

Министерство цифрового развития, связи и массовых коммуникаций  
Ордена Трудового Красного Знамени федеральное государственное  
бюджетное  
образовательное учреждение высшего образования  
«Московский технический университет связи и информатики»

Кафедра Математической кибернетики и информационных технологий

Проект  
по дисциплине «Введение в информационные технологии»

Выполнил студент группы БВТ1901  
Мартынов Николай Владимирович  
Проверила:  
Мосева Марина Сергеевна

Москва  
2021

## Оглавление

Цель работы.....	3
Задание на разработку.....	3
Теоретические сведения.....	3
Примеры работы программы.....	4
Исходный код программы .....	11
Вывод.....	16

## Цель работы

Цель работы заключается в запуске и тестировании HTTP-приложения Akka, получении предварительного обзора того, как маршруты упрощают обмен данными по HTTP.

## Задание на разработку

Приложение должно быть реализовано в следующих четырех исходных файлах:

- QuickstartApp.scala - содержит основной метод начальной загрузки приложения.
- UserRoutes.scala - HTTP-маршруты Akka, определяющие открытые эндпоинты.
- UserRegistry.scala - актор, обрабатывающий запросы на регистрацию.
- JsonFormats.scala - преобразует данные JSON из запросов в типы Scala и из типов Scala в ответы JSON.

## Теоретические сведения

В акторной модели — которая была изобретена в 1973 году Карлом Хьюиттом и др. — акторы представляют собой «фундаментальные единицы вычислений, реализующие обработку, хранение и коммуникацию». Понятие «фундаментальная единица вычислений» означает, что когда мы пишем программу в соответствии с акторной моделью, наша работа по проектированию и реализации строится вокруг акторов. В сущности, коммуникация — это асинхронный обмен сообщениями, хранение означает, что акторы могут иметь состояние, а обработка заключается в том, что акторы могут иметь дело с сообщениями. Обработка также именуется «поведением».

Актор Akka состоит из нескольких взаимодействующих компонентов. *ActorRef* — это логический адрес актора, позволяющий

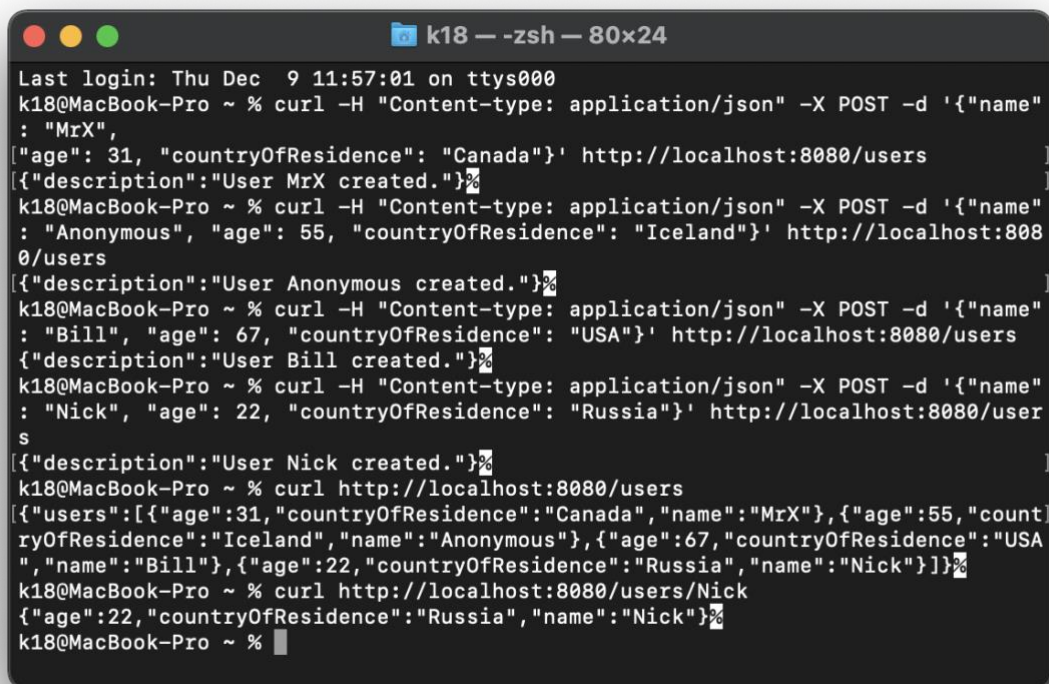
асинхронно отправлять актору сообщения по принципу «послал и забыл». Диспетчер — в данном случае по умолчанию на каждую систему акторов приходится по одному диспетчеру — отвечает за постановку сообщений в очередь, ведущую в почтовый ящик актора, а также приказывает этому ящику изъять из очереди одно или несколько сообщений, но только по одному за раз — и передать их актору на обработку. Последнее, но немаловажное: актор — обычно это единственный API, который нам приходится реализовать — инкапсулирует состояние, и поведение.

