HOMEWEB
Aplicació web per a la gestió immobiliària

Ivan Moll Moll
Grau Enginyeria Informàtica
Desenvolupament Web

Professors:
Antoni Oller Arcas
Santi Caballe Llobet

Juny 2021

Aquesta obra està subjecta a una llicència de Reconeixement-NoComercial-SenseObraDerivada 3.0 Espanya de Creative Commons

Llicències alternatives (triar alguna de les següents i substituir la de la pàgina anterior)

A) Creative Commons:

Aquesta obra està subjecta a una llicència de Reconeixement-NoComercial-SenseObraDerivada 3.0 Espanya de Creative Commons

Aquesta obra està subjecta a una llicència de Reconeixement-NoComercial-CompartirIgual 3.0 Espanya de Creative Commons

Aquesta obra està subjecta a una llicència de Reconeixement-NoComercial 3.0 Espanya de Creative Commons

Aquesta obra està subjecta a una llicència de Reconeixement-SenseObraDerivada 3.0 Espanya de Creative Commons

Aquesta obra està subjecta a una llicència de Reconeixement-CompartirIgual 3.0 Espanya de Creative Commons

Aquesta obra està subjecta a una llicència de Reconeixement 3.0 Espanya de Creative Commons

B) GNU Free Documentation License (GNU FDL)

Copyright © 2021 Ivan Moll Moll.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections,

no Front-Cover Texts, and no Back-Cover Texts.
A copy of the license is included in the section entitled "GNU Free Documentation License".

C) Copyright

© (l'autor/a)
Reservats tots els drets. Està prohibit la reproducció total o parcial d'aquesta obra per qualsevol mitjà o procediment, compresos la impressió, la reprografia, el microfilm, el tractament informàtic o qualsevol altre sistema, així com la distribució d'exemplars mitjançant lloguer i préstec, sense l'autorització escrita de l'autor o dels límits que autoritzi la Llei de Propietat Intel•lectual.

“Els meus sincers agraïments a tota la meva família que en tot moment s’ha preocupat per mi, i a tots els professors i companys que me han acompanyat durant aquest llarg camí.”

FITXA DEL TREBALL FINAL

Títol del treball:	*Web per a la gestió d'una immobiliària*
Nom de l'autor:	*Ivan Moll Moll*
Nom del consultor:	*Antoni Oller Arcas*
Data de lliurament (mm/aaaa):	*06/2021*
Àrea del Treball Final:	*Desenvolupament Web*
Titulació:	*Grau d'Enginyeria Informàtica*

Resum del Treball (màxim 250 paraules):

Aquest treball té l'objectiu de desenvolupar una aplicació web per a la gestió d'una empresa del sector immobiliari. La idea va sorgir ja que a l'illa de Menorca gràcies a la pandèmia hi ha hagut un creixement de la demanada de propietats, per tant l'objectiu d'aquest projecte cobrir una sèrie de necessitats relacionades amb aquest tipus de negoci.

Aquest projecte final de grau es composa de varies fases.

La primera fase consisteix en definir de forma clara la necessitat que pretén resoldre el projecte, el treball que es realitzarà a cada fase, i la descomposició d'aquest treball en tasques més petites.

La segona fase es realitza un anàlisis funcional i es confecciona el disseny tècnic, s'explica que fa l'aplicació i com ho fa. L'anàlisi funcional consisteix en definir quins rols tindrem dins el sistema i a partir d'aquets es defineixen els requisits que s'hauran de tenir en compte.

En quan al disseny es determina quina es l'arquitectura escollida per el projecte i quins seran els components i interfícies que s'implementaran. Per altre banda també es descriu el perfil tecnològic escollit per a portar a terme el projecte web, es detallen també els *frameworks* escollits i quines són les seves arquitectures. Es creen diagrames d'activitat amb UML i es realitza un diagrama de classes per definir la base de dades.

La tercera fase es descriu la metodologia seguida per a la implementació, també es fa referència a les tecnologies i quins elements han intervingut amb la construcció.

Abstract (in English, 250 words or less):

This work aims to develop a web application for the management of a real estate company. The idea arose because on the island of Menorca, thanks to the pandemic, there has been an increase in the demand for properties, so the aim of this project is to cover a series of needs related to this type of business.

This final degree project consists of several phases.

The first phase consists of clearly defining the need to solve the project, the work to be done in each phase and the breakdown of this assignment into smaller tasks.

In the second phase there is a functional analysis and a technical design, it is explained what the application can do and how it does it. Functional analysis consists of defining what roles we will have within the system and from these we define the requirements that must be taken into account.

In terms of design, it is determined which architecture is chosen by the project and which components and interfaces will be implemented. On the other hand, it also describes the technological profile chosen to carry out the web project, it also details the chosen frameworks and what their architectures are. Activity

diagrams are created with UML and a class diagram is made to define the database.

The third phase describes the methodology followed for the implementation, it also refers to the technologies and what elements have been involved in the construction.

Paraules clau (entre 4 i 8):

Angular, Laravel, MySQL, API REST, Aplicació web, Enginyeria de programari, HTML, CSS.

Índex

1. Introducció 3
1.1 Context i justificació del treball 3
1.2 Objectius del treball 4
1.3 Enfocament i mètode seguit 4
1.4 Planificació del treball 5
1.5 Breu sumari de productes obtinguts 10
1.6 Breu descripció dels altres capítols de la memòria 10
2. Anàlisi de requisits 11
2.1 Descripció detallada del projecte 11
2.2 Anàlisi de requisits 14
2.3 Modelització de rols d'usuari 14
2.4 Especificació requisits a partir dels rols d'usuari 15
2.5 Manteniment (CRUD) 15
2.6 Requisits no funcionals 15
2.7 Especificació dels requisits 16
2.8 Restriccions del model de negoci 17
2.9 Casos d'ús 18
3. Disseny i arquitectura 26
3.1 Arquitectura escollida 27
3.2 Arquitectura al servidor 29
3.3 Arquitectura REST 30
3.4 Arquitectura MVVM / MVW al client [17] 35
3.5 Inteficies gràfiques i mockups 36
4. Components i interfícies 39
4.1 Arquitectura de 3 capes 39
4.2 Patró d'arquitectura MVC 39
4.3 Comunicació components backend 43
4.4 Disseny base de dades 44
5. Implementació 47
5.1 Implementació amb Angular i Laravel 47
5.2 Implementació iterativa amb TDD 50
5.3 Repositoris GitHub [20] 51
5.4 Docker 51
5.5 Estimació de temps dedicat 52
6. Conclusions 53
7 Línies de futur 55
8. Glossari 56
9. Bibliografia 58
10. Annexos 60
Apèndix 1: Diagrames d'activitat backend 60
Apèndix 2: MVC amb Laravel 67
Apèndix 3: API REST amb Laravel 69
Apèndix 4: MVVM o MVW amb Angular 71
Apèndix 5: Generació projectes backend i frontend 72
Apèndix 6: Documentació API amb OpenAPI 3.0 i Swagger. 80
Apèndix 7: Taula especificació requisits. 81
Apèndix 8: Projecte amb client HTTP Postman. 81

Llista de figures

Il·lustració 1: Diagrama de Gantt PAC 1 7
Il·lustració 2: Diagrama de Gantt PAC 2 8
Il·lustració 3: Diagrama de Gantt PAC 3 9
Il·lustració 4: Estructura de la web 11
Il·lustració 5 - Cas d'ús formal i diagrama UML 19
Il·lustració 6: Diagrama UML casos d'ús 26
Il·lustració 7: Arquitectura 3 capes 27
Il·lustració 8: Arquitectura client - servidor 28
Il·lustració 9: Arquitectura orientada a serveis (SOA) 29
Il·lustració 10: Arquitectura REST 32
Il·lustració 11: Cicle de vida JWT [22] 33
Il·lustració 12: Pàgina inici 36
Il·lustració 13: Pàgina detall propietat 36
Il·lustració 14: Pàgina categories 37
Il·lustració 15: Pàgina resultats cerca 37
Il·lustració 16: Pàgina contacte 37
Il·lustració 17: Panell administració pàgina inici 38
Il·lustració 18: Panell administració pàgina propietats 38
Il·lustració 19: Panell administració detall propietat 38
Il·lustració 20: Panell administració formulari crear/editar propietat 38
Il·lustració 21: Diagrama UML arquitectura 3 capes 39
Il·lustració 22: Diagrama UML patró arquitectura MVC 40
Il·lustració 23: Diagrama UML refinament patró arquitectura MVC 41
Il·lustració 24: Diagrama de seqüència backend 43
Il·lustració 25: Diagrama classes model de dades 44
Il·lustració 26: Diagrama de classes refinament relació n-ària històric de preus 45
Il·lustració 27: Diagrama de classes refinament relació n-ària vendes 46
Il·lustració 28: Diagrama de classes refinament relació n-ària visites 46
Il·lustració 29 - Arquitectura aplicació web 50
Il·lustració 30 - Cicle implementació amb TDD 51

1. Introducció

Com a introducció es parla de quines han estat les decisions que han fet que s'arribi al projecte que s'ha escollit, també es descriu quins son els objectius principals del projecte. Per altre banda es fa una breu explicació del mètode de treball que s'ha seguit per arribar als objectius marcats, i una planificació de les parts que componen aquest projecte. Finalment es descriuen els productes obtinguts.

1.1 Context i justificació del treball

La decisió final que s'ha pres per a realitzar aquest treball final de grau, ha estat el de desenvolupar una aplicació web on és pugui gestionar el treball diari d'una immobiliària. La motivació principal per haver escollit aquesta temàtica és que ens els últims anys hi ha hagut un increment de vendes de propietats a persones estrangeres, gràcies a la atracció paisatgística i rural que tenim a l'illa de Menorca. També s'ha registrat un augment de vendes a persones estrangeres, tot això gràcies a la pandèmia que hem tingut, ja que ha donat peu a que les persones poguessin teletreballar [1].

Es vol construir un sistema de qualitat que pugui donar suport a les empreses dedicades a aquest sector, tot pensant amb l'estalvi de temps i costos, agilitzant processos.

Homeweb proporcionarà una solució integral per a la gestió de propietats, serà multiusuari, i permetrà també gestionar categories, ciutats, països, vendes, canvis de preu, visites i característiques, però aquestes accions estaran restringides o permeses a certs rols d'usuari.

1.2 Objectius del treball

Els objectius a assolir són:

- Oferir una solució integral a les empreses del sector immobiliari.
- Desenvolupar un projecte tot complint les fases i respectant les fites.
- Posar en pràctica els coneixements adquirits al Grau d'Enginyeria Informàtica, més concretament de les assignatures relacionades amb l'Enginyeria de programari.
- Saber construir un projecte real amb les darreres tecnologies que tenim al mercat i aprendre d'això.

1.3 Enfocament i mètode seguit

Aquest treball consisteix en la implementació des de zero d'una aplicació web que cobreix unes necessitats concretes dins el món de la gestió de les propietats i les immobiliàries.

Per portar a terme la implementació s'han realitzat iteracions sobre dels requisits establerts en la fase d'anàlisi. Considerant que no és un projecte molt gran no s'ha instaurat una metodologia àgil Scrum [2] al 100%, sinó que s'ha implementat sobre els casos d'ús i iterat sobre ells per anar refinant el resultat, tot això tenint en compte que s'ha utilitzat la tècnica TDD [3] per realitzar la implementació dels requisits.

Un cop acabada la implementació de la prova i en conseqüència de la funcionalitat requerida, es proven els resultats amb un client HTTP com és Postman, d'aquesta manera es pot veure si el resultat és l'esperat o no.

1.4 Planificació del treball

La planificació del projecte consisteix en planificar les diferents fases que tindrà el projecte, s'ha de quantificar el temps i els recursos necessaris per dur-lo a terme.

Aprofitant el que es va aprendre a la assignatura de ***Enginyeria de programari***, utilitzarem i aprofitarem la metodologia de gestió de projectes que ens descriu el ***SWEBOK*** (Software Engineering Body of Knowledge) [4].

FASE	DESCRIPCIÓ
Gestió del projecte	És una disciplina general en el desenvolupament de projectes, però en el cas de projectes de desenvolupament de programari s'ha de complementar el mètode general amb els mètodes, tècniques i les eines pròpies dels projectes de desenvolupament de programari.
Identificació i gestió de requisits	Els requisits serveixen per delimitar l'abast del projecte d'entre les possibles solucions que es puguin tenir.
Modelització	Aquesta fase inclou la creació de models del sistema a desenvolupar. Aquests faciliten la comprensió dels requisits i el disseny del sistema. Aquesta tècnica equival a construir maquetes o models, el llenguatge més estès per a la creació de models es el ***UML*** (Undefined Modeling Language).
Construcció i proves	La construcció inclou l'escriptura del codi font (implementació) i la realització de les proves necessàries per garantir l'absència d'errors. Un cop creat el codi es procedeix al desplegament del mateix i es posa a disposició dels usuaris.
Qualitat	Consisteix en l'establiment de mètriques que ens ajudin a determinar si el programari compleix o no els criteris de qualitat marcats.
Manteniment i reenginyeria	Aquesta fase no s'inclou en l'execució del mateix projecte, està fora de l'abast del mateix, però és necessari que al finalitzar-lo tinguem, per una banda, la fase de manteniment correctiu encarregat de corregir els errors detectats en el programari, i per un altra banda, el manteniment evolutiu consisteix en afegir noves funcionalitats o adaptar-ne les ja existents per tal de cobrir noves necessitats detectades després del final del projecte.

Per altra banda, s'han de definir els lliurables on s'hauran de presentar els corresponents documents que composen aquest TFG amb les seves dates d'inici i fi o lliurament.

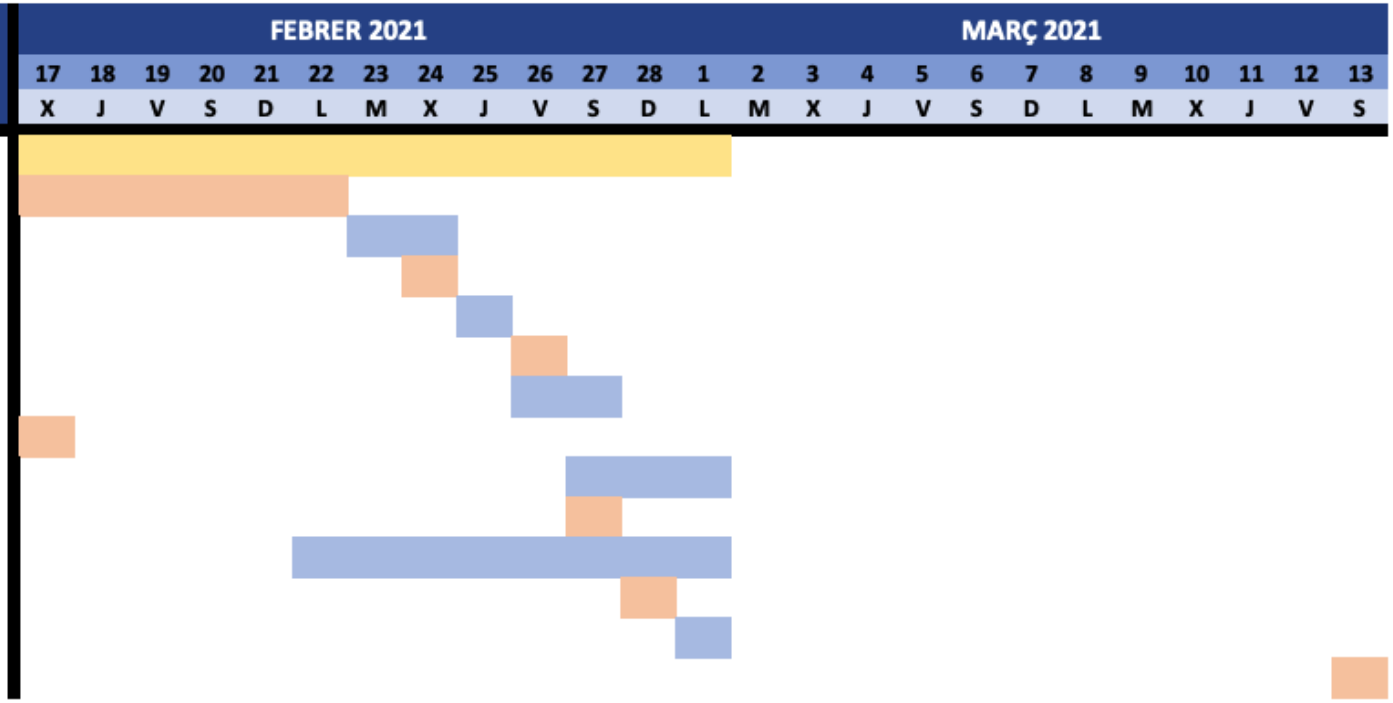
Detall de les fases a seguir:

Lliurables	Data inici	Data lliurament	Fase SWEBOK
PAC 1: Pla de treball	17/02/2021	01/03/2021	**Gestió del projecte**
PAC 2: Anàlisi i disseny	02/03/2021	01/04/2021	**Identificació i gestió de requisits + Modelització**
PAC 3: Implementació	05/04/2021	21/05/2021	**Construcció i proves**
Memòria i presentació	22/05/2021	09/06/2021	**Gestió del projecte**
Defensa	14/06/2021	21/06/2021	**-**

Un cop definits els lliurables per a cada PAC i la seva relació amb les fases de planificació d'un projecte establerts per el ***SWEBOK***, s'ha de remarcar que el pla de treball i la memòria consisteixen en una petita part del que realment representa la fase de gestió del projecte.

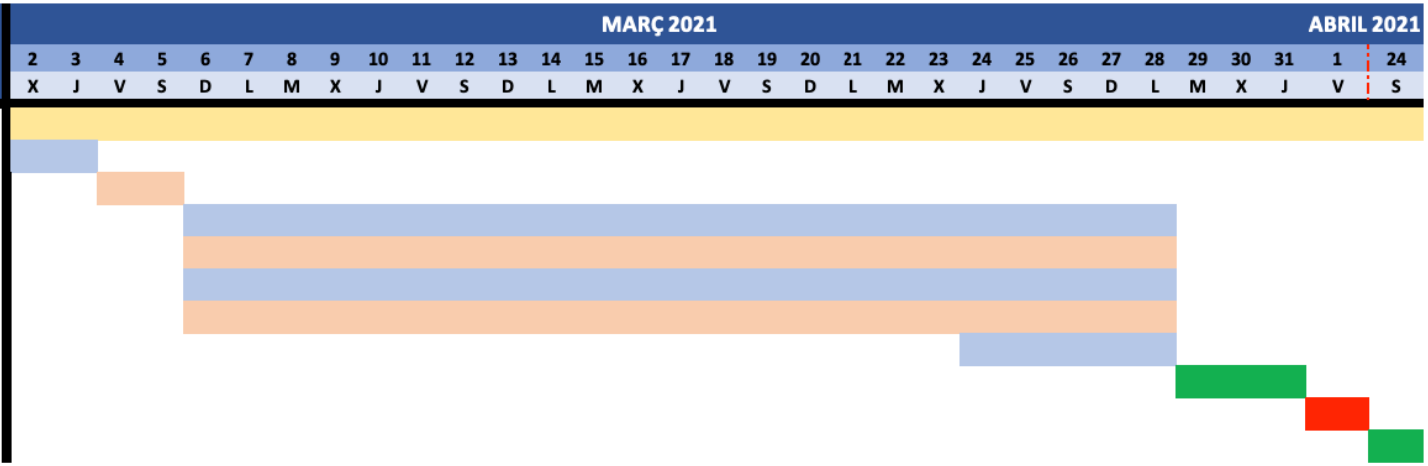
Per poder portar un control de les fites amb els respectius costos de temps a dedicar per a cada fase i dins cada fase per cada tasca a desenvolupar, s´han realitzat diferents diagrames de Gantt.

	Descripció lliurable	Inici	Num. Dies
PAC 1	Elaboració pla de treball	17/2/21	12
	Pluja de idees	17/2/21	6
	Consulta documentació	23/2/21	2
	Introudcció al pla de treba	25/2/21	1
	Descripció del projecte	25/2/21	1
	Planificació del projecte	26/2/21	1
	Diagrama de Gantt	26/2/21	2
	Tecnologies a utilitzar	17/2/21	1
	Recerca de web apps simil	27/2/21	3
	Objectius a assolir	27/2/21	1
	Bibliografia	22/2/21	8
	Revisió Pla de Treball	28/2/21	2
	Lliurament Pla de Treball	1/3/21	1
	Qualificació PAC 1	13/3/21	1



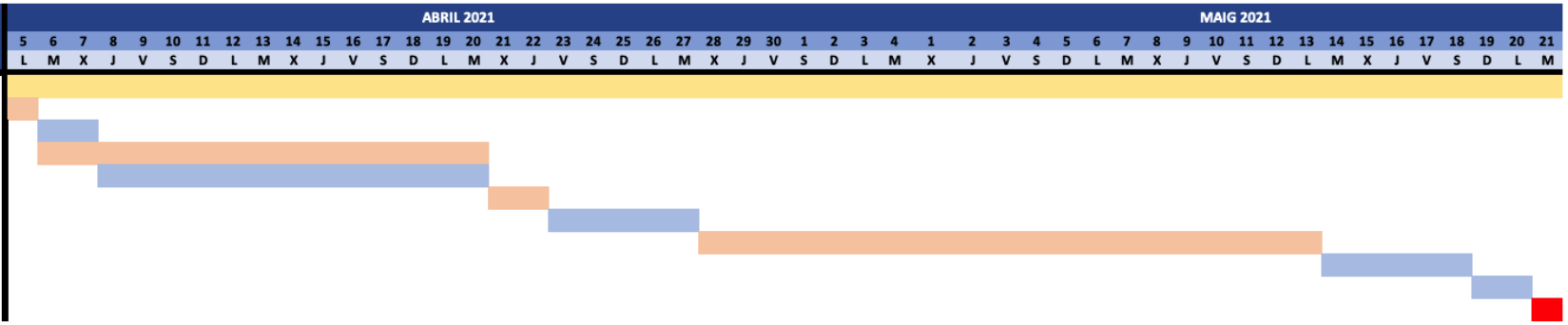
Il·lustració 1: Diagrama de Gantt PAC 1

	Descripció lliurable	Inici	Num. Dies
PAC 2	Anàlisi i disseny - PAC 2	2/3/21	30
	Identificaió de requisits	2/3/21	2
	Gestió dels requisits	4/2/21	3
	Modelització diagrames de classes	6/3/21	23
	Modelització arquitectura MVC	6/3/21	23
	Modelització casos d'ús	6/3/21	23
	Modelització diagrames de seqüència	6/3/21	23
	Disseny base de dades	24/3/21	5
	Revisió PAC 2	29/3/21	3
	Lliurament PAC 2	1/4/21	1
	Qualificació PAC 2	24/4/21	1



Il·lustració 2: Diagrama de Gantt PAC 2

	Descripció lliurable	Inici	Num. Dies
PAC 3	Implementació - PAC 3	5/4/21	46
	Preparació entorn backend	5/4/21	1
	Construcció base dades	6/4/21	2
	Construcció API Rest	6/4/21	15
	Proves API Rest	8/4/21	13
	Preparació entorn frontend	21/4/21	2
	Maquetar vistes	23/4/21	5
	Integració amb Angular	28/4/21	20
	Adjustaments al projecte	14/5/21	5
	Revisió de la documentació	19/5/21	3
	Lliurament Implementació - PAC 3	21/5/21	1



Il·lustració 3: Diagrama de Gantt PAC 3

1.5 Breu sumari de productes obtinguts

Aquí tenim un llistat de tots els productes obtinguts una vegada acabat el projecte.

- PAC 1: Pla de treball
- PAC 2: Anàlisi i disseny
- PAC 3: Implementació
- Memòria i presentació
- Repositori GitHub del backend
- Repositori GitHub del frontend
- Documentació API

1.6 Breu descripció dels altres capítols de la memòria

En el capítol 2 es descriu l'anàlisi funcional i els requisits de la aplicació. També es descriuen els casos d'ús de una manera formal i amb un diagrama UML [5].

En el capítol 3 es descriu el disseny que es segueix per crear l'aplicació web. Es fa una descripció de l'arquitectura escollida, també es fa un incís al que és una API REST [6] i com s'implementa, també es descriuen els mecanismes de seguretat utilitzats.

En el capítol 4 trobem la definició dels components i les seves interfícies. Es fa un refinament de l'arquitectura escollida fins arribar a un diagrama UML que ens mostra els components que existeixen dins l'aplicació.

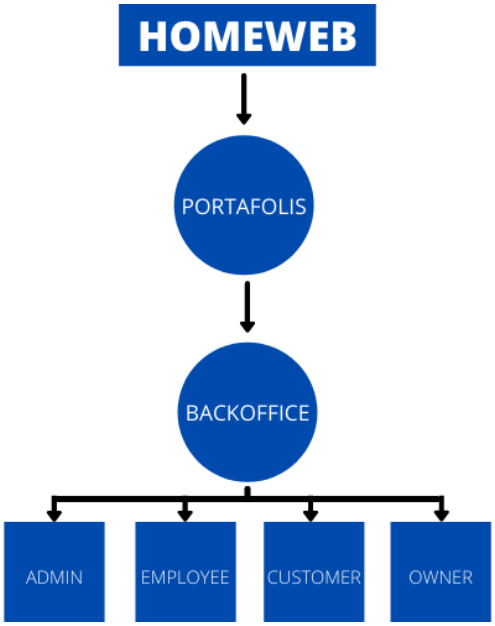
El capítol 5 conté de forma més explícita com s'ha implementat el *backend* i el *frontend*. També es comenta que s'ha hagut de fer per muntar amb *Docker* els contenidors necessaris per poder executar l'aplicació.

2. Anàlisi de requisits

En aquest apartat es fa l'anàlisi de requisits, part fonamental en la vida de un projecte, gràcies a ells es marcaran les funcionalitats que haurà de realitzar l'aplicació per a satisfer les necessitats dels stakeholders.

