

Gestió d’una falla

Projecte final de cicle formatiu



Joel fuster bosch

|  |  |
| --- | --- |
| **Nom i cognoms** | Joel Fuster Bosch |
| **NIF/NIE** | 020942567D |
| **Curs i CF** | 2n DAM |

|  |  |
| --- | --- |
| **Títol del projecte** | Gestió d’una falla |
| **Nom del tutor individual** | Juan Bautista Talens |
| **Nom del tutor del grup** | Jose Alfredo Murcia |
| **Resum** | *Gestió d’una falla que permet als usuaris fer compres amb les seues polseres de la falla amb tecnologia NFC (Near Field Communication)*. |
| **Abstract** | *Management of a Falla-making committee that provides the users doing shopping with their bracelets of the Falla-making committee with NFC technology (Near Field Communication).* |
| **Mòduls implicats** | * SGE (Sistemes de gestió empresarial) * PMDM (Programació multimèdia i dispositius mòbils) |
| **Data de presentació** | 4 de juny de 2025 |

Contents

[1.Introducció del projecte 4](#_Toc199772837)

[1.1Descripció del projecte 4](#_Toc199772838)

[1.2 Objectius del projecte: 4](#_Toc199772839)

[2.Tecnologies i ferramentes 4](#_Toc199772840)

[2.1 Tecnologies usades per al projecte 4](#_Toc199772841)

[2.1.1 Flutter 5](#_Toc199772842)

[2.1.2 Dart 6](#_Toc199772843)

[2.1.3 VSCode 7](#_Toc199772844)

[2.1.4 Github 8](#_Toc199772845)

[2.1.5 Figma 8](#_Toc199772846)

[2.1.6 Node.js 9](#_Toc199772847)

[2.1.7 PostgreSQL 9](#_Toc199772848)

[2.1.8 Odoo 10](#_Toc199772849)

[2.2 Tecnologia NFC 11](#_Toc199772850)

[3. Disseny de la solució 14](#_Toc199772851)

[3.1 Anàlisis de les possibles solucions 14](#_Toc199772852)

[3.1.1 Plataformes de software 14](#_Toc199772853)

[3.1.1.1 Solucions mòbil natives 14](#_Toc199772854)

[3.1.1.1.1 Avantatges 14](#_Toc199772855)

[3.1.1.1.2 Desavantatges 15](#_Toc199772856)

[3.1.1.1.3 Opinió personal 15](#_Toc199772857)

[3.1.1.2 Solucions multiplataforma 16](#_Toc199772858)

[3.1.1.2.1 Avantatges 16](#_Toc199772859)

[3.1.1.2.2 Desavantatges 16](#_Toc199772860)

[3.1.2 Persistència de dades 18](#_Toc199772861)

[3.1.2.1 Odoo 18](#_Toc199772862)

[3.1.2.1.1 Avantatges 18](#_Toc199772863)

[3.1.2.1.2 Desavantatges 18](#_Toc199772864)

[3.1.2.2 Servidor de Node.js + BD (PostgreSQL) 19](#_Toc199772865)

[3.1.2.2.1 Avantatges 19](#_Toc199772866)

[3.1.2.2.2 Desavantatges 19](#_Toc199772867)

[3.2 Descripció de la solució escollida 20](#_Toc199772868)

[3.3 Prototips 20](#_Toc199772869)

[3.3.1 Prototip de baixa fidelitat 20](#_Toc199772870)

[3.3.2 Prototip d’alta fidelitat 22](#_Toc199772871)

[4. Desenvolupament de la solució 24](#_Toc199772872)

[4.1 Consideracions prèvies 24](#_Toc199772873)

[4.2 Configuració i desenvolupament del sistema 25](#_Toc199772874)

[4.2.1 Configuració de Flutter i el seu entorn de desenvolupament 25](#_Toc199772875)

[4.2.1.1 Instal·lació de Flutter i les seues dependències 25](#_Toc199772876)

[4.2.1.2 Instal·lar l’entorn de desenvolupament 25](#_Toc199772877)

[4.2.2 Configuració del backend (Servidor de node.js) 26](#_Toc199772878)

[4.2.3 Configuració del backend (Base de dades) 28](#_Toc199772879)

[4.2.3 Arquitectura CLEAN 28](#_Toc199772880)

[4.2.3.1 Implementació de l’arquitectura CLEAN 31](#_Toc199772881)

[4.2.3.1.1 Estructura de directoris 31](#_Toc199772882)

[4.2.3.1.2 Capa de Infraestructura 32](#_Toc199772883)

[4.2.3.1.2.1 DataSource 32](#_Toc199772884)

[4.2.3.1.2.2 Implementació del repositori 33](#_Toc199772885)

[4.2.3.1.3 Capa de domini 34](#_Toc199772886)

[4.2.3.1.3.1 Entitats 35](#_Toc199772887)

[4.2.3.1.3.2 Repositori 35](#_Toc199772888)

[4.2.3.1.4 Capa de Presentació 36](#_Toc199772889)

[4.2.3.1.4.1 Pantalles 36](#_Toc199772890)

[4.2.3.1.5 Capa de gestor d’estats 36](#_Toc199772891)

[4.2.3.1.5.1 Provider 36](#_Toc199772892)

[4.2.3.4 Vista per rols 41](#_Toc199772893)

[5.Dificultats i millores futures 43](#_Toc199772894)

[5.1 Dificultats 43](#_Toc199772895)

[5.2 Millores futures 44](#_Toc199772896)

[5.3 Que he aprés 44](#_Toc199772897)

[6.Annexos 44](#_Toc199772898)

[6.1 Github 44](#_Toc199772899)

[6.2 Finalització del projecte 45](#_Toc199772900)

[Bibliografia 46](#_Toc199772901)

# 1.Introducció del projecte

## 1.1Descripció del projecte

Este projecte té com a objectiu unir als fallers en grups anomenats famílies, els quals es poden unir altres fallers per a una gestió més còmoda de la realitat a l’hora de poder unir-se a esdeveniments, comprar menjar/begudes en la parada de la falla..., etc. Tot en una sola aplicació perquè puguen dur a terme els esdeveniments de la falla de forma més còmoda.

## 1.2 Objectius del projecte:

Gestionar tots els esdeveniments, productes, tiquets..., etc. Gràcies a les polseres que funcionen amb la tecnologia NFC (Near-Field-Communication) i alternativament amb ús de QR (Quick-Response-Code) per als dispositius que no compten amb la tecnologia NFC en una sola aplicació.

# 2.Tecnologies i ferramentes

## 2.1 Tecnologies usades per al projecte

Primerament, vaig a fer un resum de les especificacions dels meus dispositius que han sigut involucrats en este projecte:

El projecte ha sigut desenvolupat en el meu ordinador personal, un portàtil amb les següents especificacions:

|  |  |
| --- | --- |
| Processador | Intel Core I5-11400H @ 2.70GHz, 2668MHz, 6 processadors principals, 12 processadors lògics |
| Sistema operatiu | Microsoft Windows 11 Pro |
| Memòria RAM | 16GB |

I ha sigut provat en un dispositiu mòbil Android de les següents especificacions:

|  |  |
| --- | --- |
| Versió de Android | Android 14  (o també coneguda com “**Upside Down Cake**”) |
| Processador | MediaTek Dimensity 700 8 nuclis |
| Espai d'emmagatzemament | 128GB |
| Memòria RAM | 4GB |

Ara esmentaré les eines que he usat perquè este projecte haja sigut possible.

### 2.1.1 Flutter



Framework multiplataforma desenvolupat per Google, tenia com a propòsit l’ús de software de la companyia, però amb el temps s’ha transformat en una plataforma de codi obert per al desenvolupament multiplataforma.

Permet el desenvolupament en les plataformes de:  
-Android  
-iOS  
-Windows  
-Linux  
-macOS  
-Web  
En este projecte s’ha desenvolupat únicament per a plataformes Android, pel fet que no dispose d’un sistema iOS per a fer proves i volia centrar-me en l'ecosistema d'Android que és el sistema operatiu mòbil més usat en Espanya un 78,8% en 2023 segons CNMC (Comisión Nacional de los Mercados y la Competencia), teniu un enllaç en la bibliografia de l'article de CNMC.

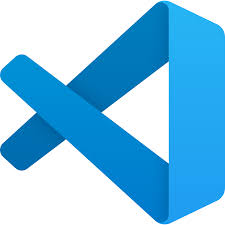
### 2.1.2 Dart



Dart és un llenguatge de programació de codi obert desenvolupat per Google, va ser revelat en la conferència “goto” el 10 d’octubre de 2011. L’objectiu de Dart és oferir una alternativa més moderna a JavaScript per al desenvolupament web, es pot veure reflectit en les declaracions de Lars Bak, enginyer de software en Google que definix a Dart com un “llenguatge estructurat però flexible per a programació web”.

He utilitzat Dart pel fet que el framework de Flutter usa el llenguatge de programació de Dart m’he vist en l’obligació de programar en ell.

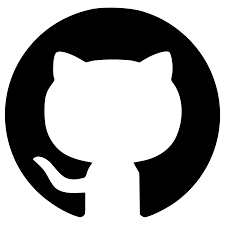
### 2.1.3 VSCode



Visual Studio Code llançat el 18 de novembre de 2015 és un editor de text pla desenvolupat per Microsoft de codi obert basat en Electron, un framework que s’utilitza per a implementar Chronium i NodeJS com a aplicacions d’escriptori, que s’executa en el motor de disseny Blink.