Акка не позволяет получить непосредственный доступ к актору и поэтому гарантирует, что единственный способ взаимодействия с актором — это асинхронные сообщения. Невозможно вызвать метод в акторе. Кроме того, необходимо отметить, что отправка сообщения актору и обработка этого сообщения актором — это две отдельных операции, которые, скорее всего, происходят в разных потоках. Разумеется, Акка обеспечивает необходимую синхронизацию, чтобы гарантировать, что любые изменения состояния будут видимы всем потокам.

## Примеры работы программы



Рисунок 1 – Снимок экрана IntelliJ IDEA после запуска сервера



```
k18 — zsh — 80x24
Last login: Thu Dec  9 11:57:01 on ttys000
k18@MacBook-Pro ~ % curl -H "Content-type: application/json" -X POST -d '{"name": "MrX", "age": 31, "countryOfResidence": "Canada"}' http://localhost:8080/users
{"description": "User MrX created."}%
k18@MacBook-Pro ~ % curl -H "Content-type: application/json" -X POST -d '{"name": "Anonymous", "age": 55, "countryOfResidence": "Iceland"}' http://localhost:8080/users
{"description": "User Anonymous created."}%
k18@MacBook-Pro ~ % curl -H "Content-type: application/json" -X POST -d '{"name": "Bill", "age": 67, "countryOfResidence": "USA"}' http://localhost:8080/users
{"description": "User Bill created."}%
k18@MacBook-Pro ~ % curl -H "Content-type: application/json" -X POST -d '{"name": "Nick", "age": 22, "countryOfResidence": "Russia"}' http://localhost:8080/users
{"description": "User Nick created."}%
k18@MacBook-Pro ~ % curl http://localhost:8080/users
{"users": [{"age": 31, "countryOfResidence": "Canada", "name": "MrX"}, {"age": 55, "countryOfResidence": "Iceland", "name": "Anonymous"}, {"age": 67, "countryOfResidence": "USA", "name": "Bill"}, {"age": 22, "countryOfResidence": "Russia", "name": "Nick"}]}%
k18@MacBook-Pro ~ % curl http://localhost:8080/users/Nick
{"age": 22, "countryOfResidence": "Russia", "name": "Nick"}%
k18@MacBook-Pro ~ %
```

Рисунок 2 – Снимок экрана терминала с добавлением записей на сервер при помощи утилиты cURL

