|  |  |
| --- | --- |
| ГУАП  КАФЕДРА № 43  ОТЧЕТ  ЗАЩИЩЕН С ОЦЕНКОЙ  ПРЕПОДАВАТЕЛЬ |  |
|  |  |
| ассистент | Кочин Д.А. |

подпись, дата

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №2

Разработка ресурса REST/JSON сервиса

по дисциплине: ТЕХНОЛОГИИ РАЗРАБОТКИ СЕРВЕРНЫХ ИНФОРМАЦИОННЫХ СИСТЕМ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. № 4936 Нестеренко М.Ю.

подпись, дата фамилия, инициалы

Санкт-Петербург 2021

**Лабораторная работа №2**

Разработка ресурса REST/JSON сервиса

**Цель работы:**   
Разработать сервер-сервис, обеспечивающий взаимодействие со сторонними клиентскими приложениями посредством REST с использованием для передачи информации формата данных JSON.  
  
**Задание на лабораторную работу:**

1. Подключите к проекту Maven фреймворк Spring (spring-boot)
2. Определите перечень Rest-сервисов, выполняющих те же действия, что и в лабораторной работе 1. Внимательно отнеситесь к вопросу какой HTTP метод использует тот или иной сервис и какие коды HTTP он может возвращать. Реализуйте эти сервисы.

Вариант №15  
Коллекционирование (нумизматика, филателия и пр)

**Ход работы:**

1. Создадим проект на основе Ktor (аналог Spring) с помощью среды разработки. В разработке использовался язык программирования Kotlin и сборщик Gradle.
2. Создадим базу данных с помощью Exposed (H2) на основе разработанной сущности в прошлой лабораторной работе. Все дальнейшее взаимодействие с информацией будет происходить через базу данных.
3. Создадим все основные функции необходимые для работы Rest-сервиса: добавление, получение, обновление и удаление данных (CRUD) из базы.
4. Для каждой функции определим путь (начиная с localhost/api/collections) и статусы, которые эта функция может вернуть при разных запросах:  
   - GET на список коллекций – OK(200) в случае возврата списка и No Content(204) в случае отсутствиях данных в базе.

- GET на один элемент – Bad Request(400) в случае некорректного запроса, Not Found(404) в случае, если такой элемент не был найден в базе, и OK(200) при успешном выполнении запроса.  
- POST – Bad Request(400), если запрос некорректен, и Created(201) в случае успешного выполнение добавления.  
- PUT – Bad Request(400), если запрос был некорректный или такого элемента нет в базе, и OK(200) при успешном обновлении состояния выбранного элемента.

- DELETE - Bad Request(400), если запрос был некорректный, Not Found(404), если такого элемента нет в базе, и Accepted(202) при принятии запроса на обработку удаления (или просто начале процесса удаления).

Полный код программы находится в Приложении 1.

1. Данные на сервер(POST, PUT) и с него (GET) передаются в текстовом формате JSON.
2. Проверим работы программы с помощью http запросов в Postman:

Просмотр данных (GET <http://127.0.0.1:8080/api/collections>):

Status: 200 OK

[

{

"id": 4,

"name": "the first collection",

"type": "Postage Stamp",

"description": "simple rare",

"cost": 1

},

{

"id": 5,

"name": "second collection",

"type": "Coin",

"description": "very rare",

"cost": 2

},

{

"id": 6,

"name": "third collection",

"type": "Postage Stamp",

"description": "not rare",

"cost": 3

}

]

Просмотр одного элемента (GET http://127.0.0.1:8080/api/collections/4):

Status: 200 OK

{

"id": 4,

"name": "the first collection",

"type": "Postage Stamp",

"description": "simple rare",

"cost": 1

}

Добавление данных (POST http://127.0.0.1:8080/api/collections):

Запрос:

Headers: Content-Type: application/json  
Body:   
{

"id": 1,

"name": "ayaya name",

"type": "not cool type",

"description": "very good description",

"cost": 111

}

Ответ:  
Status: 201 Created

Body: The new collection was created

Обновление данных (PUT http://127.0.0.1:8080/api/collections/4):

Запрос:

Headers: Content-Type: application/json

Body:

{

"id": 1,

"name": "long name",

"type": "cool type",

"description": "very bad description",

"cost": 615

}

Ответ:  
Status: 200 OK

Body: The collection is saved

Удаление данных (DELETE http://127.0.0.1:8080/api/collections/5):

Status: 202 Accepted

Body: The collection was deleted

Просмотр результирующих данных (GET http://127.0.0.1:8080/api/collections):

Status: 200 OK  
Body:

[

{

"id": 4,

"name": "long name",

"type": "cool type",

"description": "very bad description",

"cost": 615

},

{

"id": 6,

"name": "third collection",

"type": "Postage Stamp",

"description": "not rare",

"cost": 3

},

{

"id": 7,

"name": "ayaya name",

"type": "not cool type",

"description": "very good description",

"cost": 111

}

]

**Вывод:** в ходе лабораторной работы были получены основные навыки разработки REST сервисов с помощью фрейморков Spring и Ktor, изучен текстовый формат передачи данных JSON, а также закреплена работа базами данных. Детально разобраны коды состояния HTTP и применены на практике.

