

D05 - Advanced Topics

R03 - Arquitectura WIS

FECHA	VERSIÓN
31/05/2022	V1.4

REPOSITORIO: <https://github.com/marinaramirofde/Acme-Toolkits.git>

GRUPO DE PRÁCTICAS	E6.07
AUTORES	ROLES
Marina Ramiro Fernández marramfer12@alum.us.es	Manager, Developer, Tester and Operator
Ángel Lorenzo Casas anglorcas@alum.us.es	Developer, Tester

TABLA DE CONTENIDO

1. [Resumen ejecutivo](#)
2. [Tabla de revisiones](#)
3. [Introducción](#)
4. [Arquitectura de un WIS](#)
5. [Conclusión](#)
6. [Bibliografía](#)

1.- Resumen ejecutivo

El documento que se desarrolla está centrado en el término WIS (sistema web de información). A lo largo de la asignatura hemos visto las funcionalidades, la arquitectura y las posibles alternativas en las que se puede aplicar. El objetivo fundamental al que queremos llegar es la transmisión de las bases que conforman este tipo de sistemas, así como sus distintas partes.

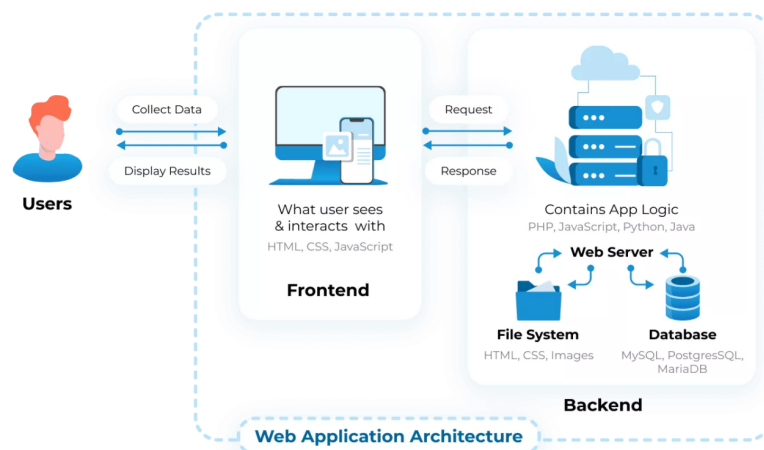
2.- Tabla de revisiones

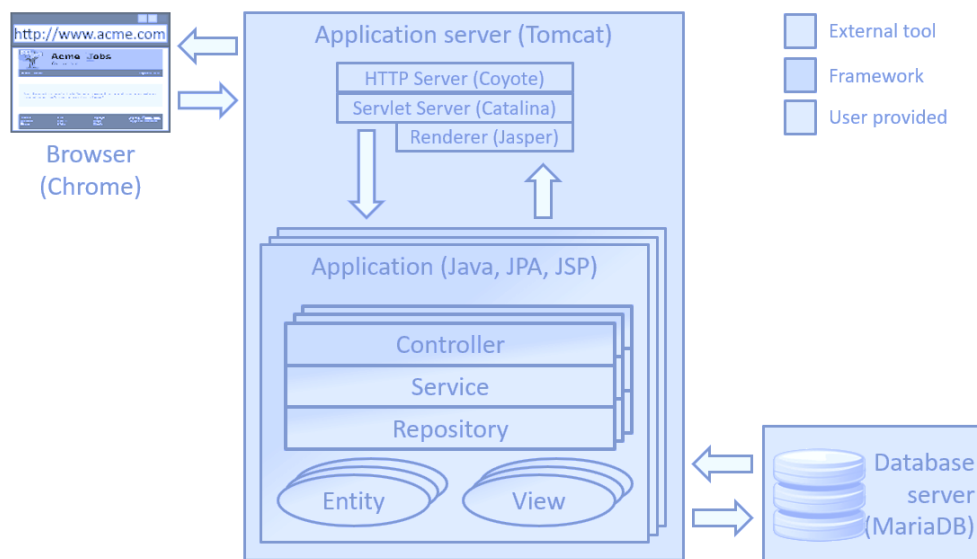
Versión	Fecha	Descripción
V1.0	24/05/2022	Creación del documento y sus partes.
V1.1	24/05/2022	Modificación
V1.2	26/05/2022	Ampliación de los puntos
V1.3	28/05/2022	Ampliación del contenido
V1.4	31/05/2022	Revisión y conclusión

3.- Introducción

Los componentes del equipo han decidido agrupar todas las ideas en un mismo documento. Se reparten los conceptos en varios apartados en los que mostramos brevemente y con nuestras palabras, los puntos claves. Como resultado obtenemos un documento en el que se brindan las bases de un WIS una vez finalizado el proyecto de la asignatura.

