

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

## "МИРЭА - Российский технологический университет"

РТУ МИРЭА

Институт информационных технологий (ИТ)

Кафедра математического обеспечения и стандартизации информационных технологий (МОСИТ)

# ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ №6

## по дисциплине

**«Разработка клиент-сервисных мобильных приложений»**

Выполнил студент группы ИКБО-11-22 Гришин А. В.

Приняла Исабекова О. А.

Москва 2025

## Практическая работа 6

В практических работах 4 и 5 были рассмотрены этапы создания клиентсерверного взаимодействия с помощью фреймворка Ktor и базы данных PosgreSQL.

В данной практической работе будет рассмотрен процесс создания сервера с помощью фреймворка Spring Boot, процесс установки базы данных MySQL, и ее подключение к серверу, а также процесс организации клиентсерверного взаимодействия.

В целом, процесс разработки клиент-серверного приложения на Ktor и Spring Boot проходит аналогично: настраивается сервер, подключается база данных, реализуются маршруты (endpoints) для обработки запросов, а затем обеспечивается взаимодействие с клиентской частью.

## Создание проекта на Spring Boot

Для того, что начать создавать сервер на Spring Boot, необходимо сформировать проект на сайте https://start.spring.io/.

Рассмотрим конфигурацию для создания нового проекта. На рисунке 1 представлена необходимая настройка для проекта к практической работе.

Project

Выбор системы сборки проекта:

• Gradle - Groovy: использование Gradle с языком написания скриптов Groovy.

• Gradle - Kotlin: использование Gradle с языком Kotlin для скриптов (более современный вариант).

• Maven: альтернативная система сборки, использующая XMLконфигурацию.

Language

Выбор языка программирования для проекта:

• Java: основной язык для Spring Boot.

• Kotlin: современный язык, полностью совместимый с JVM.

• Groovy: динамический язык, часто используемый для скриптов в Gradle. Spring Boot

Выбор версии Spring Boot:

* 3.5.0 (SNAPSHOT) / 3.5.0 (M3): превью-версии (нестабильные, для тестирования новых функций).
* 3.4.5 (SNAPSHOT) / 3.4.4: актуальная стабильная ветка (3.4.x).
* 3.3.11 (SNAPSHOT) / 3.3.10: предыдущая стабильная ветка (3.3.x), рекомендуется для production, если 3.4.x ещё не проверена. Project Metadata Основные метаданные проекта:

• Group: идентификатор группы пакетов (например, com.myproject).

• Artifact: название артефакта (библиотеки или приложения).

• Name: название проекта.

• Description: описание проекта.

• Package name: корневой пакет для исходного кода.

• Packaging:

* Jar: собирает проект в исполняемый JAR-файл (по умолчанию для Spring Boot).
* War: собирает проект как WAR-файл для развёртывания в сервлет- контейнере (например, Tomcat).
* Java: Выбор версии Java:
  + 24: Самая новая (возможно, экспериментальная).
  + 21: LTS-версия (долгосрочная поддержка).
  + 17: Предыдущая LTS (рекомендуется для большинства проектов).

В правой части экрана находится раздел, который описывает ключевые зависимости для Spring Boot-проекта, добавляемые в файл сборки (pom.xml для Maven или build.gradle для Gradle).

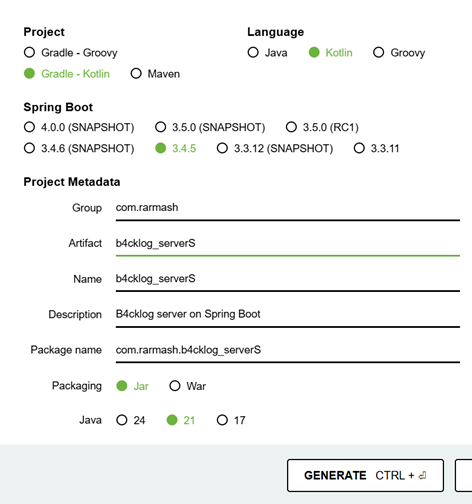


Рисунок 1 – Конфигурация создания нового проекта в Spring Boot

Рассмотрим назначение необходимых для создаваемого проекта зависимостей (рисунок 2):

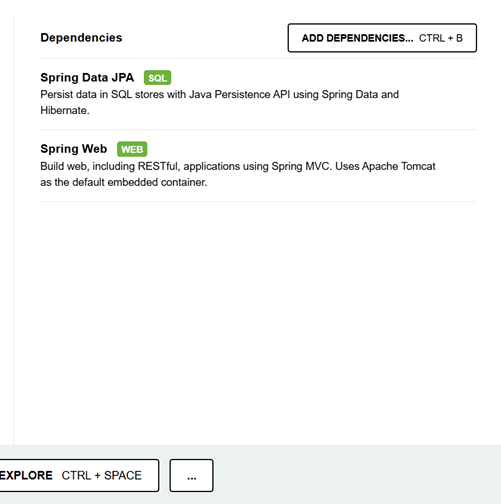
1. Spring Data JPA. Позволяет работать с базами данных через Java Persistence API (JPA). Подходит для MySQL, PostgreSQL, H2 и других реляционных СУБД.
2. Spring Web. Создает RESTful API или MVC-приложения с помощью Spring MVC. Включает встроенный сервер Tomcat (по умолчанию). Поддерживает аннотации для маршрутизации (@RestController, @GetMapping и т.д.).

Рисунок 2 – Зависимости, добавленные в проект

После окончания настройки проекта его необходимо скачать (архив) и распаковать. На рисунке 3 изображен созданный проект в SDK Intellij IDEA.

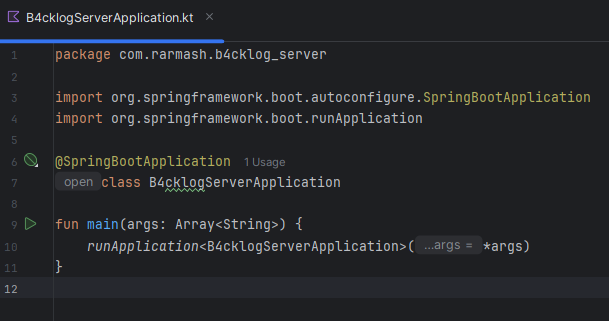


Рисунок 3 – Скачанный проект Intellij IDEA

## Настройка базы данных

Как уже было сказано ранее, в данной практической работе будет использоваться база данных MySQL. Для работы с данной БД, необходимо скачать MySQL Workbench.

