LILLYDOO

Phone Book API Project

(Back-End Developer Guide)

This is a Back-End oriented documentation of phone-book-api project. In this document an attempt has been made to explain the structure of codebase and database so that backend developers can understand and keep on developing the code.

Important Tip:
For installation and deployment of API, please follow the instructions offered in Github repository at https://github.com/msh-shf/phone-book-api

Review some installed packages of Symfony:

install Doctrine ORM to communicate with database:

```
$ composer require symfony/orm-pack: *
```

install maker-bundle to create parts of application:

```
$ composer require --dev symfony/maker-bundle
```

install orm-fixtures to add some dummy data for testing goals:

```
$ composer require --dev orm-fixtures
```

install faker to generate some meaningful mock data:

```
$ composer require fzaninotto/faker --dev
```

install phpunit and test-pack to create, configure and run testcases:

```
$ composer require --dev phpunit/phpunit symfony/test-pack
```

install this package so that your tests will be isolated and all see the same database state:

see the documentation to see how configuration should be
https://github.com/dmaicher/doctrine-test-bundle

```
$ composer require dama/doctrine-test-bundle --dev
```

install some plugins that required for upload files:

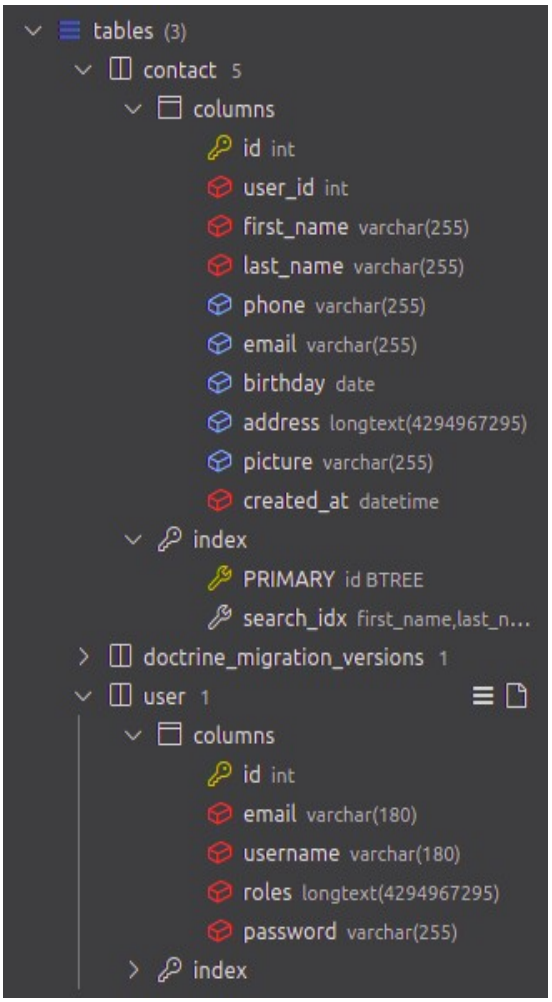
```
$ composer require symfony/translation
$ composer require symfony/mime
```

Database Structure:

After installing MySQL and setting DATABASE_URL parameter into .env and .env.test files in codebase, two databases named lillydoo and lillydoo_test have been created by below command:

```
$ php bin/console doctrine:database:create
$ php bin/console doctrine:database:create --env=test
```

There is a table named contact to store contact records and its structure is like below:



As the picture shows, there is an index for both first_name and last_name columns, because the users search contacts with name input which is implied by the concatenation of first_name and last_name.

There is an Entity class named Contact (/src/Entity/Contact.php) which corresponds to the contact table that is created by Symfony CLI make-bundle with below command:

```
$ php bin/console make:entity
```

and then with the below command the corresponding migration has been created based on the Entity class:

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

and finally with the below command the migration has been executed and the contact table created.

```
$ php bin/console doctrine:migrations:migrate
$ php bin/console doctrine:migrations:migrate --env=test  # for test database
```

