MVC

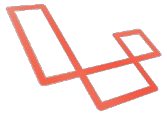
Model View Controller

¿Qué es MVC?

Model View Controller

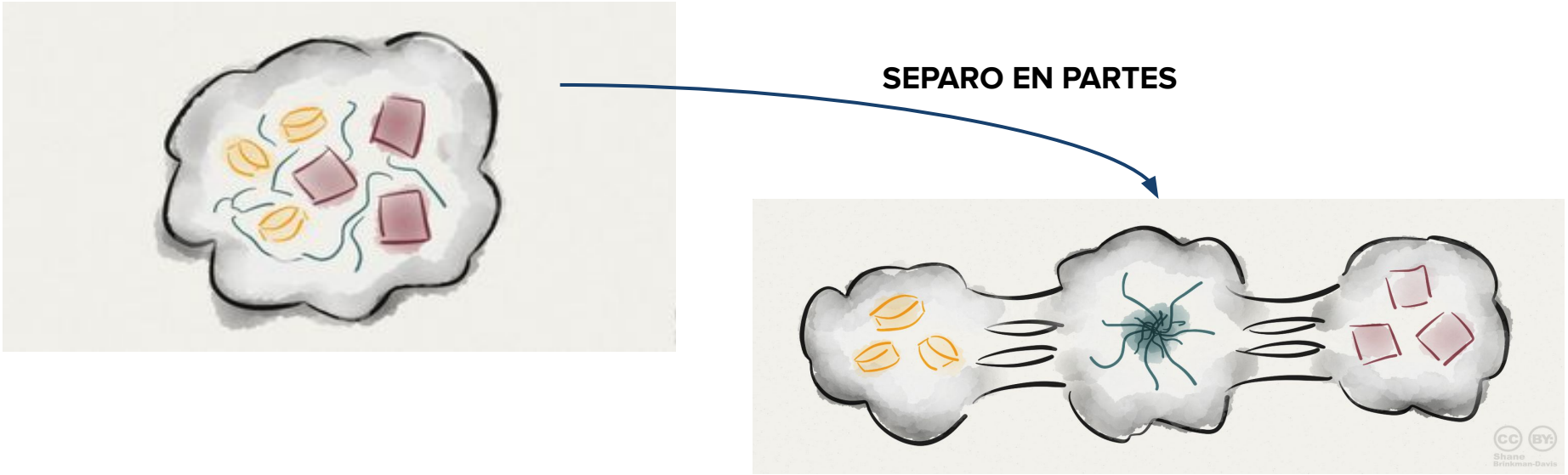
Patrón de arquitectura de software utilizado ampliamente en la industria.

- Un patrón define una estructura esencial para un sistema de software.
- Ofrecen **soluciones estándares** a problemas comunes dentro de la ingeniería del software.



¿Qué problema resuelve?

Desacopla el código de programas donde toda la *lógica,* el *acceso a datos* y la *interfaz gráfica* se encuentran bajo mismos archivos sin ninguna separación clara.



¿Qué propone MVC?

Divide la lógica del programa en **tres elementos inter-relacionados**. Cada uno cumple una función determinada.

Cada componente tiene una responsabilidad determinada y trabajan de forma coordinada:

- **MODELO**: Acceso a datos
- **VISTA**: Interfaz de usuario (Front End)
- **CONTROLADOR**: Coordinador entre vista y modelo

Modelo - Responsabilidades

MODELO

En este componente manejamos la **comunicacion con la base de datos**.

