Practical Symfony

Step by step guide to create a simple cms in Symfony 3

by Bernard Peh

Practical Symfony

Step-by-step guide to create a simple CMS in Symfony 3

bernard peh

This book is for sale at http://leanpub.com/practicalsymfony3

This version was published on 2017-03-10

Tweet This Book!

Please help bernard peh by spreading the word about this book on Twitter!

The suggested hashtag for this book is #practical-symfony-3.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

https://twitter.com/search?q=#practical-symfony-3

To everyone who loves to code.

Contents

Preface

This is a book documenting the creation of SongBird[1], a simple CMS created using Symfony Full Stack[2] which itself consists of many Symfony Components[3]. Although the resources in symfony.com is great, they are often hard to digest and you need to piece all the code snippets up to build something useful. I wanted to write an application that covers different aspects of Symfony as much as possible. I believe this approach will be helpful for people who are new to Symfony. This idea was conceptuaised in early 2015 but the implementation turned out to be much longer that I thought - I had to upgrade Symfony from 2 to 3 and dropped SonataAdmin in favor of EasyAdmin when I was 90% done.

The project was finally completed in Aug 2016 after much persistence and has gone through several revisions since.

The objective of this project is to:

- Illustrate the power of rapid development with Symfony.
- Reduce the learning curve by sharing step-by-step guide to create a working application. This process should be helpful to anyone who wants to dive into Symfony.
- Kickstart simple Symfony projects with this application.
- Share the fun of programming.

It is important to note that the rationale behind SongBird CMS is to have broad coverage. Therefore, I try not to repeat the same techniques in every chapter. In reality, you might find some techniques work really well and will want to keep reusing them. On the other hand, you might find certain implementations overkill. This is all for learning purpose.

I hope you have fun creating your own version of SongBird by following the exercises in this book, choosing your best arsenal for your future adventures.

Cheers,

Bernard Peh

[1]https://github.com/bernardpeh/songbird
[2]http://symfony.com/projects/symfonyfs
[3]http://symfony.com/components

Introduction

Building an application is like building a Pyramid. You create each piece of functionality bit by bit. You test the functionality and make sure its stable before building the next piece. This cycle continues until you reach the peak - completion.

Choosing the best framework for RAD (Rapid Application Development) has been a topic debated to death. Today, there is no longer such a thing as "The Best Framework" because all modern day frameworks follow the best practice. However, there is such a thing called "The Best Practice". In fact, you can see similar development methodologies being used across all frameworks. So knowing one framework well means you can jump between other frameworks easily. Just as human evolves, different frameworks learn from each other and adapt very fast to new and better technologies.

At the time of writing, NodeJS continues to innovate with PHP closing in fast behind. PHP is the old veteran when comes to web development with the most frameworks in the market. The 2 frameworks that stood out from the pack were Laravel and Symfony. If you are looking to learn a new framework, I highly recommend Symfony because it is one of the more stable modern framework out there. Symfony components have been widely used by many popular projects including Composer, Behat, Codeception, Drupal and Laravel (just to name a few).

Learning Symfony is never an easy task. Many people follow tutorials in google, read up all the documentation in Symfony website[4] and still find it challenging to create a simple application. Why? Because there is too much theory and not enough real world practical examples. Worst still, you can get entangled in technical jargons and advance customisations easily. The fact that Symfony is an extremely flexible framework makes it even harder to master because there are so many ways to achieve the same goal. If you are new to MVC (Model-View-Controller) and RAD, you will find that Symfony has a steep learning curve.

This book aims to lower the learning curve by providing a step by step hands-on approach to guide developers who are new to Symfony to build a simple CMS using good industry practice. Let us call the CMS "SongBird". Hopefully after following all the chapters, your eyes will be opened to RAD and the unlimited possibilities with Symfony.

Audience

This book is targeted at developers who are new to Symfony. If you are already a seasoned PHP Developer, I hope you would pick up some tips here and there.

[4]http://symfony.com

Why Re-invent the Wheel?

At the time of writing, there are already many CMS and a few popular Symfony ones out there. Symfony has the CMF project[5]. Why built a new one?

SongBird is really a tutorial project and not trying to compete in the CMS space.

Is SongBird Reusable?

Definitely. SongBird is not just a tutorial CMS, you can use it as a vanilla framework to kickstart other Symfony projects. It will be a hugh time saver because all common features have been build and configured already. Since you are the one who creates the software, you will have better idea of how the software works and know where to customise things should the need arises.

You can also think of SongBird as the foundation for the CMF project[6]. Once you are comfortable with the basics of building a CMS, you are ready for more complex development.

Chapters Overview

Chapter 1: Survival Skills

A quick introduction to the skills required to learn Symfony.

Chapter 2: What is SongBird

Introduction to what Songbird is and isn't.

Chapter 3: Creating the Dev Environment

Installing Songbird using docker. Docker is fantastic but its a shame that mac users need a work around at the time of writing.

Chapter 4: The Testing Framework Part 1 (Optional)

Introduces and Installing Codeception for Behavioural Testing.

Chapter 5: The Testing Framework Part 2 (Optional)

Writing a sample BDD acceptance test.

Chapter 6: The User Management System Part 1

Introduces and Installing FOSUserBundle for User Management.

Chapter 7: The User Management System Part 2

Generating user CRUD using the command line and a bit of doctrine appetizer.

[5]http://cmf.symfony.com/
[6]http://cmf.symfony.com/

Chapter 8: Doctrine Fixtures and Migrations

Installing and running Doctrine Fixtures and Migrations. It is important to have a consistent way of creating test data and migrating schemas.

Chapter 9: The Admin Panel Part 1

Installing EasyAdminBundle and integrating it with FOSUserBundle.

Chapter 10: BDD With Codeception (Optional)

Writing BDD acceptance tests for user management business rules.

Chapter 11: Customising the Login Process

Customising Twig templates for the login and request password pages.

Chapter 12: The Admin Panel Part 2

Tweaking EasyAdmin UI.

Chapter 13: Internalisation

Getting Songbird to support french as well.

Chapter 14: Uploading Files

Installing VichUploaderBundle and integrating it with EasyAdminBundle.

Chapter 15: Logging User Activities

Creating a simple bundle to log user activities.

Chapter 16: Improving Performance and Troubleshooting

Installing blackfire and improving Symfony performance. Introduces Gulp to manage all frontend assets.

Chapter 17: The Page Manager Part 1

Creating a custom bundle called NestablePageBundle to manage pages. Introduces PHPUnit to write functional tests.

Chapter 18: Making Your Bundle Reusable

Refactoring NestablePageBundle and making it as a separate installable component.

Chapter 19: The Page Manager Part 2

Installing CKEditor to the CMS and creating a custom locale selector.

Chapter 20: The Front View

Creating the frontend controller and view.

Chapter 21: Dependency Injection Revisited

Using Compiler Pass to add user roles to EasyAdminBundle.

Chapter Final

Congratulations. It's time to start build something yourself using Symfony.

Conventions Used in This Book

Each git branch is a chapter. Obviously, chapter_6 branch means it is Chapter 6. Otherwise stated, all path references assumes ∼**/songbird** as the root folder. Always execute commands from the root folder.

To executing commands, You will see a "->" before the command. For example

```
-> git status

On branch chapter_6
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   symfony/app/AppKernel.php
        modified:   symfony/app/config/routing.yml

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        symfony/src/myfolder/

no changes added to commit (use "git add" and/or "git commit -a")
```

This means that in the command line terminal, go to the ∼/songbird folder and type in "git status".

Likewise, a code snippet like this

```
# symfony/app/config/routing.yml
...
Songbird_user:
    resource: "@SongbirdUserBundle/Controller/"
    type:     annotation
    prefix:   /
```

means update or insert this snippet in ∼/songbird/symfony/app/config/routing.yml

or it could mean a comment for you to action like

```
# you should commit your changes now.
-> git commit -m"update file changes"
```

All symfony commands runs under the symfony dir, ie

```
# in the symfony dir
-> bin/console debug:router
```

Learning Symfony

If you are new to RAD and like to learn Symfony, I recommend you to go through the chapters in sequential order. Every time you are on a new chapter, create a new branch based on the previous chapter and try to add or update the code as suggested in the chapter. For example, you have just finished chapter 4 and going into chapter 5.

Commit all your changes in chapter 4 first.

```
-> git commit -m"This is chapter 4 commit comments"
```

Then checkout chapter 5.

```
# this command will create a new mychapter_5 branch based off your current branch
-> git checkout -b mychapter_5
```

We use mychapter_x to differentiate between your work and my work. To look at all the branches available:

```
-> git branch -a

  mychapter_4
* mychapter_5
  ...
  master
  remotes/origin/HEAD -> origin/master
  remotes/origin/chapter_4
  remotes/origin/chapter_5
  ...
```

If you are being lazy and want to use my chapter 4 instead to start chapter 5,

```
-> git checkout -b mychapter_5 origin/chapter_4
```

If you are already getting confused, here are some good git resource[7] to read.

Jumping between Chapters

I have organised the repository such that every chapter will have its own corresponding branch in the code. Feel free to jump between the different chapters and test out the code. However, remember to stash[8] or commit your changes before switching to a new branch. Also remember to clear your cache if things are broken.

Chapters that talk about Codeception Testing Framework[9] are optional. Feel free to skip them if you already know testing.

To clear the cache fully,

```
rm -rf symfony/var/cache/*
```

Regenerating Bootstrap Cache

If you are getting errors on bootstrap.php.cache, you can regenerate it easily.

```
-> symfony/vendor/sensio/distribution-bundle/Resources/bin/build_bootstrap.php
```

Composer Memory Errors

Refer to composer troubleshooting guide[10] if you have problems using composer.

A common issue is when you get allowed memory exhausted error. A quick workaround is

```
-> php -d memory_limit=-1 path_to_composer update
```

Reinstalling Symfony

Some directories are needed by Symfony but they are not version controlled (eg. the bin directory). In case they have been deleted accidentally, you can reinstall the packages. The re-installation process will not mess up with your existing code. That's the beauty of being modular.

[7]https://help.github.com/articles/good-resources-for-learning-git-and-github/
[8]https://git-scm.com/book/en/v1/Git-Tools-Stashing
[9]http://codeception.com/
[10]https://getcomposer.org/doc/articles/troubleshooting.md

```
rm -rf vendor
composer update
```

Installing the Demo (Optional)

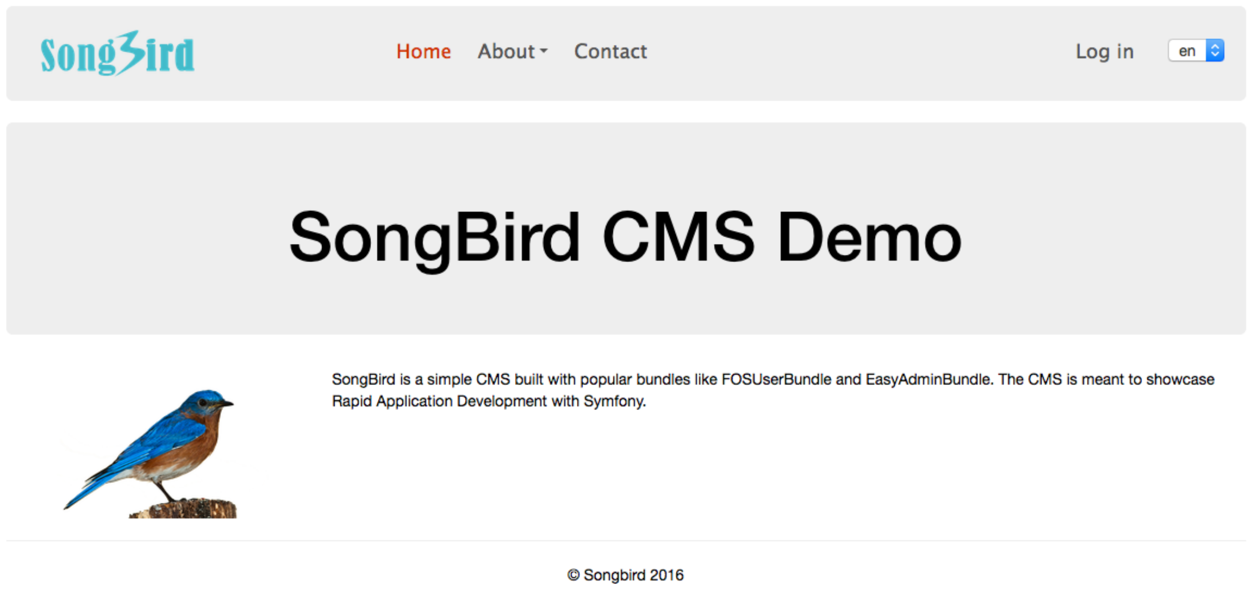
If you are already getting impatient and wants to see a demo of the completed project, you can checkout the final chapter. Make sure you have docker and docker-compose installed.

```
# If you are new to web development, you might be unfamiliar with some of the co\
mmands here. Don't worry as they will be explained as you follow through the cha\
pters sequentially.

-> git clone https://github.com/bernardpeh/songbird
-> cd songbird
-> git checkout chapter_final

# update SYMFONY_APP_PATH parameters in the .env file and leave the rest as defa\
ults
-> cp .env.dist .env

# we need to create new dir when mounting docker (if case if using nfs)
-> mkdir -p .data/db
-> mkdir -p logs/{symfony,nginx}
-> cd symfony

# See chapter 3 to improve mac performance before continuing (if you are interes\
ted)
-> docker-compose up --build -d

# To confirm all your containers are running
-> docker-compose ps

# add ip to your host file (assuming you are in unix env)
-> sudo echo "127.0.0.1 songbird.app" >> /etc/hosts

# create the uploads dir
mkdir -p web/uploads/{profiles,featured_images}

# install symfony libraries - ignore the db errors when running composer install
-> ./scripts/composer install

```

```
# install db and fixtures
-> ./scripts/resetapp

# install js libraries
-> bower install
-> npm install

# install assets
-> gulp
```

Docker could be slower for mac users. Chapter 3 provides a workaround.

Now go to http://songbird.app:8000 and you should see the homepage.



SongBird CMS Demo

SongBird is a simple CMS built with popular bundles like FOSUserBundle and EasyAdminBundle. The CMS is meant to showcase Rapid Application Development with Symfony.

© Songbird 2016

You can log into the backend by going to http://songbird.app:8000/admin using

```
user: admin
password: admin
```

To run full BDD test on the site

```
# Optional Step to check site is fully functional
-> ./scripts/runtest
```

References

- RAD[11]
- Agile Software Development[12]

[11] https://en.wikipedia.org/wiki/Rapid_application_development
[12] https://en.wikipedia.org/wiki/Agile_software_development

Chapter 1: Survival Skills

Without a doubt, the 2 biggest Symfony resources on the web at the moment are “The Book” and “The Cookbook”, both can be downloaded from Symfony Documentation Page[13]. Cudos to Fabien and the team behind the books, making Symfony one of the best documented frameworks out there. Having said that, the content in these 2 books are hard to digest and almost impossible to follow unless you have good foundation in Object Oriented Programming. There are a lot to go through. Even if you are have the skills, you will need enough determination to read them. Even if you finish reading them, you still need to have enough practical experience to digest the theory.

I hope there is really a simple formula to become a Symfony ninja overnight…

The Tools You Need

You will need to equip yourself before diving in. Ideally, you have

- A good computer. I recommend a modern day *Mac* not more than 4 years old with at least 100G of free space to setup development environment. Mac is fast becoming the new standard for coding. Linux is fine. If you insist in windows, make sure you have command line - cygwin[14] is a good option.
- Good foundation in programming. Experience with Object Oriented Programming and relational databases is recommended.
- Understand Dependency Injection (DI). Fabien wrote a good article about DI[15]. DI is the heartbeat of Symfony and most modern day framework.
- Good source control knowledge, especially Git and Git Flow.
- Basic HTML, CSS and Javascript knowledge.
- Basic Stylesheet Pre-processor language like LESS or SASS.
- Basic Linux command line knowledge.
- A good IDE. There are lots of them out there. Sublime Text[16] is OK but PHP Storm[17] is way better for serious Symfony development.

I hope the list doesn’t scare you to get started.

[13]http://symfony.com/doc/current/index.html
[14]https://www.cygwin.com/
[15]http://fabien.potencier.org/what-is-dependency-injection.html
[16]www.sublimetext.com
[17]https://www.jetbrains.com/phpstorm/

Using the Command Line

I suggest you to get comfortable with the command line. Many modern day frameworks use command line to automate tasks. In this book, I'll be using a lot of command line but I suggest you not to memorise them. Always type "app/console" to see the options and then narrow in from there.

For example, your app/console might look like this (doesn't matter if it doesn't at this point):

```
-> bin/console

...

Available commands:
  help                                  Displays help for a command
  list                                  Lists commands
 assetic
  assetic:dump                          Dumps all assets to the filesystem
  assetic:watch                         Dumps assets to the filesystem as their s\
ource files are modified
 assets
  assets:install                        Installs bundles web assets under a publi\
c web directory
 cache
  cache:clear                           Clears the cache
  cache:warmup                          Warms up an empty cache
 config
  config:debug                          Dumps the current configuration for an ex\
tension
  config:dump-reference                 Dumps the default configuration for an ex\
tension
 container
  container:debug                       Displays current services for an applicat\
ion
 debug
  debug:config                          Dumps the current configuration for an ex\
tension
  debug:container                       Displays current services for an applicat\
ion
  debug:event-dispatcher                Displays configured listeners for an appl\
ication
  debug:router                          Displays current routes for an application
  debug:swiftmailer                     Displays current mailers for an applicati\
on
  debug:translation                     Displays translation messages information
  debug:twig                            Shows a list of twig functions, filters, \
globals and tests
 doctrine
  doctrine:cache:clear-metadata         Clears all metadata cache for an entity m\
anager
  doctrine:cache:clear-query            Clears all query cache for an entity mana\
ger
  doctrine:cache:clear-result           Clears result cache for an entity manager
  doctrine:database:create              Creates the configured database
  doctrine:database:drop                Drops the configured database
  doctrine:ensure-production-settings  Verify that Doctrine is properly configur\
ed for a production environment.
  doctrine:generate:crud                Generates a CRUD based on a Doctrine enti\
ty
  doctrine:generate:entities            Generates entity classes and method stubs\
 from your mapping information
  doctrine:generate:entity              Generates a new Doctrine entity inside a \
bundle
  doctrine:generate:form                Generates a form type class based on a Do\
ctrine entity
  doctrine:mapping:convert              Convert mapping information between suppo\
rted formats.
  doctrine:mapping:import               Imports mapping information from an exist\
ing database
  doctrine:mapping:info
  doctrine:query:dql                    Executes arbitrary DQL directly from the \
command line.
  doctrine:query:sql                    Executes arbitrary SQL directly from the \
command line.
  doctrine:schema:create                Executes (or dumps) the SQL needed to gen\
erate the database schema
  doctrine:schema:drop                  Executes (or dumps) the SQL needed to dro\
p the current database schema
  doctrine:schema:update                Executes (or dumps) the SQL needed to upd\
ate the database schema to match the current mapping metadata.
  doctrine:schema:validate              Validate the mapping files.
 fos
  fos:user:activate                     Activate a user
  fos:user:change-password              Change the password of a user.
  fos:user:create                       Create a user.
  fos:user:deactivate                  Deactivate a user
  fos:user:demote                      Demote a user by removing a role
  fos:user:promote                     Promotes a user by adding a role
 generate
  generate:bundle                      Generates a bundle
  generate:controller                  Generates a controller
  generate:doctrine:crud               Generates a CRUD based on a Doctrine enti\
ty
  generate:doctrine:entities           Generates entity classes and method stubs\
 from your mapping information
  generate:doctrine:entity             Generates a new Doctrine entity inside a \
bundle
  generate:doctrine:form               Generates a form type class based on a Do\
ctrine entity
 init
  init:acl                             Mounts ACL tables in the database
 lint
  lint:twig                            Lints a template and outputs encountered \
errors
  lint:yaml                            Lints a file and outputs encountered erro\
rs
 orm
  orm:convert:mapping                  Convert mapping information between suppo\
rted formats.
 router
  router:debug                         Displays current routes for an application
  router:dump-apache                   [DEPRECATED] Dumps all routes as Apache r\
ewrite rules
  router:match                         Helps debug routes by simulating a path i\
nfo match
 security
  security:check                       Checks security issues in your project de\
pendencies
  security:encode-password             Encodes a password.
 server
  server:run                           Runs PHP built-in web server
  server:start                         Starts PHP built-in web server in the bac\
kground
  server:status                        Outputs the status of the built-in web se\
rver for the given address
  server:stop                          Stops PHP's built-in web server that was \
started with the server:start command
```

```
 swiftmailer
  swiftmailer:debug                        Displays current mailers for an applicati\
on
  swiftmailer:email:send                   Send simple email message
  swiftmailer:spool:send                   Sends emails from the spool
 translation
  translation:debug                        Displays translation messages information
  translation:update                       Updates the translation file
 twig
  twig:debug                               Shows a list of twig functions, filters, \
globals and tests
  twig:lint                                Lints a template and outputs encountered \
errors
 yaml
  yaml:lint                                Lints a file and outputs encountered erro\
rs
```

Wow, that is a lot but don't worry, you will get used to the important ones after finishing the book.

Selling Your Soul to the Demon

Many people use web frameworks[18] to create internet applications nowadays. A framework speeds up web development by giving you automation tools to create commonly used features like user management system, forms, pages, menus…etc. This means that you can create these features easily without knowing how they work. It is like buying a car without knowing how the car works. This is all good until if you want to customise the inner components or repair it. You could get someone to customise the car (hire a developer) or DIY.

If you are a developer, there is value in learning how to built a CMS with Symfony. While building the CMS, you learn how to configure and customise all the bundles to make them work together. As the builder, you will know where to start troubleshooting when things go wrong.

Let's get the ball rolling…

Summary

This is a short chapter. We discussed the basic skills required to learn a modern day framework like Symfony. You were mentally prepared and warned about the pros and cons of using a framework.

[18]http://symfony.com/why-use-a-framework

References

- git[19]
- gitflow[20]

[19]https://git-scm.com/

[20]https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow/

Chapter 2: What is SongBird

In a nutshell, SongBird is a bare bone CMS (Content Management System) consisting the following features:

- Admin Panel and Dashboard - A password protected administration area for administrators and users.
- User Management System - For administrators to manage users of the system.
- i18n Capability - Multi-lingual. No CMS is complete without this.
- Page Management System - For managing the front-end menu, slug and content of the site.
- User Logging Sytem - For logging user activities in the backend.
- Frontend - The portal where the public interacts with the site. No login required.

We will attempt to built the CMS using some popular modules available to cut down the development time. This is the best approach. However, that also means that we lose the fun of building some cool bundles ourselves. In view of that, we will attempt to build the Page management bundle and frontend ourselves.

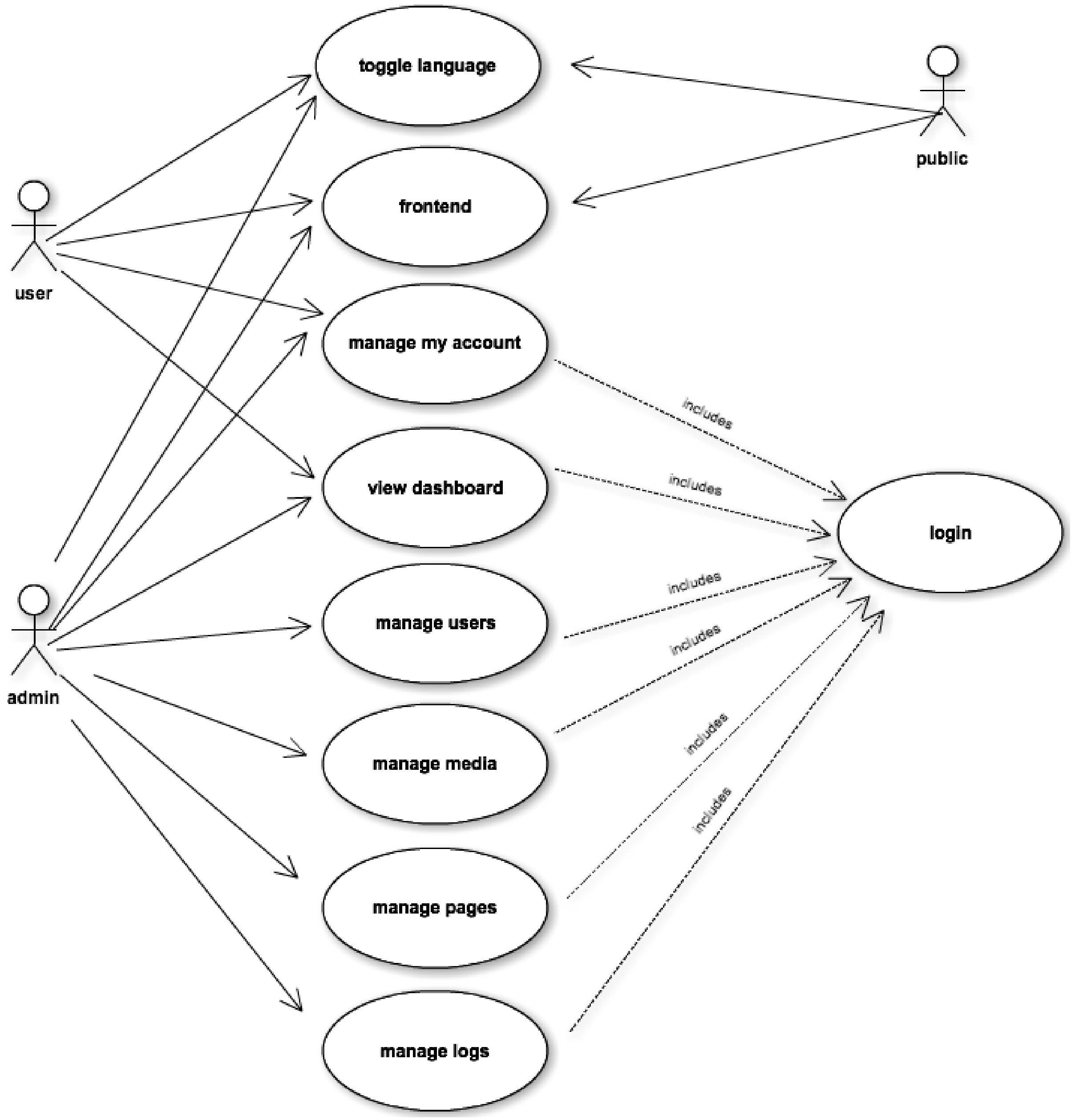
So What is the Plan?

In this chapter we are going to define the scope of the software. People spend weeks to write a proper functional specification for a software like this. Functional Specification defines the scope of the project, provides an estimate of the amount of man hours required and duration to complete the job, gives people an idea of what the software is, what it can or can't do. It is also important to use that as a reference when writing test cases as well.

Writing good functional specs is the most important part of the Software Development Life Cycle. In our case, we shall cut down the words and show only relevant information in developing SongBird.

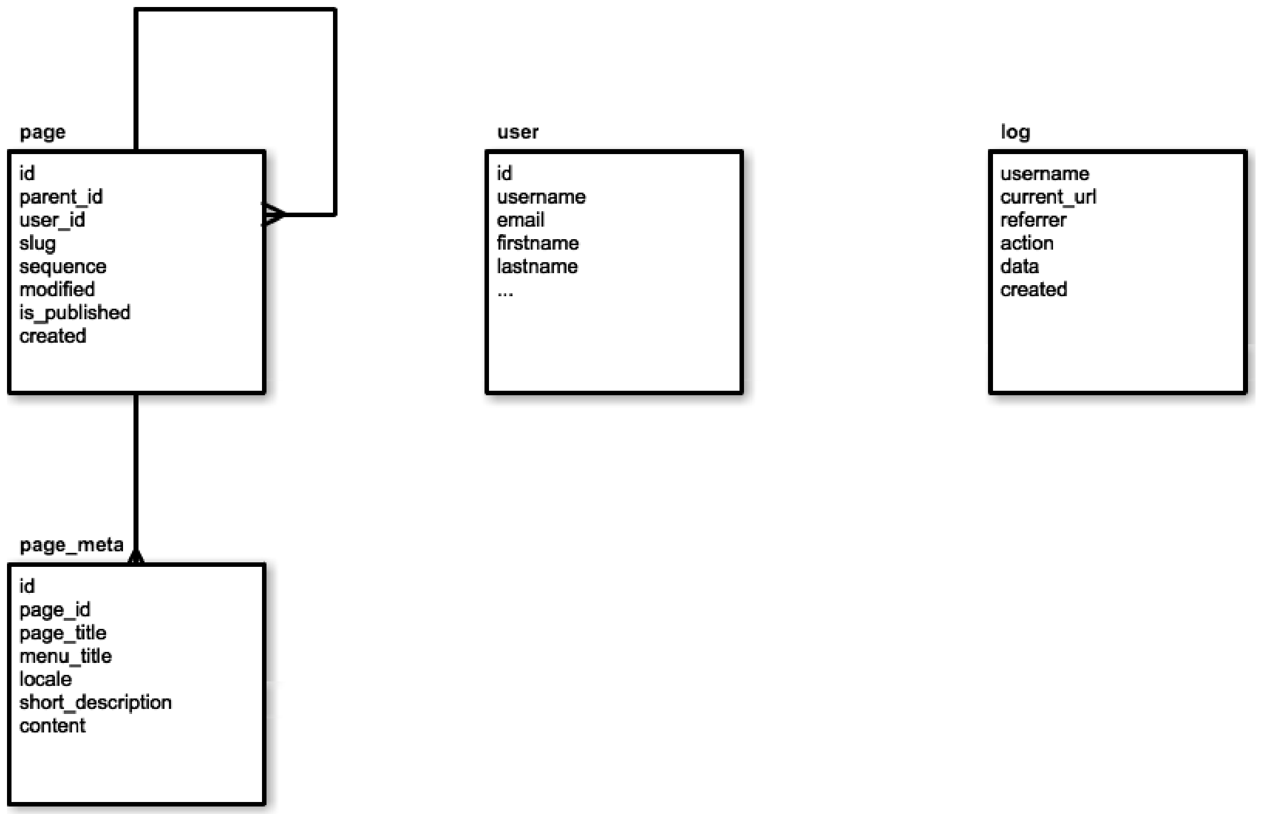
Use Case Diagram

This is a high level overview of the roles and features of SongBird.



Database Diagram

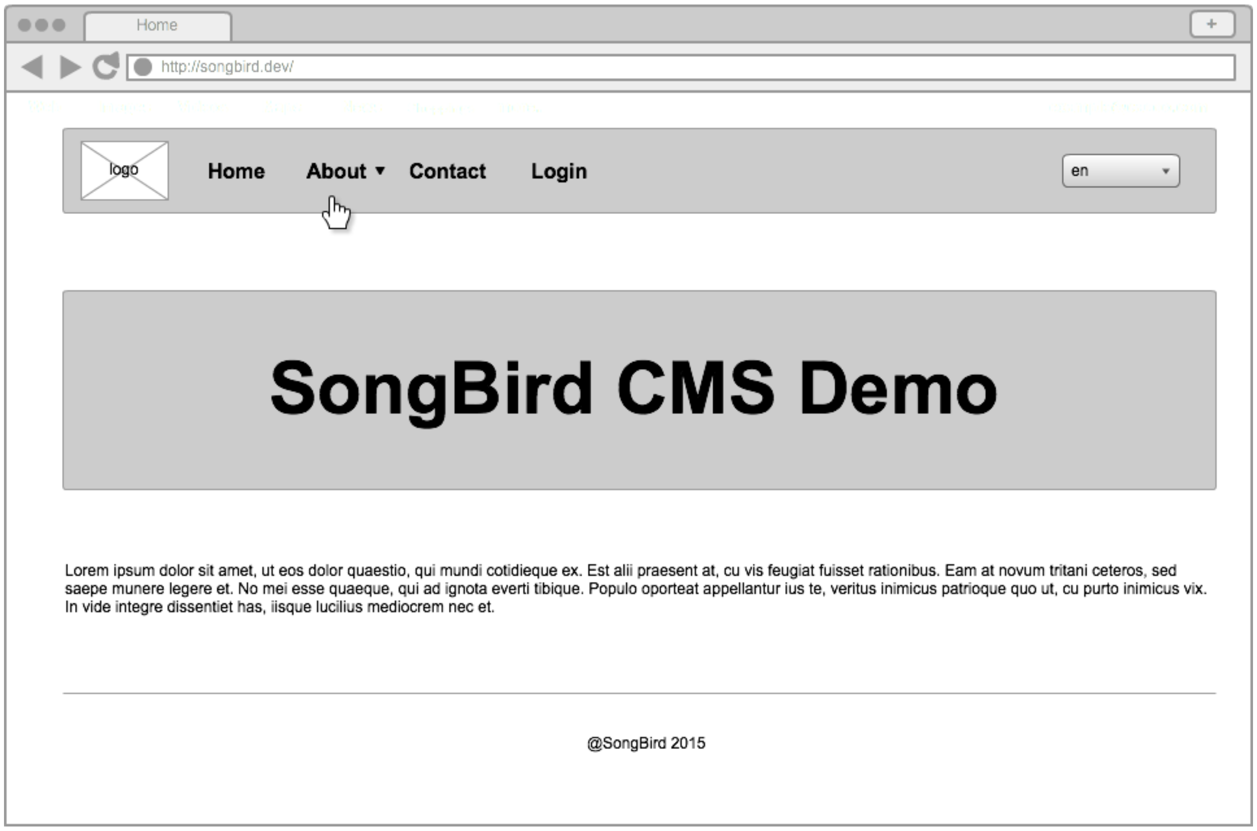
The entity relationships in a nutshell. In the real world, the relationships won't be that simple. You should see more one-to-many and many-to-many relationships.



User Journey

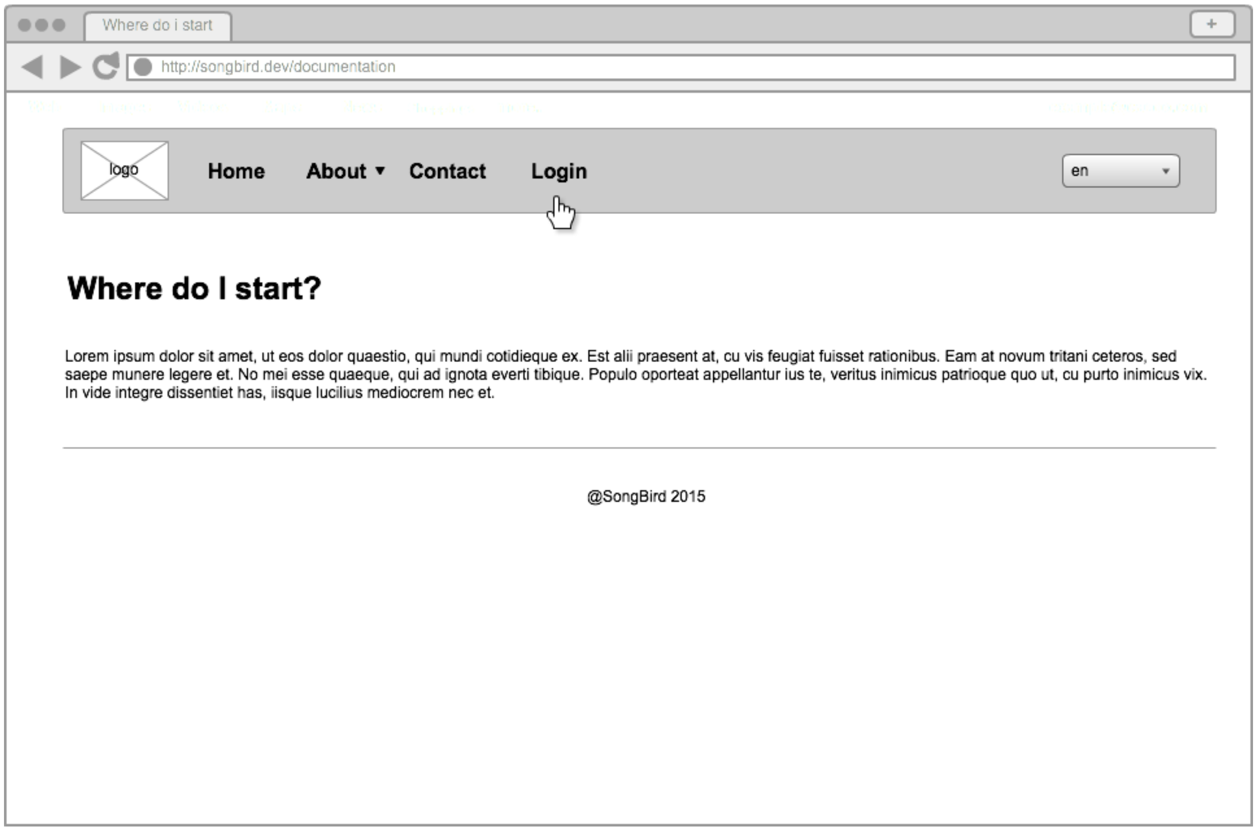
This is how I visualise a user would interact with the website. Hopefully, it gives you confidence of what we are about to build.

a) The frontend homepage:



Lorem ipsum dolor sit amet, ut eos dolor quaestio, qui mundi cotidieque ex. Est alii praesent at, cu vis feugiat fuisset rationibus. Eam at novum tritani ceteros, sed saepe munere legere et. No mei esse quaeque, qui ad ignota everti tibique. Populo oporteat appellantur ius te, veritus inimicus patrioque quo ut, cu purto inimicus vix. In vide integre dissentiet has, iisque lucilius mediocrem nec et.

b) The frontend subpage:

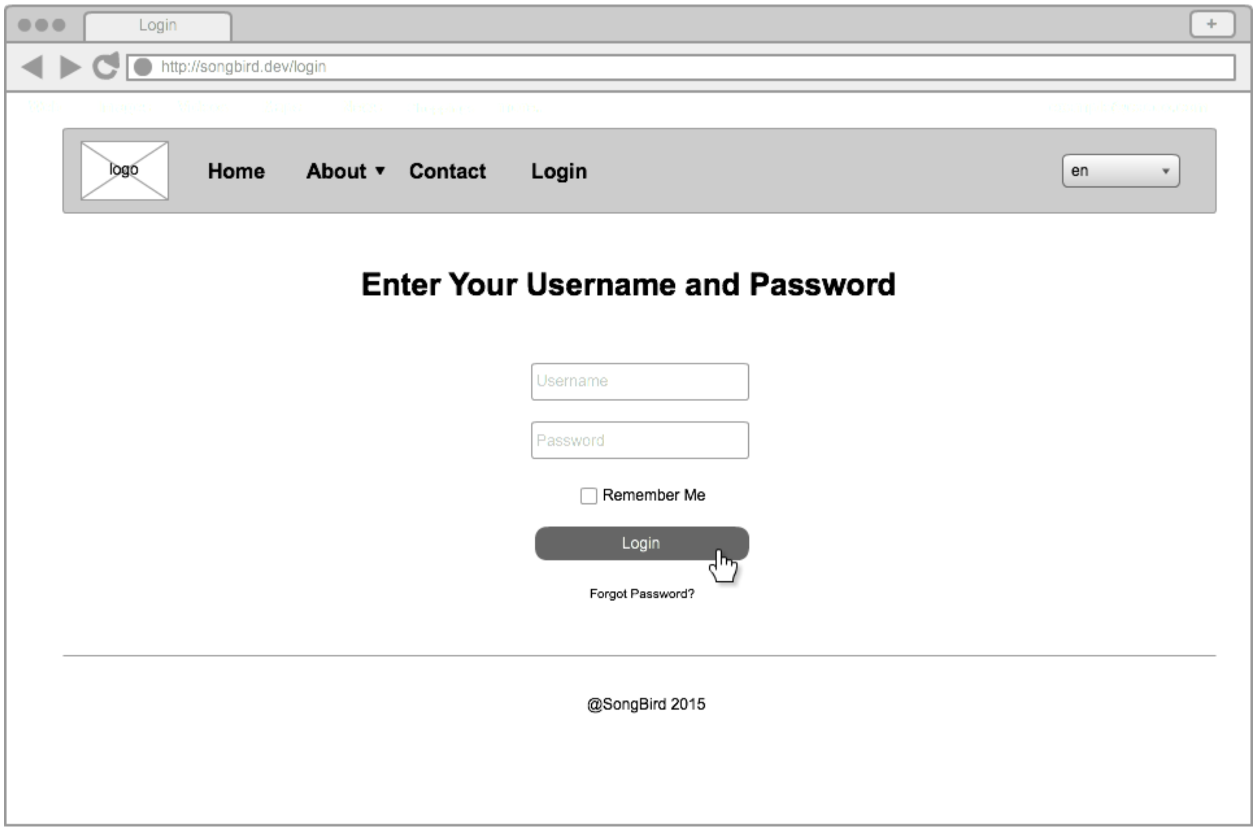


Where do I start?

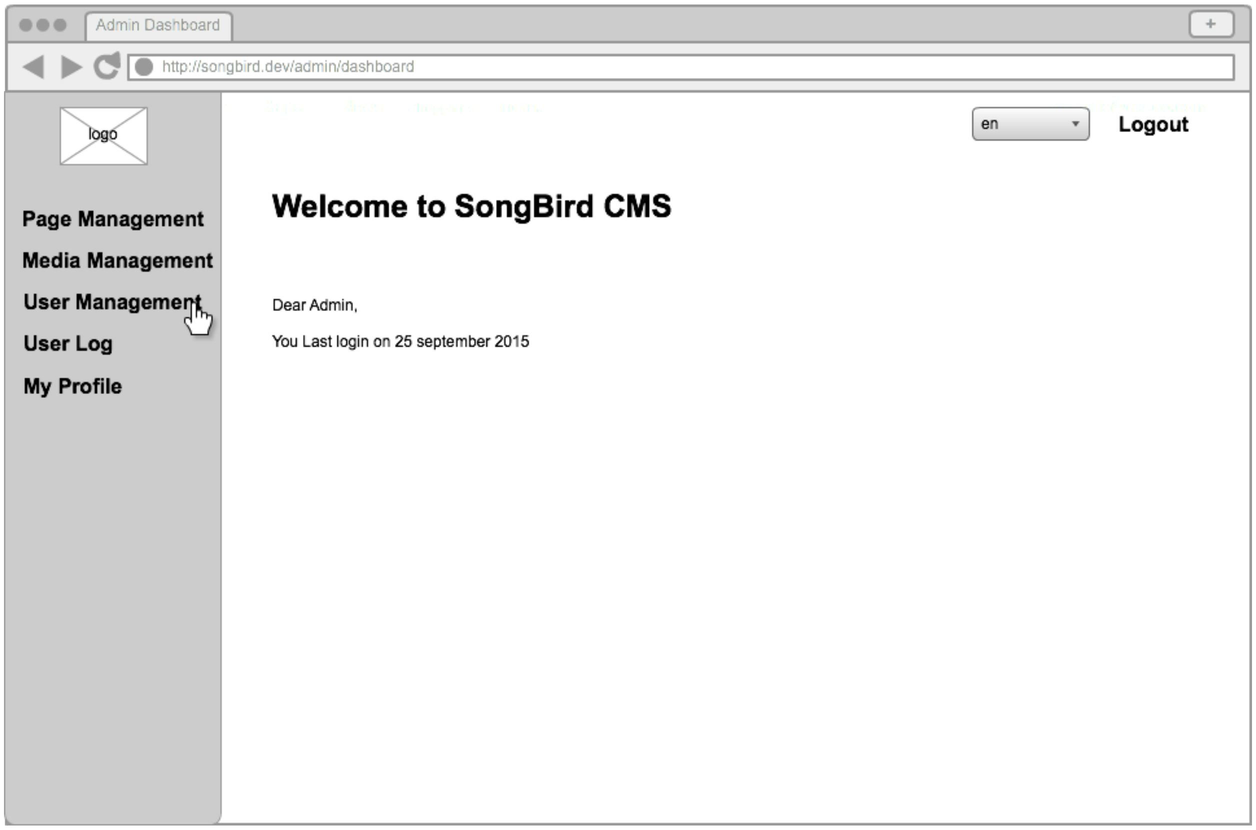
Lorem ipsum dolor sit amet, ut eos dolor quaestio, qui mundi cotidieque ex. Est alii praesent at, cu vis feugiat fuisset rationibus. Eam at novum tritani ceteros, sed saepe munere legere et. No mei esse quaeque, qui ad ignota everti tibique. Populo oporteat appellantur ius te, veritus inimicus patrioque quo ut, cu purto inimicus vix. In vide integre dissentiet has, iisque lucilius mediocrem nec et.

@SongBird 2015

c) The login page:



d) Backend dashboard:

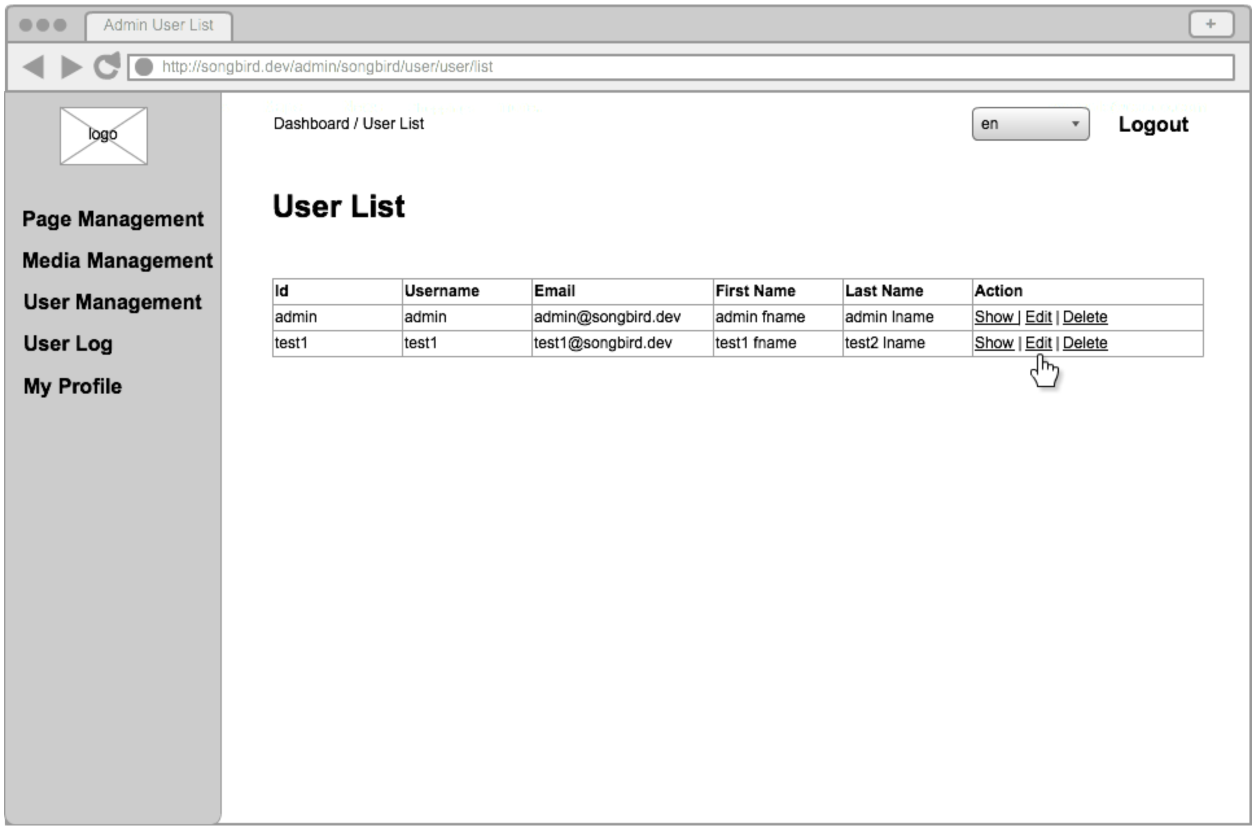


Welcome to SongBird CMS

Dear Admin,

You Last login on 25 september 2015

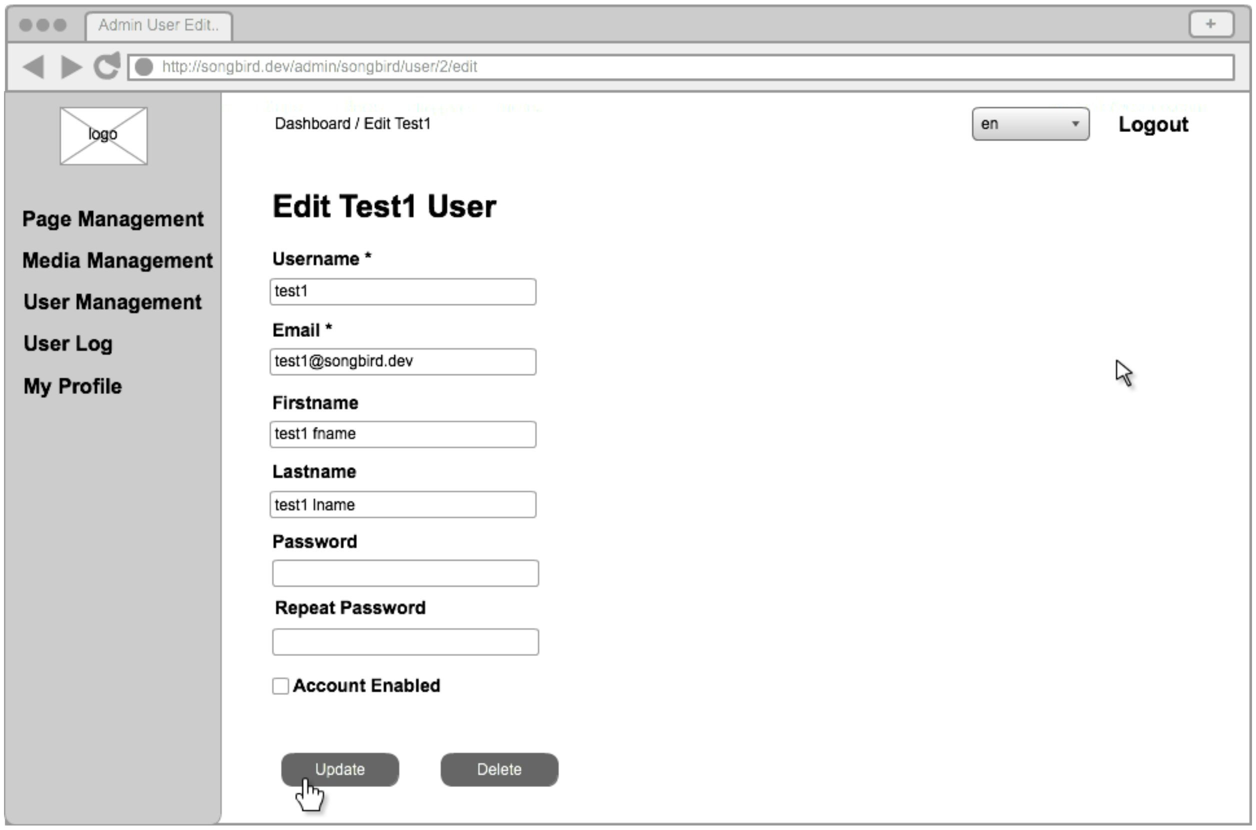
d) Backend listing page:



User List

Id	Username	Email	First Name	Last Name	Action
admin	admin	admin@songbird.dev	admin fname	admin lname	Show | Edit | Delete
test1	test1	test1@songbird.dev	test1 fname	test2 lname	Show | Edit | Delete

e) Backend record edit page:



Edit Test1 User

Username *
test1

Email *
test1@songbird.dev

Firstname
test1 fname

Lastname
test1 lname

Password

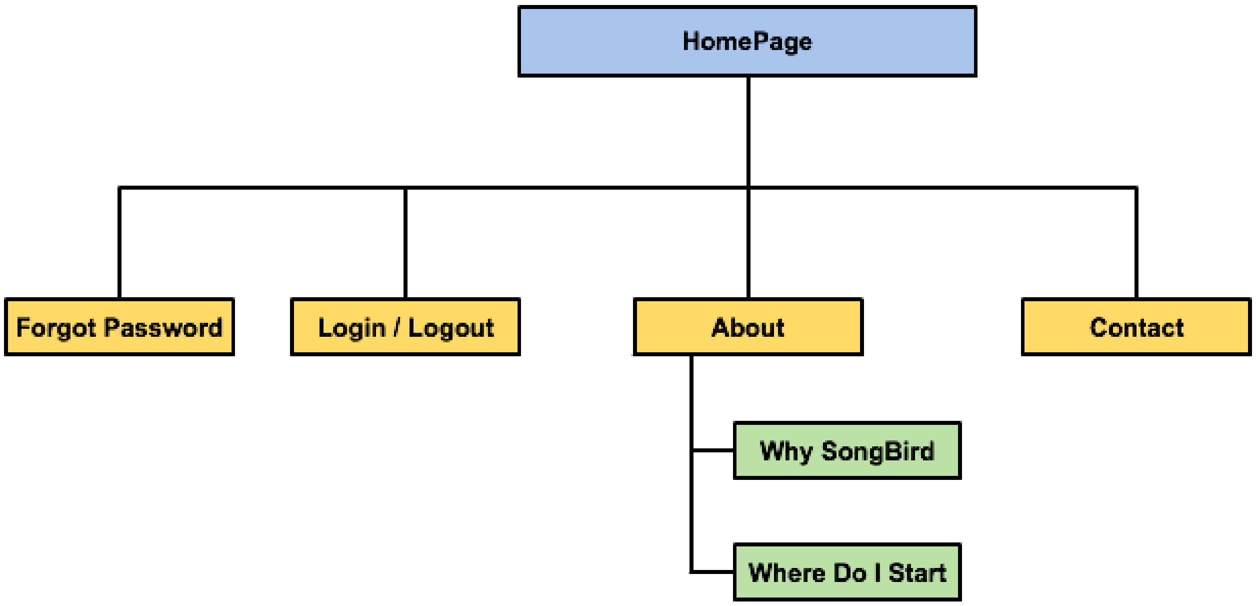
Repeat Password

Account Enabled

We haven't started coding but you already have a realistic view of the final product.

Sitemap

We are going to start with a few pages only, keeping the navigation simple.



User Stories

A user story defines the functionality that the user wants to have in plain english. We don't want to drill down to specifics at this stage. The specifics should be in the user scenarios. We make use of "As a", "I want/don't want to" and "So that" to help define good user stories.

As an example:

"As a developer, I want to create a simple CMS, so that I can use it as a vanilla CMS for more complex projects".

We will define the user stories for each chapter as we go along.

User Scenarios

User Scenarios break the user story down into further possible outcomes. I like to think of them as pseudocode. We make use of "Given", "When" and "Then" to define user scenarios. BDD tests are written based on these scenarios. Based on the example above, Possible scenarios are:

"Given the homepage, When I land on the homepage, Then I should see a big welcome text."

"Given the about us page, When I navigate to the about us page from the menu, Then I should see my name"

We will define the user scenarios based on the user stories for each chapter as we go along.

Summary

In this chapter, we tried to define what SongBird is and isn't. We defined the requirements and provided some use cases/diagrams to help define the end product. In real life, requirement docs could be a lot longer. Having well defined requirements is paramount in building robust software.

References

- User Story[21]
- Functional Specification[22]

[21]https://en.wikipedia.org/wiki/User_story
[22]https://en.wikipedia.org/wiki/Functional_specification

Chapter 3: Creating the Dev Environment

So that we speak the same language throughout the book, we need a dev (development) environment that it is consistent in everyone's host. We will use docker[23] for this purpose.

The idea is to do **actual coding in your host** (main operating system) and let docker other services like the web server, MYSQL … etc. Note that 99% of the time, you don't need to touch the docker instances except to make sure that they are all up and running.

