Creating your First Symfony App and Adding Authentication

# Getting Started

We will be building a simple tech company listing app using the latest version of Symfony. Symfony has come a long way since its first official release. Here we will work with Symfony 5. This latest version comes with a lot of benefits that we will see later in the tutorial. Our app will simply list Top tech companies in the world. Once we add authentication to the app, all logged-in users will have the privilege of knowing these top tech companies and their market value.

Symfony utilizes [Composer](http://getcomposer.org/) to manage its dependencies. So, before using Symfony, make sure you have Composer installed on your machine. We can install Symfony Framework by issuing the composer create-project command in our terminal or using the symfony installer.

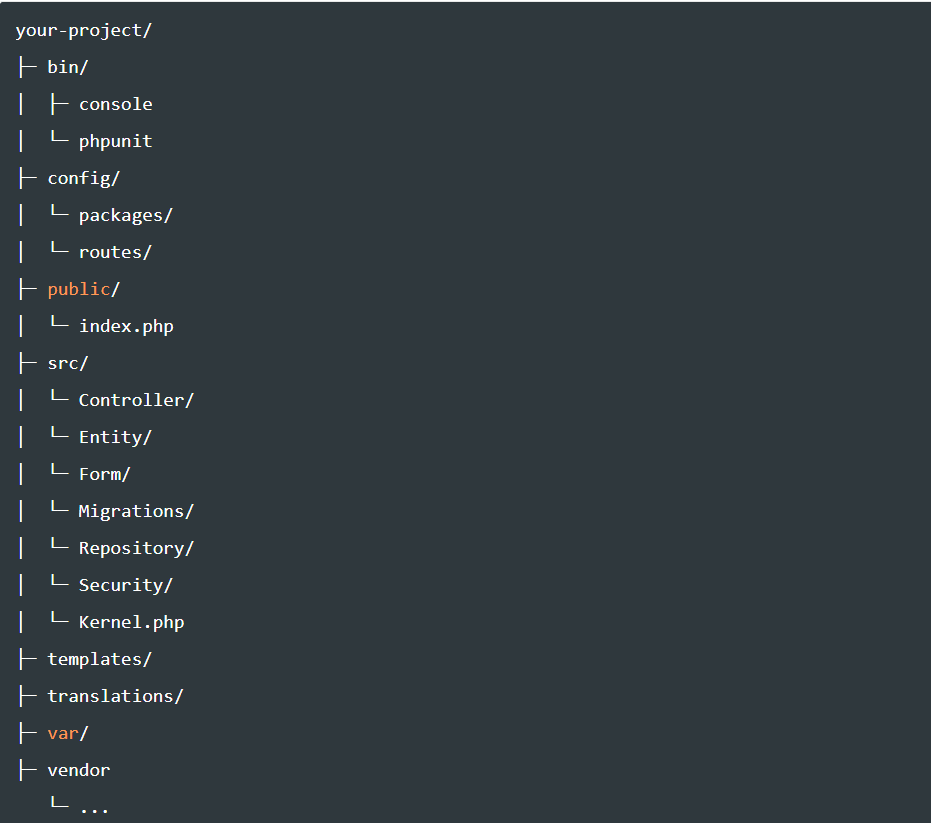
To create our application for this tutorial, run the following command to create a new web application named top-tech-companies:

symfony new --full top-tech-companies

The preceding command will create a new folder named top-tech-companies in the root folder where you ran the command from. It will also install all of the required dependencies

# Explore Directory Structure

**Symfony Framework** automatically ships with a default directory structure like the one below:



# Running the Application

Move into the newly created project and install a web server:

// Change directory

cd top-tech-companies

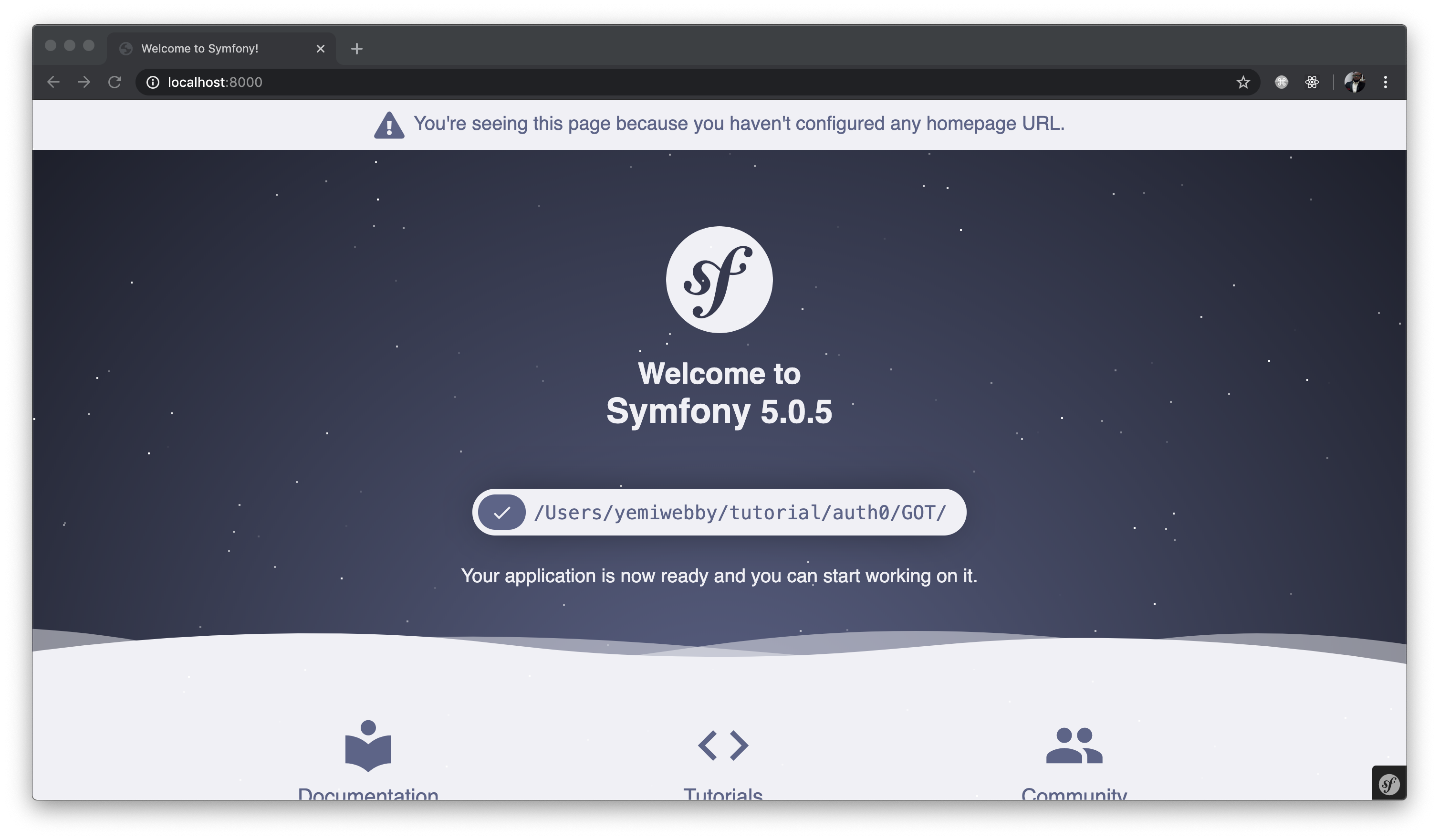
Then run the application with:

php bin/console server:run

If you installed and configured the Symfony installer, you can also use the following command to run the application:

symfony serve

You can view it on [http://localhost:8000](http://localhost:8000/).

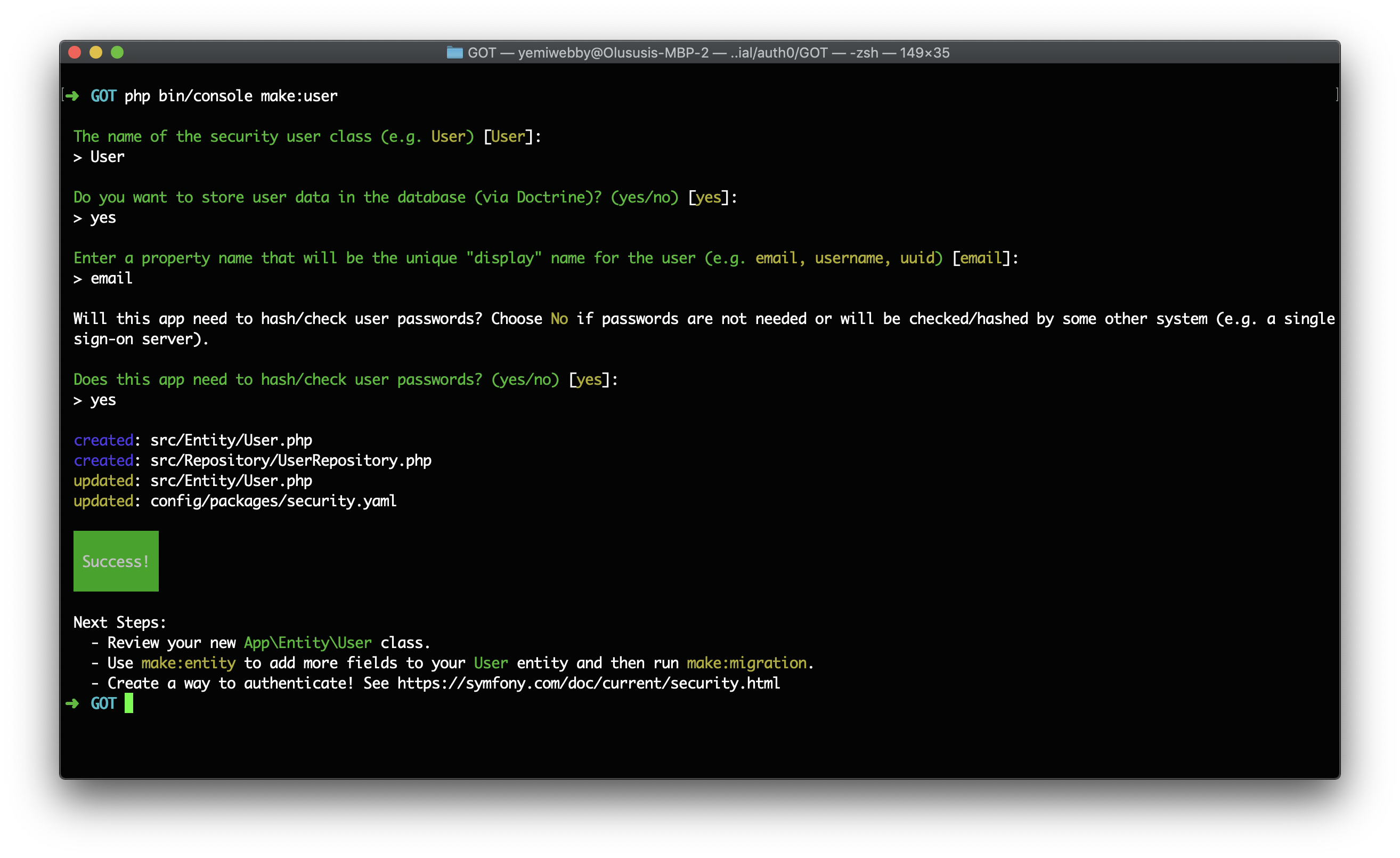


# Creating a User Class

Before we can register or authenticate a user within our application, we need to create a User class or an entity. Let’s use the [symfony/maker bundle](https://symfony.com/blog/introducing-the-symfony-maker-bundle" \t "_blank) to generate it. Stop the development server from running using CTRL + C, and run the following command afterward:

php bin/console make:user

The command above will ask you several questions so that the appropriate files can be automatically generated for you. Follow the prompt and respond accordingly, as shown here:



This will create two new files named src/Entity/User.php and src/Repository/UserRepository.php and also update the config/packages/security.yaml file. More about that later in the tutorial.

Before we wrap up this section, we need to add one more field to the User class. Open the src/Entity/User.php file and update its content, as shown below:

// src/Entity/User.php

<?php

namespace App\Entity;

use Doctrine\ORM\Mapping as ORM;

use Symfony\Component\Security\Core\User\UserInterface;

/\*\*

\* @ORM\Entity(repositoryClass="App\Repository\UserRepository")

\*/

class User implements UserInterface

{

/\*\*

\* @ORM\Id()

\* @ORM\GeneratedValue()

\* @ORM\Column(type="integer")

\*/

private $id;

/\*\*

\* @ORM\Column(type="string", length=180, unique=true)

\*/

private $email;

/\*\*

\* @ORM\Column(type="json")

\*/

private $roles = [];

/\*\*

\* @var string The hashed password

\* @ORM\Column(type="string")

\*/

private $password;

/\*\*

\* @ORM\Column(type="string", length=255)

\*/

private $name;

public function getId(): ?int

{

return $this->id;

}

public function getEmail(): ?string

{

return $this->email;

}

public function setEmail(string $email): self

{

$this->email = $email;

return $this;

}

/\*\*

\* A visual identifier that represents this user.

