ФИНАНСОВЫЙ УНИВЕРСИТЕТ
ПРИ ПРАВИТЕЛЬСТВЕ РОССИЙСКОЙ ФЕДЕРАЦИИ

Современные технологии программирования

Пальчевский Евгений Владимирович
Кандидат технических наук,
старший преподаватель кафедры информационных технологий

О курсе

Что ждёт в осеннем семестре 2024/25 учебного года?

1. **Лекции (8 штук)**
2. **Семинарские занятия (17 штук), исходя из нагрузки.**
3. **Выполнение индивидуальных заданий (решение задач) для семинарских занятий (в том числе и дома).**
4. **Контрольные работы.**
5. **За всё вы получаете баллы в соответствии с балльно-рейтинговой системой (БРС).**
6. **Экзамен.**

Темы лекционных и семинарских занятий

ФИНАНСОВЫЙ УНИВЕРСИТЕТ
ПРИ ПРАВИТЕЛЬСТВЕ РОССИЙСКОЙ ФЕДЕРАЦИИ

Основные темы лекций (то, что 100% будет на семинарских занятиях):

1. Работа с базами данных в Java. Основы объектно-реляционного отображения. Видеолекция №3, часть 3.
2. Spring Framework. Архитектура и основные сведения. Серия ОФФлайн лекций: часть 1, часть 2 и часть 3.
3. Генерация веб-страниц. Основы Thymeleaf. Серия ОФФлайн лекций: часть 1, часть 2 и часть 3.
4. Системы автоматической сборки пакетов. Реализация графического интерфейса в языке программирования Java. Видеолекция №5.
5. Потоковая организация системы ввода-вывода. Многопоточность.

Возможно выйдет на YouTube-канале/оффлайн (зависит от наличия свободного времени):

1. RESTful API-приложения.
2. Тестирование. Документирование в Java-проектах.
3. Maven. Архитектура и основные сведения.

Плейлист YouTube/плейлист ВК

Балльно-рейтинговая система (БРС)

ФИНАНСОВЫЙ УНИВЕРСИТЕТ
ПРИ ПРАВИТЕЛЬСТВЕ РОССИЙСКОЙ ФЕДЕРАЦИИ

ДО 01.04.2025

Вид поощрения	Баллы
Прохождение курса «Backend» от Я.Практикум на 100% (вплоть до проекта Фудграм не включительно) Записаться на курс	20

Ваши рейтинги с комментариями будут доступны в личном кабинете студента на org.fa.ru

Балльно-рейтинговая система (БРС)

ФИНАНСОВЫЙ УНИВЕРСИТЕТ
ПРИ ПРАВИТЕЛЬСТВЕ РОССИЙСКОЙ ФЕДЕРАЦИИ

С 01.04.2025

Вид поощрения	Баллы	Максимум	Возможно набрать согласно БРС
Посещение лекций и семинарских занятий	0,4 за каждое	**10**	**40**
Подготовка к семинарским и практическим занятиям (в том числе и выступления, онлайн-курсы)	0,765 за семинар (онлайн-курсы, решения задач)	**13**	
Контрольные работы	4,25 за каждую (4 контрольных)	**17**	
Зачет/экзамен	60	**60**	**60**
ИТОГО	**Зачтено/экзамен (удовлетворительно)** идет от 50 баллов и выше	**100**	**100**

Ваши рейтинги с комментариями будут доступны в личном кабинете студента на org.fa.ru

Контакты с преподавателем

ФИНАНСОВЫЙ УНИВЕРСИТЕТ
ПРИ ПРАВИТЕЛЬСТВЕ РОССИЙСКОЙ ФЕДЕРАЦИИ

Социальная сеть / мессенджер / почта	Контакт
Электронная почта	teelxp@inbox.ru, evpalchevskij@fa.ru
VK	https://vk.com/teelxp
WhatsApp	+7-937-485-80-48
YouTube YouTube-канал	https://www.youtube.com/@teelxp
Лекции в ВК (альтернатива ютубчику)	https://vk.com/finkablog

Требования к лекциям

1. Посещение лекций.
2. Ноутбук.

ФИНАНСОВЫЙ УНИВЕРСИТЕТ
ПРИ ПРАВИТЕЛЬСТВЕ РОССИЙСКОЙ ФЕДЕРАЦИИ

Материал на весенний семестр 2024/25 учебного года

ФИНАНСОВЫЙ УНИВЕРСИТЕТ
ПРИ ПРАВИТЕЛЬСТВЕ РОССИЙСКОЙ ФЕДЕРАЦИИ

Курсовые работы

План	Дедлайн	Группы
Определиться с темой и найти руководителя	20.02.2025 Подробнее можно уточнить на кафедре ИТ	ТРПО23-1 ТРПО23-2 ТРПО23-3 ТРПО23-4
Защита курсовых работ	Май-июнь 2025	

Список руководителей искать на кафедре информационных технологий

ФИНАНСОВЫЙ УНИВЕРСИТЕТ
ПРИ ПРАВИТЕЛЬСТВЕ РОССИЙСКОЙ ФЕДЕРАЦИИ

Какое ПО можно использовать?

Логотип	Название	Группы
	IntelliJ IDEA Ultimate	https://palchevsky.ru/materials.php
	Visual Studio Code	https://code.visualstudio.com/
	MySQL	https://dev.mysql.com/downloads/installer/
	PostgreSQL	https://www.postgresql.org/download/
	MongoDB	https://www.mongodb.com/try/download/community-edition

Курсы для дополнительного освоения материала

ФИНАНСОВЫЙ УНИВЕРСИТЕТ
ПРИ ПРАВИТЕЛЬСТВЕ РОССИЙСКОЙ ФЕДЕРАЦИИ

https://stepik.org/course/82867/promo - базовый, но очень широкий курс
https://stepik.org/course/53650/promo - JavaFX, Swing **(самая база, поэтому придется еще много чего самим прогуглить)**
https://stepik.org/course/146/promo - ORM и Hibernate, паттерны, MySQL
https://stepik.org/course/63054/promo - SQL
https://stepik.org/course/1240/promo - Введение в базы данных

Видеолекции к решению задач

ФИНАНСОВЫЙ УНИВЕРСИТЕТ
ПРИ ПРАВИТЕЛЬСТВЕ РОССИЙСКОЙ ФЕДЕРАЦИИ

Плейлист (все лекции обязательный к просмотру, НО ОСОБЕННО ВСЕ ЧАСТИ ЛЕКЦИИ №8!):
https://www.youtube.com/watch?v=w_j7PhNWkRY&list=PLNSAyqUuk6sSJn3EWtKKLLChUKQwr8Zou&pp=iAQB

ФИНАНСОВЫЙ УНИВЕРСИТЕТ
ПРИ ПРАВИТЕЛЬСТВЕ РОССИЙСКОЙ ФЕДЕРАЦИИ

Дедлайн по сдаче задач

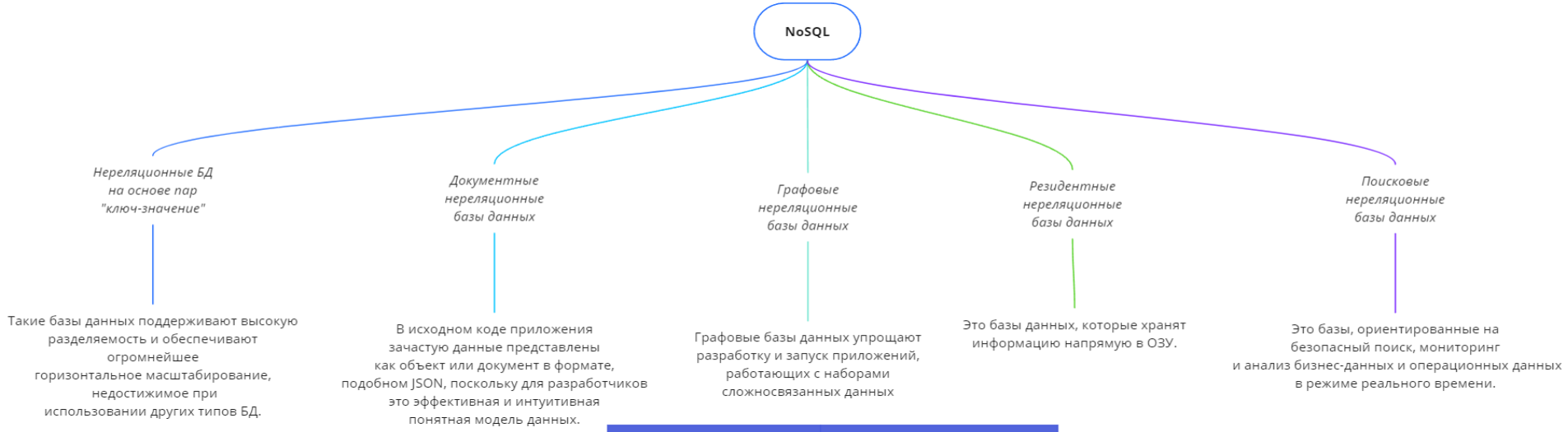
# задач	Дедлайн	Группы
1–4	30.09.2024	Все группы, у которых лектор будет вести семинарские занятия
5-9	31.10.2024	
10-12	29.11.2024	
13-17	27.12.2024	

ФИНАНСОВЫЙ УНИВЕРСИТЕТ
ПРИ ПРАВИТЕЛЬСТВЕ РОССИЙСКОЙ ФЕДЕРАЦИИ

Лекция №1. Работа с базами данных в Java. Основы объектно-реляционного отображения. Разница SQL и NoSQL?

Особенность (свойства, функциональность)	Реляционные базы данных (SQL)	Нереляционные базы данных (NoSQL)
Подходящие рабочие нагрузки	Предназначены для ***OLTP*** и ***OLAP***.	Такие базы данных работают по создаваемым шаблонам. Можем создавать собственные структуры данных.
Модель данных	Табличная	Ключ-значение, документы и графы.
Свойства ACID	SQL СУБД обеспечивают полный набор требований к транзакционной системе, обеспечивающий наиболее надёжную и предсказуемую её работу, иначе говоря ACID.	Базы данных NoSQL зачастую предлагают компромисс, смягчая жесткие требования свойств ACID ради более гибкой модели данных, которая допускает горизонтальное масштабирование.
Производительность	Зависит от дисковой подсистемы. Максимальная производительность требует определенной оптимизации структуры таблицы, запросов и индексов.	Зависит от: 1. Аппаратного обеспечения и пропускной способности сети сервера/кластера. 2. Задержки сети приложения, с которым работает NoSQL база данных.
Масштабирование	Масштабируются путем увеличения вычислительных возможностей аппаратного обеспечения или добавления отдельных копий для рабочих нагрузок чтения.	Базы данных NoSQL обычно поддерживают высокую разделяемость благодаря шаблонам доступа с возможностью масштабирования на основе распределенной архитектуры.
API	Запросы на запись и извлечение данных составляются на языке SQL. Эти запросы анализирует и выполняет реляционная база данных.	Объектно-ориентированные API позволяют разработчикам приложений без труда осуществлять запись и извлечение структур данных.

Основные отличия в понятиях SQL и NoSQL?



SQL	NoSQL
База данных	База данных
Таблица	Коллекция
Ряд	Документ
Столбец	Поле
Первичный ключ	ObjectID
Индекс	Индекс
Представление	Представление
Вложенная таблица	Встроенный документ
Массив	Массив

Синтаксис SQL (повтор). Основные понятия

База данных – это есть хранилище хранимой нами информации. Т.е. туда мы будем добавлять данные в табличном виде. Создается база данных командой CREATE DATABASE <название таблицы>.

Таблица. Таблицы являются объектами, которые содержат все данные в базах данных. В таблицах данные логически организованы в виде строк и столбцов по аналогии с электронной таблицей. Каждая строка представляет собой уникальную запись, а каждый столбец — поле записи.

Число таблиц в базе данных ограничено только числом объектов доступных в базе данных (2 147 483 647). Стандартная определяемая пользователем таблица может содержать до 1 024 столбцов. Число строк и общий размер таблицы ограничиваются только емкостью SSD-накопителей или жестких дисков на сервере.

Можно также устанавливать свойства для таблицы и каждого столбца в таблице для управления допустимыми данными и другими свойствами. Например, можно задать ограничения на столбец, чтобы в нем не допускались значения NULL, или указать значение по умолчанию, если оно не задано. Также можно присвоить ограничения ключа на таблицу, который обеспечивает уникальность, или установить связи между таблицами.

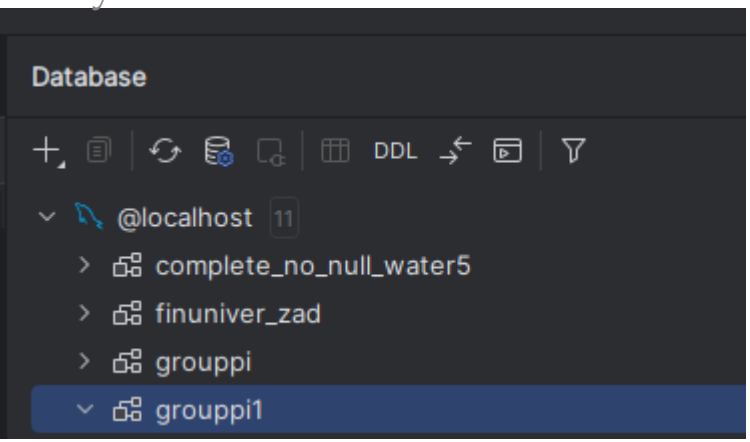
Данные в таблице могут быть сжаты либо по строкам, либо по страницам. Сжатие данных может позволить отображать больше строк на странице.

Синтаксис SQL (повтор). Практика

1. Создание базы данных

```
CREATE DATABASE
GroupPI1;
```

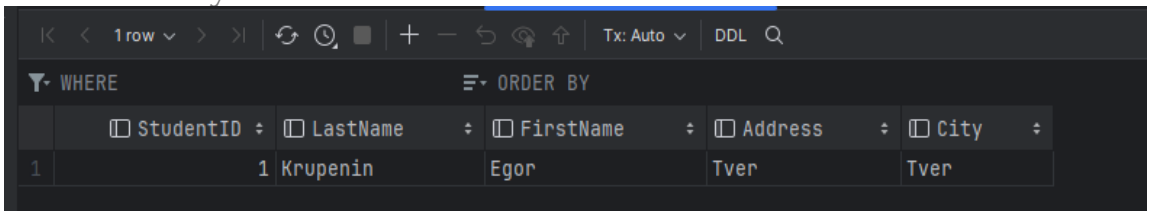
Результат:



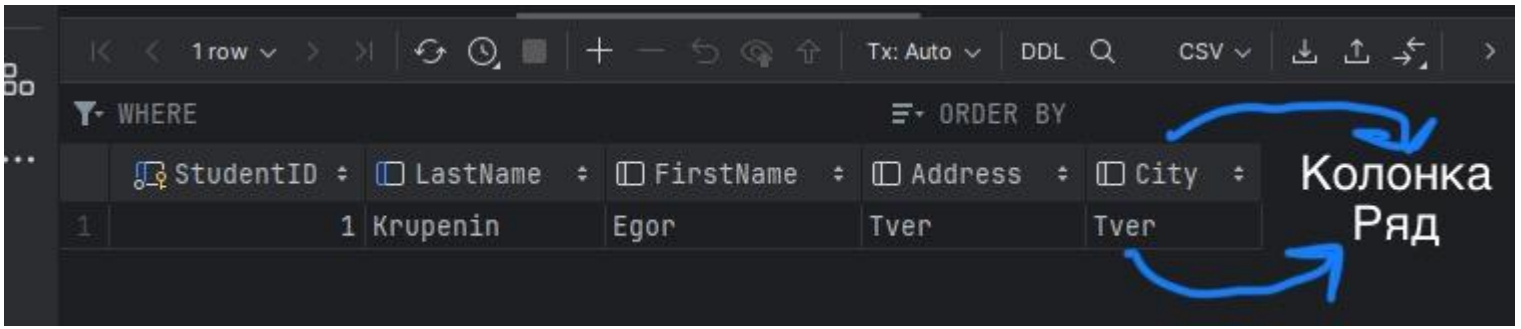
2. Создание таблицы

```
use GroupPI1;
CREATE TABLE Students (
    StudentID int,
    LastName varchar(255),
    FirstName varchar(255),
    Address varchar(255),
    City varchar(255)
);
INSERT INTO Students (StudentID, LastName, FirstName, Address, City)
VALUES ('1', 'Krupenin', 'Egor', 'Tver', 'Tver')
```

Результат:



3. Ряд и столбец



Синтаксис SQL (повтор). Основные понятия

Ряд – это строка, в которую заносим наши значения. Формируется только в том случае, если добавляем какие-либо данные. Строка может быть пустой в том случае, если там будет значение «NULL».

Столбец – это колонка, в которую также заносим значения. В каждой колонке есть ряды.

Первичный ключ – это поле, позволяющее идентифицировать каждую запись. Каждая таблица может содержать только один первичный ключ. За счет первичного ключа мы можем соединять сущности друг с другом, а также присваиваем каждому ряду уникальность и целостность данных.

ALTER TABLE – это команда, служащая для изменения макета таблицы после того, как она была создана с помощью директивы (или инструкции, команды) CREATE TABLE.

Синтаксис SQL (повтор). Основные понятия

Индексы в базе данных используются для ускорения операций поиска и сортировки. Они представляют собой специальные структуры данных, которые позволяют быстро находить записи в таблицах базы данных.

Когда вы выполняете запрос к базе данных, который включает в себя условие WHERE, сервер базы данных должен найти все строки, соответствующие этому условию. Если таблица большая, это может занять много времени. Индексы помогают сократить это время, так как они хранят информацию о расположении строк в таблице, которая соответствует определенным условиям.

Кроме того, индексы могут использоваться для оптимизации операций сортировки. Когда вы выполняете запрос, который требует сортировки данных, сервер базы данных может использовать индекс для определения порядка строк перед тем, как начать их физическое перемещение.

