Especificació d'Arquitectura Justícia

Canigo3.4 – Cloud Native

Projecte	Nom
Arquitectura	**ESPECIFICACIÓ D'ARQUITECTURA JUSTICIA – CANIGO3.4 CLOUD NATIVE**
Client	
GENERALITAT DE CATALUNYA – CTTI (CENTRE DE TELECOMUNICACIONS I TECNOLOGIES DE LA INFORMACIÓ)	
Nom de l'arxiu	Responsable tècnic
Especificacio Arquitectura JUS_Canigo3.4_CloudNative.doc	
Data	Fase actual
13/12/2022	

CONTROL DE DOCUMENTACIÓ

Identificació

Referència	Tipus document	Localització document	Difusió
ARQ_CAN34_CLOUDNATIVE	Arquitectura		**RESTRINGIDA**

Autor(s)	Revisat per

Control de canvis

Versió	Autor(s)	Motiu	Data
0.1	Arquitectura OIT	Versió draft inicial	23-07-2020
1.0	Arquitectura OIT	Versió definitiva	18-10-2021
1.1	Arquitectura OIT	Canvi eina WSO2 per Keycloak No utilitzar eina Pentaho	08-04-2022
1.2	Arquitectura OIT	Eliminar referències a decisions antigues	12-04-2022
1.3	Arquitectura OIT	Eliminar referències a decisions antigues (II)	20-04-2022
1.4	OTEC	Eliminació de referències proveïdor	13-12-2022

Distribució de còpies

Organització	Persones
Generalitat de Catalunya	

ÍNDEX

1 INTRODUCCIÓ

1.1 PRÒLEG

Aquest document descriu la solució arquitectònica per a projectes del Dept. Justícia amb Canigó 3.4 i front-end Angular 9 desplegats a plataforma Cloud.

Degut a què aquest document és una evolució de l'arquitectura del projecte eJustícia.cat (eJCAT), basats un subconjunt de mòduls en Canigo1 + JSP, i altres mòduls en Canigo 3 + JSF o Angular, en algunes parts d'aquest document es farà referència a la paraula eJCAT (referències a documentació, arquitectura, etc.) però cal entendre-ho sempre en un sentit més global, aplicant a qualsevol projecte del Dept. Justícia que hagi de fer ús d'aquesta nova arquitectura.

1.2 DESTINATARIS

Aquest document té com a destinataris els Caps de Projecte, Arquitectes Tècnics, Analistes Orgànics, Analistes Funcionals i Programadors dels projectes i serveis de Justícia, per tal que puguin comprendre l'arquitectura del projecte i d'aquesta manera desenvolupar les seves funcionalitats d'acord amb el què dicta aquesta arquitectura.

2 DIRECTRIUS ARQUITECTÒNIQUES

Aquest capítol descriu els objectius més importants d'arquitectura i les seves restriccions externes.

2.1 OBJECTIUS DE L'ARQUITECTURA

Els objectius d'arquitectura vindran dictats en gran mesura pels requeriments no funcionals i restriccions generals del projecte. Cada sistema d'informació ha de tenir definits i documentats els seus propis *Requeriments No Funcionals*.

Es poden destacar alguns principis d'arquitectura (CTTI o propis) dins l'arquitectura JUS-Canigo 3.4 Cloud:

- Arquitectura desacoblada: per permetre als components i aplicacions mantenir-se completament autònoms i independents.
- Arquitectura orientada a serveis: cada cop més, les aplicacions poden ser consumides externament (exposant la seva funcionalitat) o bé han d'integrar-se amb aplicacions de tercers. Les relacions s'han de dur a terme mitjançant serveis sempre que sigui possible.
- Utilitzar solucions transversals: sempre que sigui possible, com p.ex. GICAR o el framework Canigo.
- Generar codi estàndard i no propietari. S'assumeix que la utilització òptima d'alguns productes comercials (HCP Hitachi, LibreOffice) genera certes dependències que s'intentaran minimitzar i limitar a mòduls que n'explotin els beneficis amb escreix.

2.2 TASCA DEL SISTEMA

El projecte e-Justícia.cat comprèn l'estratègia per a modernitzar l'Administració Judicial de Catalunya i defineix tres àrees fonamentals d'acció: organització, infraestructura i tecnologia. Dins l'àrea tecnològica trobem, entre d'altres, la necessitat d'implementar un nou sistema de gestió judicial que permeti a Catalunya posicionar-se com a líder en l'àmbit de Justícia.

- Amb innovació tecnològica
- Revisant els processos dins l'àmbit de l'Administració de Justícia (Jutjats, Procuradors, Serveis Comuns, etc.), i donant cobertura a la nova Oficina Judicial.
- Obrint l'Administració de Justícia a tots els actors implicats
- Trobant eficiència i eficàcia en la posta en marxa dels serveis judicials

A nivell tècnic, el nou sistema modifica les arquitectures anteriors per adaptar-la a la nova versió del framework Canigó 3.4 i permetre desplegar nous mòduls i funcionalitats al Cloud. La nova arquitectura del Departament de Justícia utilitza un front-end basat en Angular 9.

L'arquitectura del sistema ha de permetre la convivència entre els nous projectes i sistemes més antics del Departament (p.ex. TEMIS basat en PowerBuilder, mòduls eJCAT en Canigo 1, mòduls eJCAT en Canigo 3.2, etc).

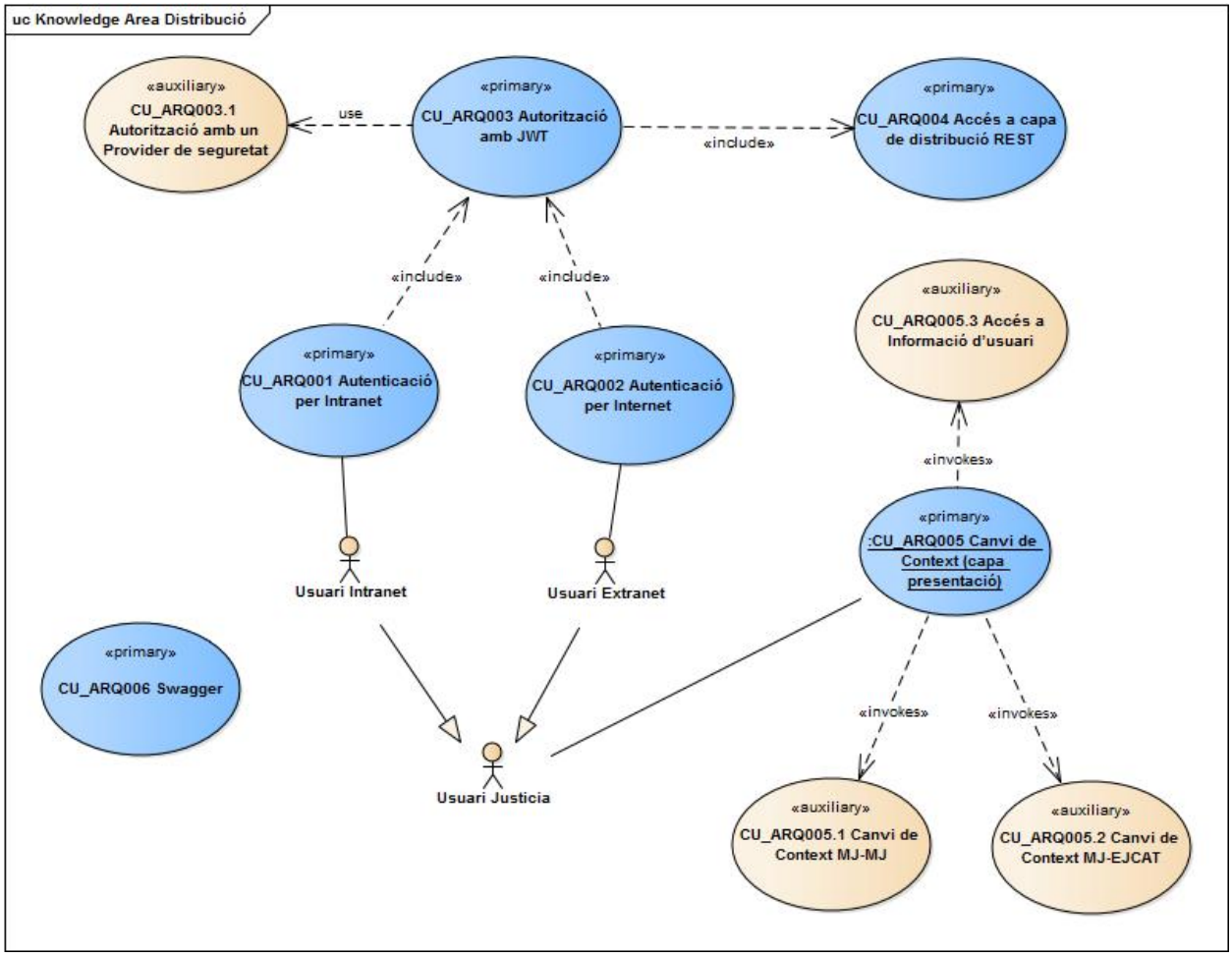
2.3 CASOS D'ÚS DEL SISTEMA RELLEVANTS D'ARQUITECTURA

En aquest capítol s'identifiquen els casos d'ús més rellevants a nivell arquitectònic perquè:

- Representen una funcionalitat central del sistema
- El seu àmbit d'influència engloba varies àrees d'arquitectura
- Fa èmfasi en un punt d'arquitectura específic i delicat

Cal tenir en compte que aquests casos d'ús es poden traduir més endavant en la implementació d'un component d'arquitectura (T-Component) o bé simplement reflectir un escenari que cal especificar com a patró arquitectònic, sense cap implementació associada.

2.3.1 Casos d'ús capa de distribució REST



- CU_ARQ001 Autenticació per Intranet

L'autenticació dels usuaris de la Intranet es delegarà en GICAR.

Després de contactar amb la seva Oficina Tècnica, i tractar possibles solucions d'integració amb el nostre projecte, s'ha optat per desenvolupar la solució basada en agent de Shibboleth .

Es desenvoluparà un T-Component a mida basat en aquest agent, encarregat de redirigir a GICAR per dur a terme l'autenticació de l'usuari. Concretament, estarà configurat per validar el seu accés contra el directori d'usuaris de la intranet, amb els mecanismes que tingui configurats (certificat, usuari-password, tarjeta criptogràfica...).

En cas d'autenticar-se correctament, GICAR enviarà al mòdul client (p.ex. Portal de la Intranet) el resultat de l'operació, en forma de capçaleres.

Finalment, amb el contingut d'aquestes capçaleres aquest mòdul prepararà l'autorització de l'usuari en el nostre sistema, basada en tokens.

- CU_ARQ002 Autenticació per Internet

L'autenticació dels usuaris d'Internet serà similar a la descrita en el punt anterior. Es configurarà Shibboleth per redirigir a un espai de GICAR que validi l'accés contra els directoris d'usuari corresponents per Internet.

Aquest resultat de l'autenticació serà posteriorment recollit i gestionat pel mòdul client (p.ex. Portal de la Extranet).

- CU_ARQ003 Autorització amb JWT

Un cop autenticat correctament l'usuari en el sistema, es procedirà a preparar la seva autorització.

Aquesta estarà basada en tokens JWT en format OpenID Connect.

La gestió de l'autorització es realitzarà mitjançant el servei d'Identity Provider ofert per el producte Keycloak. Aquest servei, basat en OAuth2, oferirà la possibilitat de definir nivells d'accessos per separat, segons el tipus d'usuari a autoritzar en el sistema: un usuari d'intranet, un usuari de l'extranet, un sistema extern, etc...

- CU_ARQ004 Accés a capa de distribució REST

Aquest cas d'ús permet accedir a la capa de lògica de negoci, independentment que l'origen de la petició sigui la capa de presentació Angular o la capa d'integració que gestiona l'entrada des de sistemes externs.

L'accés a la capa de distribució estarà garantida després de superar els filtres de seguretat definits en el punt anterior.

La definició de l'API d'operacions REST de cada mòdul de serveis es realitzarà via Spring Web MVC, amb les anotacions @RestController corresponents.

El format de transferència de la informació entre el front-end i el back-end REST serà JSON, que és un format més lleuger que d'altres com XML.

- CU_ARQ005 Canvi de Context (capa de backend)

La nova arquitectura de mòduls de EjCat+ ha de permetre canviar a mòduls implementats amb les altres arquitectures web de EJCAT:

- Canigó 1.4 amb Struts i capa de presentació JSP, seguretat implementada amb cookies
- Canigó 3.1 amb JSF, seguretat implementada també amb cookies
- Canigó 3.2 amb serveis REST i capa de presentació Angular, seguretat implementada amb tokens JWT

Per cada escenari, els mòduls hauran d'oferir serveis de backend a mida per garantir que es pot saltar d'una aplicació origen a destí, i efectuar el retorn, sense perdre el context on ens trobàvem dins l'aplicació origen. S'ha de considerar que el canvi de context es realitza al mateix navegador al que s'executa la aplicació EjCat+ i serà responsabilitat de cada aplicació garantir el bon funcionament amb aquest navegador.

- CU_ARQ005.1 Canvi de Context EjCat+-EjCat+

En aquest escenari, des d'un microfrontend origen d'un mòdul de EjCat+ volem passar a un altre implementat amb la mateixa tecnologia.

Tant l'origen com el destí tenen el seu backend de serveis REST protegits per Spring Security, validant els tokens contra els mateixos endpoints OpenIdConnect de l'Identity Provider del Keycloak.

Per tant, el mateix token obtingut en origen serveix per al mòdul destí, i no cal fer-ne cap transformació. Però sí és necessari des del mòdul destí proporcionar un servei REST que retorni tota la informació necessària al mòdul origen per entrar al seu context (la URL d'entrada i els paràmetres necessaris de canvi de context)

La informació relativa a l'usuari que canvia de context no s'ha d'enviar en aquests paràmetres, ja que viatja de forma segura en el token JWT.

La resposta que rebrà el mòdul origen a aquesta petició li permetrà fer, des del microfrontend, una redirecció al destí. Aquest destí, en detectar aquesta entrada, recuperarà tota la informació que s'ha facilitat en la crida, i prepararà el nou context en el nou microfrontend.

En cas que hi hagi una possibilitat de retorn a l'aplicació origen, aquesta informació s'haurà d'emmagatzemar en l'espai persistent local de l'usuari administrat per Angular (local storage), de manera que estigui disponible per preparar el canvi de context en la direcció contrària: el retorn.

- CU_ARQ005.2 Canvi de Context Ejcat+-EJCAT

Aquest escenari és més complex, perquè hem de canviar a contexts de mòduls amb diferents implementacions de seguretat: una cookie administrada per una shared library, o un token JWT adreçat per un mòdul de seguretat a mida (i que no és compatible amb el token JWT del Keycloak)

Només es contemplen canvis de context per aplicacions Intranet. Actualment, aquesta lògica de negoci de canvis de context la gestiona el mòdul Portal Intranet actual (implementat amb Canigó 1.4), que ofereix una entrada en el seu controlador d'Struts per gestionar qualsevol operació de canvi de context: validar les dades, enregistrar el canvi, traduir tokens o cookies, i executar la redirecció.

Abans de fer la crida al Portal CAN1.4, però, el mòdul origen EjCat+ ha d'oferir un servei REST per preparar els paràmetres de canvi de context. A més, els ha de preparar en un format que el Portal CAN1.4 pugui entendre.

A més, aquest Portal no treballa amb tokens JWT, si no que ha de rebre un token més curt (que hem anomenat *token de canvi de context*, o tokenCC) que després sigui compatible amb les diferents solucions de seguretat que

tenim a tot EJCAT. El nou servei REST implementat al mòdul origen Ejcat+ també haurà de generar aquest tokenCC.

Amb la informació del tokenCC i dels paràmetres, el mòdul EjCat+ farà la crida al Portal. Aquest validarà les dades, enregistrarà l'operació, farà la traducció corresponent a Cookie o a token JWT, i executarà la redirecció.

Per a la situació en què des d'un mòdul EJCAT hem de fer un retorn a un mòdul EjCat+, hem d'oferir un endpoint transversal que tradueixi aquest tokenCC en el token JWT de EjCat+, que està adreçat pel Keycloak.

Per aquest motiu, el mòdul Portal EjCat+ oferirà un servei REST per conversió de tokensCC en tokens JWT de EjCat+.

L'aplicació destí (EjCat+) en rebre el canvi de context executat pel Portal CAN1.4, amb tokenCC, traduirà aquest pel token JWT necessari per poder entrar ja al context del backend de serveis REST de l'aplicació destí de EjCat+.

- CU_ARQ005.3 Accés a Informació d'usuari

La informació de l'usuari es recupera del d'un component de gestió d'usuaris, i es desa en el token JWT en el seu payload de *claims*.

En cada aplicació, estarà disponible un Security Context creat per Spring, un cop la capa de Spring Security ha donat per vàlida l'autorització de l'usuari (el token JWT que ha enviat).

Com el nostre endpoint OAuth2 d'adreçament de tokens treballa amb format OpenIdConnect, podrem configurar aquesta capa de Spring Security per validar de forma offline el token, i incloure en el Security Context tots els claims que acompanyaven en el token que hem validat.

Per dur a terme aquesta validació es farà servir JWKS (JSON Web Key Set). Com dèiem, gràcies a treballar amb OpenIdConnect, no serà necessari en cada request fer una crida addicional (en un Filter) de validació i desencriptat del token per extreure els claims, sino que directament Spring Security farà aquesta extracció, sense afegir l'overhead d'una crida extra de validació, i ens deixarà el Security Context preparat.

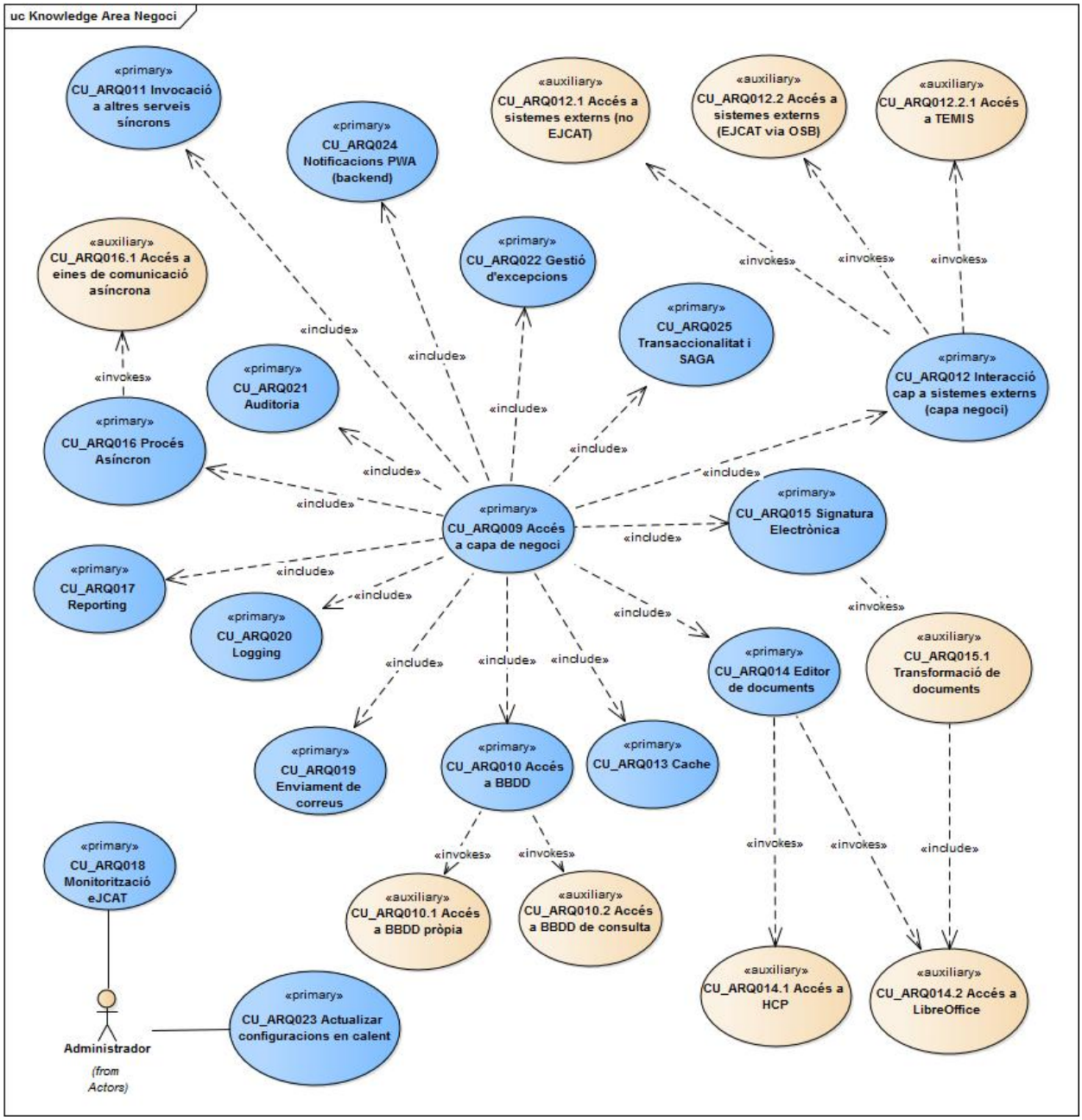
- CU_ARQ006 Swagger

Swagger és un framework per facilitar el disseny, desenvolupament, i documentació d'API RESTful.

El mecanisme és el següent:

- Configurarem el projecte de back-end REST, per incorporar les llibreries necessàries i la configuració base de Swagger2.
- Decorarem els nostres REST Controllers amb unes annotacions pròpies de Swagger2 per cada classe i servei exposat (amb els paràmetres d'entrada, codis de resposta, etc..).
- També disposarem d'annotacions pels objectes View Model dels serveis.
- Swagger2, amb component-scan d'Spring MVC, generarà automàticament tota la documentació dels serveis, en base a les annotacions que hem anat incorporant en el codi.
- Es podrà consultar de forma online la documentació generada.

2.3.2 Casos d'ús de la capa de negoci



- CU_ARQ009 Accés a capa de negoci

Les peticions arriben a la capa de distribució cap als Controller que actuen com a punt d'entrada. Els Controller fan les crides necessàries a la capa de negoci cap els components Service que contenen la lògica de negoci de la funcionalitat executada.

Un servei es:

- una representació lògica d'una activitat empresarial repetible que té un resultat especificat,
- és auto contingut.
- pot utilitzar altres serveis per fer les seves tasques
- no ha d'exposar la seva implementació
- ha de tenir definida una interfície que determina com s'ha d'utilitzar i els possibles resultats.

Com els consumidors dels serveis no s'han de conèixer com implementa un servei la seva lògica de negoci, es distingirà entre la seva interfície (Service) i la implementació (ServiceImpl). L'accés a un servei sempre es realitzarà mitjançant la interfície i no utilitzarà directament la implementació.

Els Controllers de la capa de distribució accediran als Service de la capa de negoci mitjançant el mecanisme Dependency Injection de Spring, que injectarà en els Controllers els Service que siguin necessaris.

Els paràmetres i resultats de la execució dels serveis contindran la informació del model lògic de la aplicació i podran ser objectes del domini o tipus primitius.

Cal referir-se a les especificacions de Canigó 3.4, tant als serveis *core* com als relacionats amb negoci o integració per gran part dels serveis relacionats amb la capa de negoci.

Aquest cas d'ús inclou altres casos d'ús arquitectònics (veure diagrama de casos per aquesta capa lògica).

- CU_ARQ010 Accés a BBDD

Cal referir-se a les especificacions de Canigó 3.4, tant als serveis *core* com als relacionats amb accés a dades persistents (Servei de Persistència) per gran part dels serveis relacionats amb la capa de servei de dades.

Els components de negoci interactuen amb els components d'accés a base dades per tal d'obtenir els objectes que manipulen. Qualsevol accés a dades que es vulgui fer des de negoci haurà de passar per aquesta capa.

Els serveis són els propietaris de les seves dades. Tot servei de dades està amagat per defecte darrere un servei de negoci. En els casos que requereixen la consulta de dades compartides d'altres serveis directament a la capa de dades, es resoldrà mitjançant les estratègies definides a l'apartat 4.2.3 Interfícies amb sistemes interns (capa de dades).

Tota la interacció amb les dades estarà implementat amb Canigó 3.4 que incorpora el *Servei de Persistència en MongoDB*, que està basat en Spring Data i el patró DAO *(Data Access Object)*. L'objecte de negoci implementa la lògica de les operacions funcionals amb persistència mitjançant crides al DAO que pertoqui. Aquestes classes a més de fer l'accés a base de dades, també fan un primer tractament de les excepcions de base de dades produïdes.

- CU_ARQ010.1 Accés a BBDD pròpia

Per accedir a les dades propietàries del mòdul es configura la connectivitat principal a la BBDD MongoDB amb les dades definides per propietat i que s'indicarà via YAML de configuració, o ConfigMap d'OpenShift

Aquesta configuració serà tractada per classes amb anotació **@Configuration** incloses en la llibreria comuna **jus-canigo3.4-cloud-lib** , addicionalment en la mateixa llibreria s'ha creat la classe **justiciaMongoGenericDAO** que permet, mitjançant les funcionalitats bàsiques de lectura i escriptura, l'accés a base de dades. El programador en cap moment s'ha de preocupar d'obrir o tancar connexions, sessions o transaccions programàticament, sinó que aquesta funcionalitat s'ha delegat declarativament en el framework.

- CU_ARQ010.2 Accés a BBDD de consulta

La BBDD de consulta serà una BBDD consultable pels serveis que ho requereixin on hi haurà dades de les diferents BBDD dels serveis. Aquestes dades poden estar normalitzades o replicades i les podrem trobar com a col·leccions independents amb estructures iguals o similars que a les bases de dades dels serveis, o bé les podem trobar amb altres estructures. Podrem trobar dades desnormalitzades de diferents col·leccions agrupades en una, o també grans col·leccions que permetin obtenir d'una tacada conjunts de dades de diferents serveis en una sola consulta.

Aquesta BBDD permetrà realitzar consultes creuant diferents negocis.

- CU_ARQ011 Invocació a altres serveis síncron

Per tal de comunicar serveis entre si s'utilitzen els mecanismes propis de la plataforma Openshift (serveis) i els aportats per l'API Manager i el service Mesh

  - El Service Mesh serà emprat per optimitzar el funcionament de les aplicacions.
  - L'API Manager serà el component arquitectònic encarregat d'exposar les API's de les serveis del sistema cap a l'exterior amagant detalls de la implementació.

- CU_ARQ012 Interacció cap a sistemes externs (capa negoci)

Aquest cas d'ús quedarà detallat en els dos casos d'ús fill que el componen. Hi haurà dos possibilitats a l'hora de comunicar-se amb sistemes externs:

  - Sortida directa
  - Sortida via serveis que facin feina de fluxos d'integració

- CU_ARQ012.1 Accés a sistemes externs (no EJCAT)

Si un servei d'EJCAT+ ha de comunicar-se amb un servei extern podrà fer-ho de manera directa si el servei extern té un interfície REST. Les crides REST contra serveis externs haurien de ser dotades de tècniques de resiliència com timeouts per tal d'assegurar que un possible problema a un sistema extern i no controlat per nosaltres no afecta a l'estabilitat del nostre sistema.

En sistemes més antics que no puguin oferir serveis REST i s'hagin d'accedir amb altre tipus de protocol es podrà fer ús de SI (Spring Integration). Mitjançant SI es podran crear peces (serveis d'integració) encarregades de rebre peticions REST dels serveis de EJCAT+ i transformar aquestes peticions cap als protocols necessaris segons la plataforma o aplicació que s'hagi de consumir. A més de canvis de protocol amb SI es podran oferir serveis ampliats com WS-Security, WS-Attachments, MTOM...

- CU_ARQ012.2 Accés a sistemes externs (EJCAT via OSB)

Aquest cas d'ús és un cas concret del cas d'ús 12.1 en el que es deixa constància que per accedir a EJCAT caldrà fer-ho via OSB ja que és la seva porta d'entrada i sortida estàndard per comunicar-se amb sistemes externs.

- CU_ARQ012.2.1 Accés a TEMIS

Una gran part de les aplicacions EJCAT encara han de conviure amb TEMIS (antic sistema d'informació de Justícia codificat en PowerBuilder).

La interacció entre EJCAT i TEMIS es fa actualment de la següent manera:

- Des d'una aplicació JEE cap a TEMIS, es configurarà un datasource al servidor d'aplicacions JEE que accedirà a través de procediments PL/SQL a funcionalitat de TEMIS
- Des de TEMIS cap a una aplicació JEE, mitjançant la invocació de web services (directes o via OSB), o crides HTTPS a punts concrets de l'aplicació.

Les interaccions entre EjCat+ i TEMIS es farà de la següent manera:

- Des d'una aplicació desplegada en arquitectura Cloud no s'hauria d'accedir a funcionalitats de TEMIS. Però a cassos excepcionals amb una volumetria moderada es pot exposar un servei REST a l'OSB per tal que pugui ser cridat des de l'aplicació Cloud. El servei de l'OSB s'haurà d'encarregar de realitzar la transformació de les dades que sigui adient i executar les funcions emmagatzemades a la base de dades de TEMIS per tal que realitzin la tasca o cerca d'informació adient.
- Des de TEMIS cap a una aplicació desplegada en arquitectura Cloud, a cassos amb una volumetria moderada, TEMIS invoca a un servei de l'OSB que s'encarrega de realitzar una invocació d'ún servei REST exposat per la aplicació Cloud.

A nivell de dades sí que podria existir intercanvi d'informació entre les BBDD de TEMIS i les de EjCat+. Aquest intercanvi d'informació (sigui de TEMIS a EjCat+ o de EjCat+ a TEMIS)

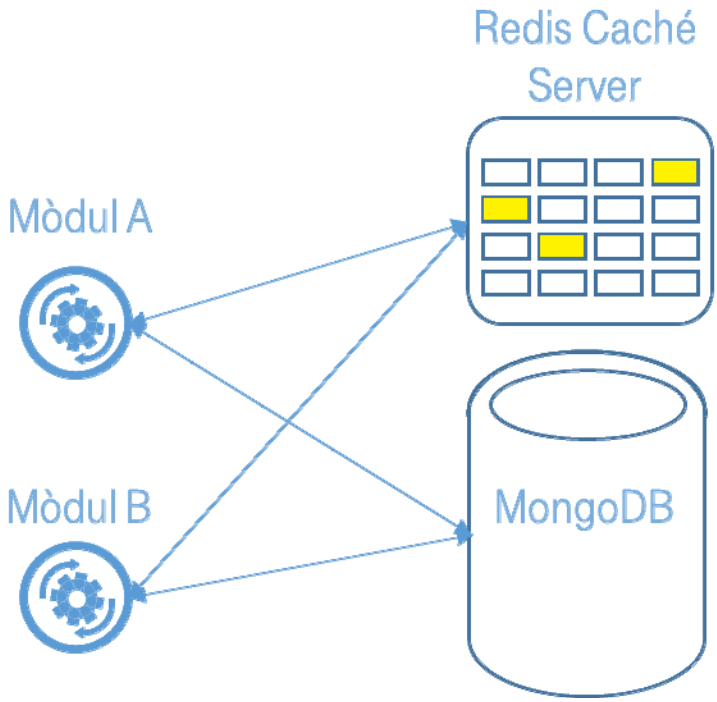
Per a més informació sobre cada accés es pot consultar el DTE corresponent de cada una de les aplicacions.

- CU_ARQ013 Cache

Dins el marc de les aplicacions de Justícia , existeix la necessitat de cachejar les respostes de les crides als serveis REST transversals.

Per aquesta necessitat, en la llibreria **jus-cache-canigo3.4-cache-cloud-lib** s'ha creat un T-Component **justiciaCacheExtDAO** que contindrà les funcions bàsiques en quant a utilització de la Redis Cache. L'ús d'aquest component s'encarregarà de cachejar les respostes diàriament per evitar un excés de crides als serveis RESTS

Aquesta llibreria per cachejar les respostes es connectarà a una instancia compartida de Redis Cache Server que es trobarà instal·lat en el Openshift, l'ús de la cache compartida evitarà que les dades puguin ser diferents en cada memòria cau, com podria ocórrer amb l'emmagatzematge en cache privat. L'emmagatzematge en memòria cau compartit garanteix que diferents instàncies d'una aplicació veuen la mateixa vista de dades en la caché. Per a això, es publicarà com a part d'un servei independent:



Addicionalment, aquesta llibreria permet que qualsevol mòdul pugui importar o crear cacheManagers per si el mòdul te una necessitat especifica de cachejar serveis propis.

Per defecte la llibreria tindrà dos cacheManagers configurades i amb la possibilitat de ser utilitzades per qualsevol mòdul que així ho requereixi:

cacheManagerHour: Cache Manager configurada a 1 hora. Al utilitzar aquesta cacheManager, totes les entrades expiraran al cap de una hora.

cacheManagerDay: Cache Manager configurada a 1 dia . Al utilitzar aquesta cacheManager, totes les entrades expiraran al cap de una hora.

- CU_ARQ014 Editor de documents

L'editor de documents és un A-Component (component funcional) de tipus JavaWebStart que s'executa en client i que és l'encarregat de editar cert tipus de documents que es generen a Justícia, en què l'usuari resol tot una sèrie de marques per tal de donar-li contingut al document.

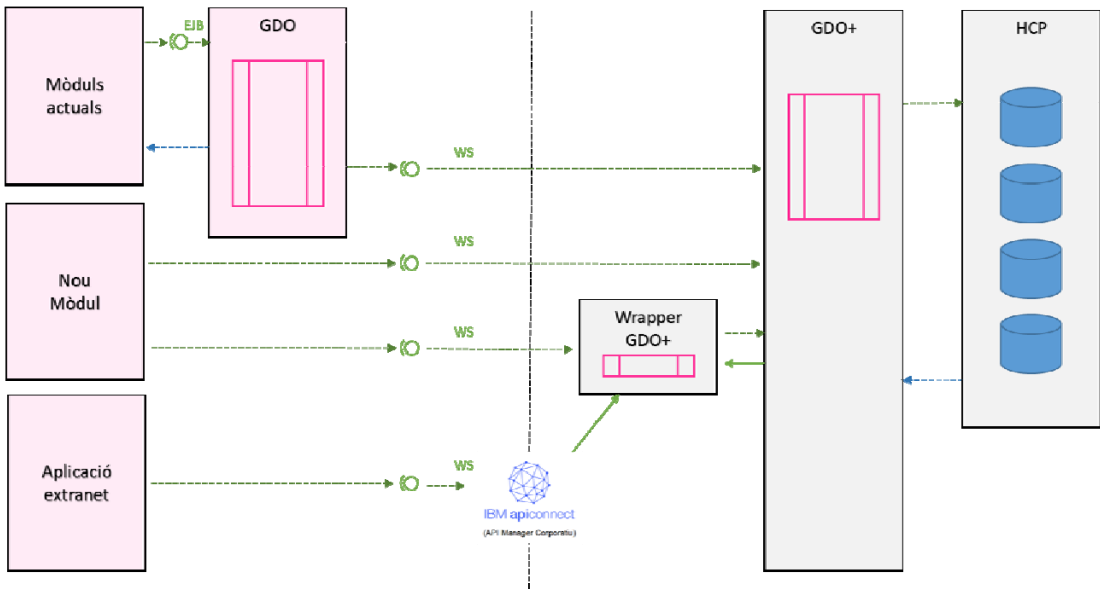
Aquests documents és connectaran amb LibreOffice (veure el CU_ARQ014.2) per tal de transformar-los al format definitiu que s'emmagatzemarà dins de HCP (CU_ARQ014.1)

- CU_ARQ014.1 Accés a HCP

Als sistemes d'informació de Justícia existeixen una gran quantitat de mòduls que necessiten interactuar amb un repositori documental. Històricament el repositori documental ha estat un servidor Documentum que en la nova arquitectura Cloud serà substituït per un HCP de Hitachi.

