Getting Started With Doctrine ORM

PHP Doctrine

When people mention "Doctrine", they are usually referring to the object-relational mapping (ORM) that is provided by the Doctrine project. The Doctrine ORM is a fantastic tool that can take away all of the effort you would have to put in to create your own ORM for interfacing with your database.

My issue with most of the on-line material covering Doctrine (especially youtube videos) is that they are usually aimed at teaching you how to use Doctrine *within the Symfony framework*. The Doctrine ORM can be used on its own, and I prefer to learn that way so I created this tutorial. If you learn how to use the tool on its own, you can apply it to *any* framework, such as Slim 3.

This tutorial will teach you how to do the following from scratch (no frameworks, just raw files/code):

1. Setup and install Doctrine.
2. Configure Doctrine for a database connection.
3. Create our first "entity".
4. Use the Doctrine CLI tools to initialize the database.
5. Use doctrine to create and return records in that database.

The complete source code for this tutorial is available on Github.

Related Posts

- PHP - Doctrine ORM Good Practices and Tricks

Setup and Install Doctrine

To get started, lets first create a structure for our project.

```
mkdir -p my-project/public_html
mkdir -p my-project/entities
mkdir -p my-project/config
mkdir -p my-project/data

touch my-project/public_html/index.php
touch my-project/bootstrap.php
touch my-project/entities/User.php
touch my-project/config/cli-config.php
```

Copy

- *public_html* - where public web content is put. E.g. our index.php file and any images/JS files.
- *entities* - folder for our doctrine entities (more on that later).
- *data* - folder for us to stick our sqlite data files that act as our database for this tutorial.
- *bootstrap.php* - This file will contain initialization code that will be used by our index.php file, and any other scripts.
- *cli-config.php* - This file, when filled in, will allow us to use the Doctrine CLI tools.

Install Doctrine

Now lets install doctrine through composer

```
cd my-project
composer require doctrine/orm
```

Copy

For this tutorial, I am using an autoloader package I created years ago, for automatically loading classes by name. You can include it by running:

```
composer require irap/autoloader
```

Copy

Configure Doctrine for a Database Connection.

Now let's fill in our `bootstrap.php` file with the following content to tell Doctrine what kind of a database we are using, and what the relevant connection details are. In this tutorial, we are configuring Doctrine to use an SQLite database to make it easy to follow along, but you could easily swap out the driver for MySQL or PostgreSQL etc..

Information about the Doctrine drivers and what configuration details they need can be found here.

```
<?php

// include the composer autoloader for autoloading packages
require_once(__DIR__ . '/vendor/autoload.php');

// set up an autoloader for loading classes that aren't in /vendor
// $classDirs is an array of all folders to load from
$classDirs = array(
    __DIR__,
    __DIR__ . '/entities',
);

new \iRAP\Autoloader\Autoloader($classDirs);

function getEntityManager() : \Doctrine\ORM\EntityManager
{
    $entityManager = null;

    if ($entityManager === null)
    {
        $paths = array(__DIR__ . '/entities');
        $config = \Doctrine\ORM\Tools\
Setup::createAnnotationMetadataConfiguration($paths);

        # set up configuration parameters for doctrine.
        # Make sure you have installed the php7.0-sqlite package.
        $connectionParams = array(
            'driver' => 'pdo_sqlite',
            'path'   => __DIR__ . '/data/my-database.db',
        );

        $entityManager = \Doctrine\ORM\EntityManager::create($connectionParams,
$config);
    }
```

```
    return $entityManager;
}
```

Copy

If you were going to bootstrap a PostgreSQL database, it would be something like below:

```
$dbParams = array(
    'driver'         => 'pdo_pgsql',
    'user'           => 'user1',
    'password'       => 'my-awesome-password',
    'host'           => 'postgresql.mydomain.com',
    'port'           => 5432,
    'dbname'         => 'myDbName',
    'charset'        => 'UTF-8',
);
```

Copy

The main bit you care about is this part which has the configuration variables required for the database and sets up the entity manager which can be used later.

```
function getEntityManager() : \Doctrine\ORM\EntityManager
{
    $entityManager = null;

    if ($entityManager === null)
    {
        $paths = array(__DIR__ . '/entities');
        $config = \Doctrine\ORM\Tools\
Setup::createAnnotationMetadataConfiguration($paths);

        # set up configuration parameters for doctrine.
        # Make sure you have installed the php7.0-sqlite package.
        $connectionParams = array(
            'driver' => 'pdo_sqlite',
            'path'   => __DIR__ . '/data/my-database.db',
        );

        $entityManager = \Doctrine\ORM\EntityManager::create($connectionParams,
$config);
    }

    return $entityManager;
}
```

Copy

I put the creation of the entity manager in a function so that you can call that function from outside bootstrap.php and you know what you are getting. E.g.

```
$em = getEntityManager();
```

Copy

I feel that this is cleaner than just leaving `$entityManager` "hanging loose" and wondering where that came from when used in other scripts. You also don't have to type-hint $entityManager to tell your IDE what it is this way either. Also, you can call that function as many times as you like from anywhere within your stack and you will still get the same single instance of the entity manager.