MySQL Workbench — это официальный графический инструмент для работы с базами данных MySQL. Он предоставляет интерфейс для администрирования, проектирования, разработки и обслуживания баз данных. Далее необходимо скачать MySQL Server – сервер базы данных MySQL,

без него MySQL Workbench или приложение Spring Boot не смогут работать с БД — им просто некуда подключаться. Основные особенности установки MySQL Server:

1. Необходимо выбрать продукт для установки - MySQL Server 8.0.41 –

X64;

1. На рисунке 5 представлен экран MySQL Installer, на котором

выполняется настройка параметров сервера MySQL перед его установкой. Разберём каждый раздел:

* 1. Server Configuration Type (Тип конфигурации сервера). Выбор профиля в зависимости от назначения сервера:
     1. Development Computer (для разработки): оптимален для локальной работы, использует умеренное количество ресурсов (CPU, RAM).
     2. Server Computer (для продакшена): максимальная производительность.
     3. Dedicated Computer (выделенный сервер): все ресурсы машины выделены под MySQL.
  2. Connectivity (Подключение). Настройка доступа к серверу:
     1. TCP/IP: основной протокол для подключений, порт по умолчанию: 3306 (для классического MySQL-протокола), дополнительный порт: 33060 (для нового X-протокола, используется для NoSQL-функций MySQL).
     2. Open Windows Firewall ports: разрешает удалённые подключения к серверу (если снять галочку, сервер будет доступен только локально).
     3. Named Pipe и Shared Memory: альтернативные методы подключения для Windows (редко используются).
  3. Advanced Configuration (Дополнительные настройки).

Если включить галочку Show Advanced and Logging Options, появятся дополнительные вкладки для тонкой настройки: настройки кодировки (character set), пути хранения данных и логов, параметры репликации и безопасности.

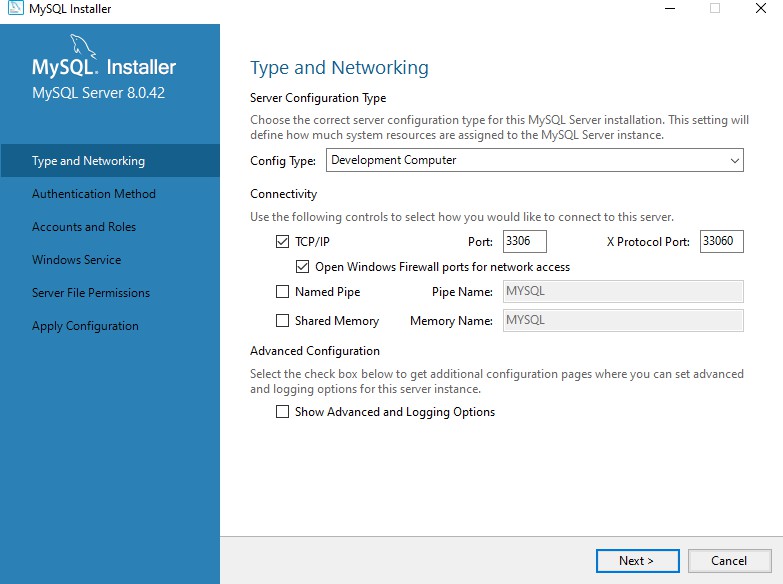


Рисунок 4 – Настройка параметров сервера

1. Далее необходимо выбрать метод аутентификации. Для MySQL 8.0+ рекомендуется Strong Password Encryption.
2. Необходимо установить пароль для пользователя root. Его необходимо запомнить!!! При необходимости можно добавить дополнительных пользователей.
3. На следующем экране происходит настройка службы, по умолчанию ее название MySQL80. Также необходимо поставить галочку для автозапуска сервера при загрузке системы.

После установки MySQL Server необходимо вернуться в MySQL Workbench, и установить соединение с сервером. Для этого необходимо на главном экране в разделе MySQL Connections нажать «+». После этого откроется окно для настройки нового соединения (рисунок 6). Рассмотрим необходимые настройки:

1. Connection Name: уникальное имя подключения (для удобства, например, по названию проекта).
2. Connection Method: способ подключения: через сетевой протокол TCP/IP (основной вариант для MySQL).
3. Параметры сервера:
   1. Hostname: 127.0.0.1 (localhost) – IP-адрес сервера, означает, что сервер MySQL установлен на этом же компьютере.
   2. Port: 3306 – порт по умолчанию для MySQL.
   3. Username: root – имя пользователя с полными правами администратора.
   4. Password. Необходимо нажать на кнопку Store in Vault…, в открывшемся окне ввести пароль пользователя, заданного при установке MySQL Server. Пароль сохраняется в зашифрованном хранилище Workbench (чтобы не вводить его каждый раз)

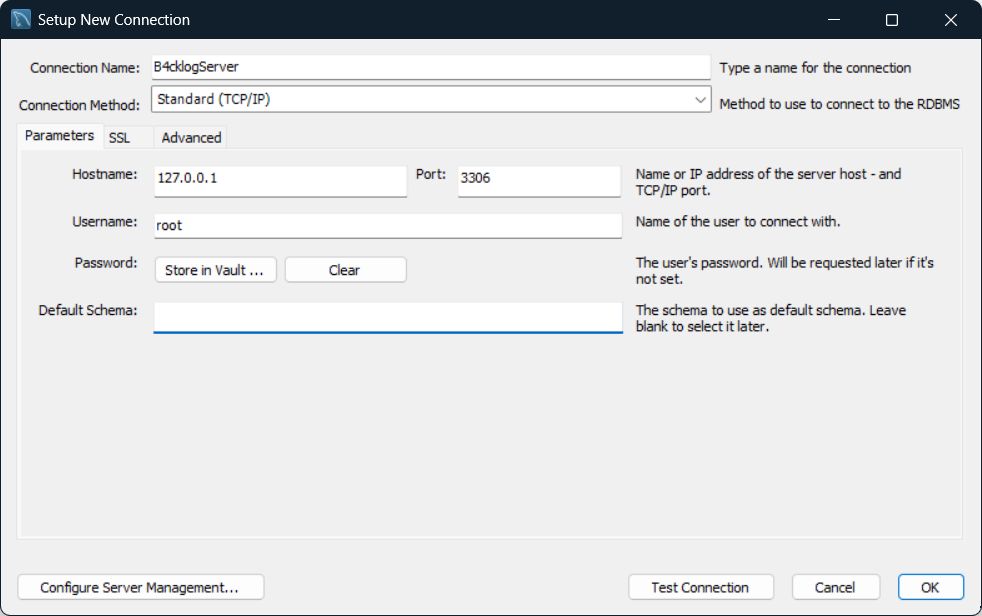


Рисунок 5 – Окно настройки нового соединения

После окончания настроки необходимо протестировать соединение, если установлено все верно, то будет получено сообщение, о том, что соединение успешно установлено, и данное соединение появится в списке на главном экране MySQL Workbench (рисунок 7, 8).

