# **Portada**

Número de grupo: E3.08

Url al repositorio: <https://github.com/antsermen/Acme-Toolkits-E3.08.git>

Miembros (+ emails):

* José Ramón Arias Expósito ([jrae1112jrae@gmail.com](mailto:jrae1112jrae@gmail.com)),
* Manuel Carnero Vergel ([mancarver1@alum.us.es](mailto:mancarver1@alum.us.es)),
* Juan Carlos Moreno Pérez ([morenoperezjuancarlos@gmail.com](mailto:morenoperezjuancarlos@gmail.com)),
* Pablo Santos Pérez ([pablo.santos.232001@gmail.com](mailto:pablo.santos.232001@gmail.com)),
* David Sabugueiro Troya ([dsabugueiro14@gmail.com](mailto:dsabugueiro14@gmail.com)),
* Antonio Roberto Serrano Mena ([robertoserranomena@gmail.com](mailto:robertoserranomena@gmail.com))

Fecha: 31/05/2022

# **Tabla de contenidos**

Contenido

[**1.** **Portada** 1](#_Toc104915444)

[**2.** **Tabla de contenidos** 2](#_Toc104915445)

[**3.** **Resumen ejecutivo** 3](#_Toc104915446)

[**4.** **Tabla de revisión** 3](#_Toc104915447)

[**5.** **Introducción** 4](#_Toc104915448)

[**6.** **Contenidos** 5](#_Toc104915449)

[**7.** **Conclusiones** 11](#_Toc104915450)

[**8.** **Bibliografía** 11](#_Toc104915451)

# **Resumen ejecutivo**

En este documento podemos encontrarnos con 8 apartados: una portada, que contiene el número de grupo, la url al repositorio en GitHub, los miembros (junto a sus emails) y la fecha de creación del documento; una tabla de contenidos, que sería un índice típicamente conocido para un documento; un resumen ejecutivo, en el que se resumen los contenidos de cada apartado del documento; una tabla de revisión, formada por los datos de las revisiones realizadas sobre el documento (número de revisión, fecha y descripción); una introducción, que resume aquellos conocimientos que tenemos respecto a la arquitectura de un WIS; los contenidos del documento, donde se recogen los conceptos que hemos considerado relevantes a la hora de hablar de la arquitectura de un WIS; unas conclusiones, extraídas a partir de la finalización del documento y una bibliografía, que contiene las fuentes que nos sirvieron de apoyo para el documento (en caso de no encontrarse ninguna, aparecerá “Intencionalmente en blanco.”).

# **Tabla de revisión**

|  |  |  |
| --- | --- | --- |
| Número de revisión | Fecha | Descripción |
| 1 | 26/05/2022 | Revisión de los contenidos iniciales del documento. |
| 2 | 30/05/2022 | Revisión del documento completo para su puesta a punto para ser entregado. |
| 3 | 31/05/2022 | Preparación para entrega |

# **Introducción**

Un WIS (Web-Based Information System) es un sistema de información que usa tecnologías web de Internet para proporcionar información y servicios a usuarios o a otros sistemas o aplicaciones.

Antes de comenzar la asignatura Diseño y Prueba II, no habíamos profundizado sobre la arquitectura de un WIS. Nuestra experiencia más reciente con algo similar ha sido profundizar en diferentes arquitecturas por capas y distintos patrones de diseño durante la asignatura Diseño y Pruebas I, por lo que empezamos algo familiarizados con el concepto.

A fecha de finalización de este documento, hemos aprendido bastante más al respecto de la estructura WIS, su funcionamiento, y los distintos elementos que la componen.

# **Contenidos**

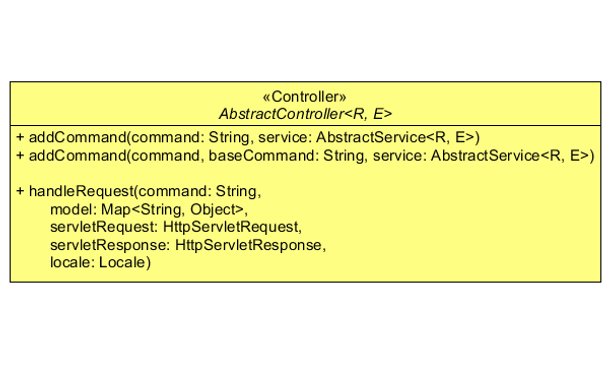
La arquitectura se divide en las siguientes capas:

* **Capa del Navegador**: La capa del navegador se utiliza para realizar solicitudes HTTP a la capa de la aplicación y para representar los resultados que devuelve.
* **Capa de** **Aplicación**: recibe solicitudes HTTP y las pasa a la aplicación adecuada según la URL.
* **Capa de la** **Base de Datos**: recibe solicitudes SQL y devuelve conjuntos de registros (seleccionar consultas) o recuentos de filas afectadas (insertar, actualizar o eliminar consultas).

En cuanto a la arquitectura de una aplicación podemos encontrar varios elementos: controladores, servicios, entidades y vistas.

* Un **controlador** atiende todas las solicitudes relacionadas con una función en particular. Los controladores de nuestro proyecto derivan de “AbstractController” y presentan 2 parámetros, indicando primero el rol relacionado con los comandos, y siendo el segundo el objeto sobre el que se aplicarán dichos comandos. La implementación se basa en los llamados flujos de trabajo, cada uno de los cuales se especializa en comandos particulares, donde encontramos “list” y “show” para representar datos, “create” y “delete” para añadir o eliminar elementos de la base de datos; y “update” y “published” (derivado de update) para modificar una entrada de la base de datos. A la hora de añadir estos comandos al controlador hemos usado el método “addCommand”, indicando el nombre del comando, el comando básico del que deriva en caso de ser necesario, y la clase servicio concreta que gestionará los métodos a los que el comando llamará.

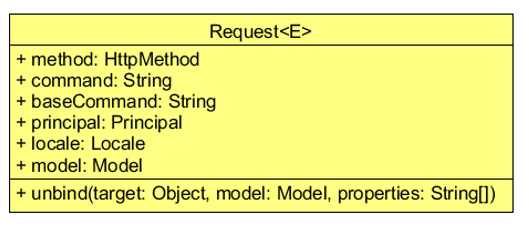
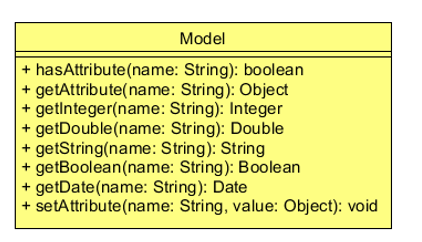
También pueden presentar un método HandleRequest, pero no ha sido necesario en nuestro proyecto. Aquí una descripción visual de un “AbstractController”, de lo que extendían nuestros controladores:

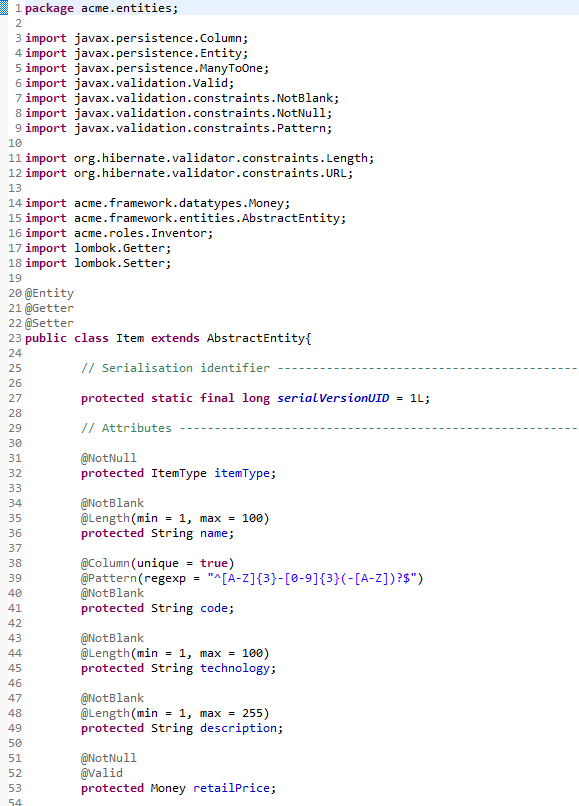


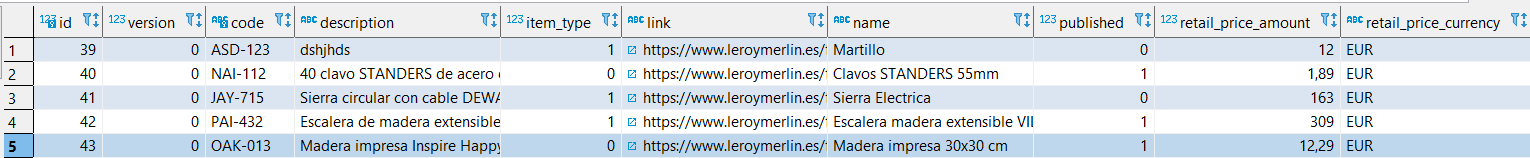
* La implementación de los flujos de trabajo implementados por los controladores se brindan a través de **servicios**. Los servicios se basan en repositorios para interactuar con las bases de datos. Reciben consultas en JPQL, las traducen a SQL, las envían a la base de datos y obtienen los resultados.

En este proyecto los servicios son implementados desde la interfaz “AbstractService” que a su vez puede ser “AbstractListService” o “AbstractShowService” dependiendo del tipo de vista sobre la que se quieran mostrar los datos.

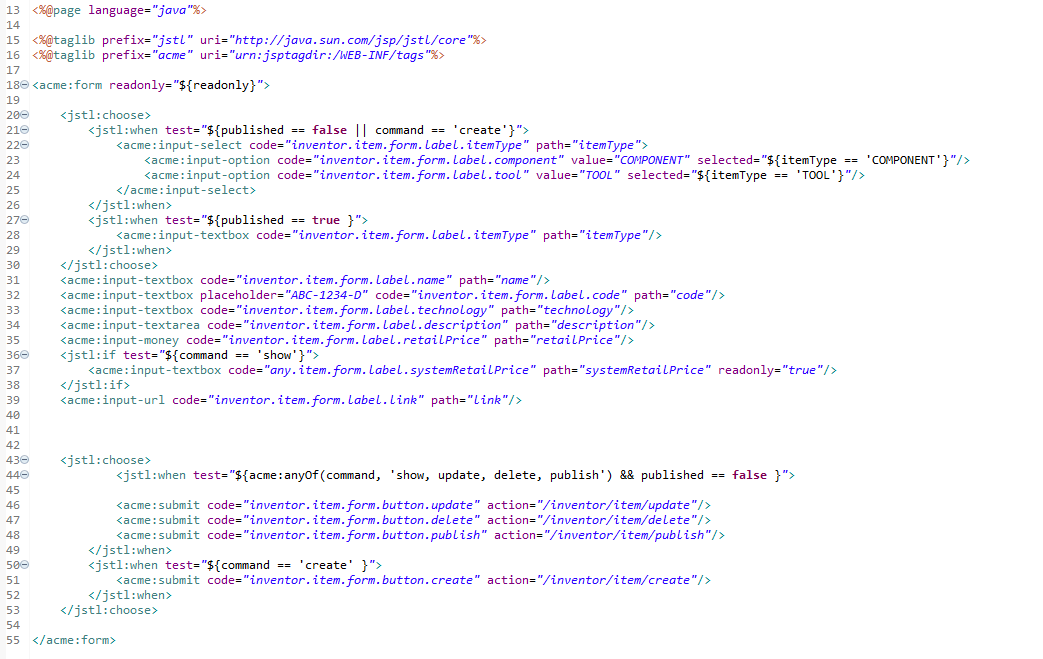
Los métodos del servicio funcionan a través de las clases:

* Request: empaqueta los parámetros que el servidor servlet pasa a sus controladores.
  + Method: puede ser "GET", que se origina en un enlace o un botón, o "POST", que se origina en un envío.
  + Command: codifica las acciones a realizar. Puede ser un comando básico, como "list" o "show", o un comando personalizado definido por el usuario.
  + baseCommand: Si el comando es un comando personalizado, debe tener un comando base que ayude al marco a comprender su semántica.
  + Principal: tiene datos sobre el usuario que realizó la solicitud.
  + Locale: indica cuál es el idioma seleccionado por el principal.
  + Model: incluye los datos que provienen del navegador, que incluye campos de formulario y parámetros de URL.
  + Unbind: copia un atributo de un objeto en un modelo.
* Model: Un modelo representa una colección de objetos como pares (atributo, valor). El marco se preocupa por la mayoría de los detalles con respecto a las colecciones, de modo que puede asumir con seguridad que un modelo siempre está listo para trabajar en un solo objeto.
  + hasAttribute: Este método devuelve un valor booleano que indica si este modelo contiene un atributo con el nombre dado o no.
  + Métodos getter: permiten obtener el valor de un atributo existente; hay un método genérico y luego algunos envoltorios que facilitan obtener el valor y convertirlo en un tipo básico específico.
  + setAttribute: permite establecer el nuevo valor de un atributo.
* Las **entidades** proporcionan una implementación a los requisitos de información. Siempre que se crea una entidad es necesaria la anotación javax.persistence.Entit al comienzo de la clase, y también indicarle el tipo y nombre de cada atributo que presenta, y de manera opcional unos validadores. Constituyen un modelo de datos que se mapea en una base de datos donde cada una de las filas representa a una instancia de dicha entidad, y cada columna refleja un atributo perteneciente a la entidad.





* Las **vistas** se utilizan para especificar cómo se debe representar una interfaz de usuario para una función determinada mediante HTML + CSS + JS. En nuestro proyecto las vistas han sido representadas mediante ficheros .jsp. Estas se componen de:
  + Encabezado
  + Librerías de etiquetas (JSTL y ACME)
  + Texto, dependiendo del tipo de vista, a la hora de mostrar las listas usaremos vistas jsp con una estructura concreta, que denominaremos “list”. Cuando queramos mostrar la información relativa a un elemento de la lista (hacer un “show”) usaremos una vista jsp que denominaremos “form”.



*Captura de una de las vistas del proyecto.*

Las solicitudes a un WIS se realizan mediante solicitudes HTTP a una URL en particular. A continuación, presentamos su estructura.

Una **URL** comienza con un protocolo. Por lo general, solo debe preocuparse por HTTP. HTTP es perfecto para el desarrollo y HTTPS se maneja externamente en producción, por lo que en realidad no tiene que preocuparse por otros protocolos.

Luego viene el **host**, que es la dirección IP o el nombre DNS al que envías tus solicitudes. En desarrollo, es comúnmente "localhost"; en producción, es comúnmente un nombre DNS como "[www.acme.com](http://www.acme.com)".

Luego viene el **puerto de comunicaciones**. Por lo general, es "8080" en desarrollo y "80" en producción.

La ruta ayuda a estructurar su solicitud. Cada ruta es atendida por un controlador particular dentro de su aplicación. La mayoría de las rutas tienen el formato "/función/entidad/comando", pero hay algunas excepciones, como en esta diapositiva.

Por último, la base de datos proviene de los creadores de MySQL, y es llamada **MariaDB**. Esta herramienta combinada con el gestor **DBeaver** nos permite tener acceso en tiempo real a nuestros datos, guardando la información necesaria, y utilizando el segundo para poder visualizar la información contenida en esta.

# 

# **Conclusiones**

Al comenzar la asignatura, teníamos unos conocimientos muy básicos respecto a la arquitectura WIS. A fecha de finalización de este documento, hemos trabajado durante ya casi cuatro meses con una arquitectura WIS y nuestras aptitudes e información al respecto ha aumentado lo suficiente como para poder entender su funcionamiento, los distintos elementos que la conforman y la utilidad de cada uno de ellos.

Su formación, dividida en tres grandes capas, navegador, servicios y base de datos, busca aumentar la compartimentación y reducir el acoplamiento.

Además, tras realizar varias búsquedas en Internet, en inglés y en español, sobre este tipo de arquitectura, hemos podido comprobar que en español la documentación sobre ella es muy escasa, aunque no ocurre lo mismo con la búsqueda en inglés.

A pesar de los escasos conocimientos que teníamos con respecto a la arquitectura WIS, hemos podido afianzar de buena manera los conceptos sobre esta que nos ofrecieron en la asignatura y somos capaces de ver su gran funcionamiento.

# 

# **Bibliografía**

Intencionalmente en blanco