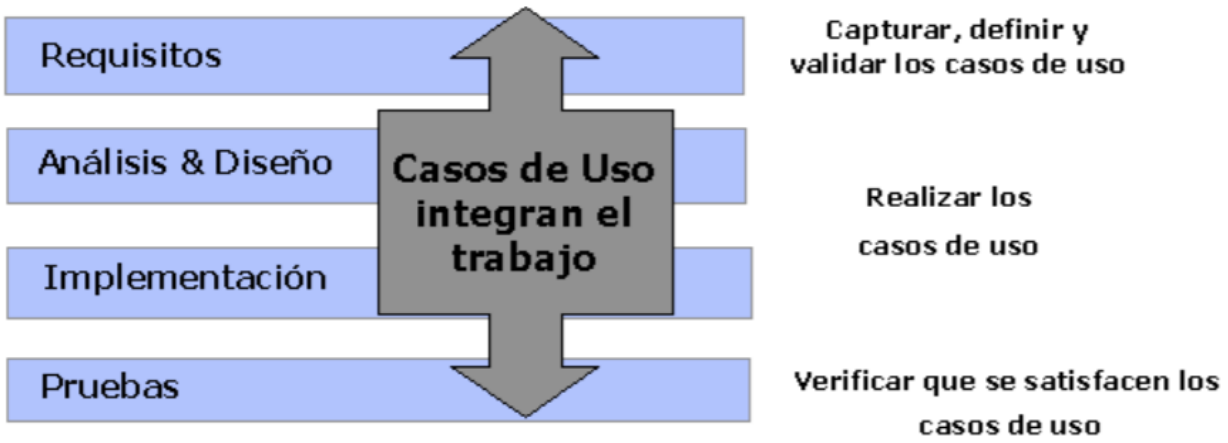
ARQUITECTURA DE AEMET RAICES

1. PRINCIPIOS Y METODOLOGÍA UTILIZADA

AEMET RAICES tiene como principal función el almacenamiento, control de calidad, explotación y difusión de las variables climáticas esenciales en diferentes ámbitos del sistema climático a través de series de datos. Esencialmente, proveerá un punto de acceso común a esas series climáticas con la finalidad de detectar evidencias de los cambios que están teniendo lugar en los diferentes componentes del sistema climático.

Para la determinación de la arquitectura del sistema se ha seguido metodología RUP, por sus siglas en inglés, ó Proceso de Desarrollo Unificado. Se trata de un proceso de desarrollo de software y junto con el Lenguaje Unificado de Modelado UML, constituye la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos.



Fases de la metodología RUP

2. PRINCIPALES PROCESOS DETECTADOS

Se han identificado 48 casos de uso del sistema AEMET RAICES que pueden clasificarse en las siguientes cuatro áreas funcionales:

Área Funcional	Descripción
Harvesting e Ingestión	Recolección de datos y series climáticas de los principales proveedores de series climáticas en España
Control de Calidad	Conjunto de procesos para asegurar la adecuada calidad de los datos que constituyen la serie y que serán comprobados en tiempo real

	durante ingestión o tras la aplicación de procesos de explotación.
Procesamiento	Conjunto de procesos para la explotación de las series. Estos procesos tienen una naturaleza interoperable y podrán ser modificados sin necesidad de grandes conocimientos informáticos
Difusión	Conjunto de procesos y funcionalidades que permiten la difusión de la información almacenada. Son requisitos de esta difusión que provean interfaces web para interfaces humanos e interfaces interoperables para máquinas y otros procesos IT.

