

Yii2 Framework

1. Instalación

- 1.1. Desde cero
- 1.2. Clonando desde GitHub un proyecto ya existente
- 1.3. Configuración del sitio virtual
- 1.4. Comprobación de requisitos previos

2. Conceptos clave

- 2.1. Componentes
- 2.2. Propiedades
- 2.3. Eventos
- 2.4. Comportamientos
- 2.5. Configuraciones
- 2.6. Alias
- 2.7. Autoloading de clases
- 2.8. Localizador de servicios
- 2.9. Contenedor de inyección de dependencias

3. Estructura de una aplicación

- 3.1. Introducción
- 3.2. Scripts de entrada
- 3.3. Aplicaciones
- 3.4. Componentes de aplicación
- 3.5. Controladores
- 3.6. Modelos
- 3.7. Vistas
- 3.8. Módulos
- 3.9. Filtros
- 3.10. Widgets
- 3.11. Assets
- 3.12. Extensiones

4. Gestión de peticiones

- 4.1. Introducción
- 4.2. Arranque (Bootstrapping)
- 4.3. Enrutado y creación de URLs
- 4.4. Peticiones
- 4.5. Respuestas
- 4.6. Sesiones y cookies

5. Trabajo con bases de datos

- 5.1. Database Access Objects (DAO)

[5.1.1. Ejemplos](#)

[5.1.2. Vinculación de parámetros](#)

[5.1.3. Consultas no-SELECT](#)

[5.2. Query Builder](#)

[5.3. Active Record](#)

[6. Recogida de datos del usuario](#)

[6.1. Creación de formularios](#)

[6.2. Validación de la entrada](#)

[6.2.1. Validaciones ad-hoc](#)

[7. Visualización de datos](#)

[7.1. Formateado de datos](#)

[7.2. Paginación](#)

[7.3. Ordenación](#)

[7.4. Proveedores de datos](#)

[7.5. Widgets](#)

[7.6. Scripts de cliente](#)

[8. Correo electrónico](#)

[9. Seguridad](#)

[9.1. Autenticación](#)

[9.1.1. Configuración de yii\web\User](#)

[9.1.2. Implementación de yii\web\IdentityInterface](#)

[9.1.3. Usar yii\web\User](#)

[9.2. Autorización](#)

[9.2.1. Filtrado de control de acceso \(ACF\)](#)

[9.2.2. Control de acceso basado en roles \(RBAC\)](#)

[9.3. Contraseñas](#)

[9.4. Criptografía](#)

[9.5. Seguridad en vistas](#)

[9.6. Clientes de autenticación](#)

[9.7. Buenas prácticas](#)

[9.7.1. Principios básicos](#)

[9.7.1.1. Filtrar la entrada](#)

[9.7.1.2. Escapar la salida](#)

[9.7.2. Evitar inyecciones de SQL](#)

[9.7.3. Evitar XSS](#)

[9.7.4. Evitar CSRF](#)

[10. Pruebas](#)

[10.1. Introducción](#)

[10.2. Desarrollo mediante pruebas](#)

[10.3. Cuándo y cómo hacer las pruebas](#)

[10.4. Codeception](#)

[10.4.1. Pruebas de aceptación](#)

[10.4.1.1. En modo headless](#)

[10.4.1.1.1. Sin Selenium](#)

[10.4.1.1.2. Con Selenium](#)

[10.4.1.2. En modo normal](#)

[10.4.2. Posibles problemas](#)

[10.5. Fixtures](#)

[10.6. Travis-CI](#)

[11. Estilo del código](#)

[12. Documentación \(obsoleto\)](#)

[13. Despliegue en Heroku](#)

[13.1. Limitaciones de la cuenta gratuita \(Free\)](#)

[14. Manipulación de imágenes](#)

1. Instalación

1.1. Desde cero

```
$ composer create-project --prefer-dist yiisoft/yii2-app-basic videoyii
```

O mejor:

```
$ proyecto.sh videoyii
```

1.2. Clonando desde GitHub un proyecto ya existente

```
$ git clone https://github.com/iesdonana/videoyii.git
```

```
$ cd videoyii
```

```
$ composer install
```

```
$ composer run-script post-create-project-cmd
```

1.3. Configuración del sitio virtual

La primera vez hay que activar el módulo rewrite de Apache:

```
$ sudo a2enmod rewrite
```

Una vez hecho esto, ya no hay que hacerlo más.

```
$ sudo vi /etc/hosts
```

Añadir la línea:

```
127.0.0.1 videoyii.local
```

```
$ ping videoyii.local (comprobar que responde)
```

```

$ cd /etc/apache2/sites-available
$ sudo cp 000-default.conf videoyii.conf
$ sudo vi videoyii.conf
    Cambiar y añadir:
    ServerName videoyii.local
    DocumentRoot /var/www/web/videoyii/web

    <Directory /var/www/web/videoyii/web>
        Options +FollowSymLinks
        IndexIgnore */*

        RewriteEngine on

        # if a directory or a file exists, use it directly
        RewriteCond %{REQUEST_FILENAME} !-f
        RewriteCond %{REQUEST_FILENAME} !-d

        # otherwise forward it to index.php
        RewriteRule . index.php
    </Directory>
$ sudo a2ensite videoyii
$ sudo service apache2 restart
Acceder a http://videoyii.local y comprobar que funciona correctamente.

```

1.4. Comprobación de requisitos previos

```

$ cd ~/web/videoyii/web
$ ln -sf ../requirements.php .
Acceder a http://videoyii.local/requirements.php y comprobar. Una vez satisfechos
todos los requisitos:
$ cd ~/web/videoyii/web
$ rm requirements.php
Paquetes que probablemente hagan falta para satisfacer los requisitos:
$ sudo apt install php7.1-curl php7.1-xml php7.1-gd php7.1-intl

```

2. Conceptos clave

2.1. Componentes

- Son los bloques básicos de construcción de aplicaciones en Yii.
- Son instancias de la clase `yii\base\Component` (o una de sus subclases).
- Proporcionan a otras clases las siguientes características:
 - Propiedades
 - Eventos
 - Comportamientos

2.2. Propiedades

- Son una extensión del concepto de “propiedad” de las clases de PHP mediante *getters* y *setters*.
- Toda clase que herede de `yii\base\Object` permite usar `getXyz()` y `setXyz()` para definir la propiedad `xyz`. Así se podrá usar `$objeto->xyz`.

2.3. Eventos

- Permite inyectar código personalizado dentro de código existente en ciertos puntos de ejecución.
- Cuando se dispara el evento, el código asociado se ejecuta automáticamente.
- Para crear un manejador de eventos, se puede usar, por ejemplo:

```
$foo->on(Foo::EVENT_HELLO, function ($event) {  
    // event handling logic  
});
```

2.4. Comportamientos

- Permiten ampliar la funcionalidad de un componente sin tener que cambiar la herencia de la clase.
- Lo hace “inyectando” los métodos y las propiedades del comportamiento dentro de la clase del componente.
- Además, los comportamientos pueden responder a los eventos disparados por el componente al que se asocia.

2.5. Configuraciones

- Se usan para crear nuevos objetos o inicializar objetos ya existentes.
- Se indica el nombre de la clase y una lista de los valores iniciales de las propiedades del objeto.
- También puede incluir manejadores de eventos y comportamientos.
- Se puede usar:
 - `Yii::createObject($config)` para crear un objeto, lo que utiliza el contenedor de inyección de dependencias y, por tanto, se aplicarán los valores iniciales que se hayan podido definir con `Yii::$container->set()`, con algo así, por ejemplo:

```
Yii::$container->set('yii\data\Pagination', [  
    'defaultPageSize' => 5,  
]);
```
 - **¡Cuidado!** En el ejemplo de arriba, si el nombre de la clase empieza con `\`, quedando `'\yii\data\Pagination'`, la cosa no funciona. Lo mejor es usar el método estático `className()`:

```
Yii::$container->set(yii\data\Pagination::className(), [  
    'defaultPageSize' => 5,  
]);
```
 - `Yii::configure($objeto, $config)` para asignar propiedades a un objeto ya existente, pero **no** utiliza el contenedor de inyección de dependencias, sino que se limita a asignar los valores a las propiedades.
 - `new Clase($config)` para crear un objeto, cuyo constructor usará internamente `Yii::configure($config)`, por lo que **tampoco** se tendrán en cuenta las dependencias definidas en el contenedor de inyección de dependencias.

2.6. Alias

- Representan rutas o URLs.
- Se usan para no tener que codificar rutas absolutas o URLs directamente en el proyecto.
- Empiezan por `@`.
- Ejemplos:
 - `Yii::getAlias('@yii')` → `"/home/ricardo/web/basic/vendor/yiisoft/yii2"`
 - `Yii::getAlias('@app')` → `"/home/ricardo/web/basic"`
- Alias predefinidos (en el orden en que se van definiendo):
 - Se define cuando se incluye el archivo `Yii.php` en el script de entrada:
 - `@yii` representa la ruta de instalación del framework Yii.
 - Se definen durante la fase de [preinit\(\)](#):
 - `@app` representa la ruta base de la aplicación.

- @runtime representa el directorio temporal donde la aplicación va generando archivos durante su ejecución (normalmente es @app/runtime).
- @vendor representa el directorio donde se instalan los paquetes de composer (normalmente es @app/vendor).
- @bower representa el directorio donde se instalan los paquetes de bower (normalmente es @vendor/bower-asset, definido así en la configuración de la aplicación).
- @npm representa el directorio donde se instalan los paquetes de npm (normalmente es @vendor/npm-asset, definido así en la configuración de la aplicación).
- En este punto se definen los alias definidos en la propiedad 'aliases' de la configuración de la aplicación (config/web.php o config/console.php).
- Se definen durante la fase de [bootstrap\(\)](#):
 - @web representa la URL base de la aplicación (tiene el mismo valor que yii\web\Request::\$baseUrl).
 - @webroot representa el DocumentRoot de la aplicación (normalmente es @app/web).
 - Las [extensiones instaladas](#) crean en este punto los alias que correspondan.
- Ver <https://github.com/yiisoft/yii2/issues/10157> y <https://stackoverflow.com/questions/28797051/setting-aliases-in-yii2-within-the-app-config-file>.

2.7. Autoloading de clases

- Es un autoloader que cumple con el [estándar PSR-4](#).
- Para que funcione, hay que seguir dos reglas:
 1. Cada clase debe pertenecer a un espacio de nombres (como en \foo\bar\MiClase).
 2. Cada clase debe guardarse en un archivo individual cuya ruta se determina por el siguiente algoritmo:

```
// $className is a fully qualified class name without the leading backslash
$classFile = Yii::getAlias('@' . str_replace('\\', '/', $className) . '.php');
```

- Al programar, hay que crear las clases en un espacio de nombres que cuelgue del raíz app. El autoloader la encontrará porque @app es un alias predefinido que siempre existe en Yii.

2.8. Localizador de servicios

- Es un objeto que sabe cómo proporcionar todo tipo de servicios (o componentes) que pueda necesitar una aplicación.

- Dentro del localizador de servicios, cada componente existe como una única instancia identificada mediante un ID, que se usa para recuperar el componente de dentro del localizador de servicios.
- El localizador de servicios es una instancia de la clase `yii\di\ServiceLocator` (o una subclase).
- El más típico en Yii es el **objeto aplicación**, al que se accede mediante `\Yii::$app`. Los servicios que proporciona se denominan **componentes de aplicación**, como `request`, `response` o `urlManager`. Esos componentes se suelen definir mediante *configuraciones*.