Create Our First Entity

The docs state that entities are PHP Objects that can be identified over many requests by a unique identifier.

Entities don't have to be a direct 1:1 mapping/relationship to the database tables, but it's easiest to think of them that way when getting started, as they will be 99% of the time.

Almost every project has a user table to represent its users, so lets create a `User` entity.

- We will use PHP annotations to tell doctrine the column types, but if you prefer, you could read up on how to use XML or YAML config files instead.
- We will use an int for the user's ID as that's quite common, but its best to switch over to UUIDs if you can.
- When creating entity classes, all properties should be private or protected, never public.

```
<?php

/**
 * @Entity @Table(name="user")
 **/

class User
{
    /** @Id @Column(type="integer") @GeneratedValue **/
    protected $id;

    /** @Column(type="string") **/
    protected $name;

    /** @Column(type="string") **/
    protected $email;


    public function __construct(string $name, sring $email)
    {
        $this->name = $name;
        $this->email = $email;
    }


    # Accessors
    public function getId() : int { return $this->id; }
    public function getName() : string { return $this->name; }
    public function getEmail() : string { return $this->email; }
}
```

Copy

The annotations such as `@Entity @Table(name="user")` are what tell Doctrine later how to set up the database so don't remove them!

Initialize The Database

Setup CLI Config

We will use the Doctrine CLI tools to setup our database. Unfortunately, they will not work straight out of the box, so we need to fill in our cli-config.php file we created earlier.

Copy/paste the following content into the file before saving.

```
<?php
require_once(__DIR__ . '/../bootstrap.php');

$entityManager = getEntityManager();
return \Doctrine\ORM\Tools\Console\
ConsoleRunner::createHelperSet($entityManager);
```

Copy

We have already done most of the heavy lifting in the *bootstrap.php* file. All we really needed was to get the entity manager and pass it to the console runner.

Use the CLI!

Now we've set up the configuration file for the CLI tools, lets use them to set up our database for us. Executethe following command:

```
php vendor/bin/doctrine orm:schema-tool:create
```

Copy

You should get the following output:

```
ATTENTION: This operation should not be executed in a production environment.

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

Obviously we wouldn't want to use that to create our database in production, but we will cover that in another tutorial.

Use Doctrine to Create and Return Records

After that has completed, you should be able to create users. Lets finally fill in our `index.php` file such that we will use doctrine to create a user, before then outputting all of the users in our system.

```
<?php

require_once(__DIR__ . '/../bootstrap.php');

// create a user
$entityManager = getEntityManager();
$user = new User("Programster", "programster@programster.org");
$entityManager->persist($user);
$entityManager->flush();

echo "Created User with ID " . $user->getId() . PHP_EOL;

// List all users:
$users = $entityManager->getRepository("User")->findAll();
print "Users: " . print_r($users, true) . PHP_EOL;
```

Copy

Below is an example of the output you will get if you call the `index.php` file 4 times.

```
Created User with ID 4
Users: Array
(
    [0] => User Object
        (
            [id:protected] => 1
            [name:protected] => Programster
            [email:protected] => programster@programster.org
        )

    [1] => User Object
        (
            [id:protected] => 2
            [name:protected] => Programster
            [email:protected] => programster@programster.org
        )

    [2] => User Object
        (
            [id:protected] => 3
            [name:protected] => Programster
            [email:protected] => programster@programster.org
        )

    [3] => User Object
        (
            [id:protected] => 4
            [name:protected] => Programster
            [email:protected] => programster@programster.org
        )

)
```

Conclusion

Hopefully that's given you a taste for using Doctrine and the benefits it can provide. However, there is still much more to learn. For now, please refer to the references below for more information.

References

- Doctrine-project.org - DBAL Configuration
- Doctrine-project.org - Doctrine ORM Docs
- Stack Overflow - Doctrine 2 - Get all Records

https://blog.programster.org/getting-started-with-doctrine-orm