Generación dinámica de páginas Web

- **Generación dinámica de páginas Web**
  - Modelos y arquitecturas de desarrollo.
  - Aplicación de PHP a dichos modelos.

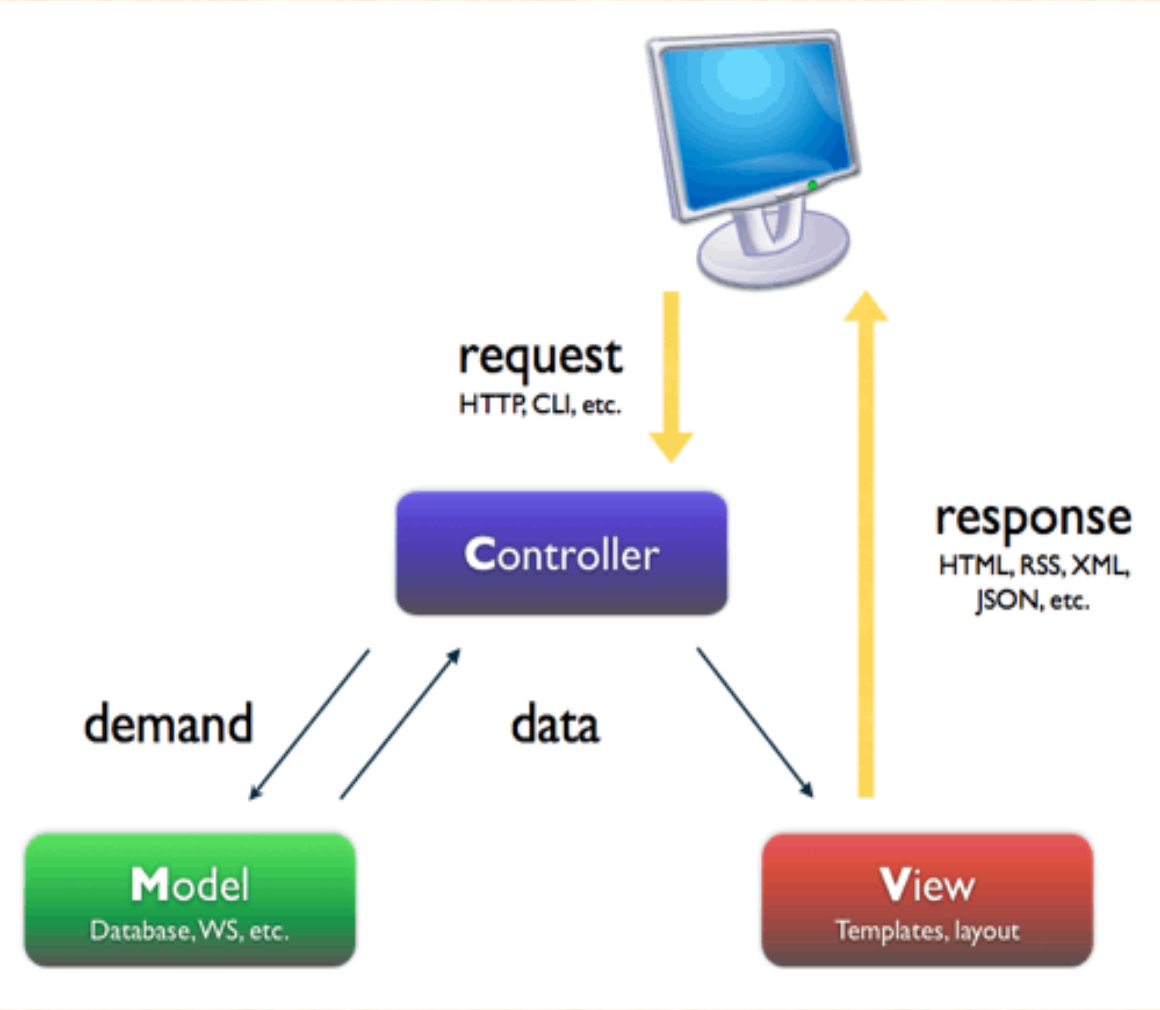
Modelos de desarrollo

- Arquitectura MVC (Model-View-Controller)
- Arquitectura n-tier (multi-nivell, multi-capa)
  - Esquema mínimo: 3-tier
  - Otros esquemas

Arquitectura MVC

- MVC = Model – View – Controller
- Se divide la aplicación en tres partes, cada una de ellas con una funcionalidad completamente definida e independiente de las demás.
- La división en tres partes de la aplicación hace el desarrollo más fácil (centrado en una sola tarea) y permite modularidad en el desarrollo.

