



UNIVERSITAT POLITÈCNICA DE CATALUNYA

Escola Superior d'Enginyeria de Manresa

---

# Gestió automatitzada d'un pàrquing

AppArkem

5 de maig de 2022

---

Treball de fi de grau que presenta

**GEMMA ROSELL GUILELLA**

en compliment dels requisits per assolir el

**GRAU D'ENGINYERIA EN SISTEMES TIC**

Direcció: Aleix LLusà Serra



This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License. To view a copy of this license, visit <https://creativecommons.org/licenses/by-nc-sa/4.0/>.

## **Agraïments**

Vull agrair al director del projecte Aleix Llusà Serra, que m'ha ajudat en el seu desenvolupament donant-me pautes per l'assoliment de l'objectiu i en la realització de la memòria.

Finalment, donar gràcies a la meva família i els amics que m'han animat i ajudat al llarg d'aquest projecte, com també durant els anys cursant la carrera.



## Resum

Aquest projecte consisteix en el disseny i el desenvolupament d'*Apparkem*, una aplicació per a la gestió automatitzada d'un pàrquing intel·ligent. S'introduceix la creació d'un servidor back-end, el front-end de l'aplicació i una Raspberry Pi que actua com a barrera. Finalment, s'observen els resultats obtinguts com també possibles propostes de millora per l'aplicació.

## Abstract

This project consists of the design and development of *Apparkem*, an application for the automated management of a smart parking lot. It introduces the creation of a back-end server, the front-end of the application and a Raspberry Pi that acts as a barrier. Finally, the results obtained are observed as well as possible improvement proposals for the application.



# Índex

<b>Resum</b>	<b>i</b>
<b>Abstract</b>	<b>i</b>
<b>I Memòria</b>	<b>1</b>
<b>1 Introducció</b>	<b>3</b>
1.1 Objectius . . . . .	3
1.2 Conceptes previs . . . . .	4
1.3 Alternatives per l'accés al pàrquing . . . . .	5
1.3.1 Lectors de matrícules . . . . .	5
1.3.2 Comunicació amb Bluetooth . . . . .	6
1.3.3 Introducció codi QR . . . . .	7
1.4 Altres aplicacions . . . . .	9
<b>2 Arquitectura del sistema</b>	<b>11</b>
2.1 Back-end . . . . .	12
2.2 Front-end . . . . .	12
2.3 Barrera . . . . .	13
<b>3 Back-end</b>	<b>15</b>
3.1 Elecció de l'entorn . . . . .	15
3.2 Base de dades . . . . .	16
3.3 Rest API . . . . .	20
3.3.1 Autentificació . . . . .	20
3.3.2 Barreres . . . . .	23
3.3.3 Cotxes . . . . .	25
3.3.4 QRs . . . . .	28
3.3.5 Pagament . . . . .	29
3.3.6 Configuracions . . . . .	30
3.4 Tests . . . . .	31
3.4.1 Test <i>AuthTest</i> . . . . .	31
3.4.2 Test <i>CarsTest</i> . . . . .	31
3.4.3 Test <i>LogsTest</i> . . . . .	32
3.4.4 Test <i>SettingsTest</i> . . . . .	32

<b>4</b>	<b>Front-end</b>	<b>33</b>
4.1	Elecció de l'entorn . . . . .	33
4.2	Pàgines . . . . .	34
4.2.1	Iniciar Sessió . . . . .	34
4.2.2	Registrar usuari . . . . .	34
4.2.3	Principal . . . . .	34
4.2.4	Codi QR . . . . .	35
4.2.5	Informativa . . . . .	36
4.2.6	Pagament . . . . .	36
4.2.7	Panell administració . . . . .	36
4.3	Eines utilitzades . . . . .	37
<b>5</b>	<b>Raspberry Pi</b>	<b>39</b>
5.0.1	Connexions . . . . .	39
5.0.2	Màquina d'estats . . . . .	41
<b>6</b>	<b>Connexions de hardware</b>	<b>43</b>
<b>7</b>	<b>Desglossament econòmic de la demo</b>	<b>45</b>
7.1	Material necessari per barrera . . . . .	45
<b>8</b>	<b>Conclusions</b>	<b>47</b>
8.1	Treball futur . . . . .	48
<b>9</b>	<b>Bibliografia</b>	<b>49</b>
<b>II</b>	<b>Annexos</b>	<b>51</b>
<b>A</b>	<b>Images del front-end de l'aplicació</b>	<b>53</b>
A.1	Alertes . . . . .	65
<b>B</b>	<b>Enllaços a vídeos de l'aplicació</b>	<b>67</b>

# Índex de figures

1.1	Exemple d'un Codi QR . . . . .	7
1.2	Estructura dels Codis QR . . . . .	8
2.1	Arquitectura del sistema . . . . .	11
2.2	Relació entre els elements del model MVC [Vik] . . . . .	12
3.1	Estructura base de dades . . . . .	16
3.2	Tests Correctes . . . . .	32
5.1	Raspberry PI 4 pinout . . . . .	39
5.2	Muntatge Raspberry Pi . . . . .	40
5.3	Màquina d'estat detecció codi QR . . . . .	41
6.1	Connexions dels dispositius . . . . .	43
A.1	Pantalla d'iniciar sessió . . . . .	53
A.2	Pantalla de registrar usuaris . . . . .	54
A.3	Pantalla principal . . . . .	55
A.4	Pantalla d'editar informació de l'usuari . . . . .	56
A.5	Pantalla editar cotxes de l'usuari . . . . .	57
A.6	Pantalla d'afegir cotxes de l'usuari . . . . .	58
A.7	Pantalla generar codi QR . . . . .	59
A.8	Pantalla d'informació de l'aparcament . . . . .	60
A.9	Pantalla del pagament . . . . .	61
A.10	Pantalla del pagament validat . . . . .	62
A.11	Pantalla del pagament cancel·lat . . . . .	63
A.12	Pantalla panell d'administració . . . . .	64
A.13	Pantalla panell d'informació de pagament a <i>Stripe</i> . . . . .	64
A.14	Alerta d'èxit . . . . .	65
A.15	Alerta d'error . . . . .	66



# Índex de taules

3.1	Taula <i>Users</i> de la base de dades . . . . .	17
3.2	Taula <i>Cars</i> de la base de dades . . . . .	18
3.3	Taula <i>Spaces</i> de la base de dades . . . . .	18
3.4	Taula <i>Settings</i> de la base de dades . . . . .	19
3.5	Taula <i>Logs</i> de la base de dades . . . . .	19
3.6	Taula rest API <i>Auth</i> . . . . .	20
3.7	Taula rest API <i>Barriers</i> . . . . .	23
3.8	Taula regles de validació rest API <i>QR</i> . . . . .	23
3.9	Taula rest API <i>Cars</i> . . . . .	25
3.10	Taula regles de validació rest API <i>Cars</i> . . . . .	26
3.11	Taula rest API <i>QR</i> . . . . .	28
3.12	Taula rest API <i>Payments</i> . . . . .	29
3.13	Taula regles de validació rest API <i>Payments</i> . . . . .	29
3.14	Taula rest API <i>Settings</i> . . . . .	30
3.15	Taula regles de validació rest API <i>Settings</i> . . . . .	31
7.1	Preus material necessari per a la demostració . . . . .	45



# **Part I**

## **Memòria**



# 1 Introducció

Aquest projecte consisteix en dissenyar una aplicació per a la gestió automatitzada d'un pàrquing intel·ligent. Aquest control es basa en l'entrada i sortida de vehicles d'aquest pàrquing fent servir altres alternatives al lector de matrícules convencional. A més a més, l'aplicació tindrà un registre de l'estança de cada usuari, com també la part de configuració pels treballadors.

Algunes d'aquestes alternatives són la comunicació Bluetooth i els codis QR. En aquest projecte se centra a crear una aplicació on s'utilitza els codis QR.

També, aquesta aplicació dóna l'eina perquè l'usuari faci els pagaments a través d'ella d'una manera segura.

## 1.1 Objectius

L'objectiu es pot dividir en els punts següents:

- Buscar informació sobre els generadors i lectors dels codis QR, com també dels lectors de matrícules i la comunicació Bluetooth.
- Aprendre a dissenyar un *backend* i un *frontend*.
- Aprendre tots els passos que comporta la creació d'una aplicació i fer-los segons les necessitats del projecte.
- Dissenyar un sistema on emmagatzemar les dades, una base de dades. Aprendre MySQL.
- Dissenyar l'aplicació perquè l'usuari tingui un bon aprenentatge quan entri per a primer cop, aquesta tingui una eficiència excepcional i sobretot hi hagi seguretat.
- Triar dues framework per al desenvolupament de l'aplicació, una per el *backend* i l'altre per el *frontend*.
- Aprendre els passos de la framework de React, com també la del Laravel.

## 1.2 Conceptes previs

- El *back-end* és la capa d'accés a les dades, és la lògica que fa que una pàgina web funcioni, cosa que queda amagada als ulls de l'usuari. Determina que tan bé s'executarà l'aplicació i quina experiència obtindrà l'usuari del seu ús, positiva o negativa. Es fan servir habitualment llenguatges com PHP, Javascript, Python i Ruby, entre d'altres.
- El *front-end* és la part del desenvolupament que es dedica el disseny, des de l'estructura del lloc fins als estils com colors, fons, mides fins a arribar a les animacions i efectes. És aquesta part de la pàgina amb què interaccionen els usuaris, és tot el que el visitant veu i experimenta de manera directa.
- *MySQL* és un sistema de gestió de bases de dades relacional que usa el llenguatge SQL per comunicar-s'hi. Algunes d'aquestes comunicacions poden ser consultar les dades, manipular-les, etc.
- Una *framework* és una eina de desenvolupament web que ens permet desenvolupar aplicacions d'una manera més àgil utilitzant les seves llibreries i funcionalitats. Ajuda als programadors a seguir una estructura, estalviar temps i esforços, ja que d'aquesta manera es centrin a buscar solucions a nous problemes i a «No reinventar la roda».
- *React* [Jor] és una llibreria de JavaScript per desenvolupar interfícies d'usuari.
- *Laravel* [Larl] és una framework del llenguatge de programació PHP. S'utilitza per al desenvolupament d'applicacions web.

## 1.3 Alternatives per l'accés al pàrquing

En aquest apartat s'avaluen tres metodologies diferents per a poder control l'accés al pàrquing que són el lector de matrícules, la connexió Bluetooth i els codis QR.

### 1.3.1 Lectors de matrícules

Els sistemes de lectors de matrícula són especialment utilitats gràcies a la seva fàcil instal·lació i la fiabilitat de la lectura. Alguns exemples on es poden trobar és en el control de pàrquings i els controls de trànsit.

#### Funcionament

- Captar en una imatge la matrícula a través d'una càmera.
- El software fa una interpretació d'aquesta imatge.
- S'inclouen les dades interpretades en una base de dades.

Els lectors de matrícules permeten transformar una imatge presa des d'una càmera i mitjançant un procés de visió artificial, a un paràmetre digital. El sistema és capaç de discriminar l'entorn i fixar-se només a la placa d'una matrícula, creant un codi únic per a ella.

#### Conclusió

La seva fiabilitat en la lectura i facilitat de la instal·lació converteix en una eina útil, segura i necessària. Tanmateix, s'ha de tenir present la resposta ràpida en un possible fet que provoqui un incident i ens desenvolupi un problema en el nostre pàrquing.

En aquesta projecte, s'estan buscant diferents solucions per poder resoldre aquestes dificultats.

### 1.3.2 Comunicació amb Bluetooth

La comunicació amb Bluetooth entre una aplicació i un dispositiu que serà col·locat en la barrera del pàrquing. Aquest dispositiu s'utilitza per al control d'entrada i sortida d'un garatge privat o d'un pàrquing d'una institució on els treballadors tenen accés a aquest servei.

#### Carecterístiques

- Obtenir el dispositiu i col·locar-lo en la barrera del pàrquing.
- Una aplicació que serà el comandament del garatge o el tiquet per entrar al pàrquing.
- Donar accés a altres persones.
- Tenir un registre d'entrades i sortides.
- Comunicació amb Bluetooth encriptat.

#### Funcionament

El Bluetooth és un protocol de comunicació que ens permet la transmissió de dades entre diferents dispositius que es troben a poca distància. La seva transmissió sense fils es fa través d'ones de ràdio. Per aquesta raó els dispositius han d'estar lleugerament a prop. Tot i això, podem descobrir diferents dispositius que es classifiquen segons la seva potència i de la distància on són detectats.

#### Conclusió

Per acabar, el Bluetooth és un dispositiu intel·ligent on tindria un paper important a l'hora d'usar-lo en els garatges o en els pàrquings. Actualment, ja existeixen alguns dispositius que fan aquest servei, ja que s'installa un dispositiu junt amb la barrera d'accés i a través aquesta tecnologia es connecta amb el dispositiu mòbil, que aquest actua com a comandament per poder entrar o sortir d'aquest. Una alternativa als codis QR utilitzada a garatges privats, o a pàrquings d'institucions privades.

### 1.3.3 Introducció codi QR

Els codis QR (Quick Response code) són un tipus de codis de barres de dues dimensions en forma de matriu que permet emmagatzemar informació. La seva forma és quadrada que el caracteritza tres quadres situats a les cantonades superiors i inferiors. Dependent a la informació que es vulgui emmagatzemar, els codis canviaran la seva mida.



Figura 1.1 – Exemple d'un Codi QR

La utilització inicial dels codis QR tenia com a objectiu principal en la indústria de l'automoció. Actualment, la possibilitat de llegir els codis QR és existent, ja sigui amb el telèfon mòbil que ens permet l'ús d'aquests codis en una infinitat d'aplicacions. Una de les raons principals són perquè aquests codis poden interactuar fàcilment amb un dispositiu mòbil i realitzar accions. En l'actualitat, a causa de la Covid-19, s'ha fet ús d'aquesta eina per poder-nos descarregar o en la lectura d'un codi QR, llegir una URL d'una pàgina web per descarregar-nos la carta o el menú d'un restaurant.

