Arquitectura Orientada a micro servicios

Consiste en construir una aplicación como un conjunto de pequeños [servicios](https://es.wikipedia.org/wiki/Servicio_(arquitectura_de_sistemas)), los cuales se ejecutan en su propio [proceso](https://es.wikipedia.org/wiki/Proceso_(inform%C3%A1tica)) y se comunican con mecanismos ligeros (normalmente una [API](https://es.wikipedia.org/wiki/Interfaz_de_programaci%C3%B3n_de_aplicaciones) de [recursos HTTP](https://es.wikipedia.org/wiki/Hypertext_Transfer_Protocol)). Cada servicio se encarga de implementar una funcionalidad completa del negocio. Cada servicio es desplegado de forma independiente y puede estar programado en distintos lenguajes y usar diferentes tecnologías de almacenamiento de datos

# Especificaciones de la aplicación

La aplicación hipotética controla las solicitudes mediante la ejecución de lógica de negocios, el acceso a bases de datos y, después, la devolución de respuestas HTML, JSON o XML. Diremos que la aplicación debe admitir una variedad de clientes, incluidos exploradores de escritorio que ejecuten aplicaciones de página única (SPA), aplicaciones web tradicionales, aplicaciones web móviles y aplicaciones móviles nativas. También es posible que la aplicación exponga una API para el consumo de terceros. También debe ser capaz de integrar sus microservicios o aplicaciones externas de forma asincrónica, para que ese enfoque ayude a la resistencia de los microservicios en caso de errores parciales.

# La aplicación constará de estos tipos de componentes:

Componentes de presentación. Son los responsables del control de la interfaz de usuario y el consumo de servicios remotos.

Lógica de dominio o de negocios. Se trata de la lógica de dominio de la aplicación.

Lógica de acceso a bases de datos. Son los componentes de acceso a datos responsables de obtener acceso a las bases de datos (SQL, MySQL, Oracle, DB2, NoSQL).

La aplicación requerirá alta escalabilidad, además de permitir que sus subsistemas verticales se escalen horizontalmente de forma autónoma, porque algunos subsistemas requerirán mayor escalabilidad que otros.

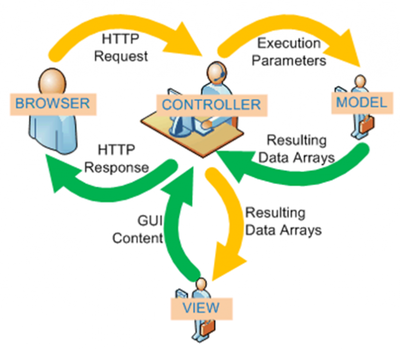
La aplicación debe ser capaz de implementarse en varios entornos de infraestructura (varias nubes públicas y locales) y debe ser multiplataforma, capaz de cambiar con facilidad de Linux a Windows (o viceversa).

Esta arquitectura está basada básicamente en el estilo de arquitectura MVC (Modelo, Vista, Controlador) el cual es un estilo de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos.

El Modelo que contiene una representación de los datos que maneja el sistema, su lógica de negocio, y sus mecanismos de persistencia.

La Vista, o interfaz de usuario, que compone la información que se envía al cliente y los mecanismos interacción con éste.

El Controlador, que actúa como intermediario entre el Modelo y la Vista, gestionando el flujo de información entre ellos y las transformaciones para adaptar los datos a las necesidades de cada uno.



*Ilustración 1 Modelo Vista Controlador*

## Sprint Boot

Para este proyecto utilizaremos una Sprint Boot que es una herramienta que nace con la finalidad de simplificar aún más el desarrollo de aplicaciones basadas en el ya popular framework Spring Core. Spring Boot busca que el desarrollador solo se centre en el desarrollo de la solución, olvidándose por completo de la compleja configuración que actualmente tiene Spring Core para poder funcionar.

Spring Boot centra su éxito en las siguientes características que lo hacen extremadamente fácil de utilizar:

Configuración: Spring Boot cuenta con un complejo módulo que autoconfigura todos los aspectos de nuestra aplicación para poder simplemente ejecutar la aplicación, sin tener que definir absolutamente nada.

