Madrid, 19 de Mayo de 2015
Universidad Complutense de Madrid
Grado de Ingeniería de Computadores
Asignatura: Sistemas Web

VEERY docs

www.veery.es

Desarrollo:
Alberto Miedes Garcés
Javier Bermúdez Blanco
Daniel Pinto Rivero
Rafael Ramirez Urbina
Ignacio María Ferreras Astorqui
Denys Sypko

Descripcion de la Arquitectura

Para generar las vistas al usuario, hemos decidido aplicar nuestros conocimientos adquiridos previamente en TP (Tecnología de la Programación) e IS (Ingeniería de Software) y trabajar con un patrón de arquitectura con el que nos sintiéramos familiarizados, por lo que decidimos utilizar el patrón de Modelo-Vista-Controlador (MVC), que además es bastante predominante en los frameworks dispònibles a nivel de programación web (Rails en Ruby, Laravel y Symfony en PHP, Angular.JS y Express.JS en JavaScript, por mencionar algunas.)

Aunque ya conocemos el patrón MVC, comentaremos brevemente cómo actúa en una aplicación web como la que hemos desarrollado.

Modelo

El modelo se encarga de manejar el estado de la aplicación. Con “estado” nos referimos al concepto de nuestra aplicación. El modelo no conoce nada sobre el HTML, solo provee los métodos para consultar el estado y modificar el estado.

Vista

La vista es la representación de la interfaz de usuario. Una vista puede consultar al modelo, pero no puede cambiar el estado de la aplicación. Una vista se implementa utilizando plantillas que representa el HTML de la página web.

Controlador

Las acciones de los usuarios en la Vista, se envían al Controlador. En un espacio web, esto es posible al permitir que el Controlador maneje las peticiones HTTP entrantes. El Controlador recibe las peticiones del usuario, y las traduce en acciones que el Modelo debe de tomar. Luego selecciona la Vista apropiada para representar la respuesta.

Las ventajas que observamos al elegir este patrón de arquitectura son las siguientes:

Separacion de peticiones y paginas

Gracias a que el Controlador se encuentra a cargo de manejar las pecitiones y seleccionar las Vistas apropiadas, no existe emparejamiento entre una peticion realizada por el usuario y una página de respuesta.

Vistas ilogicas

Debido a que las vistas solo contienen lo necesario para mostrar una página bonita al usuario, toda la lógica se encuentra en objetos ajenos a la Vista. Cambiar la lógica, no requiere tocar la disposición ni el diseño de nuestra aplicación. Y análogamente, cambiar el diseño o la disposición, no involucra cambiar la lógica. De esta forma, al hacer un rediseño de nuestra exitosa aplicación web, no será necesario modificar la lógica, haciéndola más escalable y menos “error/bug-prone”.

La logica es ciega, como el amor

Como todas las acciones que corresponden al estado de nuestra aplicación son manejadas por el Modelo, es posible hacer cambios de implementación sin tocar la interfaz de usuario.

Por otro lado, algunas limitaciones que encontramos con este patrón son los siguientes:

A donde va esto

En ocasiones puede resultar dificil comprender donde una parte especifica de la aplicación debe de ir y su finalidad. El patrón MVC, en comparación con los ejemplos básicos establecidos en clase, resulta extremadamente complejo.

Mucha clase. Mucho objeto.

Siguiendo con la linea de pensamiento de la limitación anterior, desarrollar una aplicación web siguiendo el patrón de MVC resultará en un número mayor de clases y objetos en comparación con una página con un único archivo PHP “monolítico”. Eso quiere decir que al generar la base, se tendrá que realizar mucho mas trabajo. Tardaremos mas en ver los resultados.