#### Estructura Codi QR

En aquest subapartat es fa referència a l'estructura i les especificacions del codi QR.

- *Marques d'orientació*: A dins d'aquestes marques podem trobar els patrons de posició i els d'alignació. Estan situades als extrems del codi QR. S'utilitzen per saber l'orientació del codi. Poden variar depenent de la dimensió del codi QR.
  - *Patrons de posició*: Són tres quadrats que estan als extrems del codi QR.
  - *patrons d'alignació*: S'usa per a la detecció de la posició.
- *Informació del format*: S'empra per saber quin nivell d'error i quina màscara s'ha aplicat en el codi.
- *Mètrica*: Igual a tots els codis QR i és feta servir per identificar els punts blancs i negres de la matriu.
- *Dades*: Les dades s'emmagatzemen en codewords de mòduls. Els codewords són conjunts de 8 bits que varien la seva forma. N'hi ha de dos tipus, el d'informació i

el de correcció d'errors. Les dades que hi ha tenen uns bits dedicats a especificar el format de les dades, i la quantitat de caràcters que hi ha en el QR. Actualment, els QR accepten diferents formats per les dades.

- *Codis correcció d'errors:* Proporcionen la capacitat de correcció dels codis QR, que s'expressa en un percentatge i correspon al tant per cent de dades que es podrà recuperar en cas d'error. Actualment, hi ha 4 nivells de correcció pels codis QR:
  - L 7%
  - M 15%
  - Q 25%
  - H 30%
- *Informació de la versió:* Conté la informació de la versió que s'està fent servir. Situada en els laterals de dins de les marques d'orientació.
- *Sense ús:* Es troben en algunes versions dels codis QR, l'últim codeword dels codis d'error no es fa servir.

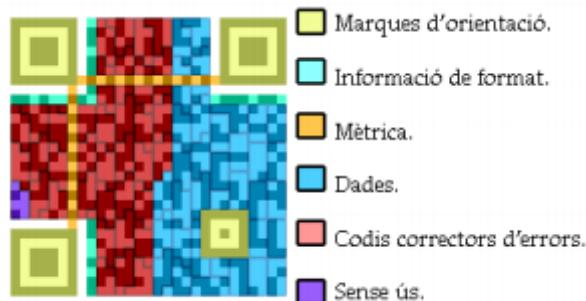


Figura 1.2 – Estructura dels Codis QR

## Generar Codi QR

En aquest projecte l'encarregat de fer el codi QR i mostrar-lo als usuaris és el *front-end* de l'aplicació. Encara que l'encarregat de generar el text és el servidor. Capítol 3

## Lectura Codi QR

En aquest projecte l'encarregat de fer la lectura del QR és la Raspberry Pi actuant com a barrera fent servir una càmera. Capítol 5

## 1.4 Altres aplicacions

De les diferents alternatives que s'ha explicat en el capítol anterior Secció 1.3 existeixen diferents aplicacions que hi ha al mercat que solucionen el problema real són les següents:

1. **telpark**: Una aplicació amb la possibilitat de reservar una plaça de pàrquing posar el pàrquing on es vol aparcar, posar la durada amb possibilitat d'augmentar el temps, el vehicle i la forma de pagament. Possibilitat de pagar el parquímetre [Tel].
2. **Parkingdoor**: Aquesta aplicació és útil en garatges privats, quan es vol estalviar el comandament per pujar la porta del garatge. S'ha d'instal·lar un dispositiu junt la porta i el dispositiu mòbil que actua com a comandament. Com s'ha explicat en l'apartat de Subsubsecció 1.3.2. Més informació a [Par].
3. **El Parking**: Aquesta aplicació dóna la possibilitat de fer pagaments a través de l'aplicació. Fer reserves amb la possibilitat de cancel·lació. Més informació a [El ].

Totes aquestes aplicacions tenen la característica de funcionar amb diferents pàrquings a la vegada. L'aplicació d'aquest projecte es basa en la creació de la gestió d'un pàrquing privat, d'una institució o la possibilitat de ser un pàrquing públic. On la configuració d'aquests és única i personal, és a dir, l'aplicació és feta a mida per a cada un dels pàrquings. A més a més, amb la funcionalitat d'usar els codis QR per les entrades i sortides del pàrquing.



## 2 Arquitectura del sistema

L'arquitectura del projecte consta en tres parts:

- Un ordinador amb un servidor de Laravel, que és el *back-end*.
- El *front-end* de l'aplicació utilitzant la llibreria de React.
- Una Raspberry Pi que actua com a barrera del pàrquing.

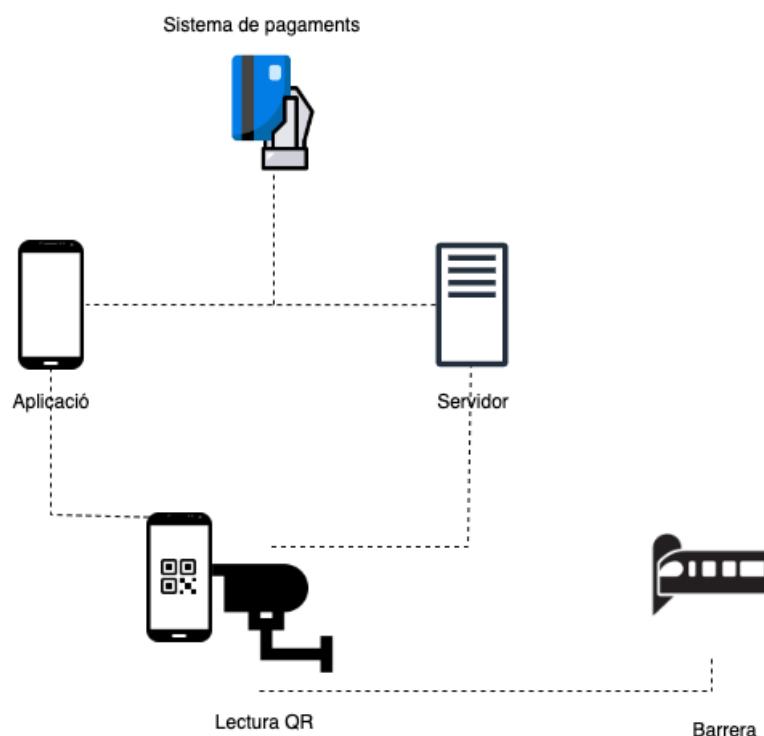


Figura 2.1 – Arquitectura del sistema

## 2.1 Back-end

L'arquitectura que utilitza la *framework* de Laravel es basa en el MVC (Model-Vista-Controlador), feta servir per la implementació d'interfícies d'usuari, dades i la lògica. Aquesta separació ens dóna una millor divisió de treball i millora el manteniment.

- Model: és qui defineix quines dades ha de tenir l'aplicació i les modifica. Si aquestes canvien, notifiquen a la vista.
- Vista: com s'han de mostrar les dades a l'aplicació per a l'usuari, és el disseny i la presentació.
- Controlador: on conté la lògica que actualitza el model. És la resposta a la sol·licitud que fa l'usuari. Es comunica amb el model.

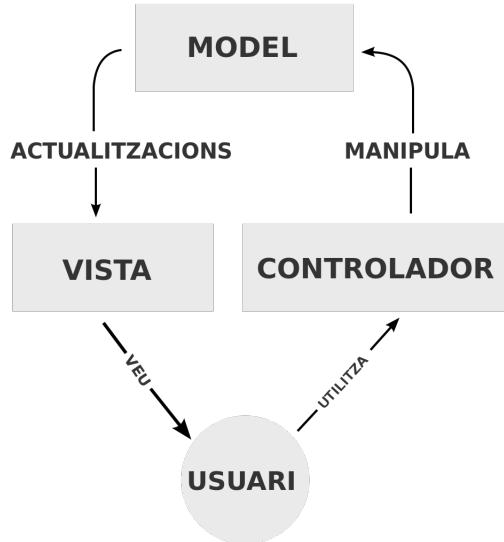


Figura 2.2 – Relació entre els elements del model MVC [Vik]

## 2.2 Front-end

El *front-end* conjuntament amb la llibreria de React la qual crea components independents que cada un d'ells és una peça on l'usuari pot interactuar. S'encarrega de crear un espai personal per l'usuari. Generar els diferents codis QR per l'accés o la sortida del pàrquing. Mostrar la informació sobre l'aparcament. Fer pagaments d'una forma senzilla i amb seguretat. A més a més disposa d'un apartat per l'administració on hi ha una part de configuració i un recull de dades.

## 2.3 Barrera

La Raspberry Pi a través de la càmera s'encarrega de detectar diferents codis QR. Aquests, són els que s'han generat amb l'aplicació del *front-end*. Aquesta detecció implica que la barrera actuï de diferent manera, pujant-la o baixant-la en funció del tipus del codi QR generat. Per aconseguir aquest efecte s'ha emprat una màquina d'estats [Enc].



# 3 Back-end

En aquest capítol es fa referència a la implementació del *back-end* de la aplicació. Està implementat amb la framework de **Laravel** [Larl] és una framework del llenguatge de programació PHP. S'utilitza per al desenvolupament d'aplicacions web.

## 3.1 Elecció de l'entorn

En aquesta secció s'explica els punts forts de la eina triada per realitzar el *back-end* de l'aplicació.

- És una de les frameworks més utilitzades [Sup].
- L' **ecosistema**, és a dir la comunitat que hi ha fent-ne ús cada dia. Facilita a l'hora de trobar solucions a problemes que es poden trobar al llarg del desenvolupament d'una aplicació. També ajuda al fet que sigui una eina on sempre hi hagi un manteniment i estigui actualitzada a noves eines per facilitar la feina o fins i tot a estalviar-ne.
- Laravel inclou **Eloquent**, mapatge d'objectes relacional (ORM) que fa que sigui agradable interactuar amb la base de dades. Cada taula de base de dades té un "Modelcorrespondent que s'utilitza per interactuar amb aquesta taula. A més de recuperar registres de la taula de la base de dades, també permeten inserir, actualitzar i eliminar registres de la taula [Larb].
- **Recursos API**, Quan l'aplicació crea una API, és possible que necessiti una capa de transformació que es trobi entre els models Eloquent i les respostes JSON que realment es retornen als usuaris de l'aplicació. Permet transformar de manera expressiva i senzilla els models i col·leccions de models en JSON. Ofereixen un control més sólid sobre la serialització JSON dels models i les seves relacions [Larc].
- Una altra eina són les **migracions**, permet definir i compartir l'esquema de la base de dades de l'aplicació [Lare].
- L'**Autenticació**, Laravel s'esforça per oferir les eines necessàries per implementar l'autenticació de manera ràpida, segura i senzilla [Lara].
- Sistema de **caché** fa que l'aplicació vagi igual de ràpid en local que en producció.

- La **validació de dades**: Laravel ofereix diversos enfocaments diferents per validar les dades entrants de l'aplicació, les sol·licituds HTTP entrants. Inclou una gran varietat de regles de validació [Larh].
- **Laravel cashier Stripe** ofereix una interfície expressiva i fluida als serveis de facturació de subscripcions de Stripe. Gestiona gairebé tot el codi de facturació de subscripció general i d'altres [Lari].
- **Laravel Sanctum** ofereix un sistema d'autenticació lleuger per a SPA (aplicacions d'una sola pàgina), aplicacions mòbils i API simples basades en tokens [Lark].

## 3.2 Base de dades

Qualsevol aplicació interactua amb una base de dades per tal de recopilar i emmagatzemar dades amb una tècnica ordenada i seguint una estructura coherent i accessible. Com s'ha esmentat en la secció anterior, Laravel utilitzà l'ORM Eloquent.

L'estructura de la base de dades és de la següent manera:

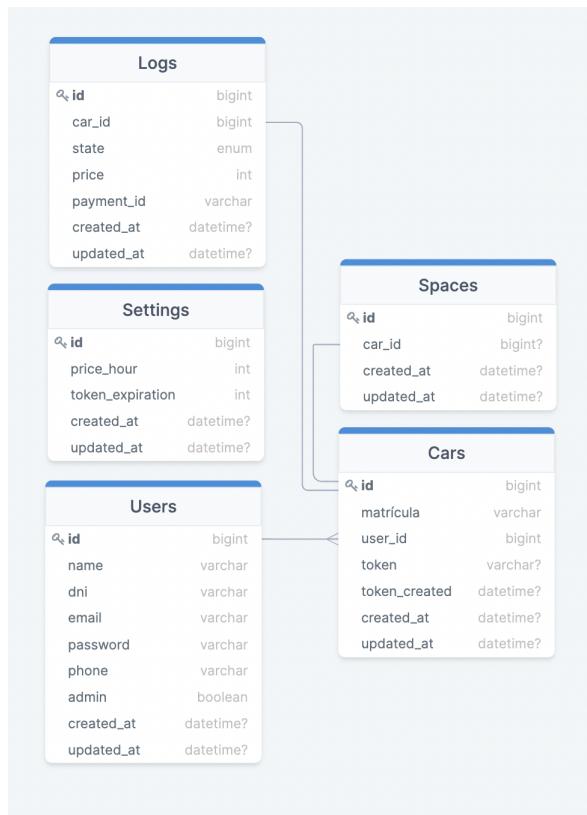


Figura 3.1 – Estructura base de dades

Columna	Tipus	Descripció
id	integer	Clau primària, única per a cada usuari. Autoincrementa el seu valor
name	varchar(255)	Nom i cognom de l'usuari.
dni	varchar(9)	Número del DNI de cada usuari. Únic per a cada usuari.
email	varchar(255)	Correu electrònic de l'usuari. Únic per a cada usuari.
password	varchar(255)	Contrasenya de l'usuari per poder entrar a l'aplicació.
phone	varchar(12)	Número de telèfon de l'usuari amb un màxim de 12 caràcters. Únic per a cada usuari.
admin	boolean	Indica si la persona és administrador. Per defecte sempre serà fals.