Es delegarà l'accés centralitzat a HCP a un altre mòdul funcional (A-Component desenvolupat en Canigo3.4 i desplegat al Cloud) anomenat GDO+ (Gestor Documental). Aquest mòdul utilitzarà l'API REST de HCP per tal d'interactuar amb HCP.

Les noves aplicacions desenvolupades en arquitectura Canigo 3.4 Cloud Native accediran directament al GDO+ per descarregar i carregar documents. Les aplicacions existents que ja interactuen amb l'aplicació original GDO ho seguiran fent així i serà GDO qui interactuarà amb GDO+ per tal que interactuï amb HCP.



Es pot consultar el document DTE de GDO i GDO+ per ampliar el detall d'aquest cas d'ús.

- CU_ARQ014.2 Accés a LibreOffice

Per tal que el A-Component (mòdul funcional) de l'editor de documents pugui assolir tots els requeriments funcionals (es pot consultar el DFU del mòdul corresponent per ampliar els detalls sobre els requeriments funcionals) es necessari que es connecti a una aplicació de tercers per a realitzar la generació de documents, en aquest cas un LibreOffice.

A diferencia del cas d'ús de transformació de documents (CU_ARQ015.1) que també necessita una aplicació de tercers per generar documents, en aquest cas l'accés es realitza de manera local a les màquines dels usuaris, que ja porten pre-instal·lades un LibreOffice per permetre l'ús de l'editor.

Per veure com es realitza aquest accés des de l'editor, es pot consultar el DTE del mòdul corresponent on es trobaran tots els detalls sobre la interacció entre l'editor i el LibreOffice.

- CU_ARQ015 Signatura Electrònica

Certs mòduls de Justícia necessiten serveis de signatura electrònica. Principalment:

  - Signar amb certificat d'usuari: s'utilitza un applet desenvolupat per CatCert a tal efecte
  - Signar amb un certificat d'aplicació: des de la capa de negoci d'un mòdul, s'invoca a un A-Component de tipus EAR anomenat SIG que actua de façana per a la interacció amb la plataforma PSIS de CatCert

- Validar un certificat d'usuari: com al cas anterior, s'utilitza la façana de SIG per a la interacció amb la plataforma PSIS de CatCert

Apart, existeix un altre A-Component de tipus EAR anomenat SSE (Safata Signatura Electrònica). Aquest mòdul interactua amb la plataforma PSA de CatCert, i proporciona funcionalitats de workflow de signatures (aprovació/denegació, multi-signatures, etc.).

Es pot ampliar la informació sobre la utilització de Signatura al document DTE del mòdul SIG. Es pot ampliar la informació sobre el mòdul Safata Signatura Electrònica al DTE del mòdul SSE.

- CU_ARQ015.1 Transformació de documents

La gran majoria de documents que es generen a Justícia són en format PDF, ja siguin documents generats per les aplicacions o documents adjuntats per l'usuari. La transformació d'aquests documents del format original (RTF, Word, Excel, ...) al format demanat PDF es realitza mitjançant un sistema extern de transformació de documents, al qual es fa arribar el document original i aquest el retorna transformat a PDF.

Per temes d'optimització de consum de memòria i temps de transformació, es fa una separació entre documents originals en format Windows (Word, Excel) i documents que poden ser transformats en UNIX (RTF, ODT). Aquesta separació es totalment transparent per l'usuari, ja que la realitza l'OSB depenent del format original del document a transformar.

Es delegarà l'accés centralitzat per la transformació de documents a un mòdul funcional (A-Component de tipus EAR) anomenat STD (Servei de Transformació de Documents). Aquest mòdul s'instal·larà dins el Oracle Service Bus (OSB) i s'invocarà via webservice.

Aquest mòdul STD proporciona les següents funcionalitats:

- Normalització de format a PDF a partir de documents ofimàtics
- Transformació de documents a partir d'un document pre-configurat mitjançant un sistema de plantilles
- Creació de codi segur de verificació del document
- Signatura digital del document normalitzat i transformat (delegant certes operacions de signatura en el mòdul SIG)

- CU_ARQ016 Procés Asíncron

S'ha de donar resposta a les possibles necessitats de comunicacions asíncrones entre serveis de l'arquitectura i a aquest apartat s'engloben els casos d'ús relacionats amb aquesta necessitat.

- CU_ARQ016.1 Accés a eines de comunicació asíncrona

Com eina per realitzar comunicacions asíncrones entre els serveis que formen part de la arquitectura es va seleccionar Kafka.

A la llibreria comuna **jus-canigo3.4-cloud-lib** existeix un T-Component que permet configurar i realitzar aquesta integració. S'ha realitzat una guia [CU_ARQ016.1 Accés a eines de comunicació asíncrona] per explicar la configuració necessària per realitzar comunicació asíncrona entre els serveis de la arquitectura cloud mitjançant l'eina Kafka.

- CU_ARQ017 Reporting

Dins el marc de les aplicacions de Justícia, existeixen diversos mòduls que generen documents, a partir de les dades existents a base de dades, per distribuir als usuaris.

Aquests documents generats, en format Excel o PDF es realitzen populant de dades les plantilles corresponents. S'ha establert un mecanisme per treballar amb JasperReport per definir aquestes plantilles i com s'utilitzen des de les aplicacions .S'ha realitzat una guia [CU_ARQ017 Reporting] de JasperReport per obtenir més informació de la interacció entre les aplicacions i els reports.

- CU_ARQ018 Monitorització

Les aplicacions poden exposar indicadors del seu comportament i estat (mètriques). Aquestes mètriques poden estar recollides per eines externes per tal d'alliberar a les aplicacions del seu tractament.

S'ha realitzat una guia [CU_ARQ018 Monitorització] on es descriu els passos per integrar la generació de mètriques de negoci a aplicacions i la seva posterior recol·lecció.

- CU_ARQ019 Enviament de correus

Per gestionar l'enviament de correu en la llibreria **jus-cache-canigo3.4-cloud-lib** s'ha creat un T-Component **JusticiaSimpleMailBuilder** que les aplicacions faran servir per enviar correus.

La configuració del component (host, port, .. .) es realitza amb ConfigMap d'OpenShift.

- CU_ARQ020 Logging

Per gestionar les traces que generen els nostres serveis utilitzarem els següents productes especialitzats en entorns cloud i basats en OpenTracing: l'stack EFK i el producte Jaeger.

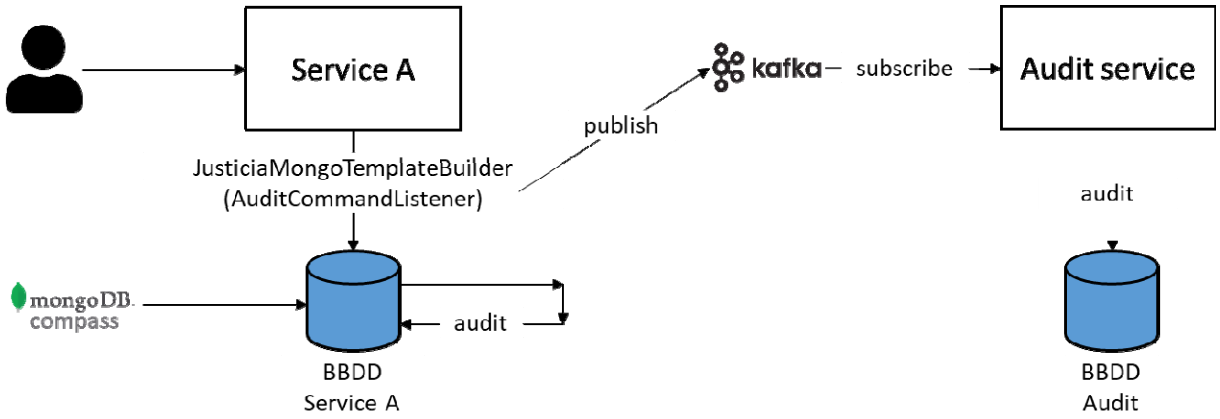
- CU_ARQ021 Auditoria

Dins el marc de les aplicacions de Justícia , existeix la necessitat d' auditar les operacions que realitzen els usuaris sobre la base de dades.

En la llibreria **jus-cache-canigo3.4- cloud-lib** s'ha creat un T-Component **AuditCommandListener** que audita totes les operacions a la base de dades, ja siguin de consulta com d'actualització i publica un missatge de tipus asíncron amb el detall de l'operació. En el missatge consta, entre altre informació:

- Nom de l'aplicació
- Base de dades sobre la que s'ha realitzat l'operació
- Col·lecció
- Usuari connectat a l'aplicació que ha realitzat l'operació.
- Data y hora
- Sentencia executada en format JSON,
- Resultat en forma JSON.
- Temps consumit per la sentencia.

S'ha creat el mòdul **justicia-audit-service** que processarà els missatges publicats per les aplicacions i els enregistra a base de dades.



- CU_ARQ022 Gestió d'excepcions

Tal com s'ha definit en altres arquitectures web similars de Justícia, definirem dos blocs diferenciats d'excepcions:

- Excepcions de negoci (checked): consistents en errors de validació o dades.
- Excepcions de runtime (unchecked): provocades per errors inesperats aliens al comportament funcional del servei: problema de comunicacions, base de dades, operacions de disc, etc...

Es definiran dos T-Components principals a mida per cada tipus d'excepció, i també classes a mida per gestionar de forma comuna la transformació de les excepcions en respostes a la crida REST efectuada.

- JusticiaBusinessException
- JusticiaSystemException
- JusticiaResponseEntityExceptionHandler

- CU_ARQ023 Actualitzar configuracions en calent

Segons la necessitat disposarem de dues eines diferents d'aconseguir fer canvis "en calent" sense haver de fer un canvi al codi.
Per tal d'aconseguir-ho podrem fer-ho:

- Via configmaps: Els serveis desplegats al cloud fan ús de descriptors de desplegaments, secrets, routes, services, configmaps ... Farem ús dels configmaps per aconseguir refrescar certes propietats que ens interessi
- Via cache: Permetrà agilitzar algunes parts del codi i a banda podrem tenir a bbdd algunes propietats que son susceptibles de ser canviades. Amb aquest sistema podem estalviar desplegaments i guanyar flexibilitat

- CU_ARQ024 Notificacions PWA (backend)

El component de notificacions Web Push és un cas d'ús principalment de frontend. Tanmateix, des del backend cal oferir una sèrie de funcionalitats en un T-Component comú de tipus servei, per gestionar-les dins el projecte EjCat+. Aquest servei oferirà:

- Manteniment de les peticions de subscripció Web Push, generades pel navegador de l'usuari, en cas que aquest accepti rebre notificacions del nostre projecte.

Aquestes subscripcions es desaran en una col·lecció MongoDB publicada a tal efecte, i posteriorment s'utilitzaran per propagar els avisos segons el grau d'especialització que es destigi: per usuari individual o per agrupació funcional d'usuaris

- Generació de les claus públiques i privades per autenticar les notificacions als diferents servidors de Web Push (Mozilla, Google Chrome, Microsoft Edge,...)

- Gestió de les peticions de baixa de subscripció. L'usuari, en qualsevol moment, pot indicar al navegador que vol deixar de rebre avisos de la plataforma, i en aquest cas, cal gestionar la corresponent baixa al sistema Web Push del navegador.

- CU_ARQ025 Transaccionalitat i SAGA

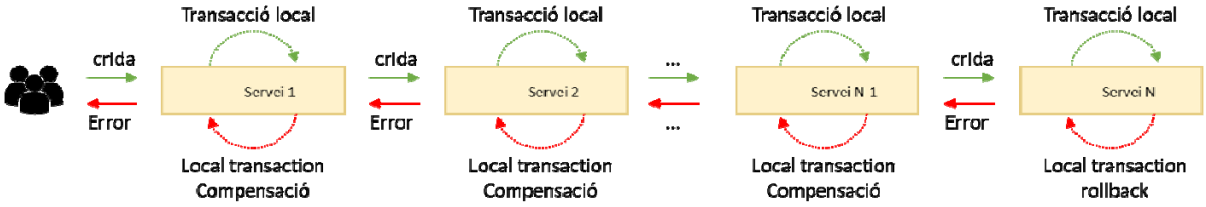
A contextos tradicionals, com poden ser les aplicacions JEE, transaccionalitat es pot entendre com equivalent a l'acrònim ACID utilitzant el protocol conegut com 2PC (two-phase commit) per realitzar transaccions ACID distribuïdes. A arquitectures encara més distribuïdes, 2PC no és una opció recomanable per motius de rendiment, i inclús no hi ha un suport consolidat a 2PC en protocols lleugers d'invocació REST. És una restricció coneguda a aquestes arquitectures que descarten l'ús de transaccions distribuïdes amb propietats ACID.

El fet de no utilitzar transaccions distribuïdes de tipus ACID no treu que la necessitat de transaccionalitat distribuïda pugui existir. L'opció per aquesta necessitat a les arquitectures distribuïdes passa per l'ús de transaccions descrites per la noció de consistència eventual. En relació a la consistència eventual, comunament es parla de BASE: Basically Available, Soft estate, Eventually consistent.

Una aproximació per descriure la consistència eventual és considerar que el sistema no evoluciona entre dos estats consistents de dades de manera atòmica sinó que des de que s'inicia una transacció, els sistemes poden passar per altres estats visibles, potencialment no consistents fins que s'arriba a un nou estat final, que mostra correctament la transacció completament aplicada. De la mateixa forma, la cancel·lació dels canvis (a ACID: rollback), no és atòmica sinó també subjecte a consistència eventual, en el que no necessàriament ens porta a un estat idèntic a l'estat inicial abans de començar la transacció. És per això que no es parla del terme rollback i es parla de compensació.

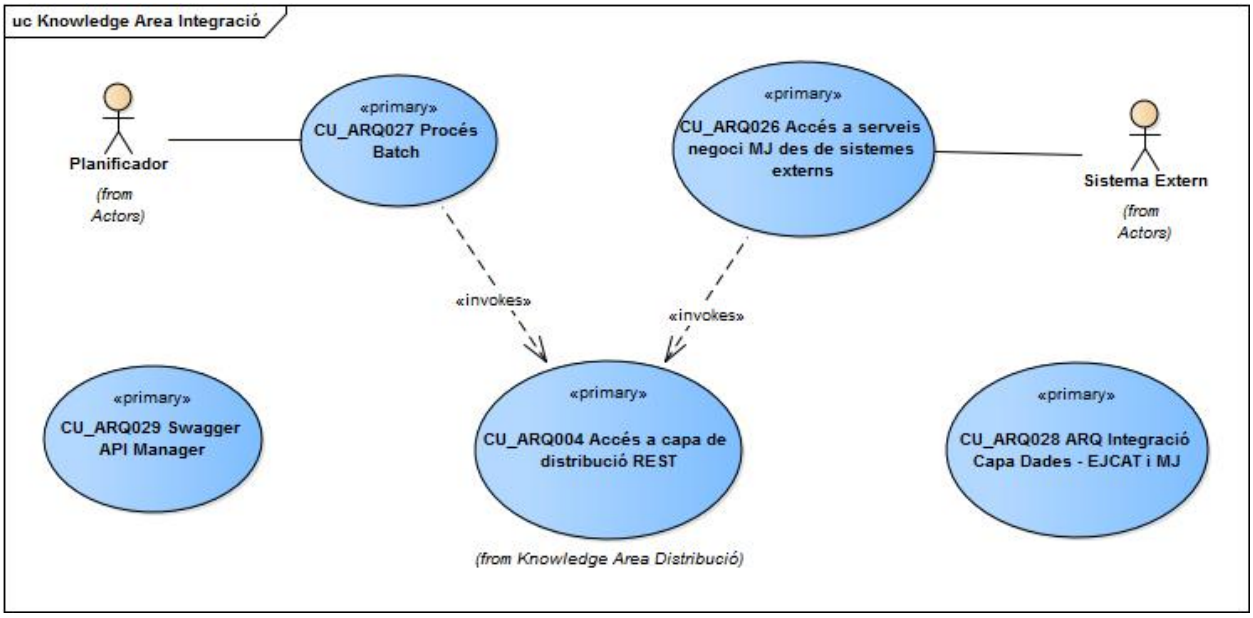
Una SAGA és una seqüència d'operacions que realitzen una unitat de treball específica i que generalment es troben intercalades entre sí. Cada operació que forma part d'una SAGA es pot "revertir" mitjançant una acció de compensació. La SAGA vol garantir que totes les operacions es completin correctament o s'executin les accions de compensació que siguin adients (per a totes les operacions executades) per revertir qualsevol treball realitzat anteriorment.

S'ha realitzat una guia [CU_ARQ025_Transaccionalitat_i_SAGA] on es descriu el patró SAGA.



Esquema teòric SAGA amb compensacions

2.3.3 Casos d'ús de la capa de integració



- CU_ARQ026 Accés a serveis negoci dins del cloud de justícia des de sistemes externs

En arquitectures de serveis és habitual tenir un API Manager que faci funcions de façana dels nostres serveis cap a l'exterior.

En l'arquitectura Canigo 3.4 + Angular 9 algunes de les comunicacions d'entrada faran ús de l'Api Manager Corporatiu IBM Api Connect. Aquest Api Manager serà la porta d'entrada de sistemes externs a Justícia que vulguin cridar serveis interns, quan aquests sistemes que criden estiguin fora del que podria ser considerat una "xarxa o connexió segura"

Les funcions principals de l'Api Manager seran:

1. Proveint el nostre sistema d'una capa externa de seguretat
2. Realitzant funcions d'enrutament cap a sistemes interns

Quines comunicacions no faran ús de Api Manager?

1. Comunicacions entre serveis desplegats a Openshift
2. Comunicacions cap a serveis interns a Openshift que vinguin del que podria ser considerat elements interns com podrien ser altres aplicacions del Departament de Justícia allotjades a la Intranet
3. Comunicacions cap a serveis exteriors (Api Manager només serà porta d'entrada, no de sortida)

- CU_ARQ027 Procés Batch

Una necessitat habitual de les aplicacions és l'execució d'un procés o tasca de manera planificada, ja sigui a uns moments determinats o d'una manera periòdica (cada cert període de temps).

Es va escollir ShedLock com mecanisme predeterminat per la planificació de tasques, sense descartar altres possibilitats com Control-M com mecanisme extern de planificació de tasques.

S'ha creat una guia [CU_ARQ027 Processos Batch] per descriure com les aplicacions han de procedir respecte als processos planificats.

- CU_ARQ028 ARQ Integració Capa Dades

Les dades són propietàries del servei que sigui el responsable (si s'ha realitzat disseny dirigit al domini, només un servei serà el responsable d'un determinat domini) i si un tercer necessita aquelles dades podrà consultar-les via invocació de serveis de negoci. Malgrat això existiran escenaris on no serà recomanable obtenir aquestes dades via invocació de serveis com per exemple:

- Volum molt elevat de dades a retornar per part del servei
- Elevat número de peticions al servei de consulta
- Necessitats derivades de la desnormalització de base de dades
- Manteniment de la coherència de dades entre sistemes diferents.

A aquests escenaris, la integració a capa de dades consisteix en realitzar una obtenció de dades, directament des d'un repositori de dades (sense utilitzar la capa de negoci per accedir a les dades) per tal de poder comunicar aquestes dades a un altre sistema que els rep i processa – o adapta – per actualitzar el seu propi repositori de dades (sense utilitzar la capa de negoci per actualitzar les dades).

En alguns escenaris d'integració de dades pot ser necessària una lògica de negoci per donar context a les dades obtingudes, així com una lògica de negoci per tal de poder adaptar la informació a les necessitats i característiques del sistema que rep les dades i per tant poden existir etapes de transformació o ampliació de la informació origen per ser enviada amb context cap a serveis destí o interessats.

S'ha realitzat una guia [CU_ARQ028 ARQ Integració Capa Dades] per descriure els escenaris d'integració de dades i les seves característiques.

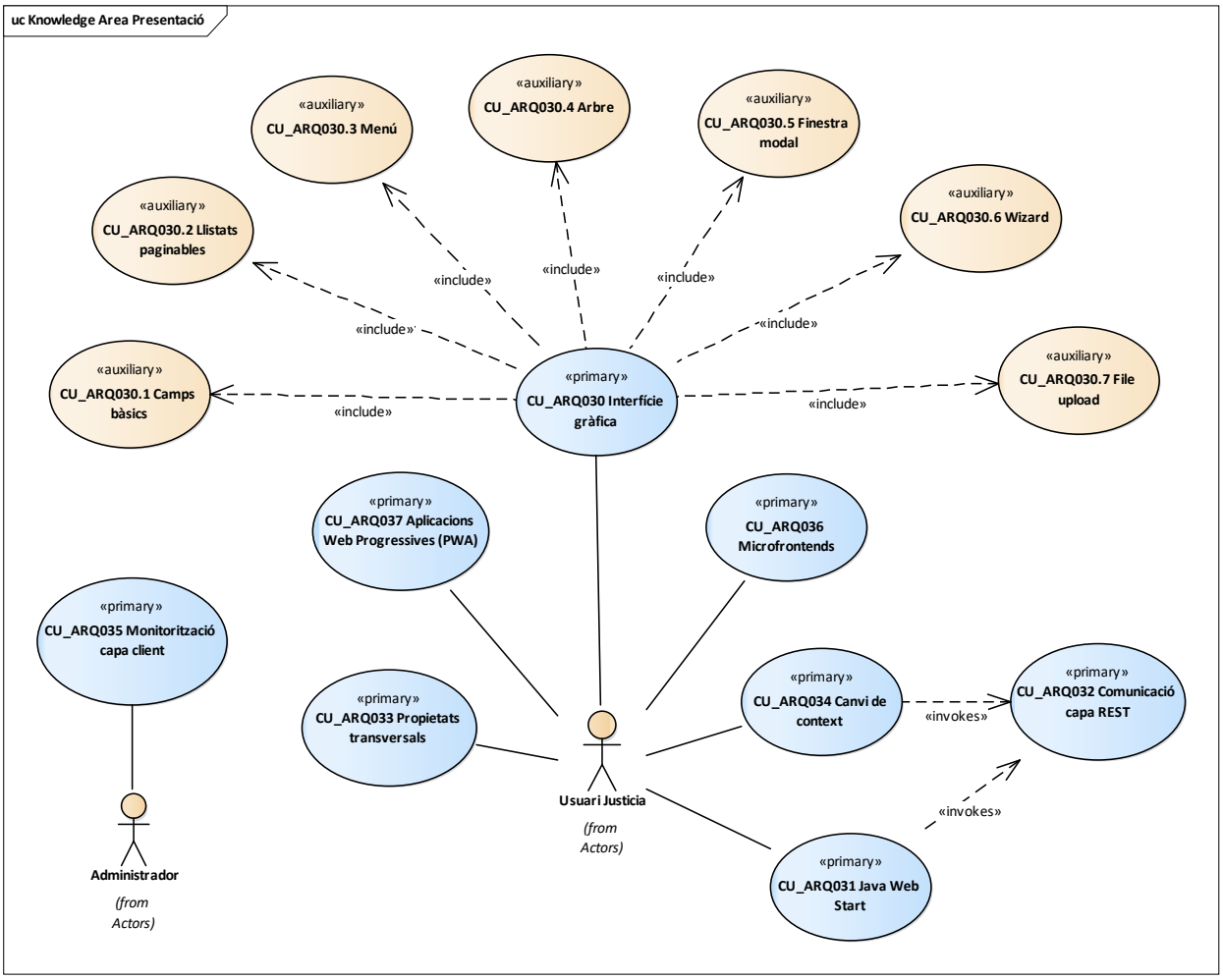
- CU_ARQ029 Swagger API Manager

En aquest punt es descriu com publicar els serveis en un API Manager a partir de la informació de Swagger.

Per aquest objectiu, es requereix tenir prèviament els serveis REST de Backend publicats a traves de Swagger, amb aquests serveis desplegats, durant la creació i publicació de la API a l'API Manager se li indicarà la URL del Swagger del servei de backend.

2.3.4 Casos d'ús de la capa de presentació

Els casos d'ús d'aquesta capa són els següents:



- CU_ARQ030 Interfície gràfica

La interfície gràfica de l'aplicació client Angular 9 estarà basada en els estàndards Web HTML versió 5, CSS versió 3 i llibreries JavaScript.

El disseny de la interfície d'usuari serà **responsive** basat en el framework CSS **Bootstrap**, de manera que l'experiència d'usuari sigui la mateixa des de la web, tauleta o mòbil. L'arquitectura, les metodologies i els components de desenvolupament a utilitzar es defineixen en les guies de desenvolupament de frontend basades en el framework Angular. Altres necessitats s'estudiaran de cas en cas prèviament a l'inici del desenvolupament.

ⓘ Més informació sobre el framework Boostrap: https://getbootstrap.com/docs

- CU_ARQ030.1 Camps bàsics

Com a camps bàsics, tant d'entrada de dades com per a mostrar informació a l'usuari, es faran servir els controls inclosos a la llibreria específica per Angular PrimeNG. 9 Els camps bàsics inclouen camps de text, quadres de llista, quadres combinats caselles de selecció, botons, botons d'opció i calendaris.

- CU_ARQ030.2 Llistats paginables

Per als llistats paginables es faran servir conjuntament els controls DataTable i Paginator inclosos a la llibreria PrimeNG.

- CU_ARQ030.3 Menú

Per als menús de l'aplicació paginables es faran servir els controls específics inclosos a la llibreria PrimeNG. Els controls de menú inclouen menús contextuals, menú vertical i menús multinivell.

- CU_ARQ030.4 Arbre

Per als controls de tipus arbre es farà servir el control específic Tree inclòs a la llibreria PrimeNG.

- CU_ARQ030.5 Finestra modal

Per a les finestres de tipus modal de les aplicacions Angular es farà servir el control específic Dialog inclòs a la llibreria PrimeNG.

- CU_ARQ030.6 Wizard

Pels components de tipus Wizard de les aplicacions Angular es farà servir el control específic Accordion inclòs a la llibreria PrimeNG.

Aquest component mostra panells que poden ser col·lapsats. Es mostrà un panell inicial on introduiran les dades bàsiques i només es mostraran els següents panells (passos) quan s'hagi emplenat la informació necessària.

Una vegada s'han guardat les dades del primer panel (pas 1) la resta de panells es van mostrant segons les dades necessàries complimentades als panells anteriors (això dependrà de la lògica de negoci de cada aplicació). Una vegada carregats els panells, sempre serà possible accedir al qualsevol panell de manera immediata.

- CU_ARQ030.7 File upload

Per a penjar fitxers al servidors des de la capa client Angular (file upload) es farà servir el control específic FileUpload inclòs a la llibreria PrimeNG.

- CU_ARQ031 Java Web Start

Java Web Start és un mecanisme de Java que permet executar aplicacions en local publicades en enllaços web i amb l'habilitat d'actualitzar-se automàticament. Es va dissenyar com una tecnologia alternativa als applets pels casos on la funcionalitat està completament dins l'aplicació.

Quan es descarrega, queda en la caché de Java, a l'apartat d'aplicacions i es pot tornar a executar. Hi ha la opció de crear accessos directes a aquesta aplicació.

Cal tenir en compte que les aplicacions JNLP no només s'inicien des d'una URL, sinó que també es mantenen automàticament actualitzades perquè cada cop que s'executa es pot comprovar si el JNLP o els JAR han canviat i actualitzar-se. A més, com és caché java no es tenen problemes de permisos d'instal·lació ni res.

També és possible fer pre-instal·lacions executant la següent comanda: javaws -import -codebase file://c:/tmp c:\tmp\prova.jnlp

Les aplicacions Java Web Start s'executen sense necessitar cap navegador i per tant no hi interactuen. Això fa més difícil l'ús de Java Web Start com a substitut dels applets però hi ha els següents escenaris de comunicació depenent del grau d'integració que es desitgi.

ⓘ Si tan sols necessitem enviar dades en una única crida cap al servidor, aquesta es pot realitzar directament per l'aplicació JWS. Un paràmetre del jnlp podria indicar una URL on cridar quan finalitza l'acció i aquesta realitzaria l'acció o accions necessàries.

ⓘ Si es vol interacció amb la pàgina del navegador, es pot implementar una crida AJAX des de javascript cap a l'aplicació JWS. En obrir la pàgina, aquesta intentaria (amb reintents i un timeout petit) connectar-se en un port TCP predefinit, i un cop connectat, esperaria resposta. En l'aplicació JWS, a questa connexió s'obriria en un fil paral·lel on esperaria en un bucle que finalitzés l'acció. Quan aquesta ha finalitzat, el fil enviaria la resposta, que la rebria el navegador.

- CU_ARQ032 Comunicació capa REST

Tota la comunicació de la capa de presentació amb la capa backend REST del servidor estarà centralitzada en un únic servei de comunicacions de l'aplicació. Per la seva banda, aquest servei farà servir el mòdul inclòs a les llibreries bàsiques d'Angular específic per a comunicacions HTTP **@angular/common/http**. Les comunicacions faran servir el protocol HTTP/S i les dades intercanviades seran en format JSON.

Les comunicacions amb la capa REST del servidor seran asíncrones i seguiran el patró dels **Observables** aplicat a fluxos de dades utilitzat a Angular (implementat a la llibreria de 'Reactive Extensions' o 'RxJS'). D'aquesta forma, es tractarà tot tipus d'informació intercanviada amb la capa REST del servidor com un flux (stream) 'observable' d'entrada i de sortida, al que se li poden agregar operacions que processen les dades.

En les comunicacions amb el servidor en que requereixi autentificació, el servei de l'aplicació Angular inclourà en el Header de la petició HTTP corresponent (Request) el token JWT obtingut durant el procés de login. Veure apartat 5.3.5.3 Intercanvi de JWT entre client i servidor per informació detallada.

- CU_ARQ033 Propietats transversals

Veure apartat 4.5. Propietats transversals del sistema per a més informació.

- CU_ARQ034 Canvi de context

En ocasions la capa de presentació d'un mòdul necessita accedir a la capa de presentació d'un altre. Certa informació de context ha de ser traspassada entre aplicacions de manera transparent per l'usuari, i a més es desitja cert control d'accés i auditoria.

En el cas de les aplicacions Angular dels sistemes de EjCat+ s'implementaran les funcionalitats necessàries per a cadascun dels escenaris de canvi de context entre aplicacions:

1. Aplicació Angular EjCat+ (Microfrontend) → Aplicació Angular EjCat+ (Microfrontend).
2. Aplicació Angular EjCat+ (Microfrontend) → Aplicació eJCAT.
3. Aplicació eJCAT → Aplicació Angular EjCat+ (Microfrontend).

- CU_ARQ035 Monitorització capa client

La monitorització de l'experiència real d'usuari al navegador (Real User Monitoring) en una aplicació Angular (Single Page Application) ens permet el següent:

- Detectar i solucionar problemes d'aplicacions Angular dins del context de la càrrega inicial de la pàgina.
- Ajudar a les decisions de negoci mitjançant l'anàlisi de dades de les aplicacions Angular a través d'anàlisis de les mesures obtingudes.
- Permet els desenvolupadors crear aplicacions amb un rendiment òptim.

Angular proporciona una sèrie d'eines com Router events, Component Decorators i LifeCycle Hooks, Property Decorators i HTTP Interceptors que, conjuntament amb la utilització de **l'API Web Performance API** ens permeten d'obtenir tota la informació sobre el rendiment de l'aplicació respecte a temps de càrrega i comunicacions.

Performance API proporciona accés a informació relacionada amb el rendiment per a la pàgina actual en el navegador. Es part de la High Resolution Time API, però està millorada per la Performance Timeline API, la Navigation Timing API, la User Timing API i la Resource Timing API.

Perfume.js ens permet fer servir la Performance API d'una forma molt senzilla. Perfume.js aprofita totes aquestes API de rendiment que ens permeten recopilar mètriques per a desenvolupar una comprensió més profunda de com els usuaris perceben el rendiment web de l'aplicació.

Per tal de fer servir les funcionalitats que ofereix **Perfume.js** i les **Performance API** en el context d'una aplicació Angular s'ha creat un **mòdul Monitoring dins del framework ra-ng** que inclou una sèrie de serveis, decorators i interceptors que ens faciliten la recollida de les dades de monitorització en una aplicació Angular.

- CU_ARQ036 Microfrontends

El terme **Microfrontend** es refereix a una aproximació d'arquitectura de desenvolupament d'aplicacions web com una composició de aplicacions frontends "petites" respecte a la aplicació sencera que es necessita construir. Les funcionalitats es poden dividir per dominis que són gestionats per microfrontends especifics, auto continguts i que poden ser desenvolupats i desplegats el més independentment possible.

La solució de microfrontends en les aplicacions Angular estarà basada en una implementació de **Web components (Custom Elements)** i en concret en la seva basant Angular, **Angular Elements**.

No tots els mòduls de l'aplicació han de construir-se en forma de Microfrontend. Aquesta solució s'aplicarà en aquells casos en que el desenvolupament d'un domini de negoci requereixi d'un cicle de vida diferent al de l'aplicació principal o bé la tecnologia o framework utilitzada sigui diferent de la de l'aplicació principal.

- CU_ARQ037 Aplicacions Web Progressives (PWA)

Es pot pensar en una PWA com un lloc web però que actua i es comporta com una aplicació. La disponibilitat dels anomenats service workers i en les API de Cache i Push donen la possibilitat als desenvolupadors web de permetre als usuaris instal·lar aplicacions web en el propi dispositiu, ja sigui un ordinador, tableta o mòbil, rebre notificacions push i fins i tot treballar sense connexió.

En el cas de les aplicacions Angular s'afegirà la dependència **@angular/pwa** als projectes. Aquest procés afegeix les llibreries, configuracions i fitxers de recursos necessaris per a implementar les funcionalitats que proporcionen les PWA en aplicacions basades en Angular.

