Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский университет ИТМО»

Факультет программной инженерии и компьютерных технологий

Лабораторная работа №7

Вариант 213888

Выполнила:

Павличенко Софья Алексеевна, P3115

Проверил:

Вербовой Александр Александрович

Санкт-Петербург 2024г.

Оглавление

Задание

Доработать программу из лабораторной работы №6 следующим образом:

1. Организовать хранение коллекции в реляционной СУБД (PostgresQL). Убрать хранение коллекции в файле.
2. Для генерации поля id использовать средства базы данных (sequence).
3. Обновлять состояние коллекции в памяти только при успешном добавлении объекта в БД
4. Все команды получения данных должны работать с коллекцией в памяти, а не в БД
5. Организовать возможность регистрации и авторизации пользователей. У пользователя есть возможность указать пароль.
6. Пароли при хранении хэшировать алгоритмом MD2
7. Запретить выполнение команд не авторизованным пользователям.
8. При хранении объектов сохранять информацию о пользователе, который создал этот объект.
9. Пользователи должны иметь возможность просмотра всех объектов коллекции, но модифицировать могут только принадлежащие им.
10. Для идентификации пользователя отправлять логин и пароль с каждым запросом.

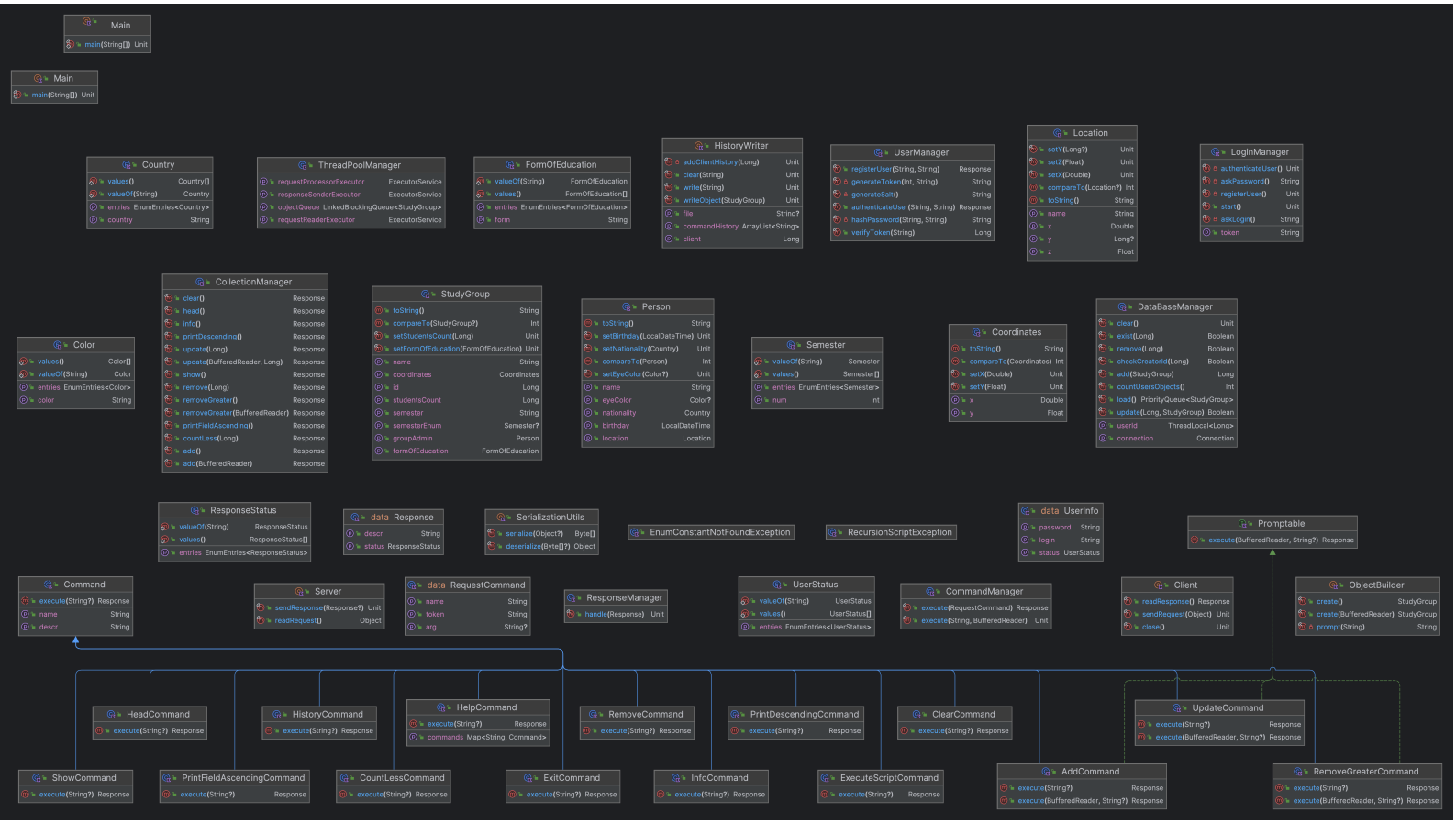
Необходимо реализовать многопоточную обработку запросов.