Taula 3.1 – Taula *Users* de la base de dades

Columna	Tipus	Descripció
id	integer	Clau primària, única per a cada usuari. Autoincrementa el seu valor.
matrícula	varchar(7)	Número de matricula del vehicle. Únic per a cada vehicle amb un màxim de 8 caràcters.
usuari_id	bigint	Clau foràna, el seu valor correspon a la ID de la taula d'usuaris. Indica de qui pertany el vehicle.
token	varchar(16)	String encriptat de 16 caràcters. Nullable. Serveix per poder construir el codi QR.
token_created_at	datetime	Es guarda en quina data s'ha generat el token anterior.

Taula 3.2 – Taula *Cars* de la base de dades

Columna	Tipus	Descripció
id	bigint	Clau primària, única per a cada usuari. Autoincrementa el seu valor
car_id	bigint	Número d'identificador del vehicle que ocupa la plaça. Clau foràna de la taula cotxes. Nullable.

Taula 3.3 – Taula *Spaces* de la base de dades

Columna	Tipus	Descripció
id	bigint	Clau primària, única per a cada usuari. Autoincrementa el seu valor
price_hour	int	Número que indica el preu hora del pàrquing.
token_expiration	int	Número que indica el temps de validesa dels codis QR

Taula 3.4 – Taula *Settings* de la base de dades

Columna	Tipus	Descripció
id	bigint	Clau primària, única per a cada usuari. Autoincrementa el seu valor
car_id	bigint	Clau foràna de la taula cotxes. Número d'identificador del vehicle.
state	enum ("enter"o "exit")	Indica si el vehicle entra o surt del pàrquing.
price	int	Preu total que li ha costat al usuari.
payment_id	varchar(255)	ID del pagament d'Stripe.

Taula 3.5 – Taula *Logs* de la base de dades

## 3.3 Rest API

En aquesta secció es fa referència a les REST API que s'han creat a l'aplicació. La *base endpoint* és <https://gemmaiguilella.cat/api>.

### 3.3.1 Autentificació

L'autentificació d'aquest projecte és a través del *Laravel Sanctum* [Lark]. Aquesta eina ofereix guardar els tokens dels usuaris en una base de dades, aquests han de ser els mateixos tokens que arriben de l'usuari a les peticions HTTP en el *Authorization header* com a *Bearer Token*.

Mètode	Ruta	Middleware	Descripció
POST	/auth/login	guest	Subsubsecció 3.3.1 Iniciar sessió
POST	/auth/logout	guest	Subsubsecció 3.3.1 Tancar sessió
POST	/auth/register	auth	Subsubsecció 3.3.1 Crear usuari
GET	/auth/user	auth	Subsubsecció 3.3.1 Obtenir usuari
PUT	/auth/user	auth	Subsubsecció 3.3.1 Modificar usuari

Taula 3.6 – Taula rest API *Auth*

El *middleware auth*, vol dir que es passa el *Authorization header* [Lard].

#### Iniciar sessió

Un exemple del *body* d'una petició HTTP per realitzar l'inici de sessió és enviar el següent JSON:

```
{
  "email": "gemma@guilella.cat",
  "password": "Password123!"
  "device_name": "web"
}
```

El servidr pot respondre amb les dades del usuari junt amb un *token* identificatiu que es necessita per autentificar futures peticions HTTP:

```
{
  "token": "10|6MT6IDkmuMqYGDzCg4SwuVcAczASx2BI5prpZiUe",
  "user": {
    "id": 2,
    "name": "Gemma",
```

```

    "email": "gemma@guilella.cat",
    "dni": "39393939B",
    "phone": "34625664404",
    "admin": 0
}
}

```

### Tancar sessió

Per realitzar la petició de HTTP per tancar la sessió és enviar aquesta informació. En format JSON és el següent:

```
{
  "device_name": "web"
}
```

### Crear usuari

Un exemple del *body* d'una petició HTTP per realitzar registre d'usuaris és enviar el següent JSON:

```
{
  "name": "Gemma",
  "email": "gemma@guilella.cat",
  "dni": "39393939B",
  "password": "Password123!",
  "password_confirmation": "Password123!",
  "phone": "34625664404",
  "device_name": "web"
}
```

La resposta amb JSON del servidor posterior d'enviar el passat exemple és amb les dades del usuari junt amb el *token* identificatiu de l'usuari.

```
{
  "token": "10|6MT6IDkmuMqYGDzCg4SwuVcAczASx2BI5prpZiUe",
  "user": {
    "id": 2,
    "name": "Gemma",
    "email": "gemma@guilella.cat",
    "dni": "39393939B",
    "phone": "34625664404",
    "admin": 0
  }
}
```

## Obtenir usuari

Un exemple de la petició HTTP per obtenir la informació d'un usuari és enviar en els *headers* el *token* de l'usuari.

```
{
  "Authorization": "Bearer
→ 16|89pozuuct0qgewfAPuZR00cu3hMSmkuUUOKzpdDz"
}
```

La resposta del servidor a la petició HTTP anterior és donar la informació de l'usuari.

```
{
  "admin": 0
  "dni": "39393939B"
  "email": "gemma@guilella.cat"
  "id": 2
  "name": "Gemma Rosell Guilella"
  "phone": "+34625664404"
}
```

## Modificar usuari

Un exemple del *body* d'una petició HTTP per modificar la informació d'un usuari és enviar en el següent JSON:

```
{
  "name": "Gemma Rosell Guilella",
  "dni": "39393939B",
  "password": "Password124!",
  "password_confirmation": "Password124!",
  "phone": "34625664404",
  "device_name": "web"
}
```

La resposta del servidor a la petició HTTP anterior és donar la informació del usuari. Seguint l'exemple anterior, el JSON de la resposta és el següent:

```
{
  "admin": 0
  "dni": "39393939B"
  "email": "gemma@guilella.cat"
  "id": 2
  "name": "Gemma Rosell Guilella"
  "phone": "+34625664404"
}
```

### 3.3.2 Barreres

Mètode	Ruta	Descripció
POST	/barriers/open	Subsubsecció 3.3.2 Pujar barrera

Taula 3.7 – Taula rest API *Barriers*

#### Pujar barrera

Aquesta API la utilitza la barrera que és la Raspberry Pi. Un exemple del *body* de la petició HTTP que pot fer és el següent JSON:

```
{
  "qr": "eyJpdiI6IlFLbVR3WFhBaUZJaU01ZHN3SDN1Q0E9PSIsInZhbHV1IjoiNHFJ
  ↵ S2RaZmdKZEtQeUNZM01kamhFVDY2QTBFZE9od0UraFN0OF1tZD1hZk13ZGtiWmRxSDd
  ↵ DNWZLT01Uc1dWU255YmNZV2hCdWtHaTVyTFhyck5GNHozQ2tGVUR2bmlivYnpIaDNhV2
  ↵ 5PTWhuOXNXTy93ME1DNUJnMXJiaEZkNmkiLCJtYWMiOiJiN2E0NmI4MTJ1MzVkJQzZ
  ↵ mNh0TQyY2Iz0DQxYT1zMDC30Tc4NTdhZjZhZjlhMTQxNTdiNmU5MTg1NGNmMjZhIiwi
  ↵ dGFnIjoiIn0=",
}
```

El servidor un cop rebuda la petició de la Raspberry el primer que fa és validar les dades. En aquesta taula es pot veure les regles de validació que ha de seguir el paràmetre de la petició:

Nom	Tipus	Descripció
qr	string	required

Taula 3.8 – Taula regles de validació rest API *QR*

Un cop s'ha validat el paràmetre del *qr* el servidor desencripta el codi QR de la sol·licitud. El codi QR està compost per la id del cotxe del usuari, un *token* i el *token\_created\_at*.

El servidor té ara el qr desencriptat i pot obtenir les dades per separat del codi QR. Un exemple visual en format JSON és el següent:

```
{
  "id": 1
  "token": "rgsL8kDf7ihZ2Bpj"
  "token\char`_created\char`_at": "2022-04-29 19:08:42"
}
```

Prèviament, el servidor fa dues comprovacions:

1. Comprova si el *token* del codi QR desencriptat és el mateix que el *token* de la base de dades de *Cars*.
2. Comprova si el *token\_created\_at* es més gran que el *token\_expiration* de la base de dades de *Settings* Secció 3.2.

Si no es compleix les dues comprovacions anteriors el servidor lanza una `HttpException`, un `abort` amb el codi 406 [MDNc], si no es compleix la segona comprovació el servidor abans de llançar aquesta `HttpException` actualitza el valor del *token* de la base de dades de *Cars* a *NULL*.

Si es compleixen, el servidor també actualitza el valor del *token* de la base de dades de *Cars* a *NULL*. Comprova si el cotxe es troba dins el pàrquing, ja que depenen del seu estat es fan accions diferents.

En el primer cas, el cotxe no es troba dins del pàrquing. L'acció és entrar a dins al pàrquing. El servidor actua de la següent manera:

1. El servidor assigna la primera plaça de pàrquing lliure per el cotxe que vol accedir.
2. Envia un esdeveniment al *front-end* del projecte utilitzant pusher [Pus].
3. Afegeix una fila a la taula *Logs* de la base de dades on es guarda:
  - A la columna *state* el valor d'entrada (Enter).
  - A la columna *price* es posa a *Null*.
  - A la columna *payment\_id* es posa a *Null*.
4. Per últim, el servidor traeix com a resposta un *no content* amb el codi 204 [MD-Na].

En el segon cas, el cotxe es troba dins del pàrquing. La seva acció és sortir d'ell després de fer el pagament. Les accions que efectua són les següents:

1. El servidor desassigna la plaça de pàrquing associada al cotxe, és a dir la posa a *NULL*.
2. Afegeix una fila a la taula *Logs* de la base de dades on es guarda:
  - A la columna *state* el valor de sortida (Exit).
  - A la columna *price* es posa a *Null*.
  - A la columna *payment\_id* es posa a *Null*.
3. Per últim, el servidor traeix com a resposta un *no content* amb el codi 204 [MD-Na].

### 3.3.3 Cotxes

Mètode	Ruta	Middleware	Descripció
GET	/cars	auth	Subsubsecció 3.3.3 Mostrar tots els cotxes
POST	/cars	auth	Subsubsecció 3.3.3 Crear un cotxe
GET	/cars/car	auth	Subsubsecció 3.3.3 Mostrar el cotxe
PUT	/cars/car	auth	Subsubsecció 3.3.3 Modificar el cotxe
DELETE	/cars/car	auth	Subsubsecció 3.3.3 Eliminar el cotxe

Taula 3.9 – Taula rest API *Cars*

El *middleware auth*, vol dir que es passa el *Authorization header* [Lard].

#### Mostrar tots els cotxes

Aquesta petició HTTP mostra els diferents cotxes que té l'usuari registrats. En el següent exemple mostra el *body* en format JSON de la resposta d'aquesta petició HTTP quan l'usuari té registrats dos cotxes.

```
{
  0: {
    "id": 3
    "is_parked": false
    "matricula": "1234ABC"
  }
  1: {
    "id": 5
    "is_parked": false
    "matricula": "1234ABB"
  }
}
```

En el següent exemple mostra el *body* en format JSON de la resposta d'aquesta petició HTTP quan l'usuari té registrat un cotxe i aquest es troba dins del pàrquing. Com es pot veure, a part de la informació del cotxe també ens informa de la plaça que està assignada al cotxe.

```
{
  "id": 3
  "is_parked": true
  "matricula": "1234ABC"
  "space": {
```

```

    "created_at": "2022-04-27T15:39:07.000000Z"
    "id": 2
    "updated_at": "2022-04-27T15:39:07.000000Z"
}
}

```

### Crear un cotxe

Nom	Tipus	Descripció
matrícula	string	required, única, un total de 7 caràcters

Taula 3.10 – Taula regles de validació rest API *Cars*

Un exemple del *body* d'una petició HTTP per crear un cotxe relacionat a un usuari és enviar en el següent JSON:

```
{
  "matricula": "1234ABC"
}
```

La resposta del servidor a la petició HTTP anterior és donar la informació del cotxe que s'ha creat. Seguint l'exemple anterior, la resposta en format JSON és la següent:

```
{
  "id": 4
  "is_parked": false
  "matricula": "1234ABC"
}
```

### Mostrar el cotxe

Un exemple del que pot ser aquesta petició HTTP per tal de mostrar un cotxe específic de l'usuari és passar-li com a paràmetre de ruta la id del cotxe. Un exemple de API és: `/cars/3`.

La resposta amb JSON del servidor posterior d'enviar el passat exemple és amb la informació del cotxe amb la id 3. Es pot observar dos casos diferents, el primer quan el cotxe no es troba dins del pàrquing i el segon quan si s'hi troba, el qual ens mostra també la informació de la plaça associada al cotxe.

```
{
  "id": 3
  "is_parked": false
}
```

```

    "matricula": "1234ABC"
}
{
  "id": 3
  "is_parked": true
  "matricula": "1234ABC"
  "space": {
    "id": 2
    "created_at": "2022-04-27T15:39:07.000000Z"
    "updated_at": "2022-04-27T15:39:07.000000Z"
  }
}

```

## Modificar el cotxe

Un exemple del que pot ser aquesta petició HTTP per tal de modificar un cotxe específic de l'usuari és passar-li com a paràmetre de ruta la id del cotxe. Un exemple de API és: `/cars/3`. Addicionalment s'ha de passar el nou paràmetre modificat del cotxe, en aquest cas el número de matrícula on aquesta dada enviada a la sol·licitud també s'ha de validar següint aquesta taula Taula 3.10.

Seguint l'exemple de l'API, en el *body* de la petició HTTP en format JSON és el següent:

```
{
  "matricula": "1234ABC"
}
```

Com a exemple de la resposta del servidor a modificar el número de matrícula del cotxe amb la id 3 és donar la informació del cotxe. El qual és el següent:

```
{
  "id": 3
  "matricula": "1234ABA"
  "is_parked": false
}
```

## Eliminar el cotxe

Un exemple del que pot ser aquesta petició HTTP per tal d'eliminar un cotxe específic de l'usuari és passar-li com a paràmetre de ruta la id del cotxe.

