«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:** 5](#_Toc160583383)

[**Выводы**: 16](#_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/lab6>

**Наиболее важные части кода:**

Client Main.java

package org.lab6;  
  
import org.lab6.network.TCPClient;  
import org.lab6.utils.Runner;  
import org.lab6.utils.console.StandardConsole;  
import org.apache.logging.log4j.LogManager;  
import org.apache.logging.log4j.Logger;  
  
import java.io.IOException;  
import java.net.ConnectException;  
import java.net.InetAddress;  
  
public class Main {  
  
 private static final int *PORT* = 25565;  
 public static final Logger *logger* = LogManager.*getLogger*("ClientLogger");  
  
 public static void main(String[] args) {  
 var console = new StandardConsole();  
  
 try {  
 var client = new TCPClient(InetAddress.*getLocalHost*(), *PORT*);  
 new Runner(console, client).interactiveMode();  
 }catch (ConnectException e){  
 *logger*.error("Сервер недоступен");  
 } catch (IOException e) {  
 *logger*.error(e);  
 } catch (ClassNotFoundException e) {  
 throw new RuntimeException(e);  
 }  
 }  
}

Client Runner.java

**v**

Server 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.CollectionManager;

**import** org.lab6.managers.CommandManager;

**import** org.lab6.managers.DumpManager;

**import** org.lab6.managers.TCPServer;

**import** org.lab6.utils.console.StandardConsole;

**import** java.io.IOException;

**import** java.net.SocketException;

**import** java.net.UnknownHostException;

**public** **class** Main {

**public** **static** **final** **int** PORT = 25565;

**public** **static** Logger logger = LogManager.getLogger("ServerLogger");

**public** **static** **void** main(String[] args) {

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 collectionManager = **new** CollectionManager(dumpManager);

**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));

register("update", **new** Update(console, collectionManager));

register("remove\_by\_id", **new** RemoveById(console, collectionManager));

register("clear", **new** Clear(console, collectionManager));

register("show", **new** Show(console, collectionManager));

register("add\_if\_max", **new** AddIfMax(console, collectionManager));

register("insert", **new** Insert(console, collectionManager));

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));

}};

**try** {

var server = **new** TCPServer(PORT, **new** CommandHandler(commandManager));

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;  
  
import common.models.Dragon;  
import common.network.Request;  
import common.network.Response;  
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.console.Console;  
  
import java.io.\*;  
import java.net.ServerSocket;  
import java.net.Socket;  
import java.util.Collections;  
import java.util.Comparator;  
import java.util.Vector;  
  
*/\*\*  
 \* TCP обработчик запросов  
 \*/*public class TCPServer {  
 private final ServerSocket serverSocket;  
 private final CommandHandler commandHandler;  
 private Runnable afterHook;  
  
 private final Logger logger = Main.*logger*;  
 private boolean running = true;  
  
 public TCPServer(int port, CommandHandler commandHandler) throws IOException {  
 this.serverSocket = new ServerSocket(port);  
 this.commandHandler = commandHandler;  
 }  
  
 public void run() {  
 logger.info("TCP Server started on port " + serverSocket.getLocalPort());  
  
 while (running) {  
 Socket clientSocket = null;  
 try {  
 clientSocket = serverSocket.accept();  
 logger.info("Client connected from " + clientSocket.getRemoteSocketAddress());  
  
 ObjectOutputStream output = new ObjectOutputStream(clientSocket.getOutputStream());  
 ObjectInputStream input = new ObjectInputStream(clientSocket.getInputStream());  
  
 while (running && !clientSocket.isClosed()) {  
 Request request = null;  
 Response response = null;  
 try {  
 request = (Request) input.readObject();  
 logger.info("Processing request: " + request);  
  
 if (request.getCommand().getName().equals("update\_commands")) {  
 response = new Response(true, null, commandHandler.manager.getCommandsWithArguments());  
 } else if (request.getCommand().getName().equals("show")) {  
 response = commandHandler.handle(request);  
 Collections.*sort*(response.getDragons(), new Comparator<Dragon>() {  
 @Override  
 public int compare(Dragon d1, Dragon d2) {  
 int size1 = SerializationUtils.*serialize*(d1).length;  
 int size2 = SerializationUtils.*serialize*(d2).length;  
 return Integer.*compare*(size1, size2);  
 }  
 });  
 //logger.printf(response.getDragons()); // Logging the sorted list  
 } else {  
 response = commandHandler.handle(request);  
 }  
 } catch (IOException e) {  
 logger.error("Error receiving request: " + e.getMessage());  
 break;  
 } catch (ClassNotFoundException e) {  
 logger.error("Error while receiving the request: " + e.getMessage(), e);  
 response = new Response(false, "Failed to process request: " + (request != null ? request.getCommand() : "unknown"), (Vector<Dragon>) null);  
 } finally {  
 if (afterHook != null) afterHook.run();  
 }  
  
 try {  
 output.reset();  
 output.writeObject(response);  
 output.flush();  
 logger.info("Response sent to client");  
 } catch (IOException e) {  
 logger.error("Error sending response to client: ", e);  
 }  
 }  
 } catch (IOException e) {  
 logger.error("Error handling client connection: " + e.getMessage(), e);  
 } finally {  
 if (clientSocket != null && !clientSocket.isClosed()) {  
 try {  
 clientSocket.close();  
 } catch (IOException e) {  
 logger.error("Failed to close client socket: " + e.getMessage(), e);  
 }  
 }  
 }  
 }  
  
 // Close server socket  
 try {  
 serverSocket.close();  
 } catch (IOException e) {  
 logger.error("Failed to close server socket: " + e.getMessage(), e);  
 }  
 }  
  
 */\*\*  
 \* Устанавливает хук для вызова функции после каждого запроса.  
 \*  
 \* @param afterHook функция которая будет запущена после каждого запроса.  
 \*/* public void setAfterHook(Runnable afterHook) {  
 this.afterHook = afterHook;  
 }  
  
 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 рамках данной лабораторной работы было выполнено разделение приложения на клиентский и серверный модули, что является важным шагом в изучении архитектуры клиент-сервер и разработки распределенных систем. Реализация серверного модуля, который отвечает за управление коллекцией, позволила осуществить централизованное управление данными. Клиентский модуль, в свою очередь, обеспечил отправку команд на сервер в интерактивном режиме и получение результатов обработки. Такое разделение обеспечило четкое разграничение логики работы с данными и пользовательским интерфейсом.

Сериализация объектов для передачи между клиентом и сервером представляет собой стандартный механизм обмена данными в распределенных приложениях, и его освоение было важным практическим навыком. Понимание принципов и особенностей сериализации несомненно окажется полезным в дальнейшей профессиональной деятельности.

Особое внимание было уделено устойчивости работы клиентского приложения при возникновении временной недоступности сервера, что улучшает общую надежность системы и повышает качество пользовательского опыта.

Использование Stream API и лямбда-выражений для работы с объектами коллекции показало значительно большую эффективность и лаконичность кода по сравнению с традиционными подходами. Это подчеркнуло значимость функционального программирования в современной разработке ПО.

Использование неблокирующего ввода-вывода при реализации обмена данными позволило повысить эффективность работы приложения, избегая ожидания операций ввода-вывода и тем самым используя ресурсы системы более рационально. Это не только увеличивает производительность, но и обеспечивает более высокое качество обслуживания пользователей.

Четко определенные обязанности серверного и клиентского приложений усиливают структурированность разработки и облегчают поддержку кода и его масштабирование.

Особенное значение в процессе работы приобрело логирование с помощью популярной библиотеки Log4J2, которое обеспечило детальный контроль за процессом выполнения серверного приложения и способствовало более эффективной отладке.

Это исследование позволило глубже понять принципы распределенных систем и важность четкого разделения функционала на клиентскую и серверную части приложения. Также было уделено внимание практике валидации данных и обработке исключений, что повышает надежность и безопасность приложения. Полученный опыт и знания станут основой для создания масштабируемых и устойчивых программных решений в будущем.