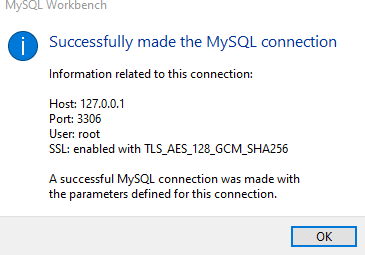


Рисунок 6 – Успешное соединение с сервером

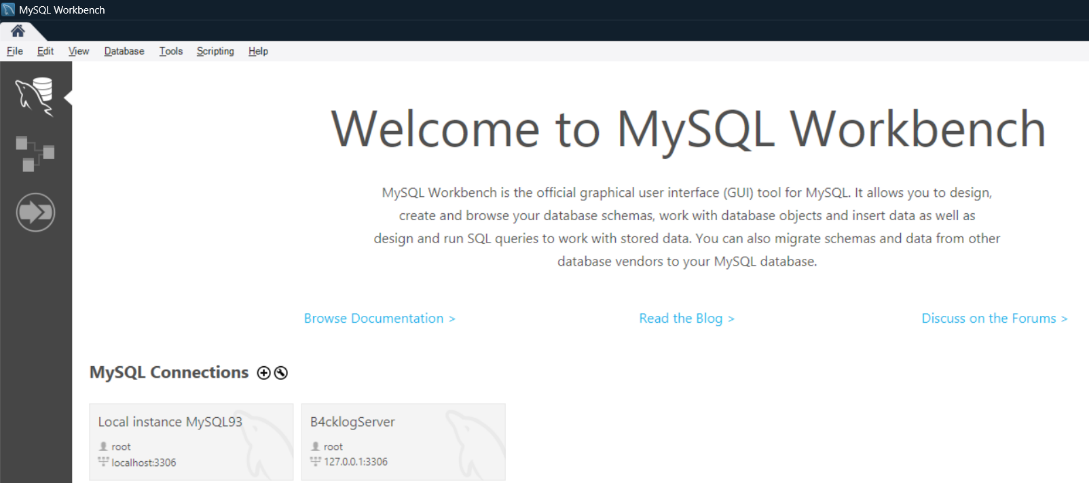


Рисунок 7 – Установленное подключение в MySQL Workbench

## Подключение БД к проекту Spring Boot

Для подключения БД к проекту Spring Boot необходимо в файле application.properties указать параметры БД, заданные ранее (рисунок 8).

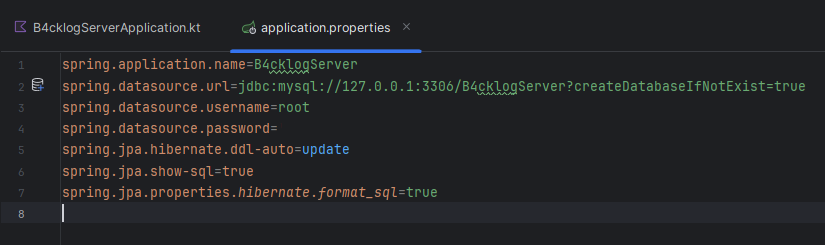


Рисунок 8 – Файл application.properties

Исходя из заданных настроек, при запуске сервера будет автоматически создана база данных (рисунок 9).

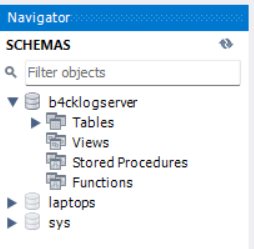


Рисунок 9 – Автоматическая созданная БД при первом запуске сервера

## Создание логики по работе с БД

После подключения к базе данных необходимо описать модели будущих таблиц. В данной практической работе необходимо создать одну таблицу Employee, в которой будут содержаться информация о сотрудниках (ID, имя, местоположение, отделение). На рисунке 12 представлен листинг класса Employee.

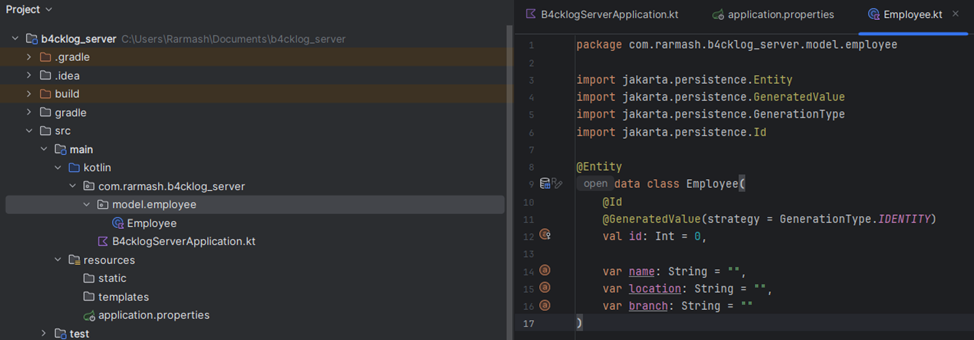


Рисунок 10 – Класс Employee Рассмотрим данный класс:

1. Аннотация @Entity помечает класс как JPA-сущность, которая будет отображена на таблицу БД.
2. Аннотация @Id помечает поле как первичный ключ.
3. Аннотация @GeneratedValue(strategy = GenerationType.IDENTITY) - автоинкремент (MySQL, PostgreSQL). БД автоматически генерирует ID.
4. Далее объявлены поля будущей таблицы employee.

Далее необходимо в класс SpringServerApplication добавить аннотацию @EnableAutoConfiguration, которая автоматически настраивает контекст приложения на основе зависимостей в вашем проекте.