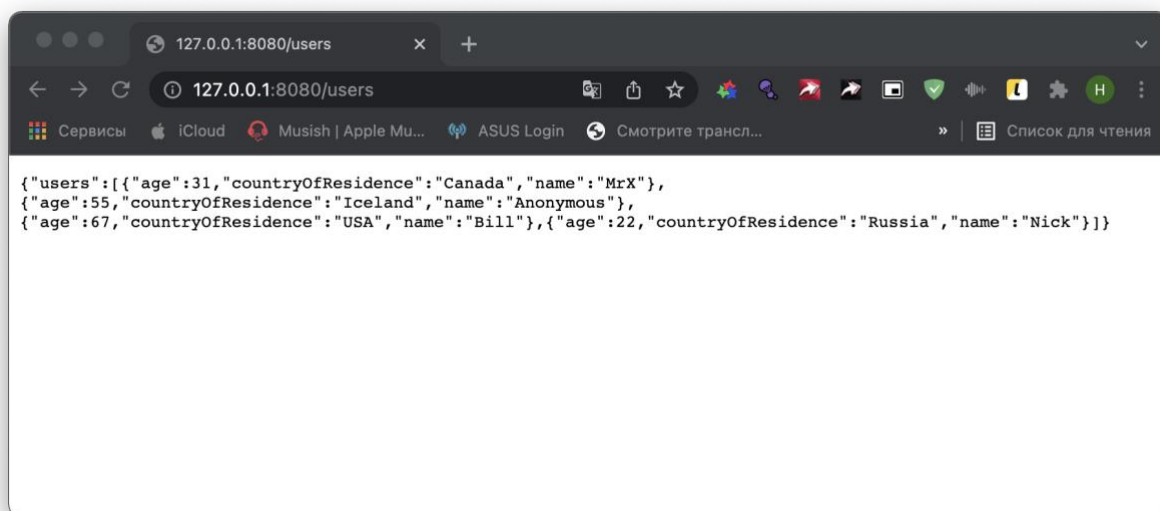
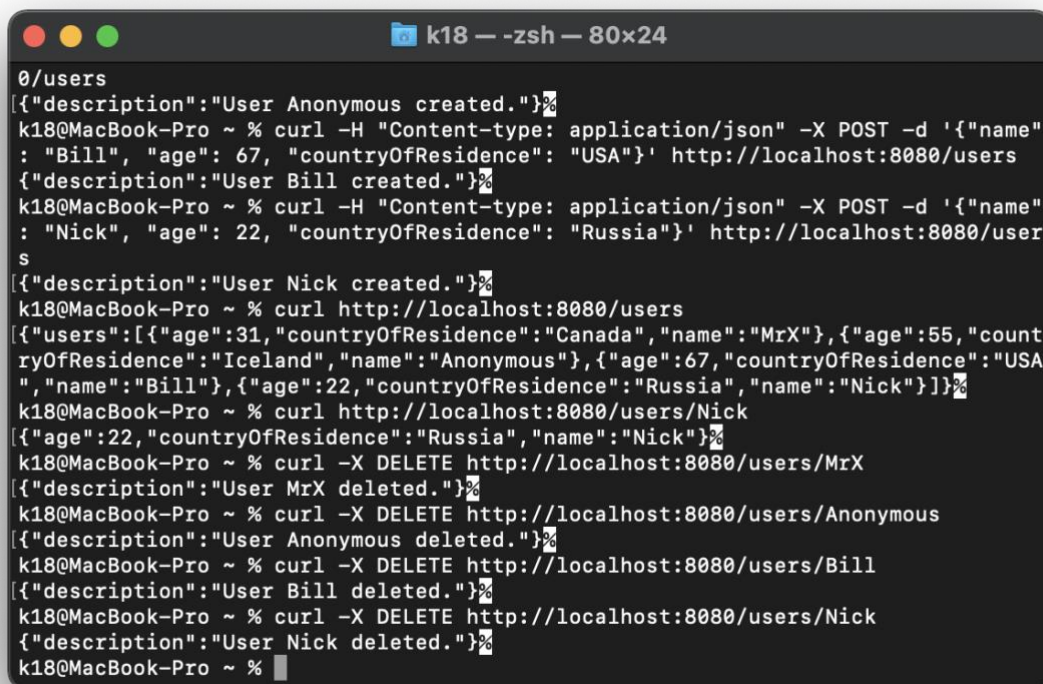


Рисунок 3 – Снимок экрана браузера Chrome с запущенным сервером



```
k18 — zsh — 80x24
0/users
{"description":"User Anonymous created."}%
k18@MacBook-Pro ~ % curl -H "Content-type: application/json" -X POST -d '{"name": "Bill", "age": 67, "countryOfResidence": "USA"}' http://localhost:8080/users
{"description":"User Bill created."}%
k18@MacBook-Pro ~ % curl -H "Content-type: application/json" -X POST -d '{"name": "Nick", "age": 22, "countryOfResidence": "Russia"}' http://localhost:8080/users
{"description":"User Nick created."}%
k18@MacBook-Pro ~ % curl http://localhost:8080/users
{"users":[{"age":31,"countryOfResidence":"Canada","name":"MrX"}, {"age":55,"countryOfResidence":"Iceland","name":"Anonymous"}, {"age":67,"countryOfResidence":"USA","name":"Bill"}, {"age":22,"countryOfResidence":"Russia","name":"Nick"}]}%
k18@MacBook-Pro ~ % curl http://localhost:8080/users/Nick
{"age":22,"countryOfResidence":"Russia","name":"Nick"}%
k18@MacBook-Pro ~ % curl -X DELETE http://localhost:8080/users/MrX
{"description":"User MrX deleted."}%
k18@MacBook-Pro ~ % curl -X DELETE http://localhost:8080/users/Anonymous
{"description":"User Anonymous deleted."}%
k18@MacBook-Pro ~ % curl -X DELETE http://localhost:8080/users/Bill
{"description":"User Bill deleted."}%
k18@MacBook-Pro ~ % curl -X DELETE http://localhost:8080/users/Nick
{"description":"User Nick deleted."}%
k18@MacBook-Pro ~ %
```

