МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ

УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«СЕВАСТОПОЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Институт Информационных технологий и управления в технических системах

Кафедра Информационные системы

|  |  |  |  |
| --- | --- | --- | --- |
| № | Дата поступления на кафедру | Подпись отв. за регистрацию | Подпись преподавателя |
|  |  |  |  |
|  |  |  |  |

**ОТЧЕТ**

о производственной (по получению профессиональных умений и опыта профессиональной деятельности) практике

в \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ООО «Центр разработки»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(наименование организации)

Выполнил Черняев Никита Георгиевич

(Фамилия И.О. обучающегося)

\_\_\_\_\_\_\_\_\_\_\_ИС/б-17-2-о\_\_\_\_\_\_\_\_\_\_\_\_

(шифр группы)

Направление / специальность 09.03.02

Информационные системы и технологии

(код, наименование)

Руководитель практики от Университета

\_\_\_\_старший преподаватель\_\_\_\_\_\_\_\_\_

(должность)

\_\_\_\_\_\_\_\_\_\_Заикина Е.Н.\_\_\_\_\_\_\_\_\_\_\_\_

(Фамилия И.О. руководителя)

Севастополь

2020 г.

Оглавление

[1. Индивидуальное задание на практику 3](#_Toc46307685)

[Техническое задание 3](#_Toc46307686)

[Цех по ремонту звездолетов 3](#_Toc46307687)

[Требования к приложению 4](#_Toc46307688)

[Ограничения 4](#_Toc46307689)

[Примеры 5](#_Toc46307690)

[Требования к реализации 5](#_Toc46307691)

[2. Отзыв руководителя практики от профильной организации 6](#_Toc46307692)

[3. Введение 6](#_Toc46307693)

[4. Основная часть 7](#_Toc46307694)

[5. Заключение 15](#_Toc46307695)

[Список использованных источников 16](#_Toc46307696)

1. Индивидуальное задание на практику

## Техническое задание

## Цех по ремонту звездолетов

В далекой галактике X был построен космический цех по обслуживанию звездолетов. В этом цехе с площадкой на N мест для звездолетов размещается робот. Робот производит обслуживание звездолетов размещенных в цехе. При этом он их обслуживает в том порядке, в каком они занимали свои места на площадке. Обслуживание каждого звездолета занимает разное количество времени. Звездолеты, которым требуется обслуживание, прибывают в разное время. Если звездолет прилетел, робот занят, а в ангаре нет мест, он улетает без обслуживания. Нужно реализовать веб-приложение, которое будет обрабатывать следующие http запросы:

POST /numberOfPlaces В теле запроса приходит кол-во мест в цеху - JSON вида {numberOfPlaces: 'N'}, где 1 <= N <= 10^5 Запрос инициализирует цех на N мест и активирует робота. POST /ship В теле второго запроса приходят данные о корабле(время прибытия, время на тех. обслуживание) - JSON вида {timeOfArrival: 'X', handleTime: 'Y'}, где X, Y целые числа GET /next Ответом на этот запрос должно быть время начала обслуживания следующего звездолета - HTTP ответ с кодом 200 и телом из JSON вида {response: 'T'}, где Т целое число. Приложение должно отвечать в том порядке, в каком звездолеты прибывали на обслуживание после инициализации цеха.

Приложение должно фиксировать время, когда начнется обслуживание каждого пришедшего корабля, либо -1, если обслужить корабль не удастся.

### Требования к приложению

После запроса на инициализацию цеха POST /numberOfPlaces и S запросов POST /ship на обслуживание корабля, каждый i-тый запрос GET /next должен вернуть ответ HTTP 200 {response: 'T'} время начала обслуживания Si корабля, при условии i <= s Если приходит запрос GET /next, а при этом звездолетов еще не было вообще или все они уже были запрошены предыдущими запросами GET /next, то отвечаем HTTP ответом с кодом 200 и пустым телом ответа На запросы по URL не описанным в API приложение должно отвечать HTTP ответом с кодом 404 При не соответствии формата тела запроса (например не корректный json) HTTP ответом с кодом 400 На все корректные POST запросы приложение должно отвечать HTTP ответом с кодом 200 и пустым телом ответа

### Ограничения

1 <= N <= 10^5

0 <= S <= 10^5

0 <= X <= 10^6

0 <= Y <= 10^3

Гарантируется, что время прилета следующего корабля больше либо равно предыдущему.

