NucleosUserBundle

Dec 29, 2021

Contents

The Symfony Security component provides a flexible security framework that allows you to load users from configuration, a database, or anywhere else you can imagine. The NucleosUserBundle builds on top of this to make it quick and easy to store users in a database, as well as functionality for reset password.

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.

The following documents are available:

CHAPTER 1

Installation

1.1 Prerequisites

1.1.1 Translations

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

```
# config/packages/framework.yaml
framework:
    translator: ~
```

For more information about translations, check Symfony documentation.

1.2 Installation

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

1.2.1 Step 1: Download NucleosUserBundle using composer

Require the bundle with composer:

```
$ composer require nucleos/user-bundle
```

If you encounter installation errors pointing at a lack of configuration parameters, such as `The child node "db_driver" at path "nucleos_user" must be configured`, you should complete the configuration in Step 5 first and then re-run this step.

1.2.2 Step 2: Enable the bundle

Enable the bundle in the kernel:

```
// config/bundles.php
return [
    // ...
    Nucleos\UserBundle\NucleosUserBundle::class => ['all' => true],
    // ...
]
```

1.2.3 Step 3: Create your User class

The goal of this bundle is to persist some `User` class to a database (MySql, MongoDB, 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)
2. Map the `id` field. It must be protected as it is inherited from the parent class.

Caution: 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.

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 or MongoDB ODM).

Note: The doc uses a bundle named `App` according to the Symfony best practices. However, you can of course place your user class in the bundle you want.

Caution: 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:

```php
// src/Entity/User.php
namespace App\Entity;

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

/**
 * @ORM\Entity
 * @ORM\Table(name="nucleos_user__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
    }
}
```

Caution: `user` is a reserved keyword in the SQL standard. If you need to use reserved words, surround them with backticks, *e.g.* `@ORM\Table(name="`user`")`

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/Document/User.php
namespace App\Document;

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

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

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

(continues on next page)

(continued from previous page)

```
    }
}
```

1.2.4 Step 4: Configure your application's security.yaml

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

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

```yaml
# config/packages/security.yaml
security:
    encoders:
        Nucleos\UserBundle\Model\UserInterface: auto

    role_hierarchy:
        ROLE_ADMIN:       ROLE_USER
        ROLE_SUPER_ADMIN: ROLE_ADMIN

    providers:
        nucleos_userbundle:
            id: nucleos_user.user_provider.username

    firewalls:
        main:
            pattern: ^/
            user_checker: Nucleos\UserBundle\Security\UserChecker
            form_login:
                provider: nucleos_userbundle
                csrf_token_generator: security.csrf.token_manager

            logout:       true
            anonymous:    true

    access_control:
        - { path: ^/change-password, role: IS_AUTHENTICATED_REMEMBERED }
        # If you have an admin backend, uncomment the following line
        # - { path: ^/admin/, role: ROLE_ADMIN }
```

Under the `providers` section, you are making the bundle's packaged user provider service available via the alias `nucleos_userbundle`. The id of the bundle's user provider service is `nucleos_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 Symfony 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 NucleosUserBundle user provider service is compatible with many other authentication methods as well. Please read the Symfony 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 `/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.yaml` file please read the Symfony security component documentation.

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

1.2.5 Step 5: Configure the NucleosUserBundle

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

Add the following configuration to your `config/packages/nucleos_user.yaml` file according to which type of datastore you are using.

```yaml
# config/packages/nucleos_user.yaml
nucleos_user:
    db_driver: orm # other valid values is 'mongodb'
    firewall_name: main
    user_class: App\Entity\User
    from_email:   "%mailer_user%"
    loggedin:
        route: 'home' # Redirect route after login
```

Only four configuration's nodes are required to use the bundle:

- The type of datastore you are using (`orm` or `mongodb`).
- 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: NucleosUserBundle 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.)

