



## MY PET CARE

### SPRINT – 2



Enllaços als  
recursos:

Project record track



Taiga



Repositoris:

Android

Servidor

Lliberies

Cognom, Nom	Rol	Contacte (@est.fib.upc.edu)	Drive (@gmail.com)	Taiga i GitHub
Campos, Xavier	Developer	xavier.campos.diaz	xavicampos99	xavier-campos
Catalan, Oriol	Developer	oriol.catalan.fernandez	catalan.oriol	oricat8
Clemente, Daniel	Developer	daniel.clemente.marin	daniclemente44	danicm47
Del Rey, Santiago	Developer	santiago.del.rey	santi.delrey	santidrrj
Hernando, Enric	Developer	enric.hernando	enrichernandopuerta	enrichp
Pinto, Albert	Developer	albert.pinto	apintogil	AlbertPG
Simó, Marc	Master	marc.simo	sgmarcsg	marcIII
Trius, Álvaro	Developer	alvaro.trius	alvarotrius94	AlvarTB

**Assignatura:** Projecte d'Enginyeria del Software

**Professor:** Silverio Martínez

**Curs:** 2019-2020 QP

**Facultat d'Informàtica de Barcelona**

**Universitat Politècnica de Catalunya**

## ÍNDIX DE CONTINGUTS

1. Introducció.....	1
1.1. Resum executiu del sprint .....	1
1.2. Breu descripció del treball individual.....	2
1.2.1. Albert Pinto i Gil .....	2
1.2.2. Álvaro Trius Béjar.....	2
1.2.3. Daniel Clemente Marín.....	2
1.2.4. Enric Hernando Puerta.....	3
1.2.5. Marc Simó Guzmán .....	3
1.2.6. Oriol Catalán Fernández .....	4
1.2.7. Santiago Del Rey Juárez.....	4
1.2.8. Xavier Campos Díaz .....	4
1.3. Avaluació dels companys (Actualitzat) .....	5
1.4. Sprint master report (Nou) .....	6
2. Requisits .....	9
2.1. Product Backlog (Actualitzat).....	9
2.1.1. Gestió de mascotes i usuaris (Actualitzat) .....	9
2.1.2. Salut de la mascota (Actualitzat) .....	10
2.1.3. Assistència a l'exercici .....	10
2.1.4. Establiments especialitzats .....	10
2.1.5. Interacció en comunitats temàtiques (Actualitzat) .....	11
2.1.6. Interacció amb xarxes socials (Actualitzat) .....	11
2.1.7. Interacció amb el calendari (Actualitzat) .....	12
2.1.8. Gestió de medalles i recompenses .....	12
2.1.9. Aspectes tecnològics .....	12
2.2. Requisits no funcionals (Actualitzat) .....	13

---

2.2.1. Rapidesa .....	13
2.2.2. Disponibilitat .....	14
2.2.3. Confidencialitat .....	14
2.2.4. Abstracció i Reutilització (Nou) .....	15
2.3. Tractament d'aspectes transversals (Nou) .....	16
2.4. Serveis de tercers (Actualitzat) .....	17
2.4.1. Firebase .....	17
2.4.2. Google Calendar (Nou) .....	18
2.4.3. OAuth2 (Nou) .....	19
3. Cerimònies àgils (Actualitzat) .....	20
3.1. Sprint Planning (Nou) .....	20
3.2. Sprint review (Nou) .....	21
3.2.1. Què s'ha pogut entregar? .....	21
3.2.2. Què no s'ha pogut entregar? .....	21
3.2.3. Per quin motiu? .....	22
3.3. Sprint retrospective (Nou) .....	22
3.3.1. Reunió General .....	22
3.3.2. Reunió Front .....	23
3.3.3. Reunió Back .....	23
3.3.4. Què s'ha fet bé i es mantindrà? .....	24
3.4. Gràfiques (Actualitzat) .....	25
3.4.1. Sprint burndown .....	25
3.4.2. Release burndown .....	25
3.4.3. Velocitat de l'equip .....	26
3.4.4. Dedicació mensual .....	26
4. Metodologia .....	27
4.1. Metodologia àgil .....	27

---

4.2. Definició de finalització .....	28
4.3. Gestió del projecte.....	28
4.4. Convencions .....	29
4.5. Gitflow.....	31
4.6. Testeig.....	32
4.7. GitHub Actions.....	33
4.8. Dependabot .....	33
4.9. GitGuardian .....	34
4.10. Qualitat del codi (Nou títol, anteriorment anomenat Codacy) .....	34
4.11. Entorns integrats de desenvolupament .....	35
4.11.1. Android Studio.....	35
4.11.2. IntelliJ IDEA.....	36
4.12. Comunicació .....	36
4.13. Gestió de bugs (Nou).....	37
5. Descripció tècnica .....	38
5.1. Concepció general de l'arquitectura .....	38
5.2. Diagrames arquitectònics .....	39
5.3. Patrons aplicats .....	41
5.3.1. Factoria abstracta .....	41
5.3.2. Factoria simple.....	42
5.3.3. Adaptador.....	42
5.3.4. Model, vista, controlador .....	43
5.3.5. DAO .....	43
5.3.6. Service Locator .....	44
5.3.7. Singletó .....	45
5.3.8. Expert.....	46
5.3.9. Controlador de transacció .....	46

5.3.10. Patró Plantilla (Nou) .....	47
5.3.11. Patró estratègia (Nou) .....	48
5.3.12. Patró observador (Nou) .....	48
5.4. Model conceptual de les dades (Actualitzat) .....	49
5.5. Altres aspectes tecnològics .....	51
5.5.1. Servidors .....	51
5.5.2. Base de dades .....	51
5.5.3. Nombre de llenguatges .....	51
5.5.4. APIs .....	52
5.5.5. Frameworks .....	52
5.5.7. Desplegament .....	52

## ÍNDIX DE FIGURES

Figura 1. Sprint burndown de la segona iteració .....	25
Figura 2. Release burndown .....	25
Figura 3. Representació de la velocitat de desenvolupament de l'equip .....	26
Figura 4. Hores de dedicació mensual del membres de l'equip .....	26
Figura 5. Arquitectura Enterprise Service Bus .....	39
Figura 6. Representació de la capa física .....	40
Figura 7. Diagrama de components .....	40
Figura 8. Exemple del disseny UML d'una factoria abstracta .....	41
Figura 9. Exemple del disseny UML d'una factoria simple .....	42
Figura 10. Exemple del disseny UML d'un adaptador .....	42
Figura 11. Exemple del disseny UML del MVC .....	43
Figura 12. Exemple del disseny UML del DAO .....	44
Figura 13. Exemple del disseny UML del Service Locator .....	45
Figura 14. Representació d'un singletó en UML .....	45

Figura 15. Exemple d'aplicació del patró expert.....	46
Figura 16. Exemple del disseny UML del controlador de transacció.....	46
Figura 17. Patró Plantilla .....	47
Figura 18. Patró estrategia.....	48
Figura 19. Patró observador.....	49
Figura 20. Model de dades final (Original) .....	49
Figura 21. Diagrama de classes final (Nova versió) .....	50
Figura 22. Diagram de classes actual (Original) .....	50
Figura 23. Diagrama de classes actual (Nova versió) .....	50

## ÍNDIX DE TAULES

Taula 1. Valoracions entre companys .....	5
Taula 2. Històries d'usuari del tema: Gestió de mascotes i usuaris .....	9
Taula 3. Històries d'usuari del tema: Salut de la mascota .....	10
Taula 4. Històries d'usuari del tema: Assistència a l'exercici.....	10
Taula 5. Històries d'usuari del tema: Establiments especialitzats .....	10
Taula 6. Històries d'usuari del tema: Interacció en comunitats temàtiques .....	11
Taula 7. Històries d'usuari del tema: Interacció amb xarxes socials .....	11
Taula 8. Històries d'usuari del tema: Interacció amb el calendari.....	12
Taula 9. Històries d'usuari del tema: Gestió de medalles i recompenses .....	12
Taula 10. Històries d'usuari del tema: Aspectes tecnològics.....	12

## 1. Introducció

En aquest apartat farem una introducció a la segona iteració del desenvolupament del projecte My Pet Care.

### 1.1. Resum executiu del sprint

En aquest segon *sprint* del projecte, hem establert uns objectius els quals han estat negociats amb el *product owner*. Aquests consisteixen en la implementació de les imatges de perfil dels usuaris i de les mascotes, la qual va quedar pendent de l'anterior *sprint*, del calendari i els esdeveniments de les mascotes i de la xarxa social pròpia de l'aplicació. L'alt nombre de funcionalitats proposades és degut al temps addicional de la Setmana Santa.

Per tal d'assolir aquests objectius, l'equip de desenvolupament ha decidit mantenir els mateixos equips que en la iteració anterior, tot i que s'ha decidit implementar alguns canvis en la manera sobre com es tracten les històries d'usuari en cadascun. Per una banda, en l'equip de *front*, hem decidit que en parelles es realitzaran de forma sincronitzada diverses històries d'usuari amb algun parentesc. D'aquesta manera, en el cas que es detecti alguna necessitat conjunta en la interfície, per exemple, els dos membres es posen d'acord i implementen conjuntament aquesta part. Per l'altre banda, en l'equip de *back*, en lloc de tenir dos equips per implementar la llibreria i el servidor, cada desenvolupador ha d'implementar la història d'usuari des del servidor fins la llibreria. D'aquesta manera no s'endarrereix la implementació i es pot començar a realitzar la comunicació amb el front.

A la tercera setmana de la iteració, hem hagut d'afegir 11 punts repartits en dues històries d'usuari ja que eren una dependència per realitzar les tasques assignades. Malgrat tot, al final hem aconseguit finalitzar quasi totes les tasques i les restants seran traslladades al principi de la pròxima iteració.



## **1.2. Breu descripció del treball individual**

### **1.2.1. Albert Pinto i Gil**

En aquesta iteració m'he centrat en la implementació de les dades de la salut de la mascota i de la xarxa social en el front. Per una banda, he realitzat un *refactor* de la vista per mostrar la informació de la mascota, la qual va ser dissenyada en la iteració anterior. D'aquesta manera, es pot navegar per diferents pàgines mostrant informació diversa sobre les mascotes. Per tal de proporcionar la informació referent a la salut de la mascota, he optat per implementar des de zero una gràfica de barres amb comportament dinàmic, és a dir, es pot interaccionar amb el component. Per una altre banda, he realitzat la part de la xarxa social al front amb l'ajuda del Xavier Campos. En particular, m'he centrat en la part gràfica i la interacció amb la resta del projecte.

### **1.2.2. Álvaro Trius Béjar**

En aquesta iteració m'he encarregat d'implementar i testejar tot l'apartat de CRUD de medicació amb les seves corresponents llibreries. També he realitzat el treball de revisar i corregir codi al GitHub. Per últim, també he realitzat una part de la documentació.

### **1.2.3. Daniel Clemente Marín**

En aquesta iteració he implementat les notificacions periòdiques de la part del front, dissenyant-ho i implementant-ho. Això m'ha portat més temps del degut ja que ho creia més enrevessat del que era però finalment me n'he sortit.



#### 1.2.4. Enric Hernando Puerta

Durant aquesta segona iteració, he estat en l'equip del front, dissenyant i implementant l'aplicació per Android. Concretament m'he encarregat de dissenyar, juntament amb Daniel Clemente i Albert Pinto, tota la part del calendari, a més de la lògica d'avís personal i generat de manera individual. També he estat el responsable del desenvolupament, interfície i lògica, de modificar el nom d'usuari, del càlcul de kcal aconsellades i de fer *login* i *logout* amb Google i Facebook, aquest primer necessari per a la integració de Google Calendar. Finalment també he creat diferents transaccions mitjançant TDD, he realitzat test amb expresso i he fet una petita part de la documentació.

#### 1.2.5. Marc Simó Guzmán

En aquesta iteració m'he encarregat d'implementar el CRUD dels àpats de les mascotes i la interacció amb el Google Calendar en el servidor i la llibreria. Concretament, per una banda he implementat, documentat i elaborat els tests pels àpats i els seus atributs tant en la llibreria com en el servidor i en l'API, i per l'altra banda m'he centrat en l'aprenentatge del funcionament de les APIs de Google en general, i del Google Calendar en concret, i del funcionament de l'autenticació amb OAuth 2.0 per tal d'accedir al Google Calendar amb un compte de Google, una vegada comprés el funcionament he implementat, documentat i elaborat la interacció amb l'API de Google Calendar en el servidor, en l'API i en la llibreria, que consisteix en un CRUD de Calendar i un CRUD de Event, per tal d'interactuar amb calendaris i events de calendaris. A més, al llarg de tota la iteració, he estat donant suport als meus companys quan sorgien dubtes sobre com realitzar certes tasques, i com a *sprint master*, he distribuït i organitzat la feina entre els meus companys per tal de complir amb el termini d'entrega i m'he encarregat de garantir el compliment de les cerimònies àgils pròpies d'un *sprint*.

### 1.2.6. Oriol Catalán Fernández

En aquesta iteració m'he centrat en la implementació en el *back-end* del CRUD de les dades de salut de les mascotes, incloent els atributs pes, calories totals, calories mitjanes, freqüència de bany, freqüència d'entrenament, entrenament setmanal, patologies i necessitats. He estat treballant també en la implementació d'aquesta part en la llibreria, i també fent els tests corresponents a cada operació. No he tingut massa problemes mes que aprendre la implementació dels CRUDs tant a servidor com a llibreria, i en general, el *sprint*, com ha opinió personal, ha anat bastant bé.

### 1.2.7. Santiago Del Rey Juárez

En aquesta segona iteració he estat a l'equip de *back-end* implementant les noves funcionalitats del servidor i fent algunes actualitzacions en aquest. Concretament, he estat l'encarregat d'actualitzar el sistema de *sign up* per als usuaris, en la part del servidor, a més d'implementar l'actualització de nom d'usuari. A més a més, m'ha estat assignada la implementació de la lògica per a la xarxa social pròpia de la nostra aplicació, és a dir, de la interacció amb els grups temàtics, els seus fòrums i el post de missatges en aquests.

### 1.2.8. Xavier Campos Díaz

En aquest segon *sprint* he estat a l'equip de front implementant les funcionalitats relacionades amb els CRUD dels àpats i de la medicació de les mascotes. A més també m'he encarregat de fer un *refactor* de la informació de la salut de la mascota, així com moltes transaccions relacionades amb les funcions de la comunitat, com ara crear o esborrar un post.

### 1.3. Avaluació dels companys (Actualitzat)

Les valoracions s'han realitzat mitjançant una enquesta realitzada amb Google Forms. Per tal de no condicionar aquestes, els membres no han tingut accés a les valoracions ja realitzades i s'han realitzat de manera anònima.

