Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Interfície de programació d'aplicacions per a la gestió de dades de la plataforma Viviendea

Treball Fi de Grau

Grau en Enginyeria Informàtica

Autora: Roser Fuster Bertomeu

Tutora: Manuela Albert Albiol

2019-2020

Agraïments

En primer lloc, vull agrair a la tutora del treball, Manoli Albert Albiol, per tota l'ajuda que m'ha brindat amb la redacció aquest projecte.

En segon lloc, a l'equip de Viviendea, que m'han acollit com a part de l'equip i han fet possible la realització d'aquest projecte.

En tercer lloc, els meus pares i el meu germà Llorenç, per la paciència infinita que han tingut amb mi i tot el suport que m'han donat al llarg d'aquests anys i en especial en aquesta última etapa.

Per últim, al meu company, Àlex, aquell que està i ha estat amb mi sempre que ho he necessitat, aquell m'ha pujat l'ànim sempre, sobretot en els moments més difícils.

Gràcies a tots.

Resum

En aquest projecte es documenta el desenvolupament d'una API REST dissenyada per a la plataforma Viviendea, una plataforma que reuneix persones interessades en habitatges d'obra nova. L'API REST facilitarà a la plataforma Viviendea l'obtenció i la modificació de les dades des d'altres aplicacions, com ara visors de dades externs, a través de serveis HTTP.

Aquesta API REST tindrà funcionalitats tant de client com de servidor, obtenint les dades proporcionades per l'usuari i emmagatzemant-les en la base de dades de Viviendea i viceversa. Està desenvolupada en Node JS, utilitza un servidor Express JS i un motor de dades MySQL. A més, es proporciona un contracte d'integració per a facilitar a l'usuari la utilització de les seues funcionalitats.

Paraules clau: API, REST, Viviendea, Node JS, Express JS, MySQL.

Resumen

En este proyecto se documenta el desarrollo de una API REST diseñada para la plataforma Viviendea. Una plataforma que reúne personas interesadas en viviendas de nueva construcción. La API REST facilitará a la plataforma Viviendea la obtención y la modificación de los datos desde otras aplicaciones, como visores de datos externos, a través de servicios HTTP.

Esta API REST tendrá funcionalidades tanto de cliente como de servidor, obteniendo los datos proporcionados por el usuario y almacenándolos en la base de datos de Viviendea, y viceversa. Está desarrollada en Node JS, utiliza un servidor Express JS y un motor de datos MySQL. Además se proporciona un contrato de integración para facilitar al usuario la utilización de sus funcionalidades.

Paraules clau: API, REST, Viviendea, Node JS, Express JS, MySQL.

Abstract

This project documents the development of a REST API designed for Viviendea platform, a platform that brings together people interested in new construction housing. The API REST will make it easier for the Viviendea platform to obtain and modify data from other applications, such as external data viewers, through HTTP services.

This REST API will have both client and server functionalities, obtaining the data provided by the user and storing it in the Viviendea database, and vice versa. It is developed in Node JS; it uses an Express JS server and a MySQL data engine. In addition, an integration contract is provided to facilitate the use of its functionalities by the user.

Keywords: API, REST, Viviendea, Node JS, Express JS, MySQL.

Índex general

Índex de imatges

Índex de taules

Acrònims

API	Application Program Interface
CSV	Comma-Separated Values
HTML	Hypertext Mark Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
JSON	JavaScript Object Notation
REST	Representational State Transfer
RFC	Request For Comments
TLS	Transport Layer Security
UML	Unified Modeling Language
URI	Uniform Resource Identifiers
URL	Uniform Resource Locator
XML	eXtensible Markup Language

1. Introducció

1.1 Motivació

Viviendea [17] és una plataforma que reuneix persones interessades en habitatges d'obra nova. La plataforma permet als usuaris introduir les seues preferències respecte a un habitatge, i la plataforma crea a grups de gent amb els mateixos interessos. Una vegada aquests grups tenen suficient gent, la plataforma els posa en contacte amb els arquitectes, promotors i immobiliàries.

Les empreses promotores poden donar d'alta projectes immobiliaris en la plataforma i crear promocions amb un alt nombre d'habitatges ja reservats pels clients. Les dades d'aquestes promocions queden emmagatzemades en la base de dades del servidor de Viviendea. En el cas que aquestes empreses promotores vulguen un visor de dades propi, és interessant que la plataforma oferisca unes funcions predefinides perquè les aplicacions puguen interactuar entre elles.

Aquest treball ve motivat per la necessitat de l'empresa que gestiona Viviendea de crear una API REST de la plataforma per tal de facilitar, en gran manera, l'obtenció o la modificació de les dades des d'altres aplicacions a través de serveis HTTP.

1.2 Objectius

L'objectiu principal d'aquest projecte és la implementació d'una API REST per a que empreses promotores associades a la plataforma puguen crear, accedir, modificar i eliminar les seues dades emmagatzemades en la base de dades de Viviendea, és a dir, les dades de promocions i habitatges que pertanyen a la promotora. Aquest objectiu es pot desglossar en tres subobjectius:

1. Definir i implementar un sistema d'autentificació de l'usuari. Per tal que sols els propietaris de les dades puguen accedir a elles, evitant així que altres empreses puguen obtenir-les.

2. Crear un sistema basat en rols. Una vegada autenticat l'usuari, en funció del seu rol podrà fer unes accions o altres. Sols els usuaris amb rol d'administradors tindran permís per a realitzar totes les accions, en canvi un usuari bàsic sols podrà obtenir les dades emmagatzemades, sense possibilitat de modificar-les, eliminar-les o crear-ne de noves.

3. Dissenyar una aplicació en el que s'han de realitzar les consultes pertinents a la base de dades per tal de proporcionar i rebre la informació sol·licitada. El format en el que es tractaran les dades també s'ha de definir permetent el maneig de les dades. A més, l'aplicació definirà les adreces URI junt amb al mètode utilitzat per a cada una d'elles.

1.3 Col·laboradors

Aquest projecte es desenvoluparà com a complement de la plataforma Viviendea, on l'alumna està realitzant les pràctiques extra-curriculars. Els companys del departament d'informàtica han desenvolupat una interfície frontal de la part del client i la base de dades per a questa plataforma. Aquestes parts ja estan desenvolupades abans de començar aquest projecte.

Aquest projecte es centra exclusivament a desenvolupar el codi d'una interfície de programació d'aplicacions per a millorar la plataforma Viviendea, on aquesta API implementarà les seqüències d'instruccions que s'han de seguir per a obtenir les dades sol·licitades i el codi necessari per a garantir la seguretat a l'hora d'accedir als recursos. Per un altre costat, tots les anàlisis realitzats en aquest Treball de Final de Grau són propis i no s'ha requerit l'ajuda de cap company.

1.4 Organització de la memòria

Aquest projecte està organitzat en nou capítols de la següent manera.

- El capítol 2 explicarà el context tecnològic en el que es troba el projecte. En este capítol s'introdueixen algunes solucions que ja existeixen i són semblants a la solució desenvolupada. A mes, és farà una breu introducció a les ferramentes i les tecnologies empleades en el projecte.
- El capítol 3 planteja les problemàtiques que existeixen a l'hora de realitzar el projecte. Es definiran els casos d'ús, els requisits funcionals i no funcionals i el pla de treball de la solució proposada.
- El capítol 4 presenta l'arquitectura que s'utilitzarà per a desenvolupar posteriorment el projecte. En aquest capítol, a més, es definirà el disseny detallat de cada peça desenvolupada en el projecte.
- El capítol 5 desenvolupa la solució proposada en el capítol 4.
- El capítol 6 detalla les funcionalitats i els paràmetres d'entrada i eixida de cada un dels serveis web que presta l'API.
- El capítol 7 detalla les proves que s'han realitzat per a comprovar el correcte funcionament de l'API.
- El capítol 8 presenta la conclusió del projecte on s'analitza si s'han assolit els objectius d'aquest projecte. En aquest capítol també es planteja la continuïtat del projecte en un futur.
- El capítol 9 recull les referències utilitzades per a desenvolupar aquest projecte.

2. Context tecnològic

Amb l'evolució de les noves tecnologies, han aparegut noves formes de gestionar dades. Ja no és habitual veure aplicacions o pàgines web que sols gestionen les seues pròpies dades. Ara també utilitzen dades d'aplicacions o pàgines web externes per a mostrar el seu contingut. Una pàgina web, per exemple, podria mostrar esdeveniments registrats en una aplicació calendari/agenda o incorporar la relació de piulades d'un compte de Twitter. Aquests exemples permetrien actualitzar els continguts segons es creen en les aplicacions i visualitzar-los sense haver d'entrar en l'aplicació que els té de manera tradicional.

És així com apareix la necessitat d'utilitzar una interfície especifica amb la que interactuar entre aplicacions de forma senzilla, aquesta interfície és la interfície de programació d'aplicacions o API. Un dels estàndards més utilitzat per a crear aquestes API és REST [15].

Definiríem REST com un estàndard lògic i eficient per a la creació de serveis web que està basat en uns principis de desenvolupament arquitectònic. El sistema ha d'estar estructurat per capes per tal que cada component no puga accedir més enllà del component en el qual es relacionen. El client es separa del servidor evitant així qualsevol dependència entre ells. Les sessions iniciades pel client no mantenen l'estat i per tant sempre ha de contenir tota la informació necessària per a sol·licitar un servei, incloent l'autenticació i l'autorització, ja que tampoc té encriptació. Les sol·licituds s'han d'etiquetar implícitament o explícitament per tal de ser emmagatzemades en memòria cau i les dades es poden emmagatzemar en la memòria cau tant del servidor com del client. El client accedeix als recursos mitjançant identificadors uniformes de recursos (URI) relatius i utilitzant el protocol HTTP. Per a manipular els recursos s'utilitzen els mètodes HTTP [18], és a dir, GET, POST, PUT i DELETE. Les representacions dels recursos segueixen un format de dades específic, normalment s'utilitzen els formats JSON, XML o CSV. Una API REST és una interfície web que utilitza XML,JSON,HTML i HTTP sense cap conjunt de capçaleres.

L'objectiu de les APIs és interactuar mitjançant mètodes , subrutines o funcions prèviament definides que indiquen com ha de comportar-se el sistema quan interactua amb ell algun element extern. Tot això les converteix, avui en dia, en una eina fonamental per a la integració de dades.

Una vegada introduït el concepte d'API, fonamental per al desenvolupament del present projecte, les següents seccions fan una revisió d'aproximacions que estan relacionades d'alguna manera amb el que es fa en aquest projecte.

2.1 Solucions semblants

Pràcticament totes les aplicacions actuals compten amb un servei API, siga gratuït o de pagament, depén de les dades que gasten. En el cas del sector immobiliari les API solen ser de pagament, de manera que sols tenen accés les seues empreses

associades. Açò es fa per una part per a evitar l'espionatge d'informació que pot provocar competència deslleial i per l'altre costat perquè ha sigut concebut per a traure un rendiment del seu ús, utilitzant-ho com a producte [1].

Tant Idealista [8], Fotocasa [6], Yaencontre [9], Habitaclia [7], entre altres utilitzen una API REST on s'obtenen dades semblants a la desenvolupada en aquest projecte.

Per exemple, l'API d'Idealista [19] mitjançant un url definida per ells i protegida per una clau els clients poden obtenir informació en un format semblant a CSV. Utilitzant diferents paràmetres en la invocació de l'URL es poden filtrar les dades sol·licitades per diferents tipus, per exemple si és lloguer o venda, la ubicació, un rang de preu, etc.

Les dades obtingudes estan en format JSON, que faciliten el tractament posterior per l'aplicació que invoca l'API. A més les dades proporcionen un llistat de les característiques de l'immoble que compleixen els requisits dels paràmetres.

2.2 Viviendea

Figura 1. Logotip de Viviendea

Viviendea és la plataforma on s'està desenvolupant aquest projecte. L'empresa que va desenvolupar la plataforma, amb el mateix nom, va nàixer a mans de Sergio López Alcover, professional del sector immobiliari, l'any 2015 i va ser recolzada en 2017 per Lanzadera, acceleradora pertanyent a Marina de Empresas impulsada per l'empresari valencià Juan Roig.

El periòdic Las provincias va escriure el 13 de desembre de 2019 sobre Viviendea el següent: "Es tracta d'una plataforma pionera en connectar a l'usuari amb el promotor, que ha desenvolupat un nou algorisme que creua els desitjos del comprador amb l'oferta del mercat."

La plataforma recull les dades d'usuaris interessats en habitatges d'obra nova i genera grups de demanda amb necessitats coincidents. A més, permet als professionals de la construcció veure quins són els tipus de projecte més demandats per adaptar l'oferta i publicar les seues propostes. Tanmateix, amb el temps s'han anat ampliant les exigències de la plataforma.

Les empreses promotores estan interessades en la sincronització de dades de la plataforma amb el seu visor de dades propi per evitar la introducció de les mateixes dades per duplicat, estalviant recursos i temps, a més de visualitzar els projectes que han promocionat sense haver de passar per la plataforma.

Davant d'aquesta demanda, Viviendea ha vist la necessitat de donar resposta i desenvolupar una API REST, què és la finalitat d'aquest projecte.

2.3 Tecnologies Utilitzades

En aquest apartat descriurem a fons les tecnologies utilitzades per a desenvolupar aquest projecte.

REST

A principi del capítol dos s'ha definit breument en què consisteix l'arquitectura REST. En aquest subapartat definirem més detalladament en què consisteix.

Com ja hem esmentat, definim REST com un estàndard lògic i eficient per a la creació de serveis web que està basat en uns principis de desenvolupament arquitectònic. Aquest estàndard ha de satisfer sis restriccions bàsiques que es mostren a continuació.

1. Client-Servidor

   El client és aquell que sol·licita els recursos i el servidor qui emmagatzema les dades i conté els recursos. S'han de desenvolupar de manera independent, amb açò millorem l'escalabilitat i facilita la portabilitat de la interfície d'usuari a diferents plataformes.

2. Sense estat

   Cada sol·licitud que realitza el client ha de contenir tota la informació perquè el servidor la gestione. El servidor no emmagatzema cap estat relacionat amb la sessió iniciada pel client, aquest ha d'introduir els paràmetres de consulta, les capçaleres, en el nostre cas l'autenticació i l'autorització com a part de la capçalera, la consulta i l'URI per tal que la sol·licitud es realitze correctament.

3. Memòria cau

   Com que s'ha d'especificar tota la informació requerida per a fer una sol·licitud, per a emmagatzemar les dades de la resposta a la memòria cau aquesta ha de ser etiquetada implícitament o explícitament en la sol·licitud cada vegada. El client pot emmagatzemar les dades obtenides per tal de millorar el rendiment i la disponibilitat.

4. Interfície uniforme

   Mantenir una interfície uniforme millora la visibilitat de les interaccions, simplifica l'arquitectura del sistema. Aquesta és una restricció important a l'hora de desenvolupar un sistema REST. Per a complir aquest requisit s'han de complir quatre restriccions:

   - Basat en recursos. Es defineixen recursos individuals per a cada sol·licitud.

