Project 0: Functional Programming / The Actor Model

Author: Dubina Valeria FAF-203

Examined by: Alexandru Osadcenco

Author: Dubina Valeria FAF-203
Examined by: **Alexandru Osadcenco**
P0W1 – Welcome…
  Minimal Tasks:
  Main Tasks:
  Bonus Tasks:
P0W2 – …to the rice fields
  Minimal Tasks:
  Main Tasks:
  Bonus Tasks:
P0W3 – An Actor is Born
  Minimal Tasks:
  Main Tasks:
P0W4 – The Actor is dead… Long live the Actor
  Minimal Tasks:
  Main Tasks:
P0W5 – May the Web be with you
  Minimal Tasks:
  Main Tasks:
Conclusions:

P0W1 – Welcome…

Minimal Tasks:

1. Follow an installation guide to install the language/development environment of your choice.

Chosen language for this project is SCALA and as the development environment IntelliJ.

1. Write a script that would print the message “Hello PTR” on the screen.

```
object Greetings {
  def printHello(): Unit = {
    println("Hello, PTR!")
  }
}
```

Main Tasks:

1. Initialize a VCS repository for your project. Push your project to a remote repo.

HERE IT IS!

Bonus Tasks:

1. Write a comprehensive readme for your repository.

Access the project README.md

1. Create a unit test for your project. Execute it.

```
import org.scalatest.funsuite.AnyFunSuite

class HelloWorldTest extends AnyFunSuite {
  test("Hello, PTR! should be printed") {
    val consoleOutput = new java.io.ByteArrayOutputStream()
    Console.withOut(consoleOutput) {
      Greetings.printHello()
    }

    assert(consoleOutput.toString.trim === "Hello, PTR!")
  }
}
```

P0W2 – …to the rice fields

This week contains practical exercises to get familiar with the programming language of our choice. Each task is implemented in Scala with test coverage.

Minimal Tasks:

Write a function that determines whether an input integer is a prime

```
def isPrime(n: Int): Boolean = {
    if (n <= 1) return false
    for (i <- 2 to Math.sqrt(n).toInt) {
      if (n % i == 0) return false
    }
    true
  }

 test("13 is prime") {
    assert(NumberUtils.isPrime(13))
  }

  test("14 is not prime") {
    assert(!NumberUtils.isPrime(14))
  }
```

Write a function to calculate the area of a cylinder, given its height and radius.

```
def getArea(height: Double, radius: Double): Double = {
    val area = 2 * math.Pi * radius * height + 2 * math.Pi * radius * radius

    BigDecimal(area).setScale(4, RoundingMode.HALF_UP).toDouble
  }

test("get area of a cylinder") {
    val height = 3
    val radius = 4
    val expectedResult = 175.9292

    val actual = Cylinder.getArea(height, radius)

    assert(actual == expectedResult)
  }
```

Write a function to reverse a list.

```
def reverseList[dataType](list: List[dataType]): List[dataType] = {
    list.reverse
  }

test("List of int is reversed") {
    val list = List(1, 2, 4, 8, 4)
    val expected = List(4, 8, 4, 2, 1)

    val actual = ListUtils.reverseList(list)

    assert(expected.size == actual.size, "Lists have different sizes")
    assert(expected.zip(actual).forall(t => t._1 == t._2), "Lists have different elements")
  }
```

Write a function to calculate the sum of unique elements in a list.

```
def sumOfUniqueElements(list: List[Int]): Int = {
    val uniqueElements = list.toSet
    uniqueElements.sum
  }

test("Sum of unique elements") {
    val list = List(1, 2, 4, 8, 4, 2)
    val expectedSum = 15

    val actualSum = ListUtils.sumOfUniqueElements(list)

    assert(actualSum == expectedSum, "Unique elements sum is different")
 }
```

Write a function that extracts a given number of randomly selected elements from a list.

```
def extractRandomElements[dataType](list: List[dataType], numberOfElements: Int): List[dataType] = {
    if (numberOfElements > list.size) {
      throw new IllegalArgumentException("The list doesn't have enough elements")
    }

    val random = new Random
    (0 until numberOfElements).map(i => list(random.nextInt(list.length))).toList
  }

test("Extract random 3 elements from a list") {
    val list = List(1, 2, 4, 8, 4)
    val numberOfElements = 3

    val result = ListUtils.extractRandomElements(list, numberOfElements)

    assert(result.size == 3, "Lists have different sizes")
    assert(result.forall(list.contains), "Lists have different elements")
  }

  test("Extract more elements from a list than list size should throw exception") {
    val list = List(1, 2, 4, 8, 4)
    val numberOfElements = 12
    intercept[IllegalArgumentException] {
      ListUtils.extractRandomElements(list, numberOfElements)
    }
  }
```

