**INSTITUTO DE ENSEÑANZA SECUNDARIA CASTELAR**

**FAMILIA PROFESIONAL DE INFORMÁTICA Y COMUNICACIONES**

**TÉCNICO SUPERIOR EN DESARROLLO DE APLICACIONES MULTIPLATAFORMA**

**MEMORIA DEL MÓDULO PROFESIONAL DE PROYECTO**

***ORGANLIST***

**ALUMNO: Mario Fernández Suárez**

**TUTOR: José María Alonso Rodríguez**

**CURSO ACADÉMICO: 2º DAM**

Índice

[Descripción 2](#_Toc86863412)

[Herramientas 2](#_Toc86863413)

[Diagrama E-R 3](#_Toc86863414)

[Esquema BBDD 4](#_Toc86863415)

[Repositorios 4](#_Toc86863416)

[Objetivos 5](#_Toc86863417)

[API 5](#_Toc86863418)

[OpenJDK 6](#_Toc86863419)

[Maven 6](#_Toc86863420)

[Definición y creación del arquetipo 7](#_Toc86863421)

[Bibliografía 13](#_Toc86863422)

# Descripción

Este proyecto, en general, trata sobre una aplicación web reactiva, crearemos una API, documentada con swagger-ui (OpenAPI 3), que gestiona la lista de la compra, pudiendo esta contener productos de distinta índole y dando la posibilidad de convertirla en una lista de cosas que hacer, que se podrán ir tachando.

En dicha lista se dará la opción de realizar la compra ayudándonos de la API de un supermercado, pudiendo guardar las diferentes listas, asociarles un nombre, descripción o pudiendo añadirlas a una lista de favoritos y dando la opción de compartirlas por Whatsapp.

El ejemplo más sencillo es el de un supermercado y sus productos, pero los productos tendrán una propiedad que indique de la categoría a la que pertenecen, pudiéndose usar en varios ámbitos.

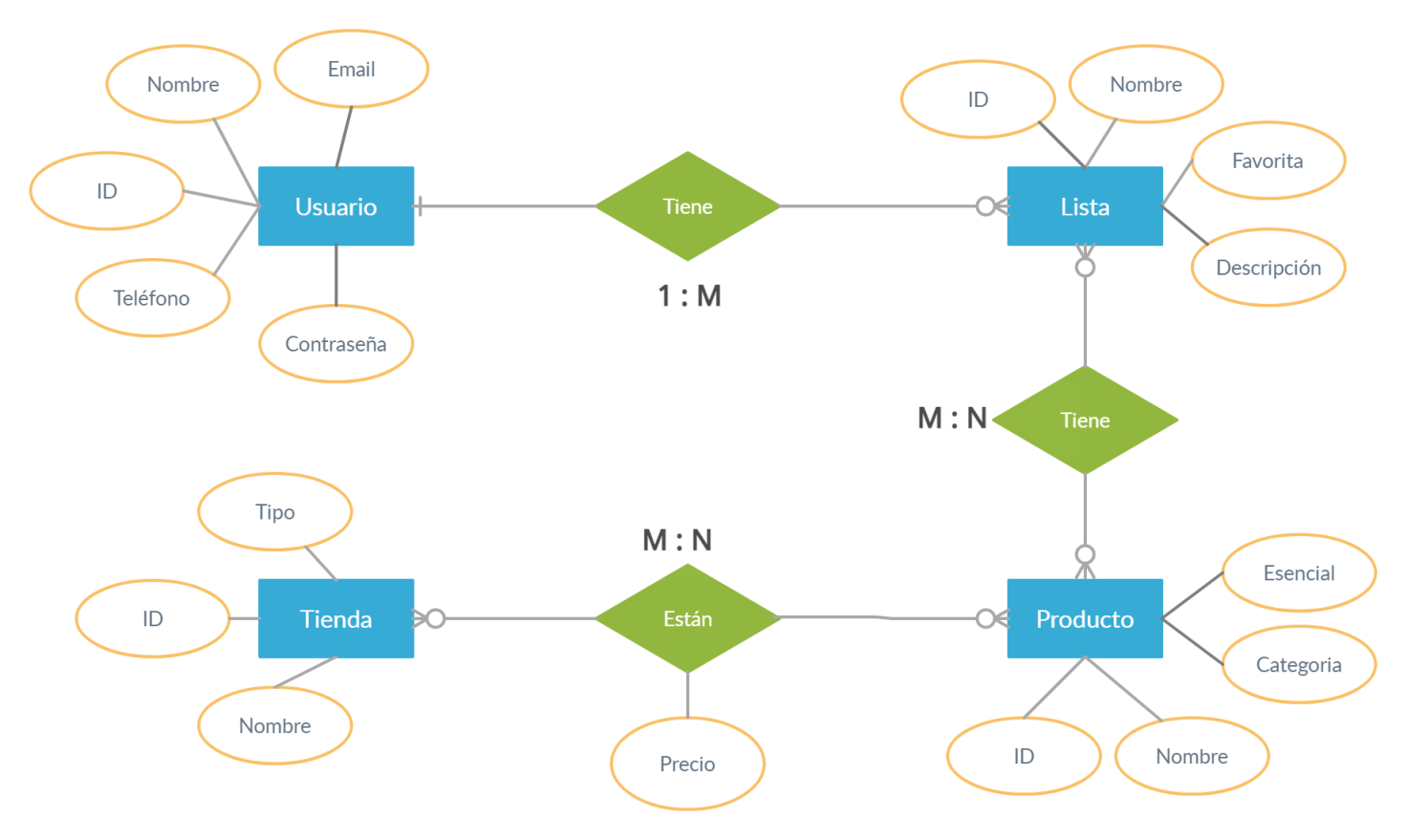
Los productos también tendrán una propiedad que indique sin son productos esenciales en dicha lista, es decir, productos que no puedan faltar.

