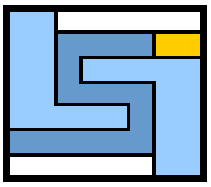
Diseño y Pruebas II



WISS REPORT

- **Cover**

Group: E7-06

Repository: https://github.com/plss12/Acme-Toolkits

Colleagues:

1. Cabezas Villalba, Juan Pablo (juacabvil@alum.us.es)
2. Martínez Jaén, Javier (javmarjae@alum.us.es)
3. Moreno Calderón, Álvaro (alvmorcal1@alum.us.es)
4. Navarro Rodríguez, Julio (julnavrod@alum.us.es)
5. Parejo Ramos, Salvador (salparram@alum.us.es)
6. Soto Santos, Pedro Luis (pedsotsan@alum.us.es)

- **Revision table**

Version	Date	Description
1.0	26/05/2022	First revision

- **Table of contents**

- Executive Sumary

En este informe se comentará de manera general el conocimiento adquirido sobre la arquitectura WIS a través de la asignatura. Antes del comienzo de la misma, como se explicó en el primer informe, se habían tratado aplicaciones similares en otras asignaturas (DP1). Gracias al uso de Acme-Framework (con todas las facilidades que este ofrece), hemos podido implementar muchas características típicas de aplicaciones WIS de una manera más o menos sencilla, sin preocuparnos por los aspectos más detallados. Finalmente se han adquirido conocimientos avanzados de esta arquitectura mediante el transcurso de la asignatura a base de ensayo y error siendo ayudados por la teoría y las prácticas.

- Introduction

En esta asignatura hemos llegado teniendo vagas ideas sobre Spring Framework, un esbozo simple de su funcionamiento y un conjunto de aplicaciones útiles por parte del mismo.
Sabíamos que Spring tenía muchísimas capacidades y opciones, era muy versátil, pero eso podía llegar a ser un problema dado que no éramos expertos y aún estábamos aprendiendo.

Diseño y Pruebas 2 ha sido una asignatura de acercamiento a ese tipo de frameworks, con esa arquitectura concreta. Gracias a Acme-Framework, que imponía una gran base a todas esas opciones posibles en Spring, nos hemos podido centrar más de lleno en comprender nociones clave del mismo sin tener que dedicar excesivo tiempo estudiando previamente todo el funcionamiento global.

- **Contents**

Contexto de la Asignatura

El contexto de la asignatura es claro y simple, simulamos que estamos en un escenario pseudo-profesional y se nos establecen unos requisitos a cumplir en plazos determinados, teniendo que tomar algunas decisiones de diseño por el camino.

Si para cumplir estos requisitos hubiésemos tenido que usar Spring Framework, probablemente habríamos tenido que emplear mucho más tiempo en realizar features simples. Spring tiene un propósito muy general, sigue la arquitectura WIS pero abarcando muchas posibilidades. A nosotros se nos pedía un estilo concreto de producto web, con unas reglas básicas, los usuarios que usaban la web debían de tener unos roles asociados, incluso el rol anónimo si no estaban registrados. Existían unas entidades concretas, con una serie de datos, algunos de estos incluso debían ser datos avanzados (de tipos no básicos ni primitivos), estas entidades tendrían unas relaciones y se podrían listar, mostrar, crear, editar o eliminar.

A simple vista este es el modelo típico de un WIS, y también el que cumple Spring Framework. Habría que comenzar a establecer estas reglas básicas de negocio en Spring de manera que funcionasen, esa sería una configuración inicial, tras esa configuración, habría que comenzar con la implementación.

Acme-Framework

Probablemente configurar Spring de manera inicial y llevar a cabo las tareas nos habría hecho comprender en totalidad el funcionamiento y la idea de WIS.

Pero se requiere más tiempo del que quizá disponemos para realizar eso, y realmente para comprender los WIS, no es estrictamente necesario crear un producto así desde cero. *No tiene sentido en ciertas situaciones reinventar la rueda.* En nuestro caso la rueda es Acme-Framework.

La premisa de este framework es sencilla, sistema de roles, atributos avanzados como el tipo Money, y en general, muchas entidades abstractas que ya implementan la parte más compleja y profunda son las que encontramos.

Controladores, servicios de tipo show, list, create, update, delete, justamente los necesarios para llevar a cabo la implementación, se nos dan ya de manera abstracta, con la definición base de los métodos necesarios para el funcionamiento.
Se comprenden perfectamente las tres posibles capas en esta arquitectura: vista, controlador y modelo; pero se nos facilita la creación de las mismas: AbstractController.java, AbstractService.java, AbstractEntity.java…

