UNIVERSITAT POLITÈCNICA DE VALÈNCIA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Politécnica Superior de Gandia

Renovación de un framework para aplicaciones web de administración electrónica: desarrollo de una aplicación para la gestión del Cementerio Municipal de Cullera

Trabajo Fin de Grado

Grado en Tecnologías Interactivas

AUTOR/A: Borja Roca, Vicent

Tutor/a: Alberola Oltra, Juan Miguel

Cotutor/a: Sánchez Anguix, Víctor

Cotutor/a externo: OLIVER COMPANY, ALFREDO

CURSO ACADÉMICO: 2021/2022

Índex de continguts

1 INTRODUCCIÓ **5**

2 OBJECTIUS **6**

2.1 Objectiu principal 6

2.2 Objectius secundaris 6

3 FRAMEWORK DE DESENVOLUPAMENT **7**

3.1 Descripció 7

3.2 Ferramentes tecnològiques 9

3.2.1 Llenguatge de programació 9

3.2.2 Accés a dades 10

3.2.3 Tecnologies de la capa de presentació 12

3.3 Arquitectura del Framework 13

3.3.1 Lògica de negoci I: Accés a dades 14

3.3.2 Lògica de negoci II: Control d'accés 15

3.3.3 Lògica de negoci III: Generació de aplicacions 17

3.3.4 Context de la sessió 21

3.3.5 Missatges d'estat i idiomes 21

4 ACTUALITZACIÓ DEL FRAMEWORK **22**

4.1 Limitacions del framework 22

4.1.1 Tecnologies deprecades i obsoletes 22

4.1.2 Seguretat 23

4.1.3 Falta de documentació i accessibilitat a nous desenvolupadors 24

4.2 Metodologia de treball 24

4.2.1 Fases de desenvolupament 25

4.2.2 Entorns de desenvolupament 25

4.2.3 Gestor de base de dades pgAdmin 25

4.2.4 Tests automàtics amb Junit 26

4.3 Investigació de tecnologies per a l'actualització 27

4.3.1 Actualització de la capa de persistència 29

4.3.2 Actualització de la capa de presentació 29

4.3.3 Actualització control d'accés 30

4.3.4 Encriptació i seguretat 31

4.3.5 Servidor 32

4.4 Dissenys 33

4.5 Implementació 34

4.5.1 Capa de persistencia 35

4.5.2 Capa de presentació 35

4.5.3 Actualització Control d'Accés 39

4.5.4 Encriptació i seguretat 42
4.5.5 Documentació 43
4.6 Proves 44

5 Aplicació cementeri municipal 45
5.1 Requeriments 45
5.2 Disseny 48
5.3 Implementació 50
5.4 Manual per al desenvolupador 55

6 CONCLUSIONS 56

7 BIBLIOGRAFIA 57

Figura 1: *Representació dels principals actors en la infraestructura d'una aplicació web.*

Figura 2: *Representació del disseny de models d'aplicacions i com el framework els processa per a generar tota la infraestructura necessària per al seu funcionament.*

Figura 3*: Arbre d'herència de les propietats de la classe Base (Id i Description) en altres classes fill diferents.*

Figura 4*: Representació de la ubicació de la interfície accés a dades dins de la infraestructura del projecte.*

Figura 5*: Representació de l'esquema d'interacció entre Client-Servidor i el procés de presentació d'aplicacions web amb Java Server Faces.*

Figura 6*: Perfil de les classes de la lògica d'accés a dades.*

Figura 7*: Esquema del funcionament del mòdul de persistència de dades.*

Figura 8*: Esquema amb el perfil de les classes de l'estructura de control d'accés i les seves relacions.*

Figura 9*: Representació de la classe d'un model de aplicació i la interfície equivalent resultant.*

Figura 10*: Esquema mostrant els dos tipus de plantilles que es poden utilitzar dins la pàgina principal per a representar la secció del formulari.*

Figura 11*: Exemple d'una aplicació sobre gestió de vivendes, on es mostra el paquet de models d'aplicació i el paquet d'accions personalitzades que poden realitzar-se dins de l'aplicació.*

Figura 12*: Interfície per a gestionar les diferents bases de dades i visualitzar les dades que contenen.*

Figura 13*: Evolució de les especificacions de Java EE indicant la versió utilitzada en el framework abans i després de l'actualització.*

Figura 14*: Sintaxi d'un fitxer de text amb format Yaml.*

Figura 15*: Taula de compatibilitats de servidors amb Jakarta 9.1.*

Figura 16*: Classes del paquet DAO amb la nova inclusió de paràmetres genèrics.*

Figura 17*: Esquema de l'estructura de control realitzant el mapeig del fitxer yaml a objectes.*

Figura 18*: Perfil de la classe CipherUtils i els mètodes implementats per a realitzar funcions d'encriptació.*

***Figura 19**: Definició d'un mètode amb paràmetres genèrics.*

***Figura 20**: Definició d'una plantilla JSF dins d'un arxiu xhtml.*

***Figura 21**: Implementació de la plantilla layout.xhtml.*

***Figura 22**: Obtenció d'arxius de recursos amb l'etiqueta loadBundle.*

***Figura 23**: Utilització de les etiquetes per a la definició de Beans.*

***Figura 24**: Barra dinàmica d'estat de l'aplicació web generada pel framework.*

***Figura 25**: Utilització de les etiquetes per a la definició de Beans.*

***Figura 26**: Antiga gestió de l'estructura de control.*

***Figura 27**: Format de l'estructura de control dins del fitxer Yaml.*

***Figura 28**: Arbre d'elements del fitxer Yaml vist des de l'Outline d'Eclipse.*

***Figura 29**: Fitxer de credencials abans i després de l'encriptació.*

***Figura 30**: Visualització de la documentació generada amb JavaDocs.*

***Figura 31**: Visualització del resultat del test realitzat amb Junit.*

***Figura 32**: Mapa del cementeri.*

***Figura 33**: Diagrama de flux de la navegació de l'usuari en l'aplicació del cementeri.*

***Figura 34**: Esquema del model de dades de l'aplicació del cementeri.*

***Figura 35**: Paquet dels models de formulari per a l'aplicació del cementeri.*

***Figura 36**: Esquema del model de dades de l'aplicació del cementeri.*

***Figura 37**: Model de formulari per al programa de gestió de difunts.*

***Figura 38**: Introducció de les dades de l'estructura de control per a l'aplicació del cementeri.*

***Figura 39**: Introducció de dades al formulari de gestió de difunts.*

***Figura 40**: Visualització de les dades en donar d'alta un difunt.*

***Figura 41**: Estructura de control per a afegir una acció.*

***Figura 42**: Visualització de l'acció personalitzada de mostrar mapa.*

RESUM

Aquest Treball de Fi de Grau (TFG) descriu detalladament tot el treball desenvolupat per a l'actualització del framework *Openadmin,* un framework implementat per l'Ajuntament de Cullera l'any 2009, així com la creació d'una nova aplicació per a la gestió del Cementeri Municipal de Cullera. Breument, el framework és una plataforma de programació en Java per a desenvolupar aplicacions web administratives basades en formularis, que poden ser utilitzades per a realitzar les funcions de les distintes àrees de l'administració pública. Actualment, el framework de l'Ajuntament de Cullera s'utilitza per a crear i mantenir aplicacions de les que que fan ús els funcionaris de diversos departaments del consistori. Amb l'actualització s'ha realitzat una renovació de les tecnologies obsoletes per a solucionar els problemes de seguretat i de compatibilitat de la versió anterior i així poder adaptar aquesta plataforma a la resta de l'administració actual de l'Ajuntament.

ABSTRACT

In this Final Degree Project it's described in detail all the work developed for the update of the *Openadmin* framework, a framework implemented by the City Council of Cullera in 2009, as well as the creation of a new application for the management of the Cullera Municipal Cemetery. In brief, the framework is a Java programming platform for developing form based administrative web applications, which can be used to perform the functions of the different areas of public administration. Currently the framework is used to create and maintain applications that are used by officials from various departments of the city council. A renovation of the framework's obsolete technologies has been conducted to solve the security and compatibility problems of the previous version, then it will be possible to adapt this new platform to the rest of the current administration of the city council.

1 INTRODUCCIÓ

Aquest Treball Final de Grau (TFG) comença arrel de les pràctiques curriculars realitzades des de setembre de 2021 a l'Ajuntament de Cullera. L'objectiu principal d'aquestes pràctiques era renovar i actualitzar la plataforma *openadmin,* un framework que s'havia desenvolupat l'any 2009 i que permet realitzar diferents aplicacions de gestió per a l'administració pública municipal.

El framework, que s'havia mantingut fins eixe moment, havia sigut utilitzat per a desenvolupar aplicacions destinades a la millora de les tasques administratives en diferents departaments de l'Ajuntament: plusvàlues, autoliquidacions, permisos, nòmines, etc. En aquest sentit, el cap del Departament d'Informàtica de l'Ajuntament de Cullera va transmetre la necessitat de fer aquesta actualització del framework per a dur endavant el nou repte que l'Ajuntament de Cullera acabava d'emprendre: implantar l'administració electrònica a tots els nivells en el consistori.

Malgrat que es porta més d'una dècada parlant de la modernització i la digitalització de l'Administració pública, l'anomenada "cultura del paper" continua instal·lada en els seus diversos estaments (Ruiz, 2022), sobretot a nivell dels municipis, ja que l'administració local és la que menys recursos disposa per aconseguir la total implantació de l'administració electrònica.

Una de les principals característiques d'aquest projecte és que totes les tecnologies utilitzades per al seu desenvolupament són de software lliure. Per tant, aquest projecte desenvolupat a l'Ajuntament de Cullera pot servir d'alternativa gratuïta a aquelles administracions locals que projecten implementar l'administració digital a la seva organització. Per contra, les empreses dedicades a la venda d'aquests tipus de sistemes utilitzen frameworks que contenen programes amb llicències de pagament, que fan que el preu d'instal·lar l'administració digital siga molt elevat, ja que no només hi ha que pagar pel servici, sinó també per totes les llicències dels programes que utilitza el sistema.

Respecte al projecte d'actualització del framework *Openadmin*, el Departament d'Informàtica necessitava implantar les tecnologies actuals de Java al framework per aconseguir més seguretat, més funcionalitat, més compatibilitat amb altres sistemes i també ser més intuïtiu de cara a nous desenvolupadors col·laboradors d'aquest projecte. Amb la nova versió del framework, es durà a terme el desenvolupament de noves aplicacions que contribuiran a completar el procés d'implantació de l'administració digital en l'Ajuntament de Cullera.

Així doncs, en la reunió inicial amb el màxim responsable del Departament d'Informàtica de l'Ajuntament de Cullera, es va posar de manifest que aquest projecte tenia les seves dificultats, ja que no debades, aquesta tasca havia sigut proposada anteriorment sense èxit a diferents estudiants de pràctiques d'Enginyeria Informàtica que havien passat per l'Ajuntament.

Tot i que el projecte d'actualització del framework *Openadmin* tenia més complexitat, pel que respecta a la programació, que els projectes software que es desenvolupen al llarg dels cursos del Grau en Tecnologies Interactives de la Universitat Politècnica de València (UPV),

aquest repte va ser acceptat ja que es tenien les capacitats suficients pels coneixements i habilitats adquirides al llarg de quatre anys al Campus de Gandia de l'UPV.

2 OBJECTIUS

En aquest apartat es defineixen els objectius de la feina plantejada a aquest treball i es descriu la metodologia de treball utilitzada per a dur a terme les diferents activitats del desenvolupament.

2.1 Objectiu principal

L'objectiu d'aquest treball es defineix en dos desenvolupaments concrets. El primer, consisteix en renovar el framework *Openadmin* desenvolupat en 2009 al Departament d'informàtica de l'Ajuntament de Cullera. L'actualització consisteix en corregir alguns dels problemes que ja residien en la versió anterior, a més d'investigar i implementar les tecnologies que millor s'adapten al projecte hui en dia, en 2022. L'altre objectiu és desenvolupar, amb aquest framework ja renovat, una de les aplicacions de gestió de l'ajuntament fent ús de les noves funcionalitats de l'actualització. Aquesta nova aplicació servirà per provar l'actualització del framework i també serà utilitzada com a referència per a futurs desenvolupaments.

2.2 Objectius secundaris

Aquest treball es descompon en els següents objectius secundaris:

- Analitzar les limitacions del framework *Openadmin* pel que fa a les seves capacitats i manteniment.
- Definir quins són els requeriments necessaris per a donar solució a aquestes limitacions.
- Investigar quines tecnologies poden donar suport a l'actualització.
- Realitzar una proposta de disseny, d'implementació i de validació de la renovació del framework per a complir amb els requisits definits.
- Una vegada s'ha completat l'actualització del framework *Openadmin*, definir els requisits de l'aplicació de gestió del cementeri municipal i realitzar el seu desenvolupament.

3 FRAMEWORK OPENADMIN

A continuació, es descriuen les característiques principals i les capacitats del framework *Openadmin* en la seva versió de 2009 i les ferramentes tecnològiques que es van utilitzar per al seu desenvolupament. En primer lloc, es descriu què aporta el framework a l'administració de l'Ajuntament i com interactuen els tècnics amb aquesta ferramenta. Després, es llisten les tecnologies utilitzades per a desenvolupar les principals funcionalitats del projecte. Finalment, es presenta l'arquitectura del framework, mostrant els detalls tècnics del seu desenvolupament.

3.1 Descripció

La fortalesa principal del framework és la seua capacitat de generar, de forma automàtica, aplicacions web per cadascun dels departaments, estalviant temps de desenvolupament. Per a entendre els avantatges que ofereix la utilització d'aquesta ferramenta, s'expliquen a continuació, a grans trets, els aspectes comuns de les aplicacions que es poden desenvolupar fent ús d'aquest framework.

En el context d'una Administració pública, com puga ser un Ajuntament, existeixen diverses aplicacions de gestió que estan basades en formularis que permeten realitzar les accions CRUD (acrònim de l'anglès Crear, Llegir, Actualitzar, Borrar), de manera que la informació introduïda i modificada per l'usuari és guardada en una base de dades.

La comunicació entre el client i la base de dades és possible gràcies a la lògica del servidor, que s'encarrega d'identificar a l'usuari i obrir una sessió de treball, durant la qual fa d'intermediari entre l'aplicació i la informació de la base de dades, aconseguint satisfer totes les accions de CRUD que vaja realitzant l'usuari dins dels formularis. A la figura 1 es mostra l'esquema d'interacció entre l'usuari, l'aplicació i les diferents parts de l'estructura del framework.

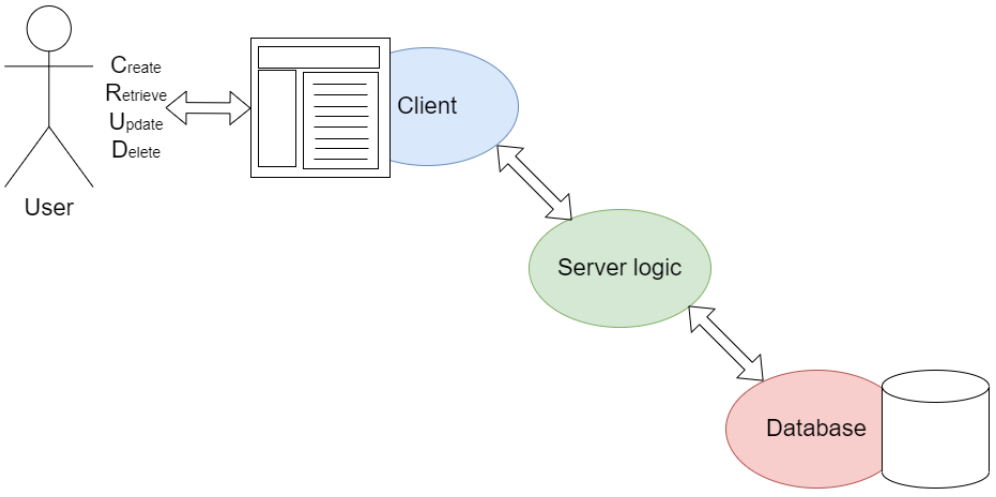


***Figura 1**: Representació dels principals actors en la infraestructura d'una aplicació web. Font: elaboració pròpia.*

Si es volgueren desenvolupar vàries aplicacions de funcionalitat similar però que tractaren temàtiques diferents, com és al cas de l'Ajuntament i els seus distints departaments, per a cada departament hauria de desenvolupar-se tant la seva part de client com la part de lògica de servidor i base de dades. Depenent de la quantitat de departaments, i també de la complexitat de les dades i operacions que siguen necessàries realitzar-se dins de la seva aplicació, el cost de desenvolupament i de manteniment d'aquestes aplicacions s'elevaria considerablement, requerint-se de més temps o de més tècnics.

Per tant, el que es busca amb el framework *Openadmin* és aprofitar les similituds d'aquestes aplicacions per a automatitzar gran part del desenvolupament, tant per a la part de client com la de lògica i base de dades.

D'aquesta manera, la feina del desenvolupador que treballa amb el framework *Openadmin* és la de crear models d'aplicacions, els quals tenen la informació necessària perquè el framework puga generar l'aplicació junt amb la infraestructura necessària per a la comunicació amb la base de dades.

L'avantatge de treballar sobre aquests models és que requereixen un menor temps de desenvolupament, estalviant tasques repetitives que haurien de realitzar-se per a cada aplicació. A més a més, és més accessible per als nous tècnics començar a desenvolupar aplicacions aprenent a utilitzar aquests models, ja que d'una altra manera, abans haurien de conèixer el funcionament de tota la infraestructura. A la figura 2 es representa la implementació d'un model d'aplicació per a generar la infraestructura del framework.

