«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Факультет программной инженерии и компьютерной техники

Отчет

По лабораторной работе №7

Вариант 167414

Студент:

*Ильин Н. С.*

*Р3110 поток 2.9*

Преподаватель:

*Бойко В.А.*

Санкт-Петербург, 2024 г.

Оглавление

[**Задание:** 3](#_Toc160583382)

[**Выполнение работы:** 4](#_Toc160583383)

[**Выводы**: 11](#_Toc160583384)

# **Задание:**

Доработать программу из [лабораторной работы №6](https://se.ifmo.ru/courses/programming#lab6) 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 SHA-384
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 создание нового потока (java.lang.Thread) --- READ
2. Для многопотчной обработки полученного запроса использовать Cached thread pool -- HANDLE
3. Для многопоточной отправки ответа использовать Cached thread pool --- SEND
4. Для синхронизации доступа к коллекции использовать синхронизацию чтения и записи с помощью java.util.concurrent.locks.ReadWriteLock

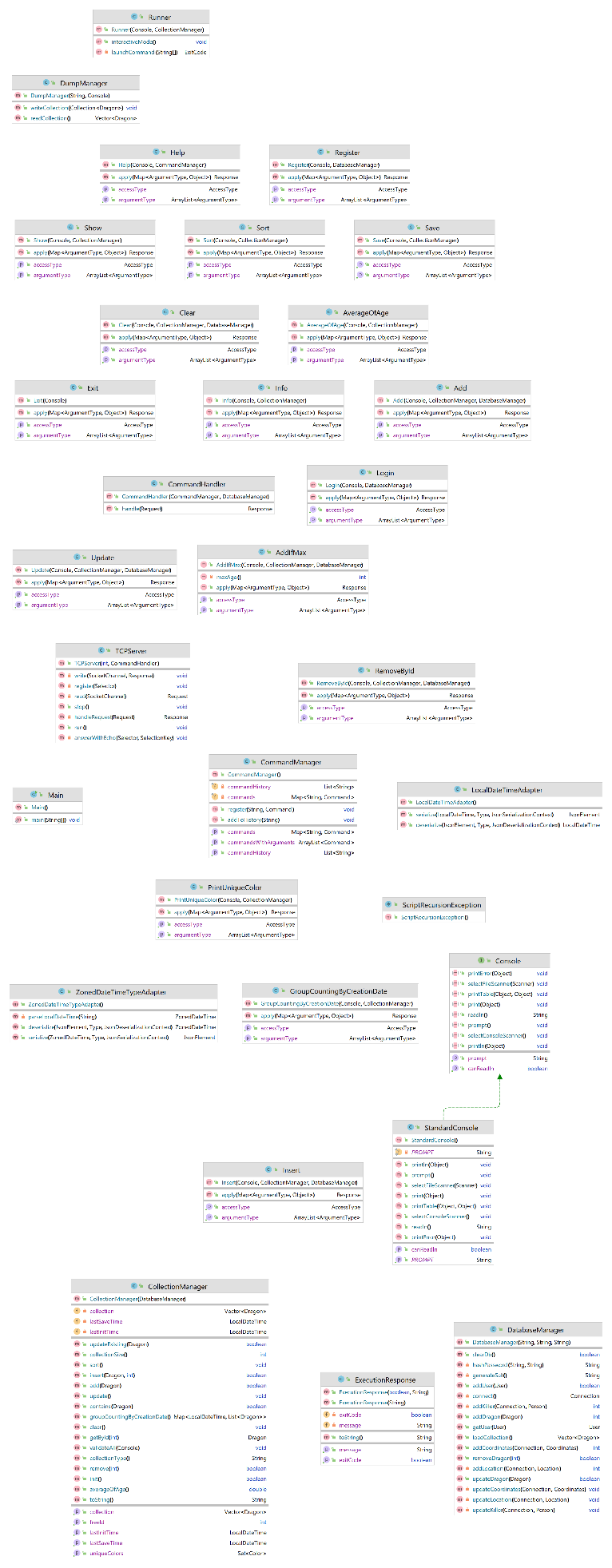
**Порядок выполнения работы:**

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.