Installation

- Fork songbird from github[24]
- Clone it

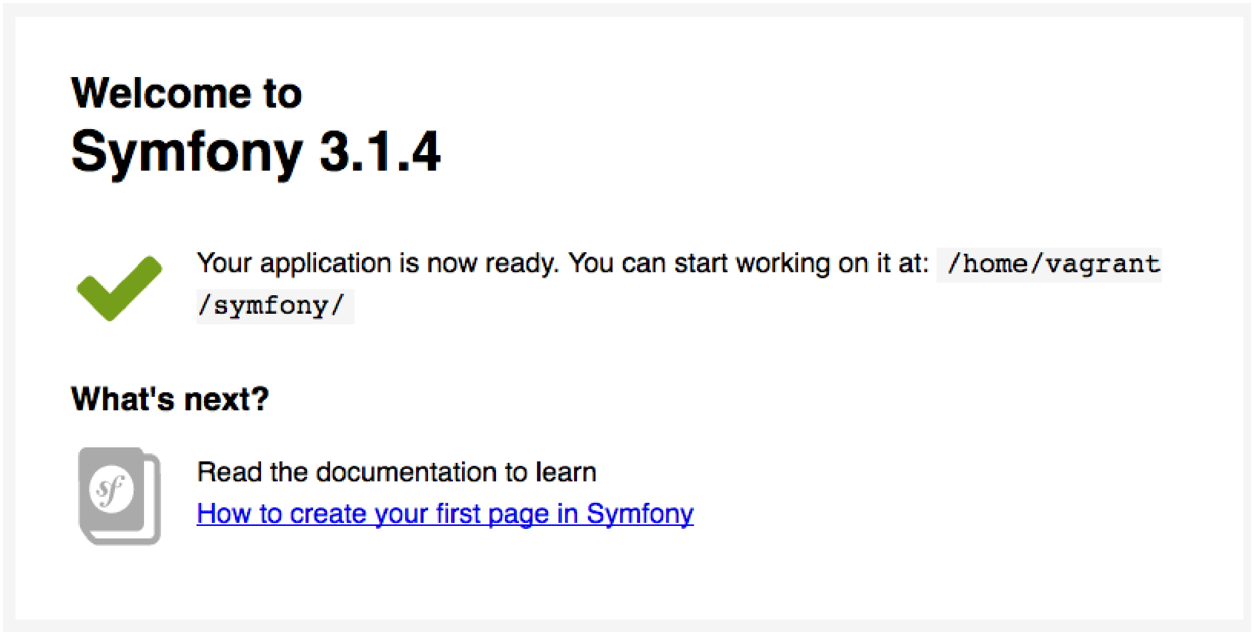
```
-> cd ~
-> git clone git@github.com:your_username/songbird.git
-> cd songbird
-> git checkout chapter_3
```

- Get the symfony[25] command line
- Install docker[26]
- Install Docker-compose[27]
- We can now fire up docker containers

[23]https://www.docker.com/
[24]https://github.com/bernardpeh/songbird
[25]http://symfony.com/doc/current/setup.html
[26]https://docs.docker.com/engine/installation/
[27]https://docs.docker.com/compose/install/

```
# update SYMFONY_APP_PATH parameters in the .env file and leave the rest as defa\
ults
-> cp .env.dist .env

# we need to create new dir when mounting docker (if case if using nfs)
-> mkdir -p .data/db
-> mkdir -p logs/{symfony,nginx}

# update parameters in the .env file if you want, then run
-> docker-compose up --build -d

# to confirm all the containers are fired up correctly
-> docker-compose ps
      Name                        Command              State               Ports    \

--------------------------------------------------------------------------------\
--------
songbird_db_1      docker-entrypoint.sh mysqld     Up      0.0.0.0:8006->3306/tc\
p
songbird_nginx_1   nginx                           Up      443/tcp, 0.0.0.0:8000\
->80/tcp
songbird_php_1     docker-php-entrypoint php-fpm   Up      0.0.0.0:9000->9000/tcp
...
```

- Add songbird.app to your host file. `# for unix systems -> sudo echo "127.0.0.1 songbird.app" >> /etc/hosts`
- We have mapped port 8000 to our nginx web server, open up browser and go to http://songbird.app:8000. If you see an installation successful page, you are on the right track.



Welcome to
Symfony 3.1.4

Your application is now ready. You can start working on it at: `/home/vagrant/symfony/`

What's next?

Read the documentation to learn
How to create your first page in Symfony

- Let us configure the dev url to allow connection from the parent host

```
# in symfony/web/app_dev.php

// comment off this ip restriction

// if (isset($_SERVER['HTTP_CLIENT_IP'])
//     || isset($_SERVER['HTTP_X_FORWARDED_FOR'])
//     || !(in_array(@$_SERVER['REMOTE_ADDR'], ['127.0.0.1', '::1']) || php_sapi\
_name() === 'cli-server')
// ) {
//     header('HTTP/1.0 403 Forbidden');
//     exit('You are not allowed to access this file. Check '.basename(__FILE__)\
.' for more information.');
// }
```

- Now try this url http://songbird.app:8000/app_dev.php and you should see the same successful page but with a little icon/toolbar at the bottom of the page. That's right, you are now in dev mode. Why the "app_dev.php"? That is like the default page for the dev environment, something unique to Symfony which we will always be using during development.
- To check that everything is working, let us look at the logs

```
-> tail -f logs/{nginx/*,symfony/*}

==> logs/nginx/error.log <==

==> logs/nginx/symfony_access.log <==
172.18.0.1 - - [19/Jan/2017:00:59:00 +0000] "GET / HTTP/1.1" 200 1947 "-" "Mozil\
la/5.0 (Macintosh; Intel Mac OS X 10_11_6) AppleWebKit/537.36 (KHTML, like Gecko\
) Chrome/55.0.2883.95 Safari/537.36"
172.18.0.1 - - [19/Jan/2017:00:59:00 +0000] "GET /favicon.ico HTTP/1.1" 200 6518\
 "http://songbird.app:8000/" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_6) Ap\
pleWebKit/537.36 (KHTML, like Gecko) Chrome/55.0.2883.95 Safari/537.36"
172.18.0.1 - - [19/Jan/2017:01:03:18 +0000] "GET /app_dev.php HTTP/1.1" 403 101 \
"-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_6) AppleWebKit/537.36 (KHTML, \
like Gecko) Chrome/55.0.2883.95 Safari/537.36"
172.18.0.1 - - [19/Jan/2017:01:04:20 +0000] "GET /app_dev.php HTTP/1.1" 403 101 \
"-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_6) AppleWebKit/537.36 (KHTML, \
like Gecko) Chrome/55.0.2883.95 Safari/537.36"
172.18.0.1 - - [19/Jan/2017:01:12:02 +0000] "GET /app_dev.php HTTP/1.1" 200 8132\
 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_6) AppleWebKit/537.36 (KHTML,\
 like Gecko) Chrome/55.0.2883.95 Safari/537.36"
172.18.0.1 - - [19/Jan/2017:01:12:14 +0000] "GET /app_dev.php/_wdt/2de5e5 HTTP/1\
.1" 200 6466 "http://songbird.app:8000/app_dev.php" "Mozilla/5.0 (Macintosh; Int\
el Mac OS X 10_11_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/55.0.2883.95 \
Safari/537.36"
172.18.0.1 - - [19/Jan/2017:01:17:49 +0000] "GET /app_dev.php HTTP/1.1" 200 8133\
 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_6) AppleWebKit/537.36 (KHTML,\
 like Gecko) Chrome/55.0.2883.95 Safari/537.36"
172.18.0.1 - - [19/Jan/2017:01:17:49 +0000] "GET /favicon.ico HTTP/1.1" 200 6518\
 "http://songbird.app:8000/app_dev.php" "Mozilla/5.0 (Macintosh; Intel Mac OS X \
10_11_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/55.0.2883.95 Safari/537.3\
6"
172.18.0.1 - - [19/Jan/2017:01:17:52 +0000] "GET /app_dev.php/_wdt/85ae68 HTTP/1\
.1" 200 6488 "http://songbird.app:8000/app_dev.php" "Mozilla/5.0 (Macintosh; Int\
el Mac OS X 10_11_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/55.0.2883.95 \
Safari/537.36"

==> logs/nginx/symfony_error.log <==

==> logs/symfony/dev.log <==
[2017-01-19 01:17:45] request.INFO: Matched route "homepage". {"route":"homepage\
","route_parameters":{"_controller":"AppBundle\\Controller\\DefaultController::i\
ndexAction","_route":"homepage"},"request_uri":"http://songbird.app:8000/app_dev\
.php/","method":"GET"} []
[2017-01-19 01:17:46] security.INFO: Populated the TokenStorage with an anonymou\
s Token. [] []
[2017-01-19 01:17:51] request.INFO: Matched route "_wdt". {"route":"_wdt","route\
_parameters":{"_controller":"web_profiler.controller.profiler:toolbarAction","to\
ken":"85ae68","_route":"_wdt"},"request_uri":"http://songbird.app:8000/app_dev.p\
hp/_wdt/85ae68","method":"GET"} []
```

Good, nginx and symfony is logging stuff.

Every time your machine restarts, remember to start docker, then run `docker-compose up -d` in the songbird folder to start the dev environment.

Finally, let us ignore .env in .gitignore

```
# .gitignore

/.vagrant/
.idea/
logs/
...
/.env
```

Mac Users (Optional)

Docker is such an amazing tool and I think it will only get more popular. However at the time of writing, mac operating system suffer performance issues due to osxfs. We can improve the disk access speed by using nfs instead. You can google about this and read about the technical details.

To mount via nfs, click on the docker icon on the top of your desktop -> Preferences -> File Sharing (remove all mounted dirs except /tmp) -> Restart docker.

We can then export the whole /Users dir

```
-> cd ~
-> git clone https://github.com/IFSight/d4m-nfs
-> cd d4m-nfs
-> echo "/Users:/Users" > etc/d4m-nfs-mounts.txt
-> sudo ./d4m-nfs.sh
# restart docker containers
-> cd ~/songbird
-> docker-compose down
-> docker-compose up -d
```

Summary

In this chapter, we setup the development environment using docker. We have installed Symfony and configured the host to access SongBird from the host machine.

Remember to commit all your changes before moving on.

Exercises (Optional)

- Try running Symfony's build-in webserver. What command would you use? What are the pros and cons of using the build-in webserver?
- Delete the symfony dir. Reinstall Symfony following the Symfony Installation[28] instructions.
- How many ways are there to install Symfony? What are the pros and cons of each?

References

- Docker Compose[29]

[28]https://symfony.com/doc/current/book/installation.html
[29]https://docs.docker.com/compose/

Chapter 4: The Testing Framework Part 1 (Optional)

This chapter talks about Codeception[30]. Feel free to skip it if you already have a testing framework in place.

No application is complete without going through a rigorous testing process. Software Testing is a big topic by itself.

Today, many developers know TDD[31] and BDD[32]. Test First Development ensures that your software is reliable but requires a lot of patience and extra work to implement it correctly. Think of it like a quality control process. The more checks you have, the less bugs your have. Of course, you can cost cut by not having checks and hope that your product is still bug free. This is quite unlikely especially if the software is complex.

Personally, I prefer to write user stories and scenarios first rather than spending time coding the tests. Think of them as pseudocode. Once we have the user stories and scenarios defined, we will jump in and code functionality A. When functionality A is completed, we will code the test cases and ensure they pass before moving on. We will repeat the cycle for functionality B before moving on to functionality C. The idea is to not break existing functionalities while adding on new functionalities.

Everyone's testing approach is different. You could implement your own approach.

There are many frameworks for acceptance testing. Behat[33] and Mink[34] are the industrial standard at the moment. In this book, we will be using Codeception[35] to write acceptance tests in most cases. We will also be writing some functional test in phpunit.

Installation

[30]http://codeception.com/
[31]https://en.wikipedia.org/wiki/Test-driven_development
[32]https://en.wikipedia.org/wiki/Behavior-driven_development
[33]http://docs.behat.org/
[34]http://mink.behat.org/
[35]http://codeception.com/

```
-> cd symfony
# only thing about running docker is that for anything relating to db connection,
# we need to execute commands in the docker instance. we will create a wrapper f\
or this in the future
-> docker-compose exec php composer require codeception/codeception --dev
```

The "–dev" means we only need this in dev mode. If everything is working, you will see composer adding the dependency in composer.json

```
# symfony/composer.json

# add the codeception line under require-dev
"require-dev": {
    ...
    "codeception/codeception": "^2.2"
},
```

Now we can initialise codeception

```
-> vendor/bin/codecept bootstrap
```

Let us configure the acceptance test.

```
# symfony/tests/acceptance.suite.yml
class_name: AcceptanceTester
modules:
    enabled:
        - WebDriver:
            url: 'http://songbird.app'
            host: 172.25.0.5
            port: 4444
            browser: phantomjs
            window_size: 1024x768
            capabilities:
                unexpectedAlertBehaviour: 'accept'
                webStorageEnabled: true
        - \Helper\Acceptance
```

Acceptance Testing is like Black Box Testing - We try to simulate real users interacting with our app. We ignore the inner workings of the code and only care if it works from the end user's point of view.

Here, we are using the headless browser - phantomjs[36] to connect to the webserver at 172.25.0.5 (see the docker-compose.yml file). Codeception by default comes with PhpBrowser which doesn't support javascript. Selenium[37] is slow but is the veteran when comes to acceptance testing. Feel free to switch to selenium if you encounter problems.

We can now generate the acceptance actions based on the updated acceptance suite:

```
-> vendor/bin/codecept build

# we will now get all the codecept libraries for free

Building Actor classes for suites: acceptance, functional, unit
 -> AcceptanceTesterActions.php generated successfully. 0 methods added
\AcceptanceTester includes modules: WebDriver, \Helper\Acceptance
 -> FunctionalTesterActions.php generated successfully. 0 methods added
\FunctionalTester includes modules: \Helper\Functional
 -> UnitTesterActions.php generated successfully. 0 methods added
\UnitTester includes modules: Asserts, \Helper\Unit
```

The First Test

We know that the default Symfony comes with the AppBundle example. Let us now test the bundle by creating a test suite for it.

```
-> vendor/bin/codecept generate:cest acceptance AppBundle
```

The auto generated Cest class should look like this:

```
# symfony/tests/acceptance/AppBundleCest.php

class AppBundleCest
{
    public function _before(AcceptanceTester $I)
    {
    }

    public function _after(AcceptanceTester $I)
    {
    }
```

[36] http://phantomjs.org
[37] http://www.seleniumhq.org/

```

    ...
}
```

Let us write our own test. All new Symfony installation homepage should have a successful message.

```
# symfony/tests/acceptance/AppBundleCest.php
...
# replaced tryToTest function with InstallationTest function
public function InstallationTest(AcceptanceTester $I)
{
    $I->wantTo('Check if Symfony is installed successfully.');
    $I->amOnPage('/');
    $I->see('Welcome to');
}
```

We have been running the codecept command from the host machine. That is fine but we should really be running the command in the php docker container. In the symfony dir, we need to softlink the .env file as we are going to run docker commands in that dir. If not, you will get a bunch of environment variables not found error.

```
# in the symfony dir
-> ln -s ../.env
```

Now run the test:

```
-> docker-compose exec php vendor/bin/codecept run acceptance AppBundleCest

Codeception PHP Testing Framework v2.2.8
Powered by PHPUnit 5.7.5 by Sebastian Bergmann and contributors.

Acceptance Tests (1) -----------------------------------------------------------\
-------------
Testing acceptance
 AppBundleCest: Check if symfony is installed successfully. (4.82s)
-------------------------------------------------------------------------------\
-------------


Time: 5.86 seconds, Memory: 13.50MB

OK (1 test, 1 assertion)
```

Some files such as images are binary. We need to tell git not to convert the line endings (google for it if interested)

```
# .gitattributes

...
# Denote all files that are truly binary and should not be modified.
*.png binary
*.jpg binary
```

Don't forget to commit your code before moving on to the next chapter.

```
-> git add symfony
-> git commit -m"added codeception and created basic test"
# update remote repo so you dont lose it
-> git push -u origin my_chapter4
```

Summary

In this chapter, we discussed the importance of testing and touched on TDD and BDD. In our context, we will be mainly writing BDD tests. We installed codeception and wrote a simple acceptance test to tests the default symfony home page.

Exercises (Optional)

- Try configure codeception to allow the running of different acceptance testing profiles. Can you test with PhpBrowser or selenium easily? Do you see any benefit of doing that? See advanced codeception[38] for help.

Resources

- TDD[39]
- BDD[40]
- Codeception documentation[41]

[38]http://codeception.com/docs/07-AdvancedUsage
[39]https://en.wikipedia.org/wiki/Test-driven_development
[40]https://en.wikipedia.org/wiki/Behavior-driven_development
[41]http://codeception.com/docs

Chapter 5: The Testing Framework Part 2 (Optional)

This chapter talks about Codeception[42]. Feel free to skip it if you already have a testing framework in place.

Since we are ready to build the application, let us remove the route for the default homepage and we are going to make sure that we have done that correctly.

Modifying DefaultController.php

Previously, we could access the route "/" because the route exists in DefaultController.php. Removing the @route annotation will remove the route. A simple trick to do that is to take out the @.

```
#  symfony/src/AppBundle/Controller/DefaultController.php

use Sensio\Bundle\FrameworkExtraBundle\Configuration\Route;
use Symfony\Bundle\FrameworkBundle\Controller\Controller;
use Symfony\Component\HttpFoundation\Request;

class DefaultController extends Controller
{
    /**
     * not using the homepage route for now
     *
     * Route("/", name="homepage")
     */
    public function indexAction(Request $request)
    {
        // replace this example code with whatever you need
        return $this->render('default/index.html.twig', array(
            'base_dir' => realpath($this->container->getParameter('kernel.root_d\
ir').'/..'),
        ));
    }
}
```

Now, refresh http://songbird.app:8000/app_dev.php and you should see a 404 error.

[42]http://codeception.com/

```
No route found for "GET /"
```

This is correct because the url is no longer configured. How can you be sure? Let us check it out from the command line

```
# in symfony
-> docker-compose exec php bin/console debug:router

[router] Current routes
 Name                        Method Scheme Host Path
 _wdt                        ANY    ANY    ANY  /_wdt/{token}
 _profiler_home              ANY    ANY    ANY  /_profiler/
 _profiler_search            ANY    ANY    ANY  /_profiler/search
 _profiler_search_bar        ANY    ANY    ANY  /_profiler/search_bar
 _profiler_purge             ANY    ANY    ANY  /_profiler/purge
 _profiler_info              ANY    ANY    ANY  /_profiler/info/{about}
 _profiler_phpinfo           ANY    ANY    ANY  /_profiler/phpinfo
 _profiler_search_results ANY    ANY    ANY  /_profiler/{token}/search/results
 _profiler                   ANY    ANY    ANY  /_profiler/{token}
 _profiler_router            ANY    ANY    ANY  /_profiler/{token}/router
 _profiler_exception         ANY    ANY    ANY  /_profiler/{token}/exception
 _profiler_exception_css     ANY    ANY    ANY  /_profiler/{token}/exception.css
 _configurator_home          ANY    ANY    ANY  /_configurator/
 _configurator_step          ANY    ANY    ANY  /_configurator/step/{index}
 _configurator_final         ANY    ANY    ANY  /_configurator/final
 _twig_error_test            ANY    ANY    ANY  /_error/{code}.{_format}
```

Looks like there is no trace of the / path. This is all good but to do a proper job, we have to make sure that this logic is remembered in the future. We need to record this logic in a test.

Hang on, did you realise how ugly the command line is?

`docker-compose exec php bin/console debug:router`

What we are doing here is that we are trying to access `bin/console` within the php docker instance. If you have php 7 installed in your host, you could run `bin/console` straight away and achieve the same results.

We will create a much simple wrapper around the docker commands in a minute.

Making sure the / route is removed

To make sure that / route is correctly removed and not accidentally added again in the future, let us add a test for it.

```
# symfony/tests/acceptance/AppBundleCest.php
...
# replace installationTest with removalTest

   /**
     * check that homepage is not active
     *
     * @param AcceptanceTester $I
     */
    public function RemovalTest(AcceptanceTester $I)
    {
        $I->wantTo('Check if / is not active.');
        $I->amOnPage('/');
        $I->see('404 Not Found');
    }
```

and run the test again,

```
# clear prod cache because test is running in prod env
-> docker-compose exec php bin/console cache:clear --env=prod

# remember to start phantomjs server before running this command
-> docker-compose exec php vendor/bin/codecept run acceptance
...
Time: 10.47 seconds, Memory: 11.50MB

OK (1 test, 1 assertion)
```

Creating custom bash script to run acceptance test

We have to remember to clear the cache every time we run the test so that we don't test on the cached version. Let us automate this by creating a script in the scripts dir called "runtest" and make it executable.

```
# in symfony
-> touch scripts/runtest
-> chmod u+x scripts/runtest
```

In the runtest script,

```
# symfony/scripts/runtest

#!/bin/bash

docker-compose exec php bin/console cache:clear --no-warmup
docker-compose exec php vendor/bin/codecept run acceptance
```

Now test your automation by running

```
-> ./scripts/runtest
...
OK (1 test, 1 assertion)
```

We are almost done. Remember to commit all your changes before moving on to the next chapter.

Summary

In this chapter, we have removed the default / route and updated our test criteria. We have also created a few bash scripts to automate the task of running codecept test. We will add more to these scripts in the future.

References

- TDD[43]
- BDD[44]
- PhantomJS[45]

[43]https://en.wikipedia.org/wiki/Test-driven_development
[44]https://en.wikipedia.org/wiki/Behavior-driven_development
[45]http://phantomjs.org/download.html

Chapter 6: The User Management System Part 1

User Management System is the core part of any CMS. We will create this feature using the popular FOSUserBundle[46].

Pre-setup

Make sure we are in the right branch. Let us branch off from the previous chapter.

```
-> git checkout -b my_chapter6
```

Installing the FOSUserBundle

Add the bundle in composer.json

```
# in symfony
-> docker-compose exec php composer require friendsofsymfony/user-bundle ~2.0@dev
```

Now in AppKernel, we need to register the bundles

```
# app/AppKernel.php
...
public function registerBundles()
{
    $bundles = array(
        ...
        new AppBundle\AppBundle(),
        // init my fosuser
        new FOS\UserBundle\FOSUserBundle(),
        new AppBundle\User()
    );
}
```

AppBundleUser() will look for the User class in User.php (under the AppBundle namespace). We want the User class to inherit all properties of FOSUserBundle. Let us create User.php

[46]https://github.com/FriendsOfSymfony/FOSUserBundle

```
# src/AppBundle/User.php

namespace AppBundle;

use Symfony\Component\HttpKernel\Bundle\Bundle;

class User extends Bundle
{
    // use a child bundle
    public function getParent()
    {
        return 'FOSUserBundle';
    }
}
```

Next we need to configure FOSUserBundle. Don't worry if certain directives don't make sense. It will as you progress further. Note that yaml files cannot contain tabs.

```
# app/config/config.yml
...
# turn on translator
translator:      { fallbacks: ["%locale%"] }
...

framework:
    ...
    session:
        # http://symfony.com/doc/current/reference/configuration/framework.html#\
handler-id
        # we have to use the system session storage because the default doesn't \
work with vagrant.
        handler_id:  ~
        # save_path:   "%kernel.root_dir%/../var/sessions/%kernel.environment%"

# fosuser config
fos_user:
    db_driver: orm
    firewall_name: main
    user_class: AppBundle\Entity\User
    from_email:
        address: admin@songbird.app
        sender_name: Songbird
```

and setup the security and firewall, your file should look like this

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

  # http://symfony.com/doc/current/book/security.html#where-do-users-come-from-u\
ser-providers
  providers:
      fos_userbundle:
          id: fos_user.user_provider.username

  role_hierarchy:
          ROLE_ADMIN:       ROLE_USER
          ROLE_SUPER_ADMIN: ROLE_ADMIN

  firewalls:
      # disables authentication for assets and the profiler, adapt it according \
to your needs
      dev:
          pattern: ^/(_(profiler|wdt)|css|images|js)/
          security: false

      main:
          pattern: ^/
          form_login:
              provider: fos_userbundle
              csrf_token_generator: security.csrf.token_manager
          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 }
```

DB credentials

The db credentials are in app/config/parameters.yml. They are usually variables based on your environment. Since we are using docker, we can hard code them.

Have a look at the file if you are interested.

```
# symfony/app/config/parameters.yml

# your db host is the container in your docker environment
# run "docker network inspect songbird_mynet" to see the ip of the mysql instanc\
e.

parameters:
    database_host: 172.25.0.2
    database_port: 3306
    database_name: songbird
    database_user: root
    database_password: root
    mailer_transport: smtp
    mailer_host: 172.25.0.6:1025
    mailer_user: null
    mailer_password: null
    secret: ThisTokenIsNotSoSecretChangeIt
```

Creating the User Entity

Have a quick read if you are unfamiliar with doctrine and entity[47]. We will be using doctrine very often in this book.

Symfony allows us to automate lots of things using command line, including the creation of entities. We will create the user entity with 2 custom fields called firstname and lastname.

```
-> docker-compose exec php bin/console generate:doctrine:entity

# You will be prompted a series of questions.

The Entity shortcut name: AppBundle:User

Configuration format (yml, xml, php, or annotation) [annotation]: annotation

New field name (press <return> to stop adding fields): firstname
Field type [string]:
Field length [255]:
Is nullable [false]: true
```

[47] http://symfony.com/doc/current/book/doctrine.html

```
Unique [false]:

New field name (press <return> to stop adding fields): lastname
Field type [string]:
Field length [255]:
Is nullable [false]: true
Unique [false]:
```

You realised we have to use "docker-compose exec php" to run commands in the php container. Its ugly and we will automate that in a minute. Once you are familiar with the command line, you should be able to generate the entity and other files without prompts. We will be doing that in the future chapters.

FOSUserBundle Groups[48] are useful when you want to group users together. For the sake of simplicity, we won't be using this feature. However, you should be able to add this feature in easily once you are comfortable with the Symfony workflow.

Now, the entity class is generated under src/AppBundle/Entity folder. We need to extend the fosuserbundle and make the id protected because of inheritance. If you open up the file, you will see that the code has been created for you already but we still need to make some changes in order for the entity inheritance to work. Refer to comments in the code.

```
namespace AppBundle\Entity;

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

/**
 * User extending fos user
 *
 * @ORM\Table(name="user")
 * @ORM\Entity(repositoryClass="AppBundle\Repository\UserRepository")
 */
class User extends BaseUser
{
    /**
     * Needs to be protected because of inheritance
     *
     * @var int
     *
     * @ORM\Column(name="id", type="integer")
```

[48]https://github.com/FriendsOfSymfony/FOSUserBundle/blob/master/Resources/doc/groups.md

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

    /**
     * @var string
     *
     * @ORM\Column(name="firstname", type="string", length=255, nullable=true)
     */
    private $firstname;

    /**
     * @var string
     *
     * @ORM\Column(name="lastname", type="string", length=255, nullable=true)
     */
    private $lastname;

    /**
     * User constructor.
     */
    public function __construct()
    {
        parent::__construct();
    }

    /**
     * Get id
     *
     * @return int
     */
    public function getId()
    {
        return $this->id;
    }

    /**
     * Set firstname
     *
     * @param string $firstname
     *
     * @return User
     */
    public function setFirstname($firstname)
    {
        $this->firstname = $firstname;

        return $this;
    }

    /**
     * Get firstname
     *
     * @return string
     */
    public function getFirstname()
    {
        return $this->firstname;
    }

    /**
     * Set lastname
     *
     * @param string $lastname
     *
     * @return User
     */
    public function setLastname($lastname)
    {
        $this->lastname = $lastname;

        return $this;
    }

    /**
     * Get lastname
     *
     * @return string
     */
    public function getLastname()
    {
        return $this->lastname;
    }
```

```
}
```

You will noticed all the getters and setters have already been generated for you as well. Cool!

Now, we need to configure the routes. The default routes provided by FOSUser is a good start.

```
# app/config/routing.yml
...
# FOS user bundle default routing
fos_user_security:
    resource: "@FOSUserBundle/Resources/config/routing/security.xml"

fos_user_profile:
    resource: "@FOSUserBundle/Resources/config/routing/profile.xml"
    prefix: /profile

fos_user_resetting:
    resource: "@FOSUserBundle/Resources/config/routing/resetting.xml"
    prefix: /resetting

fos_user_change_password:
    resource: "@FOSUserBundle/Resources/config/routing/change_password.xml"
    prefix: /profile
```

To check that the new routes have been installed correctly,

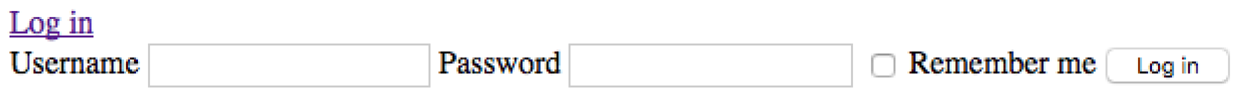
```
# in symfony
-> bin/console debug:router | grep fos

 fos_user_security_login           GET|POST ANY    ANY  /login
 fos_user_security_check           POST     ANY    ANY  /login_check
 fos_user_security_logout          GET      ANY    ANY  /logout
 fos_user_profile_show             GET      ANY    ANY  /profile/
 fos_user_profile_edit             GET|POST ANY    ANY  /profile/edit
 fos_user_resetting_request        GET      ANY    ANY  /resetting/request
 fos_user_resetting_send_email     POST     ANY    ANY  /resetting/send-email
 fos_user_resetting_check_email    GET      ANY    ANY  /resetting/check-email
 fos_user_resetting_reset          GET|POST ANY    ANY  /resetting/reset/{token}
 fos_user_change_password          GET|POST ANY    ANY  /profile/change-password
```

or we can use the router:match command to match the exact url and get more details

```
# in symfony
-> bin/console router:match /profile/
Route "fos_user_profile_show" matches

[router] Route "fos_user_profile_show"
Name         fos_user_profile_show
Path         /profile/
Path Regex   #^/profile/$#s
Host         ANY
Host Regex
Scheme       ANY
Method       GET
Class        Symfony\Component\Routing\Route
Defaults     _controller: FOSUserBundle:Profile:show
Requirements NO CUSTOM
Options      compiler_class: Symfony\Component\Routing\RouteCompiler
```

See how much work has done for you by inheriting the FOSUserBundle… This step allows you to use many default FOSUserBundle functionalities like password reset and user profile update without writing a single line of code! Now, let us test one of the routes by going to

```
http://songbird.app:8000/app_dev.php/login
```



You should see a simple login page.

To verify that the schema is correct, let us generate it:

```
# in symfony
-> docker-compose exec php bin/console doctrine:schema:create

Creating database schema...
Database schema created successfully!
```

Let us check that the schema has indeed been created correctly.

```
-> docker-compose exec db mysql -uroot -proot songbird -e "describe user"
+-----------------------+--------------+------+-----+---------+----------------+
| Field                 | Type         | Null | Key | Default | Extra          |
+-----------------------+--------------+------+-----+---------+----------------+
| id                    | int(11)      | NO   | PRI | NULL    | auto_increment |
| username              | varchar(180) | NO   |     | NULL    |                |
| username_canonical    | varchar(180) | NO   | UNI | NULL    |                |
| email                 | varchar(180) | NO   |     | NULL    |                |
| email_canonical       | varchar(180) | NO   | UNI | NULL    |                |
| enabled               | tinyint(1)   | NO   |     | NULL    |                |
| salt                  | varchar(255) | YES  |     | NULL    |                |
| password              | varchar(255) | NO   |     | NULL    |                |
| last_login            | datetime     | YES  |     | NULL    |                |
| confirmation_token    | varchar(180) | YES  | UNI | NULL    |                |
| password_requested_at | datetime     | YES  |     | NULL    |                |
| roles                 | longtext     | NO   |     | NULL    |                |
| firstname             | varchar(255) | YES  |     | NULL    |                |
| lastname              | varchar(255) | YES  |     | NULL    |                |
+-----------------------+--------------+------+-----+---------+----------------+
```

Looks like we got the right fields. Let us now create a console wrapper to make our life easier.

Wrapper Scripts

We now need a very simple wrapper to run the console commands. Let us create a console wrapper.

```
# in symfony/scripts/console

#!/bin/bash
docker-compose exec php bin/console $@
```

once the console script is created, it needs to be executable.

```
# in symfony
chmod u+x scripts/console
```

Let us try some commands

```
# you should not see an error
./scripts/console debug:router
```

Let us do the same for the composer command

```
# in symfony/scripts/composer

#!/bin/bash
docker-compose exec php composer "$@"
```

and

```
# in symfony
chmod u+x scripts/composer
```

Finally, we will create another for the mysql command

```
# in symfony/scripts/mysql

#!/bin/bash

MYSQL_DATABASE=`grep MYSQL_DATABASE .env | cut -d= -f 2`
MYSQL_ROOT_PASSWORD=`grep MYSQL_ROOT_PASSWORD .env | cut -d= -f 2`

docker-compose exec db mysql -uroot -p$MYSQL_ROOT_PASSWORD $MYSQL_DATABASE -e "$\
@"
```

now we allow executable bit to this script.

```
# in symfony
chmod u+x scripts/mysql
```

We can now use some wrapper scripts to access the php container easily. We are gearing up. Ready for more?

Summary

In this chapter, we have installed FOSUserBundle and extended it in AppBundle. We have verified that the installation was correct by looking at the default login page and database schema. We also created some helper scripts to help accessing the docker instance a bit easier.

Remember to commit all your changes before moving on.

Exercises (Optional)

- Try installing the UserBundle outside of Appbundle. Are there any pros and cons of doing that as compared to putting all the bundles in AppBundle?

References

- FOSUserBundle Doc[49]
- FOSUserBundle Installation[50]
- Routing[51]

[49]https://github.com/FriendsOfSymfony/FOSUserBundle/blob/master/Resources/doc/index.md
[50]https://symfony.com/doc/master/bundles/FOSUserBundle/index.html
[51]http://symfony.com/doc/current/book/routing.html

Chapter 7: The User Management System Part 2

We have installed the FOSUserBundle but it looks like there are still big chunks of functionalities missing. How do we (C)reate, (R)ead, (U)pdate and (D)elete a user or group for example?

You see the word "CRUD" appearing so many times because it is part of RAD. All frameworks today come with auto CRUD generation.

Automated User CRUD Generation

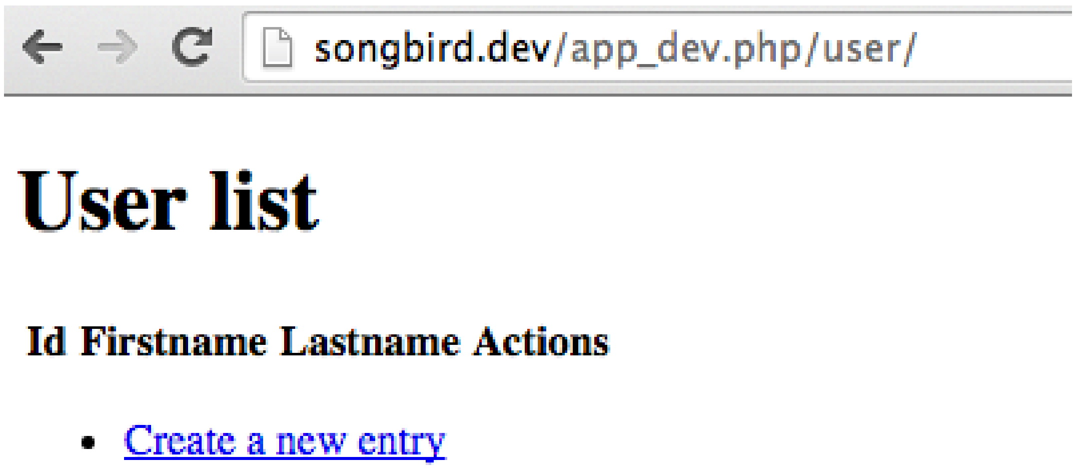
We will generate CRUD for the UserBundle.

```
-> ./scripts/console doctrine:generate:crud

The Entity shortcut name: AppBundle:User

By default, the generator creates two actions: list and show.
You can also ask it to generate "write" actions: new, update, and delete.

Do you want to generate the "write" actions [no]? yes

Determine the format to use for the generated CRUD.

Configuration format (yml, xml, php, or annotation) [annotation]: annotation

Determine the routes prefix (all the routes will be "mounted" under this
prefix: /prefix/, /prefix/new, ...).

Routes prefix [/user]:


  Summary before generation


You are going to generate a CRUD controller for "AppBundle:User"
using the "annotation" format.
```

```

Do you confirm generation [yes]?


  CRUD generation


Generating the CRUD code: OK
```

Now go to

```
http://songbird.app:8000/app_dev.php/user/
```



User list

Id Firstname Lastname Actions

- Create a new entry

We haven't added any data yet. The database should be empty as per the previous chapter.

Let us add some data. Click on "Create a new entry" or go to

```
http://songbird.app:8000/app_dev.php/user/new
```

and enter a dummy firstname and lastname, then click create.

You should see a "Integrity constraint violation: 1048 Column 'username' cannot be null" error. Why?

I am going to skip through all technicalities for now and tell you where the answer is. Look at

```
# vendor/friendsofsymfony/user-bundle/Resources/config/validation.xml
...
<property name="username">
    <constraint name="NotBlank">
        <option name="message">fos_user.username.blank</option>
        <option name="groups">
            <value>Registration</value>
            <value>Profile</value>
        </option>
        ...
```

It is possible to create a new user from command line, the code is at:

```
# vendor/friendsofsymfony/user-bundle/Command/CreateUserCommand.php
...
class CreateUserCommand extends ContainerAwareCommand
{
    /**
     * @see Command
     */
    protected function configure()
    {
        $this
            ->setName('fos:user:create')
            ->setDescription('Create a user.')
            ->setDefinition(array(
                new InputArgument('username', InputArgument::REQUIRED, 'The user\
name'),
                new InputArgument('email', InputArgument::REQUIRED, 'The email'),
                new InputArgument('password', InputArgument::REQUIRED, 'The pass\
word'),
```

Did you remember that the “fos:user:create” command is available under the `scripts/console` command? You can infer from these lines that username, email and password are compulsory. How do we add these extra fields in the user form?

Adding Fields to the User Form

The extra FOSUserBundle fields were not automatically added when we created the CRUD using the command line. The automated CRUD creation process cannot pick up inheritance yet (I hope one day it will). We have to create the fields manually.

```
# src/AppBundle/Form/UserType.php
namespace AppBundle\Form;

use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\OptionsResolver\OptionsResolver;
use Symfony\Component\Form\Extension\Core\Type\RepeatedType;
use Symfony\Component\Form\Extension\Core\Type\PasswordType;

class UserType extends AbstractType
{
    /**
     * @param FormBuilderInterface $builder
     * @param array $options
     */
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder
            ->add('username')
            ->add('email')
            ->add('firstname')
            ->add('lastname')
            ->add('password', RepeatedType::class, array(
                'type' => PasswordType::class,
                'invalid_message' => 'The password fields must match.',
                'required' => true,
                'first_options'  => array('label' => 'Password'),
                'second_options' => array('label' => 'Repeat Password'),
            ))
        ;
    }
    ...
```

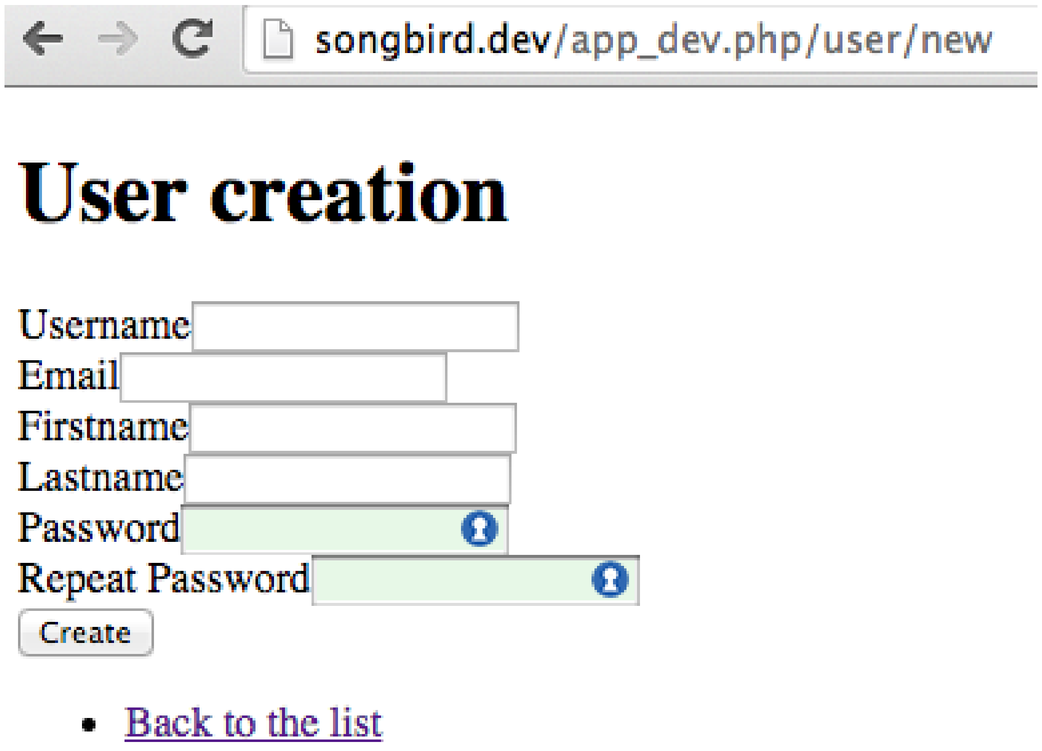
Refresh the browser and if changes are not showing up, we need to delete the cache.

```
-> ./scripts/console cache:clear
```

This `cache:clear` command is equivalent to "rm -rf var/cache/dev". It is a useful alternative to clear:cache. If no environment is set, the environment is set to develop. To delete prod cache,

```
-> ./scripts/console cache:clear -e prod
```

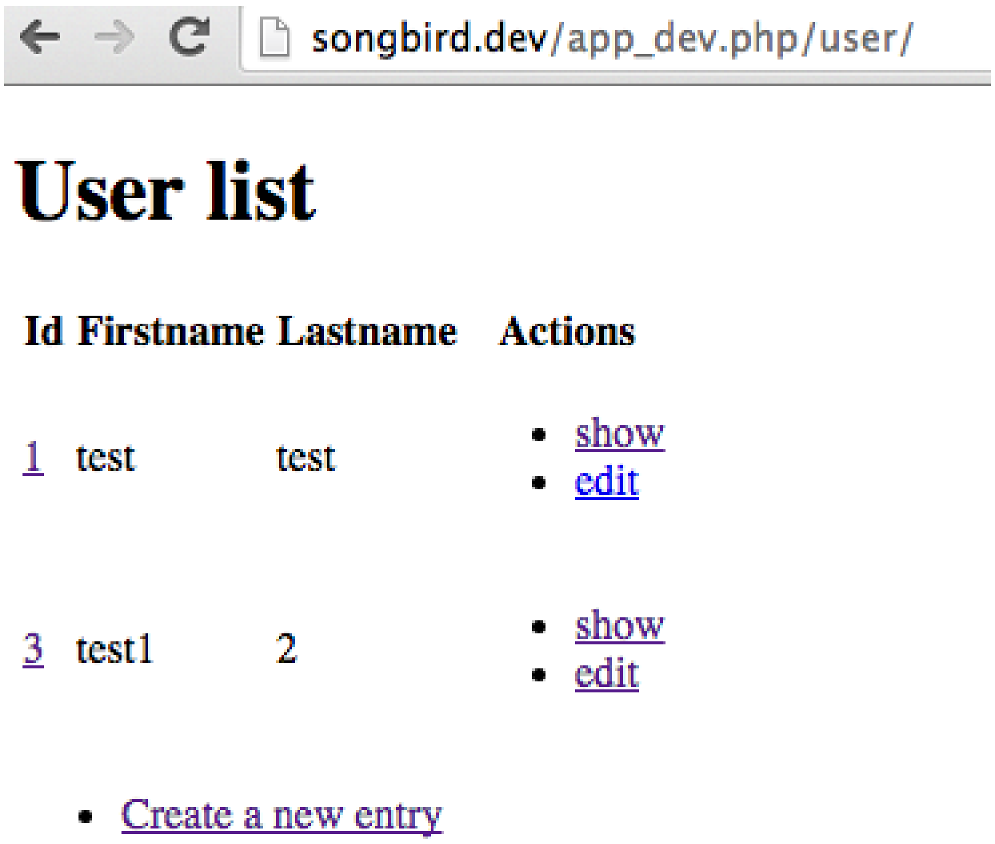
Let us create 2 test users, say "test" and "test1"


songbird.dev/app_dev.php/user/new

User creation

Username
Email
Firstname
Lastname
Password
Repeat Password

We can now list them by going to /user


songbird.dev/app_dev.php/user/

User list

Id	Firstname	Lastname	Actions
1	test	test	• show • edit
3	test1	2	• show • edit

• Create a new entry

Now verify that the new data is inserted into the user table by running some sql

```
-> ./scripts/mysql "select id,username,password from user"

+----+----------+----------+
| id | username | password |
+----+----------+----------+
|  1 | test     | test     |
|  2 | test1    | test1    |
+----+----------+----------+
```

Wow, why was the password exposed? shouldn't the password be encrypted automatically?

No, because the CRUD that we have created previously didn't know that the password was supposed to be encrypted before inserting into the db. Fortunately, FOSUserBundle has a service container that can help us with this. The word ***service*** is important in Symfony. Don't worry about it for now as we will cover this in the following chapters.

For the sake of curiousity, let us see all the FOSUserBundle service containers.

```
-> ./scripts/console debug:container | grep fos

 fos_user.change_password.form.factory                              FOSUserBundl\
eFormFactoryFormFactory
 fos_user.change_password.form.type                                 FOSUserBundl\
eFormTypeChangePasswordFormType
 fos_user.group.form.factory                                        FOSUserBundl\
eFormFactoryFormFactory
 fos_user.group.form.type                                           FOSUserBundl\
eFormTypeGroupFormType
 fos_user.group_manager                                             FOSUserBundl\
eDoctrineGroupManager
 fos_user.listener.authentication                                   FOSUserBundl\
eEventListenerAuthenticationListener
 fos_user.listener.flash                                            FOSUserBundl\
eEventListenerFlashListener
 fos_user.listener.resetting                                        FOSUserBundl\
eEventListenerResettingListener
 fos_user.mailer                                                    FOSUserBundl\
eMailerMailer
 fos_user.profile.form.factory                                      FOSUserBundl\
eFormFactoryFormFactory
 fos_user.profile.form.type                                         FOSUserBundl\
eFormTypeProfileFormType
 fos_user.registration.form.factory                                 FOSUserBundl\
eFormFactoryFormFactory
 fos_user.registration.form.type                                    FOSUserBundl\
eFormTypeRegistrationFormType
 fos_user.resetting.form.factory                                    FOSUserBundl\
eFormFactoryFormFactory
 fos_user.resetting.form.type                                       FOSUserBundl\
eFormTypeResettingFormType
 fos_user.security.interactive_login_listener                       FOSUserBundl\
eEventListenerLastLoginListener
 fos_user.security.login_manager                                    FOSUserBundl\
eSecurityLoginManager
 fos_user.user_manager                                              FOSUserBundl\
eDoctrineUserManager
 fos_user.username_form_type                                        FOSUserBundl\
eFormTypeUsernameFormType
 fos_user.util.email_canonicalizer                                  FOSUserBundl\
eUtilCanonicalizer
```

```
 fos_user.util.token_generator                                      FOSUserBundl\
eUtilTokenGenerator
 fos_user.util.user_manipulator                                     FOSUserBundl\
eUtilUserManipulator
```

The logic for all user related actions is stored in FOSUserBundleDoctrineUserManager. The service for that class is fos_user.user_manager. Let us use the service in UserController.php

```
# src/AppBundle/Controller/UserController.php
...
    /**
     * Creates a new User entity.
     *
     * @Route("/new", name="user_new")
     * @Method({"GET", "POST"})
     */
    public function newAction(Request $request)
    {
        $user = new User();
        $form = $this->createForm('AppBundle\Form\UserType', $user);
        $form->handleRequest($request);

        if ($form->isSubmitted() && $form->isValid()) {
            // CHANGE HERE
            $userManager = $this->get('fos_user.user_manager');
            $user->setPlainPassword($user->getPassword());
            $userManager->updateUser($user);
            // $em = $this->getDoctrine()->getManager();
            // $em->persist($user);
            // $em->flush();

            return $this->redirectToRoute('user_show', array('id' => $user->getI\
d()));
        }

        return $this->render('user/new.html.twig', array(
            'user' => $user,
            'form' => $form->createView(),
        ));
    }
...
    /**
```

```
     * Displays a form to edit an existing User entity.
     *
     * @Route("/{id}/edit", name="user_edit")
     * @Method({"GET", "POST"})
     */
    public function editAction(Request $request, User $user)
    {
        $deleteForm = $this->createDeleteForm($user);
        $editForm = $this->createForm('AppBundle\Form\UserType', $user);
        $editForm->handleRequest($request);
        if ($editForm->isSubmitted() && $editForm->isValid()) {
            // CHANGE HERE
            $userManager = $this->get('fos_user.user_manager');
            // we get the values that user submitted
            $user->setPlainPassword($request->request->get('user')['password']['\
first']);
            $userManager->updateUser($user);
            // $em = $this->getDoctrine()->getManager();
            // $em->persist($user);
            // $em->flush();

            return $this->redirectToRoute('user_edit', array('id' => $user->getI\
d()));
        }

        return $this->render('user/edit.html.twig', array(
            'user' => $user,
            'edit_form' => $editForm->createView(),
            'delete_form' => $deleteForm->createView(),
        ));
    }
...
```

The persist and flush statement in doctrine is a standard way to prepare and save queries to db. We have commented it off because if you look at the updateUser function in FOSUserBundleDoctrineUserManager, this part was already done.

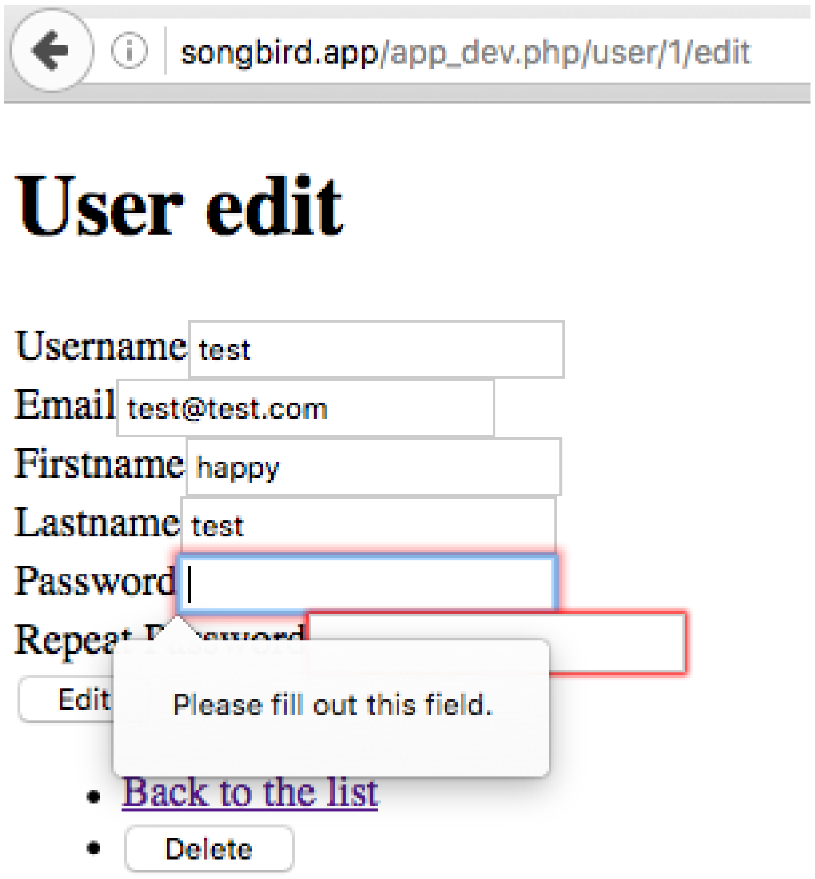
Let us try creating a new user called "test3" and view it again in mysql

```
-> ./scripts/mysql "select id,username,password from user"
+----+----------+--------------------------------------------------------------+
| id | username | password                                                     |
+----+----------+--------------------------------------------------------------+
|  1 | test     | test                                                         |
|  2 | test1    | test1                                                        |
|  4 | test3    | $2y$13$ovAu1e0C.eLof9KDsXVKP.bFFGxb82.mHf156i6PXI.XwjP9EwBr2 |
+----+----------+--------------------------------------------------------------+
```

The test3 user password is now encrypted. Update the password of another user and you will see that the encryption is working.

What's Up With Editing the User

Now, let's try editing the test user. We are going to change the first name for example,


songbird.app/app_dev.php/user/1/edit

User edit

Username test
Email test@test.com
Firstname happy
Lastname test
Password
Repeat Password

Please fill out this field.

The form is stopping us from editing because the password is a compulsory field. How do we fix that?

Let us pass a passwordRequired variable into the UserType class. If the variable is false, the password field will not be compulsory.

```
# src/AppBundle/Controller/UserController

...
    /**
     * Displays a form to edit an existing User entity.
     *
     * @Route("/{id}/edit", name="user_edit")
     * @Method({"GET", "POST"})
     */
    public function editAction(Request $request, User $user)
    {
        $deleteForm = $this->createDeleteForm($user);
        // ADD a new passwordRequired variable to the UserType Class
        $editForm = $this->createForm('AppBundle\Form\UserType', $user, array('p\
asswordRequired' => false));
        $editForm->handleRequest($request);
        ...
    }
...
```

and in UserType.php,

```
namespace AppBundle\Form;

use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\OptionsResolver\OptionsResolver;
use Symfony\Component\Form\Extension\Core\Type\RepeatedType;
use Symfony\Component\Form\Extension\Core\Type\PasswordType;

class UserType extends AbstractType
{
    /**
     * @param FormBuilderInterface $builder
     * @param array $options
     */
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder
```

```
            ->add('username')
            ->add('email')
            ->add('firstname')
            ->add('lastname')
            ->add('password', RepeatedType::class, array(
                'type' => PasswordType::class,
                'invalid_message' => 'The password fields must match.',
                // New passwordRequired variable
                'required' => $options['passwordRequired'],
                'first_options'  => array('label' => 'Password'),
                'second_options' => array('label' => 'Repeat Password'),
            ))
        ;
    }

    /**
     * @param OptionsResolver $resolver
     */
    public function configureOptions(OptionsResolver $resolver)
    {
        $resolver->setDefaults(array(
            'data_class' => 'AppBundle\Entity\User',
            // Add new variable
            'passwordRequired' => true,
        ));
    }
}
```

If the password field is null, it means that user doesn't want to update the password. We will need to override FOSUserBundle setPassword function.

```
# src/AppBundle/Entity/User.php
...
    /**
     * Override parent's method. Don't set passwd if its null.
     *
     * @param string $password
     * @return $this
     */
    public function setPassword($password)
    {
        if ($password) {
            $this->password = $password;
        }
        return $this;
    }
...
```

Updating Doctrine Fields Automatically

We like to have 2 more fields. We like to know when the user is being created and updated. How do we do that? HasLifeCycleCallBacks annotation is the magic.

```
# src/AppBundle/Entity/User.php
...
/**
 * User
 *
 * @ORM\Table()
 * @ORM\Entity(repositoryClass="AppBundle\Entity\UserRepository")
 * @ORM\HasLifecycleCallbacks()
 */
class User extends BaseUser
{
        ...
        /**
         * @ORM\Column(type="datetime")
         */
        private $modified;

        /**
         * @ORM\Column(type="datetime")
         */
        private $created;

        /**
         * @ORM\PrePersist
         */
        public function prePersist()
        {
            // update the modified time
            $this->setModified(new \DateTime());

            // for newly created entries
            if ($this->getCreated() == null) {
                $this->setCreated(new \DateTime('now'));
            }
        }

    /**
     * @ORM\PreUpdate
     */
    public function preUpdate()
    {
        // update the modified time
        $this->setModified(new \DateTime());
    }
...
```

The "@ORMHasLifecycleCallbacks()" tells doctrine to run callback functions (in this case, prePersist or preUpdate) before creating or updating an entry.

