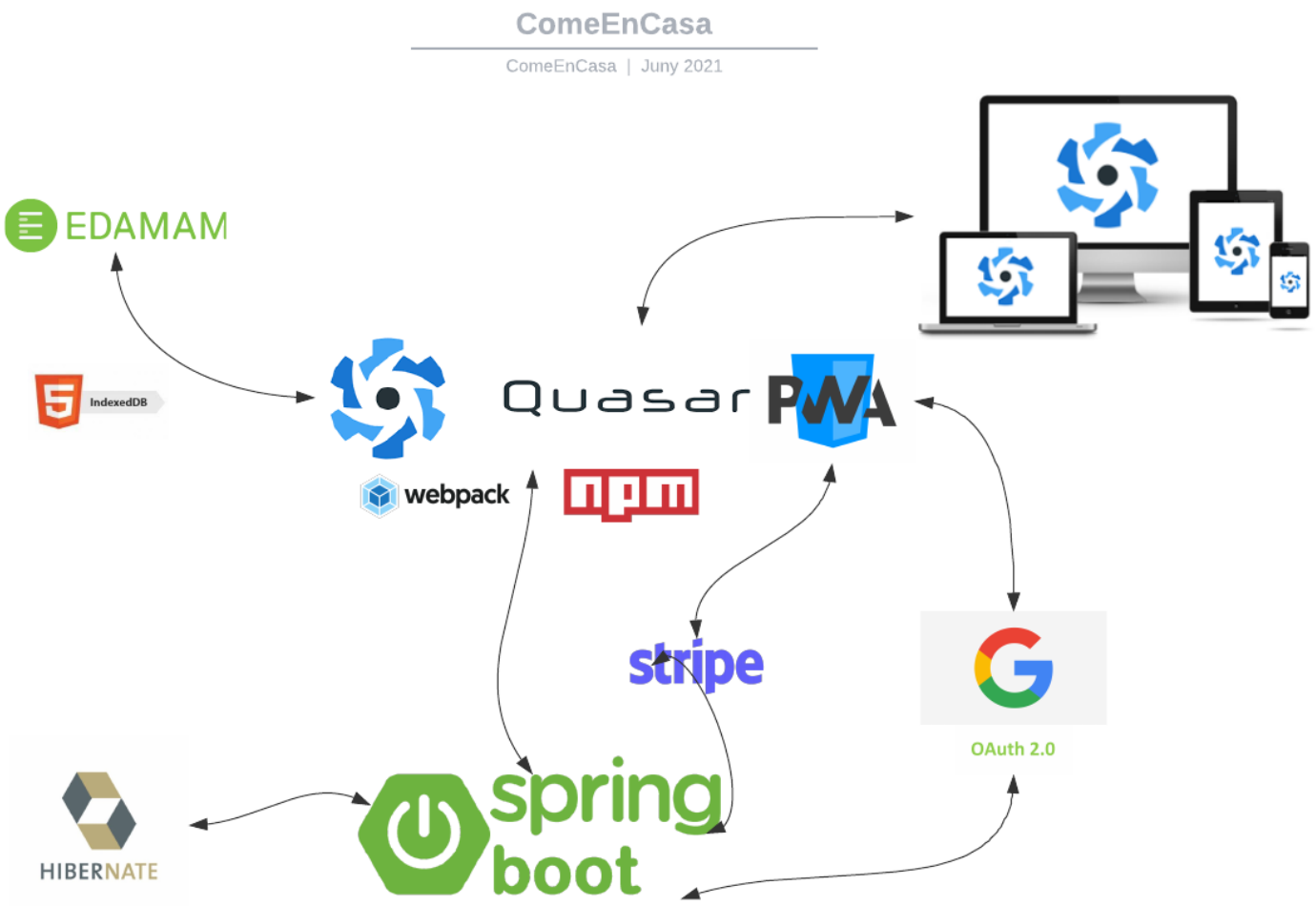
comeencasa



Bensaid Mokrane, krimo

Lopez, Jaume

06/2021

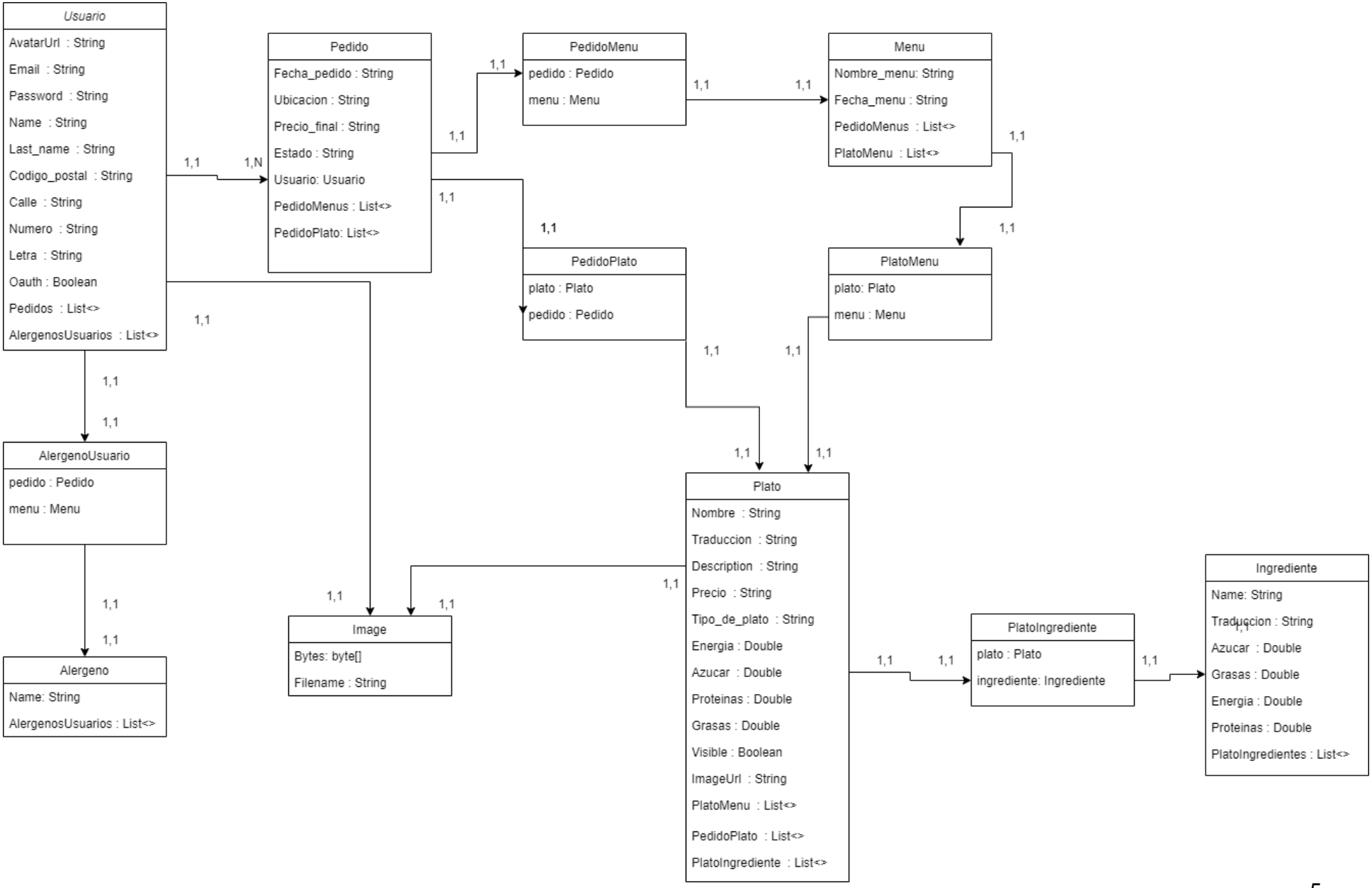
Introducció

L’objectiu d’aquesta pràctica és la creació i el disseny d’una aplicació per a poder comprovar si tenim els coneixements suficients per a poder crear la nostra propia página web. Amb aquests coneixements hem aprofitat per a crear una aplicación per a poder fer i gestionar les commandes d’un restaurant.

