Министерство образования Республики Беларусь
Учреждение образования
«Брестский государственный технический университет» Кафедра ИИТ

Отчет по лабораторной работе 7
Дисциплина “ПрИС”

Выполнила:

Студентка группы ПО-3

Ковалева А. И.

Проверил:

Лаврущик А. И.

Брест 2021

Цель: познакомиться с практической реализацией принципа инверсии зависимостей, а также протоколами межсервисного взаимодействия.
Предметная область: блог о фермерском хозяйстве.

Задание для выполнения: реализуйте механизмы хранения (persistence) для ранее разработанного приложения – технологию, фреймворк, ORM выберите сами. Это могут быть реляционная БД, NoSQL, InMemory, файловая система, Redis и т.д.
Реализуйте минимум два механизма доставки – способов вызова функций вашего приложения. Например, REST и командная строка.
Отдельно реализуйте микросервис, который не будет выполнять никакой полезной работы, кроме вызова какой-либо функции вашего приложения по протоколу gRPC (третий способ доставки для вашего приложения), используйте Google Protobuf. Большим плюсом станет, если данный микросервис будет реализован не на PHP.

Ход работы

Реализация ORM-репозитория:

```php
<?php

namespace App\Repository;

use App\Entity\Note;
use Doctrine\Bundle\DoctrineBundle\Repository\ServiceEntityRepository;
use Doctrine\Persistence\ManagerRegistry;

/**
 * @method Note|null find($id, $lockMode = null, $lockVersion = null)
 * @method Note|null findOneBy(array $criteria, array $orderBy = null)
 * @method Note[]    findAll()
 * @method Note[]    findBy(array $criteria, array $orderBy = null, $limit = null,
$offset = null)
 */
class NoteRepository extends ServiceEntityRepository
{
    public function __construct(ManagerRegistry $registry)
    {
        parent::__construct($registry, Note::class);
    }
}
```

Реализация REST API Controller:

```php
<?php

namespace App\Controller;

use App\Entity\Note;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\Routing\Annotation\Route;
use Symfony\Component\Serializer\Encoder\JsonEncoder;
use Symfony\Component\Serializer\Serializer;
use Symfony\Component\Serializer\Normalizer\ObjectNormalizer;
use Symfony\Component\HttpFoundation\Response;

class NoteApiController extends AbstractController
{
    /**
     * @Route("/api/notes", name="api_notes")
     */
    public function getAllNotes()
    {
        $entityManager = $this->getDoctrine()->getManager();
        $notes = $entityManager->getRepository(Note::class)->findAll();

        $encoders = [new JsonEncoder()];
        $normalizers = [new ObjectNormalizer()];
        $serializer = new Serializer($normalizers, $encoders);

        $notes_json = [];

        foreach ($notes as $note) {
            $note_json = [];
            $note_json['id'] = $note->getId();
            $note_json['title'] = $note->getTitle();
            $note_json['content'] = $note->getContent();

            $user_json=[];
            $user_json['id'] = $note->getUser()->getId();
            $user_json['name'] = $note->getUser()->getName();
            $user_json['surname'] = $note->getUser()->getName();
            $user_json['specialization'] = $note->getUser()->getSpecialization()-
>getTitle();

            $note_json['user'] = $user_json;

            array_push($notes_json, $note_json);
        }
```

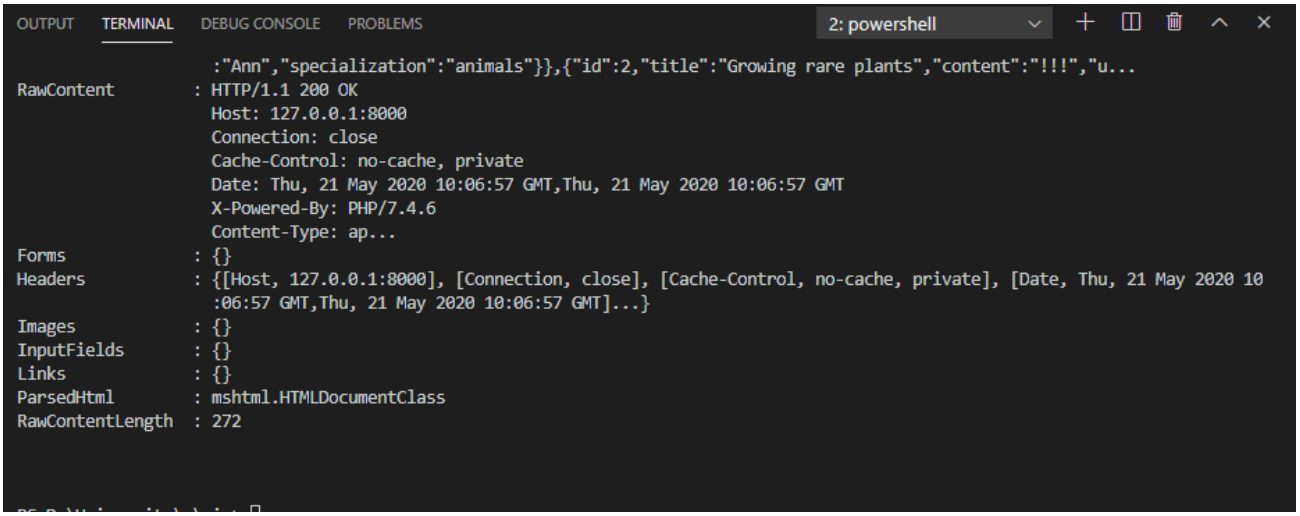
```
        $response = new Response($serializer->serialize($notes_json, 'json'));
        $response->headers->set('Content-Type', 'application/json');

        return $response;
    }
}
```

Результат:

Через браузер

Через консоль



Реализация Protobuf:

Car.proto

```
syntax = "proto3";

message Car
{
  int32 id = 1;
  string model = 2;
  string year = 3;
  int32 mileage = 4;
}
```

ProtoFileReader.php

```php
<?php
include_once './vendor/autoload.php';
include_once 'GPBMetadata/Car.php';
include_once 'Car.php';

$car = new Car();
$car->setId(23);
$car->setModel('Volvo S80');
$car->setYear('1999');
$car->setMileage(700000);

file_put_contents('car.txt', $car->serializeToString());

$car_str = file_get_contents('./car.txt');

$car_updated = new Car();

try {
    $car_updated->mergeFromString($car_str);
} catch (Exception $e) {
    echo 'ERROR!';
}

echo $car_updated->serializeToJsonString();

$car_updated->setMileage(800000);
$car_updated->setModel($car_updated->getModel() . ' S90');

echo $car_updated->serializeToJsonString();

file_put_contents('car.txt', $car_updated->serializeToString());
```

Вывод в консоль:

```
{"id":23,"model":"Volvo S80","year":"1999","mileage":700000}{"id":23,"model":"Volvo S80 S90","year":"1999","mileage":800000}
```

Вывод: я познакомилась с практической реализацией принципа инверсии зависимостей, а также протоколами межсервисного взаимодействия.