3 CONVENCIONS I RESTRICCIONS GENERALS

3.1 CONCEPTES I COMPONENTS

3.1.1 Arquitectura de Referència

Pel que fa a les aplicacions Angular, es seguirà, a nivell general, l'especificació definida a la documentació oficial d'Angular definida per Google: https://angular.io/guide/styleguide

3.1.2 Serveis, Components, Frameworks, Llibreries

3.1.2.1 Front-end d'aplicació

Pel que fa a la capa de presentació, els principals components i versions serien:

Component	Versió	Descripció
@angular/common	9.1	Directives i serveis d'Angular habitualment necessàries.
@angular/core	9.1	Llibreries bàsiques del framework d'Angular.
@angular/forms	9.1	Directives i serveis d'Angular especialitzades en la creació i gestió de formularis (forms).
@angular/router	9.1	Llibreries necessàries per a controlar la navegació i enrutament en el context d'una aplicació Angular.
@angular/cdk	9.1	Representa una abstracció de les funcionalitats centrals que es troben a la llibreria Angular Material (necessari per primeng).
@angular/elements	9.1	Implementació de HTML Custom Elements per a aplicacions Angular.
@angular/service-worker	9.1	Llibreries per a la implementació de Service Workers a aplicacions Angular.
document-register-element	1.14.3	Implementació de HTML Custom Elements.
justicia-ng	9.0	Serveis i components transversals per a aplicacions Angular de Justícia.
log4javascript	1.4.15	Llibreria JavaScript necessària per a la generació de informació de log.
moment	2.24	Llibreria per a validar, manipular, i mostrar dates i hores.
ngx-translate	12.1.2	Llibreria per a la internacionalització (i18n) d'aplicacions Angular.
primeng	9.0	Col·lecció de components d'interfície d'usuari per Angular.
ra-ng	9.0	Framework per a propietats transversals d'Aplicacions Angular.
rxjs	6.5.4	Conjunt de llibreries en JavaScript per desenvolupar programes asíncrons i basats en events.
zone.js	0.10.2	Llibreria necessària per a Angular que gestiona el seu context d'execució.
perfume.js	5.1.0	Llibreria per a la monitorització del rendiment web.
ngx-build-plus	9.0.6	Llibreria per a estendre les funcionalitats d'Angular CLI.

3.1.2.2 Back-end d'aplicació

El backend de l'aplicació estarà principalment basat en la versió 2.1.8 del Framework Spring Boot i en la versió 3.4 del Framework Canigó del CTTI.

Les llibreries més rellevants de backend són:

Component	Versió	Descripció
canigo.core	4.2.0	Llibreries base del Framework Canigó 3.4
canigo.web.core	2.2.0	
canigo.web.rs	2.2.0	
canigo.persistence.mongodb	2.3.0	
spring-boot-starter-logging	2.1.8	Spring Logback
spring-boot-starter-web	2.1.8	Spring Web MVC
spring-boot-starter-security	2.1.8	Spring Security amb OAuth2 Resource Server
spring-boot-starter-oauth2-resource-server	2.1.8	
spring-data-mongodb	2.1.8	Spring Data MongoDB
spring-boot-starter-data-mongodb	2.1.8	
mongodb-driver-legacy	3.12.3	Drivers de MongoDB adaptats a 3.12, compatibles amb Canigó
mongodb-driver-sync	3.12.3	
mongodb-driver-core	3.12.3	
springfox-swagger2	2.7.0	Llibreries per suport a Framework Swagger 2
springfox-swagger-ui	2.7.0	
spring-boot-starter-actuator	2.1.8	Endpoints de mètriques Actuator i exportador en format Prometheus
micrometer-core	1.1.6	
micrometer-registry-prometheus	1.1.6	
opentracing-api	0.33.0	OpenTracing, gestió distribuïda de logs en contenidors
opentracing-spring-jaeger-web-starter	3.1.2	
opentracing-spring-cloud-starter	0.3.12	
jaeger-client	1.2.0	
spring-cloud-stream	2.1.0	Publicació i consumició de Kafka Topics (operacions asíncrones)
spring-cloud-starter-stream-kafka	2.1.0	
shedlock-spring	4.14.0	Planificador de tasques en cloud
lombok	1.18.8	Simplificació de codi
jus-canigo3.4-cloud-lib	TBD	Llibreries de components comuns
jus-canigo3.4-cache-cloud-lib	TBD	Llibreria de components del CU_ARQ13 – Cache
justicia-jasper-fonts	1.0.0	Fonts més comuns utilitzades per jasper-report

3.1.3 Bones Pràctiques de la Tecnologia de Referència

Pel que fa a les aplicacions Angular, es seguiran les bones pràctiques especificades a la documentació oficial d'Angular definida per Google: https://angular.io/guide/styleguide

3.2 ALTRES CONVENCIONS I RESTRICCIONS GENERALS

3.2.1 Normatives de programació

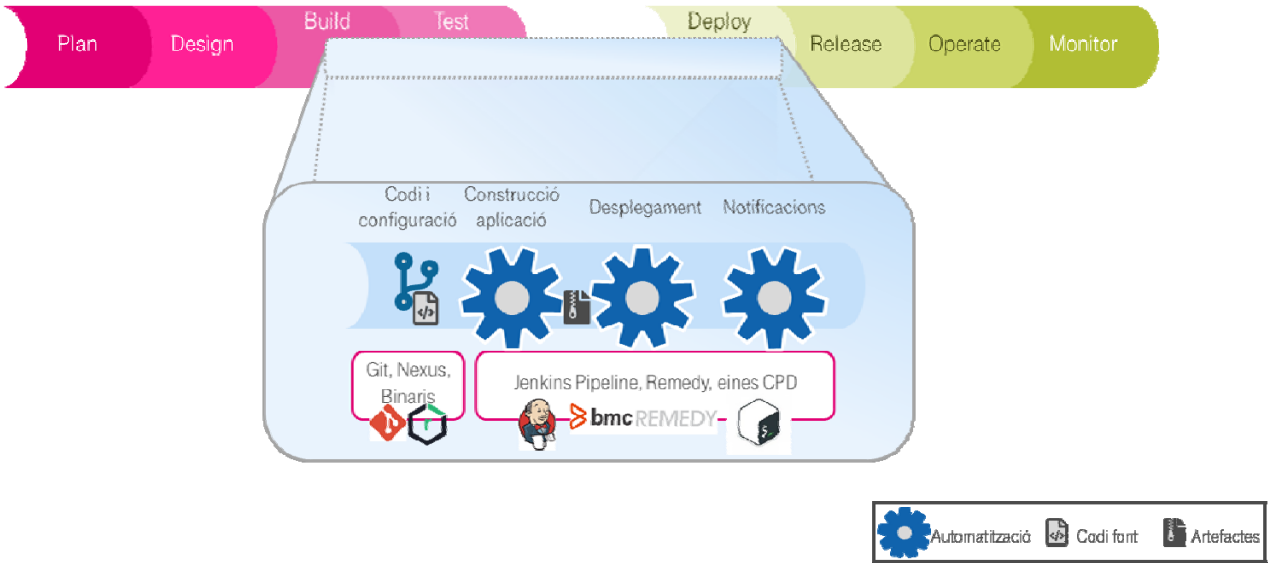
Al projecte es defineixen les següents normatives sobre la capa de negoci:

- Java anguage specification https://docs.oracle.com/javase/specs/jls/se8/html/index.html
- Arquitectura CTTI APIs RESTful: millors pràctiques https://canigo.ctti.gencat.cat/blog/2016/01/api/

3.2.2 Gestió de la configuració

Els entregables es deixen generalment a un gestor documental del Dept. anomenat PORTIC, malgrat això, aquest punt s'ha d'acordar per cada projecte.

La entrega de codi a client, la construcció i el desplegament des farà segons normatives del SIC corporatiu, consultables a https://canigo.ctti.gencat.cat/sic/



3.2.3 Procés de desenvolupament

Per a entregables del client, es seguirà la metodologia MQS pels entregables (gestió de projecte i enginyeria del software). Hi ha plantilles de client definides per a cada tipus d'entregable.

3.2.4 Eines de Desenvolupament i Àrea de Treball

S'ha generat una guia per tal de poder preparar un entorn de desenvolupament per part dels desenvolupadors.

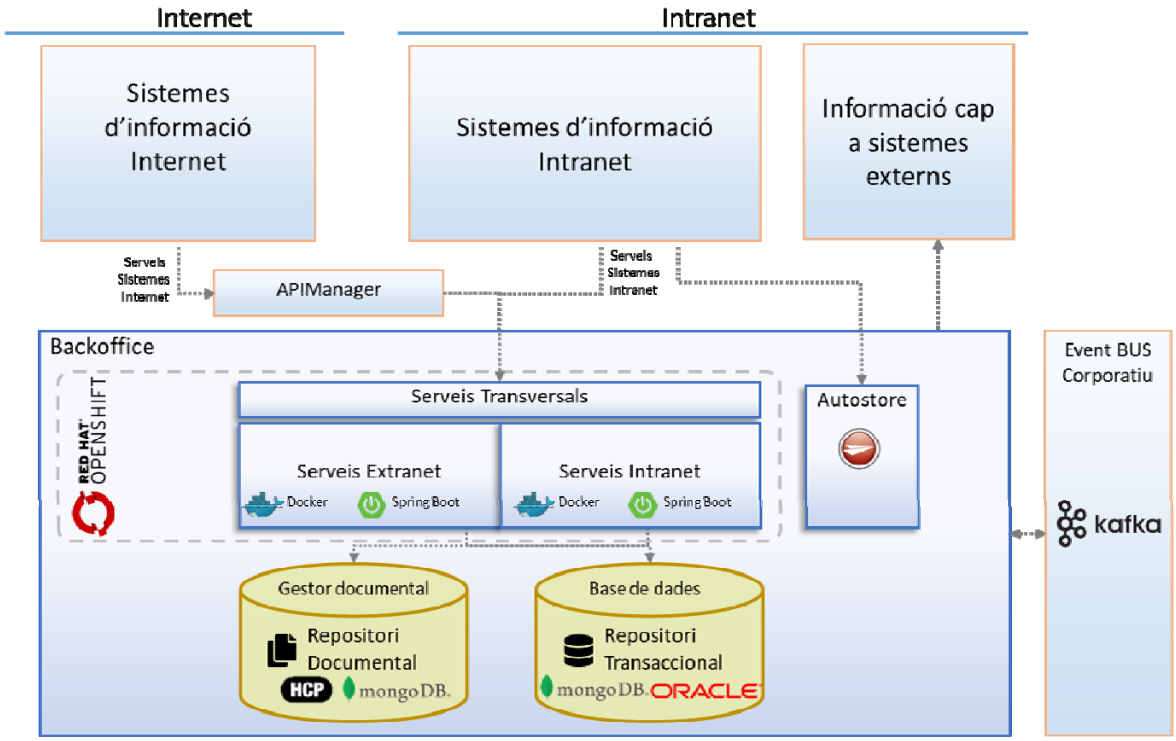
S'han d'utilitzar les eines acordades per part dels proveïdors amb CTTI i el Departament de Justícia a partir de les definides de manera corporativa a les webs oficials (https://canigo.ctti.gencat.cat/sic/ i MQS-Eines: https://qualitat.solucions.gencat.cat/eines/).

4 ESPECIFICACIÓ D'ARQUITECTURA

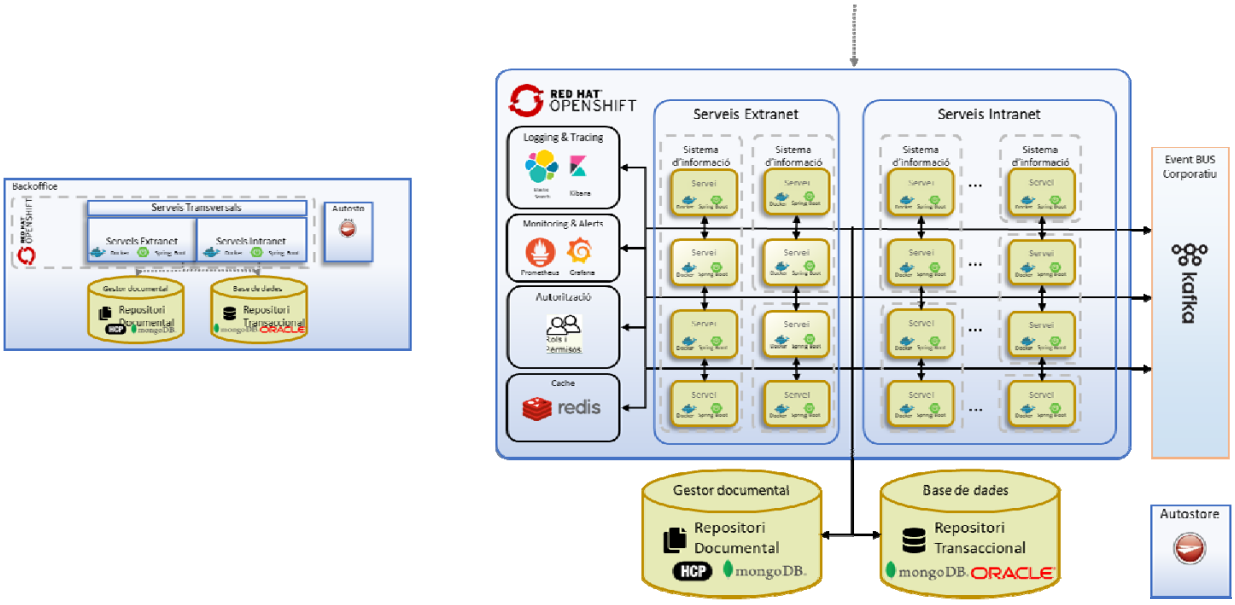
4.1 VISTA GENERAL

La vista general mostra les parts del sistema més importants des del punt de vista funcional i tècnic, a alt nivell. Mostra com les parts del sistema son distribuïdes a través dels elements de la infraestructura tècnica (TI).

El següent diagrama mostra les capes i tecnologies a alt nivell més rellevants de l'arquitectura JEE del Departament



El següent diagrama mostra el mapa de serveis



Els serveis al sistema es comunicaran entre si via invocació de serveis. La comunicació de l'exterior cap a negoci del sistema es farà via API Manager.

4.2 VISTA DE CONTEXT

La vista de context descriu el sistema en el context de tots els seus sistemes veïns.

En aquest punt s'especificarà com la arquitectura es relaciona amb sistemes externs, tant en una direcció en relacions que son d'entrada al sistema, com en l'altra direcció on les relacions son de sortida del sistema.

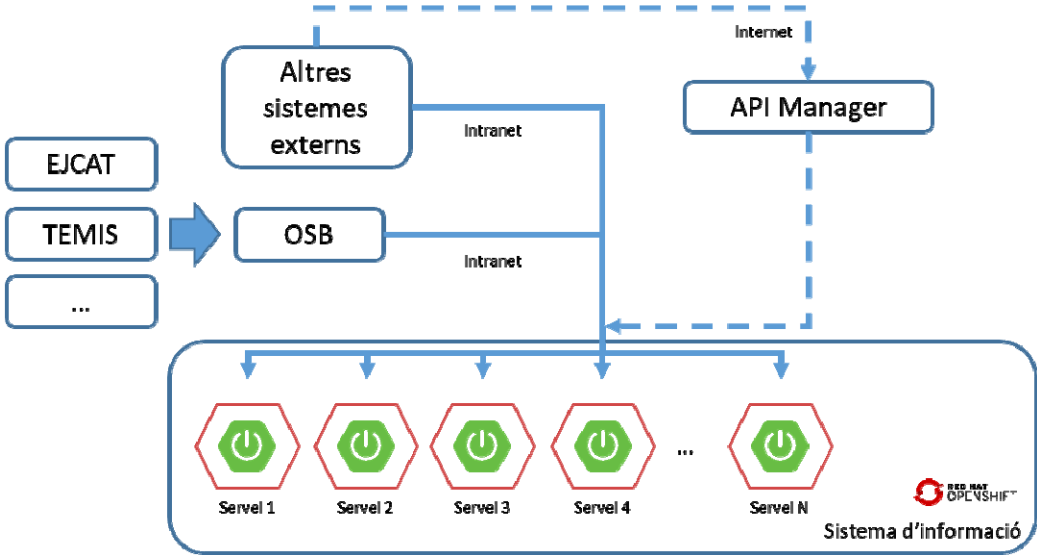
4.2.1 Invocació des de sistemes externs cap a aplicacions internes

Entendrem per sistemes externs:

1. Aplicacions EJCAT tradicionals desplegades a servidors WebLogic
2. Altres aplicacions o sistemes d'informació al Departament de Justícia
3. Aplicacions o sistemes externs al Departament de Justícia

La manera com qualsevol dels sistemes que accedeixi per internet realitzar invocacions a les aplicacions del sistema EjCat+ serà a través del API Manager de la nova arquitectura.

L'API Manager serà el punt d'entrada des d'Internet per a qualsevol petició externa que arribi al sistema.



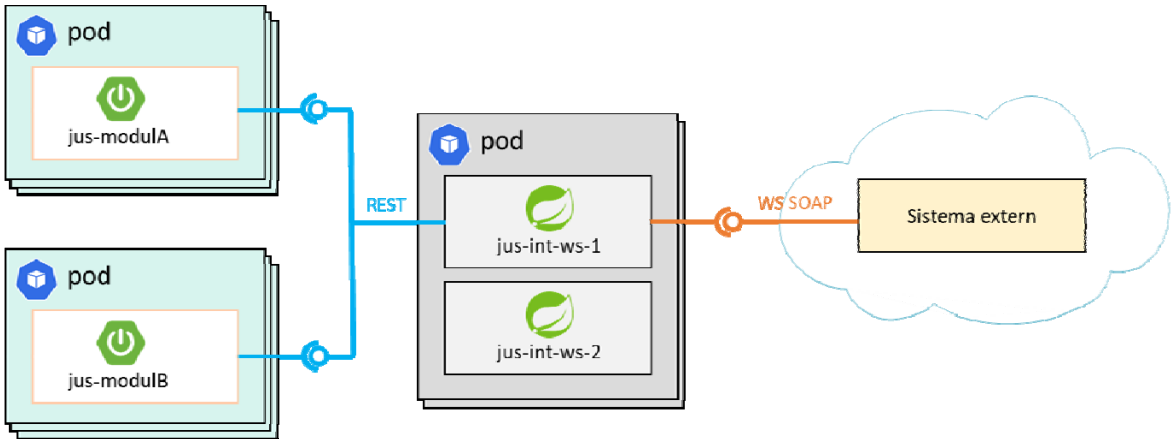
Gràcies a l'API Manager podrem tenir un únic punt d'entrada al sistema:

- Centralitzant temes de seguretat
- Enrutament cap a serveis finals
- Definició de quotes i assignació de recursos
- Gestió multi-tenant

4.2.2 Invocació des d'aplicacions EjCat+ cap a sistemes externs

Els mecanisme preferent quan s'hagi de consumir un servei extern al sistema serà consumir una API Rest. En sistemes més antics que no puguin oferir serveis REST i s'hagin d'accedir amb altre tipus de protocol es podrà fer ús de SI (Spring Integration). Mitjançant SI es podran crear peces (serveis d'integració) encarregades de rebre peticions REST dels serveis interns i transformar aquestes peticions cap als protocols necessaris segons la plataforma o aplicació que s'hagi de consumir.

Veure la guia [CU_ARQ012_Interacció_cap_a_sistemes_externs_(capa_negoci)] per més detall.



4.2.3 Interfícies amb sistemes interns

Considerarem sistemes interns totes aquelles aplicacions o serveis que segueixin arquitectura Canigó 3.4 Cloud desplegats en el marc del mateix sistema d'informació

Cada aplicació/servei és propietari de les seves dades (al igual que en l'arquitectura CMO) i si necessita dades d'una altra aplicació/servei, les ha d'obtenir invocant el servei de negoci que publiqui cada aplicació per a què li faci arribar les dades sol·licitades. En casos excepcionals pot necessitar disposar directament d'aquestes dades externes, encara que no és la solució recomanada. Dins aquest mateix capítol explicarem aquests dos casos i quan pot estar justificat accedir a dades externes sense invocar a un servei de negoci, tal i com ja passava a l'arquitectura CMO.

L'arquitectura permet les següents tipologies d'interfícies entre aplicacions internes:

- Comunicació interna capa presentació → capa distribució : (online / síncron) Serveis REST amb paràmetres JSON , per a comunicació front-end → back-end dins la mateixa aplicació

- Comunicació entre serveis interns del sistema d'informació

    - A la capa de presentació: (online / síncron) Canvi de context entre les capes de presentació de dues aplicacions o microfrontends, mitjançant protocol HTTPS

    - A la capa de negoci: (online o batch)

        - (síncron) La capa de negoci de l'aplicació A invocarà a la capa d'integració de l'aplicació B mitjançant invocació de serveis via REST amb l'existència intermèdia d'un Service Mesh per enrutar les peticions cap al destí correcte

        - (asíncron) La capa d'accés a dades de l'aplicació A enviarà un missatge a un servidor Kafka que serà recollit de forma asíncrona per les aplicacions B que estigui subscrites al tòpic en qüestió

    - A la capa de dades: (online / síncron) Cada aplicació o servei només podrà accedir a dues bases de dades diferents:

        - L'accés habitual serà a la seva BBDD local del servei on tindrà totes les dades necessàries per donar resposta al negoci, ja siguin dades pròpies o bé dades desnormalitzades d'altres serveis (normalment de taules mestres, ocasionalment de negoci).

        - En cas que per necessitats de negoci un servei concret hagi de fer creuament de dades amb col·leccions d'altres serveis tindrà la possibilitat d'anar a la BBDD de consultes on disposarà de les col·leccions per a fer aquests creuaments o podrà trobar col·leccions específiques on ja hi hagi aquestes dades creuades. Aquest escenari no és el recomanable donat que crea dependències de N serveis contra una mateixa BBDD, la qual cosa és un anti-patró d'arquitectures de serveis on cerquem la independència en el seu cicle de vida. Per tant el seu ús ha d'estar justificat.

4.2.3.1 Escenaris d'integració a capa de dades.

Tal i com es ja va indicar les dades són propietat i responsabilitat d'un servei i si algú necessita aquelles dades podrà consultar-les via invocació de serveis de negoci. Existiran escenaris on no serà recomanable obtenir aquestes dades via invocació de serveis com per exemple:

- Volum molt elevat de dades a retornar per part del servei
- Elevat número de peticions al servei de consulta

Per aquests escenaris on les invocacions via servei de negoci no sigui recomanable, es tractarà de garantir que les dades estan allà on son necessàries per donar resposta al negoci. Quan una dada d'un servei hagi d'estar disponible també per a altres, pot ser necessari fer-la arribar a la resta de serveis que la necessitaran. En aquest sentit hi ha diverses possibilitats com poden ser la rèplica de dades (desnormalització entre serveis) o l'accés a una base de dades global de consulta.

A aquests escenaris mencionats, si no són recomanables les invocacions via servei s'hauran de seguir les recomanacions de la guia [CU_ARQ028 ARQ Integració Capa Dades] d'integració a capa de dades on s'expliquen els diferents escenaris i eines que es poden utilitzar en funció del les característiques de la integració.

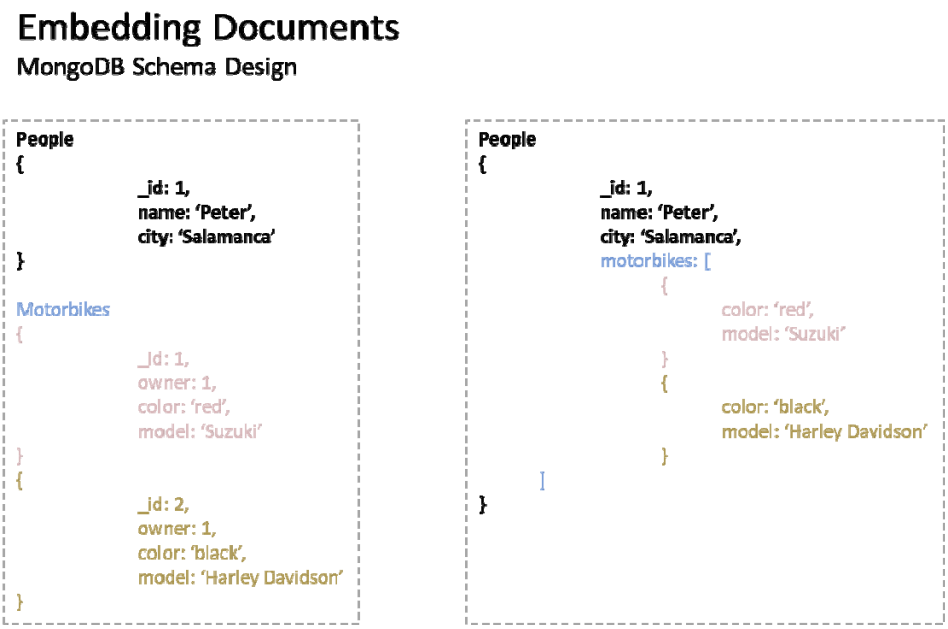
4.2.3.2 Elecció d'estratègies d'integració

- Desnormalització

  Al treballar amb MongoDB existeix un canvi de paradigma respecte a les BBDD relacionals. En les BBDD relacionals la informació se separa en diferents taules i aquestes taules es relacionen entre elles mitjançant FK. Quan arriba el moment de consultar dades a la BBDD es fan consultes i si cal, a la consulta, es relacionen les taules pertinents per obtenir les dades desitjades. Amb BBDD MongoDB la idea ja no és aquesta doncs les dades s'han d'emmagatzemar de manera que la seva consulta estigui optimitzada.

  Quan una aplicació existent passa a d'utilitzar BBDD relacionals a BBDD no relacionals la idea no és agafar les taules del sistema i canviar-les per col·leccions per replicar un model com l'existent. El que cal fer és un anàlisi més profund a nivell de negoci per saber quines entitats tenen sentit i com seran consultades pels diferents processos de negoci.

  Les dades son susceptibles de ser desnormalitzades dins les col·leccions que en fan ús. D'aquesta manera passem d'un escenari on tenim, per exemple, dues taules relacionades a un escenari on tenim una col·lecció que dins, com atributs de la col·lecció, té les dades desnormalitzades i per tant el que abans era una consulta on hi havia dues taules implicades ara passa a ser una consulta sobre una col·lecció que ja conté les dades que necessitem.



A MongoDB serà interessant aquest tipus d'estratègia donat que encara que es poden fer joins entre col·leccions podria afectar negativament al rendiment. Per estudiar els criteris recomanats dins un modelat orientat a documents, cal consultar la guia [CU_ARQ010.1 - Desnormalització i modelatge del model de dades].

Tot l'anterior aplica normalment a nivell intern de la base de dades MongoDB que utilitza el nostre servei. Però dins aquest capítol d'arquitectura d'integració, estenem el concepte anterior de desnormalització, i incloem l'escenari on una dada cal actualitzar-la dins una col.lecció externa al nostre servei, és a dir, a la base de dades MongoDB d'un o varis serveis externs. Aquest escenari es produirà normalment amb dades mestres, poc volàtils i accessibles molt freqüentment per tothom. En situacions excepcionals es podria considerar incloure altres dades de negoci, però sempre amb justificació i autorització dels comitès d'aprovació d'arquitectura (interns i/o de client), donat que sempre que sigui possible cal obtenir les dades via invocació de serveis de negoci.

Quan una dada canvia en origen (en la base de dades responsable d'aquelles dades), si hem escollit estratègia de desnormalització amb serveis externs, cal fer arribar el canvi d'aquella dada a totes les col·leccions de totes les bases de dades on estigui desnormalitzada.

- BBDD Consulta

  Aquest és un cas particular encara que especial de dades replicades.

  La BBDD de consulta serà una BBDD consultable pels serveis que ho requereixin on hi haurà dades de les diferents BBDD dels serveis. Aquestes dades poden estar normalitzades o replicades i les podrem trobar com a col·leccions independents amb estructures iguals o similars que a les bases de dades dels serveis, o bé les podem trobar amb altres estructures. Podrem trobar dades desnormalitzades de diferents col·leccions agrupades en una, o també grans col·leccions que permetin obtenir d'una tacada conjunts de dades de diferents serveis en una sola consulta.

  Aquesta BBDD permetrà realitzar consultes creuant diferents negocis.

4.3 ARQUITECTURA DE SEGURETAT

Els requeriments de seguretat determinats pels següents requisits no funcionals

Això és tradueix, a nivell d'arquitectura:

RNF	CU_ARQ associat
Proveir d'un mecanisme d'autenticació pels accessos al sistema des d'Intranet	CU_ARQ001_Autenticació Intranet CU_ARQ002_Autenticació Internet
Proveir d'un mecanisme de autorització pels accessos al sistema	CU_ARQ003_Autorització

4.3.1 Nivells de seguretat

Les aplicacions són responsables de la seguretat i l'accés sobre les seves funcionalitats i sobre les dades que manega o exposa. La seguretat és un concepte transversal a les funcionalitats i s'ha d'incloure al disseny de la aplicació. El principi mestre de la seguretat es el del "mínim privilegi" que consisteix en que per defecte no es concedeix accés a cap funcionalitat sense un permís explícit que garanteix que només qui ha d'accedir, accedeix.

Definim la seguretat en diferents nivells:

- Relacionat amb el context de la aplicació (*nivell 0* de seguretat):
    - modificació dels permisos per defecte per administrar els components associats a la aplicació que són responsabilitat d'ella: per exemple, si a un contenidor tenim un *Nginx* o un *Tomcat* al que es desplega la nostra aplicació, s'han de modificar els permisos per defecte per altres desconeguts pels desenvolupadors. Ho mateix al que es refereix als usuaris *root* dels sistemes operatius dels contenidors. A aquest nivell es troben les propietats definides com "*secrets*" d'*Openshift* que permetin treballar sense tenir que conèixer els usuaris i contrasenyes definides als entorns productius. CTTI indica que s'han d'utilitzar imatges publicades al seu repositori corporatiu harbor donat que es troben certificades respecte a la seva seguretat per CESICAT
    - Seguretat associada a elements externs a la aplicació: la aplicació no és responsable de si hi ha un *Firewall* que evita l'accés a les seves serveis exposats, o d'un filtre de continguts extern, ... però una vegada té coneixement de la seva existència, si que s'ha d'adaptar per tal de donar servei: pot-ser ha de fer un canvi de port, demanar una excepció al *Firewall* o al filtre de continguts, ...
- Relacionats amb l'accés a les funcionalitats (*nivell 1* de seguretat):
    - Dins de les seves possibilitats, una aplicació ha de tractar de garantir que no es produeix "*impersonation*" es a dir, que un usuari es pugui fer passar per un altre.
        - La utilització d'un *token JWT* signat i comprovat sobre el *Proveïdor d'Identitat* permet obtenir dades "fiables". A les aplicacions *Spring* configurades per fer aquesta validació del *token*, les dades incloses al context de seguretat es poden considerar "fiables". És responsabilitat de la aplicació utilitzar aquestes dades per validar que un usuari no tracta de realitzar accions en nom d'un altre.
        - Una aplicació ha de superar una auditoria de l'organisme encarregat de la seguretat al client abans de poder utilitzar-se en entorns productius. A la Generalitat es ho realitza habitualment *CESICAT* que participa a les "*fase 0*" dels projectes i realitza una auditoria sobre la aplicació a un entorn no productiu (*preproducció* generalment). Aquesta auditoria

fa una comprovació dels mecanismes d'explotació de vulnerabilitats més greus o habituals a les aplicacions de tipus *Web*.

- Si una aplicació exposa funcionalitats, al disseny s'ha d'indicar que han d'acomplir els usuaris per poder utilitzar-les:
  - Si és possible un accés públic: per exemple accés a funcionalitats exposades a una *web* pública a internet, o el sistema de *login*.
  - Si és un accés púbic per tots els usuaris del sistema: es necessari que l'usuari hagi fet *login* al sistema i no li cal més privilegis.
  - Si és només un conjunt d'usuaris ho han de poder utilitzar: definir al disseny que han d'acomplir aquests usuaris: pertànyer a un grup o tenir una característica, etc... aquesta decisió es pot realitzar sobre les dades de seguretat de l'usuari (*Context de seguretat*), fent validacions sobre les dades rebudes amb altres dades pròpies de la aplicació, etc...
  - Les validacions de seguretat d'aquest nivell es fan als components que s'encarreguen de la comunicació amb capes superiors, normalment els *REST Controllers*.

- Relacionats amb les dades (*nivell 2* de seguretat):
  - Un usuari pot pertànyer a un grup autoritzat a una funcionalitat però no tenir accés a la visualització o execució de totes les accions associades a aquesta funcionalitat. Es responsabilitat de la aplicació verificar que cada usuari concret només pot veure i executar aquelles accions sobre les que te privilegis.
  - En moltes ocasions aquest nivell de seguretat s'ha de "*programar*" com part de les *consultes*: selecció de les dades del propi usuari, o selecció de les dades sobre aquelles dades funcionals a les que l'usuari ha de tenir accés: "*d'una unitat sobre la que l'usuari té permisos*". Normalment com forma part de com s'accedeix a les dades, aquest nivell de seguretat està implementat a les *consultes* de les dades, als *DAO*.
  - Les dades exposades cap al context també s'han de verificar (*logs* exposats, per exemple), *mètriques*, etc, per tal de no mostrar informació "sensible" sense la protecció adient.

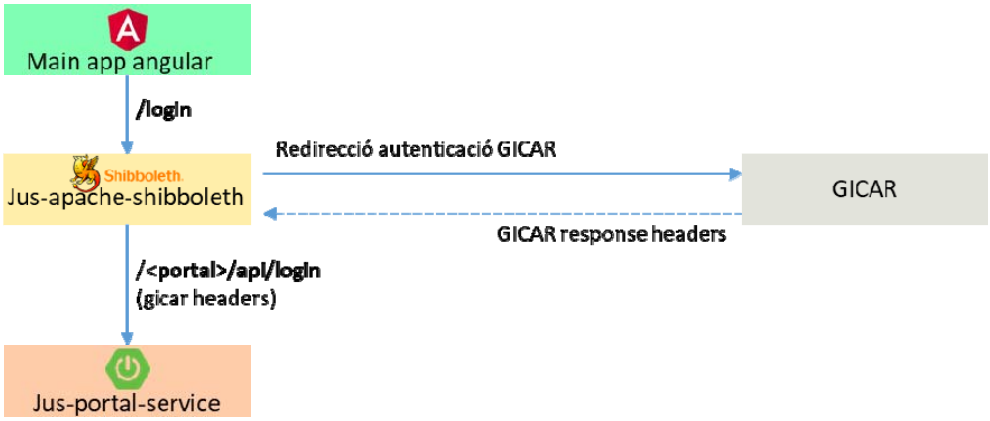
Al definir les comunicacions entre components del sistema s'ha de tenir en consideració que:

- Comunicacions entre serveis del mateix sistema d'informació que es trobin desplegats al mateix namespace no s'hauria de passar per l'API Manager.
- Comunicacions entre serveis intranet no haurien de passar per Api Manager
- Comunicacions entre serveis del propi Departament de Justícia no haurien de passar per Api Manager
- Comunicacions amb origen intranet en altres departaments caldria valorar si han de passar per API Manager.
- Comunicacions amb origen internet haurien de passar per Api Manager

4.3.2 Descripció tècnica de la solució de seguretat

- Integració amb GICAR

El diagrama lògic d'integració amb GICAR és el següent:



L'agent de Shibboleth amb la configuració pertinent per autenticar als usuaris contra als directoris corresponents de GICAR per Justícia, en aquest cas, de la Intranet, amb les credencials que tingui disponibles a tal efecte. (usuari i contrasenya, targeta criptogràfica, certificats...)

