Universidad Autónoma de San Luis Potosí
Facultad de Ingeniería

Manual del programador de la aplicación web *BetCalories*

Hecho por: Martínez García Alberto Enrique

Materia: Aplicaciones Web Interactivas

Semestre: 2020/2021 II

Profesor: Francisco Everardo Estrada Velázquez

Índice

Tabla de contenido

1. Introducción

En este documento se describe el proceso para la instalación del proyecto, los requerimientos para la misma, las configuraciones necesarias, una breve explicación de los servicios que se utilizaron, la estructura del proyecto, así como una explicación detallada de la parte de la programación.

2. Inicialización

2.1. Requerimientos

- Se recomienda utilizar el programa Laragon para facilitar el manejo de los servicios utilizados.
- Laravel versión 7.x.
- PHP versión 7 o superior.
- MySQL versión 5.7.24.
- Node versión 12 o superior.
- Se utiliza Git como controlador de versiones.

2.2. Descarga

Para facilitar la descarga del proyecto se recomienda hacer uso de una terminal en la cual se puedan ejecutar comandos de Git. Los pasos necesarios para la descarga son los siguientes:

- Dirigirse a la carpeta donde desea guardar el proyecto.
- Utilizar el comando *git clone* para poder clonar el repositorio de GitHub desde la siguiente liga: *https://github.com/Appwebuaslp/proyecto-AlbertoMartinezGar.git*
- Hacer una copia de un archivo *".env*" y colocarla en la carpeta raíz del proyecto.
- Una vez que el archivo mencionado anteriormente esté en la carpeta se deberá cambiar la línea *DB_DATABASE* y poner el nombre de una base

de datos que se haya designado para el proyecto.

2.3. Instalación

Para poder hacer un uso correcto del proyecto y que este no presente errores al momento de estar usándolo es necesario ejecutar algunos comandos para incorporar todos los servicios utilizados. Para esto lo primero que se debe de hacer es ir a la carpeta raíz del proyecto, una vez estando allí se debe de abrir una terminal y ejecutar los siguientes comandos:

- composer install
- npm install
- composer require laravel/ui:^2.4
- php artisan ui vue –auth
- composer require spatie/laravel-permission
- php artisan vendor:publish --provider="Spatie\Permission\PermissionServiceProvider"
- php artisan optimize:clear
- composer require barryvdh/laravel-dompdf

Una vez ejecutados estos comandos el proyecto esta listo para ser utilizado.

2.4. Visualización

Para lograr visualizar el proyecto es necesario abrir la ventana principal de Laragon, dar click en el botón de *Start all*, dar click derecho en cualquier parte de la ventana, poner el cursor sobre la opción *www* y dar click en la opción *proyecto-AlbertoMartinezGar*, esto abrirá el proyecto en una nueva pestaña en el navegador en la cual estará el proyecto.

3. Servicios

Para la realización del proyecto fueron utilizados distintos servicios, a continuación, se presenta una descripción breve de cada uno de ellos.

3.1. Laravel

Laravel es un framework de aplicaciones web basado en PHP, es de tipo MVC (Modelo-Vista-Controlador). El backend del proyecto y algunas partes del frontend están construidas con él.

3.2. Bootstrap

Bootstrap es un framework front-end utilizado para desarrollar aplicaciones web y sitios mobile first, o sea, con un layout que se adapta a la pantalla del dispositivo utilizado por el usuario.

3.3. Laravel-permission de Spatie

Laravel-permission es un paquete que nos permite integrar la función de roles y permisos en un sistema. Nos proporciona el poder de otorgar o quitar acceso a un módulo del sistema a cualquier usuario.

3.4. DOMPFD

DOMPDF es un motor de diseño y renderizado HTML escrito en PHP. En otras palabras, es un conversor de HTML a PDF.