# **Выполнение работы:**

**UML-диаграмма:**

Сервер**:**



Клиент:



**Github:**  
<https://github.com/MrTheFall/study/tree/master/2sem/lab7>

**Наиболее важные части кода:**

Client Main.java

package org.lab6;  
  
import org.apache.logging.log4j.LogManager;  
import org.apache.logging.log4j.Logger;  
import org.lab6.commands.\*;  
import org.lab6.handlers.CommandHandler;  
import org.lab6.managers.\*;  
import org.lab6.utils.console.StandardConsole;  
  
import java.io.IOException;  
import java.net.SocketException;  
import java.net.UnknownHostException;  
import java.sql.SQLException;  
  
public class Main {  
 public static final int *PORT* = 25565;  
  
 public static Logger *logger* = LogManager.*getLogger*("ServerLogger");  
  
 public static void main(String[] args) throws SQLException {  
 var console = new StandardConsole();  
  
*// String filename = System.getenv("FILENAME");  
// if (filename == null || filename.isEmpty()) {  
// System.out.println("Задайте имя загружаемого файла с помощью переменной среды FILENAME");  
// System.exit(1);  
// }  
// System.out.println(filename);  
 //var dumpManager = new DumpManager(filename, console);* var dbUrl = System.*getenv*("PG\_URL");  
 var dbUsername = System.*getenv*("PG\_USERNAME");  
 var dbPassword = System.*getenv*("PG\_PASSWORD");  
 var databaseManager = new DatabaseManager(dbUrl, dbUsername, dbPassword);  
 var collectionManager = new CollectionManager(databaseManager);  
 if (!collectionManager.init()) { System.*exit*(1); }  
  
 collectionManager.validateAll(console);  
 *//Runtime.getRuntime().addShutdownHook(new Thread(collectionManager::saveCollection));* var commandManager = new CommandManager() {{  
 register("help", new Help(console, this));  
 register("info", new Info(console, collectionManager));  
 register("add", new Add(console, collectionManager, databaseManager));  
 register("update", new Update(console, collectionManager, databaseManager));  
 register("remove\_by\_id", new RemoveById(console, collectionManager, databaseManager));  
 register("clear", new Clear(console, collectionManager, databaseManager));  
 register("show", new Show(console, collectionManager));  
 register("add\_if\_max", new AddIfMax(console, collectionManager, databaseManager));  
 register("insert", new Insert(console, collectionManager, databaseManager));  
 register("sort", new Sort(console, collectionManager));  
 register("average\_of\_age", new AverageOfAge(console, collectionManager));  
 register("group\_counting\_by\_creation\_date", new GroupCountingByCreationDate(console, collectionManager));  
 register("print\_unique\_color", new PrintUniqueColor(console, collectionManager));  
 register("register", new Register(console, databaseManager));  
 register("login", new Login(console, databaseManager));  
  
 register("exit", new Exit(console)); *// Server-only command* register("save", new Save(console, collectionManager)); *// Server-only command* }};  
  
 try {  
 var server = new TCPServer(PORT, new CommandHandler(commandManager, databaseManager));  
 *//server.setAfterHook(collectionManager::saveCollection);* server.run();  
 } catch (SocketException e) {  
 logger.fatal("Случилась ошибка сокета", e);  
 } catch (UnknownHostException e) {  
 logger.fatal("Неизвестный хост", e);  
 } catch (IOException e) {  
 throw new RuntimeException(e);  
 }  
 }  
}

