

## СОДЕРЖАНИЕ

ВВЕДЕНИЕ .....	2
Задание № 1 «В среде IntelliJ IDEA Ultimate и используя Spring Framework Java написать клиент серверное приложение с микросервисной архитектурой» ....	3
Задание № 2 «Сервер должен принимать два аргумента и возвращать их сумму/разность» .....	10
Задание № 3 «Сгенерировать веб-интерфейс для ваших SpringBoot API с помощью Swagger 3.0.0» .....	12
Задание № 4 «Написать тесты на проект» .....	15
Задание № 5 «Подключить in-memory БД и сохранять туда все результаты расчетов калькулятора» .....	18
ЗАКЛЮЧЕНИЕ .....	27

## ВВЕДЕНИЕ

В данной практической работе я буду заниматься разработкой клиент-серверного приложения на языке Java с использованием Spring Framework. Моя цель - создать микросервисную архитектуру, где сервер с применением Spring Framework, принимает два числа от клиента и возвращает их сумму или разность. Также познакомлюсь с веб-интерфейсом Swagger и тестами. Кроме того, планируется подключить внутреннюю базу данных, где будут сохраняться результаты вычислений.

Задание № 1 «В среде IntelliJ IDEA Ultimate и используя Spring Framework Java написать клиент серверное приложение с микросервисной архитектурой»

Начало установки программы «IntelliJ IDEA» для дальнейшей работы с кодом (Рисунок 1).

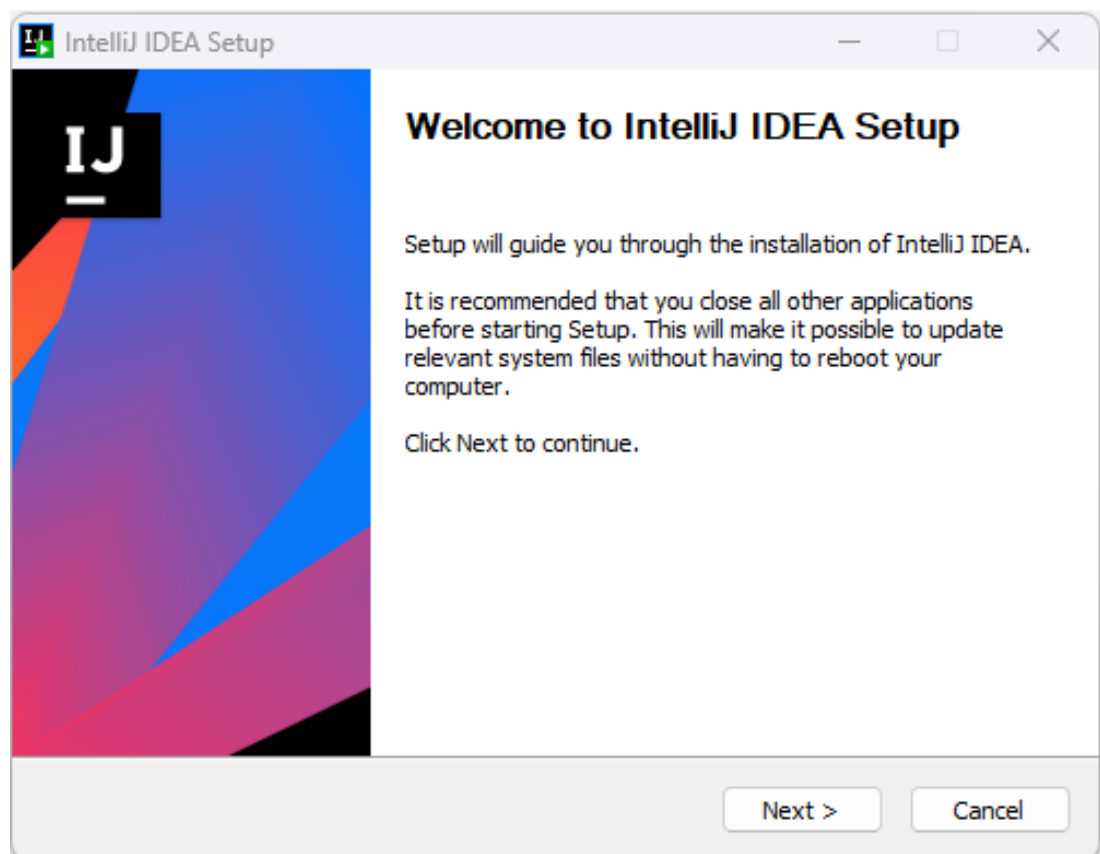


Рисунок 1– начало установка IntelliJ IDEA

Определение директории установки программы IntelliJ IDEA (Рисунок 2).

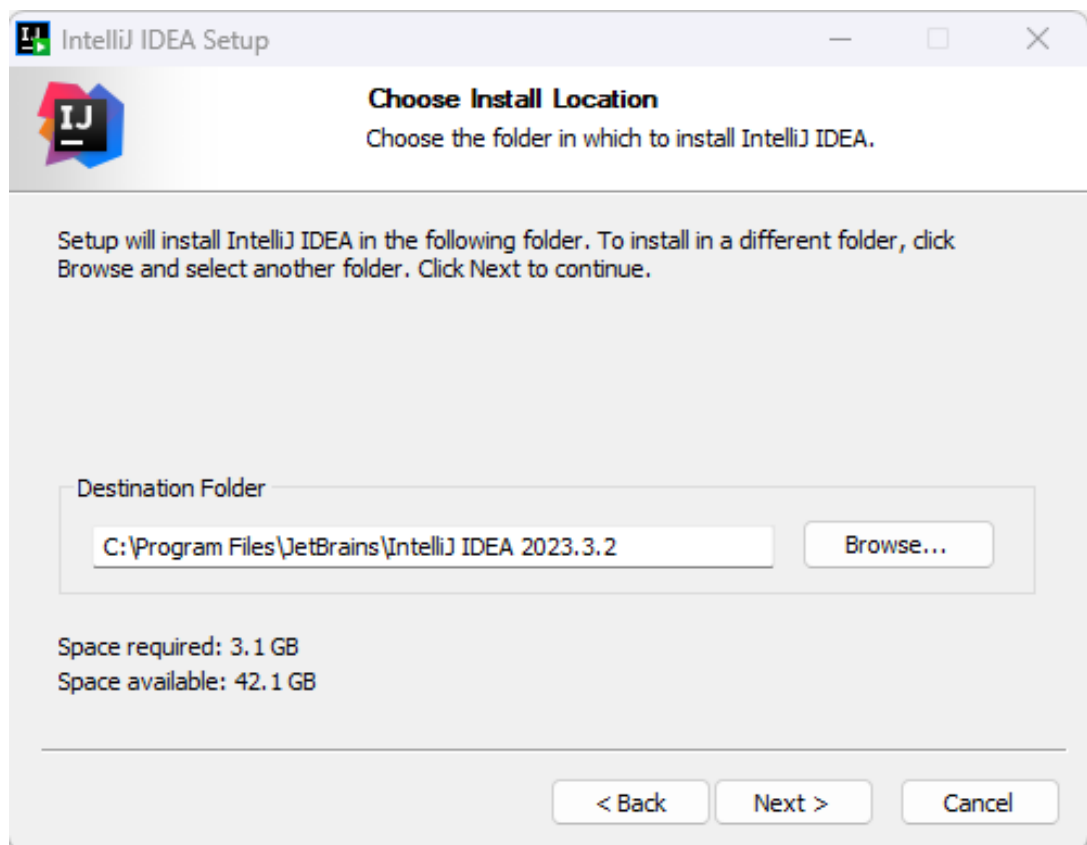


Рисунок 2 – выбор папки установки

Установка опции ‘Создать ярлык на рабочем столе’, а также создания ассоциации .java (Рисунок 3).

