**Guía de uso Arquetipo (elemental-archetype)**

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Change reason | Owner |
| 1.0 | 14/03/2017 | Versión inicial | Optimissa |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

ÍNDICE

[1. Objetivos y alcance 3](#_Toc477274063)

[2. Descripción del proyecto 4](#_Toc477274064)

[3. Descripción de elemental-archetype 6](#_Toc477274065)

[4. Descripción de elemental-archetype-core 8](#_Toc477274066)

[5. Descripción de elemental-archetype-data 9](#_Toc477274067)

[6. Descripción de elemental-archetype-messaging 11](#_Toc477274068)

[7. Descripción de elemental-archetype-webapp 13](#_Toc477274069)

[8. Enlaces de interés 14](#_Toc477274070)

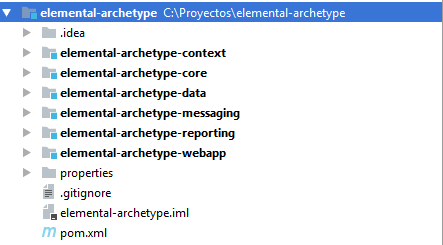
# Objetivos y alcance

El objetivo de este documento es describir el arquetipo que se ha desarrollado con el fin de que las nuevas aplicaciones que se desarrollen partan de una base inicial común, de manera que cada una de ellas crezca según las necesidades de la misma.

# Descripción del proyecto

Este proyecto usa **Spring Boot**, de modo que se generará un WAR ejecutable tanto en servidores Tomcat como JBOSS, usando como gestor de dependencias y constructor del proyecto la herramienta **Maven**.

Está compuesto por varios módulos de Maven, tal y como puede verse en la siguiente imagen:



A continuación se describen *grosso modo* los módulos que lo componen:

* **Elemental-archetype**: Módulo padre del resto de módulos. En su fichero pom.xml se recogen los números de versión que usarán en todo el proyecto así como dependencias comunes en todos los módulos maven.
* **Elemental-archetype-core**: Este módulo **es necesario** en cualquier caso. Entre sus funciones se encuentran arrancar la aplicación, definición de controladores (en este caso, de tipo REST) y servicio de envío de correos.
* **Elemental-archetype-data**: Este módulo, usando **spring-data-jpa**, se encarga de realizar la conexión a base de datos, mapear las tablas en entidades y ofrecer DAOs para realizar consultas. En caso de que el proyecto/aplicación/nodo en cuestión no necesite acceso a BB.DD., se puede prescindir de este módulo.
* **Elemental-archetype-context**: Mediante el uso de este módulo se establece la comunicación con **redis**, de modo que, además de la conexión misma, se proveen mecanismos de lectura y guardado en redis. Como en el caso anterior, si no es necesario el uso de redis, se puede eliminar este módulo.
* **Elemental-archetype-messaging**: Con este módulo se gestionan la conexión al bróker de mensajería RabbitMQ así como el envío y/o recepción de mensajes haciendo uso del mismo. También es susceptible de ser desechado.
* **Elemental-archetype-reporting.** Ofrece ejemplos de cómo crear un PDF con una plantilla de **JasperReports** y cómo generar un Excel con **Apache POI**. De la misma manera que los módulos anteriores, este módulo puede ser descartado en caso de no ser necesaria la generación de reportes.
* **Elemental-archetype-webapp**: Hace referencia al resto de módulos ‘hermanos’, de modo que **encapsula la aplicación en** sí **en un WAR** que será **ejecutable**. Nótese que este módulo **sí es necesario** para la correcta construcción del proyecto, en todo caso.

Nótese bienque **al eliminar un módulo, han de eliminarse las referencias al mismo**. A saber, referencia al módulo en el pom.xml del padre, elemental-archetype, y referencia en el pom.xml de elemental-archetype-webapp, así como eliminación de las posibles referencias en código desde otros módulos a clases, constantes o beans del mismo.

Además, al arrancar el proyecto **debe encontrarse establecida la variable de sistema ELEMENTAL\_ARCHETYPE\_HOME** que apunta a los ficheros .properties de configuración, por ejemplo, pasando al servidor el parámetro jvm -DELEMENTAL\_ARCHETYPE\_HOME=Path-a-los-ficheros-de-configuracion.