Write a function that, given a dictionary, would translate a sentence. Words not found in the dictionary need not be translated

```
def translate(dict: Map[String, String], sentence: String): String = {
    val words = sentence.split(" ")
    val translatedWords = words.map(word => dict.getOrElse(word, word))
    translatedWords.mkString(" ")
 }

test("Dictionary is applied on sentence") {
    val dictionary = Map(
      "mama" -> "mother",
      "papa" -> "father",
    )
    val originalText = "mama is with papa"
    val expectedText = "mother is with father"

    val actualText = WordsUtils.translate(dictionary, originalText)

    assert(expectedText == actualText, "Expected and actual texts are different")
  }
```

Write a function that receives as input three digits and arranges them in an
order that would create the smallest possible number. Numbers cannot start with a 0.

```
def createSmallestNumber(a: Int, b: Int, c: Int): Int = {
    var digits = List(a, b, c).sorted

    if (digits.head == 0) {
      digits = digits.updated(0, digits(1)).updated(1, digits.head) // swap
    }

    100 * digits.head + 10 * digits(1) + digits(2)
  }

  test("smallest possible number") {
    val expected: Int = 123
    val actual = NumberUtils.createSmallestNumber(3, 1, 2)
    assert(expected == actual)
  }

  test("smallest possible number with 0") {
    val expected: Int = 109
    val actual = NumberUtils.createSmallestNumber(0, 1, 9)
    assert(expected == actual)
  }
```

Write a function that would rotate a list n places to the left.

```
def rotateToLeft[dataType](list: List[dataType], numberOfPlaces: Int = 1): List[dataType] = {
    val rotated = numberOfPlaces % list.length
    list.drop(rotated) ++ list.take(rotated)
  }

test("Rotate a list 3 places to the left") {
    val list = List(1 , 2, 4, 8, 4)
    val numberOfPlaces = 3
    val expected = List(8, 4, 1, 2, 4)

    val actual = ListUtils.rotateToLeft(list, numberOfPlaces)

    assert(expected.size == actual.size, "Lists have different sizes")
    assert(expected.zip(actual).forall(t => t._1 == t._2), "Lists have different elements")
  }
```

Write a function that lists all tuples a, b, c such that a

```scala
def listRightAngleTriangles(): List[(Int, Int, Int)] = {
    for {
      a <- 1 to 20
      b <- 1 to 20
      c <- 1 to 70
      if a * a + b * b == c * c
    } yield (a, b, c)
  }.toList

test("Get tuples where a^2 + b^2 = c^2") {
    val rightAngle = (3, 4, 5)
    val tuplesList = NumberUtils.listRightAngleTriangles();

    assert(tuplesList.contains(rightAngle))
 }
```

Main Tasks:

Write a function that eliminates consecutive duplicates in a list.

```scala
def removeConsecutiveDuplicates[A](list: List[A]): List[A] = {
    list.foldRight(List[A]()) { (currentElement, accumulatedFromPreIteration) =>
      if (accumulatedFromPreIteration.isEmpty || accumulatedFromPreIteration.head != currentElement) {
        currentElement :: accumulatedFromPreIteration
      }
      else {
        accumulatedFromPreIteration
      }
    }
  }

test("Remove consecutive duplicated elements from list") {
    val list = List(1 , 2, 2, 2, 4, 8, 4)
    val expected = List(1, 2, 4, 8, 4)

    val actual = ListUtils.removeConsecutiveDuplicates(list)

    assert(expected.size == actual.size, "Lists have different sizes")
    assert(expected.zip(actual).forall(t => t._1 == t._2), "Lists have different elements")
  }
```

Write a function that, given an array of strings, will return the words that can be typed using only one row of letters on an English keyboard layout.

```scala
def filterOneRowWords(words: List[String]): List[String] = {
    val row1 = Set('q', 'w', 'e', 'r', 't', 'y', 'u', 'i', 'o', 'p')
    val row2 = Set('a', 's', 'd', 'f', 'g', 'h', 'j', 'k', 'l')
    val row3 = Set('z', 'x', 'c', 'v', 'b', 'n', 'm')

    words.filter { word =>
      val lowerCaseWord = word.toLowerCase()

      lowerCaseWord.forall(c => row1.contains(c)) ||
        lowerCaseWord.forall(c => row2.contains(c)) ||
        lowerCaseWord.forall(c => row3.contains(c))
    }
  }

test("Only strings typed using one row of the letters on an English keyboard") {
    val list = List("Hello", "Alaska", "Dad", "Peace")
    val expected = List("Alaska", "Dad")

    val actual = WordsUtils.filterOneRowWords(list)
    println(actual)

    assert(expected.size == actual.size, "Lists have different sizes")
    assert(expected.zip(actual).forall(t => t._1 == t._2), "Lists have different elements")
  }
```

Create a pair of functions to encode and decode strings using the Caesar cipher.

```
object CaesarCipher {
  def encode(s: String, shift: Int): String = {
    s.map(c => (c + shift).toChar).mkString
  }

  def decode(s: String, shift: Int): String = {
    s.map(c => (c - shift).toChar).mkString
  }
}

test("Caesar encryption and decryption works fine") {
    val shift = 3
    val input = "lorem"
    val encoded = "oruhp"

    val actualEncode = CaesarCipher.encode(input, shift)
    val actualDecode = CaesarCipher.decode(encoded, shift)

    assert(input == actualDecode)
    assert(encoded == actualEncode)
  }
```