- Proteger y persistir los datos del usuario.
- Asegurar la integridad y consistencia de datos.
- Proveer métodos para:
  - Consultar datos
  - Insertar/Modificar datos
  - Borrar Datos

Vista - Responsabilidades

VISTA

En este componente generamos la **interfaz de usuario.**

- Presentar la información al usuario (front-end).
- Permitir al usuario interactuar con la aplicación

Controlador - Responsabilidades

CONTROLADOR

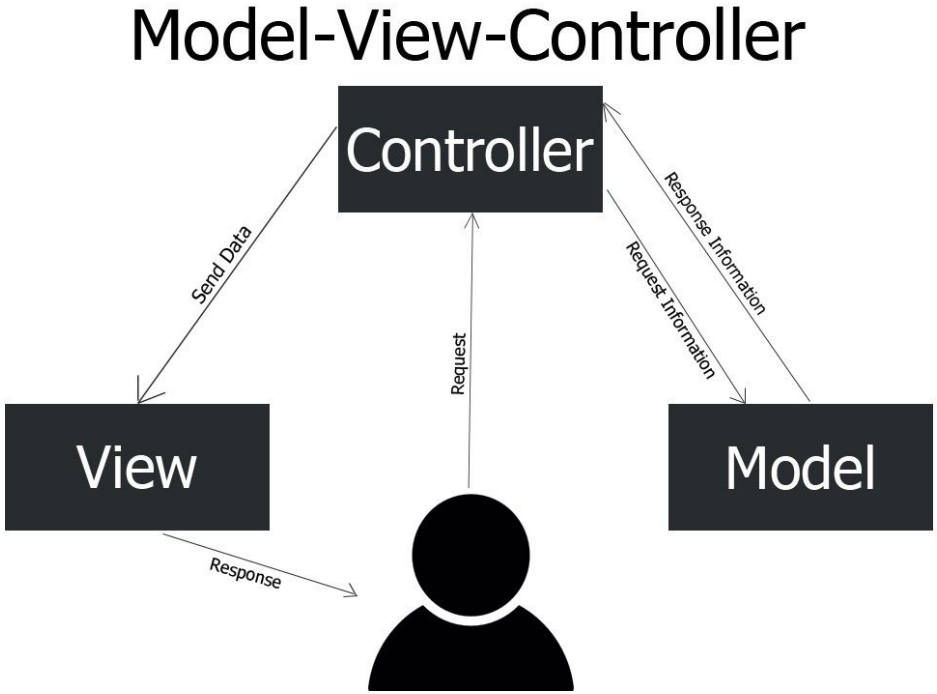
Es el **intermediario** (coordinador) entre la vista y el modelo.

- **Controla y coordina** el flujo de la aplicación.
- **Obtiene y procesa** los pedidos del usuario.
- **Valida** la entrada de datos del usuario.

EJEMPLO
Cuando una nueva entrada de un usuario llega, el **Controller** la valida y llama al **Model** para modificar los datos, luego actualiza la **View**.

¿Qué es MVC?

