Axel Bernal Ruiz

Disseny d'una eina per a l'organització interna d'una entitat de persones voluntàries pel benestar animal

TREBALL DE FI DE GRAU

Dirigit per
Dr. Pere Millán Marco

Grau d'Enginyeria Informàtica

UNIVERSITAT ROVIRA I VIRGILI

Tarragona

2023

Resum.

El propòsit d'aquest projecte és desenvolupar una eina administrativa per als voluntaris de l'entitat La Caseta Dels Gats en Tarragona per tal de centralitzar les dades i la gestió d'aquesta. Ara mateix l'entitat utilitza diversos grups de WhatsApp i eines online per gestionar i comunicar-se amb els usuaris, així com eines online gratuïtes per compartir fitxers com fotos o documents.

La solució proposada ofereix el disseny d'una base de dades que s'ajusti a les necessitats de l'entitat, així com una aplicació web administrativa. A més, l'entitat demana una aplicació mòbil compatible amb Android i iOS, per a que els usuaris puguin comunicar incidències, absències, o els resultats de les seves tasques assignades amb el servidor.

Resumen.

El propósito de este proyecto es desarrollar una herramienta administrativa para los voluntarios de la entidad La Caseta Dels Gats en Tarragona para centralizar los datos y la gestión de ésta. Ahora mismo la entidad utiliza varios grupos de WhatsApp y herramientas online para gestionar y comunicarse con los usuarios, así como herramientas online gratuitas para compartir ficheros como fotos o documentos.

La solución propuesta ofrece el diseño de una base de datos que se ajuste a las necesidades de la entidad, así como una aplicación web administrativa. Además, la entidad pide una aplicación móvil compatible con Android e iOS, para que los usuarios puedan comunicar incidencias, ausencias, o los resultados de sus tareas asignadas con el servidor.

Abstract.

The purpose of this project is to develop an administrative tool for the volunteers of the organization La Caseta Dels Gats in Tarragona to centralize their data and management. Currently, the organization uses several WhatsApp groups and online tools to manage and communicate with users, as well as free online tools for sharing files such as photos or documents.

The proposed solution offers the design of a database that fits the needs of the organization, as well as an administrative web portal. In addition, the organization demands a mobile application compatible with both Android and iOS, so that users can communicate incidents, absences, or the results of their assigned tasks with the server.

Índex

Índex de taules

Índex de figures

1 Introducció

A dia d'avui, la digitalització s'ha convertit en una eina indispensable per gestionar qualsevol part de les nostres vides. Ha transformat la forma en la que ens comuniquem, interactuem, gestionem informació i, el més important, com facilita la operació d'un negoci o entitat. En conseqüència, moltes organitzacions reconeixen la necessitat d'adoptar tecnologies per tal de facilitar la seva operació i millorar l'experiència dels seus usuaris.

Administrar un refugi per a gats comporta nombrosos reptes. La Caseta Dels Gats (a partir d'ara abreviat com a LCDG) depèn actualment de diversos grups de WhatsApp per compartir informació important, com ara tasques o absències i comunicar incidències entre diferents "rols" d'usuaris. El problema d'utilitzar aquests grups és que la comunicació resulta ser molt fragmentada i això porta a ineficàcia, problemes de comunicació, i pèrdua de temps. LCDG llavors necessita d'una solució centralitzada per poder, almenys, tenir tota la informació en un sol lloc i automatitzar certs aspectes de l'entitat.

L'objectiu del projecte és desenvolupar una aplicació que inclogui un *backend* (REST API), amb totes les estructures necessàries per poder gestionar el CRUD (*Create Read Update Delete*) de les taules principals. Aquest *backend*, ha de poder implementar un sistema d'usuaris i rols per tal que només els administradors de LCDG puguin consultar i/o editar dades sensibles, mentre que els usuaris voluntaris només podran consultar les seves tasques assignades, informar d'absències, disponibilitat horària, i escriure informes sobre les tasques realitzades.

Per a la gestió, es desenvolupa una aplicació web per realitzar la gestió administrativa de l'entitat. Aquesta app permetrà als administradors de LCDG gestionar totes les dades necessàries per facilitar la gestió de l'entitat, com per exemple la creació i assignació de tasques, la creació i edició de totes les dades, pujar documents i fotos relacionats amb els gats, alta i baixa d'usuaris, visió de la planificació, ...

Finalment, cal desenvolupar una aplicació mòbil per a que els voluntaris puguin visualitzar els seus torns horaris, tasques assignades, comunicar absències, i escriure informes sobre les tasques realitzades.

Tot plegat, aquesta consolidació amb una única plataforma facilitarà el seguiment de totes les tasques realitzades a l'entitat i la col·laboració entre els usuaris, deixant així els grups de WhatsApp per a comunicats realment importants.

Figura 1. Logotip de LCDG

2 Descripció general del projecte

Per a entendre millor la funcionalitat del projecte cal descriure alguns aspectes del funcionament del refugi. El refugi consisteix en dos tipus d'usuaris, administradors/responsables i voluntaris. Els administradors s'encarreguen de portar la gestió i administració de LCDG, mentre que els voluntaris es dediquen a dur a terme tasques de manteniment.

Els voluntaris de LCDG han de complir dues funcions. Una són els torns: un voluntari ha de fer un (o varis) torns setmanals, que consisteixen en tasques bàsiques com la neteja i manteniment del refugi, així com atendre el benestar dels gats. La segona és la realització de tasques puntuals: aquestes poden ser, per exemple, donar la medicació o portar un gat al veterinari, entre altres.

Per a gestionar un gat, l'entitat manté una sèrie de dades. A part de les dades bàsiques com el nom, sexe, pes, ... cal mantenir també constància sobre l'estat de vacunació, esterilització, quarantena, desparasitacions, tests FeLV[1] i FIV[2] realitzats, informes de veterinaris, àlbum de fotos, ... La quarantena d'un gat implica posar-lo en una habitació en isolació uns dies, a causa d'una malaltia infecciosa.

A més dels usuaris i gats, l'entitat manté també un registre dels medicaments, veterinaris, i colònies ferals[3].

Un cop analitzats els requisits de LCDG, podem separar el projecte en tres grans parts definides a continuació:

2.1 *Backend*

S'implementa un *backend* mitjançant MySQL i PHP. Aquest *backend* serà l'encarregat de rebre i respondre peticions REST API per als *frontends*, tenint en compte les sessions i rols dels usuaris. Els objectius principals són:

- Gestionar el CRUD (*Create Read Update Delete*) de les taules principals; gats, usuaris, tasques, veterinaris, medicaments, i colònies.
- Gestionar de forma transparent al *frontend* les taules intermèdies N:M i d'informació addicional 1:N. Aquestes taules són per exemple, el llistat de documents, fotos, vacunes, tests, ... d'un gat, les preferències horàries i absències d'un usuari, o totes les dates i usuaris assignats d'una tasca.
- Generar una planificació setmanal basada en les tasques assignades a usuaris i els torns, respectant les seves preferències horàries.
- Generar un contracte d'adopció mitjançant les dades en BD del gat.
- Gestionar de forma transparent al *frontend* els arxius pujats i generar un *thumbnail* de les fotos per tal de reduir l'ample de banda utilitzat i agilitzar la càrrega d'imatges.

[1] Feline Leukemia Virus
[2] Feline Immunodeficiency Virus
[3] Una colònia es refereix a un conjunt de gats "salvatges" que conviuen en una ubicació determinada

2.2 *Frontend* administratiu (web)

Es desenvolupa un *frontend* web utilitzant ReactJS a petició d'un membre de l'entitat amb coneixements informàtics. De forma general, les funcions d'aquesta app són:

- Encarregat de poder mostrar totes les dades del sistema, així com gestionar-les utilitzant les operacions exposades per la REST API.
- Permetre visualitzar les tasques i torns setmanals amb una pàgina del tipus Google Calendar.
- Poder donar d'alta i baixa usuaris.
- Veure de forma general les tasques pendents o no finalitzades, absències d'usuaris, i properes tasques de la setmana.
- Opció de posar un gat en adopció.

2.3 *Frontend* de voluntaris (app mòbil)

L'aplicació mòbil es desenvolupa mitjançant React Native a petició del membre de l'entitat amb coneixements informàtics. Aquesta aplicació només permet fer les següents funcions:

- Poder gestionar la disponibilitat de l'usuari, és a dir, introduir fins a dues preferències horàries (dia i hora), indicar si pot fer dos torns a la setmana, i si la preferència pot ser variable.
- Rebre notificacions quan s'assigna un torn o tasca i unes hores abans que comenci.
- Rebre una notificació quan acabi una tasca assignada, per tal de recordar l'usuari d'escriure un petit informe i indicar si l'ha acabada.
- Consultar les dades bàsiques d'un gat, medicament, o veterinari.
- Consultar les properes tasques del mateix usuari.

3 Requisits

A continuació es detallen els requisits funcionals del projecte.

S'anomena "usuari" a qualsevol usuari, independentment del seu rol. Els rols poden ser "administrador" i "voluntari".

1. Els usuaris han de poder iniciar sessió si compleixen el rol necessari per a l'aplicació.
2. Els usuaris han de poder tancar sessió.
3. Els usuaris han de poder canviar la seva contrasenya.
4. Els administradors han de poder regenerar una contrasenya per a un usuari.
5. Els usuaris han de poder especificar fins a dues preferències horàries.
6. Els usuaris han de poder especificar dies d'absència.
7. Els usuaris han de poder rebre notificacions.
8. Els usuaris han de poder escriure informes sobre les tasques completades.
9. Els usuaris han de poder consultar les dades d'un gat.
10. Els usuaris han de poder consultar les dades d'un medicament.
11. Els usuaris han de poder consultar les dades d'un veterinari.
12. Els administradors han de poder crear, editar i arxivar gats.
13. Els administradors han de poder crear i esborrar vacunacions.
14. Els administradors han de poder crear i esborrar tests FeLV i FIV.
15. Els administradors han de poder crear i esborrar desparasitacions.
16. Els administradors han de poder posar i treure gats de quarantena.
17. Els administradors han de poder iniciar el procés d'adopció d'un gat.
18. Els administradors han de poder crear i esborrar fotos d'un gat.
19. Els administradors han de poder seleccionar una foto per ser utilitzada com l'avatar d'un gat.
20. Els administradors han de poder crear i esborrar documents PDF d'un gat.
21. Els administradors han de poder crear, veure, editar i esborrar colònies ferals.
22. Els administradors han de poder crear, editar i esborrar medicaments.
23. Els administradors han de poder crear, editar i esborrar veterinaris.
24. Els administradors han de poder veure i modificar la planificació setmanal.
25. Els administradors han de poder crear, veure, editar i esborrar tasques.
26. Els administradors han de poder planificar tasques i assignar-les a un usuari.
27. El sistema ha de poder generar una planificació setmanal.

4 Anàlisi dels requisits funcionals

En aquest apartat es mostren els diagrames de casos d'ús (cdu); s'especifica "gestionar X" com l'acció de crear, editar o esborrar un element X (per simplificar el diagrama).

4.1 Casos d'ús App administrativa

En aquest cas se suposa que un usuari ha de tenir com a prerequisit haver iniciar sessió per tal d'accedir a tots els altres cdu, cosa que no s'indica per a que el diagrama sigui més fàcil de entendre.

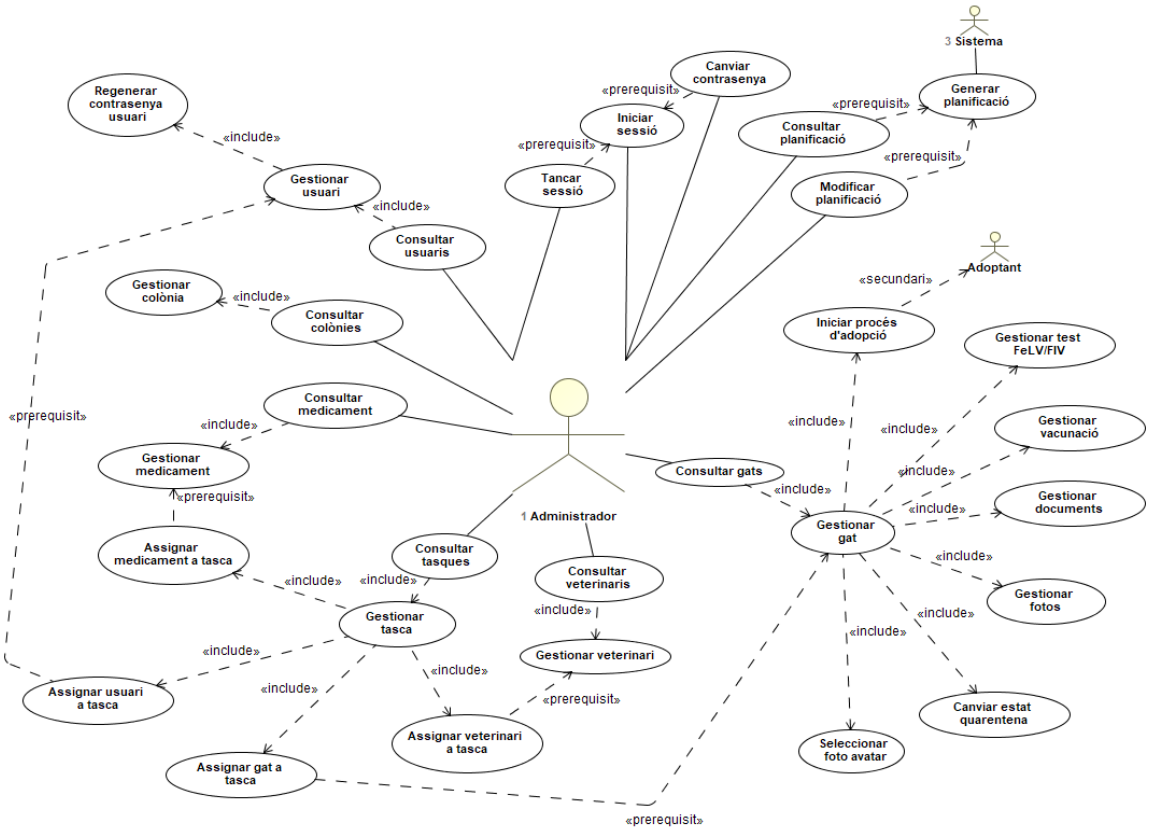


Figura 2. Casos d'ús de l'app administrativa.

4.2 Casos d'ús App mòbil

En aquest cas se suposa que un usuari ha de tenir com a prerequisit haver iniciar sessió, per poder accedir a tots els altres cdu, cosa que no s'indica per a que el diagrama sigui més fàcil d'entendre.

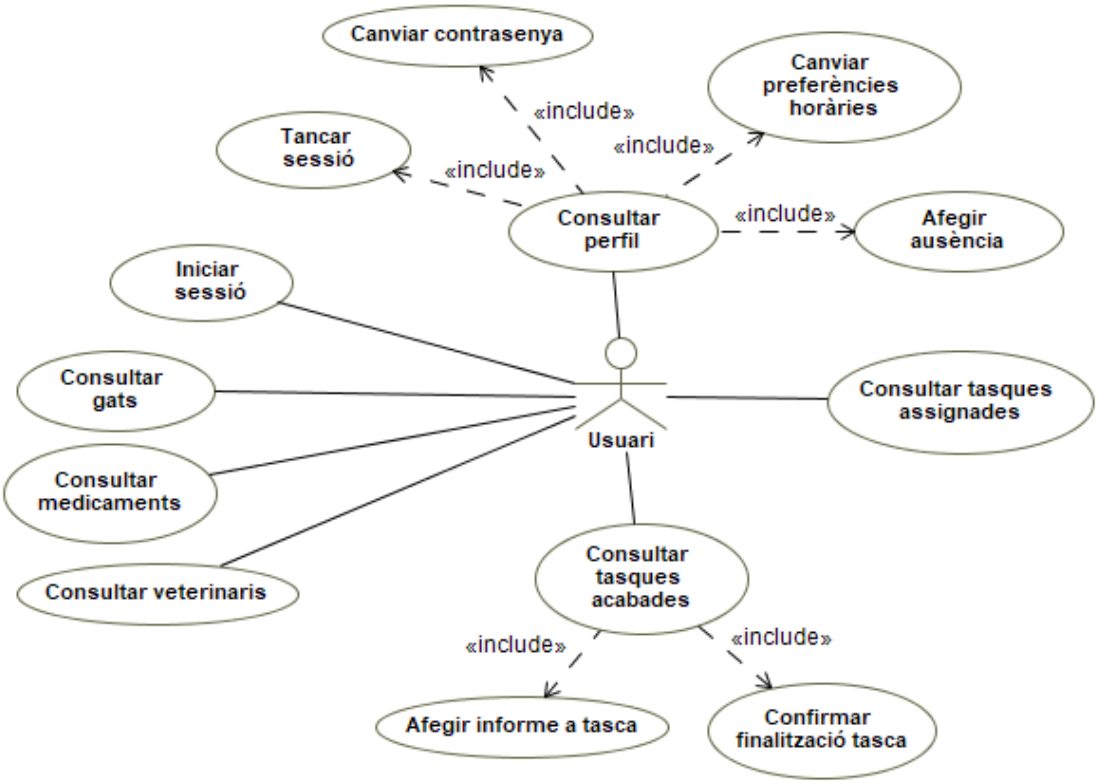


Figura 3. Casos d'ús de l'app mòbil

5 Disseny

En aquest apartat es detalla l'arquitectura i llenguatge utilitzats, uns exemples d'interfícies gràfiques i el diagrama de classes utilitzat per a la BD.

