**GRUP 4 PROJECTE FINAL**

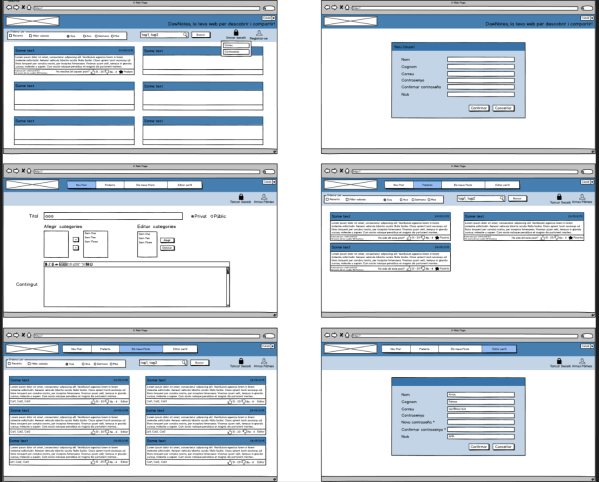
**Autors:**

Jonatan, Martí , Arnau i Josep

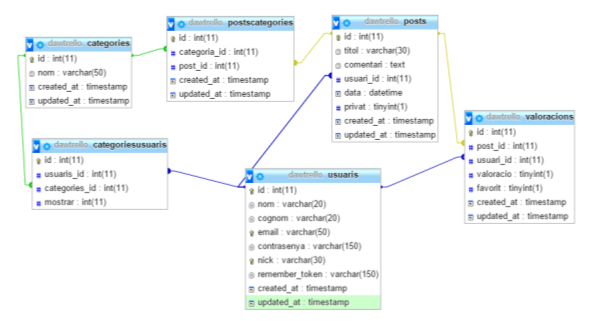
**Títol del projecte:**

Dawtrello

1. [**Mockups**](#Mockups)
2. [**Base de dades**](#Base_de_dades)
3. [**Casos d’ús**](#CasosDus)
4. [**Funcionament del projecte**](#Funcionament)
5. [**Pressupost**](#Pressupost)
6. **Mockups**



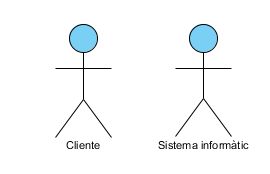
**2.** **Base de dades**



Els usuaris tenen posts dins de categories creades per ell mateix i poden pertànyer a varies categories. Els usuaris poden valorar tots els posts positiu o negativament i tenir-los com a favorits. També poden elegir si el post es privat o públic, si es privat no sortirà en la recerca de posts.

1. **Casos d’ús**

Actors



**Client: Usuari que utilitza la pàgina web.**

**Sistema informàtic**: Sistema que gestiona i automatitza les diferents possibilitats que té el usuari dins la pàgina web.

**Identificació de casos d’ús:**

* **Registre de usuaris:** Permet donar d’alta un usuari a la web.
* **Alta de notes:** Permet donar d’alta notes i també permet crear tags.
* **Selecció de preferits:** Permet escollir notes preferides per guardar-les.
* **Recerca de notes**: Permet buscar les notes publiques de tots els usuaris.
* **Visualització de categories**: Permet visualitzar les notes endreçades per categories en forma de categories.

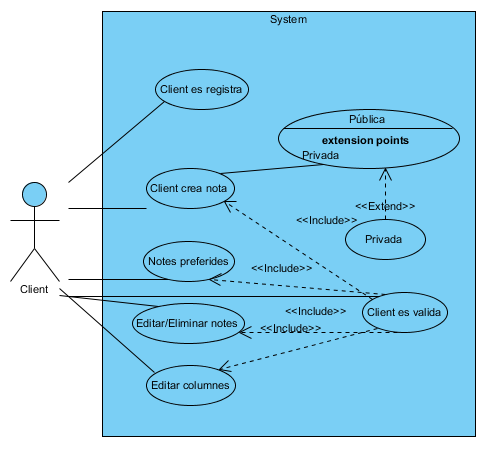
**Detall de casos d’us:** Publicar una nota

**Objectiu:**

L’objectiu és que l’usuari doni d’alta una nota per poder guardar-la i compartir-la si vol, escollint si aquella nota és pública o privada i classificada per tags.

**Precondició:**

L’usuari ha d’estar registrat.

