Universidad de Valladolid

Escuela de Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática
Mención en Ingeniería de Software

API unificada de acceso a datos meteorológicos para la agricultura

Autor:
Christiam Alexander Mansutti Rosón

Universidad de Valladolid

Escuela de Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática
Mención en Ingeniería de Software

API unificada de acceso a datos meteorológicos para la agricultura

Autor:
Christiam Alexander Mansutti Rosón
Tutor:
Yania Crespo González Carvajal

Resumen

En la actualidad existen multitud de páginas web, aplicaciones web, servicios web, APIs, etc. que proporcionan, de forma gratuita o de pago, datos procedentes de diferentes estaciones meteorológicas tanto públicas como privadas, información de gran importancia para la agricultura.

El análisis e interpretación de los datos agrometeorológicos y agroclimáticos son especialmente útiles para la toma de decisiones en el desarrollo y la optimización de los cultivos.

Desde Argotec, empresa de ingeniería orientada a diseñar y fabricar soluciones de monitorización y control para el mercado del Internet de las cosas (IoT), plantea la necesidad de tener un punto de acceso único a todos esos datos, con un diseño que permita fácilmente el crecimiento del proyecto a futuro y la ampliación de las fuentes de datos, eliminando la complejidad de su obtención, y facilitando el análisis de los mismos.

Este Trabajo Fin de Grado tiene como objetivo la implementación de una API que permita unificar el acceso a diferentes datos de manera homogénea y transparente a otros desarrolladores o analistas de datos para su posterior explotación.

Abstract

Currently there are many web pages, web applications, web services, APIs, etc. that provide, free or paid, datas from different meteorological stations, both public and private, information of great importance for agriculture.

The analysis and interpretation of agrometeorological and agroclimatic data are especially useful for decision making in the development and optimization of crops.

From Argotec, an engineering company focused on designing and manufacturing monitoring and control solutions for the Internet of Things (IoT) market, it raises the need to have a single access point to all this data, with a design that easily allows the growth of the future project and the expansion of data sources, eliminating the complexity of obtaining them, and facilitating the analysis of them.

This End-of-Degree Project aims to implement an API that allows unified access to different data in a homogeneous and transparent manner to other developers or data analysts for later exploitation.

Agradecimientos

A mis amigos, que han estado a mi lado.
A mis compañeros, con los que he compartido camino.
A Francisco Huidobro y Antonio Sainz de Argotec, por darme la oportunidad.
A Yania Crespo, por guiarme y su paciencia en la realización de este proyecto.
Y en especial a mi madre, que siempre me ha apoyado y ha confiado en mí durante este viaje.
A todos, Gracias!

Índice

Lista de figuras

Lista de tablas

Imágenes referenciadas

1. Introducción

En la primera parte de introducción, se expondrá el origen de la idea que motiva la realización de este Trabajo Fin de Grado, sus principales objetivos, el alcance del mismo, y una breve descripción de la organización de la memoria.

1.1. Motivación

Este proyecto nace como respuesta a una necesidad específica dentro de la empresa Argotec, y más concretamente, relacionada con uno de sus proyectos más importantes, el proyecto Qampo.

Argotec es una empresa de Ingeniería especializada en el diseño y la fabricación de soluciones de control y monitorización en el mercado del Internet de las Cosas.

Ilustración 1 - Argotec

Uno de sus proyectos estrella es Qampo, una familia de productos dirigidos a profesionales de la agricultura y asesorías agronómicas para la monitorización y análisis de parámetros agrometeorológicos, y cuya finalidad es optimizar la producción de los cultivos, y mejorar su calidad.

Ilustración 2 - Qampo

Para más información se puede visitar tanto la página oficial de argotec como la página más específica del proyecto Qampo, disponibles en:

https://argotec.es y https://qampo.es

Actualmente es posible obtener datos meteorológicos de diversas fuentes de información, pero cada una proporciona una forma diferente de consumir los datos, ya sea mediante consulta directa a páginas web o a través de APIs.

Estas diferentes formas implican que el acceso a la información es de forma heterogénea, y es en este contexto, donde surge la necesidad de este proyecto:

- Disponer de un punto de acceso unificado y homogéneo a las distintas fuentes de datos.

1.2. Objetivos

Los principales objetivos del proyecto son:

- Crear una arquitectura que permita integrar diferentes fuentes de información meteorológica.
- Obtener datos meteorológicos de una determinada fuente de datos.
- Obtener ciertas medidas útiles para la generación de recomendaciones de riego.
- Proporcionar una API a través de la cual se pueda acceder a los datos de interés de forma rápida y sencilla.
- Almacenar la información para facilitar su análisis y procesamiento.

1.3. Alcance

Teniendo en cuenta la duración estimada para el Trabajo de Fin de Grado, de unas 300 horas aproximadamente, este proyecto se plantea como la base y el punto de partida de un proyecto con perspectivas a largo plazo. Adaptando los requisitos al tiempo limitado, las principales prioridades son:

- Realizar una fase de investigación, en busca de plantear la arquitectura más adecuada a los objetivos, y seleccionar las tecnologías que mejor se adapten a nuestras necesidades.
- Integrar una primera fuente de datos: SiAR, es decir, obtener los datos recogidos a través de la red de estaciones agrometeorológicas del Sistema de Información Agroclimática para el Regadío (SiAR) del Ministerio de Agricultura y Pesca, Alimentación y Medio Ambiente.
- Almacenar los datos obtenidos de SiAR en una base de datos, proporcionando además una API REST que permita acceder a la información deseada.

1.4. Organización de la memoria

A lo largo del siguiente documento intentare explicar los aspectos fundamentales y más relevantes dentro del proceso de desarrollo de este proyecto.

La memoria se dividida divide principalmente en los siguientes bloques:

- **Estado del arte**: visión general del contexto que rodea al proyecto.
- **Metodologías/Marco tecnológico**: técnicas y herramientas utilizadas.
- **Plan de desarrollo**: medios empleados para lograr los objetivos del proyecto.
- **Proceso de desarrollo**: descripción de las diferentes etapas en el desarrollo.
- **Conclusiones/Líneas futuras**: valoraciones finales y propuestas futuras.
- **Anexos**: material de interés relacionado con el proyecto.
- **Bibliografía**: principales referencias de este Trabajo de Fin de Grado.

2. Estado del arte

En esta sección el objetivo es mostrar algunos de los elementos más importantes relacionados con el proyecto, de forma que se ofrezca una visión general para una mejor comprensión.

2.1. Agricultura y medio ambiente

La agricultura es el conjunto de técnicas y conocimientos aplicados en la explotación de los recursos que la tierra origina por la acción del ser humano. Desde las primeras sociedades, ha sido una de las actividades más importantes, y base para el desarrollo de las mismas.

Son muchos los problemas asociados a las actividades agrícolas, y uno de los más importantes es: el fuerte impacto que la agricultura genera en el medio ambiente.

En muchos casos la agricultura implica la modificación o destrucción de entornos naturales, contaminación de ecosistemas, y es responsable de producir gran parte de las emisiones de gases de efecto invernadero.

Estos hechos, el continuo deterioro de nuestro entorno, hacen que sea especialmente importante realizar un mayor esfuerzo en mejorar el uso de unos recursos limitados, aplicando procesos más eficientes, contribuyendo lo máximo posible a preservar el medio ambiente.

2.2. Meteorología

La meteorología es la ciencia encargada de estudiar los diferentes fenómenos atmosféricos, mediante una serie de parámetros que se presentan en un lugar y momento determinado, como son:

- Temperatura: grado de calor del aire
- Humedad: cantidad de vapor de agua en el aire
- Precipitación: cualquier hidrometeoro que cae de la atmosfera a la superficie terrestre
- Viento (Velocidad y dirección): corriente de aire que se produce en la atmósfera por variaciones de presión
- Radiación solar: conjunto de radiaciones electromagnéticas emitidas por el sol
- Presión atmosférica: fuerza por unidad de superficie que ejerce el aire sobre la superficie terrestre.

El conocimiento de la meteorología ofrece la posibilidad de realizar predicciones de gran importancia y utilidad para la agricultura. Además, a partir de los datos meteorológicos, también es posible obtener información relevante para generar recomendaciones que favorecen la toma de decisiones en los procesos productivos agrícolas.

2.3. Estaciones meteorológicas

Una estación meteorológica es un conjunto de instrumentos para medir y registrar diversos parámetros meteorológicos. Algunos de los elementos más comunes son:

- Termómetro: mide la temperatura.
- Geotermómetro: mide la temperatura del suelo a diferentes niveles de profundidad.
- Higrómetro: mide la humedad.
- Pluviómetro: mide la cantidad de agua caída sobre el suelo por metro cuadrado, ya sea en forma de lluvia, nieve o granizo.
- Anemómetro: mide la velocidad del viento.
- Veleta: indica la dirección del viento.
- Piranómetro: mide la radiación solar global, es decir directa más difusa
- Barómetro: mide la presión atmosférica

Ilustración 3 - Instrumentos meteorológicos Argotec

Existe toda una red de estaciones meteorológicas repartidas por todo el territorio español, algunas de propiedad pública y otras de carácter privada, y en algunos casos conectadas entre sí compartiendo información.

Por ejemplo, desde Argotec se ofrece a los clientes la posibilidad de instalar sus propias estaciones meteorológicas, compuestas por varios sensores, y un datalogger inalámbrico profesional, que permite monitorizar la meteorología en el punto del terreno que más interese al cliente, proporcionando información en tiempo real sobre temperatura, humedad, precipitación, viento, o radiación solar.

Ilustración 4 - Estación meteorológica Argotec

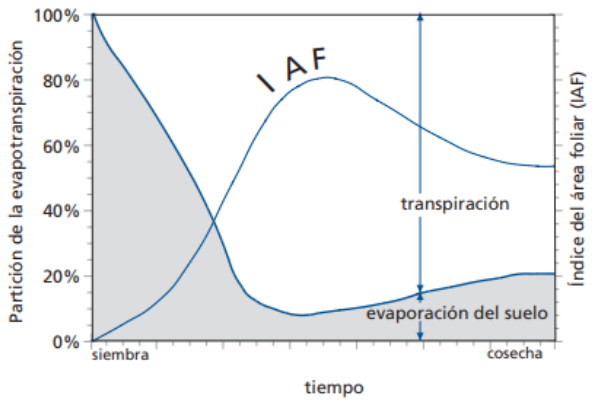
A partir de los datos recopilados por las diferentes estaciones, se pueden analizar y procesar los datos, para generar una de serie de recomendaciones específicas para un cultivo concreto, como por ejemplo, las necesidades hídricas.

La posibilidad de generar recomendaciones precisas y efectivas, empleando la información obtenida por nuestro sistema, es uno de los fines principales, y que dan verdadero sentido a este proyecto.

2.4. Evapotranspiración

Un concepto importante es la evapotranspiración, la cual representa la unión de la evaporación más la transpiración, es decir, la pérdida de humedad de una superficie por evaporación directa y la pérdida de humedad por transpiración de las plantas. Ambos procesos ocurren simultáneamente y resulta difícil distinguir entre ambos procesos.

La evaporación de un suelo cultivado depende principalmente de la radiación solar que llega a la superficie del suelo, por lo que en las primeras etapas del cultivo, es el principal proceso de pérdida de agua, pero con el desarrollo del cultivo y finalmente cuando este cubre totalmente el suelo, la transpiración se convierte en el proceso principal.



Tomada de [1]

Su principal interés radica en su aplicación para la cuantificación de los recursos hídricos demandados por los cultivos, en otras palabras, para estimar cuánta agua necesita un cultivo específico.

Los principales factores que influyen en la evapotranspiración son:

- Variables climáticas: los principales parámetros climáticos que afectan en la evapotranspiración son la radiación, la temperatura del aire, la humedad atmosférica y la velocidad del viento.

- Factores de cultivo: las diferencias en resistencia a la transpiración, la altura del cultivo, la rugosidad del cultivo, el reflejo, la cobertura del suelo y las características radiculares del cultivo dan lugar a diferentes niveles de evapotranspiración en diversos tipos de cultivos aunque se encuentren bajo condiciones ambientales similares.

- Manejo y condiciones ambientales: los factores tales como salinidad o baja fertilidad del suelo, uso ilimitado de fertilizantes, presencia de horizontes duros o impenetrables en el suelo, ausencia de control de enfermedades y de parásitos y el mal manejo del suelo pueden limitar el desarrollo del cultivo y reducir la evapotranspiración.

A continuación voy explicar brevemente los principales conceptos asociados a la evapotranspiración:

- ETo: denominada evapotranspiración de referencia o del cultivo de referencia, y es la tasa de evapotranspiración de una superficie de referencia, sin restricciones de agua. Depende exclusivamente de las condiciones meteorológicas. Este concepto estudia la demanda de evapotranspiración de la atmósfera, independientemente del tipo y desarrollo del cultivo, y de las prácticas de manejo.

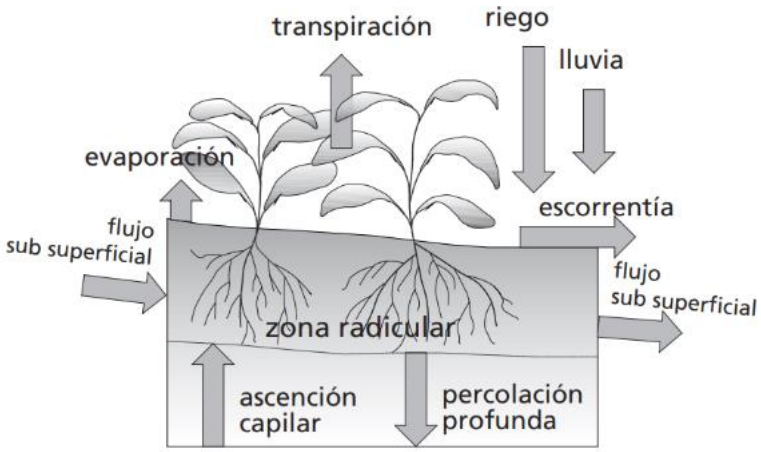
- ETc: denominada evapotranspiración del cultivo, tiene en cuenta tanto las condiciones atmosféricas, como el nivel de reservas de humedad del suelo, y las características propias del cultivo.

- Kc: denominado coeficiente del cultivo, depende de las características propias de cada cultivo, de las características del suelo, y de los niveles de humedad. Permite calcular la ETc a partir de la ETo. El uso de este coeficiente es menos preciso que otros métodos, pero también es más fácil de aplicar.

$$ET_c = K_c\, ET_o$$

Tomada de [2]

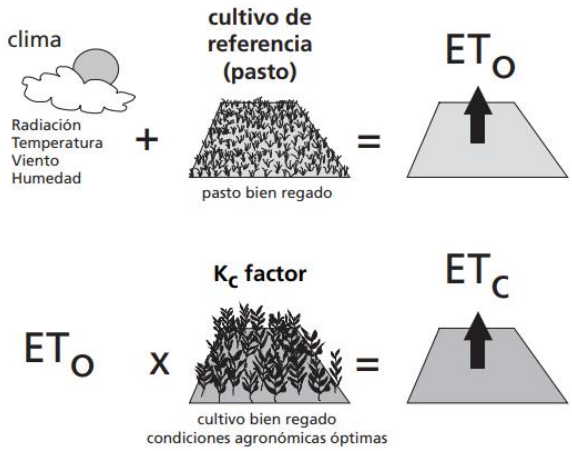
- Pe: la denominada precipitación efectiva, es la parte de la precipitación total que es aprovechada por las plantas. Depende de múltiples factores como pueden ser la intensidad de la precipitación o la aridez del clima, y también de otros como la inclinación del terreno, contenido en humedad del suelo o velocidad de infiltración.



Tomada de [3]

- ETc – Pe: representa las necesidades de riego de un cultivo concreto menos la precipitación efectiva, y por tanto el agua que no tienes que aportar mediante tu sistema de riego.

- El cálculo de la ETo se puede hacer de varias formas, pero se recomienda que se realice mediante el método de Penman-Monteith a partir de los datos meteorológicos, que requiere los datos de radiación, temperatura del aire, humedad atmosférica y velocidad del viento.



Tomada de [4]

2.5. SiAR

El Sistema de Información Agroclimática para el Regadío (SiAR) es una infraestructura que captura, registra y divulga los datos agroclimáticos necesarios para el cálculo de la demanda hídrica de las zonas de riego. Esto permite una mejor planificación, gestión y control de las explotaciones de regadío.

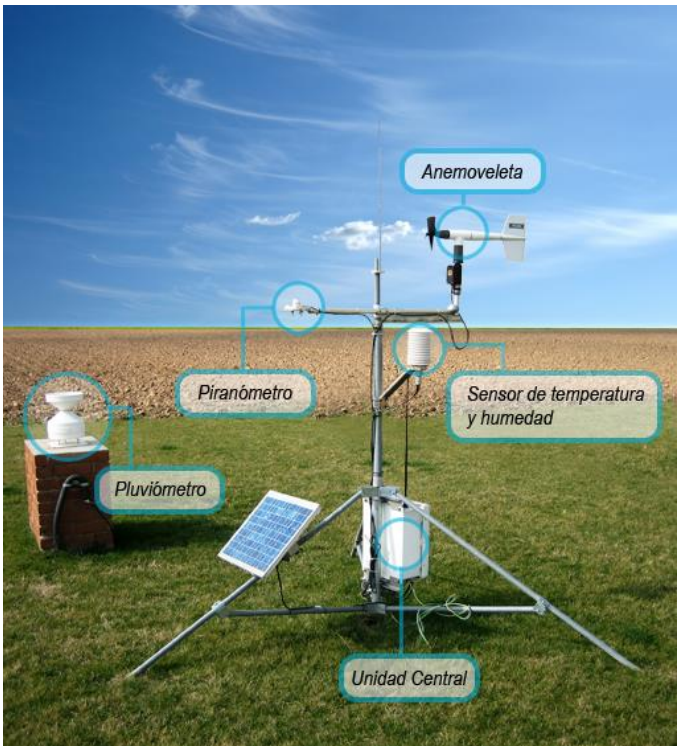
El SiAR consta de 468 Estaciones Agroclimáticas, 12 Centros Zonales y 1 Centro Nacional:

- Estaciones Agroclimáticas: repartidas por todo el territorio nacional, se encargan de recoger los datos climáticos para enviarlos a los Centros Zonales.

- Los Centros Zonales: se encargan de obtener, almacenar y procesar los datos recogidos diariamente de las estaciones instaladas en su Comunidad Autónoma.

- El Centro Nacional: tiene como objetivo almacenar y publicar los datos a nivel Nacional, y divulgar la información a nivel institucional.

Configuración de una estación genérica del SiAR:



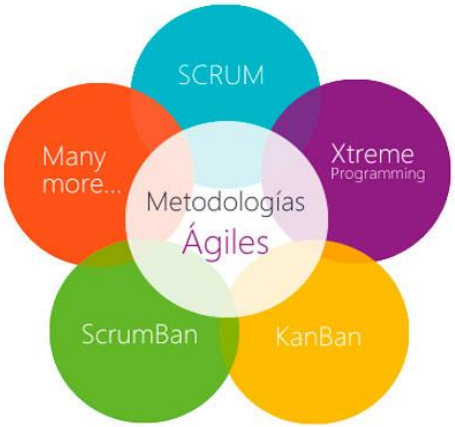
Tomada de [5]

Más información en su página web oficial:

http://eportal.mapama.gob.es/websiar/Inicio.aspx

3. Metodología

En este punto describiré la metodología empleada para la realización de este proyecto, que se corresponde con una metodología ágil, basada principalmente en SCRUM, aunque con influencias del Xtreme Programming, e incluso de Kanban.



Tomada de [6]

3.1. Metodologías ágiles

Hablar de metodología ágil o desarrollo ágil, es hablar de un enfoque donde el desarrollo del proyecto se realiza de forma iterativa e incremental, los requisitos evolucionan en el tiempo en función de las necesidades del proyecto, y se mantiene una relación continua con el cliente.

Algunos rasgos que la caracterizan son:

- Una mayor flexibilidad
- Capacidad de adaptación ante los cambios e imprevistos
- Retroalimentación continua

Tomada de [7]

Es una visión algo diferente a las metodologías más tradicionales, y se basa en doce principios fundamentales recogidos en el denominado Manifiesto Ágil:

"Nuestra mayor prioridad es satisfacer al cliente mediante la entrega temprana y continua de software con valor.

Aceptamos que los requisitos cambien, incluso en etapas tardías del desarrollo. Los procesos Ágiles aprovechan el cambio para proporcionar ventaja competitiva al cliente.

Entregamos software funcional frecuentemente, entre dos semanas y dos meses, con preferencia al periodo de tiempo más corto posible.

Los responsables de negocio y los desarrolladores trabajamos juntos de forma cotidiana durante todo el proyecto.

Los proyectos se desarrollan en torno a individuos motivados. Hay que darles el entorno y el apoyo que necesitan, y confiarles la ejecución del trabajo.

El método más eficiente y efectivo de comunicar información al equipo de desarrollo y entre sus miembros es la conversación cara a cara.

El software funcionando es la medida principal de progreso.

Los procesos Ágiles promueven el desarrollo sostenible. Los promotores, desarrolladores y usuarios debemos ser capaces de mantener un ritmo constante de forma indefinida.

La atención continua a la excelencia técnica y al buen diseño mejora la Agilidad.

La simplicidad, o el arte de maximizar la cantidad de trabajo no realizado, es esencial.

Las mejores arquitecturas, requisitos y diseños emergen de equipos auto-organizados.

A intervalos regulares el equipo reflexiona sobre cómo ser más efectivo para a continuación ajustar y perfeccionar su comportamiento en consecuencia."

Estos principios se pueden resumir en cuatro:

"**Individuos e interacciones** sobre procesos y herramientas
Software funcionando sobre documentación extensiva
Colaboración con el cliente sobre negociación contractual
Respuesta ante el cambio sobre seguir un plan"

Y como en el mismo manifiesto exponen, aunque se valora los elementos de la derecha, se valora más los de la izquierda en negrita.

Puedes consultar directamente el manifiesto en la siguiente página web:

http://agilemanifesto.org/iso/es/manifesto.html

http://agilemanifesto.org/iso/es/principles.html

3.2. Scrum

Scrum se define como un marco de desarrollo ágil para la gestión de proyectos, y a continuación, para explicar en qué consiste Scrum, voy a describir las prácticas más relevantes realizadas durante el desarrollo de este Trabajo de Fin de Grado, desde mi propia experiencia dentro de la empresa, lo cual considero más interesante que dar una descripción teórica y genérica de las características principales de Scrum.

Tomada de [8]

3.2.1. ¿Qué son las historias de usuario?

Las denominadas historias de usuario son el elemento básico para la descripción de las funcionalidades a incorporar en el software, y su implementación aporta valor al cliente.

E las metodologías ágiles son la manera de especificar los requisitos, representados por una o dos frases en lenguaje común.

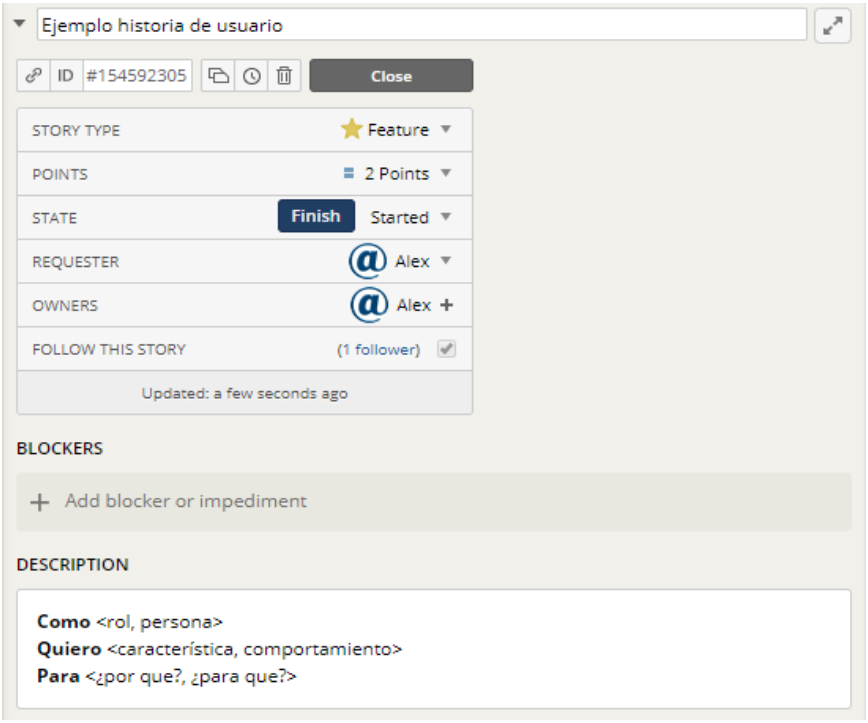


Ilustración 5 - Ejemplo historia de usuario

En mi caso, las historias de usuario fueron especificadas a través de Trello, y en la última fase por medio de Pivotal Tracker.

3.2.2. ¿Qué es el Product Backlog?

El product backlog es una lista priorizada de las historias de usuario con todas las funcionalidades necesarias, de forma que se pueda tener una perspectiva clara de lo que se quiere hacer durante el desarrollo del proyecto, y cuáles son las prioridades desde el punto de vista del cliente.

La pila de producto, como también se la conoce, no se trata de una lista inmutable, sino que durante la vida del proyecto se pueden introducir cambios, si el cliente lo considera necesario.

Las tareas son visibles para todo el equipo, y ofrecen una visión a futuro de lo que se espera, por lo que se puede ver como una herramienta de planificación.

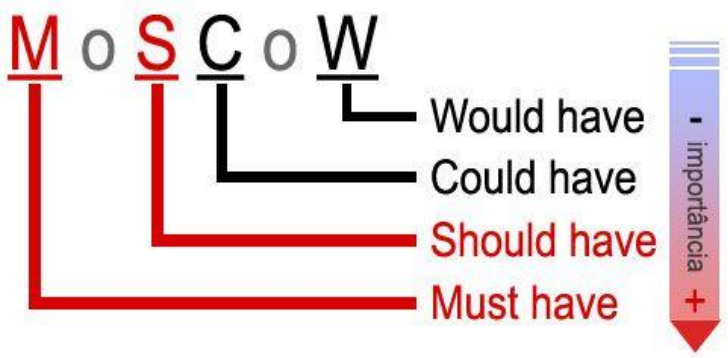
La priorización de las tareas se basa en la técnica **MoSCoW**, que clasifica los requisitos desde un punto de vista que para el cliente tiene un significado real:

- **M** - Must have: requisito imprescindible que debe estar implementado en la versión final del proyecto. Son los requisitos mínimos.

- **S** - Should have: requisito importante que debería formar parte de la versión final del proyecto, pero no obligatoriamente necesario, puede quedar fuera si hay motivos que los justifiquen.

- **C** - Could have: requisito interesante para el cliente pero no necesario, a implementar en caso de disponer del tiempo suficiente, y se ve más bien como un extra positivo dentro del proyecto.

- **W** - Won't have: requisito que no se tiene pensado incluir por el momento, es algo opcional que más bien se propone a futuro.



Tomada de [9]

Se debe desarrollar siguiendo ese orden, y esa priorización ayuda al equipo a tener más claro cuáles son las cosas realmente importantes, cuáles no tanto, y cuales se pueden dejar en segundo plano, o directamente obviar.

3.2.3. ¿Qué es la Iteration Planning Meeting?

La IPM es una reunión de todo el equipo, donde se comentan las siguientes historias de usuario a desarrollar durante el sprint, se leen, se explican si es necesario alguna aclaración, y se estima entre todos la duración de la tareas y el esfuerzo de desarrollo.

La estimación se realiza empleando la sucesión de Fibonacci, asignando a cada tarea un valor entre 1, 2, 3 o 5. Es una adaptación del conocido como Planning Póker, una técnica más al Xtreme Programming (XP).

Sentados en una mesa, tras leer y comentar una tarea, se pasa a votar de la siguiente forma: primero se levanta la mano con el puño cerrado a la vista de todos, después se cuenta hasta tres en voz alta, y todos al mismo tiempo valoran la tarea asignando un valor con los dedos de la mano según su criterio.

De esta forma se puede ver de un golpe de vista todas las estimaciones propuestas. Después, en función de las principales estimaciones, se discute y de forma consensuada se asigna un valor final a la tarea.

Tomada de [10]

La IPM se realizaba al principio de cada sprint, normalmente los lunes a media mañana, y a parte de reducir los tiempos de estimación, y por lo tanto, bastante útil, también resulta una práctica divertida.

Desde el punto de vista del Scrum se corresponde con el Sprint Planning, y da como resultado el Sprint Backlog.

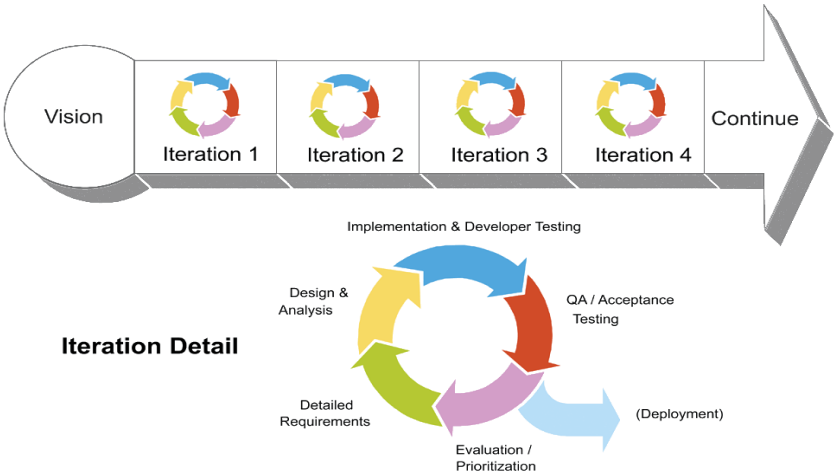
3.2.4. ¿Qué es un Sprint?

Llamamos sprint a cada uno de los periodos de tiempo en los que se divide el desarrollo del proyecto. Se corresponden con una iteración completa, donde se desarrolla un incremento para el proyecto.

Los sprint tienen una duración determinada, establecida desde el principio, y todos deberían tener la misma duración a lo largo de la vida del proyecto. Como la propia Scrum Guide lo define, es el corazón de Scrum.

La duración de cada sprint en este proyecto ha sido de dos semanas, sumando un total de 8 sprint al finalizar el Trabajo de Fin de Grado.

En cada Sprint se realizan las diferentes actividades necesarias para alcanzar los objetivos del sprint.



Tomada de [11]

3.2.5. ¿Qué es el Sprint Backlog?

El Sprint Backlog es la lista con las historias de usuario que se pretenden realizar a lo largo de una iteración

Cada historia de usuario puede implicar una serie de tareas asociadas, y en este proyecto, el seguimiento de las tareas se realizó usando el método Kanban, donde cada tarea se encuentra en un estado entre los siguientes:

- To Do: Tarea pendiente de hacer
- Doing: Tarea en estado de desarrollo
- Done: Tarea terminada

Tomada de [12]

3.2.6. ¿Qué es la Standup?

La Standup o Daily es una reunión diaria entre todos los miembros del equipo, de unos diez minutos de duración, que normalmente se realiza a la media hora de empezar a trabajar, y tras colocarnos en círculo, uno por uno se expone lo que se ha hecho el día anterior, y se propone que se tiene pensado hacer ese día.

El objetivo principal es poner en contexto a todo el equipo sobre la evolución diaria del proyecto, de forma que todos los miembros pueden tener una visión general del estado y evolución del mismo, y en el que además, se puede aprovechar para exponer cualquier tipo de problema que se encuentre durante el desarrollo o plantear dudas.

En nuestro caso, como valor añadido, se decidió realizar las daily completamente en inglés.

Tomada de [13]

3.2.7. ¿Qué es la Retrospectiva?

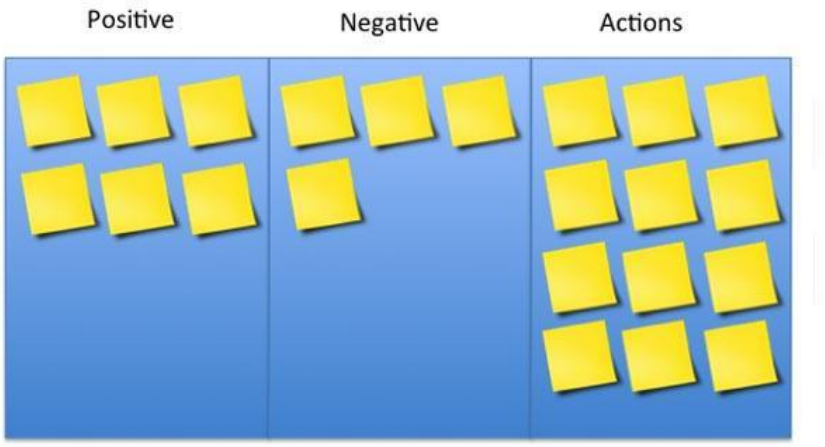
La retrospectiva es una reunión entre todos los miembros del equipo, un momento para hacer autoevaluación e identificar posibles problemas que se estén produciendo en el desarrollo, es una herramienta de mejora para el equipo.

