Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Владимирский государственный университет
имени Александра Григорьевича и Николая Григорьевича Столетовых»
(ВлГУ)

Кафедра информационных систем и программной инженерии

Лабораторная работа №1
по дисциплине "Распределенные программные системы"

ТЕМА РАБОТЫ:
Разработка веб-приложения на базе Spring MVC

Выполнил:
студент гр. ПРИ-120
Парахин К.В.

Приняла:
Проскурина Г.В.

Владимир 2022 г.

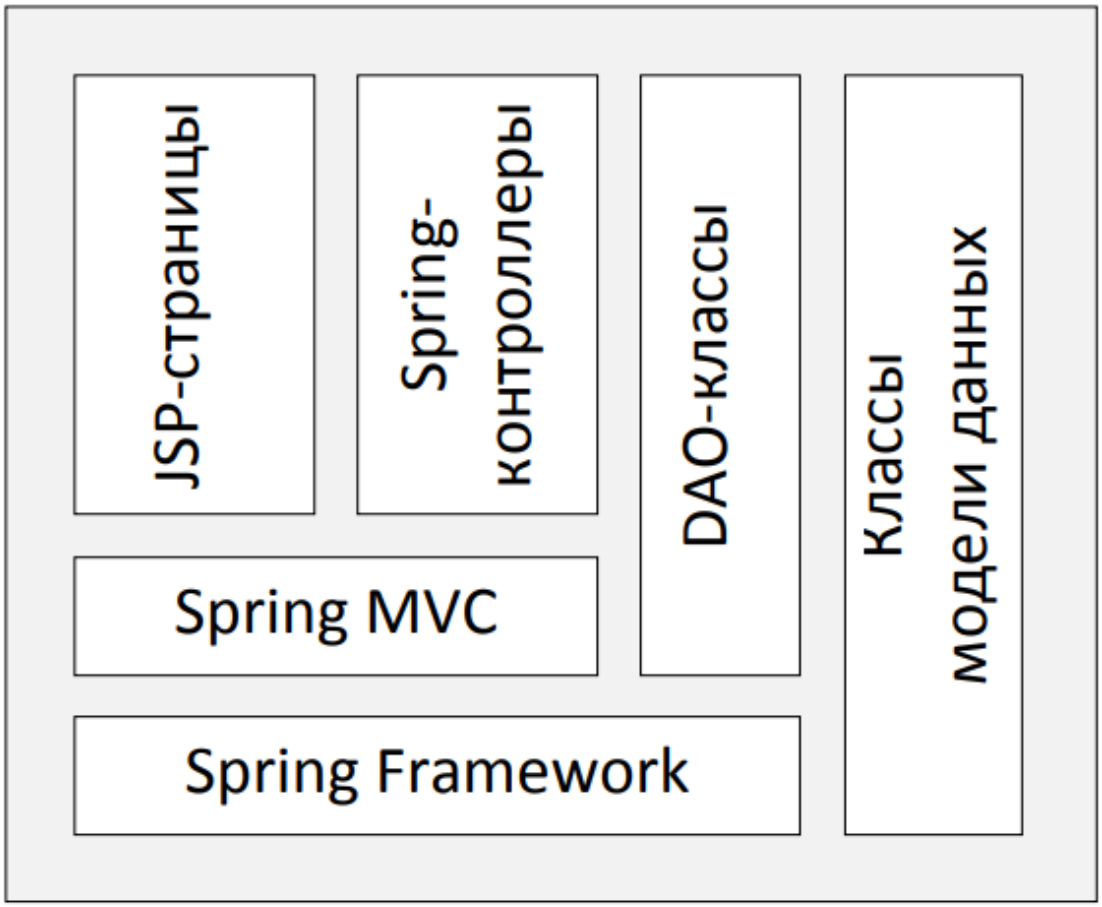
Цель работы:

Познакомиться с принципами построения веб-приложений с использованием шаблона MVC и на базе принципов REST, получить практические навыки реализации MVC - приложения на платформе Spring MVC.

Выполнение работы:

1. **Платформа Spring и инъекция зависимостей**

   Согласно постановке задачи лабораторной работы, архитектура разрабатываемог веб-приложения должна выглядеть следующий образом:



Для создания Spring MVC – приложения подобной архитектуры – необходимо разделить логику и методы контроллера, сделав тем самым слой сервиса и слой логики, который используется сервисом благодаря механизму инъекции зависимостей (Dependency Injection).

В остальном, приложение из предыдущей лабораторной работы удовлетворяет всем требованиям представленной выше архитектуры, а именно: содержит Spring – контроллеры (Controllers), JSP – страницы (Views), а также классы с моделями

данных, то есть Beans (Models). Отдельно особняком стоят DAO – классы, используемые для взаимодействия моделей, логики сервиса и БД.