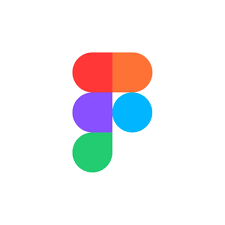
He usat Visual Studio Code com el meu IDE perquè és l'IDE que més conec i més gaste actualment, però has de tindre Android Studio instal·lat per les dependències d'Android que proveïx este últim junt amb les extensions de Flutter, i este últim té per defecte les llibreries de Git, així que m’ha ajudat molt per al control de versions Github.

### 2.1.4 Github



Github, publicat en 2008 i desenvolupat amb Ruby on Rails és una forja (plataforma de desenvolupament col·laboratiu) per a allotjar projectes utilitzant el sistema de control de versions de Git. S’usa principalment per a la creació de codi font de programes d’ordinador.  
He fet servir GitHub com a repositori públic, perquè qualsevol persona puga veure el codi font del projecte, descarregar-lo o clonar-lo. Està tot organitzat per carpetes i podeu trobar tant el frontend com la resta del projecte. No he treballat amb branques ni commits complexos, però l’he fet servir com a lloc de referència i accés al codi.

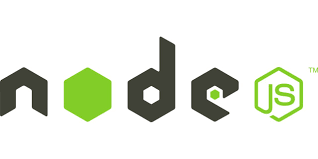
### 2.1.5 Figma



Figma, desenvolupat per Dylan Field i Evan Wallace començaren a treballar en Figma en 2012 i el van llançar el 27 de setembre de 2016 és un editor de gràfics vectorial i una ferramenta de generació de prototips, principalment basada en la web, amb característiques “off-line” addicionals habilitades per aplicacions d’escriptori en macOS i Windows.

He utilitzat Figma únicament per a fer el prototip d’alta fidelitat del frontend. Em va servir per a planificar el disseny visual abans de començar a programar l’aplicació, i assegurar-me que l’estructura i els estils quedaven clars.

### 2.1.6 Node.js



Node.js, publicat el 27 de maig de 2009 és un entorn de desenvolupament de codi obert per a la capa de servidor basat en el llenguatge de JavaScript, asíncron amb E/S (perifèric d’entrada/eixida) amb dades en una arquitectura basada en esdeveniments i basat en el motor V8 de Google.  
He usat node.js com a backend i com a API, en Flutter faig peticions en el servidor de Node, que esta es connectarà a la base de dades i torna la petició amb les dades que es mostraran en pantalla, però originalment anava a usar Odoo, després ho explicaré.

### 2.1.7 PostgreSQL



PostgreSQL també conegut com a Postgres, és un sistema de gestió de bases de dades relacional orientat a objectes i de codi obert publicat en 1996 baix la llicencia PostgreSQL similar a la BSD o MIT.

He usat PostgreSQL com a base de dades que està connectada al servidor amb NodeJS, esta guarda tota la informació sobre usuaris, famílies, productes..., etc. El servidor de Node.js arreplegara esta informació i la passara a l’aplicació en Flutter.

### 2.1.8 Odoo



Odoo, abans conegut com a OpenERP i més enrere conegut com a TinyERP, va ser publicat el 4 de juliol de 2004 és un software d'ERP integrat. Compta amb una versió “Community” de codi obert baix la llicencia LGPLv3 i una versió empresarial baix llicencia comercial que complementa la versió comunitària amb característiques i serveis comercials i desenvolupada per la companyia belga Odoo S.A.

És un software empresarial tot en un que inclou CRM, lloc web i comerç electrònic, facturació, comptabilitat, fabricació, gestió de magatzems i projectes i inventari..., etc.

Originalment, el backend del meu projecte abans que fera el servidor de Node, és paregut el que fa el servidor de Node i la base de dades PostgreSQL, però amb una sola plataforma, va ser substituïda pel meu tutor per a implementar el servidor de Node, el qual està en eta presentació, però per falta de temps no va ser possible implementar les funcions de Node a Odoo.

## 2.2 Tecnologia NFC

NFC o per les seues sigles (Near Field Communication) funciona a través d’ones electromagnètiques que aconseguiran comunicar un dispositiu a un altre fins a una distància de 10 a 15 cm a una velocitat de 424Kb/s, ara esmentaré alguns exemples.

Exemples de NFC:

|  |  |
| --- | --- |
| Targetes (tant de crèdit com d’accés) |  |
| Polseres  (esta serà en la que treballarem): |  |
| Alguns rellotges intel·ligents |  |
| Etiquetes |  |
| Algunes figuretes col·leccionables | (per exemple els Amiibo de Nintendo per als sistemes de Nintendo 3DS, Nintendo Switch i possiblement també en Nintendo Switch 2 en data d’esta memòria) |

Els dispositius que compten amb la tecnologia NFC tenen 2 formes per a comunicar-se:

|  |  |
| --- | --- |
| Passiva | Sols un dels dispositius està actiu i l’altre intercanviara les dades. |
| Activa | Els 2 dispositius generen un camp electromagnètic i intercanvien dades. |

2.3 Codi QR

També conegut com (Quick-Response-Code), va ser inventat en 1994 per l'enginyer japonés Masahiro Hara, va ser desenvolupat per l'empresa Denso Corporation la qual és una subsidiària de Toyota, el codi QR és una millora al codi de barres tradicional, pel fet que és un mètode per a emmagatzemar informació mitjançant contractes de color de forma bidimensional (2D), tenint com a objectiu una lectura ràpida del contingut del codi, per això el seu nom de Quick-Response-Code, es pot usar de diverses formes, com per exemple:

|  |  |
| --- | --- |
| Productes en general |  |
| Cartes de restaurant |  |
| Correus electrònics |  |
| Anuncis |  |
| Accés ràpid a documents governamentals |  |
| Alguns videojocs (per a aconseguir algunes recompenses o per a dur-te a la pàgina oficial del videojoc): |  |

# 3. Disseny de la solució

## 3.1 Anàlisis de les possibles solucions

### 3.1.1 Plataformes de software

#### 3.1.1.1 Solucions mòbil natives

En temes de compatibilitat i potència a major escala desenvolupar de forma nativa tant per a Android com iOS, tenen una base sòlida de documentació per a ajudar al programador a desenvolupar en estes eines, però clar té els seus avantatges i desavantatges.

##### 3.1.1.1.1 Avantatges

✔️ Accés directe al hardware del sistema: Ens dona la facilitat d’accedir al hardware del dispositiu de forma nativa.

✔️ Compatibilitat: Com he dit abans la compatibilitat i la potència es desenvolupen al seu màxim potencial.

✔️ Espai: Les aplicacions natives ocupen molt poc d’espai perquè aprofiten al màxim les característiques del sistema on es desenvolupa.

##### 3.1.1.1.2 Desavantatges

❌ Plataformes diverses: Per a desenvolupar en diverses plataformes hauries d'aprendre diversos llenguatges de programació i tardaries més amb la documentació aprendre que fa cada cosa per fer la mateixa aplicació de per si.

❌ Corba d’aprenentatge elevada: Aprendre a desenvolupar de forma nativa un llenguatge de programació és molt tardat, i una empresa menuda o que té poc capital no podria permetre's a desenvolupar software de forma nativa a diverses plataformes.

##### 3.1.1.1.3 Opinió personal

En el mòdul de PMDM vam donar ferramentes per a desenvolupar de forma nativa en Android com per exemple Android Studio, encara que molt potent a l’hora de fer aplicacions de forma nativa vam deixar d’utilitzar Android Studio molt ràpid per a començar a desenvolupar en Flutter, i per a posar més pes no vam donar alguna ferramenta per a desenvolupar de forma nativa en iOS, tampoc sé si existixen ferramentes per a desenvolupar de forma nativa en iOS en Windows o en Linux, en macOS segur que si existix, però no anàvem a perdre temps en alguna eina per a desenvolupar en iOS també quan podíem desenvolupar de forma multiplataforma en Flutter i ens obrien les portes a Windows, Linux, macOS..., etc. Fent que expandir l’aplicació a noves experiències siga tardat, costós i difícil d’implementar.



3.1.1.2 Solucions multiplataforma

Encara que no oferixen ni la compatibilitat ni la potència de multiplataforma com Flutter permet desenvolupar per a molts sistemes operatius de forma senzilla i ràpida, però clar, no tot pot ser de color rosa...:

##### 3.1.1.2.1 Avantatges

✔️ Multiplataforma: Com bé diu el seu nom les ferramentes multiplataforma ens permetran programar una sola vegada i tindre accés a altres sistemes operatius amb una inversió de temps molt baixa.  
✔️ Compilació de codi natiu: Generalment, oferixen compilació de forma nativa, encara que arrosseguen certes dependències que estan al nivell del rendiment d’una aplicació nativa.  
✔️ Fàcil aprenentatge: Estes plataformes són fàcils d’aprendre, solen fer ús de llenguatges de programació populars o reconeguts com Flutter per exemple i tenen una àmplia documentació, fent que els costos tant de desenvolupament, documentació i producció es reduïsquen de forma abismal.

##### 3.1.1.2.2 Desavantatges

❌ Dependències: Encara que solen compilar amb codi natiu arrosseguen algunes dependències pròpies del SDK junt amb altres dependències per a l’accés al hardware perquè les ferramentes multiplataforma no poden tindre accés al hardware sense l'ajuda de dependències.