Se realiza enfrente de una pizarra blanca, dividida en tres columnas que se corresponden con las cosas que han ido bien, las cosas a mejorar, y las cosas que han ido mal, y cada uno de los integrantes del equipo puede escribir en la pizarra todo lo que considere oportuno.

Aunque la teoría dice que las retrospectivas se realizan al final de cada sprint o iteración, en nuestro caso se acordó realizar una por semana, los viernes a última hora, y en cada semana le tocaba a una persona diferente ser el responsable de dirigir la reunión.

El funcionamiento es sencillo, el encargado de guiar la retro va leyendo uno por uno cada comentario, poniendo en común lo anotado, y acto seguido, da el turno de palabra a la persona que ha escrito el comentario, para que pueda explicar sus razones o cualquier cosa que quiera compartir con el resto del equipo, normalmente relacionado con las personas, las relaciones, los procesos o las herramientas.

Para el caso de las cosas a mejorar o negativas, en caso de ser necesario, se anotan las posibles acciones para resolver los problemas en cuestión, y cada acción se asigna a uno o varios para ser ejecutadas.



Tomada de [14]

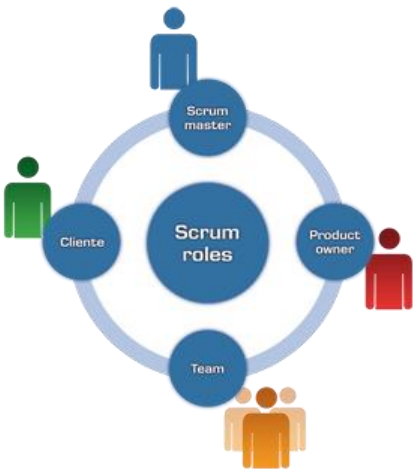
3.2.8. Roles en Scrum

Los principales roles en este proyecto han sido:

- Product Owner: encargados de optimizar el valor del producto y gestionar el Product Backlog, Francisco Huidobro y Antonio Sainz.

- Scrum Master: encargado de gestionar el proceso Scrum y eliminar impedimentos, Jorge Lebrato

- Team: encargado de desarrollar los incrementos, Christiam Mansutti



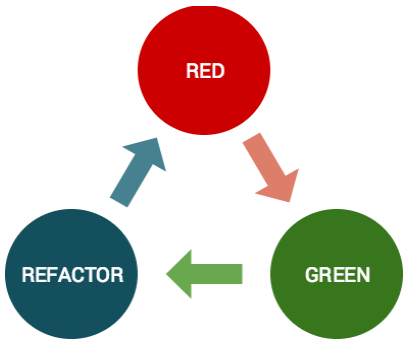
Tomada de [15]

Aunque por la propia naturaleza de este proyecto, el team o equipo real solo está formado por mí, las diferentes actividades descritas anteriormente se realizaban en conjunto con otros miembros de otros equipos diferentes, pero que están directamente relacionados con el proyecto, como son responsables de hardware, otros desarrolladores, o diseñador gráfico. Los principales stakeholders en este proyecto son los analistas de datos.

3.3. TDD

El TDD (Test-Driver Development) o desarrollo guiado por pruebas, si hubiera que describir en pocas palabras en qué consiste, se podría decir que se basa en escribir primero las pruebas.

El procedimiento es crear primero un test de prueba que falle, después implementar el código estrictamente necesario para pasar la prueba, y por último refactorizar el código mejorando lo que ya está implementado.



Tomada de [16]

El uso de TDD es una práctica habitual dentro de las metodologías ágiles, y el desarrollo de este proyecto se ha visto influenciado por el TDD, sin llegar a un seguimiento completamente estricto, sí que ha marcado un estilo a la hora de desarrollar el código, prestando un especial interés a las pruebas unitarias y de integración.

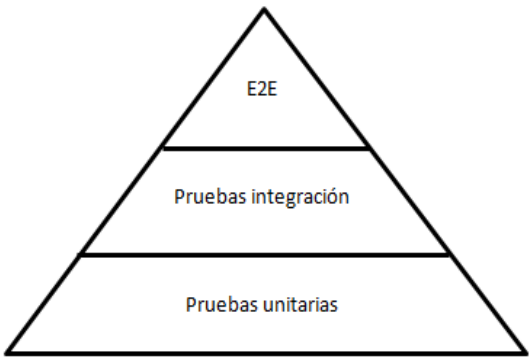


Ilustración 6 - Pirámide pruebas

Algunas de las ventajas del desarrollo con TDD son:

- Favorece la creación de código limpio
- Mejora la calidad del software
- Detección temprana de errores
- Código con menos errores
- Evita código innecesario
- Código más fácil de mantener

En este proyecto las pruebas se han realizado con JUnit, que describiré en el apartado 4.

3.3.1. Pruebas unitarias

Las pruebas unitarias son las responsables de comprobar el correcto funcionamiento de una unidad del software individual, como una función o una clase, de forma aislada del resto del código, determinando si el resultado es el esperado.

Las pruebas unitarias deben cumplir una serie de requisitos:

- Automatizable: se tiene que poder ejecutar de forma automática y rápida
- Completa: debe cubrir el mayor código posible para ser completa
- Repetible: en caso de repetir la prueba el resultado debe ser siempre el mismo
- Independiente: se prueba una parte concreta de código y que no se relaciona con el resto del código

3.3.2. Pruebas de integración

Las pruebas de integración tienen como objetivo probar un conjunto de elementos del software, y comprobar que funciona tal y como se espera. Se realizan después de las pruebas unitarias.

3.3.3. Pruebas E2E

Las pruebas end to end se refieren a las pruebas para verificar que todos los componentes del sistema software se relacionan e interactúan correctamente. Dado el alcance para este proyecto, estas pruebas no se van a contemplar por el momento, ya que con las pruebas unitarias y de integración, se cubre todo lo necesario, dejando para un futuro, cuando se incluyan más componentes, el uso de estas pruebas E2E.

4. Tecnologías y herramientas

A lo largo de este punto pretendo dar una breve descripción de las principales herramientas utilizadas durante el desarrollo del proyecto, con una imagen que las represente para que sean más fácilmente identificables por cualquiera que esté interesado en ellas, y por supuesto, el enlace a sus respectivas páginas oficiales donde podréis obtener más información o descargarlas.

En el apartado de conclusiones de esta memoria (9), daré mi opinión personal respecto a las distintas tecnologías y herramientas.

Todas las herramientas empleadas para el desarrollo de este proyecto son completamente gratuitas, o gratuitas con ciertas restricciones, pero en cualquier caso, supone evitar tener que hacer frente al pago de licencias con costes elevados.

4.1. Debian 9 Stretch

El sistema operativo utilizado ha sido Debian, una de las distribuciones Linux más estables, más concretamente, su versión Debian 9 Stretch.

Personalmente Debian siempre me ha gustado y me ha resultado más interesante en comparación con otras alternativas para Linux, por lo que lo primero que hice al arrancar este proyecto, fue realizar la instalación desde cero en una máquina de 64 bits puesta a mi disposición por parte de la empresa.

Más información en su página web oficial:

https://www.debian.org

4.2. Java

Respecto a Java, ¿qué puedo decir? estamos hablando de uno de los lenguajes de programación más populares y extendidos en todo el mundo, tras analizar y discutir cuál podría ser el lenguaje de programación más adecuado para el desarrollo de la aplicación, y aunque en un momento tuve mis dudas sobre si probar con Groovy, lo tuve bastante claro, la mejor opción era Java.

Más información en su página web oficial:

https://www.java.com/es

4.3. Intellij IDEA

El entorno de desarrollo integrado elegido para desarrollar la aplicación objeto de este proyecto, ha sido Intellij IDEA, para muchos el mejor IDE para Java.

Después de haber probado y usado otros IDEs también muy populares como NetBeans o Eclipse, sin duda considero a Intellij IDEA como la mejor opción.

Este entorno de desarrollo tiene dos versiones disponibles, una gratuita, la Community Edition, y otra con licencia de pago, la Ultimate Edition.

La versión utilizada ha sido la Community Edition, que aunque en esta versión no incluye algunos plugin interesantes como pueden ser el de Spring Initializr para generar un proyecto Spring Boot, cumple de sobra con todo lo necesario para el desarrollo de una aplicación.

Más información en su página web oficial:

https://www.jetbrains.com/idea

4.4. Spring Boot

Spring Boot es un framework de Spring que simplifica tanto la creación como el despliegue de una aplicación, permitiendo al desarrollador centrarse en lo verdaderamente importante, crear el código, evitando algunos dolores de cabeza conocidos por todos los que habitualmente trabajan en el desarrollo de aplicaciones.

Al no disponer del plugin para Intellij, para crear un proyecto Spring Boot es necesario ir a la página web:

https://start.spring.io/

Y una vez allí, en unos pocos pasos, se puede crear un proyecto Spring Boot:

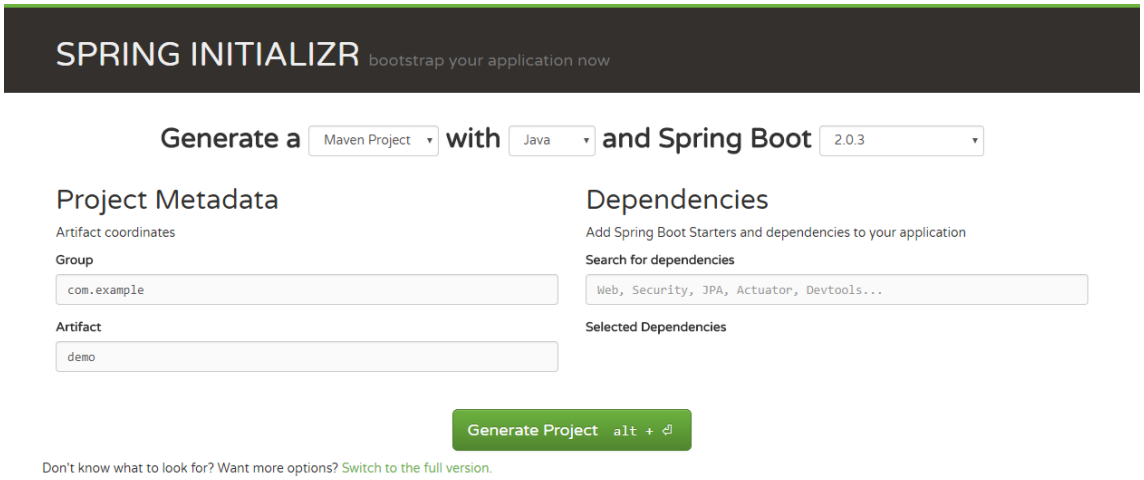


Ilustración 7 - Spring Initializr

4.5. Gradle

Como herramienta para la automatización de builds, he usado Gradle, una evolución de Maven para la gestión de dependencias.

Más información en su página web oficial:

https://gradle.org/

4.6. Sistemas de Control de Versiones/Repositorio

Como herramienta para el control de versiones, no había mucho que pensar, Git hace lo que tiene que hacer, de forma simple y eficiente, facilitando el seguimiento del código de tu proyecto, un indispensable. Y como repositorio he usado GitLab.

Más información en sus páginas web oficiales:

https://git-scm.com/ https://about.gitlab.com/

4.7. Herramientas API

Para la realización de peticiones HTTP REST y probar la API, uso Postman, y como herramienta para documentar la API, Swagger, la cual te permite especificar los recursos disponibles en la API, y los endpoint a los que se puede llamar.

Más información en sus páginas web oficiales:

https://www.getpostman.com/ https://swagger.io/

4.8. Web

4.8.1. HTML, CSS, JavaScript y JQuery

Para el desarrollo del proyecto, ha sido esencial el manejo de HTML, CSS, JavaScript y JQuery, en el estudio y análisis de las páginas web, para su manipulación y conseguir obtener la información deseada por medio de los web scraper.

Más información en sus páginas web oficiales:

https://www.w3schools.com/htmL https://www.w3schools.com/Css/ https://jquery.com/

4.9. Herramientas Web Scraping

Para obtener directamente la información de la página web SiAR, al no disponer de ninguna API proporcionada por el propio sitio web, ha sido necesario emplear técnicas de web scraping. Fue necesaria una fase de investigación, y puesta a prueba de diferentes herramientas, en la búsqueda de alcanzar los objetivos marcados, con resultados distintos.

Primero JSoup, una librería Java para parsear páginas HTML, usando DOM, CSS y JQuery, y a pesar de ser bastante efectiva y recomendable para página web estáticas, se mostró insuficiente en el caso de páginas dinámicas que contienen JavaScript, por lo que fue descartada como herramienta para el web scraping, aunque en la fase final del proyecto, tuvo una segunda oportunidad, haciendo muy bien su trabajo.

Después HtmlUnit, un navegador web sin interfaz de usuario en Java, que permite parsear los datos de tanto páginas web estáticas como dinámicas, más potente que JSoup, pero que después de una serie de pruebas, también resultó ser insuficiente, además de demostrar baja fiabilidad en determinados contextos, por lo que al igual que en el caso anterior, se descartó como opción para el web scraping.

Y por último, de entre todas las herramientas disponibles para llevar a cabo el web scraping, la que dio mejores resultados, fue sin duda Selenium, una herramienta originalmente destinada para la automatización de pruebas, pero que ha demostrado ser una opción muy recomendable en este tipo de tareas.

Más información en sus páginas web oficiales:

https://jsoup.org/ http://htmlunit.sourceforge.net/ https://www.seleniumhq.org/projects/webdriver

4.10. Herramientas persistencia de datos

A la hora de elegir la base de datos, se discutió primero entre elegir un modelo de base de datos relacional o si mejor emplear una base de datos no relacional, evaluando las ventajas y desventajas de cada uno, pero finalmente la elección fue usar una relacional.

En las primeras etapas del desarrollo he usado como gestor de base de datos, H2, específicamente en el modo en memoria, y aunque realiza prácticamente lo mismo que cualquier otra base de datos como MySQL o PostgreSQL, realmente no es comparable y tiene sus limitaciones, pero es bastante rápida y cómoda de usar.

Como servidor de base de datos he usado PostgreSQL, una de las bases de datos más conocidas y usadas junto a MySQL.

También he usado JDBC, una interface que permite la conexión entre mi aplicación Java y la base de datos.

Como ORM o herramienta de mapeo objeto-relacional he usado Hibernate, que mediante anotaciones, simplifica todo lo relacionado con la persistencia.

Más información en sus páginas web oficiales:

http://www.h2database.com/html/main.htm https://www.postgresql.org http://hibernate.org/

4.11. Herramientas para test


Para la realización de las pruebas he usado principalmente JUnit, Hamcrest y Mockito.

- JUnit es el conjunto de bibliotecas por excelencia para la realización de pruebas unitarias en Java.
- Hamcrest es una librería que proporciona una serie matchers para hacer los test más legibles.
- Mockito es un framework para crear mocks en Java, o lo que es lo mismo, simular el comportamiento de componentes u objetos para test de prueba.

Más información en sus páginas web oficiales:

https://junit.org/junit5 http://hamcrest.org http://site.mockito.org/

4.12. Herramientas planificación y gestión

Como herramienta principal de seguimiento he usado Trello, basada en el método Kanban, que ayuda en las tareas de planificación, mediante la organización de tareas mediante listas, tableros y tarjetas.

En la última fase del proyecto se incorporó el uso de Pivotal Tracker a modo de prueba para futuro, la cual es la herramienta por excelencia en proyectos ágiles para el seguimiento de las tareas.

Más información en sus páginas web oficiales:

https://trello.com https://www.pivotaltracker.com

4.13. Servicios Google

He usado Drive como espacio de almacenamiento en la nube, que permite compartir y editar documentos en línea, y como servicio de correo principal, Gmail.

Más información en sus páginas web oficiales:

https://www.google.com/drive https://www.google.com/gmail

5. Plan de desarrollo software

5.1. Planificación temporal inicial

El Trabajo Fin de Grado tiene asignado 12 créditos ECTS, como cada crédito ECTS equivale a 25 horas, por tanto el proyecto debe tener una duración como mínimo de 300 horas.

El comienzo del proyecto se marcó para el 6 de Noviembre de 2017, y teniendo en cuenta el calendario laboral se estimó la finalización del proyecto para 4 de Marzo de 2018.

Se tiene en cuenta la posibilidad de ampliar algo más la duración temporal del proyecto, en caso de ser necesario, ya que se ha buscado tener un margen amplio de tiempo hasta su necesaria finalización y entrega con motivo de los plazos establecidos desde la propia universidad para su presentación y defensa.

El proyecto se desarrollará a lo largo de 14 semanas de trabajo, con una media de 25 horas semanales, dando como resultado un mínimo de 350 horas. Se establecen 7 iteraciones de 2 semanas de duración por cada sprint:

Tabla 1 - Calendario trabajo

Semana	Fecha	Sprint
1	6 Noviembre - 12 Noviembre	**1**
2	27 Noviembre - 3 Diciembre	**1**
3	4 Diciembre - 10 Diciembre	**2**
4	11 Diciembre - 17 Diciembre	**2**
5	18 Diciembre - 24 Diciembre	**3**
6	1 Enero - 7 Enero	**3**
7	8 Enero - 14 Enero	**4**
8	15 Enero - 21 Enero	**4**
9	22 Enero - 28 Enero	**5**
10	29 Enero - 4 Febrero	**5**
11	5 Febrero - 11 Febrero	**6**
12	12 Febrero - 18 Febrero	**6**
13	19 Febrero - 25 Febrero	**7**
14	26 Febrero - 4 Marzo	**7**

5.1.1. Ajuste final a la planificación temporal

Al llegar al final de la estimación inicial, aunque el proyecto había alcanzado los objetivos marcados desde un principio, se decidió añadir una iteración más, principalmente para terminar de rematar algunos pequeño detalles, lo que ha supuesto la ampliación en dos semanas más, el tiempo de trabajo dedicado al Trabajo de Fin de Grado.

Tabla 2 - Ampliación calendario trabajo

Semana	Fecha	Sprint
15	18 Junio - 24 Junio	**8**
16	25 Junio - 1 Julio	**8**

Esta ampliación, hace que el número de horas final dedicado a este proyecto se eleve a 400 horas, 50 horas más de las inicialmente planificadas, aunque de haber sido necesaria la entrega del producto final, hubiese sido posible en plazo y forma.

5.2. Análisis de costes inicial

Para realizar una estimación inicial de los costes del proyecto es necesario tener en cuenta diferentes aspectos como:

- **Salario**

De las diferentes categorías salariales, la que más se ajusta a mi perfil es la de Analista programador.

Analista programador; Diseñador pagina web (Grupo III)

Tomada de [17]

Según la información proporcionada en el Boletín Oficial del Estado, del Martes 6 de marzo de 2018, del Ministerio de Empleo y Seguridad Social, para las TIC, el sueldo para un analista programador es de:

22.993,74

En base a eso:

22993,74 € / 12 meses / 4 semanas / 40 horas = 11,98 €/hora

Se obtiene que el coste por hora es aproximadamente de 11,98 € la hora, y como en la planificación inicial, la duración del proyecto se estima en 14 semanas, con una media de 25 horas semanales:

14 semanas x 25 horas/semana = 350 horas

Y por tanto:

350 horas x 11,98 €/hora = 4193 €

Dando como resultado un coste salarial aproximado de unos 4193 €.

- **Hardware**

El hardware empleado en el desarrollo del proyecto es de un PC con un coste de adquisición de 1200€, y de forma complementaria, un portátil Lenovo G50/70 con un coste de adquisición de 600€.

Aplicando el coeficiente de amortización lineal máximo obtenemos:

1200 € x 0,25 % = 300 €/año 300 € x $\frac{4}{12}$ meses = 100 €

En el caso del PC el coste es de 100€ aproximadamente.

600 € x 0,25 % = 150 €/año

150 € x $\frac{4}{12}$ meses = 50 €

En el caso del portátil el coste es de 50€ aproximadamente.

➢ **Licencias**

Todo el software que se va a utilizar a lo largo del desarrollo es gratuito y por lo tanto esta partida no representa ningún coste.

0 €

➢ **Suministros**

Dada la dificultad para realizar una estimación real sobre el coste que suponen gastos como pueden ser la electricidad o el agua, se establece una cantidad simbólica mensual en relación a estos costes.

5 € x 4 meses = 20 €

➢ **Factor multiplicativo**

Incremento sobre la estimación inicial en base a otros gastos relacionados, aplicando un 18%. De forma que la suma total de todos los puntos anteriores, dan como resultado:

4193 € + 100 € + 50 € + 0 € + 20 € = 4363 €

Aplicando el factor multiplicativo:

4363 € x 1,18 % = 5148,34 €

Por lo tanto, el presupuesto final es de:

Tabla 3 - Presupuesto costes proyecto

5148,34 €

5.2.1. Ajuste final de costes

Como consecuencia de la ampliación temporal es necesario reajustar los costes iniciales:

50 horas x 11,98 €/hora = 599 €
300 € x $\frac{0,5}{12}$ meses = 12,5 €
150 € x $\frac{0,5}{12}$ meses = 6,25 €
5 € x 0,5 meses = 2,5 €

Dando como resultado un total de:

599 € + 12,5 € + 6,25 € + 2,5 € = 620,25 €

Aplicando el factor multiplicativo:

620,25 € x 1,18 % = 731,90 €

Eso significa que ha habido un incremento en los costes de 731,90 €, dando un coste final total de:

Tabla 4 - Presupuesto final proyecto

5148,34 € + 731,90 € = 5880,24 €

5.3. Plan de riesgos

5.3.1. Matriz de riesgos

Tabla 5 - Matriz de riesgos

Impacto / Probabilidad	**Bajo**	**Medio**	**Alto**
Baja	Bajo	Bajo	Medio
Media	Bajo	Medio	Alto
Alta	Medio	Alto	Alto

Se prestará especial atención a aquellos riesgos de carácter alto, y respecto a los riesgos medios y bajos se monitorizara su evolución.

El proceso para la planificación de riesgos será el siguiente:

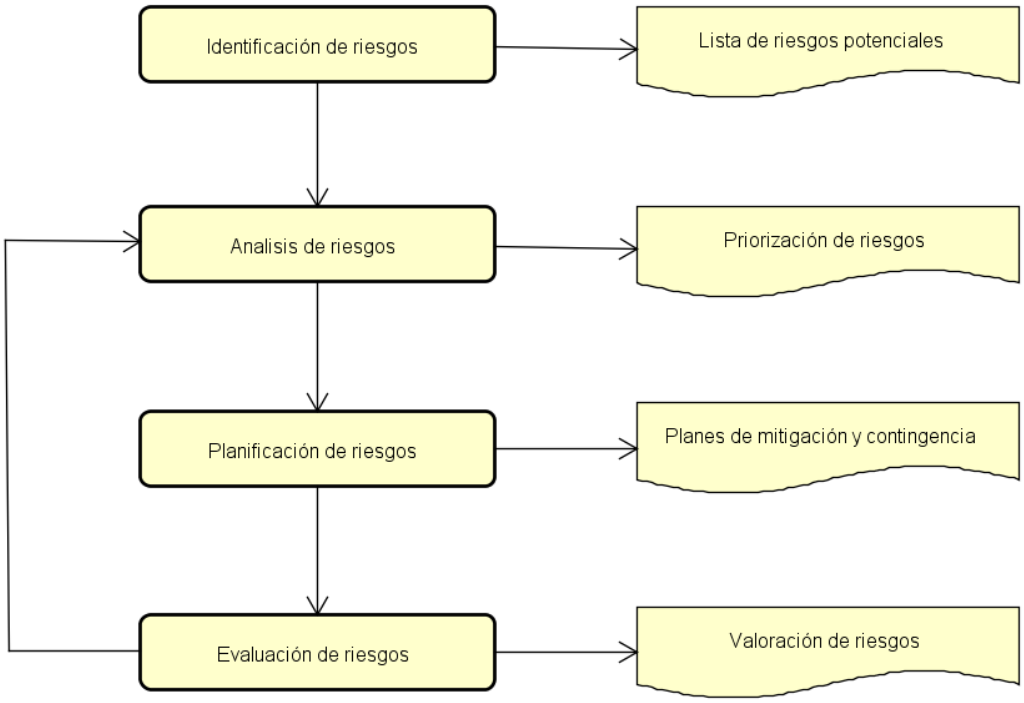


Ilustración 8 - Diagrama planificación de riesgos

5.3.2. Identificación, análisis y planificación

Tabla 6 - Riesgo 001

RISK-001	Retrasos en la planificación
Descripción	Retrasos en la planificación debido a una estimación incorrecta de los tiempos que no permita finalizar el proyecto en el plazo fijado
Probabilidad	Alta
Impacto	Alto
Plan de mitigación	Tener en cuenta, tanto en la planificación inicial como en las de asignación de tareas, los posibles retrasos, ampliando los tiempos en función del caso, de forma que permita afrontar las desviaciones sin retrasar la entrega final del proyecto
Plan de contingencia	Aumentar el trabajo diario hasta normalizar y alcanzar los objetivos en los tiempos establecidos, y en caso necesario, ampliar en una iteración más la duración del proyecto

Tabla 7 - Riesgo 002

RISK-002	Alcance del proyecto mal definido
Descripción	El alcance del proyecto no está bien definido o no es viable
Probabilidad	Baja
Impacto	Medio
Plan de mitigación	Evaluar al final de cada Sprint la evolución del proyecto
Plan de contingencia	Realizar reunión extraordinaria de todos los miembros del equipo para analizar el alcance actual del proyecto, y tomar las decisiones necesarias para definir correctamente su alcance

Tabla 8 - Riesgo 003

RISK-003	Conocimientos insuficientes / Falta de experiencia
Descripción	Tanto la falta de conocimientos técnicos específicos asociados a las nuevas tecnologías, herramientas, procedimientos, etc, como la falta de experiencia, puede suponer cometer un mayor número de errores, y ser menos productivo, produciendo retrasos en el proceso de desarrollo y un incremento de los costes
Probabilidad	Baja
Impacto	Alto
Plan de mitigación	Realizar investigación, documentación, tutoriales, pruebas tecnológicas. Disponer de formación continua, y acceso a plataforma de cursos online
Plan de contingencia	Aumentar el tiempo destinado a la formación

Tabla 9 - Riesgo 004

RISK-004	Problemas tecnológicos
Descripción	Debido a la tecnologías y herramientas empleadas, que no permitan alcanzar los objetivos deseados
Probabilidad	Baja
Impacto	Medio
Plan de mitigación	Investigar sobre las diferentes opciones disponibles
Plan de contingencia	Sustituir por otra opción alternativa

Tabla 10 - Riesgo 005

RISK-005	Medidas contra Web Scraping
Descripción	Existe la posibilidad de que las fuentes de datos implementen medidas de protección contra el Web Scraping
Probabilidad	Baja
Impacto	Alto
Plan de mitigación	Investigar si la fuente objetivo tiene implementada alguna medida contra el web scraping
Plan de contingencia	Cambiar de fuente de datos

Tabla 11 - Riesgo 006

RISK-006	Mala comunicación entre las partes
Descripción	Una mala comunicación, en tiempo o forma, puede producir efectos negativos en el desarrollo del proyecto
Probabilidad	Baja
Impacto	Alto
Plan de mitigación	Realizar reuniones continuas, a lo largo de todo el desarrollo del proyecto, entre todos los miembros del equipo (Reunión diaria, reunión al final de cada Sprint, etc.)
Plan de contingencia	Realizar reunión extraordinaria para plantear el problema, buscar una solución, y resolver el problema de comunicación

Tabla 12 - Riesgo 007

RISK-007	Mal diseño
Descripción	Definir una arquitectura inadecuada, o realizar un diseño incorrecto de los diferentes elementos, puede generar problemas graves en el desarrollo
Probabilidad	Baja
Impacto	Alto
Plan de mitigación	Seguir patrones de arquitectura y patrones de diseño
Plan de contingencia	Redefinir el diseño

Tabla 13 - Riesgo 008

RISK-008	Cambios en las historias de usuario
Descripción	Cambios significativos en las historias de usuario durante el proceso de desarrollo que impliquen consecuencias negativas
Probabilidad	Media
Impacto	Medio
Plan de mitigación	Realizar reuniones continuas, a lo largo de todo el desarrollo del proyecto, entre todos los miembros del equipo (Reunión diaria, reunión al final de cada Sprint, etc.)
Plan de contingencia	Evaluar la viabilidad para realizar o rechazar el cambio

Tabla 14 - Riesgo 009

RISK-009	Problemas Hardware / Software
Descripción	Fallos en los recursos físicos o software, puede producir retrasos en el desarrollo del proyecto
Probabilidad	Baja
Impacto	Bajo
Plan de mitigación	Realizar mantenimiento de equipos y los correspondientes backup
Plan de contingencia	Sustituir o reparar el recurso hardware/software

Tabla 15 - Riesgo 010

RISK-010	Baja calidad
Descripción	El resultado final del proyecto no cumple con los requisitos de calidad esperados
Probabilidad	Baja
Impacto	Alto
Plan de mitigación	Seguir estándares de calidad (Normas ISO)
Plan de contingencia	Corregir los errores y fallos principales hasta alcanzar unos niveles de calidad aceptables

5.3.3. Evaluación

Durante el desarrollo del proyecto se han producido dos de los riesgos potenciales:

- RISK-001: Retrasos en la planificación
- RISK-004: Problemas tecnológicos

Respecto a los retrasos en la planificación era de entre todos los riesgos, el que tenía más probabilidades de ocurrir, y en cierta manera, se podría haber evitado por completo, pero se decidió que teniendo la posibilidad de ampliar el desarrollo en una iteración más, no tenía mucho sentido forzar para terminar en el plazo estimado inicialmente, disponiendo de un margen temporal tan amplio.

Y en relación con los problemas tecnológicos, durante el desarrollo me he encontrado en un par de ocasiones con herramientas que aunque a priori parecían adecuadas, terminaron por demostrar ser insuficientes, por lo que procedí a sustituir la herramienta por otra alternativa.

Al ser riesgos contemplados desde el principio, con sus propios planes de mitigación y planes de contingencia, una vez se presentaron, la respuesta fue directa, evitando un impacto fuerte en el desarrollo del proyecto.