В целом, индексы являются важным инструментом для повышения производительности базы данных и ускорения выполнения запросов.

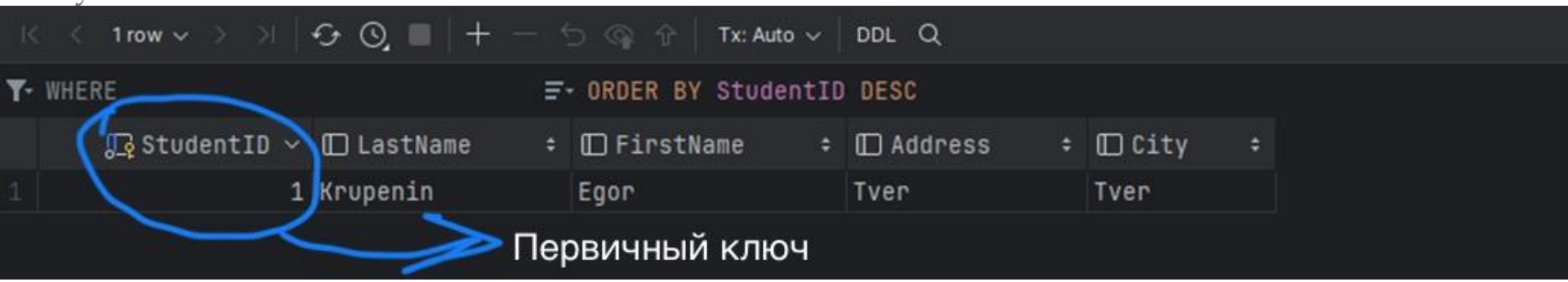
Синтаксис SQL (повтор). Практика

4. Первичный ключ

```
ALTER TABLE grouppi1.students
ADD PRIMARY KEY (StudentID)
```

За счет команды **ADD PRIMARY KEY** присваиваем столбцу **StudentID** уникальный ключ. Т.е. этот столбец теперь является уникальным ключом.

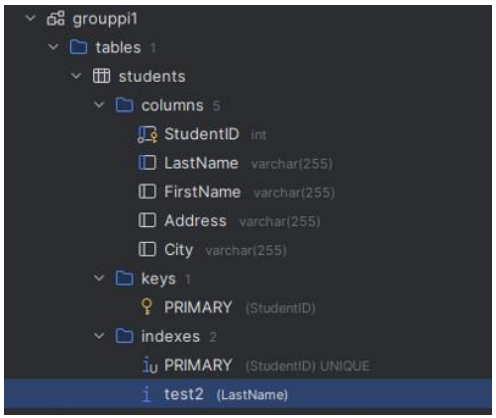
Результат:



5. Индексы

```
CREATE INDEX test2
ON grouppi1.students (LastName);
```

Результат:



Индексы – специальные таблицы, которые могут быть использованы поисковым двигателем базы данных (далее – БД), для ускорения получения данных.

Создаются такие таблицы с помощью команды **CREATE INDEX**.

Синтаксис SQL (повтор). Основные понятия

Представление. Представления в SQL являются особыми таблицами, которые содержат данные, полученные запросом SELECT из обычных таблиц. Это виртуальная таблица, к которой можно обратиться как к обычным таблицам и получить хранимые данные. Грубо говоря, мы извлекаем необходимые данные из нескольких таблиц и помещаем их в одну. Это намного проще, чем заново создавать таблицы и выполнять кучу других запросов. Более того, это можно сравнить с инкапсуляцией в ООП (объектно-ориентированном языке программирования): там мы можем создавать переменные с различными модификаторами доступа, а тут создаем таблицу с теми данными, которые хотим, чтобы видел пользователь в веб-интерфейсе реализованной системы.

Вложенные таблицы. Вложенную таблицу можно рассматривать как одномерный массив, в котором индексами служат значения целочисленного типа. Грубо говоря, небольшая таблица в виде коллекции внутри таблицы. Вложенная таблица может иметь пустые элементы, которые появляются после их удаления встроенной процедурой DELETE. Количество элементов может динамически увеличиваться. Максимальный размер – 2 ГБ. Очень удобно использовать при работе с большими однородными или темпоральными однородными данными. Однородные данные – это данные одного типа, а темпоральные – данные, привязанные к дате и времени.

ФИНАНСОВЫЙ УНИВЕРСИТЕТ
ПРИ ПРАВИТЕЛЬСТВЕ РОССИЙСКОЙ ФЕДЕРАЦИИ

6. Представление

6.1. Создаем еще одну таблицу в БД:

```
use GroupPI1;
CREATE TABLE Students1 (
    StudentID int,
    LastName varchar(255),
    FirstName varchar(255),
    Address varchar(255),
    City varchar(255)
);
INSERT INTO Students1 (StudentID, LastName, FirstName, Address, City)
VALUES ('1', 'Titov', 'Semen', 'Moscow', 'Moscow')
```

6.2. Присваиваем первичный ключ:

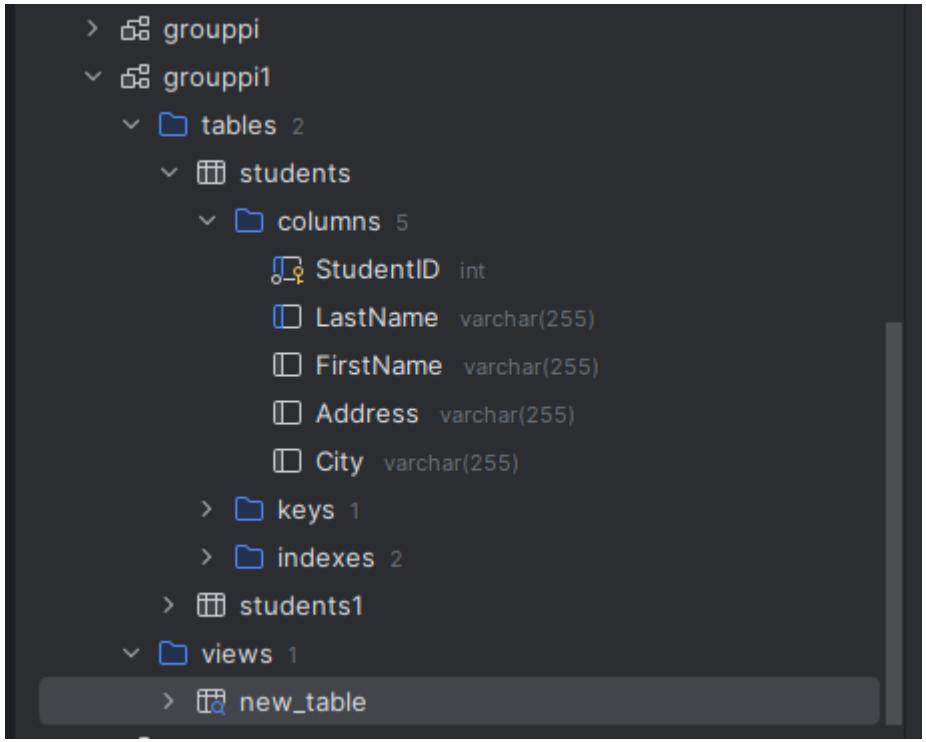
```
ALTER TABLE grouppi1.students
ADD PRIMARY KEY (StudentID)
```

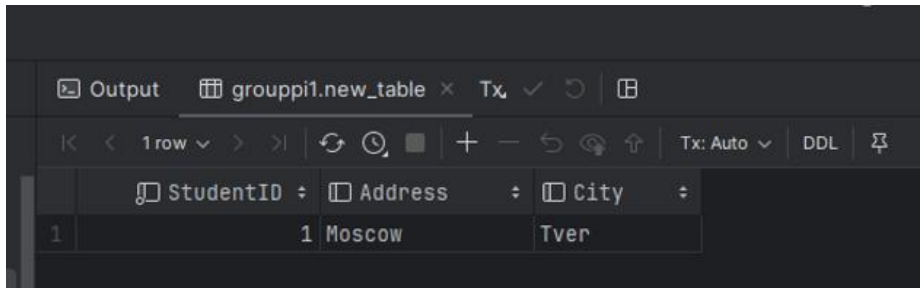
6.3. Создаем представление:

```
CREATE VIEW new_table
AS SELECT grouppi1.students.StudentID, grouppi1.students1.Address,
grouppi1.students.City
FROM students, students1
WHERE students.StudentID = students1.StudentID
```

```
SELECT * FROM new_table
```

6.4. Результат:





7. Вложенные таблицы (Nested Table)

Вложенную таблицу можно рассматривать как одномерный массив, в котором индексами служат значения целочисленного типа. Грубо говоря, небольшая таблица в виде коллекции внутри таблицы. Вложенная таблица может иметь пустые элементы, которые появляются после их удаления встроенной процедурой DELETE. Количество элементов может динамически увеличиваться. Максимальный размер – 2 ГБ. Очень удобно использовать при работе с большими однородными или темпоральными однородными данными. Однородные данные – это данные одного типа, а темпоральные – данные, привязанные к дате и времени.

8. Массивы

```
CREATE TABLE group2 (
    FirstName       text,
    LastName        text[],
    GroupHistory    text[][]
);
```

Задача

Разработка информационной системы «Кафе».

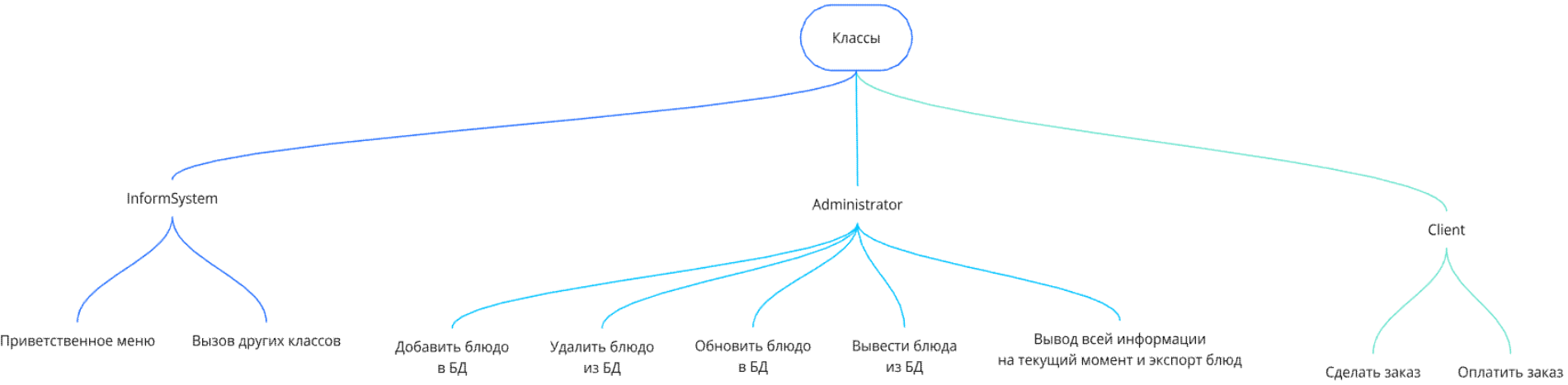


Рисунок 1 – Пример бизнес-логики программы

Решение задачи. Класс InformSystem

```
package test;

import java.sql.*;
import java.util.Scanner;

public class InformSystem {
   protected static Scanner scan = new Scanner(System.in);
   protected static String tablename = scan.nextLine();
   protected static String tablename1 = scan.nextLine();
   protected static String mysqlUrl =
"jdbc:mysql://localhost/test1000";
   protected static Connection con;

   static {
      try {
         con = DriverManager.getConnection(mysqlUrl, "root",
"root");
      } catch (SQLException e) {
         throw new RuntimeException(e);
      }
   }
```

Одна база для администратора, вторая – для клиента

```
public static void main(String[] args) throws SQLException {
      int x = 0;
      String s = "";
      while (!"3".equals(s)) {
         System.out.println("Добро пожаловать в наше кафе! Вы -
администратор или клиент?");
         System.out.println("1. Администратор.");
         System.out.println("2. Клиент.");
         System.out.println("3. Выход");
         s = scan.next();

         try {
            x = Integer.parseInt(s);
         } catch (NumberFormatException e) {
            System.out.println("Неверный формат ввода");
         }
         switch (x) {
            case 1 -> Administrator.main();
            case 2 -> Client.main();
         }
      }
      System.out.println("Пока! :)");
   }
}
```

Решение задачи. Класс Administrator

Начало

```
package test;

import java.sql.*;

public class Administrator extends InformSystem {
   protected static void main() throws SQLException {
      int x = 0;
      String s = "";
      String s1;
      Statement stmt = con.createStatement();
      String query = "CREATE TABLE IF NOT EXISTS " + tablename + " (ID int, Dish varchar(255), Tableintheres int, Quantity int, Price float,
Ingredients varchar(1000))";
      stmt.executeUpdate(query);
      while (!"8".equals(s)) {
         System.out.println("Привет, администратор! Выбери свое действие: ");
         System.out.println("1. Добавить блюдо в базу данных.");
         System.out.println("2. Обновить блюдо в базе данных.");
         System.out.println("3. Вывести блюда из базы данных.");
         System.out.println("4. Удалить блюдо из базы данных.");
         System.out.println("5. Добавить количество свободных столиков на текущий момент.");
         System.out.println("6. Вывести полную информацию на текущий момент.");
         System.out.println("7. Экспортировать все блюда и количество свободных столиков в локальное хранилище.");
         System.out.println("8. Выход в главное меню.");
         s = scan.next();
```

Решение задачи. Класс Administrator
Продолжение

```
try {
    x = Integer.parseInt(s);
} catch (NumberFormatException e) {
    System.out.println("Неверный формат ввода");
}
if (x == 1) {
    System.out.println("Добавляем блюдо в базу данных! Продолжить?");
    while (scan.hasNext()) {
        s1 = scan.next();
        if (s1.equals("Выход")) {
            break;
        }
        System.out.print("Введите ID блюда: ");
        int ID = scan.nextInt();
        scan.nextLine();
        System.out.print("Введите название блюда: ");
        String Dish = scan.nextLine();
        System.out.print("Введите количество блюд в наличии на сегодня: ");
        int Quantity = scan.nextInt();
        System.out.print("Введите цену на это блюдо: ");
        float Price = scan.nextFloat();
        scan.nextLine();
        System.out.print("Введите ингредиенты блюда: ");
        String Ingredients = scan.nextLine();
        String query1 = "INSERT INTO " + tablename + " (ID, Dish, Quantity, Price, Ingredients) VALUES (?, ?, ?, ?, ?)";
        PreparedStatement stmt1 = con.prepareStatement(query1);
        stmt1.setInt(1, ID);
        stmt1.setString(2, Dish);
        stmt1.setInt(3, Quantity);
        stmt1.setFloat(4, Price);
        stmt1.setString(5, Ingredients);
        stmt1.executeUpdate();
        System.out.println("Блюдо '" + Dish + "' успешно внесено в базу данных!");
        System.out.println("Продолжить ввод? Если нет, то введите 'Выход' для выхода в меню администратора.");
    }
}
```

Решение задачи. Класс Administrator

Продолжение

```
if (x == 2) {
    System.out.println("Изменяем блюдо в базе данных! Продолжить?");
    while (scan.hasNext()) {
        s1 = scan.next();
        if (s1.equals("Выход")) {
            break;
        }
        System.out.print("Введите ID изменяемого блюда: ");
        int ID = scan.nextInt();
        scan.nextLine();
        System.out.print("Введите новое название блюда: ");
        String Dish = scan.nextLine();
        System.out.print("Введите новое количество блюд на сегодня: ");
        int Quantity = scan.nextInt();
        System.out.print("Введите новую цену блюда: ");
        float Price = scan.nextFloat();
        System.out.print("Введите новые ингредиенты блюда: ");
        scan.nextLine();
        String Ingredients = scan.nextLine();
        String query2 = "UPDATE " + tablename + " SET Dish = '" + Dish + "', Quantity = '" + Quantity + "', Price = '" + Price + "', Ingredients = '" + Ingredients + "' WHERE ID = " +
ID + "";
        PreparedStatement stmt1 = con.prepareStatement(query2);
        stmt1.executeUpdate();
        System.out.println("Блюдо '" + Dish + "' добавлено. Хотите еще обновить блюдо? Если нет, то введите 'Выход' для выхода в меню администратора.");
    }
}
```

Решение задачи. Класс Administrator

Продолжение

```
if (x == 3) {
    Statement stmt1 = con.createStatement();
    ResultSet rs = stmt1.executeQuery("SELECT * FROM " + tablename + "");
    System.out.printf("%1s | %-20s | %-5s | %-6s | %-5s \n", "ID", "Название", "Количество", "Цена", "Ингредиенты");
    while (rs.next()) {

        int ID = rs.getInt("ID");
        String Dish = rs.getString("Dish");
        int Quantity = rs.getInt("Quantity");
        float Price = rs.getFloat("Price");
        String Ingredients = rs.getString("Ingredients");

        System.out.printf("%1d. | %-20s | %-10s | %.2f | %-5s \n", ID, Dish, Quantity, Price, Ingredients);
    }
}
if (x == 4) {
    System.out.println("Удаляем блюдо из базы данных! Продолжить?");
    while (scan.hasNext()) {
        s1 = scan.next();
        if (s1.equals("Выход")) {
            break;
        }
        System.out.print("Введите ID удаляемого блюда: ");
        int ID = scan.nextInt();
        String query2 = "SET SQL_SAFE_UPDATES = 0";
        String query3 = "DELETE FROM " + tablename + " WHERE ID = '" + ID + "'";
        String query4 = "SET SQL_SAFE_UPDATES = 1";
        PreparedStatement stmt1 = con.prepareStatement(query2);
        PreparedStatement stmt2 = con.prepareStatement(query3);
        PreparedStatement stmt3 = con.prepareStatement(query4);
        stmt1.executeUpdate();
        stmt2.executeUpdate();
        stmt3.executeUpdate();
        System.out.println("Блюдо с ID '" + ID + "' успешно удалено из базы данных!");
        System.out.println("Хотите еще удалить блюдо? Если нет, то введите 'Выход' для выхода в меню администратора.");
    }
}
```