4. Estructura del proyecto

4.1. Base de datos

La estructura principal de la base de datos se observa en la siguiente imagen:

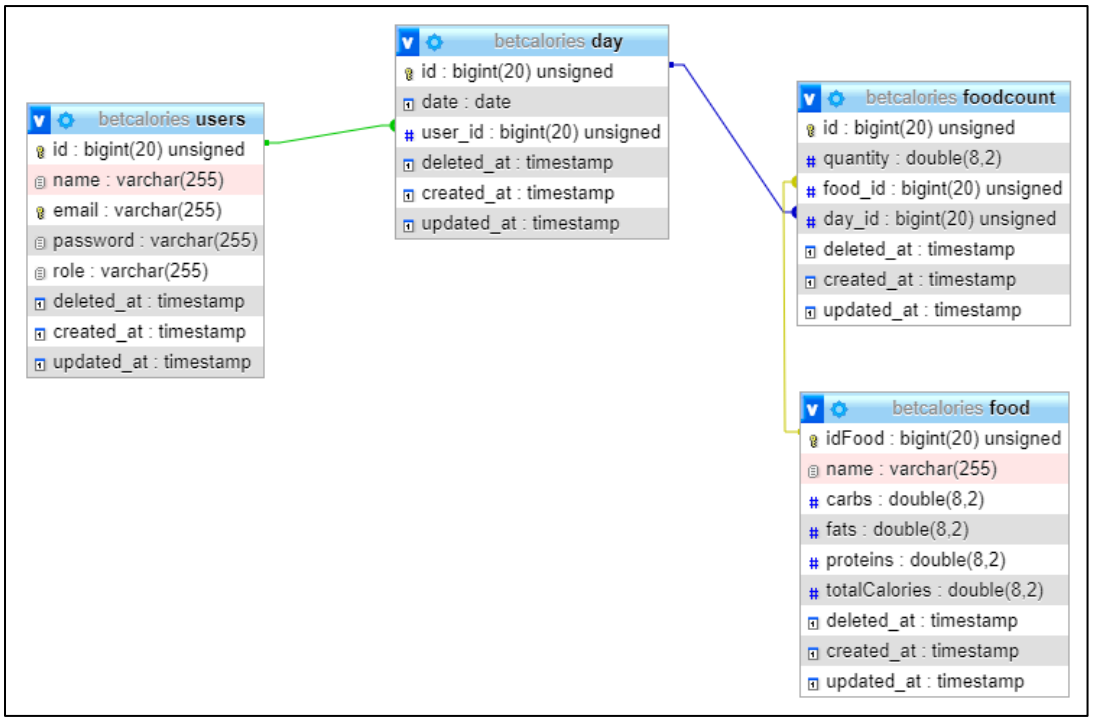

Ilustración 1- Base de datos del sistema

4.1.1 Tabla Users

Esta tabla es la encargada de almacenar la información del usuario.

Campo de la tabla	Tipo	Descripción
name	string	Nombre del usuario.
email	string	Correo electrónico del usuario.
password	string	Contraseña de la cuenta del usuario.
role	string	Rol que tendrá el usuario dentro del sistema.

4.1.2 Tabla Day

Esta tabla es la encargada de almacenar la relación entre el usuario y la cantidad ingerida de un alimento especifico.

Campo de la tabla	Tipo	Descripción
date	date	Día en el que se registran los alimentos.
user_id	unsignedBigInteger	Llave foránea para hacer la relación con el usuario.

4.1.3 Tabla foodcount

Esta tabla es la encargada de almacenar la relación entre el día que se ingirió un alimento y el alimento ingerido.

Campo de la tabla	Tipo	Descripción
quantity	float	Cantidad ingerida de un alimento.
food_id	unsignedBigInteger	Llave foránea para hacer la relación con el alimento.
day_id	unsignedBigInteger	Llave foránea para hacer la relación con el día.

4.1.4 Tabla food

:

Esta tabla es la encargada de almacenar la información individual de cada alimento.

Campo de la tabla	Tipo	Descripción
name	float	Nombre del alimento.
carbs	float	Cantidad de carbohidratos del alimento.

fats	float	Cantidad de grasas del alimento.
proteins	float	Cantidad de proteínas del alimento.
totalCalories	float	Cantidad de calorías totales del alimento.

4.2. Rutas del sistema

En esta sección se muestra el nombre de todas las rutas del sistema, así como una breve descripción de su funcionalidad. Las rutas utilizadas después de realizar el login están divididas por el tipo de usuario.

4.2.1 User - Auth

Estas rutas son generadas automáticamente por los comandos *composer require laravel/ui:^2.4* y *php artisan ui vue –auth.* Las rutas son:

Nombre de la ruta	**Función**
`POST: /login`	Iniciar sesión.
`POST: /logout`	Cerrar sesión.