   - Manipulació de recursos mitjançant representacions. Sempre que el client tinga permís ha de tindre representació del recurs i ha de tenir la informació per a modificar o eliminar el recurs del servidor.

- Missatges autodescriptius. Tots els missatges han de tenir informació suficient per a descriure com serà processat el missatge pel servidor.

- Hipermèdia com a motor d'estat de l'aplicació. Han d'existir enllaços per a cada resposta perquè trobar altres recursos siga fàcil per a l'usuari.

5. Sistema en capes

   L'arquitectura del sistema ha d'estar organitzada per capes jeràrquiques. L'objectiu és evitar que cap capa puga accedir a altres capes que no siguen les seues immediates.

6. Codi a demanda

   Aquest principi és opcional. El servidor pot proporcionar codi executable al client. Açò permet simplificar la feina del client a l'hora d'implementar funcions.

Per a accedir als recursos han de definir-se les URIs, una correcta utilització és important a l'hora d'identificar un recurs. Per a fer un bon ús de les URIs s'han d'utilitzar substantius per a definir-les evitant utilitzar verbs o accions. Una vegada definits els recursos els verbs HTTP defineixen l'acció que realitzarà. Els verbs HTTP més utilitzats són GET, PUT, POST, DELETE i UPDATE.

OAuth 2.0

OAuth 2.0 [11] és un protocol estàndard d'autorització que permet controlar a les aplicacions quan accedeixen a les dades dels usuaris sense haver de proporcionar credencials.

Per a oferir aquesta funcionalitat no s'utilitzen les credencials del propietari de recursos per a accedir als recursos protegits, s'utilitzen codis d'accés temporals que els clients de tercers obtenen a partir d'un servidor d'autorització. Aquests codis d'accés temporals estan formats per una cadena que conté una vida útil, un àmbit específic, i altres atributs d'accés.

Els àmbits són els permisos representats pel *token* d'accés. Per a accedir als recursos protegits, l'aplicació a d'especificar quins àmbits vol i si són autoritzats pel propietari, el *token* d'accés contindrà aquests àmbits autoritzats.

Els rols que es poden identificar en qualsevol flux OAuth 2.0 són els següents:

- Propietari de recursos. L'entitat propietària que pot concedir l'accés als recursos protegits.

- Servidor de recursos. El servidor que emmagatzema els recursos. Normalment és l'API per la qual s'ha d'accedir per a obtenir els recursos protegits.

- Client. L'aplicació que vol accedir a un recurs protegit i el sol·licita en nom del propietari.

- Servidor d'autorització. OAuth, en aquest cas, és el servidor que autentica el propietari dels recursos i proporciona el *token* una vegada ha obtingut l'autorització.

Una vegada esmentats els rols d'OAuth 2.0 procedim a veure el funcionament del protocol, aquest generalment és el següent:

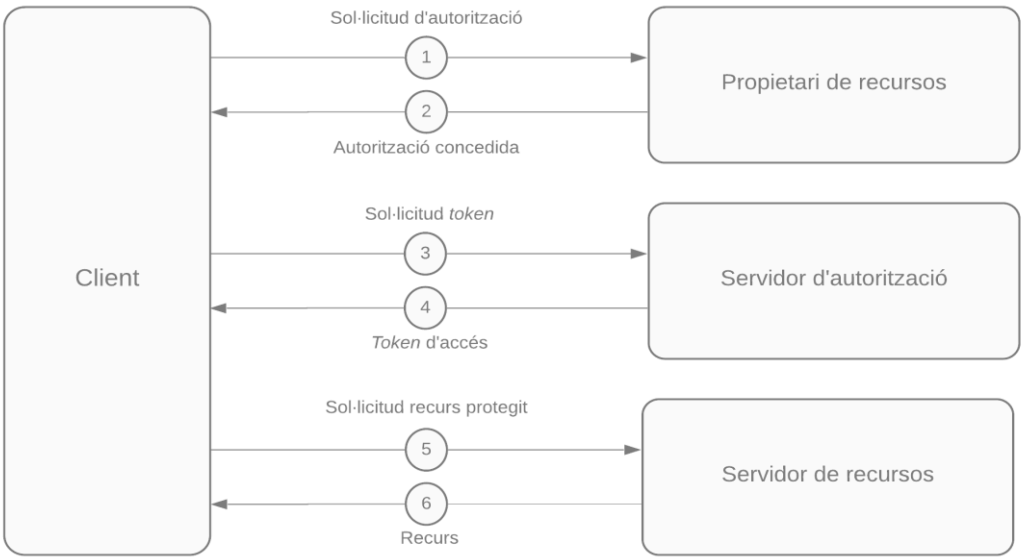

Figura 2. Flux de funcionament de OAuth 2.0

1. El client sol·licita l'autorització del propietari per a accedir als recursos.

2. En cas que el propietari autoritze l'accés al recurs, l'aplicació rep una credencial que representa l'autorització del propietari.

3. L'aplicació sol·licita al servidor d'autorització un *token* d'accés a l'hora que presenta la credencial d'autorització i s'autentica.

4. Si el client s'autentica i proporciona l'autorització concedida vàlida, el servidor d'autorització envia un *token* d'accés a l'aplicació.

5. L'aplicació sol·licita accedir al recurs protegit emmagatzemat en el servidor de recursos a l'hora que presenta el *token* d'accés.

6. Sempre que el *token* siga vàlid, el servidor de recursos proporcionarà els recursos sol·licitats al client.

És un protocol molt utilitzat per a securitzar APIs per la seua flexibilitat, ja que es basa en SSL i xifra les connexions permetent enviar el *token* d'accés de forma segura. A més, el fet que els *tokens* tinguen una vida útil permet a les aplicacions tindre un accés limitat fins que aquesta venç. En utilitzar aquest protocol es garanteix la transferència de dades sense revelar informació personal. Tot açò fa que moltes ferramentes, i cada dia més, permeten el seu ús en diferents entorns i tecnologies.

2.4 Ferramentes utilitzades

En aquesta secció introduïm les eines que han sigut utilitzades per al desenvolupament de l'API per a la gestió de dades de Viviendea.

JavaScript

Figura 3. Logotip de JavaScript

JavaScript [4] és un llenguatge de programació interpretat, orientat a objectes i principalment enfocat al desenvolupament web. Brendan Eich el va crear en 1995 mentre treballava en Netscape Communications com a un complement del llenguatge Java. L'objectiu era fer les pàgines web més dinàmiques i atractives per a l'usuari sense la necessitat d'utilitzar una programació en la part del servidor.

Actualment tots els navegadors interpreten el llenguatge i no necessita cap editor ni compilador en particular, es pot introduir directament el script en el codi HTML. No obstant JavaScript no és un llenguatge exclusivament de la xarxa, s'utilitza en molts programes per a automatitzar tasques.

Node JS

Figura 4. Logotip de Node JS

Node JS [20][21] és un entorn d'execució de codi obert per a JavaScript que naix de la necessitat de treballar amb una arquitectura basada esdeveniments asincrònics d'E/S. Va ser creat per Ryan Dahl en 2009 per a optimitzar l'escalabilitat i el rendiment d'una aplicació web en temps real. Les aplicacions de Node JS s'escriuen en JavaScript, s'executen en la part del servidor i estan formades per mòduls. Utilitzant aquest entorn d'execució es permeten establir connexions bidireccionals on, ja siga el client com el servidor, poden iniciar la comunicació. Açò permet generar pàgines web, recopilar dades de formularis, accedir a bases de dades remotes, gestionar fitxers del servidor, etc.

Express JS

Figura 5. Logotip de Express JS

Express JS [5] és una infraestructura minimalista que es troba dins de la funcionalitat de Nodejs. Express JS va ser dissenyat en 2010 per a millorar l'entorn d'execució Node JS. Es van afegir mètodes de programa d'utilitats amb diferents verbs HTTP de Node JS, facilitar l'organització de l'aplicació amb programari intermediari, per tal de gestionar les tasques, i encaminament de les sol·licituds, és a dir, depenent del mètode URL i HTTP permet modificar el comportament de l'aplicació.

JSDoc

Figura 6. Logotip de JSDoc

JSDoc [22] és un generador de documentació API de JavaScript molt semblant a Javadoc. La primera versió va sorgir en 1999 en mans de Michael Mathews i va anar evolucionant fins que l'any 2012 van publicar la versió JSDoc 3 que és la qual s'utilitza hui en dia.

Mitjançant comentaris dins del codi font els programadors poden documentar l'aplicació i quan s'executa JSDoc analitza els comentaris i genera una pàgina HTML, basant-se en una plantilla predefinida, amb la documentació de tot el programa. Hi ha un ample ventall de característiques que es poden documentar com per exemple l'autor, les classes, els constructors, les funcions, etc.

JSON Web Tokens

Figura 7. Logotip de JSON Web Tokens

JSON Web Tokens (JWT) [10] és l'estàndard obert RFC 7519 que xifra de forma segura la informació a transmetre entre client i servidor com a objecte JSON. JWS xifra mitjançant un parell de claus públiques i privades l'objecte de forma autònoma i compacta, permetent comprovar la integritat del missatge i l'autorització de l'usuari una vegada iniciada la sessió autoritzant l'accés a rutes, serveis i recursos.

El testimoni que proporciona JWT està compost de tres parts, la capçalera, la càrrega útil o *payload* i la signatura i estan separats per punts, de forma que té una aparença semblant a xxxxx.yyyyy.zzzzz. La capçalera indica quin algorisme s'utilitza en la signatura, la càrrega útil que conté, normalment, la informació sobre l'usuari i la signatura conté la capçalera i la càrrega útil codificades, i una clau privada.

MySQL

Figura 8. Logotip de MySQL

MySQL [2] és un sistema de gestió de base de dades *Open Source* conegut. Es va donar a conèixer a finals de la dècada dels 90. Es caracteritza per la seua fiabilitat, el seu rendiment i la seua estabilitat, a més és gratuït.

El servidor MySQL capta les peticions formulades pels clients i les transforma en un pla d'execució, seguidament recupera les dades segons la petició i les envia al client. Està compost per protocols de comunicació amb els clients, permisos d'accés als recursos, diferents tipus de registres de servidor, emmagatzemament de dades, memòria cau per a estalviar accessos al disc i l'optimització i execució de les peticions.

3. Anàlisi del problema

En aquest capítol entrarem en detall sobre els problemes que es presenten a l'intentar desenvolupar l'API plantejada en aquest TFG, els quals intentarem donar-los solució.

3.1 Identificació dels principals reptes

El problema principal a resoldre és la transferència de dades emmagatzemades entre un usuari extern i la plataforma. Ara mateix no existeix cap forma d'extraure ni modificar les dades que l'usuari promotor ha emmagatzemat en crear una promoció des de fora de la plataforma. Per tant, es pretén construir una interfície de programació per a satisfer la necessitat d'obtenció de les dades a partir d'un senzill enllaç URI sense necessitat de saber com està implementat.

A l'hora d'intercanviar les dades és convenient que seguisquen un format concret per tal de facilitar-ne el tractament. En recuperar la informació requerida de la base de dades no sempre s'obté d'una forma ordenada i unificada, igual que quan es rep informació externa normalment hi ha que tractant-la abans d'introduir-la en la base de dades. Per a resoldre aquest problema s'ha d'implementar un entorn d'integració de dades que les transforme en un dels tipus d'estàndards que són utilitzats per a la transmissió de dades, com ara bé, JSON, XML o CSV.

Una vegada resolts els anteriors problemes se'n presenta un de nou. Cada usuari té assignat un rol diferent dins la seua empresa i per tan dins la plataforma. És per això que no tots els usuaris han de tindre els mateixos privilegis a l'hora de fer servir l'API. Una diferenciació de rols permet tindre un control més específic del nombre d'usuaris que accedeixen als recursos i permet sols als usuaris de rol més avançat la realització d'operacions més sensibles.

Per a accedir als recursos de l'API és tan senzill com conèixer l'identificador de recursos uniforme. És ací on sorgeix l'últim problema, les dades són propietat de cada usuari i sols ell ha de tindre accés a elles. Per a evitar que un usuari alié a les dades tinga accés cal implementar un sistema d'autenticació, no obstant les API REST no mantenen la sessió i fa impossible la utilització de les variables de sessió tradicionals. Per a evitar aquesta situació s'ha d'implementar un model d'autenticació basat en *tokens*. Una vegada s'autentica l'usuari ha d'obtenir un identificador amb el qual realitzarà les consultes a l'API i aquesta comprovarà si l'usuari pot accedir al recurs que ha demanat.

3.2 Casos d'ús

A partir dels problemes identificats en l'apartat anterior, en aquest punt elaborarem els casos d'ús. Els casos d'ús ens permeten definir els requisits de l'API a desenvolupar des del punt de vista de l'usuari. La figura 3.1 mostra el diagrama de casos d'ús.

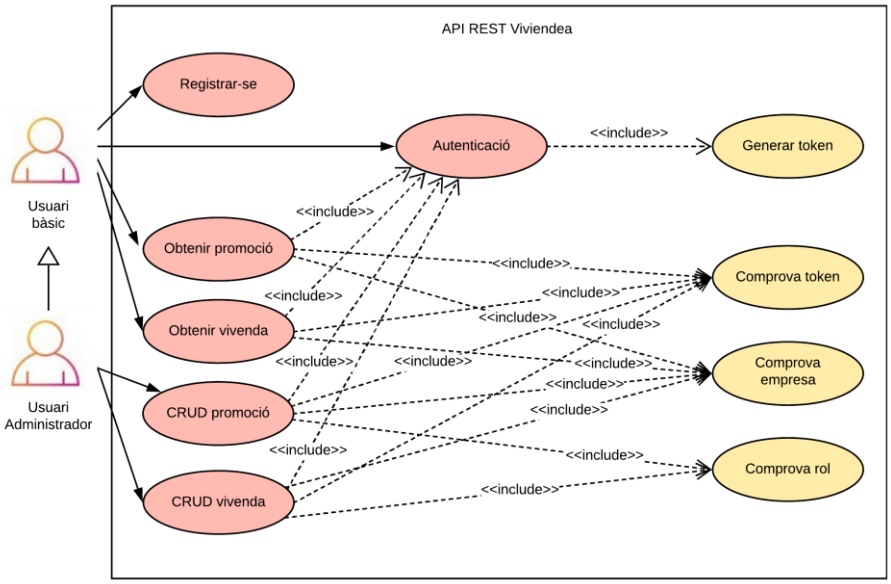

Figura 9. Casos d'ús

A continuació, es detalla cada cas d'ús.

Registrar usuari:

La primera vegada que algú utilitza l'API ha de registrar-se com a usuari.

Autenticació:

L'usuari introdueix les seues credencials i comprova si són vàlides. En cas de ser valides a l'usuari li és proporcionat un *token* únic. Quan s'executa la instància "Autenticació" utilitza el flux d'esdeveniment "Generar *token*" si les credencials son vàlides.

