

UD-08: El patrón software MVC

Desarrollo Web en Entorno Servidor

Curso 2020/2021

El patrón MVC en el desarrollo de aplicaciones web

La ingeniería del software se dedica a buscar maneras de resolver los diferentes problemas. A lo largo de la historia de esta disciplina se han elaborado muchos y variados esquemas_resolutivos, conocidos con el nombre de patrones de diseño software. Su conocimiento y aplicación son de una inestimable ayuda a la hora de diseñar y construir una aplicación informática. Un patrón de diseño nos indica como solucionar ‘un problema’.

Posiblemente, uno de los más conocidos y utilizados sea el patrón **Modelo, Vista, Controlador** (MVC), que propone organizar una aplicación en tres partes bien diferenciadas y acopladas entre sí, de manera que los cambios que se produzcan en una no afecten demasiado a las otras (lo ideal sería que no afectaran nada).

En el patrón MVC los conceptos están separados. De esta manera cuando la aplicación crezca, podamos mantenerlos separados.

En resumen, la arquitectura MVC, es la manera en que estructuramos una aplicación, es decir, “otra forma de organizar el código”

El patrón MVC en el desarrollo de aplicaciones web

Las tres partes de patrón MVC son:

- La **Vista** → Es la interfaz. Todo usuario interactuará a través de un cliente (navegador web), y mediante “la vista” el usuario le pide “algo” al Controlador. La vista, sólo se comunica con el Controlador, nunca con el Modelo. La vista se encarga de producir documentos HTML, XML, JSON, etc., junto con los datos que se hayan calculado previamente en la aplicación a través del Controlador, quien ha obtenido el resultado a través del Modelo.
- El **Controlador** → Aquí se incluye lo referente a la lógica de control de la aplicación, que no tiene nada que ver con las características propias del negocio para el que se está construyendo la aplicación. El Controlador es el encargado de gestionar las peticiones del usuario (la vista), procesarlas invocando al Modelo (quien consulta la base de datos) y mostrarlas al usuario a través de las Vistas.
- El **Modelo** → Es donde se implementa todo lo relativo a la lógica de la aplicación, es decir, los aspectos particulares del problema que la aplicación resuelve. Es el encargado de conectar con la BD.

El patrón MVC en el desarrollo de aplicaciones web

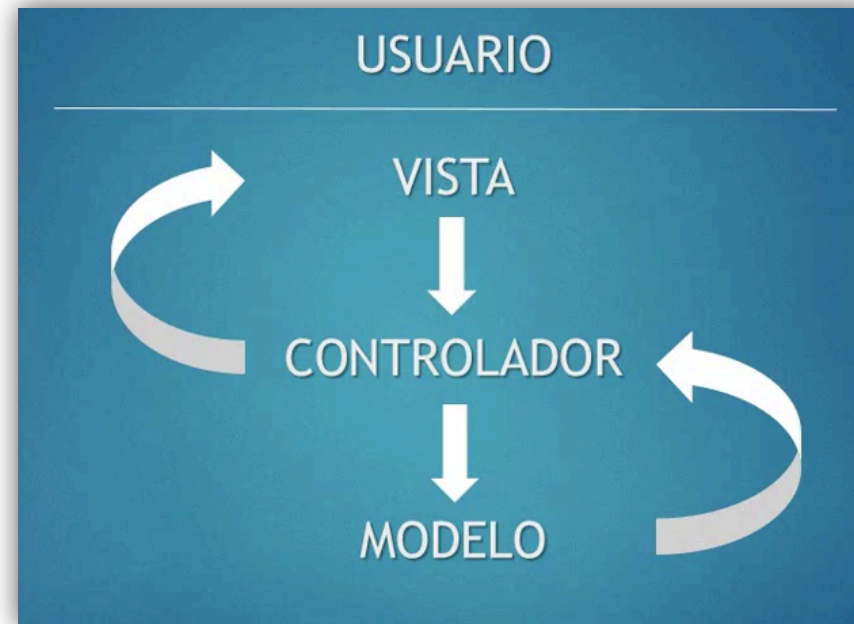
En resumen, el Controlador:

- Coge la información del usuario, a través de la Vista
- Se comunica con el Modelo, con la Base de Datos
- Recibe los datos del Modelo
- Envía los datos recibidos del Modelo a la Vista, es decir, al usuario.

Para que el conjunto funcione, las partes deben interaccionar entre sí.

Podemos encontrar en la literatura distintas soluciones.

Nuestra propuesta de MVC es la mostrada en la figura.



El patrón MVC en el desarrollo de aplicaciones web

Entrando en detalles, el usuario, a través de la Vista, hace una petición y esa petición puede pasar por un primer filtro. Es decir, puede haber una serie de validaciones, que conocemos como validación del cliente. Validaciones que podemos hacer con JavaScript, para poder mostrar al usuario que la información que está ingresando es correcta, inválida, está tratando de atacar la aplicación o está tratando de colocar información no adecuada.

Después de esa petición que hace el usuario que pasa por el primer filtro, esa información llega a un segundo filtro, a una segunda capa que es el controlador. El controlador hace la validación pero del lado del servidor. Mira si es coherente, que no es información extraña. Si es información válida, pasa el segundo filtro para dirigirse a la tercera capa que es el modelo.

El modelo decide qué hacer con esa información, es decir, si va a generar alguna interacción, si irá a la base de datos y seleccionará datos, etc.

Cuando el modelo responde, esta respuesta se devuelve al controlador y el controlador procesa la respuesta para mostrarla a través de la vista al usuario y de esta forma el usuario recibe su respuesta.