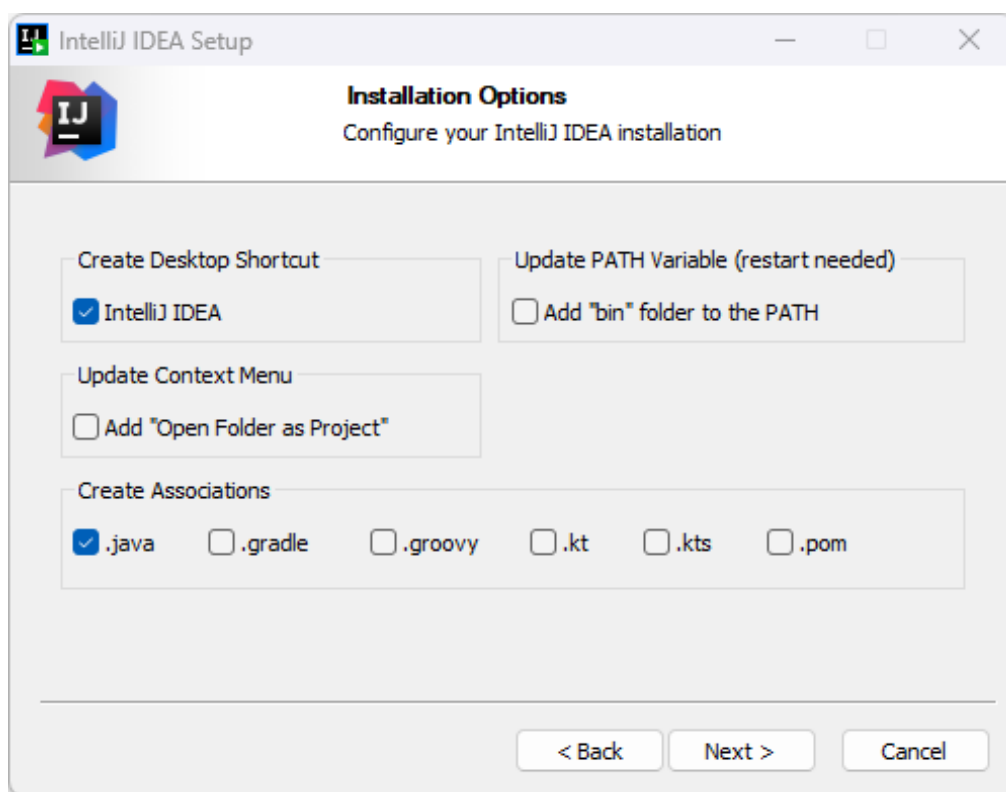


Рисунок 3 – конфигурация программы

Название папки, где будут располагаться файлы программы (Рисунок 4).

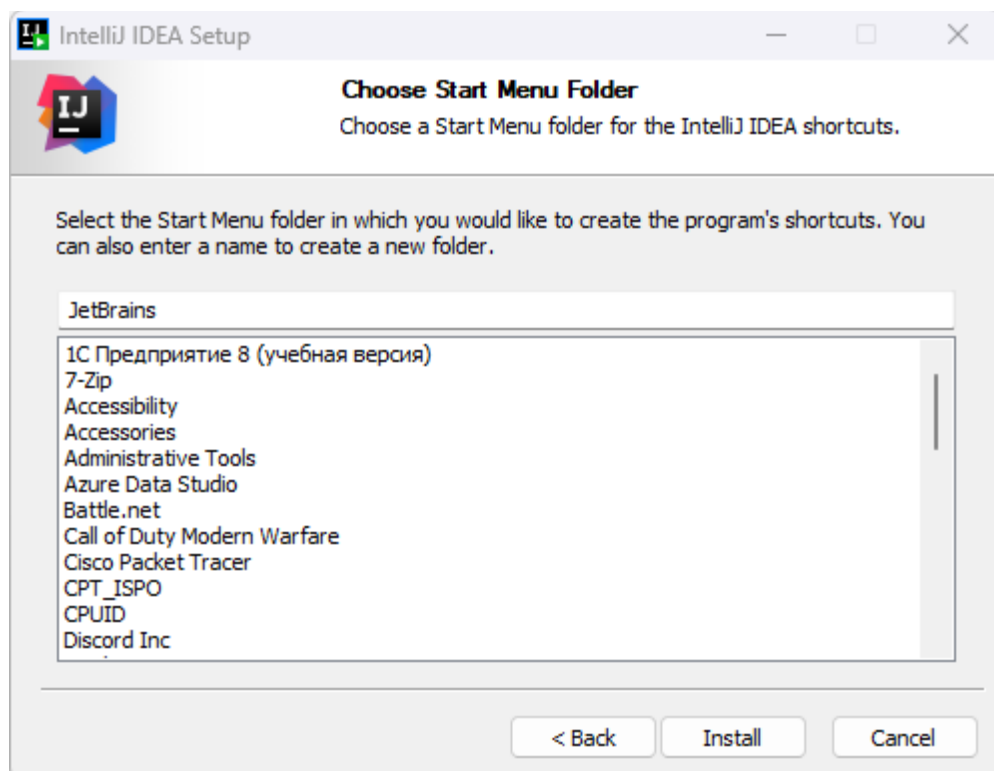


Рисунок 4 – выбор названия папки

Процесс установки программы на компьютер (Рисунок 5).