Решение задачи. Класс Administrator

Продолжение

```
if (x == 5) {
    System.out.print("Добавьте количество свободных столиков на текущий момент: ");
    int Tableintheres = scan.nextInt();
    String query1 = "SET SQL_SAFE_UPDATES = 0";
    String query2 = "UPDATE " + tablename + " SET Tableintheres = '" + Tableintheres + "' WHERE Tableintheres IS NULL";
    String query3 = "SET SQL_SAFE_UPDATES = 1";
    PreparedStatement stmt1 = con.prepareStatement(query1);
    PreparedStatement stmt2 = con.prepareStatement(query2);
    PreparedStatement stmt3 = con.prepareStatement(query3);
    stmt1.executeUpdate();
    stmt2.executeUpdate();
    stmt3.executeUpdate();
    System.out.println("Столики успешно добавлены!");
}
if (x == 6) {
    Statement stmt1 = con.createStatement();
    ResultSet rs = stmt1.executeQuery("SELECT * FROM " + tablename + "");
    System.out.printf("%1s | %-20s | %-5s | %-6s | %-20s | %-5s \n", "ID", "Название", "Количество", "Цена", "Количество свободных столиков", "Ингредиенты");
    while (rs.next()) {

        int ID = rs.getInt("ID");
        String Dish = rs.getString("Dish");
        int Quantity = rs.getInt("Quantity");
        float Price = rs.getFloat("Price");
        int Tableintheres = rs.getInt("Tableintheres");
        String Ingredients = rs.getString("Ingredients");

        System.out.printf("%1d. | %-20s | %-10s | %.2f | %-29s | %-5s \n", ID, Dish, Quantity, Price, Tableintheres, Ingredients); // % - спецификатор формата, после которого указываются аргументы, тире и число - количество пробелов, d (десятичное целое), f (флоат), s (стринг) - типы данных.
    }
}
```

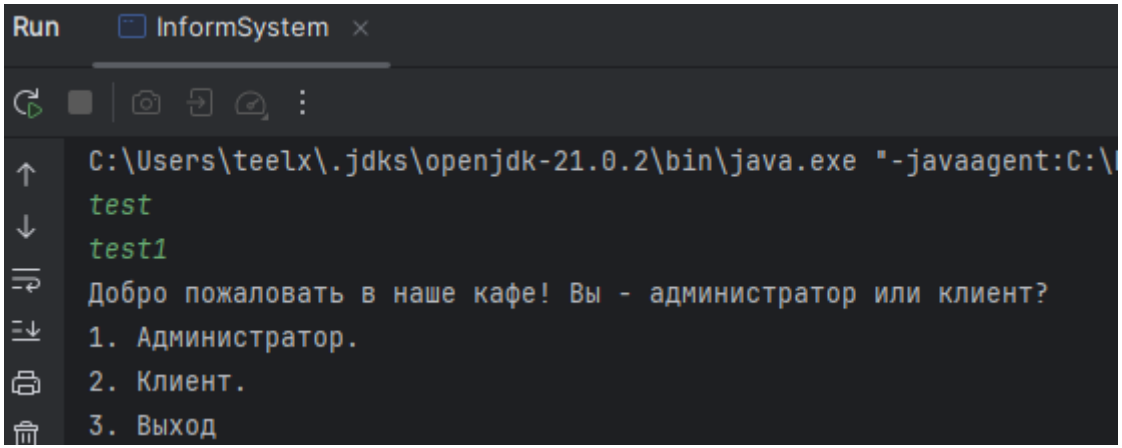
Решение задачи. Класс Administrator
Окончание

```
        if (x == 7) {
            scan.nextLine();
            System.out.print("Введите название файла с расширением: ");
            String file1 = scan.nextLine();
            String query1 = "SET SQL_SAFE_UPDATES = 0";
            String query2 = "UPDATE " + tablename + " SET Price = ROUND(Price, 2)";
            String query3 = "SET SQL_SAFE_UPDATES = 1";
            String query4 = "SELECT 'ID', 'Dish', 'Tableintheres', 'Quantity', 'Price', 'Ingredients' UNION ALL SELECT * FROM " + tablename + " INTO OUTFILE
'C:/Users/teelx/Desktop/mysql/" + file1 + "' CHARACTER SET cp1251";
            PreparedStatement stmt1 = con.prepareStatement(query1);
            PreparedStatement stmt2 = con.prepareStatement(query2);
            PreparedStatement stmt3 = con.prepareStatement(query3);
            PreparedStatement stmt4 = con.prepareStatement(query4);
            stmt1.executeUpdate();
            stmt2.executeUpdate();
            stmt3.executeUpdate();
            stmt4.executeQuery(); // Когда SELECT, тогда всегда пишем executeQuery, а не executeUpdate.
            System.out.println("Данные успешно экспортированы!");
        }
    }
  }
}
```

Решение задачи. Класс Client

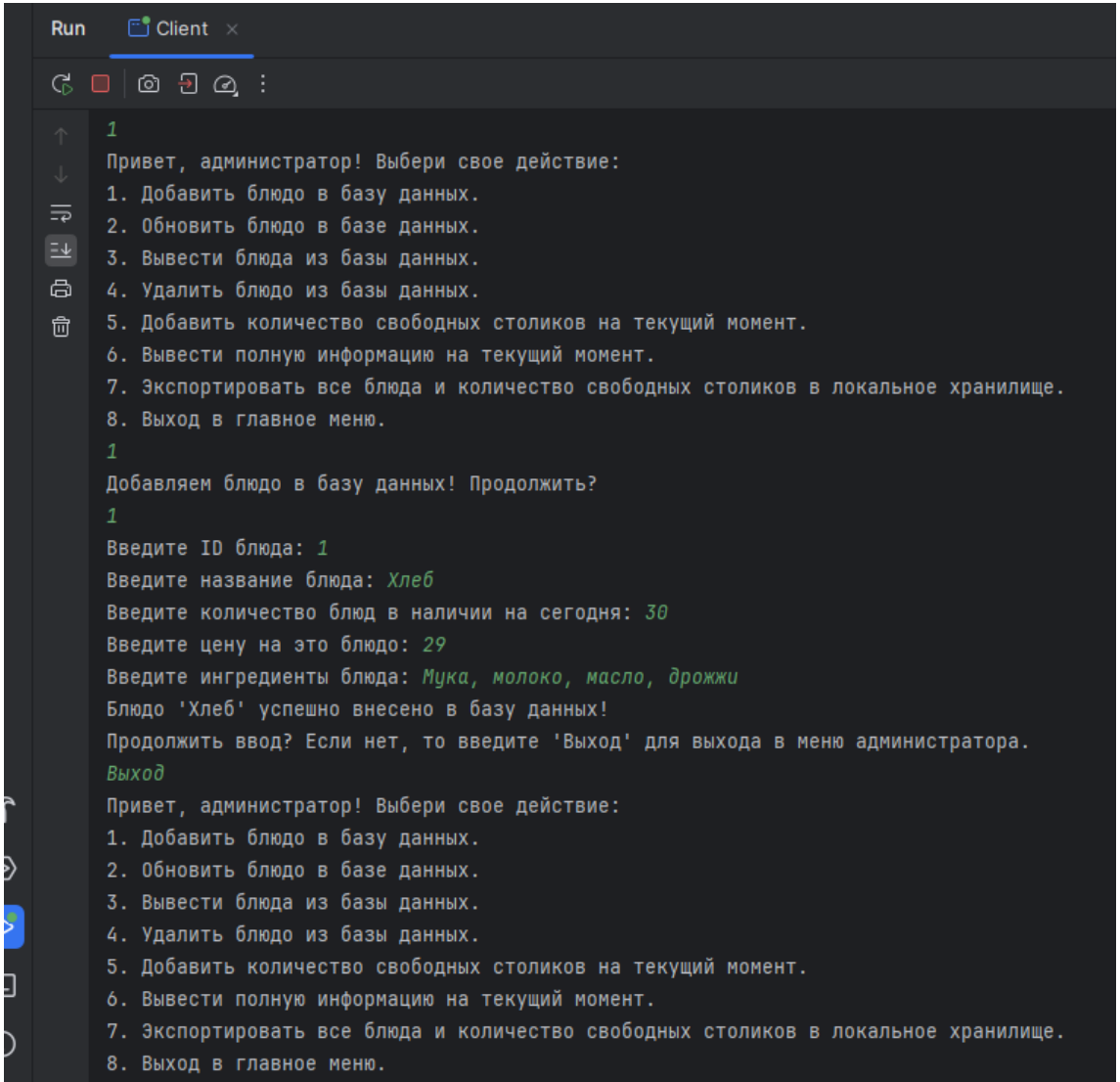
```
package test;
import java.sql.*;
import java.util.Random;

public class Client extends InformSystem {
   public static void main() throws SQLException {
      Random random = new Random();
      int x = 0;
      String s = "";
      String s1;
      Statement stmt = con.createStatement();
      String query = "CREATE TABLE IF NOT EXISTS " + tablename1 + " (ID int, Dish varchar(255),
Tableintheres int)";
      stmt.executeUpdate(query);
      while (!"3".equals(s)) {
         System.out.println("Привет, клиент! Выбери свое действие: ");
         System.out.println("1. Сделать заказ.");
         System.out.println("2. Оплатить.");
         System.out.println("3. Выйти в главное меню.");
         s = scan.next();

         try {
            x = Integer.parseInt(s);
         } catch (NumberFormatException e) {
            System.out.println("Неверный формат ввода");
         }
         if (x == 1) {
            System.out.println("Вы хотите сделать заказ! Продолжить?");
            while (scan.hasNext()) {
               s1 = scan.next();
               if (s1.equals("Выход")) {
                  break;
               }
               System.out.println("Доступные на текущий момент блюда:");
               Statement stmt1 = con.createStatement();
               ResultSet rs = stmt1.executeQuery("SELECT * FROM " + tablename
+ "");
               System.out.printf("%1s | %-20s | %-5s | %-6s | %-5s \n", "ID",
"Название", "Количество", "Цена", "Ингредиенты");
               while (rs.next()) {

                  int ID = rs.getInt("ID");
                  String Dish = rs.getString("Dish");
                  int Quantity = rs.getInt("Quantity");
                  float Price = rs.getFloat("Price");
                  String Ingredients = rs.getString("Ingredients");

                  System.out.printf("%1d. | %-20s | %-10s | %.2f | %-5s \n", ID,
Dish, Quantity, Price, Ingredients);
               }
               scan.nextLine();
               System.out.print("Выберите название блюда: ");
               String Dish = scan.nextLine();
               System.out.println("Вы выбрали блюдо " + Dish);
               int a = random.nextInt();
               System.out.println("Ваш заказ № " + a + ". Оплатите его и
присаживайтесь за любой свободный столик.");
               System.out.println("Введите 'Выход' для выхода в меню с целью
оплаты заказа.");
            }
         }
         if (x == 2) {
            System.out.println("Спасибо за оплату заказа!");
         }
      }
   }
}
```

Решение задачи. Результат. Вывод результатов выполнения функций (методов) класса «InformSystem»



Нажимаем «1»

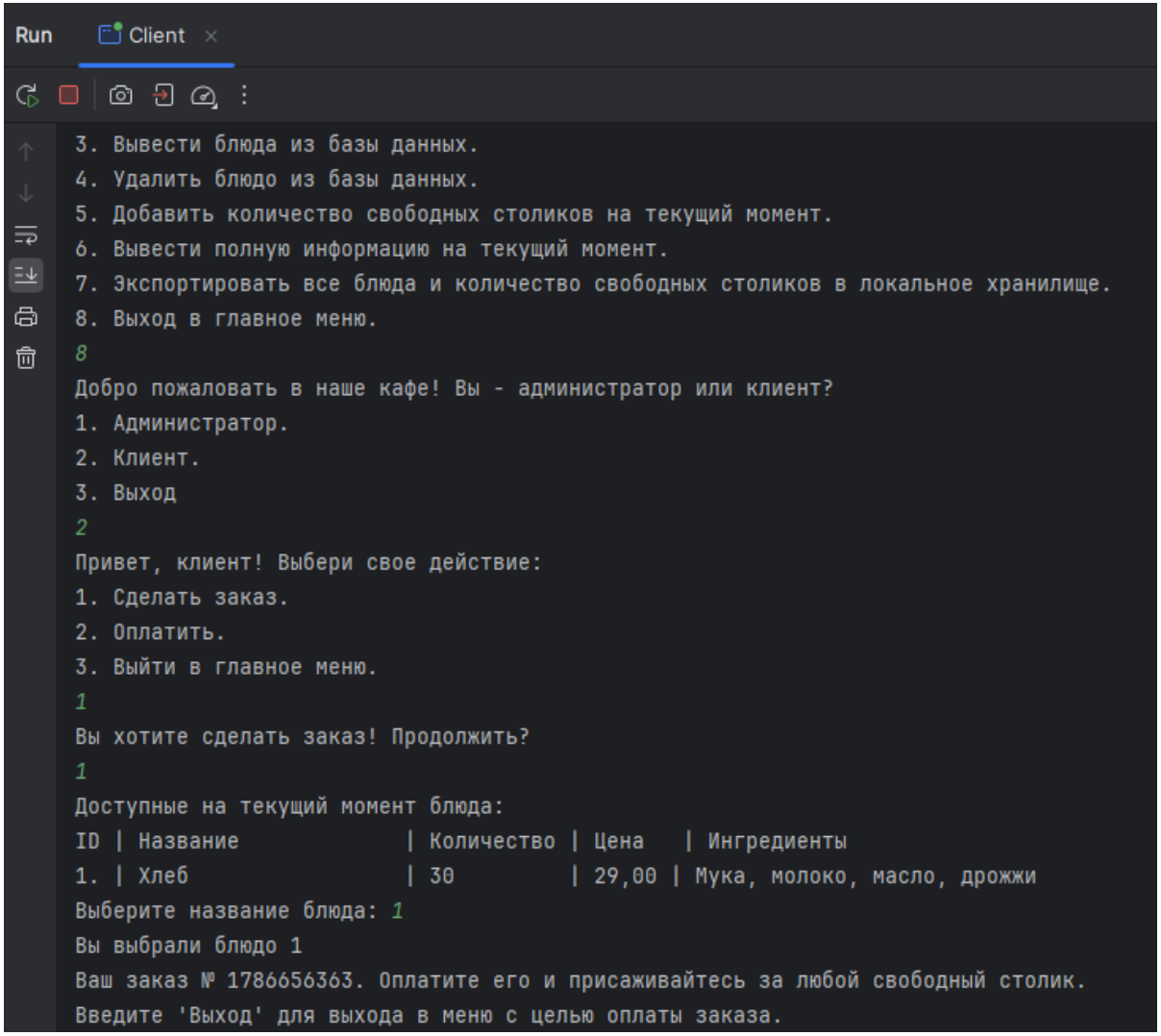
Решение задачи. Результат. Вывод результатов выполнения функций (методов) класса «Administrator»



Решение задачи. Результат. Вывод результатов выполнения функций (методов) класса «Administrator»

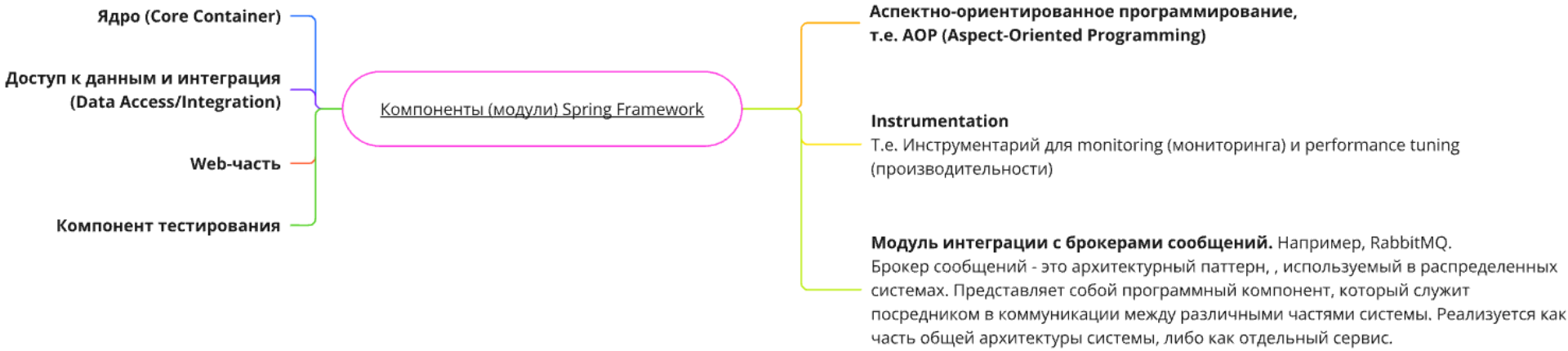
```
Привет, администратор! Выбери свое действие:
1. Добавить блюдо в базу данных.
2. Обновить блюдо в базе данных.
3. Вывести блюда из базы данных.
4. Удалить блюдо из базы данных.
5. Добавить количество свободных столиков на текущий момент.
6. Вывести полную информацию на текущий момент.
7. Экспортировать все блюда и количество свободных столиков в локальное хранилище.
8. Выход в главное меню.
6
ID | Название            | Количество | Цена    | Количество свободных столиков | Ингредиенты
1. | Хлеб                | 30         | 29,00 | 0                                | Мука, молоко, масло, дрожжи
Привет, администратор! Выбери свое действие:
1. Добавить блюдо в базу данных.
2. Обновить блюдо в базе данных.
3. Вывести блюда из базы данных.
4. Удалить блюдо из базы данных.
5. Добавить количество свободных столиков на текущий момент.
6. Вывести полную информацию на текущий момент.
7. Экспортировать все блюда и количество свободных столиков в локальное хранилище.
8. Выход в главное меню.
```

Решение задачи. Результат. Вывод результатов выполнения функций (методов) класса «Client»

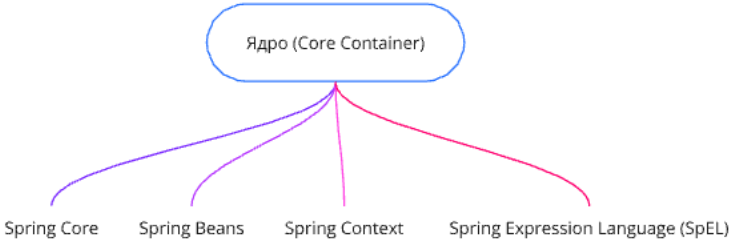


Spring Framework. Архитектура и основные сведения

Spring Framework — один из самых распространенных фреймворков для разработки корпоративных приложений на языке Java. Данный фреймворк включает в себя обширный набор инструментов, упрощающих разработку и тестирование приложений (систем).