2.9. Contenedor de inyección de dependencias

- Es un objeto que sabe cómo instanciar y configurar objetos y todos los objetos de los que depende.
- Soporta los siguientes tipos de inyección de dependencias:
 - Inyección de constructores
 - Inyección de *setters* y propiedades
- Es una instancia de la clase `yii\di\Container`. Yii crea uno accesible a través de `Yii::$container`.
- `\Yii::$container->set()` sirve para registrar una dependencia.
- `\Yii::$container->get()` sirve para crear nuevos objetos:
 - A partir de un nombre de clase, interfaz o alias de dependencia.
 - Resuelve automáticamente las posibles dependencias.
- Al llamar al método `Yii::createObject()`, éste llama a `Yii::$container->get()` para crear el nuevo objeto. Esto permite personalizar globalmente la inicialización de objetos. Por ejemplo:

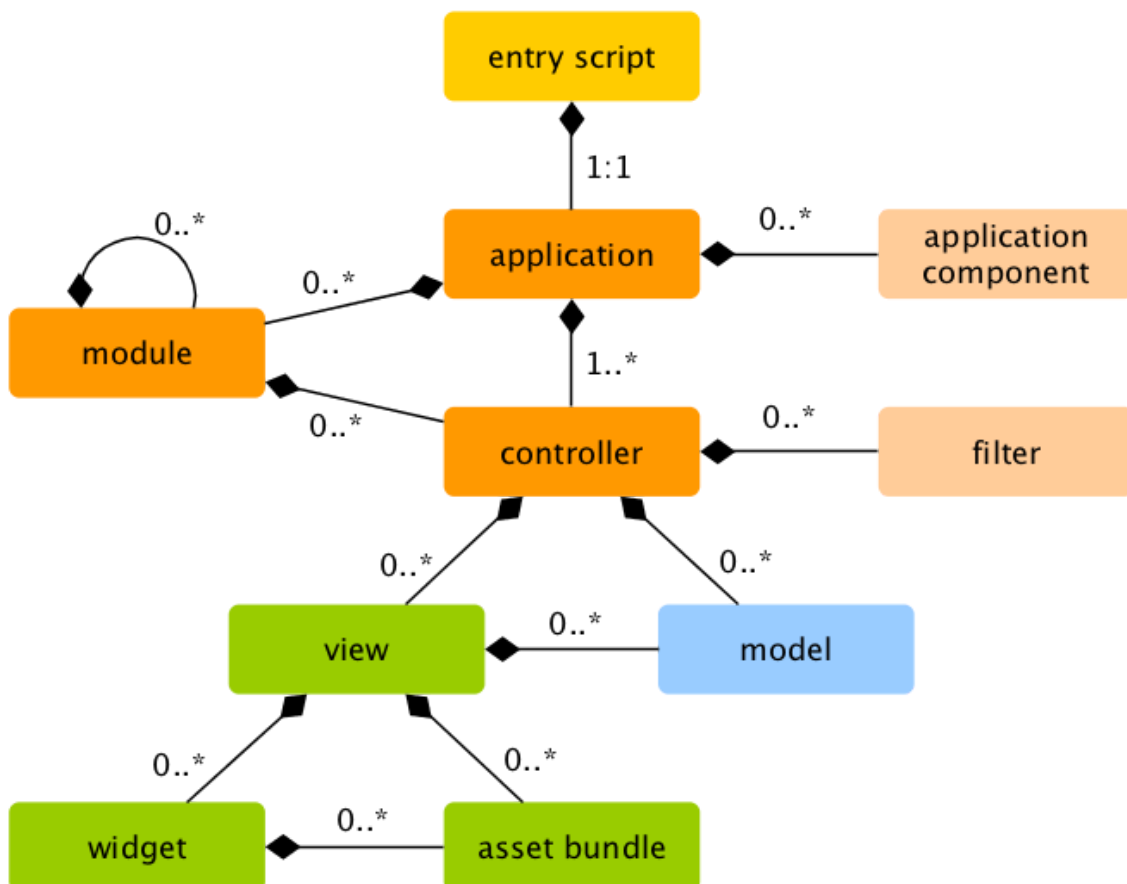

```
\Yii::$container->set('yii\widgets\LinkPager', ['maxButtonCount' => 5]);
```
- También se pueden registrar dependencias en la configuración de la aplicación (`config/web.php` o `config/console.php`):

```
...
'container' => [
    'definitions' => [
        'yii\widgets\LinkPager' => ['maxButtonCount' => 5]
    ],
    'singletons' => [
        // Dependency Injection Container singletons configuration
    ],
],
```

- Más información en el apartado [2.5. Configuraciones](#).

3. Estructura de una aplicación

3.1. Introducción



3.2. Scripts de entrada

- Son el primer paso en el proceso de arranque de la aplicación.
- Una aplicación (ya sea web o de consola) tiene un único script de entrada.
- Los usuarios hacen peticiones al script de entrada, el cual instancia al objeto aplicación y le reenvía a éste la petición.

3.3. Aplicaciones

- Son los objetos que gobiernan la estructura general y el ciclo de vida de una aplicación Yii.
- Cada aplicación Yii contiene un único objeto aplicación que se crea en el script de entrada.

- El objeto aplicación es accesible globalmente a través de la expresión `\Yii::$app`.
- El objeto aplicación es un **localizador de servicios**.

3.4. Componentes de aplicación

- El *objeto aplicación* aloja un conjunto de *componentes de aplicación*, los cuales proporcionan distintos servicios durante el procesamiento de las peticiones.
- Cada componente de aplicación tiene un ID que lo identifica unívocamente entre todos los demás componentes de la misma aplicación.
- Se puede acceder a un componente de aplicación mediante la expresión `\Yii::$app->IDcomponente`.
- Ejemplos:
 - El componente `urlManager` se encarga de enrutar las peticiones web a los controladores apropiados. Se accede mediante `\Yii::$app->urlManager`.
 - El componente `db` proporciona servicios relacionados con bases de datos. Se accede mediante `\Yii::$app->db`.
- Los componentes de aplicación se crean la primera vez que se accede a ellos mediante la expresión anterior. Las demás veces se accede a la misma instancia.

3.5. Controladores

- Son parte de la arquitectura MVC.
- Son instancias de clases que heredan de `yii\base\Controller`.
- Son responsables de procesar las *peticiones* y generar las *respuestas*.
- En concreto, tras recibir el control de la aplicación, el controlador:
 - a. Analizará los datos de la petición entrante.
 - b. Los enviará a los modelos.
 - c. Inyectará los resultados del modelo dentro de las vistas.
 - d. Finalmente, generará la respuesta saliente.

3.6. Modelos

- Son parte de la arquitectura MVC.
- Son objetos que representan:
 - Datos de negocio.
 - Reglas de negocio.
 - Lógica de negocio.
- Son instancias de clases que heredan de `yii\base\Model` (o una subclase).
- La clase base `yii\base\Model` proporciona muchas características útiles:
 - **Atributos**: representan los datos de negocio y se pueden acceder como cualquier otra propiedad del objeto.
 - **Etiquetas de atributos**: especifican las etiquetas que acompañan a los atributos durante su visualización.

- **Asignación masiva:** permite rellenar varios atributos en un solo paso.
- **Reglas de validación:** comprueba si los datos de entrada cumplen las reglas de validación declaradas.
- **Exportación de datos:** permite exportar los datos del modelo en forma de arrays con formatos personalizables.
- La clase `Model` también es la clase base para modelos más avanzados, como **Active Record**.

3.7. Vistas

- Son parte de la arquitectura MVC.
- Representan el código responsable de mostrar datos a los usuarios finales.
- En una aplicación web, las vistas se crean normalmente mediante scripts PHP que contienen, principalmente, código HTML y PHP.
- Son gestionadas por el componente de aplicación `view`, el cual proporciona métodos que facilitan la composición y el dibujo de las vistas.

3.8. Módulos

- Son unidades de software autocontenidas formadas por modelos, vistas, controladores y otros componentes de soporte.
- Los usuarios pueden acceder a los controladores de un módulo cuando está instalado en una aplicación.
- Por ello, se ven como mini-aplicaciones.
- Se diferencian de las aplicaciones en que no pueden desplegarse solos y deben residir dentro de una aplicación.

3.9. Filtros

- Son objetos que se ejecutan antes y/o después de acciones de un controlador.
- Puede consistir en un *pre-filtro* (lógica de filtrado aplicada antes de acciones) y/o un *post-filtro* (lógica aplicada después de acciones).
- En esencia, son un tipo especial de comportamiento. Por tanto, se usan como si fueran comportamientos.
- Ejemplos:
 - Un filtro de control de acceso puede ejecutarse antes de las acciones para garantizar que el acceso sólo está permitido a ciertos usuarios finales
 - Un filtro de compresión de contenido puede ejecutarse después de las acciones para comprimir el contenido de la respuesta antes de enviarla al usuario final.

3.10. Widgets

- Son bloques de construcción reutilizables usados en las vistas para crear elementos de interfaces de usuario complejos y configurables en un estilo orientado a objetos.

- Ejemplo:

```
<?php
use yii\jui\DatePicker;
?>
<?= DatePicker::widget(['name' => 'date']) ?>
```

- Hay un montón de widgets predefinidos en Yii: ActiveForm, menú, widgets de jQuery UI widgets, widgets de Twitter Bootstrap...

3.11. Assets

- Son archivos a los que se les hace referencia desde una página web.
- Ejemplos: un archivo CSS, un archivo JavaScript, una imagen, un vídeo, etc.
- Se almacenan en directorios accesibles por el servidor web y son servidos directamente por éste.
- A menudo es preferible manejar los assets programáticamente.
 - Por ejemplo, cuando se usa el widget `yii\jui\DatePicker` en una página, se incluirán automáticamente los archivos CSS y JavaScript necesarios, en lugar de tener que incluirlos a mano.
 - Cuando se actualice el widget a una nueva versión, usará automáticamente la nueva versión de los assets.

