# Getting Started With FOSUserBundle

The Symfony2 security component provides a flexible security framework that allows you to load users from configuration, a database, or anywhere else you can imagine. The FOSUserBundle builds on top of this to make it quick and easy to store users in a database.

So, if you need to persist and fetch the users in your system to and from a database, then you're in the right place.

## Prerequisites

This version of the bundle requires Symfony 2.1+. If you are using Symfony 2.0.x, please use the 1.2.x releases of the bundle.

### Translations

If you wish to use default texts provided in this bundle, you have to make sure you have translator enabled in your config.

```
# app/config/config.yml

framework:
    translator: ~
```

For more information about translations, check Symfony documentation.

## Installation

Installation is a quick (I promise!) 7 step process:

1. Download FOSUserBundle using composer
2. Enable the Bundle
3. Create your User class
4. Configure your application's security.yml
5. Configure the FOSUserBundle
6. Import FOSUserBundle routing
7. Update your database schema

### Step 1: Download FOSUserBundle using composer

Add FOSUserBundle by running the command:

```
$ php composer.phar require friendsofsymfony/user-bundle "~2.0@dev"
```

Composer will install the bundle to your project's `vendor/friendsofsymfony` directory.

## Step 2: Enable the bundle

Enable the bundle in the kernel:

```
<?php
// app/AppKernel.php

public function registerBundles()
{
    $bundles = array(
        // ...
        new FOS\UserBundle\FOSUserBundle(),
    );
}
```

## Step 3: Create your User class

The goal of this bundle is to persist some `User` class to a database (MySql, MongoDB, CouchDB, etc). Your first job, then, is to create the `User` class for your application. This class can look and act however you want: add any properties or methods you find useful. This is *your* `User` class.

The bundle provides base classes which are already mapped for most fields to make it easier to create your entity. Here is how you use it:

1. Extend the base `User` class (from the `Model` folder if you are using any of the doctrine variants, or `Propel` for propel)
2. Map the `id` field. It must be protected as it is inherited from the parent class.

**Warning:**

> When you extend from the mapped superclass provided by the bundle, don't redefine the mapping for the other fields as it is provided by the bundle.

Your `User` class can live inside any bundle in your application. For example, if you work at "Acme" company, then you might create a bundle called `AcmeUserBundle` and place your `User` class in it.

In the following sections, you'll see examples of how your `User` class should look, depending on how you're storing your users (Doctrine ORM, MongoDB ODM, or CouchDB ODM).

**Note:**

> The doc uses a bundle named `AcmeUserBundle`. If you want to use the same name, you need to register it in your kernel. But you can of course place your user class in the bundle you want.

**Warning:**

If you override the __construct() method in your User class, be sure to call parent::__construct(), as the base User class depends on this to initialize some fields.

### a) Doctrine ORM User class

If you're persisting your users via the Doctrine ORM, then your `User` class should live in the `Entity` namespace of your bundle and look like this to start:

#### Annotations

```
<?php
// src/Acme/UserBundle/Entity/User.php

namespace Acme\UserBundle\Entity;

use FOS\UserBundle\Model\User as BaseUser;
use Doctrine\ORM\Mapping as ORM;

/**
 * @ORM\Entity
 * @ORM\Table(name="fos_user")
 */
class User extends BaseUser
{
    /**
     * @ORM\Id
     * @ORM\Column(type="integer")
     * @ORM\GeneratedValue(strategy="AUTO")
     */
    protected $id;

    public function __construct()
    {
        parent::__construct();
        // your own logic
    }
}
```

**Note:**

`User` is a reserved keyword in SQL so you cannot use it as table name.

#### yaml

If you use yml to configure Doctrine you must add two files. The Entity and the orm.yml:

```
<?php
// src/Acme/UserBundle/Entity/User.php

namespace Acme\UserBundle\Entity;
```

```
use FOS\UserBundle\Model\User as BaseUser;

/**
 * User
 */
class User extends BaseUser
{
    public function __construct()
    {
        parent::__construct();
        // your own logic
    }
}
```

```
# src/Acme/UserBundle/Resources/config/doctrine/User.orm.yml
Acme\UserBundle\Entity\User:
    type:  entity
    table: fos_user
    id:
        id:
            type: integer
            generator:
                strategy: AUTO
```