Spring Framework. Ядро



Spring Core. Основной модуль, содержащий базовые функции фреймворка, включая Dependency Injection (DI) и Inversion of Control (IoC).

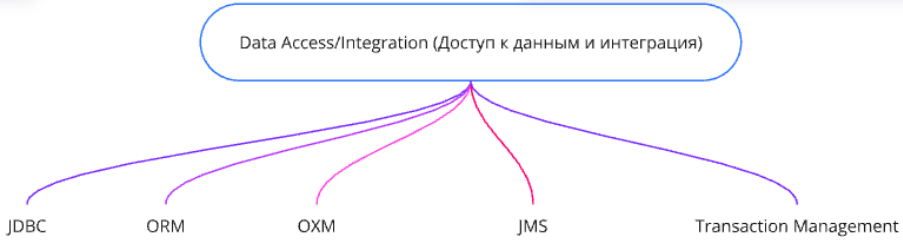
Spring Beans. Модуль, реализующий DI и управляемый жизненный цикл бинов.

Spring Context. Модуль-надстройка над Core и Beans, обеспечивающая доступ к IoC контейнеру. Включает в себя расширенные функции, такие как Event Propagation, Internationalization (i18n) и т.д.

Spring Expression Language. Мощный язык выражений для работы с объектами в контексте Spring.

IoC-контейнер — библиотека или фреймворк, позволяющие упростить и автоматизировать написание кода.

Spring Framework. Data Access/Integration



JDBC (Java DataBase Connectivity — соединение с базами данных на Java). Упрощает работу с JDBC API, позволяет избежать шаблонного кода, позволяет интегрировать проект с базами данных.

ORM (Object-Relational Mapping). Поддержка различных ORM-фреймворков, таких как Hibernate, JPA, MyBatis. Простыми словами, позволяет разрабатывать структуру базы данных и выполнять запросы посредством языка программирования, т.е. без использования нативных SQL-запросов

OXM (Object XML Mapping). Поддержка для JAXB, Castor и других инструментов маппинга объектов на XML. Простыми словами, поддержка интеграции с XML.

JMS (Java Messaging Service). Поддержка для асинхронного обмена сообщениями.

Transaction Management. Управление транзакциями, поддержка как программной, так и декларативной модели.

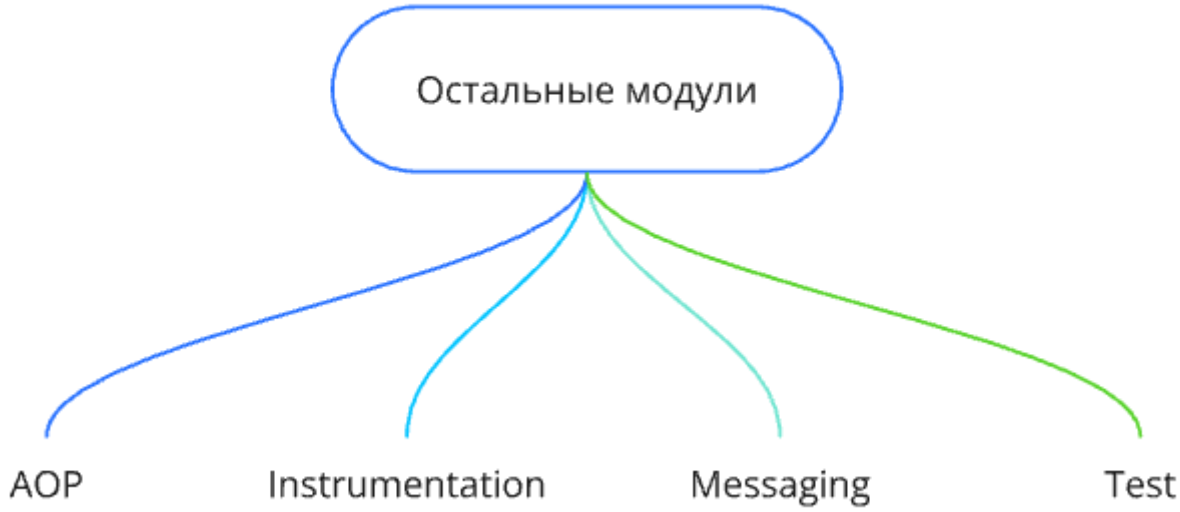
Spring Framework. WEB-часть

При помощи данного компонента мы можем реализовывать веб-интерфейс.

1. **Spring MVC.** Компонент (модуль) для разработки веб-приложений на основе паттерна Model-View-Controller. Поддерживает RESTful веб-сервисы и легко интегрируется с различными видами представлений, включая JSP, Thymeleaf и др.
2. **Spring WebFlux.** Асинхронный, неблокирующий веб-фреймворк, использующий реактивное программирование и поддержку сервлетов 3.1+.

Реактивное программирование — это асинхронность, соединенная с потоковой обработкой данных. То есть если в асинхронной обработке нет блокировок потоков, но данные обрабатываются все равно порциями, то реактивность добавляет возможность обрабатывать данные потоком. Грубо говоря, мы можем обрабатывать равное количество данных потоком.

Spring Framework. Остальные компоненты (модули)



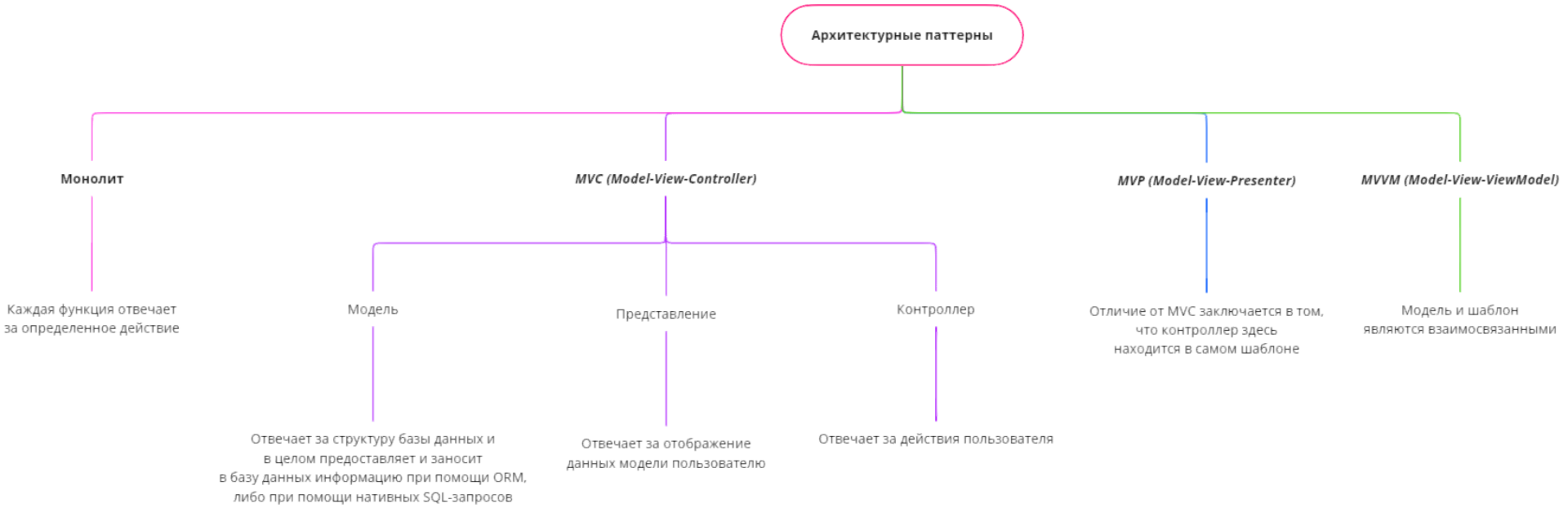
AOP (Aspect-Oriented Programming). Модуль, позволяющий реализовать аспектно-ориентированное программирование и отделяющий кросс-срезные проблемы, такие как логирование, безопасность, управление транзакциями и т.д.
Instrumentation. Инструментарий для поддержки инструментов для мониторинга и повышения производительности.
Messaging. Модуль, поддерживающий интеграции с брокерами сообщений.
Test. Модуль, предоставляющий инструментарий для тестирования.

Spring Framework. Основные концепции

1. **Dependency Injection (DI) и Inversion of Control (IoC).** Это два принципа: внедрение зависимостей и инверсия управления. Основные принципы, которые позволяют создавать гибкие и легко расширяемые приложения.
Например, DI позволяет разделять ответственность между компонентами, что облегчает тестирование и поддержку кода. Т.е. каждый модуль или класс отвечает за свои функции. Этот принцип мы будем активно использовать в течение семестра.
2. **Aspect-Oriented Programming (AOP, Аспектно-ориентированное программирование).** Это парадигма программирования, направленная на отделение кросс-срезных задач (concerns), таких как логирование, управление транзакциями, безопасность, мониторинг и обработка исключений, от основной бизнес-логики приложения.
В контексте Spring Framework, AOP помогает уменьшить дублирование кода и повысить его читаемость и понимаемость. Пример: в Spring есть классы конфигурации или конфигурационные файлы, которые мы создаем сами, либо созданы по умолчанию (application.properties).
3. **Model-View-Controller (MVC).** Архитектурный паттерн, разделяющий приложение на три основные компонента: модель, представление и контроллер, что делает код более структурированным и понятным.
4. **Transaction Management (Управление транзакциями)**. Управление транзакциями, которое можно настраивать программно или декларативно с помощью аннотаций.
5. **Reactive Programming**. Поддержка реактивного программирования через WebFlux для создания асинхронных и масштабируемых приложений.

ФИНАНСОВЫЙ УНИВЕРСИТЕТ
ПРИ ПРАВИТЕЛЬСТВЕ РОССИЙСКОЙ ФЕДЕРАЦИИ

Spring Framework. Какие архитектурные паттерны существуют и какие можно применять в Spring?



Spring Framework. Какие архитектурные паттерны существуют и какие можно применять в Spring?

1. ~~***Монолит***~~**.** По сути, привычная нам модель. Тут каждая функция отвечает за определенное действие. Зачастую такие приложения реализовываются в виде одного модуля, разделяемого на отдельные файлы (подмодули). Каждый подмодуль – отдельная функция.
2. ***MVC (Model-View-Controller)*** или по-русски (Модель-Представление-Контроллер) – архитектура, представляющая собой схему разделения приложения на три отдельных компонента: модель, представление и контроллер. За что отвечает каждый компонент? **Модель** отвечает за структуру базы данных и в целом предоставляет и заносит в базу данных информацию при помощи ORM, либо при помощи нативных SQL-запросов. Нативный SQL-запрос – это, по сути, обычный SQL-запрос. Проще говоря, мы создаем файл с моделью, в котором определяем саму сущность и структуру таблицы базы данных, а также, при необходимости, создаем репозиторий с нативными SQL-запросами. **Представление** отвечает за отображение данных модели пользователю. Если совсем просто говорить, то это наши HTML-шаблоны. **Контроллер** отвечает за действия пользователя. Например, при переходе на какой-то URL у нас будет вызываться определенная модель в определенной функции.
3. ***MVP (Model-View-Presenter).*** Отличие от MVC заключается в том, что контроллер здесь находится в самом шаблоне.
4. ***MVVM (Model-View-View-Model)*** тоже шаблон (т.е. паттерн) проектирования архитектуры приложения, основанный на MVC. Суть в том, что тут модель и шаблон являются взаимосвязанными. Т.е. если в MVC и MVP шаблон на модель не влияет, то в MVVM влияет напрямую.

Spring Framework. Выводы

1. Модульная архитектура. Можно использовать только те компоненты, которые необходимы.
2. Гибкость. Поддержка различных подходов к разработке, включая реактивное и RESTful программирование.
3. Универсальность. Поддержка множества технологий, стандартов и фреймворков, таких как JPA, Hibernate, JMS и других.
4. Большая экосистема. Легко интегрируется с другими библиотеками и фреймворками, такими как Spring Boot, Spring Cloud, которые расширяют возможности Spring Framework.
5. Spring Framework широко используется для разработки веб-приложений, микросервисов, а также крупных корпоративных систем благодаря своей гибкости, расширяемости и поддержке передовых технологий и стандартов.

Пример приложения

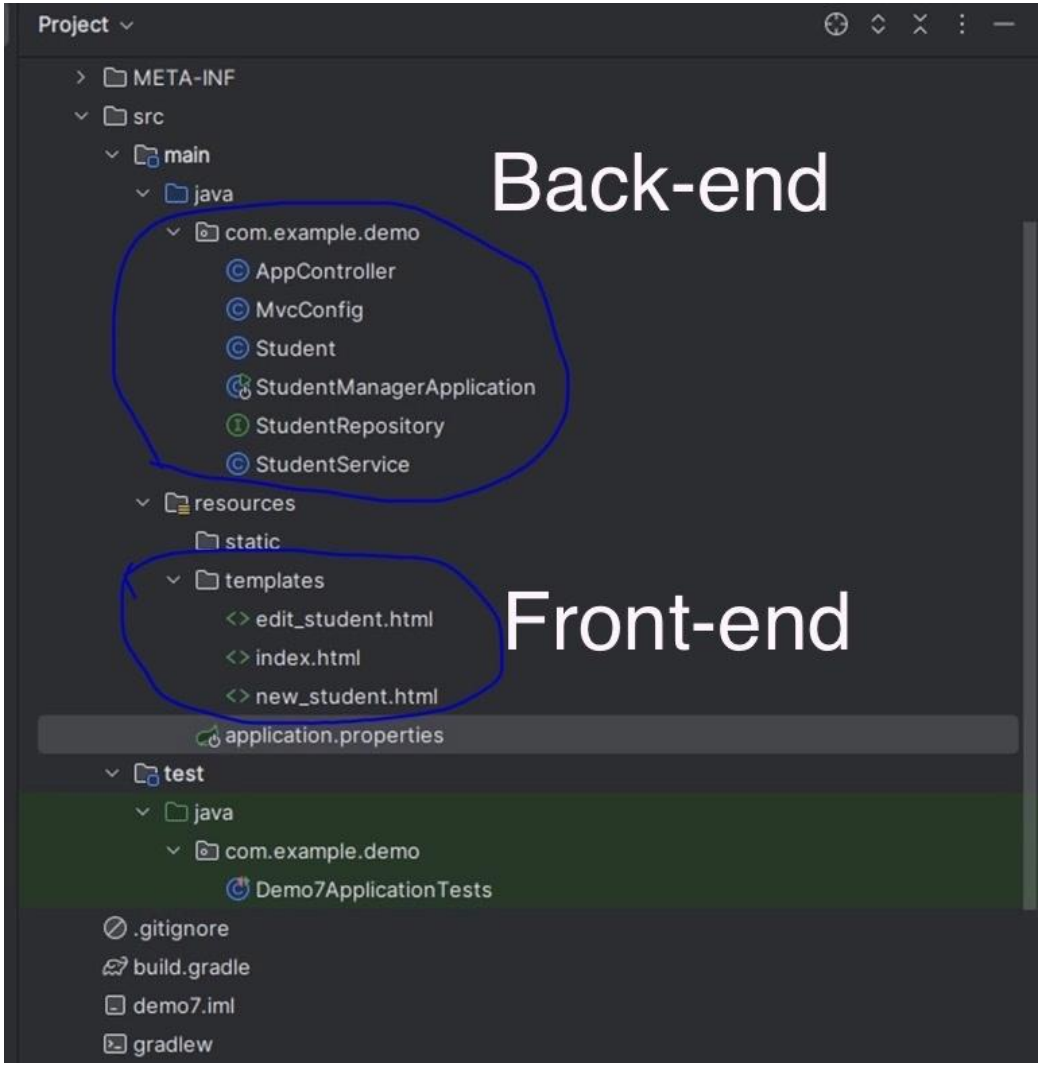
Небольшое техническое задание.

