[PORTADA]

Índice

[1. Resumen 4](#_Toc47429055)

[2. Introducción 4](#_Toc47429056)

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

[3.1. Alcance 4](#_Toc47429058)

[3.2. Objetivos 4](#_Toc47429059)

[3.3. Planificación 4](#_Toc47429060)

[3.4. Metodología 8](#_Toc47429061)

[3.5. Herramientas 8](#_Toc47429062)

[3.6. Relación con las asignaturas del Grado 8](#_Toc47429063)

[4. Realización del proyecto 8](#_Toc47429064)

[4.1. Backend 8](#_Toc47429065)

[4.1.1. Análisis 8](#_Toc47429066)

[4.1.2. Diseño 8](#_Toc47429067)

[4.1.3. Implementación 8](#_Toc47429068)

[4.1.4. Pruebas 8](#_Toc47429069)

[4.2. Frontend 8](#_Toc47429070)

[4.2.1. 9](#_Toc47429071)

[4.2.2. Diseño 9](#_Toc47429072)

[4.2.3. Implementación 9](#_Toc47429073)

[4.2.4. Pruebas 9](#_Toc47429074)

[5. Conclusión 9](#_Toc47429075)

[I. Anexo I – Glosario de términos. 9](#_Toc47429076)

[II. Anexo II – Control de cambios. 9](#_Toc47429077)

# Resumen

Este proyecto se centra en la extracción de datos de distintas 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 mas importantes de este proyecto son los distintos Scrappers personalizados que han de extraer los datos de las distintas páginas web junto a el 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: Debido a la pandemia actual, es posible que surjan contratiempos?

Planes de mitigación 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: Mascarillas, distanciamiento social?

**AÑADIR TABLA CON IMPACTO Y PROBABILIDAD**

**Desarrollar medidas de control**

Criterios de aceptación

El proyecto será aceptado si y solo sí se cumples los siguientes criterios de aceptación:

* Cumple los requisitos establecidos.
* Es entregado dentro del plazo.
* ¿?

## Objetivos

## 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. | | |  |
| * Añadir estilo al frontend | | | | 6h |
| 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 |
| * Datos de las fuentes coherentes y mostrados correctamente. | | | | 3h |
| TF.02 | Mejorar frontend – tareas extra frontend | | | B |
| * Menú principal. | | | | 5h |
| * Búsqueda y filtrado de series. | | | | 10h |
| * 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 |
| * Búsqueda y filtrado de series. | | | | 3h |

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 **276.44 €.**

## Metodología

Metodología tradicional -> todos los requisitos y la planificación desde el principio.

Separación de frontend/backend.

Control de calidad: sonarlint, checkstyle

## 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.

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

Requisitos de información:

Se quiere guardar la siguiente información acerca de cada serie (como mínimo)

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

En cada serie, hay que guardar los datos correspondientes a cada punto en el tiempo, los cuales constan de:

* Momento (en el tiempo)
* Valor

### Diseño

Arquitectura

### 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 paginas de datos con un Crawler estas paginas no existen realmente, si no 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.

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.

A 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:

Captura de pantalla de un celular

Descripción generada automáticamente

Ilustración 1: 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 2: 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

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

Reactividad, estilo, componentes importados,

Comunicación con el backend

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

# Anexo II – Control de cambios.

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

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