ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ БЮДЖЕТНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

МОСКОВСКИЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ СВЯЗИ И ИНФОРМАТИКИ

(МТУСИ)

Проект

Введение в Информационные технологии

Выполнил студент

группы БВТ1903:

Сушков И.А.

Проверила:

Мосева М.С.

Москва
2021

Содержимое файла JsonFormats.scala

```
package com.example

import com.example.UserRegistry.ActionPerformed

//#json-formats
import spray.json.DefaultJsonProtocol

object JsonFormats  {
  // import the default encoders for primitive types (Int, String, Lists etc)
  import DefaultJsonProtocol._

  implicit val userJsonFormat = jsonFormat3(User)
  implicit val usersJsonFormat = jsonFormat1(Users)

  implicit val actionPerformedJsonFormat = jsonFormat1(ActionPerformed)
}
```

Содержимое файла QuickstartApp.scala

```
package com.example

import akka.actor.typed.ActorSystem
import akka.actor.typed.scaladsl.Behaviors
import akka.http.scaladsl.Http
import akka.http.scaladsl.server.Route

import scala.util.{Failure, Success}

//#main-class
object QuickstartApp {
  //#start-http-server
  private def startHttpServer(routes: Route)(implicit system:
ActorSystem[_]): Unit = {
    // Akka HTTP still needs a classic ActorSystem to start
    import system.executionContext

    val futureBinding = Http().newServerAt("localhost", 8080).bind(routes)
    futureBinding.onComplete {
      case Success(binding) =>
        val address = binding.localAddress
        system.log.info("Server online at http://{}:{}/",
address.getHostString, address.getPort)
      case Failure(ex) =>
        system.log.error("Failed to bind HTTP endpoint, terminating system",
ex)
        system.terminate()
    }
  }
  //#start-http-server
  def main(args: Array[String]): Unit = {
    //#server-bootstrapping
    val rootBehavior = Behaviors.setup[Nothing] { context =>
      val userRegistryActor = context.spawn(UserRegistry(),
"UserRegistryActor")
      context.watch(userRegistryActor)

      val routes = new UserRoutes(userRegistryActor)(context.system)
      startHttpServer(routes.userRoutes)(context.system)

      Behaviors.empty
```

```
    }
    val system = ActorSystem[Nothing](rootBehavior, "HelloAkkaHttpServer")
    //#server-bootstrapping
  }
}
//#main-class
```