Els mòduls Portals de la Intranet o la Extranet estaran configurats com destinataris de les operacions d'autenticació, i seran els responsables de generar les autoritzacions pertinents en forma de tokens JWT.

No es contemplarà, de moment, possibilitat de contingència en cas que GICAR no es trobi actiu.

- Autorització amb tokens

Es basa a l'autorització en tokens JWT en format OpenID Connect (OIDC).

Un cop rebuda l'autenticació correcte de l'usuari, el portal del sistema adient haurà de recollir del corresponent component de gestió d'usuaris la resta d'informació necessària de l'usuari connectat.

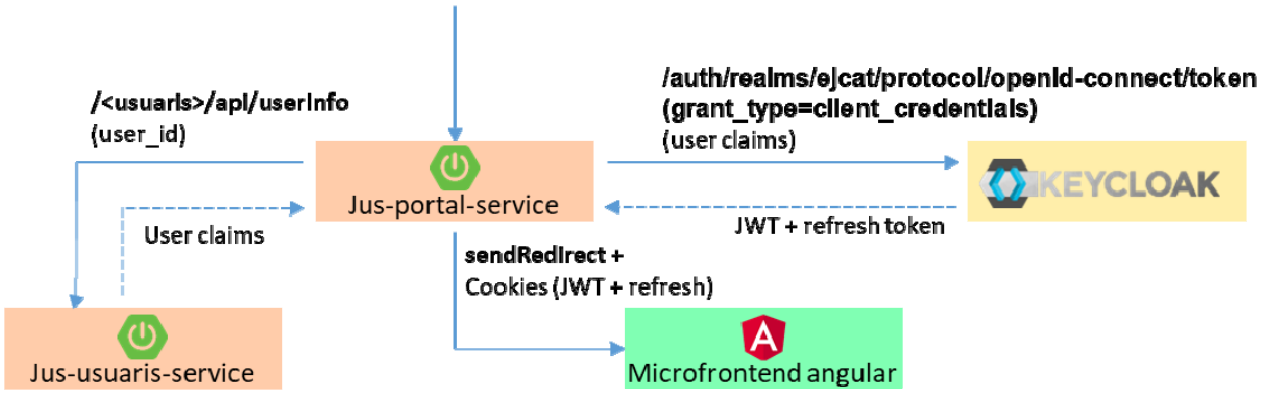
Per generar els tokens JWT utilitzarem la funcionalitat de *Identity Provider* Oauth2 del producte Keycloak.

S'hauràn de definir diferents tipus d'autoritzacions, segons cada tipus de destinatari:

- Usuari d'intranet o d'extranet (per exemple, grant_type: client_credentials)
- Sistema extern (grant_type: client_credentials)

A partir d'aquí, la informació rellevant de l'usuari s'enviarà com paràmetres a les crides OAuth2 (/access_token, /refresh_token), i quedarà incorporada dins el payload del token, en forma de *claims*.

Finalment, la resposta serà aquest token JWT en format OIDC, que els mòduls transferiran a les aplicacions destí per poder autoritzar a l'usuari connectat.



Un cop generat el token, qualsevol petició a les aplicacions destí hauran d'incloure la capçalera:

```
Authorization:  Bearer + <token JWT>
```

El format del token OIDC serà el següent:

```
{
  "aud": ...,
  "sub": "XXXXX",
  "application": ...,
  "scope": ...,
  "iss": "https://...keycloak.justicia.intranet.gencat.cat/.../openid-connect/token",
  "tierInfo": ...,
  "keytype": ...,
  "subscribedAPIs": ...,
  "consumerKey": ...,
  "exp": 1593519921,
  "iat": 1593516321,
  "jti": xxx,
  "userInfo": {
     "param1":"value1",
     ...
   }
}
```

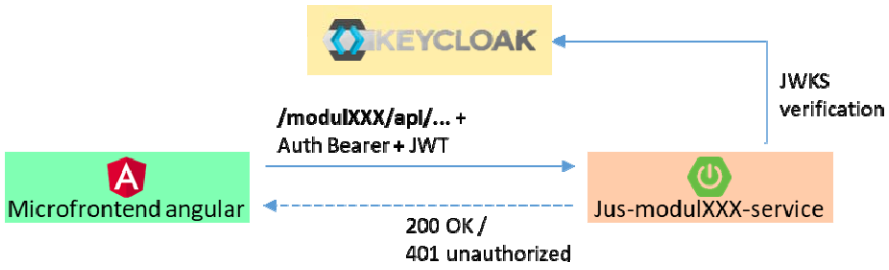
Tots els serveis incorporen una capa d'Spring Security, configurada per validar en cada petició REST el token JWT rebut. Al estar en format OIDC, no serà necessari que en cada crida el mòdul tingui que invocar al proveïdor d'identitats per desxifrar i validar el token.

Dins la configuració aportada en el mòdul ja s'inclourà un endpoint OIDC per recollir la clau pública del certificat amb que el proveïdor d'identitats ha signat cada token, d'aquesta manera els mòduls els podran validar de forma autònoma, eliminant requests innecessàries al sistema. Aquest endpoint s'exposa en format JWKS (JSON Web Key Set).

Si l'autorització es correcte, Spring Security permetrà executar la crida REST al servei generant un *Security Context* dins Spring amb la informació de l'usuari connectat. Aquesta informació s'extreu dels *claims* inclosos en el token, tal com s'ha comentat prèviament.

Si el token és invàlid, o està caducat, es retornarà la corresponent excepció de seguretat, que s'acabarà transformant en un error *401 : Unauthorized*.

Des del frontend es podrà invocar un endpoint de refresc de token, en cas que s'apropi la seva data de caducitat.



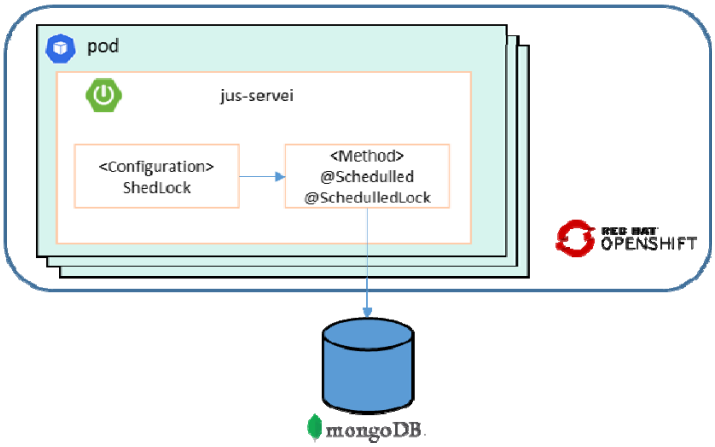
Com a T-Components per a la gestió de l'autorització a s'inclouen:

- JusticiaAuthenticationEntryPoint: Gestió de la resposta davant dels accessos no autoritzats a l'aplicació.
- JusticiaCryptoHelper: Encriptació de dades amb un parell de claus pública/privada
- JusticiaCustomEncoder: Codificació de payload entre frontend i backend per dades a protegir
- JusticiaGrantedAuthoritiesExtractor: Conversió de token JWT en un Security Context d'Spring

4.4 ARQUITECTURA PROCESSOS PLANIFICATS

El producte estàndard per la gestió dels processo planificats serà ShedLock als projectes del departament de Justícia.

Aquest producte permet la planificació de tasques als pods amb control per evitar execucions simultànies de la mateixa tasca per les diferents instancies en execució.



4.5 PROPIETATS TRANSVERSALS DEL SISTEMA

Tota arquitectura té propietats transversals que ha de resoldre. Per tal de no augmentar en excés la quantitat d'informació inclosa en aquest document, aquestes propietats es descriuran dins les diferents guies de frontend. A la guia [CU_ARQ_32-33 - Capa de presentació (general)] es diferenciaran les propietats transversals de la capa de presentació de la resta de capes.

Les propietats transversals per a la capa de presentació (Angular) que es descriuran la guia [CU_ARQ_32-33 - Capa de presentació (general)] :

- Autentificació i seguretat
- Configuració multi-entorn
- Internacionalització
- Gestió d'errors
- Cache
- Gestió de l'estat (components 'Stateful')
- Context d'usuari
- Gestió d'events
- Generació de Logs
- Navegació i enrutament
- Microfrontends
- PWA
- Monitorització

Les propietats transversals que es descriuran de la resta de capes són:

- Gestió de Transaccions
- Gestió d'Excepcions
- Gestió de la sessió d'usuari
- Validacions a capa client i servidor
- Logging
- Caching
- Configurabilitat
- Internacionalització

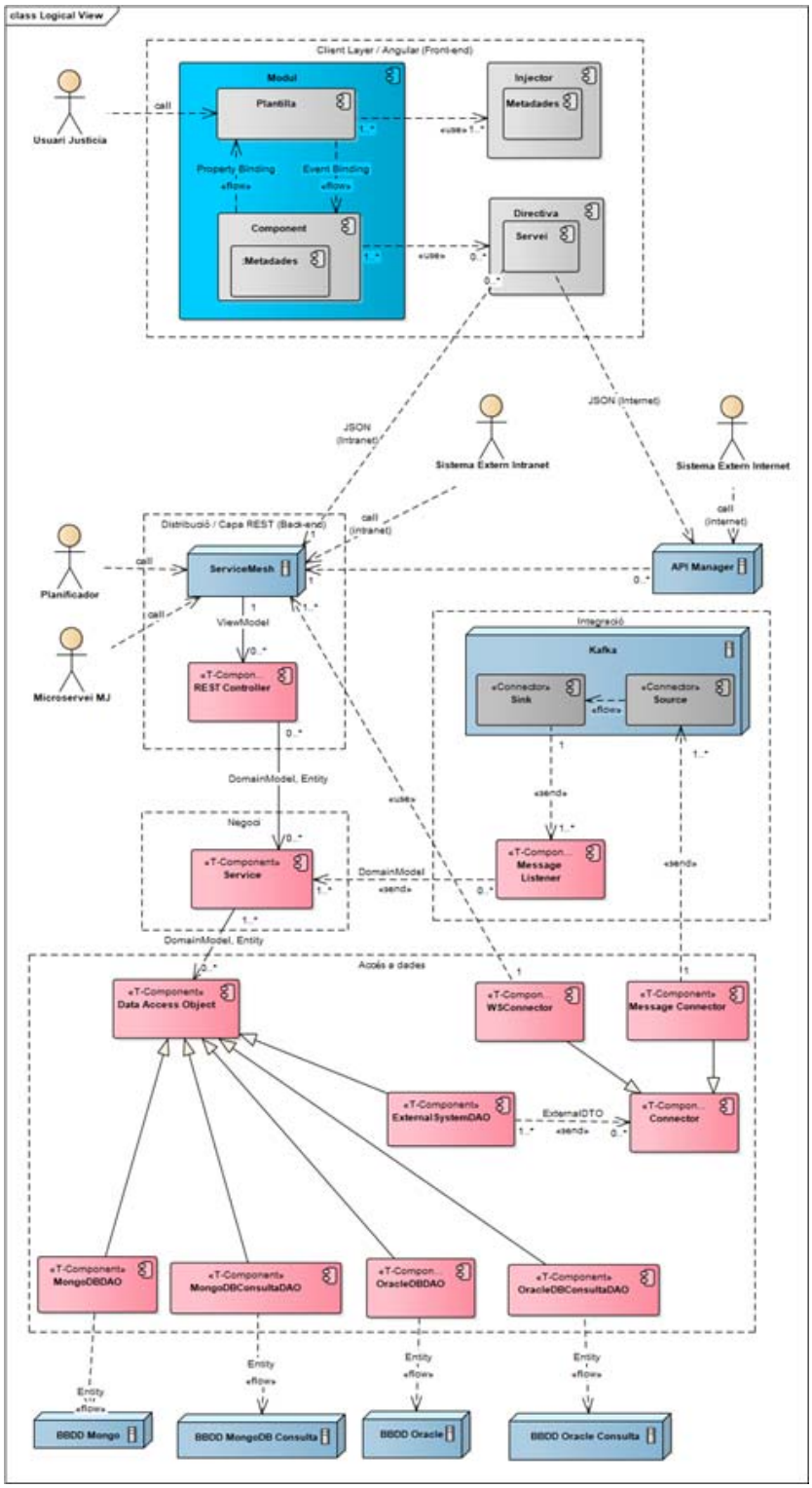
5 VISTES DE L'ARQUITECTURA DE REFERÈNCIA

5.1 GENERAL

La vista general mostra les parts del sistema més importants des del punt de vista funcional i tècnic, a alt nivell. Mostra com les parts del sistema son distribuïdes a través dels elements de la infraestructura tècnica (TI).

El següent diagrama mostra les capes i tecnologies a alt nivell més rellevants de l'arquitectura JEE del Departament:

5.1.1 Vista lògica



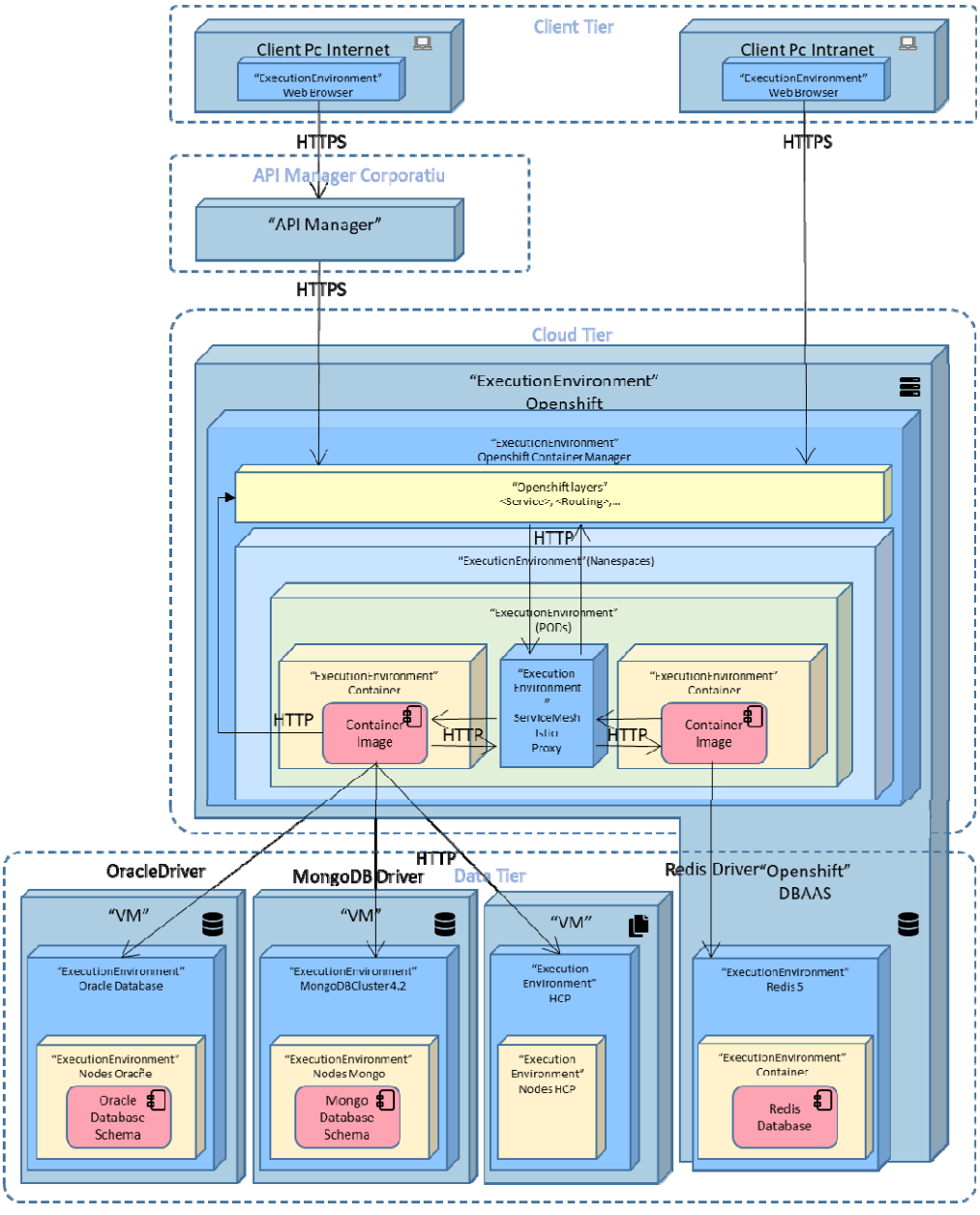
Aquest diagrama mostra les principals tipologies de components que conté cada aplicació. Alguns elements de tipus T-Component, per facilitar la comprensió, no s'han inclòs en aquest diagrama, encara que es farà referència en altres apartats d'aquest document.

Tampoc s'ha inclòs dins el diagrama, pel mateix motiu, la referència a tota la capa del framework Canigó que dona suport a la resta de capes d'aplicació. Aquesta dependència es pot observar a l'apartat Vista d'implementació.

5.1.2 Vista de desplegament

5.1.2.1 Entorn de Producció (JUPRO)

La vista de desplegament descriu les configuracions dels components hardware on s'executa el sistema. Documenta els nodes, que representen tant entorns d'execució (servidors amb certa capacitat de processador i memòria) com software instal·lat en ells (servidors d'aplicacions, servidors web, base de dades, etc.)
El diagrama indica també els artefactes que formen cada aplicació, i com es fa el desplegament de cada part al node corresponent.
Disposarem d'entorn d'Integració(INT), de Preproducció (PRE), de Producció (PRO) i de Formació (FOR)
Els entorns de PRE i PRO tindran la mateixa configuració.
L'entorn de INT tindrà els mateixos components però no caldrà garantir alta disponibilitat ni ser un entorn igual a PRO i per tant és susceptible de ser més reduït que PRE i PRO
L'entorn de FOR compartirà maquinària amb PRE per estalviar costos i per tant tindrà la mateixa configuració.
A continuació detallem la vista de desplegament de l'Productiu (PRO) on es mostrarà dos tipus d'informació: vista de desplegament lògica i infraestructura hardware.



Podem observar les següents característiques al diagrama anterior:

- Distingim diferents tipus de client:
    - un client de tipus web, on començarà la navegació a través del browser. Segons el tipus de client accedirà a través d'un API Manager cap la plataforma Openshift o accedirà directament.
    - un client de tipus aplicació (exclusivament un altre servei del sistema) que podrà accedir a la lògica de negoci mitjançant una crida a les funcionalitats del servei destí.
    - altres sistemes externs que vulguin accedir a un mòdul de Justícia.
- El contingut estàtic no es desplegarà a aquests servidors, s'haurà de desplegar al contenidor corresponent a l'Openshift.
- L'API Manager redirigeix les peticions dels Clients cap a l'Openshift
- Els containers serveixen els recursos de presentació (Angular) i a ells es desplega la part estàtica de l'aplicació (imatges, estils, javascript d'Angular, etc.).
- Als contenidors es despleguen els artifacts amb les aplicacions.
- Al servidor de HCP es poden instal·lar documents, estructura de carpetes, etc, mitjançant l'API exposada pel servei GDO+.

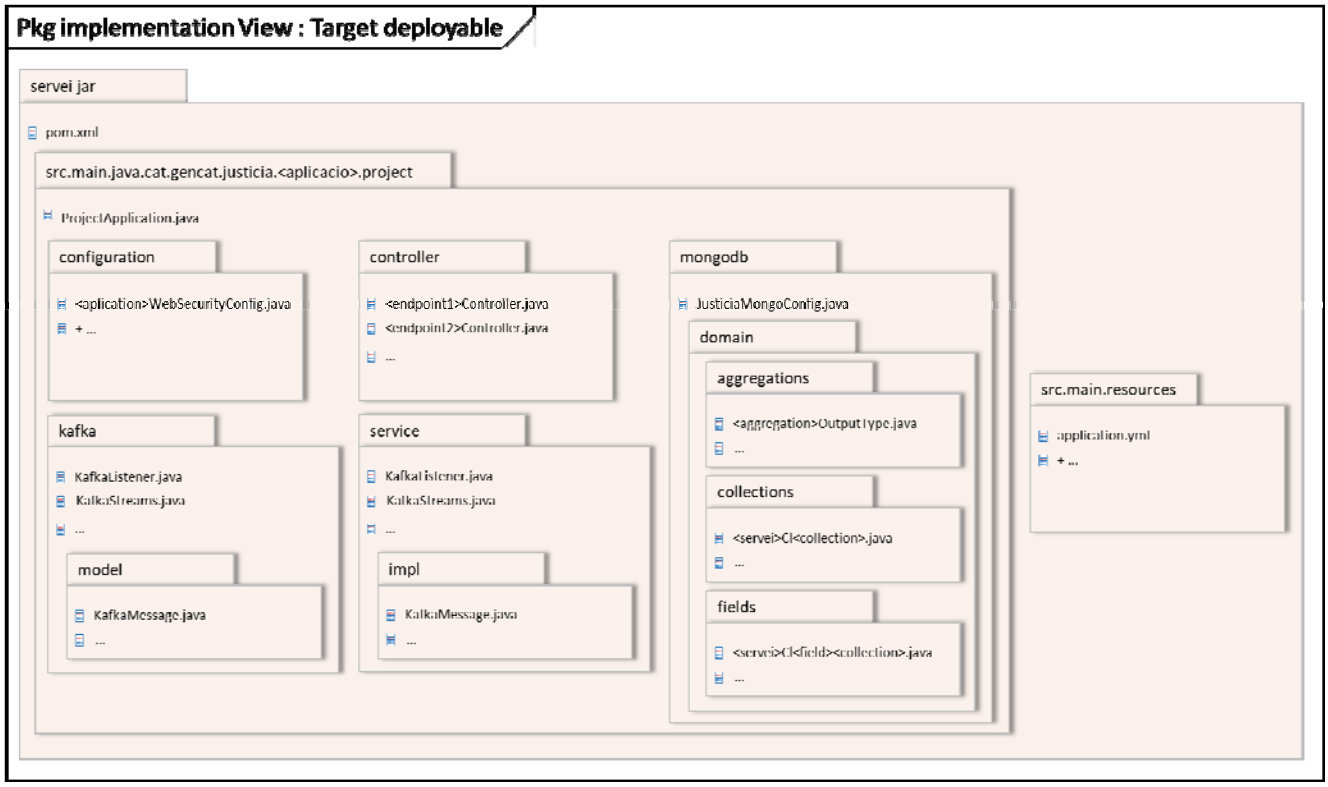
5.1.3 Vista d'implementació

La vista d'implementació descriu la organització dels elements d'aplicació des del punt de vista del desenvolupador.

5.1.3.1 Organització del projecte, capa presentació

La vista d'implementació de la capa de presentació es descriu a l'apartat Vista d'implementació de la capa de presentació.

5.1.3.2 Organització del projecte, serveis



A continuació es descriuen els subsistemes principals del servei i la seva funció:

- **Subsistema configuration:** Conté informació de configuració de Swagger i Seguretat
- **Subsistema controller:** correspon a la capa de distribució de la Vista lògica de l'apartat 5.1.1. En els projectes Canigo 3.4 Cloud en aquesta capa web només exposa l'API de serveis RESTful.. Contingut:
  - REST Controllers
- **Subsistema model:** Conté les interfícies i classes que han de ser accedides des de la resta de capes lògiques. Dins d'aquest subsistema no pot haver-hi lògica de negoci. Contingut:
  - Domain Model objects, que no requereixen de persistència
  - Model de persistència generat amb les entitats JPA, i les seves extensions modelats com entitats a mida (EntityCustom)
- **Subsistema mongodb: Conté tota la lògica de negoci relacionada amb l'accés a BBDD**
  - Classes per mapejar col·leccions i objectes
  - Classes amb lògica: agregations i collections

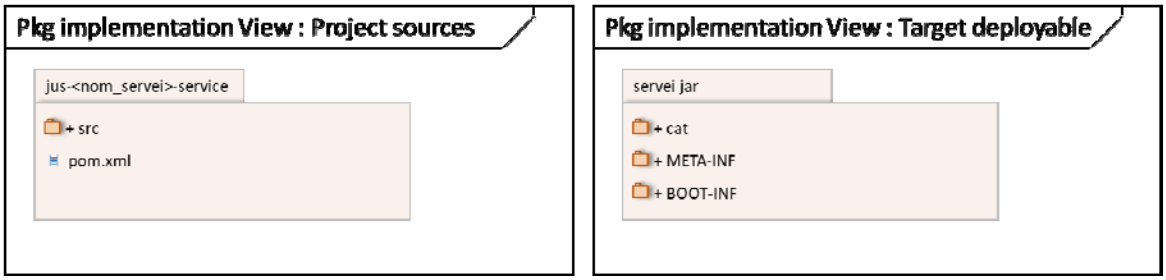
- **Subsistema service:** correspon a la capa de negoci de la Vista lògica de l'apartat 5.1.1. Contingut:
  - Interfícies dels serveis
  - Implementacions dels serveis

Altres subsistemes:

- **Subsistema audit:** Conté la lògica de negoci relacionada amb l'auditoria
  - Classes per personalitzar la informació d'auditoria
- **Subsistema Kafka:** Conté tota la lògica de negoci relacionada amb les integracions amb Kafka
  - Classes per mapejar missatges kafka
  - Classes amb lògica: listeners i streams
- **Subsistema reports:** correspon a la capa de negoci de la Vista lògica de l'apartat 5.1.1. Contingut:
  - Interfícies dels serveis
  - Implementacions dels serveis

```
jus-[NOM_SERVEI]-service/src/main/java:
       cat.gencat.justicia.[NOM_SERVEI]
              .project                                    (Aplicacio)
                     .common.audit                        (auditoria)
                     .common.crosscutting.exceptions      (Excepcions)
                     .configuration                       (Configuracions)
                     .controller                          (REST Controllers)
                            .[AMBIT FUNCIONAL 1]
                            .[AMBIT FUNCIONAL 2]
                     .kafka                               (Kafka Listeners)
                            .model                        (Kafka Model)
                     .model                               (Domain Model objects)
                            .[AMBIT FUNCIONAL 1]
                            .[AMBIT FUNCIONAL 2]
                            .view                         (View Model objects)
                                   .[AMBIT FUNCIONAL 1]
                                   .[AMBIT FUNCIONAL 2]
                            .extdto                       (DTOs amb serveis externs)
                            .adapter                      (View Model ↔ Domain Model)
                                                          (ExternalDTO ↔ Domain Model)
                            .connector                    (Connectors per protocol)
                     .mongodb
                            .domain
                                   .aggregations
                                   .collections
                                   .fields                (Entities)
                                          .custom         (Extensions Entities)
                            .template
                                   .dao                   (Interfícies DAO)
                                          .impl           (Implementacions DAO)
                                          .external       (External DAO)
                     .reports                             (Reports)
                     .service                             (Interfícies service)
                            .[AMBIT FUNCIONAL 1]
                            .[AMBIT FUNCIONAL 2]
                            .impl                         (Implementacions service)
                                   .[AMBIT FUNCIONAL 1]
                                   .[AMBIT FUNCIONAL 2]
                     .shedlock                            (Tasques planificades)
                            .[AMBIT FUNCIONAL 1]
                            .[AMBIT FUNCIONAL 2]
                     .security                            (Extensions JWT de cada mòdul)
                     .util
```

Per tant, la part dinàmica d'un projecte, i la del desplegable final, tindrà aquesta estructura de components



Els fonts s'organitzaran als projectes segons s'indica a l'apartat Organització dels packages – serveis.

Els Controller de la capa de distribució només fan referència en compilació al model de domini, i les interfícies dels Serveis associats amb els que comunica amb la capa de negoci.

És Spring en temps de runtime qui fa la injecció (@Autowired) dels components de negoci als controladors de la capa REST.

Un Controlador REST no cridarà mai directament a un servei d'un altre mòdul. Sempre delegarà en un Servei aquesta tasca, que alhora utilitzarà un ExternalSystemDAO.

Els components encarregats de la integració també accedeixen a la capa de negoci mitjançant injeccions Spring. De nou, en compilació accedeix a les interfícies, i per injecció d'Spring (@Autowired) en temps de runtime s'accedeixen als serveis associats.

5.1.3.3 Organització dels packages – capa de presentació

La organització dels packages a la capa de presentació es descriu a l'apartat Vista d'implementació de la capa de presentació.

5.1.3.4 Organització dels packages – serveis

Els noms dels paquets Java han de començar amb `cat.gencat.justicia.[NOM_SERVEI]`.

Els paquets han d'estar estructurats a partir d'aquest nivell segons aquests nivells jeràrquics:

- En primer lloc, segons capes tècniques (horitzontal)
- En segon lloc, segons àmbit funcional (vertical): només si hi ha molts elements, i s'aplica a l'últim nivell de la jerarquia (controller, view, etc):

```
jus-[NOM_SERVEI]-service/src/main/java:
       cat.gencat.justicia.[NOM_SERVEI]
              .project                                          (Aplicacio)
                     .audit                                     (auditoria)
                     .common.crosscutting.exceptions            (Excepcions)
                     .configuration                             (Configuracions)
                     .controller                                (REST Controllers)
                            .[AMBIT FUNCIONAL 1]
                            .[AMBIT FUNCIONAL 2]
                     .kafka                                     (Kafka Listeners)
                            .model                              (Kafka Model)
                     .model                                     (Domain Model objects)
                            .[AMBIT FUNCIONAL 1]
                            .[AMBIT FUNCIONAL 2]
                            .view                               (View Model objects)
                                   .[AMBIT FUNCIONAL 1]
                                   .[AMBIT FUNCIONAL 2]
                            .extdto                             (DTOs amb serveis externs)
                            .adapter                            (View Model ⇔ Domain Model)
                                                                (ExternalDTO ⇔ Domain Model)
                            .connector                          (Connectors per protocol)
                     .mongodb
                            .domain
                                   .aggregations
                                   .collections
                                   .fields                      (Entities)
                                          .custom               (Extensions Entities)
                            .template
                                   .dao                         (Interfícies DAO)
                                          .impl                 (Implementacions DAO)
                                          .external             (External DAO)
                     .reports                                   (Reports)
                     .service                                   (Interfícies service)
                            .[AMBIT FUNCIONAL 1]
                            .[AMBIT FUNCIONAL 2]
                            .impl                               (Implementacions service)
                                   .[AMBIT FUNCIONAL 1]
                                   .[AMBIT FUNCIONAL 2]
                     .shedlock                                  (Tasques planificades)
                            .[AMBIT FUNCIONAL 1]
                            .[AMBIT FUNCIONAL 2]
                     .security                                  (Extensions JWT de cada mòdul)
                     .util
```

Els Controller de la capa de distribució només fan referència en compilació al model de domini, i les interfícies dels Serveis associats amb els que comunica amb la capa de negoci.

És Spring en temps de runtime qui fa la injecció (@Autowired) dels components de negoci als controladors de la capa REST.

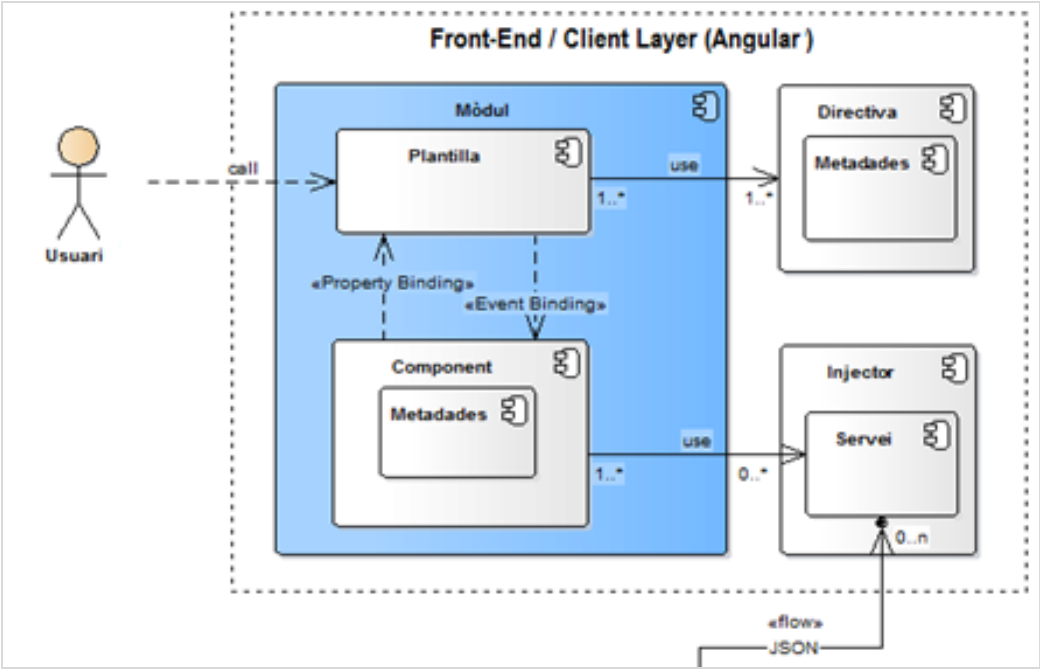
Un Controlador REST no cridarà mai directament a un servei d'un altre mòdul. Sempre delegarà en un Servei aquesta tasca, que alhora utilitzarà un ExternalSystemDAO.

Els components encarregats de la integració també accedeixen a la capa de negoci mitjançant injeccions Spring. De nou, en compilació s'accedeix a les interfícies, i per injecció d'Spring (@Autowired) en temps de runtime s'accedeixen als serveis associats.