❌ Accés al hardware: Per a accedir a funcions específiques al hardware so ser necessari introduir programació nativa del dispositiu que estem treballant o dependències de tercers que no funcionen en tots els sistemes, la qual ens obliga a fer ús d’altres tecnologies de cada plataforma i augmenta el pes de l’aplicació i la possibilitat d’error en les nostres aplicacions.  
❌ Proves: Desenvolupar per a diferents plataformes amb diferents mides i resolucions, interfícies i funcionament obliga a dur a terme una gran quantitat de proves perquè funcione en cada dispositiu, per conseqüència augmentant els costos de desenvolupament i manteniment del software.  
3.1.1.2.3 Opinió personal

Esta és l'opció que més m’agrada i recomane si vols fer aplicacions tant per a diferents dispositius tant com si sols és per a un sol dispositiu, perquè al final va ser l’opció que més vam treballar en el mòdul de PMDM i resulta molt còmode fer aplicacions en una eina multiplataforma, sols és centrar-se en un sol llenguatge de programació per a tindre una gran possibilitat de desenvolupar aplicacions per a molts dispositius a la vegada si t’animes a fer-ho i no sols quedar-te en una sola plataforma, encara que he parlat de forma més generalitzada Flutter és un molt bon framework que permet el desenvolupament d’aplicacions multiplataforma i és i serà una bona ferramenta per a desenvolupar aplicacions amb facilitat i un gran nivell d’especialització a pesar d’apuntar a diversos dominis.

### 3.1.2 Persistència de dades

L’aplicació depèn de persistència de dades, per a guarda informació sobre fallers, productes, famílies, esdeveniments..., etc. Junt amb l’administració d’estos últims, ara mencionaré les eines que he utilitzat i el perquè.

#### 3.1.2.1 Odoo

Odoo com havia mencionat en la seua explicació és una plataforma on podem administrar molt de la lògica de negoci de l’aplicació, però com tot en esta vida no és perfecte:

##### 3.1.2.1.1 Avantatges

✔️ Fàcil d’usar: Odoo dirigix molt bé tot el tema d’usuaris, codis de barres, rols..., etc. Fent una experiència agradable al gestionar una falla.  
✔️ Creació de mòduls personalitzats: Pots crear mòduls personalitzats per a personalitzar la gestió de l'Odoo i afegir noves funcionalitats.  
✔️ Diferents versions: Odoo té 2 versions la “community” i la “enterprise”, la “community” es pot usar de forma gratuïta i pot ser usada per xicotetes empreses i la “enterprise” cal pagar, però és molt còmoda a l’hora de gestionar usuaris.

##### 3.1.2.1.2 Desavantatges

❌ Difícil instal·lació per a persones amb poca experiència en la informàtica (en la part de servidor): N’hi han poques alternatives per a instal·lar Odoo perquè funcione, pots o instal·lar-lo en un servidor que és un procés car i tardat o pots obtindre'l a partir d’un contenidor de Docker i alguns fitxers de configuració perquè funcione correctament, cosa que per a principiants o persones que tenen o poca o nul·la experiència en la informàtica és una feina molt complicada.

❌ Difícil implementació en l'app de Flutter: No sols has de gastar la llibreria de http, sinó que també has de gastar xmlrpc o jsonrpc per a connectar-te al servidor d'Odoo, sinó que has de fer JSON per a fer funcions.

#### 3.1.2.2 Servidor de Node.js + BD (PostgreSQL)

De les 2 l'opció més còmoda també té certs desavantatges que podrien tirar cap enrere a algú:

##### 3.1.2.2.1 Avantatges

✔️ Personalitzable: Desenvolupant el backend amb una APIREST amb node.js amb la llibreria d’express i la base de dades de PostgreSQL, és de lluny l'opció més personalitzable possible: Podem crear les rutes necessàries per a donar servei a les nostres necessitats, tot dins d’una estructura de base de dades en PostgreSQL de la grandària justa i necessària.  
✔️ Escalable: Al complir amb les nostres necessitats específiques el software no haurà de fer res més del necessari, reservant tota la potència del sistema a les feines concretes que necessitem.

##### 3.1.2.2.2 Desavantatges

❌ Car: Els servidors dedicats a node no són molt econòmics, especialment si l’aplicació és molt gran.

❌ Manteniment elevat: Si fem un servidor dedicat hem d'estar atents a no fer errors mentre el desenvolupem, perquè si passa som nosaltres qui haurem d'arreglar els errors que es produïxen i això pot consumir molt de temps del desenvolupament.

❌Temps de desenvolupament: Este és l’inconvenient més important, desenvolupar un backend sol tindre un cost de desenvolupament molt elevat.

## 3.2 Descripció de la solució escollida

Amb tota l’explicació feta donaré el meu veredicte sobre les eines utilitzades en este projecte les quals són en el frontend Flutter i en el backend el servidor de node+Postgres, perquè els he escollit?  
En Flutter tinc accés al desenvolupament multiplataforma i si més endavant vull expandir-me a altres sistemes haver de tocar molt poc del codi original, junt amb què és el llenguatge que més experiència tinc a l’hora d’aplicacions multiplataforma.  
El servidor de node encara que siga una opció que consumisca molt de temps és una opció escalable i personalitzable que en sòls definir les entitats en un no he de preocupar-me a l’hora d'inserir-ho en l’altre, perquè sé que si falla no és per les entitats, sinó per altra cosa, i no he de preocupar-me res en Odoo.

## 3.3 Prototips

Ací mostraré els prototips que vaig fer en diferents parts del desenvolupament l’aplicació:

### 3.3.1 Prototip de baixa fidelitat

El prototip de baixa fidelitat el vaig fer al principi del projecte quan a penes tenia molta idea del que havia de fer, però em va servir de primera guia de què volia fer:

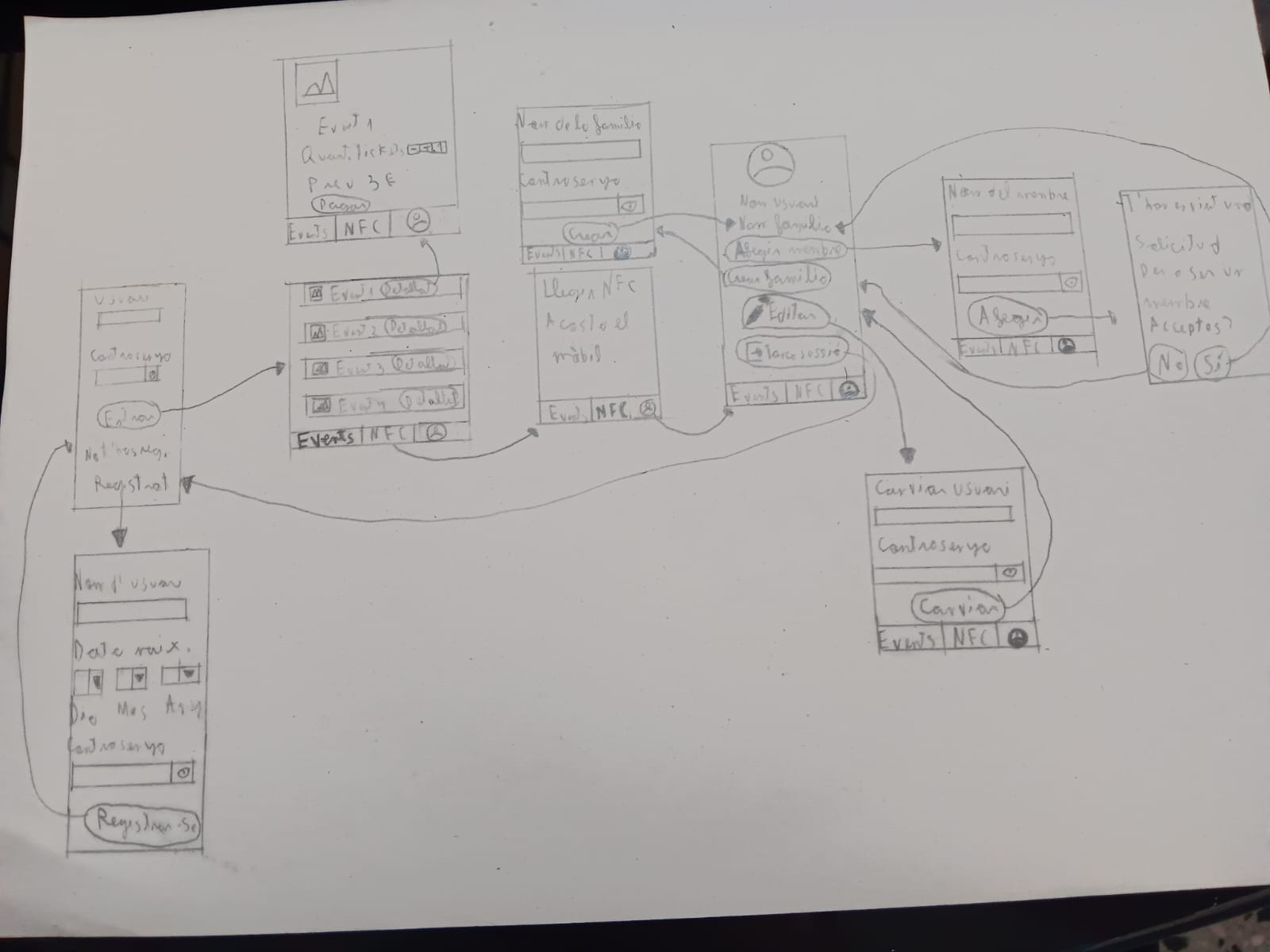


Fig.1 Prototip de baixa fidelitat fet a mà

### 3.3.2 Prototip d’alta fidelitat

Aquest prototip el vaig fer més endavant en el desenvolupament ja tenint una idea del que volia fer junt amb la comanda de fer una pantalla de lector QR:

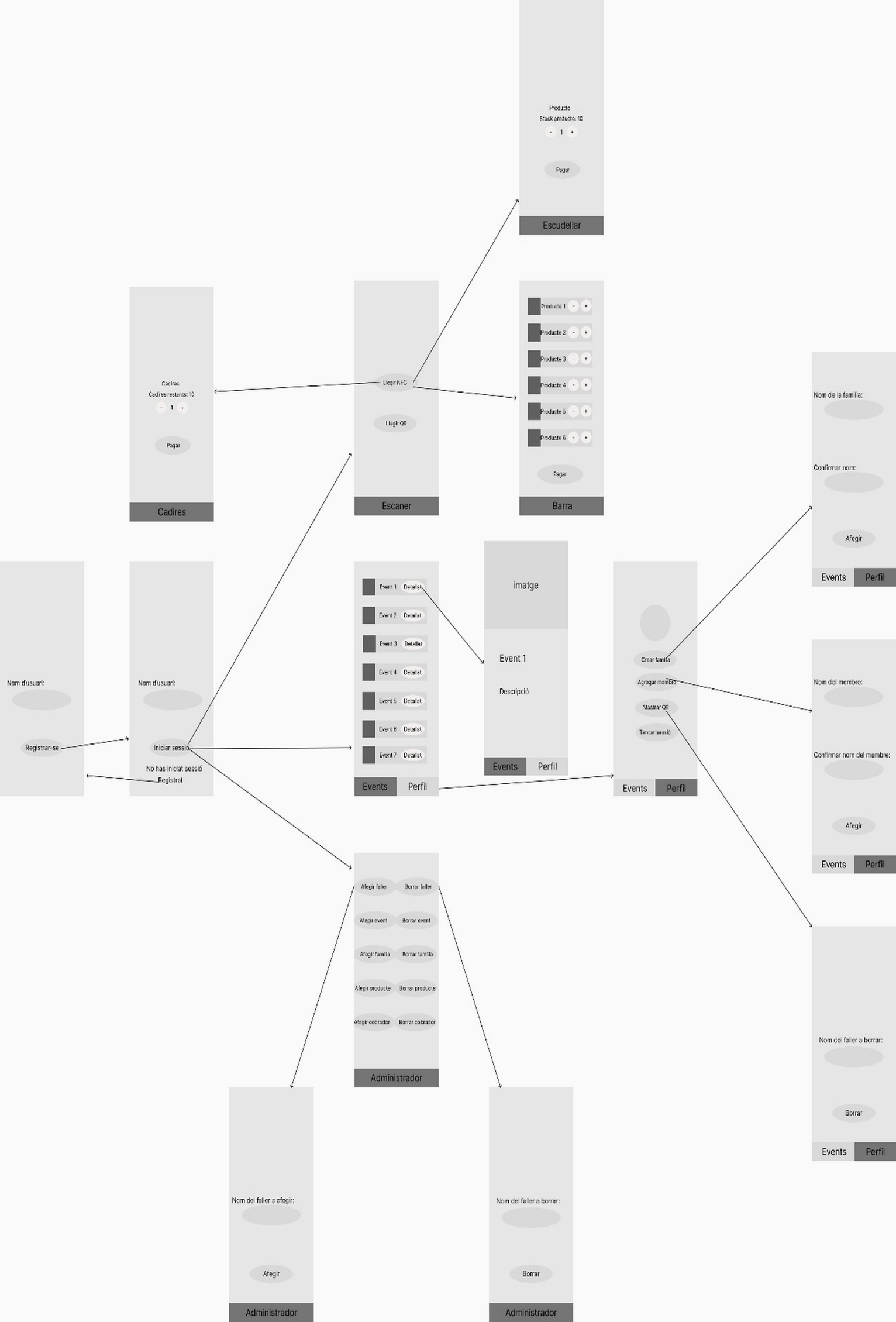


Fig.2 Prototip d’alta fidelitat fet en Figma

# 4. Desenvolupament de la solució

## 4.1 Consideracions prèvies

Abans de començar el desenvolupament del projecte, vaig tindre en compte diversos aspectes importants per tal de planificar bé el treball i evitar problemes posteriors:

-Limitacions tècniques: Com que només disposava d’un dispositiu Android i no de cap dispositiu iOS, vaig decidir centrar el desenvolupament únicament en Android per qüestió de compatibilitat i proves reals.

-Tecnologia seleccionada: Per experiència prèvia en el mòdul de PMDM, vaig escollir Flutter com a framework de desenvolupament, ja que em permetia desenvolupar de forma ràpida, modular i amb suport per a dispositius mòbils.

-Elecció del backend: Inicialment, estava previst usar Odoo, però per flexibilitat i control total del codi vaig optar per desenvolupar un backend propi amb Node.js i PostgreSQL.

-Organització del projecte: Per tal de seguir una estructura clara i escalable, vaig decidir aplicar l’arquitectura CLEAN, separant clarament la lògica de negoci, la interfície d’usuari i la infraestructura.

-Ús de prototips previs: Abans d’escriure codi, vaig dissenyar dos prototips (de baixa i alta fidelitat) per visualitzar l’estructura de l’app i validar la distribució de pantalles segons els rols.

-Divisió per rols: Un punt clau del projecte era la vista personalitzada per a cada tipus d’usuari (faller, cobrador, administrador), així que vaig estructurar l’aplicació pensant en estos accessos des del principi.

## 4.2 Configuració i desenvolupament del sistema

En aquesta part anem a configurar tota la aplicació per a que funcione en qualsevol dispositiu tant per la part de frontend (Flutter) tant com per la part de backend (Node.js)

### 4.2.1 Configuració de Flutter i el seu entorn de desenvolupament

En aquesta part desenvoluparem com configurar l’app de Flutter, examinant des de les dependències fins a l’entorn de desenvolupament.

#### 4.2.1.1 Instal·lació de Flutter i les seues dependències

Tenim a la nostra disposició una documentació completa de com dur a terme la <https://docs.flutter.dev/get-started/install>, el procediment per a instal·lar en cada sistema operatiu és diferent, i com s’ha comentat anteriorment implica la instal·lació de determinades dependències, una vegada ho tenim ja instal·lat executem en el terminal “flutter doctor” si posa “OK” ja tenim tot preparat per a programar en flutter.

#### 4.2.1.2 Instal·lar l’entorn de desenvolupament

Si volem desenvolupar per a Android és indispensable tindre una versió recent d'Android Studio, preferiblement les últimes versions pel que és més comú tindre un IDE configurat simultàniament. Així ha sigut amb l’IDE que he usat jo Visual Studio Code de Microsoft. El motiu principal és la facilitat d’escriure codi i la bona d’algunes característiques de Flutter amb els plugins desenvolupats per este IDE. Encara que per descomptat és necessari la instal·lació de certs paquets de Flutter perquè funcione correctament al Visual Studio Code més alguns que m’han facilitat la vida els quals els vam instal·lar en el mòdul de PMDM:

Noms de les extensions:

· Flutter, creat per: Dart Code [dartcode.org](https://dartcode.org/)

· Dart, creat per: Dart Code [dartcode.org](https://dartcode.org/)

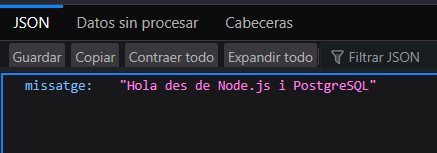
· Awesome Flutter Snippets, creat per: Neevash Ramdial (Nash)

### 4.2.2 Configuració del backend (Servidor de node.js)

El que has de fer perquè el servidor de node.js i la base de dades funcionen perquè l’aplicació reba totes les dades és obrir la carpeta de fakeApiOdoo que està en el projecte de GitHub i en un terminal poses “node app.js” i l’aplicació estarà ja funcionant perquè en el terminal posarà el següent:

Servidor escoltant en [http://0.0.0.0:3000/](http://localhost:3000/)

  
IMPORTANT:  
Com el servidor de node has d'arrancar-lo tu mateix això significa que depèn d’on estigues i del dispositiu el servidor tindrà una IP o una altra, com en el meu cas el servidor està en una màquina virtual en NAT per a connectar-te des del mateix dispositiu serà sempre la mateixa “127.0.0.1:3000” o “localhost:3000”, però per a l’app de flutter és una altra història, hauràs de posar la ip del teu dispositiu real “192.168.x.x:3000” perquè l’app de flutter es connecte al servidor de Node.js.

