ListToPlan: Desenvolupament d'una aplicació web d'organització personal i de grups escalable i flexible seguint el patró MVC

José Jiménez López
Màster en Enginyeria Informàtica
Desenvolupament d'aplicacions web

Ignasi Lorente Puchades
César Pablo Córcoles Briongos

11/06/2018

FITXA DEL TREBALL FINAL

Títol del treball:	*ListToPlan: Desenvolupament d'una aplicació web d'organització personal i de grups escalable i flexible seguint el patró MVC*
Nom de l'autor:	*José Jiménez López*
Nom del consultor/a:	*Ignasi Lorente Puchades*
Nom del PRA:	*César Pablo Córcoles Briongos*
Data de lliurament (mm/aaaa):	*06/2018*
Titulació o programa:	*Màster universitari en Enginyeria Informàtica*
Àrea del Treball Final:	*Desenvolupament d'aplicacions web*
Idioma del treball:	*Català*
Paraules clau	*Arquitectura web, API REST, MVC*

Resum del Treball (màxim 250 paraules): *Amb la finalitat, context d'aplicació, metodologia, resultats i conclusions del treball*

La societat avança cap a un món cada cop més connectat on la tecnologia és cada cop més el nucli de molts processos socials i empresarials. La tecnologia avança ràpidament i cada dia sorgeixen noves arquitectures i paradigmes a l'hora de desenvolupar noves aplicacions.

L'objectiu d'aquest treball és el desenvolupament d'una aplicació que segueixi el patró Model-Vista-Controlador per tal que sigui flexible i mantenible. A més es proposa una arquitectura de tres capes, totalment diferenciades i desacoblades que permeten que l'aplicació sigui escalable tant verticalment com horitzontalment.

L'aplicació que es desenvoluparà en aquest projecte és una aplicació amb un fort enfocament social que permet tant el treball col·laboratiu com la compartició de continguts. L'aplicació es basa en dos ítems principals: les notes i les llistes. Els usuaris poden crear diferents grups on crear i desenvolupar el contingut de ambdós tipus d'ítems. A més, permet compartir les llistes de cada usuari o grup, de manera que poden ser fetes servir per la resta d'usuaris per tal de poder organitzar events que ja han sigut organitzats pels usuaris que comparteixen la llista.

Aquest projecte no es centra únicament en el desenvolupament de les funcionalitats pròpies de l'aplicació sinó que fa molt d'èmfasi tant en la tecnologia emprada com en l'arquitectura de l'aplicació, així com de totes les fases del procés de desenvolupament de programari: planificació, disseny, desenvolupament, proves i identificació de futures millores.

Abstract (in English, 250 words or less):

Society is moving towards a world which increasingly more connected where technology is increasingly the core of many social and business processes. Technology is rapidly advancing and every day new architectures and paradigms emerge when developing new applications.

The aim of this work is the development of an application that follows the pattern of Model-View-Controller in order to be flexible and maintainable. In addition, it proposes a three layered architecture, which are completely differentiated and decoupled, which allows the application to be scalable both vertically and horizontally.

The application that will be developed in this project is an application with a strong social focus that allows both collaborative work and sharing contents. The application is based in two principal items: Notes and Lists. The Users can create different groups where they can create and develop contents of both types of items. In addition, it also allows sharing the lists from the users or groups, so they can be used by other users to organize events that have already been organizes by the users, who share the list.

This project focuses not only on the development of the application's own functionalities, but rather emphasizes both on the technology used and the application's architecture, as well as all the phases from the process of software's development: planing, design, development, testing and identification of future improvements.

Índex

1. Introducció 1
   1.1. Context i justificació del Treball 1
   1.2. Objectius del Treball 2
      1.2.1. Objectius funcionals: Funcionalitats de l'aplicació 2
      1.2.2. Objectius tècnics: Arquitectura, disseny i tecnologies 3
   1.3. Enfocament i mètode seguit 4
   1.4. Planificació del Treball 4
   1.5. Breu sumari de productes obtinguts 6
   1.6. Breu descripció dels altres capítols de la memòria 7
2. Disseny de l'aplicació 8
   2.1. Diagrames UML 8
      2.1.1. Diagrama de casos d'ús 8
      2.1.2. Diagrama de classes 9
      2.1.3. Diagrama entitat-relació de base de dades 10
      2.1.4. Diagrames d'activitat 12
         2.1.4.1. Usuaris 12
         2.1.4.2. Grups 12
         2.1.4.3. Notes 13
         2.1.4.4. Llistes 13
   2.2. Disseny de la interfície d'usuari i usabilitat 14
      2.2.1. Perfil d'usuaris 14
      2.2.2. Histories d'usuari 15
      2.2.3. Prototip de baixa definició (wireframe) 18
      2.2.4. Prototip d'alta definició 21
   2.3. Arquitectura de l'aplicació 26
      2.3.1. Arquitectura global del sistema 26
      2.3.2. Arquitectura capa de persistència de dades 27
      2.3.3. Arquitectura Servidor 28

2.3.4. Arquitectura Client 29
2.3.5. Arquitectura aplicació distribuïda 30
3. Desenvolupament de l'aplicació 31
3.1. Extractes de codi 31
3.1.1. Extractes de codi de base de dades 31
3.1.2. Extractes de codi del back-end 35
3.1.3. Extractes de codi del front-end 43
3.2. Seguretat 50
3.3. Tests 52
3.3.1. Definició dels casos de proba 52
3.3.2. Resultats dels casos de prova 57
3.4. Versionat i control de versions 58
3.5. Bugs i millores 59
3.6. Pressupost 60
4. Conclusions 61
5. Glossari 63
6. Bibliografia 64
7. Annexos 65
7.1. Instruccions instal·lació de la Base de dades 65
7.2. Instruccions instal·lació del servidor *Back-end* 65
7.3. Instruccions instal·lació del servidor Front-end 66

Llista de figures

Imatge 1: Diagrama de Gantt de la planificació del projecte 5
Imatge 2: Detall de les tasques identificades en la planificació del projecte 5
Imatge 3: Diagrama de casos d'ús per a un usuari individual 8
Imatge 4: Diagrama de casos d'ús per un ús grupal de l'aplicació 9
Imatge 5: Diagrama de classes de les classes dels Models 9
Imatge 6: Diagrama de classes complet de l'aplicació 10
Imatge 7: Diagrama entitat-relació de base de dades 11
Imatge 8: Diagrama d'activitats per a la creació d'un usuari 12
Imatge 9: Diagrama d'activitats per a la gestió de grups 12
Imatge 10: Diagrama d'activitats per a la gestió de notes 13
Imatge 11: Diagrama d'activitats per a la gestió de llistes 13
Imatge 12: Wireframe pantalla login 18
Imatge 13: Wireframe pantalla dashboard principal 19
Imatge 14: Wireframe pantalla modal afegir/modificar nota 20
Imatge 15: Wireframe pantalla modal afegir/modificar llista 20
Imatge 16: Prototip pantalla de login 21
Imatge 17: Prototip pantalla de registre (modal) 22
Imatge 18: Prototip pantalla de visualització de notes 22
Imatge 19: Prototip pantalla de visualització de llistes 23
Imatge 20: Prototip pantalla creació i modificació de notes (modal) 23
Imatge 21: Prototip pantalla creació i modificació de llistes (modal) 24
Imatge 22: Prototip pantalla de creació i modificació de grups (modal) 24
Imatge 23: Prototip pantalla de cerca de llistes compartides 25
Imatge 24: Diagrama arquitectura general del sistema 26
Imatge 25: Diagrama arquitectura de Base de Dades 27
Imatge 26: Diagrama arquitectura del back-end 28
Imatge 27: Diagrama arquitectura del front-end 30
Imatge 28: Diagrama aplicació distribuïda 30

1. Introducció

1.1. Context i justificació del Treball

La aplicació que es vol desenvolupar en aquest projecte s'emmarca en el grup de les aplicacions d'organització personal. No obstant, en quant a funcionalitats, es busca diferenciar-se de la resta donant-li un enfoc més social i col·laboratiu. El que es vol aconseguir amb l'aplicació resultant, és una aplicació on l'usuari tingui centralitzada totes les seves gestions, tant a nivell personal com a nivell dels diferents grups al quals pugui pertànyer: amics, família, companys de feina... A més, l'usuari podrà compartir i accedir a altres elements compartits per altres usuaris per facilitar l'organització de determinats events que ja hagin sigut organitzats per altres usuaris de la plataforma, com per exemple, una llista de coses a fer per organitzar una festa d'aniversari, organitzar una mudança, tot el necessari per adoptar una mascota... L'usuari podrà crear notes, diferents tipus de llistes (ordenades, *checklists*, de repartició de tasques entre usuaris...), tractar-les a nivell individual o en grup i reutilitzar-les i compartir-les per tal que altres usuaris en puguin fer ús.

A part de les funcionalitats de la aplicació, durant el desenvolupament del projecte es vol fer molt d'èmfasi en l'arquitectura de l'aplicació. Es persegueix que l'aplicació faci servir tecnologies i metodologies modernes, que sigui escalable horitzontalment i que sigui flexible alhora d'afegir noves funcionalitats.

Actualment existeixen una sèrie d'aplicacions similars que cobreixen necessitats similars, com keep de google, notes de apple o evernote. Amb aquesta aplicació es busca fer una iteració més sobre aquest tipus d'aplicacions i donar-li un enfoc més social i de grups i compartició, fent èmfasi en els punts forts: accés descentralitzat i treball col·laboratiu.

La primera versió de l'aplicació coberta durant el desenvolupament del projecte implementarà les funcionalitats i solucions de disseny plantejades en els paràgrafs anteriors i que seran desenvolupades amb més detalls en els punts següents.

1.2. Objectius del Treball

Com s'ha introduït en l'apartat anterior, l'objectiu de l'aplicació és que els usuaris puguin fer servir l'aplicació per organitzar-se individualment o en grup, compartir llistes o notes amb la resta d'usuaris i poder treballar de manera col·laborativa. Per aconseguir això l'aplicació haurà de proporcionar una sèrie de funcionalitats que, juntament amb un disseny i una arquitectura adequada, conformaran els objectius del projecte.

1.2.1. Objectius funcionals: Funcionalitats de l'aplicació

Funcionalitats com a usuari individual:

- Poder crear i modificar notes
- Poder crear i modificar diferents tipus de llistes:
    - Llistes ordenades
    - *Checklists*
- Compartir llistes
- Crear, gestionar i afegir-se a grups

Funcionalitats de grups:

- Administrar els grups: modificar-lo, afegir i treure usuaris...
- Crear notes col·laboratives accessibles i modificables per tots els usuaris del grup
- Crear diferents tipus de llistes, accessibles i modificables pels usuaris del grup:
    - Llistes ordenades
    - *Checklists*
    - Llistes de repartició de tasques

Funcionalitats socials:

- Compartir llistes creades per l'usuari
- Accedir a llistes creades per altres usuaris i fer-les servir com a plantilla

1.2.2. Objectius tècnics: Arquitectura, disseny i tecnologies

Arquitectura de l'aplicació:

- L'aplicació estarà dividida en tres capes: front-end, back-end i capa de persistència
- Les tres capes estaran totalment desacoblades: el back-end serà una API REST que serà accessible de manera oberta pels usuaris que en vulguin fer us i es proporcionarà un frontal en AngularJS que en farà us d'aquesta API. Això permetrà que en el futur es pugui desenvolupar una aplicació mòbil que pugui fer ús d'aquesta interfície.
- Pel que fa a la informació bàsica de les sessions aquestes es gestionarà per mitjà de tokens (JWT, JSON Web Token). Quan l'usuari fa login amb èxit rep un token que conté la informació de la sessió juntament amb altra informació relativa al propi token. En cada petició que el client fa al servidor, ha d'afegir aquest token com a *header* de la petició. Finalment, el servidor valida la validesa i les dades que conté, si tot està bé retorna la resposta corresponent i, si es produeix un error en la validació, es retorna una resposta d'error amb codi HTTP 401: *Unauthorized*.

Disseny del frontal:

- La interfície gràfica serà responsiva (*responsive*) per tal que sigui visible i accessible des de el major número de dispositius possibles.
- La interfície serà usable i accessible i es seguirà un mètode de desenvolupament centrat en l'usuari (DCU).
- Per facilitar i agilitzar el desenvolupament del frontal, la interfície estarà bassada en una estructura predefinida i s'utilitzarà el *framework Bootstrap.*

Tecnologies emprades:

- Capa de persistència de dades: Base de dades MySQL per a les dades pròpies de l'aplicació.
- Capa de back-end: Java 8 juntament amb el framework *Spring Boot* per al desenvolupament de la API REST. L'aplicació correrà sobre un servidor d'aplicacions Tomcat.
- Capa de front-end: HTML i JavaScript amb els *frameworks* jQuery, AngularJS i Bootstrap.

1.3. Enfocament i mètode seguit

L'estratègia de desenvolupament es basa en crear una aplicació pròpia, des de cero. Es descarta doncs, basar-se en aplicacions *open source* i extendre-les, o integrar certes funcionalitats de l'aplicació amb altres sistemes que proporcionin una interfície d'aplicació (API). Amb això s'aconsegueix no haver de dependre de terceres persones, permisos, restriccions i permet alliberar-se de qualsevol rigidesa en el desenvolupament. A més es pot dissenyar una arquitectura pròpia fent servir les tecnologies més convenients per a la aplicació. Això no treu que algunes de les funcionalitats ofertes per l'aplicació ja siguin presents en d'altres i que irremeiablement serveixen d'inspiració a l'hora d'afegir-les al sistema.

La metodologia en quant a la planificació i les fases del projecte es tractaran en l'apartat següent.

1.4. Planificació del Treball

A l'hora d'escollir en la metodologia de desenvolupament s'han plantejat dos tipologies:

- Desenvolupament en cascada
- Desenvolupament iteratiu

A l'hora d'escollir s'ha tingut en compte la naturalesa i l'abast del projecte. Al tractar-se d'un projecte individual, amb un abast i uns terminis tancats, s'ha decidit fer servir la **metodologia en cascada**. Aquesta metodologia potencia les característiques amb les quals es desenvoluparà el projecte: desenvolupament del projecte de manera seqüencial degut a que només hi haurà un únic desenvolupador, extensió temporal ben definida i delimitada en el temps i abast, requeriments i funcionalitats majoritàriament tancades.

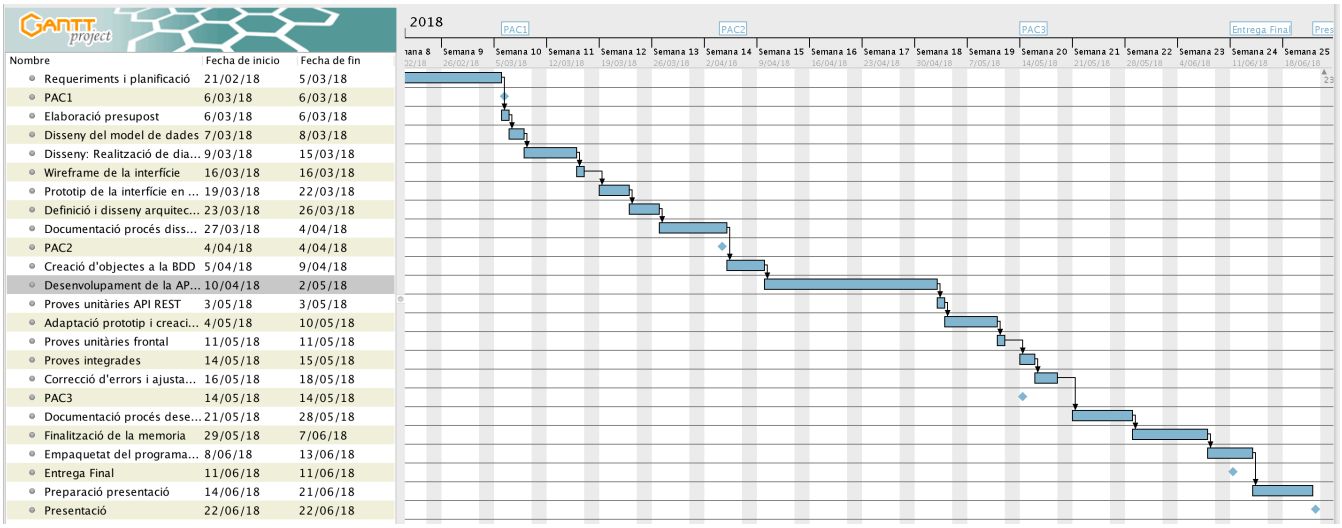
Les fases del projecte seran les següents [1]:

1. Anàlisi de requeriments
2. Disseny de l'aplicació
3. Desenvolupament
4. Proves
5. Implantació i instal·lació
6. Manteniment (serà definir però està fora de l'abast)

Finalment es definiran cinc fites temporals on es lliuraran diferents parts del treball realitzat:

- PAC 1: 06/03/2018
- PAC 2: 04/04/2018
- PAC 3: 13/05/2018
- Entrega final: 11/06/2018
- Presentació: 22/06/2018

Un cop definida la metodologia del projecte i les fases en les quals es dividirà la planificació de les diferents tasques i fites en el temps serà la següent:



Nombre	Fecha de inicio	Fecha de fin
Requeriments i planificació	21/02/18	5/03/18
PAC1	6/03/18	6/03/18
Elaboració presupost	6/03/18	6/03/18
Disseny del model de dades	7/03/18	8/03/18
Disseny: Realització de dia...	9/03/18	15/03/18
Wireframe de la interfície	16/03/18	16/03/18
Prototip de la interfície en ...	19/03/18	22/03/18
Definició i disseny arquitec...	23/03/18	26/03/18
Documentació procés diss...	27/03/18	4/04/18
PAC2	4/04/18	4/04/18
Creació d'objectes a la BDD	5/04/18	9/04/18
Desenvolupament de la AP...	10/04/18	2/05/18
Proves unitàries API REST	3/05/18	3/05/18
Adaptació prototip i creaci...	4/05/18	10/05/18
Proves unitàries frontal	11/05/18	11/05/18
Proves integrades	14/05/18	15/05/18
Correcció d'errors i ajusta...	16/05/18	18/05/18
PAC3	14/05/18	14/05/18
Documentació procés dese...	21/05/18	28/05/18
Finalització de la memoria	29/05/18	7/06/18
Empaquetat del programa...	8/06/18	13/06/18
Entrega Final	11/06/18	11/06/18
Preparació presentació	14/06/18	21/06/18
Presentació	22/06/18	22/06/18