2.1 Descripció detallada del projecte

La finalitat principal d'aquest projecte és desenvolupar una aplicació web relacionada amb el món de les propietats i les immobiliàries. Aquesta aplicació web la podem segmentar en dues parts, per una banda un portafolis de propietats i un cercador que es mostraran just entrar a la web homeweb.com , i per l'altra el desenvolupament d'un registre d'usuari, un *login* i un *backoffice* on els diferents rols d'usuari del nostre sistema podran gestionar, consultar i modificar els diferents aspectes de l'aplicació web.
Aquesta seria la estructura general de la nostra aplicació web.



Il·lustració 4: Estructura de la web

Just visitar la web es troba amb una pàgina *home* on es poden visualitzar algunes propietats del sistema, un cercador, un formulari de contacte i un peu de pàgina.

Després tindrem el *backoffice* on un usuari amb un rol determinat pot iniciar sessió i se li mostra un panell amb les diferents accions que pot realitzar sobre

el nostre model de dades. També es mostren diferents informacions relacionades amb el seu perfil d'usuari.

Cada part de l'aplicació està formada per components, aquests són els que es descriuen a continuació:

	COMPONENT	DESCRIPCIÓ
PORTAFOLIS	HOME	Pàgina inicial on es mostra un cercador i les últimes propietats afegides al sistema. També hi ha un formulari de contacte, i es possible iniciar sessió o registrar-se en el sistema.
	CERCADOR	Serà el cercador de la part pública on es podrà visualitzar en diferents llocs dins la mateixa web, i els camps per els quals es podrà fer un cerca, seran: - Referència - Preu - Localitat - Categoria També tindrà una pàgina de resultats, que mostrarà les propietats que compleixen amb els camps de cerca.
	CATEGORIES	Aquest apartat servirà per poder escollir la categoria que es desitgi consultar/visualitzar, on es mostraran totes les propietats que pertanyin a la categoria seleccionada.
	CONTACTE	Apartat on hi haurà un formulari de contacte, en el qual l'usuari podrà realitzar una consulta tot introduint les seves dades de contacte, l'assumpte i el missatge que es vol transmetre.
BACKOFFICE	ADMINISTRADOR	L'usuari amb rol d'administrador, serà el que podrà realitzar tot tipus d'accions a sobre la lògica de negoci de la nostra

		aplicació web. Entre elles, podrà donar d'alta, llegir, modificar i eliminar els diferents models especificats en el nostre model de classes o model de negoci (Propietats, Visites, Canvis de preu, Categories, Localitats i Vendes).
	EMPLEAT	El rol d'empleat podrà realitzar diferents accions en el model, però no tindrà tants permisos com l'usuari amb rol d'administrador. Entre ells, podrà donar d'alta, modificar i llegir propietats, donar d'alta visites a propietats i canvis de preus. També manipular (CRUD [7]) categories, canvis de preu, vendes i localitats.
	CLIENT	El rol de client serà bàsicament per poder crear i modificar visites a propietats, D'aquesta manera els empleats podran realitzar accions al respecte, i es tindrà un registre de les visites que els clients volen realitzar. També si un client vol comprar una propietat tindrà que estar d'alta al sistema per enregistrar la venda.
	PROPIETARI	El rol de propietari tindrà com a principal funció el de donar d'alta les seves propietats que té en venda, així com poder fer modificacions i poder llistar les seves propietats en venda. També podrà realitzar canvis de preu en les seves propietats si així ho veu convenient. Una altra funció serà la de

		veure les visites que té programades per a les seves propietats..

2.2 Anàlisi de requisits

Per determinar els requisits s'ha de realitzar un estudi dels mateixos conjuntament amb els *stakeholders [8]* (persones i entitats que tenen algun impacte o interès en aquest), ja que aquests seran determinants per a l'abast del projecte. Per tant, s'hauran de fer diferents reunions i entrevistes als diferents *stakeholders* relacionats amb el projecte i d'aquesta manera confeccionar un llistat ben detallat i clar de cada requisit i el que ha de fer, per aconseguir satisfer les necessitats de cada *stakeholder*.

En resum, la funció dels requisits és comunicar les necessitats i objectius dels *stakeholders* als desenvolupadors. Si no es fa un bon anàlisi i definició dels requisits i aquests no queden ben clars, al final el projecte podria no satisfer les necessitats dels stakeholders i per tant es podria donar com a projecte inviable i no arribar mai a la seva posada en marxa.
De requisits tenim dos grans grups, els requisits funcionals i els no funcionals.

- **Requisits funcionals:** Són els que ens indiquen el comportament del sistema davant les accions que li arriben dels usuaris o altres sistemes. Fan referència a la funcionalitat que ha de proporcionar el sistema.
- **Requisits no funcionals:** Són els encarregats de proporcionar les restriccions pertinents sobre el conjunt de possibles solucions que pot adoptar el sistema. Aquests requisits es mostren com a restriccions, de fet no són els encarregats de definir el comportament que haurà de tenir el sistema, sinó que especifiquen criteris que avaluïn la qualitat general del sistema (cost, rendiment, seguretat, etc.).

2.3 Modelització de rols d'usuari

Per poder extreure els requisits que el sistema tindrà, utilitzarem la tècnica de modelització de rols d'usuari. És una tècnica orientada a identificar els diferents rols d'usuari que hi haurà al sistema. A partir de l'agrupació de requisits segons el rol d'usuari, es podran fer agrupacions de requisits similars.

Per tant podem definir els requisits del sistema a partir dels diferents rols d'usuari que em anomenat anteriorment.

2.4 Especificació requisits a partir dels rols d'usuari

Per determinar els requisits del nostre sistema ho fem a partir de dos variables, els rols d'usuari i els models o recursos de la nostra capa de negoci.
El sistema tindrà una funcionalitat de gestió de permisos sobre els diferents models, el qual haurà de gestionar ja que alguns rols tindran restringit l'accés a certes accions del sistema.

Un usuari equival a un sol rol, això s'ha determinat així perquè, si un usuari hagués pogut prendre diferents rols, la complexitat del sistema haguera augmentat considerablement, per tant s'ha pres aquesta decisió a causa de la complexitat a l'hora de la implementació.

2.5 Manteniment (CRUD)

Identificant els casos d'ús que haurem de definir per poder dissenyar el comportament que tindrà el sistema, es pot afirmar que consistirà en un manteniment, per tant es pot obviar un nombre determinat de casos d'ús. S'especificarà un únic cas d'ús de nivell general que aglutini els diversos objectius d'usuari del manteniment. Això tindrà l'avantatge de simplificar la feina i evitar la repetició innecessària.

2.6 Requisits no funcionals

A continuació detallarem els requisits no funcionals. Aquests requisits fan referència a les restriccions que tindrà el sistema. A continuació es defineixen aquests requisits no funcionals.

Requisits de seguretat: S'ha de considerar un aspecte molt important com la seguretat ja que els sistemes avui en dia han de aportar aquesta característica, per donar integritat i privacitat al conjunt d'accions que es poden realitzar en el sistema.

En el nostre sistema és construirà un control d'accés perquè no es pugui veure compromesa la integritat i la privacitat del sistema, i que només aquells usuaris autoritzats puguin realitzar canvis al sistema.

També s'implementarà un sistema de polítiques d'accés basades amb els rols d'usuari identificats, que ajudarà a restringir l'accés a segons quines accions del sistema.

2.7 Especificació dels requisits

A continuació s'exposa una taula de requisits de manera formal, per especificar quins seran els requisits de cada rol d'usuari i quines restriccions de seguretat (requisit no funcional de seguretat) s'aplicaran a cadascuna d'elles.

ROL USUARI	MODEL	REQUISITS
ADMINISTRADOR	PROPIETATS	Crear[PT], Recuperar[PT], Actualitzar[PT] i Eliminar[PT]
	USUARIS	Crear[SP], Recuperar[PT], Actualitzar[PT] i Eliminar[SP]
	CATEGORIES	Crear[PT], Recuperar[PT], Actualitzar[PT] i Eliminar[PR]
	VENDES	Crear[PT], Recuperar[PT], Actualitzar[PT] i Eliminar[SP]
	VISITES	Crear[PT], Recuperar[PT], Actualitzar[PT] i Eliminar[PT]
	CUITATS	Crear[PT], Recuperar[PT], Actualitzar[PT] i Eliminar[PR]
	PAÏSOS	Crear[PT], Recuperar[PT], Actualitzar[PT] i Eliminar[PR]
	CANVIS DE PREU	Crear[PT], Recuperar[PT], Actualitzar[SP] i Eliminar[SP]
	CARACTERÍSTIQUES	Crear[PT], Recuperar[PT], Actualitzar[PT] i Eliminar[PR]
EMPLEAT	PROPIETATS	Crear[PT], Recuperar[PT], Actualitzar[PT] i Eliminar[SP]
	USUARIS	Crear[SP], Recuperar[PT], Actualitzar[PT] i Eliminar[SP]
	CATEGORIES	Crear[PT], Recuperar[PT], Actualitzar[PT] i Eliminar[SP]
	VENDES	Crear[PT], Recuperar[PT], Actualitzar[PR] i Eliminar[SP]
	VISITES	Crear[PT], Recuperar[PR], Actualitzar[PR] i Eliminar[PR]
	CUITATS	Crear[SP], Recuperar[PT], Actualitzar[SP] i Eliminar[SP]
	PAÏSOS	Crear[SP], Recuperar[PT], Actualitzar[SP] i Eliminar[SP]
	CANVIS DE PREU	Crear[PT], Recuperar[PT], Actualitzar[SP] i Eliminar[SP]
	CARACTERÍSTIQUES	Crear[SP], Recuperar[PT], Actualitzar[SP] i Eliminar[SP]
CLIENT	PROPIETATS	Crear[SP], Recuperar[PT], Actualitzar[SP] i Eliminar[SP]

	USUARIS	Crear[SP], Recuperar[PR], Actualitzar[PR] i Eliminar[SP]
	CATEGORIES	Crear[SP], Recuperar[PT], Actualitzar[SP] i Eliminar[SP]
	VENDES	Crear[SP], Recuperar[PR], Actualitzar[SP] i Eliminar[SP]
	VISITES	Crear[PR], Recuperar[PR], Actualitzar[PR] i Eliminar[PR]
	CUITATS	Crear[SP], Recuperar[PT], Actualitzar[SP] i Eliminar[SP]
	PAÏSOS	Crear[SP], Recuperar[PT], Actualitzar[SP] i Eliminar[SP]
	CANVIS DE PREU	Crear[SP], Recuperar[SP], Actualitzar[SP] i Eliminar[SP]
	CARACTERÍSTIQUES	Crear[SP], Recuperar[PT], Actualitzar[SP] i Eliminar[SP]
PROPIETARI	PROPIETATS	Crear[PR], Recuperar[PT], Actualitzar[PR] i Eliminar[SP]
	USUARIS	Crear[SP], Recuperar[PR], Actualitzar[PR] i Eliminar[SP]
	CATEGORIES	Crear[SP], Recuperar[PT], Actualitzar[SP] i Eliminar[SP]
	VENDES	Crear[SP], Recuperar[PR], Actualitzar[SP] i Eliminar[SP]
	VISITES	Crear[SP], Recuperar[PR], Actualitzar[SP] i Eliminar[SP]
	CUITATS	Crear[SP], Recuperar[PT], Actualitzar[SP] i Eliminar[SP]
	PAÏSOS	Crear[SP], Recuperar[PT], Actualitzar[SP] i Eliminar[SP]
	CANVIS DE PREU	Crear[PR], Recuperar[PR], Actualitzar[SP] i Eliminar[SP]
	CARACTERÍSTIQUES	Crear[SP], Recuperar[PT], Actualitzar[SP] i Eliminar[SP]

Llegenda: [PT] permís total, [PR] permís amb restriccions, [SP] sense permís.

- **Permís total:** Es podran realitzar accions damunt totes les instàncies del recurs.
- **Permís amb restriccions:** Es podran realitzar accions damunt un subconjunt d'instàncies relacionades amb el rol de l'usuari.
- **Sense permís:** No es podrà realitzar cap acció damunt cap instància del recurs.

Com a puntualització cal remarcar que possiblement quan es passi al disseny de ben segur sorgirà la necessitat de crear altres funcionalitats sobre els models presentats, però les principals funcions seran les descrites anteriorment a la taula de requisits.

2.8 Restriccions del model de negoci

A part de l'especificació per rols d'usuari i manteniment, també s'analitzaran les restriccions que haurà de complir el sistema per tal de complir amb les regles de negoci, és a dir una sèrie de restriccions que s'hauran d'acomplir per obtenir una

consistència i lògica de les dades que tindrem. Aquestes restriccions estaran relacionades amb els diferents models de dades que hi ha al sistema o aplicació web. Seguidament es detallen per cada model de dades les restriccions necessàries perquè les polítiques de negoci es compleixin.

MODEL	RESTRICCIONS O POLÍTIQUES DE NEGOCI
Propietats	⇒ Les propietats tindran una referència que les identifica unívocament.
Categories	⇒ Sense restriccions
Països	⇒ Sense restriccions
Ciutats	⇒ Sense restriccions
Canvis de preus	⇒ Els canvis de preu no es poden solapar, i han de tenir un ordre cronològic.
Visites	⇒ No es podrà realitzar una visita a una propietat, si tant la propietat, com l'empleat i el client ja tenen una visita concertada a aquella hora. ⇒ Les visites es realitzaran a hores punta i tindran una duració d'una hora.
Vendes	⇒ Quan es realitzi una venda, la propietat quedarà en estat venuda. El propietari no es modificarà.
Característiques	⇒ Sense restriccions
Usuaris	⇒ Sense restriccions

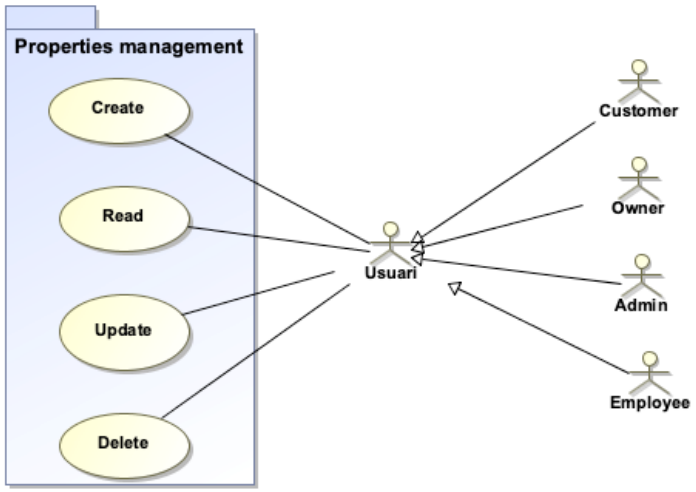
2.9 Casos d'ús

Per definir la interacció dels diferents rols d'usuari de la nostra aplicació, s'han creat els casos d'ús que ens mostraran de forma esquemàtica (taules) i gràfica (*UML*) aquesta interacció dels usuaris amb el sistema. En el cas d'aquest projecte, s'ha optat per modelitzar un cas d'ús per a cada model de dades, ja que les diferents accions que es podran dur a terme són iguals per a cada rol d'usuari.

A més a més s'ha modelitzat un cas d'ús també per la funcionalitat de autenticació al sistema.

Per confeccionar els següents casos d'ús s'ha tingut en compte que molts d'aquets comparteixen moltes similituds entre si, per tant es veurà com alguns d'ells s'han especificat per a un grup de manteniments i/o rols d'usuari.

Al final d'alguns casos d'ús, si tenim algun rol amb pocs permisos, s'indicarà en quin manteniment pot realitzar interaccions. Així he pres la decisió per no fer un altre cas d'ús amb pocs passos.

MODEL ESQUEMÀTIC (TAULA)	MODE GRÀFIC (UML)
Cas d'ús: Gestionar propietats **Actor principal:** Administrador, Empleat i Propietari **Nivell:** General **Escenari principal d'èxit:** 1. L'usuari s'identifica davant el sistema. 2. El sistema li mostra el panell d'administració. 3. L'usuari clica a sobre el menú de propietats. 4. El sistema mostra el panell propietats amb un llistat de les mateixes. a. Si el rol es administrador o empleat, es mostraran qualsevol propietats. b. Si el rol es propietari sols es mostraran les seves propietats. 5. Els passos 3 i 4 es repeteixen fins que l'usuari decideix finalitzar (cas d'ús s'acaba). **Escenaris alternatius:** ***a. Crear una nova propietat*** 1. L'usuari indica que vol crear una nova propietat. 2. El sistema mostra un formulari d'alta de propietat. i. Si el rol es un administrador o empleat, es mostrarà un camp propietari. ii. Si el rol es un propietari no es mostrarà el camp propietari. 3. L'usuari introdueix les dades al formulari. 4. El sistema enregistra la nova propietat i torna al punt 4. ***b. Eliminar una propietat*** 1. Si l'usuari es administrador pot indicar que vol eliminar una propietat. 2. El sistema elimina la propietat indicada i torna al punt 4. ***c. Actualitzar una propietat*** 1. L'usuari indica que vol actualitzar o modificar una propietat. 2. El sistema mostra un formulari amb les dades de la propietat. 3. L'usuari fa els canvis al formulari. 4. El sistema enregistra els canvis i torna al punt 4.	

Il·lustració 5 - Cas d'ús formal i diagrama UML

Cas d'ús: Gestionar propietats
Actor principal: Administrador, Empleat i Propietari
Nivell: General
Escenari principal d'èxit: 1. L'usuari s'identifica davant el sistema. 2. El sistema li mostra el panell d'administració. 3. L'usuari clica a sobre el menú de propietats. 4. El sistema mostra el panell propietats amb un llistat de les mateixes. a. Si el rol es administrador o empleat, es mostraran qualsevol propietats.

b. Si el rol es propietari sols es mostraran les seves propietats. 5. Els passos 3 i 4 es repeteixen fins que l'usuari decideix finalitzar (cas d'ús s'acaba).
Escenaris alternatius: ***a. Crear una nova propietat*** 1. L'usuari indica que vol crear una nova propietat. 2. El sistema mostra un formulari d'alta de propietat. i. Si el rol es un administrador o empleat, es mostrarà un camp propietari. ii. Si el rol es un propietari no es mostrarà el camp propietari. 3. L'usuari introdueix les dades al formulari. 4. El sistema enregistra la nova propietat i torna al punt 4. ***b. Eliminar una propietat*** 1. Si l'usuari es administrador pot indicar que vol eliminar una propietat. 2. El sistema elimina la propietat indicada i torna al punt 4. ***c. Actualitzar una propietat*** 1. L'usuari indica que vol actualitzar o modificar una propietat. 2. El sistema mostra un formulari amb les dades de la propietat. 3. L'usuari fa els canvis al formulari. 4. El sistema enregistra els canvis i torna al punt 4.

* El rol de **client** sols tindrà el permís per poder llegir propietats.

Cas d'ús: Gestionar Usuaris
Actor principal: Administrador, Empleat, Propietari i Client
Nivell: General
Escenari principal d'èxit: 1. L'usuari s'identifica al sistema. 2. El sistema li mostra el panell d'administració. 3. L'usuari clica sobre el menú d'usuaris. 4. El sistema mostra el panell amb un llistat d'usuaris. i. Si el rol es administrador o empleat se li mostren tots els usuaris. ii. Si el rol es client o propietari se li mostra només la seva conta. 5. El passos 3 i 4 es repeteixen fins que l'usuari decideix finalitzar (el cas d'ús s'acaba).
Escenaris alternatius:

a. ***Actualitzar un usuari***
    1. L'usuari indica que vol modificar un usuari.
    2. El sistema mostra un formulari amb les dades de l'usuari.
    3. L'usuari modifica les dades del formulari.
    4. El sistema enregistra el canvis i torna al pas 4.

Cas d'ús: Gestionar Categories
Actor principal: Administrador i Empleat
Nivell: General
Escenari principal d'èxit: 1. L'usuari s'identifica al sistema. 2. El sistema li mostra el panell d'administració. 3. L'usuari clica sobre el menú de categories. 4. El sistema mostra el panell de categories amb un llistat de les mateixes. 5. El passos 3 i 4 es repeteixen fins que l'usuari decideix finalitzar (el cas d'ús s'acaba).
Escenaris alternatius: a. ***Crear una nova categoria*** 1. L'usuari indica que vol crear una nova categoria. 2. El sistema mostra un formulari d'alta de categoria. 3. L'usuari introdueix les dades al formulari. 4. El sistema enregistra la nova categoria i torna al punt 4. b. ***Eliminar una categoria*** 1. L'usuari indica que vol eliminar una categoria. i. Si l'usuari es administrador i la categoria no té cap propietat relacionada, el sistema elimina la categoria. 2. El sistema torna al punt 4. c. ***Actualitzar una categoria*** 1. L'usuari indica que vol actualitzar una categoria. 2. El sistema mostra un formulari amb les dades de la categoria. 3. L'usuari modifica el nom de la categoria. 4. El sistema enregistra els canvis i torna al punt 4.

* Els rols **client** i **propietari** sols podran realitzar l'acció de llegir categories.

Cas d'ús: Gestionar Vendes

Actor principal: Administrador i Empleat
Nivell: General
Escenari principal d'èxit: 1. L'usuari s'identifica al sistema. 2. El sistema mostra un panell d'administració. 3. L'usuari clica sobre el menú de vendes. 4. El sistema mostra un panell amb un llistat de vendes. i. Si el rol es administrador se li mostra qualsevol venda. ii. Si el rol es empleat se li mostra si la venda es seva. 5. Els passos 3 i 4 es repeteixen fins que l'usuari decideixi finalitzar (el cas d'ús s'acaba).
Escenaris alternatius: ***a. Crear una nova venda*** 1. L'usuari indica que vol crear una nova venda. 2. El sistema mostra un formulari amb les dades de una venda. i. Si el rol es administrador és mostra un desplegable amb tots els empleats (venedors) i un altre amb clients. ii. Si el rol es empleat no és mostra el desplegable d'empleats, però si el de clients. 3. L'usuari introdueix les dades al formulari. 4. El sistema enregistra la nova venda i torna al punt 4. ***b. Actualitzar una venda*** 1. L'usuari indica que vol actualitzar una venda. 2. El sistema mostra un formulari amb les dades de la venda. 3. L'usuari modifica les dades de la venda. 4. El sistema enregistra les noves dades i torna al pas 4.

*Els rols **client (comprador)** i **propietari (venedor)** sols podran llegir vendes relacionades amb ells.

Cas d'ús: Gestionar Visites
Actor principal: Administrador, Empleat, Propietari i Client
Nivell: General
Escenari principal d'èxit: 1. L'usuari s'identifica al sistema. 2. El sistema mostra un panell d'administració.

3. L'usuari clica sobre el menú de visites.
4. El sistema mostra un panell amb un llistat de visites.
    i. Si el rol es administrador se li mostra qualsevol visita.
    ii. Si el rol es empleat, client o propietari se li mostraran les visites relacionades amb ell.
5. Els passos 3 i 4 es repeteixen fins que l'usuari decideixi finalitzar (el cas d'ús s'acaba).

Escenaris alternatius:

a. Crear una nova visita

1. L'usuari indica que vol crear una nova visita.
2. El sistema mostra un formulari amb les dades de una visita.
    i. Si el rol es administrador és mostra, empleat, client, propietat i data.
    ii. Si el rol es empleat és mostra, client, propietat i data.
    iii. Si el rol es client es mostra, propietat i data, l'empleat el proporciona el sistema.
3. L'usuari introdueix les dades al formulari.
4. El sistema enregistra la nova visita i torna al punt 4.

b. Actualitzar una visita

1. L'usuari indica que vol actualitzar una visita.
    i. Si el rol es propietari no pot actualitzar la visita.
2. El sistema mostra un formulari amb les dades de la visita.
3. L'usuari modifica les dades de la visita.
4. El sistema enregistra les noves dades i torna al pas 4.

c. Eliminar una visita

1. L'usuari indica que vol eliminar una visita.
    i. Si el rol es propietari el sistema no elimina la visita.
2. El sistema elimina la visita i torna al pas 4.

Cas d'ús: Gestionar canvis de preu

Actor principal: Administrador, Empleat i Propietari

Nivell: General

Escenari principal d'èxit:

1. L'usuari s'identifica al sistema.
2. El sistema mostra un panell d'administració.

3. L'usuari clica sobre el menú de propietats, canvi de preus.
4. El sistema mostra un panell amb un formulari i un llistat de canvis de preu.
   i. Si el rol es administrador o empleat se li mostra qualsevol canvi de preu.
   ii. Si el rol es propietari se li mostra els canvis de preu de les seves propietats.
5. Els passos 3 i 4 es repeteixen fins que l'usuari decideixi finalitzar (el cas d'ús s'acaba).

Escenaris alternatius:

a. Crear un nou canvi de preu

1. L'usuari indica que vol crear un canvi de preu.
2. El sistema mostra un formulari amb les dades de un canvi de preu.
   i. Si el rol es administrador o empleat se li mostra un desplegable amb totes les propietats.
   ii. Si el rol es propietari se li mostra un desplegable amb les seves propietats.
3. L'usuari introdueix les dades del canvi de preu.
4. El sistema enregistra el canvi de preu i torna al pas 4.

* El rol **client** no pot interactuar amb canvis de preu.

Cas d'ús: Gestionar països i ciutats

Actor principal: Administrador

Nivell: General

Escenari principal d'èxit:

1. L'usuari s'identifica davant el sistema.
2. El sistema li mostra el panell d'administració.
3. L'usuari clica a sobre el menú de països o ciutats.
4. El sistema mostra el panell propietats amb un llistat de les mateixes.
5. Els passos 3 i 4 es repeteixen fins que l'usuari decideix finalitzar (cas d'ús s'acaba).

