

Министерство образования Республики Беларусь
Учреждение образования
“Брестский государственный технический университет”
Кафедра ИИТ

Отчёт
По лабораторной работе №7
По дисциплине ПриС

Выполнил

Студент группы ПО-3
3-го курса
Куликович И. Т.

Проверил

Лаврущик А. И.

Инфраструктура: хранение и доставка.

Межсервисное взаимодействие

ВАРИАНТ 17

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. Управление фитнес-центром.

Ход работы

Persistence-layer реализуем через реляционную SQL базу данных, используя встроенные средства Symfony. Настроим базу данных SQLite ввиду простоты развертывания. Для использования SQLite настроим файл .env добавив следующую строку:

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

Сформируем миграции для полученных ранее моделей данных.

Версия 1:

```
public function up(Schema $schema) : void
{
    $this->addSql('CREATE TABLE trainer (id INTEGER PRIMARY KEY
AUTOINCREMENT, name VARCHAR(100), years_experience INTEGER)');
    $this->addSql('CREATE TABLE visitor (id INTEGER PRIMARY KEY
AUTOINCREMENT)');
}
```

Версия 2:

```
public function up(Schema $schema) : void
{
    $this->addSql('ALTER TABLE visitor ADD COLUMN name VARCHAR(100)');
    $this->addSql('ALTER TABLE visitor ADD COLUMN weight INTEGER');
    $this->addSql('ALTER TABLE visitor ADD COLUMN height INTEGER');
}
```

Версия 3:

```
public function up(Schema $schema) : void
{
    $this->addSql('CREATE TABLE training_session (id INTEGER PRIMARY KEY
AUTOINCREMENT NOT NULL, trainer_id INTEGER DEFAULT NULL, date DATE NOT
NULL)');
    $this->addSql('CREATE INDEX IDX_D7A45DAFB08EDF6 ON training_session
(trainer_id)');
    $this->addSql('CREATE TABLE training_session_visitor (training_session_id
INTEGER NOT NULL, visitor_id INTEGER NOT NULL, PRIMARY
KEY(training_session_id, visitor_id))');
}
```

```

    $this->addSql('CREATE INDEX IDX_6D6ADAFEDB8156B9 ON
training_session_visitor (training_session_id)');
    $this->addSql('CREATE INDEX IDX_6D6ADAFE70BEE6D ON
training_session_visitor (visitor_id)');
    $this->addSql('CREATE TEMPORARY TABLE __temp__trainer AS SELECT id, name,
years_experience FROM trainer');
    $this->addSql('DROP TABLE trainer');
    $this->addSql('CREATE TABLE trainer (id INTEGER PRIMARY KEY AUTOINCREMENT
NOT NULL, name VARCHAR(100) NOT NULL, years_experience INTEGER NOT NULL)');
    $this->addSql('INSERT INTO trainer (id, name, years_experience) SELECT
id, name, years_experience FROM __temp__trainer');
    $this->addSql('DROP TABLE __temp__trainer');
    $this->addSql('CREATE TEMPORARY TABLE __temp__visitor AS SELECT id, name,
weight, height FROM visitor');
    $this->addSql('DROP TABLE visitor');
    $this->addSql('CREATE TABLE visitor (id INTEGER PRIMARY KEY AUTOINCREMENT
NOT NULL, name VARCHAR(100) NOT NULL, weight INTEGER NOT NULL, height INTEGER
NOT NULL)');
    $this->addSql('INSERT INTO visitor (id, name, weight, height) SELECT id,
name, weight, height FROM __temp__visitor');
    $this->addSql('DROP TABLE __temp__visitor');
}