#### Original

Tots els membres han hagut de valorar als seus companys amb una puntuació d'entre 1 i 10, a més, s'han hagut de valorar amb un 0 a si mateixos per tal de poder realitzar la mitjana de les valoracions de manera correcta.

#### Nova versió

Tots els membres han hagut de valorar als seus companys en una escala de l'1 al 10, on el primer significa que ha tingut una actitud molt negativa envers el treball i l'altre que ha sigut productiu i ha tingut una bona actitud durant el desenvolupament de la iteració. Per tal de realitzar la mitjana de les valoracions de forma correcta, cada membre s'ha hagut de votar a si mateix amb una valoració de 0. Per tant, aquest valor no indica que no s'hagi treballat en la iteració sinó que no es pot valorar a si mateix.

Un cop tancat el període de valoracions, concretament el dia abans de l'entrega d'aquest document, s'han obtingut les mitjanes de totes les notes i s'han fet públiques a tot l'equip.

	<b>Sprint 1</b>	<b>Sprint 2</b>	<b>Sprint 3</b>	<b>Sprint 4</b>	<b>Mitjana</b>
<b>Albert Pinto</b>	9.29	9.14			9.22
<b>Álvaro Trius</b>	6.71	6.00			6.36
<b>Daniel Clemente</b>	7.71	5.57			6.64
<b>Enric Hernando</b>	8.57	8.43			8.50
<b>Marc Simó</b>	9.00	9.00			9.00
<b>Oriol Catalán</b>	7.14	6.71			6.93
<b>Santiago Del Rey</b>	9.57	9.00			9.29
<b>Xavier Campos</b>	8.71	8.43			8.57
<b>Mitjana</b>	8.34	7.79			8.07



Taula 1. Valoracions entre companys

#### 1.4. Sprint master report (Nou)

En aquest segon *sprint* hem continuat amb el desenvolupament del nostre projecte. Tal i com es va acordar amb el *product owner*, en aquest *sprint* hem acabat d'implementar les funcionalitats que no va donar temps d'acabar en l'*sprint* anterior, les quals són la modificació del nom d'usuari i els CRUDs d'imatge de perfil d'usuari i de mascota, i hem implementat les noves funcionalitats de modificar el idioma del sistema, Xarxa Social Interna, organitzada en Grups amb un conjunt de Fòrums en els quals els usuaris poden fer Posts, alguns CRUDs més de les dades alimentàries i de salut de les mascotes, amb gràfiques incloses en l'aplicació, un calendari amb events per cada mascota sincronitzat amb el Google Calendar de l'usuari i les notificaciones i avisos generats per l'aplicació per informar a l'usuari d'esdeveniments.

El primer dia, durant l'*sprint planning*, vam decidir continuar amb la distribució del treball seguida en l'anterior iteració, on quatre membres del equip es dediquen a implementar la part del front-end i els altres quatre el *back-end*, ja que és una distribució que ens va funcionar durant el primer *sprint* i ens permet treballar en paral·lel aprofitant més el temps. A més a més, també vam revisar i refactoritzar la puntuació assignada a les històries d'usuari restants, aplicant la informació i experiència obtinguda en el primer *sprint* per aportar un nou punt de vista.

Aquest mateix dia i seguint els passos del primer *sprint*, es va repartir de manera equitativa la documentació a realitzar per aquest *sprint*, i per tal de tenir-la acabada per la data d'entrega, però sense que suposés massa càrrega de treball de cop, es van dividir els punts a fer en dos conjunts. Així doncs, el primer conjunt de documentació es va realitzar al llarg de la primera i segona setmana i el segon durant la tercera setmana, ja que requeria que l'*sprint* estiguera més avançat; de manera que la documentació quedés completa abans d'acabar la iteració, això si, sense comptar les cerimònies de l'*sprint* que estrictament s'han de realitzar l'últim dia de la iteració, com són l'*sprint review* i *retrospective*.

En aquesta iteració, tot i que en l'anterior iteració no vam poder acabar totes les històries d'usuari, com que teníem una setmana extra degut a la Setmana Santa i que tots els membres estàvem d'acord en mantenir el treball durant aquesta setmana, vam afegir més punts que en l'anterior iteració, tot tenint en compte les històries d'usuari pendents de l'anterior *sprint*.

Durant aquestes quatre setmanes, i per seguir traient-li avantatge a la situació de confinament en la que ens trobem, hem seguit amb la realització de reunions periòdiques mitjançant videotrucades, les quals s'han realitzat en dilluns, dimecres i divendres, excepte casos excepcionals, i en les quals hem simulat un *stand up*, per posar-nos al dia de l'estat de la feina de cada un dels membres. Acte seguit, si no s'havia de discutir cap punt del projecte, ens hem estat dividint en els dos equips prèviament mencionats (*front-end* i *back-end*) per treballar en les nostres respectives parts.

En general s'ha realitzat un bon treball al llarg d'aquest *sprint*, en aquesta segona iteració hem tingut en compte els problemes sorgits durant la primera iteració, i hem millorat la distribució de càrrega al llarg de l'*sprint*, de manera que és vagin iniciant i completant les tasques al llarg de l'*sprint* i no iniciar moltes històries a l'inici i acabar-les totes al final i amb presa degut a problemes de compatibilitat. A més, tot i que la sincronització entre front i *back* a millorat respecte a l'anterior *sprint*, algunes funcionalitats s'han tingut que adaptar posteriorment a la seva finalització per falta de compatibilitat entre front i *back*, per la qual cosa serà necessària una encara més estreta comunicació entre front i *back*, que tenim previst millorar en el següent *sprint*.

Però, tot i que el volum de càrrega de treball ha estat més distribuït al llarg de l'*sprint*, han sorgit diversos inconvenients al llarg de l'*sprint*, sent el més important la necessitat d'incloure, a meitat de la tercera setmana, més històries d'usuari a les plantejades originalment durant l'*sprint planning*, ja que eren necessàries per poder implementar les històries d'usuari originals. Concretament, hem hagut d'afegir Iniciar Sessió i Tancar Sessió amb Google, dos històries de 8 i 3 punts que eren necessàries per poder implementar la interacció amb el Google Calendar, i que han endarrerit el calendari de l'*sprint* i ens han impossibilitat acabar totes les històries d'usuari a temps.

Tot i no haver pogut completar totes les històries d'usuari, hem completat un 86,33% dels punts de les històries de l'*sprint*, quedant per fer 3 històries d'usuari amb un total de 9 punts (un 13,63% del total de punts), la qual cosa tenint en compte que hem hagut d'afegir dos tasques de gran pes que representen un 16,67% del total de punts de l'*sprint* en la tercera setmana de l'*sprint*, hem pareix un molt bon resultat. Per aquests motius, no considero que la no finalització del 100% de l'*sprint* sigui un motiu d'alarma, ja que tècnicament hem realitzat al llarg de l'*sprint* més punts que els




assignats originalment, i assolit un 57% del total de punts del projecte, superant el 50% ideal de punts assolits al final del segon *sprint*.

## 2. Requisits

En aquest apartat es mostren els canvis del *product backlog*, es dona una breu explicació dels requeriments no funcionals i de les aplicacions de tercers que es troben incloses al projecte.

### 2.1. Product Backlog (Actualitzat)

A continuació es mostren les modificacions de les històries d'usuari agrupades per tema. Els canvis sobre el *product backlog* queden indicats amb els colors següents:

-  Històries d'usuari assignades al primer *sprint*.
-  Històries d'usuari assignades al segon *sprint*.
-  Històries d'usuari assignades al primer i al segon *sprint*.

#### 2.1.1. Gestió de mascotes i usuaris (Actualitzat)

Gestió del compte de l'usuari	Gestió de les dades de l'usuari	Gestió de les mascotes de l'usuari	Gestió de la informació general de la mascota
Alta usuari	Modificar Nom d'usuari	Alta mascota	Modificar Nom Mascota
Baixa usuari	Modificar Correu Electrònic	Baixa Mascota	Modificar Sexe Mascota
Modificar idioma del sistema	Modificar Contrasenya	Consulta Mascotes	Modificar Raça Mascota
	CRUD Imatge de Perfil		Modificar Data de Naixement
			CRUD imatge de perfil per a la mascota

Taula 2. Històries d'usuari del tema: Gestió de mascotes i usuaris

### 2.1.2. Salut de la mascota (Actualitzat)

Informació bàsica	Alimentació	Higiene	Veterinari
Afegir pes	CRUD àpat	Afegir rentat	Consultar historial mèdic
CRUD Dades salut mascota	Calcular kcal aconsellades	Consultar pròxim rentat	CRUD visita veterinari
CRUD perfil mèdic mascota	Consultar kcal Consumides	Consultar historial de rentats	CRUD medicació

Taula 3. Històries d'usuari del tema: Salut de la mascota

### 2.1.3. Assistència a l'exercici

Gestió de les rutes de passeig	Control de l'exercici realitzat
Realitzar passeig	Gestionar l'exercici realitzat
Consultar ruta de passeig	Compartir exercici realitzat
Eliminar ruta de passeig	Eliminar els registres més antics periòdicament
Compartir ruta de passeig	

Taula 4. Històries d'usuari del tema: Assistència a l'exercici

### 2.1.4. Establiments especialitzats

Localització d'establiments especialitzats	Valoració d'establiments especialitzats
Veure establiments propers	CRUD valoració
Filtrar la visualització dels establiments	
Mostrar la ruta més ràpida	

Taula 5. Històries d'usuari del tema: Establiments especialitzats



### 2.1.5. Interacció en comunitats temàtiques (Actualitzat)

Gestió de continguts	Gestió de grups
Pujar, veure, comentar i eliminar contingut/fòrum (CRUD)	Crear/Esborrar/Editar grup (CRUD)
Donar <i>like/dislike</i> al contingut/fòrum	Subscriure's al grup
Denunciar contingut/fòrum	Desubscriure's al grup
	Rebre notificacions del grup

Taula 6. Històries d'usuari del tema: Interacció en comunitats temàtiques

### 2.1.6. Interacció amb xarxes socials (Actualitzat)

En aquesta iteració hem afegit, e implementat, la èpica *Google*, ja que era necessària entre altres coses per implementar la comunicació amb el *Google Calendar*, al qual no es pot accedir sense iniciar sessió amb *Google* i demanar autenticació.

Facebook	Twitter	Instagram	WhatsApp	Google
Iniciar sessió	Iniciar sessió	Iniciar sessió	Compartir fotos de la galeria	Iniciar sessió
Tancar sessió	Tancar sessió	Tancar sessió	Compartir aplicació	Tancar sessió
Compartir fotos de la galeria	Compartir fotos de la galeria	Compartir fotos de la galeria	Compartir publicacions de fòrums	
Compartir aplicació	Compartir aplicació	Compartir aplicació	Compartir aplicació	
Compartir publicacions de fòrums	Compartir publicacions de fòrums	Compartir publicacions de fòrums	Compartir informació de les mascotes	
Compartir informació	Compartir informació	Compartir informació		

Taula 7. Històries d'usuari del tema: Interacció amb xarxes socials

### 2.1.7. Interacció amb el calendari (Actualitzat)

Avisos personals	Avisos automatitzats	Notificacions a l'usuari
CRUD avís personal	CRUD avís generat	CRUD notificació periòdica

Taula 8. Històries d'usuari del tema: Interacció amb el calendari

### 2.1.8. Gestió de medalles i recompenses

Control del progrés de les medalles	Control medalles disponibles
Consultar requisits i progrés d'una medalla	CRUD medalla
Actualitzar progrés de les medalles	

Taula 9. Històries d'usuari del tema: Gestió de medalles i recompenses

### 2.1.9. Aspectes tecnològics

Aquest tema el vam afegir al principi del primer *sprint* ja que vam considerar que era necessari considerar les històries d'usuari que conté.

Aspectes relacionats amb les tecnologies utilitzades
Navegació entre diferents funcionalitats
CRUD Firebase
Creació API Gestió d'usuaris
Menú de settings
Menú principal
Preparació accés als serveis

Taula 10. Històries d'usuari del tema: Aspectes tecnològics

## 2.2. Requisits no funcionals (Actualitzat)

(Aquest apartat ha estat unificat amb l'anteriorment apartat 2.1.10 per evitar repeticions innecessàries)

### Original

En aquest primer sprint del nostre projecte ens hem centrat en tres requisits no funcionals, la rapidesa, la disponibilitat i la confidencialitat de la nostra aplicació. En el projecte, més concretament en el backlog d'aquest primer sprint, hem creat tres històries d'usuari per validar el seu compliment.

### Nova versió

En aquest segon *sprint* del nostre projecte ens hem centrat en mantenir els tres requisits no funcionals que ja vam proposar en el primer *sprint*, la rapidesa, la disponibilitat i la confidencialitat de la nostra aplicació.

### 2.2.1. Rapidesa

**Com a** usuari del sistema

**Vull poder** accedir als recursos proporcionats

**Per tal de** navegar entre les diferents funcionalitats el més ràpid possible

**Criteris d'acceptació**

- El temps de resposta ha de ser inferior a 5s.

### Original

Aquesta primera història l'hem pogut validar per l'estat actual de l'aplicació.

### Nova versió

Per tal d'assolir aquest objectiu, hem començat a paral·lelitzar, en l'aplicació, les tasques més costoses que són independents de la interfície gràfica. Com per exemple, l'actualització de dades al servidor és independent de l'actualització en local i, per tant, es pot realitzar en paral·lel amb el *thread* de la interfície gràfica. Malgrat tot, altres tasques com l'obtenció grups de la xarxa social interna han de bloquejar la interfície gràfica perquè es necessiten les dades per poder mostrar-los. En aquests

casos, estem executant en paral·lel tot allò que és possible. En la pròxima iteració, realitzarem un petit estudi sobre el temps d'execució de les principals tasques de l'aplicació, i l'eficiència en la seva execució.

### 2.2.2. Disponibilitat

**Com a** usuari del sistema

**Vull poder** accedir als recursos proporcionats sempre i quan disposi d'internet

**Per tal de** poder accedir a l'aplicació des de qualsevol lloc

**Criteris d'acceptació**

- L'aplicació ha de poder funcionar encara que es disposi d'una baixa connexió.

#### Original

Aquesta segona història l'hem pogut validar per l'estat actual de l'aplicació, tot i que, cal dir que sense connexió l'aplicació no funciona.

#### Nova versió

Aquesta segona història l'hem pogut validar per l'estat actual de l'aplicació, tot i que, cal dir que sense connexió l'aplicació no funciona. La implementació actual ens permet accedir a certa informació, ja que aquesta es guarda en local, però no ens permetria interactuar amb altres usuaris ni amb el servidor.