Какие действия были проделаны в данной лабораторной работе:

1) Были созданы интерфейсы логики (IConnector и IHandler) и реализующие их классы (Connector и Handler). Класс Connector осуществляет подключение к БД (рис.1), а класс Handler содержит в себе методы взаимодействия сервиса с DAO – классами и моделями данных (Beans) (рис.2)

```java
public class Connector implements IConnector {
    private final String userName = "postgres";
    private final String password = "root";
    private final String dbms = "postgresql";
    private final String serverName = "localhost";
    private final String portNumber = "5432";
    private final String DBName = "SportSpring";

    public Connection getConnection() throws SQLException {
        try {
            Class.forName("org.postgresql.Driver").newInstance();
        } catch (ClassNotFoundException ex){
            System.err.println("Драйвер PostgreSQL не найден");
        } catch (InstantiationException e) {
            throw new RuntimeException(e);
        } catch (IllegalAccessException e1) {
            throw new RuntimeException(e1);
        }
        Connection conn = null;
        Properties connectionProps = new Properties();
        connectionProps.put("user", userName);
        connectionProps.put("password", password);
        conn = DriverManager.getConnection(
                url: "jdbc:" + dbms + "://" +
                        serverName + ":" +
                        portNumber + "/" +
                        DBName,
                connectionProps);
        if (conn != null)
            System.out.println("Connected to database " + DBName);
        else
            throw new SQLException("Database was not found and connected");
        return conn;
    }
}
```

Рисунок 1. Connector.java

```java
public class Handler implements IHandler {
    @Autowired
    private IClubDAO clubDAO;

    @Autowired
    private ISportsmanDAO sportsmanDAO;

    @Autowired
    private IConnector _connector;

    @Override
    public void ClubFormOpen(Long id, Model model) {
        Connection connection = null;
        try{
            connection = _connector.getConnection();
            if (id == null) {
                model.addAttribute( attributeName: "isCreate",  attributeValue: true);
                model.addAttribute( attributeName: "club", new Club());
            }
            else {
                model.addAttribute( attributeName: "isCreate",  attributeValue: false);
                model.addAttribute( attributeName: "club", clubDAO.getById(id, connection));
            }
            System.out.println(clubDAO.getAllClubs(connection).size());
            model.addAttribute(clubDAO.getAllClubs(connection));
        }
        catch (SQLException ex){
            ex.printStackTrace();
        }
        finally {
            try{
                if (connection != null){
                    connection.close();
                }
            }
            catch (SQLException exception){
                exception.printStackTrace();
            }
        }
```

Рисунок 2. Фрагмент класса Handler.java

2) Затем из Spring – контроллера была убрана вся логика и подключена зависимость к классу Handler, который уже в свою очередь работает с Connector и DAO – классами.
Теперь контроллер является по своей сути сервисом, который принимает некоторые данные по HTTP от пользователя, вызывает через использованные зависимости некоторую логику (в Hanlder), работающую с данными внутри приложениями, а затем возвращающий в качестве результата команду на показ JSP – страницу с уже обработанными данными. (рис.3).

Осталось только по условии лабораторной работы разделить имеющийся MainController на 2 отдельных контроллера для каждой из модели данных, то есть ClubController (для сущности Club – рис. 3.1) и SportsmanController (для сущности Sportsman – рис 3.2.).

```java
import ru.vlsu.ispi.logic.abstractions.IHandler;

import javax.validation.Valid;
import java.sql.*;
import java.util.ArrayList;
import java.util.List;
import java.util.Properties;

@Controller

public class ClubController {
    @Autowired
    private IHandler _handler;

    @GetMapping("/hello/{id}")
    public String handle(@PathVariable Long id, @RequestParam(required = false, name = "playerId") Long playerId, Model mode
        _handler.HandleIndex(id, playerId, model);

        return "index";
    }

    @GetMapping("/club")
    public String clubForm(@RequestParam(required = false, name = "id") Long id, Model model){
        _handler.ClubFormOpen(id, model);

        return "clubs";
    }

    @PostMapping("/club")
    public String clubSubmit(@ModelAttribute Club club, Model model){
        _handler.ClubFormSubmit(club, model);

        return "club";
    }

    @GetMapping("/club/{id}")
    public String deleteClub(@PathVariable Long id, Model model){
        _handler.DeleteClub(id, model);

        return "club";
```