Теперь проверим работоспособность сервера. Если все настроено верно, то в базе данных появится таблица, описанная в классе Employee (рисунок 13).

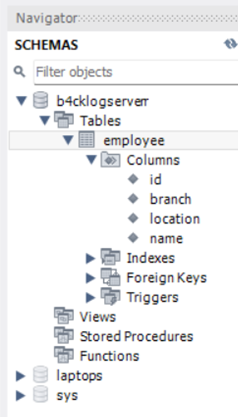


Рисунок 11 – Обновленная БД

На рисунке 12 представлено описание полей созданной таблицы employee.

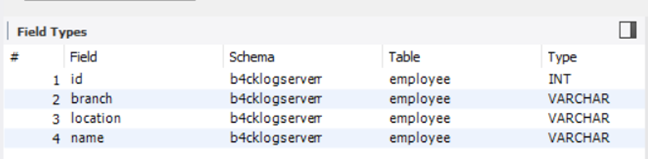


Рисунок 12 – Описание полей таблицы employee

Далее необходимо создать интерфейс EmployeeRepository для работы с сущностью Employee (рисунок 13).

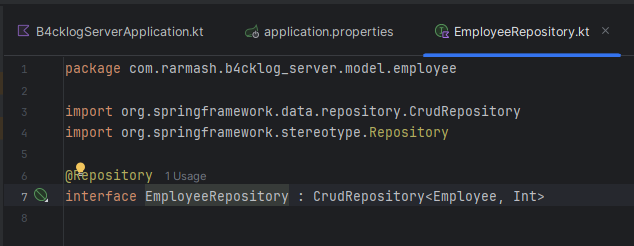


Рисунок 13 – Интерфейс EmployeeRepository

Рассмотрим интерфейс EmployeeRepository:

1. Аннотация @Repository помечает интерфейс как компонент Spring, отвечающий за доступ к данным.
2. CrudRepository — базовый интерфейс Spring Data для CRUD-операций (Create, Read, Update, Delete). Employee — тип сущности, с которой работает репозиторий. Int — тип первичного ключа (соответствует id: Int в классе Employee).

Далее необходимо создать класс EmployeeDao, который реализует сервисный слой приложения Spring Boot, работающий с репозиторием EmployeeRepository. На рисунке 16 представлен листинг данного класса:

1. Аннотация @Service помечает класс как Spring-компонент, содержащий бизнес-логику.
2. Метод save(employee: Employee) сохраняет или обновляет сущность Employee в БД.
3. Метод getAllEmployees() получает данные о всех сотрудниках из БД
4. Метод delete(employeeId: Int) удаляет сотрудника из БД по ID



Рисунок 14 – Класс EmployeeDao

Далее необходимо реализовать REST-контроллер Spring Boot для управления сущностями Employee (рисунок 15).

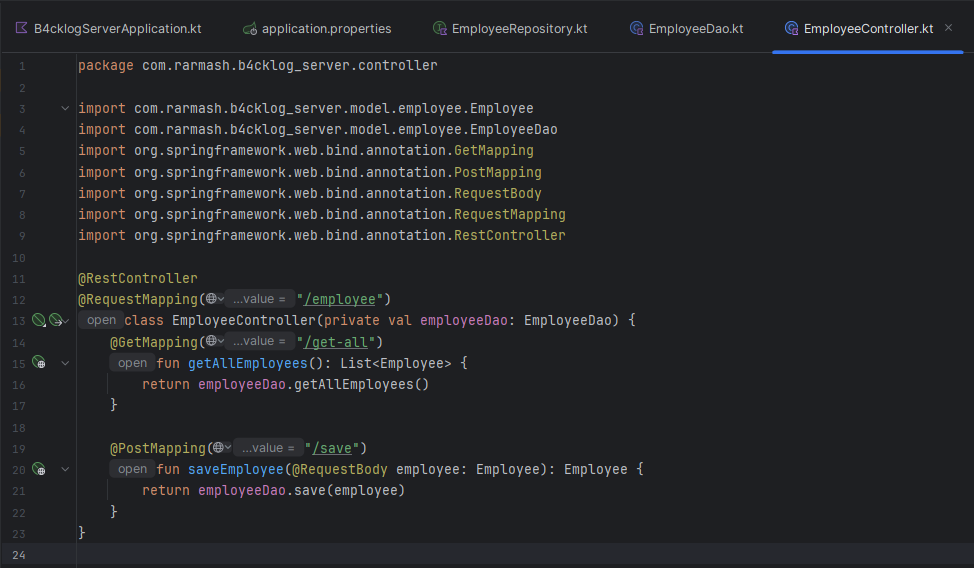


Рисунок 15 – Класс EmployeeController Работоспособность сервера при помощи Postman, представлена на рисунке 16.

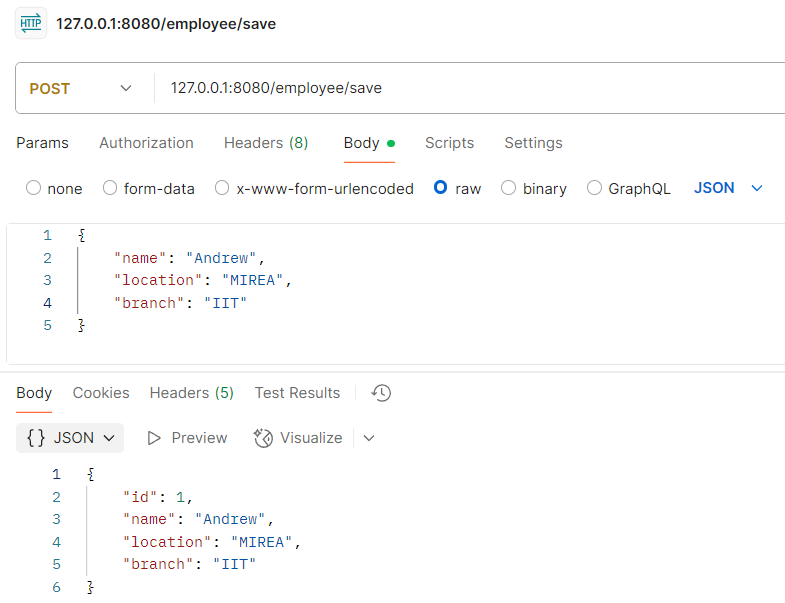


Рисунок 16 – Работоспособность сервера

## Создание клиентской части и ее взаимодействия с сервером

Во-первых, необходимо создать data-класс Employee, который будет представлять собой модель данных для сущности Employee (рисунок 18).