5.4. Product Backlog

Tabla 16 - Product Backlog

Product Backlog	
User Story	**Descripción**
US-001	Crear API REST
US-002	Crear scraper web
US-003	Acceder a fuente SiAR
US-004	Acceder a “Consulta de datos”
US-005	Seleccionar comunidad autónoma
US-006	Filtrar por provincia / comunidad de regantes
US-007	Seleccionar provincia
US-008	Seleccionar comunidad de regantes
US-009	Seleccionar estación
US-010	Seleccionar varias estaciones
US-011	Seleccionar tipo de dato para la consulta
US-012	Marcar periodo inicial
US-013	Marcar periodo final
US-014	Añadir parámetros de consulta
US-015	Seleccionar todos los parámetros de consulta
US-016	Deseleccionar todos los parámetros de consulta
US-017	Realizar consulta
US-018	Acceder a “Necesidades Netas”
US-019	Filtrar por comarca
US-020	Seleccionar cultivo
US-021	Obtener comunidades autónomas disponibles
US-022	Obtener provincias disponibles en una comunidad
US-023	Obtener comunidades de regantes disponibles en una comunidad

US-024	Obtener estaciones disponibles en una provincia
US-025	Obtener estaciones disponibles en una comunidad de regantes
US-026	Obtener cultivos disponibles de una estación por provincia
US-027	Obtener cultivos disponibles de una estación por comunidad de regantes
US-028	Almacenar datos en una base de datos
US-029	Obtener parámetros meteorológicos
US-030	Obtener variables para recomendaciones
US-031	Obtener longitud y latitud de las diferentes estaciones
US-032	Obtener otros datos de interés de las estaciones

5.5. Sprint Planning

Tabla 17 - Sprint planning 1

Sprint planning 1	
• Investigar sobre API REST y conceptos relacionados • Familiarización con Intellij IDEA, Spring Boot, Gradle, etc. • Implementar API REST • Realizar pruebas	
Entregable	Demo API REST que implemente los principales métodos

Tabla 18 - Sprint planning 2

Sprint planning 2	
• Investigar sobre web scraping y scrapers • Familiarización con diferentes páginas web, su estructura HTML, CSS, JavaScript, Jquery, etc • Familiarización con diferentes herramientas de web scraping como Jsoup, Jaunt, HtmlUnit, Selenium, etc • Implementar web scraper • Realizar pruebas	
Entregable	Demo Web Scraper con JSoup + Demo Web Scraper con HTMLUnit

Tabla 19 - Sprint planning 3

Sprint planning 3	
• Definir arquitectura del proyecto • Crear modelo de dominio • Investigar sobre TDD (Test-Driven Development), BDD (Behavior-Driven development), ATDD (Acceptance Test Driven Development), etc. • Iniciar desarrollo del proyecto web en Java, con Spring Boot, Selenium Web Driver, y API REST, aplicando los conocimientos adquiridos • Realizar pruebas	
Entregable	Propuesta arquitectura para el proyecto
Entregable	Versión 1.0 de la aplicación empleando técnicas de Web Scraper mediante Selenium Web Driver, con API REST básica

Tabla 20 - Sprint planning 4

Sprint planning 4	
• Realizar formalmente la elaboración de la documentación necesaria para el proyecto, y su revisión • Continuar con el desarrollo de la aplicación, implementando más funcionalidades • Realizar pruebas	
Entregable	Estructura básica del documento TFG del proyecto, versión 1, incluyendo los primeros apartados de planificación
Entregable	Versión 2.0, que permite realizar una consulta específica a partir de una serie de parámetros proporcionados a través de la API

Tabla 21 - Sprint planning 5

Sprint planning 5	
• Continuar con la documentación del proyecto • Continuar ampliando la aplicación implementando nuevas funcionalidades • Realizar pruebas	
Entregable	Documentación ampliada, versión 2
Entregable	Versión 3.0, que permite realizar consultas respecto a otros parámetros

Tabla 22 - Sprint planning 6

Sprint planning 6	
• Continuar con la elaboración de la documentación • Continuar con la implementación de nuevas funcionalidades • Integrar la base de datos PostgreSQL • Realizar pruebas	
Entregable	Documentación ampliada, versión 3
Entregable	Versión 4.0, que incluye mejoras en sus funcionalidades, refactorización del código, e inclusión de la persistencia de los datos

Tabla 23 - Sprint planning 7

Sprint planning 7	
• Ampliar la documentación • Ampliar funcionalidades para la obtención de información de las estaciones • Realizar la persistencia de datos • Realizar pruebas	
Entregable	Documentación ampliada, versión 4
Entregable	Versión 5.0, que añade la obtención de la latitud y longitud de las estaciones, y otra información relevante, además del refinamiento de la persistencia de datos

Tabla 24 - Sprint planning 8

Sprint planning 8	
• Revisar la documentación y añadir últimos detalles • Hacer últimos ajustes a la aplicación • Realizar últimas pruebas	
Entregable	Memoria final del Trabajo de Fin de Grado
Entregable	Entrega de la versión final de la aplicación

5.6. Historias de usuario

Tabla 25 - Historias de usuario

Historias de usuario	
US-001	Como analista de datos, quiero una API REST, para poder realizar consultas y peticiones mediante diferentes endpoints
US-002	Como analista de datos, quiero un scraper web, para poder obtener información directamente de diferentes páginas web
US-003	Como analista de datos, quiero acceder a la página SiAR como fuente principal de datos, para realizar una consulta de datos
US-004	Como analista de datos, quiero acceder a la página “Consulta de datos” de SiAR, para poder realizar una consulta a través de diferentes parámetros

US-005	Como analista de datos, quiero seleccionar la comunidad autónoma, para realizar una consulta de datos
US-006	Como analista de datos, quiero poder filtrar por provincia o comunidad de regantes, para realizar una consulta de datos
US-007	Como analista de datos, quiero seleccionar la provincia, para realizar una consulta de datos
US-008	Como analista de datos, quiero seleccionar la comunidad de regantes, para realizar una consulta de datos
US-009	Como analista de datos, quiero seleccionar la estación, para realizar la consulta de datos
US-010	Como analista de datos, quiero seleccionar varias estaciones, para realizar una consulta de datos de todas las estaciones
US-011	Como analista de datos, quiero seleccionar el tipo de dato de la consulta, para obtener la información diaria, semanal, o mensual
US-012	Como analista de datos, quiero fijar la fecha o periodo inicial, para realizar la consulta de datos
US-013	Como analista de datos, quiero fijar la fecha o periodo final, para realizar la consulta de datos
US-014	Como analista de datos, quiero seleccionar ciertos parámetros de consulta, como temperatura, humedad, velocidad del viento, dirección del viento, precipitación, etc., para obtener la información deseada

US-015	Como analista de datos, quiero seleccionar todos los parámetros de consulta, para realizar la consulta de datos
US-016	Como analista de datos, quiero deseleccionar todos los parámetros de consulta, para realizar la consulta de datos
US-017	Como analista de datos, quiero realizar la consulta de los datos a partir de todos los parámetros elegidos, para obtener la información necesaria
US-018	Como analista de datos, quiero acceder a la página “Necesidades netas” de SiAR, para poder realizar una consulta a través del tipo de cultivo
US-019	Como analista de datos, quiero filtrar por comarca, para realizar la consulta de datos
US-020	Como analista de datos, quiero seleccionar un cultivo para la comarca seleccionada, para obtener la información deseada
US-021	Como analista de datos, quiero obtener las comunidades autónomas disponibles, para conocer las diferentes opciones disponibles
US-022	Como analista de datos, quiero obtener las provincias disponibles en una comunidad, para conocer las diferentes opciones disponibles
US-023	Como analista de datos, quiero obtener las comunidades de regantes disponibles en una comunidad, para conocer las diferentes opciones disponibles

US-024	Como analista de datos, quiero obtener las estaciones disponibles en una provincia, para conocer las diferentes opciones disponibles
US-025	Como analista de datos, quiero obtener las estaciones disponibles en una comunidad de regantes, para conocer las diferentes opciones disponibles
US-026	Como analista de datos, quiero obtener los cultivos disponibles de una estación por provincia, para conocer las diferentes opciones disponibles
US-027	Como analista de datos, quiero obtener los cultivos disponibles de una estación por comunidad de regantes, para conocer las diferentes opciones disponibles
US-028	Como analista de datos, quiero poder almacenar la información obtenida de una consulta en una base de datos, para su acceso más rápido en una próxima consulta
US-029	Como analista de datos, quiero obtener los parámetros meteorológicos y guardar los datos en la base de datos, para poder recuperar la información de forma más rápida en una próxima consulta
US-030	Como analista de datos, quiero obtener ciertas variables como Kc, ETo, ETc o Pe, y guardar los datos en la base de datos, para analizar y procesar la información
US-031	Como analista de datos, quiero obtener la longitud y latitud de una estación concreta, para poder determinar la distancia respecto a un punto
US-032	Como analista de datos, quiero obtener toda la información disponible de una estación concreta, para poder estudiar la relación

	con determinadas variables

5.7. Sprint backlog

Tabla 26 - Sprint Backlog 1

Sprint backlog 1	
T-001	Realizar planificación temporal inicial
T-002	Realizar análisis de costes inicial
T-003	Identificar principales riesgos y elaborar plan de riesgos inicial
T-004	Instalar Sistema Operativo
T-005	Instalar/configurar entorno de trabajo
T-006	Investigar API RESTful
T-007	Investigar Java vs Groovy, Maven vs Gradle, SQL vs NoSQL, etc.
T-008	Crear y configurar proyecto Idea Intellij + Spring Boot
T-009	Realizar pruebas iniciales con las diferentes herramientas
T-010	Implementar método GET para API REST (US-001)
T-011	Implementar método POST para API REST (US-001)
T-012	Implementar método PUT para API REST (US-001)
T-013	Implementar método DELETE para API REST (US-001)
T-014	Realizar pruebas

Tabla 27 - Sprint Backlog 2

Sprint backlog 2	
T-015	Investigar sobre web scraping
T-016	Analizar estructura HTML/CSS página SiAR
T-017	Implementar web scraper con Jsoup (US-002)
T-018	Obtener información de diferentes páginas mediante JSoup
T-019	Realizar diferentes pruebas para JSoup
T-020	Implementar web scraper con JAunt
T-021	Analizar JavaScript/JQuery página SiAr
T-022	Implementar web scraper con HTMLUnit (US-002)
T-023	Obtener información de diferentes páginas mediante HTMLUnit
T-024	Realizar diferentes pruebas para HTMLUnit
T-025	Analizar alternativa Selenium

Tabla 28 - Sprint Backlog 3

Sprint backlog 3	
T-026	Definir arquitectura alto nivel
T-027	Definir arquitectura interna
T-028	Crear modelo de dominio
T-029	Investigar sobre TDD, BDD, ATDD, etc.
T-030	Implementar Web Scraper con Selenium Web Driver (US-002)
T-031	Diseñar pruebas iniciales
T-032	Integrar Base de Datos en memoria H2
T-033	Implementar acceder a diferentes páginas web (US-003, US-004, US-018)
T-034	Implementar seleccionar comunidad autónoma (US-005)
T-035	Implementar seleccionar Provincia (US-007)

Tabla 29 - Sprint Backlog 4

Sprint backlog 4	
T-036	Implementar filtrar por provincia / comunidad de regantes (US-006)
T-037	Implementar seleccionar comunidad de regantes (US-008)
T-038	Implementar seleccionar estación (US-009)
T-039	Implementar añadir más de una estación (US-010)
T-040	Implementar seleccionar tipo de dato a consultar (US-011)
T-041	Implementar fijar periodo inicial (US-012)
T-042	Implementar fijar periodo final (US-013)
T-043	Implementar añadir parámetros de consulta (US-014)
T-044	Implementar seleccionar todos los parámetros (US-015)
T-045	Implementar deseleccionar todos los parámetros (US-016)
T-046	Implementar realizar consulta (US-017)
T-047	Realizar pruebas
T-048	Generar documentación

Tabla 30 - Sprint Backlog 5

Sprint backlog 5	
T-049	Implementar acceder a Necesidades Netas (US-018)
T-050	Implementar filtrar por comarca (US-019)
T-051	Implementar seleccionar cultivo (US-020)
T-052	Implementar obtener todas las comunidades autónomas disponibles (US-021)
T-053	Implementar provincias disponibles en una comunidad (US-022)
T-054	Realizar pruebas
T-055	Generar documentación

Tabla 31 - Sprint Backlog 6

Sprint backlog 6	
T-056	Implementar comunidades de regantes disponibles en una comunidad (US-023)
T-057	Implementar estaciones disponibles en una provincia (US-024)
T-058	Implementar estaciones disponibles en una comunidades de regantes (US-025)
T-059	Implementar obtener cultivos disponibles de una estación por provincia (US-026)
T-060	Implementar obtener cultivos disponibles de una estación por comunidades de regantes (US-027)
T-061	Integrar PostgreSQL (US-028)
T-062	Refactorizar código
T-063	Realizar pruebas
T-064	Generar documentación

Tabla 32 - Sprint Backlog 7

Sprint backlog 7	
T-065	Implementar obtener latitud y longitud de las estaciones (US-031)
T-066	Implementar obtener otros parámetros de las estaciones (US-032)
T-067	Guardar comunidades en la base de datos (US-028)
T-068	Guardar provincias en la base de datos (US-028)
T-069	Guardar comunidades de regantes en la base de datos (US-028)
T-070	Guardar estaciones en la base de datos (US-028)
T-071	Guardar cultivos (US-028)
T-072	Guardar datos meteorológicos de la última semana disponible (US-029)
T-073	Guardar datos de las variables para las recomendaciones de la última semanas disponible de algunos cultivos
T-074	Realizar pruebas
T-075	Generar documentación

Tabla 33 - Sprint Backlog 8

Sprint backlog 8	
T-076	Realizar pequeños cambios en el código
T-077	Realizar última revisión de la documentación
T-078	Realizar pruebas finales

5.8. Especificación historias de usuario

En la especificación de las historias de usuario, han sido agrupadas aquellas historias de usuario que tienen un comportamiento similar por razones de simplificación.

Tabla 34 - US001

US-001	Disponer de una API		
Prioridad	**Como**	**Quiero poder**	**Para poder**
M	analista de datos	disponer de una API	realizar consultas y peticiones mediante diferentes endpoints
Escenario	Acceso satisfactorio a la página de inicio		
Dado que	escribo el endpoint /v1/index en la barra del navegador		
Cuando	realizó la petición de ir a la página de inicio		
Entonces	se debería mostrar la página de inicio de la API con un mensaje de bienvenida		
Escenario	Acceso insatisfactorio a la página de inicio		
Dado que	escribo incorrectamente el endpoint poniendo /v1/home en la barra del navegador		
Cuando	realizó la petición de ir a la página de inicio		
Entonces	se muestra un mensaje de error		

Escenario	Petición GET con éxito
Dado que	escribo un endpoint determinado /v1/resources
Cuando	realizo la petición GET
Entonces	se recibe el recurso solicitado con código de respuesta 200 (OK)
Escenario	Petición GET sin éxito
Dado que	escribo un endpoint determinado /v1/resources
Cuando	realizo la petición GET
Entonces	se produce un error, no se recibe el recurso solicitado, y se recibe código de respuesta 404 (Not Found)
Escenario	Petición POST con éxito
Dado que	escribo un endpoint determinado /v1/resources
Cuando	realizo la petición POST
Entonces	se crea un nuevo recurso con código de respuesta 201 (Created)
Escenario	Petición POST sin éxito
Dado que	escribo un endpoint determinado /v1/resources
Cuando	realizo la petición POST
Entonces	se produce un error, y se recibe el código de respuesta que lo genera, por ejemplo, 500 (Internal Server Error)
Escenario	Petición PUT con éxito

Dado que	escribo un endpoint determinado /v1/resources/{id}
Cuando	realizo la petición PUT
Entonces	se actualiza el recurso con código de respuesta 200 (Ok)
Escenario	Petición PUT sin éxito
Dado que	escribo un endpoint determinado /v1/resources/{id}
Cuando	realizo la petición PUT
Entonces	se produce un error, y se recibe el código de respuesta que lo genera, por ejemplo, 404 (Not Found)
Escenario	Petición DELETE con éxito
Dado que	escribo un endpoint determinado /v1/resources/{id}
Cuando	realizo la petición DELETE
Entonces	se elimina el recurso con código de respuesta 200 (Ok)
Escenario	Petición DELETE sin éxito
Dado que	escribo un endpoint determinado /v1/resources/{id}
Cuando	realizo la petición DELETE
Entonces	se produce un error, y se recibe el código de respuesta que lo genera, por ejemplo, 404 (Not Found)

Tabla 35 - US002

US-002	Crear scraper web		
Prioridad	**Como**	**Quiero poder**	**Para poder**
	analista de	extraer datos de	usar, procesar y almacenar

M	datos	una determinada página web mediante web scraping	información procedente de páginas web

Escenario	Extraer correctamente ciertos datos iniciales de diferentes páginas web
Dado que	escribo el endpoint /{recursos}
Cuando	realizó la petición GET
Entonces	se debería mostrar correctamente los datos solicitados en JSON

Escenario	Extraer incorrectamente los datos de las diferentes páginas web
Dado que	escribo el endpoint /{recursos}
Cuando	realizó la petición GET
Entonces	se muestran datos que no son correctos

Escenario	No es posible extraer datos de la página web
Dado que	escribo el endpoint /{recursos}
Cuando	realizó la petición GET
Entonces	se produce un error, y se recibe el código de respuesta que lo genera, por ejemplo, 500 (Internal Server Error)

Tabla 36 - US003/US004/US018

US-003 US-004 US-018	Acceder a una página específica		
Prioridad	**Como**	**Quiero poder**	**Para poder**
M	analista de datos	acceder a una página deseada	realizar un tipo de consulta específica en función de la fuente y unos parámetros

Escenario	Acceso satisfactorio a la página
Dado que	especifico una URL concreta al web scraper
Cuando	ejecuto el web scraper
Entonces	se debería mostrar la página a la que se accede
Escenario	Acceso insatisfactorio a la página
Dado que	especifico una URL concreta al scraper web
Cuando	ejecuto el web scraper
Entonces	se produce un error y no es posible acceder a la página

Tabla 37 - US006/US011/US019

US-006 US-011 US-019	Filtrar entre diferentes opciones		
Prioridad	**Como**	**Quiero poder**	**Para poder**
M	analista de datos	filtrar entre diferentes opciones	realizar una consulta de datos teniendo en cuenta la opción seleccionada
Escenario	Filtrado satisfactorio		
Dado que	especifico una opción concreta		
Cuando	ejecuto el web scraper		
Entonces	se selecciona la opción deseada		
Escenario	Filtrado insatisfactorio		

Dado que	especifico una opción concreta
Cuando	ejecuto el web scraper
Entonces	se produce un error y no se selecciona la opción deseada

Tabla 38 - US005/US007/US008/US009/US020

US-005 **US-007** **US-008** **US-009** **US-020**	Seleccionar un elemento concreto		
Prioridad	**Como**	**Quiero poder**	**Para poder**
M	analista de datos	seleccionar un elemento concreto entre diferentes opciones disponibles	usar el elemento al realizar una consulta de datos
Escenario	Elemento seleccionado correctamente		
Dado que	especifico un elemento concreto		
Cuando	ejecuto el web scraper		
Entonces	el elemento es seleccionado		
Escenario	Elemento no seleccionado		
Dado que	especifico un elemento concreto		
Cuando	ejecuto el web scraper		
Entonces	se produce un error y el elemento no es seleccionado		

Tabla 39 - US010

US-010	Seleccionar varias estaciones		
Prioridad	**Como**	**Quiero poder**	**Para poder**
M	analista de datos	seleccionar varias estaciones	realizar una consulta de datos de todas las estaciones al mismo tiempo
Escenario	Estaciones seleccionadas con éxito		
Dado que	selecciono varias estaciones		
Cuando	ejecuto el web scraper		
Entonces	las estaciones son seleccionadas correctamente		
Escenario	Estaciones seleccionadas sin éxito		
Dado que	selecciono varias estaciones		
Cuando	ejecuto el web scraper		
Entonces	se produce un error y las estaciones no son seleccionadas		

Tabla 40 - US012/013

US-012 US-013	Marcar periodo de tiempo		
Prioridad	**Como**	**Quiero poder**	**Para poder**
M	analista de datos	fijar una fecha o periodo determinado	realizar la consulta de datos entre las fechas deseadas
Escenario	Seleccionar un periodo de tiempo con éxito		
Dado que	especifico un periodo de tiempo determinado		
Cuando	ejecuto el web scraper		

Entonces	se selecciona la fecha deseada
Escenario	Seleccionar un periodo de tiempo determinado sin éxito
Dado que	especifico un periodo de tiempo determinado
Cuando	ejecuto el web scraper
Entonces	se produce un error y no se selecciona la fecha deseada

Tabla 41 - US014/015/016

US-015	Parámetros de consulta		
Prioridad	**Como**	**Quiero poder**	**Para poder**
M	analista de datos	seleccionar parámetros de consulta	realizar la consulta de datos y obtener la información deseada
Escenario	Seleccionar parámetros de consulta con éxito		
Dado que	especifico los parámetros de consulta		
Cuando	ejecuto el web scraper		
Entonces	se seleccionan los parámetros de consulta		
Escenario	Seleccionar parámetros de consulta sin éxito		
Dado que	especifico los parámetros de consulta		
Cuando	ejecuto el web scraper		
Entonces	se produce un error y no se seleccionan los parámetros de consulta		

Tabla 42 - US017

US-017	Realizar consulta		
Prioridad	**Como**	**Quiero poder**	**Para poder**
M	analista de datos	realizar una consulta de datos a partir de todos los parámetros seleccionados	obtener, procesar y almacenar la información necesaria
Escenario	Realizar consulta con éxito		
Dado que	realizo una consulta en función de una serie de opciones concretas		
Cuando	ejecuto el web scraper		
Entonces	se devuelve la información en formato JSON		
Escenario	Realizar consulta sin éxito		
Dado que	realizo una consulta en función de una serie de opciones concretas		
Cuando	ejecuto el web scraper		
Entonces	se produce un error, y no se realiza la consulta		

Tabla 43 - US021/US022/US023/US24/US025/US026/US027

US-021 US-022 US-023 US-024 US-025 US-026 US-027	Obtener opciones disponibles		
Prioridad	**Como**	**Quiero poder**	**Para poder**
M	analista de datos	obtener las opciones disponibles de una categoría	tener conocimiento de las diferentes opciones disponibles

Escenario	Consulta de opciones disponibles con éxito
Dado que	solicito información sobre las opciones disponibles para una categoría
Cuando	ejecuto el web scraper
Entonces	se recibe la información deseada en JSON
Escenario	Consulta de opciones disponibles sin éxito
Dado que	solicito información sobre las opciones disponibles para una categoría
Cuando	ejecuto el web scraper
Entonces	se produce un error, y se recibe el código de respuesta que lo genera

Tabla 44 - US028

US-028	Integrar base de datos		
Prioridad	**Como**	**Quiero poder**	**Para poder**
M	analista de datos	almacenar la información obtenida de una consulta en una base de datos	para poder disponer y recuperar la información de forma más rápida en una próxima consulta
Escenario	Guardar información en la base de datos con éxito		
Dado que	realizo una consulta		
Cuando	la consulta se ha realizado con éxito		
Entonces	la información se almacena correctamente en la base de datos		
Escenario	Guardar información en la base de datos con éxito		

Dado que	realizo una consulta
Cuando	la consulta se ha realizado con éxito
Entonces	se produce un error, y no es posible almacenar la información

Tabla 45 - US029/US030

US-029 US-030	Parámetros y variables		
Prioridad	**Como**	**Quiero poder**	**Para poder**
M	analista de datos	obtener diferentes parámetros/variables y almacenarlos en la base de datos	analizarlos y procesarlos si se desea
Escenario	Obtener parámetros/variables con éxito		
Dado que	Realizo una consulta		
Cuando	la consulta se ha realizado con éxito		
Entonces	Se obtiene la información en JSON y se guarda en la base de datos		
Escenario	Obtener parámetros/variables sin éxito		
Dado que	Realizo una consulta		
Cuando	la consulta se ha realizado con éxito		
Entonces	se produce un error, y no es posible obtener la información ni guardarla en la base de datos		

Tabla 46 - US030/US031

US-030 US-031	Obtener longitud/latitud y otra información		
Prioridad	**Como**	**Quiero poder**	**Para poder**

M	analista de datos	obtener la longitud y latitud de una estación concreta, además de otra información de interés	tener conocimiento de las estaciones más cercanas a un punto, y las características propias de una estación

Escenario	Obtener información de una estación con éxito
Dado que	solicito la información de una estación
Cuando	ejecuto el web scraper
Entonces	se obtiene la información en formato JSON
Escenario	Obtener información de una estación sin éxito
Dado que	solicito la información de una estación
Cuando	ejecuto el web scraper
Entonces	se produce un error, y se recibe el código de respuesta que lo genera

A continuación, algunos ejemplos de test que se han desarrollado para comprobar que se cumplen los criterios de aceptación:

```
@Test
@UserStory("US-001")
public void shouldGetWelcomeMessageFromHomeController() throws Exception {
    mockMvc.perform(get( restEndpoint: "/v1/index"))
            .andExpect(content().string( expectedContent: "Welcome to your API REST to get meteorological data"));
}

@Test
@UserStory("US-001")
public void shouldReturn404WhenDoAGetToAnInvalidResourcePath() throws Exception {
    mockMvc.perform(get( restEndpoint: "/v1/home"))
            .andExpect(status().isNotFound());
}

@Test
@UserStory("US-001")
public void shouldReturn200WhenDoAGetToAValidResourcePath() throws Exception {
    mockMvc.perform(get( restEndpoint: "/v1/index"))
            .andExpect(status().isOk());
}
```

Ilustración 9 - Test US001

```
@Test
@UserStory("US-002")
public void shouldScrapeDataFromAWebsite() throws Exception {
    mockMvc.perform(get( restEndpoint: "/v1/irrigation-communities/Aleje?communityName=Castilla y Leon&ws=true"))
            .andExpect(jsonPath( expression: "$.name", equalTo( operand: "Aleje")))
            .andExpect(jsonPath( expression: "$.autonomousCommunity.name", equalTo( operand: "Castilla y León")));
}
```

Ilustración 10 - Test US002

```
@Test
@UserStory({"US-003", "US-004", "US-018"})
public void shouldScraperConnectToAValidWebsite() {
    try (Scraper scraper = source("https://www.uva.es")) {
        assertThat(scraper.getTitle(), equalTo( operand: "Universidad de Valladolid - SSO"));
    }
}

@Test
@UserStory({"US-003", "US-004", "US-018"})
public void shouldReturnAnErrorWhenScraperConnectToAnInvalidWebsite() {
    try (Scraper scraper = source("https://www.webSite.that.doesNotExist")) {
    } catch (WebDriverException ex) {
        assertThat(ex.getMessage(), startsWith("Reached error page: about:neterror?e=dnsNotFound&u=https%3A//www.website.that.doesnotexist"));
    }
}
```

Ilustración 11 - Test US003/US004/US018

```
@Test
@UserStory({"US-005", "US-007", "US-008", "US-009", "US-020"})
public void shouldSelectAnOptionOfASelectElement() {
    try (Scraper scraper = source(ScrapingUrl.getUrlDq)) {
        scraper.doAction(selectAuCommunityAction( autonomousCommunityName: "Castilla y León"));
    }
}

@Test
@UserStory({"US-005", "US-007", "US-008", "US-009", "US-020"})
public void shouldThrowAnExceptionWhenTryingToSelectAnInvalidOptionOfASelectElement() {
    try (Scraper scraper = source(ScrapingUrl.getUrlDq)) {
        scraper.doAction(selectAuCommunityAction( autonomousCommunityName: "aCommunityThatDoesNotExist"));
    } catch (IllegalArgumentException ex) {
        assertThat(ex.getMessage(), equalTo( operand: "ERROR: Incorrectly introduced AC"));
    }
}
```

Ilustración 12 - Test US005/US007/US008/US009/US020

```
@Test
@UserStory({"US-006", "US-011", "US-019"})
public void shouldFilterUsingAnOptionOfASelectElement() {
    try (Scraper scraper = source(ScrapingUrl.getUrlDq)) {
        scraper.doAction(selectIrrigationCommunityRadioButton())
                .doAction(selectIrrigationCommunityAction("Alamo Gordo"));
    }
}

@Test
@UserStory({"US-006", "US-011", "US-019"})
public void shouldThrowAnExceptionWhenTryingToFilterUsingAnInvalidOptionOfASelectElement() {
    try (Scraper scraper = source(ScrapingUrl.getUrlDq)) {
        scraper.doAction(selectIrrigationCommunityRadioButton())
                .doAction(selectIrrigationCommunityAction("aOptionDoesNotExist"));
    } catch (IllegalArgumentException ex) {
        assertThat(ex.getMessage(), equalTo( operand: "Error: Incorrectly introduced irrigation community"));
    }
}
```

Ilustración 13 - Test US006/US011/US019

```java
@Test
@UserStory("US-010")
public void shouldSelectSeveralStations() {
    try (Scraper scraper = source(ScrapingUrl.getUrlDq)) {
        scraper.doAction(addStation("Adra"));
    }
}

@Test
@UserStory("US-010")
public void shouldThrowAnExceptionWhenSelectAnInvalidStation() {
    try (Scraper scraper = source(ScrapingUrl.getUrlDq)) {
        scraper.doAction(addStation("aStationThatDoesNotExist"));
    } catch (IllegalArgumentException ex) {
        assertThat(ex.getMessage(), equalTo( operand: "ERROR: aStationThatDoesNotExist - Incorrectly introduced station"));
    }
}
```

Ilustración 14 - Test US010

```java
@Test
@UserStory({"US-012", "US-013"})
public void shouldSpecifyADate() {
    try (Scraper scraper = source(ScrapingUrl.getUrlDq)) {
        scraper.doAction(setStartDate(LocalDate.now()));
    }
}
```

Ilustración 15 - Test US012/US013

```java
@Test
@UserStory("US-017")
public void shouldRetrieveDataSuccessfullyAfterDoAMeasurementsRequest() throws Exception {
    mockMvc.perform(get( restEndpoint: "/v1/stations/Arroyo/measurements?startDate=01052018&endDate=01052018"))
            .andExpect(jsonPath( expression: "$", hasSize(1)))
            .andExpect(jsonPath( expression: "$[0].date", equalTo( operand: "2018-05-01")));
}

@Test
@UserStory("US-017")
public void shouldReturnAnErrorWhenMeasurementsRestResourceInvocationIsWrong() throws Exception {
    mockMvc.perform(get( restEndpoint: "/v1/stations/Arroyo/measurements?startDate=00052018&endDate=01052018"))
            .andExpect(jsonPath( expression: "$.error", equalTo( operand: "Invalid provided date (00052018)")));
}

@Test
@UserStory("US-017")
public void shouldRetrieveDataWithAllParametersAfterDoAMeasurementsRequest() throws Exception {
    mockMvc.perform(get( restEndpoint: "/v1/stations/Arroyo/measurements?startDate=01052018&endDate=01052018"))
            .andExpect(jsonPath( expression: "$", hasSize(1)))
            .andExpect(jsonPath( expression: "$[0].averageTemp", not(isEmptyOrNullString())))
            .andExpect(jsonPath( expression: "$[0].maxTemp", not(isEmptyOrNullString())))
            .andExpect(jsonPath( expression: "$[0].hourMaxTemp", not(isEmptyOrNullString())))
            .andExpect(jsonPath( expression: "$[0].minTemp", not(isEmptyOrNullString())))
            .andExpect(jsonPath( expression: "$[0].hourMinTemp", not(isEmptyOrNullString())))
            .andExpect(jsonPath( expression: "$[0].averageHum", not(isEmptyOrNullString())))
            .andExpect(jsonPath( expression: "$[0].maxHum", not(isEmptyOrNullString())))
            .andExpect(jsonPath( expression: "$[0].hourMaxHum", not(isEmptyOrNullString())))
            .andExpect(jsonPath( expression: "$[0].minHum", not(isEmptyOrNullString())))
            .andExpect(jsonPath( expression: "$[0].hourMinHum", not(isEmptyOrNullString())))
            .andExpect(jsonPath( expression: "$[0].speedWind", not(isEmptyOrNullString())))
            .andExpect(jsonPath( expression: "$[0].directionWind", not(isEmptyOrNullString())))
            .andExpect(jsonPath( expression: "$[0].maxSpeedWind", not(isEmptyOrNullString())))
            .andExpect(jsonPath( expression: "$[0].hourMaxSpeedWind", not(isEmptyOrNullString())))
            .andExpect(jsonPath( expression: "$[0].directionMaxSpeedWind", not(isEmptyOrNullString())))
            .andExpect(jsonPath( expression: "$[0].radiation", not(isEmptyOrNullString())))
            .andExpect(jsonPath( expression: "$[0].rainfall", not(isEmptyOrNullString())))
            .andExpect(jsonPath( expression: "$[0].rainfallEffect", not(isEmptyOrNullString())))
            .andExpect(jsonPath( expression: "$[0].eto", not(isEmptyOrNullString())));
}
```

Ilustración 16 - Test US017

```
@Test
@UserStory({"US-029", "US-030"})
public void shouldRetrieveCorrectlyDataFromVariablesRestResource() throws Exception {
    mockMvc.perform(get( restEndpoint: "/v1/stations/Arroyo/crops/Arroz/variables?startDate=01052018&endDate=03052018"))
            .andExpect(jsonPath( expression: "$", hasSize(3)))
            .andExpect(jsonPath( expression: "$[0].date", equalTo( operand: "2018-05-01")))
            .andExpect(jsonPath( expression: "$[1].date", equalTo( operand: "2018-05-02")))
            .andExpect(jsonPath( expression: "$[2].date", equalTo( operand: "2018-05-03")));
}

@Test
@UserStory({"US-029", "US-030"})
public void shouldReturnAnErrorWhenVariablesRestResourceInvocationIsWrong() throws Exception {
    mockMvc.perform(get( restEndpoint: "/v1/stations/Arroyo/crops/Arroz/variables?startDate=WRONG_DATE&endDate=03052018"))
            .andExpect(jsonPath( expression: "$.error", equalTo( operand: "Invalid provided date (WRONG_DATE)")));
}

@Test
@UserStory({"US-030", "US-031"})
public void shouldRetrieveLongitudeAndLatitudeFromAnExistingStation() throws Exception {
    mockMvc.perform(get( restEndpoint: "/v1/stations/Adra"))
            .andExpect(jsonPath( expression: "$.utmx", notNullValue()))
            .andExpect(jsonPath( expression: "$.utmy", notNullValue()));
}

@Test
@UserStory({"US-030", "US-031"})
public void shouldReturnAnErrorWhenRequestStationDoesNotExist() throws Exception {
    mockMvc.perform(get( restEndpoint: "/v1/stations/INVALID_STATION"))
            .andExpect(jsonPath( expression: "$.error", equalTo( operand: "Station 'INVALID_STATION' not found")));
}
```

Ilustración 17 - Test US029/US030

6. Desarrollo del proyecto

A lo largo de este punto explicare algunos de los aspectos más relevantes en el proceso de desarrollo de la aplicación.

En cada Sprint se ha pasado por las siguientes fases:

- Análisis
- Diseño
- Implementación
- Pruebas

La intensidad de cada una de ellas ha variado en función de la iteración y de las necesidades.

6.1. Iteración 1

El objetivo principal para esta primera iteración es crear una API REST, que implemente sus principales métodos, es decir, los métodos GET, POST, PUT y DELETE.

6.1.1. Análisis

Empecemos por definir que es una API, abreviatura de Application Programming Interface, que traducido sería interfaz de programación de aplicaciones, y explicado de forma simple, no es más que un programa que permite la comunicación entre dos aplicaciones distintas, proporcionando una capa de abstracción.

El concepto REST proviene de REpresentational State Transfer, cuya traducción sería transferencia de estado representacional, y es un conjunto de principios para crear un estilo de arquitectura en la definición de la comunicación entre un cliente y un servidor, a través de la red, bajo el protocolo HTTP.

Algunas de sus principales características son:

- Arquitectura Cliente/Servidor: donde el cliente es el que solicita la información, y el servidor el que provee la información
- Sin estado: eso significa que cada petición contiene toda la información necesaria para ser procesada, y no se guarda el estado de las comunicaciones
- Operaciones: que se aplican por igual a todos los recursos, y las más importantes son las siguientes::

Tabla 47 - Métodos API

❖ GET: Leer ❖ POST: Crear ❖ PUT: Actualizar ❖ DELETE: Borrar

➢ URL: el acceso a los recursos es mediante una URL (Uniform Resource Locator) o localizador de recursos uniforme, que permite localizar a un recurso, mediante una dirección con el formato protocolo/servidor/ruta-recurso, que a modo de ejemplo podría ser:

http://www.apiunificada.org/memoria.pdf

Sumando todo, una API REST es una interfaz que facilita el intercambio de información entre un cliente y un servidor, por medio de la red, mediante una URL (URI), usando HTTP, y que permite leer, crear, actualizar y borrar información.