Arquitectura MVC



Arquitectura MVC

- ***Model* (Modelo):->independiente del sistema operativo**
  - Gestiona el almacenamiento de la información.
  - También se le llama "**capa de persistencia**".
  - Puede implementar almacenamiento en memoria, en base de datos, en archivos XML, texto plano JSON, CSV, formatos propietarios.
  - Permite añadir encriptación en la información almacenada.
  - Ofrecerá métodos (funciones) en uno o varios archivos php plenamente funcionales, sencillos de operación sobre los datos al controlador.
    Por ejemplo, métodos llamados `guardarVenta()` o `validarLogin()` o `recogerDato()` o `resolverOperacion()`

Arquitectura MVC

- ***View* (Vista): para diferentes sistemas operativos… web, mobile, etc**
  - Genera las vistas de usuario que mostrarán los resultados de las acciones que el usuario realice a través de la aplicación.
  - Puede trabajar en cualquier formato (no exclusivamente HTML): imágenes, documentos PDF, Excel, etc.
  - Puede también ofrecer información codificada y preparada para ser gestionada por otras aplicaciones: JSON, XML, formatos propietarios, etc.

    JSON

```
{"employees":[
  { "firstName":"John", "lastName":"Doe" },
  { "firstName":"Anna", "lastName":"Smith" }
]}
```

Arquitectura MVC

XM

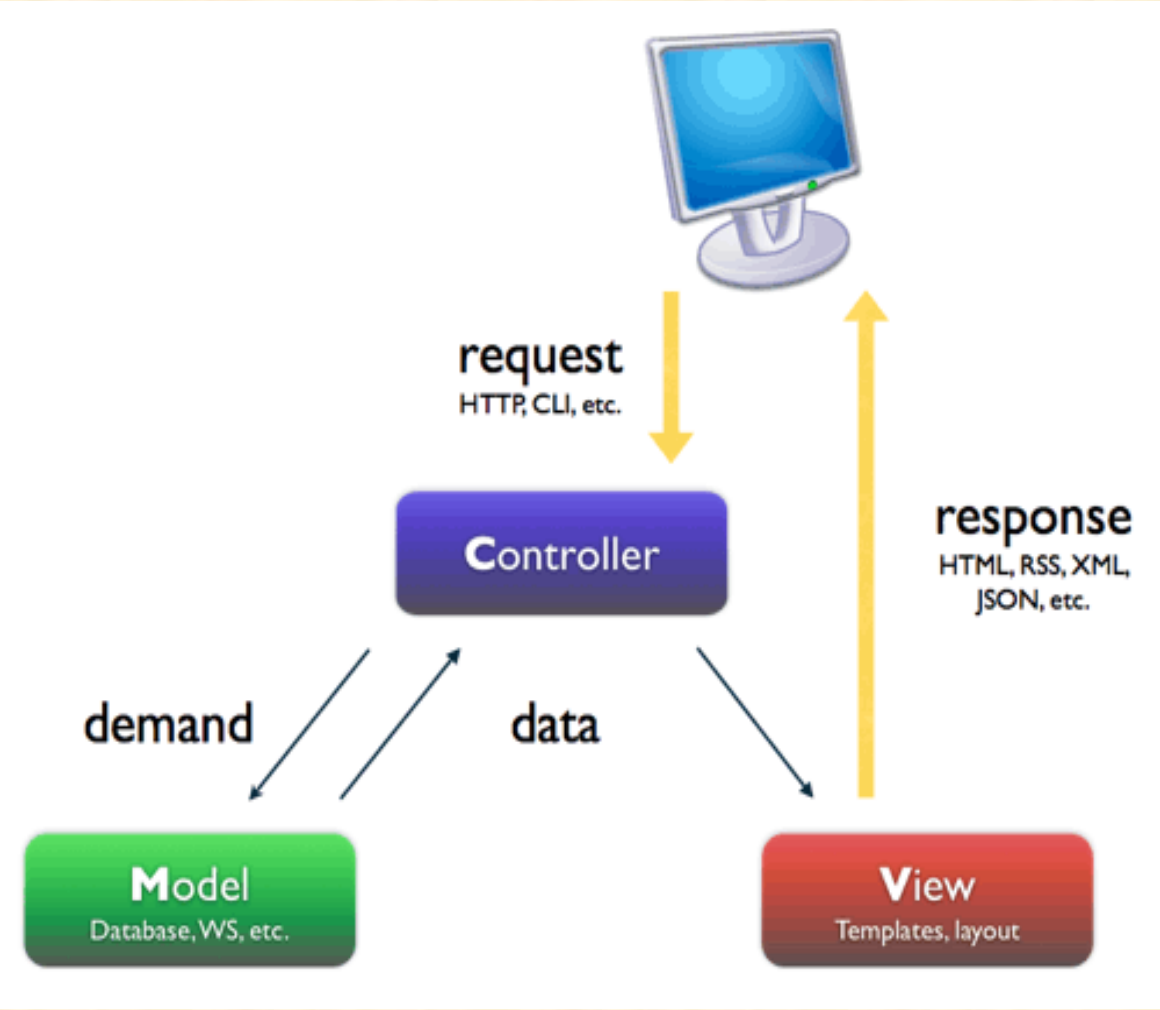
```
<employees>
  <employee>
    <firstName>John</firstName>
    <lastName>Doe</lastName>
  </employee>
  <employee>
    <firstName>Anna</firstName>
    <lastName>Smith</lastName>
  </employee>
</employees>
```

