



## MY PET CARE

### SPRINT – 1



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	Master	santiago.del.rey	santi.delrey	santidrrj
Hernando, Enric	Developer	enric.hernando	enrichernandopuerta	enrichp
Pinto, Albert	Developer	albert.pinto	apintogil	AlbertPG
Simó, Marc	Developer	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.....	1
1.2.1. Albert Pinto i Gil .....	1
1.2.2. Álvaro Trius Béjar.....	2
1.2.3. Daniel Clemente Marín.....	2
1.2.4. Enric Hernando Puerta.....	2
1.2.5. Marc Simó Guzmán .....	2
1.2.6. Oriol Catalán Fernández .....	3
1.2.7. Santiago Del Rey Juárez.....	3
1.2.8. Xavier Campos Díaz .....	3
1.3. Avaluació dels companys .....	4
1.4. Sprint master report .....	5
2. Requisits .....	7
2.1. Product Backlog.....	7
2.1.1. Gestió de mascotes i usuaris .....	7
2.1.2. Salut de la mascota.....	8
2.1.3. Assistència a l'exercici .....	8
2.1.4. Establiments especialitzats .....	9
2.1.5. Interacció en comunitats temàtiques.....	9
2.1.6. Interacció amb xarxes socials .....	10
2.1.7. Interacció amb el calendari .....	10
2.1.8. Gestió de medalles i recompenses .....	11
2.1.9. Aspectes tecnològics (Nou).....	11
2.1.10. Requisits no funcionals (Nou) .....	12



---

2.2. Requisits no funcionals .....	12
2.2.1. Rapidesa .....	12
2.2.2. Disponibilitat .....	13
2.2.3. Confidencialitat .....	13
2.3. Tractament d'aspectes transversals .....	14
2.4. Serveis de tercers .....	14
2.4.1. Firebase .....	14
3. Cerimònies àgils .....	16
3.1. Sprint planning .....	16
3.2. Sprint review .....	17
3.3. Sprint retrospective .....	18
3.3.1. Que s'ha fet bé i es mantindrà? .....	18
3.3.2. Què no ha funcionat? .....	19
3.4. Gràfiques .....	20
3.4.1. Sprint burndown .....	20
3.4.2. Release burndown .....	20
3.4.3. Velocitat de l'equip .....	21
3.4.4. Dedicació mensual .....	21
4. Metodologia .....	22
4.1. Metodologia àgil .....	22
4.2. Definició de finalització .....	23
4.3. Gestió del projecte .....	23
4.4. Convencions .....	24
4.5. Gitflow .....	26
4.6. Testeig .....	27
4.7. GitHub Actions .....	28
4.8. Dependabot (Nou) .....	29

---

4.9. GitGuardian (Nou) .....	29
4.10. Codacy .....	29
4.11. Entorns integrats de desenvolupament (Nou).....	30
4.11.1. Android Studio.....	30
4.11.2. IntelliJ IDEA (Nou).....	31
4.12. Comunicació (Nou) .....	32
5. Descripció tècnica .....	33
5.1. Concepció general de l'arquitectura .....	33
5.2. Diagrames arquitectònics .....	34
.....	35
5.3. Patrons aplicats .....	35
5.3.1. Factoria abstracta .....	35
5.3.2. Factoria simple.....	37
5.3.3. Adaptador.....	37
5.3.4. Model, vista, controlador .....	38
5.3.5. DAO .....	38
5.3.6. Service Locator .....	39
5.3.7. Singletó .....	40
5.3.8. Expert.....	41
5.3.9. Controlador de transacció .....	41
5.4. Model conceptual de les dades .....	42
5.5. Altres aspectes tecnològics .....	43
5.5.1. Servidors .....	43
5.5.2. Base de dades .....	43
5.5.3. Nombre de llenguatges .....	43
5.5.4. APIs .....	44
5.5.5. Frameworks .....	44

5.5.6. Integració contínua.....	44
5.5.7. Desplegament.....	45

## ÍNDIX DE FIGURES

Figura 1. Sprint burndown de la primera iteració.....	20
Figura 2. Release burndown .....	20
Figura 3. Representació de la velocitat de desenvolupament de l'equip.....	21
Figura 4. Hores de dedicació mensual del membres de l'equip .....	21
Figura 5. Representació de la capa física .....	34
Figura 6. Diagrama de components .....	35
Figura 7. Exemple del disseny UML d'una factoria abstracta.....	36
Figura 8. Exemple del disseny UML d'una factoria simple .....	37
Figura 9. Exemple del disseny UML d'un adaptador .....	37
Figura 10. Exemple del disseny UML del MVC .....	38
Figura 11. Exemple del disseny UML del DAO .....	39
Figura 12. Exemple del disseny UML del Service Locator .....	40
Figura 13. Representació d'un singletó en UML.....	40
Figura 14. Exemple d'aplicació del patró expert.....	41
Figura 15. Exemple del disseny UML del controlador de transacció.....	41
Figura 16. Model de dades final .....	42
Figura 17. Model de dades implementat actualment.....	42


## ÍNDEX DE TAULES

Taula 1. Valoracions entre companys .....	4
Taula 2. Històries d'usuari del tema: Gestió de mascotes i usuaris .....	7
Taula 3. Històries d'usuari del tema: Salut de la mascota .....	8
Taula 4. Històries d'usuari del tema: Assistència a l'exercici.....	8
Taula 5. Històries d'usuari del tema: Establiments especialitzats .....	9
Taula 6. Històries d'usuari del tema: Interacció en comunitats temàtiques .....	9
Taula 7. Històries d'usuari del tema: Interacció amb xarxes socials .....	10
Taula 8. Històries d'usuari del tema: Interacció amb el calendari.....	10
Taula 9. Històries d'usuari del tema: Gestió de medalles i recompenses .....	11
Taula 10. Històries d'usuari del tema: Aspectes tecnològics.....	11
Taula 11. Històries d'usuari del tema: Requisits no funcionals.....	12

## 1. Introducció

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

### 1.1. Resum executiu del sprint

En aquest primer *sprint* del projecte, hem establert uns objectius els quals han estat negociats amb el *product owner*. Aquestes consisteixen en la implementació de les operacions bàsiques de creació, consulta, modificació i eliminació d'usuaris i de les seves mascotes. El baix nombre d'històries proposades és degut al fet que s'han de crear tots els recursos necessaris per a dur a terme el projecte, entre els quals en destaca el nostre servidor web i la base de dades 

Per tal d'assolir aquests objectius, l'equip de desenvolupament es dividirà en dos equips gestionats de manera independent. Malgrat tot, el fet que es treballa en paral·lel no implica que no hi hagi una comunicació habitual amb els membres de l'altre equip. Un dels equips s'encarregarà del disseny i implementació de la interfície gràfica de l'aplicació, la qual realitza les peticions de l'usuari al servidor. L'altre equip, implementa el servidor web, la base de dades i la comunicació entre aquests dos aspectes. Cal destacar que la mida dels equips pot variar en funció de les necessitats d'un dels equips i que aquests no seran els mateixos durant tot el projecte.