```

Версия 4:

```

public function up(Schema $schema) : void
{
    $this->addSql('CREATE TABLE membership (id INTEGER PRIMARY KEY
AUTOINCREMENT NOT NULL, duration INTEGER NOT NULL, cost INTEGER NOT NULL)');
    $this->addSql('DROP INDEX IDX_D7A45DAFB08EDF6');
    $this->addSql('CREATE TEMPORARY TABLE __temp__training_session AS SELECT
id, trainer_id, date FROM training_session');
    $this->addSql('DROP TABLE training_session');
    $this->addSql('CREATE TABLE training_session (id INTEGER PRIMARY KEY
AUTOINCREMENT NOT NULL, trainer_id INTEGER DEFAULT NULL, date DATE NOT NULL,
CONSTRAINT FK_D7A45DAFB08EDF6 FOREIGN KEY (trainer_id) REFERENCES trainer
(id) NOT DEFERRABLE INITIALLY IMMEDIATE)');
    $this->addSql('INSERT INTO training_session (id, trainer_id, date) SELECT
id, trainer_id, date FROM __temp__training_session');
    $this->addSql('DROP TABLE __temp__training_session');
    $this->addSql('CREATE INDEX IDX_D7A45DAFB08EDF6 ON training_session
(trainer_id)');
    $this->addSql('DROP INDEX IDX_6D6ADAFE70BEE6D');
    $this->addSql('DROP INDEX IDX_6D6ADAFEDB8156B9');
    $this->addSql('CREATE TEMPORARY TABLE __temp__training_session_visitor AS
SELECT training_session_id, visitor_id FROM training_session_visitor');
    $this->addSql('DROP TABLE training_session_visitor');
    $this->addSql('CREATE TABLE training_session_visitor (training_session_id
INTEGER NOT NULL, visitor_id INTEGER NOT NULL, PRIMARY
KEY(training_session_id, visitor_id), CONSTRAINT FK_6D6ADAFEDB8156B9 FOREIGN
KEY (training_session_id) REFERENCES training_session (id) ON DELETE CASCADE
NOT DEFERRABLE INITIALLY IMMEDIATE, CONSTRAINT FK_6D6ADAFE70BEE6D FOREIGN KEY
(visitor_id) REFERENCES visitor (id) ON DELETE CASCADE NOT DEFERRABLE
INITIALLY IMMEDIATE)');
    $this->addSql('INSERT INTO training_session_visitor (training_session_id,
visitor_id) SELECT training_session_id, visitor_id FROM
__temp__training_session_visitor');
    $this->addSql('DROP TABLE __temp__training_session_visitor');
    $this->addSql('CREATE INDEX IDX_6D6ADAFE70BEE6D ON
training_session_visitor (visitor_id)');
}

```

```

    $this->addSql('CREATE INDEX IDX_6D6ADAFEDB8156B9 ON
training_session_visitor (training_session_id)');
    $this->addSql('CREATE TEMPORARY TABLE __temp__visitor AS SELECT id, name,
weight, height FROM visitor');
    $this->addSql('DROP TABLE visitor');
    $this->addSql('CREATE TABLE visitor (id INTEGER PRIMARY KEY AUTOINCREMENT
NOT NULL, membership_id INTEGER DEFAULT NULL, name VARCHAR(100) NOT NULL
COLLATE BINARY, weight INTEGER NOT NULL, height INTEGER NOT NULL, CONSTRAINT
FK_CAE5E19F1FB354CD FOREIGN KEY (membership_id) REFERENCES membership (id)
NOT DEFERRABLE INITIALLY IMMEDIATE)');
    $this->addSql('INSERT INTO visitor (id, name, weight, height) SELECT id,
name, weight, height FROM __temp__visitor');
    $this->addSql('DROP TABLE __temp__visitor');
    $this->addSql('CREATE INDEX IDX_CAE5E19F1FB354CD ON visitor
(membership_id)');
}

```

Далее создадим репозитории для взаимодействия с данными.

MembershipRepository

```

class MembershipRepository extends ServiceEntityRepository
{
    public function __construct(ManagerRegistry $registry)
    {
        parent::__construct($registry, Membership::class);
    }

    public function addMembership(int $cost, int $duration): Membership
    {
        $entityManager = $this->getEntityManager();
        $membership = new Membership();
        $membership->setCost($cost);
        $membership->setDuration($duration);

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

        return $membership;
    }

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

```

TrainerRepository

```

class TrainerRepository extends ServiceEntityRepository
{
    public function __construct(ManagerRegistry $registry)
    {
        parent::__construct($registry, Trainer::class);
    }

    public function addTrainer(string $name, int $years_experience): Trainer
    {
        $entityManager = $this->getEntityManager();
    }
}

```

```

        $trainer = new Trainer();
        $trainer->setName($name);
        $trainer->setYearsExperience($years_experience);

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

        return $trainer;
    }

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

