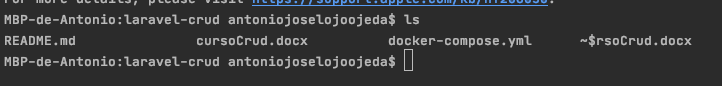
CRUD

0 PREPARACIÓN

1º Creo un **directorio para**  **ubicar docker-compose.yml**



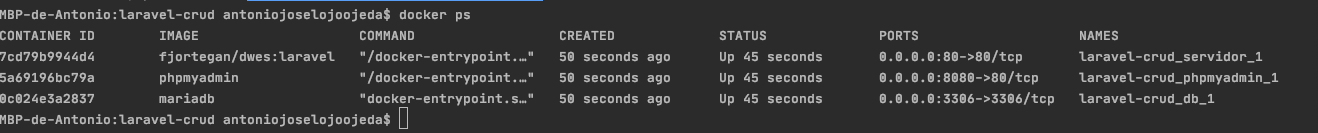
Este va ha levantar 3 contenedores. Uno con debían y nginx (**donde se creará el proyecto**)**,** un segundo con **mysql** y un tercero con **phpmyadmin** que jará más sencillo el trabajo con la BBDD.

2º **docker-compose up**



Ya tenemos en el directorio dos directorios **db-init** y **laravel**.

Además de tres contenedores funcionando. (bd, phpmyadmin, servidor)



Para crear el proyecto en el directorio del **root**

(el nombre se mira con **docker ps**)

**docker exec -it laravel-crud\_servidor\_1 /bin/bash 🡪** Ejecuta el bash del servidor

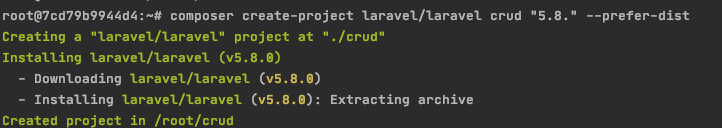
****

**3º CREAR EL PROYECTO**

Vamos a **root** y desde aquí **ejecuto el comando para crear el proyecto.**

**composer create-project laravel/laravel crud "5.8.\*" --prefer-dist**

(Crea el Proyecto laravel/laravel en /laravel con la version 5.8.lo que sea)

****

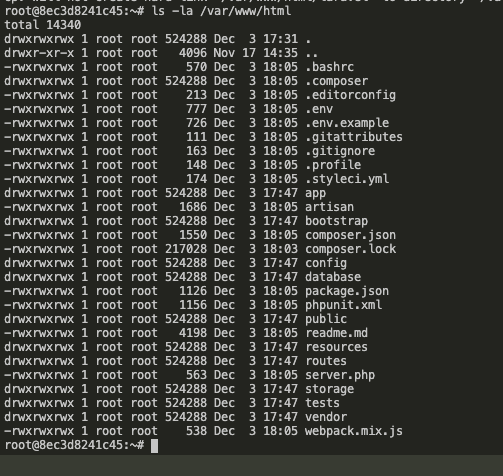
**4º DESPLEGAR EL PROYECTO**

A continuación, copiamos todo al directorio donde publica **nginx** y ya estaría laravel funcionando.

**cp -r laravel/\* /var/www/html**

**cp -R laravel/.\* /var/www/html**

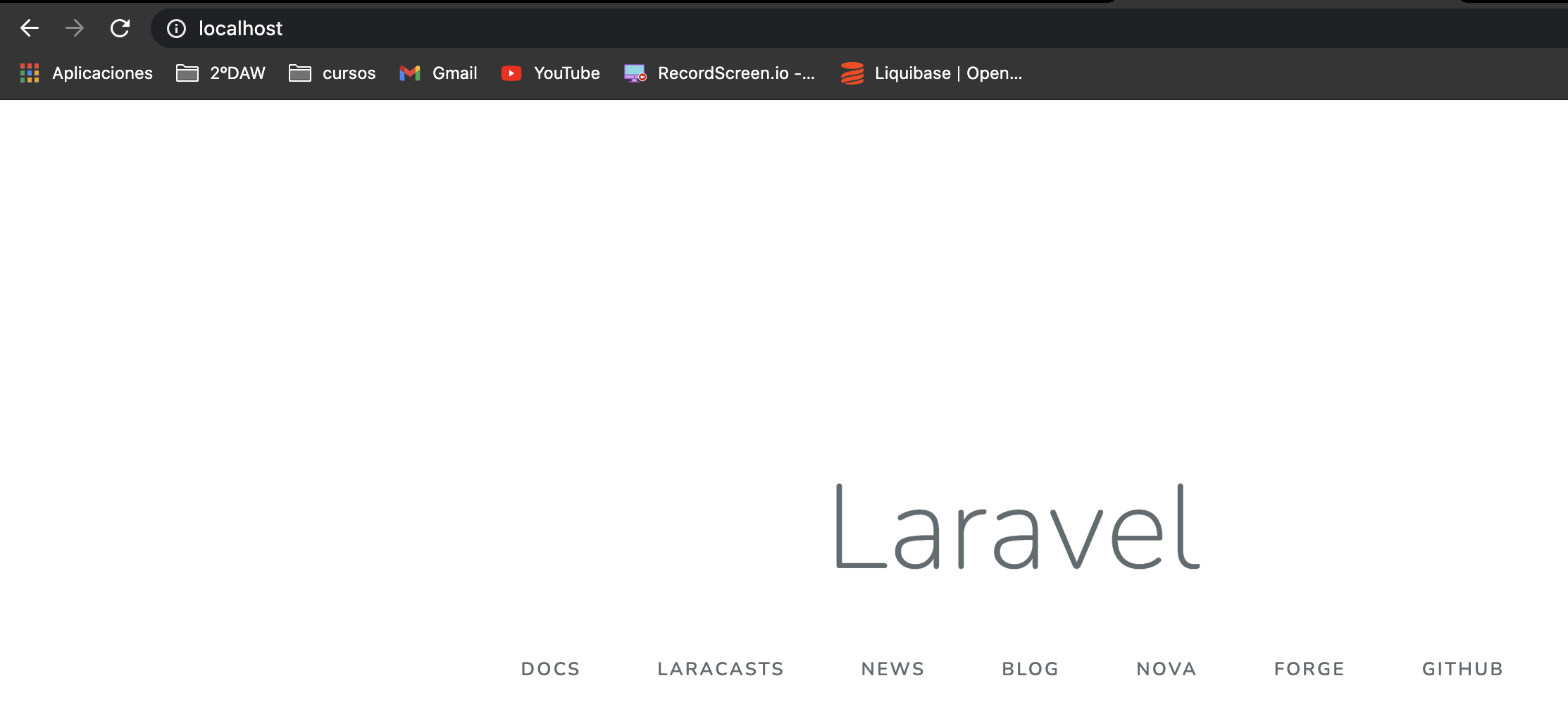
****



A plicamos permisos para poder trabajar con **nginx**.

**chown -R www-data:www-data /var/www/html**

****

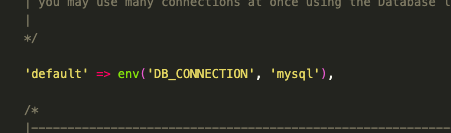
****

1 DISEÑO DE BASE DE DATOS Y CREAR PROYECTO

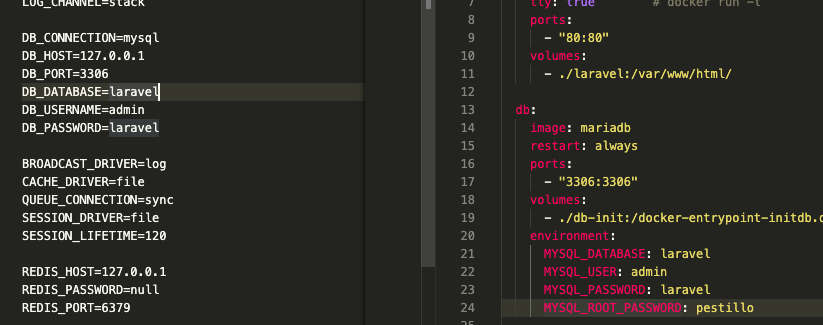
**0 CONEXIÓN CON MYSQL (MIGRATE)**

Ya podemos empezar con VSCode u otro IDE. Comenzamos comprobando que estamos usando **mysql**

**Config/Database.php**

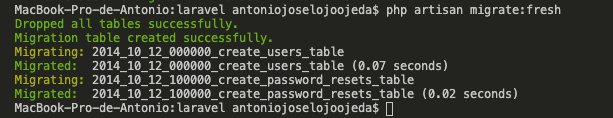
****

Comprobamos que en **.env** usamos la misma configuración que en **docker-compose.yml**