Содержимое файла UserRoutesSpec.scala

```
package com.example

import akka.actor.testkit.typed.scaladsl.ActorTestKit
import akka.http.scaladsl.marshalling.Marshal
import akka.http.scaladsl.model._
import akka.http.scaladsl.testkit.ScalatestRouteTest
import org.scalatest.concurrent.ScalaFutures
import org.scalatest.matchers.should.Matchers
import org.scalatest.wordspec.AnyWordSpec

//#set-up
class UserRoutesSpec extends AnyWordSpec with Matchers with ScalaFutures with
ScalatestRouteTest {
  //#test-top

  // the Akka HTTP route testkit does not yet support a typed actor system
(https://github.com/akka/akka-http/issues/2036)
  // so we have to adapt for now
  lazy val testKit = ActorTestKit()
  implicit def typedSystem = testKit.system
  override def createActorSystem(): akka.actor.ActorSystem =
    testKit.system.classicSystem

  // Here we need to implement all the abstract members of UserRoutes.
  // We use the real UserRegistryActor to test it while we hit the Routes,
  // but we could "mock" it by implementing it in-place or by using a
TestProbe
  // created with testKit.createTestProbe()
  val userRegistry = testKit.spawn(UserRegistry())
  lazy val routes = new UserRoutes(userRegistry).userRoutes

  // use the json formats to marshal and unmarshall objects in the test
  import akka.http.scaladsl.marshallers.sprayjson.SprayJsonSupport._
  import JsonFormats._
  //#set-up

  //#actual-test
  "UserRoutes" should {
    "return no users if no present (GET /users)" in {
      // note that there's no need for the host part in the uri:
      val request = HttpRequest(uri = "/users")

      request ~> routes ~> check {
        status should ===(StatusCodes.OK)

        // we expect the response to be json:
        contentType should ===(ContentTypes.`application/json`)

        // and no entries should be in the list:
        entityAs[String] should ===("""{"users":[]}""")
      }
    }
    //#actual-test
```

```
    //#testing-post
    "be able to add users (POST /users)" in {
      val user = User("Kapi", 42, "jp")
      val userEntity = Marshal(user).to[MessageEntity].futureValue //
futureValue is from ScalaFutures

      // using the RequestBuilding DSL:
      val request = Post("/users").withEntity(userEntity)

      request ~> routes ~> check {
        status should ===(StatusCodes.Created)

        // we expect the response to be json:
        contentType should ===(ContentTypes.`application/json`)

        // and we know what message we're expecting back:
        entityAs[String] should ===("""{"description":"User Kapi
created."}""")
      }
    }
    //#testing-post

    "be able to remove users (DELETE /users)" in {
      // user the RequestBuilding DSL provided by ScalatestRouteSpec:
      val request = Delete(uri = "/users/Kapi")

      request ~> routes ~> check {
        status should ===(StatusCodes.OK)

        // we expect the response to be json:
        contentType should ===(ContentTypes.`application/json`)

        // and no entries should be in the list:
        entityAs[String] should ===("""{"description":"User Kapi
deleted."}""")
      }
    }
    //#actual-test
  }
  //#actual-test

  //#set-up
}
```

Содержимое файла UserRoutes.scala

```
package com.example

import akka.http.scaladsl.server.Directives._
import akka.http.scaladsl.model.StatusCodes
import akka.http.scaladsl.server.Route

import scala.concurrent.Future
import com.example.UserRegistry._
import akka.actor.typed.ActorRef
import akka.actor.typed.ActorSystem
import akka.actor.typed.scaladsl.AskPattern._
import akka.util.Timeout
```

```
//#import-json-formats
//#user-routes-class
class UserRoutes(userRegistry: ActorRef[UserRegistry.Command])(implicit val
system: ActorSystem[_]) {

  //#user-routes-class
  import akka.http.scaladsl.marshallers.sprayjson.SprayJsonSupport._
  import JsonFormats._
  //#import-json-formats

  // If ask takes more time than this to complete the request is failed
  private implicit val timeout =
Timeout.create(system.settings.config.getDuration("my-app.routes.ask-
timeout"))

  def getUsers(): Future[Users] =
    userRegistry.ask(GetUsers)
  def getUser(name: String): Future[GetUserResponse] =
    userRegistry.ask(GetUser(name, _))
  def createUser(user: User): Future[ActionPerformed] =
    userRegistry.ask(CreateUser(user, _))
  def deleteUser(name: String): Future[ActionPerformed] =
    userRegistry.ask(DeleteUser(name, _))

  //#all-routes
  //#users-get-post
  //#users-get-delete
  val userRoutes: Route =
  pathPrefix("users") {
    concat(
      //#users-get-delete
      pathEnd {
        concat(
          get {
            complete(getUsers())
          },
          post {
            entity(as[User]) { user =>
              onSuccess(createUser(user)) { performed =>
                complete((StatusCodes.Created, performed))
              }
            }
          })
      },
      //#users-get-delete
      //#users-get-post
      path(Segment) { name =>
        concat(
          get {
            //#retrieve-user-info
            rejectEmptyResponse {
              onSuccess(getUser(name)) { response =>
                complete(response.maybeUser)
              }
            }
            //#retrieve-user-info
          },
          delete {
            //#users-delete-logic
            onSuccess(deleteUser(name)) { performed =>
              complete((StatusCodes.OK, performed))
            }
            //#users-delete-logic
```

```
          })
      })
    //#users-get-delete
  }
  //#all-routes
}
```