4.2.2 Administrador

Todas estas rutas se encuentran dentro del siguiente grupo, esto para que solamente el administrador tenga acceso a ellas:

```
Route::group(['middleware' => ['role:admin']], function
() {
     /*Rutas del admin*/
});
```

Nombre de la ruta	Función
`GET: /registerfood`	Obtiene la vista para ver todos alimentos y para agregar uno nuevo.
`POST: /guardaalimento`	Registra un nuevo alimento en la base de datos.
`POST: /borraralimento/{id}`	Borra un alimento de la base de datos basándose en el ID de éste.
`GET: /editaralimento/{id}`	Obtiene la vista para poder editar los valores de un alimento.
`POST: /editaralimento/{id}`	Edita los valores de un alimento en la base de datos.
`GET: /homeadmin`	Muestra la vista de la pantalla de inicio del administrador.

4.2.3 Usuario autenticado

Todas estas rutas se encuentran dentro del siguiente grupo, esto para que solamente un usuario autenticado tenga acceso a ellas:

```
Route::group(['middleware' => ['role:user']], function
() {
     /*Rutas del usuario*/
});
```

Nombre de la ruta	Función
`GET: /addfood/{date}`	Obtiene la vista para que el usuario vea todos los alimentos en la base de datos y pueda agregar alguno. Recibe la fecha del día al cual se agregará el alimento.
`POST: /addfood/{date}`	Obtiene la vista para que el usuario

	vea los alimentos que coinciden con su búsqueda.
`POST: /savefood/{id}`	Registra un alimento en el día en el que el usuario se encuentra. Recibe como parámetro el ID del alimento a registrar.
`GET: /mycalories/{date}`	Obtiene la vista para que el usuario vea un recuento de sus calorías consumidas ese día, así como de sus alimentos registrados ese mismo día. Recibe como parámetro el día a visualizar.
`POST: /mycalories/{date}`	Obtiene la vista para que el usuario vea solo los alimentos que coincidan con su búsqueda, el conteo de calorías se limita a los alimentos mostrados.
`POST: /deletefood/{id}`	Elimina el registro de un alimento en un día especifico. Recibe como parámetro el ID del alimento a eliminar.
`POST: /getreport`	Obtiene un reporte PDF para que el usuario vea un conteo de sus calorías en los últimos 3 días.

4.3 Vistas

Las vistas son archivos con extensión "*.blade.php*" y básicamente son nuestro frontend, es decir, lo que ve el usuario final.

A continuación, se muestra un listado de las vistas existentes dentro del proyecto,

así como una pequeña descripción.

Nombre de la vista	Descripción
login	Muestra el formulario de inicio de sesión.
register	Muestra el formulario para que un usuario nuevo se registre en el sistema.
welcome	Página principal del sistema. Muestra una breve descripción de lo que hace el sistema, así como los datos del programador.
Vistas del usuario autenticado	
addfood	Muestra el catálogo completo de los alimentos para que un usuario agregue una comida a su día. Cuenta con un buscador de alimentos.
home	Muestra la página principal para el usuario. Tiene dos opciones, registrar un alimento en el día y obtener un reporte.
recuento	En esta página se muestra un recuento de las calorías que el usuario ha consumido durante el día. También se muestran los alimentos consumidos, tiene la opción de poder eliminarlos de su día. Cuenta con un buscador y con una navegación entre días.
report	Esta vista es para darle el formato al reporte obtenido cuando el usuario lo genere.
Vistas del administrador	
editfood	Muestra los datos actuales de un alimento para que, en caso de ser requeridos, estos se puedan actualizar.
homeadmin	Muestra la página principal del administrador con la opción de ver un listado de todos los alimentos.
registerfood	Muestra un catálogo de todos los alimentos registrados en la base de datos, con las opciones de

	eliminar o editar cada uno individualmente, además tiene la opción de agregar un nuevo alimento a la base de datos.

