por lo que para acceder a sus datos se construye la URL: [www.ieca.es/obtenerSerie?cod=1234](http://www.ieca.es/obtenerSerie?cod=1234)

Así, tras recorrer la cesta en la que se incluyen todas las series disponibles, tenemos los datos de todas las series de esta fuente.

Scrapper MINECO

Con esta segunda fuente ocurre como con la primera, a pesar de intentar recorrer las páginas de datos con un Crawler estas páginas no existen realmente, sino que son mostradas con funciones JavaScript, lo que hace imposible recorrerlas de manera automática.

La solución encontrada finalmente es parecida a la de IECA. Se descargan los datos completos en un directorio zip. Esta carpeta se recorre y se extraen todos los datos de cada serie.

Se han encontrado distintas dificultades respecto a la fuente anterior. La primera viene dada por el formato de los datos. Al contrario que IECA, que permite la descarga en distintos formatos conocidos, el ministerio de economía tiene su propio formato y solo permite la descarga de todas las series en ese formato, lo que ha provocado un esfuerzo adicional a la hora de extraer los datos y estructurarlos a nuestra manera. Por otra parte, la cantidad de series es muy elevada (22.000 aprox), lo que provoca una gran caída del rendimiento al extraer todos los ficheros de datos. Este último problema se solventó adaptando el algoritmo del Scrapper para que extraiga la información de los ficheros individualmente.

MongoDB

La implementación de la base de datos es importante a pesar de su sencillez. El primer paso es incluir las dependencias de MongoDB en el POM de la aplicación. Gracias a Spring Initializr, este paso se realizó automáticamente en la creación de la aplicación base.

El siguiente paso es adaptar las clases del dominio a MongoDB, lo que también es sencillo, únicamente tendremos que usar la anotación @Document(collection = "examples") para especificar un nombre para la colección. Sin embargo, este paso se puede omitir, lo que provocará que el nombre de la colección sea el mismo del de la clase en cuestión. Además, queremos que MongoDB guarde un identificado ID único para cada objeto, por lo que hay que especificar ese atributo en la clase y anotarlo con @Id, mongoDB se encargará de gestionar el valor para este atributo internamente.

Otra característica de MongoDB en Spring Boot que hay que tener en cuenta es que no hace falta la existencia previa de la base de datos, la misma aplicación la generará en el caso de que no la encuentre. Por defecto, el nombre de la base de datos autogenerada es “test”, pero se puede modificar especificando un nombre en las propiedades de la aplicación mediante el siguiente parámetro:

*spring.data.mongodb.database=databasename*

Sin embargo, este no es el único parámetro que se ha tenido que especificar, para que los índices especificados en el dominio se generen en la base de datos hay que añadir este otro parámetro:

spring.data.mongodb.auto-index-creation = true

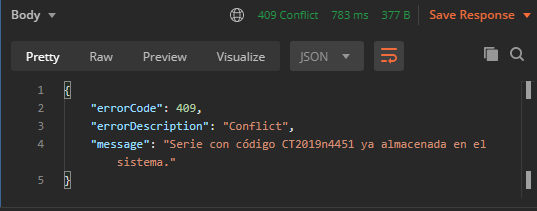
Esta opción permite añadir índices de MongoDB desde el dominio java, en concreto, se ha usado para añadir la restricción de unicidad a los códigos de las series. Esta restricción va a asegurar que no haya series duplicadas en el sistema, si se intentase insertar una serie con un mismo código, MongoDB genera una excepción tipo *“DuplicateKeyException”*. A continuación, se muestra el resultado de dicha excepción junto con el manejo de excepciones explicado más adelante:

Ilustración : Excepción por código duplicado.

API

Al tener una aplicación con arquitectura desacoplada, es necesario que el frontend y el backend se comuniquen mediante algún mecanismo. Para conseguir este objetivo, el backend se ha realizado en formato API, es decir, desde el punto de vista del frontend, va a haber una serie de endpoints disponibles con los que obtener o enviar datos al backend. Esto es muy fácil de realizar en Spring Boot, ya que únicamente hace falta poner las siguientes anotaciones a los controladores que vayan a formar parte de la API:

*@RestController*: Identifica al controlador como parte de la API.

*@RequestMapping*: Indica la ruta hacia el controlador. p.e: (@RequestMapping(“/serie”).

*@GetMapping*: Se colocan sobre los métodos del controlador para especificar el tipo de solicitud HTTP del endpoint y especificar la ruta si es necesario. Existen varios tipos de solicitud HTTP, los más usados son Get, Post, Put y Delete. p.e: (@GetMapping("/{id}")).

Una vez configurada la aplicación con estas anotaciones ya está lista para interactuar con el frontend. Pero para que las dos aplicaciones se puedan comunicar es necesario habilitar el CORS desde el backend hacia el frontend, ya que si no las peticiones serán bloqueadas por este mecanismo.

Hay distintas alternativas para habilitar el CORS, la más simple es especificar mediante la anotación *@CrossOrigin*, la cual se puede colocar en los métodos del controlador o en todo el controlador para aplicar los ajustes a todos los métodos a la vez. No obstante, se ha optado por aplicar una configuración global para toda la aplicación. Esto se ha conseguido modificando la clase de configuración de la aplicación, añadiéndole la siguiente línea de código:

*registry.addMapping("/serie/\*\*").allowedOrigins("http://localhost:5000");*

Esto permite el acceso desde localhost:5000 al recurso serie y todos sus endpoints.

Control de excepciones

Para que los errores que ocurran durante la ejecución de la aplicación estén controlados y los mensajes de error sean comprensibles se ha hecho uso del manejo de excepciones que ofrece Spring Boot, además de crear excepciones propias de la aplicación y mensajes de error personalizados.