### 1.2. Breu descripció del treball individual

#### 1.2.1. Albert Pinto i Gil

Durant aquesta primera iteració, he estat en l'equip del front, dissenyant i implementant l'aplicació per Android. En concret, m'he centrat en la implementació del *navigation drawer*, és a dir, el menú lateral, el registre de mascotes i la gestió de les imatges de les mascotes. A més, he col·laborat en el disseny global de l'aplicació i he ajudat els meus companys d'equip amb els problemes i dubtes que els han sorgit sobre la realització de tests i, sobretot, del TDD.

### 1.2.2. Álvaro Trius Béjar

Durant el transcurs d'aquesta iteració he estat a l'equip del *back*, dissenyant i implementant les llibreries utilitzades per les APIs. Concretament, d'una banda, m'he dedicat a l'aprenentatge del funcionament dels diversos sistemes de *back* i, per l'altra, juntament amb l'Oriol Catalán, m'he encarregat de dissenyar les llibreries de gestió d'usuari i de mascota que serveixen per a que a front es puguin realitzar fàcilment les crides HTTP a la nostra base de dades.

### 1.2.3. Daniel Clemente Marín

En aquesta iteració he format part de l'equip de front, i per tant, he dissenyat i desenvolupat la interfície i les transaccions de varies històries d'usuari. En concret les meves històries assignades han sigut la consulta de mascotes, l'esborrament d'una mascota i la modificació de les dades de l'usuari.

### 1.2.4. Enric Hernando Puerta

Durant aquesta primera iteració, he estat en l'equip del front, dissenyant i implementant l'aplicació per Android. Concretament me encarregat de dissenyar, juntament amb Xavier Campos, la interfície de *Log In* i *Sign Up*, a més de la lògica d'aquesta primera de manera individual. També he estat el responsable del desenvolupament del menú de *settings*, del *logout*, del *update* de mascota, de modificar la contrasenya dels usuaris i de la verificació via email de la creació de comptes. Finalment també he creat diferents transaccions mitjançant TDD i he fet una petita part de la documentació.

### 1.2.5. Marc Simó Guzmán

Durant aquesta primera iteració, he estat en l'equip del *back*, dissenyant i implementant el servidor. Concretament, d'una banda m'he centrat en l'aprenentatge dels diversos sistemes i *frameworks* utilitzats en *back*, i per l'altra, en l'elaboració i implementació de les diferents funcionalitats del servidor, el DAO (*Data Access Object*) que defineix i estableix els diferents accessos a la base de dades des del servidor i la realització i implementació de les APIs utilitzades per entendre i tractar les peticions rebudes pel servidor.



### 1.2.6. Oriol Catalán Fernández

Durant aquesta iteració, he estat treballant en el desenvolupament del *back-end*. Més en concret, he estat treballant i implementant, juntament amb el meu company Álvaro, una llibreria que permet connectar les peticions a servidor amb les de la interfície. Aquesta implementació ens ha portat més temps del previst, ja que ni l'Álvaro ni jo havíem fet mai una llibreria d'aquestes característiques, així que la part de recerca d'informació i d'enteniment del treball a realitzar ha sigut més lent del esperat; això si, un cop ja havíem tingut clar que estàvem fet, la feina ha anat més rodada.

### 1.2.7. Santiago Del Rey Juárez

Durant aquesta iteració he format part de l'equip de *back*, dissenyant i implementant el servidor. Concretament he estat l'encarregat d'implementar la comunicació entre la Rest API i la base de dades per obtenir la informació dels usuaris, així com d'implementat les operacions que permeten enregistrar un nou usuari. També he estat donant suport als meus companys quan sorgien dubtes sobre com realitzar certes tasques. Finalment, com a *sprint master*, he hagut de distribuir i organitzar la feina entre els meus companys per tal de complir amb el termini d'entrega.

### 1.2.8. Xavier Campos Díaz

Durant aquest *sprint* he format part de l'equip de *front-end*. Concretament me encarregat de dissenyar, juntament amb Enric Hernando, la interfície de *Log In* i *Sign Up*, a més de la lògica d'aquesta última de manera individual.

També he estat el responsable del desenvolupament del menú principal, de la baixa de l'usuari i de la interfície del canvi d'imatge de perfil de l'usuari.

Finalment també he realitzat la traducció de l'aplicació al català i una petita part de la documentació.

### 1.3. Avaluació dels companys

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. 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. 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.

	Sprint 1	Sprint 2	Sprint 3	Sprint 4	Mitjana
<b>Albert Pinto</b>	9.29				9.29
<b>Álvaro Trius</b>	6.71				6.71
<b>Daniel Clemente</b>	7.71				7.71
<b>Enric Hernando</b>	8.57				8.57
<b>Marc Simó</b>	9.00				9.00
<b>Oriol Catalán</b>	7.14				7.14
<b>Santiago Del Rey</b>	9.57				9.57
<b>Xavier Campos</b>	8.71				8.71




Taula 1. Valoracions entre companys

## 1.4. Sprint master report

En aquest primer *sprint* hem iniciat la primera fase del desenvolupament del projecte. Tal y com es va acordar amb el *product owner*, en aquest *sprint* hem implementat les operacions necessàries per a la creació d'usuaris, així com de les seves mascotes.


El primer dia vam decidir fer una distribució del treball on quatre membres de l'equip es dedicarien a implementar la part del *front-end* i els altres quatre el *back-end*, d'aquesta manera podríem treballar en paral·lel aprofitant més el temps. Aquest mateix dia, es va repartir de manera equitativa la documentació a realitzar per aquest *sprint*. Per tal de tenir-la acabada per a la data d'entrega, però sense que suposés massa càrrega de treball de cop, es van dividir els punts a fer en dos grups. Així doncs, el primer grup de documentació es va realitzar al llarg de la primera setmana i el segon durant la segona, de manera que quedés completa abans de terminar la iteració, això si, sense comptar els punts que estrictament s'han de realitzar l'últim dia de la iteració, com són el *sprint review* i *retrospective*.