El patrón MVC en el desarrollo de aplicaciones web

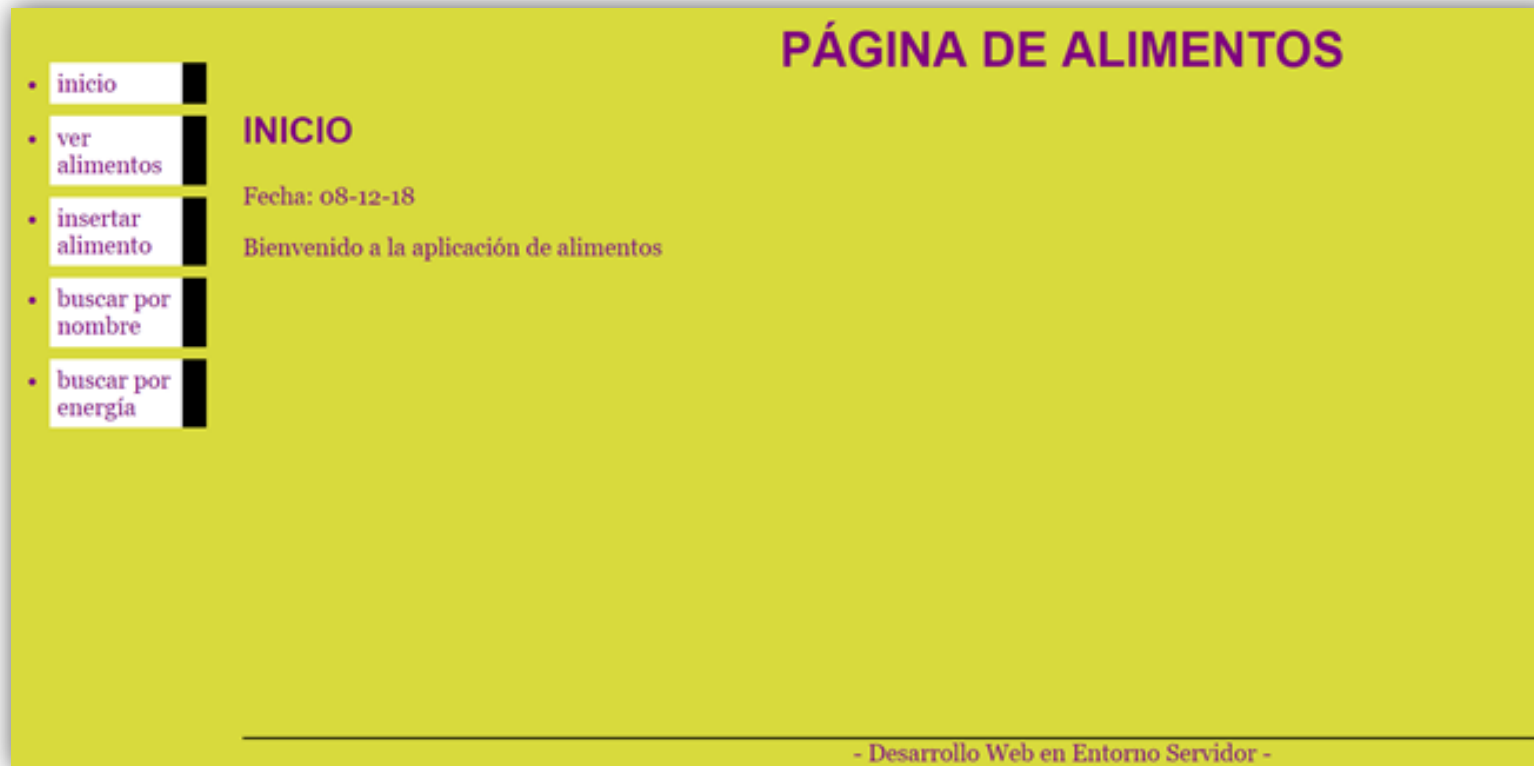
Es decir, NO es una interacción directa para llegar hasta la base de datos. La información tiene que pasar por una serie de filtros para poder llegar hasta la base de datos y así protegerla. La respuesta al usuario nuevamente, por el mismo camino pero a la inversa, llegará al usuario y éste se quedará con la vista.

El usuario, verá el resultado final solamente en la vista y no sabrá qué está sucediendo en el interior de la aplicación. Por estas razones, se recomienda trabajar bajo este modelo cuando estamos utilizando seguridad informática.

Para poder trabajar con este modelo, o con este patrón modelo vista controlador, es necesario que la forma en que escribamos el código lo hagamos en Programación Orientada a Objetos (POO) porque lo que vamos a hacer es instanciar, heredar una serie de clases que van a estar desde el controlador o desde el modelo y que se verán reflejadas en la vista, o que se van a ver reflejadas en el controlador, o que se verán reflejadas en el mismo modelo.

Aplicación de ejemplo

A continuación, se plantea y desarrolla un ejemplo de aplicación web usando como guía de diseño este patrón MVC. De esta manera veremos sus ventajas y nos servirá como introducción a la arquitectura que utiliza [Symfony](#).




Aplicación de ejemplo

Descripción de la aplicación

Vamos a construir una aplicación web para elaborar y consultar un **repositorio de alimentos** con datos acerca de sus propiedades dietéticas. Usaremos una BD para almacenar dichos datos que consistirá en una sola tabla con la siguiente información sobre alimentos:

- Nombre del alimento
- Energía (kcal)
- Cantidad de proteínas
- Cantidad hidratos de carbono (g)
- Cantidad de fibra (g)
- Cantidad de grasa (g)

[todo ello por cada 100 gramos de alimento]

Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Comentarios	Extra
id 	int(11)			No	Ninguna		AUTO_INCREMENT
nombre	varchar(255)	utf8_general_ci		No	Ninguna		
energia	decimal(10,0)			No	Ninguna		
proteina	decimal(10,0)			No	Ninguna		
hidratocarbono	decimal(10,0)			No	Ninguna		
fibra	decimal(10,0)			No	Ninguna		
grasatotal	decimal(10,0)			No	Ninguna		

La aplicación permitirá, entre otros, insertar nuevos alimentos, realizar consultas por nombre de alimento y por energía, etc.

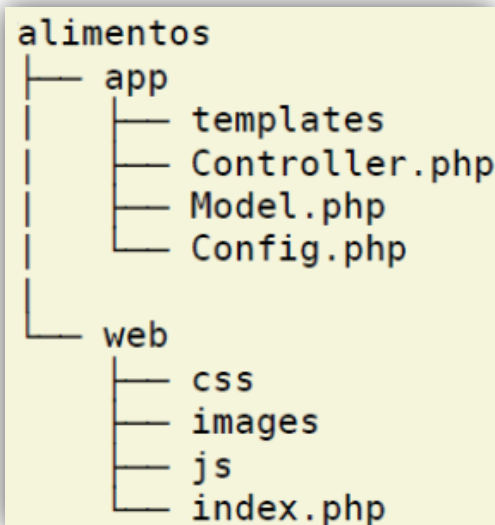
Además, contará con un menú accesible desde cualquier parte de la aplicación.

Aplicación de ejemplo

Organización de los archivos

La "anatomía" de una aplicación web típica consiste en:

1. El código que será procesado en el servidor (PHP, Java, Python, etc.) para construir dinámicamente la respuesta.
2. Los assets (activos) de la aplicación, que son aquellos archivos servidos directamente, sin ningún tipo de proceso. Suelen ser imágenes, css y js.