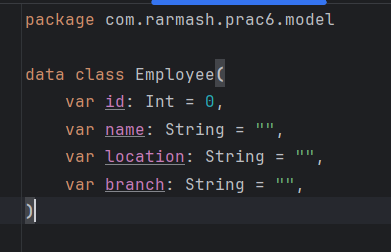


Рисунок 17 – Класс Employee

Далее необходимо создать все необходимые класс для работы библиотеки retrofit (поместите их в отдельный пакет). Данные классы представлены на рисунках 18 – 19.



Рисунок 18 – Класс RetrofitService

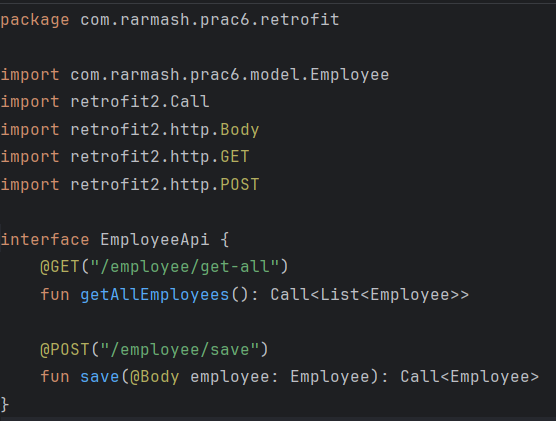


Рисунок 19 – Класс EmpoyeeApi

Далее создадим разметку необходимых экранов. В данном случае необходимо создать разметку для основного экрана, на котором будет расположен список сотрудников, разметку отдельного элемента списка и экрана, на котором можно будет внести нового сотрудника в БД. Созданные экраны отображены на рисунках 20 - 22.

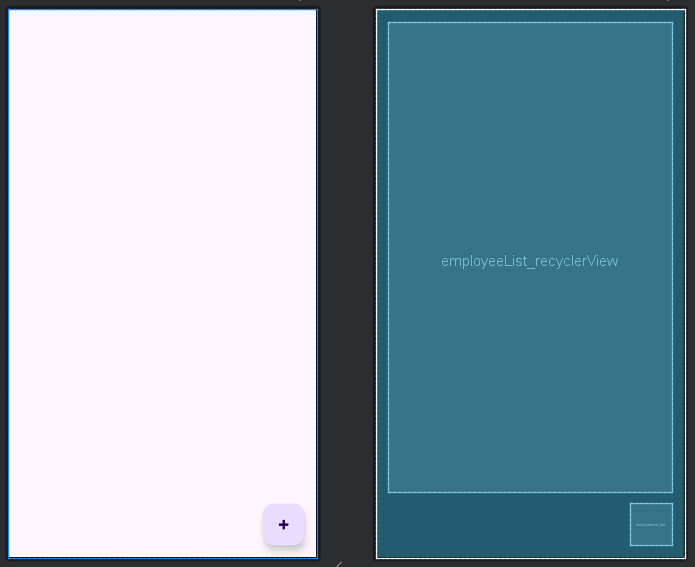


Рисунок 20 – Файл activity\_employee\_list

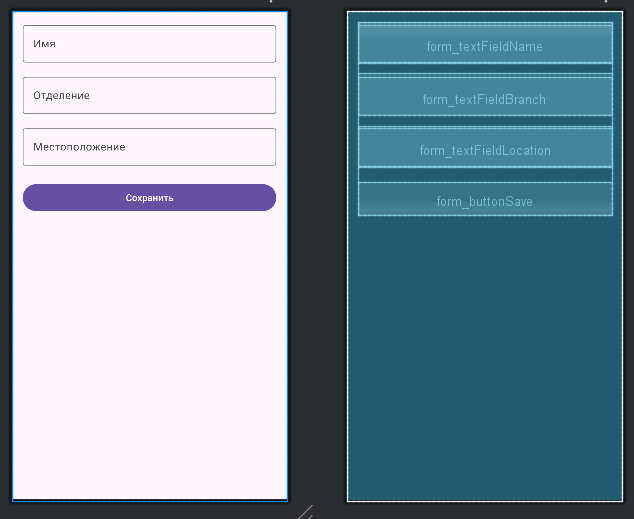


Рисунок 21 – Файл activity\_main

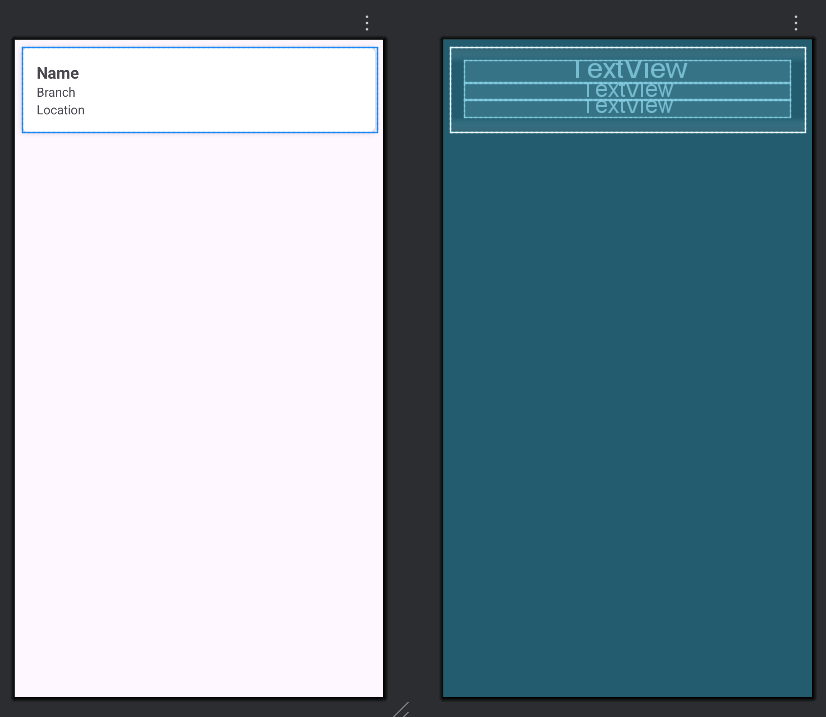


Рисунок 22 – Файл list\_employee\_item

Далее создадим класс EmployeeHolder и адаптер к нему для корректного отображения списка сотрудников (рисунок 23).