Durant aquestes tres setmanes, per tal de treure-li avantatge a la situació de confinament en la que ens trobem, hem estat realitzant reunions periòdiques mitjançant videotrucades, 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 per treballar en les nostres respectives parts.

En general s'ha fet una bona feina al llarg d'aquest *sprint*, potser però, hem volgut abastar massa històries d'usuari en aquesta primera iteració, i això ha implicat una càrrega de treball més gran de l'esperada per a cada membre, així com haver de passar algunes històries a la següent iteració. Tot i això, tenint en compte que era la nostra primera iteració, no hem estat massa allunyats de l'objectiu que volíem assolir. Per aquest motiu, considero que això ens servirà per poder planificar millor les següents iteracions.

També cal mencionar que ens ha costat bastant començar a veure el nostre treball reflectit en el percentatge de punts realitzats de la iteració, donat que no s'ha fet una gestió massa bona de les tasques, ja que vam començar moltes històries d'usuari alhora en comptes de fer-les d'una en una. Per aquest motiu ens hem mantingut al

voltant del 5% del punts acabats encara que moltes de les històries ja tenien quasi totes les seves tasques tancades, i no ha sigut fins arribar a la tercera setmana on s'ha pogut veure un creixement molt pronunciat del nombre d'històries acabades, arribant així al 88% de punts completats.


Tot i no haver pogut arribar al 100%, considero que no hauria de ser un motiu d'alarma, ja que tenint en compte tot el projecte ja hem assolit el 25% dels punts a realitzar  xò ens indica que hem d'acabar de polir el nostre criteri alhora de decidir el nombre d'històries d'usuari que s'han d'afegir als *sprints*, encara que no anem mal encaminats.

## 2. Requisits

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

### 2.1. Product Backlog

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*

#### 2.1.1. Gestió de mascotes i usuaris

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**

<b>Informació bàsica</b>	<b>Alimentació:</b>	<b>Higiene:</b>	<b>Veterinari:</b>
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**

<b>Gestió de les rutes de passeig:</b>	<b>Control de l'exercici realitzat:</b>
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**

<b>Localització d'establiments especialitzats:</b>	<b>Valoració d'establiments especialitzats:</b>
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**

<b>Gestió de continguts:</b>	<b>Gestió de grups:</b>
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**

<b>Facebook:</b>	<b>Twitter:</b>	<b>Instagram:</b>	<b>WhatsApp:</b>
Iniciar sessió.	Iniciar sessió.	Iniciar sessió.	Compartir fotos de la galeria.
Tancar sessió.	Tancar sessió.	Tancar sessió.	Compartir aplicació.
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 informació de la teva mascota.
Compartir publicacions de fòrums.	Compartir publicacions de fòrums.	Compartir publicacions de fòrums.	
Compartir informació de la teva mascota.	Compartir informació de la teva mascota.	Compartir informació de la teva mascota.	

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

**2.1.7. Interacció amb el calendari**

<b>Avisos personals:</b>	<b>Avisos automatitzats:</b>	<b>Notificacions a l'usuari:</b>
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 (Nou)

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 <i>settings</i>
Menú principal
Preparació accés als serveis

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

### 2.1.10. Requisits no funcionals (Nou)

Aquest tema el vam afegir durant el primer *sprint* ja que encara que el cost assignat a cada història d'usuari és 0, és necessari tenir en compte tots els requisits no funcionals que s'han de seguir durant tot el procés de creació de l'aplicació.

Requisits no funcionals
Rapidesa
Disponibilitat
Confidencialitat

Taula 11. Històries d'usuari del tema: Requisits no funcionals

Algunes de les històries d'usuari que conformen un CRUD estaven originalment dividides en crear, eliminar, modificar i visualitzar durant la primera entrega del *product backlog* i en aquest *product backlog* les hem agrupat en CRUD.

## 2.2. Requisits no funcionals

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.


### 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. 

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

**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. 

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.

**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.

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.

### 2.3. Tractament d'aspectes transversals

En aquest primer *sprint* del nostre projecte ens hem centrat en dos aspectes transversals, el multi-idioma i els *stakeholders* reals. El multi-idioma ja té preparada l'arquitectura i està implementat correctament per anglès, català i castellà en aquest *sprint*. Per altra banda, en els *stakeholders* reals principalment ens hem centrat en nosaltres mateixos, ja que l'aplicació encara no té les funcionalitats que són més importants pel públic en general, ja que ha sigut una iteració molt tècnica, i la situació general actual no és la millor. En un futur ens agradaria que almenys unes 10 persones provessin la nostra aplicació, ens donessin un feedback de qualitat i amb una freqüència d'una setmana. 

Els aspectes transversals que tenim pensats assolir en els següents *sprints* són la geolocalització amb mapes, les xarxes socials fent *login* i podent compartir contingut, els xats amb històric, la gamificació amb trofeus, la refutació amb bloqueig de comptes, el calendari sincronitzat amb el de google i per últim, la web app que en un futur definirem el seu abast. 

### 2.4. Serveis de tercers

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.

Firebase 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.

### 3. Cerimònies àgils

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

Aquest nou Sprint va començar el dilluns 9 de març i està previst que acabarà el 27 de març. En la reunió realitzada el dimecres 11 de març amb el *product owner*, es van tractar els següents punts:

1. Es va revisar tota la documentació i la idea de l'aplicació i es va decidir fer un canvi a la seva arquitectura: s'implementarà una arquitectura de serveis, concretament *enterprise service bus*, i no una de tres capes com es tenia previst al principi. No es va descartar, però, que al final es puguin acabar implementant ambdues, però això és una qüestió que es decidirà en el futur en funció de l'evolució de l'aplicació.
2. Es va decidir quin seria l'objectiu d'aquest *sprint*: elaborar una primera versió de l'aplicació amb les funcionalitats bàsiques de la gestió d'usuaris i de mascotes i la comunicació amb a base de dades.
3. Es va dividir aquest treball en dos equips: quatre membres destinats al front-end i els altres quatre destinats al back-end. Els del front-end s'encarregarien d'implementar les interfícies i la lògica dels CRUDs de mascotes i d'usuaris, repartint equitativament les històries d'usuari; paral·lelament, els de back-end s'encarregarien d'implementar la base de dades, designant un membre, i la lògica del servidor i les APIs, tres membres. No es va descartar, però, que en un futur poguessin haver-hi canvis: així, podria ser possible que algun membre del front-end passés al back-end o a l'inrevés.
4. Es va fer una revisió de la documentació exigida per a aquest nou *sprint* i es va dividir, un altre cop, en quatre equips de dos membres cadascun: un equip s'encarregaria de la redacció del *product backlog* d'aquest *sprint* i de la redacció de l'actual document, un altre de la explicació dels patrons arquitectònics, un altre de la elaboració dels diagrames del model de les dades i, finalment, un altre que s'encarregaria de revisar i actualitzar la metodologia

de treball i de realitzar el resum del sprint. Es va proposar com a objectiu haver acabat aquests documents el dimecres 18 de març.