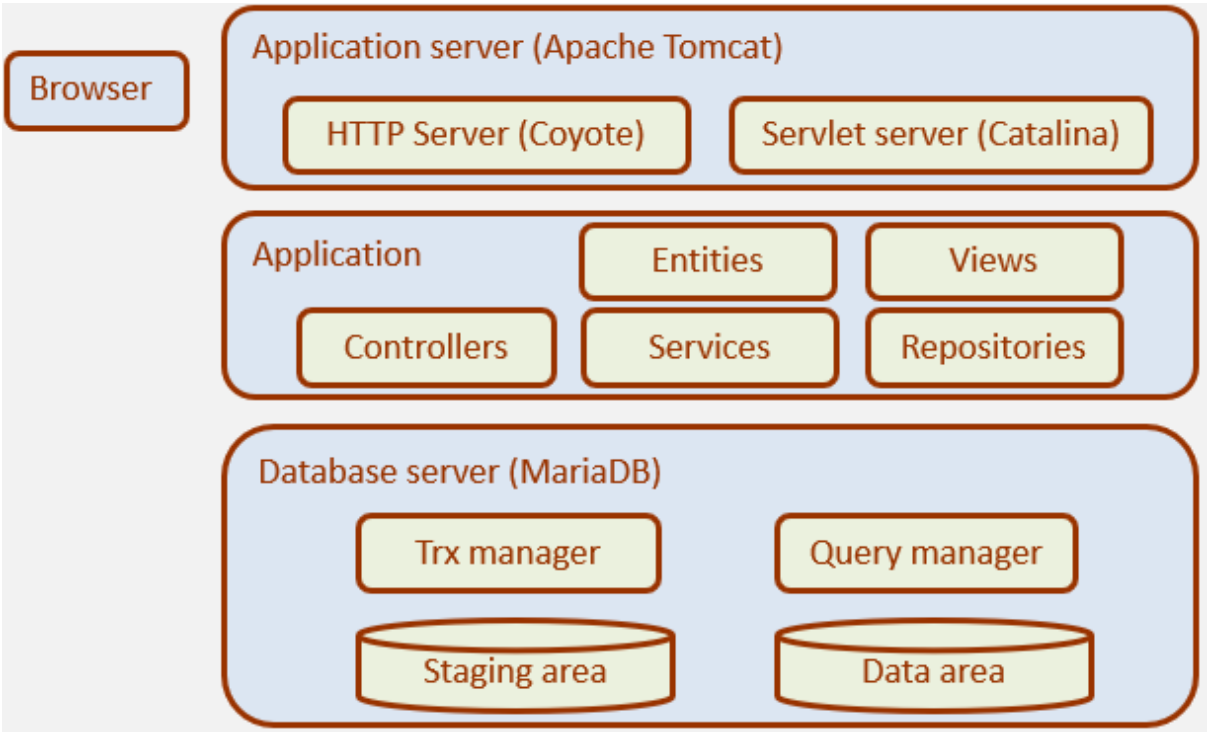
Tras todo eso solo nos tenemos que centrar en desarrollar aquellos requisitos que se nos piden, teniendo ya una base más que suficiente para llevarlo a cabo.
Cabe recalcar, que al ser Acme-Framework una simplificación y ayuda para la implementación de arquitecturas WIS, tareas que se quieran realizar fuera de las establecidas inicialmente podrían llegar a ser difíciles de implementar o podrían suponer "toquetear" el framework, una tarea ya más avanzada.

Arquitectura WIS

Hemos aprendido la estructura de una arquitectura WIS, arquitectura que sigue un patrón MVC ("Model View Controller").

- Modelo: es la representación específica de la información con la que se opera, donde se junta la lógica de negocio con el repositorio de datos y generalmente brindan la información en forma de servicios (API)
- Vista: se le ofrecen los datos del Modelo al cliente de forma usable a través de una interfaz de usuario por lo general.
- Controlador: responde a los eventos que se producen en la Vista (peticiones HTTP) y provoca cambios en el modelo y por extensión, en la Vista.

En cuanto a la estructura de la arquitectura WIS de Acme-Framework:



Está compuesta por 4 capas diferenciables:

- El navegador (“browser”) : renderiza los documentos HTML y convierte las interacciones de ususario en peticiones HTTP.
- El servidor de aplicación (en nuestro caso, Apache Tomcat): en él se trata el protocolo HTTP, la infrastructura de JAVA servlet (encargado de generar la página web) y la renderización de las vistas.
- La capa de aplicación (está relacionado con el patrón MVC) está compuesta por:
    - Entidades: elementos de la aplicación que tienen una serie de atributos. Estos quedan almacenados en la base de datos.
    - Repositorios: encargado de hacer peticiones a la base de datos a través de las queries.
    - Servicios: hacen uso de los repositorios y son los encargados de prestarles servicios a los controladores.
    - Controladores: hace uso de los servicios y es el que contiene la lógica que actualiza las vistas.
    - Vistas: definen cómo se deben mostrar los datos en la aplicación.
- El servidor de la base de datos (en nuestro caso, MariaDB): es el encargado de manejar los datos. Se compone de:

- Trx Manager (transaction manager): es el encargado de organizar las queries que modifican la base de datos para que puedan ser ejecutadas en un batch (sistema que ejecuta un programa sin la supervisión de un usuario).
- Query manager: es el encargado de ejecutar las queries select y organizar las queries insert, update o delete.
- Staging area: almacena los diarios de transacciones. Dichos diarios se componen de una lista de operaciones insert, update o delete a realizar en la base de datos en el contexto de una transacción particular.
- Data area: almacena los datos que han sido comiteados a la base de datos. Las select queries leen los datos de este area. Aunque si alguna query insert, update o delete los han modificado, dichos datos son leídos en la staging area.

- Conclusions

Siendo Acme-Framework nuestro primer paso hacia las WIS en un terreno pseudo-profesional, hemos comprendido de manera bastante clara y orgánica todos los elementos necesarios en este tipo de proyectos. Hemos experimentado el por qué se necesitan vistas, modelos y controladores, la utilidad de los mismos y sus relaciones. Además hemos visto el sentido que tiene separar la aplicación de esta manera, haciendo que los elementos lleven a cabo tareas que de manera separada se pueden realizar y que al juntarlas se obtenga la aplicación buscada.

- **Bibliography**

Intencionadamente en blanco.