

Yii2

*Para
Principiantes*

PHP desarrollo de aplicaciones web

Una guía paso a paso para Yii 2

De la configuración a los fundamentos de codificación

Traducido por Victor Hugo Garcia

Bill Keck

Yii 2 Para Principiantes

Bill Keck y Victor Hugo Garcia

Este libro está a la venta en <http://leanpub.com/yii2paraprincipiantes>

Esta versión se publicó en 2017-01-29



This is a [Leanpub](#) book. Leanpub empowers authors and publishers with the Lean Publishing process. [Lean Publishing](#) is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

© 2015 - 2017 Bill Keck y Victor Hugo Garcia

Este libro está dedicado a todos los que se han tomado el tiempo de ayudar a otros a aprender algo sobre programación, ya sea a través de un artículo en línea, blog, foro de una comunidad, o libro. Así como me he beneficiado de la ayuda de otros, yo también quisiera contribuir...

También quisiera agradecer al equipo principal de Yii 2, comenzando con el fundador de Yii Qiang Xue, quien, con su creación de Yii 2, ha traído a este mundo algo hermoso y útil. Además, muchas gracias a los incansables esfuerzos de Samdark, Cebe, Orey, y Kartik, que se toman el tiempo de ayudar a gente como yo, y a otros como yo, que sienten pasión por la programación, pero que a veces necesitan un poco de ayuda para completar los detalles.

Y finalmente, gracias a mi esposa e hijos, simplemente porque sin ellos, no me aventuraría a apuntar tan alto, aún si sólo soy capaz de comprender tan poco. Al final, es el sueño de lograr algo lo que me motiva, y es sólo con su amor y apoyo que soy capaz de lograr esto en absoluto.

Índice general

Capítulo Uno: Introducción	1
Introducción	1
Características	1
¿Qué hace tan especial al framework Yii 2?	2
Ventajas	3
Desventajas	3
Por qué elegí Yii 2	3
Otras opciones	4
Yii 2 llega	4
Gii	5
Enfoque base de datos primero	5
MySql	6
Flujo de trabajo mejorado	6
Habilidades mínimas en PHP	7
Herramientas que necesitará	8
Errata	10
Información de Contacto	10
Resumen	11
Capítulo Dos: Instalación de la Plantilla Avanzada	12
Configuración rápida de la de la Plantilla Avanzada de Yii 2	12
Paso 1 - Crear el Directorio	13
Paso 2 - Configuración de Apache	13
Paso 3 - Host Local	14
Paso 4 - Reiniciar Apache	15
Paso 5 - Crear el Proyecto en el IDE	15
Paso 6 - Encontrar la Ruta de la Línea de Comandos	18
Paso 7 - Composer Self-Update	18
Paso 8 - Instalar Yii 2	19
Paso 9 - Revisar su directorio Yii 2	20
Paso 10 - Ejecutar Php Init	20
Paso 11 - Crear la Base de Datos	21
Paso 12 - Establecer la conexión a la BD	21

ÍNDICE GENERAL

Paso 13 - Ejecutar la Migración	22
Paso 14 - Crear el Repositorio en Git	23
Paso 15 - Confirmar que la Aplicación está funcionando	24
Solución de problemas	25
Resumen	25
Capítulo Tres: Bienvenido al MVC	26
Patrón MVC	26
Index.php	28
La Instancia de la Aplicación	29
Ruteo	30
Usando Gii	30
Bootstrap	31
Depurador	32
Resumen	32
Capítulo Cuatro: Modificando el Modelo de Usuario	34
Rol y Estado	35
El modelo User	37
Propiedades del Modelo	45
Constantes	46
Interfaz Identidad (Identity Interface)	47
Comportamientos (Behaviors)	50
Rules	51
Métodos de Identidad (Identity Methods)	52
Métodos Predefinidos	54
Otros modelos que acceden a User	56
Modelo SignupForm	56
Resumen	60
Capítulo Cinco: Creando Nuevos Modelos con Gii	61
Creando Tablas	61
Tabla Rol	62
Tabla Estado	63
Tabla Tipo Usuario	63
Tabla Genero	64
Tabla Perfil	64
Sincronice	66
Configurando Gii	67
Construyendo Modelos con Gii	69
Crear el Modelo Rol	70
Agregar Registros a la Tabla Rol	72
Agregar la Relación a Rol	73

ÍNDICE GENERAL

Actualizar el Modelo User con Rol	74
Crear el Modelo Estado	77
Actualizar el Modelo User con getEstado	79
Aregar registros a la Tabla Estado	80
Crear Modelo Tipo Usuario	81
Actualizar el Modelo User con TipoUsuario	83
Aregar Registros a la Tabla tipo_usuario	84
Crear el Modelo Genero	85
Aregar Registros a la Tabla genero	87
Crear Modelo Perfil	87
El Modelo Perfil Completo	96
Actualizar el Modelo User con Perfil	102
Finalizar el Modelo User	103
El Modelo User Completo	105
Resumen	117
Capítulo Seis: Helpers (Auxiliares)	119
Helpers de Valor	119
Permisos Helpers	128
Registros Helpers	132
Resumen	134
Capítulo Siete: Controlador del Sitio	136
Behaviors	137
Acciones	138
Acción Index	139
Acción Login	140
Modelo Login Form	140
Acción Logout	144
Acción Contacto	144
Modelo Contact Form	145
Captcha	146
La Vista del Formulario de Contacto	150
Acción About	154
Acción Signup	154
Modelo Signup Form	155
ResetPasswordForm Model	161
Controlador de Sitio del Backend	164
Comenzando el Control de Acceso	166
Método loginAdmin	167
Resumen	169
Capítulo Ocho: Crud de Perfil	171

ÍNDICE GENERAL

CRUD	171
Controlador de Perfil	173
Búsqueda de Perfil	173
_search	173
_form	173
Index	173
View	174
Create	174
Update	175
Modificando el Controlador y las Vistas de Perfil	176
Modificando el Controlador de Perfil	177
Acción Index	179
Acción View	181
Acción Create	183
Acción Update	185
Acción Delete	187
Acción FindModel	188
Modificando la Vistas de Perfil	189
View.php	189
Genero	195
Parcial Formulario	196
Create	199
Update	199
Layout del Sitio	200
Link a Perfil	204
DatePicker	205
Resumen	208
Capítulo Nueve: Upgrade y Control de Acceso	209
Controlador Upgrade	210
Vista Index de Upgrade	211
Requerir Upgrade A	211
Control de Acceso	212
Pasando una Variable desde el Controlador	218
Resumen	220
Capítulo Diez: Widgets Sociales de la Homepage	221
Implementando Widgets Sociales en la Homepage	221
Index	221
Widget de Facebook	222
Configuración de la App de Facebook	223
Configuración de Facebook	229
Extensiones	231

ÍNDICE GENERAL

Helper HTML	232
Widget Collapse	238
Widget Modal	240
Widget Alert	240
Font-Awesome	241
Asset Bundle	242
Agregar Font-Awesome al Layout	244
Resumen	252
Capítulo Once: Creación del Backend	253
Main.php	257
Modificando las Vistas del Backend	262
backend/views/perfil/_form.php	262
backend/views/perfil/view.php	263
backend/views/user/view.php	268
backend/views/user/_form	270
Cambios más profundos al Backend	272
backend/views/user/index.php	272
backend/views/perfil/index.php	274
backend/views/perfil/_search.php	277
backend/views/user/_search.php	278
User Search	280
IU de Admin	300
Behaviors del Controlador	307
Match Callback	308
Resumen	311
Sobre el Autor	311
Capítulo Doce: Material Extra	312
AutoResponder	312
Navegación mediante Dropdown	330
FAQ	337
Test Controller	377
Componentes	379
Creando un Widget personalizado	384
CDN	404
Resumen	407
Capítulo Trece: Material ExtraUrls Amigables y Slugs	409
URLs Amigables	409
Vhost de Apache	410
Reinic peace Apache	411
.htaccess	411

ÍNDICE GENERAL

Slugs	413
Sluggable Behavior	413
Columna Slug	414
Elimine las Faqs viejas y cree nuevas	415
Añadir Reglas al Url Manager	416
Modificar la Acción View en FaqController	416
Modificar las acciones Create y Update en el controlador del backend	418
Cambiar la URL de la columna acción de la Gridview	419
Resumen	426
Capítulo 14: Material Extra Login y Registro con Redes Sociales	428
Yii2 - AuthClient	428
Instalar yii2authclient via Composer	429
Configuración	429
Problema con Twitter	430
Aplicaciones de Proveedores	431
App de Google	434
App de LinkedIn	440
Faq Controller	610
Frontend Faq view.php	611
_rating-form.php	616
FAQRatings Controller	618
Faq Model	626
Faq Index View	626
Faq View	627
Signup Form Model	628
signup.php	631
terms.php	633
termsoverflow.css	635
Frontend AppAsset.php	636
Improving The Carousel	637
Modify marketing_imageTable	638
Modify MarketingImage Model	638
MarketingImage Views	640
Marketing Image view.php	640
Update View	642
MarketingImage Controller	644
Create Action	644
Update Action	646
Delete Action	648
Entire File	649
CarouselSettings Model Rules	649
CarouselWidget	650

ÍNDICE GENERAL

validateSize Method	651
Entire CarouselWidget File	651
carousel.php	651
Summary	659
Chapter 17: Bonus Material Returning Calculated Values in Gridview	661
Sorting A Calculated Value In Gridview	661
Donate To Kartik	663
Average Rating For Gridview	664
Times Rated	667
Summary	669

Capítulo Uno: Introducción

Introducción

Bienvenido a Yii 2 Para Principiantes. Este libro lo llevará paso a paso a través de la configuración e instalación y luego a la codificación en el framework PHP más excitante disponible a la fecha, Yii 2.

Yii 2 viene en dos variedades, básico y avanzado, y puede parecer contraintuitivo usar la plantilla avanzada en un libro para principiantes, pero irónicamente, ésta es más fácil de usar si su aplicación requiere un modelo de usuario funcional que almacene usuarios en una base de datos. La mayoría de las aplicaciones web modernas necesitarán esta funcionalidad y la plantilla avanzada tiene una solución disponible para ello.

El otro gran beneficio de la plantilla avanzada es que divide a la aplicación en frontend y backend, lo que responde la inevitable pregunta de “¿dónde pongo mi área de administración?”

No sólo quiero introducirlo a este maravilloso framework, sino que también quiero que tenga un punto de partida para sus proyectos que incluya todo lo que necesita para construir una aplicación web orientada a datos robusta. Aunque la plantilla avanzada lista para usarse es extremadamente útil, le faltan algunas piezas clave, que completaremos con este libro.

La meta es proveerlo con una plantilla de base que pueda usar para todos sus futuros proyectos.

Características

Algunas de las características que obtendrá con la instalación de la plantilla avanzada incluyen:

- un esquema predefinido para la tabla de usuario
- inicio de sesión de usuario y formularios de registro
- funcionalidad de olvido de contraseña
- dominios separados para frontend y backend
- generación automática de código para modelos, controladores, y vistas
- integración incorporada con Twitter Bootstrap y diseño para móviles
- widgets robustos y helpers para presentación de datos

Si no entiende algo en esa lista, no se preocupe, la cubriremos en detalle. Sólo considere que es realmente asombroso lo que Yii 2 hace por usted. Pero sin importar cuán grandioso sea un framework, aún necesitará hacer más para lograr que soporte una aplicación real.

Así que a las características listas para usar, añadiremos:

- agradables refinamientos de interfaz de usuario para el frontend tales como el selector de fechas (DatePicker) de jquery
- métodos de relación de modelos que facilitan la visualización de datos
- métodos de controlador para restringir quién ve cada cosa
- estructura de datos extendida que será común a todos sus sitios futuros
- control de acceso basado en roles (RBAC)
- restricción del contenido para el usuario basado en su tipo, tal como gratuito o pago
- módulos sociales que permiten compartir
- registro e inicio de sesión con un único clic a través de Facebook

Estas son las cosas que probablemente requiera su aplicación, sin importar de qué tipo de sitio se trate. Así que, a medida que aprenda Yii 2 con este libro, estará construyendo una plantilla que podrá expandir para todas sus futuras aplicaciones.

Este libro es perfecto para programadores PHP principiantes que están listos para iniciarse en el desarrollo usando frameworks. El framework de PHP Yii 2 es altamente escalable y extensible, y cargado con características. Lo introducimos a este maravilloso framework y explicamos en detalle todo lo que necesita saber para ponerse en marcha. ¡Le va a encantar Yii 2!

Los artesanos Php avanzados serán capaces de avanzar velozmente a través de este libro y ponerse en marcha rápidamente con Yii 2, un framework php fenomenal. Esto no sólo les ahorrará tiempo en los proyectos, sino que también aprovechará plenamente los beneficios de un framework de código abierto que tiene una comunidad entera tras él.

El estilo principal de este libro, sin embargo, es para principiantes. Hay mucho de detalle fino a fin de ayudar a la gente que tiene algo de experiencia con PHP pero que aún no ha realizado el salto hacia la programación orientada a objetos.

Tratamos de asegurarnos de que entendemos completamente cómo funciona el framework, cómo usa POO para crear una capa de desarrollo intuitiva que permite a programadores de muchos niveles diferentes alcanzar los resultados que desean.

En todo caso, aprender Yii 2 le otorga experiencia concreta con la programación orientada a objetos con resultados prácticos. Terminará con un sitio web funcional.

¿Qué hace tan especial al framework Yii 2?

Los programadores tienen que tomar decisiones, es un hecho que se encuentra en el corazón de lo que es la programación. Así que una de las mayores decisiones que deberá tomar como programador, y más probablemente, una decisión que tomará como parte de un equipo de programadores, es si usar o no un framework y en caso afirmativo, cuál.

Con respecto a la pregunta de si usar un framework PHP, hay tantos beneficios en hacerlo, que se vuelve algo obvio.

Ventajas

Aquí están algunos de los beneficios obvios:

- Usa formas estandarizadas de realizar las cosas, reduciendo o eliminando el código spaghetti.
- Reduce el tiempo empleado en tareas de plomería tales como la validación de formularios y seguridad.
- Facilita el trabajo en equipo al forzar estándares.
- Facilita el mantenimiento del código al utilizar una arquitectura y métodos comunes.
- Usted obtiene el beneficio de una comunidad activa de desarrolladores que mantienen el framework y apoyan tareas comunes y nuevas características.

Desventajas

Existen un par de desventajas al usar un framework que deben señalarse. Primero, todo el código que compone el framework crea sobrecarga en el servidor y esto puede ser un problema real. Afortunadamente hay disponibles opciones de cacheo que reducirán los efectos de esto, y para aplicaciones empresariales, puede usar sql puro para minimizar el tiempo de consulta. Así que no permita que la sobrecarga del servidor le impida usar un framework.

La otra cosa es que obviamente cuando está trabajando dentro del framework, está trabajando con una gran cantidad de código que no escribió y toma tiempo descubrir cómo funciona. Parte del código del framework puede ser bastante crítico dependiendo de su nivel de habilidad y experiencia, así que no espere comprender todo instantáneamente. No sucederá.

Por supuesto usted ya sabía que existía una curva de aprendizaje, lo que motiva que esté leyendo este libro. Y aunque toma tiempo aprender el código de alguien más, lo que puede resultar una molestia, es mucho más tedioso tener que escribir un framework personalizado de cero. Considerando todo, usar un framework para el desarrollo empresarial es una sabia decisión.

Bien, así que la parte fácil es descubrir que utilizar un framework lo ayudará a desarrollar un proyecto más organizado y robusto, pero ahora viene la parte difícil. Tiene que decidir qué framework utilizar.

Por qué elegí Yii 2

No puedo decirle cuál es el mejor framework para usted, eso es algo que usted y nadie más debe decidir, pero puedo compartir un poco del trayecto que me llevó a Yii 2. Esta decisión no fue motivada por la necesidad de encontrar la forma más fácil de aprender PHP, eso es seguro.

En mi compañía allá por 2012, formaba parte de un equipo de desarrolladores que examinaba varios frameworks y tenía que decidir cuál usar. Nunca soñé en aquel momento que terminaría escribiendo un libro sobre uno de ellos.

De cualquier modo, colectivamente investigamos todo lo que pudimos encontrar sobre los principales frameworks PHP. Personalmente leí toda la documentación y mantuvimos largas discusiones de ingeniería sobre lo que pensábamos que funcionaría. No puede imaginarse mi frustración con el hecho de que leí toda esa documentación y me aparté de ella sintiéndome con menos conocimiento que antes de comenzar a leerla.

Nuestro equipo de programadores sin embargo, sí tenía su preferido. Sintieron que Yii 1.1.14 era su mejor opción. Esta era la versión de Yii que se encontraba disponible en aquel momento cuando estábamos decidiendo esto. Así que el equipo adoptó aquel framework y nunca miró hacia atrás. Lo adoraron.

Yo, por otra parte, permanecía frustrado. Ya que era sólo un programador novato, realmente luchaba por aprenderlo. No me parecía muy intuitivo. Especialmente al compararlo con otros frameworks, dónde se estaban esforzando tanto por hacer que todo se integrara bellamente, la arquitectura de Yii simplemente parecía horrible.

Me frustré tanto en un punto, que comencé a buscar otras opciones.

Otras opciones

Encontraba una documentación bellamente escrita sobre un nuevo framework y se la llevaba al equipo. Siempre obtenía la misma respuesta. El equipo estaba feliz con Yii.

Me decían que tal vez era difícil de aprender, pero que era fácil de usar, una vez que entendías cómo trabajaba. Debido a eso, me comprometí conmigo mismo a aprenderlo. Fue un camino lento y difícil. No lo estaba entendiendo. Me encontraba trabajando en el capítulo 10 de un libro sobre Yii 1.1.14, pensando que nunca sería realmente capaz de construir una aplicación por mi mismo en menos de cien años. Demasiados caminos parecían conducir a ninguna parte.

Luego ocurrió un milagro.

Yii 2 llega

Me enteré de la versión alfa de Yii 2. Sentía curiosidad por ver cuáles eran las diferencias en Yii 2, que llevaba 3 años de desarrollo hasta ese momento. Así que comencé y para mi total y completa sorpresa, instantáneamente conecté con él. Comprendí las estructuras. ¡Podía escribir código que realmente funcionaba! Qué gran sentimiento era.

Personalmente he hallado que Yii 2 es el más intuitivo y elegante de todos los frameworks PHP que he estudiado. Siento tanto entusiasmo por él que quiero compartirlo con cada programador que conozco, y aún con aquellos que no conozco, lo que me ha motivado a escribir este libro.

Con Yii 2, aún como principiante, fui capaz de levantar un sitio web funcional con un modelo de usuario orientado a datos, tanto con un frontend como con un backend. Desde el comienzo, obtuve un modelo de usuario funcional, con funcionalidad de recuperación de contraseña, que también tenía

Bootstrap integrado para un diseño adaptativo móvil, sin tener que realizar nada de programación. ¿Qué tan genial es eso?

Aunque era un programador principiante cuando estaba estudiando los frameworks PHP, sí tenía experiencia trabajando con bases de datos y en mi opinión esta es un área dónde Yii 2 realmente sobresale.

Gii

Yii 2 tiene una herramienta de generación de código llamada Gii. La pronuncio con una “g” suave, pero no tengo idea si esa es la manera correcta de hacerlo o no.

De cualquier modo, Gii analiza las tablas de su base de datos y automáticamente construye modelos PHP a partir de ellas. No sólo eso, sino que también analiza las relaciones entre tablas y automáticamente genera el código relacional en los modelos. Por ejemplo, si usted tuviera una estructura de datos con 30 tablas, y la mitad de ellas tuvieran una columna usuario_id que fuera una referencia a la id de la tabla usuario, Gii construiría las relaciones apropiadas por usted, cada vez que construyera un modelo. No sólo esto ahorra tiempo, sino que también le da un código muy consistente porque es hecho siempre de la misma manera y le ayuda a adoptar esta disciplina.

Vale la pena mencionar que otros frameworks trabajan exactamente de manera opuesta. Con ellos, construye primero el modelo, luego realiza una migración a la base de datos para crear la tabla y columnas correspondientes. Así que la gran diferencia es que está construyendo su estructura de datos por partes a medida que avanza, mientras que en Yii 2 tiene la opción de tener una estructura de datos más completa para comenzar.

Ambos enfoques funcionan, sin embargo representan flujos de trabajo drásticamente diferentes. En mi opinión, el enfoque migración/paso a paso para la estructura de datos sólo funciona realmente para un único desarrollador o para un equipo muy pequeño trabajando en un proyecto pequeño. La razón por la que digo esto es porque aunque la democracia probablemente sea el mejor sistema político, imagine un mundo dónde cada desarrollador desarrolla su propia estructura de datos y la implementa. ¿Qué tan consistente sería? ¿Qué sucedería si la mano derecha no supiera lo que la izquierda está haciendo? En equipos mayores, esta es una receta para el caos. Es por ello que los equipos de desarrollo empresarial usualmente tienen un administrador de base de datos, también conocido como DB, y sólo él puede crear o eliminar estructuras de datos.

Enfoque base de datos primero

Ya que Yii 2 le permite esencialmente importar modelos desde la estructura de datos, puede comenzar su proyecto pensando realmente en su estructura de datos. En general prefiero evitar hablar demasiado de teoría porque el tiempo se emplea mejor trabajando en ejemplos concretos, pero creo que vale la pena tomarse un momento para pensar sobre lo que significa realmente una estructura de datos bien pensada.

Ya sea usted un único desarrollador o parte de un equipo de nivel empresarial, se le otorga esencialmente la misma tarea, la misma misión global. Tiene que servir datos desde una base de datos en un formato amigable con el navegador, típicamente usando PHP, HTML, y Javascript. Usamos un framework PHP para facilitar esta tarea, y al decir eso, estamos admitiendo desde un comienzo que la tarea no es fácil. ¿Por qué es esto?

La base de datos es una pieza de software muy confiable y consistente, que nos permite crear una estructura de datos relacional.

MySql

A través de este libro usaremos Mysql como base de datos, la cuál, además de ser libre, es capaz de alimentar datos empresariales para aplicaciones web.

Debido a la estructura de la base de datos, con sus índices y claves primarias, una base de datos puede servir datos muy eficientemente. En los términos más simples, esto significa que es muy rápida. También es muy basta. Puede contener millones de registros, que pueden ser recuperados, si están estructurados de manera adecuada, en milisegundos.

Otro aspecto clave de la base de datos es que nos permite estructurar los datos de manera tal de conectar cosas tales como la dirección del usuario y su nombre de usuario como si se tratasesen de un único registro, pero mantenerlas en tablas separadas como registros diferentes. Mientras más pueda separar la estructura de datos en componentes discretos como ese, más poderosa será. Esto se llama normalización de datos.

El problema es que mientras más refinada es la base de datos, mientras más normalizada se encuentra, es más compleja de manipular con PHP. Termina teniendo que conectar muchos modelos PHP para representar correctamente los datos.

Ahora esto puede estar poniéndose demasiado pesado con la teoría para un libro para principiantes, pero el punto es comprender la naturaleza del problema que el framework resuelve. Mientras más fácil sea para usted conectar los modelos a través del framework, mayor es el poder que obtendrá de su base de datos.

Flujo de trabajo mejorado

En mi opinión, Yii 2 sobresale en la manera en que lo ayuda a conectar los modelos a la base de datos, llevando a un flujo de trabajo mejorado, eficiencia, y capacidades de diseño generales. Lo libera para que pueda construir una estructura de datos rica lo que redundará en última instancia en un usuario final más involucrado. Creo que Yii 2 hace esto más eficiente y profundamente que cualquiera de los otros frameworks PHP, es por ello que estoy tan comprometido en compartirlo.

Habilidades mínimas en PHP

Aprender un framework PHP es simple siempre y cuando sea un programador PHP talentoso. De forma bastante habitual PHP es un segundo o tercer lenguaje para un programador que ya es competente en un lenguaje orientado a objetos tal como C o Java, así que aprender PHP es simplemente cuestión de ajustar la sintaxis y lo aprenden rápidamente. Eso es fantástico para los ninjas, ¿pero qué ocurre si está aprendiendo su primer lenguaje de programación?

Como alguien que aprendió PHP como su primer lenguaje de programación, puedo decirle por experiencia directa que es difícil moverse de principiante a avanzado porque no hay demasiado apoyo en el terreno intermedio. Ese hecho es una de mis motivaciones para escribir este libro.

De cualquier modo, si busca en línea, encontrará ya sea ejemplos complejos que involucran múltiples interfaces con objetos anidados o ejemplos que son tan rudimentarios, que aunque son fáciles de comprender, no hacen nada por mejorar sus habilidades.

Para trabajar a lo largo de este libro, necesitará una comprensión decente de PHP orientado a objetos. Puede obtenerlo de una variedad de recursos en línea. Tuve mi primer contacto con PHP en thenewboston.org, que tiene 200 videos sobre PHP. Fantástico para una introducción, pero no mucho más. Continué con una rápida lectura del libro de Richard Reese's sobre Java, lo que me ayudó a comprender mejor la programación orientada a objetos, ya que todo en Java involucra una clase. Además, cuando volví a mirar a PHP, pareció más simple. También fui a través de las bases en:

[W3 Schools](#)

W3schools.com es un gran recurso de aprendizaje en línea. Puede jugar con el código en línea en ese sitio.

Y luego por supuesto está [PHP.net](#) que es donde encontramos toda la documentación para el lenguaje y a veces ejemplos muy complejos. Aprendí mucho allí y también me he perdido mucho, así es como sucede. Intentelo, y verá a qué me refiero.

En cualquier instancia, para ser capaz de trabajar con Yii 2, debería comprender las bases sobre objetos, arreglos, y estructuras de control tales como bucles foreach. Debería conocer los componentes de una clase, propiedades y métodos, etc. De un vistazo a:

[OOP for Beginners](#)

Debería ser capaz de realizar ese tutorial muy fácilmente. Si no es así, regrese y estudie antes de intentar Yii 2. Además, Yii 2 usa PHP 5.4 o superior, que soporta una nueva sintaxis de arreglos y espacios de nombres, ambos de los cuales serán utilizados extensamente.

Si carece de experiencia en la programación, pero está lleno de entusiasmo, le irá bien, siempre y cuando esté dispuesto a trabajar y ser paciente. En cualquier momento, sino comprende algo, puede detenerse y tomarse el tiempo para investigarlo en [Google](#) o [StackOverflow](#) o [PHP.net](#). PHP es un lenguaje bien documentado y soportado, usado por innumerables programadores que tratarán de ayudarlo.

También, he puesto mucho cuidado en etiquetar las subsecciones de este libro, para que pueda encontrar fácilmente lo que busca, si necesita revisarlo con posterioridad. Muchas veces querrá regresar a una sección para referirse a algo y he puesto mi mayor esfuerzo en hacerlo tan intuitivo como sea posible.

Herramientas que necesitará

Hay un número de herramientas que le recomiendo usar para desarrollar en Yii 2, todas las cuales, al igual que el framework mismo, son libres. Estas son simplemente recomendaciones, y no es necesario seguirlos exactamente, pero mis instrucciones asumirán que las está usando. Así que si usted es lo suficientemente avanzado, puede usar las que desee, no es demasiado importante. Mientras tenga un entorno de desarrollo funcional, estará bien. Si no, trate de usar exactamente estas herramientas, le será más fácil a largo plazo.

A veces la parte más difícil de un proyecto es configurar el entorno de desarrollo. Debido a que lo uso personalmente, estaré usando xampp en una máquina windows. Xampp incluye PHP, Mysql, Apache, y PhpMyadmin, así que es perfecto para crear un entorno de desarrollo. Además es gratuito.

[Descargar Xampp](#)

[Video de Instalación para Xampp y PhpMyadmin](#)

Cualquier alternativa que le permita ejecutar estos programas está bien, no necesita Xampp para seguir este libro. Por otro lado, es muy fácil ponerse en marcha con Xampp, lo que es una razón para usarlo. La parte complicada es la configuración de las variables de entorno en una máquina Windows, pero eso está bien documentado y he provisto de enlaces de descarga y configuración para su conveniencia, así que puede revisarlos si lo necesita.

Aún cuando todo lo relacionado a Mysql puede realizarse con PhpMyadmin, también recomiendo configurar Mysql workbench. Los diagramas de ERM (Entidad Relación Mejorado) de Workbench lo ayudan a ver las relaciones y a que la creación de tablas y claves foráneas sea sencillísima. Usaremos fotos de Mysql Workbench para mostrarle la estructura de las tablas más adelante en este libro.

[Descargar Mysql Workbench](#)

Debería familiarizarse con cómo crear una base de datos, como sincronizar un modelo con la base de datos, y obviamente con cómo crear una tabla y columnas. Para construir una aplicación orientada a datos, necesita una comprensión básica de sql, nada demasiado profundo, pero debería conocer cómo trabajan las sentencias básicas y el concepto de la unión de tablas. Y ya que usamos MySql, necesita estar familiarizado con él. Si algo de esto es nuevo para usted, la buena noticia es que puede buscar en google algunos tutoriales y encontrar todo lo que necesita de forma gratuita. [W3 Recursos tutorial MySql](#) son una gran referencia.

En cuanto a mi IDE, uso PhpED. IDE significa Entorno de Desarrollo Integrado (por sus siglas en inglés), y lo ayuda a organizar proyectos y código. La mayoría de los desarrolladores usan alguna forma de IDE en lugar de un simple editor de texto. Recomiendo Eclipse o Netbeans para este

proyecto, sin embargo, debido a que ambos son gratuitos mientras que PhpEd es un IDE pago. Para instalar Eclipse, deberá instalar el sdk de Java en primer lugar.

[Descargar Eclipse](#)

[Descargar Netbeans](#)

También necesitará instalar Composer, lo que debería hacer luego de instalar xampp, lo que significa luego de que PHP esté instalado. Para ejecutar Composer, primero debe habilitar curl en su build de PHP. También necesitará configurar una variable de entorno si está usando windows.

[Descargar composer](#)

[Habilitar Curl](#)

También recomiendo usar git, que provee de control de versión. El control de versión es una manera práctica de guardar su trabajo de forma en que pueda retroceder si lo necesitara. Cuando está tratando con un gran número de archivos que se actualizan permanentemente, esto es una gran ayuda. Git también protege su trabajo de sobreescritura por parte de alguien más en un entorno de equipo simplemente porque puede regresar a una versión anterior.

[Descargar Git](#)

Finalmente, recomiendo console2 para usuarios de Windows, que es una herramienta de línea de comando un poco más agradable que el prompt habitual de windows. Esto lo vuelve más agradable a los ojos y sólo un poco más fácil de trabajar.

[Descargar Console 2](#)

Para lograr que su entorno de trabajo funcione con Yii 2, necesitará agregar tanto una entrada a vhost en Apache como una entrada en el host local dentro de su archivo de hosts. Iremos a través de cada paso en detalle.

Como dije antes, si prefiere usar herramientas diferentes o, por ejemplo, una máquina linux para el desarrollo, esa es su elección.

He provisto de vínculos y páginas de referencia para la instalación, pero para los principiantes, esto puede resultar difícil. Puede usar la instalación del entorno de desarrollo como una de las pruebas para ver si está listo para abordar Yii 2. Simplemente no se rinda fácilmente. Si no resulta, siempre puede obtener ayuda de un programador más experimentado.



Consejo

También, y este es un consejo para los principiantes, casi todo por lo que atravesará como programador es algo por lo que han pasado todos los programadores antes y esto es especialmente cierto para los errores de configuración. No tenga miedo de usar [Google](#) para la solución de problemas con la configuración. Terminará usandolo más a menudo que no haciéndolo.

Una vez que todo está en marcha, pase un poco de tiempo aprendiendo cómo moverse por las herramientas. Hará que sus esfuerzos en desarrollar con Yii 2 sean mucho más sencillos.

Errata

Aunque he revisado cada línea de código de este libro al menos cien veces y he construido los ejemplos desde cero dos veces sólo para asegurarme de que podía seguir las instrucciones, los errores sucederán, tal es la naturaleza de la escritura técnica. Estoy actualizando los errores activamente sobre la marcha, así que espero ser capaz de corregir rápidamente cualquier error sobre el que se me llame la atención. Puede ayudarme enviandome un correo si encontró algo, todos estarán agradecidos.



Aviso sobre el formato

En ciertos casos, tuve que formatear mi código usando dos líneas cuando debería haber una, para evitar los saltos de línea del ajuste de línea en PDF y otros formatos. El ajuste de línea en PDF provoca que aparezcan caracteres especiales, que rompen el código, así que tuve que evitarlo lo más que pude. Como resultado, no le recomiendo seguir los ejemplos de código como una guía de estilo. Sí le recomiendo seguir la Guía PSR-2, disponible aquí: [PSR-2 Guía de Estilo de Codificación](#)

Puede formatear su código con un formateador en [Php Formatter](#), si quiere volverlo más legible. Obviamente sea cuidadoso de no romper el código. También estaré proveyendo de Gists para cada bloque de código que escribamos en el libro, en dónde el bloque de código excede de las 3 líneas. Si no sabe lo que es un Gist, no se preocupe, lo cubriremos en detalle más adelante.

Información de Contacto

Por favor notifiquenos de cualquier error en nuestra traducción contactando a Victor Garcia en vihugarcia@gmail.com. Si el problema que encuentra se debe a un defecto o error de tipeo en el código puede contactar a Bill Keck en ikon321@yahoo.com, pero por favor tenga en cuenta que él no habla español, sólo habla inglés. Si necesita contactarlo en forma directa, le recomendamos usar el traductor de Google para darle formato a su mensaje si no habla inglés. De forma alternativa, puede contactar al traductor, Víctor Garcia, y el contestará su pregunta directamente, o se la transmitirá a Bill Keck en inglés.

También puede dejar un comentario para Bill Keck en su blog sobre Yii 2:

[El Blog de Yii 2 de Bill Keck](#)

Su blog es también una buena fuente de las últimas noticias sobre Yii 2, tendencias en PHP, tutoriales, y unos pocos pensamientos al azar sobre los que decide bloguear en ocasiones. Por favor siéntase en libertad de dejar un comentario.

Por favor tenga en cuenta que el precio de compra de este libro no incluye soporte técnico.

La forma más rápida de solucionar errores es buscándolos en google, muy probablemente alguien se ha enfrentado al mismo problema. También, por favor tenga en mente que Yii 2 es desarrollado

continuamente y nuevas versiones pueden no soportar el código ofrecido en este libro. Esto no es inusual en libros de programación.

Voy a hacer mi mayor esfuerzo para estar al tanto de ello, pero pueden existir ocasiones en que Yii 2 haya realizado un cambio de versión que todavía no he tenido en cuenta. Una vez enterado del asunto, típicamente puedo solucionarlo con rapidez, así que por favor haga su parte y notifíqueme si se percata de algún problema de versión.

Mencionaré numerosas veces y en numerosos lugares que las actualizaciones a este libro están disponibles gratuitamente para usted durante la vida del libro. Planeo añadir material extra en forma regular, así que por favor aprovechelo. Simplemente inicie sesión en su cuenta de leanpub.com para obtener la última versión.

Los avisos de actualizaciones mayores se envían por email. Las actualizaciones menores simplemente se publican, y éstas típicamente simplemente cubrirán un error de tipeo.

Para ver si tiene la última versión, puede ir a la página principal de leanpub.com para el libro y observar la última fecha de actualización, que le mostrará cuando fue mi última modificación.

También, los principiantes se enfrentaran con un gran volumen de mensajes de error debido a errores de tipeo y código faltante. Es perfectamente natural y parte del proceso de aprendizaje. Aprenderá más solucionando errores que simplemente copiando y pegando código.

En la mayoría de los casos, encontrará las respuestas a sus problemas si es paciente. Los foros de Yii 2 son una excelente fuente de apoyo y hay muchos programadores grandiosos que lo ayudarán. Siempre haga su mayor esfuerzo por solucionar el problema primero porque sería tonto limitar el tiempo de un programador con solicitudes de soporte por errores de tipeo. Sin embargo, eso ocurrirá. Simplemente recuerde ser amable y considerado con los otros y le irá bien.

Resumen

Sé que puede ser un poco intimidante al comienzo, especialmente cuando descubre que Yii 2 no es simplemente un conjunto trivial de archivos de librería que puede dominar en unos pocos días, pero continúe y sea paciente. Vamos a abordarlo un paso a la vez.

Así que déjeme concluir la introducción con el siguiente pensamiento. Aprender Yii 2 resultará fácil para algunas personas y ellas son afortunadas. Si usted se encuentra en la otra vereda, la de aquellos que trabajan duro por aprenderlo, puedo decirle que sé exactamente cómo se siente. Fue difícil para mí también. Pero de igual forma puedo decirle que puede ser optimista. Puede lograrlo. Simplemente insista y muévase a su propio ritmo. Y pronto se asombrará de la manera en que utiliza Yii 2 para potenciar sus aplicaciones y se asombrará aún más de lo que puede crear con él.

Capítulo Dos: Instalación de la Plantilla Avanzada

Configuración rápida de la de la Plantilla Avanzada de Yii 2

Bien, ¡comencemos! Vamos a usar yii2build como directorio raíz y nombre del proyecto. Lo desarrollaremos y alojaremos en una máquina Windows con xampp instalado y usaremos PhpED como nuestro IDE. Si desea usar un IDE gratuito, Netbeans es popular así como lo es Eclipse. Busque en Google o vea el capítulo 1 para encontrar links y descargar gratuitamente.

En este punto, asumiremos que tiene su entorno de desarrollo configurado y probado, y que ha pasado algún tiempo familiarizándose con la manera en que trabajan las herramientas. Necesita:

- Eclipse o Netbeans o algún otro IDE
- Composer
- xampp o algún otro entorno apache, php, mysql
- Mysql Workbench
- PhpMyAdmin (incluido con xampp)
- console 2 (opcional)
- GIT o algún otro control de versión

Vea el capítulo 1 para obtener links a descargas gratuitas para las herramientas arriba mencionadas, si no las ha instalado aún.

Si aún no se encuentra en esta etapa, debe regresar a la introducción y asegurarse de que tiene instaladas todas las herramientas necesarias.

¿Odia Windows o Xampp? No es problema. Obviamente, no necesita contar con Windows para leer este libro. Si está trabajando directamente en un entorno LAMP o algún otro, sólo necesita conocer los comandos de linux. No los incluyo aquí, pero puede buscarlos en google fácilmente. Sólo para aclarar, estas instrucciones son para xampp en Windows, pero sólo hay diferencias menores, así que debería ser capaz de resolverlas si está usando un sistema diferente.

Para su conveniencia, también incluyo un link a la guía de Yii 2 para la Instalación de la Aplicación Avanzada:

[Yii 2 Configuración de la Aplicación Avanzada](#)

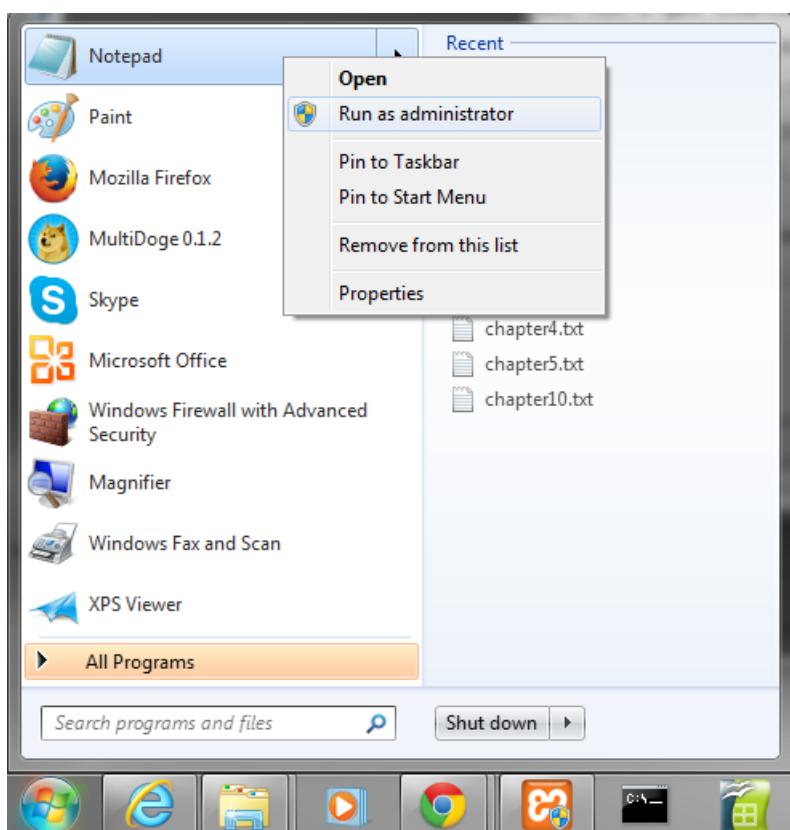
Bien, comencemos:

Paso 1 - Crear el Directorio

Diríjase al directorio que almacena las raíces de los proyectos, en mi caso es \var\www y cree un nuevo directorio llamado yii2build. Debería tener un directorio \var\www\yii2build.

Paso 2 - Configuración de Apache

Prepare la configuración de apache. Desde una máquina windows lo editaremos usando notepad, ejecutándolo como administrador. Encuentre notepad en el botón inicio de la barra de tareas. Realice un clic con el botón derecho del mouse y seleccione ejecutar como administrador.



Notepad en Modo Administrador

Notepad se abrirá. Seleccione abrir archivo y la ruta a vhosts, en mi caso:

C:\xampp\apache\conf\extra\

Seleccione Todos los archivos en la opción tipos de archivo:



Notepad Todos los archivos

Luego seleccione:

`httpd-vhosts.conf`

Añada la siguiente entrada al archivo:

```
NameVirtualHost *
<VirtualHost yii2build.com>
    DocumentRoot "C:\var\www\yii2build\frontend\web"
    ServerName localhost
    ServerAlias www.yii2build.com
</VirtualHost>
NameVirtualHost *
<VirtualHost yii2build.com>
    DocumentRoot "C:\var\www\yii2build\backend\web"
    ServerName localhost
    ServerAlias backend.yii2build.com
</VirtualHost>
```

Por favor tenga en cuenta que `c:\var\www` es el lugar donde almaceno el directorio de mi proyecto. Si lo ha colocado en un directorio diferente, `c:\xampp\htdocs` por ejemplo, necesitará usar eso en cambio en las entradas de host arriba mostradas.



Consejo de resolución de problemas

Asegúrese de que la línea: `Include "conf/extra/httpd-vhosts.conf"` se encuentra descomentada en su archivo `xampp/apache/conf/httpd`, de otra manera la configuración anterior no funcionará.

Paso 3 - Host Local

Configure una entrada en el host local:

En windows, abra notepad como administrador y diríjase a:

`c:\Windows\System32\drivers\etc`

seleccione Todos los archivos en tipos de archivo, y luego seleccione:

`hosts`

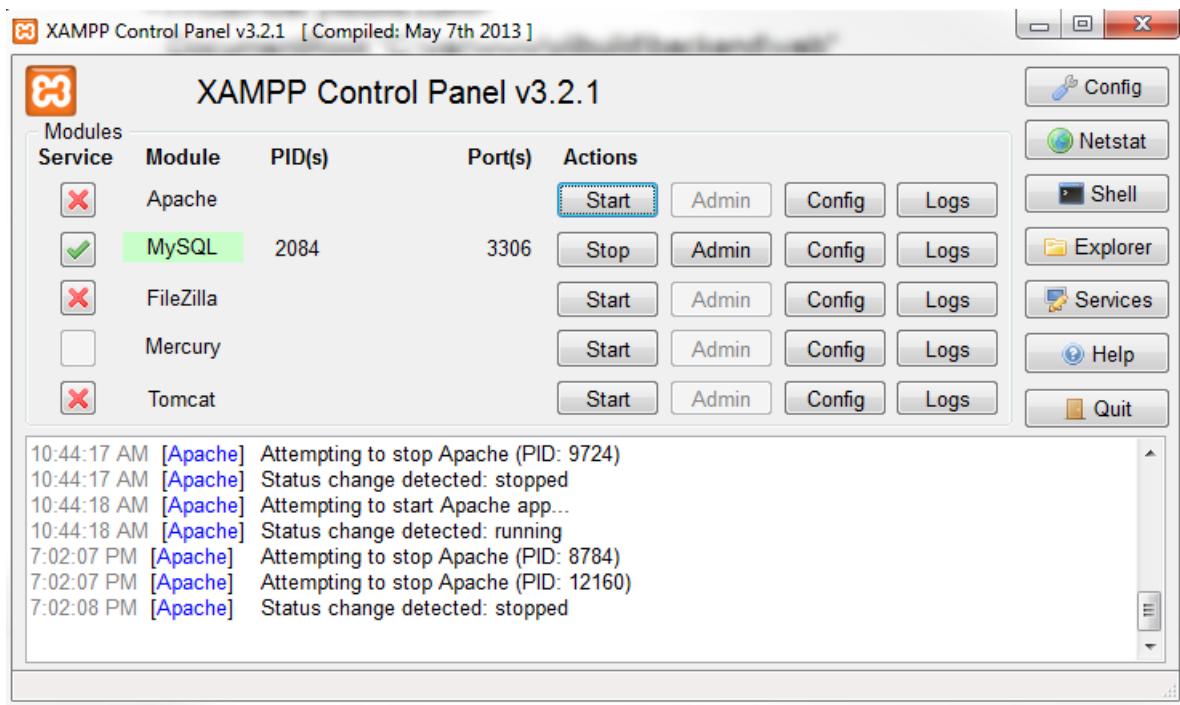
agregue lo siguiente y guarde:

127.0.0.1 yii2build.com www.yii2build.com

127.0.0.1 backend.yii2build.com

Paso 4 - Reiniciar Apache

Haga clic en el panel de control de xampp y reinicie apache:



Panel de Control de Xampp

Tenga en cuenta que estamos ejecutando MySQL como servicio, pero no apache. Si aún no ha configurado xampp, obviamente, necesitará hacerlo antes de continuar. Le recomiendo que tenga todas sus herramientas preparadas y configuradas antes de proceder y que se tome algún tiempo para familiarizarse con ellas. He incluído un link de video de xampp en el capítulo 1 al que también puede referirse.

Paso 5 - Crear el Proyecto en el IDE

Cree el proyecto yii2build en su IDE usando yii2build como directorio raíz. Si no está seguro de cómo hacer esto, busque en google un tutorial para el IDE que está usando. Tenga en cuenta que uso barras invertidas para designar una ruta dentro del IDE.

Ahora podemos realizar una prueba para comprobar si hemos configurado nuestros archivos de host files correctamente. En su directorio yii2build, cree un directorio denominado frontend y otro

llamado backend. Dentro de cada uno de estos directorios, cree un directorio llamado web. Ahora debería tener yii2build/frontend/web/ y yii2build/backend/web/.

Ahora cree un archivo php llamado index, con la única línea siguiente:

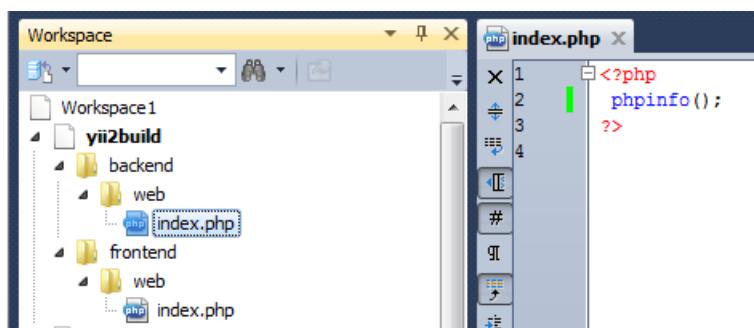
```
<?PHP
```

```
phpinfo();
```

```
?>
```

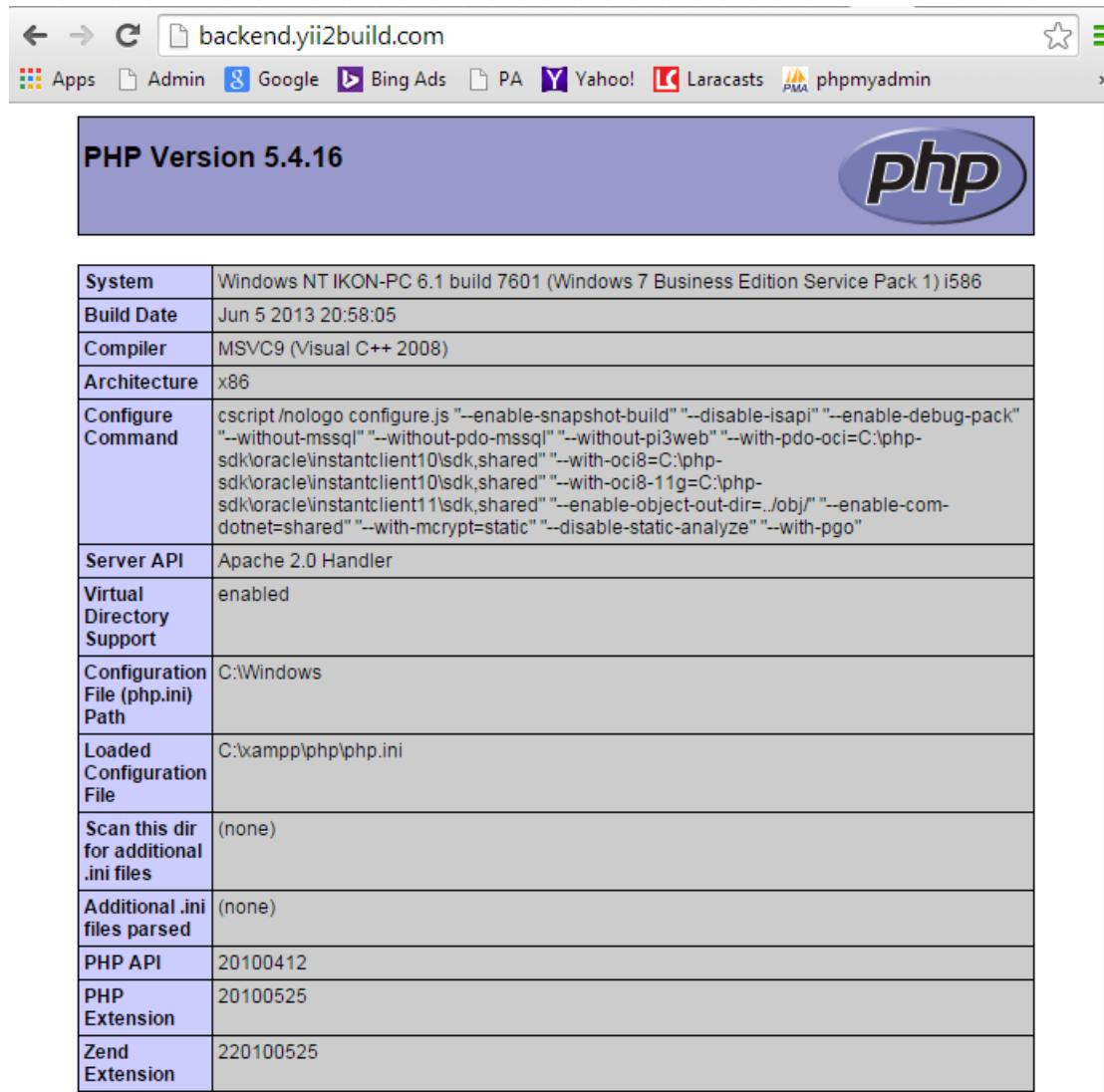
Guarde una copia de ambos directorios. Ahora debería tener:

```
yii2build/frontend/web/index.php  
yii2build/backend/web/index.php
```



Estructura de Directorio para el Host de Prueba

Si tipea yii2build.com y backend.yii2build.com en su navegador, ambos deberían mostrar la salida de phpinfo, lo que convenientemente también le da la oportunidad de ver si tiene instalada PHP 5.4 o superior, que es lo que necesita para ejecutar Yii 2.



The screenshot shows a web browser window with the URL `backend.yii2build.com` in the address bar. The page title is "PHP Version 5.4.16". The content is a table of PHP configuration parameters:

System	Windows NT IKON-PC 6.1 build 7601 (Windows 7 Business Edition Service Pack 1) i586
Build Date	Jun 5 2013 20:58:05
Compiler	MSVC9 (Visual C++ 2008)
Architecture	x86
Configure Command	cscript /nologo configure.js "--enable-snapshot-build" "--disable-isapi" "--enable-debug-pack" "--without-mssql" "--without-pdo-mssql" "--without-pi3web" "--with-pdo-oci=C:\php-sdk\oracle\instantclient10\ sdk\shared" "--with-oci8=C:\php-sdk\oracle\instantclient10\ sdk\shared" "--with-oci8-11g=C:\php-sdk\oracle\instantclient11\ sdk\shared" "--enable-object-out-dir=../obj/" "--enable-com-dotnet=shared" "--with-mcrypt=static" "--disable-static-analyze" "--with-pgo"
Server API	Apache 2.0 Handler
Virtual Directory Support	enabled
Configuration File (php.ini) Path	C:\Windows
Loaded Configuration File	C:\xampp\php\php.ini
Scan this dir for additional .ini files	(none)
Additional .ini files parsed	(none)
PHP API	20100412
PHP Extension	20100525
Zend Extension	220100525

Php Info

Si la página no se muestra, regrese y revise su archivo de hosts y/o su archivo `httpd-vhosts.conf`. Asegúrese de reiniciar Apache luego de realizar los cambios. Cerciórese de tener entradas en el host local para el dominio, `yii2build`. Refiérase a los pasos 2 y 3 si fuera necesario.

En este punto, debería ser capaz de ver que sus entradas de host son correctas y de que está ejecutando la versión correcta de Php. Esto es independiente de Yii 2 y composer, por lo que implementar exitosamente el paso 5 le da un punto de verificación para la primera parte de nuestra configuración.

Si todo funciona correctamente, habrá probado exitosamente sus entradas de host y debería eliminar los directorios web y sus contenidos. Obviamente debe dejar el directorio raíz, `yii2build`, en su lugar.

Consejo de solución de problemas: si está usando windows, puede tener problemas eliminando un directorio. Esto es debido a que los permisos del archivo han sido seteados a sólo lectura. Puede realizar un clic derecho en el directorio y usar el menú propiedades para realizar los ajustes. Use

Google para obtener detalles específicos si necesita ayuda con ello ya que puede variar dependiendo de la versión de Windows que esté usando.

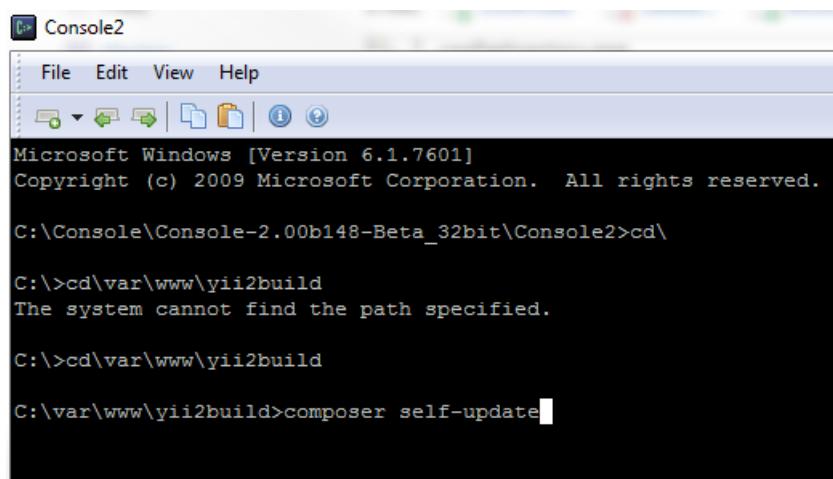
Paso 6 - Encontrar la Ruta de la Línea de Comandos

Desde la línea de comandos de windows, diríjase a su directorio yii2build. Primero cd\, luego cd\var\www\yii2build si es que tiene yii2build en \var\www. Se que esto puede ser algo confuso, así que permítame reiterarlo. \var es un directorio en mi disco c:, dentro hay un directorio llamado www, y dentro de www, he creado un directorio denominado yii2build, dónde residirá el proyecto.

No tiene que seguir esto exactamente, sólo necesita saber dónde se encuentra su directorio raíz y asegurarse de que tiene las entradas de host apropiadas.

Paso 7 - Composer Self-Update

Asegúrese de que composer está instalado y actualizado. Desde la línea de comandos, en el directorio raíz de su proyecto, debería ejecutar: composer self-update.



```
C:\Console\Console-2.00b148-Beta_32bit\Console2>cd\

C:\>cd\var\www\yii2build
The system cannot find the path specified.

C:\>cd\var\www\yii2build

C:\var\www\yii2build>composer self-update
```

composer self-update

Si obtiene un mensaje de error, revise su instalación de composer. Si no ha instalado composer, busque en Google instrucciones de instalación en windows y xampp.

También deberá asegurarse de que el siguiente plugin está instalado en composer. Envíe el siguiente comando desde el mismo directorio donde realizó self-update:

```
composer global require "fxp/composer-asset-plugin:1.0.0-beta4"
```

```
C:\var\www\yii2build>composer self-update
Updating to version a309e1d89ded6919935a842faeaed8e888fbfe37.
  Downloading: 100%
Use composer self-update --rollback to return to version d79f2b0fd33ee9b89f3d9f1969f43dc3d570a33a
C:\var\www\yii2build>composer global require "fxp/composer-asset-plugin:1.0.*@dev"
```

Asset Plugin

Si el plugin arriba mostrado no está instalado, composer no se comportará correctamente. La buena noticia es que mientras tenga composer funcionando, el plugin es fácil de instalar con el único simple comando de arriba. Por favor tenga en cuenta que para acceder al plugin, puede tener que ingresar con su cuenta de Github porque puede solicitarle su usuario y contraseña. Si no tiene una cuenta de Github, simplemente diríjase a [Github.com](https://github.com) y regístrese con una cuenta gratuita. Sólo toma un minuto y es gratis. Tendrá que hacerlo sólo una vez.



Consejo

He revisado la Guía de Yii 2 y la última versión recomendada para el plugin es “fxp/composer-asset-plugin:1.0.0-beta4”. Si por cualquier razón, esa versión del plugin estuviera desactualizada, use Google para encontrar la versión correcta. También puede intentar @dev, lo que debería funcionar, pero nunca se sabe. Haré mi mayor esfuerzo para mantener el libro actualizado, pero estas son el tipo de cosas sobre las que será difícil mantenerse al tanto. Al recorrer la configuración en libros de programación, estos son problemas comunes, así que esto es sólo un aviso.

Paso 8 - Instalar Yii 2

Instale Yii 2 vía composer. Hacemos esto desde la línea de comandos en el directorio arriba señalado:

```
composer create-project --prefer-dist yiisoft/yii2-app-advanced
```

```
C:\var\www\yii2build>composer create-project --prefer-dist yiisoft/yii2-app-advanced
```

Instalación de Yii 2 con Composer

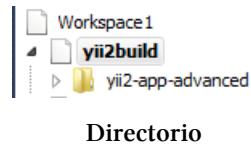


Consejo

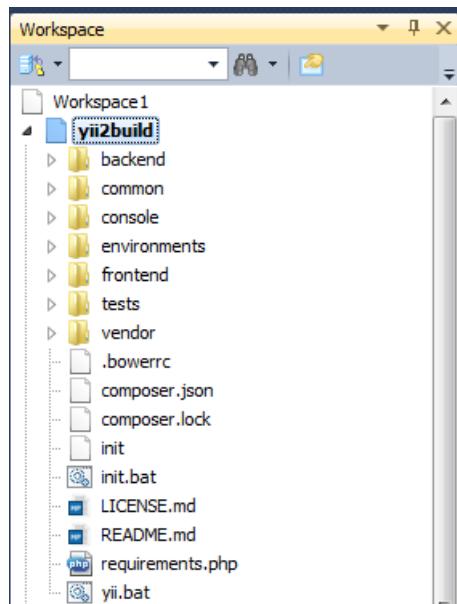
Las instrucciones en la guía son ligeramente diferentes en el hecho de que puede crear el directorio del proyecto nombrándolo como el último parámetro de la instalación. Esto puede resultar confuso para principiantes sin embargo, por lo que recomiendo que siga estas instrucciones, que tienen un paso extra, pero le permiten comprobar que las entradas de host están funcionando antes de instalar Yii 2.

Paso 9 - Revisar su directorio Yii 2

Ahora tendrá un directorio llamado yii2-app-advanced en su directorio yii2build.



Usando el explorador de windows, abra este directorio, y verá todos los archivos del framework. Seleccione todos los archivos y copielos un nivel arriba en el directorio raíz yii2build, luego **borre** el directorio yii2-app-advanced. Sólo para ser perfectamente claros, ahora debería tener el directorio raíz, en este caso yii2build, con los archivos del framework dentro en el primer nivel. No debería haber ningún directorio yii2-app-advanced en este punto. Debería verse así luego de eliminarlo:



Paso 10 - Ejecutar Php Init

Nuevamente en la línea de comandos. En la \ruta\al\yii2build, en mi caso es \var\www\yii2build, ejecute: php init.

```
C:\>cd\var\www\yii2build
C:\var\www\yii2build>php init
```

Ejecutar PHP Init

Le preguntará si desea inicializar en desarrollo o producción. Seleccione **0** para desarrollo. Luego confirme **Si**.

```
C:\var\www\yii2build>php init
Yii Application Initialization Tool v1.0

Which environment do you want the application to be initialized in?

[0] Development
[1] Production

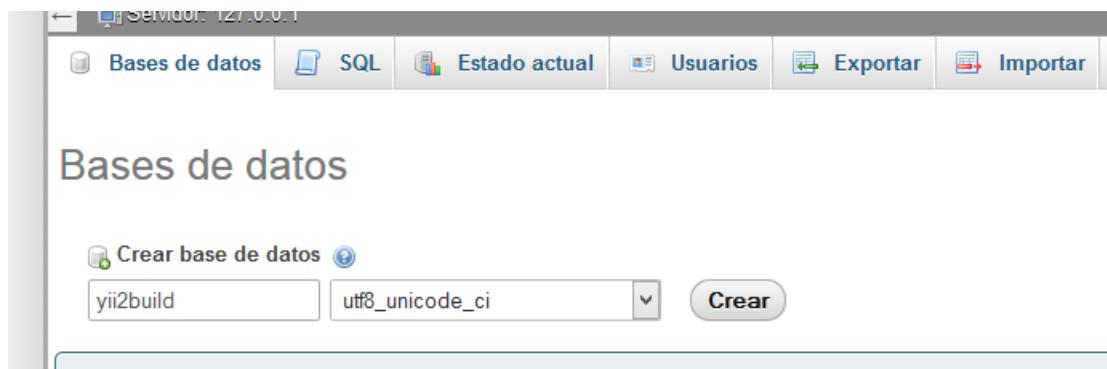
Your choice [0-1, or "q" to quit] 0

Initialize the application under 'Development' environment? [yes|no] yes
```

Configuración de Desarrollo

Paso 11 - Crear la Base de Datos

Cree la base de datos. Diríjase a Phpmyadmin. Vaya a la pestaña bases de datos, cree una bd llamada yii2build, con cotejamiento utf8_unicode_ci.



Crear la Base de Datos

Paso 12 - Establecer la conexión a la BD

Ajuste el arreglo de componentes en yii2build/common/config/main-local.php según corresponda. Debería verse así:

```
'db' => [
    'class' => 'yii\db\Connection',
    'dsn' => 'mysql:host=localhost;dbname=yii2build',
    'username' => 'root',
    'password' => 'yourpassword',
    'charset' => 'utf8',
],
]
```

Obviamente sustituya por su contraseña real en config. No olvide guardar los cambios.

Paso 13 - Ejecutar la Migración

De regreso a la línea de comandos. Puede tener una ruta diferente, si es así, debería ser ruta\al\directorioraiz\yii2build>yii migrate. En mi configuración, es \var\www\yii2build, ejecutar yii migrate. Se ve así:

```
\var\www\yii2build>yii migrate
```

```
... initialization completed.

C:\var\www\yii2build>yii migrate[]
```

Migrar

Confirme si (yes).

```
C:\var\www\yii2build>yii migrate
Yii Migration Tool (based on Yii v2.0.0)

Creating migration history table "migration"...done.
Total 1 new migration to be applied:
    m130524_201442_init

Apply the above migration? (yes|no) [no]:
```

Confirmar Si

Esto construirá las tablas necesarias en su base de datos. Puede revisar PhpMyadmin donde debería tener las siguientes tablas en yii2build:

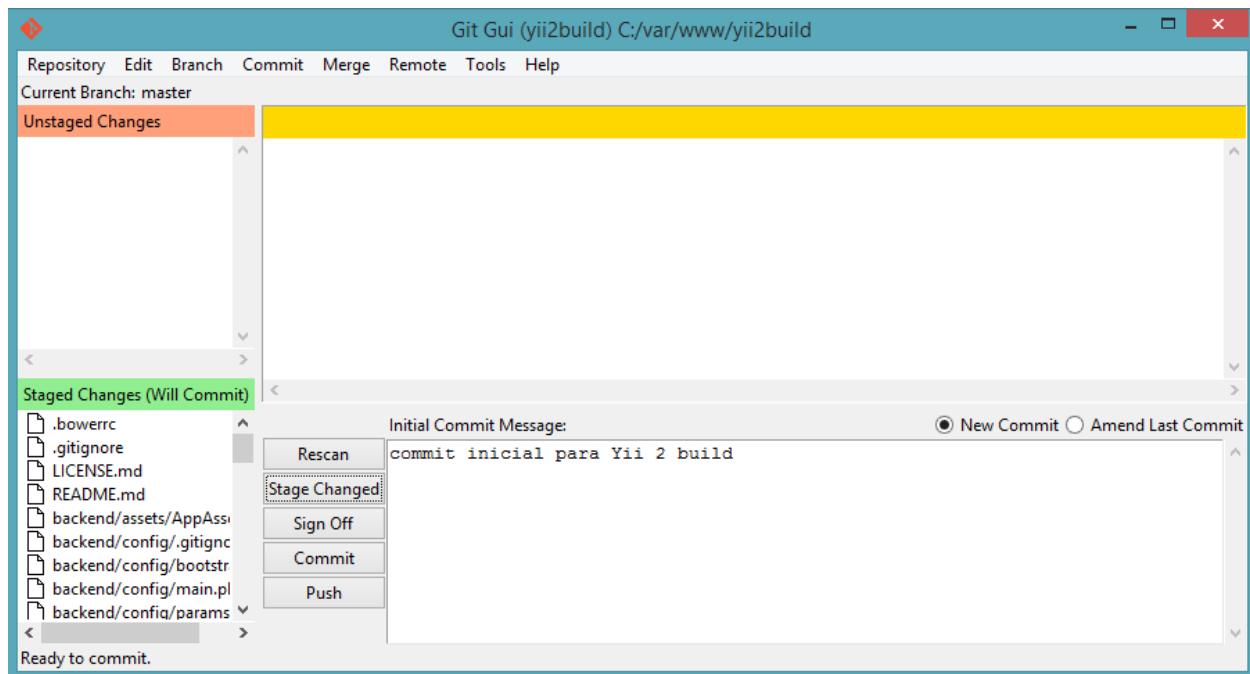
```
migrations
user
```

Tabla	Acción	Filas	Tipo	Cotejamiento	Tamaño
migration	Examinar Estructura Buscar Insertar Vaciar Eliminar	2	InnoDB	utf8_unicode_ci	16 KB
user	Examinar Estructura Buscar Insertar Vaciar Eliminar	0	InnoDB	utf8_unicode_ci	16 KB
2 tablas	Número de filas			2 InnoDB	utf8_unicode_ci 32 KB

Migración Exitosa

Paso 14 - Crear el Repositorio en Git

Paso 14. Cree el repositorio en GIT. Abra la interfaz gráfica de Git. Seleccione “create new repository”. Seleccione yii2build. Seleccione “stage changed”. Puede aparecer una advertencia sobre los índices. Seleccione continuar (continue) si esto ocurre. Escriba commit inicial para Yii 2 build en el área de mensaje, luego seleccione commit.



Commit en Git

Tiene que guardar los cambios antes de realizar el commit. Puede tener que desbloquear el índice o hacer clic en continuar en una ventana emergente. También necesitará ingresar un comentario antes de realizar el commit.

Para ver el repositorio desde el menú repositorio, seleccione “visualize all branch history”. Esto le mostrará la branch principal actual y su historia, algo grandioso para mantenerse al tanto de los

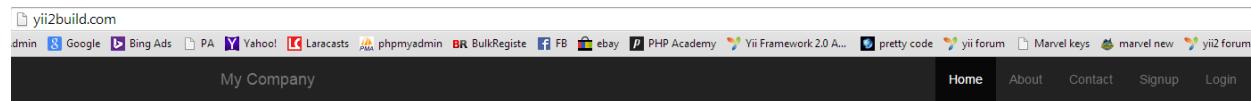
cambios y deshacerlos si es necesario. El control de versión es muy importante para un proyecto de este tamaño. Es improbable que pudiera realizar el proyecto sin él, así que no salte este paso. Recuerde realizar, como mínimo, un commit al final de cada capítulo de este libro. Probablemente debería realizarlo con mayor frecuencia.

Por favor tenga en cuenta que no es necesario que guarde su proyecto en un repositorio de Github, puede hacerlo si gusta, pero no cubrimos eso en este libro. Sólo usamos GIT para un control de versión local.

Paso 15 - Confirmar que la Aplicación está funcionando

Confirme que la aplicación avanzada funciona tipeando yii2build.com en su navegador. Debería ver la plantilla de la aplicación avanzada que le permitirá registrar un usuario e iniciar sesión con ese usuario.

Debería registrar un usuario e ingresar para probar que la aplicación está funcionando. Una vez que cree un usuario, puede testear el backend también.



Congratulations!

You have successfully created your Yii-powered application.

[Get started with Yii](#)

Heading

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.

Heading

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.

Heading

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.

Yii 2 Build

Ya que en este punto no hay control de acceso diferenciado entre el frontend y el backend, puede iniciar sesión en el backend yendo a backend.yii2build.com e ingresando. En ambos casos, el inicio de sesión simplemente lo llevará a la página principal y mostrará en la barra de navegación el nombre de usuario y el link para salir.

Solución de problemas

Si no se muestra, entonces revise su archivo de hosts y su archivo httpd-vhosts.conf. Asegúrese de que Apache se está ejecutando y quexampp se ha reiniciado luego de realizar cambios a los archivos host. Cerciórese de que su versión de PHP es 5.4 o mayor, lo que es necesario para Yii 2.

Si ve un árbol de directorio, en lugar de la página principal, entonces no ha ejecutado exitosamente init, regrese al paso 10. Si puede ver la página principal, pero obtiene un error cuando trata de registrarse, asegúrese en PhpMyAdmin que la base yii2build existe, que tiene la contraseña correcta para ella, y que ha ingresado esos datos en yii2build/common/conf/main-local.php. También asegúrese de que Mysql se está ejecutando enxampp, vea la foto en el paso 4 como referencia.

Si aún así no logra que funcione, comience nuevamente o al menos desde el punto en donde confirmó que las entradas de host estaban trabajando y de que estaba ejecutando PHP 5.4.

Resumen

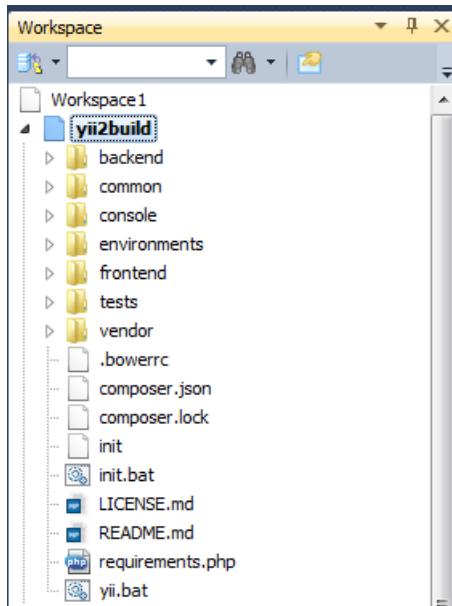
Felicitaciones, la parte más difícil del libro ha concluido. Con suerte, todo habrá ido bien para usted. Si tuvo problemas con la configuración, repita los pasos hasta que lo logre. Si está seguro de que todo está bien, pero aún no funciona, consulte la documentación individual de los componentes para ver si algo ha cambiado desde que este libro fue escrito. Google típicamente es efectivo para esto, cuando se lo necesita.

En los próximos capítulos, comenzaremos a trabajar en el desarrollo con Yii 2. Comenzaremos con breve recorrido por la arquitectura MVC, pero no pasaremos demasiado tiempo en la teoría, a menos que podamos usarla para codificar. En su lugar, nos sumergiremos rápidamente en los capítulos subsiguientes.

He aprendido a través de la experiencia personal que las explicaciones de conceptos más amplios resultan mejor cuando están acompañadas de la implementación práctica, lo que es la razón de que no haya aprendido casi nada de la mayoría de mis lecciones de POO en línea, sólo impresiones vagas de interfaces y herencia de clases. A no preocuparse. Una de las cosas grandiosas sobre Yii 2 es que unifica tantos de los principios de la POO de una manera tan intuitiva, que comprenderá la teoría a medida que avance. Al menos la verá demostrada.

Capítulo Tres: Bienvenido al MVC

Ahora que tenemos instalada la plantilla avanzada, tomémonos unos minutos para familiarizarnos con la estructura de nuestra aplicación. Así que aquí estamos:

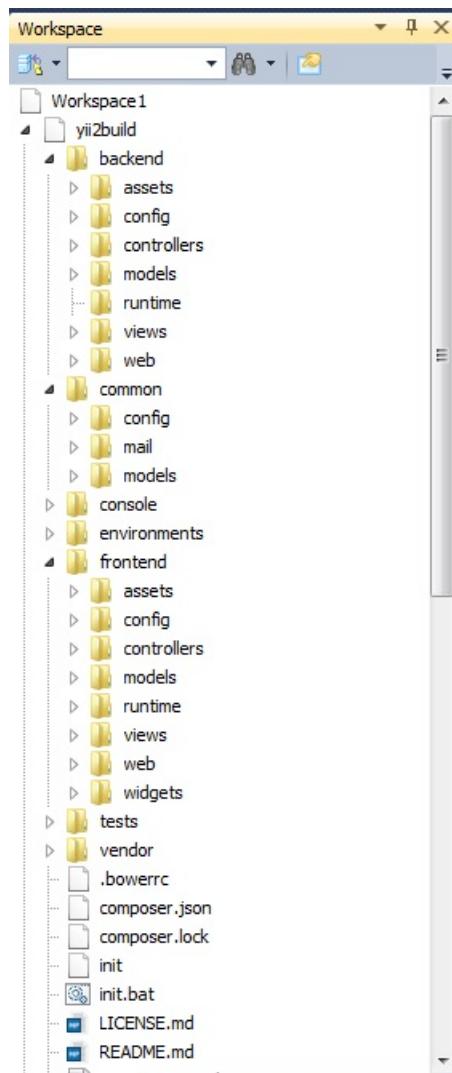


Estructura de directorios

Puede ver que la aplicación se encuentra dividida en los directorios backend, common, console, environments, frontend, tests, y vendor.

Patrón MVC

Yii 2 sigue el patrón MVC, dónde M significa Modelo, V significa Vista, y C significa controlador. Vamos a discutir esto brevemente, pero sólo como una visión general. La mejor manera de comprender como funciona es trabajar con el código y las estructuras de directorio directamente, lo que haremos en breve. Aquí hay otra vista de la estructura con algunos directorios abiertos:



Estructura de la Aplicación

Puede ver que los directorios backend y frontend tienen directorios denominados models, controllers, y views. El directorio common tiene models, pero no controller o views. Puede querer tomarse unos momentos para revisar todos los directorios y ver que hay en su interior.

En Yii 2, el modelo es responsable de ingresar y recuperar datos de la base de datos. Esto incluye cualquier relación que necesite con modelos conectados, por ejemplo, un usuario y un perfil de usuario.

Cuando una solicitud web llega, el controlador típicamente la rutea al modelo, quien se comunica con la base de datos, luego regresa los resultados para ser visualizados en la vista. Esto permite separación entre la lógica y la presentación. Obtiene modelos “gordos” llenos de php, y controladores “delgados” que en su mayor parte sólo se encargan del ruteo, y vistas que emplean poco PHP y usan más HTML y javascript para realizar la presentación.

Esto es probablemente todo lo que necesitamos decir sobre teoría abstracta. Funciona bien y veremos

como Yii 2 implementa este patrón y lo fácil que es comprenderlo en la práctica.

Index.php

Hay exactamente dos puntos en esta aplicación que deberían ser accesibles desde la web. Tanto el backend como el frontend tienen un directorio llamado web en su interior y dentro de ese directorio hay un archivo llamado index.php. Si lo recuerda, configuramos nuestras entradas de hosts para buscar ese archivo, así que backend.yii2build.com se dirige a la versión que se encuentra en el directorio backend y yii2build.com se dirige a la del frontend. Cada uno de esos archivos es idéntico y se ve así:

```
<?php
defined('YII_DEBUG') or define('YII_DEBUG', true);
defined('YII_ENV') or define('YII_ENV', 'dev');

require(__DIR__ . '/../vendor/autoload.php');
require(__DIR__ . '/../vendor/yiisoft/yii2/Yii.php');
require(__DIR__ . '/../common/config/bootstrap.php');
require(__DIR__ . '/config/bootstrap.php');

$config = yii\helpers\ArrayHelper::merge(
    require(__DIR__ . '/../common/config/main.php'),
    require(__DIR__ . '/../common/config/main-local.php'),
    require(__DIR__ . '/config/main.php'),
    require(__DIR__ . '/config/main-local.php')
);

$application = new yii\web\Application($config);
$application->run();
```

Las primeras dos líneas revisan si las constantes para debug y dev existen o en su defecto las define.

```
defined('YII_DEBUG') or define('YII_DEBUG', true);
defined('YII_ENV') or define('YII_ENV', 'dev');
```

Luego se encuentran las sentencias require para los archivos que son necesarios a fin de ejecutar la aplicación, incluyendo el autoloader:

```
require(__DIR__ . '/../vendor/autoload.php');
require(__DIR__ . '/../vendor/yiisoft/yii2/Yii.php');
require(__DIR__ . '/../common/config/bootstrap.php');
require(__DIR__ . '/config/bootstrap.php');
```

\$config es configurado por el método merge de ArrayHelper, el cual requiere los archivos especificados:

```
$config = yii\helpers\ArrayHelper::merge(
    require(__DIR__ . '/../common/config/main.php'),
    require(__DIR__ . '/../common/config/main-local.php'),
    require(__DIR__ . '/config/main.php'),
    require(__DIR__ . '/config/main-local.php')
);
```

Puede ver que sube dos directorios para encontrar el directorio common para esa configuración. Luego sube un directorio para encontrar la configuración para el frontend o backend, dependiendo de qué archivo index.php realice la llamada.

Luego finalmente, creamos una nueva instancia del modelo de la aplicación, cargando la configuración en el constructor, así que ahora \$application se convierte en la instancia de la aplicación. Luego lanzamos su método run:

```
$application = new yii\web\Application($config);
$application->run();
```

La Instancia de la Aplicación

La aplicación se encuentra ahora disponible globalmente como Yii::\$app. Existen muchos métodos importantes disponibles desde Yii::\$app y hablaremos sobre ellos luego ya que es muy conveniente llamarlos.

No se desanime si no entiende esto ahora mismo. Aprenderá la arquitectura con el tiempo, este capítulo sólo busca ser una introducción.

Hablando de manera general, me mantengo a distancia del ámbito de la arquitectura de alto nivel y me concentro más en lo que se encuentra frente a nosotros, los entresijos de cómo hacer que funcione.

Ruteo

Así que regresemos al index.php, el archivo actúa como un portal a la aplicación, creando una instancia de ella. Cuando estamos escribiendo una url para nuestra aplicación, siempre llamamos a index.php. Yii 2 maneja todo el ruteo por nosotros, de manera que cuando queremos ir a la página de inicio de nuestro sitio por ejemplo, la ruta se ve así:

```
yii2build.com/index.php?r=site/index
```

No luce muy bonita. Puede configurar url “agradables” en el config, lo que ayuda a su amigabilidad para los motores de búsqueda y también puede eliminar la necesidad de mostrar index.php en la url, pero no cubriremos eso hasta el capítulo 13. Al esperar, eliminamos la necesidad de tener que depurar la url o apache, si un problema con la página se presentara.

Bien, de regreso al ruteo:

La parte r=site/index le dice a Yii 2 que deseamos el controlador del sitio y la acción index. Si una solicitud entrante no especifica una ruta, lo que ocurre cuando alguien tipea simplemente yii2build.com por ejemplo, entonces, la ruta especificada por yii\web\Application::\$defaultRoute se usará. El valor por defecto es site/index, que, como mencionamos arriba, especifica el controlador del sitio y la acción index.

Si no se especifica una acción, el controlador asume que desea la acción index. Ejemplo:

```
yii2build.com/index.php?r=site
```

Esto retorna la acción index del controlador del sitio. En la mayoría de los casos, la acción mostrará la vista asociada, una vista con el mismo nombre de la acción del controlador. Acciones y vistas comunes son index, view, create, update, delete.

Usualmente nos referimos a las acciones create, read, update, and delete como CRUD.

Usando Gii

Usaremos el módulo estrella de Yii 2, Gii, la herramienta de generación de código más grandiosa jamás construida, para ayudarnos a realizar mucho del CRUD. Y cuando usamos Gii para crear CRUD, usualmente creamos el controlador al mismo tiempo, por lo que podemos esperar en forma genérica que el CRUD incluya el controlador. No se preocupe si esto es un poco confuso ahora, tendrá mucho más sentido luego cuando creemos nuestros archivos. Y si, adoro Gii, y estoy muy seguro de que para cuando hayamos acabado, usted también lo hará.

Si mira en el directorio views bajo frontend, puede ver un directorio llamado site, que tiene un archivo index.php en su interior. Esta es la página de vista que es mostrada por la acción index del controlador del sitio. El controlador del sitio en sí mismo se encuentra en frontend/controllers/SiteController.php

Navegue por los directorios. Dentro de backend, encontrará los directorios controllers, models, views. Hallará lo mismo en el directorio frontend. En el directorio common, verá los directorios

config, mail, y models. En general, puede apreciar la consistencia en las convenciones de nombre y ella hace que Yii 2 sea fácil de comprender desde el punto de vista del MVC.

Así que obviamente, esto es muy diferente de las aplicaciones web simples dónde tendría una url como sitioejemplo.com/sobre.php. Si está tentado a evitar aprender Yii 2 porque los requerimientos actuales de su aplicación no necesitan ser tan robustos, tenga en mente que, con el tiempo, los requisitos de las aplicaciones tienden a crecer.

Si hoy su cliente no necesita un formulario con reglas de validación robustas, no significa que no lo necesitará mañana.

Bootstrap

Además, con Yii 2, obtiene de forma predeterminada integración con el framework frontend Bootstrap. Si no está familiarizado con el framework Bootstrap de Twitter, le recomiendo que lo revise, se ha convertido rápidamente en el estándar de la industria. Puede encontrarlo aquí:

Obtener Bootstrap

No necesita descargar o hacer nada sin embargo, porque como he dicho antes, Yii 2 viene con él integrado por defecto. Esto significa que obtiene css adaptativo que escala junto con el dispositivo, permitiéndole crear diseño centrado en móviles desde el comienzo. ¡Y eso, mis amigos, es sólo la cereza del postre!

Un día su cliente quiere un sitio web trivial y al día siguiente quiere css para móviles. Puede cumplir porque ya se encuentra allí. En fin, no estoy tratando de sonar como un vendedor. Verdaderamente amo esta plataforma y eso se nota.

En nuestros proyectos anteriores, podemos haber creado archivos para cabecera y pie de página, simple pero ineficiente. ¿Qué sucede si ha olvidado incluir el archivo o ha tipeado incorrectamente una versión anterior? ¿Qué hay del uso de temas y otros enfoques avanzados?

Yii 2 tiene una solución genial para ello en el uso de layouts. Las vistas se inyectan en el layout y existen métodos disponibles a nivel de configuración del sitio o a nivel del controlador para especificar qué layout usar. Ya existe un layout por defecto, así que no necesita hacer nada sino quiere cambiarlo. También puede usar layouts anidados, si cree que es necesario.

Para nuestros propósitos, vamos a mantener el layout por defecto, que se encuentra en `frontend/views/layouts/main.php`. La única cosa que vamos a considerar en este momento es esta etiqueta en medio de la página:

```
<?= $content ?>
```

Aquí es donde se inyecta la vista de la página. Así que ahora sabe que la cabecera se encuentra arriba de `$content` y el pie de página se encuentra debajo de él. No se preocupe, estaremos realizando cambios a este archivo y regresaremos a él más adelante en el proyecto.

Depurador

Otra cosa que debemos mencionar es el conspicuo depurador de Yii en la parte inferior de la página, cuando ve la plantilla avanzada en el navegador. Este tiene muchas funcionalidades útiles, tales como

- Configuración
- Logs
- Profiling
- Base de Datos
- Asset Bundles
- Correo Electrónico

No le dedicamos mucho tiempo en este libro, pero en su flujo de programación, esto es increíblemente útil. Puede revisar qué sentencias se ejecutan, cuánto tiempo toman y muchos otros detalles útiles sobre cómo funciona su aplicación. Tómese algún tiempo para familiarizarse con él. Lo comprenderá simplemente jugando con él. Si no lo está usando, puede hacer clic en la flecha en la esquina inferior derecha del navegador y ocultarlo.

Resumen

Como dijimos en la introducción, Yii 2 no es una implementación trivial del patrón MVC. Es un framework extenso que es robusto y fácil de usar, una vez que se familiariza con él. La curva de aprendizaje para los principales puede ser empinada, pero continúe, vale la pena.

Voy a hacer todo lo posible para ayudarlo, método por método. Será un poco confuso al principio, pero a medida que avancemos, y nos sumerjamos en el proyecto, comenzará a tener sentido, y las piezas comenzarán a encajar.

Así que, ¿qué deberíamos construir como nuestra aplicación de ejemplo? ¿Qué demostraría características útiles que muchos proyectos comparten? ¿Y podemos usar algo de lo que construyamos aquí en un proyecto real?

Hacerme esas preguntas me llevó a concluir que este libro debería construir una plantilla de aplicación llamada Yii 2 Build. Pero espera un minuto, ¿acaso la instalación de la aplicación avanzada no es en sí misma una plantilla? Si lo es. Pero vamos a llevarla un poco más lejos.

Vamos a crear un sistema RBAC básico que nos permita definir roles, estados, y tipos de usuario para controlar el acceso tanto al frontend como al backend de la aplicación. Vamos a construir un controlador upgrade, de forma tal que si deseamos un área paga para nuestra aplicación, podamos forzar esa regla.

También vamos a crear un modelo de perfil de usuario que puede ser extendido o modificado para cubrir sus necesidades, pero que nos muestra cómo controlar el acceso a las vistas que deberían ser privadas para el usuario que las posee.

Nuestra meta será crear una aplicación funcional que pueda usar como modelo para futuros proyectos, una que esté mucho más desarrollada que la plantilla de la aplicación avanzada. A pesar de lo genial que es la plantilla predefinida, podemos hacer mucho más, y aprender los pormenores del framework en el proceso. ¡Comencemos y divirtámonos un poco!

También tenga en cuenta que al final de los capítulos, verá:



Este es un recordatorio para realizar un commit de sus cambios al control de versión. No hay necesidad de hacerlo ahora, ya que no hemos cambiado nada. Pero manténgase al tanto, será de una gran ayuda si necesita deshacer un paso por cualquier razón.

Capítulo Cuatro: Modificando el Modelo de Usuario

Ahora vamos a concentrarnos en el modelo de Usuario. La plantilla avanzada de Yii 2 nos entregó un modelo de Usuario funcional, dónde los usuarios pueden iniciar sesión y recuperar su contraseña si es necesario, pero vamos a modificarlo. Así que lo primero que tenemos que comprender es por qué vamos a modificarlo.

Ahora mismo nuestra aplicación no trata el inicio de sesión del frontend de forma distinta al del backend. Pero el punto de crear áreas separadas para frontend y backend es forzar diferentes niveles de acceso.

Yii 2 sólo llega hasta cierto punto de forma predeterminada. Le deja a usted un amplio rango de implementaciones sobre cómo quiere manejar el acceso a su aplicación. La parte de la autenticación que determina combinaciones válidas de usuario y contraseña, ya se encuentra de manera predefinida en la plantilla de la aplicación avanzada. Sabemos esto porque podemos iniciar sesión y registrarnos como usuario.

La autorización, que determina la forma en que los usuarios tienen acceso a diferentes páginas se deja para nosotros. Yii 2 tiene un componente RBAC (control de acceso basado en rol) predefinido que podemos usar, pero yo prefiero un enfoque diferente.

Normalmente, trato de hacer las cosas de la manera en que Yii 2 las espera, algunos de los mejores programadores del mundo en PHP han trabajado en este framework y realmente han pensado en prácticamente todo. Por otra parte, una solución de RBAC involucra muchas decisiones personales así que en ocasiones es mejor desarrollarlo de forma personalizada, lo que lleva más cerca del código.

De cualquier modo, quiero proveer de una solución funcional para el RBAC, sin demasiadas molestias. Pero aún quiero ser capaz de controlar la autorización y control de acceso a través de una IU en el backend que me permita crear roles y administrar usuarios y sus estados. Explicaré más a medida que avancemos.

Nuestra meta es construir una plantilla que podamos usar para muchas clases diferentes de proyectos, así que por ejemplo, ¿qué sucedería si tuviéramos un sitio que necesitara permisos para usuarios gratuitos y pagos? Eso es más un tipo de usuario, que un rol. Pienso en un rol como algo más básico tal como admin, usuario, representante de servicio al cliente, etc. El rol describe su relación con la aplicación.

El tipo de usuario, por otra parte, es un concepto flexible que puede aplicarse a usuarios gratuitos versus usuarios pagos, o diferentes tipos de usuarios del frontend. Por ejemplo, si usted tuviera un sitio de música y algunos usuarios fueran músicos mientras que otros fueran simplemente fanáticos.

También es importante considerar el estado, cuando estamos ensamblando nuestro esquema. Necesitamos una manera de determinar si un usuario está activo, pendiente, o retirado o cualquier otra denominación que se nos pueda ocurrir.

La flexibilidad es clave, y he encontrado que al crear modelos separados para estos conceptos, es más fácil manipularlos. Así que luego de prueba y error y varias iteraciones, encontré un enfoque simple pero poderoso para la autorización. Nos llevará a través de múltiples configuraciones de modelos y nos introducirá en las relaciones. La mayoría del código es muy, muy simple.

Verá lo fácil que es crear todo esto con Gii, el generador de código integrado de Yii. Probablemente ha escuchado sobre Gii y es una herramienta asombrosa, pero no estamos listos para usarla aún, regresaremos a ella en el próximo capítulo.

En lugar de ello, simplemente necesitamos comenzar con algunas modificaciones a lo que ya tenemos. La tabla user (*N. del T. de aquí en adelante, se hará referencia a la tabla que contiene los datos del usuario, usando su nombre original “user”, con el que fuera creada por la migración inicial y que se ha conservado. Lo mismo se aplica al modelo correspondiente, que se seguirá denominando “User”*) y el modelo User fueron creados para nosotros de forma automática por la aplicación avanzada cuando la instalamos, y aunque se acerca a lo que necesitamos, tenemos que realizar algunos cambios importantes.

Rol y Estado

Verá que en la base de datos, hay una columna para el estado (status) en la tabla user. También necesitaremos una columna para el rol del usuario. Tanto rol como estado son de tipo de datos int. El problema es que queremos crear una tabla llamada rol y una llamada estado y queremos darle a estas nuevas tablas los nombres más intuitivos, que son, y no trato de hacerme el gracioso, rol y estado. Recuerde que en Mysql, usamos letras minúsculas como convención.

Así que no tiene sentido, y de hecho ocasionaría problemas de ambigüedad si tuviéramos una columna estado en la tabla user. La ambigüedad ocurre cuando una consulta de Mysql no puede determinar el campo o tabla correcta de donde obtener los datos debido a una confusión en los nombres. Esto puede ocurrir con la columna ‘id’, que la mayoría de las tablas tendrán. Si tuviéramos una columna llamada rol en la tabla user y una tabla diferente denominada rol, provocaría esta clase de problemas que pueden ser difíciles de depurar. Así que haremos nuestro mayor esfuerzo por evitarlos completamente.

En este caso, necesitamos cambiar la columna **estado (status)** de la tabla user, a **estado_id**, y agregar una nueva columna para **rol_id**. La mejor forma de hacer esto, si está usando todas las herramientas, es a través de Mysql Workbench. Es un poco más gráfica que PhpMyadmin, pero cualquiera servirá si tiene una preferencia.

Así que agreguemos una columna **rol_id** en la tabla user y asegurémonos de que tenga un valor por defecto de ‘1’. De manera similar, cambiemos status a estado_id y démosle también un valor por defecto de ‘1’. Ahora agreguemos una columna más a la tabla user, y llamemosla tipo_usuario_id, y démosle también un valor por defecto de ‘1’.

Aquí está una captura de pantalla de Mysql Workbench:

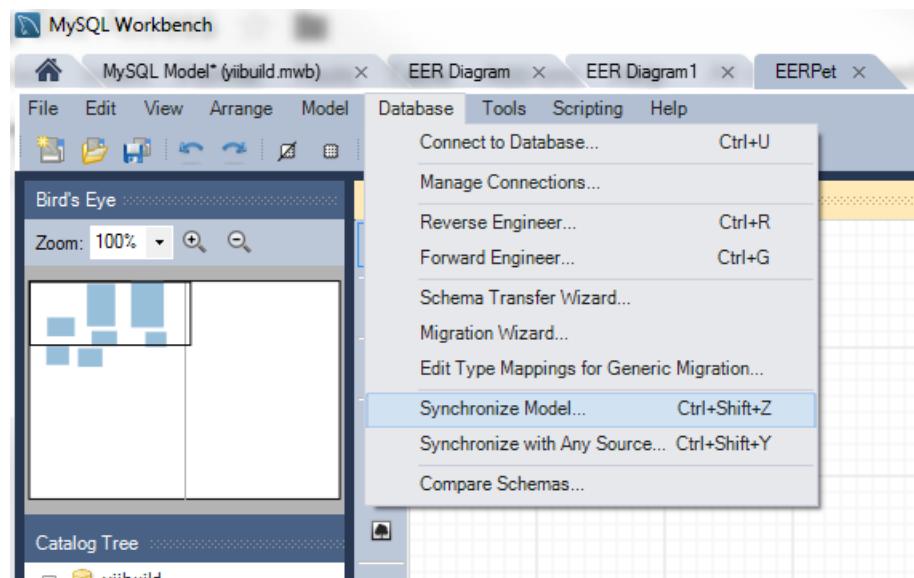
Column Name	Datatype	PK	NN	UQ	BIN	UN	ZF	AI	Default
id	INT(11)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
username	VARCHAR(255)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
auth_key	VARCHAR(32)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
password_hash	VARCHAR(255)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
password_reset_token	VARCHAR(255)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
email	VARCHAR(255)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
rol_id	SMALLINT(6)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	'1'
estado_id	SMALLINT(6)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	'1'
tipo_usuario_id	SMALLINT(6)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	'1'
created_at	DATETIME	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
updated_at	DATETIME	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	

Tabla User en Mysql Workbench

Note el uso de unsigned en la columna id. Esto significa que id no puede ser un número negativo. También asegúrese de que los campos created_at y updated_at sean del tipo DATETIME. Por alguna razón el build inicial está configurado para guardar como int esos campos, pero vamos a trabajar con behaviors en el modelo User para asegurarnos de que sean guardados correctamente como DATETIME.

Nota: si ha seguido instrucciones anteriores y ha creado un usuario de prueba, probablemente debería borrarlo antes de sincronizar y cambiar la estructura de datos en la BD porque puede no sobrescribir el registro existente de manera correcta.

No olvide sincronizar con la base de datos:



Sincronizar con la Base de Datos

Otra nota: si se siente más cómodo usando PhpMyAdmin para realizar estos cambios, está perfectamente bien hacerlo de esa manera, mientras siga la estructura dada.

Una vez que hagamos el cambio, no se moleste en probar el sitio, nada va a funcionar. Vamos a tener que modificar el modelo User antes de probar el sitio nuevamente.



Aviso

Antes de que comencemos, un rápido aviso en caso de que no lo haya notado. Sólo los archivos de las vistas tienen los tags de cierre ?>. No incluya tags de cierre ?> en sus modelos y controladores.

El modelo User

Bien, demos un vistazo serio al modelo user. Cuando configuramos nuestra plantilla avanzada , Yii 2 hizo todo el trabajo por nosotros. La ventaja de ello es que obviamente, no tuvimos que escribir nada de código. La desventaja es que no estamos realmente seguros de cómo trabaja. Y por supuesto si vamos a controlar el acceso de los usuarios, vamos a tener que conocer mucho mejor el modelo user de lo que lo hacemos ahora.

Para mantener la brevedad, no voy a decir mucho sobre el modelo por defecto que obtiene con la plantilla avanzada, ya que tenemos mucho por cubrir. Hay tanto del modelo principal en nuestro modelo revisado, que entenderá la mayoría de él de cualquier manera.

Bien, hagamos esto.

Si alguna vez ha trabajado con código de un PDF o formato de libro digital antes, sabe lo doloroso que puede resultar copiar y pegar correctamente. También, debido a los límites de las líneas, hay feas separaciones de línea que normalmente no estarían allí. Considerando todo eso, proveo de links a gists, que tendrán el código en un formato mucho más agradable que es fácil de copiar y pegar. En la mayoría de los formatos, abre una nueva ventana o puede hacer clic con el botón derecho sobre el link del gist y abrirlo en una nueva ventana, de manera que no pierda su lugar en el libro.

Debido a que estoy transfiriendo código a un gist, podemos terminar con un error de tipoeo. Si eso ocurre, refiérase a la versión del libro, debería ser una fuente autorizada en este punto. De cualquier manera, espero hacer que esta experiencia sea más fácil para usted.

Reemplace su modelo User existente, ubicado dentro de common/models/User.php, con el código del gist o con el del libro.

Gist:

[Modelo User](#)

Del libro:

```
<?php  
namespace common\models;  
  
  
use Yii;  
use yii\base\NotSupportedException;  
use yii\behaviors\TimestampBehavior;  
use yii\db\ActiveRecord;  
use yii\db\Expression;  
use yii\web\IdentityInterface;  
use yii\helpers\Security;  
  
  
/**  
 * User model  
 *  
 * @property integer $id  
 * @property string $username  
 * @property string $password_hash  
 * @property string $password_reset_token  
 * @property string $email  
 * @property string $auth_key  
 * @property integer $rol_id  
 * @property integer $estado_id  
 * @property integer $tipo_usuario_id  
 * @property integer $created_at  
 * @property integer $updated_at  
 * @property string $password write-only password  
 */  
  
class User extends ActiveRecord implements IdentityInterface  
  
{  
  
const ESTADO_ACTIVO = 1;
```

```
public static function tableName()

{
    return 'user';
}

/***
 * behaviors
 */
public function behaviors()
{
    return [
        'timestamp' => [
            'class' => 'yii\behaviors\TimestampBehavior',
            'attributes' => [
                ActiveRecord::EVENT_BEFORE_INSERT => ['created_at', 'updated_at'],
                ActiveRecord::EVENT_BEFORE_UPDATE => ['updated_at'],
            ],
            'value' => new Expression('NOW()'),
        ],
    ];
}

/***
 * reglas de validación
 */
public function rules()
{
    return [

```

```
[ 'estado_id', 'default', 'value' => self::ESTADO_ACTIVO],  
  
[ 'rol_id', 'default', 'value' => 1],  
  
[ 'tipo_usuario_id', 'default', 'value' => 1],  
  
[ 'username', 'filter', 'filter' => 'trim'],  
[ 'username', 'required'],  
[ 'username', 'unique'],  
[ 'username', 'string', 'min' => 2, 'max' => 255],  
  
[ 'email', 'filter', 'filter' => 'trim'],  
[ 'email', 'required'],  
[ 'email', 'email'],  
[ 'email', 'unique'],  
  
];  
}  
  
/* Las etiquetas de los atributos de su modelo */  
  
public function attributeLabels()  
{  
    return [  
        /* Sus otras etiquetas de atributo */  
    ];  
}  
  
/**  
 * @findIdentity  
 */  
  
public static function findIdentity($id)
```

```
{  
    return static::findOne(['id' => $id, 'estado_id' => self::ESTADO_ACTIVO]);  
}  
  
/**  
 * @inheritDoc  
 */  
  
public static function findIdentityByAccessToken($token, $type = null)  
{  
  
    throw new NotSupportedException  
    ('"findIdentityByAccessToken" is not implemented.');  
}  
  
/**  
 * Encuentra usuario por username  
 * dividida en dos líneas para evitar ajuste de línea * @param string $username  
 * @return static|null  
 */  
  
public static function findByUsername($username)  
{  
  
    return static::findOne(['username' => $username, 'estado_id' =>  
        self::ESTADO_ACTIVO]);  
}  
  
/**  
 * Encuentra usuario por clave de restablecimiento de password  
 */
```

```
* @param string $token clave de restablecimiento de password
* @return static|null
*/
public static function findByPasswordResetToken($token)
{
    if (!static::isPasswordResetTokenValid($token)) {
        return null;
    }

    return static::findOne([
        'password_reset_token' => $token,
        'estado_id' => self::ESTADO_ACTIVO,
    ]);
}

/**
 * Determina si la clave de restablecimiento de password es válida
 *
 * @param string $token clave de restablecimiento de password
 * @return boolean
 */
public static function isPasswordResetTokenValid($token)
{
    if (empty($token)) {
        return false;
    }
    $expire = Yii::$app->params['user.passwordResetTokenExpire'];
    $parts = explode('_', $token);
    $timestamp = (int) end($parts);
    return $timestamp + $expire >= time();
}

/**
 * @getId
 */

```

```
public function getId()
{
    return $this->getPrimaryKey();
}

/**
 * @getAuthKey
 */

public function getAuthKey()
{
    return $this->auth_key;
}

/**
 * @validateAuthKey
 */

public function validateAuthKey($authKey)
{
    return $this->getAuthKey() === $authKey;
}

/**
 * Valida password
 *
 * @param string $password password a validar
 * @return boolean si la password provista es válida para el usuario actual
 */

public function validatePassword($password)
{
    return Yii::$app->security->validatePassword($password, $this->password_hash);
}
```

```
/*
 * Genera hash de password a partir de password y la establece en el modelo
 *
 * @param string $password
 */

public function setPassword($password)
{
    $this->password_hash = Yii::$app->security->generatePasswordHash($password);
}

/*
 * Genera clave de autenticación "recuerdame"
 */

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

/*
 * Genera nueva clave de restablecimiento de password
 * dividida en dos líneas para evitar ajuste de línea
 */
public function generatePasswordResetToken()
{
    $this->password_reset_token = Yii::$app->security->generateRandomString()

        . '_' . time();

}

/*
 * Remueve clave de restablecimiento de password

```

```
* /  
  
public function removePasswordResetToken()  
{  
    $this->password_reset_token = null;  
}  
  
}
```



Consejo

Si va a usar el código del libro y no del Gist, el código de User.php no está formateado exactamente cómo sería deseable en su archivo. Hay unas pocas instancias de dos líneas de código donde debería haber una. La razón es que PDF y otros formatos dividen la línea con un ajuste de línea e insertan caracteres especiales que arruinan el código, así que tuve que formatear el código proactivamente para que la línea no se quiebre. No siempre se ve bonito, pero al menos el código funcionará. Debería encontrar estas instancias y convertirlas a una sola línea removiendo el espacio en blanco. La función pública estática `findByUsername($username)` y función pública `generatePasswordResetToken()` tienen la línea extra en el cuerpo de la función.

Propiedades del Modelo

Así que, ¿dónde están las propiedades de la clase que representa el modelo? Puede verla en los comentarios, pero no están listadas en la clase. ¿Por qué ocurre esto? Resulta que Yii 2, a través de su magia interna, conoce las propiedades del modelo por los nombres de columna de las tablas, así que no tiene que declararlas. ¿Qué tan genial es eso? Ciertamente hace que sea difícil olvidarse de incluirlas.

Esto aplica a los modelos que extienden de ActiveRecord. Los modelos de formulario, que exploraremos más adelante, extienden de Model y tienen propiedades que deben ser declaradas, pero no nos preocuparemos sobre eso ahora.

Bien, ahora vayamos al modelo user. No hicimos demasiados cambios, pero obviamente agregamos campos y cambiamos status a estado_id. Aunque añadimos tipo_usuario_id a la tabla user, no vemos mucha evidencia de ello allí, excepto en el método rules. Y aún así, como hemos descrito anteriormente, el modelo conoce sus atributos basado en la estructura de la tabla, así que ya estamos agregando profundidad al modelo user.

Dicho esto, no podemos entender realmente el modelo User sin alguna idea de que como vamos a soportarlo, qué otros modelos vamos a crear. Adelantándonos, estos son los modelos que sabemos vamos a crear:

- Rol
- Estado
- TipoUsuario
- Perfil
- Genero

En el próximo capítulo, vamos a crear las tablas de la base de datos para estos modelos, y luego los propios modelos.

Muchos programadores crean la estructura de la base de datos una tabla a la vez y avanzan según creen necesario. Típicamente usan migraciones para lograr esto. Aparte de la migración inicial que creó el modelo user original, no usaremos migraciones.

Soy un gran creyente de pensar en la estructura de datos y crearla de una vez, en contraste con el enfoque adhoc. Eso no quiere decir que no pueda refinar y realizar cambios a medida que avanza, pero un poco de previsión da una gran ventaja. Por supuesto tiene libertad de usar migraciones si lo desea, especialmente si se siente cómodo con ellas. Vea la Guía de Yii 2 por detalles:

[Migraciones en Yii 2](#)

Constantes

Una cosa que puede llamarle la atención de esa lista de nuevos modelos, especialmente con Rol, Estado, y TipoUsuario, es que éstas estructuras de datos podrían haber sido manejadas alternativamente por constantes. Mientras que ello sería probablemente más fácil de implementar, me inclino por poner cosas como valores de estado en la BD. La razón para hacer esto es que puedo crear un IU de Admin que me permita actualizar y crear nuevos valores, sin tener que ir al código.

Tome Rol por ejemplo. Digamos que tiene un rol llamado admin, que otorga acceso al backend. Tiene un valor de 20. Usted configura su constante como sigue:

```
const ROL_VALOR_ADMIN = 20;
```

Pero luego decide que necesita un rol aún más abarcativo, llamemoslo SuperUser. Necesitaría ir al código, encontrar cada instancia donde usa la constante, crear otra constante, y agregarla a todos los métodos de soporte que poblarán los nombres de Rol para las listas desplegables, etc. Es bastante fácil de hacer, pero en mi opinión, no es la mejor forma.

Prefiero tener una IU en el backend que me permita simplemente agregar un registro a la BD que defina el nuevo rol y le de un valor. Entonces, si he codificado mis métodos correctamente, lo tendré disponible en todas partes. A medida que avancemos en el libro, verá cómo esto entra en juego.

Ahora si revisa bajo nuestra declaración de clase, encontrará que dejamos una constante:

```
const ESTADO_ACTIVO = 1;
```

Dejé la constante allí por una buena razón, aún cuando viola el principio DRY (al menos para lo que vamos a construir), porque el valor de estado activo es vital para el sistema de registro y recuperación de contraseña y aún no tenemos nuestras tablas y modelos auxiliares. Dejo la constante en su lugar, para que podamos tener el sitio funcionando. El sitio necesita este valor para funcionar y es uno de esos casos donde estoy dispuesto a duplicar a cambio de la facilidad de uso. Puede reemplazarlo más adelante con un método si así lo elige. Es un asunto trivial en etapas posteriores realizar los cambios si lo desea.

Interfaz Identidad (Identity Interface)

Regresemos a la declaración de la clase por un momento:

```
class User extends ActiveRecord implements IdentityInterface
{
```

Esta es una estructura de clase de Yii 2 que no escribí, pero no es un problema, aún podemos notar algunas cosas sobre el modelo.

En este caso, extendemos de `ActiveRecord` e implementamos `IdentityInterface`, que significa que tenemos que crear los métodos de la interfaz en nuestra clase `User`. Veremos en la clase y en los comentarios escritos por los desarrolladores de Yii 2 sobre qué métodos debemos usar. Le dará una idea de cómo trabaja, pero no se preocupe si no entiende instantáneamente como escribir los métodos, y verá por qué en un momento.

```
<?php

/**
 * @link http://www.yiiframework.com/
 * @copyright Copyright (c) 2008 Yii Software LLC
 * @license http://www.yiiframework.com/license/
 */

namespace yii\web;

interface IdentityInterface
{



    /**
     * Finds an identity by the given ID.
     * @param string/integer $id the ID to be looked for
     */
}
```

```
* @return IdentityInterface the identity object that
*matches the given ID.
* Null should be returned if such an identity cannot be found
* or the identity is not in an active state
*(disabled, deleted, etc.)
*/
public static function findIdentity($id);
/**
 * Finds an identity by the given token.
 * @param mixed $token the token to be looked for
 * @param mixed $type the type of the token. The value of this parameter depends\
on the implementation.
 * For example, [\yii\filters\auth\HttpBearerAuth] will set this parameter to \
be `yii\filters\auth\HttpBearerAuth`.
 * @return IdentityInterface the identity object that matches the given token.
 * Null should be returned if such an identity cannot be found
 * or the identity is not in an active state (disabled, deleted, etc.)
*/
public static function findIdentityByAccessToken($token, $type = null);
/**
 * Returns an ID that can uniquely identify a user identity.
 * @return string/integer an ID that uniquely identifies a user identity.
*/
public function getId();
/**
 * Returns a key that can be used to check the validity of a given identity ID.
 *
 * The key should be unique for each individual user, and should be persistent
 * so that it can be used to check the validity of the user identity.
 *
 * The space of such keys should be big enough to defeat potential identity atta\
cks.
 *
```

```

* This is required if [[User::enableAutoLogin]] is enabled.
* @return string a key that is used to check the validity of a given identity I\
D.
* @see validateAuthKey()
*/

public function getAuthKey();

/*
 * Validates the given auth key.
 *
 * This is required if [[User::enableAutoLogin]] is enabled.
 * @param string $authKey the given auth key
 * @return boolean whether the given auth key is valid.
 * @see getAuthKey()
 */

public function validateAuthKey($authKey);

}

```

Una interface es como un contrato con la subclase. Dice “si quiere usar mi interface, debe seguir los siguientes métodos”. Si no los incluye todos, retornará un error. Los programadores usan interfaces para controlar la arquitectura.

Así que esta IdentityInterface es el contrato que nuestro modelo User necesita implementar. Bien, así que dije que no necesitamos preocuparnos por la interfaz, ¿por qué? Afortunadamente, la plantilla avanzada ya la implementa por nosotros, y usted ya puede encontrar estos métodos en el modelo User, así que no necesita escribir una sola línea de código. ¡Gracias Yii 2 por la Plantilla Avanzada!

La plantilla básica no viene con esta implementación, lo que es una razón de por qué la plantilla avanzada en realidad es más fácil de implementar que la plantilla básica. Es una de las principales razones por las que elegimos la plantilla avanzada para este libro.

Hemos realizado un pequeño cambio a un número de métodos de la interfaz que fue provista con la plantilla, cambiando el atributo ‘status’ a ‘estado_id’ para reflejar los cambios que realizamos en nuestra estructura de datos. Mencionaré estos cambios a medida que nos movamos por cada uno de los métodos del modelo user.

Así que regresemos a un nuestro modelo User. A medida que recorramos los métodos, también señalaré qué hemos incluido en las sentencias use para soportar los métodos cuando sea necesario.

El primer método que vemos es:

```
public static function tableName()
{
    return 'user';
}
```

Con suerte este método es bastante autoevidente. ¡Desearía que todos fueran tan fáciles!

Comportamientos (Behaviors)

El siguiente método es:

```
public function behaviors()
{
    return [
        'timestamp' => [
            'class' => 'yii\behaviors\TimestampBehavior',
            'attributes' => [
                ActiveRecord::EVENT_BEFORE_INSERT => ['created_at', 'updated_at'],
                ActiveRecord::EVENT_BEFORE_UPDATE => ['updated_at'],
            ],
            'value' => new Expression('NOW()'),
        ],
    ];
}
```

Creo que el concepto de comportamientos en Yii 2 es bastante intuitivo, escrito con sintaxis clara y bello código. El método le indica al modelo como comportarse, dados ciertos eventos.