Описание работы цеха Если корабль прибывает и цех пуст, то робот приступает к работе немедленно. Если в цехе уже есть звездолеты, но при этом остается место, то робот приступит к обслуживанию данного звездолета, когда закончит со всеми предыдущими. Если робот занят и мест в цехе нет, то обслужить корабль не удастся.

### Примеры

POST /numberOfPlaces {numberOfPlaces: '1'} //запрос на инициализацию цеха на 1 место POST /ship {timeOfArrival: '0', handleTime: '1'} //запрос на обслуживание корабля GET /next - Ответ HTTP 200 {response: '0'} //тело ответа содержит 0, т.к. на обслуживании один единственный корабль, и т.к. когда он прилетел цех был пуст, время начала его обслуживания 0.

POST /numberOfPlaces {numberOfPlaces: '1'} //запрос на инициализацию цеха на 1 место POST /ship {timeOfArrival: '0', handleTime: '1'} //запрос на обслуживание корабля POST /ship {timeOfArrival: '0', handleTime: '1'} //запрос на обслуживание корабля GET /next - Ответ HTTP 200 {response: '0'} //тело ответа содержит 0, т.к. первый корабль который прилетел, прилетел в пустой цех и робот сразу приступил к его обслуживанию GET /next - Ответ HTTP 200 {response: '-1'} // тело ответа содержит -1, т.к. когда корабль прилетел, единственное место в цехе было занято, а робот еще не закончил обслуживание предыдущего корабля

POST /numberOfPlaces {numberOfPlaces: '1'} POST /ship {timeOfArrival: '0', handleTime: '1'} POST /ship {timeOfArrival: '1', handleTime: '1'} GET /next - Ответ HTTP 200 {response: '0'} GET /next - Ответ HTTP 200 {response: '1'}

### Требования к реализации

Язык программирования Scala версии 2.12.8 Scala framework для реализации сервера на выбор play framework, akka-http, http4s последней стабильной версии Проект должен собираться и запускаться с помощью scala SBT Сервер должен обрабатывать запросы на localhost:8888 БД не использовать, все операции должны храниться и производиться в памяти с использованием подходящих структур данных

1. Отзыв руководителя практики от профильной организации

За время прохождения практики Черняев Н.Г. проявил себя как грамотный специалист. Студент легко обучаем и открыт для изучения новых технологий, а также способен оценить сложившуюся ситуацию и самостоятельно принять все необходимые меры для ее решения.

Студент Черняев Н.Г. вежлив, целеустремлен. Зарекомендовал себя грамотным и ответственным работником.

1. Введение

В ходе прохождения практики были изучены и закреплены знания языка программирования Scala и фрейморка Play.

Play - это высокопроизводительный фреймворк для веб-приложений на Java и Scala, который объединяет компоненты и API, необходимые для разработки современных веб-приложений.

Play основан на облегченной архитектуре без поддержки состояния, имеет веб-интерфейс и обеспечивает предсказуемое и минимальное потребление ресурсов (ЦП, память, потоки) для масштабируемых приложений благодаря своей реактивной модели на основе Akka Streams.

Первой задачей являлось создание REST API. Для этого была взята готовая MVC модель с официального github’а разработчиков Play framework.

1. Основная часть

По условию запросы должны были обрабатываться на localhost:8888.

Код выбора порта из файла application.conf:

play.server.provider = play.core.server.AkkaHttpServerProvider

play {

  server {

    dir = ${?user.dir}

    # HTTP configuration

    http {

      # The HTTP port of the server. Use a value of "disabled" if the server

      # shouldn't bind an HTTP port.

      port = 8888

      port = ${?PLAY\_HTTP\_PORT}

      port = ${?http.port}

      # The interface address to bind to.

      address = "0.0.0.0"

      address = ${?PLAY\_HTTP\_ADDRESS}

      address = ${?http.address}

      idleTimeout = 75 seconds

    }

  }

}

Следующим шагом было создание модели, которая бы и решала задачу. Для создания модели было создано три класса, описывающих цех, корабли и количество мест в цеху соответственно.