# Herramientas

* **Visual Studio Code**: es un editor de código fuente, compatible con varios lenguajes de programación, con características y extensiones disponibles que facilitan el desarrollo web.
* **IntelliJ IDEA Ultimate/Community Edition**: es un entorno de desarrollo integrado (IDE), desarrollado por JetBrains, cuya versión gratuita está orientada para desarrollos java, permitiendo desarrollos en kotlin y Android y cuya versión de pago permite el desarrollo con tecnologías de la parte front.
* **Spring Boot**: framework java orientado al desarrollo web, en el cual utilizaremos una programación reactiva mediante la librería WebFlux. Este IDE contiene un servidor Tomcat embebido que nos permitirá arrancar la aplicación. Utilizando la última versión de java libre (OpenJDK 17).
* **MariaDB (MySQL):** es un sistema de base de datos relacional de código abierto.
* **DBeaver:** es un gestor de BBDD multiplataforma.
* **Xampp:** es crear una distribución que trae instalado MariaDB, PHP y Perl, entre otras herramientas. Únicamente voy a utilizarlo para iniciar el servidor MySQL de una manera sencilla y rápida.
* **OpenApi 3**: es una especificación que define un formato de descripción abierto e independiente de los fabricantes para los servicios de API sirve para describir, desarrollar, probar y documentar las APIs de los servicios web RESTful.
* **Angular**: plataforma de desarrollo construida en TypeScript, basado en componentes para crear aplicaciones web escalables. Lo utilizaremos en su última versión hasta el momento, la v12.
* **Postman**: software para crear peticiones HTTP y poder probar las APIs, pudiendo definir colecciones de las mismas y diferentes entornos con la definición de sus propias variables, entre otras cosas.
* **Maven:** es una herramienta de construcción y organización de proyectos.Lo utilizaremos en su última versión hasta el momento, la v3.8.
* **Git/GitHub**: GitHub es una web para alojar proyectos utilizando el sistema de control de versiones Git.