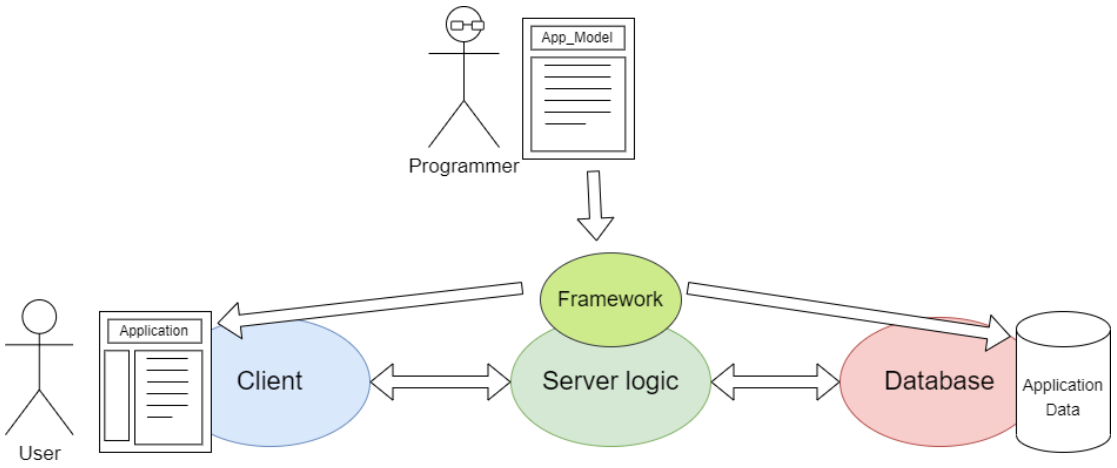


***Figura 2**: Representació del disseny de models d'aplicacions i com el framework els processa per a generar tota la infraestructura necessària per al seu funcionament. Font: elaboració pròpia.*

Així doncs, amb aquest paradigma de treball es busca que el conjunt d'aplicacions de gestió d'una organització siga més escalable i més fàcil de mantenir.

3.2 Ferramentes tecnològiques

A continuació es mostra un resum de les especificacions del projecte, llistant les tecnologies software que s'han utilitzat per a desenvolupar les principals funcionalitats del framework *Openadmin*. Amb l'actualització del framework, les tecnologies llistades a continuació han sigut renovades a una nova versió, o en cas de no existir una versió nova, han sigut substituïdes per una altra alternativa.

3.2.1 Llenguatge de programació

L'entorn Java oferix suport per a executar aplicacions web, a més de poder desenvolupar el codi necessari per a les funcionalitats del framework dins de la lògica de negoci.

En este sentit, un dels trets a destacar d'aquest llenguatge, que ha sigut profitós per al desenvolupament del framework, és el *polimorfisme*. Segons definix Luaces (2012), dins de la programació orientada a objectes, el polimorfisme és la capacitat que té un objecte de tindre formes diferents. En Java, una classe pot fer referència a més d'un tipus de classes.

En les classes de la lògica de negoci del framework s'utilitza un polimorfisme per assignació, és a dir, una variable d'un tipus pot adoptar una altra forma sempre i quan existisca una relació d'herència o implementació. A la figura 3 es pot observar l'existència de tres classes diferents (*User, Role* i *Document*) que tenen en comú les propietats heretades de la classe *Base*.

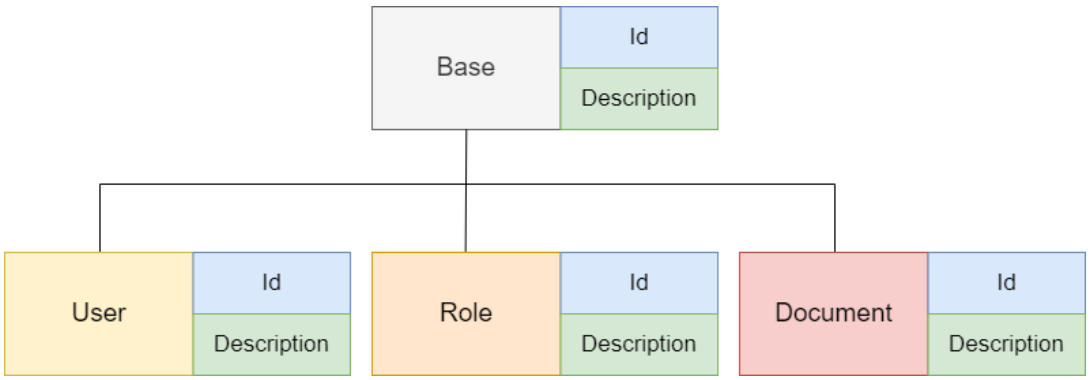


***Figura 3**: Arbre d'herència de les propietats de la classe Base (Id i Description) en altres classes fill diferents. Font: elaboració pròpia.*

Aquesta capacitat li permet al framework la flexibilitat de reconèixer i gestionar molts objectes diferents, sempre que aquests hereten les qualitats d'un objecte *Base*. D'aquesta manera, la lògica pot llegir models diferents d'aplicació i generar automàticament la infraestructura necessària per al seu funcionament.

La lògica del framework s'ha definit segons les especificacions del marc de desenvolupament Java EE[1], un estàndard on s'indiquen una sèrie de normes i pautes de programació per a la creació d'aplicacions empresarials basades en Java.

3.2.2 Accés a dades

La lògica del framework s'encarrega de llegir els models d'aplicació i generar la seua infraestructura. Una vegada s'han generat les aplicacions i estan en funcionament, la lògica s'encarregarà d'identificar els usuaris i de satisfer totes les accions CRUD que van realitzant-se des de les aplicacions de tipus client.

Per a poder complir aquestes funcionalitats, és necessari establir una comunicació amb una base de dades. Per tant, es requereix un sistema de base de dades i uns mecanismes per a poder connectar-se.

En aquest sentit, la comunicació entre la lògica i la base de dades està implementada utilitzant el patró de desenvolupament DAO (Data Acces Object), mitjançant la definició d'un objecte que serveix d'interfície per a separar el codi de la lògica amb el codi necessari per a la persistència d'objectes en la base de dades (Garrido, 2016).

Cal tenir en compte que implementar les funcions d'accés a dades dins de la lógica pot ser problemàtic. En cas que fóra necessari utilitzar un sistema de base de dades diferent, s'hauria d'adaptar la lògica per a aquest sistema, o si es requerira utilitzar diversos sistemes diferents, caldria adaptar-se per a cadascun d'ells. Per aconseguir que el framework siga el més flexible possible i puga aplicar-se a diverses organitzacions sense dependre d'un sistema de base de dades concret, la lògica es treballa seguint el patró DAO.

Breument, el patró DAO consisteix en crear una interfície software on s'implementen totes les funcions necessàries per a la persistència d'objectes.

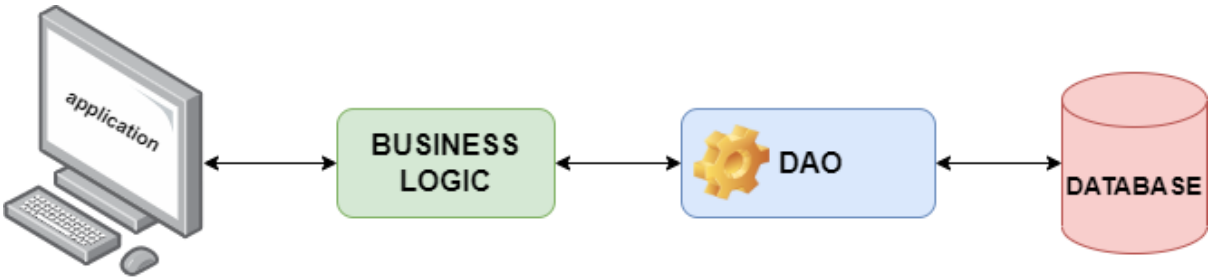


***Figura 4**: Representació de la ubicació de la interfície accés a dades dins de la infraestructura del projecte. Font: elaboració pròpia.*

[1] https://www.oracle.com/java/technologies/javaee/javaeetechnologies.html

Com podem veure a la figura 4, la interfície DAO serveix com una passarel·la per a transferir els objectes de la capa del negoci als objectes del sistema de base de dades.

Per tant, des de la capa de la lògica de negoci no es necessita saber els detalls de com s'accedeix a les dades o amb quin sistema de base de dades es treballa. Quan es necessite fer la persistència, la lògica utilitzarà aquesta interfície, la qual pot adaptar-se per a treballar amb múltiples sistemes bases de dades.

Així, la passarel·la DAO s'ha implementat dins del framework segons les especificacions del API de persistència de *Java EE,* conegut com JPA (Java Persistence Api), on s'indiquen les especificacions per a definir la persistència d'objectes Java dins d'una base de dades relacional, conegut com el mecanisme mapeig objecte-relacional (Pech et al., 2012, p 5).

Per a configurar el framework de manera que puga comunicar-se amb els sistemes de base de dades, s'utilitza la ferramenta Hibernate[2], que és una implementació de JPA que conté suport per a treballar amb els diferents sistemes de base de dades existents (per exemple, PostgreSQL[3], MySQL[4], etc). Amb aquesta ferramenta, es configura el framework de manera que s'aconsegueix una traducció del món orientat a objectes de la lògica de negoci amb el dialecte del sistema de taules relacionals que hi ha a la base de dades.

3.2.3 Tecnologies de la capa de presentació

La part de la lògica encarregada de generar les interfícies web també es defineix basant-se en les especificacions de *Java EE*, en concret, el marc de desenvolupament web JSF (Java Server Faces). En aquest sentit, el desenvolupament d'aplicacions web amb JSF està enfocat al servidor, on les pàgines es defineixen en format XHTML mitjançant un llenguatge de components. En la figura 5 podem observar el procés de presentació de pàgines JSF dins d'un servidor d'aplicacions web.

[2] https://hibernate.org/
[3] https://www.postgresql.org/
[4] https://www.mysql.com/

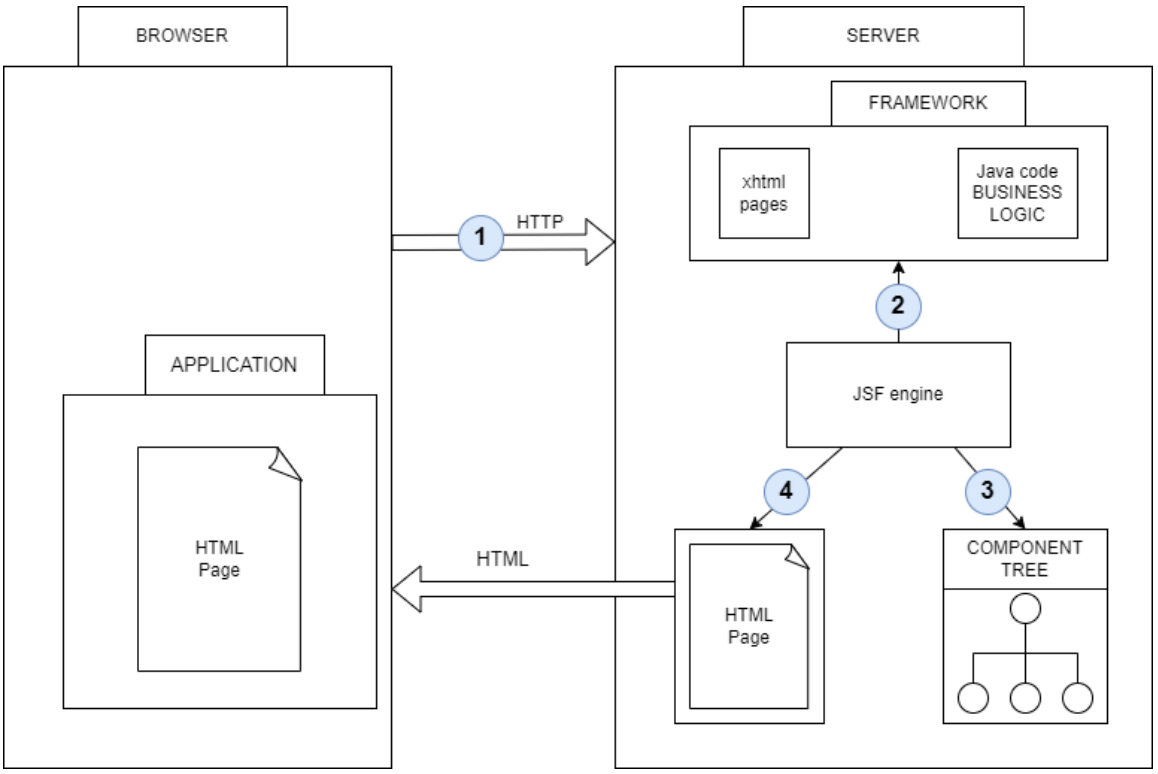


***Figura 5**: Representació de l'esquema d'interacció entre Client-Servidor i el procés de presentació d'aplicacions web amb Java Server Faces. Font: elaboració pròpia.*

Quan es realitza una petició al servidor per a obtindre una pàgina, el motor de JSF construeix una rèplica de l'estructura de la pàgina original com un arbre de components Java. Després, a partir d'aquest arbre es genera la pàgina HTML que serà representada al navegador.

Utilitzar aquest enfocament de desenvolupament basat en servidor simplifica les tasques de la lògica de negoci, ja que la definició de la interfície d'usuari i les dades de la lògica es troben al mateix lloc. Gràcies a l'arbre de components Java que genera el motor JSF, l'estructura de la pàgina pot modificar-se de manera programàtica a través de la lògica, aconseguint la generació automàtica d'aplicacions i permetent una navegació dinàmica.

Per a desenvolupar els formularis de les aplicacions i les seves funcionalitats, s'ha utilitzat la col·lecció de components *Richfaces*[5], la qual implementa el marc de JSF per a la definició dels components. Cada component conté una funcionalitat única (per exemple, un menú, un camp de text, un calendari per escollir una data, etc).

3.3 Arquitectura del Framework

Per entendre el funcionament del framework *Openadmin*, en aquest apartat es presenten els dissenys dels diferents mòduls del framework. Es descriuen les principals funcionalitats,

[5] https://docs.jboss.org/richfaces/latest_3_3_X/en/devguide/html/ArchitectureOverview.html

la seva arquitectura, tecnologies utilitzades i alguns aspectes del procés de la implementació.

3.3.1 Lògica de negoci I: Accés a dades

En aquest apartat s'exposa l'estructura i el funcionament del mòdul dedicat a la persistència dels objectes de la lògica. Com s'ha comentat anteriorment a l'apartat 3.2.2, la capa de persistència està definida segons les especificacions de JPA.

Per a establir la connexió amb la base de dades s'utilitza un dels objectes de JPA *EntityManager*, el qual s'encarrega d'obrir i mantenir la connexió amb la base de dades i de gestionar totes les transaccions.

En la figura 6 es representen els dissenys de les dues classes del paquet de persistència, ConnectionDao on s'implementa la connexió i DaoJpaHibernate la interfície amb els mètodes d'accés a dades.

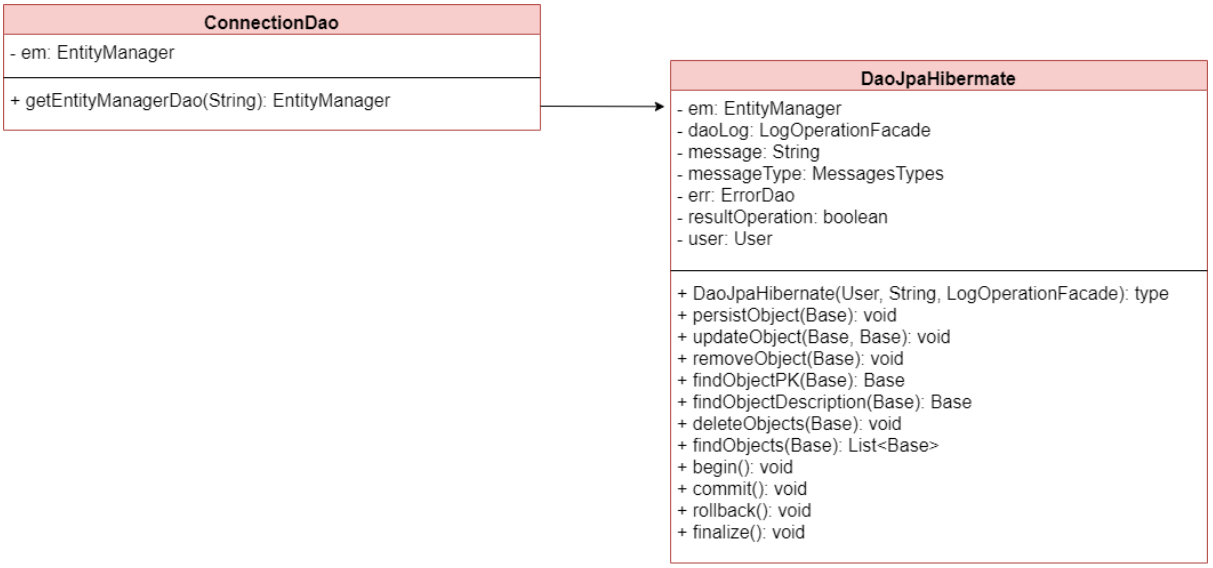


***Figura 6**: Perfil de les classes de la lògica d'accés a dades. Font: elaboració pròpia.*

Els mètodes DAO de la classe DaoJpaHibernate reben com a paràmetre objectes de tipus Base, per a aprofitar la flexibilitat del polimorfisme. Sempre que els objectes de la lògica hereten de la classe Base, podran ser persistits amb aquesta interfície DAO, ja que els mètodes estan definits de manera que les operacions es fan en funció dels paràmetres de Base (*Id* i *Description*). D'aquesta manera, s'aconsegueix delegar totes les accions de persistència a una sola interfície, evitant el fet d'haver de crear una interfície DAO per a cada tipus d'objecte. Amb una sola interfície s'aconsegueixen el control i la flexibilitat necessaris per a realitzar les tasques del framework.