Obtenir Promoció:

Qualsevol usuari realitza una petició al sistema per a obtenir les característiques d'una promoció. El sistema comprova si el *token* de l'usuari és vàlid i si pertany a l'empresa que va crear la promoció. Quan s'executa la instància "Obtenir Promoció" sempre utilitza el flux d'esdeveniments de "Comprova empresa" i "Comprova *token*".

Obtenir Vivenda:

Qualsevol usuari realitza una petició al sistema per a obtenir les característiques d'una vivenda. El sistema comprova si el *token* de l'usuari és vàlid i si la promoció que conté la vivenda pertany a l'empresa de l'usuari. Quan s'executa la instància "Obtenir Promoció" sempre utilitza el flux d'esdeveniments de "Comprova empresa" i "Comprova *token*".

CRUD Promoció:

Un usuari administrador realitza una petició al sistema per a obtenir, crear, actualitzar o eliminar una promoció. El sistema comprova si el *token* de l'usuari és vàlid, si l'usuari pertany a l'empresa a la qual està assignada la promoció i si l'usuari té el rol d'administrador dins l'empresa. Quan s'executa la instància "CRUD Promoció" sempre utilitza el flux d'esdeveniments de "Comprova empresa", "Comprova *token*" i "Comprova rol".

CRUD Vivenda:

Un usuari administrador realitza una petició al sistema per a obtenir, crear, actualitzar o eliminar una vivenda. El sistema comprova si el token de l'usuari és vàlid, si la promoció que conté la vivenda pertany a l'empresa de l'usuari i si l'usuari té el rol d'administrador dins l'empresa. Quan s'executa la instància "CRUD Promoció" sempre utilitza el flux d'esdeveniments de "Comprova empresa", "Comprova *token*" i "Comprova rol".

3.3 Requisits funcionals i no funcionals

En aquest punt es detallen els requisits funcionals i no funcionals que hem identificat per a l'API a desenvolupar. Aquests requisits estan descrits des del punt de vista del sistema a diferència dels casos d'ús que es defineixen des del punt de vista de l'usuari. La definició d'aquests requisits ens permet tindre una visió més completa de les característiques que ha de tindre l'API.

Requisits Funcionals

RF01: Registrar usuari

Aquest requisit permet crear un usuari en la plataforma Viviendea sense necessitat d'accedir a ella.

RF02: Autenticació

Permet a l'usuari obtenir un *token* propi introduint l'usuari i la contrasenya, el qual farà servir posteriorment per a fer les cridades al sistema.

RF03: Crear Promoció

Ha de permetre a l'usuari administrador crear una nova promoció amb les característiques especificades en la sol·licitud i introduir-les en la base de dades.

RF04: Obtenir Promoció

Permet obtenir les característiques d'una promoció emmagatzemada en la base de dades que pertany a l'empresa de l'usuari

RF05: Obtenir totes les Promocions

L'usuari ha de poder obtenir les característiques de totes les promocions que pertanyen a la seua empresa emmagatzemades en la base de dades.

RF06: Actualitzar Promoció

Permet a l'usuari administrador actualitzar les característiques d'una promoció emmagatzemada en la base de dades que pertany a l'empresa.

RF07: Eliminar Promoció

L'usuari administrador ha de poder eliminar una promoció i les seues vivendes que pertanyen a la seua empresa.

RF08: Crear Vivenda

Permet crear una vivenda dins d'una promoció que pertany a l'empresa de l'usuari administrador

RF09: Obtenir Vivenda

Ha de permetre a l'usuari obtenir una vivenda que pertany a una promoció de la seua empresa.

RF10: Obtenir totes les vivendes d'una Promoció

Ha de permetre a l'usuari obtenir totes les vivendes que pertanyen a una promoció de la seua empresa.

RF11: Actualitzar Vivenda

L'usuari administrador ha de poder actualitzar una vivenda d'una promoció que pertany a la seua empresa.

RF12: Eliminar Vivenda

Ha de permetre que l'usuari administrador elimina una vivenda que pertany a una promoció de la seua empresa

Requisits No Funcionals

RNF01: Seguretat

Sols podran modificar i accedir a les dades les persones autoritzades, es a dir, les propietàries de dites dades. Es comprovarà el rol de cada usuari abans de permetre realitzar cap operació sol·licitada. Els perfils d'usuari seran usuari registrat i usuari administrador. Aquest requisit l'ampliarem amb un anàlisi en l'apartat 3.4

RNF02: Mantenibilitat

L'aplicació ha d'estar estructurada de manera intuïtiva per tal que siga fàcilment modificable, per tal que garantir la manejabilitat del sistema i resoldre els possibles errors en el menor temps possible. S'utilitzaran els mateixos patrons de disseny per a desenvolupar els mètodes dels diferents objectes per tal que tots estiguen estructurats de la mateixa forma.

RNF03: Escalabilitat

> Ha de permetre ampliar el nombre de mètodes i funcionalitats de forma senzilla i sense alterar l'estructura de l'aplicació i garantint l'escalabilitat.

RNF04: Protecció de dades

> Com que l'aplicació emmagatzema dades personals i sensibles de l'usuari en la base de dades, s'ha de tindre en conter el marc legal. Atesos a la Llei Orgànica de Protecció de Dades s'ha de garantir la privacitat, confidencialitat i disponibilitat de les dades de l'usuari. En l'apartat 3.5 farem un anàlisi més exhaustiu sobre aquest requisit.

3.4 Anàlisi de la seguretat

La principal funció de l'aplicació és la transferència de dades. Quan l'usuari es registra es transfereixen dades personals i sensibles de l'usuari, per tant és important que les connexions entre el client i el servidor estiguen xifrades en la capa de transport. Per a evitar que un agent extern modifique les dades transferides han de ser xifrades per l'estandar TLS que s'utilitza en el protocol HTTPS. El protocol TLS permet mantenir una connexió privada en Internet i verifica que les dades estan xifrades i no han sigut falsificades.

L'aplicació requerirà l'autenticació de l'usuari a l'hora de fer peticions a les rutes URI per tal d'evitar que es processen peticions sense autorització. Quan l'usuari inicie sessió el servidor li proporcionarà un *token*. Amb aquest *token* l'usuari podrà realitzar peticions introduint-lo en la capçalera *Authorization* de la sol·licitud i podrà accedir als recursos que li són permesos.

Al tindre un sistema d'autenticació, l'aplicació conté una taula en la base de dades on s'emmagatzemen les credencials de l'usuari. Com que molts usuaris utilitzen la mateixa contrasenya per a diversos llocs web, és fonamental emmagatzemar aquest camp xifrat per si la base de dades en algun moment es veiés compromesa. Per a codificar les contrasenyes s'utilitzarà la funció criptogràfica *hash*. La funció criptogràfica *hash* codifica les dades, sense importar la longitud d'aquestes, en una cadena única de caràcters utilitzant una sèrie de processos matemàtics i lògics. Quan l'usuari inicie sessió s'utilitzarà un mecanisme per a verificar de manera interna que el text introduït per l'usuari correspon a la contrasenya emmagatzemada.

3.5 Anàlisi del marc legal i ètic

Anàlisi de protecció de dades

Com s'ha esmentat abans, l'aplicació emmagatzema dades personals i sensibles de l'usuari en la base de dades, per tant s'ha de tindre en conter el marc legal per garantir la privacitat, confidencialitat i disponibilitat de les dades de l'usuari.

Atesos a la Llei Orgànica de Protecció de Dades [3] Viviendea recull les polítiques de privacitat [14] següents:

- Les dades sols es recullen, es tracten i s'utilitzen per a gestionar correctament les sol·licituds d'informació requerides a través dels formularis pertinents i per a remetre informació periòdica sobre les novetats de l'empresa.

- La base legal per al tractament de les seues dades es el propi consentiment de l'usuari.
- Les dades seran emmagatzemades el temps necessari per a complir amb les obligacions legals establertes.

- Les dades personals podran ser comunicades a promotors o cooperatives que intervinguin en la construcció i promoció de les vivendes dissenyades pels usuaris.

- En cas de necessitar comunicar les dades personals a altres tipus de tercers es sol·licitarà el consentiment abans de realitzar la comunicació.

- Les dades son emmagatzemades en el sistema d'informació en el que s'han implantat mesures de seguretat tècniques i organitzatives per a previndre qualsevol pèrdua o ús no autoritzat de tercers.

- Per visitar el portal web o utilitzar els serveis de Viviendea no s'emmagatzema de cap forma automàtica cap dada de caràcter personal. No obstant s'utilitzen *cookies* durant la navegació.

- Tots els usuaris poden exercir qualsevol dret atorgat per la normativa de protecció de dades, com el dret d'accés, rectificació, limitació del tractament, supressió portabilitat de dades i oposició.

3.3 Pla de treball

Per a desenvolupar el projecte s'han dividit les tasques que es mostren en el diagrama de Gantt i s'expliquen a continuació:

- Identificació dels casos d'ús, que ens proporcionen els escenaris en els quals l'usuari interactua amb el sistema.

- Definició de requisits, ens detallen quines característiques més concretes ha de tindre l'API.

- Recerca de ferramentes, on s'avaluen quines ferramentes serien les adequades per a desenvolupar el projecte.

- Desenvolupament de l'API REST, on es definiran l'estructura, les operacions a realitzar, els permisos dels usuaris i l'autenticació per *token*.

- En la implantació s'engegarà el servidor *Express JS* i es comprovarà que no hi haja cap error de sintaxi.

- Es realitzaran sol·licituds per a comprovar que tots els requisits es compleixen i el funcionament és el que s'esperava una vegada finalitzat.

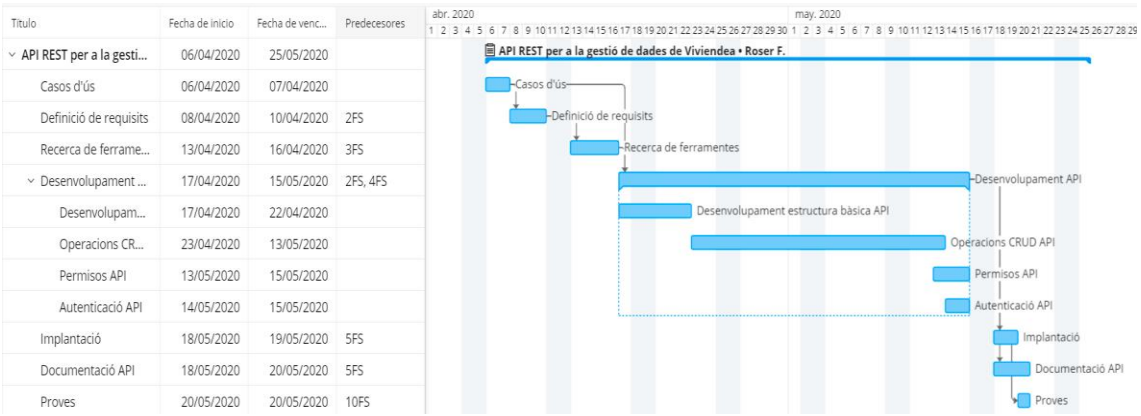

Figura 10. Diagrama de Gantt

4. Disseny de la solució

Una vegada analitzada la problemàtica del projecte en aquest capítol donem pas al disseny de la solució.

4.1 Arquitectura del Sistema

En la Figura 11 es mostra l'arquitectura client/servidor que s'emprarà per a desenvolupar la solució. L'arquitectura està composta per tres nivells. A continuació es procedirà a explicar cada un dels nivells.

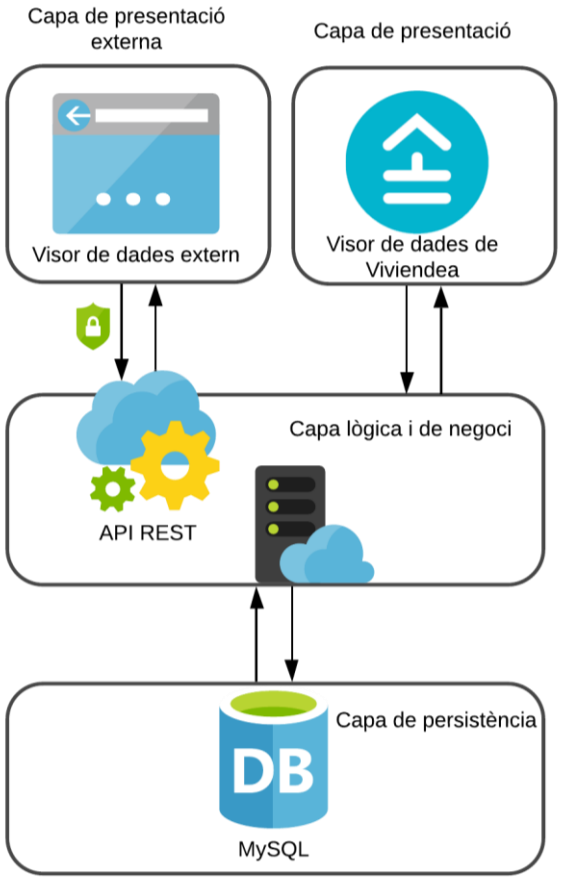

Figura 11. Arquitectura del sistema

Capa de presentació. Es el nivell més alt de l'arquitectura, on es troben els clients externs. En el nostre cas, existeix com a client la pàgina web de Viviendea, que és un visor de dades dinàmic, i altres visors de dades externs, com el desenvolupat per l'empresa Gestión [16].

Capa de lògica i negoci. Es el nivell inter-mig de l'arquitectura, on es dona suport a la lògica de negoci. Aquesta capa és l'encarregada de implementar la funcionalitat que ofereix el sistema. Aquesta funcionalitat està relacionada en el tractament de dades i amb regles de seguretat.

L'API desenvolupada en aquest treball es troba en el nivell d'aplicació on s'executa el servidor. L'API és un connector incrustat en la capa de lògica i negoci que la connecta amb la capa de presentació externa a la plataforma. D'aquesta manera quan el client realitze una petició des de la capa de presentació, l'API serà l'encarregada de gestionar-la i transmetre-la a la capa lògica i de negoci de la plataforma. La petició una vegada processada serà enviada a la capa de persistència. A més la petició del client es realitzarà de forma segura tal com hem analitzat en l'apartat 3.4.

Capa de persistència. Per últim, el nivell més baix d'aquesta arquitectura és la capa de persistència formada per un gestor de dades MySQL. Aquest nivell es comunica amb la capa lògica i de negoci que proporcionarà o gestionarà les dades de la petició. A continuació, en la Figura 11 es mostra la relació de taules que s'utilitzaran en aquest projecte per a facilitar la comprensió del següent apartat.

