WIS ARCHITECTURE REPORT

1 DE JUNIO DE 2022

E2.07

https://github.com/anac29/Acme-Toolkits

Jaime Borrego Conde: jaiborcon@alum.us.es

Antonio Campos Gil: antcamgil@alum.us.es

Ana Conde Marrón: anaconmar@alum.us.es

Gonzalo Martínez Fernández: gonmarfer2@alum.us.es

Jaime Moscoso Bernal: jaimosber@alum.us.es

Enrique Muñoz Pérez: enrmunper@alum.us.es

Tabla de contenido

Resumen ejecutivo

El objetivo del presente documento es exponer los conocimientos que el equipo ha ido aprendiendo y desarrollando respecto la arquitectura WIS a través de la asignatura Diseño y Pruebas II.

Este documento no trata de exponer cómo funciona cualquier tipo de arquitectura WIS, sino de los conocimientos adquiridos de la arquitectura respecto a las lecciones tanto teóricas como prácticas. Es por ello por lo que estará enfocado en la arquitectura seguida en el proyecto Acme-Toolkits, desde el que se han sacado gran parte de los conocimientos.

Tabla de revisión

Número de revisión	Fecha	Descripción
0	26/05/2022	Primera versión

Introducción

Este documento detalla todo lo aprendido respecto a la arquitectura WIS. Se seguirá el mismo esquema de conocimiento que el seguido en las diversas clases teóricas y prácticas. Se comenzará con una explicación del modelo de datos, donde se tratará a entidad AsbtractEntity y sus diversos usos en la aplicación Acme-Toolkits.

Más adelante, se tratarán las taxonomías y las relaciones, hablando de su presencia e importancia para la navegabilidad y el flujo de datos.

A continuación, se nombrará la entidad base, que se usará para la creación de cualquier tipo de entidad, y las restricciones en las entidades, que son fundamentales para la gestión y control de la información.

Luego, se englobará el concepto de datos de ejemplo, hablando de la importancia de trabajar en los rangos y de cómo generar un buen conjunto de datos de ejemplo.

A partir de ahí, se hablará del lenguaje JPQL, altamente usado en Acme-Toolkits y que permite la realización de consultas de una manera algo diferente al extendido lenguaje SQL.

Una vez tocado todo lo que engloba la arquitectura, se entrará de lleno en ella. Se tratarán los controladores, los servicios, los repositorios, las diversas capas, como funcionan y se conectan, los tipos de peticiones, cómo se realizan, las vistas, etc.

Más tarde, se englobará el concepto de edición de datos, y como ocurre todo el proceso para la creación, edición y borrado.

Por último, se explicará los flujos de trabajo, esenciales para entender cómo se transmitirán los datos en los servicios.

Contenido

Modelos de datos

Los modelos de datos permiten representar los requisitos de información en un lenguaje que sea fácil de entender y manipular para los desarrolladores.

En primer lugar, se hablará de las entidades, que son objetos persistentes manipulados y creados en Java, pero guardados en la base de datos. Se utilizará la clase "AbstractEntity" que ofrece una base para las entidades que se creen. Se debe indicar la etiqueta <Entity> para cada objeto de tipo Entidad que aparezca en el diagrama UML. Cada entidad que se cree tendrá:

- Un id identificativo
- Un número de versión
- Un atributo transient
- Un método hashcode()
- Un método equals()
- Un método toString()

Todo esto será proporcionado por la entidad abstracta.

A la hora de crear las entidades en Acme-Toolkits, estas tendrán que heredar de <AbstractEntity>. Se añaden los atributos que conforman la clase indicando el nombre de ese atributo, su tipo y se indicará si es derivado o no.

Taxonomías

Las taxonomías permiten clasificar las entidades jerárquicamente. Algunas entidades pueden ser instancias de otras entidades como pasaba en el proyecto de Acme Jobs con Employer y UserRole. Los niveles de la taxonomía los dictará el tipo de proyecto, pero generalmente de 2 a tres es lo idóneo.