La classe *ConnectionDao* té un únic mètode, el qual s'encarrega de realitzar tots els passos necessaris per a generar l'objecte de connexió *EntityManager*. Per a poder generar-lo, s'han de definir els detalls de la connexió en un fitxer de configuració. Mitjançant les anotacions de

la llibreria Hibernate es pot indicar el tipus de base de dades i el dialecte de les transaccions. La configuració és processada per l'objecte *EntityManagerFactory*, que llegirà aquest fitxer de configuració per a fabricar l'objecte de connexió, el qual es podrà obtindre cridant al seu mètode *createEntityManager()*.

A la figura 7 es representa l'estructura del mòdul de persistència, on es pot observar el procés de generació de l'objecte EntityManager, que serà utilitzat dins la interfície *DaoJpaHibernate* per a implementar els mètodes de persistencia d'objectes.

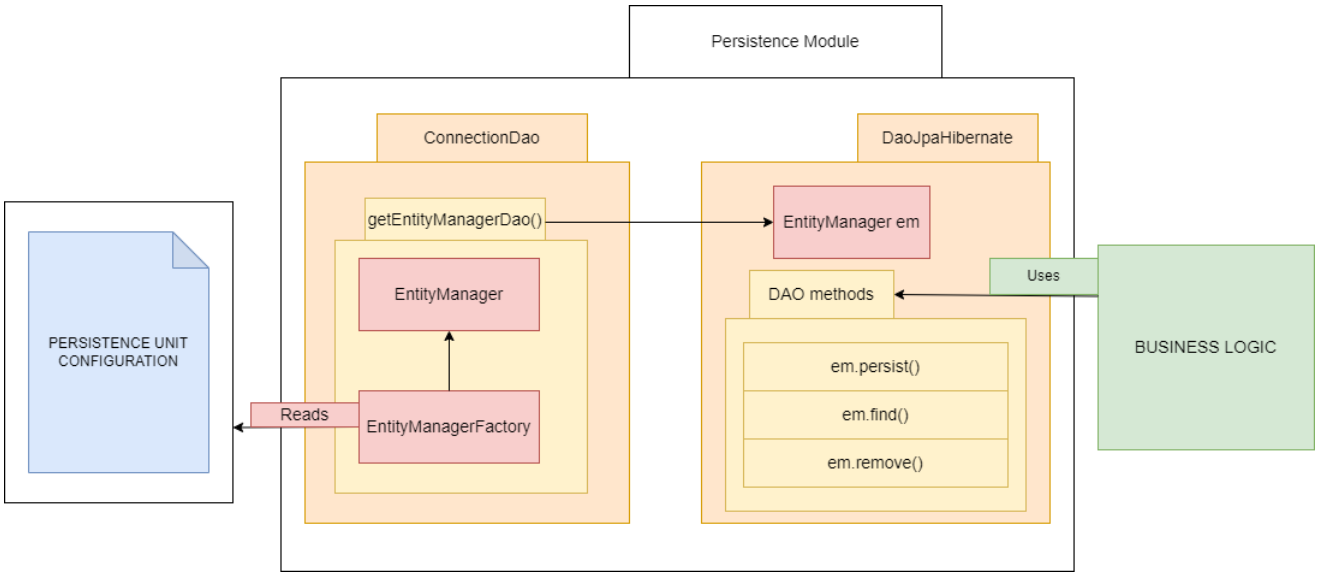


***Figura 7**: Esquema del funcionament del mòdul de persistència de dades. Font: elaboració pròpia.*

3.3.2 Lògica de negoci II: Control d'accés

El mòdul de control d'accés té la funció d'inicialitzar la infraestructura necessària per a guardar la informació de tots els usuaris i privilegis, de manera que el framework pot associar a quines aplicacions i continguts poden accedir. A la figura 8 està representat el disseny de les diferents classes de l'estructura de control i les relacions que existeixen entre elles.

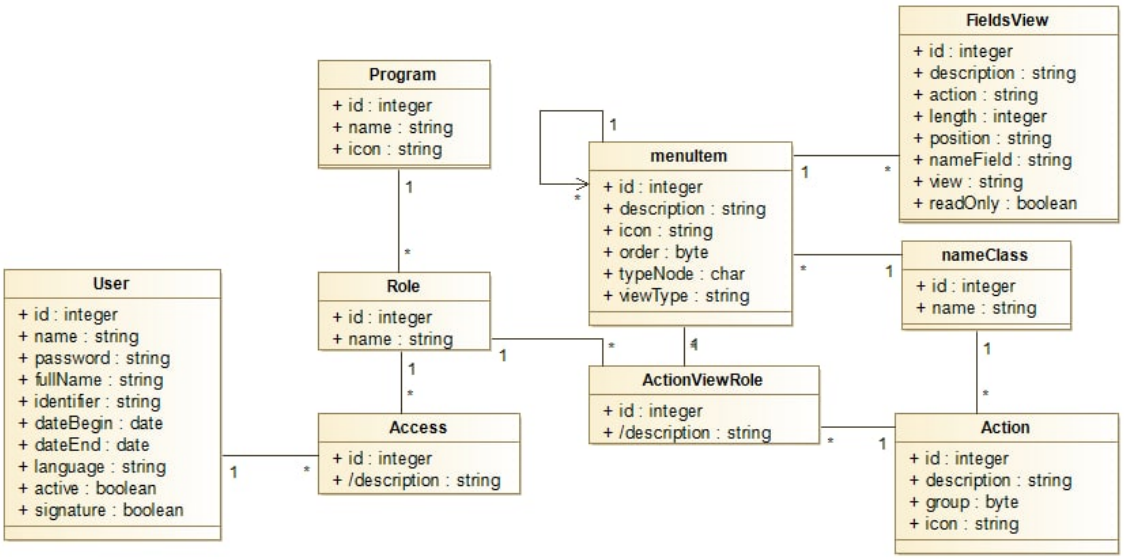


***Figura 8**: Esquema amb el perfil de les classes de l'estructura de control d'accés i les seves relacions. Font: elaboració pròpia.*

A continuació, es descriuen breument cadascuna de les classes que conformen l'estructura de control. Com pot observar-se a la figura, cadascun dels objectes conté els atributs *id* i *description* heretats de la classe *Base*, ja que els propis elements de control també són gestionats i persistits per la lògica:

- **User**: Informació de l'usuari. Conté els detalls sobre la seua identificació.

- **Program**: Fa referència a cadascuna de les aplicacions desenvolupades, corresponent amb les funcions dels distints departaments de l'organització (per exemple, gestió de multes, gestió de pressupostos, control urbanístic).

- **Role**: Fa referència als diferents rols que poden exercir els usuaris dins dels programes. Aquesta classe delimita a quin tipus de continguts pot accedir l'usuari per a un programa determinat.

- **Access**: Indica l'accés que té cada usuari a una aplicació. Serveix de taula relacional entre User, Role i Program.

- **MenuItem**: Fa referència a les diferents pantalles de la navegació i els seus continguts. Té relació autoreferencial, ja que és possible crear continguts dins de continguts.

- **Action**: Les diferents accions que poden realitzar-se dins de les aplicacions, com les CRUD o també altres accions personalitzades.

- **ActionViewRole**: Indica les accions a les que té accés un rol en un formulari, i serveix de taula relacional entre Role, MenuItem i Action.

- **NameClass**: Indica la ubicació i el nom del model de l'aplicació dins de l'estructura de paquets del projecte (per exemple, per a la gestió de zones del cementeri, openadmin.model.cementeri.Zona).

La informació per a l'estructura de control pot definir-se prèviament a la càrrega del framework, com per exemple dins d'un fitxer de text. D'aquesta manera, el framework pot processar la seua informació i transformar-la en instàncies de les classes de control per a ser persistides en la base de dades. La classe *InitDataLoad* del mòdul s'encarrega de realitzar aquesta funció.

Cada vegada que es desenvolupa una aplicació nova, cal incloure-la a l'estructura de control d'accés perquè puga ser interpretada pel framework. Una vegada dins, podrà configurar-se la relació que té eixa aplicació amb la resta d'elements de la infraestructura, com els rols necessaris per a accedir a ella o les diferents accions que es poden realitzar dins la seva interfície.

3.3.3 Lògica de negoci III: Generació d'aplicacions

El mòdul desenvolupat per a la capa de presentació s'encarrega de llegir els models de les aplicacions per a construir les interfícies. També conté les definicions de tots els components web que s'utilitzaran, adaptant-los per a poder funcionar amb les dades de la lògica.

Els models d'aplicació, que seran interpretats pel framework, són classes Java, on cada atribut de la classe fa referència a cadascun dels elements que contindrà el formulari. El tipus de cada atribut indica el tipus d'informació que rebrà el seu respectiu element en el formulari. Per exemple, un atribut de tipus *String* serà interpretat com a un camp de text, mentre que per a un de tipus *Date* serà generat un component per a seleccionar una data d'un calendari. A la figura 9 es representa l'aspecte de la interfície generada a partir d'un model definit en una classe Java.

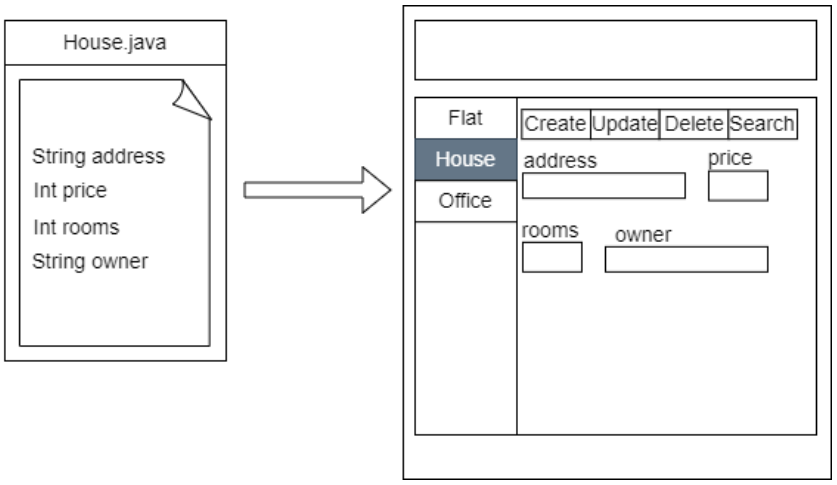


***Figura 9**: Representació de la classe d'un model d'aplicació i la interfície equivalent resultant.*
Font: elaboració pròpia.

Els diferents tipus d'elements que poden representar-se al formulari estan definits en el paquet de components web *openadmin.widgets.jsf.component*. En aquest paquet es troba una col·lecció de classes on s'implementen els components de la llibreria *Richfaces*, adaptant-los per funcionar amb la lògica.

La lògica representa automàticament els elements del formulari combinant la utilització de plantilles amb els mecanismes de JSF per a omplir l'arbre de components. Les plantilles són classes Java que permeten definir l'estructura de la pàgina i les funcions de navegació.

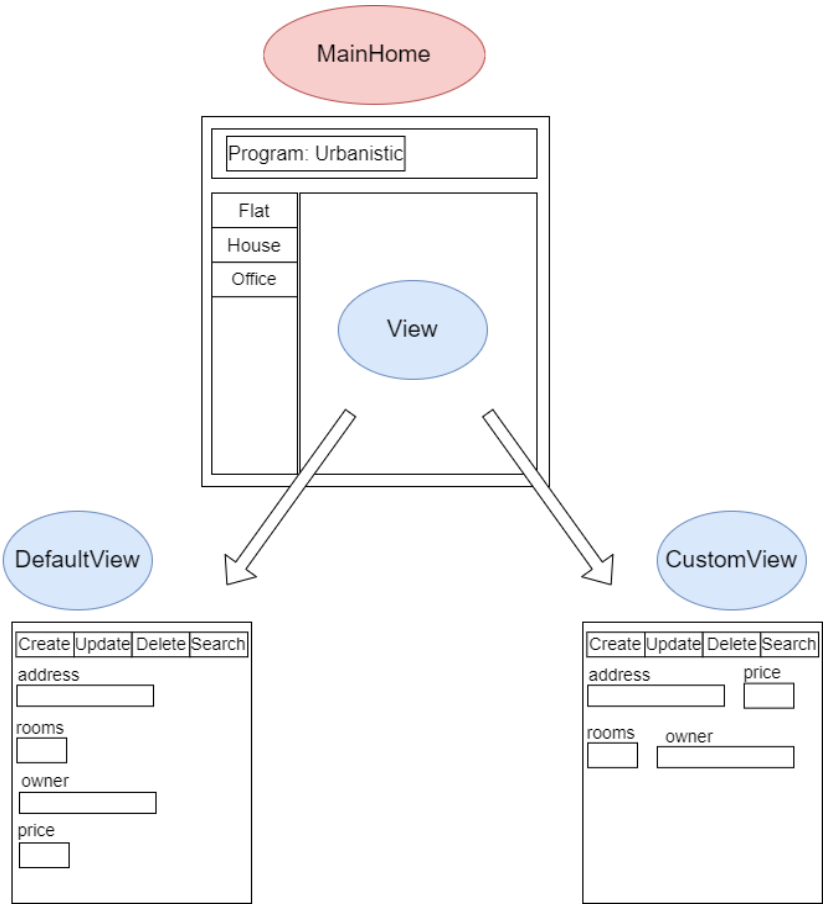


***Figura 10**: Esquema mostrant els dos tipus de plantilles que es poden utilitzar dins la pàgina principal per a representar la secció del formulari. Font: elaboració pròpia.*

En la figura 10 es mostren els diferents tipus de plantilla. La *MainHome* és la plantilla base de tota la web que conté la selecció de programa i la selecció de formulari. L'altre tipus de plantilla és la que representa al formulari, on podem trobar dues variacions:

- **DefaultView**: No requereix configuració. Cada element es mostra en sèrie de dalt a baix.

- **CustomView**: Poden col·locar-se els elements lliurement de forma matricial. Cal indicar la posició de cada element en l'estructura de control.

Per a cada aplicació hi ha un conjunt d'accions relacionades amb la informació tractada al formulari, les quals es representen amb objectes *Action* definides en l'estructura de control. Cada aplicació generada pel framework té, per defecte, la funcionalitat de les quatre accions CRUD per a persistir la informació del formulari. Les accions estan situades dalt del formulari en una barra de ferramentes, amb quatre botons: Alta, Modifica, Esborrar, Buscar.
En cas de la necessitat d'afegir una acció més personalitzada diferent de les accions CRUD, pot desenvolupar-se la seva funcionalitat en una classe dins d'un paquet d'accions creat específicament per al programa, com es mostra a la figura 11 amb el desenvolupament de dues accions personalitzades: *MapAction* per a mostrar un mapa i *BudgetAction* per a calcular un pressupost.

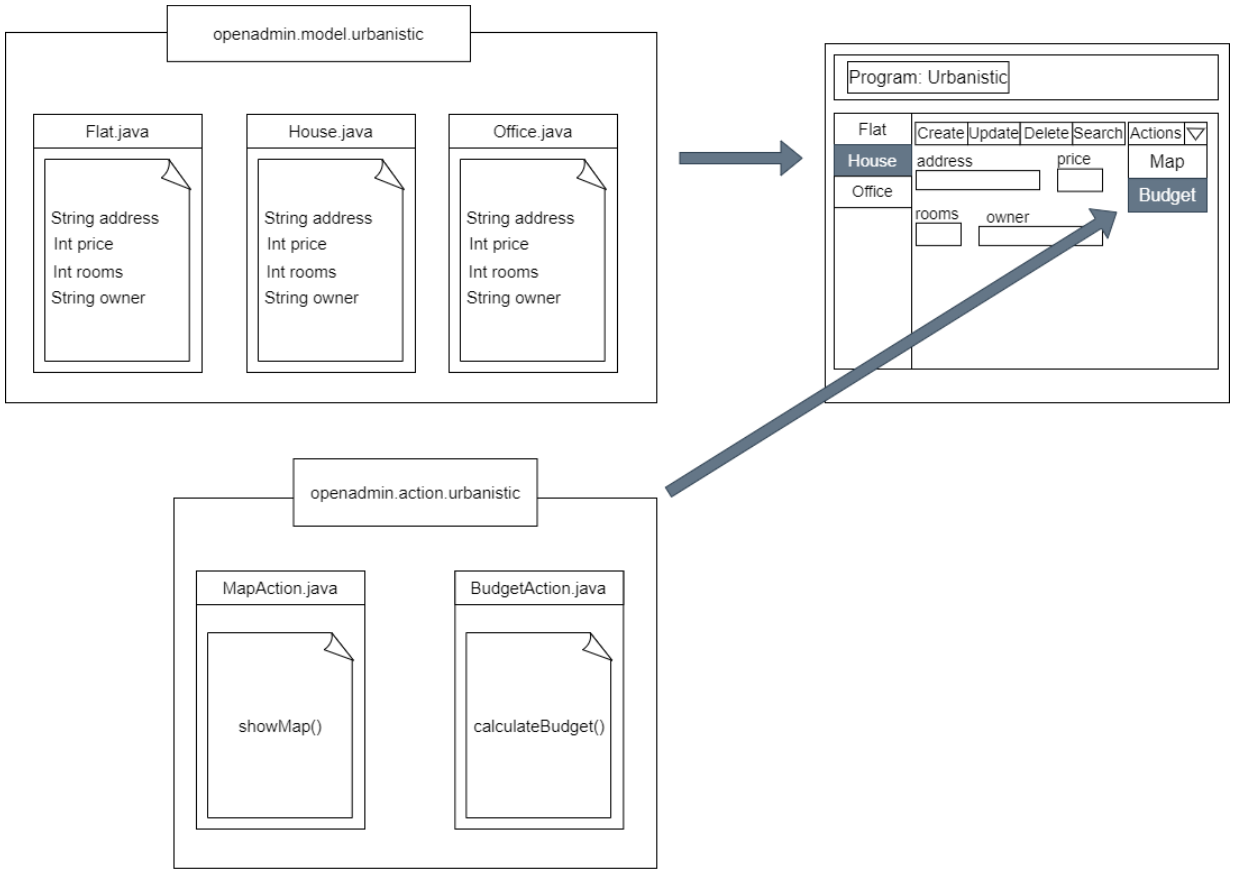


***Figura 11**: Exemple d'una aplicació sobre gestió de vivendes, on es mostra el paquet de models d'aplicació i el paquet d'accions personalitzades que poden realitzar-se dins de l'aplicació. Font: elaboració pròpia.*

Tant les aplicacions com les accions, s'han d'incloure en l'estructura de control d'accés per a poder definir quins usuaris poden accedir, i una vegada dins, s'ha d'especificar quines accions poden realitzar. Un usuari amb un rol de més autoritat podrà realitzar totes les accions CRUD, mentre que per a un rol amb un accés menys privilegiat de només consulta, únicament tindrà l'acció de Buscar.