Код файла Manufactory.scala:

package models

import scala.collection.mutable

import scala.collection.mutable.Queue

class Manufactory(){

  var timeRobotGoesToNextShip: Int = 0

  var places: Queue[Place] = new Queue[Place](100000)

  var ships: Queue[Ship] = new Queue[Ship](100000)

  def Solve(): Int = {

    var firstEl = ships.dequeue

    if (timeRobotGoesToNextShip == 0){

      timeRobotGoesToNextShip = firstEl.timeOfArrival

    }

    for(place <- places){

      if (place.timeShipLeavePlace <= firstEl.timeOfArrival){

        timeRobotGoesToNextShip += firstEl.handleTime

        place.timeShipLeavePlace = timeRobotGoesToNextShip

        return timeRobotGoesToNextShip - firstEl.handleTime

      }

    }

    return -1

  }

  def AddShip(ship: Ship){

    ships.enqueue(ship);

  }

  def AddPlace(place: Place){

    places.enqueue(place);

  }

}

Код файла Ship.scala:

package models

class Ship(var timeOfArrival: Int, var handleTime: Int)

Код файла Place.scala:

package models

class Place(var timeShipLeavePlace: Int = 0)

Затем нужно было создать 3 http запроса, но для начала зафиксировать к ним маршруты.

Код создания маршрутов из файла routes:

# Routes

POST /numberOfPlaces controllers.ManufactoryController.numberOfPlaces()

POST /ship controllers.ManufactoryController.ship()

GET /next controllers.ManufactoryController.next()

А после этого создать контроллер, где и будет описана реализация данных запросов.

Код реализации запросов из файла ManufactoryController.scala:

  def numberOfPlaces = Action(validateJson[NumberOfPlacesRequest]) { implicit request =>

    val placesRequest = request.body

    manufactory = new Manufactory()

    for (i <- 0 until placesRequest.numberOfPlaces){

      manufactory.AddPlace(new Place)

    }

    Ok(Results.EmptyContent())

  }

  def ship() = Action(validateJson[ShipRequest]) { implicit request =>

    val shipRequest = request.body

    var ship = new Ship(shipRequest.timeOfArrival, shipRequest.handleTime)

    manufactory.AddShip(ship)

    Ok(Results.EmptyContent())

  }

  def next() = Action { implicit request =>

    if (manufactory.ships.isEmpty) {

      Ok(Results.EmptyContent())

    }

    else{

      val answer = manufactory.Solve()

      val response = NextResponse(answer)

      Ok(Json.toJson(response))

    }

  }

}

Библиотека play.api.libs.json использовалась для чтения и записи JSON данных. Это реализовано в «reader’е» и «writer'е».

Код реализации «reader’а» и «writer'а» из файла ManufactoryController.scala:

  case class ShipRequest(timeOfArrival: Int, handleTime: Int)

  case class NumberOfPlacesRequest(numberOfPlaces: Int)

  case class NextResponse(response: Int)

  implicit val writer: Writes[NextResponse] = (

    (JsPath \ "response").write[Int].contramap(\_.response)

  )

  implicit val reader1: Reads[NumberOfPlacesRequest] = (

    (JsPath \ "numberOfPlaces").read[Int](min(1).keepAnd(max(100000))).map(NumberOfPlacesRequest(\_))

  )

  implicit val reader2: Reads[ShipRequest] = (

    (JsPath \ "timeOfArrival").read[Int](min(0).keepAnd(max(1000000)))

    .and((JsPath \ "handleTime").read[Int](min(0).keepAnd(max(1000))))

  )(ShipRequest.apply \_)

Также была добавлена валидация и обработчик ошибок.

Код реализации валидации из файла ManufactoryController.scala:

  def validateJson[A: Reads] = parse.json.validate(

  \_.validate[A].asEither.left.map(e => BadRequest(JsError.toJson(e)))

  )

Код реализации обработки ошибок из файла ErrorHandler.scala:

import javax.inject.{Inject, Provider}

import play.api.\_

import play.api.http.DefaultHttpErrorHandler

import play.api.http.Status.\_

import play.api.libs.json.Json

import play.api.mvc.Results.\_

import play.api.mvc.\_

import play.api.routing.Router

import play.core.SourceMapper

import scala.concurrent.\_

class ErrorHandler(environment: Environment,

                   configuration: Configuration,

                   sourceMapper: Option[SourceMapper] = None,

                   optionRouter: => Option[Router] = None)

    extends DefaultHttpErrorHandler(environment,

                                    configuration,

                                    sourceMapper,

                                    optionRouter) {

  private val logger =

    org.slf4j.LoggerFactory.getLogger("application.ErrorHandler")

  // This maps through Guice so that the above constructor can call methods.