5. Programación

5.1. Migraciones

Las migraciones son la forma en que creamos nuestra base de datos desde Laravel. Para este proyecto solo fueron usadas 4 tablas, por lo que solo se crearon 4 migraciones. Para utilizarlas hacemos uso de *Schema,* todas las migraciones se crean en la función *up()*, creada automáticamente por artisan al ejecutar el comando *php artisan make:migration nombre.* En ellas indicamos como queremos que se llame nuestra tabla en la base de datos y después los datos que ésta contendrá con la notación *$table->tipoDeDato('nombreDeLaVariable)*.
A continuación, se presentan las migraciones a detalle.

5.1.1 create_food_table

Describe los elementos de la tabla **food**.

```
Schema::create('food', function (Blueprint $table) {
      $table->bigIncrements('idFood');
      $table->string('name');
      $table->float('carbs');
      $table->float('fats');
      $table->float('proteins');
      $table->float('totalCalories');
      $table->softDeletes();
      $table->timestamps();
});
```

5.1.2 create_users_table

Describe los elementos de la tabla **users**.

```
Schema::create('users', function (Blueprint $table) {
     $table->bigIncrements('id');
     $table->string('name');
     $table->string('email')->unique();
     $table->string('password');
     $table->string('role')->nullable();
     $table->softDeletes();
     $table->timestamps();
 });
```

5.1.3 create_day_table

Describe los elementos de la tabla **day**.

```
Schema::create('day', function (Blueprint $table) {
     $table->bigIncrements('id');
     $table->date('date')->nullable();
     //Relación con el alimento
     $table->unsignedBigInteger('user_id')->nullable();
     $table->foreign('user_id')
          ->references('id')
          ->on('users')->nullable();
     $table->softDeletes();
     $table->timestamps();
});
```

5.1.4 create_foodcount_table

Describe los elementos de la tabla **foodcount**.

```
Schema::create('foodcount', function (Blueprint $table)
{
     $table->bigIncrements('id');
     $table->float('quantity');
```

```
            //Relación con el alimento
            $table->unsignedBigInteger('food_id')->nullable();
            $table->foreign('food_id')
                  ->references('idFood')
                  ->on('food')->nullable();

            //Relación con el dia
            $table->unsignedBigInteger('day_id')->nullable();
            $table->foreign('day_id')
                  ->references('id')
                  ->on('day')->nullable();
            $table->softDeletes();
            $table->timestamps();
      });
```

5.2. Seeders

Los seeders son la manera en la que introducimos datos de prueba a nuestra base de datos, son muy útiles para no tener que introducir los datos que queramos visualizar directamente desde la aplicación.

Para el proyecto se definieron seeders para cada tabla que se utiliza, cada una con un número diferente de campos. Para su implementación se utilizaron los modelos de cada tabla, los cuales se definen en la siguiente sección.

A continuación, se presentan los seeders a detalle.

5.2.1. DaySeeder

Para la tabla de día se definieron 3 días, los cuales son: el día de hoy, el día de ayer y el día de antier, que son los 3 días que se utilizan para la generación de reportes. De igual manera, a los días se les asignó el *user_id* con un valor de 2, que es el usuario autenticado.

```
$actualDate = date("Y-m-d");
$dateMenos1 = date("Y-m-d", strtotime($actualDate."- 1
```

```
days"));
$dateMenos2 = date("Y-m-d", strtotime($actualDate."- 2
days"));

Day::create([
     'date' => $actualDate,
     'user_id' => '2'
]);

Day::create([
     'date' => $dateMenos1,
     'user_id' => '2'
]);

Day::create([
     'date' => $dateMenos2,
     'user_id' => '2'
]);
```

5.2.2. FoodCountSeeder

Este seeder fue utilizado para que el usuario tuviera por lo menos 3 alimentos en sus 3 días mostrados en la sección anterior, asignando una cantidad para cada alimento. La manera de crearlos fue la siguiente:

```
FoodTo::create([
     'quantity' => '45.2',
     'food_id' => '2',
     'day_id' => '1'
]);
```

En total fueron creadas 9 instancias, todas de la misma manera, el único cambio que hubo entre ellas fue la cantidad que les fue asignada, el alimento asignado y el día al que correspondería dicho alimento.