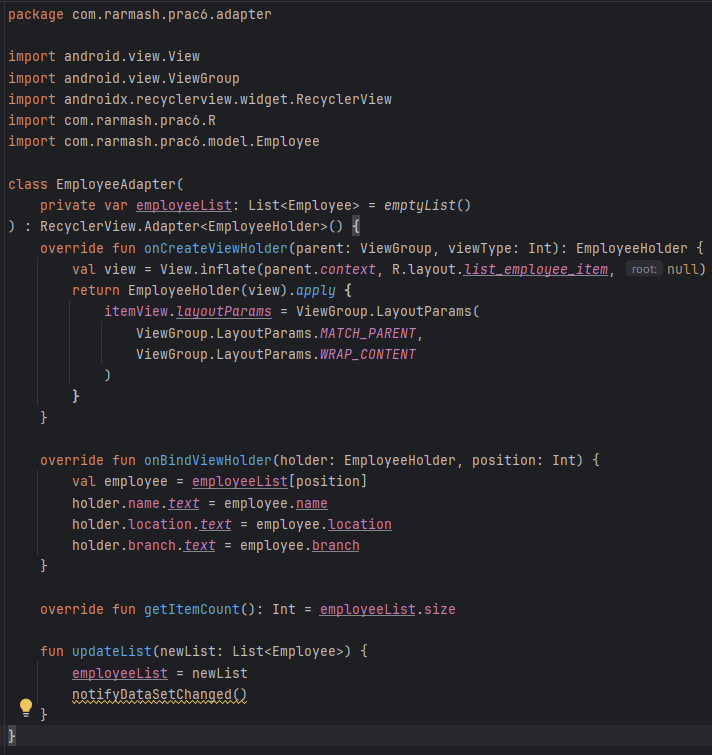


Рисунок 23 – Класс EmployeeAdapter

Далее необходимо создать класс (рисунок 24), который представляет собой Activity отображающее список сотрудников с возможностью ю добавления новых через Floating Action Button (FAB).

Рисунок 24 – Класс EmployeeListActivity

Далее необходимо создать класс (рисунок 25), представляющий собой Activity для формы добавления/редактирования сотрудников в Android- приложении, которое взаимодействует с Spring Boot REST API.

Рисунок 25 – Класс EmployeeForm

1. **Проверка работоспособности клиент-серверного взаимодействия** Теперь необходимо запустить мобильное приложение. При запуске приложения на главном экране будет отображаться список сотрудников,

внесенных в базу данных (рисунок 26).



Рисунок 26 – Список сотрудников при запуске приложения

При нажатии на кнопку происходит переход на экран с формой для добавления нового сотрудника в базу данных (рисунок 27).

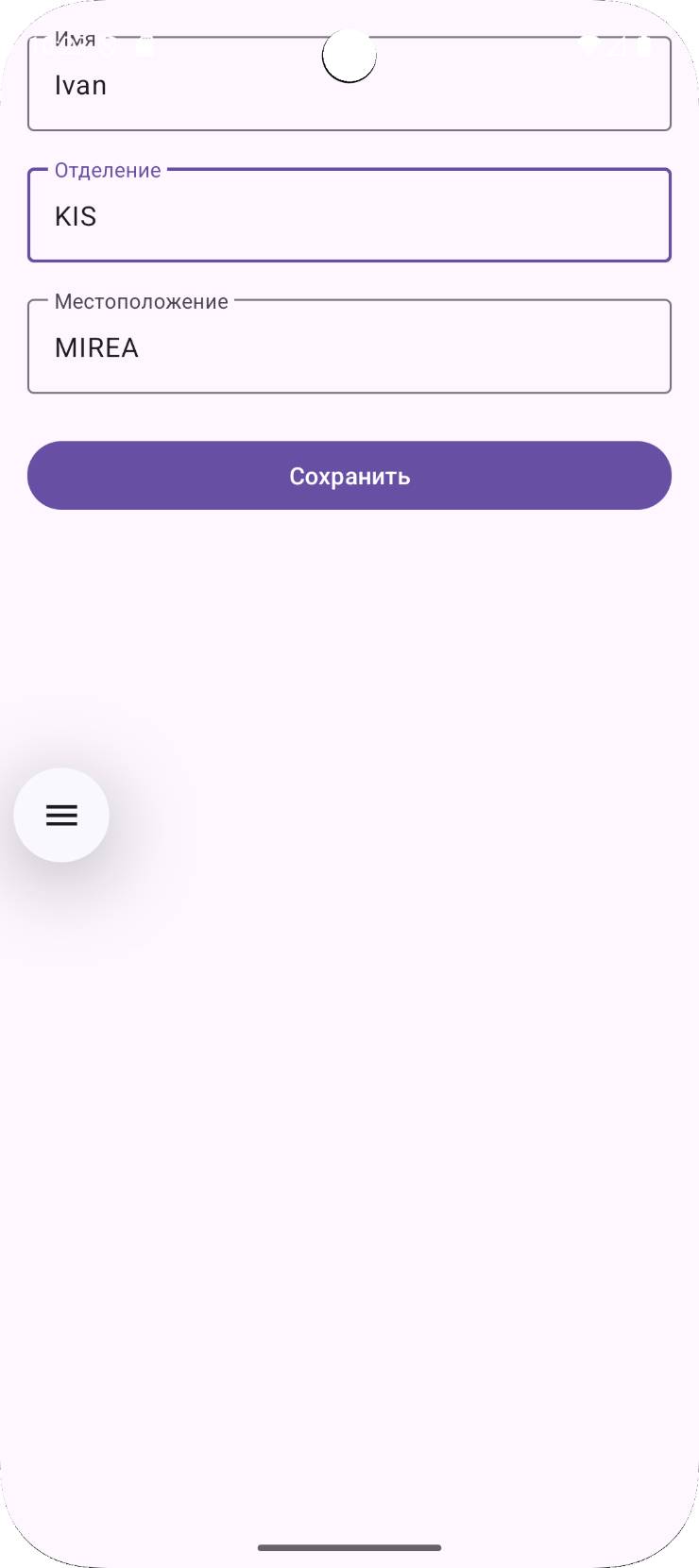


Рисунок 27 – Экран добавление нового сотрудника в БД

При возврате на главный экран, данные о новом сотруднике должны появиться в списке (рисунок 29). На рисунке 30 отображено изменение состояние БД.