Escenaris alternatius:

d. Crear una nou país o ciutat

1. L'usuari indica que vol crear una nou recurs.
2. El sistema mostra un formulari d'alta de recurs.
3. L'usuari introdueix les dades al formulari.
4. El sistema enregistra el nou recurs i torna al punt 4.

e. Eliminar un país o ciutat

1. Si l'usuari es administrador pot indicar que vol eliminar un país o ciutat.

i. Si un país no té cap relació amb ciutat o si una ciutat no té cap relació amb propietats, aquesta és podrà eliminar. 2. El sistema elimina la propietat indicada i torna al punt 4. ***f. Actualitzar un país o ciutat*** 1. L'usuari indica que vol actualitzar o modificar un país o ciutat. 2. El sistema mostra un formulari amb les dades del país o ciutat. 3. L'usuari fa els canvis al formulari. 4. El sistema enregistra els canvis i torna al punt 4.

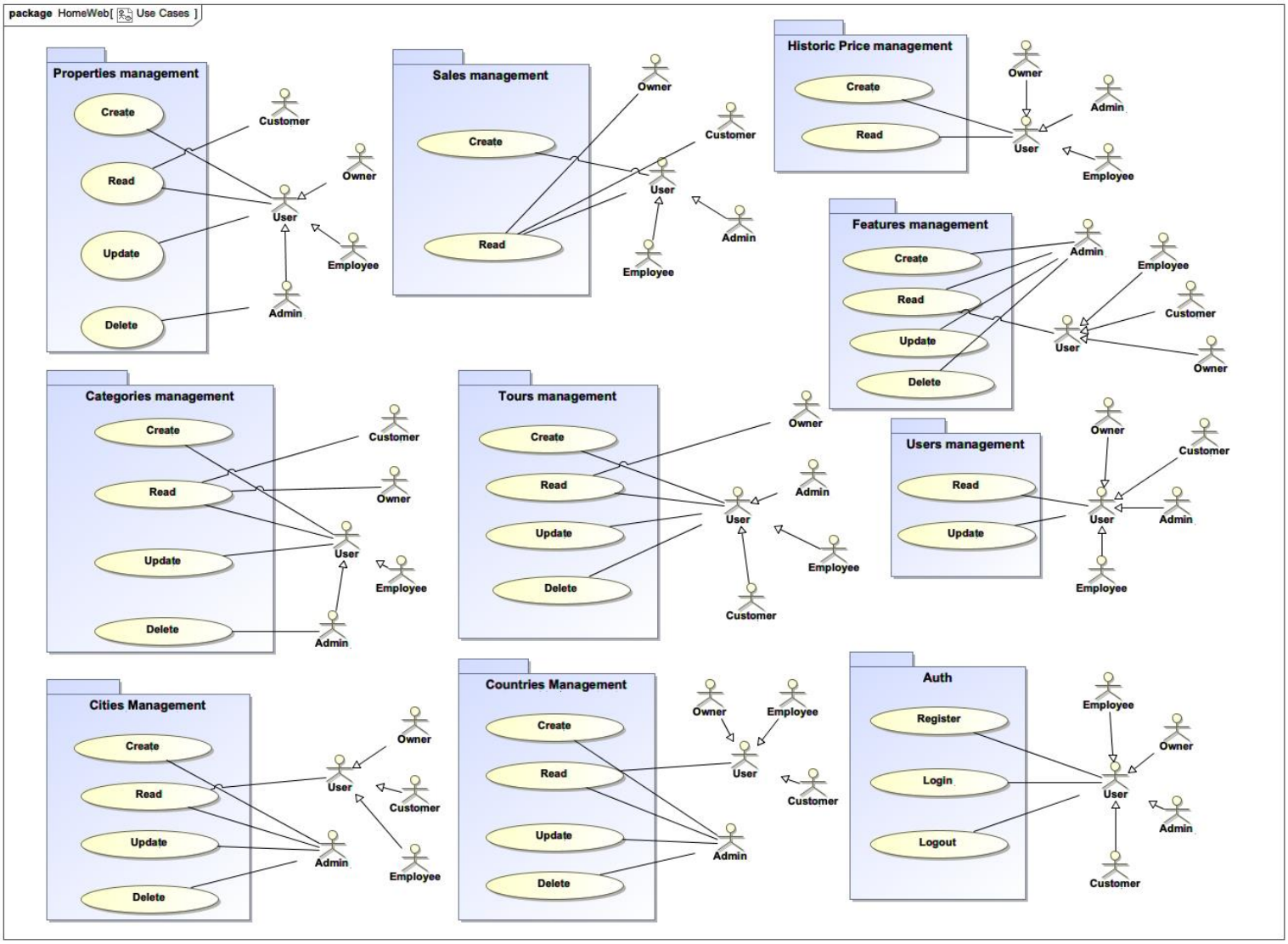
Per aquest cas d'ús els altres rols sols podran llegir els països i/o ciutats.

Cas d'ús: Gestionar característiques
Actor principal: Administrador
Nivell: General
Escenari principal d'èxit: 1. L'usuari s'identifica davant el sistema. 2. El sistema li mostra el panell d'administració. 3. L'usuari clica a sobre el menú de característiques. 4. El sistema mostra el panell de característiques amb un llistat de les mateixes. 5. Els passos 3 i 4 es repeteixen fins que l'usuari decideix finalitzar (cas d'ús s'acaba).
Escenaris alternatius: ***g. Crear una nova característica*** 1. L'usuari indica que vol crear una nova característica. 2. El sistema mostra un formulari d'alta de característica. 3. L'usuari introdueix les dades al formulari. 4. El sistema enregistra la nova característica i torna al punt 4. ***h. Eliminar una característica*** 1. Si l'usuari es administrador pot indicar que vol eliminar una característica. i. Si una característica no té cap relació amb propietats és podrà eliminar. 2. El sistema elimina la característica indicada i torna al punt 4. ***i. Actualitzar una característica*** 1. L'usuari indica que vol actualitzar o modificar una característica. 2. El sistema mostra un formulari amb les dades de la característica. 3. L'usuari fa els canvis al formulari.

4. El sistema enregistra els canvis i torna al punt 4.

Els rols de **client**, **propietari** i **empleat** no tindran permisos per poder gestionar característiques

Especificació amb diagrama UML



Il·lustració 6: Diagrama UML casos d'ús

3. Disseny i arquitectura

En primer lloc cal esmentar i explicar quina és l'arquitectura que governa en la nostra aplicació, també s'explica per què s'ha escollit aquesta arquitectura i quins beneficis ens aporta a curt i llarg termini.

3.1 Arquitectura escollida

Per desenvolupar l'aplicació s'han escollit diferents arquitectures, sent així una arquitectura heterogènia:

⇒ ***Arquitectura de tres capes [9]:*** L'arquitectura en capes descriu l'organització conceptual dels elements del disseny en grups independentment de l'empaquetament o desplegament físic. Cada capa representa un element gran, sovint compost de diversos paquets o subsistemes. Aquesta arquitectura és molt utilitzada en el món de les aplicacions web. En ella es troba una divisió en tres nivells.

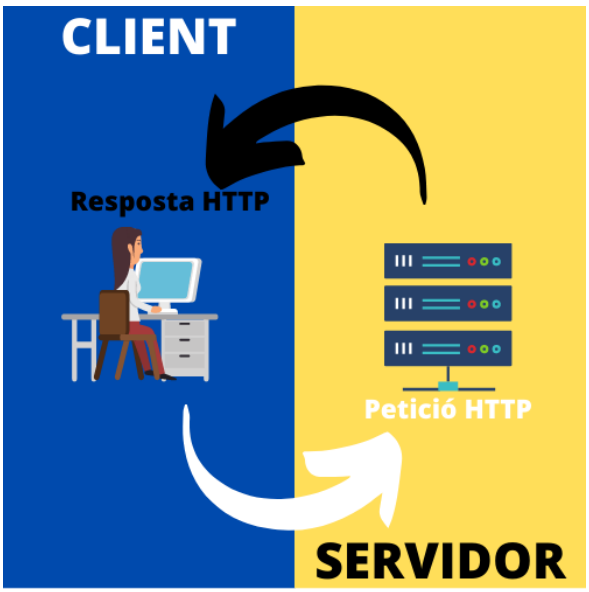
- ***Nivell presentació:***
  - Aquesta capa es l'encarregada de generar la interfície d'usuari depenent de les accions que aquest puguin prendre.
- ***Nivell lògic o de negoci:*** Aquesta capa és l'encarregada de modelar els processos de negoci, necessaris per determinar les regles que imposen les polítiques de la empresa. També és l'encarregada de processar les peticions de l'usuari.
- ***Nivell de dades o integració:*** Aquesta capa és l'encarregada de realitzar la persistència de tota la informació, també s'encarrega del subministrament i emmagatzematge de la informació per el nivell lògic. En aquest cas, sols hi ha una sola font de dades, per tant aquesta sols gestiona aquesta font de dades.

Il·lustració 7: Arquitectura 3 capes

El principal avantatge que proporciona aquesta arquitectura de tres capes és que ens fa escalable l'aplicació, ja que el que s'està fent és aïllar funcionalitats, d'aquesta manera l'aplicació podrà créixer si en un futur fos necessari.

⇒ ***Client-Servidor [10]:*** La principal tasca per a què fou dissenyada aquesta arquitectura és per repartir les tasques entre els proveïdors de serveis (servidors) i els demandants de servies (clients). Consisteix en que els clients fan peticions als servidors, i aquests donen respostes a aquets clients. L'arquitectura client servidor és fer una separació lògica on el servidor no necessàriament s'ha d'executar dins una sola màquina ni necessàriament en un sol programa. És una arquitectura força utilitzada dins el món web.

Els principals avantatges són centralització de control, escalabilitat i fàcil manteniment. Per altra banda, els desavantatges són la congestió de tràfic i els servidors necessiten hardware i software específic.

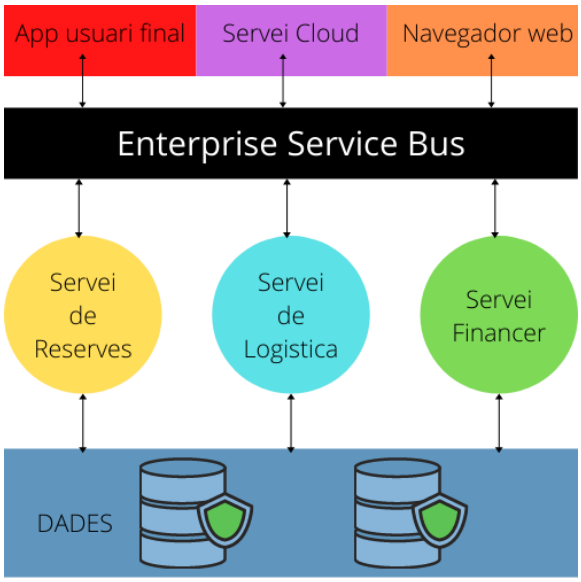


Il·lustració 8: Arquitectura client - servidor

⇒ ***SOA (Service Oriented Architecture) [11]***: Aquesta arquitectura està basada en orientació a serveis, el que fa és descompondre la lògica de negoci en petites porcions funcionals anomenades serveis. El que intenta resoldre SOA és amb un inventari de serveis poder combinar-los per

resoldre les necessitats de negoci. Per tant, el que intenta evitar és haver de construir una aplicació per a cada necessitat.

Els principals avantatges són aplicacions reutilitzables i adaptables, abstracció, baix acoblament, autonomia, capacitat de composició.



Il·lustració 9: Arquitectura orientada a serveis (SOA)

A mode de conclusió, s'han escollit aquestes arquitectures principalment per els beneficis que donen en la implementació i manteniment de l'aplicació, també s'ha de dir que en el món de les aplicacions web aquestes arquitectures són molt utilitzades ja que en un entorn distribuït donen molt bon rendiment i resultats.

3.2 Arquitectura al servidor

La part servidor o també anomenat *backend* s'implementarà amb el *framework Laravel [12],* el qual ens facilitarà la construcció del nostre servidor, ja que dona suport al model de tres capes, client-servidor i a l'arquitectura basada en serveis *SOA*.

Arquitectura MVC [13]

L'arquitectura escollida per implementar el projecte *HomeWeb* és MVC. És un patró d'arquitectura basat en el model de tres capes, és força utilitzat avui en dia per muntar aplicacions de tot tipus. L'essència rau en que el patró separa les dades de l'aplicació:

- ⇒ **Model**: Conté una representació de les dades que maneja el sistema, la seva lògica de negoci i els seus mecanismes de persistència
- ⇒ **Vista**: També anomenat interfície d'usuari, que compon la informació que s'envia al client i els mecanismes d'interacció amb aquest.
- ⇒ **Controlador**: És l'encarregat d'actuar d'intermediari entre el Model i la Vista, gestionant el flux d'informació entre ells i les transformacions per adaptar les dades a les necessitats de cadascú.

El *framework* utilitzat, *Laravel*, dona suport a MVC, separant totalment en tres capes els components que formen cada una d'aquestes capes.

3.3 Arquitectura REST

REST (Representational State Transfer) és una arquitectura de programari relacionat amb el món del desenvolupament web, la principal funció és la seva simplicitat, la qual cosa s'aconsegueix gràcies a una interfície web que utilitza híper mitjans per la representació i transició de la informació. *REST* segueix molts principis de *SOA*.

Fou dissenyada l'any 2000 per Roy Fielding gràcies a la seva tesis doctoral sobre la web. L'autor també es considera un dels principals autors de les especificacions del protocol HTTP. Avui en dia és àmpliament utilitzat per la comunitat de desenvolupament web.

La idea principal de REST es representar els recursos o substantius especialment els noms que se li assignen a cada recurs. Al contrari d'altres especificacions com pot ser *RPC (Remote Procedure Call) [14]* que fa èmfasis a la diversitat d'operacions del protocol o verbs, com per exemple:

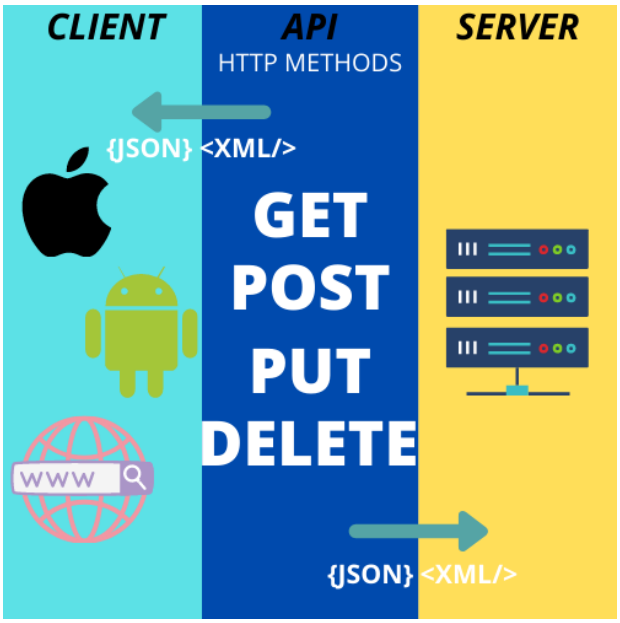
- ⇒ RPC → getUser() (Verb)
- ⇒ REST → User {} (Substantiu)

REST com ja em dit fa èmfasis al recurs en sí, i mitjançant les operacions estàndards HTTP o verbs HTTP (GET, POST, PUT, DELETE) [15] s'efectua la transferència de la representació d'estat del recurs que es vol treballar. Però REST proporciona algunes indicacions sobre com realitzar aquest tipus

d’accions, ja que els verbs HTTP no proporcionen cap mecanisme estàndard per a descobrir recursos, sinó que els desenvolupadors hauran de conèixer bé les URL’s per poder fer les accions a cada tipus de recurs.

La seva principal avantatge és que aporta a la web una major escalabilitat, és a dir, donen suport a un major nombre de components i les interaccions entre ells, aquesta avantatge és gràcies a una sèrie de característiques que presenta la arquitectura REST:

- ⇒ Es un protocol sense estat, degut a que no guarda la informació en el servidor.
- ⇒ Defineix molt bé un conjunt d’operacions, que utilitza sobre tots els recursos, aquestes són les més importants.
    - **GET**: El mètode GET sol·licita una representació de un recurs específic. Les peticions que utilitzen GET sols han de recuperar dades.
    - **POST**: El mètode POST s’utilitza per enviar una entitat a un recurs específic, causant un canvi en el estat o efectes secundaris en el servidor.
    - **PUT**: El mètode PUT reemplaça totes les representacions actuals del recurs de destí amb la càrrega útil de la petició.
    - **DELETE**: El mètode DELETE elimina un recurs específic del servidor.
- ⇒ Utilitza **URIs** (*Uniform Resource Identifier*) úniques, utilitzant una sintaxis universal. Han de seguir unes pautes.
- ⇒ Utilitza híper mitjans per representar la informació, aquets solen ser **HTML**, **XML** o **JSON**. En aquest cas, s’utilitza **JSON** per representar els recursos del sistema.



Il·lustració 10: Arquitectura REST

Mecanisme de seguretat JWT (JSON WEB TOKEN)

Per complir amb el requisit no funcional especificat al apartat de requisits no funcionals, s'ha de dotar la API REST d'un sistema de seguretat que garanteixi l'autenticació dels usuaris i que conseqüentment garanteixi també la integritat i la privadesa del sistema.

Actualment, aquesta seguretat en entorns web es pot implementar gràcies al *JSON WEB TOKEN [16]*, en endavant *JWT*. *JWT* és un estàndard que esta dins el document RFC 7519. Aquest estàndard defineix un mecanisme per poder propagar entre dos parts (client i servidor, o entre APIs) de forma segura, la identitat de un usuari, a més a més d'una sèrie de privilegis o *claims*. Aquests privilegis estan codificats en objectes *JSON (Java Script Object Annotation)* que s'incrusten dins el *payload* o cos d'un missatge que va firmat digitalment.
Aquest token consta de tres parts:

- ⇒ ***Header***: Encapçalament on s'indica l'algorisme de codificació i el tipus de *token*.
- ⇒ ***Payload***: És on apareixen les dades de l'usuari i els seus privilegis, així com tota la informació que vulguem afegir.
- ⇒ ***Signature***: Aquesta firma permet aportar autenticitat. És a dir, poder saber si un usuari és qui diu ser, ja que si el *token* s'ha modificat en qualsevol

moment, no és podrà verificar i podrem saber amb certesa que l'usuari no és el qui diu ser.

Per tant un usuari del nostre sistema quan es vulgui identificar, se li retornarà aquest *JWT* que haurà de guardar d'alguna manera, per poder en un futur enviar en cada petició HTTP mitjançant la capçalera *Authorization: Bearer <JWT>* el *token* que el servidor li havia enviat. Si no és així i no l'envia o aquest és modificat i no es pot validar, la petició al recurs serà denegada immediatament.

En la següent imatge es pot veure com és el funcionament d'aquest mecanisme.

Il·lustració 11: Cicle de vida JWT [22]

<u>Comunicació entre servidor i client</u>

Per poder tenir una comunicació estable i que el client sàpiga en tot moment què ha passat en el servidor i d'aquesta manera pugui prendre una decisió o un altre, s'utilitzaran respostes amb objectes d'anotació JavaScript o *JSON (JavaScript Object Annotation)* i també s'utilitzaran els codis de resposta que proporciona el protocol HTTP, gràcies a aquestes dues mecàniques s'aconsegueix que la comunicació sigui exitosa.

Estructura general JSON:

- ⇒ ***success:*** valor *true* o *false* d'aquesta manera el client sabrà si la petició ha anat bé o no.
- ⇒ ***data:*** Contindrà per les peticions majoritàriament GET les dades sol·licitades.
- ⇒ ***message:*** Contindrà un missatge més específic de la resposta.

Codis de resposta HTTP utilitzats:

CODIS HTTP	QUE INDICA, I QUAN ES FARÀ SERVIR?
200	Indica èxit en la petició. S'utilitzarà majoritàriament a les peticions GET.
201	Indica que el recurs s'ha creat. S'utilitzarà majoritàriament a les peticions POST o PUT.
204	No hi ha contingut per els valors sol·licitats. S'utilitzarà majoritàriament a les peticions PUT o DELETE.
401	Indica que un usuari no està autoritzat. S'utilitzarà per indica si un usuari no esta autenticat al sistema o si un usuari no te permisos per accedir a un determinat servei.
409	Indica que hi ha hagut un conflicte. Quan hi hagi alguna restricció que no es compleix, s'utilitzarà aquest codi.
422	Indica que l'entitat no s'ha pogut processar. Aquest codi s'enviarà quan les dades POST no hagin passat la validació de forma correcte.
500	Indicar error intern del servidor. Per exemple quan hagi algun error en l'acció de persistència a la base de dades, s'utilitzarà aquest codi HTTP.

3.4 Arquitectura MVVM / MVW al client [17]

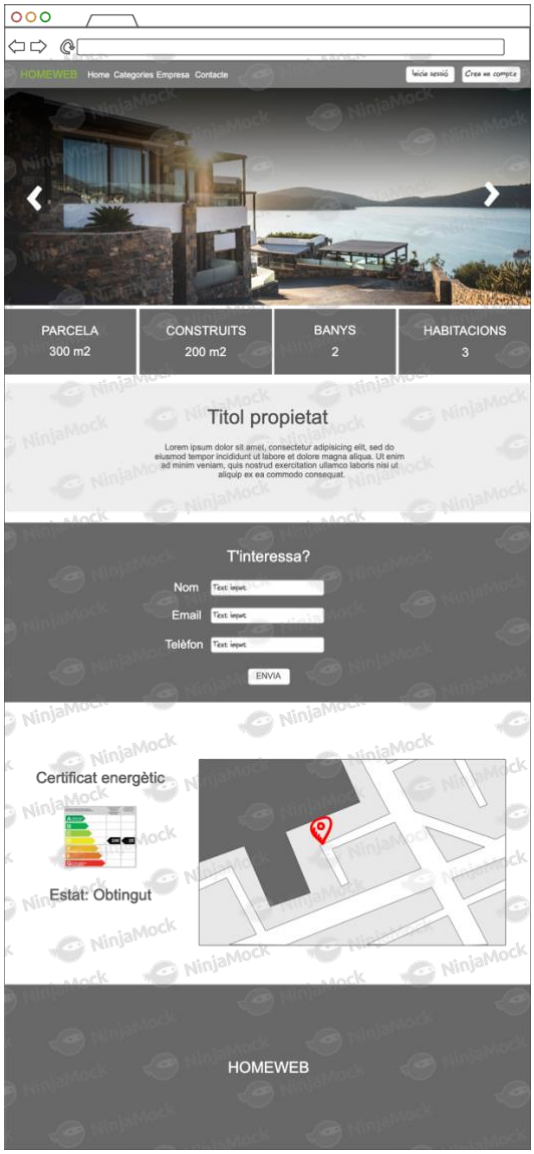
Per a la capa de presentació de l'arquitectura de tres capes, s'ha decidit desacoblar-la totalment del servidor, d'aquesta manera serà el client qui la gestionarà.

Com s'ha dit abans, la part de la **Vista** del model MVC es desacoblarà del servidor o *backend* i serà gestionant per el ***framework Angular [18]*** que es pot dir que segueix una arquitectura MVVM *(mode-view view-model)* o MVW *(model-view-whatever).* Hi ha un gran debat sobre aquest concepte, primer de tot se li diu que segueix MVVM per el *two-way data binding* que aporta Angular, però per altra banda no és tota la comunitat que estigui d'acord amb que sigui MVVM, per això se li anomena també MVW, ja que no es sap molt bé on identificar-lo.