White a function that, given a string of digits from 2 to 9, would return all
possible letter combinations that the number could represent (think phones with buttons).

```
def phoneLettersCombinations(digits: String): List[String] = {
    val digitMap = Map(
      '2' -> "abc",
      '3' -> "def",
      '4' -> "ghi",
      '5' -> "jkl",
      '6' -> "mno",
      '7' -> "pqrs",
      '8' -> "tuv",
      '9' -> "wxyz"
    )

    def helper(digits: List[Char], current: String, result: List[String]): List[String] = {
      if (digits.isEmpty) {
        return result :+ current // push to end
      }
      val letters = digitMap(digits.head)

      letters.flatMap(letter => helper(digits.tail, current + letter, result)).toList
    }

    if (digits.isEmpty) {
      List()
    }
    else {
      helper(digits.toList, "", List())
    }
  }
  test("Phone number combination fo 23") {
    val digitsToString = "23"
    val expected = List("ad", "ae", "af", "bd", "be", "bf", "cd", "ce", "cf")

    val actual = WordsUtils.phoneLettersCombinations(digitsToString)

    assert(expected.size == actual.size, "Lists have different sizes")
    assert(expected.zip(actual).forall(t => t._1 == t._2), "Lists have different elements")
  }
```

White is a function that, given an array of strings, would group the anagrams together.

```
def groupAnagrams(words: Array[String]): List[Array[String]] = {
    val map = words.groupBy(_.sorted)
    map.foreach(kv => println(s"${kv._1}: ${kv._2.mkString(", ")}")) // just to print
    map.values.toList
}
```

Bonus Tasks:

Write a function to convert Arabic numbers to roman numerals.

```
def toRoman(num: Int): String = {
    val numerals = List(
      (1000, "M"), (900, "CM"), (500, "D"), (400, "CD"),
      (100, "C"), (90, "XC"), (50, "L"), (40, "XL"),
      (10, "X"), (9, "IX"), (5, "V"), (4, "IV"), (1, "I")
    )

    def convert(remaining: Int, result: String): String = {
      if (remaining == 0) result
      else {
        val (arabic, roman) = numerals.find {
          case (arabic, roman) => arabic <= remaining
        }.get

        convert(remaining - arabic, result + roman)
      }
    }

    convert(num, "")
  }

test("Convert arabic 13 to romans") {
    val numbers = 999
    val expected = "CMXCIX"

    val actual = NumberUtils.toRoman(999)

    assert(expected == actual)
  }
```

Write a function to calculate the prime factorization of an integer.

```
def primeFactorization(n: Int): List[Int] = {
    def primeFactors(x: Int, factors: List[Int]): List[Int] = {
      if (isPrime(x)) x :: factors
      else {
        val nextFactor = (2 to x).find(x % _ == 0).get
        primeFactors(x / nextFactor, nextFactor :: factors)
      }
    }

    primeFactors(n, Nil)
  }

test("Number factorization of prime") {
    val number = 13
    val expected = List(13)

    val actual = NumberUtils.primeFactorization(13)
    assert(expected.size == actual.size, "Lists have different sizes")
    assert(expected.zip(actual).forall(t => t._1 == t._2), "Lists have different elements")
  }

  test("Number factorization of number") {
    val number = 42
    val expected = List(7, 3, 2)

    val actual = NumberUtils.primeFactorization(42)

    assert(expected.size == actual.size, "Lists have different sizes")
    assert(expected.zip(actual).forall(t => t._1 == t._2), "Lists have different elements")
  }
```

P0W3 – An Actor is Born

This week focuses on learning what is an actor and how it basically works.

Minimal Tasks:

Create an actor that prints on the screen any message it receives.

```
class PrintMessageActor extends Actor {
   def receive: Receive = {
     case message => println(message);
   }
}

object PrintMessageActor extends App {
  private val actorSystem = ActorSystem("ActorSystem")
  private val printActor = actorSystem.actorOf(Props[PrintMessageActor], "printActor")

  printActor ! "Hello, PTR!"
  actorSystem.terminate()
}
```

Create an actor that returns any message it receives, while modifying it. Infer
the modification from the following example:

```
> Pid ! 10. % Integers
2 Received : 11
3 > Pid ! " Hello ". % Strings
4 Received : hello
5 > Pid ! {10 , " Hello "}. % Anything else
6 Received : I don ’ t know how to HANDLE this !
```