Let us auto-generate the setters and getters for the new $modified and $created variables.

```
-> ./scripts/console doctrine:generate:entities --no-backup AppBundle:User
```

The –no-backup option tells the command not to back up your original entity file.

Verify that the new getters and setters for $created and $modified have been added to src/AppBundle/Entity/User.php. The schema is now changed and we need to update it.

```
# run this and you will see what the sql is doing
-> ./scripts/console doctrine:schema:update --dump-sql

# once you are comfortable with that, force update it
-> ./scripts/console doctrine:schema:update --force
```

Try adding a new user and see if the created and modified time have been updated.

```
-> ./scripts/mysql "select id,password,modified,created from user"
+----+--------------------------------------------------------------+----------\
----------+---------------------+
| id | password                                                     | modified  \
          | created             |
+----+--------------------------------------------------------------+----------\
----------+---------------------+
|  1 | $2y$13$.yitE0Zj6kK9zJ6DYS7X0eYZMY7MfRR97OCwvTbjn59tfr4dPuOZG | 2017-01-27\
 06:39:08 | 2017-01-27 06:38:17 |
+----+--------------------------------------------------------------+----------\
----------+---------------------+
```

Deleting Users

No problem. This should work out of the box. Test it out in your browser to convince yourself.

Cleaning Up

let us clean up the Controller by deleting the DefaultController.php

```
-> git rm src/AppBundle/Controller/DefaultController.php
```

and we need to update our runtest script

```
# symfony/scripts/runtest

#!/bin/bash

scripts/console cache:clear --no-warmup
docker-compose exec php vendor/bin/codecept run acceptance
```

Run a quick test again and make sure that whatever you have done doesn't break anything. Still remember how to do it?

```
-> scripts/runtest
...

Time: 2.78 seconds, Memory: 13.50MB

OK (1 test, 1 assertion)
```

You will soon realised you need a consistent set of test data to make testing easier. That is why data fixtures are so important.

Summary

We have created User CRUD using command line, digged into the code and fixed up a few things. Even though things still doesn't work out of the box, we owed a lot to RAD to help us create a user management system in a short time. In reality, most CMS should allow you to configure user management system out of the box. It is still a good practice for us to go through it.

In addition to the basic CRUD, we have added 4 extra fields (firstname, lastname, created, modified). Unlike username, email and password fields, the firstname and lastname fields are not compulsory. On the edit page, the password field is also not compulsory.

Remember to commit all your changes before moving on.

Exercises (Optional)

- FOSUserBundle provides a functionality to manage users via command line. Try adding a user from the command line.
- Looking at AppBundleFormUserType, what happens if you change the password field to be called "plainPassword" instead? What changes would you make to the UserController.php class if that is the case?
- Can you think of another way to pass variable from the controller to the form?

References

- FOSUserBundle Doc[52]
- Repeated fields in forms[53]
- Service Container[54]
- Dependency Injection[55]

[52]https://github.com/FriendsOfSymfony/FOSUserBundle/blob/master/Resources/doc/index.md
[53]http://symfony.com/doc/current/reference/forms/types/repeated.html
[54]http://symfony.com/doc/current/book/service_container.html
[55]http://symfony.com/doc/current/components/dependency_injection/introduction.html

Chapter 8: Doctrine Fixtures and Migrations

As of now, we could create and manage users via the command line (scripts/console fos:user:xxx) or using the basic CRUD UI that we have created. What if we messed up the data or if we want to reset the data with certain values confidently? How can we do that efficiently? We need an automation mechanism to create consistent schema and dummy data.

Install DoctrineFixturesBundle

Install via composer

```
-> ./scripts/composer require doctrine/doctrine-fixtures-bundle ^2.3 --dev
```

Now that the data-fixtures-bundle is installed, we can update the kernel.

```
# app/AppKernel.php

...
if (in_array($this->getEnvironment(), array('dev', 'test'))) {
    ...
    $bundles[] = new Doctrine\Bundle\FixturesBundle\DoctrineFixturesBundle();
    ...
}
...
```

We register the bundle under the array('dev', 'test') environment because we don't need this bundle in the production environment.

To prove that the install is successful, we should have a new entry in the console

```
-> ./scipts/console | grep fixtures
  doctrine:fixtures:load              Load data fixtures to your database.
```

Create User Fixtures

Let's create the the data fixtures directory structure

```
-> mkdir -p src/AppBundle/DataFixtures/ORM
```

Now create the class. We are going to create 3 users. One super admin, 3 test users, ie test1, test2 and test3.

The username and password for the 3 users are as follows:

```
# in this format, username:password
admin:admin
test1:test1
test2:test2
test3:test3
```

Remember these 3 users credentials as we will be using them a lot throughout the whole book.

Now the actual fixtures class:

```
# src/AppBundle/DataFixtures/ORM/LoadUserData.php

namespace AppBundle\DataFixtures\ORM;

use Doctrine\Common\DataFixtures\AbstractFixture;
use Doctrine\Common\DataFixtures\OrderedFixtureInterface;
use Doctrine\Common\Persistence\ObjectManager;
use Symfony\Component\DependencyInjection\ContainerAwareInterface;
use Symfony\Component\DependencyInjection\ContainerInterface;

class LoadUserData extends AbstractFixture implements OrderedFixtureInterface, C\
ontainerAwareInterface
{

    /**
     * @var ContainerInterface
     */
    private $container;

    /**
     * {@inheritDoc}
     */
    public function setContainer(ContainerInterface $container = null)
    {
        $this->container = $container;
    }

    /**
     * {@inheritDoc}
     */
    public function load(ObjectManager $manager)
    {
        $userManager = $this->container->get('fos_user.user_manager');

        // add admin user
        $admin = $userManager->createUser();
        $admin->setUsername('admin');
        $admin->setEmail('admin@songbird.app');
        $admin->setPlainPassword('admin');
        $userManager->updatePassword($admin);
        $admin->setEnabled(1);
        $admin->setFirstname('Admin Firstname');
        $admin->setLastname('Admin Lastname');
        $admin->setRoles(array('ROLE_SUPER_ADMIN'));
        $userManager->updateUser($admin);

        // add test user 1
        $test1 = $userManager->createUser();
        $test1->setUsername('test1');
        $test1->setEmail('test1@songbird.app');
        $test1->setPlainPassword('test1');
        $userManager->updatePassword($test1);
        $test1->setEnabled(1);
        $test1->setFirstname('test1 Firstname');
        $test1->setLastname('test1 Lastname');
        $userManager->updateUser($test1);

        // add test user 2
        $test2 = $userManager->createUser();
        $test2->setUsername('test2');
        $test2->setEmail('test2@songbird.app');
        $test2->setPlainPassword('test2');
        $userManager->updatePassword($test2);
        $test2->setEnabled(1);
        $test2->setFirstname('test2 Firstname');
        $test2->setLastname('test2 Lastname');
        $userManager->updateUser($test2);

        // add test user 3
        $test3 = $userManager->createUser();
        $test3->setUsername('test3');
        $test3->setEmail('test3@songbird.app');
        $test3->setPlainPassword('test3');
        $userManager->updatePassword($test3);
        $test3->setEnabled(0);
        $test3->setFirstname('test3 Firstname');
        $test3->setLastname('test3 Lastname');
        $userManager->updateUser($test3);

        // use this reference in data fixtures elsewhere
        $this->addReference('admin_user', $admin);
    }

    /**
     * {@inheritDoc}
     */
    public function getOrder()
    {
        // load user data
        return 1;
    }
}
```

Now, let us insert the fixtures by running the command line

```
-> ./scripts/console doctrine:fixtures:load -n
```

The “-n” option simply answer yes when prompted for data purging. Try it without the “-n” option for yourself. Verify that the data is inserted by running a simple query

```
-> ./scripts/mysql "select * from user"
```

The nice thing about creating fixtures is that you learn a lot about the Entity when you insert the data. Take the FOSUserBundle for example, you need to know about the userManager in order to create encrypted passwords correctly. This knowledge is valuable when writing test cases.

This line shows the power of a modern day framework:

```
$userManager = $this->container->get('fos_user.user_manager');
```

We are trying to use the userManager class using the fos_user.user_manager service. Where is this class?

```
-> ./scripts/console debug:container | grep fos_user.user_manager
 fos_user.user_manager                                        FOS\UserBundle\Doctrine\UserMa\
nager

 # you can know a great deal about this service from
-> ./scripts/console debug:container fos_user.user_manager

Information for Service "fos_user.user_manager"
===============================================

 ------------------ -------------------------------------
  Option             Value
 ------------------ -------------------------------------
  Service ID         fos_user.user_manager
  Class              FOS\UserBundle\Doctrine\UserManager
  Tags               -
  Public             yes
  Synthetic          no
  Lazy               yes
  Shared             yes
  Abstract           no
  Autowired          no
  Autowiring Types   -
 ------------------ -------------------------------------
```

So basically, we are instantiating FOSUserBundleDoctrineUserManager without including the class and we do it as and when we want it. This is called Lazy Loading[56]. Traditionally, we would require the class and use the "new" keyword, something like this:

```
 require Class.php
 $myClass = new Class();
```

Remember we talked about services[57] in the previous chapter? We will see a lot more of these in the later chapters.

[56]https://en.wikipedia.org/wiki/Lazy_loading
[57]http://symfony.com/doc/current/book/service_container.html

Doctrine Migrations

Doctrine migrations allow us to migrate db changes easily. This is important especially when we want to make changes to production db. For example, if production db is a few versions behind, do we upgrade the db sequentially and safely?

Let us start the installation:

```
-> ./scripts/composer require doctrine/doctrine-migrations-bundle "^1.0"
```

and update AppKernel

```
# symfony/app/AppKernel.php

public function registerBundles()
{
    $bundles = array(
        //...
        new Doctrine\Bundle\MigrationsBundle\DoctrineMigrationsBundle(),
    );
}
```

We also need to configure it.

```
# app/config/config.yml
doctrine_migrations:
    dir_name: "%kernel.root_dir%/../src/AppBundle/DoctrineMigrations"
    namespace: AppBundle\DoctrineMigrations
    table_name: migration_versions
    name: AppBundle Migrations
```

If the installation is successful, you should see some new migrations commands added:

```
 ./scripts/console | grep migration
 doctrine:migrations:diff                Generate a migration by comparing your \
current database to your mapping information.
   doctrine:migrations:execute            Execute a single migration version up\
 or down manually.
   doctrine:migrations:generate           Generate a blank migration class.
   doctrine:migrations:latest             Outputs the latest version number
   doctrine:migrations:migrate            Execute a migration to a specified ve\
rsion or the latest available version.
   doctrine:migrations:status             View the status of a set of migration\
s.
   doctrine:migrations:version            Manually add and delete migration ver\
sions from the version table.
```

and

```
-> ./scripts/console doctrine:migrations:status

 == Configuration

    >> Name:                                               AppBundle Migrations
    >> Database Driver:                                    pdo_mysql
    >> Database Name:                                      songbird
    >> Configuration Source:                               manually configured
    >> Version Table Name:                                 migration_versions
    >> Version Column Name:                                version
    >> Migrations Namespace:                               AppBundle\DoctrineMig\
rations
    >> Migrations Directory:                               AppBundle/DoctrineMig\
rations
    >> Previous Version:                                   Already at first vers\
ion
    >> Current Version:                                    0
    >> Next Version:                                       Already at latest ver\
sion
    >> Latest Version:                                     0
    >> Executed Migrations:                                0
    >> Executed Unavailable Migrations:                    0
    >> Available Migrations:                               0
    >> New Migrations:                                     0
```

Its the first time we are using it, so we need to generate the initial migration class

```
-> ./scripts/console doctrine:migrations:generate
   Generated new migration class to "/var/www/symfony/app/../src/AppBundle/Doctr\
ineMigrations/Version20170128004532.php"
```

Look at "Version20170128004532.php" and you won't see much in there.

Create Script to Reset Schema and Fixtures

Every time we want to work cleanly, we want to be able to run a script to reset the database and insert dummy records. Let us create a script called resetapp that resides in scripts dir.

```
# scripts/resetapp

#!/bin/bash
rm -rf var/cache/*
# scripts/console cache:clear --no-warmup
scripts/console doctrine:database:drop --force
scripts/console doctrine:database:create
scripts/console doctrine:schema:create
scripts/console doctrine:fixtures:load -n
```

Make sure the script is executable

```
-> chmod u+x ./scripts/resetapp
```

Now we can run the test

```
-> ./scripts/resetapp
./scripts/resetapp

...
  > purging database
  > loading [1] AppBundle\DataFixtures\ORM\LoadUserData
```

Update runtest script

The runtest script can now call the scripts/resetapp script to have a cleaner start before running the test

```
# scripts/runtest

#!/bin/bash
scripts/resetapp
vendor/bin/codecept run acceptance $@
```

What is “$@”? In bash, it means putting in the command line options that was passed into the runtest script. We can now execute only the RemovalTest like so:

```
# remember to scripts/start_phantomjs in a new terminal if not done.

-> ./scripts/runtest AppBundleCest.php:RemovalTest

...
Acceptance Tests (1)
...
Time: 25.43 seconds, Memory: 11.50MB

OK (1 test, 1 assertion)
```

Summary

In this chapter we learned how to install the doctrine fixtures and migrations bundle. We also created a fixture class for our user bundle. We then upgraded our runtest script to reset the db and load the fixtures before running the test.

Remember to commit all your changes before moving on.

References

- DoctrineFixturesBundle[58]
- DoctrineMigrationsBundle[59]

[58]http://symfony.com/doc/current/bundles/DoctrineFixturesBundle/index.html
[59]http://symfony.com/doc/current/bundles/DoctrineMigrationsBundle/index.html

Chapter 9: The Admin Panel Part 1

We have used FOSUserBundle to create a User CRUD in the previous chapters. It's looking ugly at the moment but its functional. However, anyone can access the user management if they have the right url. We need an admin area where administrators can login and manage the users. All administrative activities should happen behind the admin url, something along the lines of /admin/users for example.

Again, we will try to simplify the process by reusing a 3rd party module that others have created. SonataAdmin[60] and EasyAdmin[61] are quite popular at the moment. SonataAdmin is more advanced but more complex to setup. In this book, we will be using EasyAdmin to build the admin panel.

It wouldn't be fun if we just use the ready made solution. In this and the next few chapters, we will attempt to build up the admin area bit by bit.

Install EasyAdminBundle

As usual, let us add the required bundles in the composer.json file

```
-> ./scripts/composer require javiereguiluz/easyadmin-bundle ^1.16.5
```

and remember to activate the required bundles in AppKernel.php

```
# app/AppKernel.php
...
    public function registerBundles()
    {
        $bundles = array(
            // ...
            new JavierEguiluz\Bundle\EasyAdminBundle\EasyAdminBundle(),
        );
    }
...
```

Create a new easyadmin config file

[60] https://github.com/sonata-project/SonataAdminBundle
[61] https://github.com/javiereguiluz/EasyAdminBundle

```
# app/config/easyadmin/user.yml

easy_admin:
    entities:
        User:
            class: AppBundle\Entity\User
```

The main config file then needs to load everything under the easyadmin folder

```
# app/config/config.yml
imports:
    - { resource: parameters.yml }
    - { resource: security.yml }
    - { resource: services.yml }
    - { resource: easyadmin/ }
...
```

and routing file

```
# app/config/routing.yml
...
easy_admin_bundle:
    resource: "@AppBundle/Controller/AdminController.php"
    type:     annotation
    prefix:   /admin
```

If everything goes well, there will be new routes added

```
-> ./scripts/console debug:router | grep admin
   easyadmin                           ANY        ANY      ANY    /admin/
   admin                               ANY        ANY      ANY    /admin/
```

We will install the default styles from the bundle

```
-> ./scripts/console assets:install --symlink
```

Say for now, we want ROLE_USER to access the admin dashboard.

```
# app/config/security.yml
...
    access_control:
        - { path: ^/login$, role: IS_AUTHENTICATED_ANONYMOUSLY }
        # We do not allow user registration
        # - { path: ^/register, role: IS_AUTHENTICATED_ANONYMOUSLY }
        - { path: ^/resetting, role: IS_AUTHENTICATED_ANONYMOUSLY }
        - { path: ^/admin/, role: ROLE_USER }
```

Let us create the new admin controller

```
# src/AppBundle/Controller/AdminController.php
namespace AppBundle\Controller;

use JavierEguiluz\Bundle\EasyAdminBundle\Controller\AdminController as BaseAdmin\
Controller;

class AdminController extends BaseAdminController
{
    public function createNewUserEntity()
    {
        return $this->get('fos_user.user_manager')->createUser();
    }

    public function prePersistUserEntity($user)
    {
        $this->get('fos_user.user_manager')->updateUser($user, false);
    }

    public function preUpdateUserEntity($user)
    {
        $this->get('fos_user.user_manager')->updateUser($user, false);
    }
}
```

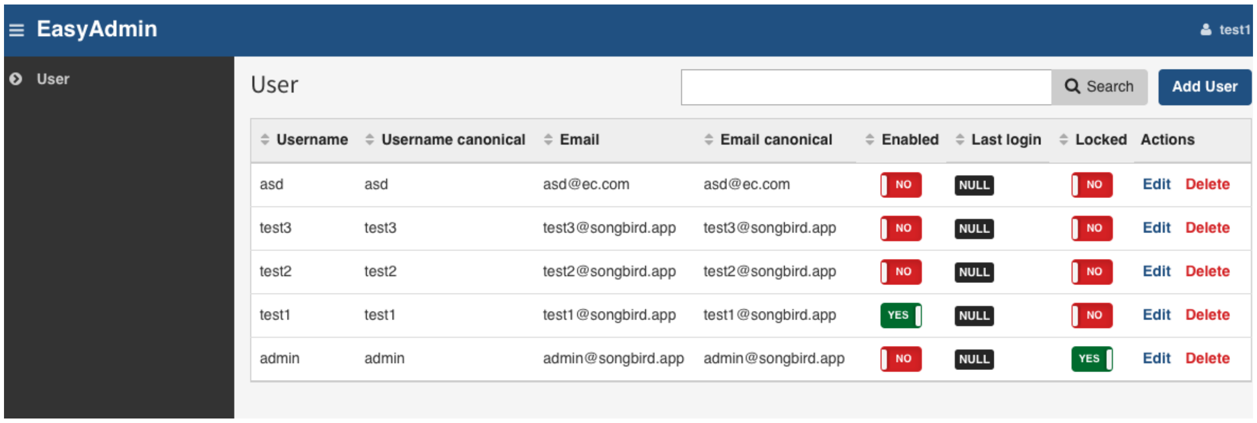
Now, try logging in

```
http://songbird.app:8000/app_dev.php/admin
```

By default, the admin page requires ROLE_ADMIN and above (see app/config/security.yml). So let us login as the administrator

```
username: admin
password: admin
```

wow, we can now see the admin dashboard. If you have accidentally deleted or modified the admin user, remember that you can reset the db with `scripts/resetapp`.



User

Username	Username canonical	Email	Email canonical	Enabled	Last login	Locked	Actions
asd	asd	asd@ec.com	asd@ec.com	NO	NULL	NO	Edit Delete
test3	test3	test3@songbird.app	test3@songbird.app	NO	NULL	NO	Edit Delete
test2	test2	test2@songbird.app	test2@songbird.app	NO	NULL	NO	Edit Delete
test1	test1	test1@songbird.app	test1@songbird.app	YES	NULL	NO	Edit Delete
admin	admin	admin@songbird.app	admin@songbird.app	NO	NULL	YES	Edit Delete

Looks pretty empty huh?

Services

services.yml is important because that is where we define reusable components. Let us create a dummy one for now.

```
# src/AppBundle/Resources/config/services.yml

services:
    # note that this name is important. Its how we reference the class throughou\
t the site.
```

Next, we need to create the a yml service extension[62] and the configuration class so that the framework can load it during the bootstrap.

```
-> mkdir -p src/AppBundle/DependencyInjection
-> touch src/AppBundle/DependencyInjection/AppExtension.php
```

and AppExtensions.php contains

[62] http://symfony.com/doc/current/cookbook/bundles/extension.html

```
# src/AppBundle/DependencyInjection/AppExtension.php

namespace AppBundle\DependencyInjection;

use Symfony\Component\DependencyInjection\ContainerBuilder;
use Symfony\Component\Config\FileLocator;
use Symfony\Component\HttpKernel\DependencyInjection\Extension;
use Symfony\Component\DependencyInjection\Loader\YamlFileLoader;

/**
* This is the class that loads and manages your bundle configuration
*
* To learn more see {@link http://symfony.com/doc/current/cookbook/bundles/exten\
sion.html}
*/
class AppExtension extends Extension
{
    /**
     * {@inheritdoc}
     */
    public function load(array $configs, ContainerBuilder $container)
    {
        $configuration = new Configuration();
        $config = $this->processConfiguration($configuration, $configs);

        $loader = new YamlFileLoader($container, new FileLocator(__DIR__ . '/../\
Resources/config'));
        $loader->load('services.yml');
    }
}
```

now the configuration file

```
# src/AppBundle/DependencyInjection/Configuration.php

namespace AppBundle\DependencyInjection;

use Symfony\Component\Config\Definition\Builder\TreeBuilder;
use Symfony\Component\Config\Definition\ConfigurationInterface;

/**
 * This is the class that validates and merges configuration from your app/confi\
g files
 *
 */
class Configuration implements ConfigurationInterface
{
    /**
     * {@inheritdoc}
     */
    public function getConfigTreeBuilder()
    {
        $treeBuilder = new TreeBuilder();
        $treeBuilder->root('app');

        // Here you should define the parameters that are allowed to
        // configure your bundle. See the documentation linked above for
        // more information on that topic.

        return $treeBuilder;
    }
}
```

Filtering the User fields

The user table has many fields. Remember that you specified what fields you want to display in src/AppBundle/Form/UserType.php? By using EasyAdmin, the creation of forms is now managed by the config. It should be self explanatory. Let us modify the fields.

```
# app/config/easyadmin/user.yml

easy_admin:
    entities:
        User:
            class: AppBundle\Entity\User
            label: 'User Management'
            # for new user
            new:
                fields:
                  - username
                  - firstname
                  - lastname
                  - { property: 'plainPassword', type: 'repeated', type_options:\
 { type: 'Symfony\Component\Form\Extension\Core\Type\PasswordType', first_option\
s: {label: 'Password'}, second_options: {label: 'Repeat Password'}, invalid_mess\
age: 'The password fields must match.'}}
         - { property: 'email', type: 'email', type_options: { trim: tr\
ue } }
         - roles
         - enabled
      edit:
         actions: ['-delete', '-list']
         fields:
          - username
          - firstname
          - lastname
          - { property: 'plainPassword', type: 'repeated', type_option\
s: { type: 'Symfony\Component\Form\Extension\Core\Type\PasswordType', required: \
false, first_options: {label: 'Password'}, second_options: {label: 'Repeat Passw\
ord'}, invalid_message: 'The password fields must match.'}}
          - { property: 'email', type: 'email', type_options: { trim: \
true } }
          - roles
          - enabled
      show:
         actions: ['edit', '-delete', '-list']
         fields:
          - id
          - username
          - firstname
          - lastname
          - email
          - roles
          - enabled
          - { property: 'last_login', type: 'datetime' }
          - modified
          - created
      list:
        title: 'User Listing'
        actions: ['show']
        fields:
         - id
         - username
         - email
         - firstname
```

```
                    - lastname
                    - enabled
                    - roles
                    - { property: 'last_login', type: 'datetime' }
```

Thanks to easyadmin, we have just created CRUD with this yaml file. We have trimmed down all the fields to include only the relevant ones. Note the plainPassword field - We have created 2 password fields with just a simple configuration.

Navigate the site and make sure they are looking good. Looking at mysql, you can see that the password has also been encrypted correctly, indicating that the AdminController's preUpdate function is working.

Redirecting Users to Dashboard After Login

Easy.

```
# app/config/security.yml
...
    firewalls:
        main:
            pattern: ^/
            form_login:
                provider: fos_userbundle
                csrf_provider: security.csrf.token_manager
                default_target_path: /admin
                ...
```

User Roles and Security

What if we want ROLE_USER to login to /admin but restrict them to certain areas only?

We need to subscribe to some events so that we can add some rules based on user's role. Remember the services.yml? It will save the day.

```
# src/AppBundle/Resources/services.yml

services:

    app.subscriber:
            class: AppBundle\EventListener\AppSubscriber
            arguments:
                - "@service_container"
            tags:
                - { name: kernel.event_subscriber }
```

Let's now create the AppSubscriber class

```
# src/AppBundle/EventListener/AppSubscriber.php

namespace AppBundle\EventListener;

use Symfony\Component\EventDispatcher\EventSubscriberInterface;
use Symfony\Component\EventDispatcher\GenericEvent;
use Symfony\Component\Security\Core\Exception\AccessDeniedException;
use JavierEguiluz\Bundle\EasyAdminBundle\Event\EasyAdminEvents;
use Symfony\Component\DependencyInjection\ContainerInterface;

class AppSubscriber implements EventSubscriberInterface
{
    protected $container;

    /**
     * AppSubscriber constructor.
     * @param ContainerInterface $container
     */
    public function __construct(ContainerInterface $container) // this is @servi\
ce_container
    {
        $this->container = $container;
    }

    /**
     * @return array
     */
    public static function getSubscribedEvents()
    {
        // return the subscribed events, their methods and priorities
        return array(
            EasyAdminEvents::PRE_LIST => 'checkUserRights',
            EasyAdminEvents::PRE_EDIT => 'checkUserRights',
            EasyAdminEvents::PRE_SHOW => 'checkUserRights',
            EasyAdminEvents::PRE_NEW => 'checkUserRights',
            EasyAdminEvents::PRE_DELETE => 'checkUserRights'
        );
    }

    /**
     * show an error if user is not superadmin and tries to manage restricted st\
uff
     *
     * @param GenericEvent $event event
     * @return null
     * @throws AccessDeniedException
     */
    public function checkUserRights(GenericEvent $event)
    {

        // if super admin, allow all
        if ($this->container->get('security.authorization_checker')->isGranted('\
ROLE_SUPER_ADMIN')) {
            return;
        }

        $entity = $this->container->get('request_stack')->getCurrentRequest()->q\
uery->get('entity');
        $action = $this->container->get('request_stack')->getCurrentRequest()->q\
uery->get('action');
        $user_id = $this->container->get('request_stack')->getCurrentRequest()->\
query->get('id');

        // if user management, only allow ownself to edit and see ownself
        if ($entity == 'User') {
            // if edit and show
            if ($action == 'edit' || $action == 'show') {
                // check user is himself
                if ($user_id == $this->container->get('security.token_storage')-\
>getToken()->getUser()->getId()) {
                    return;
                }
            }
        }

        // throw exception in all cases
        throw new AccessDeniedException();
    }
}
...
```

Basically, we have created a checkUserRights function to ensure that other than the super admin, only the rightful owner can edit and see his own profile only.

Try logging in as test1:test1 (user id = 2) and see own profile

```
http://songbird.app:8000/app_dev.php/admin/?action=show&entity=User&id=2
```

If test1 tries to see other people's profile, we should get an access denied error.

```
http://songbird.app:8000/app_dev.php/admin/?action=show&entity=User&id=3
```

User list url should give us access denied as well.

```
http://songbird.app:8000/app_dev.php/admin/?action=list&entity=User
```

There is one more thing we need to clean up. If I login as ROLE_USER, I should not be able to see certain fields.

Under the "edit" action, I should not see the roles, enabled, locked, and expired fields.

Under the "show" action, I should not see the created field. User should also be redirected to the show page when the details are updated.

```
# src/AppBundle/Controller/AdminController.php

use JavierEguiluz\Bundle\EasyAdminBundle\Event\EasyAdminEvents;

    ....
     /**
      * @return \Symfony\Component\HttpFoundation\Response
      */
     public function showUserAction()
    {
        $this->dispatch(EasyAdminEvents::PRE_SHOW);
        $id = $this->request->query->get('id');
        $easyadmin = $this->request->attributes->get('easyadmin');
        $entity = $easyadmin['item'];

        $fields = $this->entity['show']['fields'];

        if (!$this->isGranted('ROLE_SUPER_ADMIN')) {
            unset($fields['created']);
        }

        $deleteForm = $this->createDeleteForm($this->entity['name'], $id);

        return $this->render($this->entity['templates']['show'], array(
            'entity' => $entity,
            'fields' => $fields,
            'delete_form' => $deleteForm->createView(),
        ));
    }

    /**
     * when edit user action
     *
     * @return Response|\Symfony\Component\HttpFoundation\RedirectResponse|\Sym\
fony\Component\HttpFoundation\Response
     */
    protected function editUserAction()
    {
        $this->dispatch(EasyAdminEvents::PRE_EDIT);
        $id = $this->request->query->get('id');
        $easyadmin = $this->request->attributes->get('easyadmin');
        $entity = $easyadmin['item'];

        if ($this->request->isXmlHttpRequest() && $property = $this->request->q\
uery->get('property')) {
            $newValue = 'true' === strtolower($this->request->query->get('newVa\
lue'));
            $fieldsMetadata = $this->entity['list']['fields'];

            if (!isset($fieldsMetadata[$property]) || 'toggle' !== $fieldsMetad\
ata[$property]['dataType']) {
```

```
                throw new \RuntimeException(sprintf('The type of the "%s" prope\
rty is not "toggle".', $property));
            }

            $this->updateEntityProperty($entity, $property, $newValue);

            return new Response((string)$newValue);
        }

        $fields = $this->entity['edit']['fields'];

        $editForm = $this->createEditForm($entity, $fields);
        if (!$this->isGranted('ROLE_SUPER_ADMIN')) {
            $editForm->remove('enabled');
            $editForm->remove('roles');
        }

        $deleteForm = $this->createDeleteForm($this->entity['name'], $id);

        $editForm->handleRequest($this->request);
        if ($editForm->isValid()) {
            $this->preUpdateUserEntity($entity);
            $this->em->flush();

            $refererUrl = $this->request->query->get('referer', '');

            return !empty($refererUrl)
                ? $this->redirect(urldecode($refererUrl))
                : $this->redirect($this->generateUrl('easyadmin', array('action\
' => 'show', 'entity' => $this->entity['name'], 'id' => $id)));
        }

        return $this->render($this->entity['templates']['edit'], array(
            'form' => $editForm->createView(),
            'entity_fields' => $fields,
            'entity' => $entity,
            'delete_form' => $deleteForm->createView(),
        ));
    }
```

Easyadmin allows creation of isolated entity functions like "editUserAction". This is brilliant because updating this function won't affect other entities.

Cleaning up

Since we are not going to use FOSUserBundle /profile url to change update user profile, let us remove it from the routing.yml

```
# app/config/routing.yml

#fos_user_profile:
#    resource: "@FOSUserBundle/Resources/config/routing/profile.xml"
#    prefix: /profile

#fos_user_change_password:
#    resource: "@FOSUserBundle/Resources/config/routing/change_password.xml"
#    prefix: /profile
```

Now let us do some cleaning up. Since we are now using EasyAdmin, a lot of files that we have generated using the command line are no longer needed. As you can see, automation is only good if you know what you are doing.

```
git rm src/AppBundle/Controller/UserController.php
git rm src/AppBundle/Form/UserType.php
git rm -rf app/Resources/views/default
git rm -rf src/AppBundle/Tests/Controller/UserControllerTest.php
git rm -rf src/AppBundle/Tests/Controller/DefaultControllerTest.php

# All efforts gone? Don't worry, we will write new tests in the next chapter
git rm -rf tests
git rm codeception.yml
# add all changes
git add .
```

Summary

We have installed a popular Admin system called EasyAdminBundle. We then integrated FOSUser-Bundle with EasyAdminBundle and customised some fields. We have also configured the security of the system such that unless the logged in user is a super admin, the user can only see or update his own profile.

Remember to commit your changes before moving on to the next chapter.

Exercises

- Try installing the SonataAdminBundle[63] yourself and see the differences in both approach.

References

- EasyAdminBundle[64]
- EasyAdminBundle and FOSUserBundle Integration[65]
- EasyAdminBundle views configuration[66]
- Event Listener and Subscribers[67]

[63]https://sonata-project.org/bundles/admin/master/doc/index.html

[64]https://github.com/javiereguiluz/EasyAdminBundle

[65]https://github.com/javiereguiluz/EasyAdminBundle/blob/master/Resources/doc/tutorials/fosuserbundle-integration.md

[66]https://github.com/javiereguiluz/EasyAdminBundle/blob/master/Resources/doc/book/4-edit-new-configuration.md#customizing-the-behavior-of-edit-and-new-views

[67]https://symfony.com/doc/current/doctrine/event_listeners_subscribers.html

Chapter 10: BDD With Codeception (Optional)

This chapter is optional, feel free to skip it if you already have your own testing framework in place.

Behavioural-Driven-Development (BDD) is best used as integration testing[68]. It is the concept of writing tests based on user's behaviour. The way users interact with the software defines the requirements for the software. Once we know the requirements, we are able to write tests and simulate user's interaction with the software.

In BDD, each user's requirement (user story) can be created using the following template:

```
As a ...
I (don't) want to ...
So that ...
```

We can then further breakdown the story into scenarios. For each scenario, we define the "When" (user's action) and the "Then" (acceptance criteria).

```
Given scenario
When ...
Then ...
```

It is a good idea to create a matrix for user stories and test scenarios to fully capture user's requirement as part of the functional specifications.

User Stories

Let us define the user stories for this chapter. We will define the user stories before each chapter from now on.

User Story 10: User Management

[68]https://en.wikipedia.org/wiki/Integration_testing

Story Id	As a	I	So that I
10.1	test1 user	want to login	can access admin functions
10.2	admin user	want to login	can access admin functions
10.3	test3 user	don’t want to login	can prove that this account is disabled
10.4	test1 user	want to manage my own profile	can update it any time
10.5	test1 user	dont’t want to manage other profiles	don’t breach security
10.6	admin user	want to manage all users	can control user access of the system

User Scenarios

We will break the individual story down with user scenarios.

Story ID 10.1: As a test1 user, I want to login, so that I can access admin functions.

Scenario Id	Given	When	Then
10.1.1	Wrong login credentials	I login with the wrong credentials	I should see an error message
10.1.2	See my dashboard content	I login correctly	I should see Access Denied
10.1.3	Logout successfully	I go to the logout url	I should be redirected to the home page
10.1.4	Access admin url without logging in	go to admin url without logging in	I should be redirected to the login page

Story ID 10.2: As a admin user, I want to login, so that I can access admin functions.

Scenario Id	Given	When	Then
10.2.1	Wrong login credentials	I login with the wrong credentials	I should see an error message
10.2.2	See my dashboard content	I login correctly	I should see the text User Management
10.2.3	Logout successfully	go to the logout url	I should be redirected to the home page
10.2.4	Access admin url without logging in	go to admin url without logging in	I should be redirected to the login page

Story ID 10.3: As a test3 user, I don’t want to login successfully, so that I can prove that this account is disabled.

Scenario Id	**Given**	**When**	**Then**
10.3.1	Account disabled	I login with the right credentials	I should see an "account disabled" message

Story ID 10.4: As a test1 user, I want to manage my profile, so that I can update it any time.

Scenario Id	Given**	**When**	Then**
10.4.1	Show my profile	I go to "/admin/?action=show&entity=User&id=2"	I should see test1@songbird.app
10.4.2	Hid uneditable fields	I go to "/admin/?action=edit&entity=User&id=2"	I should not see enabled and roles fields
10.4.3	Update Firstname Only	I go to "/admin/?action=edit&entity=User&id=2" And update firstname only And Submit	I should see content updated
10.4.4	Update Password Only	I go to "/admin/?action=edit&entity=User&id=2" And update password And Submit And Logout And Login Again	I should see content updated And be able to login with the new password

Story ID 10.5: As a test1 user, I don't want to manage other profiles, so that I don't breach security.

Scenario Id	**Given**	**When**	**Then**
10.5.1	List all profiles	I go to "/admin/?action=list&entity=User" url	I should get an "access denied" error.
10.5.2	Show test2 profile	I go to "/admin/?action=show&entity=User&id=3"	I should get an "access denied" error.
10.5.3	Edit test2 user profile	I go to "/admin/?action=edit&entity=User&id=3"	I should get an "access denied" error
10.5.4	See admin dashboard content	I login correctly	I should not see User Management Text

Story ID 10.6: As an admin user, I want to manage all users, so that I can control user access of the system.

Scenario Id	**Given**	**When**	**Then
10.6.1	List all profiles	I go to "/admin/?action=list&entity=User" url	I should see a list of all users in a table
10.6.2	Show test3 user	I go to "/admin/?action=show&entity=User&id=4" url	I should see test3 user details
10.6.3	Edit test3 user	I go to "/admin/?action=edit&entity=User&id=4" url And update lastname	I should see test3 lastname updated on the "List all users" page
10.6.4	Create and Delete new user	I go to "/admin/?action=new&entity=User" And fill in the required fields And Submit And Delete the new user	I should see the new user created and deleted again in the listing page.

Creating the Cest Class

Since we have already deleted the test directory, let us create the testing framework under src/AppBundle

```
# in symfony dir
-> vendor/bin/codecept bootstrap src/AppBundle
-> vendor/bin/codecept build -c src/AppBundle
```

and update the acceptance file again

```yaml
# src/AppBundle/tests/acceptance.suite.yml
class_name: AcceptanceTester
modules:
    enabled:
        - WebDriver:
            url: 'http://songbird.app'
            host: 172.25.0.5
            port: 4444
            browser: phantomjs
            window_size: 1024x768
            capabilities:
                unexpectedAlertBehaviour: 'accept'
                webStorageEnabled: true
        - \Helper\Acceptance
```

Codeception is really flexible in the way we create the test scenarios. Take User Story 1 for example, we can break the user story down into directories and scenarios into cest class. Let us create the files:

```
-> vendor/bin/codecept generate:cest acceptance As_Test1_User/IWantToLogin -c sr\
c/AppBundle
-> vendor/bin/codecept generate:cest acceptance As_An_Admin/IWantToLogin -c src/\
AppBundle
-> vendor/bin/codecept generate:cest acceptance As_Test3_User/IDontWantTologin -\
c src/AppBundle
-> vendor/bin/codecept generate:cest acceptance As_Test1_User/IWantToManageMyOwn\
Profile -c src/AppBundle
-> vendor/bin/codecept generate:cest acceptance As_Test1_User/IDontWantToManageO\
therProfiles -c src/AppBundle
-> vendor/bin/codecept generate:cest acceptance As_An_Admin/IWantToManageAllUser\
s -c src/AppBundle
```

We will create a common class in the bootstrap and define all the constants we need for the test.

```
# src/AppBundle/tests/acceptance/_bootstrap.php

define('ADMIN_USERNAME', 'admin');
define('ADMIN_PASSWORD', 'admin');
define('TEST1_USERNAME', 'test1');
define('TEST1_PASSWORD', 'test1');
define('TEST2_USERNAME', 'test2');
define('TEST2_PASSWORD', 'test2');
// test3 Account is disabled. See data fixtures to confirm.
define('TEST3_USERNAME', 'test3');
define('TEST3_PASSWORD', 'test3');


class Common
{
        public static function login(AcceptanceTester $I, $user, $pass)
    {
        $I->amOnPage('/login');
        $I->fillField('_username', $user);
        $I->fillField('_password', $pass);
        $I->click('_submit');
    }
}
```

Let us try creating story 10.6

```php
# src/AppBundle/tests/acceptance/As_An_Admin/IWantToManageAllUsersCest.php

namespace As_An_Admin;
use \AcceptanceTester;
use \Common;

class IWantToManageAllUsersCest
{
    public function _before(AcceptanceTester $I)
    {
    }

    public function _after(AcceptanceTester $I)
    {
    }

    protected function login(AcceptanceTester $I)
    {
        Common::login($I, ADMIN_USERNAME, ADMIN_PASSWORD);
    }

    /**
     * Scenario 10.6.1
     * @before login
     */
    public function listAllProfiles(AcceptanceTester $I)
    {
        $I->amOnPage('/admin/?action=list&entity=User');
        // the magic of xpath
        $I->canSeeNumberOfElements('//table/tbody/tr',4);
    }
}
```

Noticed the xpath[69] selector?

```
//table/tbody/tr
```

This is the xpath for the show button. How do we know where it is located? We can inspect the elements with the developer tool (available in many browser).

[69]https://msdn.microsoft.com/en-us/library/ms256086(v=vs.110).aspx

You also noticed that the login class is protected rather than public. **Protected class won't be executed** when we run the "runtest" command but we can use it as a pre-requisite when testing listAppProfiles scenario for example, ie the @before login annotation.

listAllProfiles function goes to the user listing page and checks for 4 rows in the table. How do I know about the amOnPage and canSeeNumberOfElements functions? Remembered you ran the command "/bin/codecept build" before? This command generates the AcceptanceTester class to be used in the Cest class. All the functions of the AcceptanceTester class can be found in the "src/AppBundle/Tests/_support/_generated/AcceptanceTesterActions.php" class.

In the test, I used the user listing url directly rather than clicking on the "User Management" link. We should be simulating user clicking on the "User Management" link instead. We will update the test again once we work on the updated UI.

Let us update the runtest script

```
# symfony/scripts/runtest

#!/bin/bash

scripts/resetapp
docker-compose exec php vendor/bin/codecept run acceptance $@ -c src/AppBundle
```

and update the gitignore path

```
# symfony/.gitignore
...
src/AppBundle/tests/_output/*
```

Then, run the test only for scenario 10.6.1

```
-> ./scripts/runtest As_An_Admin/IWantToManageAllUsersCest.php:listAllProfiles
...
OK (1 test, 1 assertion)
```

Looking good, what if the test fails and you want to look at the logs? The log files are all in the "src/AppBundle/tests/_output/" directory.

Let us write another test for scenario 10.6.2. We will simulate clicking on test3 show button and check the page is loading fine.

```
# src/AppBundle/tests/acceptance/As_An_Admin/IWantToManageAllUsersCest.php
...
   /**
    * Scenario 10.6.2
    * @before login
    */
   public function showTest3User(AcceptanceTester $I)
   {
       // go to user listing page
       $I->amOnPage('/admin/?action=list&entity=User');
       // click on show button
       $I->click('Show');
       $I->waitForText('test3@songbird.app');
       $I->canSee('test3@songbird.app');
   }
...
```

run the test now

```
-> ./scripts/runtest As_An_Admin/IWantToManageAllUsersCest.php:showTest3User
```

and you should get a success message.

We will now write the test for scenario 10.6.3

```
# src/AppBundle/tests/acceptance/As_An_Admin/IWantToManageAllUsersCest.php
...
    /**
     * Scenario 10.6.3
     * @before login
     */
    public function editTest3User(AcceptanceTester $I)
    {
        // go to user listing page
        $I->amOnPage('/admin/?action=list&entity=User');
        // click on edit button
        $I->click('Edit');
        // check we are on the right url
        $I->canSeeInCurrentUrl('/admin/?action=edit&entity=User');
        $I->fillField('//input[@value="test3 Lastname"]', 'lastname3 updated');
        // update
        $I->click('//button[@type="submit"]');
        // go back to listing page
        $I->amOnPage('/admin/?action=list&entity=User');
        $I->canSee('lastname3 updated');
        // now revert username
        $I->amOnPage('/admin/?action=edit&entity=User&id=4');
        $I->fillField('//input[@value="lastname3 updated"]', 'test3 Lastname');
        $I->click('//button[@type="submit"]');
        $I->amOnPage('/admin/?action=list&entity=User');
        $I->canSee('test3 Lastname');
    }
...
```

Run the test now to make sure everything is ok before moving on.

```
-> ./scripts/runtest As_An_Admin/IWantToManageAllUsersCest.php:editTest3User
```

and scenario 10.6.4

```
# src/AppBundle/tests/acceptance/As_An_Admin/IWantToManageAllUsersCest.php
...
   /**
    * Scenario 10.6.4
    * @before login
    */
   public function createAndDeleteNewUser(AcceptanceTester $I)
   {
       // go to create page and fill in form
       $I->amOnPage('/admin/?action=new&entity=User');
       $I->fillField('//input[contains(@id, "_username")]', 'test4');
       $I->fillField('//input[contains(@id, "_email")]', 'test4@songbird.app');
       $I->fillField('//input[contains(@id, "_plainPassword_first")]', 'test4');
       $I->fillField('//input[contains(@id, "_plainPassword_second")]', 'test4');
       // submit form
       $I->click('//button[@type="submit"]');
       // go back to user list
       $I->amOnPage('/admin/?entity=User&action=list');
       // i should see new test4 user created
       $I->canSee('test4@songbird.app');

       // now delete user
       // click on edit button
       $I->click('Delete');
```

```
        // wait for model box and then click on delete button
        $I->waitForElementVisible('//button[@id="modal-delete-button"]');
        $I->click('//button[@id="modal-delete-button"]');
        // I can no longer see test4 user
        $I->cantSee('test4@songbird.app');
    }
...
```

createNewUser test is a bit longer. I hope the comments are self explainatory.

Let's run the test just for this scenario.

```
-> ./scripts/runtest As_An_Admin/IWantToManageAllUsersCest.php:createAndDeleteNe\
wUser
```

Feeling confident? We can run all the test together.

```
-> ./scripts/runtest

Dropped database for connection named `songbird`
Created database `songbird` for connection named default
ATTENTION: This operation should not be executed in a production environment.

Creating database schema...
Database schema created successfully!
  > purging database
  > loading [1] AppBundle\DataFixtures\ORM\LoadUserData
Codeception PHP Testing Framework v2.2.8
Powered by PHPUnit 5.7.5 by Sebastian Bergmann and contributors.

Acceptance Tests (9) -----------------------------------------
Testing acceptance
✔ IWantToLoginCest: Try to test (0.00s)
✔ IWantToManageAllUsersCest: List all profiles (26.24s)
✔ IWantToManageAllUsersCest: Show test3 user (16.23s)
✔ IWantToManageAllUsersCest: Edit test3 user (37.66s)
✔ IWantToManageAllUsersCest: Create and delete new user (30.73s)
✔ IDontWantToManageOtherProfilesCest: Try to test (0.00s)
✔ IWantToLoginCest: Try to test (0.00s)
✔ IWantToManageMyOwnProfileCest: Try to test (0.00s)
✔ IDontWantTologinCest: Try to test (0.00s)
---------------------------------------------------------------
```

```


Time: 1.86 minutes, Memory: 15.00MB

OK (9 tests, 7 assertions)
```

Want more detail output? Try this

```
-> ./scripts/runtest --steps
```

How about with debug mode

```
-> ./scripts/runtest -d
```

Tip: If you are using mac and got "too many open files" error, you need to change the ulimit to something bigger

```
-> ulimit -n 2048
```

Add this to your ~/.bash_profile if you want to change the limit everytime you open up a shell.

If your machine is slow, sometimes it might take too long before certain text or element is being detected. In that case, use the "waitForxxx" function before the assert statement, like so

```
# wait for element to be loaded first
# you can see all the available functions in src/AppBundle/Tests/_support/_gener\
ated/AcceptanceTesterActions.php
$I->waitForElement('//div[contains(@class, "alert-success")]');
# now we can do the assert statement
$I->canSeeElement('//div[contains(@class, "alert-success")]');
```

We have only written the BDD tests for user story 10.6. Are you ready to write acceptance tests for the other user stories?

Writing tests can be a boring process but essential if you want your software to be robust. A tip to note is that every scenario must have a closure so that it is self-contained. The idea is that you can run a test scenario by itself without affecting the rest of the scenarios. For example, if you change a password in a scenario, you have to remember to change it back so that you can run the next test without worrying that the password being changed. Alternatively, you could reset the db after every test but this could make running all the tests longer. There are also other ways to achieve this. How could you do it so that it doesn't affect performance?

The workflow in this book is just one of many ways to write BDD tests. It is worth knowing that at the time of writing, many people uses behat[70].

[70]http://docs.behat.org/en/v3.0/

Summary

In this chapter, we wrote our own CEST classes based on different user stories and scenarios. We are now more confident that we have a way to test Songbird's user management functionality as we add more functionalities in the future.

Remember to commit your changes before moving on the next chapter.

Exercises

- Write acceptance test for User Stories 10.1, 10.2, 10.3, 10.4 and 10.5 and make sure all test passes.
- (Optional) Can you think of other business rules for user management? Try adding your own CEST.

References

- More BDD Readings[71]
- User Story[72]
- integration testing[73]

[71]https://en.wikipedia.org/wiki/Behavior-driven_development
[72]https://en.wikipedia.org/wiki/User_story
[73]https://en.wikipedia.org/wiki/Integration_testing

Chapter 11: Customising the Login Process

In the previous chapters, we have created the admin area and wrote tests for managing users in the admin area. The login page and admin area is still looking plain at the moment. There are still lots to do but let us take a break from the backend logic and look at frontend templating. Symfony is using twig as the default templating engine. If you new to twig, have a look at twig[74] website. In this chapter, we will touch up the login interface.

Defining User Stories and Scenarios

11. Reset Password

Story Id	As a	I	So that I
11.1	test1 user	want to reset my password without logging in	have a way to access my account in case I forget or loses my password.

Story ID 11.1: As a test1 user, I want to be able to reset my password without logging in, so that I have a way to access my account in case I forget or loses my password.

Scenario Id	Given	When	Then
11.1.1	Reset Password Successfully	I click on forget password in the login page and go through the whole resetting process	I should be redirected to the dashboard.

Customise the Login Page

I have installed twitter bootstrap[75] in the public dir and created a simple logo for Songbird. You can get all the files by checking out from chapter_11 repo. My Resources dir looks like this:

[74]http://http://twig.sensiolabs.org/doc/templates.html
[75]http://getbootstrap.com/

```
# src/AppBundle/Resource

public/
├── css
│   ├── bootstrap-theme.css
│   ├── bootstrap-theme.css.map
│   ├── bootstrap-theme.min.css
│   ├── bootstrap-theme.min.css.map
│   ├── bootstrap.css
│   ├── bootstrap.css.map
│   ├── bootstrap.min.css
│   ├── bootstrap.min.css.map
│   └── signin.css
├── fonts
│   ├── glyphicons-halflings-regular.eot
│   ├── glyphicons-halflings-regular.svg
│   ├── glyphicons-halflings-regular.ttf
│   ├── glyphicons-halflings-regular.woff
│   └── glyphicons-halflings-regular.woff2
├── images
│   └── logo.png
└── js
    ├── bootstrap.js
    ├── bootstrap.min.js
    ├── jquery.min.js
    └── npm.js
```

Let us create our own base layout. The idea is to extend this layout for all twig files that we create in the future.

```
# clean up old files
-> git rm -rf app/Resources/views/user
```

and base.html.twig

```
# app/Resources/views/base.html.twig

<!DOCTYPE HTML>
<html lang="en-US">
<head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
        <title>{% block title %}{% endblock %}</title>
        {% block stylesheets %}
           <link href="{{ asset('bundles/app/css/bootstrap.min.css') }}" rel="styleshee\
t" />
    {% endblock %}
</head>
<body>
{% block body %}

    {% block content %}{% endblock %}

{% endblock %}

{% block script %}
    <script src="{{ asset('bundles/app/js/jquery.min.js') }}"></script>
    <script src="{{ asset('bundles/app/js/bootstrap.min.js') }}"></script>
{% endblock %}

</body>
</html>
```

To override FOSUserBundle template, have a look at the vendors/friendsofsymfony/Resources/views. Thanks to inheritance, we can override login.html.twig based on the layout that we have created.

Let us create the login file

```
-> mkdir -p app/Resources/FOSUserBundle/views/Security
-> touch app/Resources/FOSUserBundle/views/Security/login.html.twig
```

Now the actual login file:

```
# app/Resources/FOSUserBundle/views/Security/login.html.twig

{% extends "base.html.twig" %}

{% block title %}
    {{ 'layout.login'|trans({}, 'FOSUserBundle') }}
{% endblock %}

{% block stylesheets %}
    {{ parent() }}
    <link href="{{ asset('bundles/app/css/signin.css') }}" rel="stylesheet" />
{% endblock %}

{% trans_default_domain 'FOSUserBundle' %}

{% block content %}

<img src="{{ asset('bundles/app/images/logo.png') }}" class="center-block img-re\
sponsive" alt="Songbird" />

<div class="container">
    <div class="text-center">
        {% if is_granted("IS_AUTHENTICATED_REMEMBERED") %}
            {{ 'layout.logged_in_as'|trans({'%username%': app.user.username}, 'F\
OSUserBundle') }} |
            <a href="{{ path('fos_user_security_logout') }}">
                {{ 'layout.logout'|trans({}, 'FOSUserBundle') }}
            </a>
        {% endif %}
        <form class="form-signin" action="{{ path("fos_user_security_check") }}"\
 method="post">
            {% if error %}
                <div class="alert alert-danger">{{ error.messageKey|trans(error.\
messageData, 'security') }}</div>
            {% endif %}

            <input type="hidden" name="_csrf_token" value="{{ csrf_token }}" />

            <input type="text" id="username" name="_username" class="form-contro\
l" value="{{ last_username }}" required="required" placeholder="{{ 'security.log\
in.username'|trans }}" />

```

```
            <input type="password" id="password" name="_password" class="form-co\
ntrol" required="required" placeholder="{{ 'security.login.password'|trans }}" /\
>

            <div class="checkbox">
              <label>
                    <input type="checkbox" id="remember_me" name="_remember_me" \
value="on" />{{ 'security.login.remember_me'|trans }}
              </label>
            </div>

            <input class="btn btn-lg btn-primary btn-block" type="submit" id="_s\
ubmit" name="_submit" value="{{ 'security.login.submit'|trans }}" />
        </form>
    </div>
</div>
{% endblock %}
```

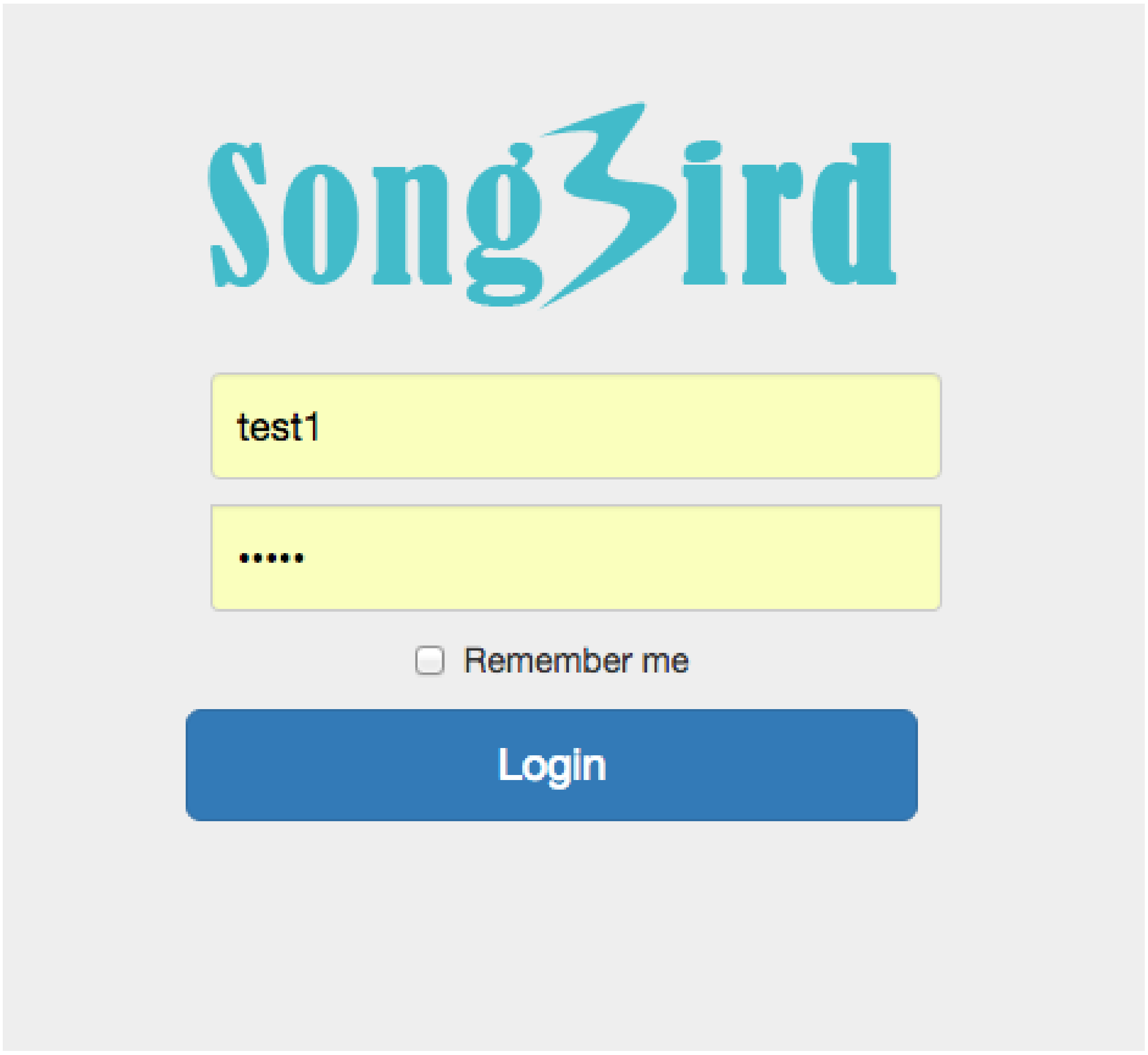
Once we are done, we can get the assets over to the web dir.

```
-> ./scripts/console assets:install --symlink
```

This command basically soft linked[76] everything under the public dir of all bundles over to the document root dir (/web).