Esta organización pretende que no se pueda acceder desde el navegador más que al código imprescindible para que funcione. Consiste, simplemente, en colocar en el *Directorio Principal* sólo los activos y los scripts PHP de entrada a la aplicación. El resto de archivos, fundamentalmente librerías PHP's y ficheros de configuración (XML, YAML, JSON, etcétera), se ubicarán fuera del *Directorio Principal* (*Document Root*) y serán incluidos por los scripts de inicio según lo requieran.

Aplicación de ejemplo

Controlador frontal

La manera más directa de construir una aplicación en PHP consiste en escribir un script PHP para cada página de la aplicación. Sin embargo, esta práctica presenta problemas cuando la aplicación adquiere cierto tamaño y pretendemos que siga creciendo.

Además, todos los scripts de una aplicación realizan tareas comunes (interpretar y manipular la petición, comprobar las credenciales de seguridad, cargar la configuración, etc.). Esto significa que parte del código podría compartirse entre los scripts. Esto podemos solucionarlo con la inclusión de ficheros PHP.

Pero, ¿qué ocurre si en un momento dado, habiendo escrito ya mucho código, queremos añadir a todas las páginas de la aplicación una nueva característica que requiere el uso de una nueva librería? Tenemos, entonces, que añadir dicha modificación a todos los scripts *PHP* de la aplicación, lo cual supone una degradación en el mantenimiento y un motivo que aumenta la probabilidad de fallos una vez que el cambio se haya realizado.

Aplicación de ejemplo

Si el problema lo genera el hecho de tener muchos scripts, que además comparten bastante código, usaremos un solo script que se encargue de procesar todas las peticiones. A este único script de entrada se le conoce como controlador frontal.

Entonces, ¿cómo puedo crear muchas páginas distintas con un solo script? La clave está en usar la query string de la URL como parte de la ruta que define la página que se solicita. Según los parámetros enviados en la query string, el controlador frontal determinará qué acciones debe hacer para construir la página solicitada.

```
href = "index.php?ruta=listar"
```

Es decir, en este ejemplo de MVC con PHP puro, habrá un controlador frontal y el acceso a cada apartado de la página se basará en el query string, que es realmente lo que se añade a la dirección cuando se envía un formulario GET.

Aplicación de ejemplo

Creación de la estructura de directorios

Crearemos una clase para la parte del controlador que denominaremos Controller, otra para el modelo que denominaremos Model, y para los parámetros de configuración de la aplicación utilizaremos una clase que llamaremos Config. Los archivos donde se definen estas clases los ubicaremos en el directorio app. Por otro lado, las Vistas serán implementadas como plantillas PHP en el directorio app/templates.

Los archivos css, javascript , imágenes y controlador frontal los colocaremos en el directorio web.

Cada apartado de la página tendrá:

- CONTROLADOR → Una función en **Controller.php**
- MODELO → Una función en **Model.php**
- VISTA → Un archivo en la carpeta **templates**

Además de una ruta definida en index.php y un enlace en el menú en layout.php



En Symfony, la estructura de carpeta será más larga... 😓 Pero podremos con ella 💪

```
alimentos
├── app
│   ├── templates
│   ├── Controller.php
│   ├── Model.php
│   └── Config.php
└── web
    ├── css
    ├── images
    ├── js
    └── index.php
```

El controlador frontal será index.php

Config.php

```
<?php
```

```
class Config
{
    static public $mvc_bd_hostname = "localhost";
    static public $mvc_bd_nombre = "alimentos";
    static public $mvc_bd_usuario = "root";
    static public $mvc_bd_clave = "";
    static public $mvc_css = "estilo.css";
}

?>
```



En Symfony tocaremos algún archivo de configuración, pero no será tan cortito y estará escrito en otro lenguaje

En este archivo, como ves, sólo se definen algunas variables que se usarán en los otros archivos. Están relacionadas con la base de datos y con el estilo.

Aplicación de ejemplo

El mapeo de rutas en el controlador frontal

En una aplicación web definiremos las URL's asociadas a cada una de sus páginas. Para la nuestra definiremos las siguientes:

URL	Acción
http://localhost/alimentos/index.php?ruta=inicio	Mostrar la pantalla de inicio
http://localhost/alimentos/index.php?ruta=listar	Apartado de listar alimentos
http://localhost/alimentos/index.php?ruta=insertar	Apartado para insertar alimentos
http://localhost/alimentos/index.php?ruta=buscar	Apartado para buscar alimentos
http://localhost/alimentos/index.php?ruta=ver&id=X	Apartado para ver el alimento X

A cada una de estas URL's les asociaremos un método público de la clase [Controller](#).

Como ves, en el ejercicio que hagamos de MVC, realmente vamos a estar siempre en `index.php`. Pero visualmente no será así, ya que una parte de la página irá cambiando según el parámetro `ruta`.



En Symfony, las rutas funcionarán de otra manera. Ya lo veremos...

Aplicación de ejemplo

Estos métodos se suelen denominar acciones. Cada acción se encarga de calcular dinámicamente los datos requeridos para construir su página. Podrá utilizar, si le hace falta, los servicios de la clase Model. Una vez calculados los datos, se los pasará a una plantilla donde se realizará la construcción del documento HTML que será devuelto al cliente.

Todos estos elementos serán "orquestrados" por el controlador frontal, que será implementado en un script llamado [index.php](#) ubicado en el directorio web.

En concreto, la responsabilidad del controlador frontal será:

- ✓ Cargar la configuración del proyecto y las implementaciones del Modelo, del Controlador y de la Vista.
- ✓ Analizar los parámetros de la petición HTTP (*request*) comprobando si la página solicitada en ella tiene asignada alguna acción del Controlador. Si es así la ejecutará, si no dará un error 404 (page not found).

