

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

Отчет по лабораторной работе 7
Дисциплина “ПриС”

Выполнил:

Студент группы ПО-3

Кабачук Д. С.

Проверил:

Лаврущик А. И.

Брест 2021

Вариант 11

“Инфраструктура: хранение и доставка. Межсервисное взаимодействие“

Цель работы: Познакомиться с практической реализацией принципа инверсии зависимостей, а также протоколами межсервисного взаимодействия

Постановка задачи: 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.

Предметная область: Космический туризм.

Ход работы

PassengersApiController.php:

```
<?php

namespace App\Controller;

use App\Entity\Passenger;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\Console\Application;
use Symfony\Component\Console\Input\ArrayInput;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;

class PassengersApiController extends AbstractController
{
    /**
     * @Route("/api/create_passenger", name="api_create_passenger",
methods="POST")
     * @throws \Exception
     */
    public function createPassenger(Request $request)
    {
        $em = $this->getDoctrine()->getManager();
```

```

        $passenger = new Passenger();
        $data = json_decode($request->getContent());
        $passenger->setLastName($data->lastName);
        $passenger->setTix($data->tix);
        $em->persist($passenger);
        $em->flush();

        exec('node ../protobuf/write.js 0 1 0');

        return new Response(json_encode(200));
    }

    /**
     * @Route("/api/update_passenger/{id}",
name="api_update_passenger")
     */
    public function updatePassenger(Request $request, $id)
    {
        $em = $this->getDoctrine()->getManager();

        $passenger = $em->getRepository(Passenger::class)->find($id);
        $data = json_decode($request->getContent());
        $passenger->setLastName($data->lastName);
        $passenger->setTix($data->tix);
        $em->persist($passenger);
        $em->flush();

        return new Response(json_encode(["id" => $passenger->getId(),
"last_name" => $passenger->getLastName(),
        "tix" => $passenger->getTix()]));
    }

    /**
     * @Route("/api/delete_passenger/{id}",
name="api_delete_passenger", methods="DELETE")
     */
    public function deletePassenger($id)
    {
        $em = $this->getDoctrine()->getManager();
        $passenger = $em->getRepository(Passenger::class)->find($id);
        $em->remove($passenger);
        $em->flush();

        return new Response(json_encode(["id" => $passenger->getId(),
"last_name" => $passenger->getLastName(),
        "tix" => $passenger->getTix()]));
    }

    /**
     * @Route("/api/all_passengers", name="api_all_passengers")
     */
    public function allPassengers()
    {
        $em = $this->getDoctrine()->getManager();
        $passengers = $em->getRepository(Passenger::class)->findAll();
        $arr = [];
        foreach ($passengers as $passenger) {
            $newShip = ["id" => $passenger->getId(), "last_name" =>

```

```

$passenger->getLastName(),
        "tix" => $passenger->getTix()];
        array_push($arr, $newShip);
    }
    return new Response(json_encode($arr));
}
}

```

TravelsApiController.php:

```
<?php
```

```
namespace App\Controller;
```

```

use App\Domains\ChooseShip;
use App\Entity\Passenger;
use App\Entity\Ship;
use App\Entity\Travel;
use App\Form\TravelType;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\Console\Application;
use Symfony\Component\Console\Input\ArrayInput;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;

```

```

class TravelsApiController extends AbstractController
{
    /**
     * @Route("/api/create_travel", name="api_create_travel",
methods="POST")
     * @throws \Exception
     */
    public function createTravel(Request $request)
    {
        $em = $this->getDoctrine()->getManager();
        $travel = new Travel();
        $data = json_decode($request->getContent());
        try {
            $chooseShip = new ChooseShip($data->shipId, $em);
            $ship = $chooseShip->getShip();
            $travel->setShip($ship);
            $travel->setPrice($data->price);
            $travel->setShipId($data->shipId);
            $travel->setWhereFrom($data->whereFrom);
            $travel->setWhereTo($data->whereTo);
            $em->persist($ship);
            $em->persist($travel);
            $em->flush();
        } catch (\Error $exception) {
            return new Response(json_encode(500));
        }

        exec('node ../protobuf/write.js 0 0 1');

        return new Response(json_encode(200));
    }
}