Разработать информационную систему для работы со студентами. Необходимые функции:

1. Добавить студента
2. Удалить студента
3. Редактировать студента
4. Поиск по любому критерию

ФИНАНСОВЫЙ УНИВЕРСИТЕТ
ПРИ ПРАВИТЕЛЬСТВЕ РОССИЙСКОЙ ФЕДЕРАЦИИ

Информационная система работы со студентами. Структура



ФИНАНСОВЫЙ УНИВЕРСИТЕТ
ПРИ ПРАВИТЕЛЬСТВЕ РОССИЙСКОЙ ФЕДЕРАЦИИ

Информационная система работы со студентами. Application.Properties

```
#Подключение к MySQL
spring.datasource.url=jdbc:mysql://localhost:3306/student
spring.datasource.username=root
spring.datasource.password=root
#Движок базы данных
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect
#Автоматическое создание SQL-таблиц
spring.jpa.hibernate.ddl-auto=create
#Включаем логирование (Дебаг-режим)
logging.level.org.hibernate.SQL=DEBUG
#Эта строка устанавливает уровень логирования для модуля
org.hibernate.type в режим TRACE.
#Это означает, что Hibernate будет записывать в журнал все события,
связанные с типом данных, включая подробную информацию о том,
какие операции выполняются над данными и какие значения
передаются.
#Это может быть полезно для отладки и анализа работы Hibernate с
данными.
logging.level.org.hibernate.type=TRACE
#Учетные данные от Spring-Security
spring.security.user.name=root
spring.security.user.password=root
spring.security.user.roles=manager
#Кэширование
spring.cache.type=none
#Включение автоматического "подтягивания" статического контента,
таких как CSS, JavaScript и изображений из каталога resources и static.
spring.web.resources.add-mappings=true
```

JPA (Java Persistence API) - это спецификация API, которая предоставляет возможность сохранять Java-объекты в базе данных. Она поддерживает различные аспекты сохранения данных, включая API, язык запросов и метаинформацию.

Hibernate - это библиотека для языка программирования Java, которая является одной из самых популярных реализаций JPA. Она позволяет сократить объемы низкоуровневого программирования при работе с реляционными базами данных и предоставляет средства для автоматической генерации и обновления таблиц, построения запросов и обработки данных. Hibernate также предоставляет поддержку для использования SQL-подобного языка запросов и аннотаций для проверки верности данных.

Информационная система работы со студентами. StudentManagerApplication

Данный класс отвечает за сборку и запуск нашего веб-приложения. Класс **StudentManagerApplication** является точкой входа в приложение Spring Boot. Класс наследуется от **SpringBootServletInitializer**, что позволяет ему работать как сервлетный контейнер.

Сервлетный контейнер – это наш веб-сервер (по умолчанию Apache Tomcat).

Класс помечен аннотацией @**SpringBootApplication**, которая указывает Spring Boot, что это основной класс приложения. Это включает в себя автоматическую конфигурацию Spring, такие как автоматическое обнаружение компонентов приложения и их сборку.

Метод **main** запускает приложение, вызывая метод **run** класса **SpringApplication** с классом **StudentManagerApplication** и аргументами командной строки.

```
package com.example.demo;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.web.servlet.support.SpringBootServletInitializer;

@SpringBootApplication
public class StudentManagerApplication extends SpringBootServletInitializer {

    public static void main(String[] args) {
        SpringApplication.run(StudentManagerApplication.class, args);
    }

}
```

Информационная система работы со студентами. Класс Student. Начало

```
package com.example.demo;

import jakarta.persistence.Entity; // Интерфейс для определения класса сущностей, который будет храниться в БД
import jakarta.persistence.GeneratedValue; // Интерфейс для автоматической генерации значений ID
import jakarta.persistence.GenerationType; // Метод для представления стратегии автоматической генерации значений ID
import jakarta.persistence.Id; // Интерфейс для аннотации @ID, что позволяет обозначить в базе данных поле ID в качестве первичного ключа
import lombok.Getter; //Интерфейс, позволяющий указывать на то, переменные можно обозначить геттерами
import lombok.Setter; //Интерфейс, позволяющий указывать на то, переменные можно обозначить сеттерами


@Setter // Указываем, что все наши переменные являются сеттерами
@Entity
// Это означает, что класс Student представляет сущность в базе данных, и Hibernate будет использовать этот класс для управления данными в базе данных.
public class Student { // Название нашей таблицы в БД
    private Long id; // ID
    @Getter // Указываем, что геттер
    private String first; // Имя
    @Getter
    private String last; // Фамилия
    @Getter
    private String num; // номер студенческого
    @Getter
    private float av; // средний балл

    protected Student() { // Создаем метод
    }
```

Информационная система работы со студентами. Класс Student. Окончание

```
//Аннотация @Id указывает, что свойство id является первичным ключом (primary key) в базе данных.
// Это означает, что каждый объект класса будет иметь уникальный идентификатор, который будет использоваться для связи с другими объектами и для поиска объектов в базе данных.
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY) // Создаем автоинкремент в БД. Всего есть 5 стратегий.
public Long getId() {
    return id;
}

@Override
public String toString() {
    return "student [id=" + id + ", first=" + first + ", last=" + last + ", num=" + num + ", av=" + av + "]";
}
```

Интерфейс StudentRepository

Интерфейс **StudentRepository** является репозиторием в Spring Data JPA. Репозитории в Spring Data JPA предоставляют абстракции над базовыми CRUD-операциями (**C**REATE – создание, **R**EAD – чтение, **U**PDATE – обновление, **D**ELETE – удаление) для работы с данными в базе данных.

Интерфейс **StudentRepository** расширяет (наследует) интерфейс Jpa**R**epository, который предоставляет базовые операции CRUD для сущностей, идентифицируемых по Long. **JpaRepository** автоматически генерируется Spring Data JPA на основе информации, полученной из аннотаций **@Entity** и **@Id** в классе **Student**.

Интерфейс **StudentRepository** также содержит метод **search**, который использует аннотацию **@Query** для определения SQL-запроса, который будет выполняться для поиска студентов по заданному поисковому запросу. Запрос использует функцию **CONCAT** для объединения различных полей студента в одно строковое значение, которое затем сравнивается с шаблоном, включающим **%?1%**, где **?1** представляет собой параметр запроса, который будет заполнен значением **keyword**.

```
package com.example.demo;

import java.util.List;

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

public interface StudentRepository extends JpaRepository<Student, Long> {
   @Query("SELECT p FROM Student p WHERE CONCAT(p.first, ' ', p.last, ' ', p.num, ' ', p.av) LIKE %?1%")
   List<Student> search(String keyword); // сформированный список, используемый во всем исходном коде. Все переменные в запросе берутся класса Student
}
```

Класс StudentService

```
package com.example.demo;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service
public class StudentService {
   @Autowired
   private StudentRepository repo; // Автоматически создаем и внедряем интерфейс StudentRepository и переименовываем в его repo. Вот так и выглядит внедрение зависимостей (dependency injection)
 //Этот метод возвращает список студентов, возможно, фильтруя их по заданному поисковому запросу. Если keyword не равен null, то используется метод search репозитория для поиска студентов по ключевому слову. В противном случае используется метод findAll для получения всех студентов. Сам список берем из нашей базы данных при помощи репозитория
   public List<Student> listAll(String keyword) {
      if (keyword != null) {
         return repo.search(keyword); // Созданный метод в репозитории
      }
      return repo.findAll(); // Встроенный в коллекцию метод
   }
 //Этот метод сохраняет студента в базе данных, используя репозиторий
   public void save(Student student) {
      repo.save(student);
   }
 //Этот метод получает студента по его идентификатору.
   public Student get(Long id) {
      return repo.findById(id).get();
   }
 //Этот метод удаляет студента из базы данных по его идентификатору
   public void delete(Long id) {
      repo.deleteById(id);
   }

}
```

Информационная система работы со студентами. Класс MvcConfig

Класс **MvcConfig** реализует интерфейс **WebMvcConfigurer**. Этот интерфейс содержит методы, которые позволяют настраивать поведение архитектурного паттерна Spring **MVC** (**Model-View-Controller**), включая добавление контроллеров, обработчиков исключений, фильтров и многое другое. Интерфейс **WebMvcConfigurer** имеет метод **addViewControllers**, который позволяет добавлять контроллеры в Spring MVC. В данном случае, метод **addViewControllers** пустой, что означает, что в приложении нет необходимости добавлять дополнительные контроллеры.

Класс **MvcConfig** помечен аннотацией **@Configuration**, что указывает Spring, что этот класс содержит конфигурационные настройки для приложения. Создается на тот случай, если недостаточно класса **AppController**.

```
package com.example.demo;

import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.ViewControllerRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@Configuration
public class MvcConfig implements WebMvcConfigurer {

    @Override
    public void addViewControllers(ViewControllerRegistry registry) {
    }

}
```

Информационная система работы со студентами. Класс AppController. Начало

```
package com.example.demo;

import java.util.List; // Импортируем коллекцию (класс списков)
import org.springframework.beans.factory.annotation.Autowired; // Аннотация @Autowired используется в Spring Framework для переопределения и автоматического внедрения зависимостей (dependency injection)
import org.springframework.data.repository.query.Param; // Аннотация @Param используется в Spring Data для параметризации методов репозитория. Она указывает, что параметр метода является параметром запроса, который будет использован в SQL-запросе или другом запросе к базе данных.
import org.springframework.stereotype.Controller; // Используется в Spring Framework для маркировки классов как контроллеров. Контроллеры в Spring MVC отвечают за обработку HTTP-запросов и возвращение ответов в виде представлений, JSON, XML и т.д.
import org.springframework.ui.Model; // Класс (Модель), который используется в Spring MVC для передачи данных из контроллера в представление
import org.springframework.web.bind.annotation.ModelAttribute; // Аннотация @ModelAttribute используется в Spring MVC для связывания параметров запроса с атрибутами модели. Это позволяет легко передавать данные из формы в контроллер и обратно
import org.springframework.web.bind.annotation.PathVariable; // Аннотация @PathVariable используется в Spring MVC для параметризации методов контроллера с использованием частей пути запроса. Это позволяет легко маршрутизировать запросы на основе определенных частей URL.
import org.springframework.web.bind.annotation.RequestMapping; // Задаем наш URl. Аннотация @RequestMapping используется в Spring MVC для указания, какие методы контроллера должны обрабатывать определенные HTTP-методы и пути
import org.springframework.web.bind.annotation.RequestMethod; // Какие запросы выполнять: POST или GET?
import org.springframework.web.servlet.ModelAndView; // Класс ModelAndView представляет собой абстрактный класс, который используется в Spring MVC для возвращения модели и представления из контроллера. Он объединяет модель, которая содержит данные для представления, и имя представления, которое будет отображено.
```

Информационная система работы со студентами. Класс AppController. Продолжение

```
@Controller // Весь наш класс будет контроллером
public class AppController { // Создаем класс с модификатором доступа Public, так как контроллер должен быит открытым полностью из-за аннотации @Controller

    @Autowired // Автоматически создаем и внедряем класс StudentService и переименовываем в его service. Вот так и выглядит внедрение зависимостей (dependency injection)
    private StudentService service;

    @RequestMapping("/") // Наша главная страница
    //1. Вызывается метод service.listAll(keyword), который, вероятно, возвращает список студентов, соответствующих заданному поисковому запросу.
    //2. Полученный список добавляется в модель под именем listStudents.
    //3. Значение keyword также добавляется в модель.
    //4. Возвращается имя представления index, которое будет отображаться в ответ на запрос.
    public String viewHomePage(Model model, @Param("keyword") String keyword) {
        //Реализация поиска на главной странице по критериям
        List<Student> listStudents = service.listAll(keyword); // Наш список студентов. Элементы в список передаются из класса StudentService
        model.addAttribute("listStudents", listStudents); //  Создаем модель и добавляем в нее атрибут. На главной странице будет выводиться список студентов
        model.addAttribute("keyword", keyword); // Создаем модель и добавляем в нее атрибут. На главной странице будет выводиться поиск студентов
        return "index"; // Выводится все то, что отображено в шаблоне index.html, модели будут туда также добавляться
    }
```

Информационная система работы со студентами. Класс AppController. Окончание

```
@RequestMapping("/new") // Добавление студента
  public String showNewStudentForm(Model model) {
    Student student = new Student(); // Создаем экземпляр класса Student, внутри которого "лежит" наша модель базы данных
    model.addAttribute("student", student); // Добавляем в модель данные (грубо говоря, модель данных в нашем случае - список)
    return "new_student"; // В браузере отображается все то, что есть в шаблоне new_student.html
  }

  @RequestMapping(value = "/save", method = RequestMethod.POST) // Передаем данные из модели (получили в методе
showNewStudentForm) в базу данных
  public String saveStudent(@ModelAttribute("student") Student student) {
    service.save(student); // Сохраяем наш список
    return "redirect:/"; // После сохранения данных нас перенаправит на главную страницу, где уже будет обновленный список с учетом
внесенных данных
  }

  @RequestMapping("/edit/{id}") // Страница редактирования данных о студенте
  public ModelAndView showEditStudentForm(@PathVariable(name = "id") Long id) {
    ModelAndView mav = new ModelAndView("edit_student"); // Добавляем шаблон в модель
    Student student = service.get(id); // Передаем ID, по которому будем редактировать
    mav.addObject("student", student);
    return mav; // Возвращаем полностью модель
  }

  @RequestMapping("/delete/{id}") // Удаялем студента
  public String deleteStudent(@PathVariable(name = "id") Long id) {
    service.delete(id);
    return "redirect:/";
  }
}
```

ФИНАНСОВЫЙ УНИВЕРСИТЕТ
ПРИ ПРАВИТЕЛЬСТВЕ РОССИЙСКОЙ ФЕДЕРАЦИИ

Информационная система работы со студентами. Главная страница (index.html)

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org" lang="ru">
<head>
   <meta charset="UTF-8">
   <title>Система работы со студентами</title>
   <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet"
        integrity="sha384-1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3" crossorigin="anonymous">
</head>
<body>
<div class="bg-image"
    style="background-image: url('https://palchevsky.ru/images/background.jpg'); height: 100vh; overflow: auto">
   <blockquote class="blockquote text-center"><h1>Студенты факультета ИТиАБД</h1></blockquote>
   <div class="row">
      <div class="col-md-8 offset-md-4">
         <h4>Поиск студента по любому критерию:</h4>
         <form th:action="@{/}">
            <input type="text" name="keyword" id="keyword" size="70" th:value="${keyword}" required/>
            <input type="submit" class="btn btn-success btn-sm" value="Поиск"/>
            <input type="button" class="btn btn-warning btn-sm" value="Очистить" id="btnClear"
                 onclick="clearSearch()"/>
         </form>
      </div>
   </div>
   <table id="1" class="table table-striped table-hover">
      <thead>
      <tr>
         <th scope="col">ID студента</th>
         <th scope="col">Имя</th>
         <th scope="col">Фамилия</th>
         <th scope="col">Номер студенческого</th>
         <th scope="col">Средний балл</th>
         <th scope="col">Действия</th>
      </tr>
      </thead>
```

ФИНАНСОВЫЙ УНИВЕРСИТЕТ
ПРИ ПРАВИТЕЛЬСТВЕ РОССИЙСКОЙ ФЕДЕРАЦИИ

Информационная система работы со студентами. Главная страница (index.html) - продолжение

```
  <tbody>
  <tr th:each="student: ${listStudents}">
    <th scope="row" class="text-white" th:text="${student.id}">ID студента отстутсвует</th>
    <th scope="row" class="text-white" th:text="${student.first}">Имя студента отсутствует</th>
    <th scope="row" class="text-white" th:text="${student.last}">Фамилия студента отсутствует</th>
    <th scope="row" class="text-white" th:text="${student.num}">Номер студенческого отсутствует</th>
    <th scope="row" class="text-white" th:text="${student.av}">Средний балл студента отсутствует</th>
    <td>
      <a th:href="@{'/edit/'+${student.id}}"><button type="button" class="btn btn-info">Редактировать</button></a>
      <a th:href="@{'/delete/'+${student.id}}"><button type="button" class="btn btn-danger">Удалить</button></a>
    </td>
  </tr>
  </tbody>
</table>
```

ФИНАНСОВЫЙ УНИВЕРСИТЕТ
ПРИ ПРАВИТЕЛЬСТВЕ РОССИЙСКОЙ ФЕДЕРАЦИИ

Информационная система работы со студентами. Главная страница (index.html) - окончание

```
    <p class="text-white">
        <script type="text/javascript">
            function getRowsColumn() {
                let table = document.getElementById('1')
                let tBody = table.querySelector('tbody')
                const count = tBody.querySelectorAll('tr').length;
                document.write('Количество студентов в таблице: ' + count)
            }

            getRowsColumn()
        </script>
    </p>
    <blockquote class="blockquote text-center">
        <a href="/new">
            <button type="button" class="btn btn-primary" data-togge="button" aria-pressed="false" autocomplete="off">
                Добавить студента
            </button>
        </a>
    </blockquote>
</div>
<script type="text/javascript">
    function clearSearch() {
        window.location = "[[@{/}]]";
    }
</script>
</body>
</html>
```

ФИНАНСОВЫЙ УНИВЕРСИТЕТ
ПРИ ПРАВИТЕЛЬСТВЕ РОССИЙСКОЙ ФЕДЕРАЦИИ

Информационная система работы со студентами. Главная страница (index.html) - окончание

```
    <p class="text-white">
        <script type="text/javascript">
            function getRowsColumn() {
                let table = document.getElementById('1')
                let tBody = table.querySelector('tbody')
                const count = tBody.querySelectorAll('tr').length;
                document.write('Количество студентов в таблице: ' + count)
            }

            getRowsColumn()
        </script>
    </p>
    <blockquote class="blockquote text-center">
        <a href="/new">
            <button type="button" class="btn btn-primary" data-togge="button" aria-pressed="false" autocomplete="off">
                Добавить студента
            </button>
        </a>
    </blockquote>
</div>
<script type="text/javascript">
    function clearSearch() {
        window.location = "[[@{/}]]";
    }
</script>
</body>
</html>
```

ФИНАНСОВЫЙ УНИВЕРСИТЕТ
ПРИ ПРАВИТЕЛЬСТВЕ РОССИЙСКОЙ ФЕДЕРАЦИИ

Информационная система работы со студентами. Главная страница (index.html) - окончание