3. ARQUITECTURA DEL SISTEMA

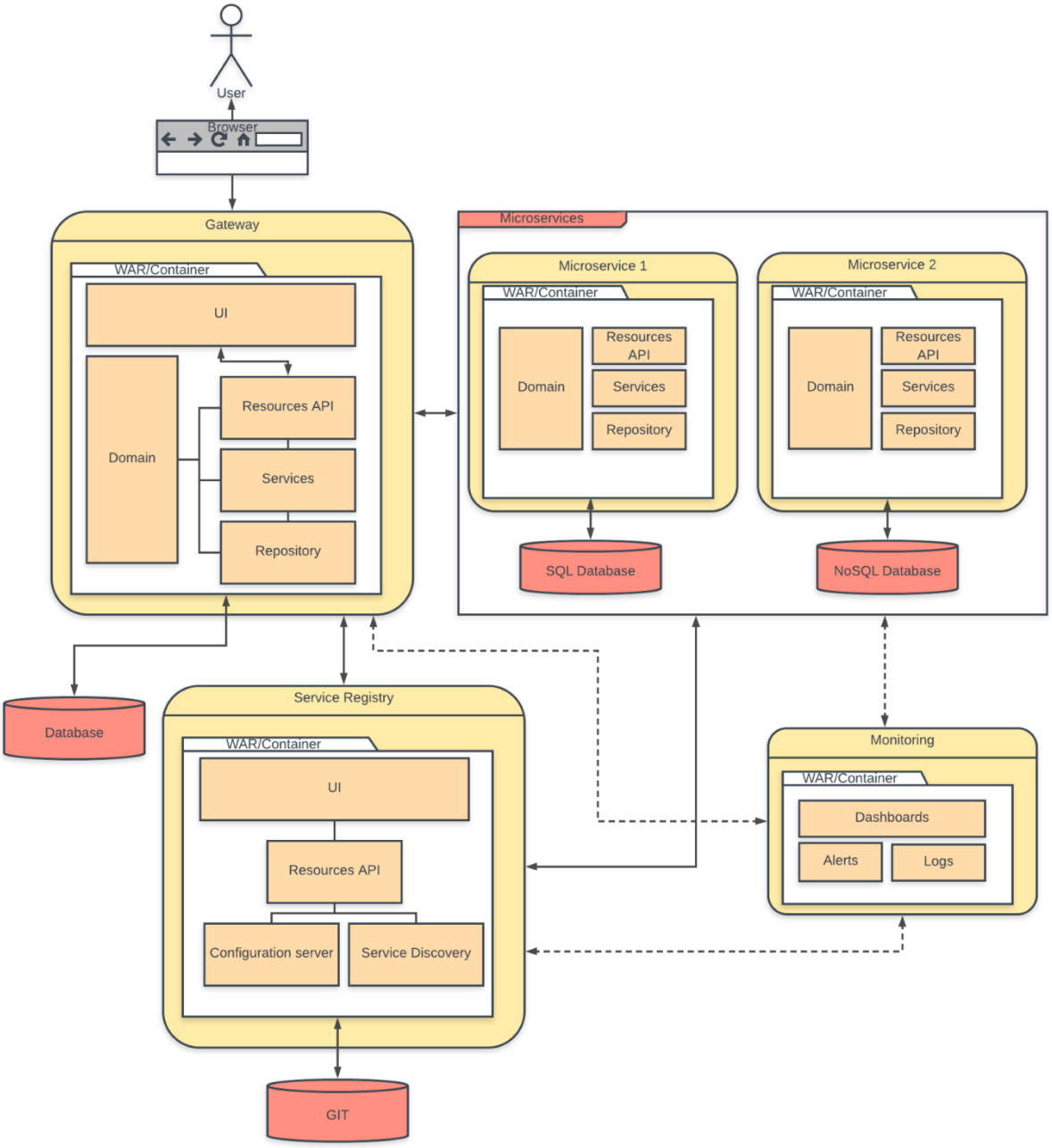
Para la implementación de los casos de uso anteriores se ha decidido emplear la arquitectrua de microservicios (microservices). En particular, microservicios reactivos, esto APIs REST síncronas no bloqueantes y servicios asíncronos basados en eventos. Los microservicios -también conocidos como arquitectura de microservicios- son un estilo arquitectónico que estructura una aplicación como una colección de servicios que son

- Muy mantenibles y comprobables
- Poco acoplados
- Se pueden desplegar de forma independiente
- Organizados en torno a las capacidades del negocio
- Que son propiedad de un pequeño equipo

La arquitectura de microservicios permite la entrega rápida, frecuente y fiable de aplicaciones grandes y complejas. También permite a una organización evolucionar su pila tecnológica. En el caso de AEMET RAICES la pila tecnológica seleccionada es el framework Spring.

Se han identificado los siguientes microservicios:

- Microservicio ***Organismo*** (microservice 1)
- Microservicio **Serie** (microservice 2)



Cada componente de la arquitectura anterior se describe a continuación:

Gateway : En una arquitectura de microservicios impulsada por una pasarela, necesitamos un punto de entrada para acceder a todos los servicios en ejecución. Por lo tanto, necesitamos un servicio que actúe como puerta de enlace o servicio de borde. Este será el proxy o la ruta de las peticiones de los clientes a los respectivos servicios. En AEMET RAICES, se emplea la pasarela *JHipster Gateway* para ello.