El primer elemento del arreglo, ‘timestamp’ identifica el comportamiento, y le indicamos que clase usar. Luego definimos los eventos que afectarán a los atributos, en este caso ActiveRecord::EVENT_BEFORE_INSERT y ActiveRecord::EVENT_BEFORE_UPDATE. Estos señalan a los atributos, ‘created_at’ and ‘updated_at’, que también están representados como campos en la tabla user.

Note que también estamos definiendo el valor y usará:

```
'value' => new Expression('NOW()'),
```

Le entrega la cadena ‘NOW()’ a Mysql, que es una sintaxis de Mysql para el DateTime actual, así que la clase de la expresión la formatea por nosotros. Sin ello, insertaría un entero, lo que no es el

comportamiento que deseamos. Así que este método se disparará cada vez que un registro sea creado o actualizado y pondrá la entrada apropiada en la base de datos en el formato DateTime correcto.

El concepto de comportamientos se usa extensamente en Controladores y estaremos viéndolo luego.
Note también que para usar la expresión, tuvimos que incluir la sentencia use apropiada:

```
use yii\db\Expression;
```

Bien, pasemos a nuestro próximo método:

Rules

```
public function rules()
{
    return [
        ['estado_id', 'default', 'value' => self::ESTADO_ACTIVO],
        ['rol_id', 'default', 'value' => 1],
        ['tipo_usuario_id', 'default', 'value' => 1],

        ['username', 'filter', 'filter' => 'trim'],
        ['username', 'required'],
        ['username', 'unique'],
        ['username', 'string', 'min' => 2, 'max' => 255],

        ['email', 'filter', 'filter' => 'trim'],
        ['email', 'required'],
        ['email', 'email'],
        ['email', 'unique'],
    ];
}
```

Así de fácil nos lo pone Yii 2 para imponer el uso reglas de validación en el modelo. Es un formato de arreglo, donde el primer valor es el atributo, el segundo el validador que es llamado, y luego vienen los parámetros y condiciones. Puede chequear la guía para una lista más completa de validadores y cómo usarlos:

[Guía de Yii 2 sobre Reglas](#)

Las primeras tres reglas tratan con el establecimiento de valores por defecto. Puse espacio en blanco entre grupos por razones cosméticas para que sea más fácil trabajar con reglas para un atributo en particular, pero el orden en que están no es relevante.

Si miramos el último conjunto de reglas, las que tratan con email, veremos que se aseguran de remover espacios en blanco, de que el email sea obligatorio, sea de tipo email, y que sea único. Yii 2 hace todo esto por usted con esta sintaxis sencilla, ¿qué tan asombroso es eso?

Métodos de Identidad (Identity Methods)

El siguiente método del modelo User es `findIdentity`, que es una implementación de `IdentityInterface`, que cubrimos previamente.

```
public static function findIdentity($id)
{
    return static::findOne(['id' => $id, 'estado_id' => self::ESTADO_ACTIVO]);
}
```

Este es uno de los lugares donde cambiamos ‘status’ a ‘estado_id’. Este es seguido por:

```
public static function findIdentityByAccessToken($token, $type = null)
{
    throw new NotSupportedException
    ("findIdentityByAccessToken" is not implemented.);

}
```

También creado por nosotros por la plantilla avanzada:

```
/**
 * Encuentra usuario por username
 * dividida en dos líneas para evitar ajuste de línea * @param string $username
 * @return static|null
 */

public static function findByUsername($username)
{
    return static::findOne(['username' => $username, 'estado_id' =>
        self::ESTADO_ACTIVO]);
}
```

Aquí nuevamente usamos el atributo ‘estado_id’ en lugar de ‘status’. Otro método de `IdentityInterface`, con el mismo cambio al atributo ‘estado_id’:

```

public static function findByPasswordResetToken($token)
{
    if (!static::isPasswordResetTokenValid($token)) {
        return null;
    }

    return static::findOne([
        'password_reset_token' => $token,
        'estado_id' => self::ESTADO_ACTIVO,
    ]);
}

```

Este es seguido por un método para verificar si la clave de restablecimiento es válida:

```

public static function isPasswordResetTokenValid($token)
{
    if (empty($token)) {
        return false;
    }
    $expire = Yii::$app->params['user.passwordResetTokenExpire'];
    $parts = explode('_', $token);
    $timestamp = (int) end($parts);
    return $timestamp + $expire >= time();
}

```

Y otro método más de la interfaz:

```

public function getId()
{
    return $this->getPrimaryKey();
}

```

Y los dos últimos métodos de la interfaz provista por la plantilla avanzada:

```

public function getAuthKey()
{
    return $this->auth_key;
}

public function validateAuthKey($authKey)
{
    return $this->getAuthKey() === $authKey;
}

```

Ya que la plantilla avanzada nos provee de los métodos de la interfaz, no los cubriremos en gran detalle. Si desea leer más sobre ellos, puede chequear:

[Yii 2 Autenticación de Seguridad](#)

Métodos Predefinidos

Los siguientes métodos son todos parte del código predefinido, que no cambiamos. Nuevamente, los comentarios proveen una mejor explicación de la que yo podría dar, ya que son métodos profundos del framework que no he escrito:

```

/**
 * Valida password
 *
 * @param string $password password a validar
 * @return boolean si la password provista es válida para el usuario actual
 */

public function validatePassword($password)
{
    return Yii::$app->security->validatePassword($password, $this->password_hash);
}

/**
 * Genera hash de password a partir de password y la establece en el modelo
 *
 * @param string $password
 */

```

```
public function setPassword($password)
{
    $this->password_hash = Yii::$app->security->generatePasswordHash($password);
}

/**
 * Genera clave de autenticación "recuerdame"
 */

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

/**
 * Genera nueva clave de restablecimiento de password
 * dividida en dos líneas para evitar ajuste de línea
 */
public function generatePasswordResetToken()
{
    $this->password_reset_token = Yii::$app->security->generateRandomString()
        . '_' . time();
}

/**
 * Remueve clave de restablecimiento de password
 */

public function removePasswordResetToken()
{
    $this->password_reset_token = null;
}
```

Si todo fue bien con la modificación de la tabla user y la copia del nuevo modelo User, debería ser capaz de usar nuevamente la aplicación para registrar un usuario. Si por alguna razón no funciona,

revise sus pasos y chequé su ortografía cuidadosamente. Asegúrese de que la BD está actualizada con los campos correctos.

→ Nota: Ya que cambiamos nuestro campo a estado_id, la funcionalidad de recuperación de contraseña predefinida ahora está rota. No se preocupe, lo arreglaremos luego.

Otros modelos que acceden a User

Antes de finalizar nuestro capítulo sobre el modelo User, debemos discutir el hecho de que un controlador no siempre accede al modelo User de la misma manera. Hay diferentes modelos que un controlador puede usar para actualizar la tabla user en momentos diferentes. Por ejemplo, en nuestra aplicación, si vamos a crear un usuario desde el formulario de registro del sitio, el controlador usará el modelo `SignupForm` ubicado en `frontend/models/SignupForm.php`, que es provisto por Yii 2 como parte de la aplicación avanzada.

Eso puede sonar confuso en un primer momento, pero tiene mucho sentido. En las arquitecturas `MVC` avanzadas, los formularios `típicamente tienen modelos que gobiernan su comportamiento`. El modelo de formulario trabaja `en sintonía con el controlador` para proveer toda la lógica necesaria para `validar y procesar el formulario`.

Modelo SignupForm

Demos un vistazo el modelo SingupForm, ubicado en `frontend/models/SignupForm.php`. Verá que hay sólo 3 atributos:

```
class SignupForm extends Model
{
    public $username;
    public $email;
    public $password;
```

La razón por la que no hay atributos o reglas para `rol_id`, `estado_id`, `tipo_usuario_id`, por ejemplo, es que estamos estableciéndolos `por defecto en segundo plano`, no desde el formulario, así que no son necesarios. Recuerde, que establecimos el valor por defecto de `rol_id` a 1, y automáticamente es grabado en esa forma cuando se crea un registro de usuario.

Usualmente, los datos de usuario son entregados a un modelo de formulario para imponer reglas de validación u otros métodos. Los datos vienen desde un controlador, que le entrega al modelo los datos enviados desde una vista que es típicamente un formulario. Esto suena más complicado de lo que realmente es.

Es importante saber cómo se crea un usuario en su aplicación, así que veamos cómo funciona mirando el método `actionSignup()` en `frontend/controllers/SiteController`:

```

public function actionSignup()
{
    $model = new SignupForm();
    if ($model->load(Yii::$app->request->post())) {
        if ($user = $model->signup()) {
            if (Yii::$app->getUser()->login($user)) {
                return $this->goHome();
            }
        }
    }

    return $this->render('signup', [
        'model' => $model,
    ]);
}

```

Puede ver que el método llama a una instancia del modelo SignupForm. El método principal es signup(), que crea una instancia del modelo User si el formulario ha pasado la validación:

```

public function signup()
{
    if ($this->validate()) {
        $user = new User();
        $user->username = $this->username;
        $user->email = $this->email;
        $user->setPassword($this->password);
        $user->generateAuthKey();
        $user->save();
        return $user;
    }

    return null;
}

```

Tratará de validar, y si puede hacerlo, llamará a una instancia de la clase user, para establecer las propiedades del usuario a los valores que fueron entregados via formulario, crear la contraseña y aplicarle el hash, generar la clave de autorización, guardar y retornar \$user. Es importante notar que una sentencia return, cuando es ejecutada, finaliza la función, así que no necesita una sentencia else aquí. Si hay un \$user, es devuelto y el código nunca ejecuta return null. Si la sentencia if evalúa a falso, retornará nulo. Será falso si la validación falla o existió algún otro problema.

Bien, de regreso a la acción en SiteController, donde tenemos una bella sentencia if anidada, que dividiremos para comprenderla:

```
if ($model->load(Yii::$app->request->post())) {
```

Si el modelo (SignupForm) puede cargar los datos enviados desde Yii::\$app->request->post(), lo que sólo ocurre si hay datos enviados vía post. La sintaxis para obtener datos enviados por post es clara y precisa:

```
Yii::$app->request->post()
```

Esto trae todos los atributos del formulario siempre y cuando el formulario y el modelo de formulario estén construidos de forma correcta. Los datos del post solamente pueden venir de alguien que haya completado el formulario en la vista y son pasados por la acción de la vista. Si eso ocurre, entonces continúa. En este caso la vista es signup.php bajo frontend/views/site/signup.php. No entraremos en detalle sobre el formulario ahora, pero puede revisarlo por sí mismo si así lo desea.

Siguiente if:

```
if ($user = $model->signup()) {
```

Llama al método signup de SignupForm. Lo primero que hace el método signup es validar, de modo que si no pasamos las reglas, no registrará al usuario y retornará un mensaje de error, basado en el comportamiento de la regla. Si todo está bien obtenemos una instancia de \$user, continúa:

Luego el tercer if:

```
if (Yii::$app->getUser()->login($user)) {
```

Estamos accediendo a getUser y login user desde una instancia de aplicación de Yii, que tiene acceso a esos métodos. Hablamos acerca de crear la instancia de la aplicación a partir de Index.php en el capítulo 3, que aquí está siendo utilizada para llamar un par de métodos en cadena.



Aviso

Note, que para ser capaces de usar Yii::\$app, necesitamos tener la sentencia use Yii; en la parte superior del archivo.

Si podemos encontrar al usuario e iniciar sesión, entonces:

```
return $this->goHome();
```

Esto simplemente lo lleva nuevamente a la vista Index.php, pero en un estado de sesión iniciada, de lo contrario, muestra el formulario de registro:

```
return $this->render('signup', [
    'model' => $model,
]);
```

Y con toda la validación y métodos internos de Yii 2, si intentó registrarse y algo salió mal, también mostrará los mensajes de error.

El método login de SiteController es similar:

```
public function actionLogin()
{
    if (!\Yii::$app->user->isGuest) {
        return $this->goHome();
    }

    $model = new LoginForm();
    if ($model->load(Yii::$app->request->post()) && $model->login()) {
        return $this->goBack();
    } else {
        return $this->render('login', [
            'model' => $model,
        ]);
    }
}
```

Primero comprueba si ha iniciado sesión o no llamando al método isGuest. Usamos ! delante, de modo que si no es un invitado, ya ha iniciado sesión y se dirige a la página principal.

Luego usa un modelo diferente, el modelo LoginForm y o bien inicia sesión y lo lleva a la página donde se encontraba previamente, pero un estado de sesión iniciada, o le muestra el formulario de inicio de sesión nuevamente, con errores si intentó ingresar y lo hizo de forma incorrecta.

Bien, así que tomamos un pequeño desvío del modelo User para darle una idea de cómo los usuarios son creados y darle un vistazo de los modelos que mueven datos de usuario a través del sitio. Realmente no profundizamos mucho en el controlador, cubriremos los controladores en más detalle luego, esto fue más sobre los modelos que controlan al usuario. Aquí vimos 3 modelos diferentes, User, SignupForm, y LoginForm que controlan los datos del usuario.

Resumen



¡No olvide realizar un Commit!

Bien, eso fue mucho para absorber. Si todo esto es nuevo para usted y está luchando un poco, no se preocupe, se volverá más claro con el tiempo a medida que se habitúe a ver los mismos tipos de métodos usados para mover datos alrededor del sitio. Veremos todo esto en detalle más adelante.

Así que estamos creando una plantilla reutilizable y comenzamos a modificar el modelo User, que tiene muchos métodos en él que penetran en el framework.

El modelo User es siempre drásticamente diferente que otros modelos debido a cosas como el método set password y otros métodos que son únicos para los usuarios. También tocamos el hecho de que los controladores pueden en ocasiones usar otros modelos para crear y modificar registros de usuarios.

Los otros modelos que vamos a construir, tales como Rol, Estado, TipoUsuario, etc., tienden a ser más sencillos y fáciles de comprender, sin mencionar, mucho más breves.

En la siguiente sección, usaremos Gii, la herramienta de generación de código de Yii 2, y verá cuán asombroso realmente es y cuánto más rápido es el flujo de trabajo.

Capítulo Cinco: Creando Nuevos Modelos con Gii

Antes de que podamos usar Gii para crear nuevos modelos, tenemos que crear las tablas primero. Como dijimos en el último capítulo, nuestra meta es crear una estructura de datos que nos permita administrar usuarios y controlar el acceso al sitio.

Los modelos que crearemos son:

- Rol
- Estado
- Genero
- TipoUsuario
- Perfil

Note que en la lista de arriba, ya que hablamos de modelos, usamos mayúsculas, y puede ver que en TipoUsuario, uso el formato que Gii creará siguiendo la convención para el caso donde el nombre de la tabla es tipo_usuario. Lo entenderemos mejor más adelante en el libro cuando creamos el CRUD para TipoUsuario.

Creando Tablas

Ahora es tiempo de que creamos el resto de las tablas. Voy a proveer capturas de pantalla de Mysql workbench, que nos dará una referencia fácil no sólo sobre qué campos necesitamos, sino también de las restricciones y tipos de datos.



Aviso

LAS RESTRICCIONES MySQL se usan para definir reglas que permiten o restringen los valores que pueden guardarse en las columnas.

LAS RESTRICCIONES MySQL imponen la integridad de la base de datos.

LAS RESTRICCIONES MySQL se declaran en el momento de creación de la tabla.

LAS RESTRICCIONES MySQL son:

- NOT NULL

- UNIQUE
- PRIMARY KEY
- FOREIGN KEY
- CHECK
- DEFAULT

Para un tutorial de referencia de Mysql, revise [W3Resource](#).

Tabla Rol

Aquí está la tabla para rol:

The screenshot shows the MySQL Workbench interface with the 'rol - Table' window open. The table structure is defined as follows:

Column Name	Datatype	PK	NN	UQ	BIN	UN	ZF	AI	Default
<code>id</code>	<code>SMALLINT(6)</code>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>					
<code>rol_nombre</code>	<code>VARCHAR(45)</code>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>					
<code>rol_valor</code>	<code>INT(11)</code>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>					

tabla rol

Note que hemos usado minúsculas para el nombre de la tabla. Si un nombre de tabla requiere dos palabras, las separaremos con un guión bajo. También usaremos un guión bajo para separar palabras en nombres de columnas como puede ver arriba.

La tabla rol es muy simple. Pk significa clave primaria (primary key), NN significa No Nulo (Not Null), y AI es autoincremento (auto-increment). Autoincrementamos las ids de los registros. Usamos varchar para rol_nombre e integer para rol_valor. Probablemente puede usar small int para rol_valor, le dejaré a usted esa opción.

A veces cuando esté construyendo aún estructuras de datos triviales, deseará los campos created_at and updated_at, además de created_by y updated_by, simplemente para mantener la pista de quién está haciendo qué y cuándo. Pero ya que ésta sólo contiene los nombres y valores de los roles, no necesitamos esos campos.

Tabla Estado

Bien, continuemos y ahora construyamos una para el estado:

Column Name	Datatype	PK	NN	UQ	BIN	UN	ZF	AI	Default
id	SMALLINT(6)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>					
estado_nombre	VARCHAR(45)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>					
estado_valor	SMALLINT(6)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>					

Tabla Estado

Esta es idéntica a rol, sólo que es para estado. En las dos tablas que hemos creado hasta ahora, hemos seleccionado PK para la clave principal en la primera columna, que es id. También seleccionamos NN, que es no nulo, lo que significa que no se le permite ser nula.

Tabla Tipo Usuario

Ahora construyamos la tabla tipo_usuario:

Column Name	Datatype	PK	NN	UQ	BIN	UN	ZF	AI	Default
id	SMALLINT(6)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>					
tipo_usuario_nombre	VARCHAR(45)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>					
tipo_usuario_valor	INT(11)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>					

Tabla Tipo Usuario

Tiene la misma estructura de datos que las dos primeras tablas que creamos, sólo que tenemos un nombre de tabla con un guión bajo en él. Gii crea una convención de nombres específica para manejar estos casos, que veremos más adelante cuando creemos el modelo, controlador, y vistas.

Tabla Genero

Aquí tenemos la tabla genero:

Column Name	Datatype	PK	NN	UQ	BIN	UN	ZF	AI	Default
id	SMALLINT(6)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
genero_nombre	VARCHAR(45)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Tabla Genero

Esta es aún más simple, sólo id y genero_nombre.

Tabla Perfil

Y finalmente, la tabla perfil:

The screenshot shows the Yii2 Gii Model Generator interface. On the left is a toolbar with icons for creating, updating, deleting, and managing models. In the center, a modal window titled 'perfil' displays the table structure:

```

    id INT(11)
    user_id INT(11)
    nombre TEXT(60)
    apellido TEXT(60)
    fecha_nacimiento DATETIME
    genero_id SMALLINT(6)
    created_at DATETIME
    updated_at DATETIME
  
```

Below the modal is a table named 'perfil - Table' with the following schema:

Column Name	Datatype	PK	NN	UQ	BIN	UN	ZF	AI	Default
id	INT(11)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
user_id	INT(11)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
nombre	TEXT(60)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
apellido	TEXT(60)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
fecha_nacimiento	DATETIME	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
genero_id	SMALLINT(6)	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
created_at	DATETIME	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
updated_at	DATETIME	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	

On the right, there are fields for 'Table Name' (empty), 'Schema' (set to 'yii2build'), 'Collation' (set to 'utf8mb4_unicode_ci'), and 'Comments' (empty). Below the table is the title 'Tala Perfil'.

Nuestro plan es permitirle a cada usuario crear un único perfil, de modo que este tendrá una relación uno a uno con el modelo User. Así que agreguemos el siguiente método al modelo User en la parte inferior de la clase.

Gist:

[Get Perfil](#)

Del libro:

```

public function getPerfil()
{
    return $this->hasOne(Perfil::className(), ['user_id' => 'id']);
}
  
```

Asegúrese de que ha actualizado el modelo User con la función de arriba. Estamos saltando un poco de un lugar a otro, pero eso no puede evitarse.

Puede ver que en este caso la id de user es establecida como user_id en el registro perfil. Y esto establece la relación entre los dos modelos.

Haremos lo mismo para nuestros otros modelos en unos pocos minutos, luego de que hayamos configurado los nuevos modelos. Entonces podremos actualizar el modelo User con todas los otros métodos de relación que necesita para poder hablar con los otros modelos.

Bien, de regreso a la tabla perfil.

Note que en las columnas int de perfil, he desmarcado UN, que significa sin signo (unsigned) y no permite números negativos.

También puede ver un diamante rojo en la columna genero_id que representa una clave foránea. Las claves foráneas se establecen para relacionar dos tablas y Gii puede leer estos datos y configurar la relación por usted cuando crea el modelo. Más adelante veremos esto en acción.

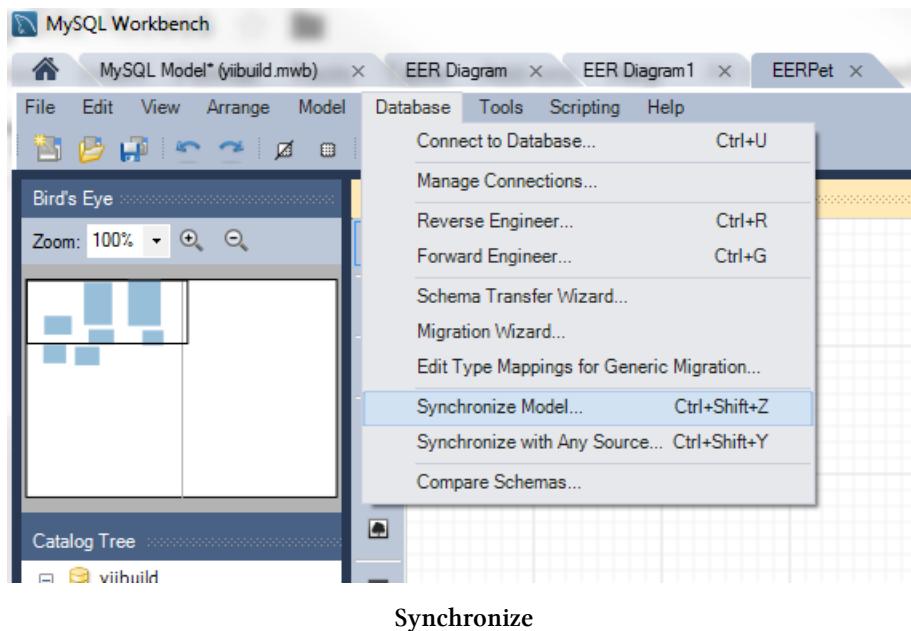
Si no está familiarizado con las claves foráneas en MySql y MySql Workbench, debería tomarse algún tiempo para buscar en google y aprender sobre ellas, son una parte importante de la estructura de la base de datos. No es necesario usarlas para seguir este libro, pero sí necesita asegurarse de que todas las relaciones estén definidas en los casos dónde Gii normalmente las hubiera creado por usted debido a una clave foránea. No es un problema, simplemente ponga mucho cuidado a las relaciones necesarias cuando las veamos más tarde.

Por ahora todo lo que necesita saber, es que la clave foránea para genero_id en la tabla perfil se mapea a id en la tabla genero.

Nota: si tiene problemas estableciendo la clave foránea, asegúrese de que los tipos de datos son exactamente los mismos. Refiérase a las capturas de pantalla de arriba como referencia.

Sincronice

No se olvide de sincronizar el modelo con la base de datos real:



Synchronize

Asegúrese de revisar PhpMyadmin para confirmar que la sincronización fue exitosa:

The screenshot shows the PhpMyAdmin interface. The left sidebar lists databases: "Nueva", "cdcol", "information_schema", "laravel-angular-comment-db", "mysql", "performance_schema", "phpmyadmin", "restaurant", and "test". The main area shows the structure of the "yiibuild" database. It has a table named "user" with one row. Other tables listed are "estado", "genero", "migration", "perfil", "rol", and "tipo_usuario". Each table has columns like "Acción", "Filas", "Tipo", "Cotejamiento", and "Tamaño".

Tabla	Acción	Filas	Tipo	Cotejamiento	Tamaño
estado	Examinar Estructura Buscar Insertar Vaciar Eliminar	0	InnoDB	utf8_unicode_ci	16 K
genero	Examinar Estructura Buscar Insertar Vaciar Eliminar	0	InnoDB	utf8_unicode_ci	16 K
migration	Examinar Estructura Buscar Insertar Vaciar Eliminar	2	InnoDB	utf8_unicode_ci	16 K
perfil	Examinar Estructura Buscar Insertar Vaciar Eliminar	0	InnoDB	utf8_unicode_ci	32 K
rol	Examinar Estructura Buscar Insertar Vaciar Eliminar	0	InnoDB	utf8_unicode_ci	16 K
tipo_usuario	Examinar Estructura Buscar Insertar Vaciar Eliminar	0	InnoDB	utf8_unicode_ci	16 K
user	Examinar Estructura Buscar Insertar Vaciar Eliminar	1	InnoDB	utf8_unicode_ci	16 K

PhpMyAdmin

Y eso es todo. Considerando el conjunto, es una estructura de datos muy simple y vamos a divertirnos mucho con ella. Vamos a usar Gii para crear modelos, controladores, y vistas, mucho código que generará por nosotros.

Configurando Gii

Por supuesto necesitamos asegurarnos de que tenemos instalado Gii. Vaya a la siguiente url en su navegador:

`yii2build.com/index.php?r=gii`

Si no resuelve, entonces necesita revisar su archivo Composer.Json para ver si tiene el módulo Gii requerido. composer.json se encuentra en su directorio raíz y debería ser visible en su IDE.

Nuevamente como ocurre con bloques grandes de código, para su conveniencia:

Gist:

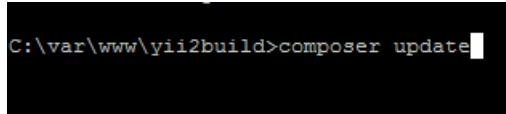
[composer.json](#)

Así es como se ve mi archivo:

```
{  
    "name": "yiisoft/yii2-app-advanced",  
    "description": "Yii 2 Advanced Application Template",  
    "keywords": ["yii2", "framework", "advanced", "application template"],  
    "homepage": "http://www.yiiframework.com/",  
    "type": "project",  
    "license": "BSD-3-Clause",  
    "support": {  
        "issues": "https://github.com/yiisoft/yii2/issues?state=open",  
        "forum": "http://www.yiiframework.com/forum/",  
        "wiki": "http://www.yiiframework.com/wiki/",  
        "irc": "irc://irc.freenode.net/yii",  
        "source": "https://github.com/yiisoft/yii2"  
    },  
    "minimum-stability": "stable",  
    "require": {  
        "php": ">=5.4.0",  
        "yiisoft/yii2": "*",  
        "yiisoft/yii2-bootstrap": "*",  
        "yiisoft/yii2-swiftmailer": "*",  
        "kartik-v/yii2-social": "dev-master",  
        "fortawesome/fontawesome": "4.2.0"  
    },  
    "require-dev": {  
        "yiisoft/yii2-codeception": "*",  
        "yiisoft/yii2-debug": "*",  
        "yiisoft/yii2-gii": "*",  
        "yiisoft/yii2-faker": "*",  
        "yiisoft/yii2-jui": "*"  
    },  
    "config": {  
        "process-timeout": 1800  
    },  
    "extra": {  
        "asset-installer-paths": {  
            "npm-asset-library": "vendor/npm",  
            "bower-asset-library": "vendor/bower"  
        }  
    }  
}
```

```
}
```

Puede ver que bajo “require-dev”, tengo la línea para gii. Tengo unas pocas extensiones incluidas para ser usadas más tarde, incluyendo Kartik social, font-awesome y otros. Tiene sentido simplemente copiar esta versión de composer.json en su archivo, así que hágalo, luego ejecute composer update desde la línea de comandos:



Composer Update

Ahora si regresa a su url:

yii2build.com/index.php?r=gii

Debería ir a Gii.

Tenga en cuenta que si no está ejecutando desde localhost, deberá ajustar su configuración. Por favor consulte la guía para obtener detalles si es necesario:

Configuración de Gii no en LocalHost

Si tenía Gii desde un principio, asegúrese de ejecutar composer update de cualquier manera, para poder obtener las extensiones que necesitaremos luego. Aquí hay algo sobre lo que debemos ser precavidos. Si se saltea pasos, como composer update con el nuevo archivo composer.json, causará problemas más adelante cuando se asuma que tiene esas extensiones. Por favor sea cuidadoso en seguir las instrucciones exactamente, obtendrá mejores resultados.

De cualquier forma, si puede ver Gii, felicitaciones y tómese un respiro, está a punto de divertirse.

Construyendo Modelos con Gii

Una cosa que debería mencionar antes de comenzar. Vamos a construir cinco modelos nuevos y continuar actualizando el modelo User con las relaciones. Esta es mucha información, así que no sienta que tiene que memorizarla o comprender instantáneamente cada detalle. Todo tendrá sentido a medida que avancemos y usted vea cómo utilizar los modelos y cómo ensamblamos todo.

También tenga en cuenta, que proveeré de archivos completos para los dos modelos mayores cerca del final del capítulo, de manera que aún si se pierde algo, tendrá los modelos completos con los cuáles comparar. Dicho eso, ¡iniciemos Gii!

Apunte su navegador a:

backend.yii2build.com/index.php?r=gii

Crear el Modelo Rol

Y construyamos nuestro primer modelo. Comenzaremos con el modelo Rol. Una decisión que tenemos que tomar desde ahora es dónde ubicar el modelo. Las decisiones lógicas son los directorios frontend, common, y backend.

La guía para Yii 2 recomienda usar el directorio common para los modelos, y luego extenderlos a diferentes modelos para frontend y backend si fuese necesario. Yo realmente prefiero usar common para otros tipos de clases y modelos. Así que eso significa ponerlos ya sea en frontend o backend. Con el uso de espacios de nombres, tiene mucha libertad para estructurarlo como lo deseé.

En cuanto a si debería ir en el directorio frontend, common, o backend, voy a ponerlo en el directorio backend. Podría fácilmente estar en alguno de los otros directorios, pero esta decisión me parece intuitiva, es lo que esperaría al buscarlo, así que voy a ponerlo en el backend.

Aquí está una captura de pantalla para la tabla rol:

Model Generator

This generator generates an ActiveRecord class for the specified database table.

Table Name
rol

Model Class
Rol

Namespace
backend\models

Base Class
yii\db\ActiveRecord

Database Connection ID
db

Use Table Prefix

Generate Relations

Generate Labels from DB Comments

Gii con tabla Rol

Preste mucha atención al campo Namespace. Puede ver que tenemos el namespace backend\models, así que Rol.php residirá en ese directorio y se agregarán ese namespace al archivo. Luego cada vez que queramos usarlo, simplemente tendremos que incluir la sentencia use:

```
use backend\models\Rol;
```

Haga clic en el botón Preview. Generará de forma automática el archivo. Puede revisarlo al hacer clic sobre el nombre del archivo. Para realmente generar el código, de un clic sobre el botón Generate. Ahora revise backend/models y verá Rol.php, perfectamente formateado para nosotros:

```
<?php

namespace backend\models;

use Yii;

/**
 * This is the model class for table "rol".
 *
 * @property integer $id
 * @property string $rol_nombre
 * @property integer $rol_valor
 */
class Rol extends \yii\db\ActiveRecord
{
    /**
     * @inheritdoc
     */
    public static function tableName()
    {
        return 'rol';
    }

    /**
     * @inheritdoc
     */
    public function rules()
    {
        return [
            [['rol_nombre', 'rol_valor'], 'required'],
            [['rol_valor'], 'integer'],
            [['rol_nombre'], 'string', 'max' => 45]
        ];
    }

    /**
     * @inheritdoc
     */
```

```
/*
public function attributeLabels()
{
    return [
        'id' => 'ID',
        'rol_nombre' => 'Rol Nombre',
        'rol_valor' => 'Rol Valor',
    ];
}

}
```

Así que allí está en backend/models. Luce muy familiar en este punto. Ya he explicado en detalle que hacen las reglas en el modelo User, y las reglas funcionan aquí del mismo modo. Este es un modelo tan sencillo, que no tenemos mucho para decir sobre él.

El método attributeLabels simplemente establece a partir de atributos los nombres de etiquetas que serán visibles en la aplicación.

Tomémonos un momento para apreciar lo limpio y sencillo que es todo esto. Puede ver cómo, una vez que se acostumbra al flujo, las cosas avanzan rápidamente.

Bien, continuemos.

Agregar Registros a la Tabla Rol

Ahora que tenemos una tabla rol y un modelo Rol, usemos PhpMyadmin para crear un par de registros de rol con los que podremos jugar. Ayudará a entender cómo se ensambla todo:

Use la pestaña insertar en la tabla rol:

Column	Type	Function	Null	Value
id	smallint(6)			
role_name	varchar(45)			User
role_value	int(11)			10

Column	Type	Function	Null	Value
id	smallint(6)			
role_name	varchar(45)			Admin
role_value	int(11)			20

PhpMyAdmin Insertar Registros en Rol

Puede ver que no necesitamos establecer el campo id, ya que es autoincremental. Estamos creando 2 registros, 1 llamado Usuario, 1 llamado Admin. Usuario tiene un rol_valor de 10 y Admin tiene un rol_valor de 20. Una vez que los registros han sido agregados, debería ver esto en la pestaña examinar de la tabla rol:

+ Opciones				
	← →	id	rol_nombre	rol_valor
<input type="checkbox"/>	Editar Copiar Borrar	1	Usuario	10
<input type="checkbox"/>	Editar Copiar Borrar	2	Admin	20

PhpMyAdmin Registros de Rol

Si recuerda, establecimos rol_id en la tabla user con un valor por defecto de 1 cuando se crea un registro de usuario. Así que tiene sentido que rol_id mapee a id, lo que significa que un nuevo usuario obtiene un rol de usuario cuando es creado. Ahora sólo necesitamos unos cuantos métodos en ambos modelos para enlazar todo.

Agregar la Relación a Rol

En Rol.php, el modelo Rol, en la parte superior, debajo de namespace, agregaremos:

```
use common\models\User;
```

Y al final de la clase agregue:

```
public function getUsers()
{
    return $this->hasMany(User::className(), ['rol_id' => 'id']);
}
```

La sentencia use le da visibilidad al modelo User. El método getUsers es una forma estándar de establecer la relación de rol a usuarios. En este caso, estamos creando una relaciónhasMany porque un solo rol puede tener muchos usuarios. Por ello es que el método es llamado getUsers en lugar del singular getUser. En el arreglo, puede ver:

```
['rol_id' => 'id']
```

El primer campo es el de la relación y el campo al que apunta en el arreglo es del modelo en el que está trabajando actualmente. La sintaxis es muy intuitiva, pero vale la pena decirlo claramente:

```
hasMany(User::className(), ['rol_id' => 'id'])
```

className es un método del modelo relacionado. El modelo relacionado viene primero, seguido del nombre del campo del modelo actual, así que ahora rol_id en la tabla user mapea a id en la tabla rol. ¡Eso es todo!

Actualizar el Modelo User con Rol

Por supuesto, necesitamos el mismo tipo de método en el modelo User. Abramos el modelo User en common/models/User.php y agreguemos la sentencia use:

```
use backend\models\Rol;
```

Ahora agreguemos los siguientes métodos al final de la clase:

Gist:

[Relaciones User Rol](#)

Del libro:

```
/**
 * relación get rol
 *
 */
```

```
public function getRol()
```

```
{
```

```
    return $this->hasOne(Rol::className(), ['id' => 'rol_id']);

}

/**
 * get rol nombre
 *
 */
public function getRolNombre()
{
    return $this->rol ? $this->rol->rol_nombre : '- sin rol -';

}

/**
 * get lista de roles para lista desplegable
 */
public static function getRolLista()
{
    $dropciones = Rol::find()->asArray()->all();

    return ArrayHelper::map($dropciones, 'id', 'rol_nombre');
```

```
}
```

Bien, observémoslos uno por uno. El primero es la relación getRol, que es el otro extremo de la relación getUsers que agregamos a Rol. En este caso, los Usuarios sólo cumplen un rol, así que se trata de una relaciónhasOne y el método tiene el nombre singular getRol en ella. Por lo demás, el formato es exactamente el mismo que el del modelo Rol. Simplemente dice que id en la tabla rol mapea a rol_id en la tabla user. Esto debería ser fácil de comprender.

El segundo método es getRolNombre. Este nos permite devolver el nombre del rol, lo que querremos hacer para nuestra IU del backend. Ponemos un operador ternario para ver si se ha asignado un rol, y si es así, retornar el nombre o la cadena ‘- sin rol -’.

Finalmente, queremos retornar una lista de ids y nombres de roles para usarla en listas desplegables en la IU. Así que creamos el método getRolLista como sigue:

```
public static function getRolLista()
{
    $dropciones = Rol::find()->asArray()->all();
    return ArrayHelper::map($dropciones, 'id', 'rol_nombre');
}
```

Aquí creamos la variable local \$dropciones y le asignamos una instancia de Rol, con todos los registros que son retornados como un arreglo. Observe lo intuitiva que es la sintaxis de Yii2, se lee como una sentencia. Luego usamos el método **ArrayHelper::map** para listar los valores y nombres de rol. Para usar ArrayHelper, necesitamos incluir una sentencia use en la parte superior del archivo en el bloque de sentencias use:

```
use yii\helpers\ArrayHelper;
```

Es un formato muy común que tengamos estos 3 métodos de relación y construiremos métodos similares para los otros modelos. También note, que la guía de Yii 2 hace un gran trabajo en listar los tipos de relaciones, refiérase a ella cuando esté construyendo sus aplicaciones:

Yii Guía BD

También, ahora que tenemos nuestro método getRolLista, podemos usarlo para imponer una regla de validación en el modelo User. Sólo queremos que el modelo acepte valores para rol_id que estén en el rango de valores del campo rol_valor de la tabla rol. Podemos obtenerlos usando lo siguiente:

```
array_keys($this->getRolLista())
```

Si recuerda, en getRolLista, las claves son las ids de los registros. Así que esta sentencia devuelve 1, 2 con base a lo que tenemos en nuestra BD hasta ahora. Si agregamos nuevos valores, estos serán agregados automáticamente a la lista.

Así que para construir una regla a partir de esto, hacemos lo siguiente:

```
[['rol_id'], 'in', 'range'=>array_keys($this->getRolLista())],
```

Nuevamente esta sintaxis es muy intuitiva. Simplemente agregue esto en su método rules en el modelo User y ahora tendrá un rango de valores que deben ser forzosamente entradas en la columna rol_id de la BD. Esta es una técnica muy común y faremos exactamente lo mismo con otros dos atributos en el modelo User, que veremos a la brevedad.

Crear el Modelo Estado

Continuemos con nuestro siguiente modelo, Estado. Vamos a ubicar también al modelo en backend. Así que iniciemos Gii y repitamos los pasos que seguimos para Rol sólo que esta vez usando la tabla estado.

Nuevamente, asegúrese de que su espacio de nombres es ingresado como backend\models. De un clic en Preview, luego en el botón Generate. Si todo ha ido bien, debería tener un archivo Estado.php en backend/models que se ve de esta forma:

```
<?php

namespace backend\models;

use Yii;

/**
 * This is the model class for table "estado".
 *
 * @property integer $id
 * @property string $estado_nombre
 * @property integer $estado_valor
 */
class Estado extends \yii\db\ActiveRecord
{

    /**
     * @inheritdoc
     */
    public static function tableName()
    {
        return 'estado';
    }

    /**
     * @inheritdoc
     */
}
```

```

/*
public function rules()
{
    return [
        [['estado_nombre', 'estado_valor'], 'required'],
        [['estado_valor'], 'integer'],
        [['estado_nombre'], 'string', 'max' => 45]
    ];
}

/**
 * @inheritdoc
 */
public function attributeLabels()
{
    return [
        'id' => 'ID',
        'estado_nombre' => 'Estado Nombre',
        'estado_valor' => 'Estado Valor',
    ];
}
}

```

Y creamos la relación con user. Primero agregue la sentencia use a Estado.php:

```
use common\models\User;
```

Luego agregamos el método de relación.

Gist:

[Relación Estado a User](#)

Del libro:

```

public function getUsers()
{
    return $this->hasMany(User::className(), ['estado_id' => 'id']);
}

```

No daré muchas explicaciones aquí, este es igual que el modelo Rol. E igual que como hicimos con Rol, necesitamos agregar métodos de relación para los modelos Estado y User.

Actualizar el Modelo User con getEstado

Entonces, en User.php, agregue la sentencia use:

```
use backend\models\Estado;
```

Ahora agregue los siguientes 3 métodos:

Gist:

[Relaciones User Estado](#)

Del libro:

```
/**
 * relación get estado
 *
 */
public function getEstado()
{
    return $this->hasOne(Estado::className(), ['id' => 'estado_id']);
}

/**
 * * get estado nombre
 *
 */
public function getEstadoNombre()
{
    return $this->estado ? $this->estado->estado_nombre : '- sin estado -';
}

/**
 * get lista de estados para lista desplegable
 */
public static function getEstadoLista()
{
    $dropoptions = Estado::find()->asArray()->all();
    return ArrayHelper::map($dropoptions, 'id', 'estado_nombre');
}
```

Y ahora agreguemos la regla de validación para el rango, ya que tenemos acceso a getEstadoLista:

```
[['estado_id'], 'in', 'range'=>array_keys($this->getEstadoLista())],
```

El orden en que la pongamos no importa, seguirá la regla, pero para la legibilidad del código, se recomienda que lo haga debajo de la regla para estado de esta forma:

```
[ 'estado_id', 'default', 'value' => self::ESTADO_ACTIVO],
[[ 'estado_id'], 'in', 'range'=>array_keys($this->getEstadoLista())],
```

Agregar registros a la Tabla Estado

Antes de que lo olvidemos, agreguemos un par de registros a la tabla estado a través de PhpMyadmin, así seremos capaces de probar su funcionalidad luego.

Agregue esto usando la pestaña Insertar en la tabla estado:

estado_nombre: Activo

estado_valor: 10

estado_nombre: Pendiente

estado_valor: 5

The screenshot shows the phpMyAdmin interface with the following details:

- Server:** 127.0.0.1
 - Base de datos:** yii2build
 - Tabla:** estado
- Insertar:** Two separate insert operations are shown, each with three rows of data.
- Row 1 (Top):**

Columna	Tipo	Función	Nulo	Valor
id	smallint(6)			
estado_nombre	varchar(45)			Activo
estado_valor	smallint(6)			10
- Row 2 (Bottom):**

Columna	Tipo	Función	Nulo	Valor
id	smallint(6)			
estado_nombre	varchar(45)			Pendiente
estado_valor	smallint(6)			5
- Buttons:** Continuar (Continue) for each insert operation, and Insertar como una nueva fila (Insert as a new row) and y luego (and then) at the bottom.

Insertar en Estado

Al terminar, debería ver lo siguiente bajo la pestaña examinar en la tabla estado:

+ Opciones			id	estado_nombre	estado_valor
		← →	▼		
<input type="checkbox"/>	 Editar	 Copiar	 Borrar	1	Activo
<input type="checkbox"/>	 Editar	 Copiar	 Borrar	2	Pendiente

Registros de Estado

Crear Modelo Tipo Usuario

Bien, ahora podemos continuar con la tabla tipo_usuario. Como veremos en un momento, esta es un poco diferente porque hay dos palabras en el nombre de la tabla separadas por un guión bajo. Gii elegirá el nombre de modelo cómo:

TipoUsuario

Esa es la convención para el nombre de un modelo basado en una tabla con un guión bajo entre dos palabras. Es posible cambiar esto manualmente, pero debería tener una buena razón para hacerlo, y no tenemos tal razón en este caso. Así que continuaremos con el nombre de modelo de Gii.

Ahora asegúrese de que namespace tenga el valor backend\models. Luego realice los pasos de previsualización y generación y deberíamos concluir con TipoUsuario.php en backend\models que se ve de esta manera:

```
<?php

namespace backend\models;

use Yii;

/**
 * This is the model class for table "tipo_usuario".
 *
 * @property integer $id
 * @property string $tipo_usuario_nombre
 * @property integer $tipo_usuario_valor
 */
class TipoUsuario extends \yii\db\ActiveRecord
{

    /**
     * @inheritdoc
     */
    public static function tableName()
}
```

```

{
    return 'tipo_usuario';
}

/**
 * @inheritdoc
 */
public function rules()
{
    return [
        [['tipo_usuario_nombre', 'tipo_usuario_valor'], 'required'],
        [['tipo_usuario_valor'], 'integer'],
        [['tipo_usuario_nombre'], 'string', 'max' => 45]
    ];
}

/**
 * @inheritdoc
 */
public function attributeLabels()
{
    return [
        'id' => 'ID',
        'tipo_usuario_nombre' => 'Tipo Usuario Nombre',
        'tipo_usuario_valor' => 'Tipo Usuario Valor',
    ];
}
}

```

Este es otro modelo sencillo del mismo tipo que los otros dos que hemos creado. Así que continuemos y agreguemos la sentencia use:

```
use common\models\User;
```

Ahora agreguemos el método getUsers para establecer la relación con los usuarios.

Gist:

[TipoUsuario Get Users](#)

Del libro:

```
public function getUsers()
{
    return $this->hasMany(User::className(), ['tipo_usuario_id' => 'id']);
}
```

Actualizar el Modelo User con TipoUsuario

Ahora continuemos con el archivo User.php en common/models para realizar allí los cambios que necesitamos:

Añadimos la sentencia use en la parte superior:

```
use backend\models\TipoUsuario;
```

Y añadimos los siguientes métodos:

Gist:

[Relaciones Usuario a TipoUsuario](#)

Del libro:

```
public function getTipoUsuario()
{
    return $this->hasOne(TipoUsuario::className(), ['id' => 'tipo_usuario_id']);
}

/**
 * get tipo usuario nombre
 *
 */
public function getTipoUsuarioNombre()
{
    return $this->tipoUsuario ? $this->tipoUsuario
        ->tipo_usuario_nombre : '- sin tipo usuario -';
}

/**
 * get lista de tipos de usuario para lista desplegable
 */
public static function getTipoUsuarioLista()
{
    $dropciones = TipoUsuario::find()->asArray()->all();
    return ArrayHelper::map($dropciones, 'id', 'tipo_usuario_nombre');
}

/**
 * get tipo usuario id

```

```

/*
*/
public function getTipoUsuarioId()
{
    return $this->tipoUsuario ? $this->tipoUsuario->id : 'ninguno';
}

```

Bien, agregamos 4 métodos en lugar de 3. Los primeros tres deberían parecer familiares, ya que construimos los mismos tipos en los otros modelos, y podemos agregar rápidamente una regla de validación para el rango en el campo tipo_usuario_id:

```
[['tipo_usuario_id'], 'in', 'range'=>array_keys($this->getTipoUsuarioLista())],
```

Una cosa no tan obvia es la convención de nombre para getTipoUsuarioNombre. El nombre del método parece lógico, ya que siempre comenzamos con una letra minúscula g para get y luego capitalizamos las restantes palabras. Dónde se vuelve un poco engañoso es en:

```
return $this->tipoUsuario ? $this->tipoUsuario->tipo_usuario_nombre : '- sin tipo usuario -';
```

\$this->tipoUsuario usa un método mágico get, donde get está implícito, y en este caso, Yii 2 no desea que usted comience con una letra mayúscula, la convención cambia en este caso especial. Esto puede resultar confuso así que tiene que ser muy cuidadoso en seguir las convenciones de nombres exactamente o las cosas no funcionarán.

El cuarto método getTipoUsuarioId retorna la id del registro del TipoUsuario, lo que necesitaremos para un uso futuro, así que no hay más que discutir hasta ahora.

Agregar Registros a la Tabla tipo_usuario

Insertemos dos registros en la tabla tipo_usuario:

nombre = Gratuito, valor = 10

nombre = Pago, valor = 30

Cuando haya terminado, use el navegador para ver si los registros están allí:

	← ↑ →	▼	id	user_type_name	user_type_value
<input type="checkbox"/>	Edit Copy Delete	1	Free		10
<input type="checkbox"/>	Edit Copy Delete	2	Paid		30

Registros Tipo Usuario

Si tenía alguna duda de cuál era el propósito del modelo TipoUsuario, estos nombres deberían darle una muy buena idea. Esta estructura de datos será de mucha utilidad cuando deseemos restringir partes de la aplicación a usuarios pagos solamente.

Crear el Modelo Genero

Ahora continuaremos con el modelo Genero. Quiero construir este modelo en primer lugar ya que es un modelo más pequeño y se relaciona con perfil, y ya que hemos usado una clave foránea para conectar las tablas perfil y genero, Gii incluso creará el método de relación por nosotros.

Ya que Genero se relaciona estrechamente con Perfil, he decidido ubicar ambos modelos en frontend. Elijo frontend porque me parece mas fácil de recordar de esa manera, ya que perfil y genero involucran elecciones de los usuarios. Es en gran medida una decisión cosmética, ya que no estamos siguiendo la recomendación de Yii 2 de ubicarlo en common.

Una razón por la que no hago eso es que me agrada mantener common relativamente pequeño, y de esta manera poder crear clases auxiliares que son fáciles de encontrar. Me gusta poner mis clases auxiliares en common, simplemente me parece intuitivo hacerlo así. Pero dado que los modelos tienen espacios de nombres, puede ponerlos en cualquier lugar donde los alias de la aplicación puedan encontrarlos, así que no hay diferencia entre elegir frontend, backend, o common.

Lo que sí importa es que ponga el espacio de nombres deseado de manera correcta en Gii cuando crea el modelo, así que para el caso de Genero usamos:

```
frontend\models
```

Si todo ha ido bien, debería ver un Genero.php in frontend/models que luce así:

```
<?php

namespace frontend\models;

use Yii;

/**
 * This is the model class for table "genero".
 *
 * @property integer $id
 * @property string $genero_nombre
 *
 * @property Perfil[] $perfils
 */
class Genero extends \yii\db\ActiveRecord
{

    /**
     * @inheritdoc
     */
    public static function tableName()
```

```

{
    return 'genero';
}

/**
 * @inheritdoc
 */
public function rules()
{
    return [
        ['genero_nombre'], 'required',
        ['genero_nombre'], 'string', 'max' => 45
    ];
}

/**
 * @inheritdoc
 */
public function attributeLabels()
{
    return [
        'id' => 'ID',
        'genero_nombre' => 'Genero Nombre',
    ];
}

/**
 * @return \yii\db\ActiveQuery
 */
public function getPerfiles()
{
    return $this->hasMany(Perfil::className(), ['genero_id' => 'id']);
}
}

```

Puede ver que debido a la clave foránea que agregamos, Gii fue lo suficientemente listo para agregar la relación por nosotros. ¡Cuán genial es eso?

Agregar Registros a la Tabla genero

Esto está bien y completo y no tenemos que hacer nada. Tomémonos un momento y agreguemos dos registros a la tabla genero:

1 = masculino 2 = femenino

Debería verse así:

		+ Opciones			
		← T →	▼	id	genero_nombre
		<input type="checkbox"/> Editar	Copiar	Borrar	1 masculino
		<input type="checkbox"/> Editar	Copiar	Borrar	2 femenino

Registros Genero

No necesitamos actualizar el modelo User porque no llama a este modelo. En vez de ello, se relaciona estrechamente con Perfil.

Hasta ahora, la mayoría de los modelos en los que nos hemos concentrado contienen una estructura de datos que cada usuario que se registra con la aplicación debe tener. Los campos rol_id, estado_id, y tipo_usuario_id son establecidos por defecto cuando un usuario se registra, y los hemos conectado con modelos que tienen una estructura de datos que otorga profundidad al usuario.

Crear Modelo Perfil

Ahora vamos a crear un modelo denominado Perfil. Todos los usuarios serán capaces de crear un perfil, pero un perfil sólo existirá si el usuario lo crea explícitamente. Así que por supuesto el perfil de usuario tiene su propia tabla, que ya hemos creado.

Como un recordatorio, tenemos las siguientes columnas en la tabla perfil:

- id
- user_id
- nombre
- apellido
- fecha_nacimiento
- genero_id
- created_at
- updated_at

Puede ponerse creativo y agregar otros atributos, sólo hemos seleccionado esos con propósitos demostrativos. Simplemente recuerde que si agrega más, debe hacerlo tanto en la estructura de la tabla como en el modelo.

Bien, ya decidimos que Perfil estaría en el frontend, así que asegúrese de que el campo namespace contenga:

```
frontend\models
```

Model Generator

This generator generates an ActiveRecord class for the specified database table.

Table Name

```
perfil
```

Model Class

```
Perfil
```

Namespace

```
frontend\models
```

Base Class

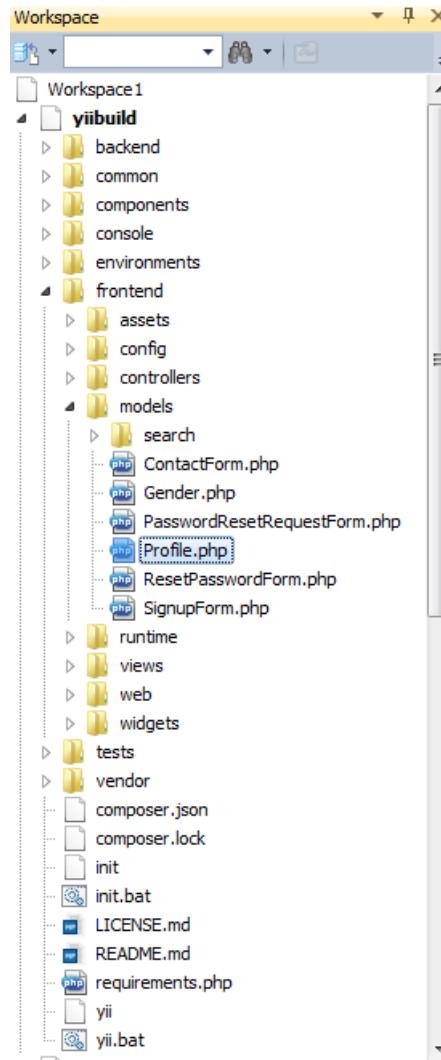
```
yii\db\ActiveRecord
```

Database Connection ID

```
db
```

Modelo Perfil en Gii

Ahora daremos clic a través de los pasos de previsualización y generación y si todo ha ido bien, tendremos Perfil.php en frontend/models. Si todo ha ido bien, podrá ver el archivo:



Perfil en el Directorio Models

Nota: Perfil.php está en el directorio models, no en el directorio search.

Aquí están los contenidos del archivo:

```
<?php

namespace frontend\models;

use Yii;

/**
 * This is the model class for table "perfil".
 *
 * @property string $id
```

```
* @property string $user_id
* @property string $nombre
* @property string $apellido
* @property string $fecha_nacimiento
* @property integer $genero_id
* @property string $created_at
* @property string $updated_at
*
* @property Genero $genero
*/
class Perfil extends \yii\db\ActiveRecord
{

    /**
     * @inheritdoc
     */
    public static function tableName()
    {
        return 'perfil';
    }

    /**
     * @inheritdoc
     */
    public function rules()
    {
        return [
            [['user_id', 'genero_id'], 'required'],
            [['user_id', 'genero_id'], 'integer'],
            [['nombre', 'apellido'], 'string'],
            [['fecha_nacimiento', 'created_at', 'updated_at'], 'safe']
        ];
    }

    /**
     * @inheritdoc
     */
    public function attributeLabels()
    {
        return [
            'id' => 'ID',
            'user_id' => 'User ID',
        ];
    }
}
```

```

        'nombre' => 'Nombre',
        'apellido' => 'Apellido',
        'fecha_nacimiento' => 'Fecha Nacimiento',
        'genero_id' => 'Genero ID',
        'created_at' => 'Created At',
        'updated_at' => 'Updated At',
    ];
}

/**
 * @return \yii\db\ActiveQuery
 */
public function getGenero()
{
    return $this->hasOne(Genero::className(), ['id' => 'genero_id']);
}

}

```

Para este momento este ejemplo de un modelo debería verse familiar, así que no voy a explicarlo nuevamente. Si necesita refrescar su conocimiento, revise los modelos anteriores.

Necesitamos agregar algunas cosas, sin embargo. Comencemos añadiendo las siguientes sentencias use.

Gist:

Sentencias Use de Perfil

Del libro:

```

use yii\db\ActiveRecord;
use common\models\User;
use yii\helpers\Url;
use yii\helpers\Html;
use yii\helpers\ArrayHelper;
use yii\db\Expression;

```

Luego tenemos una adición a nuestras reglas, no se preocupe, añadiremos el método getGeneroLista que estamos llamando aquí en el fondo de la clase:

```
[[ 'genero_id'], 'in', 'range'=>array_keys($this->getGeneroLista())],
```

También, tenemos una regla que agregar para formatear correctamente la fecha_nacimiento:

```
[[ 'fecha_nacimiento'], 'date', 'format'=>'Y-m-d'],
```

Un lector me escribió para decirme que él no podía lograr que lo anterior funcionaría a menos que añadiera el prefijo php:

```
[[ 'fecha_nacimiento' ], 'date', 'format'=>'php:Y-m-d'],
```

No pude reproducir este problema, pero pensé que valía la pena mencionarlo en caso de que se encuentre con un problema similar. Realizaremos algún formateo extra a la fecha en el capítulo 8, así que volveremos al formato de fecha nuevamente en el capítulo.

Descendiendo en la clase, necesitamos agregar algunas etiquetas de atributos para los métodos de relación, para que podamos usarlos más adelante en nuestros widgets a lo largo de la aplicación. No necesitamos ir demasiado lejos en este punto, más que decir que usamos la sintaxis mágica del nombre del método y que es usada por el método `Yii::t`, que es el método de traducción de la aplicación y pondremos la etiqueta a disposición a lo largo de la aplicación:

Gist:

[Agregado de las Etiquetas de Atributo](#)

Del libro:

```
'generoNombre' => Yii::t('app', 'Genero'),
'userLink' => Yii::t('app', 'User'),
'perfilIdLink' => Yii::t('app', 'Perfil'),
```

Ya que nuestro modelo Perfil trata con DATETIME en varios campos, agreguemos el método behaviors que automáticamente inserte nuestra timestamp.

Gist:

[Behaviors de Perfil](#)

Del libro:

```
/**
 * behaviors to control time stamp, don't forget to use statement for expression
 */

```

```
public function behaviors()
{
    return [
        'timestamp' => [
            'class' => 'yii\behaviors\TimestampBehavior',
            'attributes' => [
```

```

ActiveRecord::EVENT_BEFORE_INSERT => ['created_at', 'updated_at'],
ActiveRecord::EVENT_BEFORE_UPDATE => ['updated_at'],
],
'value' => new Expression('NOW()'),
],
];
}

```

Asegurémonos de que tenemos la sentencia use apropiada en la parte superior del archivo:

```
use yii\db\Expression;
```

Para TimestampBehavior, usamos la ruta absoluta:

```
'class' => 'yii\behaviors\TimestampBehavior'
```

Así que no hay necesidad de una sentencia use allí.



Recordatorio

Sólo un recordatorio. El código en behaviors no está formateado exactamente como el que ve en su IDE. La razón es que PDF y otros formatos quiebran la línea con un ajuste de línea e insertan caracteres especiales que arruinan el código, así que tengo que formatearlo proactivamente para que la línea no se rompa. No se ve siempre bonito, pero al menos el código funcionará.

Ahora avancemos a las relaciones. La relación getGenero ya se encuentra allí, generada por Gii.

Puede obtener los otros:

Gist:

[Relaciones de Perfil](#)

Del libro:

```
/**
* @return \yii\db\ActiveQuery
*/
```

```
public function getGeneroNombre()
{
```

```
    return $this->genero->genero_nombre;
}

/**
* get lista de generos para lista desplegable
*/
public static function getGeneroLista()
{
    $droptions = Genero::find()->asArray()->all();
    return ArrayHelper::map($droptions, 'id', 'genero_nombre');
}

/**
* @return \yii\db\ActiveQuery
*/
public function getUser()
{
    return $this->hasOne(User::className(), ['id' => 'user_id']);
}

/**
* @get Username
*/
public function getUsername()
```

```
{  
    return $this->user->username;  
}  
  
/**  
 * @getUserId  
 */  
  
public function getUserId()  
{  
    return $this->user ? $this->user->id : 'none';  
}  
  
/**  
 * @getUserLink  
 */  
  
public function getUserLink()  
{  
    $url = Url::to(['user/view', 'id'=>$this->userId]);  
    $opciones = [];  
    return Html::a($this->getUserName(), $url, $opciones);  
}  
  
/**  
 * @getPerfilLink  
 */
```

```
public function getPerfilIdLink()
{
    $url = Url::to(['perfil/update', 'id'=>$this->id]);
    $opciones = [];
    return Html::a($this->id, $url, $opciones);
}
```

Los únicos métodos de los que no hemos visto un ejemplo antes son los últimos dos. Así que por una cuestión de brevedad, omitiré aquellos que ya comprendemos y me centraré en los nuevos. Sin embargo, sí necesitamos asegurarnos de que hemos incluido la sentencia use ArrayHelper en la parte superior de la clase:

```
use yii\helpers\ArrayHelper;
```

Podemos utilizar el método getGeneroLista para imponer la restricción sobre los valores del modelo tal como lo hicimos en el modelo user:

```
[['genero_id'], 'in', 'range'=>array_keys($this->getGeneroLista())]
```

De esta forma, nadie podrá agregar una id de genero que no sea válida.

Tanto getUserLink como getPerfilIdLink utilizan las clases auxiliares Html y Url, así que necesitamos asegurarnos de que tenemos las siguientes sentencias use:

```
use yii\helpers\Url;
use yii\helpers\Html;
```

Tanto getUserLink como getPerfilIdLink hacen el mismo tipo de cosa. Son métodos que crean links al usuario relacionado y a la id de perfil del usuario. Los usamos en parte la IU más adelante y es una forma elegante de crear links que relacionan modelos. No se preocupe si no lo entiende completamente, lo hará cuando trabajemos en esa parte.

El Modelo Perfil Completo

Debería tener todo lo que necesita para el modelo Perfil, pero sólo para asegurarnos de que tenemos todo, voy a proveerle con el modelo completo como referencia.

Gist:

[Modelo Perfil Completo](#)

Del libro:

```
<?php

namespace frontend\models;

use Yii;
use yii\db\ActiveRecord;
use yii\db\Expression;
use common\models\User;
use yii\helpers\ArrayHelper;
use yii\helpers\Url;
use yii\helpers\Html;

/**
 * This is the model class for table "perfil".
 *
 * @property string $id
 * @property string $user_id
 * @property string $nombre
 * @property string $apellido
 * @property string $fecha_nacimiento
 * @property integer $genero_id
 * @property string $created_at
 * @property string $updated_at
 *
 * @property Genero $genero
 */
class Perfil extends \yii\db\ActiveRecord
{
    /**
     * @inheritdoc
     */
    public static function tableName()
    {
        return 'perfil';
    }
}
```

```
}
```

```
/**
```

```
* behaviors
```

```
*/
```

```
public function behaviors()
{
    return [
        'timestamp' => [
            'class' => 'yii\behaviors\TimestampBehavior',
            'attributes' => [
                ActiveRecord::EVENT_BEFORE_INSERT => ['created_at', 'updated_at'],
                ActiveRecord::EVENT_BEFORE_UPDATE => ['updated_at'],
            ],
            'value' => new Expression('NOW()'),
        ],
    ];
}
```

```
/**
```

```
* @inheritdoc
```

```
*/
```

```
public function rules()
{
    return [
        [['user_id', 'genero_id'], 'required'],
        [['user_id', 'genero_id'], 'integer'],
        [['genero_id'], 'in', 'range'=>array_keys($this->getGeneroLista())],
        [['nombre', 'apellido'], 'string'],
        [['fecha_nacimiento', 'created_at', 'updated_at'], 'safe'],
        [['fecha_nacimiento'], 'date', 'format'=>'Y-m-d']
    ];
}
```

```
/**  
 * @inheritDoc  
 */  
  
public function attributeLabels()  
{  
    return [  
        'id' => 'ID',  
        'user_id' => 'User ID',  
        'nombre' => 'Nombre',  
        'apellido' => 'Apellido',  
        'fecha_nacimiento' => 'Fecha Nacimiento',  
        'genero_id' => 'Genero ID',  
        'created_at' => 'Created At',  
        'updated_at' => 'Updated At',  
        'generoNombre' => Yii::t('app', 'Genero'),  
        'userLink' => Yii::t('app', 'User'),  
        'perfilIdLink' => Yii::t('app', 'Perfil'),  
    ];  
}  
  
/**  
 * @return \yii\db\ActiveQuery  
 */  
  
public function getGenero()  
{  
    return $this->hasOne(Genero::className(), ['id' => 'genero_id']);  
}  
  
/**  
 * @return \yii\db\ActiveQuery  
 */
```

```
public function getGeneroNombre()
{
    return $this->genero->genero_nombre;
}

/**
 * get lista de generos para lista desplegable
 */

public static function getGeneroLista()
{
    $dropciones = Genero::find()->asArray()->all();
    return ArrayHelper::map($dropciones, 'id', 'genero_nombre');
}

/**
 * @return \yii\db\ActiveQuery
 */

public function getUser()
{
    return $this->hasOne(User::className(), ['id' => 'user_id']);
}

/**
 * @get Username
 */

public function getUsername()
{
    return $this->user->username;
}
```

```
/**  
 * @getUserId  
 */  
  
public function getUserId()  
{  
    return $this->user ? $this->user->id : 'ninguno';  
}  
  
/**  
 * @getUserLink  
 */  
  
public function getUserLink()  
{  
    $url = Url::to(['user/view', 'id'=>$this->UserId]);  
    $opciones = [];  
    return Html::a($this->getUserName(), $url, $opciones);  
}  
  
/**  
 * @getPerfilLink  
 */  
  
public function getPerfilIdLink()  
{  
    $url = Url::to(['perfil/update', 'id'=>$this->id]);  
    $opciones = [];  
    return Html::a($this->id, $url, $opciones);  
}
```

Probablemente lo mencionaré unas cien veces en este libro, pero el estilo del código está establecido para evitar el ajuste de línea en PDF, por ello es que aparece en la forma en que lo hace en el libro. Para obtener una representación más clara del código, refiérase al Gist. En cualquier caso, el código ha sido probado y funciona.

Actualizar el Modelo User con Perfil

Así que ahora actualizamos el modelo con métodos para obtener la relación con perfil. Ya debería contar con getPerfil, así que no la incluiremos aquí.

En la parte superior de User.php, agregue:

```
use frontend\models\Perfil;
```

Agregue los siguientes métodos a User.php:

Gist:

[Relaciones User Perfil](#)

Del libro:

```
/**  
 * @getPerfilId  
 *  
 */  
  
public function getPerfilId()  
{  
    return $this->perfil ? $this->perfil->id : 'ninguno';  
}  
  
/**  
 * @getPerfilLink  
 *  
 */  
  
public function getPerfilLink()  
{  
    $url = Url::to(['perfil/view', 'id'=>$this->perfilId]);  
    $opciones = [];  
    return Html::a($this->perfil ? 'perfil' : 'ninguno', $url, $opciones);  
}
```

Así que el método getPerfil es muy familiar en este punto, simplemente estamos mapeando user_id en la tabla perfil a la columna id en la tabla user. De esta forma se asocian ambas.

El método getPerfilId es una sentencia ternaria que muestra ‘ninguno’ si el usuario no tiene un perfil. De esta forma cuando llamamos a este método en la IU, y el usuario no tiene un perfil, obtenemos una respuesta en lugar de nulo. Retornar nulo cuando se espera otra respuesta puede llevar a errores y a mucho tiempo de depuración. Es mejor tener en cuenta nulo cuando se tiene la oportunidad.

Necesitamos el método getPerfilId para alimentar a nuestro método getPerfilLink en el método Url::to de éste. El método getPerfilLink es simplemente como el método getUserLink del modelo perfil. Utiliza la clase auxiliar Url y la clase auxiliar Html, así que tenemos que incluir las sentencias use en la parte superior del archivo:

```
use yii\helpers\Url;
use yii\helpers\Html;
```

Finalizar el Modelo User

Mientras estamos aquí, vamos a añadir dos métodos más a User.php. Estos son del mismo tipo de métodos que ya hemos añadido, pero sirven a un propósito muy específico en la IU más adelante.

Gist:

[GetUserIdLink y GetUserLink](#)

Del libro:

```
/**
 * get user id Link
 *
 */

public function getUserIdLink()
{
    $url = Url::to(['user/update', 'id'=>$this->id]);
    $opciones = [];
    return Html::a($this->id, $url, $opciones);
}

/**
 * @getUserLink
 *
```

```
/*
public function getUserLink()
{
    $url = Url::to(['user/view', 'id'=>$this->id]);
    $opciones = [];
    return Html::a($this->username, $url, $opciones);
}
```

Por último un poco más de trabajo en el modelo User. Necesitamos agregar etiquetas para todos los siguientes métodos a través del método attributeLabels.

Gist:

[Etiquetas de Atributo](#)

Del libro:

```
/* Las etiquetas de los atributos de su modelo */
public function attributeLabels()
{
    return [
        /* Sus otras etiquetas de atributo */
        'rolNombre' => Yii::t('app', 'Rol'),
        'estadoNombre' => Yii::t('app', 'Estado'),
        'perfilId' => Yii::t('app', 'Perfil'),
        'perfilLink' => Yii::t('app', 'Perfil'),
        'userLink' => Yii::t('app', 'User'),
        'username' => Yii::t('app', 'User'),
        'tipoUsuarioNombre' => Yii::t('app', 'Tipo Usuario'),
        'tipoUsuarioId' => Yii::t('app', 'Tipo Usuario'),
        'userIdLink' => Yii::t('app', 'ID'),
    ];
}
```

Las etiquetas de atributo le indican a Yii 2 como mostrar sus atributos cuando aparecen en el sitio. En algunos casos, como las hemos usado aquí, el atributo es el nombre de un método. Por ejemplo, rolNombre es una etiqueta para el método getRolNombre, que construimos para este modelo User. Está apareado con el método Yii::t(), que es el método de traducción para la aplicación completa, así que establecerlo aquí debería hacerlo aparecer en cualquier lugar.

Si esto parece un poco confuso, no se preocupe por ello ahora. Se aclarará cuando vea las etiquetas de atributo aparecer en los archivos de vistas y en los widgets. Lo referenciaré nuevamente cuando lleguemos a ese punto.

El Modelo User Completo

Como referencia, voy a incluir una copia del archivo User.php, para que pueda asegurarse de que tiene todo lo necesario hasta este punto. Esto es sólo como referencia, no debería tener que copiar este archivo. Note que podría no estar en el mismo orden que el suyo.

Gist:

Modelo User

Del libro:

```
<?php
```

```
namespace common\models;
```

```
use Yii;
use yii\base\NotSupportedException;
use yii\db\ActiveRecord;
use yii\db\Expression;
use yii\web\IdentityInterface;
use yii\helpers\ArrayHelper;
use yii\helpers\Url;
use yii\helpers\Html;
use yii\helpers\Security;
use frontend\models\Perfil;
use backend\models\Rol;
use backend\models\Estado;
use backend\models\TipoUsuario;
/**
```

```
* User model
*
* @property integer $id
* @property string $username
* @property string $password_hash
* @property string $password_reset_token
* @property string $email
```

```
* @property string $auth_key
* @property integer $rol_id
* @property integer $estado_id
* @property integer $tipo_usuario_id
* @property integer $created_at
* @property integer $updated_at
* @property string $password write-only password
*/

```

```
class User extends ActiveRecord implements IdentityInterface
{
    const ESTADO_ACTIVO = 1;

    public static function tableName()
    {
        return 'user';
    }

    /**
     * behaviors
     */
}

public function behaviors()
{
    return [
        'timestamp' => [
            'class' => 'yii\behaviors\TimestampBehavior',
            'attributes' => [
                ActiveRecord::EVENT_BEFORE_INSERT => ['created_at', 'updated_at'],
                ActiveRecord::EVENT_BEFORE_UPDATE => ['updated_at'],
            ],
            'value' => new Expression('NOW()'),
        ],
    ];
}
```

```
}
```

```
/**
```

```
 * reglas de validación
 */
```

```
public function rules()
```

```
{
```

```
return [
```

```
['estado_id', 'default', 'value' => self::ESTADO_ACTIVO],  
[['estado_id'],'in', 'range'=>array_keys($this->getEstadoLista())],
```

```
['rol_id', 'default', 'value' => 1],  
[['rol_id'],'in', 'range'=>array_keys($this->getRolLista())],
```

```
['tipo_usuario_id', 'default', 'value' => 1],  
[['tipo_usuario_id'],'in', 'range'=>array_keys($this->getTipoUsuarioLista())],
```

```
['username', 'filter', 'filter' => 'trim'],  
['username', 'required'],  
['username', 'unique'],  
['username', 'string', 'min' => 2, 'max' => 255],
```

```
['email', 'filter', 'filter' => 'trim'],  
['email', 'required'],  
['email', 'email'],  
['email', 'unique'],  
];  
}
```

```
/* Las etiquetas de los atributos de su modelo */

public function attributeLabels()

{
    return [
        /* Sus otras etiquetas de atributo */
        'rolNombre' => Yii::t('app', 'Rol'),
        'estadoNombre' => Yii::t('app', 'Estado'),
        'perfilId' => Yii::t('app', 'Perfil'),
        'perfilLink' => Yii::t('app', 'Perfil'),
        'userLink' => Yii::t('app', 'User'),
        'username' => Yii::t('app', 'User'),
        'tipoUsuarioNombre' => Yii::t('app', 'Tipo Usuario'),
        'tipoUsuarioId' => Yii::t('app', 'Tipo Usuario'),
        'userIdLink' => Yii::t('app', 'ID'),
    ];
}

/**
 * @findIdentity
 */

public static function findIdentity($id)
{
    return static::findOne(['id' => $id, 'estado_id' => self::ESTADO_ACTIVO]);
}

/**
 * @inheritDoc
 */
```

```
public static function findIdentityByAccessToken($token, $type = null)
{
    throw new NotSupportedException
    ('"findIdentityByAccessToken" is not implemented.');
}

/**
 * Encuentra usuario por username
 * dividida en dos líneas para evitar ajuste de línea * @param string $username
 * @return static|null
 */

public static function findByUsername($username)
{
    return static::findOne(['username' => $username, 'estado_id' =>
        self::ESTADO_ACTIVO]);
}

/**
 * Encuentra usuario por clave de restablecimiento de password
 *
 * @param string $token clave de restablecimiento de password
 * @return static|null
 */

public static function findByPasswordResetToken($token)
{
    if (!static::isPasswordResetTokenValid($token)) {
        return null;
    }
}
```

```
return static::findOne([
    'password_reset_token' => $token,
    'estado_id' => self::ESTADO_ACTIVO,
]);
}

/**
 * Determina si la clave de restablecimiento de password es válida
 *
 * @param string $token clave de restablecimiento de password
 * @return boolean
 */
public static function isPasswordResetTokenValid($token)
{
    if (empty($token)) {
        return false;
    }

    $expire = Yii::$app->params['user.passwordResetTokenExpire'];
    $parts = explode('_', $token);
    $timestamp = (int) end($parts);
    return $timestamp + $expire >= time();
}

/**
 * @getId
 */
public function getId()
{
```

```
    return $this->getPrimaryKey();
}

/**
 * @getAuthKey
 */

public function getAuthKey()
{
    return $this->auth_key;
}

/**
 * @validateAuthKey
 */

public function validateAuthKey($authKey)
{
    return $this->getAuthKey() === $authKey;
}

/**
 * Valida password
 *
 * @param string $password password a validar
 * @return boolean si la password provista es válida para el usuario actual
 */

public function validatePassword($password)
{
    return Yii::$app->security->validatePassword($password, $this->password_hash);
}
```

```
/**  
 * Genera hash de password a partir de password y la establece en el modelo  
 *  
 * @param string $password  
 */  
  
public function setPassword($password)  
  
{  
    $this->password_hash = Yii::$app->security->generatePasswordHash($password);  
}  
  
/**  
 * Genera clave de autenticación "recuerdame"  
 */  
  
public function generateAuthKey()  
{  
    $this->auth_key = Yii::$app->security->generateRandomString();  
}  
  
/**  
 * Genera nueva clave de restablecimiento de password  
 * dividida en dos líneas para evitar ajuste de línea  
 */  
  
public function generatePasswordResetToken()  
{  
    $this->password_reset_token = Yii::$app->security->generateRandomString()  
        . '_'. time();  
}
```

```
/**  
 * Remueve clave de restablecimiento de password  
 */  
  
public function removePasswordResetToken()  
{  
    $this->password_reset_token = null;  
}  
  
  
public function getPerfil()  
{  
    return $this->hasOne(Perfil::className(), ['user_id' => 'id']);  
}  
  
  
/**  
 * @getPerfilId  
 *  
 */  
  
public function getPerfilId()  
{  
    return $this->perfil ? $this->perfil->id : 'ninguno';  
}  
  
  
/**  
 * @getPerfilLink  
 *  
 */  
  
public function getPerfilLink()  
{
```

```
$url = Url::to(['perfil/view', 'id'=>$this->perfilId]);
$opciones = [];
return Html::a($this->perfil ? 'perfil' : 'ninguno', $url, $opciones);
}

/** 
 * relación get rol
 *
 */
public function getRol()
{
    return $this->hasOne(Rol::className(), ['id' => 'rol_id']);
}

/** 
 * get rol nombre
 *
 */
public function getRolNombre()
{
    return $this->rol ? $this->rol->rol_nombre : '- sin rol -';
}

/** 
 * get lista de roles para lista desplegable
 */
public static function getRolLista()
{
    $dropciones = Rol::find()->asArray()->all();
    return ArrayHelper::map($dropciones, 'id', 'rol_nombre');
}
```

```
/**  
 * relación get estado  
 *  
 */  
  
public function getEstado()  
{  
    return $this->hasOne(Estado::className(), ['id' => 'estado_id']);  
}  
  
/**  
 * * get estado nombre  
 *  
 */  
  
public function getEstadoNombre()  
{  
    return $this->estado ? $this->estado->estado_nombre : '- sin estado -';  
}  
  
/**  
 * get lista de estados para lista desplegable  
 */  
  
public static function getEstadoLista()  
{  
    $dropciones = Estado::find()->asArray()->all();  
    return ArrayHelper::map($dropciones, 'id', 'estado_nombre');  
}  
  
  
public function getTipoUsuario()  
{  
    return $this->hasOne(TipoUsuario::className(), ['id' => 'tipo_usuario_id']);  
}
```

```
/**  
 * get tipo usuario nombre  
 *  
 */  
  
public function getTipoUsuarioNombre()  
{  
    return $this->tipoUsuario ?  
        $this->tipoUsuario->tipo_usuario_nombre : '- sin tipo usuario -';  
}  
  
/**  
 * get lista de tipos de usuario para lista desplegable  
 */  
  
public static function getTipoUsuarioLista()  
{  
    $dropciones = TipoUsuario::find()->asArray()->all();  
    return ArrayHelper::map($dropciones, 'id', 'tipo_usuario_nombre');  
}  
  
/**  
 * get tipo usuario id  
 *  
 */  
  
public function getTipoUsuarioId()  
{  
    return $this->tipoUsuario ? $this->tipoUsuario->id : 'ninguno';  
}  
  
/**  
 * get user id Link  
 *  
 */
```

```

public function getUserIdLink()
{
    $url = Url::to(['user/update', 'id'=>$this->id]);
    $opciones = [];
    return Html::a($this->id, $url, $opciones);
}

/**
 * @getUserLink
 *
 */
}

public function getUserLink()
{
    $url = Url::to(['user/view', 'id'=>$this->id]);
    $opciones = [];
    return Html::a($this->username, $url, $opciones);
}
}

```

Resumen



Este capítulo fue monstruoso. Hemos creado 5 tablas nuevas para agregar a nuestra estructura de datos. Hemos creado 5 modelos nuevos y actualizado el modelo User. No hemos hecho mucho de codificación aún, nos hemos apoyado principalmente en Gii para crear nuestros modelos. Luego

agregamos métodos de relación, reglas, y sentencias use, comportamientos y otros varios métodos y fines para liberar el poder de Yii 2.

Muchos de los métodos de relación que agregamos nos ayudarán mucho en la construcción de una IU intuitiva que permita la administración de usuarios y controlar su acceso a varias partes de la aplicación. Estamos construyendo la aplicación con un ojo puesto en la reutilización de código y la extensibilidad. Queremos construir una plantilla que sea un buen punto de partida para cualquier aplicación.

No hemos visto aún como todo esto se traduce en una aplicación, pero no se preocupe lo haremos. Y verá cuánto nos lo facilita Yii 2.

Capítulo Seis: Helpers (Auxiliares)

Para este momento habrá notado que este libro se organiza en torno a conceptos claves, no flujo de trabajo, al menos hasta este punto. La realidad del flujo de trabajo es que se salta un poco entre modelos, controladores y vistas, y esto puede volverse realmente confuso cuando se es nuevo en un framework.

Por ejemplo, observamos el modelo User extensamente, pero todavía no sabemos mucho sobre cómo ingresan los usuarios o aún sobre cómo los modelos que creamos en el capítulo darán soporte a ello. Pero no se preocupe, ya llegará. Estamos estableciendo la base para todo y las piezas caerán en su lugar. Y para el momento en que haya concluido, tendrá una plantilla funcional, con un modelo de usuario que funciona y control de administración completo a través de una IU que puede extender para obtener una aplicación robusta.

Este capítulo está dedicado a los helpers, clases especiales que creamos para dar formato y retornar ciertos valores. Esto puede sonar trivial, pero estos helpers van a acelerar rápidamente nuestro ciclo de desarrollo una vez que los tengamos.

Si lo piensa, un framework es un conjunto gigantesco de clases helper, y Yii 2 ciertamente se ajusta a esta descripción. Pero sin importar cuánto del framework haya, cada programador individual necesita sus propios helpers. Estos son clases que lo ayudarán a avanzar su desarrollo rápidamente con código reutilizable que habrá escrito usted mismo.

Ya hemos hablado sobre el hecho de que tenemos planeado controlar el acceso de los usuarios. Así que podríamos preguntarnos a nosotros mismos algunas cuestiones claves. **¿Cómo sabrá el controlador quién tiene un rol de Admin? ¿Cómo sabrá el estado o tipo_usuario_nombre del usuario?** Cuando alguien de clic sobre el link perfil en la navegación, ¿cómo sabrá el controlador si ya tiene un perfil o es necesario crear uno?

Para controlar el acceso basado en este tipo de cosas, necesitamos ser capaces de extraer los valores que deseamos de forma **sencilla y con facilidad**. Para ponerme las cosas más fáciles, he creado un número de métodos que retornarán los valores que necesitaré para operaciones más complejas.

Helpers de Valor

Podría haber puesto todos estos métodos en una única clase de Utilidades o Helper, pero recientemente leí Clean Code de Robert Martin, y uno de los resultados de leer libros de programación es que uno trata de adoptar los principios que le agradan. En este caso, realmente me agrada la idea de obtener ayuda semántica de los nombres y métodos de las clases. Así que me decidí por clases helper más pequeñas que sean más descriptivas.

Por ejemplo, podríamos tener una clase llamada ValorHelpers. Podríamos usarla para retornar valores con métodos nombrados de la siguiente manera:

- rolCoincide - ve si el rol de los usuarios coincide con un rol específico
- getRolValor - encuentra el valor de un rol
- getUsersRolValor - encuentra el valor de un rol de usuario específico
- esRolNombreValido - confirma que tenemos un nombre de rol válido
- estadoCoincide - determina si el estado del usuario actual coincide con el estado deseado
- getEstadoId - retorna la id de estado basado en un nombre de estado ingresado.
- tipoUsuarioCoincide - determina si el tipo usuario del usuario actual coincide con el tipo deseado

Estos métodos van a formar los bloques de nuestro control de acceso, que nos ayudará a construir una plantilla lista para producción.

Podría haber provisto con una solución mucho más simple o confiar en el RBAC de Yii 2, pero no quise hacer eso porque no se ajusta a esta plantilla. Quiero proveer de una base lo suficientemente robusta para ser capaces de seguir construyendo sobre ella y mejorar a medida que nuestras necesidades evolucionan.

La desventaja es que esto es un poco más complicado que simplemente dar un rápido recorrido por ‘esto es como funciona en Yii 2.’ La ventaja es que se acerca más al código que va a administrar su control de acceso, y por lo tanto, cuando necesite personalizarlo en el futuro de alguna forma que no podemos anticipar hoy, tendrá un mucho mejor entendimiento de cómo hacerlo.

También, a medida que avancemos en la creación de nuestros métodos helper, haremos uso de la implementación de Active Record de Yii para encontrar y devolver resultados. También usaremos sql puro en algunos casos, para que pueda ver el contraste entre las diferentes maneras de acceder a los datos de su base de datos.

Continuemos y creamos ValorHelpers.php en el directorio common/models. Para conservar la consistencia y no tener que movernos de un sitio a otro, simplemente voy a darle la clase completa. Luego discutiremos los métodos.

Gist:

[ValorHelpers](#)

Del libro:

```
<?php  
namespace common\models;  
  
use yii;  
use backend\models\Rol;  
use backend\models\Estado;  
use backend\models\TipoUsurio;  
use common\models\User;
```

```

class ValorHelpers
{
    public static function rolCoincide($rol_nombre)
    {
        $userTieneRolNombre = Yii::$app->user->identity->rol->rol_nombre;
        return $userTieneRolNombre == $rol_nombre ? true : false;
    }

    public static function getUsersRolValor($userId=null)
    {
        if ($userId == null){
            $usersRolValor = Yii::$app->user->identity->rol->rol_valor;
            return isset($usersRolValor) ? $usersRolValor : false;
        } else {
            $user = User::findOne($userId);
            $usersRolValor = $user->rol->rol_valor;
            return isset($usersRolValor) ? $usersRolValor : false;
        }
    }

    public static function getRolValor($rol_nombre)
    {
        $rol = Rol::find('rol_valor')
            ->where(['rol_nombre' => $rol_nombre])
            ->one();
        return isset($rol->rol_valor) ? $rol->rol_valor : false;
    }

    public static function esRolNombreValido($rol_nombre)
    {
        $rol = Rol::find('rol_nombre')
            ->where(['rol_nombre' => $rol_nombre])
            ->one();
        return isset($rol->rol_nombre) ? true : false;
    }
}

```

rol enviado ejemplo: docente

Si mandas un iduser,
devuelve su valor de rol
sino devuelve el valor
del rol del usuario
actual

```

public static function estadoCoincide($estado_nombre)
{
    $userTieneEstadoName = Yii::$app->user->identity->estado->estado_nombre;
    return $userTieneEstadoName == $estado_nombre ? true : false;
}

public static function getEstadoId($estado_nombre)
{
    $estado = Estado::find('id')
        ->where(['estado_nombre' => $estado_nombre])
        ->one();
    return isset($estado->id) ? $estado->id : false;
}

public static function tipoUsuarioCoincide($tipo_usuario_nombre)
{
    $userTieneTipoUsurioName = Yii::$app->user->identity
->tipoUsuario->tipo_usuario_nombre;
    return $userTieneTipoUsurioName == $tipo_usuario_nombre ? true : false;
}

```

Por favor no se preocupe por memorizar estos métodos. No es necesario que lo haga. Sin embargo vamos a revisarlos en detalle, así obtendrá una visión más general de cómo trabaja todo esto a la vez que será introducido al Active Record de Yii 2.

De cualquier forma, puede ver que tenemos 7 métodos bastante simples que retornan valores asociados con los modelos que creamos.

Así que por ejemplo, si queremos conocer el valor de Admin, podemos llamar:

```
getRolValor('Admin');
```

Y de acuerdo a lo que hay en nuestra BD:

+ Opciones			id	rol_nombre	rol_valor	
<input type="checkbox"/>	 Editar	 Copiar	 Borrar	1	Usuario	10
<input type="checkbox"/>	 Editar	 Copiar	 Borrar	2	Admin	20

Registros de Rol

La respuesta es 20. Así que para controlar el acceso al backend, por ejemplo, escribo un método que restringe el acceso a los usuarios que tienen un valor de rol mínimo de 20. O tengo uno que coincide exactamente con 20 si quiero que solamente un tipo de rol tenga acceso.

No estamos realizando esta decisión o construyendo ese método aún, sólo estamos anticipando que necesitaremos el valor para decisiones que tomaremos en el futuro.

Una cosa que puede querer saber es el propósito de rol_valor, ¿por qué no usar simplemente la clave primaria id? La razón por la que he añadido una columna rol_valor, sin embargo, fue para darnos mayor flexibilidad en el diseño del sistema. Verá como esto entra en juego más adelante.

También notará que nuestros métodos son métodos públicos y estáticos, lo que significa que pueden ser llamados como:

```
ValorHelpers::getRolValor('Admin');
```

Siempre y cuando haya incluido la sentencia use:

```
use common\models\ValorHelpers;
```

Al principio del archivo, puede usar el método de esa manera.

Hemos hecho nuestro mayor esfuerzo con los métodos helper para hacerlos lo más agradable semánticamente y lo más fácil de comprender que sea posible. Esto permite que sea más fácil comprender su propósito cuando regrese a él después de un periodo de tiempo.

Estos son métodos simples, pero veamoslos. Aquí está el primero:

```
public static function rolCoincide($rol_nombre)
{
    $userTieneRolNombre = Yii::$app->user->identity->rol->rol_nombre;
    return $userTieneRolNombre == $rol_nombre ? true : false;
}
```

Lo llamamos rolCoincide como si realizaramos una pregunta, así que tiene sentido que devuelva verdadero o falso.

Podríamos usarlo de la siguiente manera:

```
if(ValorHelpers::rolCoincide('Admin')){

    //entonces permitirle al usuario hacer esto

}
```

Para que el método realice ese cálculo, nos apoyamos en la relación entre los modelos User y Rol en lo siguiente:

```
Yii::$app->user->identity->rol->rol_nombre;
```

Así que, la primera cosa a notar sobre ello es que los atributos del usuario están siempre disponibles para nosotros a través de una llamada como:

```
Yii::$app->user->identity->username;
```

Eso devolverá el username del usuario actual, que deberá estar logueado para que esto funcione. También necesita:

```
use yii;
```

Eso debe ir al principio del archivo en el que desea acceder al usuario con Yii::\$app.

Así que, de regreso a:

```
Yii::$app->user->identity->rol->rol_nombre;
```

getRol ()

Estamos usando la sintaxis mágica get para el método getRol en el modelo User, por ello es que usted ve ->rol en lugar de ->getRol(). Una vez que los modelos están enlazados a través de la relación, como lo hemos hecho en capítulo anterior, podemos acceder a las propiedades del modelo relacionado, en ese caso, su rol_nombre, que pertenece a Rol.

Puede ver cuánto poder hay en una sola línea de código. Es simplemente increíble. Esto hace que sea un placer trabajar con el framework.

Para mantener las líneas cortas, especialmente para este libro, lo asigno a una variables:

```
$userTieneRolNombre = Yii::$app->user->identity->rol->rol_nombre;
```

A partir de allí realizamos una comparación de igualdad simple a través de un operador ternario para comprobar si coincide con el nombre de rol que le fue entregado y terminamos:

```
return $userTieneRolNombre == $rol_nombre ? true : false;
```

Sintaxis agradable e intuitiva, y eso la vuelve muy mantenible.

Bien, siguiente método:

```

public static function getUsersRolValor($userId=null)
{
    if ($userId == null){
        $usersRolValor = Yii::$app->user->identity->rol->rol_valor;
        return isset($usersRolValor) ? $usersRolValor : false;
    } else {
        $user = User::findOne($userId);
        $usersRolValor = $user->rol->rol_valor;
        return isset($usersRolValor) ? $usersRolValor : false;
    }
}

```

Al establecer \$userId a nulo en la firma del método, nos damos la opción de ingresar un valor para ella o no. En el caso de un usuario que ha iniciado sesión, no necesitamos ingresar \$userId, ya la tenemos.

Así que en ese caso, obtenemos:

```

if ($userId == null){

$usersRolValor = Yii::$app->user->identity->rol->rol_valor;

return isset($usersRolValor) ? $usersRolValor : false;

```

Nuevamente la relación entre User y Rol ya está allí y es fácil de usar. Podemos comprobar si \$usersRolValor está establecido, y si no devuelve falso. De otra manera devuelve \$usersRoleValue.

Pero en algunos casos, el usuario puede no haber iniciado sesión, en cuyo caso necesitamos ingresar la id del usuario. Es sólo ligeramente más complicado. Tenemos que crear una instancia ActiveRecord del usuario, así que podemos usar la relación para encontrar el \$usersRolValor:

```
$user = User::findOne($userId);
```

Eso se traduce a groso modo en SQL como:

```
Select id FROM User WHERE id = $userId;
```

Obviamente el formateo de ActiveRecord hace que establecer una consulta sea fácil. La guía de Yii 2 tiene muchos ejemplos de como usar ActiveRecord.

Así que en lugar de reproducir lo que hay en la guía de Yii 2, lo referiré a ella con la sugerencia de que se tome algunos minutos para mirarla:

Yii 2 Active Record

No se preocupe sin embargo, no lo dejaré varado. Continuaré explicando completamente todo lo que usemos.

Una vez que creamos una instancia de User, usamos nuestra relación con rol para terminar:

```
$usersRolValor = $user->rol->rol_valor;

return isset($usersRolValor) ? $usersRolValor : false;
```

Así que tuvimos que pasar a través de algunas etapas para obtener el valor de rol de nuestro usuario, pero lo vale. Nos dará la flexibilidad de construir el resto de nuestro control de acceso. Y para el momento en que estemos usando nuestros métodos helper en nuestros controladores, estará muy feliz de que hayamos hecho todo esto.

A continuación, simplemente queremos un valor de rol sin el usuario. Ingresamos el nombre del rol y usamos una instancia de ActiveRecord para accederlo:

```
public static function getRolValor($rol_nombre)
{
    $rol = Rol::find('rol_valor')
        ->where(['rol_nombre' => $rol_nombre])
        ->one();
    return isset($rol->rol_valor) ? $rol->rol_valor : false;
}
```

En esta llamada a ActiveRecord, decidí cambiar y utilizar el método find en lugar de findOne debido a que no se trata de una clave primaria y findOne es un atajo para buscar por la clave primaria.

También, simplemente un adelanto, para usar ActiveRecord y acceder a los resultados, necesita incluir sentencias use para los modelos que quiere devolver. Puede ver que hemos incluido lo siguiente al comienzo del archivo:

```
use yii;
use backend\models\Rol;
use backend\models\Estado;
use backend\models\TipoUsuario;
use common\models\User;
```

Luego tenemos una breve comprobación para ver si el nombre de rol es válido, esto ayuda a prevenir errores de programación:

```
public static function esRolNombreValido($rol_nombre)
{
    $rol = Rol::find('rol_nombre')
        ->where(['rol_nombre' => $rol_nombre])
        ->one();
    return isset($rol->rol_nombre) ? true : false;
}
```

Para implementar el control de acceso, tener control sobre el estado del usuario también es importante:

```
public static function estadoCoincide($estado_nombre)
{
    $userTieneEstadoName = Yii::$app->user->identity->estado->estado_nombre;
    return $userTieneEstadoName == $estado_nombre ? true : false;
}
```

El método estadoCoincide es exactamente como el método rolCoincide al comienzo de la clase, usando la misma técnica con relaciones, así que revise ese método si no está seguro sobre cómo trabaja.

A continuación:

```
public static function getEstadoId($estado_nombre)
{
    $estado = Estado::find('id')
        ->where(['estado_nombre' => $estado_nombre])
        ->one();
    return isset($estado->id) ? $estado->id : false;
}
```

Aquí simplemente estamos ingresando el nombre de estado para devolver la id a través de ActiveRecord. Esto será de utilidad cuando queramos remover la constante estado del modelo user. Pero no se preocupe por eso ahora.

Y finalmente, tenemos:

```
public static function tipoUsuarioCoincide($tipo_usuario_nombre)
{
    $userTieneTipoUsurioName =
        Yii::$app->user->identity->tipoUsuario->tipo_usuario_nombre;
    return $userTieneTipoUsurioName == $tipo_usuario_nombre ? true : false;
}
```

Como rolCoincide y estadoCoincide, simplemente devuelve verdadero o falso si el tipoUsuario del usuario coincide con el especificado en la firma.

Permisos Helpers

Al imaginar cómo usaríamos nuestros helpers, conocer el valor no es suficiente. Si vamos a controlar el acceso a áreas de nuestra aplicación, deseamos helpers que usan el valor y definen permisos. Así que vamos a crear una clase PermisosHelpers en common/models.

Continuemos y creemos PermisosHelpers.php en common/models.

Gist:

[PermisosHelpers](#)

Del libro:

```
<?php
namespace common\models;

use common\models\ValorHelpers;
use yii;
use yii\web\Controller;
use yii\helpers\Url;

class PermisosHelpers
{
    public static function requerirUpgradeA($tipo_usuario_nombre)
    {
        if (!ValorHelpers::tipoUsuarioCoincide($tipo_usuario_nombre)) {
            return Yii::$app->getResponse()->redirect(Url::to(['upgrade/index']));
        }
    }
}
```

```

public static function requerirEstado($estado_nombre)
{
    return ValorHelpers::estadoCoincide($estado_nombre);
}

public static function requerirRol($rol_nombre)
{
    return ValorHelpers::rolCoincide($rol_nombre);
}

public static function requerirMinimoRol($rol_nombre, $userId=null)
{
    if (ValorHelpers::esRolNombreValido($rol_nombre)){
        if ($userId == null) {
            $userRolValor = ValorHelpers::getUsersRolValor();
        } else {
            $userRolValor = ValorHelpers::getUsersRolValor($userId);
        }
        return $userRolValor >= ValorHelpers::getRolValor($rol_nombre) ?
true : false;
    } else {
        return false;
    }
}

public static function userDebeSerPropietario($model_nombre, $model_id)
{
    $conexion = \Yii::$app->db;
    $userid = Yii::$app->user->identity->id;
    $sql = "SELECT id FROM $model_nombre WHERE user_id=:userid AND id=:model_id";
    $comando = $conexion->createCommand($sql);
    $comando->bindValue(":userid", $userid);
    $comando->bindValue(":model_id", $model_id);
    if($result = $comando->queryOne()) {
        return true;
    } else {
        return false;
    }
}

```

Superusuario Docente
5 > 10

```
}
```

Bien, genial, sólo 5 métodos y veremos cómo utilizan ValorHelpers para ayudarnos a controlar el acceso de formas que serán importantes para nosotros. No vamos a profundizar demasiado, sin embargo, entraremos en mayores detalles cuando realmente usemos los métodos en la aplicación.

Recuerde, todos estos métodos helper son públicos y estáticos de modo que pueden ser llamados así:

```
PermisosHelpers::requerirUpgradeA('Pago');
```

Y así nos dirigimos directamente a nuestro primer ejemplo. Si en efecto construye una aplicación que tiene un área para usuarios de un tipo diferente, como en el ejemplo de arriba, usuarios de Pago, entonces esto es perfecto para su controlador. Comprueba al usuario actual para ver si el tipo de usuario coincide con el que entregó al método.

Hace esto usando nuestro práctico método ValorHelpers::tipoUsuarioCoincide. Así que ya estamos reutilizando código de nuestra clase ValorHelpers, tiene que encantarle eso.

Bien, de regreso al método. Si el tipo usuario coincide, bien, continuar, sino, redirigir a la página de upgrade. Por supuesto si tiene un destino diferente en mente, simplemente ponga el controlador/accion en el método.

Si tiene más de una página upgrade o redirect, podría reescribir el método para aceptar un segundo argumento, tal como:

```
public static function requiereUpgradeA($tipo_usuario_nombre,
    $redireccion_destino)
{
    if (!ValorHelpers::tipoUsuarioCoincide($tipo_usuario_nombre)) {
        return Yii::$app->getResponse()->redirect(Url::to([$redireccion_destino]));
    }
}
```

En ese caso, simplemente pasaría la acción del controlador como una cadena ‘upgrade/salespitch’ por ejemplo, como segundo argumento. De cualquier manera, sólo incluí el método que tiene la redirección codificada porque no anticipo que la aplicación sea más complicada que eso y me agrada la sintaxis simple del argumento único.

Los siguientes 2 métodos, requerirEstado y requerirRol simplemente llaman a los métodos ya existentes estadoCoincide y rolCoincide de ValorHelper y no son absolutamente necesarios, ya que no hacen nada nuevo.

La razón por la que existen es para que pueda tener una consistencia cosmética y sintáctica que me facilite trabajar con mis métodos de control de acceso.

Puede ver el tema que surge de PermisosHelpers::requerirAlgo en la forma de un administrador de permisos. Es simplemente super sencillo para mi de recordar, trabajar, y mantener.

De cualquier forma, ambos métodos requieren una coincidencia exacta para devolver verdadero.

Luego, construí un método requerirMinimoRol con el operador `>=`, así que si quisiera por ejemplo que los roles Admin y SuperUsuario fueran capaces de acceder al backend, podría controlar el acceso al backend con este método.

Primero comprobamos para ver esRolNombreValido:

```
if (ValorHelpers::esRolNombreValido($rol_nombre)){
```

Si eso falla, inmediatamente devuelve falso. Si está bien, entonces comprobamos para ver si hemos ingresado una \$userId:

```
if ($userId == null) {  
  
    $userRolValor = ValorHelpers::getUsersRolValor();  
  
}
```

Si es nulo, llama a getUsersRolValor sin entregar la \$userId y establece el resultado a \$userRolValor. Esto asume que el usuario ha iniciado sesión y el método getUsersRolValor manejará este escenario.

Si no es nulo, y hemos ingresado un valor para \$userId, entonces el método getUsersRolValor también manejará ese escenario.

Luego realizamos una simple comprobación con `>=` para ver si el usuario cumple los requerirMinimoRol que hemos ingresado:

```
return $userRolValor >= ValorHelpers::getRolValor($rol_nombre) ? true : false;
```

Este tenía unas cuantas partes móviles, que pueden ser difíciles de seguir, pero si quiere llamarlo, mire lo intuitiva que es la sintaxis:

```
PermisosHelpers::requerirMinimoRol('Admin');
```

Típicamente, va a usarlo como parte de una sentencia if, ya que devuelve verdadero o falso.

Aún cuando no lo hemos explicado completamente, está comenzando a tener una idea de cómo contestamos la pregunta, “¿Cómo controlaremos el acceso al backend?”

No se preocupe si no lo entiende completamente ahora, lo hará cuando lo vea en acción.

El último método, toma dos parámetros, nombre de modelo e id de modelo. Luego realiza una consulta para ver si el usuario actual es el propietario de ese registro específico.

Puede usar el método como un ejemplo de la manera de ejecutar una consulta pura:

```

public static function userDebeSerPropietario($model_nombre, $model_id)
{
    $conexion = \Yii::$app->db;
    $userid = Yii::$app->user->identity->id;
    $sql = "SELECT id FROM $model_nombre WHERE user_id=:userid AND id=:model_id";
    $comando = $conexion->createCommand($sql);
    $comando->bindValue(":userid", $userid);
    $comando->bindValue(":model_id", $model_id);
    if($result = $comando->queryOne()) {
        return true;
    } else {
        return false;
    }
}

```

Si el usuario posee el registro, devuelve verdadero, de lo contrario devuelve falso. Cuando lo usemos, la sintaxis típicamente lucirá así:

```

if (PermisosHelpers::userDebeSerPropietario ('perfil', $model->id)) {

    //hacer algo

}

```

¿Así que cómo lo usaría? Bien, digamos que tiene un grupo de posts u otros registros que son visibles para todos, pero sólo el autor puede actualizarlos o borrarlos y desea que la navegación sólo sea visible para el propietario del registro. Así que el “hacer algo” en el ejemplo de arriba, podría ser mostrar la navegación en un archivo de vista. Usaremos este ejemplo exactamente más adelante en nuestro libro.

Lo escribí de esta manera porque me agrada la sintaxis y sentí que esta sería una buena manera de trabajar con él. Pero tenga en mente que típicamente hay muchas maneras de lograr la misma cosa y un helper no siempre es necesario. Me agrada usarlos porque también me dan consistencia en la codificación, pero esta es definitivamente un área dónde debe usar su propio juicio.

Registros Helpers

Bien, tenemos un último archivo helper, RegistrosHelpers. Continuemos y creemos RegistrosHelpers.php en common/models y pongamos el siguiente contenido en el archivo:

Gist:

[RegistrosHelpers](#)

Del libro:

```
<?php  
namespace common\models;  
  
use yii;  
  
class RegistrosHelpers  
{  
    public static function userTiene($modelo_nombre)  
    {  
        $conexion = \Yii::$app->db;  
        $userid = Yii::$app->user->identity->id;  
        $sql = "SELECT id FROM $modelo_nombre WHERE user_id=:userid";  
        $comando = $conexion->createCommand($sql);  
        $comando->bindValue(":userid", $userid);  
        $resultado = $comando->queryOne();  
        if ($resultado == null) {  
            return false;  
        } else {  
            return $resultado['id'];  
        }  
    }  
}
```

Esta clase tiene solamente un método. Lo que he planeado para nuestra aplicación es un perfil de usuario y quiero un link a Perfil que cuando haga clic en él, determine si el usuario tiene o no un perfil o si necesita crear uno.

Quise mantener la sintaxis en mi controlador muy intuitiva y que el resultado sea formateado ya sea a falso o a la id del registro. De esa forma si retorna falso, puedo hacer que el usuario cree el registro, y si retorna la id del registro debido a que el usuario ya tiene uno, puedo mostrar la vista. Algo así:

```
If ($already_exists = RecordHelpers::userTiene('perfil')) {

    // mostrar perfil con id con valor de $ya_existe

} else {

    // ir al formulario de creación de perfil

}
```

Este tipo de sintaxis hace que sea increíblemente fácil de comprender lo que sucede. Si la sentencia if devuelve una id de registro, mostrar el perfil con esa id de registro, que ahora es referenciada por la variable \$ya_existe. Si devuelve falso, ir al formulario de creación.

También lo escribí de manera de poder usarlo con otros modelos, simplemente necesito entregarle el nombre de modelo como una cadena.

También debería notar que este método está escrito para devolver un único registro, deberá modificarlo si existe la posibilidad de que el usuario pueda tener múltiples registros, múltiples perfiles por ejemplo.

Resumen



Bien, para ahora está dándose cuenta de que aprender Yii 2 es divertido, pero también es mucho trabajo. Este es un framework enorme, elegante y poderoso, capaz de hacer muchas cosas. Ya hemos hecho mucho. Configuramos la aplicación, repasamos brevemente la arquitectura MVC, y modificamos el modelo User. También construimos 5 modelos nuevos y pusimos en su lugar sus relaciones con el modelo User, y construimos un número de métodos helper para simplificar la codificación cuando profundicemos en la aplicación, específicamente para el control de acceso.

Hemos hecho todo esto, y nuestra aplicación aún no hace actualmente más que lo que hacía cuando la instaló. Gracias por su paciencia. En los siguientes capítulos, comenzaremos a agregar características a nuestra aplicación.

Capítulo Siete: Controlador del Sitio

Vamos a continuar nuestro desarrollo con una exploración detallada del Controlador del Sitio y sus vistas relacionadas. Esto nos introducirá a un número mayor de conceptos dentro de los controladores y modelos de Yii 2, para que podamos obtener conocimiento sobre cómo trabajan.

Estaremos saltando un poco de un lugar a otro, así que no se preocupe si no memoriza instantáneamente toda esta información. Piense en ella como una visita guiada, donde nos vemos introducidos a una base de conocimiento a la que podemos referirnos para construir sobre ella, mejorando nuestro entendimiento de cómo opera Yii 2.

Lo primero que debería mencionar es que hay dos controladores de sitio, uno para el backend y otro para el frontend.

Comenzaremos discutiendo el controlador de sitio del frontend, lo que nos llevará a través del registro e inicio de sesión, luego nos moveremos al controlador de sitio del backend, señalando las diferencias. Y luego agregaremos nuestra primera funcionalidad nueva desde que instalamos la plantilla avanzada. Haremos que el controlador de sitio del backend imponga un nivel de acceso diferente para el inicio de sesión.

Bien, comencemos. Observaremos el frontend/controllers/SiteController.php en partes, no hay necesidad de reproducir el archivo completo, ya que no lo modificaremos.

En primer lugar, namespace y sentencias use:

```
<?php  
namespace frontend\controllers;  
  
use Yii;  
use common\models\LoginForm;  
use frontend\models\PasswordResetRequestForm;  
use frontend\models\ResetPasswordForm;  
use frontend\models\SignupForm;  
use frontend\models>ContactForm;  
use yii\base\InvalidArgumentException;  
use yii\web\BadRequestHttpException;  
use yii\web\Controller;  
use yii\filters\VerbFilter;  
use yii\filters\AccessControl;
```

Usa bastantes modelos y los veremos en acción. Luego tenemos una declaración de clase:

```
class SiteController extends Controller
```

Puede ver que extiende a Controller. Cuando tenga tiempo, revise Controller, le dará una mejor idea de cómo funcionan las cosas, pero esté advertido, el código del framework es a veces difícil de seguir, especialmente para principiantes. El código que ve en la superficie es mucho más amigable que lo que verá en el interior.

Behaviors

Luego tenemos algo similar, un método behaviors. Lo vimos en algunos de nuestros modelos con el behavior TimeStamp. Nuestros controladores usan el behavior AccessControl:

```
public function behaviors()
{
    return [
        'access' => [
            'class' => AccessControl::className(),
            'only' => ['logout', 'signup'],
            'rules' => [
                [
                    'actions' => ['signup'],
                    'allow' => true,
                    'roles' => ['?'],
                ],
                [
                    'actions' => ['logout'],
                    'allow' => true,
                    'roles' => ['@'],
                ],
            ],
            'verbs' => [
                'class' => VerbFilter::className(),
                'actions' => [
                    'logout' => ['post'],
                ],
            ],
        ];
}
```

Este es el método predefinido de Yii 2 para controlar el acceso al sitio, y principalmente diferencia entre logueado e invitado. El ‘?’ es invitado y ‘@’ es logueado. Así que ahora que sabemos eso,

podemos ver la sintaxis increíblemente intuitiva de Yii en acción. Pero sólo para asegurarnos descompongámosla. Primero:

Nombre del behavior:

```
return [
    'access' => [
```

Este es llamado acceso, pero podría llamarlo con cualquier cadena de texto y funcionaría de igual manera. Luego viene la clase:

```
'class' => AccessControl::className(),
```

Esto simplemente nos dice que clase aplicar. Luego vemos a qué acciones aplicar el comportamiento:

```
'only' => ['logout', 'signup'],
```

Así que estas reglas sólo se aplicarán a logout y signup. Luego vienen las reglas, y aquí es donde podemos aplicarlas a acciones específicas:

```
[
    'actions' => ['signup'],
    'allow' => true,
    'roles' => ['?'],
],
[
    'actions' => ['logout'],
    'allow' => true,
    'roles' => ['@'],
],
```

Así, la primer regla dice signup, true, invitado, en otras palabras a los invitados se les permite registrarse. La segunda regla dice, logout, true, usuario logueado, en otras palabras a los usuarios logueados se les permite acceder a la acción logout. Ya que especificamos la parte 'only' del método, todas las otras acciones no son controladas por estas reglas.

A medida que recorramos las otras acciones, veremos por qué esto tiene sentido.

Acciones

El primer método debajo de behaviors es actions:

```
public function actions()
{
    return [
        'error' => [
            'class' => 'yii\web\ErrorAction',
        ],
        'captcha' => [
            'class' => 'yii\captcha\CaptchaAction',
            'fixedVerifyCode' => YII_ENV_TEST ? 'testme' : null,
        ],
    ];
}
```

Este es un método de configuración, que nos dice qué clase usar para error y que clase usar para Captcha. La configuración en actions hace que estas acciones estén disponibles para el controlador. Así que si quiere usar Captcha para algo, necesita configurarla en el controlador, como se muestra arriba. También necesitamos configurarla como un widget, que veremos en acción pronto.

Acción Index

Continuemos con actionIndex:

```
public function actionIndex()
{
    return $this->render('index');
}
```

¡Si! Las cosas tenían que ponerse más fáciles tarde o temprano... Esta simplemente llama al método render para la vista, en este caso ‘index’.

Esto nos da una oportunidad de refreshar como trabaja el ruteo. La ruta a index se ve así:

`yii2build.com/index.php?r=site/index`

Ya que dejamos las feas url en su lugar, es muy explícito. Index.php es la página de bootstrap, todo pasa por ella. La r por ruta, = site/index. En este caso, site es el controlador, index es la acción. Si omite la acción del controlador, buscará una acción index por defecto. Si no existe, devolverá un error.

También una nota rápida, el valor por defecto a site está establecido al mostrado arriba, así que si simplemente tipea el dominio, yii2build.com, esa es la ruta que obtendrá.

La acción en la mayoría de los casos mostrará una vista, usando la sintaxis que vemos en la acción index de arriba, y así es como llegamos a ver la página. En cualquier instancia, debería tener un sentido de cómo el controlador/accion nos desplaza por el sitio.