Funcionamiento básico de MVC



Ventajas MVC

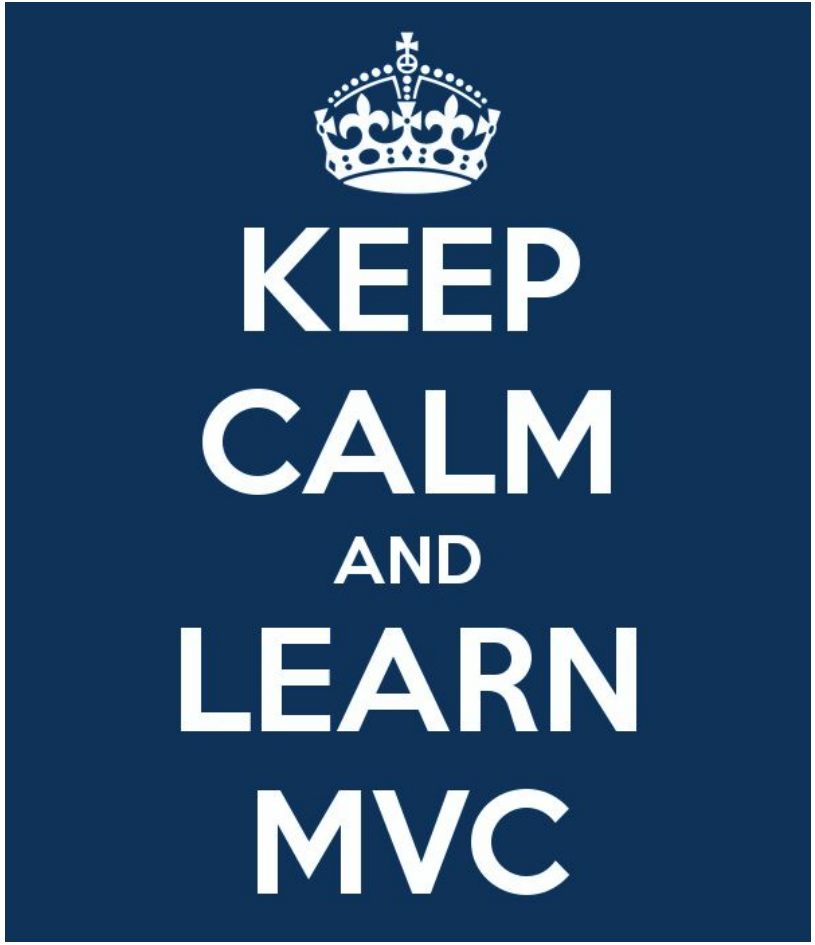
- MVC crea un sistema desacoplado
  - Reduce la complejidad de cada parte del sistema
- Alineado al mundo real
  - Desarroladores FrontEnd
  - Desarroladroes BackEnd
- Facilita
  - Escalabilidad
  - Mantenimiento

Desventajas MVC

- Agrega complejidad a la solución
- La estructura predefinida puede no ser lo que estábamos buscando

¿Cómo saber cuándo **no** usarlo?

- Donde hay elementos que no aplican a la tripla MVC


KEEP
CALM
AND
LEARN
MVC

Ejemplo

Vamos a convertir a MVC nuestra aplicación de lista de tareas.

Usaremos una solución orientada a objetos (**POO)**

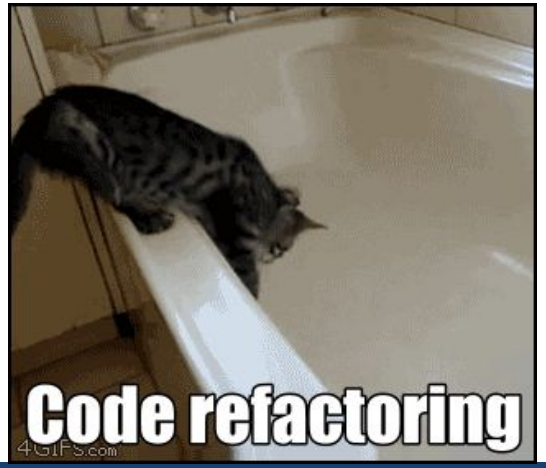
- Cada código que teníamos en las páginas principales es un método de un Controller.
- Los agrupamos por significado en clases.

Vamos a hacer un *REFACTOR*

Refactor

La refactorización (refactoring) es una técnica de la ingeniería de software para **reestructurar un código fuente**, mejorando su estructura interna **sin cambiar su comportamiento externo**.

“Código más bonito, que hace lo mismo.”



Objetos

TareasView

Renderiza (muestra) la lista de tareas.

Renderiza una sola tarea.

TareasModel

Administra las tareas en la base de datos

TareasController

Atiende las acciones del usuario

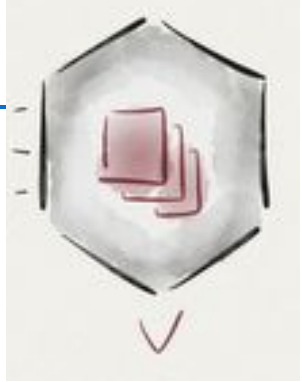
Paso a Paso

Vamos a **refactorizarlo** paso a paso en 3 iteraciones:

- Ver tareas
- Crear tareas
- Eliminar tareas

(Mismas iteraciones que al hacerlo)

Pensemos...