1. Для многопоточного чтения запросов использовать Fixed thread pool
2. Для многопоточной обработки полученного запроса использовать Fixed thread pool
3. Для многопоточной отправки ответа использовать Cached thread pool
4. Для синхронизации доступа к коллекции использовать синхронизацию чтения и записи с помощью java.util.concurrent.locks.ReentrantLock

Порядок выполнения работы:

1. В качестве базы данных использовать PostgreSQL.
2. Для подключения к БД на кафедральном сервере использовать хост pg, имя базы данных - studs, имя пользователя/пароль совпадают с таковыми для подключения к серверу.

Диаграмма классов реализованной объектной модели



Решение

Исходный код программы

client.Main.java

```
package client

import client.managers.LoginManager
import client.managers.ResponseManager
import client.network.Client
import common.communication.RequestCommand
import common.communication.Response
import common.communication.ResponseStatus
import java.util.*

/**
 * Основной класс приложения, отвечающий за запуск клиентской части
программы.
 */
object Main {
    @JvmStatic
    fun main(args: Array<String>) {
        val sc = Scanner(System.`in`)
        val responseManager = ResponseManager()
        val loginManager = LoginManager(responseManager)

        loginManager.start()

        while (sc.hasNext()) {
            val line = sc.nextLine().trim()
            if (line.isNotEmpty()) {
                val tokens = line.split(' ')
                var arg: String? = null
                if (tokens.size > 1) arg = tokens[1]
                val request = RequestCommand(tokens[0], arg,
loginManager.token)
                Client.sendRequest(request)
                var response: Response? = null
                while (response == null || (response.status !=
ResponseStatus.SUCCESS && response.status != ResponseStatus.ERROR)) {
                    response = Client.readResponse()
                    responseManager.handle(response)
                }
            }
        }
        sc.close()
        Client.close()
    }
}
```

client.managers.LoginManager

```
package client.managers


import client.network.Client
import common.communication.Response
import common.communication.ResponseStatus
import common.communication.UserInfo
```

```
import common.communication.UserStatus
import java.util.*

class LoginManager(private val responseManager: ResponseManager) {
    lateinit var token: String
    private val sc = Scanner(System.`in`)

    fun start() {
        println("Работа с коллекцией доступна только авторизованным
пользователям!")
        while (true) {
            println("Вы зарегистрированы?")
            when (sc.nextLine().lowercase().trim()) {
                in arrayOf("да", "yes", "д", "y") -> {
                    authenticateUser(); break
                }

                in arrayOf("нет", "no", "н", "n") -> {
                    registerUser(); break
                }
                else -> println("\u001B[31mОтвет не распознан. Попробуйте еще
раз!\u001B[0m")
            }
        }
    }


    private fun askLogin(): String {
        while (true) {
            print("Введите логин: ")
            val login = sc.nextLine()
            if (login.isEmpty()) println("\u001B[31mЛогин не может быть
пустым! Попробуйте ещё раз.\u001B[0m")
            else return login
        }
    }

    private fun askPassword(): String {
        while (true) {
            print("Введите пароль: ")
            val password = sc.nextLine()
            //скрыть!!!!!!!!!
            if (password.isEmpty()) println("\u001B[31mПароль не может быть
пустым! Попробуйте ещё раз.\u001B[0m")
            else return password
        }
    }

    private fun registerUser() {
        var response: Response? = null
        while (response == null || response.status != ResponseStatus.SUCCESS)
{
            Client.sendRequest(UserInfo(askLogin(), askPassword(),
UserStatus.REGISTRATION))
            response = Client.readResponse()
            responseManager.handle(response)
            if (response.status == ResponseStatus.SUCCESS) {
                token = Client.readResponse().descr
            } else println("Попробуйте ещё раз.")
        }
    }

    private fun authenticateUser() {
        var response: Response? = null
```

```
        while (response == null || response.status != ResponseStatus.SUCCESS)
{
            Client.sendRequest(UserInfo(askLogin(), askPassword(),
UserStatus.AUTHENTICATION))
            response = Client.readResponse()
            responseManager.handle(response)
            if (response.status == ResponseStatus.SUCCESS) {
                token = Client.readResponse().descr
            } else println("Попробуйте ещё раз.")
        }
    }
}
```