Рисунок 29 – Обновленный список сотрудников на главном экране

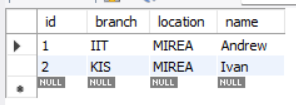


Рисунок 30 – Изменение состояние БД после добавления нового сотрудника

# ВЫВОД

В ходе практической работы было успешно разработано клиент- серверное приложение на Spring Boot с интеграцией MySQL, включая создание REST API, настройку подключения к БД, реализацию CRUD- операций через JPA-репозитории и организацию взаимодействия с клиентской частью через Retrofit, что подтвердило работоспособность выбранного стека технологий для построения полноценных backend-решений.

# СПИСОК ЛИТЕРАТУРЫ

1. Сомон., П. И. Волшебство Kotlin: руководство / П. И. Сомон.; перевод с английского А. Н. Киселева.. — Москва: ДМК Пресс, 2020. — 536 с. — ISBN 978-5-97060-801-2. — Текст: электронный // Лань: электронно- библиотечная система. — URL: https://e.lanbook.com/book/140599 (дата обращения: 15.02.2024). — Режим доступа: для авториз. пользователей.
2. Калгина, И. С. Разработка мобильных приложений: учебное пособие / И. С. Калгина. — Чита: ЗабГУ, 2022. — 163 с. — ISBN 978-5-9293- 3137- 4. — Текст: электронный // Лань: электронно-библиотечная система. — URL: https://e.lanbook.com/book/363323 (дата обращения: 15.02.2024). — Режим доступа: для авториз. пользователей.
3. Алпатов, А. Н. Архитектура, проектирование и разработка программных средств: учебное пособие / А. Н. Алпатов, И. Е. Рогов. — Москва: РТУ МИРЭА, 2023. — 120 с. — ISBN 978-5-7339-1972-0. — Текст: электронный // Лань: электронно-библиотечная система. — URL: https://e.lanbook.com/book/386189 (дата обращения: 15.02.2024). — Режим доступа: для авториз. пользователей.

**Приложение**

Листинг 1 – EmployeeListActivity

|  |
| --- |
| package com.rarmash.prac6  import android.os.Bundle import android.content.Intent import android.widget.Toast import androidx.appcompat.app.AppCompatActivity import androidx.recyclerview.widget.RecyclerView import androidx.recyclerview.widget.LinearLayoutManager import com.google.android.material.floatingactionbutton.FloatingActionButton import com.rarmash.prac6.adapter.EmployeeAdapter import com.rarmash.prac6.model.Employee import com.rarmash.prac6.retrofit.EmployeeApi import com.rarmash.prac6.retrofit.RetrofitService import retrofit2.Call import retrofit2.Callback import retrofit2.Response  class EmployeeListActivity : AppCompatActivity() {  private lateinit var recyclerView: RecyclerView  private lateinit var adapter: EmployeeAdapter  override fun onCreate(savedInstanceState: Bundle?) {  super.onCreate(savedInstanceState)  setContentView(R.layout.*activity\_employee\_list*)  initViews()  setupRecyclerView()  }  //Инициализация элементов экрана  private fun initViews() {  recyclerView = findViewById(R.id.*employeeList\_recyclerView*)  val fab: FloatingActionButton = findViewById(R.id.*employeeList\_fab*)  fab.setOnClickListener **{** startActivity(Intent(this, EmployeeForm::class.*java*))  **}** }  //Работа с RecyclerView  private fun setupRecyclerView() {  recyclerView.*layoutManager* = LinearLayoutManager(this)  adapter = EmployeeAdapter(*emptyList*())  recyclerView.*adapter* = adapter  }  //Загрузка данных через Retrofit  private fun loadEmployees() {  val retrofitService = RetrofitService()  val employeeApi = retrofitService.getRetrofit().create(EmployeeApi::class.*java*)   employeeApi.getAllEmployees().enqueue(object : Callback<List<Employee>> {  override fun onResponse(call: Call<List<Employee>>, response: Response<List<Employee>>)  {  response.body()?.*let* **{** employees **->** adapter.updateList(employees)  **}** }  override fun onFailure(call: Call<List<Employee>>, t: Throwable) {  Toast.makeText(  this@EmployeeListActivity,  "Failed to load employees: ${t.message}",  Toast.*LENGTH\_SHORT* ).show()  }  })  }  //Вызывается при каждом открытии Activity, обновляет список сотрудников  override fun onResume() {  super.onResume()  loadEmployees()  } } |

Листинг 2 – EmployeeForm

|  |
| --- |
| package com.rarmash.prac6  import android.os.Bundle import android.widget.Toast import androidx.appcompat.app.AppCompatActivity import com.google.android.material.button.MaterialButton import com.google.android.material.textfield.TextInputEditText import com.rarmash.prac6.model.Employee import com.rarmash.prac6.retrofit.EmployeeApi import com.rarmash.prac6.retrofit.RetrofitService import retrofit2.Call import retrofit2.Callback import retrofit2.Response import java.util.logging.Level import java.util.logging.Logger  class EmployeeForm : AppCompatActivity() {  private lateinit var inputEditTextName: TextInputEditText  private lateinit var inputEditBranch: TextInputEditText  private lateinit var inputEditLocation: TextInputEditText  private lateinit var buttonSave: MaterialButton  override fun onCreate(savedInstanceState: Bundle?) {  super.onCreate(savedInstanceState)  setContentView(R.layout.*activity\_main*)  initializeViews()  setupListeners()  }  //Инициализация элементов экрана  private fun initializeViews() {  inputEditTextName = findViewById(R.id.*form\_textFieldName*)  inputEditBranch = findViewById(R.id.*form\_textFieldBranch*)  inputEditLocation = findViewById(R.id.*form\_textFieldLocation*)  buttonSave = findViewById(R.id.*form\_buttonSave*)  }  //Обработка нажатия на кнопку  private fun setupListeners() {  val retrofitService = RetrofitService()  val employeeApi = retrofitService.getRetrofit().create(EmployeeApi::class.*java*)  buttonSave.setOnClickListener **{** val name = inputEditTextName.*text*?.toString() ?: ""  val branch = inputEditBranch.*text*?.toString() ?: ""  val location = inputEditLocation.*text*?.toString() ?: ""  //Проверка заполенностей полей  if (name.*isBlank*() || branch.*isBlank*() || location.*isBlank*()) {  Toast.makeText(this, "Please fill all fields", Toast.*LENGTH\_SHORT*).show()  return@setOnClickListener  }  //Создание объекта Employee  val employee = Employee(name = name, branch = branch, location = location)  employeeApi.save(employee).enqueue(object : Callback<Employee> {  override fun onResponse(call: Call<Employee>, response: Response<Employee>) {  if (response.*isSuccessful*) {  Toast.makeText(this@EmployeeForm, "Save successful!", Toast.*LENGTH\_SHORT*)  .show()  clearFields()  } else {  Toast.makeText(  this@EmployeeForm,  "Save failed: ${response.code()}",  Toast.*LENGTH\_SHORT* ).show()  }  }  override fun onFailure(call: Call<Employee>, t: Throwable) {  Toast.makeText(  this@EmployeeForm,  "Save failed: ${t.message}",  Toast.*LENGTH\_SHORT* ).show()  Logger.getLogger(EmployeeForm::class.*java*.*name*)  .log(Level.*SEVERE*, "Error occurred", t)  }  })  **}** }  //Очистка полей формы  private fun clearFields() {  inputEditTextName.*text*?.clear()  inputEditBranch.*text*?.clear()  inputEditLocation.*text*?.clear()  } } |