¡Ojo! El controlador frontal y la clase Controller, son cosas distintas y tienen distintas responsabilidades (sus nombres puede dar lugar a confusiones).

index.php

```
<?php
// carga del modelo y los controladores
require_once __DIR__ . '/../app/Config.php';
require_once __DIR__ . '/../app/Model.php';
require_once __DIR__ . '/../app/Controller.php';

// RUTAS
// Este array asociativo se usa para saber qué acción (función del
// controlador) se debe disparar
$map = array(
    'inicio' => array('controller' => 'Controller', 'action' =>
        'inicio'),
    'listar' => array('controller' => 'Controller', 'action' =>
        'listar'),
    'insertar' => array('controller' => 'Controller', 'action' =>
        'insertar'),
    'buscarPorNombre' => array('controller' => 'Controller', 'action'
=> 'buscarPorNombre'),
    'ver' => array('controller' => 'Controller', 'action' => 'ver')
);

// Parseo de la ruta
if (isset($_GET['ruta']))
{
    if (isset($map[$_GET['ruta']]))
    {
        $ruta = $_GET['ruta'];
    }
    else
```

Este archivo (controlador frontal) sólo se encarga de manejar las rutas y lanzar el controlador que toque. Apenas lo tocaremos (sólo para añadir las rutas de los nuevos apartados de la web)

```

    {
        //Este error saltará si no está definida la ruta arriba en
        //el array asociativo
        header('Status: 404 Not Found');
        echo '<html><body><p style="color:red"><b>ERROR: No existe
la ruta ' . $_GET['ruta'] . '</b></p></body></html>';
        exit;
    }
}
else
{
    $ruta = 'inicio';
}

$controlador = $map[$ruta];

// Ejecucion del controlador asociado a la ruta
if(method_exists($controlador['controller'], $controlador['action']))
{
    call_user_func(array(new $controlador['controller'],
        $controlador['action']));
}
else
{
    //Este error saltará si no está definida la función asociada a
    //la ruta en Controller.php
    header('Status: 404 Not Found');
    echo '<html><body><p style="color:red"><b>ERROR: El controlador
' . $controlador['controller'] . '-' . $controlador['action'] . ' no
existe</b></p></body></html>';
}
?>
```



En Symfony no existirá este concepto de controlador frontal

Aplicación de ejemplo

Controlador → Clase Controller

- Esta clase implementa una serie de métodos públicos que hemos denominado **acciones** para indicar que son métodos asociados a URL's. Fíjate como en cada una de las acciones se declara un array asociativo (**params**) con los datos que maneja dicho apartado de la web y que serán mostrados en la plantilla, pero, en ningún caso, hay información acerca de cómo se mostrarán dichos datos.
- Por otro lado, casi todas las acciones utilizan un objeto de la clase **Model** para realizar operaciones relativas a la lógica de negocio, en nuestro caso a todo lo relativo con la gestión de los alimentos.
- Todas las acciones, al final, llaman a la vista que corresponda. Ella es la que define cómo se muestran los datos (¿h1? ¿tabla? ¿lista?...).

Controller.php

```
<?php
class Controller
{
    //Acción para el apartado "inicio"
    //El único dato que manejamos es la fecha actual, que
    //la obtenemos con date
    public function inicio()
    {
        $params = array('fecha' => date('d-m-y'));

        require __DIR__ . '/templates/inicio.php';
    }

    //Acción para el apartado "ver alimentos"
    //En $params tenemos un array alimentos, que llenamos
    //llamando al método dameAlimentos de Model
    public function listar()
    {
        $params = array('alimentos' => array());

        $m = new
        Model(Config::$mvc_bd_hostname,Config::$mvc_bd_usuario,Config:
        :$mvc_bd_clave,Config::$mvc_bd_nombre);

        $params['alimentos'] = $m->dameAlimentos();

        require __DIR__ .
        '/templates/mostrarAlimentos.php'; //vista de este apartado
    }

    //Acción para el apartado "insertar alimento"
    //En $params tenemos un campo para cada input, que
    //llenamos al enviar el formulario por si hubiera que conservar
    //alguno
    public function insertar()
    {
        $params = array('nombre' => '', 'energia' => '',
        'proteina' => '', 'hc' => '', 'fibra' => '', 'grasa' => '',
        'mensaje' => NULL);

        $m = new
        Model(Config::$mvc_bd_hostname,Config::$mvc_bd_usuario,Config:
        :$mvc_bd_clave,Config::$mvc_bd_nombre);
```

```
        if ($_SERVER['REQUEST_METHOD'] == 'POST')
        {
            if ($m->validarDatos($_POST['nombre'],
            $_POST['energia'], $_POST['proteina'], $_POST['hc'],
            $_POST['fibra'], $_POST['grasa']))
            {
                $m-
                >insertarAlimento($_POST['nombre'], $_POST['energia'],
                $_POST['proteina'], $_POST['hc'], $_POST['fibra'],
                $_POST['grasa']);

                header('Location:
                index.php?ruta=listar');
            }
            else
            {
                $params['nombre'] =
                $_POST['nombre'];
                $params['energia'] =
                $_POST['energia'];
                $params['proteina'] =
                $_POST['proteina'];
                $params['hc'] = $_POST['hc'];
                $params['fibra'] =
                $_POST['fibra'];
                $params['grasa'] =
                $_POST['grasa'];
                $params['mensaje'] = 'No se ha
                podido insertar el alimento. Revisa el formulario';
            }
        }

        require __DIR__ .
        '/templates/insertarAlimento.php'; //vista de este apartado
    }

    //Acción para el apartado "buscar por nombre"
    //En $params tenemos un campo para el input del nombre,
    //y un array resultado que llenamos llamando al método
    //buscarAlimentosPorNombre de Model
    public function buscarPorNombre()
    {
        $params = array('nombre' => '', 'resultado' =>
        array());

        $m = new
        Model(Config::$mvc_bd_hostname,Config::$mvc_bd_usuario,Config:
        :$mvc_bd_clave,Config::$mvc_bd_nombre);
```

```
        if ($_SERVER['REQUEST_METHOD'] == 'POST')
        {
            $params['nombre'] = $_POST['nombre'];
            $params['resultado'] = $m-
            >buscarAlimentosPorNombre($params['nombre']);
        }

        require __DIR__ .
        '/templates/buscarPorNombre.php'; //vista de este apartado
    }

    //Acción para cuando se pulsa encima del nombre de un
    //alimento
    //Este es un poco diferente, ya que decimos
    //directamente que $params es el alimento
    public function ver()
    {
        if (!isset($_GET['id']))
        {
            throw new Exception('Pagina no
            encontrada');
        }

        $id = $_GET['id'];

        $m = new
        Model(Config::$mvc_bd_hostname,Config::$mvc_bd_usuario,Config:
        :$mvc_bd_clave,Config::$mvc_bd_nombre);

        $alimento = $m->dameAlimento($id);

        require __DIR__ . '/templates/verAlimento.php';
    }
}
?>
```

Este archivo es clave, ya que en él están los controladores de cada uno de los apartados de la web (en forma de función). Cada función (acción) maneja los datos que toquen en un array \$params. Luego llama a Model si lo necesita, y finalmente siempre llama a su vista correspondiente.

