Министерство образования Республики Беларусь
Учреждение образования
«Брестский государственный университет»
Кафедра «ЭВМ и системы»

Лабораторная работа №7

Инфраструктура: хранение и доставка. Межсервисное взаимодействие

Выполнил:
студент группы:
ПО-3
Будяков В.В.
Проверил:
Лаврущик А.И.

Брест 2021

Тема: «Инфраструктура: хранение и доставка. Межсервисное взаимодействие»

Цель: познакомиться с практической реализацией принципа инверсии зависимостей, а также протоколами межсервисного взаимодействия.
Предметная область: управление парковкой.

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.

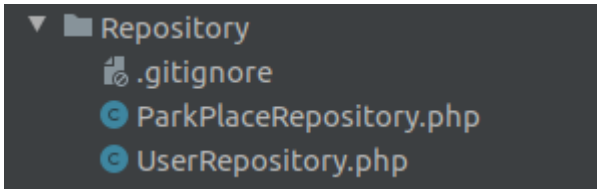
Ход работы

Для начала реализуем ORM-репозиторий:

```
<?php
namespace App\Repository;
use App\Entity\User;
use Doctrine\Bundle\DoctrineBundle\Repository\ServiceEntityRepository;
use Doctrine\Common\Persistence\ManagerRegistry;
use Doctrine\ORM\OptimisticLockException;
use Doctrine\ORM\ORMException;
use Symfony\Component\Security\Core\Exception\UnsupportedUserException;
use Symfony\Component\Security\Core\User\PasswordUpgraderInterface;
use Symfony\Component\Security\Core\User\UserInterface;
/**
 * @method User|null find($id, $lockMode = null, $lockVersion = null)
 * @method User|null findOneBy(array $criteria, array $orderBy = null)
 * @method User[]    findAll()
 * @method User[]    findBy(array $criteria, array $orderBy = null, $limit = null, $offset = null)
 */
class UserRepository extends ServiceEntityRepository implements PasswordUpgraderInterface
{
   public function __construct(ManagerRegistry $registry)
   {
      parent::__construct($registry, User::class);
   }
   /**
```

```
    * Used to upgrade (rehash) the user's password automatically over time.
    */
   public function upgradePassword(UserInterface $user, string $newEncodedPassword): void
   {
      if (!$user instanceof User) {
         throw new UnsupportedUserException(sprintf('Instances of "%s" are not supported.',
\get_class($user)));
      }
      $user->setPassword($newEncodedPassword);
      $this->_em->persist($user);
      $this->_em->flush();
   }
   public function addUser(User $user): bool
   {
      try {
         $this->_em->persist($user);
         $this->_em->flush();
         return true;
      } catch (OptimisticLockException|ORMException $e) {
         return false;
      }
   }
}
```

Список всех репозиториев:



Реализуем REST Controller:

```
<?php
namespace App\Controller;
use App\Service\ParkPlaceService;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;
use Symfony\Component\Serializer\Encoder\JsonEncoder;
use Symfony\Component\Serializer\Normalizer\ObjectNormalizer;
use Symfony\Component\Serializer\Serializer;
class ParkPlaceRestController extends AbstractController
{
   private $parkPlaceService;
   public function __construct(ParkPlaceService $placeService)
   {
      $this->parkPlaceService = $placeService;
```

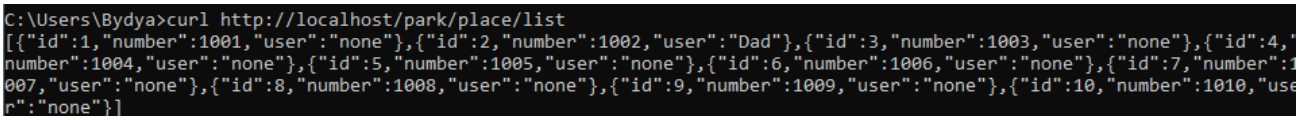
```
    }
    /**
     * @Route("/park/place/list", name="park_place_list")
     * @return Response
     */
    public function listParkingPlaces()
    {
        $parkPlaces = $this->parkPlaceService->findAll();
        $encoders = [new JsonEncoder()];
        $normalizers = [new ObjectNormalizer()];
        $serializer = new Serializer($normalizers, $encoders);
        $parkPlacesJson = [];
        foreach ($parkPlaces as $parkPlace) {
            $parkPlaceJson = [];
            $parkPlaceJson['id'] = $parkPlace->getId();
            $parkPlaceJson['number'] = $parkPlace->getNumber();
            $user = $parkPlace->getUser();
            $parkPlaceJson['user'] = $user != null ? $user->getUsername() : 'none';
            array_push($parkPlacesJson, $parkPlaceJson);
        }
        $response = new Response($serializer->serialize($parkPlacesJson, 'json'));
        $response->headers->set('Content-Type', 'application/json');
        return $response;
    }
}
```