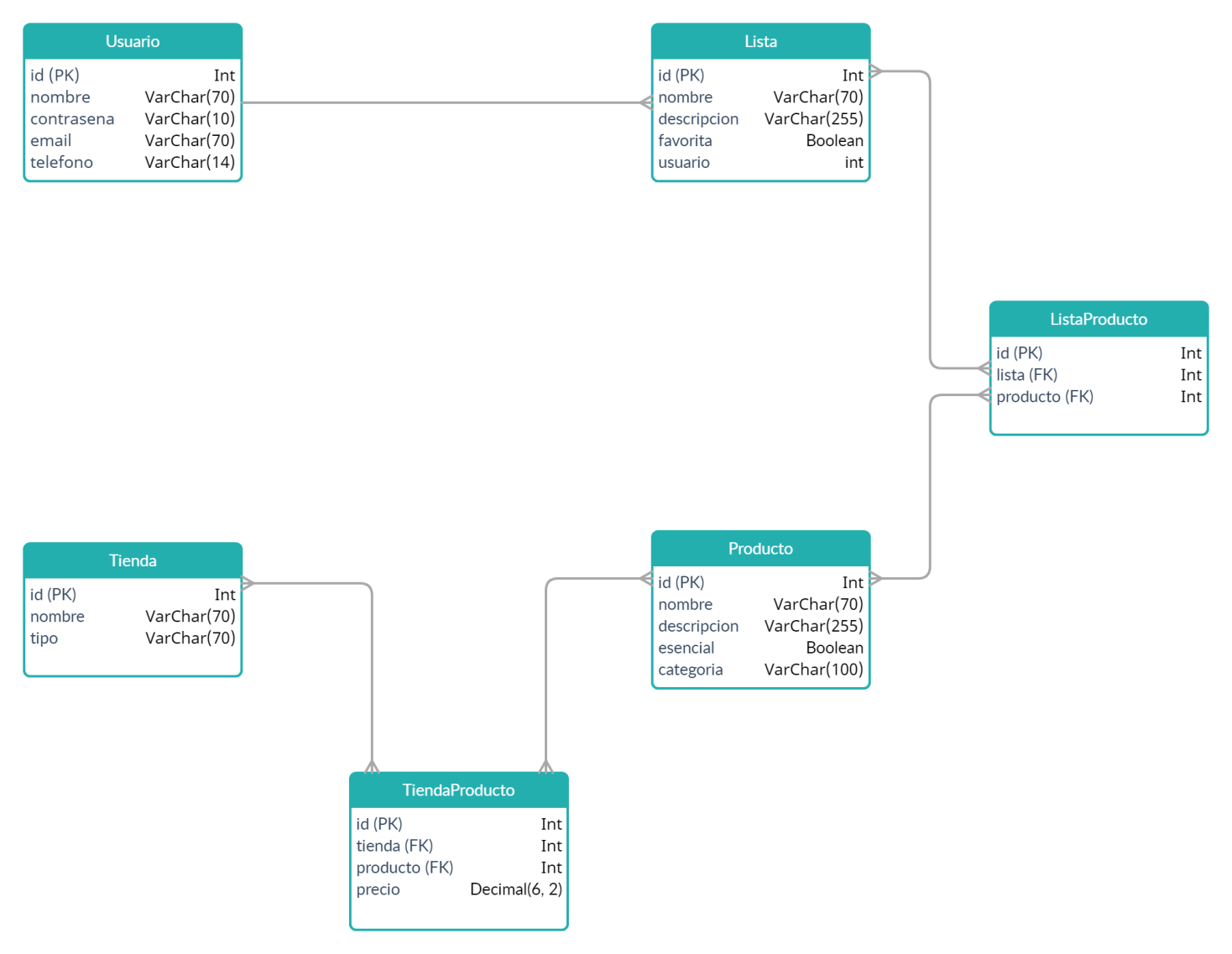
# Diagrama E-R

Este diagrama lo he digitalizado con ayuda de la aplicación web de Creately.



# Esquema BBDD

Este es el esquema de la BBDD, también digitalizado con ayuda de la aplicación web de Creately.



# Repositorios

Las versiones/desarrollos de esta aplicación se subirán a los siguientes repositorios públicos de [GitHub](https://github.com/MarioFS94):

* [Parte BackEnd](https://github.com/MarioFS94/OrganList-Back)
* [Parte FrontEnd](https://github.com/MarioFS94/OrganList-Front)

# Objetivos

Los objetivos que tengo con este proyecto es hacer una aplicación incluyendo:

* La creación de un arquetipo Maven customizado, a través de comandos.
* La generación del proyecto a partir del arquetipo Maven creado.
* Implementar el flujo de desarrollo back, con librerías reactivas, como WebFlux.
* Implementar el flujo de desarrollo front, con la librería Akita que es una herramienta de gestión de estado para aplicaciones Angular. Sigue modelos como Flux y Redux.

# API

Una API (Application Programming Interfaces) es un conjunto de definiciones y protocolos que se utiliza para desarrollar e integrar el software de las aplicaciones.