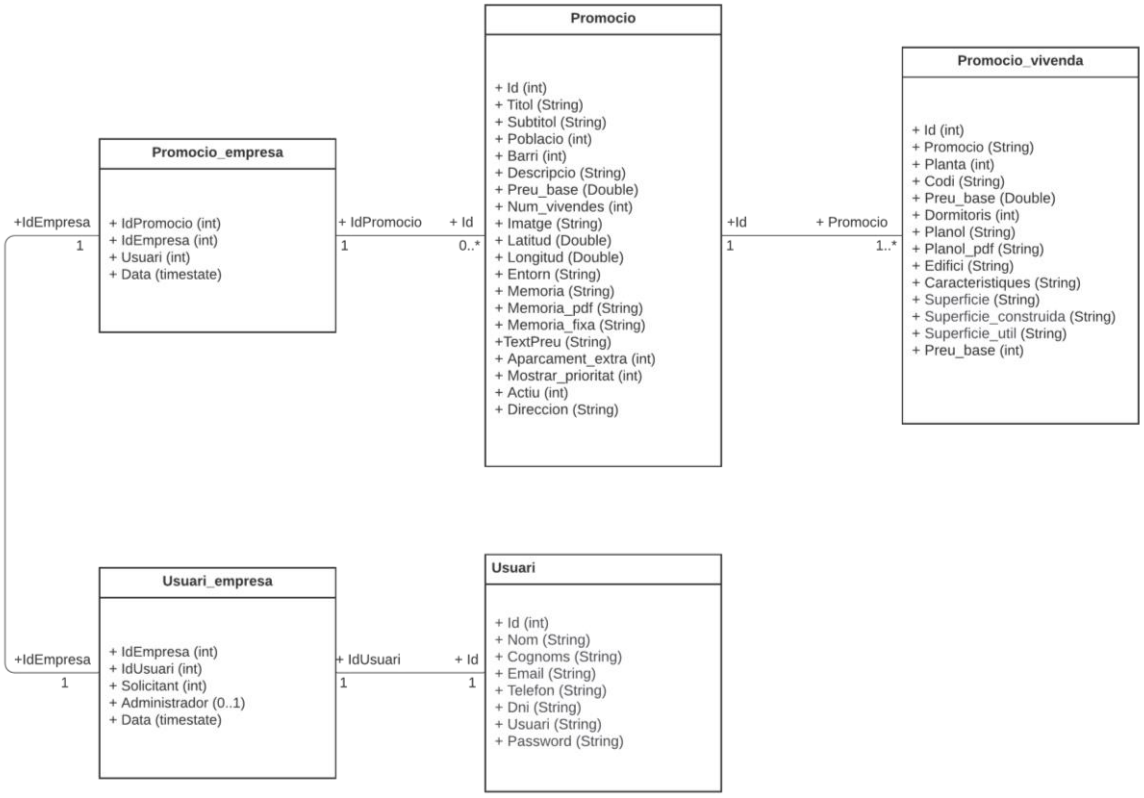

Figura 12. Diagrama UML de la base de dades

4.2 Disseny detallat de l'API REST

Una vegada explicada l'arquitectura del sistema, anem a detallar el disseny de la API REST amb més profunditat.

L'estructura de la api està formada per quatre blocs.

El servidor, el middleware d'autenticació, els controladors i els mòduls de la capa lògica.

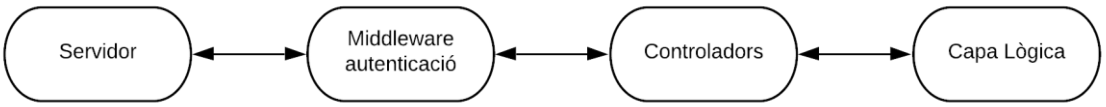

Figura 13. Estructura de l'API REST

- El servidor escolta en el port 443, així cada petició que es realitze aquest port es redirigirà segons la ruta a què es reba per la petició HTTPS.

- Middleware d'autenticació, s'encarrega de comprovar si el *token* enviat en la sol·licitud és vàlid. Si és així passa la petició al controlador.

- Els controladors, es dediquen a filtrar i obtenir les dades d'interés, que arriben en la petició. Posteriorment, enviaran aquestes dades a la capa lògica de negoci perquè s'executen les funcions pertinents.

- Els mòduls de la capa lògica, s'encarreguen de processar, transformar i enviar les dades a la capa de persistència. És la que realitza les consultes a la base de dades.

Servidor

El servidor està compost per una aplicació d'Express, aquesta aplicació encamina les peticions HTTPS segons el verb HTTP utilitzat i l'estructura de l'URL. S'ha afegit el mòdul *body-parser* de NodeJS que permet processar sols les peticions amb la capçalera *Content-Type* amb l'opció *application/json*, açò significa que sols llegirà peticions en format JSON per tal de facilitar l'extracció de dades del cos de la petició. Per un altre costat crea i defineix el servidor, en el que s'especifica en quin port escolta, en el nostre cas el 443.

Middleware d'autenticació

El *middleware* d'autenticació comprova si en la capçalera de la petició s'ha inclòs el camp *Authorization* junt amb el *token* corresponent de l'usuari. En cas que siga proporcionat el descodifica i comprova si està xifrat amb la clau privada i si ha expirat en el temps. Si el *token* és vàlid, sols aleshores, envia la petició al controlador identificant quin usuari està realitzant-la.

Controladors

Els controladors, tal com s'ha esmentat, filtren les dades que arriben en la petició i envien a la capa lògica les necessàries per a tractar les dades. A més, porten un control dels errors executats en cas que existisquen i proporcionen el codi de resposta de l'HTTP. Els codis definits en els controladors són els següents:

Tipus	Descripció
200	OK
400	Bad Request
401	Unauthorized
403	Forbidden
500	Internal Error

Taula 1. Codis de resposta

200 *OK:* Aquest codi indica que la petició s'ha acceptat, i s'ha realitzat correctament i en el cas que es requerisca es retornen les dades sol·licitades.

400 *Bad Request:* La sol·licitud s'ha realitzat de forma errònia i el servidor no pot processar-la. Pot ser degut a una sol·licitud mal formulada o a un error de sintaxis.

401 *Unauthorized:* Aquest error es dóna a l'hora d'autentificar a l'usuari en el cas que faça servir un *token* que no siga vàlid.

403 *Forbidden:* Aquest codi indica que les credencials de l'usuari són correctes però no té l'autorització i privilegis per a accedir al recurs sol·licitat.

500 *Internal Error:* El servidor és incapaç de processar una petició realitzada per algun error intern.

Capa lògica

En la capa lògica de l'API podem trobar les classes Promoció, Vivenda, Usuari i NouUsuari. En la Figura 14 es mostra el diagrama de classes per tal de facilitar la visualització de les relacions entre les classes que conformen el sistema.

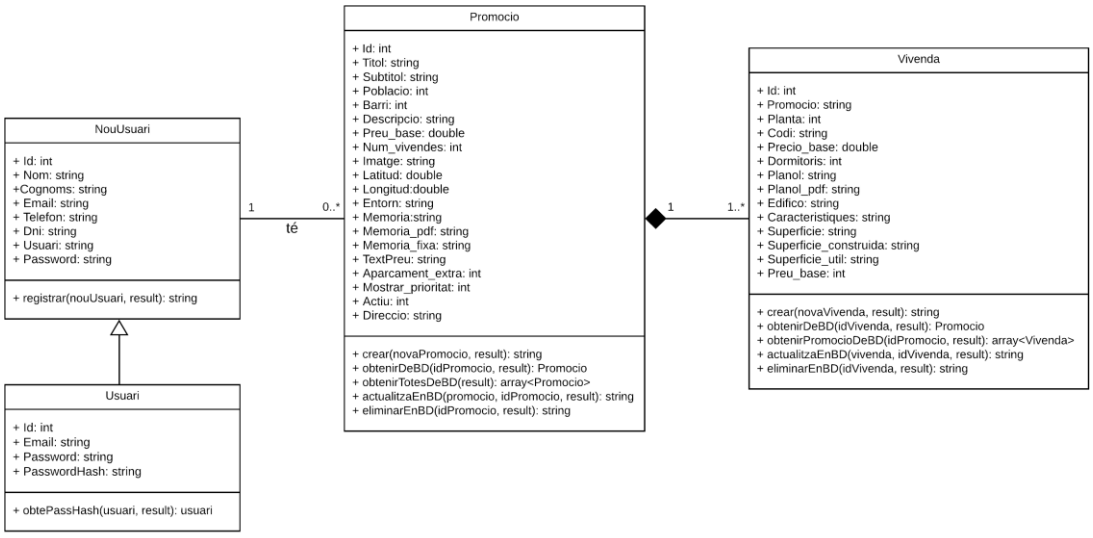

Figura 14. Diagrama de classes en UML

Com veiem al diagrama de classes cada classe està composta pel nom, un nombre d'atributs i els mètodes que la componen. Tots els atributs pels quals estan compostes les classes són públics, és a dir, són accessibles des de dins i fora de la classe perquè puguen relacionar-se entre elles, a més, està indicat quin tipus de dada és cada atribut. Els mètodes també són públics, ja que en alguns casos una classe crida als mètodes d'una altra. Entre parèntesis observem els arguments que es proporcionen a l'hora d'executar el mètode, totes elles contenen l'argument *result,* ja que són funcions cridades per funcions *callback* des dels controladors.

Les funcions *callback* permeten invocar un mètode amb una altra funció com argument. Per tant l'argument *result* serà el resultat de totes les instruccions executades en el mètode i una vegada finalitzades el controlador podrà seguir amb la seua execució.

La classe NouUsuari recull totes les dades de l'usuari necessàries per a crear un perfil en la plataforma Viviendea. Segons es veu al diagrama un usuari pot tindre un nombre indefinit de promocions.

La classe Usuari hereta de la classe NouUsuari tots els atributs i els mètodes agafant així totes les característiques de les quals es compon un usuari en registrar-se i afegint un atribut per a la contrasenya encriptada. En aquesta classe es comprovarà si la contrasenya introduïda per l'usuari i l'emmagatzemada en la base de dades coincideixen, de ser així proporcionarà un *token* a l'usuari amb l'identificador, l'e-mail, la contrasenya xifrada, la data actual i la data d'expiració.

La classe Promoció està composta per totes les característiques que ha de tindre una promoció en la plataforma Viviendea. Els mètodes definits en aquesta classe corresponen a cada una de les operacions CRUD que es poden realitzar sobre una promoció. En el cas de l'obtenció podem diferenciar dos mètodes, l'obtenció d'una única promoció especificant l'identificador com a paràmetre i l'obtenció de totes les promocions pertanyents a l'usuari. Tota promoció ha d'estar composta de mínim una vivenda.

La classe Vivenda, igual que la classe Promoció, està composta pels atributs necessaris per a definir una vivenda en la plataforma Viviendea. Els mètodes implementats corresponen a les operacions CRUD que es poden realitzar sobre una vivenda. A diferència de la classe Promoció, el mètode de l'obtenció de totes les vivendes s'obtenen les que pertanyen a una promoció en concret, la qual s'ha d'especificar l'identificador d'aquesta.

5. Desenvolupament de la solució proposada

Una vegada presentat el disseny de la solució, en aquest capítol procedirem a descriure com s'ha desenvolupat l'API pas per pas.

Com s'ha esmentat en el capítol anterior, l'API està estructurada en capes. Per a respectar els principis de la construcció de codi orientat a objectes, s'ha optat per utilitzar la programació modularitzada. El codi s'ha dividit en mòduls, un mòdul rep com entrada les dades d'entrada del sistema o l'eixida d'un altre mòdul. En el nostre cas estan organitzats jeràrquicament, és a dir, el mòdul principal realitzarà les cridades pertinents als mòduls de nivells inferiors. A l'hora d'explicar els diferents mòduls intentarem mantenir l'alta cohesió de manera que, explicarem en blocs els diferents fitxers que contenen funcionalitats semblants corresponents als mòduls.

L'estructura la separarem en els següents blocs:

- Servidor: On es troba el mòdul que compon el servidor i emmagatzemen els atributs estàtics de l'aplicació.
- *Middlewares*: On s'allotgen els mòduls que componen del *middleware* d'autentificació i el mapatge de les rutes URI
- Controladors: On es troben els mòduls que gestionen les peticions que es realitzen.
- Lògica: On s'emmagatzemen els mòduls de la capa lògica, aquests mòduls processen les dades i gestionen la connexió i accés a les dades.

Els mòduls tenen dependències entre sí, en la següent il·lustració es mostra la relació que existeix entre ells.

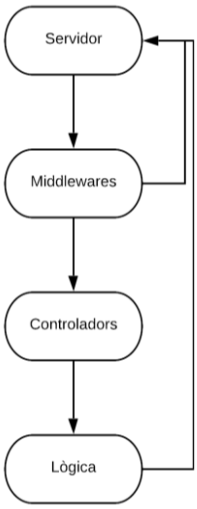

Figura 15. Relació entre mòduls

Una vegada comentada l'estructura i la relació que existeix entre els blocs, comentarem la funcionalitat de cada bloc de mòduls implementat.

Servidor

L'arxiu *server.js* correspon al mòdul principal, el del servidor. Com s'ha comentat en el capítol anterior, està compost per una aplicació d'Express. La principal funció d'aquesta aplicació és carregar els mòduls d'Express i mantenir el servidor HTTP actiu i escoltant en el port 443. A més, estableix el valor de la clau d'encriptació del *token,* que extrau del fitxer de configuració, com a una propietat de l'aplicació. Quan arribe una petició requerirà l'execució del mòdul del *middleware* d'encaminament.

La configuració principal de l'aplicació es troba en el document *config.js*, en ell es defineix un mòdul amb els valors del servidor, el port, l'usuari administrador de la base de dades i la seua contrasenya, la base de dades que s'utilitzarà per a realitzar les consultes i la clau privada per a encriptar els *tokens*. Tota aquesta informació s'emmagatzema ací, ja que no són paràmetres que s'hagen de modificar constantment.

Middlewares

El bloc *Middlewares* consta de dos fitxers, *rutes.js* i *middleware.js*, els quals procedirem a explicar.

L'arxiu *rutes.js* correspon al *middleware* d'encaminament. Està configurat per a mapetjar les rutes que arriben en la petició i el mòdul del controlador que s'ha d'executar. Quan la ruta està protegida envia les dades al *middleware* d'autenticació. Per a redirigir les peticions ho fa com es mostra en les següents línies de codi, on trobem un exemple d'una ruta protegida i una ruta lliure.

```
//Ruta lliure
const usuari = require("../controladors/usuari_controlador.js");
app.post("/registre", usuari.registrar);

//Ruta protegida
const vivenda = require("../controladors/vivenda_controlador.js");
app.get("/vivenda/:vivendaId", ruta.autorizacio, vivenda.obtenir);
```

D'aquesta manera quan es cride a l'adreça api.viviendea.com seguit de les rutes que es mostren a continuació es redirigiran a cada un dels mòduls, seguint la relació de cada una de les rutes que es mostren a continuació:

Mètode	URI	Funció	Protegida
POST	/registre	Registrar usuari	No
POST	/token	Autenticació	No
POST	/promocio	Crear Promoció	Si
GET	/promocio/{id}	Obtenir promoció	Si
GET	/promocio	Obtenir totes les promocions d'un usuari	Si
PUT	/promocio/{id}	Actualitzar promoció	Si
DELETE	/promocio/{id}	Eliminar promoció	Si
POST	/vivenda/{id}	Crear vivenda	Si
GET	/vivenda/{id}	Obtenir vivenda	Si
GET	/vivenda/promocio/{id}	Obtenir totes les vivendes d'una promoció	Si
PUT	/vivenda/{id}	Actualitzar vivenda	Si
DELETE	/vivenda/{id}	Eliminar vivenda	Si

Taula 2. Rutes de l'API.