5. Per tal de complir amb les exigències de la assignatura, es va iniciar una negociació amb l'equip FIBness, encarregats del tema dels esports. Es va acordar la utilització d'un servei sobre l'organització d'esdeveniments.
6. Finalment es va fer una revisió de la metodologia de treball: es va establir Slack com a eina de comunicació principal entre els membres del grup i IntelliJ IDEA i Android Studio com a entorns de desenvolupament. La resta del temps va ser dedicat a instal·lar les eines necessàries i a sincronitzar-les entre elles.

### 3.2. Sprint review

El dia 27 de març es va realitzar la reunió amb el *product owner* per tal de presentar els resultats d'aquest *sprint*.

Es va començar per realitzar una demostració del funcionament de l'aplicació, mostrant les funcionalitats que havien quedat implementades. Durant aquesta van sorgir alguns errors en el funcionament de l'aplicació, però va quedar explicat que ja els teníem detectats i que serien una tasca prioritària al següent *sprint*. Acte seguit vam passar a revisar les històries d'usuari que havien quedat completades per aquesta iteració i quines s'havien de passar a la següent.

Les històries que van quedar completades per aquest *sprint* van ser:

- Alta usuari
- Baixa usuari
- Modificar les dades d'un usuari: contrasenya i e-mail
- Alta mascota
- Baixa mascota
- Consultar mascota
- Modificar les dades d'una mascota: nom, gènere, data de naixement, raça, llistat de patologies, kcal recomanades i freqüència de banys

Les històries que no es van poder completar en aquesta iteració van ser:

- **CRUD d'imatge de perfil d'usuari i de mascota:** al poc de començar la iteració vam veure que aquesta era una història d'usuari massa difícil per al temps i la experiència del moment. Per això vam decidir deixar-la per a més endavant.
- **Canvi de nom d'usuari:** al programar la lògica del servidor, ens vam adonar que degut a la estructura d'aquest, modificar els noms d'usuari implicava una lògica més complexa del que s'esperava. És per això que vam decidir deixar-los per a la següent iteració i centrar-nos en altres aspectes.

### 3.3. Sprint retrospective

El dia 27 de març es va realitzar el *sprint retrospective* per tal de valorar el funcionament de l'equip en aquesta iteració.

#### 3.3.1. Que 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.

### 3.3.2. Què no ha funcionat?

- **Organització de les històries d'usuari:** un dels principals problemes que hem tingut en aquesta iteració és que vam seleccionar massa històries d'usuari, de tal manera que a la *release* només hem pogut entregar un 88% d'aquestes. Això es deu, principalment, a la nostra manca d'experiència treballant en un projecte de desenvolupament del software com és aquest. Encara que saber apreciar les capacitats del grup és una habilitat que som conscients que només s'adquireix amb temps i per tant és ben possible que en la següent iteració cometrem un error de la mateixa natura. Malgrat tot, procurarem ser més prudents a l'hora de decidir les històries d'usuari a desenvolupar.
- **Manca de sincronització entre *front-end* i *back-end*:** A la última setmana va succeir que el grup del *front-end* estava llest per provar la seva part, però el *back-end* encara no havia acabat ni les llibreries ni la lògica del servidor: això va portar a un increment en la pressió del grup del *back-end* i temps perdut per al grup del *front-end*. Al reunir-nos, ens hem adonat que això es deu a que no vam ser massa granulars a l'hora de definir quines històries d'usuari a desenvolupar: només vam decidir quines s'havien de fer i vam intentar fer-les totes de cop. A més, altre cop, degut a la nostra inexperiència, no vam ser totalment conscients de les tasques concretes a desenvolupar fins que vam arribar a l'equador del *sprint*. De cara a la següent iteració, per tal de que hi hagi una millor sincronització entre els dos equips, intentarem seguir el mateix mètode que utilitzem per a la documentació: definirem quines històries d'usuari realitzarem cada setmana i procurarem complir-los a temps.



### 3.4. Gràfiques

#### 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, en pocs dies va començar a disminuir considerablement. De fet, en qüestió de tres dies es va passar de 34 punts per tancar a només 2.