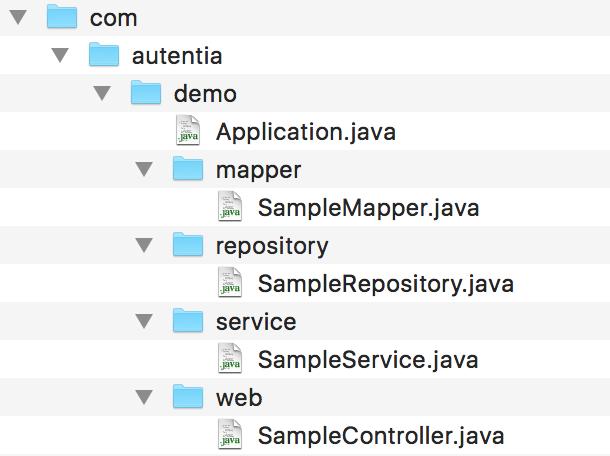
Resolución de dependencias: Con Spring Boot solo hay que determinar que tipo de proyecto estaremos utilizando y el se encarga de resolver todas las librerías/dependencias para que la aplicación funcione.

Despliegue: Spring Boot se puede ejecutar como una aplicación Stand-alone, pero también es posible ejecutar aplicaciones web, ya que es posible desplegar las aplicaciones mediante un servidor web integrado, como es el caso de Tomcat, Jetty o Undertow.

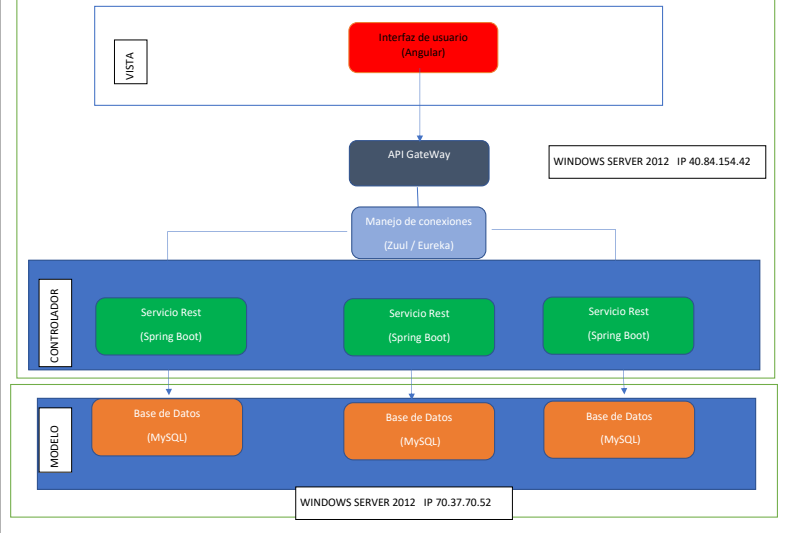
Métricas: Por defecto, Spring Boot cuenta con servicios que permite consultar el estado de salud de la aplicación, permitiendo saber si la aplicación está prendida o apagada, memoria utilizada y disponible, número y detalle de los Bean’s creado por la aplicación, controles para el prendido y apagado, etc.

Extensible: Spring Boot permite la creación de complementos, los cuales ayudan a que la comunidad de Software Libre cree nuevos módulos que faciliten aún más el desarrollo.