4.- Arquitectura de un WIS





Estructura del proyecto - Imagen de las diapositivas

MVC (MODELO, VISTA, CONTROLADOR)

Un WIS implementa el patrón arquitectónico MVC o modelo-vista-controlador, el cual determina la manera en la que los distintos componentes se comunicarán entre ellos. En este patrón se distinguen 3 partes:

- **Modelo:**
Está constituido por toda la información almacenada en la aplicación y se encarga de gestionar tanto el acceso a ella como su actualización.
- **Vista:**
Presenta la información a los usuarios de forma adecuada.
- **Controlador:**
Monitoriza los eventos que generan las vistas e invoca peticiones adecuadas al modelo para responder a estas. También responde a los cambios que puedan surgir en el modelo y actualiza las vistas en función de dichos cambios.

FUNCIONAMIENTO DE LA ARQUITECTURA

Para cada una de las funciones que realiza la aplicación es necesario implementar un modelo MVC, formada por controladores, servicios, repositorios, entidades y vistas. Un posible ejemplo sería que un Usuario utilizara el navegador para entrar en la aplicación web. Una vez dentro, realiza ciertas solicitudes (HTTP GET/POST) al servidor que a su vez

las envía mediante el servidor servlet y éste se las pasa a la aplicación. Una vez lo procesa mediante un controlador, que utiliza al servicio, que se basa en un repositorio, las entidades y las vistas, recupera y/o almacena los datos necesarios de la base de datos y devuelve una respuesta.

En los controladores nos encontramos con las direcciones url asociadas a los servicios donde al acceder a las mismas y poseer los permisos adecuados comprobados por el authorise del servicio, será el usuario permitido la visualización a la vista o la realización de la acción asociada a la url como podría ser, la acción de un update, delete, publish o un create o la vista asociada a un list o un show.

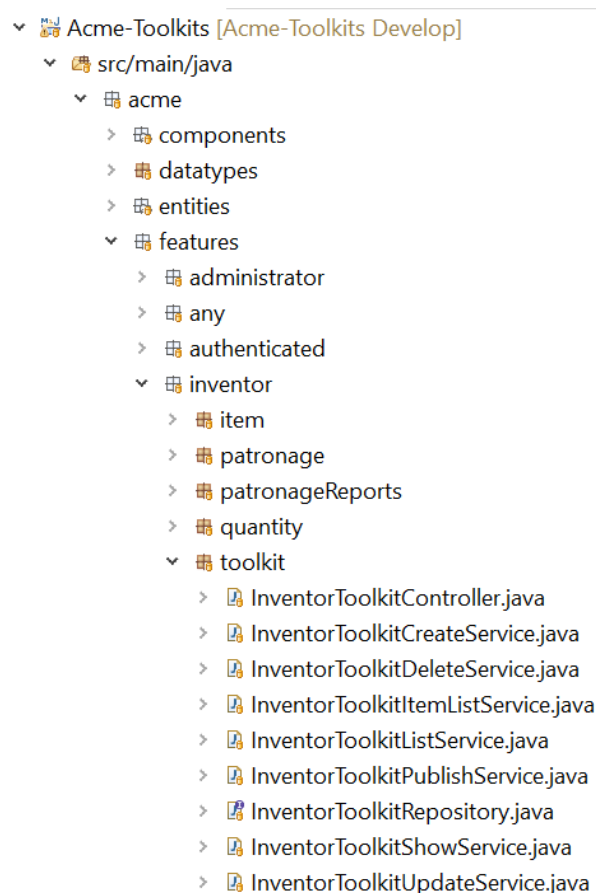


Imagen de ejemplo de las distintas clases creadas

Concretamente, las tecnologías y los tipos de archivos utilizados en el proyecto son:

- TomCat

Tecnología usada para contener servlets que se usa también para ejecutar y compilar aplicaciones web realizadas en java como la nuestra. Además tomcat da soporte a JSP o Java Server Pages.

- Dbeaver

Aplicación que se emplea para observar y modificar la base de datos donde se guarda la información de las entidades creadas en la aplicación web.