Рисунок 4 – Снимок экрана терминала с удалением записей

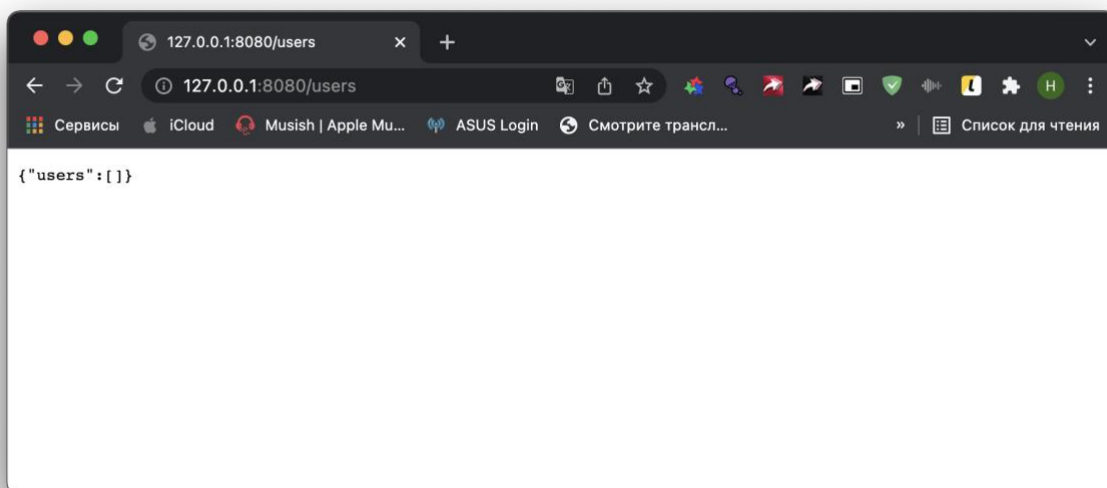


Рисунок 5 – Снимок экрана браузера Chrome с запущенным сервером после удаления записей

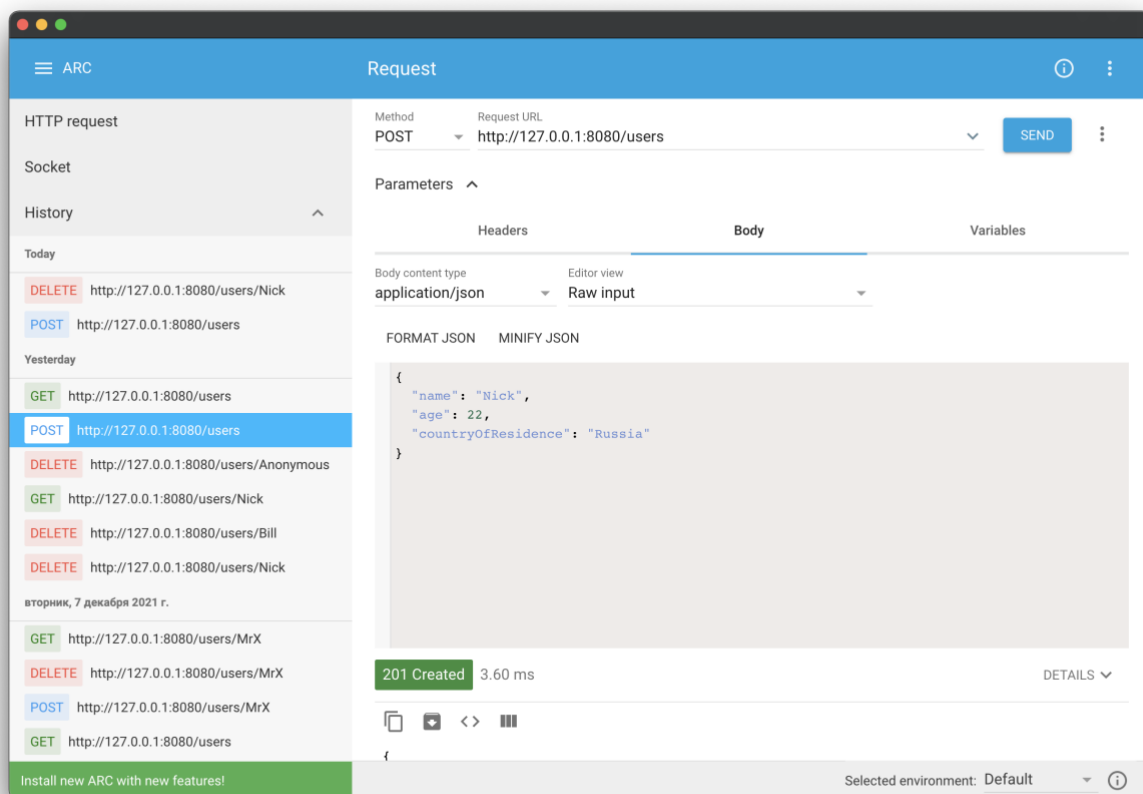


Рисунок 6 – Снимок экрана утилиты Advanced Rest Client с формированием POST запроса для добавления записей