Let us go to http://songbird.app:8000/app_dev.php/login and look at our login page now

[76]https://en.wikipedia.org/wiki/Symbolic_link



Installing Mailcatcher

Mailcatcher[77] is excellent for debugging and testing email related functionality. An excellent use case is to test the "forget password" feature.

When using docker, installing a new service is easy (don't need to worry about dependencies). We just need to get a pre-made mailcatcher image and off we go.

[77]https://mailcatcher.me/

```
# docker-compose.yml
...
    mailcatcher:
        image: yappabe/mailcatcher
        ports:
          - 1025:1025
          - 1080:1080
        networks:
            mynet:
                ipv4_address: 172.25.0.6
```

Now let us fire up the new container

```
# under songbird dir
-> docker-compose down
-> docker-compose up -d
```

We also need to make sure swiftmailer is configured to talk to the new mailcatcher host

```
# symfony/app/config/parameters.yml
# configure based on your own settings

parameters:
    database_host: 172.18.0.2
    database_port: 3306
    database_name: songbird
    database_user: root
    database_password: root
    mailer_transport: smtp

    # port 1025 is the mailcatcher sending port
    mailer_host: '172.25.0.6:1025'
    mailer_user: null
    mailer_password: null
    secret: mysecret
```

Customise the Request Password Page

A full login process should also include the password reset functionality in case the user forgets his password. Fortunately again, FOSUSerBundle has all these features built-in already. We just need to make minor tweaks to the process and customise the templates.

The password reset process is as follows:

1. User goes to the forget password page.
2. User enters the username or email.
3. User gets an email a reset link.
4. User clicks on the email and goes to a password reset page.
5. User enters the new password and click submit.
6. User automatically gets redirected to the dashboard.

We will put a link on the login page to the request password page. We can find all the links from the debug:router command (a command you should be familiar by now)

```
-> ./scripts/console debug:router | grep fos
  fos_user_security_login          GET|POST   ANY      ANY     /login             \

  fos_user_security_check          POST       ANY      ANY     /login_check       \

  fos_user_security_logout         GET|POST   ANY      ANY     /logout            \

  fos_user_resetting_request       GET        ANY      ANY     /resetting/request\

  fos_user_resetting_send_email    POST       ANY      ANY     /resetting/send-em\
ail
  fos_user_resetting_check_email   GET        ANY      ANY     /resetting/check-e\
mail
  fos_user_resetting_reset         GET|POST   ANY      ANY     /resetting/reset/{\
token}
```

Let us add the new request password link

```
# app/Resources/FOSUserBundle/views/Security/login.html.twig
...
  <input class="btn btn-lg btn-primary btn-block" type="submit" id="_submit" nam\
e="_submit" value="{{ 'security.login.submit'|trans }}" />

  <div class="checkbox">
    <a href="{{ path('fos_user_resetting_request') }}">forget password</a>
  </div>
...
```

By looking at vendors/friendsofsymfony/Resources/views, we can create all the required twig files to override.

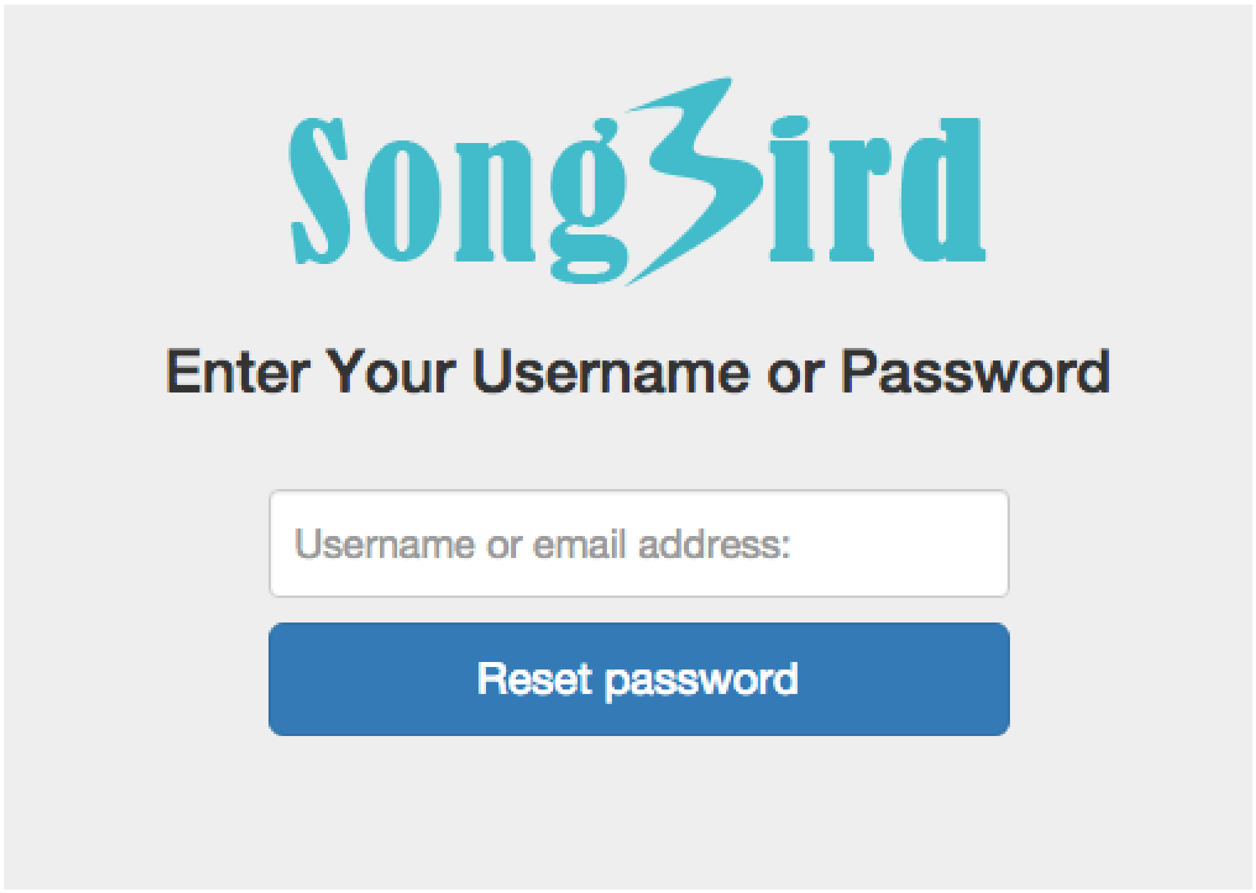
```
-> cd app/Resources/FOSUserBundle/views/
-> mkdir Resetting
-> touch Resetting/checkEmail.html.twig
-> touch Resetting/passwordAlreadyRequested.html.twig
-> touch Resetting/request.html.twig
-> touch Resetting/reset.html.twig
```

Let us create the request password page based on the base.twig.html that we have created earlier.

```
# app/Resources/FOSUserBundle/views/Resetting/request.html.twig

{% extends "base.html.twig" %}

{% trans_default_domain 'FOSUserBundle' %}

{% block title %}
    {{ 'resetting.request.submit'|trans }}
{% endblock %}

{% block stylesheets %}
    {{ parent() }}
    <link href="{{ asset('bundles/app/css/signin.css') }}" rel="stylesheet" />
{% endblock %}

{% block content %}

<img src="{{ asset('bundles/app/images/logo.png') }}" class="center-block img-re\
sponsive" alt="Songbird" />

<div class="container">
    <div class="text-center">
        <h3>Enter Your Username or Password</h3>
        <form class="form-signin" action="{{ path('fos_user_resetting_send_email\
') }}" method="post">
            {% if invalid_username is defined %}
                <div class="alert alert-danger">{{ 'resetting.request.invalid_us\
ername'|trans({'%username%': app.request.get('username')}) }}</div>
            {% endif %}
            <input type="text" id="username" name="username" class="form-control\
" required="required" value="{{ app.request.get('username') }}" placeholder="{{ \
'resetting.request.username'|trans }}" />

```

```
            <input class="btn btn-lg btn-primary btn-block" type="submit" id="_s\
ubmit" name="_submit" value="{{ 'resetting.request.submit'|trans }}" />
        </form>
    </div>
</div>
{% endblock %}
```

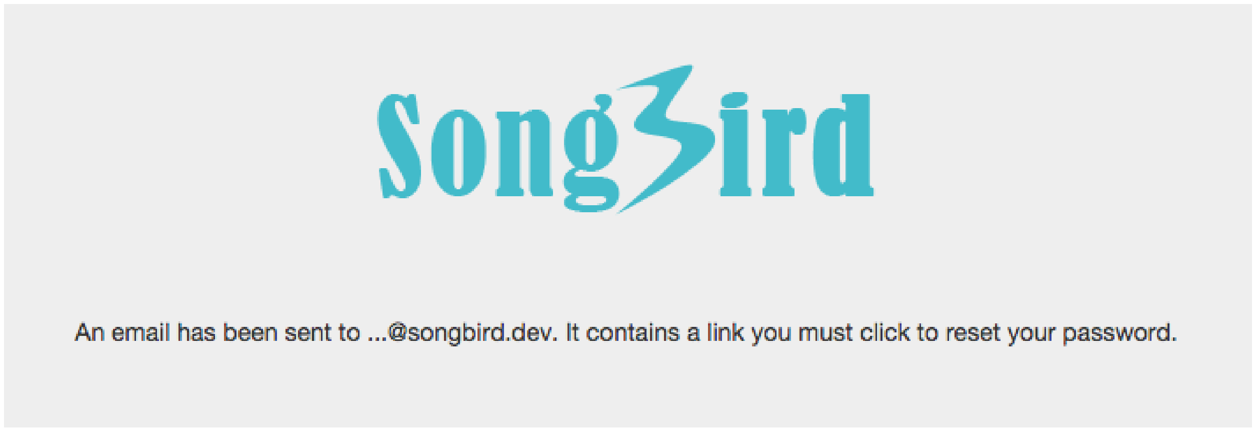
From the login page, click on the forget password link and you should go to the request password page



Likewise we are going to customise the password request success message.

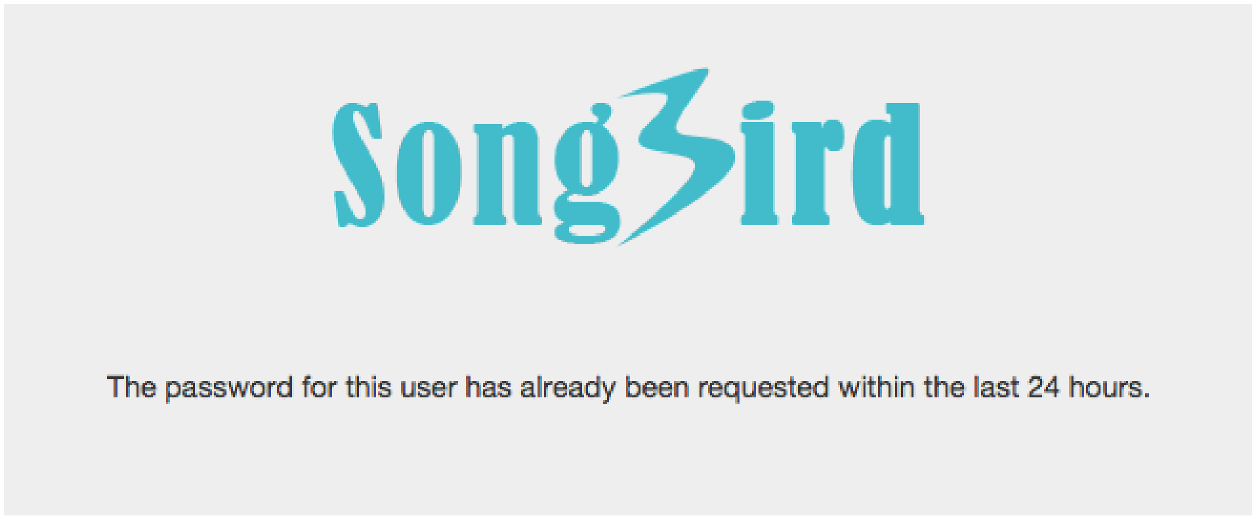
```
# app/Resources/FOSUserBundle/views/Resetting/check_email.html.twig

{% extends "base.html.twig" %}

{% trans_default_domain 'FOSUserBundle' %}

{% block title %}
    {{ 'resetting.request.submit'|trans }}
{% endblock %}

{% block stylesheets %}
    {{ parent() }}
    <link href="{{ asset('bundles/app/css/signin.css') }}" rel="stylesheet" />
{% endblock %}

{% block content %}

    <img src="{{ asset('bundles/app/images/logo.png') }}" class="center-block im\
g-responsive" alt="Songbird" />

    <div class="container">
        <h3> </h3>
        <div class="text-center">
            {{ 'resetting.check_email'|trans({'%tokenLifetime%': tokenLifetime})\
 }}
        </div>
    </div>
{% endblock %}
```

A successful password request looks like this:


An email has been sent to ...@songbird.dev. It contains a link you must click to reset your password.

What if you request password reset more than once in a day? FOSUserBundle actually doesn't allow you to do that.

A screenshot of the password already requested error:

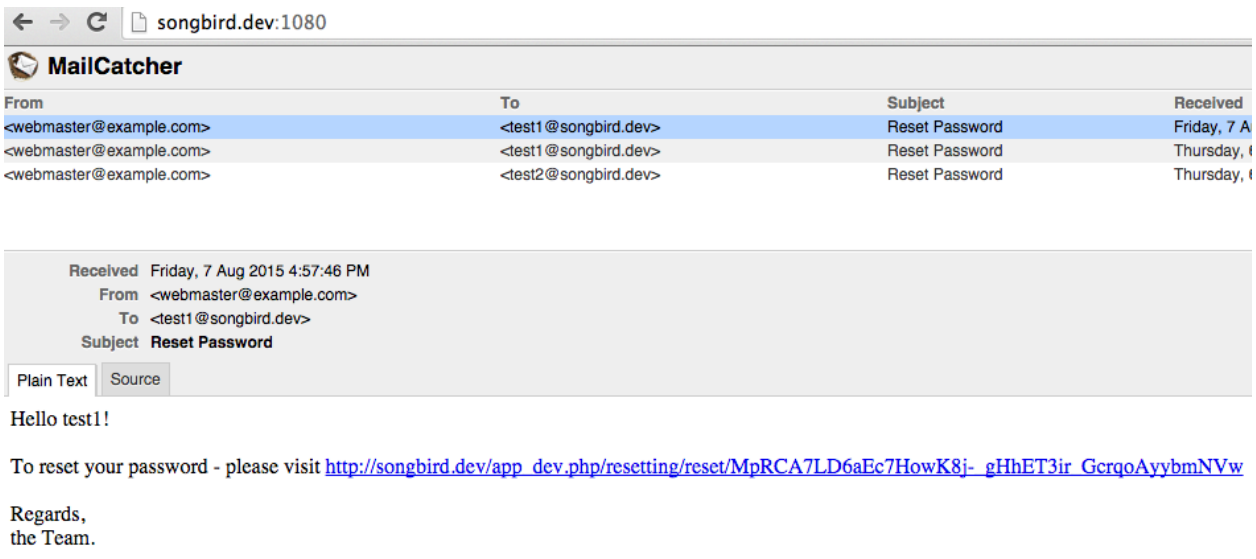


The password for this user has already been requested within the last 24 hours.

When the password request email is send successfully, the user should request a link to reset the password. Our mailcatcher is configured to capture all emails fired.

Let us go to

```
http://songbird.app:1080
```



From	To	Subject	Received
<webmaster@example.com>	<test1@songbird.dev>	Reset Password	Friday, 7 A
<webmaster@example.com>	<test1@songbird.dev>	Reset Password	Thursday,
<webmaster@example.com>	<test2@songbird.dev>	Reset Password	Thursday,

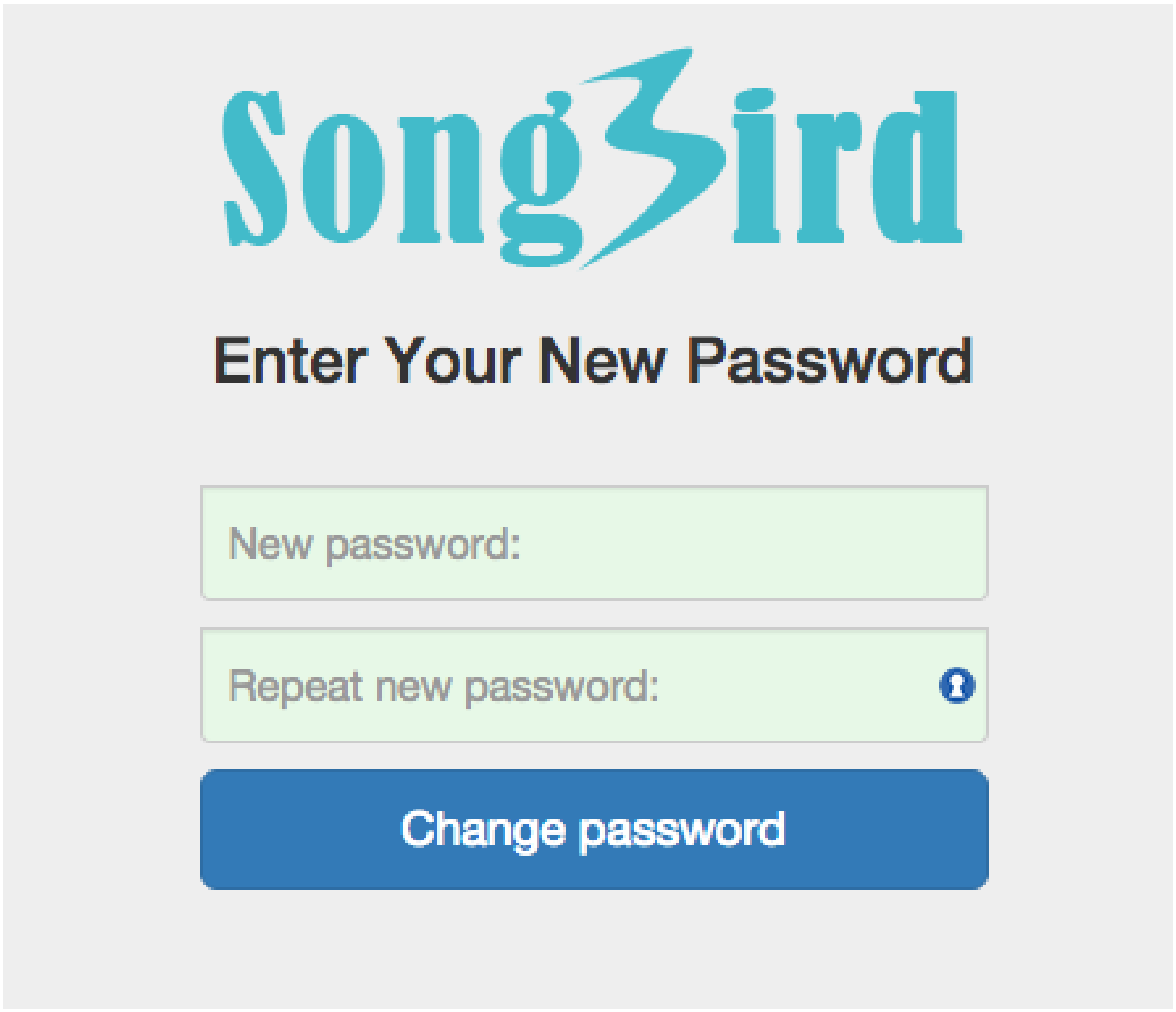
Received Friday, 7 Aug 2015 4:57:46 PM
From <webmaster@example.com>
To <test1@songbird.dev>
Subject **Reset Password**

Hello test1!

To reset your password - please visit http://songbird.dev/app_dev.php/resetting/reset/MpRCA7LD6aEc7HowK8j-_gHhET3ir_GcrqoAyybmNVw

Regards,
the Team.

If you click on the link, you will go to the actual reset page to enter the new password.



Try entering a new password and see what happens. Nothing? Because we haven't add the reset.html.twig. Let us do it now.

```
# app/Resources/FOSUserBundle/views/Resetting/reset.html.twig

{% extends "base.html.twig" %}

{% trans_default_domain 'FOSUserBundle' %}

{% block title %}
    {{ 'resetting.request.submit'|trans }}
{% endblock %}

{% block stylesheets %}
    {{ parent() }}
    <link href="{{ asset('bundles/app/css/signin.css') }}" rel="stylesheet" />
{% endblock %}

{% block content %}

    <img src="{{ asset('bundles/app/images/logo.png') }}" class="center-block im\
g-responsive" alt="Songbird" />

    <div class="container">
        <div class="text-center">
            <h3>Enter Your New Password</h3>
            <form class="form-signin" action="{{ path('fos_user_resetting_reset'\
, {'token': token}) }}" method="post">

                {% if (app.request.get('fos_user_resetting_form')['plainPassword\
']['first']) is defined and (app.request.get('fos_user_resetting_form')['plainPa\
ssword']['first'] != app.request.get('fos_user_resetting_form')['plainPassword']\
['second']) %}
                    <div class="alert alert-danger">
                        {{ 'fos_user.password.mismatch' | trans({}, 'validators'\
) }}
                    </div>
                {% endif %}

                {{ form_widget(form._token) }}
                <input type="password" id="fos_user_resetting_form_plainPassword\
_first" name="fos_user_resetting_form[plainPassword][first]" class="form-control\
" required="required" placeholder="{{ 'form.new_password'|trans }}" />

                <input type="password" id="fos_user_resetting_form_plainPassword\
_second" name="fos_user_resetting_form[plainPassword][second]" class="form-contr\
ol" required="required" placeholder="{{ 'form.new_password_confirmation'|trans }\
}" />

                <input class="btn btn-lg btn-primary btn-block" type="submit" id\
="_submit" name="_submit" value="{{ 'resetting.reset.submit'|trans }}" />
            </form>

        </div>

    </div>
{% endblock %}
```

After you entering the new password and clicking submit, FOSUserBundle will try to redirect you to the fos_user_profile_show route (the profile page which we deleted earlier in route.yml). Since the route no longer exists, you will get an error saying the route no longer exists.

To see what is going on, have a look at vendors/friendsofsymfony/user-bundle/Controller/ResettingController.php::resetAction function. The redirection magic happens after successful form submission.

```
# vendors/friendsofsymfony/user-bundle/Controller/ResettingController.php

namespace FOS\UserBundle\Controller;
...
if ($form->isValid()) {
    $event = new FormEvent($form, $request);
    $dispatcher->dispatch(FOSUserEvents::RESETTING_RESET_SUCCESS, $event);

    $userManager->updateUser($user);

    if (null === $response = $event->getResponse()) {
        $url = $this->generateUrl('fos_user_profile_show');
        $response = new RedirectResponse($url);
    }

    $dispatcher->dispatch(FOSUserEvents::RESETTING_RESET_COMPLETED, new FilterUs\
erResponseEvent($user, $request, $response));

    return $response;
}
...
```

Let's say we want to change the redirection to user's dashboard after successful form submission. What can we do?

Customise the Reset Password Process

We noted that the system dispatches a FOSUserEvents::RESETTING_RESET_SUCCESS event after the form submission is successful. This give us the opportunity to change the response so that the whole redirection logic could be skipped.

Let us update the subscriber class to do our own redirection. "‘ # src/AppBundle/EventListener/AppSubscriber.php

… use FOSUserBundleFOSUserEvents; use FOSUserBundleEventFormEvent; use SymfonyComponentHttpFoundationRedirectResponse; …

```
/**
 * @return array
 */
public static function getSubscribedEvents()
{
    // return the subscribed events, their methods and priorities
    return array(
        ...
        FOSUserEvents::RESETTING_RESET_SUCCESS => 'redirectUserAfterPasswordRese\
t'
    );
}

...

/**
 * Redirect user to another page after password reset is success
 *
 * @param  Configure $event GetResponseUserEvent
 * @return null
 */
public function redirectUserAfterPasswordReset(FormEvent $event)
{
    $url = $this->container->get('router')->generate('admin');
    $event->setResponse(new RedirectResponse($url));
}
...
```

```
Now try to go through the full password reset functionality and see if it works \
for you.

If everything goes well, you should be redirected to the admin dashboard after t\
he password is reset successfully. Its OK if the dashboard is access denied at t\
he moment.

## Update BDD (Optional)

Based on the user story, let us create a new cest file
```

# in symfony dir -> vendor/bin/codecept generate:cest acceptance As_Test1_User/IWantToReset-PasswordWithoutLoggingIn -c src/AppBundle “‘

To automate the checking of emails, we need the mailcatcher module for codeception. Let us update composer

```
-> ./scripts/composer require captbaritone/mailcatcher-codeception-module "1.*" \
--dev
```

Let us now update the acceptance.suite.yml to use the new mailcatcher library

```
# src/AppBundle/tests/acceptance.suite.yml
class_name: AcceptanceTester
modules:
    enabled:
        - WebDriver:
            url: 'http://songbird.app'
            host: 172.25.0.5
            port: 4444
            browser: phantomjs
            window_size: 1024x768
            capabilities:
                unexpectedAlertBehaviour: 'accept'
                webStorageEnabled: true
        - MailCatcher:
                url: 'http://172.25.0.6'
                port: '1080'
        - \Helper\Acceptance
```

we can now rebuild the libraries `# this command will create the mail functions for us to use -> vendor/bin/codecept build -c src/AppBundle`

Do a git diff to see all the mail functions added if you want.

```
-> git diff src/AppBundle/tests/_support/_generated/AcceptanceTesterActions.php
```

See the mailcatcher module[78] for more information.

Let us write the Cest file:

[78] https://github.com/captbaritone/codeception-mailcatcher-module

```
#src/AppBundle/tests/acceptance/As_Test1_User/IWantToResetPasswordWithoutLogging\
InCest.php

namespace As_Test1_User;
use \AcceptanceTester;
use \Common;

/**
 * AS test1 user
 * I WANT to reset my password without logging in
 * SO THAT have a way to access my account in case I forget or loses my password.
 *
 * Class IWantToResetPasswordWithoutLoggingInCest
 * @package As_Test1_User
 */
class IWantToResetPasswordWithoutLoggingInCest
{
    public function _before(AcceptanceTester $I)
    {
    }

    public function _after(AcceptanceTester $I)
    {
    }

    protected function login(AcceptanceTester $I, $username=TEST1_USERNAME, $pas\
sword=TEST1_PASSWORD)
    {
        Common::login($I, $username, $password);
    }

    /**
     * GIVEN Reset Password Successfully
     * WHEN I click on forget password in the login page and go through the whol\
e resetting process
     * THEN I should be redirected to the dashboard.
     *
     * Scenario 11.1.1
     * @param AcceptanceTester $I
     */
    public function resetPasswordSuccessfully(AcceptanceTester $I)
    {
        // reset emails
        $I->resetEmails();
        $I->amOnPage('/login');
        $I->click('forget password');
        $I->fillField('//input[@id="username"]', 'test1');
        $I->click('_submit');
        $I->canSee('It contains a link');

        // Clear old emails from MailCatcher
        $I->seeInLastEmail("Hello test1");
        $link = $I->grabFromLastEmail('@http://(.*)@');
        $I->amOnUrl($link);

        // The password has been reset successfully
        $I->fillField('//input[@id="fos_user_resetting_form_plainPassword_first"\
]', '1111');
        $I->fillField('//input[@id="fos_user_resetting_form_plainPassword_second\
"]', '1111');
        $I->click('_submit');
        // at dashbpard, i should see access denied
        $I->canSee('403');
        // now at show page
        $I->amOnPage('/admin/?action=show&entity=User&id=2');
        $I->canSee('The password has been reset successfully');

        // now login with the new password
        $this->login($I, TEST1_USERNAME, '1111');

        // db has been changed. update it back
        $I->amOnPage('/admin/?action=edit&entity=User&id=2');
        $I->fillField('//input[contains(@id, "_plainPassword_first")]', TEST1_US\
ERNAME);
        $I->fillField('//input[contains(@id, "_plainPassword_second")]', TEST1_P\
ASSWORD);
        $I->click('//button[@type="submit"]');
        // i am on the show page
        $I->canSeeInCurrentUrl('/admin/?action=show&entity=User&id=2');

        // i not should be able to login with the old password
        $this->login($I);
        $I->canSee('403');
    }
```

```
}
```

ready for testing?

```
-> scripts/runtest As_Test1_User/IWantToResetPasswordWithoutLoggingInCest.php
```

Summary

We have updated the aesthetics of the Login and request password change pages. By listening to the reset password event, we redirected user to the dashboard when the event is triggered. Finally, we wrote BDD tests to make sure this functionality is repeatable in the future.

Exercises

- (Optional) Try to be fancy with the login layout and css. How do you use FOSUserBundle's layout.html.twig?

References

- Twig Templating[79]
- Overriding FOSUserBundle Templates[80]
- FOSUserBundle Emails[81]
- Codeception Mailcatcher Module[82]

[79]http://twig.sensiolabs.org/doc/templates.html
[80]http://symfony.com/doc/master/bundles/FOSUserBundle/overriding_templates.html
[81]http://symfony.com/doc/current/bundles/FOSUserBundle/emails.html
[82]https://github.com/captbaritone/codeception-mailcatcher-module

Chapter 12: The Admin Panel Part 2

Let us continue with tweaking EasyAdmin by changing the layout and try more adventurous stuff like creating our own dashboard.

Tweaking the UI

Its easy to change the theme colour and add our own custom css. For the sake of manageability, let us create a new file, design.yml

```
# app/config/easyadmin/design.yml

easy_admin:
  design:
    brand_color: '#5493ca'
    assets:
      css:
        - /bundles/app/css/style.css
```

and create a new css

```
# src/AppBundle/Resources/public/css/style.css

.user-menu a{
    color: rgba(255, 255, 255, 0.8);
}
```

Next, We will will overwrite layout.html.twig by copying it into our own views dir.

```
-> cp vendor/javiereguiluz/easyadmin-bundle/Resources/views/default/layout.html.\
twig app/Resources/EasyAdminBundle/views/default/
```

We will change the logo and top menu. The top menu will include a link to edit the user and logout.

```
# app/Resources/EasyAdminBundle/views/default/layout.html.twig

...

{% block header_logo %}
    <a class="logo {{ easyadmin_config('site_name')|length > 14 ? 'logo-long' }}\
" title="{{ easyadmin_config('site_name')|striptags }}" href="{{ path('easyadmin\
') }}">
        <img src="/bundles/app/images/logo.png" />
    </a>
{% endblock header_logo %}
...
{% block user_menu %}
    <span class="sr-only">{{ 'user.logged_in_as'|trans(domain = 'EasyAdminBundle\
') }}</span>
    <i class="hidden-xs fa fa-user">
        {% if app.user %}
            <a href="{{ path('easyadmin') }}/?entity=User&action=show&id={{ app.\
user.id }}">{{ app.user.username|default('user.unnamed'|trans(domain = 'EasyAdmi\
nBundle')) }}</a>
        {% else %}
            {{ 'user.anonymous'|trans(domain = 'EasyAdminBundle') }}
        {% endif %}
    </i>
    <i class="hidden-xs fa fa-sign-out"><a href="{{ path('fos_user_security_logo\
ut') }}">Logout</a></i>
{% endblock user_menu %}
```

Let us create the dashboard content.

```
{%  extends '@EasyAdmin/default/layout.html.twig' %}

{% block main %}
<p>
    Dear {{ app.user.firstname }} {{ app.user.lastname }},
</p>
<p>
    You are last logged in at {{ app.user.lastLogin | date('Y-m-d H:i:s') }}
</p>

<p>
    The whole project can be forked from <a href="https://github.com/bernardpeh/\
songbird">github</a>
</p>

{% endblock %}
```

Noticed how I extended the layout.html.twig and just change the relevant blocks?

Your Dashboard

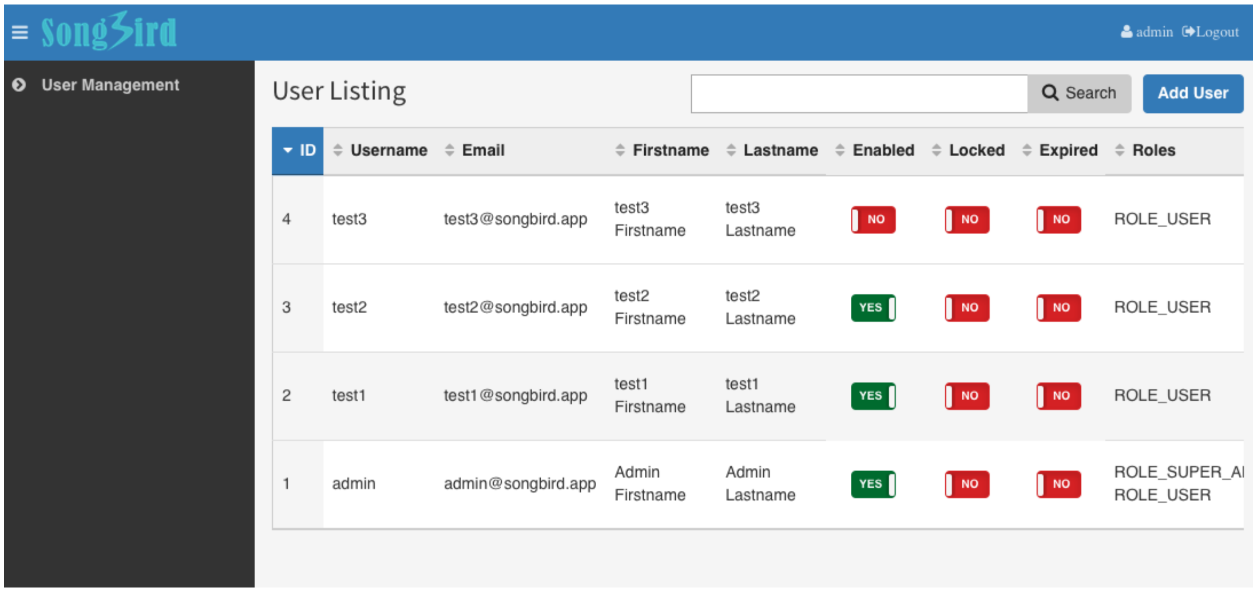
Let us now create a new dashboard page via the standard way. We need a new route.

```
# src/AppBundle/Controller/AdminController.php

use Sensio\Bundle\FrameworkExtraBundle\Configuration\Route;
use Symfony\Component\HttpFoundation\Request;
...


    /**
     * @Route("/dashboard", name="dashboard")
     *
     * @param Request $request
     * @return \Symfony\Component\HttpFoundation\Response
     *
     */
    public function dashboardAction(Request $request)
    {
        return $this->render('@EasyAdmin/default/dashboard.html.twig');
    }
    ...
```

now copy the assets to the web dir from command line.

```
-> ./scripts/console assets:install --symlink
```

login and then refresh the browser.



User Listing

ID	Username	Email	Firstname	Lastname	Enabled	Locked	Expired	Roles
4	test3	test3@songbird.app	test3 Firstname	test3 Lastname	NO	NO	NO	ROLE_USER
3	test2	test2@songbird.app	test2 Firstname	test2 Lastname	YES	NO	NO	ROLE_USER
2	test1	test1@songbird.app	test1 Firstname	test1 Lastname	YES	NO	NO	ROLE_USER
1	admin	admin@songbird.app	Admin Firstname	Admin Lastname	YES	NO	NO	ROLE_SUPER_AI ROLE_USER

Menu Tweaking

Normal users should not see the left entities menus. Again, let us copy the menu.html.twig modify it.

```
-> cp vendor/javiereguiluz/easyadmin-bundle/Resources/views/default/menu.html.tw\
ig app/Resources/EasyAdminBundle/views/default/
```

and the actual menu.html.twig

```
# app/Resources/EasyAdminBundle/views/default/menu.html.twig

...
{% block main_menu_before %}{% endblock %}
<ul class="sidebar-menu">
    {% block main_menu %}
        {% if is_granted('ROLE_SUPER_ADMIN') %}
            {% for item in easyadmin_config('design.menu') %}
                <li class="{{ item.type == 'divider' ? 'header' }} {{ item.child\
ren is not empty ? 'treeview' }} {{ app.request.query.get('menuIndex')|default(-\
1) == loop.index0 ? 'active' }} {{ app.request.query.get('submenuIndex')|default\
(-1) != -1 ? 'submenu-active' }}">
                    {{ helper.render_menu_item(item, _entity_config.translation_\
domain|default('messages')) }}
```

```

                    {% if item.children|default([]) is not empty %}
                        <ul class="treeview-menu">
                            {% for subitem in item.children %}
                                <li class="{{ subitem.type == 'divider' ? 'heade\
r' }} {{ app.request.query.get('menuIndex')|default(-1) == loop.parent.loop.inde\
x0 and app.request.query.get('submenuIndex')|default(-1) == loop.index0 ? 'activ\
e' }}">
                                    {{ helper.render_menu_item(subitem, _entity_\
config.translation_domain|default('messages')) }}
                                </li>
                            {% endfor %}
                        </ul>
                    {% endif %}
                </li>
            {% endfor %}
        {% endif %}
    {% endblock main_menu %}
</ul>
{% block main_menu_after %}{% endblock %}
```

This way of filtering menu access is rather *stupid* and serves just as an exercise for now. We will describe a better way to user manage our admin area in the later chapters.

Removing hardcoding of admin prefix

There is one more thing to mention before we end this chapter. At the moment, the admin url seems to be prefixed to '/admin/xx'. What if we want it to be a bit harder to guess, like 'admin9/xx'? This is a good security feature. Let us create a variable in the config.yml

```
# app/config/config.yml
...
parameters:
    admin_path: admin
    ...
```

We can now use this variable in other places. Once we change this variable, it will automatically update the prefix in all places for us.

```
# app/config/routing.yml
...
# easyadmin
easy_admin_bundle:
    resource: "@AppBundle/Controller/AdminController.php"
    type:     annotation
    prefix:   /%admin_path%

...
```

and

```
# app/config/security.yml
...
    firewalls:
        dev:
            pattern: ^/(_(profiler|wdt|error)|css|images|js)/
            security: false

        main:
            anonymous: ~
            pattern: ^/
            form_login:
                provider: fos_userbundle
                csrf_provider: security.csrf.token_manager
                default_target_path: /%admin_path%/dashboard
            logout:       true

    access_control:
        - { path: ^/login$, role: IS_AUTHENTICATED_ANONYMOUSLY }
        # We do not allow user registration
        # - { path: ^/register, role: IS_AUTHENTICATED_ANONYMOUSLY }
        - { path: ^/resetting, role: IS_AUTHENTICATED_ANONYMOUSLY }
        - { path: ^/%admin_path%/, role: ROLE_USER }
...
```

The default admin page is now default_target_path: /%admin_path%/dashboard

Try changing admin_path to something else and check if all the routes have been updated. Let’s change the admin_path back to ‘admin’ after back.

Update BDD Test (Optional)

Now that we have defined the admin layout, we should update BDD tests for seeMyDashboardContent to test on the dashboard content.

Scenario Id	Given	When	Then
10.1.2	See my dashboard content	I login correctly	I should not see the text "User Management" and should see the text "Dear test1"

Scenario Id	Given	When	Then
10.2.2	See my dashboard content	I login correctly	I should see the text "User Management" and "Dear Admin"

Also with the left menu installed, we should be clicking on the links rather than going to the page directly. In all the test files, replace all amOnPage methods to "click" method.

```
# go to page directly
$I->amOnPage('/admin/?action=list&entity=User');

# replace it with
$I->click('User Management');
```

Once you are confident that all your tests are correct, run it and fix it till everything passes.

Summary

In this chapter, we have touched up the admin area and created a simple dashboard block.

The admin area is now looking more polished.

Exercises

- Try creating another url and view yourself. (Optional)
- Review and Update BDD for all admin and test1 user stories. (Optional)

References

- Twig Templating[83]

[83]http://symfony.com/doc/current/templating.html

Chapter 13: Internalisation

No CMS is complete with being being able to support multiple languages (i18n[84]). So far we have been typing english directly into the twig templates. This is quick and easy but not the best practice. What if we are marketing our software to the french market? Wouldn't it be nice if the interface could be in french rather than english? Its time consuming to create translations for every term that we use but its worth the effort if you want to make your software global.

What about Google Translate[85]? This should be the last option and not be used for professional purposes. Internalisation is something you want to work on early in the software development phase rather than later.

Define User Story

13. Internalisation

Story Id	As a	I	So that I
13.1	test1 user	want to be able to switch language	can choose my preferred language anytime.

Story ID 13.1: As a test1 user, I want to be able to switch language, so that I can choose my preferred language anytime.

Scenario Id	Given	When	Then
13.1.1	Locale in french	I login and switch language to french	I should be able to see the dashboard in french till I switched back to english

Translations for the AppBundle

let us create the translation files in the AppBundle. The naming convention for the file is domain.language_prefix.file_format, eg app.en.xlf.

Let us create the translation directory.

[84] https://en.wikipedia.org/wiki/Internationalization_and_localization
[85] http://translate.google.com.au

```
-> mkdir -p src/AppBundle/Resources/translations
```

and the actual app.en.xlf

```
# src/AppBundle/Resources/translations/app.en.xlf

<?xml version="1.0"?>
<xliff version="1.2" xmlns="urn:oasis:names:tc:xliff:document:1.2">
    <file source-language="en" datatype="plaintext" original="file.ext">
        <body>
            <trans-unit id="1">
                <source>dashboard.welcome.title</source>
                <target>Welcome to SongBird CMS</target>
            </trans-unit>
            <trans-unit id="2">
                <source>dashboard.welcome.credit</source>
                <target>The whole project can be forked from <![CDATA[
                <a href="https://github.com/bernardpeh/songbird">github</a>
                ]]></target>
            </trans-unit>
            <trans-unit id="3">
                <source>dashboard.welcome.last_login</source>
                <target>You last login at</target>
            </trans-unit>
        </body>
    </file>
</xliff>
```

Likewise, we need to create the translation file for french.

```
# src/AppBundle/Resources/translations/app.fr.xlf

<?xml version="1.0"?>
<xliff version="1.2" xmlns="urn:oasis:names:tc:xliff:document:1.2">
    <file source-language="en" datatype="plaintext" original="file.ext">
        <body>
            <trans-unit id="1">
                <source>dashboard.welcome.title</source>
                <target>Bienvenue à SongBird CMS</target>
            </trans-unit>
            <trans-unit id="2">
                <source>dashboard.welcome.credit</source>
                <target>L'ensemble du projet peut être fourchue de <![CDATA[
                <a href="https://github.com/bernardpeh/songbird">github</a>
                ]]></target>
            </trans-unit>
            <trans-unit id="3">
                <source>dashboard.welcome.last_login</source>
                <target>Vous dernière connexion au</target>
            </trans-unit>
        </body>
    </file>
</xliff>
```

Update the Dashboard

How do we get the twig files to do the translation? You would have seen glimpse of it while working with the login files.

Let us update the dashboard template.

```
# app/Resources/EasyAdminBundle/views/default/dashboard.html.twig

...
{% block main %}
<p>
    Dear {{ app.user.firstname }} {{ app.user.lastname }},
</p>
<p>
    {{ 'dashboard.welcome.last_login' | trans({}, 'app') }} {{ app.user.lastLogi\
n | date('Y-m-d H:i:s') }}
</p>

<p>
    {{ 'dashboard.welcome.credit' | trans({}, 'app') | raw }}
</p>

{% endblock %}
```

refresh your browser and have a look. If things are not working, remember to clear the cache.

```
-> ./scripts/resetapp
```

By default, we are using english, so you should see that the english version is translated. To see all the translations in english for the AppBundle,

```
-> ./scripts/console debug:translation en AppBundle
```

You should see a lot of missing translations for the FOSUserBundle. Don't worry about that for now.

Tip: Again, don't remember this command. Just type in "scripts/console debug:translation" in the command line to see the options.

What about french? How do we set the locale? Just update the parameters in the config.yml

```
# app/config/config.yml
...
parameters:
    locale: fr
    admin_path: admin
...
```

Now refresh the dashboard and you should see the welcome block translated.

Its french. Viola!

How do we make the language dynamic? Perhaps we should have a selector on the top menu for users to select the language and persists it throughout the session.

Let us update the translation files

```
# src/AppBundle/Resources/translations/app.en.xlf

<?xml version="1.0"?>
<xliff version="1.2" xmlns="urn:oasis:names:tc:xliff:document:1.2">
    <file source-language="en" datatype="plaintext" original="file.ext">
        <body>
            ...
            <trans-unit id="4">
                <source>admin.link.user_management</source>
                <target>User Management</target>
            </trans-unit>
            <trans-unit id="5">
                <source>admin.link.profile</source>
                <target>My Profile</target>
            </trans-unit>
        </body>
    </file>
</xliff>
```

and

```
# src/AppBundle/Resources/translations/app.fr.xlf

<?xml version="1.0"?>
<xliff version="1.2" xmlns="urn:oasis:names:tc:xliff:document:1.2">
    <file source-language="en" datatype="plaintext" original="file.ext">
        <body>
            ...
            <trans-unit id="4">
                <source>admin.link.user_management</source>
                <target>Gestion des utilisateurs</target>
            </trans-unit>
            <trans-unit id="5">
                <source>admin.link.profile</source>
                <target>Mon profil</target>
            </trans-unit>
        </body>
    </file>
</xliff>
```

and

```
# app/config/easyadmin/user.yml
...
    entities:
        User:
            class: AppBundle\Entity\User
            label: admin.link.user_management
            ...
```

Now let us update the menu translation in menu.html.twig

```
# app/Resources/EasyAdminBundle/views/default/menu.html.twig
...
<ul class="sidebar-menu">
    {% block main_menu %}
        {% if is_granted('ROLE_SUPER_ADMIN') %}
            {% for item in easyadmin_config('design.menu') %}
                <li class="{{ item.type == 'divider' ? 'header' }} {{ item.child\
ren is not empty ? 'treeview' }} {{ app.request.query.get('menuIndex')|default(-\
1) == loop.index0 ? 'active' }} {{ app.request.query.get('submenuIndex')|default\
(-1) != -1 ? 'submenu-active' }}">

```

```
                    {{ helper.render_menu_item(item, 'app') }}

                    {% if item.children|default([]) is not empty %}
                        <ul class="treeview-menu">
                            {% for subitem in item.children %}
                                <li class="{{ subitem.type == 'divider' ? 'heade\
r' }} {{ app.request.query.get('menuIndex')|default(-1) == loop.parent.loop.inde\
x0 and app.request.query.get('submenuIndex')|default(-1) == loop.index0 ? 'activ\
e' }}">
                                    {{ helper.render_menu_item(subitem, _entity_\
config.translation_domain|default('messages')) }}
                                </li>
                            {% endfor %}
                        </ul>
                    {% endif %}
                </li>
            {% endfor %}
        {% endif %}
    {% endblock main_menu %}
</ul>
```

Sticky Locale

Let us create the supported languages in config.yml

```
# app/config/config.yml
...
parameters:
    # set this to english as default
    locale: en
    supported_lang: [ 'en', 'fr']
    admin_path: admin
...
twig:
    debug:            "%kernel.debug%"
    strict_variables: "%kernel.debug%"
    globals:
        supported_lang: %supported_lang%
...
```

We have created a variable called supported_lang (consisting of an array) and passed it to twig as a global variable.

Now in the layout twig

```
# app/Resources/EasyAdminBundle/views/default/easy_admin/layout.html.twig

...
<title>
    {% block page_title %}
        {{ 'dashboard.welcome.title' | trans({}, 'app') }}
    {% endblock %}
</title>

{% set urlPrefix = (app.environment == 'dev') ? '/app_dev.php/' : '/' %}
...
{% block user_menu %}
    <i class="fa fa-language" aria-hidden="true">
        <select id="lang" name="lang">
            {% for lang in supported_lang %}
                <option value="{{ lang }}">{{ lang }}</option>
            {% endfor %}
        </select>
    </i>
    <i class="hidden-xs fa fa-user">
    {% if app.user %}
        <a href="{{ path('easyadmin') }}/?entity=User&action=show&id={{ app.user\
.id }}">{{ app.user.username|default('user.unnamed'|trans(domain = 'EasyAdminBun\
dle')) }}</a>
    {% else %}
        {{ 'user.anonymous'|trans(domain = 'EasyAdminBundle') }}
    {% endif %}
        </i>
    <i class="hidden-xs fa fa-sign-out"><a href="{{ path('fos_user_security_logo\
ut') }}">{{ 'layout.logout'|trans({}, 'FOSUserBundle') }}</a></i>
{% endblock user_menu %}
...
{% block body_javascript %}
    <script>
        // select the box based on locale
        var lang = document.getElementById('lang');
        lang.value = '{{ app.request.getLocale() }}';
        // redirect user if user change locale
        lang.addEventListener('change', function () {
            window.location = '{{ urlPrefix }}' + document.getElementById('lang'\
).value + '/locale';
```

```
        });
    </script>
{% endblock body_javascript %}
```

Note that we have made logic and css tweaks to the top nav. The new CSS is as follows:

```
# src/AppBundle/Resources/public/css/styles.css

.user-menu a{
    color: rgba(255, 255, 255, 0.8);
}

i {
    padding: 5px;
}

#lang {
    color: #333;
}
```

The new language dropdown box allows user to select a language and if there is a change in the selection, the user is redirected to a url /{_locale}/locale where the change of locale magic is supposed to happen.

and create a new controller from the command line.

```
-> ./scripts/console generate:controller --controller=AppBundle:Locale -n
```

Tip: This command can save you some time but not much in this case. You don't have to memorise it. Always use the "help" option if unsure, ie

```
-> ./scripts/console help generate:controller
```

The controller code in full:

```
# src/AppBundle/Controller/LocaleController.php

namespace AppBundle\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\Controller;
use Sensio\Bundle\FrameworkExtraBundle\Configuration\Route;
use Symfony\Component\HttpFoundation\Request;
use Sensio\Bundle\FrameworkExtraBundle\Configuration\Method;

class LocaleController extends Controller
{
    /**
     * Redirects user based on their referer
     *
     * @Route("/{_locale}/locale", name="app_set_locale")
     * @Method("GET")
     */
    public function setLocaleAction(Request $request, $_locale)
    {
        $auth_checker = $this->get('security.authorization_checker');

        // if referrer exists, redirect to referrer
        $referer = $request->headers->get('referer');
        if ($referer) {
            return $this->redirect($referer);
        }
        // if logged in, redirect to dashboard
        elseif ($auth_checker->isGranted('ROLE_USER')) {
            return $this->redirectToRoute('dashboard');
        }
        // else redirect to homepage
        else {
            return $this->redirect('/');
        }
    }
}
```

As you can see, the annotation defines the the new route /{_locale}/locale. To make sure that this route is working,

```
-> ./scripts/console debug:router | grep locale
 app_set_locale                          GET      ANY    ANY  /{_locale}/locale
```

The AdminController gets the request object and redirects the user to the referer if there is one. If not, it redirects the user to either the admin dashboard or the homepage depending if the user is logged in or not. Again, don't memorise security.authorization_checker. Google around, make intelligent guesses and use the command line to verify the containers.

```
-> ./scripts/console debug:container | grep security
...
```

We said that the controller is the place where magic happens... but where is the magic? We haven't even change the locale session yet! We cannot change it at the controller level because it is too late. We have to change it very early on in the Http workflow[86]

Basically, what we need to do is to hook on to the kernel.request event and modify some logic there. The symfony cookbook has good information on sticky sessions[87].

We have create an event subscriber before. As a practice, let us create an event listener this time round.

```
# src/AppBundle/Resources/config/services.yml

...
  app.locale.listener:
    class: AppBundle\EventListener\LocaleListener
    arguments:
      - "%kernel.default_locale%"
    tags:
      - { name: kernel.event_listener, event: kernel.request, priority: 17 }
...
```

Why did we use priority 17? Every listener has a priority. The higher the priority, the earlier the listener will be executed. We want our custom LocaleListener to be earlier than the Kernel's LocaleListener. According to Kernel events[88], The kernel LocaleListener has priority 16. Let us go abit higher, ie 17.

Now we need to create the LocaleListener class.

[86]http://symfony.com/doc/current/components/http_kernel/introduction.html
[87]http://symfony.com/doc/current/cookbook/session/locale_sticky_session.html
[88]http://symfony.com/doc/current/reference/events.html

```
# src/AppBundle/EventListener/LocaleListener.php

namespace AppBundle\EventListener;

use Symfony\Component\HttpKernel\Event\GetResponseEvent;
use Symfony\Component\HttpFoundation\RedirectResponse;

class LocaleListener
{
    private $defaultLocale;

    public function __construct($defaultLocale = 'en')
    {
        $this->defaultLocale = $defaultLocale;
    }

    public function onKernelRequest(GetResponseEvent $event)
    {
        $request = $event->getRequest();
        if (!$request->hasPreviousSession()) {
            return;
        }

        // try to see if the locale has been set as a _locale routing parameter
        if ($locale = $request->attributes->get('_locale')) {
            $request->getSession()->set('_locale', $locale);
        } else {
            // if no explicit locale has been set on this request, use one from \
the session
            $request->setLocale($request->getSession()->get('_locale', $this->de\
faultLocale));
        }
    }
}
```

To see what is going on with the events sequencing,

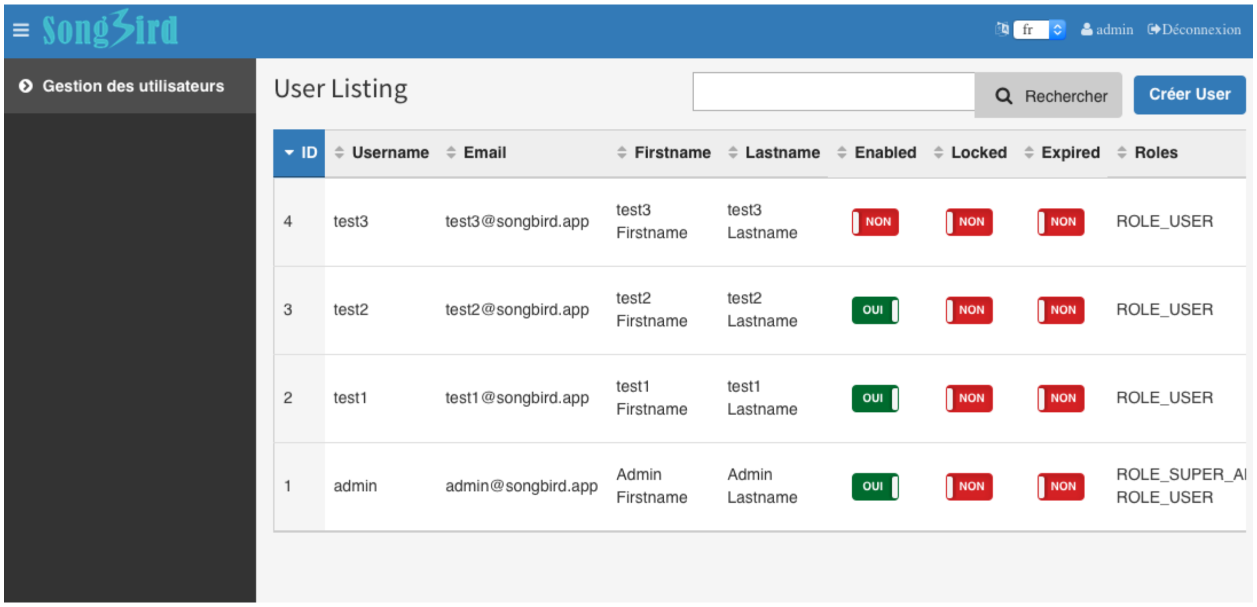
```
# now view the event dispatcher list
-> ./scripts/console debug:event-dispatcher kernel.request
```

Look at the kernel.request section and you should see our custom event listener ranked just above the kernel LocaleListener.