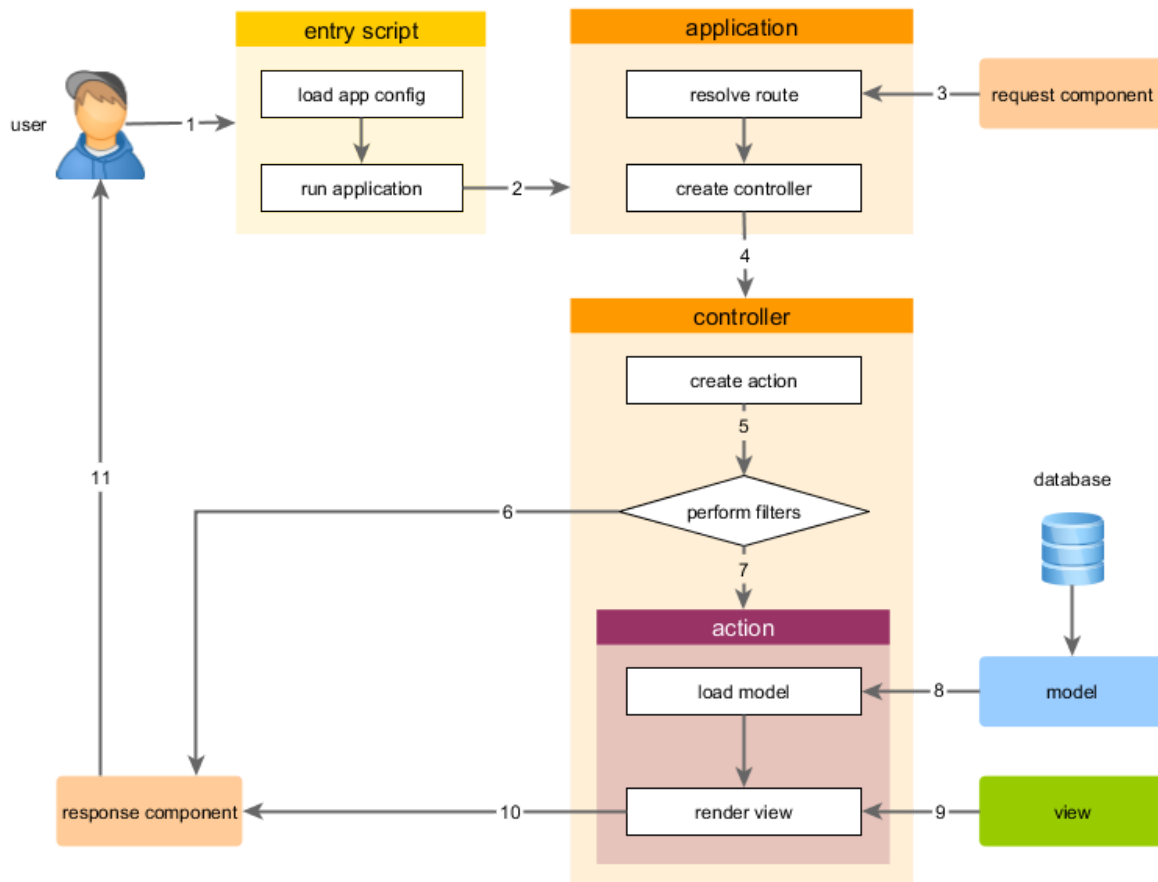
3.12. Extensiones

- Son paquetes de software redistribuibles diseñados específicamente para ser usados en aplicaciones Yii.
- Proporcionan características listas para usar.
- Ejemplo: la extensión `yiiisoft/yii2-debug` añade una barra de herramientas de depuración debajo de cada página de la aplicación para ayudar a captar más fácilmente cómo se generan las páginas.
- Se usan para acelerar el proceso de desarrollo.
- También se puede empaquetar el código propio en forma de extensiones para compartirlo con otras personas.
- Cuando hablamos de un paquete de software de propósito general, no diseñado específicamente para Yii, nos referimos a él simplemente como *paquete* o *biblioteca*.

4. Gestión de peticiones

4.1. Introducción

- Cada vez que una aplicación tiene que manejar una petición, se sigue un flujo similar a éste:
 1. Un usuario hace una petición al script de entrada `web/index.php`.
 2. El script de entrada carga la configuración de la aplicación y crea una instancia de la aplicación para manejar la petición.
 3. La aplicación resuelve la ruta solicitada con ayuda del componente de aplicación `request`.
 4. La aplicación crea una instancia del controlador para manejar la petición.
 5. El controlador crea una instancia de una acción y ejecuta los filtros de esa acción.
 6. Si falla algún filtro, se cancela la acción.
 7. Si pasan todos los filtros, se ejecuta la acción.
 8. La acción carga un modelo de datos, posiblemente desde una base de datos.
 9. La acción renderiza una vista a partir de la información del modelo de datos.
 10. El resultado renderizado se devuelve al componente de aplicación `response`.
 11. El componente `response` envía el resultado renderizado al navegador del usuario.



4.2. Arranque (Bootstrapping)

- Es el proceso que consiste en preparar el entorno antes de que una aplicación empiece a resolver y procesar una petición entrante.
- El arranque se lleva a cabo en dos sitios: el script de entrada y la aplicación:
 - En el script de entrada:
 - Se registran los autoloaders, lo que incluye:
 - El autoloader de Composer.
 - El autoloader de Yii.
 - Se carga la configuración de la aplicación y se crea una instancia de la aplicación.
 - En el constructor de la aplicación:
 1. Se llama a `preInit()`, que configura algunas propiedades de alto nivel de la aplicación, como `basePath`.
 2. Se registra el manejador de errores.
 3. Se inicializan las propiedades de la aplicación usando la configuración dada.
 4. Se llama a `init()`, el cual llama a `bootstrap()` para arrancar los componentes a arrancar durante el bootstrapping.
- Como el proceso de arranque tiene que hacerse antes del manejo de cada petición, es muy importante mantener este proceso lo más ligero y optimizado posible.

4.3. Enrutado y creación de URLs

- Cuando una aplicación Yii empieza a procesar una URL solicitada, el primer paso es analizar la URL y convertirla en una ruta.
 - Esa ruta se usa luego para instanciar la correspondiente *acción de controlador* que manejará la petición.
 - A este proceso completo se le denomina **enrutado**.
- El proceso contrario al enrutado se denomina **creación de URLs**:
 - Consiste en crear una URL a partir de una ruta dada y sus posibles parámetros de consulta asociados.
 - Cuando la URL creada se solicite más adelante, el proceso de enrutado podrá convertirla de nuevo en la combinación original de ruta y parámetros de consulta.
- La pieza central responsable del enrutado y la creación de URLs es el **gestor de URLs**, el cual se registra como el *componente de aplicación* `urlManager`.
- El gestor de URLs proporciona:
 - El método `parseRequest()`, que convierte una petición entrante en una ruta y sus parámetros de consulta asociados.
 - El método `createUrl()`, que crea una URL a partir de una ruta y sus parámetros de consulta asociados.
- Por ejemplo, para crear una URL a partir de la acción `post/view`:

```
use yii\helpers\Url;
// Url::to() calls UrlManager::createUrl() to create a URL
$url = Url::to(['post/view', 'id' => 100]);
```

Dependiendo de la configuración del `urlManager`, la URL creada puede parecerse a alguna de la siguientes:

```
/index.php?r=post%2Fview&id=100
/index.php/post/100
/posts/100
```

4.4. Peticiones

- Las peticiones que se hacen a una aplicación se representan como instancias de la clase `yii\web\Request` y proporcionan datos tales como parámetros de consulta, cabeceras HTTP, cookies, etc.
- Dada una petición, se puede acceder al correspondiente objeto que la contiene a través del componente de aplicación `request`, el cual es una instancia de `yii\web\Request`.
- Diferencia entre `queryParams`, `queryString`, `get()`, `post()` y `bodyParams`:
 - `queryParams` devuelve el array `$_GET`
 - `queryString` devuelve la cadena `$_SERVER['QUERY_STRING']`

- `get()` devuelve `$_GET` o uno de sus elementos
- `post()` devuelve `$_POST` o uno de sus elementos
- `bodyParams` es para PUT, PATCH y otros métodos

4.5. Respuestas

- Cuando una aplicación termina de manejar una petición, genera un objeto respuesta y lo envía al usuario final.
- El objeto respuesta contiene, entre otras cosas:
 - El código de estado HTTP.
 - Las cabeceras HTTP.
 - El cuerpo.
- El objetivo final del desarrollo de una aplicación web es, básicamente, convertir peticiones en objetos respuesta.
- En la mayoría de los casos, se trabaja con el componente de aplicación `response`, que es una instancia de la clase `yii\web\Response`.

4.6. Sesiones y cookies

- Permiten el mantenimiento de datos persistentes entre varias peticiones de usuario.
- En PHP plano se puede acceder a ellos mediante las variables globales `$_SESSION` y `$_COOKIE`.
- Yii encapsula las sesiones y las cookies en forma de objetos.
- Como ocurre con las peticiones y las respuestas, se puede acceder a las sesiones mediante el componente de aplicación `session`, el cual es una instancia de `yii\web\Session`.
- Yii representa cada cookie como un objeto de la clase `yii\web\Cookie`. Tanto `yii\web\Request` como `yii\web\Response` mantienen una colección de cookies por medio de la propiedad llamada `cookies`. En el primero, representa las cookies enviadas en la petición, y en el segundo representa las que se envían al usuario.
- El controlador es la parte de la aplicación que trata directamente con las peticiones y las respuestas. Por tanto, las cookies se deben leer y enviar en el controlador.

5. Trabajo con bases de datos

5.1. Database Access Objects (DAO)

- Se define sobre PDO.

- Proporciona una API orientada a objetos para acceder a bases de datos relacionales.
- Es la base de otros métodos de acceso más avanzados, como *Query Builder* y *Active Record*.
- Se trabaja con **SQL plano y arrays de PHP**, como en PDO.
 - *Ventaja*: Por lo anterior, es el modo más eficiente de acceder a bases de datos.
 - *Inconveniente*: la sintaxis SQL varía de un SGBD a otro.
- El DAO en Yii soporta los siguientes SGBD:
 - MySQL
 - MariaDB
 - SQLite
 - PostgreSQL
 - CUBRID versión >= 9.3
 - Oracle
 - MSSQL versión >= 2008

5.1.1. Ejemplos

```
// return a set of rows. each row is an associative array of column names and values.
// an empty array is returned if the query returned no results
$post = Yii::$app->db->createCommand('SELECT * FROM post')
    ->queryAll();

// return a single row (the first row)
// false is returned if the query has no result
$post = Yii::$app->db->createCommand('SELECT * FROM post WHERE id=1')
    ->queryOne();

// return a single column (the first column)
// an empty array is returned if the query returned no results
$title = Yii::$app->db->createCommand('SELECT title FROM post')
    ->queryColumn();

// return a scalar value
// false is returned if the query has no result
$count = Yii::$app->db->createCommand('SELECT COUNT(*) FROM post')
    ->queryScalar();
```

5.1.2. Vinculación de parámetros

Se implementa por medio de **sentencias preparadas**.

```
$post = Yii::$app->db->createCommand('SELECT * FROM post WHERE id=:id AND status=:status')
    ->bindValue(':id', $_GET['id'])
```

```

->bindValue(':status', 1)
->queryOne();

$params = [':id' => $_GET['id'], ':status' => 1];
$post = Yii::$app->db->createCommand('SELECT * FROM post WHERE id=:id AND status=:status')
->bindValues($params)
->queryOne();

$post = Yii::$app->db->createCommand('SELECT * FROM post WHERE id=:id AND status=:status',
$params)
->queryOne();

$command = Yii::$app->db->createCommand('SELECT * FROM post WHERE id=:id');
$post1 = $command->bindValue(':id', 1)->queryOne();
$post2 = $command->bindValue(':id', 2)->queryOne();

```

5.1.3. Consultas no-SELECT

```

Yii::$app->db->createCommand('UPDATE post SET status=1 WHERE id=1')
->execute();

// INSERT (table name, column values)
Yii::$app->db->createCommand()->insert('user', [
    'name' => 'Sam',
    'age' => 30,
])->execute();

// UPDATE (table name, column values, condition)
Yii::$app->db->createCommand()->update('user', ['status' => 1], 'age > 30')->execute();

// DELETE (table name, condition)
Yii::$app->db->createCommand()->delete('user', 'status = 0')->execute();

// table name, column names, column values
Yii::$app->db->createCommand()->batchInsert('user', ['name', 'age'], [
    ['Tom', 30],
    ['Jane', 20],
    ['Linda', 25],
])->execute();

```

5.2. Query Builder

- Se define sobre *Database Access Objects*.
- Sólo es válido para consultas SELECT. Para INSERT, UPDATE y DELETE hay que usar DAO.
- Permite construir consultas SELECT SQL de forma **programática e independiente del SGBD**.
- El código es más legible y genera consultas SQL más seguras.
- Actúa en dos pasos:

- a. Crear un objeto `yii\db\Query` y construir a partir de él las diferentes partes (SELECT, FROM, etc.) de una consulta SELECT.
 - b. Ejecutar un método de consulta (p.ej. `all()`) de `yii\db\Query` para recuperar los datos de la base de datos.
- Ejemplo:

```
$rows = (new \yii\db\Query())
->select(['id', 'email'])
->from('user')
->where(['last_name' => 'Smith'])
->limit(10)
->all();
```

5.3. Active Record

- Proporciona una interfaz orientada a objetos para acceder y manipular datos de una base de datos.
- Una *clase* de *Active Record* se asocia con una *tabla* de la base de datos.
- Una *instancia* de *Active Record* se corresponde con una *fila* de esa tabla.
- Un *atributo* de una instancia de *Active Record* representa el valor de una *columna* de esa fila.
- Las instancias de *Active Record* se consideran **modelos**. Por esta razón, normalmente se colocan las clases de *Active Record* dentro del espacio de nombres `app\models`.
- Como `yii\db\ActiveRecord` hereda de la clase `yii\base\Model`, hereda de ella todas sus características: atributos, reglas de validación, serialización de datos, etc.
- En lugar de escribir consultas SQL directamente, se puede acceder a los atributos de *Active Record* y llamar a los métodos para acceder y manipular los datos almacenados en las tablas de la base de datos.
- Por ejemplo:
 - Supongamos que `Cliente` es una clase de *Active Record* asociada con la tabla `clientes` y `nombre` es una columna de la tabla `clientes`.
 - Para insertar una nueva fila en la tabla, podemos escribir:

```
$cliente = new Cliente();
$cliente->nombre = 'Pepe';
$clientes->save();
```

Que es equivalente al siguiente código SQL para MySQL, que resulta menos intuitivo, más propenso a errores y con posibles problemas de compatibilidad si usa otro SGBD:

```
$db->createCommand('INSERT INTO `clientes` (`nombre`) VALUES (:nombre)', [
    ':nombre' => 'Pepe',
])->execute();
```

- Para consultar datos provenientes de la base de datos, se dan tres pasos:
 - Crear un nuevo objeto de consulta (de tipo `\yii\db\ActiveQuery`) llamando al método `yii\db\ActiveRecord::find()`.
 - Dar forma al objeto Query llamando a los métodos de *Query Builder*.
 - Llamar a un método para recuperar los datos en forma de **instancias de *Active Record***.
- Como puede verse, es un procedimiento muy similar al que se usa con *Query Builder*. La única diferencia es que, en lugar de usar el operador `new` para crear un objeto `yii\db\Query`, se llama a `yii\db\ActiveRecord::find()` para obtener un nuevo objeto de la clase `yii\db\ActiveQuery`. La otra diferencia es que el resultado son objetos, no arrays.
- Como `yii\db\ActiveQuery` hereda de `yii\db\Query`, se pueden usar todos los métodos relativos a *Query Builder*.
- Ejemplos:

```
// return a single customer whose ID is 123
// SELECT * FROM `customer` WHERE `id` = 123
$customer = Customer::find()
    ->where(['id' => 123])
    ->one();
```

```
// return all active customers and order them by their IDs
// SELECT * FROM `customer` WHERE `status` = 1 ORDER BY `id`
$customers = Customer::find()
    ->where(['status' => Customer::STATUS_ACTIVE])
    ->orderBy('id')
    ->all();
```

```
// return the number of active customers
// SELECT COUNT(*) FROM `customer` WHERE `status` = 1
$count = Customer::find()
    ->where(['status' => Customer::STATUS_ACTIVE])
    ->count();
```

```
// return all customers in an array indexed by customer IDs
// SELECT * FROM `customer`
$customers = Customer::find()
    ->indexBy('id')
    ->all();
```

6. Recogida de datos del usuario

6.1. Creación de formularios

- La principal forma de crear formularios en Yii es a través de `yii\widgets\ActiveForm`.
- Es la forma preferida cuando el formulario se basa en un modelo.
- Si no, también hay otros métodos útiles en `yii\helpers\Html` que se usan normalmente para añadir botones y texto de ayuda a cualquier formulario.
- En la mayoría de los casos, un formulario tiene su correspondiente **modelo** que se usa para validar en el servidor los datos introducidos.
- Para crear un formulario basado en modelos, el primer paso es definir ese modelo.
 - Puede estar basado en una clase *Active Record*.
 - O puede basarse en la clase genérica `yii\base\Model` si se van a recolectar datos arbitrarios (por ejemplo, un formulario de *login*).
- Ejemplo:

- Modelo:

```
<?php

class LoginForm extends \yii\base\Model
{
    public $username;
    public $password;

    public function rules()
    {
        return [
            // define validation rules here
        ];
    }
}
```

- Vista:

```
<?php
use yii\helpers\Html;
use yii\widgets\ActiveForm;

$form = ActiveForm::begin([
    'id' => 'login-form',
]); ?>

<?= $form->field($model, 'username') ?>
<?= $form->field($model, 'password')->passwordInput() ?>

<div class="form-group">
    <div class="col-lg-offset-1 col-lg-11">
        <?= Html::submitButton('Login', ['class' => 'btn
btn-primary']) ?>
    </div>
```

```

    </div>
<?php ActiveForm::end() ?>

```

- `ActiveForm::begin()` crea una instancia de formulario (clase `yii\widgets\ActiveForm`) y al mismo tiempo marca el comienzo del mismo.
- Todo el contenido que haya entre `ActiveForm::begin()` y `ActiveForm::end()` irá encerrado entre `<form>` y `</form>`.
- `ActiveForm::field()` crea un nuevo campo de formulario, con su etiqueta correspondiente y todo el código JavaScript asociado que necesite para validación de cliente.
 - Lo hace creando una instancia de la clase `yii\widgets\ActiveField` que, cuando se muestra a la salida (con `echo` o `<?=>`), se transforma en el código necesario que se volcará a la salida.

6.2. Validación de la entrada

- Dado un modelo rellenado con entradas del usuario, se pueden validar dichas entradas llamando al método `yii\base\Model::validate()`.
- El método devuelve un valor lógico que indica si la validación tiene éxito o no.
- En caso de que no, se pueden consultar los mensajes de error correspondientes en la propiedad `yii\base\Model::$errors`.
- Ejemplo:

```

$model = new \app\models\ContactForm();

// populate model attributes with user inputs
$model->load(\Yii::$app->request->post());
// which is equivalent to the following:
// $model->attributes = \Yii::$app->request->post('ContactForm');

if ($model->validate()) {
    // all inputs are valid
} else {
    // validation failed: $errors is an array containing error messages
    $errors = $model->errors;
}

```

- Para declarar las reglas de validación, se sobrescribe el método `yii\base\Model::rules()`:

```

public function rules()
{
    return [
        // the name, email, subject and body attributes are required
        [['name', 'email', 'subject', 'body'], 'required'],

        // the email attribute should be a valid email address
        ['email', 'email'],
    ];
}

```

6.2.1. Validaciones ad-hoc

- A veces hay que llevar a cabo *validaciones ad-hoc* sobre valores que no están vinculados a ningún modelo.
- Si sólo hay que hacer un tipo de validación (p.ej. validar direcciones de email), se puede llamar al método `validate()` del validador deseado, así:

```
$email = 'test@example.com';  
$validator = new yii\validators\EmailValidator();  
  
if ($validator->validate($email, $error)) {  
    echo 'Email is valid.';  
} else {  
    echo $error;  
}
```

- **Nota:** No todos los validadores soportan este tipo de validación. Por ejemplo, el validador integrado `unique` sólo funciona con un modelo.
- Si hay que llevar a cabo varias validaciones sobre varios valores, se puede usar `yii\base\DynamicModel` para declarar atributos y reglas al mismo tiempo y sobre la marcha. Se usa así:

```
public function actionSearch($name, $email)  
{  
    $model = DynamicModel::validateData(compact('name', 'email'), [  
        [['name', 'email'], 'string', 'max' => 128],  
        ['email', 'email'],  
    ]);  
  
    if ($model->hasErrors()) {  
        // validation fails  
    } else {  
        // validation succeeds  
    }  
}
```

- El método `yii\base\DynamicModel::validateData()` crea una instancia de la clase `DynamicModel`, define los atributos usando los datos enviados (`name` y `email` en este ejemplo) y luego llama a `yii\base\Model::validate()` con las reglas indicadas.

7. Visualización de datos

7.1. Formateado de datos

- Para trabajar correctamente con fechas, horas, instantes y zonas horarias:
 - Configurar PostgreSQL para trabajar con `timezone = 'UTC'` en `postgresql.conf` (por defecto vale `'localtime'`, tomando la del sistema). En Heroku está correcto ya.
 - Como en PostgreSQL vamos a trabajar siempre en UTC, no tiene mucho sentido usar columnas de tipo `timestamptz`, ya que PostgreSQL nunca va a tener que hacer ningún tipo de conversión de UTC a/desde otra zona horaria. Por eso se puede usar `timestamp` sin zona horaria perfectamente. Mejor aún, `timestamp(0)` para no guardar millonésimas de segundos.
 - Configurar PostgreSQL para trabajar con `intervalstyle = 'iso_8601'` en `postgresql.conf` (por defecto vale `'postgres'`). En Heroku tiene el mismo valor por defecto, por lo que tenemos que cambiarlo cada vez que establecemos una conexión a la base de datos. Para ello, añadimos en la configuración de la aplicación:

```
'db' => [  
    // ...  
    'on afterOpen' => function ($event) {  
        $event->sender->createCommand("SET intervalstyle = 'iso_8601'");  
    }  
]
```
 - Configurar PHP para trabajar con `date.timezone = 'UTC'` en `php.ini` (por defecto toma la zona horaria del sistema). En Heroku está correcto ya.
 - Configurar el componente de aplicación `formatter` en `config/web.php` de esta forma:

```
'components' => [  
    'formatter' => [  
        'timeZone' => 'Europe/Madrid',  
    ],  
],
```
 - **NO** se configura la zona horaria a nivel de aplicación con la propiedad `Application::timeZone`. Se configura sólo en el `formatter`, ya que sólo necesitamos la zona horaria del usuario al visualizar las fechas. Si lo ponemos a nivel de aplicación, desde el primer momento el framework hará un [date default timezone set\(Yii::\\$app->timeZone\)](#), y estaremos trabajando siempre en la zona horaria definida, en lugar de trabajar en UTC, como debería ser.
 - Por contra, si cada usuario configura su propia zona horaria, se puede dejar la propiedad `timeZone` sin definir y añadir lo siguiente en `config/web.php`:

```
'on beforeRequest' => function ($event) {  
    Yii::$app->formatter->timeZone =  
        Yii::$app->user->isGuest ?
```



```

        'Europe/Madrid' :
        Yii::$app->user->identity->zona_horaria;
    }

```

- Si recibimos una fecha formateada en forma de cadena con nombres en español, como por ejemplo '25-enero-2018 17:01:43', podemos usar la extensión [Intl](#):

```

$fmt = new IntlDateFormatter(
    'es-ES',
    IntlDateFormatter::LONG,
    IntlDateFormatter::LONG,
    null,
    null,
    'dd-MMMM-yyyy HH:mm:ss'
);
$fmt->parse('25-enero-2018 17:01:43'); // => 1516896103