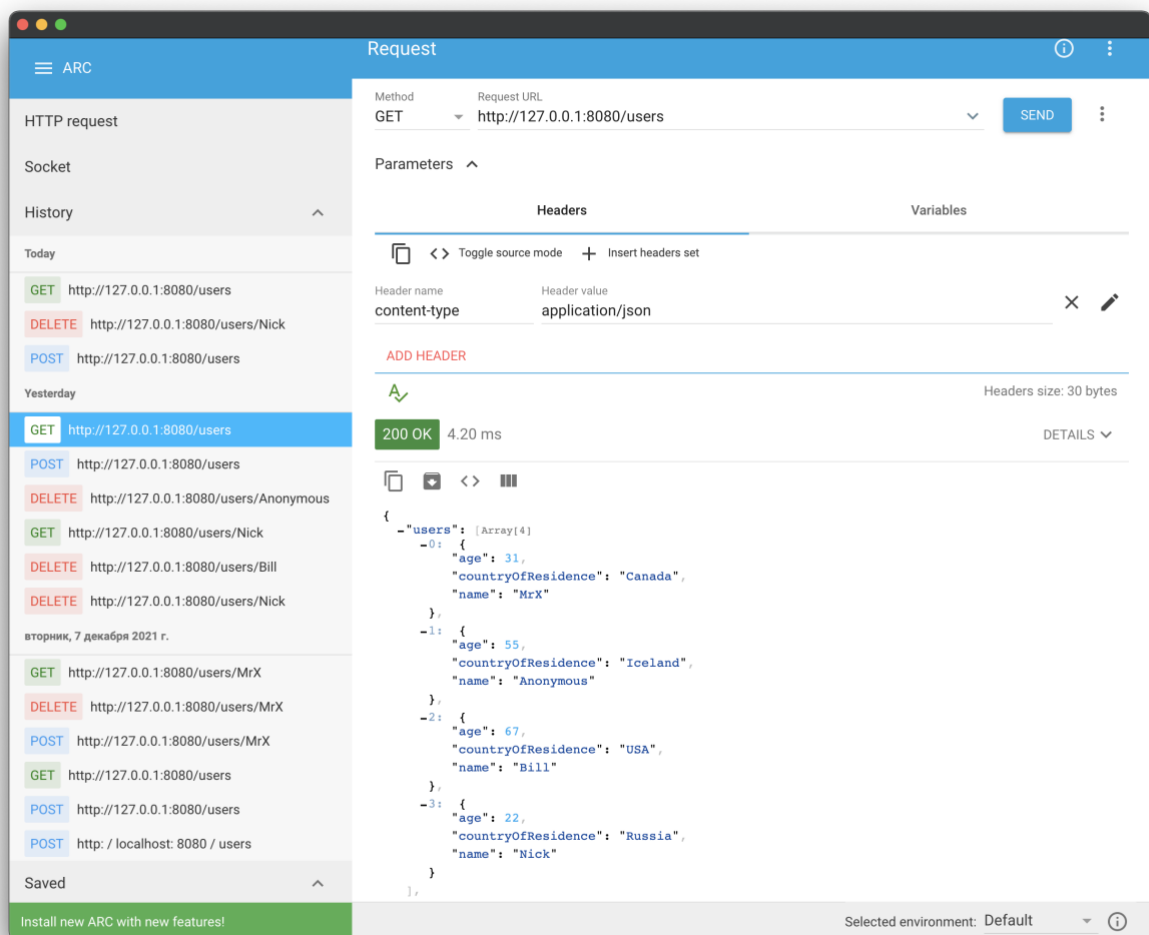


Рисунок 7 – Снимок экрана утилиты Advanced Rest Client с формированием GET запроса для получения записей