**Приложение 1.**

**Application.kt:**

package com.example  
  
import com.example.database.CollectionTable  
import com.example.plugins.configureHTTP  
import com.example.plugins.configureRouting  
import com.example.plugins.configureSerialization  
import io.ktor.application.\*  
import org.jetbrains.exposed.sql.Database  
import org.jetbrains.exposed.sql.SchemaUtils  
import org.jetbrains.exposed.sql.deleteAll  
import org.jetbrains.exposed.sql.transactions.transaction  
  
  
  
fun main(args: Array<String>): Unit = io.ktor.server.netty.EngineMain.main(args)  
  
@Suppress("unused")  
fun Application.module() {  
 Database.connect("jdbc:h2:./Collections", "org.h2.Driver")  
 *transaction* {  
 SchemaUtils.create(CollectionTable)  
 CollectionTable.*deleteAll*()  
 }  
  
 *configureRouting*()  
 *configureHTTP*()  
 *configureSerialization*()  
}

**ApiRoutes.kt:**

package com.example.routes  
  
import com.example.database.CollectionDao  
import com.example.models.Collection  
import io.ktor.application.\*  
import io.ktor.http.\*  
import io.ktor.request.\*  
import io.ktor.response.\*  
import io.ktor.routing.\*  
  
fun Route.apiRouting() {  
  
 val dao = CollectionDao()  
  
 route ("/api/collections") {  
  
 get {  
 val collections = dao.getAll()  
 if (collections.*isNotEmpty*()) *call*.respond(HttpStatusCode.OK, collections)  
 else *call*.respondText("List of collections empty", status = HttpStatusCode.NoContent)  
 }  
  
 get("{id}") {  
 val id = *call*.parameters["id"] ?: return@get *call*.respondText(  
 "Missing or malformed id",  
 status = HttpStatusCode.BadRequest  
 )  
 try { id.*toInt*() } catch (e: Exception) { return@get *call*.respondText(  
 "Wrong id: $id",  
 status = HttpStatusCode.BadRequest)  
 }  
 val collection = dao.get(id.*toInt*()) ?: return@get *call*.respondText(  
 "No collection with id $id",  
 status = HttpStatusCode.NotFound  
 )  
 *call*.respond(status = HttpStatusCode.OK, collection)  
 }  
  
 post {  
 val collection = *call*.receive<Collection>()  
 if (collection.cost < 0 ||  
 collection.description.*isEmpty*() ||  
 collection.name.*isEmpty*() ||  
 collection.type.*isEmpty*())  
 return@post *call*.respondText(  
 "The data is incorrect",  
 status = HttpStatusCode.BadRequest  
 )  
 dao.add(collection)  
 *call*.respondText("The new collection was created", status = HttpStatusCode.Created)  
 }  
  
 delete("{id}") {  
 val id = *call*.parameters["id"] ?: return@delete *call*.respondText(  
 "Missing or malformed id",  
 status = HttpStatusCode.BadRequest  
 )  
 try { id.*toInt*() } catch (e: Exception) { return@delete *call*.respondText(  
 "Wrong id: $id",  
 status = HttpStatusCode.BadRequest)  
 }  
 if (dao.delete(id.*toInt*())) *call*.respondText("The collection was deleted", status = HttpStatusCode.Accepted)  
 else *call*.respondText("The collection does not exist", status = HttpStatusCode.NotFound)  
 }  
  
 put("{id}") {  
 val id = *call*.parameters["id"] ?: return@put *call*.respondText(  
 "Missing or malformed id",  
 status = HttpStatusCode.BadRequest  
 )  
 try { id.*toInt*() } catch (e: Exception) { return@put *call*.respondText(  
 "Wrong id: $id",  
 status = HttpStatusCode.BadRequest)  
 }  
 if (dao.update(id.*toInt*(), *call*.receive()))  
 *call*.respondText("The collection is saved", status = HttpStatusCode.OK)  
 else  
 *call*.respondText("The collection does not exist", status = HttpStatusCode.BadRequest)  
 }  
 }  
}

**HTTP.kt:**

package com.example.plugins  
  
import io.ktor.http.\*  
import io.ktor.features.\*  
import io.ktor.application.\*  
import io.ktor.response.\*  
import io.ktor.request.\*  
  
fun Application.configureHTTP() {  
 *install*(CORS) {  
 method(HttpMethod.Options)  
 method(HttpMethod.Put)  
 method(HttpMethod.Post)  
 method(HttpMethod.Get)  
 method(HttpMethod.Delete)  
 method(HttpMethod.Patch)  
 //header(HttpHeaders.Authorization)  
 header(HttpHeaders.ContentType)  
 allowCredentials = true  
 anyHost()  
 }  
  