Содержимое файла UserRegistry.scala

```
package com.example

//#user-registry-actor
import akka.actor.typed.ActorRef
import akka.actor.typed.Behavior
import akka.actor.typed.scaladsl.Behaviors
import scala.collection.immutable

//#user-case-classes
final case class User(name: String, age: Int, countryOfResidence: String)
final case class Users(users: immutable.Seq[User])
//#user-case-classes

object UserRegistry {
  // actor protocol
  sealed trait Command
  final case class GetUsers(replyTo: ActorRef[Users]) extends Command
  final case class CreateUser(user: User, replyTo: ActorRef[ActionPerformed])
extends Command
  final case class GetUser(name: String, replyTo: ActorRef[GetUserResponse])
extends Command
  final case class DeleteUser(name: String, replyTo:
ActorRef[ActionPerformed]) extends Command

  final case class GetUserResponse(maybeUser: Option[User])
  final case class ActionPerformed(description: String)

  def apply(): Behavior[Command] = registry(Set.empty)

  private def registry(users: Set[User]): Behavior[Command] =
    Behaviors.receiveMessage {
      case GetUsers(replyTo) =>
        replyTo ! Users(users.toSeq)
        Behaviors.same
      case CreateUser(user, replyTo) =>
        replyTo ! ActionPerformed(s"User ${user.name} created.")
        registry(users + user)
      case GetUser(name, replyTo) =>
        replyTo ! GetUserResponse(users.find(_.name == name))
        Behaviors.same
      case DeleteUser(name, replyTo) =>
        replyTo ! ActionPerformed(s"User ${name} deleted.")
        registry(users.filterNot(_.name == name))
    }
}
//#user-registry-actor
```