### 2.2.3. Confidencialitat

**Com a** usuari del sistema

**Vull poder** guardar les meves dades de manera privada

**Per tal de** poder mantenir segura la meva informació sensible

**Criteris d'acceptació**

- Un usuari no pot accedir a informació privada d'altres usuaris.
- Un usuari no pot modificar informació d'altres usuaris.

### Original

Aquesta última història l'hem pogut dur a terme posant un requisit de privacitat a l'inici de sessió, ja que només es tindrà accés a les dades d'un compte estant connectat en aquest.

### Nova versió

Aquesta tercera història l'hem pogut validar per l'estat actual de l'aplicació. Açò ho hem pogut dur a terme posant un requisit de privacitat a l'inici de sessió, ja que només tindrà accés a les dades de la base de dades de l'usuari logejat.

## 2.2.4. Abstracció i Reutilització (Nou)

**Com a** desenvolupador del sistema

**Vull poder** reutilitzar parts del codi i disminuir la complexitat

**Per tal de** poder escriure nou codi i modificar el codi ja implementat sense tanta dificultat

### Criteris d'acceptació

- Un desenvolupador ha d'intentar reutilitzar i fer codi reutilitzable sempre que sigui possible.

Aquest últim requisit no funcional l'hem pogut dur a terme aplicant molts dels patrons explicats posteriorment en la documentació, concretament els patrons que ens han permès garantir l'abstracció i reutilitzar codi han estat:

- Patró factoria abstracta: per crear famílies d'objectes relacionats sense especificar les seves classes concretes.
- Patró factoria simple: consisteix en utilitzar una classe constructora abstracta amb uns quants mètodes definits i altres abstractes, garanteix reutilització per part de les subclasses.
- Patró Adaptador: s'aplica quan es vol utilitzar una classe però la seva interfície no concorda amb la que necessitem, o quan es vol reutilitzar una classe.

- Patró model, vista, controlador: aquest patró divideix l'aplicació en tres parts interconnectades: el model de dades, la interfície d'usuari i la lògica de control; garantint abstracció entre elles, ja que una no coneix el funcionament intern de l'altra.
- Patró DAO: consisteix separar del tot la lògica de negoci de la lògica per accedir a les dades, l'apliquem al servidor per garantir abstracció.
- Patró Service Locator: serveix per a encapsular els serveis en una capa abstracta, l'hem utilitzat en l'accés a la nostra llibreria d'Android.
- Patró Singletó: restringeix la creació d'objectes, el seu rol és permetre l'existència de només una única instància de l'objecte i proporcionar una referència global a aquest, garantint la reutilització.
- Patró controlador de transaccions: fa d'intermediari entre la interfície i l'algoritme que l'implementa, reduint la complexitat i reutilitzant codi.
- Patró plantilla: és un patró de disseny de comportament que defineix l'esquelet d'un algoritme en un mètode que, en ser heretat per les subclasses, difereix en alguns dels seus passos, garanteix reutilització de codi.
- Patró estratègia: aquest és una patró de disseny de comportament que ofereix diversos algoritmes al client i li dona la possibilitat de canviar entre ells segons les seves necessitats en una sola comanda, garanteix reutilització de codi.

### 2.3. Tractament d'aspectes transversals (Nou)



En aquest segon *sprint* del nostre projecte ens hem centrat en cinc aspectes transversals, les xarxes socials, el xat, la refutació, el calendari i les gràfiques. Les xarxes socials ja es pot fer *login* amb Google i Facebook, i en un futur volem poder compartir la nostra aplicació i contingut a través d'aquestes i algunes altres xarxes socials. Per altra banda, volem aconseguir implementar fòrums amb xats instantanis i històrics.

A més, també estan implementades les notificacions i els events, els quals es guarden en un calendari intern que està sincronitzat amb el de Google. Pel que respecta al tema de les gràfiques, ja estan implementades i mostren correctament els diferent estadístics, com ara les quilocalories setmanals, el pes, ... A més de presentar un comportament dinàmic.

Pel que respecta al tema del multi-idioma, hem mantingut el que ja teníem fet de l'*sprint* anterior, i l'únic que s'ha fet ha estat solucionar algunes traduccions i afegir-ne les necessàries per als nous component.

De cara a futurs *sprint* ens agradaria poder tenir més *stakeholders* reals, ja que malgrat que hem tingut alguns, la situació actual dificulta molt obtenir més feedback. Per tal d'aconseguir aquest feedback, proporcionarem als “testers” un formulari de Google amb preguntes sobre el funcionament de l'aplicació i amb la possibilitat de donar la seva opinió sobre aspectes a millorar. Probablement els “testers” acabaran sent familiars i amics ja que la situació actual originada per la COVID-19 redueix bastant el possible abast.

També voldríem assolir la geolocalització amb mapes, la gamificació amb trofeus i medalles, la refutació amb bloqueig de comptes, i per últim, la *web app*, de la qual en un futur en definirem l'abast i la seva viabilitat.

## **2.4. Serveis de tercers (Actualitzat)**

A continuació es dona una breu explicació dels serveis externs utilitzats en el projecte.

### **2.4.1. Firebase**

Firebase és una plataforma que proporciona diverses eines per desenvolupar aplicacions. Aquesta plataforma proporciona una gran quantitat d'eines de les quals nosaltres hem utilitzat concretament dues, Cloud Firestore com a contenidor online per la nostra base de dades i Firebase Authentication per la gestió de l'autenticació dels usuaris. A continuació s'explica en més detall per a que serveixen aquestes eines.

Cloud Firestore és el servei que hem utilitzat per allotjar la nostra base de dades, aquest proporciona una base de dades NoSQL, flexible, escalable i en el núvol per tal d'emmagatzemar i sincronitzar dades per la programació tant des del client com des del servidor. Nosaltres hem utilitzat únicament l'emmagatzematge i sincronització des del servidor.

Firestore Authentication és el servei que hem utilitzat per gestionar l'autenticació d'usuaris, aquest proporciona serveis de backend, SDK fàcils d'utilitzar i biblioteques de IU ja elaborades per autenticar els usuaris en la teva app. Per aquest motiu i per la seva fàcil integració amb Cloud Firestore, l'hem utilitzat per gestionar l'autenticació d'usuaris.

#### **2.4.2. Google Calendar (Nou)**

Google Calendar, és un servei gratuït d'agenda i calendari en línia desenvolupat per Google, que et permet sincronitzar-lo amb gmail i compartir-lo amb altres persones si volem, i et permet afegir esdeveniments i invitacions i fer cerques d'esdeveniments que poden interessar-te a la web.

Algunes característiques de Google Calendar són:

- Ús compartit del calendari: podem crear diversos calendaris per a, per exemple, compartir-los amb un grup d'alumnes o d'amics, amb la família, amb companys de treball, etc. Més informació.
- Invitacions: Crea invitacions a esdeveniments, enviar-les als teus amics i porta el compte de les seves respostes i comentaris. Tot això en un únic lloc. Els teus amics rebran la teva invitació i introduiran les seves respostes, encara que ells no utilitzen Google Calendar. Es poden transferir calendaris a altres usuaris de Google calendar.
- Subscripcions: Et pots subscriure a informació de calendaris públics
- Cercar: Cercar a calendaris públics per descobrir esdeveniments que et puguin interessar i afegir-los a teu propi calendari.
- Accés mòbil: Rep notificacions i recordatoris d'esdeveniments al teu telèfon mòbil.
- Publicació d'esdeveniments: Comparteix els esdeveniments que creguis oportú amb qui vulguis. Al fer públic el teu calendari, tots els teus esdeveniments apareixeran en els resultats de cerca públics de Google Calendar i de Google. A més, altres usuaris podran veure aquesta informació o afegir el calendari a la seva llista de calendaris.



### 2.4.3. OAuth2 (Nou)

OAuth 2 és una estructura (framework) d'autorització que li permet a les aplicacions obtenir accés limitat a comptes d'usuari en un servei HTTP, com Google, Facebook, GitHub i DigitalOcean. Delega la autenticació de l'usuari al servei que allotja el compte del mateix i autoritza a les aplicacions de tercers l'accés a l'esmentat compte d'usuari. OAuth 2 ofereix fluxos d'autorització per aplicacions web i d'escriptori; i dispositius mòbils.

Aquest servei és necessari per facilitar l'accés al Google Calendar des de la nostra aplicació utilitzant un compte de Google, ja que les API de Google utilitzen el protocol OAuth 2.0 per autenticació i autorització.

### 3. Cerimònies àgils (Actualitzat)

En aquest apartat es detallen els tres esdeveniments més significatius d'un *sprint*: el *sprint planning*, el *sprint review* i el *sprint retrospective*.

#### 3.1. Sprint Planning (Nou)



Aquest nou Sprint va començar el divendres 30 de Març i està previst que acabarà el 24 d'Abril. Aquest document és un resum d'aquella reunió:

1. Es van revisar els objectius de la anterior iteració i es van traslladar aquells que no havien estat aconseguits, concretament: la possibilitat de canviar el nom d'usuaris i mascotes, el CRUD de les imatges i certes optimitzacions necessàries a les llibreries.
2. Es van definir els objectius d'aquesta iteració: volem, a més de finalitzar allò que no vam poder en la iteració anterior, introduir la resta de funcionalitats de les mascotes (pes, dieta, medicació, etcètera), afegir la funcionalitat del calendari i implementar els fòrums.
3. Es va decidir que es mantindrien els equips de Front End i Back End iguals. Vam considerar que ambdós equips havien aconseguit un coneixement dels seus àmbits indispensable per a la nova iteració que se'ns faria massa costós de transmetre a noves mans, especialment considerant que vam preveure un increment en les activitats de la universitat i dificultats comunicatives imprevistes degut a la present pandèmia.
4. Es va decidir que s'intentaria fer un seguiment més paral·lel entre Front End i Back End. Una de les principals dificultats que vàrem tenir en la anterior iteració va ser que la manca de comunicació entre els dos equips a vegades portava a que un equip no pogués començar una tasca fins que l'altre equip n'acabés la seva, donant pas a temps perdut. Per això vam decidir fer un Planning general sobre com aniríem implementant cadascuna de les funcionalitats. En concret, per començar, vam decidir que mentre Back implementava la funcionalitat de les imatges, Front n'implementaria de calendaris i, un cop acabats els dos, Back implementaria la seva part de calendaris i Front la seva de imatges.

5. De cara a la documentació, estàvem ben satisfets amb els resultats de la iteració anterior, per això vam decidir mantenir els mateixos equips que abans.
6. Un cop decidit tot això, ens vam separar per equips i els nostres respectius caps d'equip van distribuir el treball.

### 3.2. Sprint review (Nou)

El dia 24/4/2020 va ser el dia de la finalització de l'Sprint. Ens vam reunir per tal de veure què hem aconseguit entregar. Aquest document inclou el resultat de la nostra release.

#### 3.2.1. Què s'ha pogut entregar?

Documentació: complerta. Hem pogut entregar tots els documents requerits.

- Modificar idioma de sistema.
- CRUD Imatge de perfil de mascota i usuari.
- CRUD Modificar nom d'usuari i mascota.
- CRUD Contingut/fòrum
- CRUD Grup
- Subscripció i de-subscripció al grup
- CRUD avisos personals
- CRUD dades de salut de la mascota
- CRUD medicació
- CRUD àpat
- CRUD Kcal
- Iniciar sessió: amb compte de Google.
- Tancar sessió: amb compte de Google.
- Implementació d'un calendari

#### 3.2.2. Què no s'ha pogut entregar?

- Donar like/dislike al contingut/fòrum
- Denunciar contingut/fòrum
- Rebre notificacions de grup

### 3.2.3. Per quin motiu?

Aquestes tres històries d'usuari no van poder ser entregades perquè la implementació del calendari requeria de la implementació de dues noves històries d'usuari: l'inici i el tancament de sessió de Google. Aquestes dues històries d'usuari van provar ser molt difícils ja des d'un principi i van requerir massa temps, per això va ser necessari descartar-ne algunes altres.

## 3.3. Sprint retrospective (Nou)

Al dia 24 d'Abril, l'equip es va reunir per realitzar l'Sprint Retrospective. Es va decidir realitzar-ho en dues parts: la primera amb tot l'equip, i la segona separant Front i Back per a fer front als seus problemes concrets. Aquest document recull les conclusions d'aquesta reunió:

### 3.3.1. Reunió General

Primerament, tothom es va posar d'acord en que la distribució del treball havia anat millor que a la anterior iteració, però que, una altra vegada, s'havia acumulat massa per a la última setmana. Així, des de Front es va presentar la preocupació de que, encara que Back havia aconseguit aquesta vegada finalitzar les seves tasques abans i, per tant, havien tingut, generalment, més temps per a implementar-les i testear-les al Front, encara hi havien hagut una sèrie de funcionalitats que havien arribat massa tard.

Des de Back, d'altra banda, es va expressar la preocupació de que part d'aquest retràs en aquestes funcionalitats es deu a una falta de comunicació amb Front. I és que, a vegades, Front no ha estat massa clar a la hora d'explicar què vol, que a la llarga ha portat a tenir que refer documents enters i a molt de temps perdut.

Després, es va comentar un altre problema: el de la falta de disponibilitat. Durant aquestes setmanes hi han hagut moltes situacions en les que membres de l'equip no han estat disponibles i han respost massa tard a les seves sol·licituds. Encara que es comprèn que, generalment, tenim molt més treball apart del d'aquesta assignatura, es va demanar que tothom fes un esforç per tal d'estar més atent al Whatssap o més present al Discord.

Per últim, es va recordar que cal indicar les hores dedicades a cada tasca al Taiga.

### 3.3.2. Reunió Front

En la reunió de la retrospectiva de Front, s'ha tractat les diferents qüestions relatives a aquest subequip que han sorgit durant el desenvolupament d'aquesta iteració.

Per una banda, s'ha acordat que hi ha hagut més comunicació entre tots i, per tant, les històries que s'han aconseguit finalitzar s'han tancat correctament i sense problemes. Per l'altra banda, s'han tingut un problema amb un membre de l'equip que no ha treballat suficient en aquesta iteració degut als constants dubtes que li han sorgit al llarg d'aquesta. S'ha parlat amb ell sobre aquesta situació i s'ha compromès a realitzar més tasques i a avisar a algun altre membre de l'equip si té algun problema amb els seus codis.

Pel que fa a les decisions presses, s'ha acordat que en el cas que una història sigui d'una complexitat alta, aleshores dos membres de l'equip s'encarregaran d'aquesta.

### 3.3.3. Reunió Back

En la reunió de la retrospectiva de Back, també s'han tracta les diferents qüestions relatives al subequip que han sorgit durant el desenvolupament d'aquesta iteració.