Acción Login

El siguiente método del controlador es actionLogin:

```
public function actionLogin()
{
    if (!\Yii::$app->user->isGuest) {
        return $this->goHome();
    }

    $model = new LoginForm();

    if ($model->load(Yii::$app->request->post()) && $model->login()) {

        return $this->goBack();

    } else {
        return $this->render('login', [
            'model' => $model,
        ]);
    }
}
```

Bien, fantástico, veremos cómo el usuario inicia sesión. Lo primero que ocurre es la comprobación para ver si ya ha iniciado sesión:

```
if (!\Yii::$app->user->isGuest) {
    return $this->goHome();
}
```

Lo hace simplemente chequeando si el usuario no es un invitado. Si no es un invitado, eso significa que que ya ha iniciado sesión y en ese caso, lo enviamos a la página de inicio.

Modelo Login Form

Si no ha iniciado sesión, creamos una instancia de LoginForm:

```
$model = new LoginForm();
```

Hablamos un poco acerca de modelo LoginForm, cuando estábamos modificando el modelo User, pero vale la pena mirar más de cerca, para que podamos entender exactamente cómo funciona. Está ubicado en common/models, así que esa es la razón de que el primer bloque se vea como:

```
<?php  
namespace common\models;  
  
use Yii;  
use yii\base\Model;
```

Luego tenemos nuestra declaración y propiedades de clase:

```
class LoginForm extends Model  
{  
    public $username;  
    public $password;  
    public $rememberMe = true;  
  
    private $_user = false;
```

Recuerde, este modelo extiende Model, no User, así que tenemos que declarar las propiedades. Estamos estableciendo el valor por defecto de \$rememberMe a verdadero, esto establece la bandera en el formulario para la cookie. Establecemos el valor por defecto de \$_user a falso y veremos por qué en un momento.

Luego tenemos un método rules:

```
public function rules()  
{  
    return [  
        // username and password are both required  
        [['username', 'password'], 'required'],  
        // rememberMe must be a boolean value  
        ['rememberMe', 'boolean'],  
        // password is validated by validatePassword()  
        ['password', 'validatePassword'],  
    ];  
}
```

Yii 2 provee de comentarios para explicar las validaciones que usamos. Puede ver que password usa el método validatePassword como su validador:

```
public function validatePassword($attribute, $params)
{
    if (!$this->hasErrors()) {
        $user = $this->getUser();
        if (!$user || !$user->validatePassword($this->password)) {
            $this->addError($attribute, 'Incorrect username or password.');
        }
    }
}
```

Un método bastante intuitivo, si no hay errores, genial obtener el usuario, de otra manera declarar el error.

Luego tenemos el propio método login de acuerdo a Yii 2.0.3:

```
/** 
 * 2 líneas en la sentencia return para evitar ajuste de línea
 */

public function login()
{
    if ($this->validate()){
        return Yii::$app->user->login($this->getUser(),
            $this->rememberMe ? 3600 * 24 * 30 : 0);

    } else {
        return false;
    }
}
```

Bien, ligeramente más complejo. \$this->validate() llama al método validate de Model. y también llamamos a getUser del modelo LoginForm, que veremos en un momento. Si validamos, devolvemos el método login del usuario disponible para nosotros a través de Yii::\$app.

Este es en realidad una referencia al modelo User en vendor/yiisoft/yii/web/User. Esta es la clase que Yii 2 usa para administrar identidad y logueo y allí hay un método login bastante complicado. Se vuelve un poco confuso tener múltiples modelos y múltiples métodos con el mismo nombre, pero esa es la naturaleza de la bestia. Podemos ver que recibe al usuario y a remember me en la firma y luego loguea. No avanzaré en ese método login ya que está más allá del alcance de esta discusión y definitivamente no para principiantes, pero puede revisarlo por usted mismo si lo desea.

Al menos hace que el código del modelo LoginForm parezca muy intuitivo en comparación. De cualquier manera, funciona, y loguea al usuario y establece la cookie remember me si esa bandera ha sido establecida en el formulario.

Si algo falla, devuelve falso, y usualmente se enviarán mensajes de validación a la vista que le dicen al usuario cuál es el problema. En la superficie, es muy simple.

Bien, último método de este modelo:

```
public function getUser()
{
    if ($this->_user === false) {
        $this->_user = User::findByUsername($this->username);
    }

    return $this->_user;
}
```

Ya que sabemos que el atributo privado `$_user` tiene un valor por defecto de falso, la condición en la sentencia if va a cumplirse si este método no ha sido ejecutado ya. Así que si no hay un `username` en `$_user`, entonces usa un método estático del modelo `User` para devolver una instancia del usuario deseado y la asigna a `$_user`.

Para que esto funcione, el modelo `LoginForm` obviamente tiene que obtener los valores de sus propiedades de post, así que sabe a quién se refiere `$this->username`, y puede buscar al usuario y asignarlo a `$_user`. Así que veamos cómo obtenemos los datos del post volviendo ahora al método `actionLogin` de `SiteController` y continuando a partir de donde lo dejamos. Así que luego de llamar a una nueva instancia del modelo `LoginForm` model, obtenemos:

```
if ($model->load(Yii::$app->request->post()) && $model->login()) {
    return $this->goBack();
} else {
    return $this->render('login', [
        'model' => $model,
    ]);
}
```

Obtendrá sus datos post de:

```
Yii::$app->request->post()
```

Si podemos cargar los datos post, que validarán de acuerdo al modelo que describimos anteriormente y si puede utilizar el método `login` del modelo, devolverá al usuario a la página en la que se encontraba usando:

```
return $this->goBack();
```

Sólo que ahora estará en un estado logueado.

De cualquier otra forma, si algo falla o no hemos enviado aún el formulario, lo mostraremos:

```

} else {
    return $this->render('login', [
        'model' => $model,
    ]);
}

```

También puede ver que está pasando una instancia de \$model a la vista, que sabemos en este caso se trata del modelo LoginForm que establecimos antes con:

```
$model = new LoginForm();
```

Eso hace que el modelo esté disponible para la vista. Y eso es todo para login, con suerte ha obtenido una buena comprensión de cómo funciona.

Acción Logout

El método actionLogout es significativamente más simple:

```

public function actionLogout()
{
    Yii::$app->user->logout();

    return $this->goHome();
}

```

Usa el método logout del modelo user enterrado profundamente en los entresijos de Yii 2 y envía al usuario a la página index a través de goHome().

Acción Contacto

El siguiente método nos rutea a una página de contacto simple, con el método actionContact, pero hay algunas cosas interesantes allí.

```

public function actionContact()
{
    $model = new ContactForm();
    if ($model->load(Yii::$app->request->post()) && $model->validate()) {
        if ($model->sendEmail(Yii::$app->params['adminEmail'])) {
            Yii::$app->session->setFlash('success',
                'Thank you for contacting us. We will respond to you as soon as possible.');
        }
    }
}

```

```
    } else {

        Yii::$app->session->setFlash('error', 'There was an error sending email.');

    }

    return $this->refresh();

} else {

    return $this->render('contact', [
        'model' => $model,
    ]);
}

}
```

Modelo Contact Form

Lo primero que el método actionContact del controlador hace es llamar a un nuevo modelo de formulario, ContactForm, ubicado en frontend/models. Es otro modelo de formulario que extiende Model, así que revisémoslo:

```
<?php

namespace frontend\models;

use Yii;
use yii\base\Model;

/**
 * ContactForm is the model behind the contact form.
 */
class ContactForm extends Model
{

    public $name;
    public $email;
    public $subject;
    public $body;
    public $verifyCode;
```

```
/**  
 * @inheritdoc  
 */  
  
public function rules()  
{  
    return [  
        // name, email, subject and body are required  
        [['name', 'email', 'subject', 'body'], 'required'],  
        // email has to be a valid email address  
        ['email', 'email'],  
        // verifyCode needs to be entered correctly  
        ['verifyCode', 'captcha'],  
    ];  
}
```

Así que tenemos el namespace, las sentencias use, las propiedades de clase, y el primer método, rules.

Noten que tenemos un atributo verifyCode. Usaremos el validador captcha con este atributo. Si recuerda, la clase captcha fue configurada en el controlador a través del método actions, así que está disponible como una acción del controlador. Genial.

Captcha

Tomemos un minuto para discutir captcha en detalle, ya que es una característica muy útil de incluir en sus aplicaciones. La buena noticia es que Yii 2 hace que sea muy fácil de implementar.

Usarla en SiteController es el comportamiento por defecto, así que hay un poco más sobre ella si la utiliza en un controlador diferente. Sin embargo, es muy simple.

Si necesita implementar captcha en cualquier lugar en sus futuras aplicaciones, existen pasos para hacerlo:

Paso 1. Configurar captcha en el nuevo controlador a través del método actions.

```

public function actions()
{
    return [
        'error' => [
            'class' => 'yii\web\ErrorAction',
        ],
        'captcha' => [
            'class' => 'yii\captcha\CaptchaAction',
            'fixedVerifyCode' => YII_ENV_TEST ? 'testme' : null,
        ],
    ];
}

```

Paso 2. Incluir un validador captcha en las reglas del modelo form con un parámetro extra para controller/captcha. Esto no es necesario para usarla con SiteController, pero en todos los otros casos, debemos definirlo.

Por ejemplo digamos que queremos usarlo en la acción contact de PagesController en lugar de SiteController. Imaginemos que usamos el modelo ContactForm y modificamos el método rules. Lucirá así:

```
[ 'verifyCode', 'captcha', 'captchaAction' => 'pages/captcha' ]
```

Nota: Asegurarse de que hay una propiedad \$verifyCode coincidente en la clase del modelo de formulario. Vea el modelo real ContactForm como ejemplo.

Paso 3. Incluya el widget en la vista con el mismo parámetro 'captchaAction' en su configuración:

```

<?= $form->field($model, 'verifyCode')->widget(Captcha::className(), [
    'captchaAction' => 'pages/captcha',
    'template' => '<div class="row"><div class="col-lg-3">
        {image}</div><div class="col-lg-6">{input}</div></div>',
]) ?>

```

Paso 4. Incluye una regla access en el método behaviors para permitir captcha en el controlador:

```
public function behaviors()
{
    return [
        'access' => [
            'class' => AccessControl::className(),
            'only' => ['captcha'],
            'rules' => [
                [
                    'actions' => ['captcha'],
                    'allow' => true,
                    'roles' => ['?', '@'],
                ],
            ],
        ],
    ];
}
```

No es necesario agregar la regla access en SiteController, pero la necesitaremos en cualquier otro controlador.

Paso 5. Limpiar la cache. Puede ser necesario limpiar la cache del navegador luego de realizar los cambios.

Pueden parecer muchos pasos, pero en realidad es muy fácil de implementar. No sé de ningún otro framework PHP que lo haga tan fácil. Y no se preocupe si inicialmente no lo ha memorizado, yo tampoco tengo memoria fotográfica. Sólo úselo como referencia y sepá que está allí para usted.

De cualquier modo, nos hemos adelantado, no vamos a mirar la vista aún, además tenemos más cosas geniales sucediendo en este momento.

Primero continuemos con el modelo ContactForm. El método attributeLabels sigue a continuación:

```
public function attributeLabels()
{
    return [
        'verifyCode' => 'Verification Code',
    ];
}
```

Esta es la etiqueta que aparecerá en el formulario de la vista. No se necesitan más explicaciones.

Y finalmente, el método sendEmail:

```
public function sendEmail($email)
{
    return Yii::$app->mailer->compose()
        ->setTo($email)
        ->setFrom([$this->email => $this->name])
        ->setSubject($this->subject)
        ->setTextBody($this->body)
        ->send();
}
```

Este llama al método compuesto del mailer configurado en la aplicación que debería ser Swiftmailer. Cuando está en modo desarrollo, que es cómo establecimos nuestra aplicación durante init, los emails serán enviados a frontend/runtime/mail.

Nota: El directorio no existirá hasta que haya creado un registro, lo que puede hacer probando el formulario de contacto. Así que puede probar el formulario de contacto para ver si agrega un email allí. Por más información en las opciones de configuración de Swiftmailer, puede revisar la guía de Yii 2:

[Guía de Email de Yii 2](#)

Y eso es todo para nuestro modelo ContactForm. Regresemos ahora a SiteController y su método actionContact. Así que llamamos a la instancia de ContactForm y la asignamos a \$model. Luego tenemos:

```
if ($model->load(Yii::$app->request->post()) && $model->validate()) {

    if ($model->sendEmail(Yii::$app->params['adminEmail'])) {
        Yii::$app->session->setFlash('success',
            'Thank you for contacting us. We will respond to you as soon as possible.');

    } else {
        Yii::$app->session->setFlash('error', 'There was an error sending email.');
    }

    return $this->refresh();
}

} else {

    return $this->render('contact', [
        'model' => $model,
    ]);
}
```

Así que, si obtenemos los datos del post y los procesamos y validamos a través del modelo, y si el email puede ser enviado, instruimos al método para enviar un mensaje flash. Un mensaje flash, que es texto que aparecerá en la vista, es enviado desde el controlador a la vista a través de la sesión. Así que tenemos el método setFlash. Es todo lo que necesita para enviar un mensaje flash, Yii 2 hará el resto.

Note que en cualquier circunstancia en este método, permanecerá en la página de contacto. En los casos donde se procesa el formulario, de forma exitosa o no, refresca la página, de otra forma muestra el formulario, que es lo mismo, pero sin los mensajes flash.

Vale la pena dar un vistazo a la vista ya que ocurren un par de cosas interesantes, incluyendo el uso de captcha como método de verificación.

La Vista del Formulario de Contacto

Así que, ¿qué tan genial es que el primer archivo de una vista que miramos en profundidad sea un formulario? Mi predicción es que se asombrará de cuán conciso es este archivo. Y por último, verá como funciona el ciclo MVC completo.

Bien, así que en el directorio frontend/views/site folder, tenemos contact.php:

```
<?php  
use yii\helpers\Html;  
use yii\bootstrap\ActiveForm;  
use yii\captcha\Captcha;  
  
/* @var $this yii\web\View */  
/* @var $form yii\bootstrap\ActiveForm */  
/* @var $model \frontend\models\ContactForm */  
  
$this->title = 'Contact';  
$this->params['breadcrumbs'][] = $this->title;  
?  
<div class="site-contact">  
    <h1><?= Html::encode($this->title) ?></h1>  
  
    <p>  
        If you have business inquiries or other questions, please fill out the  
        following form to contact us. Thank you.  
    </p>  
  
    <div class="row">  
        <div class="col-lg-5">
```

```

<?php $form = ActiveForm::begin(['id' => 'contact-form']); ?>
    <?= $form->field($model, 'name') ?>
    <?= $form->field($model, 'email') ?>
    <?= $form->field($model, 'subject') ?>
    <?= $form->field($model, 'body')->textArea(['rows' => 6]) ?>
    <?= $form->field($model, 'verifyCode')
        ->widget(Captcha::className(), [
            'template' => '<div class="row"><div class="col-lg-3">
                {image}</div><div class="col-lg-6">{input}</div></div>',
        ]) ?>
    <div class="form-group">
        <?= Html::submitButton('Submit', ['class' => 'btn btn-primary',
            'name' => 'contact-button']) ?>
    </div>
    <?php ActiveForm::end(); ?>
</div>
</div>

</div>

```

37 líneas, ¡eso es todo!

Puede ver que comenzamos con nuestras sentencias use y algunos comentarios que nos dicen qué variables estamos accediendo:

```

<?php
use yii\helpers\Html;
use yii\widgets\ActiveForm;
use yii\captcha\Captcha;

/* @var $this yii\web\View */
/* @var $form yii\widgets\ActiveForm */
/* @var $model \frontend\models\ContactForm */

```

Esto es práctico como referencia porque es fácil confundir entre \$this y \$model. Pero no se preocupe, puede mirar simplemente al principio del archivo.

Ahora obtenemos el título y breadcrumbs:

```

$this->title = 'Contact';
$this->params['breadcrumbs'][] = $this->title;
?>

```

Usamos el método params para enviar el parámetro breadcrumb al archivo de layout (frontend/views/layouts/main.php), que lo llama a través del widget Breadcrumbs, de modo que aparezca en la parte superior de la página.

También, note la etiqueta de cierre php debajo. En nuestros modelos y controladores, no usamos tags ?> de cierre. En nuestras vistas, no sólo usamos tags de cierre ?, sino que también tenemos que asegurarnos de que abrimos y cerramos php correctamente ya que estamos mezclandolo con HTML.

Luego tenemos un div, sin php:

```
<div class="site-contact">
```

Luego tenemos un <H1> con Php en él:

```
<h1><?= Html::encode($this->title) ?></h1>
```



Consejo

Soy un gran fanático de los tags php rojos, especialmente en las vistas. No puedo controlar ese color en este libro, pero es simplemente un consejo para su propia codificación. Ese color realmente ayuda a resaltar sobre el código Html.

También note el tag corto de apertura <?= . Esta es la versión abreviada de:

```
<?php echo
```

Así que en nuestro h1, estamos mostrando el título con echo, dentro del método Html::encode. Este convertirá los caracteres especiales en entidades html. Por esto es que necesitamos:

```
use yii\helpers\Html;
```

Luego tenemos un <p> con instrucciones sobre cómo completar el formulario. Este es simplemente formato Html plano.

Luego tenemos dos divs para nuestro widget form:

```
<div class="row">
    <div class="col-lg-5">
```

Y luego llamamos al propio ActiveForm widget:

```
<?php $form = ActiveForm::begin(['id' => 'contact-form']); ?>
```

Note que no podemos usar ActiveForm sin la sentencia use al principio del archivo:

```
use yii\bootstrap\ActiveForm;
```

En el arreglo de configuración del widget, establecemos 'id' => 'contact-form'. Esto le dice al formulario como usar el modelo ContactForm. Mirando el resto del código, no hay nada que indique la acción del formulario. ¿Cómo sabe a qué acción enviar los datos?

Esta es una de las asombrosas características de Yii 2. Sabemos que la ubicación del formulario es un archivo de vista llamado contact.php en un directorio de vistas llamado site. Por lo tanto sabe que debería enviar a la acción site/contact. Ni siquiera tiene que decirle dónde enviar el formulario. Y ya que hemos mencionado que establecimos la id del formulario a contact-form, tiene también el modelo, así que tiene todo lo que necesita para ensamblar todo detrás de bambalinas, incluyendo validación y procesamiento.

Así que ahora definimos los campos que queremos enviar:

```
<?= $form->field($model, 'name') ?>
<?= $form->field($model, 'email') ?>
<?= $form->field($model, 'subject') ?>
<?= $form->field($model, 'body')->textArea(['rows' => 6]) ?>
<?= $form->field($model, 'verifyCode')
    ->widget(Captcha::className(),
[
    'template' => '<div class="row"><div class="col-lg-3">
        {image}</div><div class="col-lg-6">{input}</div></div>',
])
?>
```

Nota: Como he mencionado anteriormente, SiteController es el controlador para captcha, así que en este caso, no especificamos captcha' ⇒ 'site/captcha', pero en todos los demás usos, necesitará especificar el controller/captcha como en el ejemplo anterior.



Consejo

Simplemente otro recordatorio, necesita incluir el tag de cierre ?> en las vistas.

En el último campo, nos ponemos un poco mañosos, ponemos un widget en el campo. En este caso, se trata del widget Captcha que hemos estado discutiendo. Hay otros widgets que usaremos en el futuro, tales como dropDownList y DatePicker, que también son fáciles de usar.

Finalmente, agregamos las divs para el botón y el final del formulario:

```
<div class="form-group">
<?= Html::submitButton('Submit', ['class' => 'btn btn-primary',
                                'name' => 'contact-button']) ?>
</div>
<?php ActiveForm::end(); ?>
</div>
</div>
```

No se como se siente sobre eso, pero para mi, es tan simple, que inspira. Toda la validación, todas esas complicadas expresiones regex, son manejadas sin esfuerzo. Si trabajar en la validación del formulario era su pasatiempo favorito, es tiempo de encontrar uno nuevo.

Bien, lo crea o no, estamos aún en un capítulo sobre el controlador del sitio. Así que volvamos a eso ahora.

Acción About

El siguiente método es actionAbout:

```
public function actionAbout()
{
    return $this->render('about');
}
```

Obviamente, todo está en el archivo de la vista y la mayoría es sólo texto.

Acción Signup

Continuemos con actionSignup:

```

public function actionSignup()
{
    $model = new SignupForm();
    if ($model->load(Yii::$app->request->post())) {
        if ($user = $model->signup()) {
            if (Yii::$app->getUser()->login($user)) {
                return $this->goHome();
            }
        }
    }

    return $this->render('signup', [
        'model' => $model,
    ]);
}

```

Aquí vamos, nuevo modelo de formulario, SignupForm. Sólo una nota rápida. Revisé el archivo de vista signup.php y la id del widget formulario fue establecida a form-signup, lo que creo que es un error de tipeo. La convención parece ser la inversa, así que simplemente para comprobar si funcionaría, la cambié a signup-form, y con seguridad, siguió funcionando. Así que lo más probable es que Yii 2 no se preocupe por el orden en que las pone. Yo, por otra parte, me pongo nervioso con cosas como esas, así que continúo con la manera en que lo hago, que es poner la palabra ‘form’ en segundo lugar. Esto también es una indicación de que al crear un modelo de formulario, debería mantener la convención de nombres ExampleForm, por ejemplo.

Modelo Signup Form

Observamos brevemente el modelo SignupForm en el Modelo User en el capítulo Modificando el Modelo de Usuario, pero miremoslo con mayor detalle ahora. Comenzaremos con el namespace, sentencias use, declaración de clase, y atributos:

```

<?php
namespace frontend\models;

use common\models\User;
use yii\base\Model;
use Yii;

/**
 * Signup form

```

```
/*
class SignupForm extends Model
{
    public $username;
    public $email;
    public $password;
```

Nada que no hayamos visto antes. Observemos el método rules:

```
public function rules()
{
    return [
        ['username', 'filter', 'filter' => 'trim'],
        ['username', 'required'],
        ['username', 'unique',
            'targetClass' => '\common\models\User',
            'message' => 'This username has already been taken.'],
        ['username', 'string', 'min' => 2, 'max' => 255],

        ['email', 'filter', 'filter' => 'trim'],
        ['email', 'required'],
        ['email', 'email'],
        ['email', 'unique',
            'targetClass' => '\common\models\User',
            'message' => 'This email address has already been taken.'],

        ['password', 'required'],
        ['password', 'string', 'min' => 6],
    ];
}
```

Bien, tenemos algunas reglas para quitar el espacio en blanco, hacer obligatorios los campos, y mostrarnos si el username o email ya ha sido usado, ejemplos del validador unique. Note que en las reglas email y user unique, lista una clase objetivo, junto con un mensaje de respuesta. Esta es la única oportunidad en que he visto esto y no pude encontrar nada en la documentación sobre ello, así que adivinaré y diré que necesita saber que modelo usar para la validación unique, ya que tiene que realizar una consulta a la base de datos para ejecutar el validador.

Luego tenemos el método signup:

```

public function signup()
{
    if ($this->validate()) {
        $user = new User();
        $user->username = $this->username;
        $user->email = $this->email;
        $user->setPassword($this->password);
        $user->generateAuthKey();
        if ($user->save()) {
            return $user;
        }
    }

    return null;
}

```

Este es bastante simple de entender. Llama a una instancia del modelo User, luego usa los métodos del modelo para crear el usuario, siempre y cuando la validación haya pasado.

Así que una cosa a notar sobre los controladores es que pueden referenciar muchos modelos diferentes. Hasta ahora el controlador del sitio ha usado 3 modelos de formulario diferentes y el modelo usuario.

De cualquier manera, regresemos al método actionSignup en el controlador. Aquí está nuevamente como referencia:

```

public function actionSignup()
{
    $model = new SignupForm();
    if ($model->load(Yii::$app->request->post())) {
        if ($user = $model->signup()) {
            if (Yii::$app->getUser()->login($user)) {
                return $this->goHome();
            }
        }
    }

    return $this->render('signup', [
        'model' => $model,
    ]);
}

```

En este punto, estamos bastante familiarizados con la forma en que Yii 2 carga los datos enviados via post. Luego ejecuta el método signup en el modelo, que creará el usuario. Luego encuentra una instancia del usuario y lo loguea. Finalmente, lo devuelve a la página de inicio.

Si falla o no hay datos de post, muestra el formulario signup.

Los dos últimos métodos del controlador del sitio tratan con el restablecimiento de contraseña. El primero es actionRequestPasswordReset y llama al modelo PasswordResetRequestForm.

Es interesante notar que la id en el formulario de la vista es

```
<?php $form = ActiveForm::begin(['id' => 'request-password-reset-form']); ?>
```

Así que puede ver que la convención de nombres sigue lo que esperaríamos.

Bien, de regreso al modelo, revisemoslo, es realmente bastante simple en realidad:

```
<?php

namespace frontend\models;

use common\models\User;
use yii\base\Model;

/**
 * Password reset request form
 */

class PasswordResetRequestForm extends Model
{
    public $email;

    /**
     * @inheritdoc
     */
}

public function rules()
{
    return [
        ['email', 'filter', 'filter' => 'trim'],
        ['email', 'required'],
        ['email', 'email'],
        ['email', 'exist',
            'targetClass' => '\common\models\User',
            'filter' => ['estado' => User::ESTADO_ACTIVO],
            'message' => 'There is no user with such email.'
    ];
}
```

```

],
];
}

```

El único atributo aquí es \$email. Note en las reglas en el arreglo existen tenemos ‘targetClass’, ‘filter’, y ‘message’. Esto nos muestra cuán sofisticada puede ser la **validación**. También vemos que se referencia a la única constante que dejé en su lugar en el modelo User. Así que si en el futuro, queremos reemplazarla, tenemos que hacerlo aquí al igual que en el modelo. Pero espere un minuto. Cambiamos el atributo de status a estado_id, así que tendremos que cambiar eso aquí. Continúe y haga el cambio ahora.

Esa línea debería verse así ahora:

```
'filter' => ['estado_id' => User::ESTADO_ACTIVO],
```

Ahora tenemos sólo otro método más, sendEmail:

```

public function sendEmail()
{
    /* @var $user User */
$user = User::findOne([
    'status' => User::STATUS_ACTIVE,
    'email' => $this->email,
]);

if ($user) {
    if (!User::isPasswordResetTokenValid($user->password_reset_token)) {
        $user->generatePasswordResetToken();
    }

    if ($user->save()) {
        return \Yii::$app->mailer->compose(['html' => 'passwordResetToken-html',
        'text' => 'passwordResetToken-text'],
        ['user' => $user])
            ->setFrom([\Yii::$app->params['supportEmail'] => \Yii::$app->name . ' robot'])
            ->setTo($this->email)
            ->setSubject('Password reset for ' . \Yii::$app->name)
            ->send();
    }
}

return false;
}

```

También necesitamos hacer el cambio del nombre de atributo aquí. Continúe y cambie ‘status’ a ‘estado_id’. Debería verse así:

```
$user = User::findOne([
    'estado_id' => User::ESTADO_ACTIVO,
    'email' => $this->email,
]);
```

El método sendemail busca al usuario por su dirección de email para ver si está activo. Si está bien, comprobamos si existe un token válido, y si no generamos uno. Si podemos guardarlo, enviamos el token en el email. De otra manera devolvemos falso.

Así que regresemos al controlador del sitio y el método actionRequestPasswordReset. Así que luego de intentar enviar los datos y validar, trata de enviar el email, y si es válido, establece el mensaje flash de éxito. Si ha pasado la validación pero por alguna razón el email no ha podido ser enviado, muestra un mensaje de error.

Si los datos no son enviados, muestra el formulario.

Bien, una acción más del controlador del sitio, actionResetPassword(\$token). Este requiere la variable get de la url del token:

```
public function actionResetPassword($token)
{
    try {
        $model = new ResetPasswordForm($token);
    } catch (InvalidArgumentException $e) {
        throw new BadRequestHttpException($e->getMessage());
    }

    if ($model->load(Yii::$app->request->post())
        && $model->validate() && $model->resetPassword()) {
        Yii::$app->getSession()->setFlash('success', 'New password was saved.');

        return $this->goHome();
    }

    return $this->render('resetPassword', [
        'model' => $model,
    ]);
}
```

No hay sorpresas, tenemos otro modelo de formulario. Ahora porque aguardamos del token de la variable get, encerramos la llamada al modelo en un bloque try catch para que podamos manejar el error si no obtenemos el token esperado.

ResetPasswordForm Model

Observemos el modelo ResetPasswordForm:

```
<?php  
namespace frontend\models;  
  
use common\models\User;  
use yii\base\InvalidArgumentException;  
use yii\base\Model;  
use Yii;  
  
/**  
 * Password reset form  
 */  
  
class ResetPasswordForm extends Model  
{  
  
    public $password;  
  
    /**  
     * @var \common\models\User  
     */  
  
    private $_user;
```

Vemos el namespace, sentencias use, declaración de clases y propiedades de clase. Hay un comentario que nos dice que haremos referencia al modelo User. Luego tenemos un constructor que toma dos argumentos, el token y un \$config que tiene un valor por defecto de arreglo vacío. Estoy bastante seguro de que el arreglo vacío está allí debido al constructor de la clase padre de Model, la clase que está siendo extendida.

```
/*
 * Creates a form model given a token.
 *
 * @param string $token
 * @param array $config name-value pairs that will be used to initialize
 * the object properties
 * @throws \yii\base\InvalidParamException if token is empty or not valid
 * avoiding line-wrap in function. do not breakup lines in your code.
 */

public function __construct($token, $config = [])
{
    if (empty($token) || !is_string($token)) {
        throw new InvalidParamException('Password reset token cannot be blank.');
    }
    $this->_user = User::findByPasswordResetToken($token);
    if (!$this->_user) {
        throw new InvalidParamException('Wrong password reset token.');
    }
    parent::__construct($config);
}
```

Nuevamente obviamente, estamos evitando el ajuste de línea, así que hay saltos de línea donde normalmente no los habría. Hace que el código se vea descuidado pero no hay nada que pueda hacer sobre eso. Así que continuemos.

Así que las dos cosas principales que hace el constructor son. 1. Comprueba si el token está vacío o no es una cadena. 2. Hace una llamada User::findByPasswordResetToken(\$token) para asignar el usuario a \$_user.

Si recuerda cuando vimos el modelo User en detalle, vimos el método findByPasswordResetToken(\$token).

El constructor finaliza llamando al padre.

Luego tenemos el método rules:

```
public function rules()
{
    return [
        ['password', 'required'],
        ['password', 'string', 'min' => 6],
    ];
}
```

Fácil de comprender.

Terminamos el modelo con el método resetPassword:

```
/*
 * Resets password.
 *
 * @return boolean if password was reset.
 */

public function resetPassword()
{
    $user = $this->_user;
    $user->password = $this->password;
    $user->removePasswordResetToken();

    return $user->save();
}
```

Esto es bastante fácil de entender. La contraseña será establecida por los datos enviados con los que el controlador alimentará al modelo. Luego removerá el token.

Bien, así que para cerrar nuestro controlador el método actionResetPassword:

```
if ($model->load(Yii::$app->request->post()) && $model->validate()
    && $model->resetPassword()) {

    Yii::$app->getSession()->setFlash('success', 'New password was saved.');

    return $this->goHome();
}

return $this->render('resetPassword', [
```

```
'model' => $model,  
]);  
}
```

Enviar los datos al modelo, restablecer la contraseña e ir al inicio, o mostrar el formulario de la vista. Obviamente Site Controller cubrió mucho terreno, pero aún tenemos un poco más. Ese fue el frontend.

Controlador de Sitio del Backend

Tenemos un controlador de sitio en backend/controllers/SiteController.php. Este sin embargo es bastante más pequeño. Veamoslo.

```
<?php  
namespace backend\controllers;  
  
use Yii;  
use yii\filters\AccessControl;  
use yii\web\Controller;  
use common\models\LoginForm;  
use yii\filters\VerbFilter;  
  
/**  
 * Site controller  
 */  
  
class SiteController extends Controller  
{
```

Estamos tan acostumbrados a esto en este punto, que no necesitamos realmente comentar aquí, más que señalar que estaremos usando el mismo modelo LoginForm que en el frontend.

Siguiente método, behaviors:

```

public function behaviors()
{
    return [
        'access' => [
            'class' => AccessControl::className(),
            'rules' => [
                [
                    'actions' => ['login', 'error'],
                    'allow' => true,
                ],
                [
                    'actions' => ['logout', 'index'],
                    'allow' => true,
                    'roles' => ['@'],
                ],
            ],
        ],
        'verbs' => [
            'class' => VerbFilter::className(),
            'actions' => [
                'logout' => ['post'],
            ],
        ],
    ];
}

```

Nada nuevo que discutir aquí.

Ahora nos movemos al método actions, configurandolo sólo para error, ya que no necesitamos captcha:

```

public function actions()
{
    return [
        'error' => [
            'class' => 'yii\web\ErrorAction',
        ],
    ];
}

```

Then actionIndex:

```
public function actionIndex()
{
    return $this->render('index');
}
```

Super simple como lo fue el frontend.

Ahora viene actionLogin:

```
public function actionLogin()
{
    if (!\Yii::$app->user->isGuest) {
        return $this->goHome();
    }

    $model = new LoginForm();
    if ($model->load(Yii::$app->request->post()) && $model->login()) {
        return $this->goBack();
    } else {
        return $this->render('login', [
            'model' => $model,
        ]);
    }
}
```

Nuevamente, igual que frontend.

Y finalmente, actionLogout:

```
public function actionLogout()
{
    Yii::$app->user->logout();

    return $this->goHome();
}
```

No se necesita explicación en este punto (eso espero).

Comenzando el Control de Acceso

Así que finalmente, enterrado profundamente al final de un largo capítulo, una suerte de gran visita, comenzamos a realizar algunos cambios e influenciar el comportamiento de nuestra aplicación. Vamos a comenzar con un cambio muy pequeño al controlador del sitio del backend.

No necesitará un Gist, sólo un cambio de una palabra. Modifique actionLogin con lo siguiente:

```

public function actionLogin()
{
    if (!\Yii::$app->user->isGuest) {
        return $this->goHome();
    }

    $model = new LoginForm();
    if ($model->load(Yii::$app->request->post()) && $model->loginAdmin()) {
        return $this->goBack();
    } else {
        return $this->render('login', [
            'model' => $model,
        ]);
    }
}

```

Sólo hay un pequeño cambio. En lugar de llamar a login(), llamamos a loginAdmin() desde el modelo LoginForm.

Método loginAdmin

No hemos creado este método, así que continuemos y hagámoslo ahora. Inserte el siguiente método en common/models/LoginForm.php.

Gist:

[LoginAdmin](#)

Del libro:

```

public function loginAdmin()
{
    if (($this->validate())
        && PermisosHelpers::requerirMinimoRol('Admin',
        $this->getUser()->id)) {

        return Yii::$app->user->login($this->getUser(),
            $this->rememberMe ? 3600 * 24 * 30 : 0);

    } else {

        throw new NotFoundHttpException('No Pasarás.');
    }
}

```

```
}
```

```
}
```

Por favor mire el Gist si desea ver el formato apropiado, se puso muy desagradable debido a que tuve que evitar el ajuste de línea en el libro.

También añada al principio del archivo:

```
use yii\web\NotFoundHttpException;
use common\models\PermisosHelpers;
```

Así que lo que hemos hecho aquí es agregar más a la sentencia if. No sólo validamos el usuario, sino que también usamos:

```
PermisosHelpers::requerirMinimoRol('Admin', $this->getUser()->id)
```

Esto asegurará que el usuario tenga un rol mínimo de admin. La razón por la que ingresamos \$this->getUser()->id es porque el usuario no está logueado aún, así que necesitamos decirle explícitamente que usuario queremos verificar. Si recuerda, tuvimos en cuenta este escenario en las clases helper.

No necesitamos comprobar el estado en este punto porque el método getUser llama al método findByUsername y ese método lo hace por nosotros.

Con un cambio relativamente menor, ahora controlamos el acceso al logueo de admin al requerir que el usuario tenga al menos un rol de admin y un estado activo. ¡Y mire lo simple que fue!

Nuestros helpers fueron de gran utilidad y con esos métodos, no necesitamos código verboso para obtener los valores que necesitamos.

Así que ahora puede jugar con esto registrando usuarios y estableciendo su rol_id mediante PhpMyadmin. Haga que algunos de ellos tengan el rol Usuario y otros tengan el rol Admin e inicie y termine sesión tanto en el frontend como el backend. Es bastante genial.

Antes de terminar el capítulo, podemos realizar un poco de limpieza. Discutamos la constante de la tabla user:

```
const ESTADO_ACTIVO = 1;
```

Como he dicho en el capítulo cuatro, dejar la constante viola el principio DRY. Pero por facilidad de configuración, la dejaré en su lugar.

Ahora que tenemos nuestras clases helper en su lugar, podemos reemplazarla. Obviamente esto asume que tiene un registro con estado Activo en su tabla estado.

Si desea reemplazar la constante, puede hacerlo al encontrar cada instancia de:

```
'estado_id' => self::ESTADO_ACTIVO
```

Y luego reemplazarla por:

```
'estado_id' => ValorHelpers::getEstadoId('Activo')
```

Simplemente sustituyla por la constante en los lugares apropiados en el modelo User y en el modelo PasswordResetRequestForm y no olvide incluir sentencias use para ValorHelpers.

Luego puede remover la constante completamente. Si remueve la constante y obtiene un error, eso significa que se le olvidó una referencia en alguno de los modelos, así que regrese y encuentre la referencia faltante.

Resumen



Hemos cubierto mucho terreno al examinar en detalle a Site Controller. Aprendimos mucho sobre controladores, acciones, vistas, y aún sobre modelos de formularios. Ahora sabemos, habiendo visto tantos ejemplos, que la implementación típica de un formulario puede involucrar una vista de formulario, un modelo de formulario y un controlador.

Aprendimos sobre el widget ActiveForm, que reside en la vista y nos facilita mostrar el formulario. Aprendimos cómo implementar captcha y enviar mensajes flash desde el controlador a la vista.

También aprendimos sobre algunas de las acciones más interesantes, como resetPassword y sus modelos de formulario asociados. Vimos como podemos enfocarnos en la clase user en las reglas de validación.

Finalmente, pudimos implementar un par de cambios que ahora controla el acceso al backend al forzar un valor mínimo para el rol del usuario. Hicimos eso simplemente usando los métodos Helper

del capítulo anterior, así que pareció un cambio trivial, y sin embargo fue el comienzo de nuestra construcción de un sistema de control a través de la aplicación.

Hemos tomado tiempo para explicar todo con tanto detalle como sea posible. Con suerte esto está grabándosele. Si no, dele tiempo, lo hará. Yii 2 requiere paciencia y persistencia para ser aprendido. Finalmente estamos en camino a construir la aplicación. ¡Y así continuamos!

Capítulo Ocho: Crud de Perfil

Muy bien, luego de mucha paciencia aprendiendo lo que Yii 2 nos ofrece de manera predeterminada, estamos listos para expandir nuestra aplicación y crear algo. Vamos a comenzar a crear el código que le permita a los usuarios crear un perfil.

Los Perfiles serán únicos para cada usuario y un usuario solamente deberá ver su perfil. Esto requerirá cambios a lo que Gii creará para nosotros, pero por supuesto continuaremos usando Gii para que nos ayude a comenzar creando el CRUD de Perfil.

CRUD

CRUD significa crear, leer, actualizar y modificar (create, read, update and delete), un acrónimo simple de recordar. Cuando usa Gii para crear CRUD, obtiene un controlador, formularios, y un modelo de búsqueda. ¡Es fantástico!

Así que naveguemos nuevamente a Gii:

`yii2build.com/index.php?r=gii`

The screenshot shows the Gii CRUD Generator interface. On the left, a sidebar lists generators: Model Generator, CRUD Generator (selected), Controller Generator, Form Generator, Module Generator, and Extension Generator. The main area is titled "CRUD Generator" and contains the following fields:

- Model Class:** frontend\models\Perfil
- Search Model Class:** frontend\models\search\PerfilSearch
- Controller Class:** frontend\controllers\PerfilController
- View Path:** (empty)
- Base Controller Class:** yii\web\Controller
- Widget Used in Index Page:** GridView
- Enable I18N:** (checkbox)
- Code Template:** default (C:\var\www\yii2build\vendor\yiisoft\yii2-gii\generators\crud/default)

A "Preview" button is at the bottom.

CRUD de Gii

Puede ver qué convenciones de nombre seguir para los nombres de archivo. Necesita crear un directorio search dentro de frontend/models antes de usar Gii para crear el CRUD, así que asegúrese de que el directorio llamado search esté allí antes de proceder .

Ya hemos decidido poner nuestro modelo Perfil en el directorio frontend, así que haremos lo mismo para el crud. Asegúrese de que completa los campos de esta manera:

Model Class es frontend\models\Perfil

Search Class es frontend\models\search\PerfilSearch

Controller Class es frontend\controllers\PerfilController

Puede dejar vacío view path si está usando los valores por defecto en la estructura de directorios, que es lo que estaremos haciendo.

Simplemente proveemos a Gii con un poco de información y en esta ocasión creará 8 archivos para nosotros:

- PerfilController.php
- PerfilSearch.php
- views/perfil/_form.php
- views/perfil/_search.php
- views/perfil/create.php

- views/perfil/index.php
- views/perfil/update.php
- views/perfil/view.php

Luego de que haya creado el directorio search en frontend/models, continúe y ejecute el generador de CRUD para Perfil. 8 archivos. Discutamos brevemente lo que hace cada uno.

Controlador de Perfil

Este es PerfilController.php y se encuentra bajo frontend/controllers/PerfilController.php. Vamos a observarlo con gran detalle en unos pocos minutos.

Búsqueda de Perfil

PerfilSearch.php se encuentra en frontend/models/search/PerfilSearch.php y contiene la lógica que es utilizada por el formulario de búsqueda. Ésta es básicamente una extensión del modelo Perfil base que se utiliza para búsqueda. Debido a que no buscaremos en los perfiles en el frontend, no la usaremos. Los Usuarios tendrán un único perfil y por lo tanto no hay necesidad de buscar. Lo incluí aquí porque quiero asegurarme de mostrarle cómo crear este tipo de archivo. Usaremos este tipo de archivo más adelante en el backend.

_search

El propio formulario de búsqueda es un archivo parcial llamado _search.php que es renderizado por la página index de Perfil. Ya que no vamos a permitir a otros usuarios buscar perfiles, no lo usaremos.

_form

_form.php es otra vista parcial que contiene el formulario que es renderizado para las vistas create y update. El mismo formulario parcial es renderizado tanto en la página create como en update.

Index

Index.php es el archivo que incluye una widget ya preparado para mostrar resultados paginados en columnas organizadas, junto con la vista parcial _search.php sobre el formulario. Aunque no la necesitaremos para los usuarios del frontend, quienes no tendrán más que un único perfil, la

necesitaremos para los usuarios administradores del backend que pueden revisar los perfiles de todos los usuarios. Sólo para ser perfectamente claros, no usaremos este archivo en lo absoluto, pero usaremos uno similar para el backend. Lo dejaremos en su lugar como demostración.

Una vez que el archivo es construido mediante Gii, puede verlo en:

```
yii2build.com/index.php?r=perfil/index
```

View

El archivo view.php contiene los detalles de un registro individual y utiliza el widget DetailView. Esta página devolverá un error 400 ya que no le estamos pasando un id, que el controlador espera. Simplemente ignore esto por ahora, ya que vamos a modificarlo de cualquier manera.

Create

Llamar a create.php renderiza el parcial _form.php para poder ingresar los datos necesarios para crear el registro.

Una vez que el archivo ha sido construido mediante Gii, puede ver esta página en:

```
yii2build.com/index.php?r=perfil/create
```

Aquí hay una imagen de la página renderizada por la dirección de arriba:

Create Perfil

User ID

Nombre

Apellido

Fecha Nacimiento

Género ID

Created At

Updated At

[Create](#)

[Crear Perfil](#)

Aunque este formulario funcionaría si formateara la entrada apropiadamente, no recomiendo que lo pruebe ahora. Vamos a realizar profundos cambios al controlador y otros cambios al formulario, así que no tiene sentido probarlo ahora.

Update

Obviamente, esta página también falla debido a que espera una id como parámetro.

Cuando observe la vista de la página create, notará que existen muchos problemas. Campos como created_at se muestran como campos de texto y género devuelve un número.

Como he mencionado, si prueba algunas de estas url sin crear un perfil, obtendrá errores. Puede crear un perfil de manera forzada, pero no es lo que desea. Tenemos mucho trabajo por hacer antes de que esta IU esté lista.

También, debido a que no hemos ensamblado la navegación correcta, el formulario no sabe qué usuario es el correcto para asociar con el perfil. Ingresarlo manualmente expone los tópicos de

seguridad y manipulación de url, debido a que aún si usted asocia el registro correcto de manera manual, tiene que prevenir que los usuarios secuestren registros que no poseen.

Necesitamos hacer un poco de trabajo en el controlador y las vistas para generar exactamente lo que necesitamos, la generación de código sólo nos lleva hasta cierto punto.

Las buenas noticias son que aún en el estado en que están las cosas, Gii no ha provisto con mucho de lo que necesitaremos. También nos ha dado una arquitectura fácil de seguir, que, con algunas modificaciones de nuestra parte, creará una plantilla robusta para registros que pertenecen a los usuarios.

Así que, para cualesquiera requisitos que pueda tener en el futuro, cuando involucren registros que pertenecen a usuario, tales como perfil, preferencias, entradas de blog, etc., tendrá un ejemplo funcional que es fácilmente replicable. Y una vez que entre en ritmo, se asombrará con cómo fluye todo.

Al comenzar a usar un registro que pertenece a un usuario tal como perfil, estamos creando una estructura que Gii no nos entrega en perfectas condiciones, esas son las malas noticias. Las buenas noticias son que cuando construyamos la versión del backend de perfil, para ayudar en la administración de perfiles, será una aproximación más cercana, así que las cosas se volverán progresivamente más fáciles a medida que avancemos.

Modificando el Controlador y las Vistas de Perfil

Muy bien, atendamos nuestras necesidades. Podemos crear un perfil para el usuario, pero el formulario no se ve bien y no hay navegación hacia él. También, no existe control de acceso sobre el CRUD, así que el perfil del usuario, que debería ser privado, está completamente abierto y puede ser manipulado por cualquiera.

Vamos a hacer lo siguiente para corregirlo:

1. Modificaremos el controlador para controlar el acceso a las acciones del CRUD. Por ejemplo, sólo el usuario que posee el perfil será capaz de crear, actualizar o modificar el registro de perfil.
2. Modificaremos las vistas, para que tengan listas desplegables cuando corresponda. También removeremos campos innecesarios. Obtendremos el nombre apropiado para el género en lugar de simplemente listar un número de id.
3. Modificaremos nuestra vista layout principal para tener navegación al perfil de usuario.

Cuando concluyamos esta sección, nuestra plantilla Yii2Build habrá dado un enorme paso hacia adelante.

De manera predeterminada, los usuarios serán capaces de registrarse e iniciar sesión en la aplicación. Ya hemos extendido la Plantilla de la Aplicación Avanzada para respetar la diferencia entre nivel de

rol para frontend y backend, de modo que sólo los usuarios con rol mínimo de admin puedan iniciar sesión en el backend.



Consejo

Simplemente por claridad, mencionaré que el CRUD de perfil del frontend no tiene nada que ver con el backend y no aparecerá allí. Lo construiremos de manera separada cuando creemos la sección de administración del backend.

Al construir nuestro modelo de perfil del frontend totalmente funcional, a los usuarios se les requerirá iniciar sesión para construir un perfil. La aplicación sabrá si ellos ya tienen un perfil o no, y si no lo tienen, cuando hagan clic en el link de perfil, los llevará a la página de creación de perfil. La aplicación también impondrá reglas para asegurarse de que el usuario sólo es capaz de ver su propia página de perfil. También proveeremos de navegación en las páginas de vista que sepa cuándo es apropiado mostrar el link a perfil.

Así que cuando terminemos con esta parte del proyecto, tendremos una plantilla ajustada y un ejemplo a seguir si desamos crear cualquier tipo de registros que pertenecen a usuarios que necesitan ser privados para ese usuario. Esto es muy bueno.

Modificando el Controlador de Perfil

Debemos comenzar incluyendo un par más de sentencias use, así que copie esto reemplazando las sentencias use del archivo PerfilController.php.

Gist:

[Sentencias Use de Perfil](#)

Del libro:

```
namespace frontend\controllers;

use Yii;
use frontend\models\Perfil;
use frontend\models\search\PerfilSearch;
use yii\web\Controller;
use yii\web\NotFoundHttpException;
use yii\filters\VerbFilter;
use common\models\PermisosHelpers;
use common\models\RegistrosHelpers;
```

Esto hace referencia a nuestra clase RegistrosHelpers y a nuestra clase PermisosHelpers que escribimos en un capítulo anterior.

Actualmente, el método behaviors sólo tiene una restricción sobre delete que dice que debe ser realizada con el método post:

```
public function behaviors()
{
    return [
        'verbs' => [
            'class' => VerbFilter::className(),
            'actions' => [
                'delete' => ['post'],
            ],
        ],
    ];
}
```

Queremos agregar algo de lógica de acceso de control que restrinja las acciones del controlador para el usuario a menos que haya iniciado sesión. Para ello cambie el método behaviors a este:

Gist:

Controller Behaviors

Del libro:

```
public function behaviors()
{
    return [
        'access' => [
            'class' => \yii\filters\AccessControl::className(),
            'only' => ['index', 'view', 'create', 'update', 'delete'],
            'rules' => [
                [
                    'actions' => ['index', 'view', 'create', 'update', 'delete'],
                    'allow' => true,
                    'roles' => ['@'],
                ],
            ],
        ],
        'verbs' => [
            'class' => VerbFilter::className(),
            'actions' => [
                'delete' => ['post'],
            ],
        ],
    ];
}
```

El símbolo @ significa logueado, así que las acciones listadas pueden ser realizadas solamente cuando el usuario haya iniciado sesión. Eso no es suficiente control de acceso, pero es un comienzo. Haremos más luego.

Por favor tenga en cuenta que en este caso roles no se refiere a la columna rol_id del registro de usuario, las dos no tienen nada en común.

Acción Index

Bien, ahora estamos listos para las acciones. La acción index, que es la acción por defecto del controlador, se ve así:

```
/**  
 * Lists all Perfil models.  
 * @return mixed  
 */  
  
public function actionIndex()  
{  
    $searchModel = new PerfilSearch();  
    $dataProvider = $searchModel->search(Yii::$app->request->queryParams);  
  
    return $this->render('index', [  
        'searchModel' => $searchModel,  
        'dataProvider' => $dataProvider,  
    ]);  
}
```

Este está pensado para devolver una lista de resultados y usa un modelo diferente, PerfilSearch, que extiende el modelo perfil para proveer de funcionalidad de búsqueda. Pero en el caso del perfil de usuario, sólo permitimos un perfil por usuario, así que no usaremos este código.

Podríamos simplemente deshabilitar esta acción, quitarla completamente del controlador, y borrar el archivo index.php, pero parte de la navegación del bread crumb que está establecido en Yii 2 devolverá una excepción de página no encontrada y ese no es el comportamiento deseado. También, alguien podría tippear r=perfil en la url y devolvería el mismo error de página no encontrada, nuevamente un comportamiento no deseado. Queremos que toda nuestra navegación esté super ajustada, así que en lugar de ello redigiremos index.php a view.php.

View.php, si lo recuerda, lista los detalles del registro, y esto sería apropiado para alguien que quisiera ver su propio perfil.

Por supuesto no es tan simple como redirigir. También tenemos que aplicar lógica en el controlador que determine si el usuario apropiado tiene un perfil, y en ese caso, mostrarlo, y en caso contrario, enviarlo a la página de creación.

Así que reemplazamos al método actionIndex con lo siguiente:

Gist:

Index Action

Del libro:

```
public function actionIndex()
{
    if ($ya_existe = RegistrosHelpers::userTiene('perfil')) {
        return $this->render('view', [
            'model' => $this->findModel($ya_existe),
        ]);
    } else {
        return $this->redirect(['create']);
    }
}
```

Esa primera línea debería serle familiar. Hablamos sobre hacer algo así cuando construimos el método userTiene de RegistrosHelpers. Ahora estamos listos para usarlo.

Simplemente un recordatorio rápido, el método userTiene verifica un registro de usuario en el modelo especificado, así que en este caso estamos verificando si el usuario tiene un registro de perfil. Si no hay un registro, devuelve falso, si no, devuelve la id del registro.

Describamos exactamente como funciona esto. La primera cosa que hacemos es llamar al método userTiene de la clase RegistrosHelpers y asignar el resultado a \$ya_existe, encerrado por una sentencia if.

Si userTiene evalúa a verdadero, devuelve el archivo de la vista, con la instancia correcta del modelo almacenada en la variable \$ya_existe.

Luego usa el método del controlador findModel para devolver esa instancia a la vista, en ese caso llamada view.php, siendo pasada en un arreglo:

```
'model' => $this->findModel($ya_existe),
```

Ahora si \$ya_existe evalúa a falso, redirigimos a la vista crear, ya que el usuario no tiene un perfil, y necesita crear uno:

```
} else {  
  
    return $this->redirect(['create']);  
  
}
```

Al poner esfuerzo extra en la creación de la clase auxiliar, extrajimos algo de la lógica del controlador manteniéndolo simple y limpio. Si quisieramos usar la sintaxis de relación de Yii en la sentencia if, podríamos escribir:

```
if($ya_existe = Perfil::find()->where(['user_id' => Yii::$app->user->identity->id])->one())
```

Personalmente, me gusta más esta sintaxis:

```
if ($ya_existe = RegistroHelpers::userTiene('perfil'))
```

Por la mayor parte, es una diferencia cosmética trivial, pero tenemos el potencial de usar esta sintaxis en muchas partes del proyecto. Podemos usarlo en otros modelos simplemente al pasarle un nombre diferente de modelo.

Como mencioné en un capítulo anterior sobre el libro de Robert C. Martin llamado Clean Code. Él trae el punto de que los programadores pasan la mayor parte del tiempo leyendo código, no escribiéndolo. Así que las pequeñas diferencias sintácticas que vuelven el código más simple de leer, hacen una enorme diferencia en la velocidad y claridad de aquellos que tienen que trabajar en el sistema.

Otro gran cambio en el camino de la construcción de nuestros métodos del controlador es que eliminamos la variable get pasada al método, así que la id del registro no puede ser secuestrada a través del navegador. Esto agrega una capa extra de seguridad al sistema. Estaremos añadiendo más seguridad luego.

Ahora que comprendemos como RegistroHelpers::userTiene('perfil') comprueba la asociación entre el registro del usuario y el de perfil, estamos listos para revisar el resto de las acciones.

Acción View

La acción view:

```

/**
 * Displays a single Perfil model.
 * @param string $id
 * @return mixed
 */

public function actionView($id)
{
    return $this->render('view', [
        'model' => $this->findModel($id),
    ]);
}

```

Cambiemos esto a:

Gist:

[ActionView](#)

Del libro:

```

public function actionView()
{
    if ($ya_existe = RegistrosHelpers::userTiene('perfil')) {

        return $this->render('view', [
            'model' => $this->findModel($ya_existe),
        ]);
    } else {

        return $this->redirect(['create']);
    }
}

```

Hey, esa es exactamente la misma que la acción index, ¿qué ocurre con ello? Recuerde que dijimos que estábamos cambiando la acción por defecto index para ser igual a la acción vista porque no necesitamos una lista de perfiles, sólo el perfil correcto para el usuario o el formulario de creación si no tiene uno. Así que tiene un descanso y podemos simplemente copiar el código.

Acción Create

Bien, veamos la acción Create. Aquí está lo que obtiene de manera predeterminada:

```
public function actionCreate()
{
    $model = new Perfil();

    if ($model->load(Yii::$app->request->post()) && $model->save()) {
        return $this->redirect(['view', 'id' => $model->id]); } else {
        return $this->render('create', [
            'model' => $model,
        ]);
    }
}
```

Básicamente esto dice si el formulario es cargado, salvarlo e ir a la vista o mostrar el formulario de creación. Bien, tenemos un poco más de nuestra versión para ella:

Gist:

[ActionCreate](#)

Del libro:

```
public function actionCreate()
{
    $model = new Perfil;

    $model->user_id = \Yii::$app->user->identity->id;

    if ($ya_existe = RegistrosHelpers::userTiene('perfil')) {

        return $this->render('view', [
            'model' => $this->findModel($ya_existe),
        ]);
    } elseif ($model->load(Yii::$app->request->post()) && $model->save()){

        return $this->redirect(['view']);
    }
}
```

```
    } else {  
  
        return $this->render('create', [  
  
            'model' => $model,  
  
        ]);  
    }  
}
```

Comenzamos llamando una nueva instancia del modelo, luego asignamos al atributo `user_id` del modelo el usuario actual mediante `Yii::$app->user->identity->id`. Como dije antes, la id del usuario actual está siempre disponible para nosotros a través de esta llamada estática a la clase de la aplicación Yii.

Luego comprobamos si el usuario ya tiene un perfil al ejecutar nuestro método `RegistrosHelpers::userTiene('perfil')` y asignar el resultado a `$ya_existe`.

```
if ($ya_existe = RegistrosHelpers::userTiene('perfil')) {  
    return $this->render('view', [  
        'model' => $this->findModel($ya_existe),  
    ]);  
}  
}
```

Si obtenemos una id de modelo como respuesta, mostramos el archivo de vista con esa id. Necesitamos poner esta prueba en este método porque un usuario puede ser capaz de navegar directamente a la acción create, sin pasar por index o view primero.

Si \$ya_existe evalúa a falso, la siguiente que hace el código es llamar al método load sobre los datos enviados e intentar guardarlos:

```
elseif ($model->load(Yii::$app->request->post()) && $model->save()) {  
  
    return $this->redirect(['view']);  
  
}
```

Pero esto sólo ocurrirá si el método load ha recibido datos desde el formulario que fue enviado. Si recibe y puede validar, lo guardará y devolverá la página view apropiada.

Puede estar preguntándose cómo sabe que user_id asignar al modelo recientemente creado, ya que no lo asignamos en el formulario, lo que significa que no es enviado via post. Es establecida en la segunda línea del método cuando asignamos el usuario actual a \$model->user_id. Y cuando \$model llama al método load, recuerda este atributo.

Finalmente, si no hay datos enviados o si hay errores de validación mostramos el formulario:

```

else {

    return $this->render('create', [
        'model' => $model,
    ]);
}

```

Acción Update

Luego continuamos con el método actionUpdate. Aquí está lo que nos entrega así Gii:

```

/**
 * Updates an existing Perfil model.
 * If update is successful, the browser will be redirected to the 'view' page.
 * @param string $id
 * @return mixed
 */

public function actionUpdate($id)
{
    $model = $this->findModel($id);

    if ($model->load(Yii::$app->request->post()) && $model->save()) {
        return $this->redirect(['view', 'id' => $model->id]);
    } else {
        return $this->render('update', [
            'model' => $model,
        ]);
    }
}

```

Necesitamos cambiar esto a:

Gist:

[ActionUpdate](#)

Del libro:

```


/**
 * Actualiza un modelo Perfil existente.
 * Si la actualización es exitosa, el navegador será redirigido a la página 'vie\w'.
 * @param string $id
 * @return mixed
 * Sentencia if en dos líneas para evitar ajuste de línea
 */

public function actionUpdate()
{
    if($model = Perfil::find()->where(['user_id' =>
        Yii::$app->user->identity->id])->one()) {

        if ($model->load(Yii::$app->request->post()) && $model->save()) {

            return $this->redirect(['view']);

        } else {

            return $this->render('update', [
                'model' => $model,
            ]);
        }
    } else {

        throw new NotFoundHttpException('No Existe el Perfil.');
    }
}


```

De inmediato vemos algo diferente porque he usado:

```
if($model = Perfil::find()->where(['user_id' => Yii::$app->user->identity->id])->one())
```

Lo hice porque sentí que ya que el nombre de la variable es \$model, era un poco más lógico desde el punto de vista sintáctico continuar con el nombre del modelo en el cual estamos interesados, en lugar de la clase auxiliar:

```
if($model = RegistrosHelpers::userTiene('perfil'))
```

La sintaxis más larga parece dejar más en claro que estamos buscando un registro de perfil que concuerda con el usuario actual. Bien, así que ahora sabemos porque usamos:

```
if($model = Perfil::find()->where(['user_id' => Yii::$app->user->identity->id])->one())
```

Así que analicemoslo parte por parte. Asignar a la variable \$model el modelo Perfil, donde user_id es el usuario, si puede. Si evalúa a falso, saltamos a la última sentencia if y lanzamos:

```
throw new NotFoundHttpException('No Existe en Perfil.');
```

De otra forma, si obtenemos nuestra variable \$model con la instancia correcta del modelo, llamamos a los datos desde el post y tratamos de salvarlos mediante la siguiente sentencia if.

```
if ($model->load(Yii::$app->request->post()) && $model->save()) {
    return $this->redirect(['view']);
}
```

Si la actualización es exitosa, ¿cómo sabe que vista mostrar? El método redirect, que en este caso sólo es suministrado con una acción, no es renderizado. Lo rutea a la acción nombrada del controlador actual, así que en este caso, la acción view determina la vista correcta a mostrar.

Por supuesto, esto también asume que no hay datos enviados.

Si no hay datos enviados, no hay cambios sobre lo que ya está allí, devolverá el formulario, con nuestro modelo para que pueda pre-cargarlo correctamente.

```
else {
    return $this->render('update', [
        'model' => $model,
    ]);
}
```

Una vez más, no se usaron variables get, así que no habrá secuestro de los registros a través de variables get.

Acción Delete

Nuestro cambio final a PerfilController estará en la acción delete. Obtuvimos esto de Gii:

```
public function actionDelete($id)
{
    $this->findModel($id)->delete();

    return $this->redirect(['index']);
}
```

Cambiémoslo a:

Gist:

[ActionDelete](#)

Del libro:

```
public function actionDelete()
{
    $model = Perfil::find()->where([
        'user_id' => Yii::$app->user->identity->id
    ])->one();

    $this->findModel($model->id)->delete();

    return $this->redirect(['site/index']);
}
```

Aquí establecemos \$model usando el mismo método de update por idénticas razones, ya que no tenemos una variable get. Luego usamos el método delete, pasándole \$model->id.

Luego redirigimos a la página index del sitio. Si simplemente ponemos ‘index’ como valor, el controlador asumirá que era ‘perfil/index’, que no es lo que desamos.

Acción FindModel

Al último método en el controlador Perfil, no lo modificaremos:

```
protected function findModel($id)
{
    if (($model = Perfil::findOne($id)) !== null) {
        return $model;
    } else {
        throw new NotFoundHttpException('The requested page does not exist.');
    }
}
```

Este es el método para encontrar una instancia particular del modelo. Usted entrega la \$id que está buscando y Yii devuelve la instancia del modelo, en verdad un método muy útil.

Una vez que podemos apreciar sobre Yii es que el código del controlador es muy limpio, claro y conciso. Todo el trabajo duro es abstraído fuera y simplemente proveemos un pequeño conjunto de instrucciones a ser seguidos por las acciones. Esto es PHP moderno en su punto más alto.

Modificando la Vistas de Perfil

Nuestro controlador de perfil funciona como deseamos, pero nuestros formularios tienen campos de fecha que no necesitamos y campos de texto en lugar de listas desplegables, así que tenemos que revisar nuestras vistas y arreglar esto.

Vamos a realizar bastantes cambios antes de probar todo. Esto está un poco fuera del flujo de trabajo del mundo real, pero por otra parte, no tendrá que ingresar registros manualmente con los que podría realizar pruebas que podrían generar errores, si olvida algo.

Así que simplemente continúe realizando estos cambios, y cuando hayamos terminado, seremos capaces de añadir un perfil al usuario a través de la IU y probarlo verdaderamente.

View.php

Bien, comencemos con view.php. Esto es lo que obtuvimos de Gii:

```
<?php

use yii\helpers\Html;
use yii\widgets\DetailView;

/* @var $this yii\web\View */
/* @var $model frontend\models\Perfil */

$this->title = $model->id;
```

```

$this->params['breadcrumbs'][] = ['label' => 'Perfiles', 'url' => ['index']];
$this->params['breadcrumbs'][] = $this->title;
?>
<div class="perfil-view">

    <h1><?= Html::encode($this->title) ?> Perfil</h1>

    <p>
        <?= Html::a('Update', ['update', 'id' => $model->id],
                    ['class' => 'btn btn-primary']) ?>
        <?= Html::a('Delete', ['delete', 'id' => $model->id], [
                    'class' => 'btn btn-danger',
                    'data' => [
                        'confirm' => 'Are you sure you want to delete this item?',
                        'method' => 'post',
                    ],
                ]) ?>
    </p>

    <?= DetailView::widget([
        'model' => $model,
        'attributes' => [
            'id',
            'user_id',
            'nombre:text',
            'apellido:text',
            'fecha_nacimiento',
            'genero_id',
            'created_at',
            'updated_at',
        ],
    ]) ?>

</div>

```

Comenzamos el archivo añadiendo una par de helpers:

```

use yii\helpers\Html;
use yii\widgets\DetailView;

```

aqui me quede 26 sep 2023

Esta va a ser nuestra primera vista en profundidad del widget DetailWidget, que veremos en un minuto. Luego tenemos algunos comentarios de Yii, \$this es la vista del modelo, y \$model es el

modelo perfil que ha sido pasado al controlador, así que representa la instancia exacta del modelo perfil que necesitamos.

```
/* @var $this yii\web\View */
/* @var $model frontend\models\Perfil */
```

Siempre puede referirse a esos comentarios si está confundido sobre qué variables representan a qué modelos.

Luego establecemos el título a \$model->id:

```
$this->title = $model->id;
```

Luego vienen los breadcrumbs en un formato simple de usar:

```
$this->params['breadcrumbs'][] = ['label' => 'Perfils', 'url' => ['index']];
$this->params['breadcrumbs'][] = $this->title;
```

\$this->params proviene de la clase base view. Cuando renderiza una vista, en este caso view.php, llama a una instancia de yii\webView, que extiende a la clase yii\base\view class, a través de la magia del enrutado de Yii 2, haciendo que el objeto esté disponible como \$this, de ahí el comentario:

```
/* @var $this yii\web\View */
```

Es una arquitectura muy poderosa. También entrega su modelo o modelos a través del controlador, así que tiene muchas posibilidades para manipular los datos en las vistas. Yii 2 hace todo esto, y al mismo tiempo, lo hace parecer simple, manteniendo fuera de la vista tanto código y lógica de PHP como sea posible.

Bien, continuemos. Establecemos la div, el h1, y algo de navegación:

```
<div class="perfil-view">

<h1><?= Html::encode($this->title) ?> Perfil</h1>

<p>
    <?= Html::a('Update', ['update', 'id' => $model->id],
                ['class' => 'btn btn-primary']) ?>
    <?= Html::a('Delete', ['delete', 'id' => $model->id], [
        'class' => 'btn btn-danger',
        'data' => [
            'confirm' => 'Are you sure you want to delete this item?',
            'method' => 'post',
        ],
    ]) ?>
</p>
```

El estilo del botón Delete es btn btn-danger y recibe los parámetros de datos adicionales de confirm y method. Como resultado de esto se obtiene un alert de confirmación, muy útil para la funcionalidad de borrado y forma parte de una gran interfaz, de manera predeterminada.

Finalmente tenemos los widgets detailview:

```
<?= DetailView::widget([
    'model' => $model,
    'attributes' => [
        'id',
        'user_id',
        'nombre:ntext',
        'apellido:ntext',
        'fecha_nacimiento',
        'genero_id',
        'created_at',
        'updated_at',
    ],
]) ?>

</div>
```

Puede agregar o quitar atributos fácilmente y se lo mostraremos con ejemplos. Continuemos y reemplazemos la vista completa con:

Gist:

Vista Perfil

Del libro:

```
<?php

use yii\helpers\Html;
use yii\widgets\DetailView;
use common\models\PermisosHelpers;

/**
 * @var yii\web\View $this
 * @var frontend\models\Perfil $model
 */

$this->title = "Perfil de " . $model->user->username;
$this->params['breadcrumbs'][] = ['label' => 'Perfiles', 'url' => ['index']];
```

```
$this->params['breadcrumbs'][] = $this->title;  
?  
<div class="perfil-view">  
  
<h1><?= Html::encode($this->title) ?></h1>  
  
<p>  
    <?Php  
  
    //esto no es necesario pero está aquí como ejemplo  
  
    if (PermisosHelpers::userDebeSerPropietario('perfil', $model->id)) {  
  
        echo Html::a('Update', ['update', 'id' => $model->id],  
                    ['class' => 'btn btn-primary']);  
    } ?>  
  
    <?= Html::a('Delete', ['delete', 'id' => $model->id], [  
        'class' => 'btn btn-danger',  
        'data' => [  
            'confirm' => Yii::t('app', 'Are you sure to delete this item?'),  
            'method' => 'post',  
        ],  
    ]) ?>  
  
</p>  
  
<?= DetailView::widget([  
    'model' => $model,  
    'attributes' => [  
        '/id',  
        'user.username',  
        'nombre',  
        'apellido',  
        'fecha_nacimiento',  
        'genero.genero_nombre',  
        'created_at',  
        'updated_at',  
        '/user_id',  
    ],  
]) ?>
```

```
]) ?>  
</div>
```

Ahora revisemos los cambios. Para comenzar, hacemos algunos cambios cosméticos al título para mostrar el username, que son triviales, y agregamos una sentencia use:

```
use common\models\PermisosHelpers;
```

Esto nos dará acceso al método PermisosHelpers::userDebeSerPropietario() method. Este es uno de los métodos auxiliares que creamos en el capítulo de Helpers, que devuelve verdadero o falso para determinar si el usuario actual es el propietario del registro.

Vamos a usar userDebeSerPropietario(), para agregar una capa de seguridad extra en nuestra navegación en frontend/views/perfil/view.php. En este caso, encerramos al método en una sentencia if, y si es verdadera, mostramos la navegación:

```
<p>  
<?Php  
  
//esto no es necesario pero está aquí como ejemplo  
  
if(PermisosHelpers::userDebeSerPropietario( 'perfil', $model->id)) {  
  
    echo Html::a('Update', [ 'update', 'id' => $model->id],  
                [ 'class' => 'btn btn-primary']);  
}  
?>
```

El método userDebeSerPropietario recibe dos argumentos, primero, el nombre del modelo enviado como una cadena, segundo, la id de la instancia del modelo, disponible para nosotros en \$model->id debido a que enviamos la instancia del modelo a la vista a través del controlador.

Como indica el comentario en el código, esta verificación sobre el usuario no es necesaria realmente para perfil/view, no puede acceder a la página view sin ser el propietario del registro y la acción de actualización también realiza una comprobación para limitar el acceso al propietario solamente.

La he incluido aquí simplemente como un ejemplo para casos donde pueda tener registros visibles a todos lupdate sólo disponibles para los propietarios de los registros. Un blog donde los autores pueden actualizar sus posts, pero otros usuarios sólo pueden leerlos, sería un ejemplo de esto.

Podría haber hecho la misma verificación en el link de borrado, pero nuevamente, no es necesario, ya que el controlador verifica el propietario y sólo los propietarios de registros pueden acceder a la página view donde reside la navegación.

Finalmente, tenemos algunos cambios en el widget DetailView:

```
<?= DetailView::widget([
    'model' => $model,
    'attributes' => [
        //'id',
        'user.username',
        'nombre',
        'apellido',
        'fecha_nacimiento',
        'genero.genero_nombre',
        'created_at',
        'updated_at',
        //'user_id',
    ],
]) ?>
```

Hemos comentado los campos “*id*” y “*user_id*” porque estos números no significan nada para los usuarios finales. En su lugar agregamos `user.username` y `genero.genero_nombre`. Estamos accediendo a estas propiedades a través de relaciones lazy-loaded y esa es la sintaxis para ello. ¿Bastante simple verdad?

Tratando brevemente sobre lo que significa `lazy load`, significa que existirá una consulta por cada fila, en este caso dos consultas separadas. Eso está probablemente bien en una página como esta que tiene un pequeño número de consultas. Para listas largas con múltiples consultas, sería muy ineficiente y necesitamos usar `eager loading` en esos casos, y le mostraremos cómo hacerlo cuando tratemos con esa clase de resultados.

Necesitamos hacer un poco de tarea en el modelo Genero con respecto a las etiquetas de atributos, para que muestre de manera correcta lo que deseamos en la página.

Genero

Comencemos con el sencillo, `Genero.php` ubicado en `frontend/models`.

Tenemos solamente un cambio en las etiquetas de atributos. No proveeré de un Gist, ya que es sólo un cambio de una palabra:

```
public function attributeLabels()
{
    return [
        'id' => 'ID',
        'genero_nombre' => 'Genero Nombre',
    ];
}
```

Debería ser cambiado a:

```
public function attributeLabels()
{
    return [
        'id' => 'ID',
        'genero_nombre' => 'Genero',
    ];
}
```

Esa es la única etiqueta de atributo que dejé para trabajar cuando construimos los nuevos modelos en el capítulo Nuevos Modelos. Todas las otras, ya las tenemos en su lugar, así como las relaciones adicionales.

Puede ver que pusimos esas otras etiquetas fuera del flujo normal de trabajo, de otra manera estaríamos saltando entre modelos y vistas. Pensé que era mejor mantener un foco ajustado en una cosa a la vez, así las personas que son nuevas en el framework, todavía serán capaces de absorber la información más fácilmente.

Parcial Formulario

Bien, a continuación vamos a modificar `_form.php`, que es un parcial. Un parcial, que en Yii 2 se designa con un guión bajo enfrente del nombre de archivo, es una vista que es incluida en otra vista, en este caso a través de:

```
<?= $this->render('_form', [ 'model' => $model, ])??>
```

A través de la magia del ruteo de Yii 2 y la estructura de archivos, sabe a qué `_form` se refiere. Esto hace que el código sea muy conciso. Veamos como usarlo.

El código de arriba es llamado en `Update.php`, por ejemplo. Esta es una oportunidad perfecta para mencionar que `_form.php` es más simple que la vista de contacto que observamos antes.

Por ejemplo en este caso, ya que estamos realizando simplemente acciones CRUD, no necesitamos un modelo de formulario separado. Ni siquiera necesitamos especificar un modelo de formulario en absoluto porque una vez más Yii 2 sabe a partir de la estructura de archivos, y del modelo que le

es entregado a la vista desde el controlador, que modelo actualizar. Esto es genial y ahorra mucho tiempo.

Así que para crear, actualizar, borrar, típicamente no necesita un modelo separado. Es sólo cuando cuando existe alguna validación compleja u otro proceso que necesitará un modelo para el formulario.

En nuestro ejemplo de aquí, Update.php, \$model es el modelo Perfil.

Bien, de regreso a _form. Esto es lo que nos entregó Gii:

```
<?php

use yii\helpers\Html;
use yii\widgets\ActiveForm;

/* @var $this yii\web\View */
/* @var $model frontend\models\Perfil */
/* @var $form yii\widgets\ActiveForm */
?>

<div class="perfil-form">

    <?php $form = ActiveForm::begin(); ?>

    <?= $form->field($model, 'user_id')->textInput(['maxlength' => 11]) ?>

    <?= $form->field($model, 'nombre')->textarea(['rows' => 6]) ?>

    <?= $form->field($model, 'apellido')->textarea(['rows' => 6]) ?>

    <?= $form->field($model, 'fecha_nacimiento')->textInput() ?>

    <?= $form->field($model, 'genero_id')->textInput() ?>

    <?= $form->field($model, 'created_at')->textInput() ?>

    <?= $form->field($model, 'updated_at')->textInput() ?>

    <div class="form-group">
        <?= Html::submitButton($model->isNewRecord ? 'Create' : 'Update',
['class' => $model->isNewRecord ? 'btn btn-success' : 'btn btn-primary']) ?>
    </div>
```

```
<?php ActiveForm::end(); ?>

</div>
```

Este es código agradable y conciso y tiene que amar al framework por ello. Pero hay algunas cosas que no necesitamos. La id del usuario no necesita mostrarse en el formulario, está establecida en el formulario, así que es guardada de forma correcta sin entrada del formulario. También podemos eliminar los campos created_at y updated_at ya que hemos agregado behaviors en el modelo Perfil que los insertarán automáticamente por nosotros.

Luego tenemos otros dos cambios, pondremos una nota bajo el campo fecha para explicar la entrada y crear una lista desplegable para Genero. Aquí tenemos cómo se ve el archivo completo:

Gist:

Form Partial

Del libro:

```
<?php

use yii\helpers\Html;
use yii\widgets\ActiveForm;

/* @var $this yii\web\View */
/* @var $model frontend\models\Perfil */
/* @var $form yii\widgets\ActiveForm */
?>

<div class="perfil-form">

<?php $form = ActiveForm::begin(); ?>

<?= $form->field($model, 'nombre')->textInput(['maxlength' => 45]) ?>

<?= $form->field($model, 'apellido')->textInput(['maxlength' => 45]) ?>

<?= $form->field($model, 'fecha_nacimiento')->textInput() ?>
* por favor use el formato YYYY-MM-DD

<?= $form->field($model, 'genero_id')->dropDownList($model->generoLista,
['prompt' => 'Por favor Seleccione Uno']);?>

<div class="form-group">
<?= Html::submitButton($model->isNewRecord ? 'Create' : 'Update',
```

```
[ 'class' => $model->isNewRecord ? 'btn btn-success' : 'btn btn-primary' ]) ?>
</div>

<?php ActiveForm::end(); ?>

</div>
```

Note que <?= es la sentencia abreviada de <?php echo. Cada campo del formulario tiene una línea separada para y puede ver cómo lo hicimos arriba.

Regresando a _form, deberíamos prestar atención al campo para genero_id:

```
<?= $form->field($model, 'genero_id')->dropDownList($model->generoLista, [ 'prompt' =>
'Por favor Seleccione Uno' ]); ?>
```

Dos cosas a tener en cuenta. Insertamos el método dropDownList method usando \$model->generoLista, **usando un método mágico get**, de ahí la g minúscula en genero. Podemos hacer esto gracias el método getGeneroLista que agregamos a Perfil en un capítulo anterior.

También agregamos un parámetro dentro de un array para el prompt porque no queremos que la lista muestre por defecto el primer valor, que es lo que haría si el prompt no estuviera allí.

Ahora continuemos con los pequeños cambios restantes en la vista que tenemos en mente para las vistas de Perfil del frontend.

Create

Abra create.php:

y cambie esta línea:

```
$this->params['breadcrumbs'][] = ['label' => 'Perfils', 'url' => ['index']];
```

A esta otra:

```
$this->params['breadcrumbs'][] = ['label' => 'Perfil', 'url' => ['index']];
```

Simplemente estamos eliminando la 's' en Perfil.

Update

Ahora continuemos con update.php. Elimine la 2da línea del breadcrumb y cambie el título, de manera que se vea así:

```
<?php

use yii\helpers\Html;

/* @var $this yii\web\View */
/* @var $model frontend\models\Perfil */

$this->title = 'Actualizar el Perfil de ' . $model->user->username;
$this->params['breadcrumbs'][] = ['label' => 'Perfil', 'url' => ['index']];

$this->params['breadcrumbs'][] = 'Actualizar';
?>
<div class="perfil-update">

<h1><?= Html::encode($this->title) ?></h1>

<?= $this->render('_form', [
    'model' => $model,
]) ?>

</div>
```

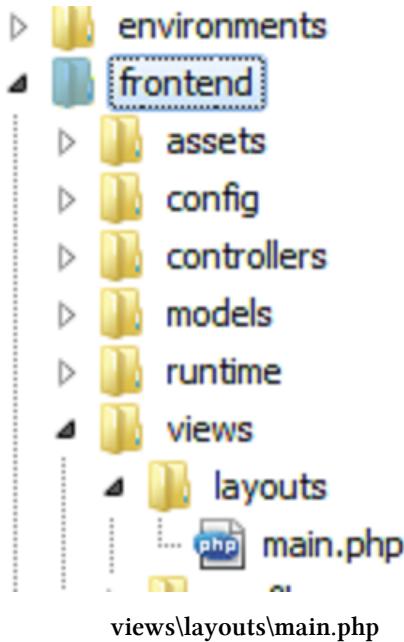
Y simplemente para mantener la consistencia, eliminemos también esa ‘s’ de la palabra Perfil en el breadcrumbs de view.php:

```
$this->params['breadcrumbs'][] = ['label' => 'Perfil', 'url' => ['index']];
$this->params['breadcrumbs'][] = $this->title;
```

Eso finaliza los cambios a las vistas de Perfil. Notará que no hemos alterado index.php o _search.php. No necesitamos esos archivos, fueron auto-generados por Gii. En lugar de eliminarlos ahora, esperaremos hasta más adelante en el proyecto porque si cambiamos de parecer y queremos utilizarlos, los tendremos y no habrá necesidad de rehacerlos. Si su sentido de flujo de trabajo se ve ofendido por esto, siéntase en libertad de eliminarlos ahora.

Layout del Sitio

Es momento de que modifiquemos el layout del sitio para incluir un link a Perfil en la cabecera. Queremos que este link aparezca junto al link para cerrar sesión, que es el que aparece cuando estamos logueados. Así que necesitamos modificar frontend/views/layout/main.php



Aquí es donde se encuentra el archivo. Hay uno similar en el backend, así que asegúrese de que está en el lugar correcto.

Esto es lo que obtiene de manera predeterminada con la plantilla avanzada:

```
<?php
use yii\helpers\Html;
use yii\bootstrap\Nav;
use yii\bootstrap\NavBar;
use yii\widgets\Breadcrumbs;
use frontend\assets\AppAsset;
use frontend\widgets\Alert;

/* @var $this \yii\web\View */
/* @var $content string */

AppAsset::register($this);
?>
<?php $this->beginPage() ?>
<!DOCTYPE html>
<html lang="= Yii::$app-&gt;language ?&gt;"&gt;
&lt;head&gt;
  &lt;meta charset="<?= Yii::$app-&gt;charset ?&gt;"&gt;
  &lt;meta name="viewport" content="width=device-width, initial-scale=1"&gt;
  &lt;?= Html::csrfMetaTags() ?&gt;
  &lt;title&gt;&lt;?= Html::encode($this-&gt;title) ?&gt;&lt;/title&gt;</pre

```

```
<?php $this->head() ?>
</head>
<body>
    <?php $this->beginBody() ?>
    <div class="wrap">
        <?php
            NavBar::begin([
                'brandLabel' => 'My Company',
                'brandUrl' => Yii::$app->homeUrl,
                'options' => [
                    'class' => 'navbar-inverse navbar-fixed-top',
                ],
            ]);
            $menuItems = [
                ['label' => 'Home', 'url' => ['/site/index']],
                ['label' => 'About', 'url' => ['/site/about']],
                ['label' => 'Contact', 'url' => ['/site/contact']],
            ];
            if (Yii::$app->user->isGuest) {
                $menuItems[] = ['label' => 'Signup', 'url' => ['/site/signup']];
                $menuItems[] = ['label' => 'Login', 'url' => ['/site/login']];
            } else {
                $menuItems[] = [
                    'label' => 'Logout (' . Yii::$app->user->identity->username . ')',
                    'url' => ['/site/logout'],
                    'linkOptions' => ['data-method' => 'post']
                ];
            }
            echo Nav::widget([
                'options' => ['class' => 'navbar-nav navbar-right'],
                'items' => $menuItems,
            ]);
            NavBar::end();
        ?>

        <div class="container">
            <?= Breadcrumbs::widget([
                'links' => isset($this->params['breadcrumbs']) ?
                    $this->params['breadcrumbs'] : []
            ]) ?>
            <?= Alert::widget() ?>
            <?= $content ?>
        </div>
```

```
</div>

<footer class="footer">
    <div class="container">
        <p class="pull-left">&copy; My Company <?= date('Y') ?></p>
        <p class="pull-right"><?= Yii::powered() ?></p>
    </div>
</footer>

<?php $this->endBody() ?>
</body>
</html>
<?php $this->endPage() ?>
```

Esta es una plantilla agradable y concisa que hace bastante. AppAsset::register(\$this); incluye las hojas de estilo y archivos js. para la plantilla. Yii 2 utiliza un sistema de publicación para guardar recursos en cache. Debe poner mucho cuidado a sus archivos de configuración.

Más adelante en el libro, cubriremos la implementación de un nuevo recurso, pero no profundizaremos mucho. Este libro realmente no cubre la decoración del frontend con mucho detalle.

Pero podemos pasar un minuto describiendo cómo funciona el layout. Si se fija en el cuerpo del layout, verá:

```
<?= $content ?>
```

Esa sentencia bastante inconspicua ubica a la vista dentro del layout. Los mecanismos de ruteo de Yii 2 saben que vista y qué layout utilizar. Puede usar múltiples layouts y temas por lo que este asunto se vuelve muy profundo rápidamente, pero como dijimos, sólo estamos cubriendo la superficie. Puede ver que la sección cerca de la parte inferior es:

```
<footer class="footer">
```

Todo lo que se encuentra debajo está en la sección del pie de página.

Todo lo que se encuentre arriba de <?= \$content ?> estará en el encabezado.

Pongamos el nombre de nuestra plantilla en el widget NavBar. Es un cambio de una palabra, así que no hay un Gist:

```
NavBar::begin([
    'brandLabel' => 'Yii 2 Build',
    'brandUrl' => Yii::$app->homeUrl,
    'options' => [
        'class' => 'navbar-inverse navbar-fixed-top',
    ],
]);
```

Link a Perfil

Lo siguiente que vamos a hacer es agregar un link a Perfil. Ahora bien, la forma en que configuramos el controlador Perfil es que la acción index primero verifica si existe un registro. Si es así, redirige a la página de visualización y si no, redirige a la vista de creación. Así que en realidad no necesitamos un link a Perfil. Las vistas update y delete ya están vinculadas dentro del archivo view.php, así que no hay necesidad de crear esa navegación.

Lo que queremos probar sin embargo, es que el usuario se encuentra logueado. No queremos mostrar el link al perfil si el usuario no se encuentra logueado. También note, que no necesitamos proveer de una variable get. Eliminamos la necesidad de ello con la forma en que escribimos la lógica del controlador.

Inserte lo siguiente:

```
$menuItems[] = ['label' => 'Perfil', 'url' => ['/perfil/view']];
```

Coloque esto en la sentencia else entre la apertura y cierre del NavBar:

```
if (Yii::$app->user->isGuest) {
$menuItems[] = ['label' => 'Signup', 'url' => ['/site/signup']];
$menuItems[] = ['label' => 'Login', 'url' => ['/site/login']];
} else {
$menuItems[] = ['label' => 'Perfil', 'url' => ['/perfil/view']];
$menuItems[] = [
    'label' => 'Logout (' . Yii::$app->user->identity->username . ')',
    'url' => ['/site/logout'],
    'linkOptions' => ['data-method' => 'post']
];
}
```

Así que estamos usando el método Yii::\$app->isGuest para determinar si el usuario está logueado a no, y si es así, le mostramos el link a perfil. El formato de arreglo \$menuItems funciona porque está entre NavBar::begin y NavBar::end.

Bien, con todos estos cambios en su lugar, podemos iniciar sesión como un usuario y jugar creando un perfil, actualizándolo, borrándolo, etc. Adelante, asegúrese de que todo funciona adecuadamente.

Intente agregar una variable get inválida a la url, como:

```
http://yii2build.com/index.php?r=perfil/update&id=7
```

Verá que sin importar lo que ponga en esa id, devolverá sólo el link al usuario correcto. Así que en este punto, debería estar trabajando de la forma esperada.

Si no, regrese sobre sus pasos y trate de encontrar el error de tipeo.

Ahora antes de concluir el capítulo, deshagamonos del feo cuadro de texto para la fecha en el parcial _form.

DatePicker

Asegurémonos de que recordamos incluir la sentencia use en _form.php:

```
use yii\jui\DatePicker;
```

Esta es un librería de terceros que no está incluida en la instalación base de Yii. Cuando actualizamos composer para asegurarnos de que teníamos Gii, se supone que también instalamos la librería jui. Debería tener:

```
"yiisoft/yii2-jui": "*"
```

Su archivo composer.json en yii2build debería verse como el siguiente:

```
"require-dev": {
    "yiisoft/yii2-codeception": "*",
    "yiisoft/yii2-debug": "*",
    "yiisoft/yii2-gii": "*",
    "yiisoft/yii2-faker": "*",
    "yiisoft/yii2-jui": "*"
},
```

Puede ver la última línea que añadimos “yiisoft/yii2-jui”: “*” No olvide la coma en la línea anterior.

Si no se encontraba allí, entonces no la tiene, por lo que debe agregar esa línea ahora. Luego diríjase a la línea de comandos y tipee:

```
\var\www\yii2build>composer update
```

Esto agregará las dependencias de yii2-jui. Ahora estamos listos para usarla.

Lo siguiente es reemplazar la línea de input en _form.php con:

```
<?php echo $form->field($model, 'fecha_nacimiento')  
    ->widget(DatePicker::className(),  
        ['clientOptions' => ['dateFormat' => 'yy-mm-dd']])); ?>
```

Continúe, realice el cambio y refresque su página. Si todo ha ido bien, debería ver una diferencia en sus formularios update y create de su Perfil.

Por favor note que no va a funcionar correctamente. Lo escribí con una versión anterior del widget, Yii 2.0 y la configuración es ahora un poco diferente, ya que estamos trabajando con Yii 2.0.3 o superior.

Tener que resolver el problema que crea la antigua configuración es instructivo, así que por el momento, voy a pretender que no existe la solución y que necesitamos encontrar un arreglo. No es tan poco frecuente encontrar tales situaciones y este es un escenario simple con el que trabajar y aprender a superarlo.

El problema aquí se reduce al hecho de que el widget no respeta el formato de fecha y el formato que usa por defecto no pasaría la validación. Así que en esencia, update y create están muertos hasta que los arreglamos.

El formato por defecto del widget es:

MMM d, Y

Esto traduce a una fecha como Oct 14, 2014

El formato que necesitamos para datetime en MySQL es:

Y-m-d

Esto traduce a 2014-10-14.

Así que obviamente tenemos un problema de conversión. Podemos solucionarlo creando un método a aplicar antes de la validación en el modelo Perfil. Yii 2 tiene este método realmente útil llamado `beforeValidate`, que será llamado automáticamente siempre y cuando el método llame al de su clase padre. Aquí está el código, que para ser claros, va en el modelo Perfil en frontend/models/Perfil.php:

Gist:

[BeforeValidate Perfil](#)

Del libro:

```

public function beforeValidate()
{
    if ($this->fecha_nacimiento != null) {

        $nuevo_formato_fecha = date('Y-m-d', strtotime($this->fecha_nacimiento));
        $this->fecha_nacimiento = $nuevo_formato_fecha;
    }

    return parent::beforeValidate();
}

```

Así que si el atributo fecha_nacimiento no está vacío, tomar la instancia \$this de fecha_nacimiento y darle formato usando:

```
date('Y-m-d', strtotime($this->fecha_nacimiento))
```

Esas son funciones predeterminadas de PHP, date y strtotime. En fin, asignamos \$this->birthdate a \$nuevo_formato_fecha, que ahora guarda la fecha de nacimiento que se le entregó, en el formato de fecha correcto.

Luego devolvemos Parent::beforeValidate(); para asegurarnos de que el método sea llamado y eso es todo.

Bien, así que puede ver cómo solucionar un problema de formateo cómo ese.

Un lector tuvo la gentileza de enviar una nueva configuración para el widget, que soluciona el problema, y también añadió una opción genial para que el año sea fácil de seleccionar.

Gist:

Solución DatePicker

Del libro:

```

<?php echo $form->field($model, 'fecha_nacimiento')->
    widget(DatePicker::className(), [
        'dateFormat' => 'yyyy-MM-dd',
        'clientOptions' => [
            'yearRange' => '-115:+0',
            'changeYear' => true
        ]
    ]) ?>

```

Por favor note que cuando usa la lista desplegable para seleccionar el año, debe seleccionar también el día o no será establecido. También, no olvide remover el método beforeValidate de Perfil.php si lo añadió allí, era sólo como demostración y no es necesario.

Resumen



Commit!

Así que allí lo tiene, un modelo de usuario funcional. Puede simplemente hacer clic en el link ahora y probar todas las páginas. Puede ver que finalmente hemos comenzado a desarrollar, sin seguir confinados a aprender sobre cómo funcionan las cosas.

Vio que pudimos usar nuestra clase auxiliar para verificar si el usuario tiene o no un perfil. Construimos nuestros helpers con anticipación debido a que consideré el concepto en su conjunto fuera del flujo de trabajo. Cuando se encuentre desarrollando una aplicación, pensará en las cosas a medida que avance, lo que está perfectamente bien.

También tuvo la oportunidad de ver un mayor número de eficiencias otorgadas por Yii 2. Pudimos usar la herramienta Gii, que nos entregó una arquitectura lista para usar que sólo necesitaba un poco de ajuste. El controlador necesitó el mayor trabajo, pero sólo porque se trata de registros privados, propiedad de un usuario y la plantilla está orientada a registros públicos. Cuando construyamos el backend, la salida de Gii se acercará más a lo que deseamos.

Y finalmente modificamos nuestras vistas, consiguiendo que la aplicación sea más intuitiva. E inmediatamente sentimos la diferencia. Está comenzando a concretarse.

Capítulo Nueve: Upgrade y Control de Acceso

Queremos ser capaces de administrar contenido que requiera de upgrade en nuestra aplicación. Si recuerda, en nuestro método PremisosHelpers::requerirUpgradeA(\$tipo_usuario_nombre), comprobamos si el tipo de usuario coincide con la cadena que se le pasa en la firma, y si no, lo redirige a 'upgrade/index'.

Esta es una característica sencilla de implementar si quiere construir una aplicación que tenga contenido reservado para miembros pagos. Pero no podemos demostrar esto hasta que construyamos un controlador upgrade y el correspondiente archivo de vista, así que hágámoslo ahora.

Dirija su navegador a:

<http://www.yii2build.com/index.php?r=gii>

Ubiquemos nuestro nuevo controlador en el frontend ya que nuestros usuarios del frontend serán los que necesiten realizar el upgrade. Agregue esto en el campo Controller Class:

frontend\controllers\UpgradeController

Ya sea que esté creando un controlador del frontend o backend, seguirán la convención arriba indicada, simplemente use el directorio apropiado.

Asegúrese de que los siguientes campos están establecidos si es que no se auto-completan:

Action Ids: index

Deje el campo view path vacío, él sabrá cuál es la ruta correcta porque estamos usando la configuración por defecto.

Debería verse así:

The screenshot shows the 'Controller Generator' section of the Yii Gii interface. On the left, a sidebar lists generators: Model Generator, CRUD Generator, Controller Generator (selected), Form Generator, Module Generator, and Extension Generator. The main panel has several input fields:

- Controller Class:** frontend\controllers\UpgradeController
- Action IDs:** index
- View Path:** (empty)
- Base Class:** yii\web\Controller
- Code Template:** default (C:\var\www\yii2build\vendor\yiisoft\yii2-gii\generators\controller/default)

A blue 'Preview' button is located at the bottom of the main panel.

Controlador Upgrade en Gii

Muy bien, generemos el código. Esto no sólo creará el controlador, sino también el correspondiente directorio y archivo de vista. En este caso sólo tenemos una acción que deseamos crear, index. Puede crear más de una acción separándolas por comas. En este caso, la acción index renderizará el archivo index, que usted usará para ofrecer al usuario las opciones de pago.

Sólo vamos a simular las opciones de pago, simplemente pondremos un poco de contenido en nuestra vista index, eso es lo más lejos que llegarán nuestras instrucciones en este libro. Si realmente desea implementar opciones de pago reales, le recomiendo revisar Stripe. Para compañías pequeñas, esta solución tiene sentido, y tienen mucha documentación para la integración:

[Stripe](#)

También, está:

[PayPal](#)

Puede desear ofrecer las dos opciones.

Controlador Upgrade

Bien, regresemos a Gii. Una vez que haya ejecutado la generación con Gii, obtendrá:

UpgradeController.php

```
<?php

namespace frontend\controllers;

class UpgradeController extends \yii\web\Controller
{
    public function actionIndex()
    {
        return $this->render('index');
    }

}
```

Vista Index de Upgrade

Bastante sencilla, simplemente renderiza la vista, que está en views/upgrade/index.php:

```
<?php
/* @var $this yii\web\View */
?>
<h1>upgrade/index</h1>

<p>
    You may change the content of this page by modifying
    the file <code><?= __FILE__; ?></code>.
</p>
```

Nuevamente, nada especial. Así que ya simplemente estamos simulando, la dejaremos así. Ahora para probar como funciona todo, simplemente podemos agregar una línea en nuestro archivo PerfilController.php.

Requerir Upgrade A

Bien, ahora que regresamos a nuestro **Controlador Perfil** en frontend/controllers/PerfilController.php, agreguemos como primera línea de actionUpdate:

```
PermisosHelpers::requerirUpgradeA('Pago');
```

Así que el método completo debería verse así:

```

public function actionUpdate()
{
    PermisosHelpers::requerirUpgradeA('Pago');

    if ($model = Perfil::find()->where(['user_id' =>
        Yii::$app->user->identity->id])->one()) {

        if ($model->load(Yii::$app->request->post()) && $model->save()) {
            return $this->redirect(['view', 'id' => $model->id]);
        } else {

            return $this->render('update', [
                'model' => $model,
            ]);
        }
    } else {

        throw new NotFoundHttpException('No Existe el Perfil.');
    }
}

```

Así que no realizamos ningún cambio excepto por la primera línea, que ahora le dice al controlador que el usuario debe tener tipo_usuario_nombre ‘Pago’. Asegúrese de guardar los cambios, y luego podrá probarlo si inicia sesión con usuario existente que sólo tiene el tipo_usuario por defecto, que es Gratuito.



Consejo

Debería registrar varios usuarios en la aplicación para experimentar con su configuración de tipo_usuario_id en PhpMyAdmin, para que pueda probar diferentes escenarios.

Ahora que tiene un usuario de prueba con tipo_usuario ‘Gratuito’, puede intentar actualizar el perfil, y cuando lo haga, será redirigido a la vista upgrade. ¿Qué tan simple es eso?

Control de Acceso

Mientras nos encontramos en el controlador Perfil podemos hablar un poco más acerca del control de acceso. Una de las ideas más no obvias sería la de requerir que el usuario tenga un estado Activo para acceder al controlador Perfil. Por supuesto la aplicación lo requiere para iniciar sesión, para

que nadie pueda hacerlo si no tiene un estado activo. ¿Pero qué sucede si alguien degrada su estado, y luego presiona el botón regresar? Teóricamente aún pueden acceder todo y eso es un control descuidado sobre el contenido.

Así que vamos a agregar una regla de acceso que requiera un estado activo. No se preocupe, será realmente simple porque ya hemos anticipado esto y hemos construido un helper para ello. Y también, la clase behaviors de Yii 2 incluye control de acceso, que ya hemos visto en acción. Ahora simplemente vamos a agregarle algo más:

Cambie el método behaviors existente en el controlador Perfil por el siguiente:

Gist:

Behaviors del Controlador Perfil

Del libro:

```
public function behaviors()
{
    return [
        'access' => [
            'class' => \yii\filters\AccessControl::className(),
            'only' => ['index', 'view', 'create', 'update', 'delete'],
            'rules' => [
                [
                    'actions' => ['index', 'view', 'create', 'update', 'delete'],
                    'allow' => true,
                    'roles' => ['@'],
                ],
            ],
        ],
        'access2' => [
            'class' => \yii\filters\AccessControl::className(),
            'only' => ['index', 'view', 'create', 'update', 'delete'],
            'rules' => [
                [
                    'actions' => ['index', 'view', 'create', 'update', 'delete'],
                    'allow' => true,
                    'roles' => ['@'],
                    'matchCallback' => function ($rule, $action) {
                        return PermisosHelpers::requerirEstado('Activo');
                    }
                ],
            ],
        ],
    ];
}
```

```

] ,
] ,
'verbs' => [
    'class' => VerbFilter::className(),
    'actions' => [
        'delete' => ['post'],
    ],
],
];
}

```

Si obtiene un error de clase VerbFilter no encontrada, asegúrese de tener esta sentencia use al comienzo del archivo:

```
use yii\filters\VerbFilter;
```

Esta fue una ruta de clase que fue cambiada por el propio Framework Yii 2 durante el tiempo en que estaba escribiendo este libro, así que si lo olvidé en cualquier otro lugar, esa es la solución. Cuándo se encuentra desarrollando una aplicación, es fácil olvidarse de una sentencia use. Las buenas noticias son que Yii 2 se quejará y le señalará el archivo faltante, simplemente necesita formatear la sentencia use apropiadamente.

Las sentencias use completas deberían verse así:

```

use Yii;
use frontend\models\Perfil;
use frontend\models\search\PerfilSearch;
use yii\web\Controller;
use yii\web\NotFoundHttpException;
use yii\filters\VerbFilter;
use common\models\PermisosHelpers;
use common\models\RegistrosHelpers;

```

He incluido una para PerfilSearch en caso de que quiera jugar con ella. Siéntase en libertad de eliminar cualquier sentencia use que no sea necesaria en el controlador.

Bien, de regreso behaviors. Puede ver que hemos agregado otro arreglo denominado ‘access2’:

```
'access2' => [
    'class' => \yii\filters\AccessControl::className(),
    'only' => ['index', 'view', 'create', 'update', 'delete'],
    'rules' => [
        [
            'actions' => ['index', 'view', 'create', 'update', 'delete'],
            'allow' => true,
            'roles' => ['@'],
            'matchCallback' => function ($rule, $action) {
                return PermisosHelpers::requirirEstado('Activo');
            }
        ],
    ],
],
```

La clave del arreglo, access2, simplemente es una cadena y puede llamarla como desee. Puede ver lo que hicimos aquí. Copiamos el arreglo access y agregamos el elemento ‘matchCallback’, que es un invocable php que espera como valor verdadero o falso.

Afortunadamente, ya tenemos un método PermisosHelpers listo que devuelve verdadero o falso, y que comprueba si el usuario actual tiene un estado activo. De lo contrario, no permite el acceso, y ya que hemos establecido la regla para todas las acciones, no pueden acceder a ninguna parte bajo el control del controlador Perfil. Todo es realizado de manera muy simple.

Ahora bien lo he creado de esta forma porque quiero demostrar que puede tener múltiples capas de reglas de acceso. Pero en realidad, ya que matchCallback se aplica a todas las acciones, así como lo hacen las otras reglas, simplemente podría haber añadido matchCallback al primer arreglo, en lugar de crear access2.

También puede anidar arreglos bajo reglas cuando tiene reglas que sólo se aplican a ciertas acciones, por ejemplo:

```
return [
    'access' => [
        'class' => \yii\filters\AccessControl::className(),
        'only' => ['index', 'view', 'create', 'update', 'delete'],
        'rules' => [
            [
                'actions' => ['create', 'update', 'delete'],
                'allow' => true,
                'roles' => ['@'],
            ],
        ],
    ],
],
```

```
],  
  
'rules' => [  
  
[  
  'actions' => ['index', 'view', 'create', 'update', 'delete'],  
  'allow' => true,  
  'roles' => ['@'],  
  'matchCallback' => function ($rule, $action) {  
    return PermisosHelpers::requerirEstado('Activo');  
  }  
],  
  
],  
  
],
```

La clase de control de acceso iterará a través de cada conjunto de reglas. Así que este es un método de control de acceso muy flexible y fácil de usar.

Puede ver que cuando simplemente queremos restringir el acceso, podemos usar behaviors. Cuando necesitamos restringir basados en una condición y redirigir o realizar alguna acción, podemos construir un helper que mantendrá el código de nuestro controlador muy limpio.

Regresemos ahora al Controlador Upgrade. En esta oportunidad usaremos una versión más concisa de behaviors, para mantener el código más limpio:

Gist:

Behaviors de Upgrade

Del libro:

```

public function behaviors()
{
    return [
        'access' => [
            'class' => \yii\filters\AccessControl::className(),
            'only' => ['index'],
            'rules' => [
                [
                    'actions' => ['index'],
                    'allow' => true,
                    'roles' => ['@'],
                    'matchCallback' => function ($rule, $action) {
                        return PermisosHelpers::requerirEstado('Activo');
                    }
                ],
            ],
        ],
        'verbs' => [
            'class' => VerbFilter::className(),
            'actions' => [
                'delete' => ['post'],
            ],
        ],
    ];
}

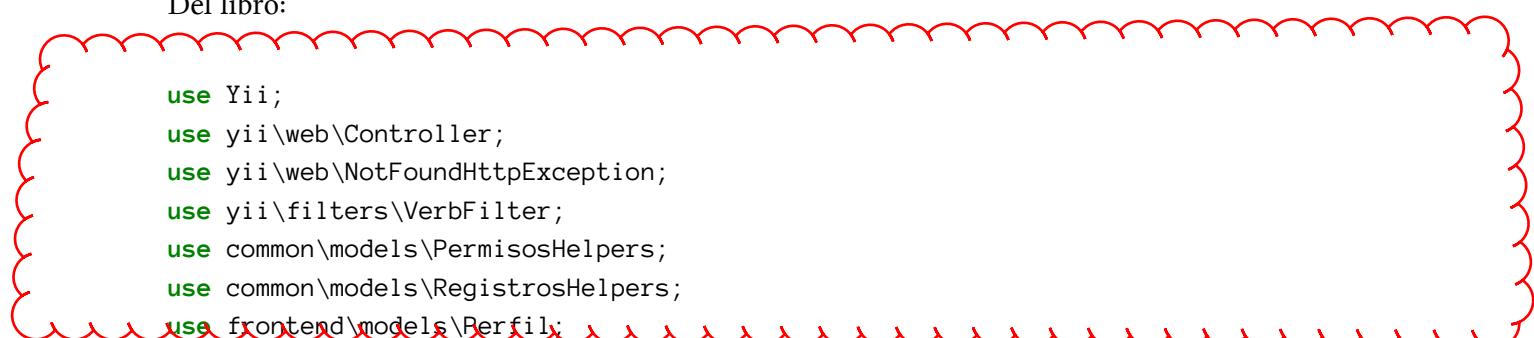
```

Ahora tendremos que agregar algunas sentencias use, cuando terminemos debería verse así,

Gist:

Sentencias Use

Del libro:



```

use Yii;
use yii\web\Controller;
use yii\web\NotFoundHttpException;
use yii\filters\VerbFilter;
use common\models\PermisosHelpers;
use common\models\RegistrosHelpers;
use frontend\models\Perfil;

```

Pasando una Variable desde el Controlador

Otro pequeño cambio que voy a realizar al controlador Upgrade es que voy a agregar una variable que será enviada a la vista:

```
$nombre = Yii::$app->user->identity->username;
```

Haré esto en actionIndex:

```
public function actionIndex()
{
    $nombre = Yii::$app->user->identity->username;

    return $this->render('index', ['name' => $nombre]);
}
```

Así que puede ver que he asignado a \$nombre el username del usuario actual. Estoy haciendo esto para mostrarle lo fácil que es mover variables y objetos hacia la vista usando el método render. Simplemente añada el arreglo con el nombre del elemento y la variable como valor, así:

```
return $this->render('index', ['nombre' => $nombre]);
```

Si tiene más de un objeto o variable, puede separarlos por comas en el mismo arreglo.

Así que ahora que tenemos el username disponible para nosotros, pongámoslo a prueba en la vista. Reemplace index.php con:

Gist:

[Upgrade Index](#)

Del libro:

```
<?php
/* @var $this yii\web\View */
?>
<h1>Ey "<?php echo $nombre; ?>, " Esto Requiere un Upgrade</h1>

<p>
    Puede obtener el acceso que desea realizando un upgrade, pero
    <?php echo $nombre; ?> , eso no es todo.
    Podrá ir a cualquier lugar, ¿no es eso genial?
</p>
```

Esto es simplemente diversión, pero comprende el punto. Esa es una simple variable, intentemos un objeto:

Gist:

Upgrade Index 2

Del libro:

```
public function actionIndex()
{
    $persona = Perfil::find()->where(['user_id' =>
        Yii::$app->user->identity->id])->one();

    return $this->render('index', ['persona' => $persona]);
}
```

Aquí estamos asignando a \$persona una instancia de Perfil, donde user_id es el usuario actual. Para lograr que esto funcione, tendremos que asegurarnos de que tenemos una sentencia use para acceder a Perfil:

```
use frontend\models\Perfil;
```

Luego en nuestra vista, podemos cambiarla a:

Gist:

Vista Index de Upgrade

Del libro:

```
<?php
/* @var $this yii\web\View */
?>
<h1>Ey "<?php echo $persona->nombre; ?>, " Esto Requiere un Upgrade</h1>

<p>
    Puede obtener el acceso que desea realizando un upgrade, pero
    <?php echo $persona->nombre; ?> , eso no es todo.
    Podrá ir a cualquier lugar, ¿no es eso genial?
</p>
```

Ahora hemos personalizado la salida para mostrar el primer nombre del usuario. Puede divertirse de muchas formas con esto, pero el punto es que es extremadamente simple traer un objeto y accederlo. Asegúrese de que ha iniciado sesión cuando lo intente o no funcionará.

Más tarde, cuando codifiquemos el backend, devolveremos objetos que contienen listas de usuarios y perfiles, usando el iterador predefinido de Yii 2 y widgets.

Resumen



En este capítulo, adquirimos un poco más de control sobre nuestra aplicación. Creamos un controlador y una vista upgrade y luego impusimos reglas para llevar al usuario a la página si no cumplen con el mínimo tipo usuario requerido para permitirle el acceso. Logramos esto agregando un método de PermisosHelper, requerirUpgradeA('Pago') al método update en el controlador de perfil.

También agregamos el requerimiento de que el usuario tenga un estado activo. Esto refuerza la seguridad y lo logramos agregando un requerimiento matchCallback a las reglas de acceso en el método behaviors. Estos fueron cambios simples que no inflaron ni volvieron confuso el código, debido en parte al hecho de que extrajimos la lógica en nuestros helpers.

Finalmente, jugamos un poco moviendo variables y objetos desde el controlador a la vista, para darnos un idea de lo fácil que es trabajar con los datos que están disponibles para nosotros.

Nos llevó un tiempo llegar a un punto donde pudimos utilizar los modelos que construimos y divertirnos un poco con ellos, pero al menos para ahora debería estar comenzando a percibir cómo es el desarrollo con Yii 2.

Capítulo Diez: Widgets Sociales de la Homepage

Implementando Widgets Sociales en la Homepage

Vamos a decorar sólo un poco la página de inicio. Queremos agregar algunos widgets sociales y un par de cosas para darle más color a la plantilla. No podemos hacer demasiado porque el propósito de ella es simplemente ser un punto de partida para otras aplicaciones.

Index

Abra frontend/views/site/index.php. Comenzaremos por agregar un par de sentencias use:

Gist:

[Use Statements Site Index](#)

Del libro:

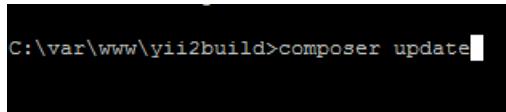
```
use \yii\bootstrap\Modal;
use kartik\social\FacebookPlugin;
use \yii\bootstrap\Collapse;
use \yii\bootstrap\Alert;
use yii\helpers\Html;
```

Esas sentencias van en la parte superior del archivo luego de la etiqueta php de apertura.

Si aún no ha implementado la extensión social de Kartik, la necesitará ahora. Compruebe si tiene lo siguiente en su archivo composer.json:

```
"minimum-stability": "stable",
"require": {
    "php": ">=5.4.0",
    "yiisoft/yii2": "*",
    "yiisoft/yii2-bootstrap": "*",
    "yiisoft/yii2-swiftmailer": "*",
    "kartik-v/yii2-social": "dev-master",
    "fortawesome/fontawesome-free": "4.2.0"
},
}
```

Puede verlo en la segunda línea desde el final. Si en este punto no tiene la línea de font-awesome en su archivo composer.json, debería agregarla también, la necesitaremos luego. Luego ejecute composer update desde la línea de comandos, como hemos hecho muchas veces antes.



Composer Update

Eso debería importar la extensión, si no la tiene aún. Puede comprobar bajo su directorio vendor si existe un directorio llamado Kartik-v, que es el directorio de la extensión.



Consejo

Puede encontrar muchas extensiones útiles por Kartik en su sitio, Krajee.com. Él es un desarrollador superestrella. Al momento de escribir esto, Kartik tiene 28 extensiones/productos que cubren todo desde los widgets sociales que estamos usando hasta extensiones para GridView y más. Si usa sus extensiones, sea amable y realice una donación si puede. Las donaciones lo mantienen trabajando en nuevas cosas que agregar al framework y eso ayuda a todos.