Relaciones

Las relaciones permiten unir entidades y por consiguiente conseguir la navegabilidad tan necesaria en este tipo de arquitecturas. Gracias a la navegabilidad se pueden obtener atributos de otras entidades y tratar con sus datos.

Hay dos tipos de navegabilidad:

- *Single-way navigation*: las relaciones se interpretan en único sentido.
- *Two-way relations*: en este caso se interpretan en ambos sentidos.

La multiplicidad puede ser muy variadas. Existen estos cuatro tipos de relaciones:

- *OneToOne*
- *ManyToOne*
- *OneToMany*
- *ManyToMany*
- Conglomerados
- Agregaciones

Cada una pudiendo ser *single-way* o *two-way*.

Restricciones

A la hora de modelar las entidades se debe añadir a los atributos ciertas restricciones que permitan que no cualquier dato se pueda introducir en la base de datos.

Se encuentran varios tipos de restricciones:

- Restricciones en objetos: @NotNull, @Valid
- Restricciones en números: @Range(min, mx), @Max(max)…
- Restricciones en texto: @NotBlank, @Length(min, max)…
- Restricciones en momentos: @Past, @Future…

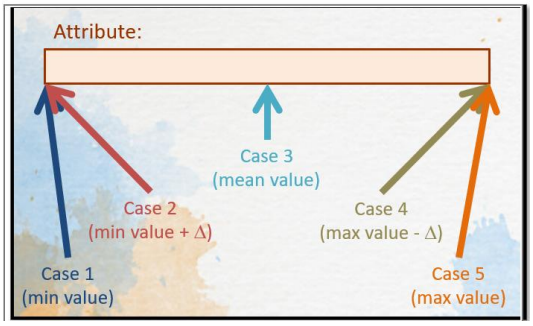
Los patrones también juegan un papel muy importante pues permiten comprobar que ciertos atributos siguen un formato específico.

Entidades Base

Es importante ordenar las entidades topológicamente de menos a más dependencias. Lo primero sería crear entidades base: una entidad base es aquella cuyos valores son “normales”. Por ejemplo, una entidad que está formada por un conglomerado de otras entidades no será “normal”, pero si es un de estas entidades del conglomerado y esta no tiene hijos será considerada como “normal”.

Importancia de los rangos

Es significativo que los datos de ejemplo oscilan en un intervalo determinado que viene explícito en las propias restricciones.



Se necesita crear datos de ejemplo, no datos de prueba. Por tanto, tienen que ser válidos, es decir, que pasen los validadores. Cuando se crean los datos de ejemplo, se deben variar los valores dentro del rango de cada atributo. Esto quiere decir que se deben poner atributos repartidos por todo el rango que delimitan las restricciones, pues no se quieren datos aleatorios, sino unos que prueben la eficiencia de los validadores.

Cómo generar datos de ejemplo

Colecciones

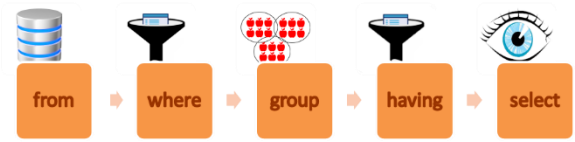
A la hora de poblar tablas de entidades con atributos de tipo colección, se deben crear objetos con colecciones de 1 elemento, de más de 1 y de ninguno.

Conglomerados

Un conglomerado no es un solo objeto, sino una colección de objetos relacionados. Por tanto, esta colección no está guardada explícitamente en un atributo, pero existe. Es por ello que habrá que generar datos con 1 objeto en el conglomerado, con más de uno y con ninguno.

JPQL

Gracias a JPQL se pueden realizar consultas navegando por los atributos de las entidades Java. Esto hace bastante fácil obtener resultados, pues es bastante predictivo y Java da bastantes facilidades para crear estas consultas. Difiere de la sintaxis habitual de SQL pero no demasiado.