server.Main.java

```
package server

import common.communication.*
import common.models.StudyGroup
import server.managers.*
import server.network.Server
import server.utils.HistoryWriter
import java.util.concurrent.LinkedBlockingQueue


/**
 * Основной класс приложения, отвечающий за запуск серверной части программы.
 */
object Main {
    @JvmStatic
    fun main(args: Array<String>) {

        val requestQueue = LinkedBlockingQueue<Any>()

        val threadPoolManager = ThreadPoolManager()
        val dataBaseManager = DataBaseManager()
        val collectionManager = CollectionManager(dataBaseManager,
threadPoolManager)
        val userManager = UserManager(dataBaseManager.connection)
        val commandManager = CommandManager(collectionManager,
threadPoolManager)

        while (true) {
            threadPoolManager.requestReaderExecutor.submit {
                requestQueue.put(Server.readRequest())
            }
            val request = requestQueue.take()
            threadPoolManager.requestProcessorExecutor.submit {
                when (request) {
                    is RequestCommand -> {
                        val response: Response
                        val userId = userManager.verifyToken(request.token)
                        if (userId != 0L) {
                            dataBaseManager.userId.set(userId)
                            HistoryWriter.setClient(userId)
                            response = commandManager.execute(request)
                        } else response =
Response(ResponseStatus.INVALID_TOKEN, "Срок действия вашей сессии истек или
токен недействителен.")
                        threadPoolManager.responseSenderExecutor.submit {
                            Server.sendResponse(response)
```

```
                        }
                        dataBaseManager.userId.remove()
                    }

                    is UserInfo -> {
                        val response =
                            if (request.status == UserStatus.REGISTRATION)
                                userManager.registerUser(request.login,
request.password)
                            else userManager.authenticateUser(request.login,
request.password)
                        threadPoolManager.responseSenderExecutor.submit {
                            Server.sendResponse(response)
                        }
                    }

                    is StudyGroup -> {
                        threadPoolManager.objectQueue.put(request)
                    }
                }
            }
        }
    }
}
```