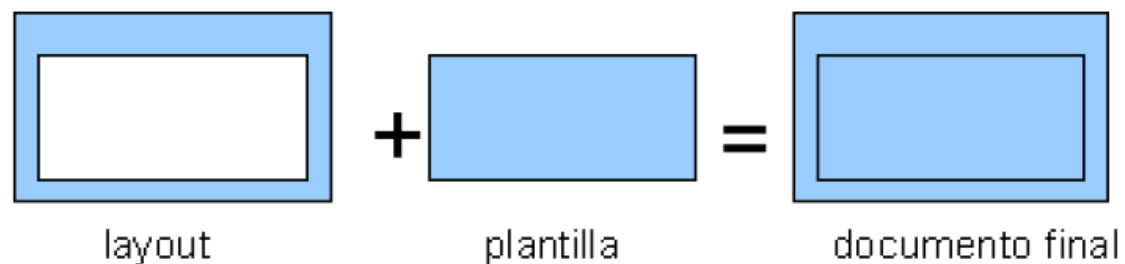
Aplicación de ejemplo

La implementación de la Vista

El próximo paso es construir el archivo HTML que será enviado al navegador con los datos calculados en la acción. Para llevar a cabo la tarea debemos pensar en el requisito que obliga a la aplicación a mostrar un menú de navegación en todas sus vistas, así como en el principio de programación DRY (Don't Repeat Yourself).

El problema será resuelto mediante la descomposición de la vista en dos partes:

- ❖ **layout**, es una plantilla que representa todo el marco común a la aplicación como es la cabecera con el menú y el pie de página.
- ❖ **plantilla**, que representa la visualización de la acción que está siendo ejecutada.



layout.php

(Dentro de templates)

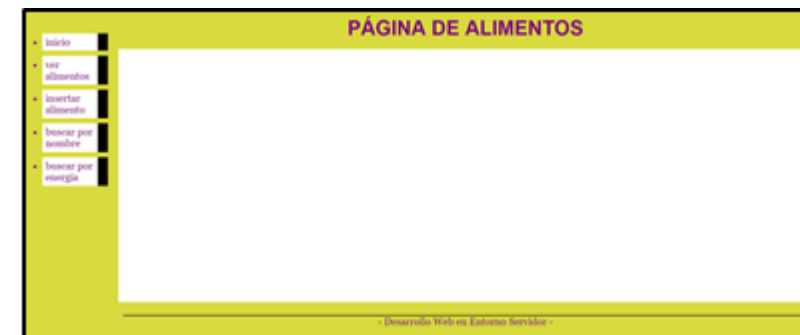
Este archivo crea la **vista de la parte fija de la página web** (lo que no va a cambiar, estemos en el apartado que estemos). En este caso: el header (parte de arriba), el nav (parte de la izquierda) y el footer (parte de abajo). La parte cambiante será lo restante, o sea, el main (zona central). En dicha parte, como ves, sólo se hace un echo de una variable llamada **\$contenido**. Dicha variable se “llenará” en cada uno de los apartados con un contenido distinto (en el archivo destinado a la vista de cada una de las secciones de la página web).

```
<!doctype html>
<html lang="es">
  <head>
    <meta charset="utf-8">
    <title>Página de Alimentos</title>
    <link rel="stylesheet" href="<?php echo
'css/'.Config::$mvc_css ?>"/>
  </head>
  <body>
    <header>
      <h1>Página de alimentos</h1>
    </header>
    <nav>
      <ul>
        <li><a
href="index.php?ruta=inicio">inicio</a></li>
        <li><a href="index.php?ruta=listar">ver
alimentos</a></li>
        <li><a
href="index.php?ruta=insertar">insertar alimento</a></li>
```

```
        <li><a
href="index.php?ruta=buscarPorNombre">buscar por
nombre</a></li>
        <li><a
href="index.php?ruta=buscarPorEnergia">buscar por
energía</a></li>
      </ul>
    </nav>
    <main>
      <?php echo $contenido; ?>
    </main>
    <footer>
      - Desarrollo Web en Entorno Servidor -
    </footer>
  </body>
</html>
```



En Symfony, las distintas vistas no se “juntarán” de esta manera. Lo veremos...



Aplicación de ejemplo

Las plantillas PHP

Una plantilla es un fichero de texto con la información necesaria para generar documentos en cualquier formato de texto (HTML, XML, JSON, etc.). Cualquier tipo de plantilla consiste en un documento con el formato que se quiere generar, y con variables expresadas en el lenguaje propio de la plantilla y que representan a los valores que son calculados dinámicamente por la aplicación.

Cuando desarrollamos aplicaciones web con PHP, la forma más sencilla de implementar plantillas es usando el propio PHP como lenguaje de plantillas. ¿Qué significa esto? Pues que cuando queremos introducir un dato dinámico utilizamos el lenguaje PHP para hacerlo (exactamente, lo que hemos venido haciendo hasta ahora a lo largo del curso).

mostrarAlimentos.php

(Dentro de templates)

```
<?php ob_start() ?>
```

```
<h2>Ver alimentos</h2>
```

```
<table>
```

```
<tr>
```

```
<th>alimento (por 100g)</th>
```

```
<th>energia (Kcal)</th>
```

```
<th>grasa (g)</th>
```

```
</tr>
```

```
<?php foreach ($params['alimentos'] as $alimento) : ?>
```

```
<tr>
```

```
<td><a href="index.php?ruta=ver&id=<?php echo $alimento['id']?>">
```

```
<?php echo $alimento['nombre'] ?></a></td>
```

```
<td><?php echo $alimento['energia']?></td>
```

```
<td><?php echo $alimento['grasatotal']?></td>
```

```
</tr>
```

```
<?php endforeach; ?>
```

```
</table>
```

```
<?php $contenido = ob_get_clean() ?>
```

```
<?php include 'layout.php' ?>
```

Este archivo crea la vista del apartado “ver alimentos”.

Lo que hace es:

- Gracias a `ob_start()` y `ob_get_clean()`, se introduce en la variable `$contenido` todo lo que se quiere mostrar en dicho apartado de la web.
- Después, una vez “llenada” la variable `$contenido`, se incluye `$layout`.