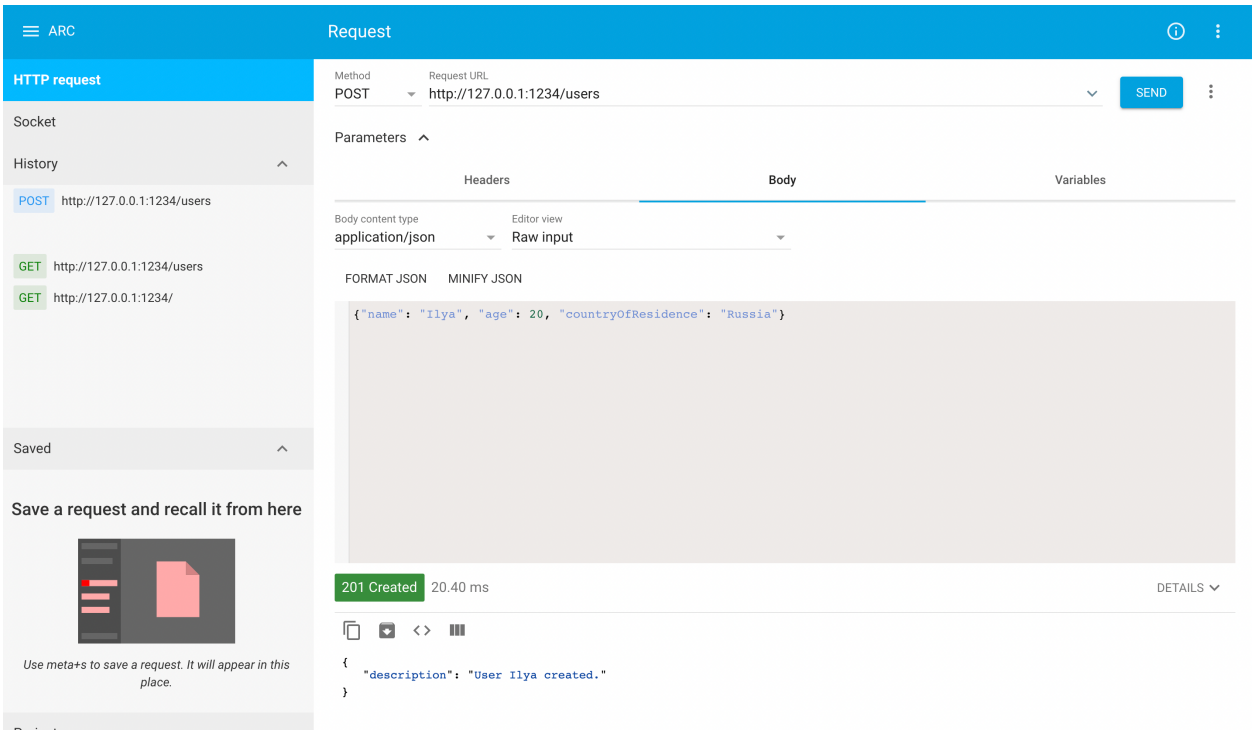

Рис. 1 - Создание пользователя через RESTClient

Рис. 2 - Создание, вывод и удаление пользователя через консоль

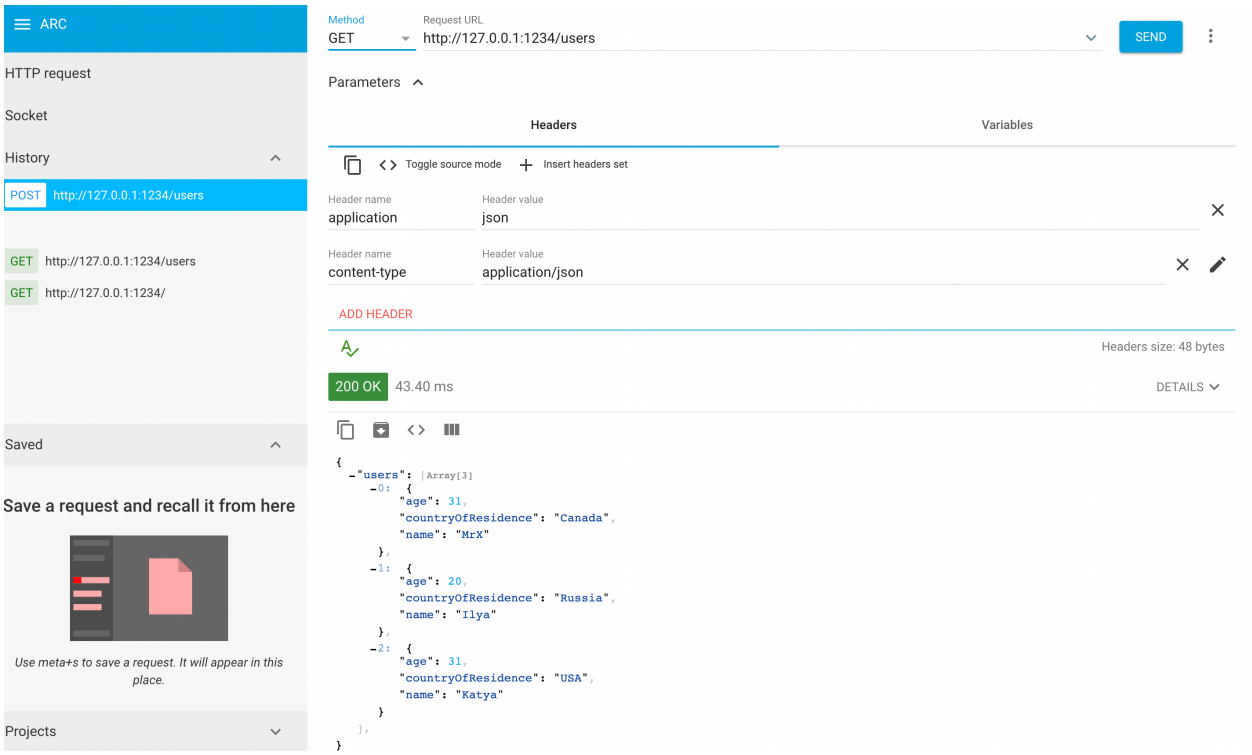

