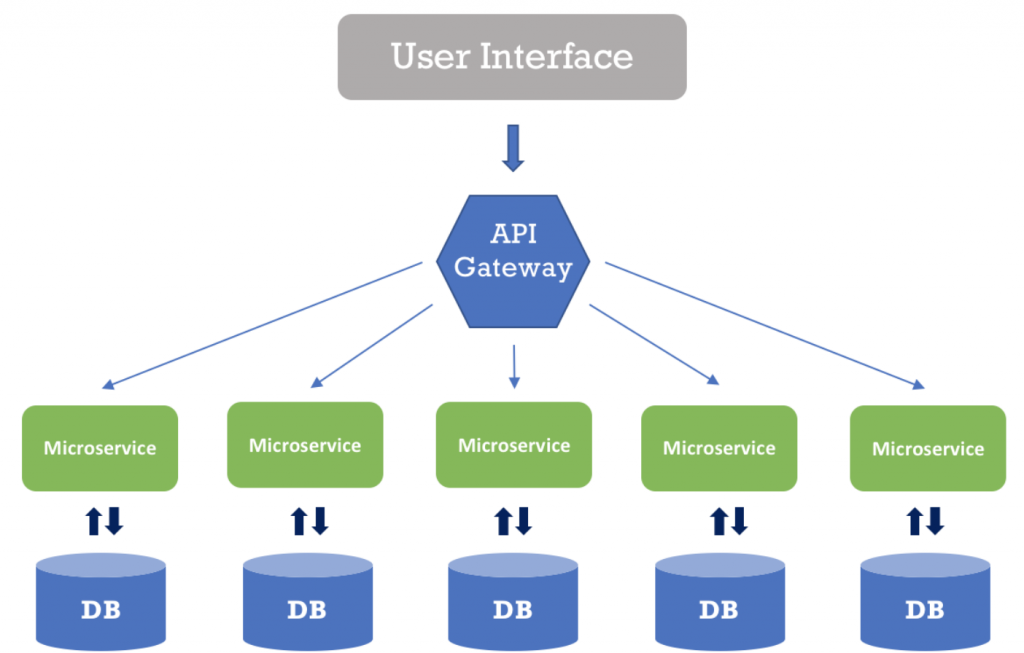
¿Que son los microservicios?

bueno primero que todo los microservicios son elementos independientes que funcionan en conjunto para llevar las mismas tareas y que se comunican atreves de API definidas. Como se puede ver en la siguiente imagen



Este servicio es especialmente adecuado cuando se quiere procurar la compatibilidad con un amplio sector de diferentes plataformas y tiene un mejor aislamiento de fallos, ya que si uno falla ,otro puede seguir funcionando.

Un ejemplo claro de microservicios es Netflix, esta plataforma tiene una arquitectura generalizada que se pasó a los microservicios para el funcionamiento de sus productos. A diario recibe una media de mil millones de llamadas a sus diferentes servicios y es capaz de adaptarse a más de 800 tipos de dispositivos mediante su API de streaming de vídeo, la cual, para ofrecer un servicio más estable, por cada solicitud que le pedimos, ésta realiza cinco solicitudes a diferentes servidores para no perder nunca la continuidad de la transmisión.

un Codigo de ejemplo que se puede utilizar en cualquier IDE Java

Para crear el proyecto, en STS selecciona el menú «*File/New/Spring Starter Project…*» y en la ventana del asistente, ingresa los datos de tu proyecto, por ejemplo en nuestro caso, se verían así:

[Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente](http://sinbugs.com/wp-content/uploads/2017/08/2.sts-crear-proyecto.png)En la siguiente ventana selecciona las librerías **Web** (clases para páginas y servicios web), **JPA** (Hibernate y otras librerías para acceder a la base de datos) y **H2** (base de datos en memoria para facilitar el desarrollo y pruebas).

[Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente](http://sinbugs.com/wp-content/uploads/2017/08/3.librerias.png) Con este último, el log de la consola se verá en colores facilitando su lectura. Si todo va bien, el proyecto debería iniciar sin errores en el log.

**Agrega la capa DTO**

Vamos a crear una clase simple (POJO) para representar a un contacto. Una buena práctica es agrupar las clases en paquetes según su responsabilidad: **dao, dto, service, api**, etc., así que crea estos paquetes (o como los quieras llamar), así:

[Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente](http://sinbugs.com/wp-content/uploads/2017/08/5.ws-rest-paquetes.png)

En el paquete dto o entity o como hayas llamado al que contendrá tus objetos de datos, crea la clase **Contact** y añade campos como un id único, nombre, apellido, teléfono, email y otros que consideres:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | package com.sinbugs.contacts.dto;    public class Contact {   Long id;   String firstName;   String lastName;   String phoneNumber;   String email;  } |

**Creando el API**

Para crear la interfaz REST del servicio web, crea la clase ContacsApi en el paquete Api y agrega un método básico para consultar un contacto por un ID, así:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | package com.sinbugs.contacts.api;    import com.sinbugs.contacts.dto.Contact;    public class ContactsApi {        public Contact getById(){          return new Contact(1L, "John", "Doe", "+57 311 222 3344", "john@sinbugs.com");      }  } |

Por ahora el contacto estará «quemado» *(hard-coded)* en el API pero más adelante cambiaremos esto para que lo consulte de la base de datos.

Ya tenemos nuestra clase Api pero aún le falta para que sea un servicio web: para que Spring reconozca una clase como servicio web y la trate como tal, debemos anotar la clase con **@RestController**. También debemos anotar cada método público que deseemos exponer a los clientes con la anotación **@RequestMapping**, así:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | @RestController  public class ContactsApi {        @RequestMapping(value="/product", method=RequestMethod.GET)      public Contact getById(){          return new Contact(1L, "John", "Doe", "+57 311 222 3344", "john@sinbugs.com");      }  } |

La anotación @RequestMapping recibe parámetros que nos ayudan a cambiar la definición del método hacia el exterior: los content type que recibe o entrega, los headers que requiere, etc. En este caso los básicos son ***value***, para indicar la url en la cual «escuchará» este método y ***method*** que indica el método HTTP que iniciará la ejecución de esta operación. Por defecto es GET, así que en este caso podríamos no dejar sólo el parámetro *value* y el servicio funcionaría igual.

Ahora ejecuta la aplicación y dirige el navegador hacia <http://localhost:8080/product> (si no has cambiado las configuraciones por defecto) y verás algo similar a esto, según tu navegador:

[Interfaz de usuario gráfica, Texto, Aplicación, Word

Descripción generada automáticamente](http://sinbugs.com/wp-content/uploads/2017/08/6.ws-rest-get-one.png)

Y con eso se termina un microservicio básico