Server TCPServer.java

**package org.lab6.managers;**package org.lab6.managers;  
  
import common.models.Dragon;  
import common.network.Request;  
import common.network.Response;  
import common.utils.ArgumentType;  
import common.utils.CommandDTO;  
import org.apache.commons.lang3.SerializationUtils;  
import org.apache.logging.log4j.Logger;  
import org.lab6.Main;  
import org.lab6.handlers.CommandHandler;  
import org.lab6.utils.Runner;  
import org.lab6.utils.console.Console;  
  
import javax.annotation.processing.SupportedSourceVersion;  
import java.io.\*;  
import java.net.InetSocketAddress;  
import java.net.SocketException;  
import java.nio.ByteBuffer;  
import java.nio.channels.\*;  
import java.util.\*;  
import java.util.concurrent.ExecutorService;  
import java.util.concurrent.Executors;  
  
*/\*\*  
 \* TCP обработчик запросов  
 \*/*public class TCPServer {  
 private final Selector selector;  
 private final ServerSocketChannel serverSocket;  
 private final CommandHandler commandHandler;  
 private final Logger logger = Main.logger;  
 private final ByteBuffer buffer;  
 private boolean running = true;  
  
 *// Create two different CachedThreadPool for handling and sending response* private final ExecutorService handlePool = Executors.newCachedThreadPool();  
 private final ExecutorService sendPool = Executors.newCachedThreadPool();  
  
 public TCPServer(int port, CommandHandler commandHandler) throws IOException {  
 this.buffer = ByteBuffer.allocate(16000);  
 this.selector = Selector.open();  
 this.serverSocket = ServerSocketChannel.open();  
 this.serverSocket.bind(new InetSocketAddress(port));  
 this.serverSocket.configureBlocking(false);  
 this.serverSocket.register(selector, SelectionKey.OP\_ACCEPT);  
 this.commandHandler = commandHandler;  
 }  
 public void run() throws IOException {  
 logger.info("TCP Server started on port " + serverSocket.socket().getLocalPort());  
  
 while (running) {  
 selector.select();  
 Set<SelectionKey> selectedKeys = selector.selectedKeys();  
 Iterator<SelectionKey> iter = selectedKeys.iterator();  
  
 while (iter.hasNext()) {  
 SelectionKey key = iter.next();  
 if (key.isValid() && key.isAcceptable()) {  
 register(selector);  
 }  
  
 if (key.isValid() && key.isReadable()) {  
 new Thread(() -> {  
 answerWithEcho(selector, key);  
 }).start();  
 }  
  
 iter.remove();  
  
 if (System.in.available() > 0) {  
 BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));  
 String[] line= (bufferedReader.readLine().trim() + " ").split(" ", 2);  
 Request request = new Request(new CommandDTO(line[0], "", new ArrayList<>()), new HashMap<>());  
 commandHandler.handle(request);  
 }  
 }  
 }  
 }  
  
 private void register(Selector selector) throws IOException {  
 SocketChannel client = serverSocket.accept();  
 client.configureBlocking(false);  
 SelectionKey key = client.register(selector, SelectionKey.OP\_READ);  
 }  
  
  
  
 private Request read(SocketChannel client) throws IOException, ClassNotFoundException {  
 ByteArrayOutputStream baos = new ByteArrayOutputStream();  
 buffer.clear();  
 int r = 0;  
 try {  
 while ((r = client.read(buffer)) > 0) {  
 buffer.flip();  
 baos.write(buffer.array(), 0, r);  
 buffer.clear();  
 }  
 } catch (SocketException se) {  
 logger.error("Connection reset by client");  
 client.close();  
 return null;  
 }  
  
 if (r < 0) {  
 return null;  
 }  
  
 byte[] bytes = baos.toByteArray();  
 ByteArrayInputStream bis = new ByteArrayInputStream(bytes);  
 try {  
 ObjectInputStream ois = new ObjectInputStream(bis);  
 Request request = (Request) ois.readObject();  
 logger.info("Processing request: " + request);  
  
 return request;  
 } catch (EOFException e) {  
 *//System.out.println("Incomplete object received");* return null;  
 }  
 }  
 private void write(SocketChannel client, Response response) throws IOException {  
 ByteArrayOutputStream bos = new ByteArrayOutputStream();  
 ObjectOutputStream oos = new ObjectOutputStream(bos);  
  
