



ARQUITECTURA WIS

Grupo E7.02



Álvaro Úbeda Ruiz (alvuberui@alum.us.es)

Mario Pérez Coronel (marpercor8@alum.us.es)

Carlos Garrido Rodríguez (cargarrod12@alum.us.es)

Ramón Rodríguez Bejarano (ramrodbej@alum.us.es)

Mario Rodríguez García (marrodgar62@alum.us.es)

Juan Carlos Gómez Rodríguez (juagomram4@alum.us.es)

Repositorio: <https://github.com/alvuberui/Acme-Toolkits>

Tablero: <https://github.com/users/alvuberui/projects/5>

29 DE MAYO DE 2022

Contenido

1.	Resumen ejecutivo	2
2.	Tabla de revisión	2
3.	Introducción	2
4.	Contenido	2
4.1	Organización del proyecto	2
4.2	Test	3
4.3	Servicios, controladores y repositorios	5
5.	Conclusión	7
6.	Bibliografía	7

1. Resumen ejecutivo

El presente documento explicaremos los conocimientos que hemos adquirido tras la realización de la asignatura Diseño y Pruebas II complementando así los conocimientos ya obtenidos en asignaturas como Diseño y Pruebas I, Arquitectura e Integración de Sistemas Software o Introducción a la Ingeniería del Software y los Sistemas de Información.

2. Tabla de revisión

Num. Revisión	Fecha	Descripción
1	28/05/2022	Creado el documento y apartados 1 y 3.
2	29/05/2022	Realización del apartado 4.
3	29/05/2022	Realizada la conclusión.

3. Introducción

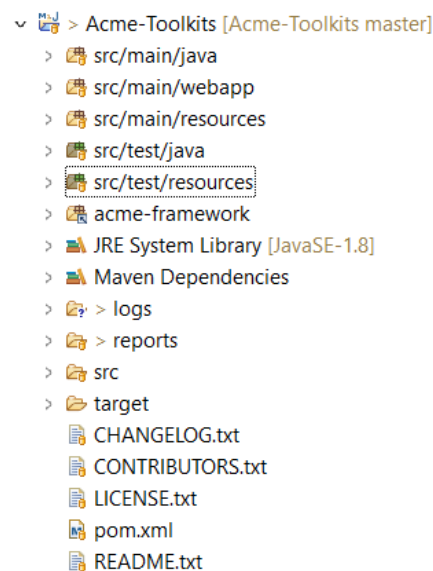
Como bien ya se ha explicado en el apartado uno “Resumen ejecutivo” este documento tiene como objetivo explicar que conocimientos hemos adquirido en la asignatura Diseño y Pruebas II una vez hemos terminado. Para ello vamos a desarrollar los nuevos conocimientos adquiridos sobre Arquitectura WIS.

4. Contenido

4.1 Organización del proyecto

En cuanto a la organización del proyecto se ha realizado mediante carpetas.






























- La carpeta “src/main” la cual incluye:
 - La carpeta “java” donde se realizarán todas las clases de código .java
 - La carpeta “webapp” que incluirán los archivos .jsp que serán las vistas, así como los datos iniciales y los datos de ejemplo.
 - La carpeta “resources” que incluye “properties” de la aplicación
- La carpeta “src/test” la cual incluye:
 - La carpeta “java” donde estarán las pruebas realizadas.
 - La carpeta “resources” la cual contendrá los datos de ejemplo de las pruebas.
- Las dependencias:
 - La carpeta acmé-framework que contiene el framework con el que hemos trabajado durante el curso.
 - La librería JRE
 - Las dependencias de Maven.



Por último, dentro de la carpeta “src/main/java” hemos organizado de la siguiente forma:

- Los paquetes `acme.entities.X` donde se han creado las diferentes entidades del proyecto.
- Los paquetes `acme.features` donde se han realizados las diferentes tareas pedidas.
- El paquete `acme.roles` donde se encuentran los distintos roles creados en este proyecto.

```

>  acme.entities.announcement
>  acme.entities.artefact
>  acme.entities.chirp
>  acme.entities.patronages
>  acme.entities.patronageReport
>  acme.entities.systemConfiguration
>  acme.entities.toolkit
>  acme.features.administrator.announcement
>  acme.features.administrator.dashboard
>  acme.features.administrator.systemConfiguration
>  acme.features.any.artefact
>  acme.features.any.chirp
>  acme.features.any.toolkit
>  acme.features.any.userAccount
>  acme.features.authenticated.announcement
>  acme.features.authenticated.inventor
>  acme.features.authenticated.moneyExchange
>  acme.features.authenticated.patron
>  acme.features.authenticated.systemConfiguration
>  acme.features.inventor.announcement
>  acme.features.inventor.artefact
>  acme.features.inventor.patronagereports
>  acme.features.inventor.patronages
>  acme.features.inventor.toolkits
>  acme.features.patron.dashboard
>  acme.features.patron.patronage
>  acme.features.patron.patronageReport
>  acme.forms
>  acme.roles

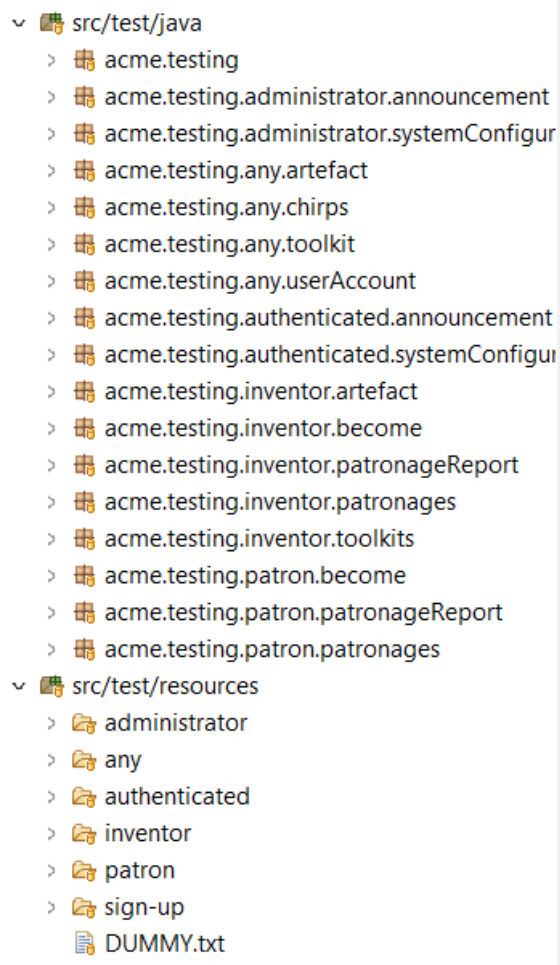
```

4.2 Test

Realización de los Test:

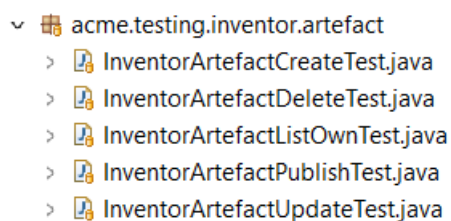
Otra parte sobre la que hemos aprendido en DP2 es la realización de los tests para las features creadas a lo largo de este proyecto.

Primeramente, hemos estudiado la estructura necesaria para la creación de los mismos.



Como podemos observar, tenemos una captura donde estan la organización de los diferentes tests creados por nuestro grupo. Vemos, que tenemos un paquete dedicado a las clases en las que se encuentran dichos tests, y por otro lado tenemos otro paquete donde se encuentran los ficheros con los datos en .csv que se utilizarán para testear la aplicación.

Después de saber la estructura general, tenemos que saber que dentro de cada test que realizamos para una feature, se puede volver a dividir en varios tests, que incluyen uno para el listado, otro para el show y los restantes correspondientes a crear, actualizar o borrar dependiendo si la feature que estamos testeando tienen esta posibilidad en nuestra aplicación.



Aquí vemos una captura de lo anteriormente mencionado, donde dividimos dichos los tests del Rol inventor con la entity Artefact.

Por último, vamos a observar lo aprendido y realizado mostrando un ejemplo de uno de los test de nuestro proyecto.

```
7
0 public class InventorArtefactCreateTest extends TestHarness {
1
2     @ParameterizedTest
3     @CsvFileSource(resources = "/inventor/artefact/create-positive.csv", encoding = "utf-8", numLinesTo
4     @Order(10)
5     public void positiveTest(final int recordIndex, final String type, final String name, final String
6         super.signIn("inventor1", "inventor1");
7
8         super.clickOnMenu("Inventor", "Own artefacts");
9         super.checkListingExists();
10
11         super.clickOnButton("Create");
12         super.fillInputBoxIn("type", type);
13         super.fillInputBoxIn("name", name);
14         super.fillInputBoxIn("code", code);
15         super.fillInputBoxIn("technology", technology);
16         super.fillInputBoxIn("description", description);
17         super.fillInputBoxIn("retailPrice", retailPrice);
18         super.fillInputBoxIn("moreInfo", moreInfo);
19         super.clickOnSubmit("Create");
20
21         super.checkListingExists();
22         super.sortListing(2, "desc");
23
24         //Revisar que la cabecera esta bien
25         super.checkColumnHasValue(recordIndex, 0, type);
26         super.checkColumnHasValue(recordIndex, 1, name);
27         super.checkColumnHasValue(recordIndex, 2, code);
28     }
```

Aquí podemos ver una captura de uno de nuestros tests. Primero, comentar que dentro de estas clases podemos hacer tests positivos donde comprobamos que la ejecución del mismo se realiza correctamente (como la captura adjuntada) y también hacemos tests negativos donde comprobamos que algo ha ido mal ya sea porque no tengamos autorización, o para comprobar una validación entre otras cosas. Dentro de estos tests hemos aprendido a usar TestHarness , que es el paquete importado que hemos utilizado, y sus funciones con super para rellenar input-box, comprobar que existen listados, logearnos con un Rol que queramos, comprobar que el propio test ha fallado si es negativo, y un sinfín de posibilidades.