Ara ignorant el problema si aneu al vostre navegador i poseu localhost:3000 o “127.0.0.1:3000” podeu veure el següent:

Això significa que el servidor ja està funcionant i esperant a rebre peticions i la base de dades s’executa amb el servidor de node.js, açò seria un exemple en la pantalla de faller:

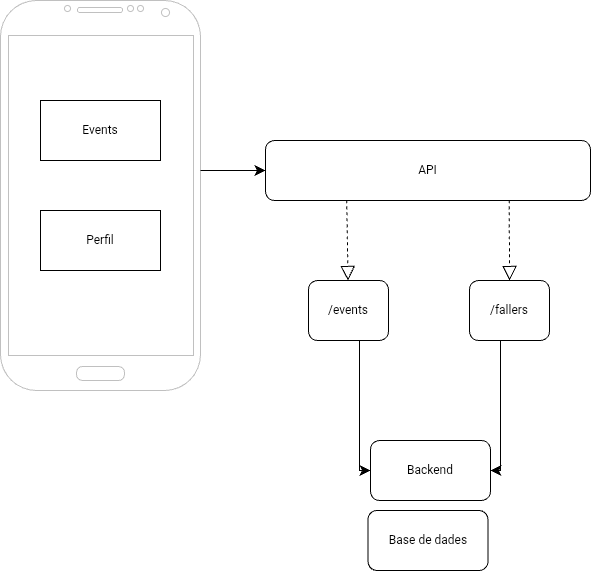


Fig.3 Diagrama sobre el recorregut de l’aplicació de flutter fins al Backend en la base de dades de PostgreSQL

### 4.2.3 Configuració del backend (Base de dades)

No fa falta configurar res per a que funcione si tens la base de dades, sols posant en marxa el servidor de node.js la base de dades deuria de funcionar, sinó en executar en el terminal de PostgreSQL l’script de AfegirScript.sql és més que suficient per a fer-lo funcionar, ací teniu un esquema de la base de dades en un esquema Entitat-Relació (E/R):

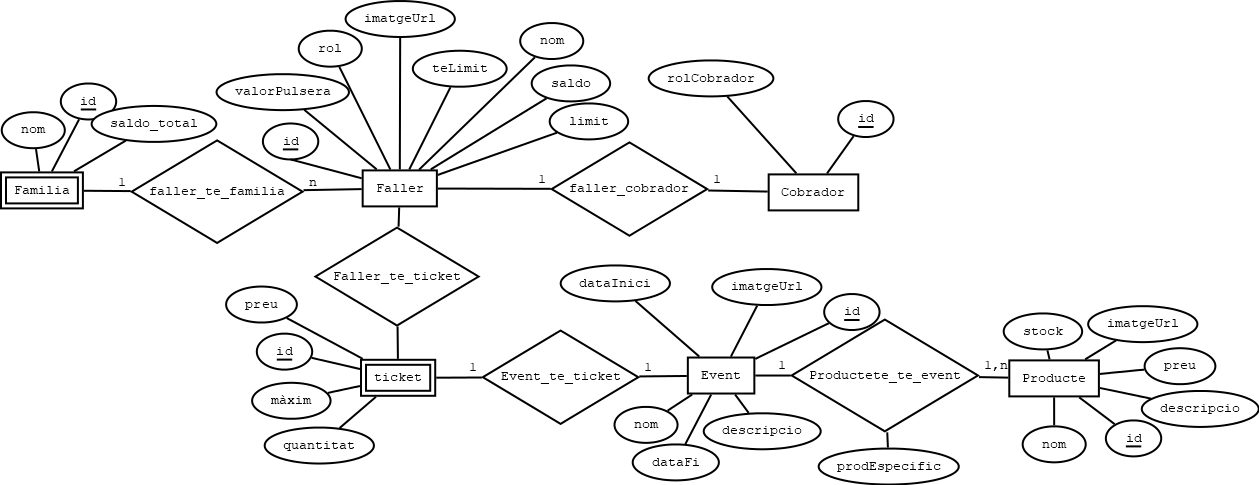
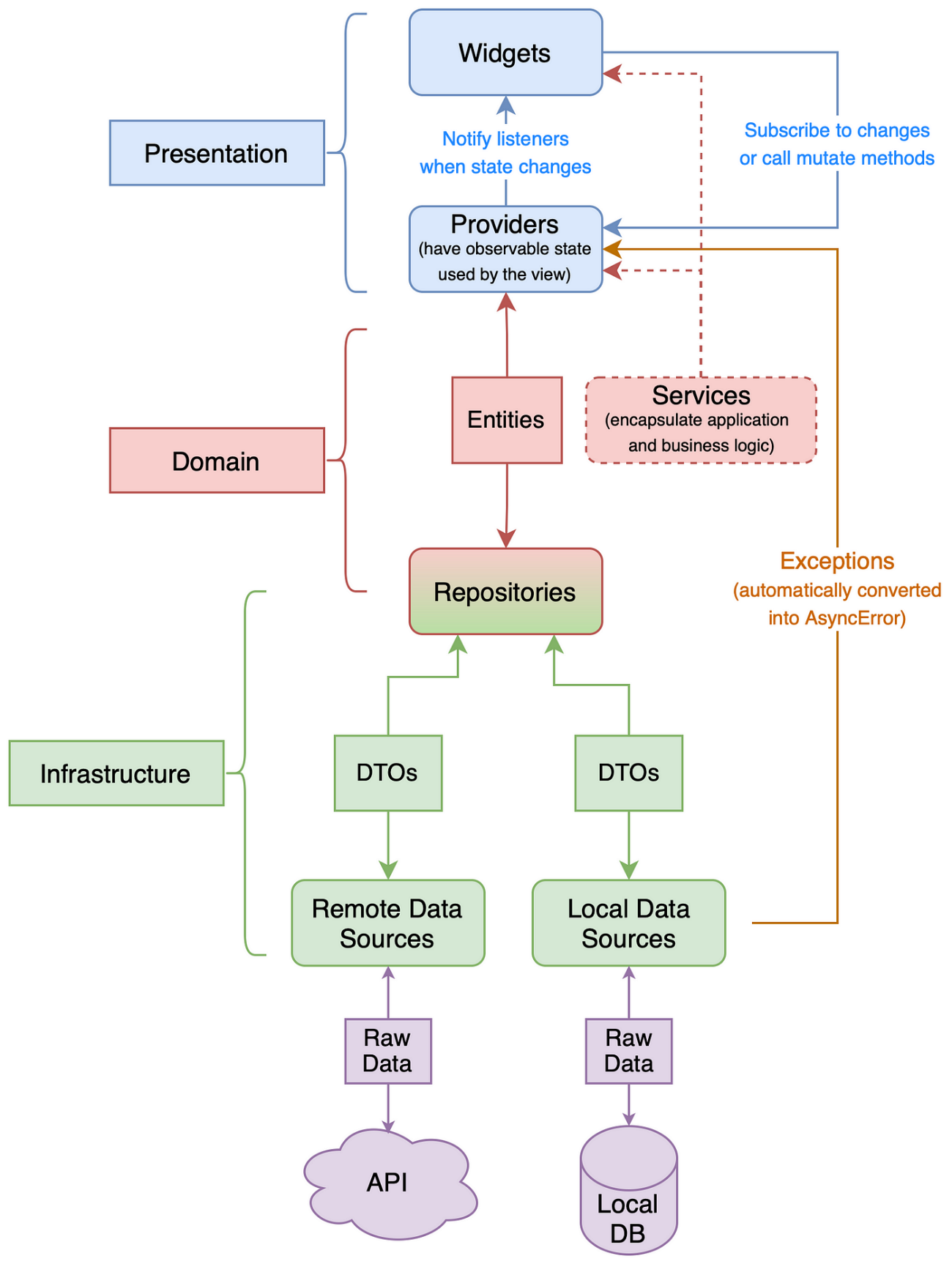