Listado de Scripts

PHP

Controlador

Genericos

Scripts utilizados a lo largo de la aplicacion, con la misma finalidad

```
/**
* Instanciar una clase.
*/
public function __construct();
```

CRUD (Create, read, update, delete)

Funciones basicas utilizadas por las clases. (Usuarios, Posts, etc.)

```
/**
* Representar la vista de alguna parte de la aplicación. Ej: Un usuario, un album, una canción.
*/
public function view();

/**
* Crear una nueva entrada/objeto. Ej: Un usuario, un album, una canción.
*/
public function create();

/**
* Actualizar una nueva entrada/objeto. Ej: Un usuario, un album, una canción.
*/
public function update();

/**
* Eliminar una nueva entrada/objeto. Ej: Un usuario, un album, una canción.
*/
public function delete();
```

Controlador de Usuarios

```
/**
* Verificar las credenciales de registro de un usuario y activarlo en la app y BD.
*/
public function verify();

/**
* Creación  de relación de “follow” de un usuario a otro.
*/
public function followUser();

/**
* Borrado de relación de “follow” de un usuario a otro.
*/
public function unfollowUser();
```

Controlador de Autenticacion

```
/**
* Iniciar la sesión del usuario. Acceso a las funciones de la app.
*/
public function login();

/**
* Destruir la sesión. Cerrar la sesión del usuario.
*/
public function logout();
```

Controlador de Password

Clase creado en el 2012 por Anthony Ferrera (ircmaxell@php.net).

Ofrece compatibilidad con el manejo simplificado en PHP 5.5 de los passwords.

```
/**
* Hacer hash a la contraseña utilizando el algoritmo especificado.
*/
public function make();

/**
* Obtener información sobre el hash de la contraseña. Regresa un array de la información
* utilizada para generar el hash.
* array(
*    'algo' => 1,
*    'algoName' => 'bcrypt',
*    'options' => array(
*        'cost' => 10,
*    ),
* )
*/
public function password_get_info();

/**
* Determinar si la contraseña necesita pasar por un proceso de hash nuevamente.
* De necesitarla, hacer el hash de nuevo.
*/
public function needs_rehash();

/**
* Verificar una contraseña contra un hash.
*/
public function verify();
```

Controladores que forman parte del nucleo de la app

Clase Database

```
/**
* Instancia la conexión con la base de datos.
*/
public function get();
```

Clase Router

Clase creado por Noah Buscher (https://github.com/NoahBuscher/Macaw).

Macaw es un router PHP simple y open source.

Clase Session

```
/**
* Inicializar la sesión.
*/
public function init();

/**
* Agregar valor a una sesión por medio de un indice.
*/
public function set();

/**
* Extraer un elemento de la sesión, eliminando el indice y su valor.
*/
public function pull();

/**
* Obtener el valor de un indice en la sesión.
*/
public function get();

/**
* Destruir la sesión.
*/
public function destroy();
```

Clase View

```
/**
* Representar una vista.
*/
public function render();

/**
* Representar una vista parcial, una plantilla que se repita a lo largo de la app. Ej: Header.
*/
public function renderpartial();

/**
* Agregar cabecera HTTP.
*/
public function addheader();

/**
* Agregar array de cabeceras HTTP.
*/
public function addheaders();
```

Clase URL

```
/**
* Redireccionar al URL indicado.
*/
public function redirect();

/**
* Retornar a la página previa.
*/
public function previous();

/**
* Genera un 'slug' para colocar “friendly url” a los accesos en perfiles, albumes y canciones.
*/
public function generateSlug();
```

Modelo

VeeryModel

```
/**
* Get the latest posts of the users that the main user follows.
*/
public function getFeed();
```

AuthModel

```
/**
* Obtener objeto Usuario a través de su userHandle/nombre de usuario.
*/
public function getUser();

/**
* Obtener objeto Usuario a través de su correo electrónico.
*/
public function getUserByEmail();
```

userModel

```
/**
* Obtener un array con todos los objeto Post correspondientes al userID provisto.
*/
public function getUserPosts();

/**
* Verificar si el usuario es un artista, teniendo en cuenta el userID provisto.
*/
public function isArtist();

/**
* Obtener un array con todos los objeto User que “siguen” al usuario correspondiente al userID
* provisto.
*/
public function getFollowers();

/**
* Obtener un array con todos los objeto User al que usuario correspondiente al userID
* provisto “sigue”.
*/
```

```
public function getFollowing();

/**
* “Seguir” a un usuario.
*/
public function followUser();

/**
* Dejar de “seguir” a un usuario.
*/
public function unfollowUser();

/**
* Obtener los albums de un usuario.
*/
public function getAlbums();

/**
* Obtener los videos de un usuario.
*/
public function getVideos();

/**
* Verificar que no existe un usuario con el mismo correo o nombre de usuario.
*/
public function uniqueUser();

/**
* Verificar si el usuario existe.
*/
public function verifyUser();
```