****

**Hacemos la migración.**

**php artisan migrate:fresh (limpia artisan)**

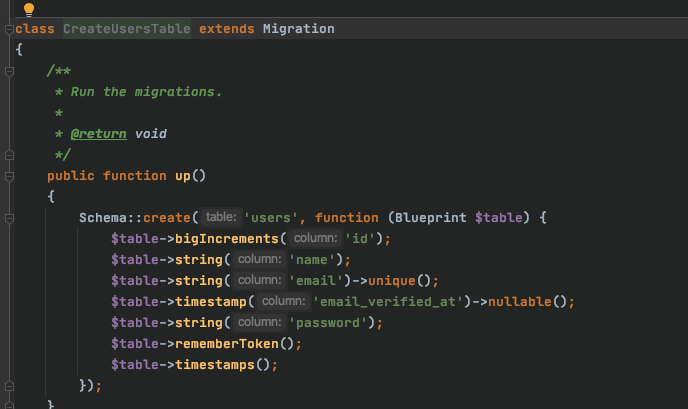
****

Estos están en **Database/migrations**

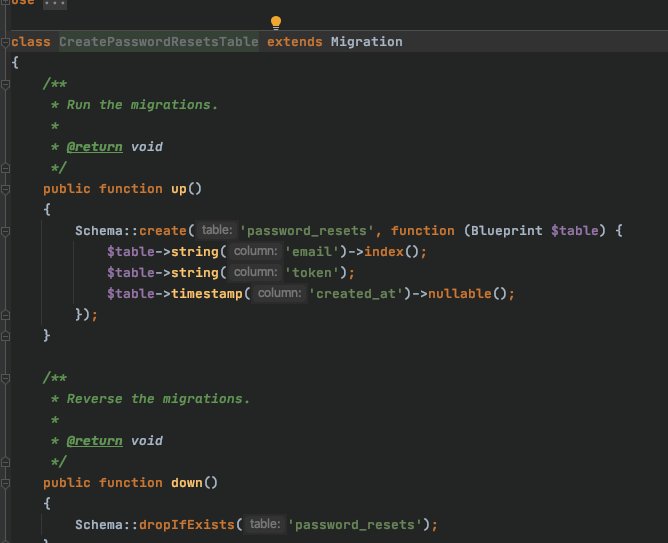
****

Ya existen dos migraciones listas **create\_users\_table, create\_password\_resets**

**createUsersTable**

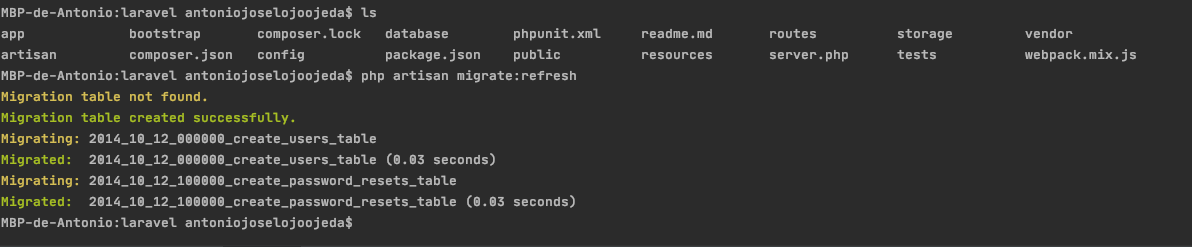
****

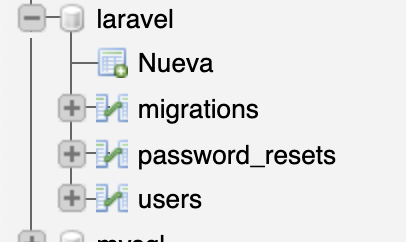
**Create\_password\_resets:table**

****

Para ejecutar todas las migraciones usamos el comando **php artisan migrate:refresh**

Este comando **laravel lee todos los datos de las tablas en el directorio migrates y los inserta en la base de datos.**



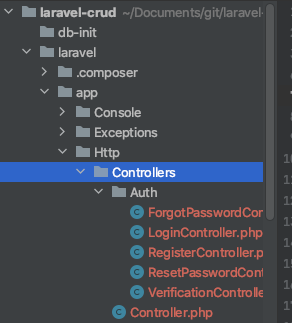


MODELO VISTA CONTROLADOR

RUTA 🡪 Controlador (Puede hacer uso de un modelo que a su vez está creado a partir de una tabla de la base de datos) 🡪 vista 🡪 Renderizar un html

**¿DÓNDE ESTÁN ESTAS CARPETAS EN LARAVEL?**

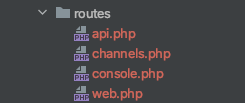
**Controladores 🡪 app/http/controllers**

****

**Vistas 🡪 resoruces/views**

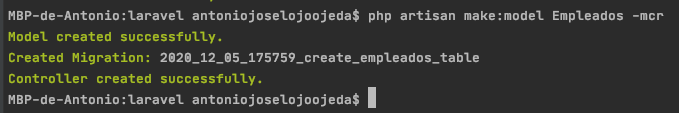
****

**Rutas routes/**

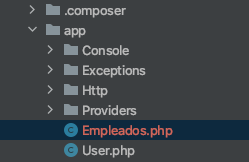
****

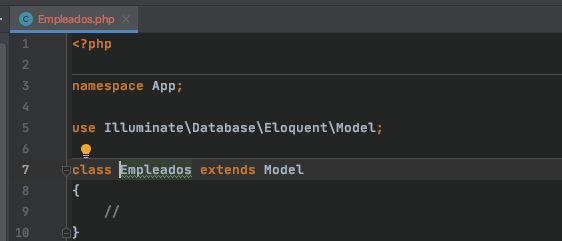
**VAMOS A CREAR ESTA ESTRUCTRA**

**php artisan make:model Empleados -mcr** 🡪 Construye el modelo Empleado junto a su controlador y recursos necesarios para empleados

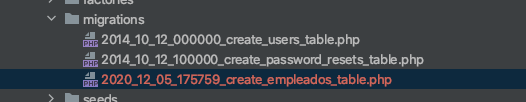


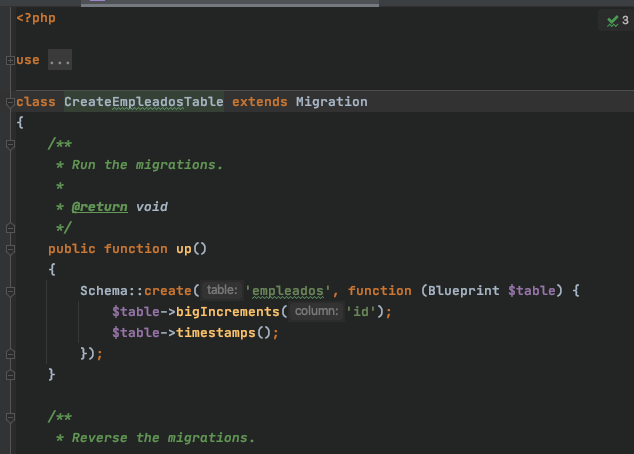
Nos indica que **se ha creado** **un modelo**, una **migración** y un **controlador**



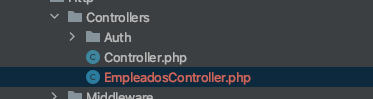


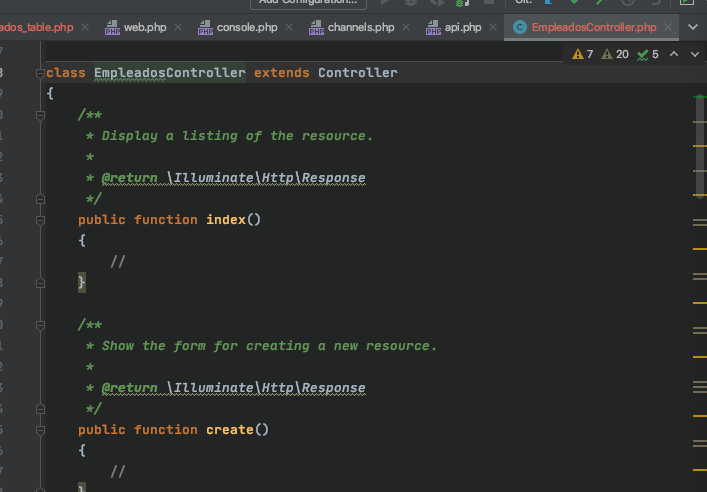
**Una migración 🡪 Esta migración es la que vamos a tener que modificar para poder generar la estructura de la tabla sobre la que vamos a trabajar**