1.2.6 Step 6: Import NucleosUserBundle routing files

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

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

```yaml
# config/routes/nucleos_user.yaml
nucleos_user_security:
    resource: "@NucleosUserBundle/Resources/config/routing/security.php"

nucleos_user_resetting:
```

(continues on next page)

(continued from previous page)

```
    resource: "@NucleosUserBundle/Resources/config/routing/resetting.php"
    prefix: /resetting

nucleos_user_change_password:
    resource: "@NucleosUserBundle/Resources/config/routing/change_password.php"
    prefix: /security

nucleos_user_deletion:
    resource: "@NucleosUserBundle/Resources/config/routing/deletion.php"
    prefix: /deletetion
```

1.2.7 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 bin/console doctrine:schema:update --force
```

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

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

CHAPTER 2

Hooking into the Controllers

The controllers packaged with the NucleosUserBundle provide a lot of functionality that is sufficient for general use cases. But, you might find that you need to extend that functionality and add some logic that suits the specific needs of your application.

For this purpose, the controllers are dispatching events in many places in their logic. All events can be found in the constants of the `Nucleos\UserBundle\NucleosUserEvents` class.

All controllers follow the same convention: they dispatch a `SUCCESS` event when the form is valid before saving the user, and a `COMPLETED` event when it is done. Thus, all `SUCCESS` events allow you to set a response if you don't want the default redirection. And all `COMPLETED` events give you access to the response before it is returned.

Controllers with a form also dispatch an `INITIALIZE` event after the entity is fetched, but before the form is created.

For instance, this listener will change the redirection after the password resetting to go to the homepage.

```php
// src/App/EventListener/PasswordResettingListener.php
namespace App\EventListener;

use Nucleos\UserBundle\Event\FormEvent;
use Nucleos\UserBundle\NucleosUserEvents;
use Symfony\Component\EventDispatcher\EventSubscriberInterface;
use Symfony\Component\HttpFoundation\RedirectResponse;
use Symfony\Component\Routing\Generator\UrlGeneratorInterface;

/**
 * Listener responsible to change the redirection at the end of the password resetting
 */
class PasswordResettingListener implements EventSubscriberInterface
{
    private $router;

    public function __construct(UrlGeneratorInterface $router)
    {
        $this->router = $router;
    }
```

(continues on next page)

(continued from previous page)

```
    public static function getSubscribedEvents(): array
    {
        return [
            NucleosUserEvents::RESETTING_RESET_SUCCESS => 'onPasswordResettingSuccess
↪',
        ];
    }

    public function onPasswordResettingSuccess(FormEvent $event): void
    {
        $url = $this->router->generate('homepage');

        $event->setResponse(new RedirectResponse($url));
    }
}
```

You can then register this listener:

```
# config/services.yaml
services:
    app.password_resetting:
        class: App\EventListener\PasswordResettingListener
        arguments: ['@router']
        tags:
            - { name: kernel.event_subscriber }
```

CHAPTER 3

User Manager

In order to be storage agnostic, all operations on the user instances are handled by a user manager implementing `Nucleos\UserBundle\Model\UserManagerInterface`. Using it ensures that your code will continue to work if you change the storage. The controllers provided by the bundle use the configured user manager instead of interacting directly with the storage layer.

If you configure the `db_driver` option to `orm`, this service is an instance of `Nucleos\UserBundle\Doctrine\UserManager`.

If you configure the `db_driver` option to `mongodb`, this service is an instance of `Nucleos\UserBundle\Doctrine\UserManager`.

3.1 Accessing the User Manager service

The user manager is available in the container as a `Nucleos\UserBundle\Model\UserManagerInterface` service.

```
use Nucleos\UserBundle\Model\UserManagerInterface;

public function someAction(UserManagerInterface $manager)
{
    // ...
}
```

3.2 Creating a new User

A new instance of your User class can be created by the user manager.