Fig.4 Diagrama Entitat/Relació (E/R) de la base de dades

### 4.2.3 Arquitectura CLEAN

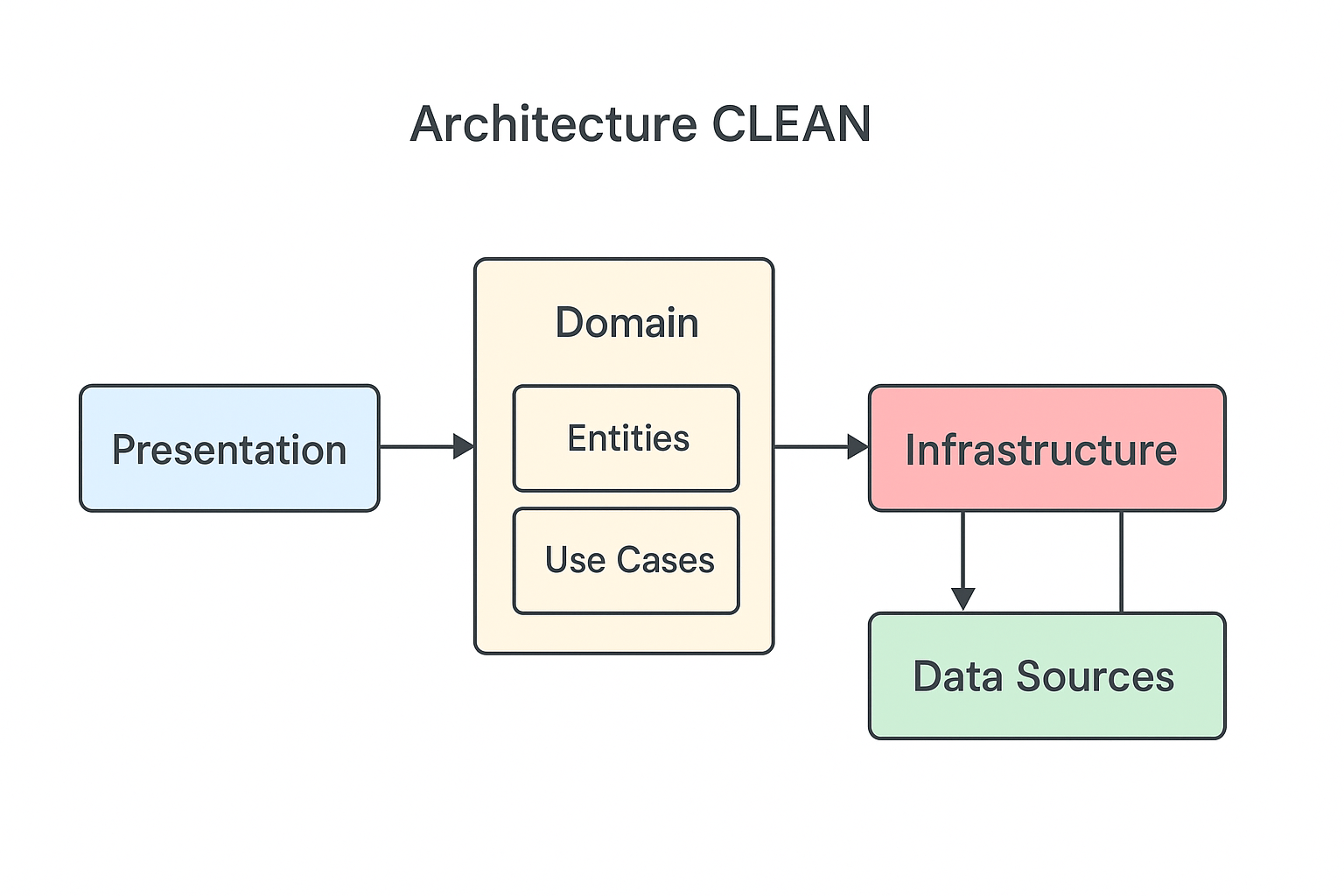
L’arquitectura CLEAN proposada per Robert C. Martin (Uncle Bob) és un acrònim que prové de “Clean Architecture” és una forma per a organitzar el codi de forma que separe clarament les responsabilitats, millore l'escalabilitat, sostenibilitat i “testabilitat” en una aplicació, encara que l’arquitectura CLEAN no és exclusiva de Flutter es pot aplicar perfectament en projectes de Flutter/Dart per a estructurar de forma correcta una app, l’arquitectura CLEAN es pot resumir en la següent imatge:



Explicació:

|  |  |
| --- | --- |
| Presentation | 👉**UI:** Interfície de l’aplicació (el que es mostrara en l'app). |
| Domain | 👉**Repository(implementació):** Implementa les funcions del DataSource, hereta del Repository (sense implementació) i l’usa la interfície (UI).  👉**Entities:** Els objectes de l’aplicació que rebran els canvis. |
| Infrastructura | 👉**Repository:** Capa intermèdia que connecta i implementa el DataSource i heretara a la implementació del Repositori en la capa de “Domain”.  👉**DataSource:** Capa que fa peticions a APIs o a altres llocs per a arreplegar la informació i després mostrar-la en la interficie (UI). |
| Gestor d’estats | 👉**Provider:** Gestor d’estat per a saber l’estat de les peticions. |

Ací teniu una imatge per a explicar-ho millor:

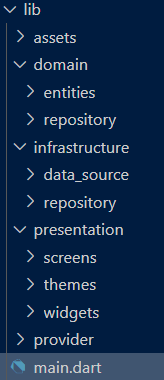


Ara ja *explicada* la teoria, anem a la pràctica.

#### 4.2.3.1 Implementació de l’arquitectura CLEAN

##### 4.2.3.1.1 Estructura de directoris

En el vostre projecte de Flutter per a seguir amb l’arquitectura CLEAN heu de seguir la següent cadena de directoris:



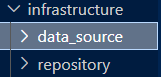
##### 4.2.3.1.2 Capa de Infraestructura

Primerament creem la següent cadena de carpetes si no les teniu creades:



###### 4.2.3.1.2.1 DataSource

Anem a la carpeta de “data\_source” i creem un nou fitxer:



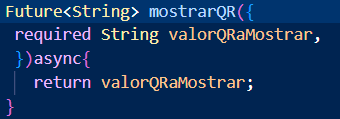


Ara en el fitxer inicialitzarem la classe si volem afegir alguna variable si volem, en este cas com no fa falta ho deixem buit.



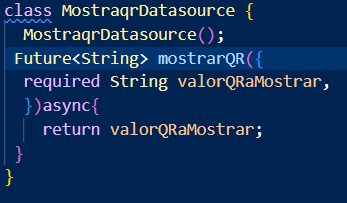
Inicialització de la classe

Quan ja hem inicialitzat el DataSource farem les funcions que faran falta, com en este cas sols és una i és molt senzilla sols torna el valor de forma asíncrona per a poder mostrar un QR que està en la pantalla, sols ens interessa el valor que tindrà el QR, així que la implementem:



Funcions del DataSource

Quan implementem la funció hauria de quedar de la següent forma:



Classe completa del DataSource

###### 4.2.3.1.2.2 Implementació del repositori

Anem a la carpeta del repositori i creem un nou fitxer:

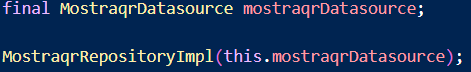




Quan creem la classe és important que implemente al repositori (sense implementació) perquè herete totes les funcions del repositori (sense implementació) i les pugues implementar:

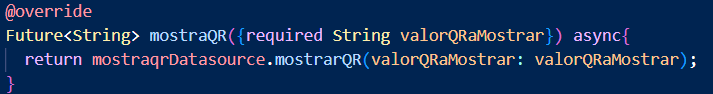
Implementació del repositori (sense implementació)

Per a fer les funcions del repositori és important que s’inicialitze el DataSource, perquè gastarem este últim per a fer les funcions i estalviar-mon codi:



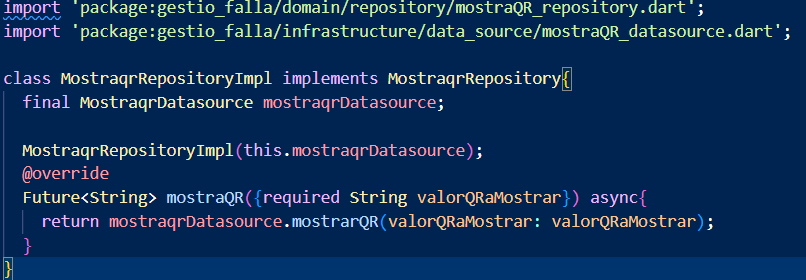
Inicialització del DataSource

Com he dit abans les funcions seran exactament les mateixes que el repositori (sense implementació) així que és important que dalt de la funció poseu “@override” perquè sobreescriurà la funció del repostitori (sense implementació) i cridarem a la instància de DataSource que havíem creat abans:



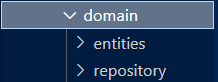
Funcions del repositori (implementació) amb @override del repositori (sense implementació)

Al final deuria quedar de la següent forma:



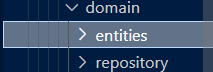
##### 4.2.3.1.3 Capa de domini

Primerament creem la següent cadena de carpetes si no les teniu creades:



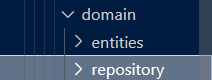
###### 4.2.3.1.3.1 Entitats

En este cas l'entitat no forma part directament en esta funció, però com en la pantalla que implementem l'arquitectura parlaré un poc d’esta, així que anem a la carpeta d'entitats:



###### 4.2.3.1.3.2 Repositori

Anem a la carpeta de repositoris i creem un nou fitxer:





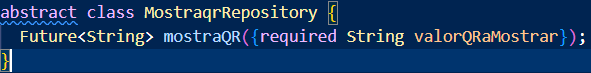
Ara per a crear la classe l'hem de fer abstracta perquè servixen per a definir una estructura comuna o un contracte que altres classes han de seguir, però sense implementar completament els detalls, aleshores implementem la classe abstracta:



Implementació de la classe abstracta del repositori

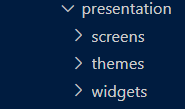
I per últim fem les funcions que han de ser iguals al DataSource perquè el repositori amb implementació puga usar el DataSource:

Al final deuria quedar de la següent forma:



##### 4.2.3.1.4 Capa de Presentació

Creeu la següent cadena de carpetes si no les teniu creades:



Sols tinc les pantalles perquè no me s’ha vingut al cap algun widget personalitzat o un tema en específic.

###### 4.2.3.1.4.1 Pantalles

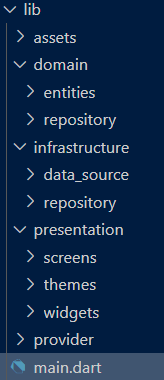
Ací és on ocorrerà la màgia que aparega en la pantalla, no n’hi ha molt més misteri, sols que ací podràs aplicar tant els DataSources, repositoris i/o gestors d’estats (BLOC,Provider, Riverpod..., etc).