- Spring boot

Tecnología para la creación de aplicaciones autocontenidas.

- Vistas

Son archivos jsp que se utilizarán para concretar lo que mostrará el navegador una vez esté corriendo la aplicación y se haya accedido a la url asociada. Entre ellas están: Form.jsp, List.jsp, etc.

- Datos

En nuestra aplicación encontramos el "Initial data", que popula inicialmente la bases de datos con los datos de los roles y usuarios, además de la configuración del sistema mediante .csv. Por otro lado tenemos el "Sample data", el cual popula de datos a las otras tablas. Ambos se encuentran en nuestra aplicación en src/main/webapp/WEB-INF/resources.

- Controlador

Existe uno por par rol, entidad y relaciona los servicios de la rol, entidad con las vistas de la misma.

- Servicios

Show

El cual muestra los datos concretados de un objeto en específico de la base de datos en la vista, su jsp es el form.

ListMine

El cual muestra una lista de los datos concretados de un grupo de objetos en específico de la base de datos en la vista, su jsp es el list.

Create

El cual crea un objeto nuevo en la base de datos con la información proporcionada por la vista su jsp es el form.

Delete

El cual te permite borrar un objeto en específico de la base de datos en la vista, su jsp es el form.

Update

El cual te permite actualizar los datos de un objeto en específico de la base de datos en la vista, su jsp es el form.

Publish

El cual te permite actualizar el valor del published de false a true de un objeto en específico de la base de datos en la vista, su jsp es el form.

- Repositorio

Donde se encuentran y realizan los métodos para realizar las consultas o acciones a la base de datos por parte de la aplicación en tiempo de ejecución.

Algunos métodos de repositorios comunes son:

findManyXByY:

Este método saca muchos objetos del tipo de entidad X a través de un filtro realizado por el atributo Y y que suele ser proporcionado para realizar el método

findOneXbyId:











Este método saca un objeto del tipo de entidad X a través de la id asociada a este.

Ejemplos más específicos

Ejemplo Inventor > Item

Inventor es un rol de nuestro proyecto que hace uso de la entidad Item. Como vemos, para poder mostrar una lista con los items asociados a un inventor concreto debemos de crear un servicio específico denominado “InventorListMineItemService”. Además, para realizar operaciones CRUD y publish añadimos otros servidores más que se encargan de cada una de las acciones. Todos los servicios están recogidos en un mismo controlador y están basados en un único repositorio, que es exclusivo para esa relación Inventor-Item.

```

  ▼  inventor
    ▼  item
      >  InventorItemController.java
      >  InventorItemCreateService.java
      >  InventorItemDeleteService.java
      >  InventorItemPublishService.java
      >  InventorItemRepository.java
      >  InventorItemShowService.java
      >  InventorItemUpdateService.java
      >  InventorListMineItemService.java

```

Aparte de estas clases Java, hacemos uso de archivos .csv para popular la base de datos y archivos .jsp para la creación de las vistas.

- **Ejemplos de archivos .csv en nuestro java:**

```

  v Acme-Toolkits [Acme-Toolkits Develop]
    > src/main/java
    v src/main/webapp
      > META-INF
      v WEB-INF
        v resources
          v initial-data
            administrator.csv
            anonymous.csv
            system-configuration.csv
            user-account.csv
          v sample-data
            announcement.csv
            chirp.csv
            consumer.csv
            inventor.csv
            item.csv
            patron.csv
            patronage.csv
            patronage-report.csv
            provider.csv
            quantity.csv
            toolkit.csv

```

- **Ejemplos de archivos .jsp en nuestro java:**

- ▼ 📁 views
 - > 📁 administrator
 - > 📁 any
 - > 📁 authenticated
 - > 📁 fragments
 - ▼ 📁 inventor
 - ▼ 📁 item
 - 📄 form.jsp
 - 📄 list.jsp
 - 📄 messages-en.i18n
 - 📄 messages-es.i18n
 - > 📁 patronage
 - ▼ 📁 patronage-report
 - 📄 form.jsp
 - 📄 list.jsp
 - 📄 messages-en.i18n
 - 📄 messages-es.i18n
 - > 📁 quantity
 - > 📁 toolkit
 - > 📁 patron

5.- Conclusión

Gracias a la estructura seguida, podemos tener una visión general sobre cada componente del proyecto así como los archivos con los que se relaciona.

6.- Bibliografía

[1] Nuestro proyecto Acme-Toolkit

[2] Transparencias de clase