3.3.4 Context de la sessió

Quan un usuari s'identifica correctament i accedeix a la pàgina principal, tota la seva informació referent a l'estructura de control (Programes, Rols, Accions, Vistes) és guardada en un context utilitzant els *Beans* de JSF.

Els *Beans* són objectes *Java* que contenten la lógica de negoci de les aplicacions, són instanciats al servidor i les seves dades poden ser modificades per les accions de les aplicacions *Client* (Pech et al., 2012, p. 6). La informació d'aquests objectes pot ser configurada per a ser accessible durant tot el temps que dure la sessió de treball de l'usuari.

El *Bean,* on es guarda la informació del context de l'usuari i de l'aplicació, està definit dins la classe *ContextAction*. En identificar-se l'usuari, tota la seua informació de control d'accés (programes, rols i accions) es guarden dins d'aquest Bean. La lògica accedeix a la informació *ContextAction* per a saber quins programes carregar per a aquest usuari i amb quin ordre s'han de representar en funció de la navegació que va realitzant.

3.3.5 Missatges d'estat i idiomes

A continuació, es descriuen els diferents fitxers que utilitza el framework per aconseguir textos multi idioma. Al llenguatge que utilitza cada usuari es pot accedir des de la classe de control d'accés "Usuari", de manera que cada vegada que s'identifica, tots els continguts de les aplicacions se li mostraran en el llenguatge que tinga configurat.

El nom dels fitxers finalitza amb l'abreviatura de l'idioma al qual pertany, per exemple "_es" per a l'espanyol i "_ca" per a català. Aquests són els fitxers de llenguatge de les aplicacions diferenciats per al tipus de contingut:

- **messagesOperation_es**: Són els continguts relacionats amb les accions personalitzades.
- **messagesMain_es**: Continguts que apareixen en la pàgina principal i la pàgina d'identificació, abans d'entrar a un programa.
- **labels_es**: La traducció de cadascun dels elements que té cada formulari.
- **errorsDao_es**: Missatges d'error durant la utilització dels formularis.
- **operationDao_es**: Missatges d'operació CRUD satisfactòria.

A la classe *MessatgesTypes* de la lògica, estan implementats els mètodes per accedir a la informació d'aquests fitxers. Quan s'introdueix una nova aplicació al framework, ha d'afegir-se la nova informació als fitxers d'idiomes.

4 ACTUALITZACIÓ DEL FRAMEWORK

Una vegada s'han descrit les principals característiques del framework *Openadmin* per desenvolupar aplicacions, en aquesta secció s'exposen les limitacions que requereixen d'una actualització del framework per donar-li continuïtat. A continuació, s'exposa la investigació realitzada sobre les noves tecnologies a implementar per a solventar aquestes limitacions. Finalment, es detalla el procés d'implementació per a cadascuna, mostrant els tests automàtics realitzats per a confirmar el seu correcte funcionament.

Cal remarcar que l'objectiu de l'actualització és migrar les funcionalitats de l'anterior versió del framework a tecnologies més recents, que aporten més funcionalitats, seguretat i que donen garantia de continuïtat per mantindre el projecte.

4.1 Limitacions del framework Openadmin

En aquest apartat es descriuen les limitacions principals que requereixen d'una actualització del framework, mostrant les carències que presenta la versió del framework que ha sigut descrita anteriorment.

4.1.1 Tecnologies deprecades i obsoletes

Un dels motius principals de l'actualització és el fet que el framework segueix utilitzant les mateixes tecnologies des del seu desenvolupament inicial en 2009. Per això, moltes de les tecnologies utilitzades estan deprecades i ja no tenen manteniment. Açò suposa un clar inconvenient de cara a la continuïtat del framework i de les ferramentes que es desenvolupen, ja que el software que no es manté és més propens a ser explotat degut a vulnerabilitats de seguretat. Hui en dia hi ha alternatives més eficients tant a l'hora de simplificar el desenvolupament com per al mateix funcionament de les aplicacions.

Les aplicacions administratives que han estat utilitzant-se en l'antiga versió del framework no poden combinar-se amb la resta de l'administració implantada a l'Ajuntament, on predomina la utilització de micro-servicis per a intercanviar informació a través de la web. Les llibreries antigues del framework contenen molts problemes de vulnerabilitats que fan aquesta unió de sistemes impossible.

A més a més, com s'utilitza una versió antiga de Java, no pot migrar-se el framework a una versió nova de servidor més segura.

També, les versions antigues tenen menys funcionalitats pel que fa a la part de la interfície web, mentre que les noves disposen de nous components que poden millorar l'experiència dels treballadors que utilitzen les aplicacions administratives.

Per açò és necessari investigar no només quines noves alternatives s'adapten millor al projecte, sinó també, quines tecnologies aportaran més funcionalitats al framework en comparació amb l'anterior versió.

A continuació es mostra un llistat de totes les tecnologies obsoletes que tenen problemes de vulnerabilitats i compatibilitat que fan necessària la seua renovació:

- **Jboss Seam**: Aquesta dependència utilitzada per a definir el context de les aplicacions s'ha quedat obsoleta per l'arribada al cicle de fi de vida de les ferramentes Red Hat Web Framework Kit en 2015, a causa del sorgiment de nous estàndards per al desenvolupament web.

- **RichFaces**: Una de les llibreries de components més utilitzades en el seu moment per a aplicacions amb Java Server Faces. Segons la recerca de Wulftange (2018), s'ha demostrat que totes les seves versions contenen vulnerabilitat. Arribà al cicle de fi de vida a principis de 2016.

- **Java EE 1.2**: És l'estàndard de creació d'aplicacions Java, el qual conté les especificacions de JPA i JSF. Les especificacions són documents amb normes i per tant no necessiten manteniment, però les llibreries que utilitzen aquestes especificacions sí que poden quedar obsoletes. Com que requereix llibreries noves per a l'actualització, llavors sí que s'utilitzarà una nova versió d'aquestes especificacions.

- **Hibernate 3.3.1.GA**: Implementació de les especificacions de JPA. Aquesta versió ha quedat obsoleta a causa de vulnerabilitats. Les vulnerabilitats de Hibernate són especialment problemàtiques, ja que estan relacionades amb l'accés a fitxers de configuració de la base de dades.

4.1.2 Seguretat

La seguretat és un element molt important en un projecte com aquest, ja que requereix d'un servidor que tracta amb dades sensibles, com és en aquest cas dins del context de l'administració d'una entitat pública.

En l'apartat anterior s'ha parlat sobre els problemes de vulnerabilitats de la versió antiga del framework, front a la necessitat d'unir el sistema del framework amb el de la resta de l'administració. L'actualització de les llibreries aportarà millores de seguretat de cara a la xarxa exterior.

Pel que fa a la seguretat de la intranet (o xarxa interna) de l'Ajuntament, també és important protegir-se de possibles atacs que es realitzen des de l'interior. Encara que la intranet siga una xarxa privada a la qual tenen accés grups ben definits i limitats, aquesta no es troba exenta d'atacs que puguen posar en risc la informació delicada de l'organització, ja que molts d'aquests atacs són provocats pels mateixos usuaris (*Áudea*, 2022).

En aquest sentit, és necessari implementar mecanismes dins del projecte que puguen garantir la confidencialitat de les dades. Per açò, es va desenvolupar en el seu moment l'estructura de control d'accés a dades, on es va definir una jerarquia de rols i privilegis dins de l'administració per a controlar l'accés dels usuaris a les aplicacions.

A part de l'estructura de control, el framework no implementa cap altra mesura de seguretat. A la xarxa interna encara hi ha informació sensible que pot estar exposada a atacs, sobretot de cara al futur, amb l'adaptació del framework per a treballar amb la xarxa externa.

4.1.3 Falta de documentació i accessibilitat a nous desenvolupadors

Una altra limitació que no cal passar per alt és la falta de documentació sobre el software desenvolupat. Açò dificulta a nous col·laboradors fer ús d'aquest framework i també poder mantenir-lo realitzant les actualitzacions necessàries en el futur.

Així doncs, el framework *Openadmin* requereix una documentació del software del projecte i una guia per a futurs usuaris de com instal·lar-lo i configurar-lo. D'aquesta manera podrà aplicar-se aquest sistema a les administracions d'altres organitzacions.

4.2 Metodologia de treball

En aquesta secció, es descriu la metodologia de treball utilitzada per a l'elaboració del projecte, entrant en detall sobre l'ús de les ferramentes que han sigut utilitzades durant les diferents fases de desenvolupament. En primer lloc, es descriuen les fases de desenvolupament i l'entorn. Després, s'analitzen els gestors de dependències de la base de dades i, finalment, es detalla el procediment per realitzar els tests automàtics.

4.2.1 Fases de desenvolupament

Amb la finalitat de donar context sobre el desenvolupament del projecte, a continuació es descriuen breument quines són les diferents fases de treball que s'han dut a terme a l'hora d'aconseguir cadascuna de les funcionalitats que es descriuen a la secció 4.3:

- **Investigació**: Consisteix en realitzar una recerca de les tecnologies necessàries per a desenvolupar una determinada funcionalitat, comparant les diferents solucions i elegint l'alternativa que millor s'adapte al conjunt del projecte, de manera consensuada amb el responsable del Departament.

- **Disseny**: Consisteix en desenvolupar un disseny de la funcionalitat a implementar. Principalment, aquest disseny consisteix en un diagrama on es veuen les classes i els mètodes que s'han d'implementar per aconseguir aquesta funcionalitat.

- **Implementació**: Una vegada tenim el disseny de l'estructura del programa i quines tecnologies van a utilitzar-se, aquesta fase consisteix en desenvolupar el codi. Dins de l'estructura del projecte, cada funcionalitat està representada com un paquet que integra les classes o altres sub-paquets relacionats amb cada funcionalitat.

- **Proves**: Finalment, aquesta fase consisteix en realitzar els tests necessaris per a comprovar que la funcionalitat està implementada correctament.

4.2.2 Entorns de desenvolupament

Per a desenvolupar el framework s'ha utilitzat l'entorn de desenvolupament Eclipse[6]. Aquest entorn és multi-plataforma i de codi obert, permetent escriure el codi del projecte, compilar-lo i executar-lo dins d'un dels servidors virtuals que poden instal·lar-se en l'entorn.

Cal comentar també que, per a instal·lar les diferents llibreries que necessita el framework, s'utilitza Maven[7] com a gestor de dependències. Aquesta ferramenta permet simplificar els processos d'actualització de dependències i de compilació del projecte. A nivell general, és important remarcar que es disposa d'un fitxer de configuració (pom.xml) on es defineix la configuració i s'especifiquen els paràmetres de compilació del projecte, així com les llibreries utilitzades.

4.2.3 Gestor de base de dades pgAdmin

A l'Ajuntament s'utilitza el sistema gestor de base de dades PostgreSQL[8], ja que utilitza base de dades d'objectes relacionals (ORDBMS) que la fa diferent a altres sistemes com MySQL. Un ORDBMS combina les característiques de les bases de dades relacionals amb la programació orientada a objectes, de manera que aquest sistema de base de dades aporta les ferramentes necessàries per emmagatzemar i gestionar tots els objectes de la lògica del framework.

Les diferents bases de dades utilitzades al projecte s'han gestionat mitjançant l'aplicació pgAdmin[9], que és un entorn visual on es poden crear i configurar servidors de base de dades postgreSQL.

També resulta una ferramenta útil per a validar el funcionament del projecte, comprovant de manera visual si s'han realitzat correctament les operacions mitjançant una interfície molt intuitiva (Figura 12).

[6] https://www.eclipse.org/
[7] https://maven.apache.org/
[8] https://www.postgresql.org/
[9] https://www.pgadmin.org/

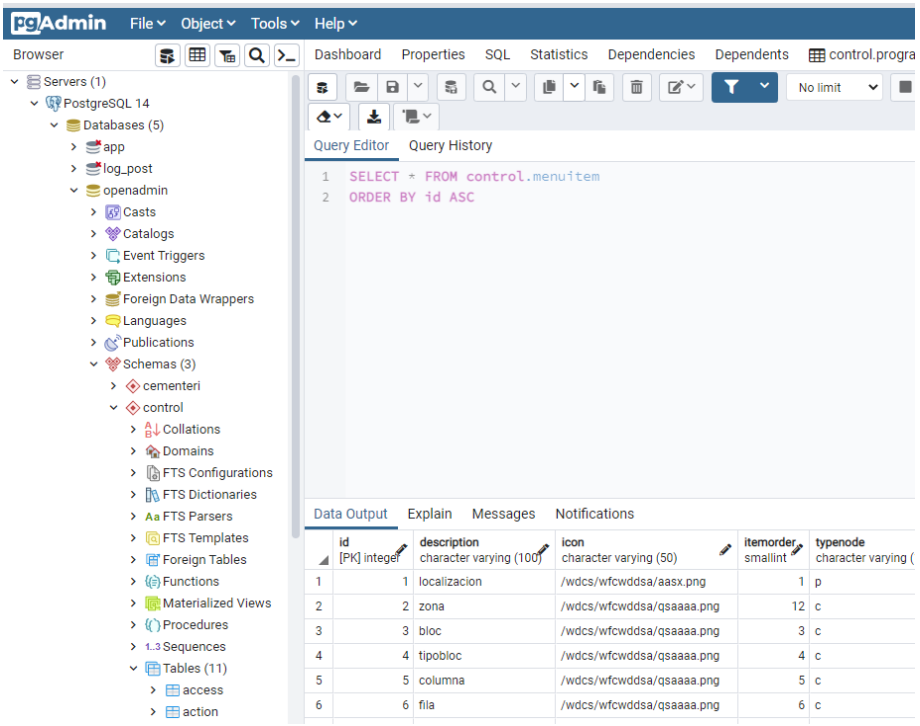


***Figura 12**: Interfície per a gestionar les diferents bases de dades i visualitzar les dades que contenen.*
Font: elaboració pròpia.

4.2.4 Tests automàtics amb Junit

Per a comprovar el correcte funcionament del framework i les seves aplicacions, es fa ús de la ferramenta de tests automàtics Junit[10]. Cada vegada que hi ha una nova funcionalitat implementada, aquesta pot validar-se de manera controlada mitjançant un test programat amb certes condicions de validació. En aquest sentit, la llibreria Junit és la ferramenta de proves automàtiques més utilitzada per a la programació d'aplicacions en Java.

Aquesta ferramenta aporta classes i mètodes per a programar tests on plantejar un escenari i on saber si una determinada funcionalitat presenta un comportament esperat o inesperat. D'esta manera, segons es van realitzant canvis en el codi de les funcionalitats, es poden anar passant aquests tests per a comprovar que els canvis s'han aplicat satisfactòriament.

4.3 Investigació de tecnologies per a l'actualització

Per entendre la necessitat de l'actualització, s'exposa a continuació una introducció breu sobre el context del desenvolupament d'aplicacions *Java* i com ha anat canviant durant els darrers anys.

[10] https://junit.org/junit5/

Les especificacions de *Java EE* aporten un ecosistema robust per a desenvolupar aplicacions empresarials amb *Java*. Com s'ha nomenat anteriorment, el framework es basa en l'especificació *Java EE*, definint la lògica d'accés a dades amb JPA (Java Persistence Api) i la capa de presentació amb JSF (Java Server Faces).

Aquest conjunt d'especificacions estava sent mantingut per *Oracle*, des de 1999 (Figura 13) fins a l'any 2017, on no va poder mantenir el ritme d'actualitzacions. La competència directa i altres alternatives, com per exemple *Spring*, abordaven el mercat ràpidament (Vermeer, 2021). Així doncs, *Oracle* va alliberar la propietat intel·lectual de *Java EE* i finalment va ser adoptada per la companyia *Eclipse Foundation*, que s'ha encarregat de seguir realitzant actualitzacions, passant a canviar el nom de la plataforma a *Jakarta EE*[11].