server.managers.DataBaseManager

```
package server.managers

import common.models.*
import java.io.FileInputStream
import java.sql.*
import java.time.ZoneOffset
import java.util.*

class DataBaseManager {
    var userId = ThreadLocal<Long>()
    val connection: Connection

    init {
        val info = Properties()
        info.load(FileInputStream("db.cfg"))
        try {
            connection =
DriverManager.getConnection("jdbc:postgresql://localhost:5432/studs", info)
        } catch (e: SQLException) {
            throw RuntimeException(e.message)
        }
    }

    fun load(): PriorityQueue<StudyGroup> {
        val collection = PriorityQueue<StudyGroup>()
        try {
            val resultStudyGroups = connection.prepareStatement("SELECT *
FROM study_groups;").executeQuery()
            while (resultStudyGroups.next()) {
                val studyGroup = StudyGroup()
                studyGroup.id = resultStudyGroups.getLong("id")
                studyGroup.name = resultStudyGroups.getString("name")

                val coordinates = Coordinates()
```

```
                val getCoordinates = connection.prepareStatement("SELECT *
FROM coordinates WHERE id = (?);")
                getCoordinates.setLong(1,
resultStudyGroups.getLong("coordinates"))
                val resultCoordinates = getCoordinates.executeQuery()
                if (resultCoordinates.next()) {
                    coordinates.x = resultCoordinates.getDouble("x")
                    coordinates.y = resultCoordinates.getFloat("y")
                } else throw SQLException()

                resultCoordinates.close()
                getCoordinates.close()

                studyGroup.coordinates = coordinates
                studyGroup.studentsCount =
resultStudyGroups.getLong("students_count")
                studyGroup.formOfEducation =
FormOfEducation.valueOf(resultStudyGroups.getString("form_of_education"))
                val semester = resultStudyGroups.getString("semester_enum")
                studyGroup.semesterEnum = if (resultStudyGroups.wasNull())
null else Semester.valueOf(semester)

                val admin = Person()
                val getAdmin = connection.prepareStatement("SELECT * FROM
persons WHERE id = (?);")
                getAdmin.setLong(1, resultStudyGroups.getLong("group_admin"))
                val resultAdmin = getAdmin.executeQuery()


                if (resultAdmin.next()) {
                    admin.name = resultAdmin.getString("name")
                    admin.birthday =
resultAdmin.getTimestamp("birthday").toLocalDateTime()
                    val eyeColor = resultAdmin.getString("eye_color")
                    admin.eyeColor = if (resultAdmin.wasNull()) null else
Color.valueOf(eyeColor)
                    admin.nationality =
Country.valueOf(resultAdmin.getString("nationality"))


                    val adminLocation = Location()
                    val getAdminLocation =
connection.prepareStatement("SELECT * FROM locations WHERE id = (?);")
                    getAdminLocation.setLong(1,
resultAdmin.getLong("location"))
                    val resultAdminLocation = getAdminLocation.executeQuery()
                    if (resultAdminLocation.next()) {
                        adminLocation.x = resultAdminLocation.getDouble("x")
                        val y = resultAdminLocation.getLong("y")
                        adminLocation.y = if (resultAdminLocation.wasNull())
null else y
                        adminLocation.z = resultAdminLocation.getFloat("z")
                        adminLocation.name =
resultAdminLocation.getString("name")

                        admin.location = adminLocation
                    } else throw SQLException()

                    resultAdminLocation.close()
                    getAdminLocation.close()

                } else throw SQLException()

                resultAdmin.close()
```

```
                getAdmin.close()

                studyGroup.groupAdmin = admin

                collection.add(studyGroup)
            }
            resultStudyGroups.close()
        } catch (e: SQLException) {
            println(e.message)
        }
        return collection
    }

    fun add(studyGroup: StudyGroup): Long {
        try {
            val insertStudyGroup =
                connection.prepareStatement("INSERT INTO study_groups (name,
coordinates, students_count, form_of_education, semester_enum, group_admin,
created_by) VALUES (?, ?, ?, ?::form_of_education, ?::semester, ?, ?);",
                    Statement.RETURN_GENERATED_KEYS)
            insertStudyGroup.setString(1, studyGroup.name)

            val insertCoordinates = connection.prepareStatement("INSERT INTO
coordinates (x, y) VALUES (?, ?);", Statement.RETURN_GENERATED_KEYS)
            insertCoordinates.setDouble(1, studyGroup.coordinates.x)
            insertCoordinates.setFloat(2, studyGroup.coordinates.y)
            insertCoordinates.executeUpdate()

            val coordinatesKeys = insertCoordinates.generatedKeys
            if (coordinatesKeys.next()) insertStudyGroup.setLong(2,
coordinatesKeys.getLong(1))
            else throw SQLException()
            coordinatesKeys.close()

            insertCoordinates.close()

            insertStudyGroup.setLong(3, studyGroup.studentsCount)
            insertStudyGroup.setString(4, studyGroup.formOfEducation.name)
            if (studyGroup.semesterEnum != null) {