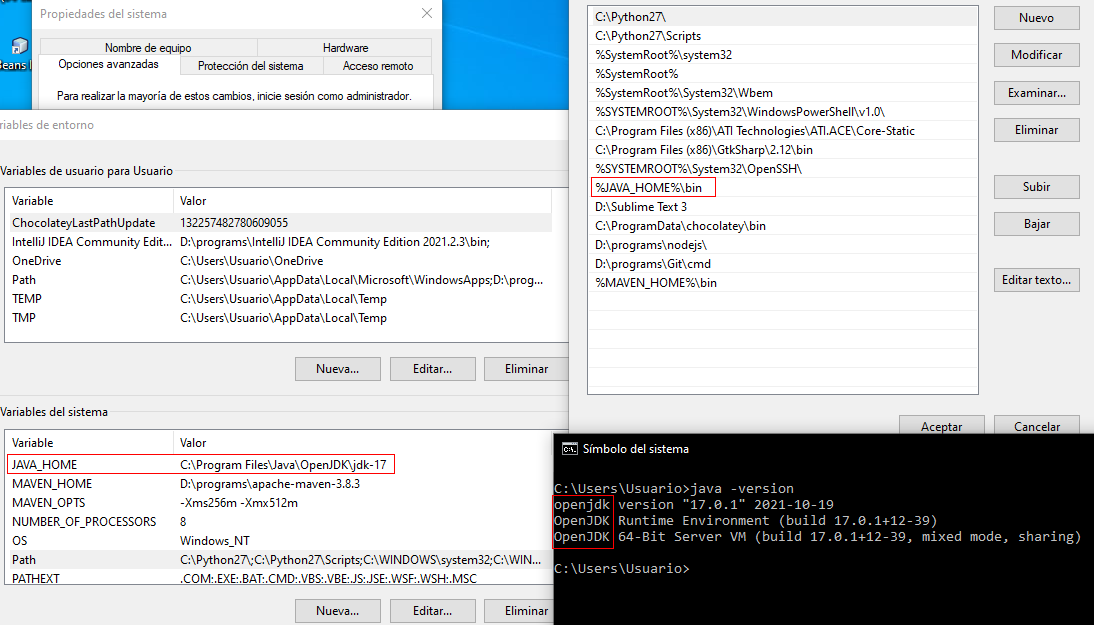
Las API permiten que sus productos y servicios se comuniquen con otros, sin necesidad de saber cómo ni en que lenguaje están implementados. Esto simplifica el desarrollo de las aplicaciones y permite ahorrar tiempo y dinero. Las API le otorgan flexibilidad; simplifican el diseño, la administración y el uso de las aplicaciones, y proporcionan oportunidades de innovación, lo cual es ideal al momento de diseñar herramientas y productos nuevos (o de gestionar los actuales).

A la hora de acceder al swagger-ui y la documentación de la API, pondremos en el navegador <http://localhost:8080/swagger-ui.html> y no redireccionará a la url correcta de la documentación.

# OpenJDK

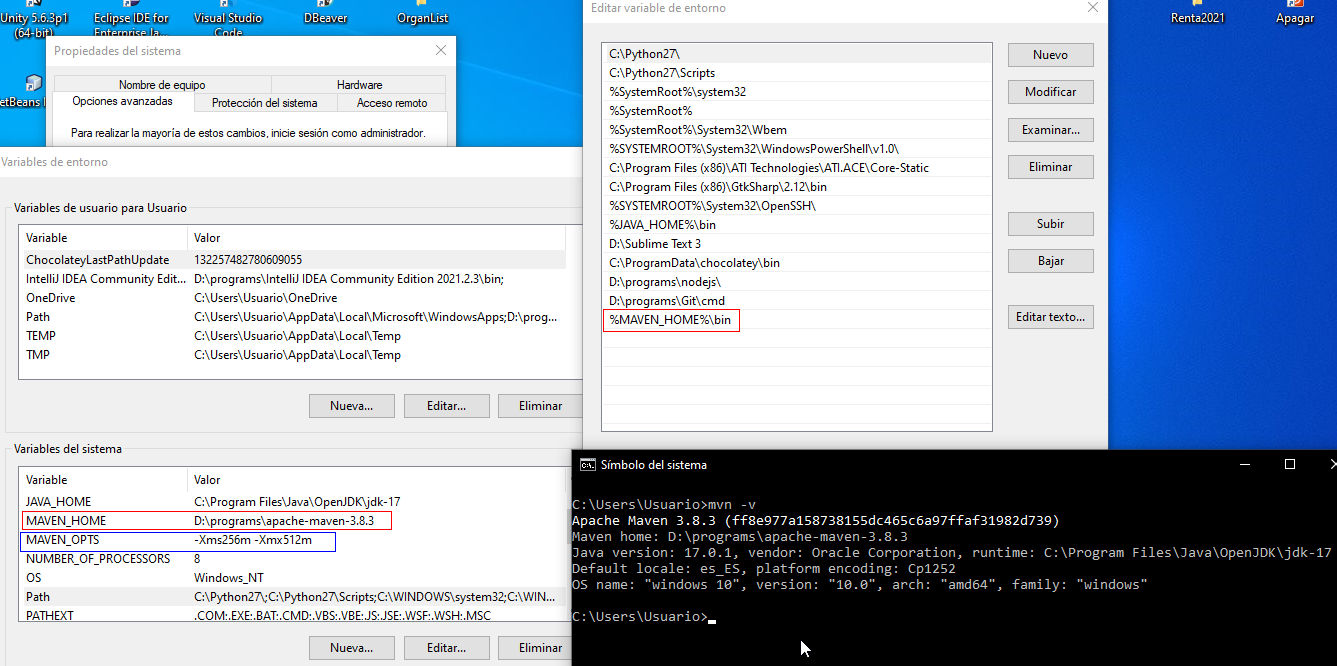
Java a partir de su versión 11, se ha hecho de pago, para aquellas aplicaciones que se quieran subir a producción con dicho jdk. Sin embargo, Oracle también ha sacado una versión open source, OpenJDK, para poder usar las versiones superiores necesidad de pagar.