Abans de fer aquesta petició per tal d'eliminar el cotxe amb id 4, per mostrar el seu funcionament fem ús la primera API Subsubsecció 3.3.3 per veure tots els cotxes del usuari.

```
{
  0: {"id": 4, "matricula": "1234ABC", "is_parked": false}
  1: {"id": 5, "matricula": "1234ABA", "is_parked": false}
}
```

Un exemple d'aquesta API és: `/cars/4`, en aquest cas s'elimina el cotxe amb ID 4. Desde el servidor no s'obté resposta.

Per veure el seu funcionament utilitzant la petició HTTP per mostrar els cotxes de l'usuari vist a Subsubsecció 3.3.3, seguit l'exemple s'obté aquest *body* en format JSON:

```
{
  0: {"id": 5, "matricula": "1234ABA", "is_parked": false}
}
```

### 3.3.4 QRs

Mètode	Ruta	Middleware	Descripció
GET	<code>/cars/car/qr</code>	auth	Subsubsecció 3.3.4 Generar QR

Taula 3.11 – Taula rest API *QR*

El *middleware* auth, vol dir que es passa el *Authorization header* [Larf].

#### Generar QR

Per realitzar aquesta acció és necessari crear un controlador únicament per aquesta API. Es pot tenir més informació llegint la documentació a [Larf].

Aquesta petició HTTP es porta a terme únicament si el cotxe no es troba a dins del pàrquing. Si és a dins, és a dir si el cotxe està aparcat, el servidor llança una `HttpException`, un *abort* de codi 403 [MDNb].

Si es porta a terme, aquesta petició actualitza dues columnes de la base de dades de *Cars*. Com a resposta el servidor retorna un *no content* amb el codi 204 [MDNa].

Els paràmetres que actualitza de la base de dades de *Cars* són:

- El *token*: Nombre aleatori de 16 caràcters.
- El *token\_created\_at*: La data en què s'ha creat el token anterior.

### 3.3.5 Pagament

Per poder fer el pagament s'ha utilitzat *Stripe*, un sistema de pagament en línia pensat per ser integrat a la mateixa pàgina web. A més a més, aquest sistema no envia el comprador a una altra web externa per finalitzar el pagament. [Strb]

Mètode	Ruta	Middleware	Descripció
POST	/checkout/car	auth	Subsubsecció 3.3.5 Generar pagament
GET	/checkout/car/error	-	Subsubsecció 3.3.5 Pagament cancel·lat
GET	/checkout/car/success	-	Subsubsecció 3.3.5 Pagament acceptat

Taula 3.12 – Taula rest API *Payments*

El *middleware auth*, vol dir que es passa el *Authorization header* [Lard].

#### Generar pagament

S'utilitza aquesta petició HTTP quan l'usuari vol treure el cotxe del pàrquing, és a dir sortir d'aquest. En aquesta petició l'usuari ha passat dos atributs que han de ser validats per el servidor. Aquestes validacions han de passar les següents regles:

Nom	Tipus	Descripció
success_url	string	required, url
cancel_url	string	required, url

Taula 3.13 – Taula regles de validació rest API *Payments*

Un cop aquests atributs són validats el servidor obté la informació del preu total calculant-la a travès de la taula de la base de dades de *Spaces*. Com a norma general sempre es cobra un mínim d'1 € al client i s'arrodoneix aquest pagament a l'alça.

#### Pagament cancel·lat

Petició HTTP que redirigeix al client a l'URL que s'ha passat a Subsubsecció 3.3.5 Generar pagament si el pagament ha estat cancel·lat.

#### Pagament acceptat

Quan es defineix la *success\_url*, es pot indicar a Stripe que afegeixi l'ID de sessió de pagament com a paràmetre de la consulta quan s'invoca l'URL [Larj].

Aquesta petició HTTP agafa el SDK del client de *stripe*, obté del *body* la ID del cotxe de l'usuari que vol sortir del pàrquing. El servidor també actualitza el *token* i el *token\_created\_at* de la taula *Cars* de la base de dades.

Crea una fila a la taula *Logs* de la base de dades on guarda:

- A la columna *state* el valor de pagament (*Payment*).
- A la columna *price* es posa el preu total en cèntims que el servidor ha calculat.
- A la columna *payment\_id* es posa la sessió de pagament.

### 3.3.6 Configuracions

Mètode	Ruta	Middleware	Descripció
GET	/settings	auth	Subsubsecció 3.3.6 Obtenir settings
PUT	/settings	admin	Subsubsecció 3.3.6 Modificar settings

Taula 3.14 – Taula rest API *Settings*

El *middleware auth*, vol dir que es passa el *Authorization header*. En el cas del *middleware admin* comprova si l'usuari és administrador [Lard].

#### Obtenir settings

Dóna com a resposta els atributs de la base de dades *settings*. El preu per hores en cèntims (*price\_hour*) i l'expiració del *token* en minuts (*token\_expiration*).

Per defecte *Settings* està configurat de la següent manera:

```
{
  "price_hour": 270
  "token_expiration": 15
}
```

#### Modificar settings

Per filtrar les sol·licituds HTTP s'utilitza els *middleware* [Lard]. Com s'ha indicat abans a la taula, per ser utilitzada aquesta API és necessari usar el *middleware* d'administració.

En la petició HTTP com a *body* es passen uns paràmetres, els quals han de ser validats. Aquestes regles de validació són:

Nom	Tipus	Descripció
price_hour	integer	required, numeric, mínim: 0
token_expiration	integer	required, numeric, mínim: 5, màxim 60

Taula 3.15 – Taula regles de validació rest API *Settings*

Fent un exemple d'aquesta petició HTTP indicant el *body* en format JSON és:

```
{
  "price_hour": 260
  "token_expiration": 5
}
```

Com a resposta del servidor s'obté el mateix *body* en format JSON, ja que retorna la informació de *Settings*.

## 3.4 Tests

Els tests són molt importants en una aplicació. Ajuden a trobar errors i a comprovar el correcte funcionament de l'aplicació. Fent ús dels tests que Laravel dóna com a eina, es poden fer testos a peticions HTTP completes. En aquest projecte s'utilitzen els tests d'integracions. Són aquells que comproven la funcionalitat de diferents mòduls en un conjunt [Larg].

L'aplicació té quatre blocs de tests d'integracions, els quals són els tests d'autenticació, de cotxes, de *logs* i de *settings*.

### 3.4.1 Test *AuthTest*

Els tests d'autenticació són els que fan proves sobre el registre d'usuaris, iniciar sessió, tancar sessió, poder agafar la informació de l'usuari i editar l'usuari.

### 3.4.2 Test *CarsTest*

Els tests de cotxes són aquells que comproven el seu funcionament, és a dir, comprova que es puguin veure tots els cotxes d'un usuari, crear cotxes nous, que no es puguin crear cotxes amb la matrícula repetida, veure la informació d'un cotxe, actualitzar els cotxes, és a dir modificar el número de matrícula i per últim eliminar un cotxe.

### 3.4.3 Test *LogsTest*

En els tests dels *Logs* únicament n'hi ha dos i serveixen per obtenir els logs, ja que es comprova que l'usuari és administrador o no. Si ho és, pot veure els logs, si no ho és, retorna un error 403 en HTTP [MDNb]. El lloc on es creen els logs són en el *payment controller* i en el *Barrier Controller*, per tant, si es volgués fer tests d'aquesta acció està en els tests d'aquests controladors.

### 3.4.4 Test *SettingsTest*

Per últim, els tests de *settings*. L'aplicació disposa de tres tests, els quals mira si l'usuari és administrador, ja que pot modificar els *settings*. En canvi si l'usuari no ho és no podrà fer aquesta acció retornant un error 403 en HTTP [MDNb]. En tots dos casos l'usuari pot veure aquests *settings*.

En la següent imatge es pot veure com han passat tots els testos:

```

Parking git:(main) ✘ php artisan test
  ✓ Tests\Feature\API\CarsTest
    ✓ it can show all cars
    ✓ it can create cars
    ✓ it fails to create car with same matricula
    ✓ it can show cars
    ✓ it can update cars
    ✓ it can delete cars

  ✓ Tests\Feature\API\LogsTest
    ✓ it can get logs
    ✓ it cant get logs

  ✓ Tests\Feature\API\SettingsTest
    ✓ it can update settings
    ✓ it can not update settings
    ✓ it can read settings

  ✓ Tests\Feature\AuthTest
    ✓ it can register users
    ✓ it can login users
    ✓ it can get user
    ✓ it can logout users
    ✓ it can edit users

Tests: 16 passed
Time: 0.46s

```

Figura 3.2 – Tests Correctes

# 4 Front-end

La implementació del *front-end* de l'aplicació, és a dir la part que l'usuari interactua amb ella s'ha usat la *framework* de *NextJS* [Vera] basada en React.

## 4.1 Elecció de l'entorn

*React*, és una llibreria de codi obert de JavaScript amb l'objectiu de desenvolupar interfícies d'usuari. ReactJS permet als desenvolupadors crear aplicacions que fan servir dades que poden canviar amb el temps sense la necessitat de recarregar la pàgina [Jor]. Les seves grans característiques són la rapidesa, simplicitat i l'escalabilitat. Aquests objectius s'aconsegueixen gràcies a:

- *DOM Virtual Document Object Model*: és el que permet renderitzar només els elements gràfics que han estat modificats, això aporta una gran velocitat. És útil quan es té moltes dades i hi ha petites modificacions.
- *JSX*: és una extensió del llenguatge de programació JavaScript optimitzat en velocitat i captura errors en temps de compilació. L'usuari té la possibilitat d'utilitzar-la perquè ajuda al desenvolupador, ja que té una sintaxi semblant a la de l'HTML, això implica que fa el codi més llegible i escriure'l és una experiència similar a l'HTML, el fa més fàcil.
- Les propietats conegudes com les *props*, són els atributs de configuració d'un component.

Existeixen algunes *frameworks* que utilitzen la llibreria de *React*, com per exemple, *NextJS* utilitza tot el que dóna la llibreria de *React*. A més, dóna un sistema de rutes amb la separació de codi per ruta. Permet en la utilització d'unes que ajuden al desenvolupador. Algunes d'elles utilitzades són:

- *Router*, per l'encaminament i la navegació.
- *Link*, un component que permet que l'aplicació enllaci pàgines i carregui les seves dades.

## 4.2 Pàgines

En els annexos s'hi pot trobar un recull de fotografies per mostrar com és la part de *front-end*. Pel disseny de l'aplicació s'utilitza la *framework* de CSS Tailwind. Usant aquesta eines permet al desenvolupador evitar escriure arxius CSS [Ada].

### 4.2.1 Iniciar Sessió

En aquesta vista permet a l'usuari iniciar sessió posant el seu correu electrònic i la contrasenya en els *inputs* que es veuen a la imatge Figura A.1. Seguidament, l'usuari ha de prémer el botó d'**Iniciar sessió**. Si l'usuari no està registrat en aquesta aplicació, ho pot fer prement el botó de *Crear un nou compte* que està sota del primer botó.

### 4.2.2 Registrar usuari

Aquesta vista mostra uns *inputs* per incloure les dades de l'usuari per poder-se registrar a l'aplicació. Aquests són el nom, el DNI, el telèfon mòbil, el correu electrònic, la contrasenya i la confirmació de la contrasenya. Una vegada inscrites aquestes dades s'ha de prémer el botó de *Registrar-se*, si es vol tirar enrere i anar a la pàgina d'iniciar sessió es pot clicar el botó de *Ja tinc compte* o prémer el *logo* de la *navbar*. La imatge relacionada aquesta vista es pot trobar a Figura A.2.

### 4.2.3 Principal

La imatge que mostra un exemple d'aquesta vista està a Figura A.3. En la fotografia es pot veure tres seccions.

La primera és que dóna la benvinguda a l'usuari i hi ha un botó que únicament es pot clicar quan està disponible. La disponibilitat varia segons si l'usuari és administrador. La imatge corresponent a la part d'administradors està a Figura A.12.

La segona, hi ha una *card* on hi ha la informació de l'usuari, a la part superior a la dreta hi ha una icona d'un llapis. Aquest llapis és un botó per editar aquesta informació. Quan l'usuari el clica, l'aplicació redirigeix a l'usuari a una pantalla per editar-la. La pantalla d'editar està a Figura A.4.

Finalment, es troba la tercera secció on també hi ha una *card* on hi ha la informació dels vehicles registrats de l'usuari. A la part superior d'aquesta *card* hi ha un botó amb un +, si el client clica aquest botó, l'aplicació el redirigeix a una pantalla per afegir vehicles, una imatge d'aquesta vista està a Figura A.6.

Quan hi ha vehicles guardats es mostra el número de matrícula i tres botons. Aquests botons tenen tres accions diferents, el primer canvia depenent d'on està el cotxe. Si el cotxe no està dins del pàrquing, el botó és un codi QR, ja que per entrar al pàrquing

s'ha de clicar. Es pot veure aquesta pantalla a Figura A.7. En canvi, si el cotxe està dins del pàrquing la vista mostra una P de pàrquing, on hi ha la informació relacionada d'aquest aparcament. Un exemple d'aquesta pantalla és Figura A.8. El segon botó és un llapis, s'utilitza per editar el número de la matrícula.

La vista que controla aquesta funcionalitat és a Figura A.5. Per acabar, hi ha un botó d'una brossa de color vermell. Aquest botó és per eliminar el vehicle. Els botons d'editar i eliminar es deshabiliten quan el vehicle està estacionat dins del pàrquing.

