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

Отчет
По лабораторной работе №7
по дисциплине «ПрИС»

Выполнила
студентка 3 курса
группы ПО-3
Пивчик В.Г.
Проверил
Лаврущик А.И.

Брест 2021

Лабораторная работа №7

Инфраструктура: хранение и доставка
Межсервисное взаимодействие

ВАРИАНТ 9

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

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.

Предметная область. Продажа электрооборудования.

Ход работы

Для реализации Persistence-layer'а используем реляционную SQL базу данных. Для использования данной SQLite раскомментируем строку в файле .env.

DATABASE_URL="sqlite:///%kernel.project_dir%/var/data.db"

Сформируем миграции:

```
public function up(Schema $schema): void
    {
        // this up() migration is auto-generated, please modify it to your needs
        $this->addSql('CREATE TABLE new_client (id INTEGER PRIMARY KEY
AUTOINCREMENT NOT NULL, name VARCHAR(255) NOT NULL)');
        $this->addSql('DROP INDEX IDX_6117D13B1FB354CD');
        $this->addSql('CREATE TEMPORARY TABLE __temp__purchase AS SELECT id,
membership_id, name, cost, version FROM purchase');
        $this->addSql('DROP TABLE purchase');
        $this->addSql('CREATE TABLE purchase (id INTEGER PRIMARY KEY
AUTOINCREMENT NOT NULL, membership_id INTEGER DEFAULT NULL, name VARCHAR(255)
NOT NULL COLLATE BINARY, cost INTEGER NOT NULL, version INTEGER NOT NULL,
CONSTRAINT FK_6117D13B1FB354CD FOREIGN KEY (membership_id) REFERENCES membership
(id) NOT DEFERRABLE INITIALLY IMMEDIATE)');
        $this->addSql('INSERT INTO purchase (id, membership_id, name, cost,
version) SELECT id, membership_id, name, cost, version FROM __temp__purchase');
```

```
        $this->addSql('DROP TABLE __temp__purchase');
        $this->addSql('CREATE INDEX IDX_6117D13B1FB354CD ON purchase
(membership_id)');
        $this->addSql('DROP INDEX IDX_CECE98A619EB6921');
        $this->addSql('CREATE TEMPORARY TABLE __temp__shopping_session AS SELECT
id, client_id, date FROM shopping_session');
        $this->addSql('DROP TABLE shopping_session');
        $this->addSql('CREATE TABLE shopping_session (id INTEGER PRIMARY KEY
AUTOINCREMENT NOT NULL, client_id INTEGER NOT NULL, date DATE NOT NULL,
CONSTRAINT FK_CECE98A619EB6921 FOREIGN KEY (client_id) REFERENCES client (id)
NOT DEFERRABLE INITIALLY IMMEDIATE)');
        $this->addSql('INSERT INTO shopping_session (id, client_id, date) SELECT
id, client_id, date FROM __temp__shopping_session');
        $this->addSql('DROP TABLE __temp__shopping_session');
        $this->addSql('CREATE INDEX IDX_CECE98A619EB6921 ON shopping_session
(client_id)');
        $this->addSql('DROP INDEX IDX_93CC5B6B558FBEB9');
        $this->addSql('DROP INDEX IDX_93CC5B6B7EF00B89');
        $this->addSql('CREATE TEMPORARY TABLE __temp__shopping_session_purchase
AS SELECT shopping_session_id, purchase_id FROM shopping_session_purchase');
        $this->addSql('DROP TABLE shopping_session_purchase');
        $this->addSql('CREATE TABLE shopping_session_purchase
(shopping_session_id INTEGER NOT NULL, purchase_id INTEGER NOT NULL, PRIMARY
KEY(shopping_session_id, purchase_id), CONSTRAINT FK_93CC5B6B7EF00B89 FOREIGN
KEY (shopping_session_id) REFERENCES shopping_session (id) ON DELETE CASCADE NOT
DEFERRABLE INITIALLY IMMEDIATE, CONSTRAINT FK_93CC5B6B558FBEB9 FOREIGN KEY
(purchase_id) REFERENCES purchase (id) ON DELETE CASCADE NOT DEFERRABLE
INITIALLY IMMEDIATE)');
        $this->addSql('INSERT INTO shopping_session_purchase
(shopping_session_id, purchase_id) SELECT shopping_session_id, purchase_id FROM
__temp__shopping_session_purchase');
        $this->addSql('DROP TABLE __temp__shopping_session_purchase');
        $this->addSql('CREATE INDEX IDX_93CC5B6B558FBEB9 ON
shopping_session_purchase (purchase_id)');
        $this->addSql('CREATE INDEX IDX_93CC5B6B7EF00B89 ON
shopping_session_purchase (shopping_session_id)');
    }
```