Imatge 1: Diagrama de Gantt de la planificació del projecte

A continuació es mostra una imatge amb el detall de les tasques:

Nombre	Fecha de i...	Fecha de fin
Requeriments i planificació	21/02/18	5/03/18
PAC1	6/03/18	6/03/18
Elaboració presupost	6/03/18	6/03/18
Disseny del model de dades	7/03/18	8/03/18
Disseny: Realització de diagrames UML	9/03/18	15/03/18
Wireframe de la interfície	16/03/18	16/03/18
Prototip de la interfície en HTML	19/03/18	22/03/18
Definició i disseny arquitectura	23/03/18	26/03/18
Documentació procés disseny	27/03/18	4/04/18
PAC2	4/04/18	4/04/18
Creació d'objectes a la BDD	5/04/18	9/04/18
Desenvolupament de la API REST	10/04/18	2/05/18
Proves unitàries API REST	3/05/18	3/05/18
Adaptació prototip i creació frontal	4/05/18	10/05/18
Proves unitàries frontal	11/05/18	11/05/18
Proves integrades	14/05/18	15/05/18
Correcció d'errors i ajustaments	16/05/18	18/05/18
PAC3	14/05/18	14/05/18
Documentació procés desenvolupament	21/05/18	28/05/18
Finalització de la memoria	29/05/18	7/06/18
Empaquetat del programari i instruccions	8/06/18	13/06/18
Entrega Final	11/06/18	11/06/18
Preparació presentació	14/06/18	21/06/18
Presentació	22/06/18	22/06/18

Imatge 2: Detall de les tasques identificades en la planificació del projecte

1.5. Breu sumari de productes obtinguts

L'objectiu, un cop finalitzat el projecte, és aconseguir una sèries de productes o *outputs* que són en si mateixos els resultats del projecte. Aquests productes són:

- Una aplicació web: El producte principal serà el codi de la aplicació web, testejat, compilat i preparat per al seu desplegament. El codi englobarà les tres capes de de l'aplicació: Base de dades, *back-end* i *front-end*.
- Memòria del treball: És aquest mateix document, on figuren les motivacions, el disseny, l'arquitectura, els detalls més importants i les proves realitzades sobre l'aplicació.
- Presentació i vídeo de presentació: Es generarà una presentació amb diapositives i un vídeo que conté un resum funcional i tècnic de l'aplicació així com una presentació del funcionament de l'aplicació en un entorn local. L'objectiu és mostrar de manera visual i dinàmica els resultats dels projectes i les principals característiques de l'aplicació.

1.6. Breu descripció dels altres capítols de la memòria

El bloc principal de la memòria fa èmfasi en tot el procés de planificació i desenvolupament del projecte. La estructura del document és molt similar al procés de desenvolupament en cascada i presenta de manera diferenciada les diferents parts del procés de desenvolupament.

Un cop plantejada la planificació del projecte, el següent capítol tracta sobre el disseny funcional i tècnic així com els casos d'us de l'aplicació. Aquest disseny fa servir les eines estàndard que s'utilitzen en els processos de dissenys de *software:* Els dissenys funcionals segueixen la notació UML mentre que els cassos d'ús es recullen en forma d'històries d'usuari.
Posteriorment es persegueix fer una síntesi de tot el procés de desenvolupament del codi de l'aplicació fent un recull dels fragments més significatius de l'aplicació, que donen una idea global dels principals mecanismes que fa servir l'aplicació així com de la seva arquitectura.
Un cop explorat el procés de disseny, el següent pas és explorar la seguretat que implementa la aplicació i els tests que s'han realitzat. Per simplificar el procés i la visualització dels resultats, s'ha optat per plantejar els tests en forma de casos de prova, indicant les condicions prèvies, el resultat esperat i, finalment, recollir els resultats obtingut en la execució de cada cas.
Finalment, un cop finalitzat la fase de desenvolupament i proves, és fa un resum de les diferents versions per la qual ha passat la aplicació durant el procés de desenvolupament, així com les línies futures que han de seguir marcant el procés de millora i evolució de l'aplicació. Al final del document, es pot consultar les conclusions, on es fa una reflexió crítica sobre els resultats i el projecte, el glossari amb les definicions dels conceptes més utilitzats en la memòria, la bibliografia amb les principals fonts consultades i l'Annex on es recull les instruccions d'instal·lació i desplegament del programari per a cada capa de l'aplicació.

2. Disseny de l'aplicació

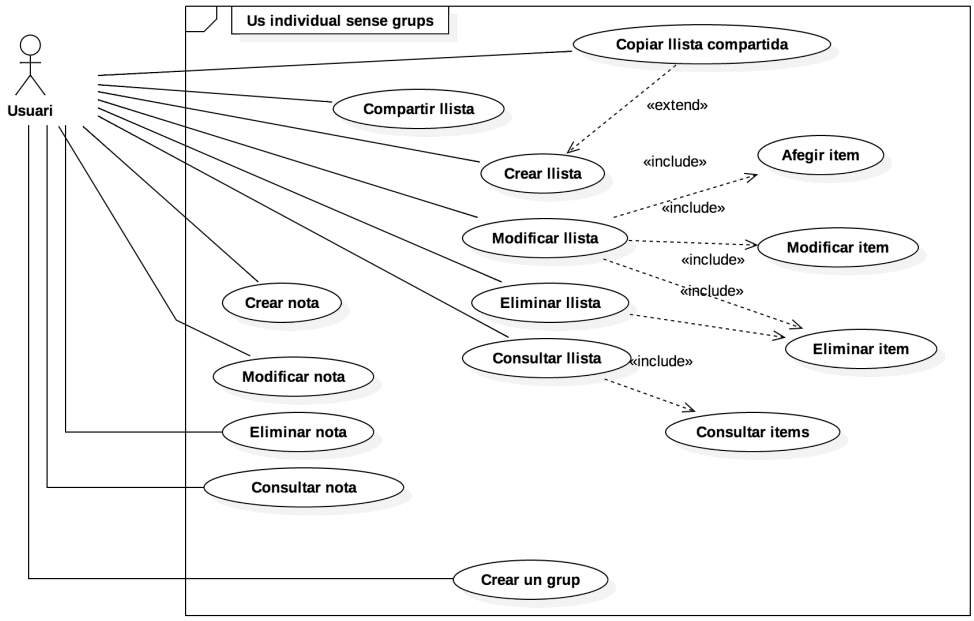
2.1. Diagrames UML

En el següent capítol s'exposa el disseny de l'aplicació des d'un punt de vista funcional per mitjà de diagrames UML [2]. Per a representar les funcionalitats, usos i rols de l'aplicació s'han desenvolupat els següents diagrames:

- Diagrama de casos d'ús
- Diagrama de classes
- Diagrama d'entitat-relació de base de dades
- Diagrames d'activitat

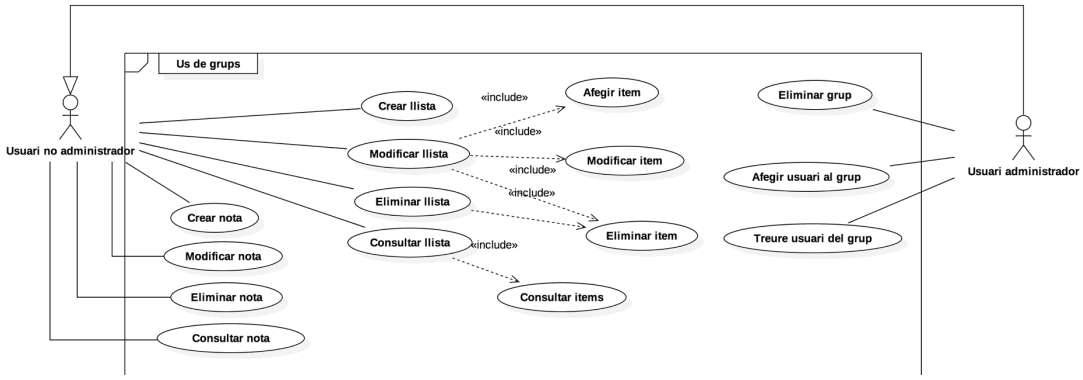
2.1.1. Diagrama de casos d'ús

Per a representar els casos d'ús l'aplicació, s'ha dividit l'aplicació en dos àmbits d'ús: ús de l'aplicació com a usuari individual i ús com a usuaris dins d'un grup.
El diagrama següent representa els casos d'ús quan l'usuari fa ús de l'aplicació com a usuari individual:



Imatge 3: Diagrama de casos d'ús per a un usuari individual

En el següent diagrama s'exposa el cas en que l'usuari fa servir l'aplicació dintre d'un grup d'usuaris:



Imatge 4: Diagrama de casos d'ús per un ús grupal de l'aplicació

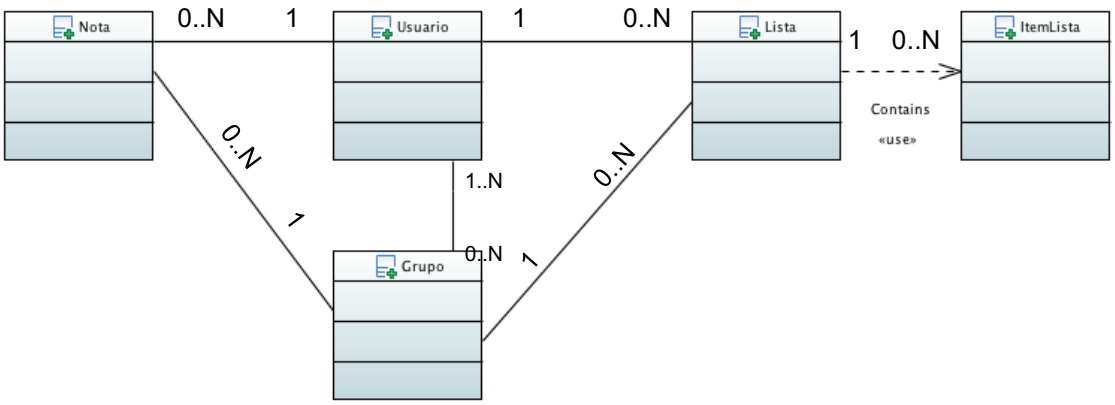
2.1.2. Diagrama de classes

La major part de la lògica de l'aplicació es troba al back-end de l'aplicació. Aquest back-end està programat en Java seguint un paradigma de programació orientat a objectes. El fet de estructurar el codi en classes i objectes fa que sigui molt més fàcil estructurar l'aplicació i que sigui possible realitzar una associació entre les classes i els models que representen.

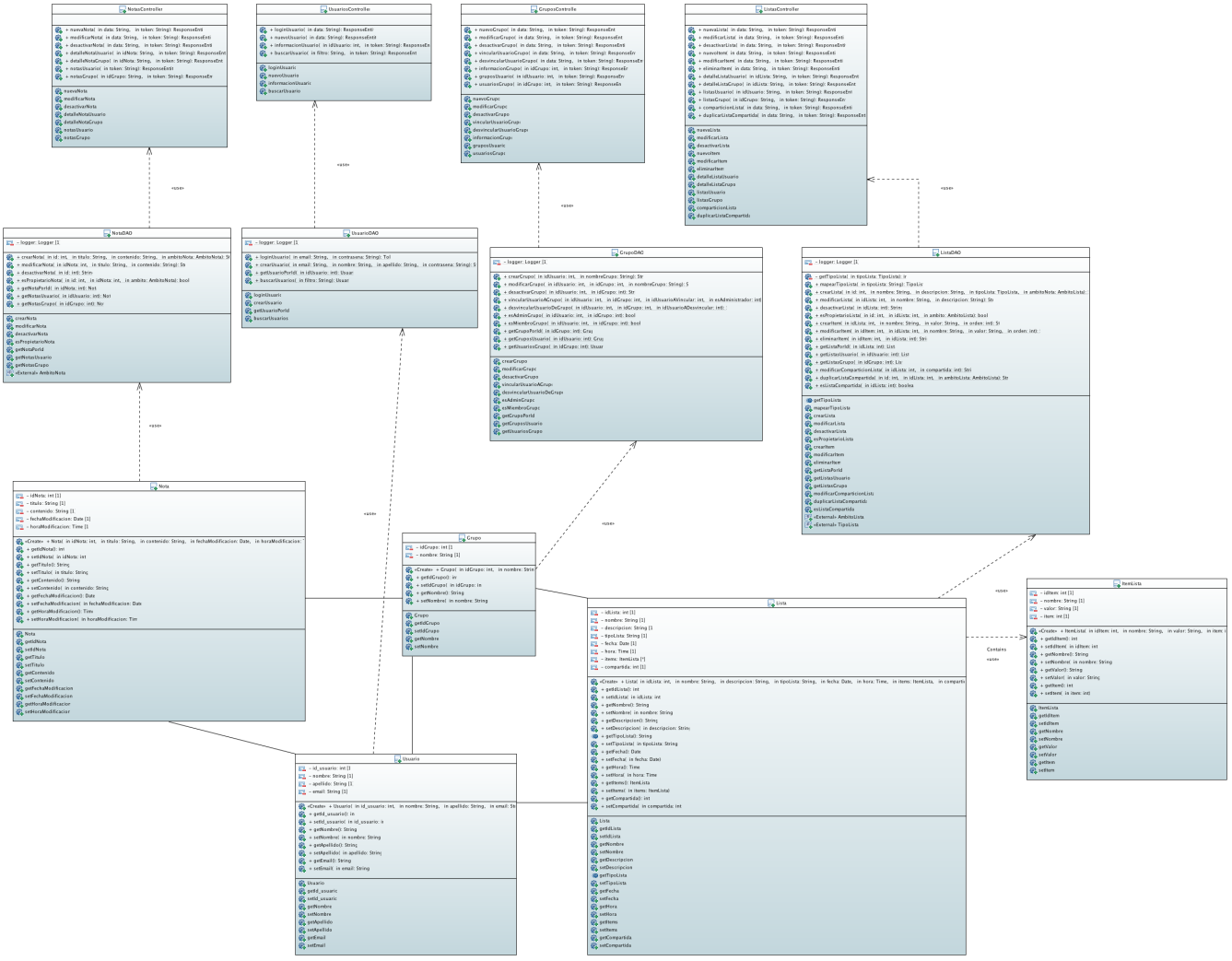
Convé especificar, que en el back-end es segueix el patró Model-Controlador. Al tractar-se d'una API REST no es gestionen ni generen Vistes, per tant, aquesta responsabilitat es transfereix al front-end.

Així doncs, les classes i les relacions entre aquestes que s'han fet servir en el back-end apareixen representades en el següent diagrama de classes.

Primer de tot, es mostra el detall de les relacions entre els diferents Models que es fan servir a la aplicació i que tenen relació directa amb les entitats de base dades:



Imatge 5: Diagrama de classes de les classes dels Models

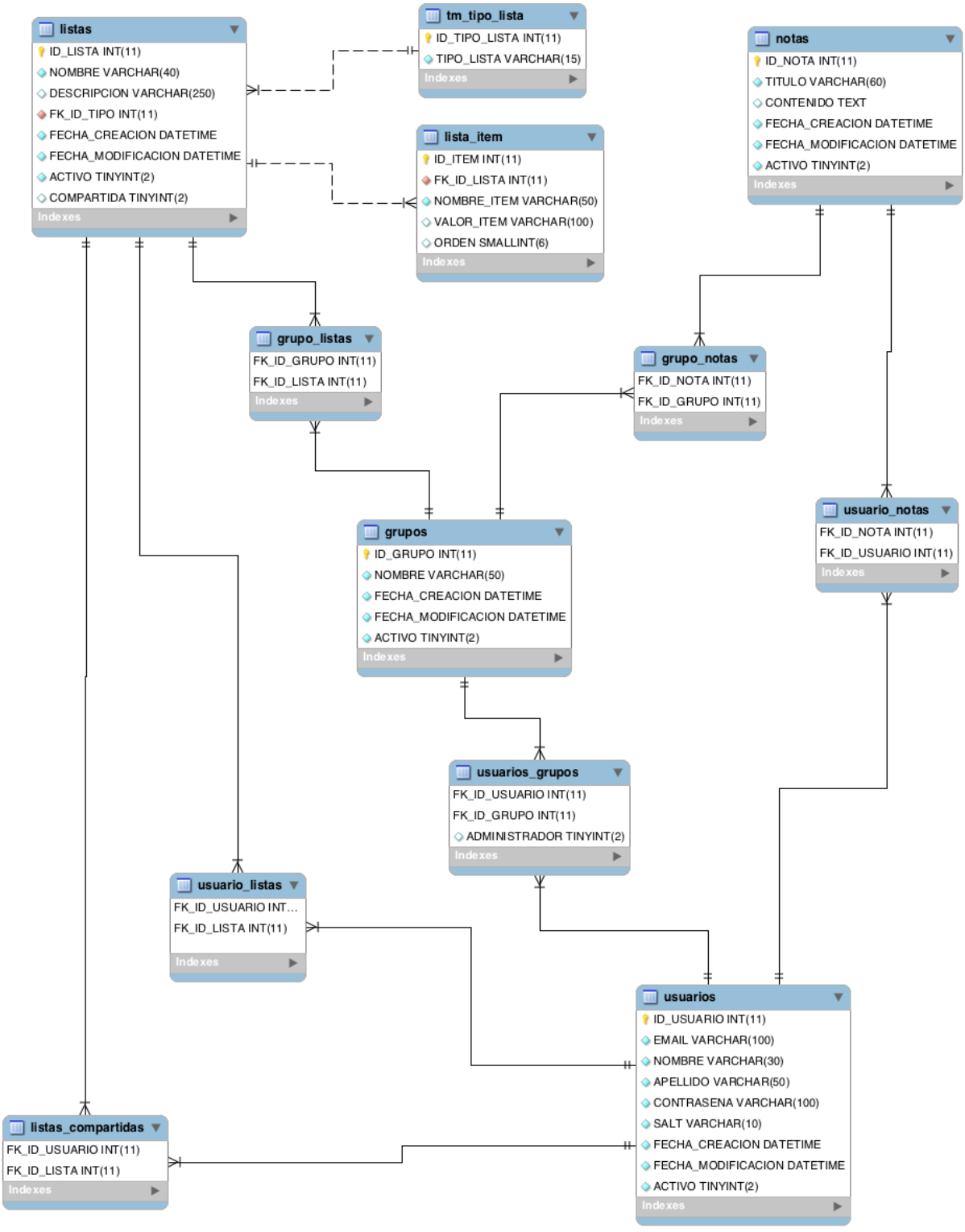


Imatge 6: Diagrama de classes complet de l'aplicació

2.1.3. Diagrama entitat-relació de base de dades

Per a la base de dades i el seu disseny s'ha seguit un model relacional tradicional. Les diferents entitats estan representades per taules, cada element s'identifica amb una clau primària única i les relacions entre els diferents elements es modela per mitjà de claus foranies.