**xml**

If you use xml to configure Doctrine you must add two files. The Entity and the orm.xml:

```
<?php
// src/Acme/UserBundle/Entity/User.php

namespace Acme\UserBundle\Entity;

use FOS\UserBundle\Model\User as BaseUser;

/**
 * User
 */
class User extends BaseUser
{
    public function __construct()
    {
        parent::__construct();
        // your own logic
    }
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<!-- src/Acme/UserBundle/Resources/config/doctrine/User.orm.xml -->
<doctrine-mapping xmlns="http://doctrine-project.org/schemas/orm/doctrine-mapping"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://doctrine-project.org/schemas/orm/doctrine-mapping
http://doctrine-project.org/schemas/orm/doctrine-mapping.xsd">
```

```
    <entity name="Acme\UserBundle\Entity\User" table="fos_user">
        <id name="id" type="integer" column="id">
            <generator strategy="AUTO"/>
        </id>
    </entity>
</doctrine-mapping>
```

### b) MongoDB User class

If you're persisting your users via the Doctrine MongoDB ODM, then your `User` class should live in the `Document` namespace of your bundle and look like this to start:

```
<?php
// src/Acme/UserBundle/Document/User.php

namespace Acme\UserBundle\Document;

use FOS\UserBundle\Model\User as BaseUser;
use Doctrine\ODM\MongoDB\Mapping\Annotations as MongoDB;

/**
 * @MongoDB\Document
 */
class User extends BaseUser
{
    /**
     * @MongoDB\Id(strategy="auto")
     */
    protected $id;

    public function __construct()
    {
        parent::__construct();
        // your own logic
    }
}
```

### c) CouchDB User class

If you're persisting your users via the Doctrine CouchDB ODM, then your `User` class should live in the `CouchDocument` namespace of your bundle and look like this to start:

```
<?php
// src/Acme/UserBundle/Document/User.php

namespace Acme\UserBundle\CouchDocument;

use FOS\UserBundle\Model\User as BaseUser;
use Doctrine\ODM\CouchDB\Mapping\Annotations as CouchDB;

/**
 * @CouchDB\Document
 */
```

```
class User extends BaseUser
{
    /**
     * @CouchDB\Id
     */
    protected $id;

    public function __construct()
    {
        parent::__construct();
        // your own logic
    }
}
```

**d) Propel User class**

If you don't want to add your own logic in your user class, you can simply use `FOS\UserBundle\Propel\User` as user class and you don't have to create another class.

If you want to add your own fields, you can extend the model class by overriding the database schema. Just copy the `Resources/config/propel/schema.xml` file to `app/Resources/FOSUserBundle/config/propel/schema.xml`, and customize it to fit your needs.

## Step 4: Configure your application's security.yml

In order for Symfony's security component to use the FOSUserBundle, you must tell it to do so in the `security.yml` file. The `security.yml` file is where the basic security configuration for your application is contained.

Below is a minimal example of the configuration necessary to use the FOSUserBundle in your application:

```
# app/config/security.yml
security:
    encoders:
        FOS\UserBundle\Model\UserInterface: sha512

    role_hierarchy:
        ROLE_ADMIN:       ROLE_USER
        ROLE_SUPER_ADMIN: ROLE_ADMIN

    providers:
        fos_userbundle:
            id: fos_user.user_provider.username

    firewalls:
        main:
            pattern: ^/
            form_login:
                provider: fos_userbundle
                csrf_provider: form.csrf_provider
            logout:       true
            anonymous:    true

    access_control:
        - { path: ^/login$, role: IS_AUTHENTICATED_ANONYMOUSLY }
        - { path: ^/register, role: IS_AUTHENTICATED_ANONYMOUSLY }
        - { path: ^/resetting, role: IS_AUTHENTICATED_ANONYMOUSLY }
        - { path: ^/admin/, role: ROLE_ADMIN }
```

Under the `providers` section, you are making the bundle's packaged user provider service available via the alias `fos_userbundle`. The id of the bundle's user provider service is `fos_user.user_provider.username`.

Next, take a look at and examine the `firewalls` section. Here we have declared a firewall named `main`. By specifying `form_login`, you have told the Symfony2 framework that any time a request is made to this firewall that leads to the user needing to authenticate himself, the user will be redirected to a form where he will be able to enter his credentials. It should come as no surprise then that you have specified the user provider service we declared earlier as the provider for the firewall to use as part of the authentication process.