                insertStudyGroup.setString(5, studyGroup.semesterEnum?.name)
            } else {
                insertStudyGroup.setNull(5, Types.OTHER)
            }

            val insertPerson =
                connection.prepareStatement("INSERT INTO persons (name,
birthday, eye_color, nationality, location) VALUES (?, ?, ?::color,
?::country, ?);",
                    Statement.RETURN_GENERATED_KEYS)
            insertPerson.setString(1, studyGroup.groupAdmin.name)
            insertPerson.setTimestamp(2,
Timestamp(studyGroup.groupAdmin.birthday.toEpochSecond(ZoneOffset.UTC) *
1000))
            if (studyGroup.groupAdmin.eyeColor != null) {
                insertPerson.setString(3,
studyGroup.groupAdmin.eyeColor?.name)
            } else insertPerson.setNull(3, Types.OTHER)
            insertPerson.setString(4, studyGroup.groupAdmin.nationality.name)

            val insertLocation =
                connection.prepareStatement("INSERT INTO locations (x, y, z,
name) VALUES (?, ?, ?, ?);",
                    Statement.RETURN_GENERATED_KEYS)
            insertLocation.setDouble(1, studyGroup.groupAdmin.location.x)
```

```
            if (studyGroup.groupAdmin.location.y != null) {
                insertLocation.setLong(2, studyGroup.groupAdmin.location.y!!)
            } else insertLocation.setNull(2, Types.OTHER)
            insertLocation.setFloat(3, studyGroup.groupAdmin.location.z)
            insertLocation.setString(4, studyGroup.groupAdmin.location.name)
            insertLocation.executeUpdate()

            val locationKeys = insertLocation.generatedKeys
            if (locationKeys.next()) insertPerson.setLong(5,
locationKeys.getLong(1))
            else throw SQLException()
            locationKeys.close()

            insertLocation.close()

            insertPerson.executeUpdate()

            val personKeys = insertPerson.generatedKeys
            if (personKeys.next()) insertStudyGroup.setLong(6,
personKeys.getLong(1))
            else throw SQLException()
            personKeys.close()

            insertPerson.close()

            insertStudyGroup.setLong(7, userId.get())
            insertStudyGroup.executeUpdate()

            val studyGroupKeys = insertStudyGroup.generatedKeys
            var id = 0L
            if (studyGroupKeys.next()) id = studyGroupKeys.getLong(1)
            studyGroupKeys.close()
            insertStudyGroup.close()
            if (id == 0L) throw SQLException()
            return id
        } catch (e: SQLException) {
            println(e.message)
            return 0
        }
    }

    fun clear() {
        connection.prepareStatement("DELETE FROM
study_groups;").executeUpdate()
        connection.prepareStatement("DELETE FROM
coordinates;").executeUpdate()
        connection.prepareStatement("DELETE FROM persons;").executeUpdate()
        connection.prepareStatement("DELETE FROM locations;").executeUpdate()
    }

    fun remove(id: Long): Boolean {
        try {
            val getIds =
                connection.prepareStatement("SELECT coordinates, group_admin
FROM study_groups WHERE id = (?);")
            getIds.setLong(1, id)
            val resultIds = getIds.executeQuery()
            val coordinatesId: Long
            val groupAdminId: Long
            if (resultIds.next()) {
                coordinatesId = resultIds.getLong("coordinates")
                groupAdminId = resultIds.getLong("group_admin")
            } else throw SQLException()
            resultIds.close()
```

```
            getIds.close()

            val getLocationId = connection.prepareStatement("SELECT location
FROM persons WHERE id = (?);")
            getLocationId.setLong(1, groupAdminId)
            val resultLocationId = getLocationId.executeQuery()
            val locationId: Long
            if (resultLocationId.next()) locationId =
resultLocationId.getLong("location")
            else throw SQLException()
            resultLocationId.close()

            getLocationId.close()

            val deleteStudyGroup = connection.prepareStatement("DELETE FROM
study_groups WHERE id = (?);")
            deleteStudyGroup.setLong(1, id)
            deleteStudyGroup.executeUpdate()

            deleteStudyGroup.close()

            val deleteCoordinates = connection.prepareStatement("DELETE FROM
coordinates WHERE id = (?);")
            deleteCoordinates.setLong(1, coordinatesId)
            deleteCoordinates.executeUpdate()

            deleteCoordinates.close()

            val deleteGroupAdmin = connection.prepareStatement("DELETE FROM
persons WHERE id = (?);")
            deleteGroupAdmin.setLong(1, groupAdminId)
            deleteGroupAdmin.executeUpdate()

            deleteGroupAdmin.close()

            val deleteLocation = connection.prepareStatement("DELETE FROM
locations WHERE id = (?);")
            deleteLocation.setLong(1, locationId)
            deleteLocation.executeUpdate()

            deleteLocation.close()
            return true
        } catch (e: SQLException) {
            println(e.message)
            return false }
    }

    fun update(id: Long, studyGroup: StudyGroup): Boolean {
        try {
            val getIds =
                connection.prepareStatement("SELECT coordinates, group_admin
FROM study_groups WHERE id = (?);")
            getIds.setLong(1, id)
            val resultIds = getIds.executeQuery()
            val coordinatesId: Long
            val groupAdminId: Long
            if (resultIds.next()) {