5.1 Arquitectura de l'aplicació

5.1.1 PHP

PHP és un dels llenguatges de programació per a web més populars, ja que és ràpid, flexible i fàcil d'aprendre. Aquest llenguatge ofereix *frameworks* i llibreries per simplificar tasques com connexions a BD, modificar els *headers* de respostes HTTP, manipular *cookies*, rebre dades de formularis (ja sigui POST o GET), permetre l'enviament de fitxers, ... PHP, a més, ofereix una extensió bastant útil per a aquest projecte anomenada "GD" que és una llibreria per al processament d'imatges. Aquesta llibreria és útil a l'hora de pujar fotos per a poder guardar una còpia com a *thumbnail* i estalviar ample de banda, a més d'agilitzar la vista de certes pàgines.

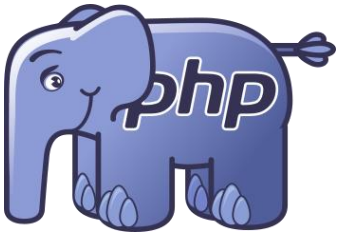


Figura 4. Logotip de PHP

5.1.2 MySQL

Dintre del món de les bases de dades relacionals tenim diversos DBMS[4], cadascun amb les seves peculiaritats. S'ha optat per utilitzar MySQL ja que és el que es va aprendre a l'assignatura de bases de dades i PHP ofereix una llibreria per facilitar la connexió a aquesta. Realment, l'aplicació a desenvolupar és relativament petita. Per tant, optar per qualsevol altre DBMS, o fins i tot NoSQL, no canviaria gaire el rendiment o l'ocupació de memòria.

Figura 5. Logotip de MySQL

[4] DBMS: *Data Base Management System.*

5.1.3 ReactJS / React Native

React és un *framework* FOSS[5] de Meta (Facebook) que utilitza JavaScript per desenvolupar interfícies gràfiques basades en components reutilitzables. Aquests components funcionen per estats, és a dir, la vista d'un component no s'actualitza si no s'actualitzen les dades o no hi ha interactivitat. React generalment se sol "separar" en dos tipus: ReactJS per al desenvolupament web i React Native per al desenvolupament d'aplicacions mòbils natives, ja sigui Android o iOS. La forma en la que React Native funciona és principalment la mateixa que ReactJS, excepte que en comptes de manipular el DOM[6] existeix un procés en segon pla que intercepta les crides de JavaScript i les tradueix a crides de components natius.

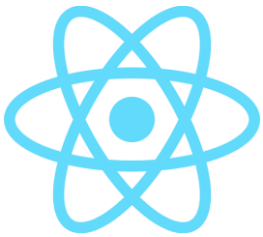


Figura 6. Logotip de React

5.1.4 Bootstrap

Bootstrap és un *framework* FOSS desenvolupat per Twitter, que conté components HTML, CSS, i JS per facilitar la implementació d'una interfície gràfica que sigui responsiva i visualment consistent. Per a la nostra aplicació, Bootstrap pot donar alguns problemes de compatibilitat amb React quan s'utilitzen components que modifiquen el DOM (p. ex. modals o toasts). Per aquest motiu s'utilitza també la llibreria react-bootstrap que implementa aquestes incompatibilitats com a components React.

Figura 7. Logotip de Bootstrap

[5] FOSS: *Free and Open-Source Software*.
[6] *Document Object Model*. En HTML representa l'arbre d'etiquetes.

5.2 Disseny de la BD

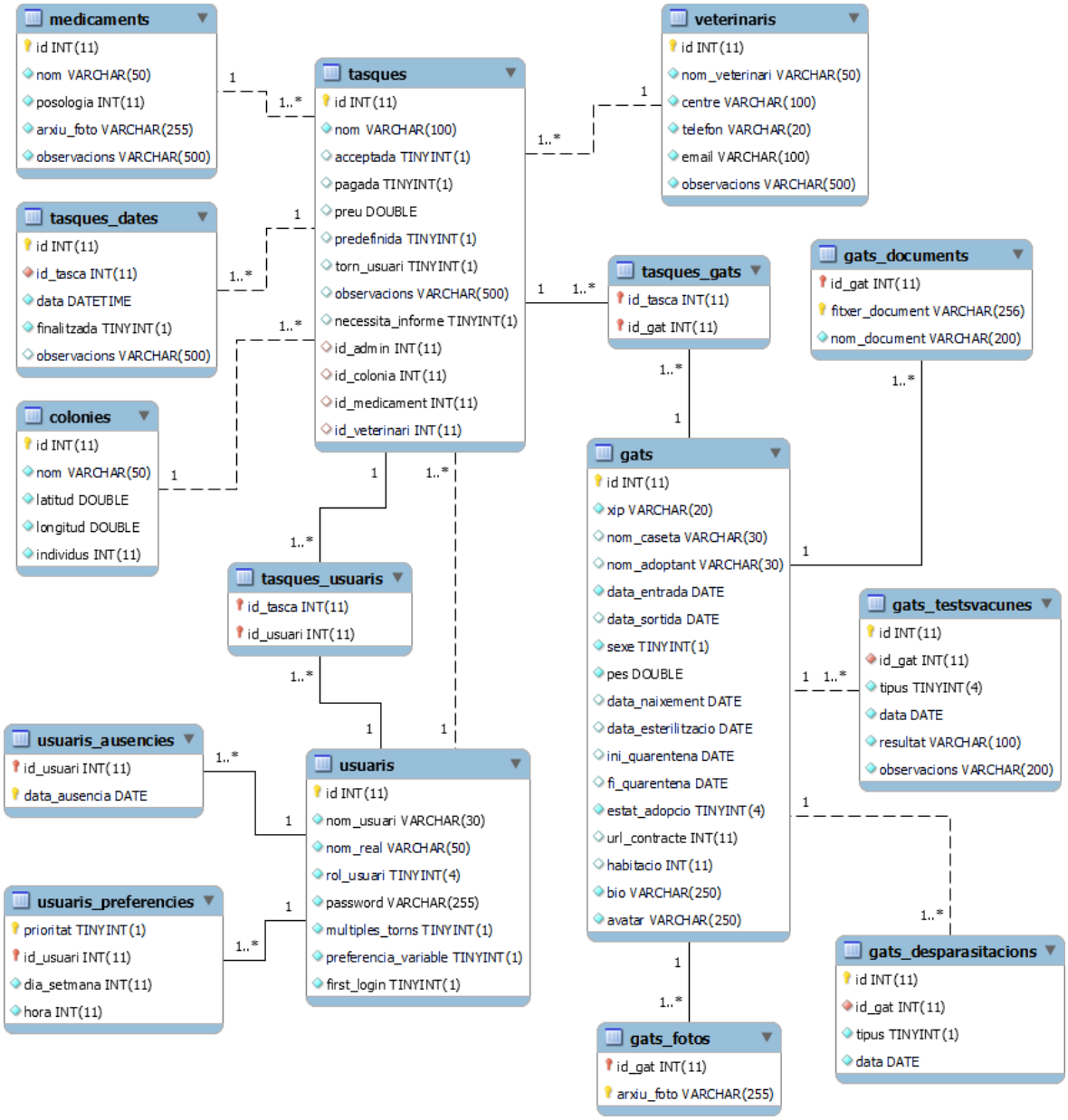


Figura 8. Diagrama entitat-relació de la base de dades

Seguidament es detallen les taules, les relacions, i el propòsit dels camps que inclouen.

5.2.1 Taula medicaments

Guarda la informació relacionada sobre un medicament en concret. La seva relació amb *tasques* serveix per indicar que aquell medicament s'ha d'administrar durant la tasca.

- **nom:** Nom del medicament.
- **posologia:** Dosi/Kg del medicament.
- **arxiu_foto:** Nom del fitxer que conté la foto del medicament.
- **observacions:** Possibles observacions sobre el medicament.

5.2.2 Taula veterinaris

Guarda la informació sobre un veterinari. La seva relació amb *tasques* indica que la tasca és relacionada amb un veterinari en concret (per exemple, una visita).

- **nom_veterinari:** Nom del veterinari.
- **centre:** Nom del centre veterinari.
- **telefon:** Telèfon de contacte.
- **email:** Adreça de correu electrònic.
- **observacions:** Possibles observacions sobre el veterinari.

5.2.3 Taula colònies

Guarda la informació sobre una colònia. La seva relació amb *tasques* indica que la tasca s'ha de realitzar sobre una colònia en concret.

- **nom:** Nom de la colònia.
- **latitud:** Latitud geogràfica.
- **longitud:** Longitud geogràfica.
- **individus:** Número d'individus.

5.2.4 Taula tasques_dates

Conté totes les dates en les que una tasca s'ha planificat, el seu estat, i les observacions realitzades en aquella data.

- **id_tasca:** Tasca relacionada (FK *tasques*).
- **data:** Data i hora en la que un dels torns de la tasca s'ha planificat.
- **finalitzada:** Si la tasca s'ha pogut completar o no.
- **observacions:** Petit informe relacionat sobre el torn.

5.2.5 Taula tasques

Conté la definició i informació bàsica d'una tasca.

- **nom:** Nom de la tasca.
- **acceptada:** Si la tasca és acceptada o no. Només és poden planificar tasques acceptades.
- **pagada:** Si la tasca és completament pagada o no.
- **preu:** El preu que cal pagar per la tasca. Si és diferent a 0 vol dir que la tasca requereix d'un pagament.
- **predefinida:** Si la tasca és predefinida o no. Si una tasca és predefinida, llavors no és podrà planificar. Les tasques predefinides permeten tenir un esquelet per crear noves tasques a partir d'aquesta.
- **torn_usuari:** Indica que la tasca pertany al torn de neteja d'un usuari. Aquestes tasques es generen automàticament en crear un usuari.
- **observacions:** Possibles observacions sobre la tasca.
- **necessita_informe:** Si la tasca necessita un informe quan l'usuari la completi en la data assignada.
- **id_admin:** Administrador responsable assignat a la tasca (FK *usuaris*).
- **id_colonia:** Colònia relacionada (FK *colònies*).
- **id_medicament:** Medicament relacionat (FK *medicaments*).
- **id_veterinari:** Veterinari relacionat (FK *veterinaris*).

5.2.6 Taula usuaris

Guarda les dades del compte d'un usuari.

- **nom_usuari:** Nom que utilitzarà l'usuari per iniciar sessió.
- **nom_real:** Nom real de l'usuari. És útil per a que un administrador pugui identificar més fàcilment un usuari.
- **rol_usuari:** Rol assignat l'usuari. Aquests rols poden ser:
  - **0:** Usuari desactivat (no pot iniciar sessió).
  - **1:** Administrador.
  - **2:** Voluntari.
- **password:** Contrasenya encriptada amb *salt*[7].
- **multiples_torns:** Si l'usuari indica que pot fer múltiples torns o no.
- **preferencia_variable:** Si l'usuari vol tenir una preferència horària variable o no.
- **first_login:** Indicat internament per saber si un usuari encara no ha iniciat sessió per primer cop o si la seva contrasenya ha estat reiniciada per un administrador. Aquest camp s'utilitza a les aplicacions per forçar el canvi de contrasenya d'un usuari un cop utilitzi la contrasenya d'un sol ús.

5.2.7 Taula usuaris_ausencies

Guarda un registre de les absències relacionades amb un usuari.

- **data_ausencia:** Dia en el que l'usuari no pot rebre tasques.

5.2.8 Taula usuaris_preferencies

Guarda les dues possibles preferències horàries d'un usuari. Ja que la planificació de torns de neteja i manteniment és setmanal, només cal guardar un dia de la setmana i l'hora. La clau primària és la prioritat i l'id_usuari. Per tant, per a claus es força una relació 0..2 entre *usuaris* i *usuaris_preferencies*.

- **prioritat:** Prioritat de la preferència. Un 0 indica prioritat alta, mentre que un 1 indica prioritat baixa.
- **id_usuari:** Usuari al que pertany la preferència (FK *usuaris*).
- **dia_setmana:** Dia de la setmana indicat numèricament.
- **hora:** Hora de la preferència.

5.2.9 Taula gats

Conté les dades que identifiquen un gat.

- **xip:** Codi del xip identificatiu d'un gat.
- **nom_caseta:** Nom del gat proporcionat per LCDG.
- **nom_adoptant:** Nom del gat donat per l'adoptant.
- **data_entrada:** Data d'entrada a LCDG.
- **data_sortida:** Data de sortida de LCDG.
- **sexe:** Sexe del gat. Un 0 indica mascle mentre que un 1 indica femella.
- **pes:** Pes en kg del gat.
- **data_naixement:** Data de naixement.

[7] En criptografia, una *salt* és un conjunt de bits aleatoris que s'afegeixen a les dades un cop encriptades.

- **data_esterilitzacio:** Data en que s'ha esterilitzat el gat. Si és null vol dir que no és estèril.
- **ini_quarentena:** Data d'inici de quarantena.
- **fi_quarentena:** Data fi de quarantena. Si és null i *ini_quarentena* no ho, és llavors el gat es troba sota quarantena.
- **estat_adopcio:** Estat d'adopció ,indicat com:
  - **0:** No adoptat
  - **1:** En procés d'adopció
  - **2:** Adoptat
- **url_contracte:** URL al contracte d'adopció
- **habitacio:** Habitació on es troba el gat.
- **bio:** Descripció breu.
- **avatar:** Nom de l'arxiu que s'utilitza com a avatar.

5.2.10 Taula gats_documents

Conté informació sobre els documents relacionats amb un gat.

- **id_gat:** Gat relacionat amb el document (FK *gats*).
- **fitxer_document:** Nom del fitxer del document.
- **nom_docment:** Nom o petita descripció sobre el document.

5.2.11 Taula gats_fotos

Conté informació sobre les fotos relacionades amb un gat.

- **id_gat:** Gat relacionat amb la foto (FK *gats*).
- **arxiu_foto:** Nom del fitxer de la foto.

5.2.12 Taula gats_testsvacunes

Guarda la informació sobre tots els tests i vacunes realitzats a un gat.

- **id_gat:** Gat relacionat amb el registre (FK *gats*)
- **tipus:** Tipus de test o vacuna. Els possibles valors són:
  - **0:** Vacuna
  - **1:** Test FeLV
  - **2:** Test FIV
- **data:** Dia en el que s'ha realitzat.
- **resultat:** Breu descripció del resultat.
- **observacions:** Breu descripció sobre el registre.

5.2.13 Taula gats_desparasitacions

Guarda la informació sobre les desparasitacions realitzades a un gat.

- **id_gat:** Gat relacionat amb el registre (FK *gats*)
- **tipus:** Tipus de desparasitació. Els possibles valors són:
  - **0:** Interna
  - **1:** Externa
- **data:** Dia en el que s'ha realitzat.
- **resultat:** Breu descripció del resultat.

5.3 Disseny de la interfície gràfica

A l'hora d'implementar la interfície gràfica, primer s'han dissenyat alguns *mockups* de com es vol implementar i, finalment, s'ha anat modificant durant el desenvolupament per poder adaptar-ho al que realment es pot fer, així com petits canvis o decisions que s'han vist durant la implementació.

A continuació es mostren alguns exemples de com s'ha implementat la interfície gràfica i una breu descripció d'alguns elements interactuables.

5.3.1 Interfície administrativa

Per a aquesta interfície cal comentar primer una decisió de disseny. A l'hora de demanar dades a l'usuari (formularis en HTML) es fa mitjançant modals de Bootstrap. Un modal és un tipus de finestra emergent, centrada horitzontalment, que enfosqueix el fons per tal de donar-li èmfasi. Això permet crear un component que només s'encarregui de mostrar el formulari/modal i que aquest component fill avisi al component pare mitjançant un *callback* amb les dades introduïdes, que el pare enviarà al *backend* i actualitzarà la vista segons sigui necessari.

Una altra decisió de disseny és que està pensada principalment per ser utilitzada en un ordinador o pantalla amb suficient resolució horitzontal. Bootstrap ens permet adaptar fàcilment qualsevol pàgina a dispositius mòbils, però la natura dels components utilitzats complica adaptar-ho.

5.3.1.1 *Sidebar*

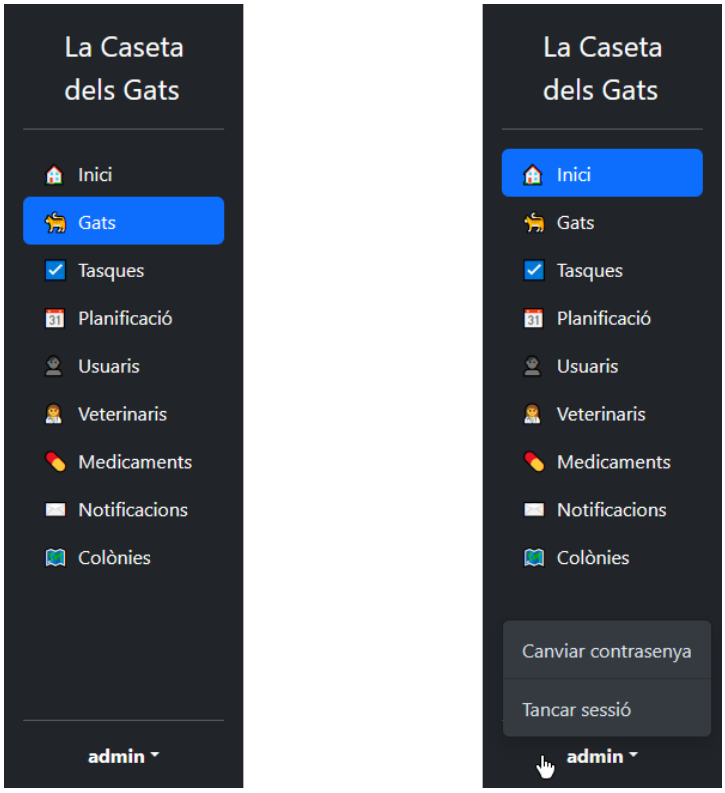


Figura 9. *Sidebar* mostrant canvis de pàgina i l'element clicable

La *sidebar* és un element que conté la navegació principal present en totes les pàgines. Aquesta consisteix en enllaços que indiquen la pàgina en la que és situat l'usuari, així com el seu nom d'usuari on, en fer clic, mostra opcions com tancar sessió i canviar contrasenya.