# Descripción de elemental-archetype

El punto más importante de este módulo es su fichero pom.xml:

*<?***xml version="1.0" encoding="UTF-8"***?>*<**project xmlns="http://maven.apache.org/POM/4.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"**>  
 <**modelVersion**>4.0.0</**modelVersion**>  
  
 <**groupId**>com.optimissa</**groupId**>  
 <**artifactId**>elemental-archetype</**artifactId**>  
 <**packaging**>pom</**packaging**>  
 <**version**>1.0.0-SNAPSHOT</**version**>  
  
 *<!-- Child modules -->* <**modules**>  
 <**module**>elemental-archetype-core</**module**>  
 <**module**>elemental-archetype-context</**module**>  
 <**module**>elemental-archetype-messaging</**module**>  
 <**module**>elemental-archetype-data</**module**>  
 <**module**>elemental-archetype-webapp</**module**>  
 <**module**>elemental-archetype-reporting</**module**>  
 </**modules**>  
  
 *<!-- Name of the project and versions used whithin the whole project -->* <**properties**>  
 <**project.name**>elemental-archetype</**project.name**>  
 <**maven.compiler.source**>1.7</**maven.compiler.source**>  
 <**maven.compiler.target**>1.7</**maven.compiler.target**>  
 <**version.jboss.bom.eap**>6.4.0.GA</**version.jboss.bom.eap**>  
 <**spring.platform.bom**>Athens-SR4</**spring.platform.bom**>  
 ...

</**properties**>  
  
 *<!--Manage the versions of the dependencies for this module and all its children modules -->* <**dependencyManagement**>  
 <**dependencies**>  
 *<!-- Dependencies of jboss. If you don't have to use tomcat or don't want the dependencies to be dependent of the jboss version, remove this dependency. -->* <**dependency**>  
 <**groupId**>org.jboss.bom.eap</**groupId**>  
 <**artifactId**>jboss-javaee-6.0-with-hibernate</**artifactId**>  
 <**version**>${version.jboss.bom.eap}</**version**>  
 <**type**>pom</**type**>  
 <**scope**>import</**scope**>  
 </**dependency**>  
 *...*

</**dependencies**>  
 </**dependencyManagement**>  
  
 *<!-- Common dependencies for the whole project. -->* <**dependencies**>  
 <**dependency**>  
 <**groupId**>javax</**groupId**>  
 <**artifactId**>javaee-api</**artifactId**>  
 <**version**>7.0</**version**>  
 <**scope**>provided</**scope**>  
 </**dependency**>  
 ...   
 </**dependencies**>  
  
</**project**>

A continuación se describen los puntos clave de este fichero pom.xml:

* Se define el *packaging* o modo de empaquetado como“pom”, ya que este módulo se dedica esencialmente sólo a encapsular otros módulos.
* Bajo la etiqueta <modules> se definen todos los módulos hijos de éste.
* Bajo la etiqueta <properties> se definen el nombre del war a generar (project.name) así como las versiones de todas las dependencias a lo largo de todo el proyecto.
* En el apartado *dependencyManagement* se centralizan ciertas dependencias que heredan los módulos hijos. Por poner un ejemplo, en el caso de que en el apartado dependencyManagement se defina que para la *dependency* A se use la versión 1.0.0.RELEASE y que exista un hijo que tenga la dependencia A con *<version>* 1.1.0.RELEASE, la versión que finalmente se compilará será la 1.0.0.RELEASE, ya que es la que se definió en el dependencyManagement del módulo padre. Asimismo, si se ha definido una dependencia dentro del dependencyManagement, no es necesario indicar la versión para los módulos hijos.

Nótese además que en nuestro caso usamos dos BOM (Bill Of Materials), dependencias de tipo pom, permitiéndonos mantener varias dependencias relacionadas entre sí (por ejemplo, de spring) de manera coherente.

* Bajo <dependencies> se definen todas las dependencias que necesita cada uno de los módulos para funcionar correctamente.