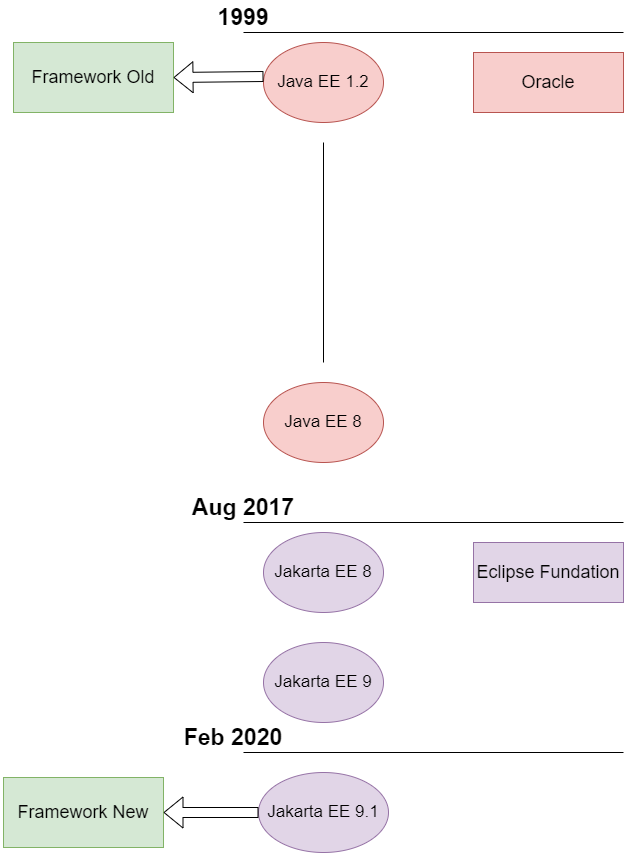


***Figura 13**: Evolució de les especificacions de Java EE indicant la versió utilitzada en el framework abans i després de l'actualització. Font: elaboració pròpia.*

Com es pot observar a la figura 13, hi ha un interval de temps de 20 anys entre les tecnologies utilitzades per al framework.

Jakarta naix de la necessitat de fer més àgil a *Java EE*, per preparar-lo per al nou ecosistema de micro-servicis i adopcions ràpides de tecnologies que l'antiga versió de *Java EE* no oferia.

[11] https://jakarta.ee/

Un detall a destacar, és el canvi de nomenclatura de la plataforma de *Java EE* a *Jakarta EE*, on també es va canviar la nomenclatura dels paquets, com per exemple el paquet JPA de *javax.persistence*, que en la versió actual ha passat a anomenar-se *jakarta.persistence*. Per tant, en migrar el framework per a treballar amb les noves especificacions de *Jakarta EE* hi haurà que canviar el nom a totes les dependències de *Java EE* del projecte.

Amb l'actualització es busca treballar amb l'última versió de *Jakarta EE*, la 9.1. Aquesta versió és important, ja que és compatible amb un entorn de Java més modern, el *Java SE 11*. Als següents apartats es mostra com afecta aquesta nova especificació a la resta de tecnologies utilitzades i quines són les solucions elegides.

4.3.1 Actualització de la capa de persistència

Per aconseguir que l'anterior lògica de la capa de persistència siga compatible amb la resta de noves especificacions de l'actualització, i per tant amb la versió Java SE 11, va a ser necessari migrar el mòdul de persistència a les noves especificacions de Jakarta EE, amb la seua nova versió de JPA (Jakarta Persistence Api).

En aquest sentit, amb la renovació de l'entorn de Java a la versió 11, podran implementar-se dins la lògica d'accés a dades nous mecanismes que abans no existien, com la utilització de paràmetres de tipus *Generic*[12] en els mètodes de la interfície DAO, que aportaran un major control dins la lògica.

4.3.2 Actualització de la capa de presentació

Vist que la llibreria de components del framework, *RichFaces*, s'ha quedat obsoleta, hi ha que buscar tecnologies alternatives que siguen compatibles amb les especificacions de la nova versió del Framework, utilitzant la versió de Java 11.

Segons comenta Caules en un article (2022), a l'ecosistema de desenvolupament d'aplicacions web actual predominen els frameworks de JavaScript basats en components com *Vue*[13], *Angular*[14] i *React*[15]. Per altra banda, la utilització de l'estàndard JSF ha anat cada vegada més de capa caiguda durant els últims anys.

El gran increment d'usuaris d'Internet i la millora del rendiment dels dispositius dels usuaris, ha fet que moltes aplicacions lliuren al servidor de la responsabilitat de carregar les pàgines web, optant per delegar la càrrega de les aplicacions al Client, fent que els frameworks anomenats anteriorment *Vue*, *Angular* i *React* reben més atenció.

---

[12] https://docs.oracle.com/javase/tutorial/extra/generics/index.html
[13] https://vuejs.org/
[14] https://angular.io/
[15] https://es.reactjs.org/

Encara que aquestes alternatives poden ser interessants per a desenvolupar les aplicacions del framework *Openadmin*, s'ha decidit que la millor opció segueix sent utilitzar l'estàndard de JSF més una llibreria de components per a desenvolupar les aplicacions web.

En el cas d'aquest projecte, delegar la càrrega de les aplicacions al servidor no és cap problema, ja que a l'organització només hi ha un grup reduït d'usuaris actius, comparat amb el gran tràfic d'usuaris que hi ha a les pàgines basades en Client.

La gran ventaja d'utilitzar JSF és que és molt estable, ja que al tractar-se d'un estàndard tindrà molta longevitat, al contrari que els altres frameworks i implementacions, que una vegada es queden obsolets no queda més remei que migrar a una altra plataforma i per tant cambiar tota l'estructura, un canvi que pot resultar molt costós.

Com també comenta Caules al seu article, en moltes ocasions per a solucions que requereixen un cert grau de complexitat dins d'una intranet, JSF resulta una solució raonable, com és el cas del framework que es vol desenvolupar en aquest treball.

Pel que fa a la substitució de *RichFaces*[16], el framework de JSF més utilitzat hui en dia en el mercat és *PrimeFaces*[17], una llibreria de components que continua sent mantinguda i aportarà noves funcionalitats respecte a l'anterior.

Així doncs, les tecnologies escollides per a la capa de presentació són la nova versió de JSF Jakarta Server Faces 3.0 junt al framework de PrimeFaces.

4.3.3 Actualització control d'accés

Un altre dels requeriments que es busca amb la nova versió del framework és fer més accessible la configuració de l'estructura de control, passant d'utilitzar múltiples fitxers de text pla diferents a utilitzar-ne només un, recollint totes les dades dins d'un fitxer d'extensió *Yaml*[18].

A la figura 14 es mostra l'aspecte d'aquest format i com pot utilitzar-se per a definir objectes amb propietats.

[16] https://richfaces.jboss.org/
[17] https://www.primefaces.org/
[18] https://yaml.org/spec/1.2.2/

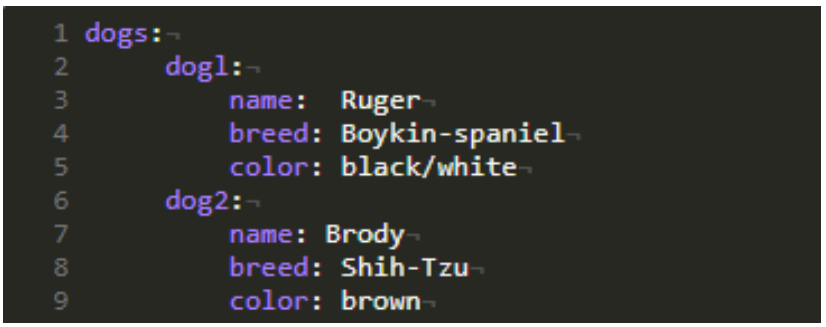


***Figura 14**: Sintaxi d'un fitxer de text amb format Yaml. Font: elaboració pròpia.*

L'avantatge de treballar amb un fitxer Yaml és la possibilitat d'utilitzar llibreries especialitzades en processar la seva informació.

En aquest sentit, la llibreria *Jackson*[19] és una ferramenta popular per a serialitzar objectes Java a diferents formats de text, incloent Yaml. També és possible a la inversa, serialitzar un text Yaml per a obtenir objectes Java. Aquesta funció pot ser de gran utilitat al projecte a l'hora de processar la configuració de control i simplificarà el codi de la càrrega inicial de l'estructura de control.

4.3.4 Encriptació i seguretat

Com s'ha comentat anteriorment, amb l'actualització del framework es busca implementar nous mecanismes de seguretat per a protegir els fitxers que contenen informació sensible.

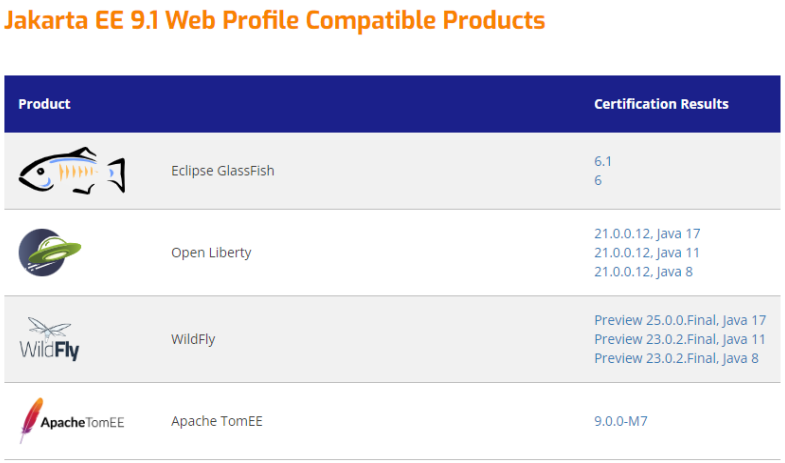
En aquest sentit, és necessari utilitzar el xifrat amb clau (Vargas i Mnedez, 2015, p. 9) per a identificar als administradors encarregats de gestionar el framework i per a protegir els fitxers de configuració del servidor.

Entre les opcions de software lliure que aporten funcionalitats de criptografia, la llibreria *BouncyCastle*[20] és la més reconeguda, ja que implementa l'estàndard de Java JCE (*Java Cryptography Extension*) afegint-li més funcions i algoritmes d'encriptació.

4.3.5 Servidor

Amb la renovació del projecte, s'ha de buscar un nou servidor on allotjar-lo i poder executar-lo. En la figura 15 es mostra una taula de la pàgina web de *Jakarta EE* on es llisten les compatibilitats que té la versió 9.1 amb diversos tipus de servidor.

[19] https://github.com/FasterXML/jackson/
[20] https://www.bouncycastle.org/



Jakarta EE 9.1 Web Profile Compatible Products

Product		Certification Results
	Eclipse GlassFish	6.1 6
	Open Liberty	21.0.0.12, Java 17 21.0.0.12, Java 11 21.0.0.12, Java 8
WildFly	WildFly	Preview 25.0.0.Final, Java 17 Preview 23.0.2.Final, Java 11 Preview 23.0.2.Final, Java 8
ApacheTomEE	Apache TomEE	9.0.0-M7

***Figura 15**: Taula de compatibilitats de servidors amb Jakarta 9.1. Font: https://jakarta.ee/compatibility/certification/9.1/*

En aquest sentit, el framework de l'anterior versió estava instal·lat dins d'un servidor Tomcat 6.0, el qual no suporta la nova plataforma de *Jakarta*. Per tant, és imprescindible que les proves de l'actualització estiguen suportades amb un servidor *Tomcat 9.0.0 M7*[21], per a aprofitar les similituds en la configuració de l'anterior versió.

Per altra banda, es realitzaran proves amb el servidor Jboss *Wildfly 23.0.2*[22], ja que permet l'ús de transaccions de tipus JTA (Java Transaction Api), que són més segures, ja que es delega la gestió de les transaccions al contenidor, en aquest cas el servidor *Wildfly*. En habilitar aquest tipus de transaccions, també serà possible comunicar diferents tipus de sistemes de base de dades, ja que la lògica de Jboss s'encarrega de configurar automàticament les connexions perquè es produïsquen les transaccions de forma segura.

4.4 Disseny

Una vegada s'han vist les tecnologies necessàries per a cobrir els requeriments establerts, a continuació es mostren els dissenys clau que s'han elaborat:

- **Disseny del nou perfil de les classes de persistència** de dades *ConnecionDao i DaoJpaHibernate*, on s'ha implementat la utilització de paràmetres genèrics en l'interfície DAO (figura 16).

[21] https://tomcat.apache.org/
[22] https://www.wildfly.org/

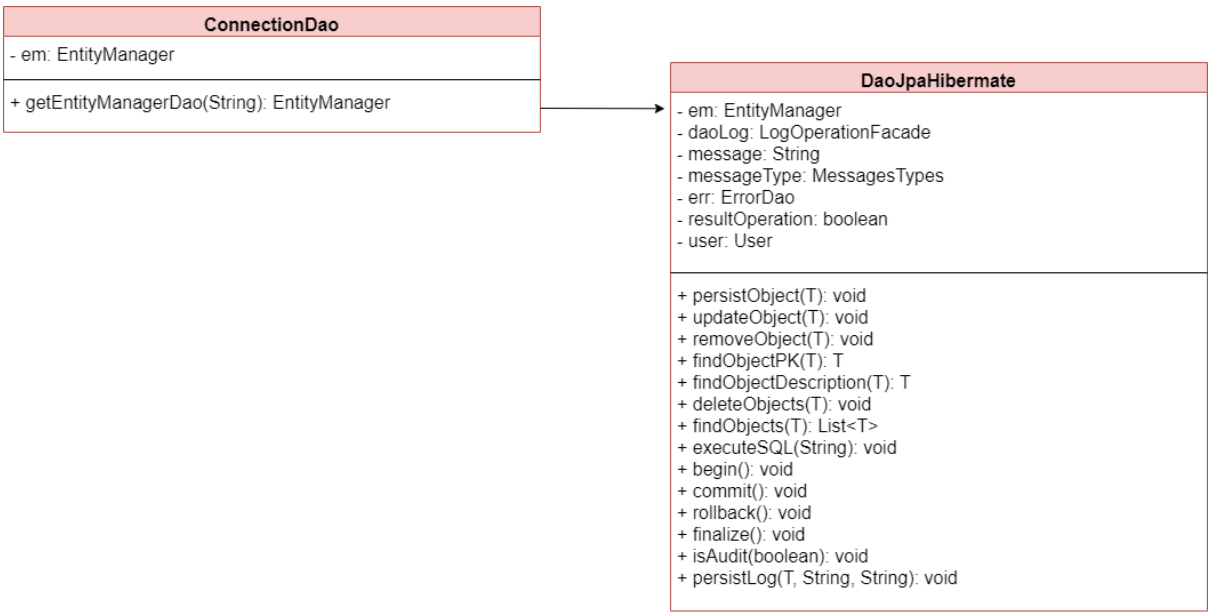


***Figura 16**: Classes del paquet DAO amb la nova inclusió de paràmetres genèrics. Font: elaboració pròpia.*

- A la figura 17 es mostra el **procés per a persistir l'estructura de control**, utilitzant el processament d'un fitxer yaml, que finalment serà utilitzada per la lògica per a realitzar les funcions del framework. La nova classe *InitDataLoad* realitza la càrrega inicial del model de dades de control del framework. Aquesta classe implementa mecanismes de processat de fitxers yaml i interactua amb els mètodes DAO per a realitzar la persistència de les dades obtingudes.

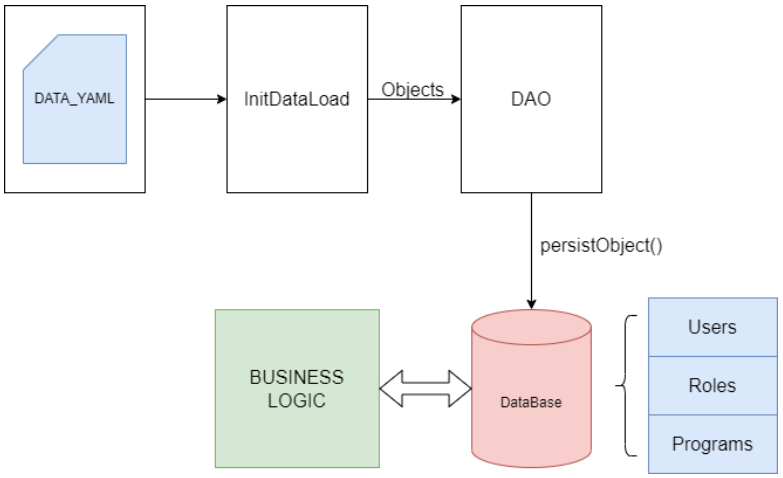


***Figura 17**: Esquema de l'estructura de control realitzant el mapeig del fitxer yaml a objectes. Font: elaboració pròpia.*

- El **disseny de la classe *CipherUtils*** que implementa les noves funcionalitats d'encriptació es mostra a la figura 18. Aquesta classe servirà com a ferramenta per a encriptar i desencriptar els fitxers importants del framework.

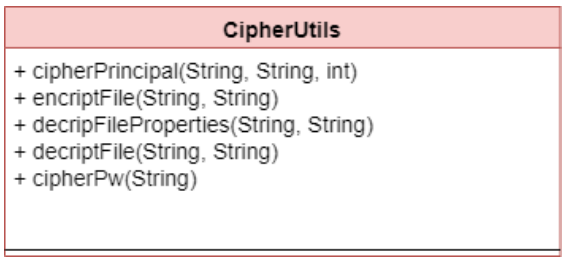


***Figura 18**: Perfil de la classe CipherUtils i els mètodes implementats per a realitzar funcions d'encriptació. Font: elaboració pròpia.*

4.5 Implementació

En aquest apartat es descriu la renovació del framework explicant la implementació de les noves tecnologies i les seues funcionalitats.