1er Iteración

Que vamos a tener en la **vista**?

- **HTML+PHP**
  - **Título**
  - **Formulario**
  - **Lista de tareas**

¿Cual es la entrada de datos que necesita esta vista para renderizarse?

```
function verTareas($tareas) {
        ...
}
```


CHALLENGE ACCEPTED
LET'S CODE IT

views/task.view.php

```php
<?php
class TaskView {

    // [TBC]
}
```

```php
<?php
include_once 'libs/Smarty.class.php';

class TaskView {
  private $smarty;

  function __construct() {
      $this->smarty = new Smarty;
  }

  public function showPage($tareas){
      $this->smarty->assign("tareas", $ta
      $this->smarty->display('index.tpl')
  }
}
?>
```

Pensemos...

Qué vamos a tener en el **modelo**?

- **Consulta a la BBDD**
  - **PHP + PDO**
- **Método que devuelva la lista de tareas**

```
function getTareas() {
    ...
}
```

1er Iteración


CHALLENGE ACCEPTED
LET'S CODE IT

models/task.model.php

```php
<?php
class TaskModel {

    // [TBC]

}
```

Acá va la invocación a la base de datos usando PDO!

Pensemos...

Qué vamos a tener en el **controlador**?

1. **Pide al modelo las tareas**
2. **Se las da a la vista para que las muestre**

1er Iteración


CHALLENGE ACCEPTED
LET'S CODE IT

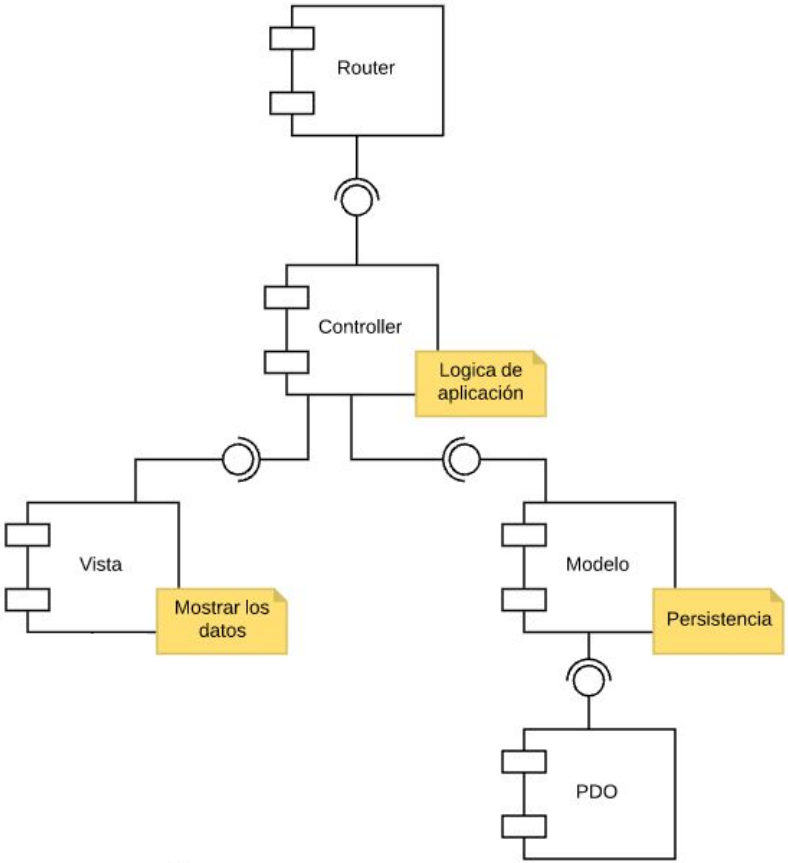
controller/task_controller.php

```php
<?php
require_once 'models/task.model.php';
require_once 'views/task.view.php';

class TaskController {

    // [TBC]

}
```

