FOSUserBundle FTW! (v1.3)

With <3 from SymfonyCasts

Chapter 1: FOSUserBundle

FOSUSERBUNDLE!¶

If you're using Symfony2 and need to load users from your database, then you're in the right place! In the next few minutes, we're going to learn the most important pieces of Symfony's most popular open source bundle: FOSUserBundle. FOSUserBundle offers a ton of features when working with users, including a login form, registration form, forgot password functionality, integration with the database and a lot more. FOSUserBundle isn't really a bundle that provides security, it's more of a bundle that adds a lot of the most common features you need when leveraging Symfony's security system.

Installing FOSUserBundle¶

We're starting out with a fresh Symfony Standard project. Like when installing any bundle, the first step is to find its documentation. FOSUserBundle has excellent documentation, but it's a big bundle, so we'll help you navigate to the important things. Start by copying the needed code into your `deps` file. As of this recording, the latest stable version is 1.1.0 which we'll set here. Now, run the `bin/vendor` script. While that's downloading, head back to the docs, grab the autoload line, and paste it into the autoload file. Finally, enable the bundle in `AppKernel` by copying once again from the docs.

Tip

You can also install FOSUserBundle using Composer. This is now the preferred method for installing bundles.

Generating the User Entity¶

Let's forget about security for a second and pretend that all we care about is creating a `User` entity that is stored in the database. Let's create a new bundle called `UserBundle` to house the new entity. Instead of using the `doctrine:generate:bundle` task, I'll just create a `UserBundle` and make sure there's a properly-named bundle class inside. Don't forget to enable your bundle in `AppKernel`!

To create the new User entity, we can cheat and copy in a skeleton from the documentation. Be sure to change the namespace to the namespace in your real project - ours matches up with the documentation only by coincidence. Change the table name if you'd like. Notice that we're extending a base `User` entity from inside `FOSUserBundle`. This gives us a bunch of fields and functionality related to security. We'll talk about some of these fields in a moment. We won't add any here, but feel free to start adding any other fields you need here - like `firstName` or `birthday`.

Integrating the User Entity with the Security System¶

Now that we have a working `User` entity, we need to integrate it with our security system. Open up `security.yml`. Before we do anything, remove the two extra firewalls. Give the remaining firewall some normal paths for login and logout. Finally, uncomment the anonymous key so that anonymous users are allowed to access our site.

Next, copy the providers key from the documentation. This tells the security system to load users from the database using a class inside `FOSUserBundle`. Next, remove the `encoders` key entirely. Finally, add the provider key under `form_login`. This explicitly states that users should be loaded from the database when using the login form. Since we only have one "provider", this isn't necessary, but it's nice to be clear.

Tip

In the latest `FOSUserBundle`, you will need to add the following encoders key:

```
# app/config/security.yml
# ...

encoders:
    "FOS\UserBundle\Model\UserInterface": sha512
```

Configuring FOSUserBundle¶

Ok, we're almost done! To actually configure some settings inside the bundle itself, copy the configuration from the docs into `app/config/config.yml`. Change the `user_class` to be the fully-qualified namespace to your actual user class. Again, our example just happens to match up with the documentation. The `firewall_name` key needs to match up with the name of our firewall in `security.yml`. The name isn't really important, so let's just rename our firewall to

main.

Importing the Routing¶

Our security system is ready to go, but the bundle also gives us a bunch of pages, like login and registration pages, for free. To get this, we just need to import the routes from `FOSUserBundle`. To keep things organized, create a `routing.yml` file inside `UserBundle` and paste the routes there. Now, go into the main `routing.yml` file and import our new one. To prove that things are working, run the `router:debug` task. Nice! We've just inherited a lot of free functionality! And really, who doesn't like free stuff?

Updating the Database¶

Quickly, we also need to build our database and schema. As you can see, an `acme_user` table is going to be inserted with a number of columns on it. These columns are from the `User` class inside `FOSUserBundle` that we're inheriting.

Activating the Translator¶

Finally, let's see this all in action! Head to the login page by going to `/login`. This is just a normal route and controller page that's provided by `FOSUserBundle` as a convenience. Immediately, we can see that all the text is super-crazy. Internally, `FOSUserBundle` uses translation keys for all of its text. This makes any text you see super-easy to customize. To make it look right, enable the translator library and refresh.

Creating a User¶

If we try to login, it seems to be working, but we don't actually have any users in our database yet. The easiest way to create one is by using one of the commands provided by FOSUserBundle. Run the `fos:user:create` command and fill in some details. When it's done, we can login! Checkout the web debug toolbar to see that we have one role by default: `ROLE_USER`. We can use the `fos:user:promote` task to easily add another role to our user. These roles are stored on an array roles field on our user.