 oos.writeObject(response);  
 oos.flush();  
  
 buffer.clear();  
 buffer.put(bos.toByteArray());  
 buffer.flip();  
 while (buffer.hasRemaining()) {  
 client.write(buffer);  
 }  
 }  
  
 private Response handleRequest(Request request) {  
 Response response;  
 if (request.getCommand().getName().equals("update\_commands")) {  
 response = new Response(true, null, commandHandler.manager.getCommandsWithArguments());  
 } else {  
 response = commandHandler.handle(request);  
 }  
 return response;  
 }  
  
 private void answerWithEcho(Selector selector, SelectionKey key) {  
 SocketChannel client = (SocketChannel) key.channel();  
  
 if (!client.isOpen()) {  
 return;  
 }  
  
 try {  
 Request request = read(client);  
 if(request == null) return;  
  
 *// Use handlePool to handle the request* handlePool.execute(() -> {  
 Response response = handleRequest(request);  
  
 *// Use sendPool to send the response* sendPool.execute(() -> {  
 try {  
 if (client.isOpen()) {  
 write(client, response);  
 }  
 else System.out.println("NOT OPEN");  
 } catch (IOException e) {  
 logger.error("Failed to write response", e);  
 }  
 });  
 });  
 } catch (IOException | ClassNotFoundException e) {  
 logger.error("Client disconnected", e);  
 try {  
 client.close();  
 } catch (IOException ex) {  
 logger.error("Failed to close client", ex);  
 }  
 }  
 }  
  
  
  
 public void stop() {  
 running = false;  
 try {  
 serverSocket.close();  
 } catch (IOException e) {  
 logger.error("Failed to close server socket when stopping: " + e.getMessage(), e);  
 }  
 }  
}

Common Request.java

**package** common.network.requests;

**import** java.io.Serializable;

**import** java.util.Objects;

**public** **abstract** **class** Request **implements** Serializable {

**private** **final** String name;

**public** Request(String name) {

**this**.name = name;

}

**public** String getName() {

**return** name;

}

@Override

**public** **boolean** equals(Object o) {

**if** (**this** == o) **return** **true**;

**if** (o == **null** || getClass() != o.getClass()) **return** **false**;

Request response = (Request) o;

**return** Objects.equals(name, response.name);

}

@Override

**public** **int** hashCode() {

**return** Objects.hash(name);

}

@Override

**public** String toString() {

**return** "Request{" +

"name='" + name + '**\'**' +

'}';

}

}

Common Response.java

package common.network;  
  
import common.models.Color;  
import common.models.Dragon;  
import common.utils.ArgumentType;  
import common.utils.Command;  
  
import java.io.Console;  
import java.io.Serializable;  
import java.time.LocalDateTime;  
import java.util.\*;  
  
public class Response implements Serializable {  
 private final boolean success;  
 private final String message;  
 private final Vector<Dragon> dragons;  
 private final ArrayList<Command> commands;  
 private final Set<Color> uniqueColors;  
 private final Map<LocalDateTime, List<Dragon>> dragonGroups;  
  