Resultado

BOLIVAR: https://gitlab.com/unicen/Web2/livecoding2019/bolivar/todo-list/commit/637e44f9de182c695362698c775c3415fed92d56
TANDIL: https://gitlab.com/unicen/Web2/livecoding2019/tandil/todo-list/commit/4e3ed305224a5d57addc9c949b9b0a8d5e983cb3

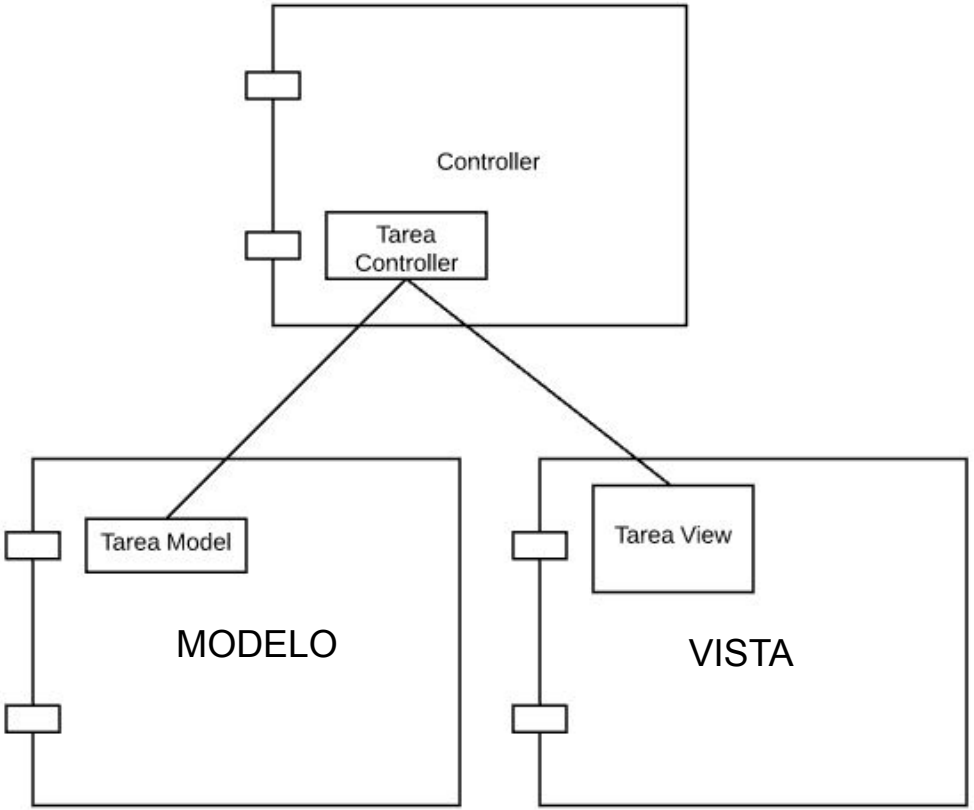
Cómo queda la arquitectura?