(en todas las vistas de los otros apartados, se repetirá este proceso)

VER ALIMENTOS

ALIMENTO (POR 100G)	ENERGIA (KCAL)	GRASA (G)
pepino	3	3
calabaza	8	8
fresa	4	4



Tras este ejercicio, no volveremos a usar las funciones `ob_start()` y `ob_get_clean()`. En Symfony veremos que las plantillas funcionan de un modo distinto. Se hacen con *Twig* y se basan en bloques y herencia.

El resto de plantillas. Todas usan el método descrito antes...

inicio.php (Dentro de templates)

```
<?php ob_start() ?>
```

```
<h2>Inicio</h2>
<time> Fecha: <?php echo $params['fecha'] ?> </time>
<p>Bienvenido a la aplicación de alimentos</p>
```

```
<?php $contenido = ob_get_clean() ?>
```

```
<?php include 'layout.php' ?>
```

INICIO

Fecha: 08-12-18

Bienvenido a la aplicación de alimentos

INSERTAR ALIMENTO

Nombre

Energia (Kcal)

Proteina (g)

H. de carbono (g)

Fibra (g)

Grasa total (g)

insertar

* Los valores deben referirse a 100 g del alimento

insertarAlimento.php (Dentro de templates)

```
<?php ob_start() ?>
```

```
<h2>Insertar alimento</h2>
```

```
<?php if(isset($params['mensaje'])) :?>
```

```
<div class="mensaje"><?php echo $params['mensaje'] ?></div>
```

```
<?php endif; ?>
```

```
<form name="formInsertar" action="index.php?ruta=insertar" method="POST">
```

```
<label for="nombre">Nombre</label>
```

```
<input type="text" name="nombre" id="nombre" value="<?php echo
$params['nombre'] ?>" />
```

```
<label for="energia">Energia (Kcal)</label>
```

```
<input type="text" name="energia" id="energia" value="<?php
echo $params['energia'] ?>" />
```

```
<label for="proteina">Proteina (g)</label>
```

```
<input type="text" name="proteina" id="proteina" value="<?php
echo $params['proteina'] ?>" />
```

```
<label for="hc">H. de carbono (g)</label>
```

```
<input type="text" name="hc" id="hc" value="<?php echo
$params['hc'] ?>" />
```

```
<label for="fibra">Fibra (g)</label>
```

```
<input type="text" name="fibra" id="fibra" value="<?php echo
$params['fibra'] ?>" />
```

```
<label for="grasa">Grasa total (g)</label>
```

```
<input type="text" name="grasa" id="grasa" value="<?php echo
$params['grasa'] ?>" />
```

```
<input type="submit" value="insertar" name="insertar" />
```

```
<p>* Los valores deben referirse a 100 g del alimento</p>
```

```
</form>
```

```
<?php $contenido = ob_get_clean() ?>
```

```
<?php include 'layout.php' ?>
```

buscarPorNombre.php

(Dentro de templates)

```
<?php ob_start() ?>
```

```
    <h2>Buscar por nombre</h2>
    <form name="formBusqueda" action="index.php?ruta=buscarPorNombre"
method="POST">
        <label for="nombre">nombre alimento:</label>
        <input type="text" name="nombre" id="nombre"
value="<?php echo $params['nombre'] ?>" />
        <span>(puedes escribir sólo una parte del nombre)</span>
        <input type="submit" value="buscar" />
    </form>

    <?php if (count($params['resultado'])>0): ?>
        <table>
            <tr>
                <th>alimento (por 100g)</th>
                <th>energia (Kcal)</th>
                <th>grasa (g)</th>
            </tr>
            <?php foreach ($params['resultado'] as $alimento) : ?>
                <tr>
                    <td><a href="index.php?ruta=ver&id=<?php echo
$alimento['id'] ?>"><?php echo $alimento['nombre'] ?></a></td>
                    <td><?php echo $alimento['energia'] ?></td>
                    <td><?php echo $alimento['grasatotal'] ?></td>
                </tr>
            <?php endforeach; ?>
        </table>
    <?php endif; ?>
```

```
<?php $contenido = ob_get_clean() ?>
<?php include 'layout.php' ?>
```

BUSCAR POR NOMBRE

nombre alimento:

(puedes escribir sólo una parte del nombre)

buscar

verAlimento.php

(Dentro de templates)

```
<?php ob_start() ?>
```

```
<h1><?php echo strtoupper($alimento['nombre']) ?></h1>
<table>
    <tr>
        <td>Energia</td>
        <td><?php echo $alimento['energia'] ?></td>
    </tr>
    <tr>
        <td>Proteina</td>
        <td><?php echo $alimento['proteina'] ?></td>
    </tr>
    <tr>
        <td>Hidratos de Carbono</td>
        <td><?php echo $alimento['hidratocarbono'] ?></td>
    </tr>
    <tr>
        <td>Fibra</td>
        <td><?php echo $alimento['fibra'] ?></td>
    </tr>
    <tr>
        <td>Grasa total</td>
        <td><?php echo $alimento['grasatotal'] ?></td>
    </tr>
</table>
```

```
<?php $contenido = ob_get_clean() ?>
<?php include 'layout.php' ?>
```

PEPINO

Energia	3
Proteina	3
Hidratos de Carbono	3
Fibra	3
Grasa total	3

Aplicación de ejemplo

El Modelo: acceder a la base de datos

Ya sólo nos queda el Modelo. En nuestra aplicación se ha implementado en la clase **Model** y está compuesto por una serie de funciones para persistir datos en la base de datos, recuperarlos y realizar su validación.

Dependiendo de la complejidad del negocio con el que tratemos, el modelo puede ser más o menos complejo y, además de tratar con la persistencia de los datos, puede incluir funciones para ofrecer otros servicios relacionados con el negocio en cuestión.