**Note:**

> Although we have used the form login mechanism in this example, the FOSUserBundle user provider service is compatible with many other authentication methods as well. Please read the Symfony2 Security component documentation for more information on the other types of authentication methods.

The `access_control` section is where you specify the credentials necessary for users trying to access specific parts of your application. The bundle requires that the login form and all the routes used to create a user and reset the password be available to unauthenticated users but use the same firewall as the pages you want to secure with the bundle. This is why you have specified that any request matching the `/login` pattern or starting with `/register` or `/resetting` have been made available to anonymous users. You have also specified that any request beginning with `/admin` will require a user to have the `ROLE_ADMIN` role.

For more information on configuring the `security.yml` file please read the Symfony2 security component documentation.

**Note:**

> Pay close attention to the name, `main`, that we have given to the firewall which the FOSUserBundle is configured in. You will use this in the next step when you configure the FOSUserBundle.

## Step 5: Configure the FOSUserBundle

Now that you have properly configured your application's `security.yml` to work with the FOSUserBundle, the next step is to configure the bundle to work with the specific needs of your application.

Add the following configuration to your `config.yml` file according to which type of datastore you are using.

```
# app/config/config.yml
fos_user:
    db_driver: orm # other valid values are 'mongodb', 'couchdb' and 'propel'
    firewall_name: main
    user_class: Acme\UserBundle\Entity\User
```

Or if you prefer XML:

```
<!-- app/config/config.xml -->

<!-- other valid 'db-driver' values are 'mongodb' and 'couchdb' -->
<fos_user:config
    db-driver="orm"
    firewall-name="main"
    user-class="Acme\UserBundle\Entity\User"
/>
```

Only three configuration values are required to use the bundle:

- The type of datastore you are using (`orm`, `mongodb`, `couchdb` or `propel`).
- The firewall name which you configured in Step 4.
- The fully qualified class name (FQCN) of the `User` class which you created in Step 3.

**Note:**

> FOSUserBundle uses a compiler pass to register mappings for the base User and Group model classes with the object manager that you configured it to use. (Unless specified explicitly, this is the default manager of your doctrine configuration.)

## Step 6: Import FOSUserBundle routing files

Now that you have activated and configured the bundle, all that is left to do is import the FOSUserBundle routing files.

By importing the routing files you will have ready made pages for things such as logging in, creating users, etc.

In YAML:

```
# app/config/routing.yml
fos_user:
    resource: "@FOSUserBundle/Resources/config/routing/all.xml"
```

Or if you prefer XML:

```
<!-- app/config/routing.xml -->
<import resource="@FOSUserBundle/Resources/config/routing/all.xml"/>
```

**Note:**

> In order to use the built-in email functionality (confirmation of the account, resetting of the password), you must activate and configure the SwiftmailerBundle.

### Step 7: Update your database schema

Now that the bundle is configured, the last thing you need to do is update your database schema because you have added a new entity, the `User` class which you created in Step 4.

For ORM run the following command.

```
$ php app/console doctrine:schema:update --force
```

For MongoDB users you can run the following command to create the indexes.

```
$ php app/console doctrine:mongodb:schema:create --index
```

For Propel users you have to install the TypehintableBehavior before to build your model. First, install it:

```
{
    "require": {
        "willdurand/propel-typehintable-behavior": "~1.0"
    }
}
```

You now can run the following command to create the model:

```
$ php app/console propel:build
```

> To create SQL, run the command `propel:build --insert-sql` or use migration commands if you have an existing schema in your database.

You now can login at `http://app.com/app_dev.php/login`!

## Next Steps

Now that you have completed the basic installation and configuration of the FOSUserBundle, you are ready to learn about more advanced features and usages of the bundle.

The following documents are available:

- Overriding Templates
- Hooking into the controllers
- Overriding Controllers
- Overriding Forms
- Using the UserManager
- Command Line Tools
- Logging by username or email
- Transforming a username to a user in forms
- Emails
- Using the groups
- More about the Doctrine implementations
- Supplemental Documentation
- Replacing the canonicalizer
- Using a custom storage layer
- Configuration Reference
- Adding invitations to registration
- Advanced routing configuration