```
    <p class="text-white">
        <script type="text/javascript">
            function getRowsColumn() {
                let table = document.getElementById('1')
                let tBody = table.querySelector('tbody')
                const count = tBody.querySelectorAll('tr').length;
                document.write('Количество студентов в таблице: ' + count)
            }

            getRowsColumn()
        </script>
    </p>
    <blockquote class="blockquote text-center">
        <a href="/new">
            <button type="button" class="btn btn-primary" data-togge="button" aria-pressed="false" autocomplete="off">
                Добавить студента
            </button>
        </a>
    </blockquote>
</div>
<script type="text/javascript">
    function clearSearch() {
        window.location = "[[@{/}]]";
    }
</script>
</body>
</html>
```

ФИНАНСОВЫЙ УНИВЕРСИТЕТ
ПРИ ПРАВИТЕЛЬСТВЕ РОССИЙСКОЙ ФЕДЕРАЦИИ

Информационная система работы со студентами. Добавление студента (new_student.html)

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org" lang="ru">
<title>Добавить студента</title>
<header th:replace="index :: head"></header>
<body>
<div class="bg-image"
    style="background-image: url('https://palchevsky.ru/images/background.jpg'); height: 100vh; overflow:
auto">
   <blockquote class="blockquote text-center"><h1>Добавить студента факультета
ИТиАБД</h1></blockquote>
```

ФИНАНСОВЫЙ УНИВЕРСИТЕТ
ПРИ ПРАВИТЕЛЬСТВЕ РОССИЙСКОЙ ФЕДЕРАЦИИ

Информационная система работы со студентами. Добавление студента (new_student.html) – продолжение

```
<div class="row">
  <div class="col-md-8 offset-md-4">
    <form action="#" th:action="@{/save}" th:object="${student}" method="post">
      <table>
        <tr>
          <td>ID студента:</td>
          <td><input type="text" th:field="*{id}"></td>
        </tr>
        <tr>
          <td>Имя:</td>
          <td><input type="text" th:field="*{first}"></td>
        </tr>
        <tr>
          <td>Фамилия:</td>
          <td><input type="text" th:field="*{last}"></td>
        </tr>
        <tr>
          <td>Номер студенческого:</td>
          <td><input type="text" th:field="*{num}"></td>
        </tr>
        <tr>
          <td>Средний балл:</td>
          <td><input type="text" onkeyup="this.value = this.value.replace(/\.(?=.*\.)|[^\d\.-]/g, '');"
                     onchange="this.value = this.value.replace(/\.(?=.*\.)|[^\d\.-]/g, '');"
                     th:field="*{av}"></td>
        </tr>
        <tr>
          <td>
            <button type="submit" class="btn btn-primary" data-tooggle="button" aria-pressed="false" autocomplete="off">Сохранить</button>
          </td></tr></table></form></div></div></div></body></html>
```

ФИНАНСОВЫЙ УНИВЕРСИТЕТ
ПРИ ПРАВИТЕЛЬСТВЕ РОССИЙСКОЙ ФЕДЕРАЦИИ

Информационная система работы со студентами. Редактирование студента (edit_student.html)

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org" lang="ru">
<title>Редактировать информацию о студенте</title>
<header th:replace="index :: head"></header>
<body>
<div class="bg-image"
    style="background-image: url('https://palchevsky.ru/images/background.jpg'); height: 100vh; overflow: auto">
   <blockquote class="blockquote text-center"><h1>Редактирование студента факультета ИТиАБД</h1></blockquote>
```

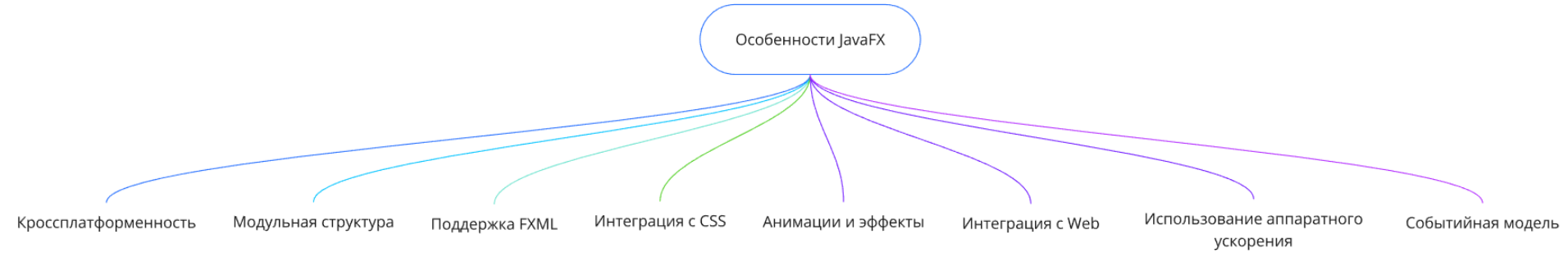
ФИНАНСОВЫЙ УНИВЕРСИТЕТ
ПРИ ПРАВИТЕЛЬСТВЕ РОССИЙСКОЙ ФЕДЕРАЦИИ

Информационная система работы со студентами. Редактирование студента (edit_student.html)

```
<div class="row">
  <div class="col-md-8 offset-md-4">
    <form action="#" th:action="@{/save}" th:object="${student}" method="post">
      <table>
        <tr>
          <td>ID студента:</td>
          <td><input type="text" th:field="*{id}" readonly="readonly"></td>
        </tr>
        <tr>
          <td>Имя:</td>
          <td><input type="text" th:field="*{first}"></td>
        </tr>
        <tr>
          <td>Фамилия:</td>
          <td><input type="text" th:field="*{last}"></td>
        </tr>
        <tr>
          <td>Номер студенческого:</td>
          <td><input type="text" th:field="*{num}"></td>
        </tr>
        <tr>
          <td>Средний балл:</td>
          <td><input type="text" onkeyup="this.value = this.value.replace(/\.(?=.*\.)|[^\d\.-]/g, '');"
                     onchange="this.value = this.value.replace(/\.(?=.*\.)|[^\d\.-]/g, '');"
                     th:field="*{av}"></td>
        </tr>
        <tr>
          <td colspan="2">
            <button type="submit" class="btn btn-primary" data-toggle="button" aria-pressed="false" autocomplete="off">Отправить</button>
          </td>
        </tr>
  </table></form></div></div></div></body></html>
```

Что такое JavaFX и для чего используется? Платформа JavaFX, особенности, компоненты. Часть 1

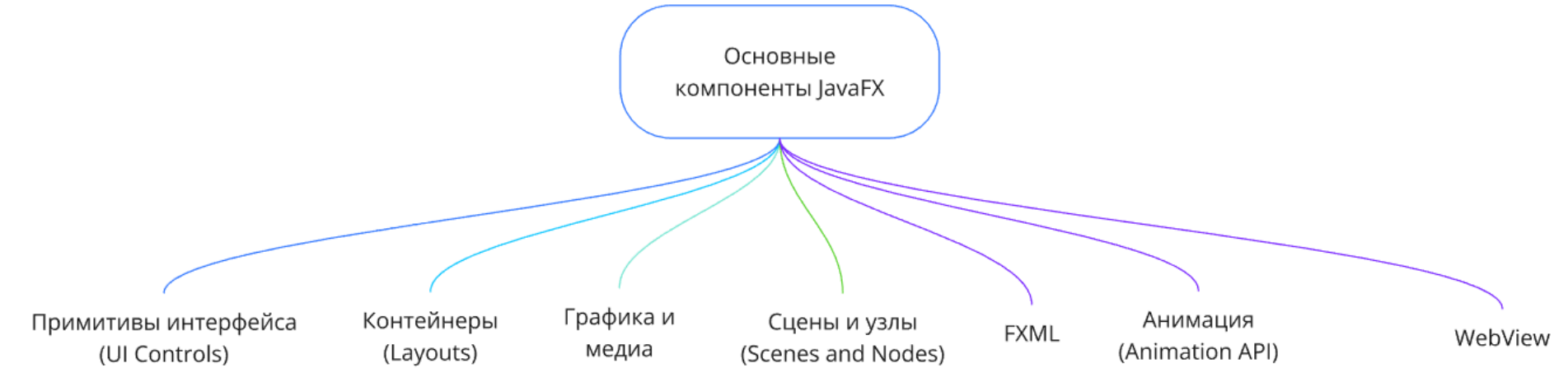
JavaFX — современная платформа для разработки графических пользовательских интерфейсов (Graphical user interface, GUI) на языке Java. Она является заменой старой технологии Swing и предоставляет более мощные инструменты для создания современных, кроссплатформенных приложений с богатым функционалом.



1. **Кроссплатформенность.** JavaFX поддерживает Windows, macOS и Linux. Приложения работают одинаково на всех платформах благодаря JVM.
2. **Модульная структура.** JavaFX с версии Java 11 выделена в отдельный модуль (OpenJFX), что позволяет использовать только те части платформы, которые нужны для проекта.
3. **Поддержка FXML.** FXML — это XML-формат для описания пользовательских интерфейсов, который отделяет визуальный слой от бизнес-логики. Это упрощает разработку и позволяет дизайнерам и разработчикам работать параллельно.
4. **Интеграция с CSS.** JavaFX поддерживает стилизацию интерфейсов через CSS, что позволяет легко настраивать внешний вид компонентов.
5. **Анимации и эффекты.** Платформа JavaFX предоставляет встроенный инструментарий для создания сложных анимаций и эффектов, например, размытие, тени и трансформации.

Что такое JavaFX и для чего используется? Платформа JavaFX, особенности, компоненты. Часть 2

6. **Интеграция с Web.** WebView — компонент JavaFX, предназначенный для встраивания веб-контента на основе движка WebKit.
7. **Использование аппаратного ускорения.** JavaFX использует аппаратное ускорение для рендеринга 2D и 3D графики, что делает её производительной для сложных визуализаций.
8. **Поддержка событийной модели.** Поддержка событийного программирования позволяет обрабатывать пользовательские действия, такие как клики, ввод текста и т.д.



1. **Примитивы интерфейса (UI Controls).** Label, Button, TextField, PasswordField — основные элементы интерфейса. ComboBox, ChoiceBox, Slider, ProgressBar, ListView, TableView — для работы с данными.
2. **Контейнеры (Layouts).** Используются для управления расположением компонентов. HBox и VBox: горизонтальное и вертикальное размещение. GridPane: таблицы с заданным количеством строк и столбцов. BorderPane: организация по краям и в центре. StackPane, AnchorPane — разработка более сложных макеты.

Что такое JavaFX и для чего используется? Платформа JavaFX, особенности, компоненты. Часть 3

3. **Графика и медиа.** Поддержка 2D/3D-графики. Класс Canvas для создания макета. Присутствуют компоненты для работы с аудио и видео.
4. **Сцены и узлы (Scenes and Nodes).** Класс ***Scene*** — базовый класс-контейнер для размещения элементов интерфейса. ***Node*** — базовый класс для всех графических элементов. ***Stage*** – класс, отвечающий за создание окна приложения. Это самый верхний уровень, отвечающий за отображение интерфейса. Класс ***Stage*** управляет классом ***Scene***, который, в свою очередь, содержит иерархию ***Node***.
5. **FXML.** FXML — это XML-формат для описания пользовательских интерфейсов, который отделяет визуальный слой от бизнес-логики. Это упрощает разработку и позволяет дизайнерам и разработчикам работать параллельно. Позволяет выполнять разметку интерфейсов в XML-формате и связывать элементы с логикой через контроллеры.
6. **Анимация (Animation API).** Animation – базовый класс для всех анимаций, поддерживающий управление временем (начало, пауза, остановка) и циклами повторений. Timeline – один из наиболее используемых классов для создания анимаций, позволяющий изменять значения свойств объекта во времени и использующий класс KeyFrame для задания временных точек, в которых свойства объекта изменяются.
7. **WebView**. Компонент для отображения веб-контента внутри JavaFX-приложения.

Пример реализации графического приложения с использованием JavaFX и MySQL. Предварительная настройка проекта

1. Обновление версии MySQL-connector в файле pom.xml

```
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>8.0.33</version>
</dependency>
```

2. Обновление версии javafx-controls в файле pom.xml

```
<dependency>
  <groupId>org.openjfx</groupId>
  <artifactId>javafx-controls</artifactId>
  <version>24-ea+15</version>
</dependency>
```

3. Обновление версии javafx-fxml в файле pom.xml

```
<dependency>
  <groupId>org.openjfx</groupId>
  <artifactId>javafx-fxml</artifactId>
  <version>24-ea+15</version>
</dependency>
```

4. Обновление версии maven-compiler-plugin в файле pom.xml

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-compiler-plugin</artifactId>
  <version>4.0.0-beta-1</version>
  <configuration>
    <source>23</source>
    <target>23</target>
  </configuration>
</plugin>
```

5. Указать главный класс в файле pom.xml. Данный класс отвечает за точку входа в программу JavaFX

```
<mainClass>com.example.demo3/com.example.demo3.App</mainClass>
```

Пример реализации графического приложения с использованием JavaFX и MySQL. Главный класс (MainApp.java) – часть 1

```
package com.example.demo4; // Основной пакет проекта

import javafx.application.Application; // Импорт базового класса JavaFX для приложений.
import javafx.scene.Scene; // Импорт класса для создания сцены.
import javafx.scene.control.Button; // Импорт класса для кнопок.
import javafx.scene.layout.VBox; // Импорт класса для вертикального размещения элементов.
import javafx.stage.Stage; // Импорт класса для создания основного окна.
import javafx.scene.control.Alert; // Импорт класса для отображения уведомлений.

import java.sql.Connection; // Импорт интерфейса для подключения к базе данных.
import java.sql.DriverManager; // Импорт класса для управления подключением к базе данных.
import java.sql.Statement; // Импорт интерфейса для выполнения SQL-запросов.

public class MainApp extends Application {

    /**
     * Метод для отображения уведомлений.
     *
     * @param alertType Тип уведомления (информация, ошибка и т.д.).
     * @param title     Заголовок уведомления.
     * @param message   Сообщение, отображаемое в уведомлении.
     */
    private void showAlert(Alert.AlertType alertType, String title, String message) {
        Alert alert = new Alert(alertType); // Создаем новое уведомление.
        alert.setTitle(title); // Устанавливаем заголовок.
        alert.setHeaderText(null); // Убираем заголовок уведомления.
        alert.setContentText(message); // Устанавливаем текст сообщения.
        alert.showAndWait(); // Показываем уведомление и ждем действия пользователя.
    }
```

Пример реализации графического приложения с использованием JavaFX и MySQL. Главный класс (MainApp.java) – часть 2

```
/**
 * Точка входа в JavaFX приложение.
 *
 * @param primaryStage Главное окно приложения.
 */
@Override
public void start(Stage primaryStage) {
    primaryStage.setTitle("JavaFX и MySQL"); // Устанавливаем заголовок главного окна.

    Button createTableButton = new Button("Создать таблицу"); // Кнопка для создания таблицы.
    Button addButton = new Button("Добавить данные"); // Кнопка для добавления данных.
    Button viewButton = new Button("Посмотреть данные"); // Кнопка для просмотра данных.

    // Логика для кнопки "Создать таблицу". event - алиас в лямба-выражении, обозначющий событие нажатия на кнопку.
    // Даже если в конкретной ситуации объект события (алиас) "event" не нужен, Java требует его указать, так как это часть сигнатуры (способа определения) метода,
который реализуется лямбдой.
    createTableButton.setOnAction(event -> createTable());

    // Логика для кнопки "Добавить данные".
    addButton.setOnAction(event -> AddDataWindow.display());

    // Логика для кнопки "Посмотреть данные".
    viewButton.setOnAction(event -> ViewDataWindow.display());

    VBox layout = new VBox(10, createTableButton, addButton, viewButton);
    // Создаем вертикальный контейнер для кнопок с отступом в 10 пикселей.

    Scene scene = new Scene(layout, 300, 200);
    // Создаем сцену с указанным размером (300x200 пикселей).

    primaryStage.setScene(scene); // Устанавливаем сцену в главное окно.
    primaryStage.show(); // Показываем главное окно.
}
```

Пример реализации графического приложения с использованием JavaFX и MySQL. Главный класс (MainApp.java) – часть 3

1

```
/**
 * Метод для создания таблицы в базе данных.
 */
private void createTable() {
    String url = "jdbc:mysql://localhost:3306/test"; // URL подключения к базе данных.
    String user = "root"; // Имя пользователя базы данных.
    String password = "root"; // Пароль пользователя базы данных. Замените на ваш.

    try (Connection conn = DriverManager.getConnection(url, user, password)) {
        // Устанавливаем подключение к базе данных.

        String query = """
                CREATE TABLE IF NOT EXISTS data (
                    id INT AUTO_INCREMENT PRIMARY KEY, -- Уникальный идентификатор записи.
                    name VARCHAR(100), -- Поле для имени.
                    value VARCHAR(100) -- Поле для значения.
                );
                """;
        // SQL-запрос для создания таблицы, если её ещё нет.

        Statement stmt = conn.createStatement(); // Создаем объект для выполнения SQL-запроса.
        stmt.execute(query); // Выполняем запрос.

        // Уведомление об успешном создании таблицы.
        showAlert(Alert.AlertType.INFORMATION, "Успех", "Таблица успешно создана!");
    } catch (Exception ex) { // Обрабатываем возможные ошибки.
        ex.printStackTrace(); // Выводим информацию об ошибке в консоль.
        showAlert(Alert.AlertType.ERROR, "Ошибка", "Не удалось создать таблицу!");
        // Уведомление о неудачной попытке создать таблицу.
    }
}
```

2

```
    /**
     * Точка входа в программу.
     *
     * @param args Аргументы командной строки.
     */
    public static void main(String[] args) {
        launch(args); // Запуск JavaFX приложения.
    }
}
```

Пример реализации графического приложения с использованием JavaFX и MySQL. Класс работы с данными (Data.java) для ввода и вывода данных из БД

```
package com.example.demo4; // Указание пакета, где находится класс `Data`.

/**
 * Record class `Data` для хранения данных записи.
 *
 * Record — это особый вид класса, который автоматически генерирует:
 * 1. Конструктор для всех полей.
 * 2. Методы-геттеры для всех полей (без префикса `get`).
 * 3. Методы `equals()`, `hashCode()` и `toString()`.
 *
 * Особенности record:
 * - Поля являются неизменяемыми (`final`).
 * - Объект record считается immutable (его состояние не может быть изменено после создания).
 * - Используется для упрощения кода при создании классов, которые предназначены для хранения данных.
 */
public record Data(int id, String name, String value) {
    // Поля `id`, `name`, `value` автоматически объявляются `private final`.
    // Конструктор, принимающий параметры (id, name, value), создается автоматически.
    // Методы `id()`, `name()`, `value()` позволяют получить значения полей.
}
```