En primer lloc, s'ha fet front a un major problema que s'ha tingut des de la iteració anterior: encara que la distribució de treball ha estat millor, un altre cop s'ha deixat massa coses per al final. Això ha portat a que la release no ha estat de tan bona qualitat com s'esperava i s'ha hagut de fer *Sprints* esgotadors a l'últim moment. L'equip s'ha compromès a fer una millor distribució del treball i procurar, d'ara endavant, fer treball més constant.

En segon lloc, es va fer una crida d'atenció a la qualitat dels merges: ha estat molt freqüent que, tot i que els tests passen, la funcionalitat no ha funcionat al servidor i per tant ha estat necessari refer codi. És per això que s'ha demanat que, d'ara endavant, es facin tests molt més exhaustius amb el Postman. Això ha estat un problema també amb fragments de codi similars amb altes: encara que es copiï codi i es canviïn unes quantes coses, és convenient revisar-ho profundament, encara que passi els tests.

En tercer lloc, respecte al problema mencionat a la reunió general, el de la manca de comunicació amb Front, s'ha decidit ser molt més insistents amb els detalls de les funcionalitats demanades.

Per últim, l'equip ha reconegut que, generalment, hi ha hagut bona comunicació entre els companys i tothom ha estat receptiu a ajudar als altres.

#### 3.3.4. Què s'ha fet bé i es mantindrà?

- **Mètode de realització de documentació:** al principi del *sprint* vam dividir la documentació en tasques a realitzar setmana a setmana. Cada setmana, durant les nostres reunions, dedicàvem 20 minuts a comprovar que tota la documentació d'aquesta setmana estigués llesta i decidíem què s'havia de fer per a la següent setmana i qui s'encarregaria de fer què. Aquest mètode ens ha permès organitzar-nos de tal manera que ningú s'ha vist compromès entre dedicar temps a programació i a documentació i, a més, ens ha permès entregar-la a temps.
- **Reunions fora d'horari de classe:** com que el nostre grup té reunions amb el *product owner* dimecres i divendres, ens vam adonar que l'espai de temps entre aquests dos dies era massa petit per a fer cap avanç important i, a la vegada, massa gran per a poder respondre adequadament a problemes que poguessin sorgir en el desenvolupament del software. És per això que vam decidir que faríem reunions també els dilluns. Aquesta petita mesura ens ha dotat d'una millor agilitat i capacitat de planificació per tal de fer front als diversos esdeveniments que porta el desenvolupament d'una aplicació.
- **Comunicació per Discord:** la situació actual del COVID-19 ens ha forçat a tots a estar confinats en casa, per tant, vam perdre l'avantatge de la comunicació cara a cara. Afortunadament, vam decidir que ens comunicàrem per Discord, una plataforma de comunicació per a jugadors gratuïta i àgil que ens ofereix la capacitat de fer xat (tant oral com escrit) i *streaming* del nostre ordinador. Així, si algú tenia cap problema i necessitava ajuda, podíem respondre ràpidament.

### 3.4. Gràfiques (Actualitzat)

#### 3.4.1. Sprint burndown

En la gràfica de la figura 1, podem observar l'evolució del punts d'històries d'usuari pendents en cada dia del *sprint*. A l'inici, aquesta evolució es va mantenir pràcticament constant; malgrat tot, cap al final va començar a disminuir considerablement. De fet, en d'una setmana es va passar de 50 punts per tancar a només 24.

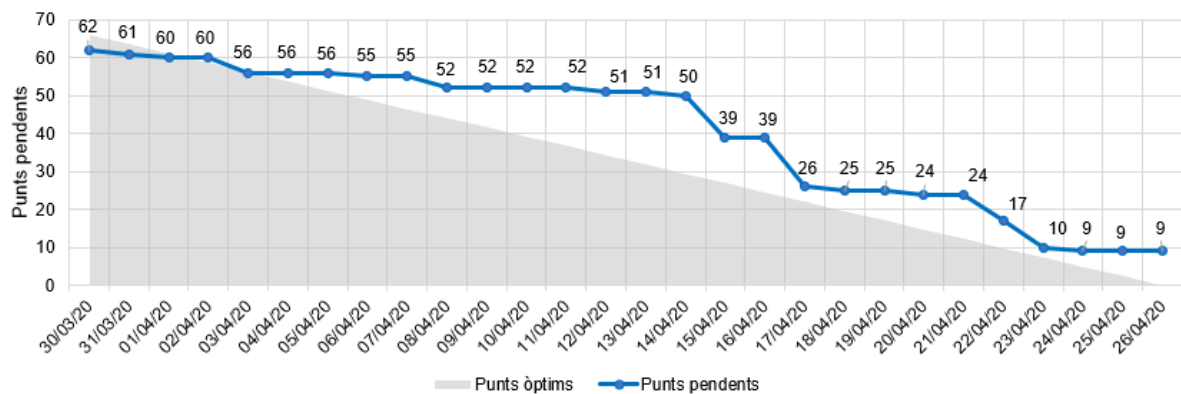


Figura 1. Sprint burndown de la segona iteració

#### 3.4.2. Release burndown

En la figura 2, podem observar l'evolució general dels punts de les històries d'usuari pendents per implementar. Malgrat les dificultats que hem tingut a l'inici de la iteració, al final hem assolit menys d'una mitja part del projecte.

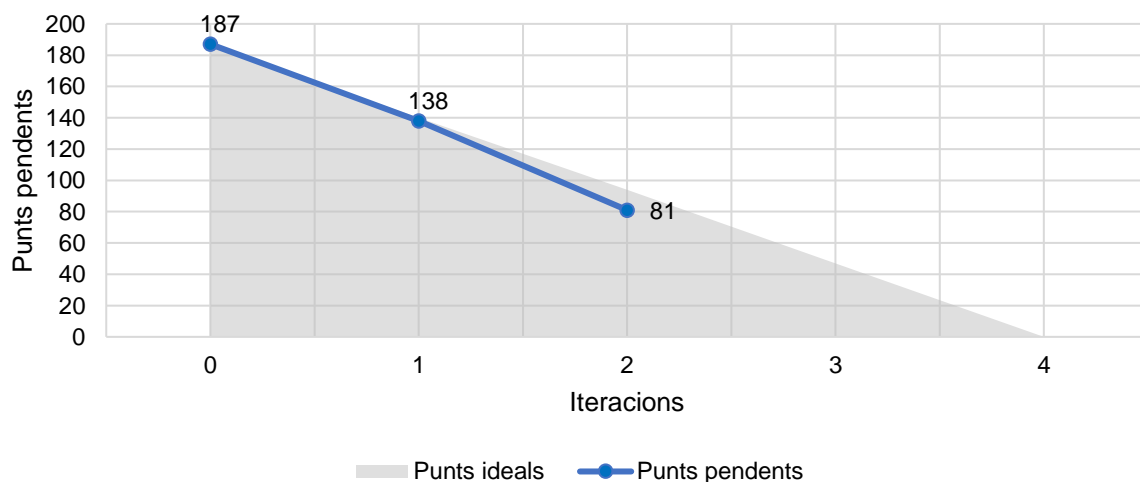


Figura 2. Release burndown

### 3.4.3. Velocitat de l'equip

En la figura 3, es poden observar les velocitats de l'equip durant el transcurs del projecte, fins el moment de la publicació d'aquest document. En aquesta es comparen els punts tancats i els planificats.

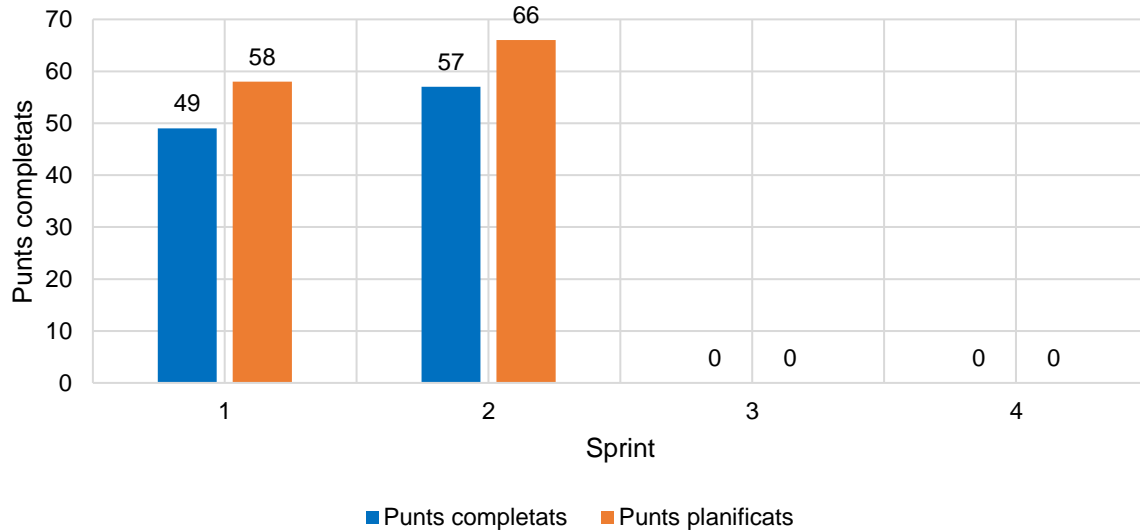


Figura 3. Representació de la velocitat de desenvolupament de l'equip

### 3.4.4. Dedicació mensual

En la figura 4, es troben representades les hores invertides per a cada desenvolupador en els darrers mesos. Podem observar que la quantitat d'hores dedicades durant el mes de març és molt superior a les del mes anterior. Aquest fet és degut a l'inici de la fase de desenvolupament.

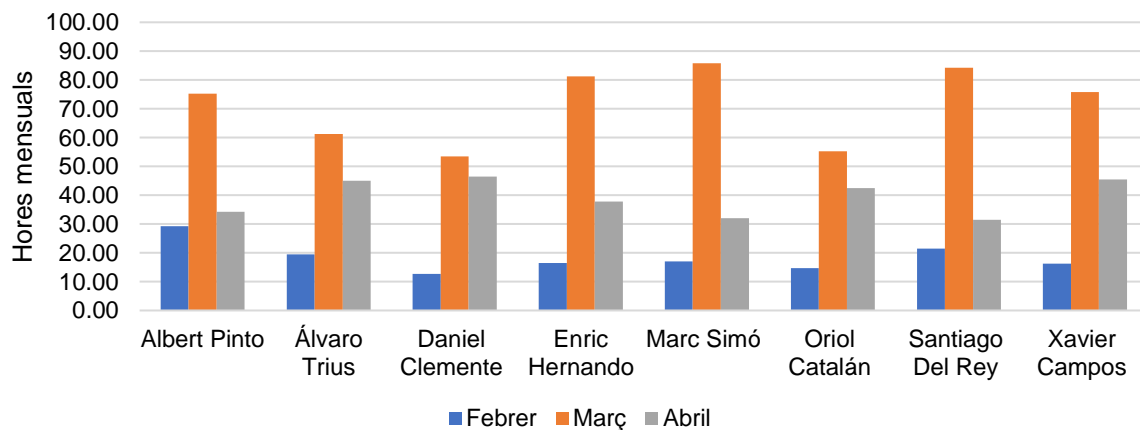


Figura 4. Hores de dedicació mensual del membres de l'equip



## 4. Metodologia

En aquest apartat s'explica el conjunt d'eines i convencions que s'utilitzaran per realitzar el projecte.

### 4.1. Metodologia àgil

Per a realitzar el projecte, hem decidit utilitzar SCRUM, una de les principals metodologies de desenvolupament de software àgils utilitzades actualment. En aquesta, es divideix el desenvolupament del projecte en iteracions o *sprints*, al final de les quals es disposarà d'una versió entregable del sistema.

En aquesta metodologia àgil, es defineixen tres rols destacats. Primerament, el product owner, el qual vetllarà per al compliment de les funcionalitats acordades i, en el nostre cas, serà sempre la mateixa persona. En segon lloc, tenim el *scrum master*, qui verificarà que s'estigui aplicant la metodologia de treball correctament i redactarà la documentació necessària per la iteració actual. Aquest rol serà rotatiu entre els diferents membres de l'equip. Finalment, els desenvolupadors són els encarregats d'implementar les funcionalitats acordades amb el client.

Per tal de coordinar les nostres activitats, durant les iteracions realitzarem les diferents reunions que estableix aquesta metodologia. Per una banda, al principi de cada iteració, realitzarem una reunió per decidir que realitzarem en aquesta i durant el seu transcurs cada dia que ens trobem en persona indicarem quines tasques hem fet durant la setmana, què farem avui i quins problemes hem tingut. Per l'altre banda, al final de cadascuna de les iteracions realitzarem dues reunions, una amb el client per mostrar les funcionalitats implementades i l'altre per debatre sobre quins aspectes del desenvolupament podríem millorar. La duració de cadascuna de les iteracions serà de tres setmanes.

## 4.2. Definició de finalització

Per tal de decidir quan una història d'usuari està acabada hem establert un conjunt de criteris definits a continuació:

1. Està definida i conté els seus criteris d'acceptació en el Taiga.
2. Està implementada amb els seus tests necessaris per a complir els seus criteris d'acceptació.
3. Els mètodes de les classes estan documentats.
4. Passa la revisió automàtica i és verificada per com a mínim un altre membre.
5. S'ha afegit el codi a la branca de desenvolupament.
6. Els diagrames de classes estan actualitzats.

## 4.3. Gestió del projecte

Per tal de realitzar el nostre projecte, hem decidit utilitzar un conjunt d'eines per la gestió d'aquest. Primerament, per tal de controlar les històries d'usuari que hem dissenyat, ens hem decantat per fer servir Taiga, una eina gratuïta específica per a gestionar projectes àgils. Aquesta ens permet definir les èpiques i les seves històries, puntuar-les segons la seva complexitat, organitzar el product backlog, planificar els *sprints* i observar l'evolució de l'estat de les històries.

En segon lloc, per tal de gestionar el codi del projecte hem optat per fer servir GitHub, un sistema de control de versions (VCS). En aquest, utilitzem una metodologia de treball anomenada Gitflow, en la qual s'organitza el codi en branques independents entre elles. A més, aquesta eina és compatible amb les altres eines que utilitzarem en el projecte.

Finalment, per tal de controlar les hores de dedicació personal al projecte, hem creat un full de càlcul anomenat Project record track. En aquest, hem d'afegir les diferents activitats vinculades al projecte que anem realitzant, com per exemple, la realització del codi o la redacció de la documentació.