Para su configuración he modificado y borrado variables del sistema, quedando así:



# Maven

Maven es una herramienta de construcción y organización de proyectos, que tiene el objetivo de simplificar los procesos de build (compilar y generar ejecutables a partir del código fuente).



La variable de **MAVEN\_OPTS** no es necesaria configurarla, indica que capacidad tendrá en la memoria disponible para proyectos java.

Al igual veremos que se puede crear el archivo **settings.xml** en la carpeta **C:\Users\[nombre\_usuario]\.m2**, para configuraciones más detalladas, pero en principio no es necesario, ya veremos en la sección de definición y creación de un arquetipo que nos obliga para poder crear nuestro arquetipo customizado.

# Definición y creación del arquetipo

En esencia, un arquetipo de Maven es un conjunto de herramientas para crear plantillas para proyectos. Un arquetipo se define como una estructura o modelo original a partir del cual se hacen todas las demás cosas del mismo tipo. Los arquetipos ayudarán a los autores a crear plantillas de proyectos Maven, proporcionando los medios para generar versiones parametrizadas de esas plantillas de proyectos.

El plugin maven-archetype-plugin utiliza, de forma interna, el motor de plantillas Velocity para la generación y adaptación de los ficheros de nuestra plantilla de proyecto.

El uso de arquetipos proporciona a los desarrolladores una manera rápida y consistente, con las mejores prácticas, para crear un proyecto con unas determinadas clases y organización de paquetería, a partir de una serie de ficheros predeterminados.

También permite que el mecanismo de arquetipos sea aditivo, y con ello nos referimos a permitir que partes de un proyecto sean guardadas para un determinado arquetipo, de modo que piezas, componentes o aspectos de un proyecto puedan ser añadidos a proyectos ya existentes.

Si ejecutamos el comando:

**mvn archetype:generate**

Con este comando descargariamos todos los arquetipos configurados en el repositorio, para descargar uno en concreto ejecutaremos:

**mvn archetype:generate -Dfilter=[archetype name].**

Los arquetipos predefinidos por Maven son los siguientes:

|  |  |
| --- | --- |
| **Archetype ArtifactIds** | **Description** |
| maven-archetype-archetype | An archetype to generate a sample archetype project. |
| maven-archetype-j2ee-simple | An archetype to generate a simplifed sample J2EE application. |
| maven-archetype-mojo | An archetype to generate a sample a sample Maven plugin. |
| maven-archetype-plugin | An archetype to generate a sample Maven plugin. |
| maven-archetype-plugin-site | An archetype to generate a sample Maven plugin site. |
| maven-archetype-portlet | An archetype to generate a sample JSR-268 Portlet. |
| maven-archetype-quickstart | An archetype to generate a sample Maven project. |
| maven-archetype-simple | An archetype to generate a simple Maven project. |
| maven-archetype-site | An archetype to generate a sample Maven site which demonstrates some of the supported document types like APT, XDoc, and FML and demonstrates how to i18n your site. |
| maven-archetype-site-simple | An archetype to generate a sample Maven site. |
| maven-archetype-webapp | An archetype to generate a sample Maven Webapp project. |

En la ejecución nos hará varias preguntas, con sus respectivas sugerencias, para poder identificar nuestro proyecto y rellenar nuestro **pom.xml**. Nos preguntará por el **groupId**, que es el grupo mediante el cual identificaremos nuestro proyecto, se suele definir como la estructura de un dominio o si ya existe nuestra página, con el nombre de nuestro dominio invertido, por ejemplo:

Imaginemos nuestro dominio, organlist.es -> es.organlist, sería nuestro groupId.