Let's look around. Things are simple, but we already have a working profile page at `/profile` and an edit page at `/profile/edit`. We could even change our password out-of-the-box. If we logout, we can see the registration page, which is also totally functional.

Making the Templates use our Base Layout¶

Back on the login page, things are working, but we're stuck inside an ugly base layout. Let's look inside the bundle's code to see what's going on. The login page stores its content inside a block called `fos_user_content` and extends a `layout.html.twig` template that's also in the bundle. That's fine, but how can we make the login form appear in our layout instead of this plain one?

Setting a Parent Bundle¶

To answer this question, open up the UserBundle class and add a new method called `getParent`. By setting FOSUserBundle as the parent, we're telling Symfony that we'd like to be able to override certain things inside that bundle. For example, by creating a file called `layout.html.twig` in the exact same directory as FOSUserBundle, our template will override the original one. This is great! Make our template extend the correct base layout in our project. If we look inside it, we'll see that our main content area is called body. By using a little Twig magic, we can take the content that's placed in the `fos_user_content` block and echo it into our body block.

Refresh the page to check this out. Awesome! The login page now extends our `layout.html.twig` file, which pushes the content into the correct block and extends the real base layout. If we go to the register page, it works too! In fact, every template inside FOSUserBundle extends `layout.html.twig`. So, by fixing it, we've fixed everything.

Overriding Individual Templates¶

Let's keep going with this. The login page looks terrible. Copy `login.html.twig` from the bundle and place it into a Security directory. Like before, this template now overrides the one from the bundle. Add a little bit of markup to clean things up. Refresh to see a much better-looking page. Overriding templates from the bundle is not an edge-case, it's something you'll probably do for nearly all of its templates. So, starting customizing!

Translating Form Labels¶

The biggest problem on the registration page now are the labels, which are a bit "technical". Fortunately, these are really easy to fix. Head to the bundle and copy the `FOSUserBundle.en.yml` file. Paste it into a translations directory in the bundle. This is the English translation file from the bundle, and we can customize any text from the bundle just by changing it. You can either copy the whole file or only include the individual translations you want to override. If you don't see the updated translation immediately, just clear your cache: Symfony doesn't see the new translation file until you do.

Introducing the UserManager¶

Now that everything looks good, let's dive into some of the internal workings of the bundle. Head to the `WelcomeController` that's part of the standard edition. To fetch our user, grab a service called `fos_user.user_manager` and call `findUserByUsername` on it. This object is called a `UserManager` and it's basically the command center for doing anything related to users. If you want to figure out how the bundle works, check it out to find out more. And while you'll commonly see the UserManager used to fetch and save objects, you can still just query and save objects through the entity manager like normal.

Head to the browser and go to the homepage. The user manager finds our user and dumps out all of its fields. There are quite a few, so let's look at some of the more important ones:

usernameCanonical and emailCanonical¶

`username` and `email` are easy, but what the heck is `usernameCanonical` and `emailCanonical`? GREAT question. This is probably the biggest complaint we hear about the bundle, but it's quite simple. The username and email field contain the exact string you enter during registration, including all lowercase or capital letters. The canonical fields are set automatically when you save a user, and are the all-lowercase version. When you login, the username or email you enter is converted to lowercase and is queried for on the canonical fields. This was done so that even if your database is case-sensitive, the username or email on your login form is not. Long-story short, don't worry about these fields... ever.

MySQL is not case-sensitive, which is why this is unnecessary for most users Most of the rest of the fields deal with the password reset functionality, optional opt-in confirmation, user-disabling feature and more. The most important one to notice is roles, which as mentioned earlier, holds the array of roles that this user should have.

More Configuration¶

Well, that's about it for now! As you can see from the documentation, this bundle is HUGE. You'll undoubtedly need to override controllers and forms when using the bundle, so definitely read those sections. Inside the configuration section, we can see a few important options. In particular, in the registration/confirmation section you can turn on an opt-in registration feature. If this is on, a user will be sent an email after registration. Their account won't be active until they click that link.

You now have a great installation and understanding of `FOSUserBundle`. The only real downside to using FOSUserBundle is that you will run into added complexity when trying to override some of its features. For example, there's just a little bit of work involved to override the registration form in order to add more fields to it. But remember, the bundle is here to help you, not bind you. If you ever get stuck, you can always create your own functionality. You can use the bundle but still create your own registration form. Like always with Symfony2, you're in the driver's seat.

Alright, that's it for now! I hope I'll see you in future Knp screencasts. Also, be sure to checkout knpbundles.com if you're curious about all the open source bundles that you can bring into your app. Seeya next time!