#### 4.4. Convencions

Per tal de gestionar correctament les iteracions hem definit un conjunt de convencions de nomenclatura.<sup>1</sup> Els convenis que tenen un asterisc entre parèntesis “(\*)” són verificades pel Codacy:


- Tot allò que no sigui documentació o un comentari s'ha de realitzar en anglès.
- Les release tindran un nom en el format vX.Y.Z, començant en la v1.0.0, tal que:
  - Per a cada nova release s'incrementa la X.
  - Per a cada hotfix solucionat s'incrementa la Z.
  - Si la solució d'un hotfix ha provocat un gran canvi en l'aplicació, s'incrementa la Y.
- Els *pull request* realitzats s'anomenaran com la branca des d'on provenen, amb una breu descripció si es creu convenient.
- Els missatges dels *commits* s'escriuen en present.
- Exceptuant el *project record track*, cal iniciar sessió en totes les eines amb el compte de GitHub, si és possible.
- Els noms de les classes han de seguir un dels següents patrons, en funció de quin sigui el seu contingut:
  - Activitat: *NomActivity*
  - Fragment: *NomFragment*
  - Gateway: *NomGateway*
  - Vista (component): *NomView*
  - Intent: *targetActivityIntent*
  - JUnit test: *NomClassePerTestejarTest*
  - Altres: poden tenir el nom que es consideri oportú.
- En tots els fitxers en java s'utilitzarà camel case en tots els noms (\*); malgrat tot, en xml s'escriurà tot en minúscula separant les paraules amb guions baixos, excepte amb els identificadors els quals hauran d'estar amb camel case.
- Els noms de les variables i dels mètodes han de ser significatius.
- Les interfícies no poden començar amb el prefix “I”.

---

<sup>1</sup> Per consultar tots els convenis establerts, consulteu el següent [enllaç](#).

- Els identificadors dels components han de començar amb un prefix significatiu, els quals s'aniran concretant sota demanda. Per exemple, *btn*Nom pels botons o *lb*/Nom per les etiquetes de text.
- Les funcions no poden tenir més de 15 línies de codi, sense comptar els comentaris (\*).
- Les funcions no poden tenir més de 5 paràmetres (\*).
- L'idioma utilitzat en el codi és l'anglès, excepte el contingut dels elements del fitxer de recursos Strings.xml amb un locale assignat.
- Les diferents pantalles de l'aplicació, excepte el log in, han de ser implementades utilitzant fragments. Per tal de poder comprovar el funcionament n'hi ha prou amb crear una activitat i incloure el fragment, tot i que aquesta no pot estar en el *pull request* realitzat.
- S'ha de respectar l'arquitectura en tres capes establerta.
- Per accedir als diferents elements de la interfície gràfica, s'ha de fer servir el View Binding, introduït a Android Studio en la versió 3.6.
- Els noms de les branques han de seguir els següents criteris:
  - Si la branca està relacionada amb una nova funcionalitat, aleshores s'anomenarà *feature\_user\_story\_name*.
  - Si la branca implementa una llibreria per accedir al servei, aleshores s'anomenarà *library\_service\_name*.
  - Si la branca implementa un aspecte tècnic que no estigui en cap història d'usuari, aleshores s'anomenarà *technical\_action\_name*.
  - Si es tracta d'un altre tipus de branca s'anomenarà *type\_reason\_name*.
- Per a crear un intent des d'una activitat a una altra, s'ha de posar com a primer paràmetre "*NomActivity.this*" en lloc de "*this*".
- Els noms dels mètodes de les classes dels tests unitaris (JUnit) han de ser autoexplicatius.

## 4.5. Gitflow

Per tal d'utilitzar el repositori de codi eficientment i evitar problemes d'incompatibilitat de versions, hem decidit utilitzar una metodologia de treball anomenada gitflow. Aquesta estableix en quin tipus de branca del repositori s'ha d'incorporar el codi que hem realitzat i quin és el procediment a seguir en cada cas. En total distingim 5 tipus de branques: *master*, *develop*, *feature*, *release* i *hotfix*. 

Primerament, en la branca de *master* es troba el codi de l'aplicació que forma part d'una versió publicada de l'aplicació, és a dir, el codi resultant al final d'una iteració. Per tant, no es pot realitzar cap *commit* que provingui directament de l'entorn de desenvolupament en aquesta branca. En el nostre projecte, hem acordat que l'únic cas on està permès fer un *commit* sense passar pel procediment habitual és quan incorporem al repositori la documentació d'aquest, la qual es troba únicament en aquesta branca.

Acte seguit, trobem la branca *develop*, la qual prové de la *master*. En aquesta s'aniran incorporant les diferents funcionalitats que es vagin desenvolupant durant el projecte, un cop hagin passat els diferents controls de qualitat establerts. De forma similar a la branca *master*, no es pot realitzar un *commit* directament en aquesta, sinó cal passar abans per un altre tipus de branca.

A continuació, disposem de les branques de tipus *feature*, les quals contenen el codi de les noves funcionalitats de l'aplicació o que solucionen problemes trobats durant el desenvolupament. Aquestes branques provenen de la branca *develop* i es on es realitzen tots els *commits* des de l'entorn de desenvolupament. D'aquesta manera el codi nou queda aïllat del codi comprovat i potencialment funcional de la branca original. Per tal de poder incorporar aquests canvis a la *develop*, és a dir, realitzar la fusió entre les branques, és necessari realitzar un *pull request*, en el qual es passaran un seguit de tests d'estil de forma automàtica i una comprovació de compatibilitat entre classes. A més, hem establert que el codi ha de ser verificat totalment per una altre membre del grup, el qual no hagi participat en el desenvolupament d'aquesta part. Malgrat tot, tots els membres poden comentar el codi i suggerir millores d'aquest. Un cop tots els tests han passat i el codi ha estat verificat, es pot realitzar la fusió amb la branca *develop* i, acte seguit, s'eliminarà la branca original.

Quan s'acosta el final de la iteració, es decideix finalitzar la incorporació de noves funcionalitats i iniciar una branca del tipus *release*, la qual prové de la branca *develop*. Totes les funcionalitats que no han estat incorporades a la branca *develop* abans de l'inici d'aquesta nova branca queden fora de la versió actual de l'aplicació. En aquesta, es solucionaran petits errors o *bugs* que s'hagin detectat i es prepararà el codi per a ser lliurat com una versió de l'aplicació. Aquestes comprovacions es realitzaran mitjançant un *pull request* en el qual tots els membres del grup hauran de revisar alguna part del codi per tal de trobar inconsistències o detectar problemes. Un cop ha estat verificat es realitza primer la fusió amb la branca *master* i, acte seguit, amb la *develop*.

Finalment, si un cop s'ha publicat la versió de l'aplicació és detecta algun problema amb l'aplicació, sobretot provinent de les proves amb usuaris reals, s'ha d'iniciar una branca anomenada *hotfix* des de la *master*. Aquesta s'encarrega de solucionar aquests problemes i, acte seguit, s'inicia un *pull request*. Un cop el codi ha estat verificat i s'ha comprovat que aquests han estat solventats, es realitza la fusió amb la branca *master* i, a continuació, amb la *develop*.

#### 4.6. Testeig

Durant el desenvolupament del projecte, hem de realitzar un conjunt de tests per assegurar la integritat i el correcte funcionament de l'aplicació. Per a dur-los a terme, farem servir tres *frameworks* diferents: JUnit, el qual permet realitzar tests unitaris, Mockito, que permet realitzar *mocks* en tests unitaris, i Espresso, que permet realitzar tests instrumentals. L'ús d'aquests *frameworks* permetrà que puguem aplicar el TDD per a desenvolupar el codi relacionat amb la lògica de l'aplicació.

Per tal de calcular quin percentatge de les classes ha estat provat, utilitzem la llibreria Jacoco. Aquesta, genera un report detallat on s'indica el percentatge d'instruccions i branques que disposen de tests. A més, indica la quantitat total de línies, mètodes i classes que no estan sent testejades.

Per aprofitar al màxim aquests tests i la utilització de Jacoco, utilitzem les *pipelines* de GitHub per córrer els tests a cada push a les branques *feature* y en cada *pull request* cap a *develop* i *master*. Un cop es realitzen els tests i es genera el report, aquest s'envia a Codacy, el qual ens indicarà si el repositori compleix amb l'estàndard de cobertura que hem decidit.

#### 4.7. GitHub Actions

Per tal d'aplicar els principis de *continuous integration* i *continuous deployment*, farem servir les *pipelines* que ens proporciona GitHub, ja que és el nostre sistema de control de versions i, per tant, tindrà una millor integració amb el repositori. Aquestes *pipelines* ens proporcionen un mètode per automatitzar la realització de tests, la creació de paquets i muntar o desplegar la nostra aplicació.

Primerament, es defineix el nom que portarà la *pipe* i els esdeveniments que l'activen. Per a cadascun d'aquests, es pot escollir en quina o quines branques es pot iniciar. A continuació, es creen les tasques a realitzar, les quals s'executaran en paral·lel si no s'indica el contrari. Per a la creació de cadascuna d'aquestes s'indica el seu nom i, després, s'especifica en quin o quins sistemes operatius s'executarà, podent escollir entre diferents versions de Linux, mac Os o Windows. Un cop arribats a aquest punt, es defineixen les etapes que ha de seguir la tasca. Aquestes poden ser comandes del terminal o altres accions definides per GitHub o per altres usuaris.

#### 4.8. Dependabot

Per tal de mantenir les dependències actualitzades i evitar així problemes de compatibilitat o seguretat, incorporem Dependabot. Aquesta és una eina que revisa el codi en busca de dependències desactualitzades. En cas de trobar-ne alguna, Dependabot obre un *pull request* informant de quina és la versió que ha trobat i quina és la que hauria d'haver-hi i, en el moment que detecta que s'ha resolt el problema, aquest tanca automàticament el *pull request*.

#### 4.9. GitGuardian

Tenint en compte que el nostre projecte es troba a un repositori públic, on qualsevol pot veure el contingut del nostre codi, s'ha de tenir cura de no pujar informació que pugui comprometre la integritat de la nostra aplicació, com podrien ser les API keys de Google Maps o Firebase. Per aquest motiu, hem incorporat GitGuardian als nostre repositoris. Aquest eina de monitorització per a repositoris, s'encarrega d'analitzar tots els fitxers ubicats dins d'un repositori i alertar de possibles bretxes de seguretat cada cop que es realitza un push. En cas de trobar-ne alguna, s'envia un correu a la o les persones especificades indicant quin fitxer conté informació que pot ser considerada sensible.

#### 4.10. Qualitat del codi (Nou títol, anteriorment anomenat Codacy)

Per tal de mantenir el nostre codi net i sense *code smells*, utilitzarem una eina de revisió i anàlisi de codi anomenada Codacy. Aquesta eina disposa d'un seguit de regles, les quals pots activar i personalitzar, per tal de crear uns criteris de revisió pel codi. Amb aquestes, s'intenta que tot el codi desenvolupat dins del projecte segueixi els mateixos criteris d'estil, és a dir, que estigui estructurat i definit de la mateixa manera independentment de qui l'hagi escrit.

Primerament, l'eina porta un control dels problemes detectats als codis, com poden ser errors en l'estil, zones de codi duplicat o no utilitzat, o parts de codi que poden ser propenses a errors, entre altres. Tots aquests problemes es mostren en una taula on apareix la quantitat de cadascun. Un altre lloc on els podem trobar és a la gràfica de qualitat, en la qual es mostra l'evolució de la qualitat del projecte. Aquesta s'obté de mesurar la quantitat d'errors en el codi, la complexitat d'aquest i la quantitat de codi repetit.

Acte seguit, podem veure el llistat de *commits* i *pull requests* que s'han anat realitzant juntament amb la quantitat d'errors que aquest ha generat i quins han siguts arreglats. També ens permet veure informació més detallada dels nostres fitxers mostrant-nos, per exemple, la seva complexitat i els errors que es troben en aquests. En el cas dels errors, aquests apareixen ressaltats sobre el codi, juntament amb el motiu i una breu explicació d'aquest, cosa que facilita molt la seva correcció.



Finalment, Codacy disposa d'integració amb GitHub, permetent així enllaçar el nostre repositori amb el revisor. D'aquesta manera, podem configurar que s'iniciïn revisions del codi cada cop que es realitzi un *commit* o un *pull request* a una determinada branca. A més, Codacy s'encarrega d'enviar els resultats de les revisions com a resposta de l'acció que l'ha activat. Això ens permetrà detectar i corregir ràpidament possibles problemes al nou codi, sense haver de perdre temps en que un dels revisors el llegeixi sencer.

#### **4.11. Entorns integrats de desenvolupament**

Per tal de desenvolupar el nostre projecte, utilitzarem dues de les eines més completes que hi actualment per a treballar amb Java i, sobretot, amb android: Android Studio i IntelliJ IDEA.

##### **4.11.1. Android Studio**

Per a realitzar l'aplicació mòbil que es connectarà amb el nostre servidor, utilitzarem Android Studio. Aquest IDE, especialitzat en aplicacions android, disposa d'un conjunt d'eines per a realitzar tant la interfície gràfica de l'aplicació com la lògica d'aquesta, de forma senzilla i fàcilment usable.

Per una banda, les aplicacions android utilitzen XML per tal de dissenyar l'aparença de les pantalles. Malgrat tot, aquest llenguatge pot ser difícil d'utilitzar, sobretot si no s'hi està acostumat. Per aquest motiu, aquest entorn de desenvolupament ofereix la possibilitat de dissenyar-les utilitzant un entorn gràfic, en el qual els components es poden col·locar arrossegant-los a la posició desitjada dins d'un contenidor d'elements i es poden modificar cadascuna de les seves propietats fàcilment.

Per l'altra banda, per tal de dissenyar la lògica de l'aplicació, aquest IDE disposa d'unes eines eficients per a mantenir el codi organitzat, llegible i eficient. A més, es poden afegir eines de testeig, com per exemple JUnit, de forma senzilla i s'ofereix una vista de testeig pròpia per aquestes.

Finalment, a part de poder desenvolupar l'aplicació, aquest entorn de desenvolupament es pot sincronitzar amb una eina de control de versions, com per exemple GitHub. Des del mateix programa, és possible crear noves branques, fer *commits* i *pushes* al repositori i iniciar els *pull request* per tal de realitzar la fusió entre les branques del projecte. D'aquesta manera no és necessari utilitzar un intèrpret de comandes per tal de realitzar aquestes accions en el repositori.

#### 4.11.2. IntelliJ IDEA

En el nostre projecte, tota la lògica es troba en el nostre servidor web. Per tal de configurar-lo, utilitzarem IntelliJ IDEA, un dels IDEs més destacats actualment. Aquest disposa d'un gran conjunt d'integracions amb els *frameworks* més utilitzats, entre ells Spring, el qual utilitzarem per a desenvolupar el servidor. A més, com que és dels mateixos creadors que Android Studio, segueix un mètode de funcionament similar.