Amb això, el disseny del model de dades de l'aplicació està representat en el següent diagrama entitat-relació:



Imatge 7: Diagrama entitat-relació de base de dades

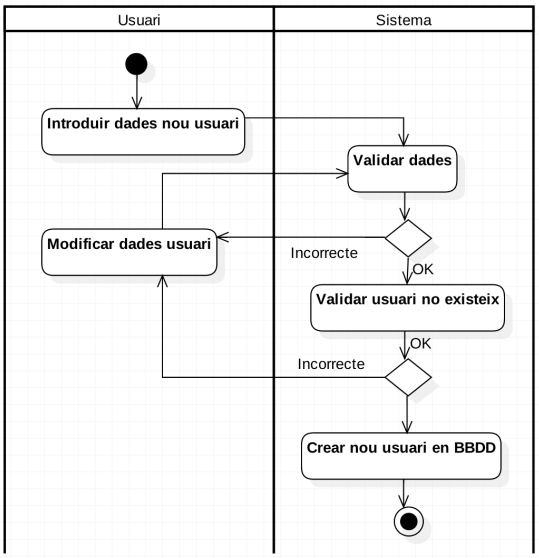
2.1.4. Diagrames d'activitat

Per a modelar les principals activitats de l'aplicació, s'han dividit les activitats en quatre grups segons els àmbits de les funcionalitats:

- Creació d'usuaris
- Gestió de grups
- Gestió de notes
- Gestió de llistes

2.1.4.1. Usuaris

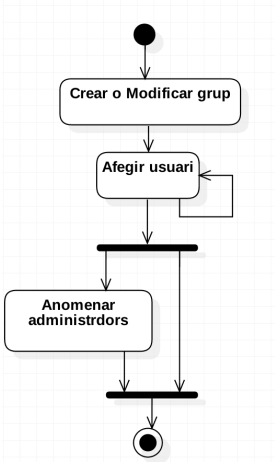
El diagrama d'activitats per a la creació d'usuaris es mostra a continuació:



Imatge 8: Diagrama d'activitats per a la creació d'un usuari

2.1.4.2. Grups

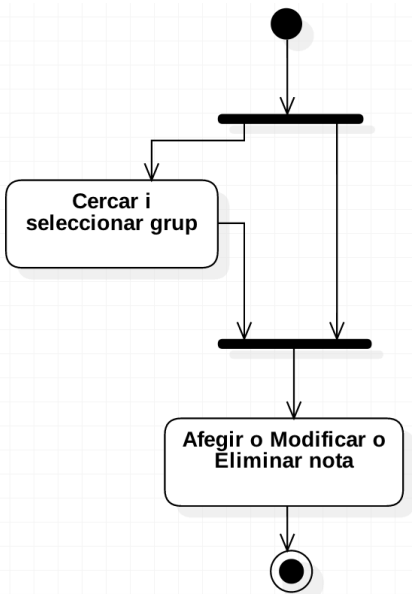
El diagrama d'activitats per a la gestió de grups es mostra a continuació:



Imatge 9: Diagrama d'activitats per a la gestió de grups

2.1.4.3. Notes

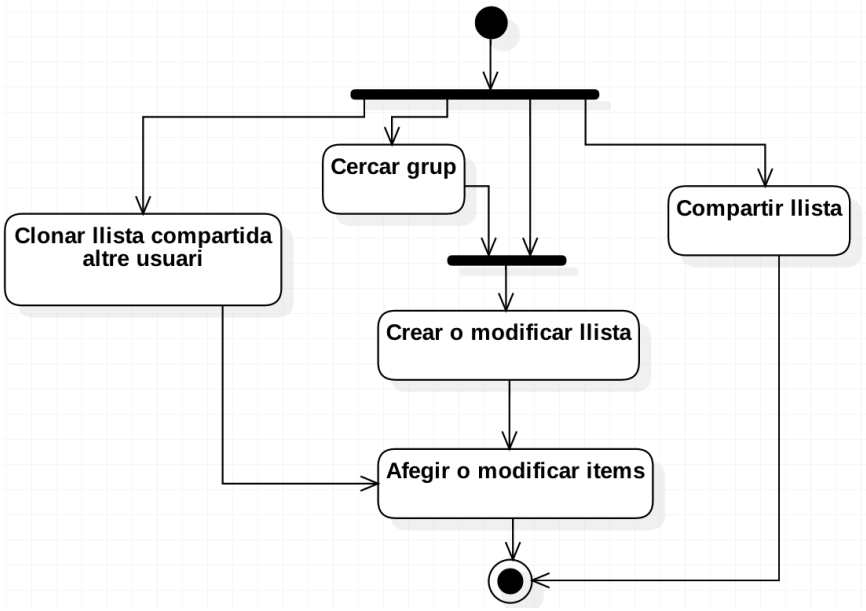
El diagrama d'activitats per a la gestió de notes es mostra a continuació:



Imatge 10: Diagrama d'activitats per a la gestió de notes

2.1.4.4. Llistes

El diagrama d'activitats per a la gestió de llistes es mostra a continuació:



Imatge 11: Diagrama d'activitats per a la gestió de llistes

2.2. Disseny de la interfície d'usuari i usabilitat

2.2.1. Perfil d'usuaris

Per dissenyar una aplicació que sigui usable per als seus usuaris, és necessari entendre quina tipologia d'usuari faran us de l'aplicació [3]. A continuació s'han definit dos perfils d'usuaris que potencialment poden fer us de l'aplicació i que, per tant, tot el procés de de disseny ha de tenir en comptes les seves necessitats per tal de poder satisfer-les correctament i que la seva experiència fent us de l'aplicació sigui el més positiva possible.

Perfil d'usuari 1:

Característica	Descripció
Rang d'edat	16-32
Nivell formatiu	Estudiants i formació de nivell mitjà o superior
Experiència tecnològica	Natius digitals
Dispositiu habitual	Majoritàriament *smartphone*
Ús de l'aplicació	Organització d'events amb amics
Mitja setmanal d'ús de dispositius electrònics (PC, *tablet*, mòbil,...)	21-30 hores setmanals

Perfil d'usuari 2:

Característica	Descripció
Rang d'edat	28-45
Nivell formatiu	Formació de nivell mitjà majoritàriament
Experiència tecnològica	Interès per la tecnologia però amb certes dificultats a l'hora de ferla servir
Dispositiu habitual	Ordinador personal i *tablet* o *smartphone* en menor grau
Ús de l'aplicació	Organització personal
Mitja setmanal d'ús de dispositius electrònics (PC, *tablet*, mòbil,...)	15-20 hores setmanals

Un cop determinada la tipologia d'usuaris que potencialment faran servir l'aplicació, es poden extreure les següents conclusions, que a l'hora hauran de servir de requisits a l'hora de dissenyat la interfície i funcionalitats de l'aplicació:

- La interfície de l'aplicació ha de ser responsiva (de l'anglès *responsive*), és a dir, la interfície s'ha d'adaptar a diferents tipus de pantalles per tal de que sigui usable des d'un gran numero de dispositius diferents.
- Tot i que els usuaris tenen cert bagatge en l'ús d'aplicacions i dispositius, l'aplicació no ha d'utilitzar funcionalitats molt avançades o fora de l'estàndard habitual.
- L'aplicació ha de ser àgil de fer servir i ha de funcionar de manera correcta; és important testejar-la correctament, per tal que l'usuari no busqui altres alternatives que es puguin assimilar en el mercat.
- L'aplicació no ha de suposar la necessitat d'haver de fer una despesa econòmica per a l'usuari, per tal que tothom la pugui fer servir i no es redueixi considerablement el número de possibles usuaris potencials.

2.2.2. Histories d'usuari

Per tal de recollir els requisits que s'han definit per a l'aplicació de manera que puguin ser considerats durant les fases de disseny, desenvolupament i proves, a continuació es mostra un recull en forma d'històries d'usuari de les funcionalitats i requisits, tant funcionals, com no funcionals, que ha de cobrir l'aplicació [3]:

1. Com a usuari vull que les meves dades personals es guardin de forma segura i privada per tal de garantir la meva privacitat.
2. Com a usuari vull poder registrar-me en l'aplicació de manera ràpida, àgil i fàcil, sense haver d'introduir dades innecessàries o trigar un temps elevat.
3. Com a usuari vull poder accedir a l'aplicació des del navegador de qualsevol dispositiu per tal de poder accedir a les meves notes i llistes de manera flexible en qualsevol lloc.
4. Com a usuari vull poder accedir de manera centralitzada a les meves notes i llistes per tal de poder consultar-les i modificar-les des de qualsevol dispositiu i ubicació
5. Com a usuari vull poder compartir les meves llistes per que altres usuaris les puguin clonar i fer servir com a base.
6. Com a usuari vull poder copiar les llistes compartides d'altres usuaris per tal de poder fer-la servir com a plantilla.
7. Com a usuari vull poder crear grups privats per tal de poder gestionar llistes i notes de manera col·laborativa i poder compartir el contingut amb els integrants.

8. Com a usuari vull poder donar de baixa el meu usuari quan consideri oportú per tal de no tenir un usuari actiu de manera no desitjada.
9. Com a usuari vull poder gestionar els privilegis d'administració dels grups que creï per tal de poder cedir l'administració a la resta de usuaris que consideri oportú.
10. Com a usuari de l'aplicació vull que l'aplicació sigui àgil, ràpida i fàcil de fer servir per tal que la meva experiència sigui satisfactòria.
11. Com a creador de l'aplicació vull que l'aplicació sigui flexible i escalable per tal de poder afegir noves funcionalitats fàcilment i poder afegir nous servidors per cobrir la demanda en cas de ser necessari.
12. Com a propietari de l'aplicació vull que es pugui afegir publicitat en l'aplicació sense trencar la interfície en cas que vulgui monetitzar l'aplicació per tal de poder treure un benefici econòmic sense causar un fort impacte a l'usuari.
13. Com a propietari de l'aplicació vull que la tecnologia emprada per al seu desenvolupament i ús sigui estàndard i no requereixi de llicències per tal de poder instal·lar-la en els principals proveïdors de *clouds* sense tenir problemes de compatibilitat o costos addicionals.

Un cop definides les histories d'usuari, a continuació es relacionen aquestes amb els diferents actors i els estats d'entrada i sortida de l'aplicació i el flux dintre de l'aplicació:

Història d'usuari	Actor	Estat inicial	Flux	Estat final
1	Usuari genèric	*Requeriments no funcionals*		
2	Usuari genèric			
3	Usuari genèric			
4	Usuari genèric	Estar registrat i logat a l'aplicació	Crear una nova nota o modificar les ja existents a la pàgina principal	Nova nota creada, modificada o consultada
5	Usuari genèric	Estar registrat i logat a l'aplicació i haver creat alguna llista	Anar a la pantalla d'edició de llista i polsar sobre el botó "compartir llista"	La llista passarà a estar compartida i altres usuaris se la podran copiar
6	Usuari genèric	Estar registrat i logat a l'aplicació	Anar a la pestanya de llistes, polsar el botó "cercar llista compartida" i cercar segons el títol o la descripció i polsar sobre el botó de copiar llista	La llista s'haurà copiat a l'àrea de l'usuari o alguns dels seus grups i podrà modificar-se lliurement.
7	Usuari genèric	Estar registrat i logat a l'aplicació	En el menú lateral, seleccionar la opció "nou grup", triar un nom i afegir els usuaris.	S'haurà creat un nou grup on es poden afegir notes i llistes que poden ser tractades de manera col·laborativa
8	Usuari genèric	Estar registrat i logat a l'aplicació	En el menú superior, sobre la icona d'usuari, triar la opció "desactivar usuari"	L'usuari s'haurà desactivat i ja no es podrà fer servir per accedir a l'aplicació.
9	Usuari administrador	Estar registrat i logat a l'aplicació i ser administrador d'un grup	Editar el grup i, a l'hora d'afegir un nou usuari, seleccionar o no la opció "administrador"	Si es selecciona la opció "administrador" el nou usuari podrà modificar lliurement el grup. Si no, només podrà editar els ítems del grup (notes i llistes)
10	Usuari genèric	*Requeriments no funcionals*		
11	Administrador aplicació			
12	Administrador aplicació			
13	Administrador aplicació			

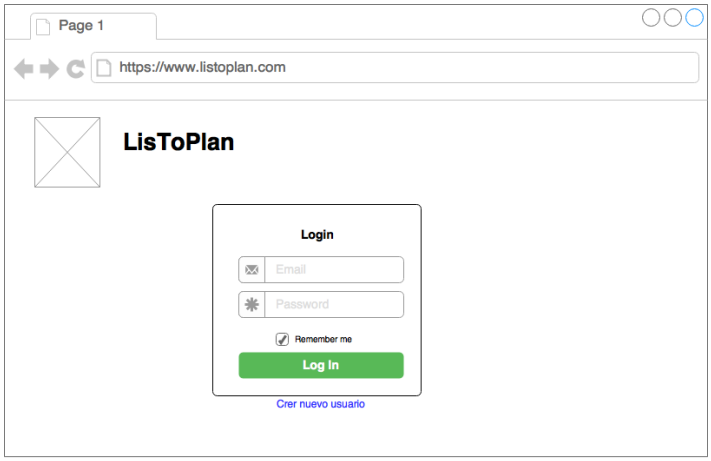
2.2.3. Prototip de baixa definició (wireframe)

Prèviament al disseny de la interfície d'usuari, a continuació es detalla un conjunt de prototips d'alt nivell o *wireframes* on es pretén donar una visió preliminar de la interfície gràfica, emfatitzant el elements més destacats de la interfície així com de la estructura general i de la forma en que es visualitzaran i introduiran les dades [3].

Es important destacar la importància que se li donarà a l'ús de "finestres modals" dinàmiques. L'ús d'aquest recurs permetrà poder carregar les dades i realitzar accions sobre aquestes per mitjà de crides asíncrones al servidor (AJAX), evitant haver de recarregar constantment la pàgina, fent que el temps de resposta sigui molt més àgil, fet que fa que millori la usabilitat de l'aplicació.

Login:

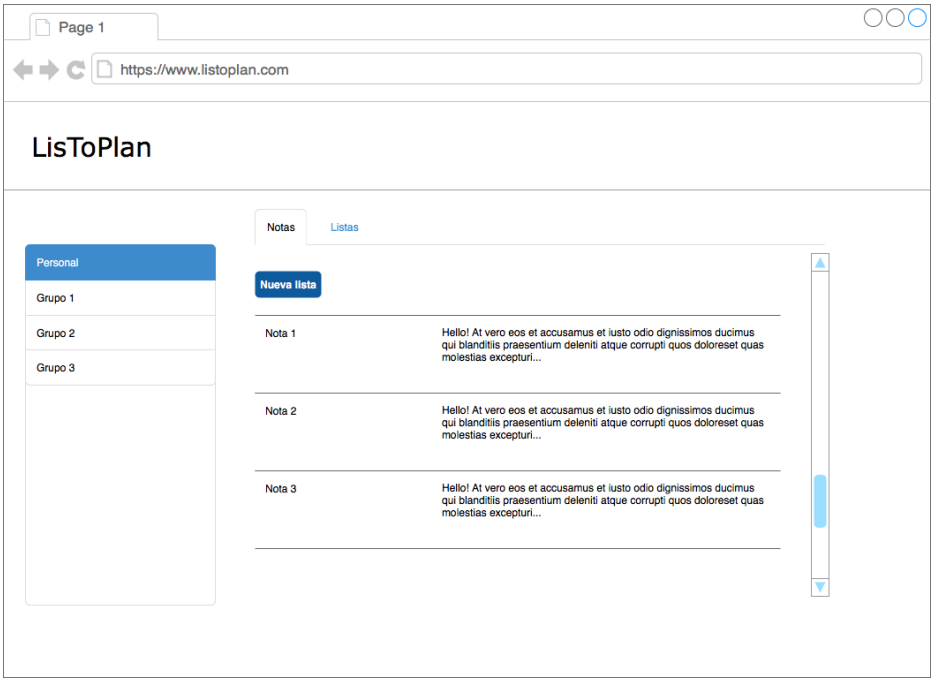
L'objectiu d'aquesta vista és que l'usuari introdueixi les seves dades d'autenticació (e-mail i password). En cas d'error d'autenticació s'ha de mostrar clarament la causa i, si la autenticació es produeix correctament, ha de redirigir a la pàgina principal de l'aplicació.