# Descripción de elemental-archetype-core

En el pom.xml de este módulo, como en el de todos los hijos, se define básicamente cuál es el módulo padre, su tipo de empaquetado (en este caso, es un jar) y qué dependencias necesitará para funcionar correctamente:

*<?***xml version="1.0" encoding="UTF-8"***?>*<**project xmlns="http://maven.apache.org/POM/4.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"**>  
 <**parent**>  
 <**artifactId**>elemental-archetype</**artifactId**>  
 <**groupId**>com.optimissa</**groupId**>  
 <**version**>1.0.0-SNAPSHOT</**version**>  
 </**parent**>  
 <**modelVersion**>4.0.0</**modelVersion**>  
  
 <**artifactId**>elemental-archetype-core</**artifactId**>  
 <**packaging**>jar</**packaging**>  
  
 <**dependencies**>  
 <**dependency**>  
 <**groupId**>com.optimissa</**groupId**>  
 <**artifactId**>elemental-archetype-messaging</**artifactId**>  
 <**version**>${project.version}</**version**>  
 </**dependency**>  
 *<!-- Dependency needed to send emails -->* <**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-mail</**artifactId**>  
 </**dependency**>  
 </**dependencies**>  
  
</**project**>

Dentro del mismo, se encuentra la clase **RootConfig**, punto de entrada al arrancar la aplicación.

Se encuentra anotada con **@SpringBootApplication(scanBasePackages = {"com.optimissa"})** y **@EnableWebMvc**, de modo que se indica así qué paquetes han de escanearse en búsqueda de clases anotadas con @Component y se activa el WebMvc, cuya configuración se establece en la clase WebConfig.

Podemos encontrar además tres controladores: uno para generar y descargar reportes, otro para realizar un envío de mensajes a través de RabbitMQ y otro más para el envío de correos, con sus correspondientes servicios o interfaces de los mismos (que serán implementadas por el módulo que correspondan en este último caso).

# Descripción de elemental-archetype-data

La configuración de este módulo se encuentra descrita en el fichero ${ELEMENTAL\_ARCHETYPE\_HOME}/database.properties, en el que han de describirse:

* Los parámetros de conexión en base de datos, que se tomarán en el caso de que se encuentre activo el perfil de spring “dev”, con el parámetro

-Dspring.profiles.active=dev

* El nombre del JNDI, que se tomará cuando el perfil de spring sea nulo o distinto de dev.
* El nombre de la persistenceUnit, por defecto elementalArchetypeUnit.
* En la propiedad **persistence.packages.to.scan**, el paquete bajo el que se **encontrarán las @Entity a usar**. **Por defecto**, se establece a "**com.optimissa.elementarchetype.model**". **En caso de** que se desee almacenar estas @Entity en **otro paquete**, **ha de cambiarse** esta propiedad al paquete correspondiente.

En este módulo se encuentran definidas dos clases de configuración: JpaConfiguration, que se limita a leer los parámetros del fichero arriba mencionado, y **JpaLoader**, que crea la conexión con la base de datos (teniendo en cuenta el perfil de spring, generará un datasource u otro).

En este módulo se encuentran además dos paquetes sin ningún contenido:

El paquete **com.optimissa.elementarchetype.model**, que contendrá los componentes @Entity que mapearán las tablas de base de datos en clases java. A modo de ejemplo, se presenta la siquiente:

@Entity  
@Table(name = **"QP\_RFI\_PERFIL\_EMISION"**)  
**public class** PerfilEmisionModel {  
  
 @Id  
 @Column(name = **"ISIN"**, nullable = **false**)  
 **private** String **isin**;  
  
 @Column(name = **"PRODUCTO"**)  
 **private** String **product**;el   
  
 **public** String getIsin() {  
 **return isin**;  
 }  
  
 **public void** setIsin(String isin) {  
 **this**.**isin** = isin;  
 }  
  
 **public** String getProduct() {  
 **return product**;  
 }  
  
 **public void** setProduct(String product) {  
 **this**.**product** = product;  
 }  
}

Como puede verse, se mapea la tabla QP\_RFI\_PERFIL\_EMISION a la clase java PerfilEmisionModel, teniendo en cuenta la columna ISIN como Identificador y la columna PRODUCTO. Nótese que en este ejemplo no se define ninguna estrategia de creación de @Id, ya que el modelo o entidad es únicamente obtenido en la base de datos, no se realiza guardado o modificación del mismo (en el caso de que se realizase, sería necesario establecer esta política de generación de Id, usando una secuencia, por ejemplo).

En el paquete **com.optimissa.elementarchetype.dao**, apuntado desde la anotación **@EnableJpaRepositories de la clase JpaLoader (si es otro paquete, cambiar el valor de la anotación),** se almacenarán los componentes con la anotación @Repository.