### Vista per editar l'usuari

En aquesta vista l'usuari pot modificar les dades del seu compte com el nom, la contrasenya, el telèfon i el DNI. Es troben dos botons al final de la *card* on el primer és per tirar enrere cap a la pantalla d'índex Figura A.3, també es pot anar a la pantalla principal clicant el *logo* de la *navbar*. El segon botó és per guardar les noves dades a la base de dades. La imatge relacionada a aquesta vista és Figura A.4.

### Vista per editar els cotxes de l'usuari

La imatge que mostra aquesta pantalla és Figura A.5. Es veu una *card* on només hi ha un *input* per modificar el número de la matrícula. El format pel número de la matrícula és *xxxxXXX*, on els quatre primers caràcters són números del 0-9 i els tres caràcters del final són lletres de l'A-Z. La matrícula és única per cada usuari, és a dir, no es pot repetir.

### Vista per afegir els cotxes de l'usuari

Aquesta vista és molt similar a l'anterior, ja que únicament canvia el títol i l'acció resultant, és a dir afegir un cotxe a la base de dades amb l'usuari que ha iniciat sessió. La següent imatge mostra aquesta vista Figura A.6.

## 4.2.4 Codi QR

Quan l'usuari es troba a la pàgina principal i vol accedir amb un vehicle al pàrquing, clica el botó del codi QR que pertany en aquest vehicle. Aquest botó redirigeix a aquesta pantalla Figura A.7. Es veu una *card* on a dins hi ha el codi QR generat per aquell vehicle. A sota d'aquest codi QR hi ha un botó, aquest botó serveix per generar un altre codi QR. Aquest botó s'utilitza quan el codi QR generat no és vàlid i es necessita crear-ne un altre per accedir dins del pàrquing.

A sota hi ha dos botons, el botó d'*Enrere* per anar a la pantalla d'inici, també es pot usar el *logo* de la *navbar* que redirigeix a la mateixa pàgina principal. A la dreta es pot veure un botó on diu *Informació* que s'activa quan el codi QR ha estat validat pel servidor i es pot entrar al pàrquing. En aquesta es veu la informació de l'estacionament. Aquest

botó, en principi no se'n fa ús, ja que el servidor envia un esdeveniment a l'aplicació a temps real i redirigeix a la pàgina d'informació. Es pot veure un exemple a Figura A.8. Tot i això, aquest botó és útil per si hi ha algun problema que impedís l'enviament d'aquest esdeveniment. D'aquesta manera l'aplicació es pot continuar fent servir.

### 4.2.5 Informativa

Aquesta pantalla ens mostra la informació de quina plaça té associada el cotxe, el dia i hora que ha entrat al pàrquing, el preu en hora, la data actual i el preu en temps real. És una vista informativa. Quan es vol sortir del pàrquing s'ha de prémer el botó de sortir que hi ha al final de la *card* on es redirigeix a la pantalla de pagament que es pot veure a Figura A.9. La següent imatge és un exemple del que pot mostrar aquesta pantalla informativa Figura A.8.

### 4.2.6 Pagament

Un cop s'ha pres el botó de sortir de la pantalla anterior es redirigeix a una pantalla de *stripe* per fer el pagament. Per simular un pagament amb èxit existeixen una llista de targetes per fer *testing*. Aquesta llista es pot trobar a [Stra]. Un exemple d'aquesta pantalla de pagament de *Stripe* [Strb] està a Figura A.9.

#### Pagament validat

La pantalla de quan el pagament és validat consta d'un *tick* de color verd per mostrar que tot ha funcionat correctament. Un codi QR generat per sortir del pàrquing amb un text explicatiu per saber que ha anat tot correctament i de quant temps es disposa per la validesa d'aquest codi QR. Si aquests minuts s'accedeixen, s'ha de tornar a pagar. Disposa d'un botó per anar a la pàgina principal, com també si es clica el *logo* de la *navbar*. La vista es pot veure a Figura A.10.

#### Pagament cancel·lat

En aquest cas, el pagament ha estat cancel·lat per l'usuari. La vista és una creu de color vermell que indica que hi ha hagut un error, a més una descripció posa que el pagament no s'ha efectuat. Hi ha un botó per anar a la pàgina principal, com també si es clica el *logo* de la *navbar*. La imatge d'aquesta vista es pot veure a Figura A.11

### 4.2.7 Panell administració

En aquest panell d'administració només hi pot accedir els usuaris que són *admin*. Els altres usuaris no hi poden accedir i a més tenen el botó deshabilitat. Com es pot observar en la imatge d'aquesta vista a Figura A.12 hi ha dues *cards*. La primera l'administrador

pot modificar el preu en hores del pàrquing, s'ha de posar en cèntims i també pot modificar el temps de validesa dels codis QR. La segona hi ha una taula on es mostren tots els *logs*, és a dir, totes les entrades, sortides i pagaments efectuats al pàrquing. A la columna de *pagament* hi ha un enllaç cap a *stripe* on hi ha la informació del pagament. En aquesta pantalla es pot fer accions com descarregar les factures, tramitar devolucions, veure detalls de la transacció i client [Strb]. La imatge es pot veure a Figura A.13.

## 4.3 Eines utilitzades

En aquesta aplicació es fan servir diferents eines importants per fer un correcte funcionament. La primera de totes, és *Axios*. Aquesta eina permet fer peticions HTTP passant-li en els *Authorization Header* el *Bearer Token* de l'usuari. El fet aquest es pot configurar de manera automàtica si s'activa les *withCredentials* d'axios o es pot passar manual.

Algunes altres eines són:

- `classnames` [jed]: Una senzilla utilitat de JavaScript per unir condicionalment class-Names.
- `dayjs` [day]: Una llibreria de JavaScript de dates.
- `pusher-js` [Pus]: Els canals de Pusher són una solució als Web Sockets. Eina útil per crear aplicacions interactives en temps real.
- `react-qr-codes` [ros]: Un component de React per crear codis QR.
- `swr` [Verb]: stale-while-revalidate. Retornar les dades de la memòria, enviar una nova sollicitud i revalidar les dades.



# 5 Raspberry Pi

La part de la barrera del pàrquing està composta per una Raspberry Pi 4 model B, una càmera HQ, un LED RGB que simbolitzen les accions de la barrera, tres resistències de 470 Ohms per fer les connexions amb el LED i per últim un brunzidor elèctric.

## 5.0.1 Connexions

Aquests components han d'estar connectats a la Raspberry Pi, seguidament es mostra en els pins que s'han connectat cada un dels LEDs i el brunzidor. Adjuntant un esquema dels pins de la Raspberry Pi per veure quins són:

- LED Red: GPIO 18
- LED Blue: GPIO 23
- LED Green: GPIO 24
- Brunzidor: GPIO 25



Figura 5.1 – Raspberry PI 4 pinout

Finalment, tots els components utilitzats amb el seu muntatge són els següents:

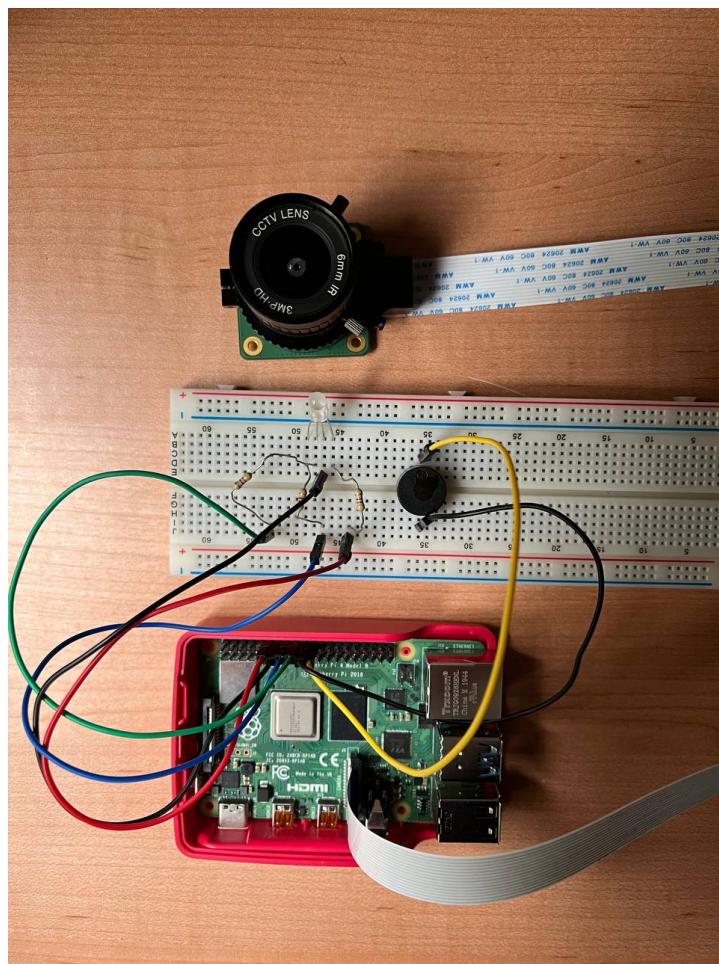


Figura 5.2 – Muntatge Raspberry Pi

### 5.0.2 Màquina d'estats

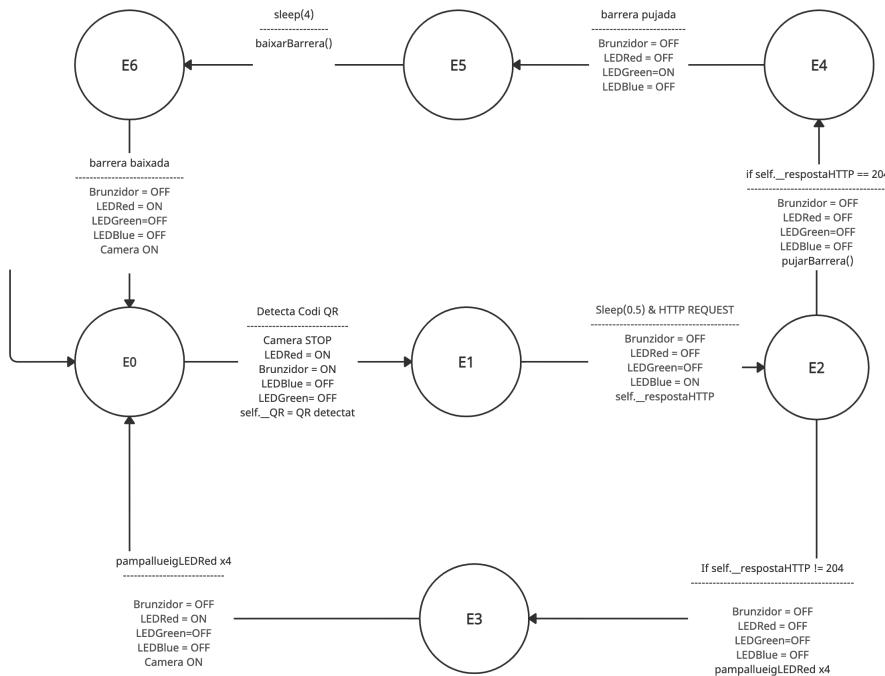


Figura 5.3 – Màquina d'estat detecció codi QR

- Estat 0:** Indica que la barrera està baixada posant el LED de color vermell a *ON*. Obra la càmera connectada a la Raspberry Pi on es prepara per detectar codis QR. Quan en detecta un llegeix el seu contingut i el guarda a la variable anomenada *qr*. Deshabilita la càmera, d'aquesta manera no pot detectar més codis QR. Quan detecta el codi QR s'activa el brunzidor que dura cinc segons i canvia al següent estat, l'estat 1.
- Estat 1:** Es deshabilita el brunzidor. Apaga el LED vermell i posa a *ON* el LED blau, ja que aquest LED indica que està fent la petició HTTP. Es guarda la resposta d'aquesta petició a una variable anomenada *respostaHTTP*. Canvia al següent estat, l'estat 2.
- Estat 2:** Apaga el LED blau, i comprova la resposta HTTP. Si la resposta és 204, canvia l'estat a Estat 4. Si no ho és, canvia l'estat a l'estat 3.
- Estat 3:** Per indicar a l'usuari que la resposta no ha sigut satisfactòria, el LED vermell farà quatre pampallugues. Finalment, l'estat passa a ser l'estat inicial, l'estat 0.
- Estat 4:** Aquest estat és quan la resposta HTTP ha sigut satisfactòria. En aquest estat el LED Verd farà quatre pampallugues indicant que la barrera està pujant. Quan la barrera ha pujat, l'estat passa a ser l'estat 5.

6. **Estat 5:** Posa el LED verd a *ON*. Simbolitza la barrera pujada. Quan ha passat el temps corresponent canvia l'estat a l'estat 6.
7. **Estat 6:** Aquest estat es comporta igual que l'estat 4. El LED Verd fa quatre pampallugues per indicar que la barrera està baixant. Quan es troba a abaixada passa a l'estat inicial. L'estat 0.

## Llibreries

Les llibreries utilitzades per fer la màquina d'estats anteriors són les següents:

- Requests: per fer peticions HTTP [Com].
- Enum: Per declarar els estats [Pyt].
- Time: Per fer la durada. S'utilitza la funció *sleep()*. Com que el codi no fa res més no passa res que sigui bloquejant [Pro]
- RPi.GPIO: Per configurar els inputs de la Raspberry Pi [Ben].
- cv2: Per obrir la càmera [Oll].
- Pyzbar: Detectar codis QR, llegir-los i descodificar-los [Law].

# 6 Connexions de hardware

El projecte està compost per tres conjunts, el servidor de Laravel, l'aplicació amb què l'usuari interactua i la barrera. Pel desenvolupament d'aquest projecte s'ha organitzat de la manera que el servidor i l'aplicació estan a un ordinador i la barrera està en una Raspberry Pi.