```

7.2. Paginación

- Yii usa un objeto `yii\data\Pagination` para representar la información de un esquema de paginación, que es:
 - `totalCount` indica el número total de elementos de datos. Normalmente es mucho mayor que el número de elementos de datos necesarios para visualizar una sola página.
 - `pageSize` indica cuántos elementos de datos contiene cada página. Por defecto es 20.
 - `page` indica el número de página actual (se cuenta a partir de cero). Por defecto es 0, que representa la primera página.
- Con un objeto `yii\data\Pagination` totalmente especificado, se pueden recuperar y visualizar los datos parcialmente. Por ejemplo, si estás recuperando datos de una base de datos, se pueden usar `OFFSET` y `LIMIT` para consultar la base de datos correctamente. Por ejemplo:

```

use yii\data\Pagination;

// build a DB query to get all articles with status = 1
$query = Article::find()->where(['status' => 1]);

// get the total number of articles (but do not fetch the article data yet)
$count = $query->count();

// create a pagination object with the total count
$pagination = new Pagination(['totalCount' => $count]);

// limit the query using the pagination and retrieve the articles
$articles = $query->offset($pagination->offset)
    ->limit($pagination->limit)
    ->all();

```

- La página que se recupera en cada momento depende del parámetro `page` indicado en la petición GET. Por defecto, la paginación intentará asignar al atributo `page` el valor del parámetro `page`. Si no se ha indicado dicho parámetro, por defecto tomará el 0.
- Por tanto, no funcionará bien desde la shell (`./yii shell`), porque una aplicación de consola no tiene el componente de aplicación request para recoger los valores de la petición GET (no se mira el array `$_GET`). Se puede asignar a la propiedad `params` un array en el que buscará el parámetro `page` en lugar de buscarlo en la petición.
- Yii proporciona el widget `yii\widgets\LinkPager` que visualiza una lista de botones de páginas para que los usuarios puedan pulsar e indicar qué página quieren visualizar. El widget usa un objeto de paginación. Por ejemplo:

```
use yii\widgets\LinkPager;
echo LinkPager::widget([
    'pagination' => $pagination,
]);
```

7.3. Ordenación

- Cuando se visualizan varias filas de datos, a menudo se necesita mostrarlas ordenadas en función de algunas columnas indicadas por los usuarios finales. Yii usa un objeto `yii\data\Sort` para representar la información de un esquema de ordenación. En concreto:
 - `attributes` indica los *atributos* por los que se podrán ordenar los datos. Un atributo puede ser tan simple como un *atributo de un modelo*, o puede estar compuesto de varios atributos de un modelo o de varias columnas de base de datos.
 - `attributeOrders` indica las direcciones de ordenación solicitadas actualmente para cada atributo.
 - `orders` indica las direcciones de ordenación en términos de las columnas de bajo nivel.
- Para usar `yii\data\Sort`, primero se declaran qué atributos pueden ser ordenados. Después, se recoge la información de ordenación solicitada actualmente mediante `attributeOrders` u `orders` y se usa para personalizar la consulta. Por ejemplo:

```
use yii\data\Sort;
$sort = new Sort([
    'attributes' => [
        'age',
        'name' => [
            'asc' => ['first_name' => SORT_ASC, 'last_name' => SORT_ASC],
            'desc' => ['first_name' => SORT_DESC, 'last_name' => SORT_DESC],
            'default' => SORT_DESC,
            'label' => 'Name',
        ],
    ],
```

```

    ],
  });
  $articles = Article::find()
    ->where(['status' => 1])
    ->orderBy($sort->orders)
    ->all();

```

- En el ejemplo de arriba se declaran dos atributos en el objeto `Sort`: `age` y `name`.
- El atributo `age` es un atributo *simple* y corresponde al atributo `age` de la clase de Active Record `Article`. Equivale a la siguiente declaración:

```

'age' => [
  'asc' => ['age' => SORT_ASC],
  'desc' => ['age' => SORT_DESC],
  'default' => SORT_ASC,
  'label' => Inflector::camel2words('age'),
]

```

- Cuando el objeto ya ha sido creado, al asignarle un valor a la propiedad `attributes` hay que usar forzosamente la declaración completa. No vale la sintaxis abreviada que se ha usado arriba en `age`.
- El atributo `name` es un atributo *compuesto* definido en función de `first_name` y `last_name` de `Article`. Se declara usando el siguiente array:
 - Los elementos `asc` y `desc` especifican cómo ordenar el atributo en las direcciones ascendente y descendente, respectivamente. Sus valores representan las columnas reales y las direcciones por las que se deben ordenar los datos. Se pueden especificar una o más columnas indicando ordenación simple o compuesta.
 - El elemento `default` especifica la dirección por la que se debe ordenar el atributo cuando se solicite por primera vez. Por defecto, el orden es ascendente.
 - El elemento `label` especifica la etiqueta que se debe usar cuando se llame a `\yii\data\Sort::link()` para crear un hipervínculo de ordenación. Por defecto, se llama a `\yii\helpers\Inflector::camel2words()` para generar una etiqueta a partir del nombre del atributo. No se codifica en HTML.
- Se puede usar directamente el valor de `orders` al consultar la base de datos para definir la cláusula `ORDER BY`. No se debe usar `attributeOrders` porque es posible que alguno de los atributos sean compuestos y la consulta de base de datos no los reconozca.
- `\yii\data\Sort` comprueba el parámetro `sort` de la petición GET para determinar qué atributos se deben ordenar. Se puede especificar una ordenación por defecto mediante `\yii\data\Sort::$defaultOrder` que se usará cuando no exista ese parámetro. También se puede cambiar el nombre del parámetro configurando la propiedad `sortParam`.
- Por tanto, no funcionará bien desde la shell (`./yii shell`), porque una aplicación de consola no tiene el componente de aplicación `request` para recoger los valores

de la petición GET (no se mira el array `$_GET`). Se puede asignar a la propiedad `params` un array en el que buscará el parámetro `sort` en lugar de buscarlo en la petición.

- En lugar de `orders` se puede usar `getOrders(true)` para obligar a recalcular el su valor.

7.4. Proveedores de datos

- Como las tareas de paginar y ordenar información se dan juntas frecuentemente, en Yii existe el concepto de *proveedor de datos* para encapsular ambos.
- Un proveedor de datos es una clase que implementa la interfaz `yii\data\DataProviderInterface`. Recupera información de forma paginada y ordenada. Normalmente se usan con `widgets de datos` para que los usuarios finales puedan paginar y ordenar datos de forma interactiva.
- En Yii existen los siguientes proveedores de datos:
 - `yii\data\ActiveDataProvider`: usa `yii\db\Query` o `yii\db\ActiveQuery` para recuperar información de bases de datos y la devuelve en forma de arrays o instancias de `Active Record`.
 - `yii\data\SqlDataProvider`: ejecuta una sentencia SQL y devuelve información de base de datos en forma de arrays.
 - `yii\data\ArrayDataProvider`: recibe un array grande y devuelve una rodaja suya en base a las especificaciones de paginación y ordenación.
- Todos los proveedores de datos se usan de la misma forma:

```
// create the data provider by configuring its pagination and sort properties
$provider = new XYZDataProvider([
    'pagination' => [...],
    'sort' => [...],
]);

// retrieves paginated and sorted data
$models = $provider->getModels();
// get the number of data items in the current page
$count = $provider->getCount();
// get the total number of data items across all pages
$totalCount = $provider->getTotalCount();
```

- El comportamiento de la paginación y la ordenación del proveedor de datos se especifica configurando sus propiedades `pagination` y `sort`, que representan las configuraciones de `yii\data\Pagination` y `yii\data\Sort`, respectivamente. Poniendo alguno de ellos a `false` se desactiva.
- Los `widgets de datos`, como por ejemplo `yii\grid\GridView`, tienen una propiedad llamada `dataProvider` que recibe una instancia de un proveedor de datos y visualiza los datos que proporciona. Por ejemplo:

```
echo yii\grid\GridView::widget([
    'dataProvider' => $dataProvider,
]);
```

7.5. Widgets

7.6. Scripts de cliente

8. Correo electrónico

- Crear una cuenta de correo en Gmail, activar la autenticación en dos pasos y crear una contraseña de aplicación. Usar la contraseña de aplicación en lugar de la contraseña normal de la cuenta evita muchos problemas posteriores.

- En config/web.php:

```
'mail' => [
    'class' => 'yii\swiftmailer\Mailer',
    'viewPath' => '@app/mail',
    //set this property to false to send mails to real email addresses:
    'useFileTransport' => false,
    //comment the following array to send mail using php's mail function:
    'transport' => [
        'class' => 'Swift_SmtpTransport',
        'host' => 'smtp.gmail.com',
        'username' => 'username@gmail.com',
        'password' => 'password',
        'port' => '587',
        'encryption' => 'tls',
    ],
],
```

- En el controlador:

```
Yii::$app->mailer->compose()
->setFrom('from@domain.com')
->setTo($form->email)
->setSubject($form->subject)
->setTextBody('Plain text content')
->setHtmlBody('<b>HTML content</b>')
->send();
```


- Otro ejemplo:



```
$message = \Yii::$app->mail->compose('your_view', ['params' => $params]);
if (Yii::$app->user->isGuest) {
    $message->setFrom('from@domain.com');
} else {
    $message->setFrom(Yii::$app->user->identity->email);
}
$message->setTo('to_email@xx.com')
```

```
->setSubject('This is a test mail ' )  
->send();
```

- Si no se envía el correo y llega un mensaje de vuelta parecido al siguiente, es que tu correo ha sido identificado como SPAM por Gmail y se ha bloqueado. Prueba escribiendo un mensaje diferente, quizás más largo o con otras palabras:

Message subject Papelera x

 **iesdonana2018@gmail.com**
Plain text content

 **Mail Delivery Subsystem** <mailer-daemon@googlemail.com>
para mí 



El mensaje se ha bloqueado

Tu mensaje para **ricardo@iesdonana.org** se ha bloqueado.
Consulta más información en los siguientes datos técnicos.

[MÁS INFORMACIÓN](#)

La respuesta fue:

Message rejected. See <https://support.google.com/mail/answer/69585> for more information.

Final-Recipient: rfc822; ricardo@iesdonana.org

Action: failed

Status: 5.0.0

Diagnostic-Code: smtp; Message rejected. See <https://support.google.com/mail/answer/69585> for more information.

Last-Attempt-Date: Tue, 13 Feb 2018 02:42:18 -0800 (PST)

9. Seguridad

9.1. Autenticación

- Es el proceso de verificar la identidad de un usuario. Normalmente se usa un *identificador* (p.ej. un nombre de usuario o una dirección de email) y un *secreto* (p.ej. una contraseña o un token de acceso) para determinar si el usuario es quien dice que es.
- Es la base del login.
- Yii usa varios componentes interconectados para hacer el login. Para que funcione, hay que:

- Configurar el componente de aplicación `user`.
- Crear una clase que implemente la interfaz `yii\web\IdentityInterface`.