Widget de Facebook

Luego, regresemos al archivo de la vista del sitio Index.php y cambiemos el título a:

```
$this->title = 'Yii 2 Build';
```

Encuentre la primera div y reemplacela con:

Gist:

[1era Div del Index del Sitio](#)

Del libro:

```
<div class="site-index">

    <div class="jumbotron">

        <?php if (Yii::$app->user->isGuest) {
            echo Html::a('Get Started Today', ['site/signup'],
                ['class' => 'btn btn-lg btn-success']);} ?>
        </p>

        <h1>Yii 2 Build</h1>

        <p class="lead">Use esta plantilla de Yii 2 para comenzar Proyectos.</p>

        <br/>

        <?php echo FacebookPlugin::widget(['type'=>FacebookPlugin::LIKE,
            'settings' => []]); ?>

    </div>
```

Ahora si guarda los cambios y presiona refrescar, obtendrá el siguiente error:

```
Invalid Configuration - yii\base\InvalidConfigException
The Facebook 'appId' has not been set.
```

Configuración de la App de Facebook

Lo que necesitamos hacer es configurar nuestra app de Facebook, tanto en la configuración de Yii 2 como también configurando una app de Facebook real.

Tenga en cuenta que necesitará una cuenta de Facebook real para continuar. Si no tiene y no quiere o no puede obtener una, entonces tendrá que saltarse esta lección y remover todos los widgets sociales del proyecto.

Voy a continuar, asumiendo que tiene una cuenta de Facebook. Así que comience por ir a su cuenta de facebook e inicie sesión. En la parte inferior de la página, encuentre el link para desarrolladores.

Luego siga paso a paso para configurar una aplicación. Voy a proveer de muchas capturas de pantalla para servir como referencia, pero por favor tenga en cuenta que las cosas cambian con el tiempo, y puede no verse de la misma manera. De cualquier manera, es muy intuitivo, así que debería ser capaz de resolverlo.

Vaya a facebook y encuentre su link a configuración a partir de la flecha hacia abajo de la extrema derecha:



Navegación de Facebook

Una vez que se encuentre en su página de configuración, podrá ver el pie de página y el link de desarrolladores:

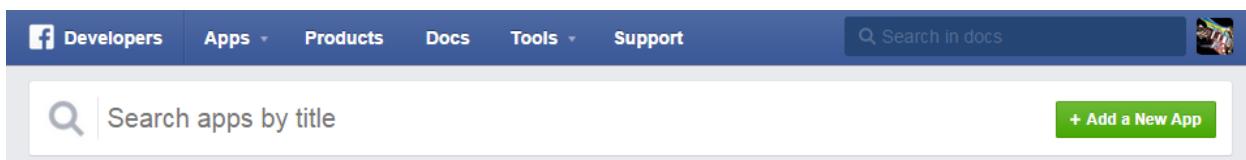


Facebook © 2014 · English (US)

Pie de Página de Facebook

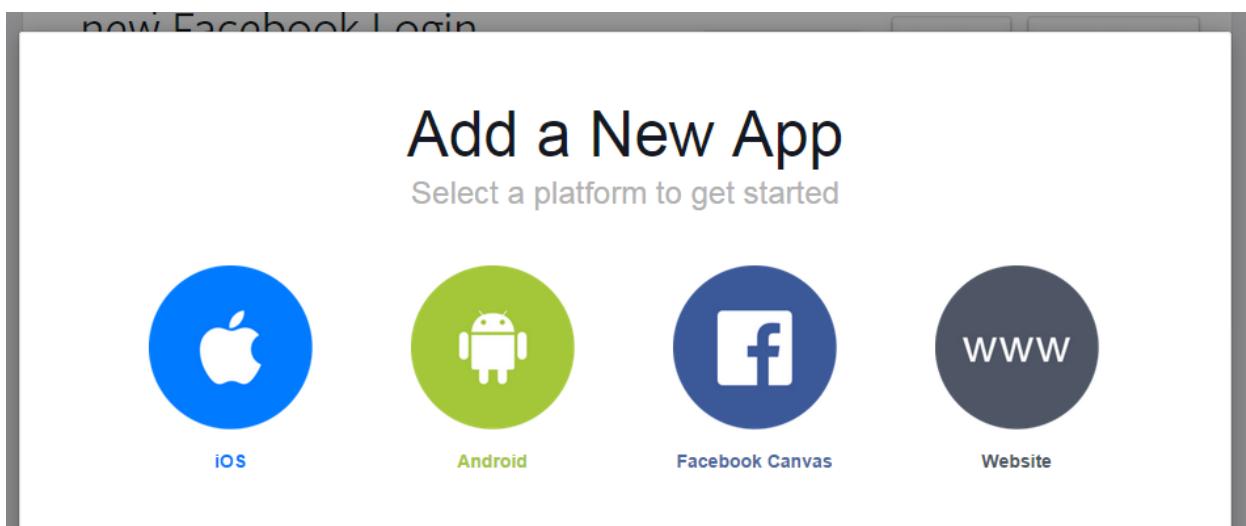
O puede simplemente dirigirse a:

<https://developers.facebook.com/apps>



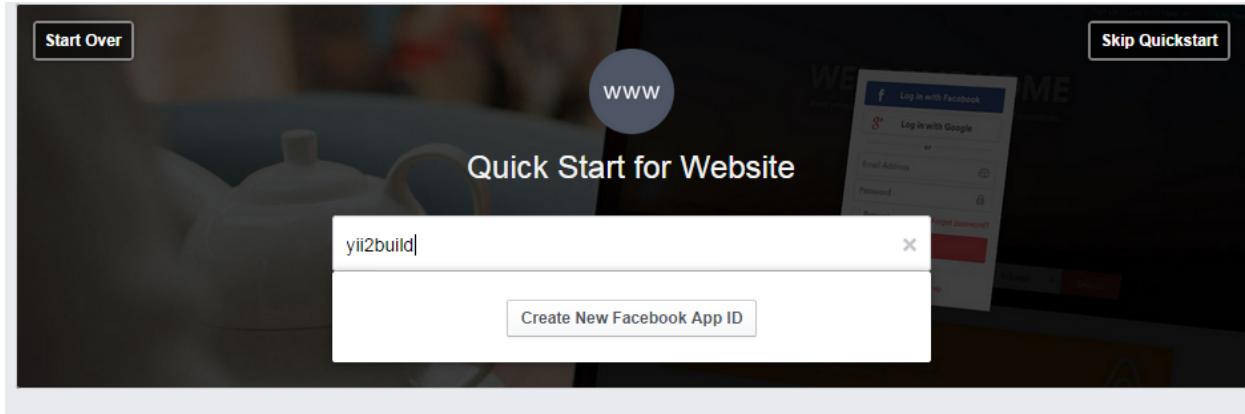
Pantalla de las App de Facebook

Seleccione Add New App:



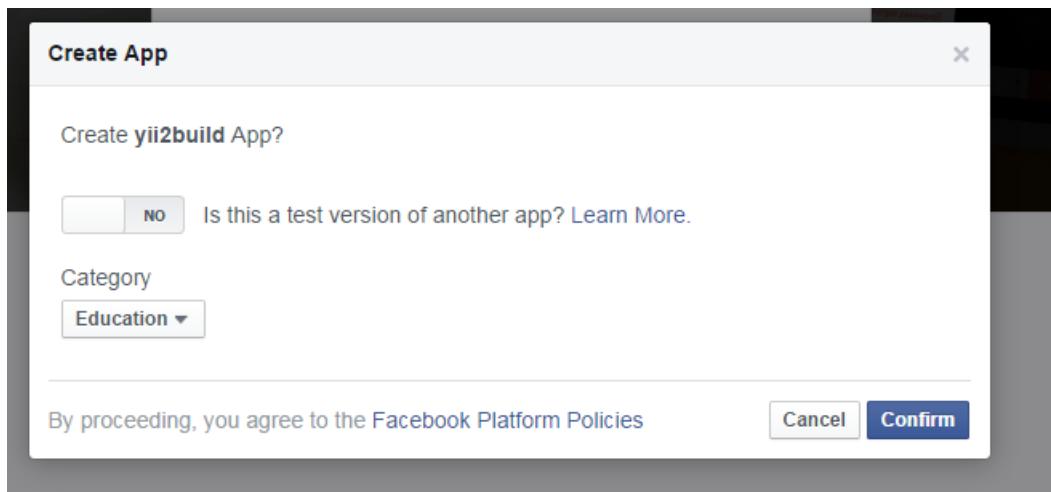
Plataforma de Facebook

seleccione website como plataforma y luego elija un nombre para la aplicación:



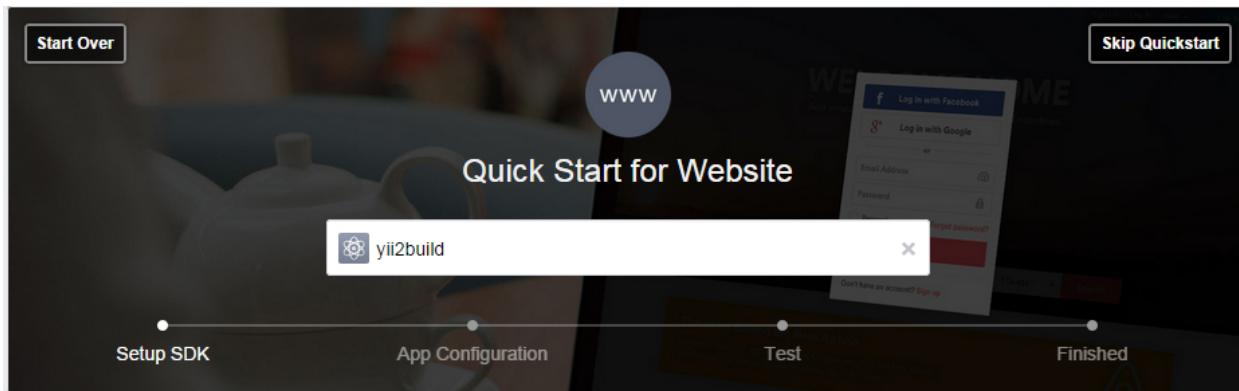
Selección de Facebook

Seleccione la Categoría (Category) y confirme:



Confirmación de Facebook

Obtendrá la pantalla de inicio rápido:



Setup the Facebook SDK for JavaScript

The following snippet of code will give the basic version of the SDK where the options are set to their most common defaults. You should insert it directly after the opening `<body>` tag on each page you want to load it:

```
<script>
  window.fbAsyncInit = function() {
    FB.init({
      ...
    });
  };
</script>
```

Inicio rápido de Facebook

Al final de la pantalla de inicio rápido, complete con la url de su sitio. Por favor note que estoy usando [Yii2build.com](#) y usted no será capaz de usarla. En el ejemplo de implementación que veremos luego, asegúrese de usar su dominio y no [Yii2build.com](#). Su dominio no necesita ser un sitio en vivo, así que continúe e ingréselo:

Tell us about your website

Site URL

Mobile Site URL

Next

Formulario de Inicio Rápido de Facebook

Hemos concluido con la configuración del SDK (pero aún no hemos acabado, haga clic en el link [Skip to Developer Dashboard](#) bajo Next Steps).

Test your Facebook Integration

Now that you've got the SDK setup, you can use it to perform a few common tasks. Social Plugins such as the Like Button and Comments Plugin can be inserted into HTML pages using the JavaScript SDK.

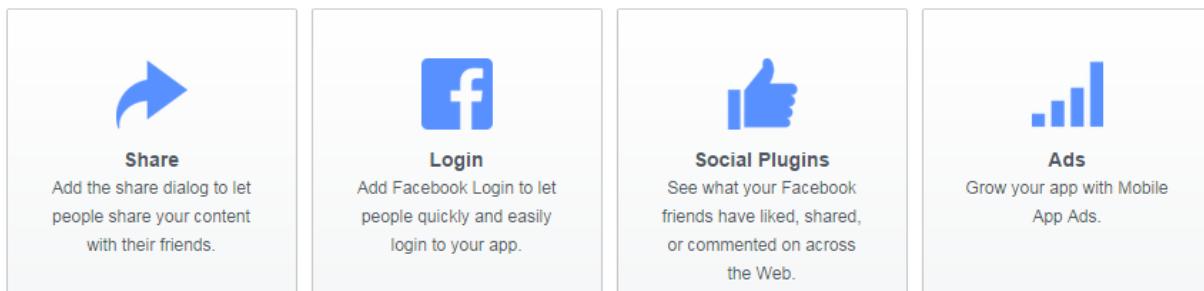
Let's try adding a Like button, just copy and paste the line of code below anywhere inside the `<body>` of your page:

```
<div  
  class="fb-like"  
  data-share="true"  
  data-width="450"  
  data-show-faces="true">  
</div>
```

Reload your page, and you should see a Like button on it.

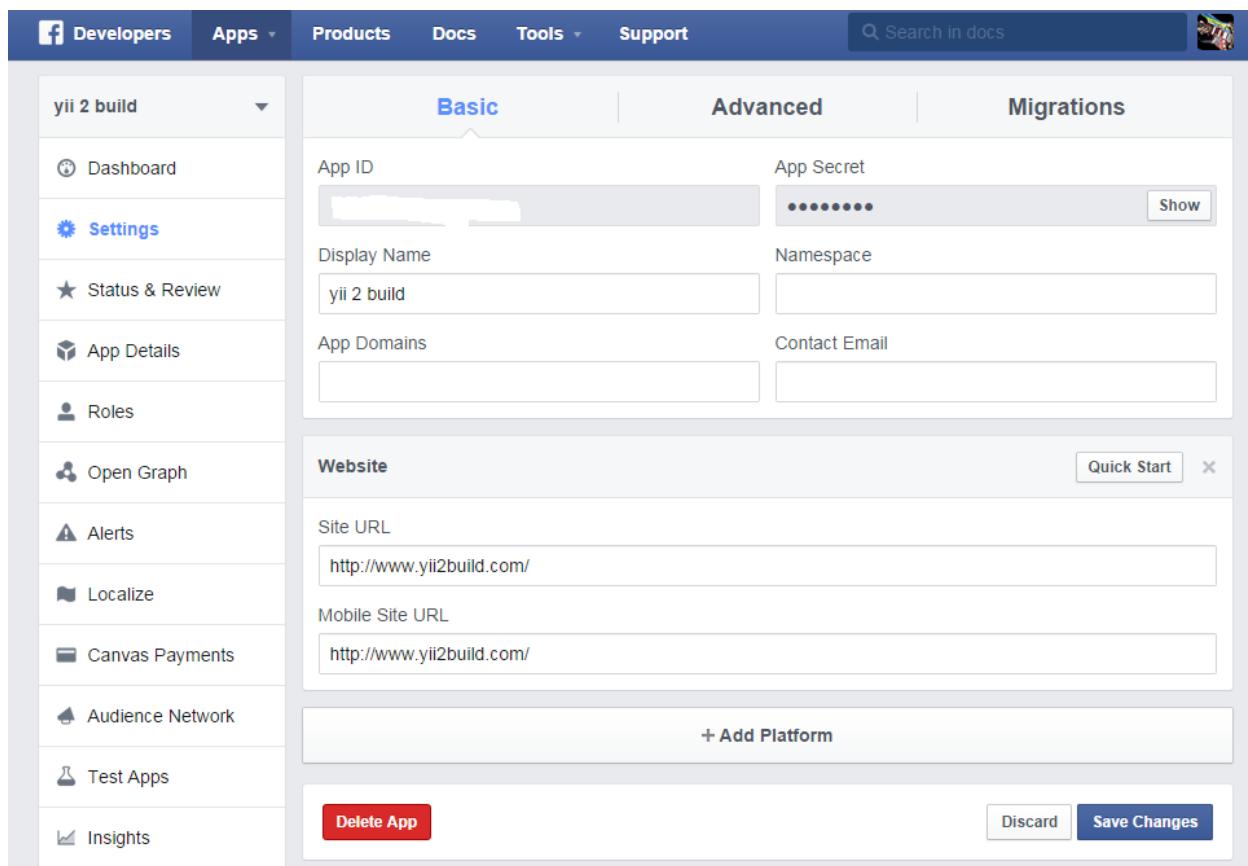
Next Steps

Congratulations! You have added the Facebook SDK to your project. You are now in the next stage in integrating your app with Facebook. What do you want to do next? Skip to Developer Dashboard or Documentation



Formulario de Inicio Rápido de Facebook Terminado

Esto lo llevará al tablero, donde hará clic en settings:



Tablero de la App de Facebook

Obviamente, he tachado la id de mi app. La suya aparecerá allí. Para copiar el app secret, seleccione el botón show.

Deberá copiar la app id y app secret en el área de configuración apropiada como describo abajo, pero no lo intente aún.

```
// la configuración global para el widget del plugin social de facebook
```

```
'facebook' => [
    'appId' => 'su id',
    'secret' => 'su secret',
],
```

No lo hemos agregado aún, pero lo haremos en un momento. También, recuerde proveer una dirección de email real dentro de la página de configuración de Facebook, de otra manera la app no funcionará.

Configuración de Facebook

Bien, continuemos con la configuración común para reconocer el módulo social de Kartik, que nos permite usar sus widgets sociales. Si recuerda, incluimos:

```
"kartik-v/yii2-social": "dev-master",
```

en nuestro archivo composer.json. Ahora necesitamos decirle a config que se encuentra allí. El archivo original common/config/main.php es:

```
<?php
return [
    'vendorPath' => dirname(dirname(__DIR__)) . '/vendor',
    'components' => [
        'cache' => [
            'class' => 'yii\caching\FileCache',
        ],
    ],
];
```

Si no ve eso, asegúrese de que tiene el archivo correcto. Yii2 tiene varios archivos llamados main.php en config. Necesita aquel que se encuentra en el directorio common. Necesita modificarlo a:

Gist:

[Actualización de Common/Config/Main](#)

Del libro:

```
<?php
return [
    'vendorPath' => dirname(dirname(__DIR__)) . '/vendor',
    'extensions' => require(__DIR__ . '/../../vendor/yiisoft/extensions.php'),
    'modules' => [
        'social' => [
            // la clase del módulo
            'class' => 'kartik\social\Module',
            // la configuración global para el widget Disqus
            'disqus' => [
                'settings' => ['shortname' => 'DISQUS_SHORTNAME'] // conf por defecto
            ],
            // la configuración global para el widget de los plugins de facebook
        ],
    ],
];
```

```
'facebook' => [
    'appId' => 'su id',
    'secret' => 'su secret',
],

// la configuración global para el widget del plugin de google
'google' => [
    'clientId' => 'GOOGLE_API_CLIENT_ID',
    'pageId' => 'GOOGLE_PLUS_PAGE_ID',
    'profileId' => 'GOOGLE_PLUS_PROFILE_ID',
],

// la configuración global para el widget del plugin de google analytic
'googleAnalytics' => [
    'id' => 'TRACKING_ID',
    'domain' => 'TRACKING_DOMAIN',
],

// la configuración global para el widget del plugin de twitter
'twitter' => [
    'screenName' => 'TWITTER_SCREEN_NAME'
],
],
// sus otros módulos
],

'components' => [
    'cache' => [
        'class' => 'yii\caching\FileCache',
    ],
],
];

```

Tome su app id de Facebook y su secret y copielos en:

```
// la configuración global para el widget del plugin social de facebook

'facebook' => [
    'appId' => 'su id',
    'secret' => 'su secret',
],
]
```

Por favor tenga en cuenta que cuando ve los widgets de facebook implementados en mis ejemplos, uso Yii2Build.com como mi dominio. Obviamente usted usará su propio dominio de ejemplo.

Extensiones

Sólo una nota sobre como es referenciado el módulo social de Kartik en config. Bajo vendor/yiisoft existe un archivo llamado extensions.php y este contiene el alias para el módulo:

```
'kartik-v/yii2-social' =>
array (
    'name' => 'kartik-v/yii2-social',
    'version' => '9999999-dev',
    'alias' =>
array (
    '@kartik' => $vendorDir . '/kartik-v/yii2-social',
),
),
```

Puede ver que kartik-v/yii2-social es el mismo nombre que aparecía en composer.json cuando lo incluimos allí. Composer automáticamente lo ingresó por nosotros en el archivo de extensiones. También note que el arreglo para los alias '@kartik' => \$vendorDir . '/kartik-v/yii2-social , que permite esta línea en:

```
return [
    'vendorPath' => dirname(dirname(__DIR__)) . '/vendor',
    'extensions' => require(__DIR__ . '/../../vendor/yiisoft/extensions.php'),
    'modules' => [
        'social' => [
            // la clase del módulo

        'class' => 'kartik\\social\\Module',
    ],
]
```

Incluí la ruta a vendor como referencia, pero eso no es nuevo, no la agregamos, ya se encontraba allí. Sí le indicamos que use extensions en la ruta especificada en el config y allí es donde encuentra el alias que conecta todo.

Así que básicamente, dice que para el módulo social, use la clase kartik\\socialModule. No hay un directorio kartik, pero kartik es un alias para \$vendorDir . '/kartik-v/yii2-social', que cuando se combina con /social/Module, provee la ubicación de la clase.

El widget real se llama FacebookPlugin y es el que se encuentra referenciado con su espacio de nombres en la parte superior del archivo:

```
use kartik\social\FacebookPlugin;
```

El alias usado en la extensión funciona también en el espacio de nombres. Así que así es como Yii sabe donde hallar todo. En fin, la página index del sitio ya no debería devolver un error, una vez que tenga los cambios mencionados.

Helper HTML

Bien, de regreso al archivo index.php del sitio. Notemos que he hecho que el botón ‘Get Started Today’ se muestre sólo si no hay una sesión iniciada, no hay necesidad de mostrarlo si está logueado, ya que tiene un link al formulario de registro. También, he usado la clase HTML que identificamos en nuestra sentencia Use:

```
use yii\helpers\Html;
```

Luego usé el método de la clase Html para formatear el link:

```
<p>

<?php

if (Yii::$app->user->isGuest){

    echo Html::a('Get Started Today', ['site/signup'],
        ['class' => 'btn btn-lg btn-success']);

}

?>

</p>
```

3 parámetros para el método a, el primero es el texto del link, ‘Comience Hoy’ El siguiente es el controlador/acción, ‘site/signup’ en este caso. Y luego tenemos la clase para el css, que, como vimos en el código, es un botón.

Un par de cosas a tener en cuenta. El método a es bastante listo. Sabe que si se encuentra en un archivo de perfil view.php, simplemente puede pasarle la acción que desea, por ejemplo, update, y sabrá la ruta a la acción update correcta. También, tenga en cuenta que puede agregar, separada por una coma, una opción de parámetro en ese arreglo para pasar una variable get, por ejemplo:

```
[ 'update', 'id' => $model->id],
```

Bien, Ok, descendiendo por el resto de la página, remueva este código:

```
<div class="body-content">

    <div class="row">
        <div class="col-lg-4">
            <h2>Heading</h2>

            <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed
                do eiusmod tempor incididunt ut labore et
                dolore magna aliqua. Ut enim ad minim veniam, quis nostrud
                exercitation ullamco laboris nisi ut aliquip
                ex ea commodo consequat. Duis aute irure dolor in
                reprehenderit in voluptate velit esse cillum dolore eu
                fugiat nulla pariatur.</p>

        <p><a class="btn btn-default" href="http://www.yiiframework.com/doc/">
            Yii Documentation &raquo;</a></p>
        </div>
        <div class="col-lg-4">
            <h2>Heading</h2>

            <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed
                do eiusmod tempor incididunt ut labore et
                dolore magna aliqua. Ut enim ad minim veniam, quis nostrud
                exercitation ullamco laboris nisi ut aliquip
                ex ea commodo consequat. Duis aute irure dolor in
                reprehenderit in voluptate velit esse cillum dolore eu
                fugiat nulla pariatur.</p>

        <p><a class="btn btn-default" href="http://www.yiiframework.com/forum/">
            Yii Forum &raquo;</a></p>
        </div>
        <div class="col-lg-4">
            <h2>Heading</h2>

            <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed
```

```

do eiusmod tempor incididunt ut labore et
dolore magna aliqua. Ut enim ad minim veniam, quis nostrud
exercitation ullamco laboris nisi ut aliquip
ex ea commodo consequat. Duis aute irure dolor in
reprehenderit in voluptate velit esse cillum dolore eu
fugiat nulla pariatur.</p>

<p><a class="btn btn-default" href="http://www.yiiframework.com/extensions/">
    Yii Extensions &raquo;</a></p>
</div>
</div>

</div>
</div>

```

Reemplacelo con:

Gist:

[Resto del Index del Sitio](#)

Del libro:

```

<?php

echo Collapse::widget([
    'items' => [
        [
            'label' => 'Características Principales' ,
            'content' => FacebookPlugin::widget([
                'type'=>FacebookPlugin::SHARE,
                'settings' => ['href'=>'http://www.yii2build.com','width'=>'350']

            ]),
            // open its content by default
            //'contentOptions' => ['class' => 'in']
        ],
    ],
]

```

```
// another group item

[

    'label' => 'Recursos Principales',
    'content' => FacebookPlugin::widget([
        'type'=>FacebookPlugin::SHARE,
        'settings' => ['href'=>'http://www.yii2build.com','width'=>'350']

    ]),


    // 'contentOptions' => [],
    // 'options' => [],

],


]

]);


Modal::begin([
    'header' => '<h2>Últimos Comentarios</h2>',
    'toggleButton' => ['label' => 'comentarios'],



]);



echo FacebookPlugin::widget([
    'type'=>FacebookPlugin::COMMENT,
    'settings' => ['href'=>'http://www.yii2build.com','width'=>'350']


]);



Modal::end();



?>



<br/>
```

```
<br/>

<?Pphp

echo Alert::widget([
    'options' => [
        'class' => 'alert-info',
    ],
    'body' => 'Lance su proyecto como un cohete...',
]);
?>

<div class="body-content">

    <div class="row">
        <div class="col-lg-4">
            <h2>Gratis</h2>

            <p>
                <?php

                if (!Yii::$app->user->isGuest) {

                    echo Yii::$app->user->identity->username . ' está haciendo cosas geniales.';
                }

            ?>
        </div>
    </div>
</div>
```

Partiendo de esta plantilla de código abierto y gratuita de Yii 2 ahorrará mucho tiempo.

Puede entregar proyectos al cliente rápidamente, con mucho de código ya disponible, para que pueda concentrarse en los asuntos complejos.</p>

```
<p>

    <a class="btn btn-default"
        href="http://www.yiiframework.com/doc-2.0/guide-index.html">
        Documentación de Yii &raquo;</a>

</p>
```

```
<?php  
  
echo FacebookPlugin::widget([  
  
    'type'=>FacebookPlugin::LIKE,  
    'settings' => []  
  
]);
```

```
?>
```

```
</div>  
<div class="col-lg-4">  
    <h2>Ventajas</h2>  
  
<p>
```

La facilidad de uso es una enorme ventaja. Hemos simplificado el RBAC y le hemos otorgado tipos de usuario Gratuito/Pago de manera predeterminada. Los plugins sociales son tan sencillos y rápidos de instalar, ¡que los amará!

```
</p>
```

```
<p>
```

```
    <a class="btn btn-default"  
    href="http://www.yiiframework.com/forum/">  
        Foro de Yii &raquo;</a>
```

```
</p>
```

```
<?php
```

```
echo FacebookPlugin::widget([  
  
    'type'=>FacebookPlugin::COMMENT,  
    'settings' => [ 'href'=>'http://www.yii2build.com', 'width'=>'350' ]  
  
]);
```

```
?>
```

```
</div>
<div class="col-lg-4">
    <h2>iCodifique Rápidamente, Codifique Correctamente!</h2>
```

```
<p>
```

Libere el poder del asombroso framework Yii 2 con esta plantilla mejorada. Con base en la plantilla avanzada de Yii 2, obtiene una implementación de frontend y backend completa que presenta una IU rica para la administración del backend.

```
</p>
```

```
<p>
```

```
    <a class="btn btn-default"
        href="http://www.yiiframework.com/extensions/">
        Extensiones de Yii &raquo;</a>
```

```
</p>
```

```
</div>
```

```
</div>
```

```
</div>
```

```
</div>
```

Simplemente un recordatorio, y esto no es parte del código o página, no olvide guardar. También, asegúrese de que usa su dominio en los widgets sociales, no [Yii2build.com](#).

Widget Collapse

Puede ver en el código de arriba que he referenciado un widget collapse, así que necesita agregar esa sentencia use al principio del archivo:

```
use \yii\bootstrap\Collapse;
```

Lo que le permite referenciar el widget directamente a través de una llamada estática:

```
echo Collapse::widget
```

El código del widget es bastante sencillo de seguir:

```
<?php

echo Collapse::widget([
    'items' => [
        [
            'label' => 'Características Principales' ,
            'content' => FacebookPlugin::widget([
                'type'=>FacebookPlugin::SHARE,
                'settings' => ['href'=>'http://www.yii2build.com', 'width'=>'350']
            ]),
            // open its content by default
            // 'contentOptions' => ['class' => 'in']
        ],
        // another group item
        [
            'label' => 'Recursos Principales',
            'content' => FacebookPlugin::widget([
                'type'=>FacebookPlugin::SHARE,
                'settings' => ['href'=>'http://www.yii2build.com', 'width'=>'350']
            ]),
            // 'contentOptions' => [],
            // 'options' => []
        ],
    ]
]);
```

Establecemos las etiquetas de los items, ‘Características Principales’ y ‘Recursos Principales’ y

luego para el contenido agregamos FacebookPlugin::widget para compartir. Las otras opciones están comentadas.

Widget Modal

Luego de ello tenemos un modal con un botón para enviar comentarios de facebook, nuevamente un código bastante sencillo de comprender:

```
Modal::begin([
    'header' => '<h2>Últimos Comentarios</h2>',
    'toggleButton' => ['label' => 'comments'],
]);

echo FacebookPlugin::widget([
    'type'=>FacebookPlugin::COMMENT,
    'settings' => ['href'=>'http://www.yii2build.com','width'=>'350']
]);

Modal::end();
```

Widget Alert

Luego simplemente quería jugar un poco con un widget alert, así que incluí:

```
<?Php

echo Alert::widget([
    'options' => [
        'class' => 'alert-info',
    ],
    'body' => 'Lance su proyecto como un cohete...',
]);
?>
```

Probablemente no es la forma en que usaríamos realmente un alert, estaría ligado a una acción, pero incluirlo nos da una idea de cómo se ve.

Luego la última cosa que hicimos en index.php fue esta línea en el primer <p>

```
<p>

<?php

if (!Yii::$app->user->isGuest) {

echo Yii::$app->user->identity->username . ' está haciendo cosas geniales. ';
}

?>
```

Simplemente otra pequeña diferencia para probar iniciando y cerrando sesión y asegurarnos de que está obteniendo el username correcto. Y eso es todo para nuestra plantilla de la página index.

Font-Awesome

Yii 2 tiene un conjunto un poco complejo de métodos para manejar assets tales como bootstrap, jquery, etc., and they did this to maximize efficiency by caching resources. I'm not going to cover it much in this book, but we will dabble in it to the extent that we want to have access to font-awesome, a popular css icon library.

Ok, so here's what we need to make this work.

First, if you have not already done so, we are going to pull in font-awesome via composer, add the following to your composer.json file:

```
"minimum-stability": "stable",
"require": {
    "php": ">=5.4.0",
    "yiisoft/yii2": "*",
    "yiisoft/yii2-bootstrap": "*",
    "yiisoft/yii2-swiftmailer": "*",
    "kartik-v/yii2-social": "dev-master",
    "fortawesome/fontawesome": "4.2.0"
},
```

Ahora si mira con atención, es fortawesome, no es un error de tipeo. Continúe y ejecute composer update y eso lo agregará por usted.

Asset Bundle

Luego necesita crear y agregar la siguiente línea, en dos ocasiones, frontend/assets y backend/assets, con sus correspondientes espacios de nombres. El nombre del archivo es FontAwesomeAsset.php. Aquí está el archivo completo:

Gist:

[FontAwesomeAsset.php file](#)

Del libro:

```
<?php

/**
 * @link http://www.yiiframework.com/
 * @copyright Copyright (c) 2008 Yii Software LLC
 * @license http://www.yiiframework.com/license/
 */

namespace frontend\assets;

use yii\web\AssetBundle;

/**
 * @author Joao Marques<joao@jmf.com>
 */

class FontAwesomeAsset extends AssetBundle
{
    # sourcePath points to the composer package.

    public $sourcePath = '@vendor/fortawesome/fontawesome';

    # CSS file to be loaded.
    public $css = [
        'css/fontawesome.min.css',
    ];

    /**

```

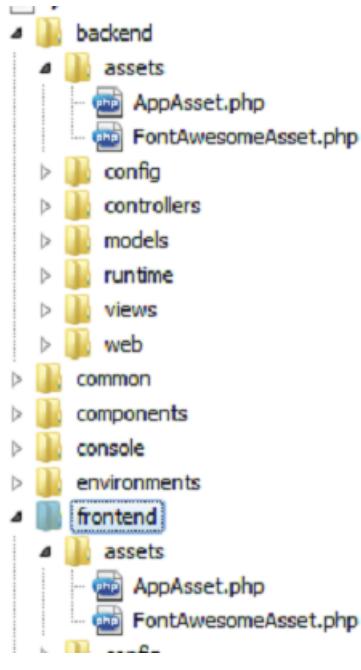
```
* Sets the publishOptions property.  
* Needed because it's necessary to  
*concatenate  
* the namespace value.  
*/  
  
public function init()  
{  
    $this->publishOptions = [  
        'forceCopy' => YII_DEBUG,  
        'beforeCopy' => __NAMESPACE__ .  
            '\FontAwesomeAsset::filterFolders'  
    ];  
  
    parent::init();  
}  
  
/**  
 * Filters the published files and folders.  
 * It's not necessary publish all files and folders  
 * from the font-awesome package  
 * Just the CSS and FONTS folder.  
 * @param string $from  
 * @param string $to  
 * @return bool true to publish to file/folder.  
 */  
  
public static function filterFolders($from, $to)  
{  
    $validFilesAndFolders = [  
        'css',  
        'fonts',  
        'font-awesome.css',  
        'font-awesome.min.css',  
        'FontAwesome.otf',  
        'fontawesome-webfont.eot',  
        'fontawesome-webfont.svg',  
        'fontawesome-webfont.ttf',  
        'fontawesome-webfont.woff',  
    ];
```

```
$pathItems = array_reverse(explode(DIRECTORY_SEPARATOR, $from));

if (in_array($pathItems[0], $validFilesAndFolders)) return true;
else return false;
}
```

Simplemente copie una segunda versión a backend\assets y establezca el espacio de nombre de esa copia a:

```
namespace backend\assets;
```



FontAwesomeAsset

Así que este es un asset bundle. Debo señalar que obtuve esto de un foro de Yii 2, el autor del archivo está en la parte superior del archivo. Él realizó un trabajo maravilloso comentando el código:

<http://www.yiiframework.com/forum/index.php/topic/57902-using-fontawesome/>

También puede averiguar más sobre el asunto en la guía de Yii 2:

<http://www.yiiframework.com/doc-2.0/guide-structure-assets.html>

Agregar Font-Awesome al Layout

En frontend/views/layout/main.php, agregue la siguiente sentencia use en la parte superior:

```
use frontend\assets\FontAwesomeAsset;
```

Y luego debajo, cerca de la otra llamada a register agregue una llamada adicional de esta forma:

```
AppAsset::register($this);
FontAwesomeAsset::register($this);
```

Y eso debería ser suficiente, deberíamos tener acceso a font-awesome. Así que pongámoslo a prueba insertando lo siguiente:

```
<i class="fa fa-plug"></i>
```

Vamos a ponerlo en dos lugares:

En main.php, el primer div, ‘brand label’:

```
'brandLabel' => 'Yii 2 Build <i class="fa fa-plug"></i>',
```

Luego dirigímosnos a frontend\views\site\index.php y agreguemoslo en la primera etiqueta `<h1>` así:

```
<h1>Yii 2 Build <i class="fa fa-plug"></i></h1>
```

Eso es todo. Ahora debería tener una página de inicio que se ve así:

The screenshot shows the Yii 2 Build homepage. At the top, there's a dark header with 'Yii 2 Build' on the left and navigation links for Home, About, Contact, Signup, and Login on the right. A large green button labeled 'Comience Hoy' (Start Today) is centered above the main content. The main title 'Yii 2 Build' is displayed with a plug icon, followed by the subtitle 'Use esta plantilla de Yii 2 para comenzar Proyectos.' Below this, there's a Facebook 'Like' button with the text '6 people like this. Be the first of your friends.' A section titled 'Características Principales' lists 'Recursos Principales' and has a 'comentarios' button. A comment input field says 'Lance su proyecto como un cohete...' with a close button ('X'). On the left, a 'Gratis' section describes the template as open-source and time-saving, with a 'Documentación de Yii »' button. On the right, a 'Ventajas' section highlights ease of use, RBAC simplification, and built-in payment methods, with a 'Foro de Yii »' button. A comment form includes a placeholder 'Add a comment...', a 'Also post on Facebook' checkbox, and a 'Extensiones de Yii »' button.

Y simplemente con el propósito de resolver problemas, proveeré del archivo frontend/views/site/index.php completo como referencia. No debería necesitar hacer nada en este punto, pero si se ha perdido de algo, puede referirse a este archivo.

Gist:

Frontend Site Index

Del libro:

```
<?php

use \yii\bootstrap\Modal;
use kartik\social\FacebookPlugin;
use \yii\bootstrap\Collapse;
use \yii\bootstrap\Alert;
use yii\helpers\Html;

/* @var $this yii\web\View */
```

```
$this->title = 'Yii 2 Build';
?>
<div class="site-index">

    <div class="site-index">

        <div class="jumbotron">

            <?php if (Yii::$app->user->isGuest) {
                echo Html::a('Get Started Today', ['site/signup'],
                    ['class' => 'btn btn-lg btn-success']);
            }
?>
        </p>

        <h1>Yii 2 Build <i class="fa fa-plug"></i></h1>

        <p class="lead">Use this Yii 2 Template to start Projects.</p>

        <br/>

            <?php echo FacebookPlugin::widget(['type'=>FacebookPlugin::LIKE,
                'settings' => []]); ?>

</div>

<?php

echo Collapse::widget([
    'items' => [
        [
            'label' => 'Características Principales' ,
            'content' => FacebookPlugin::widget([
                'type'=>FacebookPlugin::SHARE,
                'settings' => ['href'=>'http://www.yii2build.com','width'=>'350']
            ]
        ]
    ]
]); ?>
```

```
]),

// open its content by default
//'contentOptions' => ['class' => 'in']

],

// another group item

[
    'label' => 'Recursos Principales',
    'content' => FacebookPlugin::widget([
        'type'=>FacebookPlugin::SHARE,
        'settings' => ['href'=>'http://www.yii2build.com','width'=>'350']
    ]),
    // 'contentOptions' => [],
    // 'options' => [],
],

])

]);


Modal::begin([
    'header' => '<h2>Últimos Comentarios</h2>',
    'toggleButton' => ['label' => 'comentarios'],
]);

echo FacebookPlugin::widget([
    'type'=>FacebookPlugin::COMMENT,
    'settings' => ['href'=>'http://www.yii2build.com','width'=>'350']
])
```

```
    ]);

Modal::end();

?>

<br/>
<br/>

<?Php

echo Alert::widget([
    'options' => [
        'class' => 'alert-info',
    ],
    'body' => 'Lance su proyecto como un cohete...', 
]);
?>

<div class="body-content">

    <div class="row">
        <div class="col-lg-4">
            <h2>Gratis</h2>

            <p>
                <?php

if (!Yii::$app->user->isGuest) {

    echo Yii::$app->user->identity->username . ' esta haciendo cosas geniales.';
}

?>
```

Partiendo de esta plantilla de código abierto y gratuita de Yii 2 ahorrará mucho tiempo. Puede entregar proyectos al cliente rápidamente, con mucho de código ya disponible, para que pueda concentrarse en los asuntos complejos.</p>

```
<p>
```

```
  <a class="btn btn-default"  
      href="http://www.yiiframework.com/doc-2.0/guide-index.html">  
    Documentación de Yii &raquo;</a>
```

```
</p>
```

```
<?php
```

```
echo FacebookPlugin::widget([  
  
  'type'=>FacebookPlugin::LIKE,  
  'settings' => []  
]);
```

```
?>
```

```
  </div>  
  <div class="col-lg-4">  
    <h2>Ventajas</h2>
```

```
<p>
```

La facilidad de uso es una enorme ventaja. Hemos simplificado el RBAC y le hemos otorgado tipos de usuario Gratuito/Pago de manera predeterminada. Los plugins sociales son tan sencillos y rápidos de instalar, ¡que los amará!

```
</p>
```

```
<p>
```

```
  <a class="btn btn-default"  
      href="http://www.yiiframework.com/forum/">  
    Foro de Yii &raquo;</a>
```

```
</p>
```

```
<?php
```

```
echo FacebookPlugin::widget([
```

```
'type'=>FacebookPlugin::COMMENT,
'settings' => [ 'href'=>'http://www.yii2build.com', 'width'=>'350']

]);
?>
</div>
<div class="col-lg-4">
    <h2> ¡Codifique Rápidamente, Codifique Correctamente! </h2>

<p>
Libere el poder del asombroso framework Yii 2 con esta plantilla mejorada.
Con base en la plantilla avanzada de Yii 2, obtiene una implementación de
frontend y backend completa que presenta una IU rica para la
administración del backend.

</p>

<p>
<a class="btn btn-default"
href="http://www.yiiframework.com/extensions/">
Extensiones de Yii &raquo;</a>

</p>

</div>
</div>

</div>
</div>
```

Resumen



Commit!

En este capítulo, configuramos nuestra app de Facebook e instalamos nuestra extensión social de Kartik para poder usarla. Gracias a la facilidad de uso de composer, instalar la extensión fue sencillo. La extensión de Kartik es muy útil, y como recordatorio, puede revisar muchas de sus grandiosas extensiones en:

<http://www.krajee.com>

Insertamos el widget social dentro de un widget collapsible para divertirnos un poco con la decoración de la página de inicio. Puede obtener más widgets de bootstrap para usar en sus proyectos en:

<http://www.yiiframework.com/doc-2.0/guide-widget-bootstrap.html>

Ya que ya hemos agregado la extensión jui para DatePicker, tenemos acceso a todos los widgets listados en:

<http://www.yiiframework.com/doc-2.0/guide-widget-jui.html>

Yii 2 soporta un gran número de widgets de Jquery y Bootstrap como collapsible y modal. Este libro no es en realidad sobre desarrollo de front end, así que no profundizamos demasiado, pero al menos obtuvo una muestra con la que jugar.

Finalmente, agregamos un app asset para font-awesome, para que pueda agregarle un poco de sabor a la presentación, sin demasiado esfuerzo. Puede encontrar muchos íconos para agregar a su presentación en:

[Font-Awesome](#)

Capítulo Once: Creación del Backend

Muy bien, estamos listos para comenzar con la creación de nuestra área de administración del backend. Antes de que podamos crear todos los archivos que necesitamos, es necesario agregar un directorio dentro de backend/models, queremos agregar un directorio search, así tendremos backend/models/search. Hagamoslo ahora.

Ahora podemos trabajar en con el generador de CRUD de Gii para crear controladores y vistas para todos nuestros modelos. Puede estar preguntándose si podemos usar los controladores y vistas del frontend que ya hemos construido, sin embargo, no es posible.

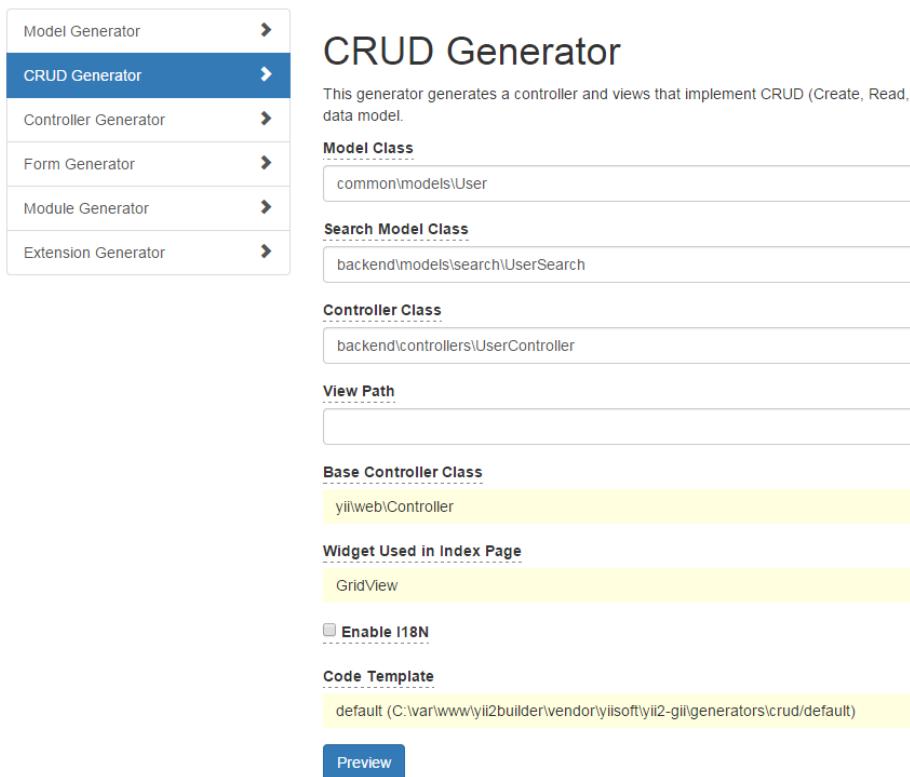
El backend opera diferente al frontend, y esta es la razón de tener una sección de la aplicación separada para él. Señalaré las diferencias a medida que avancemos, pero por ahora, creemos un controlador y el crud para cada uno de los siguientes modelos:

- Perfil
- Rol
- Estado
- User
- TipoUsuario

La url para Gii es:

<http://backend.yii2build.com/?r=gii>

El espacio de nombres para el modelo user es como sigue:



CRUD de User

Si la imagen de arriba no es clara, el ejemplo luce así:

Model Class: common\models\User

Search Model Class: backend\models\search\UserSearch

Controller Class: backend\controllers\UserController

Referenciamos common\models\user, ya que es allí donde se encuentra nuestro modelo user, pero estamos creando todos estos otros archivos en el backend. Puede ver que también necesitamos proveer de una entrada para un modelo de búsqueda (search model). Asegúrese de que ha creado el directorio search dentro de backend/models antes de ejecutar Gii.

Yii2 proporciona una clase separada para los parámetros de búsqueda y me alegra mucho que hicieran esto. Separa mucho código del modelo base, lo que lo vuelve más fácil de seguir. Verá como funciona luego.

Note que podemos dejar view path en blanco porque estamos adhiriendo a los valores por defecto.

Para asegurarnos de que tiene los nombres de espacio correctos y de que las ubicaciones de los archivos han sido generadas, listaré todos los modelos restantes a partir de los cuales deberá generar el CRUD, con su espacios de nombres especificados.

Perfil:

Model Class: frontend\models\Perfil

Search Model Class: backend\models\search\PerfilSearch

Controller Class: backend\controllers\PerfilController

User:

Model Class: common\models\User

Search Model Class: backend\models\search\UserSearch

Controller Class: backend\controllers\UserController

Rol:

Model Class: backend\models\Rol

Search Model Class: backend\models\search\RolSearch

Controller Class: backend\controllers\RolController

Estado:

Model Class: backend\models\Estado

Search Model Class: backend\models\search\EstadoSearch

Controller Class: backend\controllers\EstadoController

Tipo Usuario:

Model Class: backend\models\TipoUsuario

Search Model Class: backend\models\search\TipoUsuarioSearch

Controller Class: backend\controllers\TipoUsuarioController

Ya que el proceso de crear el CRUD es exactamente el mismo para cada modelo de los listados arriba, no veremos cada uno de ellos aquí. En este punto, simplemente asumiremos que ha creado los archivos y continuaremos desde allí.

Una cosa a tener en cuenta. La convención de nombres para las vistas con múltiples palabras en el nombre es poner un - entre las dos palabras, así que el directorio de las vistas para TipoUsuario es tipo-usuario. Las urls para los controladores son de la misma forma. Aún cuando el archivo del controlador se llama TipoUsuarioController, para alcanzarlo por la url, debería llamar, por ejemplo:

```
yii2build.com/index.php?r=tipo-usuario/index
```

Bien, puede comprobar los resultados individualmente escribiendo una url, por ejemplo:

```
backend.yii2build.com/index.php?r=user
```

Esto llamará a la acción index y asumiendo que tiene al menos un registro allí, lo mostrará. Obviamente, deberá iniciar sesión en backend.yii2build.com y el usuario con el que inicia sesión deberá tener un rol_id que concuerde con un rol llamado 'Admin,' ya que nuestro PermisosHelper impone esa regla.

La página index de user también tiene links a view y update y delete, esos son los íconos que ve a la derecha de la grilla, así que puede verificar si estos también funcionan. Luego pruebe la misma url de arriba con los diferentes modelos que ha creado para asegurarse de que todo funciona.

Si obtiene errores, compruebe las ubicaciones de sus archivos y verifique los espacios de nombres en cada uno de ellos. Por ejemplo, en PerfilController.php:

```
namespace backend\controllers;

use Yii;
use frontend\models\Perfil;
use backend\models\search\PerfilSearch;
use yii\web\Controller;
use yii\web\NotFoundHttpException;
use yii\filters\VerbFilter;
```

Puede ver que usa frontend\models para el modelo, pero usa backend\models\search para el modelo de búsqueda.

Si bien es posible duplicar un modelo en más de una ubicación, definitivamente no se recomienda, ya que interfiere con el propósito de tener un directorio common y viola el principio DRY.

Usé frontend\models para Perfil, pero podría haber usado common\models. Simplemente elegí lo primero debido al flujo de trabajo. Pero puede ver lo fácil que es hacer referencia al directorio a través del espacio de nombres cuando está creando el controlador y las vistas.

Vamos a realizar algunos cambios a los archivos de las vistas y al modelo de búsqueda, pero los controladores permanecerán mayormente sin cambios, con excepción de los behaviors. Esto se debe a que los controladores y vistas que Gii crea se prestan a un enfoque del backend, esto significa un usuario logueado que puede buscar una lista de usuarios, actualizar todos los registros, etc. Los controladores predefinidos permiten esto, así que las buenas noticias son que la auto-generación del código es realmente útil y ahorra mucho tiempo.

Así que simplemente al controlar el acceso al área de administración al requerir un valor mínimo a rol_id en el registro user, sólo los usuarios con un nivel de administrador pueden acceder a los controladores del backend. También demostraremos lo fácil que es agregar un rol por encima de

admin que pueda modificar registros que admin no puede. En nuestra plantilla, los usuarios con nivel de Administrador puede usar la IU del backend para modificar registros de usuario.

Por supuesto incluiremos alguna restricción, por ejemplo, admin no será capaz de modificar las contraseñas de los usuarios, ni siquiera las verán. Realizaremos varios cambios a nuestra IU del backend para que sea más fácil desplazarse por registros relacionados, por ejemplo, desearemos tener acceso al rol_id de un usuario así como la habilidad de cambiarlo, para que podamos otorgar privilegios adicionales a los usuarios.

Puede ver lo poderoso que Yii 2 es en realidad al permitirnos configurar todo esto rápidamente. Y debido a que nos tomamos el tiempo anteriormente para construir modelos para cosas como rol y estado, ahora vamos a tener una IU completa para controlarlos.

Antes de que cambiemos los archivos de vistas individuales, realicemos algunos cambios a backend/views/layout/main.php. Agregaremos numerosos links para que nos sea más sencillo navegar a través de diferentes vistas.

Main.php

Diríjase a backend/views/layout/main.php.

Cambie main.php a:

Gist:

[cambio de la vista principal 1](#)

```
<?php

use backend\assets\AppAsset;
use yii\helpers\Html;
use yii\bootstrap\Nav;
use yii\bootstrap\NavBar;
use yii\widgets\Breadcrumbs;
use common\models\PermisosHelpers;
use backend\assets\FontAwesomeAsset;

/**
 * @var \yii\web\View $this
 * @var string $content
 */

AppAsset::register($this);
FontAwesomeAsset::register($this);
```

```
?>

<?php $this->beginPage() ?>

<!DOCTYPE html>

<html lang="= Yii::$app-&gt;language ?&gt;"&gt;

&lt;head&gt;
&lt;meta charset="<?= Yii::$app-&gt;charset ?&gt;"/&gt;

&lt;meta name="viewport"
content="width=device-width,
initial-scale=1"&gt;

&lt;?= Html::csrfMetaTags() ?&gt;

&lt;title&gt;&lt;?= Html::encode($this-&gt;title) ?&gt;&lt;/title&gt;

&lt;?php $this-&gt;head() ?&gt;

&lt;/head&gt;

&lt;body&gt;
&lt;?php $this-&gt;beginBody() ?&gt;

&lt;div class="wrap"&gt;

&lt;?php

if (!Yii::$app-&gt;user-&gt;isGuest){

$es_admin = PermisosHelpers::requerirMinimoRol('Admin');

NavBar::begin([
'brandLabel' =&gt; 'Yii 2 Build &lt;i class="fa fa-plug"&gt;&lt;/i&gt; Admin',
'brandUrl' =&gt; Yii::$app-&gt;homeUrl,
'options' =&gt; [
</pre
```

```
        'class' => 'navbar-inverse navbar-fixed-top',
    ],
]);
}

} else {

NavBar::begin([
    'brandLabel' => 'Yii 2 Build <i class="fa fa-plug"></i>',
    'brandUrl' => Yii::$app->homeUrl,
    'options' => [
        'class' => 'navbar-inverse navbar-fixed-top',
    ],
]);
}

$menuItems = [
    ['label' => 'Home', 'url' => ['site/index']],
];
}

if (!Yii::$app->user->isGuest && $es_admin) {

$menuItems[] = ['label' => 'Usuarios', 'url' => ['user/index']];
$menuItems[] = ['label' => 'Perfiles', 'url' => ['perfil/index']];
$menuItems[] = ['label' => 'Roles', 'url' => ['rol/index']];
$menuItems[] = ['label' => 'Tipos de Usuario', 'url' => ['tipo-usuario/index']];
$menuItems[] = ['label' => 'Estados', 'url' => ['estado/index']];
}

if (Yii::$app->user->isGuest) {

$menuItems[] = ['label' => 'Login', 'url' => ['site/login']];
} else {
```

```
$menuItems[] = [
    'label' => 'Logout (' . Yii::$app->user->identity->username . ')',
    'url' => ['/site/logout'],
    'linkOptions' => ['data-method' => 'post']
];
}

echo Nav::widget([
    'options' => ['class' => 'navbar-nav navbar-right'],
    'items' => $menuItems,
]);

NavBar::end();

?>

<div class="container">

<?= Breadcrumbs::widget([
    'links' => isset($this->params['breadcrumbs']) ? $this->params['breadcrumbs'] : [],
])?>

<?= $content ?>

</div>
</div>

<footer class="footer">

<div class="container">

<p class="pull-left">&copy; Yii 2 Build <?= date('Y') ?></p>

<p class="pull-right"><?= Yii::powered() ?></p>
```

```
</div>

</footer>

<?php $this->endBody() ?>

</body>
</html>

<?php $this->endPage() ?>
```

Notarás que he usado mucho espacio en blanco para que el código sea más fácil de leer. Tenga en cuenta, que la legibilidad se ve impactada al estar en un libro.

También agregamos una sentencia use:

```
use common\models\PermisosHelpers;
```

Lo hicimos porque vamos a usar PermisosHelpers::requerirMinimoRol('Admin'); para agregar una capa de seguridad extra.

Puedes ver que en lugar de simplemente añadir los links a las diferentes vistas, los escondemos si de alguna manera el usuario ha llegado a esta página, pero no tiene un acceso con nivel de admin:

```
if (!Yii::$app->user->isGuest && $es_admin) {

    $menuItems[] = ['label' => 'Users', 'url' => ['user/index']];
}
```

La variable \$es_admin contiene el valor de:

```
PermisosHelpers::requerirMinimoRol('Admin');
```

La lógica detrás de esto es que si no se trata de un invitado y el usuario actual tiene una rol_id mayor o igual a la que se necesita para 'admin,' muestra el link. Es una forma muy simple de controlar el acceso a los links.

```

if (!Yii::$app->user->isGuest && $es_admin) {

    $menuItems[] = ['label' => 'Usuarios', 'url' => ['user/index']];

    $menuItems[] = ['label' => 'Profiles', 'url' => ['profile/index']];

    $menuItems[] = ['label' => 'Roles', 'url' => ['/role/index']];

    $menuItems[] = ['label' => 'Tipos de Usuario', 'url' => ['/user-type/index']];

    $menuItems[] = ['label' => 'Estados', 'url' => ['/status/index']];
}

```

Usamos el arreglo \$menuItems para contener la url debido a que estamos trabajando con el widget NavBar.

Note, pasamos por alto el bloque donde agregamos una sentencia if para ver si el usuario estaba logueado o no, luego mostramos ya sea:

```
'brandLabel' => 'Yii 2 Build Admin',
```

O, si no se encuentra logueado:

```
'brandLabel' => 'Yii 2 Build',
```

Hice eso simplemente porque aunque es sólo cosmético, ni siquiera quiero dar a conocer la palabra 'admin' a los usuarios que no han iniciado sesión.

Modificando las Vistas del Backend

Para completar nuestra funcionalidad básica del backend, necesitamos actualizar los formularios y las grillas de las vistas. En el caso de los formularios, como lo hicimos en las vistas del frontend, necesitamos agregar opciones de listas desplegables y remover los campos no deseados.

Los cambios a GridView, que es el nombre del widget principal en Index.php, son un poco más profundos. Vamos a cambiar las columnas que se muestran, así como agregar columnas para modelos relacionados, para que, por ejemplo, una lista de usuarios tenga un link a los perfiles. En la grilla de usuario, deberíamos mostrar el nombre de rol del usuario, etc. Estas son cosas pequeñas, que harán que la IU sea más útil y fácil de comprender.

backend/views/perfil/_form.php

Comencemos con la más sencilla primero. Simplemente copiaremos frontend/views/perfil/_form.php a backend/views/perfil/_form.php.

Ahora tenemos el datepicker de jui y la lista desplegable para genero en el formulario, que exactamente lo que queríamos.

backend/views/perfil/view.php

Bien, ahora trabajemos en view.php para backend/views/perfil/view.php. Esto es lo que Gii nos entregó de manera predeterminada:

```
<?php

use yii\helpers\Html;
use yii\widgets\DetailView;

/* @var $this yii\web\View */
/* @var $model frontend\models\Perfil */

$this->title = $model->id;
$this->params['breadcrumbs'][] = ['label' => 'Perfiles', 'url' => ['index']];
$this->params['breadcrumbs'][] = $this->title;
?>
<div class="perfil-view">

    <h1><?= Html::encode($this->title) ?></h1>

    <p>
        <?= Html::a('Update', ['update', 'id' => $model->id],
                    ['class' => 'btn btn-primary']) ?>
        <?= Html::a('Delete', ['delete', 'id' => $model->id],
                    ['class' => 'btn btn-danger',
                     'data' => [
                         'confirm' => 'Are you sure you want to delete this item?',
                         'method' => 'post',
                     ],
                ]) ?>
    </p>

    <?= DetailView::widget([
        'model' => $model,
        'attributes' => [
            'id',
            'user_id',
            'nombre:text',
            'apellido:text',
            'fecha_nacimiento',
            'genero_id',
        ]
    ]) ?>
</div>
```

```

        'created_at',
        'updated_at',
    ],
]) ?>

</div>

```

Cambiemoslo por:

Gist:

[View.php de Perfil del Backend](#)

Del libro:

```

<?php

```



```

use yii\helpers\Html;
use yii\widgets\DetailView;
use common\models\PermisosHelpers;

/**
 * @var yii\web\View $this
 * @var frontend\models\Perfil $model
 */

$this->title = $model->user->username;

$mostrar_esta_nav = PermisosHelpers::requerirMinimoRol('SuperUsuario');

$this->params['breadcrumbs'][] = ['label' => 'Perfiles', 'url' => ['index']];
$this->params['breadcrumbs'][] = $this->title;
?>
<div class="perfil-view">

    <h1>Perfil: <?= Html::encode($this->title) ?></h1>

    <p>

<?php if (!Yii::$app->user->isGuest && $mostrar_esta_nav) {
    echo Html::a('Update', ['update', 'id' => $model->id],
        ['class' => 'btn btn-primary']);?>

```

```
<?php if (!Yii::$app->user->isGuest && $mostrar_esta_nav) {  
    echo Html::a('Delete', ['delete', 'id' => $model->id], [  
        'class' => 'btn btn-danger',  
        'data' => [  
            'confirm' => Yii::t('app', 'Are you sure you want to delete this item?'),  
            'method' => 'post',  
        ],  
    ]);}?>  
  
</p>  
  
<?= DetailView::widget([  
    'model' => $model,  
    'attributes' => [  
        ['attribute'=>'userLink', 'format'=>'raw'],  
        'nombre',  
        'apellido',  
        'fecha_nacimiento',  
        'genero.genero_nombre',  
        'created_at',  
        'updated_at',  
        'id',  
    ],  
])?>  
  
</div>
```

Bien, unos pocos cambios. Incluimos al principio:

```
use common\models\PermisosHelpers;
```

aqui llegue 10/10/23

No daña mantener la consistencia al encerrar los elementos de navegación en sentencias if que verifican si el usuario tiene el permiso para realizar la acción, antes de mostrar los links.

Así que para demostrar esto completamente, vamos a requerir un rol mayor que admin para actualizar o borrar perfiles. Llámemoslo SuperUsuario.

Continúe y cree un usuario en la aplicación y a través de PhpMyAdmin, ingrese un registro de rol para SuperUsuario y asignele al nuevo usuario ese valor. No olvide que debe ser mayor que 20, que es el valor que le asignamos a Admin. Usemos 30 en la tabla rol para rol_valor como ejemplo.

	<input type="button" value="←"/> <input type="button" value="→"/>		<input type="button" value="▼"/>	id	rol_nombre	rol_valor
<input type="checkbox"/>	<input type="button" value="Editar"/>	<input type="button" value="Copiar"/>	<input type="button" value="Borrar"/>	1	Usuario	10
<input type="checkbox"/>	<input type="button" value="Editar"/>	<input type="button" value="Copiar"/>	<input type="button" value="Borrar"/>	2	Admin	20
<input type="checkbox"/>	<input type="button" value="Editar"/>	<input type="button" value="Copiar"/>	<input type="button" value="Borrar"/>	3	SuperUsuario	30

tabla rol

Más adelante, también modificaremos nuestro controlador para impedir que alguien sin acceso de admin acceda a ésta página y me gusta ocultar el link si no se encuentra disponible para el usuario, así tendremos un comportamiento consistente entre la navegación y las reglas de acceso en el controlador.

Bien, continuemos con los cambios, cambiamos el título a:

```
$this->title = $model->user->username;
```

Debajo de él, agregamos una llamada para que el usuario tenga al menos el rol de SuperUsuario para poder ver el link a update:

```
$mostrar_esta_nav = PermisosHelpers::requerirMinimoRol('SuperUsuario');
```

El siguiente cambio está en el tag <h1>:

```
<h1>Perfil: <?= Html::encode($this->title) ?></h1>
```

Ahora muestra el nombre de usuario en lugar del número de id, lo que es mucho más amigable para el usuario.

Luego agregamos nuestros links para update y delete:

```
<p>

<?php if (!Yii::$app->user->isGuest && $show_this_nav) {
    echo Html::a('Update', ['update', 'id' => $model->id],
                 ['class' => 'btn btn-primary']);?>

<?php if (!Yii::$app->user->isGuest && $show_this_nav) {
    echo Html::a('Delete', ['delete', 'id' => $model->id], [
        'class' => 'btn btn-danger',
        'data' => [
            'confirm' => Yii::t('app', 'Are you sure you want to delete this item?'),
            'method' => 'post',
        ],
    ]);?>

</p>
```

Continuando, cambiamos el DetailView::widget. Modificamos el formato del primer atributo a:

```
[ 'attribute'=> 'userLink', 'format'=> 'raw' ] ,
```

Nuestra aplicación sabe a qué estamos haciendo referencia porque agregamos el método getUserLink al modelo Perfil y creamos la etiqueta:

```
'userLink' => Yii::t('app', 'User'),
```

para nuestras etiquetas de atributo en common/models/User.php.

El método en el modelo:

```
public function getUserLink()
{
$url = Url::to(['user/view', 'id'=>$this->id]);
$options = []; //
return Html::a($this->username, $url, $options);
}
```

Este método devuelve el link a la página de visualización del usuario que deseamos. Muestro aquí la etiqueta y el método porque este es el lugar donde probablemente, en su flujo de trabajo, los habría creado, ya que aquí es dónde habría visto que los necesita. Obviamente los construimos anticipadamente, así que ya se encuentran en su lugar. En el futuro, si puede anticipar la necesidad de esta clase de métodos, puede crearlos de con antelación como le hicimos, como parte de los métodos predefinidos que siempre agrega a un modelo, cuando lo crea. Sus decisiones sobre el flujo de trabajo, sin embargo, son cuestión suya.

El otro cambio importante es la relación via lazy loading:

```
'genero.genero_nombre',
```

Esó simplemente le dice que devuelva el atributo genero_nombre del modelo Genero, así que obtenemos ‘masculino’ en lugar de ‘1’, lo que, nuevamente, es mucho más amigable para el usuario. Tenemos acceso a él debido a que en el modelo Perfil, tenemos el siguiente método:

```
public function getGenero()
{
    return $this->hasOne(Genero::className(), ['id' => 'genero_id']);
}
```

En este punto puede estar preguntándose a qué nos referimos por una relación de lazy loading. Una consulta de lazy loading consultará la BD por cada fila de resultados, lo que es muy ineficiente. 1000 resultados requerirían 1001 consultas (lo que también se conoce como el problema n + 1).

Está bien realizar lazy loading cuando sólo hay un resultado, pero debe ser cuidadoso con ello. Demostraremos la versión con eager loading de una consulta cuando construyamos la página Index.

backend/views/user/view.php

Continuando con la página backend/views/user/view.php, modifiquemosla por la siguiente:

Gist:

[Backend User View.php](#)

Del libro:

```
<?php

use yii\helpers\Html;
use yii\widgets\DetailView;
use common\models\PermisosHelpers;

/* @var $this yii\web\View */
/* @var $model common\models\user */

$this->title = $model->username;
$show_this_nav = PermisosHelpers::requerirMinimoRol('SuperUsuario');

$this->params['breadcrumbs'][] = ['label' => 'Users', 'url' => ['index']];
$this->params['breadcrumbs'][] = $this->title;
?>
<div class="user-view">

    <h1><?= Html::encode($this->title) ?></h1>

    <p>

<?php if (!Yii::$app->user->isGuest && $show_this_nav) {
    echo Html::a('Update', ['update', 'id' => $model->id],
        ['class' => 'btn btn-primary']);?>

<?php if (!Yii::$app->user->isGuest && $show_this_nav) {
    echo Html::a('Delete', ['delete', 'id' => $model->id], [
        'class' => 'btn btn-danger',
        'data' => [
            'confirm' => Yii::t('app', 'Are you sure you want to delete this item?'),
            'method' => 'post',
        ],
    ]);?>
```

```

</p>

<?= DetailView::widget([
    'model' => $model,
    'attributes' => [
        ['attribute'=>'perfilLink', 'format'=>'raw'],
        ['username'],
        ['auth_key'],
        ['password_hash'],
        ['password_reset_token'],
        ['email:email'],
        ['rolNombre'],
        ['estadoNombre'],
        ['tipoUsuarioNombre'],
        ['created_at'],
        ['updated_at'],
        ['id'],
    ],
]) ?>

</div>

```

Las sentencias use son exactamente las mismas que las de la página de visualización de perfil. Un pequeño cambio a \$title, ya que queremos mostrar el nombre de usuario, es un atributo del modelo actual:

```
$this->title = $model->username;
```

El resto es el mismo excepto por el DetailView::widget:

```

<?= DetailView::widget([
    'model' => $model,
    'attributes' => [
        ['attribute'=>'perfilLink', 'format'=>'raw'],
        ['username'],
        ['auth_key'],
        ['password_hash'],
        ['password_reset_token'],
        ['email:email'],
        ['rolNombre'],
        ['estadoNombre'],

```

```
'tipoUsuarioNombre',
'created_at',
'updated_at',
'id',
],
]) ?>
```

Nos deshicimos de los campos no deseados que se mostraban comentándolos. Podríamos haberlos cortado definitivamente, pero me gusta dejarlos con propósitos de depuración, si alguna vez necesitara usarlos.

Así que obviamente nuestro primer atributo es un link a perfil, que debido a los métodos que pusimos en el modelo user al comienzo, nos permiten referenciarlos como profileLink. Ya que esto es idéntico a que lo hicimos anteriormente en la vista Perfil, no lo explicaré nuevamente, pero puede mirar los métodos en el modelo user para refreshar su conocimiento.

Note que en el atributo email, usamos ‘email:email’ y esto formatea un link mailto en la dirección cuando aparece en la vista, una característica agradable y práctica.

Vemos que se incluyen ‘rolNombre’ y ‘estadoNombre’ a través de las relaciones definidas en el modelo user.

Cuando compruebe todo esto en su navegador, note lo fácil que es moverse desde las vistas de usuario a las de perfil al tener esos items vinculados. Esta es una buena práctica para la IU y los usuarios finales lo apreciarán.

backend/views/user/_form

Hagamos que el _form para la vista de usuario tenga listas desplegables. Reemplace los archivos existentes con lo siguiente:

Gist:

[Backend User _form View](#)

Del libro:

```
<?php

use yii\helpers\Html;
use yii\widgets\ActiveForm;

/**
 * @var yii\web\View $this
 * @var common\models\User $model
 * @var yii\widgets\ActiveForm $form
 */
?>

<div class="user-form">

    <?php $form = ActiveForm::begin(); ?>

    <?= $form->field($model, 'estatdo_id')->dropDownList($model->estadoLista,
        [ 'prompt' => 'Por Favor Elija Uno' ]); ?>

    <?= $form->field($model, 'rol_id')->dropDownList($model->rolLista,
        [ 'prompt' => 'Por Favor Elija Uno' ]); ?>

    <?= $form->field($model, 'tipo_usuario_id')
        ->dropDownList($model->tipoUsuarioLista,
        [ 'prompt' => 'Por Favor Elija Uno' ]); ?>

    <?= $form->field($model, 'username')->textInput(['maxlength' => 255]) ?>

    <?= $form->field($model, 'email')->textInput(['maxlength' => 255]) ?>

    <div class="form-group">
        <?= Html::submitButton($model->isNewRecord ? 'Create' : 'Update',
            [ 'class' => $model->isNewRecord ? 'btn btn-success' : 'btn btn-primary' ]) ?>
    </div>

    <?php ActiveForm::end(); ?>

</div>
```

Este es un uso sencillo del widget ActiveForm, que hemos visto anteriormente cuando estábamos mirando nuestro archivo view. Debería estar acostumbrado a este formato en este punto. Note que he usado 2 líneas en lugar de una para evitar errores por el ajuste de línea que suceden cuando este libro trata de realizar el ajuste de línea en el código.

Note el uso de la clase auxiliar Html en el botón submit. Una sentencia ternaria determina si un registro es nuevo o necesita ser actualizado.

Otra cosa a señalar sobre esto. No necesitamos identificar la id de la acción en el formulario. Debido a la lógica del framework Yii 2, sabe que acción asociar al formulario, basado en la ubicación del archivo y el modelo pasado a la vista.

Es sólo cuando sus formularios son más complicados que necesita crear un modelo de formulario separado y es ahí cuando se necesita una id en el formulario, para que el controlador sepa que modelo usar. Vimos ejemplos de este formulario en el controlador del sitio, donde se usan numerosos modelos de formulario para cosas tales como contact, requestPasswordReset, etc.

Cambios más profundos al Backend

Probablemente habrá notado que hemos evitado conspicuamente cambiar la vista index para backend/views/perfil y backend/views/user. Esto se debe a que queremos realizar cambios al widget principal GridView. Estos cambios son un poco más profundos porque involucrarán la modificación del modelo de búsqueda y cambios a la configuración de GridView.

backend/views/user/index.php

Así que reemplazamos el archivo en backend/views/user/index.php con:

Gist:

[Backend User Index View](#)

Del libro:

```
<?php

use yii\helpers\Html;
use yii\grid\GridView;
use \yii\bootstrap\Collapse;

/* @var $this yii\web\View */
/* @var $searchModel backend\models\search\UserSearch */
/* @var $dataProvider yii\data\ActiveDataProvider */

$this->title = 'Users';
$this->params['breadcrumbs'][] = $this->title;
?>
<div class="user-index">
```

```
<h1><?= Html::encode($this->title) ?></h1>

<?php echo Collapse::widget([
    'items' => [
        // equivalente a lo de arriba
        [
            'label' => 'Search',
            'content' => $this->render('_search', ['model' => $searchModel]) ,
            // open its content by default
            //'contentOptions' => ['class' => 'in']
        ],
    ]
]);;

?>

<?= GridView::widget([
    'dataProvider' => $dataProvider,
    'filterModel' => $searchModel,
    'columns' => [
        ['class' => 'yii\grid\SerialColumn'],

        // 'id',
        ['attribute'=>'userIdLink', 'format'=>'raw'],
        ['attribute'=>'userLink', 'format'=>'raw'],
        ['attribute'=>'perfilLink', 'format'=>'raw'],

        'email:email',
        'rolNombre',
        'tipoUsuarioNombre',
        'estadoNombre',
        'created_at',

        ['class' => 'yii\grid\ActionColumn'],
    ]
]);;
```

```
// 'updated_at',  
  
],  
]); ?>  
</div>
```

Así que obviamente, en la sentencias use, incluimos:

```
use \yii\bootstrap\Collapse;
```

Esto nos permite usar el widget collapse, que usamos para contener la sentencia render, que incluye el parcial del formulario para búsqueda. El efecto es que limpia la página. En la vista, cuando pasa el cursor sobre la palabra search, se convierte en un link. Si hace clic sobre él, se despliega el formulario. Ya que hemos cubierto el widget collapse en un próximo capítulo, así que no lo veremos otra vez.

Realizaremos cambios a nuestro formulario de búsqueda, así que en este punto no hay necesidad de molestarte probándolo.

En el widget GridView, dejamos comentados algunos atributos para ser usados como referencia. Puede ver que hemos agregado userIdLink, userLink, PerfilLink, email@email, rolNombre, tipoUsuarioNombre, y estadoNombre. Estas son las etiquetas que asignamos a los métodos en el método attributes de User. En el caso de userIdLink, userLink, y perfilLink, tenemos un formato específico que tenemos que usar para devolver un link. El formato email@email crea un link mailto, práctico si quiere enviar un email al usuario. El método para userLink muestra el nombre de usuario, en caso de que se lo está preguntando.

backend/views/perfil/index.php

Hagamos algo similar para Perfil, mientras tanto. Pegue lo siguiente a backend/views/perfil/index.php:

Gist:

[Vista de Perfil del Backend](#)

Del libro:

```
<?php

use yii\helpers\Html;
use yii\grid\GridView;
use \yii\bootstrap\Collapse;

/* @var $this yii\web\View */
/* @var $searchModel backend\models\search\PerfilSearch */
/* @var $dataProvider yii\data\ActiveDataProvider */

$this->title = 'Perfiles';
$this->params['breadcrumbs'][] = $this->title;
?>
<div class="perfil-index">

<h1><?= Html::encode($this->title) ?></h1>

<?php echo Collapse::widget([
    'items' => [
        // equivalent to the above
        [
            'label' => 'Search',
            'content' => $this->render('_search', ['model' => $searchModel]) ,
            // open its content by default
            //'contentOptions' => ['class' => 'in']
        ],
    ]
]); ?>

<?= GridView::widget([
    'dataProvider' => $dataProvider,
    'filterModel' => $searchModel,
    'columns' => [
        ['class' => 'yii\grid\SerialColumn'],
        // 'id',
    ]
]); ?>
```

```

['attribute'=>'perfilIdLink', 'format'=>'raw'],
['attribute'=>'userLink', 'format'=>'raw'],
'nombre',
'apellido',
'fecha_nacimiento',
'genderName',
['class' => 'yii\grid\ActionColumn'],

// 'created_at',
// 'updated_at',
// 'user_id',

],
]); ?>

</div>

```

El widget GridView es similar al que se encuentra en la vista index de usuario, pero con menos columnas.

Cuando estaba haciendo esto en el flujo de trabajo, noté que podíamos enlazar el atributo id a la vista de actualización del perfil, lo que nos daría una manera rápida de actualizar el perfil. Así que agregué el siguiente método al modelo perfil:

```

public function getPerfilIdLink()
{
    $url = Url::to(['perfil/update', 'id'=>$this->id]);
    $options = [];
    return Html::a($this->id, $url, $options);
}

```

También agregué al modelo perfil, la siguiente etiqueta de atributo:

```
'perfilIdLink' => Yii::t('app', 'Perfil'),
```

Luego finalmente, comenté la id y agregué 'perfilIdLink' al widget en backend/views/perfil/index:

```
[ 'attribute'=>'perfilIdLink', 'format'=>'raw' ],
```

Pero la mayoría de esto fue obviamente realizado con anterioridad, cuando creamos los modelos. Al menos ahora sabe por qué lo hicimos.

Aunque esto nos da nuestro link, no tenemos capacidades de ordenamiento. Ya que el ordenamiento es algo que deseamos, haremos esos cambios, pero esperaremos para agregarlos porque tendremos que realizar otros cambios al modelo de búsqueda al mismo tiempo.

backend/views/perfil/_search.php

Ahora actualicemos backend/views/perfil/_search.php. Reemplace el contenido completo del archivo con lo siguiente:

Gist:

[Vista _search de Perfil del Backend](#)

Del libro:

```
<?php

use yii\helpers\Html;
use yii\bootstrap\ActiveForm;
use frontend\models\Perfil;
/**
 * @var yii\web\View $this
 * @var backend\models\search\PerfilSearch $model
 * @var yii\widgets\ActiveForm $form
 */
?>

<div class="perfil-search">

<?php $form = ActiveForm::begin([
    'action' => ['index'],
    'method' => 'get',
]); ?>

<?= $form->field($model, 'nombre') ?>

<?= $form->field($model, 'apellido') ?>

<?= $form->field($model, 'fecha_nacimiento') ?>

<?= $form->field($model, 'genero_id')->dropDownList(Perfil::getGeneroLista(),
    [ 'prompt' => 'Por Favor Elija Uno' ]);?>

<?php // echo $form->field($model, 'created_at') ?>

<?php // echo $form->field($model, 'updated_at') ?>
```

```
<?php // echo $form->field($model, 'user_id') ?>

<div class="form-group">
    <?= Html::submitButton('Search', ['class' => 'btn btn-primary']) ?>
    <?= Html::resetButton('Reset', ['class' => 'btn btn-default']) ?>
</div>

<?php ActiveForm::end(); ?>

</div>
```

No se moleste en comprobar la precisión de la búsqueda aún, tenemos que trabajar en el modelo de búsqueda, lo haremos pronto.

backend/views/user/_search.php

Bien, ahora tenemos que trabajar con el archivo de vista _search del usuario y en el archivo backend/models/search/UserSearch.php. El archivo UserSearch.php provee el modelo para el archivo _search.php en backend/views/user/_search.php, que es renderizado dentro de backend/views/user/index.php.

Básicamente, es el formulario de búsqueda en la parte superior del archivo index.

Comencemos reemplazando los contenidos de _search.php con lo siguiente:

Gist:

[Vista _search de User del Backend](#)

Del libro:

```
<?php

use yii\helpers\Html;
use yii\widgets\ActiveForm;
use common\models\User;

/* @var $this yii\web\View */
/* @var $model backend\models\search\UserSearch */
/* @var $form yii\widgets\ActiveForm */
?>

<div class="user-search">
```

```

<?php $form = ActiveForm::begin([
    'action' => ['index'],
    'method' => 'get',
]); ?>

<?= $form->field($model, 'id') ?>

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

<?php echo $form->field($model, 'email') ?>

<?= $form->field($model, 'rol_id')->dropDownList(User::getRolLista(),
    [ 'prompt' => 'Por Favor Elija Uno' ]); ?>

<?= $form->field($model, 'tipo_usuario_id')
    ->dropDownList(User::getTipoUsuarioLista(),
        [ 'prompt' => 'Por Favor Elija Uno' ]); ?>

<?= $form->field($model, 'estado_id')->dropDownList($model->estadoLista,
    [ 'prompt' => 'Por Favor Elija Uno' ]); ?>

<div class="form-group">
    <?= Html::submitButton('Search', ['class' => 'btn btn-primary']) ?>
    <?= Html::resetButton('Reset', ['class' => 'btn btn-default']) ?>
</div>

<?php ActiveForm::end(); ?>

</div>

```

Una cosa que puede haber notado es que en nuestro método ActiveForm::begin, listamos acción y método. La razón por la que hacemos esto es que esperamos datos dinámicos por parte del usuario. Ellos van a enviar variables al controlador, así que necesitamos especificar ‘get’ como método. Así es como Gii nos la entrega.

Y puede ver que apenas hemos cambiado el archivo de otro modo, excepto para hacer listas desplegables a partir de métodos en el modelo User.

Ahora si lo prueba en el navegador, verá que funciona muy bien, pero notará que la lista desplegable para tipoUsuario muestra las opciones, pero no filtra los resultados. También, necesitamos

asegurarnos de que realizamos un eager load de los resultados.

Eager loading, si lo recuerda, es la forma en que evitamos el problema n+1, donde se realiza una consulta por cada fila de resultados. En una base de datos donde se tienen un gran número de resultados, un problema n+1 puede inutilizar la página porque tardará muchísimo en devolver los resultados, si es que puede hacerlo.

Lo evitamos con eager loading. Haremos esto cuando modifiquemos el modelo UserSearch.

User Search

El modelo UserSearch es una extensión del modelo User, que el controlador utiliza para darle instrucciones sobre cómo realizar consultas al modelo.

El archivo está ubicado en backend/models/search/UserSearch.php. El método principal es search(\$params), así que observémoslo:

```
public function search($params)
{
    $query = user::find();

    $dataProvider = new ActiveDataProvider([
        'query' => $query,
    ]);

    if (!($this->load($params) && $this->validate())) {
        return $dataProvider;
    }

    $query->andFilterWhere([
        'id' => $this->id,
        'rol_id' => $this->rol_id,
        'estado_id' => $this->estado_id,
        'created_at' => $this->created_at,
        'updated_at' => $this->updated_at,
    ]);

    $query->andFilterWhere(['like', 'username', $this->username])

    ->andFilterWhere(['like', 'auth_key', $this->auth_key])
    ->andFilterWhere(['like', 'password_hash', $this->password_hash])
    ->andFilterWhere(['like', 'password_reset_token', $this->password_reset_token])
    ->andFilterWhere(['like', 'email', $this->email]);
```

```

        return $dataProvider;
    }
}

```

\$params es enviado por el formulario. Esto ocurre a través de UserController, que comienza la acción index de esta manera:

```

public function actionIndex()
{
    $searchModel = new UserSearch();
    $dataProvider = $searchModel->search(Yii::$app->request->queryParams);
}

```

Para mostrar los resultados, llamamos a una instancia del modelo, en este caso new UserSearch(), luego le asignamos a la variable \$dataProvider una instancia del modelo, con el método search pasandole los parámetros de consulta a través de Yii:\$app->request->queryParams. \$dataProvider será usado por el widget de la vista para mostrar los resultados.

Lo importante a tener en cuenta es que cuando la acción index sea llamada, buscará los parámetros del formulario y ejecutará el método search, aún si no existen parámetros de búsqueda. Sin los parámetros de búsqueda, simplemente devolverá todos los registros de la BD.

Así que ahora observemos el método search en detalle porque es importante saber cómo funciona. El método search en el modelo UserSearch, comienza por asignar a \$query el método User::find.

Luego de asignar a query la instancia del modelo, creamos una instancia de la clase ActiveDataProvider de Yii 2, pasando query al interior del arreglo de configuración.

```

$dataProvider = new ActiveDataProvider([
    'query' => $query,
]);

```

ActiveDataProvider crea un poderoso iterador a partir de objetos resultado, en este caso user, donde obtenemos todos los resultados, ya que a \$query le fue asignada originalmente User::find().

Una vez que instanciamos ActiveDataProvider, comprobamos si hemos agregado parámetros de búsqueda a través del formulario, y si no, devolvemos los resultados sin filtrar de \$query, que como ya hemos indicado, devolverá todos los usuarios.

```

if (!$this->load($params) && !$this->validate())) {
    return $dataProvider;
}

```

Si hay \$params entregados desde el formulario, entonces evaluamos a falso, lo que significa que no devolvemos aún \$dataProvider, y continuamos con el siguiente bloque y luego llamamos al método andFilterWhere() method del Modelo user, para filtrar los parámetros.

```
$query->andFilterWhere([
    'id' => $this->id,
    'rol_id' => $this->rol_id,
    'estado_id' => $this->estado_id,
    'created_at' => $this->created_at,
    'updated_at' => $this->updated_at,
]);
```

Luego vemos otra llamada al mismo método para cubrir los parámetros ‘like’:

```
$query->andFilterWhere(['like', 'username', $this->username])
->andFilterWhere(['like', 'auth_key', $this->auth_key])
->andFilterWhere(['like', 'password_hash', $this->password_hash])
->andFilterWhere(['like', 'password_reset_token',
    $this->password_reset_token])
->andFilterWhere(['like', 'email', $this->email]);
```

Puede ver como las llamadas al método son encadenadas sucesivamente, con el punto y coma en la última línea. Luego finalmente, devolvemos \$dataProvider:

```
return $dataProvider;
```

Esa es nuestra versión predeterminada. Pero necesitamos una versión más robusta. Tenemos que obtener datos relacionados de Roles, TipoUsuario, etc. y necesitamos realizar eager loading, así que lo que necesitamos es un poco más complejo. Reemplace el viejo modelo UserSearch con el siguiente:

Gist:

Modelo User Search del Backend

Del libro:

```
<?php

namespace backend\models\search;

use Yii;
use yii\base\Model;
use yii\data\ActiveDataProvider;
use common\models\User;

/**
 * UserSearch represents the model behind the
 * search form about `common\models\User`.

```

```
*/\n\n\nclass UserSearch extends User\n{\n\n    /**\n     * attributes\n     *\n     * @var mixed\n    */\n\n    public $rolNombre;\n    public $tipoUsuarioNombre;\n    public $tipo_usuario_nombre;\n    public $tipo_usuario_id;\n    public $estadoNombre;\n    public $perfilId;\n\n    /**\n     * @inheritDoc\n    */\n\n    public function rules()\n    {\n        return [\n\n            [['id', 'rol_id', 'estado_id', 'tipo_usuario_id'], 'integer'],\n            [['username', 'email', 'created_at', 'updated_at', 'rolNombre',\n                'estadoNombre','tipoUsuarioNombre', 'perfilId',\n                'tipo_usuario_nombre'], 'safe'],\n        ];\n    }\n\n    /**\n     * @inheritDoc\n    */\n}
```

```
public function scenarios()
{
    // bypass scenarios() implementation in the parent class
    return Model::scenarios();
}

/**
 * Creates data provider instance with search query applied
 *
 * @param array $params
 *
 * @return ActiveDataProvider
 */

```



```
public function search($params)
{
    $query = User::find();
    $dataProvider = new ActiveDataProvider([
        'query' => $query,
    ]);

    /**
     * Setup your sorting attributes
     * Note: This is setup before the $this->load($params)
     * statement below
     */

    $dataProvider->setSort([
        'attributes' => [
            'id',
            'userIdLink' => [
                'asc' => ['user.id' => SORT_ASC],
                'desc' => ['user.id' => SORT_DESC],
                'label' => 'User'
            ],
            'userLink' => [

```

```
        'asc' => ['user.username' => SORT_ASC],
        'desc' => ['user.username' => SORT_DESC],
        'label' => 'User'
    ],
    'perfilLink' => [
        'asc' => ['perfil.id' => SORT_ASC],
        'desc' => ['perfil.id' => SORT_DESC],
        'label' => 'Perfil'
    ],
    'rolNombre' => [
        'asc' => ['rol.rol_nombre' => SORT_ASC],
        'desc' => ['rol.rol_nombre' => SORT_DESC],
        'label' => 'Rol'
    ],
    'estadoNombre' => [
        'asc' => ['estado.estado_nombre' => SORT_ASC],
        'desc' => ['estado.estado_nombre' => SORT_DESC],
        'label' => 'Estado'
    ],
    'tipoUsuarioNombre' => [
        'asc' => ['tipo_usuario.tipo_usuario_nombre' => SORT_ASC],
        'desc' => ['tipo_usuario.tipo_usuario_nombre' => SORT_DESC],
        'label' => 'Tipo Usuario'
    ],
    'created_at' => [
        'asc' => ['created_at' => SORT_ASC],
        'desc' => ['created_at' => SORT_DESC],
        'label' => 'Created At'
    ],
    'email' => [
        'asc' => ['email' => SORT_ASC],
        'desc' => ['email' => SORT_DESC],
        'label' => 'Email'
    ],
]
]);
```

```
if (!($this->load($params) && $this->validate())) {  
  
    $query->joinWith(['rol'])  
        ->joinWith(['estado'])  
        ->joinWith(['perfil'])  
        ->joinWith(['tipoUsuario']);  
  
    return $dataProvider;  
}  
  
$this->addSearchParameter($query, 'id');  
$this->addSearchParameter($query, 'username', true);  
$this->addSearchParameter($query, 'email', true);  
$this->addSearchParameter($query, 'rol_id');  
$this->addSearchParameter($query, 'estado_id');  
$this->addSearchParameter($query, 'tipo_usuario_id');  
$this->addSearchParameter($query, 'created_at');  
$this->addSearchParameter($query, 'updated_at');  
  
// filtrar por rol  
  
$query->joinWith(['rol' => function ($q) {  
  
    $q->andFilterWhere(['=' , 'rol.rol_nombre', $this->rolNombre]);  
  
}])  
  
// filtrar por estado  
  
->joinWith(['estado' => function ($q) {  
  
    $q->andFilterWhere(['=' , 'estado.estado_nombre', $this->estadoNombre]);  
  
}])  
  
// filtrar por tipo usuario  
  
->joinWith(['tipoUsuario' => function ($q) {  
  
    $q->andFilterWhere(['=' , 'tipo_usuario.tipo_usuario_nombre',  
    
```

```
        $this->tipoUsuarioNombre]);
    })
}

// filtrar por perfil

->joinWith(['perfil' => function ($q) {
    $q->andFilterWhere(['=', 'perfil.id', $this->perfilId]);
}]);

return $dataProvider;
}

protected function addSearchParameter($query, $attribute, $partialMatch = false)
{
    if (($pos = strpos($attribute, '.')) !== false) {
        $modelAttribute = substr($attribute, $pos + 1);
    } else {
        $modelAttribute = $attribute;
    }

    $value = $this->$modelAttribute;

    if (trim($value) === '') {
        return;
    }

/*
 * The following line is additionally added for right aliasing
 * of columns so filtering happen correctly in the self join
 */
}

$attribute = "user.$attribute";

if ($partialMatch) {
    $query->andWhere(['like', $attribute, $value]);
} else {
    $query->andWhere([$attribute => $value]);
}
}
```

```
}
```

Bien, veamos esta bestia. Parece mucho, pero no es tan malo una vez que lo descompone. También, debo señalar que aprendí esto siguiendo un tutorial de Kartik, el mismo autor que escribió el widget social que estamos usando en la página de inicio de nuestra aplicación. Lo refactorice un poco por motivos cosméticos, y probablemente no gané mucha claridad con ello, pero al menos lo intenté.

La primera cosa a notar es que la clase UserSearch extiende a User, y nos aseguramos de incluir en nuestra lista de atributos aquellos que son referenciados por un método del modelo, rolNombre por ejemplo, porque sabemos que vamos a usar una lista desplegable para devolver nombres de rol. Si no es listado explícitamente como un atributo, el formulario se rompe y la página no se renderiza. Así que si se le presenta esa clase de problema, asegúrese de que tiene una lista completa de los atributos que necesita. Esto no es siempre obvio porque se supone que el modelo padre de User conoce todos sus atributos mediante reflección en algún lugar de las clases base.

Lo que he hallado, juntando todo, es que necesito declarar los atributos \$tipo_usuario_id y \$tipo_usuario_nombre. Simplemente no estoy seguro del por qué. Uso estos atributos en cláusulas where, así que tal vez esa es la causa del problema, quizás no puede usar al modelo padre en esa ubicación para identificar el atributo. Esto, por supuesto, es sólo una suposición.

Cuando esté trabajando con un framework grande como Yii 2, va a encontrarse ocasionalmente con cosas que no comprende totalmente. Está bien, le ocurre a todos, y yo ciertamente puedo atestiguarlo de manera personal. Lo importante es que tratamos de aprender tanto como podamos sobre la marcha porque cuando se trata de usar un framework, el conocimiento es poder.

Bien, luego se encuentran las reglas usadas en la validación. El primer arreglo nos dice cuáles atributos son solamente enteros. El segundo arreglo nos dice que atributos son seguros. Necesitamos esta regla debido al método setAttributes de la clase /yii/base/Model, que ignora atributos si un atributo no tiene al menos una regla de validación y no está marcado como seguro por la regla safe. Así que cuando los contenidos de \$_Post son enviados al método, sólo a los valores aceptados se les permite ingresar.

De cualquier modo, Gii incluye un arreglo safe en las reglas que es auto-generado, así que lo he mantenido, y me he asegurado de que contiene los atributos actuales, que son agregados a aquellos del modelo padre. Nuevamente tuve que usar prueba y error para asegurarme de que tenía lo necesario.

Luego tenemos el método:

```
public function scenarios()
{
    // bypass scenarios() implementation in the parent class
    return Model::scenarios();
}
```

Este método permite pasar por alto al método scenarios del padre, lo que le permitiría crear sus propios escenarios. No lo usaremos, así que simplemente lo dejaremos en su lugar, ya que es lo que Gii nos entregó.

Luego tendríamos el método attributeLabels, pero no lo necesitamos porque estamos heredando todas las etiquetas de atributo que necesitamos del modelo padre y no hemos añadido nada nuevo que requiera una.

Bien, continuemos con el método search:

```
public function search($params)
{
    $query = User::find();
    $dataProvider = new ActiveDataProvider([
        'query' => $query,
    ]);
}
```

Este es exactamente como el código que Gii genera para nosotros, así que no hay cambios allí. Creamos una instancia de User con el método find, que devolverá todos los resultados. Luego creamos una instancia de ActiveDataProvider e injectamos el modelo user a través de \$query. Así que ahora \$dataProvider está cargado con el modelo User y todos sus registros. Luego, usaremos el controlador para pasar este \$dataProvider a la vista, donde el widget GridView puede usarlo.

Bien, de regreso al modelo UserSearch y su método search.

Luego tomamos el método setSort de \$dataProvider y lo configuramos de modo que las columnas que queremos que sean ordenables en la Grilla tengan el comportamiento deseado:

```
$dataProvider->setSort([
    'attributes' => [
        'id',
        'userIdLink' => [
            'asc' => ['user.id' => SORT_ASC],
            'desc' => ['user.id' => SORT_DESC],
            'label' => 'ID'
        ],
        'userLink' => [
            'asc' => ['user.username' => SORT_ASC],
            'desc' => ['user.username' => SORT_DESC],
            'label' => 'User'
        ],
        'perfilLink' => [

```

```
'asc' => ['perfil.id' => SORT_ASC],
'desc' => ['perfil.id' => SORT_DESC],
'label' => 'Perfil'
],

'rolNombre' => [
  'asc' => ['rol.rol_nombre' => SORT_ASC],
  'desc' => ['rol.rol_nombre' => SORT_DESC],
  'label' => 'Rol'
],
'estadoNombre' => [
  'asc' => ['estado.estado_nombre' => SORT_ASC],
  'desc' => ['estado.estado_nombre' => SORT_DESC],
  'label' => 'Estado'
],
'tipoUsuarioNombre' => [
  'asc' => ['tipo_usuario.tipo_usuario_nombre' => SORT_ASC],
  'desc' => ['tipo_usuario.tipo_usuario_nombre' => SORT_DESC],
  'label' => 'Tipo Usuario'
],
'created_at' => [
  'asc' => ['created_at' => SORT_ASC],
  'desc' => ['created_at' => SORT_DESC],
  'label' => 'Created At'
],
'email' => [
  'asc' => ['email' => SORT_ASC],
  'desc' => ['email' => SORT_DESC],
  'label' => 'Email'
],
]
]);
});
```

Esto es limpio y fácil de comprender.

Luego tenemos una sentencia if:

```

if (!($this->load($params) && $this->validate())) {

    $query->joinWith(['rol'])
        ->joinWith(['estado'])
        ->joinWith(['perfil'])
        ->joinWith(['tipoUsuario']);

    return $dataProvider;

}

```

Observemos el if:

```
if (!($this->load($params) && $this->validate()))
```

Vimos esto antes cuando observamos la versión que Gii nos dio. Esta opera de la misma forma. Lo que dice es cargar los parámetros y ejecutar el método de validación, luego evaluar a verdadero o falso. El ! puede ser confuso, así que lo explicaré completamente.

Si la sentencia evalúa a verdadero, sólo hay una pequeña cantidad de código que sigue a una sentencia return. Esto es muy fácil de seguir. Sólo para ser claros, si no se tienen parámetros evaluaría a verdadero, luego ejecutaría el pequeño bloque de código con la sentencia return.

Si la sentencia if evalúa a falso, y hay parámetros siendo pasados desde el formulario de búsqueda, nos vemos hacia el siguiente bloque de código para continuar.

Bien, veamos la condición verdadera en primer lugar:

Si no hay parámetros evalúa verdadero en la sentencia if, agregar las relaciones via el método joinWith (para eager loading) y devolver el modelo con sus relaciones, almacenada en el \$dataProvider, ya que \$query ya ha sido inyectado en el \$dataProvider. El controlador pasará \$dataProvider a la vista.

```

{
    $query->joinWith(['rol'])
        ->joinWith(['estado'])
        ->joinWith(['perfil'])
        ->joinWith(['tipoUsuario']);

    return $dataProvider;

}

```

Si observa cuidadosamente, verá que tipoUsuario tiene una letra mayúscula 'U', lo que tiene que ver con la forma en que la relación del modelo es nombrada debido a que existen dos palabras en su

composición. Si la nombre de manera equivocada o si no hay un método get para la relación en el modelo user, obtendrá un error. Las convención de nombre le da una letra mayúscula a la segunda palabra en un nombre de modelo donde hay más de una palabra.

\$query es una instancia del modelo User y está configurada dentro de \$dataProvider, así que aún si no entregamos parámetros, podemos devolver resultados sin filtrar. Así que nuevamente, para ser perfectamente claros, si no hay parámetros para la búsqueda, devolvemos todos los registros.

Note que hemos encadenado el modelo user con otros 4 modelos, rol, estado, perfil, y tipoUsuario, para poder hacer eager loading de los resultados, lo que significa que no tendremos que realizar una consulta separada por cada fila de resultados. Estos joins nos permiten sincronizar al Usuario con el correspondiente perfil, rol, etc.

Eager loading es el opuesto a lazy load, y para conjuntos grandes de datos, como los registros en el modelo User por ejemplo, es preferible ya que exige menos a la BD.

Ahora observemos la posibilidad más larga y compleja de la sentencia if. Si la sentencia if (!(\$this->load(\$params) && \$this->validate())) evalúa a falso, esto significa que tenemos parámetros para la búsqueda, y nos movemos al siguiente bloque de código, donde usamos un método llamado **addSearchParameter** para agregar condiciones a la consulta:

```
$this->addSearchParameter($query, 'id');
$this->addSearchParameter($query, 'username', true);
$this->addSearchParameter($query, 'email', true);
$this->addSearchParameter($query, 'rol_id');
$this->addSearchParameter($query, 'estado_id');
$this->addSearchParameter($query, 'tipo_usuario_id');
$this->addSearchParameter($query, 'created_at');
$this->addSearchParameter($query, 'updated_at');
```

Puede ver que ejecutamos una instancia del método por cada atributo. Así que observemos el método **addSearchParameter** para tener una mejor idea de lo que ocurre:

```
protected function addSearchParameter($query, $attribute, $partialMatch = false)
{
    if (($pos = strpos($attribute, '.')) !== false) {
        $modelAttribute = substr($attribute, $pos + 1);
    } else {
        $modelAttribute = $attribute;
    }

    $value = $this->$modelAttribute;

    if (trim($value) === '') {
```

```

        return;
    }

/*
 * The following line is additionally added for right aliasing
 * of columns so filtering happen correctly in the self join
 */

$attribute = "user.$attribute";

if ($partialMatch) {
    $query->andWhere(['like', $attribute, $value]);
} else {
    $query->andWhere([$attribute => $value]);
}
}

```

La primera parte determina si hay un ‘.’ en el atributo. He añadido espacios en blanco para que sea más fácil de leer:

```

if (($pos = strrpos($attribute, '.')) !== false) {

    $modelAttribute = substr($attribute, $pos + 1);

} else {

    $modelAttribute = $attribute;
}

```

Si el parámetro tiene un ‘.’, entonces el método posiciona el atributo para parámetro de búsqueda correctamente, **así que sabe cuál es el modelo y cuál es el atributo.**

El método elimina lo que se encuentra antes del punto, lo logra eliminar un problema de ambigüedad, ya que la tabla rol por ejemplo, también tiene una columna id. Esto es algo complicado y no funcionará correctamente si no lo hacemos de esta forma exactamente.

Ya sea que halla o no un ‘.’, or not, asigna \$attribute a \$modelAttribute.

La siguiente línea:

```
$value = $this->$modelAttribute;
```

Establece el valor del atributo. Simplemente un rápido recordatorio de cómo funciona esto.

El atributo es pasado a este método como una cadena en la variable \$attribute, donde es formateado para tener en cuenta si tiene un punto o no.

En cualquier caso, ya sea que haya tenido un punto o no, la variable es renombrada como `$modelAttribute`. Pero esto aún representa la cadena que fue pasada a través de `$attribute`. Así que cuando llamamos `$this->$modelAttribute`, estamos insertando la variable donde normalmente iría una cadena. La variable `$value` toma el resultado de esta expresión, cualquiera sea su tipo, cadena, entero, booleando.

Por ejemplo, si leemos `$value = $this->username`, sería más intuitivo para nosotros esperar que `$value` tenga el nombre de usuario real, que es una cadena. En su lugar obtenemos `$value = $this->$modelAttribute`, lo que es grandioso porque podemos usarlo para todos los atributos y pasará el formato correcto a la variable `$value`.

`$this` se refiere a una instancia de `UserSearch`, que como sabemos, extiende a `User`, así que `$this` puede tener un atributo llamado `username` o cualquiera de los otros que proveímos cuando llamamos al método.

Ahora usted puede estar preguntándose, si está usando una cadena, por ejemplo, ¿cómo sabe qué valor específico devolver? Este es un rompecabezas para mí, no tan obvio al mirar el código. La respuesta es que ya ha adquirido el valor desde el formulario:

```
if (!($this->load($params) && $this->validate())))
```

Recuerde que la sentencia `!` sólo evalúa a verdadero, forzando la sentencia `return`, si no hay parámetros. Si hay parámetros, ejecuta exitosamente los métodos `load` y `validate`, así que para el momento en que estamos usando estos nombres de campo para configurar nuestra consulta en el método `addSearchParameter`, el modelo, nuestro amistoso `$this`, ya tiene los valores que necesitamos.

Ah, tan simple una vez que vemos como trabaja. No se si usted luchó tanto con ello como yo, pero por la salud de ambos, estoy feliz de que finalmente lo entendí. También note, que si los parámetros no pueden cargarse o si la validación falla, mostrará el formulario con errores. Sin embargo esta acción ocurre en el controlador, no en el modelo.

De cualquier manera, regresemos al método `addSearchParameter`. No hemos terminado aún:

Retornar si está vacío:

```
if (trim($value) === '') {  
    return;  
}
```

En este caso, esa es otra forma de decir no hacer nada, así no termina con un conjunto de sentencias vacías en la consulta. Nuevamente para ser claros, si el campo está vacío, no es agregado como un parámetro de búsqueda.

En caso contrario:

```

/*
 * The following line is additionally added for right aliasing
 * of columns so filtering happen correctly in the self join
*/
$attribute = "user .{$attribute}";

```

El comentario de arriba explica parte. Ponemos el nombre de la tabla delante del atributo para evitar problemas de ambigüedad. Ya que este método ejecuta un atributo a la vez, podemos asumir con seguridad que \$attribute contiene la cadena que deseamos. Ya que hemos eliminado todo lo que se encontraba delante del punto con anterioridad en el método, no puede haber confusión sobre a qué tabla estamos haciendo referencia, ya que estamos diciéndole explícitamente que use user.

Bien, continuemos con \$partialMatch. El valor por defecto se establece a falso. So the statement if (\$partialMatch) will check to see if it is true. The only way it can be true is if it is handed in that way. If you check the list of calls to the addSearchParmeter method, you can see that username and email are set to true.

Partial matches are handy, especially on strings, where the user doesn't want or sometimes even know how to type in the full match.

Anyway, if \$partialMatch is true, then use the like operator in the andWhere method (which has

been inherited from somewhere else in the framework) to add a partial match to search on:

Bien, continuamos con \$partialMatch. El valor por defecto se establece como falso.

Entonces la afirmación si

```

if ($partialMatch) {

    $query->andWhere(['like', $attribute, $value]);
}

```

De lo contrario, usar el método andWhere para pasar el atributo y su valor a la consulta:

```
$query->andWhere([$attribute => $value]);
```

Bien, así que los atributos son agregados. Ahora regresamos a donde habíamos quedado en el método search y vamos a los joins que nos permitirán filtrar. Puede ver que por cada uno, añadimos un closure, una función anónima que enlaza el método andFilterWhere al modelo con el que se está realizando el join:

```
// filtrar por rol  
  
$query->joinWith(['rol' => function ($q) {  
  
    $q->andFilterWhere(['=' , 'rol.rol_nombre' , $this->rolNombre]);  
  
}])  
  
// filtrar por estado  
  
->joinWith(['estado' => function ($q) {  
  
    $q->andFilterWhere(['=' , 'estado.estado_nombre' , $this->estadoNombre]);  
  
}])  
  
// filtrar por tipo usuario  
  
->joinWith(['tipoUsuario' => function ($q) {  
  
    $q->andFilterWhere(['=' , 'tipo_usuario.tipo_usuario_nombre' ,  
                        $this->tipoUsuarioNombre]);  
  
}])  
  
// filtrar por perfil  
  
->joinWith(['perfil' => function ($q) {  
  
    $q->andFilterWhere(['=' , 'perfil.id' , $this->perfilId]);  
  
}]);
```

Note que hemos encadenado los métodos `->joinWith`, pero tenemos comentarios entre ellos, sea cuidadoso, el punto y coma de cierre sólo va al final. Esta es una sintaxis muy intuitiva en para el método `andFilterWhere`. El primer parámetro nos da nuestro operador, en este caso `=` significa igual porque estamos construyendo una consulta sql. El segundo parámetro nos da el nombre tabla y el campo. El tercer parámetro es el valor que queremos enlazar a la consulta. Nuevamente, sabemos que el modelo ya obtuvo los valores de entrada desde el formulario, así que cuando ve `$this->estadoNombre`, por ejemplo, está usando el valor que viene desde el formulario.

Y gracias a Dios hemos terminado. Estoy agotado. Aprender a programar es divertido, pero también es trabajo duro.

Tenemos que realizar cambios similares a PerfilSearch.php y necesitamos asegurarnos de que agregamos nuestro ordenamiento para perfilIdLink:

Tomemos el archivo PerfilSearch.php y reemplazemoslos con:

Gist:

Modelo Perfil Search del Backend

Del libro:

```
<?php

namespace backend\models\search;

use Yii;
use yii\base\Model;
use yii\data\ActiveDataProvider;
use frontend\models\Perfil;

class PerfilSearch extends Perfil
{
    public $generoNombre;
    public $genero_id;
    public $userId;

    public function rules()
    {
        return [
            [['id', 'genero_id'], 'integer'],
            [['nombre', 'apellido', 'fecha_nacimiento', 'generoNombre', 'userId'], 'safe'],
        ];
    }

    /**
     * @inheritdoc
     */
}

public function attributeLabels()
```

```
{  
    return [  
        'genero_id' => 'Genero',  
    ];  
}  
  
  
public function search($params)  
{  
    $query = Perfil::find();  
    $dataProvider = new ActiveDataProvider([  
        'query' => $query,  
    ]);  
  
  
    $dataProvider->setSort([  
        'attributes' => [  
            'id',  
            'nombre',  
            'apellido',  
            'fecha_nacimiento',  
            'generoNombre' => [  
                'asc' => ['genero.genero_nombre' => SORT_ASC],  
                'desc' => ['genero.genero_nombre' => SORT_DESC],  
                'label' => 'Genero'  
            ],  
            'perfilIdLink' => [  
                'asc' => ['perfil.id' => SORT_ASC],  
                'desc' => ['perfil.id' => SORT_DESC],  
                'label' => 'ID'  
            ],  
            'userLink' => [  
                'asc' => ['user.username' => SORT_ASC],  
                'desc' => ['user.username' => SORT_DESC],  
                'label' => 'User'  
            ],  
        ]  
    ]);  
  
if (!$this->load($params) && $this->validate()) {
```

```
$query->joinWith(['genero'])
    ->joinWith(['user']);

return $dataProvider;
}

$this->addSearchParameter($query, 'id');
$this->addSearchParameter($query, 'nombre', true);
$this->addSearchParameter($query, 'apellido', true);
$this->addSearchParameter($query, 'fecha_nacimiento');
$this->addSearchParameter($query, 'genero_id');
$this->addSearchParameter($query, 'created_at');
$this->addSearchParameter($query, 'updated_at');
$this->addSearchParameter($query, 'user_id');

// filtrar por genero nombre

$query->joinWith(['genero' => function ($q) {
    $q->andFilterWhere(['=' , 'genero.genero_nombre', $this->generoNombre]);
}])

// filter by user

->joinWith(['user' => function ($q) {
    $q->andFilterWhere(['=' , 'user.id', $this->user]);
}]);

return $dataProvider;
}

protected function addSearchParameter($query, $attribute, $partialMatch = false)
{
    if (($pos = strpos($attribute, '.')) !== false) {
        $modelAttribute = substr($attribute, $pos + 1);
    } else {
```

```
$modelAttribute = $attribute;  
}  
  
$value = $this->$modelAttribute;  
if (trim($value) === '') {  
    return;  
}  
  
/*  
 * La siguiente línea se agrega para un correcto uso de alias  
 * de columnas para que el filtrado funcione en el self join  
 */  
  
$attribute = "perfil.$attribute";  
  
if ($partialMatch) {  
    $query->andWhere(['like', $attribute, $value]);  
} else {  
    $query->andWhere([$attribute => $value]);  
}  
}  
}
```

Así que ahora, podemos ordenar por la columna id cuando estamos en backend/views/profile/index.php. Y con eso, deberíamos tener todo lo necesario para PerfilSearch.

IU de Admin

Realicemos un cambio a backend/views/site/index. Queremos que la navegación facilite nuestras tareas de administración, así que reemplazemos el archivo viejo con:

Gist:

[Vista Index del Sitio del Backend](#)

Del libro:

```
<?php

use yii\helpers\Html;
use common\models\PermisosHelpers;

/**
 * @var yii\web\View $this
 */

$this->title = 'Admin Yii 2 Build';

$is_admin = PermisosHelpers::requerirMinimoRol('Admin');

?>

<div class="site-index">

    <div class="jumbotron">

        <h1>iBienvenido a Admin!</h1>

        <p class="lead">

            Ahora puede administrar usuarios, roles, y más con
            nuestras sencillas herramientas.

        </p>

        <p>

<?php

if (!Yii::$app->user->isGuest && $is_admin) {

    echo Html::a('Administrador Usuarios', ['user/index'],
    ['class' => 'btn btn-lg btn-success']);

}

}
```

```
?>
```

```
</p>

</div>

<div class="body-content">

    <div class="row">
        <div class="col-lg-4">

            <h2>Users</h2>

            <p>
```

Este es el lugar para administrar usuarios. Puede editar estados y roles desde aquí.

La IU es fácil de usar e intuitiva, simplemente haga clic en el link de abajo para comenzar.

```
</p>

<p>

<?php
```

```
if (!Yii::$app->user->isGuest && $is_admin) {

    echo Html::a('Administrar Usuarios', ['user/index'],
        ['class' => 'btn btn-default']);
}
```

```
?>
```

```
</p>

</div>
<div class="col-lg-4">

    <h2>Roles</h2>
```

<p>

Aquí es donde administra Roles. Puede decidir quién es admin y quién no. Puede añadir un nuevo rol, simplemente haciendo clic en el link de abajo para comenzar.