Un controlador

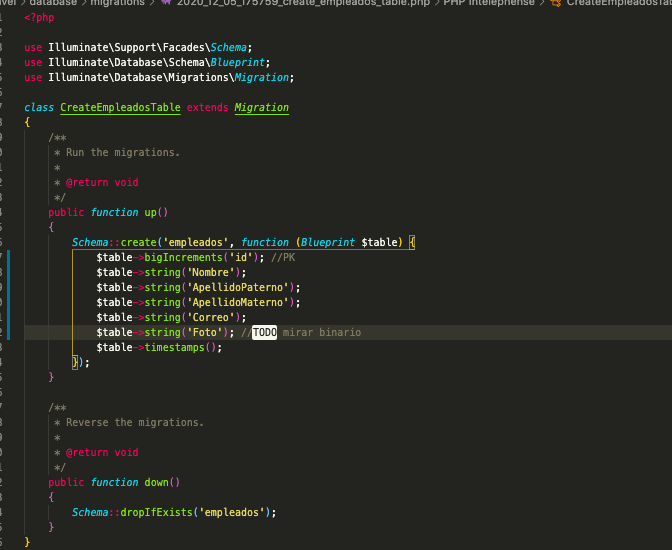




En el controlador nos crea una clase con todos los métodos para trabajar sobre una entidad **index() create() store() show() edit() update() destroy()**

MIGRATE

Vamos a **/databasemigrations/create\_empleados\_table.php** Para modificar la tabla de empleados.

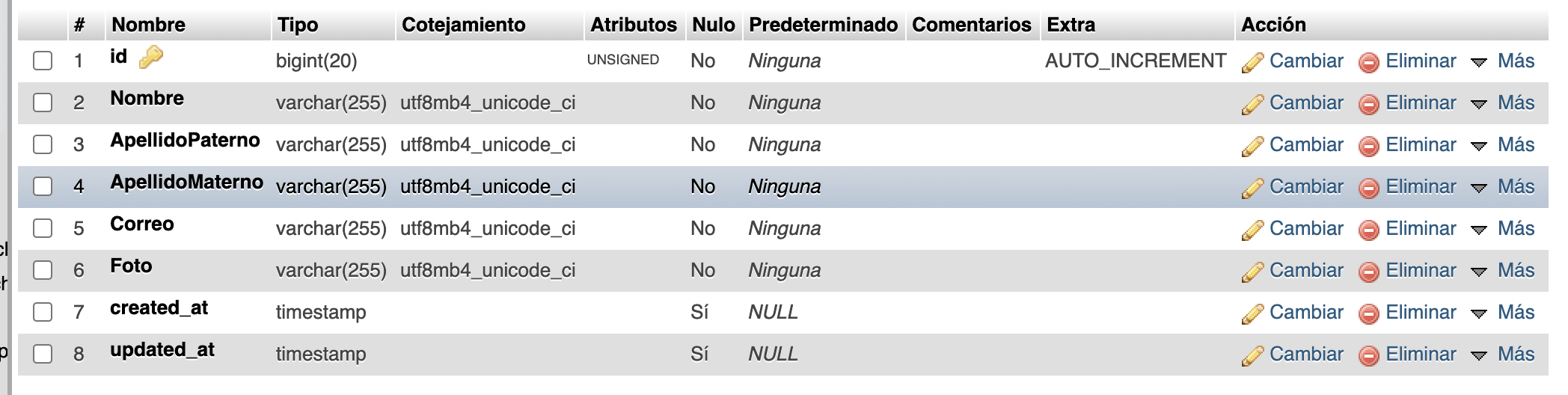


Ejecutamos la **migración para que se aplica a la base de datos**

**php artisan migrate:refresh**

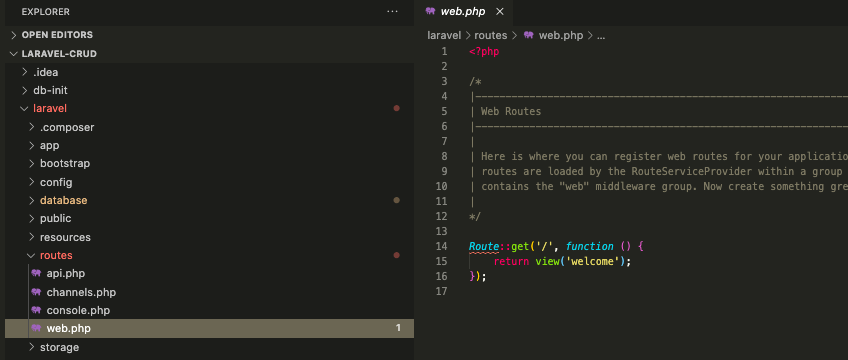
****

Comprobamos por si acaso con **phpmyadmin**

****

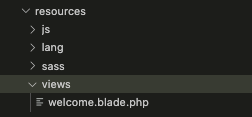
RUTA

Vamos a **routes/web.php**

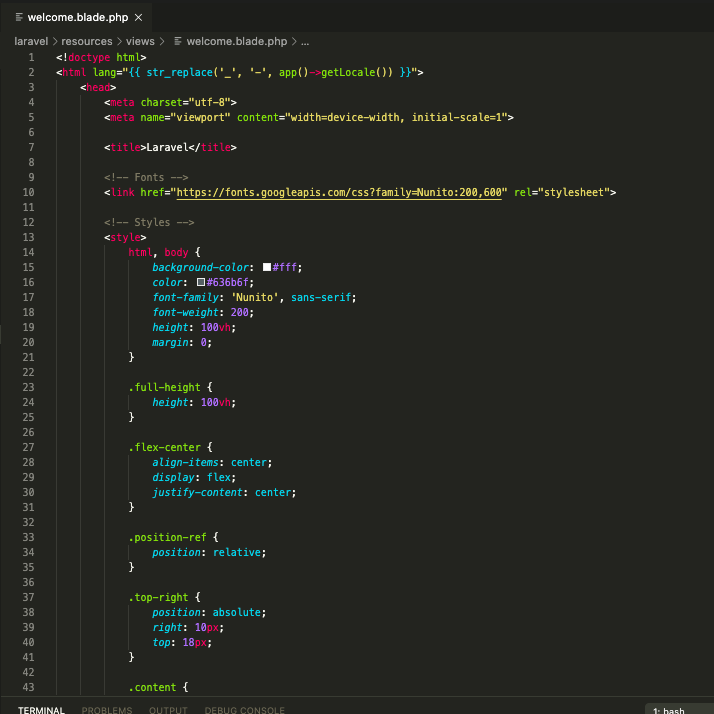
****

Aquí vemos que hay una ruta **\ (Raiz)** que nos devuelve un **vista** que se llama **welcome**

Este es el **index** por defecto **de laravel**. Este **index** está ubicado en **resources/views**

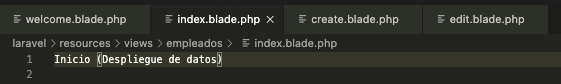
****

Corresponde a **welcome.blade.php**

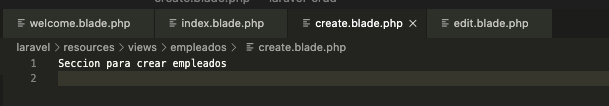
****

Para crear las nuestras vamos a iniciar creando una carpeta que se va ha llamar **empleados** dentro de esta **colocamos tres archivos (PLANTILLAS)**

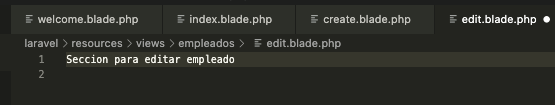
**index.blade.php** 🡪 Mostrar el contenido de index



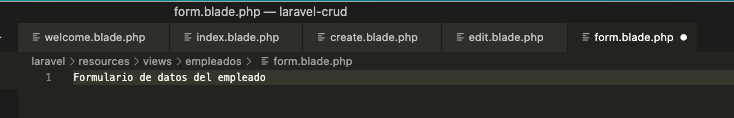
**create.blade.php**

****

**edit.blade.php**

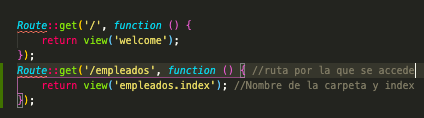
****

**CONTINUAMOS CREANDO UN FORMULARIO QUE VAMOS A USAR TANTO EN EDIT COMO EN CREATE**

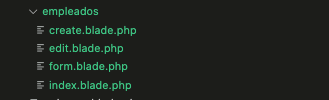
****

**PARA MOSTRAR LA INFORMACIÓN HAY QUE AÑADIR UNA RUTA IMAGINEMOS QUE QUEREMOS QUE SEA CON LA RUTA /empleados y que queremos que se vea un index**