Create two actors, actor one ”monitoring” the other. If the second actor stops, actor one gets notified via a message

```
class ModifyMessageActor extends Actor{
  override def receive: Receive = {
    case message: Int => sender() ! (message + 1)
    case message: String => sender() ! message.toLowerCase
    case _ => sender() ! "Idk what to do with that"
  }
}

object ModifyMessageActor extends App {
  implicit val timeout: Timeout = Timeout(5.seconds)
  private val actorSystem = ActorSystem("ActorSystem")
  private val modifierActor = actorSystem.actorOf(Props[ModifyMessageActor], "modifyActor")

  (modifierActor ? 10).map { response =>
    println("Received : " + response)
  }

  (modifierActor ? "Some String").map { response =>
    println("Received : " + response)
  }
  (modifierActor ? {
    "obj" -> true
  }).map { response =>
    println("Received : " + response)
  }

  actorSystem.terminate()
}
```

Create an actor which receives numbers and with each request prints out the current average.

```
class AccumulatorActor extends Actor {
  private var sum: Int = 0
  private var count: Int = 0

   def receive: Receive = {
    case number: Int =>
      sum += number
      count += 1
      println("Average: " + (sum.toDouble/count.toDouble))

    case _ =>
      println("I dont know how to process it")
   }
}

object AccumulatorActor extends App {
  private val actorSystem = ActorSystem("ActorSystem")

  private val parentActor = actorSystem.actorOf(Props[AccumulatorActor], "accumulator")

  parentActor ! 0
  parentActor ! 10
  parentActor ! "text"
  parentActor ! 10
  parentActor ! 10

  actorSystem.terminate()
}
```

Main Tasks:

Create an actor which maintains a simple FIFO queue. You should write helper functions to create an API for the user, which hides how the queue is implemented.

```
case object Pop
case class Push(value: Any)

class QueueActor extends Actor {
  private val queue: List[Any] = List.empty

  def receive: Receive = onMessage(queue)

  private def onMessage(queue: List[Any]): Receive = {
    case Push(value) =>
      context.become(onMessage(queue :+ value))
    case Pop =>
      queue match {
        case Nil => sender() ! None
        case head :: tail =>
          context.become(onMessage(tail))
          sender() ! head
      }
  }
}

object QueueAPI {
  def push(queue: ActorRef, value: Any): Unit =
    queue ! Push(value)

  def pop(queue: ActorRef): Future[Unit]  = {
    implicit val timeout: Timeout = 5.seconds
    (queue ? Pop).map { response =>
      println("Received : " + response)
    }
  }
}

object QueueApp extends App {
  implicit val timeout: Timeout = Timeout(5.seconds)
  private val actorSystem = ActorSystem("ActorSystem")

  private val queueActor = actorSystem.actorOf(Props[QueueActor])
  push(queueActor, 10);
  push(queueActor, 3);
  pop(queueActor);
  pop(queueActor);
  pop(queueActor);
}
```

Create a module that would implement a semaphore.

```
class Semaphore(private var count: Int) {
  def acquire(): Unit = synchronized {
    while (count == 0) {
      wait()
    }
    count = count - 1
  }

  def release(): Unit = synchronized {
    count = count + 1
    notify()
  }
}

object Semaphore extends App {
  private val semaphore = new Semaphore(1)
  private val threads = (1 to 3).map { i =>
    new Thread(() => {
      semaphore.acquire()
        println(s"Thread $i entered the critical section")
        // Simulate some work
        Thread.sleep(1000)
        semaphore.release()
        println(s"Thread $i exited the critical section")
    })
  }

  threads.foreach(_.start())
  threads.foreach(_.join())

  println("All threads finished")
}
```

P0W4 – The Actor is dead… Long live the Actor

Minimal Tasks:

Create a supervised pool of identical worker actors. The number of actors is static, given at initialization. Workers should be individually addressable. Worker actors should echo any message they receive. If an actor dies (by receiving a “kill” message), it should be restarted by the supervisor. Logging is welcome.

```
case class Work(msg: String)
case object KillWorker

class Master(numWorkers: Int) extends Actor {
  var router: Router = {
    val routees = Vector.fill(numWorkers) {
      val r = context.actorOf(Props[Worker]())
      context.watch(r)
      ActorRefRoutee(r)
    }
    Router(RoundRobinRoutingLogic(), routees)
  }

  def receive: Receive = {
    case w: Work =>
      router.route(w, sender())

    case KillWorker =>
      println("Killing the next available actor!")
      router.route(PoisonPill, sender())

    case Terminated(actor) =>
      println(s"${actor.path} was killed, recreating new one")
      router = router.removeRoutee(actor)
      val reference = context.actorOf(Props[Worker]())
      context.watch(reference)
      router = router.addRoutee(reference)
  }
}

class Worker extends Actor {
  override def receive: Receive = {
    case Work(msg) => println(s"${self.path} received message: $msg")
  }
}

object SupervisedPool extends App {
  private val system = ActorSystem("TestSystem")

  private val master = system.actorOf(Props(classOf[Master], 2), "master")

  master ! Work("Message 1")
  sleep(1000)
  master ! Work("Message 2")
  sleep(1000)
  master ! Work("Message 3")
  sleep(1000)
  master ! KillWorker
  sleep(1000)
  master ! Work("Message 4")
  master ! Work("Message 5")
  master ! Work("Message 6")
  master ! Work("Message 7")

  system.terminate()
}
```

Main Tasks:

Create a supervised processing line to clean messy strings. The first work in
the line would split the string by any white spaces (similar to Python’s str. split method).
The second actor will lowercase all words and swap all m’s and n’s (you nomster!). The third
actor will join back the sentence with one space between words (similar to Python’s str. join
method). Each worker will receive as input the previous actor’s output, and the last actor printing
the result on the screen. If any of the workers die because it encounters an error, the whole
processing line needs to be restarted. Logging is welcome.

```
case class StartProcessing(msg: String)
case class ProcessInput(msg: Any)
case class SplitResult(msg: Any)
case class SpawnedResult(msg: Any)
case class JoinedResult(msg: Any)

class Supervisor extends Actor {

  private val splitter = context.actorOf(Props[SplitActor], name = "splitter")
  context.watch(splitter)

  private val spawner = context.actorOf(Props[SwapAndLowercaseActor], name = "spawner")
  context.watch(spawner)

  private val joiner = context.actorOf(Props[JoinActor], name = "joiner")
  context.watch(joiner)

  private val printer = context.actorOf(Props[PrinterActor], name = "printer")
  context.watch(printer)


  override def receive: Receive = {
    case StartProcessing(msg) => splitter ! ProcessInput(msg)
    case SplitResult(msg) => spawner ! ProcessInput(msg)
    case SpawnedResult(msg) => joiner ! ProcessInput(msg)
    case JoinedResult(msg) => printer ! ProcessInput(msg)
  }

  override def supervisorStrategy: SupervisorStrategy =
    OneForOneStrategy() {
      case _: Exception =>
        println("Worker actor encountered an exception, restarting...")
        Restart
      case _: DeathPactException =>
        println("Worker actor encountered an exception, restarting...")
        Restart
    }
}

class SplitActor() extends Actor {
  def receive: Receive = {
    case ProcessInput(input: String) =>
      val splitString = input.split("\\s+")
      println(s"Message split: ${splitString.mkString("Array(", ", ", ")")}")
      sender() ! SplitResult(splitString)
  }
}

class SwapAndLowercaseActor() extends Actor {
  def receive: Receive = {
    case ProcessInput(splitString: Array[String]) =>
      val cleanedString = splitString.map(
        _.toLowerCase()
        .replace("n", "*****")
        .replace("m", "n")
        .replace("*****", "m")
      )
      println(s"Message spawn: ${cleanedString.mkString("Array(", ", ", ")")}")
      sender() ! SpawnedResult(cleanedString)
  }
}
```

```
class JoinActor() extends Actor {
  def receive: Receive = {
    case ProcessInput(cleanedString: Array[String]) =>
      val joinedString = cleanedString.mkString(" ")
      println(s"Message joined: $joinedString")
      sender() ! JoinedResult(joinedString)
  }
}

class PrinterActor() extends Actor {
  def receive: Receive = {
    case ProcessInput(cleanedString) =>
      println(s"Printing message: $cleanedString")
  }
}

object SupervisedProcessingLine extends App {
  val system = ActorSystem("SupervisedProcessingLine")

  private val supervisor = system.actorOf(Props(classOf[Supervisor]), "supervisor")

  supervisor ! StartProcessing("you nomster!")
  supervisor ! StartProcessing("you nomster2!")
}
```

P0W5 – May the Web be with you

Minimal Tasks:

Write an application that would visit https://quotes.toscrape.com/. Print out the HTTP response status code, response headers, and response body.

```
implicit val system: ActorSystem = ActorSystem("system")

private val request = HttpRequest(uri = "https://quotes.toscrape.com/")
private val responseFuture = Http().singleRequest(request)

printResponseInfo(responseFuture)

private def printResponseInfo(responseFuture: Future[HttpResponse]): Unit = {
    responseFuture.map { response =>
      println(s"HTTP status code: ${response.status}")
      println("HTTP response headers:")
      response.headers.foreach(h => println(s"${h.name}: ${h.value}"))
      println("HTTP response body:")
      response.entity.dataBytes.runForeach { chunk =>
        println(chunk.utf8String)
      }
    }
  }
```

Continue your previous application. Extract all quotes from the HTTP response body. Collect the author of the quote, the quote text, and tags. Save the data into a list of maps, each map representing a single quote.

```
private def parseQuotes(responseFuture: Future[HttpResponse]): Unit = {
    responseFuture.flatMap { response =>
      response.entity.dataBytes
        .runFold("")(_ + _.utf8String) //consume the response body bytes as a stream, and concatenate them into a single string
        .map { responseBody =>
          val quoteRegex = """<span class="text" itemprop="text">([^<]+)</span>""".r
          val authorRegex = """<span>by <small class="author" itemprop="author">([^<]+)</small>""".r
          val tagRegex = """<meta class="keywords" itemprop="keywords" content="([^<]+)" /    >""".r
          quoteRegex.findAllMatchIn(responseBody).map { quoteMatch =>
            val quoteText = quoteMatch.group(1).toString
            val authorName = authorRegex.findFirstMatchIn(quoteMatch.after.toString).get.group(1).toString
            val tags = tagRegex.findFirstMatchIn(quoteMatch.after.toString).get.group(1).split(",").toList

            Map("author" -> authorName, "quote" -> quoteText, "tags" -> tags)
          }.toList
        }
    }.onComplete{
        case Success(response) =>
          response.foreach(println)

        case Failure(e) =>
          println("Request failed: " + e.getMessage)
    }
  }
```