5.3.1.2 Consulta d'usuaris, veterinaris, i medicaments

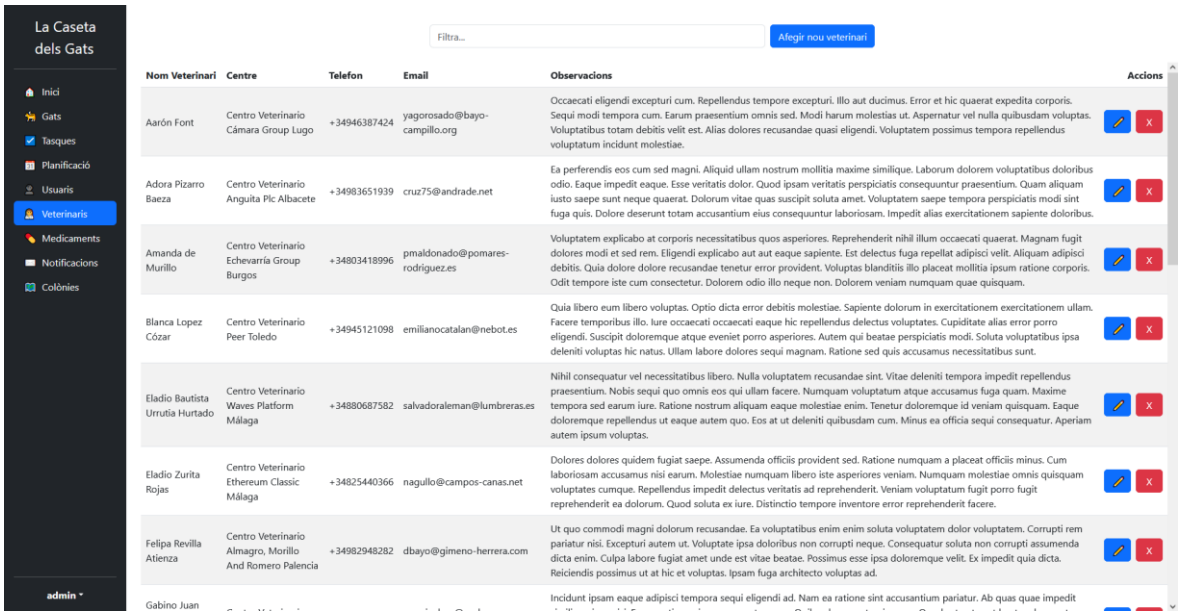


Figura 10. Pàgina amb la consulta d'una de les classes, en aquest cas veterinaris

Ja que tenim classes amb poques dades representatives i poques entrades, és més visual i espacialment eficient tenir una taula amb totes les dades rellevants. Aquestes taules incorporen botons per a gestionar l'objecte visualitzat en cada fila.

La pàgina de consulta permet filtrar els elements de la taula (per nom) mitjançant l'entrada de text situada a la part superior o afegir/editar entrades mitjançant un modal.

Els elements interactius d'aquestes pàgines són els següents:

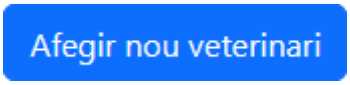


Figura 11. Botó per afegir una nova entrada

El botó per afegir una nova entrada es troba en la part superior, juntament amb el filtre. Aquests dos elements es troben visibles en tot moment per a que siguin fàcilment accessibles. En fer clic es mostra un modal a l'usuari demanant totes les dades rellevants d'aquella classe.

Figura 12. Botó per esborrar una entrada

Figura 13. Botó per editar una entrada

A totes les files existeixen dos botons, un per esborrar i altre per editar. El botó d'esborrar conté una creu i és vermell per indicar un cert perill a l'usuari. En fer clic es mostrarà un modal demanant confirmació a l'usuari. El botó d'edició conté un llapis que representa l'esmentat. En fer clic es mostrarà el mateix modal que l'acció d'afegir, però amb les dades ja emplenades amb el que representa aquella entrada.

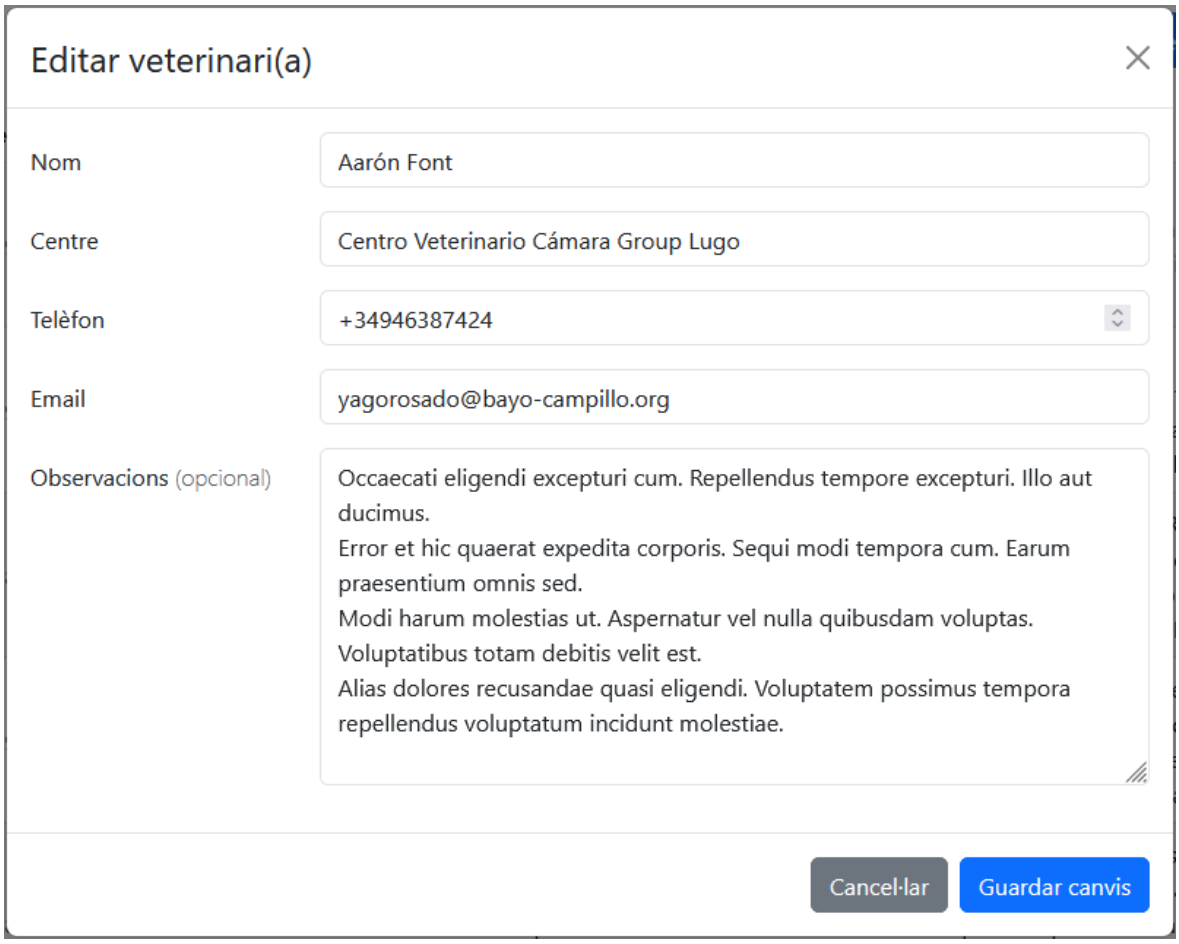


Figura 14. Exemple de modal per introduir dades, en aquest cas s'edita un veterinari

Ja sigui per afegir o editar, a l'usuari se li mostra el mateix formulari amb les dades ja emplenades o buit, respectivament. Aquests formularis segueixen la mateixa estructura independentment de la classe, és a dir, només varien les dades demanades.

L'usuari pot tancar el modal en qualsevol moment fent clic a la part enfosquida del fons, la creu, o al botó de cancel·lar, i les dades no es guardaran. Un cop l'usuari prem el botó de guardar, el modal s'amaga automàticament i notifica al component pare per a que actualitzi la taula (i BD) amb les dades introduïdes.

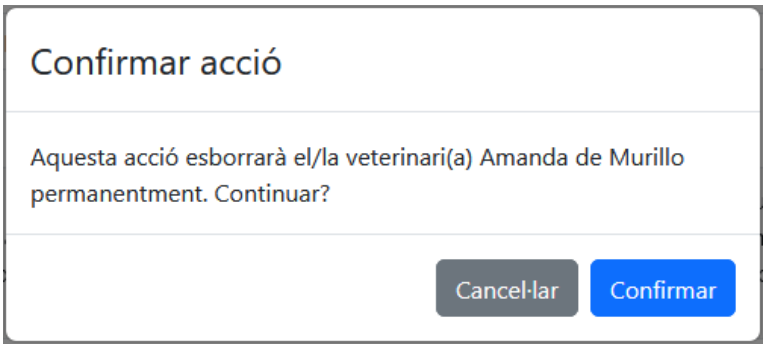


Figura 15. Exemple de modal demanant confirmació

A l'hora d'intentar esborrar una entrada (o en altres casos, esmentats més endavant) es mostra un modal que l'usuari no pot tancar fins que premi un dels botons mostrats. Si l'usuari confirma l'acció, es notifica al component pare per a que esborri l'entrada de la taula (i de BD).

Aquest pas extra és necessari per evitar que un usuari esborri accidentalment una entrada i confirmi que realment ha fet clic a l'entrada correcta, a més d'avisar-li que l'acció és permanent.

5.3.1.3 Consulta de medicaments

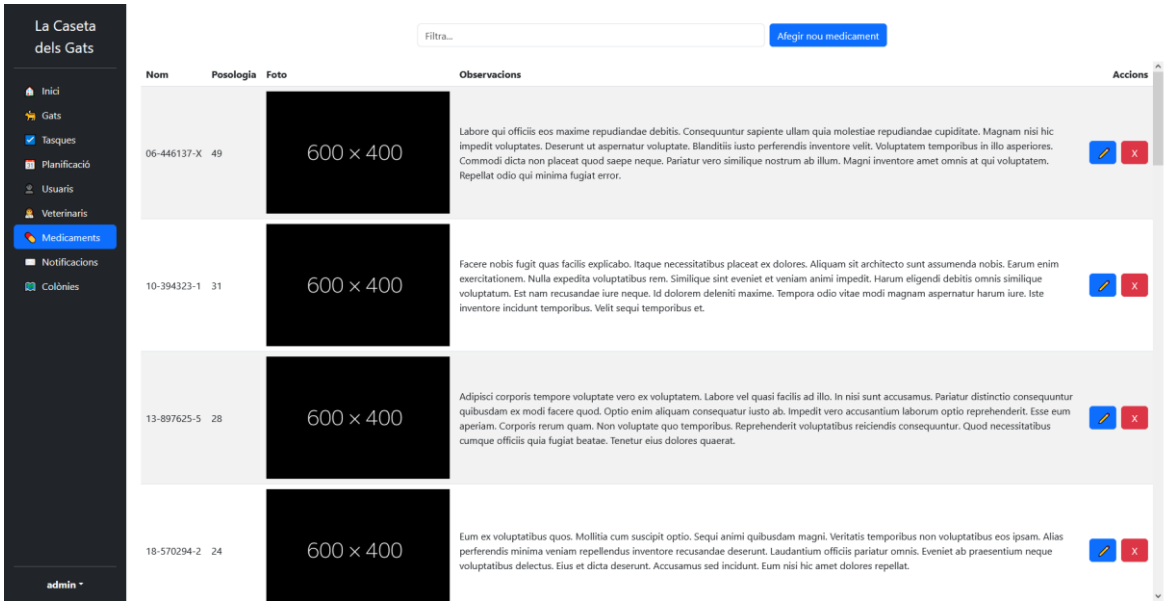


Figura 16. Llistat de medicaments

La consulta de medicaments segueix la mateixa funció que la resta de consultes, amb la diferència de mostrar una petita imatge del medicament. Aquesta imatge permet identificar fàcilment l'envàs del medicament.

5.3.1.4 Consulta d'usuaris

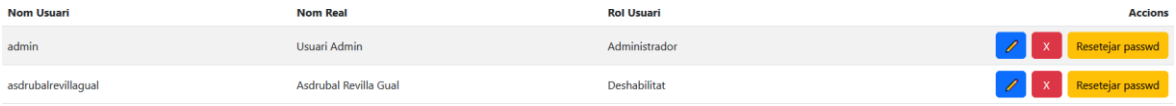


Figura 17. Entrades de la taula d'usuaris

A la consulta d'usuaris cada entrada té un botó de color groc, per indicar perill, que resetejarà el password de l'usuari d'aquella entrada. En fer clic es mostrarà un modal de confirmació a l'usuari que, en acceptar, li mostrarà el *password* generat automàticament. Cal dir que aquest *password* és temporal, quan l'usuari inicií sessió la propera vegada serà forçat a canviar-lo.

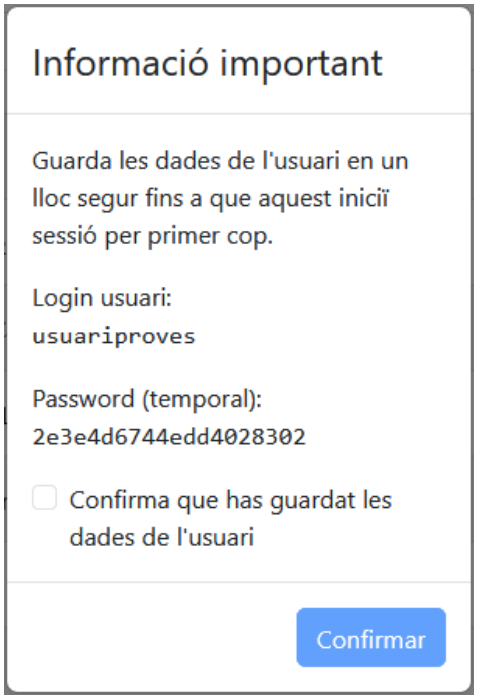


Figura 18. Informació sobre el password per defecte

En aquest modal cal destacar que l'usuari no podrà tancar-lo fins que es marqui el *checkbox* conforme ha copiat les dades en un lloc segur (i que ho ha comunicat a l'usuari en qüestió). Tot i que l'administrador pot tornar a regenerar un nou *password* per a aquest usuari en qualsevol moment.

Aquest modal es mostra també a l'hora de donar d'alta un usuari: el sistema assigna un *password* aleatori a l'hora d'inserir-lo en BD.

5.3.1.5 Consulta de gats

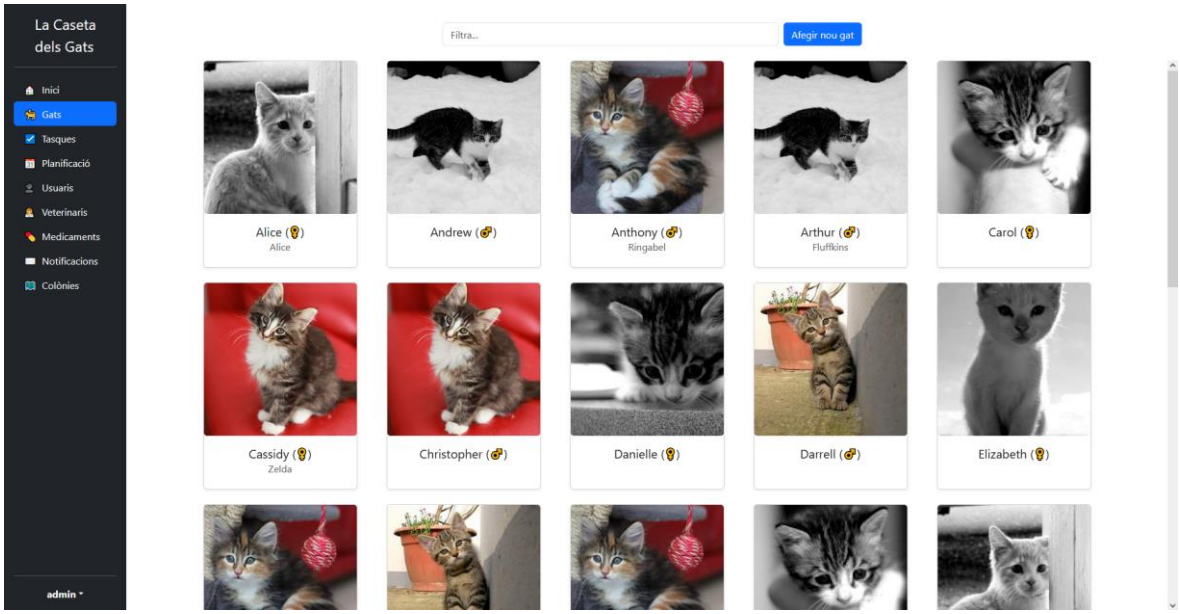


Figura 19. Pàgina amb el llistat de gats

Els gats tenen més dades per mostrar. Per tant, no es pot utilitzar una taula, ja que a simple vista no es podrien identificar. En aquesta pàgina s'utilitza una graella de 'cards' per mostrar la informació bàsica del gat, és a dir, la seva foto d'avatar, nom de LCDG, nom de l'adoptant (si escau) i el sexe. Igual que a les altres pàgines de consulta, es pot filtrar i afegir noves entrades. En fer clic a una card del gat es passa a la pàgina de detalls.