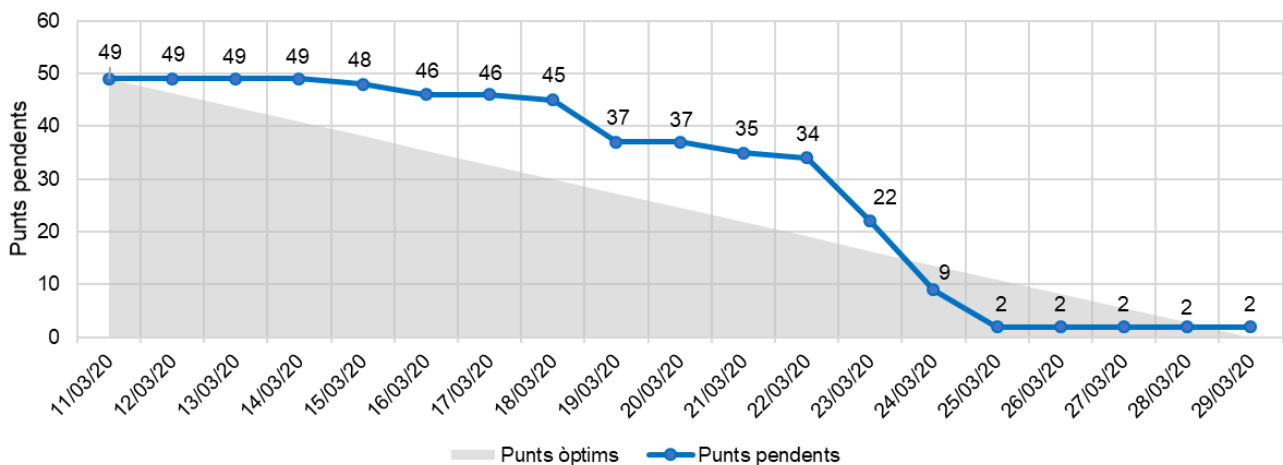


Figura 1. Sprint burndown de la primera iteració

#### 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 una quarta part del projecte. 🗨️

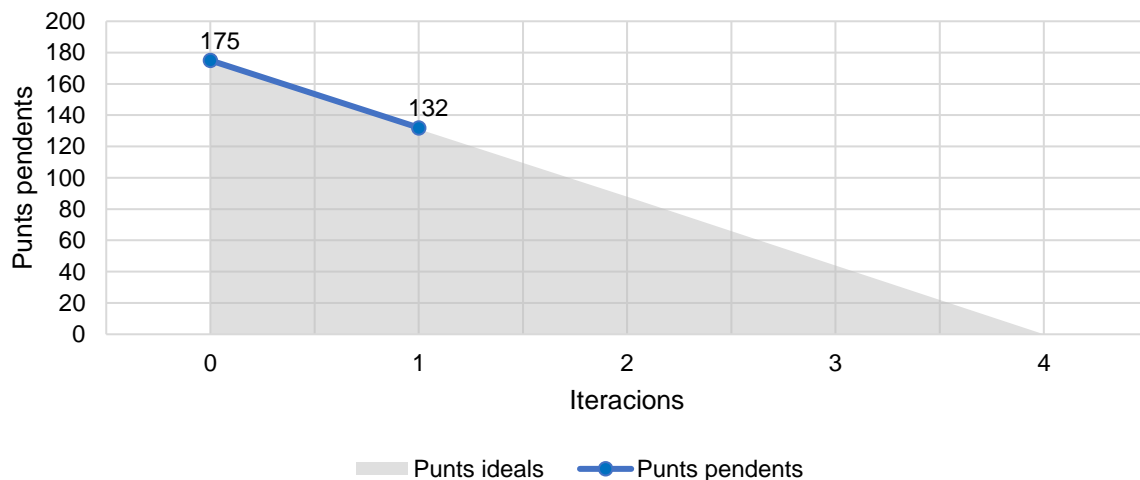
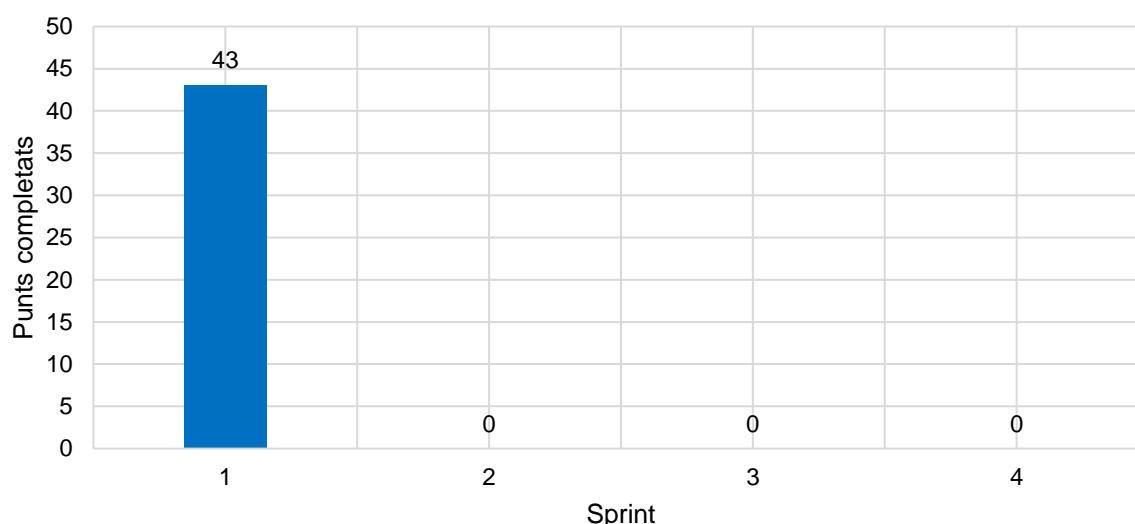


Figura 2. Release burndown

### 3.4.3. Velocitat de l'equip

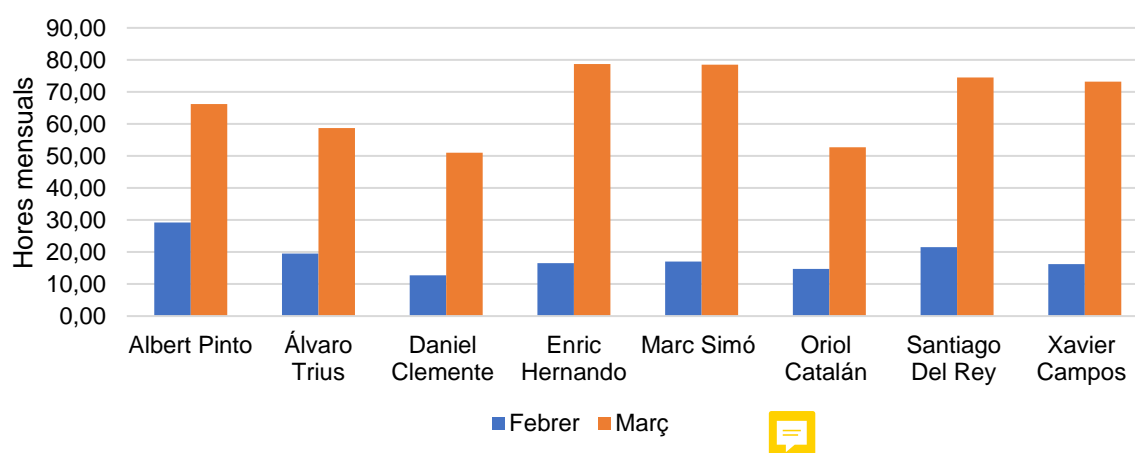
En la figura 3, es poden observar les velocitats de l'equip durant el transcurs del projecte, fins el moment de la publicació d'aquest document.



**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.

### Convencions modificades

Antiga:

- Els noms de les branques relacionades amb noves funcionalitats s'anomenaran *feature\_user\_story\_name*.

Nova:

- 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*.

### Noves convencions

- Per a crear un intent des d'una activitat a una altre, 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.

(Nou)

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 (Nou)

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 (Nou)

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. Aquesta 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. 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 (Nou)**

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 (Nou)

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 (Nou)

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.

