НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Факультет программной инженерии и компьютерной техники

Рефакторинг баз данных и приложений

Лабораторная работа № 2

Выполнил студент

Демичев Даниил Дмитриевич

Группа № P34112

Преподаватель: Логинов Иван Павлович

г. Санкт-Петербург

2021

Задание:

Дополнить реализацию таким образом, чтобы хранение данных осуществлялось в базе данных, реализовать поддержку сеанса и сохранения его состояния для разных пользователей. Как следствие, реализовать функциональность для регистрации пользователей и их авторизации.

Вариант 8 - Решатель уравнений

Программа предназначена для решения уравнений вида $aannxxnn + aann{-}1xxnn{-}1 + \cdots + aa1xx + aa0 = 0$
Пользователю предлагается ввести степень полинома, затем – ввести соответствующие коэффициенты.
Далее выполняется решение уравнения, вывод конечного уравнения и его решения(ий) в консоль.
Программа должна сохранять информацию о результатах вычислений и предоставлять возможность вывода этой информации по порядковому номеру, который присваивается решению по окончании вычислений.

Проект: https://github.com/Amur27RUS/Refactor_Lab2

Форк: https://github.com/xxPFFxx/Refactor_Lab2

Выполнение:

Для работы с базой данных в Kotlin я воспользовался ORM фреймворком Exposed. Для этого я создал две сущности: User, отвечающий за хранение информации о пользователях и Equation, содержащий сведения о введенных полиномах и связанным с User отношением многие-к-одному.

User.kt

```
class User(id: EntityID<Int>) : IntEntity(id) {
    companion object : IntEntityClass<User>(Users)
    var name     by Users.name
    var password by Users.password
    var maxEquationNumber by Users.maxEquationNumber
    val equations by Equation referrersOn Equations.user
}

object Users: IntIdTable() {
    val name = varchar( name: "name",  length: 50);
    val password = varchar( name: "password",  length: 200);
    val maxEquationNumber = integer( name: "maxEquationNumber");
}

fun findUserByUsername(username : String) =
    transaction {  this: Transaction
        User.find { Users.name eq username }.firstOrNull()
    }
```

Equation.kt

```kotlin
class Equation(id: EntityID<Int>) : IntEntity(id) {
    companion object : IntEntityClass<Equation>(Equations)
    var number     by Equations.number
    var equation   by Equations.equation
    var solution   by Equations.solution
    var user       by User referencedOn Equations.user
}

object Equations: IntIdTable() {
    val number = integer( name: "number");
    val equation = varchar( name: "equation",  length: 200);
    val solution = varchar( name: "solution",  length: 200);
    val user = reference( name: "user", Users);
}
```

К списку доступных команд я добавил register и login для осуществления регистрации и авторизации пользователей. Без авторизации пользователям недоступны команды, связанные с полиномами.

InfoMessages.kt

```kotlin
suspend fun sendAvailableCommandsLoggedIn(connection: Connection){
    connection.session.send( content: "Available commands:")
    connection.session.send( content: "show - to show solve history")
    connection.session.send( content: "find eqNum - to show equation w/ index eqNum")
    connection.session.send( content: "Enter polynomial's coeffs to start solving")
}

suspend fun sendAvailableCommandsNotLoggedIn(connection: Connection){
    connection.session.send( content: "Available commands:")
    connection.session.send( content: "register LOGIN PASSWORD - to register into system")
    connection.session.send( content: "login LOGIN PASSWORD - to login into system")
}
```

Данные об авторизации я храню в сессии. С помощью флага registered ограничивается доступ к полиномным командам, name позволяет делать запросы к БД, связанные с конкретным пользователем

Connection.kt

```kotlin
class Connection(val session: DefaultWebSocketSession) {
    companion object {
        var lastId = AtomicInteger( initialValue: 0)
    }
    var name = ""
    var registered = false
}
```

На команды пользователя вызываются соответствующие функции:

Register – Регистрирует нового пользователя, если такого никнейма еще не существует в системе

```
suspend fun registerCommand(params : Array<String>, thisConnection: Connection){
    if (params.size != 3) {
        thisConnection.session.send( content: "Bad format of register command, example: register pff 123")
        sendAvailableCommandsNotLoggedIn(thisConnection)
        return
    }
    val user = findUserByUsername(params[1]);
    if (user != null){
        thisConnection.session.send( content: "User with that username already exists")
        sendAvailableCommandsNotLoggedIn(thisConnection)
    }
    else{
        transaction { this: Transaction
            addLogger(StdOutSqlLogger)
            val user = User.new{ this: User
                name = params[1]
                password = BCrypt.hashpw(params[2], BCrypt.gensalt())
                maxEquationNumber = 0
            }
        }
        thisConnection.session.send( content: "You successfully registered")
        sendAvailableCommandsNotLoggedIn(thisConnection)
    }
}
```

Login – Позволяет пользователю войти в систему, если он указал верные никнейм и пароль, или сообщает ему об ошибке

```
suspend fun loginCommand(params : Array<String>, thisConnection: Connection){
    if (params.size != 3){
        thisConnection.session.send( content: "Bad format of login command, example: login pff 123")
        sendAvailableCommandsNotLoggedIn(thisConnection)
        return
    }
    val user = findUserByUsername(params[1]);
    if (user == null){
        thisConnection.session.send( content: "No user with that username")
        sendAvailableCommandsNotLoggedIn(thisConnection)
    }
    else {
        val password = transaction { user.password };
        if (BCrypt.checkpw(params[2], password)) {
            thisConnection.name = params[1];
            thisConnection.registered = true
            thisConnection.session.send( content: "You are logged in")
            sendAvailableCommandsLoggedIn(thisConnection)
        } else {
            thisConnection.session.send( content: "Incorrect password")
            sendAvailableCommandsNotLoggedIn(thisConnection)
        }
    }
}
```