```
$user = $userManager->createUser();
```

`$user` is now an instance of your user class.

Note: This method will not work if your user class has some mandatory constructor arguments.

3.3 Retrieving the users

The user manager has a few methods to find users based on the unique fields (username, email and confirmation token) and a method to retrieve all existing users.

- `findUserByUsername($username)`
- `findUserByEmail($email)`
- `findUserByConfirmationToken($token)`
- `findUserBy(['id'=>$id])`
- `findUsers()`

To save a user object, you can use the `updateUser` method of the user manager. This method will update the encoded password and the canonical fields and then persist the changes.

3.4 Updating a User object

```
$user = $userManager->createUser();
$user->setUsername('John');
$user->setEmail('john.doe@example.com');

$userManager->updateUser($user);
```

Note: To make it easier, the bundle comes with a Doctrine listener handling the update of the password and the canonical fields for you behind the scenes. If you always save the user through the user manager, you may want to disable it to improve performance.

```
# config/packages/nucleos_user.yaml
nucleos_user:
    # ...
    use_listener: false
```

Note: For the Doctrine implementations, the default behavior is to flush the unit of work when calling the `updateUser` method. You can disable the flush by passing a second argument set to `false`. This will then be equivalent to calling `updateCanonicalFields` and `updatePassword`.

An ORM example:

```
use Nucleos\UserBundle\Model\UserManagerInterface;

class MainController
{
```

(continues on next page)

(continued from previous page)

```
    public function updateAction(UserManagerInterface $userManager, $id)
    {
        $user = // get a user from the datastore

        $user->setEmail($newEmail);

        $userManager->updateUser($user, false);

        // make more modifications to the database

        $this->getDoctrine()->getManager()->flush();
    }
}
```

3.5 Overriding the User Manager

You can replace the default implementation of the user manager by defining a service implementing `Nucleos\UserBundle\Model\UserManagerInterface` and setting its id in the configuration. The id of the default implementation is `nucleos_user.user_manager.default`

```
nucleos_user:
    # ...
    service:
        user_manager: custom_user_manager_id
```

Your custom implementation can extend `Nucleos\UserBundle\Model\UserManager` to reuse the common logic.

3.6 SecurityBundle integration

The bundle provides several implementation of `Symfony\Component\Security\Core\UserProviderInterface` on top of the `UserManagerInterface`.

CHAPTER 4

Command line tools

The NucleosUserBundle provides a number of command line utilities to help manage your application's users. Commands are available for the following tasks:

1. Create a User
2. Activate a User
3. Deactivate a User
4. Promote a User
5. Demote a User
6. Change a User's Password

Note: You must have correctly installed and configured the NucleosUserBundle before using these commands.

4.1 Create a User

You can use the `nucleos:user:create` command to create a new user for your application. The command takes three arguments, the `username`, `email`, and `password` for the user you are creating.

For example if you wanted to create a user with username `testuser`, with email `test@example.com` and password `p@ssword`, you would run the command as follows.

```
$ php bin/console nucleos:user:create testuser test@example.com p@ssword
```

If any of the required arguments are not passed to the command, an interactive prompt will ask you to enter them. For example, if you ran the command as follows, then you would be prompted to enter the `email` and `password` for the user you want to create.

```
$ php bin/console nucleos:user:create testuser
```

There are two options that you can pass to the command as well. They are `--super-admin` and `--inactive`.

Specifying the `--super-admin` option will flag the user as a super admin when the user is created. A super admin has access to any part of your application. An example is provided below:

```
$ php bin/console nucleos:user:create adminuser --super-admin
```

If you specify the `--inactive` option, then the user that you create will no be able to log in until he is activated.

```
$ php bin/console nucleos:user:create testuser --inactive
```

4.2 Activate a User

The `nucleos:user:activate` command activates an inactive user. The only argument that the command requires is the `username` of the user who should be activated. If no `username` is specified then an interactive prompt will ask you to enter one. An example of using this command is listed below.

```
$ php bin/console nucleos:user:activate testuser
```

4.3 Deactivate a User

The `nucleos:user:deactivate` command deactivates a user. Like the activate command, the only required argument is the `username` of the user who should be activated. If no `username` is specified then an interactive prompt will ask you to enter one. Below is an example of using this command.

```
$ php bin/console nucleos:user:deactivate testuser
```

4.4 Promote a User

The `nucleos:user:promote` command enables you to add a role to a user or make the user a super administrator.

If you would like to add a role to a user you pass the `username` of the user as the first argument to the command and the `role` to add to the user as the second.