</p>

<p>

<?php

```
if (!Yii::$app->user->isGuest && $is_admin) {  
  
    echo Html::a('Administrar Roles', ['rol/index'],  
        ['class' => 'btn btn-default']);  
  
}
```

?>

</p>

</div>

<div class="col-lg-4">

<h2>Perfiles</h2>

<p>

¿Necesita revisar Perfiles? Este es lugar para hacerlo.
Estos son fáciles de administrar via IU. Simplemente haga clic en el link de abajo para administrar perfiles.

</p>

<p>

<?php

```
if (!Yii::$app->user->isGuest && $is_admin) {
```

```
echo Html::a('Administrar Perfiles', ['perfil/index'],
            ['class' => 'btn btn-default']);

}

?>

</p>

</div>
</div>

<div class="row">
<div class="col-lg-4">

    <h2>Tipos de Usuario</h2>

    <p>

        Este es el lugar para administrar tipos de usuario. Puede
        editar tipos de usuario desde aquí. La IU es fácil de usar e intuitiva,
        simplemente haga clic en el link de abajo para comenzar.

    </p>

    <p>

<?php

if (!Yii::$app->user->isGuest && $is_admin) {

    echo Html::a('Administrar Tipos de Usuario', ['tipo-usuario/index'],
                  ['class' => 'btn btn-default']);

}

?>

</p>

</div>
```

```
<div class="col-lg-4">

    <h2>Estados</h2>

    <p>

        Aquí es donde administra Estados. Puede agregar o eliminar.
        Puede añadir nuevos estados si lo desea, simplemente haga clic
        en el link de abajo para comenzar.

    </p>

    <p>

<?php

    if (!Yii::$app->user->isGuest && $is_admin) {

        echo Html::a('Administrar Estados', ['estado/index'],
            ['class' => 'btn btn-default']);

    }

?>

    </p>

</div>
<div class="col-lg-4">

    <h2>Placeholder</h2>

    <p>

        ¿Necesita revisar Perfiles? Este es el lugar para hacerlo.
        Estos son fáciles de administrar via IU. Simplemente haga clic
        en el link de abajo para administrar perfiles.

    </p>

    <p>
```

```
<?php

if (!Yii::$app->user->isGuest && $is_admin) {

    echo Html::a('Administrar Perfiles', ['perfil/index'],
        ['class' => 'btn btn-default']);

}

?>

</p>

        </div>
    </div>
</div>
</div>
```

Traemos un par de clases para ayudarnos:

```
use yii\helpers\Html;
use common\models\PermisosHelpers;
```

Las clases auxiliares nos permiten formatear las urls con lo siguiente:

```
echo Html::a('Administrar Roles', ['rol/index'], ['class' => 'btn btn-default']);
```

Como hicimos en el último capítulo, estamos usando el método ‘a’ de la clase Html.

Ya que queremos ser consistentes, hemos encerrado cada link en nuestra sentencia if, para verificar si el usuario es de hecho admin o mayor:

```
if (!Yii::$app->user->isGuest && $is_admin) {

    echo Html::a('Administrar Perfiles', ['perfil/index'],
        ['class' => 'btn btn-default']);

}
```

Como hicimos en la página anterior, establecemos la variable \$is_admin cerca de la parte superior del archivo debajo de la variable título:

```
$is_admin = PermisosHelpers::requerirMinimoRol('Admin');
```

Esta no es una solución de seguridad completa ya que si alguien de alguna manera se ha logueado al backend sin tener un rol de Admin o superior, aún podrían escribir la url, así que tendremos que agregar lógica a los controladores también.

Behaviors del Controlador

Haremos esto ahora, a través del método behaviors en backend/controllers/SiteController.php. Como hemos mostrado en un capítulo anterior, Yii 2 provee de un parámetro matchCallback en rules en AccessControl y sirve perfectamente a nuestros propósitos. Reemplazemos el método behaviors existente (no olvide la sentencia use para PermisosHelpers) con lo siguiente:

Gist:

[Behaviors del Controlador del Sitio del Backend](#)

Del libro:

```
public function behaviors()
{
    return [
        'access' => [
            'class' => AccessControl::className(),
            'rules' => [
                [
                    'actions' => ['login', 'error'],
                    'allow' => true,
                ],
                [
                    'actions' => ['index'],
                    'allow' => true,
                    'roles' => ['@'],
                    'matchCallback' => function ($rule, $action) {
                        return PermisosHelpers::requerirMinimoRol('Admin')
                            && PermisosHelpers::requerirEstado('Activo');
                    }
                ],
                [
                    'actions' => ['logout'],
                    'allow' => true,
                    'roles' => ['@'],
                ],
            ],
        ],
        'verbs' => [
            'class' => VerbFilter::className(),
            'actions' => [
                'logout' => ['post'],
            ],
        ],
    ],
}
```

```

        ],
        ],
    ];
}
```

Como mencioné arriba, no olvide agregar la sentencia use al principio del archivo:

```
use common\models\PermisosHelpers;
```

Match Callback

Agregamos la regla matchCallback para cubrir condiciones donde el estado o nivel de acceso de un usuario cambia luego de loguearse. Si tiene que disminuir el nivel de acceso de alguien, no querrá que tenga ningún acceso residual que no deba.

Necesitamos realizar cambios a nuestros controladores también, y explicaremos las reglas en detalle una vez que terminemos. Los controladores del backend para User, TipoUsuario, Estado, Perfil y Rol son un poco diferentes del ejemplo anterior y necesitan añadir el siguiente código bajo el método behaviors:

Gist:

[Behaviors del Backend Para Todos los Otros Controladores](#)

Del libro:

```
public function behaviors()
{
    return [
        'access' => [
            'class' => \yii\filters\AccessControl::className(),
            'only' => ['index', 'view', 'create', 'update', 'delete'],
            'rules' => [
                [
                    'actions' => ['index', 'create', 'view'],
                    'allow' => true,
                    'roles' => ['@'],
                    'matchCallback' => function ($rule, $action) {
                        return PermisosHelpers::requerirMinimoRol('Admin')
                            && PermisosHelpers::requerirEstado('Activo');
                    }
                ],
                [

```

```

        'actions' => [ 'update', 'delete'],
        'allow' => true,
        'roles' => [ '@'],
        'matchCallback' => function ($rule, $action) {
            return PermisosHelpers::requerirMinimoRol('SuperUsuario')
                && PermisosHelpers::requerirEstado('Activo');
        }
    ],
],
],
];

'verbs' => [
    'class' => VerbFilter::className(),
    'actions' => [
        'delete' => [ 'post'],
    ],
],
];
}

```

Asegúrese de que el método behaviors esté presente por cada uno de los controladores mencionados arriba, y una vez más no olvide la sentencia use para PermisosHelpers:

```
use common\models\PermisosHelpers;
```

Para matchCallback, proveemos de una condición:

```
'matchCallback' => function ($rule, $action) {

    return PermisosHelpers::requerirMinimoRol('Admin')
        && PermisosHelpers::requerirEstado('Activo');
```

Si evalúa a falso, no hay coincidencia (match) y obtenemos una respuesta “You do not have permission to view this page” (No tiene permiso para ver esta página). Esta es otra reutilización del método PermisosHelpers, lo que significa que estamos obteniendo una buena reutilización de código de este. El hecho de que la hayamos hecho una función pública estática significa que podemos llamarla en cualquier lugar que la necesitemos, siempre que incluyamos:

```
use common\models\PermisosHelpers;
```

Con nuestros métodos de controlador en su lugar, tenemos una cantidad decente de seguridad para impedir que alguien que no sea admin acceda los procesos de administración. A medida que lo analizamos, nos dimos cuenta de que teníamos que chequear por el estado también. ¿Qué sucedería si el estado de alguien fuera degradado durante una sesión abierta? Aún tendrían acceso a los métodos del controlador debido a que el estado sólo fue comprobado al iniciar sesión. Esto es importante en el frontend también. Así, por ejemplo, si construye un sitio donde los usuarios pueden cancelar su cuenta, no desearía que ellos se movieran por áreas del sitio que requieren un estado activo.

Note que estamos verificando si `estado_id` del usuario es igual al estado Activo.

Se supone que modificamos los behaviors en el frontend en un capítulo anterior, así que si no lo ha hecho, por favor regrese y hágalo ahora.

Bien, finalmente tenemos nuestro backend funcional. Ahora podemos ver nuestra nueva página index para admin cuando estamos logueados:

The screenshot shows the Yii 2 Admin dashboard. At the top, there's a navigation bar with links for Usuarios, Perfiles, Roles, Tipos de Usuario, Estados, and Logout (victorgarcia). Below the navigation, a large green button says "¡Bienvenido a Admin!". Underneath it, a message reads: "Ahora puede administrar usuarios, roles, y más con nuestras sencillas herramientas." There are six main sections arranged in a grid:

- Usuarios**: A brief description and a "Administrar Usuarios" button.
- Roles**: A brief description and a "Administrar Roles" button.
- Perfiles**: A brief description and a "Administrar Perfiles" button.
- Tipos de Usuario**: A brief description and a "Administrar Tipos de Usuario" button.
- Estados**: A brief description and a "Administrar Estados" button.
- Placeholder**: A brief description and a "Administrar Placeholder" button.

Resumen



Bien, eso es todo. Ahora tiene una plantilla extendida construida a partir de la plantilla avanzada de Yii 2. Hemos cubierto mucho terreno con este libro, suficiente para ponerlo en marcha, lo que era nuestra meta. Debería ser capaz de usar la plantilla de este libro para sus propios proyectos.

Este libro fue un libro para iniciarse, así que este en realidad es el comienzo de su viaje con Yii 2. Para continuar este viaje, debería consultar la guía y los foros para obtener más información sobre cómo usar el framework.

Por favor tenga en cuenta que ya he añadido capítulos extra y estaré actualizando este libro para mantenerlo al día con los cambios y para agregar nuevo material, así que búsqelo en cuanto pueda. Todas las actualizaciones son gratuitas para cualquiera que haya comprado el libro y durante la vida del mismo. Espero que sea larga.

Si le agrada este libro, por favor recomiéndelo a sus amigos. Puede visitar mi [blog](#) y dejar sus apreciaciones. Todos los comentarios positivos, links, y reseñas son grandemente apreciadas.

Gracias por acompañarme en este viaje. Espero verlo pronto.

Sobre el Autor

HASTA AQUI SE TIENE LA PLANTILLA AVANZADA

Bill Keck ha estado desarrollando aplicaciones desde 1999. En 2005, fue invitado a dar una charla en el campus de Google en Mountain View debido a un reporte privado del que fue autor. Actualmente es CEO de SERRF Corp, una compañía que utiliza Yii de manera extensa en sus productos. Es un gran creyente en el framework Yii 2 y consulta con desarrolladores sobre cómo tomar ventaja de todas las eficiencias en desarrollo que Yii 2 ofrece.

Capítulo Doce: Material Extra

Felicitaciones nuevamente por haber completado las bases del Framework Yii 2. Ahora tiene una plantilla extendida sobre la que puede construir sus futuros proyectos.

Así que construyamos sobre lo que ya tenemos. Uno de los temas sobre los que hablo en mi blog es complacer al cliente, lo que puede hacer al cumplir y superar sus expectativas. Y una de las cosas geniales sobre tener una plantilla lista es que mucho del trabajo ya estará hecho antes de que lo contraten.

Ahora si está usando Yii 2 bajo otras circunstancias, trabajando para una empresa, o una compañía que realiza desarrollo, donde nunca conoce al cliente, no se preocupe. Todo lo que cubriremos será útil sin importar el ambiente de desarrollo en el que se encuentre.

AutoResponder

A menudo un cliente tendrá necesidad de un autoresponder en algún lugar de su aplicación. Podría ser en el registro, el envío de una solicitud de soporte, o cualquier otra cosa. E inevitablemente, cuando quieran un autoresponder, dos cosas sucederán:

1. Querrán revisar el texto.
2. Desearán más autoresponders en el futuro.

Así que lo que necesitamos es una solución que podamos implementar que escala con las necesidades del cliente. Y esto puede ahorrarle muchos dolores de cabeza también.

Imagine que el cliente quiere cambiar una sola palabra en el texto de su autoresponder y quiere que usted publique una nueva versión del sitio para ello, y por si fuera poco, necesitan que lo haga durante el fin de semana, o la civilización como la conocemos llegará a su fin.

El cliente se sobreexcita, y aún así al mismo tiempo, paga las cuentas, así que tenemos que escuchar. Es una pesadilla. Todos hemos estado allí.

Así que pensé en cómo podría evitar completamente ese escenario. ¿No sería grandioso si simplemente pudieran ingresar lo que necesitan en una página de administración en el backend sin preocuparse de llamar?

Bien, por un lado, cuando llaman, nos pagan, así que no queremos que dejen de hacerlo. Por otro, mientras más poder le demos sobre el proyecto, más nos amarán por el trabajo que estamos haciendo y vendrán por más, especialmente un cliente entusiasta que aprecia la atención a los detalles.

Así que obviamente, si vamos a permitir que el cliente actualice el texto para el email via IU, más que probablemente almacenaremos ese texto en nuestra BD. Y sabemos que escribir esa IU con el auto-generador de código de Gii es sencillísimo. Así que no es difícil imaginar esa parte resolviéndose. Simplemente necesitamos una estructura de datos sencilla que trabaje muy bien.

¿Qué más necesitamos? Bien, necesitamos un modelo que busque el registro y dispare el email. Y aquí viene el aguafiestas. ¿Cómo sabe qué email enviar y dónde ponemos el código que llama el método apropiado?

Mi primer idea fue poner esto en un behavior en el controlador, pero eso no estaba del todo bien. La implementación de behaviors que he visto en controladores permite seleccionar acciones específicas, lo que está bien, pero va en contra del propósito de aplicar a todas las acciones, que es lo que deseaba.

Si sólo existiera una forma en Yii 2 de aplicar una conjunto de instrucciones a todas las acciones. En realidad la hay. Se llama afterAction. También existe un método beforeAction, pero eso no es lo que estamos buscando. Así que el método afterAction se ve así:

```
public function afterAction($action, $result)
{
    //su código aquí
    return parent::afterAction($action, $result);
}
```

Automáticamente toma dos argumentos y siempre y cuando llame al padre, se ejecutará luego de cada acción del controlador.

Así que pensé, hmm, tal vez esto funcione. Pensando en el código antes de escribirlo, pensé que simplemente podía chequear por el nombre de la acción y del controlador, para ver si existe un registro para ese par acción/controlador específico, y si es así, devolverlo y ejecutar afterAction, que tendría un método para enviar el email. O si no, no ejecutar el padre y devolver falso. O algo así.

Así que, ansioso por intentarlo, configuré mi estructura de datos para los mensajes de email. Y debido a que estoy pensando con un poco de anticipación, nombré a la tabla estado_mensaje. Me pareció que sería capaz de reutilizar el cuerpo de los emails para otros mensajes en algún punto, y quería darme flexibilidad para tener extensibilidad. Por ello es que no la llamé email_mensaje.

También, note que usé el singular. Al elegir un nombre para una tabla, trato de ceñirme a la convención establecida por Yii 2, tomando la tabla User como ejemplo. Así que siempre uso el singular.

Siéntase libre, sin embargo, de nombrarla como desee. Simplemente asegúrese de que en el caso de elegir un nombre diferente, lo referezne correctamente en el resto del tutorial.

Entonces la estructura de la tabla se ve así:

```

id (int, Primary Key, Not Null, Auto Increment)

controlador_nombre (varchar(105), Not Null)

accion_nombre (varchar(105) Not Null)

estado_mensaje_nombre (varchar(105) Not Null)

asunto (varchar(255) Not Null)

cuerpo (varchar(2025) Not Null)

estado_mensaje_descripcion (varchar(255) Not Null)

created_at (DateTime)

updated_at (DateTime)

```

estado_mensaje - Table									
Column Name	Datatype	PK	NN	UQ	BIN	UN	ZF	AI	Default
id	INT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>					
controlador_nombre	VARCHAR(105)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>					
accion_nombre	VARCHAR(105)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>					
estado_mensaje_nombre	VARCHAR(105)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>					
asunto	VARCHAR(255)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>					
cuerpo	VARCHAR(2025)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>					
estado_mensaje_descripcion	VARCHAR(255)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>					
created_at	DATETIME	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
updated_at	DATETIME	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Instrucciones Estado Mensaje

Sincronice el modelo con la BD o si está usando PhpMyAdmin, cree la tabla con esos campos y restricciones.

También, note, como mencioné al principio del libro, que no uso migraciones, esa es una decisión personal, pero es la razón por la que no proveo de una migración. Uso MySql Workbench y PhpMyAdmin para este tipo de tareas. Si está siguiendo y no usa ninguna de estas dos, siéntase en libertad de utilizar el método/herramienta de su elección.

Debería verse así cuando haya terminado:

#	Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Extra	Acción
1	id	int(11)			No	Ninguna	AUTO_INCREMENT	Cambiar Eliminar Primaria Único Más
2	controlador_nombre	varchar(105)	utf8_unicode_ci		No	Ninguna		Cambiar Eliminar Primaria Único Más
3	accion_nombre	varchar(105)	utf8_unicode_ci		No	Ninguna		Cambiar Eliminar Primaria Único Más
4	estado_mensaje_nombre	varchar(105)	utf8_unicode_ci		No	Ninguna		Cambiar Eliminar Primaria Único Más
5	asunto	varchar(255)	utf8_unicode_ci		No	Ninguna		Cambiar Eliminar Primaria Único Más
6	cuerpo	varchar(2025)	utf8_unicode_ci		No	Ninguna		Cambiar Eliminar Primaria Único Más
7	estado_mensaje_descripcion	varchar(255)	utf8_unicode_ci		No	Ninguna		Cambiar Eliminar Primaria Único Más
8	created_at	datetime			Sí	NULL		Cambiar Eliminar Primaria Único Más
9	updated_at	datetime			Sí	NULL		Cambiar Eliminar Primaria Único Más

Tabla Estado Mensaje

Bien, nada demasiado extraño aquí. Agregué un campo descripción, para poder describir el propósito del mensaje.

Ahora esta estructura de datos puede evolucionar con el tiempo, pero para comenzar las cosas, la mantuve simple.

Así que luego de crear la estructura de datos, pude ver que necesitaba una simple verificación para comprobar si un registro existía, y de ser así, devolver ya sea falso, indicando que no hay ningún mensaje para el controlador/acción o devolver la id del mensaje, que luego podría usar en otro método para recuperar las partes del mensaje que deseo enviar vía email.

¿Podría haber hecho todo en un método? Si, muy fácilmente, pero al dividirlo en múltiples partes, el código es más fácil de digerir, de escribir, y de reutilizar.

Ya tenemos el lugar perfecto para poner este método, en nuestra clase `RegistrosHelpers`, ubicada en `common/models/RegistrosHelpers.php`.

Primero agregue la sentencia `use` al comienzo para tener acceso al modelo vía `ActiveRecord`:

```
use backend\models\EstadoMensaje;
```

Luego, agreguemos este método:

Gist:

[EncontrarEstadoMensaje](#)

Del libro:

```

public static function encontrarEstadoMensaje($accion_nombre, $controlador_nombre)
{
    $result = EstadoMensaje::find('id')
        ->where(['accion_nombre' => $accion_nombre])
        ->andWhere(['controlador_nombre' => $controlador_nombre])
    ->one();

    return isset($result['id']) ? $result['id'] : false;
}

```

Como puede ver, realiza una simple búsqueda vía ActiveRecord, cuando le pasa el accion_nombre y el controlador_nombre. Obviamente tenemos un segundo parámetro y usamos andWhere() para ello.

Usamos un operador ternario para verificar si tenemos un resultado, si está todo bien, lo devolvemos. Si no devolvemos falso. El resultado nos dará la id del registro que entregaremos a los otros métodos. Muy simple.

Aquí están los otros dos métodos que recuperan el asunto y el cuerpo del mensaje, respectivamente. Agreguemoslos ahora a RegistroHelpers.php:

Gist:

[ObtenerMensajeAsunto y ObtenerMensajeCuerpo](#)

Del libro:

```

public static function obtenerMensajeAsunto($id)
{
    $result = EstadoMensaje::find('asunto')
        ->where(['id' => $id])
    ->one();

    return isset($result['asunto']) ? $result['asunto'] : false;
}

public static function obtenerMensajeCuerpo($id)
{

```

```
$result = EstadoMensaje::find('cuerpo')
    ->where(['id' => $id])
    ->one();

return isset($result['cuerpo']) ? $result['cuerpo'] : false;
\\

}
```

En ambos casos, entregamos la id que deseamos y obtenemos el asunto o el cuerpo del mensaje que necesitamos para el email. Más adelante añadiremos estos resultados a nuestro método enviarElMail, que será parte de nuestra clase MailCall que vamos a crear. No se preocupe, suena mucho más complicado de lo que es en realidad, es muy simple.

Bien, de regreso a nuestros nuevos métodos. Note que estamos usando métodos public static. Esto facilita ubicar el método dentro de otro método sin tener que usar la sintaxis de instanciación más larga. Verá lo que quiero decir en un minuto o dos.

En fin, así que ahora tenemos los auxiliares de registro que nos ayudarán a obtener datos desde la tabla.

Puede ver que estamos siguiendo el camino inverso a partir de los datos. Obviamente, necesitaremos un método mail para enviar el correo, una vez que tengamos las partes del mensaje que necesitamos.

Necesitaba descubrir como Yii 2 envía correo, así que, recordando que existe una página de contacto en la plantilla avanzada que envía un mensaje de correo, la use como guía para construir lo que necesitaba. Sólo menciono esto para señalar que puede usar mucho de lo que Yii 2 le entrega en sus plantillas como una guía para lo que desea construir.

Luego de estudiar el modelo ContactForm, construí mi método mail. Creamos un nuevo modelo para ello llamado MailCall. Continúe y cree MailCall.php dentro de common/models.

Cree una clase MailCall vacía con el siguiente espacio de nombres:

```
namespace common\models;
```

Luego agreguemos el siguiente método:

Gist:

[EnviarElMail](#)

Del libro:

```
public static function enviarElMail($mensaje_id)
{
    return Yii::$app->mailer->compose()
        ->setTo(\Yii::$app->user->identity->email)
        ->setFrom(['no-reply@yii2build.com' => 'Yii 2 Build'])
        ->setSubject(RegistrosHelpers::obtenerMensajeAsunto($mensaje_id))
        ->setTextBody(RegistrosHelpers::obtenerMensajeCuerpo($mensaje_id))
        ->send();
}
```

Bien, así que Yii 2 tiene una clase mailer, accesible desde la instancia de aplicación Yii::\$app. Ahora eso debería disparar un pequeño recuerdo de que debe incluir la sentencia use para Yii, de otro modo, no tendremos acceso a Yii::

Así que luego de namespace añada:

```
use yii;
```

Bien, así que puede ver que tenemos una gran cantidad de métodos encadenados juntos. Y ahora vemos la lógica detrás de que lo estamos haciendo si es que hasta ahora no estaba claro. El método setTo de la clase Mailer recibe como parámetro el email del usuario actual. Por ahora estamos fijando en el código la dirección y el nombre. Para setSubject, usamos nuestra útil llamada estática a:

```
RegistrosHelpers::obtenerMensajeAsunto($mensaje_id)
```

Así que nuevamente, eso debería indicarle que usaremos RegistrosHelpers, así que debemos incluirlo también. Agregue la siguiente sentencia use:

```
use common\models\RegistrosHelpers;
```

Este es código agradable y conciso para el cuerpo del mensaje, de igual manera:

```
RegistrosHelpers::obtenerMensajeCuerpo($mensaje_id)
```

Puede ver que el método recibe \$mensaje_id como parámetro, así que todo lo que necesitamos es un método que llame a enviarElMail, que tenga la habilidad de entregar la \$mensaje_id.

Así que ahora regreso a la pregunta del comienzo porque antes de construir el método, necesito saber cómo voy a convocarlo.

Estaba muy feliz con algo como:

```
public function afterAction($accion, $result)
{
    MailCall::esMailable($accion->id, getUniqueId());
    return parent::afterAction($accion, $result);
}
```

El plan era crear un método llamado esMailable en mi recientemente creada clase MailCall. Ya que estoy usando un método estático, simplemente puedo ponerlo allí. Hago eso a menudo para estos métodos auxiliares.

\$accion->id devuelve ‘accion’, así que si este fuera el controlador del sitio y la acción index fuera convocada, devolvería ‘index.’ Y getUniqueId() devolvería el nombre del controlador. Así que eso me dio las dos cosas que necesitaba para buscar un registro, ver si existe, y en ese caso, devolver la id y entregarsela al método enviarElMail.

Así que estaba muy contento. Tanto, que tomé un descanso y fui por un paseo. Y mientras estaba disfrutando de la agradable y fresca brisa del océano, aclarando mi cabeza, descubrí que estaba cometiendo un error.

El método afterAction se dispara por cada acción, que es lo que deseaba, pero no tiene forma de saber qué debe hacer como resultado de una acción exitosa. Por ejemplo, digamos que tiene una acción que guarda un registro o muestra un formulario para ingreso de datos, lo que se ve a menudo en los controladores. El método afterAction se dispara en ambos casos, sin importar lo que ocurra, porque en lo que a él respecta, se ha llamado a la acción. Pero usted desearía enviar el correo sólo en determinadas circunstancias, por ejemplo al guardar los datos, no al mostrar el formulario; así que usar afterAction quedó simplemente descartado.

Y así ocurre en la programación. Necesitaba una forma más discreta de determinar qué dispararía el email, y ya que eso podía variar grandemente de acción a acción, la única manera de hacerlo correctamente era ubicar la llamada al método dentro de la acción en el preciso momento donde quería ejecutarla.

Así que pensé nuevamente sobre mi idea original, que era anticipar las necesidades del cliente y darle control sobre el contenido de los mensajes de email enviados desde diferentes acciones. De la forma en que lo veía, no tenía opción, por un lado, podía embeber la llamada al método para enviar el email sólo cuando era solicitada de manera específica por el cliente.

Debido a que construí un sistema bastante robusto para almacenar mensajes y enviarlos, sería muy sencillo para mí añadir más ubicaciones a pedido del cliente, quién podría luego editarlas según deseara desde la IU. Este sería el enfoque estándar.

O, como alternativa, podría simplemente embeber la llamada dondequiera que pensara que podría ser necesaria en el futuro, ya que tenía planeado que ella verificara la existencia de un mensaje y devolviera falso en caso contrario.

Por supuesto no la pondría en cualquier lugar, sólo en sitios en los que probablemente se necesitaría un autoresponder. En ese caso la pregunta que surge es si vale la pena la sobrecarga para la BD. La llamada extra dondequiera que la ubicara podría ralentizar el sitio un poco.

Para dar un ejemplo más concreto de lo que estoy diciendo. Digamos que un cliente quiere un autoresponder en el registro. El usuario se registra y obtiene un email como confirmación. Nuestra clase y métodos, que no hemos completado aún, pero lo haremos en breve, manejarían esto elegantemente. Perfecto.

Pero observando nuestra plantilla, vemos que tenemos un formulario de contacto, ¿y no sería genial ser capaces de enviar una auto-respuesta cada vez que alguien nos contacte? Esta es una funcionalidad estándar en la mayoría de los sitios y el cliente puede no notarlo ahora, pero probablemente lo deseará.

Así que como un compromiso, podríamos embeber las llamadas a los métodos para enviar el mail dondequiera que pensemos que el cliente lo solicitará durante el desarrollo, luego, después de una revisión final con el cliente, remover las no deseadas del código. Esto necesitaría de un poco de trabajo de limpieza, pero en realidad sería muy sencillo de realizar.

De este modo, durante el desarrollo del sitio, cuando el cliente le ruegue agregar un autoresponder, puede decirle que simplemente agregue el registro a través de la IU, sin codificación adicional requerida. Esto lo mantendrá contento y usted estará un paso adelante.

Por supuesto el nivel de implementación depende de usted. Sólo menciono todo esto para que pueda ver que a medida que un proyecto avanza, las cosas tienden a cambiar. Quería construir un método de comportamiento que chequeara cada acción, pero eso resultó ser no realista, así que tomé una decisión de compromiso y opté por un método embebido en su lugar.

Bien, veamos lo que tenemos hasta ahora.

Podemos usar `RegistrosHelpers::encontrarEstadoMensaje($accion_nombre, $controlador_nombre)` para obtener la `$mensaje_id`, si existe un mensaje para esa acción y ese controlador en nuestra BD.

Podemos establecer la configuración del correo basados en `$mensaje_id`, lo que incluye el asunto y el cuerpo del mensaje, usando los métodos `RegistrosHelpers::obtenerMensajeAsunto($id)` y `RegistrosHelpers::obtenerMensajeCuerpo($id)`.

Podemos enviar el correo usando el método `MailCall::enviarElMail($message_id)`.

Así que ahora necesitamos el método que va a poner todo en marcha. Para tener una idea de lo que deseamos hacer, enfoquémonos en la auto-respuesta del registro. Aquí está el fragmento de la acción `signup` del controlador `site`:

```
public function actionSignup()
{
    $model = new SignupForm();
    if ($model->load(Yii::$app->request->post())) {
        if ($user = $model->signup()) {

            if (Yii::$app->getUser()->login($user)) {
```

```

        return $this->goHome();
    }
}

return $this->render('signup', [
    'model' => $model,
]);
}

```

Nada especial aquí, esto es lo que obtiene de manera predeterminada en la plantilla avanzada. Así que tenemos que agregar algo como:

```
MailCall::onMailableAction('signup', 'site');
```

Eso parece conciso y fácil de comprender. Realmente trato de reducirlo lo más posible y aún así ser intuitivo. MailCall es el nombre de la clase que creamos para el método enviarElMail. El método onMailableAction, no ha sido creado aún. Pero podemos ver que recibe dos parámetros, que son los nombres de los campos en nuestra BD.

Usaremos los nombres de la acción y del controlador para encontrar el mensaje específico que deseamos para nuestra acción.

Vale la pena notar que pensé en otra forma de referenciar los argumentos:

```
MailCall::onMailableAction(__METHOD__, getUserId());
```

La sintaxis en METHOD es un método mágico que devuelve el nombre del método actual y como dije anteriormente, getUserId() devuelve el nombre del controlador. Así que eso nos daría ‘actionSignup’ y ‘site’, que no es del todo correcto. Así que para usar este enfoque, deberíamos deshacernos de la palabra action.

Así que hice esto, de forma incorrecta:

```
$method_name = str_replace("action","", $method_name);
$action_name = $method_name;
```

Genial, elimina la palabra action reemplazandola con una cadena vacía. Si, ¿pero que ocurre si tenemos una acción llamada actionTraction? Eliminaría action de traction y devolvería un error. Obviamente podría simplemente remover los primeros seis caracteres de la cadena, lo que sería la forma correcta de hacerlo. Pero nunca me molesté en escribirlo de esa manera, aquí está el por qué.

No quería tener que recordar lo que devuelven el método mágico METHOD y getUniqueId(). Además, si alguien más tuviese que mantener el código, no sería capaz de comprender inmediatamente lo que ocurre. También, nombrar explícitamente al controlador y la acción requeriría menos código.

En ocasiones simplemente se reduce a una preferencia sobre lo que desea contemplar. Usted pasará mucho más tiempo leyendo código que escribiéndolo. Así que mientras más fácil de leer a primera vista sea ese código, mejor.

En cuanto al nombre del método, uso onMailableAction porque es ligeramente ambiguo, lo que considero apropiado debido a que nada va a ocurrir si no hay un mensaje correspondiente en la BD.

Bien, añadamos el último método en MailCall para completar esto. Agregue esto a su clase MailCall.

Gist:

[OnMailableAction](#)

Del libro:

```
public static function onMailableAction($accion_nombre, $controlador_nombre)
{
    if ($mensaje_id = RegistrosHelpers::encontrarEstadoMensaje
        ($accion_nombre, $controlador_nombre)){
        static::enviarElMail($mensaje_id);
    }
}
```

Obviamente la sentencia está dividida en dos líneas debido al ajuste de texto. Debería usarla como una única línea en su IDE.

Es tan simple. Entrega el nombre de la acción y del controlador, luego usa un método auxiliar para tratar de obtener un registro de la BD y asignarlo a \$mensaje_id. Si no puede establecer \$mensaje_id, entonces nunca llama a MailCall. Si puede establecer \$mensaje_id correctamente, entonces llama al método enviarElMail pasándole la \$mensaje_id.

Por consistencia, proveeré del archivo MailCall.php completo.

Gist:

[MailCall.php](#)

Del libro:

```
<?php

namespace common\models;

use yii;
use common\models\RegistrosHelpers;

class MailCall
{

    public static function enviarElMail($mensaje_id)
    {
        return Yii::$app->mailer->compose()
            ->setTo(\Yii::$app->user->identity->email)
            ->setFrom(['no-reply@yii2build.com' => 'Yii 2 Build'])
            ->setSubject(RegistrosHelpers::obtenerMensajeAsunto($mensaje_id))
            ->setTextBody(RegistrosHelpers::obtenerMensajeCuerpo($mensaje_id))
            ->send();
    }

    public static function onMailableAction($accion_nombre, $controlador_nombre)
    {
        if ($message_id = RegistrosHelpers::encontrarEstadoMensaje
            ($accion_nombre, $controlador_nombre)) {

            static::enviarElMail($mensaje_id);

        }
    }
}
```

Nuevamente la sentencia if está dividida en dos líneas debido al ajuste de texto. Debería usarla como una única línea en su IDE.

Así que para llamar correctamente en nuestra acción signup en el controlador site:

Gist:

[Action Signup](#)

Del libro:

```

public function actionSignup()
{
    $model = new SignupForm();
    if ($model->load(Yii::$app->request->post())) {
        if ($user = $model->signup()) {

            if (Yii::$app->getUser()->login($user)) {

                MailCall::onMailableAction('signup', 'site');

            }

        }

    }

    return $this->render('signup', [
        'model' => $model,
    ]);
}

```

Puede ver que agregamos la llamada a nuestro método onMailableAction justo después de que el usuario inicia sesión. Esto es porque, como recuerda de nuestro método enviarElMail, estamos usando:

```
->setTo(\Yii::$app->user->identity->email)
```

Y sólo tiene acceso a Yii::\$app->user->identity->email si el usuario ha iniciado sesión.

También tenga en cuenta, que se trata de una línea de código en el controlador, todo lo demás está separado en otras clases. ¿Cuán genial es eso? Código de controlador agradable y sencillo.

Simplemente no olvide agregar MailCall en SiteController:

```
use common\models\MailCall;
```

En este punto, ya que no tenemos registros en nuestra BD, no debería ocurrir nada, excepto el registro normal de un usuario. Sin embargo debería registrar un usuario para ver si no se presentan errores.

Ahora que hemos comprobado que todo marcha bien, usemos Gii para construir nuestra IU para el backend, de forma que podamos agregar algunos registros y jugar con ellos.

Podemos avanzar rápidamente en esta parte, ya que estamos acostumbrados a Gii a estar alturas, y aquí es donde puede ver lo genial que es la generación de código para constuir código rápidamente.

Bien, primer paso, construir el modelo. No proveeremos capturas de pantalla ya que debería saber cómo hacerlo en este punto. Sí mencionaré, sin embargo, que mi elección es ubicar el modelo en el backend, lo que me parece lógico.

Se que he mencionado anteriormente, que Yii 2 recomienda poner los modelos en el directorio common y luego extenderlos en el frontend y backend, pero típicamente esa no es mi opción personal. Y puede ver a medida que agregamos modelos a common, lo útil que es tener un directorio reservado para clases auxiliares y otros modelos.

Bien, así que voy a asumir que ha construido el modelo estado_mensaje, y luego construido el CRUD a partir de EstadoMensaje. Estoy seguro de que no necesito señalar lo asombroso que es Gii a estas alturas, pero lo haré sin embargo. 8 archivos están siendo generados para usted y sólo requerirán de cambios menores. No sólo obtiene estandarización, sino que logra una enorme cantidad de eficiencia tanto en tiempo como esfuerzo.

Así que movamonos a través de esos cambios menores. Agreguemos nuestro timestamp behavior al modelo, EstadoMensaje.php:

Gist:

[Behaviors de EstadoMensaje](#)

Del libro:

```
public function behaviors()
{
    return [
        'timestamp' => [
            'class' => 'yii\behaviors\TimestampBehavior',
            'attributes' => [
                ActiveRecord::EVENT_BEFORE_INSERT => ['created_at', 'updated_at'],
                ActiveRecord::EVENT_BEFORE_UPDATE => ['updated_at'],
            ],
            'value' => new Expression('NOW()'),
        ],
    ];
}
```

Y por supuesto sabe que eso significa que tendrá que agregar:

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

Y también:

```
use yii\db\Expression;
```

Para ser consistentes, cambiemos el método behaviors en nuestro EstadoMensajeController de igual manera que en los otros controladores de administración, tales como rol. Así que reemplácelo con lo siguiente:

Gist:

[EstadoMensajeController](#)

From book:

```
public function behaviors()
{
    return [
        'access' => [
            'class' => \yii\filters\AccessControl::className(),
            'only' => ['index', 'view', 'create', 'update', 'delete'],
            'rules' => [
                [
                    'actions' => ['index', 'create', 'update', 'view'],
                    'allow' => true,
                    'roles' => ['@'],
                    'matchCallback' => function ($rule, $action) {
                        return PermisosHelpers::requerirMinimoRol('Admin')
                            && PermisosHelpers::requerirEstado('Activo');
                    }
                ],
                [
                    'actions' => ['delete'],
                    'allow' => true,
                    'roles' => ['@'],
                    'matchCallback' => function ($rule, $action) {
                        return PermisosHelpers::requerirMinimoRol('SuperUsuario')
                            && PermisosHelpers::requerirEstado('Activo');
                    }
                ],
            ],
        ],
    ];
}
```

```
        ],
    ],
},
'verbs' => [
    'class' => VerbFilter::className(),
    'actions' => [
        'delete' => [ 'post' ],
    ],
],
];
}
```

Una diferencia que puede ver es que he movido la acción update a nivel de acceso de admin. Y no olvide agregar:

```
use common\models\PermisosHelpers;
```

Ahora podemos trabajar en las vistas. Puede cambiar un campo `textInput` en `backend/views/estado-mensaje/_form.php` a lo siguiente:

```
<?= $form->field($model, 'cuerpo')->  
textArea(['maxlength' => 2025, 'rows' =>12]) ?>
```

Dos líneas para evitar el odioso ajuste de línea nuevamente. Use una línea en su IDE.

Así que reemplazamos el `textInput` por una `textArea` y le pasamos el parámetro opcional `rows`, que permite controlar el tamaño del cuadro de texto. También moví `estado_mensaje_descripcion` y lo puse directamente debajo de `estado_mensaje_nombre` porque ese parece un orden más lógico para los campos.

Y obviamente podemos deshacernos de los campos `created_at` y `updated_at`, ya que tenemos behaviors para encargarse de ellos.

Y como último paso, necesitamos agregar la navegación a nuestras vistas. Regresemos a backend/-views/layouts/main.php y agreguemos un nuevo ítem de menú:

```
$menuItems[] = [ 'label' => 'Estado Mensajes', 'url' => ['estado-mensaje/index']] ;
```

Para los fines de la IU, he nombrado la etiqueta en plural. Note la forma en que el controlador es referenciado. La convención es poner un guión entre las dos palabras, aunque el nombre del archivo del controlador es EstadoMensajeController.php.

Esta IU está comenzando a ponerse rara con tantos items en la navegación. No se preocupe, realizar el cambio a una lista desplegable es la siguiente sección de nuestro material extra, así que lo cambiaremos.

Bien, ahora, si guarda los cambios, podrá ver todas las vistas y probar el funcionamiento. Proveeré de capturas de pantallas de mi primer registro, sólo para asegurarme de que comprendemos qué significa cada campo, ya que cubrimos ese punto hace algún tiempo.

Create Estado Mensaje

Controlador Nombre

Acción Nombre

Estado Mensaje Nombre

Estado Mensaje Descripción

Asunto

Cuerpo

¡Gracias por registrarse!

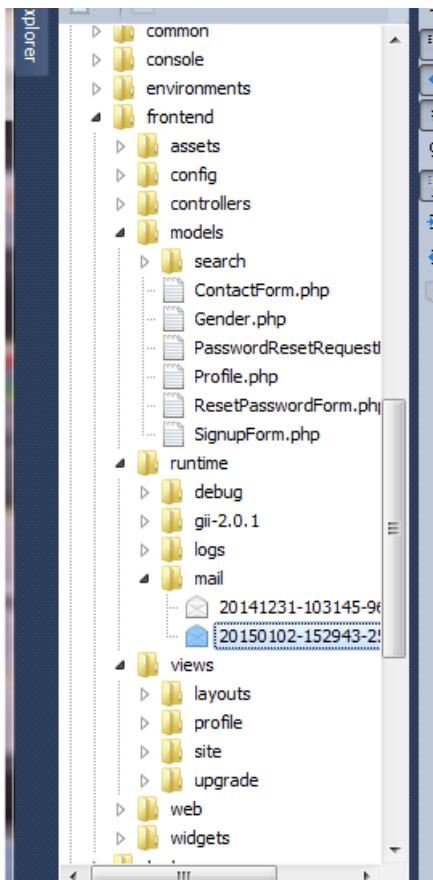
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec mi purus, placerat molestie varius porta, aliquam et libero. Nulla malesuada lorem quis mollis faucibus. Vivamus posuere porttitor diam, vitae placerat ante pellentesque eu. Suspendisse nulla lorem, incidunt sit amet maximus sed, pellentesque a odio. In scelerisque, elit vitae fringilla cursus, ipsum diam imperdiet metus, eget luctus ex erat eget turpis. Donec vel ligula a arcu varius convallis id et augue. Nunc porta venenatis massa, vel faucibus metus tempor congue. Vestibulum faucibus congue lacus eu tempor. Morbi at est vitae magna pulvinar aliquet vel sit amet nunc.

Create

Status Message Input

Comenzamos con el nombre del controlador. Luego el nombre de la acción, sensible a mayúsculas, no incluya la palabra acción. Luego se encuentra el nombre del estado mensaje, seguido por una breve descripción. Luego tenemos el asunto y el cuerpo. Así que continúe y guarde este registro.

Luego, registre un nuevo usuario. Luego chuequé su directorio frontend/runtime/mail y verá su email allí.



Runtime Mail

Se que lo he mencionado antes, pero este es el lugar a donde van todos los correos en modo desarrollo. Compruébelo y verá que en efecto se ha enviado un correo, con el contenido correcto.

Ahora en cualquier lugar de la aplicación donde desee un autorepsonder in the application, simplemente agregue en el controlador:

```
MailCall::onMailableAction($accion_nombre, $controlador_nombre);
```

Y obviamente, debe usar cadenas en la firma del método, no variables. Así que simplemente por mayor claridad, si quisiera ponerlo en cualquier lugar de la acción contact del controlador site:

```
MailCall::onMailableAction('contact', 'site');
```

Si todavía no ha incluido la sentencia use para MailCall, deberá agregarla también. Luego simplemente cree un registro estado mensaje que concuerde y funcionará. Puede ver lo sencillo que será implementar esto en la aplicación.

Y básicamente eso es todo. Como en todo código, no está pensado para ser una solución final o grandiosa, sino simplemente una posibilidad entre muchas. Muchas de las decisiones que realiza

al codificar se reducen a una preferencia personal y típicamente siempre hay lugar para mejorar. ¡Simplemente hágalo a su manera y diviértase!

Navegación mediante Dropdown

Es momento de usar navegación mediante dropdown en la parte superior de nuestro backend, de otra manera terminaremos con demasiado desorden. Esta debería ser una tarea sencilla, ¿verdad?

Desafortunadamente, no está claro en la documentación de Yii 2 la forma en que trabaja el widget dropdown, al menos para mí. No estaba satisfecho con que no funcionara, así que busqué y googleé como loco. Vi una implementación en el plugin de alguien más y descubrí que el widget Nav de Yii 2 podía funcionar de la misma forma, y así encontré una solución.

La implementación en cuestión se ve así:

```
echo Nav::widget([
    'options' => ['class' => 'navbar-nav navbar-right'],
    'items' => [
        ['label' => 'Usuarios', 'items' => [
            ['label' => 'Usuarios', 'url' => ['user/index']],
            ['label' => 'Perfiles', 'url' => ['perfil/index']],
            ['label' => 'Algo más aquí', 'url' => ['#']],
        ]],
        ['label' => 'Soporte', 'items' => [
            ['label' => 'Solicitudes de soporte', 'url' => ['/content/index']],
            ['label' => 'Mensajes de Estado', 'url' => ['/estado-mensaje/index']],
            ['label' => 'FAQ', 'url' => ['/faq/index']],
            ['label' => 'Categorías FAQ', 'url' => ['/faq-category/index']],
        ]],
        ['label' => 'RBAC', 'items' => [
            ['label' => 'Roles', 'url' => ['/rol/index']],
            ['label' => 'Tipos de Usuario', 'url' => ['/tipo-usuario/index']],
            ['label' => 'Estados', 'url' => ['/estado/index']],
        ]],
        ['label' => 'Content', 'items' => [
            ['label' => 'Content', 'url' => ['/content/index']],
        ]]];
```

```

['label' => 'Mensajes de Estado', 'url' => ['/estado-mensaje/index']],
['label' => 'FAQ', 'url' => ['/faq/index']],
['label' => 'Categorías de FAQ', 'url' => ['/faq-category/index']],
],
],
]);

```

No he provisto del Gist a propósito. Le daré el archivo completo cuando haya finalizado de explicar todo. Puede ver arriba, que simplemente he usado arreglos anidados para crear el dropdown.

Como sabe de capítulos anteriores, he encerrado las partes de la navegación correspondientes a la administración con una sentencia if para determinar si el usuario ha iniciado sesión y si tiene un estado mínimo de admin:

```
if (!Yii::$app->user->isGuest && $es_admin) {
```

Así que usted podría pensar que simplemente podría poner esto en el lugar donde antes tenía la navegación, pero no. Tenemos que poner la prueba para verificar si el usuario está logueado o no arriba:

```

if (Yii::$app->user->isGuest) {

$menuItemsLogOut[] = ['label' => 'Login', 'url' => ['site/login']];

} else {

$menuItemsLogOut[] =

['label' => 'Logout (' . Yii::$app->user->identity->username . ')',
'url' => ['/site/logout'],
'linkOptions' => ['data-method' => 'post']
];

}

echo Nav::widget([
'options' => ['class' => 'navbar-nav navbar-right'],
'items' => $menuItemsLogOut
]);
```

Porque estamos usando:

```
'options' => ['class' => 'navbar-nav navbar-right'],
```

Lleva todo a la derecha. Así que cada subsiguiente Nav::widget se apila a la izquierda del primero. Así que en esencia tengo que invertir el orden.

También, puede ver que he usado dos arreglos \$menuItems diferentes. Esto es debido a que ahora tenemos múltiples llamadas a Nav::Widget, así que no podemos usar el mismo arreglo \$menuItems para ambos.

En esta sentencia if:

```
if (Yii::$app->user->isGuest) {

$menuItemsLogOut[] = ['label' => 'Login', 'url' => ['site/login']];

} else {

$menuItemsLogOut[] =

['label' => 'Logout (' . Yii::$app->user->identity->username . ')',
 'url' => ['/site/logout'],
 'linkOptions' => ['data-method' => 'post']
];

}

echo Nav::widget([
'options' => ['class' => 'navbar-nav navbar-right'],
'items' => $menuItemsLogOut
]);
```

En primer lugar estamos estableciendo un elemento del arreglo, dependiendo de si el usuario ha iniciado sesión o no. Luego llamamos al arreglo desde dentro de Nav::Widget.

Esta es una técnica útil para probar el estado del usuario, y luego decidir que mostrarle. Note que en la sentencia if donde decidimos si mostrar o no los items, no necesitamos arreglos separados, porque se trata de una decisión entre mostrar los items o nada.

Como dije, esto puede ser un poco difícil. Voy a proporcionarle el archivo completo backend/views/layouts/main.php file, para que podamos estar seguros de que tenemos todo en el orden correcto:

Gist:

[Backend Layouts Main](#)

Del libro:

```
<?php

use backend\assets\AppAsset;
use yii\helpers\Html;
use yii\bootstrap\Nav;
use yii\bootstrap\NavBar;
use yii\widgets\Breadcrumbs;
use common\models\PermisosHelpers;
use backend\assets\FontAwesomeAsset;

/**
 * @var \yii\web\View $this
 * @var string $content
 */

AppAsset::register($this);
FontAwesomeAsset::register($this);

?>

<?php $this->beginPage() ?>

<!DOCTYPE html>

<html lang="<?= Yii::$app->language ?>">

<head>
    <meta charset="<?= Yii::$app->charset ?>" />

    <meta name="viewport"
        content="width=device-width,
        initial-scale=1">

        <?= Html::csrfMetaTags() ?>

<title><?= Html::encode($this->title) ?></title>

        <?php $this->head() ?>

</head>
```

```
<body>
    <?php $this->beginBody() ?>

    <div class="wrap">

        <?php

        if (!Yii::$app->user->isGuest){

            $es_admin = PermisosHelpers::requerirMinimoRol('Admin');

            NavBar::begin([
                'brandLabel' => 'Yii 2 Built <i class="fa fa-plug"></i> Admin',
                'brandUrl' => Yii::$app->homeUrl,
                'options' => [
                    'class' => 'navbar-inverse navbar-fixed-top',
                ],
            ]);

        } else {

            NavBar::begin([
                'brandLabel' => 'Yii 2 Built <i class="fa fa-plug"></i>',
                'brandUrl' => Yii::$app->homeUrl,
                'options' => [
                    'class' => 'navbar-inverse navbar-fixed-top',
                ],
            ]);

        }

        if (Yii::$app->user->isGuest) {

            $menuItemsLogOut[] = ['label' => 'Login', 'url' => ['site/login']];

        } else {

    }
```

```
$menuItemsLogOut[] = [
    'label' => 'Logout (' . Yii::$app->user->identity->username . ')',
    'url' => ['/site/logout'],
    'linkOptions' => ['data-method' => 'post']
];
}

echo Nav::widget([
    'options' => ['class' => 'navbar-nav navbar-right'],
    'items' => $menuItemsLogOut
]);

if (!Yii::$app->user->isGuest && $es_admin) {

echo Nav::widget([
    'options' => ['class' => 'navbar-nav navbar-right'],
    'items' => [

        ['label' => 'Usuarios', 'items' => [
            ['label' => 'Usuarios', 'url' => ['user/index']],
            ['label' => 'Perfiles', 'url' => ['perfil/index']],
            ['label' => 'Algo más aquí', 'url' => ['#']],
        ],

        ['label' => 'Soporte', 'items' => [
            ['label' => 'Solicitudes de Soporte', 'url' => ['content/index']],
            ['label' => 'Mensajes de Estado', 'url' => ['estado-mensaje/index']],
            ['label' => 'FAQ', 'url' => ['faq/index']],
            ['label' => 'Categorías de FAQ', 'url' => ['faq-category/index']],
        ]],

        ['label' => 'RBAC', 'items' => [
            ['label' => 'Roles', 'url' => ['rol/index']],
            ['label' => 'Tipos de Usuario', 'url' => ['tipo-usuario/index']],
            ['label' => 'Estados', 'url' => ['estado/index']],
        ]],

        ['label' => 'Content', 'items' => [
            ['label' => 'Content', 'url' => ['content/index']],
            ['label' => 'Mensajes de Estado', 'url' => ['estado-mensaje/index']],
            ['label' => 'FAQ', 'url' => ['faq/index']],
        ]],
    ],
]);
```

```
        ['label' => 'Categoría de FAQ', 'url' => ['faq-category/index']] ,
    ],
],
]);
}

$menuItems = [['label' => 'Home', 'url' => ['site/index']] ,
];
echo Nav::widget([
    'options' => ['class' => 'navbar-nav navbar-right'],
    'items' => $menuItems
]);
}

NavBar::end();

?>

<div class="container">

<?= Breadcrumbs::widget([
    'links' => isset($this->params['breadcrumbs']) ? 
$this->params['breadcrumbs'] : [],
])?>

<?= $content ?>

    </div>
</div>

<footer class="footer">

    <div class="container">
```

```

<p class="pull-left">&copy; Yii 2 Build <?= date('Y') ?></p>

<p class="pull-right"><?= Yii::powered() ?></p>

</div>

</footer>

<?php $this->endBody() ?>

</body>
</html>

<?php $this->endPage() ?>

```

Algo que habrá notado en el archivo presentado arriba, es que no realizamos un echo de Nav-Bar::begin. Si necesitamos, sin embargo, realizar un echo de los elementos nav.

También note, que he puesto algunos placeholders en ciertos lugares del dropdown con fines de demostración.

Cuando experimente con este, probablemente notará el genial efecto de resultado que le otorga a la opción seleccionada en el dropdown. Al principio pensé que se trataba de un error del CSS hasta que descubrí que estaba resaltando el elemento del dropdown correspondiente a la página donde me encontraba. Cuando termine, debería verse como la imagen siguiente:



Dropdown Nav

Bien, eso es todo para la navegación, espero que le haya sido de utilidad.

FAQ

A continuación vamos a construir un modelo FAQ y usar algunos elementos interesantes que no hemos utilizado antes, incluyendo el ArrayDataProvider de Yii 2, un iterador muy útil para enviar datos a las vistas.

Cuando pensamos en FAQs, pensamos en simplicidad, simplemente preguntas y respuestas, y la simplicidad nos funciona bien. Pero si pensamos en nuestros clientes demandantes y las cosas que

son importantes para ellos, el orden de las preguntas, por ejemplo, puede ser muy importante. Y obviamente, el orden que ellos desean no se corresponderá con la fecha de creación o el orden alfabético.

En la mayoría de los casos, el cliente ni siquiera sabrá en qué orden desea que aparezcan las preguntas hasta más adelante en el desarrollo del proyecto. Así que lo que necesitamos es una forma de que puedan determinar el orden a través de la IU. Ahora esto es realmente simple de realizar por supuesto. Simplemente necesitamos una columna faq_value, con un tipo de dato int, que podamos ordenar en forma ascendente o descendente dependiendo de nuestra preferencia.

Pero pensandolo nuevamente, faq_value como nombre de columna puede ser demasiado genérico. Podría llamarlo de una manera más descriptiva como faq_importance o faq_weight. Pienso que faq_importance es demasiado para tipar y propenso a errores, pero faq_weight parece una buena opción.

La razón por la que no la llamo simplemente weight, es que podría usar el concepto weight en otra tabla e intento evitar tener el mismo nombre de columna en tablas diferentes tanto como sea posible. Esto me ayuda a evitar problemas de ambigüedad con las consultas más adelante.

Así que la forma en que faq_weight funcionaría sería que en orden ascendente, una FAQ con faq_weight de 10 estaría más arriba en la lista que una FAQ con faq_weight de 20. Y en tanto le demos al cliente un método para cambiar el faq_weight de una FAQ en la IU del backend, la necesidad del cliente de controlar el orden está siendo anticipada y resuelta antes de que se convierta en un problema.

Así que pensando de la misma manera, ¿qué ocurre si un cliente quiere tomar un grupo de preguntas específicas y presentarlas en diferentes partes de la aplicación? Podemos anticipar que desearán eso, así que podemos crear un campo booleano faq_is_featured. De esa forma si necesitamos llamar a un grupo de FAQs destacadas para una presentación especial, tenemos una manera sencilla de extraer esas FAQa, simplemente llamamos a aquellas donde faq_is_featured esté establecido al valor si, o en otras palabras, a 1.

Así que comenzemos y establezcamos la estructura para FAQ. Necesitaremos las siguientes columnas en nuestra tabla faq:

id (int, PK, NN, AI)

faq_question (VARCHAR(255), NN)

faq_answer (VARCHAR(1055), NN)

faq_category_id (INT)

faq_is_featured (BOOL, DEFAULT = 0)

faq_weight (INT, DEFAULT = 100)

created_by (INT)

updated_by (INT)

created_at (DATETIME)

updated_at (DATETIME)

Aquí tenemos una captura de pantalla:

Column Name	Datatype	PK	NN	UQ	BIN	UN	ZF	AI	Default
id	INT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>					
faq_question	VARCHAR(255)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>					
faq_answer	VARCHAR(1055)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>					
faq_category_id	INT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>					
faq_is_featured	BOOL	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	'0'
faq_weight	INT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	'100'
created_by	INT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
updated_by	INT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
created_at	DATETIME	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
updated_at	DATETIME	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	

Columns Indexes Foreign Keys Triggers Partitioning Options Inserts Privileges

faq table

También necesitamos una tabla para faq_category. Aquí está la estructura de datos:

id (INT, PK, NN, AI)

faq_category_name (VARCHAR(45))

faq_category_weight (INT, DEFAULT = 100)

faq_category_is_featured (Bool, DEFAULT = 0)

Aquí hay una captura de pantalla:

Column Name	Datatype	PK	NN	UQ	BIN	UN	ZF	AI	Default
id	INT(11)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>					
faq_category_name	VARCHAR(45)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>					
faq_category_weight	INT(11)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	'100'
faq_category_is_featured	TINYINT(1)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	'0'

faq table

Bien, así que puede ver que hemos preparado una tabla faq_category que contendrá los nombres de las categorías. Y simplemente para mantenernos un paso adelante, he establecido dentro de la estructura de datos el mismo tipo de ordenamiento que establecí para faq. Aunque no lo usaremos en este capítulo, está allí, por si decidiera implementarlo.

Si nuestro molesto cliente se enamora el ordenamiento de las FAQs, podría demandar el mismo control sobre faq_categories.

Los resultados en PhpMyadmin deberían verse así:

#	Name	Type	Collation	Attributes	Null	Default	Extra	Action
1	id	int(11)			No	None	AUTO_INCREMENT	Change Drop Primary Unique Index Spatial Fulltext Distinct values
2	faq_question	varchar(255)	latin1_general_ci		No	None		Change Drop Primary Unique Index Spatial Fulltext Distinct values
3	faq_answer	varchar(1055)	latin1_general_ci		No	None		Change Drop Primary Unique Index Spatial Fulltext Distinct values
4	faq_category_id	int(11)			Yes	NULL		Change Drop Primary Unique Index Spatial Fulltext Distinct values
5	faq_is_featured	tinyint(1)			Yes	0		Change Drop Primary Unique Index Spatial Fulltext Distinct values
6	faq_weight	int(11)			Yes	100		Change Drop Primary Unique Index Spatial Fulltext Distinct values
7	created_by	int(11)			Yes	NULL		Change Drop Primary Unique Index Spatial Fulltext Distinct values
8	updated_by	int(11)			Yes	NULL		Change Drop Primary Unique Index Spatial Fulltext Distinct values
9	created_at	datetime			Yes	NULL		Change Drop Primary Unique Index Spatial Fulltext Distinct values
10	updated_at	datetime			Yes	NULL		Change Drop Primary Unique Index Spatial Fulltext Distinct values

Up Check All With selected: [Browse](#) [Change](#) [Drop](#) [Primary](#) [Unique](#) [Index](#)

[Print view](#) [Relation view](#) [Propose table structure](#) [Track table](#) [Move columns](#)

[Add](#) column(s) At End of Table At Beginning of Table After [Go](#)

+ Indexes

tabla faq

Y para faq_category:

#	Name	Type	Collation	Attributes	Null	Default	Extra	Action
1	id	int(11)			No	None	AUTO_INCREMENT	Change Drop Browse distinct value
2	faq_category_name	varchar(45)	latin1_general_ci		No	None		Change Drop Browse distinct value
3	faq_category_weight	int(11)			Yes	100		Change Drop Browse distinct value
4	faq_category_is_featured	tinyint(1)			Yes	0		Change Drop Browse distinct value

Up Check All / Uncheck All With selected: [Browse](#) [Change](#) [Drop](#) [Primary](#) [Unique](#) [Index](#)

[Print view](#) [Relation view](#) [Propose table structure](#) [Track table](#)

[Add](#) column(s) At End of Table At Beginning of Table After [Go](#)

+ Indexes

[Information](#)

Space usage	Row Statistics	
Data 16 KiB	Format	Compact
Index 0 B	Collation	latin1_general_ci
Next autoindex ,		

faq category table

Algo más que puede haber notado en nuestra estructura de datos son las columnas created_by y updated_by en nuestra tabla faq. Estas columnas nos permitirán implementar un behavior denominado blameable en el modelo que guardará de forma automática la id del usuario que crea o actualiza el registro. ¿No le parece maravilloso el nombre blameable (culpable)?

A veces es importante saber quién ha creado un registro, hay muchas razones para ello. Podría ser que no entendemos por qué existe, así que tenemos que preguntárselo al miembro del equipo que lo creó. Ese sería un escenario para el backend.

En el frontend, si alguien publica algo que de otra manera no registra al usuario que lo hace, esta es una buena forma de conseguirlo, lo que es útil por razones de seguridad.

Así que esos son un par de escenarios donde le gustaría saber quién es el usuario, y obviamente hay mucho más. Blameable es realmente un behavior útil que nos provee Yii 2, así que lo implementaremos aquí, y luego puede decidir en qué otra parte de la aplicación le gustaría usarlo.

Otra cosa que vamos a hacer diferente aquí, es tomar un modelo del backend y crear una IU separada para el frontend, esta vez utilizando el ArrayDataProvider que aún no hemos usado.

Continuemos y comencemos utilizando Gii para crear dos nuevos modelos, Faq y FaqCategory, que están basados en las dos tablas que recién construimos. Contruiremos estos modelos en el backend, así que asegúrese de que el espacio de nombres sea backend\models.

No voy a proveer de capturas de pantalla para esto porque ya debería saber como usar Gii. Si necesita un recordatorio, por favor refiérase a uno de los capítulos anteriores.

Ahora realizamos el crud para ambos modelos. Recuerde que no necesitamos proveer un path para view, usaremos el valor por defecto.

Nuevamente, no voy a proveer de capturas de pantalla para esto porque debería saber como generar el CRUD en Gii. Si necesita un recordatorio, por favor refiérase a los capítulos anteriores.

Para cada modelo, debería tener un archivo para:

- model
- search model
- controller
- view
- update
- index
- create
- _search
- _form

Continúe y revise sus directorios en el backend y asegúrese de que tiene todo lo necesario. Si algo no está bien, regrese sobre sus pasos para encontrar lo que ha salido mal. Si faltan archivos, y los ha generado, probablemente se encuentran en el lugar equivocado, lo que puede ocurrir si comete un error en las entradas de los espacios de nombre.

En este punto, asumiré que todo está en orden y continuaré modificando los archivos. Tenemos que realizar algunos cambios, y comenzaremos con el modelo FaqCategory.php primero.

Ya que he usado una clave foránea en MySQL Workbench para crear la relación entre Faq y FaqCategory, Gii ha agregado automáticamente el método de relación necesario. Lo mostraré aquí en caso de que haya omitido la clave foránea:

Gist:

[Get Faqs](#)

Del libro:

```
public function getFaqs()
{
    return $this->hasMany(Faq::className(), ['faq_category_id' => 'id']);
}
```

Lo siguiente que queremos añadir a nuestro modelo es un método simple para devolver una lista desplegable de opciones para faq_category_is_featured. La necesitamos para nuestra vista, para poder devolver si o no en la lista en lugar de tener 0 o 1 en el campos del formulario.

Hemos cubierto esto previamente cuando la lista desplegable fue creada a partir de valores en un modelo relacionado, pero esta es la primera vez que lo haremos de esta forma. Esto es simplemente formatear los datos para la vista, no es necesaria ninguna relación. Pero lo haremos de una forma tal que sea muy consistente con la manera acostumbrada de agregar nuestros otros métodos al modelo, en lugar de hacerlo en la vista.

Aquí está el método:

Gist:

[GetFaqCategoryIsFeaturedList](#)

Del libro:

```
public static function getFaqCategoryIsFeaturedList()
{
    return $droptions = [0 => "no", 1 => "yes"];
}
```

La guía de Yii 2 muestra una versión de esto en el formulario, pero prefiero crear un método reusable. Esto evita duplicación de código porque es muy probable que pudiera terminar usando la lista desplegable en más de un lugar.

También, otro beneficio es que cuando está creando relaciones en su modelo, se acostumbra a generar una lista desplegable, la que podría ser usada típicamente en sus reglas, aunque no lo haremos en esta ocasión.

Hablando de reglas, también necesitamos un poco de trabajo aquí, para que tengamos nuestro conjunto de la forma deseada:

Gist:

FaqCategory Rules

Del libro:

```
public function rules()
{
    return [
        [['faq_category_name'], 'required'],
        [['faq_category_weight'], 'faq_category_is_featured'], 'integer'],
        ['faq_category_weight', 'default', 'value' => 100],
        [['faq_category_weight'], 'in', 'range'=>range(1,100)],
        [['faq_category_name']], 'string', 'max' => 45]
    ];
}
```

Agregamos una regla para un valor por defecto asegurándonos de que sea establecido, y forzamos un rango para el peso de la categoría, de 1 a 100, usando la función de php range.

A continuación, vayamos a FaqCategoryController y cambiemos el método behaviors para hacer lo similar a nuestros otros controladores, pero en esta ocasión sólo añadiremos un arreglo de reglas:

Gist:

FaqCategoryController

Del libro:

```
public function behaviors()
{
    return [
        'access' => [
            'class' => \yii\filters\AccessControl::className(),
            'only' => ['index', 'view', 'create', 'update', 'delete'],
            'rules' => [
                [
                    'actions' => ['index', 'view', 'create', 'update', 'delete'],
                    'allow' => true,

```

```

        'roles' => ['@'],
        'matchCallback' => function ($rule, $action) {
            return PermisosHelpers::requirirMinimoRol('Admin')
                && PermisosHelpers::requirirEstado('Active');
        }
    ],
],
],
];

'verbs' => [
    'class' => VerbFilter::className(),
    'actions' => [
        'delete' => ['post'],
    ],
],
];
}

```

No olvida incluir la sentencia use en el controlador de FaqCategory:

```
use common\models\PermisosHelpers;
```

Ahora podemos trabajar en nuestras vistas. Modifiquemos la vista _form en primer lugar:

Gist:

FaqCategory Form View

Del libro:

```
<?php

use yii\helpers\Html;
use yii\widgets\ActiveForm;

/* @var $this yii\web\View */
/* @var $model backend\models\FaqCategory */
/* @var $form yii\widgets\ActiveForm */
?>

<div class="faq-category-form">
```

```
<?php $form = ActiveForm::begin(); ?>

<?= $form->field($model, 'faq_category_name')->
textInput(['maxlength' => 45]) ?>

<?= $form->field($model, 'faq_category_weight')->textInput() ?>

<?= $form->field($model, 'faq_category_is_featured')->
dropDownList($model->faqCategoryIsFeaturedList,
[ 'prompt' => 'Por favor seleccione uno' ]) ?>

<div class="form-group">

<?= Html::submitButton($model->isNewRecord ? 'Create' : 'Update',
['class' => $model->isNewRecord ? 'btn btn-success' : 'btn btn-primary']) ?>

</div>

<?php ActiveForm::end(); ?>

</div>
```

Ahora que hemos tenemos nuestro formulario, continuemos y agreguemos un registro. Inicie sesión en backend.yii2build.com y use la lista desplegable Content del menú para encontrar Categoría de FAQ. Notará que en la navegación utiliza el singular, pero en la página, está en plural, así que tomemos un momento para arreglar la navegación en main.php:

```
[ 'label' => 'Categorías de FAQ', 'url' => ['faq-category/index']] ,
```

Hemos hecho un simple cambio a la etiqueta.

Luego de realizar el cambio navegue de regreso a backend.yii2build.com y seleccione Categorías de FAQ de la lista desplegable.

Todas las páginas deberían funcionar, aunque sólo hemos modificado _form.php. Así que desde index.php, que es adonde lo llevó la navegación, haga clic en el botón create y debería ver nuestro formulario modificado. Agregue un par de registros de prueba, puede usar General y Específica como categorías de prueba.

Todas las características deberían funcionar, incluyendo aquellas por defecto. Igualmente, si intenta ingresar un número mayor a 100 en el campo de formulario Faq Category Weight, verá que obtiene un mensaje de error, de esta forma sabemos que nuestro validador de rango está funcionando. Genial.

Bien, así que una vez que hemos verificado que todo está funcionando de manera correcta, continuemos con el resto de los cambios a las vistas.

En _search, tenemos una modificación de una línea. Necesitamos cambiar el text input de faq_category_is_featured con:

```
<?= $form->field($model, 'faq_category_is_featured')->
dropDownList($model->faqCategoryIsFeaturedList,
[ 'prompt' => 'Por favor seleccione uno' ])?>
```

Note el segundo uso del método faqCategoryIsFeaturedList, así que la teoría no demoró demasiado en ser probada.

Obviamente el código está dividido en 3 líneas para evitar el ajuste de línea en el PDF. Usted debería tener una sola línea en su archivo.

Bien, continuemos. No hay cambios en este punto a create y update, así que podemos encargarnos de view.php. Simplemente unos cambios menores:

Gist:

[FaqCategory View.php](#)

Del libro:

```
<?php

use yii\helpers\Html;
use yii\widgets\DetailView;

/* @var $this yii\web\View */
/* @var $model backend\models\FaqCategory */

$this->title = $model->faq_category_name;
$this->params['breadcrumbs'][] =
['label' => 'Faq Categories', 'url' => ['index']];
$this->params['breadcrumbs'][] = $this->title;
?>
<div class="faq-category-view">

    <h1>Categoría de Faq: <?= Html::encode($this->title) ?></h1>

    <p>
        <?= Html::a('Update', ['update', 'id' => $model->id],
        ['class' => 'btn btn-primary']) ?>
```

```
<?= Html::a('Delete', ['delete', 'id' => $model->id], [
    'class' => 'btn btn-danger',
    'data' => [
        'confirm' => '¿Seguro de borrar este ítem?',
        'method' => 'post',
    ],
]) ?>
</p>

<?= DetailView::widget([
    'model' => $model,
    'attributes' => [
        'id',
        'faq_category_name',
        'faq_category_weight',
        ['attribute'=>'faq_category_is_featured',
        'format'=>'boolean'],
    ],
]) ?>

</div>
```

Tenemos 3 pequeños cambios aquí. Hemos cambiado el título a \$model->faq_category_name en lugar de la id, y agregamos las palabras “Categoría de Faq:” a nuestro h1.

El último cambio fue a un atributo en el widget DetailView:

```
[ 'attribute'=>'faq_category_is_featured', 'format'=>'boolean' ],
```

Yii 2 permite establecer el formato del atributo, así que en este caso lo seteamos a booleano, por lo que ahora tenemos Si y No en lugar de 0 o 1 en la salida de faq_category_is_featured.

Si todo ha ido bien, su página view debería verse de esta manera:

ID	2
Faq Category Name	Específica
Faq Category Weight	20
Faq Category Is Featured	No

Bien, continuemos ahora con index.php.

Gist:

[FaqCategory Index](#)

Del libro:

```
<?php

use yii\helpers\Html;
use yii\grid\GridView;
use \yii\bootstrap\Collapse;

/* @var $this yii\web\View */
/* @var $searchModel backend\models\search\FaqCategorySearch */
/* @var $dataProvider yii\data\ActiveDataProvider */

$this->title = 'Categorías de Faq';
$this->params['breadcrumbs'][] = $this->title;
?>
<div class="faq-category-index">

    <h1><?= Html::encode($this->title) ?></h1>

    <?php echo Collapse::widget([
        'items' => [
            // equivalent to the above
            [
                'label' => 'Search',
                'content' => $this->render('_search',

```

```

['model' => $searchModel] ,
    // open its content by default
    // 'contentOptions' => ['class' => 'in']
],
]

]);
?>

<p>
<?= Html::a('Crear Categorías de Faq', ['create'],
['class' => 'btn btn-success']) ?>
</p>

<?= GridView::widget([
    'dataProvider' => $dataProvider,
    'filterModel' => $searchModel,
    'columns' => [
        ['class' => 'yii\grid\SerialColumn'],

        'id',
        'faq_category_name',
        'faq_category_weight',
        ['attribute'=>'faq_category_is_featured',
        'format'=>'boolean'],

        ['class' => 'yii\grid\ActionColumn'],
    ],
]); ?>

</div>

```

Tres cambios sencillos. Agregamos el widget Collapse con una sentencia use, y llamamos a _search dentro del mismo, como lo hicimos en los otros modelos. Y luego formateamos el valor booleano para nuestro atributo faq_category_is_featured, como lo hicimos en la página view. Debería verse de esta manera:

Showing 1-2 of 2 items.

#	ID	Faq Category Name	Faq Category Weight	Faq Category Is Featured	
1	1	General	10	No	
2	2	Específica	20	No	

FaqCategory Index

Bien, genial, eso es todo para FaqCategory, todo terminado, limpio y ordenado.

Ahora estamos listos para tratar con el propio modelo FAQ. Vamos a comenzar agregando el método behaviors, que también contendrá el behavior blameable.

Gist:

[Faq behaviors](#)

Del libro:

```
public function behaviors()
{
    return [
        'timestamp' => [
            'class' => 'yii\behaviors\TimestampBehavior',
            'attributes' => [
                ActiveRecord::EVENT_BEFORE_INSERT => ['created_at', 'updated_at'],
                ActiveRecord::EVENT_BEFORE_UPDATE => ['updated_at'],
            ],
            'value' => new Expression('NOW()'),
        ],
        'blameable' => [
            'class' => BlameableBehavior::className(),
            'createdByAttribute' => 'created_by',
            'updatedByAttribute' => 'updated_by',
        ],
    ];
}
```

Necesitamos agregar las siguientes sentencias use al comienzo del archivo:

```
use yii\db\ActiveRecord;
use yii\db\Expression;
use yii\behaviors\BlameableBehavior;
```

Puede ver como hemos seteado ‘blameable’ en arreglo de behaviors y esta sintaxis es muy clara. Todo lo que tenemos que hacer es mapear ‘createdByAttribute’ a ‘created_by’, que es el nombre de columna que usamos en nuestra estructura de datos, y hacer lo mismo para updatedByAttribute, y estamos listos.

A continuación tenemos que añadir un par de cambios a nuestras reglas.

Gist:

[Faq rules](#)

Del libro:

```
/***
 * @inheritdoc
 */

public function rules()
{
    return [
        [['faq_question', 'faq_answer'], 'required'],
        [['faq_category_id', 'faq_is_featured', 'faq_weight', 'created_by',
        'updated_by'], 'integer'],
        [['faq_weight'], 'in', 'range'=>range(1,100)],
        ['faq_weight', 'default', 'value' => 100],
        [['created_at', 'updated_at'], 'safe'],
        [['faq_question'], 'string', 'max' => 255],
        [['faq_question'], 'unique'],
        [['faq_answer'], 'string', 'max' => 1055]
    ];
}
```

Note que además de la regla de rango y el valor por defecto para faq_weight, tenemos una regla unique para faq_question. Esto funciona de maravillas y evita repetir la misma pregunta en Faq.

Luego, continuemos con nuestra relación y métodos asociados. Verdaderamente tenemos bastantes para un modelo tan sencillo, pero todos serán de mucha utilidad.

Gist:

[Relaciones de Faq](#)

Del libro:

```
/**  
 * uses magic getFaqCategoryName on return statement  
 */  
  
public function getFaqCategoryName()  
{  
    return $this->faqCategory->faq_category_name;  
}  
  
/**  
 * get list of FaqCategory for dropdown  
 */  
public static function getFaqCategoryList()  
{  
  
    $droptions = FaqCategory::find()->asArray()->all();  
    return Arrayhelper::map($droptions, 'id', 'faq_category_name');  
}  
  
public static function getFaqIsFeaturedList()  
{  
    return $droptions = [0 => "no", 1 => "yes"];  
}  
  
public function getFaqIsFeaturedName()  
{  
    return $this->faq_is_featured == 0 ? "no" : "yes";  
}  
  
public function getCreatedByUser()  
{  
    return $this->hasOne(User::className(), ['id' => 'created_by']);  
}  
/**  
 * @getCreateUserName  
 *  
 */  
public function getCreatedByUsername()
```

```

{
    return $this->createdByUser ?
        $this->createdByUser->username : '- no user -';
}

public function getUpdatedByUser()
{
    return $this->hasOne(User::className(), ['id' => 'updated_by']);
}

/**
 * @getUpdateUserName
 *
 */
public function getUpdatedByUsername()
{
    return $this->updatedByUser ?
        $this->updatedByUser->username : '- no user -';
}

```

Ya que hemos cubierto la mayoría de esto previamente, sólo voy a hablar de los últimos 4. Cuando mostramos created by y update en nuestras vistas, necesitamos enlazarnos a nuestra tabla user ya sea por el método getCreatedByUser o por el método getUpdatedByUser.

Luego usamos la relación para ser capaces de devolver el nombre de usuario en el siguiente método, getCreatedByUsername, que es el que usamos en nuestras vistas.

Y por supuesto para tener acceso a los modelos User y FaqCategory, así como a las clases auxiliares, tenemos que agregar las sentencias use.

Gist:

Sentencias Use

Del libro:

```

use backend\models\FaqCategory;
use yii\helpers\ArrayHelper;
use yii\helpers\Url;
use yii\helpers\Html;
use common\models\User;

```

Por último, necesitamos reemplazar nuestro método attributeLabels por el siguiente.

Gist:

Faq Labels

Del libro:

```

public function attributeLabels()
{
    return [
        'id' => 'ID',
        'faq_question' => 'Pregunta',
        'faq_answer' => 'Respuesta',
        'faq_category_id' => 'Categoría',
        'faq_weight' => 'Peso',
        'faq_is_featured' => '¿Destacada?',
        'created_by' => 'Creada Por',
        'updated_by' => 'Actualizada Por',
        'created_at' => 'Creada El',
        'updated_at' => 'Actualizada El',
        'faqCategoryName' => Yii::t('app', 'Categoría'),
        'faqCategoryList' => Yii::t('app', 'Categoría'),
        'faqIsFeaturedName' => Yii::t('app', 'Destacada'),
        'createdByUserName' => Yii::t('app', 'Creada Por'),
        'updatedByUserName' => Yii::t('app', 'Actualizada Por'),
    ];
}

```

Simplemente como referencia, para que tenga el modelo Faq completo, voy a incluir el código para el modelo en su totalidad.

Gist:

Modelo Faq

Del libro:

```

<?php

namespace backend\models;

use Yii;
use yii\db\ActiveRecord;
use yii\db\Expression;
use yii\behaviors\BlameableBehavior;
use backend\models\FaqCategory;
use yii\helpers\ArrayHelper;
use yii\helpers\Url;
use yii\helpers\Html;
use common\models\User;

```

```
/**  
 * This is the model class for table "faq".  
 *  
 * @property integer $id  
 * @property string $faq_question  
 * @property string $faq_answer  
 * @property integer $faq_category_id  
 * @property integer $faq_is_featured  
 * @property integer $faq_weight  
 * @property integer $created_by  
 * @property integer $updated_by  
 * @property string $created_at  
 * @property string $updated_at  
 *  
 * @property FaqCategory $faqCategory  
 */  
  
class Faq extends \yii\db\ActiveRecord  
{  
    /**  
     * @inheritdoc  
     */  
    public static function tableName()  
    {  
        return 'faq';  
    }  
  
    public function behaviors()  
    {  
        return [  
            'timestamp' => [  
                'class' => 'yii\behaviors\TimestampBehavior',  
                'attributes' => [  
                    ActiveRecord::EVENT_BEFORE_INSERT => ['created_at', 'updated_at'],  
                    ActiveRecord::EVENT_BEFORE_UPDATE => ['updated_at'],  
                    ],  
                    'value' => new Expression('NOW()'),  
                ],  
                'blameable' => [  
                    'class' => BlameableBehavior::className(),  
                    'createdByAttribute' => 'created_by',  
                    'updatedByAttribute' => 'updated_by',  
                ]  
            ]  
    }  
}
```

```
        ] ,  
    ];  
}  
  
/**  
 * @inheritdoc  
 */  
  
public function rules()  
{  
    return [  
        [['faq_question', 'faq_answer'], 'required'],  
        [['faq_category_id', 'faq_is_featured', 'faq_weight',  
        'created_by', 'updated_by'], 'integer'],  
        [['faq_weight'], 'in', 'range'=>range(1,100)],  
        ['faq_weight', 'default', 'value' => 100],  
        [['created_at', 'updated_at'], 'safe'],  
        [['faq_question']], 'string', 'max' => 255],  
        [['faq_question']], 'unique'],  
        [['faq_answer']], 'string', 'max' => 1055]  
    ];  
}  
  
/**  
 * @inheritdoc  
 */  
  
public function attributeLabels()  
{  
    return [  
        'id' => 'ID',  
        'faq_question' => 'Pregunta',  
        'faq_answer' => 'Respuesta',  
        'faq_category_id' => 'Categoría',  
        'faq_weight' => 'Peso',  
        'faq_is_featured' => '¿Destacada?',  
        'created_by' => 'Creado Por',  
        'updated_by' => 'Actualizado El',  
        'created_at' => 'Creado El',  
        'updated_at' => 'Updated At',  
        'faqCategoryName' => Yii::t('app', 'Categoría'),  
        'faqCategoryList' => Yii::t('app', 'Categoría'),  
        'faqIsFeaturedName' => Yii::t('app', 'Destacada'),
```

```
    'createdByUserName' => Yii::t('app', 'Creado Por'),
    'updatedByUserName' => Yii::t('app', 'Actualizado Por'),
];
}

/**
 * @return \yii\db\ActiveQuery
 */
public function getFaqCategory()
{
    return $this->hasOne(FaqCategory::className(),
['id' => 'faq_category_id']);
}

/**
 * uses magic getFaqCategoryName on return statement
 *
 */
public function getFaqCategoryName()
{
    return $this->faqCategory->faq_category_name;
}

/**
 * get list of FaqCategory for dropdown
 */
public static function getFaqCategoryList()
{
    $droptions = FaqCategory::find()->asArray()->all();
    return Arrayhelper::map($droptions, 'id', 'faq_category_name');

}

public static function getFaqIsFeaturedList()
{
    return $droptions = [0 => "no", 1 => "yes"];
}

public function getFaqIsFeaturedName()
```

```

{
    return $this->faq_is_featured == 0 ? "no" : "yes";
}

public function getCreatedByUser()
{
    return $this->hasOne(User::className(),
['id' => 'created_by']);
}
/**
 * @getCreateUserName
 *
 */
public function getCreatedByUsername()
{
    return $this->createdByUser ?
$this->createdByUser->username : '- no user -';
}

public function getUpdatedByUser()
{
    return $this->hasOne(User::className(),
['id' => 'updated_by']);
}
/**
 * @getUpdateUserName
 *
 */
public function getUpdatedByUsername()
{
    return $this->updatedByUser ?
$this->updatedByUser->username : '- no user -';
}
}

```

Y eso finaliza el modelo Faq. Ahora observemos el modelo Faq Search. No voy a guiarlo a través de los cambios porque esto ya fue cubierto en detalle en el capítulo 11, sobre cómo modificar la búsqueda nativa para usar relaciones con eager loading.

Aun cuando la tabla FaqCategories es probablemente pequeña, no hace daño usar eager loading por eficiencia.

Gist:

Faq Search

Del libro:

```
<?php

namespace backend\models\search;

use Yii;
use yii\base\Model;
use yii\data\ActiveDataProvider;
use yii\data\ArrayDataProvider;
use yii\db\ActiveQuery;
use yii\db\Query;
use backend\models\Faq;

/**
 * FaqSearch represents the model behind the search form about `backend\models\FAQ`.
 */
class FaqSearch extends Faq
{
    public $faqCategoryName;
    public $faqCategoryList;
    public $faqIsFeaturedName;
    public $createdByUsername;
    public $updatedByUsername;
    public $faq_category;
    public $faq_weight;

    /**
     * @inheritdoc
     */
    public function rules()
    {
        return [
            [['id', 'faq_category_id', 'faq_weight', 'faq_is_featured',
            'created_by', 'updated_by'], 'integer'],
            [['faq_question', 'faq_answer', 'created_at', 'updated_at',
            'faqCategoryName', 'faqCategoryList', 'faqIsFeaturedName',
            'createdByUsername', 'updatedByUsername', 'faq_category'],
            
```

```
'faq_weight'], 'safe'],

];

}

/***
 * @inheritDoc
 */
public function scenarios()
{
    // bypass scenarios() implementation in the parent class
    return Model::scenarios();
}

/***
 * Creates data provider instance with search query applied
 *
 * @param array $params
 *
 * @return ActiveDataProvider
 */
public function search($params)
{
    $query = Faq::find();
    $dataProvider = new ActiveDataProvider([
        'query' => $query,
    ]);

    /**
     * Setup your sorting attributes
     * Note: This is setup before the $this->load($params)
     * statement below
     */
    $dataProvider->setSort([
        'defaultOrder' => [
            'faq_weight' => SORT_ASC,
        ],
        'attributes' => [
    ]);
}
```

```
'id',
'faq_question' => [
    'asc' => ['faq.faq_question' => SORT_ASC],
    'desc' => ['faq.faq_question' => SORT_DESC],
    'label' => 'Question'
],
'faq_answer' => [
    'asc' => ['faq.faq_answer' => SORT_ASC],
    'desc' => ['faq.faq_answer' => SORT_DESC],
    'label' => 'Answer'
],
'faqCategoryName' => [
    'asc' => ['faq_category.faq_category_name' => SORT_ASC],
    'desc' => ['faq_category.faq_category_name' => SORT_DESC],
    'label' => 'Category'
],
'faq_weight' => [
    'asc' => ['faq.faq_weight' => SORT_ASC],
    'desc' => ['faq.faq_weight' => SORT_DESC],
    'label' => 'Weight'
],
'faqIsFeaturedName' => [
    'asc' => ['faq.faq_is_featured' => SORT_ASC],
    'desc' => ['faq.faq_is_featured' => SORT_DESC],
    'label' => 'Featured?'
],
],
]);
);

if (!$this->load($params) && $this->validate())) {
    $query->joinWith(['faqCategory']);
    return $dataProvider;
}

$this->addSearchParameter($query, 'id');
```

```
$this->addSearchParameter($query, 'faq_category_id');
$this->addSearchParameter($query, 'faq_weight');
$this->addSearchParameter($query, 'faq_is_featured');
$this->addSearchParameter($query, 'created_by');
$this->addSearchParameter($query, 'updated_by');
$this->addSearchParameter($query, 'faq_question', true);
$this->addSearchParameter($query, 'faq_answer', true);

        // filter by category
$query->joinWith(['faqCategory' => function ($q) {
    $q->andFilterWhere(['=' , 'faq_category.faq_category_name',
$this->faqCategoryName]);
}]);

return $dataProvider;
}

protected function addSearchParameter($query, $attribute, $partialMatch = false)
{
    if (($pos = strpos($attribute, '.')) !== false) {
        $modelAttribute = substr($attribute, $pos + 1);
    } else {
        $modelAttribute = $attribute;
    }

    $value = $this->$modelAttribute;

    if (trim($value) === '') {
        return;
    }

    /*
    * The following line is additionally added for right aliasing
    * of columns so filtering happen correctly in the self join
    */
    $attribute = "faq.$attribute";

    if ($partialMatch) {
```

```
        $query->andWhere(['like', $attribute, $value]);
    } else {
        $query->andWhere([$attribute => $value]);
    }
}
}
```

Puede notar que hemos usado sentencias use en este modelo que no hemos empleado antes:

```
use yii\data\ArrayDataProvider;
use yii\db\ActiveQuery;
use yii\db\Query;
```

Las necesitaremos más adelante cuando modifiquemos esta clase, así que las he incluido ahora.

La otra cosa que es diferente es que hemos especificado un orden por defecto:

```
'defaultOrder' => [
    'faq_weight' => SORT_ASC,
],
```

Esto establecerá el orden por defecto de los resultados de manera ascendente según faq_weight, así para controlar el orden en que las vemos, todo lo que tenemos que hacer es asignar faq_weight en consecuencia, usando la IU. Por favor note que este método es solamente para el backend. Como hemos prometido, lo haremos de manera diferente en el frontend, lo que implicará un método distinto. Explicará más cuando llegue a ese punto.

Bien, realizamos nuestros tipos cambios en el método behaviors del controlador:

Gist:

Faq Behaviors

Del libro:

```

public function behaviors()
{
    return [
        'access' => [
            'class' => \yii\filters\AccessControl::className(),
            'only' => ['index', 'view', 'create', 'update', 'delete'],
            'rules' => [
                [
                    'actions' => ['index', 'view', 'create', 'update', 'delete'],
                    'allow' => true,
                    'roles' => ['@'],
                    'matchCallback' => function ($rule, $action) {
                        return PermisosHelpers::requirirMinimoRol('Admin')
                            && PermisosHelpers::requireEstado('Activo');
                    }
                ],
            ],
        ],
    ],
}

'verbs' => [
    'class' => VerbFilter::className(),
    'actions' => [
        'delete' => ['post'],
    ],
];
}

```

No olvide agregar la sentencia use:

```
use common\models\PermisosHelpers;
```

Ahora cambiemos las vistas, comenzando por _form.php.

Gist:

[Faq Form](#)

Del libro:

```
<?php

use yii\helpers\Html;
use yii\widgets\ActiveForm;

/* @var $this yii\web\View */
/* @var $model backend\models\Faq */
/* @var $form yii\widgets\ActiveForm */
?>

<div class="faq-form">

    <?php $form = ActiveForm::begin(); ?>

    <?= $form->field($model, 'faq_question') ->
textInput(['maxlength' => 255]) ?>

    <?= $form->field($model, 'faq_answer') ->
textArea(['maxlength' => 1055, 'rows' => 10]) ?>

    <?= $form->field($model, 'faq_category_id') ->
dropDownList($model->faqCategoryList,
[ 'prompt' => 'Por favor elija una' ]); ?>

    <?= $form->field($model, 'faq_is_featured') ->
dropDownList($model->faqIsFeaturedList,
[ 'prompt' => 'Por favor elija uno' ]); ?>

    <?= $form->field($model, 'faq_weight') ->textInput() ?>

<div class="form-group">
    <?= Html::submitButton($model->isNewRecord ? 'Create' : 'Update',
[ 'class' => $model->isNewRecord ? 'btn btn-success' : 'btn btn-primary' ]) ?>
</div>

<?php ActiveForm::end(); ?>

</div>
```

Algo nuevo que hicimos aquí fue cambiar `faq_answer` a una `textArea`. Con el método `textArea`, puede especificar el número de filas, que hemos establecido a 10.

Continuemos con la vista _search.

Gist:

Faq Search Partial

Del libro:

```
<?php

use yii\helpers\Html;
use yii\widgets\ActiveForm;

/* @var $this yii\web\View */
/* @var $model backend\models\search\FaqSearch */
/* @var $form yii\widgets\ActiveForm */
?>

<div class="faq-search">

    <?php $form = ActiveForm::begin([
        'action' => ['index'],
        'method' => 'get',
    ]); ?>

    <?= $form->field($model, 'id') ?>

    <?= $form->field($model, 'faq_question') ?>

    <?= $form->field($model, 'faq_answer') ?>

        <?= $form->field($model, 'faq_category_id')->
dropDownList($model->getFaqCategoryList(),
[ 'prompt' => 'Por favor elija una' ]);?>

        <?= $form->field($model, 'faq_is_featured')->
dropDownList($model->faqIsFeaturedList,
[ 'prompt' => 'Por favor elija uno' ]);?>

    <?= $form->field($model, 'faq_weight') ?>

<div class="form-group">
    <?= Html::submitButton('Search',
```

```
[ 'class' => 'btn btn-primary']) ?>
    <?= Html::resetButton('Reset',
['class' => 'btn btn-default']) ?>
</div>

<?php ActiveForm::end(); ?>

</div>
```

Ahora continuemos con view.php.

Gist:

[Faq View](#)

Del libro:

```
<?php

use yii\helpers\Html;
use yii\widgets\DetailView;

/* @var $this yii\web\View */
/* @var $model backend\models\Faq */

$this->title = 'FAQ: '. $model->faq_question;
$this->params[ 'breadcrumbs' ][] =
[ 'label' => 'Faqs', 'url' => [ 'index' ]];
$this->params[ 'breadcrumbs' ][] = $this->title;
?>
<div class="faq-view">

    <h1><?= Html::encode($this->title) ?></h1>

    <p>
        <?= Html::a('Update', [ 'update', 'id' => $model->id], [
['class' => 'btn btn-primary']) ?>
        <?= Html::a('Delete', [ 'delete', 'id' => $model->id], [
            'class' => 'btn btn-danger',
            'data' => [
                'confirm' => '¿Seguro de borrar este ítem?',
                'method' => 'post',
            ],
        ]) ?>
    </p>
```

```

</p>

<?= DetailView::widget([
    'model' => $model,
    'attributes' => [
        'id',
        'faq_question',
        'faq_answer',
        'faqCategory.faq_category_name',
        'faq_weight',
        ['attribute'=>'faq_is_featured', 'format'=>'boolean'],
        ['attribute'=>'createdByUsername', 'format'=>'raw'],
        ['attribute'=>'updatedByUsername', 'format'=>'raw'],
        'created_at',
        'updated_at',
    ],
]) ?>

</div>

```

Puede ver que hemos cambiado el título y hecho los cambios de atributo en el widget DetailView para poder tener el formato apropiado cuando vemos el registro.

Ahora trabajemos con el archivo index.php file de Faq.

Gist:

Faq Index

Del libro:

```

<?php

use yii\helpers\Html;
use yii\grid\GridView;
use \yii\bootstrap\Collapse;

/* @var $this yii\web\View */
/* @var $searchModel backend\models\search\FaqSearch */
/* @var $dataProvider yii\data\ActiveDataProvider */

$this->title = 'Faqs';
$this->params['breadcrumbs'][] = $this->title;
?>
<div class="faq-index">

```

```
<h1><?= Html::encode($this->title) ?></h1>
<?php echo Collapse::widget([
'items' => [
    // equivalent to the above
    [
        'label' => 'Search',
        'content' => $this->render('_search', ['model' => $searchModel]) ,
            // open its content by default
            //'contentOptions' => ['class' => 'in']
    ],
],
]); ?>

<p>
    <?= Html::a('Create Faq', ['create'],
['class' => 'btn btn-success']) ?>
</p>

<?= GridView::widget([
    'dataProvider' => $dataProvider,
    'filterModel' => $searchModel,
    'columns' => [
        ['class' => 'yii\grid\SerialColumn'],

        'id',
        'faq_question',
        'faq_answer',
        ['attribute'=>'faqCategoryName', 'format'=>'raw'],
        'faq_weight',
        ['attribute'=>'faqIsFeaturedName', 'format'=>'raw'],

        ['class' => 'yii\grid\ActionColumn'],
    ],
]); ?>

</div>
```

Debería verse familiar. Una diferencia sin embargo es que usamos el método getFaqIsFeaturedName (a través de una llamada mágica) para mostrar si o no para el valor booleano. Podríamos haber usado:

```
[ 'attribute'=> 'faq_is_featured', 'format'=>'boolean' ],
```

Pero pensé que sería mejor usar la llamada mágica porque usaremos ese método en nuestro método de búsqueda y es mejor ser consistente, ya que nuestro modelo devuelve los resultados en index.

Si aún no lo ha hecho, cree al menos seis registros de Faq, para que pueda ver como luce todo.

Así que ahora vamos a continuar con la creación de la versión del frontend de las Faqs. Aunque podríamos usar el mismo dataprovider y widgets que usamos en el backend, quiero liberarnos de ello y al mismo tiempo, demostrar el uso de la clase ArrayDataProvider, que devuelve los resultados en un arreglo, donde podemos establecer con facilidad cosas como el orden y que atributos devolver.

Comenzaremos agregando un método a nuestro modelo FaqSearch llamado frontendProvider. Lo llamó de ese modo de manera que sea posible determinar su uso a partir del nombre.

Como dije, el propósito de este método es proveer los datos para una lista de FAQs ordenada por faq_weight. Esto permitirá al usuario final controlar en qué lugar de la lista aparece una pregunta, simplemente al aumentar o disminuir el peso.

Así que continúe y agregue el siguiente método a FaqSearch.php.

Gist:

Frontend Provider

Del libro:

```
public function frontendProvider()
{
    $query = new Query;
    $provider = new ArrayDataProvider([
        'allModels' => $query->from('faq')->all(),
        'sort' => [
            'defaultOrder' => [
                'faq_weight' => SORT_ASC,
            ],
            'attributes' => ['faq_question', 'faq_answer',
                'faq_weight'],
        ],
        'pagination' => [
            'pageSize' => 10,
        ],
    ]);

    return $provider;
}
```

```
}
```

Bien, revisemoslo. Comenzamos creando una instancia de Query, que nos permitirá crear una consulta dentro de una instancia de ArrayDataProvider. Por favor tenga en cuenta que necesitamos agregar los siguiente para que funcione:

```
use yii\data\ArrayDataProvider;
use yii\db\ActiveQuery;
use yii\db\Query;
```

Así que asegúrese de que las sentencias use están en el lugar apropiado al comienzo del archivo. Yii 2 hace un excelente trabajo al protestar, así que si lo olvida, se lo hará saber.

Bien, de regreso al código. Usé como referencia este ejemplo:

<http://www.yiiframework.com/doc-2.0/guide-output-data-providers.html>

Así que si revisa la guía verá que tiene exactamente ese formato.

El ArrayDataProvider está configurado con el tipo de arreglo de configuración que hemos visto en muchos lugares en Yii 2. En este caso, tenemos una clave 'allModels', que apunta a una consulta como valor, que devuelve todos los registros de faq. Y ya que estamos usando ArrayDataProvider, sabemos que devolveremos un array.

La guía no muestra como establecer un orden por defecto, pero lo descubrí. Establecemos el orden por defecto a faq_value, SORT_ASC, que significa que ordenará de manera ascendente. Esto significa que una FAQ con un valor de 1 estará primera en la lista.

Luego le indicamos que atributos queremos en el arreglo. En este caso, ya que no estamos usando esto para la creación y mantenimiento de registros en el backend, podemos incluir menos atributos. En ese caso, sólo necesitamos:

```
'attributes' => [ 'faq_question', 'faq_answer', 'faq_weight' ],
```

También estamos estableciendo el tamaño de la paginación a 10. Probablemente no lo necesitará, pero si lo hace, ahí está. Si tiene más de 10 registros, también necesita establecer paginación, pero lo haremos luego en una próxima iteración, ahora no es necesario. Así que es algo muy simple.

Luego necesitamos crear un controlador en el frontend para Faq, que usa el modelo Faq del backend y el modelo del backend FaqSearch. Solamente vamos a tener 2 acciones view e index. No hay necesidad para las acciones create, update y delete porque son administradas en el backend por el admin, no el frontend por los usuarios. Debido a que estan son FAQs, queremos que sean visibles para todos los usuarios, así que no restringiremos el acceso, así que esto hace que el controlador sea muy simple.

Aquí está el controlador completo.

Gist:

Faq Frontend Controller

Del libro:

```
<?php

namespace frontend\controllers;

use Yii;
use backend\models\Faq;
use backend\models\search\FaqSearch;
use yii\web\Controller;
use yii\web\NotFoundHttpException;
use yii\filters\VerbFilter;

/**
 * FaqController implements the CRUD actions for Faq model.
 */

class FaqController extends Controller
{
    public function behaviors()
    {
        return [
            'verbs' => [
                'class' => VerbFilter::className(),
                'actions' => [
                    'delete' => ['post'],
                ],
            ],
        ];
    }

    /**
     * Lists all Faq models.
     * @return mixed
     */
}

public function actionIndex()
{
```

```
$searchModel = new FaqSearch();

$provider = $searchModel->frontendProvider();

return $this->render('index', [
    'searchModel' => $searchModel,
    'provider' => $provider,
]);

}

/**
 * Displays a single Faq model.
 * @param integer $id
 * @return mixed
 */

public function actionView($id)
{
    return $this->render('view', [
        'model' => $this->findModel($id),
    ]);
}

protected function findModel($id)
{
    if (($model = Faq::findOne($id)) !== null) {
        return $model;
    } else {
        throw new NotFoundHttpException
('La página solicitada no existe.');
    }
}
```

Es realmente muy simple. En el método index, convocamos a una nueva instancia del modelo FaqSearch, que nos permitirá asignar una instancia de \$searchModel->frontendProvider() a \$provider, que podemos enviar a la vista. Así que ahora la vista tendrá acceso al ArrayDataProvider que construimos, con faq_weight ordenado de manera ascendente.

En la vista, para mostrar esto, necesitamos lo siguiente en frontend/views/faq/index.php. No olvide crear el directorio faq folder en frontend/views.

Gist:

Frontend Faq Index

Del libro:

```
<?php
use yii\helpers\Html;
use yii\helpers\ArrayHelper;
use yii\helpers\Url;

$this->title = 'FAQs';
$this->params['breadcrumbs'][] = $this->title;

?>

<div class="site-about">
    <h1><?= Html::encode($this->title) ?></h1>
    </BR>
<div class="panel panel-default">
    <div class="panel-heading">
        <h3 class="panel-title">
            Preguntas
        </h3>
    </div>

    <?php
    $data = $provider->getModels();
    $questions = ArrayHelper::map($data, 'faq_question', 'id');
    foreach ($questions as $question => $id){

        $url = Url::to(['faq/view', 'id'=>$id]);
        $options = [];
        echo '<div class="panel-body">' .
Html::a($question, $url, $options) . '</div>';
    }
?>
```

```
</div>
</div>
```

Así que ahora estamos asignando \$provider->getModels() a \$data. Si mira en la guía, este es el método que usan en su ejemplo, así que es exactamente el que usé. Luego cree pares clave valor a partir de la Id de pregunta usando el método ArrayHelper::map. Si intenta esto con registros que tienen diferentes faq_weight, verá que respeta el orden de faq_weight. Así que puede entregar a su cliente el control al proveerle de una IU en el backend que le permita establecer faq_weight.

Luego uso un loop foreach para hacer algo con cada pregunta y su correspondiente id. En este caso, ese algo es primero crear una url que tenga esa \$id como parámetro. Luego realizo un eco de esa línea, usando el método Html::a method para crear un link que usa \$question como texto y \$url como la url a la vista.

Utilicé un poco de bootstrap para estilizar la página sólo como diversión. Puede jugar con bootstrap en:

[Layoutit.com](#)

Tiene como característica el diseño mediante drag and drop, luego puede copiar su código, así que es perfecto para diseñar pequeños snippets como el de arriba. La pagina index resultante debería verse así:

The screenshot shows a web page with a header bar containing 'Home / FAQs'. Below the header is a section titled 'Preguntas' (Questions) which contains the following list of questions:

- ¿Puedo aprenderlo?
- ¿Cuánto tiempo lleva?
- ¿Puedo hacer dinero programando?
- ¿Puede ayudarme este sitio?

Volviendo a la lista de faq, los links que creamos en nuestro loop foreach enviarán una solicitud a la acción view en el controlador faq del frontend:

```
public function actionView($id)
{
    return $this->render('view', [
        'model' => $this->findModel($id),
    ]);
}
```

Toma la \$id de la variable get, que fue asignada en la url, y llama a una instancia de la vista, usando la instancia correcta del modelo al emplear el último método del controlador:

```
protected function findModel($id)
{
    if (($model = Faq::findOne($id)) !== null) {
        return $model;
    } else {
        throw new NotFoundHttpException
            ('La página solicitada no existe.');
    }
}
```

Puede ver lo limpio y sencillo que es todo.

Para que esto funcione, en frontend/views/faq/view.php, necesitamos lo siguiente.