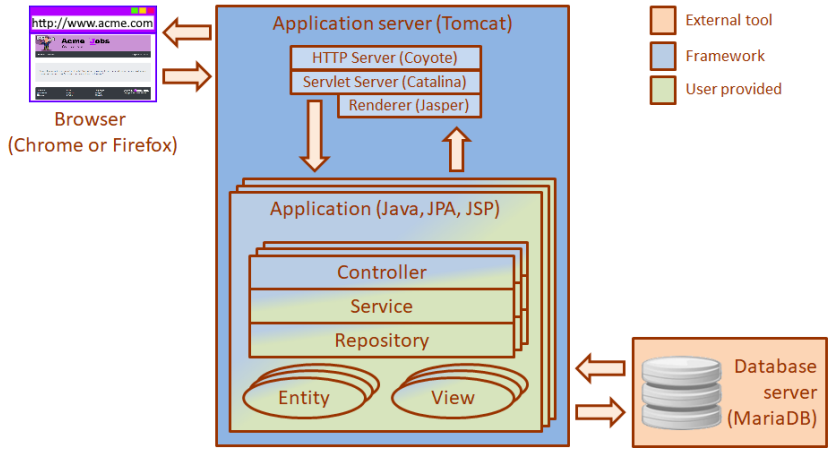
Con JPQL se puede obtener datos de diferentes entidades gracias a la navegabilidad. Dependiendo del tipo de relación, la obtención de datos se hará de una manera u otra. Si es de tipo to-one, la relación es directa, pero, en el caso de ser to-many, la obtención de atributos se complica. Es por ello que hay que potenciar las relaciones to-one.

También se pueden usar subqueries para comprobar ciertas condiciones, como si el resultado está vacío o no.

Arquitectura

A continuación, se ahondará en la arquitectura del sistema WIS.

Esquema general:



Se puede dividir en distintas capas:

- La capa del navegador:
  - Sirve para hacer peticiones HTTP a la capa de aplicación y para renderizar los datos que devuelve.
- La capa de Aplicación:
  - Recibe las peticiones HTTP y las pasa a la aplicación correcta (controladores) dependiendo de la URL.
- La capa de la Base de Datos: Recibe peticiones SQL y devuelve selects o (insert, update y delete)

Controladores

Un controlador atiende todas las peticiones relativas a una función concreta. Su implementación ocurre debido a los flujos de trabajo (crear, listar, mostrar, actualizar y eliminar).

El Base Controller

Es una clase que proporciona los métodos necesarios que servirán de esqueleto para los controladores que se creen en la aplicación. Se pide que se especifiquen tanto el rol de usuario como el tipo de objeto. Será en esta clase donde se incluya el método del handleRequst que implementará el flujo de trabajo para servir las peticiones.

Servicios

Los flujos de trabajos implementados en los controladores son genéricos. La implementación de sus pasos individuales depende de las características específicas para las que fueron diseñados y se proporcionan por medio de servicios.

Al igual que en el caso de los controladores, el framework también provee una clase abstracta Servicio (“AbstractService”) de la que heredarán el resto de las clases de tipo servicio. De nuevo, es necesario pasarle dos parámetros: el rol y el tipo de objeto. Existen servicios de tipo crear, editar, eliminar…

Dependiendo del tipo de servicio se añadirá unos métodos a implementar. Como es coherente, si se trata de un servicio que permite eliminar entidades, se añadirá un método delete que permita eliminar la entidad especificada. Si es un show, un método findOne que permita buscar un objeto concreto de entre todos y así con todos los tipos de servicio.