9.1.1. Configuración de `yii\web\User`

- El componente de aplicación `user` gestiona y mantiene el estado de autenticación del usuario. Necesita que se especifique una `clase identidad` que contendrá la lógica de autenticación. En la siguiente configuración de aplicación, se indica que la `clase identidad` para `user` será `app\models\User`:

```
return [
    'components' => [
        'user' => [
            'identityClass' => 'app\models\User',
        ],
    ],
];
```

9.1.2. Implementación de `yii\web\IdentityInterface`

- La `clase identidad` debe implementar la interfaz `yii\web\IdentityInterface`, que incluye los siguientes métodos:
 - `findIdentity()`: busca una instancia de la clase identidad usando el ID de usuario indicado. Este método se usa cuando se necesita mantener el estado del login mediante sesiones.
 - `findIdentityByAccessToken()`: busca una instancia de la clase identidad usando el token de acceso indicado. Se usa cuando hay que autenticar a un usuario mediante un único token secreto (p.ej. en una aplicación RESTful sin estado).
 - `getId()`: devuelve el ID del usuario representado por esa instancia de identidad.
 - `getAuthKey()`: devuelve una clave usada para verificar logins basados en cookies. La clave se almacena en la cookie del login y luego se compara con la versión del servidor para garantizar que la cookie es válida. Este método, junto con el siguiente, se usa para implementar la función “recuérdame” del login (autologin).
 - `validateAuthKey()`: implementa la lógica para verificar la clave del login basado en cookies.
- Si no es necesario alguno de esos métodos, se puede implementar con un cuerpo vacío. Por ejemplo, si es una aplicación RESTful pura sin estado, sólo será necesario implementar `findIdentityByAccessToken()` y `getId()` dejando los demás métodos con un cuerpo vacío.
- Ejemplo de implementación de una `clase identidad` como una clase `Active Record` asociada con la tabla `user` de la base de datos:

```
<?php
```

```
use yii\db\ActiveRecord;
```

```

use yii\web\IdentityInterface;

class User extends ActiveRecord implements IdentityInterface
{
    public static function tableName()
    {
        return 'user';
    }

    /**
     * Finds an identity by the given ID.
     *
     * @param string|int $id the ID to be looked for
     * @return IdentityInterface|null the identity object that matches the given ID.
     */
    public static function findIdentity($id)
    {
        return static::findOne($id);
    }

    /**
     * Finds an identity by the given token.
     *
     * @param string $token the token to be looked for
     * @return IdentityInterface|null the identity object that matches the given token.
     */
    public static function findIdentityByAccessToken($token, $type = null)
    {
        return static::findOne(['access_token' => $token]);
    }

    /**
     * @return int|string current user ID
     */
    public function getId()
    {
        return $this->id;
    }

    /**
     * @return string current user auth key
     */
    public function getAuthKey()
    {

```



```

        return $this->auth_key;
    }

    /**
     * @param string $authKey
     * @return bool if auth key is valid for current user
     */
    public function validateAuthKey($authKey)
    {
        return $this->getAuthKey() === $authKey;
    }
}

```

- No confundir la clase identidad `User` con `yii\web\User`. La primera es la clase que implementa la lógica de autenticación, y suele ser una clase `Active Record` asociada con algún tipo de almacenamiento persistente para almacenar las credenciales del usuario. La segunda es la clase del componente de aplicación responsable de gestionar el estado de autenticación del usuario.
- Para generar la clave de autenticación (*auth key*) se puede usar algo así:

```

/**
 * Generates "remember me" authentication key
 */

public function generateAuthKey()
{
    $this->auth_key = Yii::$app->security->generateRandomString();
}

```

9.1.3. Usar `yii\web\User`

- Normalmente se usa `yii\web\User` con el componente de aplicación `user`.
- Se puede recuperar la identidad del usuario actual mediante `Yii::$app->user->identity`, que devuelve una instancia de la `clase identidad` que representa el usuario actual, o `null` si no hay ningún usuario autenticado (o sea, un invitado).
- Cómo recuperar información de autenticación mediante `yii\web\User`:

```

// the current user identity. `null` if the user is not authenticated.
$identity = Yii::$app->user->identity;

// the ID of the current user. `null` if the user not authenticated.
$id = Yii::$app->user->id;

```

```
// whether the current user is a guest (not authenticated)
$isGuest = Yii::$app->user->isGuest;
```

- Para loguear a un usuario, se puede hacer lo siguiente:

```
// find a user identity with the specified username.
// note that you may want to check the password if needed
$identity = User::findOne(['username' => $username]);

// logs in the user
Yii::$app->user->login($identity);
```

- El método `yii\web\User::login()` asigna la identidad del usuario actual en `yii\web\User`. Si las sesiones están **habilitadas**, se guardará la identidad dentro de la sesión, por lo que el estado de autenticación se mantendrá a lo largo de toda la sesión. Si el login basado en cookies (p.ej. un login del tipo “recuérdame”) está **habilitado**, además se guardará la identidad en una cookie para que el estado de autenticación del usuario se pueda recuperar a partir de la cookie mientras la cookie sea válida.
- Para habilitar el login basado en cookies, hay que poner `yii\web\User::$enableAutoLogin` a **true** en la configuración de la aplicación. Además hay que indicar como segundo parámetro el tiempo de duración de la cookie al llamar al método `yii\web\User::login()`.
- Para desconectar a un usuario, simplemente se hace:

```
Yii::$app->user->logout();
```

- Desconectar a un usuario sólo tiene sentido cuando están habilitadas las sesiones. El método limpiará el estado de autenticación del usuario tanto de la memoria como de la sesión. Además, por defecto también destruirá todos los datos de sesión del usuario. Si se desea mantener los datos de la sesión, hay que hacer `Yii::$app->user->logout(false)`.

9.2. Autorización

- Es el proceso que consiste en verificar si un usuario tiene permiso para hacer algo.
- Yii proporciona dos métodos de autorización:
 - Filtrado de control de acceso (Access Control Filter, ACF)
 - Control de acceso basado en roles (Role-Based Access Control, RBAC)

9.2.1. Filtrado de control de acceso (ACF)

- Es un método de autorización sencillo implementado en `yii\filters\AccessControl`.
- Se usa sobre todo en aplicaciones que sólo necesitan un método simple de control de acceso.
- Es un **filtro de acción** que puede usarse en un controlador o un módulo.

- Cuando un usuario solicita la ejecución de una acción, ACF comprueba una lista de reglas de acceso para determinar si el usuario puede acceder a la acción solicitada.
- Con `matchCallback()` se pueden implementar muchas reglas complejas con facilidad, sin tener que usar el RBAC.
- Ejemplo:

```
use yii\web\Controller;
use yii\filters\AccessControl;

class SiteController extends Controller
{
    public function behaviors()
    {
        return [
            'access' => [
                'class' => AccessControl::className(),
                'only' => ['login', 'logout', 'signup'],
                'rules' => [
                    [
                        'allow' => true,
                        'actions' => ['login', 'signup'],
                        'roles' => ['?'],
                    ],
                    [
                        'allow' => true,
                        'actions' => ['logout'],
                        'roles' => ['@'],
                    ],
                ],
            ],
        ];
    }
    // ...
}
```

9.2.2. Control de acceso basado en roles (RBAC)

- <https://github.com/mdmsoft/yii2-admin>
- <https://github.com/dektrium/yii2-user>
- <https://github.com/dektrium/yii2-rbac>

9.3. Contraseñas

- Las contraseñas no se deben guardar como texto plano.
- Hay quien todavía piensa que es seguro guardar contraseñas cifradas con md5 o sha1.
 - Actualmente, el hardware moderno hace posible invertir esos *hashes* (y otros incluso más fuertes) muy rápidamente usando ataques de fuerza bruta.

- Para mejorar la seguridad de las contraseñas de usuario, hay que usar un algoritmo de *hashing* resistente a ataques por fuerza bruta.
- La mejor elección actualmente es **bcrypt**.
- En PHP, se puede crear un *hash bcrypt* usando la función `crypt()`.
- Yii proporciona dos funciones auxiliares que usan `crypt()` para generar y verificar *hashes* de forma segura más fácilmente.
- Cuando un usuario proporciona una contraseña la primera vez (p.ej. durante su registro), hay que cifrar la contraseña:

```
$hash = Yii::$app->security->generatePasswordHash($password);
```

- Luego, el *hash* se puede asociar con el correspondiente atributo del modelo, para que se pueda almacenar en la base de datos para su posterior uso.
- Cuando un usuario intente iniciar sesión, se verificará la contraseña suministrada contra el *hash* almacenado previamente:

```
if (Yii::$app->security->validatePassword($password, $hash)) {
    // all good, logging user in
} else {
    // wrong password
}
```

9.4. Criptografía

9.5. Seguridad en vistas

- Al crear vistas que generen páginas HTML, es importante codificar y/o filtrar los datos provenientes de los usuarios finales antes de mostrarlos.
- En caso contrario, la aplicación estaría expuesta a ataques de *Cross-Site Scripting* (XSS).
- Para mostrar texto plano, primero se debe codificar llamando a `yii\helpers\Html::encode()`.

- Ejemplo:

```
<?php
use yii\helpers\Html;
?>

<div class="username">
    <?= Html::encode($user->name) ?>
</div>
```

- Al visualizar contenido HTML, se puede usar `yii\helpers\HtmlPurifier` para filtrar el contenido previamente.

- Ejemplo:

```
<?php
use yii\helpers\HtmlPurifier;
?>

<div class="post">
```

```
<?= HtmlPurifier::process($post->text) ?>
</div>
```

- HTMLPurifier funciona muy bien, pero es lento. Es importante *cachear* los resultados.

9.6. Clientes de autenticación

9.7. Buenas prácticas

9.7.1. Principios básicos

- Hay dos principios básicos de seguridad a tener en cuenta en cualquier aplicación:
 - a. Filtrar la entrada.
 - b. Escapar la salida.

9.7.1.1. Filtrar la entrada

- Significa que la entrada nunca debe considerarse segura y siempre se debe comprobar si el valor obtenido se encuentra entre los permitidos.
- Por ejemplo: si sabemos que se puede ordenar por los campos `title`, `created_at` y `status`, y el campo va a ser suministrado a través de la entrada, es mejor comprobar que el valor es correcto.
- En PHP, se haría así:

```
$sortBy = $_GET['sort'];
if (!in_array($sortBy, ['title', 'created_at', 'status'])) {
    throw new Exception('Invalid sort value.');
```

- En Yii, se hace mediante *validación de formularios*.

9.7.1.2. Escapar la salida

- Significa que, dependiendo del contexto en el que estemos usando los datos, habría que escaparlos adecuadamente.
- Por ejemplo:
 - En el contexto del HTML deberíamos escapar los caracteres `<`, `>` y similares.
 - En el contexto de JavaScript o SQL serían otros caracteres diferentes.
- Escapar la salida es un proceso propenso a errores si se hace manualmente.
- Yii proporciona varias herramientas para escapar la salida en diferentes contextos.