```
$ php bin/console nucleos:user:promote testuser ROLE_ADMIN
```

You can promote a user to a super administrator by passing the `--super` option after specifying the `username`.

```
$ php bin/console nucleos:user:promote testuser --super
```

If any of the arguments to the command are not specified then an interactive prompt will ask you to enter them.

Note: You may not specify the `role` argument and the `--super` option simultaneously.

Caution: Changes will not be applied until the user logs out and back in again.

4.5 Demote a User

The `nucleos:user:demote` command is similar to the promote command except that instead of adding a role to the user it removes it. You can also revoke a user's super administrator status with this command.

If you would like to remove a role from a user you pass the `username` of the user as the first argument to the command and the `role` to remove as the second.

```
$ php bin/console nucleos:user:demote testuser ROLE_ADMIN
```

To revoke the super administrator status of a user, pass the `username` as an argument to the command as well as the `--super` option.

```
$ php bin/console nucleos:user:demote testuser --super
```

If any of the arguments to the command are not specified then an interactive prompt will ask you to enter them.

Note: You may not specify the `role` argument and the `--super` option simultaneously.

Caution: Changes will not be applied until the user logs out and back in again. This has implications for the way in which you configure sessions in your application since you want to ensure that users are demoted as quickly as possible.

4.6 Change a User's Password

The `nucleos:user:change-password` command provides an easy way to change a user's password. The command takes two arguments, the `username` of the user whose password you would like to change and the new `password`.

```
$ php bin/console nucleos:user:change-password testuser newp@ssword
```

If you do not specify the `password` argument then an interactive prompt will ask you to enter one.

CHAPTER 5

Logging in by Username or Email

The bundle provides a built-in user provider implementation using both the username and email fields. To use it, change the id of your user provider to use this implementation instead of the base one using only the username:

```yaml
# config/packages/security.yaml
security:
    providers:
        nucleos_userbundle:
            id: nucleos_user.user_provider.username_email
```

CHAPTER 6

Sending E-Mails

The NucleosUserBundle has built-in support for sending emails in two different instances.

6.1 Password Reset

An email is also sent when a user has requested a password reset. The NucleosUserBundle provides password reset functionality in a two-step process. First the user must request a password reset. After the request has been made, an email is sent containing a link to visit. Upon visiting the link, the user will be identified by the token contained in the url. When the user visits the link and the token is confirmed, the user will be presented with a form to enter in a new password.

6.2 Default Mailer Implementations

The bundle comes with three mailer implementations. They are listed below by service id:

- `nucleos_user.mailer.simple` is the default implementation, and uses symfony mailer to send emails.
- `nucleos_user.mailer.noop` is a mailer implementation which performs no operation, so no emails are sent.

6.3 Configuring the Sender Email Address

The NucleosUserBundle default mailer allows you to configure the sender email address of the emails sent out by the bundle. You can configure the address globally or on a per email basis.

To configure the sender email address for all emails sent out by the bundle, update your `nucleos_user` config as follows:

```yaml
# config/packages/nucleos_user.yaml
nucleos_user:
    # ...
    from_email:   noreply@example.com
```

The bundle also provides the flexibility of allowing you to configure the sender email address for the emails individually.

You can similarly update the `nucleos_user` config to change the sender email address for the password reset request email:

```yaml
# config/packages/nucleos_user.yaml
nucleos_user:
    # ...
    resetting:
        email:
            from_email:   resetting@example.com
```

6.4 Using A Custom Mailer

The default mailer service used by NucleosUserBundle relies on the symfony mailer library to send mail. If you would like to use a different library to send emails or change the content of the email you may do so by defining your own service.

First you must create a new class which implements `Nucleos\UserBundle\Mailer\MailerInterface` which is listed below:

```php
namespace Nucleos\UserBundle\Mailer;

use Nucleos\UserBundle\Model\UserInterface;

interface MailerInterface
{

    /**
     * Send an email to a user to confirm the password reset
     *
     * @param UserInterface $user
     */
    function sendResettingEmailMessage(UserInterface $user): void;
}
```

After you have implemented your custom mailer class and defined it as a service, you must update your bundle configuration so that NucleosUserBundle will use it. Set the `mailer` configuration parameter under the `service` section. An example is listed below.