Aquesta idea, a diferencia d’altres que puguin existir, implementa carectarístiques que molt poques aplicacions contenen, a més d’estar pensada per a la implementació en un sol restaurant. Un dels detalls més importants en el que ens hem basat, ha sigut la implementació d’un càlcul d’informació nutricional, ja que trobam que avui en dia, cada vegada més gent es preocupa del que està menjant i quins són els ingredients del que están menjant. Per això considerem molt interessant el fet de que es pugui veure a primera vista tota la informació detallada, amb gràfics i colors tota la informació del plat.

El client que vulgui fer una comanda a través de la nostra aplicació, haurà d'enregistrar-se (ja sigui creant un nou compte o a través del seu compte de google) i a continuació afegir-se el plat o menú que vulgui al seu “Carrito”. Una vegada hagi escollit tots els productes que vol, haurà de pitjar el botó de “finalitzar pedido” i el redirigirá cap a la passarel·la pagar per tarjeta, on introduïrá les seves dades i fará el pagament. A continuació aquesta comanda es visible pel restaurant, i aquest prepararà les comandes.

A el diagrama que defineix les entitats i les relacions que tendrem en compte per realitzar aquest programa:


Usuario
AvatarUrl : String
Email : String
Password : String
Name : String
Last_name : String
Codigo_postal : String
Calle : String
Numero : String
Letra : String
Oauth : Boolean
Pedidos : List<>
AlergenosUsuarios : List<>
Pedido
Fecha_pedido : String
Ubicacion : String
Precio_final : String
Estado : String
Usuario: Usuario
PedidoMenus : List<>
PedidoPlato: List<>
PedidoMenu
pedido : Pedido
menu : Menu
Menu
Nombre_menu: String
Fecha_menu : String
PedidoMenus : List<>
PlatoMenu : List<>
PedidoPlato
plato : Plato
pedido : Pedido
PlatoMenu
plato: Plato
menu : Menu
AlergenoUsuario
pedido : Pedido
menu : Menu
Alergeno
Name: String
AlergenosUsuarios : List<>
Image
Bytes: byte[]
Filename : String
Plato
Nombre : String
Traduccion : String
Description : String
Precio : String
Tipo_de_plato : String
Energia : Double
Azucar : Double
Proteinas : Double
Grasas : Double
Visible : Boolean
ImageUrl : String
PlatoMenu : List<>
PedidoPlato : List<>
PlatoIngrediente : List<>
PlatoIngrediente
plato : Plato
ingrediente: Ingrediente
Ingrediente
Name: String
Traduccion : String
Azucar : Double
Grasas : Double
Energia : Double
Proteinas : Double
PlatoIngredientes : List<>
1,1
1,N

Anàlisi de requisits

StakeHolders:

- Usuaris
- Administrador
- Clients

Requisists funcionals:

- El client és por registrar.
- El client pot fer el login.
- El client pot afegir una imatge al seu perfil en el registre.
- El client pot canviar la seva imatge en el profile.
- El client pot canviar la seva direcció d'envio.
- El client pot canviar la contrasenya i el seu nom i cognom.
- El client pot definir els seus alergenos en profile.
- El client pot borrar el seu compte.
- El client por afegir plats al seu carrito.
- El client pot crear menus i afegirlos al seu carrito.
- El client pot borrar plats i menus del seu carrito.
- El client pot modificar un menu del carrito.
- El client pot pagar el carrito mitjançant stripe.
- El client pot veure tots els seus pedidos pero no modificar-los
- El client pot veure tots els platos existents pero no modifical-los
- L'administrador és pot enregistrar amb el seu email (Admin@comeencasa.com)
- L'administrador pot fer el login.
- L'administrador pot crear ingredients
- L'administrador pot esborrar ingredients
- L'administrador pot plats a partir d'una llista de ingredients
- L'administrador pot esborrar plats
- L'administrador pot modificar plats i la seva llista d'ingredients
- L'administrador pot fer visible o que deixi de ser visible uns plats per a la resta d'usuaris.
- Tots els **StakeHolders** poden veure els plats.
- També poden veure l'ubicació d'on és troba el local i d'on és troben ells mateixos.

Requisits no funcionals:

- Permetre la visualització responsive adaptada a tots els dispositius.
- Poder instal·lar l'aplicació a diferents dispositius(PWA)
- Permetre el registre a l'aplicació mitjançant google oauth
- Proporcionar a l'usuari notificacions d'error quant les consultes no poden ser realitzades o aquestes son errònees. També informar de quant han estat satisfactories.
- Implementar leaflet com a API de mapa i proporcionar a l'usuari la seva posició de gps i informar-li de la posiscio del comerç
- Utilitzar com a pasarela de pagament STRIPE

Temporalització de totes les tasques

La nostra planificació no ha estat un dels punts forts del nostre projecte. Per motius laborals i personals no vàrem poder iniciar el projecte fins uns dies abans d'acabar les pràctiques amb la conseqüencia de no tenir tant de temps per pensar com distribuïr el temps.
El que vàrem definir varen ser els requisits que voliem que tengués la nostra aplicació i ens vàrem proposar aconseguir el màxim possible dels nostres objetius en el marc temporal que disposavem, marcant sempre el marge dels darrers dies per acabar de perfilar tant la documentació com la neteja de codi. El que ens ha anat passant, es que així com anavem avançant han anat sortint més requisits, imprevistos i detalls que havíem obviat que han relantitzat el desenvolupament de l'aplicació.

Compliment de la temporalització

A l'hora d'assolir tots els requisits que varem definir en un principi, hi ha hagut 2 objectius que no ens ha donat temps a realitzar degut als entrebancs que hem tengut durant el desenvolupament.

D'un principi volíem que l'usuari a part de poder gaudir dels serveis bàsics, pogés tenir una secció "premium" on poder controlar tot el seu procés alimentari i tenir una gestió molt més completa de la seva informació nutricional.

Una altra objectiu que degut a la falta de temps no hem pogut acabar de realitzar, ha esta la part de filtració dels plats segons els "Alergenos", ja que l'usuari si que pot definir els seus alergens, pero no si el plat te "X" alergeno o no.

I un petit contratemps que varem tenir, va ser a l'hora de demanar clau api per a la base de dades dels aliments, ja que en un principi voliem utilitzar aquesta *https://www.calorieking.com/us/es/developers/food-api/*. Pero per mala sort no ens varen donar una resposta, i varem tenir que

utilitzar una de les Apis que varem utilitzar en classe l'any passat.

Principals problemes a l'hora de desenvolupar el sistema

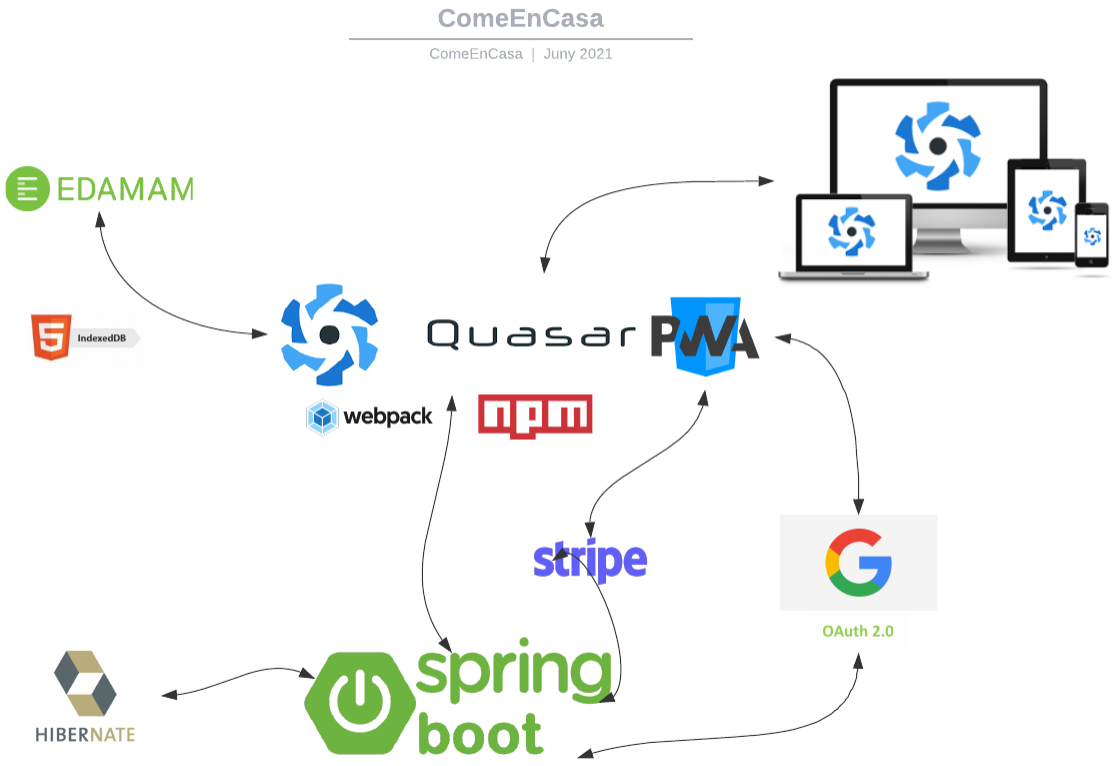
El problema mes gran a l'hora de desenvolupar un sistema complet com el que hem hagut de dissenyar es la falta d'experiència en realitzar projectes integrals. La planificació es imprescindible i tal vegada ens n'ha faltat una mica.

