**Быстрое spring приложение**



Рис. 1 Стартовое окно

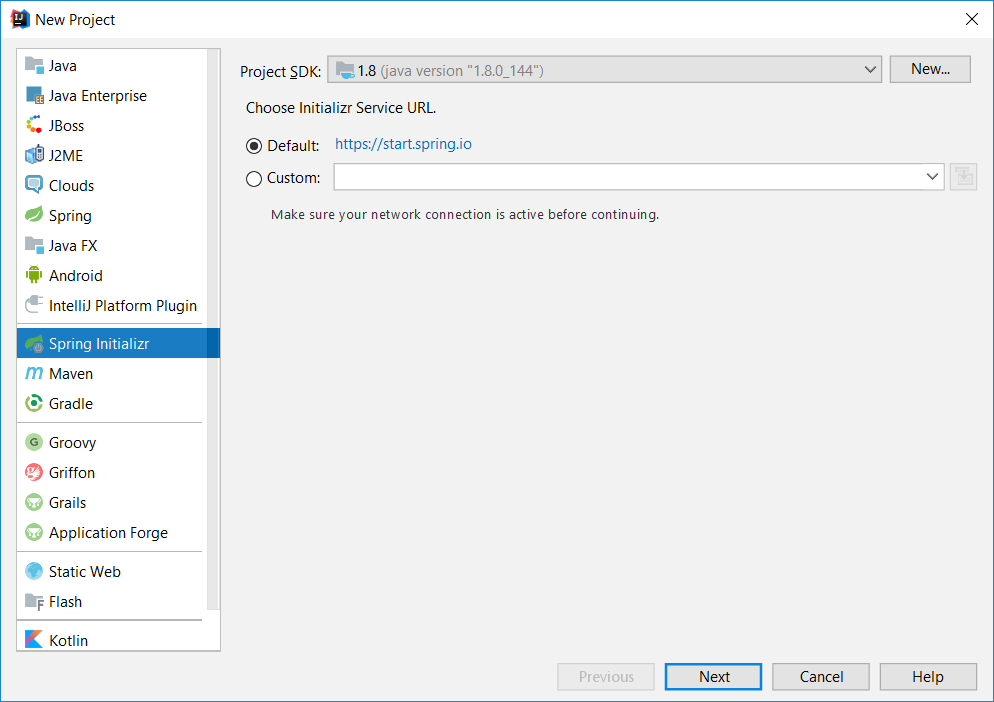


Рис. 2 Окно создания проекта шаг-1

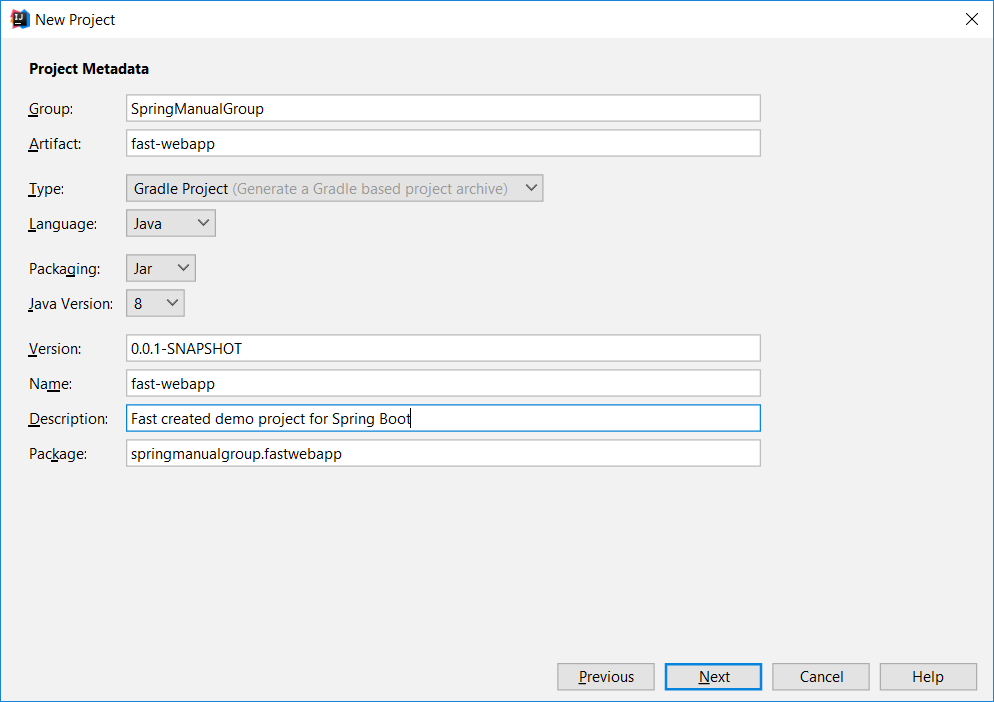


Рис. 3 Окно создания проекта шаг-2

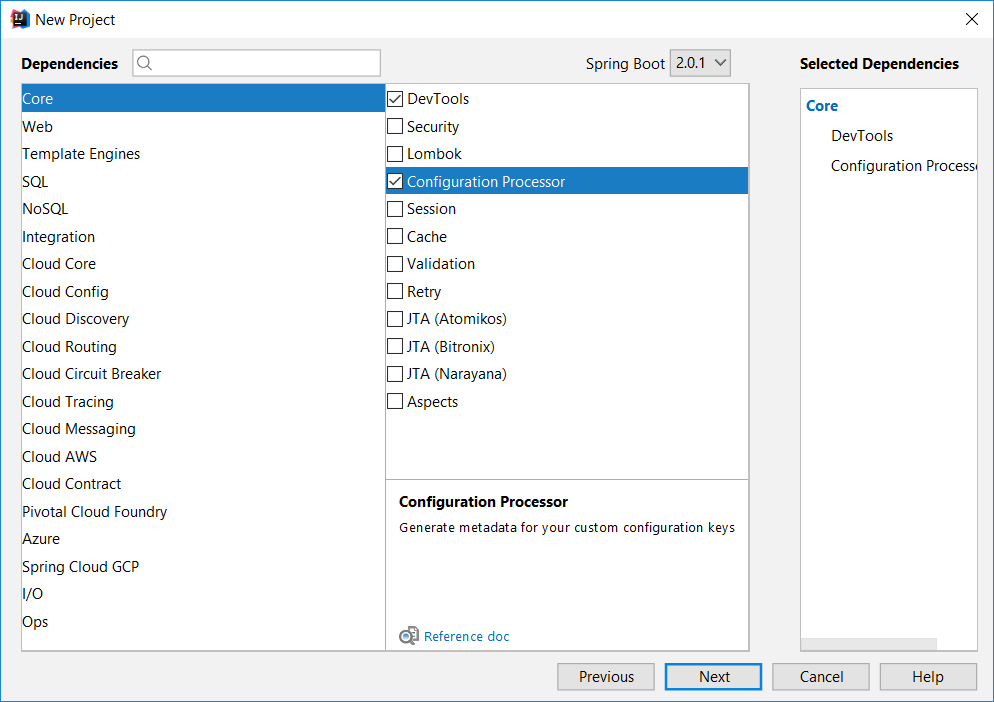


Рис. 4 Окно создания проекта –выбор зависимостей шаг-1

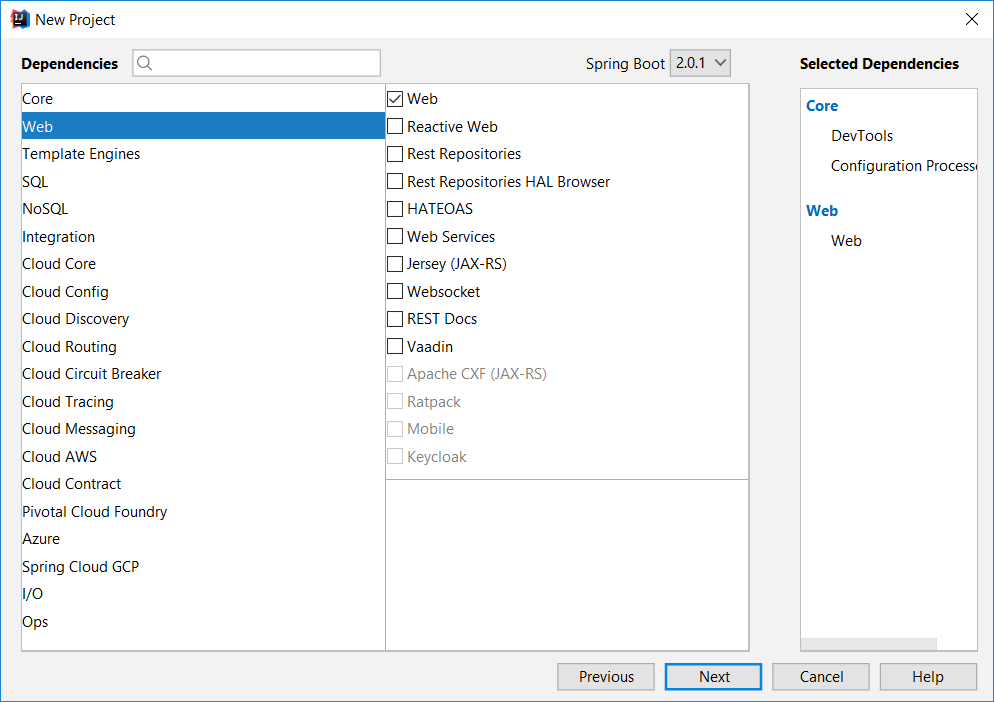
4 

Рис. 5 Окно создания проекта –выбор зависимостей шаг-2

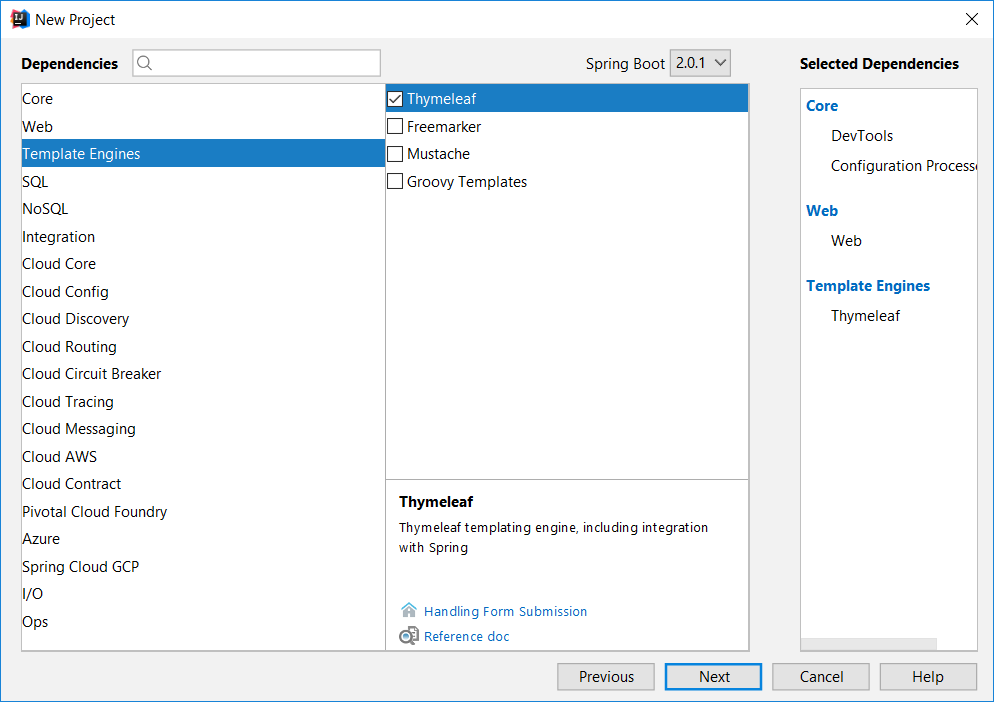


Рис. 6 Окно создания проекта –выбор зависимостей шаг-3

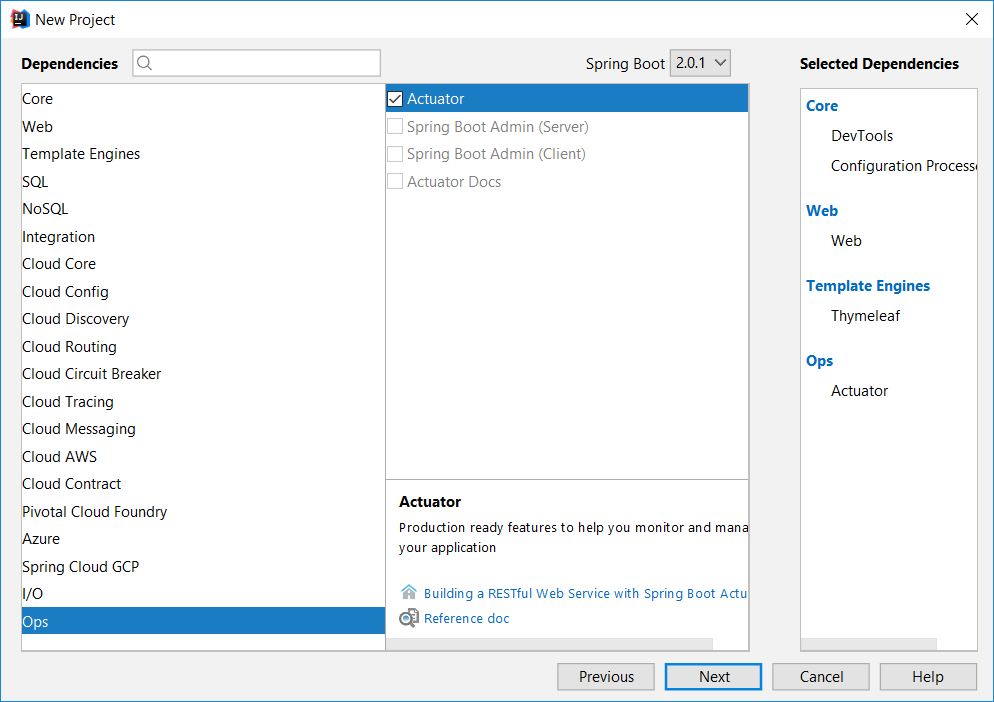


Рис. 7 Окно создания проекта –выбор зависимостей шаг-5

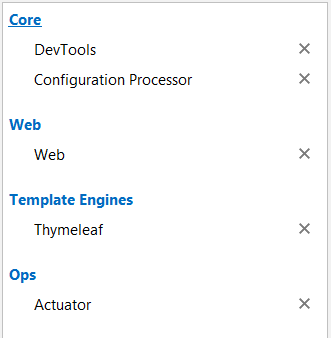


Рис. 8 Список выбранных зависимостей

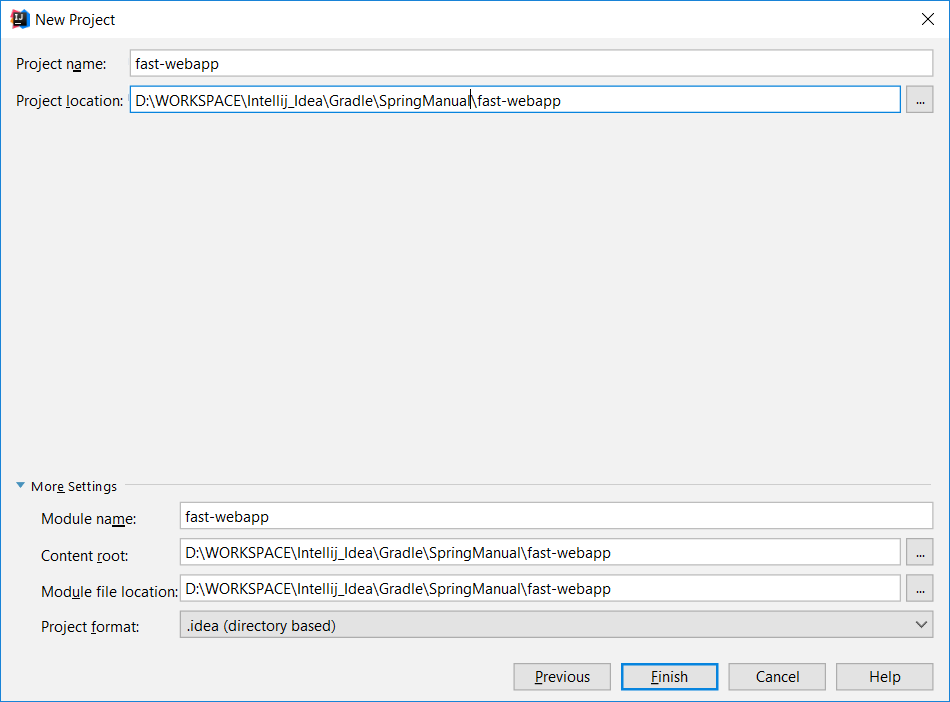


Рис. 9 Окно создания проекта шаг-3

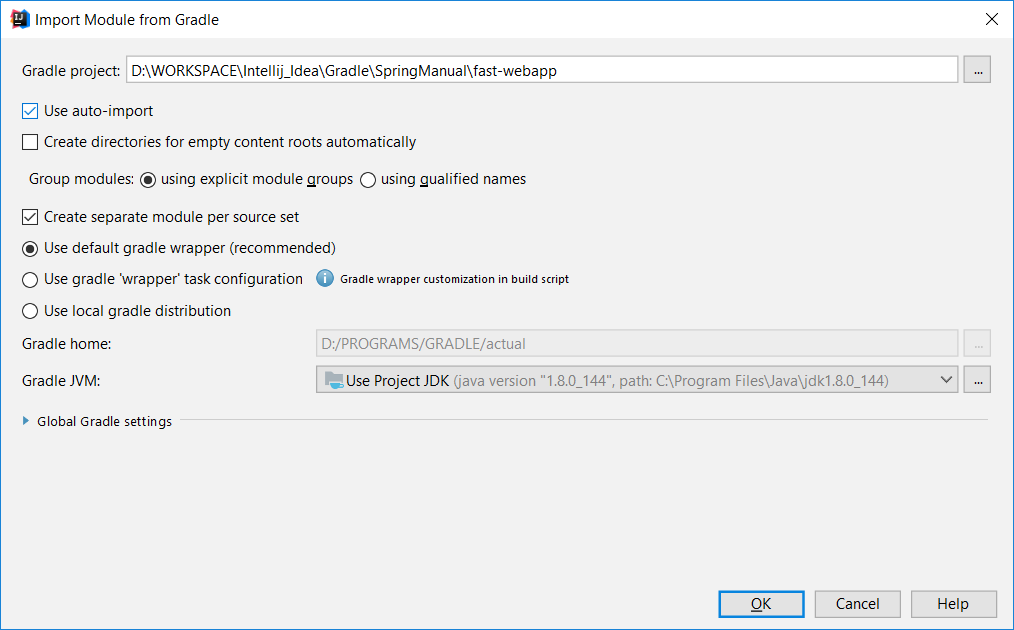


Рис. 10 Окно импорта модуля проекта

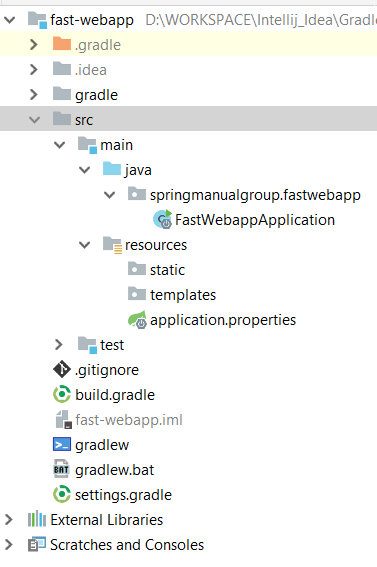


Рис. 11 Вид стартового проекта

Приложение уже можно запустить (*Рис. 12*), пока что полезной работы оно не выполняет.

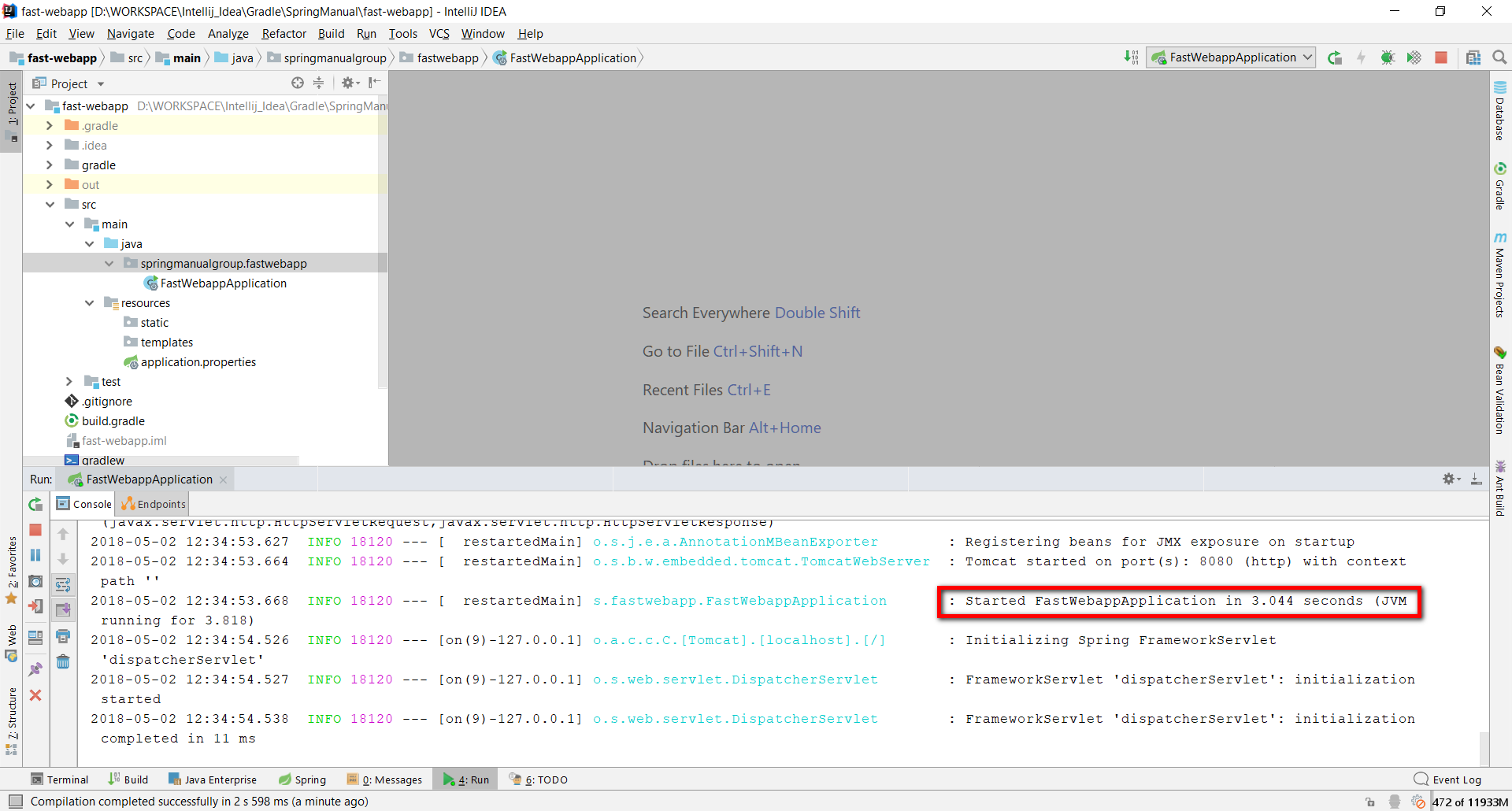


Рис. 12 Сообщение об успешном запуске

Расширим функционал, добавив контроллер Test controller. Создадим пакет controller в пакете springmanualgroup.fastwebapp, нажмем ПКМ по директории fastwebapp выберем new -> Package

.controller

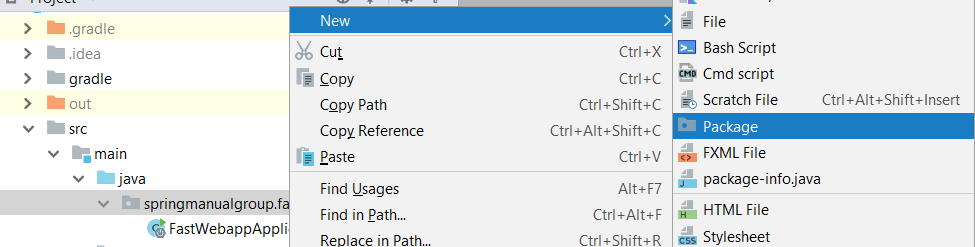


Рис. 13 Создание пакета шаг-1

шаг

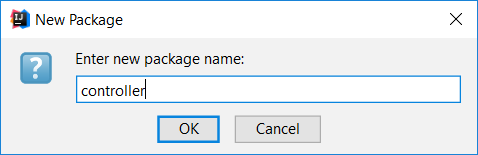


Рис. 14 Создание пакета шаг-2

Создадим класс TestController выполнив аналогичную комбинацию действий (пкм по директории controller –> new -> Java Class) (*Рис. 13* *Рис. 14*)

|  |
| --- |
| **package** springmanualgroup.fastwebapp.controller;   **import** org.springframework.stereotype.Controller;  **import** org.springframework.web.bind.annotation.GetMapping;  **import** org.springframework.web.bind.annotation.RequestMapping;  @Controller @RequestMapping(**"/"**) **public class** TestController {   @GetMapping  **public** String get() {  **return "testView"**;  } } |

В директории resources/templates создадим представление с таким именем как мы возвращаем из нашего контроллера (testView.html)

|  |
| --- |
| <!DOCTYPE **html**> <**html lang="en"**> <**head**>  <**meta charset="UTF-8"**>  <**title**>Title</**title**> </**head**> <**body**> <**h1**>Hello Spring</**h1**> </**body**> </**html**> |

Запустим приложение и по адресу <http://127.0.0.1:8080/> увидим нашу страницу. *Рис. 15*

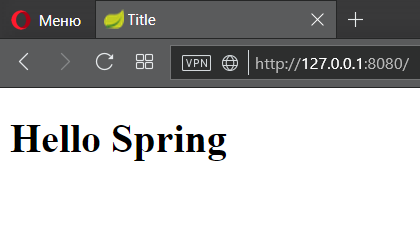


Рис. 15 Тестовая страница

Добавим обратную связь, а именно форму передачи данных и обработку этих данных.

Добавим класс модели передачи данных между страницами в пакет springmanualgroup.fastwebapp.model

|  |
| --- |
| **package** springmanualgroup.fastwebapp.model;  **public class** TestModel {  **private** String **name**;   **public** String getName() {  **return name**;  }   **public void** setName(String name) {  **this**.**name** = name;  } } |