\*

\* @see UserInterface

\*/

public function getUsername(): string

{

return (string) $this->email;

}

/\*\*

\* @see UserInterface

\*/

public function getRoles(): array

{

$roles = $this->roles;

// guarantee every user at least has ROLE\_USER

$roles[] = 'ROLE\_USER';

return array\_unique($roles);

}

public function setRoles(array $roles): self

{

$this->roles = $roles;

return $this;

}

/\*\*

\* @see UserInterface

\*/

public function getPassword(): string

{

return (string) $this->password;

}

public function setPassword(string $password): self

{

$this->password = $password;

return $this;

}

/\*\*

\* @see UserInterface

\*/

public function getSalt()

{

// not needed when using the "bcrypt" algorithm in security.yaml

}

/\*\*

\* @see UserInterface

\*/

public function eraseCredentials()

{

// If you store any temporary, sensitive data on the user, clear it here

// $this->plainPassword = null;

}

public function getName(): ?string

{

return $this->name;

}

public function setName(string $name): self

{

$this->name = $name;

return $this;

}

}

In addition to the properties automatically created by the Symfony MakerBundle, we included a name property and also created both getter and setter method for it. All the properties defined here will represent each field for the user table within the database.

# Setting up the Controllers

Now that we have a Symfony project installed, we need to generate a new controller to handle content rendering and any HTTP requests sent to our application. We will start by creating a controller that will handle render the list of tech companies, as stated earlier. Use the following command to generate the ListController:

php bin/console make:controller ListController

This will create two new files for you: a controller located in src/Controller/ListController.php and a view page in templates/list/index.html.twig. Open the ListController.php file and replace its content with:

// ./src/Controller/ListController

<?php

namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;

use Symfony\Component\HttpFoundation\Request;

use Symfony\Component\Routing\Annotation\Route;

class ListController extends AbstractController

{

/\*\*

\* @Route("/list", name="list")

\*/

public function index(Request $request)

{

$companies = [

'Apple' => '$1.16 trillion USD',

'Samsung' => '$298.68 billion USD',

'Microsoft' => '$1.10 trillion USD',

'Alphabet' => '$878.48 billion USD',

'Intel Corporation' => '$245.82 billion USD',

'IBM' => '$120.03 billion USD',

'Facebook' => '$552.39 billion USD',

'Hon Hai Precision' => '$38.72 billion USD',

'Tencent' => '$3.02 trillion USD',

'Oracle' => '$180.54 billion USD',

];

return $this->render('list/index.html.twig', [

'companies' => $companies,

]);

}

}

To keep things simple, we created and hardcoded the list of companies, as obtained from this [article](https://www.investopedia.com/articles/markets/030816/worlds-top-10-technology-companies-aapl-googl.asp) and passed it to a view named index.html.twig within the list directory. If your application is in production, you should retrieve the items within this array from the data in your database.

Next, generate a new controller to handle user registration with:

php bin/console make:controller RegistrationController

This will create two new files for you: a controller located in src/Controller/RegistrationController.php and a view page in templates/registration/index.html.twig. Open the RegistrationController.php file and replace its content with:

// ./src/Controller/RegistrationController

<?php

namespace App\Controller;

use App\Entity\User;

use App\Form\UserType;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;

use Symfony\Component\HttpFoundation\Request;

use Symfony\Component\Routing\Annotation\Route;

use Symfony\Component\Security\Core\Encoder\UserPasswordEncoderInterface;

class RegistrationController extends AbstractController

{

private $passwordEncoder;

public function \_\_construct(UserPasswordEncoderInterface $passwordEncoder)

{

$this->passwordEncoder = $passwordEncoder;

}

/\*\*

\* @Route("/registration", name="registration")

\*/

public function index(Request $request)

{

$user = new User();

$form = $this->createForm(UserType::class, $user);

$form->handleRequest($request);

if ($form->isSubmitted() && $form->isValid()) {

// Encode the new users password

$user->setPassword($this->passwordEncoder->encodePassword($user, $user->getPassword()));

// Set their role

$user->setRoles(['ROLE\_USER']);

// Save

$em = $this->getDoctrine()->getManager();

$em->persist($user);

$em->flush();

return $this->redirectToRoute('app\_login');

}

return $this->render('registration/index.html.twig', [

'form' => $form->createView(),

]);

}

}

Here, we mapped this controller to the registration route. Therefore, all HTTP requests sent to the /registration endpoint to register users within the application will be processed by the index() method defined above. This method will process the registration form and persist user data to the database. If the form has not been submitted yet, a user registration form will be rendered instead.

Lastly, generate a new controller that will handle the login process for a user:

php bin/console make:controller SecurityController

After running the command above, a new controller named SecurityController.php will be created within the src/controller directory. We will update this file in a bit.