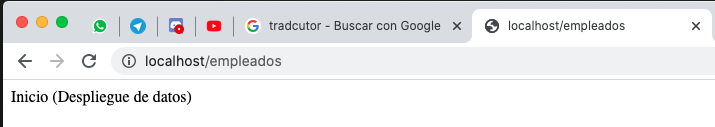
**routes/web.php**

****

**Empleados.index 🡪** Hace referencia a **index.blade.php**

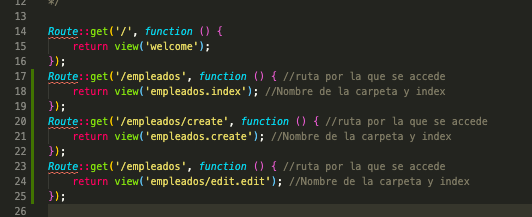
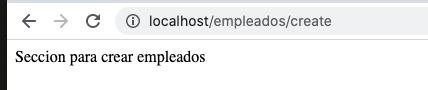
****

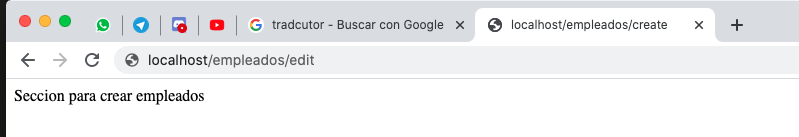
Si vamos al navegador y introducimos la ruta nos muestra lo que pusimos en la vista

****

Hacemos lo mismo con **create edit**

Teniendo en cuenta que están haciendo referencia al emnpleado se deben incluir en **/empleado**.

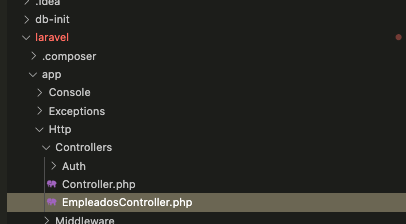
 



**¿CÓMO ACCEDER A LOS MÉTODOS QUE ESTÁN DENTRO DEL CONTROLADOR?**

Vamos al **controlador de los empleados**

/app/http/ccontrollers/EmpleadosController

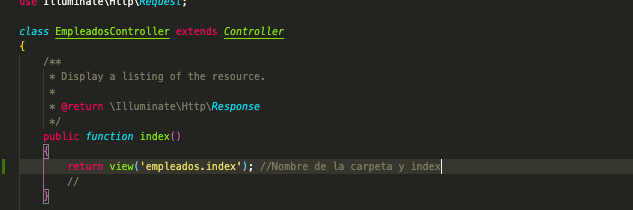


Queremos que el controlador sea el que llame a **la vista de index**.

Nos llevamos el mismo return en la función de rutas de **empleados.index**

****

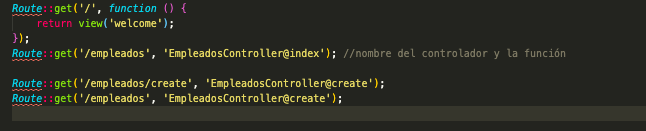
El return se lo damos a la función **index()** del controlador de los empledos.



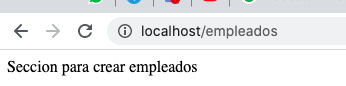
Hacemos lo mismo con **create y edit**

¿**CÓMO HACEMOS PARA QUE LA RUTA MUESTRE EL MÉTODO DEL CONTROLADOR QUE ES EL QUE VA HA LLAMAR A LA VISTA**?

Para ello usamos la siguiente sintáxis:



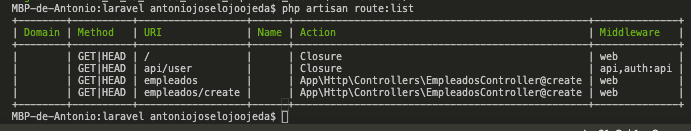
‘ruta en el navegador’, ‘controlador@función’



Sigue funcionando bien eso es **muy bien**

**PARA VER LAS RUTAS A LAS QUE PODEMOS ACCEDER TENEMOS EL COMANDO**

**php artisan route:list**



A continuación podríamos poner todas las rutas que vamos a necesitar. Pero si no las queremos poner todas podemos usar algo muy singular y que vamos a usar bastante.

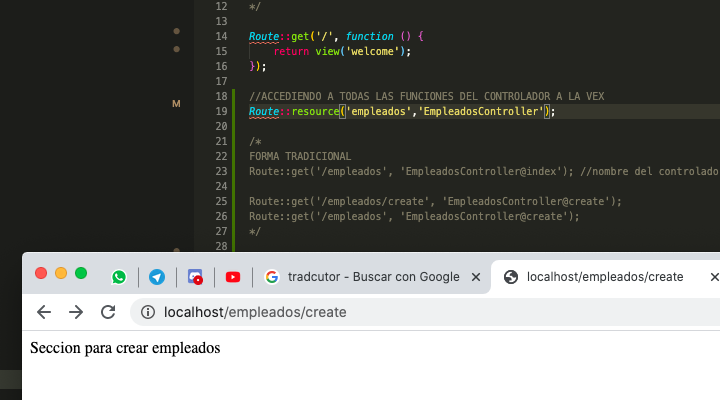
**Route::resources(‘user’, ‘UserController’)**

**‘users’ =** colocamos el nombre de la ruta principal del navegador

**‘UserController’ =** colocamos el nombre del controlador

De esta manera **laravel va ha llamar automáticamente al método del controlador siempre que este coincida con la ruta introducida**

Por ejemplo **localhost/empleado/create 🡪 EmpleadosController@create()**



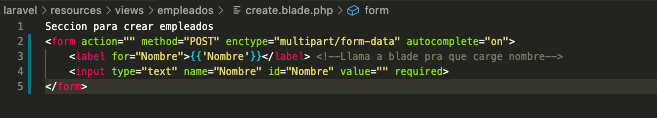
FUNCIONAS BIEEEN ESO ES BIEN

VISTA CREATE

Vamos a crear el formulario para la información del empleado que se va ha ingresar en la base de datos.

**Vamos a la vista create.blade.php**

Creamos el formulario de empleado



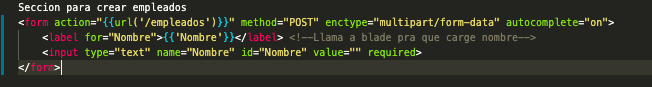
**¿HACIA DÓNDE SE HACE EL ENVÍO?**

Si hacemos **php artisan route:list**



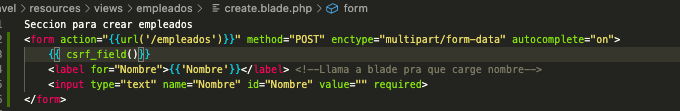
Vemos que hacemos el **método POST** al enviar a empleados y que nos lleva a **storage**

Nosotros usaremos una URL (**con el método blade**) a empleados



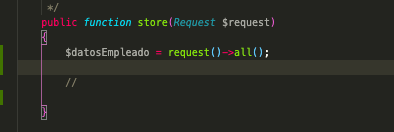
Así enviaremos a /empleados por POST de todos los empleados.

Para que esto funcione vamos a usar **una función de laravel que genera un token necesario para poder enviar POST si no es imposible**



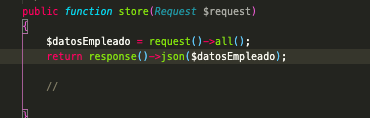
A continuación vamos a **storage()** de nuestro controlador que es donde enviamos por **POST** la información.