Imatge 12: Wireframe pantalla login

Dashboard:

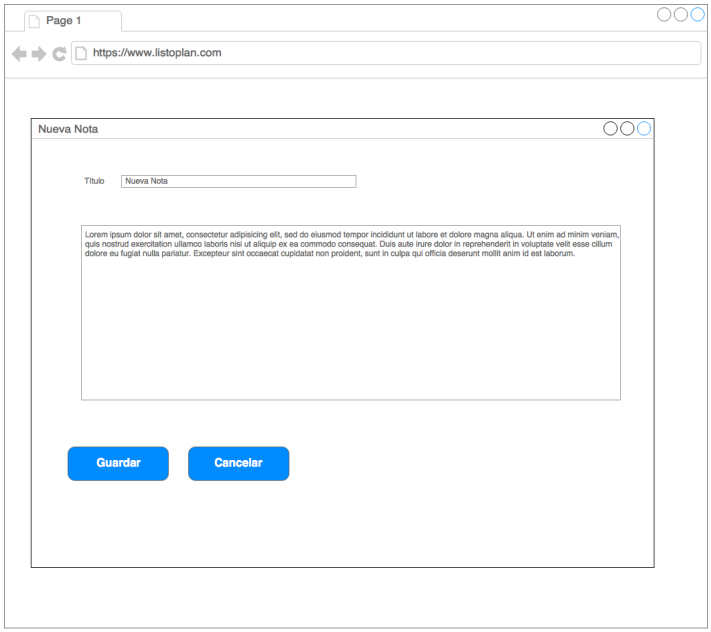
L'objectiu d'aquesta vista es la de mostrar un quadre de comandament general on l'usuari pugui veure un resum dels seus ítems personals (notes i llistes), pugui veure els seus grups i els últims moviments que s'han produït. Aquesta vista és la vista principal de l'aplicació i és la primera vista on accedeix l'usuari un cop autenticat.



Imatge 13: Wireframe pantalla dashboard principal

Afegir/Modificar nota:

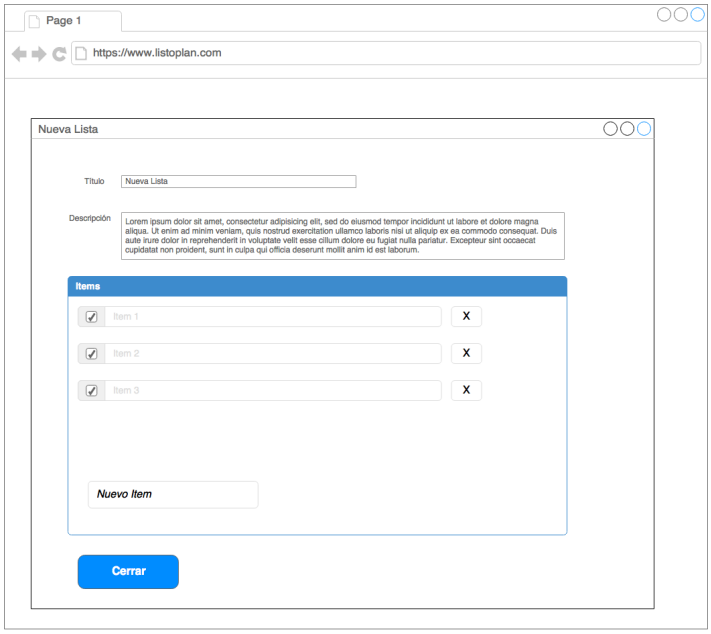
Ja sigui com a usuari individual o com a membre d'un grup, a l'hora d'afegir, modificar o eliminar una nota, al ser seleccionada, es desplegarà una finestra modal (sense haver de recarregar la pàgina) on es mostraran els quadres de text i botons corresponents als diferents atributs de la nota.



Imatge 14: Wireframe pantalla modal afegir/modificar nota

Afegir/Modificar llista:

De manera similar a la gestió de les notes, a l'hora de gestionar la llistes, tant com les pròpies de l'usuari com les d'un grup, aquesta es realitzarà des de una finestra modal on s'introduiran els diferents atributs i ítems associat a la llista.



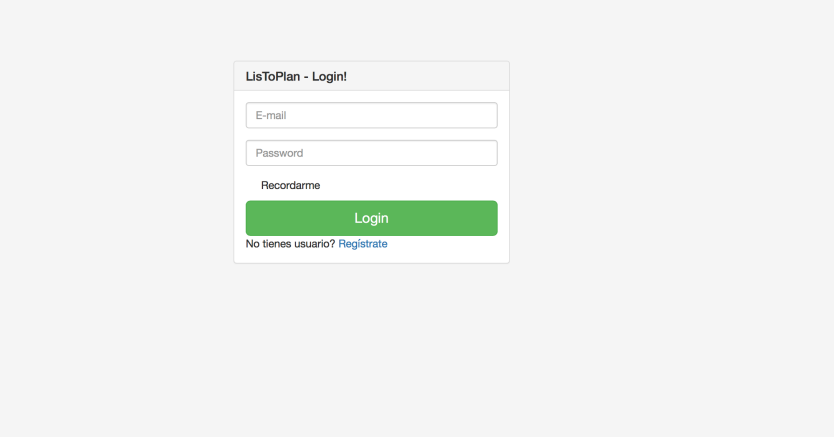
Imatge 15: Wireframe pantalla modal afegir/modificar llista

2.2.4. Prototip d'alta definició

Un cop s'ha definit com serà l'estructura gràfica de l'aplicació des d'un punt de vista d'alt nivell, el següent pas serà definir les pantalles o maquetes que es faran servir en l'aplicació. Per a l'aplicació es fa servir una llibreria d'estils anomenada *Bootstrap,* que facilita el procés de disseny i desenvolupament gràfic del frontal. Serà sobre els models presentats a continuació on es desenvoluparà tota la lògica del client per a la visualització i modificació de les dades.

Login:

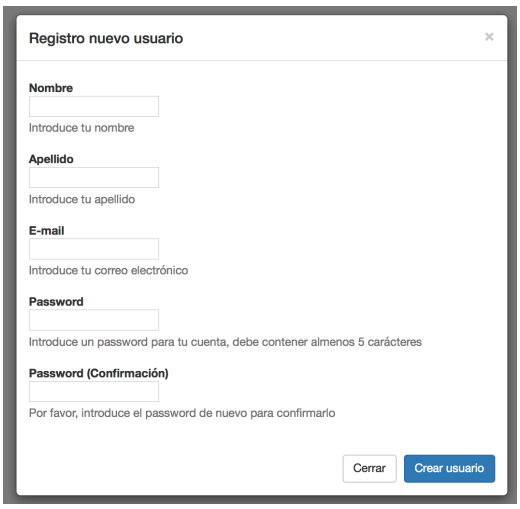
La pantalla de login és la pantalla principal que visualitza l'usuari un cop entra per primer cop a l'aplicació. L'usuari haurà d'introduir el seu e-mail i contrasenya amb el que es va registrar per poder accedir a la pantalla principal. D'es d'aquesta vista es pot accedir a la pantalla modal de registre per tal de crear un nou usuari.



Imatge 16: Prototip pantalla de login

Registre d'usuari

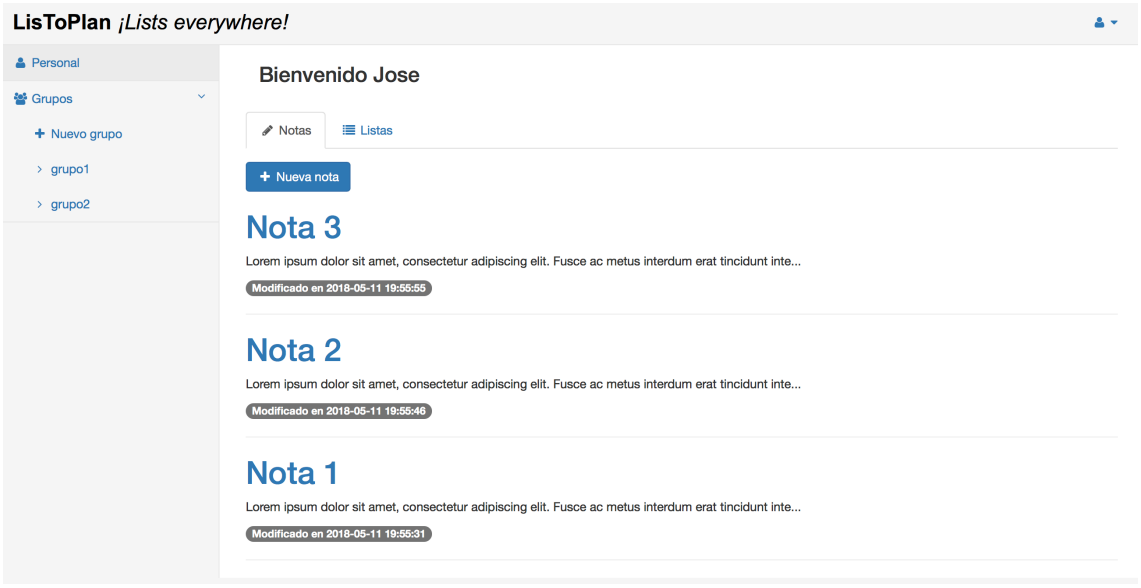
A aquesta vista s'accedeix des de la pantalla de login i des d'aquí l'usuari podrà crear-se un nou perfil. Les dades requerides per a la creació d'un nou usuari són: Nom, cognom, direcció de correu electrònic i password.



Imatge 17: Prototip pantalla de registre (modal)

Dashboard (Notes):

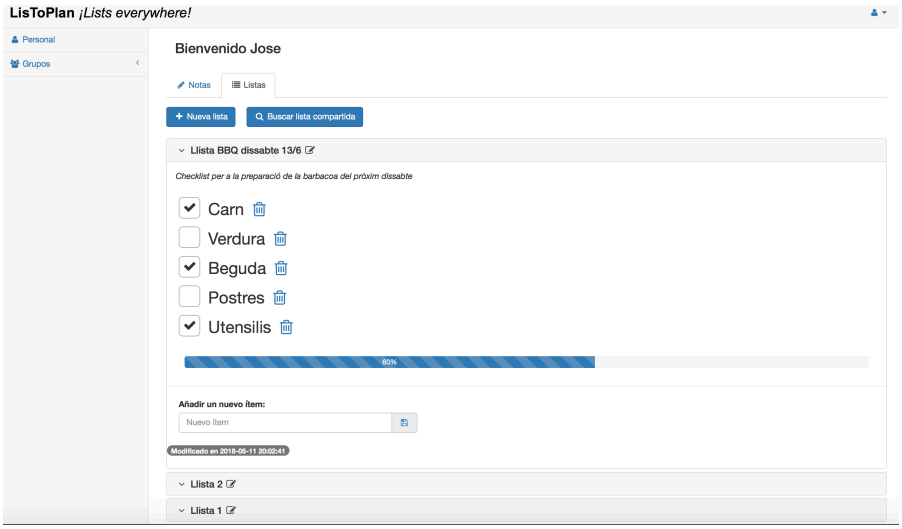
Un cop l'usuari s'ha autenticat, accedirà a la pantalla principal. Des d'aquesta vista podrà controlar les seves notes llistes i notes personals, les dels diferents grups i modificar i crear-ne nous grups. En la vista central, l'usuari podrà seleccionar entre notes i llistes. En la pestanya de notes que es mostra a continuació l'usuari podrà visualitzar, modificar i crear les llistes, ja siguin pròpies o d'un grup concret.



Imatge 18: Prototip pantalla de visualització de notes

Dashboard (Llistes):

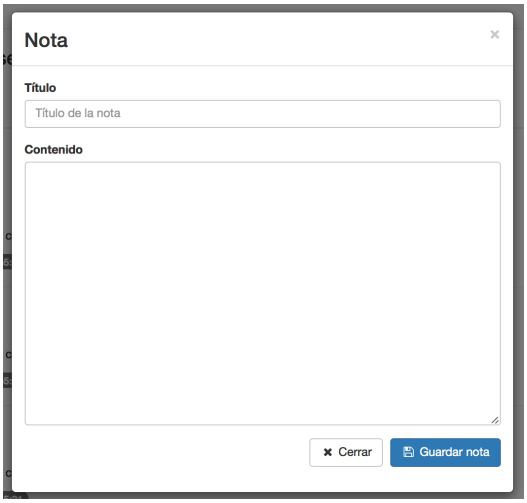
La segona pestanya de la vista central ens permet realitzar les accions equivalents de les notes però sobre les llistes. Les diferents llistes es presenten com a panels desplegables que l'usuari anirà desplegant segons convingui. Els ítems de cada llista es poden afegir, modificar i eliminar des de la pròpia pantalla.



Imatge 19: Prototip pantalla de visualització de llistes

Nova nota:

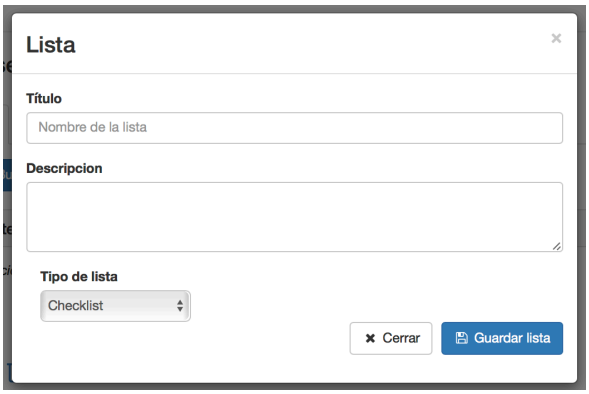
Des de la finestra modal de creació de notes es podrà crear una nova nota i modificar les existents. Simplement s'haurà d'introduir el títol i el contingut de la nota i després guardar els canvis.



Imatge 20: Prototip pantalla creació i modificació de notes (modal)

Nova llista:

La finestra modal de creació i modificació de llistes es equivalent a la pantalla de notes però per a gestionar llistes. En aquest cas, però, també a apareix la opció de seleccionar un tipus de llista (*checklist* o repartició) però només durant la fase de creació de la llista.



Imatge 21: Prototip pantalla creació i modificació de llistes (modal)

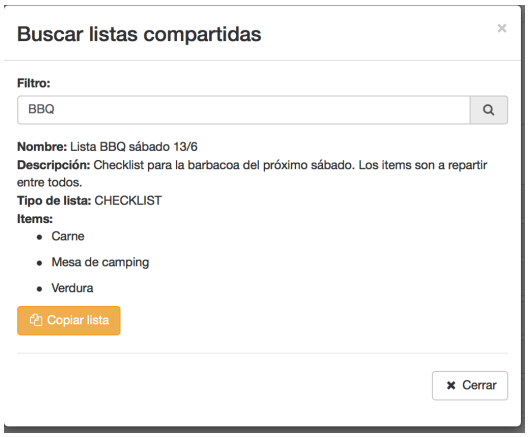
Gestió de grups:

Des de la finestra de gestió de grups, l'usuari podrà crear un grup nou, modificar els existents i vincular i desvincular usuaris. El cercador actualitza la llista d'usuaris a mesura que es van introduint caràcters en el camp de text.

Imatge 22: Prototip pantalla de creació i modificació de grups (modal)

Cercador de llistes compartides

El cercador de vistes compartides permet buscar entre les diferents llistes que els usuaris han compartit, aquelles que poden resultar útils per l'usuari i permet fer la copia per al seu ús. Introduint sobre el cercadors els valors de la cerca, s'aniran refrescant els resultats segons els caràcters introduïts.



Imatge 23: Prototip pantalla de cerca de llistes compartides

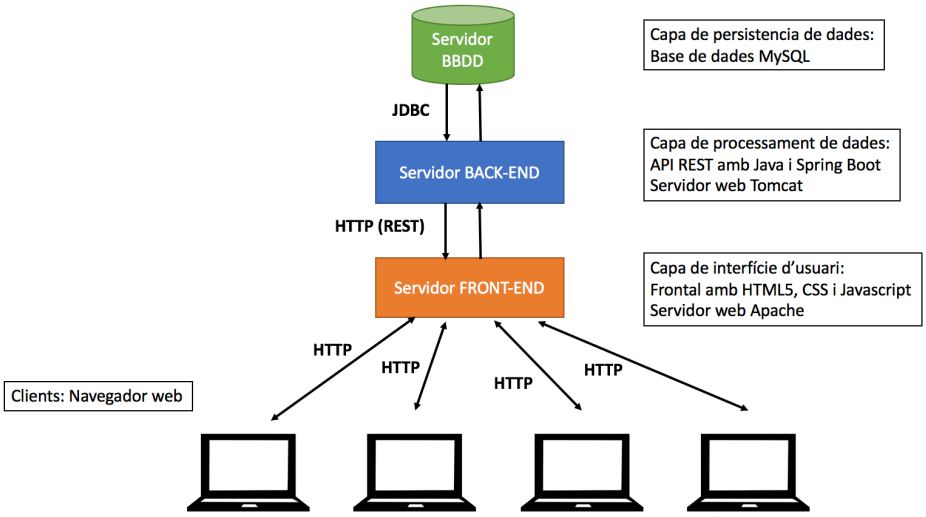
2.3. Arquitectura de l'aplicació

2.3.1. Arquitectura global del sistema

La arquitectura general de la aplicació està formada per tres capes o blocs ben diferenciats:

- Base de dades
- Back-end
- Front-end

A continuació es mostra gràficament la relació entre aquestes tres capes, les tecnologies emprades i els protocols utilitzats per a la seva comunicació:



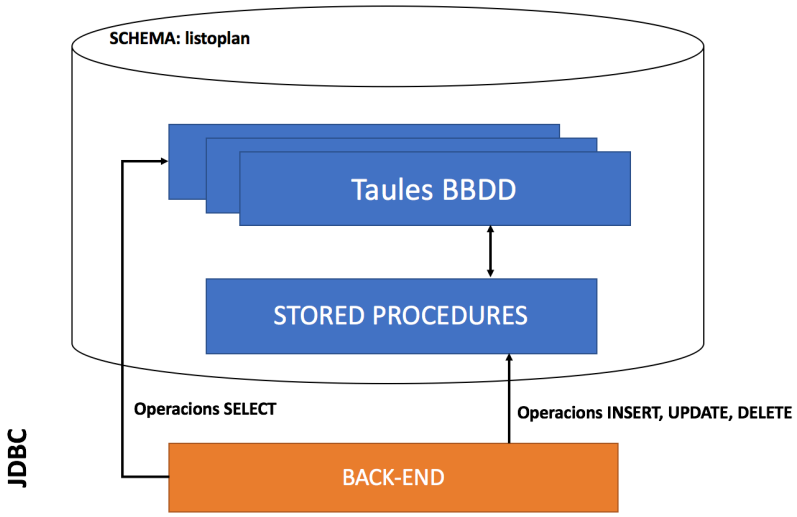
Imatge 24: Diagrama arquitectura general del sistema

En els capítols següents es dona una visió més detallada de cada capa i de les tecnologies i arquitectures que s'han fet servir per al seu desenvolupament.