L'entorn de desenvolupament integrat d'IntelliJ IDEA, disposa d'un conjunt d'eines per tal de millorar la qualitat del codi, les quals es troben també disponibles a l'Android Studio. A més, degut a la seva integració amb Spring podem gestionar el servidor i actualitzar-lo si és convenient. A més, també disposem de diverses opcions per a connectar-se amb el sistema de control de versions, és a dir, amb el nostre repositori de GitHub específic pel servidor. D'aquesta manera, podem aplicar la metodologia de treball del Gitflow sense cap dificultat.

#### 4.12. Comunicació

Durant la realització del projecte, és important que tots els membres estiguem en contacte. Per tant, hem decidit utilitzar un conjunt d'eines de comunicació específiques per a determinades situacions.

Per una banda, al principi del projecte hem utilitzat sobretot el WhatsApp, ja que és l'eina més còmoda per comunicar-se actualment. Malgrat tot, hem començat a utilitzar Slack, una aplicació que permet crear canals de text independents per a cadascun dels equips que formen el projecte, a part del canal general. En el nostre cas, hem creat un canal específic pel *front end* i un altre pel *back end*. Aquests canals, més el general, estan sincronitzats als seus respectius repositoris de GitHub, al Codacy i al

Taiga, permetent doncs estar informats de qualsevol canvi que es produeixi en aquests.

Per l'altre banda, per a la realització de reunions virtuals hem decidit utilitzar dues eines diferents, en funció de la finalitat de la reunió. Si en la reunió ha de participar el *product owner*, aleshores fem servir Skype, en el qual tenim un grup específic per a parlar amb ell. En canvi, si la reunió és per avançar en el desenvolupament de l'aplicació aleshores utilitzem Discord, en el qual hem creat un servidor amb diferents canals de veu en funció de l'equip. D'aquesta manera, els diferents equips poden parlar sense que l'altre estigui de fons i si s'ha de tractar alguna qüestió de forma general disposem del canal general i d'altres canals secundaris per a grups reduïts.

#### 4.13. Gestió de bugs (Nou)

En el moment de detectar un *bug* ja sigui en l'aplicació per Android o en el servidor es necessari obrir una *issue* al GitHub. Atès que el Taiga està sincronitzat amb GitHub, la *issue* també s'obrirà en aquest; malgrat tot, és necessari accedir-hi per tal d'assignar el revisor de la *issue* i establir el tipus, la prioritat i la gravetat d'aquesta.

Un cop la *issue* ha estat especificada correctament, el revisor obre una branca del tipus *hotfix* des de *master*. En aquesta es realitzen les modificacions necessàries per solucionar el *bug* detectat i, en el cas de que sigui possible, s'actualitza la versió de l'aplicació i s'obren dos *pull requests* a GitHub: un a *master* i l'altre a *develop*. Un cop els dos han estat aprovats, es prossegueix a eliminar la branca creada i es tanca la *issue* al GitHub i, de forma automàtica, al Taiga.

## 5. Descripció tècnica

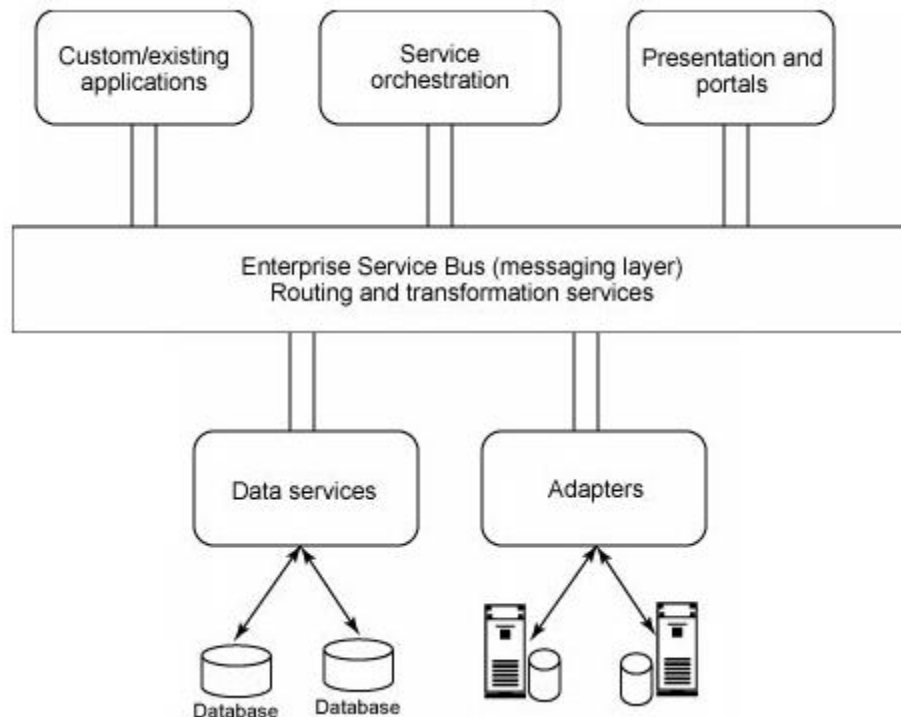
En aquest apartat es dona una explicació dels patrons i arquitectures utilitzades per realitzar el projecte.



### 5.1. Concepció general de l'arquitectura

La nostra arquitectura es basa en la utilització d'APIs que nosaltres mateixos generem en la part del servidor i també aquelles que ens proporciona Google, Whatsapp o Facebook. Sobretot utilitzarem la API de Google ja que volem sincronitzar Google Calendar amb el nostre usuari i també farem ús de Google Maps. Aquesta arquitectura orientada a serveis ens permet ser més flexibles, poder reutilitzar més fàcilment i també reduir l'acoblament ja que cada API funciona de manera independent. També ens permet modificar, quan veiem necessari, més fàcilment alguna part del nostre projecte. Creiem que el fet d'utilitzar aquesta arquitectura no només ens serà útil per desenvolupar la nostra aplicació, sinó que també serà de gran utilitat per poder col·laborar amb els altres projectes, tant per a que un altre grup pugui integrar en el seu projecte una de les nostres APIs com per a que puguem integrar nosaltres un servei d'un altre grup. Un altre aspecte a destacar és la rapidesa que ens atorga aquesta arquitectura ja que és un aspecte que els usuaris valoren positivament ja que millora la experiència d'usuari.

Per la comunicació entre la interfície i la base de dades utilitzarem una arquitectura de tipus ESB (*Enterprise service bus*), que unifica les peticions de l'usuari cap a la base de dades a través de les APIs. ESB consisteix en una integració distribuïda gràcies a la utilització de contenidors de serveis. Amb aquesta arquitectura aconseguim crear un intermediari entre les diferents plataformes i APIs de la nostra app, fent així la comunicació molt més fàcil, ja que el propi bus s'encarrega de traduir les peticions per a cada servei (la part del bus que realitza aquestes transformacions es coneix com un adaptador), encara que vinguin d'un que inicialment no sigui compatible. Un altre avantatge que ofereix ESB, és que els diferents serveis i APIs es connecten directament al bus i no entre elles, simplificant i reduint les unions entre elles i fent que sigui molt més fàcil la seva utilització.



**Figura 5. Arquitectura Enterprise Service Bus**

Altres avantatges que ofereix l'arquitectura ESB és la simplicitat de l'encaminament, la seguretat en el lliurament de missatges, el suport per a protocols tant síncrons com asíncrons i també una coordinació de serveis d'aplicació múltiples encapsulats con un de sol.

Si utilitzem aquesta arquitectura es per aprofitar els beneficis que ens ofereix, com per exemple l'acomodació de sistemes existents molt més ràpida, la creació de serveis *ready-to-use* i la facilitat d'exportació a altres aplicacions.

## 5.2. Diagrames arquitectònics

A continuació veurem el disseny físic del nostre projecte, el qual es divideix en tres capes. En el front-end, es troba l'aplicació per a dispositius Android que es comunicarà via peticions HTTP amb el servidor. A continuació, ens trobem amb el mid-tier, on es troba ubicat el nostre servidor el qual farà d'adaptador per a resoldre les peticions cap als diferents serveis que s'ofereixen als usuaris, tant els nostres propis com el de tercers com Google o Facebook. Finalment arribem al *back-end*, on es troba ubicada la nostra base de dades, contra la qual el servidor realitzarà les diferents consultes o modificacions que calgui segons les peticions que rebí.

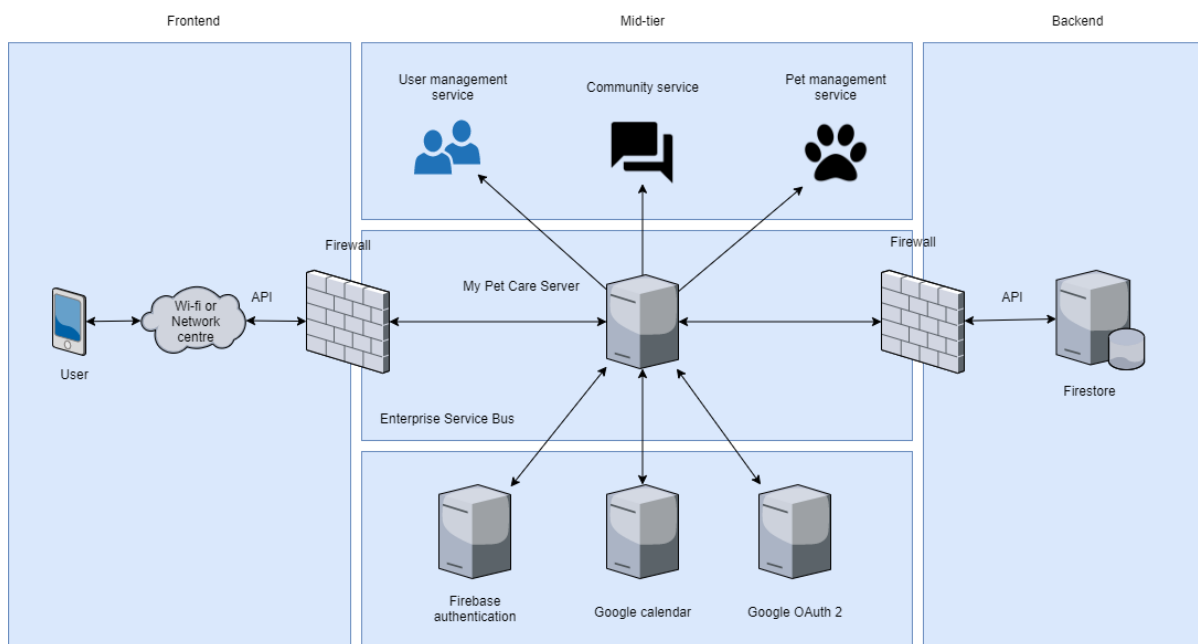


Figura 6. Representació de la capa física

Acte seguit, es mostra el diagrama de components del projecte en el qual es poden observar amb més detall la comunicació entre el dispositiu i els diferents serveis que hem desenvolupat en el projecte.

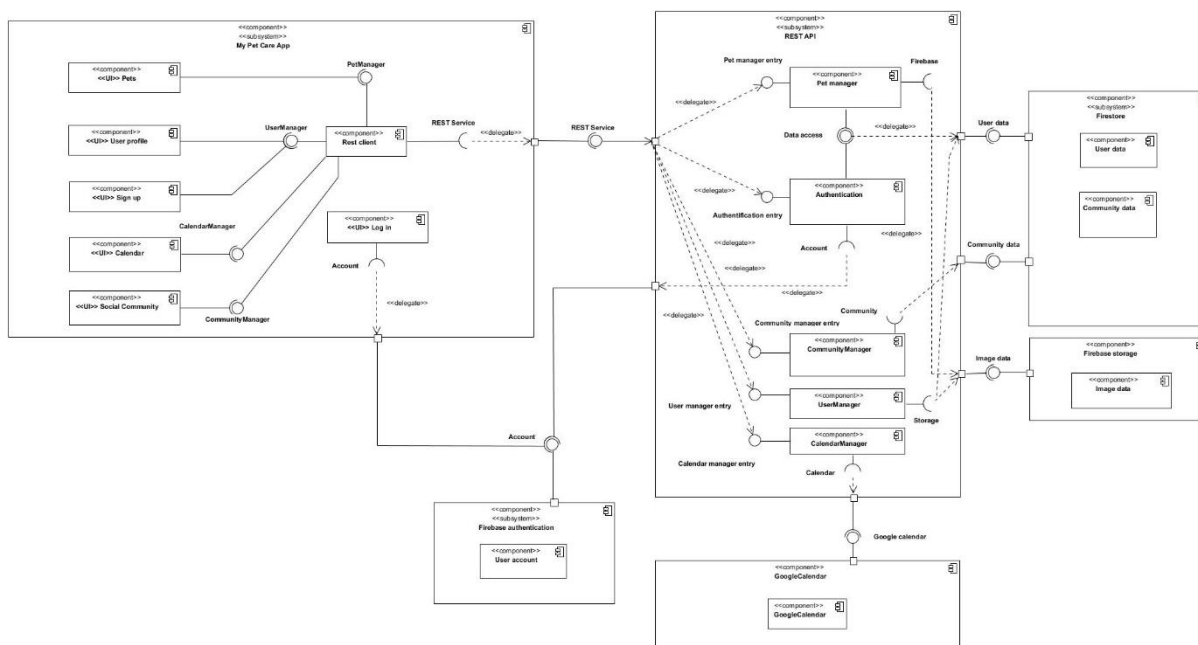


Figura 7. Diagrama de components

### 5.3. Patrons aplicats

A continuació es descriuen els patrons aplicats en el projecte fins al moment.

#### 5.3.1. Factoria abstracta

El patró factoria abstracta és un patró de disseny per el desenvolupament software que soluciona el problema de crear diferents famílies d'objectes.

S'aplica quan:

- Es preveu que s'inclouran noves famílies d'objectes.
- Existeix una dependència entre els tipus d'objectes.

Consisteix en elaborar una interfície per crear famílies d'objectes relacionats sense especificar les seves classes concretes, de la següent forma:

- Factoria abstracta: Defineix les interfícies de les factories i proveeix un mètode per l'obtenció de cada objecte que pugui crear.
- Factoria concreta: Representa les diferents famílies de productes, proveeix la instància concreta que s'encarrega crear.