4.5.1 Capa de persistència

Per a utilitzar la nova versió de JPA, s'han d'instal·lar les dependències de Jakarta EE 9.1. En actualitzar la versió de JPA també cal instal·lar i configurar una versió recent de Hibernate que implemente aquestes noves especificacions de Jakarta. Per tant, s'inclourà al fitxer pom.xml la dependència de Jakarta 9.1 i la versió compatible de *Hibernate* 5.6.5. També és necessari realitzar el canvi de nom a les dependències de *javax.persistence* a la nova *jakarta.persistence.*

A més, tal i com hem comentat abans, els mètodes DAO de la classe *DaoJpaHibernate* han passat a definir-se com mètodes genèrics. La utilització de mètodes genèrics és beneficiosa quan es vol aplicar la mateixa funcionalitat a molts tipus d'objectes diferents.

A la figura 19 es mostra la definició del mètode persistObject, que s'ha redefinit com a mètode genèric.

```
public <T extends Base> void persistObject(T obj){
```

***Figura 19**: Definició d'un mètode amb paràmetres genèrics. Font: elaboració pròpia.*

D'aquesta manera, tots els objectes que reben els mètodes de la interfície DAO requereixen heretar les propietats de *Base,* amb el que es manté la flexibilitat anterior, a la vegada que

s'eviten possibles errors en introduir un objecte de tipus inesperat durant el temps d'execució.

4.5.2 Capa de presentació

Inicialment, per a utilitzar les noves especificacions de Jakarta EE en la capa de presentació, s'inclourà al pom.xml la versió 3.0 de JSF.

De la mateixa manera que amb les especificacions de JPA, cal renombrar totes les dependències de Java EE per a realitzar la migració a Jakarta Server Faces 3.0, canviant de *javax.faces* a *jakarta.faces.*

Amb la instal·lació completada, es pot començar a treballar amb les pàgines xhtml utilitzant aquesta nova versió de Jakarta.

A continuació, es mostra la construcció de la plantilla de l'aplicació i s'exemplifica com s'introdueixen els continguts de manera programàtica a través de la lògica. A la figura 20 es mostra la plantilla *layout.xhtml* on es defineix el marc per a introduir els diferents continguts de l'aplicació.

Mitjançant l'element de JSF *ui:insert,* es defineixen els components que ha de contenir l'aplicació que implemente la plantilla *layout.xhtml*. Com es pot observar a la figura 20 s'utilitza *ui:insert* per indicar que s'han d'implementar els elements *header*, *toolbar*, *menu* i *view.*

```
<h:body>

    <div id = "idheader">
        <ui:insert name="header"/>
    </div>

    <h:form id="idform">

        <p:outputPanel id="idtoolbar" deferred="true">

            <ui:insert name="toolbar"/>

        </p:outputPanel>

        <h:panelGrid id="idmainmenuview" columns="2" columnClasses="top" rowClasses="rowTop">

            <ui:insert name="menu"/>
            <ui:insert name="view"/>

        </h:panelGrid>

    </h:form>
```

***Figura 20**: Definició d'una plantilla JSF dins d'un arxiu xhtml. Font: elaboració pròpia.*

A continuació, a la figura 21 es mostra la pàgina *home.xhtml* i com s'implementa l'element de la plantilla *header,* mitjançant *<ui:define>*. Aquest element header s'ha implementat amb un panel de *Primefaces* que mostra els noms de l'entitat, de l'usuari i del programa que està utilitzant-se.

Per a obtindre el valor d'aquests noms de forma dinàmica s'utilitza el llenguatge d'expressions EL dins de l'etiqueta *value*, que permet accedir a la informació de dins de la lògica.

```
                template="layout.xhtml">
    <ui:define name="header">
        <p:panelGrid columns="3" layout="grid" styleClass="showcase-text-align-center">
            <p:outputLabel id="identity" value="#{msg.entity}"/>
            <p:outputLabel id="iduser" value="#{msg.user} #{ctx.username}" />
            <p:outputPanel id="idprogram" ajaxRendered="true">
                    <h:outputText binding="#{main.program}" />
            </p:outputPanel>
        </p:panelGrid>
    </ui:define>
```

***Figura 21**: Implementació de la plantilla layout.xhtml. Font: elaboració pròpia.*

Una vegada hem vist com es poden definir plantilles, a continuació revisarem com adaptar la web dinàmica amb els Beans de la lògica.

En aquest sentit, per a mostrar la funcionalitat de *Jakarta Server Faces* i *PrimeFaces,* s'explica a continuació com s'obtenen els valors d'aquest *header* i com són representats de forma dinàmica.

El valor de l'entitat s'obté de la variable msg, on s'ha guardat un dels fitxers d'idiomes. A la figura 22 podem observar com s'ha obtés aquest fitxer amb un element de JSF *loadBundle,* el qual assigna el contingut del fitxer *messagesMain_es* a la variable *msg*. D'aquesta manera, utilitzant la sentència EL es pot accedir a la informació guardada en la variable *msg*.

```
<f:loadBundle basename="messagesMain_es" var="msg" />
```

***Figura 22**: Obtenció d'arxius de recursos amb l'etiqueta loadBundle. Font: elaboració pròpia.*

Pel que fa al nom de l'usuari i el nom del programa, aquests s'obtenen de forma dinàmica, utilitzant els objectes de la lògica nomenats *Bean,* els quals mitjançant mecanismes de JSF poden guardar informació durant el transcurs de la sessió del *Client.*

El nom de l'usuari s'obté de la variable *ctx,* que fa referència a l'objecte de la lògica ContextAction, el qual gestiona la informació de control d'accés (per exemple, els rols, la llista de programes) de l'usuari que ha obert la sessió.

Per a instanciar la classe *ContextAction* com a un *Bean* s'ha introduit l'etiqueta @Named, indicant el nom de la variable que s'utilitzarà en les expressions EL. També s'ha afegit l'etiqueta @SessionScoped per a indicar que la informació d'aquest *Bean* serà mantinguda durant la sessió de l'usuari. A la figura 23 es pot observar com s'indiquen aquestes anotacions dins la classe.

```
@Named("ctx")
@SessionScoped
public class ContextAction implements Serializable{
```

***Figura 23**: Utilització de les etiquetes per a la definició de Beans. Font: elaboració pròpia.*

En resum, en identificar-se l'usuari s'emmagatzema tota la informació de l'accés dins del context, en el *Bean* ContextAction, i per a obtindre la informació d'aquest *Bean*, en aquest cas el nom de l'usuari, s'ha utilitzat l'expressió *ctx.username*.

L'últim text que apareix a l'encapçalament indica el programa que té obert l'usuari i cada vegada que l'usuari canvia de programa, el text s'actualitza. El nom del programa s'obté de la *Bean MainHome*, una classe on es guarda informació relativa a la navegació i es gestiona la presentació de components. Aquest Bean, a l'igual que ContextAction, també és instanciada amb les etiquetes @Named i @SessionScoped. A la figura 24 es mostra com es veu l'element *header* representat al navegador.

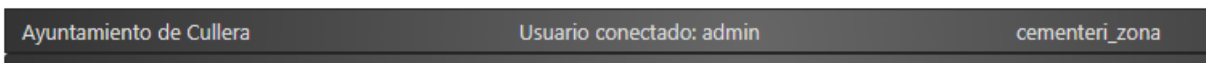


***Figura 24**: Barra dinàmica d'estat de l'aplicació web generada pel framework. Font: elaboració pròpia.*

De manera anàloga a com s'ha creat aquest element *header,* es poden crear la resta d'elements de la plantilla. A la figura 25 es mostra com és la pàgina *home.xhtml* vista des del navegador després de carregar els altres tres elements de la plantilla: el *toolbar*, el *menu* i la *view*.

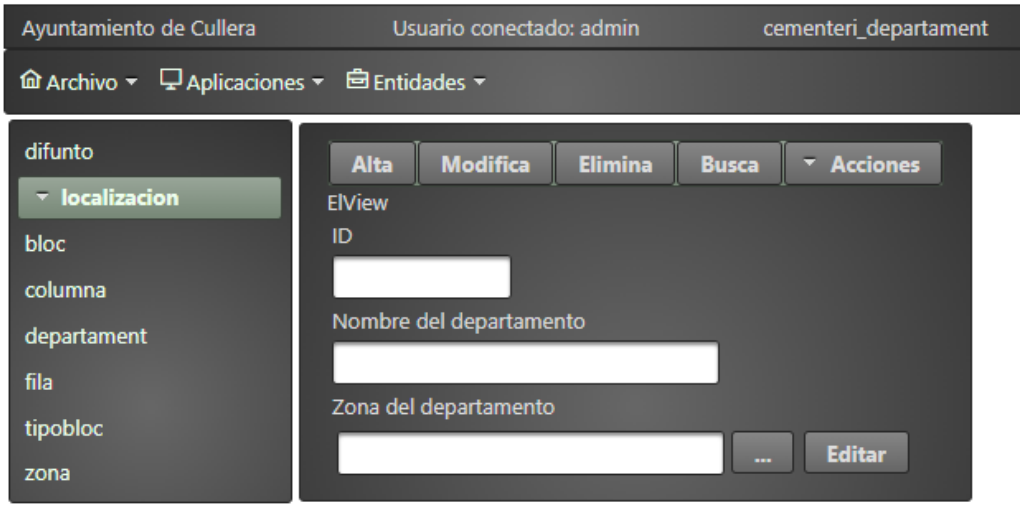


***Figura 25**: Utilització de les etiquetes per a la definició de Beans. Font: elaboració pròpia.*

4.5.3 Actualització Control d'Accés

Una part essencial per al desenvolupament d'aplicacions amb el framework és la definició de l'estructura de control d'accesos. En aquest sentit, perquè resulte més accessible treballar amb l'estructura, s'ha canviat l'estratègia amb la qual es defineix i es processa, utilitzant un fitxer Yaml.

En la versió anterior del framework *Openadmin*, cada classe de l'estructura de control tenia un fitxer propi, de manera que hi havia que gestionar múltiples fitxers diferents que contenen elements relacionats. Aquesta tasca es complica segons l'estructura va creixent. A la figura 26 es resumeix en una imatge el que era treballar abans de l'actualització.

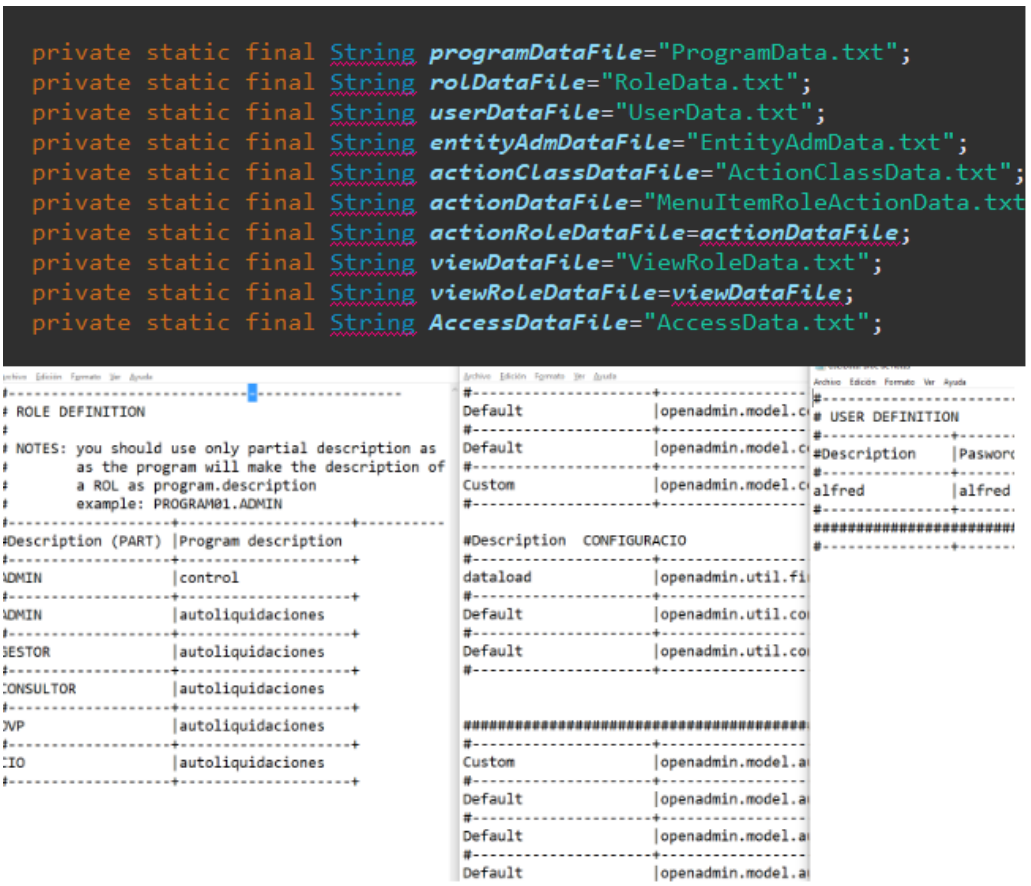


***Figura 26**: Antiga gestió de l'estructura de control. Font: elaboració pròpia.*

El fet de definir tots els objectes dins d'un fitxer Yaml, permet obtindre una millor vista general de l'estructura i de les propietats de cada element. A la figura 27 es mostra com estan definits els objectes de l'estructura de control dins del fitxer Yaml.

```yaml
#-------------------------------------PROGRAMS-------------------------

programs:
   - description: "cementeri"
     icon: "/aa/aaaaa/ssaaa.png"

#-------------------------------------ROLES----------------------------

roles:

 - description: "admin_cementeri"
   program: "cementeri"

 - description: "gestor_cementeri"
   program: "cementeri"


#-------------------------------------ENTITYADMINS---------------------

entityadmins:
 - description: "log_post"
   code: 13424
   icon: "/aa/aaaaa/ssaaa.png"
   conn: "postgres"

 - description: "openadmin"
   code: 13425
   icon: "/aa/aaaaa/ssaaa.png"
   conn: "postgres"

```

***Figura 27**: Format de l'estructura de control dins del fitxer Yaml. Font: elaboració pròpia.*

En la figura 28 es mostra la pestanya *outline* de l'editor d'Eclipse on pot observar-se de forma clara l'arbre d'elements i les seves propietats.

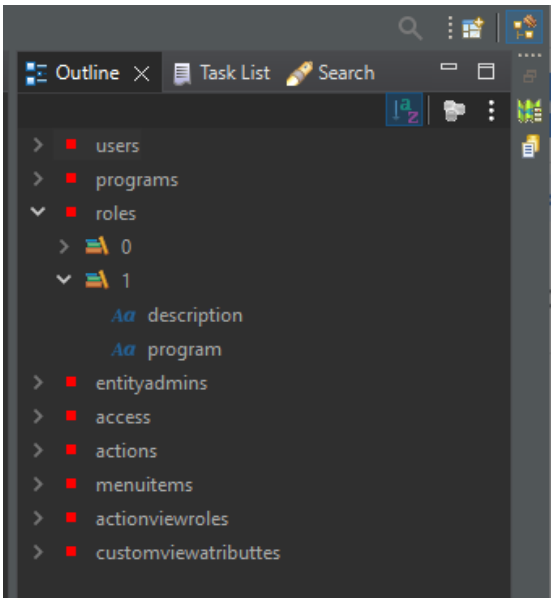


***Figura 28**: Arbre d'elements del fitxer Yaml vist des del Outline d'Eclipse. Font: elaboració pròpia.*

El processament amb *Jackson* està implementat dins la classe *InitDataLoad,* on es realitza la càrrega inicial de l'estructura de dades. Mitjançant l'objecte *ObjectMapper*, es processen els diferents elements del fitxer yaml i es retornen transformats directament a objectes Java. Després, els objectes es poden persistir mitjançant els mètodes de la interfície DAO.

4.5.4 Encriptació i seguretat

Per aportar una capa de seguretat addicional al projecte, s'ha desenvolupat un mòdul amb funcionalitats d'encriptació implementant la llibreria *BouncyCastle*. Aquestes funcionalitats estan implementades en la clase CipherUtils dins del paquet del projecte *openDao.util.cipher.*

El mètode *cipherPricipal* serveix com a façana per a realitzar les dues funcions principals de la classe d'encriptar i desencriptar. La funció rep com a paràmetre la ruta, el nom del fitxer i un número enter, per a indicar el mode de la funció on 1 vol dir encriptar i 2 desencriptar.

Pel que fa a l'algoritme utilitzat, l'encriptació es realitza utilitzant el xifrat per blocs de AES, amb el mecanisme d'emplenat PKCS5/PKCS7.

La classe *CipherUtils* s'utilitza en la identificació d'usuaris amb privilegi d'administrador. Per a iniciar la sessió com a administrador és necessari tindre un arxiu encriptat amb les credencials, a mode de clau. En la figura 29 es veu el fitxer de credencials abans i després de l'encriptació.