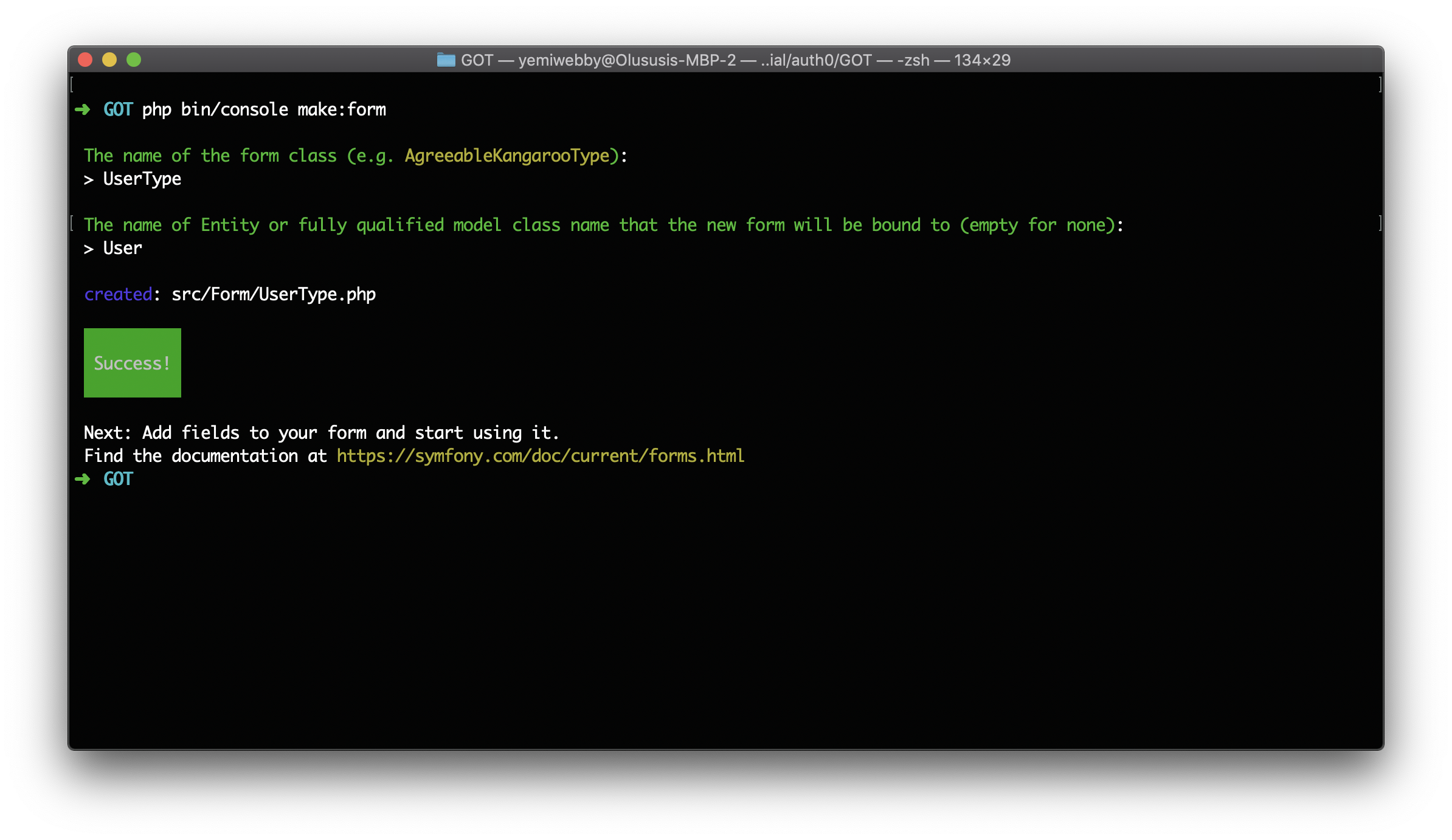
# Creating User Type

Earlier, we referenced a form within the RegistrationController.php. We will create the form in this section. To begin, we will use the Maker bundle to create a form to register users. This removes the stress involved in creating and rendering form fields, handling validation, and so on, as it is done for you.

Run the following command and follow the prompts to generate the registration form:

php bin/console make:form

Start by entering UserType as the name of the form class. Next, enter the name of the User class created earlier:



Now, open the src/Form/UserType.php file and use the following content for it:

// src/Form/UserType.php

<?php

namespace App\Form;

use App\Entity\User;

use Symfony\Component\Form\AbstractType;

use Symfony\Component\Form\Extension\Core\Type\EmailType;

use Symfony\Component\Form\Extension\Core\Type\PasswordType;

use Symfony\Component\Form\Extension\Core\Type\RepeatedType;

use Symfony\Component\Form\Extension\Core\Type\TextType;

use Symfony\Component\Form\FormBuilderInterface;

use Symfony\Component\OptionsResolver\OptionsResolver;

class UserType extends AbstractType

{

public function buildForm(FormBuilderInterface $builder, array $options)

{

$builder

->add('email', EmailType::class)

->add('name', TextType::class)

->add('password', RepeatedType::class, [

'type' => PasswordType::class,

'first\_options' => ['label' => 'Password'],

'second\_options' => ['label' => 'Confirm Password']

])

;

}

public function configureOptions(OptionsResolver $resolver)

{

$resolver->setDefaults([

'data\_class' => User::class,

]);

}

}

First, we modified the content generated for this file by including the Type for each of the form fields and also included a password confirm field. These form fields will be displayed on the registration form.

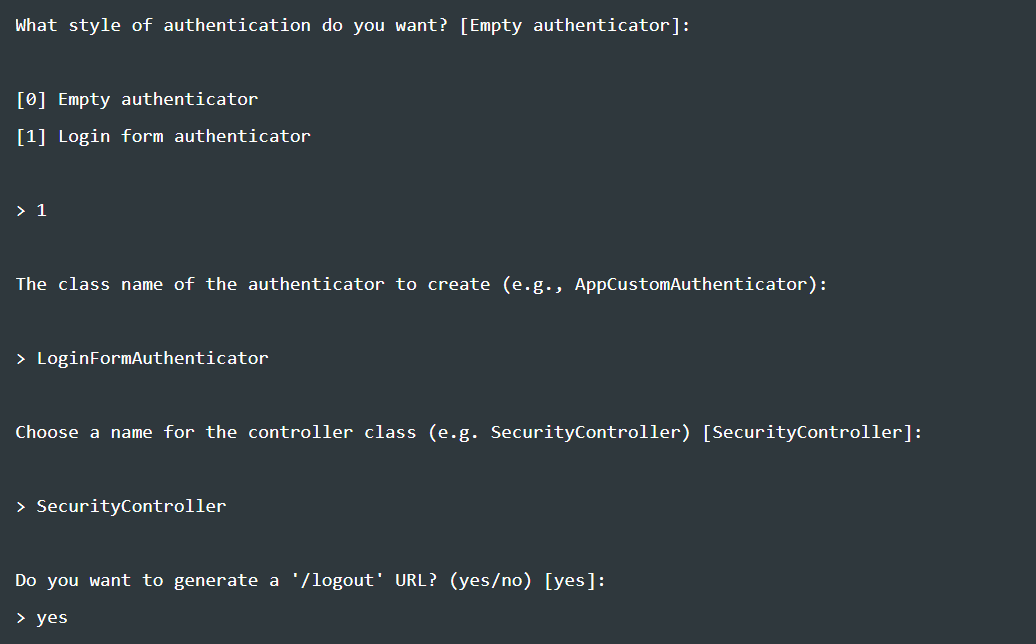
# Generating the Login Form

Creating a powerful login form for Symfony website is quite simple. The [Makerbundle](https://symfony.com/doc/current/bundles/SymfonyMakerBundle/index.html" \t "_blank) can be used to easily bootstrap a new Login form without hassle. Depending on your setup, you may be asked different questions, and your generated code may be slightly different. To create the Login form, run the following command:

php bin/console make:auth

As a response from the preceding command, you will be prompted to provide answers to a couple of questions.