Show – Отображает историю введенных полиномов для конкретного пользователя

```
suspend fun showCommand(thisConnection: Connection){
    if (thisConnection.registered) {
        val currentUser = findUserByUsername(thisConnection.name)
        val userEquations = transaction { currentUser!!.equations .toList()}

        if(userEquations.isEmpty()){
            thisConnection.session.send( content: "You have no saved equations here :(")
        }else {
            userEquations.forEachIndexed { index, equation ->
                val currentEquation = transaction { equation.equation }
                val currentSolution = transaction { equation.solution }
                thisConnection.session.send( content: "${index+1}. $currentEquation = 0, ${currentSolution}")
            }
        }
        sendAvailableCommandsLoggedIn(thisConnection)
    }
    else{
        thisConnection.session.send( content: "You are not logged in. Register first or enter a valid username and password to perform this operation")
        sendAvailableCommandsNotLoggedIn(thisConnection)
    }
}
```

Find – Выводит полином с указанным номером

```
suspend fun findCommand(params: Array<String>, thisConnection: Connection){
    if (thisConnection.registered){
        val currentUser = findUserByUsername(thisConnection.name)
        val userEquations = transaction { currentUser!!.equations .toList()}
        if(userEquations.isEmpty()){
            thisConnection.session.send( content: "You can't use 'find' command because your solving history is empty!")
        }else if(kotlin.math.abs(params[1].toInt()) > userEquations.size || params[1].toInt() < 1){
            thisConnection.session.send( content: "Please, enter a valid equation number!")
        }else{
            userEquations.forEachIndexed { index, equation ->
                if (equation.number == params[1].toInt())
                    thisConnection.session.send( content: "Selected equation: ${equation.equation}, ${equation.solution}")
            }
        }
        sendAvailableCommandsLoggedIn(thisConnection)
    }
    else{
        thisConnection.session.send( content: "You are not logged in. Register first or enter a valid username and password to perform this operation")
        sendAvailableCommandsNotLoggedIn(thisConnection)
    }
}
```

Solve – Решает полином с введенными коэффициентами, сохраняет в БД информацию о запросе

```
suspend fun solveCommand(params: Array<String>, thisConnection: Connection) {
    if (thisConnection.registered) {
        val intParams = params.map { it: String
            if (!isNumeric(it)){
                thisConnection.session.send( content: "Bad format of coeffs, the all should be integer values, separated with space (1 2 3)")
                sendAvailableCommandsLoggedIn(thisConnection)
                return
            }
            it.toInt()  ^map
        }.toTypedArray()
        val p1 = Polynomial(intParams)
        val currentUser = findUserByUsername(thisConnection.name)
```

```
        currentUser?.let { it: User
            transaction { this: Transaction
                currentUser.maxEquationNumber += 1
                Equation.new { this: Equation
                    number = currentUser.maxEquationNumber
                    equation = "Введённый полином: $p1 = 0"
                    solution = "Решение: x=${p1.solve()}"
                    user = currentUser
                } ^transaction
            }
        }
        thisConnection.session.send( content: "Введённый полином: $p1 = 0")
        thisConnection.session.send( content: "Решение: x=${p1.solve()}")
        sendAvailableCommandsLoggedIn(thisConnection)
    } else {
        thisConnection.session.send( content: "You are not logged in. Register first or enter a valid username and password to perform this operation")
        sendAvailableCommandsNotLoggedIn(thisConnection)
    }
```

Основной модуль, WebSocket, который слушает запросы и вызывает соответствующие функции:

Application.kt

```
fun main(args: Array<String>): Unit = io.ktor.server.netty.EngineMain.main(args)

@Suppress( ...names: "unused")
fun Application.module() {
    install(WebSockets)
    Database.connect( url: "jdbc:postgresql://localhost:5432/postgres", driver = "org.postgresql.Driver",
        user = "123", password = "123")
    transaction {     SchemaUtils.create (Users)
    SchemaUtils.create(Equations)}
    routing { this: Routing
        val connections = Collections.synchronizedSet<Connection?>(LinkedHashSet())
        webSocket( path: "/solver") { this: DefaultWebSocketServerSession
            println("Adding user!")
            val thisConnection = Connection( session: this)
            connections += thisConnection
            try {
                send( content: "You are connected! There are ${connections.count()} users here.")
                sendAvailableCommandsNotLoggedIn(thisConnection)
                for (frame in incoming) {
                    frame as? Frame.Text ?: continue
                    val receivedText = frame.readText()
                    val params = receivedText.split( ...delimiters: " ").toTypedArray()
```

```
                    when (params[0]){
                        "register" -> registerCommand(params, thisConnection)
                        "login" -> loginCommand(params, thisConnection)
                        "show" -> showCommand(thisConnection)
                        "find" -> findCommand(params, thisConnection)
                        else -> {
                            solveCommand(params, thisConnection)
                        }
                    }
                }
            }
            catch (e: Exception) {
                println(e.localizedMessage)
            }catch(e: NullPointerException){
                println("Your command is incorrect!")
            }
        }

}
```

Вывод:

В ходе выполнения этой лабораторной работы я разработал консольное приложение, добавил механизм регистрации и авторизации с хранением информации в базе данных для существующего приложения и познакомился с возможностями языка Kotlin для клиент-серверного взаимодействия.