Model.php

El constructor crea una conexión con la BD. Después hay dos métodos auxiliares para realizar consultas de los dos tipos (que SÍ devuelven datos y que NO devuelven). Luego -esto es lo importante- entre los `****/` hay un método para cada apartado de la web. Y, al final, un pequeño método para validar el form.

```
<?php

class Model
{
    protected $conexion;

    //Esta función sólo conecta con la BD y se llama automáticamente al crear el objeto Model
    public function __construct($dbhost, $dbuser, $dbpass, $dbname)
    {
        $mvc_bd_conexion = new mysqli($dbhost, $dbuser, $dbpass, $dbname);
        $error = $mvc_bd_conexion->connect_errno;
        if ($error != null)
        {
            echo "<p>Error ".$error.
"conectando a la base de datos: ";
            echo $mvc_bd_conexion->connect_error."</p>";
            exit();
        }
        $this->conexion = $mvc_bd_conexion;

        //Esta función recibe por parámetro una sentencia SELECT y devuelve un array de alimentos
        //El array será accesible tanto con posiciones numéricas como asociativas, al ser fetch_array
        private function devolverAlimentosSelect($sql)
        {
            $consulta = $this->conexion->query($sql);
            $alimentos = array();
            while($resultado = $consulta->fetch_array())
            {
                $alimentos[] = $resultado;
            }
            return $alimentos;
        }

        //Esta función recibe por una sentencia que NO devuelve datos (insert, delete o update) y la ejecuta
        private function consultaQueNoDevuelveDatos($sql)
        {
            return $this->conexion->query($sql);
        }

        /***/

        //Esta función obtiene todos los alimentos de la BD, para el apartado "ver alimentos"
        public function dameAlimentos()
        {
            $sql = "SELECT * FROM alimentos";
            return $this->devolverAlimentosSelect($sql);
        }

        //Esta función obtiene los alimentos que contienen la cadena recibida, para el apartado "buscar por nombre"
        public function buscarAlimentosPorNombre($nombre)
        {
            $sql = "select * from alimentos where nombre like '%" . $nombre . "%'";
            return $this->devolverAlimentosSelect($sql);
        }

        //Esta función obtiene el alimentos (sólo uno) cuya id es la id recibida, para cuando hacemos click sobre un alimento
        public function dameAlimento($id)
        {
            $sql = "select * from alimentos where id=".$id;
            return $this->devolverAlimentosSelect($sql)[0];
        }

        //Esta función obtiene los alimentos que contienen la cadena recibida, para el apartado "buscar por nombre"
        public function insertarAlimento($n, $e, $p, $hc, $f, $g)
        {
            $sql = "INSERT INTO alimentos (nombre, energia, proteina, hidratocarbono, fibra, grasatotal) values ('" . $n . "'," . $e . " ,," . $p . " ,," . $hc . " ,," . $f . " ,," . $g . "')";
            return $this->consultaQueNoDevuelveDatos($sql);
        }

        /***/

        //Esta es una pequeña función de validación que se ha usado para el formulario (podría mejorarse)
        public function validarDatos($n, $e, $p, $hc, $f, $g)
        {
            return (is_string($n) & is_numeric($e) & is_numeric($p) & is_numeric($hc) & is_numeric($f) & is_numeric($g));
        }
    }
}

?>
```



En Symfony no habrá un archivo para el modelo. Las líneas del modelo estarán en el código del controlador. Y, además, no se usará mysqli para la BD, sino una herramienta llamada Doctrine.

Aplicación de ejemplo

Incorporar la hoja de estilos

Ya tenemos todo el código de la parte de servidor. Ya sólo nos falta darle un toque de estilo a los documentos HTML que enviamos al cliente y crear la base de datos que almacenará los datos persistentes sobre los alimentos.

```
body {
  padding-left: 11em;
  font-family: Georgia, "Times New Roman",
  Times, serif;
  color: purple;
  background-color: #d8da3d
}

header { text-align: center; }

nav {
  list-style-type: none;
  padding: 0;
  margin: 0;
  position: absolute;
  top: 2em;
  left: 1em;
  width: 9em
}

h1, h2 {
  font-family: Helvetica, Geneva, Arial, SunSans-Regular, sans-
  serif ;
  text-transform: uppercase;
}

nav li {
  background: white;
  margin: 0.5em 0;
  padding: 0.3em;
  border-right: 1em solid black
}

}

nav ul a { text-decoration: none }

a:link { color: blue }

a:visited { color: purple }

main {
  display: block;
  margin: auto;
  width: auto;
  min-height:400px;
}

label, input[type="submit"] {
  display: block;
  margin-top: 10px;
  margin-bottom: 10px;
}

table { border-collapse: collapse;}

td, th { border:1px solid black; padding:5px; }

th { text-transform: uppercase; }

.mensaje { color:red;}

footer { border-top: 1px solid black; text-align:center;}
```

Aplicación de ejemplo

La Base de Datos

En este ejemplo se ha usado una BD llamada **bdalimentos**, que contiene una tabla llamada **alimentos**. Puede crearse así:

```
CREATE TABLE `alimentos` (  
  `id` int(11) NOT NULL AUTO_INCREMENT,  
  `nombre` varchar(255) NOT NULL,  
  `energia` decimal(10,0) NOT NULL,  
  `proteina` decimal(10,0) NOT NULL,  
  `hidratocarbono` decimal(10,0) NOT NULL,  
  `fibra` decimal(10,0) NOT NULL,  
  `grasatotal` decimal(10,0) NOT NULL,  
  PRIMARY KEY (`id`)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Os pasaré directamente el
archivo **alimentos.sql**

Una vez hecho esto, podríamos probar la aplicación introduciendo en el navegador la URL correspondiente: <http://localhost/alimentos/web/index.php>

public function buscarPorNombre()

```
{
    $params = array('nombre' => '', 'resultado' => array());

    $m = new Model(Config::$mvc_bd_hostname, Config::$mvc_bd_usuario, Config::$mvc_bd_clave, Config::$mvc_bd_nombre);

    if ($_SERVER['REQUEST_METHOD'] == 'POST')
    {
        $params['nombre'] = $_POST['nombre'];
        $params['resultado'] = $m->buscarAlimentosPorNombre($params['nombre']);
    }

    require __DIR__ . '/templates/buscarPorNombre.php';
}
```

CONTROLADOR

Para entender mejor el MVC, lo ideal es centrarnos sólo en un apartado. En este caso, el de **BUSCAR POR NOMBRE**. Si te fijas:

- En el Controlador, se crea \$params con un campo para el nombre que se introducirá en el formulario. También un array resultado para los alimentos que devolvamos. Si el formulario se ha enviado, llamamos al modelo para llenar dicho array.
- En el Modelo, creamos la sentencia SQL que queremos, concatenando con el nombre que introdujo el usuario. La ejecutamos gracias a una función auxiliar que hay en Model.
- En la Vista, mostramos el formulario, y debajo (SÓLO SI SE HA ENCONTRADO ALGO EN LA BD) los resultados.

public function buscarAlimentosPorNombre(\$nombre)

```
{
    $sql = "select * from alimentos where nombre like '%" . $nombre . "%'";
    return $this->devolverAlimentosSelect($sql);
}