Un dels altres inconvenients ha estat que no hem pogut realitzar pràcticament reunions presencials i no hem pogut debatre en condicions les indees i inquietuds de cada un de nosaltres. Per motius personals, laborals i altres esdevinguts pel virus COVID19 ha estat molt difícil aquesta tasca. A més que treballar en grup implica interactuar i conèixer a l'altre persona per poder arribar amb més facilitat a objectius comuns i fer més senzilla la aquesta tasca, i això ens ha estat pràcticament impossible.

Descripció i disseny de l'arquitectura

Per aquesta aplicació hem emprat:

1. Quasar com a framework de VUE(javascript) per elaborar el frontend. Tot això implica utilitzar webpack com a empaquetador i les seves advantatjes.
2. Una api a la part de front que recupera infoarmació nutricional per a guardar aquesta informació a indexedDB abans de guardar-la a la bd local del servidor.
3. SpringBoot com a framework de java EE a la part d'entorn servidor, amb hibernate per a gestio i creació de les taules a la bdd MYQL
4. Stripe com a pasarela de pagament
5. Tot això dockeritzat en contenidors per tal de fer més portable tot el desplegament.



Backend

A la part de desenvolupament de l'entorn servidor utilitzam SpringBoot, que és un instrument de mapatge objecte-relacional (ORM) per a Java, que permet el mapatge d'atributs en una base de dades clàssica, i el model d'objectes d'un aplicació per mitjà d'arxius declaratius o anotacions en els **beans** de les entitats que permeten entaular aquestes interrelacions. A més agilitza la interacció entre l'aplicació i la nostra base de dades SQL, d'una manera d'optimitzar el nostre flux de treball evitant caure en codi cíclic.

Aquestes dades s'emmagatzemen en una base de dades **MYSQL**, i cada objecte té diferents classes. Per a cada objecte hem de produir una classe que almenys possibiliti generar, inserir, remenar, consultar o canviar la informació continguda en els seus atributs.

Una cosa molt bona que te SpringBoot és la compatibilitat que te amb JPA, cosa que facilita l'accés a les dades mitjançant les seves eines de consulta.

Les anotacions son una sèrie de eines que ens proporciona el nostre framework de java que ens permet realizar accions sense ser nosaltres els que haguem de pensar la logica d'aquestes accions. SpringBoot s'encarrega de gestionar això aplicant el principi d'inversió de control.

Les anotacions també s'apliquen a la gestió de les entitats on empram **HIBERNATE** per al seu maneig, així assegurant l'integritat de les dades en tot moment. Hi ha diferents anotacions, nosaltres enumarem les que hem utilitzat per a poder crear l'estructura que necesitavem:

-Primer afegir un **@Entity** per definir al programa que aquesta classe és una entitat

-Com sempre pasa, una entitat té sempre una id, llavors podem definir l'atribut de id amb **@Id**, i a més si volem que s'autoincrementi podem afegir
@GeneratedValue(strategy = GenerationType.IDENTITY)

-Les relacions es defineixen amb les anotacions com **@ManyToOne @OneToMany i @ManytoMany,** encara que aquesta darrera no la utilitzam ja que cream taules intermitges amb un **@OneToMany** desde cada costat de les entitats, i dins la taula intermitja un **@ManytoOne** per aixi poder tenir dades emmagatzamades en aquestes taules.

-Un dels atributs de les anotacions anteriors és **@OnDelete,** que serveix per definir de quina manera s'esborren les dades.

- Resolució del programa

A l'hora d'intentar solucionar aquest programa i fer-lo funcional haurem d'emprar una arquitectura de Vista-Controlador,on més endavant entrarem més en detalls amb la part de client, li passarem respostes **JSON (amb la llibreria de Gson)** que més tard la part de client gestionarà i farà el canvis necessaris en el frontEnd. Llavors, en aquesta part de **Servidor** només ens encarregam de la part de recerca o de filtració de dades i de la part de connexió amb la base de dades, o els arxius de configuració de la base de dades o dels Servlets i fins i tot els filtres de la nostra aplicació que segons les dades que estem passant o la situació en la que ens trobem faran una cosa o una altra.

Clarament, per a completar aquesta arquitectura, hem dividit el programa en diferents directoris on en cada un d'ells trobarem subgrups d'arxius que tenen funcions similars pero actuen per cada página que és veurà desde la part de client. Els subgrups que existeixen són els *Controllers*,els *Entities*,els *Interceptors*,els *Repositoris* i els arxius *Service* que explicarem a continuació

- Configuració

La configuració del nostre servidor la definirem a un fitxer anomenat Config.java. Aquest fitxer amb l'anotació **@Configuration** l'emprarem per determinar la configuració de CORS per a només acceptar peticions des de les direccions que nosaltres ens interessi i implementarem els interceptors, on definirem les rutes que volem que tenguin efecte.

- Interceptors

Mitjançant els interceptors podem controlar l'accés a les consultes del nostre servidor. Una vegada es fa alguna petició al servidor, aquests s'executen i mitjançant les regles que hem definit retornarà un **TRUE o FALSE**. D'aquesta manera controlarem l'accés de les peticions. En el nostre projecte hem definit els diferents interceptors.

Token Interceptor

S'encarrega de capturar la capçalera de la petició i comprova si el **token** que es pasa a la capçalera es vàlid al servidor, tant per la part de oauth com pels tokens generats pel propi servidor.

Admin Interceptor

S'encarrega de capturar la capçalera de la petició i torna a comprovar si aquest token es vàlid, a més comprova que el token correspon a l'usuari administrador del sistema. D'aquesta manera només feim possible la petició als recursos de la part de admin per aquest usuari.

- Repositorys

Avui en dia, pràcticament totes les aplicacions requereixen obtenir les dades d'una base de dades, en aquest cas, una base de dades MYSQL. Per això existeix el patró arquitectònic CrudRepository que ens ofereix Spring, el qual ens permet separar la lógica d'accés a les dades, de tal forma que queda encapsulat tota la lògica d'accés de dades de la resta de la nostra aplicació.

CrudRepository és la proposta estàndard que ofereix Java per implementar un Framework Object Relational Mapping (ORM), que permet interactuar amb la base de dades per mitjà d'objectes, d'aquesta manera, és l'encarregat de convertir els objectes Java en instruccions per al Gestor de Base de dades.

Dins la carpeta de "Repos" trobarem un arxiu per cada taula del model que existeix, ja que per cada taula tendrem comandes totalment diferents ja que depenen del nom de la taula i del nom dels seus atributs.

En cadascun dels arxius, trobarem les funcions necessàries per poder crear i inserir un objecte nou a la taula, borrar l'objecte de la taula, modificar un objecte de la taula i fins i tot per poder cercar tots els valors de la taula segons totes les possibles condicions que podem arribar a posar.