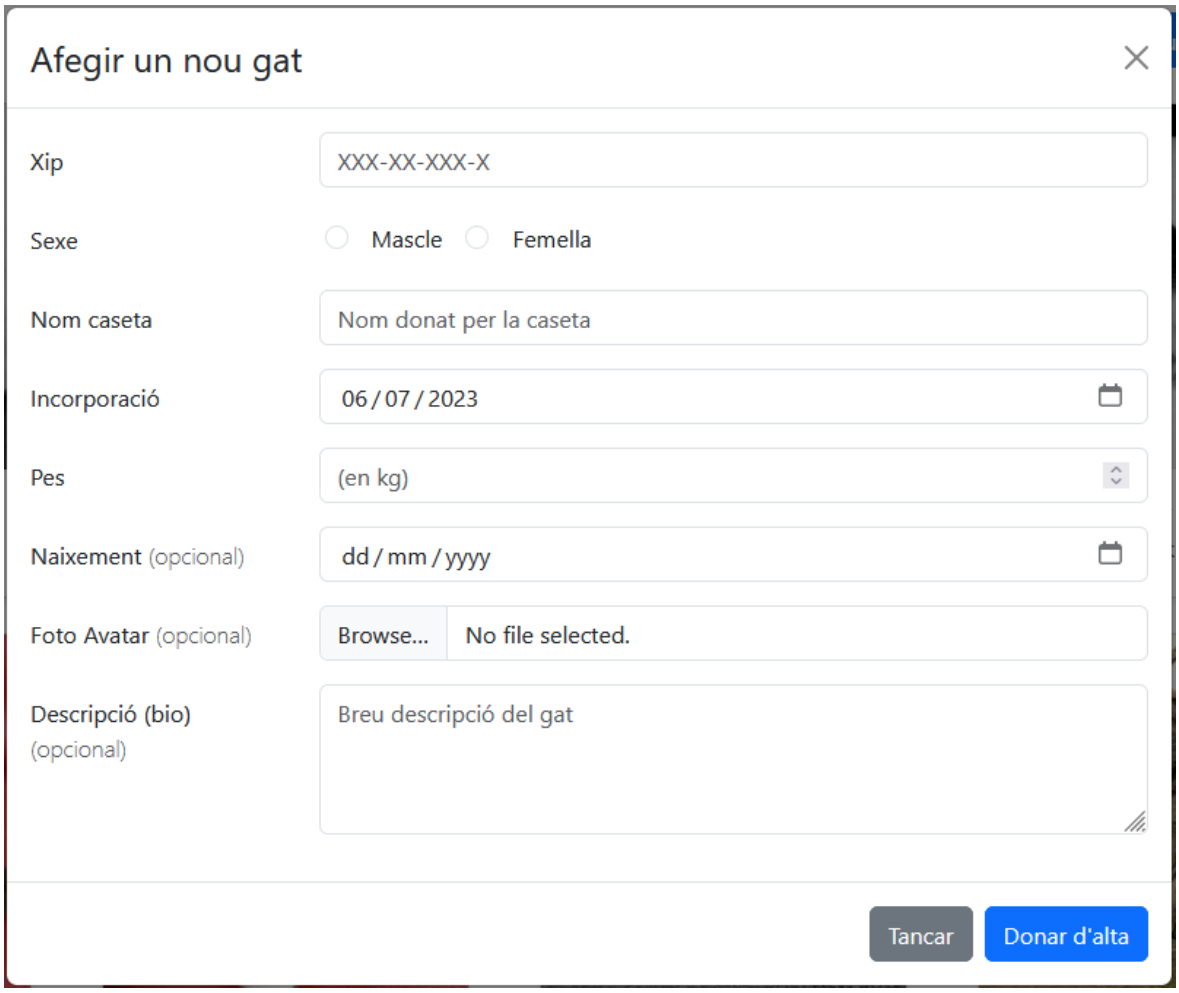


Figura 20. Formulari per introduir les dades bàsiques d'un gat

5.3.1.6 Detalls d'un gat

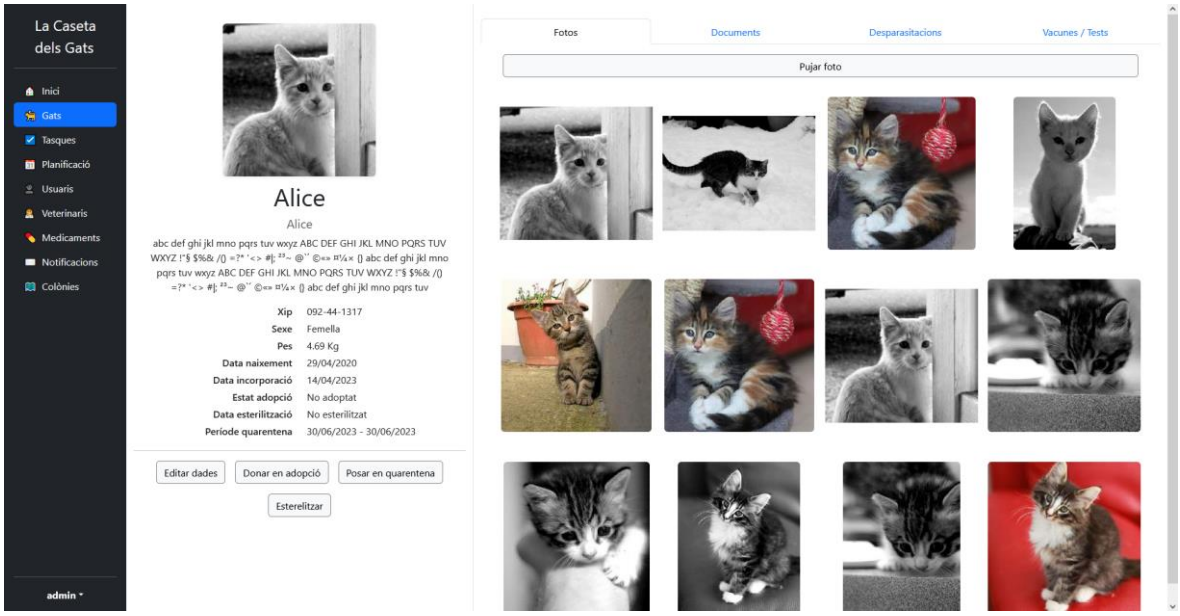


Figura 21. Pàgina amb els detalls d'un gat

En la pàgina de detalls, a la part esquerra, es troba tota la informació d'un gat en concret, com són els seus noms, el seu avatar, biografia i altres dades rellevants. A més, es troben diferents botons amb els que l'usuari pot interactuar. La funció d'aquests són:

Editar dades: Obre el modal amb les dades del gat per a poder editar-les.

Donar en adopció: Marca el gat com a adoptat i genera un contracte d'adopció.

Posar en quarantena: Si el gat no està en quarantena, aquest botó és visible. Esborra la data de fi de quarantena del gat i guarda la data actual com a data d'inici de quarantena.

Treure de quarantena: Si el gat està en quarantena, aquest botó és visible. Guarda la data actual com a data de fi de quarantena.

Esterilitzar: Si el gat no està esterilitzat, aquest botó és visible. Marca la data d'esterilització del gat amb la data actual.

Respecte a la part dreta, tenim quatre pestanyes per indicar les altres dades del gat. Aquestes dades es representen com una graella de fotos clicables, com es mostra a la figura superior o, en el cas de la resta, com una taula similar a les vistes anteriorment.

Figura 22. Finestra on es visualitza una foto a resolució completa

En fer clic en una foto, aquesta es mostra en un modal a pantalla completa. Llavors l'usuari pot fer vàries accions sobre aquesta foto, com visualitzar la imatge en si en una nova pestanya, esborrar-la, o actualitzar l'avatar del gat.

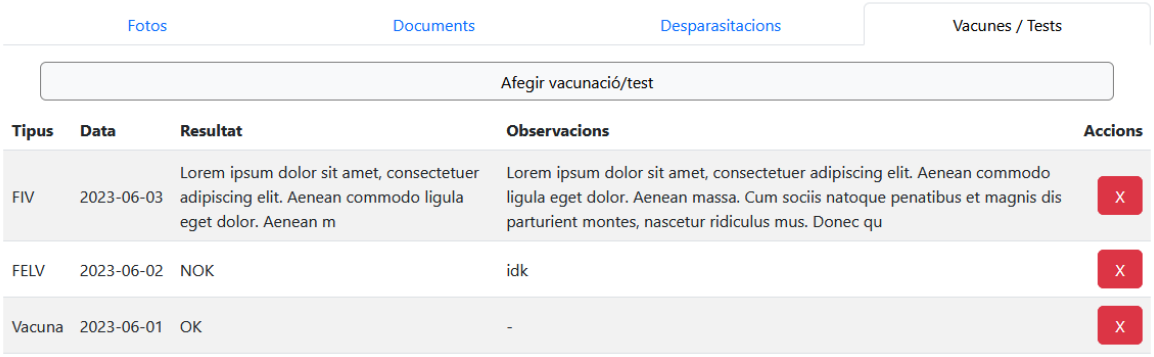


Figura 23. Pestanya de vacunes i tests d'un gat

En aquestes pestanyes l'usuari pot donar d'alta una nova entrada o eliminar-les. De la mateixa forma que s'ha comentat anteriorment, per eliminar-les és necessària una confirmació.

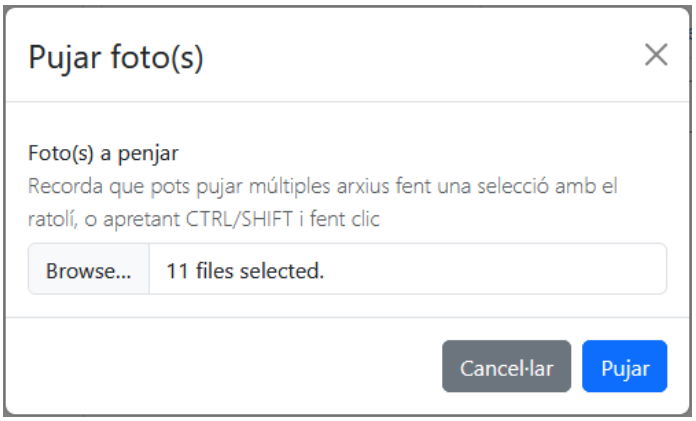


Figura 24. Modal per penjar diversos arxius

Pel que respecta a la pujada de fotos, s'utilitza un modal en el que es poden penjar múltiples arxius. S'indica a l'usuari la forma en la que pot seleccionar diversos arxius a la finestra que li mostri el seu sistema operatiu.

5.3.1.7 Gestió de tasques

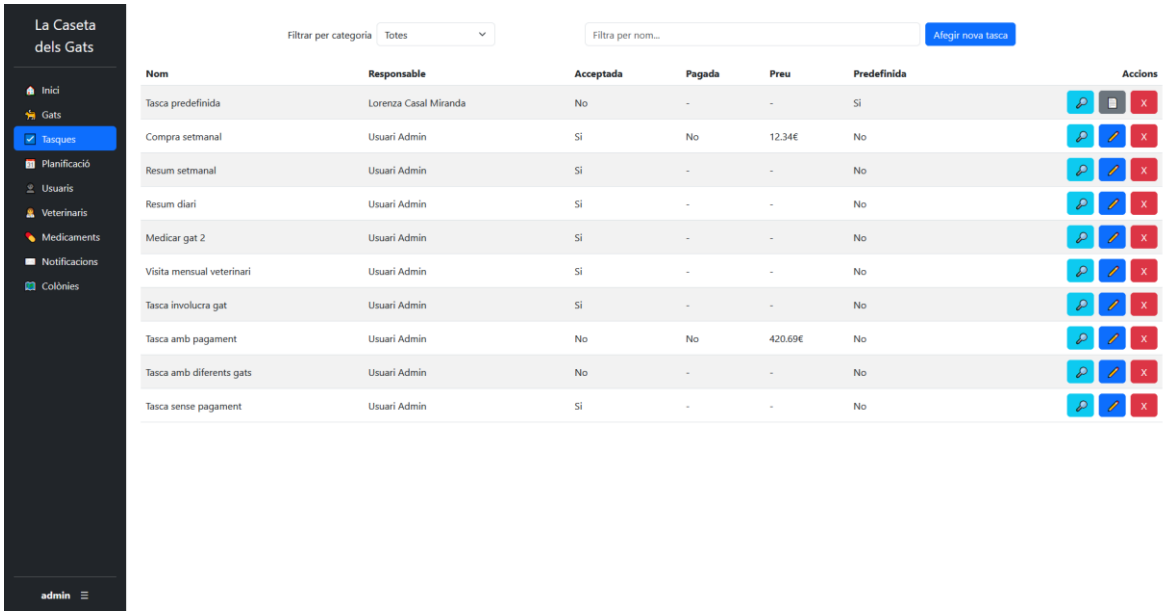


Figura 25. Pantalla de gestió de tasques

La pantalla de gestió de tasques permet a un administrador crear, editar, i esborrar les definicions de tasques. Per filtrar les dades es disposa de dos filtres, per diverses categories (totes, acceptades, manquen pagament, predefinides, o torns d'usuari) i per nom. Aquests filtres funcionen addicionalment l'un amb l'altre.

Els administradors poden també crear tasques predefinides. Aquestes tasques no són editables però permeten ser copiades. Aquesta acció obrirà un formulari amb els camps emplenats amb la informació d'aquesta tasca predefinida,

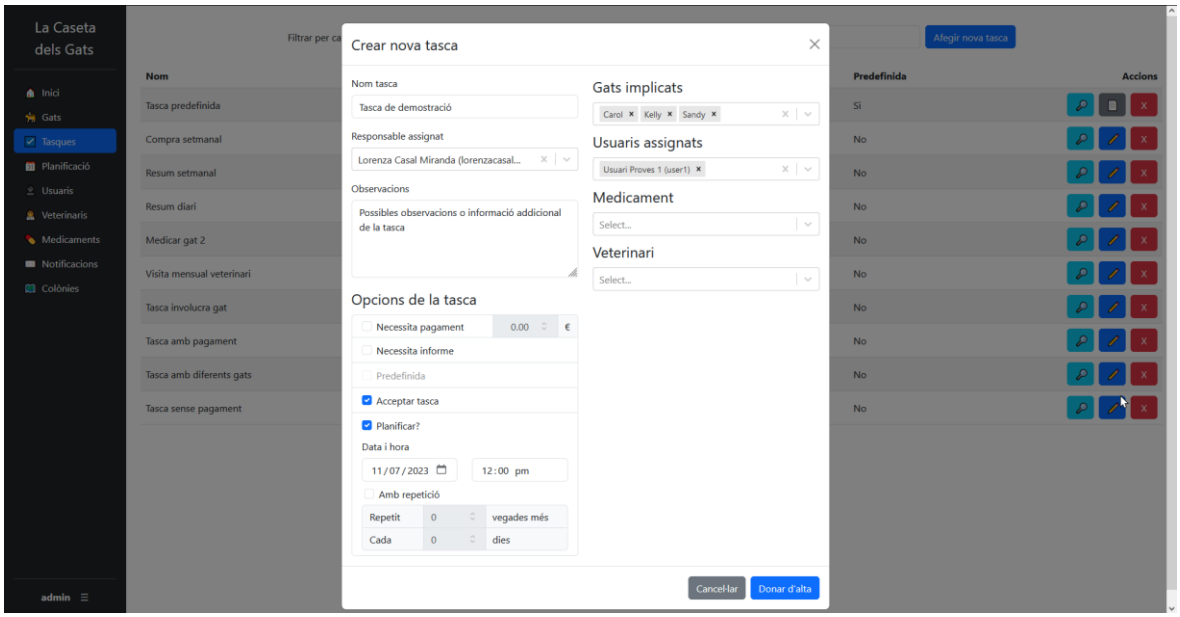


Figura 26. Modal de creació de una tasca

A l'hora de crear o editar una tasca, es mostra un modal on l'administrador pot introduir tota la informació relacionada mitjançant *dropdowns* amb valors individuals o múltiples, en el cas de gats i usuaris, així com *checkboxes* per les opcions de la tasca.

Existeixen alguns restriccions en les opcions, aquestes són:

Una tasca predefinida no pot ser acceptada i viceversa.

Una tasca nova es pot planificar immediatament si es marca com acceptada, repetint la planificació cada X dies durant Y cops, si escau. (per exemple, si es planifica una tasca el dia 1/7 cada 7 dies repetit 3 vegades es creen quatre dates: 1/7, 8/7, 15/7, i 22/7).

Una tasca ja existent no pot ser planificada des d'aquest formulari.

Una tasca ja existent no pot ser editada per ser predefinida.