```yaml
# config/packages/nucleos_user.yaml
nucleos_user:
    # ...
    service:
        mailer: app.custom_nucleos_user_mailer
```

CHAPTER 7

Using groups

NucleosUserBundle allows you to associate groups to your users. Groups are a way to group a collection of roles. The roles of a group will be granted to all users belonging to it.

Note: Symfony supports role inheritance so inheriting roles from groups is not always needed. If the role inheritance is enough for your use case, it is better to use it instead of groups as it is more efficient (loading the groups triggers the database).

The only mandatory configuration is the fully qualified class name (FQCN) of your `Group` class which must implement `Nucleos\UserBundle\Model\GroupInterface`.

Below is an example configuration for enabling groups support.

```yaml
# config/packages/nucleos_user.yaml
nucleos_user:
    db_driver: orm
    firewall_name: main
    user_class: App\Entity\User
    group:
        group_class: App\Entity\Group
```

7.1 The Group class

The simplest way to create a Group class is to extend the mapped superclass provided by the bundle.

7.1.1 a) ORM Group class implementation

```php
// src/Entity/Group.php
namespace App\Entity;

```

(continues on next page)

(continued from previous page)

```
use Doctrine\ORM\Mapping as ORM;
use Nucleos\UserBundle\Model\Group as BaseGroup;

/**
 * @ORM\Entity
 * @ORM\Table(name="nucleos_user__group")
 */
class Group extends BaseGroup
{
    /**
     * @ORM\Id
     * @ORM\Column(type="integer")
     * @ORM\GeneratedValue(strategy="AUTO")
     */
     protected $id;
}
```

Note: `Group` is a reserved keyword in SQL so it cannot be used as the table name.

7.1.2 b) MongoDB Group class implementation

```
// src/Document/Group.php
namespace App\Document;

use Doctrine\ODM\MongoDB\Mapping\Annotations as MongoDB;
use Nucleos\UserBundle\Model\Group as BaseGroup;

/**
 * @MongoDB\Document
 */
class Group extends BaseGroup
{
    /**
     * @MongoDB\Id(strategy="auto")
     */
    protected $id;
}
```

7.2 Defining the User-Group relation

The next step is to map the relation in your `User` class.

7.2.1 a) ORM User-Group mapping

```
// src/Entity/User.php
namespace App\Entity;

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

(continues on next page)

(continued from previous page)

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

    /**
     * @ORM\ManyToMany(targetEntity="App\Entity\Group")
     * @ORM\JoinTable(name="nucleos_user_user_group",
     *      joinColumns={@ORM\JoinColumn(name="user_id", referencedColumnName="id")},
     *      inverseJoinColumns={@ORM\JoinColumn(name="group_id", referencedColumnName=
↪"id")}
     * )
     */
    protected $groups;
}
```

7.2.2 b) MongoDB User-Group mapping

```
// src/Document/User.php
namespace App\Document;

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

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

    /**
     * @MongoDB\ReferenceMany(targetDocument="App\Document\Group")
     */
    protected $groups;
}
```

CHAPTER 8

Doctrine Implementations

This chapter describes some things specific to these implementations.

8.1 Using a different object manager than the default one

Using the default configuration , NucleosUserBundle will use the default doctrine object manager. If you are using multiple ones and want to handle your users with a non-default one, you can change the object manager used in the configuration by giving its name to NucleosUserBundle.