L'arxiu *middleware.js* es correspon al *middleware* d'autentificació. Aquest *middleware* es crida en aquelles rutes que estan protegides. Consta d'un mòdul que comprova, en primer lloc, si la sol·licitud ha estat realitzada junt amb una capçalera d'autorització. En cas d'existir, extraurà el *token* de la capçalera i el descodifica utilitzant la clau privada. Una vegada descodificada la càrrega útil, comprovarà si el paràmetre on s'emmagatzema la data d'expedició ha vençut o segueix vigent. Si es compleixen tots els requisits, es cridarà al següent mòdul, el qual serà a un dels controladors, i s'afegirà a la petició l'identificador de l'usuari emmagatzemat en el *token*.

Controladors

Aquest bloc està compost pels mòduls que formen els controladors, per semblança de funcionalitats explicarem els controladors que contenen els arxius *promocio_controlador.js* i *vivenda_controlador.js* per un costat i l'arxiu *usuari_controlador.js* per l'altre.

Els controladors de *promocio_controlador.js* i *vivenda_controlador.js* consten de cinc mòduls cada un, que seran exportats per tal de ser accessibles des d'altres funcions. Els mòduls els hem anomenat de la següent manera: *guardar*, *obtenir*, *obtenirTot*, *actualitzar* i *eliminar*. Aquests mòduls obtenen els paràmetres de la sol·licitud i segons la funció que han de desenvolupar envien uns paràmetres o uns altres als mòduls del bloc Models corresponents.

En els mòduls *guardar* i *actualitzar* es comprovarà si el cos del missatge està buit i en cas d'estar-ho enviarà el codi d'estat 400. Si el missatge es correcte, es crearà un objecte Promocio o Vivenda al que s'assignarà a cada atribut el valor enviat en el cos del missatge.

Una vegada s'han enviat els paràmetres necessaris als mòduls de la capa lògica, els controladors esperaran a rebre una resposta *callback.* En cas de rebre un error gestionarà el missatge i el codi d'estat a respondre segons el tipus que siga. Si la funció s'ha executat correctament i no hi ha cap error retornarà en format JSON la resposta enviada per la capa lògica. Es mostra seguidament en un fragment de codi del mòdul *obtenir* de *promoció_controlador.js*.

```
if (err) {
  if (err.kind === "Not_Allowed") {
    res.status(403).send({
      message: "No té permisos per a realitzar aquesta acció"
    });
  } else if (err.kind === "Not_Found") {
    res.status(404).send({
      message: "No s'ha trobat la promoció amb ID: " + req.params.promocioId
    });
  } else {
    res.status(500).send({
      message: "Error al recuperar la promoció amb ID:  " + req.params.promocioId
    });
  }
} else res.send(JSON.stringify(data));
```

El controlador *user_controlador.js* té un funcionament similar, no obstant està format de dos mòduls, *token* i *registrar*. En ambos mòduls comprova si el cos del missatge és buit, ja que per a realitzar aquestes funcions és necessari que l'usuari introduïsca al menys els paràmetres obligatoris. Igual que en els controladors de Promoció i Vivenda crearà un objecte amb els valors enviats i en cas de rebre un error en la resposta *callback* gestionarà el missatge i el codi d'estat. Si no rep cap error, retornarà el missatge enviat pels mòduls de la capa lògica.

Lògica

En aquest bloc és on trobem tots els mòduls que implementen la capa lògica.

Els mòduls que transformen les dades els explicarem, igual que amb els controladors; per un costat, *promocio_model.js* i *vivenda_model.js* i per l'altre costat explicarem *usuari_model.js*, ja que les funcionalitats són un poc distintes.

El fitxer *usuari_model.js* està compost pels mòduls *autenticacio*, *registrar* i una funció *rol(idUsuari, idPromocio, callback)*. El mòdul *autenticacio* comprova que l'e-mail i la contrasenya enviats per l'usuari corresponen amb les dades emmagatzemades en la base de dades, si es verifica correctament crea un *token* amb l'identificador de l'usuari, l'e-mail, la contrasenya, la data actual i la data de caducitat, una vegada xifrat amb la clau privada l'envia al controlador. El mòdul *registrar* obté els valors dels paràmetres rebuts pel controlador, xifra la contrasenya i ho emmagatzema en la base de dades. A més, s'ha definit una funció *rol(idUsuario, idPromocion, callback)* on es comprovarà si l'usuari pertany a l'empresa propietària de la promoció passada per paràmetres i en cas de pertànyer indicarà quin rol té l'usuari dins l'empresa, és a dir, si és treballador o administrador.

Els fitxers *promocio_model.js* i *vivenda_model.js* estan formats per cinc mòduls, igual que els controladors. Aquests mòduls s'han anomenat *crearEnBD, obtenirDeBD, obtenirTotsDeBD, actualitzarEnBD* i *eliminarDeBD.* Tots els mòduls cridaran la funció *rol(idUsuario, idPromocion, callback)* d'*usuari_model.js* per a saber si l'usuari té permís per a accedir a les dades o realitzar alguna modificació en aquestes. En el cas d'*obtenirDeBD* i *obtenirTotsDeBD* sols es comprovarà si l'usuari pertany a l'empresa propietària de la promoció, en la resta de mòduls comprovarà a més si l'usuari té el rol

Administrador. Aquestes comprovacions es realitzaran tant en *promocio_model.js* com en *vivenda_model.js*. Si l'usuari no té permisos per a realitzar l'acció s'enviarà un error "*Not_allowed*" al controlador.

Una vegada autoritzat a realitzar les accions cada mòdul executarà una consulta a la base de dades on s'inseriran, s'obtindran, s'actualitzaran o s'eliminaran les dades segons el mòdul executat. En cas d'obtenir un error intern al servidor de la base de dades s'enviarà al controlador l'error.

Una vegada explicats els mòduls que processen i transformen les dades, passem a explicar els mòdul que estableix la connexió amb la capa de persistència dins de la capa lògica.

Aquest mòdul enllaça la capa lògica amb la capa de persistència, és a dir, crearà la connexió entre l'aplicació i la base de dades per a que estiga disponible en cas de ser necessari. Per a crear la connexió és requerirà el mòdul MySQL de NodeJS i sol·licitarà la configuració de la base de dades emmagatzemada en el mòdul de configuració tal i com es mostra a continuació.

```
const mysql = require("mysql");
const dbConfig = require("../config.js");

const connection = mysql.createConnection({
  multipleStatements: true,
  host: dbConfig.HOST,
  user: dbConfig.USER,
  password: dbConfig.PASSWORD,
  port: dbConfig.PORT,
  database: dbConfig.DB
});
```

S'ha definit que la connexió serà de declaració múltiple, açò implica que el servidor tindrà l'autorització per a enviar diverses consultes al servidor de base de dades separades pel caràcter punt i coma.

Una vegada definida la configuració, establirà la connexió amb la base de dades mitjançant la funció *connect()*. Si a l'hora de realitzar la connexió amb la base de dades sorgeix un error, s'invocarà la funció de reconnexió, que tanca les connexions establertes i intenta connectar de nou amb la base de dades. Quan l'aplicació deixe de realitzar consultes es tancarà la connexió amb la base de dades.

6. Contracte d'integració

A continuació es detallen les funcionalitats i els paràmetres d'entrada i eixida de cada un dels serveis web que presta l'API descrivint així com s'han de consumir els recursos correctament.

6.1 Registrar usuari

Aquest servei permet registrar un usuari en la plataforma Viviendea.

Paràmetres entrada

El servei rep com entrada un objecte *NouUsuari.* En aquest objecte s'han de proporcionar obligatòriament els atributs *Nom, Cognoms, Email, Password* i *Usuari.* A continuació es detallen les característiques de cada atribut i s'inclouen d'altres opcionals.

Paràmetre	Descripció	Tipus	Màscara	Longitud màxima	Obligatori
Nom	Nom del client	String		50	Si
Cognoms	Cognoms del client	String		150	Si
Dni	NIF/CIF/NIE correcte	String	XNNNNNNX	8	No
Telefon	Telèfon del client	String	+NN NNNNNNNNN	20	No
Email	Email del client	String	usuari_email@domini.extensió	150	Si
Password	Contrasenya del client	String	Combinació majúscules, minúscules i símbols	100	Si
Usuari	Usuari del client	String	Text sense espais en blanc	250	Si

Taula 3. Atributs NouUsuari

Paràmetres eixida

El servei retorna com a eixida un objecte *Resposta.* El valor de la resposta és el codi d'estat de l'operació. Aquests codis estan especificats en la Taula 1 del capítol 4.

6.2 Autenticació

Aquest servei genera un *token* vàlid per a l'usuari.

Paràmetres entrada

El servei rep un objecte *Usuari*, on tots els paràmetres, detallats a continuació, són obligatoris.

Paràmetre	Descripció	Tipus	Màscara	Longitud màxima	Obligatori
Email	Email de l'usuari	String	usuari_email@domini.extensió	150	Si
Password	Contrasenya de l'usuari	String	Combinació majúscules, minúscules i símbols	100	Si

Taula 4. Atributs Usuari

Paràmetres eixida

El servei retorna un objecte *Resposta.* Aquesta resposta conté un objecte *Token.* El paràmetre retornat es mostra en la següent taula.

Paràmetre	Descripció	Tipus	Màscara	Obligatori
Token	Token únic assignat a l'usuari	String	xxx.yyy.zzz	Si

Taula 5. Atribut Token

6.3 Crear promoció

Aquest servei permet crear una promoció en la plataforma Viviendea.

Paràmetres entrada

El servei rep un objecte de tipus *Promoció*. En aquest objecte s'han d'especificar obligatòriament els atributs *Titol, Poblacio, Direccio, Num_vivendes, Aparcament_extra, Latitud, Longitud, Preu_base, Text_preu* i *Actiu.* A més, i hi ha altres atributs que es poden afegir opcionalment com es mostra en la següent taula.

Paràmetre	Descripció	Tipus	Màscara	Longitud màxima	Obligatori
Titol	Titol de la promoció	String		100	Si
Subtitol	Subtitol de la promoció	String		200	No
Poblacio	Població en la qual s'ubica la promoció	String		80	Si
Barri	Barri de la població	String		80	No
Direccio	Adreça de la localitat on es troba la promoció	String		80	Si
Descripcio	Descripció de la promoció	String		1000	No
Num_vivendes	Nombre de vivendes	Integer		11	Si
Aparcament_extra	0 = No 1 = Sí	Integer	[0-1]	1	Si
Imatge	Nom de la imatge a carregar	String		100	No
Latitud	Coordenades de la ubicació	Double	Decimal (10,6)	16	Si
Longitud	Coordenades de la ubicació	Double	Decimal (10,6)	16	Si
Preu_base	Preu base de la promoció	Double	Decimal (10,3)	13	Si
Text_preu	Preu en numerals	String		250	No
Actiu	La promoció es mostra en la plataforma Viviendea. 0 = No 1 = Sí	Integer	[0-1]	1	Si

Taula 6. Paràmetres entrada de Crear promoció

A més per a sol·licitar aquest servei en la capçalera de la petició s'ha d'introduir el camp Authorization amb un *token* vàlid.

Paràmetre	Descripció	Tipus	Mascara	Obligatori
Authorization	*Token* vàlid assignat a l'usuari	String	Bearer xxx.yyy.zz	Si

Taula 7. Capçalera Authorization

Paràmetres eixida

El servei retorna un objecte *Resposta* on s'indica l'identificador assignat a la promoció creada.

Paràmetre	Descripció	Tipus	Màscara	Obligatori
Id	Identificador de la promoció creada	Integer		Si

Taula 8. Atribut Id de Promoció

6.4 Obtenir promoció

Aquest servei permet obtenir una promoció que pertany a l'empresa on treballa l'usuari.

Paràmetres entrada

El servei rep com entrada un objecte *Petició*, on en la ruta del servei s'ha d'especificar obligatòriament el camp *Id* de la promoció, les especificacions d'aquest camp són les mateixes que es troben en la Taula 8. A més, s'ha de proporcionar el camp *Authorization* en la capçalera com es mostra en la Taula 7.

Paràmetres eixida

El servei retorna un objecte *Resposta*. En la resposta es troba un objecte Promoció que proporcionarà els mateixos atributs, amb les mateixes característiques, que els que es proporcionen en la Taula 6.

6.5 Obtenir totes les promocions

Aquest servei permet obtenir una llista amb totes les promocions de l'empresa a la qual pertany l'usuari.

Paràmetres entrada

El servei rep un objecte *Petició.* A més, s'ha de proporcionar el camp *Authorization* en la capçalera com es mostra en la Taula 7.

Paràmetres eixida

El servei respon amb un objecte *Resposta*. En la resposta conté una llista amb els atributs següents de cada promoció.

Paràmetre	Descripció	Tipus	Màscara	Longitud màxima
Id	Identificador de la promoció	Integer		11
Titol	Títol de la promoció	String		100
Subtitol	Subtítol de la promoció	String		200
Poblacio	Població en la qual s'ubica la promoció	String		80
Barri	Barri de la població	String		80
Direccio	Adreça de la localitat on es troba la promoció	String		80
Descripcio	Descripció de la promoció	String		1000
Num_vivendes	Nombre de vivendes	Integer		11
Aparcament_extra	0 = No 1 = Sí	Integer	[0-1]	1
Imatge	Nom de la imatge a carregar	String		100
Latitud	Coordenades de la ubicació	Double	Decimal (10,6)	16
Longitud	Coordenades de la ubicació	Double	Decimal (10,6)	16
Preu_base	Preu base de la promoció	Double	Decimal (10,3)	13
Text_preu	Preu en numerals	String		250
Actiu	0 = No 1 = Sí	Integer	[0-1]	1

Taula 9. Atributs Promocio

6.6 Actualitzar promoció

Aquest servei permet actualitzar una promoció emmagatzemada en la plataforma Viviendea.

Paràmetres entrada

El servei rep un objecte de tipus *Promoció*, on en la ruta del servei s'ha d'especificar obligatòriament el camp *Id* de la promoció, les especificacions d'aquest camp són les mateixes que es troben en la Taula 8. Els paràmetres a enviar són els mateixos que els especificats en la Taula 6, a excepció que cap d'ells és obligatori. A més, s'ha de proporcionar el camp *Authorization* en la capçalera com es mostra en la Taula 7.

Paràmetres eixida

El servei retorna com a eixida un objecte *Resposta.* El valor de la resposta és el codi d'estat de l'operació. Aquests codis estan especificats en la Taula 1 del capítol 4.

6.7 Eliminar promoció

Aquest servei permet eliminar una promoció emmagatzemada en la plataforma Viviendea.