## 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.

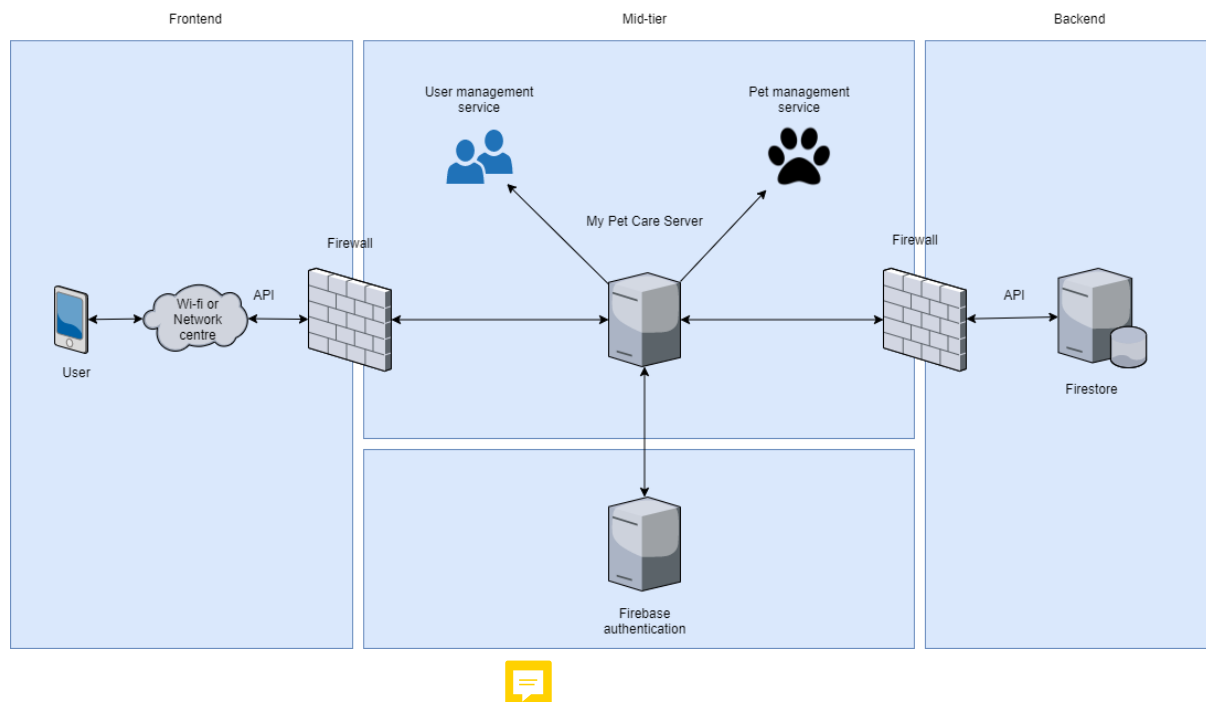
Per la comunicació entre la interfície i la base de dades utilitzarem una arquitectura de tipus ESB (*Enterprise service bus*), que unifica les peticions de l'usuari cap a la base de dades a través de les APIs. ESB consisteix en una integració distribuïda gràcies a la utilització de contenidors de serveis. 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.

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 5. 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.