También nos preguntará por el **artifactId**, que sería el nombre de nuestro proyecto.

Hay 4 formas de crear un proyecto mediante un arquetipo, por línea de comandos:

1. Utilizar un arquetipo ya predefinido por Maven.
2. Crear un arquetipo desde un proyecto ya desarrollado.
3. Utilizar un arquetipo definido por la comunidad de usuarios de Maven.
4. Crear los archivos necesarios, configurar y generar nuestro propio arquetipo desde 0.

Podríamos usar un arquetipo predefinido (forma 1) con **mvn archetype:generate -Dfilter=maven-archetype-quickstart** o uno ya creado por un usuario de la comunidad (forma 3), ejecutando el comando: **mvn archetype:generate -DarchetypeGroupId=org.springframework.boot -DarchetypeArtifactId=spring-boot-sample-simple-archetype -DarchetypeVersion=1.0.2.RELEASE.** La forma más elaborada y con la que se ahorraría menos tiempo es la forma 4, ya que hay que definir desde 0 los archivos necesarios nombrados mas adelante cuando creemos un arquetipo customizado desde un proyecto existente.

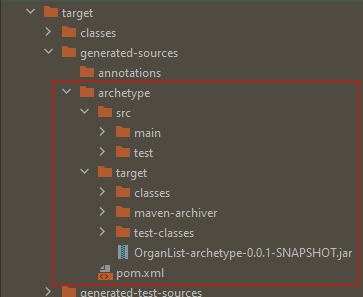
Como no hace falta “reinventar la rueda” si no es necesario, voy a crear un arquetipo desde un proyecto ya creado (forma 2), a partir de la versión ultímate de IntelliJ IDEA que utiliza la herramienta Spring initializr, que también está en la web, creamos un proyecto y comenzamos a configurar nuestra estructura propia de proyecto ([scaffolding](http://www.tugurium.com/gti/termino.php?Tr=scaffolding)), cuando ya esté creado nuestro proyecto customizado, lanzamos el siguiente comando **mvn archetype:create-from-project**.

Para ello, este comando nos obliga a crear el fichero **settings.xml** en la carpeta **.m2**, para configurar Maven, cuyo contenido únicamente será:

***<settings*** *xmlns=*[*http://maven.apache.org/SETTINGS/1.0.0*](http://maven.apache.org/SETTINGS/1.0.0) *xmlns:xsi=*[*http://www.w3.org/2001/XMLSchema-instance*](http://www.w3.org/2001/XMLSchema-instance) *xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0 https://maven.apache.org/xsd/settings-1.0.0.xsd"****>***

***</settings>***

Una vez ejecutado el comando, se nos creara en target/generated-sources el directorio archetype que contiene la configuración del mismo.



Si buceamos en el código veremos que las importaciones serán generalizadas por ${package} o ${project}.

Ej: package ${package}.controller;

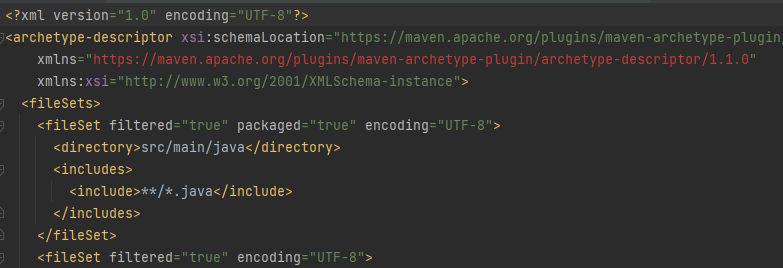
ó

import ${package}.utils.DefaultDocumentation;

Para el arquetipo, los archivos **archetype/…/archetype-resources/pom.xml** y **archetype/…/META-INF/maven/archetype-metadata.xml** son muy importantes.

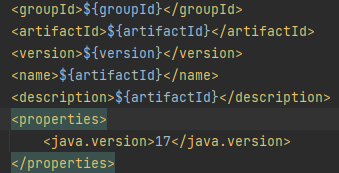
Definen los archivos que se incluirán y se necesitarán para construir el proyecto:

* El **archetype-metadata.xml** contiene los directorios que se incluirán.



La url de archetype descriptor está en rojo por que el IDE no es capaz de reconocerla, no es un error.