Gist:

Vista Faq del Frontend

Del libro:

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

$this->title = 'FAQ: ' . $model->faq_question;

$this->params['breadcrumbs'][] = ['label' => 'FAQ', 'url' => ['index']];
$this->params['breadcrumbs'][] = $this->title;
?>

</br>
<div class="panel panel-default">
    <div class="panel-heading">
        <h3 class="panel-title">

            <h1>    <?= $model->faq_question; ?> </h1>

        </h3>
    </div>

    <?= '<div class="panel-body"><h3>' .
        $model->faq_answer . '</h3></div>'; ?>

</div>
```

Las partes relevantes aquí son `$model->faq_question` y `$model->faq_answer`, que están disponibles para nosotros porque usamos `findModel` para encontrar la instancia del modelo Faq y enviarla a la vista desde el controlador.

Una última tarea, simplemente agreguémolas en nuestra navegación en `frontend/views/layouts/-main.php`. Añadamos un nuevo link al primer arreglo `$menuItems`:

```
$menuItems = [
    ['label' => 'Home', 'url' => ['/site/index']],
    ['label' => 'Sobre', 'url' => ['/site/about']],
    ['label' => 'FAQs', 'url' => ['/faq/index']],
    ['label' => 'Contacto', 'url' => ['/site/contact']],
];
```

Es un cambio de una línea, así que no es necesario un Gist. Una pregunta surgió de un lector sobre incluir o no la / antes del controlador. En el ejemplo que Yii 2 muestra en la guía, no hay una barra. El código que viene de manera predeterminada con la plantilla, la incluye. Ambas formas funcionan. Así que depende de usted que convención seguir.

Y eso es todo. Estoy seguro de que puede hacerlo más bonito, ya que yo no soy un diseñador de IU, pero espero que entienda el punto, que es cómo mover datos de las FAQ y mantener control sobre el orden, lo que sus clientes amarán.

Intencionalmente evité usar los widgets Listview y GridView en este ejemplo para mostrarle como puede usar cosas tales como el `ArrayDataProvider` sin ellos. Puede personalizar su IU tanto como lo desee, no está limitado a los widgets.

Espero que cuando tenga una reunión con sus clientes y usted les sugiera cómo puede otorgarles más control sobre el orden de sus FAQs, estarán impresionados, que es lo que deseamos. Queremos complacer a aquellos clientes realmente difíciles porque así es como nos pagan.

Aunque no usamos `FaqCategory` en el lado de la presentación, podríamos hacerlo fácilmente realizando modificaciones similares al modelo `FaqCategorySearch` e implementando a partir de allí. Ese es el beneficio de poner algo de previsión en la estructura de datos. Así que una vez más, si los clientes lo piden, ustedes se encuentran un paso adelante.

Test Controller

Algunas veces durante el desarrollo, usted querrá aislar un bloque de código y jugar con él, sin desordenar el resto de su código. Ahora bien, el control de versiones siempre debería estar presente si comete un grave error, pero realmente no quiere depender de la posibilidad de retroceder a menos que sea absolutamente necesario.

Así que lo que haremos es crear un controlador llamado `test`. Comenzaremos navegando a `Gii`, y haciendo clic sobre `Controller Generator`. No tenemos que preocuparnos por crear un modelo

primero, ya que no tenemos uno, nuestro controlador será usado para jugar con partes diferentes de distintos modelos, e introduciremos los modelos mediante sentencias use a medida que los necesitemos.

Desde nuestro generador de controlador en Gii, introduzcamos test en el primer cuadro de texto para nombrar el controlador. Crearemos sólo una acción por ahora, Index. Luego compruebe si el espacio de nombres está establecido correctamente.

He creado un test controller tanto para el frontend como para el backend. No es realmente necesario, pero es conveniente. Me ayuda cuando estoy pensando en cosas asociar intuitivamente un test controller en el backend con cosas que quiero probar en el backend. También, dejo código preparado mientras pienso sobre si quiero usarlo o no.

Bien, haga clic sobre preview en Gii, se mostrarán dos archivos, el controlador y la vista. Una vez que hagamos clic sobre generate, También creará el directorio para almacenar el archivo de la vista. Hagámoslo ahora.

Si todo ha ido bien, debería tener un archivo TestController.php file in frontend/controllers y un archivo frontend/views/test/index.php.

Bien, demos un vistazo a TestController.php.

Gist:

Test controller

Del libro:

```
<?php

namespace frontend\controllers;

class TestController extends \yii\web\Controller
{
    public function actionIndex()
    {
        return $this->render('index');
    }
}
```

Eso es todo. Puede ver que consiste en muy poco con excepción del método index, y todo lo que hace es renderizar una vista denominada index. Revisemos la vista.

Gist:

Test Index View

Del libro:

```
<?php
/* @var $this yii\web\View */
?
<h1>test/index</h1>

<p>
    You may change the content of this page by modifying
    the file <code><?= __FILE__; ?></code>.
</p>
```

Nuevamente, casi nada, así que es muy fácil experimentar con ella. Pruebe ir a yii2build.com/index.php?r=test y deberá observar la vista.

Bien, piense en ella como un lienzo en blanco que puede usar para probar diferentes fragmentos de código para descubrir cómo funcionan las cosas o lo que está contenido en `var_dump()`.

Probablemente debería crear un controlador test tanto para el frontend como para el backend. Tengo la sensación de que le será de mucha utilidad, muy pronto.

Note, que podría necesitar especificar el view path si no se genera el archivo de la vista, lo que me ocurrió cuando lo hice:

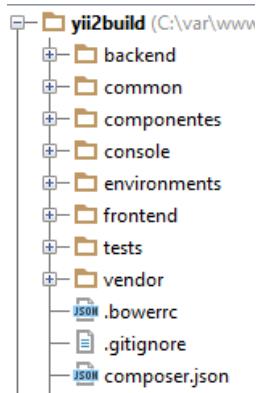
```
/var/www/yii2build/backend/views/test
```

Por favor tenga en mente que su path puede diferir si el path al directorio de su sitio es diferente. Si los archivos se crean en una ubicación no deseada, eliminelos y cree los nuevamente o muévalos. Sólo asegúrese de que los espacios de nombres son correctos.

Componentes

Vamos a crear un directorio llamado componentes con el propósito de guardar nuestro widget personalizado, que aún no hemos construido. En la guía de Yii 2, se usa un ejemplo donde muestran que el widget reside en un directorio component, así que seremos tan similares como sea posible.

Así que comenzamos por crear un nuevo directorio en nuestro directorio de proyecto denominado componentes. Su estructura de directorios debería verse así:



Directorio Componentes Agregado

Probaremos nuestra configuración creando un componente, que es diferente de un widget, pero que residirá en el mismo directorio. Los widgets se usan en las vistas y muestran algo al usuario. Los componentes trabajan detrás del escenario. Tendrá una mejor idea con este ejemplo.

Así que creamos MiComponente.php dentro del directorio componentes.

Gist:

[MiComponente.php](#)

Del libro:

```
<?php

namespace componentes;

use Yii;
use yii\base\Component;
use yii\base\InvalidConfigException;

class MiComponente extends Component
{
    public function blastOff()
    {
        echo "Houston, tenemos ignición...";
    }
}
```

Para crear un componente, extendemos Component. No olvide incluir las sentencias use necesarias.

Luego simplemente creamos el método que deseamos llamar, en nuestro caso estamos creando el método blastOff. Simplemente estoy usando una sentencia echo como demostración.

Blastoff, para nuestros lectores internacionales que pueden no entender esta frase en inglés, es lo que ocurre cuando un cohete despega. Houston tenemos ignición es lo que los controladores de suelo dicen a control de misión cuando el motor del cohete enciende. Se ha convertido en una broma en inglés que significa que las cosas están en camino, algo ha comenzado, etc., usualmente algo dramático.

De cualquier manera, de regreso al componente. Ahora, si usted intentara usar este componente desde dentro de uno de sus controladores de prueba, devolvería un error porque al autocargador no puede ver el archivo aún. Tenemos que modificar nuestro archivo common/config/bootstrap.php para darle a nuestra aplicación visibilidad del directorio.

Cambielo por lo siguiente:

Gist:

[Common/Config/Bootstrap](#)

Del libro:

```
<?php  
Yii::setAlias('common', dirname(__DIR__));  
Yii::setAlias('frontend', dirname(dirname(__DIR__)) . '/frontend');  
Yii::setAlias('backend', dirname(dirname(__DIR__)) . '/backend');  
Yii::setAlias('componentes', dirname(dirname(__DIR__)) . '/componentes');  
Yii::setAlias('console', dirname(dirname(__DIR__)) . '/console');
```

Estamos usando el método setAlias para crear un alias a nuestro directorio. Puede leer sobre el método setAlias en la guía, donde lo explican mejor de lo que yo puedo hacerlo:

[Yii 2 setAlias de la Guía](#)

Ahora necesitamos sólo un paso más, tenemos que modificar nuestro arreglo components en common/config/main.php para hacer referencia a MiComponente.php. Voy a proveerle del archivo completo porque una pequeña cosa fuera de lugar hará que nada funcione.

Por favor note, que tendrá que ingresar su app id y secret para facebook nuevamente porque estoy usando placeholders genéricos para esos valores.

Gist:

[Config/Main](#)

Del libro:

```
<?php
return [
    'vendorPath' => dirname(dirname(__DIR__)) . '/vendor',
    'extensions' => require(__DIR__ . '/../../vendor/yiisoft/extensions.php'),
    'modules' => [
        'social' => [
            // the module class
            'class' => 'kartik\social\Module',

            // the global settings for the disqus widget
            'disqus' => [
                'settings' => ['shortname' => 'DISQUS_SHORTNAME'] // default settings
            ],

            // the global settings for the facebook plugins widget
            'facebook' => [
                'appId' => 'your id',
                'secret' => 'your secret',
            ],

            // the global settings for the google plugins widget
            'google' => [
                'clientId' => 'GOOGLE_API_CLIENT_ID',
                'pageId' => 'GOOGLE_PLUS_PAGE_ID',
                'profileId' => 'GOOGLE_PLUS_PROFILE_ID',
            ],

            // the global settings for the google analytic plugin widget
            'googleAnalytics' => [
                'id' => 'TRACKING_ID',
                'domain' => 'TRACKING_DOMAIN',
            ],

            // the global settings for the twitter plugin widget
            'twitter' => [
                'screenName' => 'TWITTER_SCREEN_NAME'
            ],
        ],
        // your other modules
    ],
    'components' => [

```

```

'cache' => [
    'class' => 'yii\caching\FileCache',
],
'micomponente' => [
    'class' => 'componentes\MiComponente',
],
];
];

```

Luego tenemos que asegurarnos de que podemos realizar el blastOff como planeamos, así que modifique la acción index del controlador test. Use el frontend para seguir mi ejemplo exactamente:

Gist:

[Test Controller blastOff](#)

Del libro:

```

<?php

namespace frontend\controllers;

use yii;

class TestController extends \yii\web\Controller
{
    public function actionIndex()
    {
        Yii::$app->micomponente->blastOff();
    }
}

```

No olvide la sentencia use. Para probar esto simplemente dirija su navegador a:

<http://yii2build.com/index.php?r=test>

Si todo ha ido bien, deberá haber podido realizar el blastOff. Y ahora tiene un directorio componentes funcional.

Creando un Widget personalizado

De regreso a nuestro asunto del cliente demandante y difícil de satisfacer, podemos pensar en circunstancias donde el cliente necesite flexibilidad para mostrar una vista parcial. De hecho podemos volver a nuestras FAQs e imaginar que el cliente quiere experimentar con su ubicación, y tal vez limitar la visualización de las FAQs a aquellas destacadas, ordenadas por peso, de manera de poder controlar el orden.

De alguna forma, esto es extremadamente simple para nosotros, pero ayuda al cliente a refinar su mensaje de marketing, así que estará tan excitado sobre esta característica como con cualquiera de las características más complicadas del sitio.

Las buenas noticias son que debido a que estuvimos planeando que el cliente expandiera sus requerimientos, comenzamos con una base firme que ya se encuentra en su lugar. Por ejemplo, nuestra estructura de datos soporta una columna destacada, que hemos llamado `faq_is_featured`, y ya hemos realizado ordenamiento de acuerdo a `faq_weight`.

Así que sería fácil para nosotros realizar un nuevo método en `FaqSearch` que devolverá las Faqs que tienen un valor de `faq_is_featured` igual a 1, y ordenadas por weight. Continuemos y agreguemos el siguiente método a `FaqSearch`.

Gist:

[FaqSearch Featured Provider](#)

Del libro:

```
public function featuredProvider()
{
    $query = new Query;
    $featuredProvider = new ArrayDataProvider([
        'allModels' => $query->from('faq')->
            where(['faq_is_featured' => 1])->all(),
        'sort' => [
            'defaultOrder' => [
                'faq_weight' => SORT_ASC,
            ],
            'attributes' => ['faq_question', 'faq_answer', 'faq_weight'],
        ],
        'pagination' => [
            'pageSize' => 10,
        ],
    ]);
}
```

```
    return $featuredProvider;

}
```

Ahora necesitamos una forma de probar esto, así que usemos nuestro test controller del backend.

Modifique backend/controllers/TestController.php.

Gist:

[Backend Test Controller](#)

Del libro:

```
<?php
```

```
namespace backend\controllers;

use Yii;
use backend\models\Faq;
use backend\models\search\FaqSearch;
use yii\web\Controller;
use yii\web\NotFoundHttpException;
use yii\filters\VerbFilter;

class TestController extends \yii\web\Controller
{
    public function behaviors()
    {
        return [
            'verbs' => [
                'class' => VerbFilter::className(),
                'actions' => [
                    'delete' => ['post'],
                ],
            ],
        ];
    }

    /**
     * Lists all Faq models.
     * @return mixed

```

```

/*
public function actionIndex()
{
    $searchModel = new FaqSearch();

    $provider = $searchModel->featuredProvider();

    return $this->render('index', [
        'searchModel' => $searchModel,
        'provider' => $provider,
    ]);
}

}

```

Realmente el único cambio aquí es el método que llamamos desde \$searchModel, en este caso featuredProvider. No estamos modificando nada en los behaviors porque no es eso lo que estamos probando.

Un paso más para probar nuestro método, que es configurar backend/views/test/index.php.

Gist:

Backend Test Index

Del libro:

```

<?php
use yii\helpers\Html;
use yii\helpers\ArrayHelper;
use yii\helpers\Url;

$this->title = 'FAQs';
$this->params['breadcrumbs'][] = $this->title;

?>

<div class="site-about">
    <h1><?= Html::encode($this->title) ?></h1>
    </BR>
    <div class="panel panel-default">

```

```
<div class="panel-heading">
    <h3 class="panel-title">
        Preguntas
    </h3>
</div>

<?php
$data = $provider->getModels();
$questions = Arrayhelper::map($data, 'faq_question', 'id');
foreach ($questions as $question => $id){

    $url = Url::to(['faq/view', 'id'=>$id]);
    $options = [];
    echo '<div class="panel-body">' .
Html::a($question, $url, $options) . '</div>';
}

?>

</div>
</div>
```

Esto debería resultar familiar, todo lo que hicimos fue copiar el archivo frontend/views/faq/index.php en backend/views/test/index.php.

Sólo asegúrese de que ha creado 5 FAQs destacadas, con diferente peso para cada una, y un par de FAQs que no son destacadas, para poder verificar los resultados. Ponga mucha atención al orden para asegurarse de que es el correcto. Debería funcionar según lo planeado.

Creo que esta es una buena oportunidad para explorar algunas otras alternativas sobre cómo podemos realizar esto. Primero, intentemos usar SqlDataProvider, que nos permite emplear sentencias sql. Para aquellos que estamos más acostumbrados a trabajar con SQL, en realidad es más sencillo de utilizar.

Gist:

FaqSearch SqlDataProvider

Del libro:

```

public function featuredProvider()
{
    $count = Yii::$app->db->createCommand(
        'SELECT COUNT(*) FROM `faq` WHERE `faq_is_featured` =
        :faq_is_featured', [':faq_is_featured' => 1])->queryScalar();

    $featuredProvider = new SqlDataProvider([
        'sql' => 'SELECT * FROM `faq` WHERE `faq_is_featured` =
        :faq_is_featured ORDER BY `faq_weight` ASC',
        'params' => [':faq_is_featured' => 1],
        'totalCount' => $count,
        'sort' => [
            'attributes' => [
                'id',
                'faq_question'
            ],
            'pagination' => [
                'pageSize' => 10,
            ],
        ],
    ]);

    return $featuredProvider;
}

```

No olvidemos agregar la sentencia use:

```
use yii\data\SqlDataProvider;
```

Aquí estamos utilizando SQL para decirle a la consulta en qué orden queremos los resultados.

Comenzamos con una consulta de conteo que usaremos para la paginación. Luego tenemos la consulta principal, donde simplemente le decimos de manera explícita a SQL que queremos los resultados de faq donde faq_is_featured = 1 ordenados por faq_weight de manera ascendente.

Esto funciona perfectamente, pero debido a que estamos usando uno de los widgets de Yii 2, la paginación no funciona.

Esto expone una falla en nuestro enfoque porque podemos no tener suficientes resultados que justificarián el uso de paginación, y es una lástima no sacar provecho del objeto pagination de Yii 2 y el widget LinkPager porque son muy útiles.

Así que revisando la documentación, encontré una forma más concisa de hacer todo esto y aún así obtener todo lo que necesitamos. Vamos a ignorar el método search model e ir directamente al controlador.

Gist:

[Backend Test Controller FAQ](#)

Del libro:

```
<?php
```

```
namespace backend\controllers;

use Yii;
use backend\models\Faq;
use backend\models\search\FaqSearch;
use yii\web\Controller;
use yii\web\NotFoundHttpException;
use yii\filters\VerbFilter;
use yii\data\Pagination;

class TestController extends \yii\web\Controller
{

    public function behaviors()
    {
        return [
            'verbs' => [
                'class' => VerbFilter::className(),
                'actions' => [
                    'delete' => ['post'],
                ],
            ],
        ];
    }

    /**
     * Lists all Faq models.
     * @return mixed
     */
    public function actionIndex()
    {
        $query = Faq::find()->where(['faq_is_featured' => 1]);
    }
}
```

```

$query->orderBy(['faq_weight' => SORT_ASC]);
$countQuery = clone $query;
$pages = new Pagination(['defaultPageSize' => 3,
'totalCount' => $countQuery->count()]);
$models = $query->offset($pages->offset)
->limit($pages->limit)
->all();

return $this->render('index', [
'models' => $models,
'pages' => $pages,
]);
}

}

```

Ahora estamos liberando realmente el poder de Yii 2. Vayamos paso a paso.

```
$query = Faq::find()->where(['faq_is_featured' => 1]);
```

Comenzamos seteando nuestra consulta, una sintaxis super fácil de comprender a estas alturas. Luego le decimos cómo queremos ordenar la consulta:

```
$query->orderBy(['faq_weight' => SORT_ASC]);
```

La sentencia de Yii 2's es tan simple, que no necesitamos explicarla. Luego realizamos una copia del objeto query para devolver un conteo, que usaremos en nuestro objeto paginación.

```
$countQuery = clone $query;
```

Note el uso de clone, super eficiente y fácil de usar. Usamos clone porque necesitamos una copia separada de nuestra consulta para devolver el conteo.

Luego creamos el objeto paginación:

```
$pages = new Pagination(['defaultPageSize' => 3,
'totalCount' => $countQuery->count()]);
```

Observe que estamos estableciendo la configuración de la paginación directamente a través del constructor. Establecí el tamaño de la página en 3, para que podamos probarlo fácilmente. Y por supuesto usamos \$countQuery para devolver el número de resultados, de modo que nuestra paginación realice sus cálculos.

Luego podemos configurar nuestro objeto \$models:

```
$models = $query->offset($pages->offset)
    ->limit($pages->limit)
    ->all();
```

Puede ver que hemos pasado como parámetro nuestro objeto \$pages al método offset de query, que es la forma en la que se establece el límite de registros para cada página.

Luego viene el método render:

```
return $this->render('index', [
    'models' => $models,
    'pages' => $pages,
]);
```

Estamos entregando nuestros dos objeto, \$models y \$pages, a la vista.

Ahora tenemos que configurar la vista index.

Gist:

Test Index FAQ

Del libro:

```
<?php
use yii\helpers\Html;
use yii\helpers\Url;
use yii\widgets\LinkPager;

$this->title = 'FAQs';
$this->params['breadcrumbs'][] = $this->title;

?>

<div class="site-about">
    <h1><?= Html::encode($this->title) ?></h1>
    </BR>
    <div class="panel panel-default">
        <div class="panel-heading">
            <h3 class="panel-title">
                Preguntas
            </h3>
        </div>
```

```
<?php

foreach ($models as $model){

    $url = Url::to(['faq/view', 'id'=>$model->id]);
    $options = [];
    echo '<div class="panel-body">'.
Html::a($model->faq_question, $url, $options) . '</div>';

}

echo LinkPager::widget([
    'pagination' => $pages,
]);


?>

</div>
</div>
```

Puede ver lo conciso que es esto. No hay necesidad de configurar un arreglo, ya que estamos trabajando con el objeto. \$models entra en nuestro ciclo foreach y \$pages en nuestro widget LinkPager. Y tan sencillo como eso, tenemos paginación funcional.

Ahora si usted hace clic sobre uno de los enlaces Faq, verá que se dirige a una vista del backend. Eso no es lo que deseamos, pero por el momento está bien. Una vez que hayamos creado nuestro widget, lo llamaremos desde el frontend y los links se dirigirán hacia allí.

Está claro que hemos aprovechado mucho nuestro controlador test. Nos permitió bosquejar lo que deseábamos para la lógica de nuestro widget. Ahora sabemos exactamente cómo vamos a formatear la consulta.

Bien, ahora estamos listos para encarar el widget directamente. Comencemos creando un directorio dentro de components denominado views. Este contendrá la vista de nuestro widget.

Luego creamos un archivo en blanco para el widget llamado FaqWidget.php dentro del directorio components. Asegúrese de seguir estas instrucciones cuidadosamente, de otra manera nada funcionará.

Su estructura de directorios debería verse así:



Árbol de Directories

Ahora tenemos que dar el siguiente paso, que es incluir FaqWidget en nuestro archivo common/config/main.php. Sólo voy a mostrarle el arreglo components. Debería tener el archivo completo desde la sección de componentes, así que simplemente estaremos agregándolo aquí.

Gist:

[Components Config](#)

Del libro:

```
'components' => [
    'cache' => [
        'class' => 'yii\caching\FileCache',
    ],
    'micomponente' => [
        'class' => 'components\MiComponente',
    ],
    'faqwidget' => [
        'class' => 'components\FaqWidget',
    ],
],
```

Puede ver que estamos proveyéndole del nombre de clase en minúsculas, luego el espacio de nombres, que identifica dónde reside la clase. Esto permite al autocargador mapearlo correctamente para que podamos usarlo mediante sentencias use.

Antes de poder probar nada, necesitamos crear la clase FaqWidget y su correspondiente vista.

Comencemos con FaqWidget.php.

Gist:

[FaqWidget.php](#)

Del libro:

```
<?php

namespace componentes;

use yii\base\Widget;
use yii\helpers\Html;
use Yii;
use backend\models\Faq;
use backend\models\search\FaqSearch;
use yii\web\NotFoundHttpException;
use yii\filters\VerbFilter;
use yii\data\Pagination;

class FaqWidget extends Widget
{
    public $models;
    public $pages;

    public function init()
    {
        parent::init();

        $query = Faq::find()->where(['faq_is_featured' => 1]);
        $query->orderBy(['faq_weight' => SORT_ASC]);
        $countQuery = clone $query;
        $this->pages = new Pagination(['defaultPageSize' => 2,
'totalCount' => $countQuery->count()]);
        $this->models = $query->offset($this->pages->offset)
            ->limit($this->pages->limit)
            ->all();
    }

    public function run()
    {
        return $this->render('faq', [
            'models' => $this->models,
            'pages' => $this->pages,
        ]);
    }
}
```

Puede ver todas las sentencias use necesarias:

```
namespace components;

use yii\base\Widget;
use yii\helpers\Html;
use Yii;
use backend\models\Faq;
use backend\models\search\FaqSearch;
use yii\web\NotFoundHttpException;
use yii\filters\VerbFilter;
use yii\data\Pagination;
```

Obviamente estamos usando componentes como nuestro espacio de nombres e incluyendo todo mediante sentencias use.

Luego declaramos la clase y dos propiedades públicas:

```
class FaqWidget extends Widget
{
    public $models;
    public $pages;
```

Regresaré al por qué necesitamos declarar las propiedades en un minuto.

El resto de la clase está compuesto por sólo dos métodos, un método init y un método run. El método init:

```
public function init()
{
    parent::init();

    $query = Faq::find()->where(['faq_is_featured' => 1]);
    $query->orderBy(['faq_weight' => SORT_ASC]);
    $countQuery = clone $query;
    $this->pages = new Pagination(['defaultPageSize' => 2,
        'totalCount' => $countQuery->count()]);
    $this->models = $query->offset($this->pages->offset)
        ->limit($this->pages->limit)
        ->all();
}
```

Comenzamos el método llamando a parent::init. Lo que necesita saber sobre init es que actúa como un constructor, se ejecutará cuando la clase sea llamada. Eso significa que toda la lógica necesaria

será realizada y verá que esta es simplemente la que construimos en nuestro controlador de prueba, con una diferencia. Tuvimos que asignar las propiedades de clase \$pages y \$model usando \$this->models y \$this->pages. De esa forma podemos acceder los valores desde nuestro otro método, run():

```
public function run()
{
    return $this->render('faq', [
        'models' => $this->models,
        'pages' => $this->pages,
    ]);
}
```

En este caso, run devuelve un método render para llamar a la vista faq de nuestro widget. Así que sólo para recapitular, el método init realiza la consulta, asigna los valores a las propiedades de clase, que luego son enviadas al método run via \$this.

Me parece un flujo bastante intuitivo. Es como un mini-controlador llamando a una vista.

Ahora construyamos nuestra vista. Dentro del directorio component/views, cree el siguiente archivo, faq.php.

Gist:

Faq View

Del libro:

```
<?php
use yii\helpers\Html;
use yii\helpers\Url;
use yii\widgets\LinkPager;

?>

<div class="site-about">

    </BR>
    <div class="panel panel-default">
        <div class="panel-heading">
            <h3 class="panel-title">
                Featured Questions
            </h3>
        </div>
    <?php
```

```
foreach ($models as $model){

    $url = Url::to(['faq/view', 'id'=>$model->id]);
    $options = [];
    echo '<div class="panel-body">' .
Html::a($model->faq_question, $url, $options) . '</div>';

}

echo LinkPager::widget([
    'pagination' => $pages,
]);


?>

</div>
```

Simplemente copiamos de la página index de Test y eliminamos lo que no necesitamos como el título y el h1.

Bien, así que estamos listos para hacer que esto funcione. Vayamos a frontend/views/site/index.php y agreguemos esto entre la última y la penúltima div:

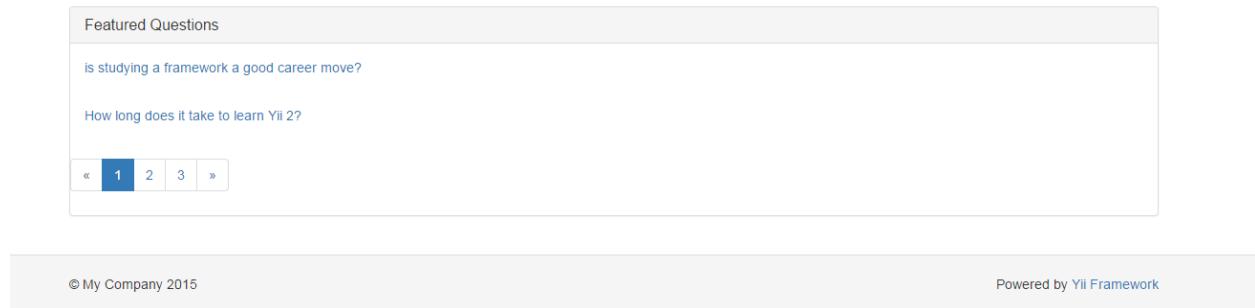
```
</div>

<?= FaqWidget::widget() ?>
</div>
```

No hay necesidad de un gist para una sola línea. ¿Qué tan sencillo fue? Oh, no olvide la sentencia use al comienzo del archivo:

```
use componentes\FaqWidget;
```

Y ahora si guarda los cambios y refresca su página index, debería ver algo como lo siguiente en la parte inferior de la página:



Así que ahora tiene un widget con paginación para faq destacadas que puede embeber en cualquier parte de su sitio, que también respeta el orden de acuerdo a weight. También está estilizado con Bootstrap, lo que significa que se adapta a cualquier dispositivo. ¡No es genial?

Es muy sencillo añadir un parámetro opcional al widget, así que voy a cubrir eso. Hagamos que el tamaño de la página para la paginación sea configurable.

Comenzamos con nuestra clase FaqWidget declarando una nueva propiedad:

```
public $pageSize;
```

Esta contendrá el valor del tamaño de página cuando lo reciba, o lo establecerá en la sentencia if, que agregaremos ahora:

```
parent::init();

if ($this->pageSize === null) {
    $this->pageSize = 2;
}
```

Así que si no está establecida, lo hacemos al valor 2.

Luego simplemente necesitamos usar una variable en lugar de un número fijo en el código:

```
$this->pages = new Pagination(['defaultPageSize' => $this->pageSize,
```

Adelante y guarde los cambios.

Luego modificamos la llamada al widget de esta manera:

```
<?= FaqWidget::widget(['pageSize' => 3]) ?>
```

Si lo establece a 5 y sólo tiene 5 registros destacados, LinkPager no le mostrará la paginación, así que este es un widget muy inteligente con el que jugar.

Aquí está el archivo FaqWidget.php completo como referencia.

Gist:

[FaqWidget](#)

Del libro:

```
<?php

namespace componentes;

use yii\base\Widget;
use yii\helpers\Html;
use Yii;
use backend\models\Faq;
use backend\models\search\FaqSearch;
use yii\web\NotFoundHttpException;
use yii\filters\VerbFilter;
use yii\data\Pagination;

class FaqWidget extends Widget
{
    public $models;
    public $pages;
    public $pageSize;

    public function init()
    {
        parent::init();

        if ($this->pageSize === null) {
            $this->pageSize = 2;
        }

        $query = Faq::find()->where(['faq_is_featured' => 1]);
        $query->orderBy(['faq_weight' => SORT_ASC]);
        $countQuery = clone $query;
        $this->pages = new Pagination(['defaultPageSize' => $this->pageSize,
'totalCount' => $countQuery->count()]);
        $this->models = $query->offset($this->pages->offset)
->limit($this->pages->limit)
->all();
    }
}
```

```
}

public function run()
{
    return $this->render('faq', [
        'models' => $this->models,
        'pages' => $this->pages,
    ]);
}
}
```

Así que estaba feliz con esto, pero luego pensé nuevamente sobre nuestro cliente difícil de satisfacer. Si el widget Faq le gustaba mucho, quizás querría una versión para todas sus FAQs. De hecho podrían cambiar de idea sobre el asunto varias veces. Así qué, ¿qué deberíamos hacer? ¿Cómo podemos facilitarnos las cosas?

Bien, en lugar de recibir un sólo parámetro, ¿por qué no recibir un arreglo de configuración? Entonces podríamos probar algo como 'featuredOnly' con valor verdadero o falso, y dependiendo de la respuesta, darles sólo las FAQs destacadas o todas ellas.

Si lo observamos desde la llamada a nuestro widget y trabajamos hacia atrás a partir de allí, será fácil ver cómo funciona esto:

```
<?= FaqWidget::widget(['settings' => ['pageSize' => 3,
                                             'featuredOnly' => true]]) ?>
```

Así que ahora le estamos entregando un arreglo denominado settings, con dos pares clave valor. Siempre y cuando tengamos una propiedad de clase para guardar el valor de settings estaremos bien. Podemos luego acceder a él a través de la palabra clave \$this para probar los valores y realizar la lógica adecuada basada en los resultados.

Aquí hay una actualización del archivo FaqWidget.php.

Gist:

[FaqWidget con Settings](#)

Del libro:

```
<?php

namespace componentes;

use yii\base\Widget;
use yii\helpers\Html;
use Yii;
use backend\models\Faq;
use backend\models\search\FaqSearch;
use yii\web\NotFoundHttpException;
use yii\filters\VerbFilter;
use yii\data\Pagination;

class FaqWidget extends Widget
{
    public $models;
    public $pages;
    public $settings = [];

    public function init()
    {
        parent::init();

        if (!isset($this->settings['pageSize'])) {
            $this->settings['pageSize'] = 2;
        }

        if ($this->settings['featuredOnly'] === true) {
            $query = Faq::find()->where(['faq_is_featured' => 1]);
        } else {

            $query = Faq::find();
        }
        $query->orderBy(['faq_weight' => SORT_ASC]);
        $countQuery = clone $query;
        $this->pages = new Pagination(['defaultPageSize' =>
$this->settings['pageSize'], 'totalCount' => $countQuery->count()]);
        $this->models = $query->offset($this->pages->offset)
->limit($this->pages->limit)
->all();
    }
}
```

```
    }

    public function run()
    {
        return $this->render('faq', [
            'models' => $this->models,
            'pages' => $this->pages,
            'settings' => $this->settings,
        ]);
    }
}
```

Bien, analicemos esto. Puede ver que ahora tenemos un arreglo de configuración que se inicializa como un arreglo vacío.

```
class FaqWidget extends Widget
{
    public $models;
    public $pages;
    public $settings = [];
}
```

El método `widget` de la clase `Widget` recibirá los valores que le pasamos en la firma del método y los asignará a este arreglo porque busca una propiedad con el mismo nombre. Esto es algo inteligente y muy poderoso. Abre todo un rango de posibilidades para usted cuando está construyendo widgets.

En nuestro caso, sólo estamos entregándolo dos pares clave valor, así que esta es una implementación relativamente simple. Puede ver como luego usamos los valores del arreglo para verificar.

```
if (!isset($this->settings['pageSize'])) {
    $this->settings['pageSize'] = 2;
}

if ($this->settings['featuredOnly'] === true) {
    $query = Faq::find()->where(['faq_is_featured' => 1]);
} else {

    $query = Faq::find();
}
```

Ahora verificamos contra las claves del arreglo para realizar nuestra lógica sobre los valores. En el caso de la segunda sentencia if, buscamos una clave llamada 'featuredOnly' y si su valor es

verdadero, entonces la consulta sólo incluirá los registros donde featured es igual a uno, de otra manera incluirá todos.

Luego modificamos nuestro método render en consecuencia:

```
public function run()
{
    return $this->render('faq', [
        'models' => $this->models,
        'pages' => $this->pages,
        'settings' => $this->settings,
    ]);
}
```

Notará que estamos enviando el arreglo \$settings a la vista. La razón para ello es que deseamos verificar la clave 'featuredOnly', y dependiendo de los resultados, mostrar un encabezado diferente.

Gist:

[Faq View](#)

Del libro:

```
<?php
use yii\helpers\Html;
use yii\helpers\Url;
use yii\widgets\LinkPager;

?>

<div class="site-about">

    </BR>
    <div class="panel panel-default">
        <div class="panel-heading">
            <h3 class="panel-title">
                <?php
                    if ($settings['featuredOnly'] == true){
                        echo 'Featured Questions';
                    } else {
                        echo 'FAQs';
                    }
                ?>
            </h3>
```

```
</div>

<?php

foreach ($models as $model){

    $url = Url::to(['faq/view', 'id'=>$model->id]);
    $options = [];
    echo '<div class="panel-body">' .
        Html::a($model->faq_question, $url, $options) . '</div>';

}

echo LinkPager::widget([
    'pagination' => $pages,
]);

?>

</div>
</div>
```

Verificamos si sólo estamos usando ‘featuredOnly’, y en ese caso, mostramos el encabezado apropiado. Alternativamente, podríamos usar un operador ternario lo cual también estaría bien.

Para concluir, realicemos la llamada al widget, aunque es un poco más larga ahora, todavía se trata de una sola línea de código, si no está evitando el ajuste de texto, por supuesto:

```
<?= FaqWidget::widget(['settings' =>
    ['pageSize' => 3, 'featuredOnly' => true]]) ?>
```

Y eso es todo. Ahora tenemos un widget FAQ configurable que podemos usar en cualquier vista, con una llamada de una sola línea. Simplemente recuerde incluir las sentencias use si lo utiliza en algún otro lugar de su sitio.

CDN

Bien, muchos de ustedes deben ya saber que usar CDN, una red de entrega de contenidos, para incluir los recursos css, js, y jquery assets puede mejorar dramáticamente el desempeño de su sitio.

La razón para ello es que las versiones CDN se guardan típicamente en el cache del navegador, y de esta forma no tienen que descargar la librería completa cada vez que visita un sitio. Y debido a que estos recursos son tan comunes, probablemente su sitio no será el primero que los utiliza, así que ya estarán en la memoria cache. Esto puede hacer una enorme diferencia en la velocidad de carga de la página.

Hay una manera sencilla de hacer esto en common/config/main.php. Como conveniencia, voy a entregarle el archivo completo, pero no olvide que uso placeholders genéricos para id y el secret de Facebook.

Gist:

Main with CDN

Del libro:

```
<?php
return [
    'vendorPath' => dirname(dirname(__DIR__)) . '/vendor',
    'extensions' => require(__DIR__ . '/../../.vendor/yiisoft/extensions.php'),
    'modules' => [
        'social' => [
            // the module class
            'class' => 'kartik\social\Module',

            // the global settings for the disqus widget
            'disqus' => [
                'settings' => ['shortname' => 'DISQUS_SHORTNAME']
            ],
            // the global settings for the facebook plugins widget
            'facebook' => [
                'appId' => 'your app id',
                'secret' => 'your secret',
            ],
            // the global settings for the google plugins widget
            'google' => [
                'clientId' => 'GOOGLE_API_CLIENT_ID',
                'pageId' => 'GOOGLE_PLUS_PAGE_ID',
                'profileId' => 'GOOGLE_PLUS_PROFILE_ID',
            ],
            // the global settings for the google analytic plugin widget
            'googleAnalytics' => [

```

```
        'id' => 'TRACKING_ID',
        'domain' => 'TRACKING_DOMAIN',
    ],
    // the global settings for the twitter plugin widget
    'twitter' => [
        'screenName' => 'TWITTER_SCREEN_NAME'
    ],
],
// your other modules
],
'components' => [
    'assetManager' => [
        'bundles' => [
            // use bootstrap css from CDN
            'yii\bootstrap\BootstrapAsset' => [
                'sourcePath' => null, // do not use file from our server
                'css' => [
                    'https://maxcdn.bootstrapcdn.com/bootstrap/3.3.0/css/bootstrap.min.css']
                ],
            // use fontawesome css from CDN
            'frontend\assets\FontAwesomeAsset' => [
                'sourcePath' => null, // do not use file from our server
                'css' => [
                    'https://maxcdn.bootstrapcdn.com/font-awesome/4.2.0/css/font-awesome.min.css']
                ],
            // use fontawesome css from CDN
            'backend\assets\FontAwesomeAsset' => [
                'sourcePath' => null, // do not use file from our server
                'css' => [
                    'https://maxcdn.bootstrapcdn.com/font-awesome/4.2.0/css/font-awesome.min.css']
                ],
            // use bootstrap js from CDN
            'yii\bootstrap\BootstrapPluginAsset' => [
                'sourcePath' => null, // do not use file from our server
                'js' =>
                    'https://maxcdn.bootstrapcdn.com/bootstrap/3.3.0/js/bootstrap.min.js']
                ],
            // use jquery from CDN
            'yii\web\JqueryAsset' => [

```

```

        'sourcePath' => null,    // do not publish the bundle
        'js' => [
            'https://ajax.googleapis.com/ajax/libs/jquery/2.1.1/jquery.min.js',
        ],
    ],
],
['cache' => [
    'class' => 'yii\caching\FileCache',
],
'mycomponent' => [
    'class' => 'components\MyComponent',
],
'faqwidget' => [
    'class' => 'components\FaqWidget',
],
],
];
];

```

Al establecer la ruta a null, estamos blanqueando la ruta, luego estableciéndola a la CDN. La única otra cosa notable aquí es que tuvimos que configurar font-awesome tanto para el frontend como para el backend, ya que hemos creado assets para font-awesome en ambos lugares.

Una vez que estos cambios estén en su lugar, creo que notará que su aplicación se mueve mucho más rápidamente, lo que siempre es un plus.

Resumen

Felicitaciones, ha completado el primer capítulo extra. He agregado algunas características geniales a la plantilla, y en el proceso de hacerlo, he aprendido mucho más sobre Yii 2.

Esa es toda la conclusión que puedo obtener ahora porque este no es en realidad el final del libro. Planeo agregar nuevo material regularmente. Yii 2 es el framework PHP principal que uso personalmente, y lo tengo en tan alta estima, que quiero continuar compartiéndolo con usted.

He llegado a comprender que escribir un libro sobre una framework es más un viaje que un destino. Por ejemplo, Yii 2 estará realizando lanzamientos periódicos de versiones y las cosas pueden cambiar.

Haré mi mayor esfuerzo para mantenerme al tanto de los cambios y revisar el libro de acuerdo a ello. Por cada actualización de Yii 2, se realizarán ajustes al libro.

Los capítulos extra, por otra parte, simplemente serán agregados de manera sucesiva, a menos que un método haya cambiado debido a un cambio de versión del propio Yii 2. Las siguientes adiciones nos darán la sensación de que estamos elaborando sobre algo.

Su compra de Yii 2 Para Principiantes le otorga el derecho a actualizaciones durante la vida del libro, así que no hay razón para no beneficiarse de ello. Simplemente busque novedades de actualización en su email o revise el sitio web leanpub.com para la última actualización.

Ahora tomaré un momento para agradecer a todos los lectores que me enviaron comentarios positivos o notificaciones de errores. Realmente me ayuda en mi búsqueda de llevar Yii 2 Para Principiantes al nivel más alto de calidad para un libro técnico. Con su ayuda, podemos hacerlo.

Cuando tengo retroalimentación de los lectores, me da aliento para continuar. Como siempre, comentarios, links, críticas y recomendación boca a boca se aprecian grandemente. Compartamos este asombroso framework con tanta gente como sea posible.

Gracias a todos, nos vemos pronto.

Capítulo Trece: Material Extra URLs Amigables y Slugs

¡Bienvenidos nuevamente! Estamos listos para más material extra. He decidido hacer los capítulos extra más pequeños, para poder publicar el material más rápidamente.

En este capítulo vamos a configurar la aplicación para usar URLs y usar slugs. Por URL amigable, nos referimos a que vamos a reemplazar:

```
http://www.yii2build.com/index.php?r=site/contact
```

Y en su lugar ver lo siguiente:

```
http://www.yii2build.com/site/contact
```

Puede ver que tenemos dos mejoras. Primero nos deshacemos de index.php, no hay razón para mostrarlo. Segundo, nos deshacemos de ?r= y lo reemplazamos con una alternativa amigable para los buscadores.

En este capítulo, también vamos a implementar slugs. Para aquellos que no lo saben, un slug es una cadena que describe el contenido de una página y que está embebida en la url. Estaremos usando slugs con nuestro modelo Faq, y la url se verá de manera similar a:

```
http://yii2start.com/faq/11/should-i-use-a-framework
```

Puede ver que no sólo no necesitamos ‘view’, pero también agregamos el slug directamente a la url en un formato agradable para los motores de búsqueda. Este formato incrementa la visibilidad de los motores de búsqueda y esto es importante si usted o los clientes desean que la página sea encontrada por google y otros motores de búsqueda. No he conocido a nadie que no piense que eso no es importante.

En fin, todo esto es bastante fácil de configurar, pero requiere algo de trabajo.

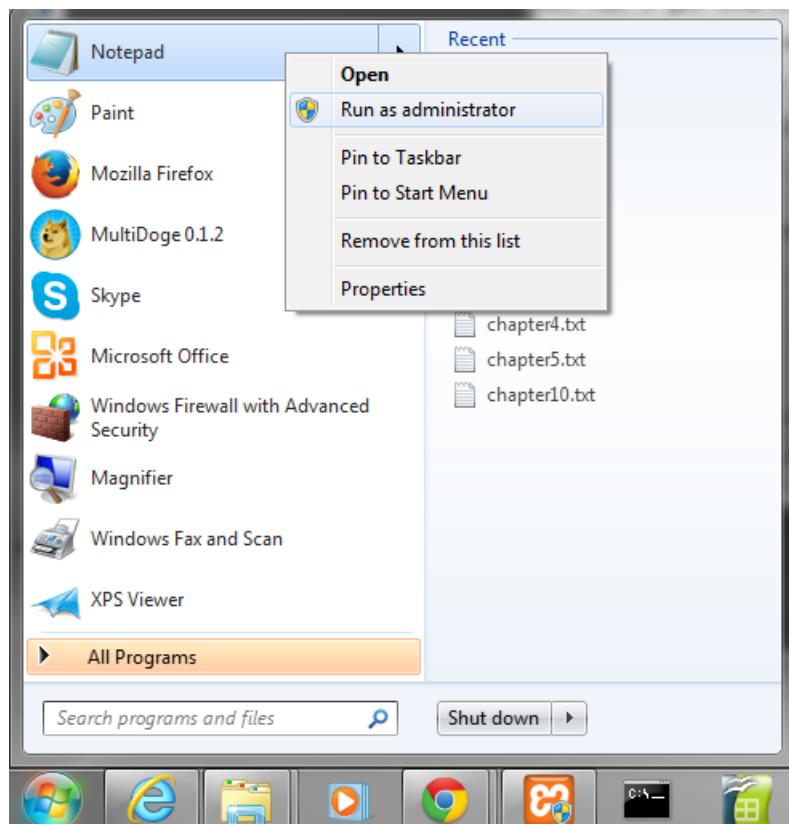
URLs Amigables

Comencemos con las URLs amigables. Los pasos necesarios son:

- Crear un archivo htaccess tanto para el frontend como para el backend en los directorios web.
- Modificar nuestro archivo Vhost para apache.
- Reiniciar Apache
- Modificar frontend/config/main.php y backend/config/main.php para el urlManager

Vhost de Apache

Desde una máquina windows editaremos esto usando notepad, ejecutándolo en modo administrador. Encuentre notepad desde el botón inicio o la barra de tareas. Clic con el botón derecho y seleccione ejecutar en modo administrador.

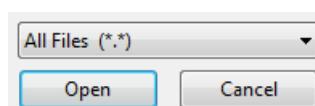


Notepad en Modo Administrador

Notepad se abrirá. Seleccione abrir archivo y la ruta a vhosts, en mi caso:

C:\xampp\apache\conf\extra\

Seleccione Todos los Archivos como tipo de archivo:



Notepad All File Types

Luego seleccione:

httpd-vhosts.conf

Modifique su entrada de hosto por la siguiente.

Gist:

[vHost Entrada en Apache](#)

Del libro:

```
NameVirtualHost *
RewriteEngine on
<VirtualHost yii2build.com>
    DocumentRoot "C:\var\www\yii2build\frontend\web"
    ServerName localhost
    ServerAlias www.yii2build.com
</VirtualHost>
NameVirtualHost *
RewriteEngine on
<VirtualHost yii2build.com>
    DocumentRoot "C:\var\www\yii2build\backend\web"
    ServerName localhost
    ServerAlias backend.yii2build.com
</VirtualHost>
```

Puede ver que todo lo que hicimos fue agregar la instrucción RewriteEngine on. Continúe guarde los cambios y cierre.

Reinic peace Apache

Asegúrese de reiniciar apache para que los cambios tengan efecto.

.htaccess

En frontend/web, cree un archivo llamado .htaccess. No olvide poner el punto enfrente del nombre.

Ponga lo siguiente en el archivo.

Gist:

[contenidos de .htaccess](#)

Del libro:

```
# 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
```

Ahora necesitamos ir a frontend/config/main.php y agregar lo siguiente al arreglo components.

Gist:

[urlManager para components](#)

Del libro:

```
'urlManager' => [ 'class' => 'yii\web\UrlManager',
    // Disable index.php
    'showScriptName' => false,
    // Disable r= routes
    'enablePrettyUrl' => true,
    'rules' => [
        '<controller:\w+>/<id:\d+>' => '<controller>/view',
        '<controller:\w+>/<action:\w+>/<id:\d+>' => '<controller>/<action>',
        '<controller:\w+>/<action:\w+>' => '<controller>/<action>',
        '<controller:\w+-\w+>/<id:\d+>' => '<controller>/view',
    ],
],
```

Como referencia, voy a incluir el archivo completo, en caso de que necesite solucionar problemas:
[frontend/config/main.php](#)

Puede ver que hemos establecido showScriptName a false, y esto se deshace de index.php. enablePrettyUrl elimina ?r= , así que de esta manera ambas se combinan para darnos las urls que deseamos.

También puede ver que como parte del arreglo urlManager, tenemos una sección rules. Las reglas usan expresiones regulares, tales como w+ y d+ para representar palabras y números. Por ejemplo, cuando pasamos a la url un controlador/acción con un parámetro id, lo tomamos en cuenta de la siguiente manera:

```
'<controller:\w+>/<action:\w+>/<id:\d+>' => '<controller>/<action>',
```

Una cosa a tener en cuenta es que cuando los controladores tienen múltiples palabras, tal como TipoUsuario, también tenemos que tomar esto en cuenta de la siguiente manera:

```
'<controller:\w+-\w+>/<id:\d+>' => '<controller>/view',
```

Recuerde que tiene que configurar backend/config/main.php exactamente de la misma manera para que esto funcione de manera correcta en toda la aplicación. La forma más fácil es simplemente copiar el archivo completo.

Nota, una vez que se implementen las URLs amigables, la ruta va a ser yii2build.com/gii.

Así que ahora que tenemos nuestras URLs amigables habilitadas, continuaremos con los slugs.

Slugs

Ya mencionamos qué son los slugs al comienzo del capítulo. Los slugs son una característica común que se encuentra en muchos sitios web, stackoverflow y yahoo por ejemplo. Añaden contexto a la url y los motores de búsqueda valoran esto, pero hay debate acerca de que tanto valor añade.

Un comentario que leí en stackoverflow afirmaba que aunque no observaron ningún incremento en el tráfico al añadir slugs, si notaron un incremento del 300% en los clics.

Mi visión particular es que los slugs son invalúables y vale la pena implementarlos. En ciertos escenarios, incluso podemos automatizar el proceso.

Sluggable Behavior

Yii 2 tiene un comportamiento pre-hecho que podemos usar para crear slugs en nuestros modelos. Nuestro modelo Faq nos provee del ejemplo perfecto sobre cómo podemos usar los slugs. Hay un número de pasos involucrados en la implementación:

- Añadir el sluggable behavior al modelo Faq
- Añadir la columna slug a la tabla faq
- Eliminar los registros antiguos y crear nuevos
- Añadir una regla a backend/config/main.php
- Cambiar la acción view en el controlador faq del backend
- Cambiar las acciones Create y Update en el controlador faq del backend
- Cambiar la url en la columna de acciones del gridview en nuestra página faq index del backend
- Cambiar el enlace de la url en el archivo de la vista faq.php para el widget
- Cambiar faq/index para usar el widget
- Cambiar el FaqController/index para simplemente renderizar la página.

Esto suena complicado, pero una vez que lo resolvamos, verá qué tan fácil es. Comencemos añadiendo el sluggable behavior al modelo Faq:

Gist:

[Sluggable Behavior](#)

Del libro:

```

public function behaviors()
{
    return [
        'timestamp' => [
            'class' => 'yii\behaviors\TimestampBehavior',
            'attributes' => [
                ActiveRecord::EVENT_BEFORE_INSERT =>
                ['created_at', 'updated_at'],
                ActiveRecord::EVENT_BEFORE_UPDATE =>
                ['updated_at'],
            ],
            'value' => new Expression('NOW()'),
        ],
        'blameable' => [
            'class' => BlameableBehavior::className(),
            'createdByAttribute' => 'created_by',
            'updatedByAttribute' => 'updated_by',
        ],
        'sluggable' => [
            'class' => SluggableBehavior::className(),
            'attribute' => 'faq_question',
        ],
        // In case of attribute that contains slug has different name
        // 'slugAttribute' => 'alias',
    ],
];
}

```

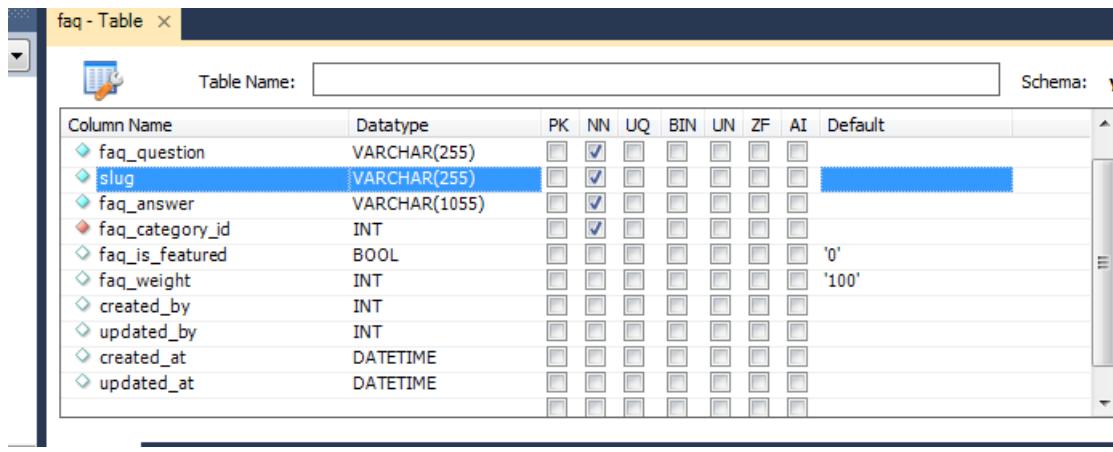
Y no olvidemos la sentencia use:

```
use yii\behaviors\SluggableBehavior;
```

Así que puede ver en el sluggable behavior, definimos un atributo, faq_question. Este es el atributo que el behavior usará para crear el slug. Puede ver que no tenemos que declarar demasiado. Simplemente necesitamos pasarle el atributo y el behavior hace el resto.

Columna Slug

Por supuesto para que esto funcione, necesitamos agregar una columna a la tabla faq:



Column Name	Datatype	PK	NN	UQ	BIN	UN	ZF	AI	Default
faq_question	VARCHAR(255)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>					
slug	VARCHAR(255)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>					
faq_answer	VARCHAR(1055)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>					
faq_category_id	INT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>					
faq_is_featured	BOOL	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	'0'
faq_weight	INT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	'100'
created_by	INT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
updated_by	INT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
created_at	DATETIME	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
updated_at	DATETIME	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	

slug column

Puede ver que la columna slug es varchar(255), no nula. Note que aunque la columna id no se muestra, está allí.

Una vez que esté en su lugar, puede crear una faq para ver si funciona.

Elimine las Faqs viejas y cree nuevas

Una vez que ha guardado exitosamente un registro, continúe y elimine las viejas FAQs y cree nuevas. Alternativamente, si ya ha poblado la tabla faq con mucho contenido que desea conservar, puede añadir slugs manualmente a través de Php MyAdmin.

Así que si tiene varias de ellas, debería obtener algo así:

Show : Start row: 0 Number of rows: 30 Headers every 100 rows

Sort by key: None

+ Options

		Edit Copy Delete	id	faq_question	slug	faq_answer	faq_cate
<input type="checkbox"/>	Edit Copy Delete	7	do slugs work?	do-slugs-work	maybe		
<input type="checkbox"/>	Edit Copy Delete	8	Is programming fun?	is-programming-fun	It might be		
<input type="checkbox"/>	Edit Copy Delete	9	How much time does it take?	how-much-time-does-it-take	It depends		
<input type="checkbox"/>	Edit Copy Delete	10	How much money can we make?	how-much-money-can-we-make	A lot if you're good		
<input type="checkbox"/>	Edit Copy Delete	11	Should I use a framework?	should-i-use-a-framework	Probably, it's a good idea		
<input type="checkbox"/>	Edit Copy Delete	12	Am I going to finish?	am-i-going-to-finish	Ysel!		
<input type="checkbox"/>	Edit Copy Delete	13	Am I still having fun?	am-i-still-having-fun	Probably		
<input type="checkbox"/>	Edit Copy Delete	14	Can this be this easy?	can-this-be-this-easy	Wouldnt' count on it		
<input type="checkbox"/>	Edit Copy Delete	15	What time is it?	what-time-is-it	Now		
<input type="checkbox"/>	Edit Copy Delete	16	is it similar to timestamp?	is-it-similar-to-timestamp	let's find out		

Slugs en la Tabla

Espero que la imagen sea lo suficientemente clara para que vea como automáticamente se eliminó el ? de los registros faq_question y se incluyó un guión entre las palabras. ¿Qué tan sencillo fue eso? Aunque hemos creado exitosamente los slugs, todavía no los vemos en la url.

Añadir Reglas al Url Manager

Necesitaremos añadir la siguiente línea al backend config/main.php tanto del frontend como del backend:

```
'<controller:\w+>/<id:\d+>/<slug:[A-Za-z0-9 -_.]+>' => '<controller>/view',
```

Simplemente agreguela luego de la última regla existente y guarde los cambios.

Ahora le hemos dado al urlManager una manera de manejar el slug en la url. Aún necesitamos modificar el controlador y las urls que apuntan a la visualización de los registros.

Modificar la Acción View en FaqController

Cambiemos la acción view en FaqController de la siguiente manera:

Gist:

[View Action FaqController](#)

Del libro:

```
public function actionView($id, $slug = null)
{
    $model = $this->findModel($id);

    if ($slug == $model->slug){

        return $this->render('view', [
            'model' => $model,
            'slug' => $model->slug
        ]);
    } else {

        throw new NotFoundHttpException('La página solicitada no existe.');
    }
}
```

Bien, la diferencia aquí es que estamos aceptando un segundo parámetro con un valor por defecto de null. La razón de por qué estamos haciendo esto es para no tener que realizar demasiado manejo de error si el slug no está incluido en la url.

Luego llamamos al registro del modelo con base en su id:

```
$model = $this->findModel($id);
```

Aún cuando permitimos que \$slug sea nulo en la firma, requerimos de una coincidencia para mostrar la página correspondiente:

```
if ($slug == $model->slug){
```

De otra manera, lanzamos una excepción.

```
else {
```

```
    throw new NotFoundHttpException('La página solicitada no existe.');
}
```

Luego simplemente pasamos \$model y \$model->slug a la vista.

```
return $this->render('view', [
    'model' => $model,
    'slug' => $model->slug
]);
```

Modificar las acciones Create y Update en el controlador del backend

Necesitamos modificar actionCreate en el controlador del backend de la siguiente manera:

Gist:

[actionUpdate](#)

Del libro:

```
public function actionCreate()
{
    $model = new Faq();

    if ($model->load(Yii::$app->request->post()) && $model->save()) {

        $url = Url::toRoute('faq/' . $model->id . '/' . $model->slug);
        return $this->redirect($url);

    } else {
        return $this->render('create', [
            'model' => $model,
        ]);
    }
}
```

Para usar Url::, necesitamos incluir la sentencia use al comienzo:

```
use yii\helpers\Url;
```

El cambio a actionUpdate es similar.

Gist:

[actionUpdate FaqController Backend](#)

Del libro:

```
public function actionUpdate($id)
{
    $model = $this->findModel($id);

    if ($model->load(Yii::$app->request->post()) && $model->save()) {

        $url = Url::toRoute('faq/' . $model->id . '/' . $model->slug);
        return $this->redirect($url);

    } else {
        return $this->render('update', [
            'model' => $model,
        ]);
    }
}
```

En ambos casos, tuvimos que formatear la url de redirección para tomar en cuenta el slug. Usamos el método Url::toRoute para formatear la url, lo que nos permite especificar una cadena de texto.

Nombramos al controlador con una / como separador, concatenamos la id del modelo, y añadimos otra / como separador y luego concatenamos el slug. ¡Y eso es todo!

Cambiar la URL de la columna acción de la Gridview

Para que el slug se visualice en la ventana de la URL, necesitamos modificar los enlaces a la visualización del registro. Ya que enlazamos a ellos a través de la Gridview en la página index, necesitamos modificar esto.

Obviamente, esto es sólo para el backend. Más tarde, haremos algunas modificaciones al frontend, dándole una solución completa.

Así que modifiquemos el widget GridView en backend/views/faq/index.php de la siguiente manera.

Gist:

[GridView](#)

Del libro:

```

<?= GridView::widget([
    'dataProvider' => $dataProvider,
    'filterModel' => $searchModel,
    'columns' => [
        ['class' => 'yii\grid\SerialColumn'],

        'id',
        'faq_question',
        'faq_answer',
        ['attribute'=>'faqCategoryName', 'format'=>'raw'],
        'faq_weight',
        ['attribute'=>'faqIsFeaturedName', 'format'=>'raw'],

        [
            'class' => 'yii\grid\ActionColumn',
            'template' => '{view} {update} {delete}',
            'buttons' => [
                'view' => function ($url,$model) {
                    return Html::a(
                        '<span class="glyphicon glyphicon-eye-open"
                        aria-hidden="true"></span>',
                        $url.'/'.$model->slug);
                },
            ],
        ],
    ],
]); ?>

```

Puede ver la diferencia. Estamos definiendo la template y los componentes de la template, {view}, por ejemplo, coincide con los botones como se muestra arriba.

En el arreglo de botones, definimos solamente el botón view, los otros usarán el comportamiento por defecto. Así que usamos una función anónima para devolver un enlace a través del método Html::a method.

Puede ver que estamos concatenando \$model->slug a la \$url:

```
$url.'/'.$model->slug
```

Así que ahora tenemos una solución completa para el backend, si intenta realizar un clic sobre los registros de faq desde la página index, obtendrá la slug, que le dará el siguiente formato para la url:

```
http://backend.yii2start.com/faq/11/should-i-use-a-framework
```

Si decide que desea un separador diferente para las palabras en el slug, puede lograrlo modificando el comportamiento de esta manera:

```
'sluggable' => [
    'class' => SluggableBehavior::className(),
    // 'attribute' => 'faq_question',
    // In case of attribute that contains slug has different name
    // 'slugAttribute' => 'alias',
    'value' => function ($event) {
        $question = rtrim($this->faq_question, '?');
        return str_replace(' ', '_', $question);
    }
],
```

Comentamos la configuración de ‘attribute’ y añadimos una configuración de un valor. Para el valor del slug, eliminamos el ? con rtrim. Luego realizamos un simple str_replace para usar un guión bajo en lugar del signo menos por defecto. Puede utilizar este formato para realizar algún otro cálculo, tal vez algo más complicado.

De cualquier manera, puede ver lo fácil que es trabajar con él. En el curso de este libro, hemos usado los cuatro comportamientos predefinidos de Yii 2 y son increíblemente útiles y fáciles de usar.

Bien, para terminar con nuestra implementación de los slugs para Faq, necesitamos modificar el frontend.

Ya que ya tenemos las reglas copiadas en nuestro urlManager del frontend, podemos continuar con el widget, faq.php.

Realicemos un cambio sencillo. En el bucle foreach, eliminemos la siguiente línea:

```
$url = Url::to(['faq/view', 'id'=>$model->id]);
```

Y reemplazémola por:

```
$url = Url::toRoute('faq/'.$model->id . '/' . $model->slug);
```

Nuevamente estamos utilizando el método Url::toRoute de Yii 2, que nos permite pasarle la url como una cadena de texto.

Eso finaliza nuestro amigable widget. ¿Pero qué ocurre con el enlace a Faq en la navegación del encabezado?

En realidad eso no va a ser sencillo de modificar, tendremos que modificar tanto el controlador como la página index.

Hmm. ¿Me pregunto qué podrá facilitarnos esto? Si sólo tuviéramos una solución pre-hecha que pudiéramos utilizar. Espere. La tenemos. ¿Por qué simplemente no usamos el widget para esta tarea, ya que está formateado?

Es verdad que todavía tenemos que modificar el FaqController del frontend, pero esto es lo que necesitamos cambiar en la acción index:

```
public function actionIndex()
{
    return $this->render('index');
}
```

Eso ni siquiera necesita de un gist. Y aquí está la acción view, exactamente igual que la versión del backend:

```
public function actionView($id, $slug = null)
{
    $model = $this->findModel($id);

    if ($slug == $model->slug){

        return $this->render('view', [
            'model' => $model,
            'slug' => $model->slug
        ]);
    } else {

        throw new NotFoundHttpException('La página solicitada no existe.');
    }
}
```

Así que nuevamente, no es necesario un Gist, puede copiar su versión del backend si lo desea.

En la página index.php, cambiemos lo siguiente.

Gist:

[Faq View Frontend](#)

Del libro:

```
use yii\helpers\Url;
use components\FaqWidget;

$this->title = 'FAQs';
$this->params['breadcrumbs'][] = $this->title;

?>

<div class="site-about">
    <h1><?= Html::encode($this->title) ?></h1>
</div>
<BR>

<?= FaqWidget::widget(['settings' =>
    ['pageSize' => 10,
     'featuredOnly' => false
    ]])?>
```

Puede ver que esto en realidad resultó más fácil. Los Widgets son asombrosos, mire cuánto trabajo nos ahorraron al no tener que preocuparnos por otro bucle for.

Hay un pequeño detalle que no está del todo bien. Estamos repitiendo la palabra Faq en la página view y eso no luce bien. ¿Por qué no modificamos nuestro widget para aceptar otro parámetro para el heading, y luego poder pasarle el valor que desamos?

Debido a el arreglo de configuración para nuestro widget acepta pares clave/valor que le entregamos como parámetro en la firma del método, simplemente podemos añadir otro par. Todo lo que tenemos que hacer es cambiar la lógica del archivo view de nuestro widget.

Así que cambiemos el archivo Faq.php de la siguiente manera:

Gist:

[FaqWidget.php](#)

Del libro:

```
<?php

use yii\helpers\Html;

use yii\helpers\Url;

use yii\widgets\LinkPager;

?>

<div class="site-about">

    <div class="panel panel-default">

        <div class="panel-heading">

            <h3 class="panel-title">

                <?= $settings['heading']; ?>

            </h3>

        </div>

    </div>

<?php

foreach ($models as $model){

    $url = Url::toRoute('faq/' . $model->id . '/' . $model->slug);
```

```
$options = [];  
  
echo '<div class="panel-body">' .  
    Htm::a($model->faq_question, $url, $options) . '</div>';  
  
}  
  
echo LinkPager::widget([  
  
    'pagination' => $pages,  
  
]);  
  
?>  
  
</div>  
  
</div>
```

Puede ver que ahora simplemente estamos realizando echo:

```
<?= $settings['heading'];?>
```

Bien, ahora simplemente tenemos que pasar el parámetro en nuestras dos llamadas al widget. Primero hágámoslo en site index.php:

```
<?= FaqWidget::widget(['settings' => [
    'pageSize' => 3,
    'featuredOnly' => true,
    'heading' => 'FAQs Destacadas'
])
]) ?>
```

Y en la vista index de Faq:

```
<?= FaqWidget::widget(['settings' => [
    'pageSize' => 3,
    'featuredOnly' => true,
    'heading' => 'Preguntas'
])
]) ?>
```

Y eso es todo. Ahora tenemos más control sobre nuestra salida. Simplemente no olvide entregar la configuración para el heading u obtendrá un error en su archivo Faq.php.

Alternativamente, puede envolverlo en una sentencia if para mostrarlo sólamente si tiene un valor:

```
<h3 class="panel-title">
<?php
if(isset($settings['heading'])){
    echo $settings['heading'];
}
?>
</h3>
```

Resumen

Al tomar el control de las URLs y los slugs, hemos hecho que nuestra aplicación sea más visible a los motores de búsqueda. Y ya que Yii 2 provee de un sluggable behavior, podemos usarlo para añadir slugs automáticamente a nuestros modelos.

Esto hace que sea sencillo y conveniente proveer de slugs para las páginas de visualización.

Gracias nuevamente a todos los lectores que escribieron para ayudar con errores u otras sugerencias. También, estamos teniendo muchas críticas grandiosas en:

[GoodReads.com Yii 2 For Beginners Reviews](#)

Por favor tómese un momento si puede para compartir sus pensamientos, todos lo apreciarán. Compartamos este asombroso framework con tanta gente como sea posible.

Simplemente como un recordatorio a nuestros lectores que pueden estar leyendo este capítulo como parte de su compra inicial del libro, tienen derecho a actualizaciones gratuitas de Yii 2 Para Principiantes durante la vida del libro. Simplemente inicie sesión en su cuenta de [leanpub.com](#) y obtenga la última versión.

Tengo mucho más material extra planeado, así que espere anuncios por email para sacar provecho de ello.

Tendremos un nuevo capítulo extra sobre Yii 2 Authclient a la brevedad, para que podamos integrar login y registro mediante Facebook.

Gracias nuevamente por continuar compartiendo el viaje de Yii 2, nos vemos pronto.

Capítulo 14: Material Extra Login y Registro con Redes Sociales

Bienvenido a otra ronda de material extra de Yii 2 Para Principiantes. Continuaremos agregando funciones a nuestra plantilla, y el foco de este capítulo será el login y registro a través de las redes sociales Facebook y Github.

La mayoría de los clientes desearán login y registro automáticos con Facebook, es una característica que se ha vuelto tan común que una aplicación web no parece moderna sin ella.

Yii2 - AuthClient

Afortunadamente para nosotros, el equipo de Yii 2 ya ha desarrollado una extensión, Yii2-authclient, que nos brinda todo lo necesario para comenzar. Voy a dejar el enlace a la sección de la Guía de Yii 2 como referencia:

[Yii2-AuthClient](#)

Ya que se trata de una entrada bastante nueva de la guía, por favor tenga en mente que está sujeta a cambios, así que podría no ajustarse exactamente a nuestra implementación. Realizo mi mayor esfuerzo para mantenerme al corriente de los cambios, pero podría pasarlo por alto, así que lo mejor para usted es tener en cuenta la guía así como el libro.

En cualquier instancia, si es diferente, debería ser capaz de resolverlo. Voy a continuar bajo la suposición de que es igual, y de que esta solución funcionará perfectamente.

Yii2-AuthClient en realidad soporta múltiples redes sociales, incluyendo, Facebook, Twitter, Github, Google y más. Para nuestros propósitos, vamos a centrarnos en Facebook y Github, pero podrá usar lo que aprenda aquí con otras redes también.

Bien, esto es lo que vamos a hacer:

- Agregar la extensión Yii2-authclient a nuestro archivo composer.json y ejecutar composer update.
- Crear una tabla auth.
- Crear un modelo Auth.
- Agregar la configuración de Yii2-authclient a nuestro arreglo de componentes en common/- config/main.php
- Crear nuestras aplicaciones auth en cada proveedor.
- Modificar nuestros métodos en el controlador del sitio.

- Agregar un método onAuthSuccess al controlador.
- Agregar enlaces de navegación a registro de Facebook y Github y sincronizar.
- Agregar una característica de sincronización social para que futuras cuentas puedan sincronizarse con las redes sociales.
- Refactorizar para mantenibilidad.

Instalar yii2authclient via Composer

Vaya a su archivo composer.json y agregue la siguiente línea a la sección require:

```
"yiisoft/yii2-authclient": "*",
```

Tenga en cuenta que no necesita la coma si se trata de la última línea. El bloque completo debería verse así en este punto:

```
"require": {  
    "php": ">=5.4.0",  
    "yiisoft/yii2": "*",  
    "yiisoft/yii2-bootstrap": "*",  
    "yiisoft/yii2-swiftmailer": "*",  
    "kartik-v/yii2-social": "dev-master",  
    "yiisoft/yii2-authclient": "*",  
    "fortawesome/fontawesome-free": "4.2.0"  
},
```

Luego, desde su línea de comandos, ejecute:

```
C:\var\www\yii2build>composer update
```

Composer Update

Configuración

El siguiente paso es configurar el componente. Diríjase a common/config/main.php y agregue lo siguiente al arreglo de componentes.

Gist:

[Auth Collection](#)

Del libro:

```
'authClientCollection' => [
    'class' => 'yii\authclient\Collection',
    'clients' => [
        'facebook' => [
            'class' => 'yii\authclient\clients\Facebook',
            'clientId' => 'your client id',
            'clientSecret' => 'your client secret',
        ],
        'github' => [
            'class' => 'yii\authclient\clients\GitHub',
            'clientId' => 'your client id',
            'clientSecret' => 'your client secret',
        ],
        'twitter' => [
            'class' => 'yii\authclient\clients\Twitter',
            'consumerKey' => 'your consumer key',
            'consumerSecret' => 'your consumer secret',
        ],
        'google' => [
            'class' => 'yii\authclient\clients\GoogleOAuth',
            'clientId' => 'your client id',
            'clientSecret' => 'your client secret',
        ],
        'linkedin' => [
            'class' => 'yii\authclient\clients\LinkedIn',
            'clientId' => 'your client id',
            'clientSecret' => 'your client secret',
        ],
    ],
],
],
```

Puede ver que hemos configurado 5 proveedores. Obviamente, reemplace los placeholders por ‘your client id’ y ‘your client secret’ con sus datos reales.

Al momento de escribir esto, Facebook, Linkedin, Google y Github funcionan maravillosamente.

Problema con Twitter

Twitter no regresa la dirección de correo del usuario, vea el siguiente asunto:

Twitter Dev Support

Luego de cuatro años de queja de los desarrolladores, finalmente parecen estar listos para proporcionar el email, pero al momento de escribir esto, y a pesar de sus promesas, aún no lo han hecho. Sin la dirección de correo, tendríamos que crear un proceso separado para que el email sea proporcionado por el usuario, lo que en este punto, no parece lógico, si es que pronto van a poner a disposición el email a través de una api.

Aplicaciones de Proveedores

Ahora necesitamos crear nuestras aplicaciones proveedoras. Como referencia, voy a darle una lista de urls de proveedores para crear las apps:

Facebook:

<https://developers.facebook.com/>

GitHub:

<https://github.com/settings/applications>

Twitter:

<https://apps.twitter.com/>

Google:

<https://console.developers.google.com/project>

LinkedIn:

<https://www.linkedin.com/secure/developer>

Yii2authclient también soporta:

- Microsoft Live
- VKontakte
- Yandex En este punto, no las implementaré en la plantilla. ## App de Facebook Dado que ya contamos con una app de Facebook, comencemos con ella. Simplemente tendremos que agregar una opción de configuración. Asegúrese de haber iniciado sesión en Facebook y diríjase a:

<https://developers.facebook.com/>

Seleccione su app de la lista desplegable My Apps dropdown en el menú de navegación de la cabecera. Luego seleccione settings. Tenemos que asegurarnos de definir los dominios de la App:

You are currently editing a test version of **yii 2 start**

Basic **Advanced** **Migrations**

App ID: [REDACTED] App Secret: [REDACTED] [Show](#)

Display Name: yii2build Namespace: [REDACTED]

App Domains: yii2build.com

Website [Quick Start](#) [X](#)

Site URL: http://www.yii2build.com/

Mobile Site URL: http://www.yii2build.com/

+ Add Platform

Obviamente, he tachado la Id de mi App. Su Id de App Id y su Secret debería estar allí luego de haberlas creado en el capítulo 10. También, asegúrese de agregar su id y secret a la configuración en components. Y eso es todo, deberíamos estar bien. ## App de Github Luego realizaremos una app para Git-

hub. Inicie sesión en Github y dirijase a:

Haga clic en el botón Registrar una nueva aplicación. Luego complete sus detalles de la siguiente for-

Search GitHub

Explore Gist Blog Help

Personal settings

Profile

Developer applications

These are applications you have registered to use the GitHub API.

Personal settings

Profile

Account settings

Emails

Notification center

Billing

SSH keys

Security

Applications

Repositories

Organizations

Applications / Register a new OAuth application

Application name

Yii2Build

Something users will recognize and trust

Homepage URL

http://www.yii2build.com

The full URL to your application homepage

Application description

Demo from Yii 2 For Beginners

This is displayed to all potential users of your application

Authorization callback URL

http://www.yii2build.com

Your application's callback URL. Read our OAuth documentation for more information

Register application

ma:

Una vez que envíe los datos, le regresará su id de cliente y su secret:

Applications / **Yii2Build**

evercode1 owns this application. [Transfer ownership.](#)

0 users

Client ID
'your id'

Client Secret
'your secret'

Revoke all user tokens **Reset client secret**

Application name

Yii2Build

Something users will recognize and trust

Homepage URL

Drag & drop

Github App Id and Secret

Obviamente, su pantalla mostrará su propia id y secret. También, asegúrese de que ha agregado su id y secret en la configuración de componentes. Bien, a esta altura ya debe haber visto como funciona y qué datos debe proporcionar.

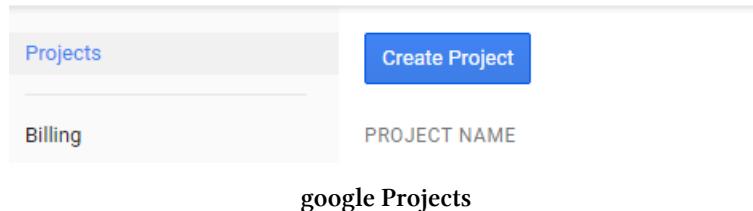
App de Google

Voy a proporcionar capturas de pantalla, pero por favor tenga en cuenta que las cosas pueden cambiar con el tiempo. Si lo que ve en Google es diferente de lo que ve aquí, aún así debería ser suficiente para ponerlo en la dirección correcta. Cuando vaya a Google:

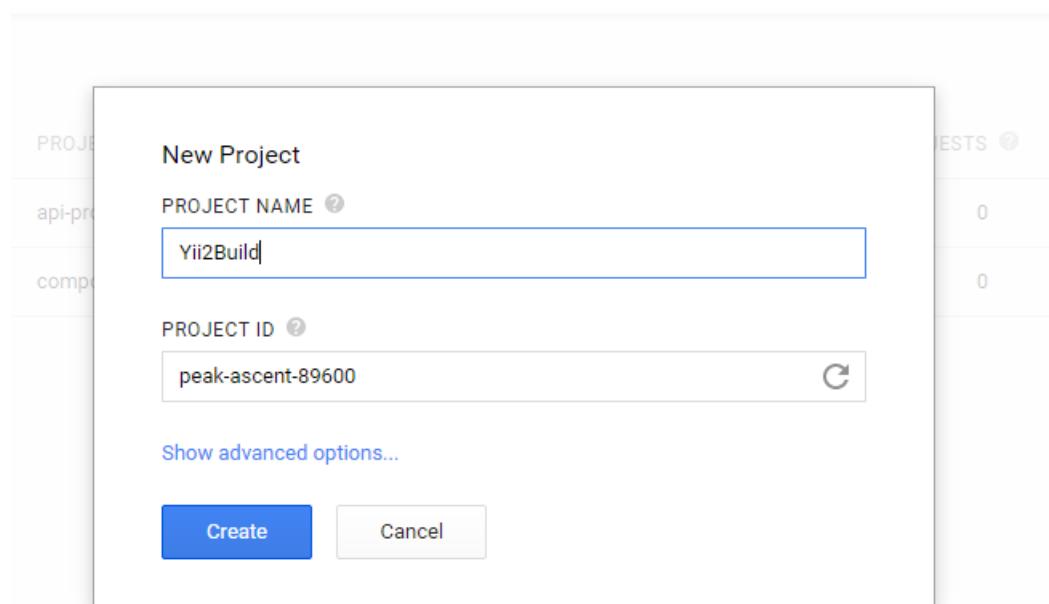
<https://console.developers.google.com/project>

Verá:

Google Developers Console

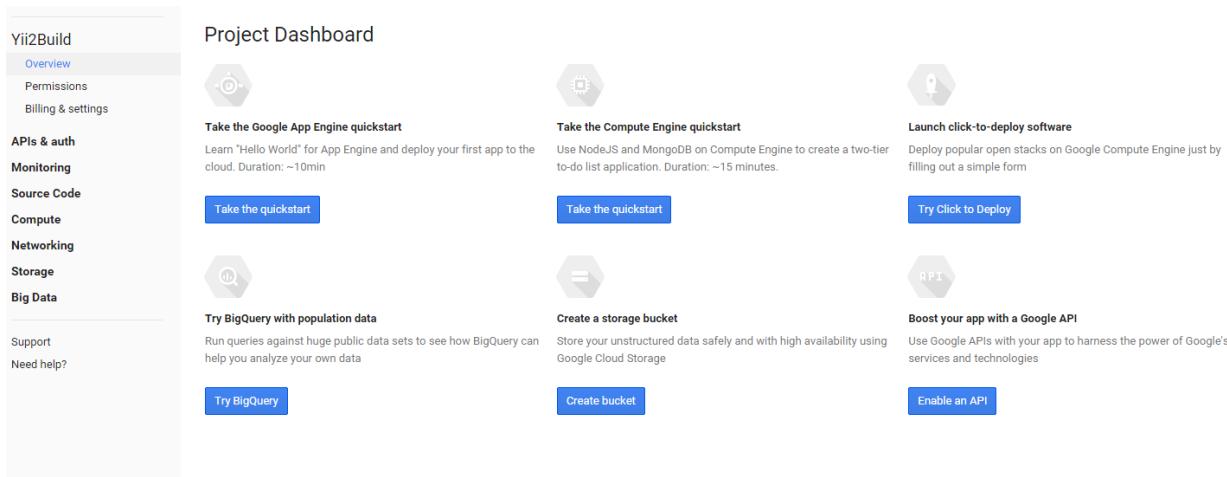


Luego cree su proyecto:



Create Google Project

Llegará a:



Google Project Dash

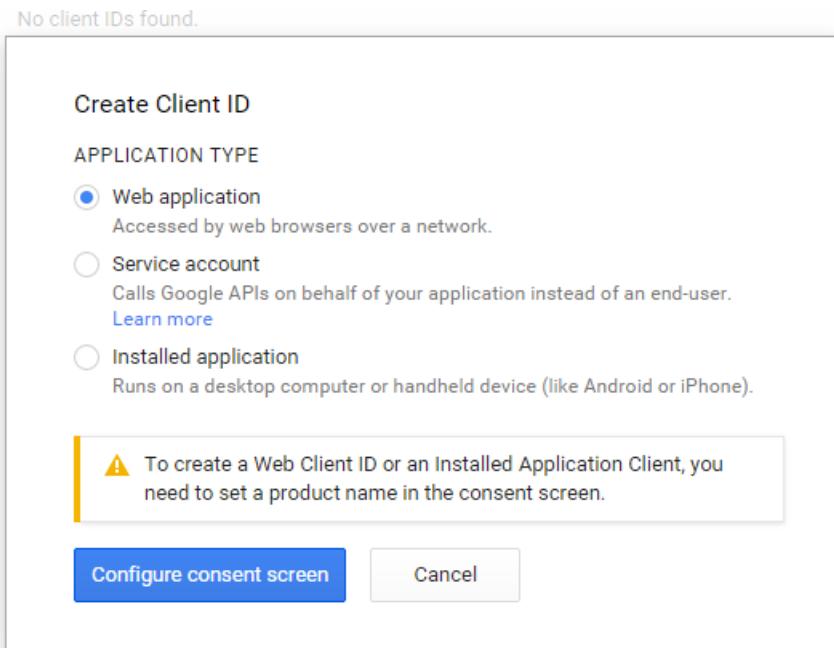
Luego seleccione Credenciales bajo APIs & auth en el menú de la izquierda. Esto lo llevará a:

The screenshot shows the Google Cloud Platform API & auth Credentials page for the 'Yii2Build' project. The left sidebar includes links for Overview, Permissions, Billing & settings, APIs & auth (which is currently selected), and other sections like Monitoring, Source Code, Compute, and Networking. The main content area is divided into sections:

- OAuth**: Describes OAuth 2.0 and has a button to 'Create new Client ID'. To the right, it says 'No client IDs found.'
- Public API access**: Describes Public API access and has a button to 'Create new Key'. To the right, it says 'No keys found.'

Create OAuth Credentials

Haga clic en Crear nueva ID de Cliente, obtendrá:



Configure Consent

< Projects

Yii2Build

- Overview
- Permissions
- Billing & settings

APIs & auth

- APIs
- Credentials
- Consent screen**
- Push

Monitoring

Source Code

Compute

Networking

Storage

Big Data

Support

Need help?

Consent screen

The consent screen will be shown to users whenever you request access to their private data using your client ID.

Note: This screen will be shown for all of your applications registered in this project

EMAIL ADDRESS

PRODUCT NAME

HOME PAGE URL (Optional)

PRODUCT LOGO (Optional) ⓘ

This is how your logo will look to end users.
Max size: 120x120 px

PRIVACY POLICY URL (Optional)

TERMS OF SERVICE URL (Optional)

GOOGLE+ PAGE (Optional) ⓘ

Save **Cancel**

Consent Screen

Continúe a:

Luego llegamos a:

Create Client ID

APPLICATION TYPE

Web application
Accessed by web browsers over a network.

Service account
Calls Google APIs on behalf of your application instead of an end-user.
[Learn more](#)

Installed application
Runs on a desktop computer or handheld device (like Android or iPhone).

AUTHORIZED JAVASCRIPT ORIGINS
Cannot contain a wildcard (`http://*.example.com`) or a path
(`http://example.com/subdir`).

`http://www.yii2build.com`

AUTHORIZED REDIRECT URIS
One URI per line. Needs to have a protocol, no URL fragments, and no relative paths. Can't be a non-private IP Address.

`http://www.yii2build.com/site/auth?authclient=google|`

Create Client ID

Create Client Id

Esto lo llevará a:

The screenshot shows the 'OAuth' section of the Google Cloud Platform API Manager. On the left, there's a sidebar for 'Yii2Build' with options like Overview, Permissions, Billing & settings, APIs & auth (with APIs selected), Credentials (which is highlighted in grey), Consent screen, and Push. The main area has a heading 'OAuth' with a sub-section 'OAuth 2.0 allows users to share specific data with you (for example, contact lists) while keeping their usernames, passwords, and other information private.' Below this is a 'Learn more' link and a blue button 'Create new Client ID'. To the right, there's a form for 'Client ID for web application' with fields for CLIENT ID, EMAIL ADDRESS, CLIENT SECRET, and REDIRECT URIS. Below these fields are 'JAVASCRIPT ORIGINS' and two buttons: 'Edit settings' and 'Reset se'.

Create Client Id

Dejé fuera la id de cliente real, pero usted la verá cuando llegue a esta página. Por favor note que la dirección de correo está entre la Id de Cliente y el Secret, así que no se confunda y tome la id incorrecta.

The screenshot shows the 'APIs' section of the Google Cloud Platform API Manager. The sidebar for 'Yii2Build' shows APIs selected. The main area has a search bar 'Search all 100+ APIs' and a 'Popular APIs' section. This section includes icons for Google Cloud APIs (Compute Engine API, BigQuery API, Cloud Storage API, Cloud Datastore API, Cloud Deployment Manager API, Cloud DNS API, and a 'More' link), Social APIs (Google+ API, Blogger API, Google+ Pages API, and Google+ Domains API), and icons for Monitoring, Source Code, Compute, Networking, Storage, and Big Data.

Luego haga clic en APIs y obtendrá:

Google Developers Console

< Projects

Disable API

Yii2Build

Overview

Permissions

Billing & settings

APIs & auth

APIs

Credentials

Consent screen

Push

Google+ API

OVERVIEW USAGE QUOTAS

The Google+ API enables developers to build...

Learn more ↗

Explore this API ↗

View reports in API Console ↗

Seleccione Google+ Api.. Habilite la API y obtendrá:

Regrese a la pantalla de APIs y habilite Google contacts API, está bajo el encabezado Google Apps APIs. Eso debería ser suficiente en cuanto a Google.

App de LinkedIn

Este proveedor tiene muchos menos pasos que Google. Comience por dirigirse a:

LinkedIn:

<https://www.linkedin.com/secure/developer>

Llegará a:

LinkedIn® Developer Network

List of Applications

Company	Application Name	View API Usage
Yii 2 For Beginners	Yii 2 Start	View API Usage

[Add New Application](#)

[« Back to LinkedIn Developer Network](#)

Linked App Dash

Haga clic en Add New y verá el siguiente formulario:



Linked App Dash

Es un formulario extenso, así que sólo le mostré las partes más importantes. Un par de cosas a tener en cuenta. Puede seleccionar sólo un scope por defecto. Obtendrá un error críptico si selecciona más de uno.

También note que la url de redirección tiene el mismo formato que la de Google, pero obviamente con linkedin como el authclient.

Llegará a la página de éxito. Necesita lo siguiente:

Consumer Key / API Key mapea a clientId

Consumer Secret / Secret Key mapea a clientSecret

Es importante usar clientId y clientSecret en common/config/main.php. Y eso es todo, debería funcionar.

Para tener una idea de como usar el logueo y registro social, mostraré algunas pantallas, seguidas de los Gist con el código relacionado. ## Cambio a la vista Index Comencemos con frontend/view/si-

The screenshot shows the Yii 2 Build homepage. At the top, there is a dark header bar with the text "Yii 2 Build" and a small logo. To the right of the header are links for "About", "FAQs", "Contact", "Signup", and "Login". Below the header, there is a row of social media icons with their respective names: Facebook, GitHub, Twitter, Google, and LinkedIn. In the center, the text "Yii 2 Build" is displayed next to a large black power plug icon. Below this, there is a blue button with the text "f Sign Up or Sign In".

The screenshot shows the "index.php" page of the Yii 2 Build application. At the top, there are two tabs: "Top Features" and "Top Resources". Below the tabs, there is a section with a "comments" button. The main content area is currently empty, indicated by the placeholder text "te/index.php:".

Bien, puede ver que tenemos algunos íconos bonitos para los proveedores. Estos aparecen debido a que usamos AuthChoice::widget: echo yii\authclient\widgetsAuthChoice::widget([‘baseAuthUrl’ => [‘site/auth’], ‘popupMode’ => false,])

El widget llamará tantos íconos como haya configurado en su arreglo components. Ya que incluimos 5, obtenemos esa cantidad en nuestra vista. Si desea que aparezcan menos íconos, remueva los proveedores de la configuración. Aún cuando no todos están funcionando ahora, los mantengo porque los incluiré una vez que funcionen.

También he agregado un botón grande para iniciar sesión o registrarse con Facebook. Obviamente depende de usted hacer que luzca como desea. He agregado tanto el botón de facebook como los widgets para demostrar ambos.

Aquí está lo que hemos modificado en el archivo index.php, reemplazando todo hasta la etiqueta de apertura del widget collapse:

Gist:

[New Index.php code] (<https://gist.github.com/evercode1/f8d5f88f3b7e7a8c9617>)

Del libro:

```
<?php use \yii\bootstrapModal; use kartik\socialFacebookPlugin; use \yii\bootstrapCollapse; use \yii\bootstrapAlert; use yii\helpers\Html; use componentsFaqWidget;  
/* @var $this yii\webView */ $this->title = 'My Yii Application'; ?> <div class="site-index">  
  
<div class="jumbotron">  
  
<?php  
  
    if (Yii::$app->user->isGuest) {  
  
        echo yii\authclient\widgets\AuthChoice::widget([  
            'baseAuthUrl' => ['site/auth'],  
            'popupMode' => false,  
        ]);  
    }  
?>  
  
    <h1>Yii 2 Start <i class="fa fa-plug"></i></h1><br>  
    <?php  
  
    if (!Yii::$app->user->isGuest) {  
  
        echo '<p class="lead">Use this Yii 2 Template to start Projects.</p>';  
    } else {
```

```
echo '<h4>'. Html::a(' <i class="fa fa-facebook"></i> Sign Up or Sign In', ['auth', 'authclient' => 'facebook'], ['class' => 'btn btn-primary ']).'</h4>';

}

?>

</div>
```

Notará que hemos encerrado nuestro widget en una sentencia **if** para comprobar si el usuario es un invitado. Luego tenemos otra sentencia **if** para ver si el usuario es un invitado y si esto es **así**, le mostramos el botón de inicio de sesión y registro con facebook.

Cambios a la vista Login

También necesitamos agregar nuestro widget a nuestras páginas de inicio de sesión y registro. **Así** es como debería verse:
! [New index.php](images/originals/newlogin.png)

Puede ver que todo lo que hicimos fue ubicar el widget estratégicamente. No hay necesidad de revisar el código, pero proporcionaré el gist con el archivo completo:

Gist:

```
[Login](https://gist.github.com/evercode1/2ad0ea265608a873bd2e)
## Signup View Change
```

Luego continuemos con la vista Signup:

```
! [New index.php](images/originals/newsSignup.png)
```

Nuevamente, para un cambio **tan** simple no necesitamos una revisión, pero aquí estás el Gist como referencia:

```
[Nueva vista Signup](https://gist.github.com/evercode1/1c092cd9969dcfb3af71)
```

Lista desplegable Social Sync

También, necesitaremos realizar un cambio en frontend/views/layouts/main.php. Queremos una lista desplegable de proveedores sociales que permitan al usuario sincronizar su cuenta existente con un proveedor social, si **así** lo desea. Se verá así:

```
! [Social Sync](images/originals/socialsync.png)
```

Aquí está el archivo el código completo de frontend/views/layouts/main.php:

Gist:

[New Main](<https://gist.github.com/evercode1/3318f762f0b84dc85145>)

Puede ver en el siguiente bloque que estamos usando font-awesome para nuestros íconos sociales:

```
$menuItems[] = ['label' => 'Social Sync', 'items' => [ ['label' => '<i class="fa fa-facebook"></i> Facebook', 'url' => ['site/auth', 'authclient' => 'facebook']], ['label' => '<i class="fa fa-github"></i> Github', 'url' => ['site/auth', 'authclient' => 'github']], ['label' => '<i class="fa fa-twitter"></i> Twitter', 'url' => ['site/auth', 'authclient' => 'twitter']], ]];
```

Estamos especificando que la 'url' vaya al controlador del sitio, auth action, y que luego le pase un parámetro como authclient='facebook' por ejemplo.

No tenemos una acción auth action, pero nos ocuparemos de ella pronto.

Para que font-awesome trabaje correctamente, tenemos que setear encodeLabels en \ Nav::widget:

```
echo Nav::widget([ 'options' => ['class' => 'navbar-nav navbar-right'], 'items' => $menuItems, 'encodeLabels' => false, ]);
```

Estructura de Datos de Auth

Ahora que nos hemos encargado de todas las cuestiones cosméticas, construiremos la estructura de datos que la soportará. Comenzaremos añadiendo una tabla a nuestra base de datos. La guía nos proporciona algo de SQL con el que trabajar pero no funciona con MySql:

```
CREATE TABLE auth ( id int(11) NOT NULL AUTO_INCREMENT PRIMARY KEY, user_id int(11) NOT NULL, source string(255) NOT NULL, source_id string(255) NOT NULL );
```

```
ALTER TABLE auth ADD CONSTRAINT fk-auth-user_id-user-id FOREIGN KEY user_id REFERENCES user(id);
```

Obviamente, string no es un tipo de datos que MySql reconocerá. Además, usar guiones en la clave **foránea**, así que tuve que usar MySql Workbench para obtener la sintaxis correcta y para solucionar otros problemas. Esto es lo que obtuve:

Gist:

[SQL **for** Auth table](<https://gist.github.com/evercode1/cc14cee611f9e4223527>)

Del libro:

```
CREATE TABLE IF NOT EXISTS yii2build.auth ( id INT(11) UNSIGNED NOT NULL AUTO_INCREMENT, user_id INT(11) UNSIGNED NOT NULL, source VARCHAR(255) NOT NULL, source_id VARCHAR(255) NOT NULL, PRIMARY KEY (id), INDEX fk_auth_user_id_user_id (user_id ASC), CONSTRAINT fk_auth_user1 FOREIGN KEY (user_id) REFERENCES yii2build.user (id) ON DELETE NO ACTION ON UPDATE NO ACTION)
```

Pensé que sería una buena idea tener un ejemplo en SQL al que pudiera referirme en el futuro, si así lo necesito. Si está colaborando con alguien o necesita subir su código en Github o en otro lugar, terminará escribiendo el SQL o creando una migración.

Modelo Auth

Ahora que tenemos la tabla, el paso siguiente es crear un modelo Auth. Vamos a usar Gii:

! [Auth Model](images/originals/authmodel.png)

Vamos a ubicar el modelo en la carpeta common, así que el campo **namespace** es:

common\models

Luego generamos el archivo y eso es todo. Gii incluirá automáticamente la relación que necesitamos:

```
public function getUser() { return $this->hasOne(User::className(), ['id' => 'user_id']); }
```

Bien, ahora necesitamos trabajar en el controlador el sitio.

Método Actions del Controlador del Sitio

Como primera medida necesitamos agregar nuestro método actions.

Gist:

[**New** Actions Method] (<https://gist.github.com/evercode1/79ff69db74e826dbaec7>)

Del libro:

```
public function actions() { return [ 'error' => [ 'class' => 'yii\webErrorAction', ], 'captcha' => [ 'class' => 'yii\captcha\CaptchaAction', 'fixedVerifyCode' => YIL_ENV_TEST ? 'testme' : null, ], 'auth' => [ 'class' => 'yii\authclient\AuthAction', 'successCallback' => [$this, 'onAuthSuccess'], ], ]; }
```

Puede ver que 'auth' especifica la clase un parámetro successCallback, lo que esencialmente les dice a las acciones que cuando la url sea site/auth **use** el método onAuthSuccess.

OnAuthSuccess

Añadiremos ese método al controlador del sitio en breve. Antes observemos el método onAuthSuccess como se encuentra en la guía:

```
public function onAuthSuccess($client) { $attributes = $client->getUserAttributes();  
  
    /** @var Auth $auth */  
    $auth = Auth::find()->where([  
        'source' => $client->getId(),  
        'source_id' => $attributes['id'],  
    ])->one();  
  
    if (Yii::$app->user->isGuest) {  
        if ($auth) { // login  
            $user = $auth->user;  
            Yii::$app->user->login($user);  
        }  
    }  
}
```

```
    } else { // signup
        if (isset($attributes['email'])
            && isset($attributes['username'])
            && User::find()->where([
                'email' => $attributes['email']]])->exists()) {
            Yii::$app->getSession()->setFlash('error', [
                Yii::t('app', "User with the same email as in {client}
                    account already exists but isn't linked to it.
                    Login using email first to link it.",
                ['client' => $client->getTitle()]),
            ]);
        } else {
            $password = Yii::$app->security->generateRandomString(6);
            $user = new User([
                'username' => $attributes['login'],
                'email' => $attributes['email'],
                'password' => $password,
            ]);
            $user->generateAuthKey();
            $user->generatePasswordResetToken();
            $transaction = $user->getDb()->beginTransaction();
            if ($user->save()) {
                $auth = new Auth([
                    'user_id' => $user->id,
                    'source' => $client->getId(),
                    'source_id' => (string)$attributes['id'],
                ]);
                if ($auth->save()) {
                    $transaction->commit();
                    Yii::$app->user->login($user);
                } else {
                    print_r($auth->getErrors());
                }
            } else {
                print_r($user->getErrors());
            }
        }
    }
} else { // user already logged in
    if (!$auth) { // add auth provider
        $auth = new Auth([
            'user_id' => Yii::$app->user->id,
```

```
        'source' => $client->getId(),
        'source_id' => $attributes['id'],
    ]);
$auth->save();
}

}

}

}
```

Si simplemente encuentra:

‘username’ \Rightarrow \$attributes[‘login’],

Y lo reemplaza por:

‘username’ \Rightarrow \$attributes[‘name’],

Eso funcionará hasta cierto punto para Facebook. Puede probar el registro e inicio de sesión básicos, debería funcionar. Pero no se ajusta del todo a nuestros propósitos.

Voy a mostrarle el método onAuthSuccess en detalle, pero hay suficientes cambios en mi versión como para llevarme a pensar que debemos ir directamente a él.

OAuth Success Actualizado

El método onAuthSuccess actualizado.

Gist:

[onAuthSuccess Revision 1](<https://gist.github.com/evercode1/7f5242458d1ab421fcb\6>)

Del libro:

```
public function onAuthSuccess($client) {
```

```
$attributes = $client->getUserAttributes();  
  
//$attributes['email'] = null;  
//var_dump($attributes);  
//die();  
  
// if provider didn't supply email  
  
if (!isset($attributes['email'])){  
  
    return Yii::$app->getSession()->setFlash('error', [  
        Yii::t('app', "Unable to finish, {client} did not  
        provide us with an email. Please check your settings on  
        {client}.", ['client' => $client->getTitle()]),  
    ]);  
}  
  
$source = $client->getId();  
  
//set $username to correct $attribute from each provider  
  
switch ($source){  
  
    case $source == 'facebook' :  
  
        $username = 'name';  
        break;  
  
    case $source == 'github' :  
  
        $username = 'login';  
        break;  
  
    case $source == 'twitter' :  
  
        $username = 'screen_name';  
        break;  
  
    default:  
  
        $username = 'name';
```

```
}

$auth = Auth::find()->where([
    'source' => $source,
    'source_id' => $attributes['id'],
])->one();

if (Yii::$app->user->isGuest) {

    if ($auth) { // login

        $user = $auth->user;
        Yii::$app->user->login($user);

    } else { // signup

        if (isset($attributes['email']))
            && User::find()->where
                ([['email' => $attributes['email']]])->exists()) {
            return Yii::$app->getSession()->setFlash('error', [
                Yii::t('app', "User with the same email as in {client} account
already exists but isn't synced. Login with username and password
and click the {client} sync link to sync accounts.",
                ['client' => $client->getTitle()]),
            ]);
        }

    } else {

        $password = Yii::$app->security->generateRandomString(6);
        $user = new User([
            'username' => $attributes[$username],
            'email' => $attributes['email'],
            'password' => $password,
        ]);
        $user->generateAuthKey();
        //$user->generatePasswordResetToken();
        $transaction = $user->getDb()->beginTransaction();
        if ($user->save()) {
            $auth = new Auth([
                'user_id' => $user->id,
                'source' => $client->getId(),
            ]);
        }
    }
}
```

```
        'source_id' => (string)$attributes['id'],
    ]);

    if ($auth->save()) {
        $transaction->commit();
        Yii::$app->user->login($user);
        MailCall::onMailableAction('signup', 'site');

    } else {
        return Yii::$app->getSession()->setFlash('error', [
            Yii::t('app', "We were unable to complete the
process and sync {client}.",
            ['client' => $client->getTitle()]),
        ]);
    }

} else {

    return Yii::$app->getSession()->setFlash('error', [
        Yii::t('app', "We were unable to complete the process
and sync {client}.",
        ['client' => $client->getTitle()]),
    ]);
}

}

} else { // user already logged in

    if (!$auth && $this->attributes['email'] ==
        Yii::$app->user->identity->email) { // add auth provider

        $auth = new Auth([
            'user_id' => Yii::$app->user->id,
            'source' => $source,
            'source_id' => (string)$attributes['id'],
        ]);

        $auth->save();

        Yii::$app->getSession()->setFlash('success', [
            Yii::t('app', "Your {client} account is successfully
synced.",
            ['client' => $client->getTitle()]),
        ],
    );
}
}
```

```

        ]);

    } else {//emails don't match

        if($attributes['email'] != Yii::$app->user->identity->email){

            Yii::$app->getSession()->setFlash('error', [
                Yii::t('app', "Your {client} account could not be
                synced.", ['client' => $client->getTitle()]),
            ]);

        } else {// account was already synced

            Yii::$app->getSession()->setFlash('success', [
                Yii::t('app', "Your {client} account is already
                synced.", ['client' => $client->getTitle()]),
            ]);

        }

    }

}
}

```

Note que no he agregado aún Google y LinkedIn. Lo haré en la versión **final**.

Escenarios de logueo y registro

Con esta solución, Facebook y Github deberían funcionar en todos los escenarios,\ incluyendo:

1. Registro de usuario cuando no existe el email en la base de datos.

2. El registro de usuario no puede completarse porque el email ya existe en \ la base de datos.

3. El registro de usuario no puede completarse porque el proveedor social n\ o proporciona el email.

4. Inicio se sesión con cuenta sincronizada.

5. Sincronizar cuenta social con un registro de usuario existente en el estado logueado.

Escenario 1 registro de usuario

Esto es para un nuevo registro en nuestra aplicación utilizando un proveedor social. La dirección de email es el identificador único que relaciona todo. Chequeamos si la dirección de email existe en nuestra tabla de usuarios, y si existe, retornamos un mensaje indicando al usuario que primero tiene que iniciar sesión para sincronizar su proveedor social en futuros logueos.

Escenario 2 el email ya está en uso

Cuando el usuario trata de registrarse a través de un proveedor social, comprobamos la existencia de un email en el sistema. Si ya pertenece a un usuario, devolvemos un mensaje indicando al registrante que debe iniciar sesión primero, luego sincronizar su cuenta.

Escenario 3 el proveedor no provee el email

En algunos casos, el proveedor no provee la dirección de email, tal como ocurre con twitter. También, un usuario puede tener una configuración errónea en su cuenta, por ejemplo, Github tiene dos lugares donde configurar el permiso de email; en las secciones perfil e email, lo que puede resultar confuso. El usuario debe asegurarse de asignar el permiso para que su dirección de email esté disponible.

En caso de faltar el email, nuestra aplicación simplemente revuelve un mensaje sugiriendo revisar la configuración en su red social.

Escenario 4 inicio de sesión con proveedor

Este escenario permite a los usuarios loguearse con su proveedor social. El usuario tiene que haber creado ya una cuenta a través de su proveedor social o sincronizado su cuenta existente para que esto funcione.

Escenario 5 sincronizar con una cuenta existente

En este escenario, el usuario ya tiene un registro de usuario, pero no está sincronizado con su proveedor social. Así que en este caso, necesita loguearse y hac

er clic en el enlace Social Sync que creamos en main.php.

Cuando su cuenta ha sido sincronizada con éxito, devolvemos un mensaje indicándolo. Si ya han sincronizado su cuenta y deciden hacer clic en el botón nuevamente, enviamos un mensaje indicando que su cuenta ya se encuentra sincronizada.

Así que ahora que sabemos cómo funciona, podemos regresar el método onAuthSuccess para ver como lo hemos implementado.

Comencemos con la siguiente firma:

```
public function onAuthSuccess($client)
```

Toma un objeto del proveedor auth, que contiene todos los atributos, que podemos acceder de la siguiente manera:

```
$attributes = $client->getUserAttributes();
```

Así que ahora tenemos un arreglo de atributos con el que jugar.

Puede haber notado las siguientes líneas de depuración:

```
// debug stuff // $attributes['email'] = null; // var_dump($this->attributes); // die();
```

Si quiere saber que ocurre cuando 'email' no forma parte de \$attributes, descomente:

```
$attributes['email'] = null;
```

Si quiere ver lo que el proveedor devuelve, descomente:

```
var_dump($this->attributes); die();
```

Sólo asegúrese de que comenta el resto del método o no se pausará y no podrá ver los resultados.

Facebook, por ejemplo, devuelve lo siguiente:

```
array(11){ ["id"]=> string(15) "1111111111111111" ["email"]=> string(17) "some@email.com" ["first_name"]=> string(4) "Bill" ["gender"]=> string(4) "male" ["last_name"]=> string(4) "Keck" ["link"]=> string(60) "https://www.facebook.com/app_scoped_user_id/1111111111111111/" ["locale"]=> string(5) "en_US" ["name"]=> string(9) "Bill Keck" ["timezone"]=> int(-0) ["updated_time"]=> string(24) "2015-02-18T02:05:22+0000" ["verified"]=> bool(true) }
```

Obviamente he cambiado algunos de los valores para no exponer datos personales.

Bien, comenté las líneas de depuración y descomente el resto del método.

Lo primero que tenemos que hacer luego de crear `$attributes` para contener nuestros valores es comprobar si se ha provisto de un email:

```
if (!isset($attributes['email'])){

    return Yii::$app->getSession()->setFlash('error', [
        Yii::t('app', "Unable to finish, {client}
            did not provide us with an email.
            Please check your settings on {client}.",
            ['client' => $client->getTitle()]),
    ]);

}
```

Al usar `Yii::t` method, podemos crear un token de cliente, usando los corchetes, y setear el valor del token, en este caso:

```
['client' => $client->getTitle()]
```

Luego usamos una sentencia `switch` para permitirnos establecer atributos de acuerdo al proveedor, ya que todos tienen nombres diferentes para las cosas:

```
$source = $client->getId();

//set $username to correct $attribute from each provider

switch ($source){

    case $source == 'facebook' :

        $username = 'name';

    break;
```

```
case $source == 'github' :  
  
    $username = 'login';  
  
    break;  
  
case $source == 'twitter' :  
  
    $username = 'screen_name';  
  
    break;  
  
default:  
  
    $username = 'name';  
  
}
```

\$client->getId() nos da el nombre del proveedor, que asignamos a una variable de\\ nominada \$source y luego realizamos el **switch** sobre ella.

También usamos \$username como un nombre de variable que contendrá el valor del a\\ tributo que estamos buscando. Puede ver que con nuestros 3 proveedores listados,\\ tenemos 3 nombres de campo diferentes que convertimos a \$username.

Usamos \$source y \$username más adelante en el código.

Luego intentamos obtener un registro auth para el usuario si éste existe:

```
$auth = Auth::find()->where([
```

```

'source' => $source,
'source_id' => $attributes['id'],
])->one();

```

Si recuerda la estructura de la tabla auth, grabamos una source_id, que es la id\ del registro otorgada por el proveedor social. Así que usando Facebook como eje\ mple, si existe un registro de la tabla auth con una source_id que coincida con \ la id del arreglo \$attributes, entonces hemos encontrado un registro para ese us\ uario.

Luego se intentará loguear al usuario:

```

if(Yii::$app->user->isGuest) {

    if ($auth) { // login

        $user = $auth->user;

        Yii::$app->user->login($user);
    }
}

```

Si el bloque de código anterior ha devuelto exitósamente un registro, logueamos \ al usuario. Si no, vamos a la sentencia **else**. Observemos la primera parte:

```

// signup
if(isset($attributes['email']) && User::find()->where(['email' => $attributes['email']])->exists()) {

    return Yii::$app->getSession()->setFlash('error', [
        Yii::t('app', "User with the same email as in {client} account already exists but isn't synced. Login with username and password and click the {client} sync link to sync accounts.", ['client' => $client->getTitle()])
    ]);
}

```

Realizamos una comprobación para ver si el email ya ha sido usado. Usamos el útil método `exists`, así que téngalo en cuenta en el futuro. Si en efecto existe, devolvemos un mensaje con los tokens apropiados.

Si aún no tenemos el email en nuestra bd, entonces ejecutamos la sentencia `email`, la que creará la cuenta de usuario. Comenzamos generando una contraseña:

```
else {  
  
    $password = Yii::$app->security->generateRandomString(6);  
  
    $user = new User([  
        'username' => $attributes[$username],  
        'email' => $attributes['email'],  
        'password' => $password,  
    ]);  
  
    $user->generateAuthKey();  
  
    // $user->generatePasswordResetToken();
```

Luego como puede ver creamos una nueva instancia de `User` y asignamos los valores que necesitamos a las columnas para el nuevo registro de usuario del arreglo `$attributes` y la contraseña recientemente generada. Puede ver que he comentado:

```
// $user->generatePasswordResetToken();
```

Simplemente mantengo esa línea como referencia ya que estaba en el método de la guía. Es completamente innecesario para nuestra implementación, así que puede moverla si lo desea.

Ya que estamos grabando registros en dos tablas, uno para auth y otro para user, usaremos una transacción. Comenzamos así:

```
$transaction = $user->getDb()->beginTransaction();
```

Luego continuamos guardando el usuario y creando un nuevo registro en la tabla auth:

```

if ($user->save()) {
    $auth = new Auth([
        'user_id' => $user->id,
        'source' => $client->getId(),
        'source_id' => (string)$attributes['id'],
    ]);
    if ($auth->save()) {
        $transaction->commit();
        Yii::$app->user->login($user);
        MailCall::onMailableAction('signup', 'site');
    }
}

```

Puede ver que si se han guardado tanto \$user como \$auth, realizamos el commit de la transacción. Luego logueamos al usuario y le enviamos un autoresponder.

Baso este uso de una transacción en el ejemplo original provisto en la guía. Normalmente, esperaría ver un bloque **try/catch** y una sentencia rollback, pero parece no ser necesario aquí.

Sí experimenté para ver si lograba romperla y salvar una de las partes de la transacción, pero por lo que puedo ver en realidad fuerza a que la transacción se realice como está escrita.

Revisé la guía:

[Yii 2 Database] (<http://www.yiiframework.com/doc-2.0/guide-db-dao.html>)

Pero no pude encontrar ninguna referencia. Así que, ya que funciona, encontré algo que proviene de otra parte de la guía y que me permite escribir mensajes de error con tokens, por lo que voy a usarla.

Esto nos lleva a la siguiente parte. Si la transacción no se completa, enviamos un mensaje:

```
else {
```

```

        return Yii::$app->getSession()->setFlash('error', [
            Yii::t('app', "We were unable to complete the
process and sync {client}." ,
            ['client' => $client->getTitle()]),
        ]);
    }
} else {

    return Yii::$app->getSession()->setFlash('error', [
        Yii::t('app', "We were unable to complete the process
and sync {client}." ,
        ['client' => $client->getTitle()]),
    ]);
}
}
}

```

Usted pensaría que hemos concluido, pero por supuesto no es el caso. Ahora tenemos que ejecutar la sentencia `else` cuando comprobamos si el usuario es o no un invitado. Así que, si no es un invitado, ya se encuentra logueado y continuamos:

```

else { // user already logged in

    if (!$auth && $this->attributes['email'] ==
        Yii::$app->user->identity->email) { // add auth provider

        $auth = new Auth([
            'user_id' => Yii::$app->user->id,
            'source' => $source,
            'source_id' => (string)$attributes['id'],
        ]);

        $auth->save();

        Yii::$app->getSession()->setFlash('success', [
            Yii::t('app', "Your {client} account
is successfully synced." ,
            ['client' => $client->getTitle()]),
        ]);
    }
}
}
}

```

Ya que anteriormente hemos tratado de devolver una instancia de `$auth` donde `source_id` y `$attributes['id']` coinciden, podemos usar una sentencia `if` para establecer la condición para crear un nuevo registro en auth:

```
if (!$auth && $this->attributes['email'] == Yii::$app->user->identity->email)
```

Si `evalúa` a verdadero, creamos un nuevo registro en auth. Note que los valores para `email` tienen que coincidir.

Notará el uso de `(string)` aquí:

```
'source_id' => (string)$attributes['id'],
```

Ello es porque necesitamos decirle que necesitamos un `string`, que es lo que precisamos para pasar la validación. Recuerde, que establecimos esa columna como un `varchar`, y esto nos da flexibilidad para acomodar los diferentes formatos de los proveedores.

Luego finalmente, tenemos:

```
else {//emails don't match
```

```
if($attributes['email'] != Yii::$app->user->identity->email){

    Yii::$app->getSession()->setFlash('error', [
        Yii::t('app', "Your {client} account could not be synced.",
            ['client' => $client->getTitle()]),
    ]);

} else {// account was already synced

    Yii::$app->getSession()->setFlash('success', [
        Yii::t('app', "Your {client} account is already synced.",
            ['client' => $client->getTitle()]),
    ]);

}
```

Estamos cubiertos en el caso de que los emails no coincidan o si la cuenta ya ha\ sido sincronizada.

Esta es una solución completamente funcional que debería ser capaz de probar con\ éxito. Pero hay un problema, que puede no ser obvio al principio. Aunque de man\ era sutil, estamos cayendo en una duplicación de código.

Note que hemos usado:

```
MailCall::onMailableAction('signup', 'site');
```

En la firma, el parámetro 'signup' designa la acción, pero esa no es la acción c\ orrecta. Así que tendríamos o bien que generar otro mensaje de estado para esta \ acción, lo que no tiene sentido, o usar onMailableAction en una forma no previst\ a.

La solución es suficientemente sencilla, y con el objeto de tener todo listo y p\ oder realizar pruebas, está bien. Pero para mantener una aplicación a lo largo d\ el tiempo, podría ser un verdadero problema. Si realiza cambios a cómo y cuándo \ quiere llamar al mismo autoresponder, tendrá que realizar modificaciones en 2 lu\ gares. Esa es una práctica horrible si puede evitarse.

Y el autoresponder no es la única duplicación de código. Login y signup deberían\ ser manejados por sus respectivas acciones del controlador. Al crear métodos in\ dependientes de esas acciones, deja la puerta abierta para errores de codificación y omisiones. Si quiere realizar cosas tales como agregar datos de sesión al p\ roceso de login del usuario o bloquear cierta dirección ip para el registro, ten\ drá nuevamente que hacerlo en 2 lugares distintos. Eso no está bien.

¿No sería mejor mover las partes relativas a registro y logueo de onAuthSuccess \ a sus respectivas acciones? Mi opinión es que sí deberíamos. Hará que actionLogi\ n y actionSignup sean más complicados, pero no hay forma de evitarlo si vamos a \ tener todo en un sólo método. Este enfoque es enteramente opcional sin embargo, \ y es libre de elegir lo que desea.

Refactorización para mantenibilidad y extensibilidad

Decidí que ya que esto es para mi plantilla, y planeo usarla en muchos proyectos\ , realizaría un poco de refactorización tanto para aumentar la legibilidad como \ para evitar duplicación de código. En el proceso, terminé creando 7 métodos nuev\ os:

```
* createUser()
* createAuth($user)
* findExistingAuth()
* emailPresent()
* matchEmail()
* formatProviderResponse($source)
* emailAlreadyInUse()
```

Algunos de estos métodos son simplemente para legibilidad cosmética y algunos para reducir la duplicación. Quería que fuera fácil regresar en el futuro y entender el código.

No se preocupe, vamos a trabajar en ello de manera incremental, de manera que lo entienda completamente.

Para mover los valores entre métodos, necesitamos establecerlos como propiedades de clase.

Nuevas propiedades de clase

Lo realizamos declarando nuevas propiedades vacías al comienzo de la clase. Ubique lo siguiente dentro de la clase SiteController:

```
private $attributes = []; private $username; private $source; private $socialUser;
```

Luego modificamos onAuthSuccess para establecer y usar esas propiedades.

Gist:

[onAuthSuccess Refactor 2](<https://gist.github.com/evercode1/b7b5674f41cc52690166>)

Del libro:

```
public function onAuthSuccess($client) {
```

```
$this->attributes = $client->getUserAttributes();  
  
//$this->attributes['email'] = null; for example in book.  
//var_dump($this->attributes);  
// die();  
  
// if provider didn't supply email  
  
if (!isset($this->attributes['email'])){  
  
    return Yii::$app->getSession()->setFlash('error', [  
        Yii::t('app', "Unable to finish, {client} did not provide  
        us with an email. Please check your settings on {client}.",  
        ['client' => $client->getTitle()]),  
    ]);  
}  
  
$this->source = $client->getId();  
  
//set $username to correct $attribute from each provider  
  
switch ($this->source){  
  
    case $this->source == 'facebook' :  
  
        $this->username = 'name';  
        break;  
  
    case $this->source == 'github' :  
  
        $this->username = 'login';  
        break;  
  
    case $this->source == 'twitter' :  
  
        $this->username = 'screen_name';  
        break;  
  
    default:  
  
        $this->username = 'name';
```

```
}

$auth = Auth::find()->where([
    'source' => $this->source,
    'source_id' => $this->attributes['id'],
])->one();

if (Yii::$app->user->isGuest) {

    if ($auth) { // login

        $user = $auth->user;
        Yii::$app->user->login($user);

    } else { // signup

        if (isset($this->attributes['email']))
            && User::find()->where
            ([ 'email' => $this->attributes['email']] )->exists()) {

            return Yii::$app->getSession()->setFlash('error', [
                Yii::t('app', "User with the same email as in {client}
account already exists but isn't synced. Login with
username and password and click the {client} sync link
to sync accounts.", [ 'client' => $client->getTitle()]),
            ]);

        } else {
            $password = Yii::$app->security->generateRandomString(6);
            $user = new User([
                'username' => $this->attributes[$this->username],
                'email' => $this->attributes['email'],
                'password' => $password,
            ]);
            $user->generateAuthKey();

            // $user->generatePasswordResetToken();

            $transaction = $user->getDb()->beginTransaction();

            if ($user->save()) {


```

```
$auth = new Auth([
    'user_id' => $user->id,
    'source' => $client->getId(),
    'source_id' => (string)$this->attributes['id'],
]);

if ($auth->save()) {
    $transaction->commit();
    Yii::$app->user->login($user);
    MailCall::onMailableAction('signup', 'site');

} else {
    return Yii::$app->getSession()->setFlash('error', [
        Yii::t('app', "We were unable to complete the process
        and sync {client}.", ['client' => $client->getTitle()]),
    ]);
}

} else {

    return Yii::$app->getSession()->setFlash('error', [
        Yii::t('app', "We were unable to complete the process
        and sync {client}.", ['client' => $client->getTitle()]),
    ]);
}

}

}

} // user already logged in

if (!$auth && $this->attributes['email'] ==
Yii::$app->user->identity->email) { // add auth provider

$auth = new Auth([
    'user_id' => Yii::$app->user->id,
    'source' => $this->source,
    'source_id' => (string)$this->attributes['id'],
]);

$auth->save();
```

```

        Yii::$app->getSession()->setFlash('success', [
            Yii::t('app', "Your {client} account is
            successfully synced.",
            ['client' => $client->getTitle()]),
        ]);

    } else { //emails don't match

        if($this->attributes['email'] != Yii::$app->user->identity->email){

            Yii::$app->getSession()->setFlash('error', [
                Yii::t('app', "Your {client} account could not be synced.",
                ['client' => $client->getTitle()]),
            ]);

        } else { // account was already synced

            Yii::$app->getSession()->setFlash('success', [
                Yii::t('app', "Your {client} account is already synced.",
                ['client' => $client->getTitle()]),
            ]);