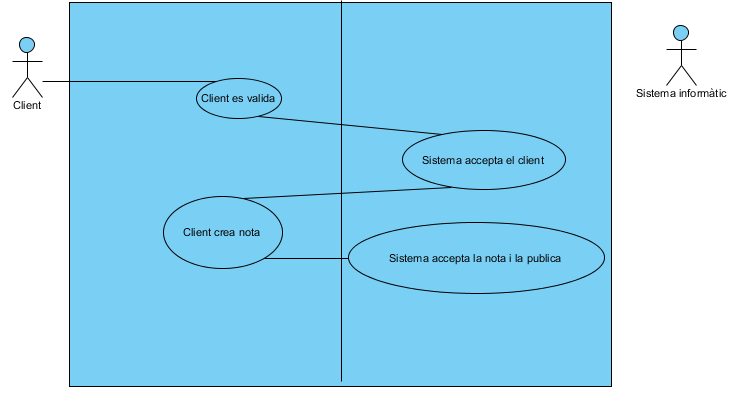


Flux bàsic

1. L’usuari es valida en el sistema.
2. L’usuari es acceptat pel sistema.
3. L’usuari crea la nota[A1]
4. El sistema publica la nota

Fluxos alternatius

[A1] -> L’usuari pot crear una nota pública o privada. Si es pública tothom podrà veure aquella nota, si es privada només l’usuari creador la veurà.



1. **Funcionament**

En aquest apartat, esmentarem per sectors com fer diverses coses que puguin ser d’utilitat per el frontend, com crear pagines, crear formularis, connexions a base de dades, controls de ruta, passar variables, el necessari per que qualsevol persona nova en el projecte pugui entendre com funciona i després de llegir-ho sàpiga crear una pàgina nova amb qualsevol contingut.

**Primer apartat:**

Com crear una pàgina

Com hem dit anteriorment, el framework laravel segueix unes pautes per a que el seu funcionament sigui òptim i correcte en tots els seus aspectes, hem de saber que totes les pàgines noves amb contingut nou les crearem dins de la següent ruta:

App->Views->Pages.

Qualsevol pagina nova allà ha de tenir una extensió .blade.php. Això ens servirà per poder programar en php(laravel) dins d’aquestes pàgines.

Per poder crear la pàgina cal tenir fixat un “layout” per defecte, és a dir, una estructura de la pàgina general a la que incorporar-hi el nou contingut.

Aquest layout ha d’estar situat a views ->layouts.

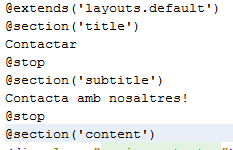
El nostre layout per defecte es el següent:



Com podem observar, el laravel ens permet fer pàgines molt mes optimes gracies a la utilització d’includes.

Aquest layout no s’ha de tocar però s’ha d’incorporar a qualsevol pàgina nova, el que podrem tocar serà el contingut, el apartat de “@yield(‘content’).

Per tant, totes les pagines començaran per el següent format:



Extenem el layout default, posem la secció del títol amb el titul sota, tanquem el head, hi posem un subtítol, el tanquem, obrim la secció, hi posem tot el contingut que vulguem i tanquem amb un @stop.

Automàticament es posarà el footer i demes components de la pagina que té per defecte el nostre layout principal.

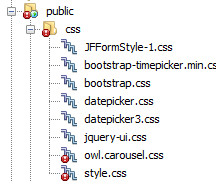
El nostre layout principal és tan òptim gracies al haver creat uns includes en el apartat views->includes, com per exemple, el nostre footer-info.blade.php :



On podem veure que es una pàgina normal i corrent amb permisos de programació php(larvel) gracies a la extensió .blade.php.

Aquests includes si els toquem, modificarem totes les pàgines que el continguin, una manera molt optima de treballar i corregir errors.

* Els arxius de css i js que totes les pàgines tenen està situat a públic->css.



Una vegada hem estructurat la nostre nova pàgina cal definir-la en una ruta, una ruta que qualsevol persona posi, li indiqui que anirà a aquella pàgina sense necessitat de posar la direcció de les carpetes ni la extensió, per fer això cal anar a **routes.php**, a la carpeta de views.

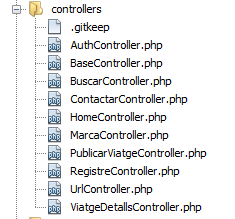
Aquest arxiu ens permetrà controlar totes les rutes de la nostre pagina web:



La imatge indica l’estructura que tenim nosaltres, cal posar el nom de la pàgina que vulguem, i enviar-ho a un controlador.

Els controladors serveixen per, tal com diu el nom, controlar que es el que fa aquesta direcció, si mostrar una pàgina directament, si fer una sèrie de funcions, si aquesta pàgina ve després d’un formulari serà un POST, etc...