Для тенерации get-set методов следует использовать комбинацию alt+insert ->Getter and Setter -> выбрать поля для генерации (*Рис. 16* *Рис. 17*)

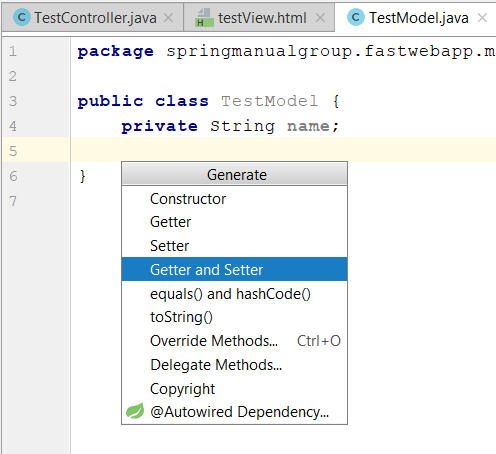


Рис. 16 Генерация get-set методов шаг-1

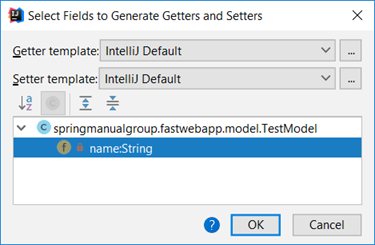


Рис. 17 Генерация get-set методов шаг-2

Изменим представление

|  |  |
| --- | --- |
| Было | Стало |
| <!DOCTYPE **html**> <**html lang="en"**> <**head**>  <**meta charset="UTF-8"**>  <**title**>Title</**title**> </**head**> <**body**> <**h1**>Hello Spring</**h1**> </**body**> </**html**> | <!DOCTYPE **html**> <**html xmlns:th="http://www.thymeleaf.org"**> <**head**>  <**meta charset="UTF-8"**>  <**title**>Title</**title**> </**head**> <**body**> <**h1 th:text="'Hello ' +((${testModel!=null && testModel.name!=null && !#strings.isEmpty(testModel.name)})?(${testModel.name}):'Spring')"**></**h1**> <**form th:action="@{/}" method="post" th:object="${testModel}"**>  <**label for="nameInput"**>Enter You name</**label**>  <**input id="nameInput" th:field="\*{name}" type="text"**>  <**button type="submit"**>Confirm</**button**> </**form**> </**body**> </**html**> |

Изменим контроллер

|  |  |
| --- | --- |
| Было | Стало |
| @Controller @RequestMapping(**"/"**) **public class** TestController {   @GetMapping  **public** String get() {  **return "testView"**;  } } | **package** springmanualgroup.fastwebapp.controller;  **import** org.springframework.stereotype.Controller; **import** org.springframework.web.bind.annotation.GetMapping; **import** org.springframework.web.bind.annotation.ModelAttribute; **import** org.springframework.web.bind.annotation.PostMapping; **import** org.springframework.web.bind.annotation.RequestMapping; **import** org.springframework.web.servlet.ModelAndView; **import** springmanualgroup.fastwebapp.model.TestModel;  @Controller @RequestMapping(**"/"**) **public class** TestController {   @GetMapping  **public** ModelAndView get() {  **return new** ModelAndView(**"testView"**, **"testModel"**, **new** TestModel());  }   @PostMapping  **public** ModelAndView post(@ModelAttribute(**"testModel"**) TestModel testModel) {  **if** (**null** == testModel.getName() || testModel.getName().trim().length() == 0) {  testModel.setName(**null**);  }  **return new** ModelAndView(**"testView"**, **"testModel"**, testModel);  } } |

Запустим приложение, и по адресу <http://127.0.0.1:8080/> увидим нашу форму, попробуем провзаимодействовать с ней. *Рис. 18* *Рис. 19*

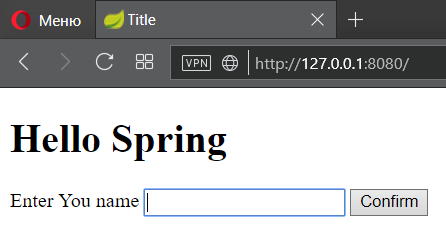


Рис. 18 Взаимодействие с формой шаг-1



Рис. 19 Взаимодействие с формой шаг-2

**Модульное spring приложение**

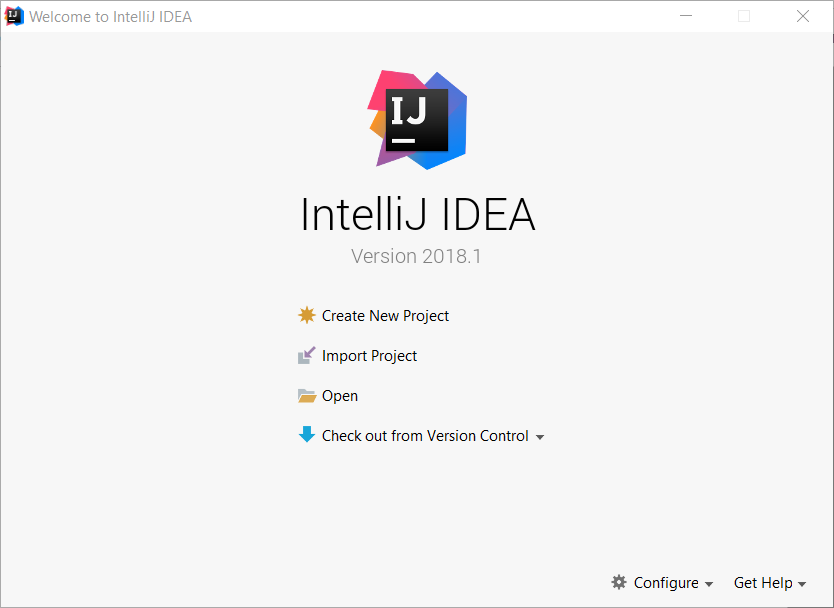


Рис. 20 Стартовое окно

В стартовом окне *Рис. 20* выбираем создание нового проекта (Create new Project)

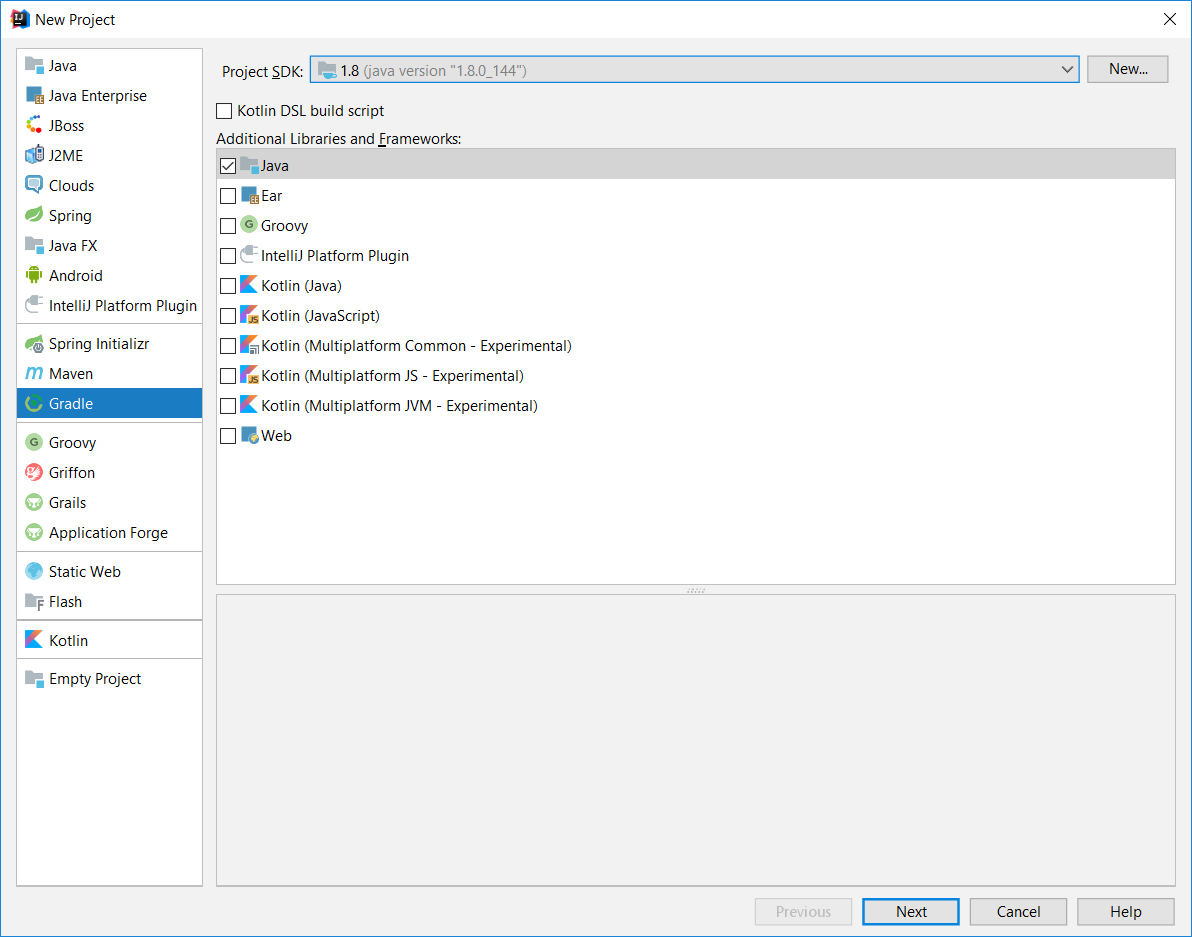


Рис. 21 Окно настройки проекта шаг-1

Далее выбираем систему сборки Gradle *Рис. 21* также выбираем версию JDK и убеждаемся что отмечены библиотеки java в области Additional Libraries and Frameworks.

Переходим далее.

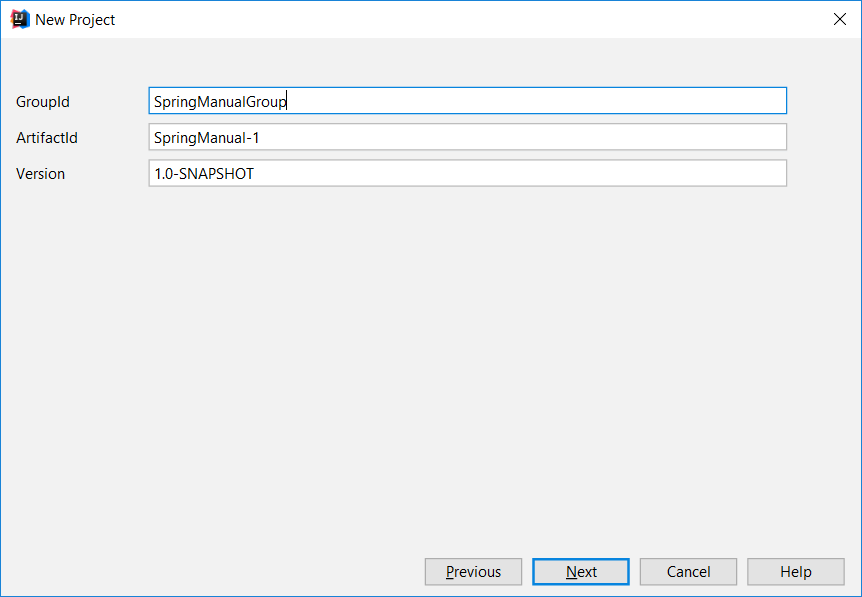


Рис. 22 Окно настройки проекта шаг-2

Заполняем поля GroupId и ArtifactId *Рис. 22* Переходим далее.

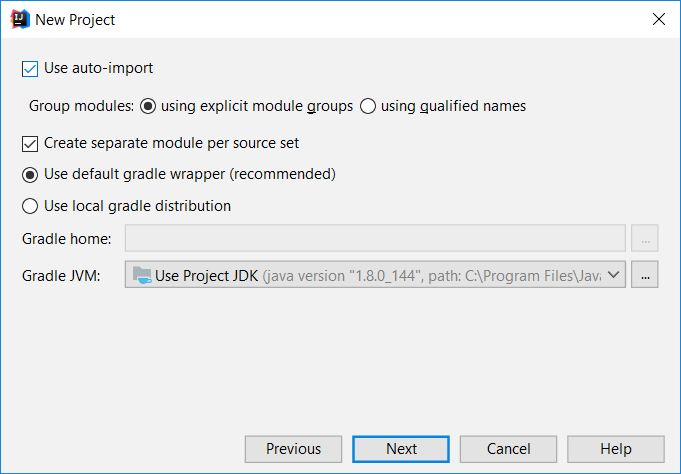


Рис. 23 Окно настройки проекта шаг-2

Устанавливаем опцию Use auto-import *Рис. 23*, для включения автоматического импорта зависимостей при изменении списка зависимостей gradle.

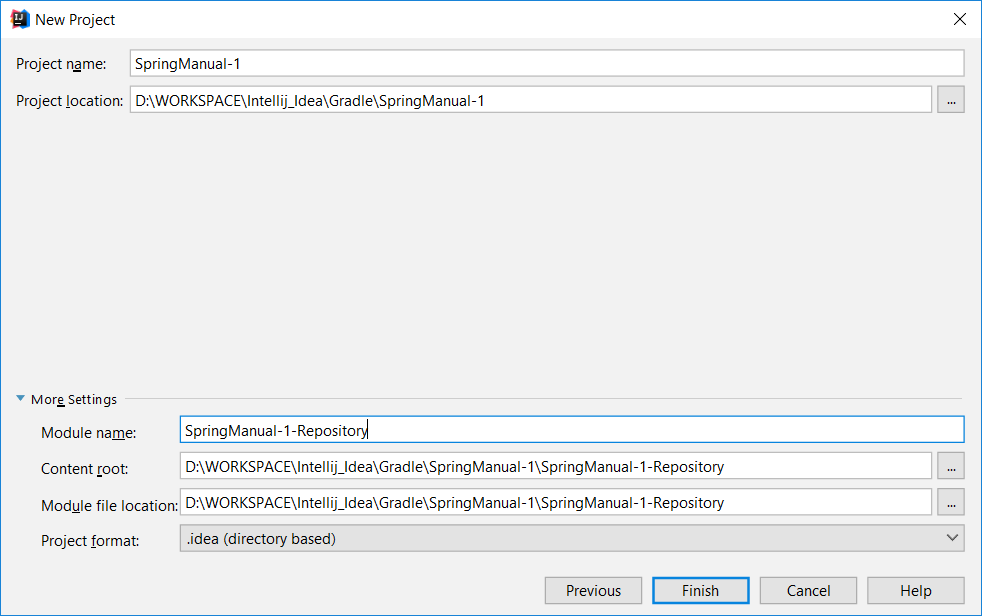


Рис. 24 Окно настройки проекта шаг-3

Вводим название проекта (Project name), выбираем расположение на диске директории с проектом (Project location), также, т.к. приложение будет из нескольких модулей, указываем имя первого модуля в скрываемой области настроек проекта (More Settings).

После нажатия кнопки Finish откроется главное окно IDE. Следует подождать пока произойдет обновление индексов проекта (прогресс отображается в нижней правой части окна ) *Рис. 25* *Рис. 26*

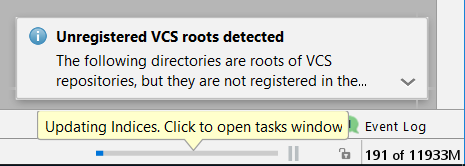


Рис. 25 Прогресс обновления индексов-1

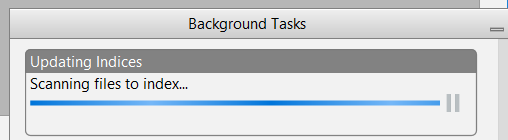
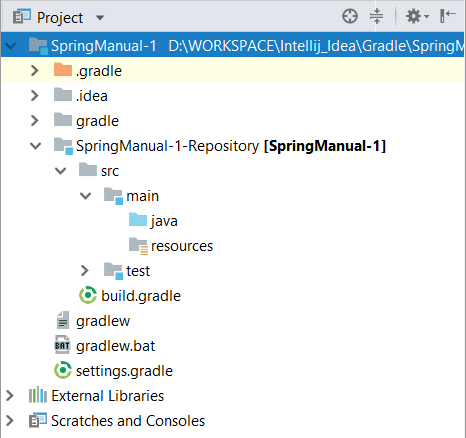


Рис. 26 Прогресс обновления индексов-2

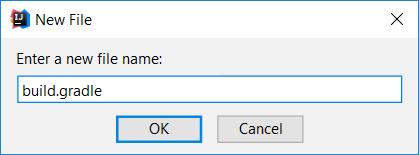


*Рис. 27 Структура нового проекта*

После обновления индексов можно приступать к работе.

В структуре проекта (Рис. 27) видим несколько файлов конфигурации Gradle (settings.gradle, build.gradle). Каждый модуль имеет свой файл настройки зависимостей - build.gradle, файл settings.gradle необходим для настройки построения модулей приложения.

Но так как у нас будет несколько модулей в которых есть одинаковые части конфигурации, создадим еще один файл build.gradle в корне проекта, в который будем писать общее для всех модулей.

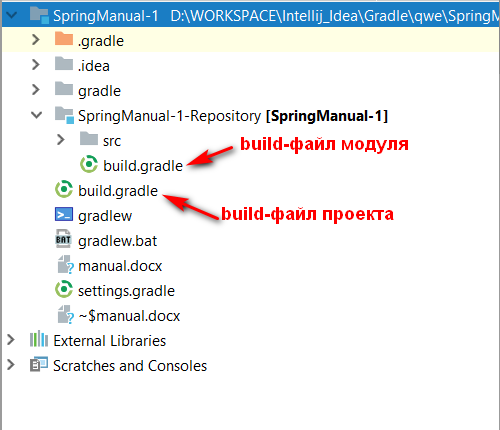


*Рис. 28 Создание нового файла*

Добавим в корень проекта новый файл build.gradle (ПКМ по корневому каталогу проекта -> New *Рис. 28*) для вынесения общего для всех модулей контента файлов build.gradle.

Содержимое корневого build.gradle:

|  |
| --- |
| allprojects {  group **'SpringManualDeveloperGroup'** version **'1.0-SNAPSHOT'** } subprojects {  buildscript {  ext {  springBootVersion = **'2.0.0.M4'** }  repositories {  mavenCentral()  maven { setUrl(**"https://repo.spring.io/snapshot"**) }  maven { setUrl(**"https://repo.spring.io/milestone"**) }  }  dependencies {  classpath(**"org.springframework.boot:spring-boot-gradle-plugin:**${springBootVersion}**"**)  }  }  apply **plugin**: **'java'** apply **plugin**: **'idea'** sourceCompatibility = 1.8  targetCompatibility = 1.8  repositories {  mavenCentral()  maven { setUrl(**"https://repo.spring.io/snapshot"**) }  maven { setUrl(**"https://repo.spring.io/milestone"**) }  }  ext {  hibernateEntitymanagerVersion = **'5.2.10.Final'** mysqlConnectorJavaVersion = **'5.1.34'** springSecurityTaglibsVersion = **'3.2.3.RELEASE'** thymeleafExtrasSpringsecurity4 = **'3.0.2.RELEASE'** tomcatEmbedJasperVersion = **'9.0.0.M21'** httpclientVersion = **'4.5.3'** junitVersion = **'4.12'** } } |



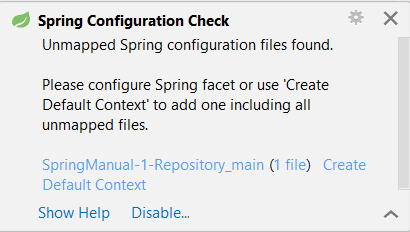
*Рис. 29 Файлы build.gradle в проекте*

В build.gradle модуля Repository добавим необходимые зависимости:

|  |
| --- |
| dependencies {  compile(**"org.springframework.boot:spring-boot-starter-data-jpa:**${springBootVersion}**"**)  compile(**"org.springframework.boot:spring-boot-configuration-processor:**${springBootVersion}**"**)  compile(**"org.hibernate:hibernate-entitymanager:**${hibernateEntitymanagerVersion}**"**)  compile(**"mysql:mysql-connector-java:**${mysqlConnectorJavaVersion}**"**)   runtime(**"org.springframework.boot:spring-boot-devtools:**${springBootVersion}**"**)  testCompile(**"org.springframework.boot:spring-boot-starter-test:**${springBootVersion}**"**) } |

Проверим файл settings.gradle в корне проекта, он должен содержать информацию о проекте и всех модулях проекта

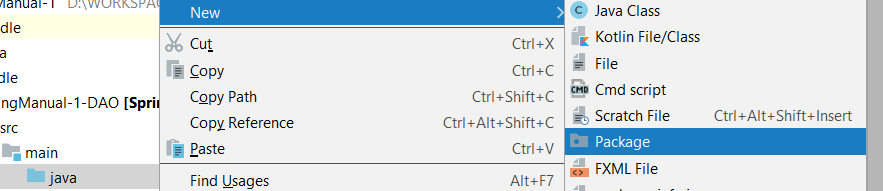
|  |
| --- |
| rootProject.name = **'SpringManual-1'** include **'SpringManual-1-Repository'** |



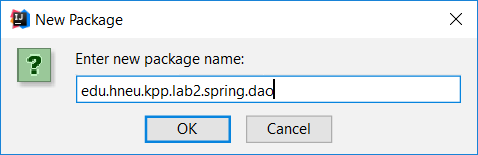
*Рис. 30 Создание нового контекста*

В случае появления в правом нижнем углу уведомления о проверке контекста необходимо нажать ссылку Create Default Context *Рис. 30*

Создадим новый пакет edu.hneu.kpp.lab2.spring.dao (*Рис. 31* - *Рис. 32*)

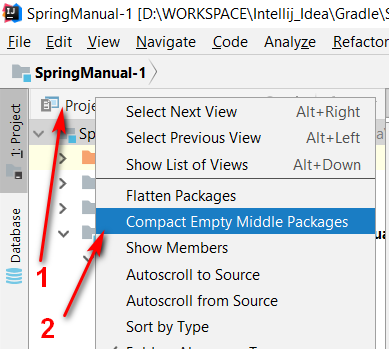


*Рис. 31 Создание нового пакета-1*



*Рис. 32 Создание нового пакета-2*

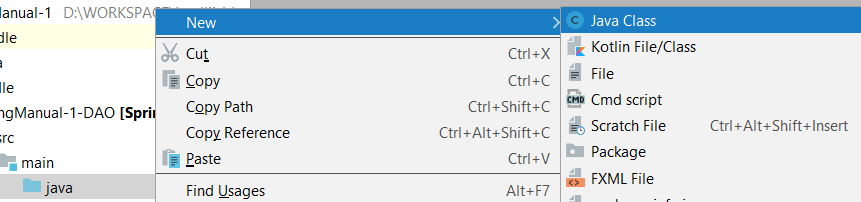
Для выключения автоматического сворачивания пустых вложенных директорий нажмем ПКM по типу отображения в области структуры проекта и снимаем галочку Compact Empty Middle Packages (). Позже для экономии полезного пространства рекомендуется вернуть эту опцию.



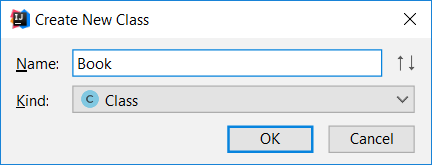
*Рис. 33* Выключение автоматического сворачивания пустых вложенных директорий

Создадим в пакете edu.hneu.kpp.lab2.spring.dao пакет entity и в нем - класс-сущность BookEntity – отображение строки таблицы БД.*Рис. 34* - *Рис. 35*

|  |
| --- |
| **package** edu.hneu.kpp.lab2.spring.dao.entity;  **import** javax.persistence.\*;  @Entity @Table(name = **"book"**) **public class** BookEntity {  @Id  @GeneratedValue(strategy = GenerationType.***IDENTITY***)  @Column(name = **"id"**, unique = **true**, nullable = **false**)  **private int id**;  @Column(name = **"title"**)  **private** String **title**;  @Column(name = **"author"**)  String **author**;   **public** BookEntity() {  }   **public** BookEntity(String title, String author) {  **this**.**title** = title;  **this**.**author** = author;  }   **public** Integer getId() {  **return id**;  }   **public void** setId(**int** id) {  **this**.**id** = id;  }   **public** String getTitle() {  **return title**;  }   **public void** setTitle(String title) {  **this**.**title** = title;  }   **public** String getAuthor() {  **return author**;  }   **public void** setAuthor(String author) {  **this**.**author** = author;  }   @Override  **public** String toString() {  **return "BookEntity [id="** + **id** + **", title="** + **title** + **", author="** + **author** + **"]"**;  } } |

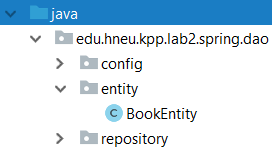


*Рис. 34 Создание нового класса-1*



*Рис. 35 Создание нового класса-2*

На уровне пакета entities создаем еще два пакета repository и config



*Рис. 36* Директория repository

Создаем репозиторий BookRepository в пакете repository

|  |
| --- |
| **package** edu.hneu.kpp.lab2.spring.dao.repository;  **import** edu.hneu.kpp.lab2.spring.dao.entity.BookEntity; **import** org.springframework.data.jpa.repository.JpaRepository;  **public interface** BookRepository **extends** JpaRepository<BookEntity, Integer> { } |

Создадим DataConfig- настройки подключения к БД в пакете config

|  |
| --- |
| **package** edu.hneu.kpp.lab2.spring.dao.config;  **import** org.hibernate.jpa.HibernatePersistenceProvider; **import** org.springframework.context.annotation.Bean; **import** org.springframework.context.annotation.Configuration; **import** org.springframework.context.annotation.PropertySource; **import** org.springframework.core.env.Environment; **import** org.springframework.data.jpa.repository.config.EnableJpaRepositories; **import** org.springframework.jdbc.datasource.DriverManagerDataSource; **import** org.springframework.orm.jpa.JpaTransactionManager; **import** org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean; **import** org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter; **import** org.springframework.transaction.annotation.EnableTransactionManagement;  **import** javax.annotation.Resource; **import** javax.sql.DataSource; **import** java.util.Properties;  @Configuration @EnableTransactionManagement @EnableJpaRepositories({**"edu.hneu.kpp.lab2.spring.dao.repository"**}) **public class** DataConfig {  **private static final** String ***entityPackage*** = **"edu.hneu.kpp.lab2.spring.dao.entity"**;  *//db prop* **private static final** String ***DB\_DRIVER*** = **"db.driver"**;  **private static final** String ***DB\_PASSWORD*** = **"db.password"**;  **private static final** String ***DB\_URL*** = **"db.url"**;  **private static final** String ***DB\_USERNAME*** = **"db.username"**;  *//hibernate prop* **private static final** String ***DB\_HIBERNATE\_DIALECT*** = **"hibernate.dialect"**;  **private static final** String ***DB\_HIBERNATE\_SHOW\_SQL*** = **"hibernate.show\_sql"**;  **private static final** String ***DB\_HIBERNATE\_FORMAT\_SQL*** = **"hibernate.format\_sql"**;  **private static final** String ***DB\_HIBERNATE\_HBM2DDL\_AUTO*** = **"hibernate.hbm2ddl.auto"**;  @Resource  **private** Environment **environment**;   @Bean  **public** DataSource dataSource() {  DriverManagerDataSource dataSource = **new** DriverManagerDataSource();  dataSource.setDriverClassName(**environment**.getRequiredProperty(***DB\_DRIVER***));  dataSource.setUrl(**environment**.getRequiredProperty(***DB\_URL***));  dataSource.setUsername(**environment**.getRequiredProperty(***DB\_USERNAME***));  dataSource.setPassword(**environment**.getRequiredProperty(***DB\_PASSWORD***));  **return** dataSource;  }   @Bean  **public** LocalContainerEntityManagerFactoryBean entityManagerFactory() {  LocalContainerEntityManagerFactoryBean entityManagerFactoryBean = **new** LocalContainerEntityManagerFactoryBean();  entityManagerFactoryBean.setDataSource(dataSource());  entityManagerFactoryBean.setPersistenceProviderClass(HibernatePersistenceProvider.**class**);  entityManagerFactoryBean.setJpaVendorAdapter(**new** HibernateJpaVendorAdapter());  entityManagerFactoryBean.setPackagesToScan(***entityPackage***);  entityManagerFactoryBean.setJpaProperties(getHibernateProperties());  **return** entityManagerFactoryBean;  }    **private** Properties getHibernateProperties() {  Properties properties = **new** Properties();  properties.put(**"hibernate.dialect"**, **environment**.getRequiredProperty(***DB\_HIBERNATE\_DIALECT***));  properties.put(**"hibernate.hbm2ddl.auto"**, **environment**.getRequiredProperty(***DB\_HIBERNATE\_HBM2DDL\_AUTO***));  properties.put(**"hibernate.show\_sql"**, **environment**.getRequiredProperty(***DB\_HIBERNATE\_SHOW\_SQL***));  properties.put(**"hibernate.format\_sql"**, **environment**.getRequiredProperty(***DB\_HIBERNATE\_FORMAT\_SQL***));  **return** properties;  }   @Bean  **public** JpaTransactionManager transactionManager() {  JpaTransactionManager transactionManager = **new** JpaTransactionManager();  transactionManager.setEntityManagerFactory(entityManagerFactory().getObject());  **return** transactionManager;  } } |

В директории ресурсов создадим файл конфигурации application-repository.properties

Spring по умолчанию ищет файлы с именем application.properties или application.yml в проекте и применяет их как файл конфигурации. Но так как в нашем проекте несколько модулей, то если в проекте будет несколько файлов application.properties то будет использован ближайший к точке входа приложения, по этому для модулей, которые будут зависимостями будем именовать файлы так: application-{suffix}.properties . Кроме того файлы с таким именем необходимо будет указать в системное свойство spring.config.location.

Добавим файл application-repository.properties в директорию resources модуля Repository

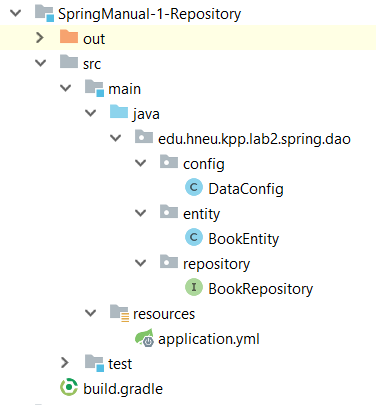
|  |
| --- |
| **db.driver**=**com.mysql.jdbc.Driver db.url**=**jdbc:mysql://localhost:3306/springManual?useUnicode=yes&characterEncoding=UTF-8&useSSL=false&createDatabaseIfNotExist=true db.username**=**root db.password**=**root** *#hibernate properties* **hibernate.dialect**=**org.hibernate.dialect.MySQL5InnoDBDialect hibernate.show\_sql**=**false** *#hibernate.show\_sql=true* **hibernate.format\_sql**=**true hibernate.hbm2ddl.auto**=**create-drop** *#hibernate.hbm2ddl.auto=update* |

Можно использовать файл application.yml вместо application.properties, но при этом необходимо отразить это в аннотации в конфигурации БД :

application-repository.yml

|  |
| --- |
| **db:  driver:** com.mysql.jdbc.Driver  **password:** root  **url:** jdbc:mysql://localhost:3306/springManual?useUnicode=yes&characterEncoding=UTF-8&useSSL=false&createDatabaseIfNotExist=true  **username:** root **hibernate:  dialect:** org.hibernate.dialect.MySQL5InnoDBDialect  **format\_sql:** true  **hbm2ddl:  auto:** update  **show\_sql:** true |

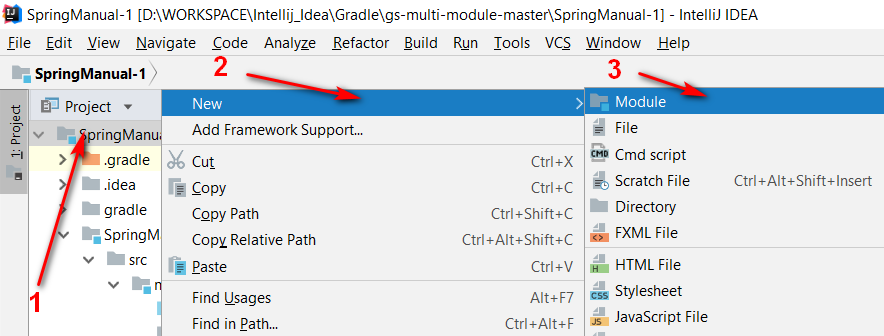
В результате имеем модуль обеспечивающий доступ к данным с такой структурой: *Рис. 37*



*Рис. 37* Структура модуля Repository

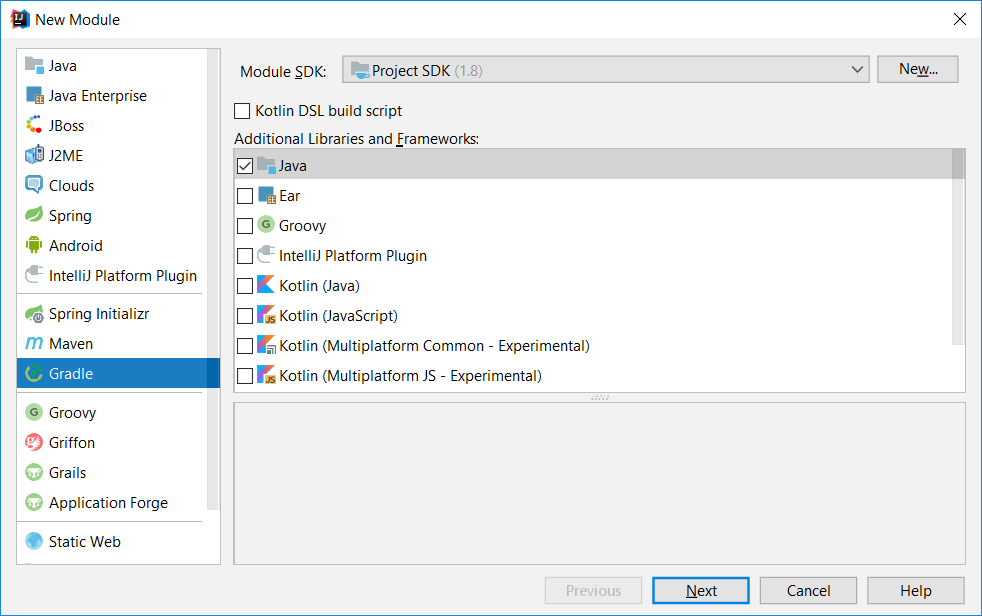
Напрямую использовать репозитории для получение данных не принято и считается плохим тоном и нарушением инкапсуляции, для этого были придуманы Services.

По этому создадим новый модуль Service, нажмем ПКМ по корневой директории проекта (*Рис. 38*) и выберем создание модуля

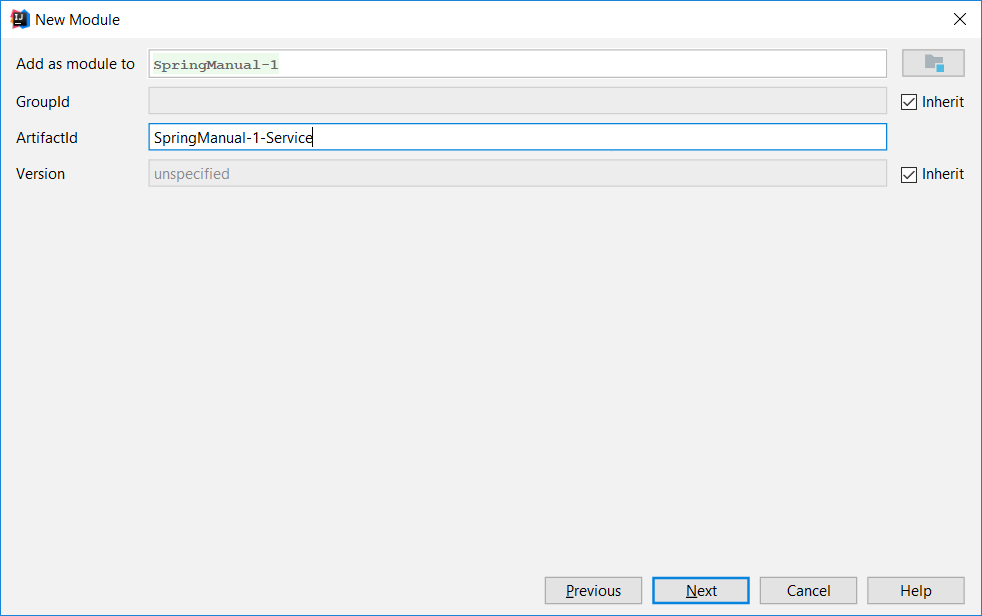


*Рис. 38* Создание нового модуля

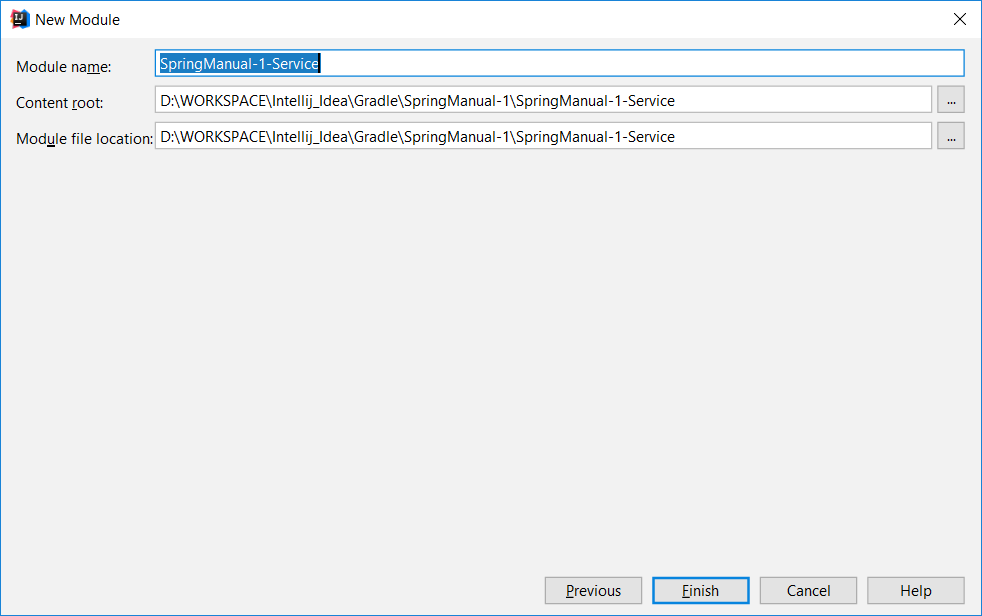
Пройдем шаги конфигурации модуля в соответствии с *Рис. 39* *Рис. 40* *Рис. 41*



*Рис. 39* Конфигурация нового модуля-1



*Рис. 40* Конфигурация нового модуля-2



*Рис. 41* Конфигурация нового модуля-3

Проверим файл settings.gradle в корне проекта, он должен содержать информацию о проекте и всех модулях проекта

|  |
| --- |
| rootProject.name = **'SpringManual-1'** include **'SpringManual-1-Repository'** include **'SpringManual-1-Service'** |

Правим build.gradle нового модуля, в зависимостях добавляем зависимость от модуля Repository

|  |
| --- |
| dependencies {  compile project(**':SpringManual-1-Repository'**)   compile(**"org.springframework.boot:spring-boot-starter-actuator:**${springBootVersion}**"**)  compile(**"org.springframework.boot:spring-boot-configuration-processor:**${springBootVersion}**"**)  compile(**"org.springframework.boot:spring-boot-starter-web:**${springBootVersion}**"**)   runtime(**"org.springframework.boot:spring-boot-devtools:**${springBootVersion}**"**)  testCompile(**"org.springframework.boot:spring-boot-starter-test:**${springBootVersion}**"**) } |

Создадим в новом модуле пакет edu.hneu.kpp.lab2.spring.service , а в нем пакеты service и app

В новом пакете service напишем BookService

|  |
| --- |
| **package** edu.hneu.kpp.lab2.spring.service.service;  **import** edu.hneu.kpp.lab2.spring.dao.entity.BookEntity;  **import** java.util.List;  **public interface** BookService {  BookEntity save(BookEntity book);   **void** delete(BookEntity book);   **void** deleteAll();   BookEntity findById(Integer id);   List<BookEntity> findAll(); } |

И его реализацию BookServiceImpl в service.implementation соответственно.

|  |
| --- |
| **package** edu.hneu.kpp.lab2.spring.service.service.impleventation;  **import** edu.hneu.kpp.lab2.spring.dao.entity.BookEntity; **import** edu.hneu.kpp.lab2.spring.dao.repository.BookRepository; **import** edu.hneu.kpp.lab2.spring.service.service.BookService; **import** org.springframework.stereotype.Service;  **import** javax.annotation.Resource; **import** java.util.List;  @Service **public class** BookServiceImpl **implements** BookService {  @Resource  **private** BookRepository **bookRepository**;   @Override  **public** BookEntity save(BookEntity book) {  **return bookRepository**.saveAndFlush(book);  }   @Override  **public void** delete(BookEntity book) {  **bookRepository**.delete(book);  }   @Override  **public void** deleteAll() {  **bookRepository**.deleteAll();  }   @Override  **public** BookEntity findById(Integer id) {  **return bookRepository**.findById(id).orElse(**null**);  }   @Override  **public** List<BookEntity> findAll() {  **return bookRepository**.findAll();  } } |

Для проверки работоспособности напишем в пакете app два класса

Application с точкой входа

|  |
| --- |
| **package** edu.hneu.kpp.lab2.spring.service.app;  **import** org.springframework.boot.SpringApplication; **import** org.springframework.boot.autoconfigure.EnableAutoConfiguration; **import** org.springframework.boot.autoconfigure.SpringBootApplication; **import** org.springframework.boot.builder.SpringApplicationBuilder; **import** org.springframework.boot.web.servlet.support.SpringBootServletInitializer; **import** org.springframework.context.ApplicationContext; **import** org.springframework.context.annotation.ComponentScan;  @SpringBootApplication(scanBasePackages = **"edu.hneu.kpp.lab2.spring"**) @EnableAutoConfiguration @ComponentScan({  **"edu.hneu.kpp.lab2.spring.service"**,  **"edu.hneu.kpp.lab2.spring.dao"**, }) **public class** Application **extends** SpringBootServletInitializer {  **public static final** Class[] ***classes*** = {Application.**class**,};  **public static** ApplicationContext *ac*;    @Override  **protected** SpringApplicationBuilder configure(SpringApplicationBuilder builder) {  **return** builder.sources(***classes***);  }   **public static void** main(String[] args) {  System.*setProperty*(**"spring.config.location"**, **"classpath:/application-service.properties,"** +  **"classpath:/application-repository.properties"**);  *ac* = SpringApplication.*run*(***classes***, args);  } } |

И StartupApplicationListener который будет запускаться каждый раз при запуске Application

|  |
| --- |
| **package** edu.hneu.kpp.lab2.spring.service.app;  **import** edu.hneu.kpp.lab2.spring.dao.entity.BookEntity; **import** edu.hneu.kpp.lab2.spring.service.service.BookService; **import** org.springframework.boot.ApplicationArguments; **import** org.springframework.boot.ApplicationRunner; **import** org.springframework.stereotype.Component;  **import** javax.annotation.Resource; **import** java.util.List;  @Component **public class** StartupApplicationListener **implements** ApplicationRunner {  @Resource  **private** BookService **bookService**;   @Override  **public void** run(ApplicationArguments args) {  test();  }   **public void** test() {  System.***out***.println(**"###### Saving book to db start ######"**);  BookEntity book1 = **bookService**.save(**new** BookEntity(**"The Brothers Karamazov"**, **"Fyodor Dostoevsky"**));  BookEntity book2 = **bookService**.save(**new** BookEntity(**"War and Peace"**, **"Leo Tolstoy"**));  BookEntity book3 = **bookService**.save(**new** BookEntity(**"Pride and Prejudice"**, **"Jane Austen"**));  System.***out***.println(**"###### Saving book to db finish ######"**);  System.***out***.println(**"Find all :"**);  List<BookEntity> books1 = **bookService**.findAll();  **for** (BookEntity b : books1) {  System.***out***.println(**"-"** + b.toString());  }  System.***out***.println(**"###### Update - start ######"**);  book1.setTitle(**"The Idiot"**);  **bookService**.save(book1);  System.***out***.println(**"Book Updated is =>"** + **bookService**.findById(book1.getId()).toString());  System.***out***.println(**"###### Update - end ######"**);  System.***out***.println(**"###### Find - start ######"**);  Integer id1 = book1.getId();  BookEntity another = **bookService**.findById(id1);  System.***out***.println(**"Book found with id "** + id1 + **" is =>"** + another.toString());  System.***out***.println(**"###### Find - end ######"**);  System.***out***.println(**"###### Delete - start ######"**);  Integer id3 = book3.getId();  **bookService**.delete(book3);  System.***out***.println(**"Deleted book with id "** + id3 + **"."**);  System.***out***.println(**"Now all books are "** + **bookService**.findAll().size() + **"."**);  System.***out***.println(**"###### Delete - end ######"**);  System.***out***.println(**"###### FindAll - start ######"**);  List<BookEntity> books2 = **bookService**.findAll();  System.***out***.println(**"Books found are :"**);  **for** (BookEntity b : books2) {  System.***out***.println(**"-"** + b.toString());  }  System.***out***.println(**"###### FindAll - end ######"**);  System.***out***.println(**"###### DeleteAll - start ######"**);  **bookService**.deleteAll();  System.***out***.println(**"Books found are now "** + **bookService**.findAll().size());  System.***out***.println(**"###### DeleteAll - end ######"**);  } } |

Запустим приложение и в консоли увидим следующие сообщения:

|  |
| --- |
| ###### Saving book to db start ######  ###### Saving book to db finish ######  Find all :  -BookEntity [id=1, title=The Brothers Karamazov, author=Fyodor Dostoevsky]  -BookEntity [id=2, title=War and Peace, author=Leo Tolstoy]  -BookEntity [id=3, title=Pride and Prejudice, author=Jane Austen]  ###### Update - start ######  Book Updated is =>BookEntity [id=1, title=The Idiot, author=Fyodor Dostoevsky]  ###### Update - end ######  ###### Find - start ######  Book found with id 1 is =>BookEntity [id=1, title=The Idiot, author=Fyodor Dostoevsky]  ###### Find - end ######  ###### Delete - start ######  Deleted book with id 3.  Now all books are 2.  ###### Delete - end ######  ###### FindAll - start ######  Books found are :  -BookEntity [id=1, title=The Idiot, author=Fyodor Dostoevsky]  -BookEntity [id=2, title=War and Peace, author=Leo Tolstoy]  ###### FindAll - end ######  ###### DeleteAll - start ######  Books found are now 0  ###### DeleteAll - end ###### |

Web приложение

Добавим новый модуль SpringManual-1-Webapp в проект.

Настроим build.gradle

|  |
| --- |
| apply **plugin**: **'org.springframework.boot'** apply **plugin**: **'io.spring.dependency-management'** dependencies {  compile project(**':SpringManual-1-Service'**)   compile(**"org.springframework.boot:spring-boot-starter-actuator:**${springBootVersion}**"**)  compile(**"org.springframework.boot:spring-boot-starter-thymeleaf:**${springBootVersion}**"**)  compile(**"org.springframework.boot:spring-boot-starter-web:**${springBootVersion}**"**)  compile(**"org.springframework.boot:spring-boot-configuration-processor:**${springBootVersion}**"**)  compile(**"org.apache.tomcat.embed:tomcat-embed-jasper:**${tomcatEmbedJasperVersion}**"**)  compile(**"org.apache.httpcomponents:httpclient:**${httpclientVersion}**"**)   runtime(**"org.springframework.boot:spring-boot-devtools:**${springBootVersion}**"**)  testCompile(**"org.springframework.boot:spring-boot-starter-test:**${springBootVersion}**"**) } |

Создаем пакет edu.hneu.kpp.lab2.spring.webapp в котором создаем пакеты controller и model.

В пакете model создадим пакет config и в нем создадим класс конфигурации web приложения WebConfiguration в котором в последствии будем добавлять конфигурацию ресурсов.

|  |
| --- |
| **package** edu.hneu.kpp.lab2.spring.webapp.model.config;  **import** org.springframework.context.annotation.Configuration; **import** org.springframework.web.servlet.config.annotation.WebMvcConfigurer;  @Configuration **public class** WebConfiguration **implements** WebMvcConfigurer { } |

В пакете model создаем класс WebApp аналогично классу App пакета Service.

|  |
| --- |
| **package** edu.hneu.kpp.lab2.spring.webapp.model;  **import** org.springframework.boot.SpringApplication; **import** org.springframework.boot.autoconfigure.EnableAutoConfiguration; **import** org.springframework.boot.autoconfigure.SpringBootApplication; **import** org.springframework.boot.builder.SpringApplicationBuilder; **import** org.springframework.boot.web.servlet.support.SpringBootServletInitializer; **import** org.springframework.context.ApplicationContext; **import** org.springframework.context.annotation.ComponentScan;  @SpringBootApplication(scanBasePackages = **"edu.hneu.kpp.lab2.spring"**) @EnableAutoConfiguration @ComponentScan({  **"edu.hneu.kpp.lab2.spring.dao"**,  **"edu.hneu.kpp.lab2.spring.service.service"**,  **"edu.hneu.kpp.lab2.spring.webapp.controller"**,  **"edu.hneu.kpp.lab2.spring.webapp.model.config"**,  **"edu.hneu.kpp.lab2.spring.webapp.model"**, }) **public class** WebApp **extends** SpringBootServletInitializer {  **public static final** Class[] ***classes*** = {WebApp.**class**,};  **public static** ApplicationContext *ac*;   @Override  **protected** SpringApplicationBuilder configure(SpringApplicationBuilder builder) {  **return** builder.sources(***classes***);  }   **public static void** main(String[] args) {  System.*setProperty*(**"spring.config.location"**, **"classpath:/application.properties,"** +  **"classpath:/application-service.properties,"** +  **"classpath:/application-repository.properties"**);  *ac* = SpringApplication.*run*(***classes***, args);  } } |

В пакете controller создадим два класса, финализированный класс констант ControllerConstants

|  |
| --- |
| **package** edu.hneu.kpp.lab2.spring.webapp.controller;  **public final class** ControllerConstants {  **public static final** String ***BOOK\_CONTROLLER\_REQUEST\_MAPPING*** = **"/bookController"**; } |

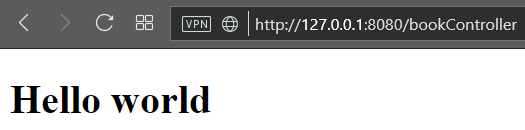
И класс контроллер BookController

|  |
| --- |
| **package** edu.hneu.kpp.lab2.spring.webapp.controller;  **import** org.springframework.stereotype.Controller; **import** org.springframework.web.bind.annotation.GetMapping; **import** org.springframework.web.bind.annotation.RequestMapping;  @Controller @RequestMapping(ControllerConstants.***BOOK\_CONTROLLER\_REQUEST\_MAPPING***) **public class** BookController {  **private static final** String ***VIEW\_NAME*** = **"bookList"**;   @GetMapping  **public** String get() {  **return *VIEW\_NAME***;  } } |

В директории resources добавим директорию templates и в ней создадим представление bookList.html

|  |
| --- |
| <!DOCTYPE **html**> <**html xmlns:th="http://www.thymeleaf.org"**> <**head**>  <**meta charset="UTF-8"**>  <**title**>Title</**title**> </**head**> <**body**> <**h1**>Hello world</**h1**> </**body**> </**html**> |

Далее можем запустить наше веб приложение (класс WebApp) и проверить его работоспособность (*Рис. 42*) по адресу <http://127.0.0.1:8080/bookController>



*Рис. 42* Первый запуск web приложения

Добавим функционал вывода данных о книге.

Но cначала добавим книги в бд. Для этого добавим в модуль Web в пакет edu.hneu.kpp.lab2.spring.service.app класс StartupApplicationListener

|  |
| --- |
| **package** edu.hneu.kpp.lab2.spring.webapp.model;  **import** edu.hneu.kpp.lab2.spring.dao.entity.BookEntity; **import** edu.hneu.kpp.lab2.spring.service.service.BookService; **import** org.springframework.boot.ApplicationArguments; **import** org.springframework.boot.ApplicationRunner; **import** org.springframework.stereotype.Component;  **import** javax.annotation.Resource;  @Component **public class** StartupApplicationListener **implements** ApplicationRunner {  @Resource  **private** BookService **bookService**;   @Override  **public void** run(ApplicationArguments args) {  initFillDB();  }   **public void** initFillDB() {  BookEntity book1 = **bookService**.save(**new** BookEntity(**"The Brothers Karamazov"**, **"Fyodor Dostoevsky"**));  BookEntity book2 = **bookService**.save(**new** BookEntity(**"War and Peace"**, **"Leo Tolstoy"**));  BookEntity book3 = **bookService**.save(**new** BookEntity(**"Pride and Prejudice"**, **"Jane Austen"**));  } } |

Изменения в классах будем изображать таблицами сравнения приводя части класса в которых были изменения и выделяя добавленное зеленым, измененное синим, а удаленное оранжевым фоном.

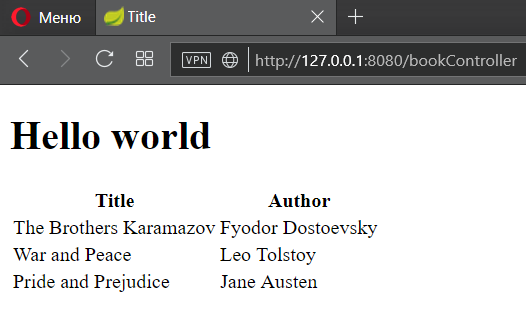
Изменим содержимое BookController:

|  |  |
| --- | --- |
| Было | Стало |
| @Controller @RequestMapping(ControllerConstants.***BOOK\_CONTROLLER\_REQUEST\_MAPPING***) **public class** BookController {  **private static final** String ***VIEW\_NAME*** = **"bookList"**;  @GetMapping   **public** String get() {  **return *VIEW\_NAME***;  } } | @Controller @RequestMapping(ControllerConstants.***BOOK\_CONTROLLER\_REQUEST\_MAPPING***) **public class** BookController {  **private static final** String ***VIEW\_NAME*** = **"bookList"**;  **private static final** String ***MODEL\_NAME*** = **"bookListModel"**;  @Resource  **private** BookService **bookService**;   @GetMapping  **public** ModelAndView get() {  **return new** ModelAndView(***VIEW\_NAME***, ***MODEL\_NAME***, **bookService**.findAll());  } } |

А также содержимое bookList.html

|  |  |
| --- | --- |
| Было | Стало |
| <!DOCTYPE **html**> <**html xmlns:th="http://www.thymeleaf.org"**> <**head**>  <**meta charset="UTF-8"**>  <**title**>Title</**title**> </**head**> <**body**> <**h1**>Hello world</**h1**> </**body**> </**html**> | <!DOCTYPE **html**> <**html xmlns:th="http://www.thymeleaf.org"**> <**head**>  <**meta charset="UTF-8"**>  <**title**>Title</**title**> </**head**> <**body**>> <**table**>  <**thead**>  <**tr**>  <**th**>Title</**th**>  <**th**>Author</**th**>  </**tr**>  </**thead**>  <**tbody**>  *<!--/\*@thymesVar id="bookListModel" type="java.util.List"\*/-->  <!--/\*@thymesVar id="book" type="edu.hneu.kpp.lab2.spring.dao.entity.BookEntity"\*/-->* <**tr th:each="book:${ bookListModel}"**>  <**td th:text="${book.title}"**></**td**>  <**td th:text="${book.author}"**></**td**>  </**tr**>  </**tbody**> </**table**> </**body**> </**html**> |

Запустим и по адресу <http://127.0.0.1:8080/bookController> увидим данные о книгах в БД *Рис. 43*



*Рис. 43* Данные о книгах

Добавим в БД еще две сущности, Автор и Издательство, для того чтоб продемонстрировать связи типа один ко многим и многие ко многим.

Сущности типа Автор:

В пакете Repository рядом с сущностью BookEntity добавим класс AuthorEntity

|  |
| --- |
| **package** edu.hneu.kpp.lab2.spring.dao.entity;  **import** javax.persistence.\*; **import** java.util.Set;  @Entity @Table(name = **"author"**) **public class** AuthorEntity {  @Id  @GeneratedValue(strategy = GenerationType.***IDENTITY***)  @Column(name = **"id"**, unique = **true**, nullable = **false**)  **private int id**;  @Column(name = **"name"**, nullable = **false**)  **private** String **name**;  @Column(name = **"surname"**)  **private** String **surname**;  @OneToMany(fetch = FetchType.***EAGER***, mappedBy = **"authorEntity"**, cascade = CascadeType.***ALL***)  **private** Set<BookEntity> **bookEntities**;   **public** AuthorEntity() {  }   **public** AuthorEntity(String name, String surname) {  **this**.**name** = name;  **this**.**surname** = surname;  }   **public int** getId() {  **return id**;  }   **public void** setId(**int** id) {  **this**.**id** = id;  }   **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** Set<BookEntity> getBookEntities() {  **return bookEntities**;  }   **public void** setBookEntities(Set<BookEntity> bookEntities) {  **this**.**bookEntities** = bookEntities;  }   @Override  **public** String toString() {  **return name** + **" "** + **surname**;  } } |

И репозиторий класса Автор – AuthorRepository в рядом с BookRepository

|  |
| --- |
| **package** edu.hneu.kpp.lab2.spring.dao.repository;  **import** edu.hneu.kpp.lab2.spring.dao.entity.AuthorEntity; **import** org.springframework.data.jpa.repository.JpaRepository;  **public interface** AuthorRepository **extends** JpaRepository<AuthorEntity, Integer> { } |

Так же изменим класс BookEntity:

|  |  |
| --- | --- |
| Было | Стало |
| @Entity @Table(name = **"book"**) **public class** BookEntity {  @Id  @GeneratedValue(strategy = GenerationType.***IDENTITY***)  @Column(name = **"id"**, unique = **true**, nullable = **false**)  **private int id**;  @Column(name = **"title"**)  **private** String **title**;  @Column(name = **"author"**)  **private** String **author**;     **public** BookEntity() { }   **public** BookEntity(String title, String author) {  **this**.**title** = title;  **this**.**author** = author;  }   **//другие get-set методы не изменились**  **public** String getAuthor() {  **return author**;  }   **public void** setAuthor(String author) {  **this**.**author** = author;  }   @Override  **public** String toString() {  **return "BookEntity [id="** + **id** + **", title="** + **title** + **", author="** + **author** + **"]"**;  } } | @Entity @Table(name = **"book"**) **public class** BookEntity {  @Id  @GeneratedValue(strategy = GenerationType.***IDENTITY***)  @Column(name = **"id"**, unique = **true**, nullable = **false**)  **private int id**;  @Column(name = **"title"**)  **private** String **title**;  @ManyToOne  @JoinColumn(name = **"author\_id"**, referencedColumnName = **"id"**, nullable = **false**)  **private** AuthorEntity **authorEntity**;   **public** BookEntity() { }   **public** BookEntity(String title, AuthorEntity authorEntity) {  **this**.**title** = title;  **this**.**authorEntity** = authorEntity;  }   **// другие get-set методы не изменились**   **public** AuthorEntity getAuthorEntity() {  **return authorEntity**;  }   **public void** setAuthorEntity(AuthorEntity authorEntity) {  **this**.**authorEntity** = authorEntity;  }   @Override  **public** String toString() {  **return "BookEntity [id="** + **id** + **", title="** + **title** + **", author="** + **authorEntity**.toString() + **"]"**;  } } |

Добавим новую функциональность в модуль Service – новый интерфейс AuthorService:

|  |
| --- |
| **package** edu.hneu.kpp.lab2.spring.service.service;  **import** edu.hneu.kpp.lab2.spring.dao.entity.AuthorEntity;  **import** java.util.List;  **public interface** AuthorService {  AuthorEntity save(AuthorEntity author);   **void** delete(AuthorEntity author);   **void** deleteAll();   AuthorEntity findById(Integer id);   List<AuthorEntity> findAll(); } |

И его реализацию AuthorServiceImpl

|  |
| --- |
| **package** edu.hneu.kpp.lab2.spring.service.service.impleventation;  **import** edu.hneu.kpp.lab2.spring.dao.entity.AuthorEntity; **import** edu.hneu.kpp.lab2.spring.dao.repository.AuthorRepository; **import** edu.hneu.kpp.lab2.spring.service.service.AuthorService; **import** org.springframework.stereotype.Service;  **import** javax.annotation.Resource; **import** java.util.List;  @Service **public class** AuthorServiceImpl **implements** AuthorService {  @Resource  **private** AuthorRepository **authorRepository**;   @Override  **public** AuthorEntity save(AuthorEntity author) {  **return authorRepository**.saveAndFlush(author);  }   @Override  **public void** delete(AuthorEntity author) {  **authorRepository**.delete(author);  }   @Override  **public void** deleteAll() {  **authorRepository**.deleteAll();  }   @Override  **public** AuthorEntity findById(Integer id) {  **return authorRepository**.findById(id).orElse(**null**);  }   @Override  **public** List<AuthorEntity> findAll() {  **return authorRepository**.findAll();  } } |

Реализована связь один ко многим.

Внесем так же изменения в StartupApplicationListener модуля Service

|  |  |
| --- | --- |
| Было | Стало |
| @Component public class StartupApplicationListener implements ApplicationRunner {  @Resource  private BookService bookService;    @Override   public void run(ApplicationArguments args) {  test();  }   public void test() {   System.*out*.println("###### Saving book to db start ######");  BookEntity book1 = bookService.save(new BookEntity("The Brothers Karamazov", "Fyodor Dostoevsky"));  BookEntity book2 = bookService.save(new BookEntity("War and Peace", "Leo Tolstoy"));  BookEntity book3 = bookService.save(new BookEntity("Pride and Prejudice", "Jane Austen"));  System.*out*.println("###### Saving book to db finish ######");  System.*out*.println("Find all :");  List<BookEntity> books1 = bookService.findAll();  for (BookEntity b : books1) {  System.*out*.println("-" + b.toString());  }  System.*out*.println("###### Update - start ######");  book1.setTitle("The Idiot");  bookService.save(book1);  System.*out*.println("Book Updated is =>" + bookService.findById(book1.getId()).toString());  System.*out*.println("###### Update - end ######");  System.*out*.println("###### Find - start ######");  Integer id1 = book1.getId();  BookEntity another = bookService.findById(id1);  System.*out*.println("Book found with id " + id1 + " is =>" + another.toString());  System.*out*.println("###### Find - end ######");  System.*out*.println("###### Delete - start ######");  Integer id3 = book3.getId();  bookService.delete(book3);  System.*out*.println("Deleted book with id " + id3 + ".");  System.*out*.println("Now all books are " + bookService.findAll().size() + ".");  System.*out*.println("###### Delete - end ######");  System.*out*.println("###### FindAll - start ######");  List<BookEntity> books2 = bookService.findAll();  System.*out*.println("Books found are :");  for (BookEntity b : books2) {  System.*out*.println("-" + b.toString());  }  System.*out*.println("###### FindAll - end ######");  System.*out*.println("###### DeleteAll - start ######");  bookService.deleteAll();  System.*out*.println("Books found are now " + bookService.findAll().size());  System.*out*.println("###### DeleteAll - end ######");  } } | @Component **public class** StartupApplicationListener **implements** ApplicationRunner {  @Resource  **private** BookService **bookService**;  @Resource  **private** AuthorService **authorService**;   @Override  **public void** run(ApplicationArguments args) {  test();  }   **public void** test() {  System.***out***.println(**"###### Saving author to db start ######"**);  AuthorEntity author1 = **authorService**.save(**new** AuthorEntity(**"Fyodor"**, **"Dostoevsky"**));  AuthorEntity author2 = **authorService**.save(**new** AuthorEntity(**"Leo"**, **"Tolstoy"**));  AuthorEntity author3 = **authorService**.save(**new** AuthorEntity(**"Jane"**, **"Austen"**));  System.***out***.println(**"###### Saving author to db finish ######"**);  System.***out***.println(**"###### Saving book to db start ######"**);  BookEntity book1 = **bookService**.save(**new** BookEntity(**"The Brothers Karamazov"**, author1));  BookEntity book2 = **bookService**.save(**new** BookEntity(**"War and Peace"**, author2));  BookEntity book3 = **bookService**.save(**new** BookEntity(**"Pride and Prejudice"**, author3));  System.***out***.println(**"###### Saving book to db finish ######"**);  System.***out***.println(**"###### FindAll - start ######"**);  **bookService**.findAll().forEach(System.***out***::println);  System.***out***.println(**"###### Update - start ######"**);  book1.setTitle(**"The Idiot"**);  **bookService**.save(book1);  System.***out***.println(**"Book Updated is =>"** + **bookService**.findById(book1.getId()).toString());  System.***out***.println(**"###### Update - end ######"**);  System.***out***.println(**"###### Find - start ######"**);  Integer id1 = book1.getId();  BookEntity another = **bookService**.findById(id1);  System.***out***.println(**"Book found with id "** + id1 + **" is =>"** + another.toString());  System.***out***.println(**"###### Find - end ######"**);  System.***out***.println(**"###### Delete - start ######"**);  Integer id3 = book3.getId();  **bookService**.delete(book3);  System.***out***.println(**"Deleted book with id "** + id3 + **"."**);  System.***out***.println(**"Now all books are "** + **bookService**.findAll().size() + **"."**);  System.***out***.println(**"###### Delete - end ######"**);  System.***out***.println(**"###### FindAll - start ######"**);  System.***out***.println(**"Books found are :"**);  **bookService**.findAll().forEach(System.***out***::println);  System.***out***.println(**"###### FindAll - end ######"**);  System.***out***.println(**"###### Authors ######"**);  System.***out***.println(**"###### FindAll - start ######"**);  **authorService**.findAll().forEach(System.***out***::println);  System.***out***.println(**"###### FindAll - end ######"**);  System.***out***.println(**"###### Books by Authors - start ######"**);  **authorService**.findAll().forEach(author ->  {  System.***out***.println(author);  author.getBookEntities().forEach(System.***out***::println);  System.***out***.println(**"=============="**);  });  System.***out***.println(**"###### Books by Authors - end ######"**);  System.***out***.println(**"###### DeleteAll - start ######"**);  **bookService**.deleteAll();  **authorService**.deleteAll();  System.***out***.println(**"Books found are now "** + **bookService**.findAll().size());  System.***out***.println(**"Authors found are now "** + **authorService**.findAll().size());  System.***out***.println(**"###### DeleteAll - end ######"**);  } } |

Запустим модуль Service на выполнение и увидим такой вывод:

|  |
| --- |
| ###### Saving author to db start ######  ###### Saving author to db finish ######  ###### Saving book to db start ######  ###### Saving book to db finish ######  ###### FindAll - start ######  BookEntity [id=1, title=The Brothers Karamazov, author=Fyodor Dostoevsky]  BookEntity [id=2, title=War and Peace, author=Leo Tolstoy]  BookEntity [id=3, title=Pride and Prejudice, author=Jane Austen]  ###### Update - start ######  Book Updated is =>BookEntity [id=1, title=The Idiot, author=Fyodor Dostoevsky]  ###### Update - end ######  ###### Find - start ######  Book found with id 1 is =>BookEntity [id=1, title=The Idiot, author=Fyodor Dostoevsky]  ###### Find - end ######  ###### Delete - start ######  Deleted book with id 3.  Now all books are 3.  ###### Delete - end ######  ###### FindAll - start ######  Books found are :  BookEntity [id=1, title=The Idiot, author=Fyodor Dostoevsky]  BookEntity [id=2, title=War and Peace, author=Leo Tolstoy]  BookEntity [id=3, title=Pride and Prejudice, author=Jane Austen]  ###### FindAll - end ######  ###### Authors ######  ###### FindAll - start ######  Fyodor Dostoevsky  Leo Tolstoy  Jane Austen  ###### FindAll - end ######  ###### Books by Authors - start ######  Fyodor Dostoevsky  BookEntity [id=1, title=The Idiot, author=Fyodor Dostoevsky]  ==============  Leo Tolstoy  BookEntity [id=2, title=War and Peace, author=Leo Tolstoy]  ==============  Jane Austen  BookEntity [id=3, title=Pride and Prejudice, author=Jane Austen]  ==============  ###### Books by Authors - end ######  ###### DeleteAll - start ######  Books found are now 0  Authors found are now 0  ###### DeleteAll - end ###### |

Реализуем связь многие ко многим, для этого добавим сущность – издательство (PublishingHouseEntity)

|  |
| --- |
| **package** edu.hneu.kpp.lab2.spring.dao.entity;  **import** javax.persistence.\*; **import** java.util.Set;  @Entity @Table(name = **"publishing\_house"**) **public class** PublishingHouseEntity {  @Id  @GeneratedValue(strategy = GenerationType.***IDENTITY***)  @Column(name = **"id"**, unique = **true**, nullable = **false**)  **private int id**;  @Column(name = **"name"**, nullable = **false**)  **private** String **name**;  @ManyToMany(fetch = FetchType.***EAGER***)  @JoinTable(name = **"book2publishing\_house"**,  joinColumns = @JoinColumn(name = **"book\_id"**, referencedColumnName = **"id"**),  inverseJoinColumns = @JoinColumn(name = **"publishing\_house\_id"**, referencedColumnName = **"id"**))  **private** Set<BookEntity> **bookEntities**;   **public** PublishingHouseEntity(String name, Set<BookEntity> books) {  **this**.**name** = name;  **this**.**bookEntities** = books;  }   **public** PublishingHouseEntity() {  }   **public int** getId() {  **return id**;  }   **public void** setId(**int** id) {  **this**.**id** = id;  }   **public** String getName() {  **return name**;  }   **public void** setName(String name) {  **this**.**name** = name;  }   **public** Set<BookEntity> getBookEntities() {  **return bookEntities**;  }   **public void** setBookEntities(Set<BookEntity> bookEntities) {  **this**.**bookEntities** = bookEntities;  }   @Override  **public** String toString() {  **return name**;  } } |

И изменим BookEntity

|  |  |
| --- | --- |
| Было | Стало |
| @Entity @Table(name = **"book"**) **public class** BookEntity {  @Id  @GeneratedValue(strategy = GenerationType.***IDENTITY***)  @Column(name = **"id"**, unique = **true**, nullable = **false**)  **private int id**;  @Column(name = **"title"**)  **private** String **title**;  @ManyToOne  @JoinColumn(name = **"author\_id"**, referencedColumnName = **"id"**, nullable = **false**)  **private** AuthorEntity **authorEntity**;  **//другие методы не изменились**   @Override  **public** String toString() {  **return "BookEntity [id="** + **id** + **", title="** + **title** + **", author="** + **authorEntity**.toString() + **"]"**;  } } | @Entity @Table(name = **"book"**) **public class** BookEntity {  @Id  @GeneratedValue(strategy = GenerationType.***IDENTITY***)  @Column(name = **"id"**, unique = **true**, nullable = **false**)  **private int id**;  @Column(name = **"title"**)  **private** String **title**;  @ManyToOne  @JoinColumn(name = **"author\_id"**, referencedColumnName = **"id"**, nullable = **false**)  **private** AuthorEntity **authorEntity**;  @ManyToMany(mappedBy = **"bookEntities"**, fetch = FetchType.***EAGER***)  **private** Set<PublishingHouseEntity> **publishingHouseEntities**;  **//другие методы не изменились**   **public** Set<PublishingHouseEntity> getPublishingHouseEntities() {  **return publishingHouseEntities**;  }   **public void** setPublishingHouseEntities(Set<PublishingHouseEntity> publishingHouseEntities) {  **this**.**publishingHouseEntities** = publishingHouseEntities;  }   @Override  **public** String toString() {  **return "BookEntity [id="** + **id** + **", title="** + **title** + **", author="** + **authorEntity**.toString() + **"]"**;  } } |

Так же добавим соответствующий репозиторий сервис и реализацию

PublishingHouseRepository

|  |
| --- |
| **package** edu.hneu.kpp.lab2.spring.dao.repository;  **import** edu.hneu.kpp.lab2.spring.dao.entity.PublishingHouseEntity; **import** org.springframework.data.jpa.repository.JpaRepository;  **public interface** PublishingHouseRepository **extends** JpaRepository<PublishingHouseEntity, Integer> { } |

PublishingHouseService

|  |
| --- |
| **package** edu.hneu.kpp.lab2.spring.service.service;  **import** edu.hneu.kpp.lab2.spring.dao.entity.PublishingHouseEntity;  **import** java.util.List;  **public interface** PublishingHouseService {  PublishingHouseEntity save(PublishingHouseEntity book);   **void** delete(PublishingHouseEntity book);   **void** deleteAll();   PublishingHouseEntity findById(Integer id);   List<PublishingHouseEntity> findAll(); } |

PublishingHouseServiceImpl

|  |
| --- |
| **package** edu.hneu.kpp.lab2.spring.service.service.implementation;  **import** edu.hneu.kpp.lab2.spring.dao.entity.PublishingHouseEntity; **import** edu.hneu.kpp.lab2.spring.dao.repository.PublishingHouseRepository; **import** edu.hneu.kpp.lab2.spring.service.service.PublishingHouseService; **import** org.springframework.stereotype.Service;  **import** javax.annotation.Resource; **import** java.util.List;  @Service **public class** PublishingHouseServiceImpl **implements** PublishingHouseService {  @Resource  **private** PublishingHouseRepository **publishingHouseRepository**;   @Override  **public** PublishingHouseEntity save(PublishingHouseEntity publishingHouseEntity) {  **return publishingHouseRepository**.saveAndFlush(publishingHouseEntity);  }   @Override  **public void** delete(PublishingHouseEntity publishingHouseEntity) {  **publishingHouseRepository**.delete(publishingHouseEntity);  }   @Override  **public void** deleteAll() {  **publishingHouseRepository**.deleteAll();  }   @Override  **public** PublishingHouseEntity findById(Integer id) {  **return publishingHouseRepository**.findById(id).orElse(**null**);  }   @Override  **public** List<PublishingHouseEntity> findAll() {  **return publishingHouseRepository**.findAll();  } } |

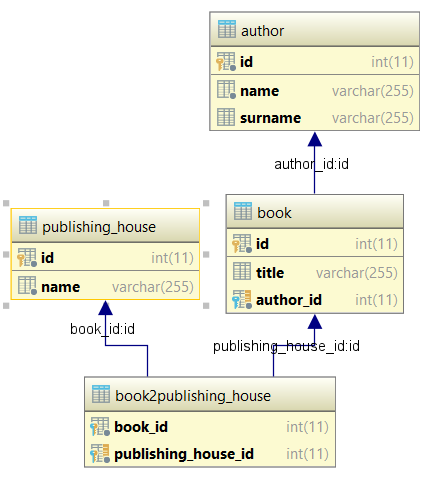
Для проверки изменим StartupApplicationListener

|  |  |
| --- | --- |
| Было | Стало |
| @Component **public class** StartupApplicationListener **implements** ApplicationRunner {  @Resource  **private** BookService **bookService**;  @Resource  **private** AuthorService **authorService**;   @Override  **public void** run(ApplicationArguments args) {  test();  }   **public void** test() {  System.***out***.println(**"###### Saving author to db start ######"**);  AuthorEntity author1 = **authorService**.save(**new** AuthorEntity(**"Fyodor"**, **"Dostoevsky"**));  AuthorEntity author2 = **authorService**.save(**new** AuthorEntity(**"Leo"**, **"Tolstoy"**));  AuthorEntity author3 = **authorService**.save(**new** AuthorEntity(**"Jane"**, **"Austen"**));  System.***out***.println(**"###### Saving author to db finish ######"**);  System.***out***.println(**"###### Saving book to db start ######"**);  BookEntity book1 = **bookService**.save(**new** BookEntity(**"The Brothers Karamazov"**, author1));  BookEntity book2 = **bookService**.save(**new** BookEntity(**"War and Peace"**, author2));  BookEntity book3 = **bookService**.save(**new** BookEntity(**"Pride and Prejudice"**, author3));  System.***out***.println(**"###### Saving book to db finish ######"**);  System.***out***.println(**"###### FindAll - start ######"**);  **bookService**.findAll().forEach(System.***out***::println);  System.***out***.println(**"###### Update - start ######"**);  book1.setTitle(**"The Idiot"**);  **bookService**.save(book1);  System.***out***.println(**"Book Updated is =>"** + **bookService**.findById(book1.getId()).toString());  System.***out***.println(**"###### Update - end ######"**);  System.***out***.println(**"###### Find - start ######"**);  Integer id1 = book1.getId();  BookEntity another = **bookService**.findById(id1);  System.***out***.println(**"Book found with id "** + id1 + **" is =>"** + another.toString());  System.***out***.println(**"###### Find - end ######"**);  System.***out***.println(**"###### Delete - start ######"**);  Integer id3 = book3.getId();  **bookService**.delete(book3);  System.***out***.println(**"Deleted book with id "** + id3 + **"."**);  System.***out***.println(**"Now all books are "** + **bookService**.findAll().size() + **"."**);  System.***out***.println(**"###### Delete - end ######"**);  System.***out***.println(**"###### FindAll - start ######"**);  System.***out***.println(**"Books found are :"**);  **bookService**.findAll().forEach(System.***out***::println);  System.***out***.println(**"###### FindAll - end ######"**);  System.***out***.println(**"###### Authors ######"**);  System.***out***.println(**"###### FindAll - start ######"**);  **authorService**.findAll().forEach(System.***out***::println);  System.***out***.println(**"###### FindAll - end ######"**);  System.***out***.println(**"###### Books by Authors - start ######"**);  **authorService**.findAll().forEach(author ->  {  System.***out***.println(author);  author.getBookEntities().forEach(System.***out***::println);  System.***out***.println(**"=============="**);  });  System.***out***.println(**"###### Books by Authors - end ######"**);  System.***out***.println(**"###### DeleteAll - start ######"**);  **bookService**.deleteAll();  **authorService**.deleteAll();  System.***out***.println(**"Books found are now "** + **bookService**.findAll().size());  System.***out***.println(**"Authors found are now "** + **authorService**.findAll().size());  System.***out***.println(**"###### DeleteAll - end ######"**);  } } | @Component **public class** StartupApplicationListener **implements** ApplicationRunner {  @Resource  **private** BookService **bookService**;  @Resource  **private** AuthorService **authorService**;  @Resource  **private** PublishingHouseService **publishingHouseService**;   @Override  **public void** run(ApplicationArguments args) {  test();  }   **public void** test() {  AuthorEntity author1 = **authorService**.save(**new** AuthorEntity(**"Fyodor"**, **"Dostoevsky"**));  AuthorEntity author2 = **authorService**.save(**new** AuthorEntity(**"Leo"**, **"Tolstoy"**));  AuthorEntity author3 = **authorService**.save(**new** AuthorEntity(**"Jane"**, **"Austen"**));  BookEntity book1 = **bookService**.save(**new** BookEntity(**"The Brothers Karamazov"**, author1));  BookEntity book2 = **bookService**.save(**new** BookEntity(**"War and Peace"**, author2));  BookEntity book3 = **bookService**.save(**new** BookEntity(**"Pride and Prejudice"**, author3));  PublishingHouseEntity publishingHouse1 =  **publishingHouseService**.save(**new** PublishingHouseEntity(**"Star PH"**, **new** HashSet<>(Arrays.*asList*(book1, book2))));  PublishingHouseEntity publishingHouse2 =  **publishingHouseService**.save(**new** PublishingHouseEntity(**"Kharkov Central PH"**, **new** HashSet<>(Arrays.*asList*(book2, book3))));   List<PublishingHouseEntity> publishingHouseEntities = **publishingHouseService**.findAll();  **for** (PublishingHouseEntity pe : publishingHouseEntities) {  System.***out***.println(**"##########################################################"**);  System.***out***.println(pe);  System.***out***.println(**"--------------------------"**);  pe.getBookEntities().forEach(bookEntity -> System.***out***.printf(**"%75s --- %25s"** + System.*lineSeparator*(), bookEntity, bookEntity.getAuthorEntity()));  } *// bookService.deleteAll(); // authorService.deleteAll(); // publishingHouseService.deleteAll();* } } |

Снова запустим модуль Service и получим такой вывод

|  |
| --- |
| ############################################################################  Star PH  --------------------------  BookEntity [id=1, title=The Brothers Karamazov, author=Fyodor Dostoevsky] --- Fyodor Dostoevsky  BookEntity [id=2, title=War and Peace, author=Leo Tolstoy] --- Leo Tolstoy  ############################################################################  Kharkov Central PH  --------------------------  BookEntity [id=2, title=War and Peace, author=Leo Tolstoy] --- Leo Tolstoy  BookEntity [id=3, title=Pride and Prejudice, author=Jane Austen] --- Jane Austen |

В результате схема БД будет иметь следующий вид *Рис. 44*



*Рис. 44* Схема БД

Web Приложение

Реализуем функциональность просмотра данных нашей БД на страницах.

Для добавления данных в бд для вывода изменим класс StartupApplicationListener модуля Webapp

|  |  |
| --- | --- |
| Было | Стало |
| @Component **public class** StartupApplicationListener **implements** ApplicationRunner {  @Resource  **private** BookService **bookService**;   @Override  **public void** run(ApplicationArguments args) {  initFillDB();  }   **public void** initFillDB() {  BookEntity book1 = **bookService**.save(**new** BookEntity(**"The Brothers Karamazov"**, **"Fyodor Dostoevsky"**));  BookEntity book2 = **bookService**.save(**new** BookEntity(**"War and Peace"**, **"Leo Tolstoy"**));  BookEntity book3 = **bookService**.save(**new** BookEntity(**"Pride and Prejudice"**, **"Jane Austen"**));  } } | @Component **public class** StartupApplicationListener **implements** ApplicationRunner {  @Resource  **private** BookService **bookService**;  @Resource  **private** AuthorService **authorService**;  @Resource  **private** PublishingHouseService **publishingHouseService**;   @Override  **public void** run(ApplicationArguments args) {  initFillDB();  }   **public void** initFillDB() {  AuthorEntity author1 = **authorService**.save(**new** AuthorEntity(**"Fyodor"**, **"Dostoevsky"**));  AuthorEntity author2 = **authorService**.save(**new** AuthorEntity(**"Leo"**, **"Tolstoy"**));  AuthorEntity author3 = **authorService**.save(**new** AuthorEntity(**"Jane"**, **"Austen"**));  BookEntity book1 = **bookService**.save(**new** BookEntity(**"The Brothers Karamazov"**, author1));  BookEntity book2 = **bookService**.save(**new** BookEntity(**"War and Peace"**, author2));  BookEntity book3 = **bookService**.save(**new** BookEntity(**"Anna Karenina"**, author2));  BookEntity book4 = **bookService**.save(**new** BookEntity(**"Pride and Prejudice"**, author3));  **publishingHouseService**.save(**new** PublishingHouseEntity(**"Star PH"**, **new** HashSet<>(Arrays.*asList*(book1, book2, book4))));  **publishingHouseService**.save(**new** PublishingHouseEntity(**"Kharkov Central PH"**, **new** HashSet<>(Arrays.*asList*(book2, book3))));  } } |

Изменим существующий контроллер BookController

|  |  |
| --- | --- |
| Было | Стало |
| @Controller @RequestMapping(ControllerConstants.***BOOK\_CONTROLLER\_REQUEST\_MAPPING***) **public class** BookController {  **private static final** String ***VIEW\_NAME*** = **"bookList"**;  **private static final** String ***MODEL\_NAME*** = **"bookListModel"**;  @Resource  **private** BookService **bookService**;   @GetMapping  **public** ModelAndView get() {  **return new** ModelAndView(***VIEW\_NAME***, ***MODEL\_NAME***, **bookService**.findAll());  } } | @Controller @RequestMapping(ControllerConstants.***BOOK\_CONTROLLER\_REQUEST\_MAPPING***) **public class** BookController {  **private static final** String ***VIEW\_LIST\_NAME*** = **"bookList"**;  **private static final** String ***MODEL\_LIST\_NAME*** = **"bookListModel"**;  **private static final** String ***VIEW\_SINGLE\_NAME*** = **"book"**;  **private static final** String ***MODEL\_SINGLE\_NAME*** = **"bookModel"**;  @Resource  **private** BookService **bookService**;   @GetMapping  **public** ModelAndView get(@RequestParam(required = **false**) Integer id) {  **if** (**null** == id) {  **return new** ModelAndView(***VIEW\_LIST\_NAME***, ***MODEL\_LIST\_NAME***, **bookService**.findAll());  } **else** {  **return new** ModelAndView(***VIEW\_SINGLE\_NAME***, ***MODEL\_SINGLE\_NAME***, **bookService**.findById(id));  }  } } |

добавим адреса маппинга контроллеров в ControllerConstants

|  |  |
| --- | --- |
| Было | Стало |
| **public final class** ControllerConstants {  **public static final** String ***BOOK\_CONTROLLER\_REQUEST\_MAPPING*** = **"/bookController"**; } | **public final class** ControllerConstants {  **public static final** String ***BOOK\_CONTROLLER\_REQUEST\_MAPPING*** = **"/bookController"**;  **public static final** String ***AUTHOR\_CONTROLLER\_REQUEST\_MAPPING*** = **"/authorController"**;  **public static final** String ***PUBLISHING\_HOUSE\_CONTROLLER\_REQUEST\_MAPPING*** = **"/pubHouseController"**; } |

Создадим AuthorController

|  |
| --- |
| **package** edu.hneu.kpp.lab2.spring.webapp.controller;  **import** edu.hneu.kpp.lab2.spring.service.service.AuthorService; **import** org.springframework.stereotype.Controller; **import** org.springframework.web.bind.annotation.GetMapping; **import** org.springframework.web.bind.annotation.RequestMapping; **import** org.springframework.web.bind.annotation.RequestParam; **import** org.springframework.web.servlet.ModelAndView;  **import** javax.annotation.Resource;  @Controller @RequestMapping(ControllerConstants.***AUTHOR\_CONTROLLER\_REQUEST\_MAPPING***) **public class** AuthorController {  **private static final** String ***VIEW\_LIST\_NAME*** = **"authorList"**;  **private static final** String ***MODEL\_LIST\_NAME*** = **"authorListModel"**;  **private static final** String ***VIEW\_SINGLE\_NAME*** = **"author"**;  **private static final** String ***MODEL\_SINGLE\_NAME*** = **"authorModel"**;  @Resource  **private** AuthorService **authorService**;   @GetMapping  **public** ModelAndView get(@RequestParam(required = **false**) Integer id) {  **if** (**null** == id) {  **return new** ModelAndView(***VIEW\_LIST\_NAME***, ***MODEL\_LIST\_NAME***, **authorService**.findAll());  } **else** {  **return new** ModelAndView(***VIEW\_SINGLE\_NAME***, ***MODEL\_SINGLE\_NAME***, **authorService**.findById(id));  }  } } |

И PublishingHouseController

|  |
| --- |
| **package** edu.hneu.kpp.lab2.spring.webapp.controller;  **import** edu.hneu.kpp.lab2.spring.service.service.PublishingHouseService; **import** org.springframework.stereotype.Controller; **import** org.springframework.web.bind.annotation.GetMapping; **import** org.springframework.web.bind.annotation.RequestMapping; **import** org.springframework.web.bind.annotation.RequestParam; **import** org.springframework.web.servlet.ModelAndView;  **import** javax.annotation.Resource;  @Controller @RequestMapping(ControllerConstants.***PUBLISHING\_HOUSE\_CONTROLLER\_REQUEST\_MAPPING***) **public class** PublishingHouseController {  **private static final** String ***VIEW\_LIST\_NAME*** = **"pHouseList"**;  **private static final** String ***MODEL\_LIST\_NAME*** = **"pHouseListModel"**;  **private static final** String ***VIEW\_SINGLE\_NAME*** = **"pHouse"**;  **private static final** String ***MODEL\_SINGLE\_NAME*** = **"pHouseModel"**;   @Resource  **private** PublishingHouseService **publishingHouseService**;   @GetMapping  **public** ModelAndView get(@RequestParam(required = **false**) Integer id) {  **if** (**null** == id) {  **return new** ModelAndView(***VIEW\_LIST\_NAME***, ***MODEL\_LIST\_NAME***, **publishingHouseService**.findAll());  } **else** {  **return new** ModelAndView(***VIEW\_SINGLE\_NAME***, ***MODEL\_SINGLE\_NAME***, **publishingHouseService**.findById(id));  }  } } |

Представления

Вынесем общую для всех представлений часть в файл navMenu.html и потом подключим в представлениях где это необходимо

|  |
| --- |
| <!DOCTYPE **html**> <**html xmlns:th="http://www.thymeleaf.org"**> <**header th:fragment="navMenuHeader"**>  <**span**><**a th:href="@{/bookController}"**>Books</**a**></**span**>  <**span**><**a th:href="@{/authorController}"**>Authors</**a**></**span**>  <**span**><**a th:href="@{/pubHouseController}"**>Publishing Houses</**a**></**span**> </**header**> </**html**> |

Далее изменим существующее представление booklist.html

|  |  |
| --- | --- |
| Было | Стало |
| <!DOCTYPE **html**> <**html xmlns:th="http://www.thymeleaf.org"**> <**head**>  <**meta charset="UTF-8"**>  <**title**>Title</**title**> </**head**> <**body**> <**table**>  <**thead**>  <**tr**>  <**th**>Title</**th**>  <**th**>Author</**th**>  </**tr**>  </**thead**>  <**tbody**>  *<!--/\*@thymesVar id="bookListModel" type="java.util.List"\*/-->  <!--/\*@thymesVar id="book" type="edu.hneu.kpp.lab2.spring.dao.entity.BookEntity"\*/-->* <**tr th:each="book:${ bookListModel}"**>  <**td th:text="${book.title}"**></**td**>  <**td th:text="${book.author}"**></**td**>  </**tr**>  </**tbody**> </**table**> </**body**> </**html**> | <!DOCTYPE **html**> <**html xmlns:th="http://www.thymeleaf.org"**> <**head**>  <**meta charset="UTF-8"**>  <**title**>Title</**title**> </**head**> <**body**> <**header th:replace="navMenu :: navMenuHeader"**></**header**> <**table**>  <**thead**>  <**tr**>  <**th**>Title</**th**>  <**th**>Author</**th**>  <**th**>Publishing House</**th**>  </**tr**>  </**thead**>  <**tbody**>  *<!--/\*@thymesVar id="bookListModel" type="java.util.List"\*/-->  <!--/\*@thymesVar id="book" type="edu.hneu.kpp.lab2.spring.dao.entity.BookEntity"\*/-->* <**tr th:each="book:${bookListModel}"**>  <**td**>  <**a th:href="@{/bookController(id=${book.id})}" th:text="${book.title}"**></**a**>  </**td**>  <**td**>  <**a th:href="@{/authorController(id=${book.authorEntity.id})}" th:text="${book. authorEntity}"**></**a**>  </**td**>  <**td**>  <**th:block th:each="pHouse:${book.publishingHouseEntities}"**>  <**a th:href="@{/pubHouseController(id=${pHouse.id})}" th:text="${pHouse.name}"**></**a**>  </**th:block**>  </**td**>  </**tr**>  </**tbody**> </**table**> </**body**> </**html**> |

Добавим необходимые представления

book.html

|  |
| --- |
| <!DOCTYPE **html**> <**html xmlns:th="http://www.thymeleaf.org"**> <**head**>  <**meta charset="UTF-8"**>  <**title**>Title</**title**> </**head**> <**body**> <**header th:replace="navMenu :: navMenuHeader"**></**header**> <**table**>  <**thead**>  <**tr**>  <**th**>Title</**th**>  <**th**>Author</**th**>  <**th**>Publishing Houses</**th**>  </**tr**>  </**thead**>  <**tbody**>  *<!--/\*@thymesVar id="bookModel" type="edu.hneu.kpp.lab2.spring.dao.entity.BookEntity"\*/-->* <**tr**>  <**td th:text="${bookModel.title}"**></**td**>  <**td th:text="${bookModel.author}"**></**td**>  <**td**>  <**th:block th:each="pHouse:${bookModel.publishingHouseEntities}"**>  <**a th:href="@{/pubHouseController(id=${pHouse.id})}" th:text="${pHouse.name}"**></**a**>  </**th:block**>  </**td**>  </**tr**>  </**tbody**> </**table**> </**body**> </**html**> |

authorList.html

|  |
| --- |
| <!DOCTYPE **html**> <**html xmlns:th="http://www.thymeleaf.org"**> <**head**>  <**meta charset="UTF-8"**>  <**title**>Title</**title**> </**head**> <**body**> <**header th:replace="navMenu :: navMenuHeader"**></**header**> <**table**>  <**thead**>  <**tr**>  <**th**>Name</**th**>  <**th**>Surname</**th**>  <**th**>Books</**th**>  </**tr**>  </**thead**>  <**tbody**>  *<!--/\*@thymesVar id="authorListModel" type="java.util.List"\*/-->  <!--/\*@thymesVar id="author" type="edu.hneu.kpp.lab2.spring.dao.entity.AuthorEntity"\*/-->* <**tr th:each="author:${authorListModel}"**>  <**td**>  <**a th:href="@{/authorController(id=${author.id})}" th:text="${author.name}"**></**a**>  </**td**>  <**td**>  <**a th:href="@{/authorController(id=${author.id})}" th:text="${author.surname}"**></**a**>  </**td**>  <**td**>  <**th:block th:each="book:${author.bookEntities}"**>  <**a th:href="@{/bookController(id=${book.id})}" th:text="${book.title}"**></**a**>  </**th:block**>  </**td**>  </**tr**>  </**tbody**> </**table**> </**body**> </**html**> |

author.html

|  |
| --- |
| <!DOCTYPE **html**> <**html xmlns:th="http://www.thymeleaf.org"**> <**head**>  <**meta charset="UTF-8"**>  <**title**>Title</**title**> </**head**> <**body**> <**header th:replace="navMenu :: navMenuHeader"**></**header**> <**table**>  <**thead**>  <**tr**>  <**th**>Name</**th**>  <**th**>Surname</**th**>  <**th**>Books</**th**>  </**tr**>  </**thead**>  <**tbody**>  *<!--/\*@thymesVar id="authorModel" type="edu.hneu.kpp.lab2.spring.dao.entity.AuthorEntity"\*/-->* <**tr**>  <**td th:text="${authorModel.name}"**></**td**>  <**td th:text="${authorModel.surname}"**></**td**>  <**td**>  <**th:block th:each="book:${authorModel.bookEntities}"**>  <**a th:href="@{/bookController(id=${book.id})}" th:text="${book.title}"**></**a**>  </**th:block**>  </**td**>  </**tr**>  </**tbody**> </**table**> </**body**> </**html**> |

pHouseList.html

|  |
| --- |
| <!DOCTYPE **html**> <**html xmlns:th="http://www.thymeleaf.org"**> <**head**>  <**meta charset="UTF-8"**>  <**title**>Title</**title**> </**head**> <**body**> <**header th:replace="navMenu :: navMenuHeader"**></**header**> <**table**>  <**thead**>  <**tr**>  <**th**>Name</**th**>  <**th**>Surname</**th**>  <**th**>Books</**th**>  </**tr**>  </**thead**>  <**tbody**>  *<!--/\*@thymesVar id="pHouseListModel" type="java.util.List"\*/-->  <!--/\*@thymesVar id="pHouse" type="edu.hneu.kpp.lab2.spring.dao.entity.AuthorEntity"\*/-->* <**tr th:each="pHouse:${pHouseListModel}"**>  <**td**>  <**a th:href="@{/pubHouseController(id=${pHouse.id})}" th:text="${pHouse.name}"**></**a**>  </**td**>  <**td**>  </**td**>  <**td**>  <**th:block th:each="book:${pHouse.bookEntities}"**>  <**a th:href="@{/bookController(id=${book.id})}" th:text="${book.title}"**></**a**>  </**th:block**>  </**td**>  </**tr**>  </**tbody**> </**table**> </**body**> </**html**> |

pHouse.html

|  |
| --- |
| <!DOCTYPE **html**> <**html xmlns:th="http://www.thymeleaf.org"**> <**head**>  <**meta charset="UTF-8"**>  <**title**>Title</**title**> </**head**> <**body**> <**header th:replace="navMenu :: navMenuHeader"**></**header**> <**table**>  <**thead**>  <**tr**>  <**th**>Name</**th**>  <**th**>Surname</**th**>  <**th**>Books</**th**>  </**tr**>  </**thead**>  <**tbody**>  *<!--/\*@thymesVar id="pHouseModel" type="edu.hneu.kpp.lab2.spring.dao.entity.PublishingHouseEntity"\*/-->* <**tr**>  <**td th:text="${pHouseModel.name}"**></**td**>  <**td**>  <**th:block th:each="book:${pHouseModel.bookEntities}"**>  <**a th:href="@{/bookController(id=${book.id})}" th:text="${book.title}"**></**a**>  </**th:block**>  </**td**>  </**tr**>  </**tbody**> </**table**> </**body**> </**html**> |

Запустим приложение и по адресам

<http://127.0.0.1:8080/bookController>

<http://127.0.0.1:8080/authorController>

<http://127.0.0.1:8080/pubHouseController>

увидим перечень соответствующих данных, по ссылкам на этих страницах можно открыть каждую сущность отдельно и совершить переход на связанные записи.

REST

// тут текст про REST

Создадим REST endpoint двумя способами

Первый – RestController

Создаем еще один контроллер BookRestController

|  |
| --- |
| **package** edu.hneu.kpp.lab2.spring.webapp.controller;  **import** edu.hneu.kpp.lab2.spring.service.service.BookService; **import** org.springframework.web.bind.annotation.GetMapping; **import** org.springframework.web.bind.annotation.RequestMapping; **import** org.springframework.web.bind.annotation.RequestParam; **import** org.springframework.web.bind.annotation.RestController;  **import** javax.annotation.Resource;  @RestController @RequestMapping(ControllerConstants.***BOOK\_REST\_CONTROLLER\_REQUEST\_MAPPING***) **public class** BookRestController {  @Resource  **private** BookService **bookService**;   @GetMapping  **public** Object getById(@RequestParam(required = **false**) Integer id) {  **if** (**null** == id) {  **return bookService**.findAll();  } **else** {  **return bookService**.findById(id);  }  } } |

Так же добавим строку в ControllerConstants

|  |  |
| --- | --- |
| Было | Стало |
| **public final class** ControllerConstants {  **public static final** String ***BOOK\_CONTROLLER\_REQUEST\_MAPPING*** = **"/bookController"**;  **public static final** String ***AUTHOR\_CONTROLLER\_REQUEST\_MAPPING*** = **"/authorController"**;  **public static final** String ***PUBLISHING\_HOUSE\_CONTROLLER\_REQUEST\_MAPPING*** = **"/pubHouseController"**; } | **public final class** ControllerConstants {  **public static final** String ***BOOK\_CONTROLLER\_REQUEST\_MAPPING*** = **"/bookController"**;  **public static final** String ***AUTHOR\_CONTROLLER\_REQUEST\_MAPPING*** = **"/authorController"**;  **public static final** String ***PUBLISHING\_HOUSE\_CONTROLLER\_REQUEST\_MAPPING*** = **"/pubHouseController"**;  **public static final** String ***BOOK\_REST\_CONTROLLER\_REQUEST\_MAPPING*** = **"/bookRest"**; } |

Дополнительно внесем изменения в сущности, т.к. сейчас при приобразовании BookEntity в json мы будем получать AuthorEntity у которого будем получать BookEntities у которых Author итд. получим бесконечную рекурсию и приложение завершится с ошибкой StackOverflowError . Чтобы этого не произошло, необходимо ограничить преобразовываемые данные.

Для этого добавим зависимость на библиотеку с необходимыми аннотациями в build.gradle модуля Repository

|  |  |
| --- | --- |
| Было | Стало |
| dependencies {  compile(**"org.springframework.boot:spring-boot-starter-data-jpa:**${springBootVersion}**"**)  compile(**"org.springframework.boot:spring-boot-configuration-processor:**${springBootVersion}**"**)  compile(**"org.hibernate:hibernate-entitymanager:**${hibernateEntitymanagerVersion}**"**)  compile(**"mysql:mysql-connector-java:**${mysqlConnectorJavaVersion}**"**)  runtime(**"org.springframework.boot:spring-boot-devtools:**${springBootVersion}**"**)  testCompile(**"org.springframework.boot:spring-boot-starter-test:**${springBootVersion}**"**) } | dependencies {  compile(**"org.springframework.boot:spring-boot-starter-data-jpa:**${springBootVersion}**"**)  compile(**"org.springframework.boot:spring-boot-configuration-processor:**${springBootVersion}**"**)  compile(**"org.hibernate:hibernate-entitymanager:**${hibernateEntitymanagerVersion}**"**)  compile(**"mysql:mysql-connector-java:**${mysqlConnectorJavaVersion}**"**)  compile(**"com.fasterxml.jackson.core:jackson-annotations:2.9.5"**)  runtime(**"org.springframework.boot:spring-boot-devtools:**${springBootVersion}**"**)  testCompile(**"org.springframework.boot:spring-boot-starter-test:**${springBootVersion}**"**) } |

Теперь добавим необходимые аннотации в сущности

BookEntity

|  |  |
| --- | --- |
| Было | Стало |
| @Entity @Table(name = **"book"**) **public class** BookEntity {  @Id  @GeneratedValue(strategy = GenerationType.***IDENTITY***)  @Column(name = **"id"**, unique = **true**, nullable = **false**)  **private int id**;  @Column(name = **"title"**)  **private** String **title**;  @ManyToOne  @JoinColumn(name = **"author\_id"**, referencedColumnName = **"id"**, nullable = **false**)  **private** AuthorEntity **authorEntity**;  @JsonManagedReference  @JsonIdentityInfo(  generator = ObjectIdGenerators.PropertyGenerator.**class**,  property = **"id"**)  @ManyToMany(mappedBy = **"bookEntities"**, fetch = FetchType.***EAGER***)  **private** Set<PublishingHouseEntity> **publishingHouseEntities**;  **//остальная часть класса не изменилась** | @Entity @Table(name = **"book"**) **public class** BookEntity {  @Id  @GeneratedValue(strategy = GenerationType.***IDENTITY***)  @Column(name = **"id"**, unique = **true**, nullable = **false**)  **private int id**;  @Column(name = **"title"**)  **private** String **title**;  @ManyToOne  @JsonBackReference  @JoinColumn(name = **"author\_id"**, referencedColumnName = **"id"**, nullable = **false**)  **private** AuthorEntity **authorEntity**;  @JsonManagedReference  @JsonIdentityInfo(  generator = ObjectIdGenerators.PropertyGenerator.**class**,  property = **"id"**)  @ManyToMany(mappedBy = **"bookEntities"**, fetch = FetchType.***EAGER***)  **private** Set<PublishingHouseEntity> **publishingHouseEntities**;  **//остальная часть класса не изменилась** |

AuthorEntity

|  |  |
| --- | --- |
| Было | Стало |
| @Entity @Table(name = **"author"**) **public class** AuthorEntity {  @Id  @GeneratedValue(strategy = GenerationType.***IDENTITY***)  @Column(name = **"id"**, unique = **true**, nullable = **false**)  **private int id**;  @Column(name = **"name"**, nullable = **false**)  **private** String **name**;  @Column(name = **"surname"**)  **private** String **surname**;  @OneToMany(fetch = FetchType.***EAGER***, mappedBy = **"authorEntity"**, cascade = CascadeType.***ALL***)  **private** Set<BookEntity> **bookEntities**;  **//остальная часть класса не изменилась** | @Entity @Table(name = **"author"**) **public class** AuthorEntity {  @Id  @GeneratedValue(strategy = GenerationType.***IDENTITY***)  @Column(name = **"id"**, unique = **true**, nullable = **false**)  **private int id**;  @Column(name = **"name"**, nullable = **false**)  **private** String **name**;  @Column(name = **"surname"**)  **private** String **surname**;  @JsonManagedReference  @OneToMany(fetch = FetchType.***EAGER***, mappedBy = **"authorEntity"**, cascade = CascadeType.***ALL***)  **private** Set<BookEntity> **bookEntities**;  **//остальная часть класса не изменилась** |

PublishingHouseEntity

|  |  |
| --- | --- |
| Было | Стало |
| @Entity @Table(name = **"publishing\_house"**) **public class** PublishingHouseEntity {  @Id  @GeneratedValue(strategy = GenerationType.***IDENTITY***)  @Column(name = **"id"**, unique = **true**, nullable = **false**)  **private int id**;  @Column(name = **"name"**, nullable = **false**)  **private** String **name**;  @ManyToMany(fetch = FetchType.***EAGER***)  @JoinTable(name = **"book2publishing\_house"**,  joinColumns = @JoinColumn(name = **"book\_id"**, referencedColumnName = **"id"**),  inverseJoinColumns = @JoinColumn(name = **"publishing\_house\_id"**, referencedColumnName = **"id"**))  **private** Set<BookEntity> **bookEntities**;  **//остальная часть класса не изменилась** | @Entity @Table(name = **"publishing\_house"**) **public class** PublishingHouseEntity {  @Id  @GeneratedValue(strategy = GenerationType.***IDENTITY***)  @Column(name = **"id"**, unique = **true**, nullable = **false**)  **private int id**;  @Column(name = **"name"**, nullable = **false**)  **private** String **name**;  @JsonManagedReference  @ManyToMany(fetch = FetchType.***EAGER***)  @JoinTable(name = **"book2publishing\_house"**,  joinColumns = @JoinColumn(name = **"book\_id"**, referencedColumnName = **"id"**),  inverseJoinColumns = @JoinColumn(name = **"publishing\_house\_id"**, referencedColumnName = **"id"**))  **private** Set<BookEntity> **bookEntities**;  **//остальная часть класса не изменилась** |

По адресу <http://127.0.0.1:8080/bookRest> наблюдаем json объект - массив книг

|  |
| --- |
| [{"id":1,"title":"The Brothers Karamazov","publishingHouseEntities":[{"id":1,"name":"Star PH","bookEntities":[{"id":1,"title":"The Brothers Karamazov","publishingHouseEntities":[1]},{"id":2,"title":"War and Peace","publishingHouseEntities":[1,{"id":2,"name":"Kharkov Central PH","bookEntities":[{"id":3,"title":"Anna Karenina","publishingHouseEntities":[2]},{"id":2,"title":"War and Peace","publishingHouseEntities":[1,2]}]}]},{"id":4,"title":"Pride and Prejudice","publishingHouseEntities":[1]}]}]},{"id":2,"title":"War and Peace","publishingHouseEntities":[1,2]},{"id":3,"title":"Anna Karenina","publishingHouseEntities":[2]},{"id":4,"title":"Pride and Prejudice","publishingHouseEntities":[1]}] |

Второй - ResponceBody

Если у нас есть необходимость сделать REST-endpoint но необходимость в отдельном контроллере отсутствует то можно реализовать такой endpoint в обычном контроллере. Попробуем на примере AuthorController

|  |  |
| --- | --- |
| Было | Стало |
| @Controller @RequestMapping(ControllerConstants.***AUTHOR\_CONTROLLER\_REQUEST\_MAPPING***) **public class** AuthorController {  **private static final** String ***VIEW\_LIST\_NAME*** = **"authorList"**;  **private static final** String ***MODEL\_LIST\_NAME*** = **"authorListModel"**;  **private static final** String ***VIEW\_SINGLE\_NAME*** = **"author"**;  **private static final** String ***MODEL\_SINGLE\_NAME*** = **"authorModel"**;  @Resource  **private** AuthorService **authorService**;   @GetMapping  **public** ModelAndView get(@RequestParam(required = **false**) Integer id) {  **if** (**null** == id) {  **return new** ModelAndView(***VIEW\_LIST\_NAME***, ***MODEL\_LIST\_NAME***, **authorService**.findAll());  } **else** {  **return new** ModelAndView(***VIEW\_SINGLE\_NAME***, ***MODEL\_SINGLE\_NAME***, **authorService**.findById(id));  }  }  } | @Controller @RequestMapping(ControllerConstants.***AUTHOR\_CONTROLLER\_REQUEST\_MAPPING***) **public class** AuthorController {  **private static final** String ***VIEW\_LIST\_NAME*** = **"authorList"**;  **private static final** String ***MODEL\_LIST\_NAME*** = **"authorListModel"**;  **private static final** String ***VIEW\_SINGLE\_NAME*** = **"author"**;  **private static final** String ***MODEL\_SINGLE\_NAME*** = **"authorModel"**;  @Resource  **private** AuthorService **authorService**;   @GetMapping  **public** ModelAndView get(@RequestParam(required = **false**) Integer id) {  **if** (**null** == id) {  **return new** ModelAndView(***VIEW\_LIST\_NAME***, ***MODEL\_LIST\_NAME***, **authorService**.findAll());  } **else** {  **return new** ModelAndView(***VIEW\_SINGLE\_NAME***, ***MODEL\_SINGLE\_NAME***, **authorService**.findById(id));  }  }   @GetMapping(**"rest"**)  @ResponseBody  **public** Object getRest(@RequestParam(required = **false**) Integer id) {  **if** (**null** == id) {  **return authorService**.findAll();  } **else** {  **return authorService**.findById(id);  }  } } |

// ОПИСАНИЕ АННОТАЦИИ ResponceBody

По адресу <http://127.0.0.1:8080/authorController/rest> наблюдаем json объект - массив Авторов