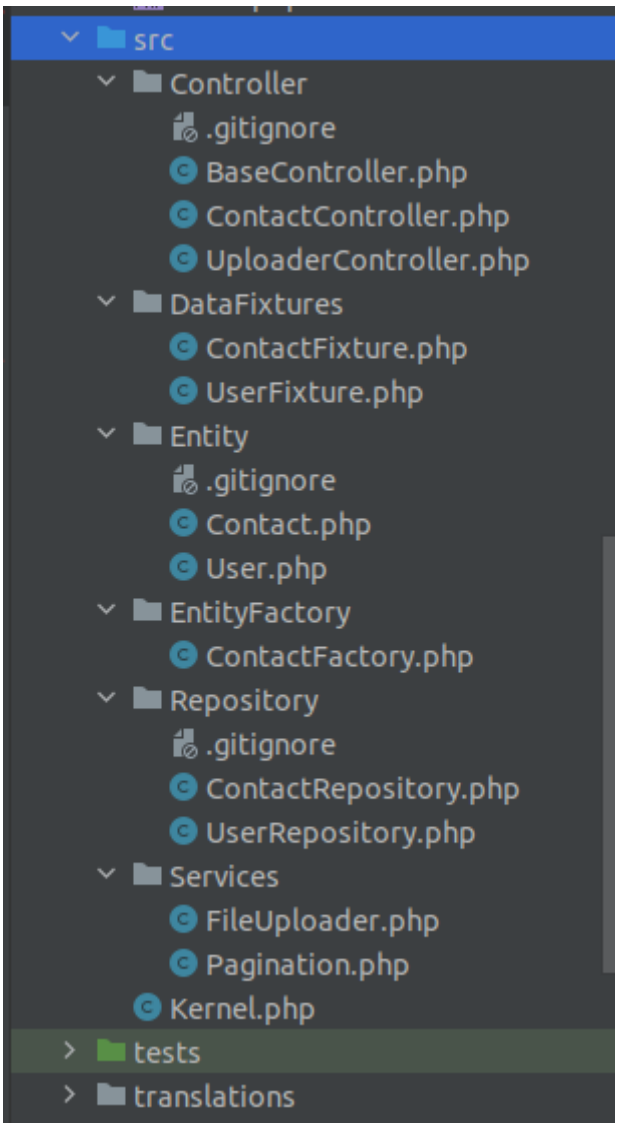
A fixture (/src/DataFixtures/ContactFixture.php) has been created by `php bin/console make:fixture ContactFixture` command to create some dummy contact data with the help of faker library for testing goals. To load fixtures the following commands has been executed for both env and test environments:

```
$ php bin/console doctrine:fixtures:load
$ php bin/console doctrine:fixtures:load --env=test  # for test database
```

To produce same data every time fixtures loaded, `$faker->seed(123456);` has been called in load method of ContactFixture. This feature is necessary for unit testing purposes.

Codebase Structure:

The main files of project shown in the following picture:



/src/Controller/BaseController.php
This is a base class for all controller classes to extend and it provides some shared features like json responses.

/src/Controller/ContactController.php
This is the main controller that implemented contact actions of the application.

/src/Controller/UploadController.php
This is a controller that used to upload contact images.
In Front-End code, if the user wants to upload an image for a contact, first a request should be sent to this route (upload/image) to upload the image and get image URL, then to create a contact the image URL should be sent among other contact parameters to the route of create contact.
The images URLs are relative urls based on public directory, so they are accessible from http://localhost:8000/IMAGE_URL

/src/EntityFactory/ContactFactory.php
This class is responsible for creating Contact entity instances and set its attributes.

/src/Services/Pagination.php
This class has been created to manage API response in a paginated manner.

/src/Services/FileUploader.php
This service implements the details of uploading a file.

JWT Token Authentication:

To enable JWT Authentication, the following steps have been done:

The required packages for jwt authentication are:

```
$ composer require jms/serializer-bundle
$ composer require friendsofsymfony/rest-bundle
$ composer require lexik/jwt-authentication-bundle
```

you should configure FOSRest Bundle in the config/packages/fos_rest.yaml file like below to match all the routes:

```
format_listener:
    rules:
        - { path: ^, prefer_extension: true, fallback_format: json, priorities: [ json, html ] }
```

To create user entity and security configs, execute this command:

```
$ php bin/console make:user
```

The username field also added to User entity manually (check the file src/Entity/User.php)

Then create and run migrations to create user table:

```
$ php bin/console make:migration
$ php bin/console doctrine:migrations:migrate
```

To generate public and private keys for jwt the following command has been executed:

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

The following section added to *config/routes.yaml* file, that shows the path of login route:

```
api_login_check:
    path: /login
```

Also the security access controls are defined in config/packages/security.yaml file. See the other sections of this file.

Automated Testing:

LoginTest:
This class (/tests/LoginTest.php) tests getting access token in two scenarios of success and failure.

Functional Tests (API Endpoints Test):

A test has been created to check API calls and the desired Responses. This test named /tests/ContactControllerTest.php and has some methods to test API Routes correspondingly.
In these tests, first an access token is taken and sent in request header parameters.

testGetContactList()	GET /api/contact?page=1&size=2
testSearchContactByName()	GET /api/contact?name=Sandy
testCreateContact()	POST /api/contact
testShowContact()	GET /api/contact/1
testUpdateContact()	PUT /api/contact/1
testDeleteContact()	DELETE /api/contact/1

Unit Tests:
/tests/ContactRepositoryTest.php is a test to check data persistence and data fetch for contact entity

Execute Tests:
to execute tests use below command:

```
$ php bin/phpunit
```

This is the result of running tests:

```
PHPUnit 9.5.24 #StandWithUkraine

Testing
.........                                                         9 / 9 (100%)

Time: 00:06.230, Memory: 28.00 MB

OK (9 tests, 41 assertions)

Remaining indirect deprecation notices (1)
```

Masih
masih118shafiee@gmail.com