                coordinatesId = resultIds.getLong("coordinates")
                groupAdminId = resultIds.getLong("group_admin")
            } else throw SQLException()
            resultIds.close()

            getIds.close()
```

```
            val updateCoordinates = connection.prepareStatement(
                """
                   UPDATE coordinates
                   SET x = ?, y = ?
                   WHERE id = (?);
            """.trimIndent()
            )
            updateCoordinates.setDouble(1, studyGroup.coordinates.x)
            updateCoordinates.setFloat(2, studyGroup.coordinates.y)
            updateCoordinates.setLong(3, coordinatesId)
            updateCoordinates.executeUpdate()

            updateCoordinates.close()

            val getLocationId = connection.prepareStatement("SELECT location
FROM persons WHERE id = (?);")
            getLocationId.setLong(1, groupAdminId)
            val resultLocationId = getLocationId.executeQuery()
            val locationId: Long
            if (resultLocationId.next()) locationId =
resultLocationId.getLong("location")
            else throw SQLException()
            resultLocationId.close()

            getLocationId.close()

            val updateLocation = connection.prepareStatement(
                """
                    UPDATE locations
                    SET x = ?, y = ?, z = ?, name = ?
                    WHERE id = (?);
                """.trimIndent()
            )
            updateLocation.setDouble(1, studyGroup.groupAdmin.location.x)
            if (studyGroup.groupAdmin.location.y != null) {
                updateLocation.setLong(2, studyGroup.groupAdmin.location.y!!)
            } else {
                updateLocation.setNull(2, Types.OTHER)
            }
            updateLocation.setFloat(3, studyGroup.groupAdmin.location.z)
            updateLocation.setString(4, studyGroup.groupAdmin.location.name)
            updateLocation.setLong(5, locationId)
            updateLocation.executeUpdate()

            updateLocation.close()

            val updateGroupAdmin = connection.prepareStatement(
                """
                    UPDATE persons
                    SET name = ?,
                    birthday = ?,
                    eye_color = ?::color,
                    nationality = ?::country
                    WHERE id = ?;
                """.trimIndent()
            )
            updateGroupAdmin.setString(1, studyGroup.groupAdmin.name)
            updateGroupAdmin.setTimestamp(
                2,

Timestamp(studyGroup.groupAdmin.birthday.toEpochSecond(ZoneOffset.UTC) *
1000)
            )
            if (studyGroup.groupAdmin.eyeColor != null) {
```

```
                updateGroupAdmin.setString(3,
studyGroup.groupAdmin.eyeColor?.name)
            } else updateGroupAdmin.setNull(3, Types.OTHER)
            updateGroupAdmin.setString(4,
studyGroup.groupAdmin.nationality.name)
            updateGroupAdmin.setLong(5, groupAdminId)
            updateGroupAdmin.executeUpdate()

            updateGroupAdmin.close()

            val updateStudyGroup = connection.prepareStatement(
                """
                    UPDATE study_groups
                    SET name = ?,
                        students_count = ?,
                        form_of_education = ?::form_of_education,
                        semester_enum = ?::semester
                    WHERE id = ?;
                """.trimIndent()
            )
            updateStudyGroup.setString(1, studyGroup.name)
            updateStudyGroup.setLong(2, studyGroup.studentsCount)
            updateStudyGroup.setString(3, studyGroup.formOfEducation.name)
            if (studyGroup.semesterEnum != null) {
                updateStudyGroup.setString(4, studyGroup.semesterEnum!!.name)
            } else updateStudyGroup.setNull(4, Types.OTHER)
            updateStudyGroup.setLong(5, id)
            updateStudyGroup.executeUpdate()

            updateStudyGroup.close()
            return true
        } catch (e: SQLException) {
            return false
        }
    }

    fun exist(id: Long): Boolean {
        val statement = connection.prepareStatement("SELECT EXISTS (SELECT id
FROM study_groups WHERE id = ? );")
        statement.setLong(1, id)
        var result: ResultSet? = null
        try {
            result = statement.executeQuery()
            result.next()
            return result.getBoolean(1)
        } finally {
            statement.close()
            result?.close()
        }
    }

    fun checkCreatorId(objectId: Long): Boolean {
        val statement = connection.prepareStatement( "SELECT created_by FROM
study_groups WHERE id = ?;")
        statement.setLong(1, objectId)
        var result: ResultSet? = null
        try {
            result = statement.executeQuery()
            result.next()
            return userId.get() == result.getLong("created_by")
        } finally {
            statement.close()
            result?.close()
        }
```

```
    }

    fun countUsersObjects(): Int {
        val statement = connection.prepareStatement("SELECT COUNT(*) FROM
study_groups WHERE created_by = ?")
        statement.setLong(1, userId.get())
        var result : ResultSet? = null
        try {
            result = statement.executeQuery()
            result.next()
            return result.getInt(1)
        } finally {
            statement.close()
            result?.close()
        }
    }
}
```