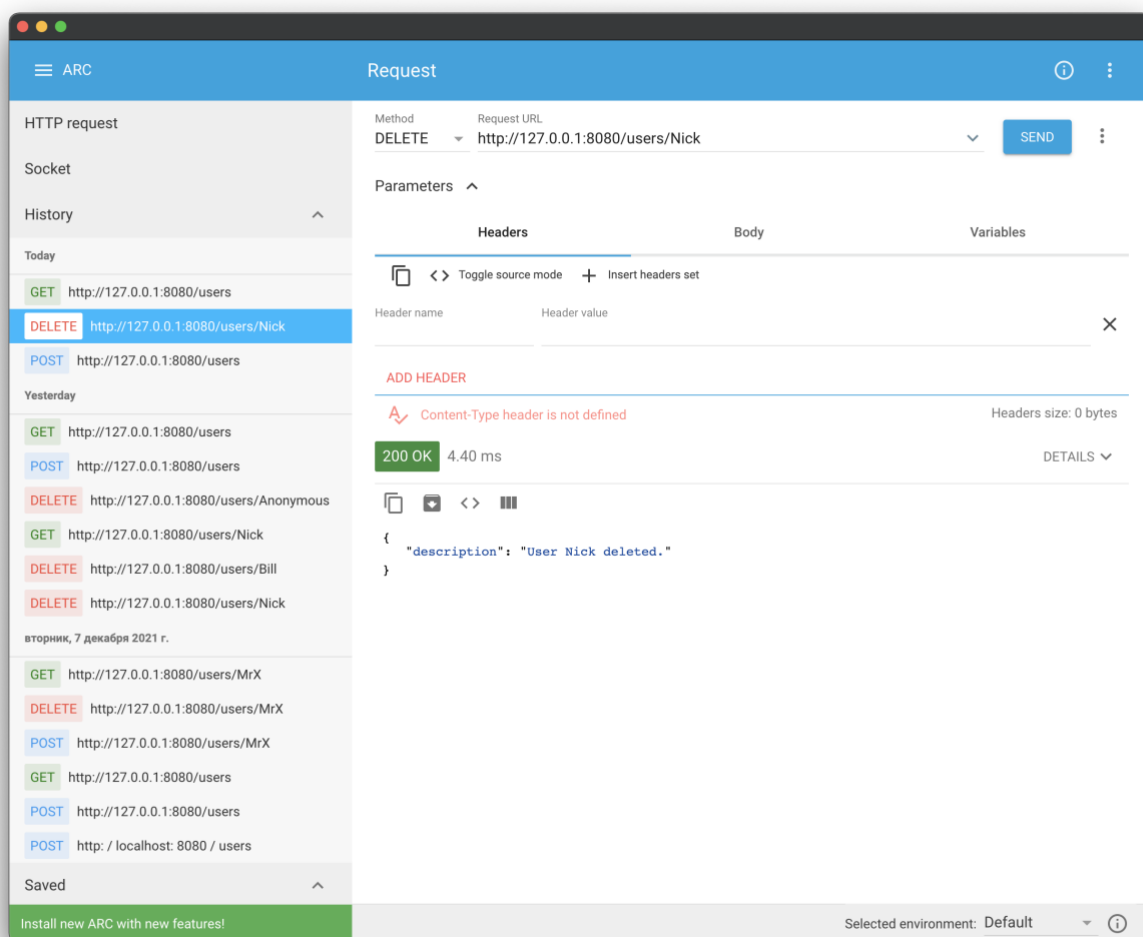


Рисунок 8 – Снимок экрана утилиты Advanced Rest Client с формированием DELETE запроса для удаления записей