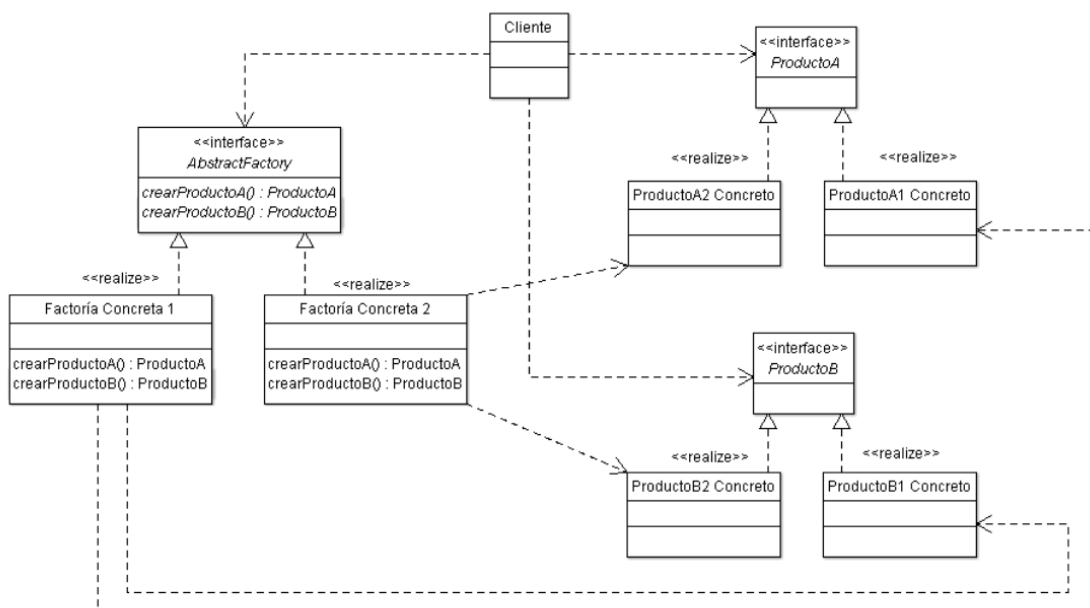


Figura 8. Exemple del disseny UML d'una factoria abstracta

Pel que respecta l'ús, aquest patró s'utilitza per la creació dels controladors de transacció, i l'utilitzem perquè es compleixen els requisits per utilitzar-lo, es preveu que s'inclouran noves famílies d'objectes i existeix una dependència entre els tipus d'objectes, i per tant ens facilita la creació de diferents famílies d'objectes.

### 5.3.2. Factoria simple

És una simplificació del patró de Factoria abstracta. Consisteix en utilitzar una classe constructora abstracta amb uns quants mètodes definits i d'un altre abstracte dedicat a la construcció d'objectes d'un subtipus d'un tipus determinat.

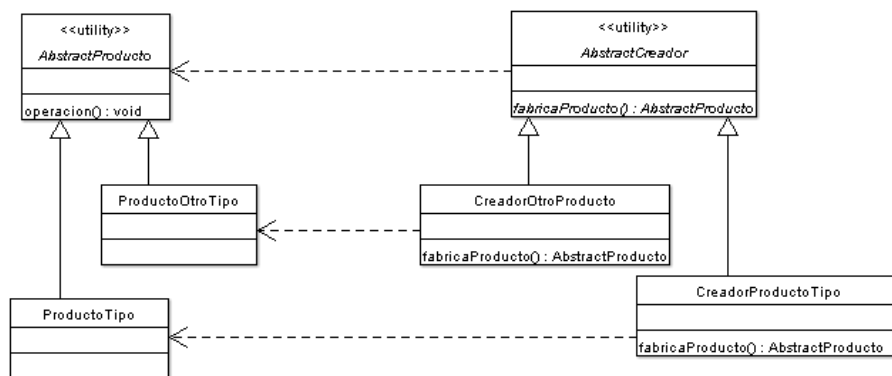


Figura 9. Exemple del disseny UML d'una factoria simple

A la nostra aplicació fem ús d'aquest patró per crear instàncies dels serveis Firebase.

### 5.3.3. Adaptador

El patró adaptador ens permet que dues classes amb diferents interfícies puguin treballar de manera conjunta a partir de la creació d'un objecte que les comunicarà i per tant, que permetrà que s'utilitzin els mètodes de la classe a adaptar.

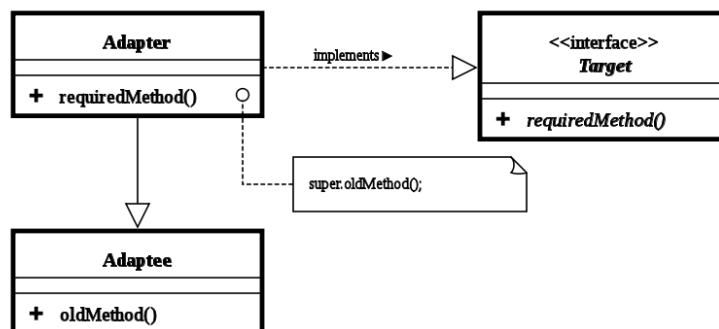


Figura 10. Exemple del disseny UML d'un adaptador



S'aplica quan es vol utilitzar una classe però la seva interfície no concorda amb la que necessitem, o quan es vol reutilitzar una classe.

Pel que respecta a l'ús, aquest patró s'utilitza per realitzar l'accés als serveis i així poder utilitzar les seves funcionalitats sense connectar-nos directament al servidor.

#### 5.3.4. Model, vista, controlador

L'arquitectura Model–Vista–Controlador (MVC) és un patró de disseny utilitzat per la implementació d'interfícies d'usuari. Aquest patró de desenvolupament de programari divideix l'aplicació en tres parts interconnectades: el model de dades, la interfície d'usuari i la lògica de control.

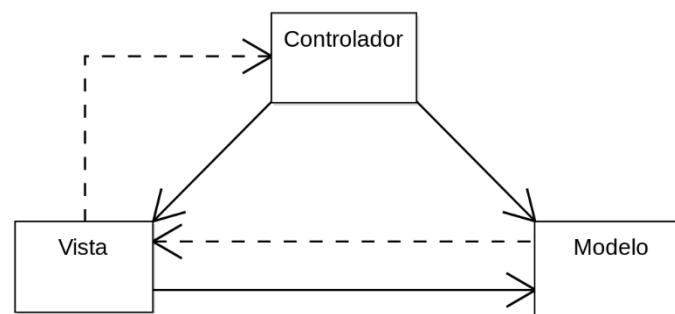
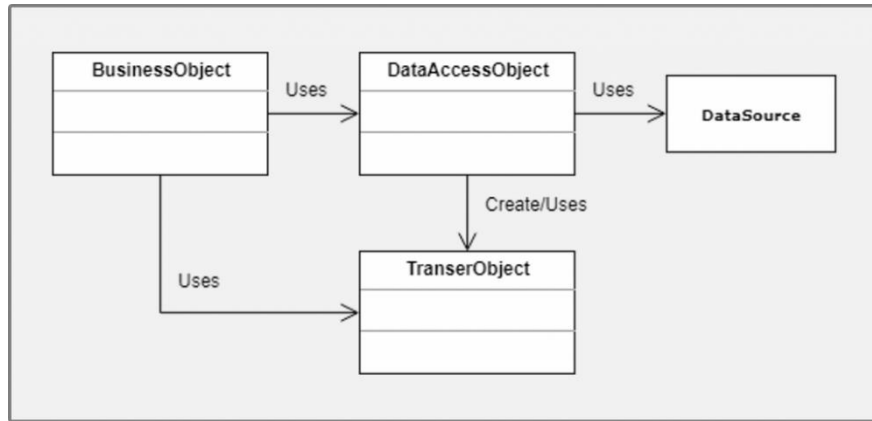


Figura 11. Exemple del disseny UML del MVC

Aquest està basat en les idees de reutilització de codi i separació de conceptes, característiques que busquen facilitar la tasca de desenvolupament d'aplicacions i el seu posterior manteniment, sent aquest el motiu de la seva utilització en el nostre projecte. El patró ha sigut utilitzat en el front-end de l'aplicació ja que és on s'implementa la interfície d'usuari.

#### 5.3.5. DAO

El patró Data Access Object (DAO) proposa separar del tot la lògica de negoci de la lògica per accedir a les dades, d'aquesta manera, el DAO proporcionarà els mètodes necessaris per inserir, actualitzar, esborrar i consultar la informació; d'altra banda, la capa de negoci només es preocupa de la lògica de negoci i utilitza el DAO per interactuar amb la font de dades.



**Figura 12. Exemple del disseny UML del DAO**

L'avantatge d'utilitzar el patró DAO és l'aïllament entre lògica de negoci i la font d'informació (generalment, una base de dades), d'aquesta manera el DAO no requereix coneixement directe del destí de la informació que manipula i la lògica de negoci és independent del tipus de font d'informació utilitzada ja que el DAO s'encarrega d'interactuar amb aquesta. Per aquest motiu hem aplicat aquest patró per l'accés a la base de dades des del servidor.

### 5.3.6. Service Locator

Aquest patró és utilitzat per a encapsular els serveis en una capa abstracta. Rep el nom pel component central *Service Locator* que retorna una instància dels serveis en ser sol·licitades pels clients.

Principalment té 4 components:

1. Service Locator: Abstrau tota la complexitat de fer ús d'un servei i li ofereix al client una interfície senzilla. Això, a més de l'avantatge que dona la senzillesa, permet també al client reutilitzar-lo.
2. InitialContext: És l'objecte del punt de sortida en el procés de cerca i creació. Els proveïdors de serveis proporcionen aquest objecte, que varia depenent del tipus de servei proporcionats.
3. Service Factory: És l'objecte que gestiona el cicle de vida de l'objecte Business Service.

4. Business Service: És un objecte que compleix el rol del servei que el client ha sol·licitat.

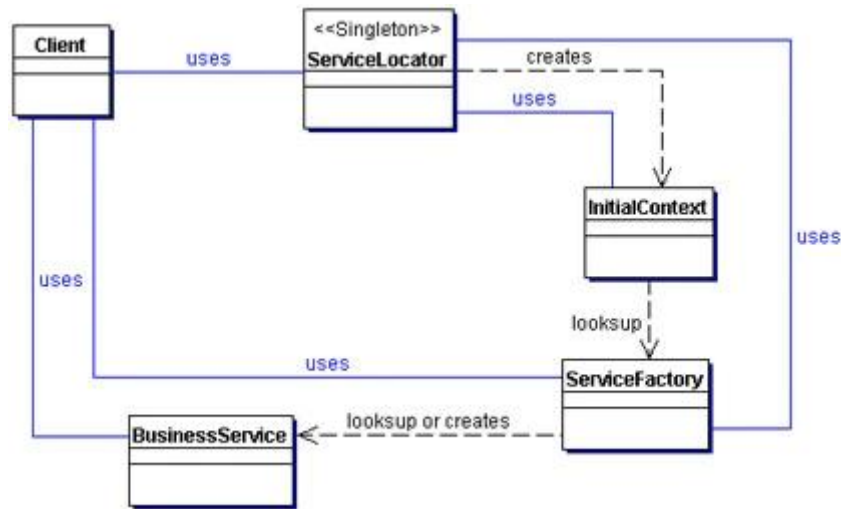


Figura 13. Exemple del disseny UML del Service Locator

Aquest patró és utilitzat per l'aplicació mòbil per accedir als serveis que proporciona el servidor.

### 5.3.7. Singletó

Aquest és un patró que restringeix la creació d'objectes. El seu rol és permetre l'existència de només una única instància de l'objecte i proporcionar una referència global a aquest. Hi ha diverses maneres d'implementar-lo, però la idea principal és la de la creació d'un mètode públic que s'encarregui de cridar a un constructor privat per crear la instància si no existeix cap altre. La classe, a més, disposarà d'un mètode públic que servirà per obtenir aquesta instància.

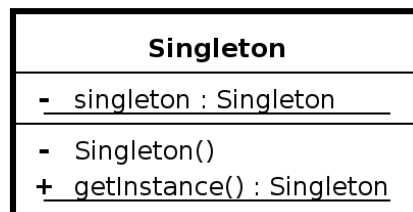


Figura 14. Representació d'un singletó en UML

En la nostra aplicació, aquest patró s'utilitza sobretot per la creació de les factories Abstractes.

### 5.3.8. Expert

Aquest és un patró lligat amb les bones pràctiques de programació que consisteix en assignar responsabilitats a classes. La idea d'aquest patró és analitzar les responsabilitats i assignar-les a la classe corresponent en funció de la informació necessària per a complir-les.

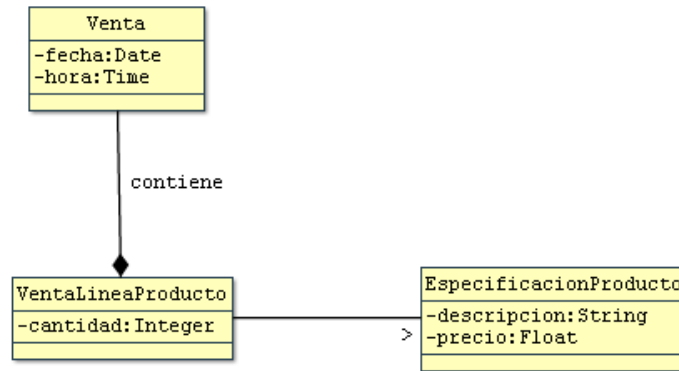


Figura 15. Exemple d'aplicació del patró expert

Aquest patró és utilitzat per totes les classes del nostre sistema per tal de mantenir un software coherent.

### 5.3.9. Controlador de transacció

Aquest patró fa d'intermediari entre la interfície i l'algoritme que implementa. Així, rep les dades de l'usuari i les envia a les diverses classes en funció del mètode cridat. A la nostra aplicació, els nostres controladors són cridats des de front per accedir als serveis.

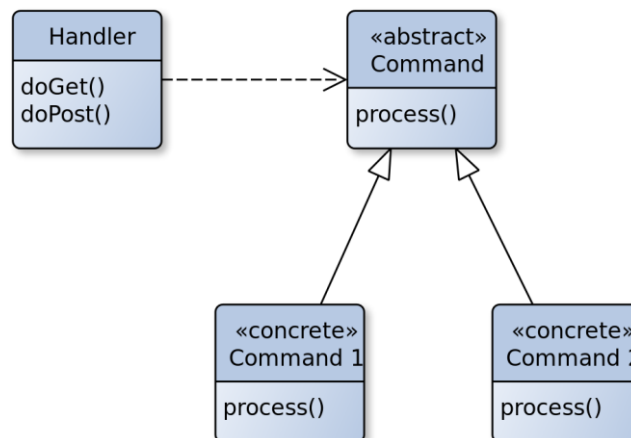


Figura 16. Exemple del disseny UML del controlador de transacció

### 5.3.10. Patró Plantilla (Nou)

El patró plantilla és un patró de disseny de comportament que defineix l'esquelet d'un algoritme en un mètode que, en ser heretat per les subclasses, difereix en alguns dels seus passos.

Un exemple seria el següent:



Figura 17. Patró Plantilla

En un principi, és un mètode dissenyat per a macros, on cadascun implementa parts invariables i deixa oberts certs “placeholders” per a personalitzar les opcions.

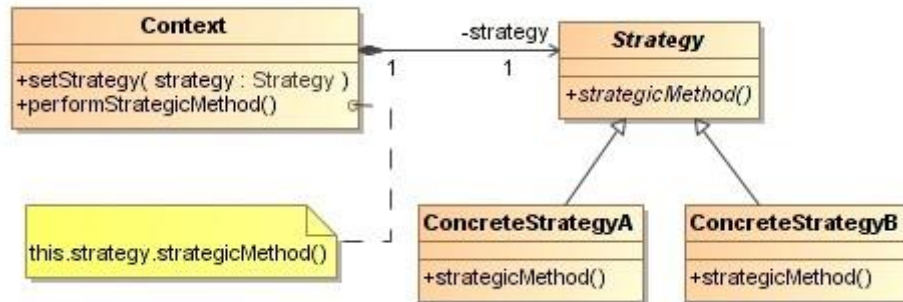
Utilitzar aquest patró aporta una sèrie d'avantatges: primerament, deixa que les classes que s'implementen tinguin un comportament que pugui variar, evita la duplicació de codi i, per últim, permet evitar una sobrecàrrega polimòrfica: en comptes d'heretar el mètode i reescriure'l completament, permet canviar només aquelles parts canviabls.

En el nostre codi, fem ús del patró plantilla per a tal d'accedir a la informació reduïda d'una mascota.

### 5.3.11. Patró estratègia (Nou)

Aquest és una patró de disseny de comportament que ofereix diversos algorismes al client i li dóna la possibilitat de canviar entre ells segons les seves necessitats.

Un exemple:



**Figura 18. Patró estratègia**

El patró estratègia és molt útil per a situacions en les que, per exemple, el client no necessita tots els algorismes en tots els cassos, o si tinguéssim diversos clients fent ús del mateix codi: en comptes de duplicar-lo, oferim estratègies diferents a cadascun.

A la nostra aplicació, fem us del patró estratègia per a alternar entre els diferents estadístics del barchart, al Front End.

### 5.3.12. Patró observador (Nou)

El patró observador és un patró de disseny utilitzat en programació d'ordinadors per a observar l'estat d'un objecte en un programa. Està relacionat amb el principi d'invocació implícita.

Aquest patró es fa servir principalment per a implementar sistemes de tractament d'esdeveniments distribuïts. En alguns llenguatges de programació, els problemes tractats per aquest patró, són tractats a la sintaxi de tractament d'esdeveniments nativa. Aquesta és una funcionalitat molt interessant en termes de desenvolupament d'aplicacions en temps real.

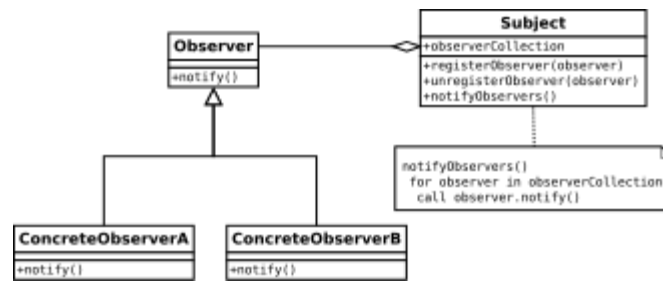


Figura 19. Patró observador

L'essència d'aquest patró és que un o més objectes (anomenats observadors o escoltadors) són registrats (o es registren ells mateixos) per a observar un esdeveniment que pot ser llençat per l'objecte observat (el subjecte). L'objecte que pot llençar un esdeveniments generalment manté una col·lecció d'observadors.

En el nostre cas utilitzem aquest patró per escoltar els canvis en la base de dades, més concretament per escoltar canvis en els posts dels fòrums.

#### 5.4. Model conceptual de les dades (Actualitzat)

A continuació es mostren els diagrames del model de dades implementat fins al moment i el que es pretén obtenir del projecte finalitzat.

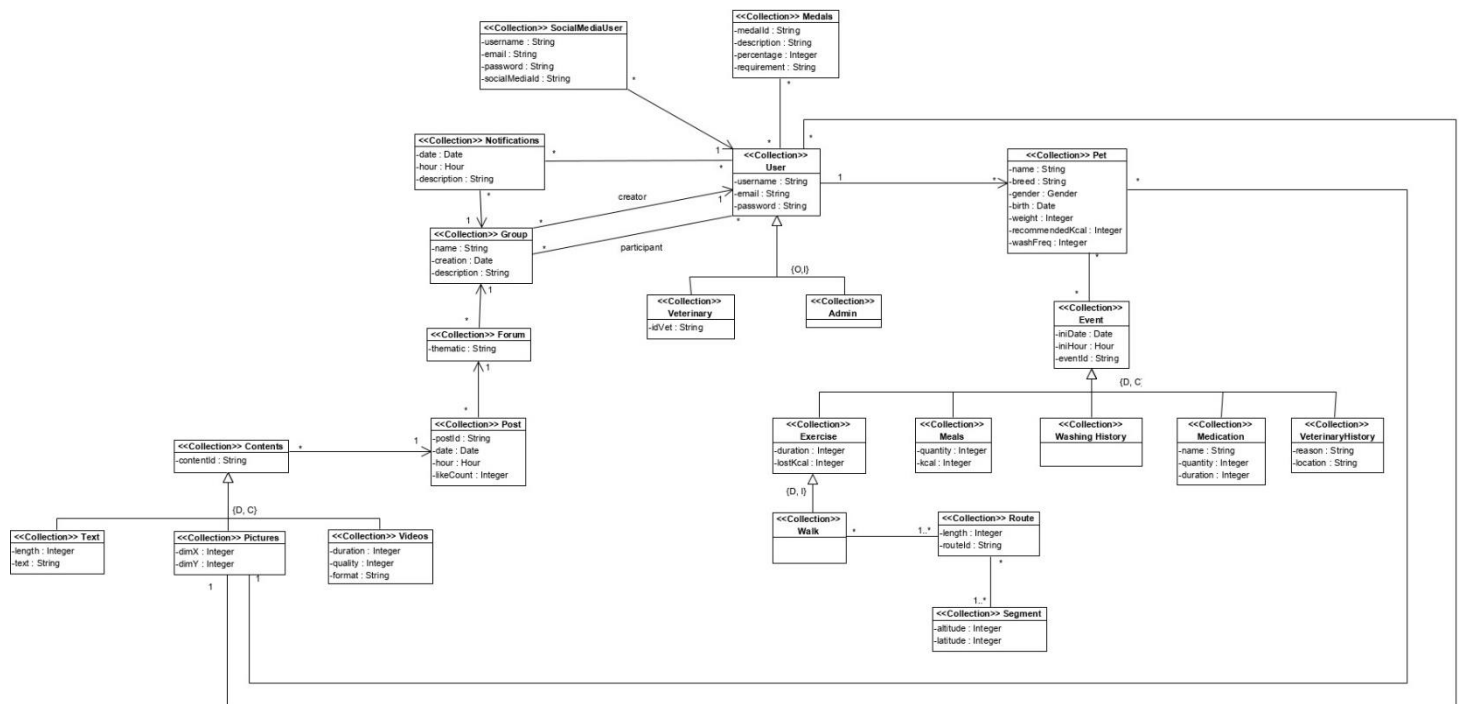


Figura 20. Model de dades final (Original)

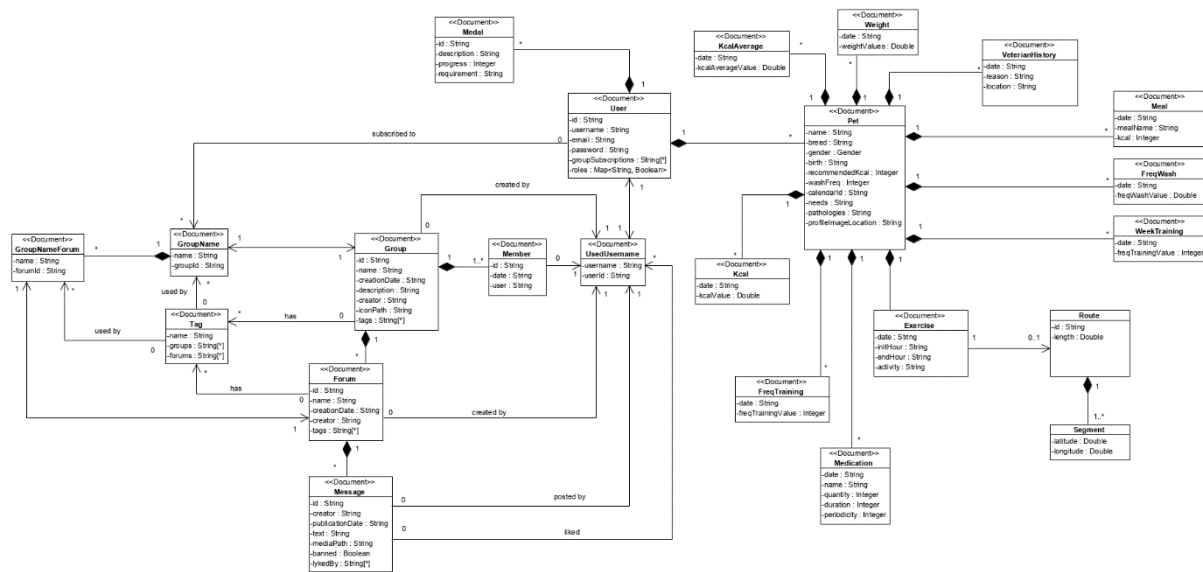


Figura 21. Diagrama de classes final (Nova versió)



Figura 22. Diagrama de classes actual (Original)

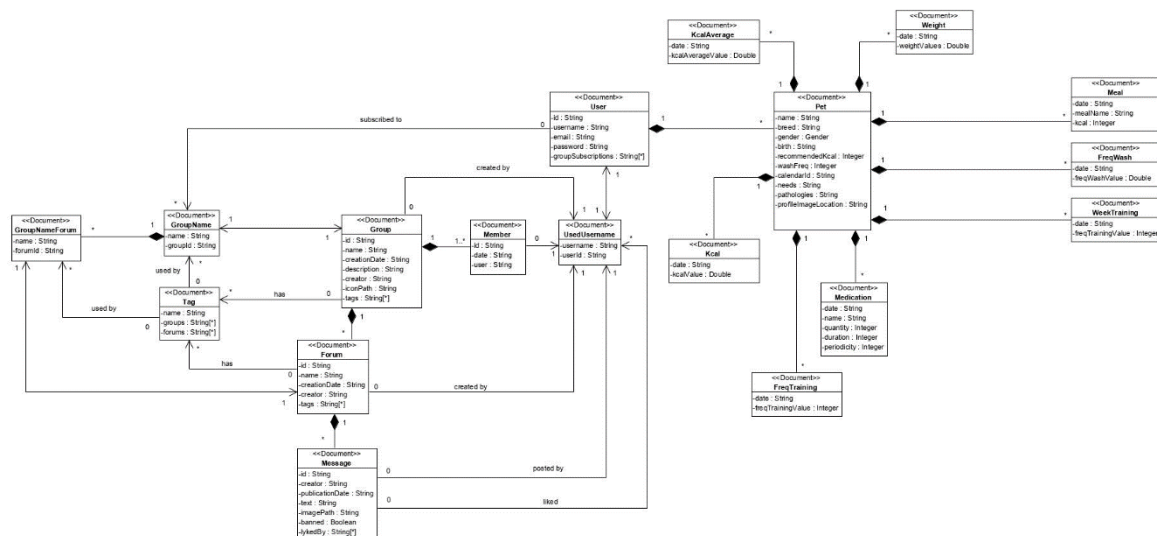


Figura 23. Diagrama de classes actual (Nova versió)



## 5.5. Altres aspectes tecnològics

A continuació es fa una breu explicació dels diferents aspectes tecnològics inclosos en el projecte.

### 5.5.1. Servidors

Disposem de dos servidors, un públic ubicat a Heroku i un altre de privat proporcionat per la FIB. El servidor públic s'utilitza per llençar les versions funcionals de l'aplicació, ja que aquest serà al que accediran els usuaris. D'altra banda, tenim el servidor de la FIB, on es van implementant les noves funcionalitats i es testegen abans de porta-les al servidor públic.

### 5.5.2. Base de dades

Utilitzem una única base de dades de tipus NoSQL gestionada per Firebase, que es un conjunt de serveis de *back-end* que proporciona Google. El versionat d'aquesta es fa mitjançant el servei de Firestore, un servei ofert per Firebase, per tant, és totalment transparent a nosaltres i se'n encarrega exclusivament el servei de Firestore.

Tenint en compte el temps del que disposem per fer el projecte, hem escollit aquesta solució degut a la seva facilitat d'implementació respecte de les altres alternatives. Un altre motiu és el nostre enfoc dels accessos a dades de la nostra aplicació. Degut a que una de les principals característiques de la nostra aplicació seran els fòrums, la nostra aplicació tindrà una alta càrrega de lectures de la base de dades. Per aquest motiu, considerem que una base de dades NoSQL ens oferirà un major rendiment a l'hora de realitzar aquestes peticions.

### 5.5.3. Nombre de llenguatges

Per a la creació de l'aplicació Android utilitzem Java i XML, Java per la lògica de l'aplicació i XML per la interfície, mentre que per a la implementació del servidor s'utilitza únicament Java.

#### 5.5.4. APIs

Cadascuna de les nostres principals funcionalitats conformarà una API, facilitant la comunicació entre les diferents parts de l'aplicació i també agilitzant el desenvolupament. Actualment disposem de dos APIs, una per a la gestió dels usuaris i una per a la gestió de les mascotes. Cal afegir, que actualment ambdues es troben empaquetades en una sola llibreria.

#### 5.5.5. Frameworks

Primerament, utilitzem Spring Boot per crear el nostre servidor i les API REST ja que és un dels frameworks més utilitzats a Java i facilita tota la creació d'aquest.

En segon lloc, utilitzem JUnit per a la realització dels tests unitaris. Aquest ens permet testejar els diferents escenaris que es poden produir en l'execució dels mètodes d'una classe.

En tercer lloc, utilitzem Mockito per a la realització dels tests unitaris, que ens proporciona un conjunt de mètodes per a la implementació de mocks, facilitant així l'aïllament dels tests.

Finalment, utilitzem Espresso per a la realització dels tests instrumentals de l'aplicació Android, ja que ens proporciona un conjunt de mètodes per a simular les accions d'un usuari i automatitzar-les.

#### 5.5.7. Desplegament

Per al desplegament del servidor fem servir Heroku, un servei de *hosting* que ens proporciona servidors en el núvol. Aquest ens permet enllaçar el repositori que conté la implementació del servidor, facilitant així que es puguin desplegar noves versions del servidor de manera automàtica, un cop ha passat tots els tests de GitHub Actions i Codacy.