Can you use the AppSubscriber class that we have created to do the same job?

Now, clear the cache and refresh the browser. Try changing the locale dropdown and see for yourself.

```
-> ./scripts/resetapp
```



User Listing

ID	Username	Email	Firstname	Lastname	Enabled	Locked	Expired	Roles
4	test3	test3@songbird.app	test3 Firstname	test3 Lastname	NON	NON	NON	ROLE_USER
3	test2	test2@songbird.app	test2 Firstname	test2 Lastname	OUI	NON	NON	ROLE_USER
2	test1	test1@songbird.app	test1 Firstname	test1 Lastname	OUI	NON	NON	ROLE_USER
1	admin	admin@songbird.app	Admin Firstname	Admin Lastname	OUI	NON	NON	ROLE_SUPER_AI ROLE_USER

dashboard with translation

Try changing the priority to 15 of kernel.event_listener tag and see what happens?

Update BDD (Optional)

Let us create the cest file based on the User Story.

```
# in symfony
-> vendor/bin/codecept generate:cest acceptance As_Test1_User/IWantToSwitchLangu\
ageCest -c src/AppBundle
```

now within the cest file:

```
#src/AppBundle/tests/acceptance/As_Test1_User/IWantToSwitchLanguageCest.php

namespace As_Test1_User;
use \AcceptanceTester;
use \Common;

/**
 * AS test1 user
 * I want to be able to switch language
 * SO THAT I can choose my preferred language anytime.
 *
 * Class IWantToSwitchLanguageCest
 * @package As_Test1_User
 */
class IWantToSwitchLanguageCest
{
    public function _before(AcceptanceTester $I)
    {
            Common::login($I, TEST1_USERNAME, TEST1_PASSWORD);
    }

    public function _after(AcceptanceTester $I)
    {
    }

    /**
     * GIVEN Locale in french
     * WHEN I login and switch language to french
     * THEN I should be able to see the dashboard in french till I switched back\
 to english
     *
     * Scenario 13.1.1
     */
    public function localeInFrench(AcceptanceTester $I)
    {
        // switch to french
        $I->selectOption('//select[@id="lang"]', 'fr');
        // I should be able to see "my profile" in french
        $I->waitForText('Déconnexion');
        $I->canSee('Déconnexion');
        $I->click('test1');
        // now in show profile page
        $I->waitForText('Éditer');
        $I->canSee('Éditer');
        // now switch back to english
        $I->selectOption('//select[@id="lang"]', 'en');
        $I->waitForText('Edit');
        $I->canSee('Edit');
    }
}
```

Lets run the test to make sure everything is working.

```
-> ./scripts/runtest As_Test1_User/IWantToSwitchLanguageCest.php
```

Since the UI has been changed, some previous BDD tests might fail. Fix them and re-run the full BDD tests till everything passes.

```
-> ./scripts/runtest
```

Summary

In this chapter, we learned how to create translation files and updated the twig files to handle the translation. We have also created a language switcher in the admin area and added a new BDD test to test internalisation.

I am not french and my french translation might not be correct as I was using google translate. The use of french in this book is just an example.

Remember to commit all your changes before moving on to the next chapter.

Exercises

- Remember all the twig files you have created in chapter 11[89]? Update them to support i18n.
- (Optional) Try creating translations in other languages other than french.

References

- Symfony translations[90]

[89]https://github.com/bernardpeh/songbird/tree/chapter_11
[90]http://symfony.com/doc/current/book/translation.html

- Translations best practices[91]
- Sticky Session[92]
- Kernel Events[93]
- EasyAdmin Translations[94]

---

[91]http://symfony.com/doc/current/best_practices/i18n.html
[92]http://symfony.com/doc/current/cookbook/session/locale_sticky_session.html
[93]http://symfony.com/doc/current/reference/events.html
[94]https://github.com/javiereguiluz/EasyAdminBundle/blob/master/Resources/doc/tutorials/i18n.md

Chapter 14: Uploading Files

Our CMS should allow uploading of files. Let's say we want to allow user to upload their own profile image. EasyAdmin has nice integration with a popular bundle called VichUploaderBundle[95].

Update User Stories

Let us update the user stories that we have created before.

Story ID 10.6: As an admin user, I want to manage all users, so that I can control user access of the system.

Scenario Id	Given	When	Then
10.6.1	List all profiles	I go show profile page" url	I should see a list of all users in a table with image fields

Story ID 10.1: As a test1 user, I want to manage my profile, so that I can update it any time.

Scenario Id	Given	When	Then
10.4.1	Show my profile	I go to show profile page	I should see test1@songbird.app and an Image field
10.4.5	Delete and Add profile image	I go to edit profile page And delete profile image and add a new image	I should see an empty profile, previous profile image gone and then a new one appearing in the file system.
10.4.6	Update profile image Only	I go to edit profile page And update profile image and submit	I should see user profile updated and previous profile image gone from file system.

Install Vich Uploader Bundle

Add the vich uploaded bundle to composer

[95] https://github.com/dustin10/VichUploaderBundle

```
-> ./scripts/composer require vich/uploader-bundle ^1.4
```

In config.yml, we need to add a few parameters

```
# app/config/config.yml
...
parameters:
    locale: en
    supported_lang: [ 'en', 'fr']
    admin_path: admin
    app.profile_image.path: /uploads/profiles
    ...
# Vich Configuration
vich_uploader:
    db_driver: orm
    mappings:
        profile_images:
            uri_prefix: '%app.profile_image.path%'
            upload_destination: '%kernel.root_dir%/../web/uploads/profiles'
            # this will allow all uploaded filenames to be unique
            namer: vich_uploader.namer_uniqid
...
```

and in Appkernel.php

```
# app/AppKernel.php
...
public function registerBundles()
{
    return array(
        // ...
        new Vich\UploaderBundle\VichUploaderBundle(),
    );
}
...
```

We have to add new image fields to the user table.

```
# src/AppBundle/Entity/User.php

namespace AppBundle\Entity;

use FOS\UserBundle\Model\User as BaseUser;
use Doctrine\ORM\Mapping as ORM;
use Symfony\Component\HttpFoundation\File\File;
use Vich\UploaderBundle\Mapping\Annotation as Vich;

/**
 * User
 *
 * @ORM\Table(name="user")
 * @ORM\Entity(repositoryClass="AppBundle\Repository\UserRepository")
 * @ORM\HasLifecycleCallbacks()
 * @Vich\Uploadable
 */
class User extends BaseUser
{
    ...

    /**
     * @ORM\Column(type="string", length=255, nullable=true)
     * @var string
     */
    private $image = '';
    /**
     * @Vich\UploadableField(mapping="profile_images", fileNameProperty="image")
     * @var File
     */
    private $imageFile;

    ...

    /**
     * @param File|null $image
     */
    public function setImageFile(File $image = null)
    {
        $this->imageFile = $image;
        // at least 1 field needs to change for doctrine to save
        if ($image) {
            $this->setModified(new \DateTime());
        }
    }
    /**
     * @return File
     */
    public function getImageFile()
    {
        return $this->imageFile;
    }
    /**
     * @param $image
     */
    public function setImage($image)
    {
        $this->image = $image;
    }
    /**
     * @return string
     */
    public function getImage()
    {
        return $this->image;
    }
    ...
```

Since we have changed the entity, we have to remember to log the db changes so that we can deploy the db changes to production easily.

```
-> ./scripts/console doctrine:migrations:diff
Generated new migration class to "/var/www/symfony/app/../src/AppBundle/Doctrine\
Migrations/Version20170208142520.php" from schema differences.
...
```

Looks good, we can now reset the app

```
-> ./scripts/resetapp
```

We can now verify that the new image field has been added.

```
-> ./scripts/mysql "show columns from user"

+-----------------------+--------------+------+-----+---------+----------------+
| Field                 | Type         | Null | Key | Default | Extra          |
+-----------------------+--------------+------+-----+---------+----------------+
| id                    | int(11)      | NO   | PRI | NULL    | auto_increment |
| username              | varchar(180) | NO   |     | NULL    |                |
| username_canonical    | varchar(180) | NO   | UNI | NULL    |                |
| email                 | varchar(180) | NO   |     | NULL    |                |
| email_canonical       | varchar(180) | NO   | UNI | NULL    |                |
| enabled               | tinyint(1)   | NO   |     | NULL    |                |
| salt                  | varchar(255) | YES  |     | NULL    |                |
| password              | varchar(255) | NO   |     | NULL    |                |
| last_login            | datetime     | YES  |     | NULL    |                |
| confirmation_token    | varchar(180) | YES  | UNI | NULL    |                |
| password_requested_at | datetime     | YES  |     | NULL    |                |
| roles                 | longtext     | NO   |     | NULL    |                |
| firstname             | varchar(255) | YES  |     | NULL    |                |
| lastname              | varchar(255) | YES  |     | NULL    |                |
| modified              | datetime     | NO   |     | NULL    |                |
| created               | datetime     | NO   |     | NULL    |                |
| image                 | varchar(255) | NO   |     | NULL    |                |
+-----------------------+--------------+------+-----+---------+----------------+
```

We need to create the new upload folder

```
-> mkdir -p web/uploads/profiles
```

but we should ignore in git. In .gitignore

```
...
/web/bundles/
/web/uploads/
...
```

Update Fixtures

Let us update the Image field to help us with automate testing.

```
# src/AppBundle/DataFixtures/ORM/LoadUserData.php

...
    public function load(ObjectManager $manager)
        {
            $userManager = $this->container->get('fos_user.user_manager');

            // add admin user
            $admin = $userManager->createUser();
            $admin->setUsername('admin');
            $admin->setEmail('admin@songbird.app');
            $admin->setPlainPassword('admin');
            $userManager->updatePassword($admin);
            $admin->setEnabled(1);
            $admin->setFirstname('Admin Firstname');
            $admin->setLastname('Admin Lastname');
            $admin->setRoles(array('ROLE_SUPER_ADMIN'));
            $admin->setImage('test_profile.jpg');
            $userManager->updateUser($admin);

            // add test user 1
            $test1 = $userManager->createUser();
            $test1->setUsername('test1');
            $test1->setEmail('test1@songbird.app');
            $test1->setPlainPassword('test1');
            $userManager->updatePassword($test1);
            $test1->setEnabled(1);
            $test1->setFirstname('test1 Firstname');
            $test1->setLastname('test1 Lastname');
            $test1->setImage('test_profile.jpg');
            $userManager->updateUser($test1);

            // add test user 2
            $test2 = $userManager->createUser();
            $test2->setUsername('test2');
            $test2->setEmail('test2@songbird.app');
            $test2->setPlainPassword('test2');
            $userManager->updatePassword($test2);
            $test2->setEnabled(1);
            $test2->setFirstname('test2 Firstname');
            $test2->setLastname('test2 Lastname');
            $test2->setImage('test_profile.jpg');
            $userManager->updateUser($test2);

            // add test user 3
            $test3 = $userManager->createUser();
            $test3->setUsername('test3');
            $test3->setEmail('test3@songbird.app');
            $test3->setPlainPassword('test3');
            $userManager->updatePassword($test3);
            $test3->setEnabled(0);
            $test3->setFirstname('test3 Firstname');
            $test3->setLastname('test3 Lastname');
            $test3->setImage('test_profile.jpg');
            $userManager->updateUser($test3);

            // use this reference in data fixtures elsewhere
            $this->addReference('admin_user', $admin);
        }
    ...
```

Just create any pic called test_profile.jpg and put it in the src/AppBundle/tests/_data dir. If you run out of ideas, you can use the jpg from my branch. We will update the resetapp script to copy the test_profile.jpg to the web folder.

```
# scripts/resetapp

#!/bin/bash
rm -rf var/cache/*
# scripts/console cache:clear --no-warmup
scripts/console doctrine:database:drop --force
scripts/console doctrine:database:create
scripts/console doctrine:schema:create
scripts/console doctrine:fixtures:load -n

# copy test data over to web folder
cp src/AppBundle/tests/_data/test_profile.jpg web/uploads/profiles/
```

Update UI

Let us update the UI to include the image field.

```
# app/config/easyadmin/user.yml

easy_admin:
    entities:
        User:
            class: AppBundle\Entity\User
            label: 'User Management'
            # for new user
            new:
                fields:
                  - username
                  - firstname
                  - lastname
                  - { property: 'plainPassword', type: 'repeated', type_options:\
 { type: 'Symfony\Component\Form\Extension\Core\Type\PasswordType', first_option\
s: {label: 'Password'}, second_options: {label: 'Repeat Password'}, invalid_mess\
age: 'The password fields must match.'}}
                  - { property: 'email', type: 'email', type_options: { trim: tr\
ue } }
                  - { property: 'imageFile', type: 'vich_image' }
                  - roles
                  - enabled
            edit:
                  actions: ['-delete', '-list']
                  fields:
                    - username
                    - firstname
                    - lastname
                    - { property: 'plainPassword', type: 'repeated', type_option\
s: { type: 'Symfony\Component\Form\Extension\Core\Type\PasswordType', required: \
false, first_options: {label: 'Password'}, second_options: {label: 'Repeat Passw\
ord'}, invalid_message: 'The password fields must match.'}}
                    - { property: 'email', type: 'email', type_options: { trim: \
true } }
                    - { property: 'imageFile', type: 'vich_image' }
                    - roles
                    - enabled
            show:
                  actions: ['edit', '-delete', '-list']
                  fields:
                    - id
                    - { property: 'image', type: 'image', base_path: '%app.profi\
le_image.path%'}
                    - username
                    - firstname
                    - lastname
                    - email
                    - roles
                    - enabled
                    - { property: 'last_login', type: 'datetime' }
                    - modified
                    - created
            list:
                title: 'User Listing'
                actions: ['show']
                fields:
                  - id
                  - { property: 'image', type: 'image', base_path: '%app.profile\
_image.path%'}
                  - username
                  - email
                  - firstname
                  - lastname
                  - enabled
                  - roles
                  - { property: 'last_login', type: 'datetime' }
```

Let us resetapp, login and have a look

```
-> ./scripts/resetapp
```

Update BDD (Optional)

In this chapter, we might need other modules like Db and Filesystem. Let us update our acceptance config file

```
# src/AppBundle/Tests/acceptance.suite.yml

class_name: AcceptanceTester
modules:
    enabled:
        - WebDriver:
            url: 'http://songbird.app'
            browser: chrome
            window_size: 1024x768
            capabilities:
                unexpectedAlertBehaviour: 'accept'
                webStorageEnabled: true
        - MailCatcher:
            url: 'http://songbird.app'
            port: '1080'
        - Filesystem:
        - Db:
        - \Helper\Acceptance
```

and our db credentials

```
# src/AppBundle/codeception.yml

actor: Tester
paths:
    tests: tests
    log: tests/_output
    data: tests/_data
    support: tests/_support
    envs: tests/_envs
settings:
    bootstrap: _bootstrap.php
    colors: true
    memory_limit: 1024M
extensions:
    enabled:
        - Codeception\Extension\RunFailed
modules:
    config:
        Db:
            dsn: 'mysql:host=172.25.0.2;dbname=songbird'
            user: 'root'
            password: 'root'
            dump: tests/_data/dump.sql
            populate: false
```

now run the build to update the acceptance library

```
-> vendor/bin/codecept build -c src/AppBundle
```

You should now have lots of new functions to use in AcceptanceTesterActions.php.

Write the stories in this chapter as a practice. Again, get all the test to pass before moving to the next chapter.

> Tip 1: To test a file upload, put a file under src/AppBundle/Tests/_data folder and you can then use the attachFile function like so

```
$I->waitForElementVisible('//input[@type="file"]');
$I->attachFile('//input[@type="file"]', 'testfile.png');
$I->click('Submit');
```

Remember to commit everything before moving on to the next chapter.

Summary

In this chapter, we have integrated VichuploadBundle with EasyAdminBundle. We made minor change to the ui and added new BDD tests.

Exercises

- Integrate SonataMediaBundle[96] instead. (Optional)
- Write BDD Test for User stories in this chapter. (Optional)

References

- EasyAdmin Vich Uploader[97]
- Codeception DB Module[98]

[96]https://sonata-project.org/bundles/media/master/doc/index.html
[97]https://github.com/bernardpeh/EasyAdminBundle/blob/master/Resources/doc/tutorials/upload-files-and-images.md
[98]http://codeception.com/docs/modules/Db

Chapter 15: Logging User Activities

A proper CMS needs a logging mechanism. We are talking about the admin area, not the front end. If something happens, we need to know what was done and what happened? We can log user activities in a file system but it is not very efficient. File system is good for logging errors - see monolog[99]. Ideally, we need a database solution.

Define User Stories

After the user logs in, we want to record the username, current_url, previous_url, CRUD action, data on every page that the user visits. These data should be recorded in a new table. When the user is deleted, we do not want the logs associated with the user to be deleted, therefore the 2 tables are not related.

There is a popular loggable doctrine extension[100] that we can use. However, it is easy enough to built one for ourselves.

15. Logging User Activitiy

Story Id	As a	I	So that I
15.1	admin user	want to manage user logs	check on user activity anytime.
15.2	test1 user	don't want to manage user logs	don't breach security

Story ID 15.1: As an admin, I want to manage user logs, so that I can check on user activity anytime.

Scenario Id	Given	When	Then
15.1.1	List user log	I click on user log on the left menu	I should see more than 1 row in the table
15.1.2	Show user log 1	I go to the first log entry	I should see the text "/admin/dashboard"

Story ID 15.2: As test1 user, I don't want to manage user logs, so that I don't breach security.

[99]http://symfony.com/doc/current/cookbook/logging/monolog.html
[100]https://github.com/Atlantic18/DoctrineExtensions/blob/master/doc/loggable.md

Scenario Id	Given	When	Then
15.2.1	List user log	I go to the user log url	I should get an access denied message
15.2.2	Show log 1	I go to the show log id 1 url	I should get an access denied message
15.2.3	Edit log 1	I go to the edit log id 1 url	I should get an access denied message

Implementation

We will create a new entity called UserLog. The UserLog entity should have the following fields: id, username, current_url, referrer, action, data, created.

```
-> ./scripts/console doctrine:generate:entity --entity=AppBundle:UserLog --forma\
t=annotation --fields="username:string(255) current_url:text referrer:text(nulla\
ble=true) action:string(255) data:text(nullable=true) created:datetime" --no-int\
eraction
```

Again, don't memorise this command. You can find out more about this command using

```
-> ./scripts/console doctrine:generate:entity --help
```

or from the online documentation[101]

In the entity, note that we are populating the username field from the user entity but not creating a constraint between the 2 entities. The reason for that is that when we delete the user, we still want to keep the user entries. We haven't really gone through doctrine yet. You can read more about association mapping here[102] if we want them to be related. We will touch on doctrine again in the later chapters.

```
# src/AppBundle/Entity/UserLog.php

<?php

/**
 * UserLog
 *
 * @ORM\Table(name="user_log")
 * @ORM\Entity(repositoryClass="AppBundle\Repository\UserLogRepository")
```

[101]http://symfony.com/doc/current/bundles/SensioGeneratorBundle/commands/generate_doctrine_entity.html

[102]http://doctrine-orm.readthedocs.org/projects/doctrine-orm/en/latest/reference/association-mapping.html

```
 */
class UserLog
{
    /**
     * @var int
     *
     * @ORM\Column(name="id", type="integer")
     * @ORM\Id
     * @ORM\GeneratedValue(strategy="AUTO")
     */
    private $id;

    /**
     * @var string
     *
     * @ORM\Column(name="username", type="string", length=255)
     */
    private $username;

    /**
     * @var string
     *
     * @ORM\Column(name="current_url", type="text")
     */
    private $current_url;

    /**
     * @var string
     *
     * @ORM\Column(name="referrer", type="text", nullable=true)
     */
    private $referrer;

    /**
     * @var string
     *
     * @ORM\Column(name="action", type="string", length=255)
     */
    private $action;

    /**
     * @var string
     *
     * @ORM\Column(name="data", type="text", nullable=true)
     */
    private $data;

    /**
     * @var \DateTime
     *
     * @ORM\Column(name="created", type="datetime")
     */
    private $created;


    /**
     * Get id
     *
     * @return int
     */
    public function getId()
    {
        return $this->id;
    }

    /**
     * Set username
     *
     * @param string $username
     *
     * @return UserLog
     */
    public function setUsername($username)
    {
        $this->username = $username;

        return $this;
    }

    /**
     * Get username
     *
     * @return string
     */
    public function getUsername()
    {
        return $this->username;
    }

    /**
     * Set currentUrl
     *
     * @param string $currentUrl
     *
     * @return UserLog
     */
    public function setCurrentUrl($currentUrl)
    {
        $this->current_url = $currentUrl;

        return $this;
    }

    /**
     * Get currentUrl
     *
     * @return string
     */
    public function getCurrentUrl()
    {
        return $this->current_url;
    }

    /**
     * Set referrer
     *
     * @param string $referrer
     *
     * @return UserLog
     */
    public function setReferrer($referrer)
    {
        $this->referrer = $referrer;

        return $this;
    }

    /**
     * Get referrer
     *
     * @return string
     */
    public function getReferrer()
    {
        return $this->referrer;
    }

    /**
     * Set action
     *
     * @param string $action
     *
     * @return UserLog
     */
    public function setAction($action)
    {
        $this->action = $action;

        return $this;
    }

    /**
     * Get action
     *
     * @return string
     */
    public function getAction()
    {
        return $this->action;
    }

    /**
     * Set data
     *
     * @param string $data
     *
     * @return UserLog
     */
    public function setData($data)
    {
        $this->data = $data;

        return $this;
    }

    /**
     * Get data
     *
     * @return string
     */
    public function getData()
    {
        return $this->data;
    }

    /**
     * Set created
     *
     * @param \DateTime $created
     *
     * @return UserLog
     */
    public function setCreated($created)
    {
        $this->created = $created;

        return $this;
    }

    /**
     * Get created
     *
     * @return \DateTime
     */
    public function getCreated()
    {
        return $this->created;
    }
}
```

Next, we will intercept the kernel.request event.

```
# src/AppBundle/EventListener/AppSubscriber.php

...
use Symfony\Component\HttpKernel\Event\GetResponseEvent;
use Symfony\Component\HttpKernel\KernelEvents;
use AppBundle\Entity\UserLog;
...
    public static function getSubscribedEvents()
    {
        // return the subscribed events, their methods and priorities
        return array(
            EasyAdminEvents::PRE_LIST => 'checkUserRights',
            EasyAdminEvents::PRE_EDIT => 'checkUserRights',
            EasyAdminEvents::PRE_SHOW => 'checkUserRights',
            FOSUserEvents::RESETTING_RESET_SUCCESS => 'redirectUserAfterPassword\
Reset',
            KernelEvents::REQUEST => 'onKernelRequest'
        );
    }
    ...

    /**
     * We will log request to db on every url change
     *
     * @param GetResponseEvent $event
     */
    public function onKernelRequest(GetResponseEvent $event)
    {
        $request = $event->getRequest();
        $current_url = $request->server->get('REQUEST_URI');
        // ensures we track admin only.
        $admin_path = $this->container->getParameter('admin_path');

        // only log admin area and only if user is logged in. Dont log search by\
 filter
        if (!is_null($this->container->get('security.token_storage')->getToken()\
) && preg_match('/\/'.$admin_path.'\//', $current_url)
            && ($request->query->get('filter') === null) && !preg_match('/\/user\
log\//', $current_url)) {

            $em = $this->container->get('doctrine.orm.entity_manager');
            $log = new UserLog();
            $log->setData(json_encode($request->request->all()));
            $log->setUsername($this->container->get('security.token_storage')->g\
etToken()->getUser()
                ->getUsername());
            $log->setCurrentUrl($current_url);
            $log->setReferrer($request->server->get('HTTP_REFERER'));
            $log->setAction($request->getMethod());
            $log->setCreated(new \DateTime('now'));
            $em->persist($log);
            $em->flush();
        }
    }
        ...
```

Let us create the new menu.

```
# app/config/easyadmin/userlog.yml

easy_admin:
    entities:
        UserLog:
            class: AppBundle\Entity\UserLog
            label: admin.link.user_log
            show:
                  actions: ['list', '-edit', '-delete']
            list:
                actions: ['show', '-edit', '-delete']
```

and the translation.

```
# src/AppBundle/Resources/translations/app.en.xlf
...
        <trans-unit id="6">
            <source>admin.link.user_log</source>
            <target>User Log</target>
        </trans-unit>
...
```

and the french version

```
# src/AppBundle/Resources/translations/app.fr.xlf
...
        <trans-unit id="6">
            <source>admin.link.user_log</source>
            <target>Connexion utilisateur</target>
        </trans-unit>
...
```

Now reset the db, re-login again, click on the user log menu and you will see the new menu on the left.

There were db changes, let us capture the change so that we can update production when we need to.

```
-> ./scripts/console doctrine:migrations:diff
```

and we can reset the db now.

```
-> ./scripts/resetapp
```

Update BDD (Optional)

Let us create the cest files.

```
-> vendor/bin/codecept generate:cest acceptance As_An_Admin/IWantToManageUserLog\
 -c src/AppBundle
-> vendor/bin/codecept generate:cest acceptance As_Test1_User/IDontWantToManageU\
serLog -c src/AppBundle
```

Tip: The assert module is very useful.

Let us add the assert module

```
# src/AppBundle/Tests/acceptance.suite.yml
...
        - Asserts:
        ...
```

Let us rebuild the libraries

```
-> vendor/bin/codecept build -c src/AppBundle/
```

Again, I will leave you to write the bdd tests. The more detail your scenario is, the better the test coverage will be. Get all the test to pass and remember to commit everything before moving on to the next chapter.

Summary

In this chapter, we created a new entity called UserLog and used the kernel request event to inject the required request data into the database.

Exercises

- Modify the UserLog entity such that deleting the user in the User entity will delete the associated user entries in the UserLog entity. (optional)
- What are the pros and cons of allowing CRUD actions on log entries?
- Can you use doctrine loggable extension to achieve what was achieved here? (optional)
- Can you implement automated entity logging using Traits[103]?

References

- Symfony Events[104]
- Doctrine Extensions[105]
- Doctrine Association Mapping[106]

[103]http://php.net/manual/en/language.oop5.traits.php
[104]http://symfony.com/doc/current/reference/events.html
[105]http://symfony.com/doc/current/cookbook/doctrine/common_extensions.html
[106]http://doctrine-orm.readthedocs.org/projects/doctrine-orm/en/latest/reference/association-mapping.html

Chapter 16: Improving Performance and Troubleshooting

If your site uses a lot of javascript and css, one good optimisation strategy is to merge the css and js into just one file each. That way, its one http request rather multiple, improving the loading time. There are also tools to find out where bottlenecks are and fix them.

Install Blackfire

Head to blackfire.io[107] (another great product by sensiolabs) and sign up for an account. In https://blackfire.io/account, get the client and server (id and token). Enter them in .env.

We only need to configure blackfire.

```
# .env
# Blackfire io
BLACKFIRE_SERVER_ID=your_id
BLACKFIRE_SERVER_TOKEN=your_id
```

Let us add the blackfire image to docker-compose.

```
...
    blackfire:
       image: blackfire/blackfire
       environment:
            - BLACKFIRE_SERVER_ID=${BLACKFIRE_SERVER_ID}
            - BLACKFIRE_SERVER_TOKEN=${BLACKFIRE_SERVER_TOKEN}
       networks:
           mynet:
               ipv4_address: 172.25.0.7
...
```

and bring up the image

[107] http://blackfire.io

```
-> docker-compose down
-> docker-compose up -d
```

Upgrade ResetApp Script

./scripts/resetapp is a script that we invoke when we want to remove the cache and reset the database. It is often called if we make changes to the template or before we run test suites. To increase the efficiency of the script, we should allow user to specify if resetting the app requires deleting the cache or not as cache generation is an expensive process and the lag time can cause inconsistency in the tests.

What we need is a an optional switch to allow deleting or cache or not. Maybe even allow an option to load fixtures or not.

```
# scripts/resetapp

#!/bin/bash

usage()
{
cat << EOF

usage: $0 [options]

This script clears the cache, resets the db and install the fixtures

OPTIONS:
   -f      Don't load fixtures
   -c      Don't clear cache (for all env)
EOF
exit 1
}

CLEAR_CACHE=1
LOAD_FIXTURES=1
while getopts "cf" o; do
    case "${o}" in
        c)
            CLEAR_CACHE=
            ;;
        f)
            LOAD_FIXTURES=
            ;;
        *)
            usage
            ;;
    esac
done

if [[ $CLEAR_CACHE ]]
then
    echo "CLEARING CACHE...";
    rm -rf app/cache/*
    # bin/console cache:clear --env=prod --no-warmup
fi

scripts/console doctrine:database:drop --force
scripts/console doctrine:database:create
scripts/console doctrine:schema:create

if [[ $LOAD_FIXTURES ]]
then
    echo "LOADING FIXTURES...";
    scripts/console doctrine:fixtures:load -n
fi

# copy test data over to web folder
cp src/AppBundle/tests/_data/test_profile.jpg web/uploads/profiles/
```

We will now use the “resetapp -c” instead to clear the db only when resetting tests.

```
# scripts/runtest

#!/bin/bash
scripts/resetapp -c
docker-compose exec php vendor/bin/codecept run acceptance $@ -c src/AppBundle
```

Optimising Composer

We can also optimise composer by building an optimised class map to help speed up searching for namespaces. We can run this once during deployment to production.

```
# scripts/optimize_composer

#!/bin/bash

# optimise composer
scripts/composer dump-autoload --optimize --no-dev --classmap-authoritative
```

Minimising JS/CSS

You might have heard of using assetic to manage assets and minimising JS/CSS from The book[108] and The Cookbook[109]. The nice thing about using assetic is that you can do compilation of sass[110] or less[111] files on the fly. If you are unsure about css preprocessor, I recommend checking them out. At the time of writing, sass is more popular.

The has been a lot of innovation in frontend technologies especially with node in recent years. gulpjs[112] is being widely to minify js and css.

Assuming you are using mac, make sure you have homebrew. If not, install it

```
-> ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/mast\
er/install)"
```

Install node if not done.

```
-> brew install node
```

If successful, "node -v" and "npm -v" should return values. Now we create package.json.

```
# in symfony folder
-> npm init
name: (songbird)
version: (1.0.0)
description: gulp config
entry point: (index.js) gulpfile.js
test command:
git repository:
keywords:
author:
license: (ISC)
```

Follow through the prompts. Then install bower.

[108] http://symfony.com/doc/current/cookbook/assetic/asset_management.html
[109] http://symfony.com/doc/current/cookbook/assetic/index.html
[110] http://sass-lang.com
[111] http://lesscss.org
[112] http://gulpjs.com

```
-> sudo npm install -g bower
```

Like npm, let us create the bower.json

```
-> bower init
```

Like before, follow through the prompts. Now, let us install all the bower dependencies.

```
-> bower install jquery bootstrap --save-dev
```

Jquery and bootstrap are the 2 most widely used libraries. It make sense for us to include the libraries outside of AppBundle.

Let us install gulp and all the dependencies.

```
# in symfony folder
-> npm install gulp gulp-util gulp-cat gulp-uglify gulp-uglifycss gulp-less gulp\
-sass gulp-concat gulp-sourcemaps gulp-if --save
```

if everything is successful, we should see these new files and folders:

```
bower.json
/bower_components
package.json
/node_modules
```

We only need the json files, we can put the bower_components and node_modules in .gitignore

```
# .gitignore
...
/node_modules
/bower_components
...
```

package.json is important. We want the default node js file to be gulpfile.js. The package.json should look something like this:

```
# package.json
{
  "name": "songbird",
  "version": "1.0.0",
  "description": "gulp config",
  "main": "gulpfile.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "gulp": "^3.9.1",
    "gulp-cat": "^0.3.3",
    "gulp-concat": "^2.6.0",
    "gulp-if": "^2.0.1",
    "gulp-less": "^3.1.0",
    "gulp-sass": "^2.3.2",
    "gulp-sourcemaps": "^1.6.0",
    "gulp-uglify": "^2.0.0",
    "gulp-uglifycss": "^1.0.6",
    "gulp-util": "^3.0.7"
  }
}
```

Let us create the gulpfile.js

```
# gulpfile.js
var gulp = require('gulp');
var gulpif = require('gulp-if');
var uglify = require('gulp-uglify');
var uglifycss = require('gulp-uglifycss');
var less = require('gulp-less');
var sass = require('gulp-sass');
var concat = require('gulp-concat');
var sourcemaps = require('gulp-sourcemaps');
var exec = require('child_process').exec;

// Minify JS
gulp.task('js', function () {
    return gulp.src(['bower_components/jquery/dist/jquery.js',
        'bower_components/bootstrap/dist/js/bootstrap.js'])
        .pipe(concat('javascript.js'))
        .pipe(uglify())
        .pipe(sourcemaps.write('./'))
        .pipe(gulp.dest('web/minified/js'));
});

// Minify CSS
gulp.task('css', function () {
    return gulp.src([
        'bower_components/bootstrap/dist/css/bootstrap.css',
        'src/AppBundle/Resources/public/less/*.less',
        'src/AppBundle/Resources/public/sass/*.scss',
        'src/AppBundle/Resources/public/css/*.css'])
        .pipe(gulpif(/[.]less/, less()))
        .pipe(gulpif(/[.]scss/, sass()))
        .pipe(concat('styles.css'))
        .pipe(uglifycss())
        .pipe(sourcemaps.write('./'))
        .pipe(gulp.dest('web/minified/css'));
});

// Copy Fonts
gulp.task('fonts', function() {
    return gulp.src('bower_components/bootstrap/fonts/*.{ttf,woff,woff2,eof,svg}\
')
    .pipe(gulp.dest('web/minified/fonts'));
});

gulp.task('installAssets', function() {
    exec('./scripts/console assets:install --symlink', logStdOutAndErr);
});

//define executable tasks when running "gulp" command
gulp.task('default', ['js', 'css', 'fonts', 'installAssets']);

gulp.task('watch', function () {
    var onChange = function (event) {
        console.log('File '+event.path+' has been '+event.type);
    };
    gulp.watch('src/AppBundle/Resources/public/js/*.js', ['default'])
        .on('change', onChange);

    gulp.watch('src/AppBundle/Resources/public/less/*.less', ['default'])
        .on('change', onChange);

    gulp.watch('src/AppBundle/Resources/public/sass/*.scss', ['default'])
        .on('change', onChange);

    gulp.watch('src/AppBundle/Resources/public/css/*.css', ['default'])
        .on('change', onChange);
});

// show exec output
var logStdOutAndErr = function (err, stdout, stderr) {
    console.log(stdout + stderr);
};
```

In short, this gulpfile.js simply says minify all relevant js and css, then copy the js, css and fonts to the web/minified directory.

Since we are only using 1 css and js file, we only need to include the files once in the base template.

```
#  src/AppBundle/Resources/views/base.html.twig
....
{% block stylesheets %}
    <link href="{{ asset('minified/css/styles.css') }}" rel="stylesheet" />
{% endblock %}
...
{% block script %}
    <script src="{{ asset('minified/js/javascript.js') }}"></script>
{% endblock %}
...
```

We no longer need to use separate css for the custom views. Remove all the stylesheet blocks in src/AppBundle/Resources/FOSUserBundle/views/Resetting and src/AppBundle/Resources/FOSUserBundle/views/Security.

Let us update gitignore:

```
# .gitignore
...
/web/minified/
...
```

and create the minified dir

```
mkdir -p web/minified
```

Since we are using bower to include common js and css, we can remove all the unncessary css and js that we have included from the previous chapters.

```
git rm src/AppBundle/Resources/public/css/bootstrap*
```

To compile the js and css, open up another terminal and enter

```
-> gulp
```

if you want to auto compile js or css files when you change the sass or javascript files

```
-> gulp watch
```

If everything is successful, you will see the new dir and files created under web/minified dir.

Now go to songbird.app/login, and verify the new javascript.js and styles.css are included by viewing the source code.

Troubleshooting

You should by now aware of the debug toolbar (profiler) at the bottom of the screen as you access the app_dev.php/* url. The toolbar provide lots of debugging information for the application like the route name, db queries, render time, memory usage, translation…etc.

If you have been observant enough, you should have seen the red alert on the toolbar. Try logging in as admin and go to http://songbird.app:8000/app_dev.php/admin/?entity=User&action=list and look at the toolbar. What happened?

You would see the obvious alert icon in the toolbar… Clicking on the red icon will tell you that you have missing translations.

There will be lots of "messages" under the domain column if there is no translation for certain text.

How would you fix the translation errors?

How about the performance link? What can you see from there?

Using the debug toolbar is straight forward and should be self explanatory.

> Tip: PHP developers should be aware of the print_r or var_dump command to dump objects or variables. Try doing it with Symfony and your browser will crash. In PHP, use var_dumper[113] and in twig, use dump[114] instead.

[113]http://symfony.com/doc/current/components/var_dumper/introduction.html

[114]http://twig.sensiolabs.org/doc/functions/dump.html

Identifying bottlenecks with blackfire.io

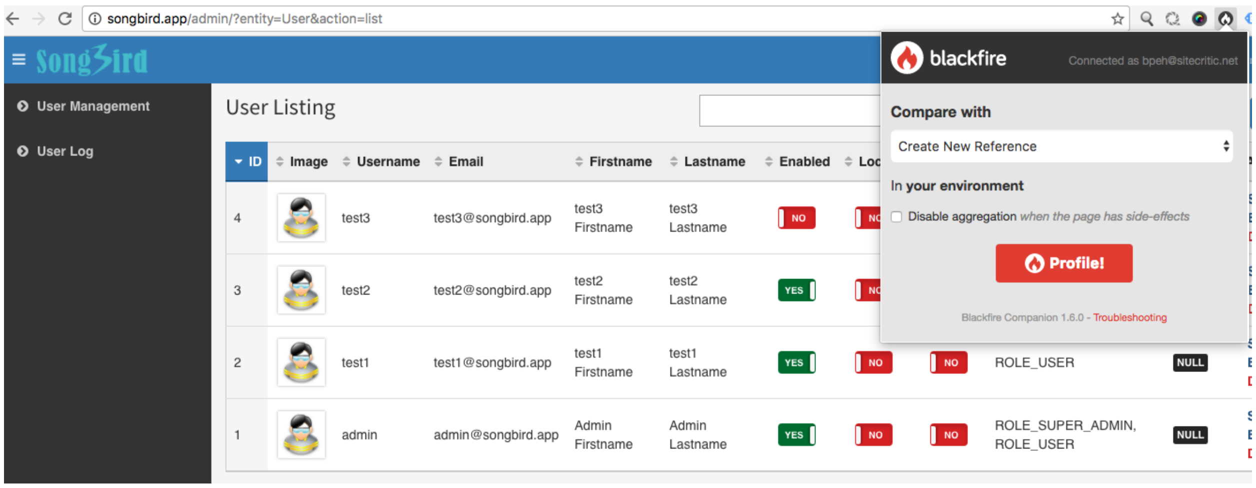
Even though the in-built debug profiler can provide the rendering time and performance information but it doesn't go into detail where the bottlenecks are. To find out where the bottlenecks are, we need Blackfire.

You should have installed blackfire from the previous section.

To make use of Blackfire is easy, install the google chrome companion extension[115].

Once done, you should see a new blackfire icon on the top right of google chrome. Let us load the user management page:

```
http://songbird.app:8000/admin/?entity=User&action=list
```

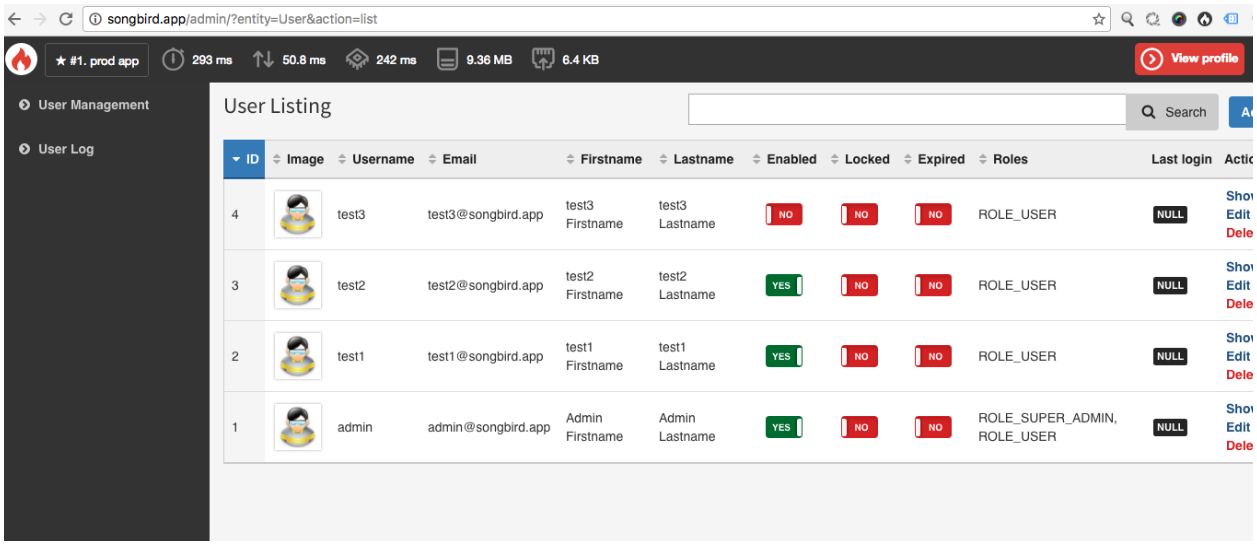

songbird.app/admin/?entity=User&action=list
SongBird
User Management
User Log
User Listing
blackfire
Connected as bpeh@sitecritic.net
Compare with
Create New Reference
In your environment
Disable aggregation when the page has side-effects
Profile!
Blackfire Companion 1.6.0 - Troubleshooting

ID	Image	Username	Email	Firstname	Lastname	Enabled	Loc			
4		test3	test3@songbird.app	test3 Firstname	test3 Lastname	NO	N			
3		test2	test2@songbird.app	test2 Firstname	test2 Lastname	YES	N			
2		test1	test1@songbird.app	test1 Firstname	test1 Lastname	YES	NO	NO	ROLE_USER	NULL
1		admin	admin@songbird.app	Admin Firstname	Admin Lastname	YES	NO	NO	ROLE_SUPER_ADMIN, ROLE_USER	NULL

and click "create a new reference", then click on on the Profile button.

At this point, the chrome browser will interact with the php docker container and tells the blackfire agent to pass the diagnostic data over to blackfire server. You will also see some values in the blackfire toolbar. So we are talking about a few sec of processing time. This is slow and thats because we are using docker.

[115]https://blackfire.io/docs/integrations/chrome


songbird.app/admin/?entity=User&action=list

★ #1. prod app | 293 ms | 50.8 ms | 242 ms | 9.36 MB | 6.4 KB | View profile

User Management

User Log

User Listing

Search

ID	Image	Username	Email	Firstname	Lastname	Enabled	Locked	Expired	Roles	Last login	Actio
4		test3	test3@songbird.app	test3 Firstname	test3 Lastname	NO	NO	NO	ROLE_USER	NULL	Sho Edit Dele
3		test2	test2@songbird.app	test2 Firstname	test2 Lastname	YES	NO	NO	ROLE_USER	NULL	Sho Edit Dele
2		test1	test1@songbird.app	test1 Firstname	test1 Lastname	YES	NO	NO	ROLE_USER	NULL	Sho Edit Dele
1		admin	admin@songbird.app	Admin Firstname	Admin Lastname	YES	NO	NO	ROLE_SUPER_ADMIN, ROLE_USER	NULL	Sho Edit Dele

blackfire profile

Once done, you will see a new profile toolbar. Give the profile a name, say “songbird prod default”.

Reverse Proxy and APCU

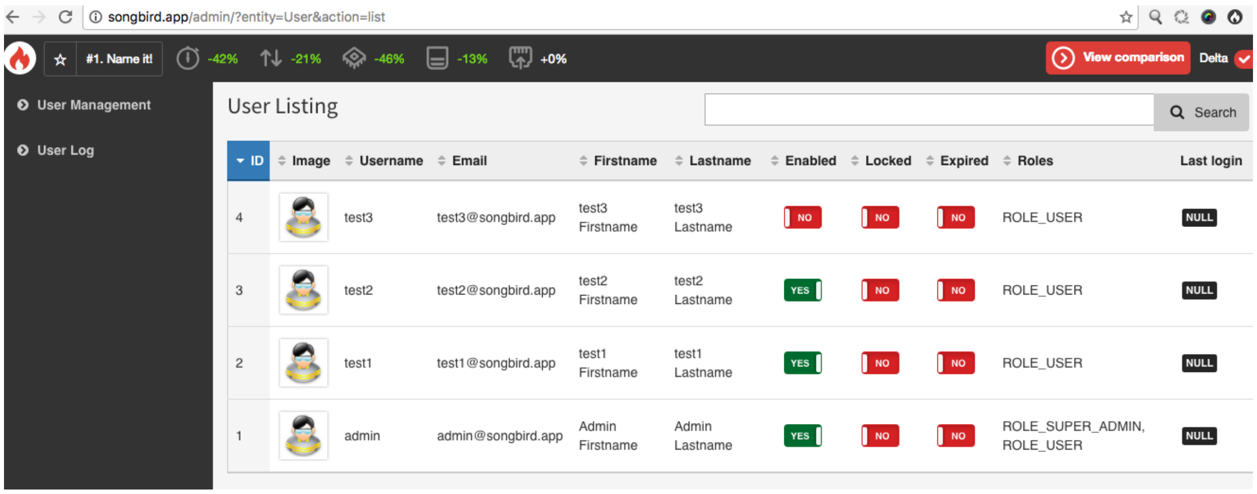
We will do another optimisation. Symfony comes with a reverse proxy[116] and the ability to use apcu[117], let us enable it.

```
# web/app.php

use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\ClassLoader\ApcClassLoader;

/**
 * @var Composer\Autoload\ClassLoader
 */
$loader = require __DIR__.'/../app/autoload.php';
include_once __DIR__.'/../app/bootstrap.php.cache';

$apcLoader = new ApcClassLoader(sha1('songbird'), $loader);
$loader->unregister();
$apcLoader->register(true);

$kernel = new AppKernel('prod', true);
```

[116] https://en.wikipedia.org/wiki/Reverse_proxy
[117] http://php.net/manual/en/book.apcu.php

```
$kernel->loadClassCache();
$kernel = new AppCache($kernel);

// When using the HttpCache, you need to call the method in your front controlle\
r instead of relying on the configuration parameter
Request::enableHttpMethodParameterOverride();
$request = Request::createFromGlobals();
$response = $kernel->handle($request);
$response->send();
$kernel->terminate($request, $response);
```

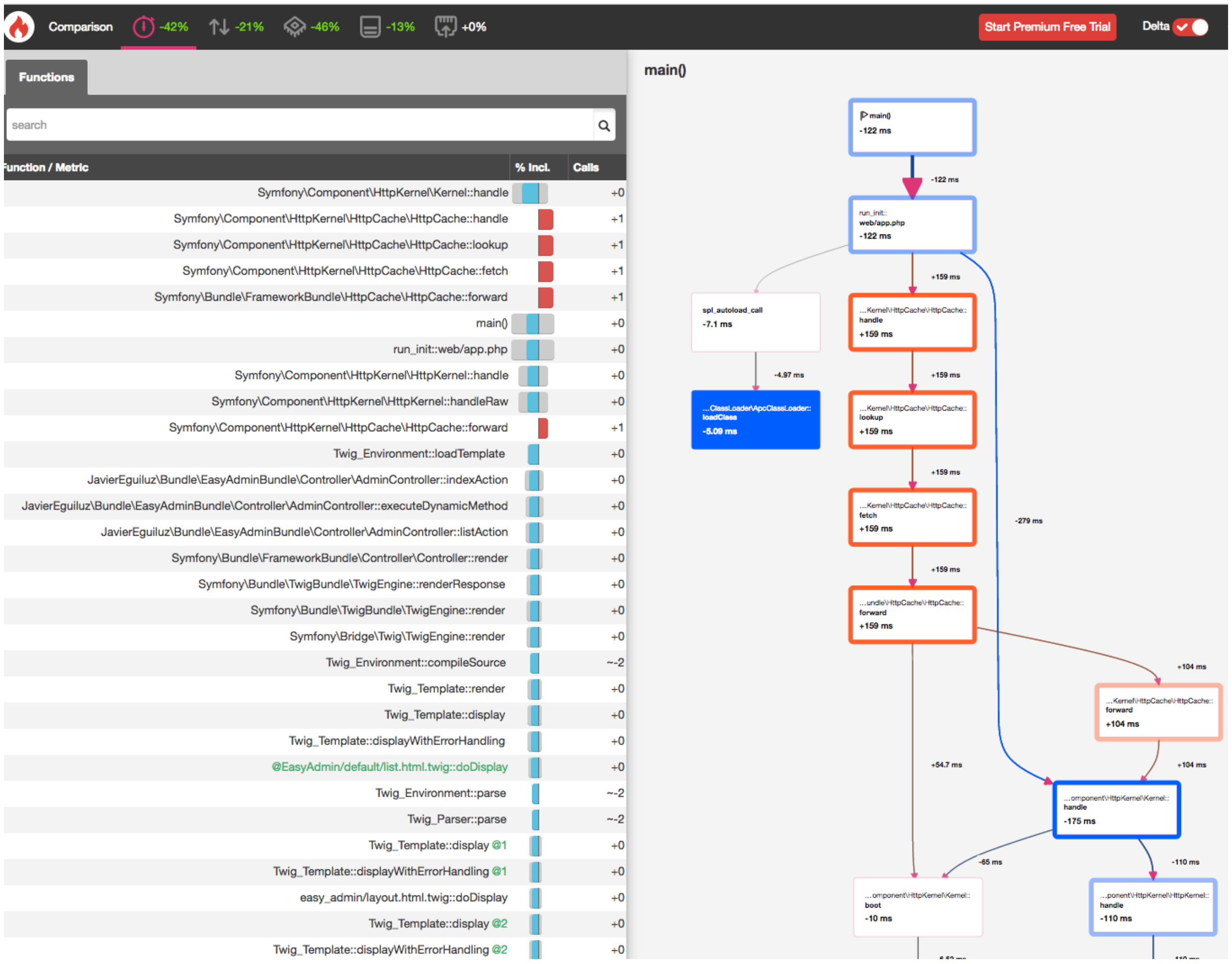
Now refresh the page and then click on the blackfire icon again. In the blackfire toolbar, compare it with the previous profile.



ID	Image	Username	Email	Firstname	Lastname	Enabled	Locked	Expired	Roles	Last login
4		test3	test3@songbird.app	test3 Firstname	test3 Lastname	NO	NO	NO	ROLE_USER	NULL
3		test2	test2@songbird.app	test2 Firstname	test2 Lastname	YES	NO	NO	ROLE_USER	NULL
2		test1	test1@songbird.app	test1 Firstname	test1 Lastname	YES	NO	NO	ROLE_USER	NULL
1		admin	admin@songbird.app	Admin Firstname	Admin Lastname	YES	NO	NO	ROLE_SUPER_ADMIN, ROLE_USER	NULL

Did you see any improvements in the loading time. What was the improvement?

Click on "View comparision"


Comparison
-42%
-21%
-46%
-13%
+0%
Start Premium Free Trial
Delta
Functions
search
main()
main() -122 ms
run_init:: web/app.php -122 ms
spl_autoload_call -7.1 ms
...ClassLoader\ApcClassLoader:: loadClass -5.09 ms
...Kernel\HttpCache\HttpCache:: handle +159 ms
...Kernel\HttpCache\HttpCache:: lookup +159 ms
...Kernel\HttpCache\HttpCache:: fetch +159 ms
...undle\HttpCache\HttpCache:: forward +159 ms
...Kernel\HttpCache\HttpCache:: forward +104 ms
...omponent\HttpKernel\Kernel:: handle -175 ms
...omponent\HttpKernel\Kernel:: boot -10 ms
...ponent\HttpKernel\HttpKernel:: handle -110 ms

Function / Metric	% Incl.	Calls
Symfony\Component\HttpKernel\Kernel::handle	🟦	+0
Symfony\Component\HttpKernel\HttpCache\HttpCache::handle	🟥	+1
Symfony\Component\HttpKernel\HttpCache\HttpCache::lookup	🟥	+1
Symfony\Component\HttpKernel\HttpCache\HttpCache::fetch	🟥	+1
Symfony\Bundle\FrameworkBundle\HttpCache\HttpCache::forward	🟥	+1
main()	🟦	+0
run_init::web/app.php	🟦	+0
Symfony\Component\HttpKernel\HttpKernel::handle	🟦	+0
Symfony\Component\HttpKernel\HttpKernel::handleRaw	🟦	+0
Symfony\Component\HttpKernel\HttpCache\HttpCache::forward	🟥	+1
Twig_Environment::loadTemplate	🟦	+0
JavierEguiluz\Bundle\EasyAdminBundle\Controller\AdminController::indexAction	🟦	+0
JavierEguiluz\Bundle\EasyAdminBundle\Controller\AdminController::executeDynamicMethod	🟦	+0
JavierEguiluz\Bundle\EasyAdminBundle\Controller\AdminController::listAction	🟦	+0
Symfony\Bundle\FrameworkBundle\Controller\Controller::render	🟦	+0
Symfony\Bundle\TwigBundle\TwigEngine::renderResponse	🟦	+0
Symfony\Bundle\TwigBundle\TwigEngine::render	🟦	+0
Symfony\Bridge\Twig\TwigEngine::render	🟦	+0
Twig_Environment::compileSource	🟦	~-2
Twig_Template::render	🟦	+0
Twig_Template::display	🟦	+0
Twig_Template::displayWithErrorHandling	🟦	+0
@EasyAdmin/default/list.html.twig::doDisplay	🟦	+0
Twig_Environment::parse	🟦	~-2
Twig_Parser::parse	🟦	~-2
Twig_Template::display @1	🟦	+0
Twig_Template::displayWithErrorHandling @1	🟦	+0
easy_admin/layout.html.twig::doDisplay	🟦	+0
Twig_Template::display @2	🟦	+0
Twig_Template::displayWithErrorHandling @2	🟦	+0

I was merely scrapping the surface of blackfire. I suggest you do the 24 days of blackfire[118] tutorials if you want to dig in deeper.

Fix Coding Standards with PHP-CS-Fixer

PHP-CS-Fixer[119] automatically fixes coding standards. Its always a good idea to use it to clean up your code before commiting.

```
 -> ./scripts/composer require friendsofphp/php-cs-fixer --dev
```

once this php-cs-fixer is installed, we can use it from the command line like so

[118] https://blackfire.io/docs/24-days/index
[119] https://github.com/FriendsOfPHP/PHP-CS-Fixer

```
-> ./vendor/bin/php-cs-fixer fix app/
      1) AppKernel.php
      2) autoload.php
   Fixed all files in 0.914 seconds, 7.000 MB memory used
```

Let us add the php-cs-fixer cache dir to .gitignore as well

```
# .gitignore
...
.php_cs.cache
```

Do it for the src directory as well. Run all the tests. We can now commit all the fixed files once we are happy with the results.

Summary

In this chapter, we briefly discussed several optimisation strategies. We installed blackfire, minified css and js using gulpjs. We have also refactored the runtest script so that it doesn't clear the cache every time it starts a new test. lastly, we walked through troubleshooting using the web toolbar and blackfire.io.