Para el proyecto se recomienda instalar y Spring Tools 4 y abrir el proyecto que se almacenara en el gitlab. Este proyecto estará estructurado por varias carpetas las cuales representan diferentes capaz de arquitectura:

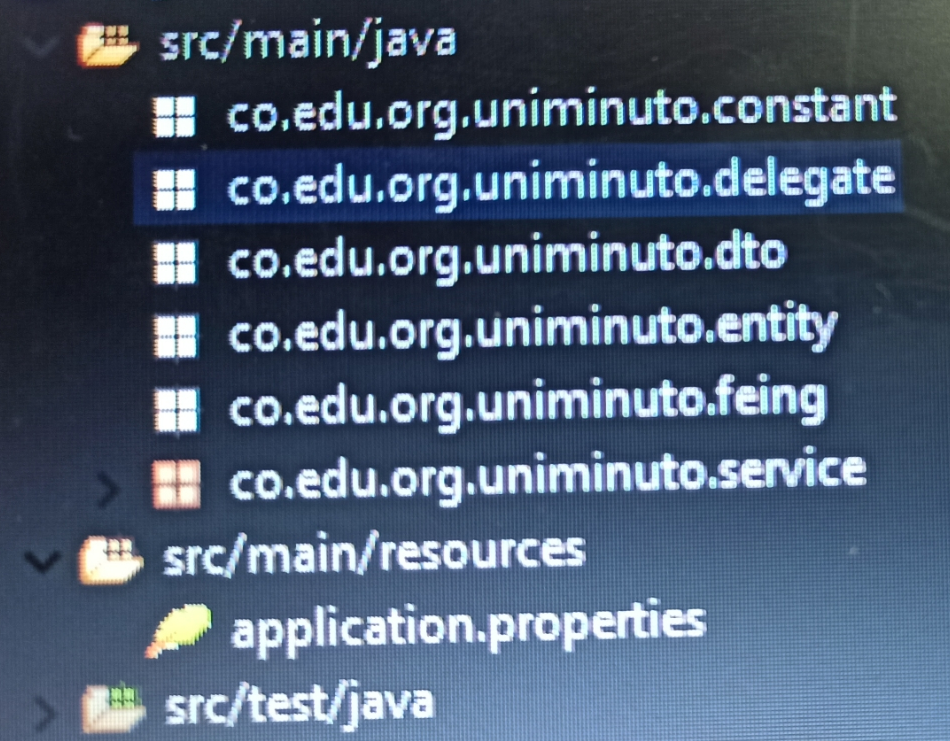


*Ilustración 2- Arquitectura del proyecto*



1. Todos los package deben estar construidos con la raíz de nombre: co.org.edu.uniminuto.nombre del servicio.capa
2. Todas las clases deben ser nombradas en inglés.
3. Los métodos deben ser declarados de acuerdo con la funcionalidad y en inglés.
4. Cada servicio tendrá su propio interfaz de cliente el cual deberá ser estándar con el angular que se entregará en el repositorio de código.
5. Cada micro servicio se deberá configurar con un puerto diferente y para realizar consultas de otras tablas Debra comunicarse con los demás microservicios, en caso de no tener la url del microservicio necesitado se deberá solicitar y7o implementar con datos quemados.

**Arquitectura de Spring**



**Entity**

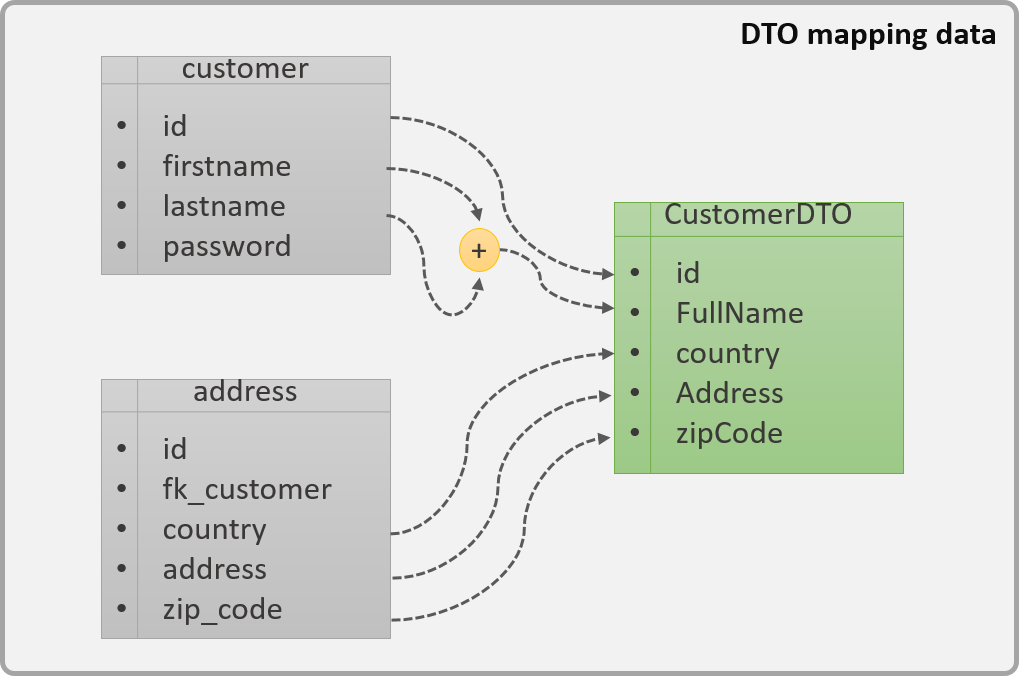
representan datos que pueden persistir en la base de datos. Una entidad representa una tabla almacenada en una base de datos. Cada instancia de una entidad representa una fila en la tabla.