Main Tasks:

Write an application that would implement a Star Wars-themed RESTful API. The API should implement the following HTTP methods:

```
• GET /movies
• GET /movies/:id
• POST /movies
• PUT /movies/:id
• PATCH /movies/:id
• DELETE /movies/:id
```

Use a database to persist your data. Populate the database with the following information:

```
[
 {
 " id ": 1 ,
 " title ": " Star Wars : Episode IV - A New Hope " ,
 " release_year ": 1977 ,
 " director ": " George Lucas "
 } ,
 {
 " id ": 2 ,
 " title ": " Star Wars : Episode V - The Empire Strikes Back " ,
 " release_year ": 1980 ,
 " director ": " Irvin Kershner "
 } ,
 {
 " id ": 3 ,
 " title ": " Star Wars : Episode VI - Return of the Jedi " ,
 " release_year ": 1983 ,
 " director ": " Richard Marquand "
 } ,
 {
 " id ": 4 ,
 " title ": " Star Wars : Episode I - The Phantom Menace " ,
 " release_year ": 1999 ,
 " director ": " George Lucas "
 } ,
 {
 " id ": 5 ,
 " title ": " Star Wars : Episode II - Attack of the Clones " ,
 " release_year ": 2002 ,
 " director ": " George Lucas "
 } ,
 {
 " id ": 6 ,
 " title ": " Star Wars : Episode III - Revenge of the Sith " ,
 " release_year ": 2005 ,
 " director ": " George Lucas "
 } ,
 {

 " id ": 7 ,
 " title ": " Star Wars : The Force Awakens " ,
 " release_year ": 2015 ,
 " director ": " J . J . Abrams "
 } ,
 {
 " id ": 8 ,
 " title ": " Rogue One : A Star Wars Story " ,
 " release_year ": 2016 ,
 " director ": " Gareth Edwards "
 } ,

 " id ": 9 ,
 " title ": " Star Wars : The Last Jedi " ,
 " release_year ": 2017 ,
 " director ": " Rian Johnson "
 } ,
 {
 " id ": 10 ,
 " title ": " Solo : A Star Wars Story " ,
 " release_year ": 2018 ,
 " director ": " Ron Howard "
 } ,
 {
 " id ": 11 ,
 " title ": " Star Wars : The Rise of Skywalker " ,
 " release_year ": 2019 ,
 " director ": " J . J . Abrams "
 }
]
```

For this laboratory task, a Scala Http4s server was created with an in-memory database, of the movies:

```
case class Movie(id: Int, title: String, release_year: Int, director: String)

  implicitly[Encoder[Movie]]
  implicitly[Decoder[Movie]]

  val moviesDB: mutable.Map[Int, Movie] = mutable.Map(
    1 -> Movie(1, "Star Wars : Episode IV - A New Hope", 1977, "George Lucas"),
    2 -> Movie(2, "Star Wars : Episode V - The Empire Strikes Back", 1980, "Irvin Kershner"),
    3 -> Movie(3, "Star Wars : Episode VI - Return of the Jedi", 1983, "Richard Marquand"),
    4 -> Movie(4, "Star Wars : Episode I - The Phantom Menace", 1999, "George Lucas"),
    5 -> Movie(5, "Star Wars : Episode II - Attack of the Clones", 2002, "George Lucas"),
    6 -> Movie(6, "Star Wars : Episode III - Revenge of the Sith", 2005, "George Lucas"),
    7 -> Movie(7, "Star Wars : The Force Awakens", 2015, "J.J. Abrams"),
    8 -> Movie(8, "Rogue One : A Star Wars Story", 2016, "Gareth Edwards"),
    9 -> Movie(9, "Star Wars : The Last Jedi", 2017, "Rian Johnson"),
    10 -> Movie(10, "Solo : A Star Wars Story", 2018, "Ron Howard"),
    11 -> Movie(11, "Star Wars : The Rise of Skywalker", 2019, "J.J. Abrams")
  )
```

The router object contains base routes of the application:

```
private def movieRoutes[F[_] : Concurrent]: HttpRoutes[F] = {
    val dsl = Http4sDsl[F]
    import dsl._
    implicit val movieDecoder: EntityDecoder[F, Movie] = jsonOf[F, Movie]

    HttpRoutes.of[F] {
      case GET -> Root / "movies" =>
        Ok(moviesDB.asJson)

      case GET -> Root / "movies" / id =>
        moviesDB.get(id.toInt) match {
          case Some(movie) => Ok(movie.asJson)
          case _ => NotFound("Not found")
        }

      case DELETE -> Root / "movies" / id =>
        moviesDB.get(id.toInt) match {
          case Some(movie) => moviesDB -= movie.id
            Ok()
          case _ => NotFound("Not found")
        }
    }
  }
```