5.2 CAPA DE PRESENTACIÓ – ANGULAR

5.2.1 Nomenclatura i responsabilitats

L'arquitectura de la capa de presentació segueix la arquitectura estàndard d'una aplicació basada en Angular composta pels següents blocs principals:



Les responsabilitats dels components d'aquesta capa són:

Component	Responsabilitat	Nomenclatura
Mòduls	Consoliden components, serveis i directives en blocs cohesius de funcionalitat, cadascun centrat en una àrea de característiques, domini de negoci de l'aplicació, flux de treball o una col·lecció comuna de serveis.	[Nom].module.ts
Components	Un component controla una part de la pàgina denominada com a vista (view). La lògica del component es defineix dins una classe —les funcionalitats per a controlar la vista—. El component interactua amb la vista mitjançant les seves propietats i mètodes.	[Nom].component.ts
Metadades	Les metadades informen a Angular com processar una classe (mòdul, component, directiva, etc.).	@NgModule, @Component, @Directive
Plantilles	Una plantilla defineix la vista (view) d'un component. Una plantilla és un HTML que indica a Angular com representar el component.	[Nom].component.html, [Nom].component.css
Directives	Donen les instruccions a Angular de com transformar el DOM durant el procés de creació de les vistes a partir de les plantilles.	[Nom].directive.ts
Bindings	Mecanisme per coordinar parts d'una plantilla amb parts d'un component. Afegeix unes marques (*binding markup*) a l'HTML de plantilla per dir a Angular com connectar ambdós costats.	N/A
Serveis	Implementen característiques que són independents de qualsevol vista específica, proporcionen lògica o dades compartides a través de components, o encapsulen interaccions externes.	[Nom].service.ts

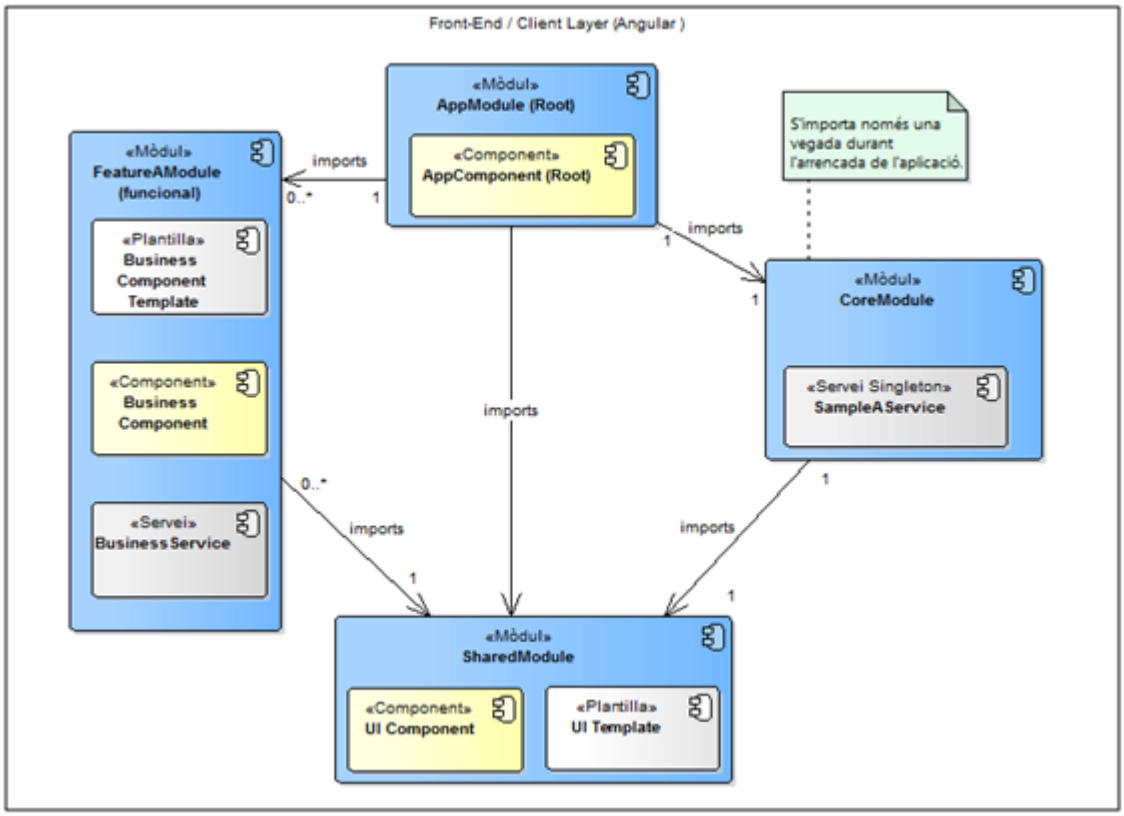
Per a cada component es defineixen els següents patrons arquitectònics:

5.2.1.1 Mòduls (Modules)

L'aplicació Angular tindrà com a mínim una classe de tipus mòdul (Module): El mòdul arrel (Root Module) i que, seguint la convenció estàndard, s'anomenarà AppModule. A més del root module l'aplicació estarà composada per mòduls funcionals, cadascun com a un bloc cohesiu dedicat a un domini d'aplicació, flux o conjunt de capacitats similars. Un mòdul d'Angular, sigui el root o qualsevol altre, és una classe (TypeScript) amb @NgModule com a decorator o metadada.

L'arquitectura modular de l'aplicació està composta pels següents 4 tipus de mòduls:

- Arrel (Root).
- Funcional (Feature).
- Core.
- Compartit (Shared).



ⓘ Els mòduls d'Angular no hereten l'accés als components o directives declarades a altres mòduls. Per exemple, tot el que sigui importat al Root Module AppModule és irrellevant per a un mòdul funcional i viceversa.

5.2.1.1.1 Metadades

Les metadades d'un mòdul d'Angular:

- Declaren quins components i directives pertanyen al mòdul.
- Fan algunes de les classes publiques per tal de que les plantilles d'altres components les puguin fer servir.
- Importen altres mòduls i els seus components, directives i pipes necessaris pels com-ponents en aquest mòdul.
- Proporcionar serveis al nivell d'aplicació que qualsevol component d'aplicació pugui utilitzar.

Les metadades més importants que descriuen el mòdul són:

- declarations – Las classes de tipus 'view' que pertanyen al mòdul. Angular té tres tipus de 'view' classes: components i directives. S'ha de declarar cada component en una (i només una) classe NgModule. Cada component creat al mòdul s'ha de incloure a l'array declarations.

- exports – El subconjunt de *declarations* que seran visibles i utilitzables a les plantilles (templates) d'altres mòduls.

- imports – Classes exportades per altres mòduls I que són necessàries per plantilles de components declarats en quest mòdul. Només classes de tipus NgModule s'inclouen en el array imports. No es pot incloure qualsevol altra tipus de classes en les importacions (imports).

- providers – Proveïdors de serveis amb els que aquest mòdul contribueix a la col·lecció global de serveis. Aquest seran accessibles en qualsevol altra part de l'aplicació Angular.

- bootstrap – La vista (view) principal de l'aplicació, anomenada root component. El component root que Angular crea i inserta a la pàgina web principal index.html i a totes les altres views de l'aplicació. Només el mòdul Root ha de configurar la propietat bootstrap.

5.2.1.1.2 Mòdul *Root*

Tota aplicació Angular té, com a mínim, una classe de tipus mòdul, el mòdul Root. Com a convenció, el mòdul Root és una classe anomenada **AppModule** i ubicada en un fitxer TypeScript anomenat **app.module.ts**. Altres característiques importants són:

- S'executa el procés d'arrancada (bootstrap) d'aquest mòdul per tal d'iniciar l'aplicació en un fitxer TypeScript anomenat **main.ts** file.
- Entre d'altres coses, el procés d'arrencada (bootstrap) crea el component (o components) inclosos en el array bootstrap (metadata) i els inserta al DOM del navegador.
- Cada component iniciat (**bootstrapped**) és la base del seu propi arbre de components.
- S'insereix el **component arrel** al **iniciar l'aplicació**. Aquest procés d'arrencada (**bootstrap**) desencadena una cascada de creacions de components que completen l'arbre de components de l'aplicació. El component arrel (root) s'anomenarà seguint la convenció com a **AppComponent**.

S'importen la resta de mòduls funcionals, que representen col·leccions de funcionalitats relacionades, dins el mòdul Root.

5.2.1.1.3 Mòduls funcionals (Feature Modules)

Un mòdul funcional és una classe TypeScript amb l'anotació **@NgModule** (decorator) i les metadades corresponents, de la mateixa forma que es defineix el mòdul Root. Les metadades d'un mòdul funcional tenen les mateixes propietats que les del mòdul Root.

Existeixen dues diferències tècniques significants:

1. L'aplicació Angular s'inicia arrencant el mòdul Root; Importem un mòdul funcional per tal d'ampliar la funcionalitat de l'aplicació.

2. Un mòdul funcional pot exposar o ocultar la seva implementació als altres mòduls.

Altres consideracions importants respecte als mòduls funcionals:

- Un mòdul funcional proporciona un conjunt de funcionalitats enfocades en un domini de negoci de l'aplicació, un flux de negoci, un servei (comunicació HTTP, enrutament) o un conjunt d'utilitats relacionades.

- El mòduls funcionals permeten particionar l'aplicació en àrees d'interès i propòsit específic.

- Un mòdul funcional col·labora amb el mòdul Root i la resta de mòduls funcionals mitjançant els serveis que proveeix i el conjunt de components i directives que es defineixen com a exportacions.

- Un mòdul funcional i tots els seus components, plantilles (views), etc., estaran ubicats en un directori específic separat per a diferenciar els elements que hi pertanyen respecte als del mòdul Root i la resta de mòduls funcionals.

- Pel que fa als components, cada mòdul ha d'importar les seves pròpies dependències sense tenir en compte si les mateixes dependències es van importar al mòdul Root o en qualsevol altre mòdul funcional. Per exemple, encara que tinguem múltiples mòduls funcionals, cadascun d'ells haurà d'importar el mòdul d'Angular CommonModule.

Si tenim una aplicació multi-modular s'implementarà **Lazy Loading**. El gran avantatge del Lazy Loading és que podem carregar els nostres recursos quan es necessitin i no tots alhora al iniciar l'aplicació. Això ajuda a disminuir el temps d'inici.

A continuació es mostren convencions i bones pràctiques en l' organització dels Lazy Loaded mòduls d'una aplicació Angular:

- Crear un mòdul NgModule per a cada àrea funcional (feature) ubicant el fitxer del mòdul en la mateixa carpeta amb nom que l'àrea funcional. Això facilita l'ús del LazyLoading i la seva reutilització.

- Col·locar el contingut de funcionalitats lazy loaded en una carpeta que contindrà un component d'enrutament, els seus components fills, i els seus assets i mòduls relacionats.

5.2.1.1.4 Mòdul compartit (*Shared*)

El mòdul compartit inclourà els components i directives comuns i les compartirà amb els mòduls que els necessitin.

- El mòdul compartit es crea per a fer ús comú de components i directives disponibles per al seu ús en les plantilles dels components en molts altres mòduls.
- No s'especifiquen proveïdors de serveis singleton a nivell d'aplicació (app-wide) en un mòdul compartit ja que un mòdul carregat de forma lazy (lazy loaded module) que importi el mòdul compartit faria la seva pròpia còpia del servei.
- Un mòdul compartit inclou només components i directives. No hauria d'incloure serveis. Els serveis estan relacionats amb funcionalitats i en la majoria de casos no s'han de incloure en un mòdul compartit.

5.2.1.1.5 Mòdul *Core*

El mòdul Core és un mòdul que s'importarà només una vegada en el moment d'iniciar l'aplicació no s'importarà enlloc més.

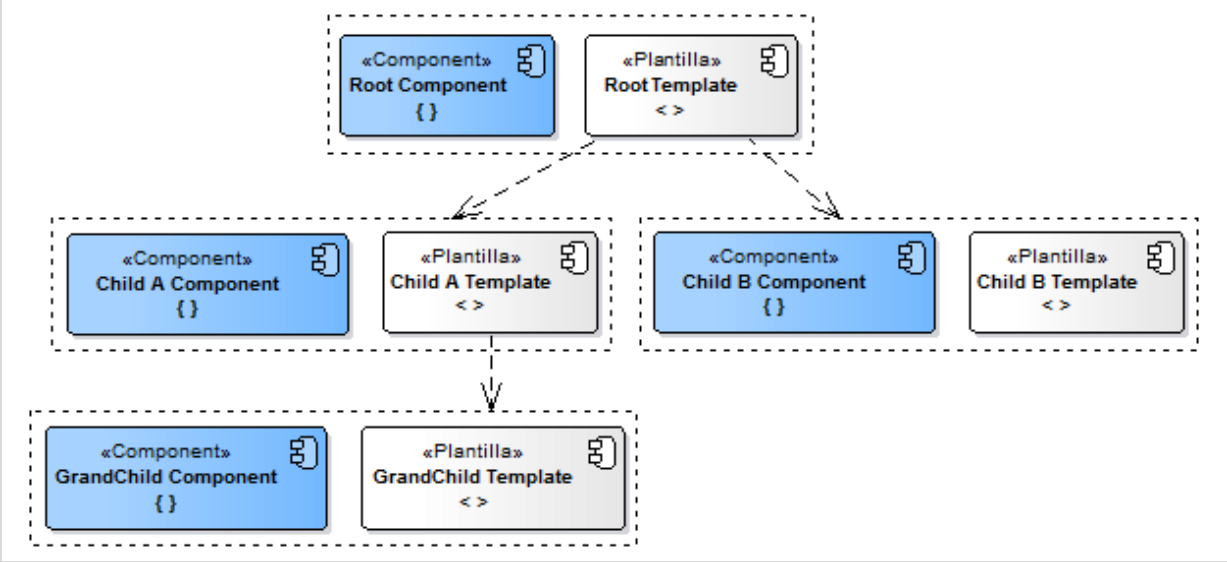
- Els serveis de l'aplicació de tipus Singleton que es registren exactament una vegada, en l'injector Root quan s'inicia l'aplicació, han de ser inclosos en el mòdul Core.
- Tots els components d'un sol ús que apareixen només a la plantilla del component Root AppComponent han de ser inclosos en el mòdul Core.
- Només el mòdul Root ha d'importar el CoreModule en la seva qualitat d'orquestrador de l'aplicació en el seu conjunt.

5.2.1.2 Components

Un component és tot allò que és visible per a l'usuari i que pot ser reutilitzat múltiples vegades dins de l'aplicació Angular. La lògica d'un component es defineix dins d'una classe amb **@Component** com a decorator de TypeScript.

- La responsabilitat d'un component és limita a l'experiència d'usuari.
- Un component ha de fer d'intermediari entre la vista (plantilla) i la lògica de l'aplicació (que sovint inclou alguna noció d'un model).
- Un component ha de contenir propietats i mètodes per a l'enllaç de dades (data binding) i delegar tota funcionalitat de negoci als serveis.

Des del punt de vista dels components l'aplicació Angular pot ser modelada com un arbre de components anidats, tenint cadascun un àmbit aïllat:



Cal diferenciar les responsabilitats entre els diferents tipus de components:

1. Components de tipus contenidor d'alt nivell i específics d'una aplicació amb accés a model de domini de l'aplicació.
2. Components de presentació responsables de la interfície d'usuari i del comportament de les entitats específiques de la seva API (propietats i events específics del component).

5.2.1.2.1 Metadades (Metadata)

Les metadades informen a Angular sobre com ha de processar la classe. En TypeScript s'assigna el ***decorator*** **@Component** a la classe per tal que Angular l'identifiqui com a component.

El decorator **@Component** permet modificar una classe i afegir-hi metadades a les propietats i a les funcions:

- **Selector** – Element (tag) que es fa servir per a informar Angular per tal de crear i inserir una instància d'aquest component.
- **templateUrl** – Ubicació (relativa al mòdul) de la plantilla HTML d'aquest component.
- **providers** – Col·lecció de proveïdors (dependency injection) per als serveis que el component requereix.

5.2.1.3 Plantilles (Templates)

Les plantilles defineixen les vistes (views) dels components. Una plantilla (template) és una forma de **HTML** que informa a Angular sobre com representar gràficament el component.

- El **component** té les responsabilitats **del controller/viewmodel**, per la seva banda la plantilla representa la **view**.
- L'ús de l'element (tag) **<script> està prohibit**.

No cal incloure els elements <html>, <body> i <base>. La resta d'elements estàndard de HTML estan acceptats.

5.2.1.4 Directives

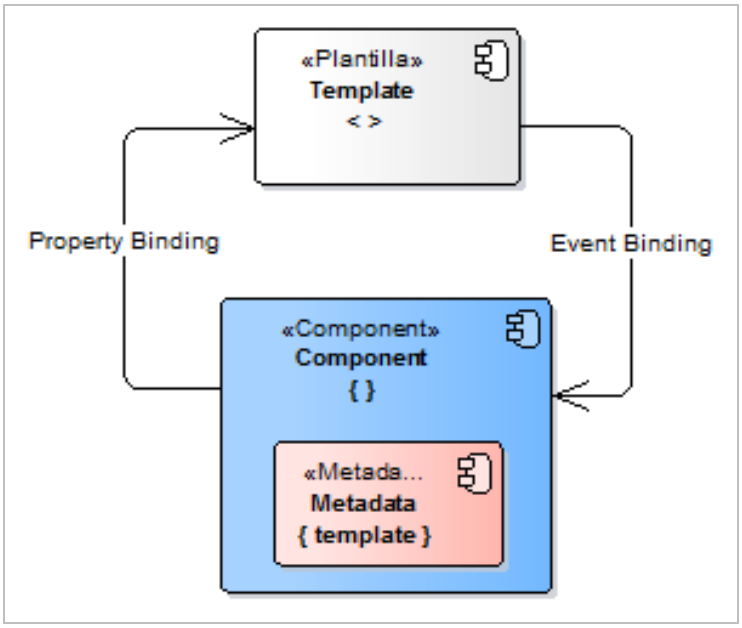
Una directiva modifica el **DOM** per tal de canviar l'aparença, el comportament o la disposició dels elements inclosos en el DOM. Les directives són un dels blocs bàsics de construcció d'aplicacions Angular. De fet, els components d'Angular són en gran part directives amb plantilles.

Hi ha tres tipus principals de directives a Angular:

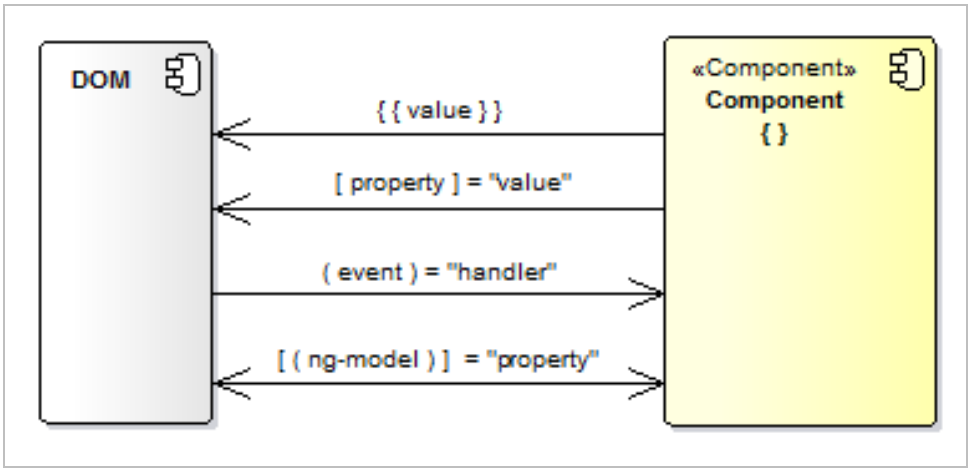
- **Component** – Directives amb una plantilla.
- **Atributs** – Directives que canvien el comportament d'un component o element però no afecten a la plantilla. hauria de funcionar de manera que el component és agnòstic i al detalls d'implementació: **ngClass**, **ngStyle**.
- **Estructurals** – Directives que canvien el comportament del component o element influint com es representa (dibuixa) la plantilla. Directives estructurals incloses a Angular:
  - ***ngIf** : Obligatori l'ús de l'asterisc (*).
  - ***ngFor**: Obligatori l'ús de l'asterisc (*).1
  - **ngSwitch**: No incloure l'asterisc (*) amb ngSwitch.
  - ***ngSwitchCase**, ***ngSwitchDefault**: Obligatori l'ús de l'asterisc (*).

5.2.1.5 Bindings

Els bindings són el mecanisme de coordinació de parts d'una plantilla amb parts d'un component. S'afegiran bindings a la plantilla HTML per tal d'informar a Angular com ha de connectar ambdós costats. Els bindings de dades són també importants per a la comunicació entre components principals (pares) i secundaris (fills).



Com mostra el següent diagrama, hi ha quatre formes de bindings de dades. Cada forma té una direcció - cap al DOM, des del DOM, o en ambdues direccions:



- La **interpolació {{value}}** mostrarà el valor de la propietat indicada del component com una cadena de text. Per exemple: {SampleComponent.Name} mostraria el valor de la propietat SampleComponent.Name dins l'element del DOM (per exemple dins d'un camp de text). Normalment es fa servir aquest mètode quan es tracta de passar cadenes de text com a valors.

- El **property binding [property]** passa el valor de una propietat del component. L'ús més comú serà el de passar el valor d'una propietat del component a una propietat d'un element del DOM. Per exemple: <img [src] = "SampleUrl" (on "SampleUrl" és una propietat del component). Quan establim una propietat d'element a un valor de dades que no sigui una cadena de text, s'ha d'utilitzar el tipus **property binding**.

- El binding de **event (event)** enllaça una acció de l'usuari (per exemple un clic a un botó) a un mètode del component (handler).

- El binding de doble sentit (**Two-way binding**) **[(ng-model)]** serveix tant per mostrar una propietat del component com per actualitzar aquesta propietat quan l'usuari realitza canvis.

Direcció de les dades	Sintaxi	Tipus de binding

One-way
from data source
to view target
	`{{expression}}`
	`[target] = "expression"`
	`bind-target = "expression"`
		Interpolation
		Property
		Attribute
		Class
		Style

One-way from view target to data source	`(target) = "statement"` `on-target = "statement"`	Event
Two-way	`[(target)] = "expression"` `bindon-target = "expression"`	Two-way

5.2.1.5.1 Binding Targets

El destí (*target*) d'un *binding* de dades és un element dins el DOM. Depenent del tipus de *binding*, el destí pot ser una propietat (element | component | directiva), un event (element | component | directiva) event, o (rarament) un nom d'atribut.

La següent taula ho resumeix:

Tipus de binding	Target	Exemples

Property	Element property
	Component property
	Directive property
		`<img [src] = "heroImageUrl">`
		`<hero-detail [hero]="currentHero"></hero-detail>`
		`<div [ngClass] = "{selected: isSelected}"></div>`

Event	Element event
	Component event
	Directive event
		`<button (click) = "onSave()">Save</button>`
		`<hero-detail (deleteRequest)="deleteHero()"></hero-detail>`
		`<div (myClick)="clicked=$event">click me</div>`

Two-way	Event and property	`<input [(ngModel)]="heroName">`
Attribute	Attribute (the exception)	`<button [attr.aria-label]="help">help</button>`
Class	class property	`<div [class.special]="isSpecial">Special</div>`
Style	style property	`<button [style.color] = "isSpecial ? 'red' : 'green'">`

5.2.1.5.2 Events personalitzats (Custom events)

Per tal de generar events personalitzats s'ha de fer servir un **EventEmitter** d'Angular:

- El component crea un **EventEmitter** i l'exposa com una propietat.
- El component executa **EventEmitter.emit(payload)** per tal de disparar un event, passant com a paràmetre informació rellevant (*payload* pot ser de qualsevol tipus).

Els components que tinguin definit un binding a aquesta propietat detectaran i tindran accés a la informació rellevant a través de l'objecte **event$**.

5.2.1.6 Serveis

Servei és una categoria àmplia que abasta qualsevol valor, funció o característica que sigui necessària per a l'aplicació:

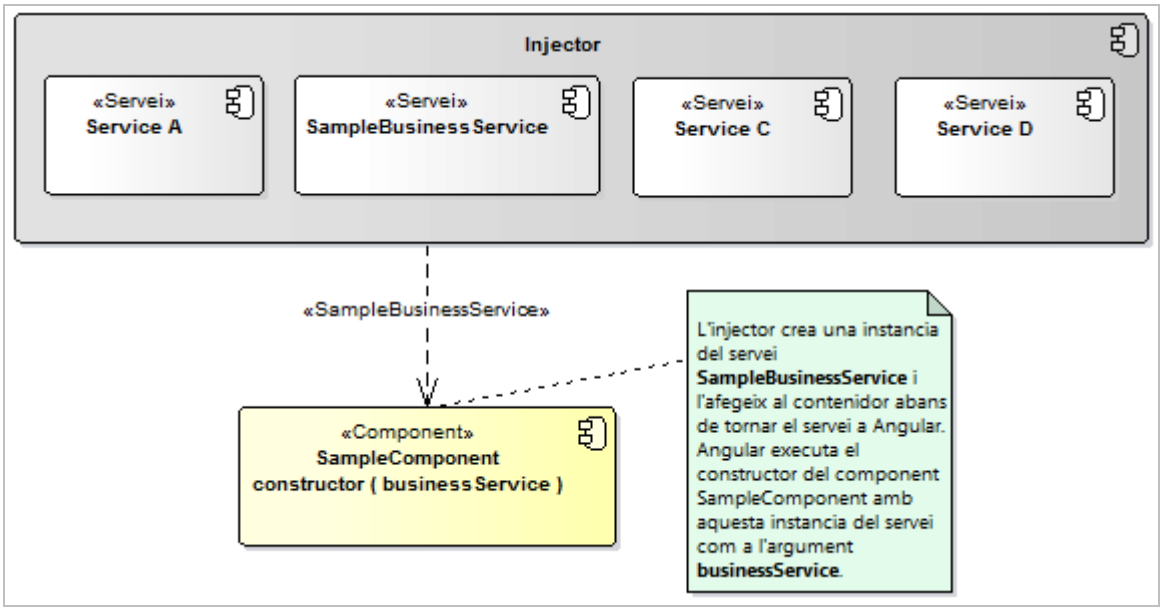
- Un servei ha de ser una classe amb un propòsit concret i ben definit.
- Angular no té una definició específica per a serveis. No existeix una classe base de servei i no existeix cap lloc a on registrar un servei.
- De forma general, els components son els consumidors del serveis.
- Les classes de tipus component han de ser simples. No han d'obtenir les dades des del servidor, validar l'entrada dels usuaris o registrar informació directament a la con-sola. Han de delegar aquestes tasques als serveis.

5.2.1.6.1 Injecció de dependències (Dependency injection)

La injecció de dependències és una manera de proporcionar una **nova instància d'una classe conjuntament amb totes les dependències que requereix**. La majoria de les dependències són serveis. Angular fa servir la injecció de dependències per a proporcionar als components els serveis que necessiten. Angular sap quins serveis necessita un component examinant els tipus dels seus **paràmetres al constructor**.

Quan Angular crea un component, primer demana un injector per als serveis que el component requereix. Un injector manté un contenidor d'instàncies de servei que ha creat anteriorment. Si una instància de servei sol·licitat no està en el contenidor, l'injector en crea una nova i l'afegeix al contenidor abans de tornar el servei a Angular.

Quan tots els serveis sol·licitats han estat resolts i retornats, Angular executa el constructor del component amb aquests serveis com a arguments:

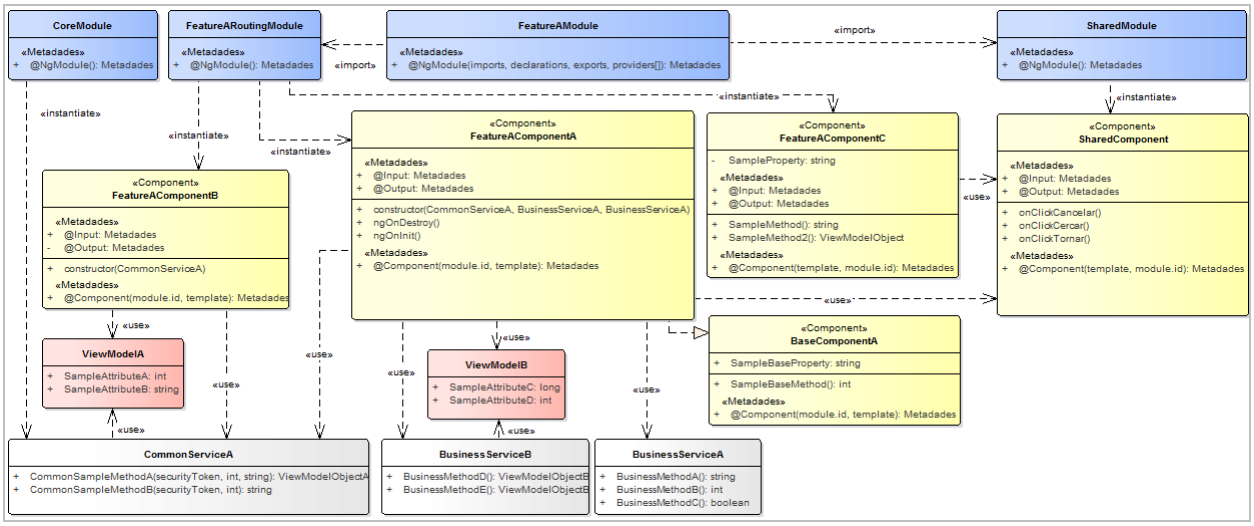


Cal afegir l'anotació **@Injectable()** a la classe de servei per tal d'informar Angular que aquesta classe es pot fer servir amb l'injector de dependències. Punts importants que cal recordar sobre la injecció de dependència a Angular:

- La injecció de dependència està per defecte a Angular i s'utilitza a tot arreu.
- Un injector manté un contenidor d'instàncies de servei que ha creat.
- Un injector pot crear una nova instància de servei fent servir un proveïdor (Provider).
- Un proveïdor és una 'recepta' per a la creació d'un servei.
- Si afegim el paràmetre **{ providedIn: 'root' }** en l'anotació @Injectable, el servei es registrarà directament en l'injector root i d'aquesta forma no hem d'afegir el servei en l'array providers de l'AppModule o del CoreModule.
- Tindrem una única instància del servei (Singleton) tant si fem servir **providedIn: root** o si incloem el servei en l'array **providers** a **l'AppModule** o al **CoreModule**.

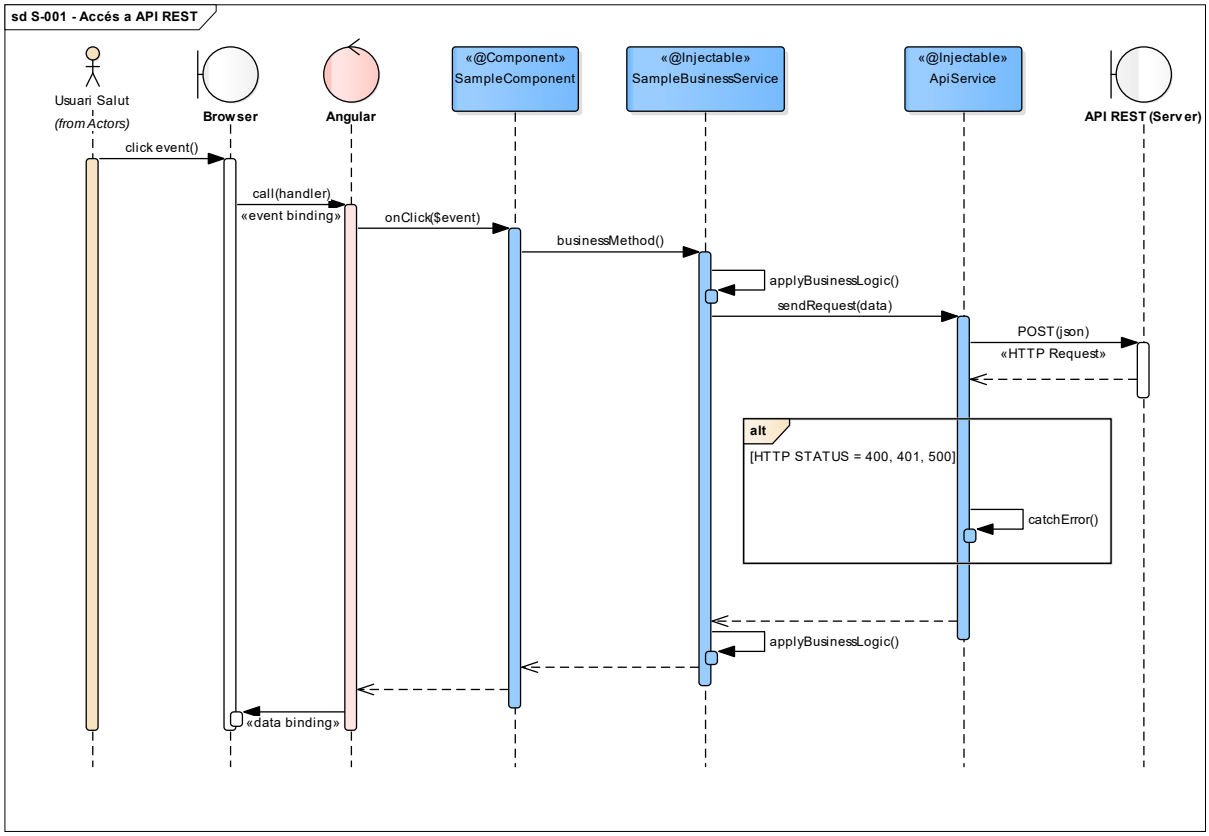
5.2.2 Vista estàtica

En el següent diagrama de classes podem veure un exemple de quins són els diferents elements que intervenen en la capa de presentació:



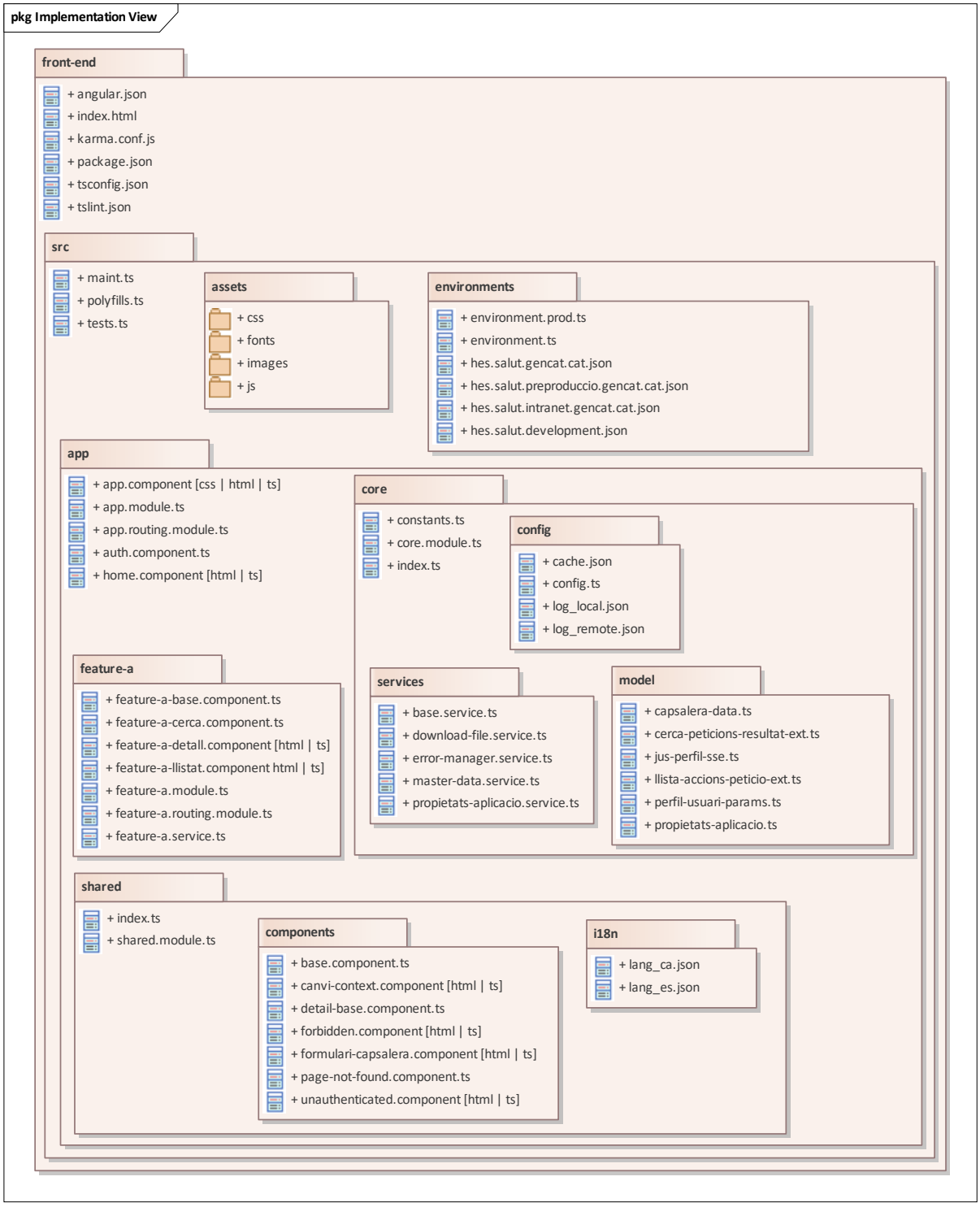
5.2.3 Vista dinàmica

En el següent diagrama es mostra un exemple molt bàsic dels processos dinàmics que s'executen entre els components d'una aplicació Angular des del navegador, enviant informació a la capa REST del servidor i mostrant les dades de nou al navegador:



5.2.4 Vista d'implementació

L'aplicació Angular utilitzarà una estructura basada en components, que és una bona manera d'assegurar-se codi fàcil de mantenir, encapsulant la nostra lògica de negoci. Un component és bàsicament una aplicació independent en general auto-continguda en un únic arxiu o una única carpeta amb cada funció com un arxiu: estil, plantilla, proves unitàries, etc. i la classe de component. La següent figura mostra l'organització en carpetes de les diferents parts d'una aplicació Angular:



A continuació es descriuen les carpetes que formen l'aplicació i la seva responsabilitat:

- src: Directori arrel de la estructura de carpetes que conformen tots els components de l'aplicació.
- assets: Carpeta on s'ubicaran els fulls d'estil, imatges i llibreries de JavaScript de tercers.
- environments: Carpeta con s'ubicaran els fitxers de configuració i classes de definició de paràmetres pels diferents entorns.
- app: Directori arrel de la estructura de carpetes del codi de l'aplicació.
- core: Directori arrel de la estructura de carpetes del codi de l'aplicació que correspon al mòdul Core (Veure Mòdul Core).
- shared: Directori arrel de la estructura de carpetes del codi de l'aplicació que correspon al mòdul compartit (Veure Mòdul compartit (Shared)).
- i18n: Conté els fitxers JSON que contenen els literals i missatges en els diferents idiomes de l'aplicació.
- config: Carpeta amb fitxers de paràmetres de configuració generals de l'aplicació (no depenen de l'entorn).
- layout: Carpeta amb les classes i plantilles dels components que defineixen les diferents parts de les que es composa el disseny visual de l'aplicació.
- featureA: Carpeta que conté tots els fitxers necessaris del mòdul funcional featureA (plantilla, mòdul d'enrutament, codi de la classe del component, estils específics, etc.).