5.2.3. FoodSeeder

Para la tabla de comidas se crearon 10 instancias diferentes, todas de la misma manera, lo único que cambió entre ellas es el nombre del alimento, los otros 4 campos (carbs, fats, proteins y totalCalories) fueron asignados de manera aleatoria.

```
Food::create([
     'name' => 'Frijoles pintos',
     'carbs' => rand(10, 80),
     'fats' => rand(10, 80),
     'proteins' => rand(10, 80),
     'totalCalories' => rand(100, 300)
]);
```

5.2.4. UserSeeder

Para los usuarios se crearon solamente dos instancias, un usuario administrador y un usuario autenticado. Ademas, en este seeder se hizo uso de *laravel-permission* de Spatie para crear y asignar roles roles y de esta forma tener una mayor seguridad dentro del sistema. La manera en que se implementó este seeder es la siguiente:

```
$role = Role::create(['name' => 'admin']);
$role = Role::create(['name' => 'user']);

//Usuario administrador
User::create([
     'name' => 'admin',
     'email' => 'adminprueba@gmail.mx',
     'password' => bcrypt('admin123'),
     'role' => 'admin'
])->assignRole('admin');

//Usuario autenticado
```

```
User::create([
     'name' => 'beto',
     'email' => 'beto@gmail.mx',
     'password' => bcrypt('betotas123'),
     'role' => 'autenticado'
])->assignRole('user');
```

5.3. Modelos

En esta parte, los modelos realmente no fueron modificados para realizar funciones extras, fueron creados para poder utilizarlos en la parte del CRUD del proyecto y poder hacerlo de una manera mas sencilla. La definición de cada uno de los modelos es la siguiente.

5.3.1. Day

```
use SoftDeletes;
protected $table="day";
protected $fillable=[
     'day',
];
}
```

5.3.2. Food

```
use SoftDeletes;
protected $table="food";
protected $fillable=[
     'name',
     'carbs',
     'proteins',
     'fats'
];
```

5.3.3. FoodTo

```
use SoftDeletes;
protected $table="foodcount";
protected $fillable=[
     'quantity',
     'food_id',
```

```
        'foodTo_id'
    ];
```

5.3.4. User

Únicamente el modelo *User* fue creado por Laravel automáticamente al momento de ejecutar los comandos necesarios para la autenticación, no se implementó nada adicional al momento de utilizarlo.

5.4. Controladores

En esta sección se detallarán cada una de las funciones utilizadas en cada controlador. Se mostrará la función y enseguida una pequeña descripción de su funcionamiento.

Para facilitar el manejo de los controladores y para tener un código mejor estructurado y organizado se optó por separar los controladores dependiendo del usuario por lo que se tienen dos controladores propios y un controlador creado automáticamente por Laravel momento de ejecutar los comandos necesarios para la autenticación.

5.4.1. HomeController

Este es el controlador generado automáticamente se modificó únicamente la función *index* para que redigiera automaticante al usuario a su página de inicio dependiendo de que usuario fue el que inició sesión (admin o usuario normal).

```
public function index()
{
    if(Auth::user()->role == 'admin'){
        return view('/homeadmin');
    }
    else{
        return view('/home');
    }
```

```
}
```

5.4.2. GeneralController

Este controlador fue utilizado para manejar las funciones por parte del administrador. A continuación, se muestran todas sus funciones seguidas de una breve descripción.

```
public function registerFood(){
     $food = Food::all();
     return view('registerFood')->with('foods', $food);
}
```

Obtiene todas las comidas registradas y retorna la vista *registerFood* con todas las comidas previamente obtenidas.

```
public function savefood(Request $request){
     $food= new Food();
     $food->name = request("txtAlimento");
     $food->totalCalories = request("txtTotalCal");
     $food->carbs = request("txtCarbos");
     $food->proteins = request("txtProte");
     $food->fats = request("txtGrasas");

     $food->save();

     return redirect("/registerfood");
}
```

Registra un nuevo alimento en la base de datos, recibe como parámetro un Request que son los datos provenientes de un formulario. Primero crea una nueva instancia haciendo uso del modelo Food, después asigna los valores correspondientes y al final lo guarda. Nos redirige a la ruta *registerfood.*

```
public function deleteFood($id){
```

```
    $comida = Food::where('idFood', '=', $id);
    $comida->delete();
    return redirect("/registerfood");
}
```

Elimina un alimento registrado en la base de datos, recibe como parámetro el ID del alimento a eliminar. Busca el alimento en la base de datos con una consulta SQL, después, una vez obtenido el alimento, lo elimina. Redirige a la ruta *registerfood.*

```
public function getFood($id){
    $comida = Food::where('idFood', '=', $id)->first();
    return view('editFood')->with('food', $comida);
}
```