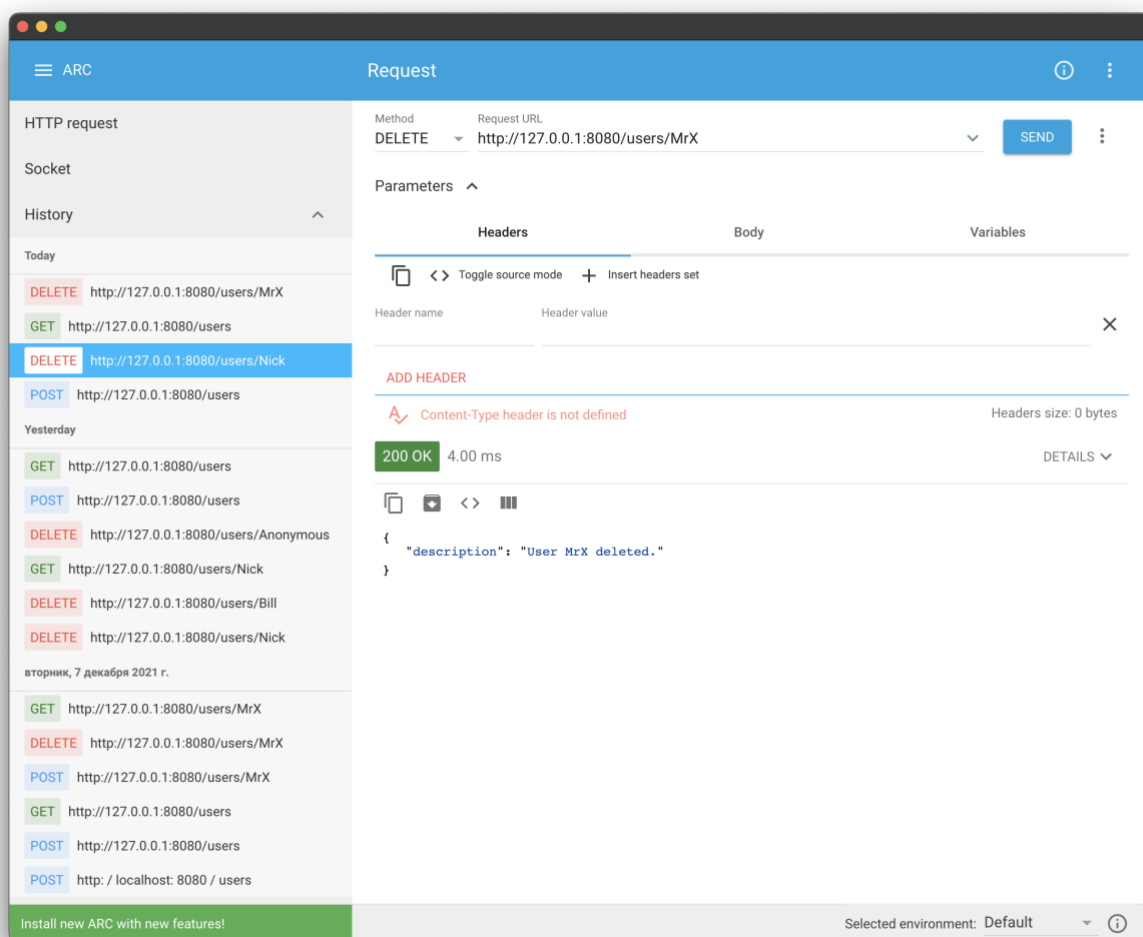


Рисунок 9 – Снимок экрана утилиты Advanced Rest Client с формированием DELETE запроса для удаления записей

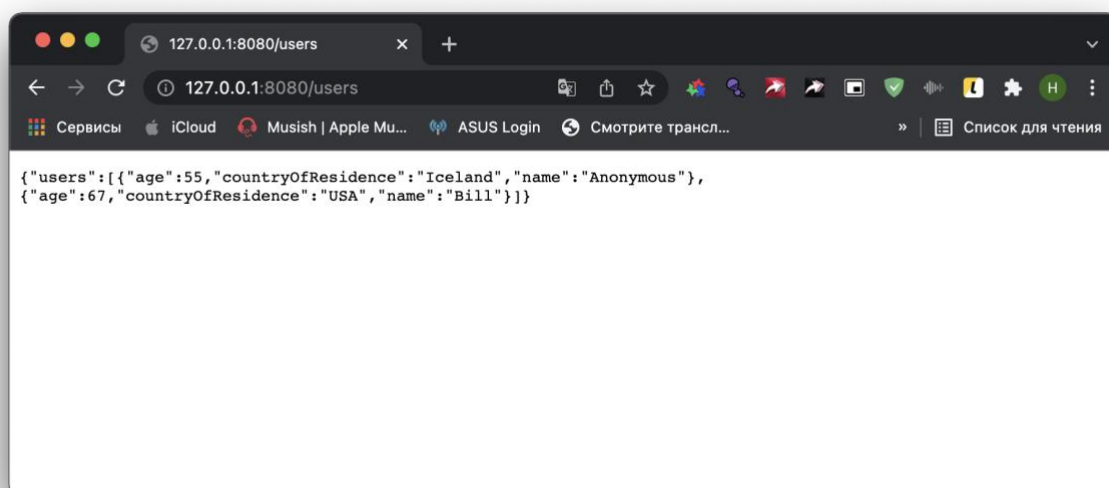


Рисунок 10 – Снимок экрана браузера Chrome с запущенным сервером после удаления записей