Sin embargo, hay que especificar que excepciones queremos controlar a Spring, esto se puede hacer a nivel de controlador, pero como ocurre también con el CORS, es más interesante solucionar este problema a nivel de aplicación, esto se consigue especificando todas las excepciones en una clase aparte marcada con la anotación *@ControllerAdvice.*

En la clase de configuración hay que crear un método para cada excepción que se quiera tratar y marcar el método con la siguiente anotación:

*@ExceptionHandler({ ExampleException.class})*

Esto permite capturar la excepción y tratarla de forma individual, así se puede especificar el código de error HTTP correspondiente a la excepción y crear una respuesta más explicativa.

Captura de pantalla de un celular

Descripción generada automáticamenteA continuación, se pueden observar las diferencias entre controlar una excepción o no, en concreto, este caso es el error provocado por intentar obtener una serie inexistente:

Ilustración : Excepción no controlada

Como se puede observar en la imagen, el error es poco descriptivo y el estatus es “*500 Internal Server Error”*, cuando debería ser “*404 Not Found”* debido a que no es un error del sistema, si no que no se ha encontrado el recurso solicitado.

A continuación, se muestra el comportamiento de la aplicación con el manejo de excepciones implementado, se puede observar que el mensaje es mucho más explicativo y el status mostrado es el correcto.

Captura de pantalla de un celular

Descripción generada automáticamente

Ilustración : Excepción controlada

Problemas encontrados

Certificado

### Pruebas

Test unitarios

## Frontend

Visualización de las series:

Los datos de las series se deben de poder visualizar en forma de tabla (¿) y de gráfica. Además, se deben poder combinar distintas series en la misma gráfica con el fin de poder compararlas.

### Análisis

Requisitos funcionales

RF.01 – El sistema permitirá al usuario listar las series almacenadas.

RF.02 – El sistema permitirá al usuario filtrar la lista de series.

RF.03 – El sistema permitirá al usuario mostrar una o varías series en un gráfico.

RF.04 – El sistema permitirá al usuario mostrar una o varias series en una tabla.

RF.05 – El sistema permitirá al usuario descargar la gráfica que esté visualizando.

RF.06 –El sistema permitirá al usuario descargar la tabla que este visualizando.

Visualización de las series:

Los datos de las series se deben de poder visualizar en forma de tabla (¿) y de gráfica. Además, se deben poder combinar distintas series en la misma gráfica con el fin de poder compararlas.

### Diseño

Arquitectura, componentes, estilo

### Implementación

Svelte

Svelte, como se explica anteriormente, es un framework nuevo sin mucha documentación aún en internet más allá de la oficial. Esto ha supuesto, junto con la falta de experiencia en frontend del desarrollador, que se inviertan muchas horas en investigación. Sin embargo, una vez se aprenden las nociones básicas de Svelte, este resulta ser muy potente, ya que se pueden implementar tareas con menos líneas de código de las que harían falta con otro framework como React. Además, Svelte permite añadir estilo a los componentes desde el mismo fichero, no hace falta crearlo apare, esto es muy útil si se quiere aplicar un estilo concreto solo a los componentes de ese mismo fichero, ya que el resto de la aplicación quedará fuera del alcance.

Por otro lado, Svelte permite realizar declaraciones reactivas. Estas declaraciones provocan una actualización del DOM al cambiar su valor. Esto simplifica mucho el desarrollo de componentes que se actualicen a menudo y deba verse reflejado en la página. Por otro lado, además de variables, también se pueden declarar sentencias como puede ser un *console.log*  o un bloque *if.*

Pero la reactividad de Svelte no acaba ahí, aunque este tipo de declaraciones sean útiles no han sido necesarias, ya que la reactividad de Svelte también es puesta en movimiento a través de las asociaciones. Esto quiere decir que, al asociar un valor a una variable también provocará que se recargue la página, visualizando así los cambios pertinentes. Sin embargo, al ser disparada por las asociaciones, el hecho de añadir un elemento a un array o realizar cualquier cambio sobre un objeto que no sea una asignación, la reactividad no se activará. No obstante, hay una alternativa muy simple a este problema, si, por ejemplo, queremos añadir un elemento a un array y que el sistema se comporte como si fuese una asociación, podemos asociar el array en cuestión a sí mismo, lo que no provocará ningún cambio en los datos, pero solucionará el problema.

Componentes externos

E-Charts

Comunicación con la API

Problemas encontrados

Framework joven, poca documentación, problemas de versiones, bugs visuales.

### Pruebas

# Conclusión

# Anexo I – Glosario de términos.

Frontend

Backend

Serie

API

Scrapper

Crawler

Endpoint

Cors

Framework

NoSQL

Reactividad

# Anexo II – Control de cambios.

27/07/2020 – Añadida tarea control de errores frontend

28/07/2020 – Subtarea de filtrado de series movida a la fase de prototipo como tarea completa.

28/07/2020 – Nueva tarea: Scrapper mineco. Movida subtarea “*Datos de las fuentes coherentes y mostrados correctamente*” a Scrapper mineco.

Se deben de poder

**Tareas**

**Desarrollo**

Importar las series de IPC y EPA de Andalucía.

Completar el back para recoger los datos que falten de las series.

Visualizar tabla de datos en formato Excel y poder descargarlo.

Poder añadir y quitar series del gráfico de forma dinámica.

Calcular la tasa de variación y añadirla al gráfico.

Botón para descargar el gráfico que se está mostrando. (canvas)

Añadir las series de IPC y EPA de España.

Completar con todas las series de Europa, Andalucía y España.

Paginación

Credenciales back

**Fallos encontrados**

Problemas al arrancar svelte

Problema mostrando las series

Credenciales scrapper

**Lecciones aprendidas**

Reactividad svelte