5.2.4.1 Angular CLI

Es farà servir Angular CLI com a única eina de gestió durant el desenvolupament cobrint les següents necessitats:

- Generació automàtica de codi bàsic per a mòduls, components, directives i serveis.
- Gestió del Module Loader (WebPack).
- Anàlisis estàtic de codi TypeScript (via tslint).
- Servidor de l'aplicació per al desenvolupament en local.
- Gestió de la configuració dels tests unitaris.
- Generació de desplegable de l'aplicació (build).

- Angular CLI és una interfície de línia de comandes que ajuda a crear nous projectes Angular des de zero o agregar–ne diversos elements a una aplicació Angular existent (scaffolding).
- El projecte es basarà en un projecte base creat amb aquesta eina que contindrà tots els elements necessaris per posar tot en funcionament i amb una estructura d'aplicacions basada en les bones pràctiques per a un projecte Angular.
- Per a més informació sobre Angular CLI veure: https://github.com/angular/angular-cli/wiki

5.2.4.2 Generació d'estils

S'utilitzarà el framework 'Bootstrap 4' per tal de dotar a l'aplicació d'un disseny que s'adaptarà al dispositiu de l'usuari. Els estils de l'aplicació Angular estaran basats en fulles d'estil CSS generades a partir de fulles d'estil SASS.

SASS (Syntactically Awesome Style Sheets) és una extensió de CSS que permet l'ús de variables, importació d'altres fulls d'estil, regles CSS jerarquitzades, etc. Al mateix temps que manté la compatibilitat amb CSS. En concret es farà servir la sintaxi SCSS per a la programació de les fulles d'estil.

- El procés de generació de fulles d'estil CSS a partir dels scripts SCSS estarà gestionat per l'eina Angular CLI.
- Per a més detalls sobre SASS veure: https://sass-lang.com/

5.2.4.3 Proves unitàries

Per a la programació i execució de les proves unitàries de l'aplicació Angular es faran servir les eines Jasmine i Karma respectivament.

Karma és una eina que ens permet, directament des de la línia de comandes, carregar (iniciar) navegadors i executar tests (per exemple, amb la llibreria *Jasmine*) dins d'aquestes instàncies. Els resultats de les proves es mostren també a la mateixa línia de comandes. *Karma* també és capaç de monitoritzar els arxius de codi en desenvolupament per detectar-hi canvis i executar de nou les proves automàticament.

- El procés d'execució dels tests unitaris mitjançant *Karma* estarà gestionat per l'eina Angular CLI.
- Per a més informació sobre Karma veure: http://karma-runner.github.io/2.0/index.html

Per a la programació de les proves unitàries de l'aplicació Angular es farà servir el framework *Jasmine*. Els tests programats fent servir el *Jasmine* descriuen les proves en un format llegible per a les persones, de manera que és més fàcil d'entendre què està sent provat.

- En crear un nou component o servei via Angular CLI es crearà també el test unitari corresponent (spec).
- Per a més informació sobre *Jasmine* veure: https://jasmine.github.io

5.3 CAPA DE DISTRIBUCIÓ – SERVEIS REST

5.3.1 Serveis RESTFul

L'arquitectura descrita en aquest document es basa en un model de serveis REST, enfocat en tractar la informació i les operacions com a "recursos".

REST (REpresentational State Transfer) ens permet definir APIs de funcionalitats orientades a Internet, utilitzades per qualsevol dispositiu capaç d'efectuar peticions per HTTP.

És un model que aporta major simplicitat que altres solucions basades en crides SOAP, o RPC-XML. El servidor (back-end) proveeix accés als recursos a través dels mètodes exposats en la seva API, i el client (front-end en HTML5) gestiona aquesta informació localment (en aquest cas, com hem indicat, amb el framework Angular).

Aquest tipus d'arquitectura, on el servidor no requereix guardar cap estat conversacional amb els clients que hi accedeixen, es coneix com RESTful. Facilita l'escalabilitat de les aplicacions, i la càrrega del servidor és menor.

Les operacions s'identifiquen per URI's, i els recursos per identificadors globals. REST pot utilitzar diferents tipus de representació de la informació intercanviada amb els seus clients. Actualment, JSON és el format més utilitzat, i és en el que es basarà l'arquitectura REST de Justícia.

Amb REST, s'utilitzen els clàssics mètodes HTTP per gestionar la informació dels recursos de la nostra API web:

- GET: proveeix accessos de només lectura als recursos
- POST: creació de nou recurs
- DELETE: eliminació de recurs
- PUT: modificació de recursos
- OPTIONS: obtenir la llista d'operacions permeses en un recurs

Exemples: una API de gestió de usuaris amb REST:

Consulta d'un usuari específic:

GET /modul-webcontext-root/rest/user/{id}

Consulta d'un llistat paginable d'usuaris:

GET /modul-webcontext-root/rest/user/list?rpp=5&first=0&filters={...}

Creació d'un usuari:

POST /modul-webcontext-root/rest/user

FORM params: { idUsuari : ... , nomUsuari : ... , càrrecUsuari : ...}

5.3.2 Bones pràctiques de disseny de serveis REST

Un servei és considera estrictament RESTful si pot satisfer les següents restriccions:

1. Identificació dels recursos: Els recursos individuals es troben identificats a les peticions mitjançant URIs. A més, aquests recursos es troben conceptualment separats de la representació que es retorna al client.

2. Manipulació dels recursos per mitjà de les seves representacions: El client -sempre que tingui permís i per mitjà de la representació d'un recurs-, té prou informació per a modificar o esborrar aquell recurs al servidor.

3. Missatges autodescriptius: Cada missatge intercanviat entre el client i el servidor conté la informació necessària per processar-lo.

4. Separació client-servidor: D'aquesta manera el client no es preocupa de l'emmagatzematge de les dades i així s'aconsegueix que el seu codi font sigui més portable. Quant al servidor, no es preocupa de l'estat del client, fent que aquest pugui ser més escalable. El desenvolupament del client i del servidor pot ser independent l'un de l'altre mentre la interfície uniforme entre els dos no sigui alterada.

5. Stateless: La comunicació client-servidor no requereix que el servidor hagi de guardar informació del client entre peticions consecutives. Com s'ha dit, cada missatge del client conté prou informació per a satisfer la petició.

6. Cacheable: Les respostes del servidor poden guardar-se en una memòria cache, sigui de manera implícita, explícita o negociada. L'objectiu és minimitzar -en els casos en què sigui possible-, les interaccions client-servidor, fent que el client accedeixi a la representació del recurs guardada en cache i millorant el rendiment del sistema.

7. Layered system: El client no assumeix que hi ha una connexió directa amb el servidor final. Poden existir sistemes software o hardware entre ells. Per exemple, hi pot haver un servidor intermedi que guardi en cache les respostes del servidor. Un altre exemple seria el d'un servidor intermedi que actuï com a balanç de càrrega, millorant l'escalabilitat i minvant els danys davant la possibilitat d'haver de fer front a atacs de denegació de servei (DDoS). Altres elements situats entre el client i el servidor final poden ajudar a millorar les polítiques de seguretat del sistema.

5.3.3 Nomenclatura i responsabilitats

Les responsabilitats dels components d'aquesta capa REST son:

Objecte	Responsabilitats	Nomenclatura
Controlador REST (@RestController)	Són els punts d'entrada al back-end de la nostra aplicació web. Es tracta d'una capa "lleugera". Defineixen l'API de serveis disponibles, i com accedir a la informació dels recursos exposats. Els Controladors REST, per si mateixos, només gestionen la seguretat de les seves crides (autorització, basada en JWT), el tractament de les dades d'entrada i sortida en format JSON, i els codis de retorn HTTP. Deleguen als serveis les crides a negoci de l'aplicació: accés a BD, o altres mòduls o components externs.	xxxController.java extends JusticiaMainController.java
View Model	Representació orientada a objectes del model de paràmetres d'entrada i sortida del serveis REST. Aquesta informació s'obté dels Serveis en forma de Domain Model (o en altres casos, Entities o extensions de les mateixes), però es fa una renderització prèvia al format únic entre front-end i back-end: JSON. El View Model s'adaptarà a les necessitats de definició de la nostra API REST. En alguns casos, la transformació de View Model a o Domain Model serà gairebé immediata. Però per altres tipus de servei, serà necessari tipus més complexes per retornar la informació (llistats paginables, ...)	xxxViewModel.java
View Adapters	Patró Singleton. Aquests objectes	xxxViewAdapter.java

	transformen els View Model en Domain Model, en els dos sentits.	
Spring boot aplication (@SpringBootApplication)	El nostre model REST serà implementat en tecnologia Spring boot. Indica el punt inicial de la aplicació.	xxxApplication.java
Web Security config (@Configuration @EnableWebSecurity)	Defineix la configuració de seguretat de Spring Security sobre l'API REST a nivell d'autorització, gestió d'errors, securitzacio a nivell de paths, etc.	
Handler Tokens JWT	Classe que implementa la generació, validació, i refresc de tokens JWT, i també la gestió la informació de l'usuari que s'emmagatzema en cada token	JusticiaTokenHandler.java JusticiaTokenDetails.java
Control REST d'errors	Components proporcionats per Canigó en el mòdul de REST, que modelen totes les respostes possibles, i gestionen de forma comuna es excepcions, retornant el codi HTTP adient: 200: OK 400: Bad Request 500: Internal Server Error Etc...	JusticiaResponseEntityExceptionHandler.java JusticiaBusinessException.java JusticiaDataAccessException.java JusticiaSystemException.java JusticiaAuthenticationEntryPoint.java
Gestió de Swagger2	Contenidor per configurar la documentació amb Swagger 2 en la nostra API REST	JusticiaSwaggerConfig.java

5.3.4 Format JSON

JSON (*Javascript Object Notation*) és un format de text lleuger per a intercanvi de dades majoritàriament utilitzat en serveis REST, gràcies a la seva simplicitat i pes reduït. Exemple estructura JSON:

```
{
    "menu": {
        "id": "file",
        "value": "File",
        "size": 1024,
        "popup": {
            "menuitem": [
                {
                    "value": "New", "onclick": "CreateNewDoc()"
                },{
                    "value": "Open", "onclick": "OpenDoc()"
                },{
```

```
                    "value": "Close", "onclick": "CloseDoc()"
                }

        }
    }
}
```

En els projectes de Justícia, tots els serveis REST utilitzaran JSON com a format pels paràmetres d'entrada i sortida.

La transformació de JSON a objectes Java és automàtica per les aplicacions Spring Boot. El suport per la conversió de missatges HTTP de Spring selecciona Jackson automàticament si ho troba al classpath de la aplicació.

5.3.5 Seguretat (JWT i Spring Security)

La seguretat en les crides a l'API de serveis REST estarà implementada amb **JWT** (*JSON Web Tokens*) seguint els estàndards fixats per Arquitectura CTTI en quant a projectes Canigó 3.

Aquesta estarà basada en tokens JWT en format OpenID Connect.

Es delega en Spring Security la gestió de la seguretat, JWT es el mecanisme de transport de la informació d'autorització de les crides, basat en tokens que es configura en Spring.

5.3.5.1 Configuració Spring Security

S'han de configurar les propietats per tal que el Spring pugi comprovar la validesa d'un token rebut com part d'una sol·licitud d'execució d'un servei. Aquestes propietats variaran d'un entorn a un altre, el següent exemple es per un entorn de desenvolupament:

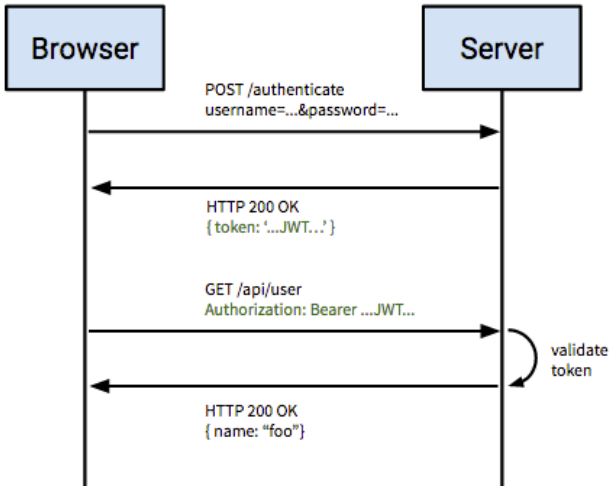
```
spring:
  security:
    oauth2:
      resourceserver:
        jwt:
          issuer-uri: http://integracio.keycloak.justicia.intranet.gencat.cat/auth/realms/ejcat
          jwk-set-uri:
http://integracio.keycloak.justicia.intranet.gencat.cat/auth/realms/ejcat/protocol/openid-
connect/certs
```

S'ha de configurar la seguretat de l'API que exposa el servicio mitjançant la definició d'una classe anotada com @Configuration de Spring Boot, on també es pot configurar el CORS.

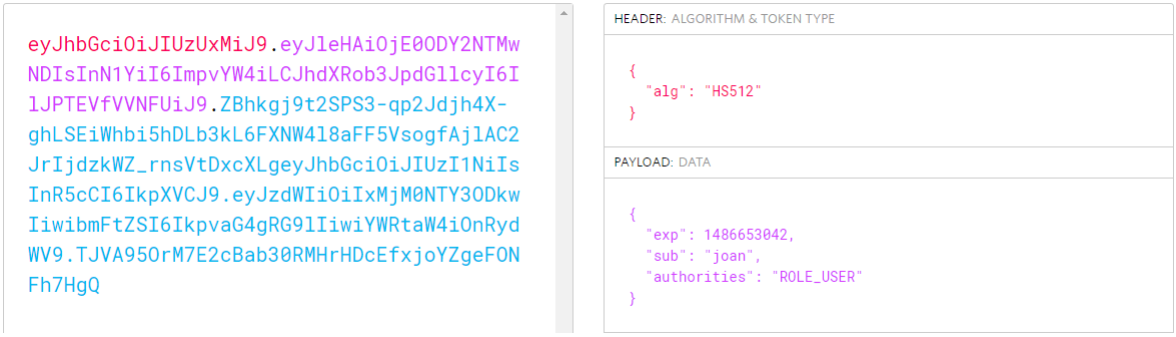
```
@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(prePostEnabled=true)
public class XXXWebSecurityConfig extends WebSecurityConfigurerAdapter {

    ...

    @Override
    protected void configure(final HttpSecurity http) throws Exception {

        http.oauth2ResourceServer().jwt().jwtAuthenticationConverter(jwtAuthenticationConverter()
);
         http.exceptionHandling().authenticationEntryPoint(new
JusticiaAuthenticationEntryPoint());
        http
        .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS).and()
        .cors().and()
        .csrf().disable()
        .authorizeRequests()
        .antMatchers(
                "/v2/api-docs",
                "/configuration/ui",
                ...
                ).permitAll()
        .antMatchers(HttpMethod.OPTIONS).permitAll()
        .anyRequest().authenticated();
    }
    @Bean
    CorsConfigurationSource corsConfigurationSource() {
        final CorsConfigurationSource source =
               …
        return source;
    }
}
```

5.3.5.2 Intercanvi de JWT entre client i servidor

Generalment, totes les crides REST de la nostra API han d'estar protegides, amb intercanvi de tokens JWT en la comunicació HTTP.



Un exemple (molt bàsic) de token JWT amb expiració, credencials, i rols:



L'autorització de crides amb *Spring Security* i JWT serà:

- El servidor sempre verificarà l'existència d'un *Header* amb nom "Authorization" en cada request a l'API REST.
- El format d'aquest *Header* ha de ser "Bearer " + token JWT. Exemple:

```
Bearer eyJhbGciOiJIUzUxMiJ9...
```

- Si el *Header* no existeix, o el token JWT és incorrecte o està expirat, es rebutjarà la petició.
- El control de la resposta de l'autorització es delega en una classe *Entry Point* a mida `JusticiaAuthenticationEntryPoint`, per centralitzar la tipologia i el format d'error a retornar. En tots els casos, es recorda que la nostra API REST sempre acaba retornant missatges JSON (excepte en casos específics d'*streaming* de fitxers, que seran gestionats pel propi component de Canigó de *File Upload*)
- *Spring Security* utilitza les propietats definides per comprovar el token enviat a la *Header* és correcte i vàlid utilitzant els serveis que proporciona Keycloak.
- Si el *token* es vàlid *Spring Security* procedirà a obtenir les credencials contingudes al token.
- En cas contrari, retornarà un error genèric: *401: Unauthorized*

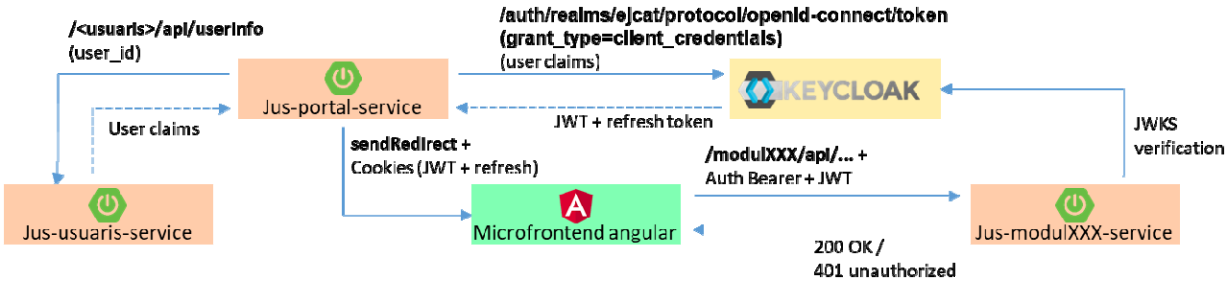
- Si es dona per vàlida la credencial associada al token, i disposa del permís correcte, permetrà executar la petició. En la resposta, adjuntarà un nou token de refresc, per evitar que l'original caduqui després d'enviar-lo moltes vegades amb el servidor.

- En cas contrari, retornarà un error genèric *401 : Unauthorized*

Lògicament, si tot el negociat de l'autorització està basat en un intercanvi de tokens, en algun moment cal definir el punt d'entrada de l'usuari, i generació del primer token.

Les aplicacions hauran de ser configurades per generar un primer token una vegada l'usuari ha superat el repte de GICAR, utilitzant les capçaleres GICAR. A partir de la informació de l'usuari que proporcioni GICAR, hauran d'obtenir la informació de l'usuari i amb aquesta sol·licitar a Keycloak la creació del token JWT per l'usuari amb aquesta informació. La creació d'aquest token proporcionarà com a resultat tant el token com un token de refresc per poder demanar un refresc del token quan estigui proper a caducar.

Al ser Keycloak qui proporciona els tokens, també es l'encarregat de fer les validacions. La configuració de Spring Security permet localitzar els serveis de Keycloak adients per tal que pugui fer aquestes tasques de validació JWKS.

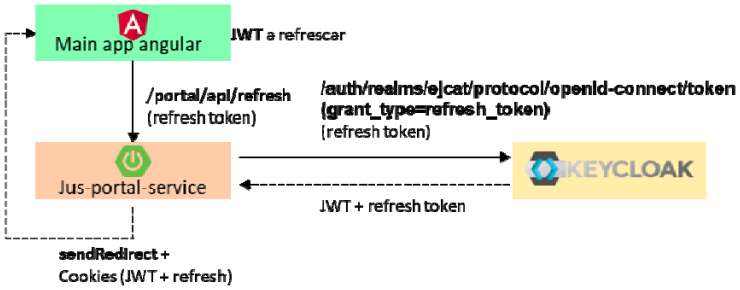


5.3.5.3 Refresh token

La sol·licitud per refrescar el token es realitza des de frontend, en cas de detectar que l'access_token en curs es troba proper a caducar. El servei consisteix en refrescar aquest token periòdicament, enlloc de fer-ho per defecte en cada invocació al backend.

Des del frontend s'emmagatzema el refresh_token rebut al autenticar-se l'usuari amb el sistema.

Un servei del A-Component que realitzi la funció de portal permetrà intercanviar aquest refresh_token per un access_token renovat i vigent. Aquest intercanvi es farà en un servei implementat al Keycloak, de tipus grant_type=refresh_token.



5.3.5.4 Canvi de context entre microfrontends

A aquest escenari es desitja, des de un microfronted Canigo 3.4 + Angular 9 anar cap a un microfrontend amb la mateixa tecnologia i que utilitzi tokens JWT que siguin vàlids pels dos serveis.

Els projectes amb la arquitectura Canigo 3.4 + Angular 9, han d'utilizar la mateixa tecnologia. Així, les serveis de backend utilizen *Spring security* configurat per adreçar les validacions de tokens JWT contra un endpoint OIDC del nostre provider de seguretat OAuth2: el Keycloak.

Per tant, no cal fer cap transformació de tokens, però si proporcionar informació al microfrontend origen sobre a quin microfrontend destí s'ha de dirigir.

El mòdul destí haurà d'oferir un servei REST que rebrà la llista de paràmetres per preparar l'entrada al seu context, i el token JWT per extreure les dades de l'usuari connectat.

La resposta serà una estructura comuna, amb la informació de la URL destí del seu microfrontend (aquesta informació només la coneix ell, i és una propietat que tindrà definida en el seu ConfigMap o application.yml) i els paràmetres necessaris per entrar-hi (en cas de tenir-ne). S'ha de considerar que la informació de l'usuari connectat es troba al token JWT i que s'ha d'enviar a la capçalera *Authoritzation* per securitzar la crida al servei i per tant, no cal que estigui definit com paràmetre d'entrada i el token s'ha de poder validar i processar pel backend destí.

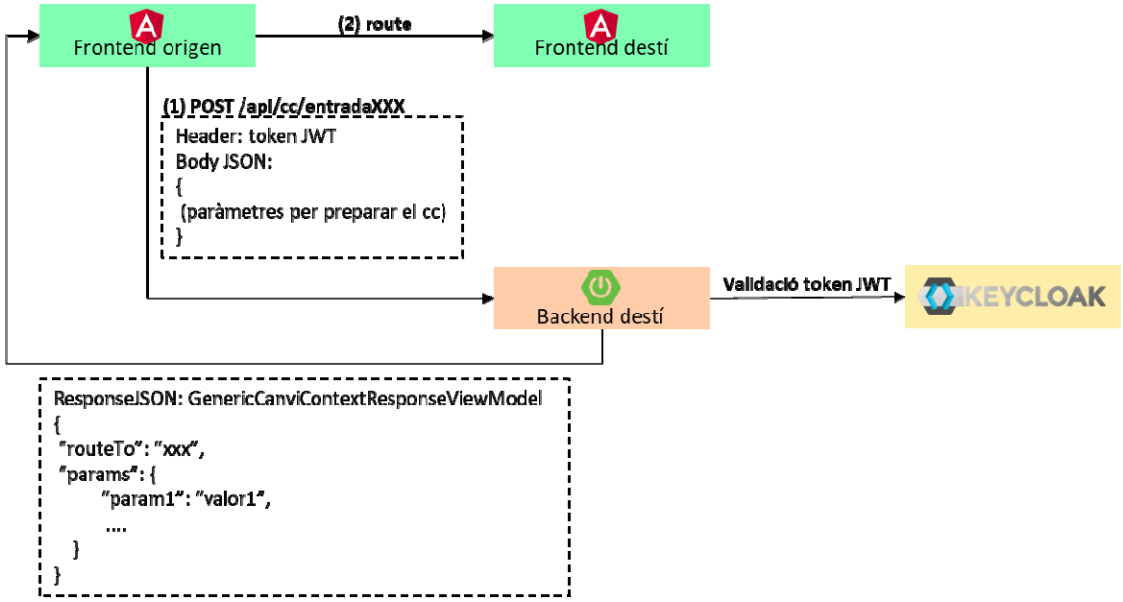
Així el mòdul destí que permet que altre mòdul realitzi un canvi de context sobre ell (es a dir que defineix un mecanisme per ser obert des de un altre microfrontend) ha de implementar:

Servei REST:

- Endpoint url del servei Rest de canvi de context al destí. Per exemple: https://.../jus-modulDestiXXX-service/api/XX/cc/entradaXXX
- Com a mínim una capçalera requerida Authoritzation amb el token JWT.
- Cos: Format propi del cos segons les necessitats del destí per tenir la informació necessària per tal de poder ser obert. Al seu backend s'implementa com una classe pròpia del model.
- Resposta: Com resultat s'utilitza una resposta estàndard que conté la informació necessària per tal que el mòdul origen faci la crida cap al mòdul destí. Es tracta de la següent classe de la llibreria jus-canigo34-cloud-lib:

```
cat.gencat.justicia.common.model.cc.GenericCanviContextResponseViewModel
    {
            "routeTo": "xxx",
            "params":
                    {
                            "param1": "xxx",
                            ...
                    }
    }
```

Amb aquesta informació el frontend origen pot realitzar el seu canvi de context per obrir el microfrontend destí.



5.3.5.5 Canvi de context de microfrontend cap a intranet del sistema EjCat.

Aquest escenari és més complex, doncs entren en funcionament les diferents arquitectures que tenim en els projectes intranet d'EJCAT

Arquitectura	Funcionament de la seguretat
Canigó 1.4 Canigó 3.1 JSF	**Cookie** administrada per *Filters* de la s*hared-library SSO.jar*
Canigó 3.2 REST	**Token JWT** administrat pel mòdul JusTokenHandler-ear
Canigó 3.4 REST	**Token JWT** validat via *JWKS* contra *endpoint Oauth2* de KeyCloak

Els canvis de context entre mòduls intranet d'EJCAT, on es combinen diferents arquitectures, els gestiona el mòdul Portal implementat en Canigó 1.4 (d'aquí endavant : "POR-Canigo1.4").

Concretament, aquest mòdul ofereix un endpoint dins el seu MVC d'Struts per rebre peticions de canvis de context entre mòduls, i executar aquest canvi de mòdul:

```
https://.../portal/AppJava/canviContext.do?reqCode=canviContextSSO
```

Abans de fer la crida al POR-Canigo1.4, però, cada mòdul origen Canigó 3.4 + Angular 9 haurà d'oferir un servei REST per preparar els paràmetres del canvi de context.

Servei REST:

- Endpoint url del servei Rest de canvi de context a l'origen. Per exemple https://.../jus-modulOrigenXXX-service/api/XX/cc/preparaEntradaXXX
- Com a mínim una capçalera requerida Authorization amb el token JWT.
- Cos: Format propi del cos segons amb la informació necessària per tal de poder preparar el canvi de context. Al backend de l'origen s'implementa com un DTO.
- Resposta: Com resultat s'utilitza una resposta estàndard que conté la informació necessària per tal que el mòdul origen faci la crida cap al mòdul destí iniciant el canvi de context. Es tracta de la següent classe de la llibreria jus-canigo34-cloud-lib:

```
cat.gencat.justicia.common.model.cc.CanviContextResponseModel
{
  "paramsCC": "..."
}
```

Addicionalment, el mòdul POR-Canigo1.4 necessita rebre també un token que NO sigui JWT (doncs aquest mòdul no treballa amb aquest format) si no amb un format diferent que va ser definit al seu moment com Token de Canvi de Context (o TokenCC).

Aquest token és comú, i per tant s'ha d'implementar un servei al mòdul Portal (li direm "POR-Cloud" en aquesta endavant) que prepararà finalment tota la informació per la crida de canvi de context.
A més, la informació ha d'estar codificada en un format concret, que el POR-Canigo1.4 pugui entendre.

Servei REST:

- Endpoint url. Per exemple https://.../jus-por-cloud-service/api/XX/cc/tokenCC
- Com a mínim una capçalera requerida Authoritzation amb el token JWT.
- Cos: Dades del canvi de context que es vol realitzar. S'ha d'utilitzar el format definit per la següent classe de la llibreria jus-canigo34-cloud-lib:

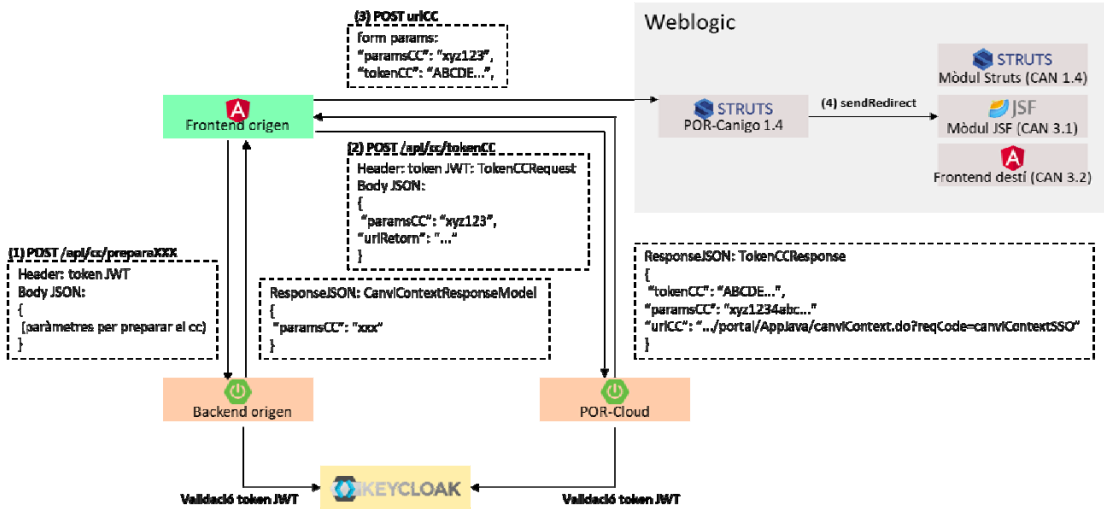
```
cat.gencat.justicia.common.model.cc.TokenCCRequest
{
  "paramsCC": "...",
  "urlRetorn": "..."
}
```

  - paramsCC: preparats pel servei REST anterior del mòdul origen
  - urlRetorn: route de frontend en cas que s'hagi de poder tornar del mòdul destí al mòdul origen de nou. Aquesta informació és una propietat que coneix el frontend.

- Resposta: Com resultat s'utilitza una resposta estàndard que conté la informació necessària per tal que el mòdul origen faci la crida cap al POR-Canigo1.4. Es tracta de la següent classe de la llibreria jus-canigo34-cloud-lib:

```
cat.gencat.justicia.common.model.cc.TokenCCResponse
{
  "tokenCC": "...",
  "urlCC": "...",
  "paramsCC": "..."
}
```

  - tokenCC: és el token que POR-Canigo1.4 necessita per determinar qui està demanant el canvi de context
  - urlCC: aquesta és l'adreça del servlet de POR-Canigo1.4 que hem indicat anteriorment que s'ocupa dels canvis de context on mesclem arquitectures
  - paramsCC: és possible que el POR-Cloud hagi d'incloure algun paràmetre addicional de forma general per a tots els mòduls MJ. Per aquest motiu, el servei retorna de nou aquest paràmetre que ja havia rebut d'entrada.



5.3.5.6 Canvi de context intranet del sistema EjCat cap a microfrontend

L'escenari més habitual d'aquest tipus de canvi de context serà el de retorn.

És a dir, un microfrontend fa un canvi de context a una aplicació intranet d'EJCAT amb diferent arquitectura (Canigo1.4+Struts, Canigo3.1+JSF, Canigo3.2+REST), i després s'ha de tornar al microfrontend.

No es contempla, funcionalment, que una aplicació intranet EJCAT per si mateixa necessiti fer un canvi de context cap a un microfrontend.

Partim de la base que l'aplicació origen (intranet EJCAT) ha obtingut un TokenCC, i s'ha fet un redirect al microfrontend Angular.