Пример реализации графического приложения с использованием JavaFX и MySQL. Класс работы с данными (AddDataWindow.java) для добавления данных в БД – часть 1

```
package com.example.demo4; // Пакет, в котором находится класс `AddDataWindow`.

import javafx.scene.Scene; // Импорт класса для создания сцены.
import javafx.scene.control.Button; // Импорт класса для кнопок.
import javafx.scene.control.TextField; // Импорт класса для текстовых полей.
import javafx.scene.layout.VBox; // Импорт класса для вертикального размещения элементов.
import javafx.stage.Modality; // Импорт класса для управления модальностью окна.
import javafx.stage.Stage; // Импорт класса для создания нового окна.
import javafx.scene.control.Alert; // Импорт класса для отображения уведомлений.

import java.sql.Connection; // Импорт интерфейса для подключения к базе данных.
import java.sql.DriverManager; // Импорт класса для управления подключением к базе данных.
import java.sql.PreparedStatement; // Импорт интерфейса для выполнения параметризованных SQL-запросов.

public class AddDataWindow { // Объявление класса AddDataWindow.

    /**
     * Метод для отображения уведомления.
     *
     * @param alertType Тип уведомления (информация, ошибка и т.д.).
     * @param title     Заголовок уведомления.
     * @param message   Сообщение, отображаемое в уведомлении.
     */
    private static void showAlert(Alert.AlertType alertType, String title, String message) {
        Alert alert = new Alert(alertType); // Создаем новый объект Alert для отображения уведомления.
        alert.setTitle(title); // Устанавливаем заголовок окна уведомления.
        alert.setHeaderText(null); // Убираем заголовок внутри окна уведомления.
        alert.setContentText(message); // Устанавливаем текст сообщения.
        alert.showAndWait(); // Показываем уведомление и ждем, пока пользователь его закроет.
    }
```

Пример реализации графического приложения с использованием JavaFX и MySQL. Класс работы с данными (AddDataWindow.java) для добавления данных в БД – часть 2

```
/**
 * Метод отображает окно для ввода данных.
 */
public static void display() {
    Stage window = new Stage(); // Создаем новое окно (Stage) для ввода данных.
    window.initModality(Modality.APPLICATION_MODAL); // Устанавливаем модальность, чтобы блокировать основное окно.
    window.setTitle("Добавить данные"); // Устанавливаем заголовок окна.

    TextField idField = new TextField(); // Создаем текстовое поле для ввода ID.
    idField.setPromptText("Введите ID (или оставьте пустым для автоинкремента)"); // Подсказка для пользователя.

    TextField nameField = new TextField(); // Создаем текстовое поле для ввода имени.
    nameField.setPromptText("Введите имя"); // Подсказка для пользователя.

    TextField valueField = new TextField(); // Создаем текстовое поле для ввода значения.
    valueField.setPromptText("Введите значение"); // Подсказка для пользователя.

    Button saveButton = new Button("Сохранить"); // Создаем кнопку для сохранения данных.
    saveButton.setOnAction(e -> { // Назначаем обработчик события нажатия кнопки.
        String idText = idField.getText(); // Считываем текст из поля ID.
        int id = idText.isEmpty() ? 0 : Integer.parseInt(idText); // Если поле пустое, присваиваем ID значение 0.
        String name = nameField.getText(); // Считываем текст из поля имени.
        String value = valueField.getText(); // Считываем текст из поля значения.

        Data data = new Data(id, name, value); // Создаем объект Data с введенными пользователем данными.

        saveToDatabase(data); // Сохраняем данные в базу данных.
        window.close(); // Закрываем окно после сохранения.
    });

    VBox layout = new VBox(10, idField, nameField, valueField, saveButton);
    // Создаем вертикальный макет (VBox) с отступом в 10 пикселей и добавляем элементы.

    Scene scene = new Scene(layout, 300, 250); // Создаем сцену с указанным размером (300x250 пикселей).
    window.setScene(scene); // Устанавливаем сцену в окно.
    window.showAndWait(); // Показываем окно и ждем его закрытия.
}
```

Пример реализации графического приложения с использованием JavaFX и MySQL. Класс работы с данными (AddDataWindow.java) для добавления данных в БД – часть 3

```
/**
 * Метод для сохранения данных в базу данных.
 *
 * @param data Объект класса Data, содержащий ID, имя и значение.
 */
private static void saveToDatabase(Data data) {
    String url = "jdbc:mysql://localhost:3306/test"; // Адрес базы данных (URL подключения).
    String user = "root"; // Имя пользователя базы данных.
    String password = "root"; // Пароль пользователя базы данных.

    try (Connection conn = DriverManager.getConnection(url, user, password)) {
        // Устанавливаем подключение к базе данных.

        String query; // Переменная для хранения SQL-запроса.
        if (data.id() == 0) {
            // Если ID = 0, используем автоинкремент и исключаем ID из запроса.
            query = "INSERT INTO data (name, value) VALUES (?, ?)";
        } else {
            // Если ID задан, включаем его в SQL-запрос.
            query = "INSERT INTO data (id, name, value) VALUES (?, ?, ?)";
        }

        PreparedStatement stmt = conn.prepareStatement(query); // Создаем подготовленный запрос.

        if (data.id() == 0) {
            // Если ID = 0, устанавливаем только параметры для имени и значения.
            stmt.setString(1, data.name()); // Устанавливаем значение для имени.
            stmt.setString(2, data.value()); // Устанавливаем значение для значения.
        } else {
            // Если ID задан, устанавливаем параметры для ID, имени и значения.
            stmt.setInt(1, data.id()); // Устанавливаем значение для ID.
            stmt.setString(2, data.name()); // Устанавливаем значение для имени.
            stmt.setString(3, data.value()); // Устанавливаем значение для значения.
        }
        stmt.executeUpdate(); // Выполняем SQL-запрос.
        showAlert(Alert.AlertType.INFORMATION, "Успех", "Данные успешно добавлены!");
        // Показываем уведомление об успешном добавлении данных.
    } catch (Exception ex) {
        ex.printStackTrace(); // Выводим информацию об ошибке в консоль.
        showAlert(Alert.AlertType.ERROR, "Ошибка", "Не удалось добавить данные!");
        // Показываем уведомление о неудачной попытке добавить данные.
    }
}
```

Пример реализации графического приложения с использованием JavaFX и MySQL. Класс работы с данными (ViewDataWindow.java) для вывода данных из БД – часть 1

```
package com.example.demo4;

import javafx.beans.property.ReadOnlyObjectWrapper; // Импорт для создания оберток свойств.
import javafx.collections.FXCollections; // Импорт для работы с ObservableList.
import javafx.collections.ObservableList; // Импорт интерфейса для создания списка.
import javafx.scene.Scene; // Импорт класса для создания сцены.
import javafx.scene.control.TableColumn; // Импорт класса для создания столбцов таблицы.
import javafx.scene.control.TableView; // Импорт класса для создания таблицы.
import javafx.stage.Stage; // Импорт класса для создания нового окна.

import java.sql.Connection; // Импорт интерфейса для подключения к базе данных.
import java.sql.DriverManager; // Импорт класса для управления подключением к базе данных.
import java.sql.ResultSet; // Импорт интерфейса для обработки результатов SQL-запросов.
import java.sql.Statement; // Импорт интерфейса для выполнения SQL-запросов.

/**
 * Класс для отображения данных из базы в виде таблицы.
 */
public class ViewDataWindow {

    /**
     * Метод отображает новое окно с таблицей, в которой выводятся данные из базы данных.
     */
    public static void display() {
        Stage window = new Stage(); // Создаем новое окно (Stage) для отображения данных.
        window.setTitle("Сохраненные данные"); // Устанавливаем заголовок окна.

        TableView<Data> table = new TableView<>(); // Создаем таблицу для отображения данных.
```

Пример реализации графического приложения с использованием JavaFX и MySQL. Класс работы с данными (ViewDataWindow.java) для вывода данных из БД – часть 2

```
        // Создаем первый столбец для ID.
        TableColumn<Data, Integer> idColumn = new TableColumn<>("ID");
        idColumn.setCellValueFactory(cellData -> new
ReadOnlyObjectWrapper<>(cellData.getValue().id()));

        // Создаем второй столбец для имен.
        TableColumn<Data, String> nameColumn = new TableColumn<>("Name");
        nameColumn.setCellValueFactory(cellData -> new
ReadOnlyObjectWrapper<>(cellData.getValue().name()));

        // Создаем третий столбец для значений.
        TableColumn<Data, String> valueColumn = new TableColumn<>("Value");
        valueColumn.setCellValueFactory(cellData -> new
ReadOnlyObjectWrapper<>(cellData.getValue().value()));

        table.getColumns().addAll(idColumn, nameColumn, valueColumn); //
Добавляем все столбцы в таблицу.

        table.setItems(fetchDataFromDatabase()); // Устанавливаем данные из
базы данных в таблицу.

        Scene scene = new Scene(table, 400, 300); // Создаем сцену с таблицей
размером 400x300 пикселей.
        window.setScene(scene); // Устанавливаем сцену в окно.
        window.show(); // Отображаем окно.
    }
```

```
    /**
     * Метод подключается к базе данных, выполняет SQL-запрос и возвращает список
данных.
     *
     * @return ObservableList<Data> - список данных для таблицы.
     */
    private static ObservableList<Data> fetchDataFromDatabase() {
        ObservableList<Data> dataList = FXCollections.observableArrayList(); // Создаем список
данных для таблицы.

        String url = "jdbc:mysql://localhost:3306/test"; // URL подключения к базе данных.
        String user = "root"; // Имя пользователя базы данных.
        String password = "root"; // Пароль пользователя базы данных.

        try (Connection conn = DriverManager.getConnection(url, user, password)) { //
Устанавливаем подключение к базе данных.
            String query = "SELECT * FROM data"; // SQL-запрос для получения всех данных из
таблицы.
            Statement stmt = conn.createStatement(); // Создаем объект для выполнения SQL-
запросов.
            ResultSet rs = stmt.executeQuery(query); // Выполняем запрос и получаем результат.

            while (rs.next()) { // Перебираем все строки результата запроса.
                dataList.add(new Data(rs.getInt("id"), rs.getString("name"), rs.getString("value")));
                // Создаем объект `Data` (record) и добавляем его в список.
            }
        } catch (Exception ex) { // Обрабатываем возможные ошибки.
            ex.printStackTrace(); // Выводим ошибку в консоль.
        }

        return dataList; // Возвращаем список данных для таблицы.
    }
}
```

Пример реализации графического приложения с использованием JavaFX и MySQL. Результаты – часть 1

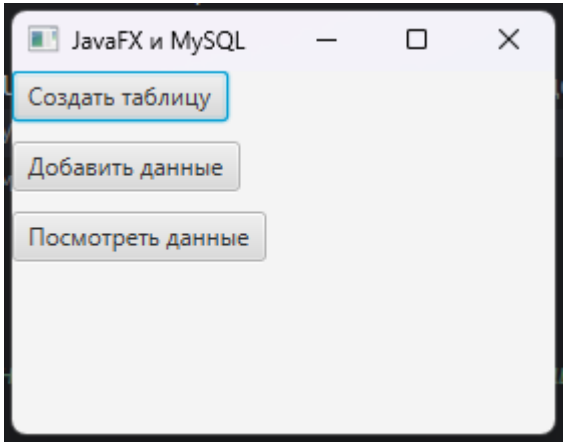


Рисунок 1 – Главное окно программы

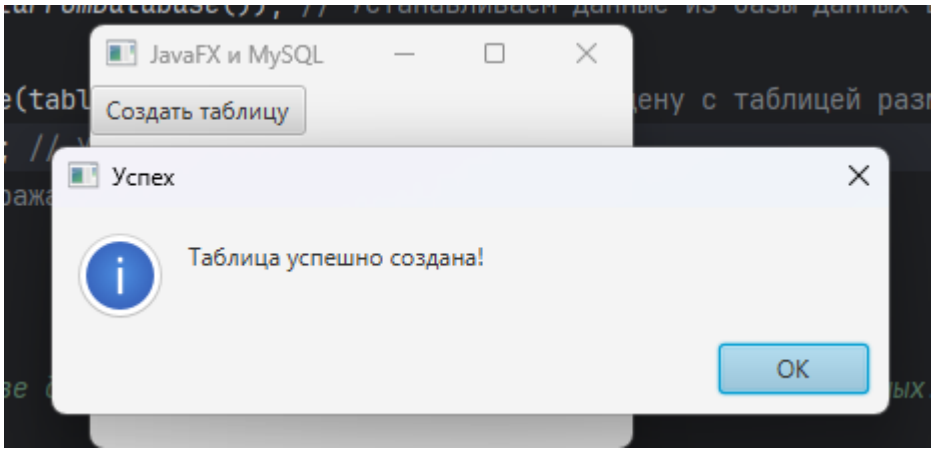


Рисунок 2 – Вывод сообщения об успешном создании таблицы в базе данных «test».

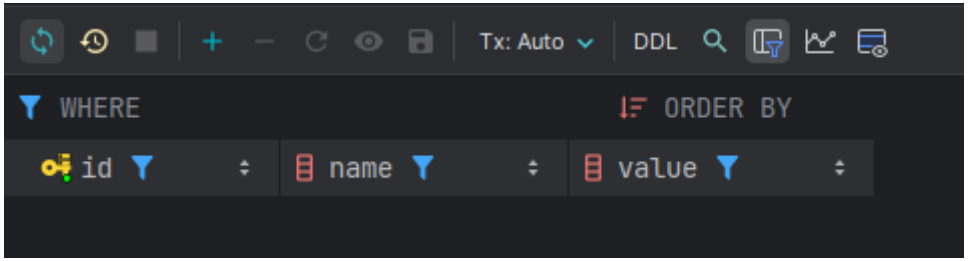


Рисунок 3 – Созданная таблица в базе данных «test».

ФИНАНСОВЫЙ УНИВЕРСИТЕТ
ПРИ ПРАВИТЕЛЬСТВЕ РОССИЙСКОЙ ФЕДЕРАЦИИ

Пример реализации графического приложения с использованием JavaFX и MySQL. Результаты – часть 2

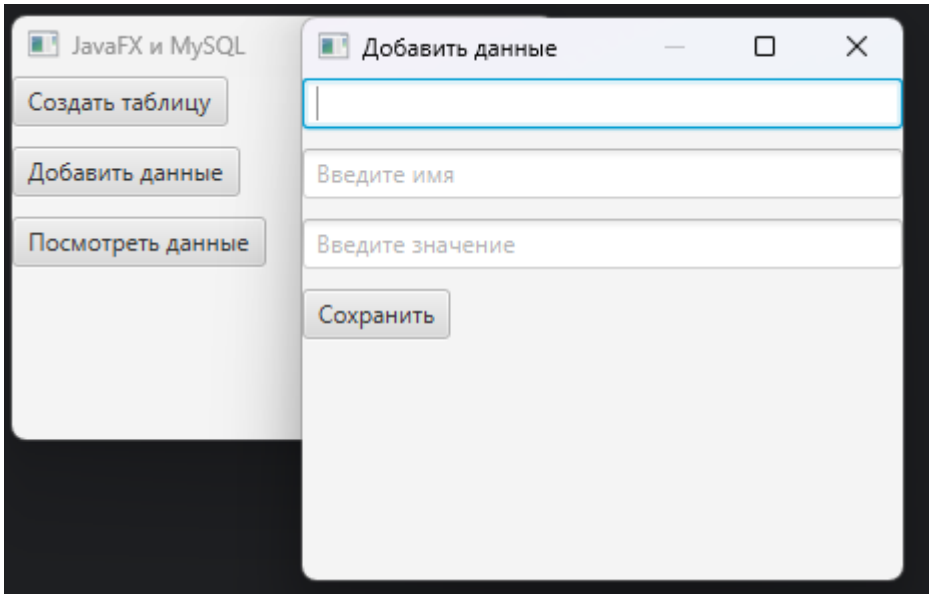


Рисунок 4 – Окно добавления данных при нажатии на кнопку «Добавить данные»