-Presentará funcionales como `mostrarFormularioLogin()`, `paginaError()` o `mostrarResumenCestaCompra()`

Arquitectura MVC

- ***Controller* (Controlador): el cerebro de la app**
  - Recibe y procesa las peticiones del usuario y activa las acciones correspondientes que deberán ser ejecutadas en consecuencia.
  - Disparará acciones sobre el modelo de datos y decidirá la vista que deberá retornarse al usuario como resultado de dichas acciones. Envía a model los datos que ha recibido, o /y envía a vista los resultados
  - Aquí es donde se procesará la información proveniente de los formularios, botones, enlaces, etc.
  - Se puede considerar esta capa como el "*cerebro*" de la aplicación.

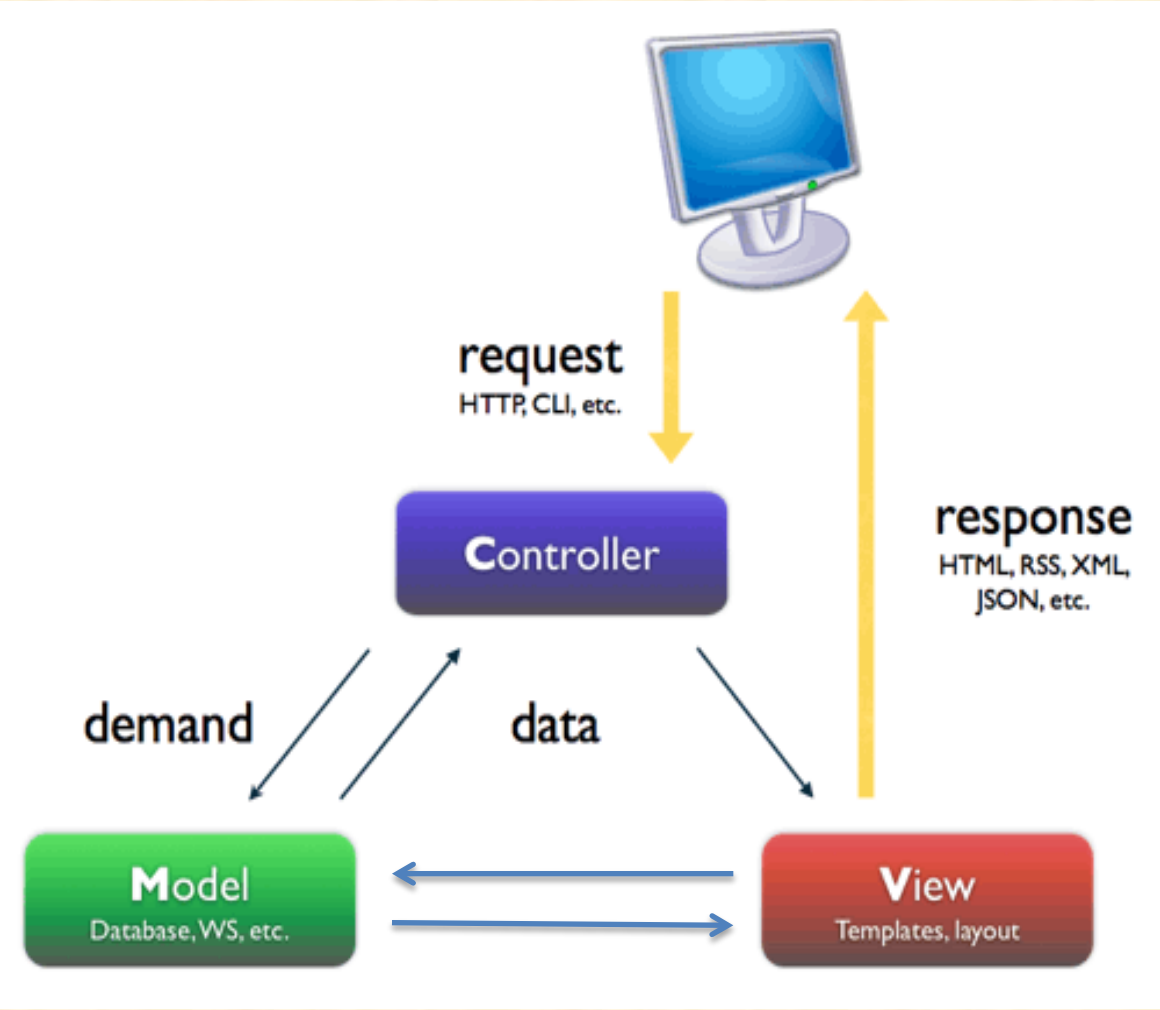
Arquitectura MVC



Arquitectura MVC

- El flujo de datos suele ser implementado siguiendo los caminos indicados por las flechas:
  - Del cliente al controlador: Petición
  - Del controlador al modelo: Acciones y resultados.
  - Del controlador a la vista: Ejecutar una vista.
  - De la vista al usuario: Entrega de la información.
- También se permite que la capa de vista acceda al modelo para acelerar el renderizado de páginas (listados, por ejemplo).

Arquitectura MVC



Arquitectura MVC

- La arquitectura MVC presenta, como principal ventaja, la **modularidad** y **escalabilidad** asociada al particionamiento del problema.