2.3.2. Arquitectura capa de persistència de dades

La capa de persistència de dades es fonamenta en una base relacionals MySQL [4].
El back-end es connecta directament a la base de dades per mitjà del connector JDBC estàndard per a MySQL. Per tal de garantir la consistència de les dades de l'aplicació, minimitzar els errors i dividir correctament les responsabilitats, les diferents accions possibles sobre les dades es realitzen de la següent manera:

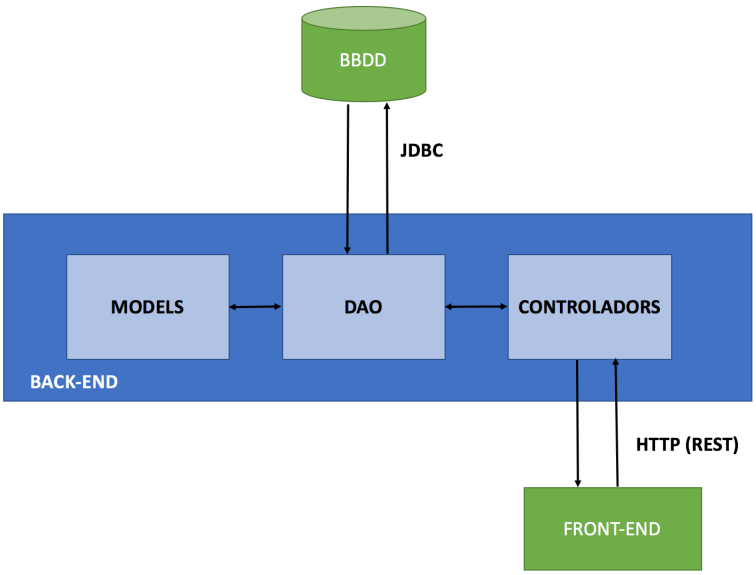
- Operacions de consulta de dades (sentències SELECT): Es realitzen directament des del back-end, integrant les deferents consultes a base de dades en el codi.
- Operació de modificació de dades (sentències INSERT, UPDATE i DELETE): Aquestes sentències no es construeixen en el back-end sinó que les diferents possibilitats estan programades en forma de STORED PROCEDURES a base de dades, d'aquesta manera, en el back-end, únicament es necessari fer una crida als diferents procediments. Amb aquest enfoc, es garanteix la **transaccionalitat** de les dades: des del procediment de la base de dades es controla que la operació finalitza quan s'han realitzat totes les accions en totes les taules que contempla el *procedure* i, en cas que es produeixi qualsevol problema durant la seva execució, no es realitza el *commit* de les dades que s'han creat durant l'execució errònia.



Imatge 25: Diagrama arquitectura de Base de Dades

2.3.3. Arquitectura Servidor

L'arquitectura del back-end es basa en un disseny Model-Controlador (al tractar-se d'una API REST [5] no s'han de gestionar les vistes des del servidor). Partint d'aquest paradigma, s'afegeix un nou element a l'estructura: les classes DAO (Data Acces Object) que segueixen el patró amb el mateix nom i que fan de interconnexió entre el Model-Controlador i la base de dades:



Imatge 26: Diagrama arquitectura del back-end

Com es pot observar en el diagrama anterior, les diferents classes del servidor estan dividides en tres tipologies segons la seva funcionalitat. Les responsabilitats dels tres tipus de classes són les següents:

- **Classes controladors**: Defineixen la ruta de la URL i el mètode HTTP (GET/POST) sobre el qual s'executarà el mètode associat a la crida. El controlador rep la petició, la gestiona i retorna la resposta corresponent.
- **Classes DAO**: Contenen els mètodes que obtenen les dades de la base de dades. Aquests mètodes són cridats des de el controlador i obtenen de la base de dades la informació necessària per crear els objectes definits per les classes del model. A més dels mètodes per obtenir informació també contenen els mètodes necessaris per realitzar accions de modificació de dades sobre la base de dades, això és, realitzar les crides als procediments de base dades.

- **Classes del Model**: Són classes que segueixen el patró *Java Bean* que representen instàncies de base de dades i, per extensió, de conceptes propis de l'aplicació i del seu context (usuaris, grups, notes, llistes,...).

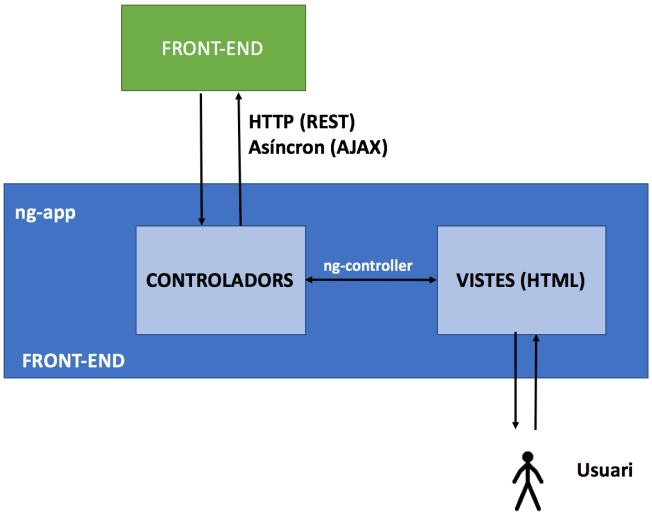
Amb la perspectiva anterior, el flux d'execució i de les dades segueix el següent camí lògic:

1. El controlador rep la petició (***Request***) i la gestiona.
2. Per obtenir o modificar les dades, el controlador fa les crides a els mètodes definits en les classes del tipus DAO.
3. Les classes DAO realitzen les operacions i/o consultes necessàries en base dades i, en cas d'haver de retornar dades, generen els diferents objectes definits per les classes del model.
4. Finalment, el controlador rep la resposta del mètode de la classe DAO corresponent en forma d'objecte (d'una classe del model) o en forma d'estat del resultat de l'operació. Amb això, retorna la resposta i l'estat HTTP corresponent al client (***Response***).

2.3.4. Arquitectura Client

L'arquitectura del client o *front-end,* està basada en el *framework angularJS* [6]. Aquest *freamework* permet desenvolupar aplicacions client seguint el model MVC de manera totalment desacoblada del servidor. Amb això, l'usuari descarrega únicament l'aplicació client estàtica (únicament programada en HTML5 i JavaScript) i aquesta recupera tota la informació necessària del servidor per mitjà de crides asíncrones al servidor (Ajax). D'aquesta manera el servidor únicament té la responsabilitat de respondre les peticions que fa el client amb les dades pertinents en format JSON, que es el client qui les gestiona i les mostra per pantalla. Amb això s'evita fer recàrregues constants de les vistes i que el servidor hagi de generar l'HTML. El resultat d'això és una navegació molt fluida i una experiència molt més positiva per l'usuari.

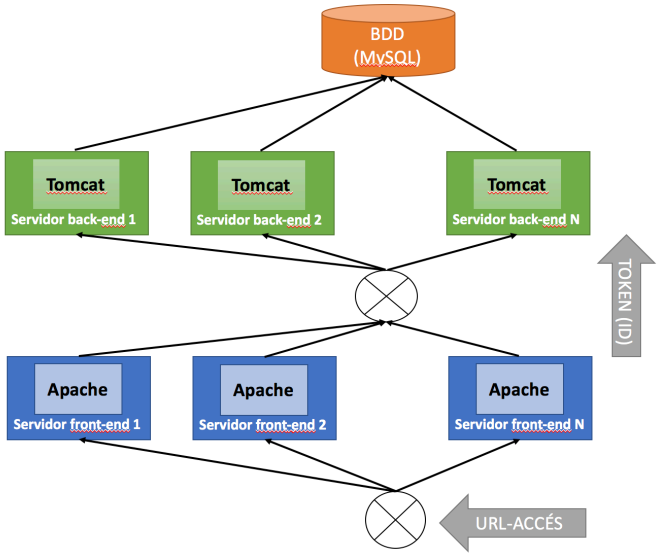
Com es pot observar en el diagrama següent, la vista i el controlador (la lògica de l'aplicació) es comuniquen per mitjà de la directiva *ng-controller*. Segons els events que genera l'usuari (carregar una pàgina, polsa un botó o un enllaç, introduir un text en un camp de text,...) els controladors realitzen una sèrie d'accions, recuperen les dades necessàries del servidor i les retornen a la vista per que les mostri a l'usuari. El servidor, amb aquestes dades tornarà a realitzar una altra sèrie d'accions que generaran uns altres events que gestionarà el servidor i es tornarà a repetir tot el flux d'accions.



Imatge 27: Diagrama arquitectura del front-end

2.3.5. Arquitectura aplicació distribuïda

Gràcies a l'arquitectura definida per a la aplicació, aquesta pot ser desplegada de manera monolítica, és a dir, totes les seves capes es poden executar en el mateix servidor, o bé de manera distribuïda, de manera que cada capa s'executa en un servidor diferent i que, a més, pot ser escalada horitzontalment afegint instàncies tant del *front-end* com del *back-end* per mitjà de balancejadors. Cada token identifica la sessió de l'usuari de manera única, per tant, cada petició pot ser tractada de manera independent per cada servidor. Aquest plantejament permet aprofitar tecnologies com recursos flexibles i dinàmica en el núvol en funció de la càrrega i desplegament de l'aplicació fent servir contenidors:



Imatge 28: Diagrama aplicació distribuïda

3. Desenvolupament de l'aplicació

3.1. Extractes de codi

En aquest apartat es presenta un conjunt d'extractes de codi de l'aplicació que representen les parts més destacades i diferenciades del desenvolupament. Com l'aplicació està dividida en tres capes de tecnologies diferents, l'apartat s'estructurarà seguint aquesta divisió: Base de dades, *Back-end* i *Front-end.*

3.1.1. Extractes de codi de base de dades

El codi referent a base de dades es pot dividir en tres tipologies: les consultes i sentencies de modificació de dades (SQL), les sentencies per a la creació de taules i vistes (DDL) i els *Stored Procedures.* Els *Stored Procedures* són un conjunt de rutines que s'emmagatzemen a la base de dades i son cridats des del servidor de l'aplicació. Amb això s'aconsegueix dividir les responsabilitats i desacoblar el codi entre el servidor i la base de dades.

Els *Stored Procedures* segueixen sempre una mateixa estructura. A continuació es detalla el procediment que s'encarrega de crear una nova nota:

```
CREATE PROCEDURE `notas_crear`(IN P_ID INT, IN P_TITULO VARCHAR(60), IN
P_CONTENIDO TEXT, IN P_TIPO VARCHAR(7))
BEGIN
     DECLARE V_ID_NOTA INT;
    SET AUTOCOMMIT=0;
     START transaction;
    INSERT INTO notas SET
    TITULO=P_TITULO,
    CONTENIDO=P_CONTENIDO,
    FECHA_CREACION=now(),
    FECHA_MODIFICACION=now(),
    ACTIVO=1;
    SELECT `AUTO_INCREMENT`-1 into V_ID_NOTA FROM INFORMATION_SCHEMA.TABLES
     WHERE TABLE_SCHEMA = 'listoplan' AND TABLE_NAME = 'notas';

     IF P_TIPO="USUARIO" THEN
           INSERT INTO usuario_notas SET
           FK_ID_USUARIO=P_ID,
           FK_ID_NOTA=V_ID_NOTA;
     ELSE
           INSERT INTO grupo_notas SET
           FK_ID_GRUPO=P_ID,
           FK_ID_NOTA=V_ID_NOTA;
     END IF;
     COMMIT;
END;
```

El codi anterior ens permet il·lustrar diferents elements importants. El primer de tot, es l'estructura. Els procediments es defineixen de les següent manera:

```
CREATE PROCEDURE `<<nom>>` (<<IN/OUT>> <<NOM_PARAMETRE>> <<TIPUS>>, ...)
BEGIN
     ...
END;
```

On els elements variables són:

- <<nom>>: Nom del procediment
- <<IN/OUT>>: Indicar si el paràmetre és d'entrada o de sortida
- <<NOM_PARAMETRE>>: Nom que se li dona al paràmetre per que pugui ser referenciat al codi
- <<TIPUS>>: Tipus de dada del paràmetre (INT, VARCHAR,...)

Per cridar el procediment anterior s'utilitza la següent sentència:

```
CALL <<nom>> (<<PARAMETRE_1>>, ..., <<PARAMETRE_N>>);
```

El cos del *Procedure* no es res més que un conjunt de sentències SQL (SELECT, INSERT, UPDATE, DELETE) que poden incorporar algunes sentències de control de fluxe (IF, WHILE, ...).

En l'exemple anterior es pot observar com en el procediment es fa un insert de la nota en la taula "notas" i després fa un insert en la taula "usuario_notas" o "grupos_nota" segons correspongui, en funció d'un paràmetre d'entrada.

Un element molt interessant de l'exemple anterior és la **transaccionalitat**. La transaccionalitat fa que tot el conjunt de les operacions del procediment es tractin com una sola. És a dir, si durant l'execució d'una operació unitària es produeix un error, es restableixen els resultats modificats en les operacions anteriors. D'aquesta manera s'aconsegueix la **consistència** de les dades. Amb la sentència "START transaction" s'indica que es comença un conjunt d'operacions que s'han de tractar de manera unitària. Amb la sentència "COMMIT" s'indica que es persisteixin els canvis de les operacions anterior. Aquesta sentència només s'executarà si no s'ha produït cap error en les operacions anteriors.

Hi ha altres *procedures* més senzills, com el que s'utilitza per editar una nota, que no necessiten de controlar la transaccionalitat ja que només s'executa una única operació:

```
CREATE PROCEDURE `notas_modificar`(IN P_ID_NOTA INT, IN P_TITULO VARCHAR(60),
IN P_CONTENIDO TEXT)
BEGIN
     UPDATE notas SET
    TITULO=P_TITULO,
    CONTENIDO=P_CONTENIDO,
    FECHA_MODIFICACION=now()
    WHERE ID_NOTA=P_ID_NOTA;

END ;
```

Finalment, l'últim exemple de procediment molt interessant és el que es fa servir per crear un nou usuari, on es calcula el camp "Salt" que es fa servir per a potenciar la seguretat i privacitat del password d'usuari (més informació a l'apartat 3.2. Seguretat):

```
CREATE PROCEDURE `usuarios_crear`(IN P_EMAIL VARCHAR(100), IN P_NOMBRE
VARCHAR(30), IN P_APELLIDO VARCHAR(50), IN P_CONTRASENA_ORIG VARCHAR(100))
BEGIN
DECLARE V_SALT VARCHAR(10);
DECLARE V_CONTRASENA VARCHAR(100);

select SUBSTRING(MD5(RAND()), 1, 10) into V_SALT;
select SHA2(CONCAT(V_SALT,P_CONTRASENA_ORIG),256) into V_CONTRASENA;

INSERT INTO usuarios SET
EMAIL=P_EMAIL,
NOMBRE=P_NOMBRE,
APELLIDO=P_APELLIDO,
CONTRASENA=V_CONTRASENA,
SALT=V_SALT,
FECHA_CREACION=NOW(),
FECHA_MODIFICACION=NOW(),
ACTIVO=1;
END ;
```

L'altra tipologia de codi fet servir en la base de dades MySQL, a banda de les sentències SQL i dels *Stored Procedures,* són les sentències per crear objectes a base de dades (DDL, *Data Definition Language*). Per mitjà d'aquestes sentències es poden crear els objectes de Base de Dades (taules, vistes,...), les seves propietats (nom i tipus de cada camps, claus primàries,...) i les relacions i *constraints* entre les diferents taules (claus foranies). En l'exemple següent es pot observar un exemple complert d'una sentència DDL per a la creació de la taula "usuario_notas":

```
CREATE TABLE `usuario_notas` (
  `FK_ID_NOTA` int(11) NOT NULL,
  `FK_ID_USUARIO` int(11) NOT NULL,
  PRIMARY KEY (`FK_ID_NOTA`,`FK_ID_USUARIO`),
  KEY `USUARIO_idx` (`FK_ID_USUARIO`),
  CONSTRAINT `NOTA` FOREIGN KEY (`FK_ID_NOTA`) REFERENCES `notas` (`ID_NOTA`)
ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `USUARIO` FOREIGN KEY (`FK_ID_USUARIO`) REFERENCES `usuarios`
(`ID_USUARIO`) ON DELETE CASCADE ON UPDATE CASCADE
)
```

Els elements més destacats de la sentència anterior són:

- Estructura de la sentència on usuario_notas és el nom de la taula i FK_ID_NOTA i FK_ID_USUARIO ambdós del tipus INT i no nuls són els dos camps de la taula que s'està definint.
- La sentència PRIMARY KEY indica quins camps conformen la clau primària (camps que identifiquen un registre de manera única). En aquests cas els dos camps FK_ID_NOTA i FK_ID_USUARIO conformen la clau primària de la taula.
- Les sentències CONSTRAINT ... FOREIGN KEY ... REFERENCES indiquen que els camps FK_ID_NOTA i FK_ID_USUARIO són claus foranies dels camps ID_NOTA i ID_USUARIO de les taules NOTAS i USUARIOS respectivament. Això implica que, de manera necessària, tots els elements que s'introdueixin en aquests camps han d'existir en els camps de les taules indicades en la sentència. En cas de que no existeixi es produirà un error i no es guardarà el registre a la taula.
- Sentència ON DELETE CASCADE ON UPDATE CASCADE indiquen les accions que s'han de realitzar en cas d'eliminar o modificar un registre respectivament en la taula on es referència la clau forana. En aquest cas, si s'elimina un registre, tots els registres de la taula "usuario_notas" seran eliminats i, d'igual manera, en cas d'editar un registre, tots els registres que el referenciïn a la taula "usuario_notas" també seran modificats en cascada.