- Model

Els models representen els objectes del problema a resoldre. El seu nucli i el més important dels arxius model solen ser els atributs i els getters i setters d'aquests, que representen la informació important de cada una de les entitats. Solen tenir a més sobre escrits els mètodes equals, hashCode i toString.

A més una altra cosa molt importat i que hem mencionat al principi d'aquesta documentació és la implementació de les @anotacions, ja que ens donen molta facilitat a l'hora de crear les taules de la base de dades i de la relació entre totes les taules. Cal destacar

que s’ha implementat una capa DTO(data transfer object) per a poder pasar les dades des del servidor a la vista de manera que no doni problemes de rendarització i controlar la informació obtinguda de la consulta.

- Services

Els “services” o serveis són aquells intermediaris que representen la lògica del negoci, ja que aquests criden als Daos per rebre informació i poder pasar-la als Controllers per així poder mantenir encapsulada tota la informació. És a dir, que la funció dels serveis és de intermediari entre els Repositorys i els Controllers i per poder mantenir una estructura de diferentes capes, ja que d’aquesta manera podem canviar algún aspecte del Repository i la gestió del Controller seguirà sent el mateix.

- Controllers

Quan és crida a una petició HTTP o HTTPS amb una URL, estem cridant a un *Controller* que té assignat aquesta URL ja sigui en java o javascript o qualsevol llenguatge, pero els nostres controllers de la part de **Servidor** tenen la funció de gestionar dades respecte a la base de dades. Els nostres **Controllers** son arxius on tenim definits URL’s que és troben escoltant i esperant la seva cridada i una vegada instànciat i segons el “method” que li estigui arribant (“GET”, “POST”, “PUT” or “DELETE”), realitzarà unes accions o altres i retornará una respostaja siguien en **Json** per a retornar informació important o per retornar respostes d’errors.

Els method que passem als controllers depenen de 2 coses:

-El method será “GET” és sol utilitzar ens els casos on volem rebre informació de la base de dades.

-El method será “POST” en el moment que li esteim passant al Controller una petició amb l’objectiu d’inserir dades a la base de dades.

-El method será “DELETE” ens servirá com diu la propia paraula, per BORRAR informació que tenim en la base de dades.

Ara explicarem els diferents Controller que tenim dins el nostre programa.

Pedido Controller

Aquest controller s'encarrega de totes les peticions relacionades amb la taula de “Pedido” és a dir, tant un “GetAll, como Saves, Deletes...”. Lògicament un Usuari té un **pedido**, i aquest **pedido** pot tenir plats i menus, llavors en aquest arxiu tindrem en compte en tot moment aquestes dades.
Ara explicarem les funcions del Controller:

- **@GetMapping("/pedidos"):** el nostre objectiu és demanar a la base de dades tots els pedidos enregistrats segons l’usuari que ens ho demana(excepte el pedido que és trobi en estat **“Pendiente”** ja que això significaria que encará no ho ha pagat i per ara ens considera com a “**Carrito**”). A continuació una llista del resultat en forma de Json.

- **@PostMapping("/getPrecioPedido"):** Aquest mapping ens servirá per a poder retornar el preu final de un pedido. Per a poder fer-ho primer hem de llegir el **requestBody** que ens passa el client, que en aquest cas és la Id del pedido, i amb la funció *findPedidoById* del **PedidoService,** ens retorna el pedido que té aquesta id i ja podrem agafar el seu preu total i retornar-lo cap a la part de client.

- **@PutMapping("/pedidos"):** Aquest mapping és per a crear un nou Pedido buit, on només afegirem la informació que tenim al moment, en aquest cas serien l’usuari que és troba connectat, el preu final que en aquest cas sempre és 0, l’ubicació d’entrega(que **l’usuari** ho **defineix** en la part de client de “**Profile**”), i l’estat del Pedido, que en aquest cas és “Pendiente” ja que encara no ha afegit cap producte i no ho ha pagat.

- **@GetMapping("getOpenPedido"):** Aquesta funció ens servirà per a retornar a l'usuari conectat el seu pedido que és trobi amb l'atribut *estat = Pendiente,* és a dir, per a poder trobar el Carrito de l'usuari conectat on trobarem tots el productes i menús que ell ha afegit .

- **@PostMapping("/addPlatoPedido"):** Com diu el nom del controller, aquest ens serveix per a poder afegir un plato al nostre Carrito. per això, primer haurem d'agafar el pedido de l'usuari que és trobi amb l'atribut *estat = Pendiente,* a continuació **sumarem** al **preu final** del **carrito** el **preu del plato** a afegir i ho guardam. Després crearem una nova instància de la taula intermitja entre pedido i plato on guardarem el Pedido i el Plato. Tot l'anterior ocurrirá en cas que l'usuari tengui un "Carrito" creat, en cas contrari crearem primer el Carrito i després farem tot l'anterior.

- **@PostMapping("/añadirMenu"):** Aquest controller ens serveix per a poder afegir un Menú al carrito, i segueix pràcticament la lógica de l'apartat anterior, és a dir, primer cercam el Carrito de l'usuari, on a continuació li canviam el preu final, cream una instancia de la taula intermitja on guardarem el menu i el pedido. I una altra vegada comprovam si l'usuari no te Carrito encara, pues ho cream i guardam totes les dades.

- **@PostMapping("/guardarMenu"):** És clar, mentres que l'usuari no hagi pagat i confirmat el seu pedido, podrá modificar el carrito tot el que vol, ja sigui afegint o borrant platos o afegint, borrant i **modificant** menus. En aquest cas tenim en compte la modificació del menu, on llegim del **payload** la nova llista de plats i la id del menu. A continuació cercam el Menu que tengui aquesta id, i del menu, tindrem que cercar totes les instàncies la taula intermitja entre menu i "platos" per a poder modificar les referencies pers als nous plats a afegir. Una vegada ja ho hem modificat, tindrem que guardar totes les dades amb la funció **save** dels repositoris.

- **@DeleteMapping("/deletePlatoPedido"):** Com ja hem dit anteriorment, l'usuari pot modificar el seu Carrito, en aquest controller tindrem en compte la part de borrar un plat del Carrito.

Per a comernçar rebem la **id** del **plato** per **payload,** cercam el plato per id, i restam el preu del plato al preu total de Carrito. Una vegada fet això l'únic que hem de fer és eliminar la relació de la taula intermitja que és troba entre el Plato i el Pedido.

- **@PostMapping("/setPedidoPagado"):** Aquest controller ens serveix per canviar l'estat del Carrito, de "Pendiente" a "Pagado", i així deixaria de ser el carrito i pasa a ser un pedido. Aquest controller només és crida quant el pagament del Carrito ha sigut correcte.

Plato Controller

Aquest arxiu s'encarrega dels mappeigos relacionats amb els plats que tenim a la base de dades, i les seves relacions entre el menú i la comanda. A continuació entraré més en detall amb cada mapeig:

- **@GetMapping("/platos"):** Aquest controller és molt facil, ja que ens serveix per a poder retornar a la part de client una llista amb tots els plats que tenim guardats en la nostra base de dades en forma de JSON.

- **@PostMapping("/setVisibility"):** Una cosa molt important que no hem xerrat, és que com a restaurant, a vegada ens interesa no borrar els nostres plats, ja sigui perque hem de fer modificacions o perque no és troba disponible en aquest moment pero pot ser que mes tard si. Llavors, hem creat la opció de que sigui visible o no als clients. En aquest cas, aquesta funció ens serveix per canviar d'estat a un producte en concret.

- **@PostMapping("/getPlatosMenu"):** Aquesta funció en serveix per a poder retornar en forma JSON una llista amb tots els plats d'ún menu en concret, menu que cercarem segons la **id** que ens passin per **payload.**

- **@PostMapping("/getPlatoById"):** Una funció molt facil i com diu el nom, **retorna** un **Plato** segons la **id** que ens passen per és **@RequestBody.**

- **@updateImagePlato:** Aquesta funció reb una imatge i una id de plat. El PlatoService s'encarrega de processar aquesta imatge i s'afageig al plat la nova adreça url de la imatge desada prèviament a la bbdd.