L'esquema que mostra com s'ha treballat durant el seu desenvolupament és el següent:

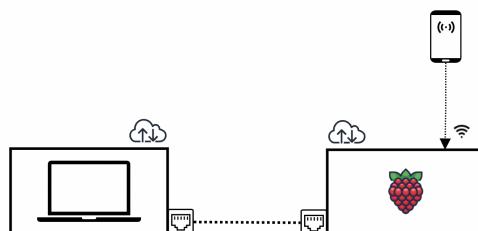


Figura 6.1 – Connexions dels dispositius

- El telèfon mòbil crea un punt d'accés personal.
- La Raspberry Pi i l'ordinador estan connectats a través d'un cable Ethernet.
- La Raspberry Pi està connectada al punt d'accés personal del telèfon mòbil i proporciona internet a l'ordinador.

Per poder aconseguir aquesta configuració a la barrera, concretament a la Raspberry Pi, se li ha assignat una adreça IP a `eth0`, la qual és 10.10.10.10. A l'ordinador, es configura l'adreça 10.10.10.1 amb l'encaminador 10.10.10.10. D'aquesta manera hi ha la possibilitat de connectar-se amb la Raspberry Pi via `ssh` fent `ssh pi@10.10.10.10` des d'un terminal de l'ordinador. D'aquesta manera es pot utilitzar la Raspberry Pi des del mateix ordinador sense la necessitat de connectar una pantalla externa i perifèrics com el ratolí i el teclat. Tot i que al principi s'accedia a la Raspberry Pi a través d'una pantalla per tal de configurar la càmera i fer les configuracions anteriors de l'esquema Figura 6.1.

Pel projecte es necessita que la barrera tingui connexió a internet, ja que fa peticions HTTP. També es vol que el servidor i l'aplicació tinguin internet, perquè si no no es poden fer ús dels esdeveniments de Web Sockets que utilitzen i altres eines o llibreries que se'n fa ús.

La solució d'aquest problema que es planteja és configurar la Raspberry Pi de tal manera que l'ordinador tingui accés a internet a través d'ella, ja que és qui li proporciona internet des de la *wlan0*. Per aconseguir aquest efecte es fa servir la comanda següent: `iptables -t nat -A POSTROUTING -s 10.10.10.0/24 -o wlan0 -j MASQUERADE`. Per fer això persistent i no posar la comanda cada vegada que es vol provar el funcionament del projecte cal modificar el fitxer d'*/etc/iptables/rules.v4* posant la comanda següent a l'apartat de *nat*:

```
-A POSTROUTING -s 10.10.10.0/24 -o wlan0 -j MASQUERADE
```

Aquesta comanda és afegir a la taula *nat* la regla *POSTROUTING*, aquesta regla consisteix a alterar o fer una determinada acció als paquets quan estan a punt de sortir on la font és l'adreça 10.10.10.0/24 i s'envien els paquets a través de la interfície *wlan0* cap a l'objectiu *MASQUERADE*. Aquesta, s'utilitza per IP dinàmiques, la qual el telèfon mòbil equival a una d'elles. Per més informació d'aquesta comanda es pot trobar a [Rus]. A més, configurar un servidor DNS a l'ordinador, el DNS 8.8.8.8.

# 7 Desglossament econòmic de la demo

Nom	Quantitat	Preu
RASPBERRY PI 4 MODEL B 4GB	1	59,95 €
CAMARA HQ OFICIAL RASPBERRY PI	1	55,95 €
CAJA OFICIAL RASPBERRY PI 4	1	6,95 €
CABLE MICRO HDMI 1M.		
NEGRE OFICIAL RASPBERRY PI	1	5,95 €
SOFTWARE NOOBS PREINSTAL·LAT EN MICROSD 16GB		
PARA RASPBERRY PI	1	7,95 €
LENT FIXA 6MM MUNTURA CS	1	28,95 €
<b>TOTAL</b>	<b>6</b>	<b>165,7€</b>

Taula 7.1 – Preus material necessari per a la demostració

## 7.1 Material necessari per barrera

- Placa Raspberry Pi 4 4GB [Rasa].
- Raspberry Pi High Quality Camera [Rasb].
- Lent fixa 6mm monturs CS [tie].
- LED RGB (tret del laboratori de DIPSE).
- Tres resistències de 470 Ohms (tret del laboratori de DIPSE).
- Brunzidor (tret del laboratori de DIPSE).



# 8 Conclusions

Pel que fa a la realització d'aquest TFG, s'ha assolit el propòsit principal que és realitzar una aplicació (una *demo*) per la gestió automatitzada d'un pàrquing intel·ligent amb el seu control d'entrades i sortides, amb la part de pagament i un panell administratiu.

Per dur a terme aquest projecte s'ha trobat algunes dificultats, les quals són:

Primer de tot, amb la part de *back-end*:

- Crear la composició del codi QR i buscar la forma de què aquests codis siguin vàlids.
- Com ha d'actuar el servidor quan rep la lectura d'un codi QR i les accions que ha de prendre respecte si és vàlid o no.
- En el sistema de pagament, la dificultat és passar-li *metadata* per obtenir la informació de qui havia fet el pagament evitant passar les dades per l'URL. D'aquesta manera s'evita que l'usuari pugui modificar-la.
- Decidir com associar les places als usuaris, és a dir de donar la possibilitat a l'usuari a escollir la plaça desitjada o que l'aplicació faci aquesta assignació.

Per l'apartat de *front-end* la utilització dels *hooks* que disposa la llibreria de *React*. Aquests són el *UseCallback*, *useMemo*, *useState* i el *useContext*. L'explicació d'aquestes eines es poden trobar a [Rea].

Finalment, per l'apartat de la *Raspberry Pi*:

- Buscar eines o llibreries per obrir la càmera de la Raspberry pi.
- Buscar llibreries per la detecció als codis QR i la lectura d'aquests.
- Buscar una eina per fer peticions HTTP.
- Realitzar una màquina d'estats més robusta i que contempli possibles casos d'error, com el d'invalidesa del codi QR.
- Buscar components electrònics per tal de poder fer una simulació d'una barrera de pàrquing.

Finalment, treballar totes aquestes aplicacions conjuntament i poder fer una demostració de la seva funcionalitat ha comportat problemes, a causa de la necessitat de controlar la Raspberry Pi per *ssh* i que tots dos conjunts obtinguessin internet.

## 8.1 Treball futur

Després de realitzar una aplicació on els seus objectius s'han assolit és moment de pensar amb possibles millors o treballs addicionals per poder millorar aquest projecte. Algunes d'ells són:

1. Afegir o crear un control per part d'administració que pugui pujar o baixar la barrera depenent de les necessitats o problemes que puguin succeir. Exceptuant una fallada de motor. Per aconseguir aquest efecte és fer que l'actual servidor, és a dir, *Laravel* (o *back-end*), passi a ser un client i la Raspberry Pi passi a ser un servidor de *Python*. Fer una connexió bidireccional. A més a més, crear una taula a la base de dades on es guarda la ID i l'estat de la barrera.
2. Afegir un espai on els usuaris puguin canviar la plaça assignada i puguin escollir la plaça desitjada. En l'aplicació fer que els usuaris vegin un mapa del parc que mostra les places ocupades i lliures en temps real.
3. Posar la possibilitat de fer reserves de places a través de l'aplicació.
4. Fins ara, el control d'entrades, sortides i pagaments la veuen els administradors. Afegir una pantalla on els clients puguin veure aquesta informació i descarregar-se la factura.
5. Configurar el servidor de *Laravel* per enviar correus als clients adjuntant la factura de cada pagament.

# 9 Bibliografia

- [Ada] Adam Wathan, Steve Schoger. *Tailwind CSS-A utility-first CSS framework*. English. [Online; visitat 7-Desembre-2021]. URL: <https://tailwindcss.com/>.
- [Ast] Astrit. *Open-source CSS, SVG and Figma UI Icons*. English. [Online; visitat Març-2021]. URL: <https://css.gg/app>.
- [Axi] Axios. *Axios*. English. [Online; visitat 22-Març-2021]. URL: <https://github.com/axios/axios>.
- [Ben] Ben Croston. *RPi.GPIO*. English. [Online; visitat 2-Maig-2022]. URL: <https://pypi.org/project/RPi.GPIO/>.
- [Com] Community GitHub. *Requests: HTTP for Humans*. English. [Online; visitat 2-Maig-2022]. URL: <https://docs.python-requests.org/en/latest/>.
- [day] dayjs. *Day.js - JavaScript date utility library*. English. [Online; visitat 3-Maig-2022]. URL: <https://day.js.org/>.
- [El] El Parking. *El Parking*. Català. [Online; visitat 2-Maig-2022]. URL: <https://elparking.com/ca/parkings>.
- [Enc] Enciclopedia. *Definició Autòmat finit*. Català. [Online; visitat 30-Abril-2022]. URL: <https://www.encyclopedia.cat/gran-encyclopedia-catalana/automat-finit>.
- [Fla] Flaticon authors. *Flaticon*. English. [Online; visitat 28-Octubre-2021]. URL: <https://www.flaticon.es/home>.
- [jed] jedwatson. *Classnames-A simple javascript utility for conditionally joining classNames together*. English. [Online; visitat 3-Maig-2022]. URL: <https://jedwatson.github.io/classnames/>.
- [Jor] Jordan Walke. *ReactJS-A JavaScript library for building user interfaces*. English. [Online; visitat 7-Desembre-2021]. URL: <https://reactjs.org>.
- [Lara] Laravel. *Laravel - Authentication*. English. [Online; visitat 30-Abril-2022]. URL: <https://laravel.com/docs/9.x/authentication>.
- [Larb] Laravel. *Laravel - Eloquent*. English. [Online; visitat 30-Abril-2022]. URL: <https://laravel.com/docs/9.x/eloquent>.
- [Larc] Laravel. *Laravel - Eloquent: API Resources*. English. [Online; visitat 30-Abril-2022]. URL: <https://laravel.com/docs/9.x/eloquent-resources>.

- [Lard] Laravel. *Laravel - Middleware*. English. [Online; visitat 1-Maig-2022]. URL: <https://laravel.com/docs/9.x/middleware#main-content>.
- [Lare] Laravel. *Laravel - Migrations*. English. [Online; visitat 30-Abril-2022]. URL: <https://laravel.com/docs/9.x/migrations>.
- [Larf] Laravel. *Laravel - single action controllers*. English. [Online; visitat 1-Maig-2022]. URL: <https://laravel.com/docs/9.x/controllers#single-action-controllers>.
- [Larg] Laravel. *Laravel - Testing*. English. [Online; visitat 1-Maig-2022]. URL: <https://laravel.com/docs/9.x/testing#main-content>.
- [Larh] Laravel. *Laravel - Validations*. English. [Online; visitat 30-Abril-2022]. URL: <https://laravel.com/docs/9.x/validation#quick-creating-the-controller>.
- [Lari] Laravel. *Laravel Cashier (Stripe)*. English. [Online; visitat 30-Abril-2022]. URL: <https://laravel.com/docs/9.x/billing>.
- [Larj] Laravel. *Laravel Cashier (Stripe) - Checkouts*. English. [Online; visitat 1-Maig-2022]. URL: <https://laravel.com/docs/9.x/billing#product-checkouts>.
- [Lark] Laravel. *Laravel Sanctum*. English. [Online; visitat 30-Abril-2022]. URL: <https://laravel.com/docs/9.x/sanctum>.
- [Larl] Laravel. *Laravel-The PHP Framework for Web Artisans*. English. [Online; visitat 7-Desembre-2021]. URL: <https://laravel.com/>.
- [Law] Lawrence Hudson. *pyzbar*. English. [Online; visitat 2-Maig-2022]. URL: <https://pypi.org/project/pyzbar/>.
- [MDNa] MDN Web Docs. *204 No Content*. English. [Online; visitat 1-Maig-2022]. URL: <https://developer.mozilla.org/en-US/docs/Web/HTTP>Status/204>.
- [MDNb] MDN Web Docs. *403 Forbidden*. English. [Online; visitat 1-Maig-2022]. URL: <https://developer.mozilla.org/en-US/docs/Web/HTTP>Status/403>.
- [MDNc] MDN Web Docs. *406 Not Acceptable*. English. [Online; visitat 1-Maig-2022]. URL: <https://developer.mozilla.org/en-US/docs/Web/HTTP>Status/406>.
- [Oll] Olli-Pekka Heinisuo. *opencv-python*. English. [Online; visitat 2-Maig-2022]. URL: <https://pypi.org/project/opencv-python/>.
- [Par] Parkingdoor. *Parkingdoor*. Castellà. [Online; visitat 2-Maig-2022]. URL: <https://parkingdoor.com/>.
- [Pro] Programiz. *Python sleep()*. English. [Online; visitat 2-Maig-2022]. URL: <https://www.programiz.com/python-programming/time/sleep>.
- [Pus] Pusher. *Pusher-Powering realtime experiences for mobile and web*. English. [Online; visitat 13-Abril-2021]. URL: <https://pusher.com>.