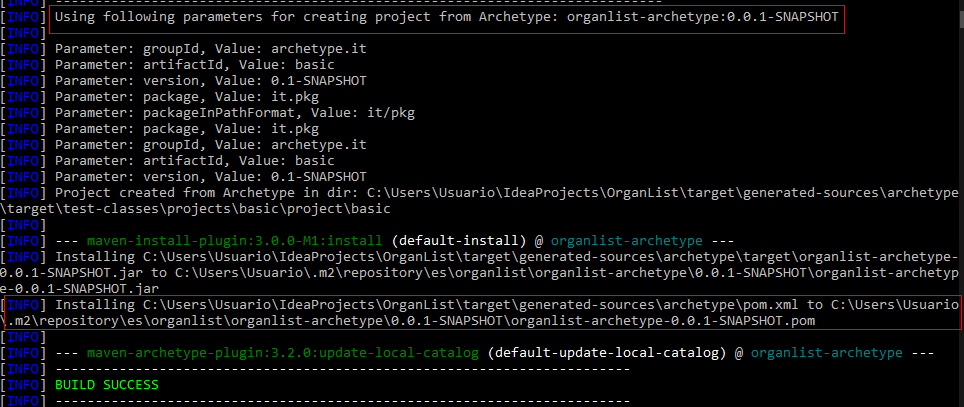
* En el **pom.xml** decidiremos las dependencias que vamos a poner.



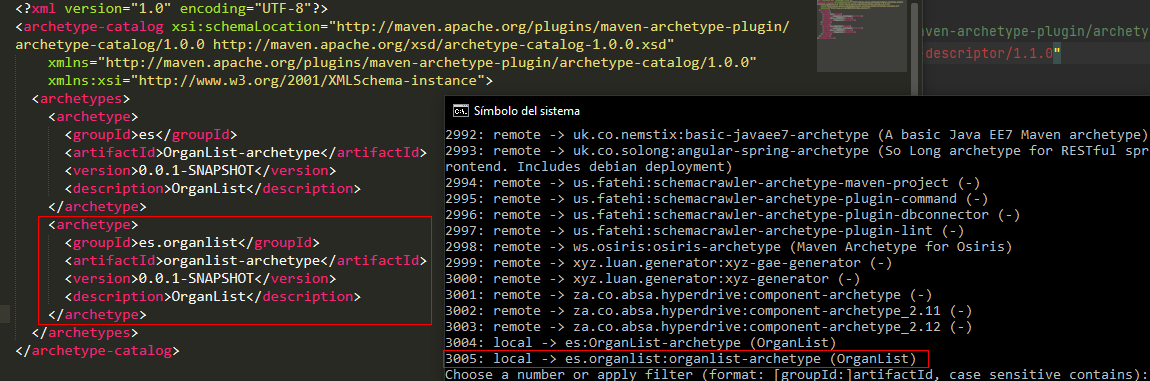
Este pom.xml cogerá los valores de groupId, artifactId, versión, name y description como argumentos informados en la ejecución del comando.

Hay que borrar las carpetas ocultas de IntelliJ y Maven (.idea y .mvn), ya que no tienen que pertenecer al arquetipo.

Para terminar la creación de nuestro arquetipo customizado, nos situaremos en la carpeta raíz del arquetipo OrganList/target/generated-sources/archetype y lanzaremos el comando **mvn clean install**, esto agregará el arquetipo al repositorio maven local, para que pueda ser usado:



Para verificar que está en nuestro repositorio local, podemos ejecutar el comando **mvn** **archetype:generate -DarchetypeCalalog=local,** al lanzar el comando vemos como se ha añadido correctamente a la lista de arquetipos, ya que se muestra la línea ‘3005: local -> es.organlist:organlist-archetype (OrganList)’ indicando que este arquetipo lo tenemos en el repositorio local. También podemos situarnos en .m2/repository/archetype-catalog.xml y ver que se ha añadido el nuevo arquetipo con su groupId, artifactId y versión.



El arquetipo con el groupId 'es' da problemas, ya que al ejecutar el comando de generación del arquetipo, las variables, usadas por el motor de plantillas Velocity (antes nombrado), generadas en los distintos ficheros java del target, sustituye ${groupId} por todos los 'es' que encuentre entre cualquier palabra, por ello es recomendable poner un groupId con la estructura como si fuese un dominio (es.organlist) y no una simple palabra.