Рис. 4 – Вывод всех пользователей

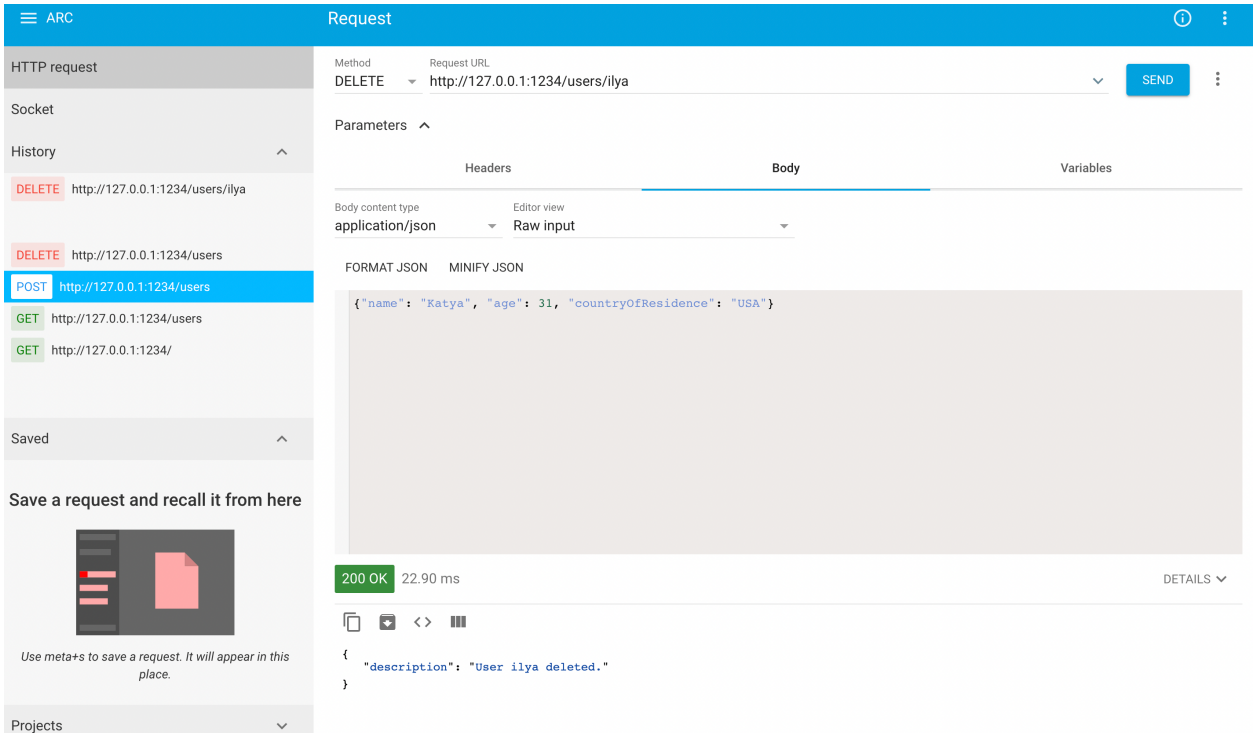

Рис. 5 – Удаление пользователя

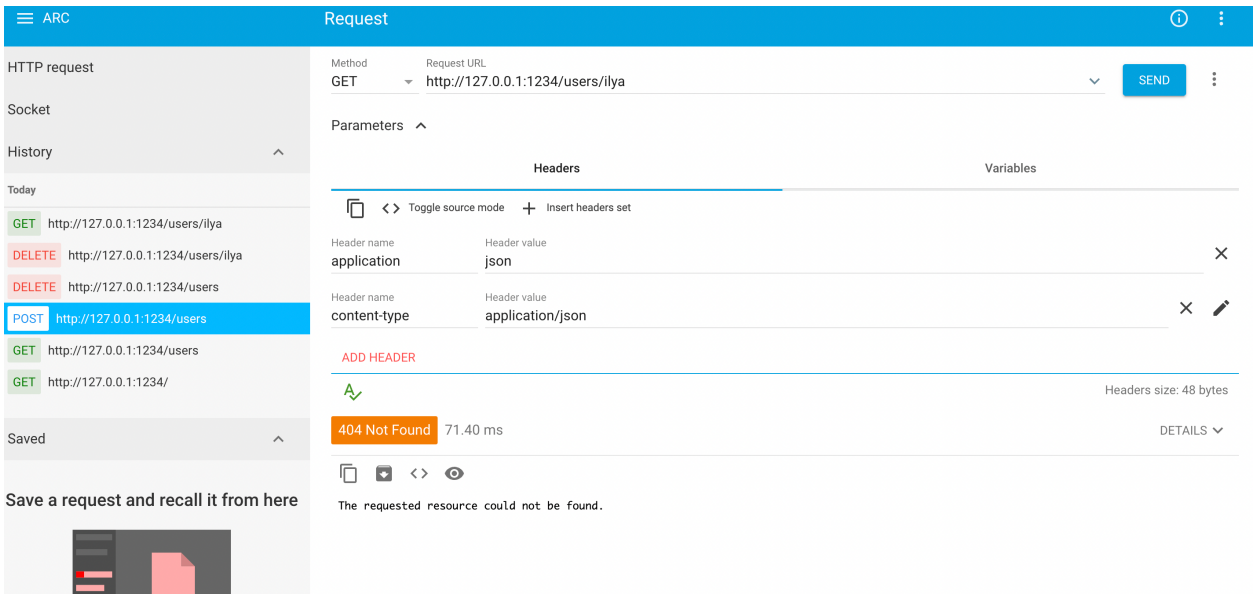

Рис. 6 - Поиск пользователя, которого не существует