El primer que necessita fer el mòdul destí es traduir aquest TokenCC (que es un format comú a totes les arquitectures intranet EJCAT) en un parell de tokens JWT vàlids (access token + refresh token) i específics per la seva arquitectura

Per aquest intercanvi, el mòdul POR-Cloud oferirà un servei REST de traducció de tokens.

Aquest servei rebrà el tokenCC, extraurà la informació de l'usuari (els seus claims), i invocarà al proveïdor d'identitats per generar els tokens JWT vàlids per invocar al backend destí MJ.

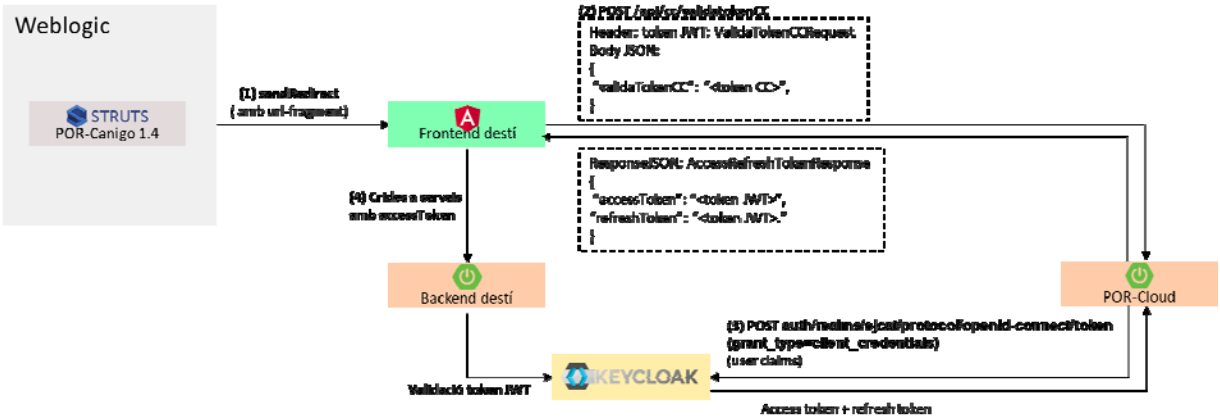
Servei REST:

- Endpoint url. Per exemple https://.../jus-por-cloud-service/api/XX/cc/validaTokenCC
- No es possible enviar un token JWT, encara no tenim un i es el resultat d'aquesta crida.
- Cos: El tokenCC rebut com a fragment. S'ha d'utilitzar el format definit per la següent classe de la llibreria jus-canigo34-cloud-lib:

```
cat.gencat.justicia.common.model.cc.ValidaTokenCCRequest
{
  "validaTokenCC": "<token CC>"
}
```

- Resposta: Com resultat s'utilitza una resposta estàndard que conté els tokens. El mòdul portal-cloud s'ha d'encarregar d'obtenir aquests tokens del proveïdor d'identitats. El resultat té el format de la següent classe de la llibreria jus-canigo34-cloud-lib:

```
cat.gencat.justicia.common.model.cc.AccessRefreshTokenResponse
{
  "accessToken": "<token JWT>",
  "refreshToken": "<token JWT>"
}
```



5.3.5.7 Canvi de context entre microfrontends i altres sistemes

Aquest cas d'ús de moment no es contempla: que es pugui fer canvis de context entre, per exemple aplicacions Extranet del sistema EjCat, i entraria en un segon abast de requeriments en cas de necessitar aquesta funcionalitat.

5.3.6 Definició dels Controllers i els mètodes de l'API RESTful

Totes les classes que siguin contrelles s'anotaràn amb **@RestController** i poden extendre de la següent classe abstracta inclosa en la llibreria justicia-canigo3.4-cloud-lib, que ofereix un tractament comú de temes com per exemple la paginació amb Spring Data:

```
cat.gencat.justicia.common.controller.JusticiaMainController
```

Cal definir en la configuració de l'aplicació el context path dels serveis que s'exposaran en la nostra API. Aquest path ha de tenir versionat obligatori en la seva nomenclatura.

```
server:
  servlet:
    context-path: /api/v1
```

Amb la configuració anterior, els nostres serveis estaran accessibles a partir de:

```
https://NOM_MODUL.namespace.domini/api/v1/...
```

Tal com s'explica a la guia [CU_ARQ022_Gestió_d_excepcions] de gestió d'excepcions cada aplicació ha d'implementar una classe anotada amb **@ControllerAdvice** a la que gestionar les excepcions pròpies de cada aplicació. Es pot extendre la classe de la llibreria justicia-canigo3.4-cloud-lib si és aplicable el tractament per defecte d'algunes de les excepcions.

```
cat.gencat.justicia.common.controller.JusticiaResponseEntityExceptionHandler
```

Aquesta classe s'ocupa de convertir les excepcions en missatges d'error unificats, segons la internacionalització (i18n) del mòdul, i retornar l'estructura d'error unificada (codi i descripció de l'error). En el nostre cas, utilitzem el següetnDTO de Canigó:

```
cat.gencat.ctti.canigo.arch.web.rs.response.ResponseError
```

5.3.7 Exposició de l'API REST amb Swagger

Swagger 2 és un framework de disseny i documentació d'API's de serveis REST. Permet generar de forma senzilla i intuïtiva la documentació dels serveis publicats en cada mòdul : mostrant model de dades d'entrada i sortida, les capçaleres, codis de retorn, etc...

El següent T-Component comú d'Arquitectura per configurar Swagger 2. Es tracta d'una classe amb les anotacions @Configuration i @EnableSwagger2 que es troba a la llibreria comuna jus-canigo34-cloud-lib

```
cat.gencat.justicia.common.configuration.JusticiaSwaggerConfig
```

Un cop arrencat el projecte amb Spring Boot, es pot consultar la documentació generada de l'API en el següent *endpoint*:

```
.../swagger-ui.html
```

També es pot recollir en format JSON, per importar-la en eines com Api Manager:

```
.../v2/api-docs
```

5.3.8 Stack de logging distribuït

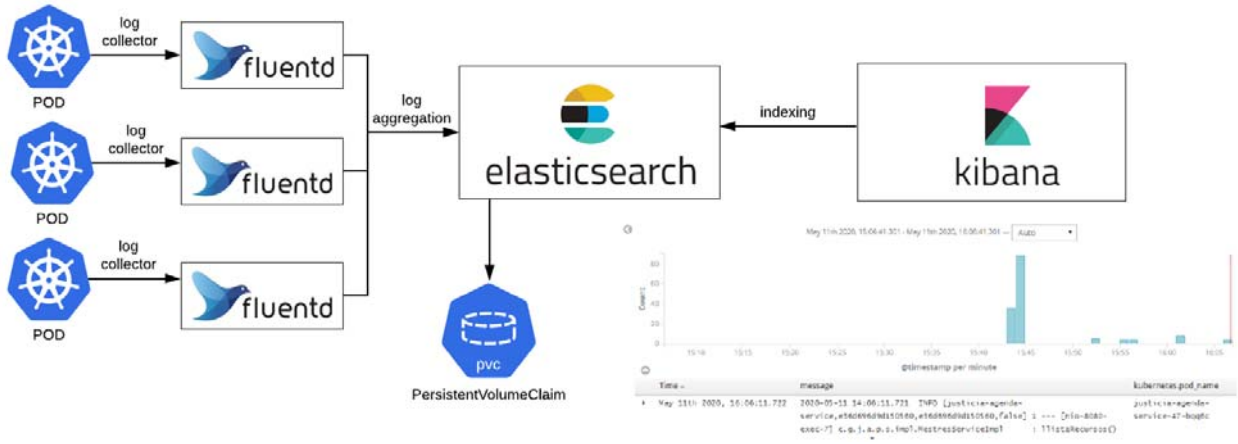
Al desplegar les nostres aplicacions en contenidors, resulta difícil gestionar la consulta dels logs que aquests generen, ja que aquesta informació es volàtil i lligada a la vida del propi contenidor.

Per aquest motiu, disposarem de l'stack **EFK** (ElasticSearch+FluentD+Kibana) per consultar els logs.

Els serveis han d'assegurar que escriuen els seus logs per sortida estàndard, en format JSON.

Un agent de **FluentD** recull els logs de cada contenidor, i els indexa en una base de dades **ElasticSearch**, optimitzada per a consultes ràpides de text.

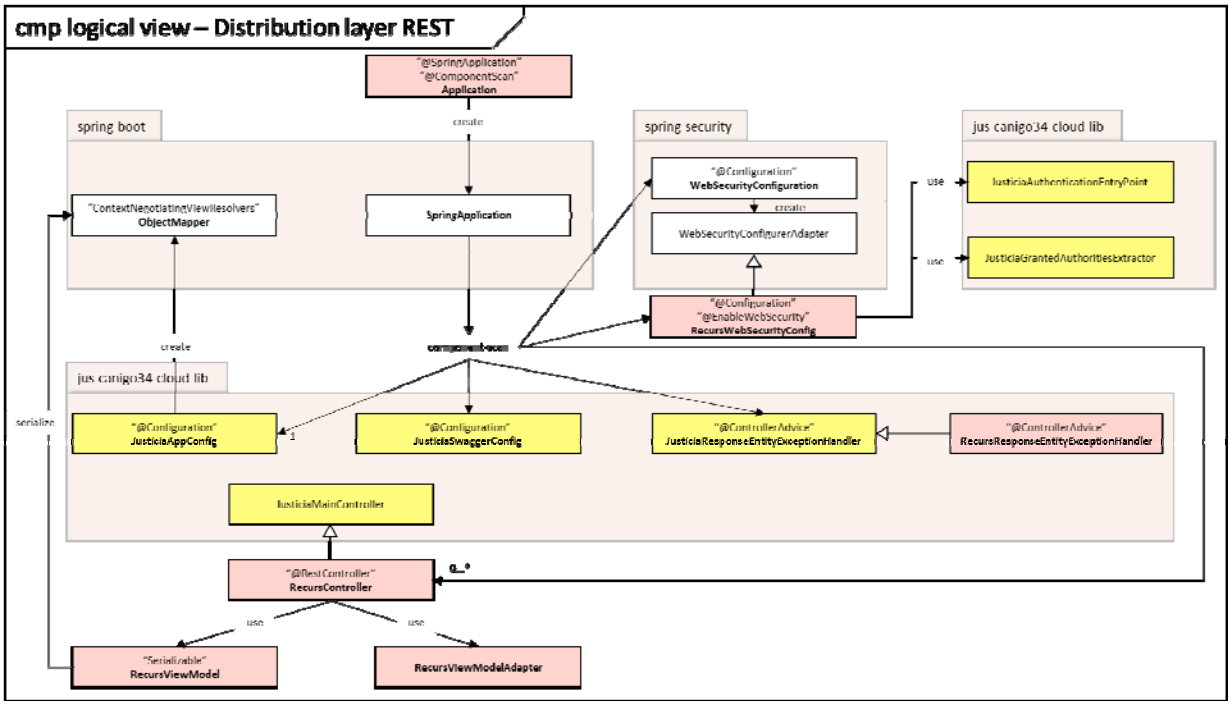
Finalment, s'habilita una eina web **Kibana** per consultar aquests logs, segons diferents paràmetres de filtre (aplicació, contenidor, nivell de traça, timestamp,...)



Per altre banda, per fer la traçabilitat d'una operació que pot passar per diferents serveis, es configurarà un agregador addicional de logs anomenat **Jaeger**, que posteriorment permet fer visualitzacions agrupades de logs pertanyents a una mateixa invocació, a través d'identificadors interns basats en *Spring Sleuth*.

5.3.9 Vista estàtica de la capa de distribució REST

Al següent diagrama de classes podem veure els diferents elements que intervenen en aquesta capa:



En fons blanc són els components ubicats en llibreries incloses amb Canigó 3:

- Spring-boot: incorpora l'aplicació Spring, els mecanismes d'autoconfiguració de Spring boot i una configuració bàsica del processador dels missatges d'entrada i sortida a format JSON (ObjectMapper)
- Spring-security: incorpora tota la gestió de la seguretat, el proveïdor d'autenticació i autorització.

En fons groc marquem els T-Components fets per Arquitectura. Pel cas de REST, la llibreria és:

- jus-canigo34-cloud-lib: diferents T-Components comuns per la capa de distribució REST dels mòduls.

Els REST Controllers, són els objectes que acaben formant la definició de la nostra API de serveis, amb les seves diferents operacions: indicant el format d'adreça (URI), la ubicació dels paràmetres, els mètodes acceptats, i el format de resposta de cada un (JSON principalment).

Els @RestControllers enllacen amb la capa de negoci a través dels Business Components (annotació @Service), que actuen com a *Service Facade* de comunicació amb altres mòduls, o d'accés a la base de dades.

Els objectes retornats per aquests BC sempre seran de tipus Domain Model. Els View Model Adapters s'ocupen de transformar-los al model definit en la nostra API REST, i els identifiquem com View Model objects.

Els Controllers s'ocuparan també del tractament dels codis de retorn HTTP adients segons el resultat de l'operació.

Els mòduls importaran la configuració de Swagger a efectes de documentació de la seva API REST.

Un cop inclòs el context de configuració, l'API de REST serà publicada també en format Swagger 2, i aquest podrà ser editat pels diferents editors de Swagger existents.

5.3.10 Comunicació entre la capa de distribució REST i la capa de negoci

La capa REST no farà cap invocació directa a la base de dades o a altres serveis.

Aquesta responsabilitat es delegarà sempre a la capa de negoci de l'aplicació, en els Components que tenen l'annotació @Service:

```
org.springframework.stereotype.Service;
```

Per tant, els REST Controllers únicament contacten amb els Components injectats amb Spring directament emprant l'anotació @Autowired:

```
org.springframework.beans.factory.annotation.Autowired
```

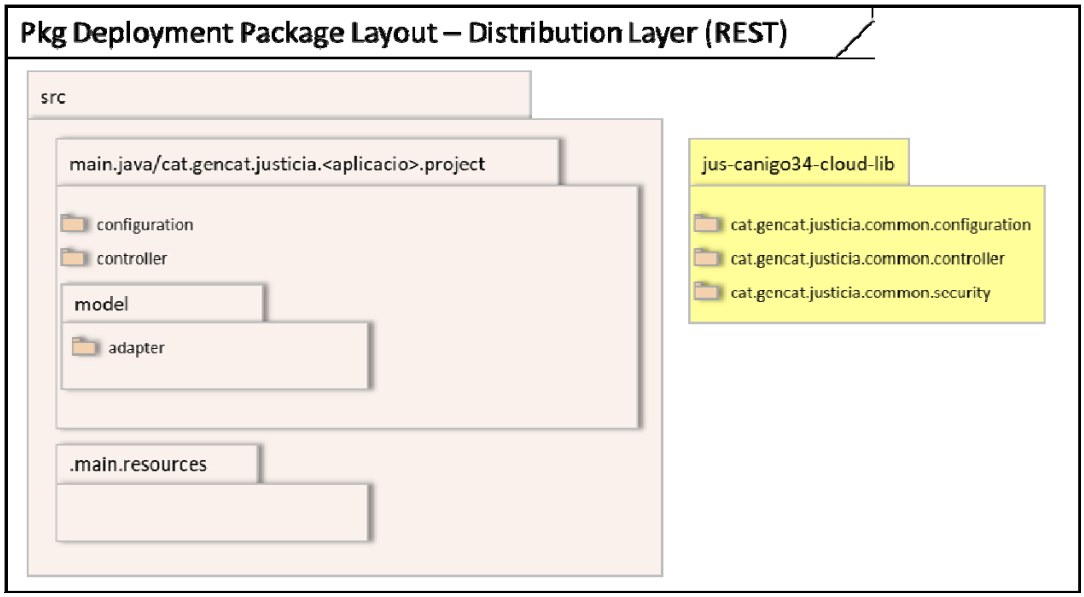
I amb els View Model Adapters faran una transformació de les dades rebudes o enviades a la capa de negoci (DomainModel), per adaptar-ho al model de dades dels serveis REST.

Exemple d'un Controlador REST de serveis per un recurs de Països, amb enllaç al seu Components corresponent, i tractament de l'adapter:

```
package cat.gencat.justicia.exemple.project.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;
import cat.gencat.justicia.exemple.model.view.PaisViewModel;
import cat.gencat.justicia.exemple.model.view.adapter.PaisAdapter;
import cat.gencat.justicia.exemple.model.domain.PaisDomainModel;

import cat.gencat.justicia.exemple.project.service.impl.PaisService;

...

@RestController
@RequestMapping({"/pais"})
public class PaisController extends JusticiaMainController {

        /** pais service */
        @Autowired
        PaisService paisService;

        @RequestMapping(value = "/traduccio/{codiIdioma}/{paisId}",
                method = RequestMethod.GET,
                produces = "application/json; charset=UTF-8")
        @ResponseBody
        public ResponseEntity<PaisViewModel> getTraduccioPais(@PathVariable("codiIdioma") String
codiIdioma, @PathVariable("paisId") String paisId) {
                PaisDomainModel domainPais = paisService.getTraduccioPais(paisId, codiIdioma);
                return ResponseEntity.ok(PaisAdapter.adapt(domainPais));
        }}
}
```

5.3.11 Vista d'implementació

A continuació es presenta un diagrama de la vista d'implementació enfocat només a la capa REST.
Per un costat tenim la implementació en els projectes, i per altre, la llibreria de T-Components comú per REST:



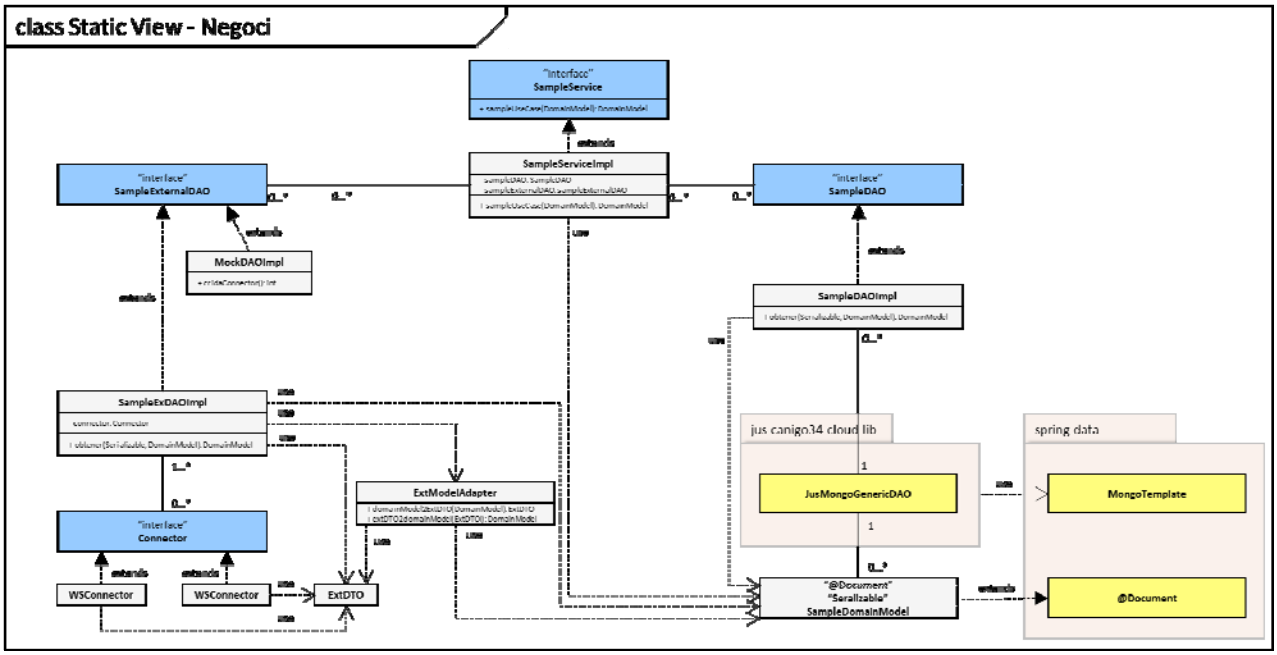
5.4 CAPA DE NEGOCI

5.4.1 Nomenclatura i responsabilitats

Les responsabilitats dels components d'aquesta capa REST son:

Objecte	Responsabilitats	Nomenclatura
Servei (@Service)	Són els objectes cridats pels controllers i que es responsabilitzen de realitzar l'execució de lògica de negoci. Si necessiten accedir a bases de dades ho demanen a la capa d'accés a dades Deleguen als data access objects les crides d'accés a dades i a altres serveis l'execució de lògica de negoci d'aquells serveis.	xxxController.java extends JusticiaMainService.java
Data Transfer Object (DTO)	Representació d'un objecte que representa el format d'un conjunt d'informació, un POJO. Aquesta informació s'intercanvia entre els Serveis i altres serveis o entre els serveis i els controllers.	
Data Access Object (DAO)	Son els objectes responsabilitzats del accés a les dades. Són executats des dels serveis.	JusticiaMongoGenericDAO.java
Document @Document	A un model de base de dades documental, un document es l'element d'accés a la informació. A MongoDB la informació s'emmagatzema en col·leccions de documents que poden ser polimòrfics. Un Document pot contenir altres documents com part d'ell.	

5.4.2 Vista estàtica



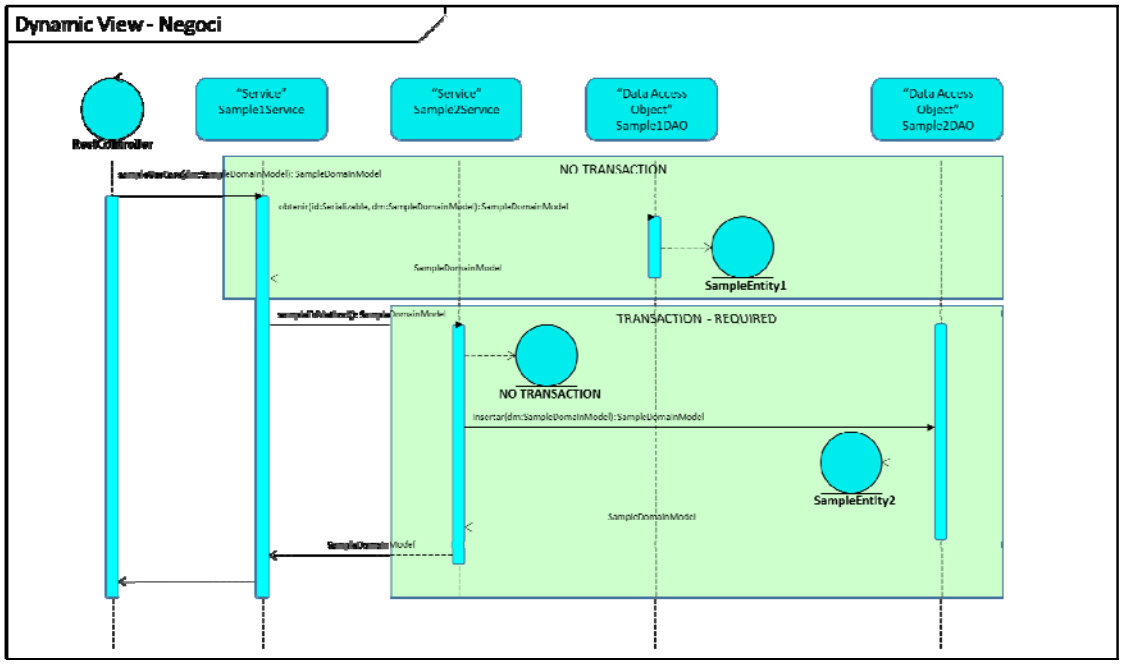
El negoci es modelat mitjançant la definició de serveis de negoci. Distingirem entre la seva interfície (Service) i la implementació (ServiceImpl).

Els paràmetres dels mètodes de negoci poden ser

- Objecte de domini (DomainModel)
- Tipus bàsics o primitius Java.

Un *"Service"* pot invocar a altres *"Services"* durant l'execució del seu mètode de negoci. També pot invocar al T-Component DAO per tal d'interaccionar amb BBDD o External DAO per interaccionar amb sistemes externs. Per entorns de desenvolupament, pot ser interessant utilitzar *"Mock Objects"* quan es tinguin limitacions de connectivitat.

5.4.3 Vista dinàmica

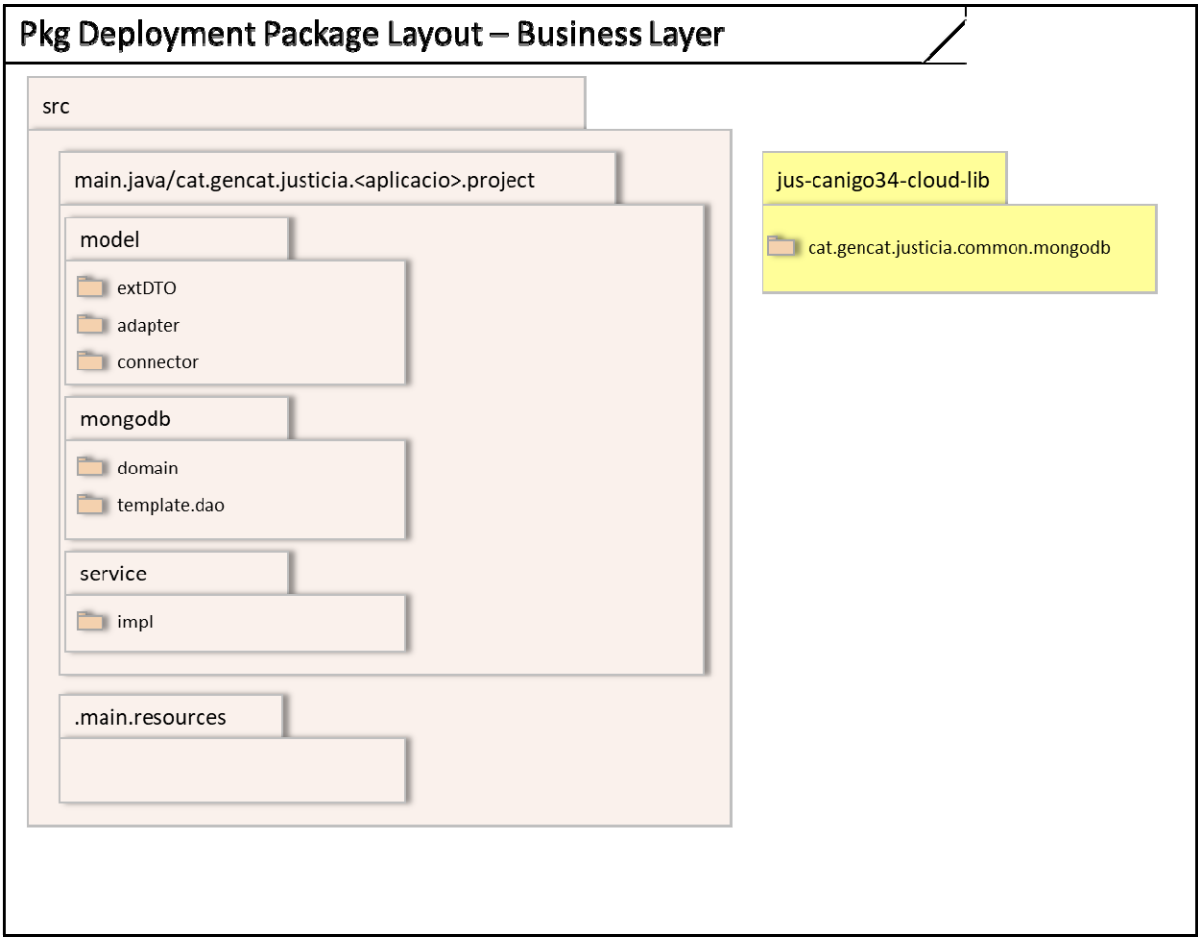


Cal destacar del diagrama anterior:

- Per tal de resoldre el cas d'ús, el controlador REST (RESTController) invoca a un ServiceImpl que farà dos accions independents. Primer realitza una cerca sense englobar-la dins una transacció per obtenir un determinat objecte del model. Posteriorment es crida a un altre ServiceImpl que inicia una transacció per tal de insertar un nou objecte a base de dades

5.4.4 Vista d'implementació

A continuació es presenta un diagrama de la vista d'implementació enfocat només a la capa de Negoci.



En el package **model.extdto** estaran els objectes de domini de serveis externs modelats com DTO's, dins el package **servcice** trobarem els components de negoci necessaris per tal d'implementar els casos d'ús especificats.
Les entitats de comunicació amb la base de dades (Document) estaran al package **mongodb**

5.5 CAPA D'INTEGRACIÓ

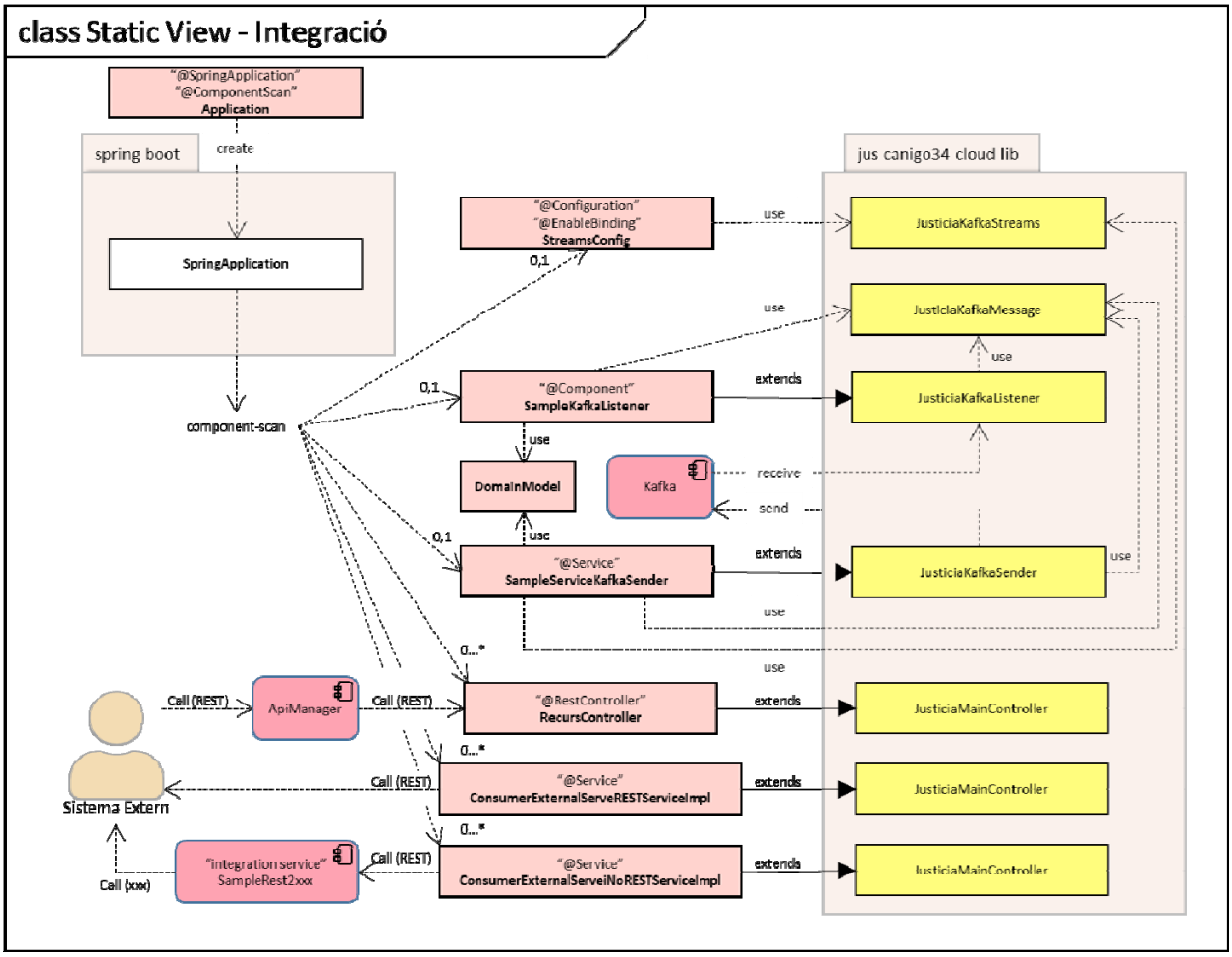
5.5.1 Nomenclatura i responsabilitats

Les responsabilitats dels components d'aquesta capa REST son:

Objecte	Responsabilitats	Nomenclatura
KafkaStreams	Definició dels elements de comunicació amb Kafka	xxxKafkaStreams.java
KafkaListener	Component responsable de romandre a l'espera de l'arribada d'un missatge pels elements de comunicació definits i de gestionar i processar el missatge . Es responsable de la gestió dels errors.	xxxKafkaStreams.java
KafkaSender	Component responsable de l'enviament de missatges utilitzant els elements de comunicació definits.	xxxKafkaSender.java
Spring boot aplication (@SpringBootApplication)	El nostre model REST serà implementat en tecnologia Spring boot. Indica el punt inicial de la aplicació.	xxxApplication.java

5.5.2 Vista estàtica

La vista estàtica corresponent a la vista lògica d'integració és:



A la capa d'integració, s'explica els mecanismes pels cuals els sistemes externs poden accedir als serveis interns. Existeixen 2 tipus principals d'integració cap als serveis de negoci:

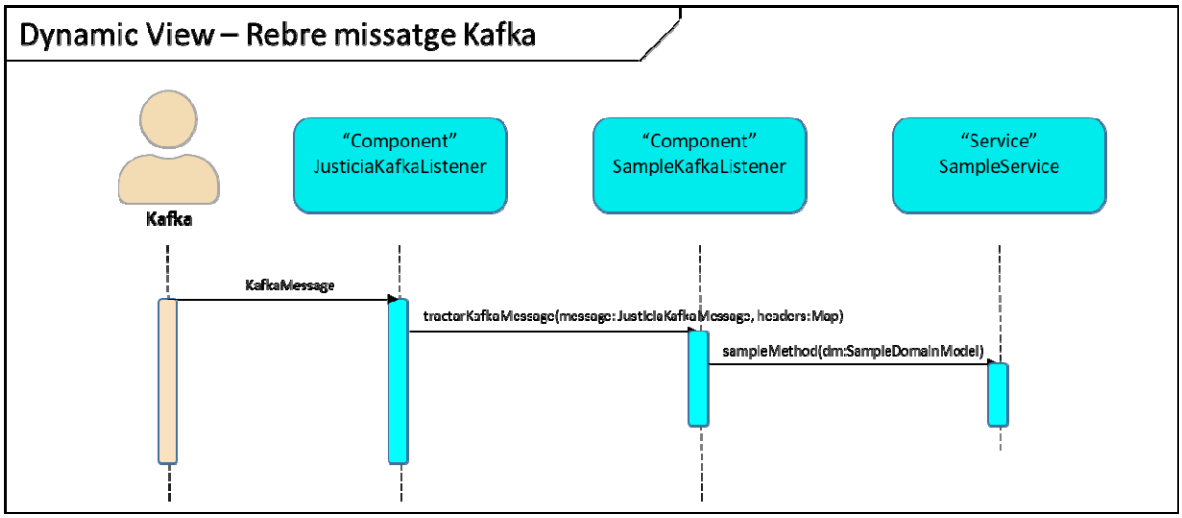
- Asíncrona: els serveis poden utilitzar els T-Components (JusticiaKafkaListener) de la llibreria jus-canigo34-cloud-lib per rebre missatges de Kafka.
- Síncrona: per tal que serveis externs puguin demanar l'execució de serveis de negoci interns s'han de acomplir diferents condicions:
    - Els serveis interns poden ser accedits de manera directa o via Api Manager segons les diferents opcions exposades al cas d'ús CU_ARQ026 Accés a serveis negoci.
    - El sistema extern ha de consumir el servei utilitzant el protocol REST.