- [Pyt] Python. *enum — Support for enumerations*. English. [Online; visitat 2-Maig-2022]. URL: <https://docs.python.org/3/library/enum.html>.
- [Rasa] Raspberry Pi Foundation. *Raspberry Pi 4*. English. [Online; visitat 19-Novembre-2021]. URL: <https://www.raspberrypi.org/products/raspberry-pi-4-model-b/?resellerType=home>.
- [Rasb] Raspberry Pi Foundation. *Raspberry Pi High Quality Camera*. English. [Online; visitat 19-Novembre-2021]. URL: <https://www.raspberrypi.org/products/raspberry-pi-high-quality-camera/?resellerType=home>.
- [Rea] React. *Hooks*. English. [Online; visitat 5-Maig-2021]. URL: <https://reactjs.org/docs/hooks-reference.html>.
- [ros] rosskhanas. *React QR Code*. English. [Online; visitat 3-Maig-2022]. URL: <https://www.npmjs.com/package/react-qr-code>.
- [Rus] Rusty Russell. *Manual Iptables*. English. [Online; visitat 3-Maig-2022]. URL: <https://linux.die.net/man/8/iptables>.
- [Stra] Stripe. *Cards by brand*. English. [Online; visitat 2-Maig-2022]. URL: <https://stripe.com/docs/testing#cards>.
- [Strb] Stripe. *Stripe-Infraestructura de pagos para Internet*. English. [Online; visitat 1-Maig-2022]. URL: <https://stripe.com/es>.
- [Sup] Support "Statistics and Data". *Most Popular Backend Frameworks – 2012/2021*. English. [Online; visitat 30-Abril-2022]. URL: <https://statisticsanddata.org/data/most-popular-backend-frameworks-2012-2021/>.
- [Swe] SweetAlet2. *SweetAlert2-A POPUP BOXES*. English. [Online; visitat 2-Maig-2022]. URL: <https://sweetalert2.github.io/>.
- [Tai] Tailwind CSS. *Heroicons-SVG icons*. English. [Online; visitat 7-Desembre-2021]. URL: <https://heroicons.com/>.
- [Tel] Telpark. *Telpark*. Castellà. [Online; visitat 2-Maig-2022]. URL: <https://www.telpark.com/como-funciona/>.
- [tie] tiendatec. *LENTE FIJA 6MM MONTURA CS*. Castellà. [Online; visitat 24-Novembre-2021]. URL: <https://www.tiendatec.es/raspberry-pi/camaras/1191-lente-fija-6mm-montura-cs-para-camara-hq-raspberry-pi-8472496016551.html>.
- [Vera] Vercel. *NextJS-The React Framework for production*. English. [Online; visitat 7-Desembre-2021]. URL: <https://nextjs.org/>.
- [Verb] Vercel. *SWR (stale-while-revalidate)-React Hooks for Data Fetching*. English. [Online; visitat 22-Març-2021]. URL: <https://swr.vercel.app>.
- [Vik] Vikipèdia. *Arquitectura MVC (model, vista, controlador)*. Català. [Online; visitat 30-Abril-2022]. URL: <https://ca.wikipedia.org/wiki/Model-Vista-Controlador>.



## **Part II**

## **Annexos**



## A Images del front-end de l'aplicació

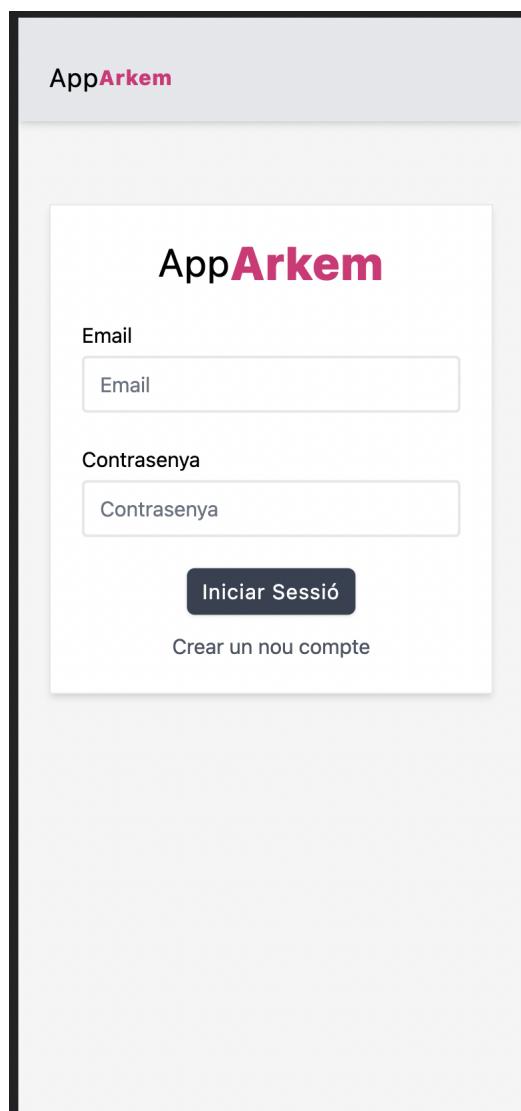


Figura A.1 – Pantalla d'iniciar sessió

The screenshot shows a registration form titled "REGISTRE D'USUARI" (User Registration) from the "AppArkem" application. The form consists of several input fields:

- Nom**: Input field labeled "Nom".
- Dni**: Input field labeled "Dni".
- Telèfon**: Input field labeled "Telèfon".
- E-mail**: Input field labeled "E-mail".
- Contrasenya**: Input field labeled "Contrasenya".
- Confirmar Contrasenya**: Input field labeled "Confirmar Contrasenya".

Below the input fields are two buttons:

- A dark blue button labeled "Registrar-se" (Register).
- A light blue button labeled "Ja tinc compte" (I have an account).