3.5 Inteficies gràfiques i mockups

Web pública

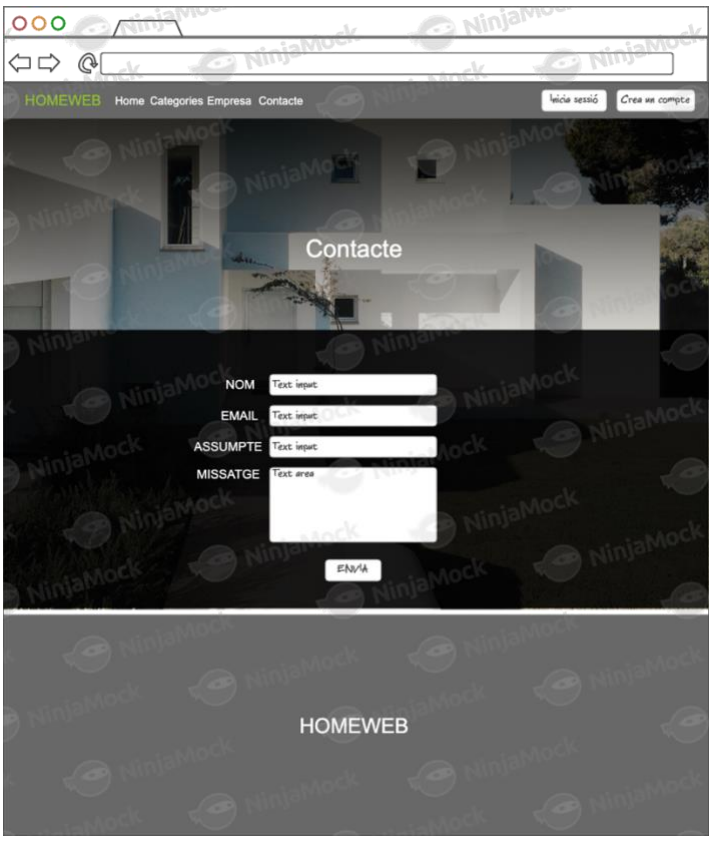
Il·lustració 12: Pàgina inici



Il·lustració 13: Pàgina detall propietat

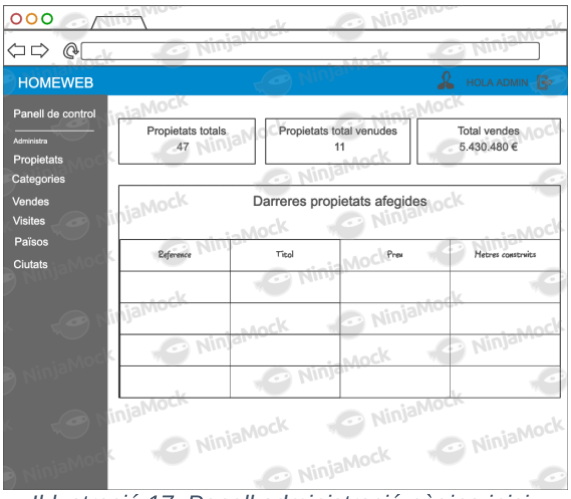
Il·lustració 14: Pàgina categories

Il·lustració 15: Pàgina resultats cerca

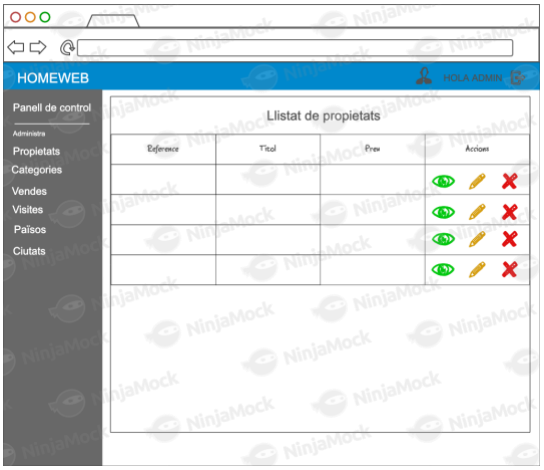


Il·lustració 16: Pàgina contacte

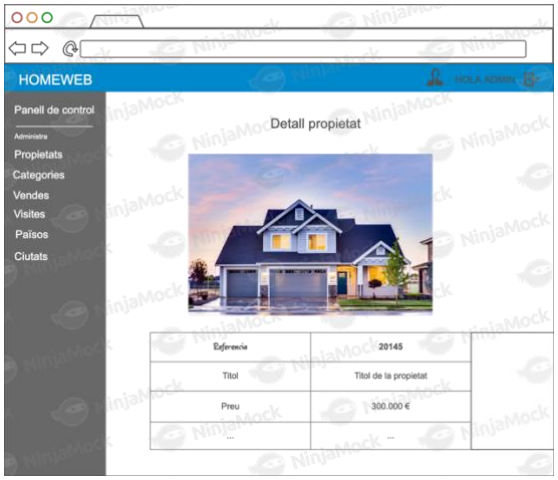
Panell d'administració



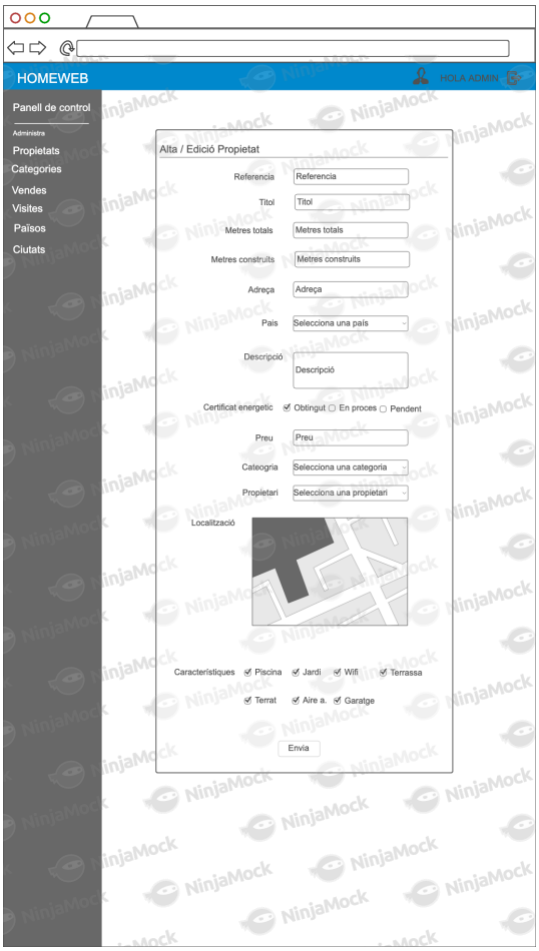
Il·lustració 17: Panell administració pàgina inici



Il·lustració 18: Panell administració pàgina propietats



Il·lustració 19: Panell administració detall propietat



Il·lustració 20: Panell administració formulari crear/editar propietat

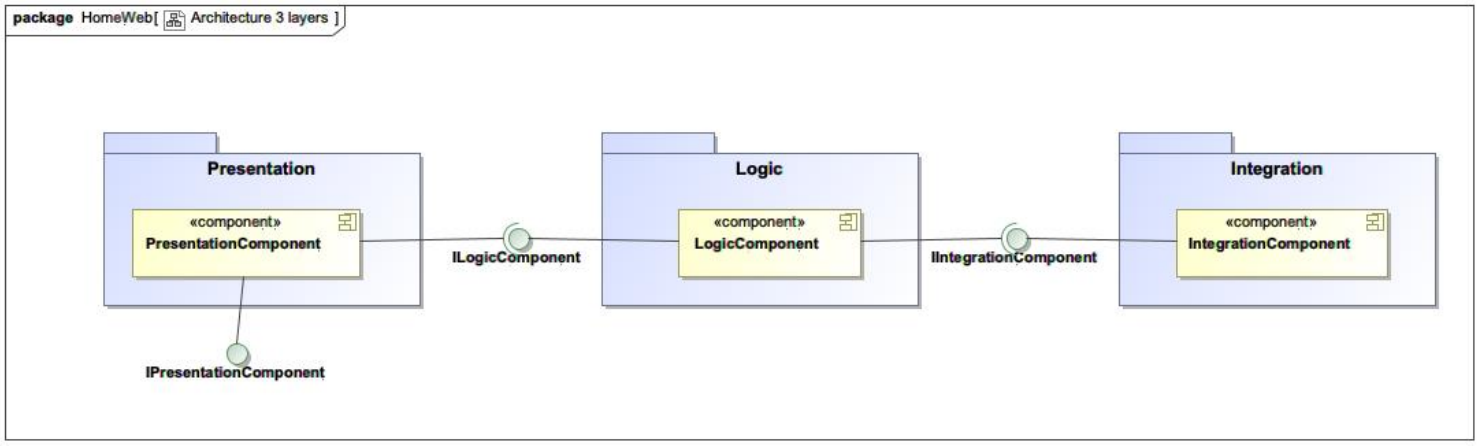
[19]

4. Components i interfícies

En aquest apartat definirem els components del nostre sistema i els seus contractes. Cal dir que fent referència a la PAC 1 on vàrem detallar les fases a seguir, aquesta fase equivaldria a la fase de **Modelització** del ***SWEBOK***.

4.1 Arquitectura de 3 capes

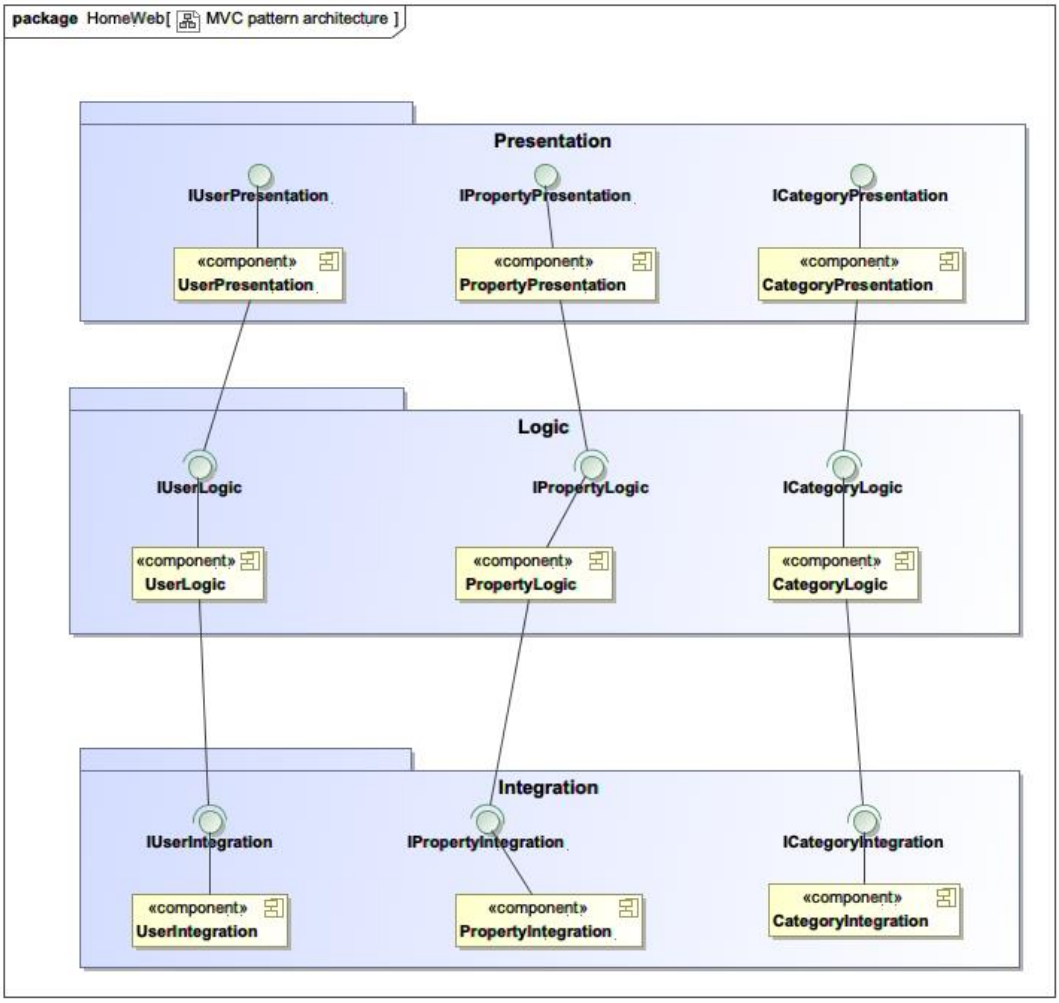
Aquí en un primer disseny es mostra les diferents parts que composen l'arquitectura de manera general. Es pot veure com cada component es comunica immediatament amb la capa que té al costat mitjançant un contracte o interfície.



Il·lustració 21: Diagrama UML arquitectura 3 capes

4.2 Patró d'arquitectura MVC

Seguint l'apartat anterior, es refina el disseny de l'arquitectura amb els components abans esmenats, i es fa un segon disseny exposant com a patró d'arquitectura l'MVC.

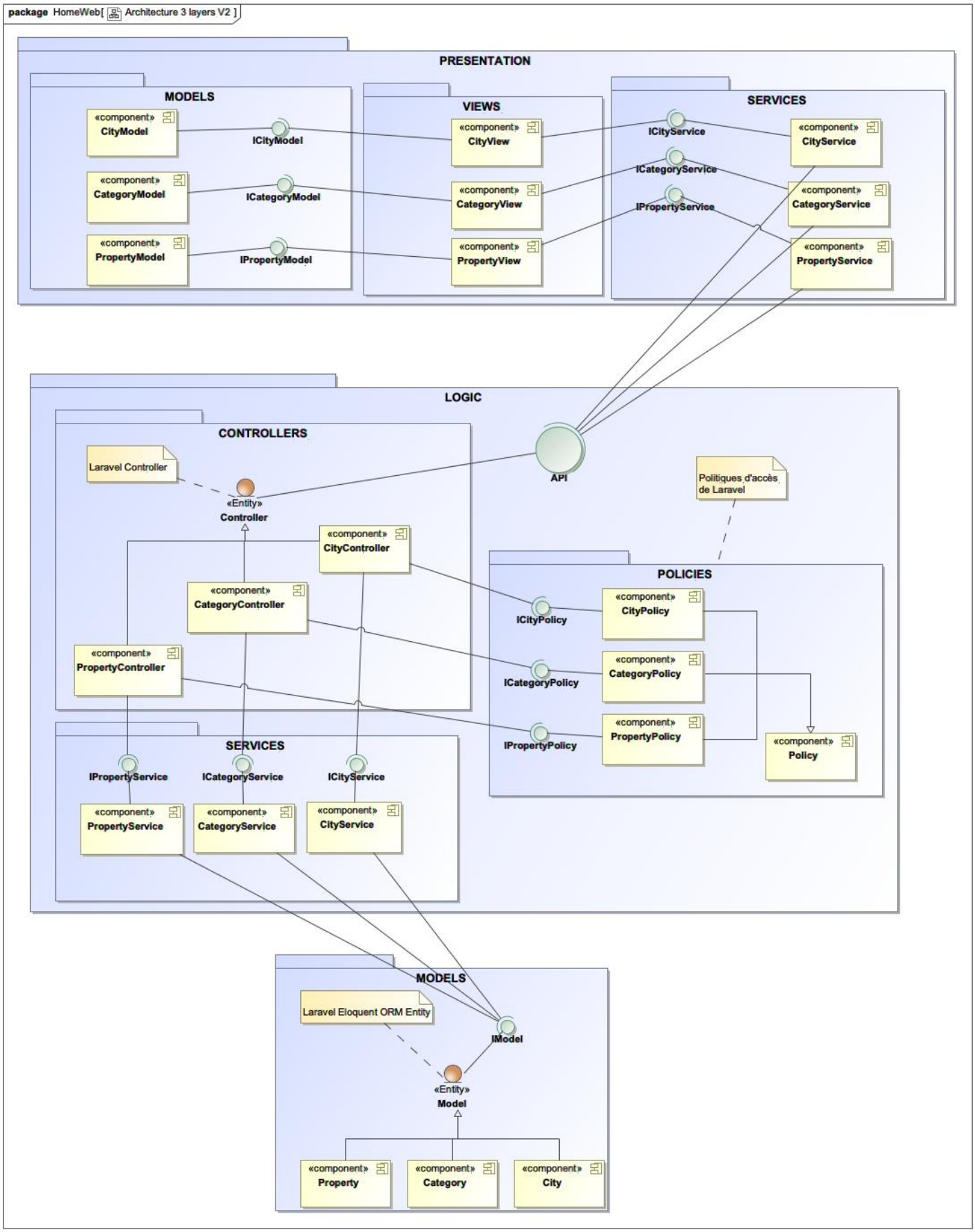


Il·lustració 22: Diagrama UML patró arquitectura MVC

En aquest disseny sols es mostren un nombre determinat de recursos amb els seus respectius components, però d'aquesta manera queda reflectida la idea d'arquitectura amb els components i les interfícies, les quals seran iguala per a tots els altres recursos.

Gràcies als contractes o interfícies, els components es podran entendre a l'hora de realitzar qualsevol acció en el nostre sistema.

Aprofundint un poc més i refinant un poc més el disseny del sistema, es poden determinar els diferents components que es veuen en la següent imatge.



Il·lustració 23: Diagrama UML refinament patró arquitectura MVC

Per una banda, en aquest diagrama de components i les seves interfícies hi ha la part servidor, en la qual hi haurà els models, els controladors, els serveis i les polítiques que mitjançant les pertinents interfícies es comunicaran i podran realitzar les tasques pertinents.

Per altra banda, tenim el ***package MODELS*** on hi haurà les classes que s'encarregaran de interaccionar amb la base de dades, i per altra hi ha el ***package SERVICES***, on els components s'encarregaran d'interaccionar amb els models, i finalment poder donar un servei als components del ***package CONTROLLERS***, tot implementant regles de negoci o lògica de negoci. Aquests serveis gràcies a la injecció de dependències s'utilitzaran dins els controladors sense necessitat de instanciar aquests components serveis cada cop que es requereixin.

Els controladors seran els encarregats de rebre les peticions del client i executar la lògica de negoci necessària per poder fer front al que està demanant el client, com s'ha dit abans farà servir els serveis per poder abastir-se de les dades necessàries per donar resposta a les peticions. Aquests mateixos controladors s'abastiran de les funcionalitats que els i ofereixen els components del ***package POLICIES***, les quals gestionaran els permisos que tenen els diferents rols del sistema.

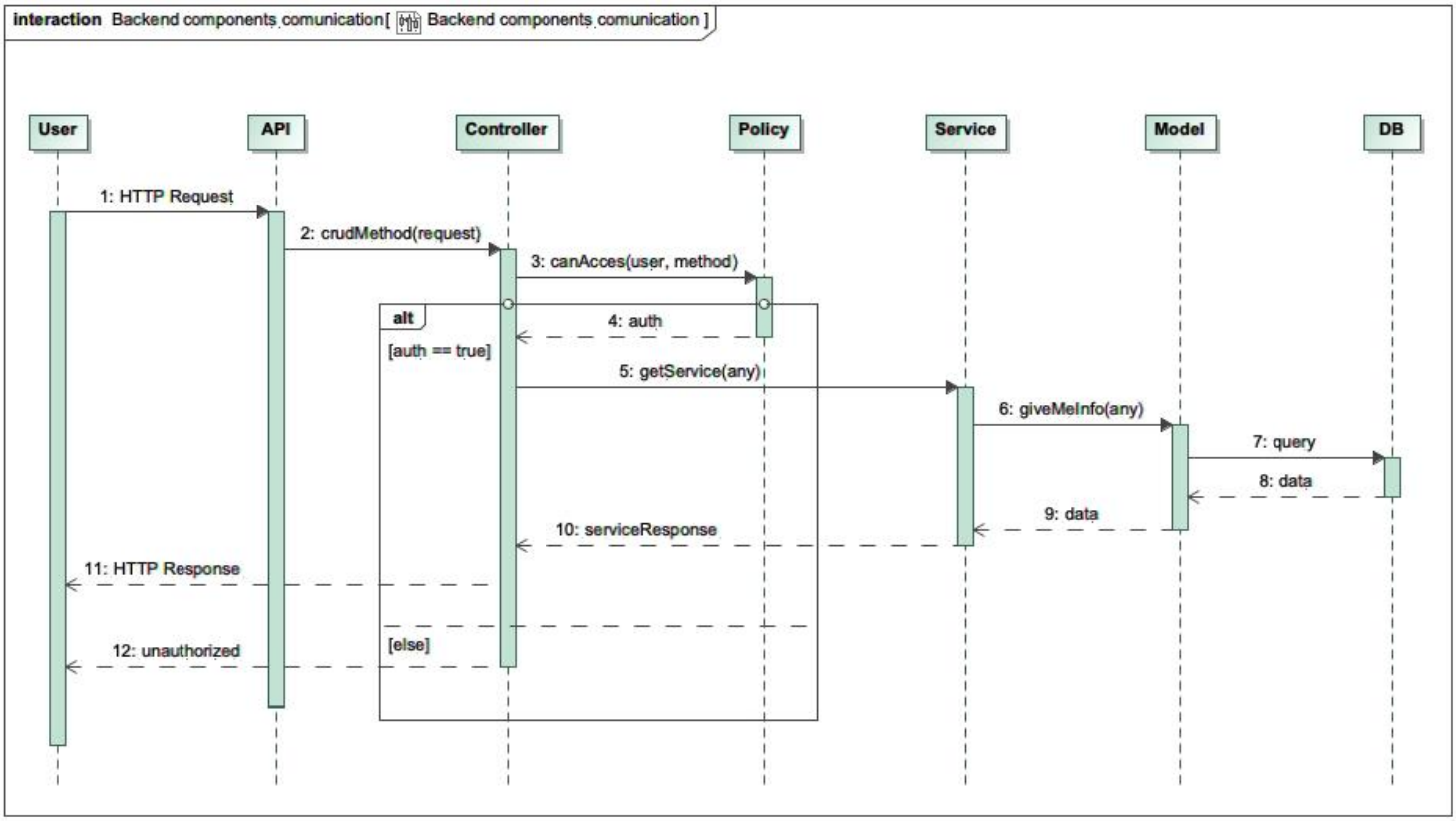
I per altra banda, tenim el ***package PRESENTATION***, el qual representa la part client o aplicació client, on hi haurà una sèrie de components separats per paquets que gràcies a la interfície de l'API es podran abastir de les dades necessàries per poder donar les diferents funcionalitats definides per els diferents tipus d'usuaris del sistema. En ell es pot trobar el ***package SERVICES***, que seran els encarregats de realitzar les peticions a la nostra API, per poder abastir de dades el nostre *frontend*.

Seguidament hi ha un ***package MODELS*** on hi ha els diferents models de dades definits que s'utilitzaran per gestionar d'una manera més fàcil aquestes dades que venen del *backend*. I per finalitzar, hi ha el ***package VIEWS*** que seran els

encarregats d'utilitzar les interfícies tant dels serveis com dels models, i d'aquesta manera poder requerir la informació i presentar-la a l'usuari.

4.3 Comunicació components backend

A continuació es mostra un diagrama de seqüència a alt nivell on es pot fer una idea de com és la comunicació o interacció entre els diferents components del servidor. És a dir, com es comuniquen en termes generals els components que allà hi resideixen.

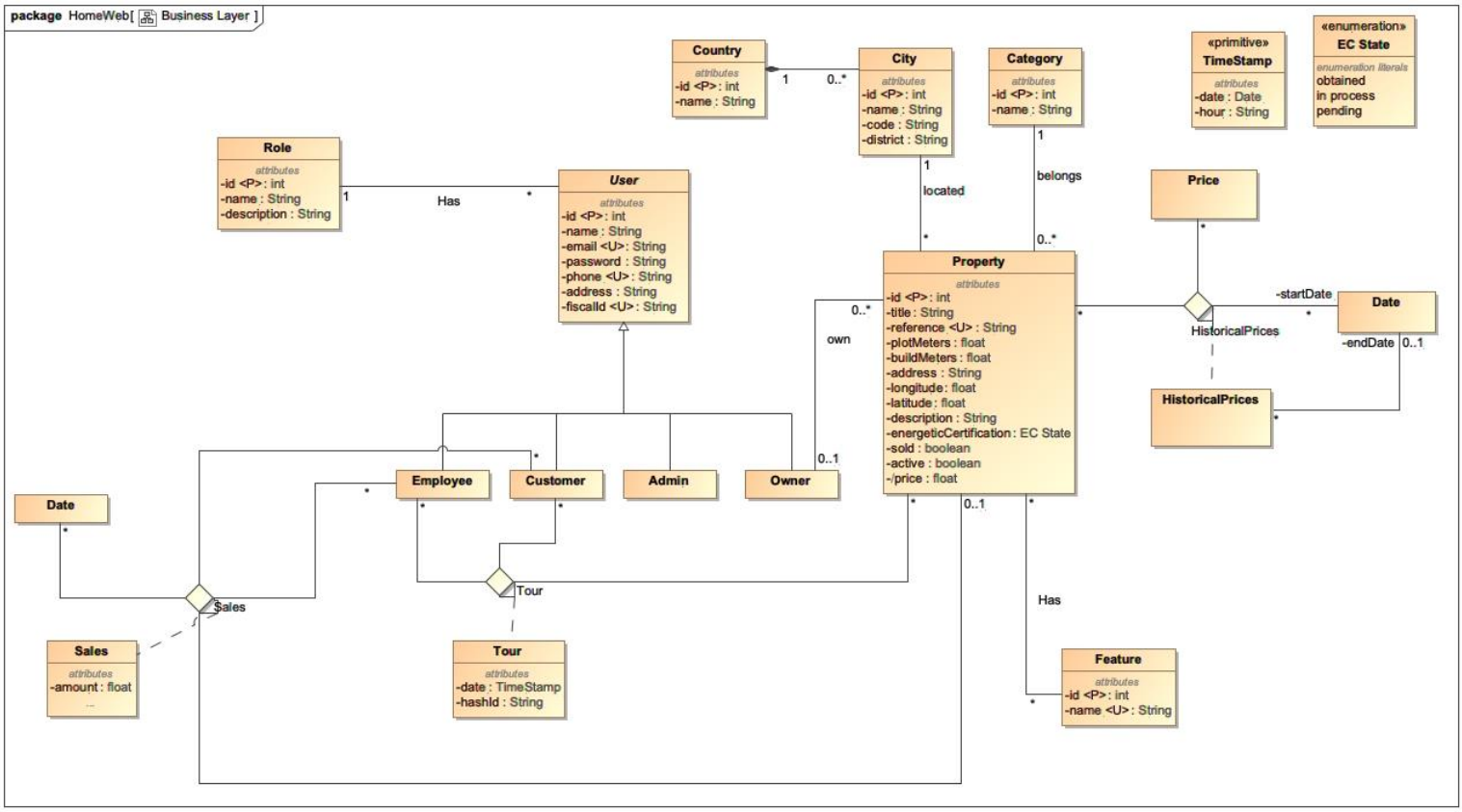


Il·lustració 24: Diagrama de seqüència backend

Aquest diagrama representa d'una manera gràfica i en termes generals la interacció que tindran els components del *backend*.

4.4 Disseny base de dades

En la capa de persistència o dades de la nostra arquitectura de tres capes, hi ha ubicada les fonts de dades. En aquest cas la font serà una base de dades ubicada al costat del servidor. S'ha dissenyat mitjançant un diagrama de classes amb l'especificació UML, on es representa la informació que haurà de persistir la nostra aplicació.