Si nos fijamos el parámetro es **$request** es la petición que **se genera al enviar los datos por POST**

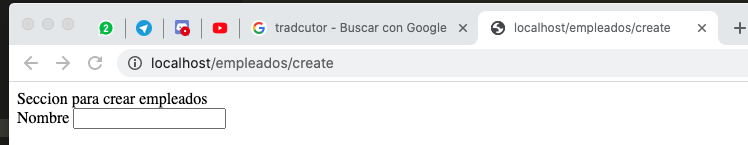


Con esto estamos accediendo a **todos los datos que hemos enviado por la petición POST al enviar el formulario.**

Para comprobar lo que se envía en lugar de una vista vamos a devolver una respuesta **json**

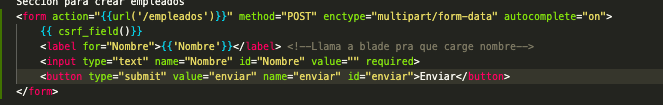


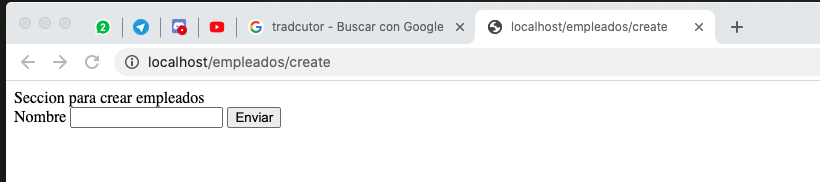
DEBERÍA DE DEVOLVERNOS UN DICCIONARIO CON LA INFORMACIÓN DEL EMPLEADO ESCRITA EN LA PANTALLA COMPROBAMOS CON EL NAVEFADOR



Carga correctamente el formulario de la vista create.blade.php que ha llamado el controlador EmpleadosCotroller@create() 🡪 VAMOS BIEN

(ME FALTA EL IMPUT ☹)





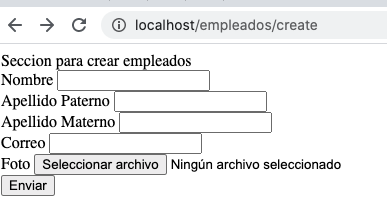


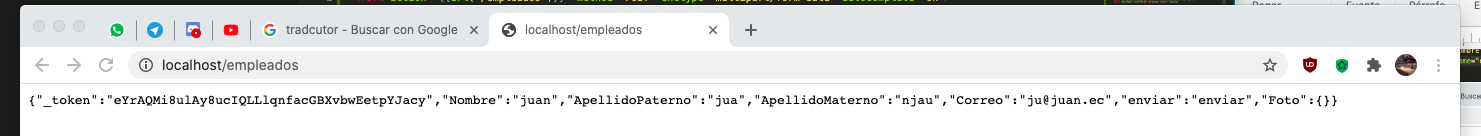
Ahora sí vemos que tenemos los valores de todos los campos genial (**Incluyendo el token**).

Una vez que ya sabemos que funciona continuamos agregando más campos.



Comprobamos si funciona





**Todo GENIAL.**

INSERTAR DATOS Y SUBIR FOTO

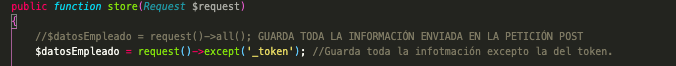
Necesitamos aislar la información y hacer que coincidan en la base de datos.

**Vamos al controller**.

En lugar de **$variable = request()->all();**

Le indico lo siguiente:

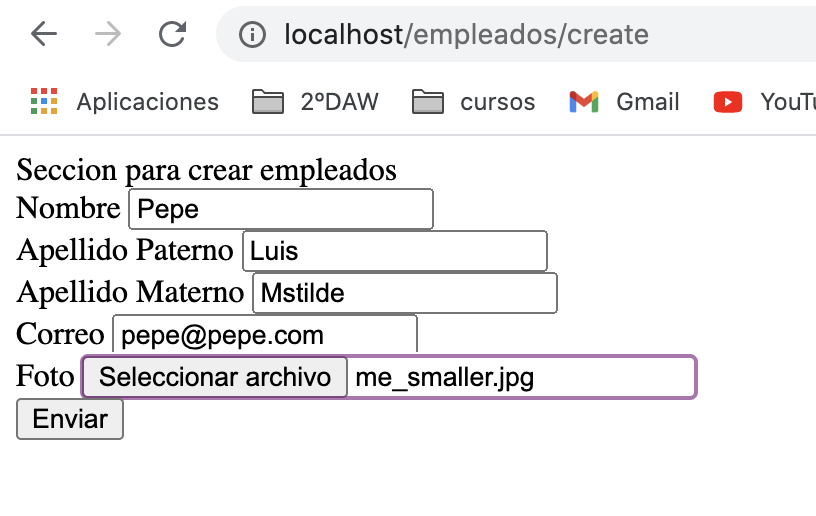
**$datosEmpleado=request()->except(‘\_token’)** Para que nos guarde toda la información de la petición menos el **token**

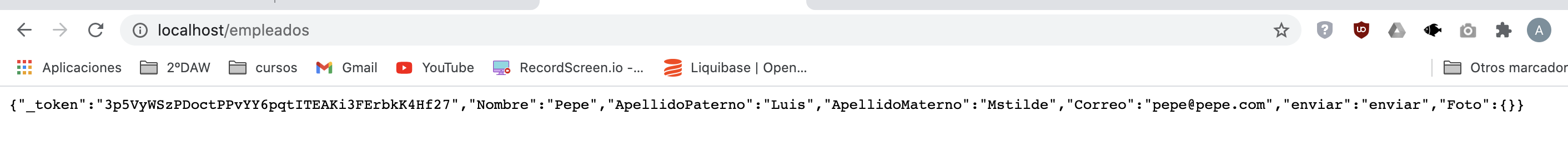
****

Para insertar los datos llamamos al modelo **Emleados::insert($array)**

****

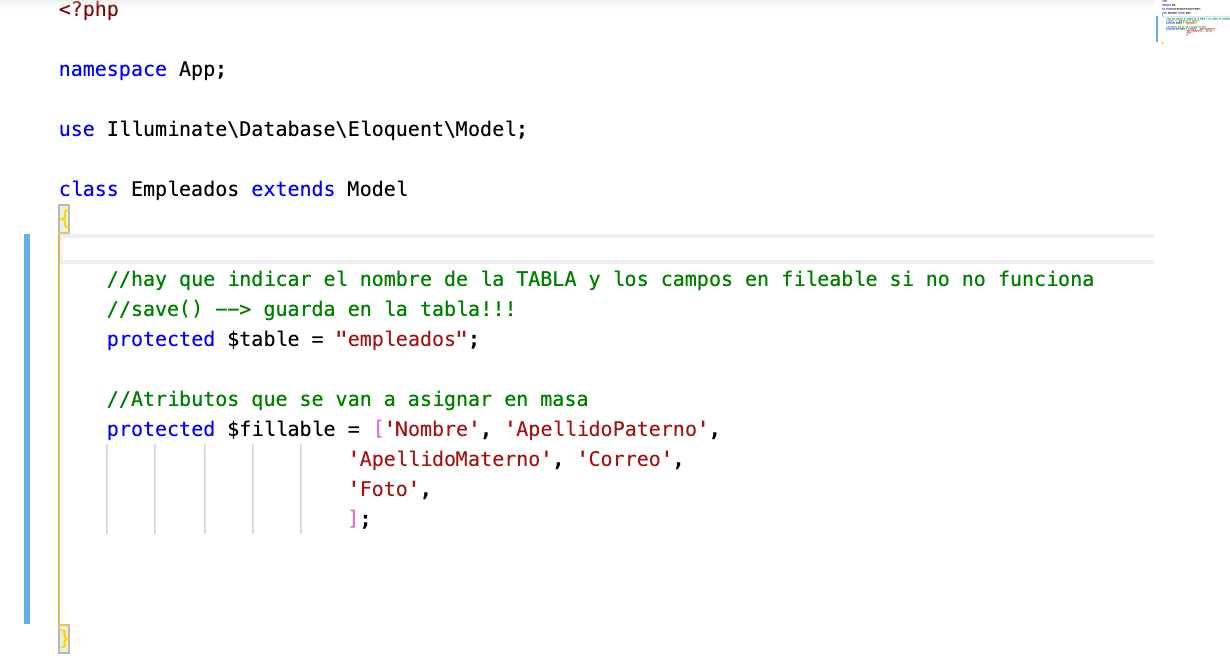
Hacemos el POST y comprobamos si se envían los datos correctamente



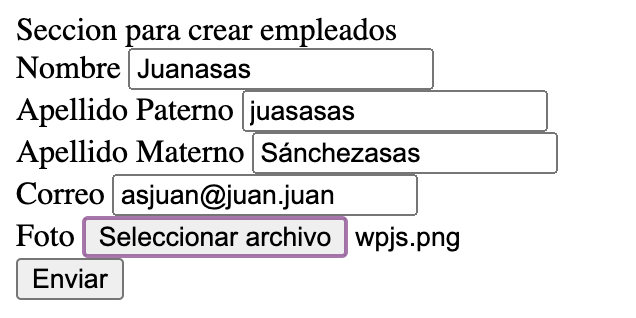


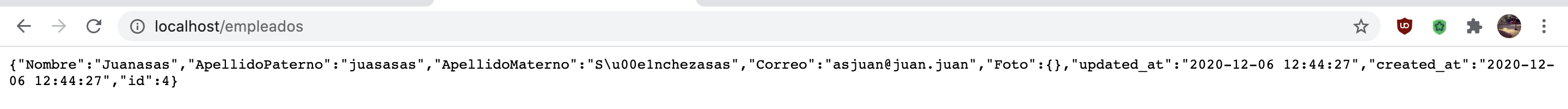
Me fallaba este método. Así que lo hice de la siguiente manera.