##### 4.2.3.1.5 Capa de gestor d’estats

###### 4.2.3.1.5.1 Provider

Provider és un gestor d’estat més avançat i més senzill que BLOC, un altre gestor de l’estat, ara diré com podeu implementar provider en els vostres projectes:

1. Tindre tant la capa de serveis tant la de repositori ja feta.



Fixeu-vos en les carpetes de:

domain -> repository

infrastructure -> repository i datasource

2. Crear la capa de provider en la carpeta anomenada “lib”.

3. Crear el fitxer “nomdelafuncioProvider.dart” i al crear la classe que implemente “ChangeNotifier”.

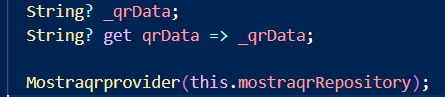


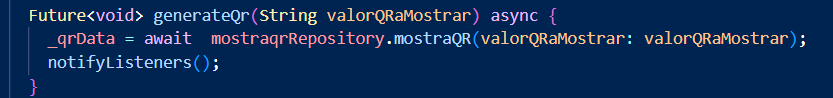


4. Inicialitza el provider amb el repositori.



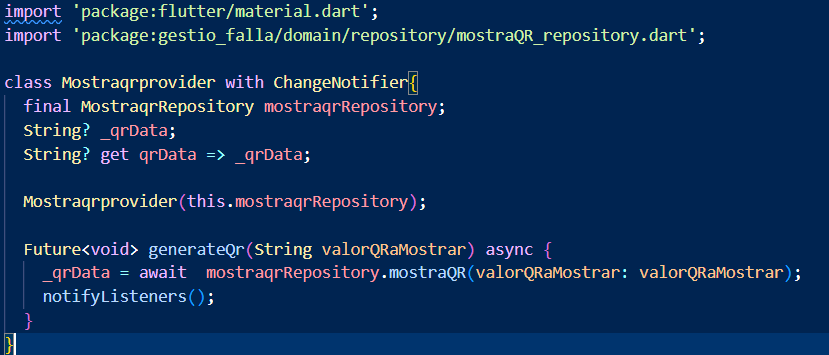
5. Completar el provider amb les funcions del repositori.



Funcions bàsiques del provider que es veuran en la pantalla  
+  
Inicialització del repository al provider

Funció que implementa el repositori i mostrara guarda en una variable de provider perquè es puga veure per pantalla

El fitxer provider deuria quedar de la següent forma:

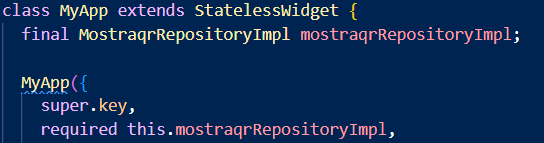


6. Inicialitzar al fitxer “main.dart”.

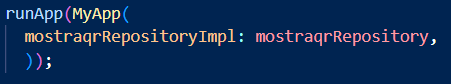




Inicialització del datasource i el repositori (la implementació)

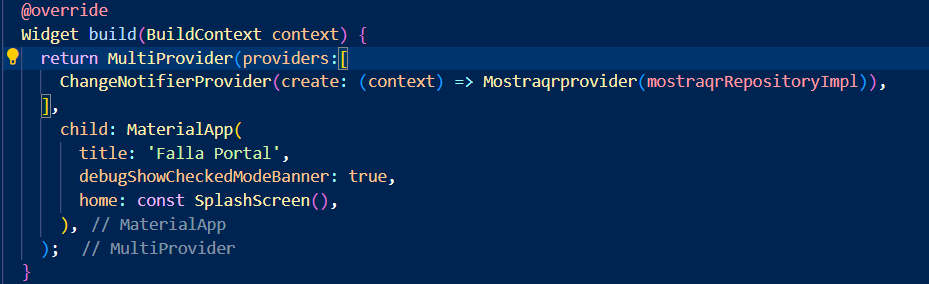


Implementem el repositori i fem que el fitxer main el requerisca per a funcionar



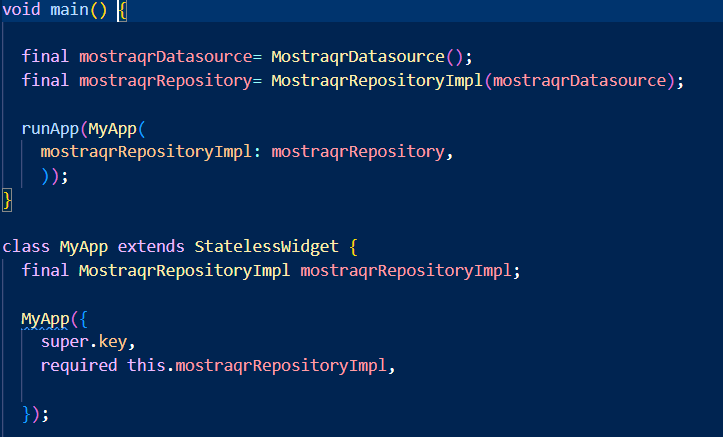
Per últim posem el repositori que havíem inicialitzat al principi

7. Fer que retorne “Multiprovider” per a tindre una llista de providers.



Fem que en lloc de retornar MaterialApp retorne el Multiprovider i el MaterialApp serà un fill de Multiprovider per a implementar la resta del main amb normalitat.

Al final el fitxer main deuria quedar de la següent forma:



8. En la pantalla que vulgues, inicialitza la funció amb “Provider.of<nomdelafuncioProvider&gt;”.



Inicialitzem el provider i li posem el valor de la funció que demanava (en este cas és un objecte de tipus String, però no n’hi haurà problema a l’hora de mostrar-lo).

Ara en la funció inicialitzada la implementem en la pantalla:



En el widget Consumer posem el nom de la classe de Provider i en el seu constructor (builder) posem el provider i ja podem usar la funció de provider dins de la pantalla.

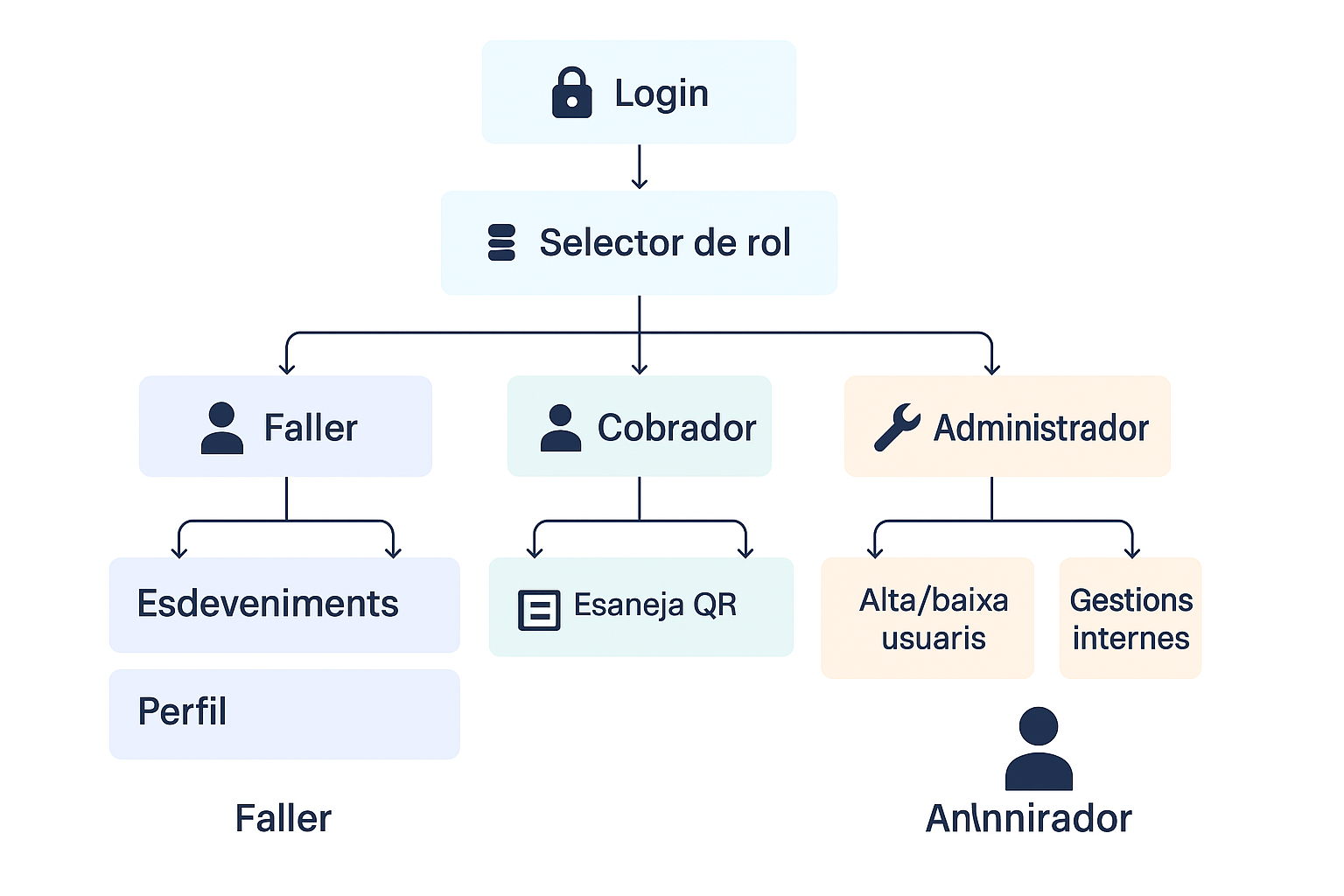
I amb això ja pots usar la teua provider en qualsevol lloc de l’aplicació.