Exercises

- Using the debug profiler, fix all the translation errors.
- What other performance enhancing tools can you think of?
- Try minimising the js and css in the admin area?

References

- Improving Symfony Performance[120]
- Symfony gateway cache[121]
- 24 days of blackfire[122]

[120]http://symfony.com/doc/current/book/performance.html
[121]http://symfony.com/doc/current/book/http_cache.html
[122]https://blackfire.io/docs/24-days/index

Chapter 17: The Page Manager Part 1

So far, we have been very lazy (a good thing?). We have offloaded bulk of the CMS functionality to FOS and EasyAdmin bundles. In this chapter, we will create a simple **reusable** page bundle and the bulk of the logic ourselves. Let us call this NestablePageBundle.

The Plan

We want our page bundle to have no dependency on other bundles like FOSUserbundle. Each page should have a unique slug and a couple of meta data such as title, short description, long description, created_date…etc. We will be using nestable js[123] to allow drag and drop + page nesting using ajax.

We will create 2 entities. The first entity is the Page entity and will consist of simple attributes like id, slug, sequence, parent and children id…etc. The second entity will be the PageMeta entity consisting of attributes like name, locale, title, short description and content. The relationship between the Page and PageMeta entity will be one to many.

This bundle creation is for **illustration only** and has lots of rooms for improvement.

Define User Stories

Since SongbirdNestableBundle is going to be decoupled from AppBundle, so we will need 2 sets of tests, one for SongbirdNestableBundle and one for the AppBundle. We will worry about the AppBundle tests in the next chapter.

SongbirdNestableBundle

Story Id	As a	I	So that I
17.1	test2 user	want to manage pages	can update them anytime.

Story ID 17.1: As test2 user, I want to manage pages, so that I can update them anytime.

[123] https://github.com/bernardpeh/Nestable

Scenario Id	Given	When	Then
17.11	List Pages	I go to /songbird_page	I should see the why_songbird slug under the about slug
17.12	Show contact us page	I go to /songbird_page/5	I should see the word “contact_us” and the word “Created”
17.13	Reorder home	I simulate a drag and drop of the home menu to under the about menu and submit the post data to /songbird_-page/reorder	I should see “reordered successfully message” in the response and menus should be updated
17.14	Edit home page meta	I go to edit homepage url and update the menu title of “Home” to “Home1” and click update	I should see the text “successfully updated” message
17.15	Create and delete test pagemeta	go to /new and fill in details and click “Create” button, then go to test page and click add new meta and fill in the details and click “create” button, then click delete button	I should see the new page and pagemeta being created and pagemeta deleted
17.16	Delete contact us page	go to /songbird_page/5 and click “Delete” button	I should see the contact_us slug no longer available in the listing page. Page id 5 should no longer be found in the pagemeta table.

Create Our Own Bundle Generation Script (Optional)

The default bundle generation script is cool. Let us customise it further to make our life easier. We will create a custom script to generate songbird bundles.

```
# scripts/createbundle

#!/bin/bash

if [ -z "$*" ]; then
        echo -e "\nUsage: $0 VendorName BundleName\n";
        exit;
fi

# using symfony bundle generation script is a quick way to generate bundles but \
doesn't mean its the best way.
scripts/console generate:bundle --namespace=$1/$2 --dir=src --bundle-name=$1$2 -\
-format=annotation --no-interaction
rm -rf src/$1/$2/Tests/*
rm -rf src/$1/$2/Resources/views/Default
rm src/$1/$2/Controller/DefaultController.php

touch src/$1/$2/Resources/views/.gitkeep
touch src/$1/$2/Controller/.gitkeep
```

now let us run the script

```
-> chmod u+x ./scripts/createbundle
-> ./scripts/createbundle Songbird NestablePageBundle
```

run a git status and you will see that the script does a lot of work for you.

```
-> git status

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   app/AppKernel.php
        modified:   app/config/routing.yml
        modified:   app/config/config.yml

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        scripts/createbundle
        src/Songbird
```

Implementation

Let us create the entities.

For Page entity:

```
-> ./scripts/console generate:doctrine:entity --entity=SongbirdNestablePageBundl\
e:Page --format=annotation --fields="slug:string(length=255 unique=true) isPubli\
shed:boolean(nullable=true) sequence:integer(nullable=true) modified:datetime cr\
eated:datetime" --no-interaction
```

and for PageMeta entity:

```
-> ./scripts/console generate:doctrine:entity --entity=SongbirdNestablePageBundl\
e:PageMeta --format=annotation --fields="page_title:string(length=255) menu_titl\
e:string(255) locale:string(4) short_description:text(nullable=true) content:tex\
t(nullable=true)" --no-interaction
```

We now need to update the relationship between the 2 entities:

```
# src/Songbird/NestablePageBundle/Entity/Page.php

namespace Songbird\NestablePageBundle\Entity;
use Doctrine\ORM\Mapping as ORM;

/**
 * Page
 *
 * @ORM\Table(name="page")
 * @ORM\Entity(repositoryClass="Songbird\NestablePageBundle\Entity\PageRepositor\
y")
 * @ORM\HasLifecycleCallbacks()
 */
class Page
{
...
    /**
     * @var string
     *
     * @ORM\Column(name="slug", type="string", length=255, unique=true)
     */
    private $slug;

    /**
     * @ORM\ManyToOne(targetEntity="Page", inversedBy="children")
     * @ORM\JoinColumn(name="parent_id", referencedColumnName="id", onDelete="CA\
SCADE")}
     * @ORM\OrderBy({"sequence" = "ASC"})
     */
    private $parent;

    /**
     * @var boolean
     *
     * @ORM\Column(name="isPublished", type="boolean", nullable=true)
     */
    private $isPublished;

    /**
     * @var integer
     *
     * @ORM\Column(name="sequence", type="integer", nullable=true)
     */
    private $sequence;

    /**
     * @ORM\OneToMany(targetEntity="Page", mappedBy="parent")
     * @ORM\OrderBy({"sequence" = "ASC"})
     */
    private $children;

    /**
     * @ORM\OneToMany(targetEntity="Songbird\NestablePageBundle\Entity\PageMeta"\
, mappedBy="page", cascade={"persist"})
     */
    private $pageMetas;

    ...

    /**
     * @ORM\PrePersist
     */
    public function prePersist()
    {
        // update the modified time
        $this->setModified(new \DateTime());

        // for newly created entries
        if ($this->getCreated() == null) {
            $this->setCreated(new \DateTime('now'));
        }
        $this->created = new \DateTime();
    }

    /**
     * convert obj to string
     *
     * @return string
     */
    public function __toString() {
        return $this->slug;
    }
...
```

and

```
# src/Songbird/NestablePageBundle/Entity/PageMeta.php
...
    /**
     * @ORM\ManyToOne(targetEntity="Songbird\NestablePageBundle\Entity\Page", in\
versedBy="pageMetas")
     * @ORM\JoinColumn(name="page_id", referencedColumnName="id", onDelete="CASC\
ADE")}
     */
    private $page;
    ...
    /**
     * @var string
     *
     * @ORM\Column(name="short_description", type="text", nullable=true)
     */
    private $short_description;

    /**
     * @var string
     *
     * @ORM\Column(name="content", type="text", nullable=true)
     */
    private $content;
    ...
    /**
     * constructor
     */
    public function __construct()
    {
        // default values
        $this->locale = 'en';
    }
    ...
    public function newAction(Request $request)
    {
        // Noticed that CRUD tries to be intelligent and change pageMeta to page\
Metum but its not really that intelligent
        $pageMetum = new PageMeta();
```

There were some new doctrine association annotations[124] used here, notably @ManyToOne and @OneToMany are the most common. Establishing the right associations can save lots of time when managing table relationships. For PageMeta.php, we set the default locale to "en" if none is set.

We can now auto generate the stubs for the 2 entities:

```
-> ./scripts/console generate:doctrine:entities SongbirdNestablePageBundle --no-\
backup
Generating entities for bundle "SongbirdNestablePageBundle"
  > generating Songbird\NestablePageBundle\Entity\Page
  > generating Songbird\NestablePageBundle\Entity\PageMeta
```

This command helps us to generate the getters and setters for the new variables that we have added. For the page entity for example, you should see new functions like setParent() and getParent() being added - another huge time saver.

We will also create a helper to help us find the page meta entries based on locale.

[124] http://docs.doctrine-project.org/projects/doctrine-orm/en/latest/reference/association-mapping.html

```
# src/Songbird/NestablePageBundle/Entity/PageRepository.php

namespace Songbird\NestablePageBundle\Repository;

/**
 * PageRepository
 *
 * This class was generated by the Doctrine ORM. Add your own custom
 * repository methods below.
 */
class PageRepository extends EntityRepository
{

    public function findPageMetaByLocale($slug, $locale) {

        $query = $this->createQueryBuilder('p')
            ->select('p', 'pm')
            ->Join('p.pageMetas','pm')
            ->where('p.isPublished = :isPublished')
            ->andWhere('pm.locale = :locale')
            ->andWhere('p.slug = :slug')
            ->setParameter('isPublished', '1')
            ->setParameter('locale', $locale)
            ->setParameter('slug', $slug)
            ->getQuery();

        return $query->getOneOrNullResult();

    }

    public function findParent() {

        $query = $this->createQueryBuilder('p')
            ->select('p')
            ->where('p.isPublished = :isPublished')
            ->andWhere('p.parent is null')
            ->setParameter('isPublished', '1')
            ->orderBy('p.sequence', 'asc')
            ->getQuery();

        return $query->getResult();

    }
}
```

Before we reset the app, let us create the doctrine migration file so that we can deploy this db changes to production (if we have one). It is a good practice to do that.

```
-> ./scripts/console doctrine:migrations:diff
```

reset the app now and verify that the 2 new tables, ie page and page_meta being created in the songbird db.

```
-> ./scripts/resetapp
```

We are going to use a variant of nestable.js[125] to create our draggable menu. Let us create the js and css directories.

```
-> mkdir -p src/Songbird/NestablePageBundle/Resources/public/{js,css}
```

Download jquery.nestable.js and put it under src/Songbird/NestablePageBundle/Resources/public/js/jquery.nestable.js

```
-> cd src/Songbird/NestablePageBundle/Resources/public/js
-> wget http://code.jquery.com/jquery-1.11.3.min.js
-> wget https://raw.githubusercontent.com/bernardpeh/Nestable/master/jquery.nest\
able.js
```

Now let us create the css

```
# src/Songbird/NestablePageBundle/Resources/public/css/styles.css
.dd { position: relative; display: block; margin: 0; padding: 0; max-width: 600p\
x; list-style: none; font-siz
e: 13px; line-height: 20px; }

.dd-list { display: block; position: relative; margin: 0; padding: 0; list-style\
: none; }
.dd-list .dd-list { padding-left: 30px; }
.dd-collapsed .dd-list { display: none; }

.dd-item,
```

[125] https://github.com/bernardpeh/Nestable

```
.dd-empty,
.dd-placeholder { display: block; position: relative; margin: 0; padding: 0; min\
-height: 20px; font-size: 13p
x; line-height: 20px; }

.dd-handle { display: block; height: 30px; margin: 5px 0; padding: 5px 10px; col\
or: #333; text-decoration: no
ne; font-weight: bold; border: 1px solid #ccc;
    background: #fafafa;
    background: -webkit-linear-gradient(top, #fafafa 0%, #eee 100%);
    background:    -moz-linear-gradient(top, #fafafa 0%, #eee 100%);
    background:         linear-gradient(top, #fafafa 0%, #eee 100%);
    -webkit-border-radius: 3px;
            border-radius: 3px;
    box-sizing: border-box; -moz-box-sizing: border-box;
}
.dd-handle:hover { color: #2ea8e5; background: #fff; }
.dd-item > button { display: block; position: relative; cursor: pointer; float: \
left; width: 25px; height: 20px; margin: 5px 0; padding: 0; text-indent: 100%; w\
hite-space: nowrap; overflow: hidden; border: 0; background: transparent; font-s\
ize: 12px; line-height: 1; text-align: center; font-weight: bold; }
.dd-item > button:before { content: '+'; display: block; position: absolute; wid\
th: 100%; text-align: center; text-indent: 0; }
.dd-item > button[data-action="collapse"]:before { content: '-'; }

.dd-placeholder,
.dd-empty { margin: 5px 0; padding: 0; min-height: 30px; background: #f2fbff; bo\
rder: 1px dashed #b6bcbf; box-sizing: border-box; -moz-box-sizing: border-box; }
.dd-empty { border: 1px dashed #bbb; min-height: 100px; background-color: #e5e5e\
5;
    background-image: -webkit-linear-gradient(45deg, #fff 25%, transparent 25%, \
transparent 75%, #fff 75%, #fff),
                      -webkit-linear-gradient(45deg, #fff 25%, transparent 25%, \
transparent 75%, #fff 75%, #fff);
    background-image:    -moz-linear-gradient(45deg, #fff 25%, transparent 25%, \
transparent 75%, #fff 75%, #fff),
                         -moz-linear-gradient(45deg, #fff 25%, transparent 25%, \
transparent 75%, #fff 75%, #fff);
    background-image:         linear-gradient(45deg, #fff 25%, transparent 25%, \
transparent 75%, #fff 75%, #fff),
                              linear-gradient(45deg, #fff 25%, transparent 25%, \
transparent 75%, #fff 75%, #fff);
```

```
    background-size: 60px 60px;
    background-position: 0 0, 30px 30px;
}

.dd-dragel { position: absolute; pointer-events: none; z-index: 9999; }
.dd-dragel > .dd-item .dd-handle { margin-top: 0; }
.dd-dragel .dd-handle {
    -webkit-box-shadow: 2px 4px 6px 0 rgba(0,0,0,.1);
            box-shadow: 2px 4px 6px 0 rgba(0,0,0,.1);
}
```

Let us now create the translation files.

The english version:

```
# src/Songbird/NestablePageBundle/Resources/translations/SongbirdNestablePageBun\
dle.en.xlf
<?xml version="1.0"?>
<xliff version="1.2" xmlns="urn:oasis:names:tc:xliff:document:1.2">
    <file source-language="en" datatype="plaintext" original="file.ext">
        <body>
            <trans-unit id="1">
                <source>menu.page_management</source>
                <target>Page Management</target>
            </trans-unit>
            <trans-unit id="2">
                <source>flash_reorder_instructions</source>
                <target>click and drag to reorder menu</target>
            </trans-unit>
            <trans-unit id="3">
                <source>flash_reorder_edit_success</source>
                <target>menu has been reordered successfully</target>
            </trans-unit>
        </body>
    </file>
</xliff>
```

and the french version:

```
# src/Songbird/NestablePageBundle/Resources/translations/SongbirdNestablePageBun\
dle.fr.xlf
<?xml version="1.0"?>
<xliff version="1.2" xmlns="urn:oasis:names:tc:xliff:document:1.2">
    <file source-language="en" datatype="plaintext" original="file.ext">
        <body>
            <trans-unit id="1">
                <source>menu.page_management</source>
                <target>Gestion de la page</target>
            </trans-unit>
            <trans-unit id="2">
                <source>flash_reorder_instructions</source>
                <target>cliquer et faire glisser pour réorganiser le menu</targe\
t>
            </trans-unit>
            <trans-unit id="3">
                <source>flash_reorder_edit_success</source>
                <target>menu a été réorganisé avec succès</target>
            </trans-unit>
        </body>
    </file>
</xliff>
```

We will now generate CRUD for the 2 entities in a quick way:

```
# dont memorise this. You can get help using the --help option

-> ./scripts/console g:doctrine:crud --entity=SongbirdNestablePageBundle:Page --\
route-prefix=songbird_page --with-write -n
-> ./scripts/console g:doctrine:crud --entity=SongbirdNestablePageBundle:PageMet\
a --route-prefix=songbird_pagemeta --with-write -n
```

Noticed we use "g" as a shortcut to "generate" in the command line. We've added the route-prefix to make sure our path is unique so that it can be reused with minimal changes.

Create Sample Data

Let us populate sample data to work with. Say we want 3 parent menu, Homepage, "About Us" and "Contact Us" and a couple of submenus.

```
# src/Songbird/NestablePageBundle/DataFixtures/ORM/LoadPageData.php

namespace Songbird\NestablePageBundle\DataFixtures\ORM;

use Doctrine\Common\DataFixtures\AbstractFixture;
use Doctrine\Common\Persistence\ObjectManager;
use Symfony\Component\DependencyInjection\ContainerAwareInterface;
use Symfony\Component\DependencyInjection\ContainerInterface;
use Songbird\NestablePageBundle\Entity\Page;
use Songbird\NestablePageBundle\Entity\PageMeta;

class LoadPageData extends AbstractFixture implements ContainerAwareInterface
{

    /**
     * @var ContainerInterface
     */
    private $container;

    /**
     * {@inheritDoc}
     */
    public function setContainer(ContainerInterface $container = null)
    {
        $this->container = $container;
    }

    /**
     * {@inheritDoc}
     */
    public function load(ObjectManager $manager)
    {

        $homepage = new Page();
        $homepage->setSlug('home');
        $homepage->setIsPublished(1);
        $homepage->setSequence(0);
        // there is no relationship with the user entity atm
        // $homepage->setUser($this->getReference('admin_user'));
        $manager->persist($homepage);

        $homemetaEN = new PageMeta();
        $homemetaEN->setPage($homepage);
        $homemetaEN->setMenuTitle('Home');
        $homemetaEN->setPageTitle('Welcome to SongBird CMS Demo');
        $homemetaEN->setShortDescription('Welcome to SongBird CMS Demo');
        $homemetaEN->setContent('<p>SongBird is a simple CMS built with popular \
bundles like FOSUserBundle and SonataAdminBundle.
            The CMS is meant to showcase Rapid Application Development with Symf\
ony.</p>');
        $manager->persist($homemetaEN);

        $homemetaFR = new PageMeta();
        $homemetaFR->setPage($homepage);
        $homemetaFR->setMenuTitle('Accueil');
        $homemetaFR->setPageTitle('Bienvenue a SongBird CMS Démo');
        $homemetaFR->setShortDescription('Bienvenue a SongBird CMS Démo');
        $homemetaFR->setLocale('fr');
        $homemetaFR->setContent('<p>SongBird est un simple CMS construit avec de\
s faisceaux populaires comme FOSUserBundle et SonataAdminBundle.
            Le CMS est destinée à mettre en valeur Rapid Application Development\
 avec Symfony .</p>');
        $manager->persist($homemetaFR);

        $aboutpage = new Page();
        $aboutpage->setSlug('about');
        $aboutpage->setIsPublished(1);
        $aboutpage->setSequence(1);
        $manager->persist($aboutpage);

        $aboutmetaEN = new PageMeta();
        $aboutmetaEN->setPage($aboutpage);
        $aboutmetaEN->setMenuTitle('About');
        $aboutmetaEN->setPageTitle('About SongBird');
        $aboutmetaEN->setShortDescription('What is Songbird?');
        $aboutmetaEN->setContent('<p>SongBird is a simple CMS (Content Managemen\
t System) consisting the following features:</p>
        <ul>
        <li>Admin Panel and Dashboard - A password protected administration area\
 for administrators and users.</li>
        <li>User Management System - For administrators to manage the users of t\
he site.</li>
        <li>Multi-lingual Capability - No CMS is complete without this.</li>
        <li>Page Management System - For managing the front-end pages of the sit\
e.</li>
        <li>Media Management System - For administrators and users to manage fil\
es and images.</li>
        <li>Frontend - The frontend of the website.</li>
        </ul>');
        $manager->persist($aboutmetaEN);

        $aboutmetaFR = new PageMeta();
        $aboutmetaFR->setPage($aboutpage);
        $aboutmetaFR->setLocale('fr');
        $aboutmetaFR->setMenuTitle('Sur');
        $aboutmetaFR->setPageTitle('Sur SongBird');
        $aboutmetaFR->setShortDescription('Qu\'est-ce que SongBird?');
        $aboutmetaFR->setContent('<p>SongBird est un simple CMS ( Content Manage\
ment System ) comprenant les caractéristiques suivantes:</p>
        <ul>
        <li>Panneau d\'administration et Dashboard - Un mot de passe protégé esp\
ace d\'administration pour les administrateurs et les utilisateurs.</li>
        <li>Système de gestion de l\'utilisateur - Pour les administrateurs de g\
érer les utilisateurs du site.</li>
        <li>Capacité multilingue - Pas de CMS est complète sans cela.</li>
        <li>Système de Management de la page - Pour gérer les pages du site fron\
taux.</li>
        <li>Système de Gestion des médias - Pour les administrateurs et les util\
isateurs de gérer des fichiers et des images.</li>
        <li>Frontend - L\'interface du site.</li>
        </ul>');
        $manager->persist($aboutmetaFR);


        $whypage = new Page();
        $whypage->setSlug('why_songbird');
        $whypage->setIsPublished(1);
        $whypage->setSequence(0);
        $whypage->setParent($aboutpage);
        $manager->persist($whypage);

        $whymetaEN = new PageMeta();
        $whymetaEN->setPage($whypage);
        $whymetaEN->setMenuTitle('Why Songbird');
        $whymetaEN->setPageTitle('Why Songbird?');
        $whymetaEN->setShortDescription('Why Another CMS?');
        $whymetaEN->setContent('<p>Learning a modern day framework is not an eas\
y task. Songbird CMS does not aim to replace any existing CMS out there.
        To put it simply, it is a play ground for people who wants to learn Symf\
ony by building a CMS from scratch.
        Creating a semi-complex application like a CMS will give the coder insig\
hts in building bigger
        things with a RAD framework like Symfony.</p>');
        $manager->persist($whymetaEN);

        $whymetaFR = new PageMeta();
        $whymetaFR->setPage($whypage);
        $whymetaFR->setMenuTitle('pourquoi SongBird');
        $whymetaFR->setPageTitle('pourquoi SongBird?');
        $whymetaFR->setShortDescription('Pourquoi un autre CMS');
        $whymetaFR->setContent('<p>Apprendre un cadre moderne est pas une tâche \
facile . Songbird CMS ne vise pas à remplacer tout CMS existant là-bas.
        Pour dire les choses simplement , il est un terrain de jeu pour les gens\
 qui veulent apprendre symfony en construisant un CMS à partir de zéro.
        Création d\'une application semi- complexe comme un CMS donnera les idée\
s de codeur dans la construction de plus
        les choses avec un cadre RAD comme Symfony</p>');
        $whymetaFR->setLocale('fr');
        $manager->persist($whymetaFR);

        $planpage = new Page();
        $planpage->setSlug('documentation');
        $planpage->setIsPublished(1);
        $planpage->setSequence(1);
        $planpage->setParent($aboutpage);
        $manager->persist($planpage);

        $planmetaEn = new PageMeta();
        $planmetaEn->setPage($planpage);
        $planmetaEn->setMenuTitle('Where do I start');
        $planmetaEn->setPageTitle('Where do I start?');
        $planmetaEn->setShortDescription('Where Do I Start?');
        $planmetaEn->setContent('<p>I recommend reading the online documentation\
 at <a href="https://leanpub.com/practicalsymfony3">leanpub</a></p>
            <p>git clone the repo. Read and Code at the same time. I believe tha\
t is the most effective way to learn.</p>');
        $manager->persist($planmetaEn);

        $planmetaFR = new PageMeta();
        $planmetaFR->setPage($planpage);
        $planmetaFR->setLocale('fr');
        $planmetaFR->setMenuTitle('Où est-ce que je commence');
        $planmetaFR->setPageTitle('Où est-ce que je commence?');
        $planmetaFR->setShortDescription('Où est-ce que je commence?');
        $planmetaFR->setContent('<p>Je recommande la lecture de la documentation\
 en ligne à <a href="https://leanpub.com/practicalsymfony3">leanpub</a></p>
            <p>git clone the repo. Lire et code en même temps . Je crois que la \
façon la plus efficace d\'apprendre.</p>');
        $manager->persist($planmetaFR);

        $contactpage = new Page();
        $contactpage->setSlug('contact_us');
        $contactpage->setIsPublished(1);
        $contactpage->setSequence(2);
        $manager->persist($contactpage);

        $contactmetaEN = new PageMeta();
        $contactmetaEN->setPage($contactpage);
        $contactmetaEN->setPageTitle('Contact Us');
        $contactmetaEN->setMenuTitle('Contact');
        $contactmetaEN->setShortDescription('Contact');
        $contactmetaEN->setContent('<p>I hope Songbird can be beneficial to anyo\
ne who aspires to learn Symfony.</p>
            <p>This project is hosted in <a href="https://github.com/bernardpeh/\
songbird" target="_blank">github</a>.</p>
            <p>To make this CMS a better learning platform for everyone, feel fr\
ee to update the code and create a pull request in github.</p>');
        $manager->persist($contactmetaEN);

        $contactmetaFR = new PageMeta();
        $contactmetaFR->setPage($contactpage);
        $contactmetaFR->setLocale('fr');
        $contactmetaFR->setPageTitle('Contactez nous');
        $contactmetaFR->setMenuTitle('Contact');
        $contactmetaFR->setShortDescription('Contact');
        $contactmetaFR->setContent('<p>Je l\'espère Songbird peut être bénéfique\
 pour toute personne qui aspire à apprendre symfony.</p>
            <p>Ce projet est hébergé dans <a href="https://github.com/bernardpeh\
/songbird" target="_blank">github</a>.</p>
            <p>Pour faire ce CMS une meilleure plateforme d\'apprentissage pour \
tout le monde , vous pouvez mettre à jour le code et créer une demande de tracti\
on dans github.</p>');
        $manager->persist($contactmetaFR);

        // now save all
        $manager->flush();
    }

}
```

reset the app to load the fixtures and check that the entries have been added to the db.

```
-> ./scripts/resetapp
```

Now go to the page url and you should see the default crud template

```
http://songbird.app:8000/app_dev.php/songbird_page/
```

Everything is looking plain at the moment, let us integrate nestablejs.

Integrating NestableJS

How do we integrate NestableJS to our bundle? The secret will be in the Page Controller. We will change the logic there.

```
# src/Songbird/NestablePageBundle/Controller/PageController.php

namespace Songbird\NestablePageBundle\Controller;

use Songbird\NestablePageBundle\Entity\Page;
use Symfony\Bundle\FrameworkBundle\Controller\Controller;
use Sensio\Bundle\FrameworkExtraBundle\Configuration\Method;
use Sensio\Bundle\FrameworkExtraBundle\Configuration\Route;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\JsonResponse;

/**
 * Page controller.
 *
 * @Route("/songbird_page")
 */
class PageController extends Controller
{

    /**
     * Lists all Page entities.
     *
     * @Route("/", name="songbird_page_index")
     * @Method("GET")
     */
    public function indexAction()
    {
        return $this->redirect($this->generateUrl('songbird_page_list'));
    }

    /**
     * Lists all nested page
     *
     * @Route("/list", name="songbird_page_list")
     * @Method("GET")
     */
    public function listAction()
    {
        $em = $this->getDoctrine()->getManager();
        $rootMenuItems = $em->getRepository('SongbirdNestablePageBundle:Page')->\
findParent();

        return $this->render('page/list.html.twig', array(
            'tree' => $rootMenuItems,
        ));
    }

    /**
     * reorder pages
     *
     * @Route("/reorder", name="songbird_page_reorder")
     * @Method("POST")
     */
    public function reorderAction(Request $request)
    {
        $em = $this->getDoctrine()->getManager();
        // id of affected element
        $id = $request->get('id');
        // parent Id
        $parentId = ($request->get('parentId') == '') ? null : $request->get('pa\
rentId');
        // new sequence of this element. 0 means first element.
        $position = $request->get('position');

        $result = $em->getRepository('SongbirdNestablePageBundle:Page')->reorder\
Element($id, $parentId, $position);

        return new JsonResponse(
            array('message' => $this->get('translator')->trans($result[0], array\
(), 'SongbirdNestablePageBundle')
            , 'success' => $result[1])
        );
    }

    /**
     * Creates a new Page entity.
     *
     * @Route("/new", name="songbird_page_new")
     * @Method({"GET", "POST"})
     */
    public function newAction(Request $request)
    {
        $page = new Page();
        $form = $this->createForm('Songbird\NestablePageBundle\Form\PageType', $\
page);
        $form->handleRequest($request);

        if ($form->isSubmitted() && $form->isValid()) {
            $em = $this->getDoctrine()->getManager();
            $em->persist($page);
            $em->flush();

            return $this->redirectToRoute('songbird_page_show', array('id' => $p\
age->getId()));
        }

        return $this->render('page/new.html.twig', array(
            'page' => $page,
            'form' => $form->createView(),
        ));
    }

    /**
     * Finds and displays a Page entity.
     *
     * @Route("/{id}", name="songbird_page_show")
     * @Method("GET")
     */
    public function showAction(Request $request, Page $page)
    {
        $em = $this->getDoctrine()->getManager();

        $pageMeta = $em->getRepository('SongbirdNestablePageBundle:PageMeta')->f\
indPageMetaByLocale($page,$request->getLocale());

        $deleteForm = $this->createDeleteForm($page);

        return $this->render('page/show.html.twig', array(
            'page' => $page,
            'pageMeta' => $pageMeta,
            'delete_form' => $deleteForm->createView(),
        ));
    }

    /**
     * Displays a form to edit an existing Page entity.
     *
     * @Route("/{id}/edit", name="songbird_page_edit")
     * @Method({"GET", "POST"})
     */
    public function editAction(Request $request, Page $page)
    {
        $deleteForm = $this->createDeleteForm($page);
        $editForm = $this->createForm('Songbird\NestablePageBundle\Form\PageType\
', $page);
        $editForm->handleRequest($request);

        if ($editForm->isSubmitted() && $editForm->isValid()) {
            $em = $this->getDoctrine()->getManager();
            $em->persist($page);
            $em->flush();

            return $this->redirectToRoute('songbird_page_edit', array('id' => $p\
age->getId()));
        }

        return $this->render('page/edit.html.twig', array(
            'page' => $page,
            'edit_form' => $editForm->createView(),
            'delete_form' => $deleteForm->createView(),
        ));
    }

    /**
     * Deletes a Page entity.
     *
     * @Route("/{id}", name="songbird_page_delete")
     * @Method("DELETE")
     */
    public function deleteAction(Request $request, Page $page)
    {
        $form = $this->createDeleteForm($page);
        $form->handleRequest($request);

        if ($form->isSubmitted() && $form->isValid()) {
            $em = $this->getDoctrine()->getManager();
            $em->remove($page);
            $em->flush();
        }

        return $this->redirectToRoute('songbird_page_index');
    }

    /**
     * Creates a form to delete a Page entity.
     *
     * @param Page $page The Page entity
     *
     * @return \Symfony\Component\Form\Form The form
     */
    private function createDeleteForm(Page $page)
    {
        return $this->createFormBuilder()
```

```
            ->setAction($this->generateUrl('songbird_page_delete', array('id' =>\
 $page->getId())))
            ->setMethod('DELETE')
            ->getForm()
            ;
    }
}
```

We have added 2 extra methods, listAction and reorderAction. As the controller should have minimum logic, we have moved the bulk of reorderAction logic to the repository.

```
# src/Songbird/NestablePageBundle/Entity/PageRepository.php
...
    /**
     * reorder element based on user input
     * @param  int $id       id of element dragged
     * @param  int $parentId parent id
     * @param  int $position new position relative to parent id. 0 is first posi\
tion
     * @return array          array([string] message, [boolean] success)
     */
    public function reorderElement($id, $parentId, $position)
    {
        // step 1: get all siblings based on old location. update the seq
        $old_item = $this->findOneById($id);

        if ($old_item === null) {
            $old_parent_id = '';
        }
        else {
            $old_parent_id = ($old_item->getParent() === null) ? '' : $old_item-\
>getParent()->getId();
        }


        // if old parent and new parent is the same, user moving in same level.
        // dont need to update old parent
        if ($old_parent_id != $parentId) {
            $old_children = $this->findBy(
                array('parent' => $old_parent_id),
                array('sequence' => 'ASC')
                );
            $seq = 0;

            foreach ($old_children as $oc) {
                $or = $this->findOneById($oc->getId());
                if ($old_item->getSequence() != $or->getSequence()) {
                    $or->setSequence($seq);
                    $this->getEntityManager()->persist($or);
                    $seq++;
                }
            }
        }

        $new_children = $this->findBy(
            array('parent' => $parentId),
            array('sequence' => 'ASC')
            );
        $seq = 0;

        $ir = $this->findOneById($id);


        if (!is_null($parentId)) {
            $parent = $this->findOneById($parentId);
            if ($ir !== null) {
                $ir->setParent($parent);
            }
        }
        else {
            if ($ir !== null) {
                $ir->setParent();
            }
        }
        foreach ($new_children as $nc) {
            // if the id is the same, it means user moves in same level

            if ($old_parent_id == $parentId) {
                // if in same level, we just need to swap position
                // get id of element with the current position then swap it
                $nr = $this->findBy(
                    array('sequence' => $position, 'parent' => $parentId)
                    );

                $nr[0]->setSequence($ir->getSequence());
                $this->getEntityManager()->persist($nr[0]);
                $ir->setSequence($position);
                $this->getEntityManager()->persist($ir);
                break;
            }
            // user drag from one level to the next, it is a new addition
            else {

                if ($position == $seq) {
                    $ir->setSequence($seq);
                    $this->getEntityManager()->persist($ir);
                    $seq++;
                }

                $nr = $this->findOneById($nc->getId());
                $nr->setSequence($seq);
                $this->getEntityManager()->persist($nr);

            }

            $seq++;
        }

        // if its the last entry and user moved to new level
        if ($old_parent_id != $parentId && $position == count($new_children)) {
            $ir->setSequence($seq);
            $this->getEntityManager()->persist($ir);
        }

        $message = '';
        $success = true;

        // step 3: run a loop, insert the new element and update the seq
        try {
            $this->getEntityManager()->flush();
            $this->getEntityManager()->clear(); // prevent doctrine from caching
            $message = 'flash_reorder_edit_success';
        }
        catch (\Exception $e) {
            // $message = $e->getMessage();
            $message = 'Cannot reorder element.';
            $success = false;
        }

        return array($message, $success);
    }
...
```

We need a custom query to get the pagemeta based on locale.

```
# src/Songbird/NestablePageBundle/Entity/PageMetaRepository.php

namespace Songbird\NestablePageBundle\Repository;

use Songbird\NestablePageBundle\Entity\Page;

/**
 * PageMetaRepository
 *
 * This class was generated by the Doctrine ORM. Add your own custom
 * repository methods below.
 */
class PageMetaRepository extends \Doctrine\ORM\EntityRepository
{
        /**
         * @param Page $page
         * @param $locale
         *
         * @return PageMeta
         */
         public function findPageMetaByLocale(Page $page, $locale) {

                $query = $this->createQueryBuilder('pm')
                              ->where('pm.locale = :locale')
                              ->andWhere('pm.page = :page')
                              ->setParameter('locale', $locale)
                              ->setParameter('page', $page)
                              ->getQuery();

                return $query->getOneOrNullResult();

        }
}
```

We then remove the created and modified date from the form as these fields should not be editable.

```
# src/Songbird/NestablePageBundle/Form/PageType.php

namespace Songbird\NestablePageBundle\Form;

use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\OptionsResolver\OptionsResolverInterface;

class PageType extends AbstractType
{
    /**
     * @param FormBuilderInterface $builder
     * @param array $options
     */
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder
            ->add('slug')
            ->add('isPublished')
            ->add('sequence')
            ->add('parent')
        ;
    }

    /**
     * @param OptionsResolver $resolver
     */
    public function configureOptions(OptionsResolver $resolver)
    {
        $resolver->setDefaults(array(
            'data_class' => 'Songbird\NestablePageBundle\Entity\Page'
        ));
    }
}
```

and we will leave the PageMetaController.php as default.

Now, we need to make changes to the view - list.html.twig.

```
# app/Resources/views/page/list.html.twig
{% extends '::base.html.twig' %}

{% block stylesheets %}
        {{ parent() }}
        <link rel="stylesheet" href="{{ asset('bundles/songbirdnestablepage/css/styles.\
css ') }}">
{% endblock %}

{% block body -%}
        <div class="alert alert-dismissable">
                {{ 'flash_reorder_instructions' | trans({}, 'SongbirdNestablePageBundle') }}
        </div>

        {% block main %}
            <button type="button" onclick="$('.dd').nestable('expandAll')">Expand All</\
button>
            <button type="button" onclick="$('.dd').nestable('collapseAll')">Collapse A\
ll</button>
            <div id="nestable" class="dd">
                <ol class="dd-list">
                    {% include "page/tree.html.twig" with { 'tree':tree } %}
                </ol>
            </div>
        {% endblock %}
        <ul class="record_actions">
            <li>
                <a href="{{ path('songbird_page_new') }}">
                    Create New Page
                </a>
            </li>
            <li>
                <a href="{{ path('songbird_pagemeta_new') }}">
                    Create New PageMeta
                </a>
            </li>
        </ul>
{% endblock %}

{% block script %}
    {{ parent() }}
    <script src="{{ asset('bundles/songbirdnestablepage/js/jquery-1.11.3.min.js'\
) }}"></script>
    <script src="{{ asset('bundles/songbirdnestablepage/js/jquery.nestable.js') \
}}"></script>
    <script>

    $(function() {

                        var before = null, after = null;

                        $('.dd').nestable({
                                afterInit: function ( event ) { }
                        });

        $('.dd').nestable('collapseAll');
        before = JSON.stringify($('.dd').nestable('serialize'));
        $('.dd').on('dragEnd', function(event, item, source, destination, positi\
on) {

                                        id = item.attr('data-id');
                                        parentId = item.closest('li').parent().closest('li').attr(

                                        // if parent id is null of if parent id and id is the same
evel.
                                        parentId = (parentId == id || typeof(parentId)  === "unde
rentId;

                                        after = JSON.stringify($('.dd').nestable('serialize'));

                if (before != after) {
                    $.ajax({
                        type: "POST",
                        url: "{{ path('songbird_page_reorder') }}",
                        data: {id: id, parentId: parentId, position: position},
                        success: function (data, dataType) {
                                                                                if (data.success)
                                                                                        $('.alert
                                                                                }
                                                                                else {
                                                                                        $('.alert
                                                                                }
                                                                                $('.alert').html(
                                                                                $('.alert').fadeTo
```

```
                                                                                  $('.alert').fadeT
                        },

                        error: function (XMLHttpRequest, textStatus, errorThrown) {
                            console.log(XMLHttpRequest);
                        }
                    });
                    before = after;
                }
            });
    });
    </script>
{% endblock %}
```

and tree.html.twig

```
# app/Resources/views/page/tree.html.twig
{% for v in tree %}
    <li class='dd-item' data-id='{{ v.getId() }}'>
        <div class='dd-handle'>
            <a class="dd-nodrag" href="{{ path('songbird_page_show', {id: v.getI\
d()}) }}">{{ v.getSlug() }}</a>
        </div>

        {% set children = v.getChildren()|length %}
        {% if children > 0 %}
            <ol class='dd-list'>
                {% include "page/tree.html.twig" with { 'tree':v.getChildren() }\
 %}
            </ol>
        {% endif %}
    </li>
{% endfor %}
```

We need to update the show.html.twig to allow user to view pagemeta.

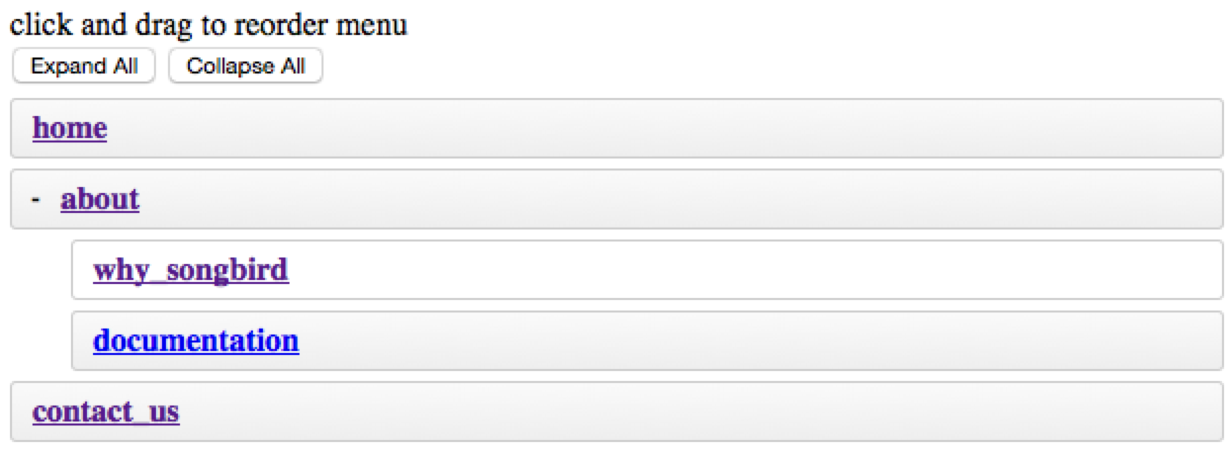
```
# app/Resources/views/pagemeta/show.html.twig

...
<ul>
        <li>
            <a href="{{ path('songbird_page_index') }}">Back to the list</a>
        </li>
        <li>
            <a href="{{ path('songbird_page_edit', { 'id': page.id }) }}">Edit P\
age</a>
        </li>
        <li>
            <a href="{{ path('songbird_pagemeta_show', { 'id': pageMeta.id }) }}\
">View PageMeta</a>
        </li>
        <li>
            <a href="{{ path('songbird_pagemeta_edit', { 'id': pageMeta.id }) }}\
">Edit PageMeta</a>
        </li>
        <li>
            {{ form_start(delete_form) }}
                <input type="submit" value="Delete">
            {{ form_end(delete_form) }}
        </li>
</ul>
```

The rest of the view templates can use the defaults. Ready to test the bundle?

```
-> ./scripts/resetapp
# remember gulp? the assets:install command is in there.
-> gulp
```

Now go to the page index and try reorder the menu.

```
http://songbird.app:8000/app_dev.php/songbird_page/
```



Create Functional Tests (Optional)

Sticking to the industrial standard, we are going to use PHPUnit rather than codeception. The main reason for doing that is to remove dependency on codeception. The only downside is that we could not simulate real browser interaction with the app.

Let us install phpunit using composer

```
-> ./scripts/composer require --dev phpunit/phpunit ^5.7
```

We need to call phpunit in docker, so we need another wrapper for it.

```
-> touch scripts/phpunit
-> chmod u+x scripts/phpunit
```

```
# in scripts/phpunit

#!/bin/bash
docker-compose exec php vendor/phpunit/phpunit/phpunit $@
```

Then, let us create the functional tests based on the user stories.

phpunit uses the phpunit.xml.dist under the symfony dir. To run the test, simply run this command in the symfony dir

```
# in symfony dir
-> ./scripts/phpunit
```

let us run testListPages function within PageControllerTest.php for example,

```
-> ./scripts/phpunit --filter testListPages src/Songbird/NestablePageBundle/Test\
s/Controller/PageControllerTest.php
```

You should get a “no tests executed” error because we haven’t write the test. Let us write the test.

```
# src/Songbird/NestablePageBundle/Tests/Controller/PageControllerTest.php

namespace Songbird\NestablePageBundle\Tests\Controller;

use Symfony\Bundle\FrameworkBundle\Test\WebTestCase;
use Symfony\Component\Console\Input\StringInput;
use Symfony\Bundle\FrameworkBundle\Console\Application;

/**
 * As test2 user
 * I WANT to manage pages
 * SO THAT I can update them anytime
 *
 * Class PageControllerTest
 * @package Songbird\NestablePageBundle\Tests\Controller
 */
class PageControllerTest extends WebTestCase
{
    protected static $application;

    protected function setUp()
    {
        self::getApplication()->run(new StringInput('doctrine:database:drop --fo\
rce'));
        self::getApplication()->run(new StringInput('doctrine:database:create'));
        self::getApplication()->run(new StringInput('doctrine:schema:create'));
        self::getApplication()->run(new StringInput('doctrine:fixtures:load -n')\
);
    }

    protected static function getApplication()
    {
        if (null === self::$application) {
            $client = static::createClient();

            self::$application = new Application($client->getKernel());
            self::$application->setAutoExit(false);
        }

        return self::$application;
    }

    /**
     * GIVEN List Pages
     * WHEN I go to /songbird_page
     * THEN I should see the why_songbird slug under the about slug
     *
     * scenario 17.11
     *
     * Test list action
     */
    public function testListPages()
    {
        $client = static::createClient();

        $crawler = $client->request('GET', '/songbird_page/list');
        // i should see why_songbird text
        $this->assertContains(
            'why_songbird',
            $client->getResponse()->getContent()
        );
        // there should be 3 parent menus
        $nodes = $crawler->filterXPath('//div[@id="nestable"]/ol');
        $this->assertEquals(count($nodes->children()), 3);

        // there should be 2 entries under the about menu
        $nodes = $crawler->filterXPath('//li[@data-id="2"]/ol');
        $this->assertEquals(count($nodes->children()), 2);
    }

    /**
     * GIVEN Show contact us page
     * WHEN I go to /songbird_page/5
     * THEN I should see the word "contact_us" and the word "Created"
     *
     * scenario 17.12
     *
     * Test show action
     */
    public function testShowContactUsPage()
    {
        $client = static::createClient();
        // go to main listing page
        $crawler = $client->request('GET', '/songbird_page/list');
        // click on contact_us link
        $crawler = $client->click($crawler->selectLink('contact_us')->link());

        // i should see "contact_us"
        $this->assertContains(
            'contact_us',
            $client->getResponse()->getContent()
        );

        // i should see "Created"
        $this->assertContains(
            'Created',
            $client->getResponse()->getContent()
        );
    }

    /**
     * GIVEN Reorder home
     * WHEN I simulate a drag and drop of the home menu to under the about menu \
and submit the post data to /songbird_page/reorder
     * THEN I should see "reordered successfully message" in the response and me\
nus should be updated
     *
     * scenario 17.13
     *
     * We simulate ajax submission by reordering menu
     */
    public function testReorderHomePage()
    {
        $client = static::createClient();

        // home is dragged under about and in the second position
        $crawler = $client->request(
            'POST',
            '/songbird_page/reorder',
            array(
                'id' => 1,
                'parentId' => 2,
                'position' => 1
            ),
            array(),
            array('HTTP_X-Requested-With' => 'XMLHttpRequest')
        );

        // i should get a success message in the returned content
        $this->assertContains(
            'menu has been reordered successfully',
            $client->getResponse()->getContent()
        );

        // go back to page list again
        $crawler = $client->request('GET', '/songbird_page/list');
        // there should be 2 parent menus
        $nodes = $crawler->filterXPath('//div[@id="nestable"]/ol');
        $this->assertEquals(count($nodes->children()), 2);
        // there should 3 items under the about menu
        $nodes = $crawler->filterXPath('//li[@data-id="2"]/ol');
        $this->assertEquals(count($nodes->children()), 3);
    }

    /**
     * GIVEN Edit home page meta
     * WHEN I go to edit homepage url and update the menu title of "Home" to "Ho\
me1" and click update
     * THEN I should see the text "successfully updated" message
     *
     * scenario 17.14
     *
     * Test edit action
     */
    public function testEditHomePage()
    {
        $client = static::createClient();

        $crawler = $client->request('GET', '/songbird_page/1/edit');

        $form = $crawler->selectButton('Edit')->form(array(
            'songbird_nestablepagebundle_page[slug]'  => 'home1',
        ));

        $client->submit($form);

        // go back to the list again and i should see the slug updated
        $crawler = $client->request('GET', '/songbird_page/list');
        $this->assertContains(
            'home1',
            $client->getResponse()->getContent()
        );
    }

    /**
     * GIVEN Create and delete test pagemeta
     * WHEN go to /new and fill in details and click "Create" button, then go to\
 test page and click add new meta and fill in the details and click "create" but\
ton, then click delete button
     * THEN I should see the new page and pagemeta being created and pagemeta de\
leted
     *
     * scenario 17.15
     *
     * Test new and delete action
     */
    public function testCreateDeleteTestPage()
    {
        $client = static::createClient();

        $crawler = $client->request('GET', '/songbird_page/new');

        $form = $crawler->selectButton('Create')->form(array(
            'songbird_nestablepagebundle_page[slug]'  => 'test_page',
            'songbird_nestablepagebundle_page[isPublished]'  => true,
            'songbird_nestablepagebundle_page[sequence]'  => 1,
            'songbird_nestablepagebundle_page[parent]'  => 2,
        ));

        $client->submit($form);

        // go back to the list again and i should see the slug updated
        $crawler = $client->request('GET', '/songbird_page/list');
        $this->assertContains(
            'test_page',
            $client->getResponse()->getContent()
        );

        $crawler = $client->click($crawler->selectLink('Create New PageMeta')->l\
ink());
        // at create new pagemeta page. new test_page is id 6
        $form = $crawler->selectButton('Create')->form(array(
            'songbird_nestablepagebundle_pagemeta[page_title]'  => 'test page ti\
tle',
            'songbird_nestablepagebundle_pagemeta[menu_title]'  => 'test menu ti\
tle',
            'songbird_nestablepagebundle_pagemeta[short_description]'  => 'short\
 content',
            'songbird_nestablepagebundle_pagemeta[content]'  => 'long content',
            'songbird_nestablepagebundle_pagemeta[page]'  => 6,
        ));

        $crawler = $client->submit($form);

        // follow redirect to show pagemeta
        $crawler = $client->followRedirect();

        $this->assertContains(
            'short content',
            $client->getResponse()->getContent()
        );

        // at show pagemeta, click delete
        $form = $crawler->selectButton('Delete')->form();
        $crawler = $client->submit($form);

        // go back to the pagemeta list again and i should NOT see the test_page\
 anymore
        $crawler = $client->request('GET', '/songbird_pagemeta');

        $this->assertNotContains(
            'test page title',
            $client->getResponse()->getContent()
        );
    }

    /**
     * GIVEN Delete contact us page
     * WHEN go to /songbird_page/5 and click "Delete" button
     * THEN I should see the contact_us slug no longer available in the listing \
page. Page id 5 should no longer be found in the pagemeta table
     *
     * scenario 17.16
     */
    public function testDeleteContactUsPage()
    {
        $client = static::createClient();
        // now if we remove contact_us page, ie id 5, all its page meta should b\
e deleted
        $crawler = $client->request('GET', '/songbird_page/5');
        $form = $crawler->selectButton('Delete')->form();
        $crawler = $client->submit($form);
        $crawler = $client->followRedirect();

        $this->assertNotContains(
            'contact_us',
            $client->getResponse()->getContent()
        );

        // we now connect to do and make sure the related pagemetas are no longe\
r in the pagemeta table.
        $res = $client->getContainer()->get('doctrine')->getRepository('Songbird\
NestablePageBundle:PageMeta')->findByPage(5);
        $this->assertEquals(0, count($res));
    }

}
```

As we are testing both page and pagemeta controller at the same time, we can remove the pagemeta controller test.

```
-> rm src/Songbird/NestablePageBundle/Tests/Controller/PageMetaControllerTest.php
```

lets run the test again and make sure everything is ok

```
-> ./scripts/phpunit src/Songbird/NestablePageBundle
Creating database schema...
Database schema created successfully!
  > purging database
  > loading Songbird\NestablepageBundle\DataFixtures\ORM\LoadPageData
  > loading [1] AppBundle\DataFixtures\ORM\LoadUserData
  > loading [2] AppBundle\DataFixtures\ORM\LoadMediaData
...
Time: 30.71 seconds, Memory: 70.75Mb

OK (6 tests, 14 assertions)
```

You might have noticed that the phpunit functional tests seemed to run much faster than codeception acceptance tests. Why? Does that makes it more attractive to you?

Whatever we do in this chapter should not affect what we have done previously. To verify that this is indeed the case,

```
-> ./scripts/runtest
```

Remember to commit all the code before moving on.

Summary

In this chapter, we have created our own page bundle and generated CRUD in a quick way using the command line. We have also customised the listing page and created a draggable menu using the jquery nestable menu. Data is submitted to the backend via ajax and updated dynamically.

Exercises

- Are there any benefits of creating a page bundle that has no dependency on Symfony at all? How would you do it? (Optional)
- KnpmenuBundle[126] is a popular bundle for handling menus. How would you integrate it with SongbirdNestableMenu? (Optional)

[126]https://github.com/KnpLabs/KnpMenuBundle

References

- Nestable js[127]
- Symfony Testing[128]
- Doctrine Association Mapping[129]

[127]https://github.com/bernardpeh/Nestable
[128]http://symfony.com/doc/current/book/testing.html
[129]http://docs.doctrine-project.org/projects/doctrine-orm/en/latest/reference/association-mapping.html

Chapter 18: Making Your Bundle Reusable

We have created a page bundle in the previous chapter using the default way. It's not perfect if you want to share it with everyone. How do we do that? Be warned, we need lots of refactoring in the code to make it sharable.

This is a long chapter. Its is a good process to go through because it makes you pause and think. If you already know the process and want to skip through, simple clone the NestablePageBundle from github[130] and follow the installation instructions in the readme file. Then, jump over to the next chapter.

Creating a separate repository

First of all, let us create a readme file.

```
-> cd src/Songbird/NestablePageBundle
-> touch readme.md
```

Update the readme file.

Let us create the composer.json file for this repo. We will do a simple one

```
-> composer init
```

Follow the prompts. You might need to read up on software licensing. MIT license[131] is becoming really popular. The sample composer.json might look like this:

[130] https://github.com/bernardpeh/NestablePageBundle
[131] https://en.wikipedia.org/wiki/MIT_License

```
{
    "name": "Yourname/nestable-page-bundle",
    "description": "your description",
    "type": "symfony-bundle",
    "require": {
        "symfony/symfony": "~3.0"
    },
    "require-dev": {
        "doctrine/doctrine-fixtures-bundle": "~2.0"
    },
    "autoload": {
        "psr-4": { "Songbird\NestablePageBundle\": "" }
    },
    "license": "MIT",
    "authors": [
        {
            "name": "your name",
            "email": "your_email@your_email.xx"
        }
    ]
}
```

Note that we have to add the "autoload" component so that Symfony can autoload the namespace post installation. PS-4[132] is the default standard at the time of writing. Next, let us create the license in a text file

```
-> touch LICENSE
```

copy the MIT LICENSE[133] and update the LICENSE file.

Init the repo

```
-> cd src/Songbird/NestablePageBundle
-> git init .
-> git add .
-> git commit -m"init commit"
```

In github[134] (create a new acct if not done), create a new repo. Let's call it NestablePageBundle for example. Once you have created the new repo, you should see instructions on how to push your code.

[132]https://getcomposer.org/doc/04-schema.md#autoload
[133]http://opensource.org/licenses/MIT
[134]http://github.com

```
-> git remote add origin git@github.com:your_username/NestablePageBundle.git
-> git push -u origin master
```

Let us give our first release a version number using the semantic versioning[135] convention.

```
-> git tag 0.1.0
-> git push --tags
```

Your repository is now available for the public to pull.

Updating Application composer.json

If we add our repo to packagist[136], we could install our bundle like any other bundles using the "composer require" command. Anyone reading this tutorial might submit their test bundle to packagist, so I thought it would be a better idea to install the bundle from git instead. Let's use github for the sake of illustration.

```
# composer.json
...
    "repositories": [
        {
            "type": "git",
            "url": "https://github.com/your_name/NestablePageBundle"
        }
    ],
...
    "require": {
        ...
        "your_name/nestable-page-bundle": ">0.1.0"
    }
...
```

Note that the bundle name is "nestable-page-bundle" under the "require" section. Why not use NestablePageBundle following Symfony's convention? Remember the composer.json file that you have created previously? "nestable-page-bundle" is the name of the bundle as specified in that composer file.

Now lets run composer update and see what happens

[135] http://semver.org
[136] https://packagist.org

```
-> cd ../../../
-> composer update
...

  - Installing your_name/nestable-page-bundle (0.1.0)
    Downloading: 100%
```

At this point, look at the vendor directory and you will see your bundle being installed in there. That's a good start.