Figura A.2 – Pantalla de registrar usuaris

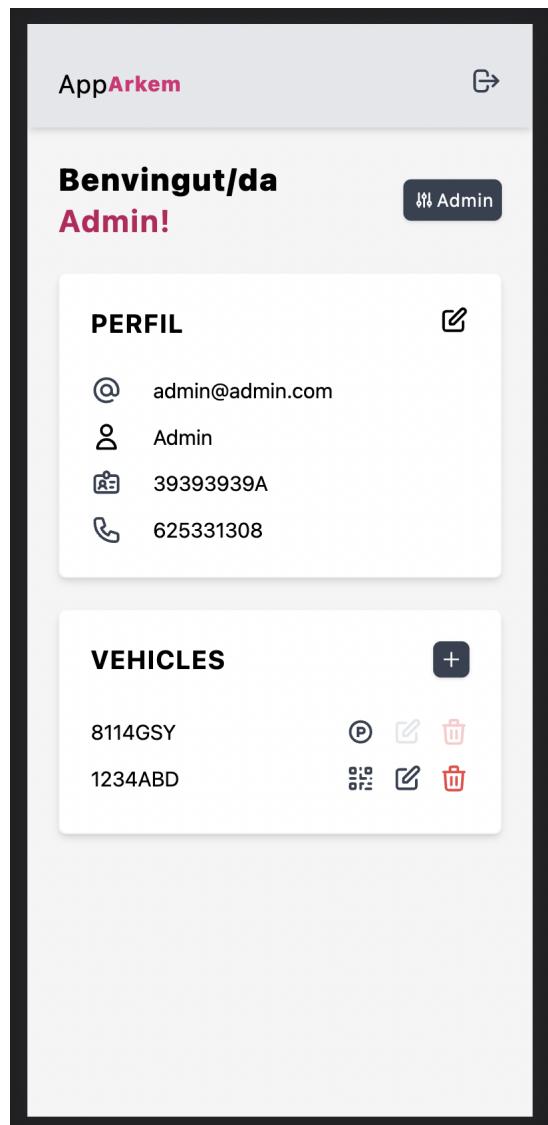


Figura A.3 – Pantalla principal

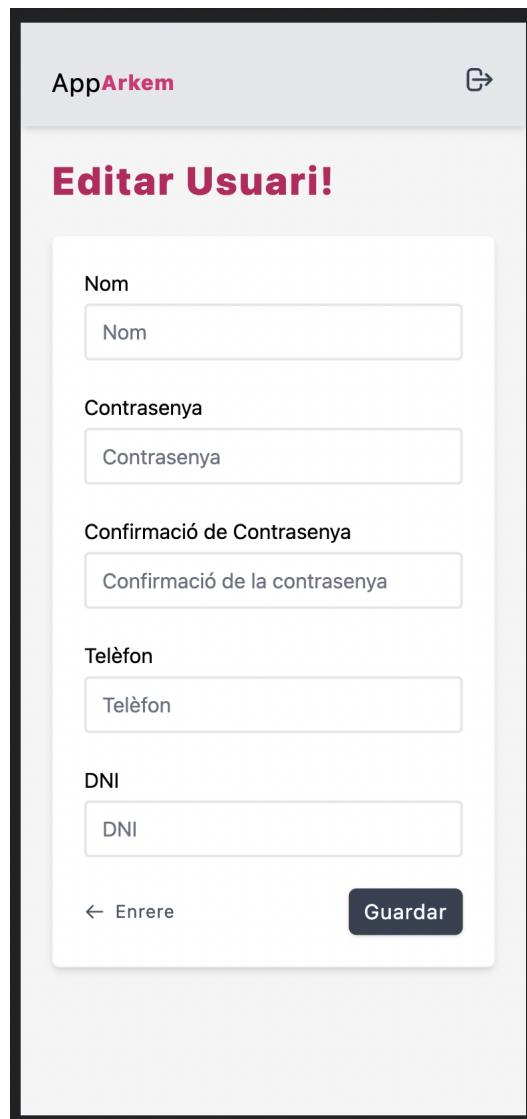


Figura A.4 – Pantalla d'editar informació de l'usuari

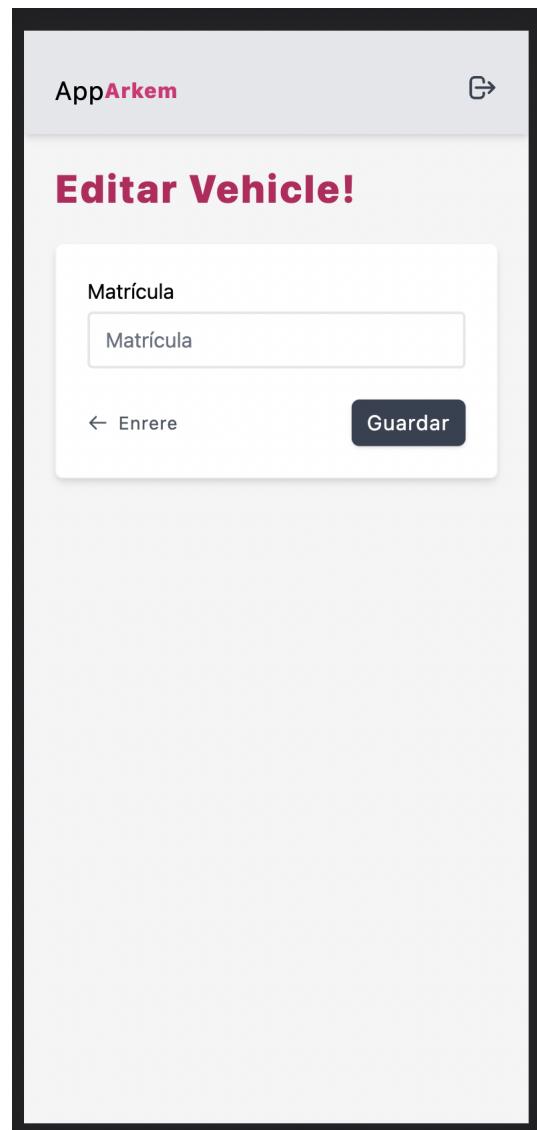


Figura A.5 – Pantalla editar cotxes de l'usuari

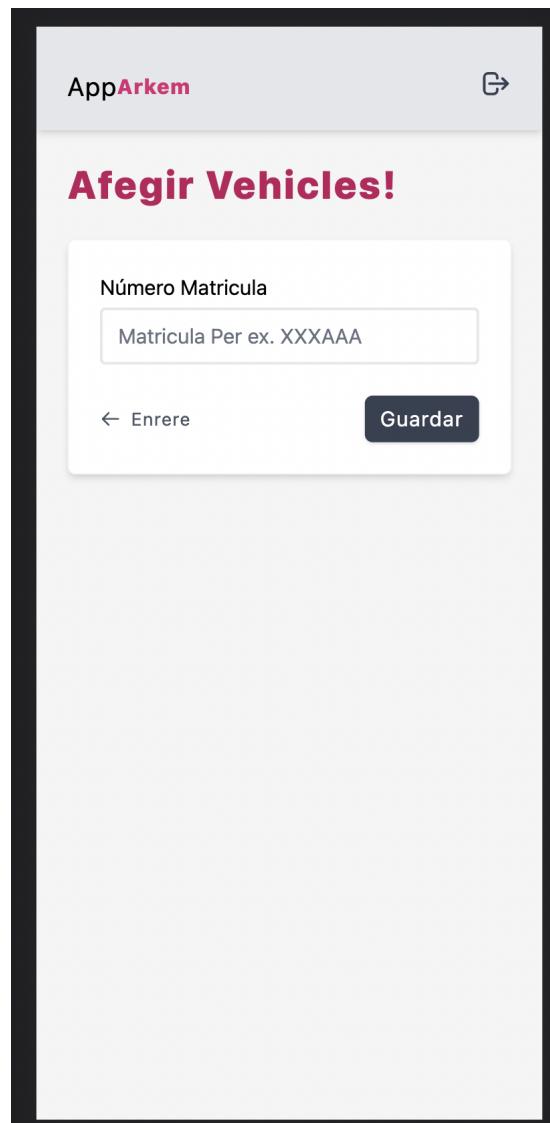


Figura A.6 – Pantalla d'afegir cotxes de l'usuari



Figura A.7 – Pantalla generar codi QR

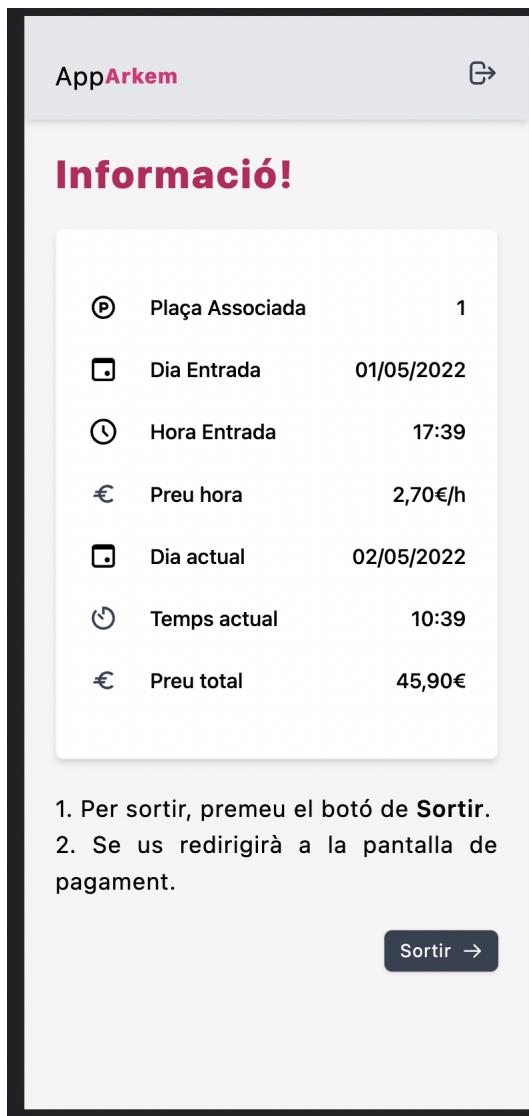


Figura A.8 – Pantalla d'informació de l'aparcament

APÈNDIX A. IMAGES DEL FRONT-END DE L'APLICACIÓ

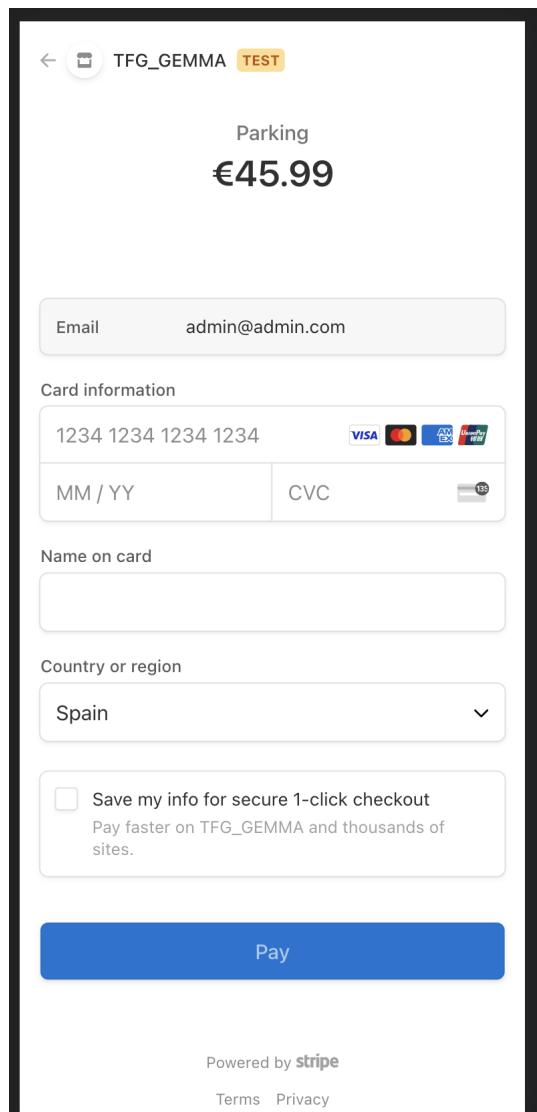


Figura A.9 – Pantalla del pagament



Figura A.10 – Pantalla del pagament validat

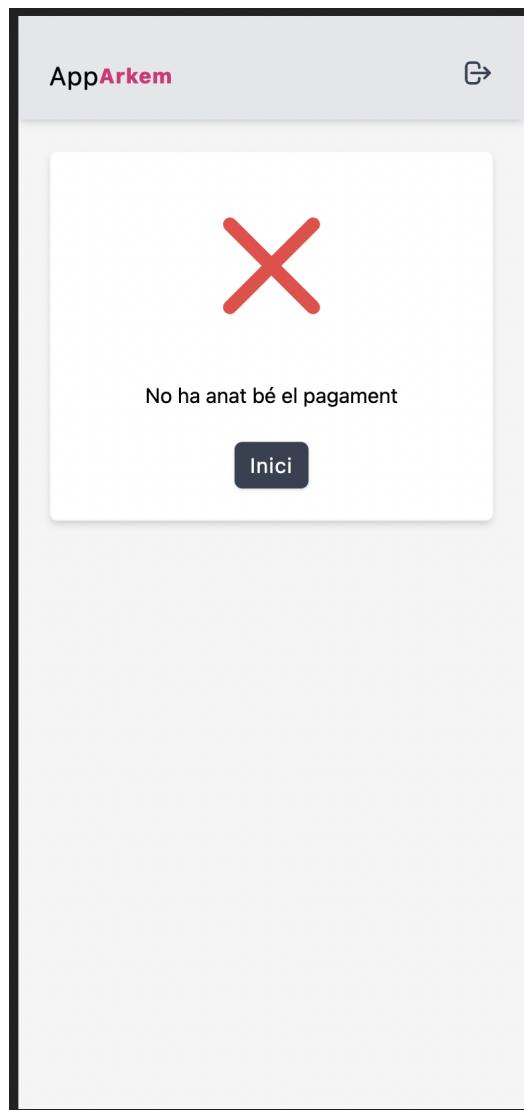


Figura A.11 – Pantalla del pagament cancel·lat

## APÈNDIX A. IMAGES DEL FRONT-END DE L'APLICACIÓ

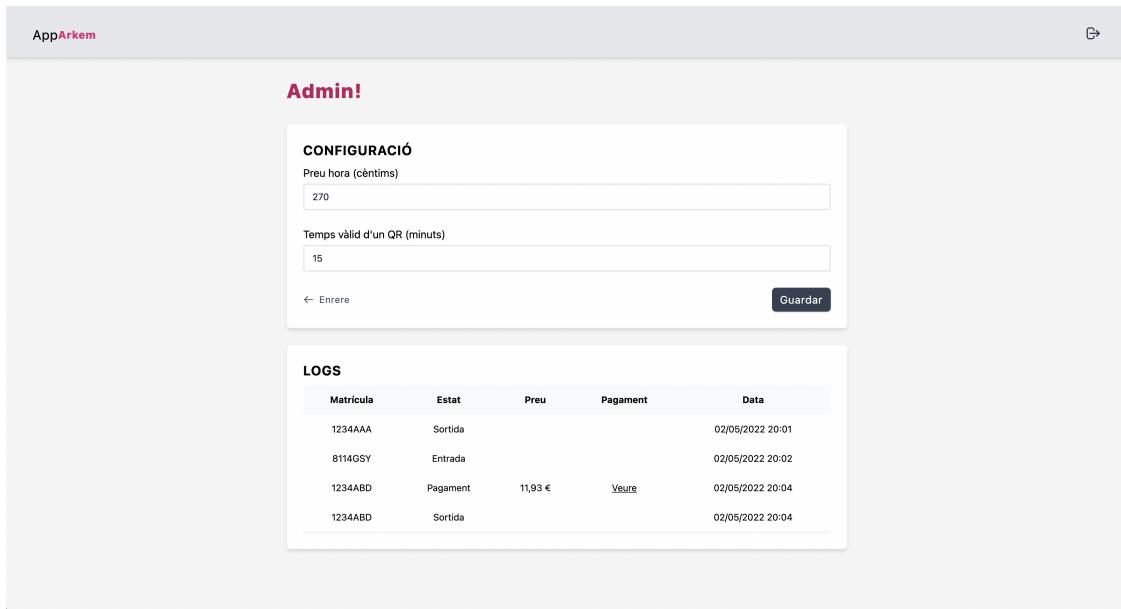


Figura A.12 – Pantalla panell d'administració

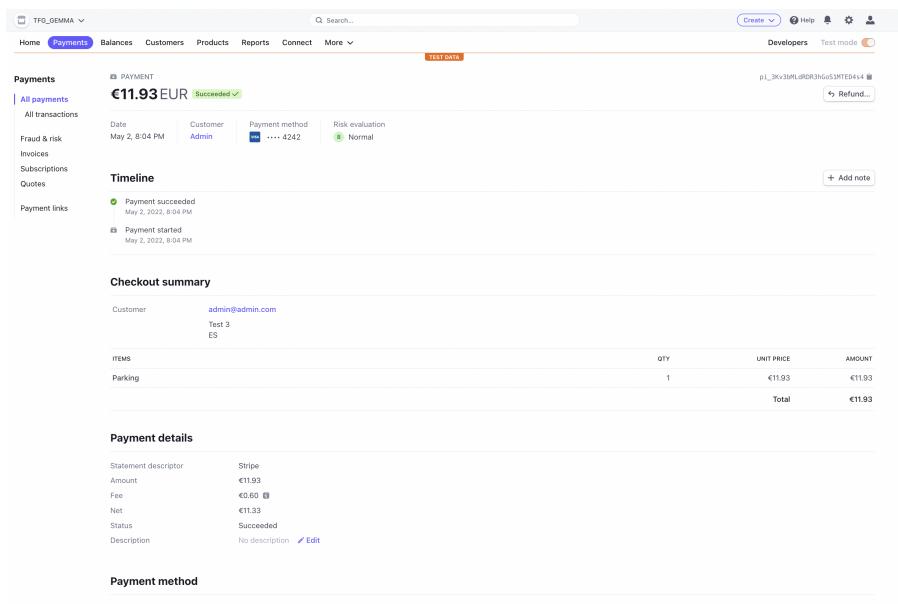


Figura A.13 – Pantalla panell d'informació de pagament a *Stripe*

## A.1 Alertes

Per fer aquests *PopUps*, l'aplicació ha fet servir la llibreria SweetAlert2 [Swe]

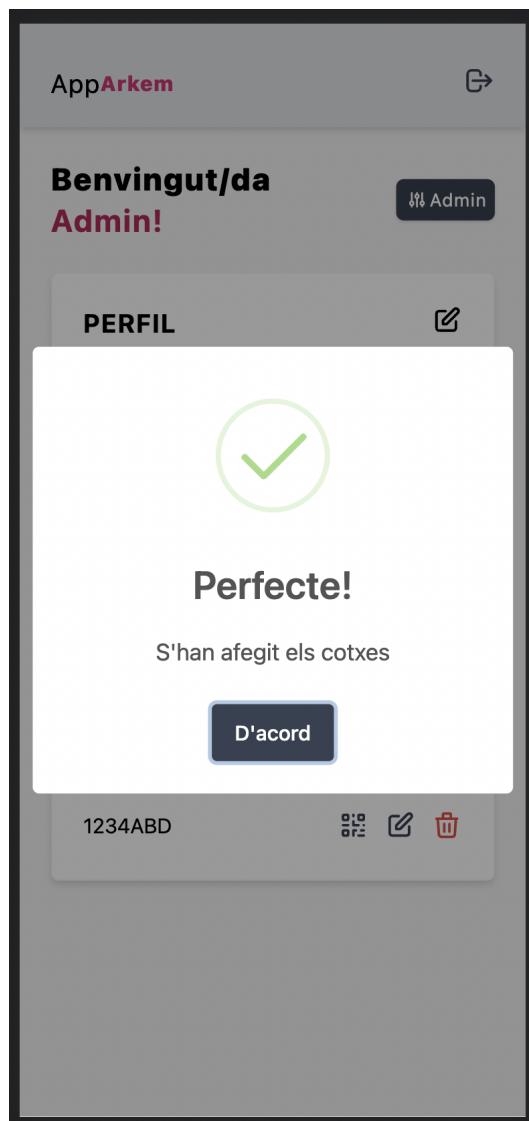


Figura A.14 – Alerta d'èxit



Figura A.15 – Alerta d'error

## B Enllaços a vídeos de l'aplicació

Per veure el funcionament real del projecte es poden veure els següents vídeos:

- Gravació del funcionament de l'aplicació. Compte d'administració inicia sessió, mostra com s'afegeixen cotxes de l'usuari Admin. Finalment, clica el botó per entrar al pàrquing, on es genera un codi QR. Aquest codi QR és llegit per la barrera (Raspberry Pi). El servidor valida el codi QR, envia un esdeveniment (WebSockets) a l'aplicació que redirigeix a la pantalla d'informació. El vídeo és a [https://youtube.com/shorts/7SgkVWy9w\\_I?feature=share](https://youtube.com/shorts/7SgkVWy9w_I?feature=share)
- Vídeo que mostra el funcionament de la barrera. On llegeix el codi QR generat per l'aplicació fa sonar el brunzidor. Fa la petició al servidor, es pot veure com el LED de color blau s'encén. La Raspberry Pi accedeix a apujar la barrera quan obté resposta del servidor. El LED verd fent pampallugues simbolitza que la barrera està pujant, un cop pujada el LED es queda encès un temps per poder entrar el vehicle dins del pàrquing. Finalment, la barrera baixa, també mostrant un LED blau fent pampallugues. Un cop ha baixat el LED es posa de color vermell. El vídeo és el següent <https://youtube.com/shorts/K7QQHMIinBw?feature=share>.
- L'usuari vol sortir del pàrquing, clica el botó de sortir i l'aplicació redirigeix l'usuari a la pantalla de pagaments. Inseriu una targeta de prova i quan el pagament és validat, l'aplicació el redirigeix a una pantalla on es genera el codi QR de sortida. La barrera el llegeix. <https://youtu.be/wfxZqMHwYiQ>
- La barrera llegeix el codi QR per sortir del pàrquing. La barrera quan llegeix el codi QR sona el brunzidor, i quan fa la petició HTTP ho mostra amb el LED blau. Quan ha rebut la resposta del servidor apuja la barrera perquè l'usuari pugui sortir del pàrquing i finalment baixa la barrera. [https://youtube.com/shorts/7WrHquP0\\_tA?feature=share](https://youtube.com/shorts/7WrHquP0_tA?feature=share).
- L'usuari mostra a la barrera un codi QR que no és el bo. La barrera quan llegeix el codi QR sona un *Bzz* el brunzidor, i quan fa la petició HTTP ho mostra amb el LED blau. La resposta del servidor no és satisfactòria, per tant, la barrera mostra amb un pampallugueig de color vermell que alguna cosa no ha funcionat correctament <https://youtube.com/shorts/xCrC1Vhb1Qo?feature=share>.
- Gravació de la pantalla d'administració. Mostra dels Logs i de la pàgina de pagament de Stripe. <https://youtu.be/p9qcisBsLWI>