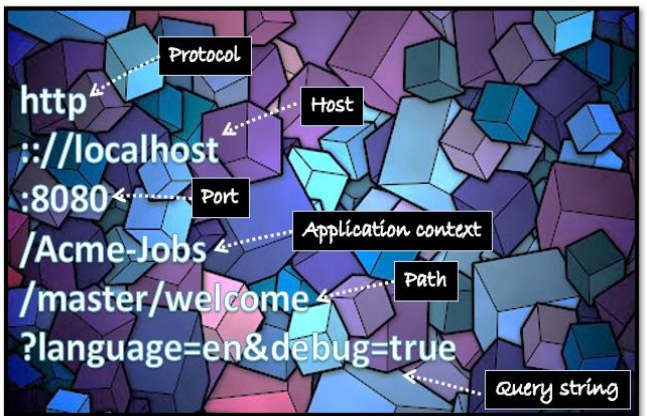
Cuando se creen servicios personalizados habrá que cerciorarse de que extiende del Servicio base correcto.

Algunos de los métodos a destacar serían:

- El método authorisation
- El método validate
- El método binding
- El método unbinding

URLs

Las solicitudes/peticiones a una arquitectura WIS se realiza a través de peticiones HTTP con una determinada estructura:



- Se usará http para el desarrollo y https para producción.
- El host debe ser o la ip o su nombre en DNS. Durante el desarrollo, se suele usar localhost
- El puerto es 8080 en desarrollo y 80 en producción. Para producción, el 443 (https)
- El contexto de aplicación es el nombre del proyecto en desarrollo y un string vacío en producción.

- El camino (path) sirve para estructurar las peticiones. Cada ruta es servida por un controlador particular dentro de su aplicación.
- La cadena de consulta (query String) proporciona parámetros a una solicitud. En Acme-Toolkits: dos parámetros por defecto: language, que controla la i18n, y debug, que controla la información de depuración que se muestra en cada página.

Cookies

Una cookie es un pequeño archivo que es creado por un servidor cuando trata una solicitud. Las cookies son enviadas por el servidor en la primera petición y devueltas por el navegador en cada petición posterior hasta que expiran o se borran.

En Acme-Toolkits se usan 3 tipos distintos:

- Languaje: Idioma
- JSESSIONID: Identifica la sesión del usuario.
- Remember: Token de autentificación encriptada.

Peticiones

Dos tipos: GET y POST. Las peticiones GET consisten en una URL más las cookies.

Repositorios

Las entidades proporcionan una implementación a los requisitos de información. Constituyen un modelo de datos que se mapea en una base de datos.

Vistas

Las vistas se utilizan para especificar cómo se debe renderizar una interfaz de usuario para una determinada característica utilizando HTML + CSS + JS.

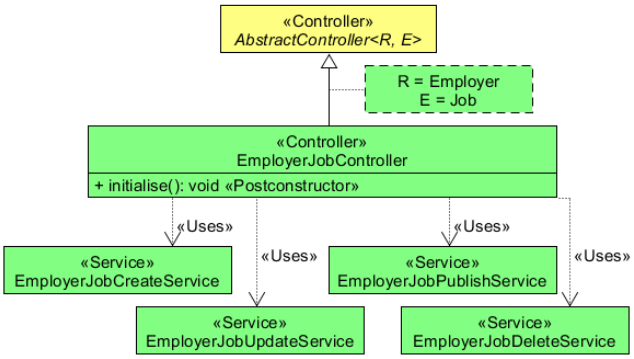
Edición de datos

La edición de datos tiene lugar en formularios. En este proyecto se usará JSP para el diseño de las vistas, así como las libreras que proporciona el framework para poder diseñarlas. Hay diferentes etiquetas que podemos utilizar para crear los formularios: inputs de texto, select, inputs de tipo integer, de tipo URL, de tipo correo…

A la hora de enviar los formularios serán necesarios botones. Se pueden insertar distintos botones en un formulario, pero no todos serán de tipo submit. Los botones de submit nos llevarán a otra URL que será dónde se haga la acción de editar el objeto en cuestión.