En el **modelo** de empleado añadí **la tabla protegida** así como los atributos **protegidos en fileable para poder asignarle valores y hacer los inserts.**

****

**Y el controlador es parecido pero diferente:**





Vemos que nos devuelve todos los datos de la base de datos. Incluidos incluso los **timesStamps**

En el controlador he incluido lo siguiente.

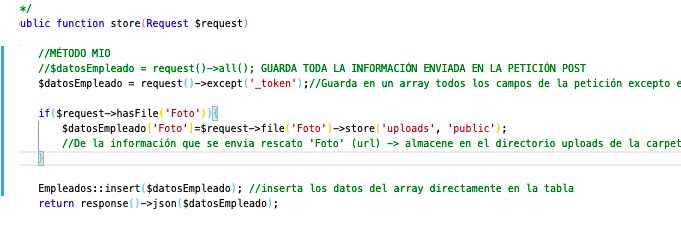
return $users->toArray();

return $user->toJson();

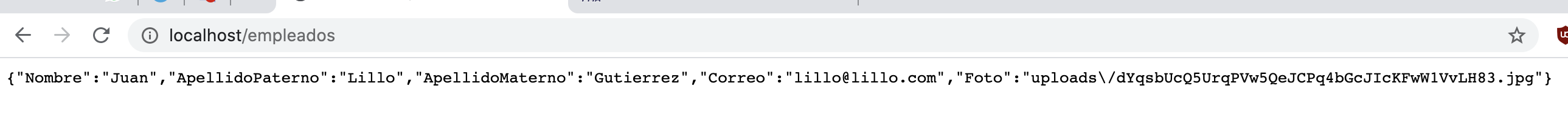
TODO revisar lo anterior.

**¿Y la fotografía?**

En el controlador Le indicamos antes de hacer el insert con un condicional si en el $\_POST[‘foto’] hay una tipo **file** que recoga este archivo y que lo almacene en **uploads** del directorio **public.**

****

Si ejecutamos esto se nos ingresarán los datos y además la fotografía se subirá a **uploads** en **/storage/app/public**

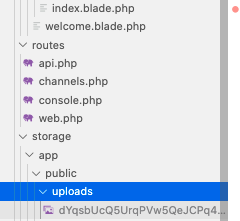


En la respuesta **json**  ya indica un valor diferente para la foto en este caso aparece **uploads\/12123132132121.jpg**

Directorio el nombre que le ha dado a la foto (Supongo que por seguridad).

¿Dónde se envio la imágen?

En la ruta de la aplicación hay un directorio llamado **storage** dentro un directorio llamado **app** y dentro de este un directorio llamado **public**

****

MOSTRAR LOS DATOS EN INDEX

Vamos al controlador de empleados método **index.**

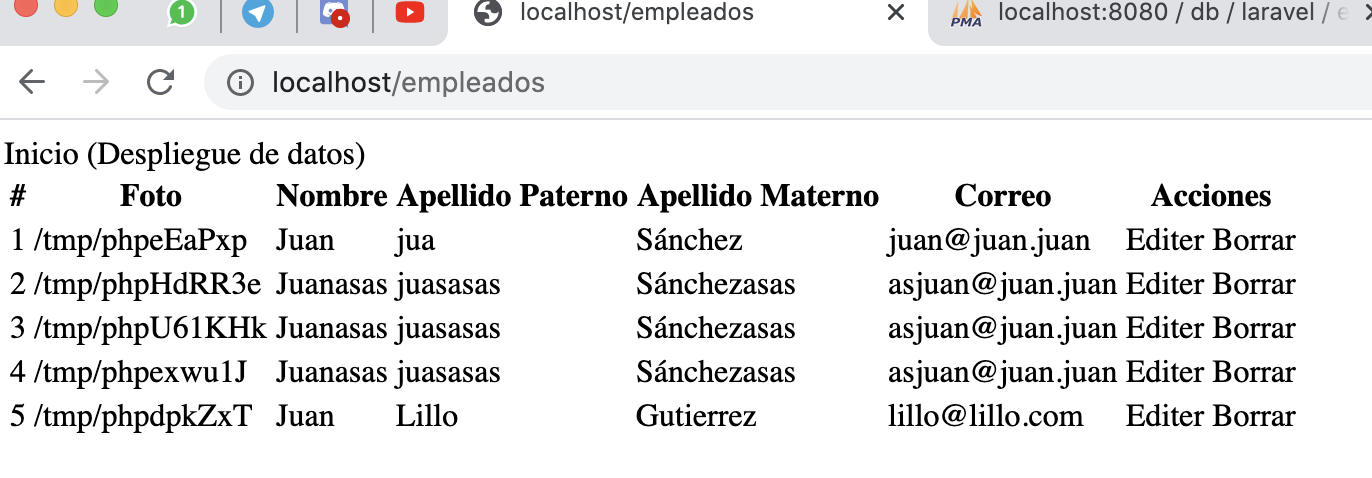
Creo un **Array** con los datos del empleados **paginando de 5 en 5 los datos**

Y se los paso al **controlador** para mostrarlo en formato **tabla usando un forEach**

****

Si vamos ahora al index **localhost/empleados**

Debería mostrarse la información de los empleaos.

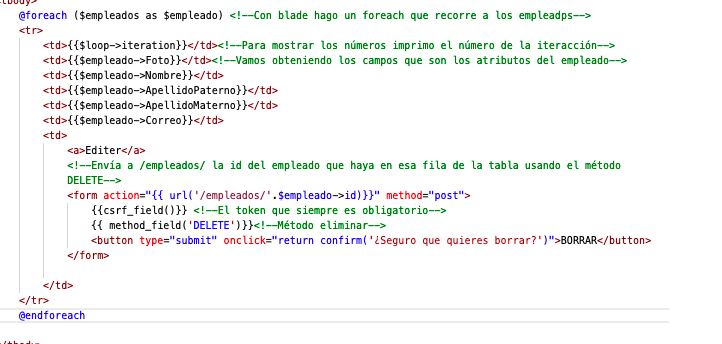


borrar

Si vemos la tabla de **rutas** para eliminar se usa el **método DESTROY** DEL CONTROLADOR con la ruta /empleados{**con un empleado**}/ USANDO EL MÉTODO DELETE

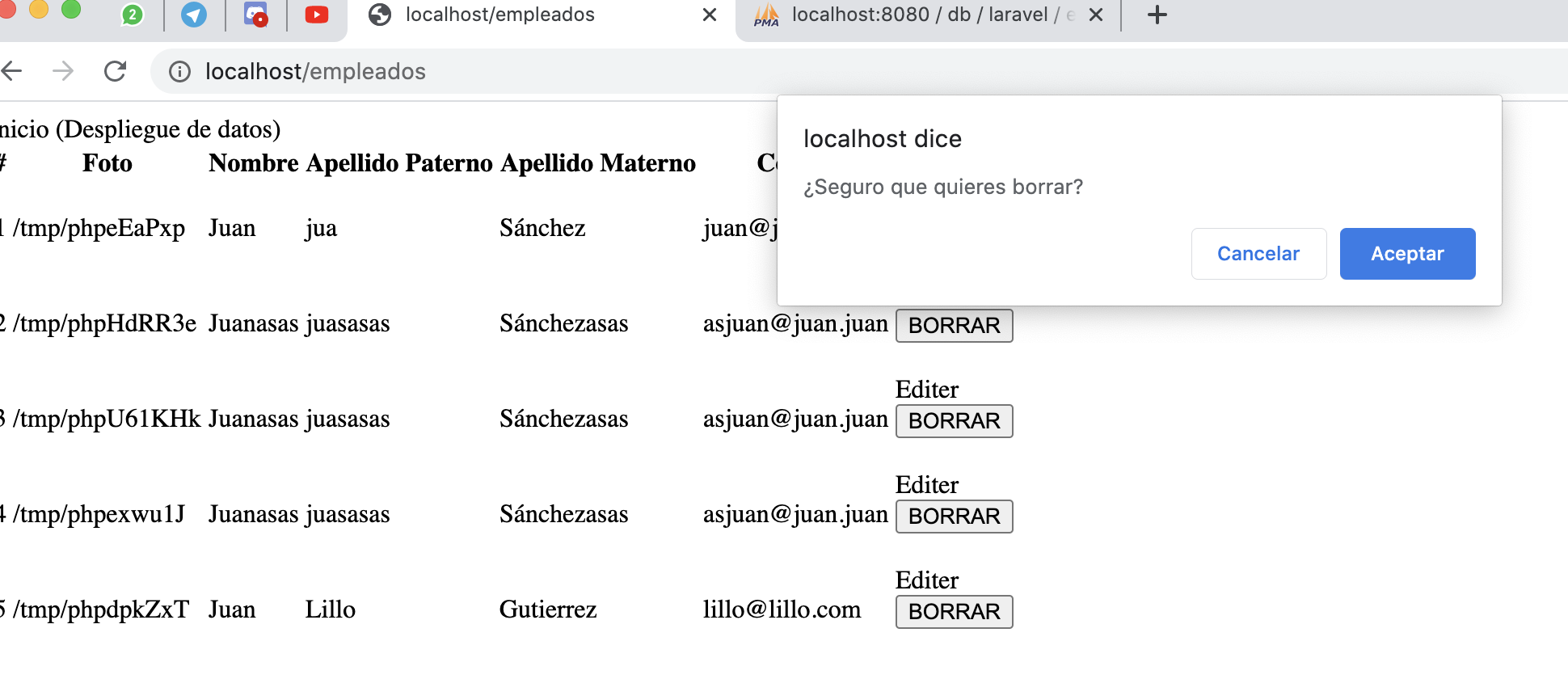
Para ello es necesario hacer un **formulario** para **el botón de borrar**. Que envíe por **POST** a esa dirección con el método **DELETE.**