9.7.2. Evitar inyecciones de SQL

- En Yii, la mayoría de las consultas a bases de datos se realizan mediante *Active Record*, el cual usa internamente **sentencias preparadas PDO**, lo que evita las inyecciones de SQL.
- A veces hay que hacer consultas mediante *Data Access Objects* o *Query Builder*. En estos casos, hay que usar formas seguras de pasar los datos.
- Si los datos se usan como valores de columnas, es recomendable usar sentencias preparadas:

```
// query builder
$userIDs = (new Query())
    ->select('id')
    ->from('user')
    ->where('status=:status', [':status' => $status])
    ->all();
```

```
// DAO
$userIDs = $connection
    ->createCommand('SELECT id FROM user where status=:status')
    ->bindValues([':status' => $status])
    ->queryColumn();
```

- Si los datos se usan para especificar nombres de columnas o tablas, lo mejor es permitir sólo un conjunto predefinido de valores:

```
function actionList($orderBy = null)
{
    if (!in_array($orderBy, ['name', 'status'])) {
        throw new BadRequestHttpException('Only name and status are
allowed to order by.');
```

```
    }
    // ...
}
```

9.7.3. Evitar XSS

- El XSS (o *cross-site scripting*) aparece si no se escapa adecuadamente la salida cuando se envía HTML al navegador.
- Por ejemplo: si un usuario, en lugar de introducir su nombre, introduce `<script>alert('Hello!');</script>`, cada vez que se muestre su nombre sin escapar se ejecutará el código JavaScript `alert('Hello!');` mostrando un mensaje de alerta en el navegador. Dependiendo del sitio web, en lugar de una alerta inocente podría enviar mensajes usando tu nombre o incluso hacer transacciones bancarias.
- Evitar el XSS es bastante fácil en Yii. Normalmente hay dos casos:
 1. Se desea visualizar datos como texto plano.

2. Se desea visualizar datos como HTML.
1. Si sólo se desea visualizar texto plano, escaparlos es tan sencillo como hacer:

```
<?= \yii\helpers\Html::encode($username) ?>
```

2. Si debe ser HTML, podemos ayudarnos de:

```
<?= \yii\helpers\HtmlPurifier::process($description) ?>
```

Téngase en cuenta que el procesamiento que hace `HtmlPurifier` es bastante pesado, por lo que se debe considerar usar cacheado.

9.7.4. Evitar CSRF

- El *CSRF* (o *cross-site request forgery*) está tras la idea de que muchas aplicaciones presuponen que las peticiones que provienen de un navegador están realizadas por el propio usuario.
- Eso puede no ser verdad.
- Por ejemplo: el sitio `un.ejemplo.com` tiene una URL `/logout`, el cual, cuando es accedida mediante GET, finaliza la sesión del usuario. Mientras sea el usuario el que lo solicite, todo está bien, pero un día, un atacante envía un post con el contenido `` en un foro que el usuario visita frecuentemente. El navegador no distingue entre pedir una imagen o una página, por lo que, cuando el usuario abre una página con esa etiqueta ``, el navegador enviará una petición GET a esa URL, y el usuario se desconectará de `un.ejemplo.com`.
- Alguien podría decir que desconectar a un usuario no es nada serio, pero los atacantes pueden hacer mucho más aplicando esta simple idea.
- Por ejemplo: un sitio tiene una URL `http://un.ejemplo.com/cuenta/transferir?a=otroUsuario&cantidad=2000`. Al acceder a ella mediante GET, se transfieren 2000 € al usuario `otroUsuario` desde la cuenta de usuario autorizada.
 - Sabemos que el navegador siempre envía una petición GET para cargar una imagen, por lo que podemos modificar el código para que sólo acepte peticiones POST a esa URL.
 - Por desgracia, eso no nos salva, porque un atacante puede poner código JavaScript en lugar de la etiqueta ``, lo que le permitiría enviar peticiones POST a esa URL.
- Para poder evitar el CSRF, siempre hay que:
 - a. Cumplir la especificación HTTP; es decir, GET nunca debe cambiar el estado de la aplicación.
 - b. Tener activada la protección CSRF de Yii.

10. Pruebas

10.1. Introducción

- Las pruebas son una parte importante del desarrollo de software.
- Continuamente estamos haciendo pruebas, aunque no seamos conscientes de ello.
- Por ejemplo:
 - Cuando escribimos una clase de PHP, la vamos depurando paso a paso o simplemente usamos sentencias `echo` o `die()` para verificar que la implementación funciona de acuerdo a nuestro plan inicial.
 - En una aplicación web, introducimos algunos datos de prueba en los formularios para comprobar que la página interactúa con el usuario como era de esperar.
- El proceso de *testing* debe automatizarse para que, cada vez que tengamos que verificar algo, simplemente tengamos que ejecutar el código que lo haga por nosotros.
- El código que verifica que el resultado se ajusta a nuestro plan, se denomina **prueba** o **test**.
- El proceso de crear y ejecutar pruebas se denomina **testing automático**.

10.2. Desarrollo mediante pruebas

- El *Test-Driven Development (TDD)* y el *Behavior-Driven Development (BDD)* son aproximaciones al desarrollo de software que se basan en describir el comportamiento de un trozo de código o de una funcionalidad completa como un conjunto de escenarios o tests antes de escribir el código real, y sólo después crear la implementación que permite pasar los tests que verifican el comportamiento deseado.
- El desarrollo de una funcionalidad se lleva a cabo así:
 - a. Crear un nuevo test que describe la funcionalidad a implementar.
 - b. Ejecutar el nuevo test y asegurarse que falla. Es lo esperado, ya que todavía no hay ninguna implementación.
 - c. Escribir código sencillo que supere el nuevo test.
 - d. Ejecutar todos los tests y comprobar que pasan todos.
 - e. Mejorar el código y comprobar que los tests aún siguen pasando.
- Una vez terminado el proceso, se vuelve a repetir con otra funcionalidad.
- Si hay que cambiar una funcionalidad, se deben cambiar también los tests correspondientes.
- El motivo por el que hay que crear los tests antes que la implementación es que así uno se centra en lo que quiere conseguir y deja para más tarde el “cómo hacerlo”.
- Normalmente, eso lleva a obtener mejores abstracciones y a facilitar el mantenimiento de los tests.
- Resumiendo: las ventajas de esta aproximación son:

- a. Te mantiene enfocado en una cosa cada vez, lo que mejora la planificación y la implementación.
 - b. Conlleva más características cubiertas por pruebas, y con mayor detalle. Por tanto, si los tests pasan, casi seguro que no hay nada roto.
- A largo plazo, es un gran ahorro de tiempo.

10.3. Cuándo y cómo hacer las pruebas

- Aunque la aproximación de “primero las pruebas” descrita anteriormente puede tener sentido para proyectos relativamente grandes y a largo plazo, puede resultar demasiado para otros más simples.
- Hay otros indicadores de cuándo puede resultar apropiado:
 - El proyecto ya es grande y complejo.
 - Los requisitos empiezan a complicarse. El proyecto crece continuamente.
 - El proyecto está destinado a existir durante mucho tiempo.
 - El coste de un fallo es demasiado alto.
- No hay nada malo en crear pruebas que comprueben el comportamiento de una implementación existente.
 - Es un proyecto antiguo que se va renovando gradualmente.
 - Te ponen a trabajar en un proyecto y no hay pruebas.
- En algunos casos, el testing automático puede resultar excesivo:
 - Cuando el proyecto es simple y nunca se va a volver complejo.
 - Cuando es un proyecto puntual en el que no se va a seguir trabajando.
- Aún así, si se tiene tiempo, también es bueno tener pruebas automáticas en estos casos.

10.4. Codeception

- Configurar bien `config/test_db.php`, o las pruebas darán error.
- Echar un vistazo también a `config/test.php` y ver que todo está como corresponde.
- Si no funciona “`composer exec codecept ...`” probar mejor con “`vendor/bin/codecept ...`”.

10.4.1. Pruebas de aceptación

- Consultar <http://phptest.club/t/how-to-run-headless-chrome-in-codeception/1544> y <http://codeception.com/docs/modules/WebDriver>
- La URL puede ser <http://localhost:8080/> (para lo cual se necesita arrancar el servidor web integrado) o también se puede usar la URL habitual <http://proyecto.local/> si se va a usar el Apache (en tal caso no hace falta arrancar el servidor web integrado). La primera opción probablemente es más recomendable en entornos de testing como Travis CI.

10.4.1.1. En modo headless

10.4.1.1.1. Sin Selenium

- Cambiar “codeception/base” por “codeception/codeception” en `composer.json` (ésto sólo hace falta para las pruebas de aceptación, porque así se instala la librería `facebook/webdriver`) y hacer `composer update`.
- Preparar la suite de aceptación y usar Chrome en lugar de Firefox:
 - `cp tests/acceptance.suite.yml.example tests/acceptance.suite.yml`
 - Crear en el nuevo archivo el siguiente contenido:

```
modules:  
  enabled:  
    - WebDriver:  
      url: http://localhost:8080/  
      browser: chrome  
      port: 9515  
      window_size: false  
      capabilities:  
        chromeOptions:  
          args: ["--headless", "--disable-gpu", "--disable-extensions"]  
          binary: "/usr/bin/google-chrome"  
    - Yii2:  
      part: orm  
      entryScript: index-test.php  
      cleanup: false
```
- Descargar y ejecutar [ChromeDriver](#):
 - Descargar de <http://chromedriver.storage.googleapis.com/index.html>
 - Descomprimir y copiar a `/usr/local/bin`
 - `chromedriver --url-base=/wd/hub`
- Arrancar el servidor web integrado:
 - `./yii serve`
- Ejecutar todas las pruebas:
 - `composer exec codecept run`
- O bien, ejecutar sólo las pruebas de aceptación:
 - `composer exec codecept run acceptance`
- O bien, sólo una prueba de aceptación en concreto:
 - `composer exec codecept run acceptance HomeCest`

10.4.1.1.2. Con Selenium

- Cambiar “codeception/base” por “codeception/codeception” en `composer.json` (ésto sólo hace falta para las pruebas de aceptación, porque así se instala la librería `facebook/webdriver`) y hacer `composer update`.
- Preparar la suite de aceptación y usar Chrome en lugar de Firefox:
 - `cp tests/acceptance.suite.yml.example tests/acceptance.suite.yml`
 - Crear en el nuevo archivo el siguiente contenido:

```
modules:  
  enabled:
```

```

- WebDriver:
  url: http://localhost:8080/
  browser: chrome
  window_size: false
  capabilities:
    chromeOptions:
      args: ["--headless", "--disable-gpu", "--disable-extensions"]
      binary: "/usr/bin/google-chrome"
- Yii2:
  part: orm
  entryScript: index-test.php
  cleanup: false

```

- Descargar [ChromeDriver](#):
 - Descargar de <http://chromedriver.storage.googleapis.com/index.html>
 - Descomprimir y copiar a /usr/local/bin
- Instalar y ejecutar Selenium:
 - composer require --dev se/selenium-server-standalone
 - composer exec selenium-server-standalone
- Arrancar el servidor web integrado:
 - ./yii serve
- Ejecutar todas las pruebas:
 - composer exec codecept run
- O bien, ejecutar sólo las pruebas de aceptación:
 - composer exec codecept run acceptance
- O bien, sólo una prueba de aceptación en concreto:
 - composer exec codecept run acceptance HomeCest

10.4.1.2. En modo normal

- Cambiar “codeception/base” por “codeception/codeception” en composer.json (ésto sólo hace falta para las pruebas de aceptación, porque así se instala la librería facebook/webdriver) y hacer composer update.
- Preparar la suite de aceptación y usar Chrome en lugar de Firefox (éste último a mí no me funciona):
 - cp tests/acceptance.suite.yml.example tests/acceptance.suite.yml
 - Sustituir firefox por chrome en tests/acceptance.suite.yml
- Descargar [ChromeDriver](#):
 - Descargar de <http://chromedriver.storage.googleapis.com/index.html>
 - Descomprimir y copiar a /usr/local/bin
- Instalar y ejecutar Selenium:
 - composer require --dev se/selenium-server-standalone
 - composer exec selenium-server-standalone
- Arrancar el servidor web integrado:
 - ./yii serve
- Ejecutar todas las pruebas:
 - composer exec codecept run
- O bien, ejecutar sólo las pruebas de aceptación:
 - composer exec codecept run acceptance