Рисунок 3.1 ClubController.java

```java
import javax.validation.Valid;

@Controller

public class SportsmanController {
    @Autowired
    private IHandler _handler;

    @GetMapping("/hello/{id}")
    public String handle(@PathVariable Long id, @RequestParam(required = false, name = "playerId") Long playerId, Model mo
        _handler.HandleIndex(id, playerId, model);

        return "index";
    }

    @GetMapping("/sportsman")
    public String sportsmanForm(Model model){
        _handler.SportsmanFormOpen(model);

        return "sportsmen";
    }

    @PostMapping("/sportsman")
    public String sportsmanSubmit(@Valid @ModelAttribute Sportsman sportsman, BindingResult result, Model model){
        if (result.hasErrors()){
            return "sportsmen";
        }
        else {
            _handler.SportsmanFormSubmit(sportsman, model);

            return "sportsman";
        }
    }
}
```

Рисунок 3.2. SportsmanController.java

3) И затем с помощью механизма Dependency Injection были объявлены зависимости на созданные классы логики (по аналогии как это делалось раньше для инъекции зависимостей используемых в контроллере DAO – классов).

Изменения были произведены в файле app-context.xml – то есть на уровне целого Spring MVC – приложения, а не только для одного контроллера. Опционально можно было разделить инъекции для каждого из контроллеров в соответствующих им файлах serlet-context.xml. Жизненный цикл для логики был объявлен – Singleton (рис.4)

```
<context:component-scan base-package="ru.vlsu.ispi"/>

<bean id="stub" class="ru.vlsu.ispi.DAO.DBImplementation" scope="singleton">
    <!-- collaborators and configuration for this bean go here -->
</bean>

<bean id="conn" class="ru.vlsu.ispi.logic.abstractions.IConnector" scope="singleton">
</bean>

<bean id="handle" class="ru.vlsu.ispi.logic.abstractions.IHandler" scope="singleton">
</bean>

<bean id="sportsmanDAO" class="ru.vlsu.ispi.DAO.SportsmanDAO">
    <constructor-arg ref="stub"/>
</bean>

<!-- more bean definitions go here -->
<bean id="clubDAO" class="ru.vlsu.ispi.DAO.ClubDAO">
    <property name="impl">
        <ref bean="stub"/>
    </property>
</bean>

<!-- more bean definitions go here -->
<bean id="Connector" class="ru.vlsu.ispi.logic.Connector">
    <property name="impl">
        <ref bean="conn"/>
    </property>
</bean>

<!-- more bean definitions go here -->
<bean id="Handler" class="ru.vlsu.ispi.logic.Handler">
    <property name="impl">
        <ref bean="handle"/>
    </property>
</bean>
```

Рисунок 4. App-context.xml

По итогу, получаем работающее Spring – приложение (рис.5.1-5.3 – добавление нового спортсмена в БД):

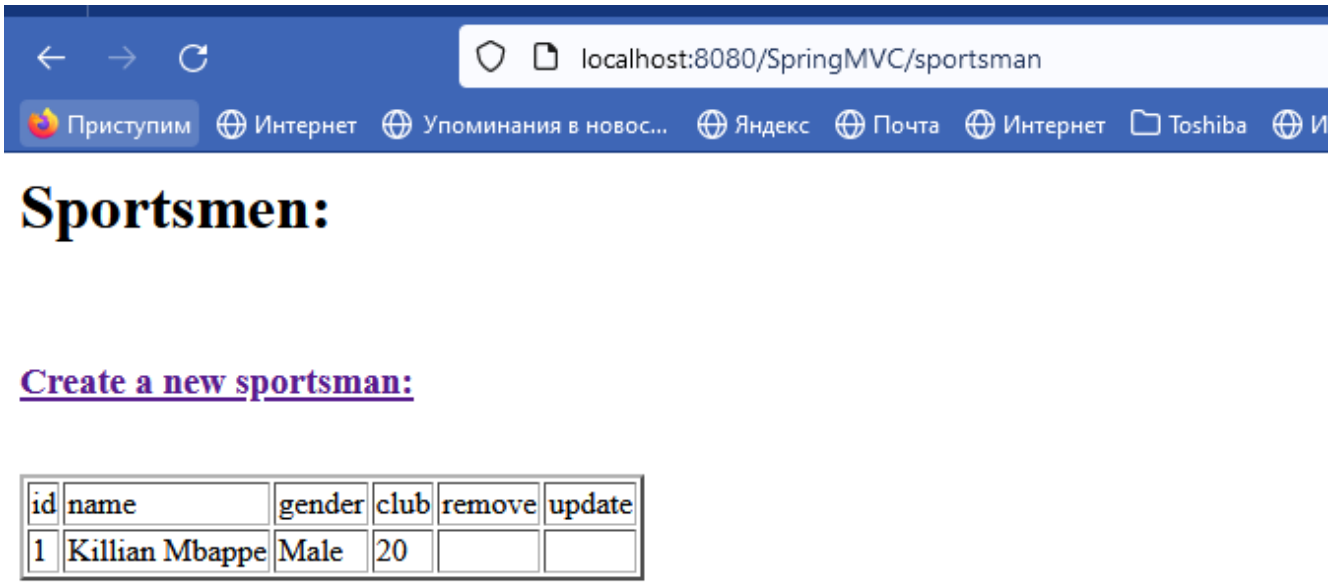


Рисунок 5.1.