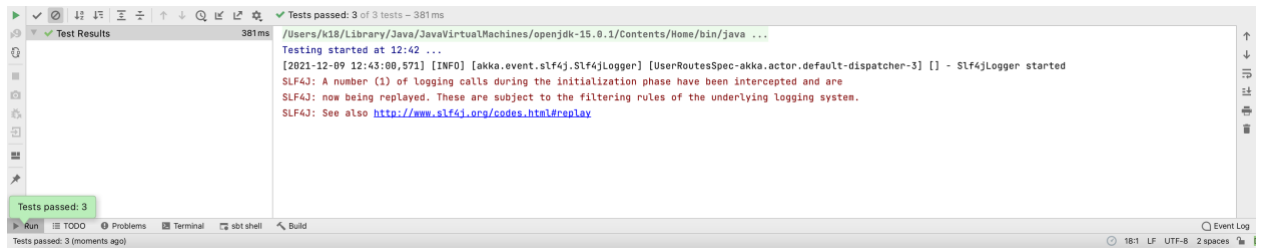


Рисунок 11 – Снимок экрана IntelliJ IDEA после выполнения модульного тестирования

## Исходный код программы

### Листинг 1 Исходный код JsonFormats

```
package com.example

import com.example.UserRegistry.ActionPerformed

//преобразует данные JSON из запросов в типы Scala и из
//типов Scala в ответы JSON

//#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)
}
//#json-formats
```

### Листинг 2 Исходный код QuickstartApp

```
package com.example

import akka.actor.typed.ActorSystem //для создания актора верхнего уровня
import akka.actor.typed.scaladsl.Behaviors //изменения поведения приема
сообщения
import akka.http.scaladsl.Http //для предоставления и использования служб
на основе HTTP
import akka.http.scaladsl.server.Route //маршрутизация

import scala.util.Failure
import scala.util.Success

//основной метод начальной загрузки приложения
//обработка, хранение и коммуникация

//#main-class
object fQuickstartApp {
  //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

```

### Листинг 3 Исходный код UserRegistry

```
package com.example
```

```

//#user-registry-actor
import akka.actor.typed.ActorRef //логический адрес актора, позволяющий
асинхронно отправлять актору сообщения
//асинхронный обмен сообщениями
import akka.actor.typed.Behavior //обработка сообщений
import akka.actor.typed.scaladsl.Behaviors //изменения поведения приема
сообщения
import scala.collection.immutable

//актор, обрабатывающий запросы на регистрацию
//

//#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

```

## Листинг 4 Исходный код UserRoutes

```

package com.example

import akka.http.scaladsl.server.Directives._ // для создания произвольно
сложных структур маршрутов
import akka.http.scaladsl.model.StatusCodes
import akka.http.scaladsl.server.Route //позволяет приложению отвечать на
входящие HTTP-запросы путем сопоставления запросов с ответами

import scala.concurrent.Future
import com.example.UserRegistry._
import akka.actor.typed.ActorRef //логический адрес актора, позволяющий
асинхронно отправлять актору сообщения
import akka.actor.typed.ActorSystem //для создания актора верхнего уровня
import akka.actor.typed.scaladsl.AskPattern._ //Шаблон запроса реализует
сторону инициатора протокола запрос-ответ.
import akka.util.Timeout

//HTTP-маршруты Akka, определяющие открытые эндпоинты.

//#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
          }
        )
      }
    )
  }
//#all-routes
}

```

## Листинг 5 Исходный код UserRoutesSpec

```
package com.example
```

```
//Модульное тестирование маршрутов
```

```
//#user-routes-spec
```

```
//#test-top
```

```
import akka.actor.testkit.typed.scaladsl.ActorTestKit //для асинхронного
тестирования типизированных актеров, предназначенный для подмешивания в
тестовый класс.
```

```

import akka.http.scaladsl.marshalling.Marshal //преобразования структуры
//более высокого уровня (объекта) в некое представление более низкого уровня,
//часто в
import akka.http.scaladsl.model._
import akka.http.scaladsl.testkit.ScalatestRouteTest //эффективное
//тестирование логики маршрутизации
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, "jap")
    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
}
//#set-up
//#user-routes-spec

```

## Вывод

В результате выполнения проекта было успешно запущено и протестировано HTTP-приложение Akka, изучено представление того, как маршруты упрощают обмен данными по HTTP.