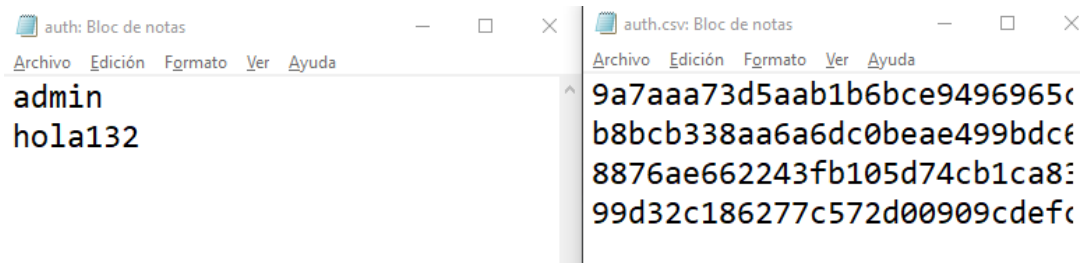


***Figura 29**: Fitxer de credencials abans i després de l'encriptació. Font: elaboració pròpia.*

Aquesta classe també s'utilitza per a encriptar el fitxer de connexió a la base de dades *PostgresSQL*, per a ocultar la contrasenya i la direcció utilitzada.

4.5.5 Documentació

Finalment, s'ha definit una metodologia per documentar el codi mitjançant la funció d'Eclipse de generar Javadocs. D'aquesta manera, es poden incloure etiquetes als comentaris del codi per tal de definir una estructura per a crear una documentació generada en format Html. A la figura 30 es pot veure la utilització d'etiquetes per a definir elements com *@param*, per a definir els paràmetres de la classe o *@see* per a indicar un enllaç.

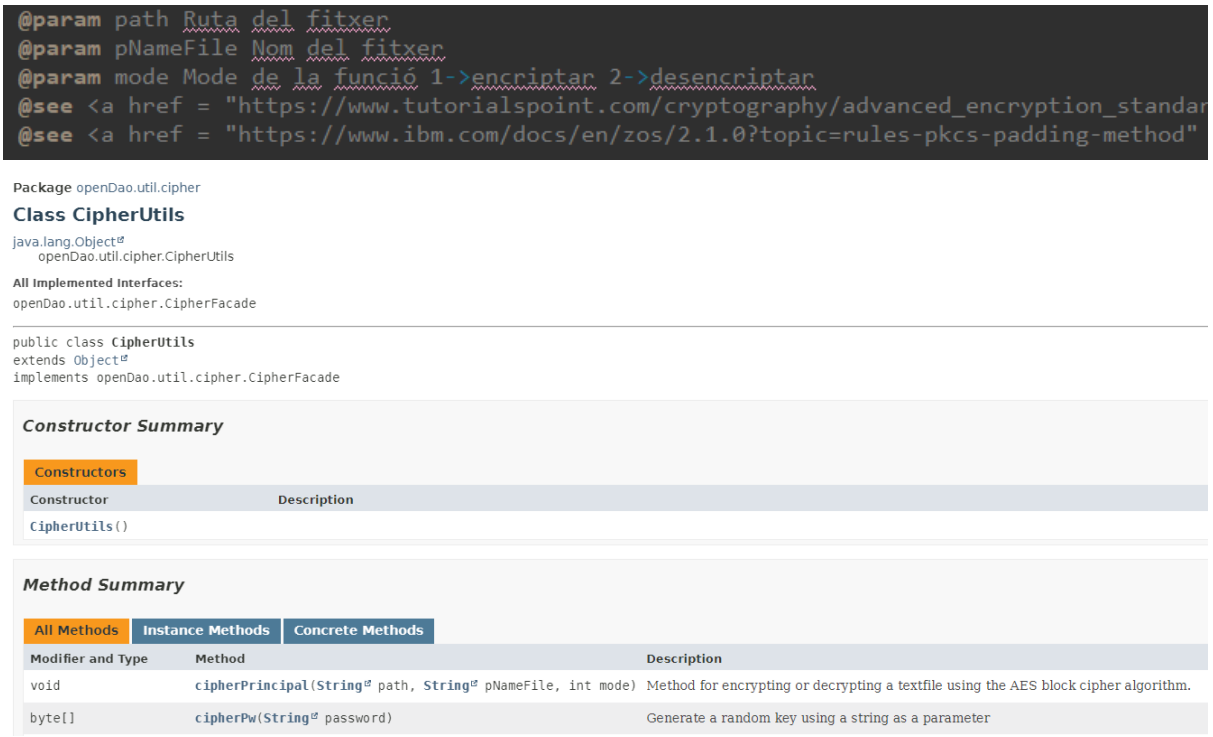


***Figura 30**: Visualització de la documentació generada amb JavaDocs. Font: elaboració pròpia.*

NOTA: Com annex a aquest projecte es troba tota la documentació generada amb Javadoc.

4.6 Proves

Els tests unitaris són classes Java amb un sol mètode, on es programa un escenari en el que es puga observar el comportament d'una determinada funcionalitat donades certes condicions.

En Junit, per a indicar que el mètode és un test, s'utilitza l'etiqueta *@Test* i és dins d'aquest mètode on es programa l'escenari de proves. Amb els mètodes *assert* es poden definir condicions de validació per a determinar si un mètode funciona correctament o no.

A continuació, es mostra un test automàtic per al mètode *cipherPrincipal()* del mòdul d'encriptació *CipherUtils,* per a simular el cas d'inici de sessió d'un administrador i observar el seu comportament front a aquest escenari.

Dins del mètode amb el decorador *@Test*, es crida a la funció cipherPrincipal() i es defineixen dues condicions *assertEqual()* per a comprovar que el resultat del mètode és el mateix que el que s'esperava.

Per a aquesta prova, es va a simular que l'usuari introdueix incorrectament la contrasenya. A la figura 31 s'observa com el test informa que una condició no s'ha complit. Es pot afirmar

que el mètode cipherPrincipal() funciona correctament ja que ha desencriptat les credencials, en canvi l'usuari ha introduït les credencials incorrectament.

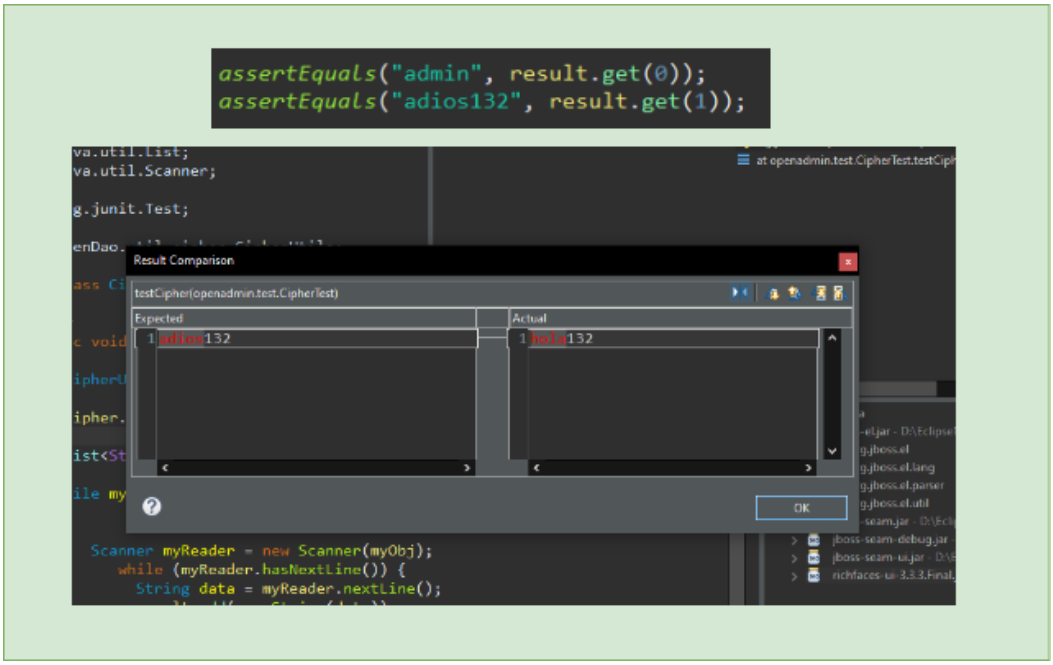


***Figura 31**: Visualització del resultat del test realitzat amb Junit. Font: elaboració pròpia.*

5 APLICACIÓ DEL CEMENTERI MUNICIPAL DE CULLERA

Una vegada hem vist com s'ha actualitzat el framework, en aquest apartat es presenta quin seria el recorregut a seguir per a elaborar una aplicació fent ús de les ferramentes del framework a través d'un exemple. En aquest cas, s'ha proposat dur a terme el desenvolupament d'una aplicació per a gestionar les dades del Cementeri Municipal de Cullera. A continuació, es descriuen els requeriments de l'aplicació, el seu disseny i finalment, la implementació que s'ha dut a terme.

5.1 Requeriments

En aquest apartat es llisten els requeriments que ha de tenir l'aplicació de gestió del cementeri, segons s'ha decidit en una reunió de treball. En aquest sentit, s'han analitzat els documents del cementeri que s'han facilitat des de l'Ajuntament de Cullera per al desenvolupament d'aquest projecte.

Els documents contenien una llista amb tots els registres dels difunts i les seves dades (nom, cognoms, data de naixement, etc…). A partir d'aquesta informació, s'ha pogut definir quins camps contindrà el formulari de registrar difunts.

El cementeri està dividit en 9 zones, on en cadascuna d'elles hi ha un nombre determinat de blocs de nínxols. Cada bloc es distribueix en columnes i files, on cada nínxol s'identifica per la columna i la fila on està ubicat. A més a més, determinades zones del cementiri tenen també departaments que tenen un nom que els defineix. Aquest nom és merament ornamental i no s'utilitza per a identificar la ubicació dels nínxols. Finalment, els blocs poden ser de tipus diferents (nínxol, panteó, sepulcre i ossari). A la figura 32 es pot observar el mapa del Cementeri Municipal de Cullera.

***Figura 32**: Mapa del cementeri. Font: Ajuntament de Cullera.*

A partir d'aquesta informació, s'ha definit la següent llista dels requeriments que ha de tindre l'aplicació, a partir dels quals es definirà el model de dades de l'aplicació i els dissenys dels formularis:

- Formularis amb possibilitat de realitzar les accions CRUD, que possibilitaran:
    - Donar d'alta un registre
    - Modificar un registre
    - Eliminar un registre
    - Buscar un registre.

- L'aplicació ha d'oferir gestionar els següents programes[23]:
    - Gestió de difunts:
        - Nom del difunt
        - Cognoms del difunt
        - Fila on es troba el difunt
        - Data de naixement
        - Data de defunció
    - Gestió de zones
        - Nom de la zona
    - Gestió de departaments.
        - Nom del departament
        - Zona a la que pertany el departament

[23] Un programa equival a un formulari amb unes determinades funcionalitats.

- Gestió de blocs de parcel·les.
    - Nom del bloc
    - Zona a la que pertany el bloc
    - Tipus de bloc

- Gestió de tipus de bloc.
    - Nom del tipus de bloc

- Gestió de columna de la parcel·la.
    - Número de columna
    - Bloc al que pertany la columna

- Gestió de fila de la parcel·la. (Nínxol)
    - Número de fila
    - Columna a la que pertany la fila

- Una funcionalitat que permeta mostrar un mapa on s'indique l'ubicació dels nínxols.

- Per a cada registre de fila es generarà un codi GIS que s'utilitzarà en el futur per a georeferenciar els nínxols. El codi ha de tenir el següent format:
    - ZZBBBCCCFFDD
    - ZZ: nº Zona
    - BBB: Tipo bloc + nº Bloc
    - CCC: nº Columna
    - FF: nº Nínxol
    - DD: nº de difunts

- L'aplicació contemplarà 3 tipus d'usuari:
    - Administrador
    - Gestor
    - Consultor

5.2 Disseny

A continuació, es representa el diagrama de flux sobre la navegació de l'usuari de l'aplicació (figura 33). Una vegada l'usuari s'identifica correctament i tria l'aplicació del cementeri, tindrà accés als diferents programes de gestió. Cada programa contindrà un formulari i una barra d'accions. Si vol donar d'alta un registre d'un determinat formulari emplenarà tots els camps i premerà el botò de l'acció *Alta*. Si vol realitzar qualsevol de les altres accions abans haurà de buscar un registre amb el botó *Cerca*. En seleccionar un dels resultats de la cerca se li omplirà el formulari automàticament i ja podrà modificar-lo, eliminar-lo o mostrar la seva ubicació al mapa.

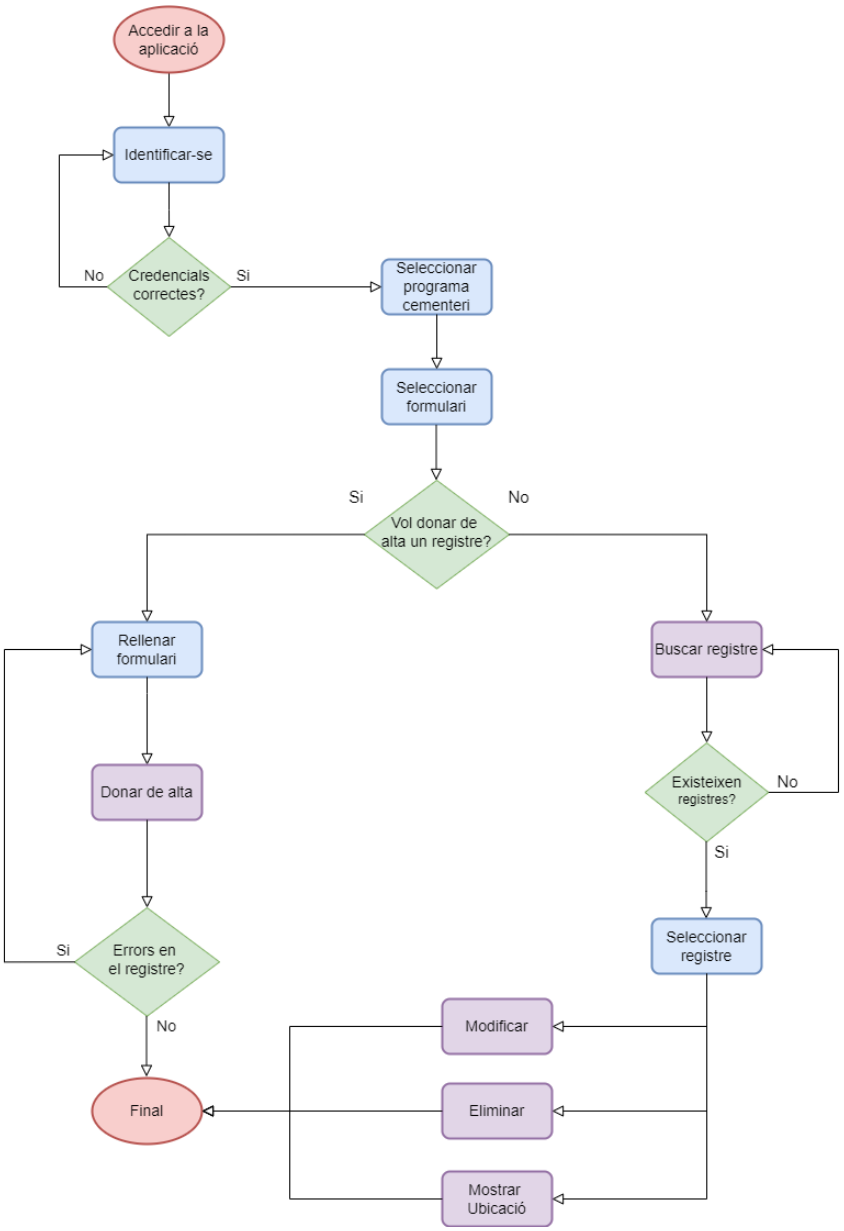


***Figura 33**: Diagrama de flux de la navegació de l'usuari en l'aplicació del cementeri. Font: elaboració pròpia.*

Les dades de l'aplicació han de complir les relacions entre entitats que estan representades en el següent model (figura 34).

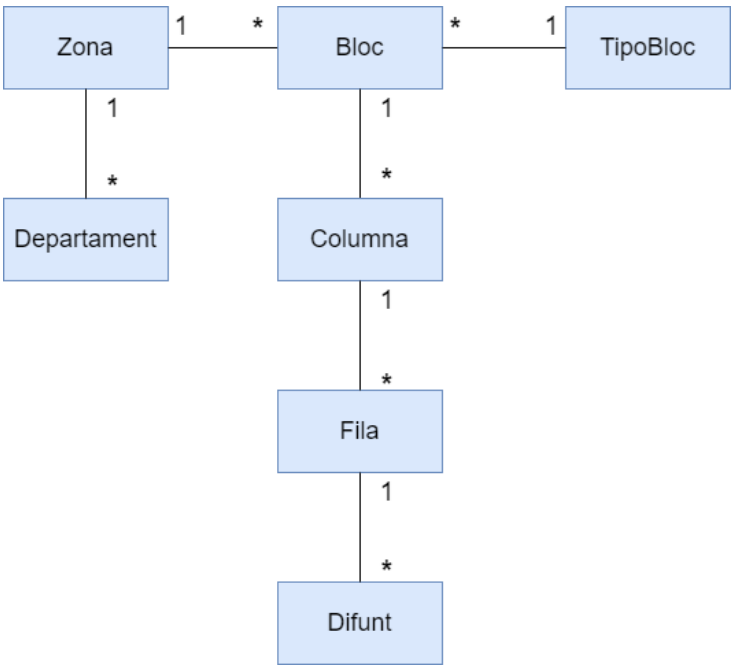


***Figura 34**: Esquema del model de dades de l'aplicació del cementeri. Font: elaboració pròpia.*

5.3 Implementació

En aquesta secció, es descriurà el procés de desenvolupament de l'aplicació proposada, fent ús de les ferramentes del framework actualitzat.