Создадим репозиторий для взаимодействия с данными ClientRepository

```
<?php

namespace App\Repository;

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

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

    public function add(Client $client)
```

```
    {
        $this->addClient($client->getName(), $client->getNumber());
    }

    public function addClient(string $name, int $number): Client
    {
        $entityManager = $this->getEntityManager();
        $client = new Client();
        $client->setName($name);
        $client->setNumber($number);
        $entityManager->persist($client);
        $entityManager->flush();

        return $client;
    }

    public function getClientsCount(): int
    {
        $entityManager = $this->getEntityManager();
        $records = $entityManager->getRepository(Client::class)->findAll();
        return count($records);
    }
}
```

PurchaseRepository

```
<?php

namespace App\Repository;

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

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

    public function addPurchase(string $name, int $cost, int $version): Purchase
    {
        $entityManager = $this->getEntityManager();
        $purchase = new Purchase();
        $purchase->setName($name);
        $purchase->setCost($cost);
        $purchase->setVersion($version);

        $entityManager->persist($purchase);
        $entityManager->flush();

        return $purchase;
    }

    public function getPurchasesCount(): int
    {
```

```
        $entityManager = $this->getEntityManager();
        $records = $entityManager->getRepository(Purchase::class)->findAll();
        return count($records);
    }
}
```

ShoppingSessionRepository

```
<?php

namespace App\Repository;

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

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

    public function addShoppingSession(\DateTimeInterface $date,
\App\Entity\Purchase $purchase, \App\Entity\Client $client)
    {
        $entityManager = $this->getEntityManager();
        $shoppingSession = new ShoppingSession();
        $shoppingSession->setDate($date);
        $shoppingSession->setClient($client);
        $shoppingSession->addPurchase($purchase);

        $entityManager->persist($shoppingSession);
        $entityManager->flush();
    }

    public function getShoppingSessionCount(): int
    {
        $entityManager = $this->getEntityManager();
        $records = $entityManager->getRepository(ShoppingSession::class)-
>findAll();
        return count($records);
    }
}
```

Создадим Rest Api Controller.

MainApiController

```
<?php

namespace App\Controller;

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;
use App\Entity\Purchase;


class MainApiController extends AbstractController
{
    #[Route('/main/api', name: 'main_api')]
    public function index(): Response
    {
        return $this->render('main_api/index.html.twig', [
            'controller_name' => 'MainApiController',
        ]);
    }

    public function getAllPurchases()
    {
        $entityManager = $this->getDoctrine()->getManager();
        $purchases = $entityManager->getRepository(Purchase::class)->findAll();
        $encoders = [new JsonEncoder()];
        $normalizers = [new ObjectNormalizer()];
        $serializer = new Serializer($normalizers, $encoders);
        $result_json = [];
        foreach ($purchases as $purchase)
        {
            $vpurchase_json = [];
            $vpurchase_json['id'] = $purchase->getId();
            $vpurchase_json['name'] = $purchase->getName();
            $vpurchase_json['cost'] = $purchase->getCost();
            $vpurchase_json['version'] = $purchase->getVersion();
            array_push($result_json, $vpurchase_json);
}
        $response = new Response($serializer->serialize($result_json,
'json'));
        $response->headers->set('Content-Type', 'application/json');
        return $response;
    }
}
```

Результат работы

```
[{"id":1,"name":"iphone","cost":200,"version":1},{"id":2,"name":"macbook","cost":1000,"version":16}]
```

Доступ из консоли реализуем посредством скрипта:

```
<?php shell_exec('php bin/console doctrine:query:sql "select * from purchases;"');
```

```
array(2) {
  [0]=>
  array(5) {
    ["id"]=>
    string(1) "1"
    ["membership_id"]=>
    string(1) "1"
    ["name"]=>
    string(6) "iphone"
    ["cost"]=>
    string(3) "200"
    ["version"]=>
    string(1) "1"
  }
  [1]=>
  array(5) {
    ["id"]=>
    string(1) "2"
    ["membership_id"]=>
    string(1) "2"
    ["name"]=>
    string(7) "macbook"
    ["cost"]=>
    string(4) "1000"
    ["version"]=>
    string(2) "16"
  }
}
valeriadmitruk@MacBook-Pro-Valeria electronics_shop %
```