3.1.2. Extractes de codi del back-end

El *back-end* està desenvolupat en llenguatge Java, fent servir el *framework Spring Boot* que facilita la gestió de les peticions i respostes HTTP que arriben al servidor. La resposta sempre es en format JSON, per lo qual en cap cas es generen o envien vistes (pàgines HTML) des del servidor.

El primer exemple és un extracte de codi on es gestiona una petició del tipus GET:

```
    @RequestMapping(value="/notas/detalle_nota_usuario/{idNota}", method=
RequestMethod.GET)
    public ResponseEntity<Nota> detalleNotaUsuario(@PathVariable String
idNota, @RequestHeader String token) {
           InfoSesion is=TokenUtils.validarToken(token);
           if(is==null) return new
ResponseEntity<Nota>(HttpStatus.UNAUTHORIZED);

     if(!NotaDAO.esPropietarioNota(is.getIdUsuario(),Integer.parseInt(idNota),
AmbitoNota.USUARIO)){
                 return new ResponseEntity<Nota>(HttpStatus.FORBIDDEN);
           }
           Nota nota= NotaDAO.getNotaPorId(Integer.parseInt(idNota));
           return new ResponseEntity<Nota>(nota, HttpStatus.OK);
    }
```

Per mitjà de l'anotació "@RequestMapping" s'indica la ruta de la petició sobre la qual s'executarà el fragment de codi (/notes/detalle_nota_usuario/<<id_nota>>). Per mitjà de la sintaxi {idNota} s'indica que el valor que rebi en aquesta part es tracta d'una variable que serà controlada en el codi. Per mitjà del atribut "method" s'indica que s'espera una petició del tipus GET.

En el codi d'aquesta petició es realitzen els següents passos:

- Es comprova que el token que es s'envia en el *header* de la petició (anotació "@RequestHeader" és vàlid. En cas que no sigui vàlid es retorna un error HTTP amb l'estatus UNAUTHORIZED (codi 401)
- Es valida que l'usuari (s'extreu del token) es propietari de la nota, ja sigui perquè sigui seva personal o d'un grup al que pertany. En cas que no sigui propietari es retorna un error HTTP amb l'estatus FORBIDDEN (codi 403)
- Es crida al mètode que recupera la informació corresponent al detall de la nota en base de dades.
- Retorna l'objecte en format JSON amb l'estatus HTTP OK (codi 200).

El següent exemple es similar a l'anterior però gestiona una petició del tipus POST:

```
    @RequestMapping(value="/notas/nueva_nota", method= RequestMethod.POST)
    public ResponseEntity<HashMap<String,String>> nuevaNota(@RequestBody
String data, @RequestHeader String token) {
          InfoSesion is=TokenUtils.validarToken(token);
          if(is==null) return new
ResponseEntity<HashMap<String,String>>(HttpStatus.UNAUTHORIZED);
          JsonParser jp = JsonParserFactory.getJsonParser();
          Map<String, Object> resultado = jp.parseMap(data);
          String idRequest=(String) resultado.get("id");
          String titulo=(String) resultado.get("titulo");
          String contenido=(String) resultado.get("contenido");
          String ambito=((String) resultado.get("ambito")).toUpperCase();
          if(ambito.equals("GRUPO") &&
!GrupoDAO.esMiembroGrupo(is.getIdUsuario(), Integer.parseInt(idRequest))){
               return new
ResponseEntity<HashMap<String,String>>(HttpStatus.FORBIDDEN);
          }
          AmbitoNota an;
          int id;
          if(ambito.equals("GRUPO")){
               an=AmbitoNota.GRUPO;
               id=Integer.parseInt(idRequest);
          }
          else if (ambito.equals("USUARIO")) {
               an=AmbitoNota.USUARIO;
               id=is.getIdUsuario();
          }
          else {
               HashMap<String,String> res = new HashMap<String,String>();
               res.put("status","Ámbito no válido, debe ser USUARIO o
GRUPO");
               res.put("resultado","KO");
               return new
ResponseEntity<HashMap<String,String>>(res,HttpStatus.BAD_REQUEST);
          }
          String status= NotaDAO.crearNota(id, titulo, contenido, an);
          HashMap<String, String> respuesta = new HashMap<String,String>();
          if(status.contains("Error")){
               respuesta.put("status", status);
               respuesta.put("resultado", "KO");
               return new ResponseEntity<HashMap<String,String>>(respuesta,
HttpStatus.INTERNAL_SERVER_ERROR);
          }else {
               respuesta.put("status", status);
               respuesta.put("resultado", "OK");
               return new ResponseEntity<HashMap<String,String>>(respuesta,
HttpStatus.OK);
          }
    }
```

En aques cas, la petició es gestiona de manera similar fent servir l'anotació ("@RequestMapping") però indicant-li que s'ha d'executar per a peticions del tipus POST. A diferencia del GET anterior, que rebia la informació per URL, en aquest cas la informació corresponent a la nova nota a crear es rep pel cos de la petició en format JSON, per tant, per extreure els diferents elements, es necessari parsejar el cos de la petició. És necessari, per tant, definir correctament els atributs que es faran servir en la petició per tal de que el servidor i el client es puguin intercanviar la informació de manera satisfactòria.

Encara que la funció anterior es més llarga que l'anterior, segueix un flux similar que és el que de manera genèrica es segueix en totes les peticions programades al servidor:

- Comprovar que el token és vàlid. En cas d'error es retorna l'estatus UNAUTHORIZED (Codi 401).
- Extraure la informació de la petició. En cas d'error al obtenir els valors es retorna l'estatus BAD REQUEST (Codi 400).
- Validar que l'usuari (informació extreta del token) està autoritzat a modificar/recuperar dades. En cas de que no ho sigui es retorna l'estatus FORBIDDEN (Codi 403).
- Si no es produeix cap error es retorna la informació requerida si s'escau en format JSON amb l'estatus OK (Codi 200).
- En cas d'error no controlat es retorna l'estatus INTERNAL SERVER ERROR (Codi 500) per defecte.

A continuació s'exposen dos exemples molt interessants: els fragments de login i creació d'usuari:

Login:

```
    @RequestMapping(value="/login", method = RequestMethod.POST)
    public ResponseEntity<Token> loginUsuario(@RequestBody String data) {
          JsonParser jp = JsonParserFactory.getJsonParser();
          Map<String, Object> resultado = jp.parseMap(data);
          Token token=UsuarioDAO.loginUsuario((String) resultado.get("email"),
(String) resultado.get("contrasena"));
          if(token.getEstadoToken()==EstadoToken.KO) {
               return new ResponseEntity<Token>(token,
HttpStatus.UNAUTHORIZED);
          }else {
               return new ResponseEntity<Token>(token, HttpStatus.OK);
          }
    }
```

Creació d'usuari:

```
    @RequestMapping(value="/usuarios/nuevo_usuario", method=
RequestMethod.POST)
    public ResponseEntity<HashMap<String,String>> nuevoUsuario(@RequestBody
String data) {
          JsonParser jp = JsonParserFactory.getJsonParser();
          Map<String, Object> resultado = jp.parseMap(data);
          String status= UsuarioDAO.crearUsuario((String)
resultado.get("email"), (String) resultado.get("nombre"),
                    (String) resultado.get("apellido"), (String)
resultado.get("contrasena"));
          HashMap<String, String> respuesta = new HashMap<String,String>();
          if(status.contains("Error")){
               respuesta.put("status", status);
               respuesta.put("resultado", "KO");
               return new ResponseEntity<HashMap<String,String>>(respuesta,
HttpStatus.CONFLICT);
          }else {
               respuesta.put("status", status);
               respuesta.put("resultado", "OK");
               return new ResponseEntity<HashMap<String,String>>(respuesta,
HttpStatus.OK);
          }
    }
```

En els exemples anteriors es pot observar com es crida al generador de Tokens i s'envia a l'usuari en cas de login correcte i, per altra banda, en la funció de creació d'usuari, s'introdueix l'estatus CONFLICT (codi 409) en cas de que s'estigui creant un usuari que ja s'ha registrat prèviament.

Un cop s'han exposat els exemples, els quals representen la gestió de les peticions i respostes del servidors, a continuació s'exposa la classe que s'encarrega de la creació i validació dels tokens:

```
public class TokenUtils {

     static Logger logger= Logger.getLogger(TokenUtils.class);
     //Time-to-live del token en segundos
     private static final int TTL=604800; //7 dias
     private static final String SECRET="sjksfdfd345jdij0345kdcv9";

     public static Token crearToken(Usuario usuario){
          if(usuario==null) {
               return new Token(null,EstadoToken.KO);
          }
          Long millisActual= System.currentTimeMillis();
          Date d= new Date(millisActual+TTL*1000);
          String jwt=Jwts.builder()
                    .setSubject(Integer.toString(usuario.getId_usuario()))
                    .setExpiration(d)
                    .claim("email",usuario.getEmail())
                    .claim("nombre",usuario.getNombre())
```

```
                    .claim("apellido", usuario.getApellido())
                    .signWith(SignatureAlgorithm.HS256, SECRET)
                    .compact();
            return new Token(jwt, EstadoToken.OK);
      }

      public static InfoSesion validarToken(String token){
            try{
                  Jws<Claims> claimsJws=
Jwts.parser().setSigningKey(SECRET).parseClaimsJws(token);
                  int idUsuario=
Integer.parseInt(claimsJws.getBody().getSubject());
                  String email= (String) claimsJws.getBody().get("email");
                  String nombre= (String) claimsJws.getBody().get("nombre");
                  String apellido= (String) claimsJws.getBody().get("apellido");
                  return new InfoSesion(idUsuario,email,nombre,apellido);
            }
            catch(SignatureException e){
                  logger.error("Error de firma token",e);
                  return null;
            }
            catch(ExpiredJwtException e){
                  logger.info("Error: token expirado",e);
                  return null;
            }
            catch(Exception e) {
                  logger.error("Error: ",e);
                  return null;
            }
      }
}
```

En la classe anterior estan definits els mètodes “crearToken” i “validarToken”. El primer és l’encarregat de generar el token. Per fer-ho es necessari definir de manera genèrica quina serà la paraula de pas secreta i la validesa que tindrà. Un cop fet això, cada token contindrà una certa informació específica de l’usuari: email, nom i cognom. Un cop generat, aquest es retorna a l’usuari (navegador) que l’emmagatzemarà i l’enviarà a les capçaleres (*headers)* de cada petició.

El mètode validarToken realitza el procés invers: Rep un token i n’extreu la informació. En cas de que el token no sigui vàlid o estigui caducat retorna un valor nul que s’interpreta com que el token no es correcte i, per tant, la autenticació no s’ha produït correctament. Això es farà arribar al client que redireccionarà a l’usuari cap a la pàgina de login.

L’aplicació segueix el model MVC. Al servidor, però, no es generen ni gestionen les vistes, per tant a part dels Controlador definits en la primera part de l’apartat, el servidor fa servir classes que representen els Models de les diferents entitats lògiques de l’aplicació. Aquests Models o entitats són 5 (Usuari, Nota, Grup, Llista i ItemLlista).

Tots segueixen la mateixa estructura (JavaBean). A mode d'exemple, a continuació s'exposa la classe que representa el model Llista:

```
public class ItemLista {
	private int idItem;
	private String nombre;
	private String valor;
	private int item;

	public ItemLista(int idItem, String nombre, String valor, int item) {
		this.idItem = idItem;
		this.nombre = nombre;
		this.valor = valor;
		this.item = item;
	}

	public int getIdItem() {
		return idItem;
	}

	public void setIdItem(int idItem) {
		this.idItem = idItem;
	}

	public String getNombre() {
		return nombre;
	}

	public void setNombre(String nombre) {
		this.nombre = nombre;
	}

	public String getValor() {
		return valor;
	}

	public void setValor(String valor) {
		this.valor = valor;
	}

	public int getItem() {
		return item;
	}

	public void setItem(int item) {
		this.item = item;
	}
}
```

Les característiques d'aquestes classes són:

- Tots els atributs són privats
- Aquells atributs que haguin de ser públics o han de fer per mitjà de *getters i setters*
- La classe ha de tenir un constructor per a instanciar cada objecte amb els seus atributs.

Finalment, l'últim que cal destacar del codi del servidor és l'accés a base de dades per mitjà de les classes DAO (*Data Access Object*). Per tal de dividir correctament els diferents àmbits de l'aplicació, hi ha una classe DAO per a cada classe Controladora que es corresponen amb les classes Model:

- Usuari
- Grup
- Nota
- Llista
- ItemLlista (només model)

L'accés a Base de Dades es realitza directament amb el controlador JDBC sense cap *framework* per sobre. Com s'ha comentat anteriorment en la memòria, des del codi només s'invoquen sentències SELECT i crides a *Stored Procedures.* Les sentències UPDATE, INSERT i DELETE només es troben dins del codi dels procediments.
Així doncs, per recuperar el detall d'una nota, es fa servir el següent mètode:

```
      public static Nota getNotaPorId(int idNota) {
            String sql=String.format("select titulo,contenido,fecha_modificacion
from notas " +
                        "where activo=1 and id_nota=%s;",idNota);
            try {
                  ResultSet rs = MysqlManager.getInstance().query(sql);
                  if(rs.next()) {
                        String titulo=rs.getString("titulo");
                        String contenido=rs.getString("contenido");
                        Date fecha=rs.getDate("fecha_modificacion");
                        Time hora=rs.getTime("fecha_modificacion");
                        Nota nota= new Nota(idNota, titulo, contenido, fecha,
hora);
                        return nota;

                  } else {
                        return null;
                  }
            } catch (SQLException e) {
                  logger.error("Error: ",e);
                  return null;
            }
      }
```

De manera genèrica, cada mètode d'una classe DAO que recupera dades segueix els següents passos:

- Definició de la consulta (*query*), deixant-la en funció dels paràmetres d'entrada definits per a la funció.
- Invocació de la consulta a base dades i recuperació dels resultats.
- Recuperar els registres i retornar el resultats:
    - En cas de que s'esperi un únic registre, es recuperen els camps i es crea l'objecte que serà retornat per la funció.
    - En cas que s'esperi un o més registre, es recuperen els camps, es crea l'objecte i s'afegeix a la col·lecció d'objectes que serà retornar per la funció.

En el següent exemple, es realitza una invocació d'un procediment emmagatzemat per a crear un nou grup:

```
	public static String crearGrupo(int idUsuario, String nombreGrupo) {
		String status;
		String	sql=String.format("CALL	grupos_crear('%s','%s');",
idUsuario,nombreGrupo);
		try {
			MysqlManager.getInstance().execute(sql);
			status="El grupo se ha creado correctamente";
		}
		catch(Exception e) {
			logger.error("Error: ",e);
			status="Error: Se ha producido un error al crear el grupo";
		}
		return status;
	}
```

Com es pot observar, el fet de fer servir *Stored Procedures* simplifica molt el codi i el fa més llegible, desacoblant molt bé el codi que executa el servidor amb el que s'executa base de dades. A més, en cas de voler modificar la lògica del procediment, sempre i quant no impliqui modificació en els paràmetres, es pot realitzar aquesta modificació sense la necessitat d'haver de recompilar i desplegar tot el codi de servidor.

De manera genèrica en l'aplicació del servidor, per a executar un procediment, simplement cal declarar la sentència, deixant-la en funció dels paràmetres d'entrada de la funció, invocar-lo en la base de dades i controlar les excepcions en cas de produir-se.

3.1.3. Extractes de codi del front-end

La última capa de l'arquitectura és el frontal, on els usuaris poden visualitzar les dades i modificar-les. El frontal està desenvolupat en HTML5, CSS i JavaScript juntament amb el framework AngularJS, que facilita el desenvolupament de l'aplicació client seguint el patró MVC.

Les parts clau del codi del *front-end* estan vinculades amb AngularJS i els seus principals elements:

- Aplicació AngularJS
- Controladors
- Serveis
- Events

Aplicació AngularJS

El primer pas en el desenvolupament d'una aplicació amb AngularJS és definir l'aplicació. Les aplicacions en AngularJS estan pensades per ser SPA (*Single Page Application*), per tant, l'aplicació haurà de contenir tot l'àmbit de l'aplicació.

Per definir l'àmbit de l'aplicació, primer cal indicar-lo al codi HTML, per mitjà de la directiva *ng-app*:

```
<html lang="es" ng-app="listoplan">
      …
</html>
```

Tot el que estigui dins del tag <html> formarà part de l'aplicació AngularJS "listoplan".

La segon part es definir la aplicació en el codi JavaScript:

```
var listoplan= angular.module('listoplan',[]);
```

Controladors

Els controladors són els elements clau d'una aplicació en AngularJS. Els controladors defineixen les operacions de l'aplicació, és a dir, implementen el seu comportament. Els controladors contenen un conjunt d'atributs i mètodes i aquests són invocats per mitjà d'events d'AngularJS, equivalents als events habituals de JavaScript (polsar un botó, carregar una pàgina, polsar una tecla,...).

Un exemple de controlador és el que es fa servir per implementar la compartició de llistes:

```
listoplan.controller('listasCompartidasController',
function($scope,$http,InfoUsuario,$rootScope){
      $scope.buscarListasCompartidas=function(){
          $http({
              method: 'GET',
              url: url+"listas/listas_compartidas?filtro="+$scope.filtro,
              headers: {"token":InfoUsuario.token}
            }).then(function successCallback(response) {
                  $scope.resultadolistas=response.data;
            }, function errorCallback(response) {
                   console.log(response.data.status);
            });
      };
      $scope.copiarLista= function(id_grupo, id_lista){
            var ambito;
            if(id_grupo==0){
                  ambito="USUARIO"
            }else{
                  ambito="GRUPO"
            }
            $http({
              method: 'POST',
              url: url+"listas/duplicacion_lista",
              headers: {"token":InfoUsuario.token},
              data:{
                    "idLista":id_lista.toString(),
                    "id": id_grupo.toString(),
                    "ambito":ambito
              }
            }).then(function successCallback(response) {
                  $("#ok_grupo").show();
                        $scope.ok_msg="La lista se ha copiado correctamente";
                        $rootScope.$broadcast('modificarListadoListas',
id_grupo);
              }, function errorCallback(response) {
                              $("#error_grupo").show();
                              $scope.error_msg="Se ha producido un error al copiar
la lista";
                              console.log(response.data.status);
              });
      }
});
```

Aquest controlador implementa dos mètodes:

- buscarListasCompartidas: Aquest mètode fa una petició al servidor per que li retorni les llistes compartides que encaixen amb el filtre indicat per l'usuari
- copiarLista: Aquest mètode es fa servir per copiar una llista compartida per un altre usuari.