  @Inject

  def this(environment: Environment,

           configuration: Configuration,

           sourceMapper: OptionalSourceMapper,

           router: Provider[Router]) = {

    this(environment,

         configuration,

         sourceMapper.sourceMapper,

         Some(router.get))

  }

  override def onClientError(request: RequestHeader,

                             statusCode: Int,

                             message: String): Future[Result] = {

    logger.debug(

      s"onClientError: statusCode = $statusCode, uri = ${request.uri}, message = $message")

    Future.successful {

      val result = statusCode match {

        case BAD\_REQUEST =>

          Results.BadRequest(message)

        case FORBIDDEN =>

          Results.Forbidden(message)

        case NOT\_FOUND =>

          Results.NotFound(message)

        case clientError if statusCode >= 400 && statusCode < 500 =>

          Results.Status(statusCode)

        case nonClientError =>

          val msg =

            s"onClientError invoked with non client error status code $statusCode: $message"

          throw new IllegalArgumentException(msg)

      }

      result

    }

  }

  override protected def onDevServerError(

      request: RequestHeader,

      exception: UsefulException): Future[Result] = {

    Future.successful(

      InternalServerError(Json.obj("exception" -> exception.toString)))

  }

  override protected def onProdServerError(

      request: RequestHeader,

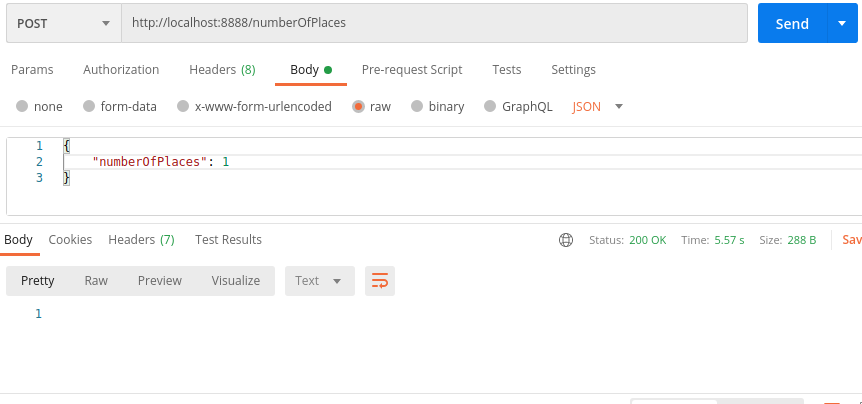
      exception: UsefulException): Future[Result] = {

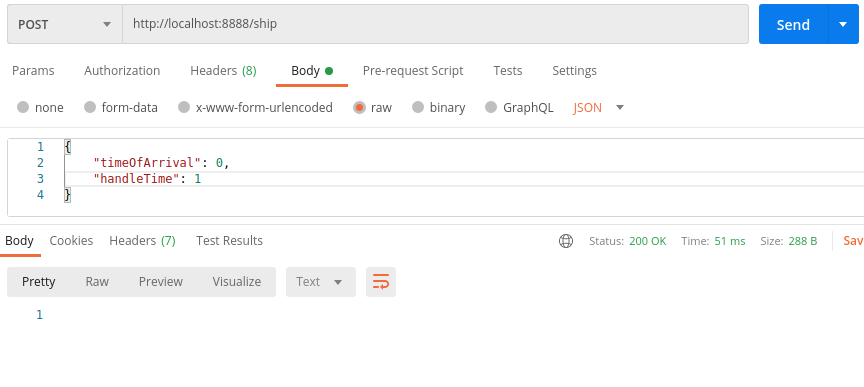
    Future.successful(InternalServerError)