- O bien, sólo una prueba de aceptación en concreto:
 - `composer exec codecept run acceptance HomeCest`

10.4.2. Posibles problemas

- Puede que algunas pruebas de aceptación no funcionen bien en i3wm debido a que, al abrirse automáticamente el navegador, éste ocupe sólo una parte de la pantalla, y entonces la página web no salga entera y no aparezcan todos los elementos en la pantalla. La solución es asegurarse de que la ventana del navegador ocupa todo el escritorio.
- Puede que algunas pruebas vayan lentísimas y consuman mucha memoria. Para resolverlo, ponemos la siguiente línea en `tests/codeception/_bootstrap.php`:
`\Codeception\Specify\Config::setDeepClone(false);`

Ver <https://github.com/Codeception/Codeception/issues/2516#issuecomment-209917193>

10.5. Fixtures

- <http://www.yiiframework.com/doc-2.0/guide-test-fixtures.html>
- <https://github.com/yiisoft/yii2-faker/blob/master/docs/guide/README.md>
- <https://github.com/fzaninotto/Faker>
- El usuario de la base de datos **debe tener permisos de administrador (superuser)** en el clúster para que no den errores de permisos al activar/desactivar triggers.
- Poner lo siguiente en `config/console.php`:

```
'aliases' => [
    '@bower' => '@vendor/bower-asset',
    '@npm'   => '@vendor/npm-asset',
    '@tests' => '@app/tests',           // añadir ésta línea
],
```
- Meter lo siguiente en `config/console.php` (ya viene comentado; sólo hay que descomentar y añadir `'language' => 'es_ES'`):

```
'controllerMap' => [
    'fixture' => [ // Fixture generation command line.
        'class' => 'yii\faker\FixtureController',
        'language' => 'es_ES',
    ],
],
```
- Crear `tests/unit/fixtures/SociosFixture.php` (crear directorios intermedios si hace falta) con el siguiente contenido:

```
<?php
```

```
namespace tests\unit\fixtures;
```

```
use yii\test\ActiveFixture;
```

```
class SociosFixture extends ActiveFixture
{
    public $modelClass = 'app\models\Socios';
}
```

- Crear tests/unit/templates/fixtures/socios.php (crear directorios intermedios si hace falta; el nombre del archivo tiene que coincidir con el de la **tabla**, no con el del modelo) con el siguiente contenido:
- <?php
/**
 * @var \$faker \Faker\Generator
 * @var \$index integer
 */
return [
 'numero' => \$faker->unique()->randomNumber(6),
 'nombre' => \$faker->name,
 'direccion' => \$faker->address,
 'telefono' => \$faker->randomNumber(9),
];
- ./yii fixture/generate-all --count=50
Comprobar que se crea tests/unit/fixtures/data/socios.php (igualmente, el nombre debe coincidir con el de la **tabla**) con datos ficticios.
- ./yii fixture/load Socios
Comprobar que se han cargado correctamente los datos ficticios en la base de datos de desarrollo.
- tests/bin/yii fixture/load Socios
Hace lo mismo, pero sobre la base de datos de pruebas.

10.6. Travis-CI

- Archivo [.travis.yml](#) con la configuración.
- El PHP de Travis-CI tiene el date.timezone = 'UTC', como Heroku.
- El PostgreSQL de Travis-CI tiene la siguiente configuración, idéntica a la de Heroku:
 - Codificación: UTF-8.
 - Collate: 'en_US.UTF-8'
 - Ctype: 'en_US.UTF-8'
 - show datestyle => 'ISO, MDY'
 - show intervalstyle => 'postgres'
 - show lc_messages => 'en_US.UTF-8'
 - show lc_monetary => 'en_US.UTF-8'
 - show lc_numeric => 'en_US.UTF-8'
 - show lc_time => 'en_US.UTF-8'
 - show default_text_search_config => 'pg_catalog.english'

11. Estilo del código

- Seguir las normas internas de Yii2 para [el código](#) y para [las plantillas de las vistas](#).
- [.php_cs.dist](#) para PHP-CS_Fixer (copiar dentro de cada proyecto)
- [.codeclimate.yml](#)
- [phpmd.xml](#) combinar con [phpmd_ruleset.xml](#)
- <https://github.com/ricpelo/pre>

12. Documentación (obsoleto)

- Mirar <https://github.com/blog/2228-simpler-github-pages-publishing>.
- Formato [GitHub flavored Markdown](#).
- Para los diagramas se puede usar [yEd](#).
- Generar la documentación usando la extensión [yii2-apidoc](#).
- Añadir la siguiente línea en `composer.json` (no vale hacer `composer require`):
`"yiisoft/yii2-apidoc": "~2.1.0"`
- Luego hacer `composer update yiisoft/yii2-apidoc cebe/markdown`.
- `mkdir docs guia`
- Se crean los archivos `.md` de la guía en `guia`.
- Generar la documentación:
 - `vendor/bin/apidoc api commands,controllers,assets,models,views docs/api --pageTitle="Videoyii API" --guide=.. --guidePrefix= --interactive=0`
 - `vendor/bin/apidoc guide guia docs --pageTitle="Guía de Videoyii" --guidePrefix= --apiDocs=./api --interactive=0`
 - `ln -sf README.html docs/index.html`
 - `touch docs/.nojekyll`
- Script que lleva a cabo los pasos anteriores [aquí](#).

13. Despliegue en Heroku

- [Información general sobre PHP en Heroku](#).
- Instalar [Heroku CLI](#).
- Cambios en `composer.json`:
 - Añadir en "require":
`"ext-gd": "*",`
`"ext-intl": "*"`
 - Si se quiere usar la depuración en Heroku, hay que pasar "yiisoft/yii2-debug" de "require-dev" a "require".
 - Recordar hacer `composer update` tras los cambios y meter todos los cambios en un nuevo commit.
- Crear archivo `Procfile` con el siguiente contenido:

web: vendor/bin/heroku-php-apache2 web/

- Cambios en web/index.php:
 - Cambiar las líneas donde se definen YII_ENV y YII_DEBUG por las siguientes:
define('YII_ENV', getenv('YII_ENV') ?: 'dev');
define('YII_DEBUG', getenv('YII_DEBUG') ?: YII_ENV == 'dev');
- Sustituir el contenido de config/db.php por el siguiente:

```
<?php
```

```
if (($url = getenv('DATABASE_URL')) !== false) {  
    // Configuración para Heroku:  
    $config = parse_url($url);  
    $host = $config['host'];  
    $port = $config['port'];  
    $dbname = substr($config['path'], 1);  
    $username = $config['user'];  
    $password = $config['pass'];  
} else {  
    // Configuración para entorno local:  
    $host = 'localhost';  
    $port = '5432';  
    $dbname = 'proyecto';  
    $username = 'proyecto';  
    $password = 'proyecto';  
}  
  
return [  
    'class' => 'yii\db\Connection',  
    'dsn' => "pgsql:host=$host;port=$port;dbname=$dbname",  
    'username' => $username,  
    'password' => $password,  
    'charset' => 'utf8',  
    'on afterOpen' => function ($event) {  
        // $event->sender refers to the DB connection  
        $event->sender->createCommand("SET intervalstyle = 'iso_8601'")->execute();  
    };  
];
```

- Crear la aplicación en el [Dashboard](#) de Heroku.
- Añadir Heroku Postgres como add-on.
- Comprobar que se crea automáticamente la variable de entorno DATABASE_URL.
- Crear la variable de configuración YII_ENV con el valor prod.
- Conectar a la aplicación desde la consola:
heroku login
- Hacer push:
git push heroku master
- Opcionalmente: se puede conectar la cuenta de Heroku con la de GitHub y desplegar directamente la aplicación pulsando un botón, o activar el despliegue automático por cada commit de un rama concreta.
- Crear las tablas de la base de datos:
heroku pg:psql < db/videoyii.sql

- El PHP Heroku de tiene el `date.timezone = 'UTC'`, como Travis-CI.
- El PostgreSQL de Heroku tiene la siguiente configuración, idéntica a la de Travis-CI:
 - Codificación: UTF-8.
 - Collate: 'en_US.UTF-8'
 - Ctype: 'en_US.UTF-8'
 - `show datestyle => 'ISO, MDY'`
 - `show intervalstyle => 'postgres'`
 - `show lc_messages => 'en_US.UTF-8'`
 - `show lc_monetary => 'en_US.UTF-8'`
 - `show lc_numeric => 'en_US.UTF-8'`
 - `show lc_time => 'en_US.UTF-8'`
 - `show default_text_search_config => 'pg_catalog.english'`
- Ante problemas, mirar los logs:
 - Usar `joni-jones/yii2-heroku-logger` para que la aplicación vuelque bien los logs.
 - Desde la consola:
`heroku logs -t`
 - Desde la interfaz web:
<https://dashboard.heroku.com/apps/secure-beyond-41312/logs> (cambiar `secure-beyond-41312` por el nombre de la aplicación que sea)

13.1. Limitaciones de la cuenta gratuita (Free)

- 550h de uso de dynos al mes.
- Los dynos gratuitos se duermen tras 30 minutos de inactividad. Un dyno dormido no consume horas de la cuota mensual.
- No se puede escalar a más de un dyno.
- 10000 filas en tablas de bases de datos.
- 20 conexiones simultáneas a base de datos.
- Para saber cuántas [horas gratis](#) nos quedan este mes:
`heroku ps`
- El sistema de ficheros de los dynos es local y efímero. Los archivos que se suben a un dyno son temporales, se pierden al reiniciar el dyno y no se comparten entre los dynos. Soluciones:
 - Aceptarlo:
 - Con un sólo dyno funcionará mientras no se reinicie (para la defensa del Proyecto Integrado es suficiente).
 - Usar Amazon S3:
 - Hay una versión gratuita pero hay que indicar tarjeta de crédito para validar la cuenta y cobrarte si te pasas de los límites.
 - Usar la base de datos.
 - Usar Dropbox API (<https://github.com/spatie/dropbox-api> y <https://github.com/spatie/flysystem-dropbox>). Según pruebas, funciona muy bien:
`//use League\Flysystem\Filesystem;`

- ```

use Spatie\Dropbox\Client;
//use Spatie\FlysystemDropbox\DropboxAdapter;
$authorizationToken =
'UwQvZxX2mHgAAAAAFAFDaKJ-ASC1mXNRECNvU-Shpd8cmWgUGJznhkh2z580
5PC';
$client = new Client($authorizationToken);
//$adapter = new DropboxAdapter($client);
//$filesystem = new Filesystem($adapter);
$client->upload('logo.png', file_get_contents('logo.png'),
'overwrite');
$res = $client->createSharedLinkWithSettings('/logo.png',
['requested_visibility'=>'public']);

```
- En `$res['url']` está la URL con el enlace compartido.

## 14. Manipulación de imágenes

- Usar la extensión [yii2-image](#) en su versión de desarrollo (la versión actual, 2.1.0, es antigua y no funciona bien con los PNG, además de faltarle el método `resize()`):

```
composer require yiisoft/yii2-image:dev-master
```

- Lo ideal sería actualizar también el paquete `image/image`, pero `yii2-image` no te deja. (*TODO*: hacer un fork de `yii2-image` que dependa de la versión en desarrollo de `image/image`).
- Para cambiar el tamaño de una imagen:

```
yii\image\Image::resize('logo.png', 50, null)->save('miniatura.png');
```