Tots els controladors estan situats a app->controllers:



Anem a veure un exemple del controlador que tenim al nostre buscarruta:



Aquest controlador és un arxiu amb una classe php, dins, hi posem la funció la qual hem determinat a la ruta, després d’un “@”.

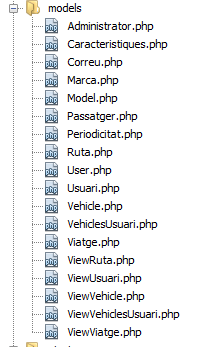
En aquest controlador, agafem una sèrie de variables per get que ens envia la pàgina de buscar ruta, aquestes variables les guardem en una array i ens envia a la pàgina “pages.buscarruta, $variables’.

Si tenim una pàgina creada amb el nom buscarruta on la direcció anterior, ens enviarà a aquesta, gracies al retorn View::make.

Els controladors serveixen per utilitzar la informació que enviem o agafem i treballar sobre aquesta, passant una sèrie de controls, com per exemple, si no s’omple un formulari, ens retorni a la pàgina anterior amb un error.

**Configurar les nostres taules de la base de dades i la base de dades**

A la ruta App->Models, tenim creats per cada taula i vista, un arxiu associat a aquest:



L’estructura de cadascun d’aquests arxius es la següent:



Al posar “use sleepingOwl...”, ens permetrà que el nostre backend pugui utilitzar la taula en concret.

Cal crear una classe anomenada igual que la taula on indiqui el nom i els camps, després podem crear funcions per poder utilitzar dins del laravel i del nostre backend orientat al laravel.

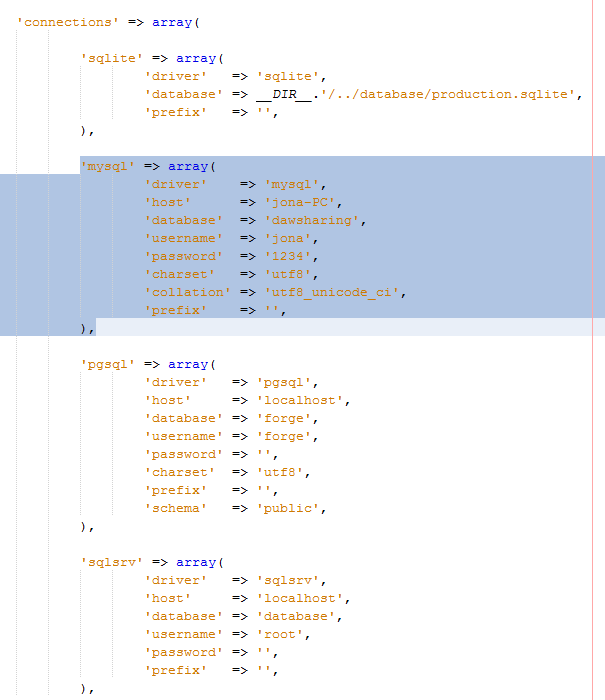
Una vegada tenim creada la classe, a la nostre pàgina amb extensió .blade.php hi podem accedir a aquestes taules amb el següent format:



Nom de la taula, funció, paràmetres i ->get(), el qual ens donarà per json els valors de la taula si es que els té.

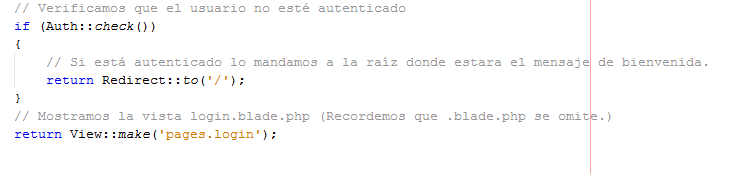
Dit això, per poder fer-ho, cal tenir configurada una base de dades en el framework, aquesta la podem editar en un arxiu anomenat database.php a app->config.

Aquest es el arxiu:

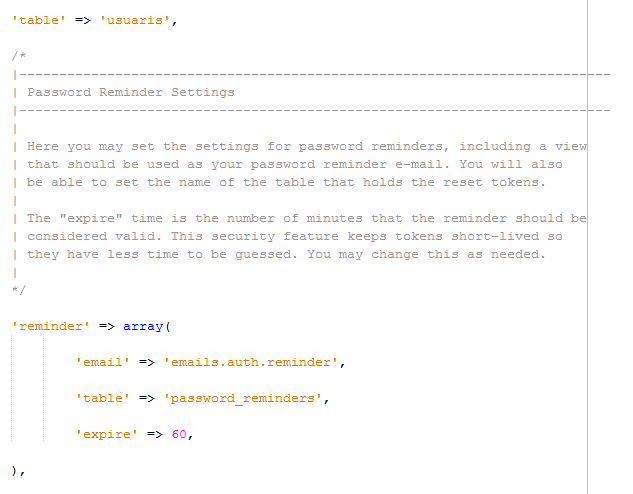


Subratllat amb color, tenim la nostre base de dades principals, on indiquem el nom que vulguem utilitzar, direcció, usuari, contrasenya etc.