- El hecho de tener separadas las capas de modelo y de vista nos reporta una **ventaja estratégica** mayúscula: Puede ser sustituida por otra implementación.

Arquitectura MVC

- Así, si deseamos pasar los datos de una colección de archivos a una base de datos MySQL, lo único que deberemos modificar es la implementación de las funciones de la capa de modelo.
- También podemos sustituir la capa de presentación por otra, para poder ofrecer una mejor experiencia de usuario o para cubrir dispositivos que ofrezcan funcionalidades mejoradas en la interfície de usuario.

Arquitectura MVC

- También podemos aprovechar la modularidad para ofrecer **diversas implementaciones del modelo** (moodle)(a escoger por el administrador del sistema o de la aplicación) y **diversas implementaciones de la vista** (a escoger por el usuario o según el entorno detectado de cliente).
- El elemento central inamobible será siempre el controlador, pues en él se definen los flujos de trabajo. Puede mejorarse éste para ofrecer **nuevas funciones** en la aplicación.

Arquitectura n-tier

- La arquitectura n-tier tiene un origen formal similar al del modelo MVC, pero ampliado a una **cantidad indeterminada de capas**.
- Cada **capa** realizará una función específica y bien definida, y recibiendo las peticiones (acciones a realizar) de la capa superior y devolviéndole los resultados asociados (datos a gestionar).