Ahora podremos crear proyectos a través de un arquetipo con una plantilla customizada con el comando:

**mvn archetype:generate**

**-DarchetypeGroupId**=es.organlist

**-DarchetypeArtifactId**=organlist-archetype

**-DarchetypeVersion**=0.0.1-SNAPSHOT

**-DgroupId**=<sampleID>

**-DartifactId**=<sampeArtifactId>

**-Dversion**=<sampleVersion>

En nuestro caso, los 3 últimos argumentos se sustituirán por:

**-DgroupId**=es.organlist

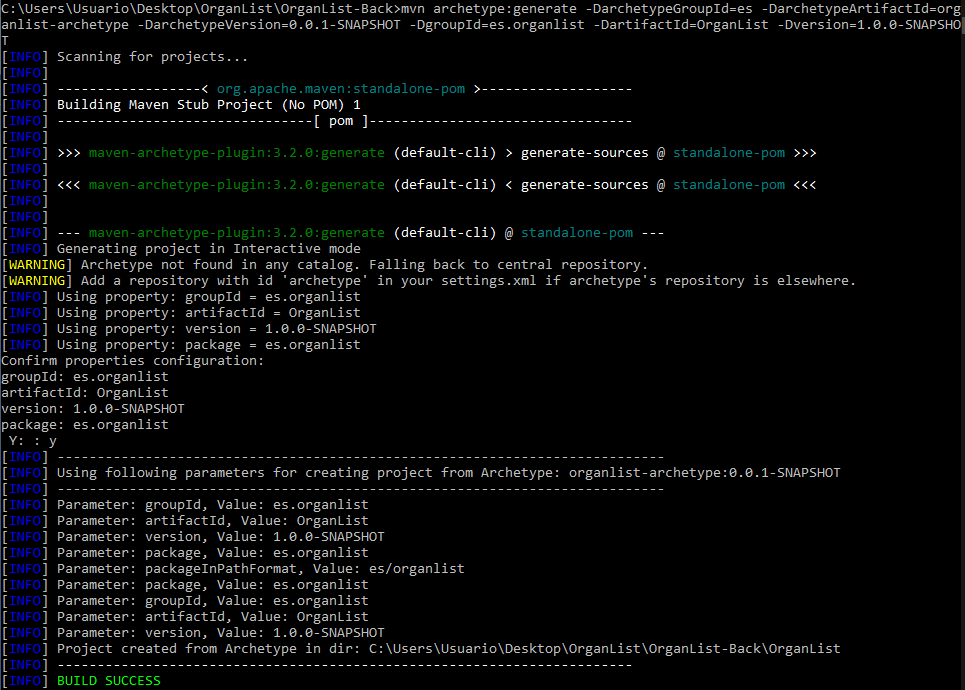
**-DartifactId**=OrganList

**-Dversion**=1.0.0-SNAPSHOT

Quedando el comando como:

**mvn archetype:generate -DarchetypeGroupId=es.organlist -DarchetypeArtifactId=organlist-archetype -DarchetypeVersion=0.0.1-SNAPSHOT -DgroupId=es.organlist -DartifactId=OrganList -Dversion=1.0.0-SNAPSHOT**

Dicho comando lo ejecutaremos en nuestro directorio con el repositorio git oficial del proyecto de D.A.M., para probar su correcto funcionamiento:



Una vez se nos se nos haya descargado nuestro arquetipo y creado el proyecto, si quisiéramos ejecutar el proyecto desde el **jar**, primero nos situaremos dentro del proyecto generado, con el comando **cd organlist** y en dicha ruta ya podemos empaquetarlo con ayuda de Maven, con el comando **mvn package** que generará la carpeta target que contiene los ejecutable y para ejecutar nuestro jar como hasta este punto el compilador no sabe donde se encuentra la clase main se lo indicamos y ejecutamos el comando con estructura **java –cp [nombre\_proyecto]-[version].jar jar.[Main.class].**

Nada más descargar el proyecto o clonar del repositorio git, haremos un **mvn clean install** para limpiar el proyecto y compilarlo.

# Bibliografía

* <https://app.creately.com/>
* <https://enmilocalfunciona.io/por-que-utilizar-arquetipos-maven/>
* <https://maven.apache.org/guides/introduction/introduction-to-archetypes.html>
* <https://maven.apache.org/guides/mini/guide-creating-archetypes.html>
* <https://www.adictosaltrabajo.com/2008/06/09/creararquetiposmaven/>
* <https://inthiraj1994.medium.com/creating-your-own-archetype-840136fc7818>
* <https://www.vojtechruzicka.com/maven-archetypes/>
* <https://maven.apache.org/settings.html>
* <https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/que-es-openapi/>
* <https://springdoc.org/>
* <https://www.arquitecturajava.com/que-es-spring-webflux/>