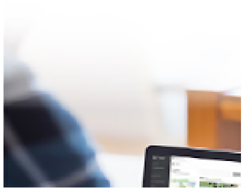
Binaryboxtuts

Symfony 6
JSON Web
Token(JWT)
Authentication
binaryboxtuts.com

cisco Meraki DOĞUS

Bulut Tabanl
Yönetim Siste

TEKLİF



January 26, 2023 ▪ PHP / Symfony

Symfony 6 JSON Web Token(JWT) Authentication

Posted by Binarytuts

Bulut
Tabanlı
Çözüm

Doğuş Elektrik
Elektronik

Hi! Today we will learn how to create an authentication on our Symfony 6 API. But before that let's have a discussion about API and what is JSON Web Token(JWT).

Recent Posts

- How To Develop Login And Registration in ReactJS App using API
- How To Develop

API stands for Application Program Interface, API is an interface that allows applications to exchange data. To make it more clear, APIs are a set of functions that can be used by programmers to build software and applications.

JWT stands for JSON Web Token, it is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. JWT is commonly used for Authorization, Information Exchange and etc.

Now that we have a glimpse of the idea on the topic, We will now proceed on building the app.

Prerequisite:

- Composer
- Symfony CLI
- MySQL
- PHP >= 8.0.2

Step 1: Install Symfony 6

First, select a folder that you want Symfony to be installed then execute this command on Terminal or CMD to install:

Install via composer:

```
composer create-project symfony/skeleton sy
```

Install via Symfony CLI:

- How To Develop Single Page Application Using ReactJS
- Symfony 6 JSON Web Token(JWT) Authentication
- Generating PDF from HTML In Laravel 8
- CodeIgniter 4 JSON Web Token(JWT) Authentication
- How to make Symfony 6 REST API
- Symfony 5 CRUD App Easy Tutorial

Step 2: Install Packages

After installing Symfony, we must install the necessary packages to our app. During the installation of the packages, it will ask you to execute the recipes, type **y** to confirm.

```
composer require jms/serializer-bundle
composer require friendsofsymfony/rest-bun
composer require symfony/maker-bundle
composer require symfony/orm-pack
composer require lexik/jwt-authentication-l
```

Step 3: Set Database Configuration

After installing, open the **.env** file and set the database configuration. We will be using MySQL in this tutorial. Uncomment the DATABASE_URL variable for MySQL and updates its configs. Make sure you commented out the other DATABASE_URL variables.

.env

```
# In all environments, the following file:
# the latter taking precedence over the fo
#
#  * .env                contains default
#  * .env.local          uncommitted file
#  * .env.$APP_ENV       committed enviro
#  * .env.$APP_ENV.local uncommitted envi
#
# Real environment variables win over .env
#
# DO NOT DEFINE PRODUCTION SECRETS IN THIS
#
# Run "composer dump-env prod" to compile
# https://symfony.com/doc/current/best_pra

###> symfony/framework-bundle ###
APP_ENV=dev
APP_SECRET=e07103178612213 71d185cc932acd1!
###< symfony/framework-bundle ###

###> doctrine/doctrine-bundle ###
# Format described at https://www.doctrine
# IMPORTANT: You MUST configure your serve
#
# DATABASE_URL="sqlite:///%kernel.project_
# DATABASE_URL="mysql://db_user:db_passwor
DATABASE_URL="postgresql://db_user:db_pas
###< doctrine/doctrine-bundle ###
```

Tags

AJAX
API
Authentication
Bootstrap
CodeIgniter 3
CodeIgniter 4
Composer
CRUD
CSV
DataTable
Email
Javascript
JQuery
JWT
Laravel
Laravel 7
Laravel 8
Laravel 9
Laravel Breeze
Laravel Passport
Login
Lumen 9
Mailtrap
Multiple File Upload
Page Numbering

PHP QR Code
React ReactJS
Registration REST API
Single Page Application
Symfony 6

After configuring the database, execute this command to create the database:

```
php bin/console doctrine:database:create
```

Step 4: Configure FOSRest Bundle

Open the file ***config/packages/fos_rest.yaml*** and add these line:

config/packages/fos_rest.yaml

```
fos_rest:
    format_listener:
        rules:
            - { path: ^/api, prefer_extens:
```

Step 5: Create User Class

We will then create a user class, by using the ***make:user*** command – this command will create a User class for security and it will automatically update the ***security.yaml***.