- **@PostMapping("/crearPlato"):** Com diu el nom, aquest mapping ens serveix per a poder crear nous plats. Lo primer de tot, és crear una instancia de "Plato" i afegir tots els atributs que ens passen per paràmetre desde el client. Una vegada afegits, guardam el **plato,** i de la llista de ingredients que ens passen desde client, crear les taules intermitges necesaries (PlatoIngrediente) on guardarem la instància de cada Ingredient, i el Plato el cual forma.

- **@PostMapping("/guardarPlato"):** És clar que com a super Usuaris que som nosaltres, ens interesa que els propietaris dels restaurants puguin modificar els plats que venen en la pagina web. Com que potser que ens canvin la recepta, hem de tenir unes cuantes coses en compte. Si en la nova llista de ingredients hi ha ingredients que no estaven guardats, hem de eliminar els vells que no s'usen i afegir a la llista els nous. Per això també haurem d'esborrar referencies en les taules intermitjes i crear-ne de noves per als nous ingredients. A més d'això també haurem de modificar al plato els altres atributs que ens passen per parametre. Una vegada fet tot aixó haurem de fer un "save" de totes les intacies de classes per desar-les en la base de dades.

- **@PostMapping("/getIngredientesByPlato"):** Aquest mappeig és molt facil ja que l'objectiu és retornar una llista d'ingredients segons un Plato. Per aixó, ens passen per **@RequestBody** la id del plato, i amb el platoService i la funció *findPlatoById* ens retornarà el plato, i com dins plato

tenim una llista de “PlatoIngredientes”, només la hem de recorré i afegir tots els ingredientes en una llista per després retornar-la en JSON.

- **@DeleteMapping("/deletePlato"):** Logicament, al restaurant també l’interesa que és borrin “platos” de manera definitiva, ja sigui perque no els interesa o per qualsevol motiu, llavors dins “PlatoService” tenim una funció que borrará el objecte “plato” que li passem per parametre.

Menu Controller

Aquest arxiu s'encarrega dels mappeigos relacionats amb els menus que tenim a la base de dades, i les seves relacions entre el plati la comanda. A continuació entraré més en detall amb cada mapeig:

- **@DeleteMapping("/deleteMenuPedido"):** A l’hora d’es borrar un menu hem de tenir en compte que un menu és una agrupació de 4 plats que és troben relacionats a aquest, llavors quan volem esborrar un menu, també haurem de borrar les seves relacions. Primer llegirem del **payload** la **id** del **menu** que ens passen. A més, també hem de tenir en compte que l’usuari només pot borrar menús del “carrito” i no d’una comanda que ja s’ha fet. Llavors, cercarem el **pedido** que **d’estat** sigui “**Pendiente**”, i agafarem la seva llista de *PedidoMenu* i esborrarem les intermitjes on **idmenu**==*PedidoMenu.getMenu().getId()*
  Per finalitzar farem un save de **pedido** i retornarem una resposta cap al client confirmant que s’ha afegit el plat correctament.

- **@PostMapping("/getPlatosByMenuId"):** Aquest controller és molt facil, l’objectiu és tornar a l’usuari la llista de plats d’un menú segons la id d’aquest menú. Llavors lo que hem de fer és llegir de **payload** la id del menu que ens passen i fer una llista de tots els plats que té gracies al seu atribut relacionat **PlatoMenu,** i retornam aquesta llista en format JSON

Image Controller

Dins aquest controller trobarem 2 mapeigos, un sobre l'usuari i l'altre sobre els plats, ja que tots 2 tenen l'opció d'afegir imatges. Llavors lo que fan és llegir la petició que els arriba i segons el "**name**" que ens arriba retorna un arrayDeBytes per resposta cap al client. A més de la funció que s'encarrega de codificar la imatge en base64 per a la seva posterior utilització.

User Controller

Aquest arxiu de controllers en servirà per poder gestionar, modificar i esborrar informació en la base de dades relacionada amb els usuaris que és troben enregistrats dins la nostra aplicació, ja sigui per canviar la imatge d'un usuari. la contraseña o fins i tot els alergenos que és troben relacionats amb aquest. A continuació explicarem cada un dels controller que formen aquest arxiu:

- **@DeleteMapping("/deleteUser"):** Aquest controller és molt facil ja que el seu objectiu és esborrar el compte del usuari que esta fent aquesta petició. Per això cercarem segons l'"**Authorization**" que ens estan passant, llegirem el token, i del token agafarem l'**email** per a poder agafar a l'usuari i poder esborrarlo, i retornar una resposta afirmativa cap a l'usuari.

- **@GetMapping("/userHaveDirection"):** Aquesta funció només ens serveix per comprovar si l'usuari té totes les dades de la direcció amb valor, per això farem com el mapping anterior, on cerarem l'usuari segons **l'Authoritzation** i d'alli mirarem si els atributs de direcció son nulls o no.

- **@GetMapping("/getUserDetails"):** Dins aquest controller lo que feim és retornar cap al client, tota la informació (no

important) de l’usuari, com seria els valor de direcció, el name i el email en un **mapa** transformat a **JSON.**

- **@PostMapping("/updateUser"):** Aquest controller és molt facil, ja que ens serveix per canviar a l’usuari l’informació del nom i cognom del seu compte. Per poder fer-ho primer agafarem l’usuari del token, i llegirem el **payload** on és trobará la nova informació a sustituir. Llavors agafarem el usuari i farem un “set” dels nous atributs, i per finalitzar farem un save del usuari.

- **@PostMapping("/updateUserDirection"):** Aquest controller és molt facil, ja que ens serveix per canviar a l’usuari l’informació de l’adreça del client del seu compte. Per poder fer-ho primer agafarem l’usuari del token, i llegirem el **payload** on és trobará la nova informació a sustituir. Llavors agafarem el usuari i farem un “set” dels nous atributs, i per finalitzar farem un save del usuari.

- **@PostMapping("/updateImageUser"):** Dins aquesta funció trobarem un mappeig que ens servirá que modificar la imatge de l’usuari. Per això agafarem a l’usuari segons el Token que ens estan passant, agafarem la foto del **payload** i farem un “set” de la url de la imatge, asi cada vegada és cridi aquesta url, seria com fer ,una petició que rebrà el controller de Imatges que retornará la imatge.

- **@PostMapping("/changePassword"):** Aques controller ens serveix per a poder canviar la contrasenya. Lògicament a l’hora de canviar la contrasenya, sempre hem de tenir una en compte, que l’anterior contrasenyaque hagi escrit l’usuari sigui igual a la que esta guardada. Si és així, l’usuari podrá canviar la contrasenya, mentres que en cas contrari no podrá ya que son diferents i això provoca un problema de seguretat.

- **@PostMapping("/getNameByEmail"):** Dins aquesta funció el nostre objectiu és retornar l'avatar d'un Usuari segons el email que ens passen per payload, per això tenim una funció del **Service** que ja ens retorna a l'usuari i l'unic que hem de fer és agafar el "AvatarUrl" d'aquest.

Carrito Controller

Aquest arxiu de controllers en servirà per poder gestionar, modificar i esborrar informació en la base de dades relacionada amb el **carrito**, que vendría a ser el **pedido** de l'Usuari que és troba amb **estat "Pendiente"**.
Explicarem cada un dels controllers a continuació:

- **@GetMapping("/getCarrito"):** Aquesta funció ens serveix realment per a poder tornar a la part del client tots els plats que te el carrito d'un determinat usuari. Per això llegirem el **token** on tenim guardat l'email de l'usuari i a continuació cercarem l'usuari segons aquest email. Després cercarem el pedido que tengui com **usuari aquest** i que **l'estat** sigui **"Pendiente"**. Després cercarem els seus platos i retornarem una llista de plats en forma de JSON cap a la part del client.

- **@PostMapping("/getCarritoById"):** Realment aquest mappeig ens serveix per retornar els plats d'un pedido segons la id que ens passen. Per això llegirem primer el **payload** on és troba la id del pedido, i cercarem el pedido segons aquesta id. A continuació agafarem tots els seus plats, els ficarem dins una Llista i el retornarem en forma de JSON cap a la part del client.