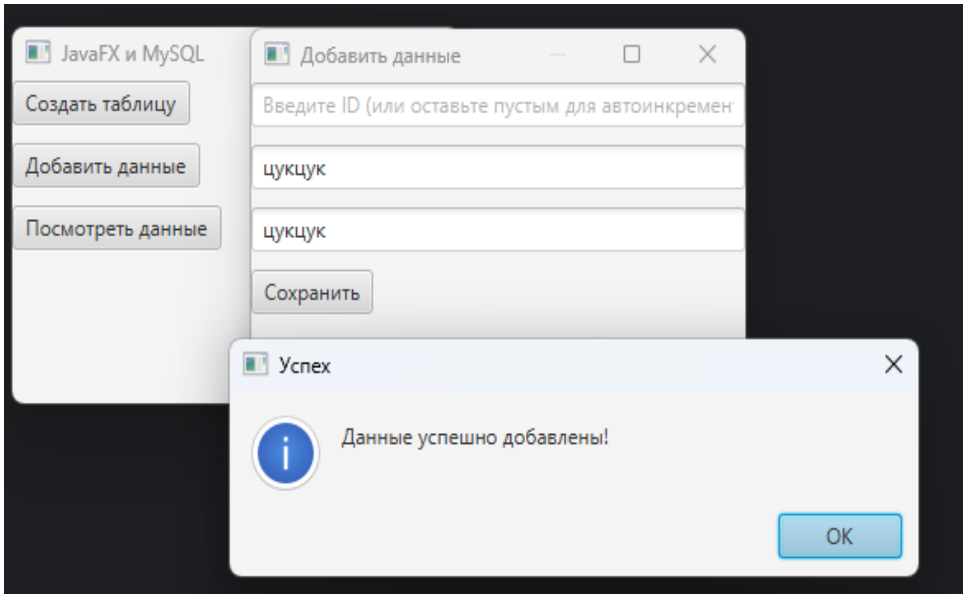


Рисунок 5 – Сообщение об успешном добавлении информации в таблицу

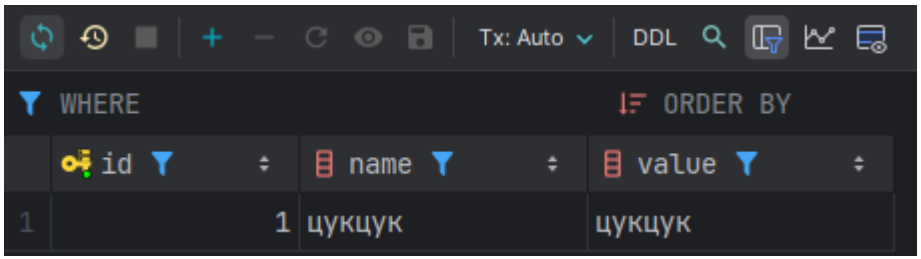


Рисунок 6 – Добавленные в таблицу данные из графического приложения

ФИНАНСОВЫЙ УНИВЕРСИТЕТ
ПРИ ПРАВИТЕЛЬСТВЕ РОССИЙСКОЙ ФЕДЕРАЦИИ

Пример реализации графического приложения с использованием JavaFX и MySQL. Результаты – часть 3

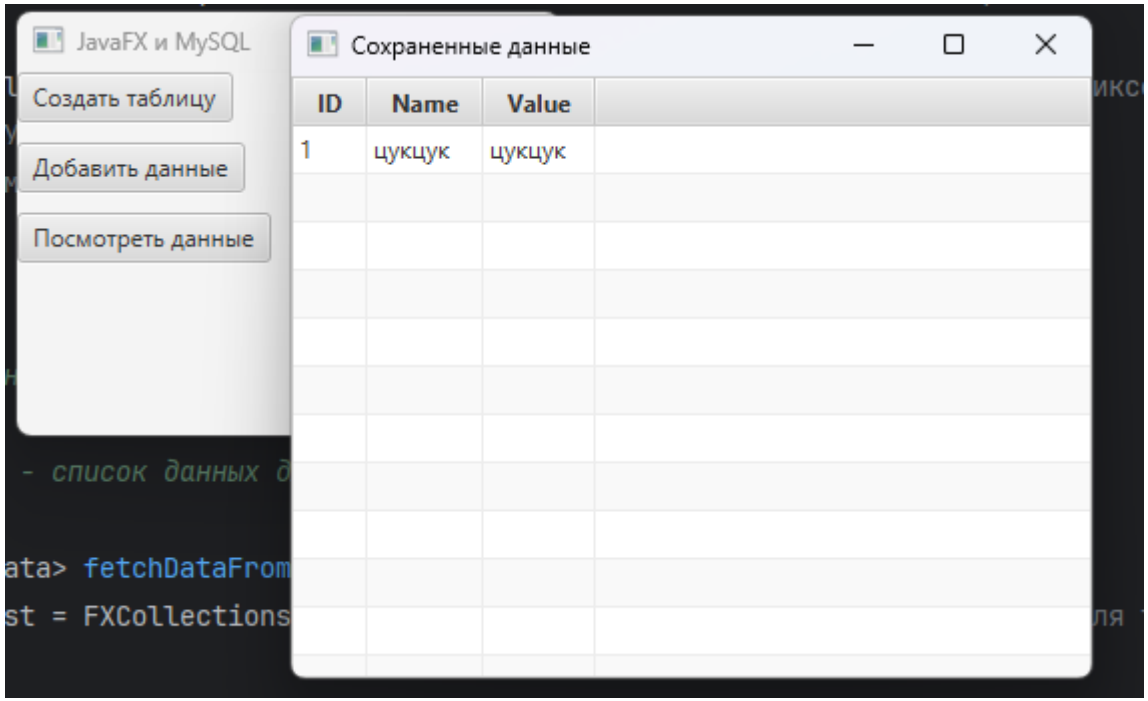


Рисунок 7 – Вывод данных из таблицы СУБД MySQL при нажатии на кнопку «Посмотреть данные»

Семинарское занятие №18

Задача №18. Сложный вариант

1. Java (ORM + Spring Boot), базы данных (MySQL), веб-программирование (HTML, CSS, JavaScript). Реализовать информационную систему «Библиотека» с веб-интерфейсом.

Необходимый функционал: добавление книги, удаление книги, редактирование книги, поиск книги по различным параметрам, отдельный фильтр (сортировка) по дате выдачи книги (на Java или JavaScript), гистограмма количества выдачи книг (на JavaScript, либо на Java) по дням, счетчик книг в таблице (на JavaScript, либо на Java).

Необходима стандартная форма авторизации от Spring Boot.

Параметры объекта «книга»: ID, Название книги, Издательство, Дата выдачи книги студенту, ФИО студента, Дата сдачи книги студентом в библиотеку.

Семинарское занятие №19

Задача №19. Сложный вариант

1. Java (ORM + Spring Boot), базы данных (MySQL), веб-программирование (HTML, CSS, JavaScript). Реализовать информационную систему «Автопарк» с веб-интерфейсом.

Необходимый функционал: добавление машины, удаление машины, редактирование машины, поиск машины по различным параметрам, отдельный фильтр (сортировка) по дате выпуска машины, гистограмма количества машин в автопарке (на JavaScript, либо на Java) по дням, счетчик элементов в таблице (на JavaScript, либо на Java).

Необходима стандартная форма авторизации от Spring Boot.

Параметры объекта «машина»: ID, Марка, Год выпуска, Дата поставки на учет в автопарке, ФИО владельца.

Семинарское занятие №20

Задача №20. Сложный вариант

1. Java (ORM + Spring Boot), базы данных (MySQL), веб-программирование (HTML, CSS, JavaScript). Реализовать информационную систему «Кинотеатр» с веб-интерфейсом.

Необходимый функционал: добавление сеанса, удаление сеанса, редактирование сеанса, поиск сеанса по различным параметрам, отдельный фильтр (сортировка) по дате сеанса (на Java или JavaScript), гистограмма количества сеансов (на JavaScript, либо на Java) по дням, счетчик сеансов в таблице (на JavaScript, либо на Java).

Необходима стандартная форма авторизации от Spring Boot.

Параметры объекта «сеанс»: ID, Название фильма, Киностудия, Дата и время сеанса, Количество билетов на сеанс.

Семинарское занятие №21

Задача №21. Сложный вариант. Контрольная работа №1

1. Java (ORM + Spring Boot), базы данных (MySQL), веб-программирование (HTML, CSS, JavaScript). Реализовать информационную систему «Склад магазина техники» с веб-интерфейсом.

Необходимый функционал: добавление техники, удаление техники, редактирование техники, поиск техники по различным параметрам, отдельный фильтр (сортировка) по дате выпуска машины, гистограмма количества отгруженной техники (на JavaScript, либо на Java) по дням, счетчик элементов (техники) в таблице (на JavaScript, либо на Java).

Необходима стандартная форма авторизации от Spring Boot.

Параметры объекта «машина»: ID, Вид техники (например, мультиварка), Группа техники (например, бытовые приборы), Дата ввоза на склад, Дата вывоза со склада, ФИО водителя, увозящего технику со склада.

Семинарское занятие №22

Задача №22. Сложный вариант

1. Java (ORM + Spring Boot), базы данных (MySQL), веб-программирование (HTML, CSS, JavaScript). Реализовать информационную систему «Театр» с веб-интерфейсом.

Необходимый функционал: добавление спектакля, удаление спектакля, редактирование спектакля, поиск спектакля по различным параметрам, отдельный фильтр (сортировка) по дате спектакля (на Java или JavaScript), гистограмма количества спектаклей (на JavaScript, либо на Java) по дням, счетчик спектаклей в таблице (на JavaScript, либо на Java), подсчет количества спектаклей за один день (на JavaScript, либо на Java).

Необходима стандартная форма авторизации от Spring Boot.

Параметры объекта «сеанс»: ID, Название спектакля, Название коллектива актеров, Дата и время спектакля, Общее количество билетов на спектакль, количество свободных билетов на спектакль.

Семинарское занятие №23

Задача №23. Сложный вариант

1. Java (ORM + Spring Boot), базы данных (MySQL), веб-программирование (HTML, CSS, JavaScript). Реализовать информационную систему «Барбершоп» с веб-интерфейсом.

Необходимый функционал: добавление клиента, удаление клиента, редактирование клиента, поиск клиента по различным параметрам, отдельный фильтр (сортировка) по дате записи клиента, гистограмма количества записей клиентов (на JavaScript, либо на Java) по дням, счетчик элементов (клиентов) в таблице (на JavaScript, либо на Java), подсчет количества клиентов за один день (на JavaScript, либо на Java), среднее количество клиентов в день (на Java, либо на JavaScript).

Необходима стандартная форма авторизации от Spring Boot.

Параметры объекта «Клиент»: ID, ФИО, Дата записи, Услуга, Обслуживающий мастер.

Семинарское занятие №24

Задача №24. Сложный вариант

1. Java (ORM + Spring Boot), базы данных (MySQL), веб-программирование (HTML, CSS, JavaScript). Реализовать информационную систему «Перевозки грузов» с веб-интерфейсом.

Необходимый функционал: добавление груза, удаление груза, редактирование груза, поиск груза по различным параметрам, отдельный фильтр (сортировка) по дате поставки груза (на Java или JavaScript), гистограмма количества грузов (на JavaScript, либо на Java) по дням, счетчик грузов в таблице (на JavaScript, либо на Java), подсчет количества прихода грузов за один день (на JavaScript, либо на Java).

Необходима стандартная форма авторизации от Spring Boot.

Параметры объекта «Груз»: ID, Название груза, Содержимое груза, Город отправки груза, Дата отправки груза, Город прибытия груза, Дата прибытия груза.

Семинарское занятие №25

Задача №25. Сложный вариант. Контрольная работа №2

1. Java (ORM + Spring Boot), базы данных (MySQL), веб-программирование (HTML, CSS, JavaScript). Реализовать информационную систему «Научно-практические конференции» с веб-интерфейсом.

Необходимый функционал: добавление конференции, удаление конференции, редактирование конференции, поиск конференции по различным параметрам, отдельный фильтр (сортировка) по дате проведения конференции, гистограмма количества конференций (на JavaScript, либо на Java) по дням, счетчик элементов (конференций) в таблице (на JavaScript, либо на Java), подсчет количества проведенных конференций за один день (на JavaScript, либо на Java), среднее количество конференций в день (на Java, либо на JavaScript).

Необходима стандартная форма авторизации от Spring Boot.

Параметры объекта «Конференция»: ID, Название конференции, Дата проведения, ФИО модератора, ФИО ведущего спикера.

Семинарское занятие №26

Задача №26. Сложный вариант

1. Java (ORM + Spring Boot), базы данных (MySQL), веб-программирование (HTML, CSS, JavaScript). Реализовать информационную систему «Перевозки грузов» с веб-интерфейсом на основе задачи **№24**.

Помимо описанного в задаче **№24** функционала добавить: систему регистрации и авторизации с разграничением прав доступа (какой функционал будет доступен определенной группе пользователей определяет сам разработчик). При этом авторизацию от Spring Boot необходимо убрать.

Семинарское занятие №27

Задача №27. Сложный вариант

1. Java (ORM + Spring Boot), базы данных (MySQL), веб-программирование (HTML, CSS, JavaScript). Реализовать информационную систему «Перевозки грузов» с веб-интерфейсом на основе задачи **№26**.

Помимо описанного в задаче **№26** функционала добавить: раздел «Автоблог» (ссылка на него должна быть в верхнем меню), в котором должны быть доступны следующие подразделы: «Административная панель управления блогом», в котором необходимо добавлять/удалять/редактировать записи; «Главная страница блога», на которой выводятся все добавленные через административную панель управления записи (Название записи, дата, кто добавил, текст записи). При добавлении записей необходимо соблюдать следующие поля: «ID записи», «Название записи», «Дата опубликования записи», «Текст записи».

Семинарское занятие №28

Задача №28. Сложный вариант. Контрольная работа №3

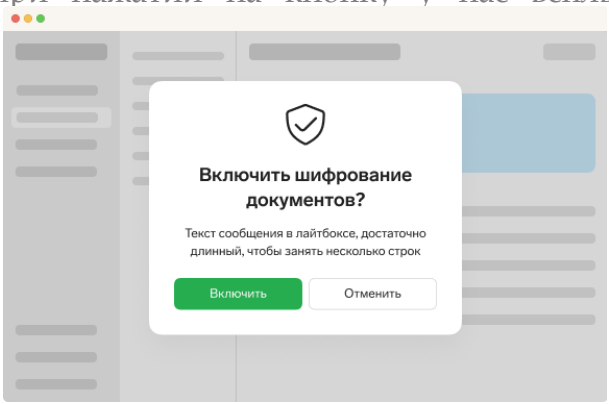
1. Java (ORM + Spring Boot), базы данных (MySQL), веб-программирование (HTML, CSS, JavaScript). Реализовать информационную систему «Перевозки грузов» с веб-интерфейсом на основе задачи **№27**.

Помимо описанного в задаче **№27** функционала добавить: в разделе «Автоблог» на главной странице через бэкенд реализовать единый поиск (т.е. поиск один, а полей для поиска несколько) по записям по нескольким критериям: «Поиск по дате записи», «Поиск по названию», «Поиск по дате и названию», «Поиск по тексту записи», «Поиск по дате и тексту записи», «Поиск по всем критериям сразу».

Семинарское занятие №29

Задача №29. Сложный вариант

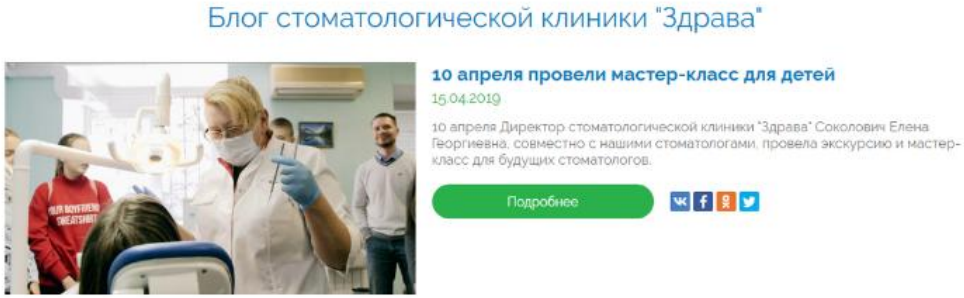
1. Java (ORM + Spring Boot), базы данных (MySQL), веб-программирование (HTML, CSS, JavaScript). В рамках реализованной в задаче **№28** информационной системы «Перевозка грузов» необходимо: реализовать вывод действий и результат выполнения функций «Добавление/удаления/редактирования записей», «Регистрация и авторизация пользователей» в виде модального окна Т.е. при нажатии на кнопку у нас всплывает окошко с необходимыми действиями/результатами. Пример:



Семинарское занятие №30

Задача №30. Сложный вариант

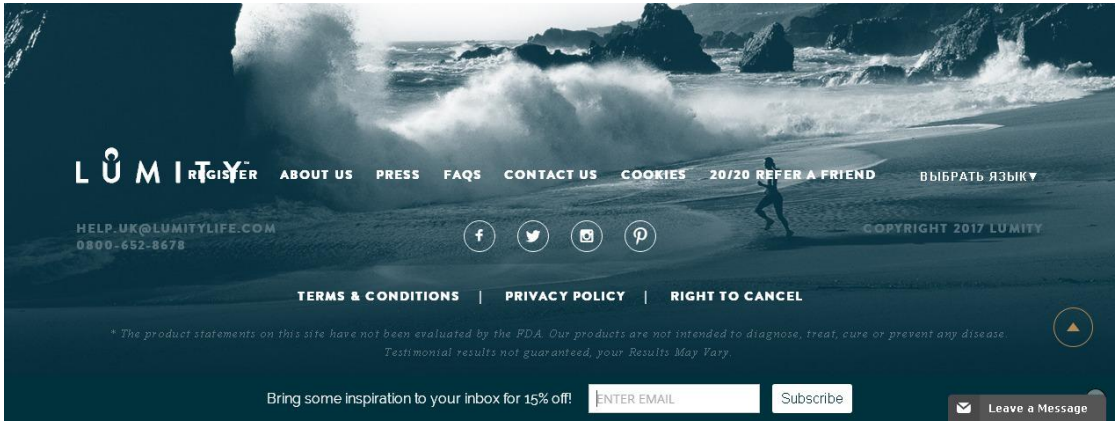
1. ***Java (ORM + Spring Boot), базы данных (MySQL), веб-программирование (HTML, CSS, JavaScript).*** В рамках реализованной в задаче **№29** информационной системы «Перевозка грузов» необходимо: оформить записи в блоге в следующем виде:



Семинарское занятие №31

Задача №31. Сложный вариант

1. Java (ORM + Spring Boot), базы данных (MySQL), веб-программирование (HTML, CSS, JavaScript). В рамках реализованной в задаче **№30** информационной системы «Перевозка грузов» необходимо: реализовать красивый футер системы, в том числе и блога. Пример:



Семинарское занятие №32

Задача №32. Сложный вариант. Контрольная работа №4

1. ***Java (ORM + Spring Boot), базы данных (MySQL), веб-программирование (HTML, CSS, JavaScript).*** В рамках реализованной в задаче **№31** информационной системы «Перевозка грузов» необходимо реализовать переключатель языков интерфейса (выбор должен быть у каждого пользователя):