```yaml
# config/packages/nucleos_user.yaml
nucleos_user:
    db_driver: orm
    model_manager_name: non_default # the name of your entity manager
```

Note: Using the default object manager is done by setting the configuration option to `null` which is the default value.

8.2 Replacing the mapping of the bundle

None of the Doctrine projects currently allow overwriting part of the mapping of a mapped superclass in the child entity.

If you need to change the mapping (for instance to adapt the field names to a legacy database), one solution could be to write the whole mapping again without inheriting the mapping from the mapped superclass. In such case, your entity should extend directly from `Nucleos\UserBundle\Model\User` (and `Nucleos\UserBundle\Model\Group` for the group). Another solution can be through doctrine attribute and relations overrides.

Caution: It is highly recommended to map all fields used by the bundle (see the mapping files of the bundle in `src/Resources/config/doctrine-mapping/`). Omitting them can lead to unexpected behaviors and should be done carefully.

CHAPTER 9

Canonicalization

NucleosUserBundle stores canonicalized versions of the username and the email which are used when querying and checking for uniqueness. The default implementation makes them case-insensitive to avoid having users whose username only differs because of the case. It uses `mb_convert_case` to achieve this result.

Caution: If you do not have the mbstring extension installed you will need to define your own canonicalizer.

9.1 Replacing the canonicalizers

If you want to change the way the canonical fields are populated, create a class implementing `Nucleos\UserBundle\Util\CanonicalizerInterface` and register it as a service:

```yaml
# config/services.yaml
services:
    app.my_canonicalizer:
        class: App\Util\CustomCanonicalizer
        public: false
```

You can now configure NucleosUserBundle to use your own implementation:

```yaml
# config/packages/nucleos_user.yaml
nucleos_user:
    # ...
    service:
        email_canonicalizer:    app.my_canonicalizer
        username_canonicalizer: app.my_canonicalizer
```

You can of course use different services for each field if you don't want to use the same logic.

Note: The default implementation has the id `nucleos_user.util.canonicalizer.simple`.

CHAPTER 10

Using a custom storage layer

NucleosUserBundle has been designed to allow you to change the storage layer used by your application and keep all of the functionality provided by the bundle.

Implementing a new storage layer requires providing two classes: the user implementation and the corresponding user manager (you will of course need two other classes if you want to use the groups).

The user implementation must implement `Nucleos\UserBundle\Model\UserInterface` and the user manager must implement `Nucleos\UserBundle\Model\UserManagerInterface`. The `Nucleos\UserBundle\Model` namespace provides base classes to make it easier to implement these interfaces.

Note: You need to take care to always call `updateCanonicalFields` and `updatePassword` before saving a user. This is done when calling `updateUser` so you will be safe if you always use the user manager to save the users. If your storage layer gives you a hook in its saving process, you can use it to make your implementation more flexible (this is done for Doctrine using listeners for instance)

10.1 Configuring NucleosUserBundle to use your implementation

To use your own implementation, create a service for your user manager. The following example will assume that its id is `app.custom_user_manager`.

```yaml
# config/packages/nucleos_user.yaml
nucleos_user:
    db_driver: custom  # custom means that none of the built-in implementation is used
    user_class: App\Model\CustomUser
    service:
        user_manager: app.custom_user_manager
    firewall_name: main
```

Note: Your own service can be a private one. NucleosUserBundle will create an alias to make it available through `nucleos_user.user_manager`.

Caution: The validation of the uniqueness of the username and email fields is done using the constraints provided by DoctrineBundle. You will need to take care of this validation when using a custom storage layer, using a custom constraint

CHAPTER 11

Advanced routing configuration

The bundle itself provides no speical security handling. There is no brutforce protection or IP blocking.

11.1 Password restrictions

There is a pattern constraint that can be used to create stronger user passwords:

```yaml
# config/validator/validation.yaml
Nucleos\UserBundle\Form\Model\ChangePassword:
    properties:
        plainPassword:
            - Nucleos\UserBundle\Validator\Constraints\Pattern:
                minUpper: 1
                minLower: 1
                minNumeric: 1
                minSpecial: 1
            - Length:
                min: 12
```

```php
// src/Entity/User.php
namespace App\Entity;

use Nucleos\UserBundle\Model\User as BaseUser;