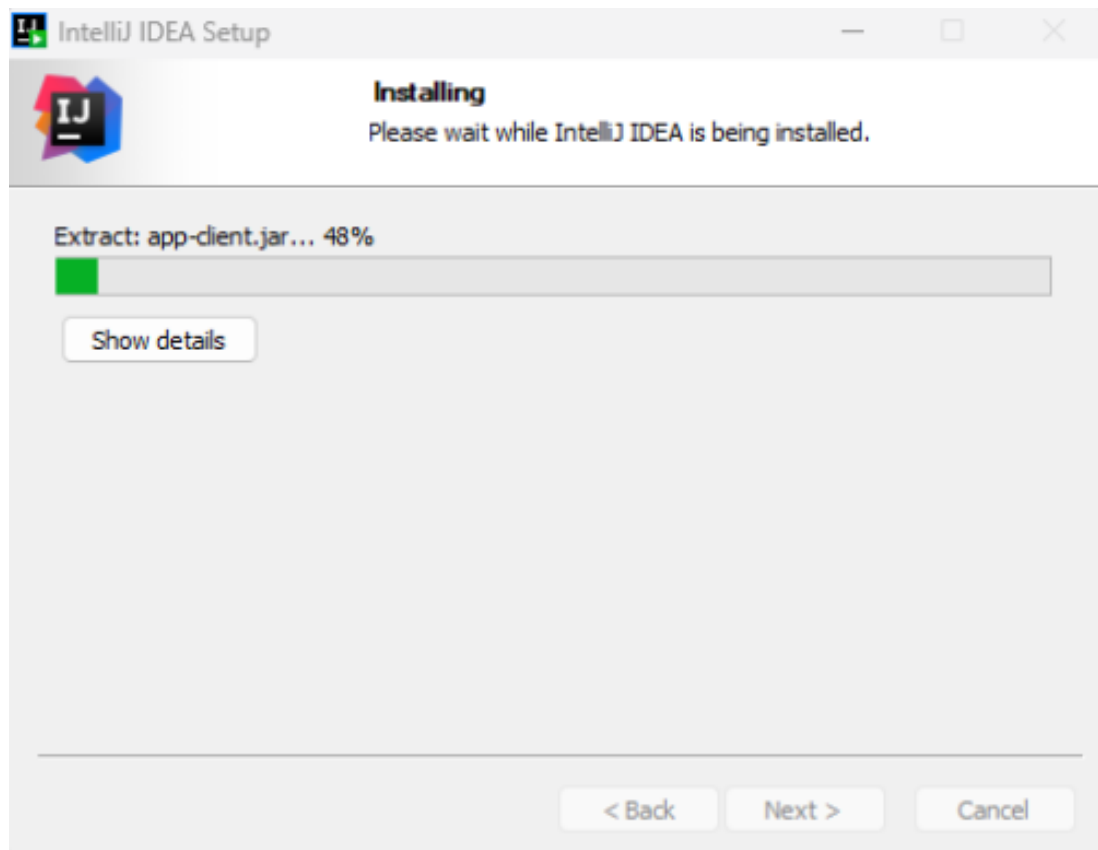


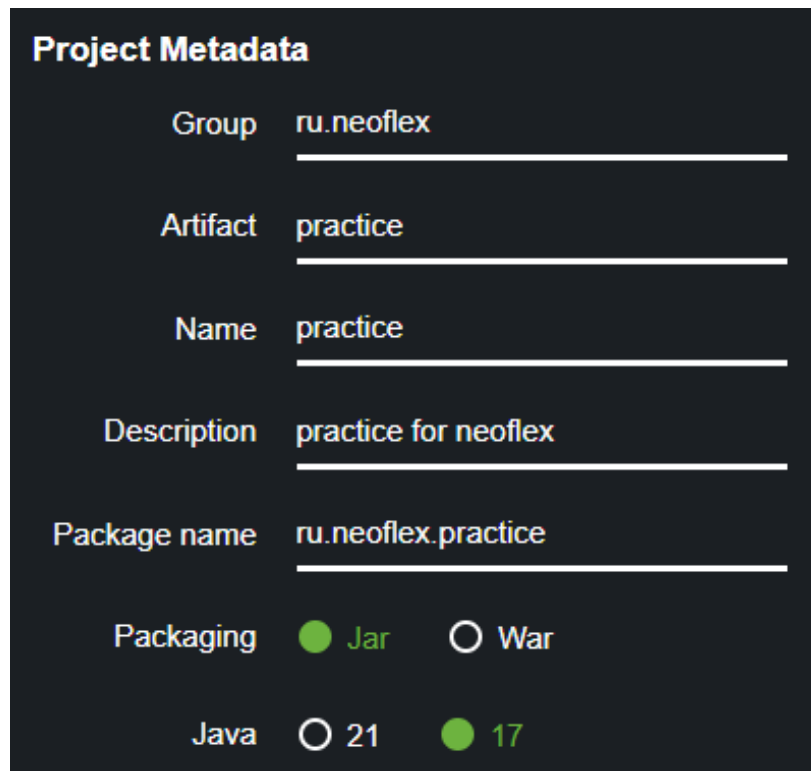
Рисунок 5 – установка программы

Конфигурация проекта через сайт <https://start.spring.io/>. Выбор проекта, языка проекта, версии Spring Bot (Рисунок 6).

Project	Language
<input type="radio"/> Gradle - Groovy	<input checked="" type="radio"/> Java <input type="radio"/> Kotlin
<input type="radio"/> Gradle - Kotlin	<input type="radio"/> Groovy
<input checked="" type="radio"/> Maven	
<b>Spring Boot</b>	
<input type="radio"/> 3.3.0 (SNAPSHOT)	<input type="radio"/> 3.2.2 (SNAPSHOT)
<input checked="" type="radio"/> 3.2.1	<input type="radio"/> 3.1.8 (SNAPSHOT) <input type="radio"/> 3.1.7

Рисунок 6 – первая часть конфигурации

После выбора основных настроек проекта происходит заполнение описания проекта, а также выбор расширения и версии java (Рисунок 7).

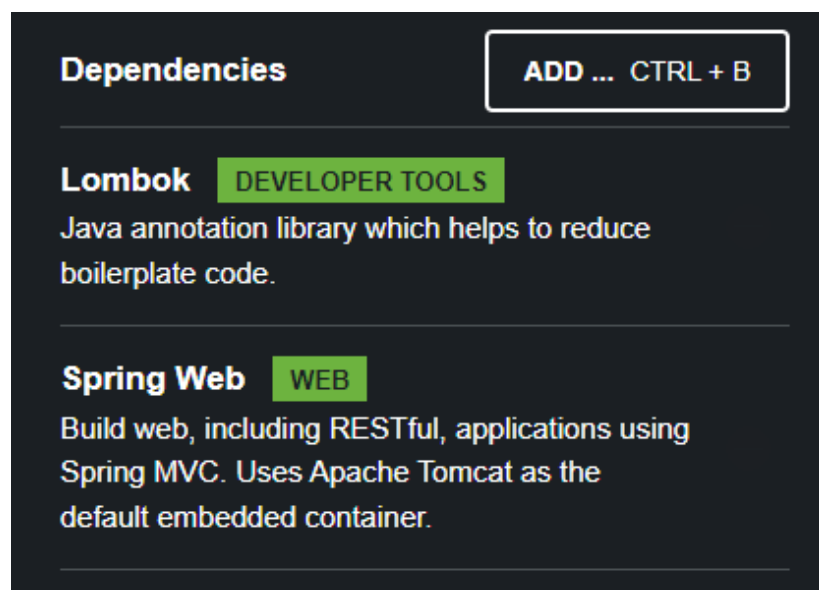


**Project Metadata**

Group	<input type="text" value="ru.neoflex"/>
Artifact	<input type="text" value="practice"/>
Name	<input type="text" value="practice"/>
Description	<input type="text" value="practice for neoflex"/>
Package name	<input type="text" value="ru.neoflex.practice"/>
Packaging	<input checked="" type="radio"/> Jar <input type="radio"/> War
Java	<input type="radio"/> 21 <input checked="" type="radio"/> 17

Рисунок 7 – вторая часть конфигурации

Добавление зависимости Lombok для автоматизации методов, с целью упрощения процесса разработки и повышения читаемости кода. Также требуется добавление зависимости Spring Web для разработки веб-приложений с использованием фреймворка Spring (Рисунок 8).



**Dependencies** ADD ... CTRL + B

---

**Lombok** DEVELOPER TOOLS  
Java annotation library which helps to reduce boilerplate code.

---

**Spring Web** WEB  
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

---

Рисунок 8 – добавление зависимостей

Генерация конфигурации проекта при нажатии на кнопку Generate (рисунок 9).

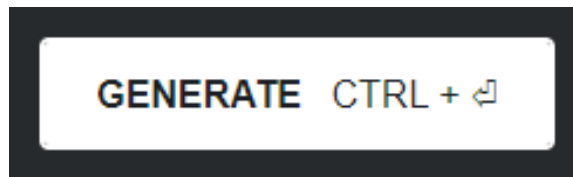


Рисунок 9 – генерация конфигурации проекта

Распаковка архива со сгенерированной конфигурации в папку проекта (Рисунок 10).

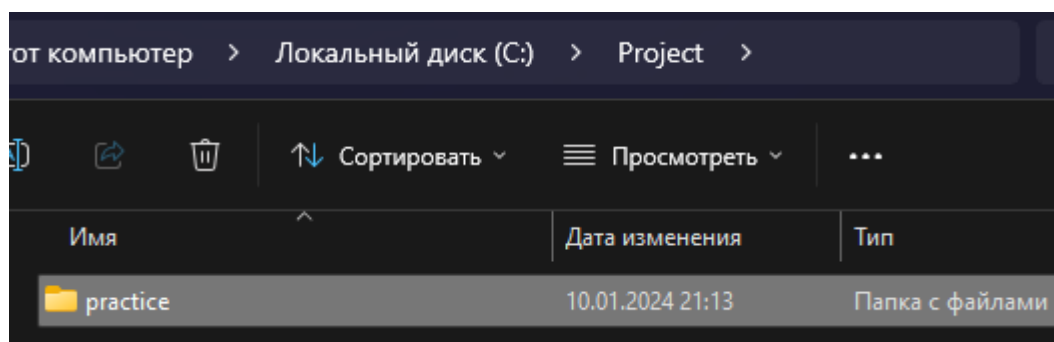


Рисунок 10 – распаковка архива

Создание папки controller в директории src/main/java/ru/neoflex/practice (Рисунок 11).

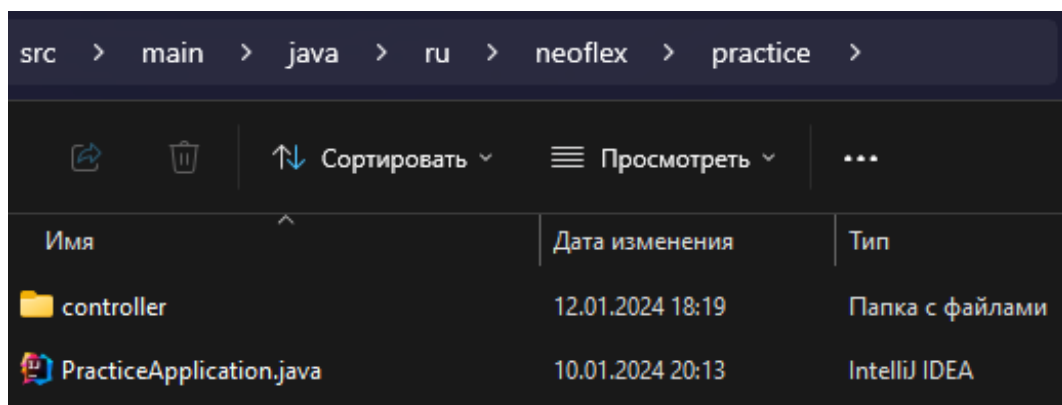


Рисунок 11 – результат метода сложения



Создание файла CalcController с расширением .java в папке controller (Рисунок 12).