```

TrainingSessionRepository

```

class TrainingSessionRepository extends ServiceEntityRepository
{
    public function __construct(ManagerRegistry $registry)
    {
        parent::__construct($registry, TrainingSession::class);
    }

    public function addTrainingSession(\DateTimeInterface $date,
\App\Entity\Trainer $trainer, \App\Entity\Visitor $visitor)
    {
        $entityManager = $this->getEntityManager();
        $training_session = new TrainingSession();
        $training_session->setDate($date);
        $training_session->setTrainer($trainer);
        $training_session->addVisitor($visitor);

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

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

```

VisitorRepository

```

class VisitorRepository extends ServiceEntityRepository
{
    public function __construct(ManagerRegistry $registry)
    {
        parent::__construct($registry, Visitor::class);
    }

    public function addVisitor(string $name, int $weight, int $height):
Visitor
    {
        $entityManager = $this->getEntityManager();
    }
}

```

```

        $visitor = new Visitor();
        $visitor->setName($name);
        $visitor->setWeight($weight);
        $visitor->setHeight($height);

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

        return $visitor;
    }

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

```

Имея репозитории можно подключать интерфейс (консольный, rest). Например сделаем простой Rest API Controller.

```

<?php
namespace App\Controller;

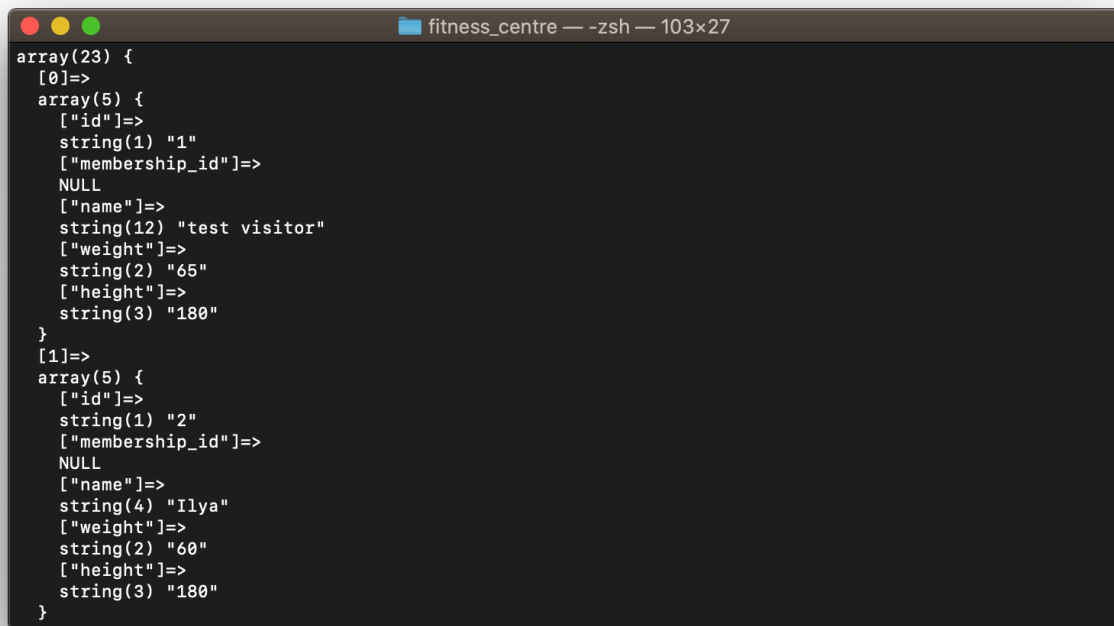
use App\Entity\Visitor;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\Routing\Annotation\Route;
use Symfony\Component\Serializer\Encoder\JsonEncoder;
use Symfony\Component\Serializer\Serializer;
use Symfony\Component\Serializer\Normalizer\ObjectNormalizer;
use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

class VisitorApiController extends AbstractController
{
    /**
     * @Route("/api/visitors", name="api_visitors")
     */
    public function getAllVisitors()
    {
        $entityManager = $this->getDoctrine()->getManager();
        $visitors = $entityManager->getRepository(Visitor::class)->findAll();
        $encoders = [new JsonEncoder()];
        $normalizers = [new ObjectNormalizer()];
        $serializer = new Serializer($normalizers, $encoders);
        $result_json = [];
        foreach ($visitors as $visitor)
        {
            $visitor_json = [];
            $article_json['id'] = $visitor->getId();
            $article_json['name'] = $visitor->getName();
            $article_json['weight'] = $visitor->getWeight();
            $article_json['height'] = $visitor->getHeight();
            array_push($result_json, $visitor_json);
        }
        $response = new Response($serializer->serialize($result_json,
'json'));
        $response->headers->set('Content-Type', 'application/json');
        return $response;
    }
}