Renaming SongbirdNestablePageBundle

Let us do some cleaning up. We no longer need the src/Songbird/NestablePageBundle since we have installed the bundle under vendor dir.

```
git rm -rf src/Songbird/
git rm -rf app/Resources/views/{page,pagemeta}
```

Let us check if the route is still there.

```
-> ./scripts/console debug:router | grep songbird
...
songbird_page            GET      ANY    ANY  /songbird_page/
songbird_page_list       GET      ANY    ANY  /songbird_page/list
songbird_page_reorder    POST     ANY    ANY  /songbird_page/reorder
```

Woah!! We have already deleted src/Songbird/NestablePageBundle and we should expect to see some errors. Why are the songbird routes still there?

We have a problem. The namespace "Songbird" is no longer relevant in vendor/your-name/nestable-page-bundle since the bundle is already decoupled from Songbird CMS. We want to change the bundle's filename and namespace so that it is more intuitive. How do we do that?

Let us re-download the repo and do some mass restructuring

```
-> cd vendor/your-name
-> rm -rf nestable-page-bundle
-> git clone git@github.com:your_name/NestablePageBundle.git nestable-page-bundle
-> cd nestable-page-bundle
```

There is no quick way for this, some bash magic helps

```
# Your-Initial can be something short but has to be unique
# let us change the namespace
-> find . -type f | grep -v .git/ | while read s; do sed -i '' 's/Songbird\Nesta\
blePageBundle/{your-initial}\NestablePageBundle/g' $s ; done
# change the bundle name
-> find . -type f | grep -v .git/ | while read s; do sed -i '' 's/SongbirdNestab\
lePage/{your-initial}NestablePage/g' $s ; done
-> find . -type f | grep -v .git/ | while read s; do sed -i '' 's/songbird_/{you\
r_initial}_/g' $s ; done
-> find . -type f | grep -v .git/ | while read s; do sed -i '' 's/songbirdnestab\
le/{your_initial}nestable/g' $s ; done
```

That should save us 90% of the time. Then visually walk through all the files and rename whatever that was not renamed by the bash commands.

Lastly, rename the bundle file

```
-> git mv SongbirdNestablePageBundle.php {your-initial}NestablePageBundle.php
-> cd DependencyInjection
-> git mv SongbirdNestablePageExtension.php BpehNestablePageExtension.php
-> cd ../Resources/translations
-> git mv SongbirdNestablePageBundle.en.xlf {your-initial}NestablePageBundle.en.\
xlf
-> git mv SongbirdNestablePageBundle.fr.xlf {your-initial}NestablePageBundle.fr.\
xlf
```

Now, here is the question. How do we test our changes without committing to git and re-run composer update? We can update our entry in vendor/composer/autoload_psr4.php

```
# vendor/composer/autoload_psr4.php
...
    # 'Songbird\NestablePageBundle\' => array($vendorDir . '/{your-name}/nestabl\
e-page-bundle'),
    '{your-initial}\NestablePageBundle\' => array($vendorDir . '/{your-name}/nes\
table-page-bundle'),
...
```

Now, let us update AppKernel

```
# app/config/AppKernel.php
...
# new SongbirdNestablePageBundle(),
new {your-initial}NestablePageBundle(),
```

and routing

```
# app/config/routing.yml

# songbird_nestable_page:
#     resource: "@SongbirdNestablePageBundle/Controller/"
#    type:     annotation
#    prefix:   /

{your-initial}_nestable_page:
    resource: "@{your-initial}/NestablePageBundle/Controller/"
    type:     annotation
    prefix:   /
```

My initial is bpeh, let us check that the routes are working.

```
-> ./scripts/console debug:router | grep bpeh
bpeh_page                GET      ANY    ANY  /bpeh_page/
bpeh_page_list           GET      ANY    ANY  /bpeh_page/list
bpeh_page_reorder        POST     ANY    ANY  /bpeh_page/reorder
...
```

We can now install the assets.

```
-> gulp
```

Now go your new page list url and do a quick test. In my case,

```
http://songbird.app:8000/app_dev.php/bpeh_page/list
```

Looks like it is working. How can we be sure? Remember our functional tests?

```
-> ./scripts/phpunit vendor/bpeh/nestable-page-bundle/
...
```

If it fails, why? Can you fix it?

Remember to commit your code before moving to the next chapter. Up your nestablepagebundle tags to 0.2.0 or something else since there were major changes.

Making the Bundle Extensible

When this bundle is initialised in AppKernel.php, running “scripts/console doctrine:schema:create will create the default tables. We should be able to extend this bundle and modify the entity name and methods easily. The war is not over. There are still lots to be done!!

Let us clean up the AppKernel and Route.

```
# app/AppKernel.php
...
// new {your-inital}NestablePageBundle(),
...
```

and in routing.yml

```
# app/config/routing.yml

# {your-initial}_nestable_page:
# resource: "@{your-initial}NestablePageBundle/Controller/"
# type:     annotation
# prefix:   /
```

and refocus our attention to the NestablePageBundle:

```
-> cd vendor/{your-initial}/NestablePageBundle
```

First of all, we need to make Page and PageMeta entities extensible. We will move the entities to the Model directory, making the entities abstract.

I’ll be using my initial “bpeh” from now onwards to make life easier when referencing paths.

```
# vendor/bpeh/nestable-page-bundle/Model/PageBase.php

namespace Bpeh\NestablePageBundle\Model;

use Doctrine\ORM\Mapping as ORM;

/**
 * Page
 *
 */
abstract class PageBase
```

```
{
    /**
     * @var integer
     *
     * @ORM\Column(name="id", type="integer")
     * @ORM\Id
     * @ORM\GeneratedValue(strategy="AUTO")
     */
    protected $id;

    /**
     * @var string
     *
     * @ORM\Column(name="slug", type="string", length=255, unique=true)
     */
    protected $slug;

    /**
     * @var boolean
     *
     * @ORM\Column(name="isPublished", type="boolean", nullable=true)
     */
    protected $isPublished;

    /**
     * @var integer
     *
     * @ORM\Column(name="sequence", type="integer", nullable=true)
     */
    protected $sequence;

    /**
     * @var \DateTime
     *
     * @ORM\Column(name="modified", type="datetime")
     */
    protected $modified;

    /**
     * @var \DateTime
     *
     * @ORM\Column(name="created", type="datetime")
     */
    protected $created;


    /**
     * @ORM\ManyToOne(targetEntity="Bpeh\NestablePageBundle\Model\PageBase", inv\
ersedBy="children")
     * @ORM\JoinColumn(name="parent_id", referencedColumnName="id", onDelete="CA\
SCADE")}
     * @ORM\OrderBy({"sequence" = "ASC"})
     */
    protected $parent;

    /**
     * @ORM\OneToMany(targetEntity="Bpeh\NestablePageBundle\Model\PageBase", map\
pedBy="parent")
     * @ORM\OrderBy({"sequence" = "ASC"})
     */
    protected $children;

    /**
     * @ORM\OneToMany(targetEntity="Bpeh\NestablePageBundle\Model\PageMetaBase",\
 mappedBy="page", cascade={"persist"}))
     */
    protected $pageMetas;

    /**
     * Get id
     *
     * @return integer
     */
    public function getId()
    {
        return $this->id;
    }

    /**
     * Set slug
     *
     * @param string $slug
     * @return Page
     */
    public function setSlug($slug)
    {
        $this->slug = $slug;

        return $this;
    }

    /**
     * Get slug
     *
     * @return string
     */
    public function getSlug()
    {
        return $this->slug;
    }

    /**
     * Set isPublished
     *
     * @param boolean $isPublished
     * @return Page
     */
    public function setIsPublished($isPublished)
    {
        $this->isPublished = $isPublished;

        return $this;
    }

    /**
     * Get isPublished
     *
     * @return boolean
     */
    public function getIsPublished()
    {
        return $this->isPublished;
    }

    /**
     * Set sequence
     *
     * @param integer $sequence
     * @return Page
     */
    public function setSequence($sequence)
    {
        $this->sequence = $sequence;

        return $this;
    }

    /**
     * Get sequence
     *
     * @return integer
     */
    public function getSequence()
    {
        return $this->sequence;
    }

    /**
     * Set modified
     *
     * @param \DateTime $modified
     * @return Page
     */
    public function setModified($modified)
    {
        $this->modified = $modified;

        return $this;
    }

    /**
     * Get modified
     *
     * @return \DateTime
     */
    public function getModified()
    {
        return $this->modified;
    }

    /**
     * Set created
     *
     * @param \DateTime $created
     * @return Page
     */
    public function setCreated($created)
    {
        $this->created = $created;

        return $this;
    }

    /**
     * Get created
     *
     * @return \DateTime
     */
    public function getCreated()
    {
        return $this->created;
    }

    /**
     * Constructor
     */
    public function __construct()
    {
        $this->children = new \Doctrine\Common\Collections\ArrayCollection();
        $this->pageMetas = new \Doctrine\Common\Collections\ArrayCollection();
    }

    /**
     * @ORM\PrePersist
     */
    public function prePersist()
    {
        // update the modified time
        $this->setModified(new \DateTime());

        // for newly created entries
        if ($this->getCreated() == null) {
            $this->setCreated(new \DateTime('now'));
        }
        $this->created = new \DateTime();
    }

    /**
     * Set parent
     *
     * @param \Bpeh\NestablePageBundle\Model\PageBase $parent
     * @return Page
     */
    public function setParent(\Bpeh\NestablePageBundle\Model\PageBase $parent = \
null)
    {
        $this->parent = $parent;

        return $this;
    }

    /**
     * Get parent
     *
     * @return \Bpeh\NestablePageBundle\Model\PageBase
     */
    public function getParent()
    {
        return $this->parent;
    }

    /**
     * Add children
     *
     * @param \Bpeh\NestablePageBundle\Model\PageBase $children
     * @return Page
     */
    public function addChild(\Bpeh\NestablePageBundle\Model\PageBase $children)
    {
        $this->children[] = $children;

        return $this;
    }

    /**
     * Remove children
     *
     * @param \Bpeh\NestablePageBundle\Model\Page $children
     */
    public function removeChild(\Bpeh\NestablePageBundle\Model\PageBase $childre\
n)
    {
        $this->children->removeElement($children);
    }

    /**
     * Get children
     *
     * @return \Doctrine\Common\Collections\Collection
     */
    public function getChildren()
    {
        return $this->children;
    }

    /**
     * Add pageMetas
     *
     * @param \Bpeh\NestablePageBundle\Model\PageMetaBase $pageMetas
     * @return Page
     */
    public function addPageMeta(\Bpeh\NestablePageBundle\Model\PageMetaBase $pag\
eMetas)
    {
        $this->pageMetas[] = $pageMetas;

        return $this;
    }

    /**
     * Remove pageMetas
     *
     * @param \Bpeh\NestablePageBundle\Model\PageMetaBase $pageMetas
     */
    public function removePageMeta(\Bpeh\NestablePageBundle\Model\PageMetaBase $\
pageMetas)
    {
        $this->pageMetas->removeElement($pageMetas);
    }

    /**
     * Get pageMetas
     *
     * @return \Doctrine\Common\Collections\Collection
     */
    public function getPageMetas()
    {
        return $this->pageMetas;
    }

    /**
     * convert object to string
     * @return string
     */
    public function __toString()
    {
        return $this->slug;
    }
}
```

Note that we have changed all variables to “protected” to allow inheritance. The references to PageBase has also been changed.

To make our bundle flexible, we also need to allow user to specify their own child entities, form type and templates to use.

```
# vendor/bpeh/nestable-page-bundle/DependencyInjection/Configuration.php

namespace Bpeh\NestablePageBundle\DependencyInjection;

use Symfony\Component\Config\Definition\Builder\TreeBuilder;
use Symfony\Component\Config\Definition\ConfigurationInterface;

/**
 * This is the class that validates and merges configuration from your app/confi\
g files
 *
 * To learn more see {@link http://symfony.com/doc/current/cookbook/bundles/exte\
nsion.html#cookbook-bundles-extension-config-class}
 */
class Configuration implements ConfigurationInterface
{
    /**
     * {@inheritdoc}
     */
    public function getConfigTreeBuilder()
    {
        $treeBuilder = new TreeBuilder();
        $rootNode = $treeBuilder->root('bpeh_nestable_page');
        // Here you should define the parameters that are allowed to
        // configure your bundle. See the documentation linked above for
        // more information on that topic.
        $rootNode
            ->children()
                ->scalarNode('page_entity')->defaultValue('Bpeh\NestablePageBund\
le\PageTestBundle\Entity\Page')->end()
                ->scalarNode('pagemeta_entity')->defaultValue('Bpeh\NestablePage\
Bundle\PageTestBundle\Entity\PageMeta')->end()
                ->scalarNode('page_form_type')->defaultValue('Bpeh\NestablePageB\
undle\PageTestBundle\Form\PageType')->end()
                ->scalarNode('pagemeta_form_type')->defaultValue('Bpeh\NestableP\
ageBundle\PageTestBundle\Form\PageMetaType')->end()
                    ->scalarNode('page_view_list')->defaultValue('BpehNestablePageBundl\
e:Page:list.html.twig')->end()
                        ->scalarNode('page_view_edit')->defaultValue('BpehNestablePageBundle:P\
age:edit.html.twig')->end()
                        ->scalarNode('page_view_show')->defaultValue('BpehNestablePageBundle:P\
age:show.html.twig')->end()
                        ->scalarNode('page_view_new')->defaultValue('BpehNestablePageBundle:Pa\
ge:new.html.twig')->end()
                        ->scalarNode('pagemeta_view_new')->defaultValue('BpehNestablePageBundl\
e:PageMeta:new.html.twig')->end()
                        ->scalarNode('pagemeta_view_edit')->defaultValue('BpehNestablePageBund\
le:PageMeta:edit.html.twig')->end()
                        ->scalarNode('pagemeta_view_index')->defaultValue('BpehNestablePageBun\
dle:PageMeta:index.html.twig')->end()
                        ->scalarNode('pagemeta_view_show')->defaultValue('BpehNestablePageBund\
le:PageMeta:show.html.twig')->end()
            ->end()
        ;
        return $treeBuilder;
    }
}
```

and the extension

```
# vendor/bpeh/nestable-page-bundle/DependencyInjection/BpehNestablePageExtension\
.php

namespace Bpeh\NestablePageBundle\DependencyInjection;

use Symfony\Component\DependencyInjection\ContainerBuilder;
use Symfony\Component\Config\FileLocator;
use Symfony\Component\HttpKernel\DependencyInjection\Extension;
use Symfony\Component\DependencyInjection\Loader\YamlFileLoader;

/**
 * This is the class that loads and manages your bundle configuration
 *
 * To learn more see {@link http://symfony.com/doc/current/cookbook/bundles/exte\
nsion.html}
 */
class BpehNestablePageExtension extends Extension
{
    /**
     * {@inheritdoc}
     */
    public function load(array $configs, ContainerBuilder $container)
    {
        $configuration = new Configuration();
        $config = $this->processConfiguration($configuration, $configs);

        $container->setParameter( 'bpeh_nestable_page.page_entity', $config[ 'pa\
ge_entity' ]);
        $container->setParameter( 'bpeh_nestable_page.pagemeta_entity', $config[\
 'pagemeta_entity' ]);
        $container->setParameter( 'bpeh_nestable_page.page_form_type', $config[ \
'page_form_type' ]);
        $container->setParameter( 'bpeh_nestable_page.pagemeta_form_type', $conf\
ig[ 'pagemeta_form_type' ]);
            $container->setParameter( 'bpeh_nestable_page.page_view_list', $config[ 'pa\
ge_view_list' ]);
            $container->setParameter( 'bpeh_nestable_page.page_view_new', $config[ 'pag\
e_view_new' ]);
            $container->setParameter( 'bpeh_nestable_page.page_view_edit', $config[ 'pa\
ge_view_edit' ]);
            $container->setParameter( 'bpeh_nestable_page.page_view_show', $config[ 'pa\
ge_view_show' ]);
            $container->setParameter( 'bpeh_nestable_page.pagemeta_view_index', $config\
[ 'pagemeta_view_index' ]);
            $container->setParameter( 'bpeh_nestable_page.pagemeta_view_edit', $config[\
 'pagemeta_view_edit' ]);
            $container->setParameter( 'bpeh_nestable_page.pagemeta_view_new', $config[ \
'pagemeta_view_new' ]);
            $container->setParameter( 'bpeh_nestable_page.pagemeta_view_show', $config[\
 'pagemeta_view_show' ]);
            $loader = new YamlFileLoader($container, new FileLocator(__DIR__ . '/../Res\
ources/config'));
            $loader->load('services.yml');
    }
}
```

Now in config.yml, anyone can define the page and pagemeta entities themselves.

We also need to run the constructor to initialise the new config parameters when the controllers are loaded. To do that, we will need to do it via the controller event listener.

```
# vendor/bpeh/nestable-page-bundle/Resources/config/services.yml

services:

  bpeh_nestable_page.init:
    class: Bpeh\NestablePageBundle\EventListener\ControllerListener
    tags:
      - { name: kernel.event_listener, event: kernel.controller, method: onKerne\
lController}
```

and in the controller listener class

```
# vendor/bpeh/nestable-page-bundle/EventListener/ControllerListener.php

namespace Bpeh\NestablePageBundle\EventListener;

use Symfony\Component\HttpKernel\Event\FilterControllerEvent;
use Bpeh\NestablePageBundle\Controller\PageController;
use Bpeh\NestablePageBundle\Controller\PageMetaController;

class ControllerListener
{

    public function onKernelController(FilterControllerEvent $event)
    {
        $controller = $event->getController();

        /*
         * controller must come in an array
         */
        if (!is_array($controller)) {
            return;
        }

        if ($controller[0] instanceof PageController || $controller[0] instanceo\
f PageMetaController) {
            $controller[0]->init();
        }
    }
}
```

The Page Controller can now use the parameters as defined in config.yml to load the entities and form types.

```
# vendor/bpeh/nestable-page-bundle/Controller/PageController.php

namespace Bpeh\NestablePageBundle\Controller;

use Bpeh\NestablePageBundle\Model\PageBase as Page;
use Sensio\Bundle\FrameworkExtraBundle\Configuration\Route;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Bundle\FrameworkBundle\Controller\Controller;
use Sensio\Bundle\FrameworkExtraBundle\Configuration\Method;
use Symfony\Component\HttpFoundation\JsonResponse;
```

```

/**
 * Page controller.
 *
 * @Route("/bpeh_page")
 */
class PageController extends Controller
{

    private $entity;

        private $entity_meta;

    private $page_form_type;

        private $page_view_list;

        private $page_view_new;

        private $page_view_edit;

        private $page_view_show;

    public function init()
    {
            $this->entity = $this->container->getParameter('bpeh_nestable_page.page_ent\
ity');
            $this->entity_meta = $this->container->getParameter('bpeh_nestable_page.pag\
emeta_entity');
        $this->page_form_type = $this->container->getParameter('bpeh_nestable_pa\
ge.page_form_type');
            $this->page_view_list = $this->container->getparameter('bpeh_nestable_page.\
page_view_list');
            $this->page_view_new = $this->container->getparameter('bpeh_nestable_page.p\
age_view_new');
            $this->page_view_edit = $this->container->getparameter('bpeh_nestable_page.\
page_view_edit');
            $this->page_view_show = $this->container->getparameter('bpeh_nestable_page.\
page_view_show');
    }

        /**
```

```
         * Lists all Page entities.
         *
         * @Route("/", name="bpeh_page")
         * @Method("GET")
         *
         * @return \Symfony\Component\HttpFoundation\RedirectResponse
         */
    public function indexAction()
    {
            return $this->redirect($this->generateUrl('bpeh_page_list'));
    }

        /**
         * Lists all nested page
         *
         * @Route("/list", name="bpeh_page_list")
         * @Method("GET")
         *
         * @return array
         */
    public function listAction()
    {
            $em = $this->getDoctrine()->getManager();
        $rootMenuItems = $em->getRepository($this->entity)->findParent();

        return $this->render($this->page_view_list, array(
            'tree' => $rootMenuItems,
        ));
    }

        /**
         * reorder pages
         *
         * @Route("/reorder", name="bpeh_page_reorder")
         * @Method("POST")
         *
         * @param Request $request
         *
         * @return JsonResponse
         */
    public function reorderAction(Request $request)
    {
            $em = $this->getDoctrine()->getManager();
            // id of affected element
            $id = $request->get('id');

            // if invalid token, fail silently
            if (!$this->isCsrfTokenValid('bpeh_page_reorder', $request->get('csrf'))) {
                    // fail silently
                    return;
            }

            // parent Id
            $parentId = ($request->get('parentId') == '') ? null : $request->get('paren\
tId');
            // new sequence of this element. 0 means first element.
            $position = $request->get('position');

            $result = $em->getRepository($this->entity)->reorderElement($id, $parentId,\
 $position);

            return new JsonResponse(
                    array('message' => $this->get('translator')->trans($result[0], array(), 'B\
pehNestablePageBundle')
                    , 'success' => $result[1])
            );
    }

        /**
         * Creates a new Page entity.
         *
         * @Route("/new", name="bpeh_page_new")
         * @Method({"GET", "POST"})
         *
         * @param Request $request
         *
         * @return array|\Symfony\Component\HttpFoundation\RedirectResponse
         */
    public function newAction(Request $request)
    {
            $page = new $this->entity();
            $form = $this->createForm($this->page_form_type, $page);
            $form->handleRequest($request);

            if ($form->isSubmitted() && $form->isValid()) {
                    $em = $this->getDoctrine()->getManager();
                    $em->persist($page);
                    $em->flush();

                    return $this->redirectToRoute('bpeh_page_show', array('id' => $page->getId\
()));
            }

            return $this->render($this->page_view_new, array(
                    'page' => $page,
                    'form' => $form->createView(),
            ));
    }

        /**
         * Finds and displays a Page entity.
         *
         * @Route("/{id}", name="bpeh_page_show")
         * @Method("GET")
         *
         * @param Request $request
         *
         * @return array
         */
        public function showAction(Request $request)
        {
                $em = $this->getDoctrine()->getManager();

                $page = $em->getRepository($this->entity)->find($request->get('id'));

                $pageMeta = $em->getRepository($this->entity_meta)->findPageMetaByLocale($page\
,$request->getLocale());

                $deleteForm = $this->createDeleteForm($page);

                return $this->render($this->page_view_show, array(
                        'page' => $page,
                        'pageMeta' => $pageMeta,
                        'delete_form' => $deleteForm->createView(),
                ));

        }

        /**
         * Displays a form to edit an existing Page entity.
         *
         * @Route("/{id}/edit", name="bpeh_page_edit")
         * @Method({"GET", "POST"})
         *
         * @param Request $request
         *
         * @return array|\Symfony\Component\HttpFoundation\RedirectResponse
         */
        public function editAction(Request $request)
        {
                $em = $this->getDoctrine()->getManager();
                $page = $em->getRepository($this->entity)->find($request->get('id'));
                $deleteForm = $this->createDeleteForm($page);
                $editForm = $this->createForm($this->page_form_type, $page);
                $editForm->handleRequest($request);

                if ($editForm->isSubmitted() && $editForm->isValid()) {
                        $em = $this->getDoctrine()->getManager();
                        $em->persist($page);
                        $em->flush();

                        return $this->redirectToRoute('bpeh_page_edit', array('id' => $page->getI
);
                }

                return $this->render($this->page_view_edit, array(
                        'page' => $page,
                        'edit_form' => $editForm->createView(),
                        'delete_form' => $deleteForm->createView(),
                ));

        }

        /**
         * Deletes a Page entity.
         *
         * @Route("/{id}", name="bpeh_page_delete")
         * @Method("DELETE")
         *
         * @param Request $request
         *
         * @return \Symfony\Component\HttpFoundation\RedirectResponse
         */
        public function deleteAction(Request $request)
        {
                $em = $this->getDoctrine()->getManager();
                $page = $em->getRepository($this->entity)->find($request->get('id'));
                $form = $this->createDeleteForm($page);
                $form->handleRequest($request);

                if ($form->isSubmitted() && $form->isValid()) {
                        $em = $this->getDoctrine()->getManager();
                        $em->remove($page);
                        $em->flush();
                }

                return $this->redirectToRoute('bpeh_page_list');
        }

        /**
         * Creates a form to delete a Page entity.
         *
         * @return \Symfony\Component\Form\Form The form
         */
        private function createDeleteForm(Page $page)
        {
                return $this->createFormBuilder()
                        ->setAction($this->generateUrl('bpeh_page_delete', array('id' => $\
page->getId())))
                        ->setMethod('DELETE')
                        ->getForm()
                        ;
        }

}
```

Likewise for PageMeta Controller

```
# vendor/bpeh/nestable-page-bundle/Controller/PageMetaController.php

namespace Bpeh\NestablePageBundle\Controller;

use Symfony\Component\HttpFoundation\Request;
use Symfony\Bundle\FrameworkBundle\Controller\Controller;
use Sensio\Bundle\FrameworkExtraBundle\Configuration\Method;
use Sensio\Bundle\FrameworkExtraBundle\Configuration\Route;
use Bpeh\NestablePageBundle\Model\PageMetaBase as PageMeta;

/**
 * PageMeta controller.
 *
 * @Route("/bpeh_pagemeta")
 */
class PageMetaController extends Controller
{

        private $entity;

        private $entity_meta;

        private $page_meta_form_type;

        private $pagemeta_view_index;

        private $pagemeta_view_new;

        private $pagemeta_view_edit;

        private $pagemeta_view_show;

        public function init()
        {
                $this->entity = $this->container->getParameter('bpeh_nestable_page.page_entity\
');
                $this->entity_meta = $this->container->getParameter('bpeh_nestable_page.pageme\
ta_entity');
                $this->page_meta_form_type = $this->container->getParameter('bpeh_nestable_pag\
e.pagemeta_form_type');
                $this->pagemeta_view_index = $this->container->getparameter('bpeh_nestable_pag\
e.pagemeta_view_index');
```

```
                $this->pagemeta_view_new = $this->container->getparameter('bpeh_nestable_page.\
pagemeta_view_new');
                $this->pagemeta_view_edit = $this->container->getparameter('bpeh_nestable_page\
.pagemeta_view_edit');
                $this->pagemeta_view_show = $this->container->getparameter('bpeh_nestable_page\
.pagemeta_view_show');
        }

        /**
         * Lists all PageMeta entities.
         *
         * @Route("/", name="bpeh_pagemeta_index")
         * @Method("GET")
         */
        public function indexAction()
        {
                $em = $this->getDoctrine()->getManager();

                $pageMetas = $em->getRepository($this->entity_meta)->findAll();

                return $this->render($this->pagemeta_view_index, array(
                        'pageMetas' => $pageMetas,
                ));

        }

        /**
         * Creates a new PageMeta entity.
         *
         * @Route("/new", name="bpeh_pagemeta_new")
         * @Method({"GET", "POST"})
         */
        public function newAction(Request $request)
        {
                $pageMeta = new $this->entity_meta();
                $form = $this->createForm($this->page_meta_form_type, $pageMeta);
                $form->handleRequest($request);

                if ($form->isSubmitted() && $form->isValid()) {
                        $em = $this->getDoctrine()->getManager();

                        if ( $em->getRepository( $this->entity_meta )->findPageMetaByLocale( $page
a->getPage(), $pageMeta->getLocale() ) ) {
                                $this->get('session')->getFlashBag()->add( 'error', $this->get('tr
->trans('one_locale_per_pagemeta_only', array(), 'BpehNestablePageBundle') );
                        } else {
                                $em->persist( $pageMeta );
                                $em->flush();
                                return $this->redirectToRoute( 'bpeh_pagemeta_show', array( 'id' =
a->getId() ) );
                        }
                }

                return $this->render($this->pagemeta_view_new, array(
                        'pageMeta' => $pageMeta,
                        'form' => $form->createView(),
                ));

        }

        /**
         * Finds and displays a PageMeta entity.
         *
         * @Route("/{id}", name="bpeh_pagemeta_show")
         * @Method("GET")
         */
        public function showAction(Request $request)
        {
                $em = $this->getDoctrine()->getManager();

                $pageMeta = $em->getRepository($this->entity_meta)->find($request->get('id'));

                $deleteForm = $this->createDeleteForm($pageMeta);

                return $this->render($this->pagemeta_view_show, array(
                        'pageMeta' => $pageMeta,
                        'delete_form' => $deleteForm->createView(),
                ));
        }

        /**
         * Displays a form to edit an existing PageMeta entity.
         *
         * @Route("/{id}/edit", name="bpeh_pagemeta_edit")
         * @Method({"GET", "POST"})
         */
        public function editAction(Request $request)
        {
                $em = $this->getDoctrine()->getManager();
                $pageMeta = $em->getRepository($this->entity_meta)->find($request->get('id'));
                $origId = $pageMeta->getPage()->getId();
                $origLocale = $pageMeta->getLocale();

                $deleteForm = $this->createDeleteForm($pageMeta);
                $editForm = $this->createForm($this->page_meta_form_type, $pageMeta);
                $editForm->handleRequest($request);

                if ($editForm->isSubmitted() && $editForm->isValid()) {

                        $error = false;

                        // if page and local is the same, dont need to check locale count
                        if ($origLocale == $pageMeta->getLocale() && $origId == $pageMeta->getPage
>getId()) {
                                // all good
                        }
                        elseif ( $em->getRepository( $this->entity_meta )->findPageMetaByLocale( $
eMeta->getPage(), $pageMeta->getLocale(), true ) ) {
                                $this->get('session')->getFlashBag()->add( 'error', $this->get('tr
->trans('one_locale_per_pagemeta_only', array(), 'BpehNestablePageBundle') );
                                $error = true;
                        }

                        // if everything is successful
                        if (!$error) {
                                $em->persist( $pageMeta );
                                $em->flush();
                                return $this->redirectToRoute( 'bpeh_pagemeta_edit', array( 'id' =
a->getId() ) );
                        }
                }

                return $this->render($this->pagemeta_view_edit, array(
                        'pageMeta' => $pageMeta,
                        'edit_form' => $editForm->createView(),
                        'delete_form' => $deleteForm->createView(),
                ));
        }

        /**
         * Deletes a PageMeta entity.
         *
         * @Route("/{id}", name="bpeh_pagemeta_delete")
         * @Method("DELETE")
         */
        public function deleteAction(Request $request)
        {
                $em = $this->getDoctrine()->getManager();
                $pageMeta = $em->getRepository($this->entity_meta)->find($request->get('id'));
                $form = $this->createDeleteForm($pageMeta);
                $form->handleRequest($request);

                if ($form->isSubmitted() && $form->isValid()) {
                        $em = $this->getDoctrine()->getManager();
                        $em->remove($pageMeta);
                        $em->flush();
                }

                return $this->redirectToRoute('bpeh_pagemeta_index');
        }

        /**
         * Creates a form to delete a PageMeta entity.
         *
         * @param PageMeta $pageMetum The PageMeta entity
         *
         * @return \Symfony\Component\Form\Form The form
         */
        private function createDeleteForm(PageMeta $pageMeta)
        {
                return $this->createFormBuilder()
                            ->setAction($this->generateUrl('bpeh_pagemeta_delete', array('id' \
=> $pageMeta->getId())))
                            ->setMethod('DELETE')
                            ->getForm()
                        ;
        }
}
```

We also need to refactor PageMetaRepository because findPageMetaByLocale can now return either an object or scalar value.

```
# vendor/bpeh/nestable-page-bundle/Repository/PageMetaRepository.php

namespace Bpeh\NestablePageBundle\Repository;

use Bpeh\NestablePageBundle\Model\PageBase;

/**
 * PageMetaRepository
 *
 * This class was generated by the Doctrine ORM. Add your own custom
 * repository methods below.
 */
class PageMetaRepository extends \Doctrine\ORM\EntityRepository {

    /**
     * @param PageBase $page
     * @param $locale
     * @param bool $count
     * @return mixed
     */
        public function findPageMetaByLocale( PageBase $page, $locale, $count = false )\
 {

                $qb = $this->createQueryBuilder( 'pm' );

                if ( $count ) {
                        $qb->select( 'count(pm.id)' );
                }

                $query = $qb->where( 'pm.locale = :locale' )
              ->andWhere( 'pm.page = :page' )
              ->setParameter( 'locale', $locale )
              ->setParameter( 'page', $page )
              ->getQuery();

                if ( $count ) {
                        return $query->getSingleScalarResult();
                }

                return $query->getOneOrNullResult();

        }
}
```

There are other stuff to be done

- Create the translations.
- Move all the related views from app/resources/views to vendor/bpeh/nestable-page-bundle/views
- Update functional tests.

Once you are happy with it, give it a new tag and commit your changes again.

The bundle is now ready to be extended.

Extending BpehNestablePageBundle

To make things easy, I've created a demo bundle[137] and you can install the demo bundle and test out it for yourself.

Let us extend BpehNestablePageBundle by copying the PageTestBundle.

```
-> cd src/AppBundle
-> cp ../../vendor/bpeh/nestable-page-bundle/PageTestBundle/PageTestBundle.php .
# let us name it page bundle
-> mv PageTestBundle.php Page.php
-> cp ../../vendor/bpeh/nestable-page-bundle/PageTestBundle/Controller/*.php Con\
troller/
-> cp ../../vendor/bpeh/nestable-page-bundle/PageTestBundle/Entity/*.php Entity/
-> cp ../../vendor/bpeh/nestable-page-bundle/PageTestBundle/Repository/*.php Rep\
ository/
-> cp -a ../../vendor/bpeh/nestable-page-bundle/PageTestBundle/Form .
-> cp ../../vendor/bpeh/nestable-page-bundle/PageTestBundle/DataFixtures/ORM/Loa\
dPageData.php DataFixtures/ORM/
```

Let us call this bundle PageBundle to keep it simple.

[137]https://github.com/bernardpeh/NestablePageBundle

```
# src/AppBundle/Page.php

namespace AppBundle;

use Symfony\Component\HttpKernel\Bundle\Bundle;

class Page extends Bundle
{
        // use a child bundle
        public function getParent()
        {
                return 'BpehNestablePageBundle';
        }
}
```

Let us configure the Entities

```
# src/AppBundle/Entity/Page.php

namespace AppBundle\Entity;

use Bpeh\NestablePageBundle\Model\PageBase;
use Doctrine\ORM\Mapping as ORM;

/**
 * Page
 *
 * @ORM\Table(name="page")
 * @ORM\Entity(repositoryClass="AppBundle\Repository\PageRepository")
 * @ORM\HasLifecycleCallbacks()
 */
class Page extends PageBase
{
    /**
     * @var integer
     *
     * @ORM\Column(name="id", type="integer")
     * @ORM\Id
     * @ORM\GeneratedValue(strategy="AUTO")
     */
    protected $id;

    /**
     * Get id
     *
     * @return integer
     */
    public function getId()
    {
        return $this->id;
    }

}
```

and PageMeta.php

```
# src/AppBundle/Entity/PageMeta.php

namespace AppBundle\Entity;

use Bpeh\NestablePageBundle\Model\PageMetaBase;
use Doctrine\ORM\Mapping as ORM;

/**
 * PageMeta
 *
 * @ORM\Table(name="pagemeta")
 * @ORM\Entity(repositoryClass="AppBundle\Repository\PageMetaRepository")
 * @ORM\HasLifecycleCallbacks()
 */
class PageMeta extends PageMetaBase
{
    /**
     * @var integer
     *
     * @ORM\Column(name="id", type="integer")
     * @ORM\Id
     * @ORM\GeneratedValue(strategy="AUTO")
     */
    protected $id;

    /**
     * Get id
     *
     * @return integer
     */
    public function getId()
    {
        return $this->id;
    }

}
```

Let us update the PageRepository.php

```
# src/AppBundle/Repository/PageRepository.php

namespace AppBundle\Repository;

use Bpeh\NestablePageBundle\Repository\PageRepository as BasePageRepository;

/**
 * PageRepository
 *
 */
class PageRepository extends BasePageRepository
{

}
```

and PageMetaRepository.php

```
# src/AppBundle/Repository/PageRepository.php

namespace AppBundle\Repository;

use Bpeh\NestablePageBundle\Repository\PageMetaRepository as BasePageMetaReposit\
ory;

/**
 * PageRepository
 *
 */
class PageMetaRepository extends BasePageMetaRepository
{

}
```

Let us update the PageController to have a route which is easier to use

```php
# src/AppBundle/Controller/PageController.php

namespace AppBundle\Controller;

use Bpeh\NestablePageBundle\Controller\PageController as BaseController;
use Sensio\Bundle\FrameworkExtraBundle\Configuration\Route;

/**
 * Page controller.
 * @Route("/page")
 */
class PageController extends BaseController
{

}
```

Now PageMetaController.php

```php
# src/AppBundle/Controller/PageMetaController.php
namespace AppBundle\Controller;

use Bpeh\NestablePageBundle\Controller\PageMetaController as BaseController;
use Sensio\Bundle\FrameworkExtraBundle\Configuration\Route;

/**
 * PageMeta controller.
 * @Route("/pagemeta")
 */
class PageMetaController extends BaseController
{

}
```

Its time to update the basic forms

```
# src/AppBundle/Form/PageType.php

namespace AppBundle\Form;

use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\OptionsResolver\OptionsResolver;
use Bpeh\NestablePageBundle\Form\PageType as BasePageType;

class PageType extends BasePageType
{

        /**
         * @param FormBuilderInterface $builder
         * @param array $options
         */
        public function buildForm(FormBuilderInterface $builder, array $options)
        {
                parent::buildForm($builder,$options);
        }

        /**
         * @param OptionsResolver $resolver
         */
        public function configureOptions(OptionsResolver $resolver)
        {
                $resolver->setDefaults(array(
                        'data_class' => 'AppBundle\Entity\Page',
                ));
        }
}
```

and PageMetaType.php

```
# src/AppBundle/Form/PageMetaType.php

namespace AppBundle\Form;

use Symfony\Component\OptionsResolver\OptionsResolver;
use Symfony\Component\Form\FormBuilderInterface;
use Bpeh\NestablePageBundle\Form\PageMetaType as BasePageMetaType;

class PageMetaType extends BasePageMetaType
```

```
{

        /**
         * @param FormBuilderInterface $builder
         * @param array $options
         */
        public function buildForm(FormBuilderInterface $builder, array $options)
        {
                parent::buildForm($builder,$options);
        }

        /**
         * @param OptionsResolver $resolver
         */
        public function configureOptions(OptionsResolver $resolver)
        {
                $resolver->setDefaults(array(
                        'data_class' => 'AppBundle\Entity\PageMeta'
                ));
        }
}
```

Let us confirm the new routes are working…

```
-> ./scripts/console debug:router | grep page
     bpeh_page                        GET        ANY      ANY    /page/
     bpeh_page_list                   GET        ANY      ANY    /page/list
     bpeh_page_reorder                POST       ANY      ANY    /page/reorder
     bpeh_page_new                    GET|POST   ANY      ANY    /page/new
     bpeh_page_show                   GET        ANY      ANY    /page/{id}
     bpeh_page_edit                   GET|POST   ANY      ANY    /page/{id}/edit
     bpeh_page_delete                 DELETE     ANY      ANY    /page/{id}
     bpeh_pagemeta_index              GET        ANY      ANY    /pagemeta/
     bpeh_pagemeta_new                GET|POST   ANY      ANY    /pagemeta/new
     bpeh_pagemeta_show               GET        ANY      ANY    /pagemeta/{id}
     bpeh_pagemeta_edit               GET|POST   ANY      ANY    /pagemeta/{id}/\
edit
     bpeh_pagemeta_delete             DELETE     ANY      ANY    /pagemeta/{id}
```

Looks good. Its time to update config.yml

```
# app/config/config.yml
...
    orm:
        auto_generate_proxy_classes: "%kernel.debug%"
        naming_strategy: doctrine.orm.naming_strategy.underscore
        auto_mapping: true
        resolve_target_entities:
            Bpeh\NestablePageBundle\Model\PageBase: AppBundle\Entity\Page
            Bpeh\NestablePageBundle\Model\PageMetaBase: AppBundle\Entity\PageMeta
...
# Nestable Page Configuration
bpeh_nestable_page:
    page_entity: AppBundle\Entity\Page
    pagemeta_entity: AppBundle\Entity\PageMeta
    page_form_type: AppBundle\Form\PageType
    pagemeta_form_type: AppBundle\Form\PageMetaType
    # Customise the template if you want.
    # page_view_list: YourBundle:list.html.twig
    # page_view_new: YourBundle:new.html.twig
    # page_view_edit: YourBundle:edit.html.twig
    # page_view_show: YourBundle:show.html.twig
    # pagemeta_view_index: YourBundle:index.html.twig
    # pagemeta_view_new: YourBundle:new.html.twig
    # pagemeta_view_edit: YourBundle:edit.html.twig
    # pagemeta_view_show: YourBundle:show.html.twig
```

Remember to clean up the routes.

```
# app/config/routing.yml

# remove these
# nestable_page:
#    resource: "@PageTestBundle/Controller/"
#    type:     annotation
#    prefix:   /
```

Init the new bundle in AppKernel.php

```
# app/AppKernel.php

...
    new Bpeh\NestablePageBundle\BpehNestablePageBundle(),
    new AppBundle\Page()
...
```

Remember to update the data fixtures.

```
# src/AppBundle/DataFixtures/ORM/LoadPageData.php

use Doctrine\Common\DataFixtures\AbstractFixture;
use Doctrine\Common\Persistence\ObjectManager;
use Doctrine\Common\DataFixtures\OrderedFixtureInterface;
use Symfony\Component\DependencyInjection\ContainerAwareInterface;
use Symfony\Component\DependencyInjection\ContainerInterface;
use AppBundle\Entity\Page;
use AppBundle\Entity\PageMeta;
...
```

There were schema changes. We have to update the sql so that we can deploy it easily if we need to. stash our work and load chapter_16 db.

```
-> git stash
-> git checkout chapter_16
-> ./scripts/resetapp
-> ./scripts/console doctrine:migrations:diff
-> git checkout mychapter_18
-> git stash pop
# we can commit everything at this stage
-> git add .
-> git commit
```

Reset the db again.

```
-> ./scripts/resetapp
```

Now go to http://songbird.app:8000/app_dev.php/page and make sure the new url should be working. run all the tests and make sure you didn't break anything.

```
-> ./scripts/runtest
```

I hope you are getting used to this… Its a pretty routine process once you get used to it.

Summary

In this chapter, we have created a new repo for the NestablePageBundle. We have updated composer to pull the bundle from the repo and auto-loaded it according to the PSR-4 standard. We learned the hard way of creating a non-extensible bundle with the wrong namespace and then mass renaming it again. Making the entities extensible was a massive job and required a lot of refactoring in our code. If you know you are creating a reusable bundle, its better to get the namespace correct and create it right from the start.

We have done so much to make NestablePageBundle as decoupled as possible. Still, there are lots of room for improvement. Was it worth the effort? Definitely! People can now install our bundle in their Symfony applications easily.

Exercises

- Delete the whole vendor directory and try doing a composer update. Did anything break?
- Update the functional test.

References

- Define relationship between abstract classes and interfaces[138]
- Short guide to licenses[139]
- Software licenses at a glance[140]
- Composer Schema[141]
- Composer versioning[142]

[138]http://symfony.com/doc/current/doctrine/resolve_target_entity.html
[139]http://www.smashingmagazine.com/2010/03/a-short-guide-to-open-source-and-similar-licenses
[140]http://tldrlegal.com
[141]https://getcomposer.org/doc/04-schema.md
[142]https://getcomposer.org/doc/articles/versions.md

Chapter 19: The Page Manager Part 2

In this chapter, we are going to integrate NestablePageBundle with EasyAdminBundle. We are also going to improve the cms by integrating a wysiwyg editor (ckeditor) and create a custom locale dropdown.

Define User Stories

19. Page Management

Story Id	As a	I	So that I
19.1	an admin	want to manage pages	update them anytime.
19.2	test1 user	don't want to manage pages	don't breach security

Story ID 19.1: As an admin, I want to manage pages, so that I can update them anytime.

Scenario Id	Given	When	Then
19.11	List Pages	I go to page list url	I can see 2 elements under the about slug
19.12	Show Contact Us Page	I go to contact_us page	I should see the word "contact_us" and the word "Created"
19.13	Reorder home	I drag and drop the home menu to under the about menu	I should see "reordered successfully message" in the response and see 3 items under the about menu
19.14	edit home page meta	I go to edit homepage url and update the menu title of "Home" to "Home1" and click update	I should see the menu updated to home1

19.15	Create and delete test page	go to page list and click “Add new page” and fill in details and click “Create” button, go to newly created test page and create 2 new test meta. Delete one testmeta and then delete the whole test page	I should see the first pagemeta being created and deleted. Then see the second testmeta being deleted when the page is being deleted.
19.16	Delete Contact Us Page	go to contact us page and click “delete”	I should see that the contact us page and its associate meta being deleted.
19.17	Create new page with existing locale	go to page list and click “Add new pagemeta” and fill in details, select locale as en, page as home and click “Create” button	I should see an exception.

Story ID 19.2: As test1 user, I don’t want to manage pages, so that I don’t breach security.

Scenario Id	Given**	**When**	**Then**
19.21	List pages	I go to the page management url	I should get a access denied message
19.22	show about us page	I go to show about us url	I should get a access denied message
19.23	edit about us page	I go to edit about us url	I should get a access denied message
19.24	List pagemeta	I go to list pagemeta url	I should get a access denied message

Adding new image field to PageMeta Entity

Let us add a new field called featuredImage to the PageMeta entity. We will configure Vich uploader to do the job.

```
# src/AppBundle/Entity/PageMeta.php

namespace AppBundle\Entity;

use Bpeh\NestablePageBundle\Model\PageMetaBase;
use Doctrine\ORM\Mapping as ORM;
use Vich\UploaderBundle\Mapping\Annotation as Vich;
use Symfony\Component\HttpFoundation\File\File;

/**
 * PageMeta
 *
 * @ORM\Table(name="pagemeta")
 * @ORM\Entity(repositoryClass="AppBundle\Repository\PageMetaRepository")
 * @ORM\HasLifecycleCallbacks()
 * @Vich\Uploadable
 *
 */
class PageMeta extends PageMetaBase
{
    /**
     * @var integer
     *
     * @ORM\Column(name="id", type="integer")
     * @ORM\Id
     * @ORM\GeneratedValue(strategy="AUTO")
     */
    protected $id;

    /**
     * @ORM\Column(type="string", length=255, nullable=true)
     * @var string
     */
    private $featuredImage;

    /**
     * @Vich\UploadableField(mapping="featured_image", fileNameProperty="feature\
dImage")
     * @var File
     */
    private $featuredImageFile;

    /**
     * Get id
     *
     * @return integer
     */
    public function getId()
    {
        return $this->id;
    }

    /**
     * @param File|null $image
     */
    public function setFeaturedImageFile(File $image = null)
    {
        $this->featuredImageFile = $image;

        if ($image) {
            $this->setModified(new \DateTime());
        }
    }

    /**
     * @return File
     */
    public function getFeaturedImageFile()
    {
        return $this->featuredImageFile;
    }

    /**
     * @param $image
     */
    public function setFeaturedImage($image)
    {
        $this->featuredImage = $image;
    }

    /**
     * @return string
     */
    public function getFeaturedImage()
    {
        return $this->featuredImage;
    }

    /**
     * @return string
     */
    public function __toString()
    {
        return $this->getLocale().': '.$this->getMenuTitle();
    }
}
```

Let us update config.yml

```
# app/config/config.yml
parameters:
    locale: en
    supported_lang: [ 'en', 'fr']
    admin_path: admin
    app.profile_image.path: /uploads/profiles
    app.featured_image.path: /uploads/featured_images
...
vich_uploader:
    db_driver: orm
    mappings:
        profile_images:
            uri_prefix: '%app.profile_image.path%'
            upload_destination: '%kernel.root_dir%/../web/uploads/profiles'
            namer: vich_uploader.namer_uniqid
        featured_image:
            uri_prefix: '%app.featured_image.path%'
            upload_destination: '%kernel.root_dir%/../web/uploads/featured_image\
s'
            namer: vich_uploader.namer_uniqid
```

Installing CKEditor

We will now install CKEditor

```
-> ./scripts/composer require egeloen/ckeditor-bundle
```

then enable the bundle

```
# app/AppKernel.php
class AppKernel extends Kernel
{
    public function registerBundles()
    {
        return array(
            // ...
            new Ivory\CKEditorBundle\IvoryCKEditorBundle(),
        );
    }
}
```

Integration with EasyAdminBundle

There is still some effort to get BpehNestablePageBundle integrate properly with EasyAdminBundle. The reason is because the big difference in controller logic between the 2 bundles.

Let us assume that we not going to use the PageController.php and PageMetaController.php except the reorder route

The new routing.yml as follows:

```
# app/config/routing.yml

admin:
  resource: "@AppBundle/Controller/AdminController.php"
  type:     annotation

locale:
  resource: "@AppBundle/Controller/LocaleController.php"
  type:     annotation

bpeh_page_reorder:
  path: /admin/reorder
  defaults:
    _controller: BpehNestablePageBundle:Page:reorder

# FOS user bundle default routing
```

```
fos_user_security:
  resource: "@FOSUserBundle/Resources/config/routing/security.xml"

fos_user_resetting:
  resource: "@FOSUserBundle/Resources/config/routing/resetting.xml"
  prefix: /resetting

easy_admin_bundle:
  resource: "@AppBundle/Controller/AdminController.php"
  type:     annotation
  prefix:   /%admin_path%
```

We now need to add actions to the AdminController. The new AdminController should look like this:

```
# src/AppBundle/Controller/AdminController.php

namespace AppBundle\Controller;

use JavierEguiluz\Bundle\EasyAdminBundle\Controller\AdminController as BaseAdmin\
Controller;
use JavierEguiluz\Bundle\EasyAdminBundle\Event\EasyAdminEvents;
use Sensio\Bundle\FrameworkExtraBundle\Configuration\Route;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;
use AppBundle\Entity\PageMeta;

class AdminController extends BaseAdminController
{
    /**
     * @Route("/dashboard", name="dashboard")
     *
     * @param Request $request
     * @return \Symfony\Component\HttpFoundation\Response
     *
     */
    public function dashboardAction(Request $request)
    {
        return $this->render('@EasyAdmin/default/dashboard.html.twig');
    }

    /**
```

```php
     * @return \Symfony\Component\HttpFoundation\Response
     */
    public function showUserAction()
    {
        $this->dispatch(EasyAdminEvents::PRE_SHOW);
        $id = $this->request->query->get('id');
        $easyadmin = $this->request->attributes->get('easyadmin');
        $entity = $easyadmin['item'];

        $fields = $this->entity['show']['fields'];

        if (!$this->isGranted('ROLE_SUPER_ADMIN')) {
            unset($fields['created']);
        }

        $deleteForm = $this->createDeleteForm($this->entity['name'], $id);

        return $this->render($this->entity['templates']['show'], array(
            'entity' => $entity,
            'fields' => $fields,
            'delete_form' => $deleteForm->createView(),
        ));
    }

    /**
     * when edit user action
     *
     * @return Response|\Symfony\Component\HttpFoundation\RedirectResponse|\Symf\
ony\Component\HttpFoundation\Response
     */
    protected function editUserAction()
    {
        $this->dispatch(EasyAdminEvents::PRE_EDIT);
        $id = $this->request->query->get('id');
        $easyadmin = $this->request->attributes->get('easyadmin');
        $entity = $easyadmin['item'];

        if ($this->request->isXmlHttpRequest() && $property = $this->request->qu\
ery->get('property')) {
            $newValue = 'true' === strtolower($this->request->query->get('newVal\
ue'));
            $fieldsMetadata = $this->entity['list']['fields'];

            if (!isset($fieldsMetadata[$property]) || 'toggle' !== $fieldsMetada\
ta[$property]['dataType']) {
                throw new \RuntimeException(sprintf('The type of the "%s" proper\
ty is not "toggle".', $property));
            }

            $this->updateEntityProperty($entity, $property, $newValue);

            return new Response((string)$newValue);
        }

        $fields = $this->entity['edit']['fields'];

        $editForm = $this->createEditForm($entity, $fields);
        if (!$this->isGranted('ROLE_SUPER_ADMIN')) {
            $editForm->remove('enabled');
            $editForm->remove('roles');
            $editForm->remove('locked');
        }

        $deleteForm = $this->createDeleteForm($this->entity['name'], $id);

        $editForm->handleRequest($this->request);
        if ($editForm->isValid()) {
            $this->preUpdateUserEntity($entity);
            $this->em->flush();

            $refererUrl = $this->request->query->get('referer', '');

            return !empty($refererUrl)
                ? $this->redirect(urldecode($refererUrl))
                : $this->redirect($this->generateUrl('easyadmin', array('action'\
 => 'show', 'entity' => $this->entity['name'], 'id' => $id)));
        }

        return $this->render($this->entity['templates']['edit'], array(
            'form' => $editForm->createView(),
            'entity_fields' => $fields,
            'entity' => $entity,
            'delete_form' => $deleteForm->createView(),
        ));
```

```
    }

    public function createNewUserEntity()
    {
        return $this->get('fos_user.user_manager')->createUser();
    }

    public function prePersistUserEntity($user)
    {
        $this->get('fos_user.user_manager')->updateUser($user, false);
    }

    public function preUpdateUserEntity($user)
    {
        $this->get('fos_user.user_manager')->updateUser($user, false);
    }

    /**
     * Show Page List page
     *
     * @return \Symfony\Component\HttpFoundation\Response
     */
    public function listPageAction()
    {
        $this->dispatch(EasyAdminEvents::PRE_LIST);

        $rootMenuItems = $this->getDoctrine()->getRepository('AppBundle\Entity\P\
age')->findParent();

        return $this->render('@EasyAdmin/Page/list.html.twig', array(
            'tree' => $rootMenuItems,
        ));
    }

    /**
     * save page meta
     *
     * @param PageMeta $pageMeta
     */
    public function prePersistPageMetaEntity(PageMeta $pageMeta)
    {
        if ( $this->em->getRepository('AppBundle\Entity\PageMeta')->findPageMeta\
ByLocale( $pageMeta->getPage(), $pageMeta->getLocale() ) ) {
            throw new \RuntimeException($this->get('translator')->trans('one_loc\
ale_per_pagemeta_only', array(), 'BpehNestablePageBundle') );
        }

    }

    /**
     * edit page meta
     *
     * @return Response|\Symfony\Component\HttpFoundation\RedirectResponse|\Symf\
ony\Component\HttpFoundation\Response
     */
    protected function editPageMetaAction()
    {
        $this->dispatch(EasyAdminEvents::PRE_EDIT);

        $id = $this->request->query->get('id');
        $easyadmin = $this->request->attributes->get('easyadmin');
        $entity = $easyadmin['item'];

        // get id before submission
        $pageMeta = $this->em->getRepository('AppBundle\Entity\PageMeta')->find(\
$id);
        $origId = $pageMeta->getPage()->getId();
        $origLocale = $pageMeta->getLocale();

        if ($this->request->isXmlHttpRequest() && $property = $this->request->qu\
ery->get('property')) {
            $newValue = 'true' === strtolower($this->request->query->get('newVal\
ue'));
            $fieldsMetadata = $this->entity['list']['fields'];

            if (!isset($fieldsMetadata[$property]) || 'toggle' !== $fieldsMetada\
ta[$property]['dataType']) {
                throw new \RuntimeException(sprintf('The type of the "%s" proper\
ty is not "toggle".', $property));
            }

            $this->updateEntityProperty($entity, $property, $newValue);

            return new Response((string) $newValue);
        }

        $fields = $this->entity['edit']['fields'];

        $editForm = $this->createEditForm($entity, $fields);
        $deleteForm = $this->createDeleteForm($this->entity['name'], $id);

        $editForm->handleRequest($this->request);
        if ($editForm->isValid()) {
            $this->dispatch(EasyAdminEvents::PRE_UPDATE, array('entity' => $enti\
ty));

            // if page and local is the same, dont need to check locale count
            if ($origLocale == $entity->getLocale() && $origId == $entity->getPa\
ge()->getId()) {
                // all good
            }
            elseif ( $this->em->getRepository('AppBundle\Entity\PageMeta')->find\
PageMetaByLocale( $pageMeta->getPage(), $pageMeta->getLocale(), true ) ) {
                throw new \RuntimeException($this->get('translator')->trans('one\
_locale_per_pagemeta_only', array(), 'BpehNestablePageBundle') );
            }

            $this->em->flush();

            $this->dispatch(EasyAdminEvents::POST_UPDATE, array('entity' => $ent\
ity));

            $refererUrl = $this->request->query->get('referer', '');

            return !empty($refererUrl)
                ? $this->redirect(urldecode($refererUrl))
                : $this->redirect($this->generateUrl('easyadmin', array('action'\
 => 'list', 'entity' => $this->entity['name'])));
        }

        $this->dispatch(EasyAdminEvents::POST_EDIT);

        return $this->render($this->entity['templates']['edit'], array(
            'form' => $editForm->createView(),
            'entity_fields' => $fields,
            'entity' => $entity,
            'delete_form' => $deleteForm->createView(),
        ));
    }
}
```

Let us create the list view. We have to recreate it because we are extending it from the easyadmin layout instead.

```
-> mkdir -p app/Resources/EasyAdminBundle/views/Page
-> touch app/Resources/EasyAdminBundle/views/Page/list.html.twig
```

and the contents of list.html.twig "‘ # app/Resources/EasyAdminBundle/views/Page/list.html.twig

```
{{ parent() }}
<link rel="stylesheet" href="{{ asset('bundles/bpehnestablepage/css/styles.css '\
) }}">
```

```
{{ parent() }}
<script src="{{ asset('bundles/bpehnestablepage/js/jquery.nestable.js') }}"></sc\
ript>
<script>
    $(function() {

        var before = null, after = null;

        $('.dd').nestable({
            afterInit: function ( event ) { }
        });

        $('.dd').nestable('collapseAll');
        before = JSON.stringify($('.dd').nestable('serialize'));
        $('.dd').on('dragEnd', function(event, item, source, destination, positi\
on) {

            id = item.attr('data-id');
            parentId = item.closest('li').parent().closest('li').attr('data-id');

            // if parent id is null of if parent id and id is the same, it is th\
e top level.
            parentId = (parentId == id || typeof(parentId)  === "undefined") ?  \
'' : parentId;
```

```

            after = JSON.stringify($('.dd').nestable('serialize'));

            token = '{{ csrf_token("bpeh_page_reorder") }}';

            if (before != after) {
                $.ajax({
                    type: "POST",
                    url: "{{ path('bpeh_page_reorder') }}",
                    data: { id: id, parentId: parentId, position: position, csrf\
: token },
                    success: function (data, dataType) {
                        if (data.success) {
                            $('.alert').addClass('alert-success');
                        }
                        else {
                            $('.alert').addClass('alert-danger');
                        }
                        $('.alert').html(data.message);
                        $('.alert').fadeTo( 0 , 1, function() {});
                        $('.alert').fadeTo( 4000 , 0, function() {});
                    },

                    error: function (XMLHttpRequest, textStatus, errorThrown) {
                        console.log(XMLHttpRequest);
                    }
                });
                before = after;
            }
        });
    });
</script>
```

```
<div class="alert alert-dismissable">
    {{ 'flash_reorder_instructions' | trans({}, 'BpehNestablePageBundle') }}
</div>

<button type="button" onclick="$('.dd').nestable('expandAll')">{{ 'expand_all'|t\
rans({}, 'BpehNestablePageBundle') }}</button>
<button type="button" onclick="$('.dd').nestable('collapseAll')">{{ 'collapse_al\
l'|trans({}, 'BpehNestablePageBundle') }}</button>
<button onclick=window.location="{{ path('easyadmin') }}?entity=Page&action=new"\
>{{ 'new_page'|trans({}, 'BpehNestablePageBundle') }}</button>
        <button onclick=window.location="{{ path('easyadmin') }}?entity=PageMeta&action\
=new">{{ 'new_pagemeta'|trans({}, 'BpehNestablePageBundle') }}</button>
<div id="nestable" class="dd">
    <ol class="dd-list">
        {% include "@BpehNestablePage/Page/tree.html.twig" with { 'tree':tree } \
%}
    </ol>
</div>
```

```
and don't forget about tree.html.twig
```

# app/Resources/EasyAdminBundle/views/Page/tree.html.twig

```
<li class='dd-item' data-id='{{ v.getId() }}'>
    <div class='dd-handle'>
        <a class="dd-nodrag" href="{{ path('easyadmin') }}?entity={{ _entity_con\
fig.name }}&action=edit&{{ _entity_config.primary_key_field_name }}={{ v.getId()\
 }}">{{ v.getSlug() }}</a>
    </div>

    {% set children = v.getChildren()|length %}
    {% if children > 0 %}
        <ol class='dd-list'>
            {% include "@EasyAdmin/BpehNestablePageBundle:Page:tree.html.twig" w\
ith { 'tree':v.getChildren() } %}
        </ol>
    {% endif %}
</li>
```

```
Finally - the easyadmin config. We do not want to display the pagemeta menu.
```

# app/config/easyadmin/design.yml

easy_admin: design: brand_color: '#337ab7' assets: css: - /bundles/app/css/style.css menu: - { entity: 'User', icon: 'user' } - { entity: 'Page', icon: 'file' } - { entity: 'UserLog', icon: 'database' } "'

and

```
# app/config/easyadmin/page.yml

easy_admin:
    entities:
        Page:
            class: AppBundle\Entity\Page
            label: admin.link.page_management
            # for new page
            new:
                fields:
                  - slug
                  - isPublished
                  - sequence
                  - parent
            edit:
                fields:
                  - slug
                  - isPublished
                  - sequence
                  - parent
                  - pageMetas
            show:
                fields:
                  - id
                  - slug
                  - isPublished
                  - sequence
                  - parent
                  - modified
                  - created
                  - pageMetas
            list:
                actions: ['show', 'edit', 'delete']
                fields:
```

```
         - id
         - slug
         - isPublished
         - sequence
         - parent
         - modified
    PageMeta:
        class: AppBundle\Entity\PageMeta
        form:
          fields:
            - page_title
            - menu_title
            - { property: 'locale', type: 'AppBundle\Form\LocaleType' }
            - { type: 'divider' }
            - { property: 'featuredImageFile', type: 'vich_image' }
            - { property: 'short_description', type: 'ckeditor' }
            - { property: 'content', type: 'ckeditor' }
            - page
```

Noticed the new field type we have used, ie ckeditor, vich_image, and AppBundleFormLocaleType. EasyAdminBundle has internal support for ckeditor and vich_image but AppBundleFormLocaleType is our own custom form selector which will be discussed in the next section.