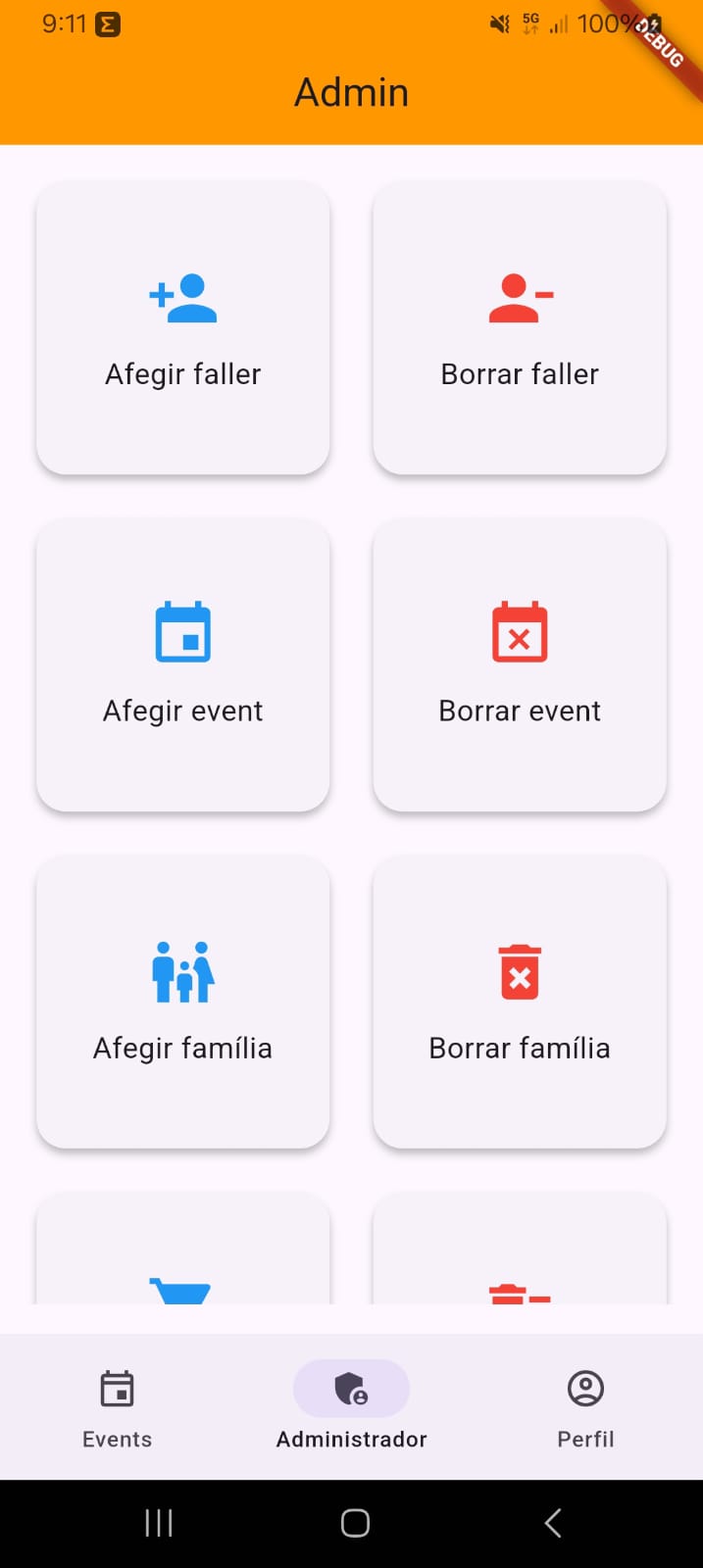
En este exemple com és un codi QR no es veu molt bé el resultat, però com el codi és generat per un *“*String*”* es pot veure de forma clara com s’ha dibuixat en la pantalla.

#### 4.2.3.4 Vista per rols

Este apartat tracta de descriure una de les funcionalitats més importants de l’aplicació, la vista per rol, què és això?  
La vista per rol és el que vora en l’aplicació per exemple un usuari normal i un usuari administrador, els 2 no tindran les mateixes pantalles, un usuari normal tindria accés al seu perfil i un llista d’esdeveniments pròxims mentre que un usuari administrador tindria una pantalla en la qual puga administrar tot el que té la falla, vegem-ho amb un exemple gràfic:



Com veiem al fer “login” en un usuari amb el rol faller sols tenim accés a la pantalla d’esdeveniments i a la de perfil, el cobrador tindrà accés a la pantalla d’escanejar nfc i qr, i per últim el usuari administrador tindrà accés a la pantalla d’administrador, ací n’hi ha un exemple de l’aplicació executant-se en cada rol:



Faller Cobrador Admin

Ací es veu de forma més detallada com cada rol té una pantalla diferent, notant-se per la part inferior que és una barra de navegació que permet el canvi de pantalles, però no de rols.

# 5.Dificultats i millores futures

## 5.1 Dificultats

✔️ Adaptar la pantalla del mòbil perquè no se sobreïsca en horitzontal era un problema que tenia en les primeres versions de l’aplicació, vaig estar batallant amb este tema unes setmanes fins que vaig poder fer que no s’isca i poder implantar un SingleChildScrollView perquè si se n’ix es puga tirar cap al costat per on se n’ix aconseguint un resultat que estic molt content.

❌ Fer que el servidor de node estiga en el núvol el qual vaig voler implementar per a no haver d'anar canviant en l'app de Flutter la IP cada 2x3, però al final no va ser possible per falta de temps i em vaig estar embossat durant setmanes mirant com solucionar-ho sense èxit.

## 5.2 Millores futures

Passar del servidor de Node a Odoo mitjançant jsonrpc, com he dit abans vaig haver de deixar Odoo de costat per a poder fer que funcione en Node, però per falta de temps no vaig poder canviar-ho altra vegada a Odoo, tenia unes funcions molt bàsiques, però res comparades al servidor de Node.js.

Compatibilitat en iOS, encara que jo no tinga un dispositiu

d'Apple o un dispositiu amb el sistema operatiu de iOS

m’agradaria poder fer una versió completament funcional en este

sistema operatiu per a no deixar als usuaris d'Apple sense

l’aplicació.

## 5.3 Que he aprés

Amb la realització d’este projecte he descobert que és la tecnologia NFC i perquè servix, junt amb el codi QR, que ja sabia que existien, però no sabia com funcionaven.

# 6.Annexos

Ací estarà l’enllaç al repositori

## 6.1 Github

<https://github.com/JoelFusterBosch/Projecte_Final_DAM_Gestio-_d_una_falla>

## 6.2 Finalització del projecte

En este apartat m’agradaria poder-me acomiadar-me d’este curs i este cicle tal com es mereix, en estos 2 anys han passat coses que recordaré per a sempre tant per a bé: els companys, els professors, els mòduls les experiències, les eixides..., etc. També tant per a mal la DANA, l'estrés que teníem per alguns exàmens, sobrecàrrega de pràctiques pel poc temps que teníem, les presentacions que em posava nerviós i em jugava la nota de l'avaluació o del curs..., etc.

Però no oblidaré el temps que he gaudit en este centre educatiu i ho recordaré en bona estima i sabent el lluny que he arribat des de que vaig entrar en este centre en 2017 amb 11 anys i ara escrivint esta part en 2025 amb 19 anys per a presentar este projecte que ha costat llàgrimes i suor fer, així que una vegada més gràcies per llegir i gràcies per esta experiència inoblidable plena d’experiències i sofriment que em van marcar com a persona, amb això m'acomiade, fins a la pròxima.

Amb estima,  
Joel Fuster Bosch

# Bibliografia

2.Tecnologies i ferramentes

2.1.1 Flutter

<https://www.cnmc.es/prensa/panel-hogares-usos-internet-20231103#:~:text=Un%2078%2C8%20%%20de%20los%20espa%C3%B1oles%20con,Internet%20y%20los%20servicios%20OTT%20en%20Espa%C3%B1a.>

<https://flutter.dev/>

<https://es.wikipedia.org/wiki/Flutter_(software)>

2.1.2 Dart

<https://dart.dev/>

<https://es.wikipedia.org/wiki/Dart>

2.1.3 Visual Studio Code

<https://code.visualstudio.com/>

<https://es.wikipedia.org/wiki/Visual_Studio_Code>

2.1.4 Github

<https://github.com/>  
<https://es.wikipedia.org/wiki/GitHub>

2.1.5 Figma

<https://www.figma.com/>

<https://es.wikipedia.org/wiki/Figma>

2.1.6 Node.js

<https://nodejs.org/es>

<https://es.wikipedia.org/wiki/Node.js>

2.1.7 PostgreSQL

<https://www.postgresql.org/>

<https://es.wikipedia.org/wiki/PostgreSQL>

2.1.8 Odoo

<https://www.odoo.com/es_ES>

<https://es.wikipedia.org/wiki/Odoo>

4.Desenvolupament de la solució

4.2Configuració i desenvolupament del sistema

4.2.1.1 Instal·lació de Flutter i les seues dependències

<https://docs.flutter.dev/get-started/install>

4.2.1.2 Instal·lar l’entorn de desenvolupament

Pàgines oficials de les extensions de Visual Studio Code:

<https://dartcode.org/>

<https://marketplace.visualstudio.com/items?itemName=Nash.awesome-flutter-snippets>

4.2.3 Arquitectura CLEAN

<https://blog.burkharts.net/practical-flutter-architecture>

6.Annexos

6.1 Github

<https://github.com/JoelFusterBosch/Projecte_Final_DAM_Gestio-_d_una_falla>