...

private function devolverAlimentosSelect($sql)
{
    $consulta = $this->conexion->query($sql);
    $alimentos = array();
    while($resultado = $consulta->fetch_array())
    {
        $alimentos[] = $resultado;
    }
    return $alimentos;
}
```

MODELO

<?php ob_start() ?>

```
<h2>Buscar por nombre</h2>
<form name="formBusqueda" action="index.php?ruta=buscarPorNombre" method="POST">
    <label for="nombre">nombre alimento:</label>
    <input type="text" name="nombre" id="nombre" value="<?php echo $params['nombre'] ?>" />
    <span>(puedes escribir sólo una parte del nombre)</span>
    <input type="submit" value="buscar" />
</form>

<?php if (count($params['resultado'])>0): ?>
<table>
    <tr>
        <th>alimento (por 100g)</th>
        <th>energia (Kcal)</th>
        <th>grasa (g)</th>
    </tr>
    <?php foreach ($params['resultado'] as $alimento) : ?>
    <tr>
        <td><a href="index.php?ruta=ver&id=<?php echo $alimento['id'] ?>"><?php echo $alimento['nombre'] ?></a></td>
        <td><?php echo $alimento['energia'] ?></td>
        <td><?php echo $alimento['grasatotal'] ?></td>
    </tr>
    <?php endforeach; ?>
</table>
<?php endif; ?>
```

VISTA

BUSCAR POR NOMBRE

nombre alimento:
pe (puedes escribir sólo una parte del nombre)
buscar

BUSCAR POR NOMBRE

nombre alimento:
pe (puedes escribir sólo una parte del nombre)
buscar

ALIMENTO (POR 100G)	ENERGIA (KCAL)	GRASA (G)
pepino	3	3

	MVC en esta unidad 8	Symfony
Controlador	En un único archivo, hay una función (acción) para cada apartado de la página web	Esto será igual
	Usamos un array \$params para manejar los datos y terminamos con la línea que llama a la vista	No usaremos un array \$params, pero la forma de operar será muy similar, y la última línea también.
Modelo	En un único archivo, hay una función para cada apartado de la página web	NO HABRÁ ARCHIVO PARA EL MODELO. Las líneas de la parte del modelo estarán incluidas en el código del controlador
	mysqli	Doctrine
Vista	Cada apartado de la página web tiene un archivo para su vista	Esto será igual
	PHP, y “metemos” la parte variable de la página en una variable	Twig, y nos basaremos en bloques y herencia
Rutas	Las definimos en el controlador frontal	No hablaremos de controlador frontal. Se pueden definir de varias formas. Lo haremos en el mismo archivo del controlador
Otras cosas	<ul style="list-style-type: none"> Hemos montado una estructura de carpetas corta Hemos creado un archivo de configuración corto en PHP Apenas hemos hecho validación. Y pocas cosas más. 	<ul style="list-style-type: none"> Al crear un proyecto, Symfony nos generará una estructura de carpetas bastante más larga Tendremos que tocar algún archivo de configuración largo en YAML Veremos herramientas de validación de formularios en Symfony. Y otras cosas de este framework como el uso de <i>bundles</i>, la seguridad en Symfony, los contenedores de servicios...

EJERCICIO 1

Realiza el mismo ejemplo completo que se explica en estas diapositivas.

- Deberás crear la estructura de directorios, con sus correspondientes archivos y copiar el código adecuadamente.
- Deberás crear en PHPMyAdmin una BD llamada **bdalimentos** e importar el *.sql* que adjuntaré en la entrega.
- Ahora, fíjate que la última opción del menú (*buscar por energía*) da error, dado que está sin hacer. Deberás hacer que funcione:



```
alimentos
├── app
│   ├── templates
│   ├── Controller.php
│   ├── Model.php
│   └── Config.php
└── web
    ├── css
    ├── images
    ├── js
    └── index.php
```

BUSCAR POR ENERGÍA

energía alimento:

BUSCAR POR ENERGÍA

energía alimento:

ALIMENTO (POR 100G)	ENERGIA (KCAL)	GRASA (G)
pepino	3	3

Después, añade estos dos apartados y haz que funcionen correctamente:

- inicio
- ver alimentos
- ver alimentos ordenados
- insertar alimento
- buscar por nombre
- buscar por energía
- eliminar por nombre

Igual que “ver alimentos”, pero debajo de la tabla saldrá esto:

Ordenar por en sentido

En el primer desplegable, las opciones son: *nombre, energía y grasa*)
Y en el segundo, son: *ascendente y descendente*.
Al pulsar en “Ordenar” los desplegables deben conservar el valor de la búsqueda.

Aparecerá lo siguiente:

nombre del alimento que quieres eliminar:

(tiene que ser exacto)

EJERCICIO 2

En el modelo MVC de EMPRESA, que adjuntaré en la entrega:

- Añade dos items mas al menú de navegación: **Ofertas** y **Novedades**, con sus correspondientes páginas de texto inventado.
- Haz que se vean como en la imagen de abajo y que funcionen correctamente.



LOGOTIPO

Inicio

Nosotros

Servicios

Contáctenos

Ofertas

Novedades

PÁGINA DE OFERTAS

PÁGINA DE OFERTAS

Lorem ipsum dolor sit amet, consectetur adipiscing elit. In sollicitudin pretium massa, eu dictum erat iaculis eget. Maecenas nisi augue, tristique sed velit at, ultrices ornare diam. Mauris interdum tellus in ligula suscipit lacinia. Sed laoreet justo lacus. Maecenas leo diam, varius vitae faucibus sed, blandit sed turpis. Sed arcu turpis, faucibus vehicula orci sit amet, iaculis pharetra dui. Ut magna lorem, ultrices euismod sapien ac, convallis sodales mi. In efficitur dui orci, quis ultricies purus lobortis sed. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Suspendisse sed turpis lacus. Etiam nec iaculis odio. In augue justo, mattis ut molestie a, cursus a leo. Mauris eget luctus eros. Duis id sapien at felis tempor hendrerit vitae vitae mauris.

Integer a quam vitae justo viverra ornare id nec ligula. Donec eget justo eros. Interdum et malesuada fames ac ante ipsum primis in faucibus. Donec magna ex, pulvinar eget aliquam vitae, tincidunt vitae arcu. Donec convallis nunc ac libero faucibus, ac eleifend neque mattis. Fusce non nisi rhoncus enim interdum viverra. Quisque id maximus nunc, vel egestas orci. Nunc vel