<https://www.baeldung.com/jpa-entities>

**Dto (Objetos de transferencia de datos)**

Una de las problemáticas más comunes cuando desarrollamos aplicaciones, es diseñar la forma en que la información debe viajar desde la capa de servicios a las aplicaciones o capa de presentación, ya que muchas veces por desconocimiento o pereza, utilizamos las clases de entidades para retornar los datos, lo que ocasiona que retornemos más datos de los necesarios o incluso, tengamos que ir en más de una ocasión a la capa de servicios para recuperar los datos requeridos.

El patrón DTO tiene como finalidad de crear un objeto plano (POJO) con una serie de atributos que puedan ser enviados o recuperados del servidor en una sola invocación, de tal forma que un DTO puede contener información de múltiples fuentes o tablas y concentrarlas en una única clase simple.



<https://www.arquitecturajava.com/jpa-dto-data-transfer-object-y-jpql/>

**Delegate**

Los delegate son una construcción útil en sistemas basados ​​en eventos. Esencialmente, los delegate son objetos que codifican un envío de método en un objeto especificado.

Use un delegate para encapsular el acceso a un servicio. El delegate oculta los detalles de implementación del servicio, como los mecanismos de búsqueda y acceso.

<https://www.dineshonjava.com/business-delegate/>

**Feing**

Feign ha sido creado para facilitar la integración entre microservicios mediante la creación de clientes HTTP de forma declarativa. Esto significa que, se simplifica tanto la creación de clientes, que el desarrollador simplemente necesita anotar una interfaz para tenerlo.

<https://blog.bi-geek.com/arquitecturas-basadas-en-microservicios-spring-cloud-feign/>

**Constant**

Una constante es una variable del sistema que mantiene un valor inmutable a lo largo de toda la vida del programa. Las constantes en Java se definen mediante el modificador final.

Las constantes son las cosas constantes en un proyecto. Es decir, es seguro asumir que cualquier proyecto tendría algún tipo de valores constantes utilizados en todo el código.

La práctica común en Java es declararlas como variables públicas, estáticas y finales para que puedan ser referenciadas fácilmente donde sea que las necesiten.

<https://veerasundar.com/blog/2012/04/java-constants-using-class-interface-static-imports/>

<https://howtodoinjava.com/spring-core/spring-declare-beans-from-final-static-field-references-using-util-constant/>

**Service**

Los componentes de servicio son el archivo de clase que contiene la anotación @Service. Estos archivos de clase se utilizan para escribir lógica de negocios en una capa diferente, separada del archivo de clase @RestController.

<https://www.tutorialspoint.com/spring_boot/spring_boot_service_components.htm>

**Tutorial Spring Boot**

<https://www.tutorialspoint.com/spring_boot/index.htm>

**Ejemplo Práctico Spring Boot**

<https://www.ibm.com/developerworks/ssa/library/j-spring-boot-basics-perry/index.html>