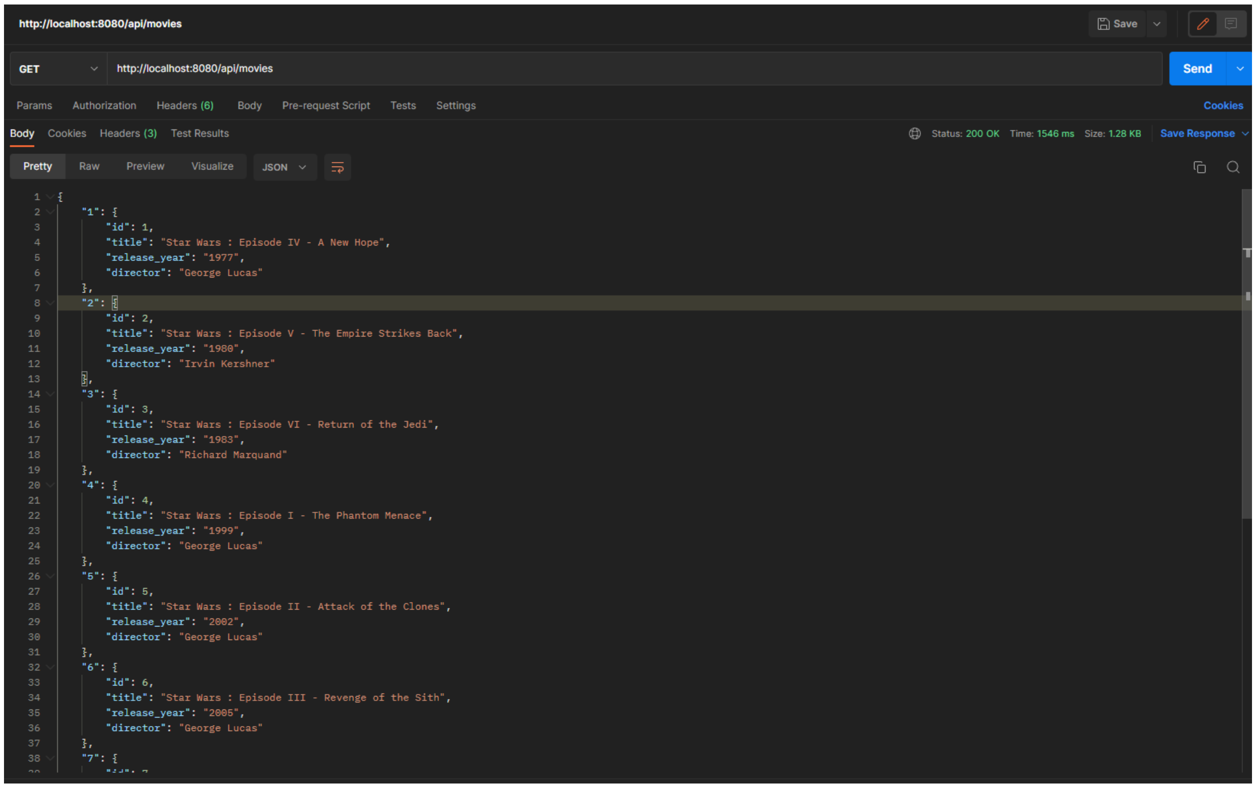
I was encountering problems with POST request body parsing so the routes that implies data sending were not implemented.

To run the application the class, it should extend `IOApp` and overwrite the `run` method. In this method instance is specified as the port, the host, the bonded routes, in case no routes match the route object, a not found is thrown.

```
object RestfullAPI extends IOApp {
 // ...
override def run(args: List[String]): IO[ExitCode] = {
    val apis = Router(
      "/api" -> movieRoutes[IO],
    ).orNotFound

    BlazeServerBuilder[IO](runtime.compute)
      .bindHttp(8080, "localhost")
      .withHttpApp(apis)
      .resource
      .use(_ => IO.never)
      .as(ExitCode.Success)
  }
}
```