Un ejemplo de estas clases es el siguiente:

@Repository  
**public interface** UserDao **extends** JpaRepository<UserModel, Long> {  
  
 */\*\* Query that returns the user with its workspaces  
 \** ***@param username*** *\** ***@return*** *\*/* @Query(**"select distinct u from UserModel u left join fetch u.workspaces as w left join fetch w.types left join fetch w.profiles left join fetch w.classifications left join fetch w.currencies left join fetch w.countries left join fetch w.sectors left join fetch w.seniors left join fetch w.ratings left join fetch w.moodies left join fetch w.fitchs left join fetch w.sps where u.username =:username "**)  
 List<UserModel> getUserAndWorkspacesByUsername(@Param(value = **"username"**)String username);  
  
}

Se recomienda que, como en el ejemplo anterior, los DAOs sean interfaces que extiendan JpaRepository<{Modelo}, {Tipo de Id del modelo}>, ya que así dispondremos de métodos CRUD por defecto.

Además, podrán definirse en esta interfaz métodos que, anotados con @Query, devuelvan el resultado sin necesidad de ninguna implementación.

Nótese además que, para consultas sencillas, al extender la interfaz JpaRepository, se pueden hacer métodos que serán “leídos sintácticamente”. Es decir, el método anterior, en el caso de que no se hubiera necesitado hacer left join, podría haber quedado tal que:

List<UserModel> getUserByUsername(String username);

Así, se obtendrían todos los usuarios que coincidiesen con el username indicado.

# Descripción de elemental-archetype-messaging

La configuración de conexión de este módulo se encuentra en el fichero **messaging.properties,** donde se establecen todos **los parámetros necesarios para conectarse** a una instancia de un **Broker** de mensajería **RabbitMQ**.

El fichero messaging.properties anteriormente mencionado es leído por la clase de configuración MessagingConfig.

La clase **MessagingLoader** se encarga de **crear la conexión con RabbitMQ**, tomando como parámetros aquellos que leyó la clase MessagingConfig. Además, se encarga de recoger los datos contenidos en otro fichero de configuración: **connections.properties**.

Este fichero connection.properties **contiene los datos necesarios para establecer una conexión punto a punto en RabbitMQ**, y deberá ser el mismo para todos los nodos que realicen la comunicación. Los parámetros son:

* **xxx.exchangeName** = Nombre del exchange que se usará para enviar un mensaje o crear una cola.
* **xxx.routingKey** = Routing key que se usará para enviar un mensaje o crear una cola.
* **xxx.receiverMethodName** = Método que actuará de listener, al que llegarán los mensajes recibidos.
* **xxx.receiverClassName** = Nombre del Bean o Clase donde se encuentra el método anteriormente mencionado.

Donde xxx indica el nombre de la conexión.

Se ha de destacar que **los dos primeros parámetros serán usados por el emisor**, que simplemente **necesita** para enviar un mensaje el **nombre del exchange y la routing key del mismo**. Por otra parte**, el receptor necesita conocer los 4 parámetros**: Para recibir un mensaje, **necesita crear un listener en un método de una clase determinada, que escuche los mensajes de un exchange con una clave determinada** (si ésta es necesaria, ya que el exchange podría ser de tipo fanout).

A modo de ejemplo se incluye un servicio **HelloSendMessageService** para enviar mensajes, que incluye dos métodos:

* **sendHelloMessage()**, que envía una String "HOLAHOLA", sin esperar a recibir respuesta.
* **sendAndReceiveHelloMessage()**, que envía una String “HOLAHOLA” y recibe otra String como respuesta.

Estos dos métodos son llamados a través de un **controlador** de elemental-archetype-core, haciendo una petición GET al endpoint **{servidor}:{puerto}/elemental-archetype/sendMessage/hello**.

En este punto se utiliza la clase **MessageHandler<G,T>**, del optimissa-platform. En este caso se ha instanciado a G= String, T = String, **siendo G el tipo de parámetro que se envía y T el tipo de parámetro que se espera recibir**.

Por otra parte, tenemos también **dos servicios que se encargan de recibir mensajes**. El primero de ellos es **MultipleReceiversService**. Este servicio en su método **afterPropertiesSet()**, que se ejecuta inmediatamente **después de la construcción del componente**, crea **dos** **listeners**, uno para cada método, **usando el servicio ConnectionService** que ha inyectado.

El primer método es **receiveHello(String message)**, que **recibe un mensaje** y no espera devolver nada, por lo que tiene de tipo de retorno void.