JavaScript

Masonry

Como plugin en JavaScript utilizamos “Masonry” para representar adecuadamente la cuadricula de posts de usuarios, es decir, que no haya espacio entre ellos como sucedería solo utilizando CSS floats.
Mas información sobre Masonry: **masonry**.desandro.com/

jPlayer

Para la reproducción de música utilizamos jPlayer. jPlayer es una libreria de Javascript para manejar media. Es completamente gratis y open source.
Mas información sobre jPlayer: jplayer.org/

jQuery/Ajax

En jQuery/Ajax tenemos dos scripts:

1. Un script encargado de dinamicamente realizar la busqueda de parametros en la base de datos y a partir de los resultados y haciendo una representación de los mismos sin hacer un reload de la web.

2. Un script encargado de “seguir” y dejar de “seguir” a usuarios que realiza una llamada a eliminar o agregar una relación en la base de datos “vry_followers”.

Misc

YouTube

Para la reproducción de video utilizamos el API de YouTube. En principio realizamos la reproducción a través del formato de embed que se ofrece por medio de iframe.

Estructura de la Base de Datos Veery

Para la página web solo utlizamos una base de datos “Veery” con 7 tablas. Las tablas se identifican con el prefijo “vry”.

vry_albums

Tabla que maneja la información relacionada a los albums.

albumID

Identificador único que representa a un album en especifico.

artistID

Identificador único que relaciona a un album en especifico con un artista.

albumName

Nombre del álbum.

albumCover

Nombre y formato de la imagen que compone la caratula del album.

vry_artists

Tabla que maneja la correspondencia entre el identificador de usuario y el identificador de artista.

artistID

Identificador único que representa a un artista en especifico.

userID

Identificador único que representa a un usuario en especifico.

vry_followers

Tabla que maneja la correspondencia entre las relaciones a la hora de “seguir” o no “seguir” entre dos usuarios.

relationID

Identificador único que representa la relación seguido-seguidor.

userID

Identificador único que representa a un usuario en especifico.

followerID

Identificador único que representa a un usuario que “sigue” al usuario del campo userID.

vry_songs

Tabla que maneja la información relacionada a las canciones.

albumID

Identificador único que representa a un album en especifico.

songID

Identificador único que representa a una canción en especifico.

songPath

Nombre y formato del archivo de audio que compone la canción.

songTitle

Titulo de la canción.

songArtist

Nombre de los artistas que cantan en la canción.

songPlayCount

Número de veces que se ha reproducido una canción.

vry_users

Tabla que maneja la información relacionada a los usuarios.

userID

Identificador único que representa a un usuario en especifico.

userName

Nombre del usuario.

userLastName

Apellidos del usuario.

userSlug

Slug del usuario, creado para implementar user-friendly URLs.

userType

Tipo de usuario. (0: Admin, 1: Normal, 2: Artista)

userHandle

Handle/Nombre de usuario del usuario.

userPassword

Contraseña del usuario.

userEmail

Correo electrónico del usuario.

userProfilePicture

Nombre y formato de la imagen que compone la foto de perfil del usuario.

userHeader

Nombre y formato de la imagen que compone la foto de cover del usuario.

userBio

Texto biografico sobre el usuario.

userHash

Cadena de texto hecha hash para verificar y completar el registro del usuario.

userActive

Numero que indica si el usuario está activo. (1: Activo 0: No Activo)

vry_user_posts

Tabla que maneja los posts de los usuarios.

postID

Identificador único que representa a un post creado por un usuario.

UserID

Identificador único que representa a un usuario en especifico.

postSlug

Slug del post, creado para implementar user-friendly URLs.

postContent

Contenido del post.

postDate

Fecha de creación del post.

vry_video

Tabla que maneja la información relacionada a los videos.

videoID

Identificador único que representa a un video publicado por un artista.

artistID

Identificador único que representa a un artista en especifico.

videoTitle

Título del video.

videoArtist

Identificador único que representa a un post creado por un usuario.

videoPath

Url de YouTube donde se aloja el video.

videoPlayCount

Número de veces que se ha reproducido un video.