Листинг 3 – EmployeeAdapter

|  |
| --- |
| package com.rarmash.prac6.adapter  import android.view.View import android.view.ViewGroup import androidx.recyclerview.widget.RecyclerView import com.rarmash.prac6.R import com.rarmash.prac6.model.Employee  class EmployeeAdapter(  private var employeeList: List<Employee> = *emptyList*() ) : RecyclerView.Adapter<EmployeeHolder>() {  override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): EmployeeHolder {  val view = View.inflate(parent.*context*, R.layout.*list\_employee\_item*, null)  return EmployeeHolder(view).*apply* **{** itemView.*layoutParams* = ViewGroup.LayoutParams(  ViewGroup.LayoutParams.*MATCH\_PARENT*,  ViewGroup.LayoutParams.*WRAP\_CONTENT* )  **}** }   override fun onBindViewHolder(holder: EmployeeHolder, position: Int) {  val employee = employeeList[position]  holder.name.*text* = employee.name  holder.location.*text* = employee.location  holder.branch.*text* = employee.branch  }   override fun getItemCount(): Int = employeeList.size   fun updateList(newList: List<Employee>) {  employeeList = newList  notifyDataSetChanged()  } } |

Листинг 4 – EmployeeHolder

|  |
| --- |
| package com.rarmash.prac6.adapter  import android.view.View import android.widget.TextView import androidx.recyclerview.widget.RecyclerView import com.rarmash.prac6.R  class EmployeeHolder(itemView: View): RecyclerView.ViewHolder(itemView) {  val name: TextView = itemView.findViewById(R.id.*employeeListItem\_name*)  val location: TextView = itemView.findViewById(R.id.*employeeListItem\_location*)  val branch: TextView = itemView.findViewById(R.id.*employeeListItem\_branch*) } |

Листинг 5 – Employee (для клиента)

|  |
| --- |
| package com.rarmash.prac6.model  data class Employee(  var id: Int = 0,  var name: String = "",  var location: String = "",  var branch: String = "", ) |

Листинг 6 – RetrofitService

|  |
| --- |
| package com.rarmash.prac6.retrofit  import com.google.gson.Gson import retrofit2.Retrofit import retrofit2.converter.gson.GsonConverterFactory  class RetrofitService {  private val retrofit: Retrofit = Retrofit.Builder()  .baseUrl("http://10.0.2.2:8080")  .addConverterFactory(GsonConverterFactory.create(Gson()))  .build()   fun getRetrofit() = retrofit } |

Листинг 7 – EmployeeApi

|  |
| --- |
| package com.rarmash.prac6.retrofit  import com.rarmash.prac6.model.Employee import retrofit2.Call import retrofit2.http.Body import retrofit2.http.GET import retrofit2.http.POST  interface EmployeeApi {  @GET("/employee/get-all")  fun getAllEmployees(): Call<List<Employee>>   @POST("/employee/save")  fun save(@Body employee: Employee): Call<Employee> } |

Листинг 8 – EmployeeRepository

|  |
| --- |
| package com.rarmash.b4cklog\_server.model.employee  import org.springframework.data.repository.CrudRepository import org.springframework.stereotype.Repository  @Repository interface EmployeeRepository : CrudRepository<Employee, Int> |

Листинг 9 – EmployeeController

|  |
| --- |
| package com.rarmash.b4cklog\_server.controller  import com.rarmash.b4cklog\_server.model.employee.Employee import com.rarmash.b4cklog\_server.model.employee.EmployeeDao import org.springframework.web.bind.annotation.GetMapping import org.springframework.web.bind.annotation.PostMapping import org.springframework.web.bind.annotation.RequestBody import org.springframework.web.bind.annotation.RequestMapping import org.springframework.web.bind.annotation.RestController  @RestController @RequestMapping("/employee") class EmployeeController(private val employeeDao: EmployeeDao) {  @GetMapping("/get-all")  fun getAllEmployees(): List<Employee> {  return employeeDao.getAllEmployees()  }   @PostMapping("/save")  fun saveEmployee(@RequestBody employee: Employee): Employee {  return employeeDao.save(employee)  } } |

Листинг 10 – EmployeeDao

|  |
| --- |
| package com.rarmash.b4cklog\_server.model.employee  import org.springframework.stereotype.Service  @Service class EmployeeDao(  private val repository: EmployeeRepository ) {  fun save(employee: Employee) = repository.save(employee)   fun getAllEmployees(): List<Employee> = repository.findAll().*toList*()   fun delete(employeeId: Int) = repository.deleteById(employeeId) } |

Листинг 11 – Employee (для сервера)

|  |
| --- |
| package com.rarmash.b4cklog\_server.model.employee  import jakarta.persistence.Entity import jakarta.persistence.GeneratedValue import jakarta.persistence.GenerationType import jakarta.persistence.Id  @Entity data class Employee(  @Id  @GeneratedValue(strategy = GenerationType.*IDENTITY*)  val id: Int = 0,   var name: String = "",  var location: String = "",  var branch: String = "" ) |