Il·lustració 25: Diagrama classes model de dades

<u>Aspectes a tenir en compte de les relacions n-àries</u>

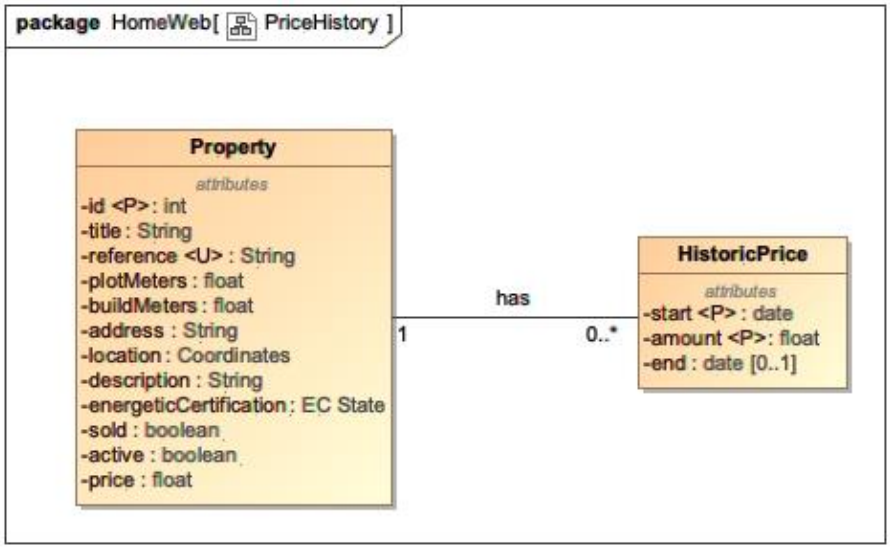
⇒ **La relació n-ària entre *Price, HistoricalPrices i Date***, és una relació que al final permetrà portar un històric cronològic dels canvis de preu que s'han anat generat al llarg de la vida de la propietat. En ella es troba una data d'inici on s'indica quan es va realitzar aquest canvi, i una data de fi que al crear el canvi de preu quedarà sense valor, indicant que es l'últim preu registrat d'aquella propietat, en canvi quan s'introdueixi un altre canvi de preu, aquesta data fi s'haurà d'establir amb el mateix valor de la data d'inici del nou canvi de preu.

⇒ **La relació n-ària *Tour***, la qual representa una visita entre un client i un empleat a una propietat en un determinat dia i hora. En ella s'haurà de tenir en compte que mai es podrà reservar en una mateixa hora una visita a una propietat, i haurà que tenir en compte que tant client com empleat no tinguin compromisos el dia que es vol reservar una visita.

⇒ **La relació n-ària *Sales*,** aquesta relació representa les vendes de les propietats que hi ha dins els sistema, entre un *empleat* (venedor) i un *customer* (comprador) en un dia concret. En ella es representaran les vendes només de les propietats que tenim registrades al sistema, per tant un cop una propietat s'hagi venut ja no estarà disponible per a la seva venda de nou.

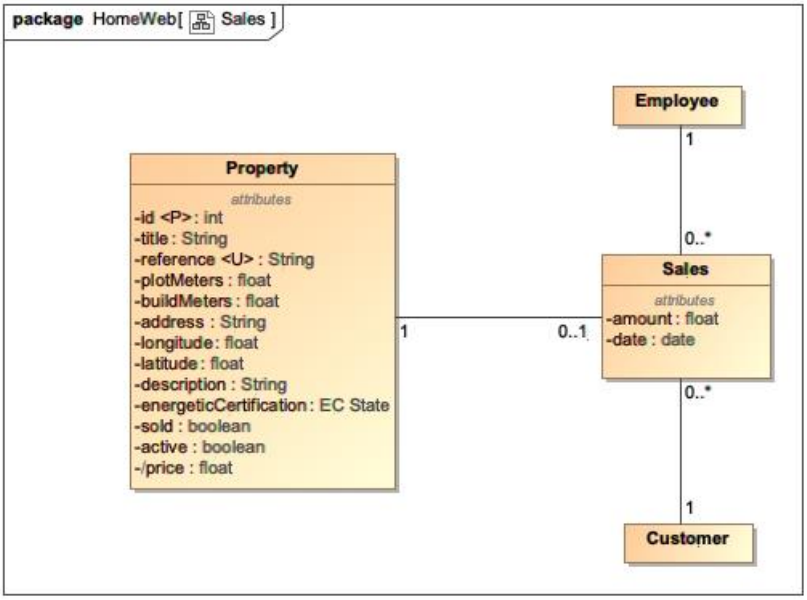
Refinament relacions n-àries

Si refinem més el disseny de la nostra capa de dades podem dir que, per exemple, l'històric de preus que mantindrà persistit queda de la següent manera.



Il·lustració 26: Diagrama de classes refinament relació n-ària històric de preus

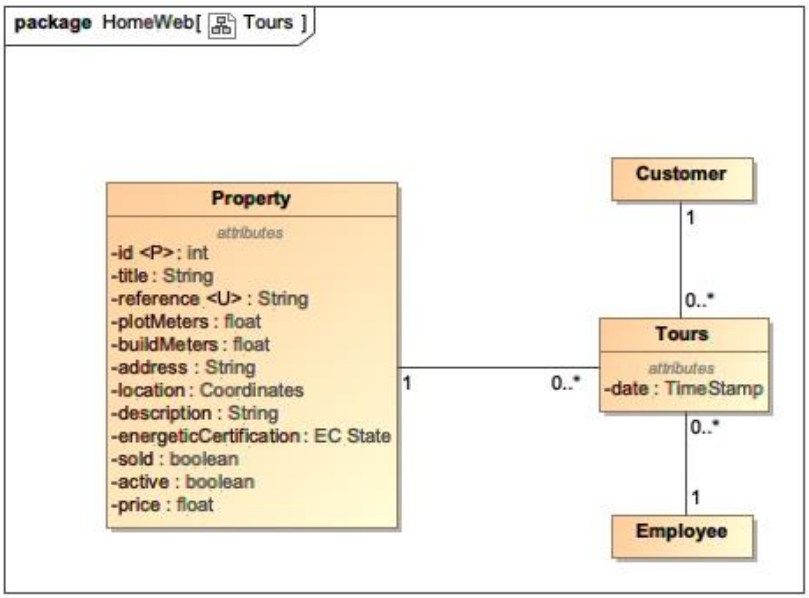
Seguint aquest disseny mai hi podrà haver una mateixa propietat amb el mateix preu en una mateixa data, d'aquesta manera podrem guardar un històric de preus sense repetir informació. També s'ha de dir que una restricció que haurà de satisfer és que la data d'inici del canvi de preu a introduir no podrà ser menor que la última data fi del darrer canvi. Perquè hi hagi coherència la data d'inici mai podrà ser major que la data fi en un canvi de preu.



Il·lustració 27: Diagrama de classes refinament relació n-ària vendes

Per a la relació n-ària que tenim en el disseny vendes, si es refina més es podrà veure com una propietat sols es podrà vendre un cop, un comprador o (*customer)* podrà comprar més de una propietat, i un venedor (*employee*) podrà realitzar diferents vendes.

I en quan a la relació n-ària de visites.



Il·lustració 28: Diagrama de classes refinament relació n-ària visites

Si es refina el disseny, es pot veure com una propietat podrà ser visitada en més d'una ocasió, un client podrà anar a fer visites a diferents propietats, i en una mateixa propietat per a diferent hora, i un empleat igual podrà realitzar visites a

diferents propietats, però a una mateixa propietat sols podrà fer-hi visites a diferents hores.

5. Implementació

Arribats al punt de la implementació, a continuació s'explica com s'ha implementat aquesta aplicació i quines tecnologies s'han utilitzat per a la seva construcció. També es fa referència a la metodologia utilitzada i per al desenvolupament i a més a més es fa una pinzellada amb Docker [21].

Finalment, es descriu el temps dedicat amb un resum on es poden veure les hores de dedicació per a cada part, tant el *backend* com el *frontend*.

5.1 Implementació amb Angular i Laravel

Frontend

El frontend s'ha implementat amb Angular v11.0.1 i Bootstrap v4.5. Del model de tres capes inicial de la nostra arquitectura s'ha decidit desacoblar la capa de presentació i generar-la amb Angular i Bootstrap per donar estils.

S'han utilitzat els següents elements que proporciona Angular:

- ⇒ Components
- ⇒ Serveis
- ⇒ Models de dades o interfícies

Les pàgines de l'aplicació que es poden visualitzar estan compostes per components, artefacte que proporciona Angular per construir pàgines web SPA. També s'han utilitzat els serveis per definir els mètodes que faran les peticions al backend, per obtenir i enviar dades. El format d'intercanvi entre backend i frontend es fa en format JSON.

Per a controlar l'accés a certes pàgines restringides, com per exemple el panell d'administració, s'han utilitzat el *Guards* que serveixen de filtre. També s'han utilitzat

També s'han implementat gràfics per donar una informació més clara a l'usuari, i els mapes per determinar i visualitzar les ubicacions de les propietats. Per donar format a les dates s'ha utilitzat una dependència dissenyada expressament per això.

L'usuari quan realitza una acció l'aplicació li dona feedback, informació necessària perquè el mateix usuari sàpiga si s'ha realitzat bé o no l'acció que ha requerit. Per exemple quan s'acaba de crear o modificar un recurs el sistema llança un modal amb feedback, per altre banda quan volem eliminar un recurs, el sistema mostrà un modal de confirmació per avisar a l'usuari si esta segur del que vol fer.

Per a realitzar certes funcionalitats s'han utilitzats algunes dependències, aquestes s'han instal·lat amb el paquet de dependències de *Node npm*, aquí es detallen algunes:

- ⇒ https://github.com/SebastianM/angular-google-maps
  - Component per Angular que permet treballar amb Google Maps.
- ⇒ https://github.com/FortAwesome/angular-fontawesome
  - Component per Angular que permet utilitzar icones.
- ⇒ https://github.com/ng-bootstrap/ng-bootstrap
  - Widgets per Angular construïts amb Bootstrap.
- ⇒ https://github.com/moment/moment
  - Llibreria per manipular dates i temps en JavaScript.
- ⇒ https://github.com/valor-software/ng2-charts
  - Components per Angular que permeten crear gràfics.
- ⇒ https://github.com/hbaltz/ng-toggle-button
  - Component per Angular que permet crear un *toggle switch button*.
- ⇒ https://github.com/sweetalert2/ngx-sweetalert2
  - Component per Angular, dona suport per la creació de modals.

Backend

El backend s'ha implementat amb *Laravel* v8.0, la capa de negoci, i la d'integració amb *l'ORM* que porta *Laravel*, *Eloquent*, i com a font de dades s'utilitza *MySQL* v5.7.

Primer de tot s'han definit les taules i els models de dades, en *Laravel* les taules les definim amb les *migrations* i si es vol poblar-les s'utilitzen els *seeders*. Per altre banda per els models de dades, es generen per representar cada registre de dins cada taula de la base de dades, i gràcies a ells es podran realitzar les accions de inserció, modificació, recuperació i eliminació de la base de dades.

Seguidament s'han generat els controladors, un per cada model de dades, ens ells es gestionarà la lògica de negoci.

A partir dels controladors es generen els *endpoints* que equivalen als mètodes que tenen els controladors, per tant quan una petició arribi a la *API* serà dirigida al mètode del controlador adient.

Al mateix temps s'han implementat les polítiques de *Laravel*, que equivalen als permisos de cada acció sobre cada rol d'usuari.

Per als recursos categories i propietats, els *endpoints* de crear s'han codificat aposta per pujar una imatge del recurs, per tant el *frontend* fa peticions utilitzat formularis amb codificació *multipart/form-data*. Aquestes imatges es guarden amb codificació *base64*.

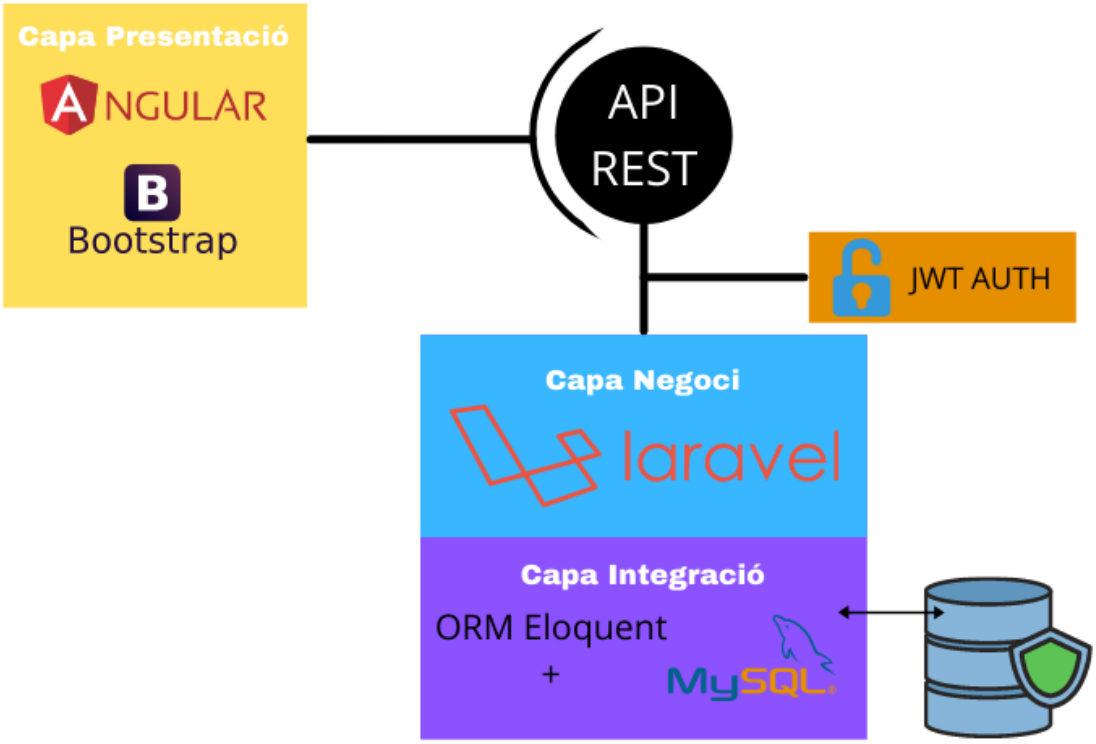
Per a la comunicació entre *backend* i *frontend*, al *backend* les respostes són en format JSON i en elles s'adjunta un codi d'estat HTTP per donar informació al *frontend* de com ha estat el resultat d'una petició HTTP.

Les dependències utilitzades en el backend:

⇒ https://github.com/laravel/passport
  - Ens aporta un servidor OAuth 2.0 i una API per poder gestionar l'autenticació dels usuaris del sistema.

⇒ https://github.com/fruitcake/laravel-cors
  - Un paquet que permet enviar les capçaleres CORS en cada resposta del servidor.

Esquema arquitectura del sistema



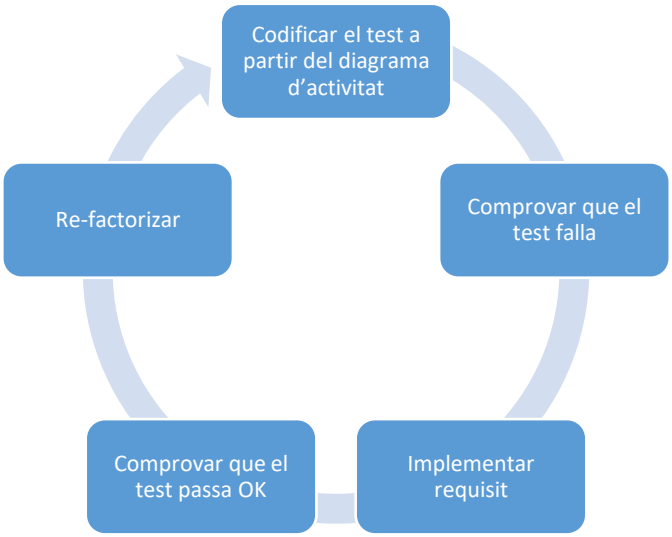
Il·lustració 29 - Arquitectura aplicació web

5.2 Implementació iterativa amb TDD

Per dur a terme la fase d'implementació, s'ha utilitzat la tècnica d'enginyeria de programari TDD.

Un cop definits els requisits i a conseqüència els cassos d'ús, s'han dissenyat tot una sèrie de diagrames d'activitat per definir el flux d'execució. Un cop arribats

aquest punt s'ha codificat el test a partir d'aquest diagrama d'activitat i s'ha seguit el procés que comporta TDD.



Il·lustració 30 - Cicle implementació amb TDD

5.3 Repositoris GitHub [20]

Per poder clonar el projecte aquí tenim els repositoris.

- ⇒ homeweb-backend
  - https://github.com/imollm/homeweb-backend
- ⇒ homeweb-frontend
  - https://github.com/imollm/homeweb-frontend

5.4 Docker

La tendència ara mateix en el món del desenvolupament de programari és utilitzar entorns de virtualització com és Docker, RKT, PodMan, etc...

La principal avantatge d'això es que podem estar treballant amb varies tecnologies al mateix temps, sense haver de fer les pertinents instal·lacions i configuracions a la nostra maquina local. Simplement Docker ens ofereix construir un contenidor aïllat de la nostra màquina local, i dins ells poder executar tot tipus d'aplicacions i comandes.

Ens fa la vida més fàcil a l'hora de desenvolupar programari i fer les pertinents proves.

5.5 Estimació de temps dedicat

A continuació es detalla el temps que s'ha dedicat per a cada projecte. Segons el diagrama de Gantt es va decidir que començaria el mes d'abril, però una vegada iniciat el projecte i ja tenint més clares les idees de que fer i com fer-ho, es va decidir que l'abril seria massa tard i que podria perillar la data d'entrega, per això es va decidir invertir hores del mes de febrer i del març, per tal d'anar més segurs en la implementació.

Es va dedicar unes hores al matí per una part (backend) i unes hores per la tarda (frontend). A les taules que venen a continuació es pot veure com al mes de març es van dedicar 5 h diàries repartides amb 2,5h per projecte, a l'abril i maig el mateix però es va augmentar a 6h diàries repartides amb 3h per projecte.

En quan als dies, es va intentar dedicar el màxim de dies possibles, però com es podrà comprovar hi ha alguns dies que no es va realitzar implementació.

Backend

Mes	Temps dedicat
Febrer	**10h**
Març	**18 dies a 2,5 h diàries = 45h**
Abril	**20 dies a 3h diàries = 60h**
Maig	**20 dies a 3h diàries = 60h**

Frontend

Mes	Temps dedicat
Febrer	**8h**
Març	**18 dies a 2,5 h diàries = 45h**
Abril	**20 dies a 3h diàries = 60h**
Maig	**20 dies a 3h diàries = 60h**

6. Conclusions

En línies generals, el treball m'ha ajudat a millorar i perfeccionar els meus coneixements sobre els aspectes tècnics relacionats amb el món del desenvolupament web, també m'ha ajudat a refinar i depurar els meus coneixements sobre les tecnologies utilitzades, ja que havia construït algunes aplicacions amb elles i ja venia amb algun coneixement, però gràcies al projecte he hagut de revisar molta documentació i visitar tutorial per poder implementar moltes funcionalitats.

Objectius generals:

L'objectiu principal del projecte ha estat aconseguir una aplicació funcional amb les mínimes funcionalitats que es requereixen aquest tipus d'aplicació (rols d'usuari, autenticació, permisos, gestió de recursos, etc). Des de el meu punt de vista, crec que s'ha aconseguit un MVP (Producte viable mínim), que pot satisfer els requisits o necessitats dels clients i proporcionar retroalimentació per al futur desenvolupament.

Objectius específics:

La part de registre i autenticació s'ha realitzat amb correctesa, el sistema autentica amb èxit l'usuari que s'identifica i envia un JWT per a futures peticions al mateix servidor.

S'ha aconseguit també la implementació de permisos sobre les accions que es poden realitzar sobre els recursos del sistema, a partir dels rols d'usuari. No tots els usuaris o rols poden executar o tenir accés a certes rutes de l'API.

S'han utilitzat els codis de resposta HTTP que específica el protocol, d'aquesta manera es molt més fàcil poder testejar i veure el funcionament de la API.

En quan a la base de dades es genera molt fàcilment gràcies a *Laravel* i de la comunicació amb el servidor web és correcte. Persisteix tota la informació que s'envia a les rutes POST i PUT, i igualment recupera la informació necessària

que es requereix mitjançant les rutes GET, tot això es possible gràcies a la terminologia *CRUD*.

Per la part client o interfície gràfica, s'ha aconseguit un mínim de disseny, ja que en un futur es podria millorar la usabilitat i el disseny de la part visual del projecte. L'usuari pot intuir a on s'ha d'anar per poder realitzar l'acció que es requereix, per tant dono per vàlid la usabilitat i navegació per la web amb uns mínims.

Objectius no assolits:

En línies generals s'han assolit tots els objectius que s'havien plantejat al principi. Però ha quedat sense codificar de la part pública de la web el "about us", però en aquesta secció cada client podria posar el que volgués de la seva empresa, no es rellevant.

Per altre banda, m'agradaria remarcar però que potser no s'han executat de la manera més òptima possible, gràcies a la manca de temps, per tant és podria fer una re-factorització de certes funcionalitats i/o maquetació per poder arribar a un nivell de optimització millor.

També es va proposar que el projecte finalitzaria amb un desplegament a Azure mitjançant les *GtiHub Actions* juntament amb les imatges de Docker, val a dir que no s'ha realitzat per la desconeixença dels procediments, la mancança de temps i la complexitat, per tant s'ha de dir que no s'ha arribat a assolir aquest objectiu.

Objectius no contemplats:

Com a puntualització m'agradaria dir que per exemple la implementació de gràfics no es un objectiu que s'hagués plantejat al principi, però durant la implementació es va decidir incloure alguns gràfics amb dades rellevants, per donar una informació més clara al usuari.

7 Línies de futur

L'objectiu principal del projecte ha estat implementar un producte funcional que pogués donar un mínim de funcionalitats. En un futur, el que es podria realitzar sobre aquest projecte és afegir noves funcionalitats que, per exemple, la competència no tingués i que poguessin ser atractives per aquest tipus d'aplicació, o fer millores o modificacions a les funcionalitats que ja hi ha, tot respectant els interessos dels *stakeholders*.

Com que el temps d'implementació ha estat limitat, es podria posar a prova el sistema i el seu rendiment, millorant els temps de resposta, això implicaria obtenir unes mètriques temporals de cada consulta a la base de dades per exemple i millorar-ne els temps de resposta.

Com a funcionalitat extra és podria, un cop implementada la aplicació en varis clients, que les diferents APIs es poguessin comunicar unes amb les altres per a fer una mena de Big Data de propietats i vendes, i d'aquesta manera poder generar informació de valor per a tots els clients.

Gràcies a la construcció de una API, en un futur es podria desenvolupar una aplicació tant per Android com per iOS, perquè el mateix client pogués gestionar el seu negoci des de qualsevol lloc del món.

8. Glossari

UML: Sigles de *Undefined Modeling Language*. És un llenguatge de modelatge de sistemes de programari.

API: Sigles de *Application Programming Interface,* és un conjunt de regles i protocols que serveixen perquè dos aplicacions es puguin comunicar.

REST: Sigles de *Representational State Transfer*. És una arquitectura de programari que permet transferir objectes hipermèdia de manera distribuïda.

TDD: Sigles de *Test Driven Development.* Tècnica d'enginyeria de programari que defineix que el desenvolupament o implementació del programari esta guiat per proves o tests.

JSON: Sigles de *Java Script Object Notation*. És un format utilitzat per l'intercanvi d'informació utilitzant dins el món de les tecnologies web. Es fàcil de llegir i escriure per humans i fàcil de generar i interpretar per maquines. És una alternativa a XML.

JWT: Sigles de *JSON Web Token*. És un estàndard especificat dins el document RFC 7519, que defineix un mecanisme de seguretat mitjançant la generació de un token amb format JSON, d'aquesta manera l'usuari es pot identificar davant el servidor.

SCRUM: És un marc de treball per al desenvolupament àgil de programari. Consisteix en l'aplicació de unes bones pràctiques en el treball de desenvolupament cooperatiu.

BACKEND: Són el conjunt de components de la part servidor que generen la sortida de dades requerida per el client.

FRONTEND: Són el conjunt de components o programari que interactua amb l'usuari.

CRUD: Acrònim de *Create, Read, Update i Delete*, s'utilitza per referir-se a les funcions bàsiques en bases de dades o la capa de persistència en programari.

SWEBOK: És un document creat per la IEEE, que es defineix com una guia al coneixement present en l'àrea de la enginyeria del programari.

ENDPOINT: És un punt final de comunicació com una espècie de node de xarxa. És pot considerar també una interfície exposada per un comunicant o un canal de comunicació.

SPA: Acrònim de *Single Page App*, és una aplicació d'una sola pàgina, per tant no es necessari la recarrega de la pàgina sencera per mostrar contingut de forma dinàmica.

MIDDLEWARE: És una capa de programari que implementa un mecanisme de filtratge de peticions HTTP.

9. Bibliografia

[1] *"Mercado inmobiliario en Menorca – Situación actual"* [Consulta 23/02/2021] – https://blog.portalmenorca.com/situaci%C3%B3n-actual-del-mercado-inmobiliario-en-menorca-520ba3db00fc

[2] *"Scrum (desarrollo de software) – Wikipedia, la enciplopedia libre"* [Consulta 23/02/2021] - https://es.wikipedia.org/wiki/Scrum_(desarrollo_de_software)

[3] *"Desarrollo guiado por pruebas – Wikipedia, la enciclopedia libre"* [Consulta 23/02/2021] - https://es.wikipedia.org/wiki/Desarrollo_guiado_por_pruebas

[4] *"The SWEBOK Guide"* [Consulta 23/02/2021] - http://eds.b.ebscohost.com.biblioteca-uoc.idm.oclc.org/eds/detail/detail?vid=0&sid=a497024d-5468-4895-98cd-82b0d19917c8%40pdc-v-sessmgr03&bdata=JnNpdGU9ZWRzLWxpdmUmc2NvcGU9c2l0ZQ== -AN=edseee.9206209&db=edseee

[5] *"Lenguaje unificado de modelado – Wikipedia, la enciclopedia libre"* [Consulta 23/02/2021] - https://es.wikipedia.org/wiki/Lenguaje_unificado_de_modelado

[6] *"API REST"* [Consulta 24/02/2021] – https://www.redhat.com/es/topics/api/what-is-a-rest-api

[7] *"CRUD – Wikipedia, la enciclopedia libre"* [Consulta 24/02/2021] - https://es.wikipedia.org/wiki/CRUD

[8] *"Interesado (empresa) – Wikipedia, la enciplopedia libre"* [Consulta 24/02/2021] - https://es.wikipedia.org/wiki/Interesado_(empresa)

[9] *"Programación por capas – Wikipedia la enciclopedia libre"* [Consulta 07/03/2021] - https://es.wikipedia.org/wiki/Programaci%C3%B3n_por_capas

[10] *"Cliente-servidor – Wikipedia, la enciclopedia libre"* [Consulta 08/03/2021] - https://es.wikipedia.org/wiki/Cliente-servidor

[11] *"SOA vs. SOAP y REST – Adictos al trabajo"* [Consulta 24/02/2021] - https://www.adictosaltrabajo.com/2014/01/10/soavs-soap-rest/

[12] *"Laravel – The PHP Framework For Web Artisans"* [Consulta 10/03/2021] - https://laravel.com/

[13] *"Modelo vista controlador (MVC). Servicio de Informática ASP ..."* [Consulta 12/03/2021] - https://si.ua.es/es/documentacion/asp-net-mvc-3/1-dia/modelo-vista-controlador-mvc.html

[14] *"Llamada a procedimiento remoto – Wikipedia, la enciclopedia libre"* [Consulta 12/03/2021] - https://es.wikipedia.org/wiki/Llamada_a_procedimiento_remoto

[15] *"Métodos de petición HTTP – HTTP | MDN"* [Consulta 16/03/2021] - https://developer.mozilla.org/es/docs/Web/HTTP/Methods

[16] *"RFC7519 – IETF Tools"* [Consulta 17/03/2021] - https://tools.ietf.org/html/rfc7519

[17] *"¿Qué patron usa Angular? MVC o MVVM | OpenWebinars"* [Consulta 17/03/2021] - https://openwebinars.net/blog/que-patron-usa-angular-mvc-o-mvvm/

[18] *"Angular.io"* [Consulta 17/03/2021] - https://angular.io/

[19] *"NinjaMock online wireframe and mockup tool"* [Consulta 18/03/2021] - https://ninjamock.com/

[20] *"GitHub: Where the world builds software · GitHub"* [Consulta 27/02/2021] - https://github.com/

[21] *"Docker: Empowering App Development for Developers"* [Consulta 10/05/2021] - https://www.docker.com/

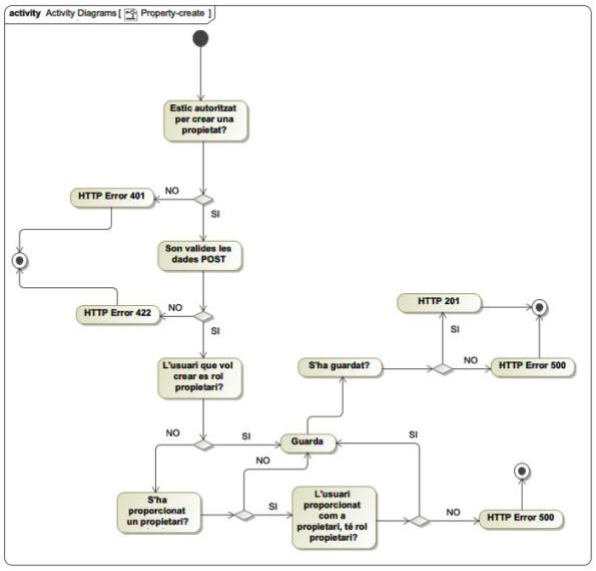
[22] *"Qué es Json Web Token y cómo funciona"* [Consulta 17/03/2021] - https://openwebinars.net/blog/que-es-json-web-token-y-como-funciona/

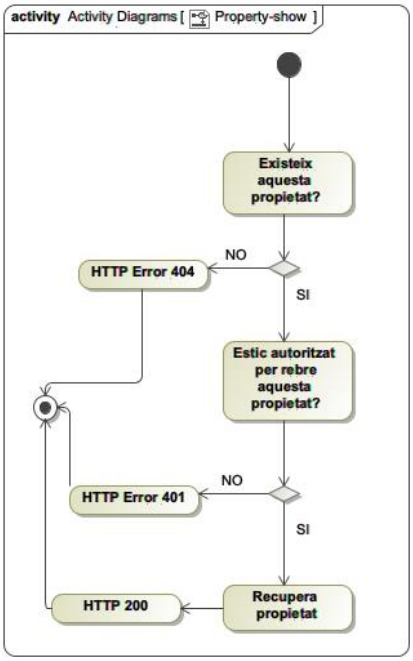
10. Annexos

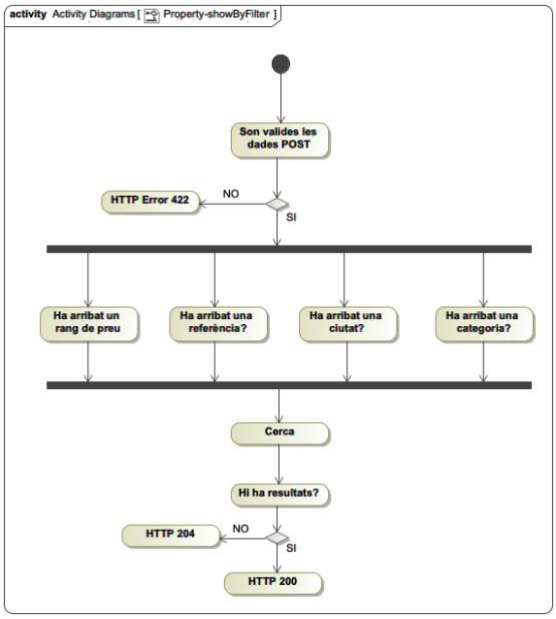
A continuació s'adjunten a aquesta memòria 8 apèndix on es mostren i s'expliquen procediment que s'han seguit, i que han estat fonamentals per el correcte desenvolupament d'aquest projecte.

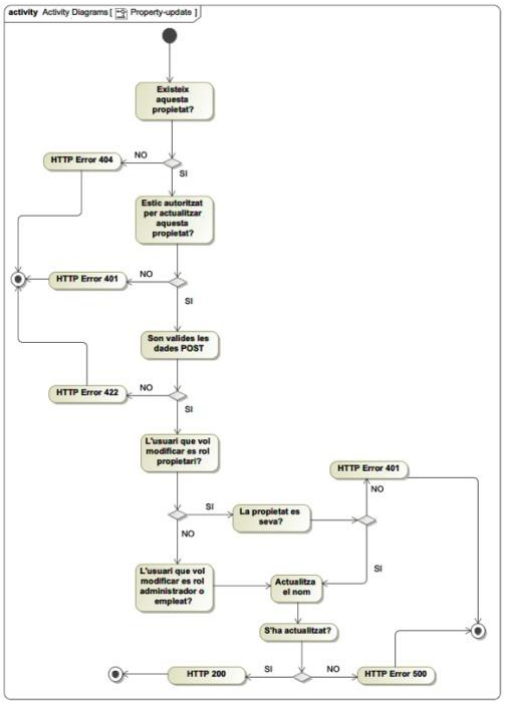
Apèndix 1: Diagrames d'activitat backend

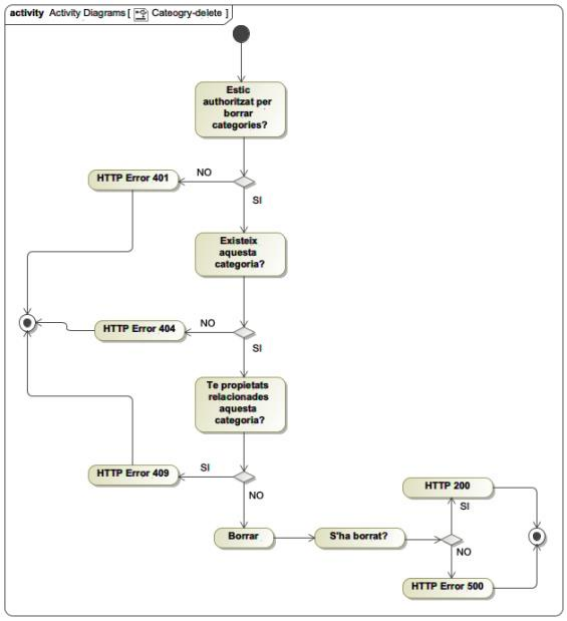
Aquets són els diagrames que s'han dissenyat i que a partir d'ells aplicant TDD s'ha construït el backend. Al títol de cada imatge es descriu el recurs i l'acció.

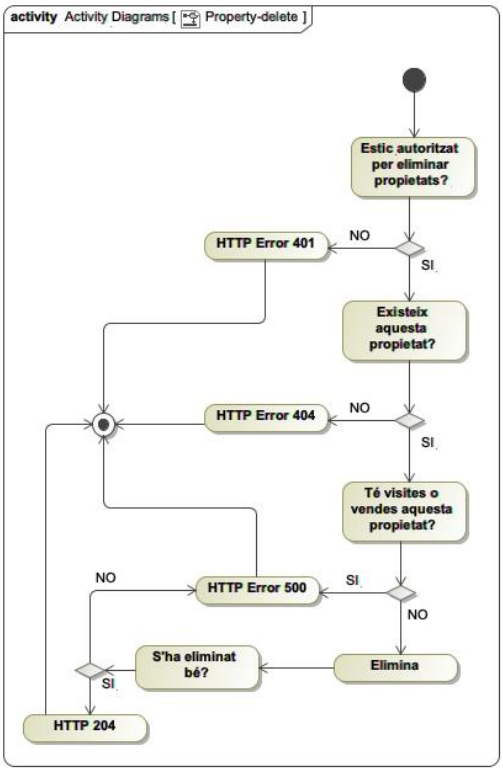


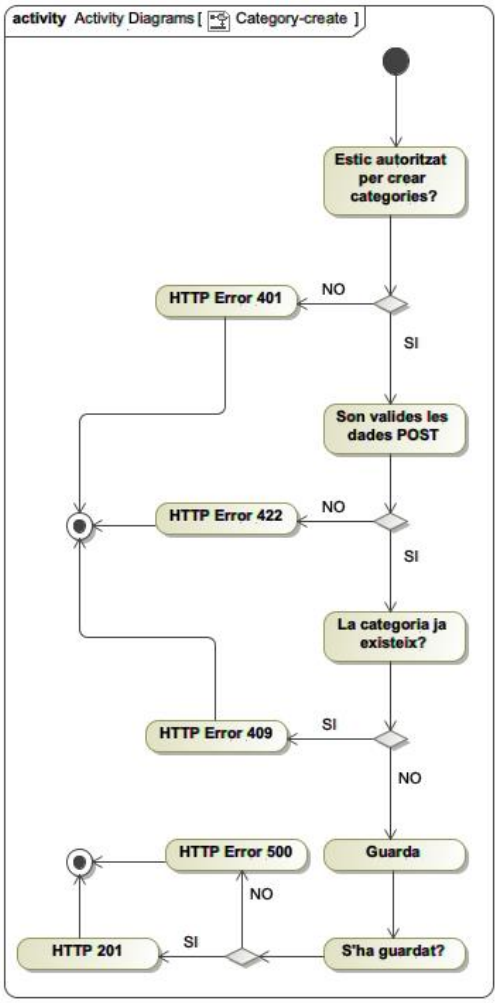


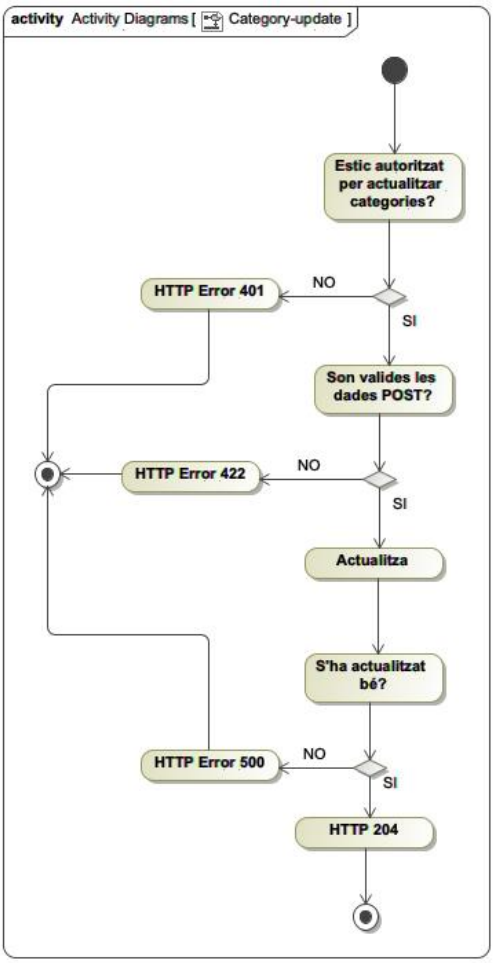


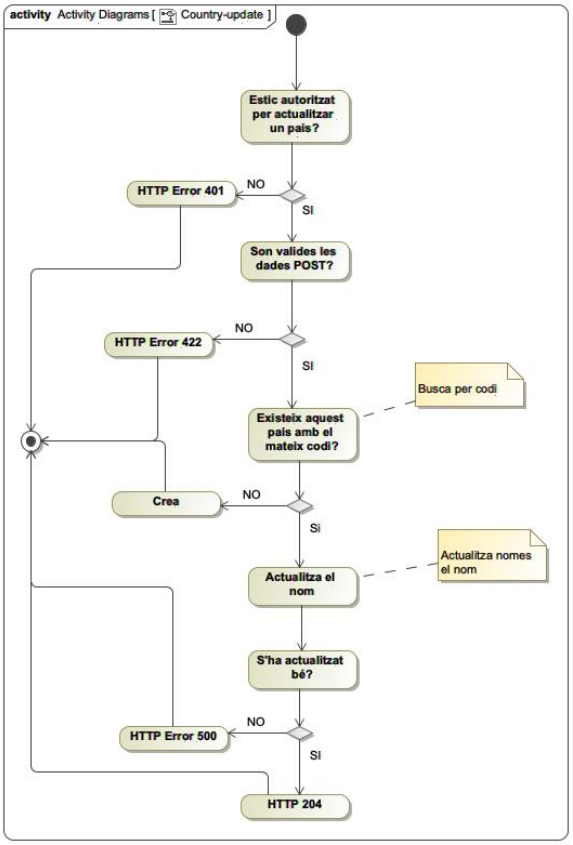


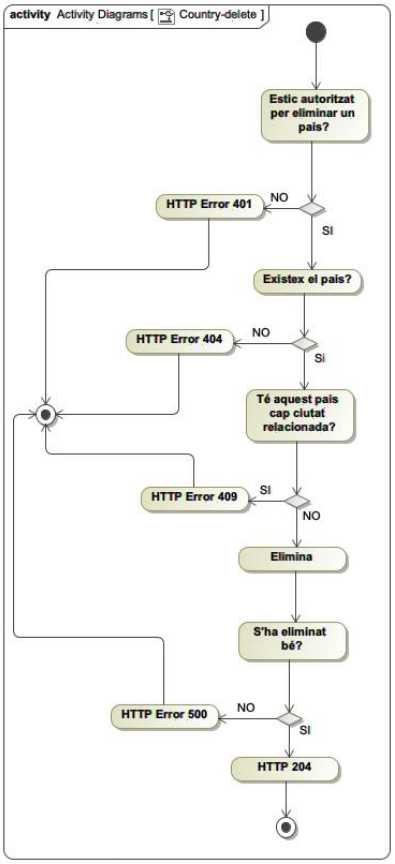

activity Activity Diagrams [Cateogry-delete]
Estic authoritzat per borrar categories?
HTTP Error 401
NO
SI
Existeix aquesta categoria?
HTTP Error 404
NO
SI
Te propietats relacionades aquesta categoria?
HTTP Error 409
SI
NO
HTTP 200
SI
Borrar
S'ha borrat?
NO
HTTP Error 500

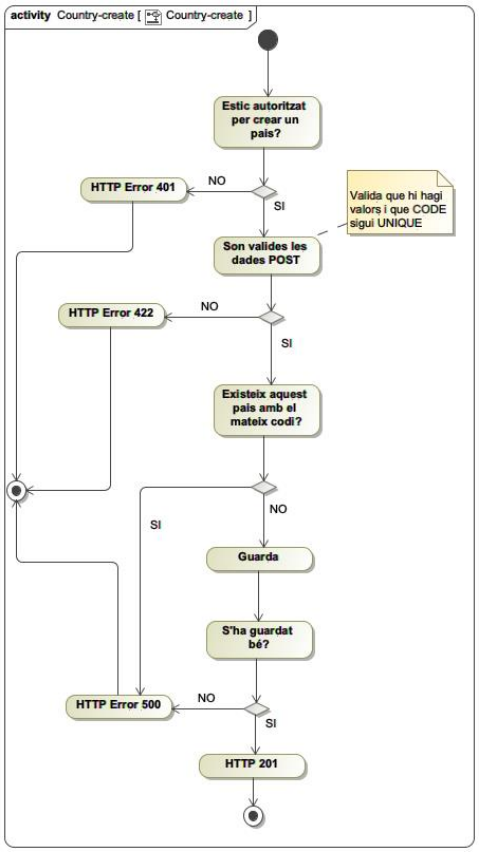

activity Activity Diagrams [Property-delete]
Estic autoritzat per eliminar propietats?
HTTP Error 401
NO
SI
Existeix aquesta propietat?
HTTP Error 404
NO
SI
Té visites o vendes aquesta propietat?
NO
HTTP Error 500
SI
NO
S'ha eliminat bé?
Elimina
SI
HTTP 204

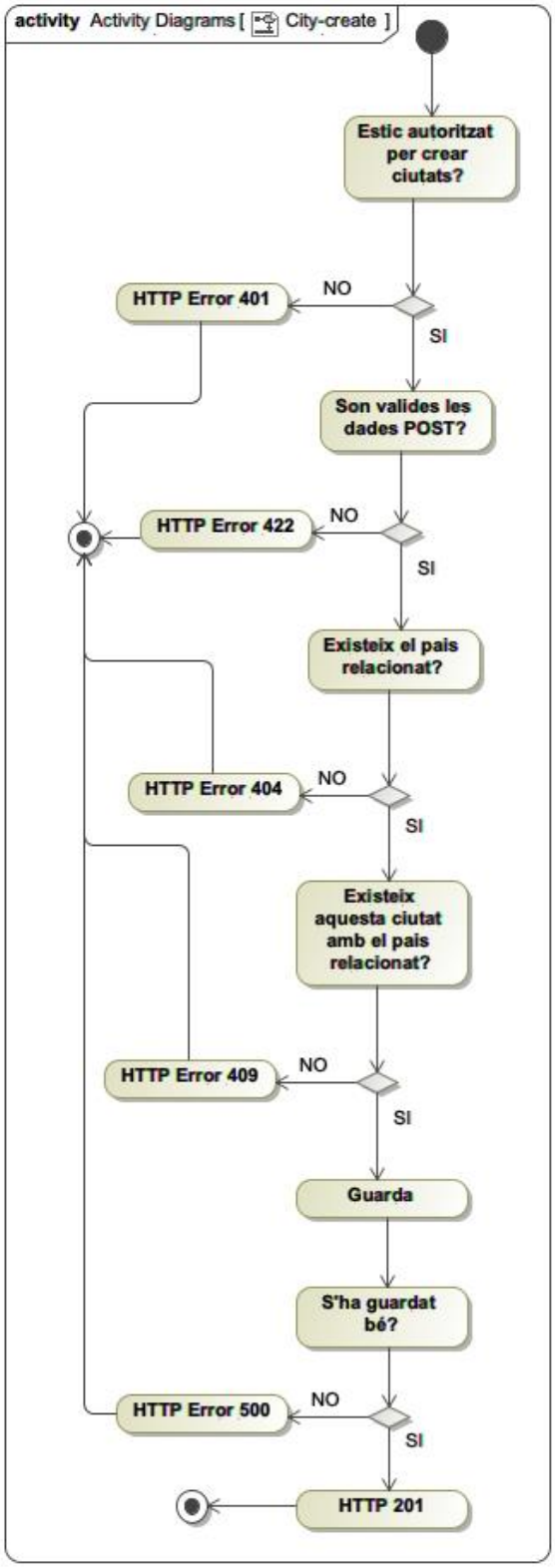

activity Activity Diagrams [Category-create]
Estic autoritzat per crear categories?
HTTP Error 401
NO
SI
Son valides les dades POST
HTTP Error 422
NO
SI
La categoria ja existeix?
HTTP Error 409
SI
NO
HTTP Error 500
Guarda
NO
HTTP 201
SI
S'ha guardat?

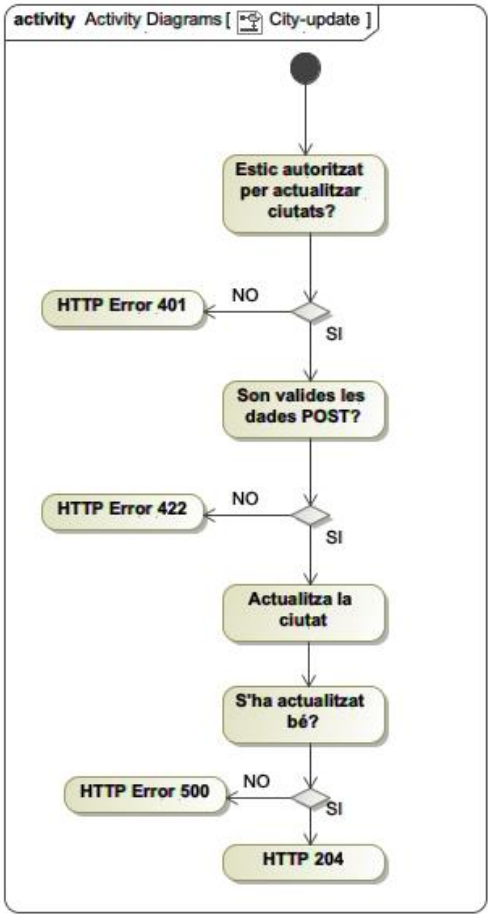

activity Activity Diagrams [Category-update]
Estic autoritzat per actualitzar categories?
HTTP Error 401
NO
SI
Son valides les dades POST?
HTTP Error 422
NO
SI
Actualitza
S'ha actualitzat bé?
HTTP Error 500
NO
SI
HTTP 204

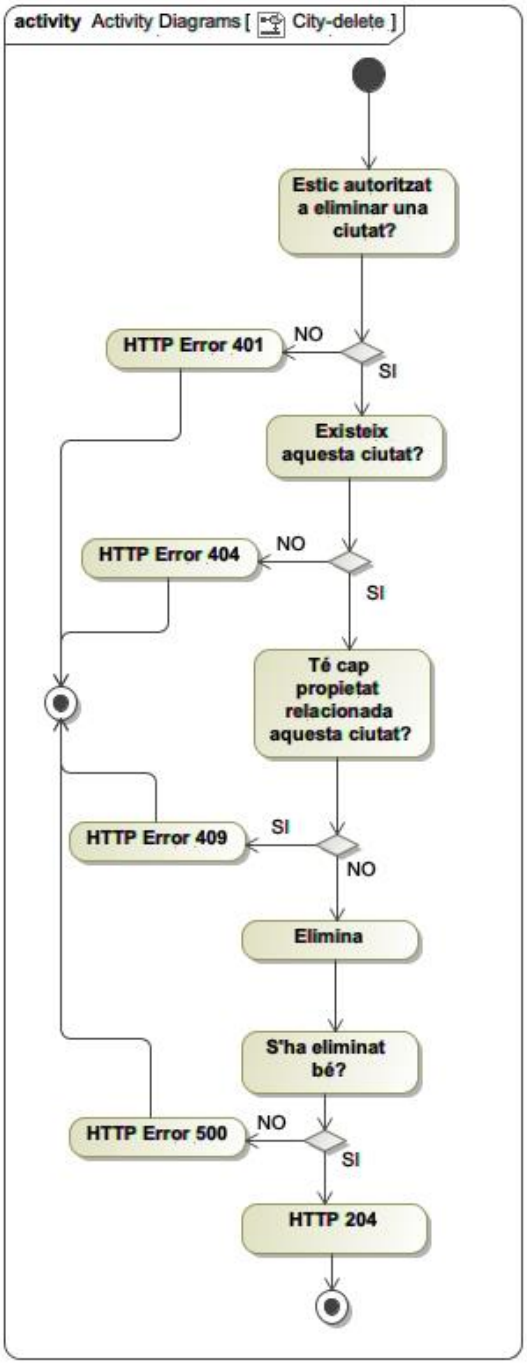

activity Activity Diagrams [Country-update]
Estic autoritzat per actualitzar un pais?
HTTP Error 401
NO
SI
Son valides les dades POST?
HTTP Error 422
NO
SI
Existeix aquest pais amb el mateix codi?
Busca per codi
Crea
NO
Si
Actualitza el nom
Actualitza nomes el nom
S'ha actualitzat bé?
HTTP Error 500
NO
SI
HTTP 204

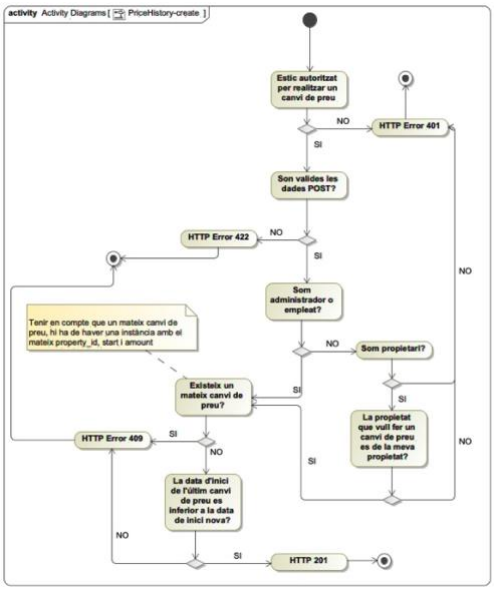

activity Activity Diagrams [Country-delete]
Estic autoritzat per eliminar un pais?
HTTP Error 401
NO
SI
Existex el pais?
HTTP Error 404
NO
Si
Té aquest pais cap ciutat relacionada?
HTTP Error 409
SI
NO
Elimina
S'ha eliminat bé?
HTTP Error 500
NO
SI
HTTP 204

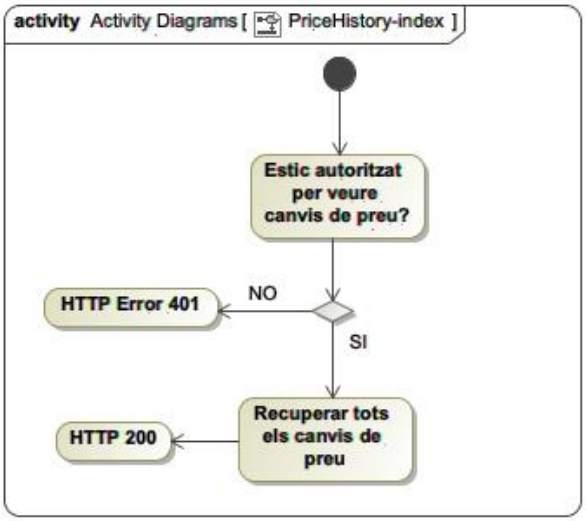

activity Country-create [Country-create]
Estic autoritzat per crear un pais?
HTTP Error 401
NO
SI
Valida que hi hagi valors i que CODE sigui UNIQUE
Son valides les dades POST
HTTP Error 422
NO
SI
Existeix aquest pais amb el mateix codi?
NO
SI
Guarda
S'ha guardat bé?
HTTP Error 500
NO
SI
HTTP 201

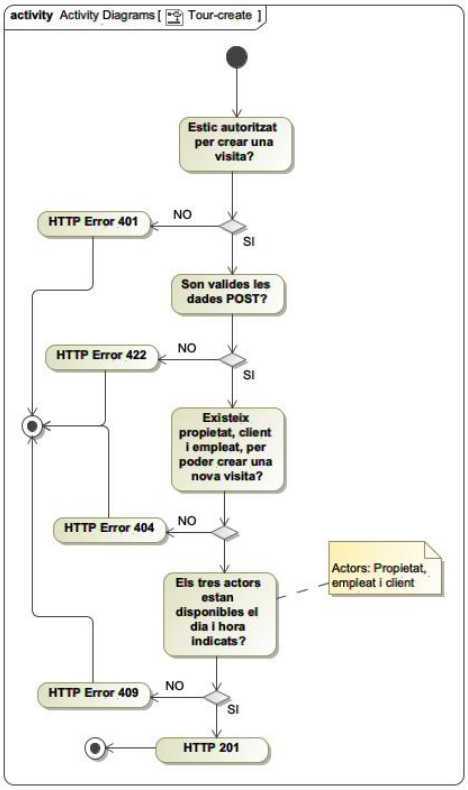

activity Activity Diagrams [City-create]
Estic autoritzat per crear ciutats?
NO
HTTP Error 401
SI
Son valides les dades POST?
NO
HTTP Error 422
SI
Existeix el pais relacionat?
NO
HTTP Error 404
SI
Existeix aquesta ciutat amb el pais relacionat?
NO
HTTP Error 409
SI
Guarda
S'ha guardat bé?
NO
HTTP Error 500
SI
HTTP 201

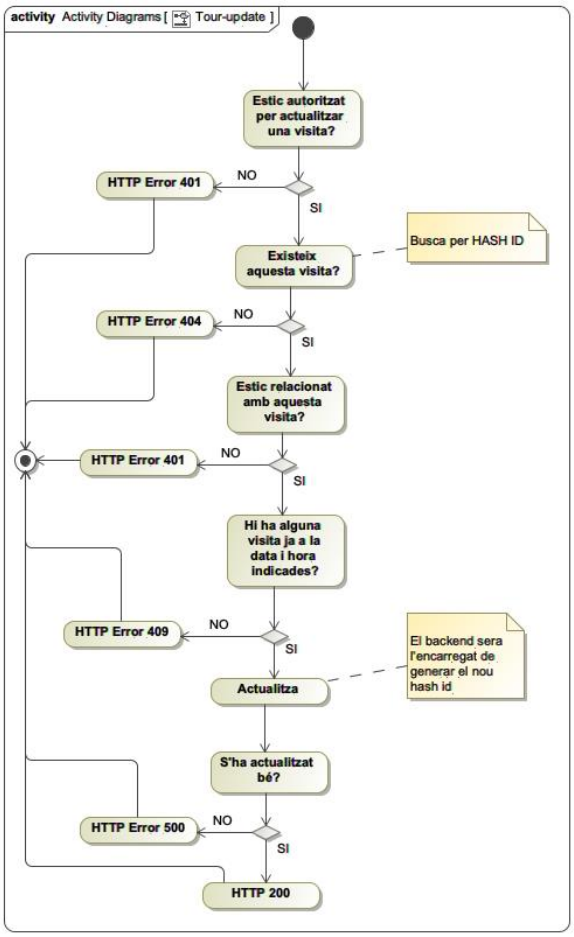

activity Activity Diagrams [City-update]
Estic autoritzat per actualitzar ciutats?
NO
HTTP Error 401
SI
Son valides les dades POST?
NO
HTTP Error 422
SI
Actualitza la ciutat
S'ha actualitzat bé?
NO
HTTP Error 500
SI
HTTP 204

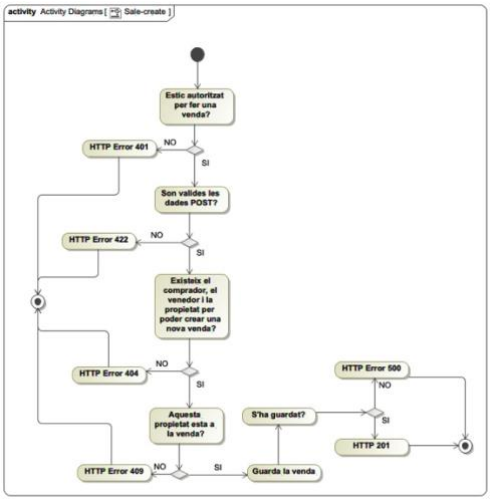

activity Activity Diagrams [City-delete]
Estic autoritzat a eliminar una ciutat?
HTTP Error 401
NO
SI
Existeix aquesta ciutat?
HTTP Error 404
NO
SI
Té cap propietat relacionada aquesta ciutat?
HTTP Error 409
SI
NO
Elimina
S'ha eliminat bé?
HTTP Error 500
NO
SI
HTTP 204

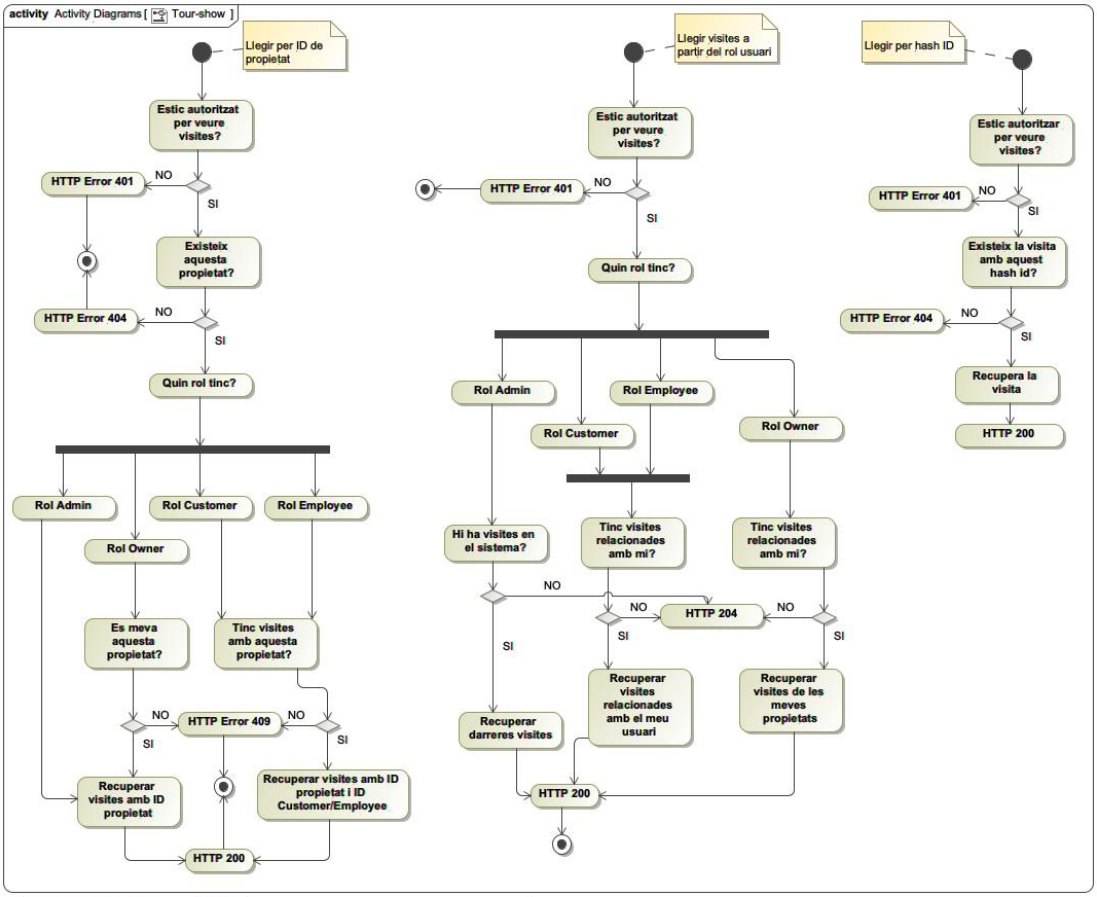

activity Activity Diagrams [PriceHistory-create]
Estic autoritzat per realitzar un canvi de preu
NO
HTTP Error 401
SI
Son valides les dades POST?
HTTP Error 422
NO
SI
Som administrador o empleat?
Tenir en compte que un mateix canvi de preu, hi ha de haver una instància amb el mateix property_id, start i amount
NO
Som propietari?
Existeix un mateix canvi de preu?
SI
SI
La propietat que vull fer un canvi de preu es de la meva propietat?
NO
HTTP Error 409
SI
NO
SI
La data d'inici de l'últim canvi de preu es inferior a la data de inici nova?
NO
SI
HTTP 201

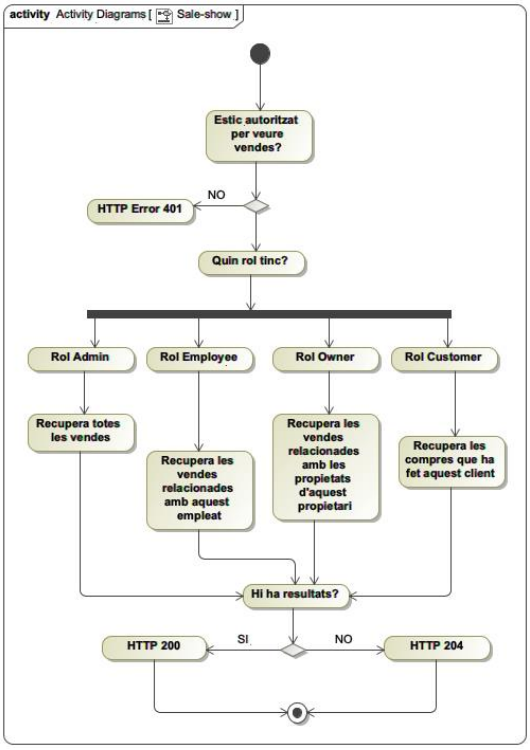

activity Activity Diagrams [PriceHistory-index]
Estic autoritzat per veure canvis de preu?
HTTP Error 401
NO
SI
Recuperar tots els canvis de preu
HTTP 200

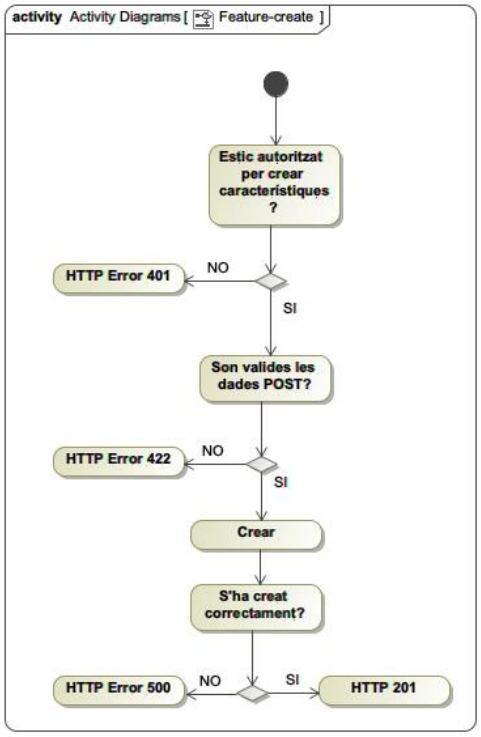

activity Activity Diagrams [Tour-create]
Estic autoritzat per crear una visita?
HTTP Error 401
NO
SI
Son valides les dades POST?
HTTP Error 422
NO
SI
Existeix propietat, client i empleat, per poder crear una nova visita?
HTTP Error 404
NO
Els tres actors estan disponibles el dia i hora indicats?
Actors: Propietat, empleat i client
HTTP Error 409
NO
SI
HTTP 201

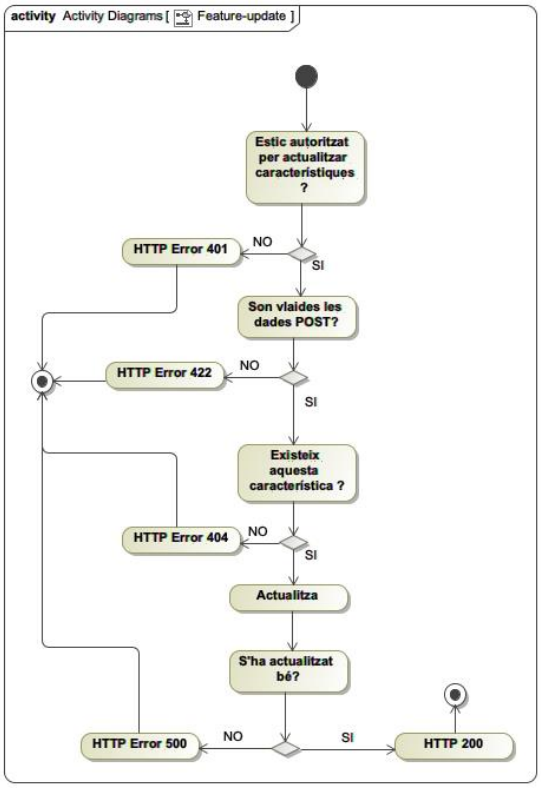

activity Activity Diagrams [Tour-update]
Estic autoritzat per actualitzar una visita?
HTTP Error 401
NO
SI
Existeix aquesta visita?
Busca per HASH ID
HTTP Error 404
NO
SI
Estic relacionat amb aquesta visita?
HTTP Error 401
NO
SI
Hi ha alguna visita ja a la data i hora indicades?
HTTP Error 409
NO
SI
Actualitza
El backend sera l'encarregat de generar el nou hash id
S'ha actualitzat bé?
HTTP Error 500
NO
SI
HTTP 200

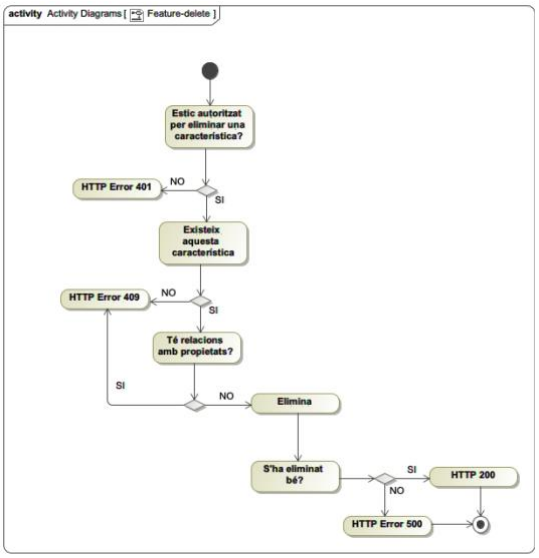

activity Activity Diagrams [Sale-create]
Estic autoritzat per fer una venda?
HTTP Error 401
NO
SI
Son valides les dades POST?
HTTP Error 422
NO
SI
Existeix el comprador, el venedor i la propietat per poder crear una nova venda?
HTTP Error 404
NO
SI
HTTP Error 500
NO
Aquesta propietat esta a la venda?
S'ha guardat?
SI
HTTP 201
HTTP Error 409
NO
SI
Guarda la venda


activity Activity Diagrams [Tour-show]
Llegir per ID de propietat
Estic autoritzat per veure visites?
HTTP Error 401
NO
SI
Existeix aquesta propietat?
HTTP Error 404
NO
SI
Quin rol tinc?
Rol Admin
Rol Customer
Rol Employee
Rol Owner
Es meva aquesta propietat?
Tinc visites amb aquesta propietat?
NO
HTTP Error 409
NO
SI
SI
Recuperar visites amb ID propietat
Recuperar visites amb ID propietat i ID Customer/Employee
HTTP 200
Llegir visites a partir del rol usuari
Estic autoritzat per veure visites?
HTTP Error 401
NO
SI
Quin rol tinc?
Rol Admin
Rol Employee
Rol Customer
Rol Owner
Hi ha visites en el sistema?
Tinc visites relacionades amb mi?
Tinc visites relacionades amb mi?
NO
NO
HTTP 204
NO
SI
SI
SI
Recuperar darreres visites
Recuperar visites relacionades amb el meu usuari
Recuperar visites de les meves propietats
HTTP 200
Llegir per hash ID
Estic autoritzar per veure visites?
HTTP Error 401
NO
SI
Existeix la visita amb aquest hash id?
HTTP Error 404
NO
SI
Recupera la visita
HTTP 200


activity Activity Diagrams [Sale-show]
Estic autoritzat per veure vendes?
HTTP Error 401
NO
Quin rol tinc?
Rol Admin
Rol Employee
Rol Owner
Rol Customer
Recupera totes les vendes
Recupera les vendes relacionades amb aquest empleat
Recupera les vendes relacionades amb les propietats d'aquest propietari
Recupera les compres que ha fet aquest client
Hi ha resultats?
HTTP 200
SI
NO
HTTP 204


activity Activity Diagrams [Feature-create]
Estic autoritzat per crear caracteristiques ?
HTTP Error 401
NO
SI
Son valides les dades POST?
HTTP Error 422
NO
SI
Crear
S'ha creat correctament?
HTTP Error 500
NO
SI
HTTP 201





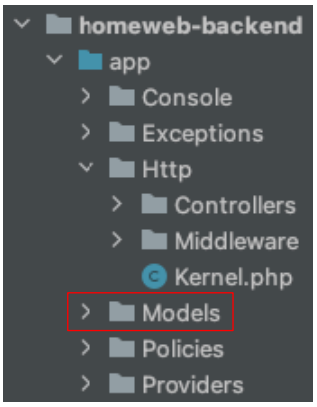
Apèndix 2: MVC amb Laravel

Model en Laravel

Per el model de dades *Laravel* utilitza un ORM (Object-Relational Mapper) molt potent i amb un sintaxis molt senzilla que el fa molt fàcil d'utilitzar, aquest es *l'Eloquent*. Aquest el que fa és que cada taula de la nostra base de dades li fa correspondència amb un model de dades definit dins el nostre Model. Ens ajuda a fer les operacions habituals de persistència i recuperació de dades (inserció, actualització, lectura i eliminació).

Una altre funcionalitat molt interessant i que ens ajuda a construir la nostra base de dades són les *migrations.* Aquestes són classes que defineixen la estructura de cada taula de la nostra base de dades i mitjançant una comanda que porta *Laravel* ens enllaça el model de dades amb la nostra taula, d'aquesta manera no ens tenim que preocupar de dissenyar totes aquestes funcionalitats de persistència i recuperació que abans hem comentat.

També utilitzarem els *seeders* són unes classes que *Laravel* utilitza per poblar la nostra base de dades amb dades de proves, per així poder fer les pertinents proves més endavant, sense haver de introduir-les a mà o mitjançant un script de MySQL.



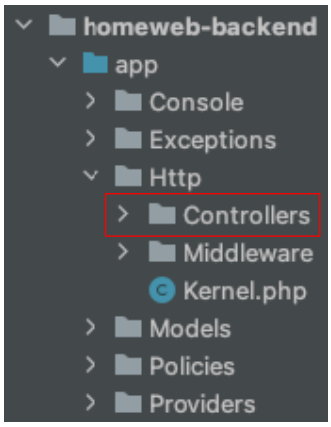
Com podem observar tenim la carpeta *app/Models* on tenim allotjats tots els models definits de la nostra aplicació.

Més endavant farem una petita explicació de les passes a seguir per aconseguir-ho.

Controlador amb Laravel

Laravel utilitza els controladors per agrupar peticions relacionades amb un model de dades. Totes les peticions que arribin a la API relacionades per exemple amb usuaris, serà el controlador *UserController* l'encarregat de gestionar aquestes peticions de clients, i per tant executarà les funcions o mètodes relacionats, normalment crear, recuperar, actualitzar i eliminar dades del nostre model.

En *Laravel* també tenim una comanda que ens ajuda a crear aquests controladors, també més endavant explicaré com funciona i quins són els resultats.



Laravel conté els controladors dins la carpeta *app/Http/Controllers*. La funció principal dels controladors serà de executar la lògica de negoci de la nostra aplicació per poder donar sentit als serveis que oferim.

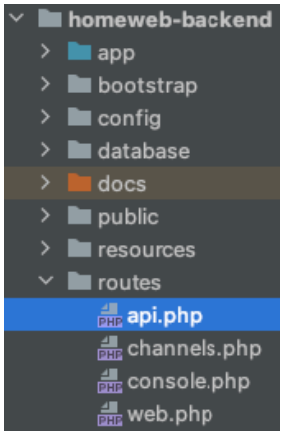
Vista amb Laravel

Laravel proporciona per a la capa de Vista un sistema de plantilles anomenat *Blade*, el qual proporciona una sofisticada estructuració de les mateixes. Però en aquest cas com que per la part client en el pla de treball vaig escollir utilitzar el *framework Angular*, no farem servir aquest motor de plantilles que ens proporciona *Laravel*. Per més endavant explicarem en que consisteix *Angular* i quina arquitectura utilitza.

Apèndix 3: API REST amb Laravel

En aquest apartat explicarem i mostrarem com el *backend Laravel* té muntat una API REST , amb les seves URL, els recursos associats, el mètode HTTP utilitzat per requerir aquest recurs i el mètode o verb que utilitzarà el controlador per aplicar la lògica de negoci, per desprès poder donar resposta HTTP al client mitjançant els codis d'estat que proporciona el protocol HTTP. Per cert *Laravel* una de les seves característiques es que esta muntat i pensat per donar suport a *RESTful*, per tant facilita a l'hora de construir aquesta API REST.

Laravel proporciona un fitxer amb extensió php, on s'han de registrar els *endpoints* de la nostra API. S'ha d'indicar quin controlador farà servir i quin mètode executarà.



Laravel conté aquest fitxer dins la carpeta app/routes, i dins ell és on definirem tots els *endpoints* que la API exposarà.

```php
/*
|--------------------------------------------------------------------------
| API Routes
|--------------------------------------------------------------------------
|
| Here is where you can register API routes for your application. These
| routes are loaded by the RouteServiceProvider within a group which
| is assigned the "api" middleware group. Enjoy building your API!
|
*/
Route::post( uri: 'register', [PassportAuthController::class, 'register']);
Route::post( uri: 'login', [PassportAuthController::class, 'login']);

// Unauthenticated routes
Route::get( uri: '/categories/all', [CategoryController::class, 'all'])->name( name: 'all');
Route::get( uri: '/users/auth', [UserController::class, 'getAuthUser'])->name( name: 'auth');

Route::middleware( middleware: 'auth:api')->group(function () {
    // Properties auth routes
    Route::prefix( prefix: 'properties')->name( value: 'properties.')->group(function () {
        Route::post( uri: '/create', [PropertyController::class, 'create'])->name( name: 'create');
        Route::get( uri: '/{id}/show', [PropertyController::class, 'show'])->where( name: 'id', expression: '[0-9]+')->name( name: 'show');
        Route::put( uri: '/{id}/update', [PropertyController::class, 'update'])->where( name: 'id', expression: '[0-9]+')->name( name: 'update');
        Route::get( uri: '/{id}/setActive/{status}', [PropertyController::class, 'setActive'])->where(['id' => '[0-9]+', 'status' => '[0-1]{1}'])->name( name: 'setActive');
        Route::get( uri: '/{id}/owner', [PropertyController::class, 'owner'])->where( name: 'id', expression: '[0-9]+')->name( name: 'owner');
        Route::get( uri: '/all', [PropertyController::class, 'all'])->name( name: 'all');
    });
    // Roles auth routes
    Route::prefix( prefix: 'roles')->name( value: 'roles.')->group(function () {
        Route::get( uri: '/{id}', [RoleController::class, 'userRole'])->where( name: 'id', expression: '[0-9]+')->name( name: 'user');
        Route::get( uri: '/myRole', [RoleController::class, 'myRole'])->name( name: 'mine');
        Route::get( uri: '/all', [RoleController::class, 'all'])->name( name: 'all');
    });
    // Categories auth routes
    Route::prefix( prefix: 'categories')->name( value: 'categories.')->group(function () {
        Route::post( uri: '/create', [CategoryController::class, 'create'])->name( name: 'create');
    });
});
```

Aquí podem veure el fitxer *api.php* amb alguns *endpoints* ja implementats, si en fixem podem veure que per una banda tenim unes rutes encapsulades dins una ruta *middleware auth:api*, d'aquesta manera aquestes rutes estan protegides per autenticació, per tant si un usuari vol accedir a elles haurà d'enviar juntament amb la petició HTTP el *JWT* (L'arquitectura *JWT* s'explica més endavant) que l'autentiqui al sistema. I per un altre banda tenim les primeres rutes que no estan protegides per aquesta capa de seguretat.

Si revisem una ruta podem veure com trobem totes les característiques que abans hem explicat.

```php
// Categories auth routes
Route::prefix( prefix: 'categories')->name( value: 'categories.')->group(function () {
    Route::post( uri: '/create', [CategoryController::class, 'create'])->name( name: 'create');
});
```

- ⇒ ***prefix***: Indica el tipus de recurs el qual estaran exposades les rutes que tenim en el segon nivell. En aquest cas tenim el recurs categories.
- ⇒ ***name***: Serveix per donar un nom genèric al conjunt de rutes, ja que desprès dins els controladors podem fer redireccions a altres *endpoints* utilitzant els noms definits dins aquest mètode. Si ens fixem a la ruta de segon nivell també tenim aquest mateix mètode que defineix el nom *create.* Per tant ara podrem fer dins un controlador en un moment donat si es requereix una redirecció i anomenar-la com *categories.create*.

⇒ ***group***: Mitjançant el mètode *group* el que estem fent es compartir el prefix que hem definit amb totes les rutes que tenim a dins, per tant en aquest cas la ruta http seria, *http://YOUR_DOMAIN/categories/create*.

JWT i Laravel amb Passport

Per poder implementar aquesta seguretat a la nostra *API* utilitzarem un *package* que ofereix *Laravel*, anomenat *Passport*, aquest implementa un *OAuth 2 Server*, però nosaltres sols utilitzarem la funcionalitat de *JWT*. Per tant la aplicació d'aquesta manera queda preparada per poder utilitzar el protocol *OAuth 2* i poder fer *login* d'usuaris amb tercers.

Apèndix 4: MVVM o MVW amb Angular

Models amb Angular

Angular utilitzar classes i interfícies per a definir models de dades. En ells s'especifiquen l'estructura de dades que té l'objecte en qüestió i si es necessari també crear algun mètode per a executar alguna funcionalitat relacionada amb el mateix. Per norma general les classes s'utilitzen per si has d'interaccionar amb l'objecte, però per altre banda les interfícies serveixen per definir una estructura de dades que poden provenir d'algun *web service* o alguna *API REST* d'aquesta manera si funcionem així no serà necessari utilitzar una classe i haver de instanciar cada cop que es vulgui recuperar la seva informació del servei web. En el nostre cas com que tenim una API REST es convenient utilitzar les interfícies per a proporcionar senzillesa al procés de recuperar les dades del recurs que vulguem requerir.

Serveis HTTP

Per a poder realitzar peticions HTTP a un servei web o API REST és fonamental utilitzar aquests tipus de servies. Per una banda és normal ja que el que fem es aïllar una funcionalitat de la resta de funcions que pugui tenir la nostra aplicació. I per altre per la senzillesa amb que es poden realitzar aquests tipus de peticions HTTP asíncrones. Per tant en la nostra aplicació es crearan serveis expressament per fer les pertinents peticions HTTP als *endpoints* del *backend*.

Components

Els components en Angular es pot dir que es la part més important que té i és amb el que es basa, en el nostre cas representaria la Vista de la nostra arquitectura MVC.
La principal funció es la reutilització de codi, es crea un component i aquest es pot reutilitzar en qualsevol altre part de l'aplicació. En ell es gestiona la vista que l'usuari final visualitzarà. Es componen de tres arxius, un *HTML* (equivaldria al a vista de la nostre arquitectura MVVM), un arxiu *TypeScript* que es l'encarregat de gestionar la vista (equivaldria al Model de la nostre arquitectura MVVM) i una fulla d'estils *CSS*.

Funcionament dins Angular

Per deixar clar el concepte de funcionament que tindrà la nostra aplicació, tindrem diferents paquets on s'ubicaran diferents tipus de components els quals tindran una tasca concreta assignada, d'aquesta manera altres components podran utilitzar les funcionalitats d'aquests altre components per poder arribar a un fi.
En el nostre cas tindrem:

- ⇒ **Models de dades**, definits per interfícies, i que representaran l'estructura d'un recurs.
- ⇒ **Serveis HTTP**, que seran els encarregats de realitzar les peticions HTTP al backend.
- ⇒ **Components**, que s'encarregaran de utilitzar tant els models com els serveis per al final poder mostrar la informació que l'usuari pugui requerir.

Apèndix 5: Generació projectes backend i frontend

Pre-requisits Laravel

Per poder fer funcionar un projecte Laravel, primer de tot hem de tenir un entorn especialment adaptat per a la seva execució.
En aquest cas es necessari tenir instal·lat el següent:

- ⇒ LAMP
    - Servidor web, per exemple Apache.

- Llenguatge de programació web PHP.
- SGBD relacional MySQL.

⇒ Gestor de dependències PHP, Composer.

⇒ Control de versions Git.

En el meu cas he desenvolupat aquesta part en un sistema operatiu Mac OS X, i he utilitzat un entorn instal·lat localment per poder crear el backend. Per poder deixar preparat l'entorn LAMP he seguit aquest tutorial.

Les versions utilitzades:

```
~ (-zsh)
❯ apachectl -v
Server version: Apache/2.4.46 (Unix)
Server built:   Aug  7 2020 10:55:45
❯ php -v
PHP 8.0.1 (cli) (built: Jan  8 2021 12:43:54) ( NTS )
Copyright (c) The PHP Group
Zend Engine v4.0.1, Copyright (c) Zend Technologies
    with Xdebug v3.0.2, Copyright (c) 2002-2021, by Derick Rethans
    with Zend OPcache v8.0.1, Copyright (c), by Zend Technologies
❯ mysql --version
mysql  Ver 14.14 Distrib 5.7.19, for macos10.12 (x86_64) using  EditLine wrapper
❯ composer -v
   ______
  / ____/___  ____ ___  ____  ____  ________  _____
 / /   / __ \/ __ `__ \/ __ \/ __ \/ ___/ _ \/ ___/
/ /___/ /_/ / / / / / / /_/ / /_/ (__  )  __/ /
\____/\____/_/ /_/ /_/ .___/\____/____/\___/_/
                    /_/
Composer version 2.0.8 2020-12-03 17:20:38
```

En contrapartida també es possible evitar tot això i utilitzar Docker, ja que avui en dia es una opció molt vàlida per executar aplicacions sense necessitat de instal·lar res al nostre sistema operatiu. Docker proporciona un contenidor on tenim totes les dependències necessàries per poder executar la nostra aplicació, en el nostre cas Apache, PHP i MySQL, gràcies això podrem compartir aquesta imatge i qualsevol persona que tingui Docker instal·lat podrà executar-lo i poder fer funcionar la nostra aplicació, independentment del sistema operatiu utilitzat.

Instal·lació Laravel

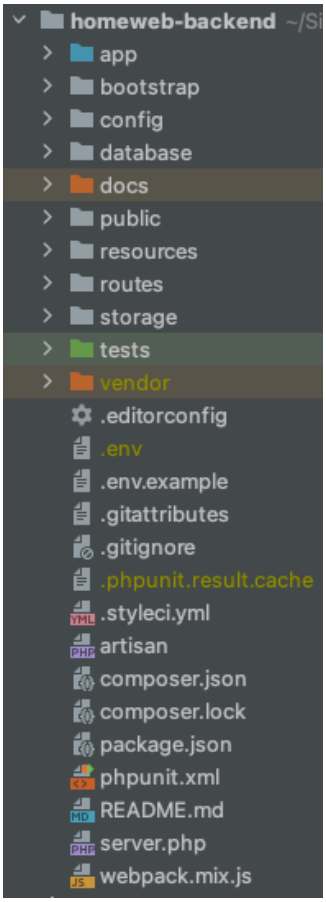
Un cop tenim l'entorn preparat per poder executar el nostre projecte Laravel, en dirigim a la consola de comandes i executem.

```
❯ composer create-project laravel/laravel homeweb-backend
Creating a "laravel/laravel" project at "./homeweb-backend"
Installing laravel/laravel (v8.5.17)
  - Downloading laravel/laravel (v8.5.17)
  - Installing laravel/laravel (v8.5.17): Extracting archive
Created project in /Users/Ivan/homeweb-backend
> @php -r "file_exists('.env') || copy('.env.example', '.env');"
Loading composer repositories with package information
Updating dependencies
Lock file operations: 104 installs, 0 updates, 0 removals
```

Gràcies a composar ens baixarà al directori o estem el framework Laravel, en aquest cas la versió 8.5.17. Tot seguit també es baixarà les dependències necessàries.

Un cop tenim això instal·lat, obrim la carpeta amb un IDE per exemple PHPStorm en el meu cas i podrem veure la estructura de carpetes que té.

A continuació es fa una explicació a molt alt nivell de les carpetes més importants, i les que s'han utilitzat per anar implementant el backend.



- ⇒ **app**: Carpeta on trobem les carpetes que fan referència a MVC.
  - Carpeta Models: On tenim els models de dades definits.
  - Carpeta Http/Controllers: On tenim els controladors.
  - Carpeta Policies: On tenim les classes que defineixen quins rols poden accedir als recursos que tenim definits.
  - Carpeta Services: On tenim les classes que els controladors s'abasteixen per obtenir les dades.
- ⇒ **config**: Carpeta on tenim totes les configuracions de la nostra aplicació.
- ⇒ **database**: Carpeta on tenim els diferents arxius i classes necessaris per poder construir la nostra base de dades
  - migrations: On tenim les migracions, cada migració especifica l'estructura de una taula.
  - seeders: On tenim les classes que ens poblen les taules de la base de dades.

- ⇒ **routes**: Carpeta on tenim els fitxers que defineixen les rutes HTTP de la nostra aplicació, en el nostra cas com que s'ha contruït una API REST, els endpoints s'han definit en l'arxiu api.php.
- ⇒ **storage**: On guardem per exemple imatges que volem pujar al servidor, també podrem torbar la cache, les sessions, els logs.
- ⇒ **tests**: On tenim les classes que executen els tests.
- ⇒ **vendor**: On tenim totes les dependències del framework. Si s'en vol afegir alguna mitjançant el comanda composar podrem instal·lar-la i aquesta s'afegirà al fitxer composar.json.
- ⇒ .**env**: Aquest fitxer es l'encarregat de guardar tota la configuració de la nostra aplicació, per exemple per a la connexió a la base de dades haurem de escriure les credencials dins aquest fitxer.
- ⇒ **artisan**: És l'intèrpret de comandes de Laravel, amb ell podrem realitzar accions molt més ràpidament per exemple damunt la nostra base de dades, entre moltes altres més.

Configuració fitxer .ENV

Base de dades

Per a configurar la nostra base de dades haurem de introduir les credencials en els següents camps.

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=homeweb
DB_USERNAME=
DB_PASSWORD=
```

Per comprovar la connexió a MySQL podem escriure aquesta comanda artisan.

```
Terminal:  Local ×  +
~/Sites/homeweb-backend  on main
php artisan db
```

```
Terminal:  Local ×  +
❯ php artisan db
mysql: [Warning] Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 6
Server version: 5.7.19 MySQL Community Server (GPL)

Copyright (c) 2000, 2017, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

Com podem veure utilitza el client MySQL que tenim instal·lat al SO i ens connecta a un terminal MySQL.

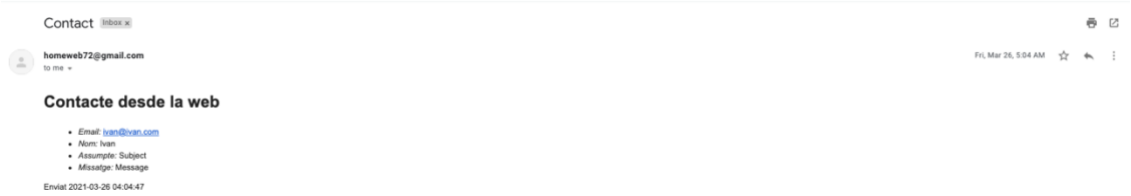
Email de contacte

En la nostra aplicació s'ha configurat un correu Gmail que serà en el que rebrem les peticions de contacte de la nostra aplicació web, per tant per a configurar aquest correu electrònic haurem de introduir les credencials.

```
MAIL_MAILER=smtp
MAIL_HOST=smtp.googlemail.com
MAIL_PORT=465
MAIL_USERNAME=homeweb72@gmail.com
MAIL_PASSWORD=q4m12ZTAIeTf
MAIL_ENCRYPTION=ssl
MAIL_FROM_ADDRESS=homeweb72@gmail.com
MAIL_FROM_NAME="${APP_NAME}"
```

Aquest correu electrònic s'ha creat expressament per aquest projecte.

Correu de prova rebut a la bústia.



Usuaris de prova

Per a poder provar la aplicació durant tota la seva implementació s'han creat diferents usuaris amb diferents rols, per a poder autenticar-se al sistema i poder realitzar accions. Per tant aquests usuaris són:

```
API_ADMIN_EMAIL=admin@homeweb.com
API_ADMIN_PASSWORD=12345678
API_CUSTOMER_EMAIL=customer@homeweb.com
API_CUSTOMER_PASSWORD=12345678
API_EMPLOYEE_EMAIL=employee@homeweb.com
API_EMPLOYEE_PASSWORD=12345678
API_OWNER_EMAIL=owner@homeweb.com
API_OWNER_PASSWORD=12345678
API_OWNER1_EMAIL=owner1@homeweb.com
API_OWNER1_PASSWORD=12345678
API_CUSTOMER1_EMAIL=customer1@homeweb.com
API_CUSTOMER1_PASSWORD=12345678
API_EMPLOYEE1_EMAIL=employee1@homeweb.com
API_EMPLOYEE1_PASSWORD=12345678
```

Perquè aquests usuaris puguin ser utilitzats, s'han de crear primerament per tant si ens dirigim a database/seeders/UserSeeder.php i allà trobarem com es creen tots aquets usuaris.

```
DB::table( table: 'users')->insert([
    'name' => 'Admin',
    'email' => 'admin@homeweb.com',
    'password' => bcrypt( value: '12345678'),
    'phone' => '890890890',
    'address' => 'Street ZYX',
    'fiscal_id' => '87654321A',
    'role_id' => 1,
    'created_at' => Carbon::now(),
    'updated_at' => Carbon::now(),
]);
```

També es necessari introduir dins el fitxer config/api.php.

```
return [
    'apiAdminEmail' => env( key: 'API_ADMIN_EMAIL'),
    'apiAdminPassword' => env( key: 'API_ADMIN_PASSWORD'),
    'apiCustomerEmail' => env( key: 'API_CUSTOMER_EMAIL'),
    'apiCustomerPassword' => env( key: 'API_CUSTOMER_PASSWORD'),
    'apiEmployeeEmail' => env( key: 'API_EMPLOYEE_EMAIL'),
    'apiEmployeePassword' => env( key: 'API_EMPLOYEE_PASSWORD'),
    'apiOwnerEmail' => env( key: 'API_OWNER_EMAIL'),
    'apiOwnerPassword' => env( key: 'API_OWNER_PASSWORD'),
    'apiOwner1Email' => env( key: 'API_OWNER1_EMAIL'),
    'apiOwner1Password' => env( key: 'API_OWNER1_PASSWORD'),
    'apiCustomer1Email' => env( key: 'API_CUSTOMER1_EMAIL'),
    'apiCustomer1Password' => env( key: 'API_CUSTOMER1_PASSWORD'),
    'apiEmployee1Email' => env( key: 'API_EMPLOYEE1_EMAIL'),
    'apiEmployee1Password' => env( key: 'API_EMPLOYEE1_PASSWORD'),
];
```

Ara ja podem utilitzar aquests usuaris utilitzant la façana Config que porta Laravel. En els tests s'ha utilitzat.

```
$baseUrl = Config::get( key: 'app.url') . '/api/auth/login';
$email = Config::get( key: 'api.apiAdminEmail');
$password = Config::get( key: 'api.apiAdminPassword');
```

Per tant la sentència Config::get('api.apiAdminEmail'), ens retorna admin@homeweb.com que tenim configurat dins el fitxer .env.

Pre-requisits Angular

Per a poder fer una instal·lació de un projecte Angular, prèviament s'han de instal·lar unes certes dependències ja que sinó no serà possible.
Necessitem tenir instal·lat prèviament al nostre sistema operatiu:

- ⇒ Node JS: Es un entorn en temps d'execució multiplataforma, per la capa del servidor i esta basat en JavaScript.

⇒ NPM (Node Package Manager): Es un gestor de paquets/dependències de Node.

⇒ CLI de Angular: Es l'eina amb la que podrem generar aplicacions. Amb ella podrem crear, depurar i publicar la nostra aplicació web.

⇒ Control de versions Git: Es un control de versions de programari, ens permet dur un control sobre el nostre desenvolupament.

Per a la instal·lació de l'entorn he seguit aquest tutorial [10].

Podem comprovar mitjançant la consola que tenim el necessari per poder instal·lar un projecte d'Angular.

```
Ivan@iMac-de-IMAC-IVAN:~
❯ node -v
v14.16.0
❯ npm -v
6.14.11
❯ git --version
git version 2.23.0
❯ ng version

Angular CLI: 11.0.7
Node: 14.16.0
OS: darwin x64

Angular: undefined
...
Ivy Workspace: <error>

Package                      Version
-----------------------------------------------------
@angular-devkit/architect    0.1100.7 (cli-only)
@angular-devkit/core         11.0.7 (cli-only)
@angular-devkit/schematics   11.0.7 (cli-only)
@schematics/angular          11.0.7 (cli-only)
@schematics/update           0.1100.7 (cli-only)

at 07:47:22 AM
```

Com hem comentat abans també es possible evitar aquest treball de preparació de l'entorn i executar la nostra aplicació web dins un contenidor Docker amb les dependències necessàries per executar Angular.

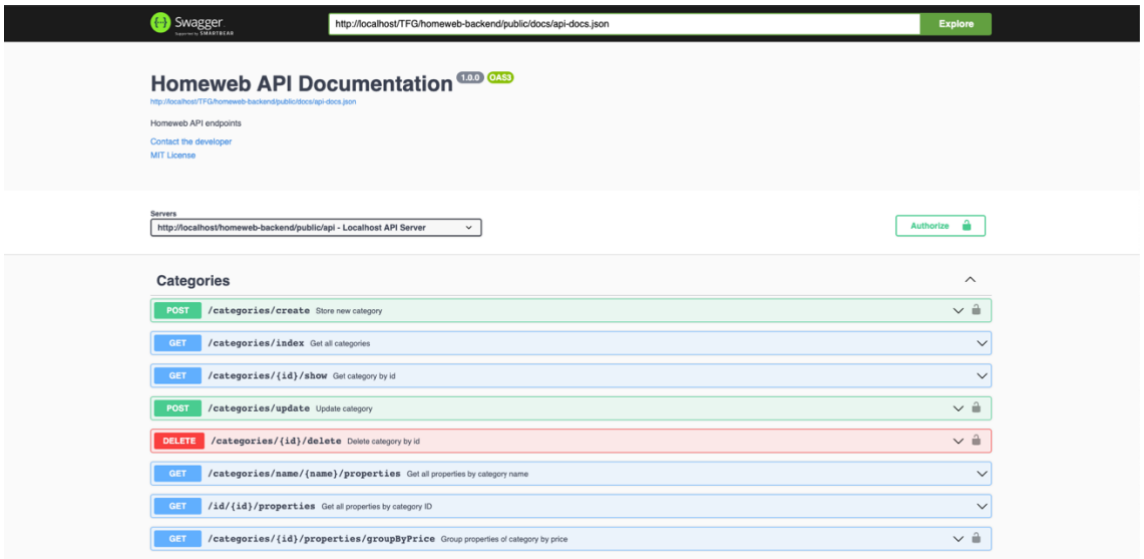
Un cop instal·lat ens fiquem dins la carpeta del projecte Angular i executem ng serve, perquè Angular ens compili el codi i ens serveixi el projecte a http://localhost:4200.

```
ng serve
Last login: Sun May 30 10:14:29 on console
❯ cd Sites/homeweb-frontend
❯ ng serve
✔ Browser application bundle generation complete.

Initial Chunk Files          | Names                      |      Size
vendor.js                    | vendor                     |   6.25 MB
main.js                      | main                       | 312.62 kB
polyfills.js                 | polyfills                  | 149.73 kB
styles.css                   | styles                     | 143.21 kB
runtime.js                   | runtime                    |   9.02 kB

                             | Initial Total              |   6.85 MB

Lazy Chunk Files             | Names                      |      Size
dashboard-dashboard-module.js | dashboard-dashboard-module |   2.50 MB

Build at: 2021-05-30T09:12:00.044Z - Hash: c6786b3492f0893ff95e - Time: 32882ms

** Angular Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/ **


✔ Compiled successfully.
```

Apèndix 6: Documentació API amb OpenAPI 3.0 i Swagger.

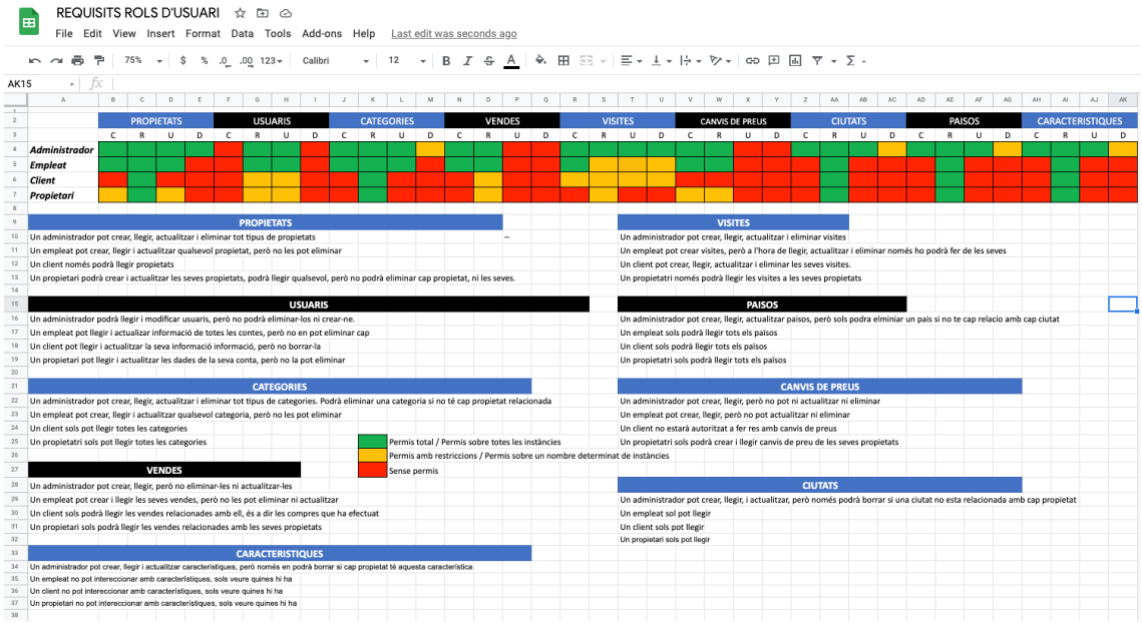
En el backend s'ha generat una documentació per a descriure totes les rutes o endpoints de la API, això gràcies a OpenAPI 3.0 i Swagger que són unes eines dissenyades expressament per a documentar APIs. En total s'han creat 75 endpoints.

Per a poder accedir a aquesta documentació un cop instal·lades les dependències amb composar ens tenim que dirigir a http://localhost/homeweb-backend/public/api/documentation o http://localhost:8080/TFG/homeweb-backend/public/api/documentation si estem utilitzant Docker.

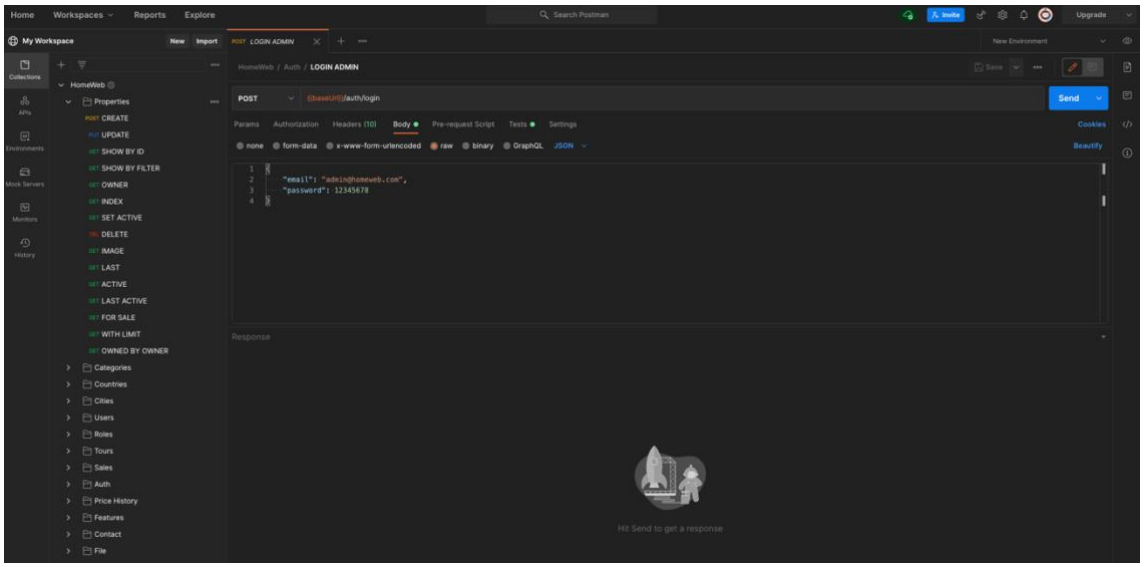


Apèndix 7: Taula especificació requisits.

Enllaç al Google Sheet - [Taula requisits]



Apèndix 8: Projecte amb client HTTP Postman.

A més a més hi ha un projecte creat amb el client HTTP Postman per poder testejar la API REST, si es vol utilitzar s'ha de baixar el document JSON que tenim a continuació i obrir-lo amb Postman.

[Projecte Postman]