/**
 * @ORM\Entity
 * @ORM\Table(name="nucleos_user__user")
 */
class User extends BaseUser
{
    /**
     * @ORM\Id
     * @ORM\Column(type="integer")
```

(continues on next page)

(continued from previous page)

```
     * @ORM\GeneratedValue(strategy="AUTO")
     */
    protected $id;

    #[\Nucleos\UserBundle\Validator\Constraints\Pattern(minUpper: 1, minLower: 1,␣
↪minNumeric: 1, minSpecial: 1)]
    protected ?string $plainPassword = null;
}
```

CHAPTER 12

Account deletion

The NucleosUserBundle has built-in support for deleting the user account.

12.1 Enable feature

The feature is disabled by default. You can enable it by using the following configuration:

```
# config/packages/nucleos_user.yaml
nucleos_user:
    # ...
    deletion:
        enable: true
```

Add the routing config:

```
# config/routes/nucleos_user.yaml
nucleos_user_deletion:
    resource: "@NucleosUserBundle/Resources/config/routing/deletion.php"
```

CHAPTER 13

Configuration reference

All available configuration options are listed below with their default values.

```
nucleos_user:
    db_driver:              ~ # Required
    firewall_name:          ~ # Required
    user_class:             ~ # Required
    use_listener:               true
    use_flash_notifications:    true
    model_manager_name:         null  # change it to the name of your entity/document␣
↪manager if you don't want to use the default one.
    use_authentication_listener: true
    from_email:             ~ # Required
    resetting:
        retry_ttl: 7200 # Value in seconds, logic will use as hours
        token_ttl: 86400
        from_email: # Use this node only if you don't want the global email address␣
↪for the resetting email
    service:
        mailer:                 nucleos_user.mailer.simple
        email_canonicalizer:    nucleos_user.util.canonicalizer.simple
        username_canonicalizer: nucleos_user.util.canonicalizer.simple
        token_generator:        nucleos_user.util.token_generator.simple
        user_manager:           nucleos_user.user_manager.default
    group:
        group_class:    ~ # Required when using groups
        group_manager:  nucleos_user.group_manager.default
    loggedin:
        route: ~ # Required
```

CHAPTER 14

Migrate from FOSUserBundle

14.1 Prerequisites

In order to migrate from FOSUserBundle, you must use a supported PHP version and supported symfony version. There is no support for unmaintained PHP or symfony versions!

14.2 Start migration

The main difference between both bundles is the service and configuration prefix. FOS uses *fos_user* and our bundles uses *nucleos_user* (respectively *nucleos_profile*)

14.2.1 Step 1: Update configuration

Install this bundle and the NucleosProfileBundle:

```
$ composer require nucleos/user-bundle nucleos/profile-bundle
```

14.2.2 Step 2: Update configuration

Inside the configurations under *config/packages/* you need to replace the prefix *fos_user* with *nucleos_user* for the most keys. Some keys related to profile management or registration need a *nucleos_profile* prefix.

14.2.3 Step 3: Update routing

The same rule applies for the routing files located under *config/routes*. You need to replace the *@FOSUserBundle* import with *@NucleosProfileBundle*.

14.2.4 Step 4: Clean cache

Now the migration is finished. The last step if to clear the cache.

```
$ php bin/console cache:clear
```

14.2.5 Optional: Use FOSUserBundle polyfill

There is a polyfill for the most used FOS classes and interfaces. This will use PHP aliases to map the old components to the new namespace.

Some interfaces/classes have a different signature and could cause problems. It is safe to use if you or a third party library is not implementing one of the old FOS classes.

Require the bundle with composer:

```
$ composer require nucleos/fos-user-bundle-polyfill
```

Warning: Be aware this library uses the composer replaces function to fake the FOSUserBundle. This can cause problems, if you need specific bundle features.

14.3 Problems

If you have problems feel free to open an issue or have a look at the docs as there are some more internal refactorings:

- Using symfony mailer instead of swiftmailer
- Add strict type hints
- Closing API. Most classes have beed marked as final
- E-Mail address and Usernames are non-nullable
- Forms do not use the user model and have specific form models