- **@GetMapping("/getPedido"):** Realment aquest mappeig ens serveix per retornar la id del pedido que és trobi en estat **"Pendiente"** en aquest cas, seria per retornar el carrito. Per això haurem de cercar segons aquest estat i segons l'usuari que vol aquesta informació.

- **@PostMapping("/getPedidoById"):** Dins aquesta funció el que feim és llegir el **payload** on ens passen la id del Pedido, a continuació cercam el pedido segons aquesta ID i ho retornam al client en forma de JSON.

- **@GetMapping("/getMenus"):** Aquest mapping l'utilitzam per retornar a la part de client tots el menus que conté un carrito. Per això cercam a l'usuari segon el email que ens passen per **Token.** A continuació cercam el pedido segons aquest usuari i el estat "Pendiente" i una vegada tenim el pedido, només hem de retornar una llista de Menus en forma de JSON cap a la part del client.

- **@PostMapping("/getMenusById"):** Aquest mapping l'utilitzam per retornar a la part de client tots el menus que conté un carrito. Per això cercam a l'usuari segon el email que ens passen per **Token.** A continuació cercam el pedido segons aquest usuari i el estat "Pendiente" i una vegada tenim el pedido, només hem de retornar una llista de Menus en forma de JSON cap a la part del client.

Ingrediente Controller

En aquest arxiu trobarem tots els controllers relacionats amb el que te a veure amb els ingredients i el seu maneig.

- **@GetMapping("/admin/getAllIngredientes"):** Funcio que retorna la llista d'ingredients de la BBDD.

- **@GetMapping("/admin/addIngredient"):** S'introdueix l'ingredient que es pasa per parametre des de la vista a la BBDD

- **@GetMapping("/admin/deleteIngredient"):** Funcio que borra l'ingredient que es pasa per parametre mitjançant el **PAYLOAD.**

Alergeno Controller

En aquest arxiu trobarem tots els controllers relacionats amb el que te a veure amb els ingredients i el seu maneig. A més de la sentència encarregada de crears els elements a la BBDD a través de la anotació @postconstruct

- **@PostConstruct:** Aquesta funció s'executa una vega s'han construit les taules de la bbdd mitjançant hibernate. En el nostre cas agafam un fitxer local en format **JSON** on tenim una llista d'alergens. Una vegada procesada la informació del fitxer, aquests s'introdueixen a la bbdd com a Alergenos.

- **@GetMapping("/profile/getAllAlergenos"):** Es recuperen de la bbdd els alergens i es pasen a la vista.

- **@GetMapping("/profile/getAllFromUser"):** Amb el token es recuperen els alergens de l'usuari en concret.

- **@PostMapping("/admin/addAlergeno"):** Funció que s'encarrega d'afegir a la llista d'alergeno aquest element.

- **@PostMapping("/profile/updateAlergenos"):** Com diu el propi nom del mappeig, aquesta funció/controller ens serveix per a modificar la llista d'al·lèrgens d'un usuari en concret. Per això cercarem a l'usuari segons el **Token** que ens passen. A continuació borrarem tots els alergenos que te l'usuari per a poder afegir tots els alergenos que ens passen per llista per el **Payload**.

Test Controller

En aquest arxiu el sistema de validació de token per a que l'accés a les zones privades del servidor siguin controlades també a la vista.

- **@GetMapping("/validateToken"):** Per tal de mantenir sempre securitzada la part de la vista, apart dels interceptors com a mètode de securització s'ha creat aquest controller per validar els usuaris que intenten entrar a zones que nomes estàn disponibles per a usuaris registrats, ja que la petició es fa a l'iniciar el VUE-ROUTER.

Login Controller

En aquest arxiu trobarem tots els controllers relacionats amb el que te a veure amb el login i el registre dels usuaris de la nostra pàgina web.

- **@PostMapping("/login"):** Funció que recoeix del **PAYLOAD** els parametres del formulari de login de la vista. Una vegada comprovat que l'usuari existeix retorna la informació de l'usuari necesaria per a procesar els diferents elements de la vista, aixi com el token imprescindible per a ficar com a header per a fer les peticions una vegada l'usuari es loggetja.

- **@PostMapping("/register"):** Després de rebre els valors del formulari de registre es prepara l'obejecte i es comprova que no existesqui previament. Una vegada comprovat, s'afegeix la imatge a l'usuari i es guarda a la BBDD.

Login Oauth Controller

- **@GetMapping("/loginOauth"):** S'encarrega de redireccionar a la url de l'autentificació per google.

- **@PostMapping("/auth/oauth2callback/"):** Una vegada google comprova que l'autentificació amb ells es positiva es redirigeix cap a aquest controller per a guardar la informacio a la BBDD de l'usuari que vol iniciar sessió. De la mateixa manera que en el controller de **REGISTER**, es pasa a la vista la informació de l'usuari necesaria per a la seva posterior utilització.

- Properties

La clase estándar de Java Properties (java.util.Properties) ens permet manipular arxius que emmagatzemen una serie de variables i valors amb el seu nom definit per després poder utilitzar-los en la resta del programa.

És a dir, que podem utilitzar aquesta arxiu per a la creació de variables generals i IMPORTANTS (no ens serveix per guardar un simple String d'una lógica de programa sino per a guardar valor com contrasenyes...).

En el nostre projecte, utilitzarem aquest arxiu per indicar a quin port volem escoltar, és a dir, quin port hem de tenir obert per a que la part de client ens cridi.

També guardarem el url cap a la nostra base de dades juntament amb el User de la db, el seu password i les línies de configuració de Spring per a poder adaptarlo a la base de dades.

Afegirem unes altres dades molt importants que serien el **secret** per a poder **sebre si el Token és valid o no** i definim també el temps que volem que duri aquest Token(el **temps** esta definit en **milisegons**, és a dir **1000 = 1 segon**).

A més. ja que aquest arxiu no el publicam a cap puesto, afegim també les credencials de les aplicacions associades, com seria Google per a fer el Login amb ells, i Stripe com a mètode de pagament. Es important crear un arxiu applicattion.properties al directori resources amb la següent informació per a poder executar l'aplicació correctament:

```
server.port=5000
token.secret=sfgdsagsfdafsdfsfgdfdagsadf
token.expiration.time=10000000000
logging.level.org.springframework.jdbc.core=TRACE
spring.datasource.url                                                                    =
jdbc:mysql://localhost:3306/hibernate?allowPublicKeyRetrieval=true&useSSL=false&createDatab
aseIfNotExist=true&serverTimezone=UTC
spring.datasource.username = root
spring.datasource.password = root
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5InnoDBDialect
spring.jpa.properties.show-sql = false
spring.jpa.properties.hibernate.format_sql = false
spring.jpa.hibernate.ddl-auto = update
client-id=979422366139-g5mjmmvbdaai6i4b431dbeba661tjqhk.apps.googleusercontent.com
client-secret=oFl77VlzTxZVgR7lRiTsEOy0
redirect-uri=http://localhost:8080/auth/oauth2callback/
```

```
server.domain=http://localhost:5000
STRIPE_SECRET_KEY=sk_test_KnQIFLmpCoWbMWGMXTP23W7V00jj1MLOZi
STRIPE_PUBLIC_KEY=pk_test_LESShQ47cPmtRV4MhkefvSax00lZTqQsOv
```

- Stripe

  Stripe és un servei basat en el núvol que permet a empreses i individus rebre pagaments a través d'Internet i ofereix biblioteques de la banda de el client i biblioteques de la banda de servidor.
  Stripe proporciona una capa d'abstracció que redueix la complexitat de rebre pagaments. Com a resultat, no necessitem tractar els detalls de la tarjeta de crèdit directament, sino que tracten amb un símbol que simbolitza una autorització per cobrar.