En esta primera fase en la que me voy a centrar en el desarrollo de la API, no plantearé aún el modelo de dominio para la aplicación, y no tiene mucho sentido incluir la API como elemento de dominio ya que se trata de algo más relacionado con la implementación, por lo que usare a modo de ejemplo para su construcción, un "recurso" genérico que me servirá en la etapa de análisis y diseño para modelar.

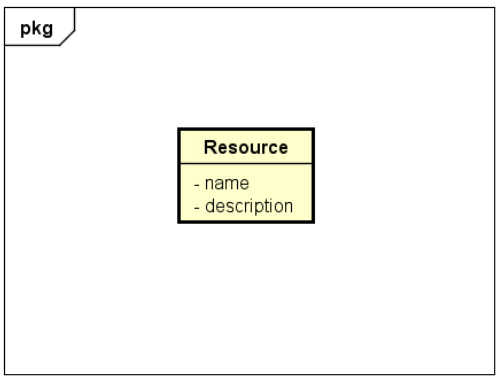


Ilustración 18 - Modelo Resource I1

Imaginamos un recurso cualquiera con dos atributos, el cual será la base sobre el que realizaran las diferentes peticiones a través de la API, como preparación para las siguientes iteraciones.

6.1.2. Diseño

No existe un estándar como tal en el desarrollo de API REST, pero si una serie de buenas prácticas como pueden ser:

- Usar nombres y no verbos

/recursos y no /~~obtenerRecursos~~

- Siempre en plural, con independencia de si solo nos referimos a un elemento individual.

/recursos

- Anidación de recursos: basada en jerarquía, por ejemplo, si queremos el sub-recurso 3 del recurso 2, seria:

/recursos/2/sub-recurso/3

- Códigos HTTP: usar de forma correcta los códigos de estado cuando se realiza una petición, algunos de los principales son:

Tabla 48 - Códigos de respuesta HTTP

• 200 (Ok): la solicitud se realiza con éxito • 201 (Created): creado con éxito un nuevo recurso • 400 (Bad Request): sintaxis incorrecta • 401 (Unauthorized): sin autorización para la petición • 404 (Not found): recurso no encontrado • 405 (Method Not Allowed): no admite el método HTTP solicitado • 500 (Internal Server Error): error interno en el servidor

- Versionar API: a través de la URL

/v1/recursos

- Devolver JSON: preferiblemente a XML

- Documentar API: se dice que una API es tan buena como lo es su documentación, dado que una API es la formalización de un contrato, esta debe contar con una buena documentación, precisa y detallada

- "Keep It Simple": se recomienda seguir una regla esencial, mantenlo simple

Retomando el ejemplo del recurso en la fase de análisis, tenemos un recurso con sus respectivos atributos y operaciones, y evidentemente al tratarse de solo un concepto, no es posible disponer de relaciones.

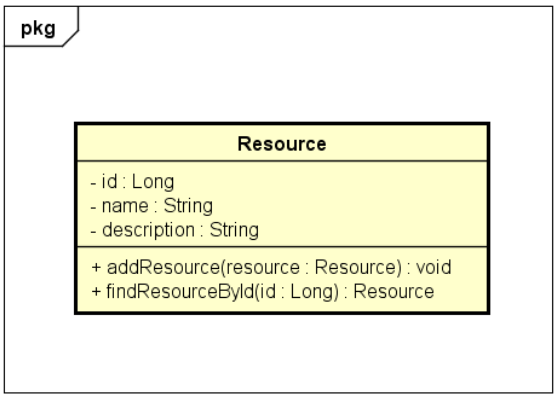


Ilustración 19 - Diseño Resource I1

Planteo una arquitectura física como la siguiente:

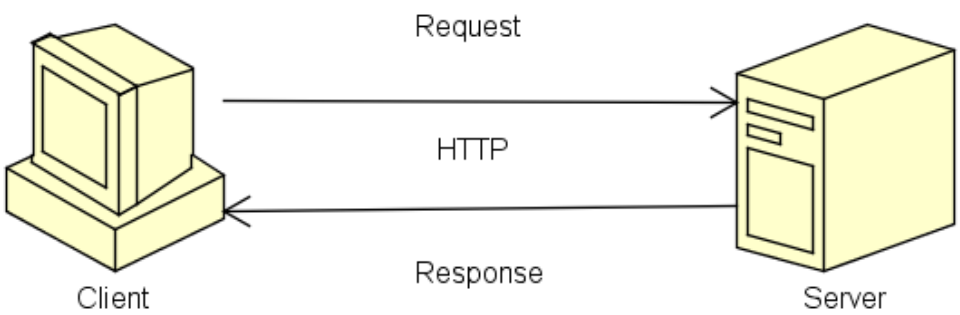


Ilustración 20 - Arquitectura física

Una vez que ya está definido lo que queremos, por un lado un recurso concreto, y por otro, los métodos de la API, paso a la fase de implementación.

6.1.3. Implementación

Antes de nada, lo primero que necesito es crear mi primer proyecto Spring Boot, al final de esta memoria, he incluido en la sección de Anexos, los pasos a seguir para crear un proyecto Spring Boot.

Una vez creado, tenemos un proyecto con una arquitectura lógica básica como la siguiente:

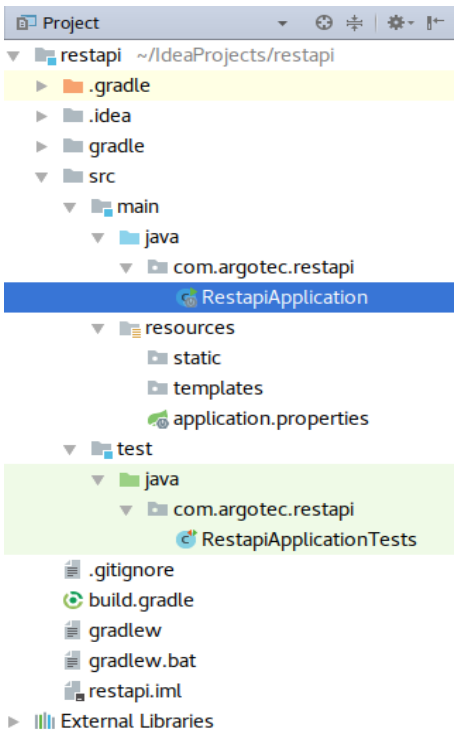


Ilustración 21 - Spring Boot: Estructura básica proyecto

Como toda aplicación Java, necesita una clase principal con el método main, y en este caso tenemos como clase principal:

```
@SpringBootApplication
public class RestapiApplication {

    public static void main(String[] args) {
      SpringApplication.run(RestapiApplication.class, args);
  }
}
```

La clase principal tiene la anotación @SpringBootApplication, que engloba a otras tres anotaciones más, @Configuration, @EnableAutoConfiguration y @ComponentScan.

- @Configuration: contiene la configuración principal
- @EnableAutoConfiguration: aplica la configuración automática
- @ComponentScan: localiza elementos con ciertas anotaciones

El fichero de dependencias "build.gradle" queda configurado con las siguientes dependencias:

```
dependencies {

    compile('org.springframework.boot:spring-boot-starter-web')
    compile('org.springframework.boot:spring-boot-starter-data-jpa')

    runtime('com.h2database:h2')

    testCompile('org.springframework.boot:spring-boot-starter-test')
}
```

Las dependencias se corresponden con:

- compile('org.springframework.boot:spring-boot-starter-web'): importa los módulos relacionados con los proyectos web.
- compile('org.springframework.boot:spring-boot-starter-data-jpa'): importa los módulos para la persistencia con JPA.
- runtime('com.h2database:h2'): importa los módulos para la base de datos H2
- testCompile('org.springframework.boot:spring-boot-starter-test'): importa los módulos para los test Spring Boot, JUnit, AssertJ, Hamcrest, Mockito, y otras bibliotecas

Planteo un modelo basado en capas, para una máxima cohesión y un mínimo acoplamiento:

- Capa de presentación: es el front-end, la interfaz gráfica.
- Capa de negocio: es la que contiene el conocimiento y las reglas de negocio, compuesta por los servicios.
- Capa de acceso a datos: clases encargadas de almacenar y extraer los datos de un repositorio.

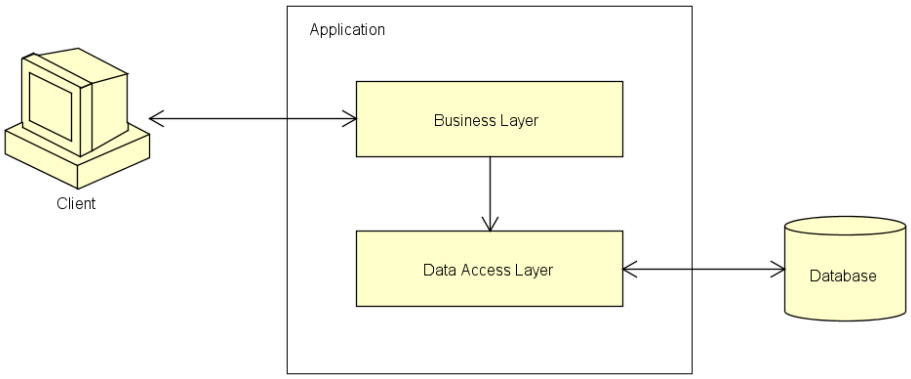
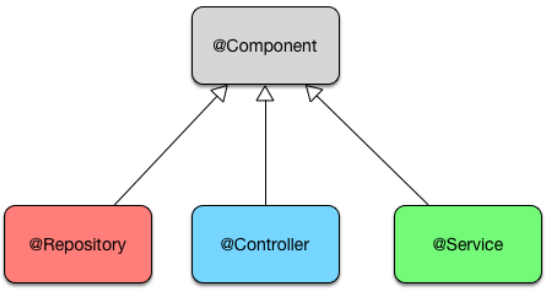


Ilustración 22 – Modelo por capas

Por el alcance de este proyecto, la capa de presentación será una línea futura, ya que aunque la API es la parte más externa de la capa de negocio, sigue formando parte de la capa de negocio, y aunque el usuario al acceder a la API recibe un mensaje de bienvenida, es solamente algo simbólico.

Una de las ventajas de usar Spring Boot es que implementa diferentes patrones de diseño y dispone de una serie de estereotipos que asocian a cada componente una responsabilidad concreta.



Tomada de [18]

Los estereotipos son:

- @Component: indica que la clase es un componente, sin especificar de que tipo
- @Controller: se encarga de gestionar la comunicación entre el usuario y la aplicación
- @Service: se encarga de gestionar la lógica de negocio, y llama a los repositorios
- @Repository: implementa el patrón repositorio, y es el encargado de almacenar los datos en la base de datos

A modo de ejemplo, implementare la API REST para un recurso concreto, así que lo primero es crear el modelo mediante la clase Resource, incluyendo al diseño un atributo más, el id, junto al resto de sus atributos, sus constructores, getters and setters, y toString().

```
@Entity
@Table("Resource")
public class Resource {

    @Id
    @GeneratedValue(strategy=  GenerationType.AUTO)
    private  Long id;

    private  String name;
    private  String description;

    public Resource (){
    }

    public Resource (String name, String description) {
        this. name = name;
        this. description = description;
    }

    public Long getId() {
        return id;
    }

    public void  setId(long id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void  setName (String name) {
        this.name = name;
    }

    public String getDescription () {
        return description;
    }

    public void  setDescription (String description) {
        this.description = description;
    }

    @Override
    public String toString() {
        return String.format(
                "Resource [id=%d, name ='%s', description ='%s']",
                id, name, description);
    }
}
```

Me valgo de una serie de anotaciones útiles como son:

- @Entity: especifica que la clase es una identidad
- @Table: especifica que la clase se va a persistir con un nombre determinado
- @Id: asocia el atributo como identificador único de la clase
- @GeneratedValue: para la generación automática de los id
- @Override: para sobrescribir el comportamiento de un método existente

Ahora creo la clase ResourceRepository, para la capa repositorio, que con Spring Boot y JPA, se simplifica bastante la tarea, siendo necesario solo definir la interface, y en este caso, además definir los dos métodos addResource() y findOneResource().

```
@Repository
public interface ResourceRepository extends JpaRepository<Resource, Long> {

    void addResource(Resource resource);
    Resource findResourceById(Long id);
}
```

Las clases que forma parte de @Controller, solo tendrían que tener acceso a las clases @Service mediante interfaces, y de igual forma, el acceso entre clases @Service y al @Repository debería ser también mediante interfaces. Defino la interface para la clase ResourceService:

```
public interface ResourceService {

  void addResource(Resource resource);
  Resource findResourceById(Long id);

}
```

Y después paso a la implementación de ResourceServiceImpl:

```
@Service
public class ResourceServiceImpl implements ResourceService {

   private final Resource resource;
```

```
    public ResourceServiceImpl (ResourceRepository resourceRepository) {
        this.resourceRepository = resourceRepository;
    }

    @Override
    public void addResource(Resource resource){
        resourceRepository.save(resource);
    }

    @Override
    public Resource findResourceById(Long id){
        return resourceRepository.findById (id);
    }
}
```

Por último, le toca el turno al controlador, a continuación muestro en modo de ejemplo una implementación básica para los métodos GET y POST que son los más importantes.

```
@RestController
public class Resource Controller {

    private final ResourceService resourceService;

    public ResourceController(ResourceService resourceService) {
        this.resourceService = resourceService;
    }

    @RequestMapping(value = "/resources/{id}", method = RequestMethod.GET)
    public ResponseEntity<Resource> getResourceById (@PathVariable Long id) {
        Resource resource = resourceService.findResourceById (id);
        return new ResponseEntity (resource, HttpStatus.OK);
    }

    @RequestMapping(value = "/resources/{id}", method = RequestMethod.POST)
    public ResponseEntity<Resource> createResource (@RequestBody Resource resource) {
        Resource resource = resourceService. addResource (resource);
        return ResponseEntity (resource, HttpStatus.CREATED);
    }
}
```

Hago uso de las anotaciones @RestController, y @RequestMapping

- @RestController: especialización de @Controller, para servicios REST
- @RequestMapping: se encarga de relacionar un método con una petición HTTP

- @ResponseBody: define el cuerpo de la respuesta del servicio
- @PathVariable: indica que variable de la URL se selecciona como parámetro
- @RequestBody: mapear el JSON

Ilustración 23 - @Controller/@Service/@Repository

Otra anotación a destacar, es @Autowired, responsable de la inyección de dependencias sobre un atributo.

La estructura final de paquetes queda de la siguiente forma:

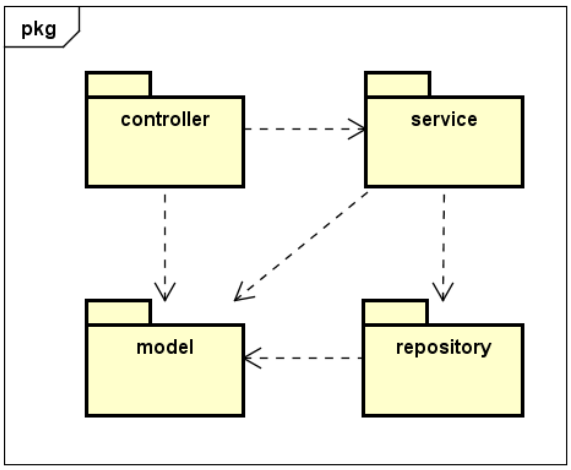


Ilustración 24 - Diagrama de paquetes I

Donde tenemos una separación por capas y un modelo que es transversal a todas:

- controller: contiene las clases encargadas de mapear los recursos de la API
- service: contiene las clases de la lógica de negocio
- repository: contiene las clases de acceso a datos
- model: contiene las entidades de la aplicación

Una vez creadas las clases, se necesitan algunos ejemplos de muestra, para lo que es necesario inicializar la base de datos en memoria H2, y utilizando la interfaz “CommandLineRunner”, se puede

añadir algunos datos de muestra, la cual se usa para indicar que un bean debe ejecutarse cuando está contenido dentro de una aplicación Spring.

```
@Bean
public CommandLineRunner setup(ResourceRepository resourceRepository) {
        return (args) -> {

            resourceRepository.save(new Resource ("R1", ”MasImportante”));
            resourceRepository.save(new Resource ("R2", ”MedioImportante”));
            resourceRepository.save(new Resource ("R3", “MenosImportante”));
      };
}
```

. Y como diagrama de despliegue:

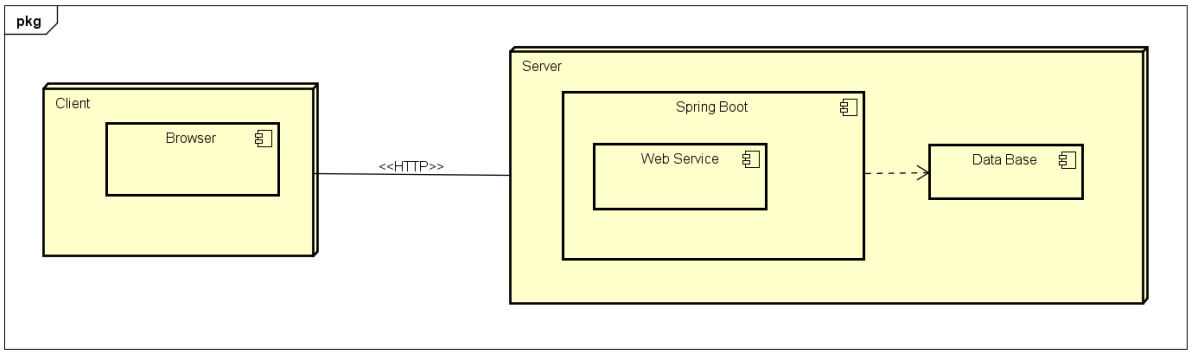


Ilustración 25 – Diagrama de despliegue I1

6.1.4. Pruebas

En esta primera iteración el proceso de pruebas ha sido general, por un lado comprobando el funcionamiento de las propias tecnologías y herramientas empleadas, y por otro, comprobando el comportamiento de la API REST bajo ejemplos reales.

Una vez implementada la API, es el momento de probar que todo funciona correctamente, así que mediante Postman, pruebo uno por uno cada uno de los endpoint, comprobando que los resultados son los esperados.

Retomando el ejemplo de las fases anteriores, si queremos realizar una petición GET de un recurso concreto, por ejemplo, un recurso con un id igual a 3, escribo el endpoint correspondiente:

```
/resources/{3}
```

Se espera obtener el recurso con id=3, y si el recurso existe, y la solicitud se realiza con éxito, obtendremos el recurso en formato JSON, con el código de respuesta 200 (OK). Por el contrario, si sucede algo, como que el recurso no existe, entonces no será posible obtener el recurso, y recibiremos un código de respuesta 404 (Not Found). En la siguiente tabla muestro los códigos de respuesta posibles en caso de que la petición se procese con éxito.

Tabla 49 - Códigos de respuesta favorables

Método	URL	Descripción	
GET	/resources	Obtiene una lista con todos los recursos	200 (OK)
GET	/resources/{id}	Obtiene el recurso con el id especificado	200 (OK)
POST	/resources	Se crea un nuevo recurso	201 (Created)
PUT	/resources/{id}	Actualiza un recurso existente	200 (OK) 201 (Created)
DELETE	/resources/{id}	Elimina un recurso	200 (OK) 202 (Accepted) 204 (No Content)

El resultado final de esta iteración ha sido disponer de una API completamente funcional.

6.2. Iteración 2

En esta segunda iteración el principal objetivo es crear un web scraper comprobando y demostrando la viabilidad del proyecto, debido a que la obtención futura de toda la información se realizará mediante técnicas de web scraping.

La creación de scrapers y la aplicación de técnicas de web scraping, es uno de los pilares fundamentales de este proyecto, y por un lado, le suma cierto grado de complejidad, y por otro, hace el proyecto más interesante y atractivo.

Empecemos por definir web scraping, la cual es una técnica que consiste en extraer información de sitios web mediante programas software. Estos programas, simulan la navegación humana dentro de la red obteniendo información de forma automática.

La idea es transformar datos no estructurados, como los que se encuentran en el código HTML de una página web, en datos estructurados que se puedan almacenar en una base de datos para su posterior análisis.

El web scraping está relacionado con los campos de la Inteligencia Artificial y el Big Data, en lo que se refiere a la extracción de datos, procesamiento, almacenamiento y visualización.

6.2.1. Análisis

En esta etapa ha sido necesaria una fase de investigación en relación al web scraping, y las diferentes herramientas disponibles para poder llevar a cabo estos fines.

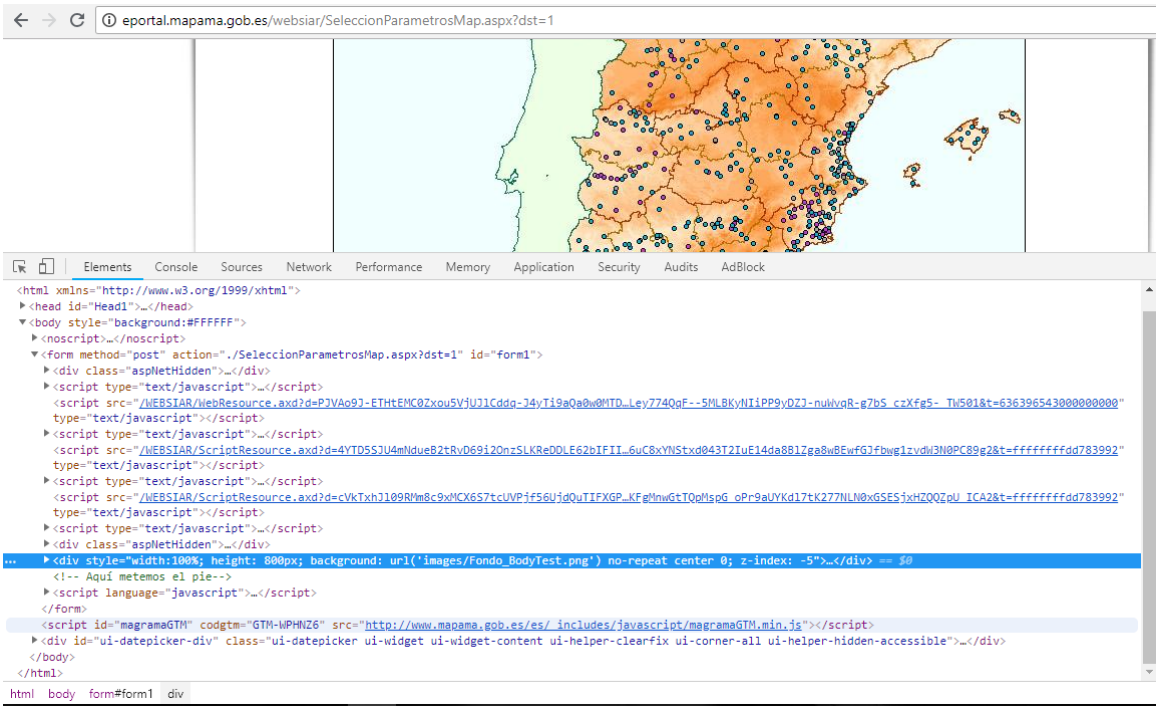


Ilustración 26 - Análisis de páginas web

Para realizar web scraping es necesario tener conocimientos de HTML, CSS, JavaScript y JQuery, para poder extraer los datos con exactitud.

Las herramientas gratuitas que mejor se pueden ajustar a nuestros intereses son:

- Jsoup
- HtmlUnit
- Selenium

6.2.2. Diseño

En esta segunda iteración no ha existido diseño como tal, ya que se continúa con los planteamientos de la iteración anterior, y todo lo que se va a desarrollar es en relación a crear el web scraper, el cual está directamente relacionado con implementación y pruebas, más que con diseño.

6.2.3. Implementación

Llegados a este punto, es el momento de implementar los primeros web scraper, que a modo de prueba, consistirán en obtener los títulos, y las entradas de diferentes páginas web.

Empiezo usando JSoup, un parser de Java con el que puedes obtener el HTML de una página web.

Dada una URL, JSoup devuelve un objeto de clase Document con el contenido HTML del sitio web, y dispone de los métodos necesarios para poder manipular el HTML y CSS.

Después de una serie de pruebas, compruebo que JSoup solo funciona en el caso de páginas web estáticas, y que no funciona correctamente cuando hay contenido JavaScript, por lo que toca pasar a otra herramienta.

El segundo intento es con HtmlUnit, un navegador web sin interfaz de usuario para Java, que permite cargar una página web y dispone de los métodos necesarios para manipular el contenido HTML, CSS, y esta vez sí, también el JavaScript.

Después de realizar varias pruebas, compruebo que HtmlUnit funciona bien en la mayoría de los casos, pero que en algunos escenarios, presenta un comportamiento que no es el esperado, por lo que es necesario pasar a otra herramienta.

6.2.4. Pruebas

Las pruebas en esta etapa han ido dirigidas a comprobar por un lado, las posibilidades del web scraping como solución en la obtención de los datos, y por otro, determinar la utilidad de las dos herramientas comentadas anteriormente.

El resultado final de las pruebas es descartar las dos primeras opciones, JSoup y HtmlUnit, por ser insuficientes para la consecución de los objetivos, y por lo tanto hay que pasar a probar la siguiente herramienta alternativa, Selenium.

El resultado final de esta iteración son las dos demos en JSoup y HtmlUnit para parsear información de diferentes páginas web. Ha servido de introducción al web scraping y a los conceptos relacionados.

6.3. Iteración 3

6.3.1. Análisis

En esta iteración el primer paso ha sido estudiar y analizar la estructura del código HTML de la página SiAR para conocer cómo es posible extraer la información.

Planteo el primer modelo de dominio de la aplicación:

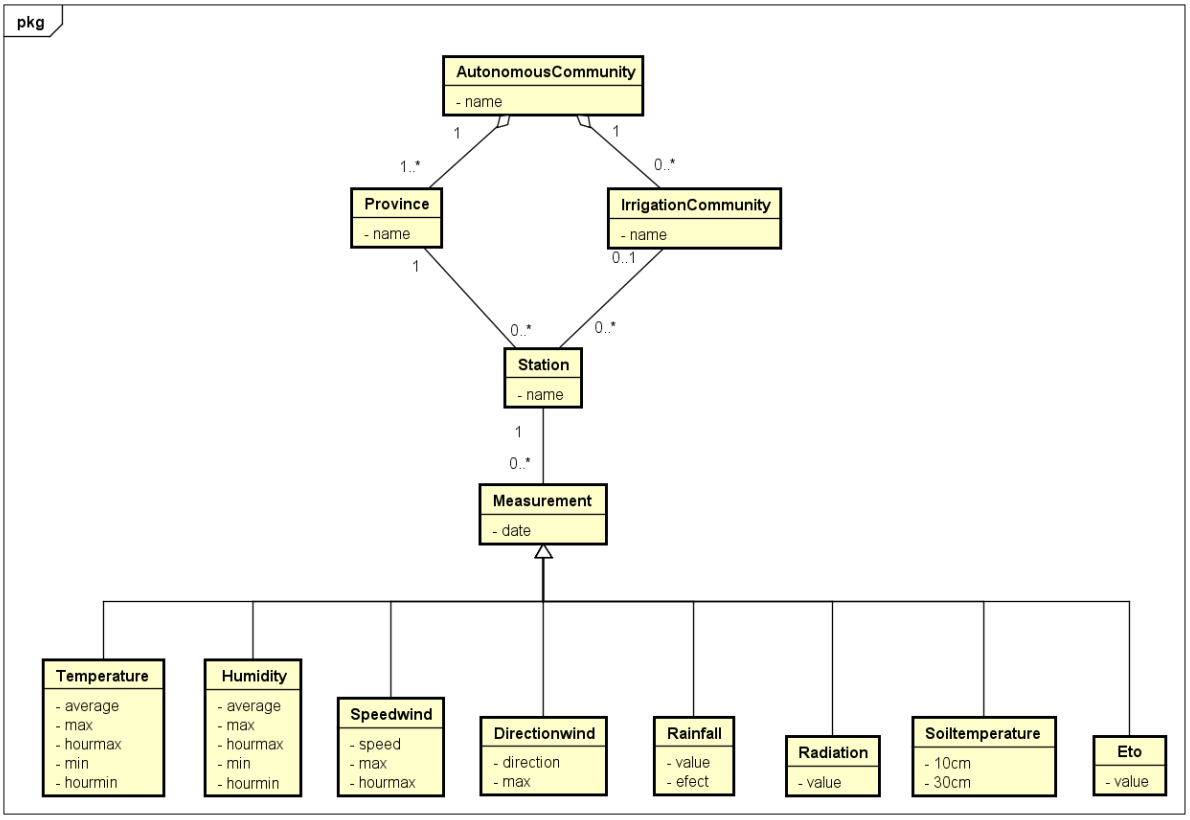


Ilustración 27 - Modelo de dominio iteración 3

6.3.2. Diseño

El diseño da como resultado el siguiente diagrama de clases:

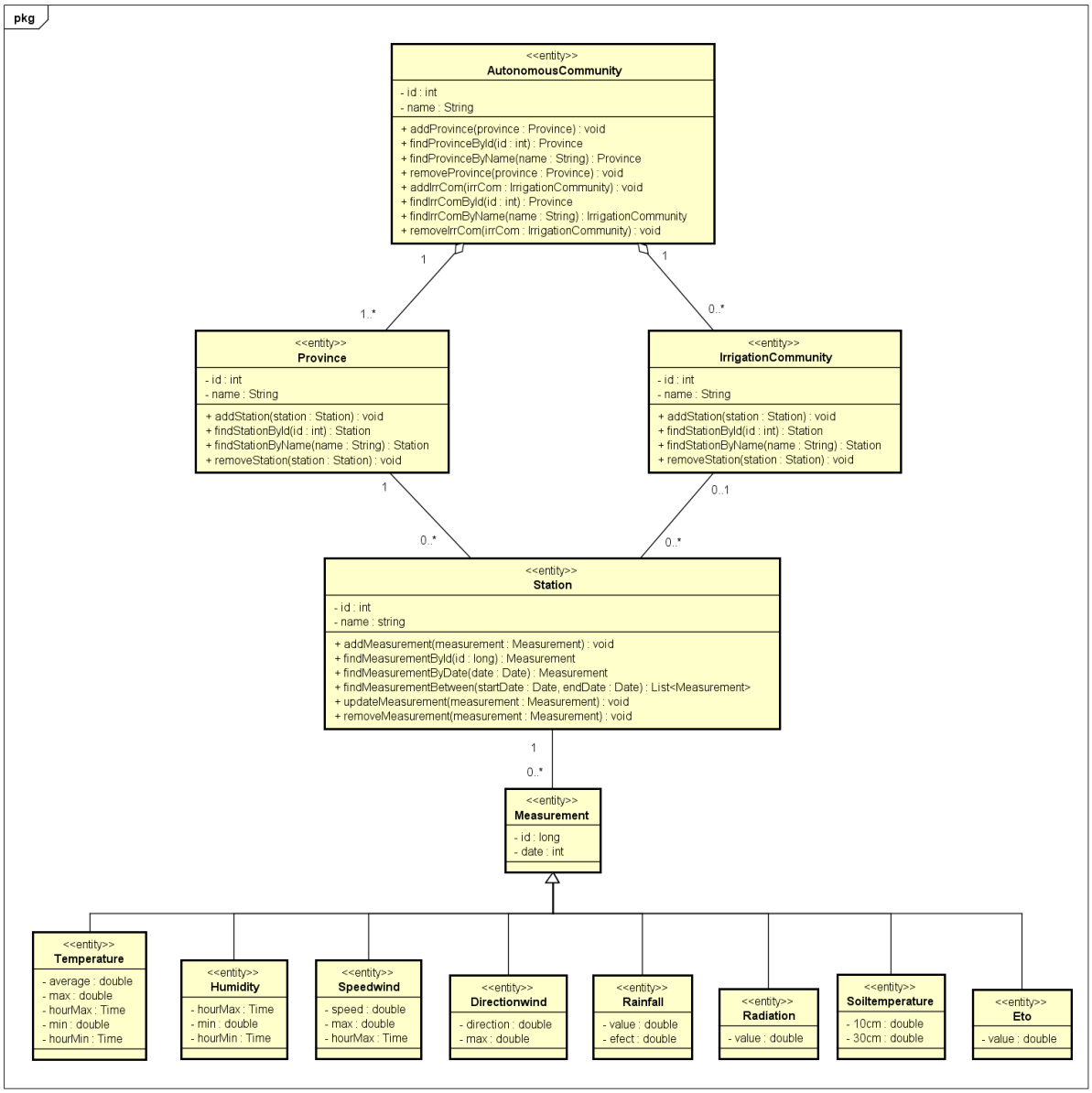


Ilustración 28 - Diagrama de clases iteración 3

6.3.3. Implementación

Es el turno de emplear Selenium, una herramienta originalmente diseñada para la automatización de pruebas, pero que se ajusta perfectamente a otros fines como el web scraping.