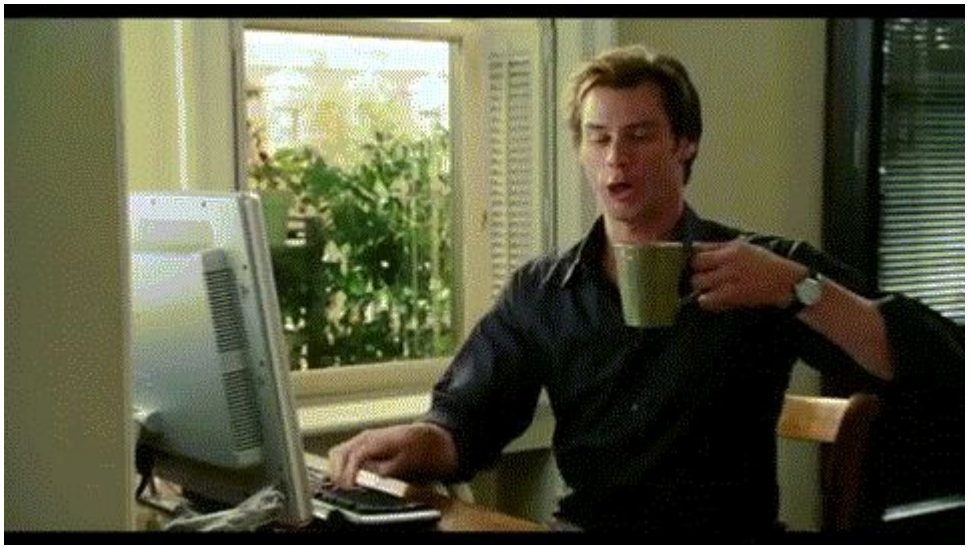
MVC

Cada componente está hecho de clases

En este ejemplo tienen una sola, pero puede haber más



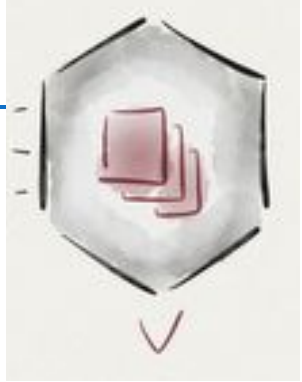
2da Iteración



Pensemos...

Qué vamos a tener en la **vista**?

- Formulario con botón para crear la tarea.



2da Iteración

Pensemos...

Qué vamos a tener en el **modelo**?

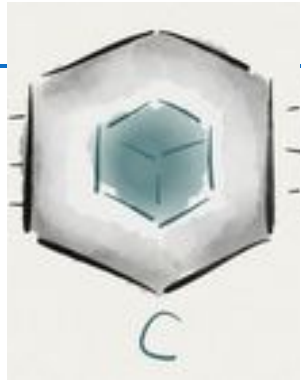
- **[TBC]**

2da Iteración

Pensemos...

Qué vamos a tener en el **controlador**?

- **[TBC]**



2da Iteración

Resultado

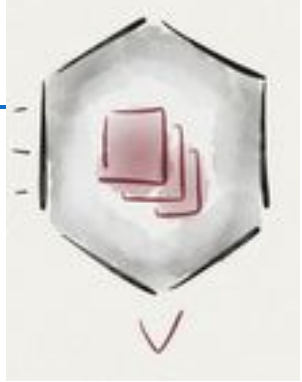
BOLIVAR: https://gitlab.com/unicen/Web2/livecoding2019/bolivar/todo-list/commit/883ef248b22b04bb4a099a1aef7c62a7c8c5ec60
TANDIL: https://gitlab.com/unicen/Web2/livecoding2019/tandil/todo-list/commit/53dfe2bb644dbb8e7b05555a8d04f402b344ce63

3ra Iteración

Pensemos...

Qué vamos a tener en la **vista**?

- **[TBC]**



3ra Iteración

Pensemos...

Qué vamos a tener en el **modelo**?

- **[TBC]**

3ra Iteración

Pensemos...

Qué vamos a tener en el **controlador**?

- **[TBC]**

3ra Iteración

Resultado

BOLIVAR: https://gitlab.com/unicen/Web2/livecoding2019/bolivar/todo-list/commit/6764a9c74aae368c72568cadea0e018a813b3091
TANDIL: https://gitlab.com/unicen/Web2/livecoding2019/tandil/todo-list/commit/d16cb3878b1525cdc475e0fdd7eb72ac5f2f5452


One more thing...

Cuantas clases vamos a tener?

- **Modelo**
  - Una clase por entidad
  - Ej. Una clase para Tareas
- **Vista**
  - Una clase por entidad por formato a mostrar
  - Ej. Una clase para página html de lista de Tareas
- **Controlador**
  - Una clase por cada lógica a controlar
  - Ej. Una clase para ABM de tareas

Referencias

- MVC (Wikipedia)
- Understanding MVC
- MVC Martin Fowler
- Patterns of Enterprise Application Architecture - Martin Fowler - Capitulo 14