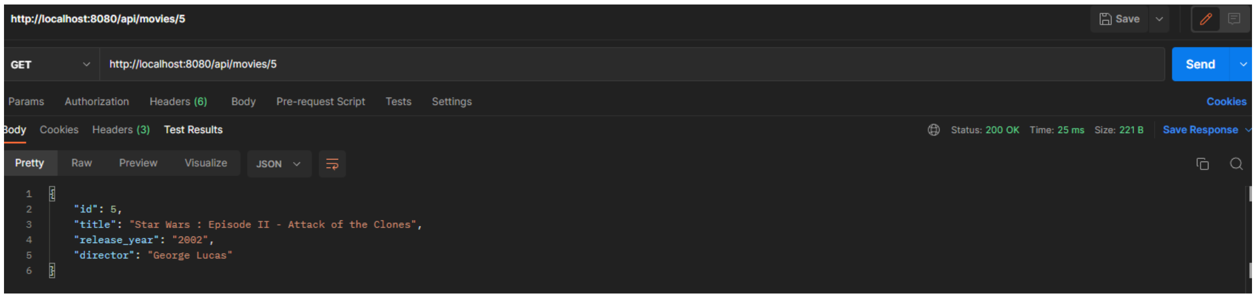
Some of the results of GET requests:



```
http://localhost:8080/api/movies
GET http://localhost:8080/api/movies
Status: 200 OK  Time: 1546 ms  Size: 1.28 KB
```

```
{
    "1": {
        "id": 1,
        "title": "Star Wars : Episode IV - A New Hope",
        "release_year": "1977",
        "director": "George Lucas"
    },
    "2": {
        "id": 2,
        "title": "Star Wars : Episode V - The Empire Strikes Back",
        "release_year": "1980",
        "director": "Irvin Kershner"
    },
    "3": {
        "id": 3,
        "title": "Star Wars : Episode VI - Return of the Jedi",
        "release_year": "1983",
        "director": "Richard Marquand"
    },
    "4": {
        "id": 4,
        "title": "Star Wars : Episode I - The Phantom Menace",
        "release_year": "1999",
        "director": "George Lucas"
    },
    "5": {
        "id": 5,
        "title": "Star Wars : Episode II - Attack of the Clones",
        "release_year": "2002",
        "director": "George Lucas"
    },
    "6": {
        "id": 6,
        "title": "Star Wars : Episode III - Revenge of the Sith",
        "release_year": "2005",
        "director": "George Lucas"
    },
    "7": {
```

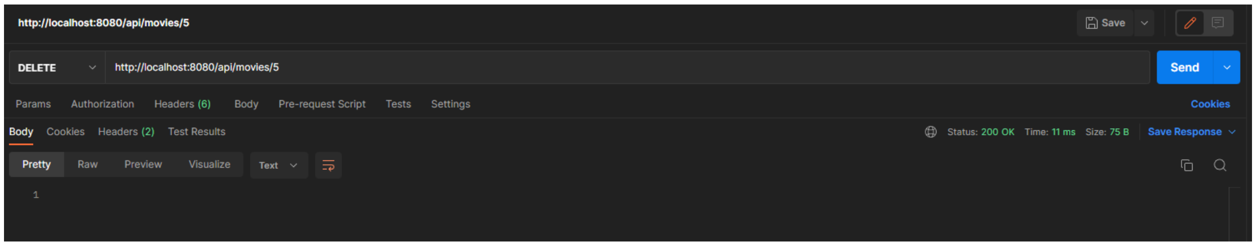
Get of a specific id:



```
http://localhost:8080/api/movies/5
GET http://localhost:8080/api/movies/5
Status: 200 OK  Time: 25 ms  Size: 221 B
```

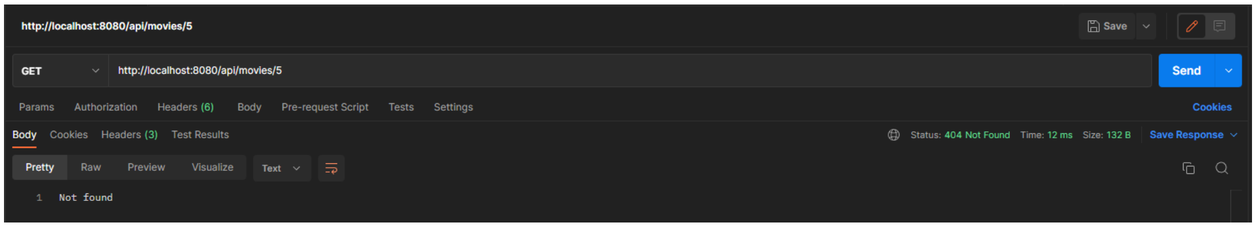
```
{
    "id": 5,
    "title": "Star Wars : Episode II - Attack of the Clones",
    "release_year": "2002",
    "director": "George Lucas"
}
```

Delete of specific id:



```
http://localhost:8080/api/movies/5
DELETE http://localhost:8080/api/movies/5
Status: 200 OK  Time: 11 ms  Size: 75 B
```

Get the previously deleted id:



```
Not found
```

Conclusions:

In this lab work, we focused on functional programming and the Actor model. The first week involved installing Scala and writing a script that printed a message on the screen. We also initialized a version control system (VCS) repository and wrote a unit test for our project.

In the second week, we wrote several functions to perform various tasks, such as determining if a number is prime, calculating the area of a cylinder, and other various tasks that will get us familiar with functional programming.

In the third and fourth weeks, we created actors that printed and modified messages. We also explored the Actor model in more depth and learned how to create actors that could communicate with each other.

In the fifth week, we put into practice all accumulated skills by implementing real-life task examples such as a Web Scraper and a RESTfull API.

Overall, this lab work provided us with a solid foundation in functional programming and the Actor model. We learned how to write functions to perform various tasks and how to use actors to build scalable systems.