 public Response(boolean success, String message, Vector<Dragon> dragons, ArrayList<Command> commands, Set<Color> uniqueColors, Map<LocalDateTime, List<Dragon>> dragonGroups) {  
 this.success = success;  
 this.message = message;  
 this.dragons = dragons;  
 this.commands = commands;  
 this.uniqueColors = uniqueColors;  
 this.dragonGroups = dragonGroups;  
 }  
 public Response(boolean success, String message, Vector<Dragon> dragons, ArrayList<Command> commands) {  
 this.success = success;  
 this.message = message;  
 this.dragons = dragons;  
 this.commands = commands;  
 this.uniqueColors = null;  
 this.dragonGroups = null;  
 }  
 public Response(boolean success, String message, Vector<Dragon> dragons) {  
 this.success = success;  
 this.message = message;  
 this.dragons = dragons;  
 this.commands = null;  
 this.uniqueColors = null;  
 this.dragonGroups = null;  
 }  
 public Response(boolean success, String message, ArrayList<Command> commands) {  
 this.success = success;  
 this.message = message;  
 this.dragons = null;  
 this.commands = commands;  
 this.uniqueColors = null;  
 this.dragonGroups = null;  
 }  
 public Response(boolean success, String message, Set<Color> uniqueColors) {  
 this.success = success;  
 this.message = message;  
 this.dragons = null;  
 this.commands = null;  
 this.uniqueColors = uniqueColors;  
 this.dragonGroups = null;  
 }  
 public Response(boolean success, String message, Map<LocalDateTime, List<Dragon>> dragonGroups) {  
 this.success = success;  
 this.message = message;  
 this.dragons = null;  
 this.commands = null;  
 this.uniqueColors = null;  
 this.dragonGroups = dragonGroups;  
 }  
 public Response(boolean success, String message) {  
 this.success = success;  
 this.message = message;  
 this.dragons = null;  
 this.commands = null;  
 this.uniqueColors = null;  
 this.dragonGroups = null;  
 }  
  
 public Vector<Dragon> getDragons() {  
 return this.dragons;  
 }  
  
 public String getMessage() {  
 return message;  
 }  
  
 public boolean isSuccess() {  
 return success;  
 }  
  
 @Override  
 public boolean equals(Object o) {  
 if (this == o) return true;  
 if (o == null || getClass() != o.getClass()) return false;  
 Response response = (Response) o;  
 return Objects.*equals*(message, response.message);  
 }  
  
 @Override  
 public int hashCode() {  
 return Objects.*hash*(success, message, dragons);  
 }  
  
 @Override  
 public String toString() {  
 return "Response{" +  
 "success='" + success + '\'' +  
 ", message='" + message + '\'' +  
 ", dragonsCount='" + ((dragons == null) ? null : dragons.size()) + '\'' +  
 '}';  
 }  
  
 public ArrayList<Command> getCommands() {  
 return commands;  
 }  
  
 public Set<Color> getUniqueColors() {  
 return uniqueColors;  
 }  
  
 public Map<LocalDateTime, List<Dragon>> getDragonGroups() {  
 return dragonGroups;  
 }  
}

# **Выводы**:

В ходе выполнения данной лабораторной работы были осуществлены значительные изменения в структуре программы. Основное внимание уделялось переходу на использование реляционной СУБД (PostgreSQL) для хранения данных. Это позволило мне познакомиться с JDBC.

Удаление хранения коллекции в файле и замена его на реляционную базу данных позволило обеспечить их надежное и безопасное хранение. Такой подход обеспечивает более удобный доступ к данным, улучшает их организацию, а также повышает потенциальную безопасность данных за счет встроенных механизмов контроля доступа и резервного копирования.

Реализованная возможность регистрации и авторизации пользователей обеспечивает контроль доступа к данным . Использование алгоритма SHA-384 для хеширования паролей обеспечивает безопасное хранение паролей. А добавляемая соль значительно затрудняет атаки на сам хэш: брут/поиск по радужным таблицы.

Организация многопоточной обработки запросов позволила повысить производительность приложения и обеспечить более эффективное использование ресурсов системы. В частности, использование различных потоков для чтения запросов, обработки полученного запроса и отправки ответа позволило оптимизировать процесс обработки запросов и улучшить отзывчивость системы.

Синхронизация доступа к коллекции с помощью java.util.concurrent.locks.ReadWriteLock обеспечивает консистентность данных при многопоточной обработке и предотвращает возникновение проблем с согласованностью данных.

В целом, данная работа представляет собой важный этап в изучении принципов работы с реляционными базами данных и многопоточной обработки в Java. Полученные знания и навыки будут полезны при разработке сложных многопользовательских и многопоточных приложений, а также при работе с системами управления базами данных.