5.3.1.8 Visualització de la planificació

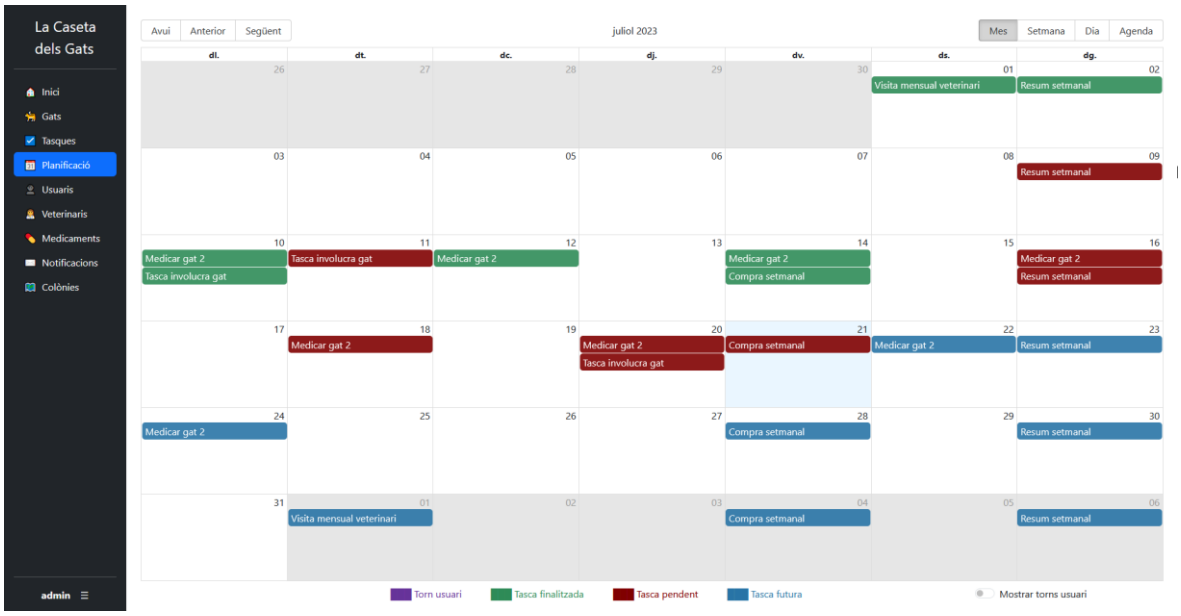


Figura 27. Pantalla de planificació

La planificació es pot visualitzar de dues formes, en la pantalla d'inici i en la pantalla de planificació. Les dues pantalles es diferencien en que la pantalla d'inici mostra un resum, mentre que la planificació ho mostra en complert i afegeix interactuabilitat.

Aquesta interactuabilitat es poder fer clic en qualsevol dia del calendari per planificar una tasca. Aquest clic farà que es mostri un modal amb el dia i hora ja emplenats on l'administrador pot escollir una tasca prèviament acceptada i definir repeticions de la mateixa forma que al crear una tasca acceptada.

Aquesta pantalla, a més, permet visualitzar la planificació de forma dinàmica. El calendari es pot visualitzar mensualment, setmanalment, diàriament, o veient un resum com a agenda.

Per millorar la visualització d'aquest, es mostra també una llegenda on s'expliquen breument el significat dels colors de cada entrada, així com un *switch* per a que no es mostrin els torns de l'usuari (per defecte, no es mostren).

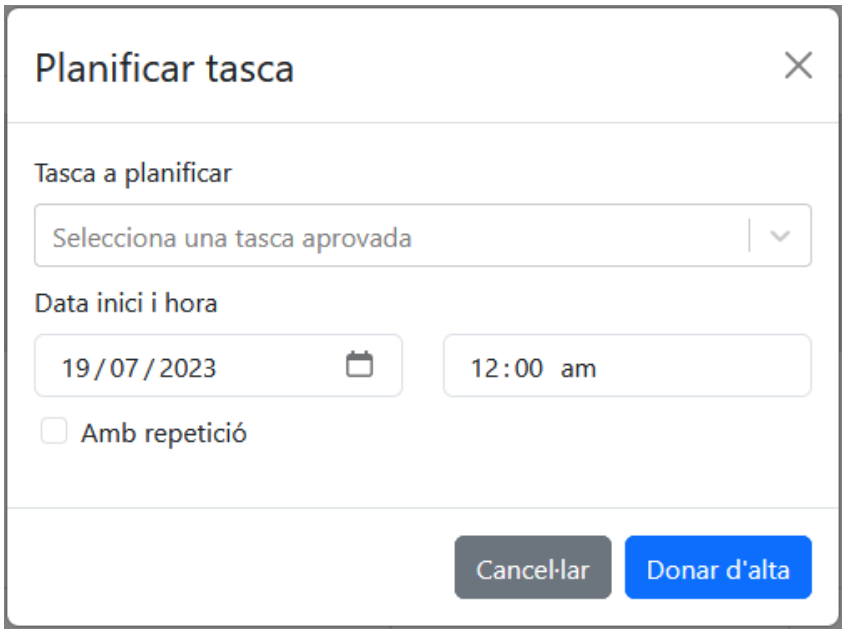


Figura 28. Modal per planificar una tasca

5.3.1.9 Pantalla de resum/inici

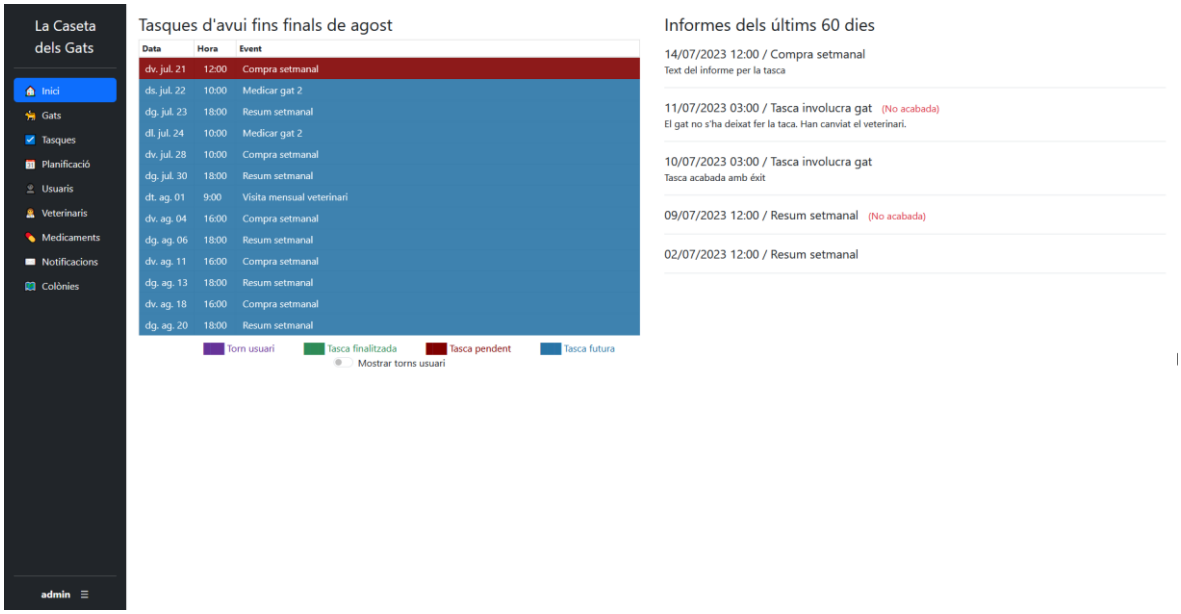


Figura 29. Pantalla d'inici

Quan un administrador inicia sessió, la primera pantalla que se li mostra es la inicial. Aquesta pantalla mostra un breu resum sobre les tasques dels pròxims dos mesos, amb la llegenda i podent filtrar els torns d'usuari com en la planificació, així com els informes i tasques pendents dels últims 60 dies.

5.3.2 Interfície app mòbil

A continuació es detallen les diverses pantalles de l'aplicació mòbil, el seu propòsit, i les funcions que ofereixen.

5.3.2.1 Login

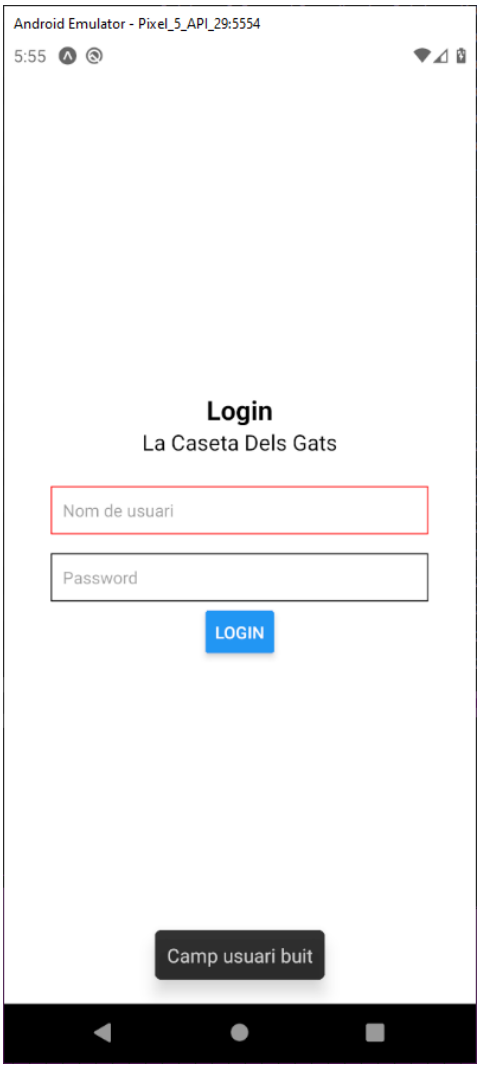


Figura 30. Pantalla de login

La pantalla de login de l'aplicació és una pantalla d'inici de sessió estàndard. Si l'usuari no ha introduït algun camp, o hi ha un error amb l'inici de sessió, es ressalta el camp en qüestió i es mostra un toast amb l'error.

5.3.2.2 Canvi de contrasenya

Figura 31. Pantalla de canvi de contrasenya

La pantalla de canvi de contrasenya permet a l'usuari canviar la seva contrasenya, ja sigui forçat per ser una contrasenya d'un sol ús o per decisió pròpia. Igual que a la pantalla de login, si hi ha algun error, es mostra un toast a l'usuari amb els detalls.

5.3.2.3 Pantalla de tasques

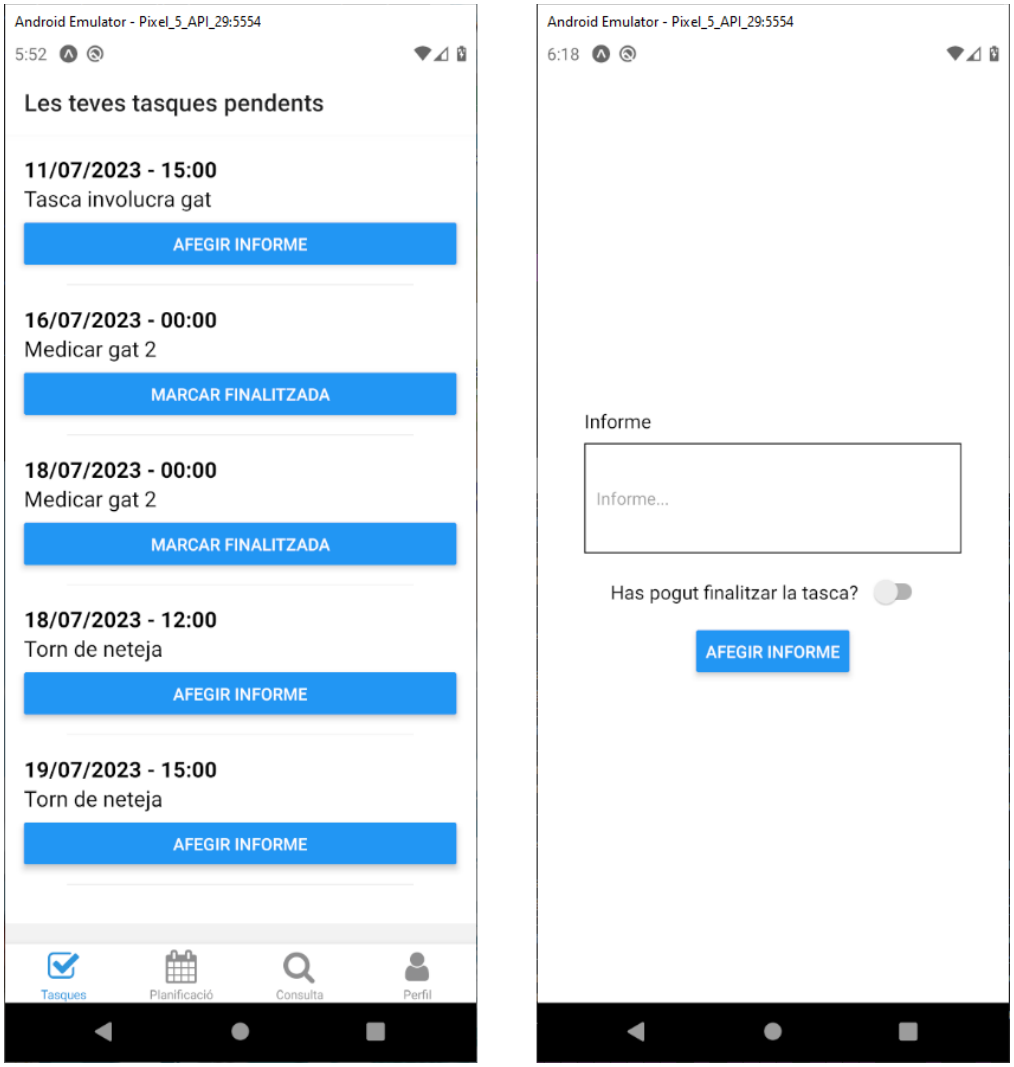


Figura 32. Pantalla de tasques

Aquesta és la pantalla mostrada a l'usuari quan obre l'aplicació. S'informa a l'usuari de les tasques passades pendents d'acabar o informar. Si la tasca requereix d'un informe, es mostra una nova pantalla on l'usuari introduirà el seu informe i podrà marcar si ha pogut acabar la tasca o no.

5.3.2.4 Pantalla de planificació

Figura 33. Pantalla de canvi de contrasenya

La pantalla de planificació proporciona a l'usuari la informació sobre les seves tasques assignades en previsió de dues setmanes. La informació mostrada és la mínima requerida per a l'usuari: dia i hora, nom de la tasca, gats, medicament, veterinari (si escauen) i les possibles observacions de la tasca.

5.3.2.5 Pantalla de consulta

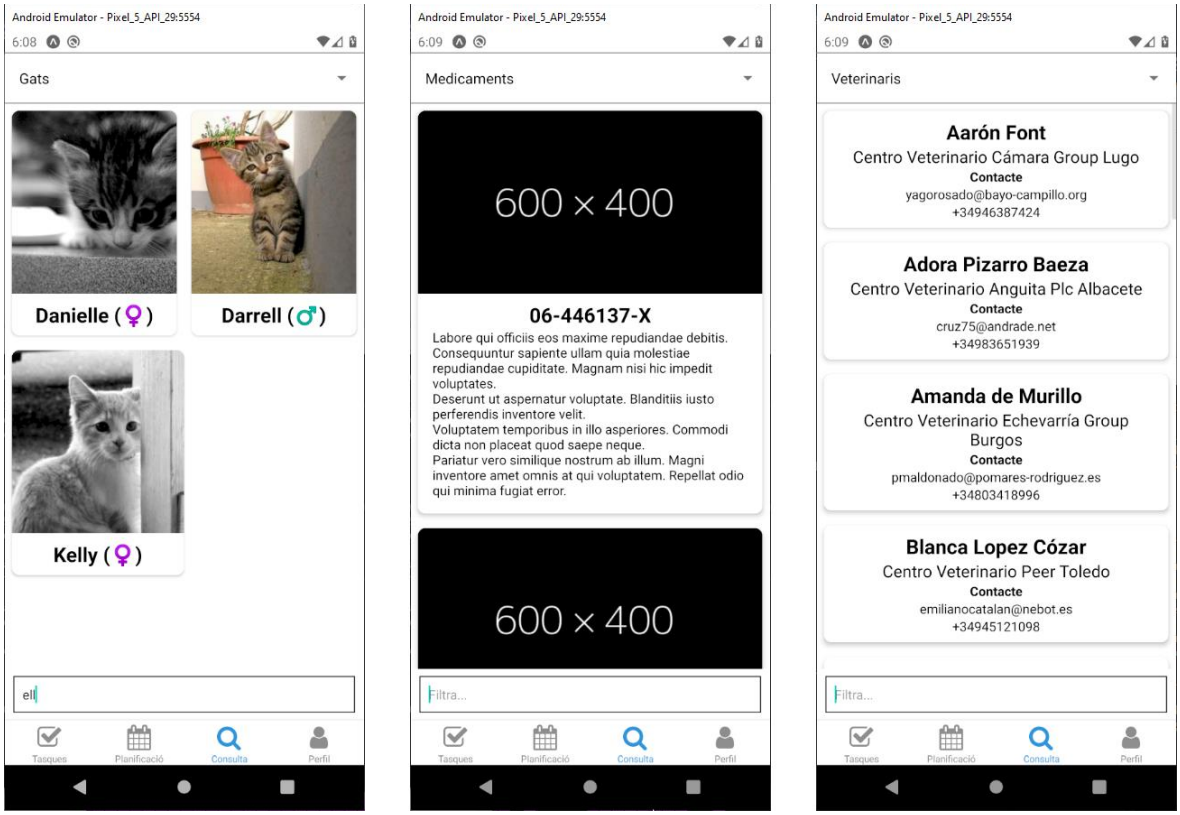


Figura 34. Pantalla de consulta de dades en les diverses vistes

La pantalla de consulta mostra a l'usuari les dades bàsiques d'un gat, medicament, o veterinari en forma de *cards*. Inicialment es mostren només els gats, però l'usuari pot escollir quina informació mostrar mitjançant el *dropdown* situat a la part superior. A més, l'usuari pot filtrar per nom en qualsevol de les vistes.