En la majoria dels casos, amb aquesta arquitectura, cada controlador agrupa una sèrie de mètodes que tenen una funcionalitat vinculada amb l'àmbit del controlador i, cada mètode, s'acaba traduint en una crida al servidor per obtenir o enviar dades.

Aquests mètodes, són cridats per mitjà d'events, que s'han de definir al codi HTML. Pels dos mètodes anteriors, els events que els invoquen són els següents:
Per buscar una llista compartida:

```
<input class="form-control" placeholder="Filtro" id="filtro" ng-model="filtro"
ng-value="filtro" ng-change="buscarListasCompartidas()"></input>
```

El mètode buscarListasCompartidas s'invoca cada cop que el camp de text "filtre" es modifica per part de l'usuari (event *ng-change*).

Per copiar una llista compartida:

```
<button type="button" class="btn btn-warning" ng-click="copiarLista(id_grupo,
lista.idLista)"><i class="fa fa-copy fa-fw"></i> Copiar lista</button>
```

En aquest cas, la funció copiarLista s'executa con es polsa sobre el botó de copiar llista (event *ng-click*).

Un altre element que s'ha de tenir en compte a l'hora de treballar amb els controladors és l'àmbit d'actuació que tindrà dintre de l'estructura de l'arbre HTML (DOM). Els controladors tenen un àmbit acotat que engloba els elements continguts dins d'un element HTML, normalment un . En l'exemple anterior, l'àmbit d'aplicació es defineix amb la següent etiqueta amb la directiva *ng-controller*:

```
<div class="row" ng-controller="listasCompartidasController">
      …
</div>
```

Aquesta es la manera com es vincula el codi d'un controlador amb el codi HTML de l'aplicació. L'àmbit del controlador és important ja que dins d'aquest àmbit podem mostrar i recuperar elements de l'scope del controlador. L'scope és un dels pilars en que es fonamenta una aplicació AngularJS i consisteix bàsicament en el conjunt d'atributs i mètodes d'un controlador que poden ser invocats o accedits des del codi HTML. Això ens permet mostrar per pantalla els atributs definits en el controlador. Per exemple, en el codi del controlador anterior es defineix l'atribut "resultadolistas" que conté les dades recuperades del servidor per el filtre indicat per l'usuari.

```
$scope.resultadolistas=response.data;
```

Aquest atribut es pot recuperar i mostrar directament en el codi HTML:

```
<div ng-repeat="lista in resultadolistas">
      <div><b>Nombre: </b>{{lista.nombre}}</div>
      <div><b>Descripción: </b>{{lista.descripcion}}</div>
      <div><b>Tipo de lista: </b>{{lista.tipoLista}}</div>
      …
</div>
```

La directiva *ng-repeat* permet repetir tot el fragment de codi contingut dins de l'etiqueta per cada element que contingui la llista. Per mitjà de la sintaxi {{variable}} es pot mostrar el contingut de cada variable del model en la pantalla.

Equivalentment, és pot recuperar informació dintre de l'àmbit del controlador en la pantalla des del codi JavaScript del controlador. Per exemple, el següent element del formulari:

```
<input class="form-control" placeholder="Filtro" id="filtro" ng-model="filtro"
ng-value="filtro" ng-change="buscarListasCompartidas()"></input>
```

Es pot recuperar en el codi per mitjà de l'atribut *$scope*:

```
url: url+"listas/listas_compartidas?filtro="+$scope.filtro
```

Com es pot observar, el contingut del camp de text identificat amb l'atribut *ng-value=filtro* es pot recuperar al controlador per mitjà de l'atribut *$scope.filtro*.

Serveis

Els serveis són elements d'AngularJS que poden ser accedits des de qualsevol controlador. Conté una sèrie d'atributs i mètodes independents de la lògica del controlador i permet desacoblar i reutilitzar aquestes funcionalitats.

En l'aplicació que s'ha desenvolupat s'han definit dos serveis: un que conté la informació de l'usuari que està fent servir l'aplicació i un altre que conté la informació del grup que està consultant:

```
listoplan.service('InfoUsuario', function() {
      this.nombre;
      this.apellido;
      this.id_usuario;
      this.token;
})
listoplan.service('InfoGrupo', function() {
      this.infoGrupo;
      this.miembros;
})
```

Per poder accedir i/o modificar els atributs de cada servei des d'un controlador, és necessari injectar-li la dependència en la definició del controlador. En l'exemple següent es pot observar com s'injecta aquesta dependència i com es fa ús del serveis dins del controlador:

```
listoplan.controller('datosUsuarioController',
function($scope,$http,InfoUsuario) {
      token=localStorage.getItem("token");
      if(token==null){
            console.log("Token no disponible");
            window.location.replace("login.html");
      }
      else{
            console.log(token);
            InfoUsuario.token=token;
      }

    $http({
       method: 'GET',
       url: url+"usuarios/informacion_usuario/0",
       headers: {"token":token},
     }).then(function successCallback(response) {
            InfoUsuario.nombre=response.data.nombre;
            InfoUsuario.apellido=response.data.apellido;
            InfoUsuario.id_usuario=response.data.id_usuario;
            $scope.nombre=InfoUsuario.nombre;
            $scope.apellido=InfoUsuario.apellido;
            $scope.id_usuario=InfoUsuario.id_usuario;
       }, function errorCallback(response) {
                  console.log(response.data.status);
                  localStorage.removeItem("token");
            window.location.replace("login.html");
       });
});
```

Events

Els events són un dels altres elements claus de les aplicacions en AngularJS. Es poden diferenciar en dos tipus, els que proporciona el propi *framework* i que són equivalents en molts casos als que proporciona JavaScript per defecte i els definits per el desenvolupadors per comunicar controladors.

Els primers ja han sigut introduïts en els exemples anteriors i són molt simples, consisteix simplement en vincular un mètode amb una etiqueta i un event, per tal que aquest mètode s'executi quan es produeixi l'event. És important destacar que la etiqueta ha d'estar dins de l'àmbit del controlador en l'arbre DOM, sinó el mètode mai s'invocarà. De manera genèrica els events es defineixen com:

```
<div ng-controller="controlador1">
      <tag ng-event="metode1()"></tag>
</div>
```

Per exemple, per invocar la creació d'una nova llista:

```
<div class="row" ng-controller="listaController">
      …
      <button type="button" class="btn btn-primary" id="accion_modal" ng-
click="guardarLista(id_lista,id_grupo)"><i class="fa fa-save fa-fw"></i>
Guardar lista</button>
      …
</div>
```

Els events més significatius que s'han fet servir en el desenvolupament de l'aplicació són:

- ng-click: Al polsar sobre un botó o enllaç
- ng-change: Al modificar el text d'un camp de text
- ng-blur: Quan es treu el focus d'un camp de text
- init: No es ben bé un event però executa el mètode al carregar-se l'element a la pàgina. És equivalent al event *onLoad* de JavaScript.

L'altre tipus d'event que es poden fer servir en aplicacions AngularJS són aquells definits pels desenvolupadors i que permeten comunicar dos controlador. Aquests tipus d'event són molt útils ja que permeten invocar mètodes de controladors diferents que tenen un àmbit diferenciat. L'ús més freqüent que s'ha fet d'aquests events personalitzats en l'aplicació és per refrescar el contingut després de realitza una acció:

Si el controlador A s'encarrega de recuperar les notes d'un usuari i el controlador B s'encarrega de crear, modificar i desactivar les notes, un cop que s'ha executat un mètode del controlador B, aquest pot generar un event que estarà escoltant el controlador A perquè, quan es produeixi aquest event, es refresquin les dades.

A mode d'exemple, a continuació es mostra el codi que s'ha fet servir per implementar la casuística anterior.

En un controlador es genera l'event:

```
$rootScope.$broadcast('modificarListadoNotas', id_grupo);
```

Per mitjà del mètode *broadcast* es propaga l'event "modificarListadoNotas" que té com a paràmetre l'id del grup.

L'altre controlador està escoltant per detectar quan es produeixi l'event i realitza una acció:

```
$rootScope.$on('modificarListadoNotas', function(event, data) {
$scope.getNotasUsuario(data)});
```

Per mitjà del mètode *on,* el controlador espera a que es produeixi l'event "modificarListadoNotas" i executa el mètode *getNotasUsuario* del mateix controlador amb els paràmetres rebuts amb l'event.

Hi ha dos tipus d'*scope* sobre els quals es poden definir els events, que funcionen de manera equivalent:

- *$rootScope:* quan l'event es propaga entre controladors que són independents, és a dir, que no tenen cap tipus de relació pare-fill.
- *$scope:* quan l'event es propaga entre controladors que tenen relació pare-fill.

Aquesta relació pare-fill es produeix segons un controlador contingui o no un altre controlador quan és defineixen en el DOM de l'aplicació. Dos o més controladors tindran una relació para fill si es defineixen de la següent manera en el DOM:

```
<div ng-controller="controladorPare">
     ...
     <div ng-controller="controladorFill">
          ...
     </div>
     ...
</div>
```

Mentre que seran independents si es defineixen de la següent manera en el DOM:

```
<div ng-controller="controlador-1">
     ...
</div>
<div ng-controller="controlador-2">
     ...
</div>
```

3.2. Seguretat

Les decisions de seguretat que s'han pres en el disseny de l'aplicació es poden dividir en dos blocs:

- Seguretat en el disseny de l'aplicació
- Seguretat en la configuració dels servidors (fora d'abast)

En el primer bloc, les decisions de disseny relacionades en la seguretat han de tenir-se en compte per tal de no exposar informació confidencial dels usuaris a altres usuaris o terceres persones, incloent-hi els propis administradors de l'aplicació. Les principals mesures de seguretat que s'han pres en el desenvolupament són les següents:

SALT:

Aquest mètode [8] té a veure amb la forma amb la que s'encripta i s'emmagatzema el password d'usuari a la base de dades. Aquest mètode afegeix seguretat extra a l'hora d'emmagatzemar el password fent servir una tècnica de xifrat. La tècnica consisteix en concatenar una sèrie de caràcters aleatoris al password (anomenat "salt") definit per l'usuari abans d'introduir-lo a la funció de xifrat. En base de dades, s'haurà d'emmagatzemar tant el password amb el salt xifrat com el propi salt, de manera que es pugui recuperar a l'hora de realitzar l'autenticació.

El que es persegueix amb aquest mètode és no emmagatzemar en cap cas el password de l'usuari xifrat amb un algoritme estàndard directament, per tal que, en cas de que algú pugui accedir a aquest password no el pugui desxifrar per mitjà de tècniques de diccionaris o força bruta.

En la aplicació, aquesta tècnica es realitza en un procediment emmagatzemat en base de dades en el moment de crear un nou usuari:

```
select SUBSTRING(MD5(RAND()), 1, 10) into V_SALT;
select SHA2(CONCAT(V_SALT,P_CONTRASENA_ORIG),256) into V_CONTRASENA;
```

Com es pot veure, es concatena un Salt de 10 caràcters al password original i aquest es xifrat per mitjà de l'algoritme SHA-256. Ambdós valors es guarden posteriorment en dos camps diferents del mateix registre en la taula d'usuaris en base dades.

Ús de elements confidencials en la URL

Aquesta decisió de disseny bàsica és molt simple i consisteix simplement en la no exposició d'elements confidencials com a variables en la URL (usuaris i contrasenyes, entre d'altres). Al tractar-se d'una API Rest, hi ha una sèrie de variables que s'han d'introduir en la URL per tal de poder recuperar els recursos necessaris. Aquesta tècnica consisteix simplement en que tot aquesta informació que hagi de ser confidencial ha d'anar en el cos d'una petició i en cap cas en la URL, fent així que en cap cas estigui exposada a terceres persones.

Validació del Token en cada petició i control de la propietat de cada element de l'aplicació

Quan es desenvolupa una API Rest, aquesta no ha d'emmagatzemar informació de la sessió en el servidor. Per mantenir i simular aquesta sessió es fa servir un Token que el servidor genera i envia a l'usuari un cop aquest s'ha autenticat. Aquest Token conté certa informació de l'usuari que serveix per identificar-lo (ID, email,...) i a més té un període de validesa i una paraula de pas única. Per tal de garantir que el Token es vàlid i es correspon amb l'usuari en cada petició s'ha de realitzar la validació d'aquest Token per tal de garantir que la "sessió" es correcta i poder recuperar la informació de l'usuari que fa cada petició.
Per altra banda, es necessari que a l'hora d'accedir a un recurs (nota o llista) es garanteixi que l'usuari que fa la petició és realment propietari, ja sigui perquè el recurs sigui seu o bé per que el recurs es d'un grup del qual forma part. L'objectiu es garantir que cada usuari pot accedir només a aquells recursos que li pertoquen (**autorització**).

Comunicació xifrada extrem a extrem

Per l'altre costat es necessari realitzar una configuració sobre els servidors de l'aplicació per tal que es faci servir comunicació xifrada extrem a extrem (SSH/HTTPS) [9], per tal de garantir que la informació que viatja per la xarxa no pot ser interpretada per cap tercera persona en cap d'interceptar-se la informació.
Per mitjà de criptografia de clau pública s'aconsegueix garantir que cap intrús accedeix a la informació que viatja entre el client i el servidor. Aquesta tècnica és imprescindible per garantir que les mesures de seguretat implementades en el disseny de l'aplicació funcionen correctament.

3.3. Tests

En aquest apartat es defineix tot el conjunt de casos de prova per a la aplicació de manera global (no per capes). Aquests casos de proba es divideixen en els següents 4 àmbits:

- Usuaris
- Grups
- Notes
- Llistes

L'objectiu és definir un pla de proves el suficientment acurat com per garantir que la aplicació funciona correctament i, funcionalment, s'adapta correctament als requisits plantejats per la aplicació.

3.3.1. Definició dels casos de proba

A continuació es detalla les taules amb la definició de cada cas de prova on, per a cada ítem, es defineixen els següents elements:

- Identificador del cas de prova
- Descripció del cas de prova
- Condicions prèvies per executar el cas de prova
- Resultat esperat del cas de prova

USUARI:

Identificador	Descripció	Cond. Prèvies	Resultat esperat
USR-001	Crear un nou usuari que no està registrat prèviament	Accedir a la pantalla de login	El nou usuari es crea correctament
USR-002	Intentar crear un usuari que ja ha estat creat prèviament	Haver creat un usuari prèviament	L'aplicació mostra un error indicant que l'usuari ja existeix
USR-003	Introduir dos contrasenyes diferents en els camps de validació de contrasenya	Accedir a la pantalla de login	L'aplicació mostra un error indicant que les contrasenyes introduïdes no coincideixen
USR-004	Introduir un e-mail que no segueix l'estructura d'e-mail	Accedir a la pantalla de login	L'aplicació mostra un error indicant que el e-mail no té el format correcte
USR-005	Fer login amb usuari/contrasenya correctes	Haver creat un usuari prèviament	El login es produeix correctament i l'aplicació redirigeix a la pantalla de dashboard principal
USR-007	Fer login amb usuari/contrasenya incorrectes	Haver creat un usuari prèviament	L'aplicació indica que la combinació login/password és incorrecta
USR-008	Tancar la sessió de l'usuari	Haver fet login prèviament	La sessió es tanca i l'aplicació redirigeix a la pantalla de login
USR-009	Accedir a l'aplicació un cop el token ha caducat	Haver fet login prèviament una setmana abans	La sessió es destrueix i l'aplicació mostra la pantalla de login

GRUP:

Identificador	Descripció	Cond. Prèvies	Resultat esperat
GRP-001	Crear un nou grup introduint totes les dades	Haver fet login prèviament	El grup es crea correctament
GRP-002	Crear un nou grup deixant camps sense omplir	Haver fet login prèviament	L'aplicació mostra un error amb els camps que s'han d'omplir obligatòriament
GRP-003	Afegir un nou usuari al grup	Haver fet login prèviament i haver creat un grup	L'usuari s'afegeix al grup correctament i hi pot accedir als ítems que contingui
GRP-004	Cercar usuaris que no existeixen en l'aplicació	Haver fet login prèviament i haver creat un grup	La pantalla cercador d'usuaris no mostra cap resultat
GRP-005	Canviar el nom a un grup ja existent	Haver fet login prèviament i haver creat un grup	El nom es canvia correctament i el grup es mostra amb el nou nom
GRP-006	Visualitzar els grups vinculats a un usuari	Haver fet login prèviament	En el menú lateral esquerra apareixen tots els grups de l'usuari
GRP-007	Donar de baixa un grup	Haver fet login prèviament i haver creat un grup	El grup deixa d'estar disponible i visible per a tots els usuaris
GRP-008	Afegir un nou usuari com administrador al grup	Haver fet login prèviament i haver creat un grup	L'usuari s'afegeix correctament i aquest té privilegis d'administració
GRP-009	Desvincular un usuari d'un grup sent administrador	Haver fet login prèviament, haver creat un grup i ser admin.	L'usuari es desvincula del grup i aquest deixa de estar disponible per a l'usuari
GRP-010	Intentar desvincular un usuari d'un grup sense ser administrador	Haver fet login prèviament i pertànyer a un grup sense ser admin.	A l'usuari no l'apareix la opció de desvincular usuari en el grup

NOTES:

Identificador	Descripció	Cond. Prèvies	Resultat esperat
NOT-001	Crear una nova nota d'usuari	Haver fet login prèviament	La nota es crea correctament
NOT-002	Modificar una nota d'usuari	Haver fet login prèviament i haver creat una nota	La nota es modifica correctament
NOT-003	Eliminar una nota d'usuari	Haver fet login prèviament i haver creat una nota	La nota s'elimina correctament i deixa d'estar disponible
NOT-004	Crear una nova nota en un grup	Haver fet login prèviament i estar vinculat a un grup	La nota es crea correctament al grup
NOT-005	Modificar una nota en un grup	Haver fet login prèviament, haver creat una nota i estar vinculat a un grup	La nota es modifica correctament al grup
NOT-006	Eliminar una nota en un grup	Haver fet login prèviament, haver creat una nota i estar vinculat a un grup	La nota s'elimina correctament i deixa d'estar disponible al grup
NOT-007	Intentar crear una nova nota sense títol	Haver fet login prèviament	L'aplicació mostra un error indicant que el títol és obligatori i no la guarda
NOT-008	Intentar modificar una nota i deixar-la sense títol	Haver fet login prèviament i haver creat una nota	L'aplicació mostra un error indicant que el títol és obligatori i no la guarda
NOT-009	Visualitzar totes les notes d'usuari (llistat)	Haver fet login prèviament i haver creat almenys una nota personal	Es mostra el llistat amb totes les notes personals
NOT-010	Visualitzar totes les notes d'un grup (llistat)	Haver fet login prèviament, pertànyer almenys a un grup i haver creat almenys una nota al grup	Es mostra el llistat amb totes les notes del grup
NOT-011	Visualitzar el detall d'una nota	Haver fet lògin prèviament i haver creat una nota	Es mostra una vista modal amb tota la informació de la nota

LLISTES:

Identificador	Descripció	Cond. Prèvies	Resultat esperat
LST-001	Crear una nova llista d'usuari	Haver fet lògin prèviament	La llista es crea correctament
LST-002	Modificar una llista d'usuari	Haver fet lògin prèviament i haver creat una llista personal	La llista es modifica correctament
LST-003	Desactivar una llista d'usuari	Haver fet lògin prèviament i haver creat una llista personal	La llista s'elimina correctament i deixa de ser visible per l'usuari
LST-004	Afegir un nou ítem en una llista d'usuari	Haver fet lògin prèviament i haver creat una llista personal	L'ítem s'afegeix correctament
LST-005	Modificar un ítem en una llista d'usuari	Haver fet lògin prèviament i haver creat una llista personal	L'ítem es modifica correctament
LST-006	Eliminar un ítem en una llista d'usuari	Haver fet lògin prèviament i haver creat una llista personal	L'ítem s'elimina i deixa de mostrar-se en la llista
LST-007	Crear una nova llista de grup	Haver fet lògin prèviament i pertànyer a un grup	La llista es crea correctament
LST-008	Modificar una llista de grup	Haver fet lògin prèviament, pertànyer a un grup i haver creat una llista en el grup	La llista es modifica correctament
LST-009	Desactivar una llista de grup	Haver fet lògin prèviament, pertànyer a un grup i haver creat una llista en el grup	La llista s'elimina correctament i deixa de ser visible per l'usuari
LST-010	Afegir un nou ítem en una llista de grup	Haver fet lògin prèviament, pertànyer a un grup i haver creat una llista en el grup	L'ítem s'afegeix correctament
LST-011	Modificar un ítem en una llista de grup	Haver fet lògin prèviament, pertànyer a un grup i haver creat una llista en el grup	L'ítem es modifica correctament
LST-012	Eliminar un ítem en una llista de grup	Haver fet lògin prèviament, pertànyer a un grup i haver creat una llista en el grup	L'ítem s'elimina i deixa de mostrar-se en la llista
LST-013	Visualitzar el llistat de llistes d'usuari	Haver fet lògin prèviament i haver creat almenys una llista	Es mostra un llistat amb totes les llistes personals de l'usuari
LST-014	Visualitzar el llistat de vistes d'un grup	Haver fet lògin prèviament, pertànyer a un grup i haver creat almenys una llista en el grup	Es mostra un llistat amb totes les llistes del grup que està consultant l'usuari
LST-015	Cercar una llista compartida	Haver fet lògin prèviament	En el cercador apareix de manera

			dinàmica el conjunt de llistes compartides disponibles segons el filtre de l'usuari
LST-016	Copiar una llista compartida a les llistes d'usuari	Haver fet lògin prèviament	La llista es copia al conjunt de llistes d'usuari
LST-017	Copiar una llista compartida a les llistes de grup	Haver fet lògin prèviament i pertànyer a un grup	La llista es copia al conjunt de llistes del grup
LST-018	Compartir una llista perquè estigui disponible per altres usuaris	Haver fet lògin prèviament i haver creat almenys una llista	La llista passa a estar compartida i es accessible per ser copiada per la resta d'usuaris.

3.3.2. Resultats dels casos de prova

Un cop definits els casos de prova, el següent pas és executar-los de manera independent sobre la aplicació. A continuació es mostra una taula amb els resultats obtinguts.

Identificador	Resultat
USR-001	Correcte
USR-002	Correcte
USR-003	Correcte
USR-004	Correcte
USR-005	Correcte
USR-007	Correcte
USR-008	Correcte
USR-009	Correcte
GRP-001	Correcte
GRP-002	Correcte
GRP-003	Correcte
GRP-004	Correcte
GRP-005	Correcte
GRP-006	Correcte
GRP-007	Correcte
GRP-008	Correcte
GRP-009	Correcte
GRP-010	Correcte
NOT-001	Correcte
NOT-002	Correcte
NOT-003	Correcte
NOT-004	Correcte
NOT-005	Correcte
NOT-006	Correcte
NOT-007	Correcte
NOT-008	Correcte
NOT-009	Correcte
NOT-010	Correcte
NOT-011	Correcte
LST-001	Correcte
LST-002	Correcte
LST-003	Correcte
LST-004	Correcte
LST-005	Correcte
LST-006	Correcte
LST-007	Correcte
LST-008	Correcte
LST-009	Correcte
LST-010	Correcte
LST-011	Correcte
LST-012	Correcte
LST-013	Correcte
LST-014	Correcte
LST-015	Correcte
LST-016	Correcte
LST-017	Correcte
LST-018	Correcte

Com es pot observar en la taula anterior, per a aquesta primera versió totes les funcionalitats plantejades han sigut validades correctament.

3.4. Versionat i control de versions

En la taula següent es mostra un seguiment de les versions de l'aplicació i una estimació de les versions que es duran a terme durant el desenvolupament del projecte:

Versió	Estat	Descripció
0.1	Realitzada	Primera versió de la API REST sense frontal
0.2	Realitzada i entregada (PAC2)	Correcció d'errors de la versió anterior fent probes integrades per mitjà d'un client REST genèric
0.5	Realitzada	Primera versió de l'aplicació integrant la API i una primera versió del frontal
0.6	Realitzada i entregada (PAC3)	Correcció de la versió anterior un cop realitzat les proves integrades de tota la aplicació. També es poden afegir possibles millores.
1.0	Realitzada i entregada (Entrega Final)	Primera versió final de la aplicació, totalment funcional. Abarca l'àmbit del projecte i es l'entregable que es realitzarà del TFM.
Versions posteriors a 1.0	Pendent. Fora d'abast	Correcció de possibles errors, millores i noves funcionalitats que es realitzaran fora de l'abast del TFM un cop l'aplicació estigui en producció.

*Aquesta taula està es sensible de patir variacions durant el desenvolupament del projecte

3.5. Bugs i millores

A continuació es detalla una relació dels bugs detectats i les millores identificades, amb el seu estat i la versió on es preveu la seva correcció o desenvolupament.

Ítem	Tipus	Estat	Versió implantació
Els formularis no es "reseteixen" un cop s'ha tancat la finestra modal.	Bug	Pendent	1.0
Instal·lació certificats SSL.	Millora	Pendent	> 1.0 (fora d'abast)
Els ítems no es refresquen a la pantalla un cop s'han modificat.	Bug	Finalitzat	0.6
El log no mostra la informació clarament	Millora	Finalitzat	0.5
El client no permet connectar-se amb el servidor degut a la política de dominis creuats (*corss domain*)	Bug	Finalitzat	0.6
El servidor no es re-connecta amb la base de dades en cas de caiguda de la connexió o que la sessió hagi caducat	Bug	Finalitzat	0.2
Millores en la gestió d'usuaris: Millora perfil d'usuari, ús d'avatars, afegir funció "cerca d'amics" i afegir comunicació directa entre usuaris.	Millora	Pendent	> 1.0 (fora d'abast)

3.6. Pressupost

En la taula següent es detalla el pressupost equivalent a la realització del projecte. Els costos de desenvolupament, anàlisi, disseny i documentació son inherents a la realització del TFM.

Concepte	Rol/Perfil	Preu/unitat	unitats	Preu total
Gestió del projecte	Project Manager	20€/hora	10	200€
Desenvolupament	Desenvolupador SW	12€/hora	120	1440€
Anàlisi, disseny y documentació	Analista/Arquitecte SW	17€/hora	25	425€
Proves	Enginyer QA	15€/hora	15	225€
Desplegament i configuració al servidor	Tècnic de sistemes	12€/hora	10	120€
Llicències		*0*	*0*	*0€*
Infraestructura (AWS)		*10€/mes*	*3*	*30€*
Total				**2440€**

4. Conclusions

Un cop finalitzat el treball i la memòria, analitzant els resultats obtinguts i comparant-los amb els reptes proposats, puc concloure que, en gran mesura, s'han complert una gran part dels objectius plantejats. És cert que a nivell de l'aplicació desenvolupada hi ha hagut algunes funcionalitats que, finalment, per la càrrega de treball, han hagut de treure's de l'abast, com per exemple la funcionalitat per a grups que es plantejava inicialment de poder fer sondejos i votacions per a poder prendre decisions de forma col·laborativa. No obstant, aquestes petites funcionalitats que s'han quedat en el *backlog* no suposen un impacte gaire gran en el resultat ja que, gracies a l'acompliment del primer gran objectiu, aquestes *features* és redueixen simplement a "posar-hi més hores". El primer gran objectiu al que faig referència i que, per a mi, l'he complert completament, és l'objectiu d'aprendre i desenvolupar-me personalment per a poder desenvolupar aquest treball. A l'inici del treball feia molt d'èmfasi en aprendre tot el necessari per a poder realitzar aquesta aplicació, no tant a nivell de funcionalitats sinó a nivell de coneixement i tecnologia: eines tècniques, llenguatges de programació i *frameworks*, arquitectures d'aplicacions, metodologies i sobretot organització i disciplina personal. Amb això assolit, la resta de coses que es queden pendents és simplement més temps que hi dedicaré segur d'ara en endavant per acabar de polir-la i que pugui ser usada per, quants més usuaris millor.

En aquest sentit, com ja he introduït en el paràgraf anterior el desenvolupament de l'aplicació no acaba aquí. Aquest treball m'ha permès posar-hi els fonaments i aconseguir les eines, els següents passos són acabar de polir el disseny, afegir algunes funcionalitats extres, una millor gestió dels usuaris, una bona configuració de la infraestructura i la realització de proves de rendiment més exhaustives. La idea es que la aplicació pugui ser usable i estigui disponible de manera productiva abans de que acabi l'any.

La planificació seguida per a arribar a aquests resultats ha estat clau. És cert que es pràcticament impossible seguir una planificació al 100% i més quan poden sorgir tants contratemps durant el desenvolupament i es treballen amb tantes incògnites. Penso que en aquest nivell, lo realment important ha estat poder seguir la estructura de la planificació. Ens molts casos no s'ha pogut complir la planificació a nivell de tasca individual, ja sigui bé perquè s'ha estimat per sobre o bé perquè s'ha estimat per sota. No obstant, el que si que s'han complert són les fites de més alt nivell i amb això, tot i que s'han hagut de balancejar les hores de les tasques més petites, a nivell de blocs

(disseny, desenvolupament de cada capa, documentació,...) la planificació si que s'ha pogut seguir correctament i amb això ha sigut possible assolir la gran part dels objectius plantejats a l'inici sense haver d'apurar a última hora.

Finalment, la millor conclusió que es pot extreure d'haver desenvolupat aquest treball és que cada projecte és un món. Mai es pot tenir tot controlat i tot planificat, per tant moltes coses queden en mans de la inspiració. No obstant, tant la experiència com la formació resulten fonamentals per reduir una gran nombre de variables i d'incertesa. Amb aquest treball me'n duc un munt de coneixement i d'eines per al futur però sobretot la confiança personal i la experiència per poder afrontar amb seguretat qualsevol repte en el futur.

5. Glossari

Terme	Definició
MVC	Sigles de Model-Vista-Controlador. És un patró de disseny on es desacoblen la lògica de negoci, les vistes i el controlador que les uneix.
DAO	Sigles de *Data Access Object*. És un patró de disseny on es separa l'accés a base de dades i les seves consultes i accions, de la resta de lògica e negoci
API REST	REST són les sigles de *Representational StateTransfer.* Es tracta d'un tipus d'API estandarditzada de comunicació mitjançant el protocol HTTP on el servidor no manté en cap cas l'estat de la sessió. Això fa que cada petició/resposta sigui independent de cap altre.
AngularJS	Es un *framework* de desenvolupament que facilita la creació de clients que segueixin el model MVC i estiguin totalment desacoblades del servidor.
Spring	Es un *framework* de desenvolupament per Java que facilita la creació d'aplicacions web i el seu desplegament. En aquest cas es fa servir la llibreria *Spring Boot.*
AJAX	Son les sigles de *Asynchronous JavaScript And XML.* És una tecnologia que permet fer crides asíncrones al servidor directament des de JavaScript sense haver de recarregar la pàgina.
Stored Procedure	És una funcionalitat pròpia d'algunes base dades que permet emmagatzemar un conjunt de procediments, que implementen una lògica juntament amb seqüències SQL, de manera que poden ser cridades i executades des de fora del servidor.
POM	Project Object Model. Es un fitxer XML que conté totes les dependències (normalment llibreries) que necessita el projecte per poder executar-se. Això facilita la gestió de les dependències, ja que, utilitzant Maven, es possible descarregar i empaquetar les diferents dependències, fent així transparent la seva gestió.
Maven	És una aplicació que, per mitja dels fitxers POM, facilita la gestió de les dependències d'un projecte.
Client/Servidor	És un tipus d'arquitectura on l'aplicació és distribueix en dos parts: Un servidor que normalment conté la lògica de negoci gestiona l'emmagatzemament i l'accés a les dades i un client que rep, envia i mostra aquestes dades. La responsabilitat, per tant, es divideix entre el client i el servidor, connectats a través de la xarxa.
Política de Domini creuat (cross domain)	La majoria de navegadors, per defecte, no permeten la connexió d'un client (aplicació HTML) amb un servidor d'un domini diferent, això és, un servidor que no tingui la mateixa URL base i el mateix port. Per permetre aquesta comunicació, és necessari que el servidor accepti aquest tipus de peticions per mitjà d'unes capçaleres específiques en la petició/resposta.
DOM	Són les sigles de *Document Object Model* i serveix per a representar documents HTML, XHTML i XML i interaccionar-hi a través d'objectes.

6. Bibliografia

[1] Ramón Rodríguez, José (2016). *Gestió de projectes TI* (UOC) [document inèdit].

[2] Pradel Miquel, Jordi i Raya Martos, Jose (2014). *Enginyeria del programari* (UOC) [document inèdit].

[3] Zapata Lluch, Mónica (2011). *Enginyeria de la usabilitat* (UOC) [document inèdit].

[4] Web: MySQL, documentació oficial. URL: *https://dev.mysql.com/doc/*. Consultada al Març de 2018.

[5] Web: Spring.io. URL: *https://spring.io/guides/gs/rest-service/*. Consultada en Març de 2018.

[6] Web: Angular.js, documentació oficial. URL: *https://docs.angularjs.org/guide/*. Consultada l'Abril de 2018.

[7] Moldes Teo, F. Javier (2011) *Java 7*. Anaya Multimedia

[8] Web Wikipedia (article: Salt). URL: https://es.wikipedia.org/wiki/Sal_(criptograf%C3%ADa). Consultada en Maig de 2018.

[9] James F. Kurose, Keith W. Ross (2012) Computer Networking. Pearson

7. Annexos

7.1. Instruccions instal·lació de la Base de dades

Per instal·lar els objectes en Base de dades és necessari disposar instal·lada una base de dades mySQL (recomanable versió 5.7) i tenir obert el port 3306.
Els passos a seguir són:

1.Crear l'schema "listoplan"
2.Crear l'usuari "listoplan"
3.Assignar a l'usuari, com a mínim, els següents permisos sobre l'schema listoplan:
    -Grants per crear taules i Stored Procedures
    -Grants de SELECT, INSERT, UPDATE i DELETE sobre les taules
    -Grants d'execució sobre els Stored Procedures
4.Sobre un client de mySQL executar l'script ObjectesBDD.sql ubicat en el directori BaseDades

7.2. Instruccions instal·lació del servidor *Back-end*

Per poder executar el servidor es necessari tenir instal·lat Java 8 en el servidor i tenir obert el port 8080.
Els passos a seguir són:

1.En el directori Servidor, modificar el fitxer "configuration.properties" amb la informació del servidor de base de dades configurat en el pas anterior.
2.Executar la comanda:

```
java –jar target/lisToPlan-Server.jar configuration.properties
```

Amb això s'aixeca automàticament l'aplicació i el servidor tomcat embedit en l'arxiu jar.

Per facilitar l'administració és recomanable crear un directori amb l'arxiu *jar* i crear un script *Linux-shell* que contingui la comanda anterior.

7.3. Instruccions instal·lació del servidor Front-end

Per a poder executar el client és necessari tenir instal·lat el servidor web Apache i tenir el port 80 obert.
Els passos a seguir són:

1. Sobre el directori 'www' (varia segons la instal·lació d'Apache) copiar els fitxers del directori Frontal/Web.
2. Arrencar el servidor web Apache.
3. Validar que la instal·lació es correcta accedint amb el navegador a la IP del servidor web Apache i veure que l'aplicació funciona correctament.