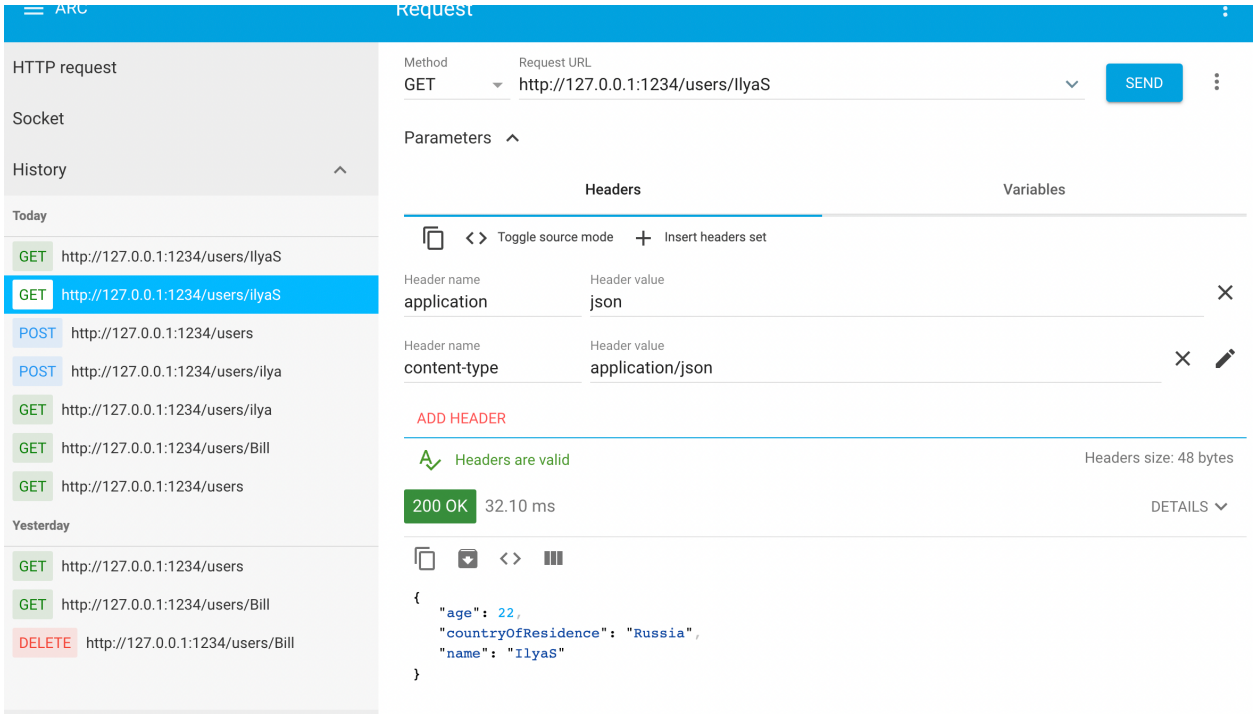

Рис. 7. Получение одного пользователя через GET

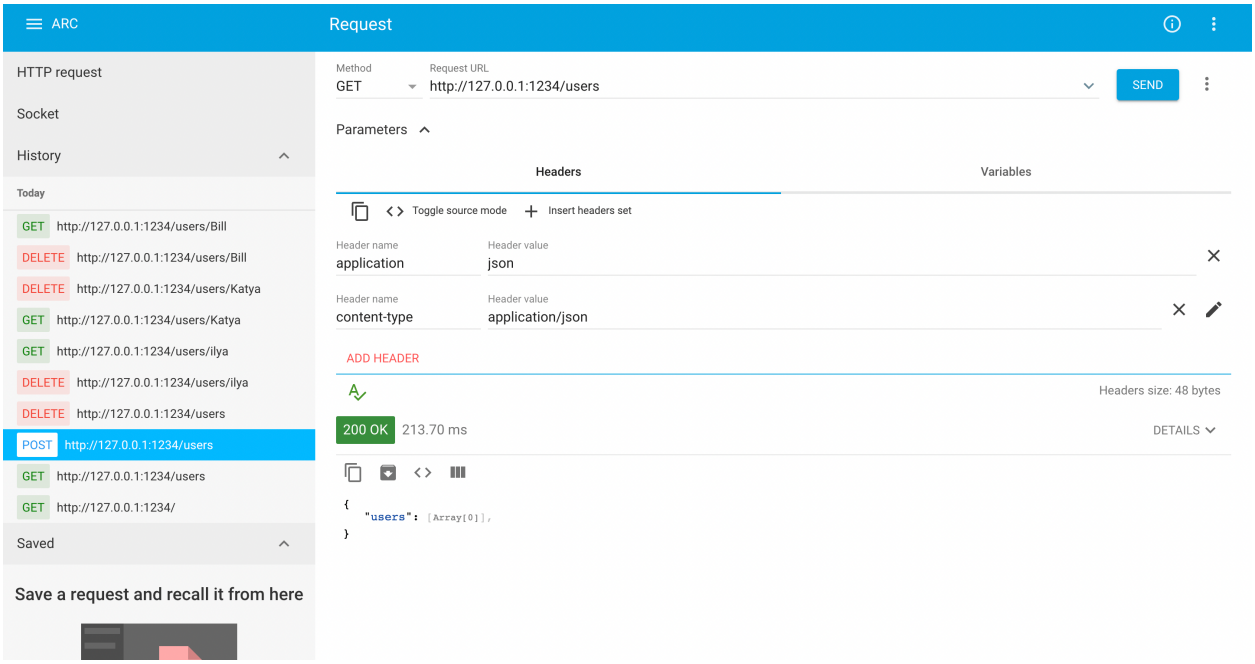

Рис. 8 – Получение пустого списка

Вывод

В ходе выполнения данной курсовой работы я получил базовые навыки работы с Akka. Я запустил и тестировал HTTP-приложение Akka, получил предварительный обзор того, как маршруты упрощают обмен данными по HTTP.