Por tanto, comienzo con Selenium WebDriver, que a su vez necesita el navegador Firefox para recorrer las páginas web y poder extraer la información.

Cuando se ve en funcionamiento Selenium, llama la atención la forma que tiene de controlar el navegador directamente como si de un usuario real se tratase.

Y aquí se presenta el primer problema con Selenium, que necesita desplegar el navegador, lo cual no termina de ser de lo más eficiente, y por ejemplo podría implicar la necesidad de enfocar la extracción de datos a procesos batch durante las noches, por razones de eficiencia, pero es un mal menor que se acepta, ya que por el momento es la mejor opción disponible.

En cualquier caso, buscare una solución más adelante, aunque como posible solución, una alternativa para resolverlo podría ser PhantomJS.

Para poder empezar a usar Selenium, es necesario añadir las dependencias al build.gradle.

```
compile "org.seleniumhq.selenium:selenium-api:${seleniumVersion}"
compile "org.seleniumhq.selenium:selenium-support:${seleniumVersion}"
compile "org.seleniumhq.selenium:selenium-remote-driver:${seleniumVersion}"
compile "org.seleniumhq.selenium:selenium-firefox-driver:${seleniumVersion}"
```

Antes de implementar los web scraper, es necesario primero crear las clases del modelo, sus controladores, sus servicios y los repositorios.

Para la creación de los id de las entidades del modelo me he valido de la anotación @GenericGenerator, que permite generar tu propia secuencia.

```
@GenericGenerator(
        name = "mySequenceGenerator",
        strategy = "org.hibernate.id.enhanced.SequenceStyleGenerator",
        parameters = {
                @Parameter(name = "sequence_name", value = "sequence_aacc"),
                @Parameter(name = "initial_value", value = "1"),
                @Parameter(name = "increment_size", value = "1")
        }
)
@Id
@GeneratedValue(generator = "mySequenceGenerator")
private Long id_aacc;
```

Ilustración 29 - @GenericGenerator/@Id/@GeneratedValue

Para poder realizar web scraping es necesario conectarse a la página objetivo, mediante el WebDriver de Selenium:

```
String url = "http://eportal.mapama.gob.es/websiar/Inicio.aspx";

WebDriver driver = new FirefoxDriver();

driver.get(url);
```

A partir de ahí, Selenium dispone de una serie de clases y métodos para poder manipular la página:

```
Select dropdownCA = new Select(driver.findElement(By.id("ContentPlaceHolder1_cboCCAA")));

List<WebElement> comunidades = dropdownCA.getOptions();

selectedCA= selectOption(comunidades, inputCA);

dropdownCA.selectByVisibleText(selectedCA);
```

Implemento los primeros scraper web, y al final del sprint la arquitectura lógica del proyecto es la siguiente:

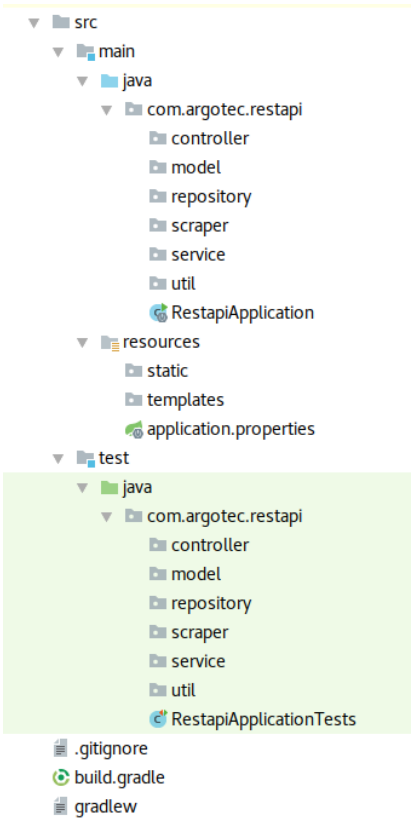


Ilustración 30 - Arquitectura lógica I3

Se mantiene la estructura de paquetes planteada en la iteración anterior, añadiendo dos nuevos paquetes, el paquete scraper, y el paquete útil:

- controller: contiene las clases encargadas de mapear los recursos de la API
- model: contiene las entidades de la aplicación
- repository: contiene las clases de acceso a datos
- scraper: contiene las clases de scraping web
- service: contiene las clases de la lógica de negocio
- util: contiene las clases de soporte con funciones generales

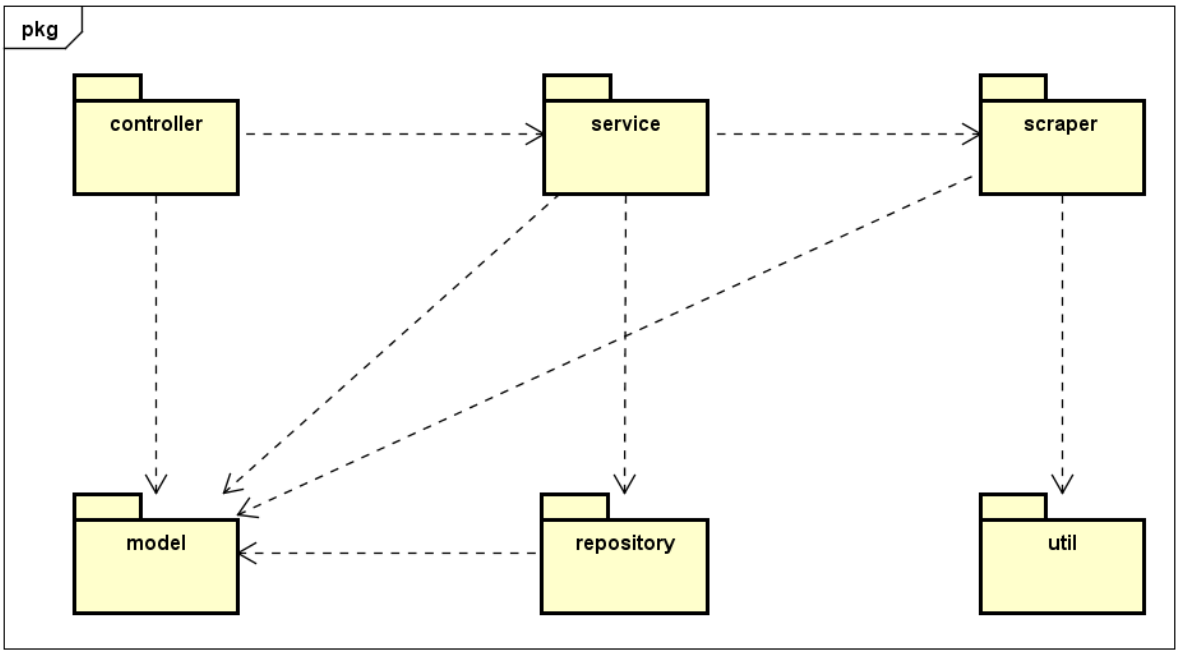


Ilustración 31 - Diagrama de paquetes I3

Ahora tenemos un componente más dentro de nuestra arquitectura:

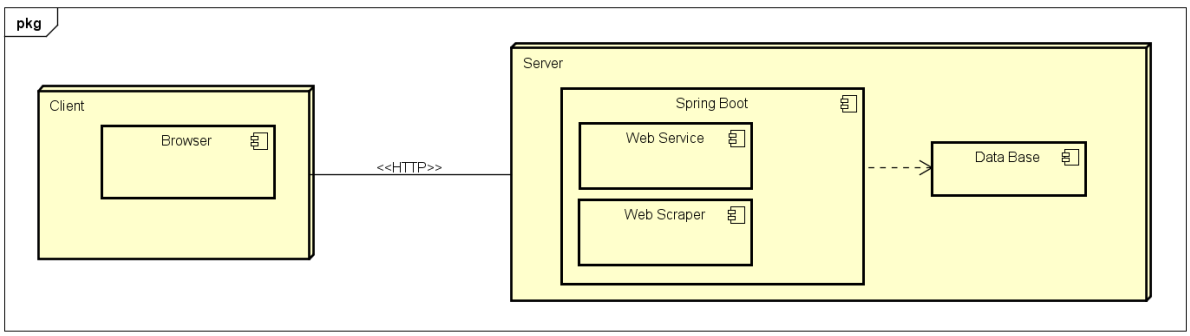


Ilustración 32 - Diagrama de despliegue I3

6.3.4. Pruebas

Recordar que en TDD, primero se define el comportamiento esperado, y como aún no está implementado, hace que la primera vez la prueba falle, y después, se hace lo necesario para pasar la prueba.

A modo de ejemplo, mi primer test de la aplicación no podía ser otro que el clásico “Hola Mundo” adaptado a mi modelo.

Creo el test unitario:

```
@Test
public void itShouldSayHolaMundo(){
    assertThat( actual: "Hola mundo!",is(autonomousCommunityService.helloWorld()));

}
```

Ilustración 33 - Test Unitario Hola Mundo

La primera vez que ejecuto la prueba, falla:

Ilustración 34 - Test falla (Rojo)

Implemento lo necesario para qué pase la prueba, y la prueba se realiza con éxito:

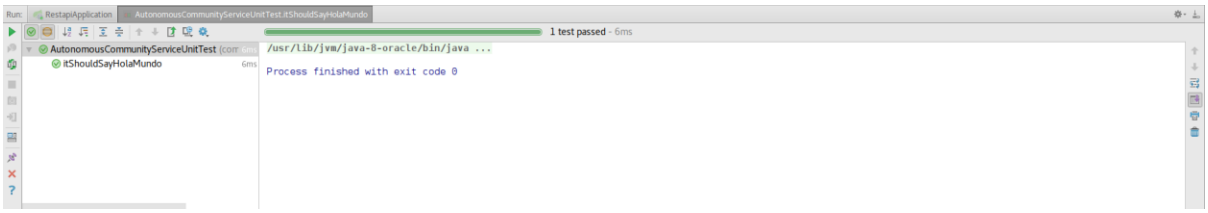


Ilustración 35 - Test pasa (Verde)

Durante esta etapa se han ido realizando diferentes test unitarios y de integración, y a modo de ejemplo, el siguiente test de integración:

```
@RunWith(SpringRunner.class)
@SpringBootTest(
        webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT,
        classes = RestapiApplication.class)
```

Ilustración 36 - @SpringBootTest

Las anotaciones se corresponde con:

- @RunWith (SpringRunner.class): dice a JUnit que se ejecute usando el soporte de pruebas de Spring. SpringRunner es el nuevo nombre de SpringJUnit4ClassRunner.

- @SpringBootTest indica a Spring que primero debe usar @Configuracion, cargando la propiedades definidas adicionalmente en application.properties, y en caso de no existir, buscar la clase primaria @SpringBootApplication.

- Con el atributo webEnviroment se puede configurar los entornos específicos de prueba, ya sea mediante un Mock o mediante un servidor HTTP real ejecutándose en un puerto aleatorio (RANDOM_PORT) o en uno concreto (DEFINED_PORT)

```
    @Test
    public void getAllCommunitiesShouldReturnFoundCommunities() throws Exception{

        AutonomousCommunity firstCommunity = new AutonomousCommunity( name: "AACC 1");
        firstCommunity.setId(1L);
        repository.save(firstCommunity);

        AutonomousCommunity secondCommunity = new AutonomousCommunity( name: "AACC 2");
        secondCommunity.setId(2L);
        repository.save(secondCommunity);

        AutonomousCommunity thirdCommunity = new AutonomousCommunity( name: "AACC 3");
        thirdCommunity.setId(3L);
        repository.save(thirdCommunity);

        mockMvc.perform(get( urlTemplate: "/communities"))
                .andExpect(status().isOk())
                .andExpect(content().contentType(TestUtil.APPLICATION_JSON_UTF8))
                .andExpect(jsonPath( expression: "$", hasSize(3)))
                .andExpect(jsonPath( expression: "$[0].id", Matchers.is( value: 1)))
                .andExpect(jsonPath( expression: "$[0].name", Matchers.is( value: "AACC 1")))
                .andExpect(jsonPath( expression: "$[1].id", Matchers.is( value: 2)))
                .andExpect(jsonPath( expression: "$[1].name", Matchers.is( value: "AACC 2")))
                .andExpect(jsonPath( expression: "$[2].id", Matchers.is( value: 3)))
                .andExpect(jsonPath( expression: "$[2].name", Matchers.is( value: "AACC 3")));
    }
}
```

Ilustración 37 - Test Integración Comunidades

Donde se pone a prueba tanto los controladores, como los servicios, como el repositorio.

El resultado de esta iteración es la implementación de las primeras funcionalidades de la aplicación, siendo capaz de extraer parte de los datos.

6.4. Iteración 4

El principal objetivo de la iteración 4 es continuar trabajando con la obtención de los parámetros meteorológicos, por lo que es una continuación directa de la iteración anterior.

6.4.1. Análisis

Se continúa con la visión planteada en la iteración anterior, no siendo necesario replantear los modelos, ya que no se han presentado cambios significativos.

6.4.2. Diseño

Sucede lo mismo que con la fase de análisis, continuo trabajando en base al diseño planteado en la iteración pasada.

A continuación muestro el diagrama de secuencia para la obtención de datos meteorológicos:

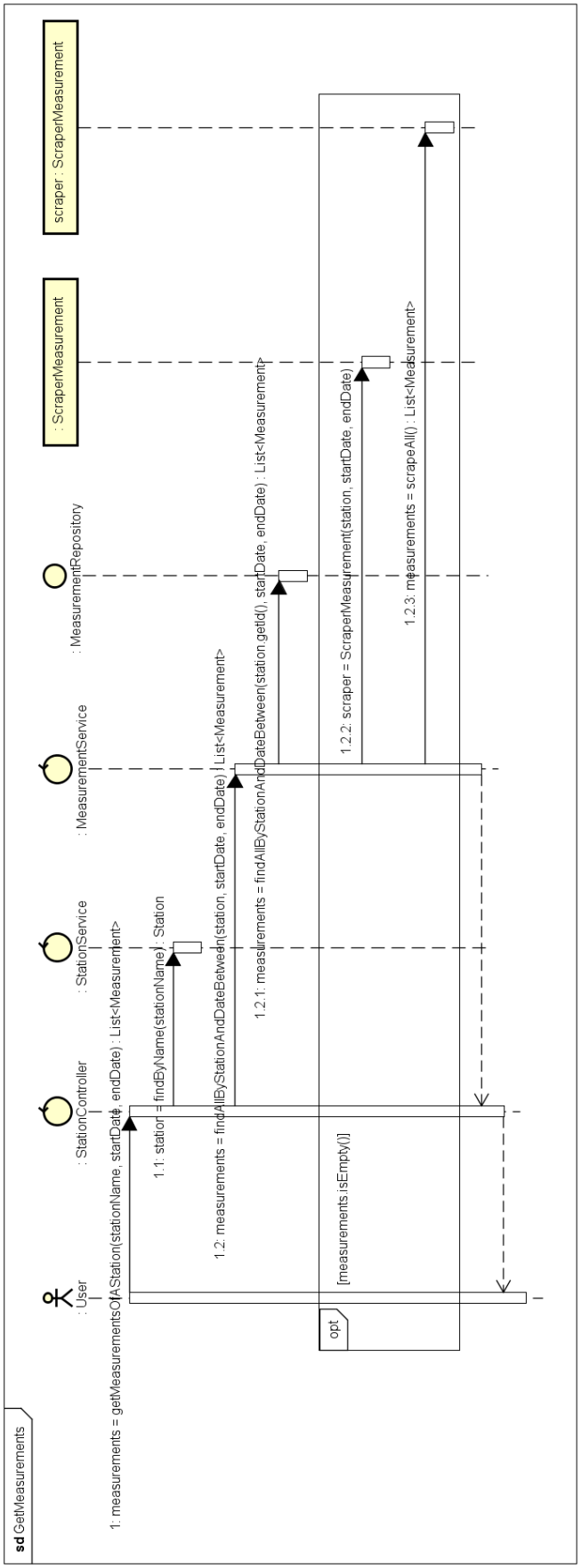


Ilustración 38 - Diagrama de secuencia Measurements

6.4.3. Implementación

En esta fase he continuado con la implementación de las funcionalidades relacionadas con la obtención de las medidas meteorológicas, y durante el proceso dos puntos a destacar son:

- Aunque se ha implementado la opción de seleccionar todas las estaciones a nivel de scraper, se ha decidido no incluir esa opción a través de la API, debido a que obtener la información de varias estaciones al mismo tiempo, sin saber exactamente cuáles van a ser esas estaciones, no tiene tanto sentido, al menos en esta primera fase de implementación de la aplicación, y se marca la historia de usuario US-010 con una prioridad W, pasando a ser en todo caso, una funcionalidad a implementar completamente a futuro.

- Por otro lado, teniendo implementado a nivel de scraper, la opción de seleccionar solo algunos parámetros meteorológicos, o deseleccionar todos los parámetros de consulta, se he decidido que finalmente se realizara las peticiones a través de la API teniendo en cuenta todos los parámetros de consulta, debido a que la finalidad es no solo obtener ciertos datos meteorológicos, sino a partir de ellos poder generar recomendaciones, y disponer de medidas aisladas no tiene realmente sentido, ya que para poder calcular variables agroclimáticas es necesario tener en cuenta todas las medidas meteorológicas en su conjunto, por lo que las historias de usuario US-014 y US-016 se descartan.

Para el seguimiento de logs, he usado la librería de Java, SLFJ4, que implementa el patrón fachada, y es muy adecuado para las tareas de seguimiento del flujo en el código.

Pasa su uso basta con importar las librerías:

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
```

Y definir:

```
private static final Logger log = LoggerFactory.getLogger(AutonomousCommunityController.class);
```

Desde ese momento es posible hacer uso de logs, con diferentes niveles de prioridad, como:

- Debug: para mensajes de depuración
- Info: para mensajes de información del programa
- Warn: para mensajes de alerta

- Error: para mensajes de error
- Fatal: para mensajes críticos

Y para poder usarlos es tan simple como:

```
log.info("Community added in the database");
```

6.4.4. Pruebas

A la hora de crear test, hay cierta libertad para decidir cuál es el mejor enfoque, y poder aprovechar las diferentes anotaciones.

Cuando es necesario simular el comportamiento de un objeto, se emplean los mocks. En relación a los mocks hay varias anotaciones como:

- @Mock: crea un mock que simula el comportamiento esperado de un componente

```
@Mock
private AutonomousCommunityRepository repository;
```

- @MockBean es similar a @Mock pero con el soporte de Spring

```
@MockBean
private AutonomousCommunityService service;
```

- @InjectMocks inyecta los mocks como dependencias dentro del componente

```
@InjectMocks
private AutonomousCommunityService service;
```

También se dispone de la anotación @Before, la cual indica que es un método que se tiene que ejecutar una vez antes de que se ejecute el test de prueba.

```
@Before
public void setUp() {

    firstCommunity = new AutonomousCommunity( name: "AACC TS1");
    AutonomousCommunity secondCommunity = new AutonomousCommunity( name: "AACC TS2");
    AutonomousCommunity thirdCommunity = new AutonomousCommunity( name: "AACC TS3");
    firstCommunity.setId(1L);
    secondCommunity.setId(2L);
    thirdCommunity.setId(3L);
    allCommunities.add(firstCommunity);
    allCommunities.add(secondCommunity);
    allCommunities.add(thirdCommunity);
}
```

Ilustración 39 - @Before

Ha continuación un test de ejemplo con Mockito, donde compruebo que al buscar una comunidad por su nombre, se devuelve la comunidad especificada:

```
@Test
public void findByNameShouldReturnThisCommunityWithMockito() throws Exception {
    AutonomousCommunityRepository repository = mock(AutonomousCommunityRepository.class);
    when(repository.findByName("AACC TSM1")).thenReturn(fCommunity);
    AutonomousCommunityServiceImpl service = new AutonomousCommunityServiceImpl(repository);
    AutonomousCommunity communityFound = service.findByName("AACC TSM1");
    assertThat(fCommunity, is(communityFound));
}
```

Ilustración 40 - Test Mockito

La ventaja de usar mocks, es por ejemplo, que al simular el comportamiento del repositorio, encargado del acceso a los datos, consigues no tener una dependencia real con la base de datos, permitiendo realizar las pruebas de manera aislada sin la necesidad de arrancar toda la aplicación.

Desde el punto de vista de la API, si se realiza una consulta de las medidas meteorológicas, recibiremos un JSON con los distintos parámetros meteorológicos

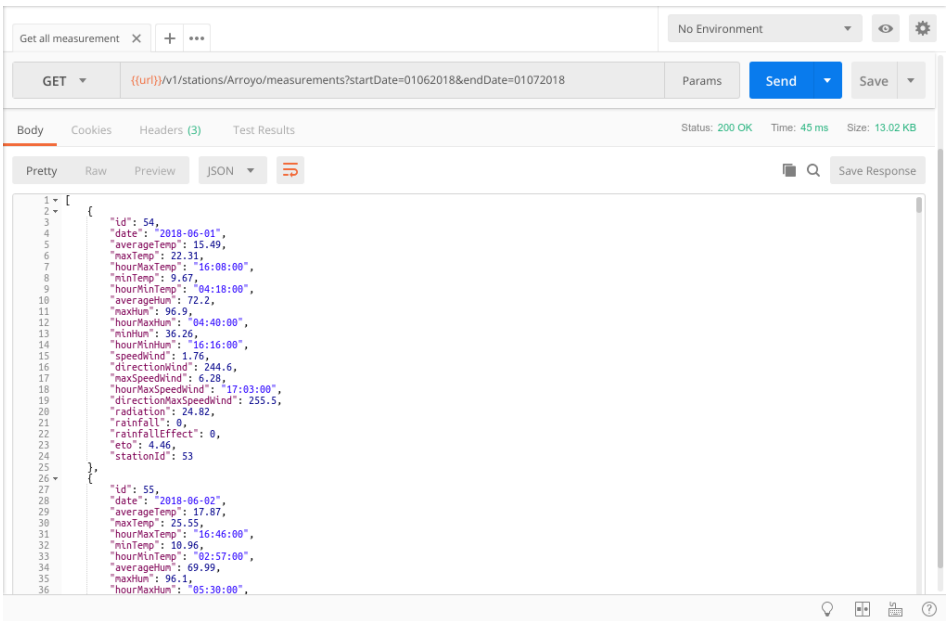
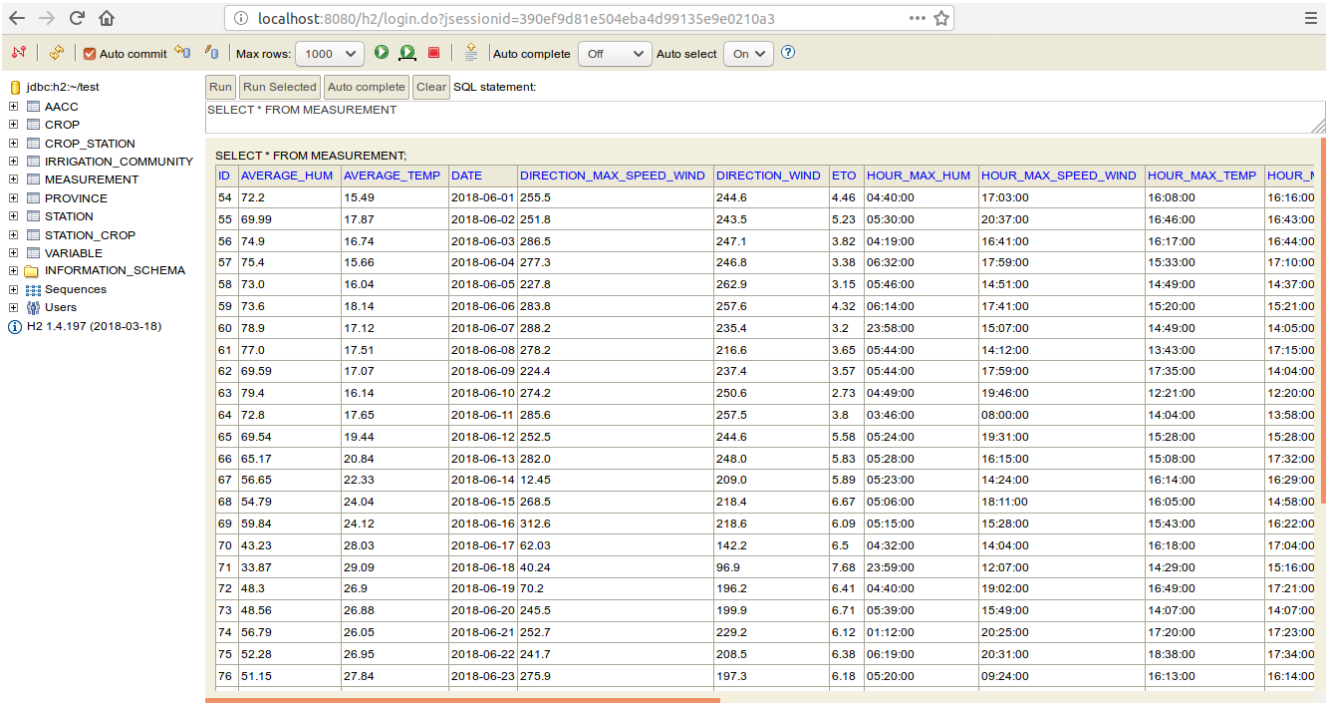


Ilustración 41 - JSON medida meteorológicas

Como podemos comprobar, la información de una consulta se almacena correctamente en la base de datos



ID	AVERAGE_HUM	AVERAGE_TEMP	DATE	DIRECTION_MAX_SPEED_WIND	DIRECTION_WIND	ETO	HOUR_MAX_HUM	HOUR_MAX_SPEED_WIND	HOUR_MAX_TEMP	HOUR_M
54	72.2	15.49	2018-06-01	255.5	244.6	4.46	04:40:00	17:03:00	16:08:00	16:16:00
55	69.99	17.87	2018-06-02	251.8	243.5	5.23	05:30:00	20:37:00	16:46:00	16:43:00
56	74.9	16.74	2018-06-03	286.5	247.1	3.82	04:19:00	16:41:00	16:17:00	16:44:00
57	75.4	15.66	2018-06-04	277.3	246.8	3.38	06:32:00	17:59:00	15:33:00	17:10:00
58	73.0	16.04	2018-06-05	227.8	262.9	3.15	05:46:00	14:51:00	14:49:00	14:37:00
59	73.6	18.14	2018-06-06	283.8	257.6	4.32	06:14:00	17:41:00	15:20:00	15:21:00
60	78.9	17.12	2018-06-07	288.2	235.4	3.2	23:58:00	15:07:00	14:49:00	14:05:00
61	77.0	17.51	2018-06-08	278.2	216.6	3.65	05:44:00	14:12:00	13:43:00	17:15:00
62	69.59	17.07	2018-06-09	224.4	237.4	3.57	05:44:00	17:59:00	17:35:00	14:04:00
63	79.4	16.14	2018-06-10	274.2	250.6	2.73	04:49:00	19:46:00	12:21:00	12:20:00
64	72.8	17.65	2018-06-11	285.6	257.5	3.8	03:46:00	08:00:00	14:04:00	13:58:00
65	69.54	19.44	2018-06-12	252.5	244.6	5.58	05:24:00	19:31:00	15:28:00	15:28:00
66	65.17	20.84	2018-06-13	282.0	248.0	5.83	05:28:00	16:15:00	15:08:00	17:32:00
67	56.65	22.33	2018-06-14	12.45	209.0	5.89	05:23:00	14:24:00	16:14:00	16:29:00
68	54.79	24.04	2018-06-15	268.5	218.4	6.67	05:06:00	18:11:00	16:05:00	14:58:00
69	59.84	24.12	2018-06-16	312.6	218.6	6.09	05:15:00	15:28:00	15:43:00	16:22:00
70	43.23	28.03	2018-06-17	62.03	142.2	6.5	04:32:00	14:04:00	16:18:00	17:04:00
71	33.87	29.09	2018-06-18	40.24	96.9	7.68	23:59:00	12:07:00	14:29:00	15:16:00
72	48.3	26.9	2018-06-19	70.2	196.2	6.41	04:40:00	19:02:00	16:49:00	17:21:00
73	48.56	26.88	2018-06-20	245.5	199.9	6.71	05:39:00	15:49:00	14:07:00	14:07:00
74	56.79	26.05	2018-06-21	252.7	229.2	6.12	01:12:00	20:25:00	17:20:00	17:23:00
75	52.28	26.95	2018-06-22	241.7	208.5	6.38	06:19:00	20:31:00	18:38:00	17:34:00
76	51.15	27.84	2018-06-23	275.9	197.3	6.18	05:20:00	09:24:00	16:13:00	16:14:00

Ilustración 42 - Persistencia medidas H2

El resultado final de esta iteración es, una aplicación potencialmente entregable, que permite obtener y almacenar la información relacionada con medidas meteorológicas procedentes de SiAR.

6.5. Iteración 5

El principal objetivo de la iteración 5 es ampliar las funcionalidades de la aplicación, permitiendo además obtener diferentes variables agroclimáticas.

6.5.1. Análisis

Al tratarse ahora de una página totalmente distinta, en esta fase es necesario realizar de nuevo el estudio y análisis de la estructura HTML, CSS y JavaScript de la página para poder extraer la información de manera precisa.

En esta iteración se plantea un nuevo modelo en el que ahora entran en juego los cultivos y las variables agroclimáticas:

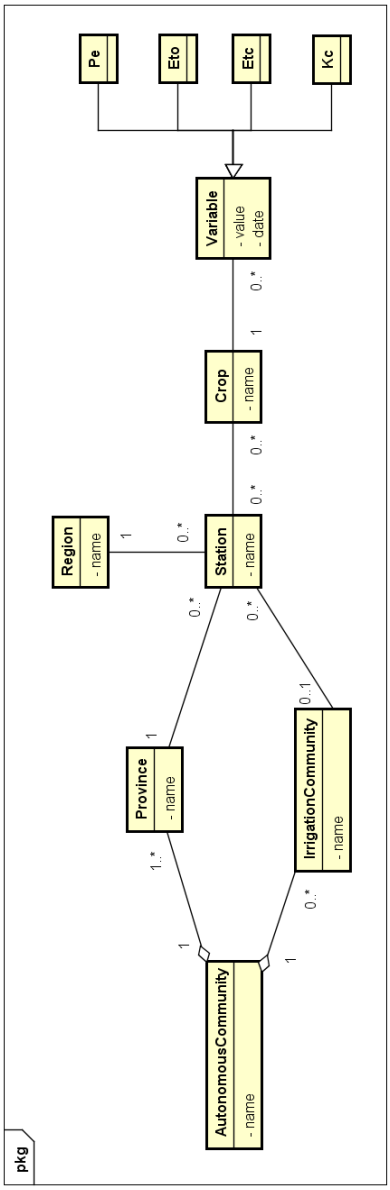


Ilustración 43- Modelo de dominio iteración 5

6.5.2. Diseño

Tenemos el siguiente diagrama de clases:

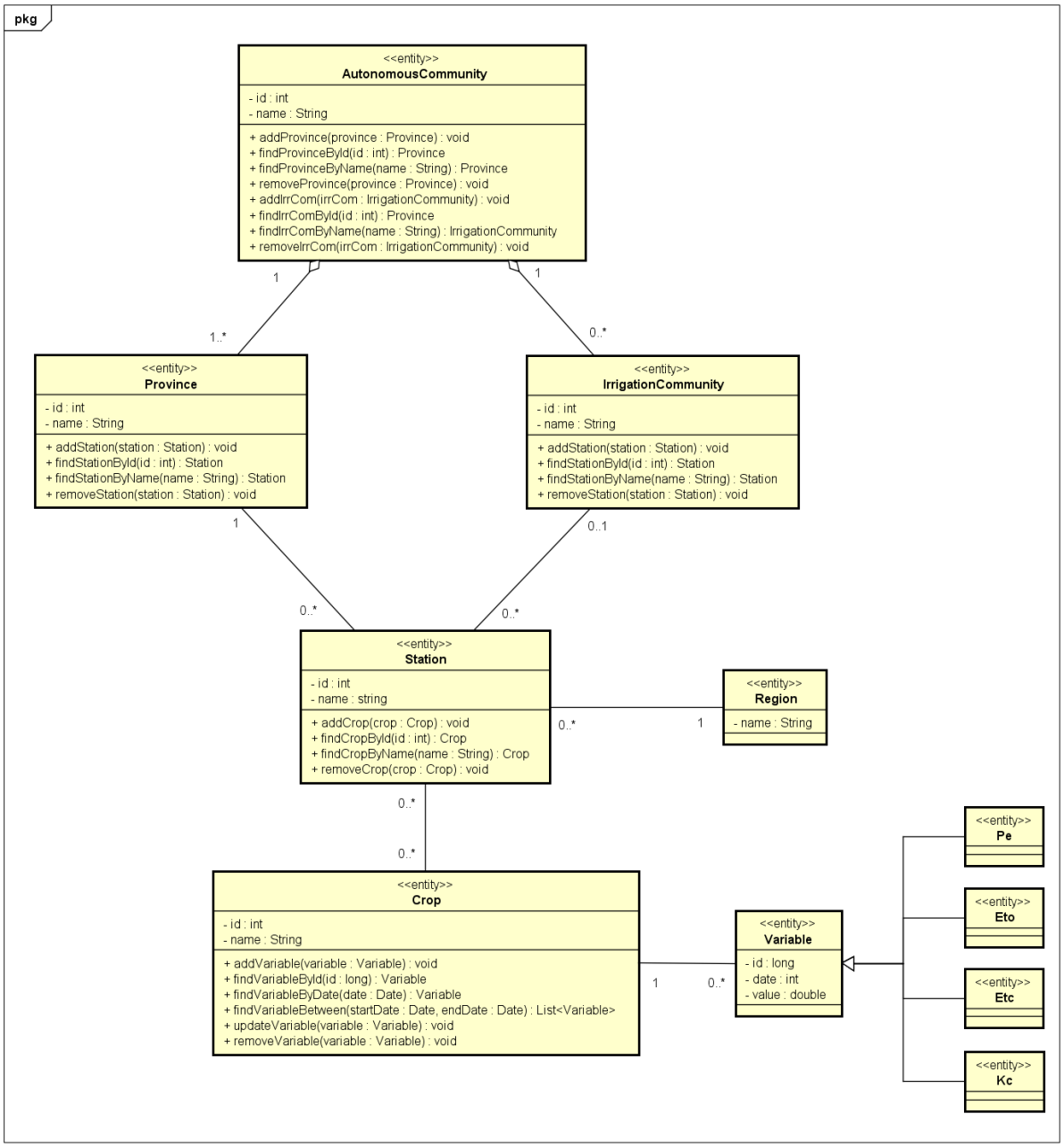


Ilustración 44 - Diagrama de clases iteración 5

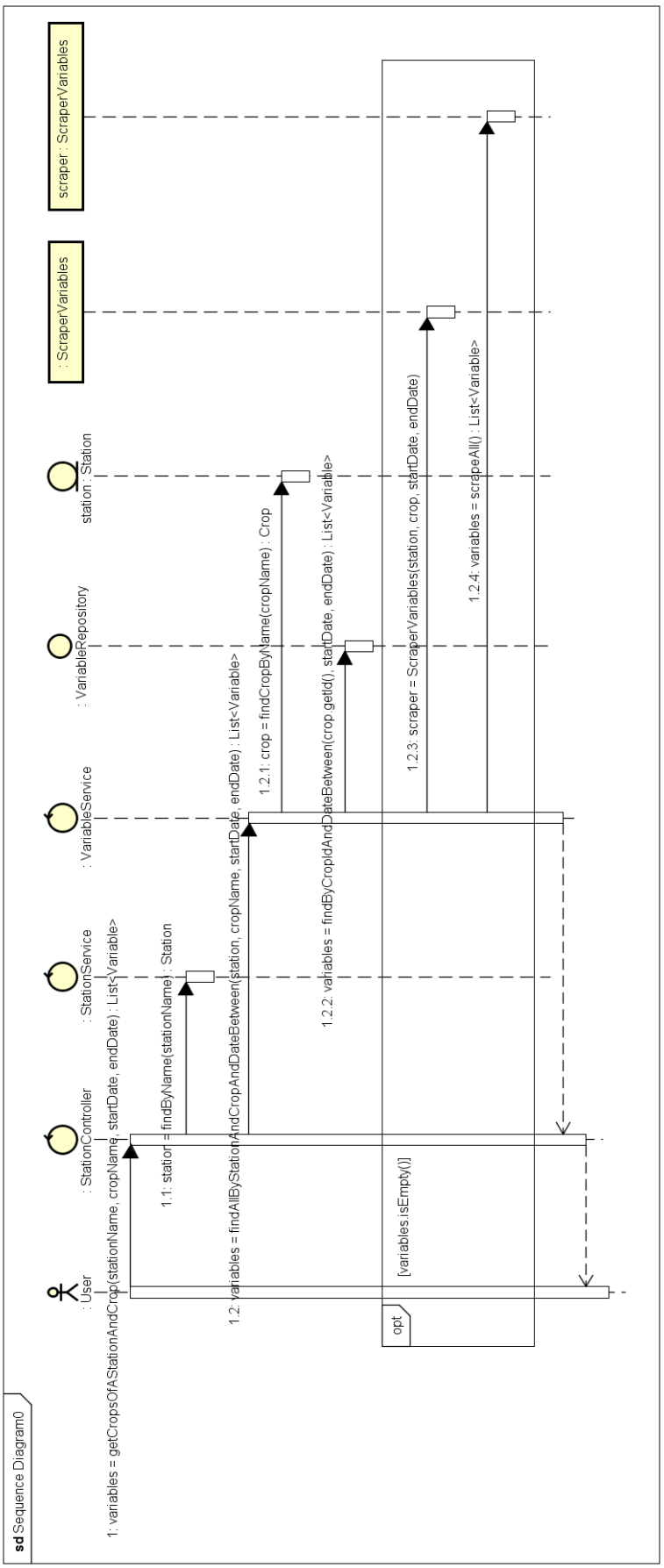


Ilustración 45 - Diagrama de secuencia Variables

6.5.3. Implementación

En esta fase comienzo con la implementación de las nuevas funcionalidades, siguiendo los planteamientos vistos anteriormente.

A parte de trabajar en la implementación de los cultivos y la obtención de las variables agroclimáticas, dedico parte del esfuerzo a realizar una refactorización general del código, con el fin de una mejor organización, para aumentar el grado de encapsulamiento y reducir el acoplamiento.

6.5.4. Pruebas

Realizo las pruebas unitarias y de integración necesarias.

Como resultado de la iteración, una aplicación potencialmente entregable, con un incremento en sus funcionalidades, y una mejora significativa en la organización y calidad del código.

6.6. Iteración 6

El objetivo principal de esta iteración es continuar con el desarrollo en relación con las variables agroclimáticas, teniendo como resultado la suma del trabajo de todas las iteraciones anteriores, e integrar PostgreSQL en la aplicación.

6.6.1. Análisis

Ahora se tiene en cuenta tanto las medidas meteorológicas como las variables agrometeorológicas, dando como resultado el siguiente modelo de dominio:

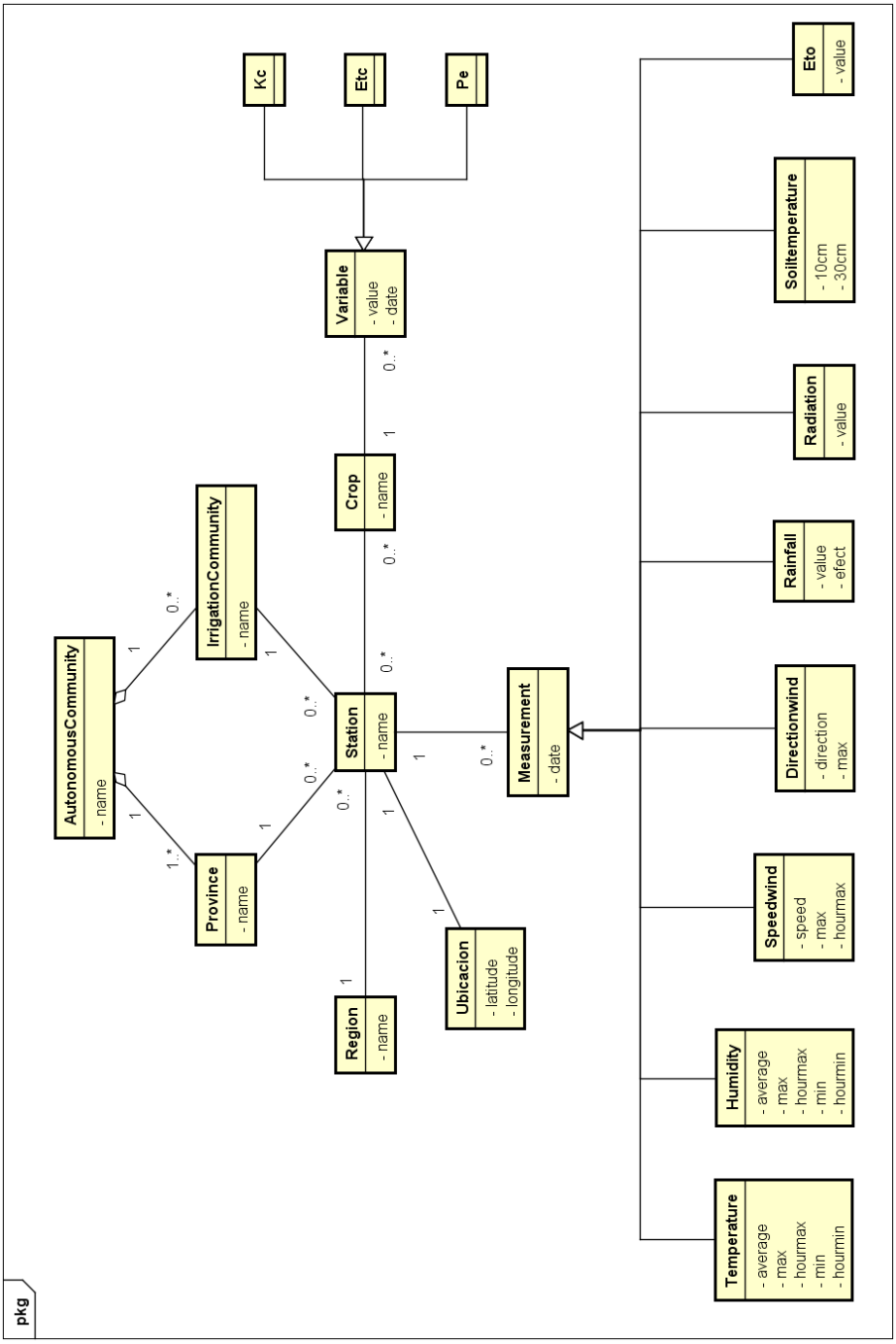


Ilustración 46 - Modelo de dominio iteración 6

6.6.2. Diseño

En consecuencia planteo el siguiente diagrama de clases:

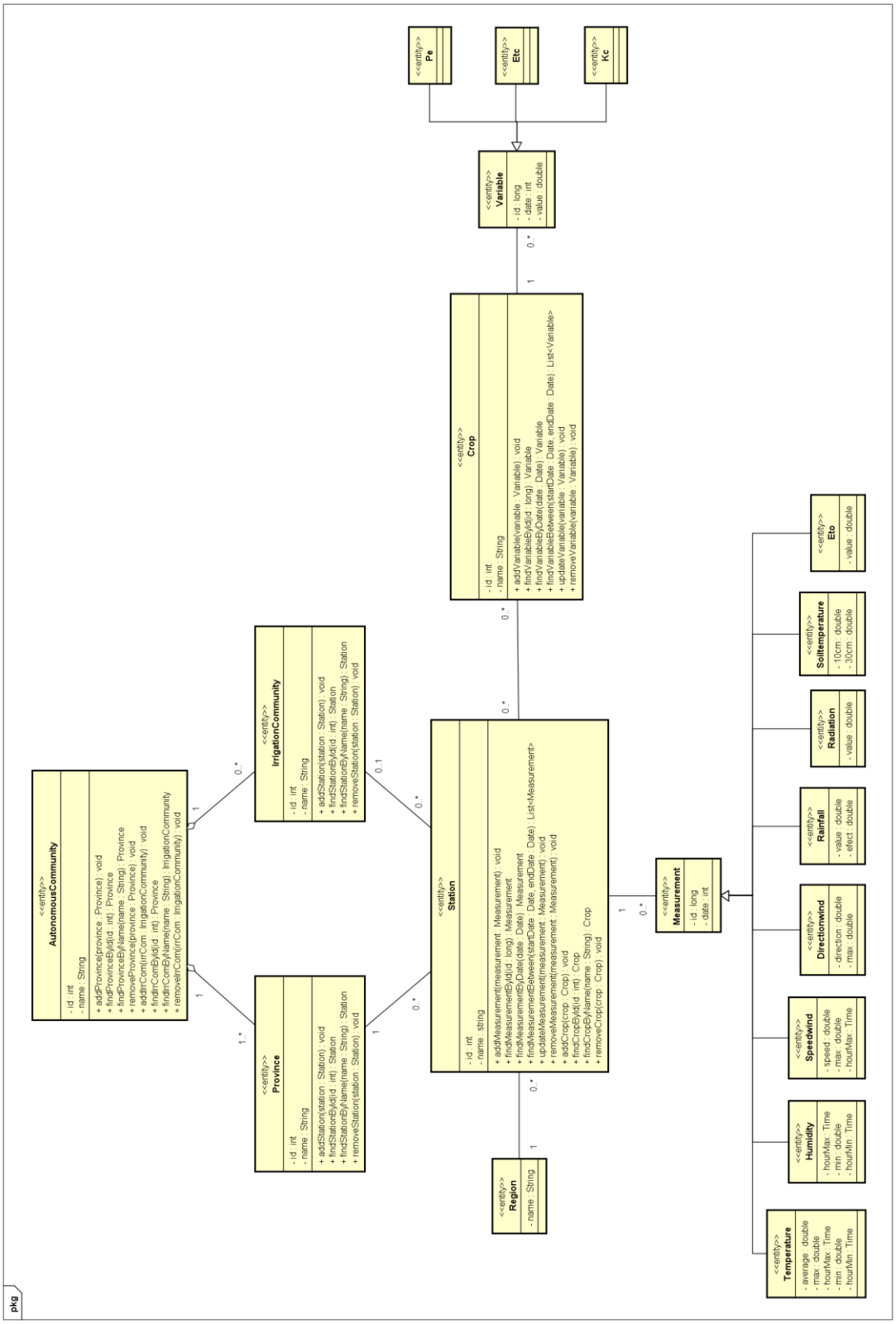


Ilustración 47 - Diagrama de clases iteración 6

6.6.3. Implementación

Durante el proceso de desarrollo han existido dos vías de consulta para los endpoint, una en que la consulta se realiza directamente a la base de datos, mediante un endpoint normal, y otra, en la que se fuerza a la aplicación a realizar web scraping para obtener los datos.

En el primer caso, la consulta es bastante rápida, solo es necesario acceder a la base de datos y devolver la información, pero si se trata de una nueva consulta, y no se dispone de los datos en la base de datos, es necesario realizar web scraping para obtener los datos directamente de la página web, un proceso que requiere de más tiempo.

La forma de especificar a la aplicación de forma explícita que realice web scraping en una consulta, es mediante un parámetro que se añade en el endpoint:

```
ws=true
```

En busca de seguir la filosofía de mantenlo simple, el siguiente paso es hacer ese proceso transparente al usuario de la aplicación, de forma que al realizar una consulta, no sea necesario especificar nada, y directamente cuando se realice una petición, se compruebe primero si la información se encuentra disponible en la base de datos, y de no ser así, se proceda a extraer la información directamente de la página web mediante web scraping.

En esta iteración realizo la integración de la base de datos postgreSQL, primero hago la instalación en mi máquina, y después añado las dependencias al build.gradle:

```
runtime('org.postgresql:postgresql:42.1.1')
```

Y añado la configuración de la base de datos al archivo de propiedades:

```
spring.datasource.url=jdbc:postgresql://localhost:5432/restdb

spring.datasource.username=admin

spring.datasource.password=****
```

Una de las ventajas de usar JPA con Hibernate, el cual implementa la especificación JPA, además de otras muchas funcionalidades, es que hace transparente la definición de las tablas, y el migrar de una base de datos a otra no supone un gran problema.

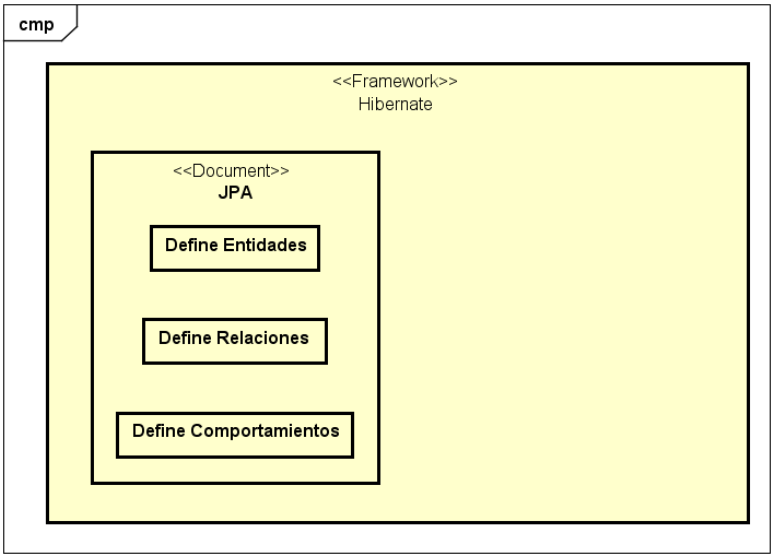


Ilustración 48 - Hibernate/JPA

A continuación muestro el diagrama Entidad-Relación de la aplicación:

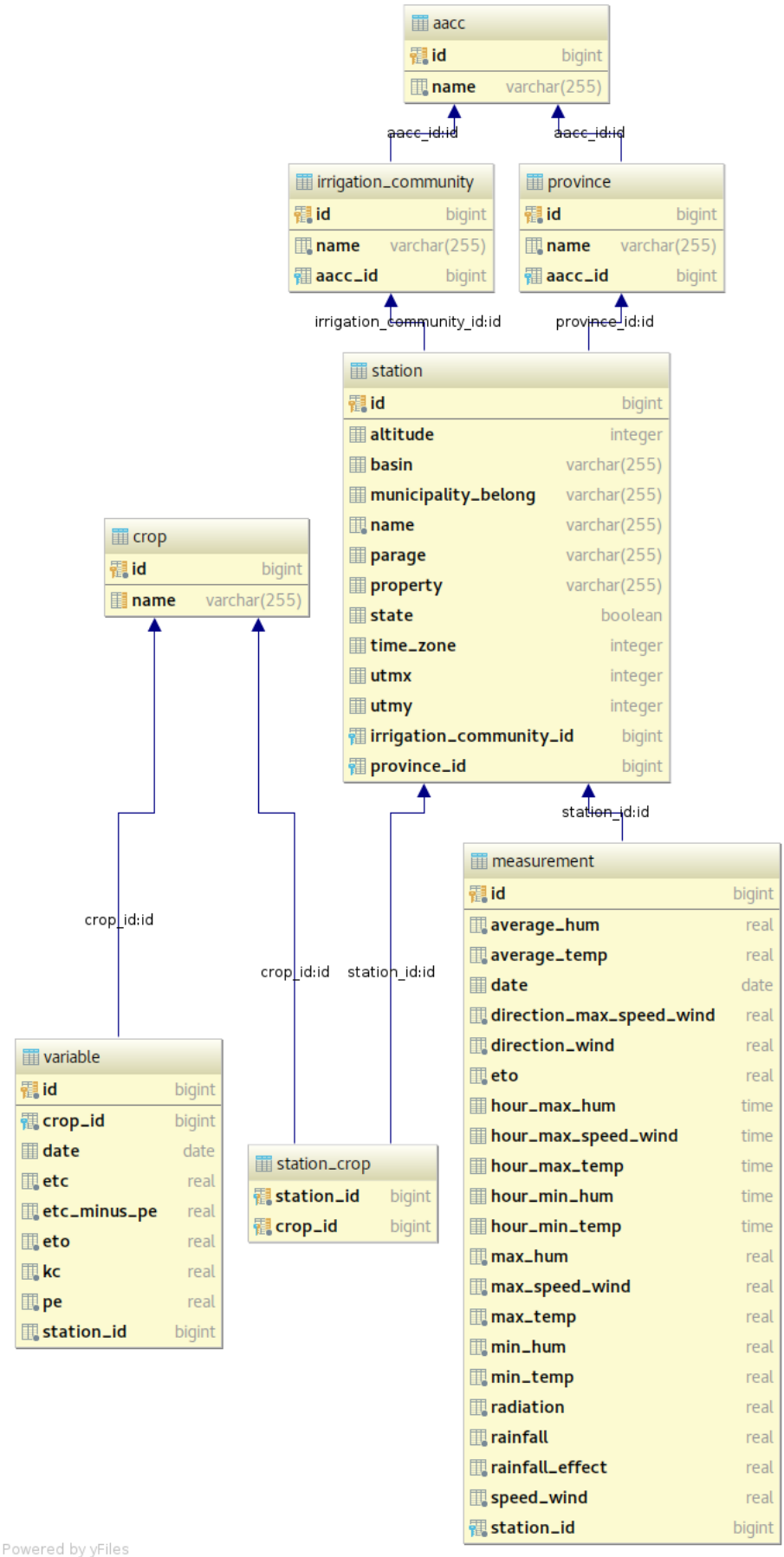


Ilustración 49 - Diagrama Entidad-Relación

6.6.4. Pruebas

Compruebo que la obtención de variables agroclimáticas funciona como se espera, y que la información se guarda correctamente en la base de datos.

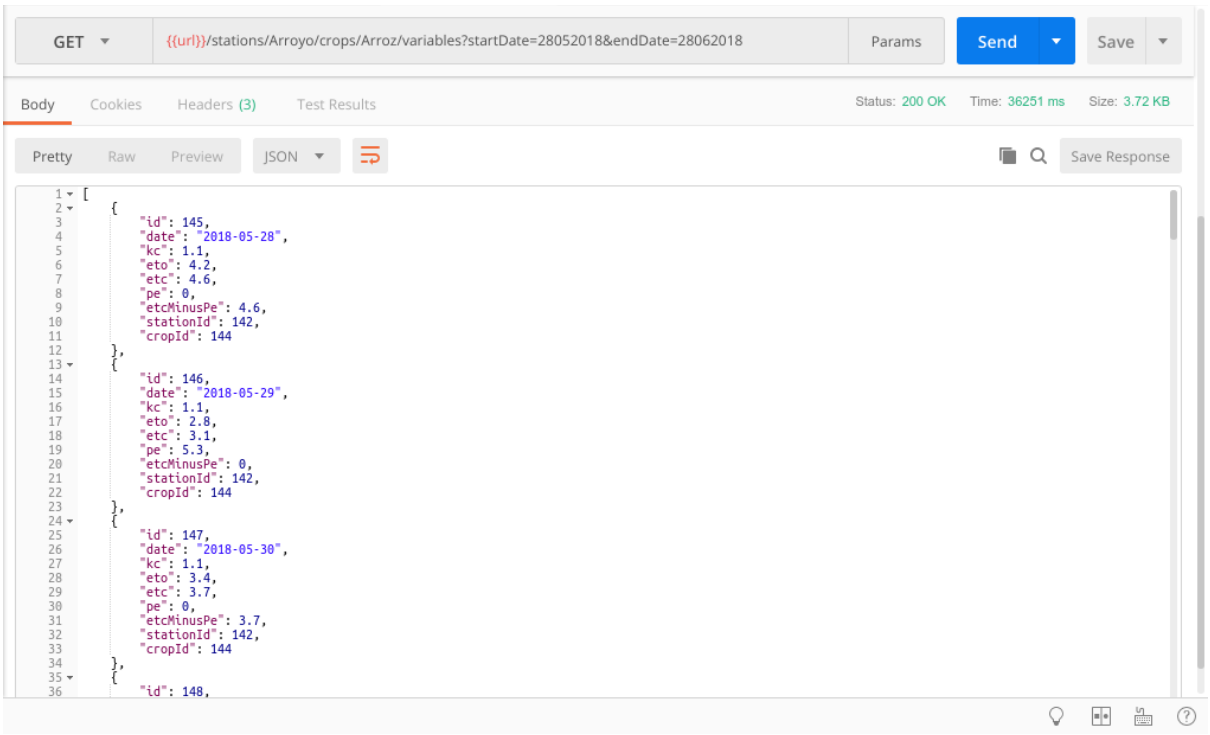


Ilustración 50 - Obtención de variables agroclimáticas

El resultado final de la iteración es una aplicación potencialmente entregable, con la que se puede obtener tanto medidas meteorológicas como variables agroclimáticas de la fuente SiAR.

6.7. Iteración 7

Esta iteración se corresponde con el último Sprint dentro de la planificación inicial del proyecto, donde los principales objetivos son ampliar algunas funcionalidades relacionadas con las estaciones, y terminar la aplicación para su entrega final.

6.7.1. Análisis

El modelo de dominio final de la aplicación es el siguiente:

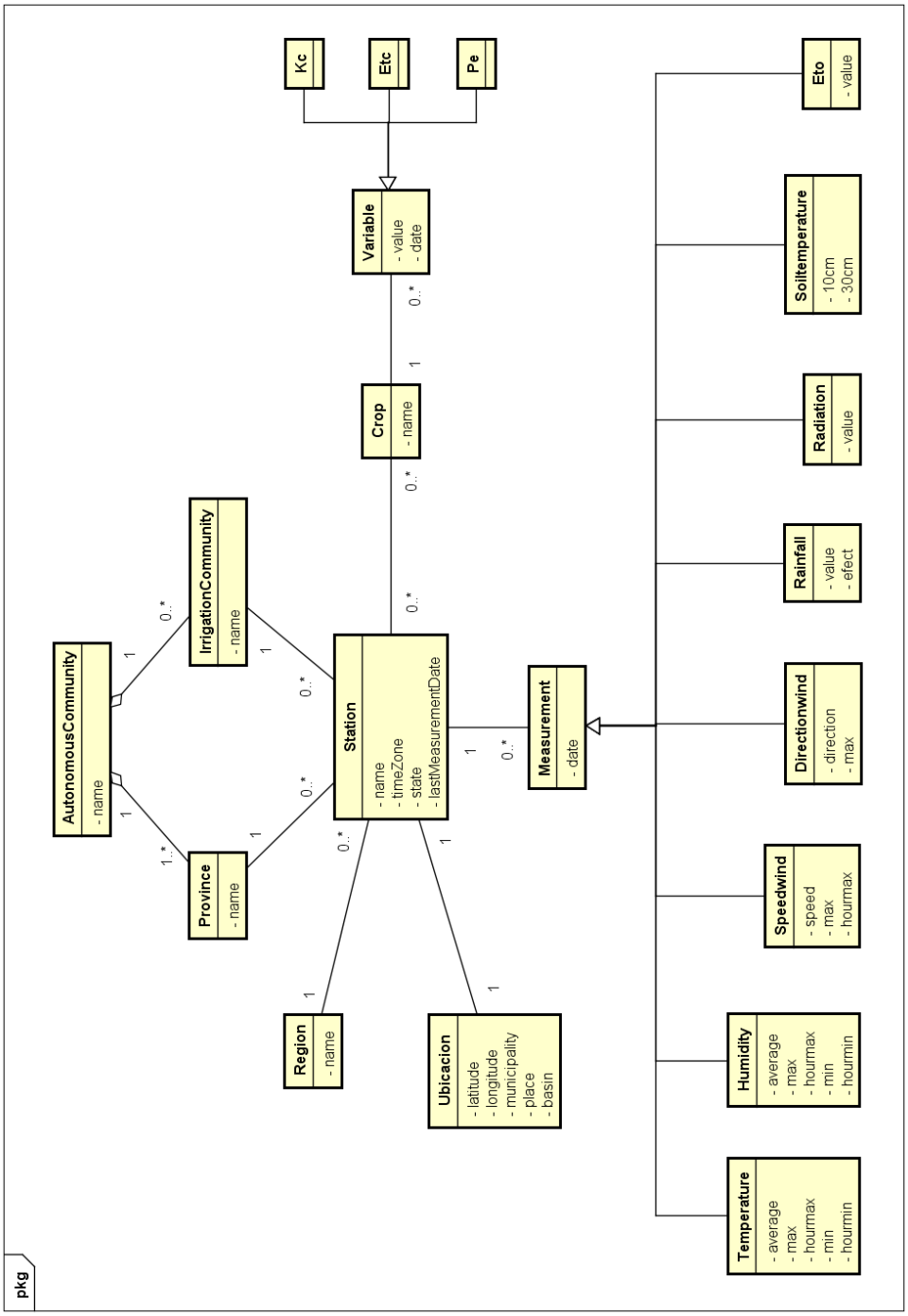


Ilustración 51 - Modelo de dominio iteración

Para poder obtener más información de las estaciones, de nuevo requiere el estudio y análisis de la estructura de las páginas web, debido a que se trata de una página diferente por cada una de las estaciones, que entre ellas mantienen similitud, pero son totalmente diferentes a las estudiadas hasta el momento.

6.7.2. Diseño

El modelo de diseño final de la aplicación es:

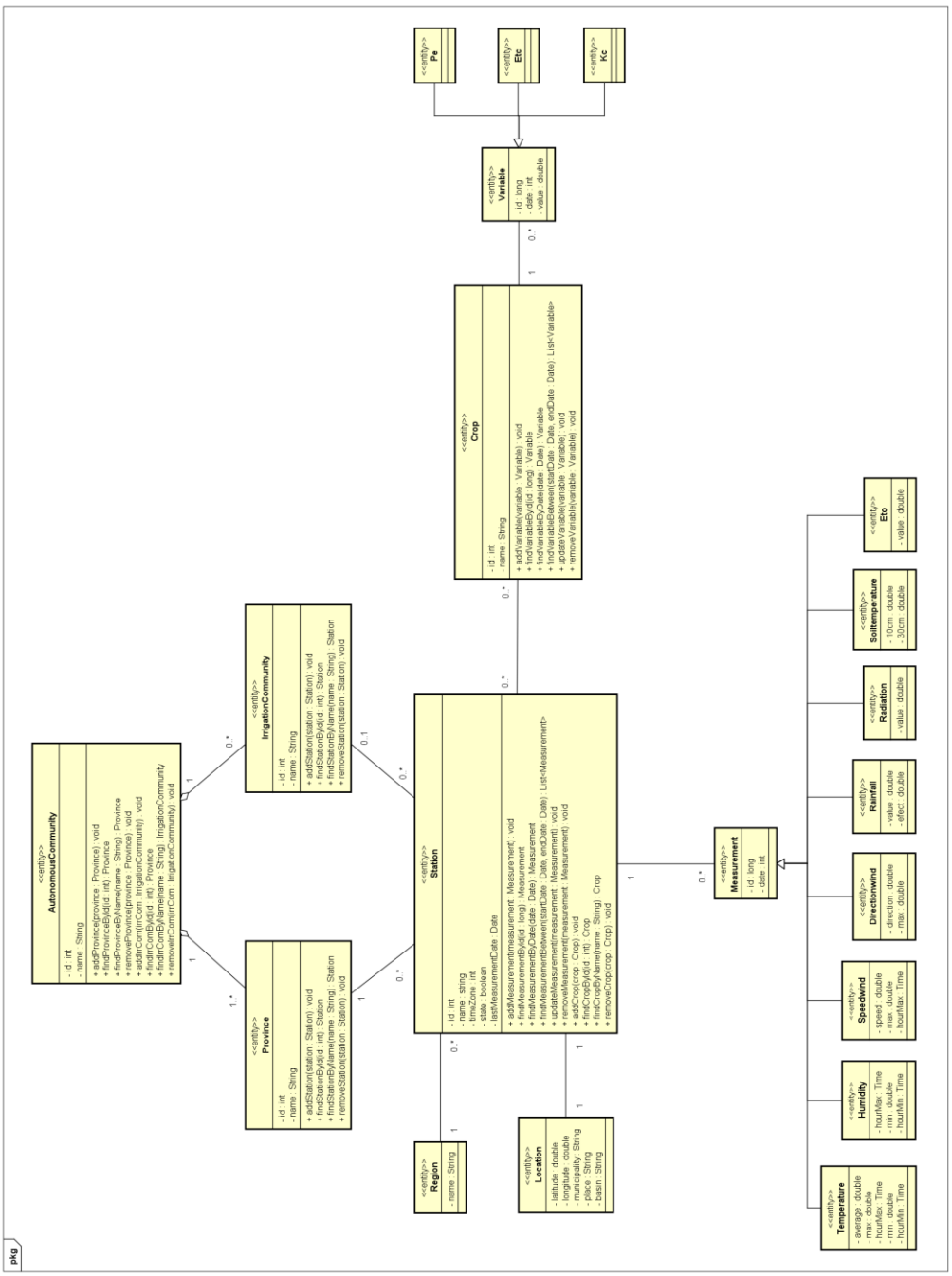


Ilustración 52 - Diagrama de clases iteración 7

6.7.3. Implementación

Finalmente la arquitectura lógica de la aplicación, está dividida en la aplicación principal, más dos componentes, el componente scraper y el componente siar.

- Aplicación principal: expone los recursos de la REST API, empleando los servicios prestados por el componente SiAR.

- Scraper: librería que encapsula las clases comunes de web scraping que pueden ser utilizadas por cualquier otro componente.

- Siar: componente con toda la lógica especifica relacionada con el servicio SiAR.

La arquitectura lógica de la aplicación con sus respectivos paquetes y clases es:

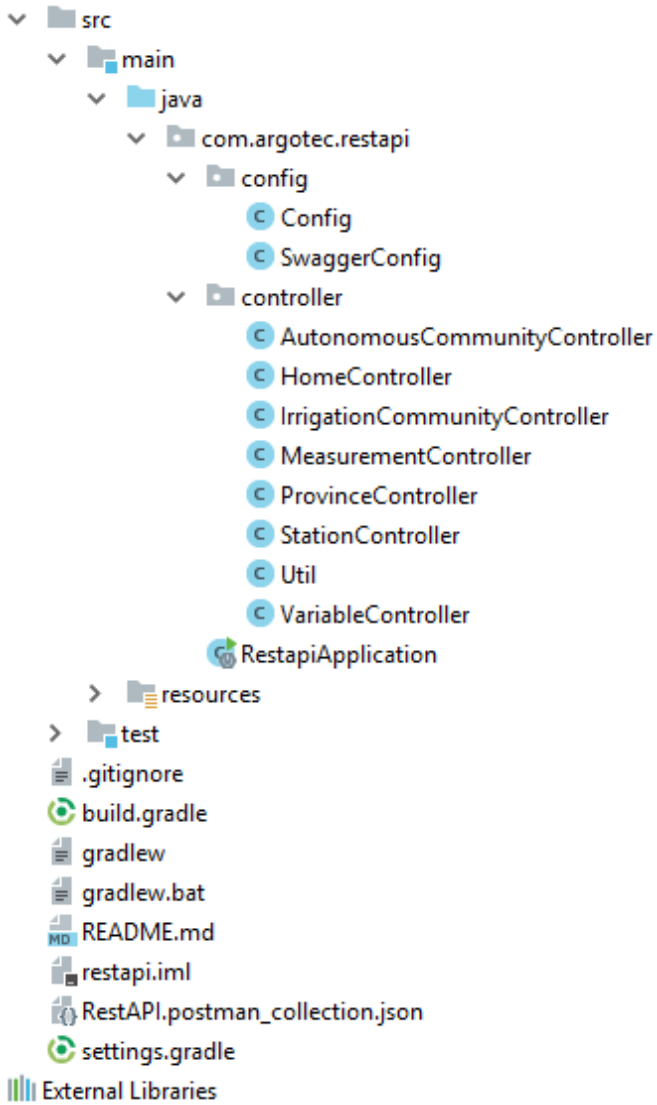


Ilustración 53 - Arquitectura lógica aplicación

La arquitectura lógica del componente SiAR, con sus respectivos paquetes y clases es:

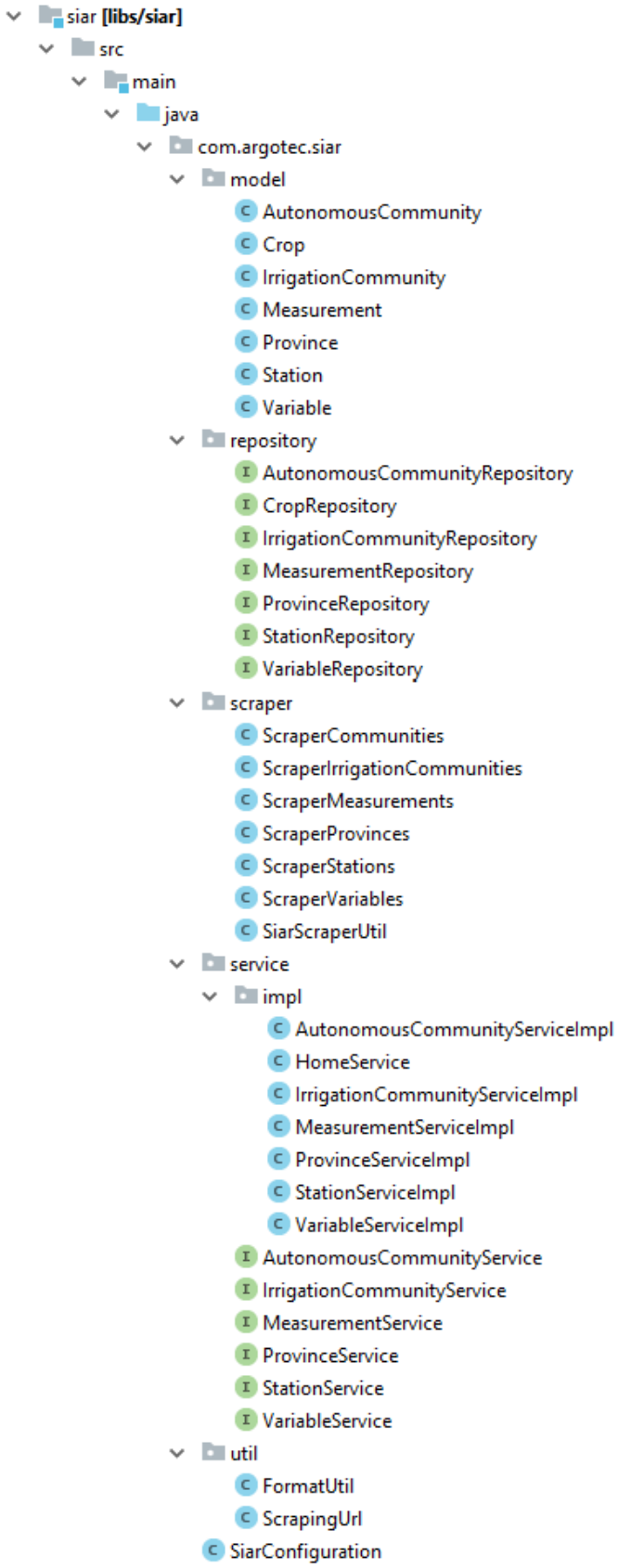


Ilustración 54 - Arquitectura lógica SiAR

La arquitectura lógica del componente scraper es:

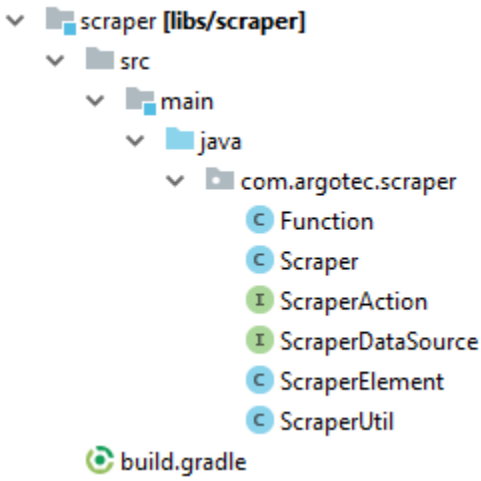


Ilustración 55 - Arquitectura lógica componente Scraper

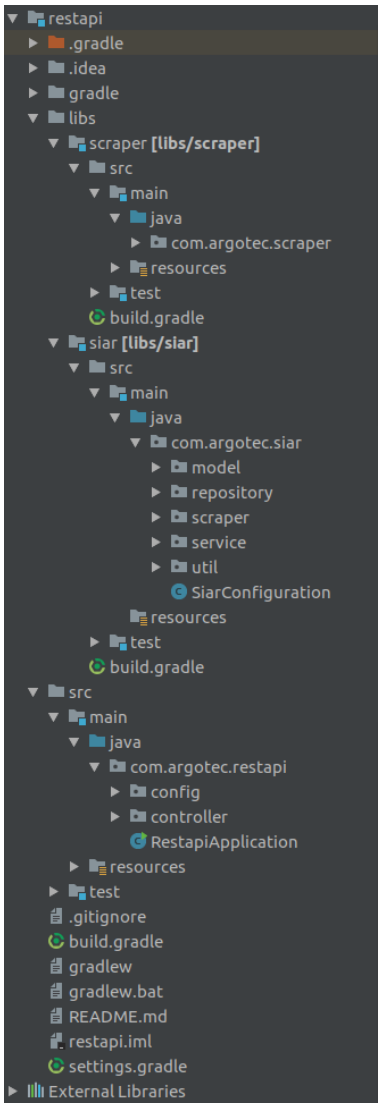


Ilustración 56 - Arquitectura lógica final

Integro Swagger, en busca de alcanzar uno de los objetivos comentados al principio sobre la importancia de disponer de una buena documentación para tu API, para ello incluyo al proyecto la implementación Springfox de la especificación de Swagger.