(Además incluyo un **confirm()** para asegurar que se desea eliminar el registro).

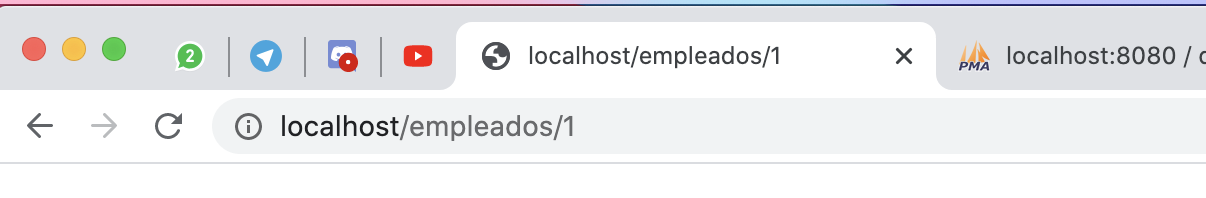


Si se hace click en **borrar** preguntará



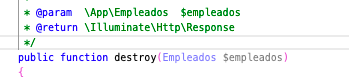


Si le damos ha aceptar nos envía a una url con **el empleado/id.** Pero no hace nada

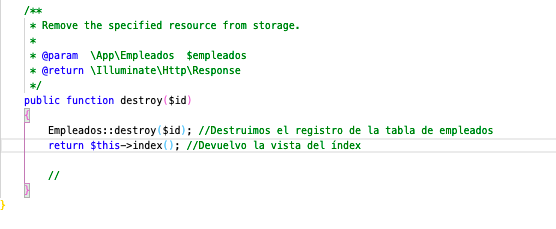


Para controlar esto nos vamos al **controlador de empleados método destroy**

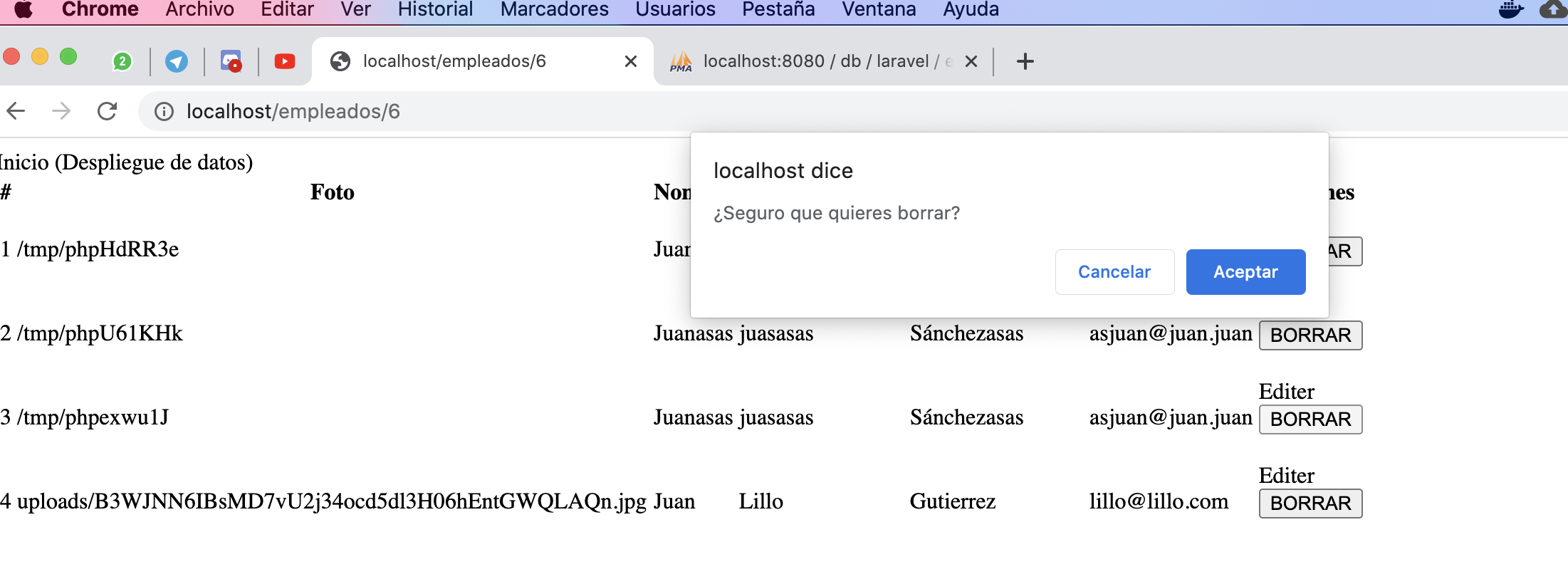
Y ejecutamos una sentencia para eliminar el empleado de la base de datos que tenga este id que estamos enviando y llame al controlador del index para que vuelva a cargar los empleados en un Array y llame a la vista del index.

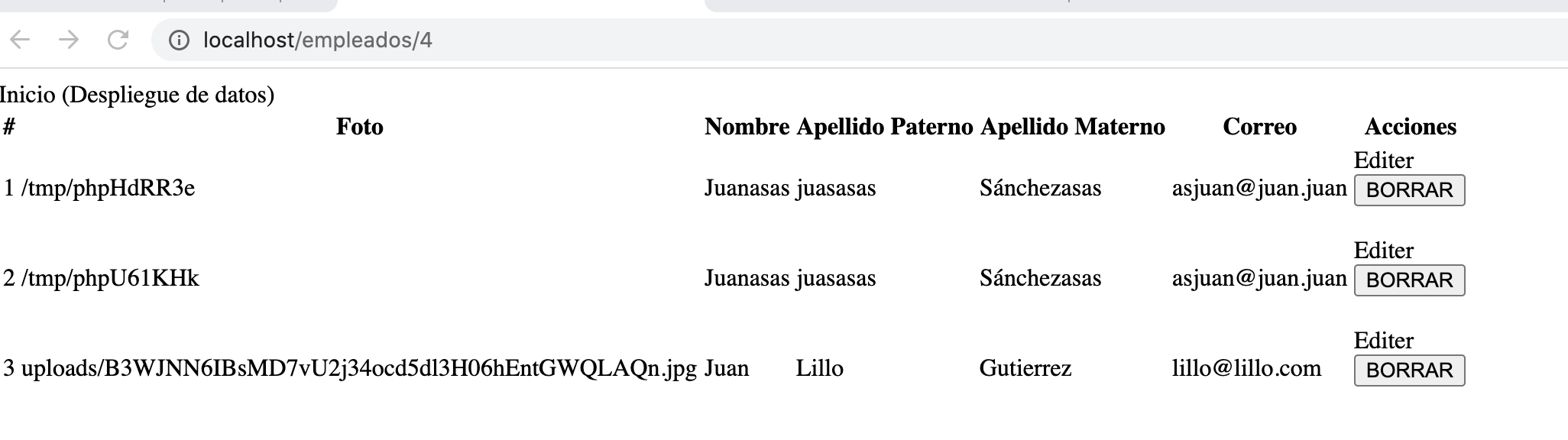


Como estamos recibiendo solo un **id** cambiamos **$empleados** por un **$id** elimiar el registro que lleve **$id** y devuelve **index**