Respond as follows:



Once the process is completed, two new files will automatically be created for you in src/Security/LoginFormAuthenticator.php and templates/security/login.html.twig. It will also update both config/packages/security.yaml and src/Controller/SecurityController.php.

Open the src/Controller/SecurityController.php and update it as shown here:

// src/Controller/SecurityController.php

<?php

namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;

use Symfony\Component\HttpFoundation\Response;

use Symfony\Component\Routing\Annotation\Route;

use Symfony\Component\Security\Http\Authentication\AuthenticationUtils;

class SecurityController extends AbstractController

{

/\*\*

\* @Route("/", name="app\_login")

\*/

public function login(AuthenticationUtils $authenticationUtils): Response

{

// get the login error if there is one

$error = $authenticationUtils->getLastAuthenticationError();

// last username entered by the user

$lastUsername = $authenticationUtils->getLastUsername();

return $this->render('security/login.html.twig', ['last\_username' => $lastUsername, 'error' => $error]);

}

/\*\*

\* @Route("/logout", name="app\_logout")

\*/

public function logout()

{

throw new \Exception('This method can be blank - it will be intercepted by the logout key on your firewall');

}

}

Here, we edited the @Route() annotation on the login() method to render the login form on the homepage of our application.

## Configuring the Database

Here, let us configure our database connection. The default driver Symfony ships with is MySQL. Open the .env file within the root directory of the application and find the DATABASE\_URL environment variable.

DATABASE\_URL=mysql://root:@127.0.0.1:3306/toptechcompanies?serverVersion=mariadb-10.4.11

Update this line with your own credentials and the name you want to use for the database, for example, techcompanies. If you don't have MySQL installed and set up on your system, [follow this guide to get started](https://dev.mysql.com/doc/mysql-getting-started/en/).

* db\_user: Replace with your database username
* db\_password: Replace with your database password
* db\_name: Replace with your database name. You don't have to create the database yet, as we'll do that in the next step.

**Note**: This is a good time to double-check that .env is listed in your .gitignore file. You should NEVER commit sensitive data to your repository.

Next, run the following command to create a database with the value of your database name:

php bin/console doctrine:database:create

At the moment, the database still has no tables. Run the following command that will instruct [Doctrine](https://symfony.com/doc/current/doctrine.html) to create the tables based on the User entity that we have created earlier:

php bin/console doctrine:schema:update --force

# Setting up Authentication

**Symfony** ships with an awesome security component called [**Guard**](https://symfony.com/doc/current/security/guard_authentication.html) that simplifies the authentication process. Let's take advantage of it in our app. The first step is to configure the Symfony security settings.

Open up config/packages/security.yaml file and configure it like so:

security:

encoders:

App\Entity\User:

algorithm: auto

# https://symfony.com/doc/current/security.html#where-do-users-come-from-user-providers

providers:

# used to reload user from session & other features (e.g. switch\_user)

app\_user\_provider:

entity:

class: App\Entity\User

property: email

firewalls:

dev:

pattern: ^/(\_(profiler|wdt)|css|images|js)/

security: **false**

main:

anonymous: lazy

provider: app\_user\_provider

guard:

authenticators:

- App\Security\LoginFormAuthenticator

logout:

path: app\_logout

target: /

Most sections within the preceding file have been pre-configured by the MakerBundle. It handles the following as indicated by each section:

* encoders: This is used to configure how passwords created within the application will be hashed. Leaving the value for the algorithm to be auto will auto-selects the best possible hashing algorithm for it.
* providers: This points to the PHP class that will be use to load a user object from the session.
* firewalls: This is used to define how users of our application will be authenticated.

Lastly, to redirect a user back to the homepage after a successful logout process, we edited the logout section by changing the target path to \;

## Setting up Views

**Symfony Framework** ships with a powerful templating engine called [Twig](https://twig.sensiolabs.org/). Twig allows you to write concise, readable templates that are more friendly to web designers and, in several ways, more powerful than PHP templates.

The views needed for authentication in this app are in the templates/security directory. The base layout for our application has also been configured in the templates/base.html.twig file. All of these views use the [Bootstrap CSS framework](https://getbootstrap.com/), but you are free to customize them however you wish.

Open up your templates/list/index.html.twig and configure it like so:

{# templates/list/index.html.twig #}

{% extends 'base.html.twig' %} {% block body %}

<div class="container">

<div class="row">

<div class="col-md-12">

<div class="card bg-light mb-3 mt-3">

<div class="card-body">

<div class="card-header">List of top technology companies</div>

{% if app.user != null %}

<table class="table">

<tr>

<th>Company Name</th>

<th>Market Value</th>

</tr>

{% for key, item in companies %}

<tr>

<td>{{ key }}</td>

<td>{{ item }}</td>

</tr>

{% endfor %}

</table>

{% endif %}

</div>

</div>

{% if app.user == null %}

<a href="{{ path('app\_login') }}" class="btn btn-info">

You need to login to see the list 😜😜 >></a

>

{% endif %}

</div>

</div>

</div>

{% endblock %}

Here, we are looping through the $companies array data passed from the ListController for appropriate rendering in the index.html.twig view.

* app.user == null — Let's you check if a user is authenticated or not. It returns true if a user is logged-in and null if a user is not.

Open templates/security/login.html.twig and templates/registration/index.html.twig templates. Configure them respectively:

{# templates/security/login.html.twig #}

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

{% block title %}Log in!{% endblock %}

{% block body %}

<div class="container">

<div class="row">

<div class="col-md-10 ml-md-auto">

<div class="">

<div class="card bg-light mb-3 mt-5" style="width: 800px;">

<div class="card-body">

<form class="form-horizontal" role="form" method="post">

{% if error %}

<div class="alert alert-danger">

{{ error.messageKey|trans(error.messageData, 'security') }}

</div>

{% endif %} {% if app.user %}

<div class="mb-3">

You are logged in as {{ app.user.username }},

<a href="{{ path('app\_logout') }}">Logout</a>

</div>

{% endif %}

<div class="card-header mb-3">Please sign in</div>

<div class="form-group">

<label for="email" class="col-md-4 control-label"

>E-Mail Address</label

>

<div class="col-md-12">

<input

id="inputEmail"

type="email"

class="form-control"

name="email"

value="{{ last\_username }}"

required

autofocus

/>

</div>

</div>

<div class="form-group">

<label for="password" class="col-md-4 control-label"

>Password</label

>

<div class="col-md-12">

<input

id="inputPassword"

type="password"

class="form-control"

name="password"

required

/>

</div>

</div>

<input

type="hidden"

name="\_csrf\_token"

value="{{ csrf\_token('authenticate') }}"