```
compile('io.springfox:springfox-swagger2:2.9.2')

compile('io.springfox:springfox-swagger-ui:2.9.2')
```

Springfox-swagger2 contiene las características principales de Springfox que permiten la creación de una documentación API con Swagger 2

Springfox-swagger-ui contiene la interfaz de usuario de Swagger que muestra la documentación de Swagger en:

http://localhost:8080/swagger-ui.html

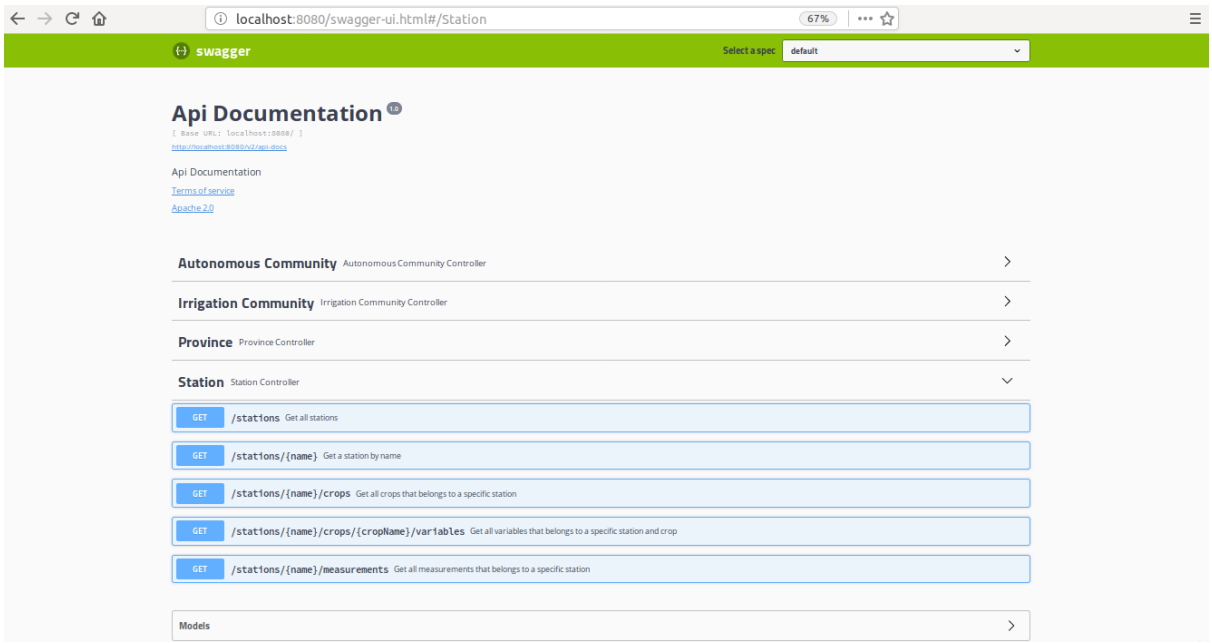


Ilustración 57 - Swagger documentación

```yaml
swagger: '2.0'
info:
  description: Api Documentation
  version: '1.0'
  title: Api Documentation
  termsOfService: 'urn:tos'
  contact: {}
  license:
    name: Apache 2.0
    url: 'http://www.apache.org/licenses/LICENSE-2.0'
host: 'localhost:8080'
basePath: /
tags:
  - name: Autonomous Community
    description: Autonomous Community Controller
  - name: Home
    description: Home Controller
  - name: Irrigation Community
    description: Irrigation Community Controller
  - name: Province
    description: Province Controller
  - name: Station
    description: Station Controller
paths:
  /v1/communities:
    get:
      tags:
        - Autonomous Community
      summary: Get all communities
      operationId: getAllCommunitiesUsingGET
      produces:
        - '*/*'
      parameters:
        - name: ws
          in: query
          description: ws
          required: false
          type: boolean
      responses:
        '200':
          description: OK
          schema:
            type: array
            items:
              $ref: '#/definitions/AutonomousCommunity'
```

Ilustración 58 - Swagger

Es el momento de resolver el problema relacionado con Selenium y el hecho de que lance el navegador Firefox en primer plano, y después de descartar PhantomJS, un navegador sin interfaz gráfica, debido a que es un proyecto al que se ha dejado de dar soporte, encuentro como solución definitiva una opción de Firefox que implementa a partir de su versión 55, y que permite usar Firefox también sin interfaz gráfica.

```java
public Scraper() {
    System.setProperty("webdriver.gecko.driver", "/home/user/codigoTFG-20180629T185202Z-001/geckodriver");

    FirefoxBinary firefoxBinary = new FirefoxBinary();
    firefoxBinary.addCommandLineOptions("--headless");

    FirefoxOptions firefoxOptions = new FirefoxOptions();
    firefoxOptions.setBinary(firefoxBinary);

    webDriver = new FirefoxDriver(firefoxOptions);
}
```

Ilustración 59 - Firefox Headless

Como resultado de los diferentes procesos de encapsulamiento, consigo disponer de una serie de métodos encadenados, que facilitan la lectura y comprensión del código, haciendo que el proceso de web scraping, sea una narración de acciones encadenadas.

```
@Override
public List<Measurement> scrapeAll() {
    try (Scraper scraper = source(ScrapingUrl.getUrlDq)) {

        scraper.doAction(selectAuCommunityAction(station.getProvince().getAutonomousCommunity().getName()))
                .doAction(selectProvinceAction(station.getProvince().getName()))
                .doAction(addStation(station.getName()))
                .doAction(setStartDate(startDate))
                .doAction(setEndDate(endDate));

        scraper.findRootElement(By.id("ContentPlaceHolder1_btnConsultar"))
                .click()
                .and()
                .waitSeconds(2)
                .switchTo()
                .waitSeconds(2);

        return toMeasurements(station, Function.createTable(scraper.getWebDriver()));
    }
}
```

Ilustración 60 - Métodos encadenados

Además aprovecho para transformar partes del código a programación funcional, aprovechando las nuevas opciones que permite Java en su última versión, de forma que el código pasa a ser más legible y simple.

```
Station station = stationRepository
        .findAll()
        .stream()
        .filter(s -> stringsAreEqualAfterNormalization(s.getName(), stationName))
        .findFirst()
        .orElseGet(() -> {
            log.info("Station '{}' not found, fetching using web scraper", stationName);
            Station _station = new ScraperStationInfo().scrapeOne(stationName);
            if (_station != null) {
                Province province = provinceRepository.findByName(_station.getProvince().getName());
                _station.setProvince(province);
                stationRepository.save(_station);
            }
            return _station;
        });
```

Ilustración 61 - Programación funcional

Por cada petición del web driver proporcionado por Selenium, se crea un proceso en el sistema que en caso de no ser explícitamente cerrado puede provocar problemas de fuga de memoria o bloqueo innecesario de recursos del sistema.

Mediante la encapsulación realizada con la clase Scraper sobre el WebDriver, se ha podido definir a éste como un recurso para ser empleado en cláusulas try. Consiguiendo que sea automático el cierre del recurso por parte del sistema independientemente de su resultado, ya sea finalizando con éxito o generado una excepción durante su ejecución. Java internamente cierra implícitamente el recurso en la sección finally.

```
@Override
public List<AutonomousCommunity> scrapeAll() {
    log.info("Fetching autonomous communities using WebScrapper");
    try (Scraper scraper = source(ScrapingUrl.getUrlDg)) {
        return findSelect(scraper)
                .getOptions()
                .stream()
                .map(this::createEntityFromOption)
                .collect(Collectors.toList());
    }
}
```

Ilustración 62 - Recurso Scraper

También añado Lombok, una librería para Java que usa un conjunto de anotaciones reducido y permite simplificar aún más el código.

```
compile group: 'org.projectlombok',  name: 'lombok', version:  '1.18.0'
```

Más información en su página oficial:

https://projectlombok.org/

Después de estudiar las paginas relacionadas con la información adicional de las estaciones, y de comprobar el excesivo tiempo que necesita Selenium para procesar y navegar por cientos de páginas, al constatar que se trata de páginas web estáticas, decido recuperar una de las herramientas empleadas en el transcurso de este proyecto, JSoup, la cual demuestra en este escenario, ser muy potente y muy eficiente, reduciendo los tiempos exponencialmente.

6.7.4. Pruebas

En el caso de que se inicie la aplicación por primera vez, la base de datos no estaría poblada, por lo que sería necesario iniciar los procesos de web scraping.

A lo largo del desarrollo he podido descubrir ciertas dificultades relacionadas con información inicial que la aplicación demanda, derivada por ejemplo, de la información que se obtiene de una estación, la cual contiene únicamente la provincia, y de esta no se puede inferir la comunidad a la que pertenece.

Por lo tanto, por razones prácticas, cuando la aplicación arranca, siempre comprueba primero si tiene las comunidades y las provincias almacenadas en la base de datos, y en caso negativo, se obtienen mediante web scraping.

En esta etapa se realizan las pruebas finales, prestando especial atención al funcionamiento de los scraper, la obtención de datos, su almacenamiento, y la definición de la API REST.

El resultado final de esta iteración es un producto final para su posible entrega.

6.8. Iteración 8

En esta iteración los objetivos principales han sido, por un lado, terminar de pulir algunos detalles en el código de la aplicación y por otro, terminar de completar la memoria, dando estilo, y mejorando algunos detalles de presentación.

7. API - Endpoints

Método	URL	Descripción
GET	/v1/index	Devuelve la página de inicio
GET	/v1/communities	Devuelve una lista con todas las comunidades
GET	/v1/communities/{name}	Devuelve la información de una comunidad concreta
GET	/v1/communities/{name}/provinces	Devuelve todas las provincias de una comunidad
GET	/v1/communities/{name}/irrigation-communities	Devuelve todas las comunidades de regantes de una comunidad
GET	/v1/provinces/	Devuelve una lista con todas las provincias
GET	/v1/provinces/{name}	Devuelve la información de una provincia concreta
GET	/v1/provinces/{name}/stations	Devuelve todas las estaciones de una provincia
GET	/v1/irrigation-communities	Devuelve todas las

		comunidades de regantes
GET	/v1/irrigation-communities/{name}	Devuelve la información de una comunidad de regantes concreta
GET	/v1/irrigation-communities /{name}/stations	Devuelve todas las estaciones de una comunidad de regantes
GET	/v1/stations/	Devuelve todas las estaciones
GET	/v1/stations/{name}	Devuelve la información de una estación concreta
GET	/v1/stations/{name}/crops	Devuelve todos los cultivos de una estación
GET	/v1/stations/{name}/measurements?startDate={startDate}&endDate={endDate}	Devuelve todos los parámetros meteorológicos de una estación en un periodo de tiempo determinado
GET	/v1/stations/{stationName}/crops/{cropName}/variables?startDate={startDate}&endDate={endDate}	Devuelve todas las variables agroclimáticas para un cultivo en una estación en un periodo de tiempo determinado
POST	/v1/stations/{name}/measurements	Crea una medida con la información contenida en el payload

POST	/v1/stations/{stationName}/crop/{cropName}/variables	Crea una variable con la información contenida en el payload
PUT	/v1/stations/{name}/measurements/{id}	Actualiza una medida con la información contenida en el payload
PUT	/v1/stations/{stationName}/crop/{cropName}/variables/{id}	Actualiza una variable con la información contenida en el payload
DELETE	/v1/stations/{name}/measurements/{id}	Borra una medida
DELETE	/v1/stations/{stationName}/crop/{cropName}/variables/{id}	Borra una variable

8. Seguimiento

8.1. Seguimiento temporal

Noviembre					
Lunes	**Martes**	**Miércoles**	**Jueves**	**Viernes**	**Horas**
-	-	-	-	-	-
5	5	5	5	5	25
-	-	-	-	-	-
-	-	-	-	-	-
5	5	5	5	-	20
				Horas Totales	**45**

Diciembre					
Lunes	**Martes**	**Miércoles**	**Jueves**	**Viernes**	**Horas**
-	-	-	-	5	5
5	5	-	5	-	15
5	5	5	5	5	25
5	5	5	5	5	25
-	-	-	-	-	-
				Horas Totales	**70**

Enero					
Lunes	**Martes**	**Miércoles**	**Jueves**	**Viernes**	**Horas**
-	5	5	5	5	20
5	5	5	5	5	25
5	5	5	5	5	25
5	5	5	5	5	25

5	5	5	-	-	15
				Horas Totales	**110**

Febrero					
Lunes	**Martes**	**Miércoles**	**Jueves**	**Viernes**	**Horas**
-	-	-	5	5	10
5	5	5	5	5	25
5	5	5	5	5	25
5	5	5	5	5	25
5	5	5	-	-	15
				Horas Totales	**100**

Marzo					
Lunes	**Martes**	**Miércoles**	**Jueves**	**Viernes**	**Horas**
-	-	-	5	5	10
-	-	-	-	-	-
-	-	-	-	-	-
-	-	-	-	-	-
-	-	-	-	-	-
				Horas Totales	**10**

Junio					
Lunes	**Martes**	**Miércoles**	**Jueves**	**Viernes**	**Horas**
-	-	-	-	-	-
-	-	-	-	-	-
-	-	-	-	-	-
5	5	5	5	5	25
5	5	5	5	5	25

Horas Totales	**50**

Dando como resultado:

Horas totales proyecto	
Noviembre	45
Diciembre	70
Enero	110
Febrero	100
Marzo	10
Junio	50
Horas Totales	**385**

Teniendo en cuenta las horas invertidas fuera del horario laboral, suman en total más de 400 horas.

8.2. Seguimiento con Trello

Ilustración 63 - Seguimiento tareas Trello

La herramienta usada para el seguimiento del proyecto ha sido Trello, que a través de sus listas y tarjetas puedes organizar el desarrollo de tu trabajo, de forma intuitiva y muy visual.

9. Conclusiones

La agricultura moderna ha aumentado los impactos negativos sobre el medio ambiente, con la destrucción de los entornos naturales, la contaminación por químicos, la salinización del suelo, o la deforestación, y en ese contexto, tenemos la responsabilidad de poner todos los medios disponibles a nuestro alcance para minimizar los efectos negativos.

Actualmente es posible aprovechar los avances tecnológicos disponibles, para cambiar los procesos productivos y optimizar el uso de unos recursos limitados, obteniendo no solo una mejora en la productividad, sino el beneficio de contribuir de forma positiva a la sostenibilidad del medio ambiente a largo plazo.

Por eso, personalmente considero que el Trabajo de Fin de Grado ha sido una gran oportunidad no solo para adquirir conocimientos y habilidades específicas, sino también para poder desarrollar una actividad que realmente me gusta, y que tiene como objetivos metas más allá de las puramente económicas, y que pueden suponer pequeñas contribuciones a mejorar la forma que tenemos de relacionarnos con el medio ambiente.

Por lo que espero, que finalmente la aplicación termine siendo de ayuda en la optimización de recursos y en la generación de recomendaciones más precisas.

Seguir una metodología ágil para mí ha sido una experiencia nueva, todo lo que había hecho anteriormente había estado ligado a metodologías más tradicionales, y he aprendido de las metodologías ágiles, que son un conjunto de buenas prácticas para ayudar a alcanzar los objetivos al finalizar un proyecto, por lo que es importante que sean lo suficientemente flexibles para adaptarse a las necesidades específicas de cada equipo y proyecto, evitando barreras y limitaciones en los procesos de desarrollo.

Aunque también he de decir, que en el fondo no se diferencia tanto de un desarrollo tradicional, es hacer lo mismo, pero de diferente manera, es decir, su punto fuerte es la forma de llegar a una misma meta, alcanzar el éxito al llegar al final de un proceso de desarrollo.

De las actividades realizadas, destacaría las Standup, que es una buena forma de estar conectado al equipo y a un proyecto de manera continua, las IPM, una forma conjunta e interesante de planificar, gestionar el trabajo y aclarar objetivos, y por ultimo las retro, una buena práctica de autoevaluación y de reflexión.

Respecto al TDD, aunque en España todavía no es una práctica tan extendida como podría ser deseable, sí que es cierto que cada vez está aumentando más su uso por parte de los desarrolladores y de las empresas, asociado al continuo aumento de la implantación de las metodologías ágiles en los entornos de trabajo, aunque todavía tiene que hacer frente a, por un lado, que a día de hoy en no muchos programas académicos se trata en profundidad el desarrollo en TDD, por lo que es bastante desconocido para muchos programadores, y por otro, el miedo de muchas desarrolladores al cambio de mentalidad que propone el TDD, ya que no se trata solo de hacer pruebas como muchos pueden creer, sino se trata de cambiar primero la forma de pensar, y luego la forma de programar.

Uno de las ventajas más interesantes desde mi punto de vista, es el hecho de que al obligarnos a pensar en las pruebas que nuestro código tiene que pasar, eso ayuda a tener más claro cuáles son los requisitos que el código tiene que cumplir, y mejora el diseño.

Sobre las pruebas unitarias y de integración me parece que aportan un gran beneficio en el desarrollo del software, y mejoran su calidad.

Las pruebas evitan muchos posibles errores de código a través de su detección de forma rápida, lo que también implica ahorro de tiempo y de costes a largo plazo. También reducen los tiempos de depuración, simplificando la resolución de los problemas, al solo probar pequeños elementos del código. Es una buena forma de refactorizar el código, evitando código duplicado, y haciendo un código más limpio.

Para mí personalmente, este proyecto ha sido una buena oportunidad para descubrir y dar los primeros pasos en el TDD, una práctica que me ha parecido tan práctica como útil, y en la que pienso continuar avanzando y mejorando para ser un mejor profesional, al menos desde mi punto de vista.

Quiero pensar que el TDD viene para quedarse, y que en un futuro no muy lejano, será una asignatura obligatoria en todas las universidades.

Considero que el uso de tecnologías y herramientas de software gratuito, conllevan muchas ventajas, desde la más evidente, el ahorro directo en los costes para cualquier desarrollo o empresa evitando tener que comprar lo que en muchos casos suponen licencias bastante caras, hasta el tener asociadas una gran comunidad, que suele ayudar y facilitar la resolución de problemas.

En relación a las tecnologías empleadas en el desarrollo del proyecto, hay varias cosas que me gustaría destacar:

- Después de finalizar el trabajo, sigo manteniendo mi preferencia de entornos Linux sobre otros como Windows o Mac.

- Personalmente me gusta la programación en Java, y es el lenguaje con el que me siento más cómodo a día de hoy. Considero la programación orientada a objetos como la referencia a la hora de programar, pero tengo que reconocer que me está resultando muy atractivo la introducción de la programación funcional en las últimas versiones de Java.

- Intellij IDEA es un IDE de lo más recomendable, que destaca por muchas cosas, pero que se pueden resumir en que te hace la vida más fácil a la hora de programar, y hace que la experiencia sea altamente satisfactoria.

- Mi experiencia con Spring Boot la describiría como sorprendente, es un framework muy potente que simplifica algunos aspectos que de otra forma se podrían transformar en verdaderos dolores de cabeza, muchos seguro que entienden de lo que hablo, así que es de agradecer poder hacer uso de una tecnología que a veces parece hasta que hace magia.

- Gradle sencillamente es genial, simplifica muchísimo la gestión de dependencias, de forma transparente, y comparado como Maven, de una forma realmente sencilla.

- Una de las cosas que he aprendido una vez fuera, digamos del círculo más académico, es la importancia de los sistemas de control de versiones, en el mundo real, no se puede desarrollar un proyecto sin hacer uso de estas herramientas, al menos si no deseas sumergirte en el caos, y para ese fin, Git es el rey.

- La experiencia con el repositorio GitLab ha sido buena, hasta ahora tenía predilección por Bitbucket, pero ahora ya no sabría con cual quedarme, algo positivo ya que siempre es mejor disponer de más de una opción.

- Respecto a las herramientas de web scraping, ha sido una experiencia completamente nueva para mí, y que me deja una buena sensación, me llevo lo aprendido en lo que se refiere a técnicas de web scraping, y destaco Selenium como medio para la obtención de información, aunque al final del proyecto, JSoup termino por demostrar también ser una herramienta bastante potente y útil.

- Postman es una herramienta bastante útil y cómoda de usar en lo que se refiere a las llamadas a los diferentes REST endpoint.

- Usar H2 en las fases iniciales de un desarrollo facilita las cosas, teniendo una base de datos funcionando de forma rápida y sencilla desde el primer minuto. Y respecto a PostgreSQL, la sensación también ha sido buena, para mí era una desconocida, siempre había usado MySQL, y puede que a partir de ahora eso cambie. Aunque me he quedado con ganas de probar una base de datos no relacional como MongoDB. Y no me gustaría dejar en segundo plano a Hibernate, que facilita el mapeo entre objetos y la base de datos.

- Mi experiencia con las herramientas de testeo también ha sido positiva, tanto JUnit, Hamcrest, como Mockito son herramientas muy adecuadas en la creación de pruebas.

- Respecto a Trello, simplemente me encanta, no solo me parece útil para un desarrollo software, sino para casi cualquier cosa en lo que se refiere a organización y planificación de tareas, es muy intuitiva, y cómoda de usar, y sobre Pivotal Tracker, aunque su papel en el proyecto ha sido más simbólico, las pruebas han sido muy satisfactorias, y desde entonces, no se ha dejado de utilizar.

- Por último, dentro de los servicios de Google, destacar Drive, siempre útil en lo que se refiere a compartir un documento, y la posibilidad de editar en línea, dejando a un lado las preocupaciones de tener que guardar y de saber si estas en la última versión.

En resumen, en general la experiencia con las diferentes herramientas y tecnologías ha sido muy buena, algunas ya las utilizaba, otras solo las conocía, y otras eran totalmente desconocidas, y su incorporación ha sido el resultado de una fase de investigación inicial en busca de las mejores opciones que se adaptasen lo mejor posible a las necesidades del proyecto, y llegados a este punto, puedo afirmar que en general ha sido un completo acierto.

Todo lo relacionado con web scraping, ha sido uno de los factores que han añadido cierta complejidad al proyecto, pero al mismo tiempo lo han hecho aún más interesante. También me ha invitado a reflexionar sobre los límites entre lo legal e ilegal, y ético, en lo que se refiere a la extracción de datos, y lo que puede implicar, si no se hace un buen uso de este tipo de herramientas, llegando a suponer en algunos casos, cruzar ciertos límites que pueden ser cuestionables, aunque

también hay que decir, que en la mayoría de los casos, la finalidad al igual que en este proyecto, no es más que proporcionar un servicio que por las razones que sean, hasta el momento no se proporciona.

A parte de que Spring Boot integra en si varios patrones de diseño, se ha buscado siempre aplicar patrones de diseño, y dar soluciones lo más estandarizadas en todo lo que ha sido posible.

Evaluando las planificaciones iniciales, tanto temporales como de costes, se puede decir que aunque ha existido cierta desviación, se puede considerar razonable. En la caso de la planificación temporal, se ha producido una desviación de algo menos del 15 %, lo que ha supuesto para la planificación de costes, un incremento de algo menos del 12’5 %. Un riesgo que se tenía contemplado desde el inicio del proyecto.

Si tuviese que destacar algún aspecto negativo durante esta etapa del TFG, haciendo autocrítica, sería lo fácil que a veces resulta perderse en los detalles, olvidando en algunos momentos lo realmente importante, y con eso me refiero a que en determinados momentos, las ganas de profundizar en algunos temas, te desvía de los verdaderos objetivos, lo cual genera un coste, y en la mayoría de las veces, al menos de tiempo, como ya he podido experimentar a lo largo de este proyecto.

Y por último, en lo que refiere a la versión final de la aplicación, creo que se han alcanzado los objetivos marcados, que es un buen punto de partida para que el proyecto siga creciendo e integrando nuevos elementos, y me siento satisfecho, no solo por los resultados finales sino por todo lo aprendido durante el camino.

10. Líneas futuras

Como líneas futuras quiero centrarme en plantear lo que a mi parecer podrían ser los cinco puntos más interesantes y relevantes para próximos pasos en el desarrollo de la aplicación:

1) Nuevas fuentes de datos

La primera línea futura y más obvia, es la de incluir otras fuentes de datos, como puede ser AEMET, la Agencia Estatal de Meteorología, ya sea por medio del acceso general a través de su página web:

Ilustración 64 - AEMET

O desde su API, AEMET OpenData API:

Ilustración 65 - Open Data AEMET

Más información en su página oficial:

http://www.aemet.es/es/portada

2) Implementar nuevas funcionalidades

Otra línea evidente, sería la implementación de nuevas funcionalidades, y una bastante interesante a modo de ejemplo, podría ser incluir geolocalización, de forma que se pueda obtener cual es la estación más cercana a un punto cualquiera, mediante la latitud y longitud, de forma que la generación de recomendaciones sería lo más precisa posible.

3) Desarrollar el front-end:

Otra línea futura sería trabajar en el back-end, añadiendo una capa de presentación a la aplicación, mejorando así la experiencia de usuario.

Dado que proporcionó una API bien definida, hay total libertad para decidir la manera de implementarla por parte de la persona encargada de desarrollarla, y planteó dos posibles alternativas, Thymeleaf y Angular.

Thymeleaf es un framework en java de plantillas para XML, XHTML o HTML5, que además dispone de una excelente integración con Spring, una buena alternativa a las JavaServer Pages (JSP), y puede facilitar mucho el trabajo de desarrollo de la vista de la aplicación.

Ilustración 66 - Thymeleaf

Más información en su página oficial:

https://www.thymeleaf.org

Angular es un framework de desarrollo para JavaScript, que permite desarrollar aplicaciones web SPA, o aplicaciones de una sola página, separando por completo el front-end del back-end en una aplicación web. Spring permite la integración con angular de una forma sencilla mediante Gradle.

Ilustración 67 - Angular

Más información en su página oficial:

https://angular.io/

De forma complementaria también se podría hacer uso de Bootstrap, uno de los framework más conocidos para diseño de páginas o aplicaciones web, basado en HTML, CSS y JavaScript, junto a JQuery. También tiene la posibilidad de integrarse fácilmente con Spring.

Ilustración 68 – Bootstrap

Más información en su página oficial:

https://getbootstrap.com

4) Migrar a MongoDB o Hadoop

El tipo de base de datos a utilizar, en su momento fue uno de los aspectos que más dudas género, y al final, por razones más que nada prácticas, se decidió por una relacional, pero gracias a la mínima dependencia que existe entre la aplicación y la base de datos, no supondría un problema realizar una migración a una base de datos no relacional como MongoDB, o pasar a un servicio como el de Google, Hadoop, aunque en este último caso, añadiría cierto grado de complejidad a la migración.

Uno de los puntos fuertes que justificaría el paso, sería su orientación al Big Data, asociado a unas expectativas de manejar datos de forma masiva con la aplicación en un futuro no muy lejano, por lo que podría ser un salto interesante.

Ilustración 69 – MongoDB

Más información en su página oficial:

https://www.mongodb.com

5) Incluir Docker:

Docker es una tecnología basada en contenedores que automatiza el despliegue de aplicaciones dentro de contenedores software, añadiendo una capa más de abstracción. De forma que por ejemplo se puede empaquetar la base de datos o cualquier otra aplicación y sus dependencias, y levantarlo con un simple comando como:

```
$ docker-compose up
```

Ilustración 70 - Docker

Más información en su página oficial:

https://www.docker.com

11. Glosario

A continuación paso a definir algunos de los términos que aparecen a lo largo de la memoria y que no han sido explicados anteriormente:

- Hidrometeoro: conjunto de partículas de agua, liquidas o sólidas, que caen de la atmosfera a la superficie terrestre, como pueden ser lluvia, nieve o granizo.

- Agricultura intensiva: basada en obtener el mayor volumen de producto por unidad de superficie, usando todo tipo de medios y maquinaria, como pueden ser fertilizantes o pesticidas químicos. Se hace un uso intensivo de la tierra, obteniendo una mayor producción que en la agricultura extensiva. Muy relacionada con el agotamiento de recursos.

- Agricultura industrial: basada en la producción masiva, mediante un alto grado de medios técnicos. Directamente relacionada con la deforestación de ecosistemas.

- Ecosistema: sistema biológico compuesto por los organismos vivos y el entorno que los rodea, donde se relacionan.

- Deforestación: proceso generalmente provocado por la acción humana que destruye las superficies forestales, como los bosques.

- Software libre: se fundamenta en los principios de tener la libertad de usar un programa con cualquier finalidad, la libertad de estudiar cómo funciona, y la libertad de poder modificarlo adaptándolo a las propias necesidades.

- Software gratuito: permite su uso sin ningún coste, de forma ilimitada, y en algunos casos, con determinadas restricciones funcionales, de ámbito o temporales.

- Framework: entorno de trabajo que integra y ofrece un conjunto estándar de soluciones conceptuales y tecnológicas para resolver un tipo de problema concreto.

- Spring: framework de Java, que introduce conceptos como la inversión de control o la inyección de dependencias. Más información en:

  https://spring.io/

- Bean: clase Java que sigue una serie de convenciones sobre construcción, nomenclatura de métodos, y comportamiento.

12. Anexos

12.1. Guía para crear un proyecto Spring Boot con Intellij IDEA

Requisitos previos:

- Instalar Java JDK
- Instalar Intellij IDEA

Para este ejemplo usaré la versión Ultimate de Intellij IDEA, la cual incluye directamente un plugin que ahorra el tener que dirigirte primero a la página de Spring Initializr para generar el proyecto.

Para crear un proyecto Spring Boot es necesario seguir los siguientes pasos:

1) Seleccionar “Create New Project”

Ilustración 71 – Spring Boot: Crear proyecto

2) Seleccionar Spring Initializr para generar un proyecto Spring Boot, y la versión JDK que en este caso es la versión 1.8

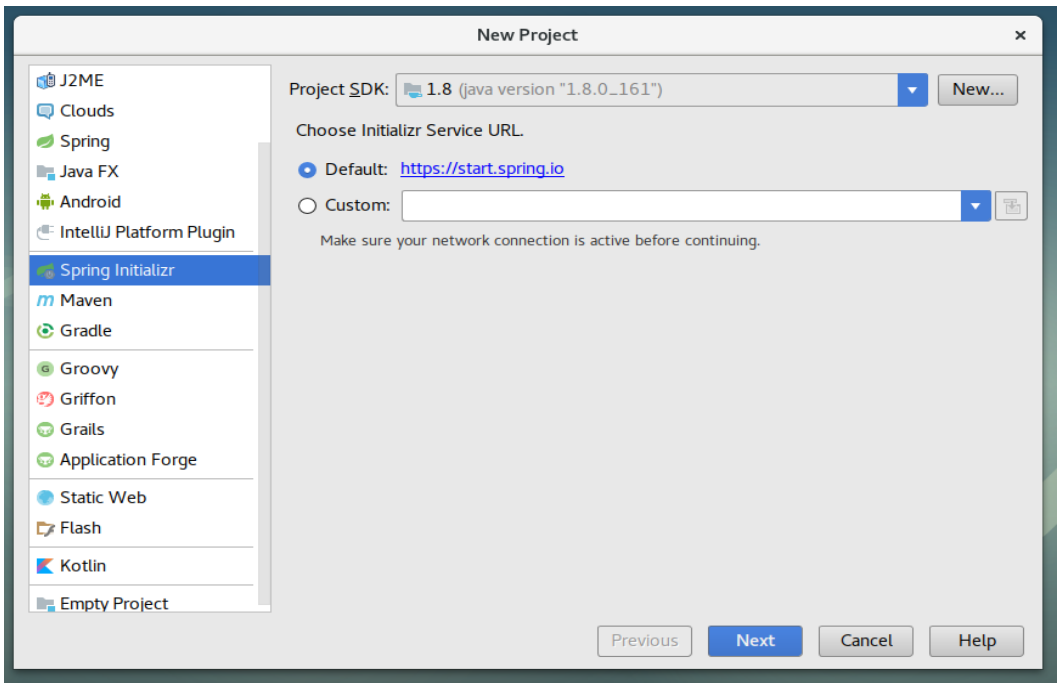


Ilustración 72 - Spring Boot: Seleccionar Spring Initializr

3) Seleccionar Gradle como gestor de dependencias (Ventajas sobre Maven), Java como Lenguaje, y otros detalles del proyecto

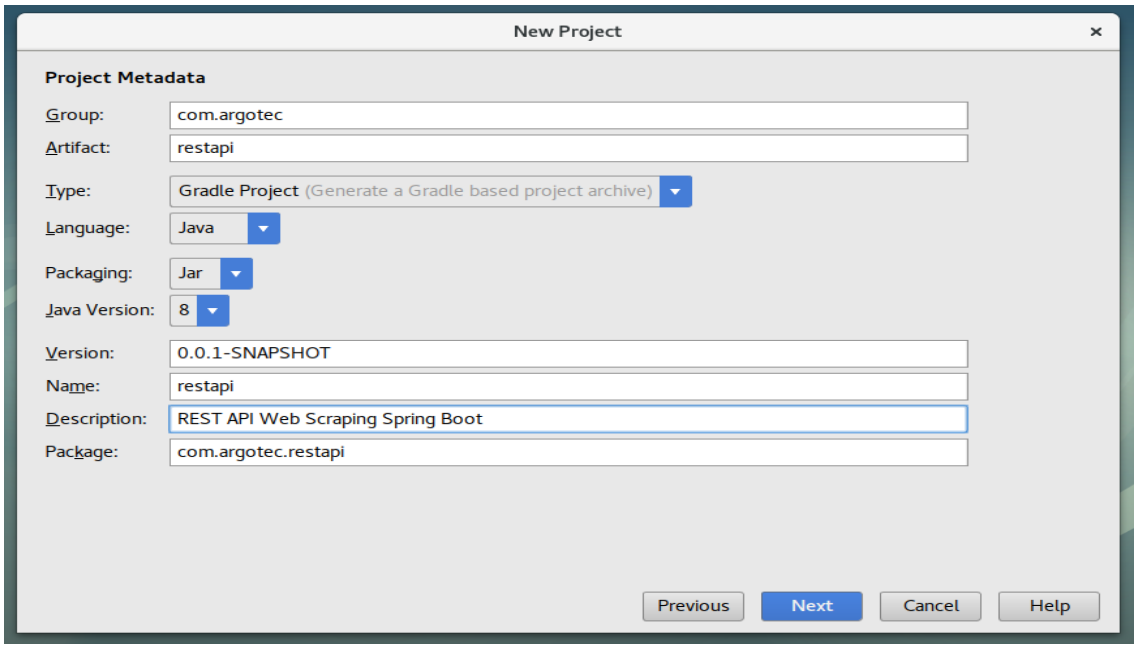


Ilustración 73 - Spring Boot: Seleccionar Gradle, Java y otros detalles

4) Seleccionar las dependencias:

Seleccionar dependencia Web

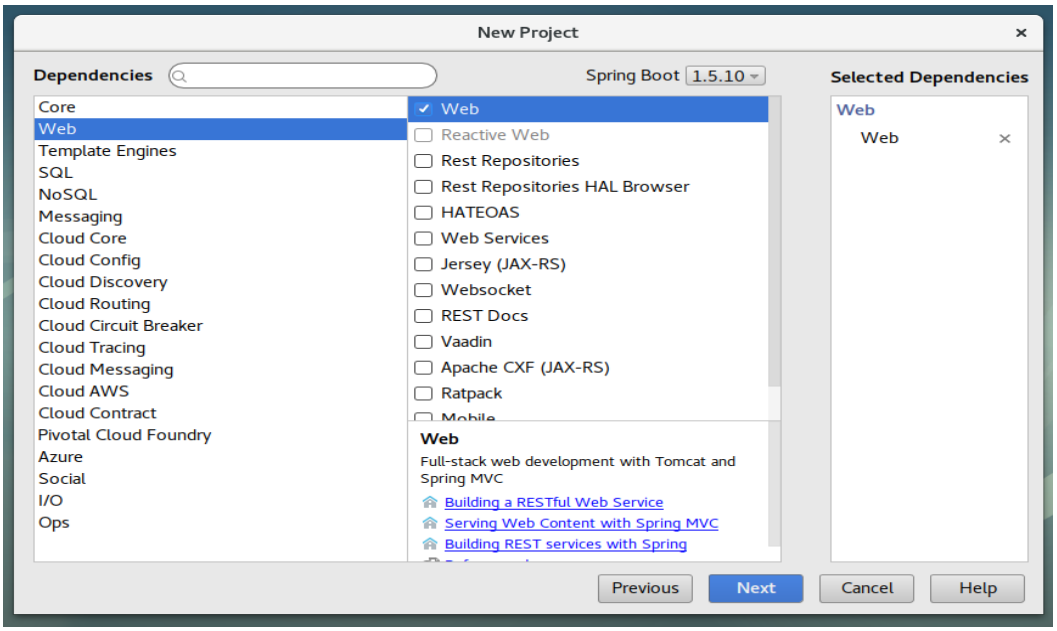


Ilustración 74 – Spring Boot: Dependencia Web

Seleccionar las dependencias JPA y PostgreSQL para la persistencia

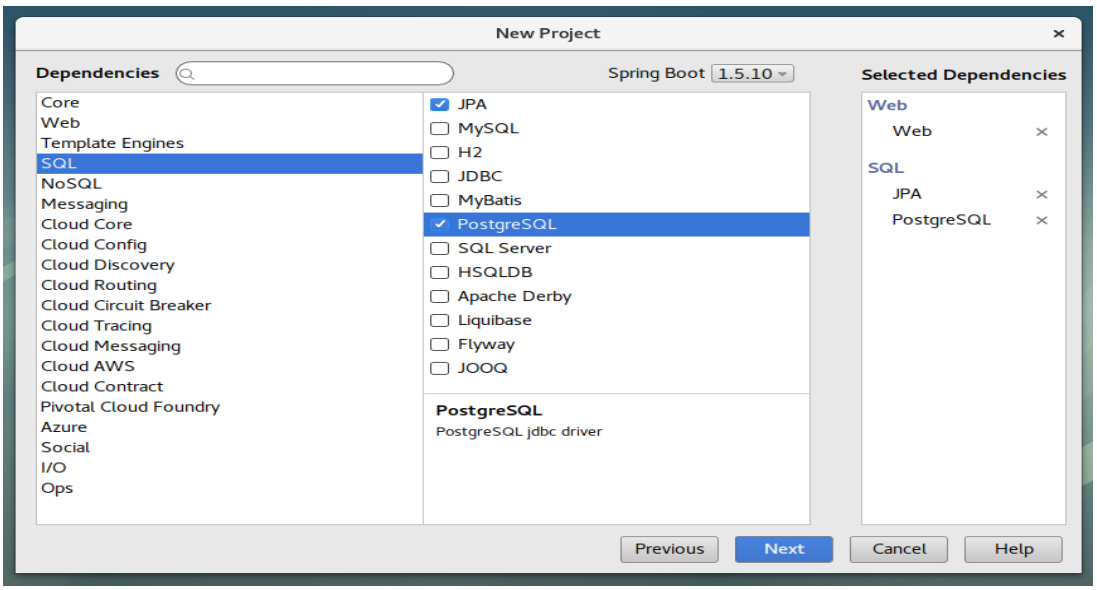


Ilustración 75 - Spring Boot: Dependencias JPA y PostgreSQL

5) Marcar auto-import y mantener el resto de opciones por defecto para el módulo Gradle

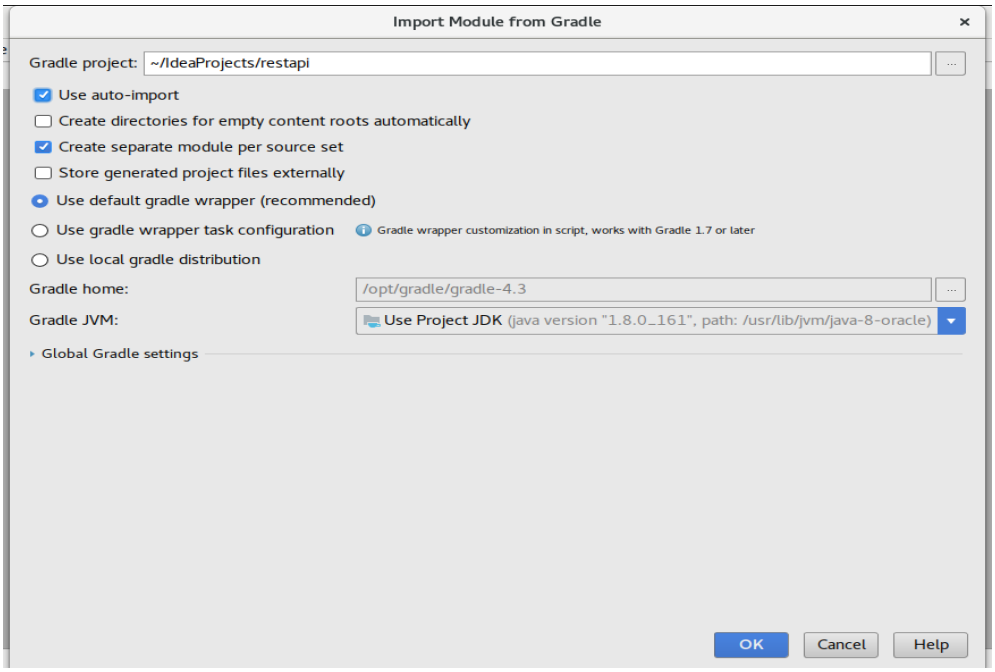


Ilustración 76 - Spring Boot: Módulo Gradle

6) Confirmar el nombre del proyecto y la ruta para finalizar

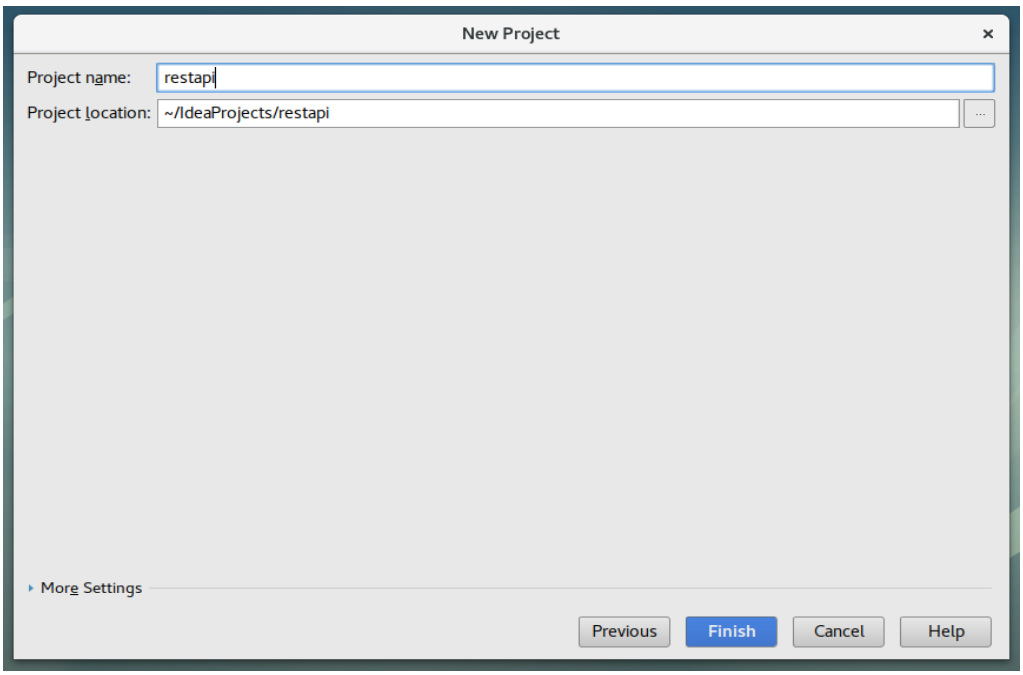


Ilustración 77 - Spring Boot: Nombre y ruta del proyecto

:

Y con estos sencillos pasos ya tenemos creado nuestro primer proyecto Spring Boot, que dispone de su primera clase principal totalmente funcional.

Aunque si intentamos ejecutar la aplicación daría un error ya que todavía no se ha añadido ninguna base de datos, habiendo incluido las dependencias para la persistencia al crear el proyecto.

La base de datos puede ser en memoria, como H2, o una más tradicional como MySQL o PostgreSQL.

```
***************************
APPLICATION FAILED TO START
***************************

Description:

Cannot determine embedded database driver class for database type NONE

Action:

If you want an embedded database please put a supported one on the classpath.


Process finished with exit code 1
```

Ilustración 78 - Error base de datos

Por lo que es tan sencillo como añadir la base de datos que prefieras, y comprobar que la conexión es correcta.

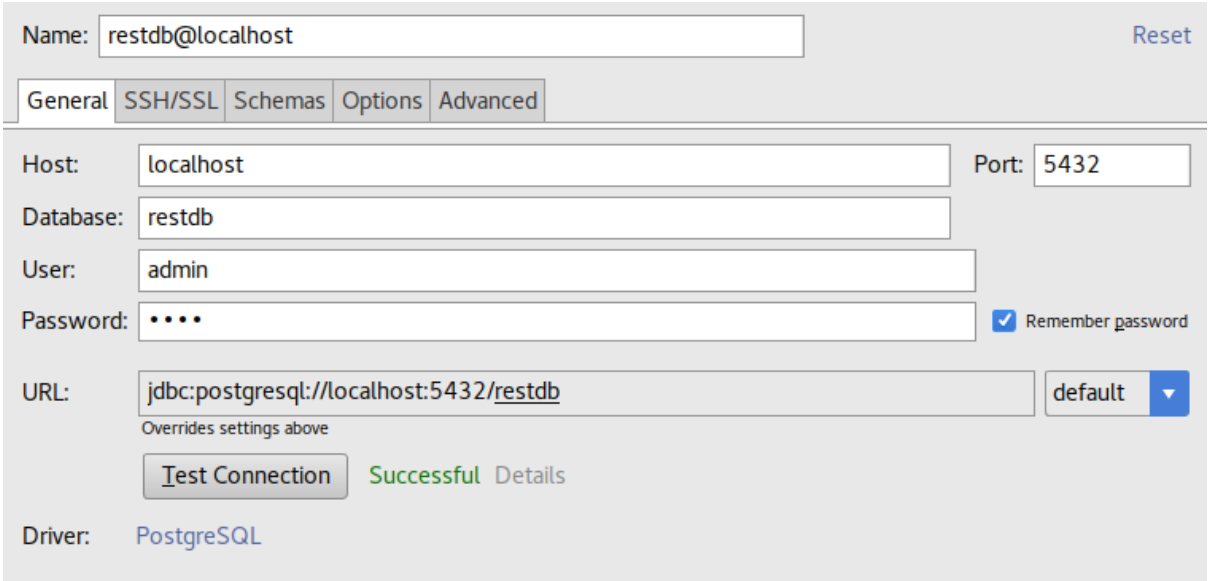


Ilustración 79 - Spring Boot: Configurar base de datos

Y ahora pasamos a añadir al archivo “application.properties” la configuración de la base de datos:

```
#########################
#                       #
#  Database PostgreSQL  #
#                       #
#########################
#                                        //HOST:PORT/database
spring.datasource.url = jdbc:postgresql://localhost:5432/restdb
spring.datasource.username=admin
spring.datasource.password=1234
```

Ilustración 80 - Spring Boot: Propiedades básicas base de datos

Tras lo cual ya podemos ejecutar la aplicación sin problemas:

```
/usr/lib/jvm/java-8-oracle/bin/java ...

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v1.5.10.RELEASE)

2018-02-23 12:18:18.062  INFO 6695 --- [           main] com.argotec.restapi.RestapiApplication
2018-02-23 12:18:18.065  INFO 6695 --- [           main] com.argotec.restapi.RestapiApplication
2018-02-23 12:18:18.109  INFO 6695 --- [           main] ationConfigEmbeddedWebApplicationContext
2018-02-23 12:18:18.977  INFO 6695 --- [           main] trationDelegate$BeanPostProcessorChecker
2018-02-23 12:18:19.280  INFO 6695 --- [           main] s.b.c.e.t.TomcatEmbeddedServletContainer
2018-02-23 12:18:19.289  INFO 6695 --- [           main] o.apache.catalina.core.StandardService
2018-02-23 12:18:19.290  INFO 6695 --- [           main] org.apache.catalina.core.StandardEngine
2018-02-23 12:18:19.379  INFO 6695 --- [ost-startStop-1] o.a.c.c.C.[Tomcat].[localhost].[/]
2018-02-23 12:18:19.380  INFO 6695 --- [ost-startStop-1] o.s.web.context.ContextLoader
2018-02-23 12:18:19.484  INFO 6695 --- [ost-startStop-1] o.s.b.w.servlet.ServletRegistrationBean
2018-02-23 12:18:19.487  INFO 6695 --- [ost-startStop-1] o.s.b.w.servlet.FilterRegistrationBean
2018-02-23 12:18:19.487  INFO 6695 --- [ost-startStop-1] o.s.b.w.servlet.FilterRegistrationBean
2018-02-23 12:18:19.488  INFO 6695 --- [ost-startStop-1] o.s.b.w.servlet.FilterRegistrationBean
2018-02-23 12:18:19.488  INFO 6695 --- [ost-startStop-1] o.s.b.w.servlet.FilterRegistrationBean
2018-02-23 12:18:19.941  INFO 6695 --- [           main] j.LocalContainerEntityManagerFactoryBean
2018-02-23 12:18:19.955  INFO 6695 --- [           main] o.hibernate.jpa.internal.util.LogHelper
```

Ilustración 81 - Spring Boot: Ejecución de Spring Boot

Inicialmente el build.gradle queda de la siguiente forma:

```
buildscript {
    ext {
        springBootVersion = '1.5.10.RELEASE'
    }
    repositories {
        mavenCentral()
    }
    dependencies {
        classpath("org.springframework.boot:spring-boot-gradle-plugin:${springBootVersion}")
    }
}

apply plugin: 'java'
apply plugin: 'idea'
apply plugin: 'org.springframework.boot'

group = 'com.argotec'
version = '0.0.1-SNAPSHOT'
sourceCompatibility = 1.8

repositories {
    mavenCentral()
}


dependencies {
    compile('org.springframework.boot:spring-boot-starter-data-jpa')
    compile('org.springframework.boot:spring-boot-starter-web')
    runtime('org.postgresql:postgresql')
    testCompile('org.springframework.boot:spring-boot-starter-test')
}
```

Ilustración 82 - Build.gradle

12.2. CD-ROM

El contenido del CD-ROM es el siguiente:

- Documento Trabajo Fin de Grado en PDF con el nombre “memoria.pdf”
- Carpeta “restapi” con el código fuente de la aplicación
- Carpeta “documentación adicional” con los ficheros Astah que contienen los diagramas realizados para el proyecto
- Carpeta “recursos adicionales” con el archivo json de Swagger que contiene la especificación de la API. Se puede visualizar entrando en la página http://editor.swagger.io/

12.3. Manual de instalación

Pre-requisitos

- Tener instalada y levantada la base de datos en tu máquina, que en mi caso es PostgreSQL, y configurada en la aplicación. Para poner en funcionamiento PostgreSQL se puede hacer desde línea de comandos con el siguiente comando:

```
$ sudo systemctl start postgresql
```

- Tener el navegador Firefox en su última versión, o al menos a partir de la versión 57.

Ejecutar la aplicación

Los pasos a seguir para poner en marchar la aplicación son muy sencillos, ahí es donde entra la magia de Spring Boot, basta con ejecutar la aplicación, bien desde el propio Intellij IDEA, bien desde un terminal con Gradle, o bien a través de un jar ejecutable.

Si decidimos hacerlo desde el propio IDE, es tan simple como pulsar el item  o seleccionar del menú la opción “Run”.

Si preferimos hacerlo desde una terminal, podemos ejecutar la aplicación usando Gradle mediante el siguiente comando:

```
$ gradle bootRun
```

Con Gradle también es posible crear un archivo jar ejecutable con:

```
$ gradle build
```

Y ejecutarlo de la siguiente forma:

```
$ java – jar build/libs/restapi-0.0.1-SNAPSHOT.jar
```

Presuponiendo que la aplicación está ejecutándose en una maquina en local, llegados a este punto, solo es necesario acceder a tu navegador favorito, y escribir una URL con alguno de los endpoint disponibles, que puedes encontrar en este mismo documento.

```
http://localhost:8080/<endpoint>
```

Y ya puedes disponer de los servicios que te ofrece mi aplicación de una forma rápida y cómoda.

12.4. Manual para la API

Si quieres usar la API la mejor opción es utilizar la herramienta Postman, con la que puedes realizar llamadas GET, POST, PUT y DELETE, además de poder crear tus propias colecciones con los endpoint que habitualmente más utilices.

Si quieres hacer una petición, basta con especificar la URL del endpoint deseado, y al enviar la petición, obtendrás como respuesta los resultados de tu consulta en formato JSON.

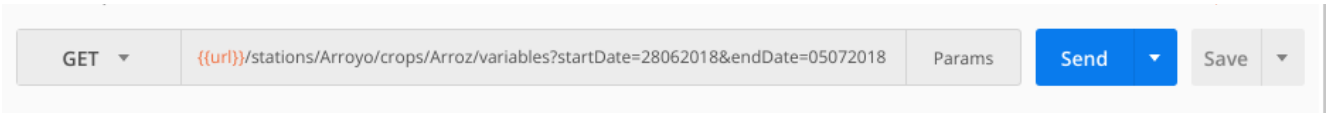


Ilustración 83 - Petición obtener variables

También puedes usar directamente la barra del navegador, aunque es ese caso solo es posible realizar peticiones GET.

13. Bibliografía

13.1. Libros

Martin, R. “Agile Estimating and Planning” (2005) Prentice Hall (ISBN: 9780131479418)

Martin, R. “Clean Code” (2008) Prentice Hall (ISBN 9780132350884)

Monte Galiano, J. “Implantar Scrum con éxito” (2016) UOC (ISBN 9788491164593)

Larman, C. “UML y patrones” (2004) Prentice Hall (ISBN: 978-8420534381)

Arlow, J. “UML 2” (2006) Anaya (ISBN: 978-8441520332)

Sommerville, I. “Ingeniería del software” (2005) Pearson (ISBN: 978-8478290741)

13.2. Webs

<<Argotec Ingeniería>> [online] Available from <https://argotec.es/> [Última consulta: 1/10/2017]

<<Proyecto Qampo >> [online] Available from <https://qampo.es/> [Última consulta, 1/10/2017]

<<SiAR>> [online] Available from <http://eportal.mapama.gob.es/websiar/Inicio.aspx> [Última consulta: 01/07/2018]

<<Wikipedia>> [online] Available from <https://es.wikipedia.org/wiki/> [Última consulta: 28/06/2018]

<<Proyectos Ágiles>> [online] Available from <https://proyectosagiles.org/> [Última consulta: 15/03/2018]

<<Scrum>> [online] Available from <https://www.scrum.org/> [Última consulta: 12/12/2017]

<<Spring >> [online] Available from <https://spring.io/>[Última consulta: 29/06/2018]

<<Docs Springs>> [online] Available from <https://docs.spring.io/> [Última consulta: 28/06/2018]

<<Un poco de java>> [online] Available from <https://unpocodejava.com/> [Última consulta: 28/06/2018]

<<Dzone >> [online] Available from < https://dzone.com/ > [Última consulta: 01/03/2018]

<<Platzi >> [online] Available from <https://platzi.com/> [Última consulta: 22/02/2018]

<<Adictos al trabajo >> [online] Available from <https://www.adictosaltrabajo.com/> [Última consulta: 12/01/2018]

<<El baúl del programador >> [online] Available from <https://elbauldelprogramador.com/> [Última consulta: 06/12/2017]

<<Genbeta: dev>> [online] Available from <https://www.genbetadev.com/> [Última consulta: 13/09/2017]

<<Jet Brains >> [online] Available from <https://www.jetbrains.com/> [Última consulta: 25/06/2018]

<<GitHub>> [online] Available from <https://github.com/> [Última consulta: 25/06/2018]

13.3. Referencias

[1] <<Evapotranspiración del cultivo>> [online] Available from <www.fao.org/3/a-x0490s.pdf> [Última consulta: 25/06/2018]

[2] <<Evapotranspiración del cultivo>> [online] Available from <www.fao.org/3/a-x0490s.pdf> [Última consulta: 25/06/2018]

[3] <<Evapotranspiración del cultivo>> [online] Available from <www.fao.org/3/a-x0490s.pdf> [Última consulta: 25/06/2018]

[4] <<Evapotranspiración del cultivo>> [online] Available from <www.fao.org/3/a-x0490s.pdf> [Última consulta: 25/06/2018]

[5] <<Gestión sostenible de regadíos>> [online] Available from <http://www.mapama.gob.es/es/desarrollo-rural/temas/gestion-sostenible-regadios/sistema-informacion-agroclimatica-regadio/presentacion.aspx> [Última consulta: 17/01/2018]

[6] << Métodos ágiles para la gestión de proyectos>> [online] Available from <http://www.ayselucus.es/noticia/m%C3%A9todos-%C3%A1giles-para-la-gesti%C3%B3n-de-proyectos-i> [Última consulta: 20/2/2018]

[7] <<Agile Methods Promote Speed & Flexibility>> [online] Available from <https://www.contegix.com/agile-methods-promote-speed-flexibility/> [Última consulta: 19/02/2018]

[8] <<Cuellos de botella dentro de los equipos de scrum>> [online] Available from <https://www.itpedia.nl/es/2018/06/22/knelpunten-binnen-scrum-teams/> [Última consulta: 25/06/2018]

[9] <<MVP and MoSCoW: Agility and Results for Your Purchasing Department>> [online] Available from <https://us.liveuniversity.com/2017/06/23/mvp-and-moscow-agility-and-results-for-your-purchasing-department/> [Última consulta: 13/12/2017]

[10] <<Assembleia Geral Ordinária >> [online] Available from <https://www.cdedjoao2.pt/index.php/eventos/item/1> [Última consulta: 12/12/2017]

[11] <<Agile Scrum Master e identificación de controles de calle>> [online] Available from <https://es.popularhowto.com/agile-scrum-master-and-identifying-roadblocks> [Última consulta: 19/01/2018]

[12] <<Why do you need a Kanban board>>[online] Available from <https://medium.com/project-management-learnings/why-do-you-need-a-kanban-board-63db13c685f8> [Última consulta: 25/6/2018]

[13] <<La comunicación, un puente a aprender>> [online] Available from <http://globedia.com/comunicacion-puente-abierto-aprender> [Última consulta: 25/6/2018]

[14] <<Eventos en Scrum II>> [online] Available from <https://www.saraclip.com/eventos-en-scrum-ii/> [Última consulta: 11/01/2018]

[15] <<Porceso y Roles de Scrum>> [online] Available from <https://www.softeng.es/es-es/empresa/metodologias-de-trabajo/metodologia-scrum/proceso-roles-de-scrum.html>[Última consulta: 21/11/2017]

[16] <<Introduction to TDD in iOS >> [online] Available from <https://blog.codecentric.de/en/2015/11/introduction-to-tdd-in-ios/> [Última consulta: 07/02/2018]

[17] <<Boletín Oficial del Estado>> [online] Available from <https://www.boe.es/diario_boe/txt.php?id=BOE-A-2018-3156> [Última consulta: 25/06/2018]

[18] <<Spring Stereotypes y anotaciones>> [online] Available from <https://www.arquitecturajava.com/spring-stereotypes/> [Última consulta: 27/06/2018]