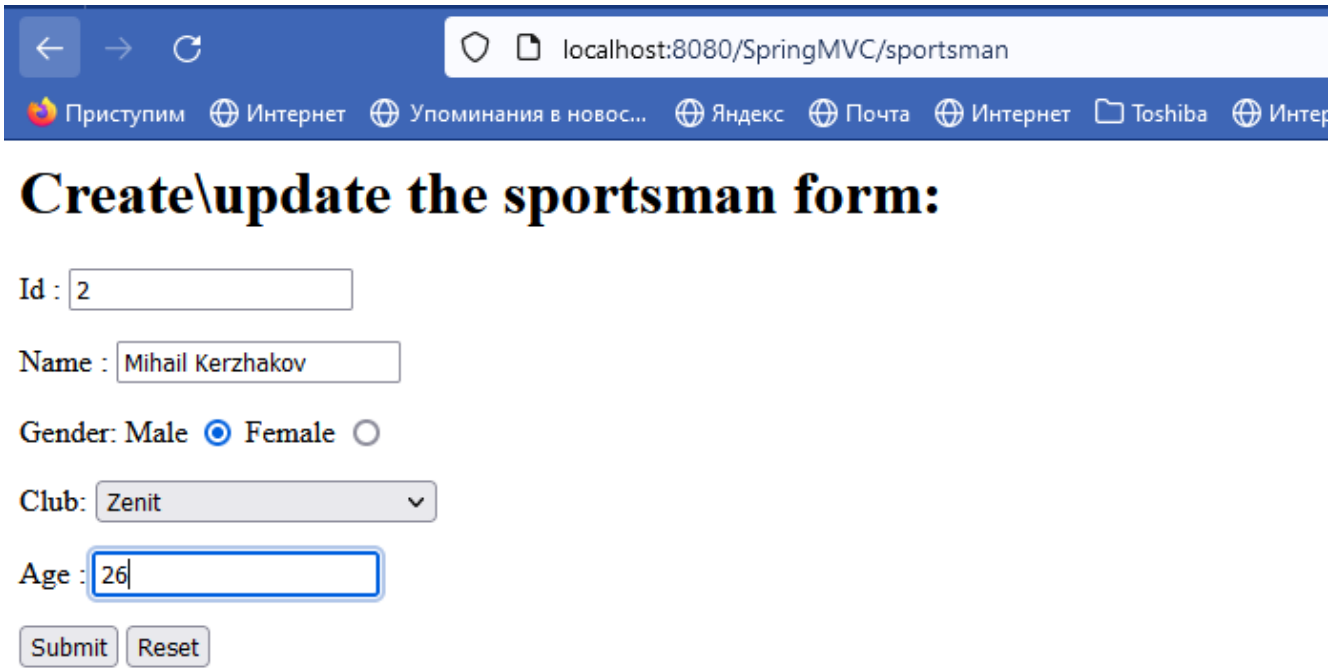
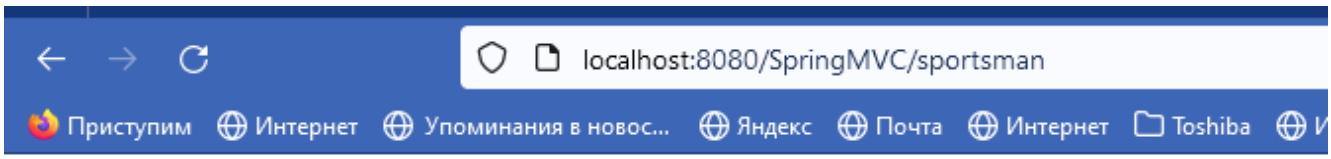


Рисунок 5.2.



Sportsmen:

Create a new sportsman:

id	name	gender	club	remove	update
1	Killian Mbappe	Male	20		
2	Mihail Kerzhakov	Male	13		

Рисунок 5.3.

Вывод

В результате выполнения работы, я познакомился с принципами построения веб-приложений с использованием шаблона MVC и на базе принципов REST, получить практические навыки реализации MVC - приложения на платформе Spring MVC.