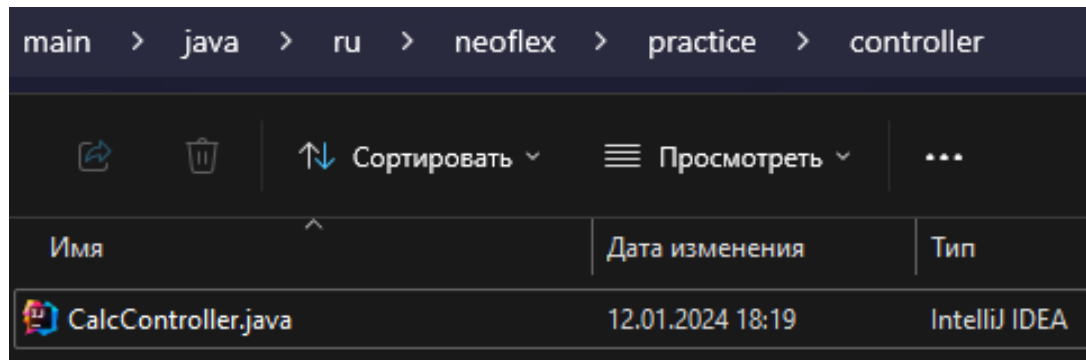


Рисунок 12 – создание файла CalcController.java

Установка SDK для обеспечения необходимых инструментов для разработки и компиляции Java-приложений (Рисунок 13).

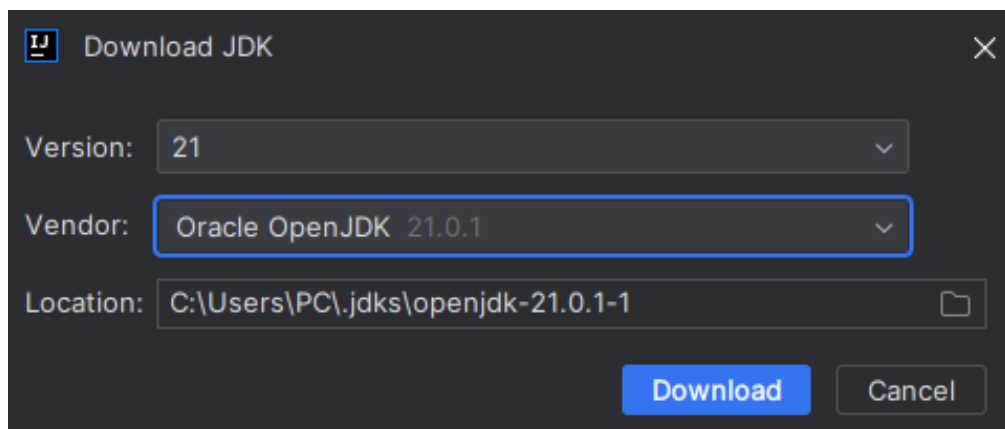


Рисунок 13 – установка SDK

## Задание № 2 «Сервер должен принимать два аргумента и возвращать их сумму/разность»

Создание кода. Первым делом импортируются необходимые классы из библиотеки фреймворка Spring. Класс `CalcController` будет обрабатывать HTTP запросы. Метод `'add'` выполняет операцию сложения указанных чисел в URL запросе. Метод принимает два числа и возвращает их сумму. Метод `'subtract'` выполняет операцию вычитания указанных чисел в URL запросе. Метод принимает два числа и возвращает их разность (Рисунок 14).

```
troller.java ×
package ru.neoflex.practice.controller;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class CalcController {
    // сложение
    @GetMapping("/add/{a}/{b}")
    public int add(@PathVariable("a") int a, @PathVariable("b") int b) { return a + b; }
    // вычитание
    @GetMapping("/subtract/{a}/{b}")
    public int subtract(@PathVariable("a") int a, @PathVariable("b") int b) { return a - b; }
}
```

Рисунок 14 – код для калькулятора

Запуск приложения при нажатии на зелёную кнопку Run (Рисунок 15).

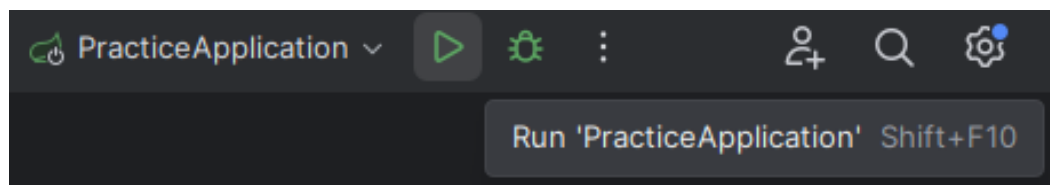


Рисунок 15 – запуск приложения

Для тестирования метода сложения двух чисел необходимо в адресной строки веб-браузера ввести <http://localhost:8080/add/10/7>, а для метода разности двух чисел ввести <http://localhost:8080/subtract/5/2> (Рисунок 16-17).

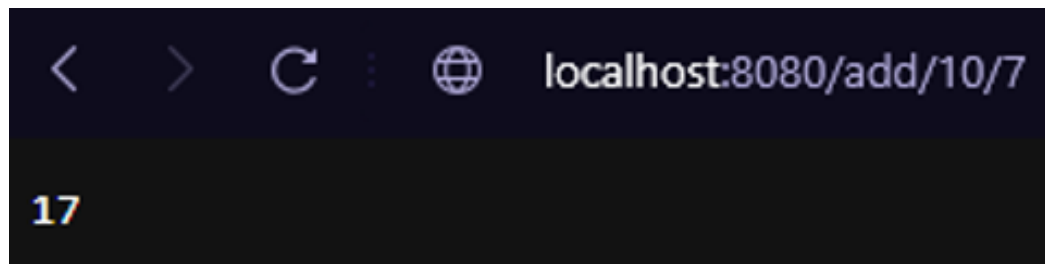


Рисунок 16 – результат метода сложения

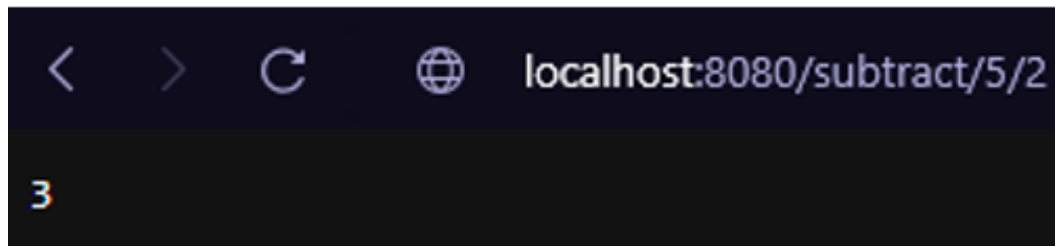


Рисунок 17 – результат метода разности

### Задание № 3 «Сгенерировать веб-интерфейс для ваших SpringBoot API с помощью Swagger 3.0.0»

Для интеграции Swagger в проект, необходимо добавить соответствующую зависимость в файл pom.xml. В данном случае, выбрана версия 2.0.2 (Рисунок 18).

```
<dependency>
  <groupId>org.springdoc</groupId>
  <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
  <version>2.0.2</version>
</dependency>
```

Рисунок 18 – код в pom.xml

Обновление изменений в файле pom.xml при нажатии на кнопку 'Load Maven Changes' (Рисунок 19).