Результат:

- Через Postman

```
[
    {
        "id": 1,
        "number": 1002,
        "user": "Bydya"
    },
    {
        "id": 2,
        "number": 1003,
        "user": "Dad"
    },
    {
        "id": 3,
        "number": 1001,
        "user": "none"
    },
    {
        "id": 4,
        "number": 1004,
        "user": "Bydya"
    },
    {
        "id": 5,
        "number": 1005,
        "user": "none"
    },
    {
        "id": 6,
        "number": 1006,
        "user": "none"
    },
    {
        "id": 7,
        "number": 1007,
        "user": "none"
    },
    ;
```

- Через консоль



Реализация Protobuf:

Серверная часть:

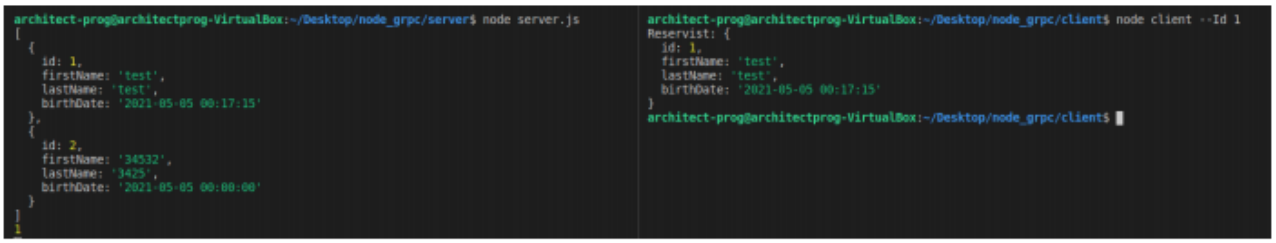
```
const PROTO_PATH = 'definitions.proto';

const grpc = require('@grpc/grpc-js');

const protoLoader = require('@grpc/proto-loader');

const packageDefinition = protoLoader.loadSync(

PROTO_PATH,

{

keepCase: true,

longs: String,

enums: String,

defaults: true,

oneofs: true

});

var hello_proto = grpc.loadPackageDefinition(packageDefinition);

function Reservist(call, callback) {

var fetch = require('node-fetch');

fetch('http://localhost:8001/Api/Reservists.php')

.then(res => {

if(res.ok) return res

else throw new Error(`The HTTP status of the reponse: ${res.status} (${res

.statusText})`)

})

.then(res => res.json())

.then(json => {

console.log(json);
```

```
console.log(call.request.id);
const data = json;
const query = data.filter(el => el.id === call.request.id)
callback(null, query[0]);
})
}
function main() {
const server = new grpc.Server();
server.addService(hello_proto.ReservistService.service, {Reservist: Reservist});
server.bindAsync('0.0.0.0:50051', grpc.ServerCredentials.createInsecure(), () => {
server.start();
});
}
main();
```

Клиентская часть:

```
const PROTO_PATH = 'definitions.proto';
var parseArgs = require('minimist');
var grpc = require('@grpc/grpc-js');
var protoLoader = require('@grpc/proto-loader');
var packageDefinition = protoLoader.loadSync(
PROTO_PATH,
{keepCase: true,
longs: String,
enums: String,
defaults: true,
oneofs: true
});
var hello_proto = grpc.loadPackageDefinition(packageDefinition);
function main() {
var argv = parseArgs(process.argv.slice(1));
```

```
const customerID = argv['Id'];

var client = new hello_proto.ReservistService('localhost:50051',

grpc.credentials.createInsecure());

client.Reservist({id: customerID}, function(err, response) {

console.log('Reservist:', response);

});

}

main();
```



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