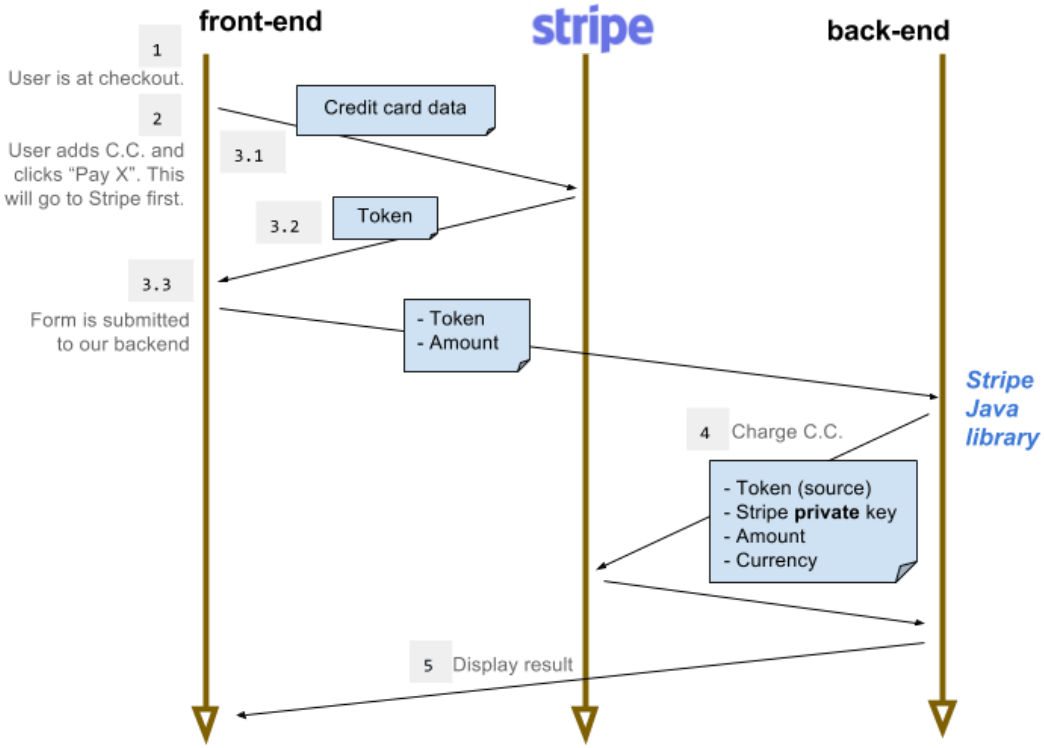
  El càrrec de la tarjeta de crèdit es realitzarà en cinc senzills passos, que inclouen el front-end (executar en un navegador), el back-end (la nostra aplicació Spring Boot) i Stripe:

- Un usuari va a la pàgina de pagament i fa clic a "Pagar amb targeta".

- A un usuari se li presenta un quadre de diàleg de superposició de Stripe Checkout, on es completen els detalls de la targeta de crèdit.

- Un usuari confirma amb "Pagar <quantitat>" que:
  - Envia la targeta de crèdit a Stripe
  - Obtingui un símbol en la resposta que s'afegirà a l'formulari existent
  - Envieu aquest formulari amb la quantitat, la clau d'API pública, el correu electrònic i el testimoni al nostre back-end
  - El nostre back-end contacta a Stripe amb el testimoni, la quantitat i la clau secreta d'API.
  - El back-end verifica la resposta de Stripe i proporciona a l'usuari comentaris sobre l'operació.

A continuació deixarém un petit esquema de com és relitza aquestes operacions per darrera:



Llavors en resum el que feim és:

- Desde la part de client agafan el compte de la tarjeta i ho passam a Stripe el cual ens confirma si aquesta tarjeta és valida i ens retorna un token de confirmació d'aquesta tarjeta.

- A continuació, una vegada hem rebut el token de la tarjeta, se la pasarem al servidor juntamente amb la quantitat total a cobrar.

- Una vegada el servidor ha rebut aquesta informació, crearem un objecte de tipus Charge on li afegirem una descripció, i la quantitat a abonar.

- Una vegada creat l'object "Charge" passarem al servidor de Stripe aquest darrer object juntament a la clau publica de Stripe i una altra vegada la quantitat a abonar.

- I per finalitzar si tot ha anat bé, ens retornará cap a la part de client amb un missatge de confirmació de que la transacció s'ha realitzat amb èxit

FrontEnd

En quant a la part de FrontEnd utilitzam HTML, CSS i JAVASCRIPT per a desenvolupar tota l'aplicació frontEnd. Com a framework de javascript empram VUE i Quasar com a framework de VUE, pero exactament, que és Quasar?? És un marc front-end de codi obert desenvolupat basant-se vue.js, que pot ajudar els desenvolupadors web a crear ràpidament llocs web adaptables, responsive, mòbil (a través de Cordova) o multiplataforma a través de Electron.

Quasar utilitza como a tecnologia el vue-router que és el paquet oficial de vuejs que ens ajuda a crear aplicacions d'una sola pàgina (Single Page Applications). En resum el **router** ens permet crear un arxiu de mappeigos on definim les url's i la vista a la cual va referenciada aquesta url. Per exemple, el vue-router ho podem trobar en la carpeta src/router/routes.js on veurem que tenim diferents apartats, el public ("/"), el privat ("/profile") i la part de admin ("/admin") i dins cada apartat, els diferents link/mappeigos de les vistes del nostre FrontEnd.

Manual tècnic

Entorn de desenvolupament

- Obrir un terminal i situar-se a la carpeta "Dev", per poder executar les intruccions correcponents
- Executar l'Script **"startup.sh"** amb els permisos corresponents per poder executar docker en el sistema que es vulgui desenvolupar

Una vegada l'escript s'hagi executat tendrem al nostre sistema de desenvolupament 3 contenidors DOCKER:

1. Contenidor **phpmyadmin** per gestionar les nostres BBDD mysql a la direccio http://localhost:8090

    a. usuari: **root**
    b. contrasenya: **root**

2. Contenidor **MYSQL** amb el servei corrent preparat per funcionar mapetjat al por 3306

3. Contenidor **NODE** amb l'execució de la comanda **npx quasar dev** per tal de poder tenir el nostre projecte de **frontEnd** corrent en mode desenvolupament per tal de veure els nostres canvis en **HOT RELOAD.**

```
CONTAINER ID   IMAGE                  COMMAND                  CREATED          STATUS          PORTS                                NAMES
0463b6f63455   phpmyadmin/phpmyadmin  "/docker-entrypoint.…"   10 seconds ago   Up 9 seconds    0.0.0.0:8090->80/tcp                 phpMyAdminVueForm
3e087465a58f   mysql:8.0              "docker-entrypoint.s…"   10 seconds ago   Up 10 seconds   0.0.0.0:3306->3306/tcp, 33060/tcp    mysql
94f949118fce   node                   "docker-entrypoint.s…"   12 seconds ago   Up 11 seconds   0.0.0.0:8080->8080/tcp               node
```

Ens assegurarem que el fitxer de settings.js del nostre projecte de VUE estiguin hablitats els paràmetres de desenvolupament.

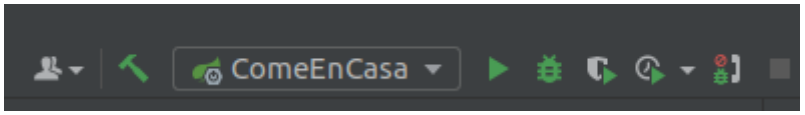
```
/////////////// SETTINGS PARA DESARROLLO ////////////

export const SETTINGS = {
  URL_SERVER_API: "http://localhost:5000/",
  STRIPE_PUBLISHABLE_KEY: "pk_test_LESShQ47cPmtRV4MhkefvSax00lZTqQsOv"
};
```

Ara només queda obrir el nostre ide (en el nostre cas INTELLI J IDEA) i comprovar que el fitxer de **application.properties** estigui en mode de

desenvolupament. Si no tenim el fitxer de application.properties per que ens hem descarregat el projecte de gitHub hem de crearlo a la carpeta RESOURCES i aferrar-li la següent informació. Es important que els parametres de **redirect-uri i server.domain** siguin els correctes per a iniciar el projecte en mode desenvolupament.