        }

    }

}

```

Ahora puede probarlo y debería funcionar correctamente. Realizar modificaciones \ incrementales hace que introducir los cambios que necesitamos sea más sencillo.

Nuevos métodos auxiliares

Continuemos agregando 7 métodos auxiliares nuevos. Estos métodos recidirán en la\ clase SiteController, así que siempre tendremos acceso a ellos a través de \$thi\ s.

Aquí está el método createUser:

Gist:

[createUser](<https://gist.github.com/evercode1/7ed72d703a514917f6ea>)

Del libro:

```
private function createUser() {  
  
    $password = Yii::$app->security->generateRandomString(6);  
    $user = new User([  
        'username' => $this->attributes[$this->username],  
        'email' => $this->attributes['email'],  
        'password' => $password,  
    ]);  
  
    $user->generateAuthKey();  
  
    return $user;  
  
}
```

Puede ver que simplemente cortamos el código del método onAuthSuccess y lo convirtimos en una función privada. Generamos una contraseña, creamos una nueva instancia de User, y generamos una clave de auth, y luego retornamos.

Ahora podemos llamarla **así**:

```
$user = $this->createUser();
```

Podrá ver esto en acción en breve y con suerte le gustará lo que hace por la legibilidad del código.

Luego tenemos createAuth:

gist:

[createAuth](<https://gist.github.com/evercode1/60273120a8a4cc957b40>)

Del libro:

```
private function createAuth($user) {
```

```
$auth = new Auth([
    'user_id' => $user->id,
    'source' => $this->source,
    'source_id' => (string)$this->attributes['id'],
]);
return $auth;

}
```

Nuevamente simplemente lo cortamos de onAuthSuccess y lo convertimos en una función privada. Ya que necesitamos asignar 'user_id' => \$user->id, necesitamos pasárselle una instancia de User en la firma del método.

Luego tenemos findExistingAuth:

Gist:

[findExistingAuth](<https://gist.github.com/evercode1/228d5cd5808ce73bbaa9>)

Del libro:

```
private function findExistingAuth() {

    $auth = Auth::find()->where([
        'source' => $this->source,
        'source_id' => $this->attributes['id'],
    ])->one();

    return $auth;

}
```

Este simplemente encuentra el registro auth, si existe. Si no existe, devuelve null.

Luego está emailPresent:

Gist:

[emailPresent](<https://gist.github.com/evercode1/e36c113f77cc59c747e7>)

Del libro:

```
private function emailPresent() {  
  
    return isset($this->attributes['email']) ? true : false;  
}
```

Este simplemente nos dice si el proveedor ha enviado un email.

Luego tenemos matchEmail:

Gist:

[matchEmail](<https://gist.github.com/evercode1/311e8c159493db4d0b51>)

Del libro:

```
private function matchEmail() {  
  
    return $this->attributes['email'] == Yii::$app->user->identity->email ? true : false;  
}
```

Obviamente debería ser una sola línea. Este método simplemente nos informa si el email devuelto por el proveedor coincide con el del usuario actual de la aplicación.

Luego está formatProviderResponse:

Gist:

[formatProviderResponse](<https://gist.github.com/evercode1/3de1c7f198274a7c6a0b>)

```
private function formatProviderResponse($source) {
```

```
switch ($source){

    case $source == 'facebook' :

        $this->username = 'name';
        break;

    case $source == 'github' :

        $this->username = 'login';
        break;

    case $source == 'twitter' :

        $this->username = 'screen_name';
        break;

    case $source == 'linkedin' :

        $this->username = 'fullName';

        $fullName = $this->attributes['first_name'] . ' ' .
        $this->attributes['last_name'];

        $this->attributes['fullName'] = $fullName;

        break;

    case $source == 'google' :

        $this->username = 'displayName';

        $emails = $this->attributes['emails'];

        foreach ($emails as $email){

            foreach ($email as $k => $v) {

                if ($k == 'value'){

                    $this->attributes['email'] = $v;
                }
            }
        }
    }
}
```

```
        }

    }

    break;

default:

$this->username = 'name';

}

}
```

Este método formatea la respuesta del proveedor. A medida que nuestra aplicación\ se vuelve más sofisticada y deseemos trabajar con más atributos, podremos forma\ tearlos aquí. También puede ver lo fácil que resulta agregar un nuevo proveedor,\ simplemente añada una nueva sentencia **case** como lo hice para Google y LinkedIn.

Por supuesto que ambos proveedores hicieron que fuera más complicado formatear l\ a respuesta. En el caso de Linkedin, tuve que crear un nombre a partir de nombre\ y apellido y lo llamé fullName. Luego lo asigné a la propiedad de clase, para \ poder usarlo más tarde.

En el caso de Google, aparentemente pueden devolvernos más de un email **así** que n\ o tienen un valor denominado email. Esto significa que tuve que extraerlo del ar\ reglo multidimensional usando ciclos **foreach** anidados.

Luego tenemos emailAlreadyInUse:

Gist:

[emailAlreadyInUse](<https://gist.github.com/evercode1/1a03a8cee8ae79bc0d94>)

Del libro:

```
private function emailAlreadyInUse() {
```

```

return User::find()
    ->where(['email' => $this->attributes['email']])
    ->exists() ? true : false;

}

```

Simplemente nos indica si el email devuelto por el proveedor ya se encuentra en uso por un usuario registrado, lo usamos antes de sincronizar para determinar si el usuario debería primero iniciar sesión o no.

Ahora a medida que avancemos, usaremos esos métodos y verá cómo hacen que el código sea más fácil de seguir.

Método OnAuthSuccess

Comencemos con el método onAuthSuccess, que has sido modificado drásticamente:

Gist:

[onAuthSuccess] (<https://gist.github.com/evercode1/c64623626012d0c47c8d>)

Del libro:

```

public function onAuthSuccess($client) {

    $this->attributes = $client->getUserAttributes();

    $this->source = $client->getId();

    $this->formatProviderResponse($this->source);

    if (!(!$this->emailPresent())){

        return Yii::$app->getSession()->setFlash('error', [
            Yii::t('app', "Unable to finish, {source} did not provide us
            with an email. Please check your settings on {source}."),
            ['source' => $this->source],
        ]);
    }

    $existingAuth = $this->findExistingAuth();

```

```
if (Yii::$app->user->isGuest) {

    if ($existingAuth) { // login steps

        $this->socialUser = $existingAuth->user;

        $viaSocial = true;

        $this->actionLogin($viaSocial);

    } else { // signup steps

        $viaSocial = true;

        $this->actionSignup($viaSocial);

    }

} else { // user already logged in, require email match

    if (!$existingAuth && $this->matchEmail()) { // add auth provider

        $auth = $this->createAuth(Yii::$app->user);

        $auth->save();

        Yii::$app->getSession()->setFlash('success', [
            Yii::t('app', "Your {source} account is successfully synced.", [
                ['source' => $this->source]
            ]),
        ]);

    } else { //emails don't match

        if (!$this->matchEmail()){

            Yii::$app->getSession()->setFlash('error', [
                Yii::t('app', "Your {source} account could not be synced.", [
                    ['source' => $this->source]
                ]),
            ]);

        }

    }

}
```

```
        } else { // account was already synced

        Yii::$app->getSession()->setFlash( 'success' , [
            Yii::t('app', "Your {source} account is already synced." ,
                [ 'source' => $this->source]),
        ]);
    }

}

}
```

Si, aún es inmenso, pero es mucho más legible. Observemos la primera parte:

```
public function onAuthSuccess($client) {  
  
    $this->attributes = $client->getUserAttributes();  
  
    $this->source = $client->getId();  
  
    $this->formatProviderResponse($this->source);  
  
    if (!$this->emailPresent()) {  
  
        return Yii::$app->getSession()->setFlash('error', [  
            Yii::t('app', "Unable to finish, {source} did not provide us  
with an email. Please check your settings on {source}.",  
            ['source' => $this->source]),  
        ]);  
    }  
}
```

Así que, igual que antes, estamos extrayendo UserAttributes del objeto \$client que ha sido enviado, y lo asignamos a nuestra propiedad de clases \$this->attribut\ es, que es un arreglo que contiene los valores.

Luego establecemos la propiedad \$source a partir de \$client->getId(). Esto es \ seguido por el método formatProviderResponse, que buscará en \$source y establece\ rá el valor de \$this->username en consecuencia.

Luego viene la sentencia **if** para ver si el email se encuentra presente:

```
if (!$this->emailPresent()) {
```

Eso es mucho más fácil de entender, **así** que cuando regrese en un año, ahorraré \ tiempo al ser capaz de comprenderlo más rápidamente.

Si esa sentencia **evalúa** a verdadero y no tenemos un email, devolvemos un mnesaje\ informadndo al usuario que necesitan chequear la configuración de su proveedor.

Ya que \$this->source está disponible como una propiedad de clase y ya tiene un v\ alor, la usamos en lugar de \$client->getTitle().

Bien, continuemos:

```
$existingAuth = $this->findExistingAuth();
```

Esto devolverá **null** si no existe un registro en auth o devolverá el registro cor\ recto si existe. Vamos a usarlo en una comprobación en nuestro siguiente bloque:

```
if (Yii::$app->user->isGuest) {  
  
    if ($existingAuth) { // login steps  
  
        $this->socialUser = $existingAuth->user;  
  
        $viaSocial = true;  
  
        $this->actionLogin($viaSocial);  
  
    } else { // signup steps
```

So we check to see **if** the user is a guest, then test to see **if** the `$existingAuth`\ evaluates **true**, **if** so, we want to login, **if not**, we want to signup.

Ya que `$existingAuth` es una instancia de Auth, y ya que tenemos una relación con\ User en el modelo Auth, podemos acceder al usuario con:

```
$existingAuth->user;
```

Así que asignamos el usuario a la propiedad de clase `$socialUser`, por lo que ten\ dremos acceso a ella en `actionLogin`:

```
$this->socialUser = $existingAuth->user;
```

Luego establecemos `$viaSocial` a verdadero, que como verá en un momento, tiene se\ ntido en el método `actionLogin`, que convocamos:

```
$this->actionLogin($viaSocial);
```

Puede ver que estamos pasando `$viaSocial` como parámetro.

Acción Login

Ahora observemos el método `actionLogin`:

Gist:

```
[actionLogin](https://gist.github.com/evercode1/1b7ec48483916af2ab26)
```

Del libro:

```
public function actionLogin($viaSocial = false) { if (!Yii::$app->user->isGuest) { return $this->goHome(); }
```

```

if($viaSocial){

    Yii::$app->user->login($this->socialUser);

} else {

    $model = new LoginForm();
    if ($model->load(Yii::$app->request->post()) && $model->login()) {
        return $this->goBack();
    } else {
        return $this->render('login', [
            'model' => $model,
        ]);
    }
}

}

```

En realidad esto no cambió mucho. Obviamente hemos asignado `$viaSocial=null` como un valor por defecto en la firma. Esto nos permite no enviar un valor para `$viaSocial` cuando iniciamos sesión a través del formulario de login normal.

Luego lo primero después de comprobar si el usuario ya se encuentra logueado, es verificar `$viaSocial`:

```

if ($viaSocial){

    Yii::$app->user->login($this->socialUser);

} else

```

Así que si `$viaSocial` es verdadero, loguemos al usuario al pasar `$this->socialUser` al método `login` de la clase `user`, que no es el mismo que `actionLogin` en el controlador. Lo hemos cubierto ante en el libro.

Ya hemos asignado `$this->socialUser` a una propiedad de clase anteriormente en el método `onAuthSuccess`, así que está disponible aquí.

Todo lo que se encuentra luego de la sentencia `else` es exactamente lo que había antes, así que puede ver que nuestro método `login` no ha cambiado demasiado drásticamente.

icamente.

El método actionSignup, sin embargo, es bastante más grande.

Acción Signup

Gist:

[actionSignup](<https://gist.github.com/evercode1/cd58b05548f1eb4664ed>)

Del libro:

```
public function actionSignup($viaSocial=false) { if ($viaSocial){

    if ($this->emailPresent() && $this->emailAlreadyInUse()) {

        return Yii::$app->getSession()->setFlash('error', [
            Yii::t('app', "User with the same email as in {source} account
already exists but isn't synced. Login with username and
password and click the {source} sync link to sync accounts.",
            ['source' => $this->source]),
        ]);
    } else {

        $user = $this->createUser();

        $transaction = $user->getDb()->beginTransaction();

        if ($user->save()) {

            $auth = $this->createAuth($user);

            if ($auth->save()) {

                $transaction->commit();

                Yii::$app->user->login($user);

                MailCall::onMailableAction('signup', 'site');

            } else {

        }
    }
}
```

```
        return Yii::$app->getSession()->setFlash('error', [
            Yii::t('app', "We were unable to complete the process
and sync {source}.", ['source' => $this->source]),
        ]);

    }
} else {

    if( User::find()->where(['username' => $this->username])){

        return Yii::$app->getSession()->setFlash('error', [
            Yii::t('app', "Username already taken, please signup
through the site Signup form and use a different
username, thanks."),
        ]);

    } else {

        return Yii::$app->getSession()->setFlash('error', [
            Yii::t('app', "We were unable to complete the process
and sync {source}.", ['source' => $this->source]),
        ]);

    }

}

}

} else {

$model = new SignupForm();

if ($model->load(Yii::$app->request->post())) {

    if ($user = $model->signup()) {

        if (Yii::$app->getUser()->login($user)) {

            MailCall::onMailableAction('signup', 'site');

        }

    }

}

}
```

```

        return $this->goHome();
    }
}

return $this->render('signup', [
    'model' => $model,
]);
}

}

```

Breaking `this` down into smaller chunks makes it easier to digest. Let's take the first part:

```

public function actionSignup($viaSocial=false) { if ($viaSocial){

    if ($this->emailPresent() && $this->emailAlreadyInUse()) {

        return Yii::$app->getSession()->setFlash('error', [
            Yii::t('app', "User with the same email as in {source} account
already exists but isn't synced. Login with username and
password and click the {source} sync link to sync accounts.",
            ['source' => $this->source]),
        ]);
    } else

```

So we **use** the same technique of setting `$viaSocial` to **false as a default** so that we can **use** the regular signup form **if** we wish to without having to pass a value in. Then we test **for**:

```
if ($this->emailPresent() && $this->emailAlreadyInUse()) {
```

Puede ver que es fácil de entender. Si **evalúa** a verdadero, devolvemos un mensaje, sino:

```
$user = $this->createUser();
```

```
$transaction = $user->getDb()->beginTransaction();

if ($user->save()) {

    $auth = $this->createAuth($user);

    if ($auth->save()) {

        $transaction->commit();

        Yii::$app->user->login($user);

        MailCall::onMailableAction('signup', 'site');

    } else {

        return Yii::$app->getSession()->setFlash('error', [
            Yii::t('app', "We were unable to complete the process
            and sync {source}.", ['source' => $this->source]),
        ]);
    }
} else {

    if( User::find()->where(['username' => $this->username])){

        return Yii::$app->getSession()->setFlash('error', [
            Yii::t('app', "Username already taken, please signup
            through the site Signup form and use a different
            username, thanks."),
        ]);
    } else {

        return Yii::$app->getSession()->setFlash('error', [
            Yii::t('app', "We were unable to complete the process
            and sync {source}.", ['source' => $this->source]),
        ]);
    }
}
}
```

Note arriba, que realizamos una comparación para ver si el registro falla debido\ a que el nombre de usuario ya se encuentra en uso, ya que la validación requiere que sea único. Si ya se encuentra en uso, el usuario ve un mensaje con un formato agradable indicándole qué debe hacer.

Comenzamos la sentencia **else** creando un usuario mediante nuestro útil método `createUser`:

```
$user = $this->createUser();
```

Luego comenzamos nuestra transacción:

```
$transaction = $user->getDb()->beginTransaction();
```

```
if ($user->save()) {  
  
    $auth = $this->createAuth($user);  
  
    if ($auth->save()) {  
  
        $transaction->commit();  
    } else {  
        $transaction->rollBack();  
    }  
}
```

Puede ver que usamos nuestro método `createAuth` al que le enviamos `$user`, así que el código aquí es un poco más conciso de lo que sería de otra manera.

Continuamos con 2 sentencias **else** que devuelven un mensaje si algo ocurre y la transacción no se completa.

Eso nos lleva a la sentencia **else final** del método, que simplemente es todo lo que teníamos antes de realizar el cambio, así que podemos procesar la entrada del formulario o presentarlo.

Ya que estamos en la acción `signup` por ejemplo, puede ver que ahora el siguiente llamado tiene sentido:

```
MailCall::onMailableAction('signup', 'site');
```

Ese fue mucho trabajo para un pequeño efecto. Sin embargo, imagine que también está realizando otras cosas:

```
// save user data to session // log login time and ip address to db  
MailCall::onMailableAction('signup', 'site');
```

Esos son simplemente un par de ejemplos como comentarios, pero capta la idea. A medida que nuestra aplicación crezca, deseará tener un único método centralizado\ al que agregar esos requisitos.

Bien, ese fue un gran desvío, pero necesario. Ahora estamos listos para regresar\ a onAuthSuccess. Si no nos estamos registrando o iniciando sesión, entonces ya\ estamos logueados:

```
else { // user already logged in, require email match
```

```
    if (!$existingAuth && $this->matchEmail()) { // add auth provider

        $auth = $this->createAuth(Yii::$app->user);

        $auth->save();

        Yii::$app->getSession()->setFlash('success', [
            Yii::t('app', "Your {source} account is successfully synced.",
                ['source' => $this->source]),
        ]);
    }
}
```

Comprobamos si no hay un `$existingAuth` y si el email coincide. Si todo está bien\ , entonces `createAuth`, pasándole el usuario actualmente logueado y grabamos. Lue\ go enviamos un bonito mensaje de confirmación.

Si algo no va bien, tenemos la sentencia `else`:

```
else { //emails don't match
```

```
    if(!$this->matchEmail()){

        Yii::$app->getSession()->setFlash('error', [
            Yii::t('app', "Your {source} account could not be synced.",
                ['source' => $this->source]),
        ]);

    } else { // account was already synced

        Yii::$app->getSession()->setFlash('success', [
            Yii::t('app', "Your {source} account is already synced.",

```

```
[ 'source' => $this->source]),
]);
}

}
```

Así que ahora estamos manejando los casos en los que los emails no coinciden o la cuenta ya se encuentra sincronizada. Algunas personas simplemente aman apretar un botón para ver qué ocurre, así que tenemos que ocuparnos de ese escenario.

Ahora todo debería funcionar y ser estable. Si revisa el código, encontrará que es mucho más legible, y por lo tanto, debería ser más mantenible.

Un efecto negativo de todo esto es que nuestro controlador site tiene mucho más código que el que solía tener. La inclusión de social auth requirió de 8 nuevos métodos y la reescritura de dos métodos existentes.

Para mi gusto, esto es demasiado para un solo controlador, si puede evitarse. Así que decidí mover actionIndex, actionContact y actionAbout a un nuevo controlador, que voy a llamar PagesController.

Any additional **static** pages, such as terms of service, privacy, etc. will be controlled by the PagesController.

Controlador Pages

No voy a cubrir cada detalle en esta modificación de la plantilla porque usted ya sabe cómo hacer la mayoría de las cosas. Sin embargo haré una lista de pasos, junto con algunos detalles clave, para que no se pierda de nada y también proveeré de Gists al final del capítulo.

Pasos a implementar:

1. Cree el controlador pages con las acciones index, about y contact.
2. Incluya lo siguiente en el controlador:

namespace frontend\controllers;

use Yii; use frontend\models\ContactForm; use yii\filters\AccessControl;

3. Mueva las vistas index, about y contact del directorio site en views al directorio pages.

4. Cambie el enlace del botón de facebook para apuntar a ‘site/auth’ .

5. Configure el captcha correctamente. Necesitará poner el siguiente método behaviors en el controlador pages:

```
public function behaviors() { return [ 'access' => [ 'class' => AccessControl::className(), 'only' => ['captcha']], 'rules' => [ [ 'actions' => ['captcha'], 'allow' => true, 'roles' => ['?', '@'], ], , ], ]; }
```

También, especifique la acción del captcha en el modelo ContactForm en el método rules:

```
[‘verifyCode’, ‘captcha’, ‘captchaAction’ => ‘pages/captcha’],
```

Y también especifíquelo en el widget del formulario:

```
<?= $form->field($model, ‘verifyCode’)-> widget(Captcha::className(), [ ‘captchaAction’ => ‘pages/captcha’, ‘template’ => ‘<div class=”row”> <div class=”col-lg-3”>{image}</div> <div class=”col-lg-6”>{input}</div></div>’, ]) ?>
```

6. Configure **default** controller/action **for** application because the **new** site controller will have the following actions when the change is done:

```
* actions
* login
* logout
* signup
* RequestPasswordReset
* ResetPassword
```

Adicionalmente, tiene los métodos siguientes:

```
* behaviors
* onAuthSuccess
* createUser
* createAuth
* findExistingAuth
* emailPresent
* matchEmail
* formatProviderResponse
* emailAlreadyInUse
```

Obviamente, si removemos index del controlador del sitio, la aplicación necesita\ rá saber dónde enviar a los usuarios. Tenemos algunas opciones. Necesitamos asignar lo siguiente:

‘defaultRoute’ => ‘pages/index’,

Podemos poner eso ya sea en common/config/main.php o en frontend/config/main.php\ . Para mi plantilla, elegí ubicarlo en frontend/config/main, de esta manera:

...

‘bootstrap’ => [‘log’], ‘controllerNamespace’ => ‘frontend\controllers’, ‘defaultRoute’ => ‘pages/index’, ‘components’ => [‘user’ => [

...

Simplemente agréguelo entre controllerNamespace y components.

La aplicación del backend tiene requisitos diferentes, ya que estamos forzando u\ n nivel de acceso de administrador y no hay páginas estáticas. Así que voy a dejarlo como está.

Entonces la pregunta es, ¿queremos incluir auth social en el backend? Este es u\ n problema de seguridad. Significa que si deja abierta una computadora o dispositivo y está logueado en Facebook por ejemplo, cualquiera que acceda a la computadora será capaz de acceder al área de administración. Esto podría tener efectos\ devastadores si la persona equivocada obtiene acceso.

Así que tiene que sopesar conveniencia vs el potencial riesgo de seguridad. En m\ i caso, elegí no implementar auth social en el backend. Esto está sujeto a camb\

io en el futuro por supuesto, pero **así** es como lo hago por ahora.

Bien, **así** que para referencia, y en caso de que lo necesite para depurar, voy a \ proveer los Gists para los nuevos controladores site y pages.

Gist:

[Site Controller](<https://gist.github.com/evercode1/bfa5dbd6ffaf0d609719>)

Gist:

[Pages Controller](<https://gist.github.com/evercode1/8b01004f6eec91739e3d>)

Resumen

Hemos implementado inicio de sesión y registro con un solo clic mediante Facebook, cubriendo cada escenario en el que pudimos pensar. Usamos la extensión authclient de Yii 2, el widget authclient, y realizamos cambios significativos al controlador del sitio.

Era importante para nosotros asegurarnos de que actionLogin y actionSignup en el\ controlador del sitio pudieran ser mantenidos de manera apropiada, **así** que tuvi\ mos que integrar nuestros auth sociales también en esos métodos.

Luego finalmente movimos el resto de las acciones del controlador, como about, contacto, privacy, etc. a un nuevo controlador, pages. Nuestro controlador pages \ se ocupará de todas las páginas del sitio que no tienen sus propios controladores, incluyendo index.

Decidimos que en nuestra aplicación, no realizaríamos los cambios al backend. Es\ ta es una elección personal y usted puede elegir hacerlo si lo desea.

Una vez más me gustaría agradecer a todos los lectores por sus contribuciones y \ sugerencias, ayuda en encontrar errores de tipeo y de código, y por supuesto sus\ críticas positivas y recomendaciones.

Por favor siéntase en libertad de contactarme a través de leanpub.com utilizando\ el enlace:

[Email Bill Keck](https://leanpub.com/yii2forbeginners/email_author/new)

Gracias nuevamente por apoyar el libro.

```
{bump-link-number}  
  
{leanpub-filename="capitulo15.txt"}  
  
# Chapter 15: Images and File Uploads
```

Welcome back to another bonus chapter. I know it's kind of crazy **if** you are reading the chapters successively to constantly welcome you back, but keep in mind \ that I'm creating these additions **and** publishing them after the core book has been finished, one chapter at a **time**.

I want to keep going in **this** style because it feels more like the actual journey\ that I'm taking with Yii 2 **and** in building the template, **sort** of a blog style a\ pproach. I hope you enjoy it too.

Ok, so let's jump into the material **for this** chapter, which is all about uploadi\ ng **and** managing images. The scenario that we are going to work with ties direct\ ly to the template, but you can **use** the skills you learn in the chapter to cover\ a wide **range** of photo **or file** management, including user content.

We don't cover user content directly, however, because it is **not** specific to the\ template. So instead, we are going to develop a MarketingImage model that will\ allow us to manage our marketing images.

We will **use** our marketing images to populate a carousel widget on the frontend i\ ndex page. A carousel on the index page is such a common feature in sites these\ days that I thought it made a lot of sense to **include** one in the template.

If for some reason the client doesn't want a carousel, then it will be easy to r\ emove it, **and** that is because **as** widget, it will simply be one line of code.

In addition to having the carousel display the images, we will also have full co\ ntrol over creating, updating, **and** deleting the files, both **as** model instances a\ nd **as** actual files on the server in our admin area.

And with the weight attribute, they will be able to control the order of the ima\ ges, regardless of what order they are created in.

This will enable you to give full control of the marketing images over to the cl\ ient, **and** they can finesse their carousel exactly how they want it.

```
## The Uploads Folder
```

Let's start our work by creating an uploads folder(**use** lowercase), located within\\n our backend web folder.

```
## Marketing Image Table
```

Let's start by defining our marketing_image table.

```
! [Marketing Images Table](images/originals/marketingimagetable.png)
```

```
## Marketing Image SQL
```

Here is the SQL

Gist:

```
[Marketing Image SQL](https://gist.github.com/evercode1/0a96bd8a51f3af4503a5)
```

From book:

```
CREATE TABLE IF NOT EXISTS yii2build.marketing_image (id INT(10) UNSIGNED NOT NULL AUTO_INCREMENT, marketing_image_path VARCHAR(45) CHARACTER SET 'utf8' COLLATE 'utf8_unicode_ci' NOT NULL, marketing_image_name VARCHAR(45) CHARACTER SET 'utf8' COLLATE 'utf8_unicode_ci' NOT NULL, marketing_image_caption VARCHAR(100) CHARACTER SET 'utf8' COLLATE 'utf8_unicode_ci' NULL DEFAULT NULL, marketing_image_is_featured TINYINT(1) NOT NULL DEFAULT 0, marketing_image_is_active TINYINT(1) NOT NULL DEFAULT 0, marketing_image_weight INT(11) NOT NULL DEFAULT 100, status_id SMALLINT(6) NOT NULL DEFAULT 1, created_at DATETIME NOT NULL, updated_at DATETIME NOT NULL, PRIMARY KEY (id), INDEX fk_marketing_image_status1_idx (status_id ASC), CONSTRAINT fk_marketing_image_status1 FOREIGN KEY (status_id) REFERENCES yii2build.status (id) ON DELETE NO ACTION ON UPDATE NO ACTION)
```

Note that we're sticking with the convention of singular **for** the table name. You can also see that we have a foreign **key**, status_id pointing at id on the status table. Because there are so few records in the status table, I didn't bother making id unsigned, so that's why status_id is **not** unsigned **as** well.

A couple of other notes about the data structure. We are using marketing_image_is_active, set **as** a boolean, to determine **if** the image should be the active item in the carousel. The marketing_image_caption will be displayed in the carousel.

We are using status_id to determine the status of the image because the status table holds the values active, pending, **and** retired. So **this** gives us a chance to reuse existing data structure **and** avoid data duplication.

This data structure is **for** our template, but you are absolutely free to change it **if** you have other requirements. **If** you want more options to designate different types of images, feel free to **do** so, but personally, I would **do** so after working through the rest of the chapter, so you know how it is supposed to work first.

Marketing Image Model

The **next** step is to build our model using Gii. We're going to locate **this** model in the backend/models folder, since working with the images will be an admin's job.

Obviously we're setting the **namespace** to:

backend\models;

By **this** point, you should be familiar enough with Gii **for** me **not** to have to repeat all the instructions. So I will just assume you have correctly created the MarketingImage model.

Next we need to create the CRUD. Since we already have our search folder in backend\models, we don't need to create it.

So go ahead **and** create the CRUD with Gii.

Don't you just love **this** workflow? Every **time** I **use** it I'm reminded about how much I enjoy working with Yii 2.

```
## Modify MarketingImage Model
```

Ok, so now that we got our basic model/crud setup, we need to modify our model.

Gist:

[Modified MarketingImage Model] (<https://gist.github.com/evercode1/587124919cf73a\6c4083>)

From book:

```
<?php

namespace backend\models;

use Yii; use yii\db\ActiveRecord; use yii\dbExpression; use yii\helpers\ArrayHelper; use yii\behaviors\TimestampBehavior

/** * This is the model class for table “marketing_image”. * * @property string $id * @property string $marketing_image_path * @property string $marketing_image_name * @property integer $marketing_image_is_featured * @property integer $marketing_image_is_active * @property integer $status_id * @property string $created_at * @property string $updated_at * * @property Status $status */ class MarketingImage extends \yii\db\ActiveRecord {

    public $file;

    /**
     * @inheritdoc
     */

    public static function tableName()
    {
        return 'marketing_image';
    }

    public function behaviors()
    {
        return [
            'timestamp' => [
                'class' => TimeStampBehavior::className(),
                'attributes' => [

```

ActiveRecord::EVENT_BEFORE_INSERT => ['created_at', 'updated_at'], ActiveRecord::EVENT_BEFORE_UPDATE => ['updated_at'],

```
        ],
        'value' => new Expression('NOW()'),
    ],
];
}

/**
 * @inheritDoc
 */

public function rules()
{
    return [
        [['marketing_image_path', 'marketing_image_name',
            'marketing_image_weight', 'file'], 'required'],

        ['marketing_image_weight', 'default', 'value' => 100 ],
        ['marketing_image_is_featured', 'default', 'value' => 0 ],
        ['marketing_image_is_active', 'default', 'value' => 0 ],

        [['marketing_image_is_featured'], 'in',
            'range'=>array_keys($this->getMarketingImageIsFeaturedList())],
        [['marketing_image_is_active'], 'in',
            'range'=>array_keys($this->getMarketingImageIsActiveList())],

        [['marketing_image_name', 'marketing_image_path'], 'trim'],
        [['marketing_image_caption'], 'string', 'max' => 100],
        [['marketing_image_is_featured', 'marketing_image_is_active',
            'marketing_image_weight', 'status_id'], 'integer'],
        [['file'], 'file', 'extensions' => ['png', 'jpg', 'gif'],
            'maxSize' => 1024*1024],
        [['marketing_image_path', 'marketing_image_name'],
            'string', 'max' => 45]
    ];
}

public function beforeValidate()
{
    $this->marketing_image_name = preg_replace('/\s+/', '',
        $this->marketing_image_name);
}
```

```
$this->marketing_image_path = preg_replace('/\s+/', '',
$this->marketing_image_path);

return parent::beforeValidate();
}

/**
 * @inheritDoc
 */

public function attributeLabels()
{
    return [
        'id' => 'ID',
        'marketing_image_path' => 'Marketing Image Path',
        'marketing_image_name' => 'Marketing Image Name',
        'marketing_image_caption' => 'Caption',
        'marketing_image_is_featured' => 'Marketing Image Is Featured',
        'marketing_image_is_active' => 'Marketing Image Is Active',
        'marketing_image_weight' => 'Marketing Image Weight',
        'status_id' => 'Status ID',
        'created_at' => 'Created At',
        'updated_at' => 'Updated At',
        'file' => 'Marketing Image'
    ];
}

public static function getMarketingImageIsFeaturedList() { return $droptions = [0 => "no", 1 => "yes"]; }

public static function getMarketingImageIsActiveList() { return $droptions = [0 => "no", 1 => "yes"]; }

/**
 * @return \yii\db\ActiveQuery
 */

public function getStatus()
{
    return $this->hasOne(Status::className(), ['id' => 'status_id']);
}

/**
```

```

* * get status name
*
*/

public function getStatusName()
{
    return $this->status ? $this->status->status_name : '- no status -';
}

/**
* get list of statuses for dropdown
*/

public static function getStatusList()
{
    $droptions = Status::find()->asArray()->all();
    return ArrayHelper::map($droptions, 'id', 'status_name');
}

}

```

We're going to step **this** rather quickly **and** just talk about what's different from the boiler plate. Let's start with the **use** statements:

use Yii; use yii\db\ActiveRecord; use yii\db\Expression; use yii\helpers\ArrayHelper; use yii\behaviors\TimestampBehavior;

We pulled in the necessary classes to support TimestampBehavior **and** the ArrayHelper for our dropdown lists.

Next we added a **public** property:

```
public $file;
```

The reason we have to add it like **this** is because the model can't pull it from the table via reflection like it does all the other properties. In **this case**, **\$file** will hold the value of the **file**, **not** a db table value.

Next we added our timestamp behavior in the behaviors method:

```
public function behaviors() { return [ 'timestamp' => [ 'class' => TimeStampBehavior::className(),  

    'attributes' => [  

        ActiveRecord::EVENT_BEFORE_INSERT => ['created_at', 'updated_at'], ActiveRecord::EVENT_-  

        BEFORE_UPDATE => ['updated_at'],
```

```

        ],
        'value' => new Expression('NOW()'),
    ],
];
}

}

```

We've done **this** a number of times, so **not** much to say there.

Now we move onto the rules:

```

public function rules() { return [
    [['marketing_image_path', 'marketing_image_name', 'marketing_image_weight', 'file'], 'required'],
    ['marketing_image_weight', 'default', 'value' => 100],
    ['marketing_image_is_featured', 'default', 'value' => 0],
    ['marketing_image_is_active', 'default', 'value' => 0],
    [
        [['marketing_image_is_featured'], 'in',
            'range' => array_keys($this->getMarketingImageIsFeaturedList())],
        [['marketing_image_is_active'], 'in',
            'range' => array_keys($this->getMarketingImageIsActiveList())],
        [['marketing_image_name', 'marketing_image_path'], 'trim'],
        [['marketing_image_caption'], 'string', 'max' => 100],
        [['marketing_image_is_featured', 'marketing_image_is_active',
            'marketing_image_weight', 'status_id'], 'integer'],
        [['file'], 'file', 'extensions' => ['png', 'jpg', 'gif'],
            'maxSize' => 1024*1024],
        [['marketing_image_path', 'marketing_image_name'],
            'string', 'max' => 45]
    ];
}

```

We are using a couple of validators that we haven't used before. the 'trim' validator gets rid of white space, but **not** space in between words. We will write a beforeValidation method **for** that, since we want to strictly enforce filenames.

You can also see that we added a validation rule **for file**:

```
[['file'], 'file', 'extensions' => ['png', 'jpg', 'gif'], 'maxSize' => 1024*1024],
```

This sets the allowable types and max size of the file, very simple stuff.

Unfortunately, when I tried to save a photo the first time, I got a nasty error \ about PHP fileInfo not being enabled.

```
## PHP FileInfo
```

For whatever reason, my PHP ini file had it commented out, so I had to get rid o\ f the semicolon:

```
! [PHP FileInfo](images/originals/fileinfoph.png)
```

You can see it in the highlighted line above. If you do need to make this change, also remember to restart Apache so the change takes effect.

Ok, moving on. Let's look at our beforeValidate method:

```
public function beforeValidate() { $this->marketing_image_name = preg_replace('/\s+/', '', $this->marketing_image_name);

$this->marketing_image_path =
preg_replace('/\s+/', '', $this->marketing_image_path);

return parent::beforeValidate();

}
```

We're doing formatting here to remove spaces from the marketing_image_path and t\ he marketing_image_name. On the controller, we will do the same for the actual \ file that's being saved to the server. That way we don't end up with filenames \ with spaces in them.

Obviously we're just using a preg_replace to replace a regular expression:

```
\s+/
```

That gets rid of the spaces between words.

Next we add a label for file and getStatusName in our attribute labels method:

'file' => 'Marketing Image', 'statusName' => Yii::t('app', 'Status'),

That will show nicely on the form, when we modify it to accept a `file` upload. Also note, we `use` the magic name `for` `getStatusName`.

Next we have a couple of methods that format the values `for` the dropdown lists we will need in our form:

```
public static function getMarketingImageIsFeaturedList() {
```

```
    return $droptions = [0 => "no", 1 => "yes"];
```

```
}
```

```
public static function getMarketingImageIsActiveList() {
```

```
    return $droptions = [0 => "no", 1 => "yes"];
```

```
}
```

And finally, we added 2 `new` status relationships, so we can `use` status `as` a dropdown `list as` well:

```
public function getStatusName() { return $this->status ? $this->status->status_name : '- no status -'; }
```

```
/**  
 * get list of statuses for dropdown  
 */
```

```
public static function getStatusList() { $droptions = Status::find()->asArray()->all(); return ArrayHelper::map($droptions, 'id', 'status_name'); }
```

We did `not` need to add `getStatus` because that was autogenerated `for` us by Gii due to the foreign Key definition.

Modify MarketingImage Search Model

Ok, so now we need to modify our search model. There's nothing `new` in `this` file, we are just using the other search models `as` an example. That said, I find when I'm doing `this` that I always seem to trip over `this` part, so after giving you the `file`, I will point out some sticking points.

Gist:

[MarketingImage Search Model](<https://gist.github.com/evercode1/9cf8b777780e83ec\89ac>)

From book:

```
<?php

namespace backend\models\search;

use Yii; use yii\baseModel; use yii\dataActiveDataProvider; use backend\modelsMarketingImage;

class MarketingImageSearch extends MarketingImage {
    public $statusName; /** * @inheritDoc */

    public function rules()
    {
        return [
            [['id', 'marketing_image_is_featured',
                'marketing_image_is_active', 'status_id'], 'integer'],
            [['marketing_image_path', 'marketing_image_name',
                'marketing_image_caption', 'marketing_image_weight',
                'created_at', 'statusName',
                'updated_at'], 'safe'],
        ];
    }

    /**
     * @inheritDoc
     */

    public function scenarios()
    {
        // bypass scenarios() implementation in the parent class
        return Model::scenarios();
    }

    /**
     * Creates data provider instance with search query applied
     *
     * @param array $params
     *
     * @return ActiveDataProvider
     */
}
```

```
*/  
  
public function search($params)  
{  
    $query = MarketingImage::find();  
  
    $dataProvider = new ActiveDataProvider([  
        'query' => $query,  
    ]);  
  
    $dataProvider->setSort([  
        'attributes' => [  
            'id',  
            'marketing_image_name',  
            'marketing_image_path',  
            'marketing_image_caption',  
            'marketing_image_is_featured',  
            'marketing_image_is_active',  
            'marketing_image_weight',  
            'statusName' => [  
                'asc' => ['status.status_name' => SORT_ASC],  
                'desc' => ['status.status_name' => SORT_DESC],  
                'label' => 'Status'  
            ],  
        ]  
    ]);  
  
    if (!$this->load($params) && $this->validate()) {  
  
        $query->joinWith(['status']);  
  
        return $dataProvider;  
    }  
  
    $this->addSearchParameter($query, 'id');  
    $this->addSearchParameter($query, 'marketing_image_name', true);  
    $this->addSearchParameter($query, 'marketing_image_path', true);  
    $this->addSearchParameter($query, 'marketing_image_caption', true);  
    $this->addSearchParameter($query, 'marketing_image_is_featured');
```

```
$this->addSearchParameter($query, 'marketing_image_is_active');
$this->addSearchParameter($query, 'marketing_image_weight');
$this->addSearchParameter($query, 'status_id');

// filter by gender name

$query->joinWith(['status' => function ($q) {
    $q->andFilterWhere(['=' , 'status.status_name', $this->statusName]);
}]);

return $dataProvider;

}

protected function addSearchParameter($query, $attribute, $partialMatch = false) { if (($pos = strpos($attribute, '.')) !== false) { $modelAttribute = substr($attribute, $pos + 1); } else { $modelAttribute = $attribute; }

$value = $this->$modelAttribute;
if (trim($value) === '') {
    return;
}

/*
* The following line is additionally added for right aliasing
* of columns so filtering happen correctly in the self join
*/
$attribute = "marketing_image.$attribute";

if ($partialMatch) {
    $query->andWhere(['like', $attribute, $value]);
} else {
    $query->andWhere([$attribute => $value]);
}

}

}
```

So the sticking points are **as** follows:

- * Make sure to set a property **for** the magic call to relation **public \$statusName**
- * Make sure to set a rule **for \$statusName**
- * Make sure **\$attribute = “marketing_image.\$attribute”;** is set correctly
- * Make sure the correct attributes have the **true** option on addSearchParameter
- * Don't expect **sort** on status to work until we modify gridView in index

Hopefully those tips will save you **time**.

Modify Index View

Now we'll work on the views. We will start with index **and** finish with _form, so\\ we have a perfect transition into the **file** upload process **and** the controller.

Here is the modification **for** index.php:

Gist:

[Index.php](<https://gist.github.com/evercode1/aaf5b36f044284ac0aa2>)

From book:

```
<?php
use yii\helpers\Html; use yii\grid\GridView; use \yii\bootstrap\Collapse;
/* @var $this yii\web\View // @var $searchModel backend\models\searchMarketingImageSearch /
@var $dataProvider yii\data\ActiveDataProvider */
$this->title = 'Marketing Images'; $this->params['breadcrumbs'][] = $this->title; ?> <div class="marketing-image-index">

<h1><?= Html::encode($this->title) ?></h1>
<?php echo Collapse::widget([
    'encodeLabels' => false,
    'items' => [
        [
            'label' => 'Marketing Images',
            'content' => <div>
                <?php echo $grid = \backend\widgets\MarketingImageGrid::widget(['dataProvider' => $dataProvider]); ?>
            </div>
        ],
        [
            'label' => 'Marketing Images',
            'content' => <div>
                <?php echo $grid = \backend\widgets\MarketingImageGrid::widget(['dataProvider' => $dataProvider]); ?>
            </div>
        ]
    ]
])>
```

```
// equivalent to the above
[
    'label' => '<i class="fa fa-caret-square-o-down"></i> Search',
    'content' => $this->render('_search', ['model' => $searchModel]) ,
    // open its content by default
    //'contentOptions' => ['class' => 'in']
],
// encodeLabels => false,
]);
?>

<p>
<?= Html::a('Create Marketing Image', ['create'],
['class' => 'btn btn-success']) ?>
</p>

<?= GridView::widget([
    'dataProvider' => $dataProvider,
    'filterModel' => $searchModel,
    'columns' => [
        ['class' => 'yii\grid\SerialColumn'],
        'id',
        'marketing_image_path',
        'marketing_image_name',
        'marketing_image_caption',
        ['attribute'=>'marketing_image_is_featured', 'format'=>'boolean'],
        ['attribute'=>'marketing_image_is_active', 'format'=>'boolean'],
        'marketing_image_weight',
        'statusName',
        // 'created_at',
        // 'updated_at',
        ['class' => 'yii\grid\ActionColumn'],
    ],
]); ?>

</div>
```

So by now **this** should look familiar. We put the search partial inside a collapse widget. One thing **new** there is that we set:

'encodeLabels' ⇒ false,

This allows us to put a font-awesome icon in the label:

'label' ⇒ '<i class="fa fa-caret-square-o-down"></i> Search',

This adds a nice **touch** to the UI. So I went around the application to all the places where I could make that change to be consistent. Here is the **list**:

```
* user index
* profile index
* faq index
* faq category Index
```

The other index pages in the backend don't have the search partial visible. However **if** you wish to change that, you can see how easy it would be.

Modify View

Ok, moving on. The only other thing to note in **this** view is that we are using \$statusName **as** one of the attributes in the GridView widget **and this** gives us the eager-loaded sortable attribute.

Let's look at the modified view **file**:

Gist:

[view.php](<https://gist.github.com/evercode1/8996775ad0ef566a9449>)

From book:

```
<?php
use yii\helpers\Html; use yii\widgetsDetailView;
/* @var $this yii\web\View // @var $model backend\models\MarketingImage */
$this->title = $model->id;
$this->params['breadcrumbs'][] = ['label' => 'Marketing Images', 'url' => ['index']];
$this->params['breadcrumbs'][] = $this->title; ?> <div class="marketing-image-view">
```

```
<p>
    <?= Html::a('Update', ['update', 'id' => $model->id],
                ['class' => 'btn btn-primary']) ?>
    <?= Html::a('Delete', ['delete', 'id' => $model->id], [
                'class' => 'btn btn-danger',
                'data' => [
                    'confirm' => 'Are you sure you want to delete this item?',
                    'method' => 'post',
                ],
            ]) ?>
</p>

<h1><?= Html::encode($model->marketing_image_name) ?></h1>

<br>
<div> <?php

echo Html::img('/'. $model->marketing_image_path . '?' . 'time=' . time() ,
               ['width' => '600px']);

?>

</div>

<br>

<?= DetailView::widget([
    'model' => $model,
    'attributes' => [
        'id',
        'marketing_image_caption',
        'marketing_image_path',
        //marketing_image_name',
        ['attribute' => 'marketing_image_is_featured',
         'format' => 'boolean'],
        ['attribute' => 'marketing_image_is_active',
         'format' => 'boolean'],
        'marketing_image_weight',
        'status.status_name',
        'created_at',
        'updated_at',
    ],
]) ?>
```

```
</div>
```

A couple of minor changes in the DetailView widget **and** in the H1. Notice the use of:

```
echo Html::img('/'. $model->marketing_image_path . '?'. 'time=' . time() , ['width' => '600px']);
```

We are using the `Html::img` helper to provide the path, then appending a get variable `?time=the-current-time`. The reason we are appending the get variable is to prevent caching, so we always see the correct image.

Sometimes where you update the image, you might see the old image **if** you have it cached in your browser. **Finally**, we are enforcing a width to the image, so that we can have all the images set at a certain size.

For our carousel, we might need a specific size, **and** I will come back to **this if** that is the **case**.

Obviously, our view can't display an image yet because we haven't uploaded any, but we will be doing that shortly.

Modify Update View

The only change we are making to the update.php view is that we are also echoing the image there **for** reference.

Gist:

[update.php](<https://gist.github.com/evercode1/bd3bf0bfaf9ca5fa011b>)

From book:

```
<?php
use yii\helpers\Html;
/* @var $this yii\web\View // @var $model backend\models\MarketingImage */
$this->title = 'Update Marketing Image: ' . ' ' . $model->id; $this->params['breadcrumbs'][] = ['label' => 'Marketing Images', 'url' => ['index']]; $this->params['breadcrumbs'][] = ['label' => $model->id, 'url' => ['view', 'id' => $model->id]]; $this->params['breadcrumbs'][] = 'Update'; ?> <div class="marketing-image-update">
```

```
<h1><?= Html::encode($this->title) ?></h1>

<br>
<div>
<?php
echo Html::img('/'. $model->marketing_image_path,
    ['width' => '600px']);

?>

</div>

<br>

<?= $this->render('_form', [
    'model' => $model,
]) ?>

</div>
```

There is no change to the create view.

Modify _search Partial

Gist:

[_search view](<https://gist.github.com/evercode1/3522eae100973125e3be>)

From book:

```
<?php
use yii\helpers\Html; use yii\widgets\ActiveForm;
/* @var $this yii\web\View // @var $model backend\models\searchMarketingImageSearch // @var
$form yii\widgets\ActiveForm */ ?>
<div class="marketing-image-search">
```

```
<?php $form = ActiveForm::begin([
    'action' => ['index'],
    'method' => 'get',
]); ?>

<?= $form->field($model, 'id') ?>

<?= $form->field($model, 'marketing_image_path') ?>

<?= $form->field($model, 'marketing_image_name') ?>

<?= $form->field($model, 'marketing_image_caption') ?>

<?= $form->field($model, 'marketing_image_is_featured')
    ->dropDownList($model->marketingImageIsFeaturedList,
        ['prompt' => 'Please Choose One']); ?>

<?= $form->field($model, 'marketing_image_is_active')
    ->dropDownList($model->marketingImageIsActiveList,
        ['prompt' => 'Please Choose One']); ?>

<?= $form->field($model, 'status_id')->dropDownList($model->statusList,
    ['prompt' => 'Please Choose One' ]); ?>

<?php // echo $form->field($model, 'created_at') ?>

<?php // echo $form->field($model, 'updated_at') ?>

<div class="form-group">
    <?= Html::submitButton('Search', ['class' => 'btn btn-primary']) ?>
    <?= Html::resetButton('Reset', ['class' => 'btn btn-default']) ?>
</div>

<?php ActiveForm::end(); ?>

</div>
```

You can see we are using model methods to populate the dropdown lists. We did not **include** marketing_image_weight on the search form, but obviously you can **if** you want to.

```
## Modify _form Partial
```

Gist:

[_form Partial](<https://gist.github.com/evercode1/2ad1587f393008504903>)

From book:

```
use yii\widgets\ActiveForm; use yii\helpers\Html;  
/* @var $this yii\web\View // @var $model backend\models\MarketingImage // @var $form  
yii\widgets\ActiveForm */ ?>  
<div class="marketing-image-form">  
  
<?php $form = ActiveForm::begin(['options'=>  
    ['enctype' => 'multipart/form-data']])); ?>  
  
<?= $form->field($model, 'marketing_image_name')  
->textInput(['maxlength' => 45]) ?>  
  
<?= $form->field($model, 'marketing_image_caption') ?>  
  
<?= $form->field($model, 'marketing_image_is_featured')  
->dropDownList($model->marketingImageIsFeaturedList,  
['prompt' => 'Please Choose One']);?>  
  
<?= $form->field($model, 'marketing_image_is_active')  
->dropDownList($model->marketingImageIsActiveList,  
['prompt' => 'Please Choose One']);?>  
  
<?= $form->field($model, 'marketing_image_weight') ?>  
  
<?= $form->field($model, 'status_id')->dropDownList($model->statusList);?>  
  
<?= $form->field($model, 'file')->fileInput(); ?>
```

```

<div class="form-group">
    <?= Html::submitButton($model->isNewRecord ? 'Create' : 'Update',
        ['class' => $model->isNewRecord ?
            'btn btn-success' : 'btn btn-primary']) ?>
</div>

<?php ActiveForm::end(); ?>

</div>

```

First thing to note is:

```
<?php $form = ActiveForm::begin(['options'=> ['enctype'=> 'multipart/form-data']]); ?>
```

Setting the options like **this** allows us to upload a **file**.

The other important thing to note is:

```
<?= $form->field($model, 'file')->fileInput(); ?>
```

This adds the button to upload the **file**. And if you remember in the model, we set the label to ‘Marketing Image.’

So your form should look like **this**:

```
! [File Form](images/originals/fileform.png)
```

Modifying the Controller

Gist:

```
[MarketingImageController](https://gist.github.com/evercode1/c0a66f25501f2fb6914\ a)
```

From book:

```

<?php
namespace backend\controllers;

use Yii; use backend\models\MarketingImage; use backend\models\search\MarketingImageSearch;
use yii\web\Controller; use yii\web\NotFoundHttpException; use yii\web\ForbiddenHttpException;
use yii\filters\VerbFilter; use common\models\PermissionHelpers; use yii\web\UploadedFile;
/** * MarketingImageController implements the CRUD * actions for MarketingImage model. */
class MarketingImageController extends Controller {
    public function behaviors() {
        return [

```

```
'access' => [
    'class' => \yii\filters\AccessControl::className(),
    'only' => ['index', 'view', 'create', 'update', 'delete'],
    'rules' => [
        [
            'actions' => ['index', 'view', 'create', 'update', 'delete'],
            'allow' => true,
            'roles' => ['@'],
            'matchCallback' => function ($rule, $action) {
                return PermissionHelpers::requireMinimumRole('Admin')
                    && PermissionHelpers::requireStatus('Active');
            }
        ],
    ],
],
'verbs' => [
    'class' => VerbFilter::className(),
    'actions' => [
        'delete' => ['post'],
    ],
],
];
}

/**
 * Lists all MarketingImage models.
 * @return mixed
 */

public function actionIndex()
{
    $searchModel = new MarketingImageSearch();
    $dataProvider = $searchModel->search(Yii::$app->request->queryParams);

    return $this->render('index', [
        'searchModel' => $searchModel,
        'dataProvider' => $dataProvider,
    ]);
}
```

```
/**  
 * Displays a single MarketingImage model.  
 * @param string $id  
 * @return mixed  
 */  
  
public function actionView($id)  
{  
    return $this->render('view', [  
        'model' => $this->findModel($id),  
    ]);  
}  
  
/**  
 * Creates a new MarketingImage model.  
 * If creation is successful, the browser will be redirected to the  
 * 'view' page.  
 * @return mixed  
 */  
  
public function actionCreate()  
{  
    $model = new MarketingImage();  
  
    if ($model->load(Yii::$app->request->post())) {  
  
        $imageName = $model->marketing_image_name;  
  
        $model->file = UploadedFile::getInstance($model, 'file');  
  
        $fileName = 'uploads/' . $imageName . '.' . $model->file->extension;  
        $fileName = preg_replace('/\s+/', '', $fileName);  
  
        $model->marketing_image_path = $fileName;  
        $model->save();  
  
        $model->file->saveAs($fileName);  
  
        //Save the path in the DB  
  
        return $this->redirect(['view', 'id' => $model->id, 'model' => $model]);  
    }  
}
```

```
    } else {
        return $this->render('create', [
            'model' => $model,
        ]);
    }
}

/**
 * Updates an existing MarketingImage model.
 * If update is successful, the browser will be
 * redirected to the 'view' page.
 * @param string $id
 * @return mixed
 */

public function actionUpdate($id)
{
    $model = $this->findModel($id);

    if ($model->load(Yii::$app->request->post())) {

        $imageName = $model->marketing_image_name;

        $oldImage = MarketingImage::find('marketing_image_name')
            ->where(['id' => $id])
            ->one();

        if ($oldImage->marketing_image_name != $imageName){

            throw new ForbiddenHttpException
                ('You cannot change the name, you must delete instead.');
        }

        $model->file = UploadedFile::getInstance($model, 'file');

        $model->save();

        $model->file->saveAs('uploads/' . $imageName . '.' . $model->file->extension);
    }
}
```

```
        return $this->redirect(['view', 'id' => $model->id]);  
  
    } else {  
        return $this->render('update', [  
            'model' => $model,  
        ]);  
    }  
}  
  
/**  
 * Deletes an existing MarketingImage model.  
 * If deletion is successful, the browser will be  
 * redirected to the 'index' page.  
 * @param string $id  
 * @return mixed  
 */  
  
public function actionDelete($id)  
{  
  
    $model = $this->findModel($id);  
  
    if(unlink($model->marketing_image_path)){  
  
        $model->delete();  
  
        return $this->redirect(['index']);;  
    }  
  
    throw new NotFoundHttpException('We were unable to Delete');  
}  
  
/**  
 * Finds the MarketingImage model based on its primary key value.  
 * If the model is not found, a 404 HTTP exception will be thrown.  
 * @param string $id  
 * @return MarketingImage the loaded model  
 * @throws NotFoundHttpException if the model cannot be found  
 */  
  
protected function findModel($id)
```

```

{
    if (($model = MarketingImage::findOne($id)) !== null) {
        return $model;
    } else {
        throw new NotFoundHttpException('The requested page does not exist.');
    }
}

}

```

Ok, so let's discuss what **else** is different. You can see we added some **new use** \ statements **and** the following is very important:

use yii\webUploadedFile;

This gives us access to the UploadedFile class, which we will need to work with \ our **file** upload.

Also, we obviously modified behaviors to make it compliant with the other backend\ controllers, so that it requires admin **or** greater **for** access.

After that, it's just 3 methods that are different, actionCreate, actionUpdate, \ **and** actionDelete.

The Create Action

Our create action starts just **as** we expect:

public function actionCreate() { \$model = new MarketingImage();

```
if ($model->load(Yii::$app->request->post())) {
```

We create a **new** instance of MarketingImage, then load the post data **if** we have a\ ny.

Next we create a local variable **\$imageName** to make things easier to work with an\ d assign it like so:

\$imageName = \$model->marketing_image_name;

Next we assign the `$model->file` property like so:

```
$model->file = UploadedFile::getInstance($model, 'file');
```

So now we are working with an instance of the actual `file` and we will be able to assign it the name we want. But before we save it, we do some formatting to make sure we are stripping any spaces out of the filename.

So just to make `this` perfectly clear, we are dealing with an instance of the actual `file`, which will be stored in our uploads folder, and also an instance of the model, which we will store in our DB. These are two separate things.

Once we have made sure the `fileName` is formatted correctly, we assign it to our `$model->marketing_image_path` property, then save the model:

```
$model->marketing_image_path = $fileName; $model->save();
```

It's important that we save the model, and therefore perform validation, before we save the the `file`, which comes next:

```
$model->file->saveAs($fileName);
```

Just a tip, if you do that out of order, the instance of the `file` is no longer available in a temp directory and it messes up the validation, and you will get an error. So make sure to save model first, then the `file`.

The validation I'm referring to is in the model:

```
[['file'], 'file', 'extensions' => ['png', 'jpg', 'gif'], 'maxSize' => 1024*1024],
```

Yii 2 gives us a simple way to define the allowable extensions and the max size of the `file`.

So, returning to the controller. After saving, we go to the view `file`:

```
return $this->redirect(['view', 'id' => $model->id, 'model' => $model,]);
```

Otherwise, if we don't have a valid post, show the form:

```
else { return $this->render('create', [ 'model' => $model,
```

```
    ]);
}

## The Update Action
```

The update action has notable differences. We need to know which instance of the model we are working on, so we take in the `$id` of the model in the method signature `and` then find that instance of the model:

```
public function actionUpdate($id) { $model = $this->findModel($id);
```

If we have valid post data, we set a local variable `$imageName` to the value in the post:

```
$imageName = $model->marketing_image_name;
```

Then we need to lookup the existing image name `and` compare the two:

```
$oldImage = MarketingImage::find('marketing_image_name') ->where(['id' => $id]) ->one();
```

```
if ($oldImage->marketing_image_name != $imageName){

    throw new ForbiddenHttpException
    ('You cannot change the name, you must delete instead.');

}
```

If they don't match, we `throw` an exception.

The reason I'm doing `this` is so we don't end up with a bunch of orphan files on the server that have no corresponding reference in the DB. You could also accomplish `this` by creating a custom validator `or` by doing a more robust implementation that covers that scenario.

I preferred to keep things simple.

So `if` we're good with the image name:

```
$model->file = UploadedFile::getInstance($model, 'file');
$model->save();
```

We get an instance of the `file` from post, then save the model. `This` is useful if we are just changing something like `marketing_image_is_featured` from no to yes.

To make an update, you will be required to upload the photo again because the validation rule requires it.

So we save the `new` instance of the `file` and redirect to view:

```
$model->file->saveAs('uploads/' . $imageName . '.' . $model->file->extension);

return $this->redirect(['view', 'id' => $model->id]);
```

`Else`, we show the form:

```
else { return $this->render('update', [ 'model' => $model, ]); }
```

The Delete Action

Ok, one last method, `actionDelete`. We take in the id and look up the appropriate model:

```
public function actionDelete($id) {

    $model = $this->findModel($id);

    if(unlink($model->marketing_image_path)){
        $model->delete();

        return $this->redirect(['index']);
    }

    throw new NotFoundHttpException('We were unable to Delete');
}
```

We call PHP's `unlink` method to delete the `file` and if successful, we also call `$\model->delete` to delete the model instance from the db.

Then we `return` to index or we `throw` an exception because we were unable to complete the `unlink`. This method probably should use a `try/catch` block to `return` a more useful error. I will make that improvement in the next section.

And that's it for basic image management. It's not a bad implementation, but we can do better.

For example, when we create an image, we should create a thumbnail to go along with it. Then we could `list` that thumbnail in the Gridview and in the other views.

Also, while we're able to update a record, the way we have it setup now is that we also have to re-upload the image or validation fails. But what if you just want to `require` the `file` on create, but not update? That would work out well because it would give you the option of not uploading the `file` again if you just wanted to change something like `marketing_image_is_featured`.

Our solution for that will utilize yii 2's scenario solution, which is pretty cool. I think you will like it.

The downside is that our update method gets a bit more complicated. The upside is that we get a much more robust solution for updating images.

Ok, let's get started.

Image Thumbnails with Imagine

We are going to take advantage of Yii 2's Imagine plugin to create our thumbnails. This is an extension created by the Yii 2 core team, and it's really easy to use.

Install Yii 2 Imagine Extension

To use this library, we need to add to our `composer.json` in the `require` section:

`"yiisoft/yii2-imagine": "~2.0.0"`

Then run composer update on the command line.

```
! [Composer Update](images/originals/composerupdate.png)
```

Create Thumbnail Folder

Inside of the backend/web/uploads folder, create a thumbnail folder:

```
! [Thumbnail Folder](images/originals/thumbnailfolder.png)
```

Alter Marketing Image Table

We will also need to modify our marketing_image table to hold the thumbnail path.

```
! [New Marketing Image Table](images/originals/newmarketingimagetable.png)
```

Here is the SQL:

```
ALTER TABLE yii2build.marketing_image ADD COLUMN marketing_image_caption_title VARCHAR(100) CHARACTER SET 'utf8' COLLATE 'utf8_unicode_ci' NULL DEFAULT NULL AFTER marketing_image_name, ADD COLUMN marketing_thumb_path VARCHAR(45) CHARACTER SET 'utf8' COLLATE 'utf8_unicode_ci' NOT NULL AFTER marketing_image_caption;
```

Note: I added marketing_image_caption_title to give us more elements to work with in the future, should we decide to expand our options.

Modify MarketingImage Model

Gist:

[Modify MarketingImage Model](<https://gist.github.com/evercode1/774f7d82ffc7be9f94f3>)

From book:

```
<?php
namespace backend\models;
use Yii; use yii\db\ActiveRecord; use yii\dbExpression; use yii\helpers\ArrayHelper; use yii\behaviors\TimestampBehavior;
use yii\helpers\Html;
/** * This is the model class for table "marketing_image". * * @property string $id * @property string $marketing_image_path * @property string $marketing_image_name * @property integer
```

```
$marketing_image_is_featured * @property integer $marketing_image_is_active * @property integer $status_id * @property string $created_at * @property string $updated_at ** @property Status $status */

class MarketingImage extends \yii\db\ActiveRecord {

    public $file;

    /**
     * @inheritdoc
     */

    public static function tableName()
    {
        return 'marketing_image';
    }

    public function behaviors()
    {
        return [
            'timestamp' => [
                'class' => TimeStampBehavior::className(),
                'attributes' => [
                    ActiveRecord::EVENT_BEFORE_INSERT => ['created_at', 'updated_at'],
                    ActiveRecord::EVENT_BEFORE_UPDATE => ['updated_at'],
                ],
                'value' => new Expression('NOW()'),
            ],
        ];
    }

    /**
     * @inheritdoc
     */

    public function rules()
    {
        return [
            [['marketing_image_path', 'marketing_image_name',
                'marketing_thumb_path', 'marketing_image_weight'],
                'required'],
            ['marketing_image_weight', 'default', 'value' => 100 ],
            ['marketing_image_is_featured', 'default', 'value' => 0 ],
        ];
    }
}
```

```
[ 'marketing_image_is_active', 'default', 'value' => 0 ],
[ 'file', 'required', 'message' =>
  '{attribute} can\'t be blank', 'on'=>'create'],
[[['marketing_image_name', 'marketing_image_path'], 'trim'],
[[['marketing_image_is_featured',
  'marketing_image_is_active', 'marketing_image_weight',
  'status_id'], 'integer'],
[['marketing_image_is_featured'],'in',
  'range'=>array_keys($this->getMarketingImageIsFeaturedList())],
[['marketing_image_is_active'],'in',
  'range'=>array_keys($this->getMarketingImageIsActiveList())],
[['file'], 'file', 'extensions' => ['png', 'jpg', 'gif'],
  'maxSize' => 1024*1024],
[['marketing_image_path', 'marketing_image_name'],
  'string', 'max' => 45],
[['marketing_image_caption', 'marketing_image_caption_title'],
  'string', 'max' => 100],


];
}

public function scenarios()
{
  $scenarios = parent::scenarios();
  $scenarios['create'] = ['file','marketing_image_path',
    'marketing_image_name', 'marketing_thumb_path',
    'marketing_image_is_featured', 'marketing_image_is_active',
    'marketing_image_caption', 'marketing_image_caption_title',
    'marketing_image_weight' ];

  return $scenarios;
}

public function beforeValidate()
{
  $this->marketing_image_name =
  preg_replace('/\s+/', '', $this->marketing_image_name);
  $this->marketing_image_path =
  preg_replace('/\s+/', '', $this->marketing_image_path);

  return parent::beforeValidate();
}
```

```
/**  
 * @inheritdoc  
 */  
  
public function attributeLabels()  
{  
    return [  
        'id' => 'ID',  
        'marketing_image_path' => 'Marketing Image Path',  
        'marketing_image_name' => 'Marketing Image Name',  
        'marketing_thumb_path' => 'Marketing thumb Path',  
        'marketing_image_caption' => 'Caption',  
        'marketing_image_caption_title' => 'Caption Title',  
        'marketing_image_is_featured' => 'Marketing Image Is Featured',  
        'marketing_image_is_active' => 'Marketing Image Is Active',  
        'marketing_image_weight' => 'Marketing Image Weight',  
        'status_id' => 'Status ID',  
        'created_at' => 'Created At',  
        'updated_at' => 'Updated At',  
        'file' => 'Marketing Image',  
        'statusName' => Yii::t('app', 'Status'),  
    ];  
}  
  
public static function getMarketingImageIsFeaturedList() { return $droptions = [0 => "no", 1 => "yes"]; }  
  
public static function getMarketingImageIsActiveList() { return $droptions = [0 => "no", 1 => "yes"]; }  
  
/**  
 * @return \yii\db\ActiveQuery  
 */  
  
public function getStatus()  
{  
    return $this->hasOne(Status::className(),  
        ['id' => 'status_id']);  
}  
  
/**  
 * * get status name
```

```

*
*/

public function getStatusName()
{
    return $this->status ? $this->status->status_name : '- no status -';
}

/**
 * get list of statuses for dropdown
 */

public static function getStatusList()
{
    $droptions = Status::find()->asArray()->all();
    return ArrayHelper::map($droptions, 'id', 'status_name');
}

public function getThumb()
{
    $image = Html::img('/'.$this->marketing_thumb_path);
    return Html::a($image, ['view', 'id' => $this->id]);
}
}
}

```

So let's talk about what's **new**. We added a **use** statement **for** `Html:`

```
use yii\helpers\Html;
```

We'll need that when we want to **return** the url **for** the thumbnail image in `Gridview`.

We have a couple of different lines in the rules. We also have a rule **for** `marketing_image_caption_title`, but it's **not** shown here:

```
[['marketing_image_path', 'marketing_image_name', 'marketing_thumb_path'], 'required'],
['file', 'required', 'message' => '{attribute} can't be blank', 'on'=>'create'],
```

You can see we added the `new` attribute, `marketing_thumb_path` to the required rule, **and** moved required rule `for file` to a separate line. **This** is because we've added a condition to the rule via the 'on' attribute.

'on'⇒'create'

This tells the rule to apply only to the `create` method. **As** we mentioned, we want the `file` to be `not` required on update because we might only be updating different attributes of the model **and not** the image `file` itself.

`## Scenarios`

So logically, the scenario method comes `next`.

```
public function scenarios() { $scenarios = parent::scenarios(); $scenarios['create'] = [ 'file','marketing_image_path', 'marketing_image_name', 'marketing_thumb_path', 'marketing_image_is_featured', 'marketing_image_is_active', 'marketing_image_caption', 'marketing_image_caption_title', 'marketing_image_weight' ];  
  
    return $scenarios;  
  
}
```

I'm moving through **this** quickly, but you should take note of `scenarios`, **this** is a powerful feature that allows us to finesse the rules.

The format here is fairly intuitive. We're setting a scenario **for** the `create` method, **and** we **list** all the attributes that will be validated under that scenario.

When we want to **use this** in the controller, we **use** it when we call the model:

```
$model = new MarketingImage(); $model->scenario = 'create';
```

All in all, very simple. We will see that in action when we work on the controller.

Next we make the additions to our `attributeLabels`:

'marketing_thumb_path' ⇒ 'Marketing thumb Path', 'marketing_image_caption_title' ⇒ 'Caption Title',

Lastly, we will add a method to `return` an image link `for` our thumbnail, which we\ will `use` on our Gridview:

```
public function getThumb() {  
  
    $image = Html::img('/'. $this->marketing_thumb_path);  
    return Html::a($image, ['view', 'id' => $this->id]);  
  
}
```

You can see we first get the image `and` set it to `$image` using `Html::img`, then we\ `return` it `as` a `link`, using `Html::a`.

You will love how easy `this` is to `use` in Gridview when we come to that part.

Modify MarketingImageSearch

We need to modify `MarketingImageSearch`, since `marketing_image_caption` will be a\ searchable field.

Gist:

[Modified `MarketingImageSearch`] (<https://gist.github.com/evercode1/235fa3486e665bfad46f>)

From book:

```
<?php  
namespace backend\models\search;  
use Yii; use yii\baseModel; use yii\dataActiveDataProvider; use backend\modelsMarketingImage;  
/** * MarketingImageSearch represents the model behind the search form about backend\models\MarketingImage.  
 */  
class MarketingImageSearch extends MarketingImage {  
    public $statusName; /** * @inheritDoc */
```

```
public function rules()
{
    return [
        [['id', 'marketing_image_is_featured',
            'marketing_image_is_active', 'status_id'], 'integer'],
        [['marketing_image_path', 'marketing_image_name',
            'marketing_image_caption',
            'marketing_image_caption_title', 'marketing_image_weight',
            'created_at', 'statusName', 'updated_at'], 'safe'],
    ];
}

/**
 * @inheritDoc
 */

public function scenarios()
{
    // bypass scenarios() implementation in the parent class
    return Model::scenarios();
}

/**
 * Creates data provider instance with search query applied
 *
 * @param array $params
 *
 * @return ActiveDataProvider
 */
public function search($params)
{
    $query = MarketingImage::find();

    $dataProvider = new ActiveDataProvider([
        'query' => $query,
    ]);

    $dataProvider->setSort([
        'attributes' => [
            'id',
    ],
}
```

```
'marketing_image_name',
'marketing_image_path',
'marketing_image_caption',
'marketing_image_caption_title',
'marketing_image_is_featured',
'marketing_image_is_active',
'marketing_image_weight',
'statusName' => [
    'asc' => ['status.status_name' => SORT_ASC],
    'desc' => ['status.status_name' => SORT_DESC],
    'label' => 'Status'
],
],
]);

if (!($this->load($params) && $this->validate())) {

    $query->joinWith(['status']);

    return $dataProvider;
}

$this->addSearchParameter($query, 'id');
$this->addSearchParameter($query, 'marketing_image_name', true);
$this->addSearchParameter($query, 'marketing_image_path', true);
$this->addSearchParameter($query, 'marketing_image_caption', true);
$this->addSearchParameter($query, 'marketing_image_caption_title', true);
$this->addSearchParameter($query, 'marketing_image_is_featured');
$this->addSearchParameter($query, 'marketing_image_is_active');
$this->addSearchParameter($query, 'marketing_image_weight');
$this->addSearchParameter($query, 'status_id');

// filter by gender name

$query->joinWith(['status' => function ($q) {
    $q->andFilterWhere(['=' , 'status.status_name', $this->statusName]);
}]);
```

```
        return $dataProvider;

    }

protected function addSearchParameter($query, $attribute, $partialMatch = false) { if (($pos = strpos($attribute, '.')) !== false) { $modelAttribute = substr($attribute, $pos + 1); } else { $modelAttribute = $attribute; }

    $value = $this->$modelAttribute;
    if (trim($value) === '') {
        return;
    }

    /*
     * The following line is additionally added for right aliasing
     * of columns so filtering happen correctly in the self join
    */

    $attribute = "marketing_image.$attribute";

    if ($partialMatch) {
        $query->andWhere(['like', $attribute, $value]);
    } else {
        $query->andWhere([$attribute => $value]);
    }
}

## Modifying Marketing Image Controller
```

Gist:

[Marketing Image Controller](<https://gist.github.com/evercode1/df65a0ecfa16bfff75\90d>)

From book:

```
<?php
namespace backend\controllers;
```

```
use Yii; use backend\models\MarketingImage; use backend\models\search\MarketingImageSearch;
use yii\web\Controller; use yii\web\NotFoundHttpException; use yii\web\ForbiddenHttpException;
use yii\filters\VerbFilter; use common\models\PermissionHelpers; use yii\web\UploadedFile; use yii\imagine\Image;
/** * MarketingImageController implements the CRUD actions for * MarketingImage model. */
class MarketingImageController extends Controller { public function behaviors() { return [
    'access' => [
        'class' => \yii\filters\AccessControl::className(),
        'only' => ['index', 'view', 'create', 'update', 'delete'],
        'rules' => [
            [
                'actions' => ['index', 'view', 'create',
                               'update', 'delete'],
                'allow' => true,
                'roles' => ['@'],
                'matchCallback' => function ($rule, $action) {
                    return PermissionHelpers::requireMinimumRole('Admin')
                        && PermissionHelpers::requireStatus('Active');
                }
            ],
        ],
    ],
    'verbs' => [
        'class' => VerbFilter::className(),
        'actions' => [
            'delete' => ['post'],
        ],
    ],
];
}

/**
 * Lists all MarketingImage models.
 * @return mixed
 */
public function actionIndex()
{
```

```
$searchModel = new MarketingImageSearch(); $dataProvider = $searchModel->search(Yii::$app->request->queryParams);

    return $this->render('index', [
        'searchModel' => $searchModel,
        'dataProvider' => $dataProvider,
    ]);
}

/**
 * Displays a single MarketingImage model.
 * @param string $id
 * @return mixed
 */

public function actionView($id)
{
    return $this->render('view', [
        'model' => $this->findModel($id),
    ]);
}

/**
 * Creates a new MarketingImage model.
 * If creation is successful, the browser will be redirected to
 * the 'view' page.
 * @return mixed
 */

public function actionCreate()
{
    $model = new MarketingImage();
    $model->scenario = 'create';

    if ($model->load(Yii::$app->request->post())) {

        $imageName = $model->marketing_image_name;

        $model->file = UploadedFile::getInstance($model, 'file');

        $fileName = 'uploads/' . $imageName . '.' . $model->file->extension; $fileName = preg_replace('/\s+/', '', $fileName);
    }
}
```

```
$thumbName = 'uploads/' . 'thumbnail/' . $imageName
. 'thumb.' . $model->file->extension;

$thumbName = preg_replace('/\s+/', '', $thumbName);

$model->marketing_image_path = $fileName;
$model->marketing_thumb_path = $thumbName;
$model->save();

$model->file->saveAs($fileName);

Image::thumbnail( $fileName , 60, 60)
->save($thumbName, ['quality' => 50]);

return $this->redirect(['view', 'id' => $model->id, 'model' => $model,]);}

} else {
    return $this->render('create', [
        'model' => $model,
    ]);
}

}

/** 
 * Updates an existing MarketingImage model.
 * If update is successful, the browser will be
 * redirected to the 'view' page.
 * @param string $id
 * @return mixed
 */
public function actionUpdate($id)
{
    $model = $this->findModel($id);

    if ($model->load(Yii::$app->request->post())) {

        $imageName = $model->marketing_image_name;

        $oldImage = MarketingImage::find('marketing_image_name')
            ->where(['id' => $id])
    }
}
```

```
->one();

if ($oldImage->marketing_image_name != $imageName){

    throw new ForbiddenHttpException
        ('You cannot change the name, you must delete instead.');

}

if( $model->file = UploadedFile::getInstance($model, 'file')){

    $thumbName =  'uploads/' . 'thumbnail/' . $imageName . 'thumb.'
    . $model->file->extension;

    $model->save();

} else {

    $model->save();

}

if ($model->file) {

    $fileName = 'uploads/' . $imageName . '.' . $model->file->extension;

    $model->file->saveAs($fileName);

    Image::thumbnail( $fileName , 60, 60)
    ->save($thumbName, ['quality' => 50]);

}

return $this->redirect(['view', 'id' => $model->id]);

} else {
    return $this->render('update', [
        'model' => $model,
    ]);
}
```

```
}

/**
 * Deletes an existing MarketingImage model.
 * If deletion is successful, the browser will be redirected
 * to the 'index' page.
 * @param string $id
 * @return mixed
 */

public function actionDelete($id)
{
    $model = $this->findModel($id);

    try {
        unlink($model->marketing_image_path);
        unlink($model->marketing_thumb_path);
        $model->delete();
        return $this->redirect(['index']);
    }

    catch(\Exception $e) {
        throw new NotFoundHttpException($e->getMessage());
    }
}

/**
 * Finds the MarketingImage model based on its primary key value.
 * If the model is not found, a 404 HTTP exception will be thrown.
 * @param string $id
 * @return MarketingImage the loaded model
}
```

```

* @throws NotFoundHttpException if the model cannot be found
*/

protected function findModel($id)
{
    if (($model = MarketingImage::findOne($id)) != null) {
        return $model;
    } else {
        throw new NotFoundHttpException
            ('The requested page does not exist.');
    }
}

}

```

So let's just discuss what's **new**. We start with a **new use** statement:

```
use yii\imagine\Image;
```

Modify Create Action

The **next new** part is **actionCreate**. We start it a little differently:

```
public function actionCreate() { $model = new MarketingImage(); $model->scenario = 'create';
```

You can see we are setting our **\$model->scenario** to 'create.'

Alternatively, we could:

```
$model = new MarketingImage(['scenario' => 'create']);
```

We will also have to account **for** the **marketing_thumb_path** attribute **and for** the \ actual thumbnail images themselves.

So after we set **\$fileName**, we will assign a local variable to **\$thumbName** **and** the \n **do** the same kind of formatting to make sure we don't have spaces in the filename:

```

$imageName . 'thumb.' . $model->file->extension;


```

Besides stripping out the spaces, **if** there are any, we are also appending ‘thumb\’ onto `$imageName` **for** the `$thumbName`. **If** we didn’t **do** that, your thumbnail images would have the same filename **as** your primary image. I’m **not** a fan of that. \ So even though they are in separate folders, I like to give them different names\ .

`$thumbName` will result in a path that we can **use** to set the `$model->marketing_th\umb_path` attribute. We have to set that one manually because we are **not** doing it in the form.

The reason **for this** is that the thumbnails are auto-generated from our primary image, so we don’t want the user to be able to set the value. It’s too easy **for** them to make a typo.

Once we’ve set that attribute, we can **do** a `$model->save()`:

```
$model->marketing_image_path = $fileName; $model->marketing_thumb_path = $thumbName;
$model->save();
```

That takes care of the DB, but we still need to save our primary image **and** create a thumbnail from it. Fortunately, our imagine Image **class makes this** a snap:

```
$model->file->saveAs($fileName);
Image::thumbnail( $fileName , 60, 60 )->save($thumbName, ['quality' => 50]);
```

In **this case**, `$fileName` is the `file` we are creating the thumbnail from. 60, 60 \ are dimensions, `$thumbName` is the path to save to, **and** ‘quality’ is set to 50. \ It’s really just one line of code, very simple indeed.

And the rest is the same **as** we had before. You can test **this and** it should work\ perfectly, however we have **not** added the thumbnail in a view anywhere yet, so you won’t see it. You can check it by checking in the thumbnail folder.

```
## Modify Update Action
```

Ok, moving on to update. **This** one got a little more complicated, but don’t worry, I will take you through it step by step.

This part is exactly **as** it was before:

```
public function actionUpdate($id) { $model = $this->findModel($id);
```

```

if ($model->load(Yii::$app->request->post())){
    $imageName = $model->marketing_image_name;

    $oldImage = MarketingImage::find('marketing_image_name')
        ->where(['id' => $id])
        ->one();

    if ($oldImage->marketing_image_name != $imageName){
        throw new ForbiddenHttpException
            ('You cannot change the name, you must delete instead.');
    }
}

```

Next we check to see **if** there is a **new** file uploaded, **and if** so, set the **\$thumbName**, which we will **use** to create the **new** thumb, **and** **\$model->save()**:

```

if ($model->file = UploadedFile::getInstance($model, 'file')){
    $thumbName = 'uploads/' . 'thumbnail/' . $imageName . 'thumb.' . $model->file->extension;

    $model->save();
}

}

```

If we don't have **file** uploaded, which is now possible because we used a scenario\ to only enforce the **require** rule on create, then we simply save the model:

```

else {
    $model->save();
}

```

Next, **if** we **do** have a **file** uploaded, we need to set the **\$fileName**, then save the\ **file** **and** create the thumbnail:

```
if ($model->file) {
```

```

$fileName = 'uploads/' . $imageName . '.'
. $model->file->extension;

$model->file->saveAs($fileName);

Image::thumbnail( $fileName , 60, 60)
->save($thumbName, ['quality' => 50]);

}

```

So we are only saving a primary image **and** thumbnail image **if** there is an image uploaded, otherwise we are happily redirected to the view with the `$model->id` that we handed in, same **as** we had before

Modify Delete Action

Ok, so we need to account **for** the thumbnail in our delete action **and** we also wanted to wrap **this** in a **try/catch** block:

```

public function actionDelete($id) {

    $model = $this->findModel($id);

    try {

        unlink($model->marketing_image_path);

        unlink($model->marketing_thumb_path);

        $model->delete();

        return $this->redirect(['index']);

    }

    catch(\Exception $e) {

        throw new NotFoundHttpException($e->getMessage());

    }

}

```

Nothing too difficult there. We obviously have to `unlink` both files to get rid \ of the image completely.

Notice the \ in the `catch`:

```
catch(Exception $e) {
```

When you are using built-in PHP classes such `as` `Exception`, you have to put a \ i\ n front `or` Yii 2 will `not` find it.

The other thing I did was `throw` an exception:

```
throw new NotFoundHttpException($e->getMessage());
```

So what that does is `if` one of the files is missing, `or` it can't complete the un\ link, it will `return` the message formatted gracefully, which means it will still\ have the page layout instead of a raw message on a white background.

```
## Modify Views
```

```
## Modify View
```

We're just adding the thumbnail, but I'll give you the entire `file for` reference.

Gist:

```
[view.php](https://gist.github.com/evercode1/1a4a1641fbda2d6b17f8)
```

From book:

```
<?php  
use yii\helpers\Html; use yii\widgetsDetailView;  
/* @var $this yii\web\View // @var $model backend\models\MarketingImage */  
$this->title = $model->id; $this->params['breadcrumbs'][] = ['label' => 'Marketing Images', 'url' => ['index']];  
$this->params['breadcrumbs'][] = $this->title; ?>  
<div class="marketing-image-view">
```

```
<p>
    <?= Html::a('Update', ['update', 'id' => $model->id],
        ['class' => 'btn btn-primary']) ?>

    <?= Html::a('Delete', ['delete', 'id' => $model->id], [
        'class' => 'btn btn-danger',
        'data' => [
            'confirm' => 'Are you sure you want to delete this item?',
            'method' => 'post',
        ],
    ]) ?>
</p>

<h1><?= Html::encode($model->marketing_image_name) ?></h1>

<br>
<div> <?php

echo Html::img('/'. $model->marketing_image_path . '?' . 'time='
    . time() , ['width' => '600px']);

?>

</div>
<br>
<div>
<?php

echo Html::img('/'. $model->marketing_thumb_path . '?' . 'time=' . time());

?>

</div>

<br>
```

```
<?= DetailView::widget([
    'model' => $model,
    'attributes' => [
        'id',
        'marketing_image_caption_title',
        'marketing_image_caption',
        'marketing_image_path',
        'marketing_thumb_path',
        'marketing_image_weight',
        //marketing_image_name,
        ['attribute' => 'marketing_image_is_featured',
            'format' => 'boolean'],
        ['attribute' => 'marketing_image_is_active',
            'format' => 'boolean'],
        'status.status_name',
        'created_at',
        'updated_at',
    ],
]) ?>
```

</div>

Modify Update

Again just adding the thumb **and** providing the entire **file for** reference.

Gist:

[update.php](<https://gist.github.com/evercode1/bdbac4903b38a7ce85b3>)

From book:

```
<?php
use yii\helpers\Html;
/* @var $this yii\web\View // @var $model backend\models\MarketingImage */
$this->title = 'Update Marketing Image: ' . ' ' . $model->id;
$this->params['breadcrumbs'][] = ['label' => 'Marketing Images', 'url' => ['index']];
$this->params['breadcrumbs'][] = ['label' => $model->id, 'url' => ['view', 'id' => $model->id]];
$this->params['breadcrumbs'][] = 'Update'; ?> <div class="marketing-image-update">
```

```
<h1><?= Html::encode($this->title) ?></h1>

<br>
<div>
<?php
echo Html::img('/'. $model->marketing_image_path, ['width' => '600px']);

?>
</div>
<br>
<div>
<?php

echo Html::img('/'. $model->marketing_thumb_path . '?' . 'time=' . time());

?>

</div>

<br>

<?= $this->render('_form', [
    'model' => $model,
]) ?>

</div>

## Modify _form
```

Just add the one line:

```
<?= $form->field($model, 'marketing_image_caption_title') ?>
```

```
## Modify _search
```

Again, we just modify one line:

```
<?= $form->field($model, 'marketing_image_caption_title') ?>
```

```
## Modify Index
```

We only have a small change here, but again I'm providing the entire `file for` reference. Doing `this` is a way `for` me to make sure I haven't missed something, since `this` code comes directly from my IDE.

Gist:

[index.php](<https://gist.github.com/evercode1/4c7c47b51fc2e439fd13>)

From book:

```
<?php
use yii\helpers\Html; use yii\grid\GridView; use \yii\bootstrap\Collapse;
/* @var $this yii\web\View // @var $searchModel backend\models\searchMarketingImageSearch /
@var $dataProvider yii\data\ActiveDataProvider */
$this->title = 'Marketing Images'; $this->params['breadcrumbs'][] = $this->title; ?> <div class="marketing-image-index">

<h1><?= Html::encode($this->title) ?></h1>
<?php echo Collapse::widget([
    'encodeLabels' => false,
    'items' => [
        // equivalent to the above
        [
            'label' => '<i class="fa fa-caret-square-o-down"></i> Search',
            'content' => $this->render('_search', ['model' => $searchModel]),
            // open its content by default
            //'contentOptions' => ['class' => 'in']
        ],
    ],
]);?
?>
```

```
<p>
    <?= Html::a('Create Marketing Image', ['create'],
        ['class' => 'btn btn-success']) ?>
</p>

<?= GridView::widget([
    'dataProvider' => $dataProvider,
    'filterModel' => $searchModel,
    'columns' => [
        ['class' => 'yii\grid\SerialColumn'],
        'id',
        'marketing_image_path',
        'marketing_image_name',
        'marketing_image_caption_title',
        'marketing_image_caption',
        ['attribute'=>'marketing_image_is_featured',
            'format'=>'boolean'],
        ['attribute'=>'marketing_image_is_active',
            'format'=>'boolean'],
        'marketing_image_weight',
        'statusName',
        //'created_at',
        //'updated_at',
        'thumb:html',
        ['class' => 'yii\grid\ActionColumn'],
    ],
]); ?>

</div>
```

So obviously the change here is the additions of:

'thumb:html', 'marketing_image_caption_title',

What a powerful little line ‘thumb:html’ is. When we call ‘thumb’, we are calling the model method, using magic get syntax, so that is actually calling getThumb from the model. The part after the colon tells gridview it’s returning html.\

And that’s it! You get a clickable thumbnail in your grid. You also get the \ column heading.

There are a lot of ways to appreciate the power of Yii 2 **and this** is certainly one of them.

Ok, so now that we got our image management up **and** running, we want to put our images to good **use** in the frontend.

URL Manager

Because we are using Yii 2’s advanced template, using the images from the backend in the frontend takes some configuration. **This** is because the frontend **and** backend are two separate applications.

So what we need to **do** is add to communicate with the backend path, **and** we **do** that by setting up a **key** in our components **array**. I should note that I got **this** solution from Skworden in the forum after I spent quite a few hours being stumped by it. Yii 2 has a great community of developers **and** that can really help you out when you need it.

Anyway, what we need to **do** is add the following to your frontend/config/main.php\ file in your components **array**:

```
//can name it whatever you want as it is custom
```

```
'urlManagerBackend' => [
    'class' => 'yii\web\UrlManager',
    'baseUrl' => 'http://backend.yii2build.com',
    'enablePrettyUrl' => true,
    'showScriptName' => false,
],
```

I should also note that `this` is still a bit of a workaround. I would prefer `not` to `use` an absolute url here `if` I don't have to because it's `not as` efficient `as` a relative `link`.

If I can't find any other solution, I will eventually reverse it so that the images `and` thumbnails are saved to the frontend `and` referenced by absolute `link` on the backend. `This` would be more efficient in terms of server resource.

Every once in a `while` you come across something you can't instantly solve, `and` this is especially `true` with a big framework like Yii 2. It takes a long `time` to learn every nuance of the framework, so everyone, including me, has to have patience `for` that.

Anyway, once you have that in place, you can reference images that are stored in the backend, like so:

```
echo Html::img(Yii::$app->urlManagerBackend->baseUrl.'/uploads/your-image.png');
```

I'm going to give you a Gist `for` reference on the entire `file`, in `case` you need to debug. Remember, the path is `frontend/config/main.php`:

Gist:

[main.php](<https://gist.github.com/evercode1/0ab628b8862d6d214ac5>)

Carousel Widget

Ok, our objective will be to have a carousel widget that we can embed on the index page of the site, which just `as` a reminder, is controlled by the pages controller.

The idea will be to be able to call our carousel with a single line of code, like so:

```
<?= CarouselWidget::widget(['settings' => [ 'height' => '300px', 'width' => '700px']]?)>
```

You can see that we will have the option to hand in settings, in `this case` it will be height `and` width of the images. We will also build our widget to have default settings, so we could call it like so:

```
<?= CarouselWidget::widget()?>
```

We will create the widget so that it sets `default` values `if` we choose to call it\ without settings.

When we created our `MarketingImages` model, we made three attributes that will be\ used to determine `if` the image will be shown in the carousel. One is the `marke\ ting_image_is` featured boolean column, which `if true`, means the image is display\ ed.

The second is the `marketing_image_is_active` boolean column, which `if true`, means\ the image is the active image, which means it is the one that is first loaded i\ n the carousel. So when you are creating images, you should only have one image\ set to `true`.

The last column we test `for` is the status column, `and` we only want the images th\ at are active.

So `this` data structure is nice `and` flexible, `while` giving the admin total contro\ l via UI to display `or not` to display images in the carousel.

Before we code the actual widget, we need to setup the files.

If you remember from chapter 12, when we introduced you to widgets, the basic st\ ructure is two files, a widget `file` directly under the components directory `and` \ a view `file`, which is in the views folder inside the components directory.

![Carousel Widget in Components](images/originals/carouselwidget.png)

So you can see the two files we added there. I will give you the files in a mom\ ent. First, let's set up our component. Go to `common\config\main.php` `and` add t\ his to your components `array`:

```
'carouselwidget' => [  
    'class' => 'components\CarouselWidget',  
],
```

Now our application will be able to see the CarouselWidget class.

```
## CarouselWidget.php
```

Gist:

[CarouselWidget.php](<https://gist.github.com/evercode1/87b7632a3347bd577033>)

from book:

```
<?php namespace components;  
use yii\baseWidget; use yii\helpersHtml; use Yii; use backend\modelsMarketingImage; use bac-  
kend\models\searchMarketingImageSearch; use yii\webNotFoundException; use yii\filtersVerbFilter;  
class CarouselWidget extends Widget {  
  
    public $activeImage;  
    public $images;  
    public $count;  
    public $settings = [];  
  
    public function init()  
{  
        parent::init();  
  
        $this->activeImage = MarketingImage::find('marketing_image_path')  
            ->where(['marketing_image_is_active' => 1])  
            ->andWhere(['marketing_image_is_featured' => 1])  
            ->andWhere(['status_id' => 1])  
            ->one();  
  
        $this->images = MarketingImage::find()  
            ->where(['marketing_image_is_active' => 0])  
            ->andWhere(['marketing_image_is_featured' => 1])  
            ->andWhere(['status_id' => 1])  
            ->orderBy('marketing_image_weight')  
            ->all();  
  
        $this->count = MarketingImage::find()
```

```
->where(['marketing_image_is_active' => 0])
->andWhere(['marketing_image_is_featured' => 1])
->andWhere(['status_id' => 1])
->count();

$this->setDefaults();

$this->validateSize();

}

public function setDefaults()
{

    if(!isset($this->settings['height'])){
        $this->settings['height'] = '300px';
    }

    if(!isset($this->settings['width'])){
        $this->settings['width'] = '700px';
    }

    if(!isset($this->settings['autoplay'])){
        $this->settings['autoplay'] = true;
    }

}

public function validateSize()
{
    if (!preg_match("/px/", $this->settings['width'])
    or !preg_match("/px/", $this->settings['height'])) {

        throw new NotFoundHttpException('You Must Use px
            and number for size, example 300px');
    }
}
```

```
}

$height = (int) preg_replace("/[^0-9]/","", $this->settings['height']);

switch ($height){

    case $height < 40 :

        throw new NotFoundHttpException('You Must Stay within
40 to 1000 px and use px and number for size, example 300px');
break;

    case $height > 1000 :

        throw new NotFoundHttpException('You Must Stay within
40 to 1000 px and use px and number for size, example 300px');
break;

}

$width = (int) preg_replace("/[^0-9]/","", $this->settings['width']);

switch ($width){

    case $width < 40 :
        throw new NotFoundHttpException('You Must Stay within
40 to 1000 px and use px and number for size, example 300px');
break;

    case $width > 1000 :
        throw new NotFoundHttpException('You Must Stay within
40 to 1000 px and use px and number for size, example 300px');
break;

}

public function run()
```

```
{
    return $this->render('carousel',
        ['activeImage' => $this->activeImage,
         'images' => $this->images,
         'count' => $this->count,
         'settings' => $this->settings]);
}

}
```

Since we covered the creation of a custom widget in chapter 12, I'm going to step through `this` quickly. Obviously, we `include` everything we need in the `use` statements, then move on to declaring the `class properties`:

```
public $activeImage; public $images; public $count; public $settings = [];
```

The `$activeImage` property will hold the ActiveRecord instance with the image that has `marketing_image_is_active` set to a value of 1. Please remember to keep only one record with that value.

The `$images` property will be an instance of ActiveRecord with multiple results, all the images that are set to featured.

Lastly, the `$settings array` will hold our settings values when they are handed in.

So moving on to `init()`:

```
public function init() { parent::init();

    $this->activeImage = MarketingImage::find('marketing_image_path')
        ->where(['marketing_image_is_active' => 1])
        ->andWhere(['marketing_image_is_featured' => 1])
        ->andWhere(['status_id' => 1])
        ->one();

    $this->images = MarketingImage::find()
        ->where(['marketing_image_is_active' => 0])
        ->andWhere(['marketing_image_is_featured' => 1])
        ->andWhere(['status_id' => 1])
        ->orderBy('marketing_image_weight')
```

```

->all();

$this->count = MarketingImage::find()
    ->where(['marketing_image_is_active' => 0])
    ->andWhere(['marketing_image_is_featured' => 1])
    ->andWhere(['status_id' => 1])
    ->count();

$this->setDefaultSize();

$this->validateSize();

}

```

You can see we call the `parent::init()` and then we use ActiveRecord to return the results we are looking for and set them to the **class properties** via `$this`.

Note in the `$this->images` instance we are ordering by `marketing_image_weight`. This is really simple stuff and it gives us complete control over the order that the images will be displayed in the carousel.

We use the `$count` property in the view to determine the correct number of carousel-indicators to display.

Next we call a couple of methods I created to set the `default` size of the images\ if none is provided and also to do just a little validation on the size input. Let's look at the `setDefaults` first:

```

public function setDefaults() {

    if(!isset($this->settings['height'])){
        $this->settings['height'] = '300px';
    }

    if(!isset($this->settings['width'])){
        $this->settings['width'] = '700px';
    }

    if(!isset($this->settings['autoplay'])){

```

```
$this->settings['autoplay'] = true;  
}  
}  
}
```

This allows us to call the widget like so:

```
<?= CarouselWidget::widget()?>
```

Next I decided to do a little validation on the size input, so that we can have \ a hint if something in the settings has a typo:

```
public function validateSize() {  
  
    if (!preg_match("/px/", $this->settings['width'])  
        or !preg_match("/px/", $this->settings['height'])) {  
  
        throw new NotFoundHttpException  
            ('You Must Use px and number for size, example 300px');  
  
    }  
  
    $height = (int) preg_replace("/[^0-9]/","", $this->settings['height']);  
  
    switch ($height){  
  
        case $height < 40 :  
            throw new NotFoundHttpException('You Must Stay within  
                40 to 1000 px and use px and number for size, example 300px');  
            break;  
  
        case $height > 1000 :  
            throw new NotFoundHttpException('You Must Stay within  
                40 to 1000 px and use px and number for size, example 300px');  
            break;  
  
    }  
  
    $width = (int) preg_replace("/[^0-9]/","", $this->settings['width']);
```

```

switch ($width){

    case $width < 40 :
        throw new NotFoundHttpException('You Must Stay within
        40 to 1000 px and use px and number for size, example 300px');
        break;

    case $width > 1000 :
        throw new NotFoundHttpException('You Must Stay within
        40 to 1000 px and use px and number for size, example 300px');
        break;

}
}

```

Ok, so in the first part, I'm just looking to see **if** we included 'px.':

```

if (!preg_match("/px/", $this->settings['width']) or !preg_match("/px/", $this->settings['height'])) {
    throw new NotFoundHttpException ('You Must Use px and number for size, example 300px');
}

```

Then I set minimum **and** maximum values **for** size. But first I had to **extract** the \ number from the string:

```
$height = (int) preg_replace("[^0-9]","", $this->settings['height']);
```

Then we get the **switch** statement to handle the ranges:

```
switch ($height){
```

```
        case $height < 40 :
            throw new NotFoundHttpException('You Must Stay within
40 to 1000 px and use px and number for size, example 300px');
        break;

        case $height > 1000 :
            throw new NotFoundHttpException('You Must Stay within
40 to 1000 px and use px and number for size, example 300px');
        break;

    }

I probably could have written a stronger set of validation rules, but I felt it \
would have been overkill, since we're just looking for a hint of what could be w\
rong if the carousel does not display properly.
```

Moving on, we get to the run method:

```
public function run() { return $this->render('carousel', ['activeImage' => $this->activeImage, 'ima-
ges' => $this->images, 'count' => $this->count, 'settings' => $this->settings]); }
```

We just render `carousel.php` **and** pass in the properties we will need in the view.

```
## carousel.php
```

Now we are ready to look at the widget view:

Gist:

[`carousel.php`] (<https://gist.github.com/evercode1/99f7f24d36fbef36ca89>)

From book:

```
<?php
use yii\helpers\Html;
?>
<div id="carouselMain" class="carousel slide"
```

```
<?php

if($settings['autoplay'] == false ){

    echo 'data-interval="false"';
}

?>

data-ride="carousel">

<ol class="carousel-indicators"> <li data-target="#carouselMain" data-slide-to="0" class="active"></li>

<!-- dynamic indicator data -->

<?php

foreach (range(1, $count) as $number) {

echo '<li data-target="#carouselMain" data-slide-to="'.$number.'></li>';

}

?>

<!-- end dynamic indicator data -->

</ol>

<!-- Wrapper for slides ?

<div class="carousel-inner" role="listbox">

<?php

$width = $settings['width'];
$height = $settings['height'];

//active item first

echo '<div class="item active">
<center>'.
```

```
        Html::img(Yii::$app->urlManagerBackend->baseUrl . '/' .
        $activeImage['marketing_image_path'],
        ['width' => $width, 'height' => $height ])
        . '</center>
<div class="carousel-caption">' . $activeImage['marketing_image_caption'] . '

        </div>
        </div>';

//all other images

foreach ($images as $image){

    echo '<div class="item">
        <center>' .
        Html::img(Yii::$app->urlManagerBackend->baseUrl . '/' .
        $image['marketing_image_path'],
        ['width' => $width, 'height' => $height ])
        . '</center>
        <div class="carousel-caption">' . $image['marketing_image_caption'] . '
        </div>
        </div>';

    }

?>

</div>

<a class="left carousel-control" href="#carouselMain" role="button" data-slide="prev"><span class="glyphicon glyphicon-chevron-left" aria-hidden="true"></span> <span class="sr-only">Previous</span> </a>
<a class="right carousel-control" href="#carouselMain" role="button" data-slide="next"> <span class="glyphicon glyphicon-chevron-right" aria-hidden="true"></span> <span class="sr-only">Next</span>
</a> </div>
```

Looks like a giant plate of spaghetti, but **this** just a straight **copy and paste** \ from Bootstrap 3, with a few dynamic elements.

When working with PHP **and** HTML together, things get ugly quickly. Sometimes I think it's actually easier to work with **this** on your own than to read someone else's code.

You have to very careful to **use** single quotes **and** place them very precisely. **Do**\ **n't worry**, we will step through **this**.

So you can see what changes we made, here is the original bootstrap code **for** reference:

```
[Bootstrap 3 Carousel](http://getbootstrap.com/javascript/#carousel)
```

We named our element `carouselMain` instead of `carousel-example-generic`:

```
<div id="carouselMain" class="carousel slide"
```

Then we get an **if** statement to see **if** we want the carousel to autoplay, which we\ can set in our settings, when we call the widget. It defaults to **true**, but **if** \ autoplay is set to **false**, we **echo** the following:

```
<?php  
    if($settings['autoplay'] == false ){  
        echo 'data-interval="false"';  
    }  
?>
```

That will stop the slides from moving on their own.

Next we need to dynamically determine the number of indicators, which are the clickable circles in the carousel that allow you to jump to any slide. The first \ indicator is set to **0** **for** the active slide.

We need to figure out how many rows **and** what number to place in the subsequent **<\ li** tags. So we **do** a **foreach** loop using PHP's built-in **range function** to set a range between **1** **and** the total number held in `$count`. Then our handy **foreach** loop\ echoes out the correct number of lines with the appropriate number in **each** line\ :

```
<?php
```

```
foreach (range(1, $count) as $number) {  
  
    echo '<li data-target="#carouselMain" data-slide-to="'. $number .'"></li>';  
  
}  
  
?>
```

Next we have the section that echoes the active item **and** the other items:

```
<?php  
  
    $width = $settings['width'];  
    $height = $settings['height'];  
  
    //active item first  
  
    echo '<div class="item active">  
        <center>' .  
        Html::img(Yii::$app->urlManagerBackend->baseUrl . '/' .  
            $activeImage['marketing_image_path'],  
            ['width' => $width, 'height' => $height ]) .  
        '</center>  
<div class="carousel-caption">' . $activeImage['marketing_image_caption'] . '  
  
        </div>  
    </div>;  
  
    //all other images  
  
    foreach ($images as $image){  
  
        echo '<div class="item">  
            <center>' .  
            Html::img(Yii::$app->urlManagerBackend->baseUrl . '/' .  
                $image['marketing_image_path'],  
                ['width' => $width, 'height' => $height ]) .  
            '</center>  
<div class="carousel-caption">' . $image['marketing_image_caption'] . '  
    </div>  
}</div>;
```

```
    }  
?  
To make the code easier to read, I'm assigning the value of $width and $height, \  
but you could easily just pop $settings['height'] into Html image tag.
```

In the **case** of the items, we **use** a **foreach** loop to carefully **echo** out the exact \
syntax need **for** the carousel, **while** injecting our dynamic data.

For the image tag, we are using:

```
Html::img(Yii::$app->urlManagerBackend->baseUrl . '/' . $image['marketing_image_path'], ['width'  
    => $width, 'height' => $height ])
```

You can see we are using our urlManagerBackend to get the baseUrl that allows us\
to reference an image from the backend.

```
## Pages Index
```

I'm giving you the entire **file for** reference, but it's only the top area that ha\
s changed. You can see where we put the widget. I set the autoplay to **false**:

```
<div class="jumbotron">  
<br> <div>  
  
    <?= CarouselWidget::widget(['settings' =>  
        ['height' => '300px', 'width' => '700px'  
        'autoplay'=>false])?>  
  
    </div>  
  
</div>
```

Gist:

[Pages Index] (<https://gist.github.com/evercode1/a033e0f4fbe812271873>)

From book:

```
<?php use \yii\bootstrapModal; use kartik\socialFacebookPlugin; use \yii\bootstrapCollapse; use  
  \yii\bootstrapAlert; use yii\helpers\Html; use componentsFaqWidget; use componentsCarouselWidget;  
  
/* @var $this yii\web\View */ $this->title = 'My Yii Application'; ?> <div class="site-index">  
  
<?php  
  
    if (Yii::$app->user->isGuest) {  
        echo yii\authclient\widgets\AuthChoice::widget([  
            'baseAuthUrl' => ['site/auth'],  
            'popupMode' => false,  
        ]);  
    }  
?>  
  
<div class="jumbotron">  
    <br> <div>  
  
        <?= CarouselWidget::widget(['settings' =>  
            ['height' => '300px', 'width' => '700px'  
            'autoplay'=>false])?>  
  
    </div>  
</div>  
<?php
```

```
echo Collapse::widget([
    'items' => [
        [
            'label' => 'Top Features' ,
            'content' => FacebookPlugin::widget([
                'type'=>FacebookPlugin::SHARE,
                'settings' => ['href'=>'http://www.yii2build.com','width'=>'350']
            ]),
            // open its content by default
            // 'contentOptions' => ['class' => 'in']
        ],
        // another group item
        [
            'label' => 'Top Resources',
            'content' => FacebookPlugin::widget([
                'type'=>FacebookPlugin::SHARE,
                'settings' => ['href'=>'http://www.yii2build.com','width'=>'350']
            ]),
            // 'contentOptions' => [],
            // 'options' => []
        ],
    ]
]);
```

```
Modal::begin([

    'header' => '<h2>Latest Comments</h2>',
    'toggleButton' => ['label' => 'comments'],
])

echo FacebookPlugin::widget([
    'type'=>FacebookPlugin::COMMENT,
    'settings' => ['href'=>'http://www.yii2build.com','width'=>'350']

]);

Modal::end();

?>

<br/> <br/>

<?Php

echo Alert::widget([
    'options' => [
        'class' => 'alert-info',
    ],
    'body' => 'Launch your project like a rocket...',
]);
?>

<div class="body-content">

    <div class="row">
        <div class="col-lg-4">
            <h2>Free</h2>

            <p>
<?php

if (!Yii::$app->user->isGuest) {
```

```
echo Yii::$app->user->identity->username . ' is doing cool stuff. ';
}

?>
```

Starting with this free, open source Yii 2 template and it will save you a lot of time. You can deliver projects to the customer quickly, with a lot of boilerplate already taken care of for you, so you can concentrate on the complicated stuff.</p>

```
<p>
<a class="btn btn-default" href="http://www.yiiframework.com/doc-2.0/guide-index.html">Yii Documentation </a>
</p>
```

```
<?php

echo FacebookPlugin::widget([
    'type'=>FacebookPlugin::LIKE,
    'settings' => []
]);
```

?>

```
</div>
<div class="col-lg-4">
    <h2>Advantages</h2>
```

<p>

Ease of use is a huge advantage. We've simplified RBAC and given you Free/Paid user type out of the box. The Social plugins are so quick and easy to install, you will love it!

```
</p>
<p>
<a class="btn btn-default" href="http://www.yiiframework.com/forum/"> Yii Forum </a>
</p>
```

```
<?php

echo FacebookPlugin::widget([
    'type'=>FacebookPlugin::COMMENT,
    'settings' => ['href'=>'http://www.yii2build.com', 'width'=>'350']

]);

?>
</div>
<div class="col-lg-4">
    <h2>Code Quick, Code Right!</h2>

<p>
Leverage the power of the awesome Yii 2 framework with this enhanced template.
Based Yii 2's advanced template, you get a full frontend and backend implementation that features
rich UI for backend management.