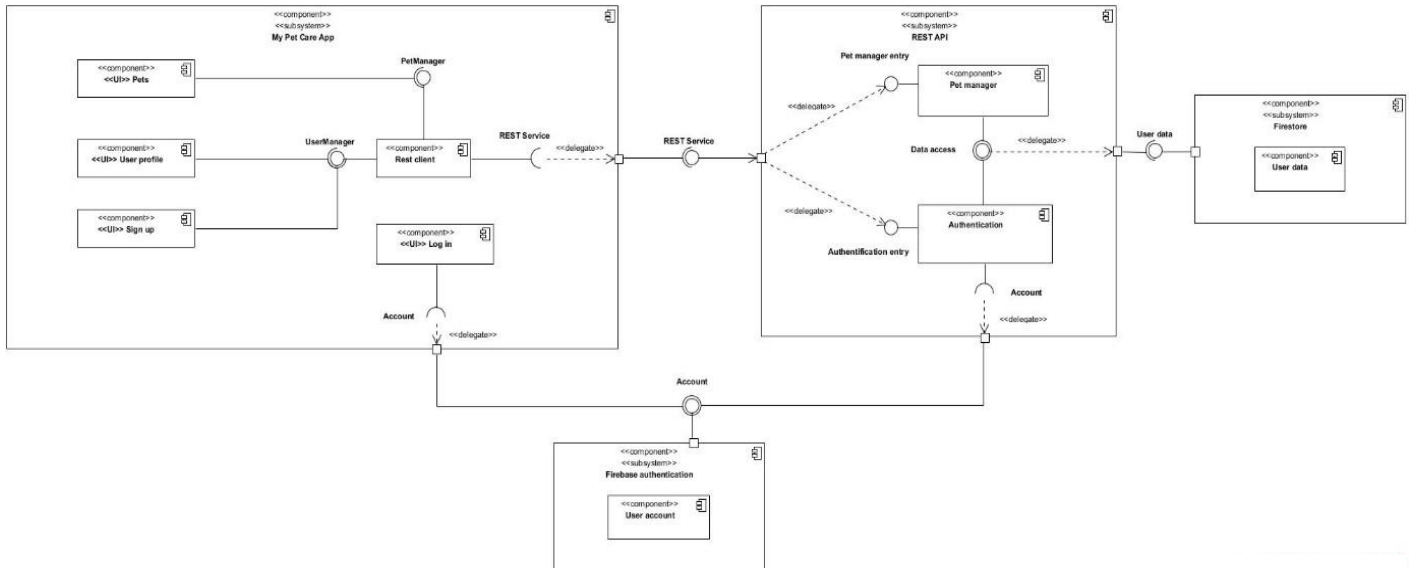


Figura 6. Diagrama de components

### 5.3. Patrons aplicats

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

#### 5.3.1. Factoria abstracta

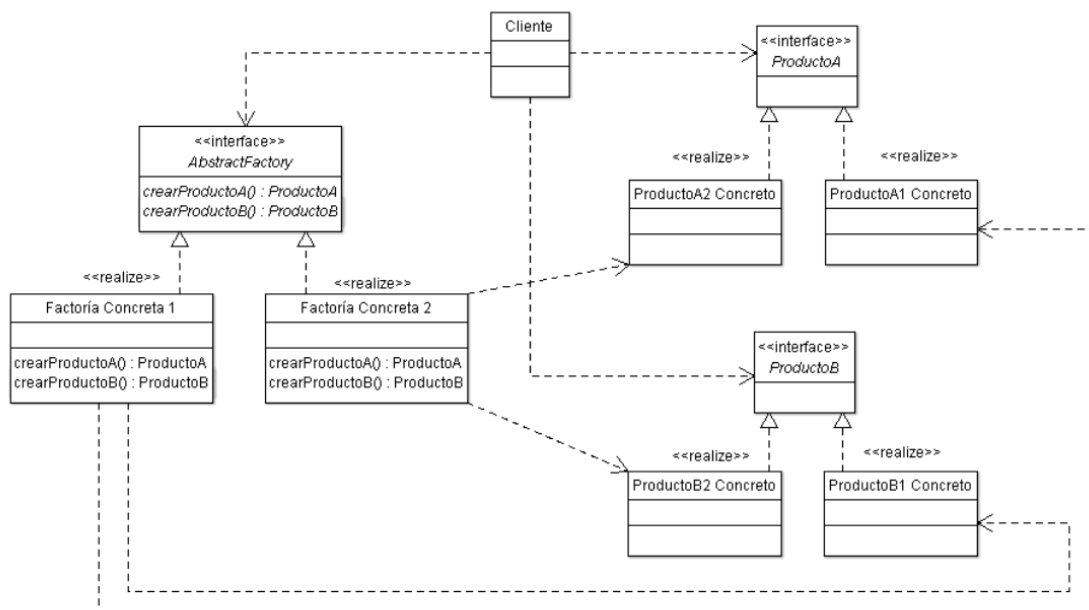
El patró factoria abstracta és un patró de disseny per el desenvolupament software que soluciona el problema de crear diferents famílies d'objectes.

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 7. 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.

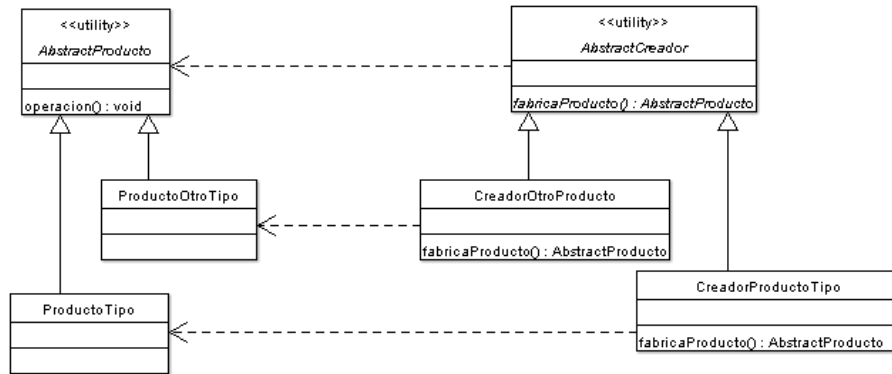


Figura 8. Exemple del disseny UML d'una factoria simple

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.

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.

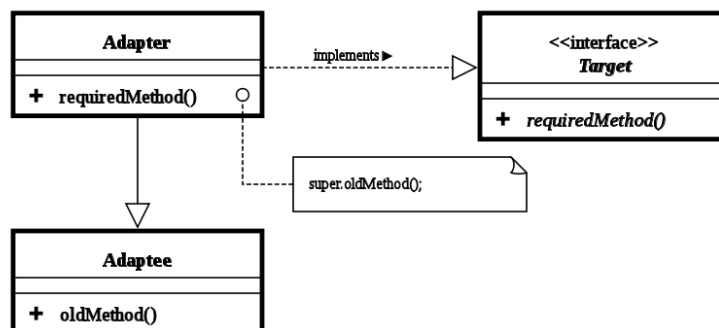
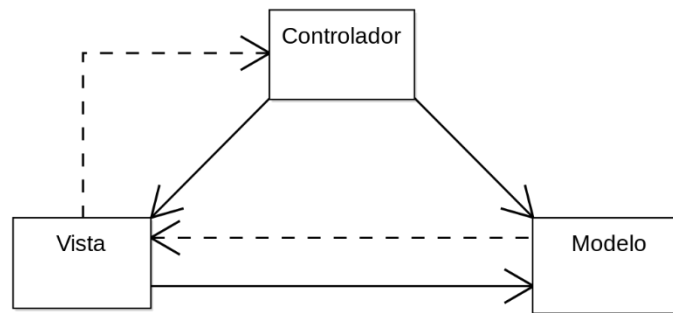


Figura 9. Exemple del disseny UML d'un adaptador

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.

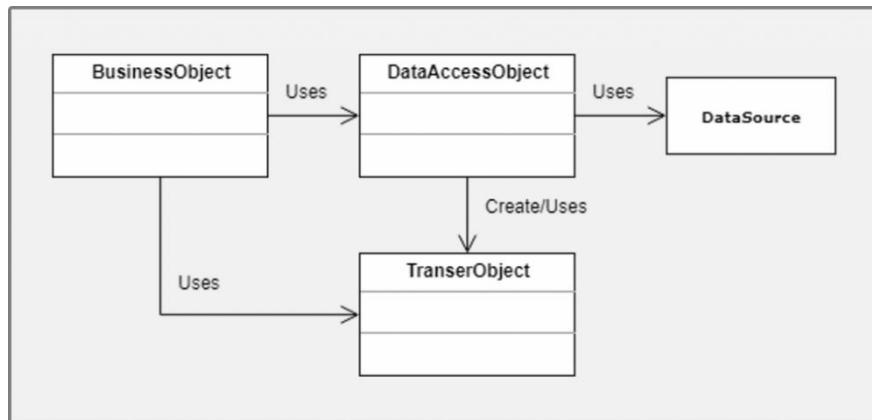


**Figura 10. Exemple del disseny UML del MVC**

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 11. 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.

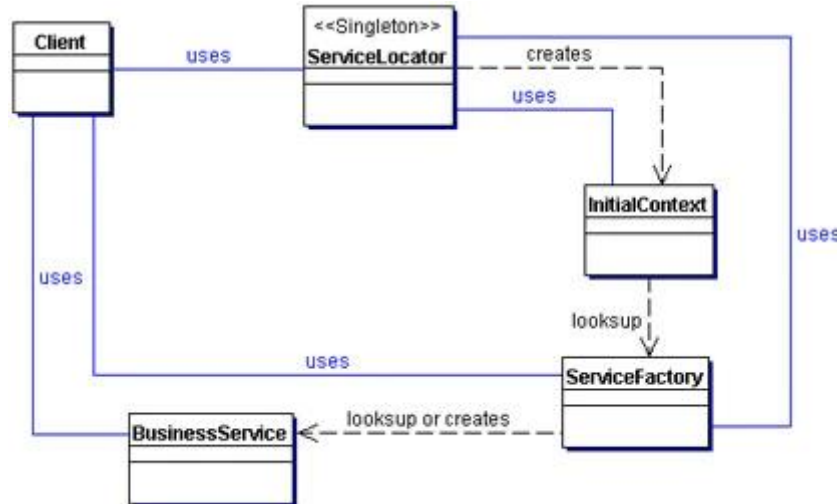


Figura 12. Exemple del disseny UML del Service Locator

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.