5.3.2.6 Pantalla de perfil

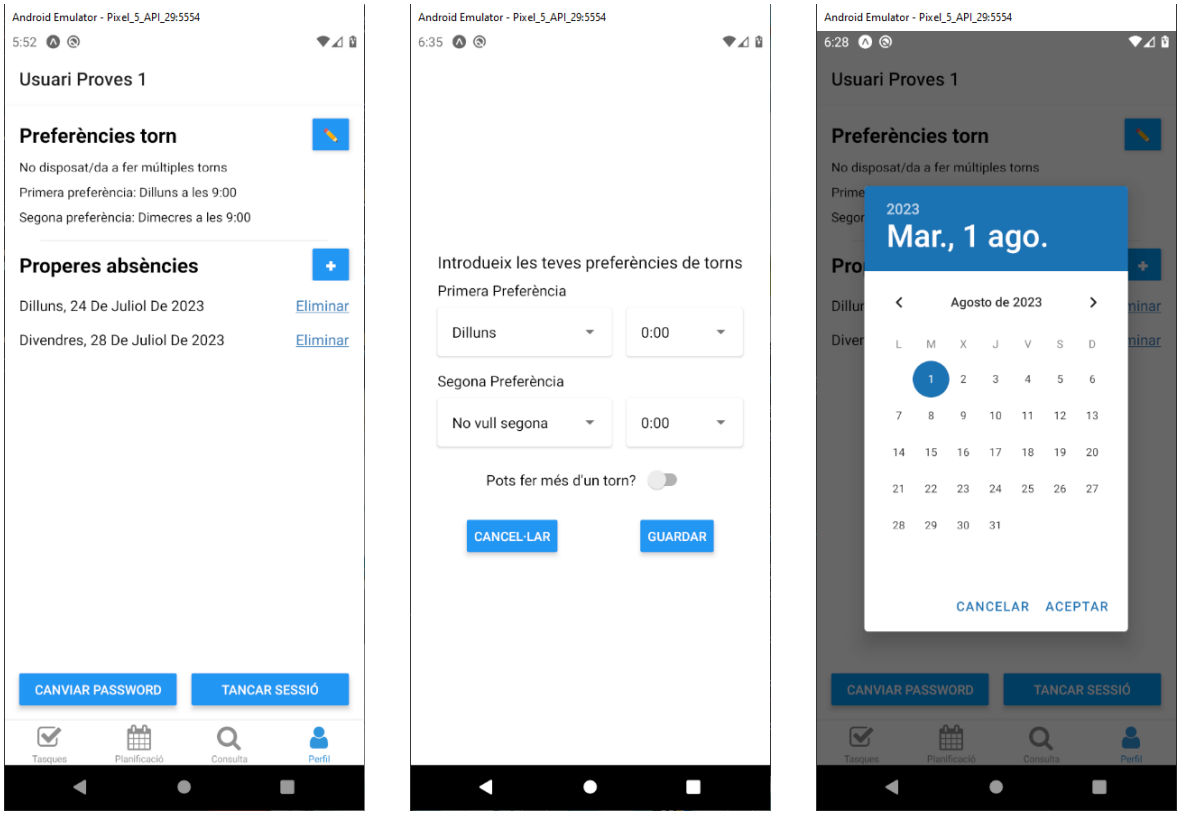


Figura 35. Pantalla de perfil, amb les pantalles addicionals

La pantalla de perfil mostra a l'usuari les seves preferències de torns i les properes absències que tingui informades. L'usuari pot interactuar amb aquesta pantalla per canviar la seva contrasenya, tancar la sessió, canviar les preferències de torn, o afegir una absència.

Per canviar les preferències, l'usuari ha de clicar a la icona amb el llapis i l'aplicació mostrarà el formulari de preferències. Aquest formulari permet a l'usuari introduir fins a dues preferències de dia de setmana i hora.

Finalment, l'usuari pot afegir i eliminar absències. El botó d'afegir mostrarà un calendari on l'usuari escollirà el dia que no hi serà present.

6 Implementació

En aquest apartat es detallarà la forma en la que s'han implementat cadascuna de les tres parts.

6.1 *Backend*

El *backend* se separa en vàries parts, que es comenten seguidament.

6.1.1 El fitxer .htaccess

.htaccess és un fitxer especial que permet al servidor rebutjar connexions a certes rutes, així com re-escriure les URIs de les peticions rebudes, entre altres funcions. En el cas d'aquest projecte el .htaccess es configura per denegar la indexació de carpetes, redireccionar totes les crides des de la URL /api/... cap a /api/index.php (explicat més endavant) i, per seguretat, bloquejar la visibilitat d'aquest mateix fitxer.

El seu contingut és el següent, amb comentaris del que fa cada part

```
# Declarar els fitxers d'indexos i opcio per habilitar la rescriptura de
URLs
DirectoryIndex index.php index.html
Options +FollowSymLinks
RewriteEngine on
# Denegar l'indexacio de carpetes
<IfModule mod_rewrite.c>
    <IfModule mod_negotiation.c>
        Options -MultiViews -Indexes
    </IfModule>
</IfModule>
# Denegar l'acces a .htaccess
<Files .htaccess>
order allow,deny
deny from all
</Files>
# Redireccionar peticions de l'api a /api/index.php
RewriteRule ^api/((?!index\.php$).+)$ api/index.php [L,NC]
# Si la peticio es sobre un arxiu que existeix, permet l'acces
RewriteCond %{REQUEST_FILENAME} -f [OR]
RewriteCond %{REQUEST_FILENAME} -d
RewriteRule ^ - [L]
```

6.1.2 L'estructura de les crides a l'API

Una URI que fa una petició a l'API pot tenir dos formats, un s'utilitza per a peticions que poden ser solucionades de manera simple amb una *query* a BD, mentre que l'altre s'utilitza més bé per fer crides més concretes, com per exemple peticions que requereixen pujar arxius.

TIPUS	FORMAT URI	EXEMPLES
Simple	/api/controlador/acció/mètode	/api/cats/get/all
		/api/users/new/user
		/api/vets/get/details
Específica	/api/controlador/acció	/api/session/login
		/api/cats/new_cat
		/api/cats/upload_document

Taula 1. Els dos tipus de crides a API

6.1.3 Index.php

Com es comenta a l'apartat del .htaccess, totes les crides es redireccionen a /api/index.php. Aquest fitxer llegeix la URI d'entrada i, a partir d'aquesta, instancia el controlador corresponent i crida l'acció i el mètode que es demana. Aquest mateix índex també és el que fa de 'porter': si s'intenta fer una crida i no hi ha un usuari amb sessió iniciada o el controlador o l'acció cridats no existeixen, es retorna un error.

Seguidament es mostra, de forma resumida, el codi font d'index.php

```
// Actions with methods allowed
$generalActions = ['new', 'get', 'update', 'delete'];
// Read the request URL and explode it into an array
// $uri => 0: '', 1: 'api', 2: controller, 3: action, 4: method
$uri = parse_url($_SERVER["REQUEST_URI"], PHP_URL_PATH);
$uri = explode('/', $uri);
$sessionController = SessionController::getInstance();
// If request does not contain a controller and action, return 404
if (!isset($uri[2], $uri[3])) {
      header("HTTP/1.1 404 Not Found");
      exit();
}
// Check user is logged in unless they attempt to call login
if ($uri[2] != 'session' && $uri[3] != 'login') {
      $sessionController->checkLogin();
}
// Instantiate the controller
switch ($uri[2]) {
      case 'cats':
            require ROOT_PATH . "controller/CatController.php";
            $feedController = new CatController();
            break;
[...]
      default:
            header("HTTP/1.1 404 Not Found");
            exit();
}
```

```
// Call the appropiate action
if (isset($feedController)) {
      $actionName = $uri[3] . "Action";

      // If it's a general action a method must be given
      if (in_array($uri[3], $generalActions)) {
            if (isset($uri[4])) {
                  $feedController->{$actionName}($uri[4]);
            } else {
                  header("HTTP/1.1 404 Not Found");
                  exit();
            }

      } else {
            $feedController->{$actionName}();
      }
}
```

6.1.4 Controlador base

El controlador base és un tipus de classe abstracta del que la resta de controladors hereten. Aquesta classe té definits mètodes que permeten simplificar el processament i la resposta de les peticions HTML, principalment obtenir els paràmetres passats per GET i POST, i enviar una resposta.

Els mètodes que conté són els següents:

MÈTODE	**DESCRIPCIÓ**
`__call()`	Mètode especial de les classes PHP que es crida quan s'intenta cridar un mètode que no existeix. A l'aplicació s'utilitza per enviar una resposta 404 si s'accedeix a una acció no existent.
`getParameters(array $params)`	Aquest mètode comprova que tots els paràmetres definits en l'array existeixen a la petició GET. Retorna un array amb claus/valors dels paràmetres demanats, o error si un dels paràmetres no existeix.
`getPostParameters(array $params)`	Mateixa funcionalitat que el mètode anterior, però es comprova la petició per POST.
`checkRequestMethod(string $request)`	Comprova que la petició és del tipus que es passa per paràmetre (GET/POST). En cas contrari es respon amb error 405.
`sendOutput(array $data, int $httpCode)`	Aquest mètode construeix el *header* HTTP per generar una resposta a la petició. Aquest *header* conté tot el string estàndard HTTP (per exemple "HTTP/1.0 418 I'm a teapot") i el tipus de contingut, que és 'application/json' per a l'API Les dades ($data) es codifiquen en un json i s'envien pel cos de la resposta.

Taula 2. Els mètodes de la classe BaseController

6.1.5 Controlador de sessió

El controlador de sessió és un controlador diferent a la resta, que s'encarrega de la gestió de les sessions dels usuaris. Aquest segueix el patró *singleton* per a tenir una única instància durant tot el cicle d'una petició.

Els mètodes destacables que conté són els següents:

MÈTODE	DESCRIPCIÓ
`getData()`	Retorna les dades guardades en sessió.
`checkRolePermissions(int $role)`	Comprova si l'usuari loguejat compleix el nivell de rol requerit; envia resposta amb error 401 en cas contrari.
`checkLogin()`	Comprova si existeix una sessió; envia resposta amb error 401 si no es dona el cas.
`get_user_infoAction()`	Acció accessible per l'API, que retorna el nom i el rol de l'usuari en sessió, si existeix. La ID de sessió s'envia per POST.
`loginAction()`	Acció accessible per l'API, que intenta loguejar a un usuari si es compleixen certes condicions. Per POST s'envia el nom i el password de l´usuari i el rol requerit per l'aplicació. Aquesta acció respon amb diferents codis de resposta segons la condició fallida: 0: Usuari loguejat correctament. 1: Usuari no existeix. 2: Usuari desactivat. 3: Contrasenya incorrecta. 4: No compleix el rol.
`logoutAction()`	Acció accessible per l'API que simplement destrueix la sessió d´un usuari.

Taula 3. Els mètodes de la classe SessionController

6.1.6 Controladors

A l'API existeix un controlador per cada entitat (Gats, Medicaments, Veterinaris, Usuaris, Tasques i Colònies). Tot i així, la seva funcionalitat individual és pràcticament idèntica. La decisió de separar-los ve arran que un controlador farà únicament les funcions específiques per a aquella entitat.

A continuació es mostra el codi font del que podria ser l'esquelet d'un nou controlador (se simplifica allà on sigui necessari per reduir la mida) i exemples de la utilitat del controlador base i de sessió:

```
require_once ROOT_PATH . "model/EntityModel.php";
class EntityController extends BaseController {
  private SessionController $sessionController;
  private EntityModel $entityModel;

  // Get instances for the SessionController and the model
  public function __construct() {
    $this->sessionController = SessionController::getInstance();
    $this->entityModel = new EntityModel();
  }
  // Action for selects
  public function getAction($method) {
    $this->sessionController->checkRolePermissions(ROLE_VOLUNTEER);
    $this->checkRequestMethod(REQUEST_GET);
    try {
      $output = null;
      switch ($method) {
        case 'all':
          $output = $this->entityModel->getData();
          break;
        case 'details':
          $p = $this->getParameters(['id']);
          $output = $this->entityModel->getDetails($p['id'])[0];
          break;
        [...]
        default:
          throw new Exception("Unsupported method: $method");
      }
      $this->sendOutput($output, 200);
    } catch (Exception $e) {
      $this->sendOutput(['exception' => $e->getMessage()], 500);
    }
  }
  [...]
}
```

6.1.7 Models i connexió a BD

Tots els models utilitzen una classe que s'encarrega de fer la connexió i executar les *quèries* necessàries utilitzant el mòdul mysqli de PHP. Aquesta classe utilitza quatre mètodes diferents per realitzar les diferents operacions del CRUD mitjançant *binding* de paràmetres.

A mysqli el *binding* de paràmetres es fa preparant una *query*, substituint els valors que interessin per ? i passant un *array* on s'indiquen els tipus dels paràmetres així com les dades en si. Aquest *binding* transforma les dades i les substitueix, per ordre, a la *query*.

Per exemple:

```
Query: INSERT INTO gats_fotos (id_gat, url_foto) VALUES (?, ?)
Paràmetres: ['is', 12, 'foto.jpg']
Resultat: INSERT INTO gats_fotos (id_gat, url_foto) VALUES (12, foto.jpg)
```

Llavors, els models, utilitzant la classe de connexió, simplement contenen mètodes que tradueixen una crida de funció estàndard en una *query* i *array* per al *binding* de paràmetres.

Utilitzant el mateix exemple anterior:

```
public function newPhoto($catID, $fileName)
{
      return $this->insert("INSERT INTO gats_fotos (id_gat, url_foto)
      VALUES(?, ?)", ["is", $catID, $fileName]);
}
```

6.1.8 FileModel

Aquest és un model apart que no interactua amb la base de dades. La seva funció és gestionar l'estructura de fitxers de l'aplicació, és a dir, comprovar si la pujada de fitxers ha estat correcta, moure els fitxers pujats allà on toqui, esborrar-los si és necessari, i generar *thumbnails* per a les imatges.

Comptant només els arxius pujats al servidor, l'estructura de directoris que es vol aconseguir és la següent:

```
/
├ doc/
|     └ {id_gat}/
|           ├ doc1.docx
|           └ doc2.pdf
└ img/
      ├ {id_gat}/
      |     ├ thumbnails/
      |     |           ├ photo1.jpg
      |     |           └ photo2.jpg
      |     ├ photo1.png
      |     └ photo2.jpg
      └ meds/
            ├ thumbnails/
            |           └ {id_medicament}.jpg
            └ {id_medicament}.jpg
```

Per aconseguir mantenir aquesta estructura i facilitar l'accés des dels *frontends*, a l'hora d'inserir un nou gat o medicament, aquest model s'encarrega de crear els directoris adients utilitzant una imatge sense contingut en el cas que no es pugi una foto juntament amb les dades. D'aquesta forma sempre hi haurà una foto per mostrar.

6.1.9 Treballs planificats

Es requereixen de dos treballs a planificar periòdicament amb un cron.

6.1.9.1 Generació de la planificació de torns de neteja

Per calcular els torns dels usuaris en una setmana es crea un algoritme que, amb les preferències de tots els usuaris i les seves ausències informades, genera una planificació per una setmana. Aquest algoritme es pensat per generar una planificació de la següent setmana a quan es executat per a que els usuaris tinguin una previsió amb temps dels seus torns.