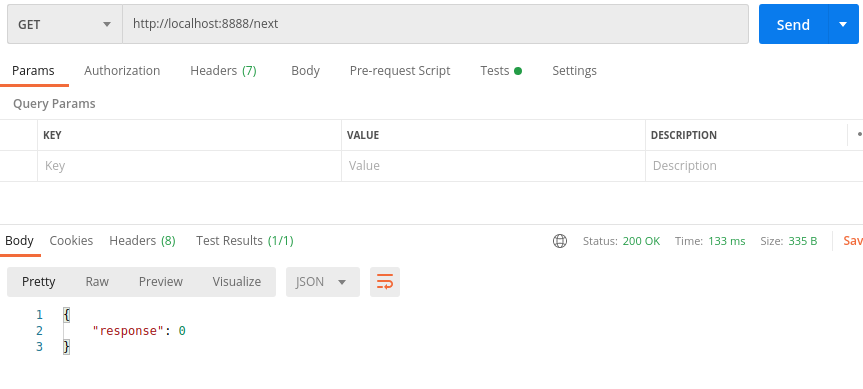
  }

}

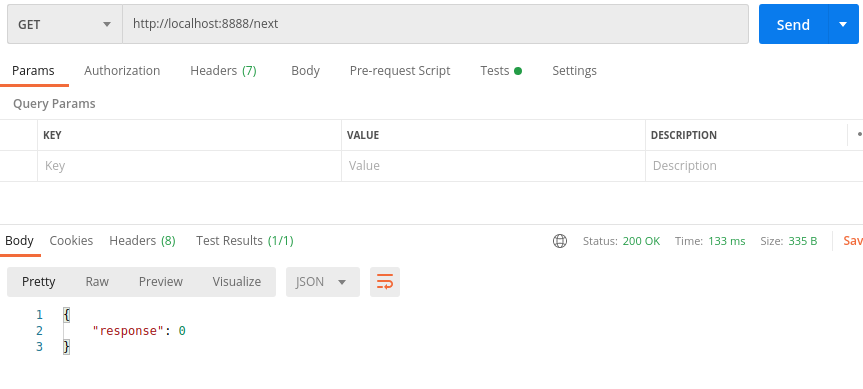
Тестирование приложения.

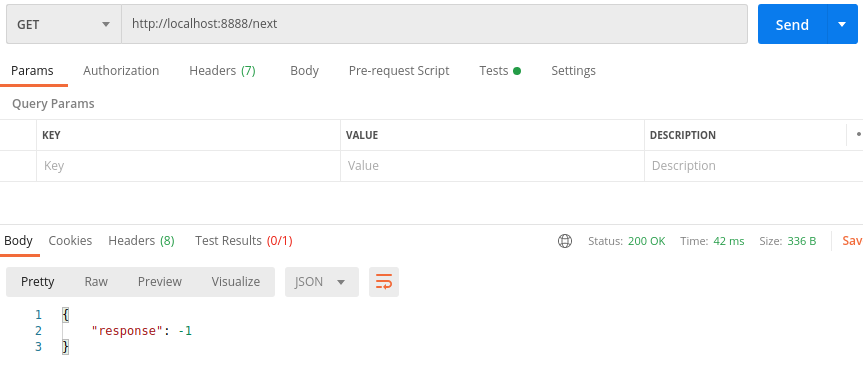






Теперь протестируем, отправив два запроса POST /ship {timeOfArrival: '0', handleTime: '1'}





1. Заключение

В ходе прохождения практики были изучены основные роли процесса разработки, стадии разработки, а также следующие технологии и инструменты: разработка серверных приложений на языке Scala, распределённая система управления версиями Git, Ubuntu - операционная система, основанная на Debian GNU/Linux, система сборки sbt, клиет-серверное взаимодействие с использованием HTTP, веб-фреймворк Play.

При помощи технологий, описанных выше были получены навыки написания REST API, создания http запросов и их валидации, создания моделей для приложения.

Также очень полезны оказались знания, полученные в ходе изучения дисциплин «WEB Технологии», «Платформа Java», «Тестирование ПО».

Код написанные в ходе практики расположен в репозитории по ссылке: <https://github.com/Pinkusgregory/testScalaApp>

# Список использованных источников

* <https://docs.scala-lang.org/>
* <https://git-scm.com/>
* <https://ubuntu.com/>
* <https://www.scala-sbt.org/>
* <https://www.playframework.com/>