Follow these steps:

```
php bin/console make:user

 The name of the security user class (e.g
 >

 Do you want to store user data in the da
 >

 Enter a property name that will be the u
 >

 Will this app need to hash/check user pa
```

```
created: src/Entity/User.php
created: src/Repository/UserRepository.pl
updated: src/Entity/User.php
updated: config/packages/security.yaml


 Success!


Next Steps:
  - Review your new App\Entity\User class
  - Use make:entity to add more fields to
  - Create a way to authenticate! See ht
```

Before we do the migration, let’s add a new field named ***username***. Update the file ***src\Entity\User.php***,

src\Entity\User.php

```
<?php

namespace App\Entity;

use App\Repository\UserRepository;
use Doctrine\ORM\Mapping as ORM;
use Symfony\Component\Security\Core\User\
use Symfony\Component\Security\Core\User\

#[ORM\Entity(repositoryClass: UserReposit
class User implements UserInterface, Pas
{
    #[ORM\Id]
    #[ORM\GeneratedValue]
    #[ORM\Column(type: 'integer')]
    private $id;

    #[ORM\Column(type: 'string', length:
    private $email;

    #[ORM\Column(type: 'string', length:
    private $username;

    #[ORM\Column(type: 'json')]
    private $roles = [];

    #[ORM\Column(type: 'string')]
    private $password;
    {
        return $this->id;
    }

    public function getEmail(): ?string
    {
        return $this->email;
    }

    public function setEmail(string $ema
    {
        $this->email = $email;

        return $this;
    }

    public function getUsername(): strin
    {
        return (string) $this->username;
    }

    public function setUsername(string $
    {
        $this->username = $username;

        return $this;
    }

    /**
     * A visual identifier that represen
     *
     * @see UserInterface
     */
    public function getUserIdentifier():
    {
        return (string) $this->email;
    }

    /**
     * @see UserInterface
     */
    public function getRoles(): array
    {
        $roles = $this->roles;
        // guarantee every user at least
        $roles[] = 'ROLE_USER';

        return array_unique($roles);
    }

    public function setRoles(array $role
    {
        $this->roles = $roles;

        return $this;
    }

    /**
     * @see PasswordAuthenticatedUserInt
     */
    public function getPassword(): strin
    public function setPassword(string $
    {
        $this->password = $password;

        return $this;
    }

    /**
     * @see UserInterface
     */
    public function eraseCredentials()
    {
        // If you store any temporary, se
        // $this->plainPassword = null;
    }


}
```

Step 6: Create Migration

Then we will create a migration file and then migrate it:

Execute this command to create a migration file:

```
php bin/console make:migration
```

Then execute this command to run the migration the file:

```
php bin/console doctrine:migrations:migrat
```

Step 7: Configure JWT Bundle

We will create first the public and private keys. Execute this to generate SSL keys:

```
php bin/console lexik:jwt:generate-keypair
```

If you encounter an error while executing the command above, you can follow the command below, the command will ask for the paraphrase, the paraphrase must match the value on ***.env [JWT_PASSPHRASE]***.

```
mkdir config/jwt
```

And then we will update ***config/routes.yaml*** file:

```
api_login_check:
    path: /api/login_check
```

Step 8: Create Controllers

Let's create a registration controller to add users. Execute this command to create a controller:

```
php bin\console make:controller Registrati
```

And add these line of codes:

src/Controller/RegistrationController.php

```
<?php

namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Contro
use Symfony\Component\HttpFoundation\Resp
use Symfony\Component\Routing\Annotation\
use Symfony\Component\HttpFoundation\Requ
use Symfony\Component\PasswordHasher\Hash
use Doctrine\Persistence\ManagerRegistry;
use App\Entity\User;

/**
 * @Route("/api", name="api_")
 */

class RegistrationController extends Abst
{
    /**
     * @Route("/register", name="register
     */
    public function index(ManagerRegistry
    {

        $em = $doctrine->getManager();
        $decoded = json_decode($request->
        $email = $decoded->email;
        $plaintextPassword = $decoded->pa

        $user = new User();
        $hashedPassword = $passwordHasher
            $user,
            $plaintextPassword
        );
        $user->setPassword($hashedPasswor
        $user->setEmail($email);
        return $this->json(['message' =>
    }
}
```