Paràmetres entrada

El servei rep com entrada un objecte *Petició*, on en la ruta del servei s'ha d'especificar obligatòriament el camp *Id* de la promoció*,* les especificacions d'aquest camp són les mateixes que es troben en la Taula 8. A més, s'ha de proporcionar el camp *Authorization* en la capçalera com es mostra en la Taula 7.

Paràmetres eixida

El servei retorna com a eixida un objecte *Resposta.* El valor de la resposta és el codi d'estat de l'operació. Aquests codis estan especificats en la Taula 1 del capítol 4.

6.8 Crear vivenda

Aquest servei permet crear una vivenda dins d'una promoció en la plataforma Viviendea.

Paràmetres entrada

El servei rep un objecte de tipus *Vivenda*. En aquest objecte s'han d'especificar obligatòriament els atributs *Promocio, Planta, Dormitoris* i *Preu_base.* A més i hi ha altres atributs que es poden afegir opcionalment com es mostra en la següent taula.

Paràmetre	Descripció	Tipus	Màscara	Longitud màxima	Obligatori
Promocio	Identificador de la promoció	Integer		11	Si
Codi	Codi de la vivenda	String		50	No
Planta	Altura de l'edifici on es troba ubicada la vivenda	Integer		10	Si
Dormitoris	Nombre de dormitoris que conté la vivenda	Integer		50	Si
Planol	Plànol de la vivenda	String		100	No
Superficie	Superfície total de la vivenda	Float			No
Superficie_construida	Superfície construïda de la vivenda	Float			No
Superficie_util	Superfície útil de la vivenda	Float			No
Preu_base	Preu base de la vivenda	Double	Decimal (10,3)	13	Si

Taula 10. Paràmetres entrada de Crear vivenda

Paràmetre eixida

El servei retorna un objecte *Resposta* on s'indica l'identificador assignat a la vivenda creada.

Paràmetre	Descripció	Tipus	Màscara	Obligatori
Id	Identificador de la vivenda creada	Integer		Si

Taula 11. Atribut Id de Vivenda

6.9 Obtenir vivenda

Aquest servei permet obtenir una vivenda que pertany a una promoció de l'empresa on treballa l'usuari.

Paràmetres entrada

El servei rep com entrada un objecte *Petició*, on en la ruta del servei s'ha d'especificar obligatòriament el camp *Id* de la vivenda, les especificacions d'aquest camp són les mateixes que es troben en la Taula 11. A més, s'ha de proporcionar el camp *Authorization* en la capçalera com es mostra en la Taula 7.

Paràmetres eixida

El servei retorna un objecte *Resposta*. En la resposta es troba un objecte Vivenda que proporcionarà els mateixos atributs, amb les mateixes característiques, que els que es proporcionen en la Taula 10.

6.10 Obtenir totes les vivendes d'una promoció

Aquest servei permet obtenir una llista amb totes les vivendes amb les quals està composta una promoció de l'empresa a la qual pertany l'usuari.

Paràmetres entrada

El servei rep un objecte *Petició,* on en la ruta del servei s'ha d'especificar obligatòriament el camp *Id* de la promoció, les especificacions d'aquest camp són les mateixes que es troben en la Taula 8. A més, s'ha de proporcionar el camp *Authorization* en la capçalera com es mostra en la Taula 7.

Paràmetres eixida

El servei respon amb un objecte *Resposta*. En la resposta conté una llista amb els atributs següents de cada promoció.

Paràmetre	Descripció	Tipus	Màscara	Longitud màxima
Id	Identificador de la vivenda	Integer		11
Promocio	Identificador de la promoció	Integer		11
Planta	Altura de l'edifici on es troba ubicada la vivenda	Integer		10
Dormitoris	Nombre de dormitoris que conté la vivenda	Integer		50
Planol	Plànol de la vivenda	String		100
Superficie	Superfície total de la vivenda	Float		
Superficie_construida	Superfície construïda de la vivenda	Float		
Superficie_util	Superfície útil de la vivenda	Float		
Preu_base	Preu base de la vivenda	Double	Decimal (10,3)	13

Taula 12. Atributs Vivenda

6.11 Actualitzar vivenda

Aquest servei permet actualitzar una vivenda emmagatzemada en la plataforma Viviendea.

Paràmetres entrada

El servei rep un objecte de tipus *Vivenda*, on en la ruta del servei s'ha d'especificar obligatòriament el camp *Id* de la vivenda, les especificacions d'aquest camp són les mateixes que es troben en la Taula 11. Els paràmetres a enviar són els mateixos que els especificats en la Taula 10, a excepció que cap d'ells és obligatori. A més, s'ha de proporcionar el camp *Authorization* en la capçalera com es mostra en la Taula 7.

Paràmetres eixida

El servei retorna com a eixida un objecte *Resposta.* El valor de la resposta és el codi d'estat de l'operació. Aquests codis estan especificats en la Taula 1 del capítol 4.

6.12 Eliminar vivenda

Aquest servei permet eliminar una promoció emmagatzemada en la plataforma Viviendea.

Paràmetres entrada

El servei rep com entrada un objecte *Petició*, on en la ruta del servei s'ha d'especificar obligatòriament el camp *Id* de la vivenda, les especificacions d'aquest camp són les mateixes que es troben en la Taula 11. A més, s'ha de proporcionar el camp *Authorization* en la capçalera com es mostra en la Taula 7.

Paràmetres eixida

El servei retorna com a eixida un objecte *Resposta.* El valor de la resposta és el codi d'estat de l'operació. Aquests codis estan especificats en la Taula 1 del capítol 4.

7. Proves

Per a comprovar el correcte funcionament s'han dut a terme proves en fase de desenvolupament. Per a simular el funcionament que tindrà posteriorment al desplegar-la en el servidor, s'ha desplegat l'aplicació en una màquina virtual de Amazon Web Service [12]. Amazon Web Service és un servei de pagament que proveeix plataformes en el núvol a nivell global. Per a fer les sol·licituds s'ha utilitzant la plataforma Postman [13], aquesta és una plataforma lliure de col·laboració per al desenvolupament d'APIs on es poden realitzar sol·licituds REST per a testejar el correcte funcionament. D'aquesta manera s'observa com es realitzen les peticions, quin es el resultat esperat i quin és el resultat que realment s'obté.

Les proves s'han realitzat basant-se en els requisits funcionals establerts en el subapartat 3.3 i s'ha utilitzat un *token* vàlid d'un usuari administrador per a comprovar que funcionen totes les funcionalitats. A més, s'han realitzat proves per a observar si l'API controla de forma correcta els errors, per a evitar la repetició en totes les funcionalitats es farà una mostra en diferents funcionalitats.

7.1 Registrar usuari

Es realitza una petició amb el mètode POST a l'URI */registre*, enviant els atributs en format JSON següents:

```
{
        "Nom": "Roser",
        "Cognoms": "Fuster Bertomeu",
        "Dni": "12345678X",
        "Email": "prova@viviendea.com",
        "Password": "Prova1234",
        "Usuari": "provaRoser"
}
```

El resultat esperat és un missatge informant que l'usuari ha sigut registrat i el codi de resposta 200.

Com veiem a continuació efectivament es rep la resposta esperada:

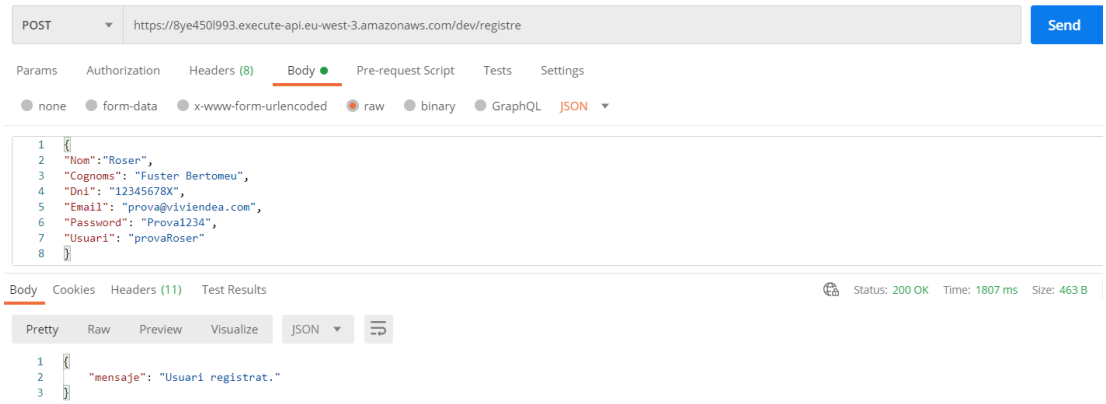

Figura 16: Resultat de registrar un usuari

Per a comprovar l'error amb codi de resposta 400, el qual es dóna quan es realitza una petició errònia, s'ha enviat una petició sense cos. I es verifica que es rep la resposta esperada com es mostra a continuació en la figura 17, on s'ha rebut un codi de resposta 400 junt amb el missatge "El contingut no pot estar buit".

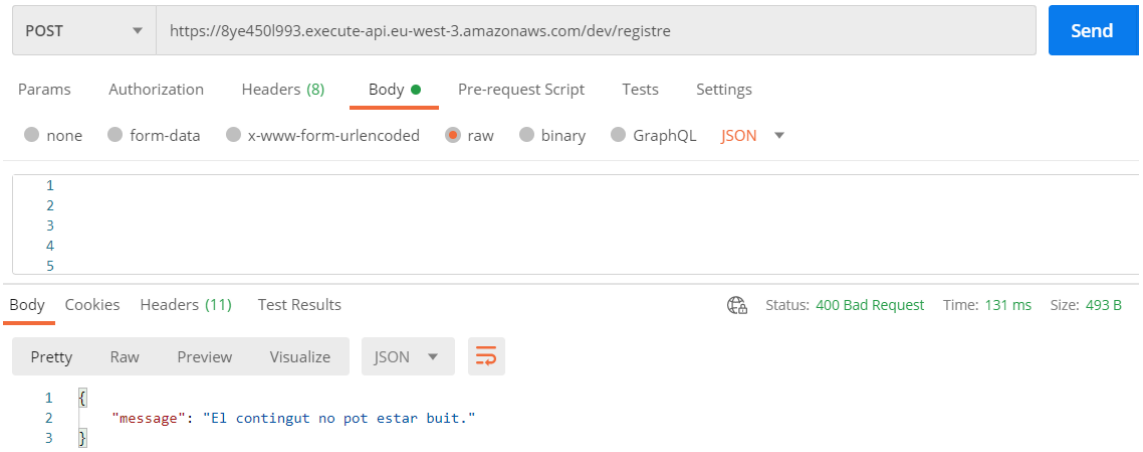

Figura 17: Resultat de petició amb arguments invàlids

7.2 Autenticació

Per a generar el *token* realitzem una petició amb el mètode POST a l'URI */token* amb l'usuari i la contrasenya en format JSON següent:

```
{
        "Email": "prova@viviendea.com",
        "Password": "Prova1234",
}
```

El resultat esperat és un *token* vàlid junt amb el missatge d'"Autenticació correcta" i el codi de resposta 200.

Després de realitzar la petició observem que obtenim els resultats que s'esperaven.

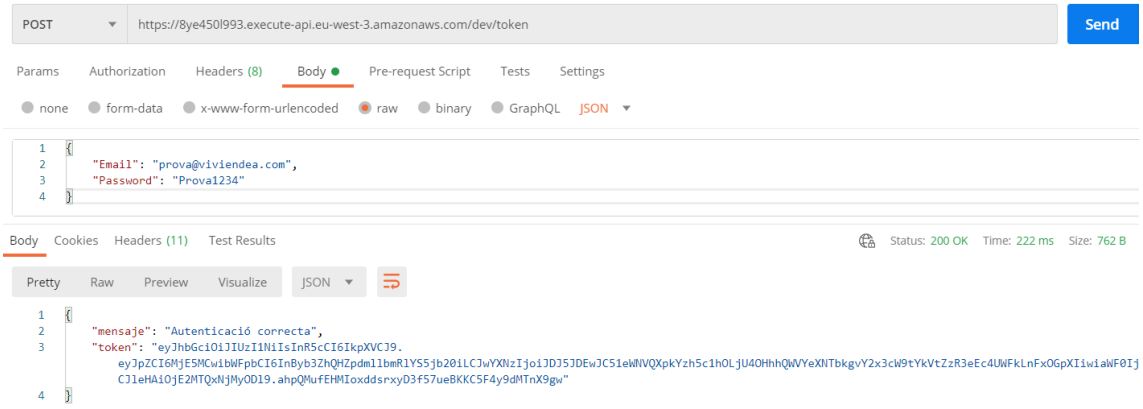

Figura 18: Resultat d'autenticació

En el cas d'introduir un usuari o una contrasenya incorrectes la resposta esperada és un codi de resposta 403 vàlid junt amb el missatge d'"Usuari o contrasenya incorrecta". Com es mostra a continuació:

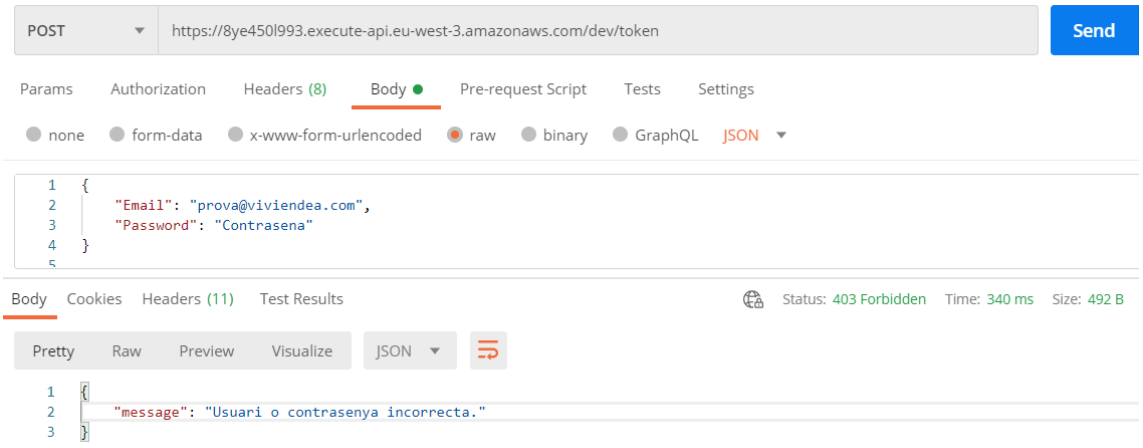

Figura 19: Resultat d'error d'autenticació

7.3 Crear Promoció

Per a crear una promoció, hem facilitat un token d'un usuari administrador, perquè ens permeta realitzar dita acció.

Una vegada realitzada la petició amb el mètode POST a l'URI */promocio*, si tots els paràmetres són vàlids, hauríem de rebre un missatge amb l'identificador de la nova promoció i un codi de resposta 200.

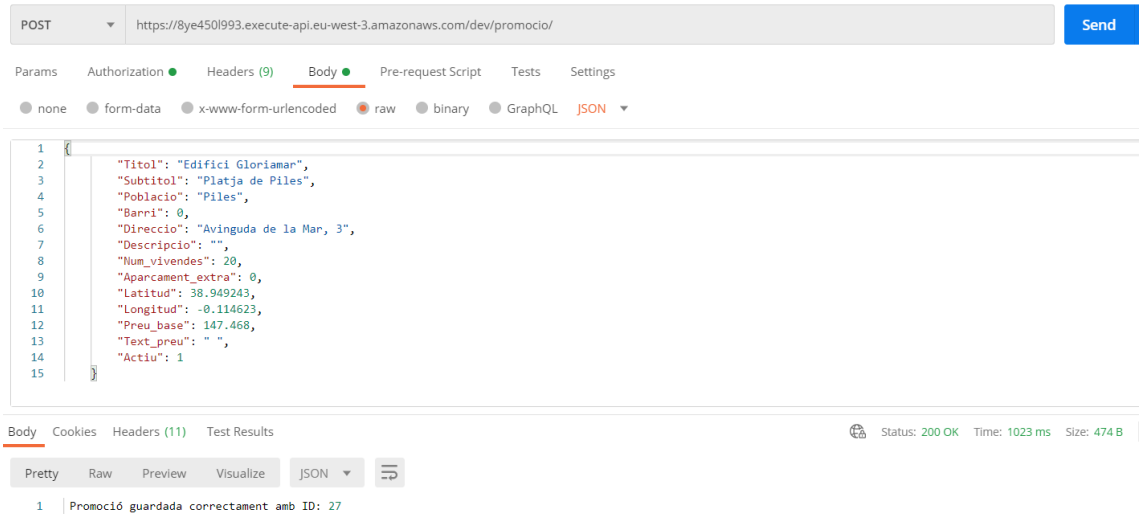

Figura 20: Resultat de crear una promoció

Veiem que efectivament s'ha creat una promoció a la qual s'ha assignat l'identificador 27. Si el *token* utilitzat és el creat en el subapartat anterior, el qual pertany a un usuari no administrador, la resposta obtinguda és la següent:

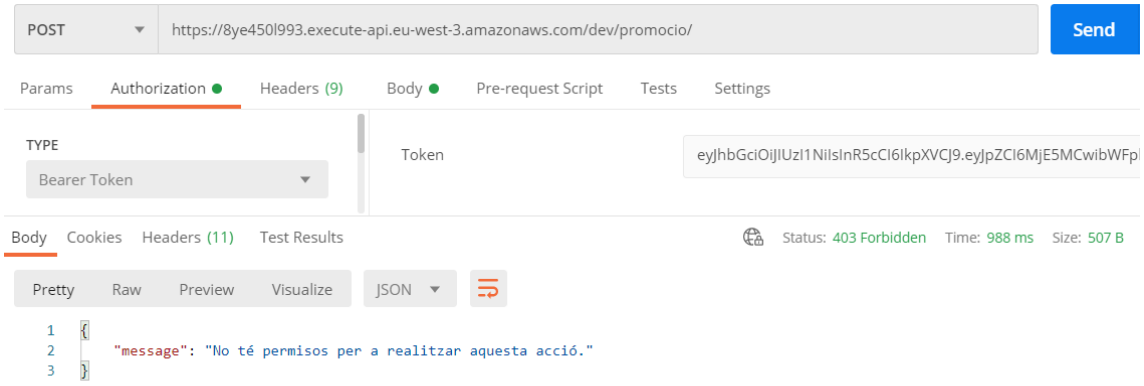

Figura 21: Resultat de permisos insuficients

Es rep un codi de resposta 403, indicant que l'usuari no té permisos per a realitzar dita operació.

7.4 Obtenir promoció

En aquesta petició provarem d'obtenir la promoció amb identificador 27, la qual s'ha creat en la prova anterior.

En realitzar la petició amb el mètode GET a l'URI */promoció/27* i proporcionar el *token* vàlid amb el que s'ha creat la promoció, esperem rebre els atributs d'aquesta promoció junt amb el codi d'estat 200.

A continuació, es mostra que sí que obtenim els resultats esperats.

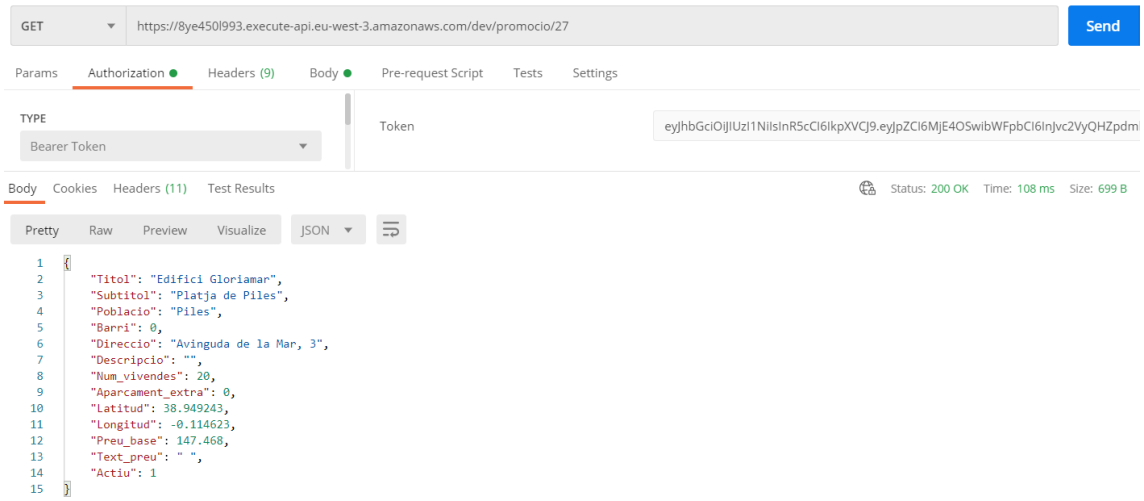

Figura 22: Resultat d'obtenir una promoció

7.5 Obtenir totes les promocions

Per a realitzar la petició on s'obtenen totes les promocions que pertanyen a l'usuari, el qual s'identifica quan adjunta el *token* en la capçalera de la petició.

Realitzarem una petició amb el mètode GET a l'URI */promoció*. S'assigna al camp d'autenticació el *token* vàlid utilitzat en els casos anteriors. El resultat hauria de ser una llista en format JSON amb totes les promocions assignades a l'usuari junt amb el codi de resposta 200.

A continuació podem observar el resultat de la petició realitzada a l'API REST:

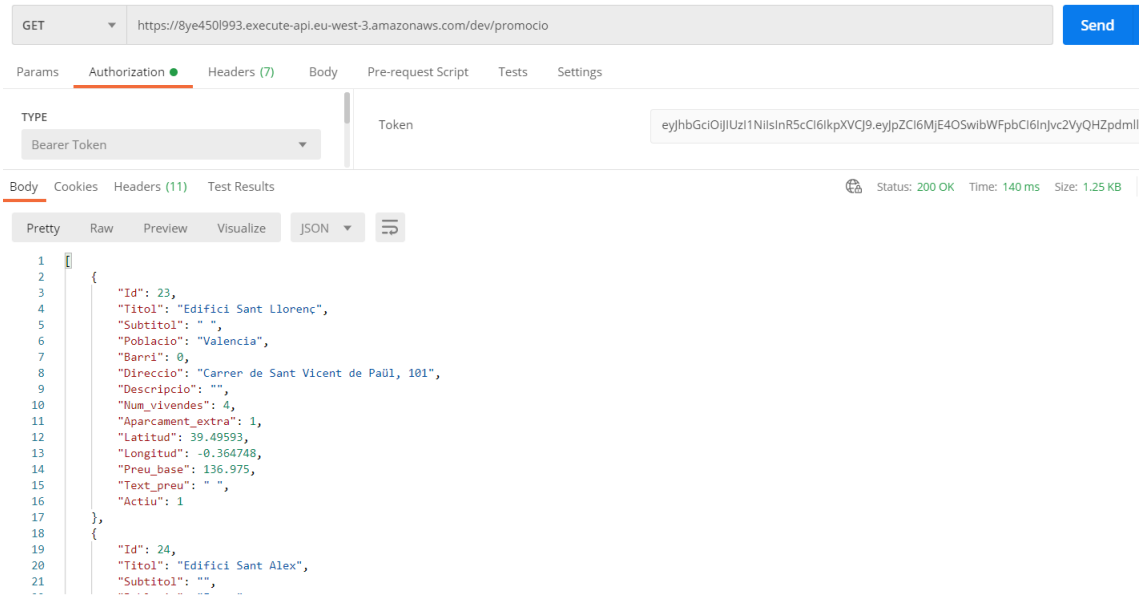

Figura 23: Resultat d'obtenir totes les promocions

7.6 Actualitzar promoció

Per a actualitzar la promoció amb identificador creada en el subapartat 7.3 realitzem una petició amb el mètode PUT a l'URI */promoció/27*. Al cos es troba la promoció amb tots els paràmetres, en aquest cas s'han modificat "Direccio", "Descripcio" i "Num_vivendes". S'introdueix a la capçalera *authorization* el *token* vàlid utilitzat en els anteriors casos.

El resultat esperat és el missatge "Promoció actualitzada" junt amb el codi d'estat 200, tal com es mostra a continuació en la Figura 24.

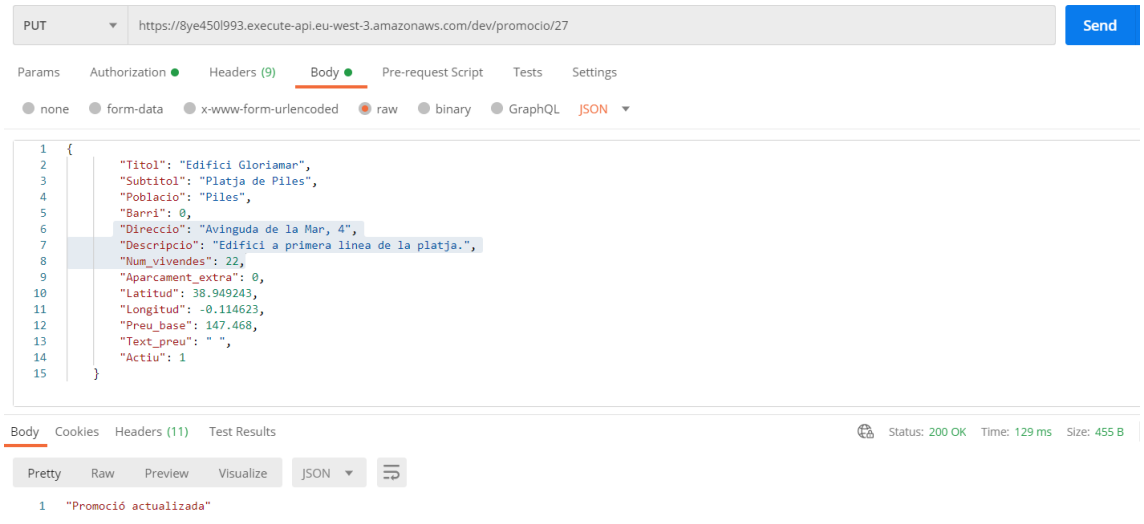

Figura 24: Resultat d'actualitzar una promoció

7.7 Eliminar promoció

Per a eliminar, en aquest cas, la promoció amb identificador 25, es realitza una petició amb el mètode DELETE a l'URI /promoció/25. S'ha assignat al camp d'autenticació el *token* vàlid utilitzat en els casos anteriors per poder realitzar l'acció.

Com es mostra a continuació, el resultat és el mateix que l'esperat. Obtenim el missatge "Promoció eliminada" junt amb el codi d'estat 200.

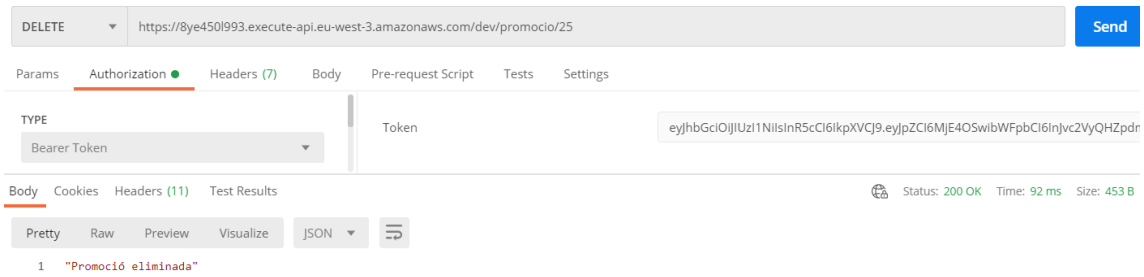

Figura 25: Resultat d'eliminar una promoció

7.8 Crear vivenda

Per a crear una vivenda, hem facilitat un token d'un usuari administrador, igual que ens els casos anteriors.

Una vegada realitzada la petició amb el mètode POST a l'URI */vivenda*, si tots els paràmetres són vàlids, hauríem de rebre un missatge amb l'identificador de la nova vivenda i un codi de resposta 200.

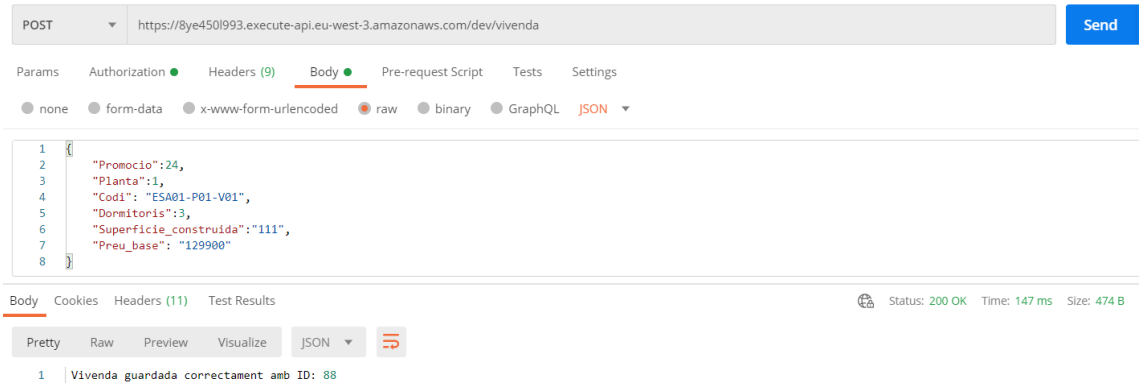

Figura 26: Resultat de crear una vivenda

Veiem que efectivament s'ha creat una vivenda a la que li s'ha assignat l'identificador 88.

7.9 Obtenir vivenda

En aquesta petició provarem d'obtenir la vivenda amb identificador 88, la qual s'ha creat en el subapartat anterior.

En realitzar la petició amb el mètode GET a l'URI */vivenda/88* i proporcionar el *token* vàlid amb el que s'ha creat la vivenda, esperem rebre els atributs d'aquesta promoció junt amb el codi d'estat 200.