Per tal que des de els serveis interns es pugin consumir serveis de sistemes externs temin els següents tipus principals d'integració:

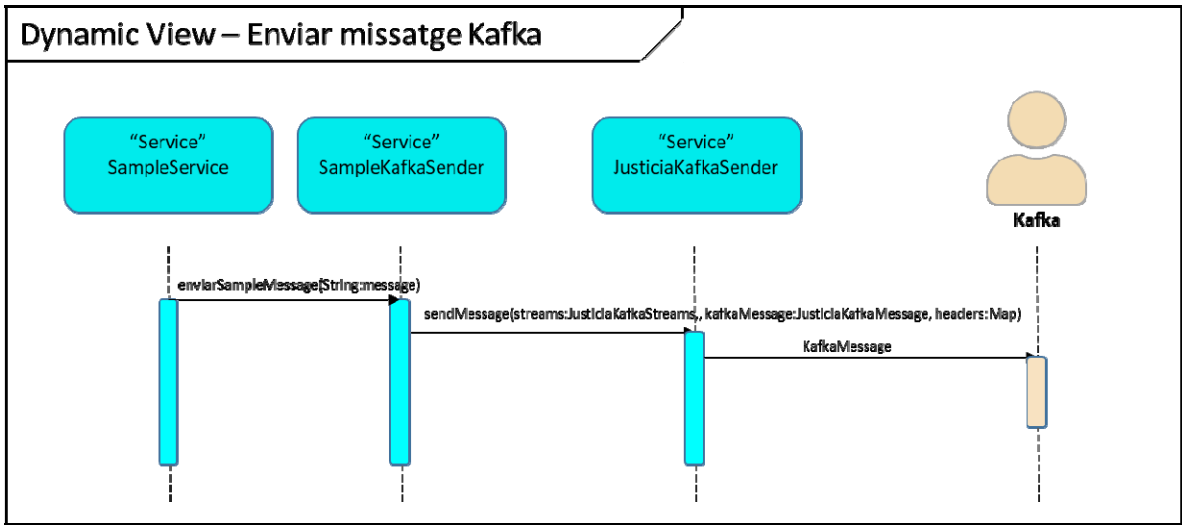
- Asíncrona: els serveis poden utilitzar els T-Components (JusticiaKafkaSender) de la llibreria jus-canigo34-cloud-lib per enviar missatges cap a Kafka.
- Síncrona; els protocol de comunicacions soportat pels serveis intenrs es el REST per això hem de distingir:
    - Servei extern utilitza protocol REST: els serveis poden executar directament aquest servei.
    - Servei extern que no utilitza protocol REST: a aquest cas, s'ha d'utilitzar un mecanisme de traducció que sigui responsable de rebre la crida REST que genera el servei intern (Integration Service), traduir de protocol REST al protocol del servei destí i de realitzar l'execució al protocol del servei destí. La implementació d'aquest servei d'integració serà responsabilitat del servei que té la necessitat de realitzar l'execució remota.

5.5.3 Vista dinàmica

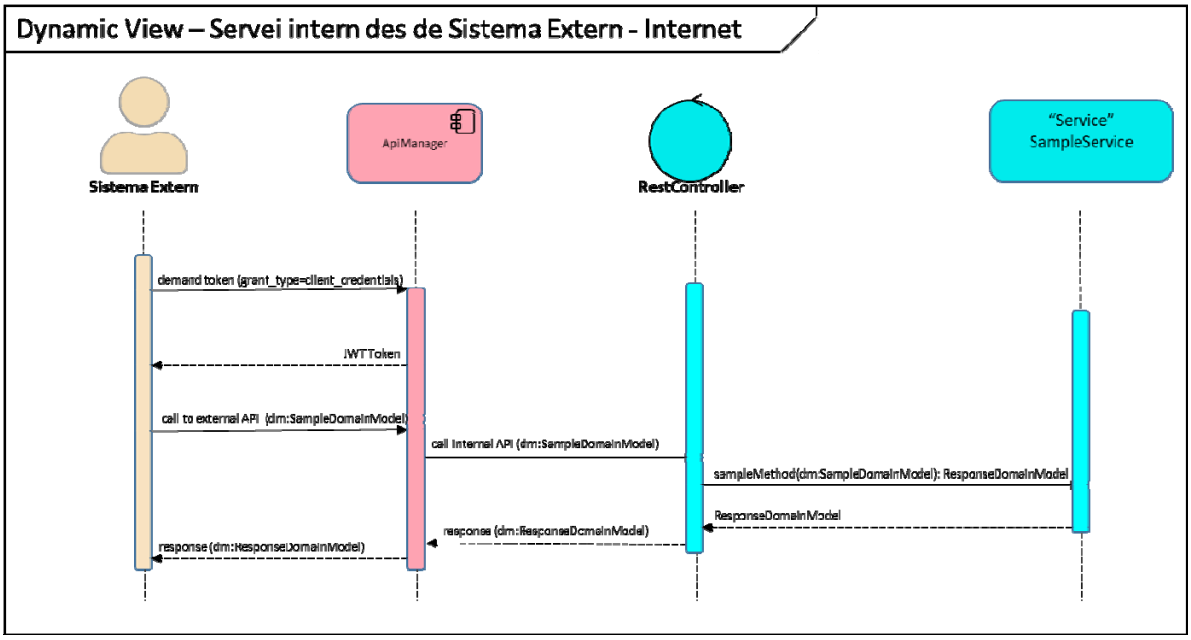
A tots els casos, el client o el destí de la comunicació a la capa d'integració no serà la capa de presentació, sinó un sistema extern o eina d'intercanvi de missatges.



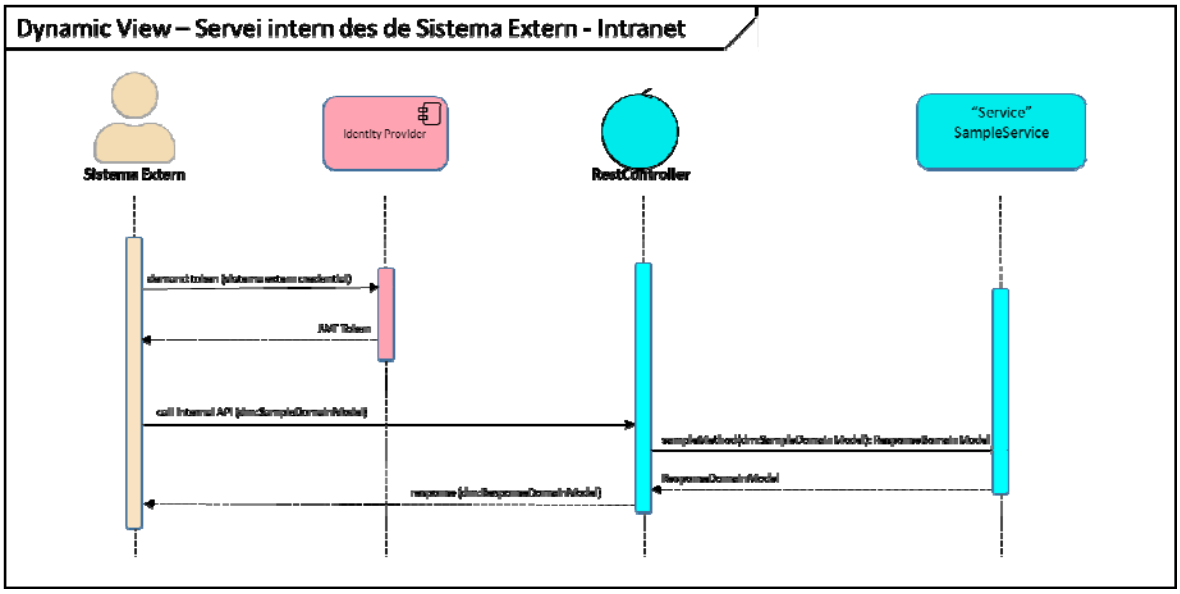
Quan el sistema rep un missatge des de Kafka i existeix un KafkaListener esperant la seva arribada, el missatge es processat pel T-Component JusticiaKafkaListener i enviat cap a SampleKafkaListener que s'encarrega de realitzar les tasques de negoci necessàries en relació al messatge rebut.



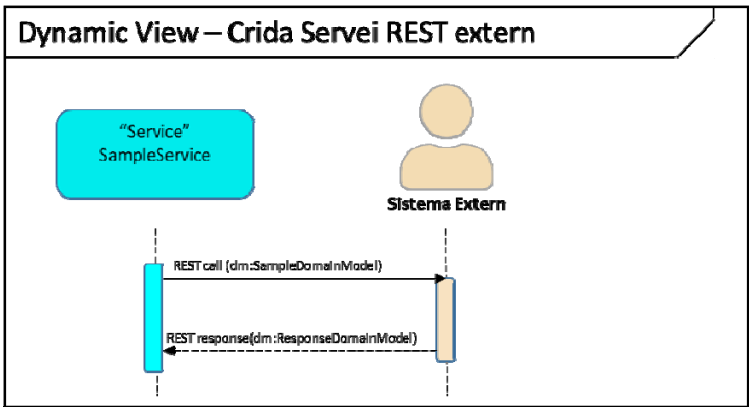
Quan des de la lògica de negoci apareix la necessitat de realitzar una comunicació asíncrona mitjançant un missatge Kafka, el servei de negoci delega aquesta tasca cap al servei d'enviament de missatges a Kafka que ho realitza utilitzant les funcionalitats del T-Component JusticiaKafkaSender.



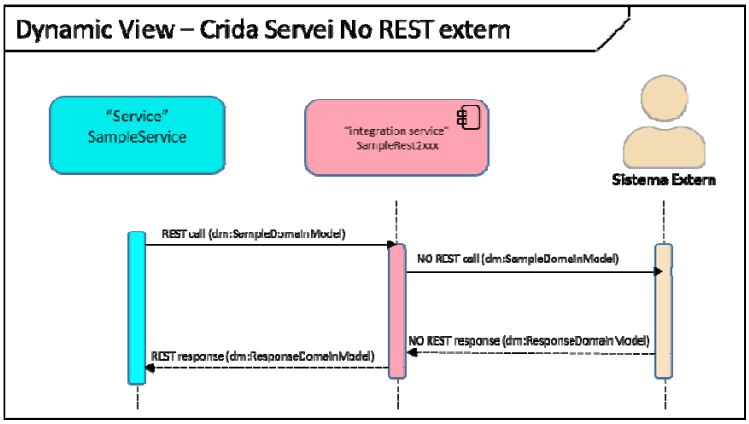
Quan un sistema extern necessita consumir un servei intern des d'Internet, primer de tot ha d'obtenir un token JWT que ho identifiqui i després ha de fer la crida mitjançant l'API exposada pel API Manager. L'API Manager realitza la subsegüent crida cap al Controler que gestiona l'API. El controller gestiona aquesta crida i executa la lògica de negoci que sigui adient i torna la resposta cap a l'API Manager. Per últim, l'API Manager retorna la resposta cap al sistema extern.



Quan un sistema extern necessita consumir un servei intern des de l'Intranet, primer de tot ha d'obtenir un token JWT que ho identifiqui de l'Identity Provider i després ha de fer la crida cap al Controler que gestiona l'API. El controller gestiona aquesta crida i executa la lògica de negoci que sigui adient i torna la resposta cap al sistema extern.



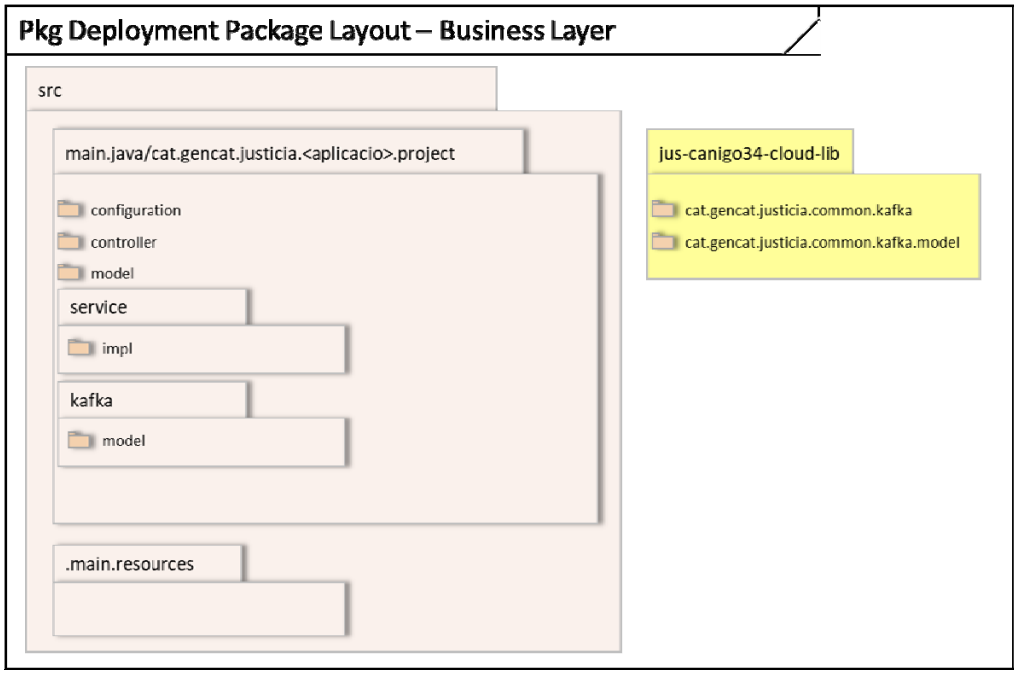
Per realitzar crides cap a serveis REST de sistemes externs, la execució es realitza directament des de un servei de negoci encarregat d'aquesta tasca.



Si el servei extern al que s'ha de cridar no utilitza el protocol REST, s'ha d'utilitzar un servei d'integració al que es cridat pel servei de negoci utilitzant el protocol REST de manera que el servei d'integració es responabilitza de fer les traduccions necessàries i la crida cap al servei del sistema extern amb el seu protocol. Quan rebre resposta, el servei d'integració s'encarrega de que el servei de negoci rebi la resposta amb protocol REST.

5.5.4 Vista d'implementació

A continuació es presenta un diagrama de la vista d'implementació enfocat només a la capa d'integració.



Al package **kafka** s'ubicaràn les classes específiques per les comunicacions amb Kafka. Per la resta de comunicacions els controllers aniran al package **controller** i els services al package **service**.

6 DECISIONS D'ARQUITECTURA

6.1 DECISIONS ARQUITECTÒNIQUES

6.1.1 Plataforma de contenidors

Al igual que els sistemes de virtualització tradicionals, els contenidors requereixen d'un sistema de gestió sobre el que executar els elements virtualitats. Com plataforma gestora dels contenidors, es va escollir Red Hat® OpenShift® és una plataforma de contenidors Kubernetes empresarial que ofereix el CPD corporatiu de referència pel departament al moment que es va definir aquesta arquitectura i per tant la que s'ha d'utilitzar per part dels desenvolupaments. Kubernetes es pot considerar el gestor de contenidors estàndard de facto a la industria al mateix moment.

6.1.2 Dades compartits entre serveis vs Dades propetat d'un únic servei

Un dels principals objectius de l'arquitectura és reduir l'acoblament entre els serveis.

El fet de tenir dades compartits a la base de dades implica diferents parts del sistema tenen accés directe a la informació compartida. També significa que canvis al model de les dades compartides podem implicar afectacions a les parts del sistema que accedeixen a aquests dades.

Per reduir l'acoblament a nivell de dades entre els diferents serveis, a aquesta arquitectura les dades seran propietàries d'un únic servei i es aquest servei el responsable del manteniment d'aquestes dades. Si un altre servei necessita aquelles dades les tindrà que demanar al servei propietari.

Aquesta decisió te implicacions importants fins al punt que el disseny funcional ha de dividir el negoci de manera que els dominis de negoci de cada servei puguin estar desacoblats i de manera que el disseny ha de evitar que una dada de negoci sigui propietària de més d'un servei. Una metodologia que facilita aquest tipus de disseny és el Disseny Orientat al Domini o DDD (https://www.domaindrivendesign.org/).

Si cal compartir dades entre serveis o creuar dades existirà una BBDD de consulta on es podran crear col·leccions amb dades de diferents serveis o on podran coexistir dades de diferents col·leccions per tal de poder fer creuament de dades entre diferents negocis. Aquesta decisió es detalla més endavant.

6.1.3 Base de dades Relacional Oracle vs Documental MongoDB

Al moment de dissenyar aquesta arquitectura els principals sistemes del departament utilitzen una base de dades Relacional Oracle RAC.

Els clients d'aquesta arquitectura són aplicacions basades amb Javascript que utilitzen el framework Angular 9. Aquests clients utilitzen models de dades en format JSON.

Com eina per tractar de reduir l'acoblament a nivell de base de dades, es va seleccionar una base de dades Documental. Va ser escollida pels desenvolupaments MongoDB 4.2. És una base de dades Documental i la seva elecció implica canviar la manera de fer els dissenys dels models de dades dels serveis.

Un des motius pel que es va seleccionar a partir de la versió 4.0 MondoDB va començar a donar suport a transaccions multi document. Aquest tipus de transaccions a les que múltiples documents d'una o diverses col·leccions de documents s'han de poder actualitzar de manera única (es a dir, s'ha de garantir que o es realitzen totes les modificacions de la transacció o no es realitza cap d'elles).

Un altre motiu es que es va evolucionar Canigó 3.4 de manera que es van incloure els drivers de MongoDB compatibles amb la versió 4.2 i que habiliten l'ús de les capacitats de transaccionalitat sobre múltiples documents a una única transacció.

MongoDB es una base de dades Documental a la que no existeix un esquema predefinit per determinar el format dels documents d'una agrupació de documents (anomenada col·lecció). Aquesta capacitat per tenir múltiples estructures als documents permet realitzar canvis d'estructura de manera progressiva en lloc de tenir que fer els canvis de manera massiva com s'ha de fer a les bases de dades relacionals. Els documents s'emmagatzemen en format BSON.

A diferencia de les bases de dades relacionals, a MongoDB les relacions i les regles de consistència de les dades a la base de dades no són gestionades directament pel motor de la base de dades. Això dona més flexibilitat al moment de realitzar modificacions a la base de dades. Per un altre costat, l'esforç de garantir la consistència de les dades recau al disseny i la implementació dels canvis de les dades sense suport per part de MongoDB.

Encara que la utilització de BBDD MongoDB és prioritària per als nous serveis desenvolupats, si el negoci d'un nou servei es considera crític aquest podria fer servir BBDD Oracle i no MongoDB. Aquesta decisió s'haurà de prendre en les fases inicials del projecte i caldrà ser validada amb els responsables de la solució al Departament.

6.1.4 BBDD de consulta

La BBDD de consulta serà una BBDD MongoDB consultable pels serveis que ho requereixin on hi haurà dades de les diferents BBDD dels serveis. Aquestes dades poden estar normalitzades o replicades i les podrem trobar com a col·leccions independents amb estructures iguals o similars que a les bases de dades dels serveis, o bé les podem trobar amb altres estructures. Podrem trobar dades desnormalitzades de diferents col·leccions agrupades en una, o també grans col·leccions que permetin obtenir d'una tacada conjunts de dades de diferents serveis en una sola consulta.

Aquesta BBDD permetrà realitzar consultes creuant diferents negocis.

6.1.5 API Manager

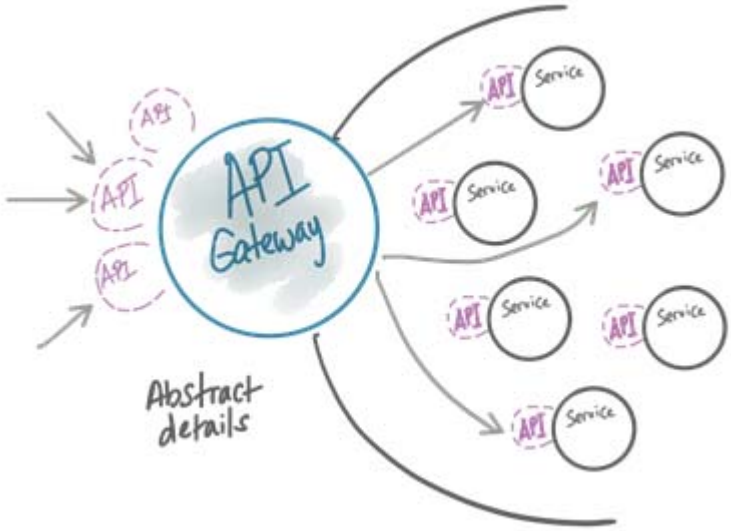
Aquesta decisió de disseny està relacionada amb els casos d'ús CU_ARQ001, CU_ARQ002, CU_ARQ003 i CU_ARQ011 i CU_ARQ029

Es va detectar la necessitat de un component que doni resposta a les següents responsabilitats:

- Ser la porta d'entrada per crides de sistemes externs que necessitin executar serveis oferts des de les aplicacions de la present arquitectura.
- Proveir d'una capa externa de seguretat
- Realitzar enrutament cap a sistemes interns

La decisió final ha estat fer ús del Api Manager Corporatiu IBM Api Connect

- Gestionat de manera transversal per una Oficina
- Contacte directe amb els seus responsables i equip de manteniment
- Totalment alineat amb normativa CTTI i els seus estàndards de Qualitat i Seguretat



6.1.6 Service Mesh

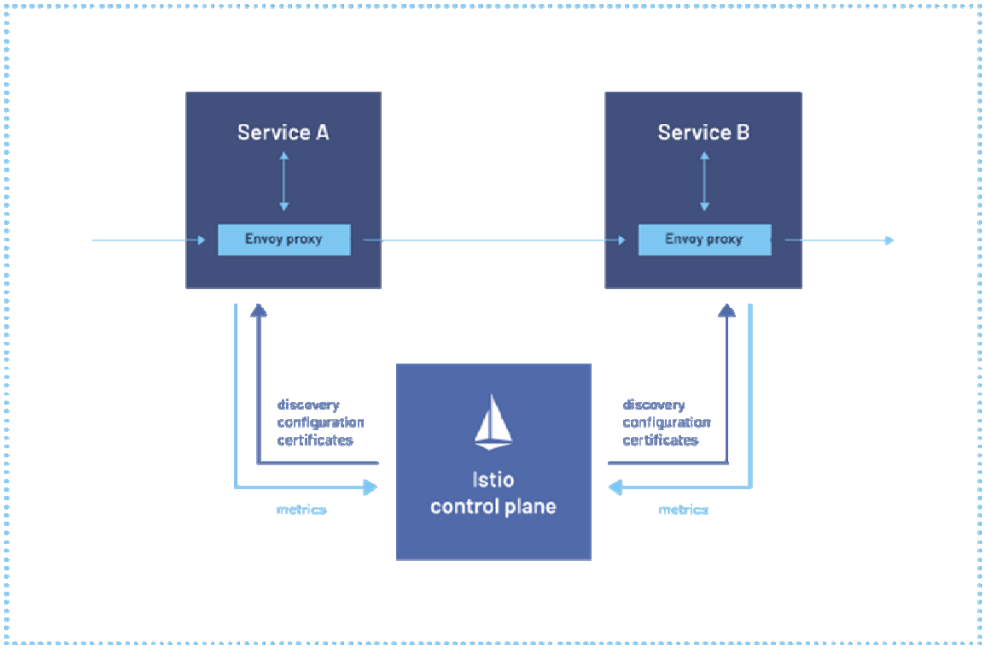
Aquesta decisió de disseny està relacionada amb el cas d'ús CU_ARQ011 Invocació a altres serveis síncrons

Els diferents serveis de la arquitectura tenen necessitat d'intercanviar informació entre ells i amb tercers. A arquitectures tradicionals, es responsabilitat dels propis serveis la gestió dels fluxos de tràfic entre serveis, el control d'accés i la recollida de traces de l'execució del intercanvi de la informació. Existeixen productes que habiliten la possibilitat d'externatlitzar dels serveis, en major o menor mesura, alguna o totes aquestes resposablitats.

La plataforma de virtualització de contenidors que disposa el departament de Justícia es un Openshift de RedHat i per tant, per raons de compatibilitat amb la plataforma i per tenir suport del fabricant, s'escull utilitzar el Service Mesh d'Openshift per oferir respostes a aquestes necessitats.

El Service Mesh d'Openshift està basat sobre el projecte open source Istio. Istio permet realitzar aquestes tasques sense modificar les aplicacions directament, amb un llenguatge de configuració propi i es compatible amb Kubernetes.

Istio utilitza Envoy Proxies per realitzar la instrumentalització de les comunicacions. Es poden utilitzar Virtual Services per realitzar tasques sobre la informació obtinguda al proxy.



6.2 AVALUACIÓ DE TECNOLOGIES

6.2.1 Implementació de serveis RESTful

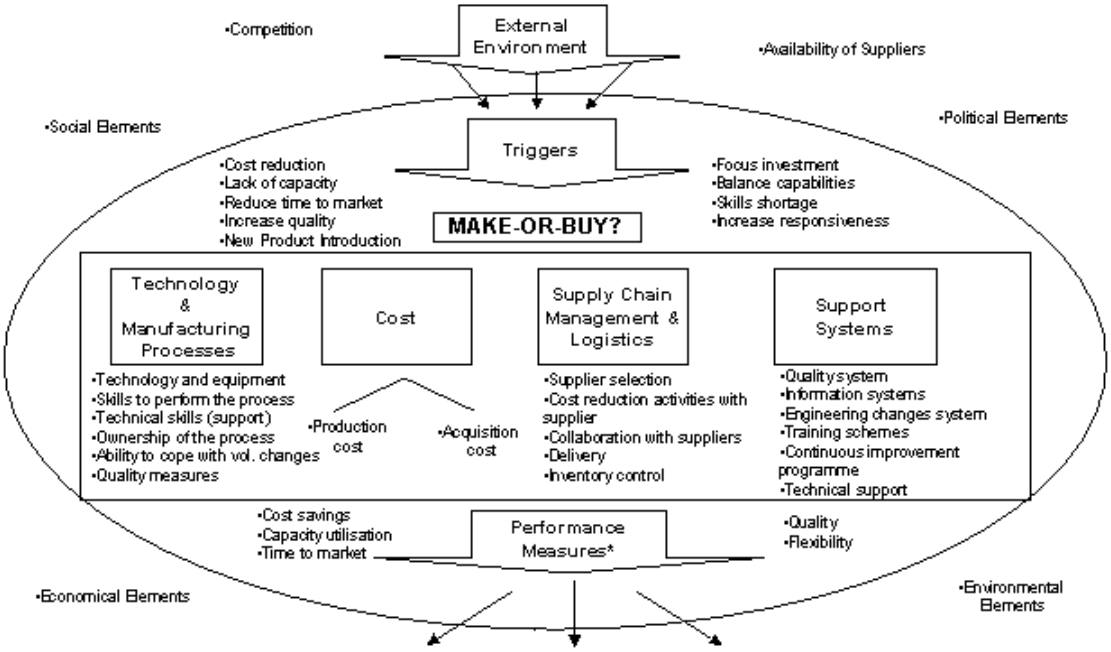
Per la implementació de la capa REST s'han considerat tres alternatives pels projectes de Justícia.

- Apache Camel (http://camel.apache.org)
- JAX-RS "Jersey" (https://jersey.java.net)
- Spring (https://spring.io/guides/gs/rest-service)

Finalment, s'ha escollit l'opció d'Spring, per la seva compatibilitat amb els frameworks Canigó 3.4 i Swagger 2.

6.3 DECISIONS SOBRE COMPRA / DESENVOLUPAMENT / REUTILITZACIÓ

El criteri general que es seguirà al projecte durant les avaluacions sobre noves necessitats tecnològiques es basarà en el criteri de decisions MBR (Make-Buy-Reuse). La decisió final sempre estarà consensuada entre Arquitectura i el Departament de Justícia.



6.4 PUNTS PENDENTS

7 OPERACIÓ, ROLLOUT I GESTIÓ APLICACIÓ

En aquest capítol es consideren els aspectes de desplegament i posada en marxa (rollout), operació i gestió de l'aplicació en termes dels seus efectes sobre l'arquitectura i a l'inrevés.

7.1 ROLLOUT

Cada aplicació haurà de desplegar els artefactes descrits a 5.1.2. Vista de desplegament, tant a entorns de desenvolupament, com a entorns client.

Caldrà seguir els procediments de pujada de codi i desplegaments a entorns client definides pel SIC (Servei d'Integració Contínua https://canigo.ctti.gencat.cat/sic/)
El proveïdor també ha de proporcionar al Departament els següents documents de la fase de rollout (plantilles MQS):

- Manual d'Instal·lació (en cas que tingui particularitats no esmentades al Manual d'Instal·lació general del sistema d'informació)
- Manual d'Explotació (en cas que tingui particularitats no esmentades al Manual d'Explotació general del sistema d'informació)

7.2 OPERACIÓ

Consultar al Portal MQS els procediments i plantilles relacionades amb l'operació del CPD (principalment, Manual d'Explotació):

- Manual d'Explotació (MEX): només cal realitzar un d'específic per a cada servei si té particularitats no recollides en el document general de Manual Explotació del sistema d'informació. Hauria de contenir els següents elements:
  - Política de backups
  - Paràmetres de monitorització (quins servidors, quins elements)
  - Ubicació dels fitxers de log que es generen
  - Llistat i descripció dels processos planificats existents al sistema
  - Processos de manteniment (esborrats periòdics de fitxers temporals, etc.)
  - Procediment de validació de desplegament
  - Seqüències d'aturada i arrancada del sistema

7.3 GESTIÓ DE L'APLICACIÓ

La gestió de l'aplicació es refereix a totes aquelles tasques que son necessàries pel manteniment, hotline, correcció d'incidències, i realització de futurs evolutius del sistema.

- Qualsevol incidència productiva arribarà via l'eina de ticketing Remedy, per tal que l'equip desenvolupador pugui atendre-la, i quedi registrada la posterior evolució
- Cal respectar el procediment de versionat de releases en entorns client, segons l'especificat a MQS https://qualitat.solucions.gencat.cat/estandards/estandard-versions-programari/

8 APÈNDIX

8.1 DOCUMENTACIÓ DE REFERÈNCIA

- CU_ARQ001_ARQ002_ARQ003_ARQ011_Seguretat

Guia dels casos d'ús relacionats amb la seguretat

- CU_ARQ004 - Accés a capa de distribució REST (extensió cloud)
- CU_ARQ004.1 - Accés a capa de distribució REST (guia base)

Guies dels casos d'ús d'accés a la capa de distribució REST

- CU_ARQ005_Canvi_de_Context_(backend)

Guia de canvi de context al backend.

- CU_ARQ006_Swagger

Guia d'ús de Swagger.

- CU_ARQ009 - Accés a capa de negoci

Guia d'accés a la capa de negoci

- CU_ARQ010 - Accés a BBDD

Guia d'accés a base de dades

- CU_ARQ010.1 - Desnormalització i modelatge del model de dades

Guia de desnormalització i modelatge del model de dades

- CU_ARQ011_Invocació_a_altres_serveis_síncrons

Guia d'invocació a altres serveis síncrons

- CU_ARQ012_Interacció_cap_a_sistemes_externs_(capa_negoci)

Guia d'interacció cap a sistemes externs des de la capa de negoci.

- CU_ARQ013 Cache

Guia d'ús de cache

- CU_ARQ014_Editor_de_documents

Guia del editor de documents

- CU_ARQ015_Signatura_Electrònica

Guia de la signatura electrònica

- CU_ARQ016 Procés Asíncron
- CU_ARQ016.1 Accés a eines de comunicació asíncrona

Guies d'eines de comunicació asíncrona.

- CU_ARQ017 Reporting

Guia de reporting

- CU_ARQ018 Monitorització

Guia de monitorització

- CU_ARQ019_Enviament_de_correus

Guia d'enviament de correus

- CU_ARQ020_Logging_distribuit

Guia de logging distribuït

- CU_ARQ021 Auditoria

Guia d'auditoria

- CU_ARQ022_Gestió_d_excepcions

Guia de gestió d'excepcions

- CU_ARQ023_Actualizar_configuracions_en_calent

Guia d'actualització de configuracions

- CU_ARQ024_Notificacions_PWA_backend

Guia de notificacions PWA part backend.

- CU_ARQ025_Transaccionalitat_i_SAGA

Guia de transaccionalitat i SAGA

- CU_ARQ026 Accés a serveis negoci des de sistemes externs

Guia d'accés a serveis des de sistemes externs.

- CU_ARQ027 Processos Batch

Guia de processament batch

- CU_ARQ028 ARQ Integració Capa Dades

Guia d'integració a capa de dades

- CU_ARQ029 Swagger API Manager

Guia de configuració de Swagger API a API Manager.

- CU_ARQ030 (ANNEX ATEC) - Guia navegació tècnica

Guia de navegació tècnica ATEC

- CU_ARQ030-31 - Interfície gràfica

Guia de casos d'ús de la interfície gràfica

- CU_ARQ034 - Canvi de context

Guia de canvi de context a frontend

- CU_ARQ035 - Monitorització capa client

Guia de monitorització a capa client.

- CU_ARQ036 – Microfrontends

Guia de microfrontends

- CU_ARQ037 – PWA

Guia de PWA a capa client.

- CU_ARQ_32-33 - Capa de presentació (general)

Guia general de la capa de presentació.

8.2 GLOSSARI DE TERMES

- A-Component: components no lligats a la tecnologia. La lògica funcional de l'aplicació és encapsulada aquí. Els típics elements podrien ser components per a la gestió de tràmits judicials, etc.
- T-Component: components d'arquitectura que utilitzen la infraestructura tècnica per tal de proveir serveis que son requerits per l'aplicació. No contenen elements específics del domini funcional de l'aplicació. Els típics elements podrien ser componenets per a SSO, Autenticació, Autorització, Accés a cues JMS, etc.
- DAO: Patró de disseny *Data Access Object*
- DTO: Patró de disseny *Data Transfer Object*
- JMS: Java Message Service
- JSON: JavaScript Object Notation
- JWT: JSON Web Tokens
- OSB: Oracle Service Bus
- POJO: Plain Old Java Object
- POJI: Plain Old Java Interface.
- QA: Quality Assurance
- REST: Representational State Transfer
- SSO: Single Sign-On
- XML: Extensible Markup Language