}

**Routing.kt:**

package com.example.plugins  
  
import com.example.routes.apiRouting  
import io.ktor.routing.\*  
import io.ktor.http.\*  
import io.ktor.application.\*  
import io.ktor.response.\*  
import io.ktor.request.\*  
  
fun Application.configureRouting() {  
  
 routing {  
 apiRouting()  
 }  
}

**Serialization.kt:**

package com.example.plugins  
  
import io.ktor.serialization.\*  
import io.ktor.features.\*  
import io.ktor.application.\*  
import io.ktor.response.\*  
import io.ktor.request.\*  
import io.ktor.routing.\*  
  
fun Application.configureSerialization() {  
 install(ContentNegotiation) {  
 json()  
 }  
}

**Collection.kt:**

package com.example.models  
  
import kotlinx.serialization.Serializable  
  
@Serializable  
data class Collection(  
  
 val id: Int? = null,  
 val name: String,  
 val type: String,  
 val description: String,  
 val cost: Int  
  
)

**CollectionTable.kt:**

package com.example.database  
  
import com.example.models.Collection  
import org.jetbrains.exposed.sql.ResultRow  
import org.jetbrains.exposed.sql.Table  
  
  
object CollectionTable: Table() {  
  
 val id = integer("id").autoIncrement().uniqueIndex()  
 val name = varchar("name", 32)  
 val type = varchar("type", 32)  
 val description = varchar("description", 255)  
 val cost = integer("double")  
  
  
 override val primaryKey = PrimaryKey(id, name = "PK\_Collection\_Id")  
  
 fun toCollection(row: ResultRow): Collection =  
 Collection(  
 id = row[id],  
 name = row[name],  
 type = row[type],  
 description = row[description],  
 cost = row[cost]  
 )  
}

**CollectionDao.kt:**

package com.example.database  
  
import com.example.models.Collection  
import org.jetbrains.exposed.sql.\*  
import org.jetbrains.exposed.sql.transactions.experimental.newSuspendedTransaction  
  
class CollectionDao {  
  
 suspend fun getAll() = newSuspendedTransaction {  
 return@newSuspendedTransaction CollectionTable.*selectAll*().*map* { CollectionTable.toCollection(it) }  
 }  
  
 suspend fun get(id: Int) = newSuspendedTransaction {  
 val c = CollectionTable.*select* { CollectionTable.id *eq* id }.*map* { CollectionTable.toCollection(it) }  
 return@newSuspendedTransaction if (c.isEmpty()) null  
 else c.*first*()  
 }  
  
 suspend fun add(c: Collection) = newSuspendedTransaction {  
 CollectionTable.*insert* {  
 it[name] = c.name  
 it[type] = c.type  
 it[description] = c.description  
 it[cost] = c.cost  
 }  
 return@newSuspendedTransaction true  
 }  
  
 suspend fun delete(id: Int) = newSuspendedTransaction {  
 val c = CollectionTable.*select* { CollectionTable.id *eq* id }.*map* { CollectionTable.toCollection(it) }  
 return@newSuspendedTransaction if (c.isEmpty()) false  
 else {  
 CollectionTable.*deleteWhere* { CollectionTable.id *eq* id }  
 true  
 }  
 }  
  
 suspend fun deleteAll() = newSuspendedTransaction {  
 return@newSuspendedTransaction CollectionTable.*deleteAll*()  
 }  
  
 suspend fun update(id: Int, newC: Collection) = newSuspendedTransaction {  
 get(id) ?: return@newSuspendedTransaction false  
 CollectionTable.*update*({ CollectionTable.id *eq* id }) {  
 it[name] = newC.name  
 it[type] = newC.type  
 it[description] = newC.description  
 it[cost] = newC.cost  
 }  
 return@newSuspendedTransaction true  
 }  
}

**Gradle (build):**

val ktor\_version: String by project  
val kotlin\_version: String by project  
val logback\_version: String by project  
val exposedVersion: String by project  
  
plugins {  
 application  
 kotlin("jvm") version "1.5.31"  
 id("org.jetbrains.kotlin.plugin.serialization") version "1.5.31"  
}  
  
group = "com.example"  
version = "0.0.1"  
application {  
 mainClass.set("com.example.ApplicationKt")  
}  
  
repositories {  
 mavenCentral()  
}  
  
dependencies {  
 // Ktor  
 implementation("io.ktor:ktor-server-core:$ktor\_version")  
 implementation("io.ktor:ktor-serialization:$ktor\_version")  
 implementation("io.ktor:ktor-server-netty:$ktor\_version")  
 implementation("ch.qos.logback:logback-classic:$logback\_version")  
  
 // Exposed  
 implementation ("org.jetbrains.exposed:exposed-core:$exposedVersion")  
 implementation ("org.jetbrains.exposed:exposed-dao:$exposedVersion")  
 implementation ("org.jetbrains.exposed:exposed-jdbc:$exposedVersion")  
 implementation ("com.h2database:h2:1.4.199")  
  
 testImplementation("io.ktor:ktor-server-tests:$ktor\_version")  
 testImplementation("org.jetbrains.kotlin:kotlin-test:$kotlin\_version")  
}