```

```

/**
 * @Route("/api/update_travel/{id}", name="api_update_travel",
methods="PUT")
 */
public function updateTravel(Request $request, $id)
{
    $em = $this->getDoctrine()->getManager();

    $travel = $em->getRepository(Travel::class)->find($id);
    $data = json_decode($request->getContent());
    $ship = $em->getRepository(Ship::class)->find($data->shipId);
    if ($travel->getShipId() !== $ship->getId()) {
        $oldShip =
$em->getRepository(Ship::class)->find($travel->getShipId());
        $oldShip->setCurrentNumber($oldShip->getCurrentNumber() -
1);

        $em->persist($oldShip);
        $currentNum = $ship->getCurrentNumber();
        if ($currentNum === $ship->getCapacity()) {
            return new Response(json_encode(500));
        }
        $ship->setCurrentNumber($currentNum + 1);
    }
    $travel->setShip($ship);
    $travel->setPrice($data->price);
    $travel->setShipId($data->shipId);
    $travel->setWhereFrom($data->whereFrom);
    $travel->setWhereTo($data->whereTo);
    $em->persist($ship);
    $em->persist($travel);
    $em->flush();
    return new Response(json_encode(["id" => $travel->getId(),
"where_from" => $travel->getWhereFrom(),
        "where_to" => $travel->getWhereFrom(), "price" =>
$travel->getPrice(), "ship_id" => $travel->getShipId()]));
}

/**
 * @Route("/api/delete_travel/{id}", name="api_delete_travel",
methods="DELETE")
 */
public function deleteTravel($id)
{
    $em = $this->getDoctrine()->getManager();
    $travel = $em->getRepository(Travel::class)->find($id);
    $ship =
$em->getRepository(Ship::class)->find($travel->getShipId);
    $passengers =
$em->getRepository(Passenger::class)->findBy(["tix" => $id]);
    foreach ($passengers as $passenger) {
        $em->remove($passenger);
    }
    $currentNum = $ship->getCurrentNumber();
    $ship->setCurrentNumber($currentNum + 1);
    $em->persist($ship);
    $em->remove($travel);
    $em->flush();
}

```

```

        return new Response(json_encode(["id" => $stravel->getId(),
"where_from" => $stravel->getWhereFrom(),
        "where_to" => $stravel->getWhereFrom(), "price" =>
$stravel->getPrice(), "ship_id" => $stravel->getShipId()]));
    }

    /**
     * @Route("/api/all_travels", name="api_all_travels")
     */
    public function allTravels()
    {
        $em = $this->getDoctrine()->getManager();
        $travels = $em->getRepository(Travel::class)->findAll();
        $arr = [];
        foreach ($travels as $stravel){
            $newTravel = ["id" => $stravel->getId(), "where_from" =>
$stravel->getWhereFrom(),
                "where_to" => $stravel->getWhereFrom(), "price" =>
$stravel->getPrice(), "ship_id" => $stravel->getShipId()];
            array_push($arr, $newTravel);
        }
        return new Response(json_encode($arr));
    }
}

```

ShipsApiController.php:

<?php

```

namespace App\Controller;

use App\Entity\Passenger;
use App\Entity\Ship;
use App\Entity\Travel;
use Symfony\Component\Console\Application;
use Symfony\Component\Console\Input\ArrayInput;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\Routing\Annotation\Route;

class ShipsApiController extends AbstractController
{
    /**
     * @Route("/api/create_ship", name="api_create_ship",
methods="POST")
     * @throws \Exception
     */
    public function createShip(Request $request)
    {
        $em = $this->getDoctrine()->getManager();
        $ship = new Ship();
        $data = json_decode($request->getContent());
        $ship->setName($data->name);
        $ship->setCapacity($data->capacity);
        $ship->setCurrentNumber(0);
        $em->persist($ship);
        $em->flush();
    }
}

```

```

        exec('node ../protobuf/write.js 1 0 0');

        return new Response(json_encode(200));
    }

    /**
     * @Route("/api/update_ship/{id}", name="api_update_ship",
methods="PUT")
     */
    public function updateShip(Request $request, $id)
    {
        $em = $this->getDoctrine()->getManager();
        $ship = $em->getRepository(Ship::class)->find($id);
        $data = json_decode($request->getContent());
        $ship->setName($data->name);
        $ship->setCapacity($data->capacity);
        $ship->setCurrentNumber($data->current_number);
        $em->persist($ship);
        $em->flush();
        return new Response(json_encode(["id" => $id, "name" =>
$ship->getName(),
            "capacity" => $ship->getCapacity(), "current_number" =>
$ship->getCurrentNumber()]));
    }

    /**
     * @Route("/api/delete_ship/{id}", name="api_delete_ships",
methods="DELETE")
     */
    public function deleteShip($id)
    {
        $em = $this->getDoctrine()->getManager();
        $ship = $em->getRepository(Ship::class)->find($id);
        $travels = $em->getRepository(Travel::class)->findBy(['ship_id'
=> $id]);
        foreach ($travels as $travel) {
            $passengers =
$em->getRepository(Passenger::class)->findBy(["tix" =>
$travel->getId()]);
            foreach ($passengers as $passenger) {
                $em->remove($passenger);
            }
            $em->remove($travel);
        }
        $em->remove($ship);
        $em->flush();
        return new Response(json_encode(["id" => $id, "name" =>
$ship->getName(),
            "capacity" => $ship->getCapacity(), "current_number" =>
$ship->getCurrentNumber()]));
    }

    /**
     * @Route("/api/all_ships", name="api_all_ships")
     */
    public function allShips()
    {

```

```

        $em = $this->getDoctrine()->getManager();
        $ships = $em->getRepository(Ship::class)->findAll();
        $arr = [];
        foreach ($ships as $ship){
            $newShip = ["id" => $ship->getId(), "name" =>
$ship->getName(),
                "capacity" => $ship->getCapacity(), "current_number" =>
$ship->getCurrentNumber()];
            array_push($arr, $newShip);
        }
        return new Response(json_encode($arr));
    }
}

```

Для реализации задания с протобафом был выбран язык JS.