</p>
<p>
<a class="btn btn-default" href="http://www.yiiframework.com/extensions/">Yii Extensions </a>
</p>

    </div>
</div>

</div>
<br>

<?= FaqWidget::widget(['settings' =>
    ['pageSize' => 3, 'featuredOnly' => true,
    'heading' => 'Featured FAQs']] ) ?>

</div>
```

Carousel Settings

Once I had all that done, I thought, wouldn't it be cool **if** you could hand all control of the carousel to admin via the backend? Clients would love that. That shouldn't be too hard **if** you think about it, we have everything we need in place, except **for** the settings model.

This seemed like fun, so I took it up **as** a challenge to see how many of the carousel settings I could manage from backend UI **and** I'm fairly satisfied with the result. It turns out there are more settings that we have previously played with.

Let's start by setting up a CarouselSettings model. **This** is where we will store all of our carousel settings. **And** we will have access to them via the backend, just like we can access roles **and** statuses, etc.

carousel_settings table

Here is a screenshot of what we need:

! [Carousel Settings Table](images/originals/carouselsettingstable.png)

SQL :

```
CREATE TABLE IF NOT EXISTS yii2build.carousel_settings ( id INT(11) NOT NULL AUTO_INCREMENT, carousel_name VARCHAR(45) CHARACTER SET 'utf8' COLLATE 'utf8_unicode_ci' NOT NULL, image_height VARCHAR(45) CHARACTER SET 'utf8' COLLATE 'utf8_unicode_ci' NOT NULL, image_width VARCHAR(45) CHARACTER SET 'utf8' COLLATE 'utf8_unicode_ci' NOT NULL, carousel_autoplay TINYINT(1) NOT NULL DEFAULT '1', show_indicators TINYINT(1) NOT NULL DEFAULT '1', show_caption_title TINYINT(1) NOT NULL DEFAULT 1, show_captions TINYINT(1) NOT NULL DEFAULT '1', show_caption_background TINYINT(1) NOT NULL DEFAULT 1, caption_font_size VARCHAR(45) NOT NULL, show_controls TINYINT(1) NOT NULL DEFAULT 1, status_id SMALLINT(6) NOT NULL, created_at DATETIME NOT NULL, updated_at DATETIME NOT NULL, PRIMARY KEY (id), INDEX fk_carousel_settings_status1_idx (status_id ASC), CONSTRAINT fk_carousel_settings_status1 FOREIGN KEY (status_id) REFERENCES yii2build.status (id) ON DELETE NO ACTION ON UPDATE NO ACTION)
```

Go ahead **and** create the table.

Basically, the table **and** model will supply the attributes **for** our settings **array**\ , so we can call the widget like so:

```
<?= CarouselWidget::widget(['settings' => [ 'height' => $carouselSettings['image_height'], 'width' => $carouselSettings['image_width'], 'autoplay' => $carouselSettings['carousel_autoplay'], 'show_indicators' => $carouselSettings['show_indicators'], 'show_captions' => $carouselSettings['show_captions'], 'show_controls' => $carouselSettings['show_controls'], 'show_caption_title' => $carouselSettings['show_caption_title'], ]])?>
```

Note that in the above, I created friendlier settings names when I could **use** a single word, like height, **for** example. **For** multiple words, I kept it exactly **as** \ it is in the DB.

CarouselSettings Model

Let's start by using Gii to create the model.

Gist:

[CarouselSettings Model] (<https://gist.github.com/evercode1/8c0ea57d838a7ae986bf>)

From book:

```
<?php
namespace backend\models;
use Yii; use yii\db\ActiveRecord; use yii\dbExpression; use yii\helpers\ArrayHelper; use yii\behaviors\TimestampBehavior;
use yii\helpers\Html;

/** * This is the model class for table "carousel_settings". */
class CarouselSettings extends \yii\db\ActiveRecord {
    /**
     * @property integer $id
     * @property string $carousel_name
     * @property string $image_height
     * @property string $image_width
     * @property integer $carousel_autoplay
     * @property integer $show_indicators
     * @property integer $status_id
     * @property string $created_at
     * @property string $updated_at
     */
    public function rules() {
        return [
            [['id', 'status_id'], 'required'],
            [['status_id'], 'integer'],
            [['image_height', 'image_width'], 'string'],
            [['carousel_name'], 'string'],
            [['carousel_autoplay', 'show_indicators'], 'integer'],
            [['created_at', 'updated_at'], 'safe'],
        ];
    }
}
```

```
public static function tableName()
{
    return 'carousel_settings';
}

public function behaviors()
{
    return [
        'timestamp' => [
            'class' => TimeStampBehavior::className(),
            'attributes' => [
                ActiveRecord::EVENT_BEFORE_INSERT => ['created_at', 'updated_at'],
                ActiveRecord::EVENT_BEFORE_UPDATE => ['updated_at'],
            ],
            'value' => new Expression('NOW()'),
        ],
    ];
}

/**
 * @inheritDoc
 */

public function rules()
{
    return [
        ['carousel_name', 'image_height',
            'image_width', 'caption_font_size', 'status_id'], 'required'],
        [['carousel_autoplay', 'show_indicators', 'show_captions',
            'status_id', 'show_controls'], 'integer'],
        [['carousel_autoplay'], 'in',
            'range'=>array_keys($this->getCarouselAutoplayList())],
        [['show_indicators'], 'in',
            'range'=>array_keys($this->getShowIndicatorsList())],
        [['show_captions'], 'in',
            'range'=>array_keys($this->getShowCaptionsList())],
        [['show_caption_background'], 'in',
            'range'=>array_keys($this->getShowCaptionBackgroundList())],
        [['show_caption_title'], 'in',
            'range'=>array_keys($this->getShowCaptionTitleList())],
        [['show_controls'], 'in',
    ];
```

```
        'range'=>array_keys($this->getShowControlsList())),
        [['status_id'], 'in', 'range'=>array_keys($this->getStatusList())),
        [['created_at', 'updated_at'], 'safe'],
        [['carousel_name', 'image_height', 'image_width'],
         'string', 'max' => 45]
    ];
}

/**
 * @inheritDoc
 */
public function attributeLabels()
{
    return [
        'id' => 'ID',
        'carousel_name' => 'Name',
        'image_height' => 'Height',
        'image_width' => 'Width',
        'carousel_autoplay' => 'Autoplay',
        'show_indicators' => 'Show Indicators',
        'show_captions' => 'Show Captions',
        'show_caption_background' => 'Show Caption Background',
        'show_caption_title' => 'Show Caption Title',
        'show_controls' => 'Show Controls',
        'caption_font_size' => 'Caption Size',
        'status_id' => 'Status',
        'created_at' => 'Created At',
        'updated_at' => 'Updated At',
        'statusName' => Yii::t('app', 'Status'),
    ];
}

/**
 * @return \yii\db\ActiveQuery
 */
public function getStatus()
{
    return $this->hasOne(Status::className(),
```

```

['id' => 'status_id']);
}

public function getStatusName()
{
    return $this->status ? $this->status->status_name : '- no status -';

}

/**
 * get list of statuses for dropdown
 */
public static function getStatusList()
{
    $droptions = Status::find()->asArray()->all();
    return ArrayHelper::map($droptions, 'id', 'status_name');
}

public static function getCarouselAutoplayList()

{ return $droptions = [0 => "no", 1 => "yes"]; }

public static function getShowIndicatorsList() { return $droptions = [0 => "no", 1 => "yes"]; } public
static function getShowCaptionsList() { return $droptions = [0 => "no", 1 => "yes"]; }
public static function getShowCaptionTitleList() { return $droptions = [0 => "no", 1 => "yes"]; }
public static function getShowCaptionBackgroundList() { return $droptions = [0 => "no", 1 => "yes"]; }
}
}

```

There really isn't anything on the model that we haven't seen before. I did shorten some of the attribute labels so we could get all the columns without scrolling.

`## CarouselSettingsSearch Model`

Gist:

[CarouselSettingsSearch Model] (<https://gist.github.com/evercode1/687289a4cd60854>)

f6021)

From book:

```
<?php

namespace backend\models\search;

use Yii; use yii\baseModel; use yii\dataActiveDataProvider; use backend\modelsCarouselSettings;

/** * CarouselSettingsSearch represents the model * behind the search form about backend\models\CarouselSettings */
* class CarouselSettingsSearch extends CarouselSettings {

    public $statusName;

    /**
     * @inheritDoc
     */

    public function rules()
    {
        return [
            [['id', 'carousel_autoplay', 'show_indicators',
                'show_captions', 'show_controls',
                'show_caption_background', 'show_caption_title',
                'status_id'], 'integer'],
            [['carousel_name', 'image_height', 'image_width',
                'caption_font_size', 'statusName', 'created_at',
                'updated_at'], 'safe'],
        ];
    }

    /**
     * @inheritDoc
     */

    public function scenarios()
    {
        // bypass scenarios() implementation in the parent class
        return Model::scenarios();
    }

    /**
     * Creates data provider instance with search query applied
     */
```

```
*  
* @param array $params  
*  
* @return ActiveDataProvider  
*/  
  
public function search($params)  
{  
    $query = CarouselSettings::find();  
  
    $dataProvider = new ActiveDataProvider([  
        'query' => $query,  
    ]);  
  
    $dataProvider->setSort([  
        'attributes' => [  
            'id',  
            'carousel_name',  
            'image_height',  
            'image_width',  
            'carousel_autoplay',  
            'show_indicators',  
            'show_captions',  
            'show_caption_title',  
            'show_caption_background',  
            'caption_font_size',  
            'show_controls',  
            'statusName' => [  
                'asc' => ['status.status_name' => SORT_ASC],  
                'desc' => ['status.status_name' => SORT_DESC],  
                'label' => 'Status'  
            ],  
            'updated_at',  
        ]  
    ]);  
  
    if (!$this->load($params) && $this->validate()) {  
  
        $query->joinWith(['status']);  
    }  
}
```

```
        return $dataProvider;
    }

    $this->addSearchParameter($query, 'id');
    $this->addSearchParameter($query, 'carousel_name', true);
    $this->addSearchParameter($query, 'image_height');
    $this->addSearchParameter($query, 'image_width');
    $this->addSearchParameter($query, 'carousel_autoplay');
    $this->addSearchParameter($query, 'show_indicators');
    $this->addSearchParameter($query, 'show_captions');
    $this->addSearchParameter($query, 'show_caption_title');
    $this->addSearchParameter($query, 'show_caption_background');
    $this->addSearchParameter($query, 'caption_font_size');
    $this->addSearchParameter($query, 'show_controls');
    $this->addSearchParameter($query, 'status_id');

    // filter by gender name

    $query->joinWith(['status' => function ($q) {
        $q->andFilterWhere(['=' , 'status.status_name', $this->statusName]);
    }]);
}

return $dataProvider;
}

protected function addSearchParameter($query, $attribute, $partialMatch = false) { if (($pos = strpos($attribute, '.')) !== false) { $modelAttribute = substr($attribute, $pos + 1); } else { $modelAttribute = $attribute; }

$value = $this->$modelAttribute;
if (trim($value) === '') {
    return;
}

/*
* The following line is additionally added for right aliasing
* of columns so filtering happen correctly in the self join
*/

```

```

$attribute = "carousel_settings.$attribute";

if ($partialMatch) {
    $query->andWhere(['like', $attribute, $value]);
} else {
    $query->andWhere([$attribute => $value]);
}

}

public static function getCarouselSettings($carousel_name)
{
    $carouselSettings = CarouselSettings::find()
        ->where(['carousel_name' => $carousel_name])
        ->one();

    return $carouselSettings;
}

}

```

Most of `this` is boilerplate, but we `do` have the `getCarouselSettings` method to return an AR instance holding our settings. You'll note that we pass `$carousel_name` into the method signature **and** also that it's `static`, so we could call it like so:

```
$carouselSettings = CarouselSettingsSearch::getCarouselSettings('Front Page');
```

We could then pass `$carouselSettings` into a view via a controller **and** have access to all the settings.

```
## CarouselSettingsController
```

Gist:

```
[CarouselSettings Controller](https://gist.github.com/evercode1/ca82b018435340d0\1076)
```

From book:

```
<?php

namespace backend\controllers;

use Yii; use backend\models\CarouselSettings; use backend\models\search\CarouselSettingsSearch;
use yii\web\Controller; use yii\web\NotFoundHttpException; use yii\filters\VerbFilter; use common\models\PermissionHelpers;

/** * CarouselSettingsController implements the CRUD actions for CarouselSettings model. */
class CarouselSettingsController extends Controller {
    public function behaviors() {
        return [
            'access' => [
                'class' => \yii\filters\AccessControl::className(),
                'only' => ['index', 'view', 'create', 'update', 'delete'],
                'rules' => [
                    [
                        'actions' => ['index', 'view', 'create', 'update', 'delete'],
                        'allow' => true,
                        'roles' => ['@'],
                        'matchCallback' => function ($rule, $action) {
                            return PermissionHelpers::requireMinimumRole('Admin')
                                && PermissionHelpers::requireStatus('Active');
                        }
                    ],
                ],
            ],
            'verbs' => [
                'class' => VerbFilter::className(),
                'actions' => [
                    'delete' => ['post'],
                ],
            ],
        ];
    }

    /**
     * Lists all CarouselSettings models.
     * @return mixed
     */
}

public function actionIndex()
{
```

```
$searchModel = new CarouselSettingsSearch();
$dataProvider = $searchModel->search(Yii::$app->request->queryParams);

return $this->render('index', [
    'searchModel' => $searchModel,
    'dataProvider' => $dataProvider,
]);
}

/**
 * Displays a single CarouselSettings model.
 * @param integer $id
 * @return mixed
 */
public function actionView($id)
{
    return $this->render('view', [
        'model' => $this->findModel($id),
    ]);
}

/**
 * Creates a new CarouselSettings model.
 * If creation is successful, the browser will be
 * redirected to the 'view' page.
 * @return mixed
 */
public function actionCreate()
{
    $model = new CarouselSettings();

    if ($model->load(Yii::$app->request->post()) && $model->save()) {

        return $this->redirect(['view', 'id' => $model->id]);
    } else {
        return $this->render('create', [
            'model' => $model,
        ]);
    }
}

/**
```

```
* Updates an existing CarouselSettings model.
* If update is successful, the browser will be
* redirected to the 'view' page.
* @param integer $id
* @return mixed
*/
public function actionUpdate($id)
{
    $model = $this->findModel($id);

    if ($model->load(Yii::$app->request->post()) && $model->save()) {
        return $this->redirect(['view', 'id' => $model->id]);
    } else {
        return $this->render('update', [
            'model' => $model,
        ]);
    }
}

/**
* Deletes an existing CarouselSettings model.
* If deletion is successful, the browser will be
* redirected to the 'index' page.
* @param integer $id
* @return mixed
*/
public function actionDelete($id)
{
    $this->findModel($id)->delete();

    return $this->redirect(['index']);
}

/**
* Finds the CarouselSettings model based on its primary key value.
* If the model is not found, a 404 HTTP exception will be thrown.
* @param integer $id
* @return CarouselSettings the loaded model
* @throws NotFoundHttpException if the model cannot be found
*/
protected function findModel($id)
{
```

```

    if (($model = CarouselSettings::findOne($id)) !== null) {
        return $model;
    } else {
        throw new NotFoundHttpException('The requested page does not exist.');
    }
}

}

```

All we're doing there is checking `for` admin permission, the rest is straight from Gii.

`## CarouselSettings _form View`

Gist:

[CarouselSettings _form view](<https://gist.github.com/evercode1/31aa2eb2bf594669290e>)

From book:

```

<?php
use yii\helpers\Html; use yii\widgets\ActiveForm;
/* @var $this yii\web\View // @var $model backend\models\CarouselSettings // @var $form yii\widgets\ActiveForm */ ?>
<div class="carousel-settings-form">

<?php $form = ActiveForm::begin(); ?>

<?= $form->field($model, 'carousel_name')->textInput(['maxlength' => 45]) ?>
<?= $form->field($model, 'image_height')->textInput(['maxlength' => 45]) ?>
<?= $form->field($model, 'image_width')->textInput(['maxlength' => 45]) ?>
<?= $form->field($model, 'carousel_autoplay')
->dropDownList($model->carouselAutoplayList,
['prompt' => 'Please Choose One']); ?>
<?= $form->field($model, 'show_indicators') ?>

```

```
->dropDownList($model->showIndicatorsList,
['prompt' => 'Please Choose One']);?>

<?= $form->field($model, 'show_caption_title')
->dropDownList($model->showCaptionTitleList,
['prompt' => 'Please Choose One']);?>

<?= $form->field($model, 'show_captions')
->dropDownList($model->showCaptionsList,
['prompt' => 'Please Choose One']);?>

<?= $form->field($model, 'show_caption_background')
->dropDownList($model->showCaptionBackgroundList,
['prompt' => 'Please Choose One']);?>

<?= $form->field($model, 'caption_font_size')
->textInput(['maxlength' => 45]) ?>

<?= $form->field($model, 'show_controls')
->dropDownList($model->showControlsList,
['prompt' => 'Please Choose One']);?>

<?= $form->field($model, 'status_id')
->dropDownList($model->statusList);?>

<div class="form-group">
    <?= Html::submitButton($model->isNewRecord ? 'Create' : 'Update',
['class' => $model->isNewRecord ? 'btn btn-success' : 'btn btn-primary']) ?> </div>

<?php ActiveForm::end(); ?>

</div>
```

We just cleaned up the form with dropdown lists, like we always do.

```
## CarouselSettings view.php View
```

Gist:

```
[CarouselSettings view.php](https://gist.github.com/evercode1/e019c9f96f7e3fb54d\10)
```

From book:

```
<?php
use yii\helpers\Html; use yii\widgetsDetailView;
/* @var $this yii\web\View // @var $model backend\models\CarouselSettings */
$this->title = $model->carousel_name; $this->params['breadcrumbs'][] = ['label' => 'Carousel Settings', 'url' => ['index']]; $this->params['breadcrumbs'][] = $this->title . ' Carousel'; ?> <div class="carousel-settings-view">

<h1><?= Html::encode($model->carousel_name) ?> Carousel</h1>

<p>
<?= Html::a('Update', ['update', 'id' => $model->id],
['class' => 'btn btn-primary']) ?>
<?= Html::a('Delete', ['delete', 'id' => $model->id], [
'class' => 'btn btn-danger',
'data' => [
'confirm' => 'Are you sure you want to delete this item?',
'method' => 'post',
],
]) ?>
</p>

<?= DetailView::widget([
'model' => $model,
'attributes' => [
'id',
'carousel_name',
'image_height',
'image_width',
['attribute' => 'carousel_autoplay', 'format' => 'boolean'],
['attribute' => 'show_indicators', 'format' => 'boolean'],
]
```

```
[ 'attribute' => 'show_caption_title', 'format' => 'boolean'],
[ 'attribute' => 'show_captions', 'format' => 'boolean'],
[ 'attribute' => 'show_caption_background', 'format' => 'boolean'],
[ 'caption_font_size',
['attribute' => 'show_controls', 'format' => 'boolean'],
[ 'status.status_name',
[ 'created_at',
[ 'updated_at',
],
])
?>

</div>
```

Again, `this` should be familiar to us at `this` point. I'm providing it `for` reference, but you shouldn't need it.

`## CarouselSettings index.php View`

Gist:

[CarouselSettings index.php](<https://gist.github.com/evercode1/c95f52f7f7ca3b16\ac48>)

From book:

```
<?php
use yii\helpers\Html; use yii\grid\GridView;
/* @var $this yii\web\View // @var $searchModel backend\models\searchCarouselSettingsSearch /
@var $dataProvider yii\data\ActiveDataProvider */
$this->title = 'Carousel Settings'; $this->params['breadcrumbs'][] = $this->title; ?> <div class="carousel-
settings-index">
```

```
<h1><?= Html::encode($this->title) ?></h1>
<?php // echo $this->render('_search', ['model' => $searchModel]); ?>

<p>
    <?= Html::a('Create Carousel Settings', ['create'],
        ['class' => 'btn btn-success']) ?>
</p>

<?= GridView::widget([
    'dataProvider' => $dataProvider,
    'filterModel' => $searchModel,
    'columns' => [
        ['class' => 'yii\grid\SerialColumn'],

        'id',
        'carousel_name',
        'image_height',
        'image_width',
        ['attribute' => 'carousel_autoplay', 'format' => 'boolean'],
        ['attribute' => 'show_indicators', 'format' => 'boolean'],
        ['attribute' => 'show_caption_title', 'format' => 'boolean'],
        ['attribute' => 'show_captions', 'format' => 'boolean'],
        ['attribute' => 'show_caption_background', 'format' => 'boolean'],
        'caption_font_size',
        ['attribute' => 'show_controls', 'format' => 'boolean'],

        'statusName',
        // 'created_at',
        'updated_at',

        ['class' => 'yii\grid\ActionColumn'],
    ],
]); ?>

</div>
```

This has the same minor formatting changes that we always have on index.php.

```
## CarouselSettings _search View
```

Gist:

[CarouselSettings _search View](<https://gist.github.com/evercode1/3ce3db25bb4395\3a54b9>)

From book:

```
<?php
use yii\helpers\Html; use yii\widgets\ActiveForm;
/* @var $this yii\web\View // @var $model backend\models\searchCarouselSettingsSearch // @var
$form yii\widgets\ActiveForm */ ?>
<div class="carousel-settings-search">

<?php $form = ActiveForm::begin([
    'action' => ['index'],
    'method' => 'get',
]); ?>

<?= $form->field($model, 'id') ?>

<?= $form->field($model, 'carousel_name')->textInput(['maxlength' => 45]) ?>

<?= $form->field($model, 'image_height')->textInput(['maxlength' => 45]) ?>

<?= $form->field($model, 'image_width')->textInput(['maxlength' => 45]) ?>

<?= $form->field($model, 'carousel_autoplay')
    ->dropDownList($model->carouselAutoplayList,
        ['prompt' => 'Please Choose One']); ?>

<?= $form->field($model, 'show_indicators')
    ->dropDownList($model->showIndicatorsList,
        ['prompt' => 'Please Choose One']); ?>

<?= $form->field($model, 'show_caption_title')
    ->dropDownList($model->showCaptionTitleList,
```

```

['prompt' => 'Please Choose One']);?>

<?= $form->field($model, 'show_captions')
 ->dropDownList($model->showCaptionsList,
 ['prompt' => 'Please Choose One']);?>

<?= $form->field($model, 'show_caption_background')
 ->dropDownList($model->showCaptionBackgroundList,
 ['prompt' => 'Please Choose One']);?>

<?= $form->field($model, 'caption_font_size')
 ->textInput(['maxlength' => 45]) ?>

<?= $form->field($model, 'show_controls')
 ->dropDownList($model->showControlsList,
 ['prompt' => 'Please Choose One']);?>

<?= $form->field($model, 'status_id')->dropDownList($model->statusList);?>

<?php echo $form->field($model, 'updated_at') ?>

<div class="form-group">
    <?= Html::submitButton('Search', ['class' => 'btn btn-primary']) ?>
    <?= Html::resetButton('Reset', ['class' => 'btn btn-default']) ?>
</div>

<?php ActiveForm::end(); ?>

</div>

```

We're **not** using search, especially since we will only have one record, but I **for\matted** it anyway, in **case** things change in the future **and** we find ourselves sear\ching **for** carousels.

PagesController

Gist:

[Pages Controller](<https://gist.github.com/evercode1/b3a3c7351b007c9007e6>)

From book:

<?php

```
namespace frontend\controllers;

use Yii; use frontend\models\ContactForm; use yii\filters\AccessControl; use backend\models\searchCarouselSettings;
use yii\web\Controller;
use yii\filters\VerbFilter;
use yii\rest\ActiveController;

class PagesController extends \yii\web\Controller {

    public function behaviors()
    {
        return [
            'access' => [
                'class' => AccessControl::className(),
                'only' => ['captcha'],
                'rules' => [
                    [
                        'actions' => ['captcha'],
                        'allow' => true,
                        'roles' => ['?', '@'],
                    ],
                ],
            ],
        ];
    }

    public function actions()
    {
        return [
            'error' => [
                'class' => 'yii\web\ErrorAction',
            ],
            'captcha' => [
                'class' => 'yii\captcha\CaptchaAction',
                'fixedVerifyCode' => YII_ENV_TEST ? 'testme' : null,
            ],
        ];
    }

    public function actionIndex()
    {

        $carouselSettings = CarouselSettingsSearch::getCarouselSettings('Front Page');
    }
}
```

```
return $this->render('index', ['carouselSettings' => $carouselSettings]);  
}  
  
public function actionAbout()  
{  
    return $this->render('about');  
}  
  
public function actionContact()  
{  
    $model = new ContactForm();  
    if ($model->load(Yii::$app->request->post()) && $model->validate()) {  
        if ($model->sendEmail(Yii::$app->params['adminEmail'])) {  
            Yii::$app->session->setFlash('success', 'Thank you for contacting  
us. We will respond to you as soon as possible.');//  
        } else {  
            Yii::$app->session->setFlash('error', 'There was an error sending email.');//  
        }  
        return $this->refresh();  
    } else {  
        return $this->render('contact', [  
            'model' => $model,  
        ]);  
    }  
}  
  
public function actionPrivacy()  
{  
    return $this->render('privacy');  
}  
  
public function actionTermsService()  
{  
    return $this->render('terms-service');  
}  
}
```

The main thing to take note of here is the index action:

```
public function actionIndex() {  
    $carouselSettings = CarouselSettingsSearch::getCarouselSettings('Front Page');  
  
    return $this->render('index', ['carouselSettings' => $carouselSettings]);  
}
```

Here we are calling our getCarouselSettings method, so we can pass it to Pages index view.

Pages Index View

Since at **this** point we're only modifying the widget, I'm just going to supply you with that change.

Gist:

[Pages Index View Change] (<https://gist.github.com/evercode1/f594cf08ef4310398d7d>)

From book:

```
<?= CarouselWidget::widget(['settings' => ['height' => $carouselSettings['image_height'], 'width' => $carouselSettings['image_width'], 'autoplay' => $carouselSettings['carousel_autoplay'], 'show_indicators' => $carouselSettings['show_indicators'], 'show_captions' => $carouselSettings['show_captions'], 'show_controls' => $carouselSettings['show_controls'], 'show_caption_title' => $carouselSettings['show_caption_title']],])?>
```

That takes us to the widget itself.

CarouselWidget

Gist:

[CarouselWidget] (<https://gist.github.com/evercode1/18bb213c03189cc19fd2>)

From book:

```
<?php namespace components;

use yii\baseWidget; use yii\helpersHtml; use Yii; use backend\modelsMarketingImage; use bac-
kend\models\searchMarketingImageSearch; use yii\webNotFoundException; use yii\filtersVerbFilter;
class CarouselWidget extends Widget {

    public $activeImage;
    public $images;
    public $count;
    public $settings = [];

    public function init()
    {
        parent::init();

        $this->activeImage = MarketingImage::find('marketing_image_path')
            ->where(['marketing_image_is_active' => 1])
            ->andWhere(['marketing_image_is_featured' => 1])
            ->andWhere(['status_id' => 1])
            ->one();

        $this->images = MarketingImage::find()
            ->where(['marketing_image_is_active' => 0])
            ->andWhere(['marketing_image_is_featured' => 1])
            ->andWhere(['status_id' => 1])
            ->orderBy('marketing_image_weight')
            ->all();

        $this->count = MarketingImage::find()
            ->where(['marketing_image_is_active' => 0])
            ->andWhere(['marketing_image_is_featured' => 1])
            ->andWhere(['status_id' => 1])
            ->count();

        $this->setDefaults();
        $this->validateSize();
    }

    public function setDefaults()
    {
```

```
if(!isset($this->settings['height'])){

    $this->settings['height'] = '300px';
}

if(!isset($this->settings['width'])){

    $this->settings['width'] = '700px';
}

if(!isset($this->settings['caption_font_size'])){

    $this->settings['caption_font_size'] = '15px';
}

if(!isset($this->settings['autoplay'])){

    $this->settings['autoplay'] = true;
}

if(!isset($this->settings['show_indicators'])){

    $this->settings['show_indicators'] = true;
}

if(!isset($this->settings['show_captions'])){

    $this->settings['show_captions'] = false;
}

if(!isset($this->settings['show_caption_background'])){

    $this->settings['show_caption_background'] = false;
}

if(!isset($this->settings['show_caption_title'])){

    $this->settings['show_caption_title'] = false;
}

if(!isset($this->settings['show_controls'])){
```

```
$this->settings['show_controls'] = true;  
}  
  
}  
  
public function validateSize()  
{  
  
    if (!preg_match("/px/", $this->settings['width'])  
    or !preg_match("/px/", $this->settings['height'])) {  
  
        throw new NotFoundHttpException  
            ('You Must Use px and number for size, example 300px');  
  
    }  
  
    $height = (int) preg_replace("/[^0-9]/", "", $this->settings['height']);  
  
    switch ($height){  
  
        case $height < 40 :  
            throw new NotFoundHttpException('You Must Stay within  
40 to 1000 px and use px and number for size, example 300px');  
        break;  
  
        case $height > 1000 :  
            throw new NotFoundHttpException('You Must Stay within  
40 to 1000 px and use px and number for size, example 300px');  
        break;  
  
    }  
  
    $width = (int) preg_replace("/[^0-9]/", "", $this->settings['width']);  
  
    switch ($width){  
  
        case $width < 40 :  
            throw new NotFoundHttpException('You Must Stay within  
40 to 1000 px and use px and number for size, example 300px');
```

```

        throw new NotFoundHttpException('You Must Stay within
40 to 1000 px and use px and number for size, example 300px');
break;

case $width > 1000 :
throw new NotFoundHttpException('You Must Stay within
40 to 1000 px and use px and number for size, example 300px');
break;

}

}

public function run()
{
    return $this->render('carousel', ['activeImage' => $this->activeImage,
                                         'images' => $this->images,
                                         'count' => $this->count,
                                         'settings' => $this->settings]);
}
}

```

Obviously we have more defaults in our `setDefaults` method. Also, a reminder that some of the settings are shortened, like in the `case` of height **and** width. Their names are different in the model **and** DB, but you can call them what you want in the widget, **as long as** you remember what you named them.

`## carousel.php`

That brings us to `carousel.php`.

Gist:

[`carousel.php`] (<https://gist.github.com/evercode1/b66717d10227e5db8dd0>)

From book:

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

```
?>

<div id="carouselMain" class="carousel slide"

<?php

if($settings['autoplay'] == false ){

    echo 'data-interval="false"';
}

?>

data-ride="carousel">

<?php

if ($settings['show_indicators']){

echo '<ol class="carousel-indicators">
    <li data-target="#carouselMain" data-slide-to="0"
        class="active"></li>';

foreach (range(1, $count) as $number) {

    echo '<li data-target="#carouselMain" data-slide-to="'.
        $number.'"></li>';

}

echo '</ol> ';



?>

<div class="carousel-inner" role="listbox">
```

```
<?php

    $width = $settings['width'];
    $height = $settings['height'];

    //active item first

    echo '<div class="item active">
        <center>' .

        Html::img(Yii::$app->urlManagerBackend->baseUrl . '/' .
            $activeImage['marketing_image_path'],
            ['width' => $width, 'height' => $height ])

        . '</center>';

    if($settings['show_captions']){
        echo '<div class="carousel-caption">';

        if ($settings['show_caption_title']){
            echo '<div><h1>' .
                $activeImage['marketing_image_caption_title'] .
                '</h1></div>';

        }

        echo $activeImage['marketing_image_caption'] . '</div>';

    }

    echo '</div>';

    //all other images

    foreach ($images as $image){

        echo '<div class="item">
            <center>' .
            Html::img(Yii::$app->urlManagerBackend->baseUrl .
            '/' . $image['marketing_image_path'],
```

```
[ 'width' => $width, 'height' => $height ])
. '</center>';

if($settings['show_captions']){
    echo '<div class="carousel-caption">';

    if ($settings['show_caption_title']){
        echo '<div><h1>' .
            $image['marketing_image_caption_title'] .
        '</h1></div>';

    }

    echo $image['marketing_image_caption'] . '</div>';

}
echo '</div>';

}

?>

<!-- end dynamic slide data -->

</div>
<?php
if ($settings['show_controls']){
    echo '<a class="left carousel-control"
        href="#carouselMain" role="button"
        data-slide="prev">
        <span class="glyphicon glyphicon-chevron-left"
        aria-hidden="true"></span>
        <span class="sr-only">Previous</span>
    </a>

    <a class="right carousel-control"
        href="#carouselMain" role="button"
        data-slide="next">
        <span class="glyphicon glyphicon-chevron-right"
        aria-hidden="true"></span>
        <span class="sr-only">Next</span>
    </a>
}
```

```

    data-slide="next">
    <span class="glyphicon glyphicon-chevron-right"
        aria-hidden="true"></span>
    <span class="sr-only">Next</span>
    </a>';
}

?>

</div>

```

The really simple explanation to that code is we simply added a bunch of **if** statements that rely on our settings **and** dynamic data. For example:

```
if($settings['show_captions']){
```

So **if** 'show_captions' is **true**, we get to see the captions. Then nested further down, we test **for** the caption title:

```
if ($settings['show_caption_title']){
```

So the **if** statements in **this** code control:

```
* autoplay
* show_indicators
* show_captions
* show_caption_title
* show_controls
```

So how **do** we control caption_font_size **and** show_caption_background?

Those are CSS elements. There may be more than one way to dynamically control C\SS, but the way I chose to **do** it was to place a small amount of style code in fr\ontend/views/layouts/main.php:

```
## Main
```

I'll give you the entire **file** first, then we'll focus on the relevant bits.

Gist:

[layouts/main.php](<https://gist.github.com/evercode1/9bd6fa727f6ec477cebc>)

Froom book:

```
<?php use yii\helpers\Html; use yii\bootstrapNav; use yii\bootstrapNavBar; use yii\widgetsBreadcrumbs;
use frontend\assetsAppAsset; use frontend\widgetsAlert; use frontend\assetsFontAwesomeAsset;
use backend\models\searchCarouselSettingsSearch;

/* @var $this \yii\webView // @var $content string */

AppAsset::register($this); FontAwesomeAsset::register($this);

?> <?php $this->beginPage() ?> <!DOCTYPE html> <html lang="<?= Yii::$app->language ?>">
<head> <meta charset="<?= Yii::$app->charset ?>"/> <meta name="viewport" content="width=device-
width, initial-scale=1"> <?= Html::csrfMetaTags() ?> <title><?= Html::encode($this->title) ?></title>
<?php

$carouselSettings = CarouselSettingsSearch::getCarouselSettings('Front Page');

if($carouselSettings['caption_font_size']){
    echo '<style>.carousel-caption{';
    if ($carouselSettings['show_caption_background']){
        echo 'background: rgba(0,0,0,0.5);';
    }
    echo 'font-size:' . $carouselSettings['caption_font_size'] . ';
    }
    '</style>';
}

?>

<?php $this->head() ?> </head> <body> <?php $this->beginBody() ?>

<div class="wrap">
<?php

    NavBar::begin([
        'brandLabel' => 'Yii 2 Start <i class="fa fa-plug"></i>',
        'brandUrl' => Yii::$app->homeUrl,
        'options' => [
            'class' => 'navbar-inverse navbar-fixed-top',
        ],
    ],

```

```
]);
```

```
$menuItems = [
    ['label' => 'Home', 'url' => ['/site/index']],
    ['label' => 'About', 'url' => ['/pages/about']],
    ['label' => 'FAQs', 'url' => ['/faq/index']],
    ['label' => 'Contact', 'url' => ['/pages/contact']],
];
```

```
if (Yii::$app->user->isGuest) {
    $menuItems[] = ['label' => 'Signup', 'url' => ['/site/signup']];
    $menuItems[] = ['label' => 'Login', 'url' => ['/site/login']];
}
```

```
} else {
    $menuItems[] = ['label' => 'Social Sync', 'items' => [
        ['label' => '<i class="fa fa-facebook"></i> Facebook',
         'url' => ['site/auth', 'authclient' => 'facebook']],
        ['label' => '<i class="fa fa-github"></i> Github',
         'url' => ['site/auth', 'authclient' => 'github']],
        ['label' => '<i class="fa fa-twitter"></i> Twitter',
         'url' => ['site/auth', 'authclient' => 'twitter']],
        ['label' => '<i class="fa fa-linkedin"></i> LinikedIn',
         'url' => ['site/auth', 'authclient' => 'linkedin']],
        ['label' => '<i class="fa fa-google"></i> Google+',
         'url' => ['site/auth', 'authclient' => 'google']],
    ]];
}
```

```
$menuItems[] = ['label' => 'Profile', 'url' => ['/profile/view']];
$menuItems[] = [
    'label' => 'Logout (' . Yii::$app->user->identity->username . ')',
    'url' => ['/site/logout'],
    'linkOptions' => ['data-method' => 'post']
];
}
```

```
echo Nav::widget([
```

```

'options' => ['class' => 'navbar-nav navbar-right'],
'items' => $menuItems,
'encodeLabels' => false,
]);

NavBar::end();
?>

<div class="container">
<?= Breadcrumbs::widget([
    'links' => isset($this->params['breadcrumbs']) ? 
        $this->params['breadcrumbs'] : [],
]) ?>
<?= Alert::widget() ?>
<?= $content ?>
</div>
</div>

<footer class="footer">
    <div class="container">
        <p class="pull-left">&copy; My Company <?= date('Y') ?></p>
        <p class="pull-right"><?= Yii::powered() ?></p>
    </div>
</footer>

<?php $this->endBody() ?>

</body> </html> <?php $this->endPage() ?>

```

So to take control of the desired CSS elements, we first have to **include** our **use**\ statement:

```
use backend\models\searchCarouselSettingsSearch;
```

Next we added a block of PHP that will **echo** out the style tags with our **if** state\ments controlling the scenario:

```
<?php
$carouselSettings = CarouselSettingsSearch::getCarouselSettings('Front Page');
```

```

if($carouselSettings['caption_font_size']){
    echo '<style>.carousel-caption{';
    if ($carouselSettings['show_caption_background']){
        echo 'background: rgba(0,0,0,0.5);';
    }
    echo 'font-size:' . $carouselSettings['caption_font_size'] . 'px';
}
echo '</style>';
}

?>

```

So obviously, we are getting the relevant settings:

```
$carouselSettings = CarouselSettingsSearch::getCarouselSettings('Front Page');
```

Then **if** a caption_font_size has been set, we enforce that **as well as** check **for** the show_caption_background setting. **If** there is no caption_font_size selected, there will be no background.

Note that the caption title is controlled by an h1 tag, **and** you could make a setting **for** that **and** control it in a similar fashion.

And that's pretty much it **for** our control over the carousel. Play around with the different settings **and** have fun.

Summary

This chapter turned out to be extensive. We built our basic image management that lets us upload, edit **and** delete images **and** their corresponding thumbnails.

Along the way we learned about scenarios **and** we used yii2-imagine extension to make thumbnail creation **as simple as** it can be. But since we're building a template, we wanted to immediately put our image management capability to good **use**. So we decided to build a dynamically-driven carousel that would display marketing images.

It's often very important **for** our clients to have control over things like their\ marketing message, so we also made a full UI **for** carousel settings that are res\ pected by the custom widget we built. **This** gives a lot of control over the caro\ usel directly to the client **and** they will love that.

While I was writing **this** chapter, some of our readers made nice comments, along \ with positive reviews **and** ratings on Goodreads.com. That really inspired me to \ work **as** hard **as** I could to deliver all the features in the carousel, which took \ multiple revisions on my part. So please keep all the comments **and** reviews comi\ ng, I really appreciate it.

Thanks once again **for** taking the Yii 2 journey with me. More will be coming. I\ hope to see you soon.

```
{bump-link-number}
```

```
{leanpub-filename="capitulo16.txt"}
```

```
# Chapter 16: Bonus Material Ratings Widget
```

We're back again **for** more fun with Yii 2. I hope you are enjoying the bonus mat\ erial.

I was looking at Kartik's widgets at:

[Krajee.com](<http://www.krajee.com>)

And I thought his big yii2-widgets extension would come in very handy **for** future\ projects.

I counted 24 widgets on the Git page, all available in one install, so **this** is 1\ ike hitting the jackpot. **And** because you have them all installed at once, you d\ on't have to install them individually, which saves you a lot of **time**.

If you are unfamiliar with that package, I suggest you take a look:

[Yii2 Widgets](<https://github.com/kartik-v/yii2-widgets>)

I actually had my eye on several of Kartik's widgets to implement **for** the templa\ te, but I wanted to start with something simple to get started. So I chose to i\ mplement the StarRating widget.

This provides a nice bootstrap rating scale that we can implement on our Faq model, since that is the only list of records in our application that faces the public and is something they could rate.

It's totally up to you if you would use this on your actual template, so this is optional of course. But this is a good way to get familiar with it.

I thought this would be super-easy to implement since I had such good luck with Kartik's social widget. Well, it got a little complicated, not because of the widget itself, which is a snap to install, but because we end up with a slightly more complex scenario than what we have been used to.

And because of that, it makes for a perfect lesson.

So let's start by adding the following to your composer.json file:

```
"kartik-v/yii2-widgets": "*",
```

Then run composer update on the command line:

```
! [composer update](images/originals/composerupdate.png)
```

Next we need to decide on a data structure. I decided to call my new table faq_rating:

```
! [faq_rating table](images/originals/faqratingtable.png)
```

Here is the SQL:

```
CREATE TABLE IF NOT EXISTS yii2build.faq_rating ( id INT(11) NOT NULL AUTO_INCREMENT, user_id INT(11) NOT NULL, faq_id INT(11) NOT NULL, faq_rating DOUBLE NOT NULL, created_at DATETIME NOT NULL, updated_at DATETIME NOT NULL, PRIMARY KEY (id))
```

You'll note that we chose double **as** our datatype **for** 'faq_rating' **and this** allow\ s us to have the decimal in the rating such **as** 2.5 etc.

Next we need to create the model from Gii. I'm going to put the model in the ba\ckend. Lately I've decided to put models in the backend, unless I have good rea\son to **do** otherwise.

You are of course free to structure it how you wish. **As long as** you **namespace** e\verything properly, you have a lot of flexibility in your choices.

You may have noticed that our table references to other primary keys, user_id an\d faq_id. **This** is also known **as** a junction table **or** a pivot table. **If** we neede\l to reference the relationships, they would be many to many relationships.

But with a rating **system**, at least our implementation of it, we don't really nee\l to reference it that way, so we didn't bother creating foreign keys **or** setting\ up the relationships.

If you wanted to get more complex, **for** example, **if** you wanted to write a report \ that showed all the faq_ratings made by a specific user, then the relationships \ would be important. But we are **not** anticipating that **use case for this**, so that\ 's just something to keep in mind **if** you **do** a more complex integration with anot\her model.

For our purposes here, we only need to store the ratings so we can calculate an \ average, store the faq_id the rating applies to, **and** store the user_id so that p\ople don't rate more than once.

We will also allow people to update their ratings. Alternatively, you could pre\vent them from doing so **if** you wish. I will point out where you could **do** that l\ater **if** you want to go in that direction.

So now we have to imagine our implementation. How does the rating fit in with t\he Faqs?

Ultimately, we want it to look like **this**:

! [Faq with Rating](images/originals/faqrating1.png)

So **if** you land on an Faq view page, you see the rating below the faq.

And if you click the Add Your Rating button:

! [Add Your Rating](images/originals/faqrating2.png)

So once we click on Add Your Rating, we get the above screen. If you mouse over the stars, they fill in, then you push the rate it button and the rating gets recorded.

So this seems incredibly easy, until you realize we are combining models, using both the Faq model and the FaqRating model.

That raises all kinds of interesting questions about what controller is going to do what, as well as how to mix the models onto a single view page, since the StarRating widget is going to pass us a value that we need move along via a form.

Taking a clue from Yii 2 itself, it seems like forms are best created as partials, so our plan will be to create a form partial for our FaqRating model and control form submission through our soon-to-be-made FaqRatingController.

But first, let's look at the model.

FaqRatings Model

Gist:

[FaqRatings Model](<https://gist.github.com/evercode1/c69f379f6d4e884ec1f5>)

From book:

```
<?php  
namespace backend\models;  
use Yii; use yii\db\ActiveRecord; use yii\dbExpression; use yii\helpers\ArrayHelper; use kartik\widgets\StarRating;  
/** * This is the model class for table "faq_rating". */  
@property integer $id  
@property integer $user_id  
@property integer $faq_id  
@property double $faq_rating  
@property string $created_at  
@property string $updated_at  
*/  
class FaqRating extends \yii\db\ActiveRecord {  
/** * @inheritdoc */  
public static function tableName() { return 'faq_rating'; }  
}
```

```
public function behaviors()
{
    return [
        'timestamp' => [
            'class' => 'yii\behaviors\TimestampBehavior',
            'attributes' => [
                ActiveRecord::EVENT_BEFORE_INSERT => ['created_at', 'updated_at'],
                ActiveRecord::EVENT_BEFORE_UPDATE => ['updated_at'],
                ],
            'value' => new Expression('NOW()'),
            ],
        ];
}

public function beforeValidate()
{
    $this->faq_rating = (double)$this->faq_rating;
    $this->faq_id = (int)$this->faq_id;

    return parent::beforeValidate();
}

/**
 * @inheritDoc
 */
public function rules()
{
    return [
        [['user_id', 'faq_id', 'faq_rating'], 'required'],
        [['user_id', 'faq_id'], 'integer'],
        [['faq_rating'], 'double'],
        [['created_at', 'updated_at'], 'safe']
    ];
}

/**
 * @inheritDoc
 */
public function attributeLabels()
{
    return [
        'id' => 'ID',
    ];
}
```

```
'user_id' => 'User ID',
'faq_id' => 'Faq ID',
'faq_rating' => 'Faq Rating',
'created_at' => 'Created At',
'updated_at' => 'Updated At',
];
}

public function showAverageRating($faq_id)
{
    $averageRating = $this->getAverageRating($faq_id);

    echo StarRating::widget([
        'name' => 'rating_' . $averageRating,
        'value' => $averageRating,
        'disabled' => true,
        'pluginOptions' => [
            'size' => 'sm',
            'stars' => 5,
            'min' => 0,
            'max' => 5,
            'step' => 0.5,
            // 'symbol' => html_entity_decode('&#xe005;', ENT_QUOTES, "utf-8"),
            // 'defaultCaption' => '{rating} hearts',
            'starCaptions'=>[]
        ]
    ]);

}

public function getAverageRating($faq_id)
{
    $ratings = FaqRating::find('faq_rating')->asArray()
        ->where(['faq_id' => $faq_id])
        ->all();
}
```

```

$ratings = ArrayHelper::map($ratings, 'id', 'faq_rating');

$ratingsSum = array_sum($ratings);

$ratingsCount = count($ratings);

if($ratingsCount){

    $averageRating = $ratingsSum/$ratingsCount;

} else {

    $averageRating = 0;
}

return $averageRating;
}

}

```

So that is our change from the boilerplate. Obviously, we added timestamp behavior **and** the appropriate **use** statements to **use** it. We also modified our rules, removing ‘created_at’ **and** ‘updated_at’ from being required.

If you don’t remove the required rule **for** those columns, you get hard-to-diagnose validation errors. **This** happens when validation fails behind the scenes, **not** in an obvious way related to the form.

For example, **if** you don’t fill out a required field on a form, you don’t get to submit, **and** it makes it fairly obvious what the problem is. However, when the problem is **not** directly related to the form, things can get tricky.

Let’s say **for** example, you forget to make the table auto-increment **and** then you **use** Gii to create the Model **and** Crud. Gii will put the id **as** an input field because it’s **not** done automatically. Let’s say you see the id field, **not** realizing your mistake **and** you remove that field from the form, thinking it would be a simple fix.

What would happen in that scenario is that when you submit the form, you would get a blank white screen **and** no record inserted. More importantly, you would get no error message telling you what’s wrong.

```
## $model->getErrors()
```

Most programmers know how to **use var_dump()** to help figure out problems, but these validation errors can be a real pain to **sort** out. Fortunately, Yii 2 gives us a model method that returns the errors that you can **use** like so:

```
var_dump($model->getErrors()): die();
```

You can test the save method like so:

```
if (!$companyrecord->save()) { var_dump($companyrecord->getErrors()); }
```

In **this** example, **\$companyrecord** is the model.

In some cases where validation fails on saving a record, you could still have errors that don't show up in that **var_dump**, so there is another method.

```
## Overwrite Save Method on Model
```

You can overwrite the save method on the model to see **if** you can find the errors. I've had to **use this** several times **and** it's really handy:

```
public function save($runValidation = true, $attributeNames = NULL)
{
```

```
    if(!parent::save()){
        echo 'something is wrong';
        die();
        //var_dump($this->getErrors());
    }
}
```

```
}
```

You would add `this` method to whichever model you are trying to troubleshoot.

Just remember that after you `use this` to diagnose a problem that you should remove it in order to restore full functionality.

Anyway, those are some tips on how to overcome validation errors that are `not` otherwise visible to you.

Ok, let's get back to our `new` model, FaqRatings. In addition to the changes we already talked about, we have two `new` methods, `showAverageRating` `and` `getAverageRating`. Let's discuss the second one first.

```
public function getAverageRating($faq_id) {  
  
    $ratings = FaqRating::find('faq_rating')->asArray()  
        ->where(['faq_id' => $faq_id])  
        ->all();  
  
    $ratings = ArrayHelper::map($ratings, 'id', 'faq_rating');  
  
    $ratingsSum = array_sum($ratings);  
  
    $ratingsCount = count($ratings);  
  
    if($ratingsCount){  
  
        $averageRating = $ratingsSum/$ratingsCount;  
  
    } else {  
  
        $averageRating = 0;  
    }  
  
    return $averageRating;  
}
```

You can see we need the id of the faq in the signature, then we **use** ActiveRecord\\ to get all ratings that have an faq_id equal to the faq_id that we handed in. \\ We're taking the results **as** an **array** because its easy to work with these values \\ in **this** format.

Since we have all our ratings stored in our **\$ratings array**, we **use** ArrayHelper::\\ map to filter everything out except the two values we want, id **and** faq_rating. \\ So now we can **use**:

```
$ratingsSum = array_sum($ratings);
```

This will add up the total value of all ratings, which we can then divide by the\\ number of ratings, **for** which we need to **count** the ratings **array**:

```
$ratingsCount = count($ratings);
```

Then we check to see if there is a count, and if so do the calculation to return the average:

```
if($ratingsCount){  
    $averageRating = $ratingsSum/$ratingsCount;  
}  
else {  
    $averageRating = 0;  
}  
  
return $averageRating;
```

So that is how we get our average rating.

We've also built a method to show our average rating:

```

public function showAverageRating($faq_id)
{
    $averageRating = $this->getAverageRating($faq_id);

    echo StarRating::widget([
        'name' => 'rating_' . $averageRating,
        'value' => $averageRating,
        'disabled' => true,
        'pluginOptions' => [
            'size' => 'sm',
            'stars' => 5,
            'min' => 0,
            'max' => 5,
            'step' => 0.5,
            'starCaptions'=>[]
        ]
    ]);

}

```

In order to show the average rating, we need to get the average rating first, which is why we built the other method first. Once we have the average rating, we can simply plug it into the widget in the appropriate places, as shown above.

I don't usually echo a widget within a model method, but I did so this time to make things easy on us. From the Faq controller, I'll pass an instance of the FaqRatings model, which will make this method available to the view.

Some things to note about the widget. We can set the size to small via 'sm', the number of stars, in this case 5, and the step, which indicates a partial value on the star. You can reference Krajee.com if you want to know more about what settings you can use.

Faq Controller

We need to modify the view action as follows.

Gist:

[Frontend Faqcontroller View Action](#)

From book:

```

public function actionView($id, $slug = null)
{
    $model = $this->findModel($id);

    $faqRating = new FaqRating();

    if ($slug == $model->slug){

        return $this->render('view', [
            'model' => $model,
            'slug' => $model->slug,
            'faqRating' => $faqRating,
        ]);
    } else {
        throw new NotFoundHttpException('The requested Faq does not exist.');
    }
}

```

You can see all we did was make a new FaqRating instance available to the view.

Frontend Faq view.php

So let's look at the view.

Gist:

[Faq View](#)

From book:

```

<?php
use yii\helpers\Html;
use kartik\widgets\Growl;

$this->title = 'FAQ: '. $model->faq_question;

$this->params['breadcrumbs'][] = ['label' => 'FAQ', 'url' => ['index']];
$this->params['breadcrumbs'][] = $this->title;
?>

</br>
<div class="panel panel-default">

```

```
<div class="panel-heading">
    <h3 class="panel-title">

        <h1>    <?= $model->faq_question;?> </h1>

    </h3>
</div>

<?= '<div class="panel-body"><h3>' . $model->faq_answer . '</h3></div>';?>

</div>
<?php

if (Yii::$app->getSession()->hasFlash('success')){

    echo Growl::widget([
        'type' => Growl::TYPE_SUCCESS,
        'title' => 'Thank you!',
        'icon' => 'glyphicon glyphicon-ok-sign',
        'body' => Yii::$app->session->setFlash('success'),
        'showSeparator' => true,
        'delay' => 0,
        'pluginOptions' => [
            'placement' => [
                'from' => 'top',
                'align' => 'right',
            ]
        ]
    ]);
}

Yii::$app->getSession()->removeFlash('success');

?>
<div id="showAverage">
    <strong> Faq Rating </strong>
<?php

$faqRating->showAverageRating($model->id);

?>
<br>
```

```

<button type="button" id="rateMe" class="btn btn-default">
    Add Your Rating
</button>
</div>

<div id="rateIt">
<?php
echo $this->render('rating-form', ['model'=> $model,
    'faqRating' => $faqRating]);
?>
</div>
<?Php
$script = <<< JS
$(document).ready(function(){
    $("#rateIt").hide();
    $("#rateMe").click(function(){
        $("#showAverage").hide();
        $("#rateIt").show();
    });
});
};

JS;

$this->registerJs($script);

?>

```

Ok, so we added some new things here. Right below the faq answer, we added a growl widget:

```

<?php

if (Yii::$app->getSession()->hasFlash('success')){

    echo Growl::widget([
        'type' => Growl::TYPE_SUCCESS,
        'title' => 'Thank you!',
        'icon' => 'glyphicon glyphicon-ok-sign',
        'body' => Yii::$app->session->setFlash('success'),
        'showSeparator' => true,
        'delay' => 0,
        'pluginOptions' => [
            'placement' => [

```

```

        'from' => 'top',
        'align' => 'right',
    ]
]
]);
}

Yii::$app->getSession()->removeFlash('success');

?>

```

What the Growl::widget does is format a flash message into an animated message. I thought it added a nice touch to the UI and the widget is already part of the Kartik widget extension, so we already had everything we need for it. We will discuss it more in detail when we work on the controller.

Next we have the call to the showAverageRatings method:

```

<div id="showAverage">
    <strong> Faq Rating </strong>
    <?php

        $faqRating->showAverageRating($model->id);

?>
    <br>
    <button type="button" id="rateMe" class="btn btn-default">
        Add Your Rating
    </button>
</div>

```

You can see that we added a div id of showAverage. We also have a button within the div with an id of rateMe.

Next we have a call to a form partial, which will allow us to submit the rating:

```

<div id="rateIt">
    <?php
    echo $this->render('_rating-form', ['model'=> $model,
        'faqRating' => $faqRating]);
?>
</div>

```

Even without seeing the form yet, you can see that it should render the form, which is odd because we are also displaying the average rating. And yet from our screenshot, we know that it is only supposed to show the form when the add your rating button is clicked.

That brings us to our next section. We manage this by using show/hide in jquery:

```
<?Php
$script = <<< JS
$(document).ready(function(){
    $("#rateIt").hide();
    $("#rateMe").click(function(){
        $("#showAverage").hide();
        $("#rateIt").show();
    });
});

JS;

$this->registerJs($script);

?>
```

We're using heredoc notation to setup our javascript:

```
<?Php
$script = <<< JS
```

You can google heredoc if you are unfamiliar with that notation. It's a very convenient way to integrate PHP and javascript. At the end, we use:

```
$this->registerJs($script);
```

This allows our view to use the asset.

In the script itself, you can see it's pretty simple. As a default, on document ready, we hide anything in the rateIt div.

```
$(document).ready(function(){
    $("#rateIt").hide();
```

Then when someone clicks the Add Your Rating button, which has an id of rateMe, then we hide showAverage and we show the rateIt div:

```
$( "#rateMe" ).click( function(){
    $( "#showAverage" ).hide();
    $( "#rateIt" ).show();
});
```

You don't have to worry about getting back to showAverage from this point because the controller that processes our _rating-form will take care of that.

So this is a demonstration of how to embed javascript directly in the page. The other way to go is to put the file in a js folder under frontend/web and reference it in the AppAsset.php file.

_rating-form.php

Since we're calling _rating-form within this view, create that now. This file should go in the frontend/views/faq folder.

Gist:

[_rating-form.php](#)

From book:

```
<?php

use yii\helpers\Html;
use yii\widgets\ActiveForm;
use kartik\widgets\StarRating;

/* @var $this yii\web\View */
/* @var $model backend\models\FaqRating */
/* @var $form yii\widgets\ActiveForm */
?>

<div class="faq-rating-form">

    <?php $form = ActiveForm::begin([
        'method' => 'post',
        'action' => ['faq-rating/rating'],
    ]); ?>

    <?= Html::activeHiddenInput($faqRating, 'faq_id',
        [ 'value' => $model->id]) ?>

    <?= $form->field($faqRating, 'faq_rating') ?>
```

```

->label('Rate This FAQ')->widget(StarRating::classname(), [
    'pluginOptions' => [
        'size' => 'sm',
        'stars' => 5,
        'min' => 0,
        'max' => 5,
        'step' => 0.5,
        // 'symbol' => html_entity_decode('&#xe005;', ENT_QUOTES, "utf-8"),
        // 'defaultCaption' => '{rating} hearts',
        'starCaptions'=>[]
    ]
])?>

<div class="form-group">
    <?= Html::submitButton('Rate It!', ['class' => 'btn btn-success']) ?>
</div>

<?php ActiveForm::end(); ?>

</div>

```

Ok, just a few things to note. We are setting the action:

```

<?php $form = ActiveForm::begin([
    'method' => 'post',
    'action' => ['faq-rating/rating'],
]); ?>

```

Most of the forms we have done so far did not require us to set the action. But this one is not being processed by the FaqController, even though the form is sitting in the faq view folder.

Instead, we are sending this form to the FaqRatings Controller, which we have not created yet.

Ok, back to the form. After setting the action, we are using the StarRating widget for the form field:

```
<?= $form->field($faqRating, 'faq_rating')
    ->label('Rate This FAQ')->widget(StarRating::classname(), [
        'pluginOptions' => [
            'size' => 'sm',
            'stars' => 5,
            'min' => 0,
            'max' => 5,
            'step' => 0.5,
            // 'symbol' => html_entity_decode('&#xe005;', ENT_QUOTES, "utf-8"),
            // 'defaultCaption' => '{rating} hearts',
            'starCaptions'=>[]
        ]
    ])
])?>
```

This will take in the value the user gives it when they select their star rating and submit via the form when the submit button is pressed.

```
<div class="form-group">
    <?= Html::submitButton('Rate It!', ['class' => 'btn btn-success']) ?>
</div>

<?php ActiveForm::end(); ?>
```

We're using the Html helper submitButton method, and we give it a label, 'Rate It!', and a style. Note that this is different from what we have seen in most of our forms, something like:

```
<?= Html::submitButton($model->isNewRecord ? 'Create' :
    'Update', ['class' => $model->isNewRecord ?
    'btn btn-success' : 'btn btn-primary']) ?>
```

In the above example, we use the \$model->isNewRecord method to determine if it's a new record or not, and then show the appropriate button.

I couldn't get \$model->isNewRecord to work correctly in this case, and once I started to work around it, I realized this would be a good example for how to do it when that method is not available. This has implications in the controller, as we will see shortly.

FaqRatings Controller

We need to create the controller, and in this case, Gii is not going to be helpful, so you can just copy this file.

Gist:

FaqRatings Controller

From book:

```
<?php

namespace frontend\controllers;

use Yii;
use backend\models\FaqRating;
use yii\web\Controller;
use yii\web\NotFoundHttpException;
use yii\filters\VerbFilter;
use backend\models\Faq;
use yii\helpers\Html;
use yii\helpers\Url;

/**
 * FaqRatingController implements the CRUD actions
 * for FaqRating model.
 */

class FaqRatingController extends Controller
{
    public function behaviors()
    {
        return [
            'verbs' => [
                'class' => VerbFilter::className(),
                'actions' => [
                    'delete' => ['post'],
                ],
            ],
        ];
    }

    /**
     * Creates a new FaqRating model.
     * If creation is successful, the browser will
     * be redirected to
     * the 'view' page.
     * @return mixed
    
```

```
*/  
  
public function actionRating()  
{  
    if(Yii::$app->user->isGuest){  
  
        return $this->redirect(['site/login']);  
    }  
  
    $model = new FaqRating();  
    $model->user_id = (int) Yii::$app->user->identity->id;  
  
  
    if ($model->load(Yii::$app->request->post())) {  
  
        $existingRating = FaqRating::find()  
            ->where(['user_id' => $model->user_id])  
            ->andWhere(['faq_id' => $model->faq_id])  
            ->one();  
  
        if(isset($existingRating->id)){  
  
            $existingRating->faq_rating = $model->faq_rating;  
  
            $existingRating->update();  
  
  
            $slug = Faq::find('slug')  
                ->where(['id' => $existingRating->faq_id])  
                ->one();  
  
            Yii::$app->session->setFlash('success',  
                'Thank you for updating this Faq to '  
                . $existingRating->faq_rating.  
                ' stars. Your result is factored into the  
                average.');
```

```
    } else {

        if($model->save()) {

            $slug = Faq::find('slug')
                ->where(['id' => $model->faq_id])
                ->one();

            Yii::$app->session->setFlash('success',
                'Thank you for rating this Faq ' . $model->faq_rating.
                ' stars. Your result is factored into the average.');

            return $this->redirect(['faq/view',
                'id' => $model->faq_id, 'slug' => $slug->slug
            ]);

        }

    }

} else {

    throw new NotFoundHttpException('There was a problem');

}

}
```

We do start with some boilerplate on the behaviors method:

```

class FaqRatingController extends Controller
{
    public function behaviors()
    {
        return [
            'verbs' => [
                'class' => VerbFilter::className(),
                'actions' => [
                    'delete' => ['post'],
                ],
            ],
        ];
    }
}

```

By only allowing delete on post, we put up another defense against delete via url manipulation. If you don't specify this, someone might be able to manipulate a get variable in the url to delete records, which is obviously not good.

In this case, specifying post probably isn't necessary, since we don't have a delete action, but I'll leave it in just as a precaution.

The only other method we have in the controller is the actionRating method. Let's look at the first part:

```

public function actionRating()
{
    if(Yii::$app->user->isGuest){

        return $this->redirect(['site/login']);
    }
}

```

This simply requires login by the user in order to submit a rating. By doing so, we don't allow a user create multiple rating records for a single Faq, since the user must be logged in and we can check to see if they have already rated it.

Next instantiate a new FaqRating model, so we can assign the user_id property to the currently logged in user, since we're not handing that in with the form:

```

$model = new FaqRating();

$model->user_id = (int) Yii::$app->user->identity->id;

```

The (int) specifies that we want an int out of Yii::\$app->user->identity->id. I did that because when I var_dumped to test, I got string as an answer. The validation requires an int however.

Next we load the post data and check to see if there is a matching record that already exists:

```
if ($model->load(Yii::$app->request->post())) {

    $existingRating = FaqRating::find()
        ->where(['user_id' => $model->user_id])
        ->andWhere(['faq_id' => $model->faq_id])
        ->one();
}
```

If it does already exist, we assign the rating from the form to this instance of the model, which is \$existingRating, and then we run update:

```
if(isset($existingRating->id)) {

    $existingRating->faq_rating = $model->faq_rating;

    $existingRating->update();
}
```

Note that we can't just run save because it will create a new record for us. This is a byproduct of not being able to use \$model->isNewRecord in the form.

Next, because our Faq view pages require a slug to resolve, we need to look up the appropriate slug:

```
$slug = Faq::find('slug')
    ->where(['id' => $existingRating->faq_id])
    ->one();
```

After that, we set a flash message for the view:

```
Yii::$app->session->setFlash('success', 'Thank you for updating this Faq to '
    . $existingRating->faq_rating.
    ' stars. Your result is factored into the
    average.');
```

We're making this flash message dynamic so it will pass back the updated rating, which we will include in the message.

Finally, we redirect to the correct view, passing the id of the faq and the slug:

```
return $this->redirect(['faq/view',
    'id' => $existingRating->faq_id,
    'slug' => $slug->slug
]);
```

If it was not an existing record, we have a simpler path. We just save, find the slug, set the flash, and redirect:

```
} else {

    if($model->save()) {

        $slug = Faq::find('slug')
            ->where(['id' => $model->faq_id])
            ->one();

        Yii::$app->session->setFlash('success',
            'Thank you for rating this Faq ' . $model->faq_rating.
            ' stars. Your result is factored into the average.');

        return $this->redirect(['faq/view',
            'id' => $model->faq_id, 'slug' => $slug->slug
        ]);
    }
}
```

You should be able to test that at this point and it all should work.

Let's return to the view for a minute to discuss the Growl widget:

```
<?php

if (Yii::$app->getSession()->hasFlash('success')){

    echo Growl::widget([
        'type' => Growl::TYPE_SUCCESS,
        'title' => 'Thank you!',
        'icon' => 'glyphicon glyphicon-ok-sign',
        'body' => Yii::$app->session->getFlash('success'),
        'showSeparator' => true,
```

```

'delay' => 0,
'pluginOptions' => [
    'placement' => [
        'from' => 'top',
        'align' => 'right',
    ]
]
]);
}

Yii::$app->getSession()->removeFlash('success');

?>

```

We wrap it in an if statement, so it only fires if:

```
if (Yii::$app->getSession()->hasFlash('success')){
```

In the body setting, we pull in our specific flash message, which has been set in the controller:

```
'body' => Yii::$app->session->setFlash('success'),
```

Obviously you can change this to handle error messages as well. I didn't feel it was necessary because if there is an error, we throw an exception in the controller. You can check Kartik's widget documentation for more details on settings.

You'll also note that we add a separate line after the widget fires:

```
Yii::$app->getSession()->removeFlash('success');
```

This prevents the default behavior, which would be a Bootstrap alert element containing the flash message, which would fire in addition to the growl.

If you wanted to code this where you didn't want someone to be able to update their rating, you would code that logic in here:

```
if(isset($existingRating->id)){  
  
    // redirect with flash message  
  
    // sorry, you are not allowed to update your rating
```

So we have a nice frontend implementation of our FaqRatings model, but we probably want to know the ratings for individual Faqs in the backend too.

Since we've already done all the heavy lifting, this will be easy to implement. We have just 3 changes.

Faq Model

We need to add to the model a method to return the ratings of each Faq. Let's add the following:

```
public function getFaqRatings($id)  
{  
  
    $rating = new FaqRating;  
  
    return $rating->getAverageRating($id) ?  
        $rating->getAverageRating($id) : 'Not Rated' ;  
  
}
```

We already have a method for getting the average rating on the FaqRating model, so we just use that to return the average rating of our Faq. If it's null or 0, we'll set it to 'Not Rated.'

Troubleshooting tip: Please make sure you added that to the Faq model. We have a number of models with Faq in the name, so watch out for that.

Faq Index View

Next we modify the index view by adding the following into the columns array in the Gridview widget:

```
[ 'attribute'=>'Rating', 'value' => function($model){  
  
    return $model->getFaqRatings($model->id);  
  
},
```

Faq View

Finally, we add one line to the DetailView widget in view.php:

```
[ 'attribute'=>'Rating', 'format'=>'raw',  
    'value' => $model->getFaqRatings($model->id)  
  
],
```

And that's it. I don't have a solution yet for making the rating column sortable, but I'm working on it.

It's complicated because we are using the setSort method of ActiveDataProvider, and that is set up to order DB columns and our ratings column is obviously not that. I will probably have to use a fairly complicated query with either raw SQL or by creating a custom query class to get what I need. It's a subject worthy of its own chapter, but I didn't want to hold up publication for that, so I will update the book when I have that solution in place.

Ok, let's move on.

When users join a website, they typically have to agree to the terms of service. This is often represented as a checkbox, and if you don't agree, you don't get to signup.

Typically, we also see the terms of service in a scrollable box near the check box, like so:

Signup



Otherwise please fill out the following fields to signup:

Username

Email

Password

Terms Of Service

We will put all the terms of service here.

1. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis at

Agree To Terms

Signup

This is easy to set up, just 5 parts to it:

- Modify SignupForm model with the new checkbox attribute and validation
- Modify signup.php view file to contain the checkbox and render terms
- Create terms.php view file to hold the terms content
- Create overflow css style to define scroll bars
- Modify AppAsset.php to hold the new css style.

Signup Form Model

Let's look at the modified SignupForm model.

Gist:

SignupForm.php

From book:

```
<?php  
namespace frontend\models;  
  
use common\models\User;  
use yii\base\Model;  
use Yii;  
  
/**  
 * Signup form  
 */  
  
class SignupForm extends Model  
{  
    public $username;  
    public $email;  
    public $password;  
    public $agreeToTerms;  
  
    /**  
     * @inheritdoc  
     */  
    public function rules()  
    {  
        return [  
            ['username', 'filter', 'filter' => 'trim'],  
            ['username', 'required'],  
            ['username', 'unique', 'targetClass' => '\common\models\User',  
                'message' => 'This username has already been taken.'],  
            ['username', 'string', 'min' => 2, 'max' => 255],  
  
            ['email', 'filter', 'filter' => 'trim'],  
            ['email', 'required'],  
            ['email', 'email'],  
            ['email', 'unique', 'targetClass' => '\common\models\User',  
                'message' => 'This email address has already been taken.'],  
  
            ['password', 'required'],  
            ['password', 'string', 'min' => 6],  
        ];  
    }  
}
```

```

        ['agreeToTerms', 'boolean'],
        ['agreeToTerms', 'compare', 'compareValue'=>true,
         'message' =>'You must agree to our Terms of Service.'],
    ];
}

/**
 * Signs user up.
 *
 * @return User|null the saved model or null if saving fails
 */
public function signup()
{
    if ($this->validate()) {
        $user = new User();
        $user->username = $this->username;
        $user->email = $this->email;
        $user->setPassword($this->password);
        $user->generateAuthKey();
        if ($user->save()) {
            return $user;
        }
    }

    return null;
}
}

```

We added the class property:

```
public $agreeToTerms;
```

And then we set up validation for it:

```

['agreeToTerms', 'boolean'],
['agreeToTerms', 'compare', 'compareValue'=>true,
 'message' =>'You must agree to our Terms of Service.'],

```

So we're defining it as a boolean. Then we use the compare validator to make sure the value is true, which means the checkbox has been checked. We also set the error message if the compare value fails.

signup.php

Gist:

[signup.php](#)

From book:

```
<?php

use yii\helpers\Html;
use yii\bootstrap\ActiveForm;

/* @var $this yii\web\View */
/* @var $form yii\bootstrap\ActiveForm */
/* @var $model \frontend\models\SignupForm */

$this->title = 'Signup';
$this->params['breadcrumbs'][] = $this->title;
?>
<div class="site-signup">
    <h1><?= Html::encode($this->title) ?></h1>
    <br>

    <?= yii\authclient\widgets\AuthChoice::widget([
        'baseAuthUrl' => ['site/auth'],
        'popupMode' => false,
    ]) ?>

    <p>Otherwise please fill out the following fields to signup:</p>

    <div class="row">
        <div class="col-lg-5">
            <?php $form = ActiveForm::begin(['id' => 'form-signup']); ?>
            <?= $form->field($model, 'username') ?>
            <?= $form->field($model, 'email') ?>
            <?= $form->field($model, 'password')->passwordInput() ?>

            <div class="col-lg-5-offset-4" id="terms">
                <?php
                    echo \Yii::$app->view->renderFile('@app/views/pages/terms.php');
                ?>
```

```

<br>
</div>

<br>
<div class="row">
<div class="col-sm-4">
<?= $form->field($model, 'agreeToTerms')->checkbox() ?>
<div class="form-group">
<?= Html::submitButton('Signup', ['class' =>
    'btn btn-primary', 'name' => 'signup-button']) ?>
</div>
</div>
</div>
<?php ActiveForm::end(); ?>
</div>
</div>
</div>

```

After the password field, we make a div for the terms:

```

<div class="col-lg-5-offset-4" id="terms">

<?php
echo \Yii::$app->view->renderFile('@app/views/pages/terms.php');
?>

<br>
</div>

<br>

```

You can see how we are rendering the terms.php view, which we have not created yet. @app is a path alias that we need so the renderFile method knows where to look for the file.

Next we have:

```
<div class="row">
<div class="col-sm-4">

<?= $form->field($model, 'agreeToTerms')->checkbox() ?>
    <div class="form-group">
<?= Html::submitButton('Signup', ['class' => 'btn btn-primary',
    'name' => 'signup-button']) ?>

</div>
</div>
</div>
```

We are getting ‘Agree To Terms’ as a default label generated by the form field. If you want to change it, use the label method. For example:

```
<?= $form->field($model, 'agreeToTerms')
    ->label('I Agree To Terms')->checkbox() ?>
```

Also note:

```
<div class="row">
<div class="col-sm-4">
```

We need that to format the css. The scroll bars will not be added, however, until we define a style for them. We will do that after the next step.

terms.php

Let’s create terms.php in the frontend/views/pages folder. The reason we are putting it here is that so later on we can add it in other places on the application, perhaps a footer, to make it more accessible.

Gist:

[terms.php](#)

From book:

Terms Of Service

We will put all the terms of service here.

<p>1. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis at condimentum tortor. Nulla vestibulum ultricies urna. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce consequat lorem nec enim vehicula vulputate. Ut lacus quam, semper sit amet urna ac, pharetra porta enim. Cras et ex convallis, interdum nunc volutpat, consequat massa. Phasellus aliquet euismod nulla vel euismod. Integer dolor diam, fringilla quis elementum vehicula, egestas sit amet tellus. Aliquam convallis viverra auctor.</p>

<p>2. Nunc eu neque vitae augue pharetra posuere. Donec et ligula in mauris mollis sodales. Maecenas neque diam, dictum consectetur nunc vel, euismod blandit mauris. Praesent ornare, mi nec lobortis elementum, felis dolor elementum metus, nec placerat quam velit ut nisi. Donecfringilla iaculis ligula, sit amet interdum augue ultricies ultricies.

Aenean bibendum ante vitae malesuada efficitur. Duis tincidunt ante quam, non lobortis metus varius at.</p>

<p>3. Integer quis rhoncus purus, quis imperdiet enim. Vestibulum venenatis, erat at consequat dapibus, sem lorem venenatis dui, eget tempus diam erat non urna. Nullam at magna nec nulla sagittis luctus eget sit amet odio. Vivamus vel condimentum quam, eu fringilla nibh. Aliquam venenatis, elit id vulputate consectetur, elit erat aliquam lectus, id finibus augue ipsum eget enim. Nullam efficitur tincidunt condimentum. In luctus, nibh sit amet imperdiet pretium, ligula magna malesuada velit, vitae pharetra quam ex sit amet ligula. Cras malesuada, justo vel dictum dignissim, magna ligula porttitor nisi, in dapibus mi erat eget elit. Proin tristique interdum enim, id \ luctus dolor mollis in.</p>

<p>4. Praesent tempus mauris convallis, ultricies lorem elementum, pharetra nulla. Pellentesque in felis sed magna sollicitudin fermentum sit amet in lectus. Etiam pharetra dolor id lectus faucibus tristique. Vivamus

malesuada eu risus vitae tristique. Mauris at dolor blandit, ullamcorper ante eu', mollis elit. Fusce ultrices ligula ut tellus semper, ut ullamcorper sem tristique.

Mauris blandit, urna at pulvinar efficitur, elit eros hendrerit urna, nec fringilla

leo metus sit amet odio. Donec eu ultrices nisl. Quisque eu enim porta, fringilla metus non, bibendum tortor. Cras eu egestas enim. Aenean interdum, eros sed pretium rutrum, nisi augue hendrerit enim, dictum feugiat est augue vitae est. Donec vitae sagittis eros. Duis congue, eros vel pellentesque molestie, justo turpis facilisis felis, mollis semper velit sem a nunc. Integer eleifend, tellus sit amet auctor bibendum, quam leo maximus est, eget feugiat velit nulla eget mauris.</p>

<p>5. Fusce vel dolor eget ex varius porta quis ac libero. Proin eget metus egestas, pretium mi non, dictum nisi. Cras fringilla varius massa vitae convallis. Nunc ut blandit odio. Sed vehicula felis in neque auctor hendrerit. Nunc a magna eget felis interdum auctor. Aliquam accumsan metus justo, eget iaculis diam pharetra nec. Donec mauris lacus, lacinia non venenatis in, interdum at purus. Donec ac mi id tortor mattis convallis id sed tellus. Nullam volutpat justo pretium odio eleifend, ac vestibulum lectus tempus. Quisque ut sem viverra lorem facilisis facilisis. Aliquam erat volutpat.

</p>

Obviously not much to that. I will mention that I used the lorem ipsum generator at:

[Lorem Ipsum Generator](#)

termsoverflow.css

To get the scroll bars to work correctly, we need to create termsoverflow.css and add it to the frontend/web/css folder.

This class consists of the following:

```
#terms {
    height: 150px;
    overflow: scroll;
}
```

Frontend AppAsset.php

Now that we have our style class to show the scroll bars, we need to tell our app to use it. We need to modify the our frontend AppAsset.php

Gist:

[Frontend AppAsset.php](#)

From book:

```
<?php
/**
 * @link http://www.yiiframework.com/
 * @copyright Copyright (c) 2008 Yii Software LLC
 * @license http://www.yiiframework.com/license/
 */

namespace frontend\assets;

use yii\web\AssetBundle;

/**
 * @author Qiang Xue <qiang.xue@gmail.com>
 * @since 2.0
 */
class AppAsset extends AssetBundle
{
    public $basePath = '@webroot';
    public $baseUrl = '@web';
    public $css = [
        'css/site.css',
        'css/termsoverflow.css'
    ];
    public $js = [
    ];
}
```

```
public $depends = [
    'yii\web\YiiAsset',
    'yii\bootstrap\BootstrapAsset',
];
}
```

That was just a simple addition, adding our new style to the public \$css array.

That should be everything you need for this implementation.

Improving The Carousel

In the last chapter, we built a carousel for marketing images that we could control from the our backend admin UI. It works well, except that the photos don't scale properly when they are scaled down to a mobile-sized browser.

This is happening for two reasons.

1. We are specifying height and width of the image, which defeats the mobile response.
2. The shape of the photo we have been working with is rectangular, while the mobile shape is square.

If we remove the height and width attributes, the image scales, but we still have the problem of the shapes not matching, and therefore scaling is imprecise.

You could solve it by making sure you have a square image in the carousel to start with, but this is a serious limitation on the space. Not good.

I imagined showing the current carousel to a client and them complaining about it not scaling the way they want it to. That makes all the hard work we've done on it previously worthless, and we can't settle for that.

So now we will be working with a mobile image that the carousel uses when we detect the device is using a smaller browser window. This will hand complete control of the image over to the client or admin.

None of the steps are too difficult to make this happen, but we impact our existing code and we have to do some revising:

- Modify marketing_image Table with a new field for mobile image path.
- Modify MarketingImage model to account for the new property, rules and scenario.
- Modify various views including the form to allow the upload of the additional image.
- Modify MarketingImageController create, update, and delete actions for new image type.
- Modify defaults in CaouselWidget.php to allow for no width or height to be set
- Modify carousel.php to show the appropriate image based on browser size via jquery.
- Modify CarouselSettings model to make height and width of the image not required.

Modify marketing_imageTable

Ok, let's jump in. Here is the SQL to alter the marketing_image table:

```
ALTER TABLE `yii2build`.`marketing_image`
ADD COLUMN `marketing_mobile_path` VARCHAR(45)
CHARACTER SET 'utf8' COLLATE 'utf8_unicode_ci'
NOT NULL AFTER `marketing_thumb_path`;
```

Modify MarketingImage Model

Now let's make the modifications on the model. We'll start by adding a public property:

```
public $mobileFile;
```

We need that to be able to upload the mobile image.

Next we'll add rules for 'marketing_mobile_path.' I'll give you the entire method for reference.

Gist:

[MarketingImage Rules](#)

From book:

```
public function rules()
{
    return [
        [['marketing_image_path', 'marketing_image_name',
            'marketing_thumb_path', 'marketing_mobile_path',
            'marketing_image_weight'], 'required'],
        ['marketing_image_weight', 'default', 'value' => 100 ],
        ['marketing_image_is_featured', 'default', 'value' => 0 ],
        ['marketing_image_is_active', 'default', 'value' => 0 ],
        ['file', 'required', 'message' =>
            '{attribute} can\'t be blank', 'on'=>'create'],
        [['marketing_image_name', 'marketing_thumb_path',
            'marketing_mobile_path','marketing_image_path'], 'trim'],
        [['marketing_image_is_featured', 'marketing_image_is_active',
            'marketing_image_weight', 'status_id'], 'integer'],
        [['marketing_image_is_featured'],'in',
            'range'=>array_keys($this->getMarketingImageIsFeaturedList())],
        [['marketing_image_is_active'],'in',
```

```

    'range'=>array_keys($this->getMarketingImageIsActiveList())],
    [['file'], 'file', 'extensions' => ['png', 'jpg',
        'gif', 'jpeg'], 'maxSize' => 1024*1024],
    [['mobileFile'], 'file', 'extensions' => ['png', 'jpg',
        'gif', 'jpeg'], 'maxSize' => 250*250],
    [['marketing_image_path', 'marketing_image_name'],
        'string', 'max' => 45],
    [['marketing_image_caption', 'marketing_image_caption_title'],
        'string', 'max' => 100],

];
}

```

You should note that I set the size limit on the mobileFile to 250 x250. You can change this if you prefer a different size in the mobile image.

Next, add into scenarios ‘marketing_mobile_path’.

Gist:

[scenarios](#)

From book:

```

public function scenarios()
{
    $scenarios = parent::scenarios();
    $scenarios['create'] = ['file', 'marketing_image_path',
        'marketing_image_name', 'marketing_thumb_path',
        'marketing_mobile_path', 'marketing_image_is_featured',
        'marketing_image_is_active', 'marketing_image_caption',
        'marketing_image_caption_title', 'marketing_image_weight'];

    return $scenarios;
}

```

We also need to add ‘marketing_mobile_path’, to attribute labels.

```
'mobileFile' => 'Mobile Image',
```

For reference, I’m going to give you a Gist for the complete file:

Gist:

[MarketingImage.php](#)

MarketingImage Views

Ok, moving on to the views. Let's start with _form.php. We just need to add a single line:

```
<?= $form->field($model, 'mobileFile')->label('Mobile Image')->fileInput(); ?>
```

Marketing Image view.php

Our view.php file had a few changes:

Gist:

[view.php](#)

From book:

```
<?php

use yii\helpers\Html;
use yii\widgets\DetailView;

/* @var $this yii\web\View */
/* @var $model backend\models\MarketingImage */

$this->title = $model->id;
$this->params['breadcrumbs'][] = ['label' =>
    'Marketing Images', 'url' => ['index']]
];
$this->params['breadcrumbs'][] = $this->title;
?>
<div class="marketing-image-view">

<p>
    <?= Html::a('Update', ['update', 'id' => $model->id], [
        'class' => 'btn btn-primary']) ?>
    <?= Html::a('Delete', ['delete', 'id' => $model->id], [
        'class' => 'btn btn-danger',
        'data' => [
            'confirm' => 'Are you sure you want to delete this item?',
            'method' => 'post',
        ],
    ]) ?>
</p>
```

```
<h1><?= Html::encode($model->marketing_image_name) ?></h1>
<br>
<div>

<h3>Primary Image:</h3>
<?php

echo Html::img('/'. $model->marketing_image_path .
    '?'. 'time=' . time() , ['width' => '600px',
    'alt' => $model->marketing_image_name]);


?>

</div>
<br>
<div>

<h3>Mobile Image:</h3>
<?php

echo Html::img('/'. $model->marketing_mobile_path .
    '?'. 'time=' . time(), [
    'alt' => $model->marketing_image_name]);


?>
</div>
<br>
<div>

<h3>Thumbnail Image:</h3>
<?php

echo Html::img('/'. $model->marketing_thumb_path .
    '?'. 'time=' . time(), [
    'alt' => $model->marketing_image_name]);


?>

</div>
<br>
```

```

<?= DetailView::widget([
    'model' => $model,
    'attributes' => [
        'id',
        'marketing_image_caption_title',
        'marketing_image_caption',
        'marketing_image_path',
        'marketing_thumb_path',
        'marketing_mobile_path',
        'marketing_image_weight',
        ['attribute' => 'marketing_image_is_featured',
            'format' => 'boolean'],
        ['attribute' => 'marketing_image_is_active',
            'format' => 'boolean'],
        'status.status_name',
        'created_at',
        'updated_at',
    ],
]) ?>

</div>

```

In addition to adding ‘marketing_mobile_path’ to the attributes in the DetailView widget, we also added h3 titles to the images, like so:

```
<h3>Mobile Image:</h3>
```

And of course we added the image itself, using our get variable to prevent caching:

```

<?php

echo Html::img('/'. $model->marketing_mobile_path .
    '?'. 'time=' . time(), [
    'alt' => $model->marketing_image_name]);

?>

```

Update View

We also add the mobile image and titles to update.php.

Gist:

Update View

From book:

```
<?php

use yii\helpers\Html;

/* @var $this yii\web\View */
/* @var $model backend\models\MarketingImage */

$this->title = 'Update Marketing Image: ' . ' ' . $model->id;
$this->params['breadcrumbs'][] = ['label' =>
    'Marketing Images', 'url' => ['index']];
$this->params['breadcrumbs'][] = ['label' => $model->id,
    'url' => ['view', 'id' => $model->id]];
$this->params['breadcrumbs'][] = 'Update';
?>
<div class="marketing-image-update">

<h1><?= Html::encode($this->title) ?></h1>

<br>
<div>
<h3>Primary Image:</h3>
<?php
echo Html::img('/'. $model->marketing_image_path,
    ['width' => '600px']);

?>
</div>
<br>
<div>
<h3>Mobile Image:</h3>
<?php

echo Html::img('/'. $model->marketing_mobile_path .
    '?' . 'time=' . time());

?>
</div>
```

```

<br>
<div>
<h3>Thumbnail Image:</h3>
<?php

echo Html::img('/'. $model->marketing_thumb_path .
'?'. 'time=' . time());

?>

</div>
<br>

<?= $this->render('_form', [
    'model' => $model,
]) ?>

</div>

```

MarketingImage Controller

On MarketingImageController, we modified create, update and delete to account for our new \$mobileFile. Let's look at these one at a time.

Create Action

Gist:

[create action](#)

From book:

```

public function actionCreate()
{
    $model = new MarketingImage();
    $model->scenario = 'create';

    if ($model->load(Yii::$app->request->post())) {

        $imageName = $model->marketing_image_name;

        $model->file = UploadedFile::getInstance($model, 'file');
    }
}

```

```

$model->mobileFile = UploadedFile::getInstance($model, 'mobileFile');

$fileName = 'uploads/' . $imageName . '.' . $model->file->extension;
$fileName = preg_replace('/\s+/', '', $fileName);

$thumbName = 'uploads/' . 'thumbnail/' . $imageName .
'-thumb.' . $model->file->extension;
$thumbName = preg_replace('/\s+/', '', $thumbName);

$mobileName = 'uploads/' . 'mobile/' . $imageName . '-mobile.' .
$model->mobileFile->extension;
$mobileName = preg_replace('/\s+/', '', $mobileName);

$model->marketing_image_path = $fileName;
$model->marketing_thumb_path = $thumbName;
$model->marketing_mobile_path = $mobileName;
$model->save();

$model->file->saveAs($fileName);
$model->mobileFile->saveAs($mobileName);

Image::thumbnail( $fileName , 60, 60)
->save($thumbName, ['quality' => 50]);

return $this->redirect(['view', 'id' => $model->id, 'model' => $model,]);
}

} else {
    return $this->render('create', [
        'model' => $model,
    ]);
}
}

```

You can see that we have used getInstance set the uploaded file to \$model->mobileFile.

We also have a block to set the name and remove spaces:

```

$mobileName = 'uploads/' . 'mobile/' . $imageName . '-mobile.' .
$model->mobileFile->extension;
$mobileName = preg_replace('/\s+/', '', $mobileName);

```

We make sure we have the name set correctly:

```
$model->marketing_mobile_path = $mobileName;
```

After saving the model, we use saveAs for the file:

```
$model->mobileFile->saveAs($mobileName);
```

Update Action

Gist:

[update Action](#)

From book:

```
public function actionUpdate($id)
{
    $model = $this->findModel($id);

    if ($model->load(Yii::$app->request->post())){
        $imageName = $model->marketing_image_name;

        $oldImage = MarketingImage::find('marketing_image_name')
            ->where(['id' => $id])
            ->one();

        if ($oldImage->marketing_image_name != $imageName){
            throw new ForbiddenHttpException
                ('You cannot change the name, you must delete instead.');
        }

        if ( $model->file = UploadedFile::getInstance($model, 'file')){
            $thumbName =  'uploads/' . 'thumbnail/' . $imageName .
                '-thumb.' . $model->file->extension;
        }
    }

    if ($model->mobileFile = UploadedFile::getInstance($model, 'mobileFile')){
```

```
$mobileName = 'uploads/mobile/' . $imageName .
'-mobile.' . $model->mobileFile->extension;

}

$model->save();

if ($model->file) {

$fileName = 'uploads/' . $imageName . '.' . $model->file->extension;

$model->file->saveAs($fileName);

Image::thumbnail( $fileName , 60, 60)
->save($thumbName, ['quality' => 50]);

}

if ($model->mobileFile) {

$model->mobileFile->saveAs($mobileName);

}

return $this->redirect(['view', 'id' => $model->id]);

} else {

return $this->render('update', [
'model' => $model,
]);
}

}
```

Just two spots to note. The first is before we save the model:

```
if ($model->mobileFile = UploadedFile::getInstance($model, 'mobileFile')){

    $mobileName = 'uploads/mobile/' . $imageName .
    '-mobile.' . $model->mobileFile->extension;

}
```

This is just like what we do for the thumbnail image, so we have already covered this in the previous chapter.

Next we add:

```
if ($model->mobileFile) {

    $model->mobileFile->saveAs($mobileName);

}
```

Note that both of these statements are wrapped in if statements because updating the mobile image is not required, whereas when we create the record, the mobile image is required.

Delete Action

Gist:

[delete Action](#)

From book:

```
public function actionDelete($id)
{
    $model = $this->findModel($id);

    try {

        unlink($model->marketing_image_path);

        unlink($model->marketing_thumb_path);

        unlink($model->marketing_mobile_path);

        $model->delete();
    }
}
```

```
    return $this->redirect(['index']);  
}  
  
catch(\Exception $e) {  
    throw new NotFoundHttpException($e->getMessage());  
}  
}
```

Ok, that's simple enough. We just added the extra unlink for the mobile image.

Entire File

I'm supplying a gist of the entire file for reference, should you need it:

Gist:

[MarketingImage Controller](#)

CarouselSettings Model Rules

So the first thing we can do is remove image_height and image_width from the first rule because they are no longer required. While we're here, I did decide that the carousel_name had to be unique. This will prevent an admin from accidentally creating two records of settings for one carousel. So here we have the entire rules method:

Gist:

[rules](#)

From book:

```

public function rules()
{
    return [
        [['carousel_name', 'caption_font_size', 'status_id'], 'required'],
        [['carousel_name'], 'unique'],
        [['carousel_autoplay', 'show_indicators', 'show_captions',
            'status_id', 'show_controls'], 'integer'],
        [['carousel_autoplay'], 'in',
            'range'=>array_keys($this->getCarouselAutoplayList())],
        [['show_indicators'], 'in',
            'range'=>array_keys($this->getShowIndicatorsList())],
        [['show_captions'], 'in',
            'range'=>array_keys($this->getShowCaptionsList())],
        [['show_caption_background'], 'in',
            'range'=>array_keys($this->getShowCaptionBackgroundList())],
        [['show_caption_title'], 'in',
            'range'=>array_keys($this->getShowCaptionTitleList())],
        [['show_controls'], 'in',
            'range'=>array_keys($this->getShowControlsList())],
        [['status_id'], 'in', 'range'=>array_keys($this->getStatusList())],
        [['created_at', 'updated_at'], 'safe'],
        [['carousel_name', 'image_height', 'image_width'],
            'string', 'max' => 45]
    ];
}

```

CarouselWidget

Moving on to the CarouselWidget class, we are going to change the setDefaults method to set height and width to null if no other value is provided. This default will be the typical implementation.

I don't need to give you a Gist, just change the value to null as below:

```

if (!isset($this->settings['height'])){
    $this->settings['height'] = null;
}

if (!isset($this->settings['width'])){
    $this->settings['width'] = null;
}

```

validateSize Method

Next we wrap everything in the validateSize method in an if statement, so that we are only validating size if height and width are not empty:

```
if (!empty($this->settings['width']) && !empty($this->settings['height'])){

}
```

Entire CarouselWidget File

For reference, I am providing the entire CarouselWidget file below:

Gist:

[CarouselWidget.php](#)

carousel.php

The last step in this is to modify `carousel.php`. Ultimately, we are going to do something similar to what we did with the faq rating, where we show/hide based, in this case, on the size of the browser.

Unfortunately, that means we have to add a second carousel to the view, which takes a giant plate of spaghetti and doubles it. There's not much I can do about that. I tried simply replacing part of the carousel, but that didn't work.

What I did do, that actually worked, is wrap each carousel in a div, one named `big` and the other named `small`. So hopefully that will keep everything clear.

The other big difference is the path to the image in the small carousel. All that is followed by the javascript at the bottom of the file. Here is the entire file:

Gist:

[carousel.php](#)

From book:

```
<?php

use yii\helpers\Html;

?>

<div id="big">
<div id="carouselMain" class="carousel slide"

<?php

if($settings['autoplay'] == false ){

    echo 'data-interval="false"';
}

?>

data-ride="carousel">

<!-- Indicators --> <?php

if ($settings['show_indicators']){
    echo '<ol class="carousel-indicators">
<li data-target="#carouselMain" data-slide-to="0"
    class="active"></li>';

    foreach (range(1, $count) as $number) {

        echo '<li data-target="#carouselMain"
            data-slide-to="'.$number.'"></li>';

    }

    echo '</ol> ';
}

?>

<!-- Wrapper for slides -->
```

```
<div class="carousel-inner" role="listbox">

<!-- dynamic slide data -->

<?php

$width = $settings['width'];
$height = $settings['height'];
//active item first

echo '<div class="item active">
<center>'.
Html::img(Yii::$app->urlManagerBackend->baseUrl. '/' .
$activeImage['marketing_image_path'], ['width' => $width,
'height' => $height ])
.</center>';

if($settings['show_captions']){

echo '<div class="carousel-caption">';

if ($settings['show_caption_title']){

echo '<div><h1>' . $activeImage
['marketing_image_caption_title'] . '</h1></div>';

}

echo $activeImage['marketing_image_caption'] . '</div>';
}

echo '</div>';

//all other images

foreach ($images as $image){

echo '<div class="item">
<center>'.
Html::img(Yii::$app->urlManagerBackend->baseUrl . '/' .
$image['marketing_image_path'], ['width' => $width,
'height' => $height ])
```

```
.  '</center>';

if($settings['show_captions']){
    echo '<div class="carousel-caption">';
    if ($settings['show_caption_title']){
        echo  '<div><h1>' .
        $image['marketing_image_caption_title'] .
        '</h1></div>';
    }
    echo $image['marketing_image_caption'] . ' </div>';
}
echo '</div>';

}

?>

<!-- end dynamic slide data -->

</div>

<!-- Controls -->
<?php
if ($settings['show_controls']){
    echo '<a class="left carousel-control" href="#carouselMain" role="button" data-slide="prev">
        <span class="glyphicon glyphicon-chevron-left" aria-hidden="true"></span>
        <span class="sr-only">Previous</span>
    </a>
    <a class="right carousel-control" href="#carouselMain" role="button" data-slide="next">
        <span class="glyphicon glyphicon-chevron-right" aria-hidden="true"></span>
        <span class="sr-only">Next</span>
    </a>
}
?>
```

```
<span class="glyphicon glyphicon-chevron-right"
      aria-hidden="true"></span>
<span class="sr-only">Next</span>
</a>';

}

?>
</div>

</div>

<div id="small">
<div id="carouselSmall" class="carousel slide"

<?php

if($settings['autoplay'] == false ){

    echo 'data-interval="false"';
}

?>

data-ride="carousel">

<!-- Indicators --> <?php

if ($settings['show_indicators']){
    echo '<ol class="carousel-indicators">
<li data-target="#carouselSmall" data-slide-to="0"
    class="active"></li>';

    foreach (range(1, $count) as $number) {

        echo '<li data-target="#carouselSmall"
            data-slide-to="'. $number .'"></li>';
    }
}
```

```
echo '</ol> ';
```

```
}
```

```
?>
```

```
<!-- Wrapper for slides -->
<div class="carousel-inner" role="listbox">
```

```
<!-- dynamic slide data -->
```

```
<?php
```

```
$width = $settings['width'];
$height = $settings['height'];
```

```
//active item first
```

```
// use the path for mobile image
```

```
echo '<div class="item active">
<center>'.
Html::img(Yii::$app->urlManagerBackend->baseUrl. '/' .
$activeImage['marketing_mobile_path']. '?' . 'time=' . time())
.</center>';
```

```
if($settings['show_captions']){
    echo '<div class="carousel-caption">';
    if ($settings['show_caption_title']){
        echo '<div><h1>' . $activeImage
        ['marketing_image_caption_title'] . '</h1></div>';
    }
    echo $activeImage['marketing_image_caption'] . '</div>';
}
```

```
echo '</div>';
```

```
//all other images

// use the path for mobile image

foreach ($images as $image){

    echo '<div class="item">
        <center>' .
        Html::img(Yii::$app->urlManagerBackend->baseUrl . '/' .
        $image['marketing_mobile_path'] . '?' . 'time=' . time())
        . '</center>';

    if($settings['show_captions']){
        echo '<div class="carousel-caption">';

        if ($settings['show_caption_title']){
            echo    '<div><h1>' .
            $image['marketing_image_caption_title']. '</h1></div>';

        }

        echo $image['marketing_image_caption']. '</div>';

    }

    echo '</div>';

}

?>

<!-- end dynamic slide data -->

</div>

<!-- Controls -->
<?php
if ($settings['show_controls']){

```

```
echo '<a class="left carousel-control" href="#carouselSmall" role="button"
      data-slide="prev">
  <span class="glyphicon glyphicon-chevron-left" aria-hidden="true"></span>
  <span class="sr-only">Previous</span>
</a>
<a class="right carousel-control" href="#carouselSmall" role="button"
      data-slide="next">
  <span class="glyphicon glyphicon-chevron-right" aria-hidden="true"></span>
  <span class="sr-only">Next</span>
</a>';

}

?>
</div>

</div>

<?Php
$script = <<< JS
if ($(window).width() <= 800){
    $("#big").hide();
    $("#small").show();
    $('#carouselSmall').carousel({
        interval: 1000
    });
} else{
    if ($(window).width() >800){
        $("#small").hide();
        $("#big").show();
        $('#carouselMain').carousel();
        interval: 1000
    }
}
$(window).resize(function(){
    if ($(window).width() <= 800){
        $("#big").hide();
        $("#small").show();
    } else{
        $("#small").hide();
        $("#big").show();
    }
});
```

```

$( "#small" ).show();
$( '#carouselSmall' ).carousel({
    interval: 1000
});
} );
} else{

    if ($(window).width() >800){
        $("#small").hide();
        $("#big").show();
        $('#carouselMain').carousel();
        interval: 1000
    }

}
} );
JS;

$this->registerJs($script);

?>

```

So let's hit the highlights.

- We have two carousels, each in a separate div, one named big, the other small.
- The carousels also have div ids as carouselMain and carouselSmall
- The path to the image in carouselSmall is obviously different than that of carouselMain.
- The javascript at the bottom of the file determines which carousel will be visible, depending on browser size.

Summary

In this chapter, we got to build a ratings system for our Faqs, using Kartik's widget extension. That worked out really well for us because now we know how to create a rating system for any type of model that users might be willing to rate.

We also implemented the Growl widget, which gave us a nice UI experience, whenever there is a save or update to the ratings. You can use the Growl widget often, whenever you are setting flash messages, and it would make the application act in a very consistent way.

We also had our first use of a checkbox, which we implemented to confirm the user's agreement to terms of service. It was simple stuff, but we also added to our frontend AppAssets.php file to pull in the css, so we are getting more familiar with Yii 2's asset publishing.

Finally, we went back into the carousel to give ourselves full control over that marketing space, so we can nuance the image in a mobile browser by using a separate image for that purpose. Even though we expanded the functionality of the carousel, we still have 100% management of the carousel through our backend UI, creating a user-friendly admin environment that will be appreciated by those responsible for the marketing in the carousel. And that's it for this chapter.

Thanks once again to everyone for the positive comments and reviews, they keep me motivated to keep going with more bonus material. Please help spread the word about the book if you can. It will be greatly appreciated.

In the meantime, I will continue to work on bonus material and I will keep working on the template. Thanks again for supporting the book. See you soon.

Chapter 17: Bonus Material Returning Calculated Values in Gridview

Sorting A Calculated Value In Gridview

Welcome back to another bonus chapter. When we built our Faq ratings system, we noticed that the rating column in the gridview widget wasn't sortable.

The reason why it wasn't sortable in our first implementation is that we need to return the average rating of the faq, and that is a calculated value, not a strait DB column value. It was easy enough for us to build a method to return the average, but we didn't know how to sort on that method.

This might seem like a trivial detail, and maybe we could just forget about it, but of course clients and users love column sorts and with good reason. They allow us to organize our view and to see the most important data with the click of a button. So not having the sort capability is not really an option.

There's always more than one way to do things, so as I thought about the solution, I thought we might have to switch to SQL data provider because I knew the query was more complex than what we are used to dealing with.

For example, if we go straight to our SQL tab in PhpMyadmin and put in the following SQL:

```
select *
from faq as a
inner join
(SELECT faq_id, AVG(faq_rating) as 'average_rating'
FROM faq_rating
GROUP BY faq_id) as b
ON a.id=b.faq_id;
```

You will get something like:

The screenshot shows a MySQL query results grid. The SQL query is:

```

SELECT *
FROM faq AS a
INNER JOIN (
    SELECT faq_id, AVG(faq_rating) AS 'average_rating'
    FROM faq_rating
    GROUP BY faq_id
)

```

The results grid has columns: id, faq_question, slug, faq_answer, faq_category_id, faq_is_featured, faq_weight, created_by, updated_by, created_at, updated_at, faq_id, and average_rating. The data includes rows for various FAQ entries, with the last row being the calculated average rating.

<code>id</code>	<code>faq_question</code>	<code>slug</code>	<code>faq_answer</code>	<code>faq_category_id</code>	<code>faq_is_featured</code>	<code>faq_weight</code>	<code>created_by</code>	<code>updated_by</code>	<code>created_at</code>	<code>updated_at</code>	<code>faq_id</code>	<code>average_rating</code>
11	Should I use a framework?	should-i-use-a-framework	Probably, it's a good idea	1	0	3	1	1	2015-02-27 18:53:40	2015-03-01 11:43:38	11	4.5
12	Am I going to finish?	am-i-going-to-finish	Ysel	1	1	85	1	1	2015-02-27 18:54:21	2015-02-27 18:54:21	12	1
13	Am I still having fun?	am-i-still-having-fun	Probably	2	1	95	1	1	2015-02-27 18:55:03	2015-02-27 18:55:03	13	3
15	What time is it?	what-time-is-it	Now	1	0	100	1	1	2015-02-28 18:33:12	2015-02-28 18:33:12	15	3
16	is it similar to timestamp?	is-it-similar-to-timestamp	let's find out	1	1	100	1	1	2015-02-28 18:35:39	2015-02-28 18:35:39	16	4

Extended Table

It's hard to see everything because it's compressed down, but we get the extra row we need for average rating. I based this query on a tutorial:

Query Tutorial

You can see this isn't the friendliest format to work with. For one thing, the above query is not returning results when there is no rating, so the query is not even correct.

The simple solution to that problem is to make it a LEFT join, which will return all the results. Ok, so we overcame that.

When I first started using Yii 2, I tended to rely more on plain SQL than ActiveRecord, for the reason illustrated above. If you need to figure out a query, you can simply search for an answer, and it can be faster to develop that way. But now that I have more experience with ActiveRecord, I really love it. It's so much more intuitive and easy to use. It's worth taking the extra time to figure out how to work with it.

There is something else to consider, however, and that is the overhead you will incur by using any framework's ORM. For large databases and complex queries, it's generally not considered practical or scalable. But with PHP 7 coming, the framework overhead might be reduced to a very workable level for complex queries. And that means we can use ActiveRecord without worrying that we are harming our ability to scale.

Since PHP 7 is due in October of this year, I'm going on the assumption that ActiveRecord will be a viable solution for complex queries.

Also, as we will see as we develop this, we do not have to sacrifice MySQL's built-in functions like AVG and COUNT when we use ActiveRecord, so we still get to benefit from MySQL's performance capabilities for those kinds of queries.

So now we have to figure out how to translate our SQL query to ActiveRecord, if we want to use it with ActiveDataProvider and continue to use the existing solution we have for our FaqSearch search method. This is what we want to do.

The problem is that we have raw SQL, but we don't know how to translate it to ActiveRecord, or even if it can be translated to ActiveRecord.

To figure this out, we should reference two sections of the Yii 2 docs:

[Query Builder](#)

And

[ActiveRecord](#)

There's quite a lot of information there on all the methods available to us. That said, I was still a little unsure. So I went to the forum for help:

[Help with SQL to ActiveRecord](#)

Very quickly I received a couple of responses, and one of those was from Kartik. It turns out, he has written a web tip on his site for this scenario:

[Kartik's Web Tip](#)

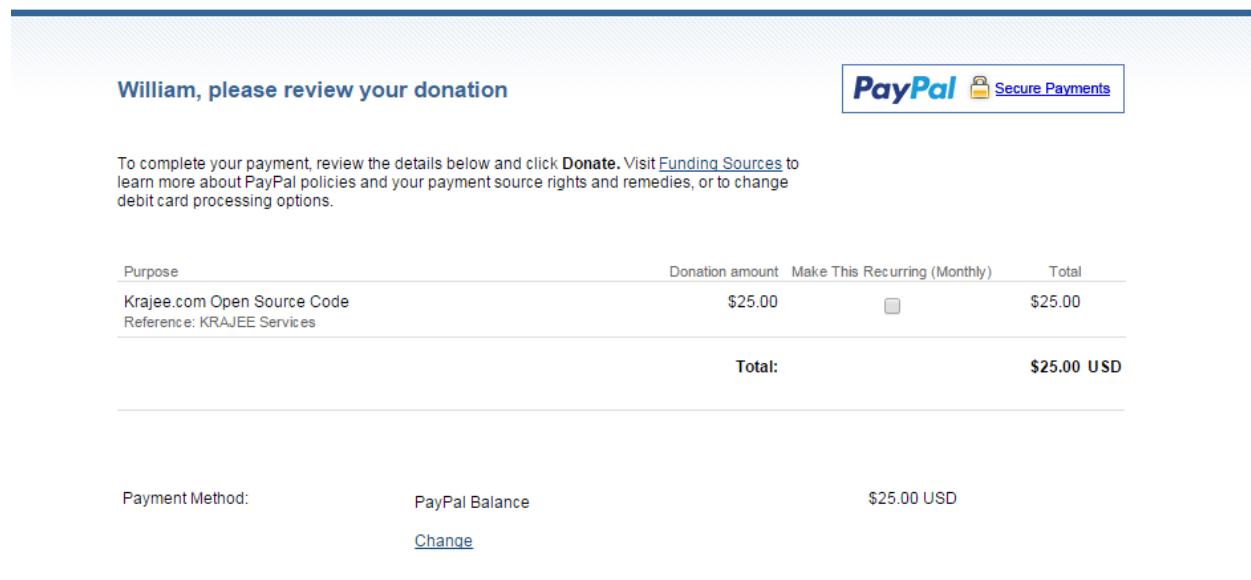
So, in addition to all the amazing widgets he has made for us, he also has anticipated that we would need some help with sorting a calculated field in Gridview. It's just incredibly helpful.

Donate To Kartik

Please donate to Kartik if you can, he is contributing a lot to the Yii 2 community. His donation button is at:

[Krajee.com](#)

Here is a copy of my donation:



The screenshot shows a PayPal donation page. At the top, it says "William, please review your donation". On the right, there is a "PayPal" logo with "Secure Payments" next to it. Below this, a message reads: "To complete your payment, review the details below and click **Donate**. Visit [Funding Sources](#) to learn more about PayPal policies and your payment source rights and remedies, or to change debit card processing options." A table then displays the donation details:

Purpose	Donation amount	Make This Recurring (Monthly)	Total
Krajee.com Open Source Code Reference: KRAJEE Services	\$25.00	<input type="checkbox"/>	\$25.00
	Total:		\$25.00 USD

At the bottom, it says "Payment Method: PayPal Balance" and "Change".

Donation

It's a great way to say thanks for all the hard work he puts into his extensions and tutorials, which he actively maintains.

I've described my research into this chapter fully, so you can understand how to approach this if you find yourself needing help with ActiveRecord or anything else.

You should always research thoroughly before going to the forum because there's no sense in asking a question that's already been answered. I made the mistake of focusing on the query, not thinking the full solution was out there, so I never saw Kartik's web tip. Next time I will try searching on the full solution before going to the forum.

Anyway, let's jump into this solution.

Average Rating For Gridview

The first step is to create a method on the base model, in this case, the Faq model:

Gist:

[getAverageRating](#)

From book:

```
public function getAverageRating()
{
    return $this
        ->hasMany(FaqRating::className(), ['faq_id'=>'id'])
        ->average('faq_rating');
}
```

So this is just a relationship, with the average method added on. Simple, but powerful.

I found the right syntax for average by referencing the query builder section of the docs. We'll see later that the average method inside of MySQL is AVG, so we have to be sure to use the right syntax in the right place.

During my research for this chapter, I also explored the possibility of creating a scope instead of the above method. However, scopes are not directly supported by Yii 2. Instead, you can create a custom query class that overrides your find method.

After looking at the two solutions, I felt the one offered by Kartik's tutorial was simpler. It is also listed in the Yii 2 guide as an alternative to creating a scope. So, all other things being equal, I chose the simplest implementation.

If I find a good use case for the custom query class on the template, I will include it in a future chapter.

Ok, back to the base model. We also add an attributeLabel:

```
'averageRating' => Yii::t('app', 'Rating'),
```

This isn't anything we haven't seen before. In case you need a reminder, we use the magic syntax, so there is no get and the average is lowercase since it's the first word.

Now we'll move on to the search model FaqSearch. Let's start with a use statement:

```
use backend\models\FaqRating;
```

and add a public property:

```
public $averageRating;
```

And we add the safe attribute in the rules:

```
public function rules()
{
    return [
        [[ 'id', 'faq_category_id', 'faq_weight',
            'faq_is_featured', 'created_by', 'updated_by'],
         'integer'],
        [[ 'faq_question', 'faq_answer', 'created_at',
            'updated_at', 'faqCategoryName', 'faqCategoryList',
            'faqIsFeaturedName', 'createdByUsername', 'updatedByUsername',
            'faq_category', 'faq_weight', 'averageRating'], 'safe'],
    ];
}
```

All we did was add the single attribute 'averageRating' to the safe rule.

Ok, moving on to our search method, we start by adding to our query definition. Before we had this:

```
$query = Faq::find();
```

Now we need this:

Gist:

```
query
```

From book:

```
$query = Faq::find();

$subQuery = FaqRating::find()
->select('faq_id, AVG(`faq_rating`) as average_rating')
->groupBy('faq_id');

$query->leftJoin([
    'faqAverage'=>$subQuery
], 'faqAverage.faq_id = faq.id');
```

You can see we created a subquery and did a left join. In the subquery, we are selecting AVG(faq_rating), which is the MySql function to calculate an average.

You can also see that we give the \$subQuery variable a name in the leftJoin array, in this case it's 'faqAverage.' And right away you can see that this alias works in the on part of the join. So instead of faq_rating.faq_id, we have faqAverage.faq_id to join on faq.id.

Doing it this way means that the AVG function will loop for each instance of FaqRating where faq_rating.faq_id is equal to faq.id. And this is what we're looking for.

The only thing I don't like about it is that we have already defined the getAverageRating method on the Faq model, so this seems a little redundant. I tested it to see if we needed both and it turns out that we do.

As far as I can tell, the sorting capability we want relies on this subQuery, and the actual line by line results rely on the model method. If we drop anything, it will not work.

So moving on to our setSort method, we add the following block.

Gist:

[setSort Block](#)

From book:

```
'averageRating'=>[
    'asc'=>[ 'faqAverage.average_rating'=>SORT_ASC],
    'desc'=>[ 'faqAverage.average_rating'=>SORT_DESC],
    'label'=>'Rating'
],
```

Remember that averageRating is our getAverageRating method on the base model and faqAverage is the label we gave our subQuery.

The last piece to pop in the search method is the line to filter by average rating:

```
// filter by average rating

$query->andWhere(['faqAverage.average_rating'=>$this->averageRating]);
```

The way we did this in previous examples was a little more verbose. It would look like this:

```
$query->joinWith(['faqRating' => function ($q) {
    $q->andFilterWhere(['faqAverage.average_rating'=>$this->averageRating]);
}]);
```

I checked the debug toolbar for DB queries and there was no difference in performance between the two, so I went with the shorter line.

Ok, so the last little bit is the addition of one line in Gridview in backend/views/faq/index.php:

```
'averageRating',
```

I put that on the line immediately following faq_weight. And that should do it. You now have a fully sortable column for a calculated value, in this the case, the average rating of the faq.

Times Rated

I was curious about how we might add additional calculated values, so I decided to return a calculated value for the number of times an faq is voted on.

I'm going to step through this quickly because most of it is exactly the same as what we just did. So we'll start with our Faq model method.

Gist:

[getRatingsCount](#)

From book:

```
public function getRatingsCount()
{
    return $this
        ->hasMany(FaqRating::className(), ['faq_id'=>'id'])
        ->count('faq_rating');
}
```

Next we make a label:

```
'ratingsCount' => Yii::t('app', 'Times Rated'),
```

Now we move on to FaqSearch model. We need to add a public property:

```
public $ratingsCount;
```

Then comes the safe rule, we just add the one attribute:

```
public function rules()
{
    return [
        [['id', 'faq_category_id', 'faq_weight', 'faq_is_featured',
            'created_by', 'updated_by'], 'integer'],
        [['faq_question', 'faq_answer', 'created_at', 'updated_at',
            'faqCategoryName', 'faqCategoryList', 'faqIsFeaturedName',
            'createdByUsername', 'updatedByUsername', 'faq_category',
            'faq_weight', 'averageRating', 'ratingsCount'], 'safe'],
    ];
}
```

Next we work on the query in the search method:

```
$query = Faq::find();

$subQuery = FaqRating::find()
    ->select('faq_id, AVG(`faq_rating`) as average_rating,
              count(faq_rating) as times_rated')
    ->groupBy('faq_id');

$query->leftJoin([
    'faqAverage'=>$subQuery
], 'faqAverage.faq_id = faq.id');
```

Ok, so you can see that we simply added to the subQuery instead of creating a new one.

In the subQuery, we are returning count(faq_rating) as times_rated, so we can use times_rated in our setSort.

That also means we use faqAverage, which identifies the subquery, for this block in the set sort.

```
'ratingsCount'=> [
    'asc'=> [ 'faqAverage.times_rated'=>SORT_ASC] ,
    'desc'=> [ 'faqAverage.times_rated'=>SORT_DESC] ,
    'label'=>'Times Rated'
],
```

And then we'll simply add the filter line in the appropriate place:

```
// filter by rating count
$query->andWhere(['faqAverage.times_rated'=>$this->ratingsCount]);
```

And finally, add the line to the index view in Gridview:

```
'ratingsCount',
```

When everything is saved, here is a screenshot of what you should see:

#	ID	Question	Answer	Category	Weight	Rating	Times Rated	Featured
1	11	Should I use a framework?	Probably, it's a good idea	general	3	4.5	2	no
2	16	is it similar to timesamp?	let's find out	general	100	4	1	yes
3	13	Am I still having fun?	Probably	Specific	95	3	1	yes

Gridview Screenshot

I'm going to give you the Faq model and the FaqSearch model gists for reference in case you need to troubleshoot:

[Faq Model](#)

[FaqSearch Model](#)

Summary

Sortable, calculated values are very important to the businesses that manage data through web applications. Important questions can be answered quickly such as what is the average rating, which one has the highest rating and which one was rated the most times.

The column sorts facilitate these answers, so admins can get to the key data quickly. This allows companies to follow trend data, which they use to make important decisions.

This was a short focused chapter on returning calculated values in Gridview, a nice way to end the book. I have had a lot of fun writing this stuff and I hope you found it helpful.

Thanks again to everyone who supports this book by sending in typo notices, leaving positive comments and positive reviews, I really appreciate it. See you soon.