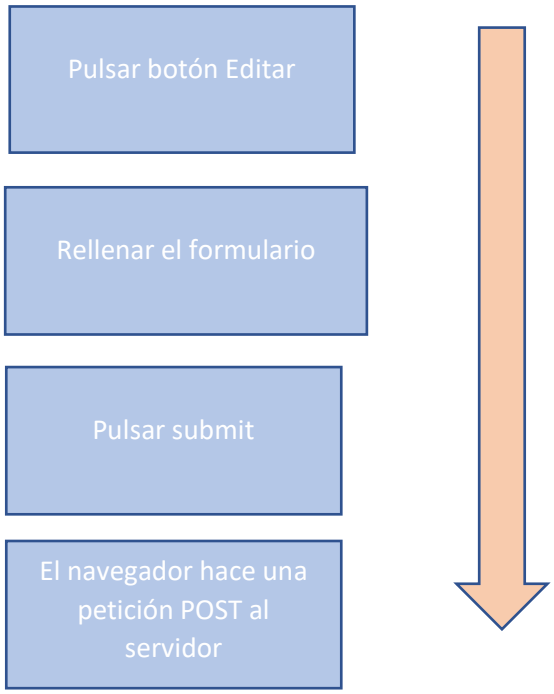
Controladores personalizados

Serán los que se desarrollarán para cada entidad o rol del que se quiera crear servicios: crear, editar o actualizar datos, entre otras operaciones.

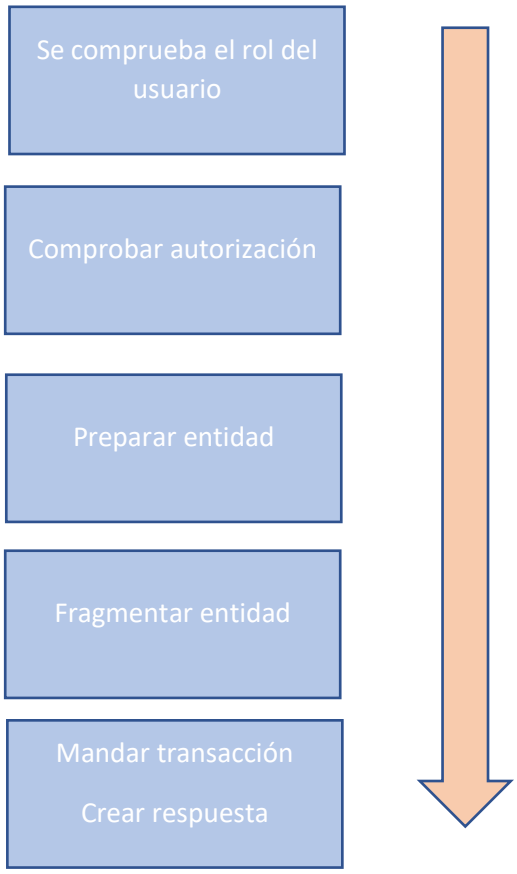


Como ya se especificó antes, en un controlador deberán indicarse tanto el rol como el tipo de objeto al que afecta, pues al heredar de la clase BaseController, deben ser provistos de ellos. El controlador se encargará de gestionar el flujo de trabajo de todos los servicios que implementemos.

Flujo de trabajo



Flujo de trabajo de peticiones GET



Conclusiones

En este documento se detalla que conocimientos se han adquirido en DP II. Han sido muchos conocimientos, pero al estar correctamente organizados no ha sido difícil adquirirlos. Además, han permitido al equipo de trabajo producir un código en su mayoría limpio y siguiendo buenas prácticas. Era la primera vez que se trabaja en profundidad usando un Framework y, aunque sí que es cierto que ha sido necesario un proceso de habituación, esto ha permitido automatizar muchos procesos y favorecer la automatización de desarrollo de código.

Al principio se creía que ya se disponía de los documentos de la arquitectura pues el equipo acababa de cursar la asignatura DP1, pero conforme fue avanzando el proyecto y se ahondó en el temario, fue observable que los conocimientos que se pensaba se tenían pecaban de un alto nivel de abstracción. Por último, el grupo se siente satisfecho con los conocimientos aprendidos y confía en que se pondrán en práctica en los años venideros.

Bibliografía

Material de la asignatura.