Obtiene un alimento especifico. Recibe como parámetro el ID del alimento solicitado. Busca el alimento en la base de datos con el ID y nos devuelve el primer resultado. Retorna la vista *editfood* con el alimento solicitado como una variable.

```
public function saveEditFood(Request $request){
    Food::where('idFood', request("txtID"))-
>update(array(
          'name' => request("txtAlimento"),
          'carbs' => request("txtCarbos"),
          'proteins' => request("txtProte"),
          'fats' => request("txtGrasas"),
          'totalCalories' => request("txtTotalCal")
      ));
      return redirect("/registerfood");
}
```

Edita los valores de un alimento. Recibe como parámetro un Request con datos provenientes de un formulario. Lo que hace es buscar un elemento

en la tabla food donde el ID coincida con *textID*, una vez que lo encuentra reemplaza los datos previos por los datos que acaban de llegar. Redirige a la routa *registerfood*.

```
public function homeadmin(){
     return view('/homeadmin');
}
```

Retorna la vista *homeadmin*.

5.4.3. UserController

Este controlador fue utilizado para manejar las funciones por parte de un usuario registrado. A continuación, se muestran todas sus funciones seguidas de una breve descripción.

```
public function addFood($date){
     $food = Food::all();
     return view('addfood')->with('foods', $food)-
     >with('date', $date);
}
```

Retorna la vista *addfood*. Recibe como parámetro una fecha que es en la cual el alimento será añadido. Primero obtiene todos alimentos registrados en la base de datos, después retorna la vista *addfood* junto con las comidas obtenidas y la fecha recibida como parámetro.

```
public function search($date){
     $palabra = request("busqueda");
     if(!is_null($palabra) && $palabra != ""){
          $food = Food::where('name', 'Like',
'%'.$palabra.'%')->get();
     }
     else{
          $food = Food::all();
```

```
        }
        return view('addfood')->with('foods', $food)-
>with('date', $date);
}
```

Funciona para que el usuario busque un alimento entre todos los alimentos registrados en la base de datos. Recibe como parámetro una fecha. Lo primero que hace es asignar a una variable llamada *palabra* la palabra que se quiere buscar que es extraída de *request("búsqueda")* después hace una verificación para que la palabra no sea nula y que no sea una cadena vacía, si cumple estas condiciones busca todos los alimentos en la base de datos que tengan una coincidencia con *palabra* y las asigna en una variable llamada *food*. Si no se cumplen las dos condiciones anteriores *food* se llena con todos los alimentos de la base de datos. Al final, retorna la vista *addfood* con *food y date* como variables.

```
public function searchRegisteredFoods($date){
        $palabra = request("busqueda");
        $day = Day::where('user_id', '=',  Auth::user()-
        >id ) ->where('date', '=', $date  )->first();

        if(!is_null($palabra) && $palabra != ""){
                $foods = FoodTo::where('day_id', '=', $day-
                >id)->join('food', 'foodcount.food_id', '=',
                'food.idFood')->where('food.name', 'Like',
                '%'.$palabra.'%')->get();;
        }
        else{
                $foods = FoodTo::where('day_id', '=', $day-
                >id)->join('food', 'foodcount.food_id', '=',
                'food.idFood')->get();
       }
```

```
    return   view('recuento')->with('foods',   $foods)-
    >with('date', $date);
}
```

Parecida a la función anterior, solo que ahora busca entre los alimentos que el usuario tiene registrados en su día. Recibe como parámetro la fecha en la cual se buscará el alimento. Asigna la palabra a buscar a la variable *palabra*. Después obtiene el día en el cual se va a buscar el alimento haciendo el uso de la variable *date* recibida como parámetro. Luego, hace las comprobaciones de que *palabra* no sea nula y que no sea una cadena vacía. Si cumple estas condiciones crea una variable llamada *foods* para asignarle los alimentos resultantes. Si las condiciones no se cumplen, *foods* se obtienen todos los alimentos registrados durante el día. Retorna la vista *recuento* con el resultado de la búsqueda y la fecha como variables.

```
public function getDailyCalories($date){
     $day = Day::where('user_id', '=',  Auth::user()-
     >id )->where('date', '=', $date  )->first();

     if(!$day){
          $day = new Day();
          $day->user_id = Auth::user()->id;
          $day->date = $date;
          $day->save();
     }

     $foods = FoodTo::where('day_id', '=', $day->id)
     ->join('food', 'foodcount.food_id', '=',
     'food.idFood')->get();

     return view('recuento')->with('foods', $foods)-
```

```
        >with('date', $date);
}
```