/>

<div class="form-group">

<div class="col-md-12">

<button type="submit" class="btn btn-primary">

<i class="fa fa-btn fa-sign-in"></i> Login

</button>

</div>

</div>

</form>

</div>

</div>

</div>

</div>

</div>

</div>

{% endblock %}

And for templates/registration/index.html.twig, paste in:

{# templates/registration/index.html.twig #}

{% extends 'base.html.twig' %} {% block body %}

<div class="container">

<div class="row">

<div class="col-md-10 ml-md-auto">

<div class="card bg-light mb-3 mt-5" style="width: 800px">

<div class="card-body">

<div class="card-header mb-3">Registration Form</div>

{{ form\_start(form) }}

<div class="form\_group">

<div class="col-md-12 mb-3">

{{ form\_row(form.name, {'attr': {'class': 'form-control'}}) }}

</div>

</div>

<div class="form\_group">

<div class="col-md-12 mb-3">

{{ form\_row(form.email, {'attr': {'class': 'form-control'}}) }}

</div>

</div>

<div class="form\_group">

<div class="col-md-12 mb-3">

{{ form\_row(form.password.first, {'attr': {'class':

'form-control'}}) }}

</div>

</div>

<div class="form\_group">

<div class="col-md-12 mb-3">

{{ form\_row(form.password.second, {'attr': {'class':

'form-control'}}) }}

</div>

</div>

<div class="form-group">

<div class="col-md-8 col-md-offset-4" style="margin-top:5px;">

<button type="submit" class="btn btn-primary">

<i class="fa fa-btn fa-user"></i> Register

</button>

</div>

</div>

{{ form\_end(form) }}

</div>

</div>

</div>

</div>

</div>

{% endblock %}

We are making use of the **Symfony** built-in [form](https://symfony.com/doc/current/book/forms.html) methods in this template.

# Update the Base Template

Update the base template with:

{# templates/base.html.twig #}

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8" />

<title>{% block title %}Welcome!{% endblock %}</title>

<link

rel="stylesheet"

href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css"

/>

{% block stylesheets %}{% endblock %}

</head>

<body>

<nav

class="navbar navbar-expand-lg navbar-light bg-light"

style="height: 70px;"