server.managers.UserManager

```
package server.managers

import com.auth0.jwt.JWT
import com.auth0.jwt.algorithms.Algorithm
import com.auth0.jwt.exceptions.JWTVerificationException
import common.communication.Response
import common.communication.ResponseStatus
import org.postgresql.util.PSQLException
import server.network.Server
import java.security.MessageDigest
import java.security.SecureRandom
import java.sql.Connection
import java.util.*

class UserManager(private val connection: Connection) {
    private val pepper = "V2138^%4#9Ux"
    private val algorithm = Algorithm.HMAC256("verbasus")

    fun registerUser(login: String, password: String): Response {
        try {
            val statement = connection.prepareStatement("INSERT INTO users
(login, password, salt) VALUES (?, ?, ?);")
            statement.setString(1, login)
            val salt = generateSalt()
            statement.setString(3, salt)
            statement.setString(2, hashPassword(password, salt))
            statement.executeUpdate()
            return authenticateUser(login, password)
        } catch (e: PSQLException) {
            return Response(ResponseStatus.ERROR, "Пользователь с таким
логином уже есть!")
        }
    }

    fun authenticateUser(login: String, password: String): Response {
        val statement = connection.prepareStatement( "SELECT id, password,
salt FROM users WHERE login = ?;")
        statement.setString(1, login)
```

```
        val resultSet = statement.executeQuery()
        if (resultSet.next()) {
            if (hashPassword(password, resultSet.getString("salt")) ==
resultSet.getString("password")) {
                Server.sendResponse(Response(ResponseStatus.SUCCESS, "Вы
успешно вошли в систему!"))
                val setToken = connection.prepareStatement("UPDATE users SET
token = ? WHERE login = ?;")
                val token = generateToken(resultSet.getInt("id"), login)
                setToken.setString(1, token)
                setToken.setString(2, login)
                setToken.executeUpdate()
                return Response(ResponseStatus.TOKEN, token)
            }
            else return Response(ResponseStatus.ERROR, "Неверный пароль!")
        }
        else {
            return Response(ResponseStatus.ERROR, "Пользователь с таким
логином не найден!")
        }
    }

    private fun hashPassword(password: String, salt: String): String {
        val md = MessageDigest.getInstance("MD2")
        val hash = md.digest((password + pepper + salt).toByteArray())
        return Base64.getEncoder().encodeToString(hash)
    }

    private fun generateToken(id: Int, login: String): String {
        return JWT.create()
            .withSubject(login)
            .withClaim("id", id)
            .withExpiresAt(Date(System.currentTimeMillis() + 3600000))
            .sign(algorithm)
    }

    private fun generateSalt(): String {
        val salt = ByteArray(10)
        SecureRandom().nextBytes(salt)
        return Base64.getEncoder().encodeToString(salt).substring(0, 10)
    }

    fun verifyToken(token: String): Long {
        try {
            JWT.require(algorithm).build().verify(token)
            return JWT.decode(token).getClaim("id").asLong()
        } catch (e: JWTVerificationException) { return 0 }
    }
}
```

server.managers.ThreadPoolManager

```
package server.managers

import common.models.StudyGroup
import java.util.concurrent.ExecutorService
import java.util.concurrent.Executors
import java.util.concurrent.LinkedBlockingQueue

class ThreadPoolManager {
    val requestReaderExecutor: ExecutorService =
```

```
Executors.newFixedThreadPool(5)
    val requestProcessorExecutor: ExecutorService =
Executors.newFixedThreadPool(5)
    val responseSenderExecutor: ExecutorService =
Executors.newCachedThreadPool()
    val objectQueue = LinkedBlockingQueue<StudyGroup>()
}
```

Заключение

В результате выполнения лабораторной работы я научилась работать с многопоточностью и взаимодействовать с базами данных в Java.