```

Доступ из консоли реализуем следующим скриптом:

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

A terminal window titled 'fitness_centre — zsh — 103x27' displays the output of a PHP script. The output is a JSON array with two elements. Each element is an object representing a database record with fields: id, membership_id, name, weight, and height. The first record has id '1', membership_id NULL, name 'test visitor', weight '65', and height '180'. The second record has id '2', membership_id NULL, name 'Ilya', weight '60', and height '180'.

```
array(23) {
  [0]=>
  array(5) {
    ["id"]=>
    string(1) "1"
    ["membership_id"]=>
    NULL
    ["name"]=>
    string(12) "test visitor"
    ["weight"]=>
    string(2) "65"
    ["height"]=>
    string(3) "180"
  }
  [1]=>
  array(5) {
    ["id"]=>
    string(1) "2"
    ["membership_id"]=>
    NULL
    ["name"]=>
    string(4) "Ilya"
    ["weight"]=>
    string(2) "60"
    ["height"]=>
    string(3) "180"
  }
}
```

Далее реализовываем микросервис с отправкой protobuf данных. Опишем данные:

```
syntax = "proto3"
```

```
message test {
  int32 associated_value = 0;
}
```

Реализуем микросервис для отправки на 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, () => {
```



```

    console.log(`Microservice started`)
  })

```

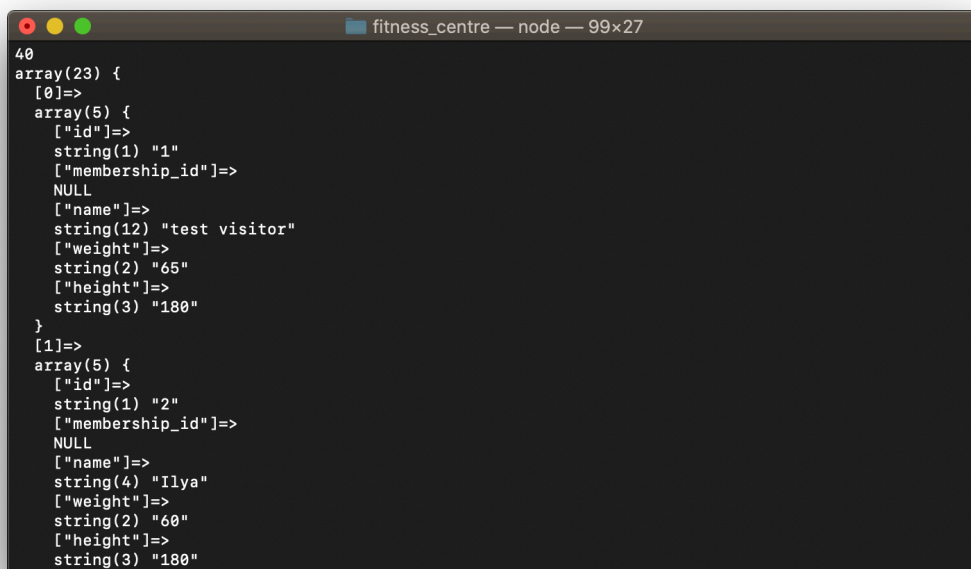
Ввиду того, что основное приложение написано на php и gRPC на данный момент не поддерживает прием данных, реализуем дополнительный слой аналогично верхнему на Node.js, который уже будет вызывать наш php код.

```

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_visitors.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()
})

```

Демонстрация работы (видим вывод аналогичный предыдущему, но с дополнительно переданным в поле `associated_data` числом.



```

40
array(23) {
  [0]=>
  array(5) {
    ["id"]=>
    string(1) "1"
    ["membership_id"]=>
    NULL
    ["name"]=>
    string(12) "test visitor"
    ["weight"]=>
    string(2) "65"
    ["height"]=>
    string(3) "180"
  }
  [1]=>
  array(5) {
    ["id"]=>
    string(1) "2"
    ["membership_id"]=>
    NULL
    ["name"]=>
    string(4) "Ilya"
    ["weight"]=>
    string(2) "60"
    ["height"]=>
    string(3) "180"
  }
}

```

Вывод

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