El segundo es **receiveAndReplyHello(String message)**, muy similar al anterior pero con la diferencia de que **devuelve una String, que será enviada al emisor de vuelta**.

El otro servicio que se dedica a escuchar mensajes es **MessageListenerReceiverService**, que **implementa la interfaz MessageListener**.

Así, esperará que llegue un mensaje (Message de AMPQ) a su método onMessage. Nótese que en el método afterPropertiesSet() se inicializa el consumidor tomando la instancia misma:

**public void** afterPropertiesSet() **throws** Exception {  
 *// In this case we need to generate the queues, because the container supposes that the queue previously exists* String queueToListen = **connectionService**.createPublishQueueAndExchange(ConnectionConstants.***HELLO\_CONNECTION***, ExchangeTypes.***DIRECT***);  
 *//Create the container to listen, and start it.* SimpleMessageListenerContainer container = **new** SimpleMessageListenerContainer();  
 container.setConnectionFactory(**rabbitConnectionFactory**);  
 container.setQueueNames(queueToListen);  
 container.setMessageListener(**new** MessageListenerAdapter(**this**));  
 container.start();  
}

El punto en donde se indica que la instancia actual será la que escuche los mensajes, usando un **MessageListenerAdapter** es, en concreto:

container.setMessageListener(**new** MessageListenerAdapter(**this**));

Además cabe indicarse que la siguiente línea es necesaria en este caso porque no existe una cola de rabbitMQ que escuchar que haya sido creada previamente, pero en el caso de que estemos seguro de que existiese, podríamos prescindir de ella:

String queueToListen = **connectionService**.createPublishQueueAndExchange(ConnectionConstants.***HELLO\_CONNECTION***

# Descripción de elemental-archetype-webapp

Este módulo **simplemente empaqueta los jars resultantes de la compilación de los módulos anteriores en un fichero .war**, que tendrá el **nombre** descrito en la propiedad **project.name** que se encuentra bajo la etiqueta **<properties> del pom.xml** del módulo padre **elemental-archetype**.

Además, bajo el directorio WEB-INF, se encuentran tres ficheros:

* **web.xml**, que contiene el atributo <display-name>, nombre de la aplicación que se mostrará.
* **Jboss-web.xml**, que se usará para servidores jboss, en el que se ha de definir el contexto de la aplicación en el atributo <context-root>
* **jboss-deployment-structure.xml**, en el que se definen dependencias del servidor jboss que no han de usarse/sobreescribirse. En este caso el fichero es el siguiente:

<**jboss-deployment-structure xmlns="urn:jboss:deployment-structure:1.2"**>  
 <**deployment**>  
 <**exclude-subsystems**>  
 <**subsystem name="jpa"**/>  
 </**exclude-subsystems**>  
 <**exclusions**>  
 <**module name="org.hibernate"**/>  
 <**module name="org.apache.log4j"**/>  
 <**module name="org.apache.commons.logging"**/>  
 <**module name="org.slf4j"** />  
 <**module name="org.slf4j.impl"** />  
 </**exclusions**>  
 </**deployment**>  
</**jboss-deployment-structure**>

Indicándose que usaremos nuestro propio jpa, hibernate, y log4j.

# Enlaces de interés

A continuación se presentan enlaces relacionados con las tecnologías usadas para desarrollar/ampliar este arquetipo:

* **Protocolo AMQP**, que implementa RabbitMQ. **Se recomienda leer el artículo completo**, para tener nociones de lo que es una Queue, un exchange, qué tipos de colas y qué tipos de exchanges existen…:

<https://www.rabbitmq.com/tutorials/amqp-concepts.html>

* **RabbitMQ** (no centrarse demasiado, ya que usamos Spring AMQP)**:**
  + **Documentación completa:**

<https://www.rabbitmq.com/documentation.html>

* + **Tutoriales (seleccionar java):**

[**https://www.rabbitmq.com/getstarted.html**](https://www.rabbitmq.com/getstarted.html)

* **Spring AMQP:**

<http://docs.spring.io/spring-amqp/docs/1.7.1.RELEASE/reference/html/>

* **Spring-data-jpa**,usado por spring-boot-starter-data-jpa. Especial atención al apartado “5.3. Query methods”:

<https://docs.spring.io/spring-data/jpa/docs/current/reference/html/>

* **Spring boot:**

<https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/>