En aquesta mateixa carpeta hi podem trobar el arxiu auth.php, aquest s’encarregarà d’autentificar els usuaris a la nostre web per després poder fer funcions en els controladors com la següent:



En aquest arxiu només hem de dir quina taula utilitza i quins camps han de concordar, amb el temps que dura aquesta sessió:



En els arxius principals de laravel, tots venen documentats amb ingés per a qualsevol modificació que hi vulguem fer, sabem relativament que es el que estem tocant.

**Com enviar correus i com configurar un “template” de correus.**

Gracies al framework de laravel, ens permet enviar e-mails d’una manera molt senzilla. Exposaré un exemple:

Tenim una pàgina amb el nom contactar.blade.php, aquesta conté un formulari de contacte:



Aquest formulari ens enviarà a contactar, una vegada li donem al botó d’enviar.

Després el nostre arxiu de rutes ens envia al nostre controlador d’aquesta pàgina per POST i a la seva funció postFormulari:



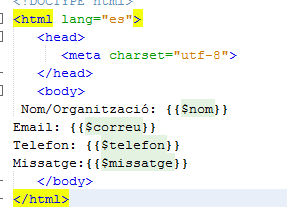
La nostre funció del controlador es la següent:



En aquest controlador hi guardem en un array, les variables del formulari anterior, després de pasar una validació de controls sobre el formulari, utilitzem la funció **Mail::send+paràmetres.**

Aquests paràmetres són els següents:

Página per defecte de l’estructura d’aquest email, en el nostre cas es “emails.contacto”, que conté la següent estructura creada a views->emails->contacto.blade.php



Després de seleccionar la pàgina hi introduïm les variables que volem utilitzar en aquesta, seguida de la funció “message” del laravel.

Després podem modificar paràmetres com a qui li enviem, el títol, subtítol, de part de qui ve, etc.

Nosaltres ho enviem a [dawsharing04@gmail.com](mailto:dawsharing04@gmail.com) amb el subjecte de “Missatge via contactar”.

Després de fer això ens retornarà a la pàgina principal, el nostre e-mail serà enviat amb les variables corresponents i amb la plantilla corresponent.

## 

1. **Pressupost**

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
| **DawTrello grup4 S.L.** |  |  |  |
| ***Software Development*** |  |  |  |
| Avenida Diagonal 567 |  |  |  |
| Barcelona, 08029 |  |  |  |
| Teléfono (93) 2025689 Fax (93) 2025689 |  |  |  |
| B67890215 |  |  |  |
|  |  |  |  |
| **Nº DE FACTURA:** | 1462/07-F |  |  |
|  |  |  |  |
| Fecha: | 19/05/2015 |  |  |
|  |  |  |  |
|  |  |  |  |
| **Cliente** |  |  |  |
|  |  |  |  |
| Att.: |  | Dawtrello grup 4 S.L |  |
| Dirección: |  | C/Artesans, 60 |  |
| Localidad: |  | Valls, 43800 |  |
| Telf.: |  | 0034 977 609061 |  |
| N.I.F./C.I.F. |  | B689615001 |  |
|  |  |  |  |
|  |  |  |  |
| **Nº Trabajo:** | 1462/07-F |  |  |
|  |  |  |  |
| **Concepto** |  |  |  |
| **Descripcion del proyecto o servicio** |  |  |  |
|  |  |  |  |
| **DESCRIPCIÓN** | **TOTAL HORAS** | **COSTE HORA** | **TOTAL IMPORTE** |
| **Programación del software** | 400 | 35,00 € | 14000,00 € |
| **Dominio** |  |  | 7,95 € |
| **Hosting** |  |  | 79,00 € |
| **Manteniment mensual** |  |  | 50 € |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  | SUBTOTAL | **14136,95 €** |
|  |  | TIPO IMPOSITIVO | **21,0%** |
|  |  | IMPUESTO SOBRE VENTAS | **2968,75 €** |
|  |  | **TOTAL** | **17105,71€** |