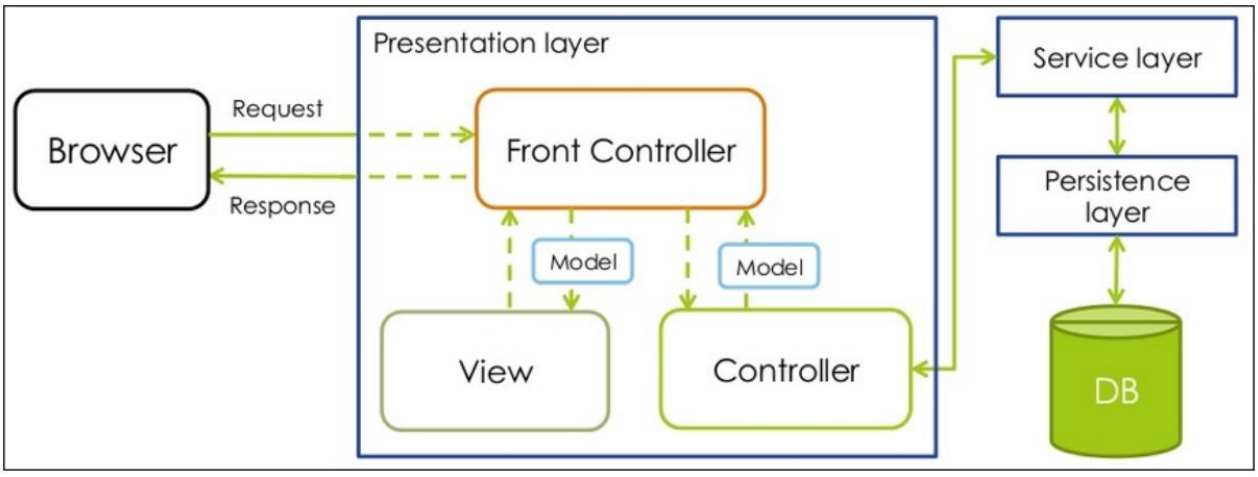
JHipster Gateway está formado, entre otros componentes por, *Netflix Eureka.* Eureka actúa como un equilibrador de carga de nivel medio que ayuda a equilibrar la carga del host de servicios de nivel medio. Proporcionará un simple equilibrio de carga de las peticiones de datos basado en round-robin por defecto. Proveerá también sesiones sticky.

El descubrimiento de los servicios se realizará utilizando *HashiCorp Consul*. Se trata un cliente de descubrimiento de servicios. Consul está enteramente escrito en Golang. Consul también proporciona descubrimiento de servicios, detección de fallos, configuración de múltiples centros de datos y almacenamiento de valores clave.

Registro: Es el componente donde reside la configuración de cada uno de los microservices de organismo y de serie. Para la implementación de estos requerimientos se utilizará S*pring Cloud Config Server* que permite tener un lugar central donde reside la configuración y las propiedades externas de todos los microservices. Estas configuraciones pueden alamcenarse en el sistema de archivos y en un sistema GIT

Esto significa que tendrá una menor huella de memoria. Además, podemos utilizar Consul con servicios escritos en cualquier lenguaje de programación.

Para la implementación de cada microservicio se ha definido una arquitectura lógica que se esquematiza en el siguiente diagrama.



En la arquitectura se distinguen las siguientes capas:

Capa de dominio: Describe los distintos objetos del dominio (entidades), sus atributos, roles y relaciones, además de las restricciones que rigen el dominio del problema.

Capa de persistencia: suele contener objetos de repositorio para acceder a los objetos de dominio. Un objeto repositorio envía consultas a la fuente de datos para obtener los datos, luego

mapea los datos de la fuente de datos a un objeto de dominio, y finalmente persiste los cambios en el objeto de dominio a la fuente de datos. Por lo tanto, un objeto de repositorio es responsable de las operaciones CRUD, acrónimos en inglés de (crear, leer, actualizar y eliminar), en los objetos de dominio.

Capa de servicio: La capa de servicio expone operaciones de negocio que pueden estar compuestas por múltiples operaciones CRUD. Dichas operaciones CRUD suelen ser realizadas por los objetos del repositorio.