****

Comprobamos que funciona correctamente:





EDITAR ENLACE

Vemos que para **EDITAR** LARAVEL utiliza **el método GET**

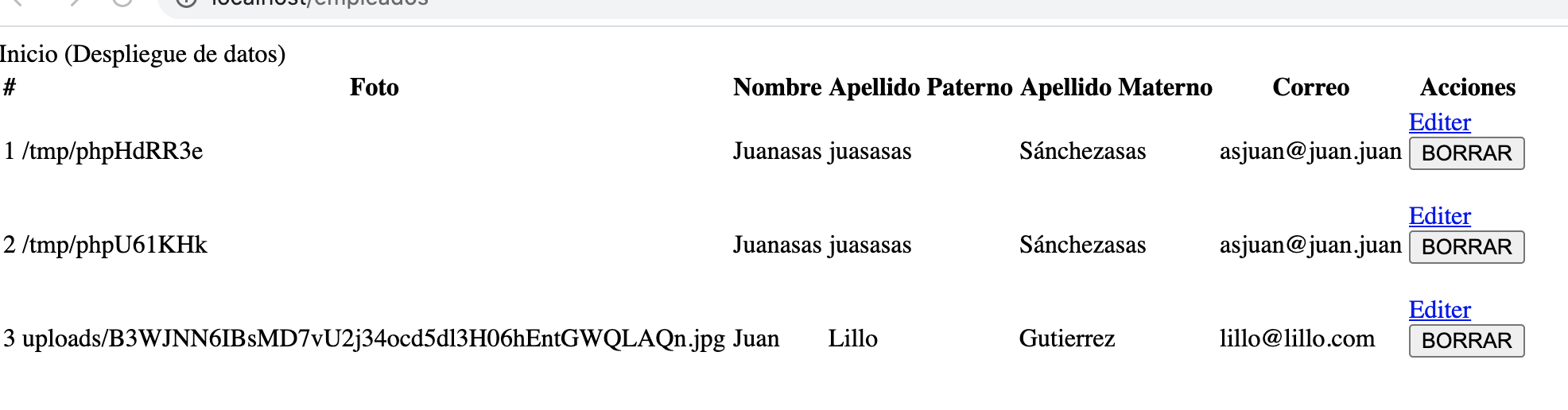


Creamos un enlace que envíe a **empleados/el empleado/edit** 🡪 Es lo que nos indica su ruta.



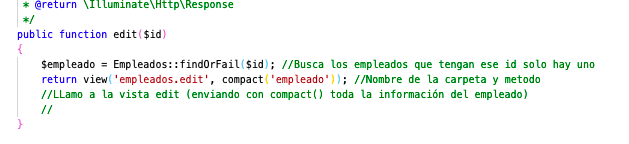


Enviamos el **id empleado al controlador de id**



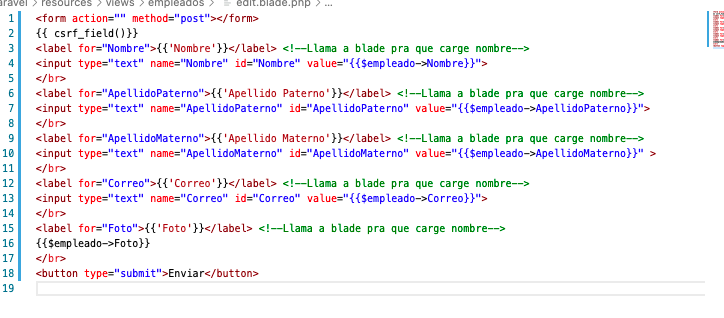
Ya aparece el botón **en forma de enlace.**

Vamos al método del controlador **edit**. Aquí seleccionamos el empleado que vamos a **editar** y lo enviamos a la vista **edit**

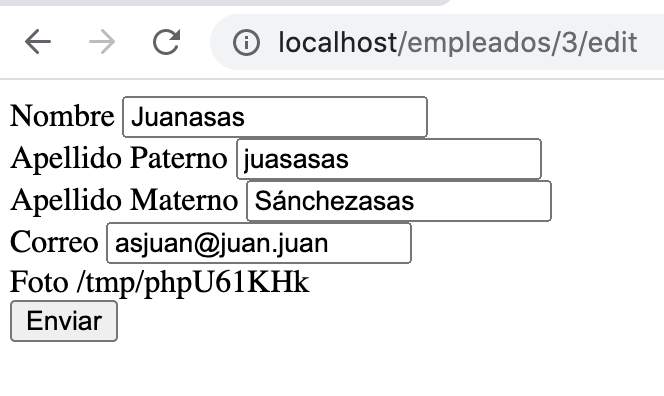
****

A **continuación en la vista incluimos el formulario pero con los valores que enviamos al cargar la vista.**

(AÚN NO ESTOY TRABAJANDO CON LA IMÁGEN)



**Ahora cuando hago click en Edit** -> Nos carga el formulario con los datos del empleado



GUARDAR EDIT EN LA BASE DE DATOS USANDO EL CONTROLADOR DE UPDATE()

Ya está funcionando el método del controlador **edit()** que es el que nos carga el formulario **para actualizar** vamos a usar el método **update()** del controlador.



Enviaremos los datos del formulario usando el método **PATCH.**

**CLASE 11**

Al igual que al ahcer DELETE enviábamos la instrucción blade {{method\_field(‘DELETE’)}}

En este caso vamos a enviar el método **PATCH**. De la misma manera:

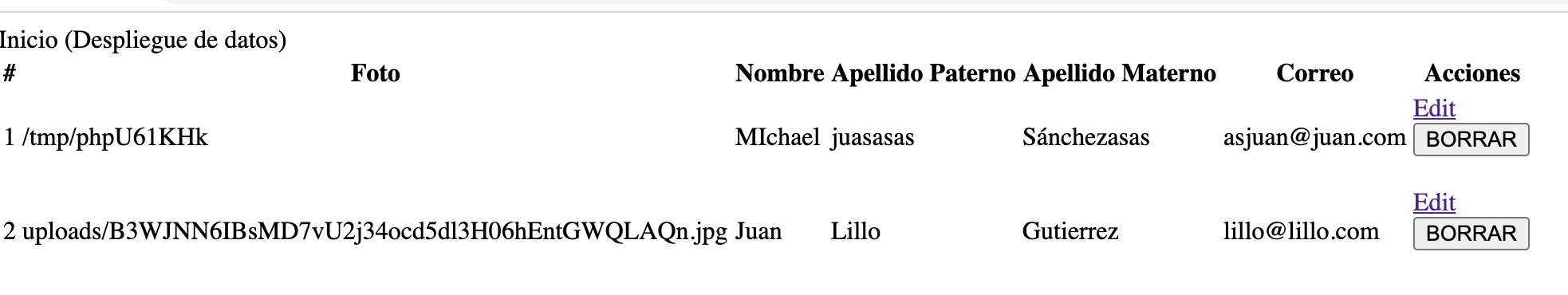


La url () será url(‘/empleados/’.$empleado.id) Enviamos por POST usando PATH al controlador **update** donde haremos el update a partir del id.



A continuación vamos al método update() del controlador. Aquí necesitamos recibir además de todos los valores enviados por post (***“menos el token y el method”***) Necesitamos el id para hacer la actualización. Así pués vamos









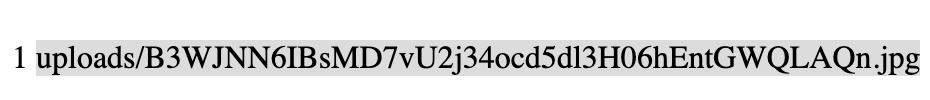
MOSTRAR IMÁGENES

Para esta labor vamos a echarle mano a:

**(PHP ARTISAN STORAGE LINK)**

Lo primero que vamos ha hacer es ir a la **vista del índex**.

Donde en la tabla en la posición e la fotografía incluimos un **img** con la url de la imagen. LA RUTA DE LA IMAGEN ES EL ATRIBUTO Foto de empleado



Quedando así el **índex**

****

A la ruta además de **empleados->Foto** hay que añadirle **asset(‘storage’).’/’.empleados->Foto**

**Asset () -> Permite un recurso y** con ‘/’’ que concatena terminamos de indicar la ruta.

Una vez echo esto las imágenes no se ven. Esto es debido a que laravel tiene quepublcar el **storage** para que se pueda acceder a sus recursos (además del **assets**)



Acabamos de hacer un link con storage. Tnemos el mismo directorio que estaba en **storage/app/public/uploads**

**En public/storage**