Muestra el recuento de las calorías diarias del usuario. Recibe como parámetro *date* que es el día que se va a mostrar. Lo primero que hace es buscar ese día que llega como parámetro en todos los días registrados del usuario que está autenticado. Si el día no existe, crea un registro de la tabla *day* nuevo en la base de datos asignándole la fecha que se recibió como parámetro y el ID del usuario. Después obtiene todos los alimentos registrados en la fecha recibida como parámetros correspondientes al usuario y los guarda en la variable *foods*. Al final retorna la vista *recuento* con *foods* y con la *date* como variables.

```
public function saveFood($id){
        $date = request("fecha");
        $day = Day::where('user_id', '=', Auth::user()-
        >id)->where('date', '=', $date)->first();

        $foodCount = new FoodTo();
        $foodCount->day_id = $day->id;
        $foodCount->food_id = $id;
        $foodCount->quantity = request("cantidad");
        $foodCount->save();

        $foods = FoodTo::where('day_id', '=', "'" . $day-
        >id . "'")->join('food', 'foodcount.food_id', '=',
        'food.idFood')->get();

        return redirect("/mycalories/$date")-
        >with('foods', $foods);
}
```

Hace un registro de un alimento en un día especifico. Recibe como

parámetro el id del alimento a registrar. Guarda en la variable *date* la fecha que llega proveniente del formulario. Después busca el día correspondiente a esa fecha y que tenga el ID del usuario. Luego, crea una nueva instancia de la tabla foodcount con el modelo FoodTo, asigna los datos correspondientes y guarda el registro. Después, obtiene todos los alimentos registrados en el día *date* guardándolos en la variable *foods*. Redirige la ruta *mycalories* indicándole a que día va a ir gracias a la variable *date* con *foods* como variable.

```
public function deleteFood($id){
     $date = request("fecha");
     $food = FoodTo::find($id);
     $food->delete();
     return redirect("/mycalories/$date");
}
```

Elimina el registro de un alimento de un día especifico. Recibe como parámetro el ID del alimento a eliminar. Lo que hace es guardar en *date* la fecha que llega de un formulario. Después encuentra en la base de datos en la tabla food el alimento correspondiente al ID que recibe la función como parámetro y después la borra. Redirige la ruta *mycalories* indicándole a que día va a ir gracias a la variable *date*.

```
public function downloadReport(){
     $days = Day::where('user_id', '=', Auth::user()-
     >id)->orderBy('date', 'DESC')->take(3)->get();


     $food1 = FoodTo::where('day_id', '=', $days[0]-
     >id)->join('food', 'foodcount.food_id', '=',
     'food.idFood')->get();
```

```
        $food2 = FoodTo::where('day_id', '=', $days[1]-
        >id)->join('food', 'foodcount.food_id', '=',
        'food.idFood')->get();

        $food3 = FoodTo::where('day_id', '=', $days[2]-
        >id)->join('food', 'foodcount.food_id', '=',
        'food.idFood')->get();

        $pdf = PDF::loadView('report', compact('food1',
        'food2', 'food3'));

        return $pdf->download('reporte.pdf');
    }
```

Obtiene los alimentos necesarios para generar el reporte del usuario. Lo primero que hace es obtener los últimos 3 días del usuario, ordenándolos de manera descendente y los almacena en la variable *days*. Después, obtiene todos alimentos registrados en cada uno de esos días, guardándolos en variables independientes. Luego, carga la vista *report* al PDF para poder generarlo y al final lo descarga.

6. Referencias

- Documentación de Laravel: https://laravel.com/docs/7.x
- Bootstrap: https://getbootstrap.com/docs/4.6/getting-started/introduction/
- DOMPDF: https://github.com/barryvdh/laravel-dompdf
- Laravel-permission: https://spatie.be/docs/laravel-permission/v4/introduction