4.3 Servicios, controladores y repositorios

Con respecto a el apartado de servicio, controladores y repositorios, su organización se realiza de la siguiente manera:

- acme.features.authenticated.inventor
 - AuthenticatedInventorController.java
 - AuthenticatedInventorCreateService.java
 - AuthenticatedInventorRepository.java
 - AuthenticatedInventorUpdateService.java
 - acme.features.authenticated.moneyExchange
 - AuthenticatedMoneyExchangeController.java
 - AuthenticatedMoneyExchangePerformService.java
 - AuthenticatedMoneyExchangeRepository.java
 - acme.features.authenticated.patron
 - acme.features.authenticated.systemConfiguration
 - AuthenticatedSystemConfigurationController.java
 - AuthenticatedSystemConfigurationRepository.java
 - AuthenticatedSystemConfigurationShowService.java

Como podemos observar, dentro de cada paquete específico tenemos los controladores, servicios y repositorios dedicados a esta tarea. Facilitando así su búsqueda, editado o creación y haciendo que todo este mas ordenado de cara a compañeros de trabajo actuales o futuros.

Si nos fijamos en la organización de cada uno de estos, en la mayoría de estos, hemos aprendido algo nuevo gracias a la asignatura. Excluimos los Repositorios ya que estos, ya obtuvimos los conocimientos actuales mayoritariamente de otras asignaturas tales como DP1 que usa el mismo patrón para estos.

Revisando los controladores, estos son mas limpios y sencillos. Ya que al usar únicamente los show o listados no necesitaremos en este caso crear un getmapping para cada una de las vistas ni su validación en el propio controlador.

Pasando al apartado de servicios, es la parte sin duda que mas contenido nuevo tenemos. Mayoritariamente muchas formas nuevas de validar o de bindear un objeto de cara al usuario o para la propia base de datos.

```
@Override
public void validate(final Request<Inventor> request, final Inventor entity, final Errors errors) {
    assert request != null;
    assert entity != null;
    assert errors != null;

    if (!errors.hasErrors("statement")) {
        errors.state(request, SpamDetector.error(entity.getStatement(), this.repository.getSystemConfiguration()), "statement")
    }

    if (!errors.hasErrors("company")) {
        errors.state(request, SpamDetector.error(entity.getCompany(), this.repository.getSystemConfiguration()), "company", "a")
    }
}

@Override
public void bind(final Request<Inventor> request, final Inventor entity, final Errors errors) {
    assert request != null;
    assert entity != null;
    assert errors != null;

    request.bind(entity, errors, "company", "statement", "link");
}

@Override
public void unbind(final Request<Inventor> request, final Inventor entity, final Model model) {
    assert request != null;
    assert entity != null;
    assert model != null;

    request.unbind(entity, model, "company", "statement", "link");
}
```

Haciendo hincapié en los métodos nuevos, los métodos bind, unbind y validate, son propicios a mejorar el rendimiento a la hora de escribir código ya que únicamente nos centramos en una tarea, como nos pasa en este caso con los CRUD.

Con respecto a el código de los show pasaría lo mismo, una nueva forma de crear un show o un list que agiliza todo el proceso tanto de búsqueda como de rendimiento.

```
@Autowired
protected AuthenticatedAnnouncementRepository repository;

@Override
public boolean authorise(final Request<Announcement> request) {
    assert request != null;

    return true;
}

@Override
public Announcement findOne(final Request<Announcement> request) {
    assert request != null;

    Announcement result;
    int id;

    id = request.getModel().getInteger("id");
    result = this.repository.findAnnouncementById(id);

    return result;
}

@Override
public void unbind(final Request<Announcement> request, final Announcement entity, final Model model) {
    assert request != null;
    assert entity != null;
    assert model != null;

    request.unbind(entity, model, "creation", "title", "body", "flag",
        "url");
}
```

5. Conclusión

Como conclusión cabe destacar que con la realización de este documento hemos conseguido tener una visión algo más profunda sobre las arquitecturas vistas en la asignatura de Diseño y Pruebas II, con lo que tenemos conocimiento para ver en que ámbitos de nuestra vida laboral tendremos una mayor capacidad para resolver los problemas que se nos plantean.

Además de todo esto, la enseñanza en el ámbito de la arquitectura WIS, nos ha enseñado a mejorar la ordenación, rendimiento y legibilidad para futuros proyectos. A si mismo el poder entender una forma mejor de estructurar todo para que nuestros compañeros futuros o actuales entiendan mejor el contexto en el que nos encontramos.

6. Bibliografía

Intencionadamente en blanco.