МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ ЛИПЕЦКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Институт компьютерных наук
Кафедра автоматизированных систем управления

ОТЧЕТ

о производственной практике

"Эксплуатационная практика"

в ООО "НЛМК-ИТ"

Студент АС-21-1 Станиславчук С.М.

Руководитель от кафедры Болдырихин О.В.

Руководитель от предприятия Мирсаитов Г.Р.

Липецк 2024 г.

ЗАДАНИЕ НА ПРОИЗВОДСТВЕННУЮ ПРАКТИКУ

Студенту Станиславчуку Сергею Михайловичу группы АС-21-1

Направление (специальность) 09.03.01 "Информатика и вычислительная техника"

Изучить информацию по общей программе практики:

1. Структуру, деятельность, продукты для анализа бизнес-процессов, задачи ООО «НЛМК-ИТ».

2. Используемые на предприятии и в подразделении средства автоматизации: Docker, PostgreSQL, Spring.

3. Действующий порядок разработки и использования средств автоматизации: техническую политику предприятий Группы НЛМК в области систем автоматизации, инструменты реализации технической политики, требования к компонентам систем автоматизации и т.п.

Разработать решения задач по изучению используемого в бэкенд отделе языка программирования - Java

1. Ознакомиться с основной документацией по работе отдела.

2. Решить задачи в целях освоения языка программирования Java.

3. Изучить и настроить Docker, Spring фреймворк.

Руководитель практики от предприятия начальник отдела Мирсаитов Г.Р

Руководитель практики от ЛГТУ Старший преподаватель Болдырихин О.В.

Задание принял к исполнению студент Станиславчук С.М.

Аннотация

Данный отчёт — это пояснительная записка по пройденному материалу производственной практики.

Документ включает в себя описание характеристик предприятия и выполнение задания, выданного руководителем практики. Содержание указанных разделов соответствует стандартам ЕСПД и СТД АСУ соответствующих наименований.

Оглавление

Введение

Для организации разработки было принято решение использовать разделение на задачи. Таким образом были выделены следующие блоки для реализации:

1. Понять принципы работы бэкенд отдела «НЛМК-ИТ».

2. Изучить язык программирования Java, написав игру «Жизнь».

3. Сгенерировать контейнер с БД Postgres, используя Docker Compose и получить доступ к БД, написав Java-программу.

4. Изучить Spring Framework, подключиться к БД Postgres, сделать простой CRUD.

Реализация третьего и четвертого пунктов подразумевает использование следующих инструментов: технологии взаимодействия с базой данных: для работы с базой данных Postgres в Java применяется драйвер JDBC, для упрощения написания кода использовался редактор текста nvim.

1. Краткое описание подразделения – места практики.

1.1. Характеристика деятельности организации.

Общество с ограниченной ответственность "НЛМК-ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ" (краткое наименование: ООО "НЛМК-ИТ" действует с 15.08.2012. Основной ОКВЭД - "разработка компьютерного программного обеспечения". Работает по 12 направлениям:

- Разработка компьютерного программного обеспечения;
- Производство прочих строительно-монтажных работ;
- Деятельность в области телевизионного вещания;
- Деятельность консультативная и работы в области компьютерных технологий;
- Торговля оптовая компьютерами, периферийными устройствами к компьютерам и программным обеспечением;
- Ремонт компьютеров и периферийного компьютерного оборудования;
- Монтаж промышленных машин и оборудования;
- Строительство местных линий электропередачи и связи;
- Производство электромонтажных работ;
- Деятельность по созданию и использованию баз данных и информационных ресурсов
- Деятельность, связанная с использованием вычислительной техники и информационных технологий, прочая;
- Деятельность по обработке данных, предоставление услуг по размещению информации и связанная с этим деятельность.

1.2. Производственная и организационная структура организации.

Производственная и организационная структура представлена на рис 1.

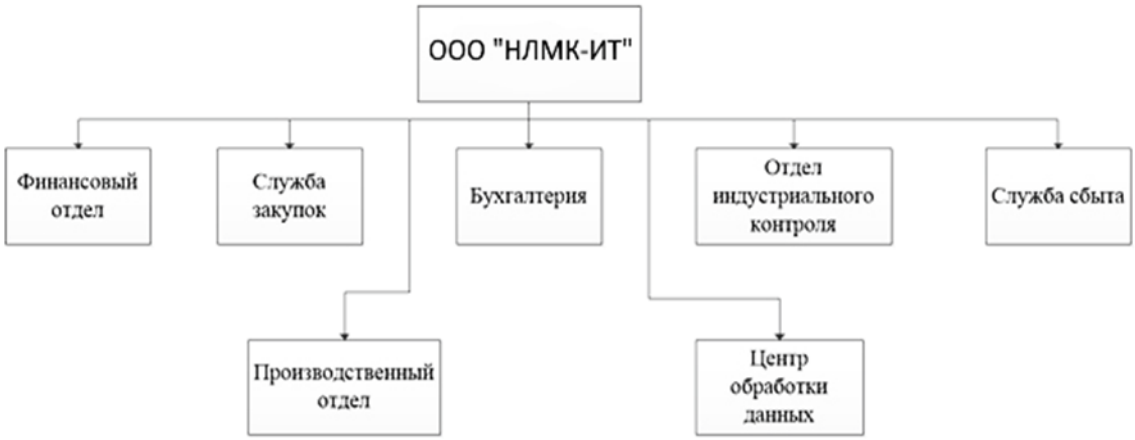

Рисунок 1 - Производственная и организационная структура организации

Источник: Studbooks.net [Электронный ресурс]: Анализ организационной структуры ОАО «НЛМК» - электронные текстовые данные.

1.3. Изучение стека разработки бэкенд отдела НЛМК.

1.3.1. Проблематика.

Необходимость быстрого освоения стека технологий, используемых в отделе разработки НЛМК, включающего Java, Spring Framework, Maven и Docker, чтобы эффективно интегрироваться в процесс разработки и поддержки приложений.

2. Описание выполнения общей программы.

2.1. Описание языка программирования Java.

Java — это строго типизированный объектно-ориентированный язык программирования общего назначения, разработанный компанией Sun Microsystems (в последующем приобретённой компанией Oracle).

Вот некоторые ключевые особенности Java:

1. Объектно-ориентированный подход (ООП). Это означает, что все в Java является объектом и позволяет разработчикам создавать модульный код, который легко поддерживать. Это повышает повторное использование и облегчает разработку сложных систем.

2. Платформонезависимость. Это означает, что приложение, написанное на Java, может работать на любой операционной системе без необходимости переписывания кода.

3. Автоматическое управление памятью, благодаря чему программисту не нужно беспокоиться о выделении и освобождении места.

4. Многопоточность языка позволяет одновременно выполнять несколько задач в одном приложении.

5. Безопасность. Java имеет встроенные механизмы безопасности, которые помогают защитить программы от вредоносного кода и предотвращают потенциальные уязвимости. Песочница (sandbox) Java ограничивает доступ приложения к ресурсам операционной системы, обеспечивая изоляцию и безопасность выполнения программ.

2.2. Обзор Spring.

По сути Spring Framework представляет собой просто контейнер внедрения зависимостей, с несколькими удобными слоями (например: доступ к базе данных, прокси, аспектно-ориентированное программирование, RPC (удаленный вызов процедур), веб-инфраструктура MVC (Модель-Представление-Контроллер)). Это все позволяет быстрее и удобнее создавать Java-приложения.

Основные возможности:

Контейнер внедрения зависимостей:

- **Inversion of Control (IoC):** Spring позволяет управлять зависимостями объектов с помощью IoC контейнера. Это способствует ослаблению связей между компонентами, улучшая тестируемость и модульность кода.
- **Configuration:** Поддержка конфигурации через аннотации (@Component, @Autowired, @Configuration) и XML.

Аспектно-ориентированное программирование:

- **Cross-Cutting Concerns:** Легкость интеграции аспектов, таких как логирование, транзакции, безопасность, без изменения основного бизнес-кода.
- **Anotations:** Использование аннотаций для определения аспектов (@Aspect, @Before, @After).

Доступ к данным:

- **Spring Data:** Упрощение работы с базами данных, поддержка JPA, Hibernate, JDBC и других технологий.
- **Repositories:** Автоматическая реализация репозиториев для стандартных операций (CRUD) с использованием интерфейсов.
- **Transaction Management:** Простое управление транзакциями с аннотациями (@Transactional).

Spring MVC:

- **Web Framework:** Обеспечивает мощный и гибкий веб-фреймворк для создания веб-приложений.

- **Annotations:** Поддержка аннотаций для конфигурации контроллеров (@Controller, @RequestMapping, @GetMapping, @PostMapping).

Spring Boot:

- **Auto-Configuration:** Упрощает настройку и конфигурацию Spring приложений с помощью автоматической конфигурации.
- **Embedded Servers:** Возможность использования встроенных серверов (Tomcat, Jetty) для разработки и тестирования.
- **Starter Dependencies:** Наборы зависимостей для различных типов приложений, которые упрощают добавление нужных библиотек.

Безопасность (Spring Security):

- **Authentication and Authorization:** Обеспечивает мощные средства для аутентификации и авторизации пользователей.
- **Annotations:** Использование аннотаций для защиты ресурсов (@Secured, @PreAuthorize).

2.3. Обзор Docker Compose.

Docker Compose — это инструментальное средство, входящее в состав Docker. Оно предназначено для решения задач, связанных с развёртыванием проектов. Это средство для определения и запуска приложений Docker с несколькими контейнерами. При работе в Compose используется файл YAML для настройки служб приложения. Затем нужно создать и запустить все службы из конфигурации путем выполнения одной команды.

3. Описание выполнения индивидуального задания.

3.1. Знакомство с Java

Прежде чем начать работу с языком Java, стоит установить набор для разработки - JDK. JDK представляет собой пакет инструментов для *разработки* программного обеспечения. JDK содержит компилятор Java. Компилятор — это программа, способная принимать исходные файлы с расширением .java, которые являются обычным текстом, и превращать их в исполняемые файлы с расширением .class.

Первым делом скачать JDK версии 21. Так как, бэкенд отдел НЛМК работает именно с этой версией Java. В Arch Linux это можно сделать при помощи команды pacman -S jdk21-openjdk. После установки, необходимо сделать скачанную JDK основной для системы. Для этого выполним 2 команды «archlinux-java status» для просмотра всех Java environment и выбрать необходимую при помощи «archlinux-java set java-21-openjdk»:

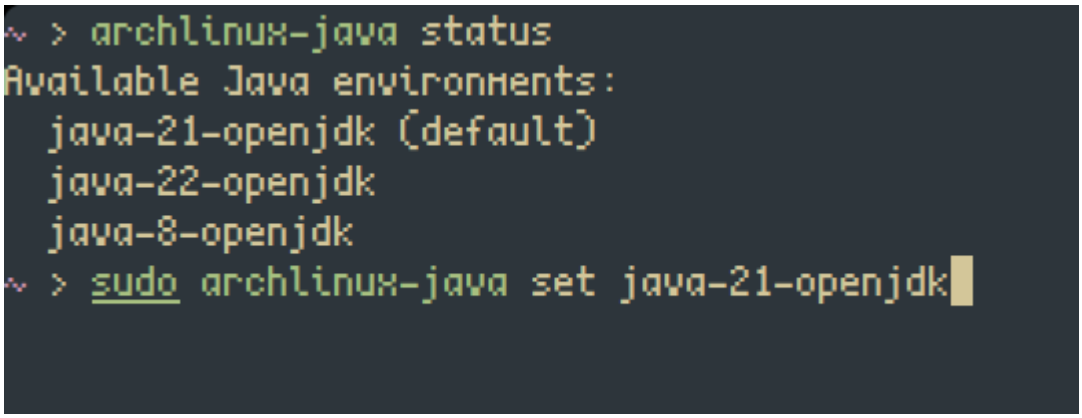

Рисунок 2 — Просмотр всех сред окружения Java и установка необходимой.

Для разработки понадобится Docker Compose. Для того, чтобы установить его, следует открыть командную строку Linux и ввести соответствующую команду (Arch Linux): pacman -Syy docker docker-compose.

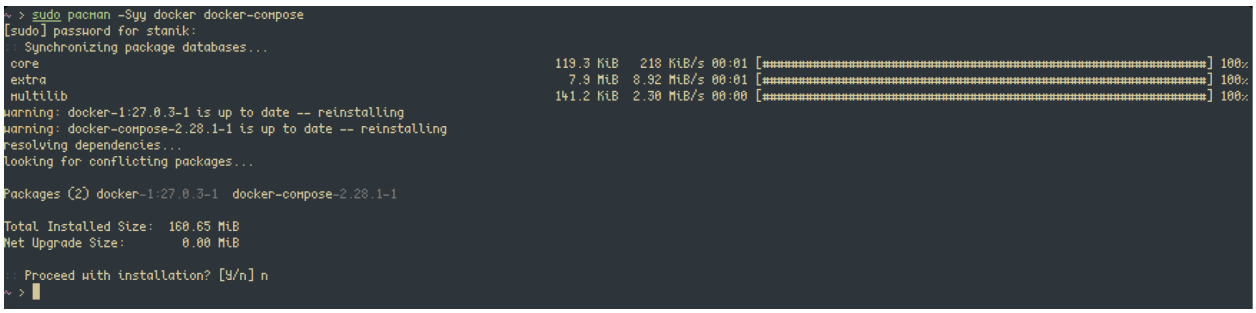

Рисунок 3 – установка Docker и Docker-Compose при помощи pacman

После завершения установки можно перейти к написанию скриптов для игры «Жизнь». Для этого создадим файл с расширением .java и откроем его с помощью nvim. Терминальный интерфейс редактора кода nvim представлен на рисунке 4.

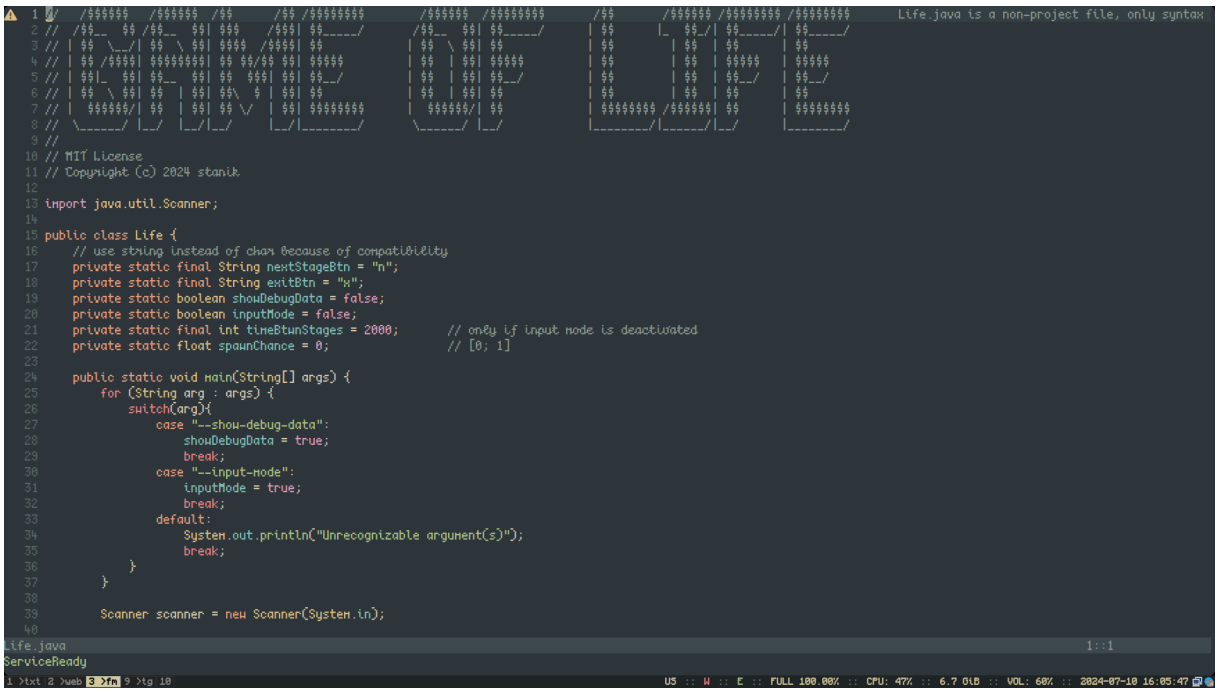

Рисунок 4 – Интерфейс nvim (терминал)

Текст программы находится в приложении А.

Для того, чтобы скомпилировать проект — то есть на основе .java создать .class файл (байт-код), нужно выполнить команду javac filename.java, в моем случае это javac Life.java. Теперь, чтобы запустить программу выполним команду: java Life (рис. 5).

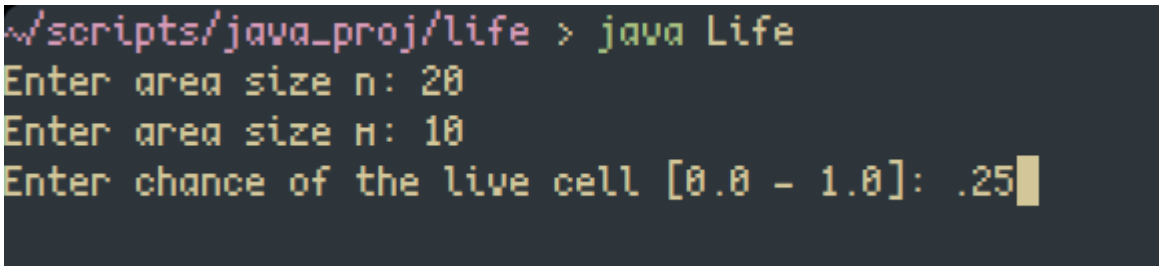

Рисунок 5 — Компиляция и запуск программы

После того, как были заданы размеры прямоугольника n и m, а также шанс на появление живой клетки, процесс игры запускается

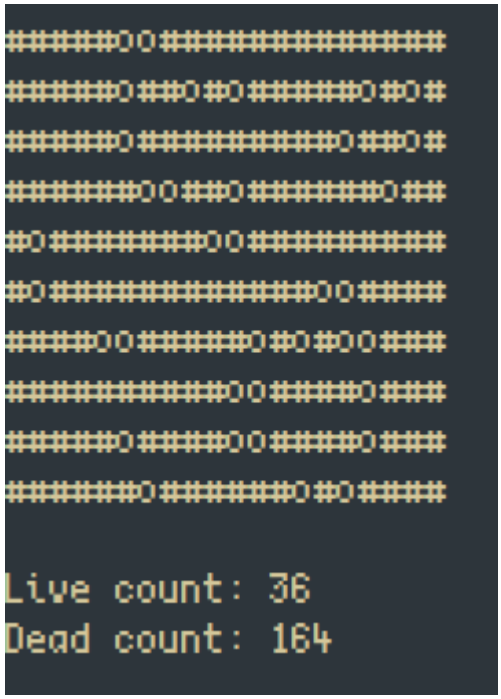

Рисунок 6 — Пример работающей java-программы

3.2 Осваивание Docker Compose

Следующей задачей является контейнеризация служб при помощи docker и docker-compose, подключение к БД PostgreSQL с помощью java скриптов.

Для того, чтобы Java смог «разговаривать» с БД, необходимо скачать JDBC Postgres — драйвер для подключения ЯП к БД. Скачать его можно на сайте jdbc.postgresql.org/download/. Выбираем последнюю версию (42.7.3) и скачиваем .jar файл. Сайт для скачивания указан на рис. 7.

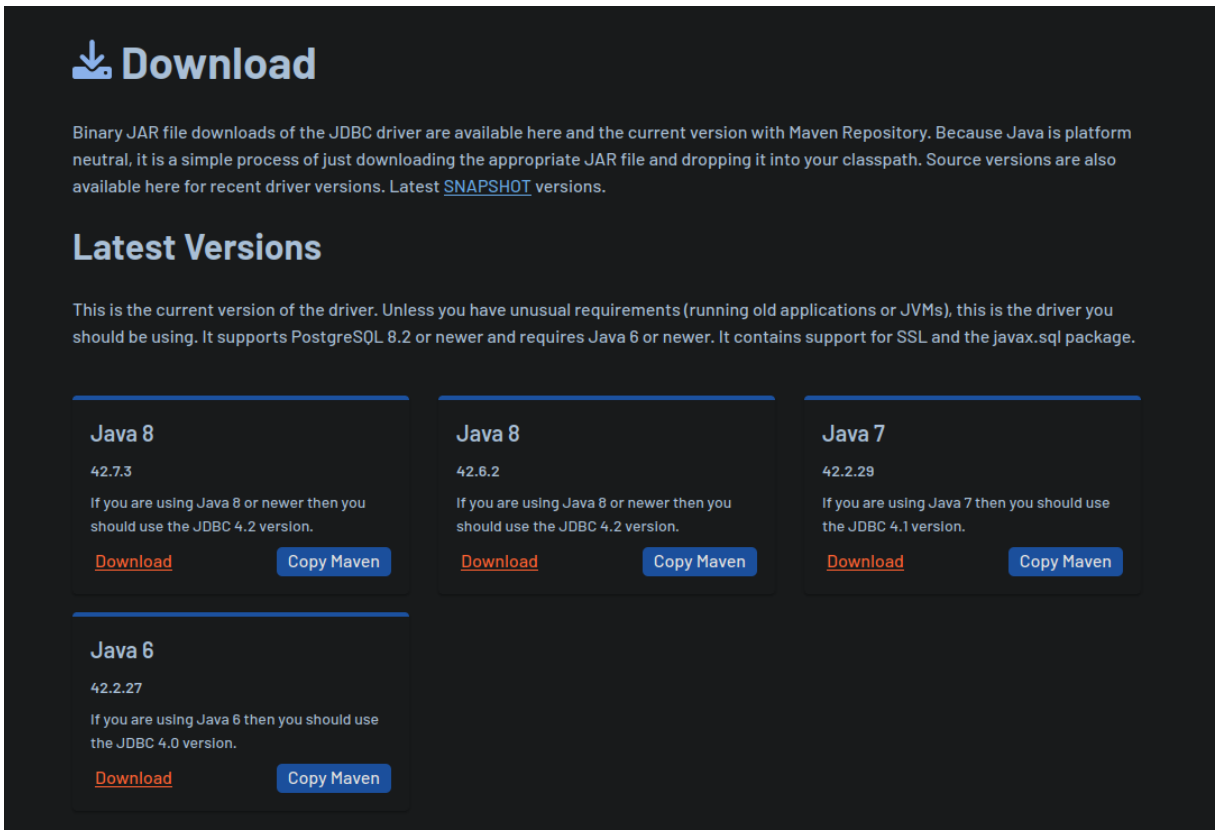

Рисунок 7 — Источник для скачивания JDBC драйвера.

Теперь напишем java-программу (приложение Б), которая будет обращаться к БД Postgres, которая, в свою очередь, будет размещена в контейнере при помощи Docker Compose. Java программа использует библиотеку java.sql, при помощи которой подключается и выполняет запросы БД под названием «lk» под пользователем «postgres». Скрипт .yml, реализующий контейнер указан в приложении В. В нем мы также указали название контейнера и имя пользователя, а также порт сервера и путь с данными локальной БД.

В терминале введем команду docker-compose up -d для того, чтобы поднять созданный контейнер с БД. Остается скомпилировать и запустить текущую java-программу, указав путь до JDBC.jar драйвера в качестве аргумента -cp (рис. 8)

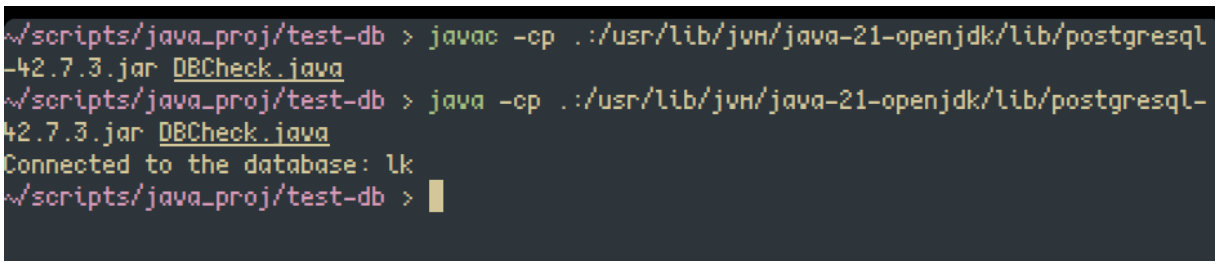

Рисунок 8 — Компиляция и запуск программы для подключения к БД.

3.3 Установка и настройка проекта с помощью Maven, работа со Spring Framework

Теперь нужно установить автоматизатор сборки проекта — Maven. Maven — это инструмент для управления проектами и автоматизации сборки, используемый в Java-проектах. Он помогает разработчикам управлять зависимостями, компилировать исходный код, запускать тесты, создавать пакеты и выполнять многие другие задачи, связанные с жизненным циклом проекта. Maven основан на концепции декларативного управления проектами, что означает, что вся конфигурация проекта описана в XML-файле, который называется pom.xml (Project Object Model). Установим Maven: sudo pacman -Syy maven.

Теперь нужно создать пустой Maven проект командой: mvn archetype:generate -DgroupId=com.example -DartifactId=my-app -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false

Теперь нужно создать и настроить pom.xml файл в корневой директории проекта (Приложение В). После чего запустить тест-проект и убедиться, что все работает. Теперь выполним mvn install для сборки проекта. После успешной сборки, произведем запуск проекта командой mvn exec:java -Dexec.mainClass="com.example.App". Если в результате выполнения программы, мы видим «Hello, World!», значит наша программа выполнилось корректно и Maven проект собрался успешно. Теперь добавим логику общения с БД Postgres. Добавим 3 новых скрипта (Приложение Г) Student.java, StudentController.java, StudentRepository.java и

несколько новых зависимостей для работы с БД Postgres и Spring фреймворком в файл pom.xml

Теперь выполним команду mvn clean install и mvn spring-boot:run.

Результат компиляции проекта указан на рисунке 9.

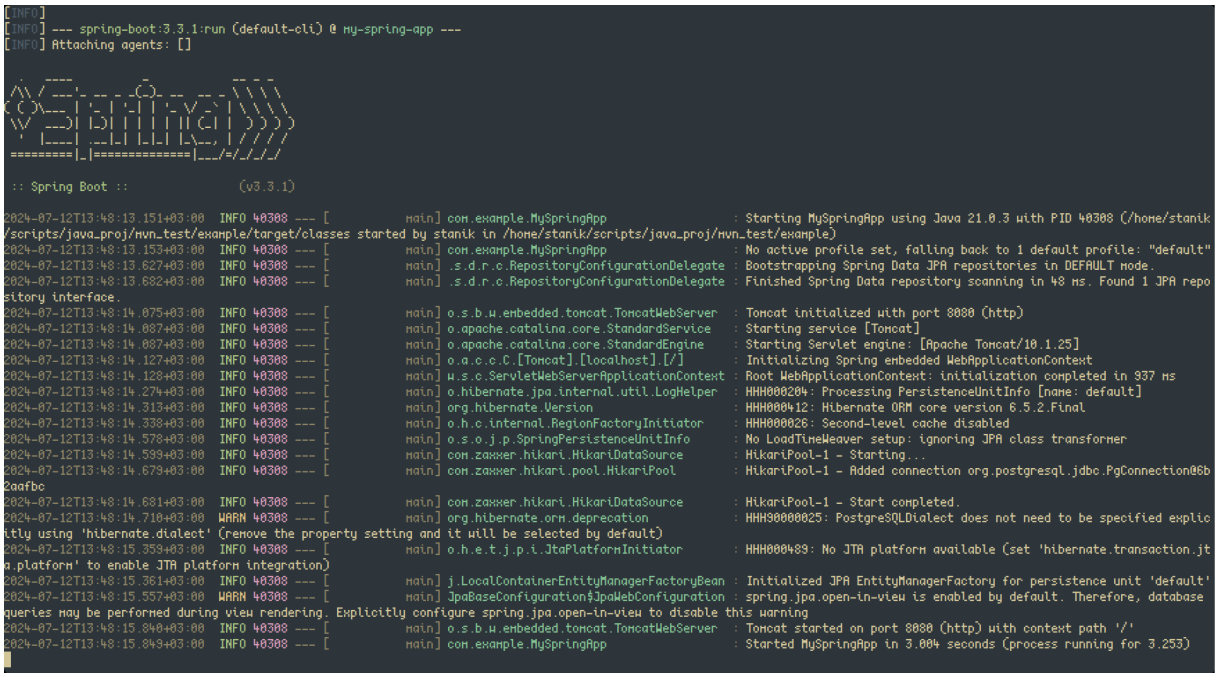

Рисунок 9 — Результат компиляции Java проекта, использующего Spring Framework

Перейдем по адресу localhost:8080/students и увидим данные обо всех студентах, указанных в БД (рис. 10).

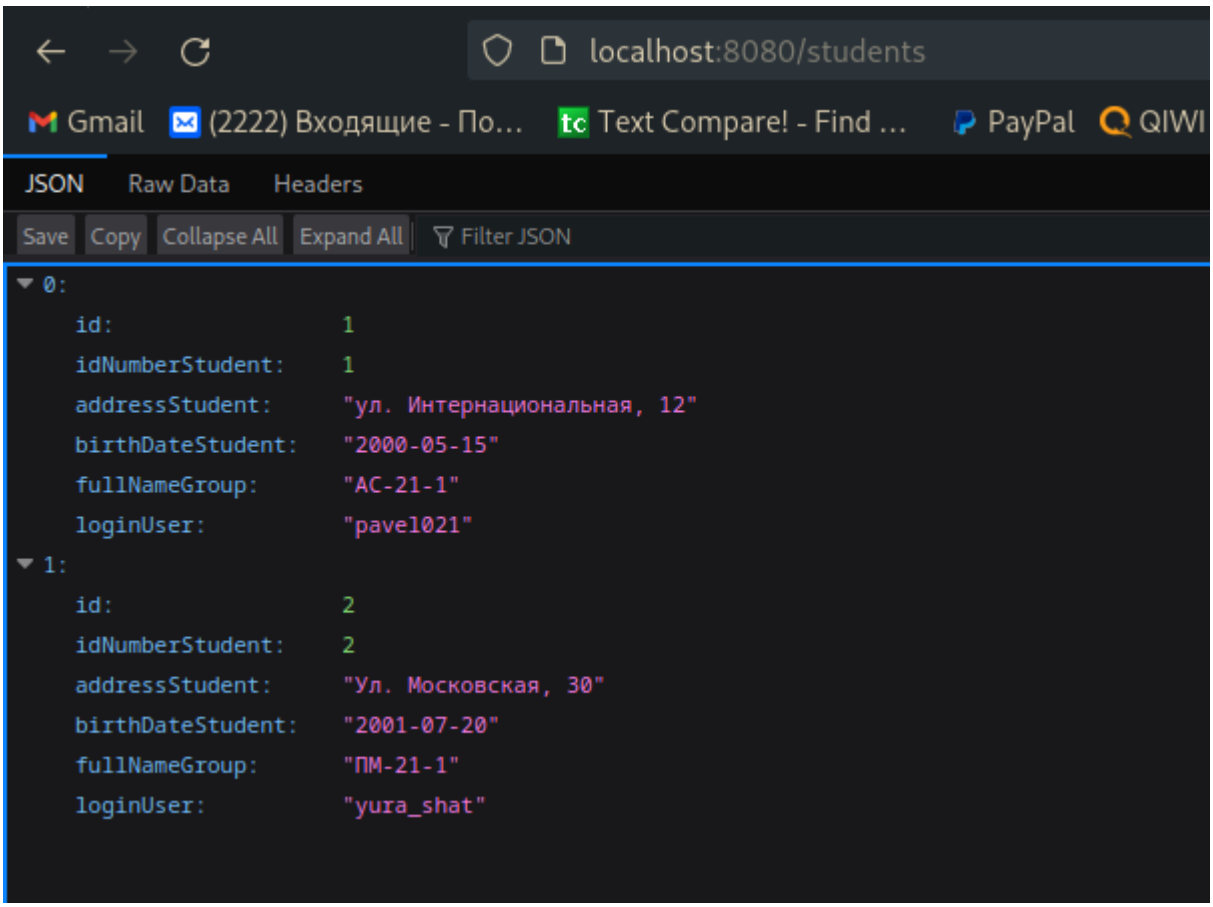

Рисунок 10 — Результат GET маппинга для запроса «students»

Для того, чтобы получить сведения о конкретном студенте, после students можно добавить id, который вернет ответную сущность с этим id (рис. 11).

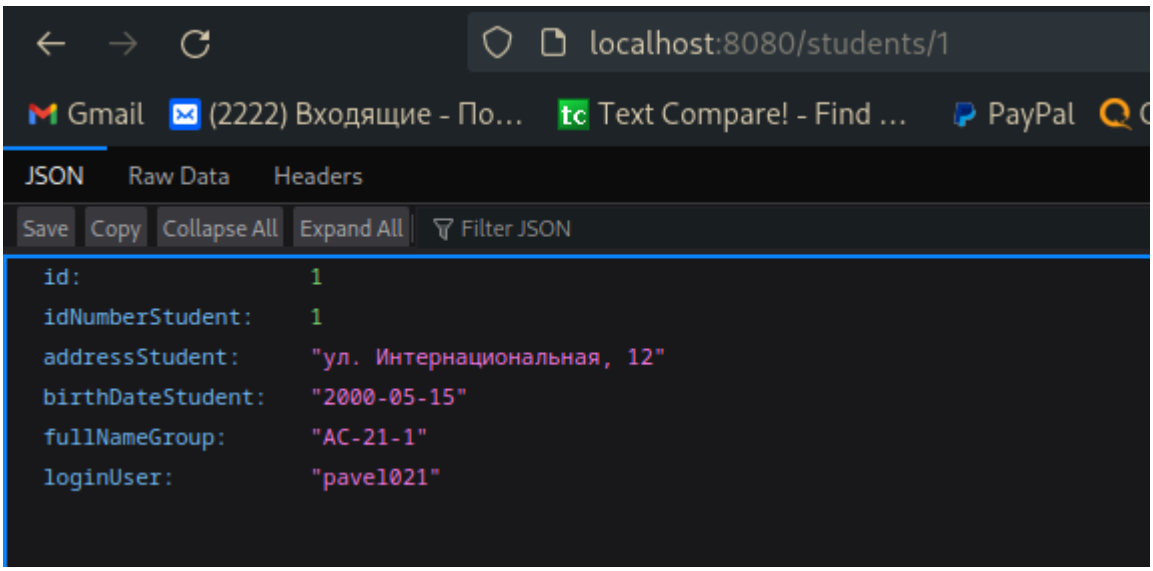

Рисунок 11 — Результат GET маппинга для запроса id студента

4. Описание выполнения программы по безопасности жизнедеятельности.

Во время прохождения практики в ООО «НЛМК-ИТ» г. Липецк был проведен инструктаж по технике безопасности при работе с персональным компьютером.

В инструктаже было сказано про основные вредные факторы при осуществлении трудовой деятельности за персональными компьютерами, а именно:

- высокая степень электромагнитного воздействия;

- высокий уровень наличия статического электричества; - низкая степень ионизации воздуха;

- нагрузки, связанные с длительным сидячим положением тела; - крайне высокая нагрузка на органы зрения;

- сопутствующие длительной сидячей работе факторы: болевые

- симптомы в пояснице и позвоночнике, венозная недостаточность, - стресс и депрессии.

Заключение

По завершении практики были получены навыки разработки скриптов на языке Java, навыки работы с Docker, Maven, Spring, опыт бэкенд разработки.

Библиографический список

1. Java API Specification [Электронный ресурс] / Режим доступа к данным: https://docs.oracle.com/javase/8/docs/api/ , свободный.

2. Spring Framework Documentation [Электронный ресурс] / Режим доступа к данным: https://docs.spring.io/spring-framework/reference/ , свободный

3. Руководство по Docker Compose [Электронный ресурс]/ Режим доступа к данным: https://docs.docker.com/compose/ , свободный

Приложение А

Исходный код игры «Жизнь»

Life.java

```
import java.util.Scanner;

public class Life {
   // use string instead of char because of compatibility
   private static final String nextStageBtn = "n";
   private static final String exitBtn = "x";
   private static boolean showDebugData = false;
   private static boolean inputMode = false;
   private static final int timeBtwnStages = 2000;       // only if input mode is deactivated
   private static float spawnChance = 0;                 // [0; 1]

   public static void main(String[] args) {
      for (String arg : args) {
         switch(arg){
            case "--show-debug-data":
               showDebugData = true;
               break;
            case "--input-mode":
               inputMode = true;
               break;
            default:
               System.out.println("Unrecognizable argument(s)");
               break;
         }
      }

      Scanner scanner = new Scanner(System.in);

      System.out.print("Enter area size n: ");
      int enteredVal = scanner.nextInt();
      int maxY = enteredVal;

      System.out.print("Enter area size m: ");
      enteredVal = scanner.nextInt();
      int maxX = enteredVal;

      System.out.print("Enter chance of the live cell [0.0 - 1.0]: ");
      float enteredSpawnChance = scanner.nextFloat();
      spawnChance = enteredSpawnChance;

      Map map = new Map(maxX, maxY, spawnChance);

      map.ShowDebugData = showDebugData;
      while (true)
      {
```

```
            ClearScreen();
            map.DisplayGrid();
            map.UpdateGrid();
            map.DisplayAddData();

            if (inputMode) {
                if (GetInput(scanner)) {
                    continue;
                }
                else {
                    break;
                }
            }
            else {
                try {
                    Thread.sleep(timeBtwnStages);
                } catch (InterruptedException e) {
                    Thread.currentThread().interrupt();
                    System.out.println("Thread was interrupted, failed to complete operation");
                }
            }
        }

    }

    private static boolean GetInput(Scanner scanner)
    {
        System.out.println("\nPress '" + nextStageBtn + "' to go to the next stage or press '" + exitBtn + "' to
exit. ");
        String input = scanner.next();

        if (input.equals(nextStageBtn)) {
            return true;
        }
        if (input.equals(exitBtn)) {
            return false;
        }

        scanner.close();

        return true;
    }

    private static void ClearScreen()
    {
        System.out.print("\033[H\033[2J");
        System.out.flush();
    }
}
```

Cell.java

```
public class Cell
{
    private boolean isAlive;
    // graphiics
    private final char symbolAlive = 'o';
    private final char symbolDeath = '#';



    // constructor
    public Cell(boolean isAlive)
    {
        this.isAlive = isAlive;
    }

    public boolean IsAlive()
    {
        return isAlive;
    }

    public void SetLifeMode(boolean isAlive)
    {
        this.isAlive = isAlive;
    }

    public char ToChar()
    {
        return isAlive ? symbolAlive : symbolDeath;
    }
}
```

Map.java

```
import java.util.Random;

public class Map
{
    public boolean ShowDebugData = false;
    public int LiveCount = 0;
    public int DeadCount = 0;



    private float spawnChance = 0.0f;
    private int w = 50;
    private int h = 50;
    private Cell[][] grid;



    public Map(int w, int h, float spawnChance)
    {
        this.w = w;
        this.h = h;
        this.spawnChance = spawnChance;
        grid = new Cell[w][h];
        InitGrid();
    }

    public void DisplayGrid()
    {
        System.out.print("\n");
        for (int x = 0; x < w; x++) {
            for (int y = 0; y < h; y++) {
                System.out.print(grid[x][y].ToChar());
            }
```

```
        System.out.print("\n");
    }
}

public void DisplayAddData()
{
    System.out.println("\nLive count: " + LiveCount);
    System.out.println("Dead count: " + DeadCount + "\n");
}

public void UpdateGrid()
{
    LiveCount = 0;
    DeadCount = 0;

    Cell[][] newGrid = new Cell[w][h];
    for (int x = 0; x < w; x++) {
        for (int y = 0; y < h; y++) {
            boolean isAlive = grid[x][y].IsAlive();
            int liveNeighbors = CountLiveNeighbors(x, y);

            // main logic
            if (isAlive) {
                // loneless or super population death -> if not, then keep life
                newGrid[x][y] = new Cell(liveNeighbors < 2 || liveNeighbors > 3 ? false : true);

                LiveCount++;
            }
            else {
                // if there some neighbors around dead cell, give this cell life -> if not, then kill it
                newGrid[x][y] = new Cell(liveNeighbors == 3 ? true : false);
```

```
                DeadCount++;
            }
        }
    }

    grid = newGrid;
}


private void InitGrid()
{
    Random rand = new Random();
    for (int x = 0; x < w; x++) {
        for (int y = 0; y < h; y++) {
            grid[x][y] = new Cell(rand.nextFloat() < spawnChance);
        }
    }
}


private int CountLiveNeighbors(int x, int y)
{
    int liveNeighbors = 0;
    int[] directions = {-1, 0, 1};

    // dx && dy:
    // [-1;  1] [0;  1] [1;  1]
    // [-1;  0] [0;  0] [1;  0]
    // [-1; -1] [0; -1] [1; -1]
    // where [0; 0] is current cell
    for (int dx : directions) {
        for (int dy : directions) {
```

```
            if (!(dx == 0 && dy == 0)) { // not a cell itself
                int neighborX = x + dx;
                int neighborY = y + dy;

                if (neighborX >= 0 && neighborX < w && neighborY >= 0 && neighborY < h) {
                    if (grid[neighborX][neighborY].IsAlive()) {
                        liveNeighbors++;
                    }
                }
            }
        }
        if (ShowDebugData) {
            System.out.print(liveNeighbors);
        }
        return liveNeighbors;
    }
}
```

Приложение Б

Подключение к локальной БД, используя Java SQL API

DBCheck.java

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;
import java.sql.ResultSet;

public class DBCheck {
   static final String URL = "jdbc:postgresql://localhost/lk";
   static final String USER = "postgres";
   static final String PASSWORD = "password";

   static Connection conn = null;
   static Statement stmt = null;
   static ResultSet rs = null;

   public static void main(String[] args) {
      try {
         conn = DriverManager.getConnection(URL, USER, PASSWORD);
         if (conn != null) {
            stmt = conn.createStatement();
            rs = stmt.executeQuery("SELECT current_database();");
            if (rs.next()) {
               String currentDatabase = rs.getString(1);
               System.out.println("Connected to the database: " + currentDatabase);
            }
         } else {
            System.out.println("Failed to make connection!");
         }
      } catch (SQLException e) {
         System.out.println("SQL Exception: " + e.getMessage());
      }
   }
}
```

Приложение В

Pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
4.0.0.xsd">
   <modelVersion>4.0.0</modelVersion>

   <groupId>com.example</groupId>
   <artifactId>my-spring-app</artifactId>
   <version>1.0-SNAPSHOT</version>

   <parent>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-parent</artifactId>
      <version>3.3.1</version>
      <relativePath /> <!-- lookup parent from repository -->
   </parent>

   <properties>
      <java.version>21</java.version>
   </properties>

   <dependencies>
      <!-- Spring Boot Starter Web -->
      <dependency>
         <groupId>org.springframework.boot</groupId>
         <artifactId>spring-boot-starter-web</artifactId>
      </dependency>

      <!-- Spring Boot Starter Data JPA -->
      <dependency>
         <groupId>org.springframework.boot</groupId>
         <artifactId>spring-boot-starter-data-jpa</artifactId>
      </dependency>

      <!-- PostgreSQL Driver -->
      <dependency>
         <groupId>org.postgresql</groupId>
         <artifactId>postgresql</artifactId>
         <version>42.2.24</version>
      </dependency>

      <!-- Spring Boot Starter Test -->
      <dependency>
         <groupId>org.springframework.boot</groupId>
         <artifactId>spring-boot-starter-test</artifactId>
         <scope>test</scope>
      </dependency>
```

```
        <dependency>
          <groupId>junit</groupId>
          <artifactId>junit</artifactId>
          <version>3.8.1</version>
          <scope>test</scope>
        </dependency>

        <!-- JavaX Persistence -->
        <dependency>
          <groupId>jakarta.persistence</groupId>
          <artifactId>jakarta.persistence-api</artifactId>
        </dependency>

    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
</project>
```

Приложение Г

Student.java

```
package com.example;

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.Column;

import java.time.LocalDate;

@Entity
public class Student {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "id_number_student")
    private int idNumberStudent;

    @Column(name = "address_student")
    private String addressStudent;

    @Column(name = "birth_date_student")
    private LocalDate birthDateStudent;

    @Column(name = "full_name_group")
    private String fullNameGroup;

    @Column(name = "login_user")
    private String loginUser;

    public Student() {}

    public Student(int idNumberStudent, String addressStudent, LocalDate birthDateStudent,
            String fullNameGroup, String loginUser, String name) {
        this.idNumberStudent = idNumberStudent;
        this.addressStudent = addressStudent;
        this.birthDateStudent = birthDateStudent;
        this.fullNameGroup = fullNameGroup;
        this.loginUser = loginUser;
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }
```

```
    public int getIdNumberStudent() {
        return idNumberStudent;
    }

    public void setIdNumberStudent(int idNumberStudent) {
        this.idNumberStudent = idNumberStudent;
    }

    public String getAddressStudent() {
        return addressStudent;
    }

    public void setAddressStudent(String addressStudent) {
        this.addressStudent = addressStudent;
    }

    public LocalDate getBirthDateStudent() {
        return birthDateStudent;
    }

    public void setBirthDateStudent(LocalDate birthDateStudent) {
        this.birthDateStudent = birthDateStudent;
    }

    public String getFullNameGroup() {
        return fullNameGroup;
    }

    public void setFullNameGroup(String fullNameGroup) {
        this.fullNameGroup = fullNameGroup;
    }

    public String getLoginUser() {
        return loginUser;
    }

    public void setLoginUser(String loginUser) {
        this.loginUser = loginUser;
    }
}
```

StudentRepository.java

```
package com.example;

import org.springframework.data.jpa.repository.JpaRepository;

public interface StudentRepository extends JpaRepository<Student, Long> {
}
```

StudentController.java

```
package com.example;

import org.springframework.beans.factory.annotation.Autowired;
```

```
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;
import java.util.Optional;

@RestController
@RequestMapping("/students")
public class StudentController {

    @Autowired
    private StudentRepository studentRepository;
    private final String getMappingIdTagName = "id";


    @GetMapping
    public List<Student> getAllStudents() {
        return studentRepository.findAll();
    }

    @GetMapping("/{id}")
    public ResponseEntity<Student> getStudent(@PathVariable Long id) {
        Optional<Student> student = studentRepository.findById(id);

        return student.isPresent() ? ResponseEntity.ok(student.get()) : ResponseEntity.notFound().build();
    }

    @PostMapping
    public Student createStudent(@RequestBody Student student) {
        return studentRepository.save(student);
    }
}
```