Далее реализуем микросервис. Описание данных:

```
syntax = "proto3"
message test {
    int32 associated_value = 0;
}
```

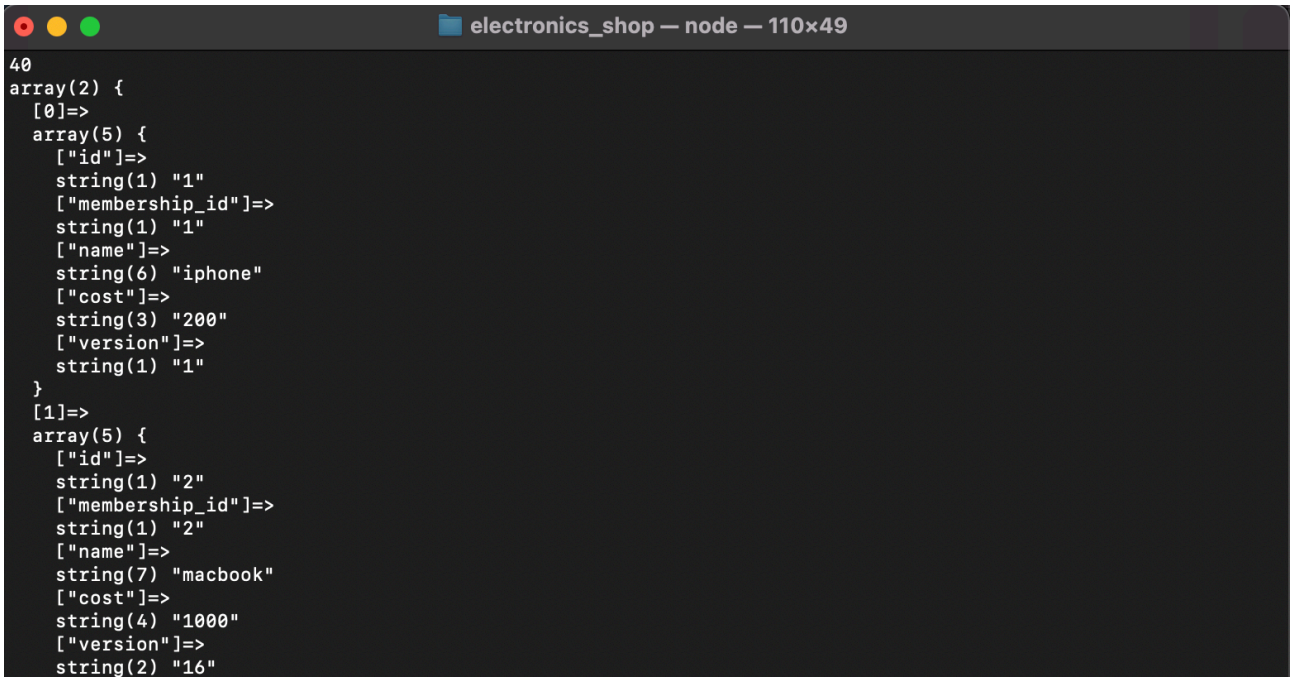
Реализуем микросервис для отправки (используя Node.js):

```
const express = require('express')
const app = express()
const port = 3000
var PROTO_PATH = __dirname + './test.proto'
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 test = grpc.loadPackageDefinition(packageDefinition).test
var client = new test.Test('localhost:50051',grpc.credentials.createInsecure())
app.get('/', (req, res) => {
    client.sendTest({associated_data: 40})
})
app.listen(port, () => {
6
console.log(`Microservice started`)
})
```

Реализуем дополнительный слой аналогично верхнему на Node.js, который будет вызывать наш php код.

```
var PROTO_PATH = __dirname + './test.proto'
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
})
const { exec } = require("child_process")
var test = grpc.loadPackageDefinition(packageDefinition).test
function sendTest(call, callback) {
    console.log(call.request.associated_date)
    exec("php read_purchases.php")
}
var server = new grpc.Server()
server.addService(test.Test.service, { sendTest: sendTest })
server.bindAsync('0.0.0.0:50051', grpc.ServerCredentials.createInsecure(), ()
=> {
    server.start()
})
```

Вывод аналогичен предыдущему, но с дополнительно передаваемым в поле associated_data числом.



Вывод

В данной лабораторной работе я познакомилась с практической реализацией принципа инверсии зависимостей, а также протоколами межсервисного взаимодействия.