Creating Custom Locale Selector Form Type

If you are looking at the pagemeta page, say http://songbird.app:8000/app_dev.php/admin/?entity=PageMeta&action for example, you should have noticed by now that user can enter anything under the locale textbox. What if we want to load only the languages that we defined in the config file (ie, english and french)? It is a good idea to create our own dropdown.

```
# src/AppBundle/Form/LocaleType.php

namespace AppBundle\Form;

use Symfony\Component\Form\AbstractType;
use Symfony\Component\OptionsResolver\OptionsResolver;
use Symfony\Component\Form\Extension\Core\Type\ChoiceType;

class LocaleType extends AbstractType
{
    private $localeChoices;
```

```php

    public function __construct(array $localeChoices)
    {
            foreach ($localeChoices as $v) {
            $this->localeChoices[$v] = $v;
        }
    }

    public function configureOptions(OptionsResolver $resolver)
    {
        $resolver->setDefaults(array(
            'choices' => $this->localeChoices,
        ));
    }

    public function getParent()
    {
        return ChoiceType::class;
    }

}
```

The array localChoices is passed into the constructor. This class can be lazy loaded if we define it in service.yml

```yaml
# src/AppBundle/Resources/config/services.yml
...
  app.form.type.locale:
    class: AppBundle\Form\LocaleType
    arguments:
      - "%supported_lang%"
    tags:
      - { name: form.type }
...
```

See how we pass the supported_lang config variable into the class? Now, go to any pagemeta new or edit page (ie http://songbird.app:8000/app_dev.php/admin/?entity=PageMeta&action=new for example) and you should see the locale dropdown updated to only 2 enties.

Let us update the translation files

```
# src/AppBundle/Resources/translations/app.en.xlf
...
<trans-unit id="7">
    <source>admin.page_management</source>
    <target>Page Management</target>
</trans-unit>
..
```

and the french version

```
# src/AppBundle/Resources/translations/app.fr.xlf
...
<trans-unit id="7">
    <source>admin.page_management</source>
    <target>Gestion de la page</target>
</trans-unit>
...
```

There were db changes. remember to run doctrine migrations

```
-> ./scripts/console doctrine:migrations:diff
```

Update BDD Tests (Optional)

Let us create the cest files,

```
-> ./vendor/bin/codecept generate:cest -c src/AppBundle acceptance As_An_Admin/I\
WantToManagePages
-> ./vendor/bin/codecept generate:cest -c src/AppBundle acceptance As_Test1_User\
/IDontWantToManagePages
```

Create the test cases from the scenarios above and make sure all your tests passes before moving on.

Remember to commit all your code before moving on to the next chapter.

Summary

In this chapter, we have extended our NestablePageBundle in EasyAdmin. We have installed CKEditor in our textarea and created a customised locale dropdown based on values from our config.yml file. Our CMS is looking more complete now.

Exercises

- From the debug toolbar, update the missing translations.
- TinyMCE is also a widely used WYSIWYG editor. How do you integrate it in Sonata Media?
- What if you want to add a new user field to the Page Management System? What is going to happen to the page if the user is deleted?
- Can you make inserting pagemeta easier for every new page added? This just shows how much thought one person need to put when creating a software.

References

- Create custom form type[143]
- EasyAdmin Templating[144]
- CKEditor Bundle[145]
- Adding Wysiwyg Editor[146]

[143]http://symfony.com/doc/current/cookbook/form/create_custom_field_type.html
[144]https://github.com/javiereguiluz/EasyAdminBundle/blob/master/Resources/doc/book/3-list-search-show-configuration.md
[145]https://github.com/egeloen/IvoryCKEditorBundle
[146]https://github.com/javiereguiluz/EasyAdminBundle/blob/master/Resources/doc/tutorials/wysiwyg-editor.md

Chapter 20: The Front View

Going to "http://songbird.app:8000/" has nothing at the moment because we have so far been focusing on the the admin area and not touched the frontend. In this chapter, we will create an automatic route based on the slug and display the frontend view when the slug matches. Any route that matches "/" and "/home" will be using the index template while the rest of the pages will be using the view template.

We will create a simple home and subpages using bootstrap and use smartmenus javascript library[147] to create the top menu which will render the the submenus as well.

Lastly, we'll add a language toggle so that the page can render different languages easily. The menu and page content will be rendered based on the toggled language. To get the menu to display different languages, we will create a custom twig function (an extension called MenuLocaleTitle).

Define User Stories

20. Frontend

Story Id	As a	I	So that I
20.1	test3 user	want to browse the frontend	I can get the information I want.

Story ID 20.1: As test3 user, I want to browse the frontend, so that I can get the information I want.

Scenario Id	Given	When	Then
20.11	Home page is working	I go to the / or /home	I can see the jumbotron class and the text "SongBird CMS Demo"
20.12	Menus are working	I mouseover the about menu	I should see 2 menus under the about menu
20.13	Subpages are working	I click on contact memu	I should see the text "This project is hosted in"
20.14	Login menu is working	I click on login memu	I should see 2 menu items only

[147] http://www.smartmenus.org/

Creating the Frontend

Let create a new frontend controller

```
# src/AppBundle/Controller/FrontendController.php

namespace AppBundle\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\Controller;
use Sensio\Bundle\FrameworkExtraBundle\Configuration\Route;
use Sensio\Bundle\FrameworkExtraBundle\Configuration\Template;
use Symfony\Component\HttpFoundation\Request;
use Sensio\Bundle\FrameworkExtraBundle\Configuration\Method;

/**
 * Class FrontendController
 * @package AppBundle\Controller
 */
class FrontendController extends Controller
{
    /**
     * @Route("/{slug}", name="app_frontend_index", requirements = {"slug" = "^(\
(|home)$)"})
     * @Template()
     * @Method("GET")
     * @param Request $request
     *
     * @return array
     */
    public function indexAction(Request $request)
    {
        $slug = $request->get('_route_params')['slug'];
        $slug = ($slug) ?: 'home';
        $page = $this->getDoctrine()->getRepository('AppBundle:Page')->findOneBy\
Slug($slug);
        $pagemeta = $this->getDoctrine()->getRepository('AppBundle:PageMeta')->f\
indPageMetaByLocale($page, $request->getLocale());
        $rootMenuItems = $this->getDoctrine()->getRepository('AppBundle:Page')->\
findParent();
        return array(
            'pagemeta' => $pagemeta,
            'tree' => $rootMenuItems,
        );
    }
    /**
     * @Route("/{slug}", name="app_frontend_view")
     * @Template()
     * @Method("GET")
     */
    public function pageAction(Request $request)
    {
        $page = $this->getDoctrine()->getRepository('AppBundle:Page')->findOneBy\
Slug($request->get('_route_params')['slug']);
        $pagemeta = $this->getDoctrine()->getRepository('AppBundle:PageMeta')->f\
indPageMetaByLocale($page, $request->getLocale());
        $rootMenuItems = $this->getDoctrine()->getRepository('AppBundle:Page')->\
findParent();
        return array(
            'pagemeta' => $pagemeta,
            'tree' => $rootMenuItems,
        );
    }
}
```

All routing magic can be done with the @Route annotation (we can even use regex as shown in the indexAction). With the new @Template annotation, the action just need to return an array rather than a response. With the new routes added, we will move the frontend routes to the last priority, so routes like /login will be executed first.

```
# app/config/routing.yml
...
fos_user_security:
  resource: "@FOSUserBundle/Resources/config/routing/security.xml"

fos_user_resetting:
  resource: "@FOSUserBundle/Resources/config/routing/resetting.xml"
  prefix: /resetting

frontend:
  resource: "@AppBundle/Controller/FrontendController.php"
  type:     annotation
```

Let us update the frontend base view.

```
# src/AppBundle/Resources/Views/base.html.twig

<!DOCTYPE HTML>
<html lang="en-US">
<head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>{% block title %}{% endblock %}</title>
    {% block stylesheets %}
        <link href="{{ asset('minified/css/styles.css') }}" rel="stylesheet" />
    {% endblock %}
</head>
<body>

{% set urlPrefix = (app.environment == 'dev') ? '/app_dev.php' : '' %}

{% block body %}

    <div class="container">
        {% block topnav %}
            <ul id="top_menu" class="sm sm-clean">
                <li id="page_logo">
                    <a href="{{ urlPrefix }}" alt="songbird">
                        <img src="{{ asset('bundles/app/images/logo.png') }}" cl\
ass="center-block img-responsive" alt="Songbird" />
                    </a>
                </li>
                {% if tree is defined %}
                    {% include "AppBundle:Frontend:tree.html.twig" with { 'tree'\
:tree } %}
                {% endif %}
                <li id="frontend_lang_toggle">
                    <select id="lang" name="lang">
                        {% for lang in supported_lang %}
                            <option value="{{ lang }}">{{ lang }}</option>
                        {% endfor %}
                    </select>
                </li>
                <li id="login_link">
                    {% if is_granted("IS_AUTHENTICATED_REMEMBERED") %}
                        <a href="{{ path('fos_user_security_logout') }}">
```

```
                            {{ 'layout.logout'|trans({}, 'FOSUserBundle') }}
                        </a>
                    {% else %}
                        <a href="{{ path('fos_user_security_login') }}">
                            {{ 'layout.login'|trans({}, 'FOSUserBundle') }}
                        </a>
                    {% endif %}
                </li>
            </ul>

        {% endblock %}

        <div class="clearfix vspace"></div>

        {% block content %}{% endblock %}

        {% block footer %}
            <hr />
            <footer>
                <p class="text-center">© Songbird {{ "now" | date("Y")}}</p>
            </footer>
        {% endblock %}
    </div>

{% endblock %}

{% block script %}
    <script src="{{ asset('minified/js/javascript.js') }}"></script>
    <script>
        $(function() {
            $('#top_menu').smartmenus();
            // select the box based on locale
            $('#lang').val('{{ app.request.getLocale() }}');
            // redirect user if user change locale
            $('#lang').change(function() {
                window.location='{{ urlPrefix }}'+$(this).val()+'/locale';
            });
        });
    </script>
{% endblock %}

</body>
```

```
</html>
```

We will now create a homepage view.

```
# app/Resources/AppBundle/views/Frontend/index.html.twig

{% extends "base.html.twig" %}

{% block title %}
        {{ pagemeta.getPageTitle() }}
{% endblock %}

{% block content %}
{% if pagemeta is not null %}
        <div class="jumbotron">
                <h1 class="text-center">{{ pagemeta.getShortDescription() | raw }}</h1>
        </div>

        <div class="row-fluid">
                <div class="pull-left col-xs-3 col-md-3">
                        {%  if pagemeta.featuredImage is not null %}
                                <img class="featured_image" src="{{ vich_uploader_asset(pagemeta,
mageFile') }}" alt="{{ pagemeta.getShortDescription() | striptags }}"/>
                        {% endif %}
                </div>
                <div class="pull-right col-xs-9 col-md-9">
                        {{ pagemeta.getContent() | raw }}
                </div>
        </div>
        <div class="clearfix"></div>

{% endif %}
{% endblock %}
```

and pages view

```
# app/Resources/AppBundle/views/Frontend/page.html.twig

{% extends "base.html.twig" %}

{% block title %}
        {{ pagemeta.getPageTitle() }}
{% endblock %}

{% block content %}

        {% if pagemeta is not null %}
        <h2>{{ pagemeta.getShortDescription() | raw }}</h2>

        {%  if pagemeta.featuredImage is not null %}
                <img class="featured_image" src="{{ vich_uploader_asset(pagemeta, 'featuredIma\
geFile') }}" alt="{{ pagemeta.getShortDescription() | striptags }}"/>
        {% endif %}

        {{ pagemeta.getContent() | raw }}

        {% endif %}

{% endblock %}
```

and lastly, recursive view for the menu

```
# app/Resources/AppBundle/views/Frontend/tree.html.twig

{% for v in tree %}

    <li>
        <a href="{{ urlPrefix }}/{{ v.getSlug() }}">{{ getMenuLocaleTitle(v.getS\
lug()) }}</a>
        {% set children = v.getChildren()|length %}
        {% if children > 0 %}
            <ul>
                {% include "AppBundle:Frontend:tree.html.twig" with { 'tree':v.g\
etChildren() } %}
            </ul>
        {% endif %}
    </li>
{% endfor %}
```

Note the new getMenuLocaleTitle function in the twig. We will create a custom function usable by twig - Twig Extension.

```
# src/AppBundle/Twig/Extension/MenuLocaleTitle.php

namespace AppBundle\Twig\Extension;

/**
 * Twig Extension to get Menu title based on locale
 */
class MenuLocaleTitle extends \Twig_Extension
{
    /**
     * @var EntityManager
     */
    private $em;

        /**
         * @var $request
         */
        private $request;

        /**
         * MenuLocaleTitle constructor.
         *
         * @param $em
         * @param $request
         */
    public function __construct($em, $request)
    {
        $this->em = $em;
        $this->request = $request->getCurrentRequest();
    }

        /**
         * @return string
         */
    public function getName()
    {
        return 'menu_locale_title_extension';
    }

        /**
```

```
         * @return array
         */
    public function getFunctions()
    {
        return array(
            new \Twig_SimpleFunction('getMenuLocaleTitle', array($this, 'getMenu\
LocaleTitle'))
        );
    }

        /**
         * @param string $slug
         *
         * @return mixed
         */
    public function getMenuLocaleTitle($slug = 'home')
    {
            $locale = ($this->request) ? $this->request->getLocale() : 'en';
            $page = $this->em->getRepository('AppBundle:Page')->findOneBySlug($slug);
            $pagemeta = $this->em->getRepository('AppBundle:PageMeta')->findPageMetaByL\
ocale($page, $locale);

            return $pagemeta->getMenuTitle();
    }
}
```

we now need to make this class available as a service.

```
# src/AppBundle/Resources/config/services.yml
...
  menu_locale_title.twig_extension:
    class: AppBundle\Twig\Extension\MenuLocaleTitle
    arguments:
      - "@doctrine.orm.entity_manager"
      - "@request_stack"
    tags:
      - { name: twig.extension }
...
```

Since we have added a new top navbar, let us remove the SongBird logo from the login and password reset pages. Update the following pages as you see fit:

```
/songbird/symfony/app/Resources/FOSUserBundle/views/Resetting/checkEmail.html.tw\
ig
/songbird/symfony/app/Resources/FOSUserBundle/views/Resetting/request.html.twig
/songbird/symfony/app/Resources/FOSUserBundle/views/Resetting/reset.html.twig
/songbird/symfony/app/Resources/FOSUserBundle/views/Security/login.html.twig
```

Let us update bower.json to pull in smartmenus js.

```
-> bower install smartmenus --S
```

then make gulp to pull the libraries in

```
# gulpfile.js
...
// Minify JS
gulp.task('js', function () {
    return gulp.src(['bower_components/jquery/dist/jquery.js',
        'bower_components/bootstrap/dist/js/bootstrap.js',
        'bower_components/smartmenus/dist/jquery.smartmenus.js'])
        .pipe(concat('javascript.js'))
        .pipe(uglify())
        .pipe(sourcemaps.write('./'))
        .pipe(gulp.dest('web/minified/js'));
});

// Minify CSS
gulp.task('css', function () {
    return gulp.src([
        'bower_components/bootstrap/dist/css/bootstrap.css',
        'bower_components/smartmenus/dist/css/sm-core-css.css',
        'bower_components/smartmenus/dist/css/sm-clean/sm-clean.css',
        'src/AppBundle/Resources/public/less/*.less',
        'src/AppBundle/Resources/public/sass/*.scss',
        'src/AppBundle/Resources/public/css/*.css'])
        .pipe(gulpif(/[.]less/, less()))
        .pipe(gulpif(/[.]scss/, sass()))
        .pipe(concat('styles.css'))
        .pipe(uglifycss())
        .pipe(sourcemaps.write('./'))
        .pipe(gulp.dest('web/minified/css'));
});
...
```

Let us update the datafixtures as well.

```
# src/AppBundle/DataFixtures/ORM/LoadPageData.php

...
        $homemetaEN = new PageMeta();
        $homemetaEN->setPage($homepage);
        $homemetaEN->setMenuTitle('Home');
        $homemetaEN->setPageTitle('SongBird CMS Demo');
        $homemetaEN->setShortDescription('SongBird CMS Demo');
        $homemetaEN->setContent('<p>SongBird is a simple CMS built with popular \
bundles like FOSUserBundle and EasyAdminBundle.
            The CMS is meant to showcase Rapid Application Development with Symf\
ony.</p>');
        copy(__DIR__.'/images/home_en.png', __DIR__.'/../../../../web/uploads/fe\
atured_images/home_en.png');
        $homemetaEN->setFeaturedImage('home_en.png');
        $manager->persist($homemetaEN);

        $homemetaFR = new PageMeta();
        $homemetaFR->setPage($homepage);
        $homemetaFR->setMenuTitle('Accueil');
        $homemetaFR->setPageTitle('SongBird CMS Démo');
        $homemetaFR->setShortDescription('SongBird CMS Démo');
        $homemetaFR->setLocale('fr');
        $homemetaFR->setContent('<p>SongBird est un simple CMS construit avec de\
s faisceaux populaires comme FOSUserBundle et EasyAdminBundle.
            Le CMS est destinée à mettre en valeur Rapid Application Development\
 avec Symfony .</p>');
        copy(__DIR__.'/images/home_fr.png', __DIR__.'/../../../../web/uploads/fe\
atured_images/home_fr.png');
        $homemetaFR->setFeaturedImage('home_fr.png');
        $manager->persist($homemetaFR);
```

I’ve added new images to the homepage. The new images are in the src/AppBundle/DataFixtures/ORM/images folder. Feel free to get the images from there.

Lastly, let us update the stylesheets. We might as well update them in scss

```
# src/AppBundle/Resources/public/sass/styles.scss

// variables
$black: #000;
$white: #fff;
$radius: 6px;
$spacing: 20px;
$font_big: 16px;
$featured_image_width: 200px;

body {
    color: $black;
    background: $white !important;
    padding-top: $spacing;
}
.vspace {
    height: $spacing;
}
.skin-black {
    .logo {
        background-color: $black;
    }
    .left-side {
        background-color: $black;
    }
}
.sidebar {
    ul {
        padding-top: $spacing;
    }
    li {
        padding-top: $spacing;
    }
}
.admin_top_left {
    padding-top: $spacing 0 0 $spacing;
}
#top_menu {
    padding: $spacing 0 $spacing;

    #page_logo {
        padding: 0 $spacing;
        margin: -$spacing/2 0;
    }

    #login_link {
        float:right;
    }

    #frontend_lang_toggle {
        float: right;
        padding: $spacing/2 $spacing;
    }
}
.sm-clean {
    border-radius: $radius;

}
// admin area
#page_logo img {
    display: inline;
    max-height: 100%;
    max-width: 50%;
}
#page_menu li {
    line-height: $spacing;
    padding-top: $spacing;
}
.navbar-brand img {
    margin-top: -8px;
}

.form-signin {
    max-width: 330px;
    padding: 15px;
    margin: 0 auto;
    .form-signin-heading {
        margin-bottom: $spacing;
    }
    .checkbox {
        margin-bottom: $spacing;
        font-weight: normal;
    }
    .form-control {
        position: relative;
        height: auto;
        box-sizing: border-box;
        padding: $spacing;
        font-size: $font_big;
        &:focus {
            z-index: 2;
        }
    }
    input[type="email"] {
        border-bottom-right-radius: 0;
        border-bottom-left-radius: 0;
    }
    input[type="password"] {
        margin-bottom: $spacing;
        border-top-left-radius: 0;
        border-top-right-radius: 0;
    }
}
.form-control {
    margin-top: $spacing;
    margin-bottom: $spacing;
}

// frontend
.featured_image {
    margin: auto;
    display: block;
    width: $featured_image_width;
}
```

We no longer need our old .css files

```
-> git rm src/AppBundle/Resources/public/css/*
```

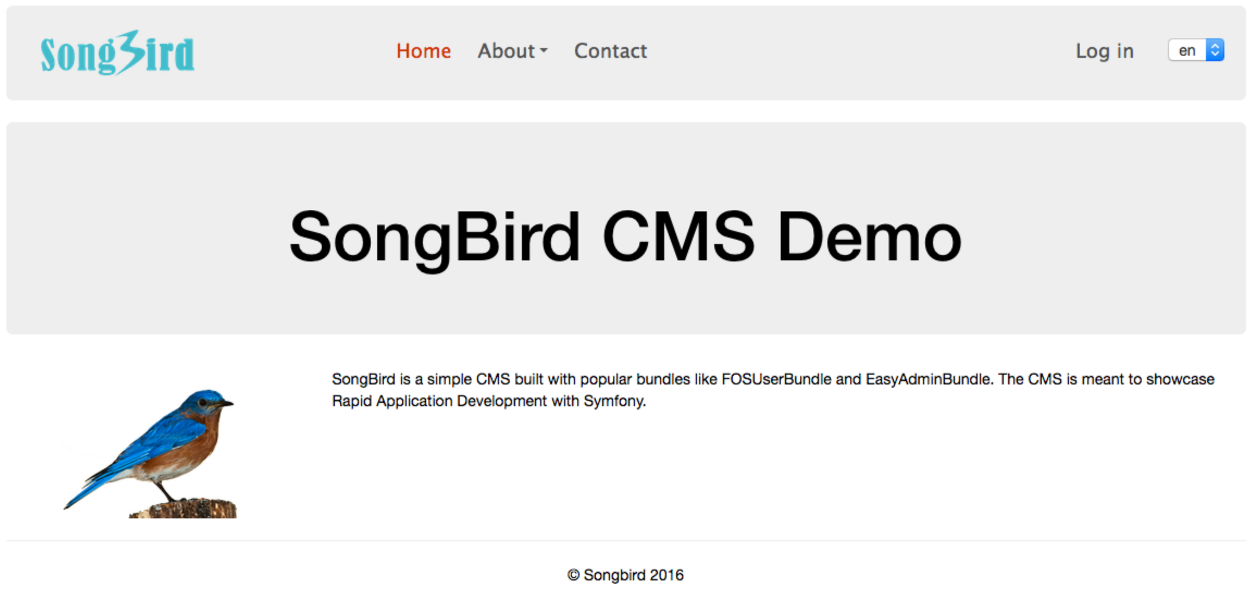
Now run gulp and refresh the homepage and everything should renders.

```
-> gulp
```

Remember to create the featured_images dir and reset the db if not done

```
-> mkdir -p web/uploads/featured_images
-> ./scripts/resetapp
```

Go to homepage and this should be the end result.



SongBird CMS Demo

SongBird is a simple CMS built with popular bundles like FOSUserBundle and EasyAdminBundle. The CMS is meant to showcase Rapid Application Development with Symfony.

© Songbird 2016

Update BDD (Optional)

Let us create the cest file:

```
-> vendor/bin/codecept generate:cest -c src/AppBundle acceptance As_Test3_User/I\
WantToViewTheFrontend
```

Write your test and make sure everything passes.

Summary

In this chapter, we have created the frontend controllers and views. We used smartmenus to render the menus and converted our css to sass. Finally, we wrote BDD tests to make sure our frontend renders correctly. The CMS is now complete.

Exercises

- Try extending the NestablePageBundle so that you can have multiple menus, say a top and bottom menu?
- One of the argument against using a language toggle is that it is bad for SEO. Language toggle can be good for usability. Can you think of a way to overcome the SEO issue?

References

- Controllers best practice[148]
- Smart Menus[149]
- Twig Extension[150]

[148]http://symfony.com/doc/current/best_practices/controllers.html
[149]http://www.smartmenus.org/
[150]http://symfony.com/doc/current/cookbook/templating/twig_extension.html

Chapter 21: Dependency Injection Revisited

Upon reflection of what we have covered in the last 20 chapters, I think there are lots of improvements that can be done. In particular, I feel that I wouldn't do justice to this book if I don't give an example of Compiler Pass[151].

This is an advance chapter. If you skipped all the chapters and came to this chapter by chance, I recommend you to read up DI and DIC before continuing.

In this chapter, I like to introduce 2 improvements to the CMS.

a) Simplifying config.yml

b) Adding Simple User Access Control to EasyAdminBundle.

Simplifying config.yml

Due to DI, the bundle Extension is called when the bundle is being initialised. The end result is a bunch of parameters and services that can be used and referenced throughout the application.

The app/config/config.yml is read by all bundle extensions so that relevant information relating to the bundle can be extracted. So far, there are many configuration parameters like fos, vich, doctrine …etc. To make the installation easier, we could move all these extra configuration to elsewhere so that developers don't have to worry about them when installing the CMS and it also makes the file looks cleaner.

The trick that does that is to implement the PrependExtensionInterface[152].

```
# src/AppBundle/DependencyInjection/AppExtension.php

namespace AppBundle\DependencyInjection;

use Symfony\Component\DependencyInjection\ContainerBuilder;
use Symfony\Component\Config\FileLocator;
use Symfony\Component\DependencyInjection\Extension\PrependExtensionInterface;
use Symfony\Component\HttpKernel\DependencyInjection\Extension;
use Symfony\Component\DependencyInjection\Loader\YamlFileLoader;
```

[151]http://symfony.com/doc/current/service_container/compiler_passes.html
[152]http://symfony.com/doc/current/bundles/prepend_extension.html

```

/**
 * This is the class that loads and manages your bundle configuration
 *
 * To learn more see {@link http://symfony.com/doc/current/cookbook/bundles/exte\
nsion.html}
 */
class AppExtension extends Extension implements PrependExtensionInterface
{
    /**
     * {@inheritdoc}
     */
    public function load(array $configs, ContainerBuilder $container)
    {
        $configuration = new Configuration();
        $config = $this->processConfiguration($configuration, $configs);

        $loader = new YamlFileLoader($container, new FileLocator(__DIR__ . '/../\
Resources/config'));
        $loader->load('services.yml');
    }

    /**
     * http://symfony.com/doc/current/bundles/prepend_extension.html
     */
    public function prepend(ContainerBuilder $container)
    {
        // doctrine config
        $doctrine = [];
        $doctrine['orm']['resolve_target_entities']['Bpeh\NestablePageBundle\Mod\
el\PageBase'] = 'AppBundle\Entity\Page';
        $doctrine['orm']['resolve_target_entities']['Bpeh\NestablePageBundle\Mod\
el\PageMetaBase'] = 'AppBundle\Entity\PageMeta';
        $container->prependExtensionConfig('doctrine', $doctrine);

        // fos config
        $fosuser = [];
        $fosuser['db_driver'] = 'orm';
        $fosuser['firewall_name'] = 'main';
        $fosuser['user_class'] = 'AppBundle\Entity\User';
        $fosuser['from_email']['address'] = 'admin@songbird.app';
        $fosuser['from_email']['sender_name'] = 'Songbird';
        $container->prependExtensionConfig('fos_user', $fosuser);

        # Nestable page config
        $page = [];
        $page['page_entity'] = 'AppBundle\Entity\Page';
        $page['pagemeta_entity'] = 'AppBundle\Entity\PageMeta';
        $page['page_form_type'] = 'AppBundle\Form\PageType';
        $page['pagemeta_form_type'] = 'AppBundle\Form\PageMetaType';
        $container->prependExtensionConfig('bpeh_nestable_page', $page);

        # Vich config
        $vich = [];
        $vich['db_driver'] = 'orm';
        $vich['mappings']['profile_images']['uri_prefix'] = '%app.profile_image.\
path%';
        $vich['mappings']['profile_images']['upload_destination'] = '%kernel.roo\
t_dir%/../web/uploads/profiles';
        $vich['mappings']['profile_images']['namer'] = 'vich_uploader.namer_uniq\
id';
        $vich['mappings']['featured_image']['uri_prefix'] = '%app.featured_image\
.path%';
        $vich['mappings']['featured_image']['upload_destination'] = '%kernel.roo\
t_dir%/../web/uploads/featured_images';
        $vich['mappings']['featured_image']['namer'] = 'vich_uploader.namer_uniq\
id';
        $container->prependExtensionConfig('vich_uploader', $vich);

    }
}
```

my config.yml then becomes like this:

```
# app/config/config.yml

imports:
    - { resource: parameters.yml }
    - { resource: security.yml }
    - { resource: services.yml }
    - {resource: easyadmin/ }

parameters:
    locale: en
    supported_lang: [ 'en', 'fr']
    admin_path: admin
    app.profile_image.path: /uploads/profiles
    app.featured_image.path: /uploads/featured_images

framework:
    #esi:             ~
    translator:      { fallbacks: ["%locale%"] }
    secret:          "%secret%"
    router:
        resource: "%kernel.root_dir%/config/routing.yml"
        strict_requirements: ~
    form:            ~
    csrf_protection: ~
    validation:      { enable_annotations: true }
    #serializer:      { enable_annotations: true }
    templating:
        engines: ['twig']
    default_locale:  "%locale%"
    trusted_hosts:   ~
    trusted_proxies: ~
    session:
        # handler_id set to null will use default session handler from php.ini
        handler_id:  ~
    fragments:       ~
    http_method_override: true

# Twig Configuration
twig:
    debug:            "%kernel.debug%"
    strict_variables: "%kernel.debug%"
    globals:
        supported_lang: '%supported_lang%'

# Doctrine Configuration
doctrine:
    dbal:
        driver:   pdo_mysql
        host:     "%database_host%"
        port:     "%database_port%"
        dbname:   "%database_name%"
        user:     "%database_user%"
        password: "%database_password%"
        charset:  UTF8
    orm:
        auto_generate_proxy_classes: "%kernel.debug%"
        naming_strategy: doctrine.orm.naming_strategy.underscore
        auto_mapping: true

# Swiftmailer Configuration
swiftmailer:
    transport: "%mailer_transport%"
    host:      "%mailer_host%"
    username:  "%mailer_user%"
    password:  "%mailer_password%"
    spool:     { type: memory }
```

Noticed that I could have moved more parameters over to the prepend function if I want to simplify the installation further.

Adding Simple Access Control to EasyAdminBundle

I still want to comment javiereguiluz[153] for creating the wonderful EasyAdminBundle[154]. As of current, the bundle doesn't support user permissions out of the box. I believe there might be plans to include this feature in the future as it is a widely requested feature.

As an exercise, let's say that we want to customise the bundle so that we can control access to certain parts of the admin area based on the user's role and we want to do that simply by changing the easyadmin yaml files.

Let us allow all authenticated users to access the admin area rather than just ROLE_USER.

```
# app/config/security.yml
...
  access_control:
      - { path: ^/login$, role: IS_AUTHENTICATED_ANONYMOUSLY }
      - { path: ^/resetting, role: IS_AUTHENTICATED_ANONYMOUSLY }
      - { path: ^/%admin_path%/, role: IS_AUTHENTICATED_FULLY }
```

The new design.yml should look like this:

[153] https://github.com/javiereguiluz

[154] https://github.com/javiereguiluz/EasyAdminBundle

```
# app/config/easyadmin/design.yml

easy_admin:
    design:
        brand_color: '#5493ca'
        assets:
            css:
                - /bundles/app/css/style.css
        menu:
          - { label: 'Dashboard', route: 'dashboard', default: true }
          - { entity: 'User', icon: 'user', role: ROLE_USER }
          - { entity: 'Page', icon: 'file', role: ROLE_ADMIN }
          - { entity: 'UserLog', icon: 'database', role: ROLE_ADMIN }
```

Noticed that we have added a new attribute called "role" to each menu item and the value (say "ROLE_ADMIN") means the mimimum permission level required to access that menu. In this case, everyone can see the dashboard, ROLE_USER and above can access the User link and only ROLE_-ADMIN can see the User and UserLog link.

We are going to do something similar for all the entities yaml, starting from the page entity

```
# app/config/easyadmin/page.yml

easy_admin:
    entities:
        Page:
            class: AppBundle\Entity\Page
            label: admin.link.page_management
            role: ROLE_ADMIN
            # for new page
            new:
                fields:
                  - slug
                  - isPublished
                  - sequence
                  - parent
            edit:
                fields:
                  - slug
                  - isPublished
                  - sequence
                  - parent
                  - pageMetas
            show:
                fields:
                  - id
                  - slug
                  - isPublished
                  - sequence
                  - parent
                  - modified
                  - created
                  - pageMetas
            list:
                actions: ['show', 'edit', 'delete']
                fields:
                  - id
                  - slug
                  - isPublished
                  - sequence
                  - parent
                  - modified
        PageMeta:
            class: AppBundle\Entity\PageMeta
            role: ROLE_ADMIN
            form:
              fields:
                - page_title
                - menu_title
                - { property: 'locale', type: 'AppBundle\Form\LocaleType' }
                - { type: 'divider' }
                - { property: 'featuredImageFile', type: 'vich_image' }
                - { property: 'short_description', type: 'ckeditor' }
                - { property: 'content', type: 'ckeditor' }
                - page
```

Now, the user entity:

```
# app/config/easyadmin/user.yml

easy_admin:
    entities:
        User:
            class: AppBundle\Entity\User
            label: admin.link.user_management
            role: ROLE_USER
            # for new user
            new:
                role: ROLE_ADMIN
                fields:
                  - username
                  - firstname
                  - lastname
                  - { property: 'plainPassword', type: 'repeated', type_options:\
 { type: 'Symfony\Component\Form\Extension\Core\Type\PasswordType', first_option\
s: {label: 'Password'}, second_options: {label: 'Repeat Password'}, invalid_mess\
age: 'The password fields must match.'}}
                  - { property: 'email', type: 'email', type_options: { trim: tr\
ue } }
                  - { property: 'imageFile', type: 'vich_image' }
                  - roles
                  - enabled
            edit:
                role: ROLE_ADMIN
                fields:
                - username
                - firstname
                - lastname
                - { property: 'plainPassword', type: 'repeated', type_options: {\
 type: 'Symfony\Component\Form\Extension\Core\Type\PasswordType', required: fals\
e, first_options: {label: 'Password'}, second_options: {label: 'Repeat Password'\
}, invalid_message: 'The password fields must match.'}}
                - { property: 'email', type: 'email', type_options: { trim: true\
 } }
                - { property: 'imageFile', type: 'vich_image' }
                - roles
                - enabled
            show:
                role: ROLE_ADMIN
                fields:
```

```
                - id
                - { property: 'image', type: 'image', base_path: '%app.profile_i\
mage.path%'}
                - username
                - firstname
                - lastname
                - email
                - roles
                - enabled
                - { property: 'last_login', type: 'datetime' }
                - modified
                - created
            list:
                role: ROLE_USER
                title: 'User Listing'
                actions: ['show']
                fields:
                  - id
                  - { property: 'image', type: 'image', base_path: '%app.profile\
_image.path%'}
                  - username
                  - email
                  - firstname
                  - lastname
                  - enabled
                  - roles
                  - { property: 'last_login', type: 'datetime' }
            delete:
                role: ROLE_ADMIN
```

and finally - userlog.yml.

```
# app/config/easyadmin/userlog.yml

easy_admin:
    entities:
        UserLog:
            class: AppBundle\Entity\UserLog
            label: admin.link.user_log
            role: ROLE_ADMIN
            show:
                actions: ['list', '-edit', '-delete']
            list:
                actions: ['show', '-edit', '-delete']
```

When parameters and services are created by the extension but not yet compiled in optimised DIC, there is a chance to manipulate them. Compiler Pass exists for this purpose.

Let us tell our AppBundle to initiate its compiler pass when it is loaded by the kernel.

```
# src/AppBundle/AppBundle.php

<?php

namespace AppBundle;

use Symfony\Component\HttpKernel\Bundle\Bundle;
use Symfony\Component\DependencyInjection\ContainerBuilder;
use AppBundle\DependencyInjection\Compiler\ConfigPass;

class AppBundle extends Bundle
{
        public function build(ContainerBuilder $container)
        {
                parent::build($container);
                $container->addCompilerPass(new ConfigPass());
        }
}
```

We have added a new compiler pass class called ConfigPass.php. Compiler Pass needs to extend the CompilerPassInterface.

```
# src/AppBundle/DependencyInjection/Compiler/ConfigPass.php

<?php

<?php

namespace AppBundle\DependencyInjection\Compiler;

use Symfony\Component\DependencyInjection\Compiler\CompilerPassInterface;
use Symfony\Component\DependencyInjection\ContainerBuilder;

class ConfigPass implements CompilerPassInterface
```

```
{
    public function process( ContainerBuilder $container ) {

        // $container->getParameterBag();
        // $container->getServiceIds();

        $config = $container->getParameter('easyadmin.config');

        // use menu to use IS_AUTHENTICATED_FULLY role by default if not set
        foreach($config['design']['menu'] as $k => $v) {
            if (!isset($v['role'])) {
                $config['design']['menu'][$k]['role'] = 'IS_AUTHENTICATED_FULLY';
            }
        }

        // update entities to use IS_AUTHENTICATED_FULLY role by default if not \
set
        foreach ($config['entities'] as $k => $v) {
            if (!isset($v['role'])) {
                $config['entities'][$k]['role'] = 'IS_AUTHENTICATED_FULLY';
            }
        }

        // update views to use entities role by default if not set
        foreach ($config['entities'] as $k => $v) {
            $views = ['new', 'edit', 'show', 'list', 'form', 'delete'];
            foreach ($views as $view) {
                if (!isset($v[$view]['role'])) {
                    $config['entities'][$k][$view]['role'] = $v['role'];
                }
            }
        }

        $container->setParameter('easyadmin.config', $config);

    }
}
```

What we have done here is to change the easyadmin.config parameter produced by the EasyAdminBundle. easyadmin.config is simply a bunch of arrays built based on the yaml config under app/config/easy_admin. Each for-loop adds a new key called “role” with the default “IS_AUTHENTICATED_FULLY” role if not specified by the config.

EasyAdmin dispatches lots of events. We were already subscribed to it.

```
# src/AppBundle/Resources/config/services.yml
  ...
  app.subscriber:
    class: AppBundle\EventListener\AppSubscriber
    arguments:
        - "@service_container"
    tags:
        - { name: kernel.event_subscriber }
```

We now need to add a bit more logic to the subscriber.

```
# src/AppBundle/EventListener/AppSubscriber.php

class AppSubscriber implements EventSubscriberInterface
{
    ...

    /**
     * show an error if user is not superadmin and tries to manage restricted st\
uff
     *
     * @param GenericEvent $event event
     * @return null
     * @throws AccessDeniedException
     */
    public function checkUserRights(GenericEvent $event)
        {

            // if super admin, allow all
            $authorization = $this->container->get('security.authorization_check\
er');
            $request = $this->container->get('request_stack')->getCurrentRequest\
()->query;

            if ($authorization->isGranted('ROLE_ADMIN')) {
                return;
            }

            $entity = $request->get('entity');
            $action = $request->get('action');
            $user_id = $request->get('id');

            // allow user to see and edit their own profile irregardless of perm\
issions
                    if ($entity == 'User') {
                        // if edit and show
                        if ($action == 'edit' || $action == 'show') {
                            // check user is himself
                            if ($user_id == $this->container->get('security.toke\
n_storage')->getToken()->getUser()->getId()) {
                                return;
                            }
                        }
                    }

            $config = $this->container->get('easyadmin.config.manager')->getBack\
endConfig();

            // check for permission for each action
            foreach ($config['entities'] as $k => $v) {
                if ($entity == $k && !$authorization->isGranted($v[$action]['rol\
e'])) {
                    throw new AccessDeniedException();
                }
            }
        }
    ...
```

We have triggered the checkUserRights function based on a few EasyAdmin events. We have allowed the logged in user to edit his own profile irregardless of role's permission. Then, the for-loop does the magic of allowing or denying user to access different parts of the admin area based on the role key in easyadmin.config.manager service.

Note that this will work only if our AdminController dispatches the events, ie

```
# src/AppBundle/Controller/AdminController.php

...
        /**
         * Show Page List page
         * @return \Symfony\Component\HttpFoundation\Response
         */
    public function listPageAction()
    {
            $this->dispatch(EasyAdminEvents::PRE_LIST);
            ...

    }
```

The menu display is not managed by the event subscriber. We have to add an is_granted statement before rendering the menu. See below:

```
# app/Resources/views/easy_admin/menu.html.twig

{% macro render_menu_item(item, translation_domain) %}
    {% if item.type == 'divider' %}
        {{ item.label|trans(domain = translation_domain) }}
    {% else %}
        {% set menu_params = { menuIndex: item.menu_index, submenuIndex: item.su\
bmenu_index } %}
        {% set path =
        item.type == 'link' ? item.url :
        item.type == 'route' ? path(item.route, item.params) :
        item.type == 'entity' ? path('easyadmin', { entity: item.entity, action:\
 'list' }|merge(menu_params)|merge(item.params)) :
        item.type == 'empty' ? '#' : ''
        %}

        {# if the URL generated for the route belongs to the backend, regenerate
           the URL to include the menu_params to display the selected menu item
           (this is checked comparing the beginning of the route URL with the ba\
ckend homepage URL)
        #}
        {% if item.type == 'route' and (path starts with path('easyadmin')) %}
            {% set path = path(item.route, menu_params|merge(item.params)) %}
        {% endif %}

        <a href="{{ path }}" {% if item.target|default(false) %}target="{{ item.\
target }}"{% endif %}>
            {% if item.icon is not empty %}<i class="fa {{ item.icon }}"></i>{% \
endif %}
            <span>{{ item.label|trans(domain = translation_domain) }}</span>
            {% if item.children|default([]) is not empty %}<i class="fa fa-angle\
-left pull-right"></i>{% endif %}
        </a>
    {% endif %}
{% endmacro %}

{% import _self as helper %}

{% block main_menu_before %}{% endblock %}

<ul class="sidebar-menu">
    {% block main_menu %}
        {% for item in easyadmin_config('design.menu') %}
            {% if is_granted(item.role) %}
                <li class="{{ item.type == 'divider' ? 'header' }} {{ item.child\
ren is not empty ? 'treeview' }} {{ app.request.query.get('menuIndex')|default(-\
1) == loop.index0 ? 'active' }} {{ app.request.query.get('submenuIndex')|default\
(-1) != -1 ? 'submenu-active' }}">

                    {{ helper.render_menu_item(item, 'app') }}

                    {% if item.children|default([]) is not empty %}
                        <ul class="treeview-menu">
                            {% for subitem in item.children %}
                                <li class="{{ subitem.type == 'divider' ? 'heade\
r' }} {{ app.request.query.get('menuIndex')|default(-1) == loop.parent.loop.inde\
x0 and app.request.query.get('submenuIndex')|default(-1) == loop.index0 ? 'activ\
e' }}">
                                    {{ helper.render_menu_item(subitem, _entity_\
config.translation_domain|default('messages')) }}
                                </li>
                            {% endfor %}
                        </ul>
                    {% endif %}
                </li>
            {% endif %}
        {% endfor %}
    {% endblock main_menu %}
</ul>

{% block main_menu_after %}{% endblock %}
```

Try logging in now as test1 and you will see that the menu and entities should be access controlled.

Adding Roles to EasyAdmin Actions

We have seen that easyadmin actions is controlled by the yml files, ie something like:

```
# app/config/easyadmin/userlog.yml
...
show:
    actions: ['list', '-edit', '-delete']
list:
    actions: ['show', '-edit', '-delete']
...
```

What if we want the actions to be "role" aware? If you look at the easyadmin twig files, you will see that it calls a Twig function "getActionsForItem" to get the actions prior to render. This gives us a chance to change the function logic by extending the class.

```
# src/AppBundle/Twig/Extension/EasyAdminTwigExtension.php

namespace AppBundle\Twig\Extension;

use JavierEguiluz\Bundle\EasyAdminBundle\Configuration\ConfigManager;
use Symfony\Component\PropertyAccess\PropertyAccessor;
use Symfony\Component\Security\Core\Authorization\AuthorizationChecker;

/**
 * Class EasyAdminTwigExtension
 * @package AppBundle\Twig\Extension
 */
class EasyAdminTwigExtension extends \JavierEguiluz\Bundle\EasyAdminBundle\Twig\\
EasyAdminTwigExtension
{
    private $checker;

    public function __construct(ConfigManager $configManager, PropertyAccessor $\
propertyAccessor, $debug = false, AuthorizationChecker $checker)
    {
        parent::__construct($configManager, $propertyAccessor, $debug);
        $this->checker = $checker;
    }

    /**
     * Overrides parent function
     *
     * @param string $view
     * @param string $entityName
     *
     * @return array
     */
    public function getActionsForItem($view, $entityName)
    {
        $entityConfig = $this->getEntityConfiguration($entityName);
        $disabledActions = $entityConfig['disabled_actions'];
        $viewActions = $entityConfig[$view]['actions'];

        $actionsExcludedForItems = array(
            'list' => array('new', 'search'),
            'edit' => array(),
            'new' => array(),
            'show' => array(),
        );
        $excludedActions = $actionsExcludedForItems[$view];

        // hid these buttons if easyadmin says so
        $actions = ['edit', 'form', 'delete', 'list', 'show'];
        foreach ($actions as $action) {
            if (isset($entityConfig[$action]['role']) && !$this->checker->isGran\
ted($entityConfig[$action]['role'])) {
                array_push($excludedActions, $action);
            }
        }

        return array_filter($viewActions, function ($action) use ($excludedActio\
ns, $disabledActions) {
            return !in_array($action['name'], $excludedActions) && !in_array($ac\
tion['name'], $disabledActions);
        });
    }
}
```

And we have to remember to call our new twig class in services.yml

```
# src/AppBundle/Resources/config/services.yml
  ...
  app.twig.extension:
    class: AppBundle\Twig\Extension\EasyAdminTwigExtension
    arguments:
        - "@easyadmin.config.manager"
        - "@property_accessor"
        - "%kernel.debug%"
        - "@security.authorization_checker"
    tags:
      - { name: twig.extension }
```

One thing to remember though is that we have to load our AppBundle after EasyAdminbundle so that our app.twig.extension can override the easyadmin.twig.extension service of EasyAdminBundle

```
# app/AppKernel.php
...
        $bundles = [
            new Symfony\Bundle\FrameworkBundle\FrameworkBundle(),
            new Symfony\Bundle\SecurityBundle\SecurityBundle(),
            new Symfony\Bundle\TwigBundle\TwigBundle(),
            new Symfony\Bundle\MonologBundle\MonologBundle(),
            new Symfony\Bundle\SwiftmailerBundle\SwiftmailerBundle(),
            new Doctrine\Bundle\DoctrineBundle\DoctrineBundle(),
            new Sensio\Bundle\FrameworkExtraBundle\SensioFrameworkExtraBundle(),
            new Vich\UploaderBundle\VichUploaderBundle(),
            // init my fosuser
            new FOS\UserBundle\FOSUserBundle(),
            new Doctrine\Bundle\MigrationsBundle\DoctrineMigrationsBundle(),
            new JavierEguiluz\Bundle\EasyAdminBundle\EasyAdminBundle(),
            new Bpeh\NestablePageBundle\BpehNestablePageBundle(),
            new Ivory\CKEditorBundle\IvoryCKEditorBundle(),
            new AppBundle\AppBundle(),
            new AppBundle\User(),
            new AppBundle\Page(),
...
```

I have disabled the “edit” action for all users, so the edit button will not show even if the user is himself. For the sake of simplicity, let us change the layout header link to use edit action instead.

```
# app/Resources/EasyAdminBundle/views/default/layout.html.twig
...
<a href="{{ path('easyadmin') }}/?entity=User&action=edit&id={{ app.user.id }}">\
{{ app.user.username|default('user.unnamed'|trans(domain = 'EasyAdminBundle')) }\
}</a>
...
```

Cleaning up

We are close to the end of the chapter. Let us clean up all our code using php-cs-fixer (Still remember this?)

```
-> vendor/friendsofphp/php-cs-fixer/php-cs-fixer fix src/
-> vendor/friendsofphp/php-cs-fixer/php-cs-fixer fix src/
# finally optimising composer
-> ./scripts/optimize_composer
```

Update BDD (Optional)

We have updated some business rules. Users can now see and do what they are allowed in the admin area based on their role in the easyadmin yaml config files. Its time to ensure we update our tests to reflect these changes.

Summary

In this chapter, we have cleaned up config.yml and provided a custom solution (Using compiler pass and event listeners) to make EasyAdmin support user permissions in the admin area. It was a huge effort but not yet a full solution. However, it should make life easy for people who wants to configure admin permissions easily.

Exercises

- Think of another way to make EasyAdmin support user permissions.
- Write your test and make sure everything passes (Optional)
- Can you implement autowiring[155] in services.yml? What are the pros and cons of using autowiring?

[155]http://symfony.com/doc/current/components/dependency_injection/autowiring.html

References

- Prepend Config[156]
- Dependency Injection Component[157]
- Service Container[158]
- Tagging Symfony Services[159]
- Autowiring[160]

[156]http://symfony.com/doc/current/bundles/prepend_extension.html
[157]https://symfony.com/doc/current/components/dependency_injection.html
[158]http://symfony.com/doc/current/service_container.html
[159]http://thorpesystems.com/blog/tagging-symfony-services/
[160]http://symfony.com/doc/current/components/dependency_injection/autowiring.html

Final Chapter: Conclusion

Congratulations for perservering for so long… It's been a long journey. In the previous chapters, we have created a simple CMS using a modular[161] approach. The CMS is *really simple* but is secure, supports user logging and internalisation. While going through the exercises, we have explored possibilities to build different parts of the CMS bit by bit.

Now, you have all the basic knowledge and foundation to create more complex applications with Symfony.

So, what's next from here? Ready for more adventures?

Here are some suggestions:

- Start building something with Symfony fullstack or its components.
- I am sure you will find bugs and typos along the way. Create pull requests for SongBird in git.
- Improve on the NestablePageBundle to reduce the amount of work required to integrate with EasyAdminBundle.
- Create API for 3rd party services to connect to.
- Add Ecommerce capability to the CMS by adding a payment module.
- Improve on look and feel. The frontend looks too plain.
- Try Implement ACL[162] for users.
- Investigate the best practices to deploy your application to a reliable server and configure the production settings.
- What if your application becomes popular and you are getting a lot of traffic? What are the options to optimise your application?
- What about configuring cron jobs to clean up user logging table?
- How about packing your application up into an installable bundle? That way, you can distribute your application easily.

Good luck on your next Symfony Journey!

[161]https://en.wikipedia.org/wiki/Modular_design
[162]http://symfony.com/doc/current/security/acl.html