>

<a class="navbar-brand" href="#">Symfony</a>

<div class="collapse navbar-collapse" id="navbarSupportedContent"></div>

<ul class="nav navbar-nav navbar-right">

{% if app.user %}

<li><a class="nav-link" href="{{ path('list') }}">View List</a></li>

<li><a class="nav-link" href="{{ path('app\_logout') }}">Logout</a></li>

{% else %}

<li><a class="nav-link" href="{{ path('app\_login') }}">Login</a></li>

<li>

<a class="nav-link" href="{{ path('registration') }}">Register</a>

</li>

{% endif %}

</ul>

</nav>

{% block body %}{% endblock %} {% block javascripts %}{% endblock %}

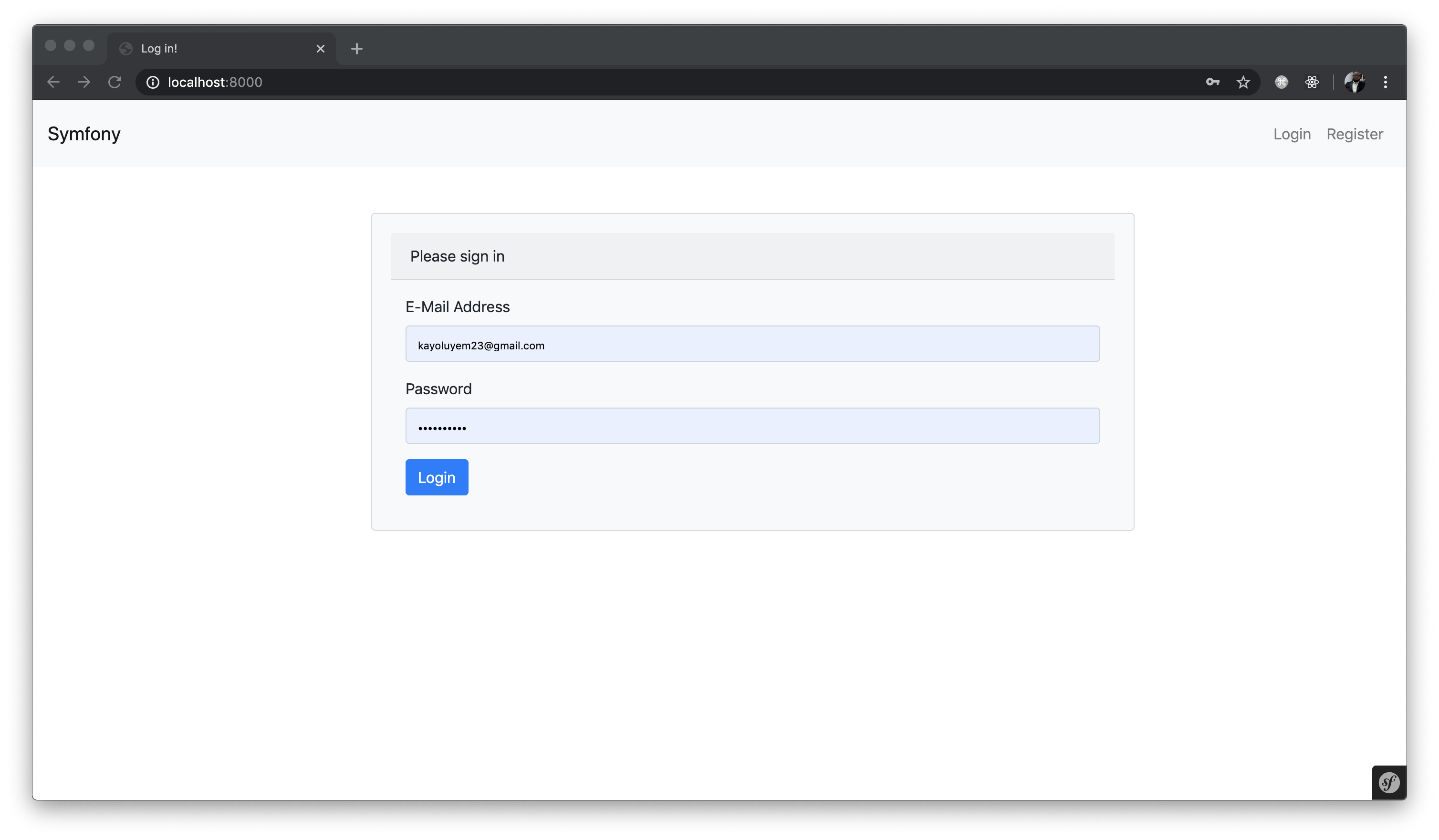
</body>

</html>

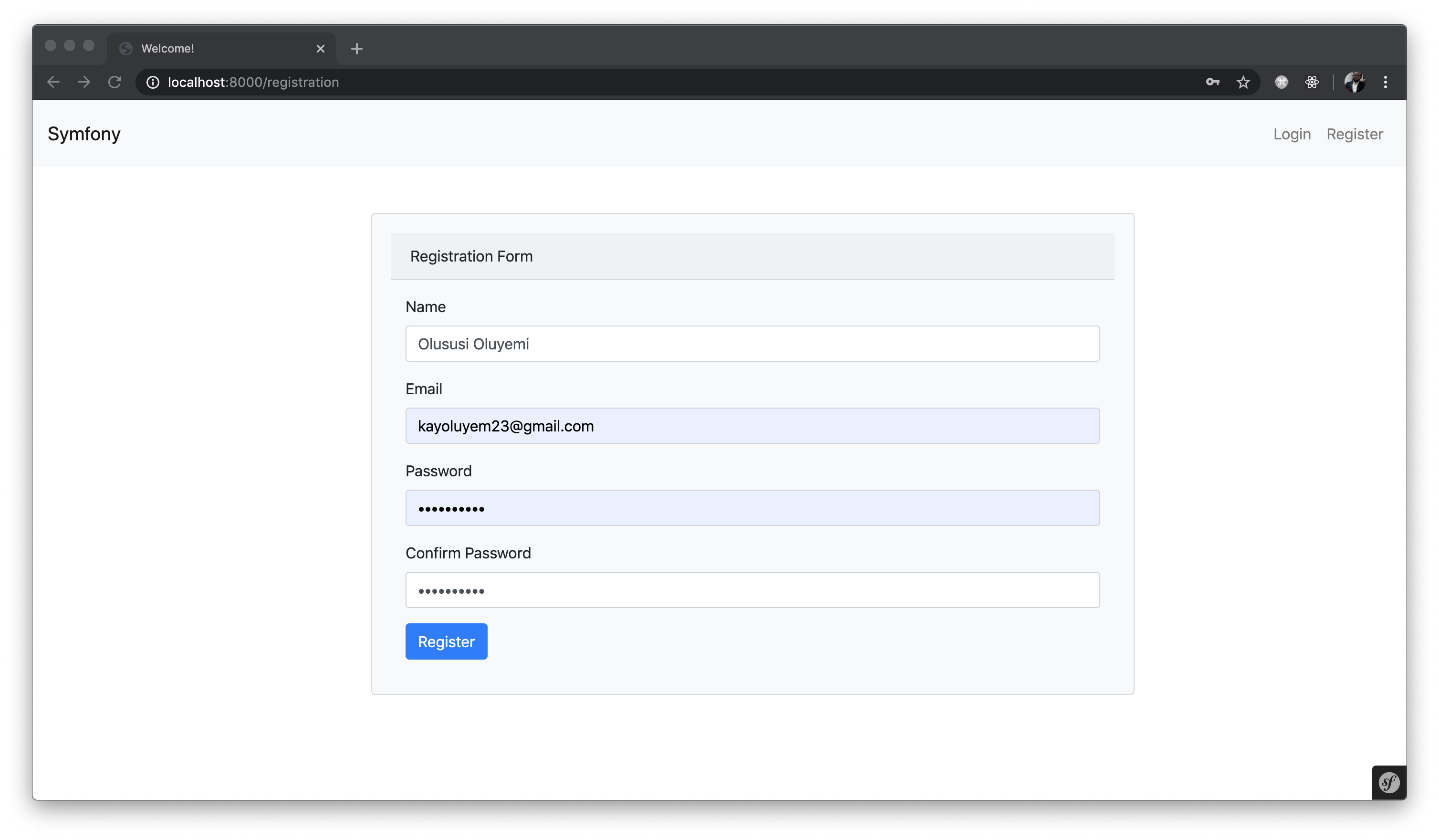
This file contains the main layout for our application. Included here is the CDN file for Boostrap CSS and a navigation bar that contains the links to different pages within this application as indicated by the route.

With all the routes and views fully set up, you can now go ahead and run the application using php bin/console server:run. View it on [http://localhost:8000](http://localhost:8000/). You will see the following:

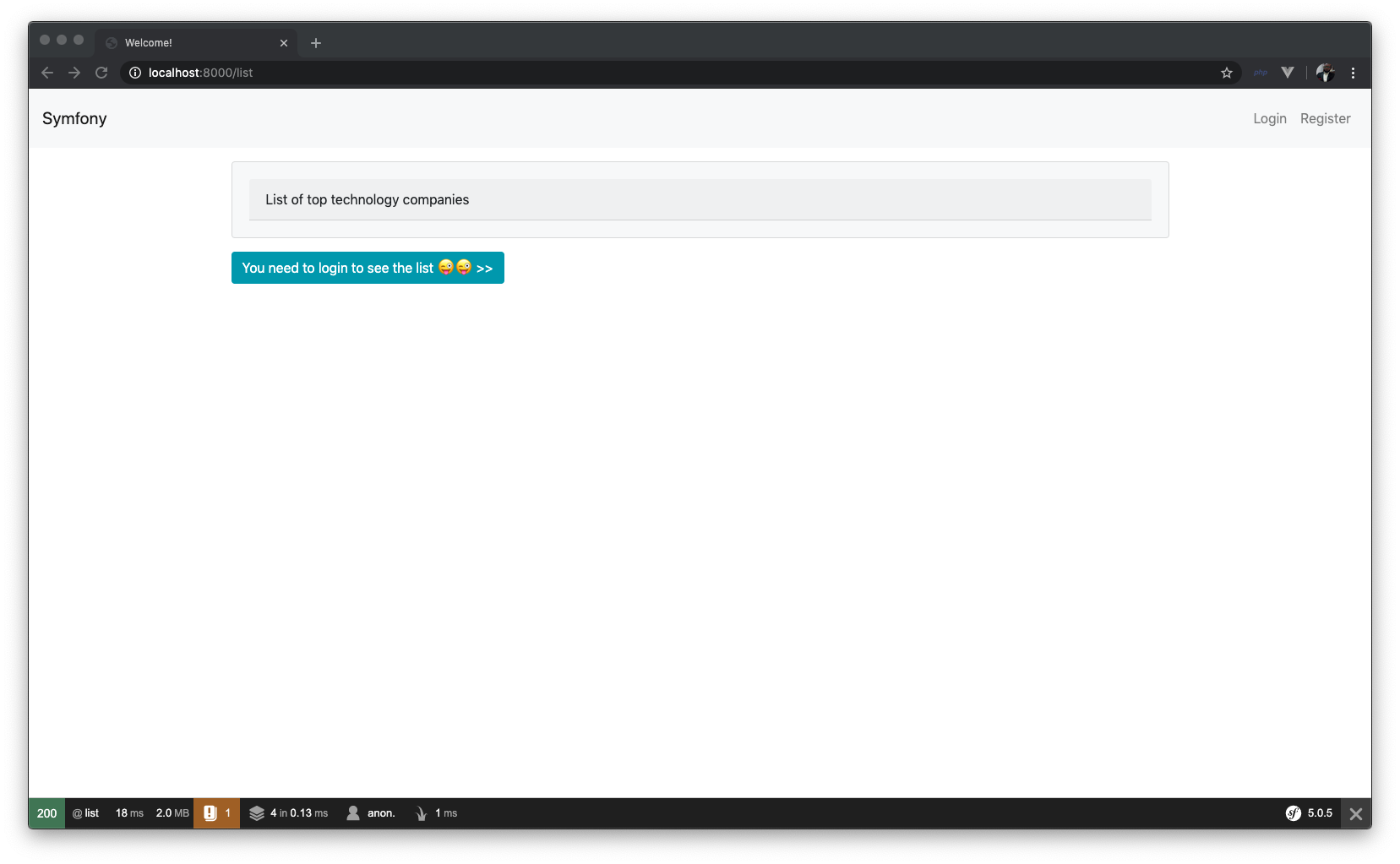
**Login Page**



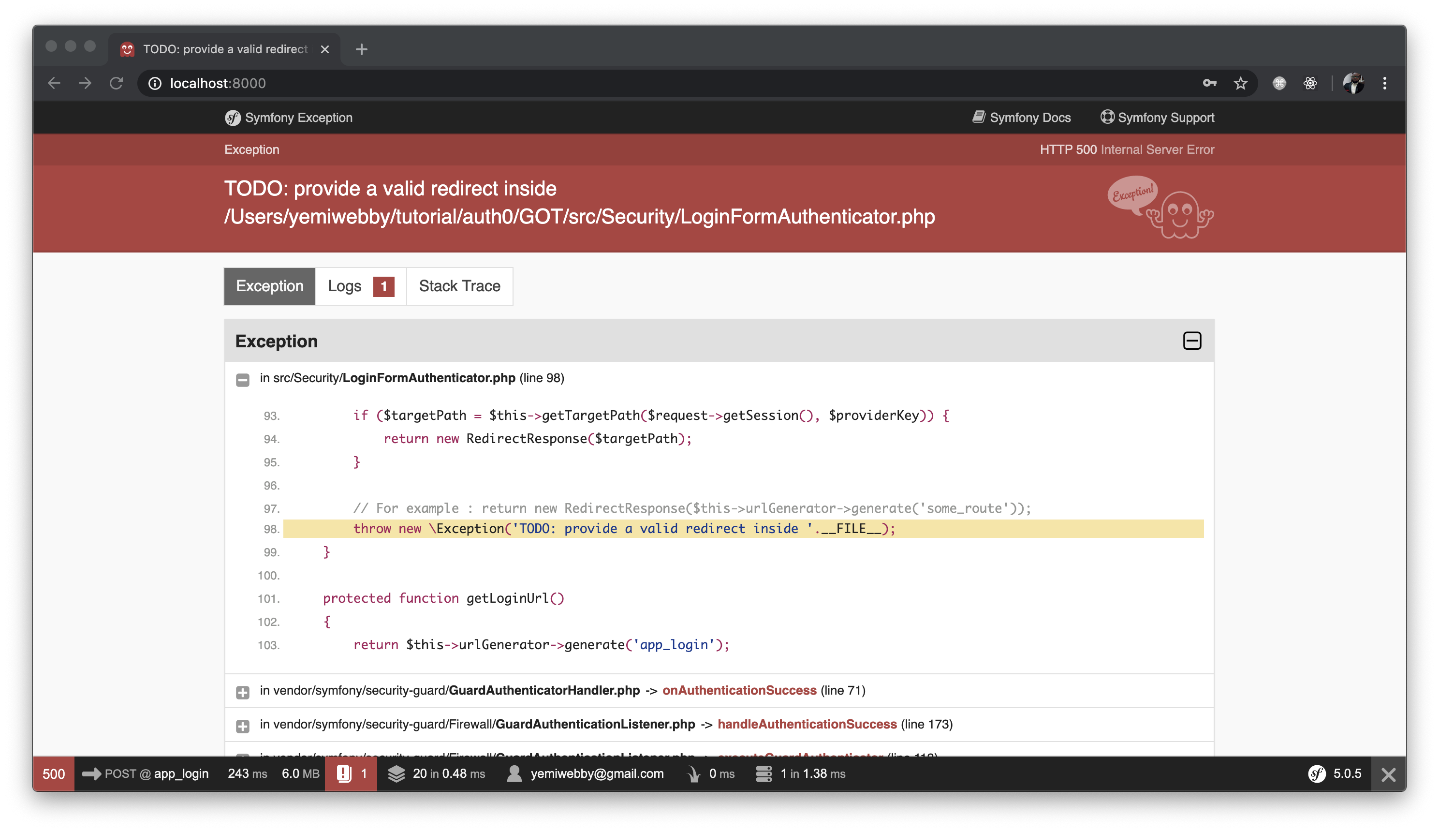
**Register Page**



**Landing Page**



Try registering a user on the application. Then proceed to log in. You will see this:



Oops! Don’t sweat it. This error occurred because a valid redirect route once authenticated has not been provided yet. To fix this, open the LoginFormAuthenticator.php file within src/Security directory and update the onAuthenticationSuccess() method as shown below:

// src/Security/LoginFormAuthenticator

public function onAuthenticationSuccess(Request $request, TokenInterface $token, $providerKey)

{

if ($targetPath = $this->getTargetPath($request->getSession(), $providerKey)) {

return new RedirectResponse($targetPath);

}

return new RedirectResponse($this->urlGenerator->generate('list'));

}

What we have done here is to redirect the users to the list route immediately after being authenticated. You can now login and see the application functioning properly as described earlier:

