**Spring Data**

**Теория**

Теория может быть найдена в файлах

* JPA маппинг сущностей – mapping.html
* Возможные настройки hibernate могут быть найдены в файле hibernate.docx
* Вся информация по SpringFramework (такая как конфигурация spring-jpa и прочее) может быть найдена по адресу <https://docs.spring.io/spring/docs/current/spring-framework-reference/htmlsingle/>
* О том, как работать с JpaRepository рассказано в файле “Spring Data JPA.html”

**Практическая часть**

# Создать базу данных в соответствии с предметной областью

В данном случае, university

# Создание проекта

Создать новый Maven проект с архитектурой **maven-archetype-quickstart,** добавить к проекту папку **resources** (см. Лабораторная работа 2)

# Сконфигурировать pom.xml для работы с Spring Data

<project xmlns=*"http://maven.apache.org/POM/4.0.0"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>

<modelVersion>4.0.0</modelVersion>

<groupId>by.vstu.*<имя\_разработчика>*</groupId>

<artifactId>lab4</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>lab4</name>

<url>http://maven.apache.org</url>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<java.version>1.8</java.version>

<spring.framework.version>4.3.10.RELEASE</spring.framework.version>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>${spring.framework.version}</version>

</dependency>

<dependency>

<groupId>org.springframework.data</groupId>

<artifactId>spring-data-jpa</artifactId>

<version>1.11.7.RELEASE</version>

</dependency>

<dependency>

<groupId>commons-dbcp</groupId>

<artifactId>commons-dbcp</artifactId>

<version>1.4</version>

</dependency>

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-core</artifactId>

<version>5.2.11.Final</version>

</dependency>

<dependency>

<groupId>org.postgresql</groupId>

<artifactId>postgresql</artifactId>

<version>9.4.1212</version>

</dependency>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>5.1.6</version>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<artifactId>maven-compiler-plugin</artifactId>

<configuration>

<source>${java.version}</source>

<target>${java.version}</target>

</configuration>

</plugin>

</plugins>

</build>

</project>

# Создать сущности с маппингом для Hibernate

## Создать пакет для сущностей

by.vstu.isap.<имя\_разработчика>.lab4.entity

## Создать сущности

**Примечание:** все аннотации, использующиеся для маппинга находятся в пакете **javax.persistence**. При использовании других аннотаций будут возникать ошибки при попытке инициализации EntityManagera (если использовать @Id из пакета spring No identifier specified for entity:)

Абстрактный суперкласс для всех сущностей должен содержать id и аннотацию @MappedSuperclass.

@MappedSuperclass

**public** **abstract** **class** AbstractEntity {

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

**protected** Long id;

**public** Long getId() {

**return** id;

}

**public** **void** setId(Long id) {

**this**.id = id;

}

}

Далее создать классы-сущности:

Группа:

@Entity

@Table(name = "groups", schema = "public")

**public** **class** Group **extends** AbstractEntity {

@Column(unique = **true**)

**private** String name;

@OneToMany(fetch = FetchType.***EAGER***, mappedBy = "group")

**private** List<Student> students;

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** List<Student> getStudents() {

**return** students;

}

**public** **void** setStudents(List<Student> students) {

**this**.students = students;

}

}

Студент

@Entity

@Table(name = "students", schema = "public")

**public** **class** Student **extends** AbstractEntity {

**private** String name;

**private** String surname;

@Column(name = "phone\_number")

**private** String phoneNumber;

@ManyToOne(fetch = FetchType.***LAZY***)

@JoinColumn(name = "group\_id", nullable = **false**)

**private** Group group;

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** String getSurname() {

**return** surname;

}

**public** **void** setSurname(String surname) {

**this**.surname = surname;

}

**public** String getPhoneNumber() {

**return** phoneNumber;

}

**public** **void** setPhoneNumber(String phoneNumber) {

**this**.phoneNumber = phoneNumber;

}

**public** Group getGroup() {

**return** group;

}

**public** **void** setGroup(Group group) {

**this**.group = group;

}

}

**Примечание:** Поля, название которых в базе данных идентичны названию полей в классе могут не помечаться аннотацией @Column, однако, если в БД название поля отличается, то оно указывается в качестве параметра **name** аннотации @Column

# Создать respositories

## Создать пакет для объектов доступа к базе данных

by.vstu.isap.<имя\_разработчика>.lab4.repository

## Создать repository для каждой сущности:

Студент:

**public** **interface** StudentRepository **extends** JpaRepository<Student, Long> {

List<Student> findByGroupId(Long id);

List<Student> findBySurname(String name);

}

Группа:

**public** **interface** GroupRepository **extends** JpaRepository<Group, Long> {

Group findByName(String name);

}

# Сконфигурировать JPA

## Создать в resources файл с настройками базы данных db.properties:

driver=com.mysql.jdbc.Driver

url=jdbc:mysql://localhost:3306/university

user=root

pass=root

## Создать в resouces файл с настройками Hibernate hibernate.properties

hibernate.dialect=org.hibernate.dialect.MySQL55Dialect

hibernate.show\_sql=true

hibernate.hbm2ddl.auto=update

## Создать файл с настройками JPA spring-jpa.xml:

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns:context=*"http://www.springframework.org/schema/context"*

xmlns:repository=*"http://www.springframework.org/schema/data/repository"*

xmlns:jpa=*"http://www.springframework.org/schema/data/jpa"*

xmlns:tx=*"http://www.springframework.org/schema/tx"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd*

*http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.3.xsd*

*http://www.springframework.org/schema/data/jpa http://www.springframework.org/schema/data/jpa/spring-jpa-1.8.xsd*

*http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.3.xsd*

*http://www.springframework.org/schema/data/repository http://www.springframework.org/schema/data/repository/spring-repository-1.11.xsd"*>

<context:property-placeholder location=*"classpath:db.properties"* />

<bean name=*"dataSource"* class=*"org.apache.commons.dbcp.BasicDataSource"*>

<property name=*"driverClassName"* value=*"${driver}"* />

<property name=*"url"* value=*"${url}"* />

<property name=*"username"* value=*"${user}"* />

<property name=*"password"* value=*"${pass}"* />

</bean>

<bean id=*"entityManagerFactory"*

class=*"org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean"*>

<property name=*"dataSource"* ref=*"dataSource"* />

<property name=*"packagesToScan"* value=*"by.vstu.isap.<имя\_разработчика>.lab4.entity"* />

<property name=*"jpaVendorAdapter"*>

<bean class=*"org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter"* />

</property>

</bean>

<bean id=*"transactionManager"* class=*"org.springframework.orm.jpa.JpaTransactionManager"* />

<tx:annotation-driven transaction-manager=*"transactionManager"* />

<jpa:repositories base-package=*"by.vstu.isap.zamok.<имя\_разработчика>.repository"* />

</beans>

# Создать Java Config для Spring приложения

## Создать пакет для классов конфигурации

by.vstu.isap.<имя\_разработчика>.lab4.config

## Создать класс конфигурации

@Configuration

@ImportResource("classpath:spring-jpa.xml")

@ComponentScan("by.vstu.isap.*<имя\_разработчика>*.lab4")

**public** **class** ApplicationConfig {

}

# Создать сервисы и их реализации в соответствии с сущностями

Смотри Лабораторная работа 2

## Создать пакет с интерфейсами сервисов:

by.vstu.isap.<имя\_разработчика>.lab4.service

## И сами сервисы

Абстрактный сервис:

**public** **interface** Service<T **extends** AbstractEntity> {

T read(Long id);

List<T> read();

**void** save(T entity);

**void** delete(Long id);

}

Студент:

**public** **interface** StudentService **extends** Service<Student> {

List<Student> readByGroupId(Long groupId);

List<Student> readBySurname(String surname);

}

Группа:

**public** **interface** GroupService **extends** Service<Group> {

Group readByName(String name);

}

## В данном пакете создается пакет с реализациями:

by.vstu.isap.<имя\_разработчика>.lab4.service.impl

## И сами реализации:

Студент

@Service

**public** **class** StudentServiceImpl **implements** StudentService {

@Autowired

**private** StudentRepository repository;

@Override

**public** Student read(Long id) {

**return** repository.findOne(id);

}

@Override

**public** List<Student> read() {

**return** repository.findAll();

}

@Override

**public** **void** save(Student entity) {

repository.save(entity);

}

@Override

**public** **void** delete(Long id) {

repository.delete(id);

}

@Override

**public** List<Student> readByGroupId(Long groupId) {

**return** repository.findByGroupId(groupId);

}

@Override

**public** List<Student> readBySurname(String surname) {

**return** repository.findBySurname(surname);

}

}

Группа:

@Service

**public** **class** GroupServiceImpl **implements** GroupService {

@Autowired

**private** GroupRepository repository;

@Override

**public** Group read(Long id) {

**return** repository.findOne(id);

}

@Override

**public** List<Group> read() {

**return** repository.findAll();

}

@Override

**public** **void** save(Group entity) {

repository.save(entity);

}

@Override

**public** **void** delete(Long id) {

repository.delete(id);

}

@Override

**public** Group readByName(String name) {

**return** repository.findByName(name);

}

}

# Создать главный класс программы

## Создать пакет для главного класса

by.vstu.isap.<имя\_разработчика>.lab4.runner

## Создать класс с главным методом main

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

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

ConfigurableApplicationContext ctx = **new** AnnotationConfigApplicationContext(ApplicationConfig.**class**);

GroupService groupService = ctx.getBean(GroupService.**class**);

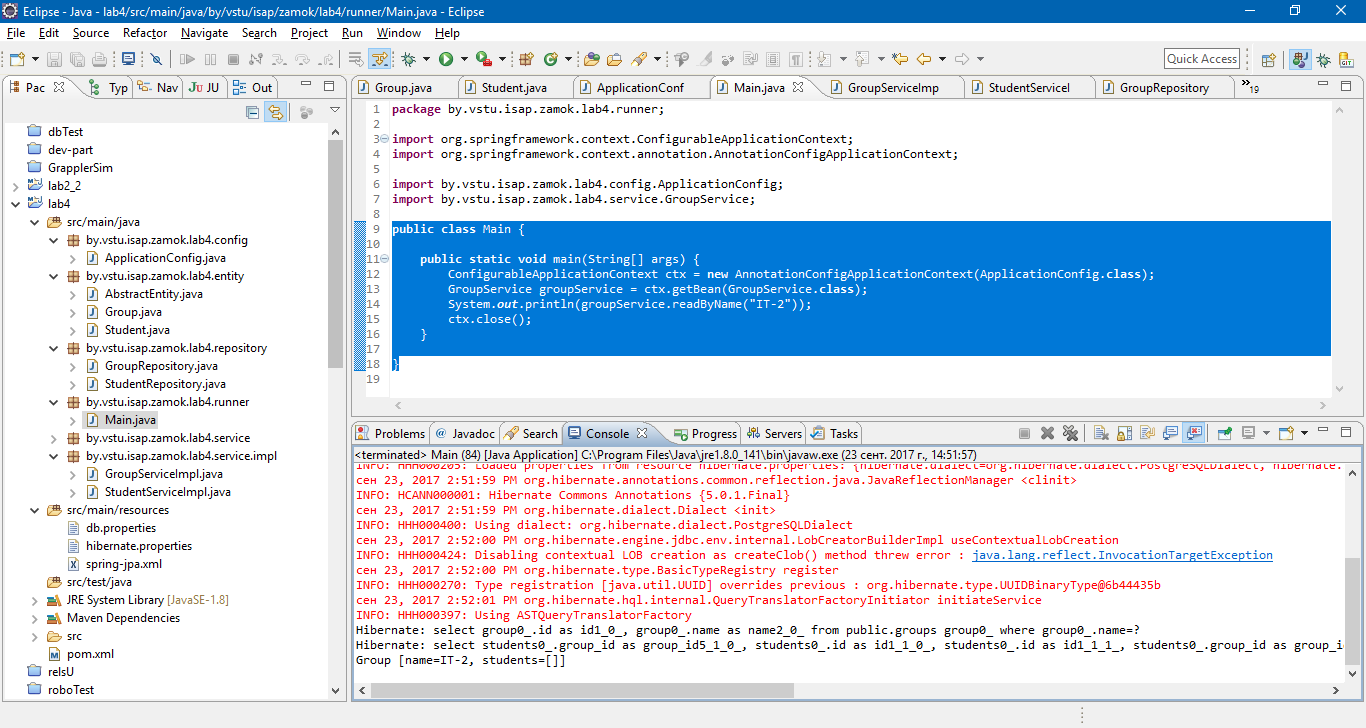
System.***out***.println(groupService.readByName("IT-2"));

ctx.close();

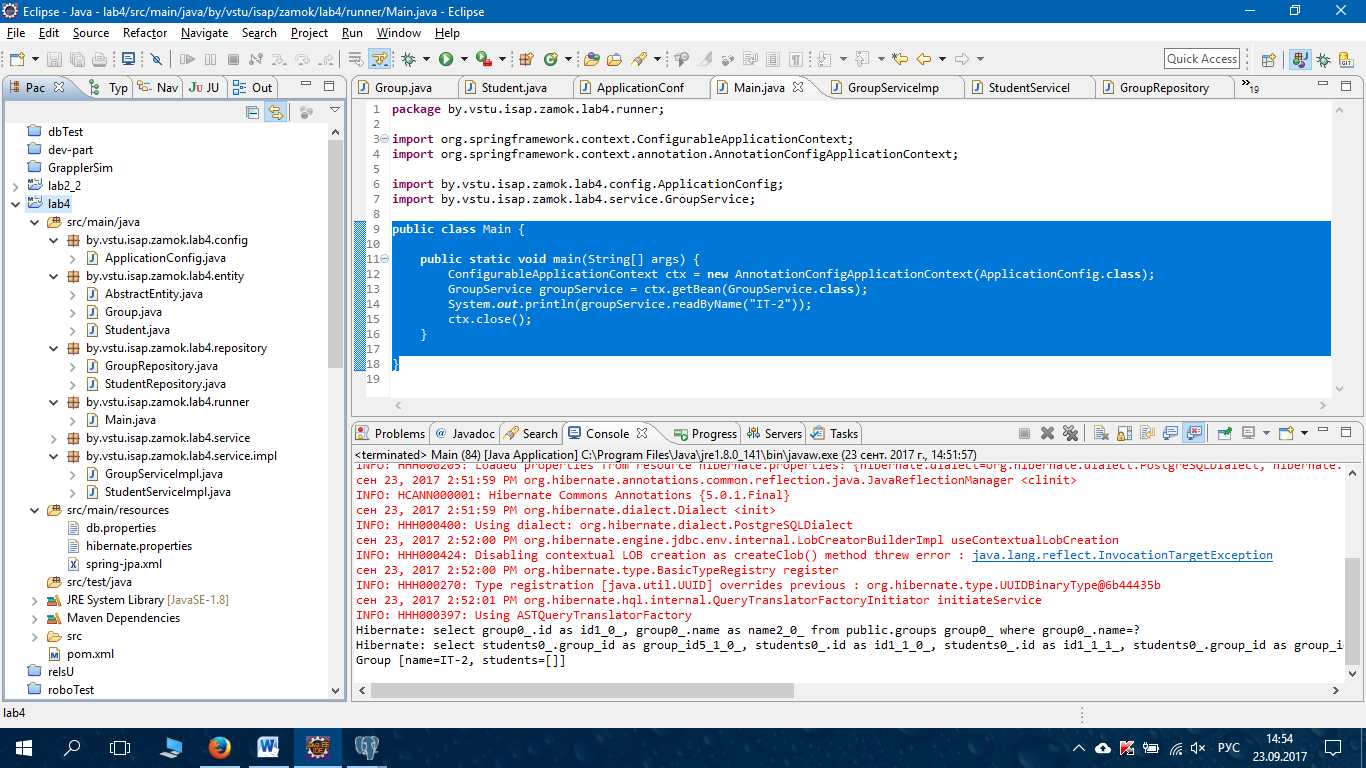
}

}

## Добавить данные в базу и запустить:



# В итоге архитектура проекта выглядит следующим образом:



# Индивидуальное задание:

Добавить к проекту вертикаль (entity, repository, service):

Варианты 1,4,7,10,13: Факультет, в котором имеется название, телефон, список групп

Варианты 2,5,8,11,14: Специальность, которая содержит список групп, название и код специальности.

Варианты 3,6,9,12,15: Куратор, имеющий группу, должность, имя, фамилию и телефон.

Для сдачи лабораторной требуется продемонстрировать работу методов CRUD в сервисах (create, read, update, delete). В данном случае, методы create и update объединены в save, а методов read может быть несколько.