Sense tenir en compte el càlcul de dates, el pseudocodi de l'algoritme és el següent:

```
preferencies = obtenirPreferenciesBD()
ausencies = obtenirAusenciesBD()
planificacio = [][]
usuaris_planificats = []
per cada(pref en preferencies) {
  // Si existeix una ausencia en el dia escollit, saltar
  si(!existeixAusencia(pref.dia)) {
    // Usuari ja ha estat planificat i no pot fer multiples torns, saltar
    si(pref.multiples_torns || !usuaris_planificats[pref.usuari]) {
      // Si el torn no s'ha agafat, assignar-lo
      si(planificacio[pref.dia][pref.hora]) {
        planificacio[pref.dia][pref.hora] = pref
        usuaris_planificats[pref.usuari] = 1
      } sino {
        // Es pot substituir un torn assignat amb 50% de possibilitats
        si(aleatori(0, 1) && pref.num_prioritats == 1) {
          usuaris_planificats[planificacio[pref.dia][pref.hora].usuari]=0
          planificacio[pref.dia][pref.hora] = pref
          usuaris_planificats[pref.usuari] = 1
        }
      }
    }
  }
}
guardarPlanificacioBD(planificacio)
guardarLogPlanificacio(planificacio)
```

Per fer que les preferències dels usuaris siguin el més justes possibles es tenen dos mecanismes:

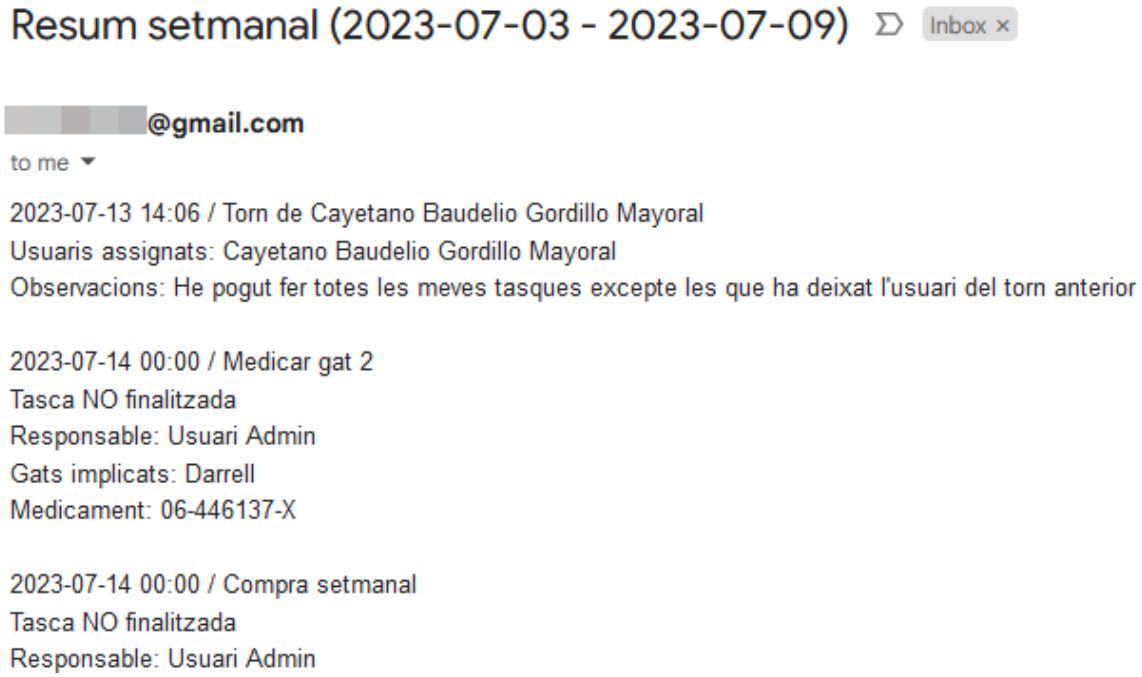
1. S'obtenen les dades de la base de dades ordenats per:
    a. Numero de preferències: Els usuaris amb menys preferències s'assignen primer. D'aquesta forma els usuaris amb una única preferència tindran més possibilitats de ser assignats un torn abans de que l'agafi un usuari amb dos preferències que pot tenir més flexibilitat horària.
    b. Prioritat: Les primeres preferències s'assignen abans.
2. Quan dos usuaris coincideixen i ambdós només tenen una prioritat, es substitueix amb un 50% de possibilitats.

6.1.9.2 Resum setmanal

Es requereix d'un resum de tasques setmanal, on s'informa per correu electrònic de l'estat de les tasques (finalitzades o no) així com els informes associats a aquestes.

Aquesta tasca s'aconsegueix mitjançant una simple consulta a base de dades semblant a la que es fa per obtenir les tasques d'un usuari en concret. Un cop obtingudes llegeixen les tasques una a una i es transformen en un format de text pla que s'envia mitjançant la funció *mail* de PHP.

El correu obtingut un cop s'executa el job és el següent:


Resum setmanal (2023-07-03 - 2023-07-09) Inbox ×

[redacted]@gmail.com

to me

2023-07-13 14:06 / Torn de Cayetano Baudelio Gordillo Mayoral
Usuaris assignats: Cayetano Baudelio Gordillo Mayoral
Observacions: He pogut fer totes les meves tasques excepte les que ha deixat l'usuari del torn anterior

2023-07-14 00:00 / Medicar gat 2
Tasca NO finalitzada
Responsable: Usuari Admin
Gats implicats: Darrell
Medicament: 06-446137-X

2023-07-14 00:00 / Compra setmanal
Tasca NO finalitzada
Responsable: Usuari Admin

Figura 36. Exemple de correu del resum setmanal

6.2 *Frontend* administratiu

L'aplicació administrativa, com s'ha comentat, és una aplicació en ReactJS i funciona mitjançant components JavaScript. Un component de React és una classe reutilitzable que renderitza un codi HTML específic i controla els esdeveniments d'aquest.

Per exemple, el component CatCard.js renderitza un element que conté l'avatar d'un gat, el seu nom posat per la caseta, i el nom posat per l'adoptant, si existeix. A més, defineix un esdeveniment que en fer clic crida el *callback* d'un component pare.

```
function CatCard(props) {
  return (
    <Card className="mx-4 mb-4 shadow-sm text-center"
        onClick={() => props.catClicked(props.dataid)}
        style={{ cursor: "pointer" }}>
      <Card.Img src={ApiUrls.getCatImageThumbnail(
        props.dataid, props.avatar)}
        className="object-fit-cover image-thumbnail" />
      <Card.Body>
        <Card.Title>{props.name}</Card.Title>
        {props.adoptName ?
          <Card.Subtitle className="mb-2 text-muted">
            {props.adoptName}
          </Card.Subtitle>
          : <></>}
        <Card.Text>
        </Card.Text>
      </Card.Body>
    </Card>
  );
}
export default CatCard;
```

6.2.1 Classes d'utilitat

Per tal de simplificar certs components, es creen un parell de classes d'ajuda. Aquestes classes no renderitzen cap element sinó que proporcionen una funció en específic.

6.2.1.1 ApiUrls.ts

Ja que l'estructura de directoris per a fotos i documents és sempre el mateix, es pot accedir a ells de forma directa. Aquesta classe genera strings contenint la URL d'una imatge o document segons faci falta.

```
public static getCatImage(catid: number, filename: string): string {
      return this.BASE_URL + this.IMGPATH + catid + '/' + filename;
}
```

6.2.1.2 AppLiterals.ts

Camps com el rol, l'estat d'adopció, els tipus de vacunes o desparasitacions, booleans, ... es guarden com a números a BD. Per tant, és necessari traduir-los a un format llegible. Aquesta classe s'encarrega de definir unes llistes estàtiques de strings que, mitjançant la funció associada, tradueixen de forma directa aquests valors a un string.

```
private static vaccineLiterals = ["Vacuna", "FELV", "FIV"];
public static getVaccineLiteral(type: number): string {
      return this.vaccineLiterals[type];
}
```

6.2.1.3 FormatData.js

Aquesta classe utilitza la llibreria 'moment' per traduir les dates de BD en format ISO (YYYY-MM-DD) al format que utilitzem a Espanya (DD-MM-YYYY) per a que sigui més llegible per a l'usuari.

6.2.2 Connexions a API – AxiosRequest

Per fer crides a l'API, es fa ús de la llibreria 'axios'. Aquesta llibreria permet realitzar peticions AJAX de forma simple mitjançant el concepte de "promeses". Axios incorpora mètodes per tractar els *headers* de la petició, la resposta de la petició i els possibles errors, però una de les utilitats més importats és la deserialització automàtica de respostes en JSON.

Per aconseguir-ho se segueix una estructura similar a la ja vista als controladors del *backend*. Primer es defineix una classe que crea una instancia d'axios amb uns paràmetres en concret, retornar la resposta si és correcta, i tractar els possibles errors.

```
// Create the instance itself
const instance = axios.create({
  baseURL: ApiUrls.BASE_URL,
  withCredentials: true,
  headers: {
    'Content-Type': 'application/x-www-form-urlencoded'
  }
});

// Axios request wrapper to check response
const axiosRequest = function (options) {
  const onResponse = function (response) {
    if(response && (response.status === 200 || response.status === 201)){
      return response;
    }
    return Promise.reject(response);
  }
  const onError = function (error) {
    if (error.response) {
      // Request was made successfully but server responded with error
      console.error('Status:', error.response.status);
      console.error('Data:', error.response.data);
      console.error('Headers:', error.response.headers);
      throw error;
    } else {
      // Ignore React.StrictMode re-mounts
```

```
      if (error.code !== "ERR_CANCELED") {
        throw error;
      }
    }
    // Fallback
    return Promise.reject(error.response || error.message);
  }
  // Return the created instance
  return instance(options).then(onResponse).catch(onError);
}
export default axiosRequest;
```

Finalment, per fer les crides es crea una classe per a cada controlador que reflecteixen tots els mètodes oferts pel controlador del mateix nom. Les crides en si es fan utilitzant la classe mencionada anteriorment. L'encapsulació d'aquesta capa permet al *frontend* accedir al *backend* de forma transparent amb un llenguatge "natural" sense haver de preocupar-se a cada crida de definir *headers*, mètode de *request*, la URL o com codificar les dades.

Un exemple de com és implementada una d'aquestes classes, i una possible crida, és la següent:

```
import axiosRequest from "./AxiosRequest"
async function create(data) {
      return axiosRequest({
            url: "/api/cats/new_cat",
            method: 'POST',
            data: data,
            headers: {
                  'Content-Type': 'multipart/form-data'
            }
      });
}
async function list(signal) {
      return axiosRequest({
            signal: signal,
            url: "/api/cats/get/all",
            method: 'GET'
      });
}
[...]
const CatAPI = { create, list, [...] }
export default CatAPI;

// Exemple de crida
CatAPI.list(abortController.signal)
      .then((response) => {
            // handle response.data
      }).catch((error) => {
            // handle error.message
      });
```

6.2.3 Estructura i components

L'estructura que segueix el *frontend* administratiu es basa en components de pàgines, és a dir, cada pàgina és un component que instancia components fills i controla el contingut (estat) d'aquests. React recomana que la interacció entre components sigui així: el pare passa dades al fill i el fill notifica els esdeveniments al pare.

Els components pares, a més de gestionar l'estat dels components fills, també s'encarreguen d'interceptar els esdeveniments (*callbacks*). D'aquesta forma els fills no tenen lògica i només mostren dades. L'inconvenient de fer-ho així és sobrecarregar el component pare amb mètodes i variables d'estat, però l'avantatge de poder reutilitzar els components fills, supera els inconvenients.

En els següents apartats es detalla el funcionament de les diferents parts d'un component React per tal de no inserir en format text tots els components implementats.

6.2.3.1 Esquelet de component

Un component renderitzable necessita com a mínim una declaració i retornar un conjunt d'etiquetes JSX. En els components es pot definir un paràmetre 'props' que contindrà les variables assignades en la instanciació del component.

```
function ExampleComponent(props) {
      return (
            <h1>{props.message}</h1>
      );
}
export default ExampleComponent;

// Exemple crida component
<ExampleComponent message='Hello world!' />
```

6.2.3.2 Estat del component (useState)

React no re-renderitza un component a no ser que el seu estat canviï. Per poder canviar el contingut dinàmicament, un component ha d'utilitzar variables que implementin estats de React amb useState en comptes de variables de JS. Tot i així, es poden utilitzar constants que incorporin aquests estats i el seu contingut també canviarà.

useState retorna un array amb dos elements: la variable amb les dades en si i una funció per actualitzar el contingut d'aquesta (getter/setter).

Per exemple, per actualitzar un missatge en fer clic en un botó:

```
function ExampleComponent(props) {
      const [number, setNumber] = useState(0);
      const click = e => setNumber(number + 1);
      const message = `Count: ${number}`;
      return (<>
            <h1>{message}</h1>
            <button onClick={(e) => click(e)}>Click</button>
      </>);
}
export default ExampleComponent;
```

6.2.3.3 Executar codi en carregar el component i en canviar la dependència (useEffect)

Els efectes en React permeten a un component executar un codi un sol cop quan es carrega el component i opcionalment quan una de les seves dependències canvia. Aquesta peculiaritat esdevé de que React executa tot el codi (que no siguin funcions) d'un component cada cop que es renderitza. Per tant, executar una funció en el cos del component causa que aquest component es re-renderitzi infinitament, ja que contínuament canviaria el seu estat.

En aquesta aplicació, els efectes s'utilitzen per agafar dades de l'API. Indicant una llista de dependències buida s'assegura que el component en qüestió faci una única petició i conservi les dades en l'estat durant tota la vida del component.

```
useEffect(() => {
  const abortController = new AbortController();
  CatAPI.list(abortController.signal)
      .then([...]).catch([...]);

  return () => {
    abortController.abort();
  }
}, []); // [] indica les dependencies
```

6.2.3.4 Llista de components

A la següent taula es llisten els components tipus pàgina utilitzats i una breu descripció de la seva funcionalitat.

COMPONENT	DESCRIPCIÓ
`MainPage`	Mostra la *sidebar* i carrega altres pàgines com a contingut principal.
`LoginPage`	Pantalla de login.

`ColonyListPage`
`MedicineListPage`
`UserListPage`
`VetListPage`
	Components que mostren la llista d'entrades d'una classe, juntament amb la possibilitat de filtrar i afegir una nova entrada.

`CatListPage`	Mostra una llista de CatCard, amb la possibilitat de filtrar i afegir un nou gat.
`CatDetailsPage`	Detalls d'un gat, mostra totes les fotos, documents, desparasitacions i vacunes, amb possibilitat d'afegir o editar qualsevol dada.
`TaskListPage`	Llista de tasques al sistema; es dona la possibilitat de filtrar segons el nom o estat, definir i planificar noves tasques.
`SchedulePage`	Mostra la planificació de torns i tasques de l'entitat.

Taula 4. Components de la pàgina de l'aplicació administrativa

A la següent taula es llista la resta de components utilitzats. Aquest components mostren una dada en concret, o serveixen per mostrar un formulari o altres interaccions amb l'usuari.

COMPONENT	DESCRIPCIÓ
`CatCard`	Una única card clicable amb l'avatar d'un gat i el seu nom.
`CenteredLabelValue`	Conjunt estilitzat d'etiqueta/valor, centrat horitzontalment. Es pot veure un exemple del seu ús a la pantalla de detalls del gat.
`LoadingSpinner`	*Spinner* que bloqueja la interacció amb la pàgina mentre està carregant. Aquest component es mostra quan es carrega un nou component i s'amaga quan aquest component ha acabat de muntar-se.
`NavPill`	Consisteix en un *link* clicable utilitzat a la *sidebar*. Aquest *link* es compon d'un emoji, una etiqueta, i una ID única per controlar el clic.
`ObjectTable`	Component que accepta una llista d'objectes (del mateix tipus) i els representa a una taula HTML.
`SingleToast`	*Toast* situat a la posició inferior-centre de la pantalla, on es pot canviar el títol, contingut i color de fons. Utilitzat principalment per mostrar errors o un missatge puntual a l'usuari.
`CatImageModal`	Modal que mostra a (gairebé) pantalla completa la imatge d'un gat. Aquest component permet esborrar la foto, assignar-la com avatar del gat o obrir la imatge en una nova pestanya.
`ConfirmationModal`	Modal que mostra un contingut variable i que no es pot tancar fins que l'usuari marqui un *checkbox*.
`ExplicitAcionModal`	Modal que mostra un títol i contingut configurables, que no es pot tancar a no ser que l'usuari confirmi o cancel·li l'acció demanada.

`CatFormModal`
`ColonyFormModal`
`DewormingFormModal`
`MedicineFormModal`
`TaskFormModal`
`UserFormModal`
`VaccineFormModal`
`VetFormModal`
	Modals amb un formulari on l'usuari pot introduir dades i on el contingut varia segons la classe que representi.
	Aquests components admeten una entrada de la classe que representa. Si la classe no és nul, s'emplena el formulari amb les dades de la classe. D'aquesta forma es pot reutilitzar el component per fer alta o edició d'una entrada.
	La comunicació de les dades a l'API no la fa aquest component, sinó que es passen les dades al component pare i aquest és l'encarregat de fer la petició.

`UploadDocModal` `UploadPhotosModal`	Mostren un petit formulari per penjar arxius. De la mateixa forma que els formularis anteriors, les dades es passen al component pare mitjançant un *callback*.

Taula 5. Components reutilitzables de l'aplicació administrativa

6.3 *Frontend* mòbil

L'aplicació mòbil s'ha desenvolupat utilitzant el framework React Native juntament amb Expo. Expo és un *framework* que ofereix diversos mòduls, eines i serveis que faciliten el desenvolupament i desplegament d'aplicacions React.

Donat que l'aplicació utilitza React Native, el cicle de vida i la funcionalitat que ofereix són idèntics, amb la diferència que no s'escriu codi HTML (però sí CSS per als estils) sinó que s'utilitzen uns components JSX més generals que després React tradueix a components natius.

Tot i així es poden aprofitar alguns elements de l'aplicació web, com ara les classes, per fer peticions i processar les respostes del servidor, traducció de literals, o construir les URLs de les imatges.

6.3.1 Navegació

Per navegar entre pantalles s'utilitza un mòdul d'Expo anomenat *router*. Aquest mòdul és un *wrapper* del mòdul React Navigation que permet definir la navegació de les pantalles de l'aplicació, així com agrupar pantalles per a que heretin un mateix estil. Per comparar-ho amb una funcionalitat nativa d'Android, les pantalles i modals correspondrien a *activities* i *fragments*, on la navegació correspon a invocar *intents*.

La funcionalitat addicional que ofereix Expo Router sobre React Navigation és poder utilitzar les rutes de fitxers com a pantalles o modals, en comptes d'haver de definir URLs internes de l'aplicació. Això ho aconsegueix mitjançant fitxers anomenats *_layout*, que defineixen un estil (opcional) de les pantalles a la carpeta on es troben així com subcarpetes. Per exemple, pot definir pestanyes, un *drawer*, submenús, ... d'un conjunt de pantalles en concret que es trobin a la mateixa carpeta.