Info.proto

```
syntax = "proto2";
```

```
package tutorial;
```

```

message Info {
    required int32 count_of_ship = 1;
    required int32 count_of_person = 2;
    required int32 count_of_tix = 3;
}

```

write.js

```

const Info = require('./info_pb').Info;
const fs = require('fs');

const info = new Info();
let oldInfo = {};
try {
    oldInfo = Info.deserializeBinary(
        fs.readFileSync('/home/safed/Рабочий стол/Политех/ПИС/lab1/protobuf/info.txt')
    ).toObject();
} catch (e) {
    oldInfo.countOfShip = 0;
    oldInfo.countOfPerson = 0;
    oldInfo.countOfTix = 0;
} finally {
    const [app, name, ship, person, tix] = [...process.argv];
    if(+ship === 1) {
        info.setCountOfShip(oldInfo.countOfShip + 1);
        info.setCountOfPerson(oldInfo.countOfPerson);
        info.setCountOfTix(oldInfo.countOfTix);
    }
    if(+person === 1) {
        info.setCountOfShip(oldInfo.countOfShip);
        info.setCountOfPerson(oldInfo.countOfPerson + 1);
        info.setCountOfTix(oldInfo.countOfTix);
    }
}

```

```

    }
    if(+tix === 1) {
        info.setCountOfShip(oldInfo.countOfShip);
        info.setCountOfPerson(oldInfo.countOfPerson);
        info.setCountOfTix(oldInfo.countOfTix + 1);
    }
    fs.writeFileSync('/home/safed/Рабочий стол/Политех/ПИС/lab1/protobuf/info.txt',
info.serializeBinary());
}

```

read.js

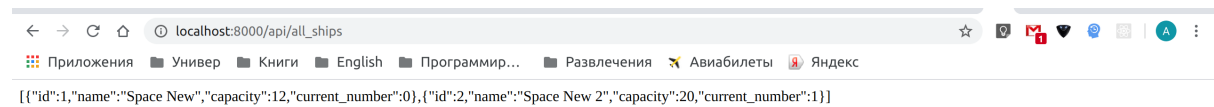
```

const Info = require('./info_pb').Info;
const fs = require('fs');

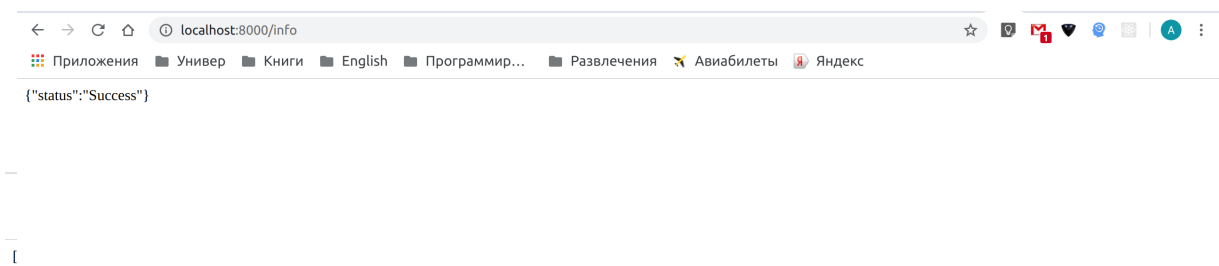
const info = Info.deserializeBinary(
    fs.readFileSync('/home/safed/Рабочий стол/Политех/ПИС/lab1/protobuf/info.txt')
).toObject();
fs.writeFileSync('/home/safed/Рабочий стол/Политех/ПИС/lab1/protobuf/statistics.json',
JSON.stringify(info));

```

Результаты выполнения:



The screenshot shows a web browser window with the address bar displaying 'localhost:8000/api/all_ships'. The browser's bookmark bar includes 'Приложения', 'Универ', 'Книги', 'English', 'Программир...', 'Развлечения', 'Авиабилеты', and 'Яндекс'. The main content area of the browser displays a JSON array of two ship objects: `[{"id":1,"name":"Space New","capacity":12,"current_number":0}, {"id":2,"name":"Space New 2","capacity":20,"current_number":1}]`.



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