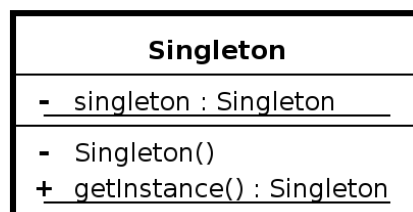
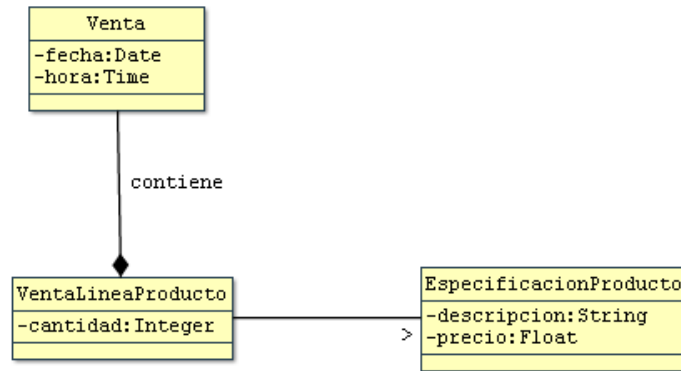


Figura 13. Representació d'un singletó en UML

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.

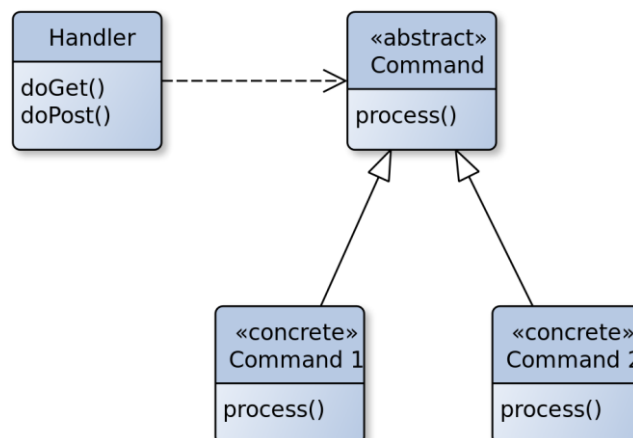


**Figura 14. Exemple d'aplicació del patró expert**

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'algorisme 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 15. Exemple del disseny UML del controlador de transacció**



### 5.4. Model conceptual de les dades

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

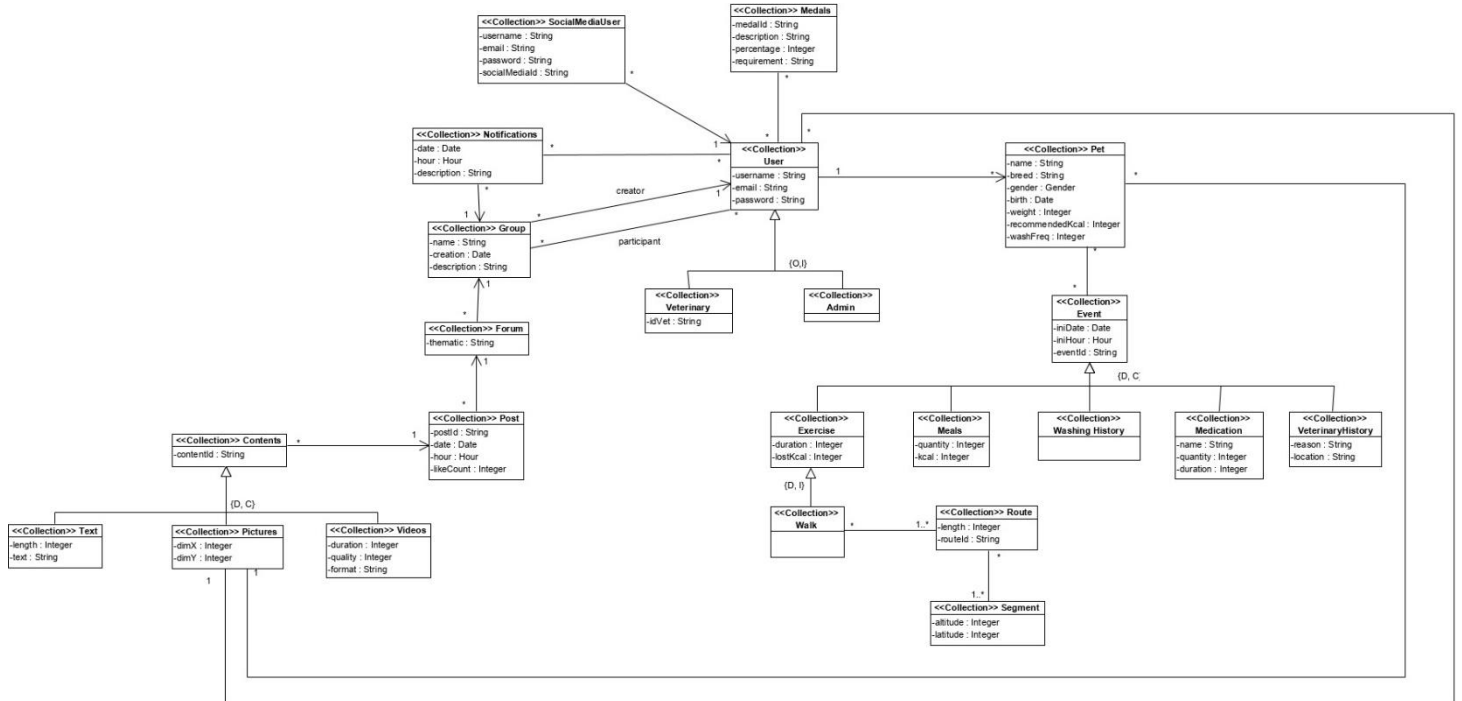


Figura 16. Model de dades final



Figura 17. Model de dades implementat actualment





## 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.

#### 5.5.6. Integració contínua

Utilitzem Github Actions ja que ens proporciona les eines suficients per anar integrant el nou codi que generem en local al codi ubicat al repositori, comprovant a cada *push* o *pull request*, que el codi no provoca errors a l'aplicació executant tots els tests que es troben definits dins del projecte. D'aquesta manera ens podem assegurar que el nostre codi s'executarà de manera correcta al moment del seu llançament.

### 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.