We then create a Dashboard Controller to test our JWT authentication.

```
php bin/console make:controller DashboardC
```

Open the file ***src/Controller/DashboardController.php*** and an ***/api*** route:

src/Controller/DashboardController.php

```
<?php

namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Contro
use Symfony\Component\HttpFoundation\Resp
use Symfony\Component\Routing\Annotation\

/**
 * @Route("/api", name="api_")
 */

class DashboardController extends Abstrac
{
    /**
     * @Route("/dashboard", name="dashboa
     */
    public function index(): Response
    {
        return $this->json([
            'message' => 'Welcome to your
            'path' => 'src/Controller/Das
        ]);
    }
}
```

And lastly, we must configure the file ***config/packages/security.yaml*** to make the JWTauthentication work.

config/packages/security.yaml

```
security:
    enable_authenticator_manager: true
    password_hashers:
        App\Entity\User: 'auto'
        Symfony\Component\Security\Core\U
            algorithm: 'auto'
            cost:      15
    providers:
        app_user_provider:
            entity:
                class: App\Entity\User
                property: username
    firewalls:
        login:
            pattern: ^/api/login
            stateless: true
            json_login:
                check_path: /api/login_ch
                success_handler: lexik_jw
                failure_handler: lexik_jw

        api:
            pattern:   ^/api
            stateless: true
            jwt: ~
        dev:
            pattern: ^/(_(profiler|wdt)|c
            security: false
        main:
            lazy: true
            provider: app_user_provider

    access_control:
        - { path: ^/api/register, roles: 
        - { path: ^/api/login, roles: PUB
        - { path: ^/api,       roles: IS_
```

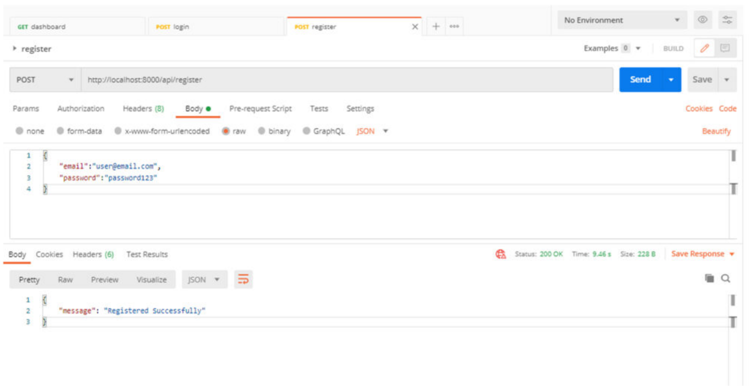
Step 10: Run the Application

After finishing the steps above, you can now run your application by executing the command below:

```
symfony server:start
```

Screenshots:

/api/register (This route will be used for registering new users)



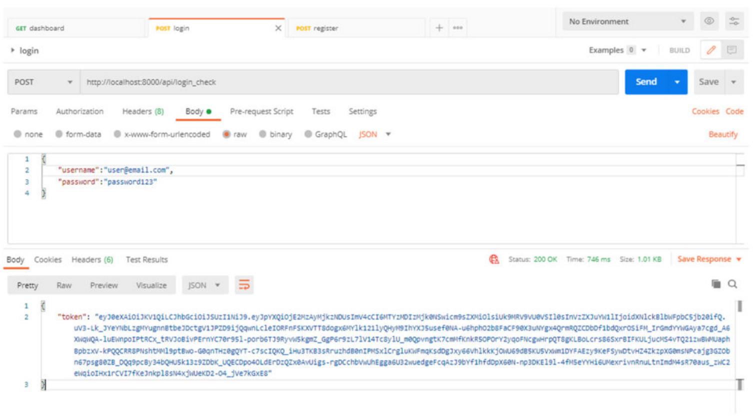
POST http://localhost:8000/api/register

```
{
    "email":"user@email.com",
    "password":"password123"
}
```