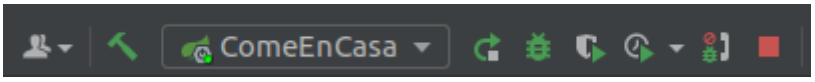
```
server.port=5000
token.secret=sfgdsagsfdafsdfsfgdfdagsadf
token.expiration.time=10000000000
logging.level.org.springframework.jdbc.core=TRACE
spring.datasource.url                                                    =
jdbc:mysql://localhost:3306/hibernate?allowPublicKeyRetrieval=true&useSSL=
false&createDatabaseIfNotExist=true&serverTimezone=UTC
spring.datasource.username = root
spring.datasource.password = root
spring.jpa.properties.hibernate.dialect                                  =
org.hibernate.dialect.MySQL5InnoDBDialect
spring.jpa.properties.show-sql = false
spring.jpa.properties.hibernate.format_sql = false
spring.jpa.hibernate.ddl-auto = update
client-id=979422366139-g5mjmmvbdaai6i4b431dbeba661tjqhk.apps.googleusercon
tent.com
client-secret=oFl77VlzTxZVgR7lRiTsEOy0

#####################################
#TRIAR UNA DE LES DUES OPCIONS
#DESENVOLUPAMENT
redirect-uri=http://localhost:8080/auth/oauth2callback/
server.domain=http://localhost:5000

#PRODUCCIO
#redirect-uri=http://comeencasa.duckdns.org/auth/oauth2callback/
#server.domain=http://comeencasa.duckdns.org:5000


#####################################
STRIPE_SECRET_KEY=sk_test_KnQIFLmpCoWbMWGMXTP23W7V00jj1MLOZi
STRIPE_PUBLIC_KEY=pk_test_LESShQ47cPmtRV4MhkefvSax00lZTqQsOv
```

Una vegada tenim tot configurat, podem iniciar el projecte.



UNA VEGADA INICIAT EL SISTEMA, PODEM ENTRAR A LA PAGINA WEB I CREAR L'USUARI ADMINISTRADOR, QUE HA DE SER UN USUARI AMB LA DIRECCIÓ DE CORREU "admin@comeencasa.com"

Una vegada haguem acabat de desenvolupar i volguem aturar el nostre entorn només hem de anar al terminal i executar l'Script **stop.sh.** D'aquesta manera s'aturen els contenidors docker i no queda cap servei en execució. També hem d'aturar el nostre projecte de l'intelliJ



Entorn de producció

- Obrir un terminal i situar-se a la carpeta "Prod", per poder executar les intruccions correcponents
- Ens assegurarem que al nostre projecte de **VUE** estan seleccionades les opcions de **PRODUCCIÓ**

L'adreça de URL_SERVER serà l'adreça on estigui allotjat el nostre Projecte de BACKEND, com a exemple hi ha la direcció on tenim allotjat el nostre projecte final.

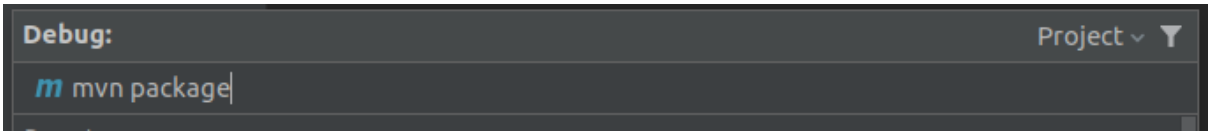
```
/////////////// SETTINGS PARA PRODUCCIÓN ////////////
export const SETTINGS = {
  URL_SERVER_API: "http://comeencasa.duckdns.org:5000/",
  STRIPE_PUBLISHABLE_KEY: "pk_test_LESShQ47cPmtRV4MhkefvSax00lZTqQsOv"
};
```

- También ens assegurarem que el fitxer **application.properties del** nostre de backend està preparat per pasar a producció seleccionant els parametres de producció corresponents. Només s'ha de modificar si el servidor de backend esta allotjat a un altre domini que no sigui localHost. En el nostre cas com a exemple hem posat la direcció del nostre servidor dedicat http://comeencasa.duckdns.org:5000

```
server.port=5000
token.secret=sfgdsagsfdafsdfsfgdfdagsadf
token.expiration.time=10000000000
logging.level.org.springframework.jdbc.core=TRACE
spring.datasource.url =
jdbc:mysql://localhost:3306/hibernate?allowPublicKeyRetrieval=true&useSSL=
false&createDatabaseIfNotExist=true&serverTimezone=UTC
```

```
spring.datasource.username = root
spring.datasource.password = root
spring.jpa.properties.hibernate.dialect                                    =
org.hibernate.dialect.MySQL5InnoDBDialect
spring.jpa.properties.show-sql = false
spring.jpa.properties.hibernate.format_sql = false
spring.jpa.hibernate.ddl-auto = update
client-id=979422366139-g5mjmmvbdaai6i4b431dbeba661tjqhk.apps.googleusercon
tent.com
client-secret=oFl77VlzTxZVgR7lRiTsEOy0

#####################################
#TRIAR UNA DE LES DUES OPCIONS
#DESENVOLUPAMENT
#redirect-uri=http://localhost:8080/auth/oauth2callback/
#server.domain=http://localhost:5000

#PRODUCCIO
redirect-uri=http://comeencasa.duckdns.org/auth/oauth2callback/
server.domain=http://comeencasa.duckdns.org:5000


#####################################
STRIPE_SECRET_KEY=sk_test_KnQIFLmpCoWbMWGMXTP23W7V00jj1MLOZi
STRIPE_PUBLIC_KEY=pk_test_LESShQ47cPmtRV4MhkefvSax00lZTqQsOv
```

- Ara generarem el nostre ftxer **.jar** mitjançant **MAVEN.** Primer de tot executam **mvn clean** per asegurar-nos que el nostre tarjet esta buid i no tenir problemes. Després ja podem generar el fitxer .jar amb la comanda **mvn package**



Aquesta comanda generarà un fitxer **.JAR** que serà el que s'emprarà per a crear la imatge **DOCKER** que pasarem a producció.

- Ara executarem el l'script **startup.sh** amb els permisos necesaris i esperarem a que acabi. Aquest Script esta preparat per funcionar amb un DokerFile que també es troba a dintre del directori.

```
FROM openjdk:14
WORKDIR /
EXPOSE 5000
COPY comeencasa.jar comeencasa.jar
ENTRYPOINT ["java", "-jar", "comeencasa.jar"]
```

Aquest dockerfile genera una imatge docker amb el fitxer jar generat anteriorment. Tots els fitxers estan configurats per funcionar amb una compte de docker hub on es puja la darrera versió generada que després s'utilitzarà per a fer el docker-compose de les imatges docker.

```
#ONLY CREATE IMAGE FROM DOCKERFILE
sudo docker build --no-cache -t comeencasa .
```

També es important modificar el fitxer del docker-compose.yml en consonància a la imatge que es vol utilitzar.

```
jdk:
   restart: on-failure
   depends_on:
    - db
   image: comeencasa:latest
   container_name: jre
   stdin_open: true # docker run -i
   tty: true # docker run -t
   ports:
    - 5000:5000
   environment:
    SPRING_DATASOURCE_URL: jdbc:mys
```

Una vegada hagi acabat d'executar-se l'script podem revisar que els contenidors estan funcionant de manera correcta amb la comanda **DOCKER PS.**

```
CONTAINER ID   IMAGE                      COMMAND                  CREATED          STATUS          PORTS                                   NAMES
04a5d3412264   neo3kk/comeencasa:latest   "java -jar comeencas…"   7 minutes ago    Up 7 minutes    0.0.0.0:5000->5000/tcp                  jre
f88b52c1e415   phpmyadmin/phpmyadmin      "/docker-entrypoint.…"   12 minutes ago   Up 12 minutes   0.0.0.0:8090->80/tcp                    adminphp
e16e57924f49   httpd:latest               "httpd-foreground"       12 minutes ago   Up 12 minutes   0.0.0.0:8080->80/tcp                    apache
7455f6a17156   mysql:latest               "docker-entrypoint.s…"   12 minutes ago   Up 12 minutes   0.0.0.0:3306->3306/tcp, 33060/tcp       mysqlcontainer
```

Si volem aturar els nostres contenidors basta en executar l'Script **stop.sh.**