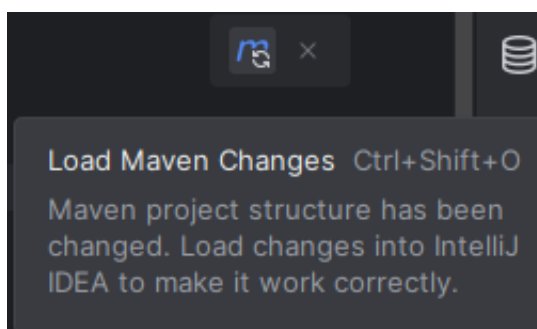


Рисунок 19 – обновление изменений pom.xml

Для доступа к веб-интерфейсу Swagger необходимо открыть веб-браузер и перейти по адресу <http://localhost:8080/swagger-ui/index.html>. Далее выбирается нужный метод из предоставленного списка, заполняются соответствующие поля для ввода чисел и выводится результат операции (Рисунок 20-22).

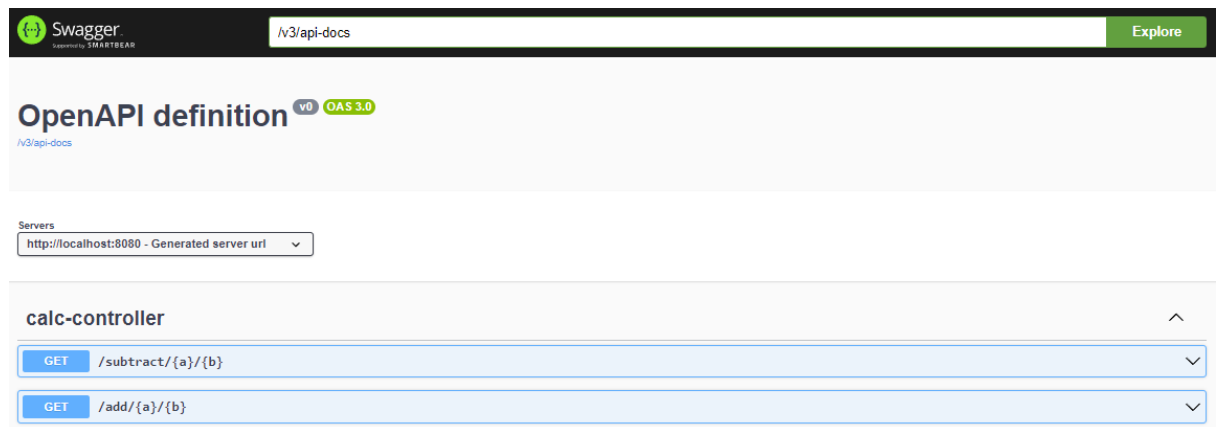


Рисунок 20 – главный вид веб-страницы

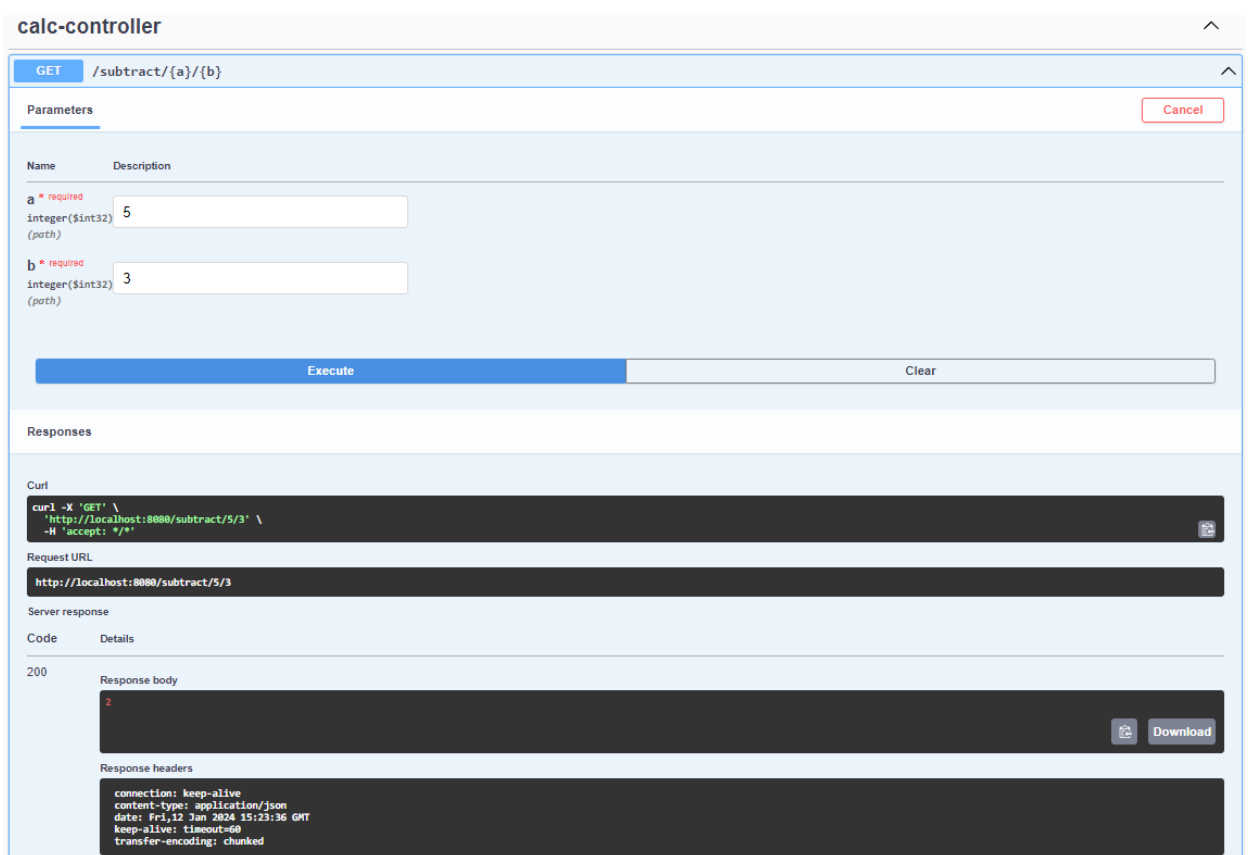


Рисунок 21 – результат метода разности

calc-controller

GET

/subtract/{a}/{b}

GET

/add/{a}/{b}

Parameters

Cancel

Name	Description
<b>a</b> <small>* required</small>	<input type="text" value="7"/>
<small>integer(\$int32)</small>	<small>(path)</small>
<b>b</b> <small>* required</small>	<input type="text" value="7"/>
<small>integer(\$int32)</small>	<small>(path)</small>

Execute

Clear

Responses

Curl

```
curl -X 'GET' \
  'http://localhost:8080/add/7/7' \
  -H 'accept: */*'
```

Request URL

```
http://localhost:8080/add/7/7
```

Server response

Code	Details
200	<div><div>Response body</div><div><pre>14</pre></div><div><div>Download</div></div></div> <div><div>Response headers</div><div><pre>connection: keep-alive content-type: application/json date: Fri, 12 Jan 2024 15:24:23 GMT keep-alive: timeout=60 transfer-encoding: chunked</pre></div></div>

Рисунок 22 – результат метода сложения

#### Задание № 4 «Написать тесты на проект»

Создание пакета controller в директории src\test\java\ru.neoflex.practice (Рисунок 23)

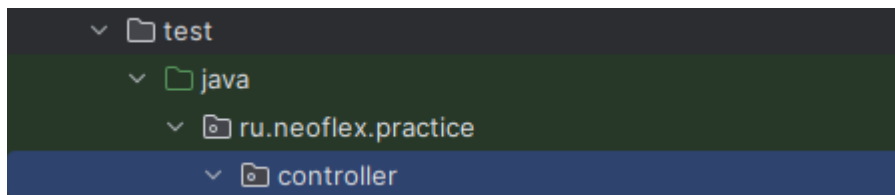


Рисунок 23 – создание пакета controller

Создание файла CalcTest.java пакете controller (Рисунок 24).

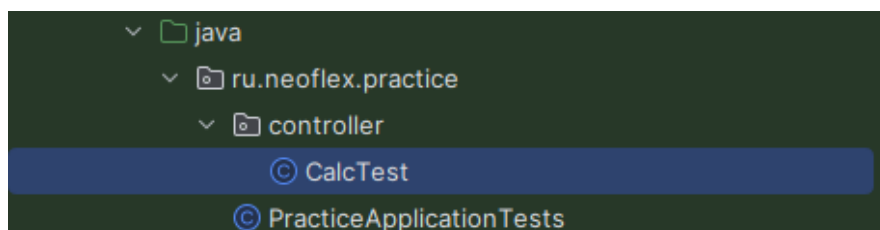


Рисунок 24 – создание файла CalcTest.java

В файле pom.xml требуется внести зависимость на JUnit Jupiter API, предоставляющую доступ к библиотеке для создания и выполнения тестов. Выбрана версия 5.10.0 (Рисунок 25).

```
<dependency>
  <groupId>org.junit.jupiter</groupId>
  <artifactId>junit-jupiter-api</artifactId>
  <version>5.10.0</version>
  <scope>test</scope>
</dependency>
```

Рисунок 25 – код в pom.xml

В представленном коде реализованы тесты для методов сложения и вычитания. Создается класс, содержащий тесты для методов сложения и вычитания. В каждом тесте задаются входные параметры и ожидаемые результат, затем запускаются тесты (Рисунок 26).

```

package ru.neoflex.practice.controller;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;

class CalcTest {

    2 usages
    CalcController calcController = new CalcController();

    1 usage
    void testAdd(int a, int b, int expected) {
        int add = calcController.add(a, b);
        Assertions.assertEquals(expected, add);
    }

    1 usage
    void testSubtract(int a, int b, int expected) {
        int subtract = calcController.subtract(a, b);
        Assertions.assertEquals(expected, subtract);
    }

    @Test
    void add() {
        testAdd(a: 0, b: 5, expected: 6);
    }

    @Test
    void subtract() { testSubtract(a: -3, b: 2, expected: -5); }
}

```

Рисунок 26 – код для теста

Тест считается успешным, если полученный результат совпадает с ожидаемым, в ином случае тест провален. Результаты примерных тестов представлены на рисунках (Рисунок 27-28).

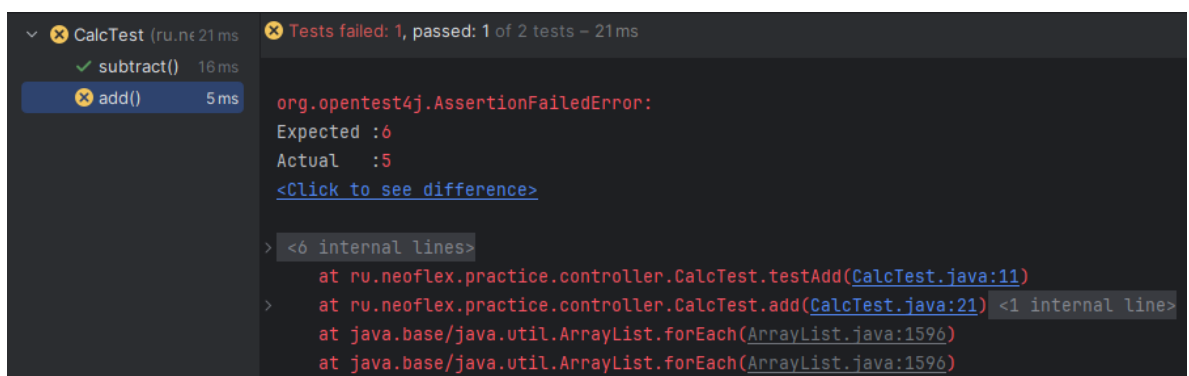


Рисунок 27 – тест не пройден



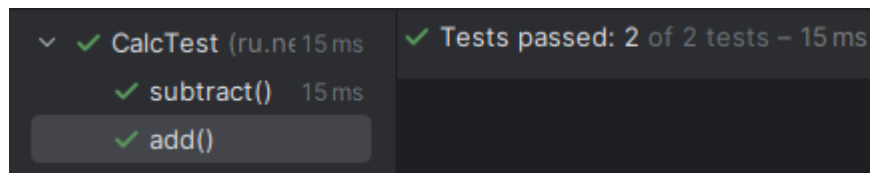


Рисунок 28 – тесты пройдены успешно

## Задание № 5 «Подключить in-memory БД и сохранять туда все результаты расчетов калькулятора»

Для сохранения расчетов калькулятора будет использоваться встроенная база данных H2, написанная на языке Java. Данная база данных довольно проста для использования и внедрения в приложение. База данных H2 предоставляет удобный веб-интерфейс, который позволяет легко и удобно взаимодействовать с базой данных. В ней можно выполнять SQL-запросы и следить за данными. Для начала в 'ru.neoflex.practice' необходимо создать пакеты 'model' и 'repository' (Рисунок 29).

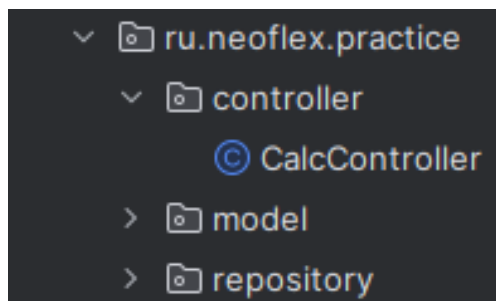


Рисунок 29 – создание двух пакетов

Создание класса 'CalculatingResults' в пакете 'model' (Рисунок 30).

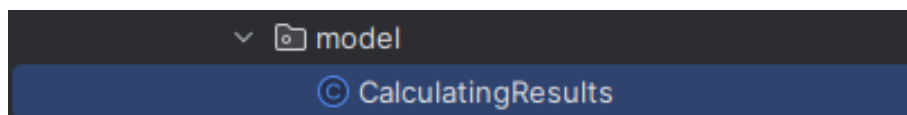


Рисунок 30 – создание класса 'CalculatingResults'

Создания кода, который служит для представления и хранения результатов вычислений в БД H2 с использованием Java Persistence API (JPA). В нашем случае поле 'id' будет использоваться как первичный ключ с автоматической генерацией значений. Более детальное и точное пояснение к каждой строчке кода представлено на рисунке кода (Рисунок 31).

```

package ru.neoflex.practice.model;

// добавление аннотаций
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.Id;

11 usages
@Entity
public class CalculatingResults {
    @Id @GeneratedValue private long id; // определение первичного ключа с автогенерацией значений
    2 usages
    private int result; // хранения результата вычислений

    public CalculatingResults() {} // создания экземпляра класса

    4 usages
    public CalculatingResults(int result) { this.result = result; // установка значения результата }

    no usages
    public int getResult() { return result; // получение значения результата }
}

```

Рисунок 31 – код с комментариями

Создание класса ‘Repository’ в пакете ‘repository’ (Рисунок 32).

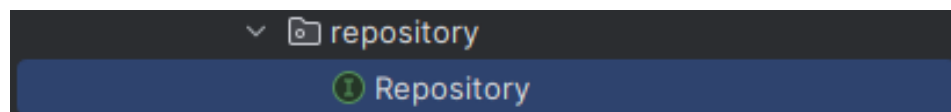


Рисунок 32 – создание класса ‘Repository’

Создание кода, который создает интерфейс репозитория, предназначенного для взаимодействия с базой данных. Данный код облегчает выполнение основных операций работы с данными: сохранение, чтение, обновление и удаление (Рисунок 33).

```

package ru.neoflex.practice.repository;

import org.springframework.data.jpa.repository.JpaRepository;
import ru.neoflex.practice.model.CalculatingResults;

4 usages
public interface Repository extends JpaRepository<CalculatingResults, Long> {}

```

Рисунок 33 – код для репозитория

Далее необходимо внести дополнения в код калькулятора с целью интеграции репозитория для сохранения результатов вычислений. Внесенные изменения в код позволяют не только выполнять операции сложения и вычитания через HTTP-запросы, но и сохранять полученные результаты в базе данных. Добавлена возможность получения списка всех предыдущих результатов. Таким образом, код обеспечивает взаимодействие с базой данных для хранения и извлечения результатов вычислений. Комментарии к каждой новой строке представлены на рисунке с кодом (Рисунок 34).

```
package ru.neoflex.practice.controller;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RestController;

// импорт новых аннотаций
import ru.neoflex.practice.model.CalculatingResults;
import ru.neoflex.practice.repository.Repository;

import java.util.List; // импорт списка для обработки результатов

@RestController
public class CalcController {

    4 usages
    private final Repository calculationRepository; // поле хранения репозитория для взаимодействия с БД
    // конструктор с параметром - репозиторием для работы с данными
    public CalcController(Repository calculationRepository) {
        // запись репозитория в поле класса
        this.calculationRepository = calculationRepository;
    }

    // сложение
    @GetMapping(@PathVariable("a") int a, @PathVariable("b") int b) {
        // сохранение результата сложения в БД
        calculationRepository.save(new CalculatingResults(a + b));
        return a + b;
    }

    // вычитание
    @GetMapping(@PathVariable("a") int a, @PathVariable("b") int b) {
        // сохранение результата разности в БД
        calculationRepository.save(new CalculatingResults(a - b));
        return a - b;
    }

    // get-запрос для получения списка всех сохраненных результатов вычислений
    @GetMapping("/calcList")
    public List<CalculatingResults> getAllCalculations() { return List.copyOf(calculationRepository.findAll()); }
```

Рисунок 34 – обновленный код калькулятора

Перед началом работы с базой данных H2 следует настроить файл конфигурации 'application.properties'. В этом файле указываются различные параметры, например порт, настройки подключения к БД и не только. В нашем случае нужно добавить только настройки подключения к БД. На рисунке с кодом представлены комментарии к каждой строчке настройки (Рисунок 35).

```
# класс драйвера БД H2
spring.datasource.driverClassName=org.h2.Driver

# диалект Hibernate для работы с H2
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect

# отключение генерации уникального имени для БД
spring.datasource.generate-unique-name=false

# установка названия БД
spring.datasource.name=neoflexDB

# установка имени пользователя для подключения к БД
spring.datasource.username=neoflexAdmin

# установка пароля для подключения к БД
spring.datasource.password=admin

# включение консоли H2 для возможности взаимодействия с БД через веб-интерфейс
spring.h2.console.enabled=true
```

Рисунок 35 – код в application.properties

После внесения всех настроек можно запустить приложение. Теперь база данных H2 работает в штатном режиме. Для подключения к БД нужно перейти по адресу /h2-console. Прежде, чем подключиться к БД будет сделана проверка получения списка всех результатов, которые сохраняются, пока приложение работает. Для проверки необходимо перейти по адресу <http://localhost:8080/calcList>. При этом никаких результатов на данный момент нет, так как еще не было выполнено никаких операций (Рисунок 36).

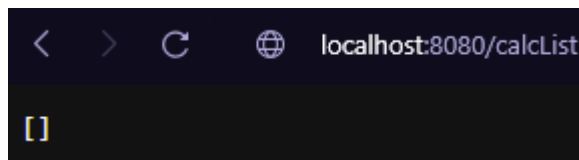


Рисунок 36 – пустой список результатов

Для проверки будет выполнено несколько операций сложения и вычитания. После данных операций необходимо обновить страницу для получения списка предыдущих результатов (Рисунок 37).

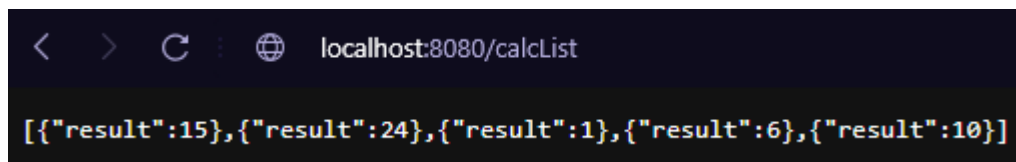


Рисунок 37 – заполненный список результатов

Проверка пройдена успешно. Теперь можно перейти к подключению БД H2 по адресу <http://localhost:8080/h2-console> (Рисунок 38).

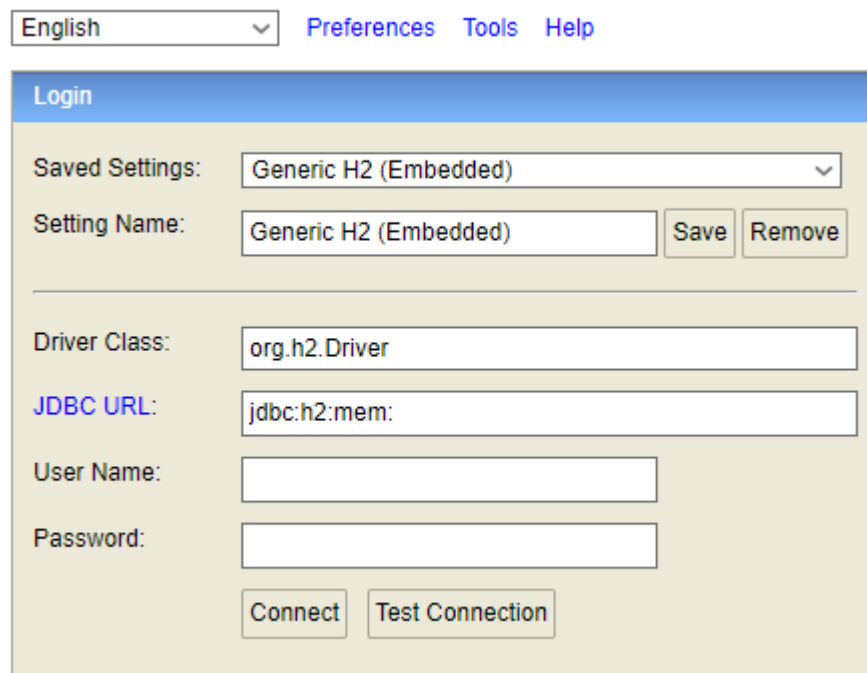


Рисунок 38 – главный интерфейс H2

На главном интерфейсе базы данных H2 представлены поля для ввода данных, а также настройки интерфейса позволяют выбрать нужный язык

интерфейса. Кроме того, можно найти справочную информацию про БД и другое. Для понимания текста будет выбран русский язык (Рисунок 39).

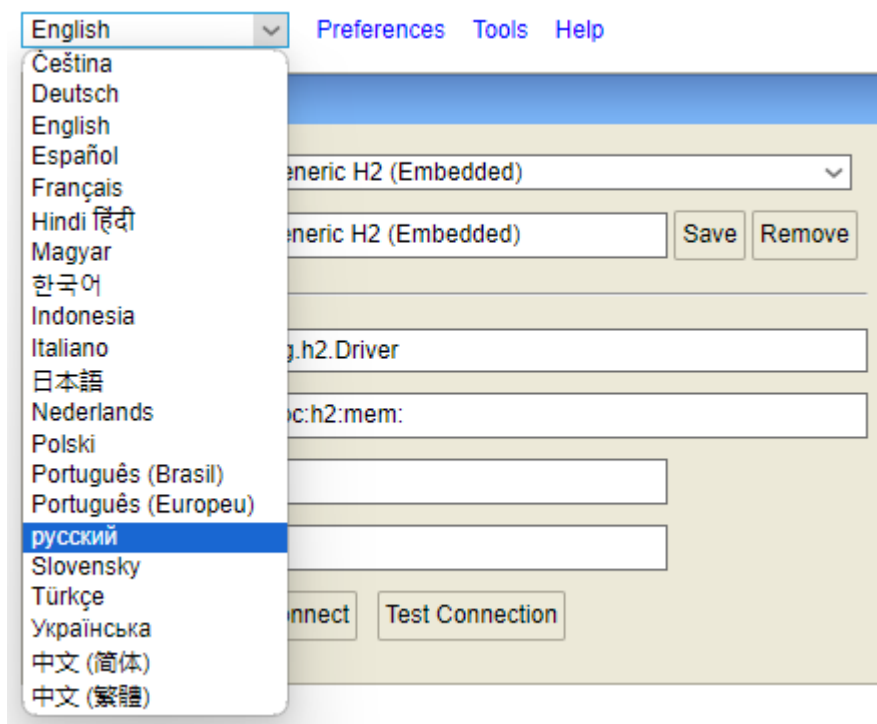


Рисунок 39 – выбор русского языка

Для подключения к БД необходимо ввести данные, которые были указаны в файле конфигурации (Рисунок 40).

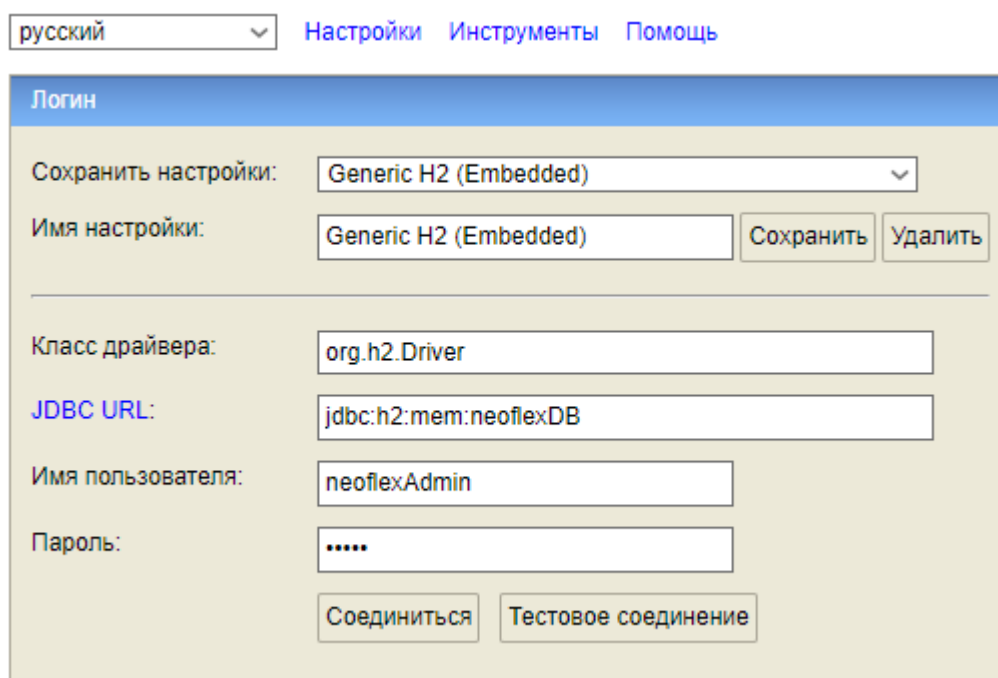


Рисунок 40 – заполнение данных

Перед соединением к БД интерфейс предлагает возможность сделать тестовое соединение. Если данные введены верно без ошибок, то тест будет успешно пройден (Рисунок 41).

The screenshot displays the 'Login' window of the H2 database interface. It features a blue title bar with the text 'Логин'. Below the title bar, there are several input fields and buttons. The 'Сохранить настройки:' field is a dropdown menu showing 'Generic H2 (Embedded)'. The 'Имя настройки:' field is a text box containing 'Generic H2 (Embedded)', with 'Сохранить' and 'Удалить' buttons to its right. Below these, there is a horizontal line. The 'Класс драйвера:' field is a text box containing 'org.h2.Driver'. The 'JDBC URL:' field is a text box containing 'jdbc:h2:mem:neoflexDB'. The 'Имя пользователя:' field is a text box containing 'neoflexAdmin'. The 'Пароль:' field is an empty text box. At the bottom, there are two buttons: 'Соединиться' and 'Тестовое соединение'. Below the main form area, there is a green banner with the text 'Тест прошел успешно'.

Рисунок 41 – успешное тестовое соединение

После успешного соединения появится основной интерфейс базы данных H2, который обладает разным полезным функционалом (Рисунок 42).



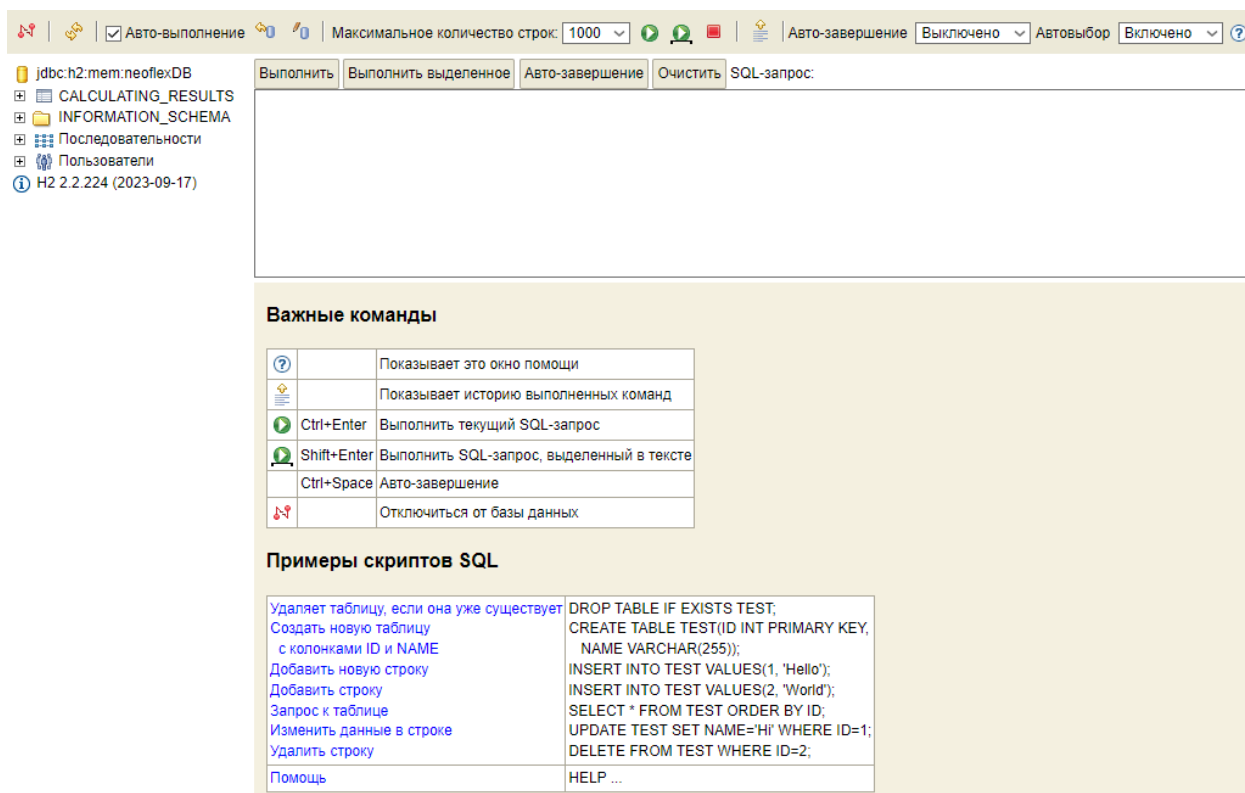


Рисунок 42 – основной интерфейс БД

Для вывода списка предыдущих результатов вычислений необходимо написать следующий SQL запрос и нажать на кнопку ‘Выполнить’ (Рисунок 43).

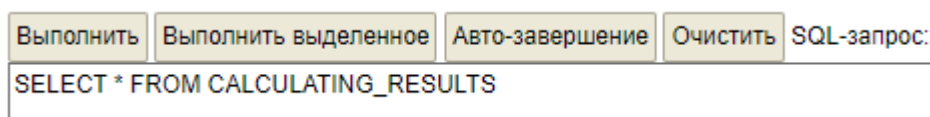


Рисунок 43 – код запроса

После выполнения запроса будет выведен список предыдущих результатов вычислений в виде таблицы с колонками RESULT и ID (Рисунок 44).

Выполнить

Выполнить выделенное

Авто-завершение

Очистить

SQL-запрос:

SELECT \* FROM CALCULATING\_RESULTS

SELECT \* FROM CALCULATING\_RESULTS;

RESULT	ID
15	1
24	2
1	3
6	4
10	5

(5 строки, 0 ms)

Править

Рисунок 44 – результат выполнения запроса

## ЗАКЛЮЧЕНИЕ

В ходе работы было успешно реализовано клиент-серверное приложение на языке программирования Java, которое принимает два числа от клиента и возвращает их сумму или разность, также был получен опыт работы с веб-интерфейсом Swagger для удобного взаимодействия с кодом. Были написаны тесты для надежности функционала приложения. Кроме того была выполнена интеграция внутренней базы данных H2. База данных дала возможность эффективно взаимодействовать с данными, обеспечивая легкость хранения и просмотр результатов вычислений.