Status: 200 OK Time: 9.46 s Size: 228 B

```
{
    "message": "Registered Successfully"
}
```

/api/login_check (This route will be used for login and for getting the bearer token)



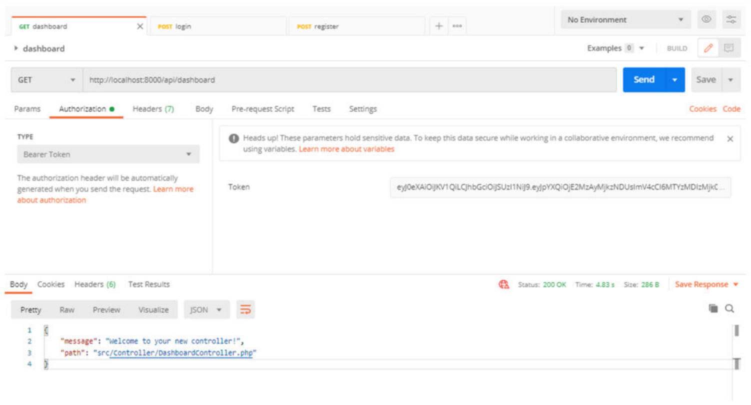
POST http://localhost:8000/api/login_check

```
{
    "username":"user@email.com",
    "password":"password123"
}
```

Status: 200 OK Time: 746 ms Size: 1.01 KB

```
{
    "token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJpYXQiOjE2MzAyMjkzNDUsImV4cCI6MTYzMDIzMjk0NSwicm9sZXMiOlsiUk9MRV9VU0VSIl0sInVzZXJuYW1lIjoidXNlckBlbWFpbC5jb20ifQ.
        uV3-Lk_JYeYNbLzgMYugnnBtbeJDctgV1JPZD9ijQqwnLcleIORFnFSKXVTT8dogx6MYlk121lyQHyM9IhYXJ5usef0NA-u6hphO2b8FaCF90X3uNYgx4QrmRQZCDbDf1bdQxrOSiFM_IrGmdYYWGAya7cgd_A6
        XWqWQA-luEWnpoIPtRCx_tRVJoBivPErnYC70r95l-porb6TJ9RyvW5kgmZ_GgP6r9zL7lV14Tc8ylU_m0QpvngtK7cmMfKnkR5OPOrY2yqoFNcgwHrpQT8gKLBoLcrs86SxrBIFKULjucMS4vTQ21zwBWMUaph
        BpbzXV-kPQQCRR8PNshtMMl9ptBwo-G0qnTHz0gQYT-c7scIQKQ_iHu3TKB3sRruzhdB0nIPMSxlCrgluKWFmqKsdDgJxy66VhlkkKjOWU69dB5KU5VXWm1DYFAEzy9KeFSywDtvHZ4ZkzpXG0msNPcajg3GZOb
        n67psg80ZB_DQq9pcBy34bQHU5k13z9ZDbK_UQECDpo4OLdErDzQZx0AvUigs-rgDCchbVwUhEgga6U32wuedgeFcqAzJ9bYf1hfdDpX60N-np3DKEl9l-4fM5eYYHi6UMexrivnRnuLtnImdM4sR70aus_zWC2
        eWqioIHx1rCVI7fKeJnkpl8sN4xjWUeKD2-O4_jVe7kGxE8"
}
```

/api/dashboard(this is a protected route), add the token on the Bearer Token to access:



GET http://localhost:8000/api/dashboard

TYPE: Bearer Token

The authorization header will be automatically generated when you send the request. Learn more about authorization

Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. Learn more about variables

Token: eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJpYXQiOjE2MzAyMjkzNDUsImV4cCI6MTYzMDIzMjkC...

Status: 200 OK Time: 4.83 s Size: 286 B

```
{
    "message": "Welcome to your new controller!",
    "path": "src/Controller/DashboardController.php"
}
```

Authentication JWT Symfony 6

« Previous Post
Generating PDF from HTML In Laravel 8

Next Post »
How To Develop Single Page Application Using ReactJS