A continuació, es mostra que sí que obtenim els resultats esperats.

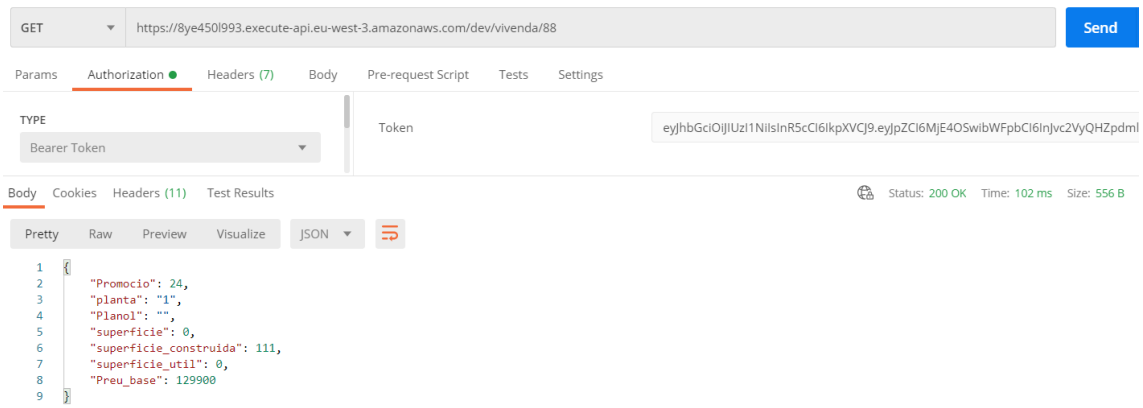

Figura 27: Resultat d'obtenir una vivenda

7.10 Obtenir totes les vivendes d'una promoció

Realitzarem una petició amb el mètode GET a l'URI vivenda*/promoció/23,* per a obtenir totes les vivendes de la promoció 23, la qual hem vist en el subapartat 7.5 que pertany a aquest usuari. S'ha assignat al camp d'autenticació el *token* vàlid utilitzat en els casos anteriors. El resultat hauria de ser una llista en format JSON amb totes les vivendes assignades a la promoció 23 junt amb el codi de resposta 200.

A continuació podem observar el resultat de la petició realitzada a l'API REST:

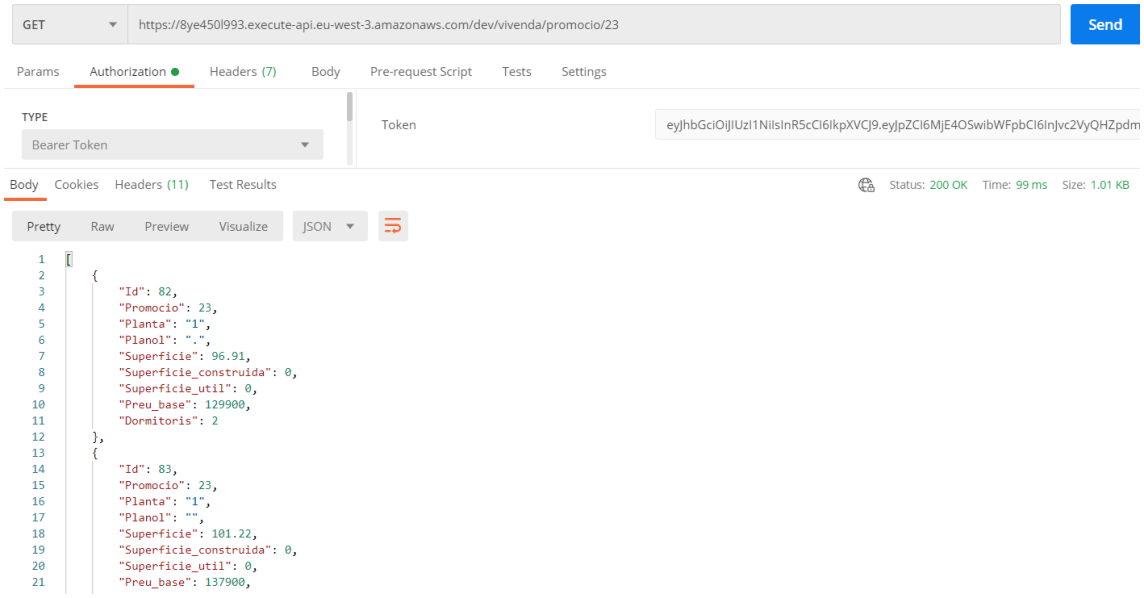

Figura 28: Resultat d'obtenir totes les vivendes d'una promoció

Si realitzem una petició per a obtenir totes les vivendes d'una promoció que no té cap assignada rebem un codi de resposta 404 indicant que no s'ha trobat ninguna vivenda per a aquesta promoció. Com es mostra en la figura 29.

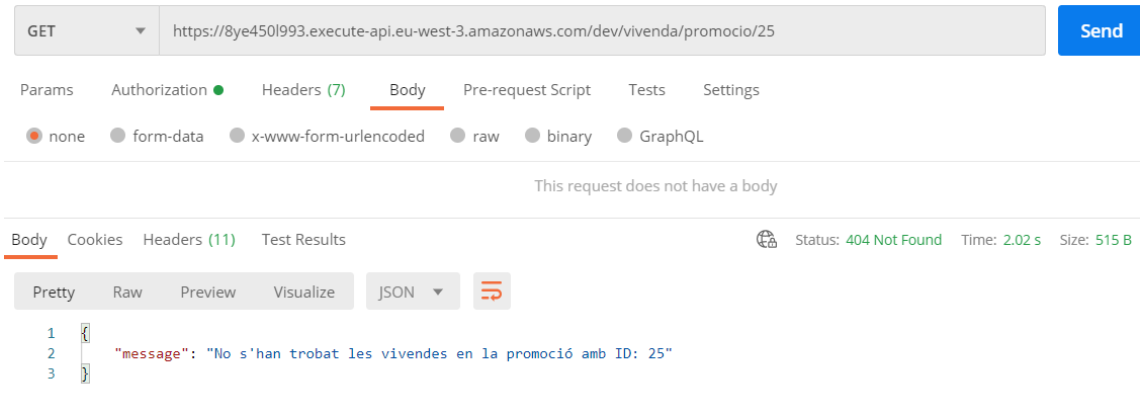

Figura 29: Resultat de vivendes no trobades.

7.11 Actualitzar vivenda

Per a actualitzar la promoció amb identificador creada en el subapartat 7.9 realitzem una petició amb el mètode PUT a l'URI */vivenda/88.* Al cos es troba la vivenda amb tots els paràmetres, en aquest cas s'ha afegit el paràmetre "Superficie". S'introdueix a la capçalera *authorization* el *token* vàlid utilitzat en els anteriors casos.

El resultat esperat és el missatge "Vivenda actualitzada" junt amb el codi d'estat 200, tal com es mostra a continuació en la Figura 30.

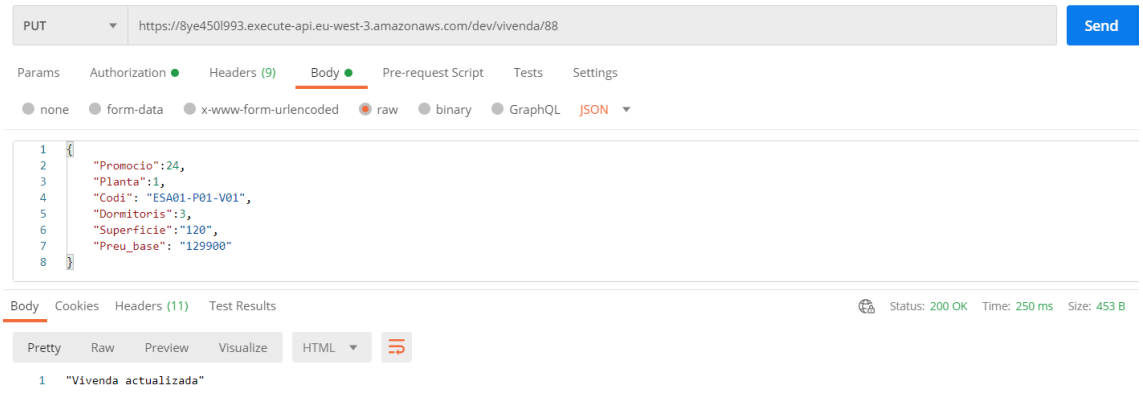

Figura 30: Resultat d'actualitzar una vivenda

7.12 Eliminar vivenda

Per a eliminar, en aquest cas, la vivenda amb identificador 87, es realitza una petició amb el mètode DELETE a l'URI */vivenda/87*. S'ha assignat al camp d'autenticació el *token* vàlid utilitzat en els casos anteriors per poder realitzar l'acció.

Com es mostra a continuació, el resultat és el mateix que l'esperat. Obtenim el missatge "Vivenda eliminada" junt amb el codi d'estat 200.

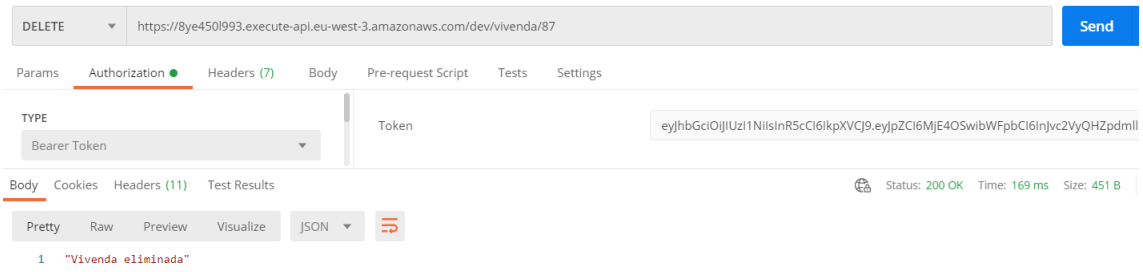

Figura 31: Resultat d'eliminar una vivenda

8. Conclusió

Després de realitzar el projecte s'ha comprovat que els objectius principals d'aquest han sigut complits amb èxit. A continuació, es destaquen els reptes aconseguits:

- S'ha implementat una API REST completament funcional on s'ha definit un sistema d'autenticació que comprova les credencials de l'usuari. A més, en base al rol assignat a l'usuari es permeten realitzar unes accions o unes altres.

- S'han complit els requisits funcionals de manera que és possible accedir als recursos des de les rutes URI especificades en aquest projecte.

- S'ha desenvolupat el codi de forma que permet el fàcil manteniment de l'aplicació i resoldre errors ràpidament en cas d'haver-los.

- En seguir els mateixos patrons de disseny permetrà implementar nous mètodes sense necessitat de canviar l'estructura ja desenvolupada, garantint així l'escalabilitat de l'aplicació.

- A nivell personal, s'ha aprés a treballar amb arquitectures de tres nivells i la importància que te a l'hora d'implementar una aplicació. A més, s'han consolidat els coneixements sobre JavaScript, NodeJS i MySQL que havia adquirit al llarg del grau universitari.

Treballs futurs

Com que aquest projecte s'ha desenvolupat com un producte dins d'una empresa es continuarà desenvolupant i millorant, de moment s'ha realitzat la estructura principal de l'API que servirà de base per a que en un futur s'implementen nous mètodes per a obtenir les característiques de les vivendes, s'ampliaran les funcions del registre per a indicar l'empresa a la qual pertany l'usuari i el rol que té assignat, i es crearan nous recursos perquè arquitectes, constructors i proveïdors puguen fer ús de l'API per a crear i modificar productes més específics del seu camp. Una vegada estiguen els nous mètodes implementats, l'equip informàtic està considerant la possibilitat de desenvolupar una Skill d'Alexa, l'assistent virtual d'Amazon, utilitzant els recursos de l'API.

9. Referències

[1] BBVA. (2020) "Tipos de APIs según los modelos de negocio" en *API_Market*, 27 de març. <https://bbvaopen4u.com/es/actualidad/tipos-de-apis-segun-los-modelos-de-negocio> [Consulta: 15 de març de 2020].

[2] COMBAUDON, S. (2016). *MySQL 5.7. Administración y optimización.* Barcelona: Ediciones ENI.

[3] Espanya. Llei Orgànica 15/1999, de 13 de desembre, de Protecció de Dades de Caràcter Personal, *BOE*, 14 de desembre de 1999, num. 298, p. 43088-43099.

[4] GUTIERREZ, E. (2009). JavaScript. Conceptos básicos y avanzados. Barcelona: Ediciones ENI.

[5] HAHN, E. (2016). *Express in Action. Writing, building, and testing Node.js applications.* Nova York: Manning Publications.

[6] *Portal oficial de Fotocasa* <https://www.fotocasa.es/es/> [Consulta: 15 de març de 2020].

[7] *Portal oficial de Habitaclia* <https://catala.habitaclia.com/> [Consulta: 15 de març l de 2020].

[8] *Portal oficial de Idealista* <https://www.idealista.com/> [Consulta: 15 de març de 2020].

[9] *Portal oficial de Yaencontre* <https://ca.yaencontre.com/> [Consulta: 15 de març de 2020].

[10] *Pàgina oficial de JSON WEB TOKENS* <https://jwt.io/> [Consulta: 20 de març de 2020].

[11] *Pàgina oficial de OAuth 2.0* <https://oauth.net/2/> [Consulta: 23 de març de 2020].

[12] *Plataforma Amazon Web Services* <https://aws.amazon.com/es/> [Consulta: 22 de maig de 2020].

[13] *Plataforma Postman* <https://www.postman.com> [Consulta: 21 de maig de 2020].

[14] *Polítiques de privacitat de Viviendea* <https://www.viviendea.com/politica-privacidad/> [Consulta: 15 de maig de 2020].

[15] *Portal API REST Tutorial* <https://restfulapi.net/> [Consulta: 20 de març de 2020].

[16] *Portal oficial de Same Gestion* <https://samegestion.com> [Consulta: 22 de març de 2020]

[17] *Portal oficial de* Viviendea <https://www.viviendea.com/> [Consulta: 22 de març de 2020].

[18] THE WORLD WIDE WEB CONSORTIUM (2001). *Rest.* <https://www.w3.org/2001/sw/wiki/REST> [Consulta: 15 de març de 2020].

[19] VIANA COLINO, M. (2012). *Minería de datos en portal inmobiliario*. Proyecto Final de Carrera. Leganés: Universidad Carlos III de Madrid, <https://core.ac.uk/download/pdf/30046689.pdf>.

[20] VIGOUROUX, C. (2019). *"Framework Node.js" en Aprender a desarrollar con JavaScript,* C. Vigouroux. Barcelona: Ediciones ENI.

[21] W3SCHOOLS.COM THE WORLD'S LARGEST WEB DEVELOPER SITE. *NodeJS*. <https://www.w3schools.com/nodejs/> [Consulta: 17 de març de 2020].

[22] WIKIPEDIA. JSDoc <https://en.wikipedia.org/wiki/JSDoc> [Consulta: 20 de març de 2020].