Les pantalles són distribuïdes en el sistema de fitxers de la següent forma:

```
/
├ (tabs)
|      ├ index
|      ├ profile
|      ├ schedule
|      └ lookup
├ changepassword
└ login
```

Per exemple, si es volen definir unes pantalles amb pestanyes, cal crear un fitxer *_layout* dins de la carpeta *(tabs)* amb un contingut similar al següent:

```
export default function TabLayout() {
  return (
    <Tabs>
      <Tabs.Screen name="schedule"
        options={{
          title: 'Planificació',
          tabBarIcon: () => <TabBarIcon name="calendar"/>,
        }}
      />
      ...
    </Tabs>
  );
}
```

D'aquesta forma, Expo Router s'encarrega automàticament de crear, gestionar els estils, noms, rutes, i interactivitat d'una barra de pestanyes amb els valors definits.

6.3.2 Comprovació de sessió

Aprofitant que Expo Router per defecte sempre carrega el fitxer *index* d'una carpeta, en arrencar l'aplicació es fan una sèrie de comprovacions per saber a quina pantalla ha d'anar l'usuari.

En ordre:

1. Si no existeixen dades de l'usuari a la memòria de l'aplicació, es navega a la pantalla *login*.
2. Si existeixen dades a memòria i el camp *first_login* és 1 (ha iniciat sessió i ha tancat l'aplicació abans de canviar la seva contrasenya d'un sol ús) es navega a la pantalla *changepassword*.
3. Es contacta amb el servidor per validar la sessió existent.
    a. Si retorna error 401 vol dir que la sessió ha expirat; llavors es navega a *login*.
    b. Si retorna el camp *first_login* a 1 vol dir que un administrador ha reiniciat la contrasenya de l'usuari; llavors es navega a la pantalla *changepassword*.
    c. En qualsevol altre cas, es procedeix a la pantalla principal *(tabs)*.

6.3.3 Resta de pantalles

Pel que respecta a la implementació de la resta de l'aplicació, se segueix el mateix flux de components que a l'aplicació web. S'utilitzen *states* per controlar les dades dels elements visuals o dades introduïdes i *effects* per fer crides a l'API en carregar un component o llençar esdevenimnts en certes dependències.

7 Avaluació

Per avaluar el funcionament de l'aplicació s'han fet proves no automàtiques donada la naturalesa de l'aplicació, és a dir, sent formats per elements més aviat visuals, no dona del tot la possibilitat d'aplicar tests unitaris.

Primerament, un cop desenvolupat el *backend*, s'han format diversos fitxers de prova per comprovar que la funcionalitat d'aquesta és correcta i verificar els JSONs retornats per l'API. Aquesta part s'ha aconseguit comprovant que el funcionament de l'API és correcte, utilitzant *links* directes als *endpoints* de l'API per als *selects*, i formularis HTML amb les dades mínimes per verificar que les operacions del CRUD funcionen correctament.

Finalment, un cop s'ha confirmat que l'API funciona correctament, s'ha utilitzat la llibreria Faker de Python per a emplenar automàticament la BD amb una sèrie de dades fictícies. Aquest pas estalvia feina a l'hora de crear dades útils per als *frontends*.

Un exemple del script utilitzat per fer el *seed* dels usuaris:

```
if __name__ == "__main__":
  fake = Faker()
  fake_es = Faker('es_ES')
  dbm = LocalDB()

  for _ in range(10):
    name = fake_es.name()
    username = unicodedata.normalize('NFD', name)
    username = username.replace(' ', '').replace('\'', '').replace('-',
'').lower()
    password = ''.join(fake.words(nb=3))
    role = random.randint(0,3)
    dbm.insert_values(
      """
      INSERT INTO usuaris (nom_usuari, nom_real, rol_usuari, password)
      VALUES(%s, %s, %s, %s)
      """,
      (username, name, role, bcrypt.hashpw(password.encode(),
bcrypt.gensalt())),
    )
    print(f"{username} -> {password}")
```

Pel que respecta als *frontends*, com s'ha comentat, no s'han fet proves automàtiques sinó que es comprova manualment que funciona correctament, juntament amb un sistema de logs simple al *backend* per comprovar que la comunicació entre aquestes dues parts és correcta.

8 Conclusions

8.1 Avaluació personal

Aquest projecte ha estat molt beneficiós per ampliar els meus coneixements i posar en pràctica l'aprés durant el grau a la URV. Per una part, existeix la complexitat d'analitzar els requisits i dissenyar una BD coherent que s'acomodi a les dades necessàries. Per altra part, hi ha la dificultat d'haver fet un treball com a *fullstack developer* i, per tant, haver desenvolupat no només un *backend*, sinó dues aplicacions que utilitzen aquesta mateixa API.

Principalment, la dificultat ha recaigut en haver d'aprendre un *framework* completament nou per a mi, en aquest cas ReactJS i React Native, juntament amb CSS i JavaScript/TypeScript. Al principi, m'ha costat bastant entendre el cicle de vida d'un component de React i com interactuen les dades amb els elements visuals mitjançant els *hooks* d'estat. L'aplicació web ha hagut de ser re-desenvolupada des de zero per males pràctiques que s'utilitzaven al principi i s'han corregit, però, al final del projecte, desenvolupar l'aplicació mòbil amb els coneixements adquirits ha estat relativament simple.

Aquest projecte m'ajudarà professionalment per demostrar que no tinc gaire dificultat en aprendre i adaptar-me a un nou entorn de desenvolupament, així com per tenir unes nocions bàsiques de PHP i JavaScript, que, donat el mercat laboral per desenvolupadors de software, són coneixements bastant valuosos.

8.2 Coneixements aplicats al TFG

A continuació s'exposa una llista d'assignatures i els seus coneixements adquirits al GEI i aplicats al TFG:

- Anàlisi i Disseny d'Aplicacions: la primera etapa del desenvolupament és realitzar una anàlisi dels requisits i, sobretot, les classes necessàries per mantenir les dades de la forma més coherent possible.
- Bases de Dades: aquesta assignatura m'ha ajudat principalment a entendre el llenguatge SQL i com transformar correctament un diagrama de classes a un model relacional i identificar claus i taules intermèdies necessàries.
- Tècniques Avançades de Programació: aquesta ha estat una de les assignatures més útils pel que respecta a React. Els patrons de programació apresos, com ara els *singleton* o *observer*, es poden relacionar directament amb els estats dinàmics de React. Addicionalment, en aquesta assignatura es va aprendre exhaustivament l'ús d'expressions *lambda*, que són bastant utilitzades per tractar dades en JavaScript.
- Sistemes Distribuïts: tot i que aquesta aplicació funciona amb un únic servidor, i per tant no aplica escalabilitat, es va aprendre a utilitzar la comunicació entre client-servidor i com formatar dades rebudes d'un JSON.
- Aplicacions Mòbils i Encastades, Aplicacions i Serveis Mòbils: en aquestes dues assignatures es va aprendre a programar per a dispositius Android. Aquests coneixements s'apliquen directament al desenvolupament amb React Native, tot i no utilitzar Java o Kotlin, és útil saber el cicle de vida d'una aplicació mòbil, així com bones pràctiques de tractament de peticions a servidor i persistència de dades.

9 Recursos utilitzats

9.1 Programari

9.1.1 VSCode

Visual Studio Code és un IDE gratuït, lleuger, i molt modular que permet programar en qualsevol llenguatge, sempre que existeixi una extensió per a aquest.

En aquest projecte s'ha fet bastant ús de les extensions PHP de DEVSENSE i Simple React Snippets de Burke Holland. Aquestes extensions han permès fer el desenvolupament més fluït i simplificar certes accions repetitives, per exemple, declarar estats en React.

Figura 37. Logotip de VSCode

9.1.2 XAMPP

XAMPP és un software FOSS que integra totes les eines necessàries per desenvolupar un aplicació PHP y SQL (entre altres funcions) sense cap tipus de configuració especial. En aquest projecte ha estat molt valuós per realitzar la implementació de la BD i el *backend* en localhost.

Figura 38. Logotip de XAMPP

9.1.3 npm

npm és un gestor de paquets JavaScript/Node.js que consisteix en un conjunt d'eines de línia de comandes. El seu ús és necessari per preconfigurar un *boilerpalte*[8] de React (create-react-app), així com per poder iniciar l'entorn de desenvolupament i compilar el codi necessari.

Figura 39. Logotip de npm

8 En programació, un *boilerplate* es refereix a una secció de codi que es repeteix molt sovint amb molt poca o cap variació. Avui dia el concepte de *boilerplate* es refereix a una aplicació preconfigurada i llesta per ser engegada i modificada.

9.2 Llibreries

A continuació s'enumeren les llibreries o extensions utilitzades i on trobar-les.

[1] PHP: DEVSENSE
https://marketplace.visualstudio.com/items?itemName=DEVSENSE.phptools-vscode

[2] Simple React Snippets: Burke Holland
https://marketplace.visualstudio.com/items?itemName=burkeholland.simple-react-snippets

[3] create-react-app: Facebook
https://github.com/facebook/create-react-app

[4] Idearia: Logger
https://github.com/Idearia/php-logger

[5] axios
https://axios-http.com/

[6] react-big-calendar: Jason Quense (jsquense)
https://www.npmjs.com/package/react-big-calendar

[7] moment
https://momentjs.com/

[8] react-bootstrap
https://react-bootstrap.github.io/

[9] Bootstrap: Mark Otto, Jacob Thornton
https://getbootstrap.com/

[10] react-select: Jed Watson
https://www.npmjs.com/package/react-select

[11] React Native Async Storage
https://www.npmjs.com/package/@react-native-async-storage/async-storage

[12] datetimepicker – React Native Community
https://www.npmjs.com/package/@react-native-community/datetimepicker

[13] React Native Picker
https://www.npmjs.com/package/@react-native-picker/picker

[14] Expo
https://expo.dev/

9.3 Webgrafía

Documentació o articles consultats per desenvolupar l'aplicació.

[1] Using Axios for Network Requests in React: Sheharyar Naseer
https://www.npmjs.com/package/react-big-calendar

[2] Generar thumbnail en PHP: FearINC@gmail.com
https://www.php.net/manual/en/function.imagecopyresized.php#55692

[3] How to Build a Simple REST API in PHP: Sajal Soni
https://code.tutsplus.com/how-to-build-a-simple-rest-api-in-php--cms-37000t

[4] HTTP status response codes: mdn web docs
https://developer.mozilla.org/en-US/docs/Web/HTTP/Status

[5] React todo list: mdn web docs
https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/React_getting_started

[6] Expo Documentation
https://docs.expo.dev/

[7] Expo Router documentation and quickstar
https://docs.expo.dev/routing/installation

[8] Build an Expo app with EAS Build
https://docs.expo.dev/build/setup/

10 Annexes

10.1 Estudi de servidors *hosting* gratuïts

En ser una organització de voluntaris, LCDG necessita una solució de *hosting* gratuïta que compleixi els requisits necessaris. Essencialment, el *hosting* a escollir ha de poder allotjar contingut HTML, disposar de PHP i SQL, permetre executar jobs cron, i un límit d'espai de disc i transferència que no sigui molt limitat.

Avui dia, la majoria de *hostings* són molt limitats o requereixen una subscripció. Per tant, s'han trobat molt pocs servidors que compleixin aquests requisits.

HOSTING	VERSIÓ PHP	ESPAI DISPONIBLE	LIMITS TRANSFERÈNCIA	DISPONIBLE CRON
byte.host	5.3	1 GB	50 GB / mes	Sí
000webhost.com	7.4	300 MB	3 GB / mes	Limitat a 1
infinityfree.com	7.4	5 GB	Il·limitat	Sí

Taula 6. Comparativa de *hostings* gratuïts.

Clarament, la millor opció és escollir infinityfree ja que permet una transferència il·limitada de dades, en comparació amb 000webost. A més, ofereix més espai disponible, que és molt útil per a poder pujar documents i imatges.

Alternativament, una opció és utilitzar un equip de baix consum, com ara una Raspberry Pi o una màquina Linux no molt potent, i utilitzar-lo com a servidor. Els avantatges són tenir control total sobre el servidor, emmagatzematge il·limitat, possibilitat de fer fàcilment còpies de seguretat, entre altres. Els únics desavantatges són la possible dificultat per configurar-los i problemes amb el proveïdor d'internet (CgNAT [9], disponibilitat de xarxa, IPs públiques dinàmiques, ...)

10.2 Desplegament, posada en marxa i manteniment de l'aplicació

El desplegament i posada en marxa del projecte es detalla a continuació, separat en tres parts, una per cada part del projecte.

10.2.1 Backend

El primer pas necessari és crear la base de dades. Cal importar l'arxiu SQL amb l'estructura de la BD al servidor.

La base de dades creada no conté dades i, per tant, no existeixen administradors que puguin donar d'alta altres administradors o usuaris. El primer usuari s'ha de crear manualment. Aquest usuari, per motius de seguretat i correctesa de l'aplicació, és recomanable que s'esborri un cop s'ha creat un administrador mitjançant l'aplicació web.

[9] El CgNAT (Carrier-grade NAT) es un *workaround* que utilitzen els proveïdors d'internet per allargar la vida útil d'IPv4 i posa els clients darrere un router compartit, és a dir, un conjunt de clients es troben en una subxarxa privada abans d'accedir a l'"exterior'. Efectivament elimina que cada client tingui la seva pròpia IP pública i, per tant, no es puguin fer connexions de l'exterior a 'casa' mitjançant la IP pública.

Per aconseguir-ho cal inserir una entrada a la taula *usuaris* amb la següent *query* SQL, substituint els valors de nom d'usuari i contrasenya amb els valors que es vulguin utilitzar per fer login.

```
INSERT INTO usuaris (nom_usuari, nom_real, rol_usuari, password,
multiples_torns, preferencia_variable, first_login)
VALUES ([nom d'usuari], '', '1', [contrasenya encriptada], '0', '0', '0')
```

La contrasenya s'ha d'introduir encriptada i no en text pla. Per obtenir el *string* necessari es pot utilitzar qualsevol eina online per generar un *hash* que PHP entengui, i utilitzi l'algorisme definit per *PASSWORD_DEFAULT*. D'altra forma, es pot aconseguir el *hash* executant un fitxer .php amb el següent contingut:

```
<?php echo password_hash([contrasenya], PASSWORD_DEFAULT); ?>
```

L'últim pas que queda per al *backend* és pujar els arxius que formen l'API REST al servidor, mitjançant FTP o altres mecanismes, tenint en compte que les rutes siguin correctes.

10.2.2 Aplicació web

Per a que l'aplicació web sigui operativa es necessiten dos requeriments. Primer, la BD i el *backend* han de ser operatius, amb l'usuari temporal creat. Segon, s'ha d'haver creat una *build* de producció a partir del codi font. Per compilar el codi font es necessita que la màquina on es compili tingui instal·lat `npm`.

Compilar el codi és senzill: cal obrir una consola de comandes a l'arrel de la carpeta on es trobi el codi i executar la comanda `'npm run build'`. El codi compilat es guarda a una carpeta a l'arrel anomenada *build*. Per a tenir l'aplicació web operativa cal pujar el contingut de *build* a l'arrel del servidor.

Finalment, s'ha de poder iniciar sessió amb l'usuari temporal creat anteriorment.

10.2.3 Aplicació mòbil

Per a compilar l'aplicació mòbil ens aprofitem de l'opció de compilació gratuïta mitjançant els servidors d'Expo.

Els prerequisits necessaris són crear un compte a Expo i crear un projecte al compte creat per enllaçar el codi font. Un altre prerequisit és tenir instal·lat *EAS CLI* a la màquina que es vulgui utilitzar. Aquest pas es pot aconseguir mitjançant npm amb la comanda `'npm install -g eas-cli'`.

El primer pas és iniciar sessió a EAS i enllaçar el projecte amb el codi font. Per aconseguir-ho s'utilitza la comanda `'eas login'`, que demanarà les credencials del compte creat i, un cop iniciada la sessió, entrant la comanda `'eas build:configure'` a la carpeta root del projecte. Aquesta última comanda preguntarà si volem enllaçar el projecte creat anteriorment amb la carpeta actual: diem que sí.

Per poder fer la distribució interna mitjançant Expo, cal configurar dispositius iOS. S'utilitza la comanda `'eas device:create'` que demanarà un compte d'Apple Developer. Un cop iniciada la sessió es mostrarà un enllaç i un codi QR per enllaçar dispositius iPhone/iPad amb el projecte.

Desafortunadament no es disposa de dispositius iOS propis ni Mac. Per tant, aquest pas queda en mans de LCDG. Alternativament, en comptes d'utilitzar un compte d'Apple Developer es pot utilitzar *Xcode* per compilar els instal·lables (.ipa/.app), però això requereix un Mac.

Seguidament, cal afegir un perfil al fitxer *eas.json*. Sota l'entrada *build*, entre les entrades de *development* i *production* s'afegeix la següent entrada:

```
"preview": {
  "distribution": "internal",
  "android": {
    "buildType": "apk",
    "gradleCommand": ":app:assembleRelease"
  },
},
```

Finalment, cal executar la comanda `'eas build --profile preview --platform all'` per enviar el codi font del projecte als servidors d'Expo, que s'encarregaran de compilar i generar els instal·lables. Aquesta comanda preguntarà per certes opcions, com ara el nom identificatiu del paquet de l'aplicació o les credencials. Es poden utilitzar les opcions per defecte.

Un cop la *build* entra en cua per ser compilada només cal esperar a que acabi. En acabar, EAS CLI mostrarà instruccions de com instal·lar l'aplicació, així com una URL i un codi QR compartibles, on qualsevol usuari pot accedir.