Inicialment, cal definir el model de l'aplicació segons els requeriments especificats. En aquest sentit, el paquet on es defineixen els models ha de començar per la nomenclatura *openadmin.model,* seguit pel nom de l'aplicació (en aquest cas *openadmin.model.cementeri*). Dins d'aquest paquet es crearà una classe Java per cadascun dels programes o formularis que s'utilitzaran en l'aplicació. A la figura 35 es poden veure els programes que s'han definit segons les indicacions dels requisits.

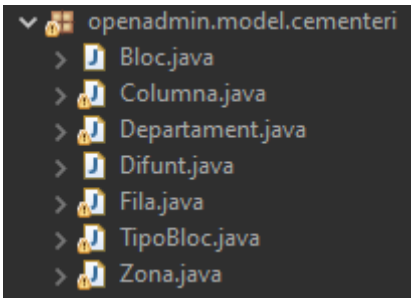


***Figura 35**: Paquet dels models de formulari per a l'aplicació del cementeri. Font: elaboració pròpia.*

Cadascuna d'aquestes classes inclourà les etiquetes de JPA @Entity i @Table per a indicar que són entitats gestionades per JPA i el nom de la taula que tenen. Amb aquestes

etiquetes es defineixen els paràmetres per a aplicar els mecanismes de mapeig objecte-relacional. A mode d'exemple, a la figura 36 es pot observar la sintaxi d'una d'aquestes taules.

```
@Entity
@Table(name = "difunt", schema = "cementeri", uniqueConstraints = @UniqueConstraint(columnNames = "description"))
```

***Figura 36**: Esquema del model de dades de l'aplicació del cementeri. Font: elaboració pròpia.*

També s'utilitzaran les etiquetes @Id i @generatedValue per a indicar que es generarà un identificador automàticament per a cadascun dels registres d'aquesta classe que es guarden en una taula. L'etiqueta @ManyToOne permet definir les relacions entre objectes segons el disseny del model de dades. Les etiquetes @Getter i @Setter són anotacions de la llibreria lombok que permeten simplificar codi estalviant tindre que escriure a mà els mètodes *getter* i *setter*.

A continuació, a la figura 37 es mostra com a exemple la classe *Difunt,* on s'han definit els seus atributs segons els requisits per al programa de Gestió de difunts. Com es pot observar, aquesta classe té 3 atributs de tipus *String*, 2 camps de data *Date* i un atribut de tipus Fila.

```
@Entity
@Table(name = "difunt", schema = "cementeri", uniqueConstraints = @UniqueConstraint(columnNames = "description"))

public class Difunt implements Base, java.io.Serializable{

    private static final long serialVersionUID = 0106202204;

    /** attribute that contain the identifier*/
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Getter @Setter
    private Integer id;

    @Length(max = 30)
    @NotNull
    @Getter @Setter
    private String description;

    @Length(max = 30)
    @NotNull
    @Getter @Setter
    private String nom;

    @Length(max = 30)
    @NotNull
    @Getter @Setter
    private String cognom;

    @ManyToOne
    @JoinColumn(name = "fila", nullable= false)
    @Getter @Setter
    private Fila fila;

    @Getter @Setter
    private Date datNaixement;

    @Getter @Setter
    private Date datDefuncio;
```

***Figura 37**: Model de formulari per al programa de gestió de difunts. Font: elaboració pròpia.*

Aquest model, en ser interpretat pel framework, representarà en el formulari de l'aplicació els següents elements: 3 camps de text, 2 camps de selecció de data i un camp de selecció d'objecte.

Una vegada s'han definit tots els models del paquet *openadmin.model.cementeri,* el següent pas consisteix en definir dins del fitxer Yaml els elements de control d'accés de l'aplicació. A la figura 38 es mostra un exemple de com definir alguns d'aquests elements de control, com els rols de l'aplicació o les accions CRUD per al programa de gestió de zones.

***Figura 38**: Introducció de les dades de l'estructura de control per a l'aplicació del cementeri.*
Font: elaboració pròpia.

Una vegada ja estan els models definits i l'estructura de control configurada, l'aplicació ja està llesta per a ser utilitzada. A la figura 39 es mostra el procés d'emplenar el formulari per a donar d'alta un registre de difunt. La lògica ha generat dos camps amb components de selecció de calendari perquè al model de l'aplicació s'han definit dos atributs de tipus *Date*.

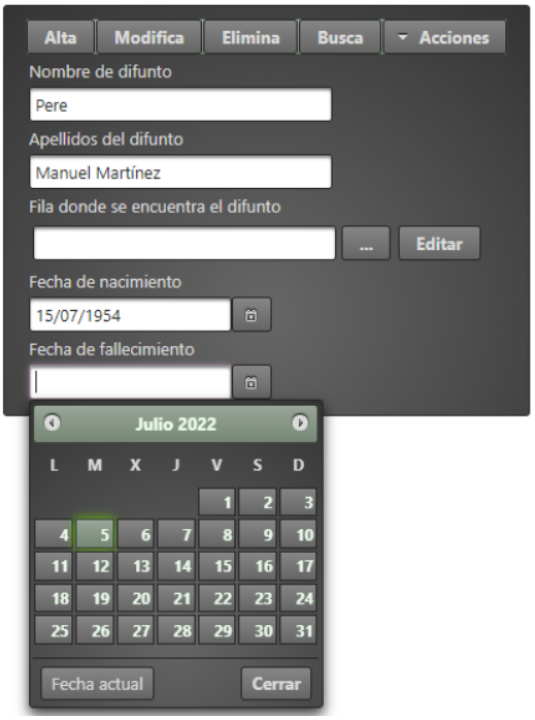


***Figura 39**: Introducció de dades al formulari de gestió de difunts. Font: elaboració pròpia.*

A la figura 40 es pot observar que s'ha guardat en la base de dades el registre del difunt en realitzar l'operació de donar d'alta.

id [PK] integer	cognom character varying (30)	datdefuncio timestamp without time zone	datnaixement timestamp without time zone	description character varying (30)
1	Manuel Martínez	05/07/2022	15/07/1954	Pere Manuel Martínez

***Figura 40**: Visualització de les dades en donar d'alta un difunt. Font: elaboració pròpia.*

Seguint els requeriments que s'han especificat en la Secció 5.1, cal també una funcionalitat que permeta mostrar un mapa on es mostre la ubicació del nínxol.

En aquest sentit, aquesta funcionalitat és considerada com una funcionalitat addicional pròpia de cada aplicació. Aquest tipus de funcionalitats addicionals de les aplicacions s'han de definir dins del paquet d'accions personalitzades *openadmin.action*. Les accions personalitzades contenen un mètode anomenat *execute()*, on s'ha de definir el codi que implementa l'acció.

A la classe UbicationAction s'ha implementat la nova acció de mostrar el mapa, segons ho indicava als requisits de l'aplicació. La funció *openMap()* construeix el component de

Primefaces Gmap i el representa en una finestra. La classe es situa dins del paquet d'accions personalitzades d'aquesta aplicació *openadmin.action.cementeri*.

Una vegada implementada l'acció, s'ha d'indicar a l'estructura de control en quin programa s'utilitza aquesta funcionalitat. Segons els requeriments, es vol representar la ubicació del nínxol, per tant l'acció de mostrar la ubicació s'implementarà a la classe *Fila* i a la classe *Difunt*. A la figura 41 es mostra com s'hauria d'indicar en l'estructura de control que l'acció de mostrar ubicació perteneix als programes de gestió de *Fila* i gestió de *Difunt*.

```
- description: ":ubication"
  groupid: 1
  icon: "/wdcs/wfcwddsa/qsaaaa.png"
  nameclass:
    description: "openadmin.model.cementeri.Difunt"
- description: ":ubication"
  groupid: 1
  icon: "/wdcs/wfcwddsa/qsaaaa.png"
  nameclass:
    description: "openadmin.model.cementeri.Fila"
```

***Figura 41**: Estructura de control per a afegir una acció. Font: elaboració pròpia.*

Una vegada configurada, aquesta acció personalitzada es troba dins del desplegable d'accions dels programes *Fila* i *Difunt*. A la figura 42 es mostra el resultat d'executar l'acció i de mostrar el mapa en l'aplicació.

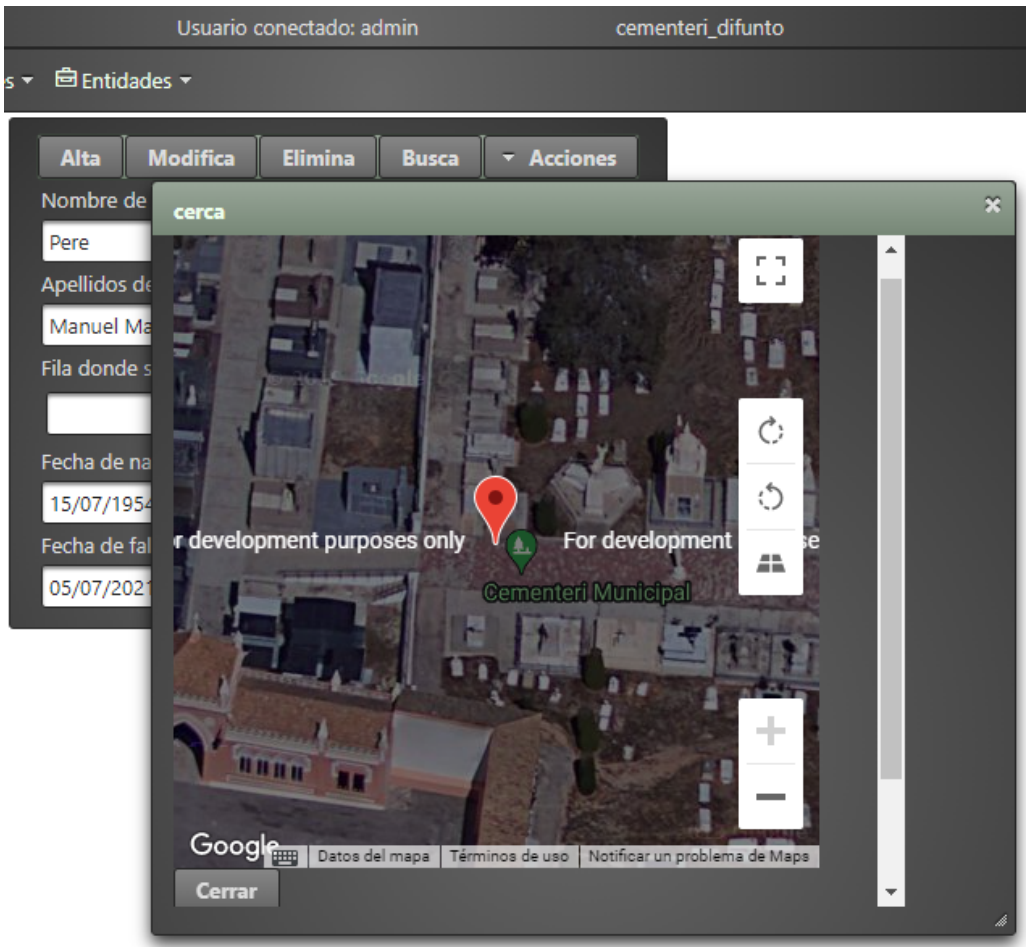


***Figura 42**: Visualització de l'acció personalitzada de mostrar mapa. Font: elaboració pròpia.*

5.4 Manual per al desenvolupador

El manual annex a l'entrega d'aquest treball, conté informació detallada sobre el procés de desenvolupament d'aplicacions i també del procediment necessari per a instal·lar el framework al servidor de l'organització.

6 CONCLUSIONS

El TFG que es presenta en este document naix de l'experiència a les pràctiques curriculars. En aquest treball, es documenta pas a pas tot el procés que s'ha seguit per a desenvolupar la renovació d'un framework que s'utilitzava per a l'administració de l'Ajuntament de Cullera i que s'havia quedat obsolet.

Un dels aspectes més interessants dels projectes que s'han desenvolupat al llarg de la carrera és la programació. En este sentit, aquest TFG es pot vore com a una oportunitat per a posar a prova els coneixements adquirits durant estos anys i demostrar que poden ser aplicats en un entorn professional, en este cas, ajudant a millorar l'administració del poble de Cullera.

Tot i que s'ha partit d'un framework ja desenvolupat, era tot un repte enfrontar-se al projecte amb un codi ja implementat i sense molta documentació, ja que té més 80 classes i més de 10.000 línies de codi. A més a més, algunes tecnologies utilitzades com PrimeFaces tenen poca documentació publicada, llavors moltes de les funcionalitats han hagut de provar-se a base de proves assaig i error i, de vegades, buscar en el mateix codi font de les llibreries.

Així i tot, l'experiència adquirida al Grau de Tecnologies Interactives gestionant projectes en equip ha sigut molt útil per planificar els objectius i definir les tasques necessàries per a dur endavant el projecte, per molta complexitat que tinguera.

Llavors, el primer pas va ser començar estudiant la versió del framework de 2009 que utilitzaven a l'Ajuntament, per a conèixer el seu funcionament i limitacions. Seguidament, es va realitzar una tasca de recerca que consistia en investigar quines noves tecnologies es podien aplicar a l'actualització del framework. Finalment, es van fer unes proves preliminars amb aquestes tecnologies abans d'implementar-les al projecte. Una vegada actualitzat el framework, es va utilitzar per a desenvolupar l'aplicació de gestió del Cementeri Municipal de Cullera. Esta aplicació va a utilitzar-se més endavant, per a georeferenciar les dades del cementeri.

A més a més, aquest framework seguirà utilitzant-se per a desenvolupar més aplicacions que completaran el procés de digitalització de l'administració de l'Ajuntament de Cullera, i potser servisca d'exemple per implementar aquest tipus de sistema a la modernització de l'administració d'altres entitats.

El desenvolupament d'aquestes pràctiques i l'elaboració del TFG, han servit per millorar la comprensió sobre sistemes software, la capacitat d'anàlisi i la resolució dels problemes que sorgeixen durant el desenvolupament. Ha sigut una bona experiència de disseny i organització d'un projecte real. A més a més, compartir aquesta experiència amb l'equip del Departament d'Informàtica de l'Ajuntament de Cullera també ajuda a potenciar aquest tipus de competències, i sobretot, cal posar en valor la importància que té la formació permanent per a combatre l'obsolescència dels sistemes.

7 BIBLIOGRAFIA

Áudea. *¿Es segura la intranet de su empresa?* (2021, 17 agosto). Recuperado 7 de marzo de 2022, de https://www.audea.com/es-segura-la-intranet-de-su-empresa/#:%7E:text=Las%20Intranets%20requieren%20varias%20medidas,indeseables%2C%20y%20controlar%20el%20tr%C3%A1fico.

Caules, C. Á. (2022, 3 junio). *¿Tiene futuro JSF con Jakarta EE?* Arquitectura Java. Recuperado 18 de marzo de 2022, de https://www.arquitecturajava.com/tiene-futuro-jsf/

Garrido Tejero, A. (2016). Implementación del patrón DAO (Data Access Objecto). https://riunet.upv.es/handle/10251/63592

Guindon, C. (s. f.). *Jakarta EE 8: Past, Present, and Future | The Eclipse Foundation*. Eclipse. Recuperado 20 de mayo de 2022, de https://www.eclipse.org/community/eclipse_newsletter/2019/august/jakartaee8.php

Jonsson, J., & Kaliski, B. (2003). *Public-key cryptography standards (PKCS)# 1: RSA cryptography specifications version 2.1* (No. rfc3447).

Langer, A. (s. f.). *Generic And Parameterized Types*. angelikalanger. Recuperado 20 de marzo de 2022, de http://www.angelikalanger.com/GenericsFAQ/FAQSections/ParameterizedTypes.html#Fundamentals%20of%20Parameterized%20Types

Novo, H. L. (2012, 27 noviembre). *Polimorfismo en Java • Héctor Luaces Novo*. Héctor Luaces. Recuperado 6 de abril de 2022, de https://www.luaces-novo.es/polimorfismo-en-java/

Parnin, C., Bird, C., & Murphy-Hill, E. (2013). Adoption and use of Java generics. *Empirical Software Engineering*, *18*(6), 1047-1089.

Pech-May, F., Gomez-Rodriguez, M. A., Luis, A., & Lara-Jeronimo, S. U. (2012). Desarrollo de Aplicaciones web con JPA, EJB, JSF y PrimeFaces. *Instituto Tecnológico Superior de los Ríos, Tabasco, México*.

Ruiz, J. (2022, 14 enero). *La inspección lamenta la baja implantación de la Administración electrónica*. Levante-EMV. Recuperado 1 de Febrero de 2022 https://www.levante-emv.com/comunitat-valenciana/2022/01/14/inspeccion-lamenta-baja-implantacion-administracion-61548849.html

Stancapiano, L. (2017). *Mastering Java EE Development with WildFly*. Packt Publishing Ltd.

Vargas, Y. T. M., & Mnedez, H. A. M. (2015). Comparación de algoritmos basados en la criptografía simétrica DES, AES y 3DES. *Mundo Fesc*, *5*(9), 14-21.

Vermeer, B. (2021, 15 noviembre). *Spring dominates the Java ecosystem with 60% using it for their main applications*. Snyk. Recuperado 26 de febrero de 2022, de https://snyk.io/blog/spring-dominates-the-java-ecosystem-with-60-using-it-for-their-main-applications/

Wulftange, M. (2018, 30 mayo). *Poor RichFaces*. codewhitesec. Recuperado 10 de marzo de 2022, de https://codewhitesec.blogspot.com/2018/05/poor-richfaces.html