Spring Boot guide:

<https://o7planning.org/ru/11267/spring-boot-tutorial-for-beginners>

<https://o7planning.org/ru/11669/spring-boot>

**Spring Boo**t  является  проектом на уровне  **IO Execution** (уровень выполнения) **IO Spring Framework**   
**Spring Boot**это следуюший шаг Spring, чтобы сделать его легче в настройке и развитии  приложений.

С **Spring Boot** конфигурации Spring  минимизируются максимально. **Spring Boot** поддерживает встроенный контейнер (embedded containers), который  позволяет веб-приложениям работать независимо и без необходимости применения на **Web Server**.  
Вы можете использовать **spring Boot** чтобы создать Java Web приложение, работающее через команду line " **Java -jar"** или экспортировать War файл для применения  на Web Server как обычно.

**Разбор  pom.xml файла для Spring Boot:**

**Spring-boot-starter-parent** готовый проект в **Spring Boot**. Стандартные зависимые библиотеки объявлены в

**spring-boot-starter-paren**t, ваш проект только  унаследовал их. Нужно просто объявить **<parent>** в файле **pom.xml**вашего проекта:

<!-- <https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-parent> -->

<parent>

    <groupId>org.springframework.boot</groupId>

    <artifactId>spring-boot-starter-parent</artifactId>

    <version>2.0.0.RELEASE</version>

    <relativePath/> <!-- lookup parent from repository -->

</parent>

### spring-boot-starter-web

Другие “ **Starter”** только обеспечивают зависимости, которые  вам могут понадобятся при развитии конкретного типа приложения. Например, при развитии веб-приложения, вам нужна зависимость **spring-boot-starter-web:**

<dependencies>

   <dependency>

       <groupId>org.springframework.boot</groupId>

       <artifactId>spring-boot-starter-web</artifactId>

   </dependency>

   <dependency>

       <groupId>org.springframework.boot</groupId>

       <artifactId>spring-boot-starter-test</artifactId>

       <scope>test</scope>

   </dependency>

</dependencies>

**spring-boot-starter-test**

Импортирует библиотеки необходимые для тестирования:

* [JUnit](http://junit.org/): The de-facto standard for unit testing Java applications.
* [Spring Test](https://docs.spring.io/spring/docs/5.0.8.RELEASE/spring-framework-reference/testing.html#integration-testing) & Spring Boot Test: Utilities and integration test support for Spring Boot applications.
* [AssertJ](https://joel-costigliola.github.io/assertj/): A fluent assertion library.
* [Hamcrest](http://hamcrest.org/JavaHamcrest/): A library of matcher objects (also known as constraints or predicates).
* [Mockito](http://mockito.org/): A Java mocking framework.
* [JSONassert](https://github.com/skyscreamer/JSONassert): An assertion library for JSON.
* [JsonPath](https://github.com/jayway/JsonPath): XPath for JSON.

**Spring-boot-maven-plugin** является плагином, который обеспечивает  необходимые библиотеки, помогает вашему проекту работать напрямю без необходимости применять на Web Server. Это помогает создать file jar, который может быть выполнен (Executable):

<plugins>

    <plugin>

        <groupId>org.springframework.boot</groupId>

        <artifactId>spring-boot-maven-plugin</artifactId>

    </plugin>

</plugins>

Общие свойства **application.properties**

<https://o7planning.org/ru/11685/spring-boot-common-properties>

**Spring Boot annotations:**

* ***@Entity*** — JPA-аннотация уровня типа, обозначающая, что данный класс проецируется на таблицу в базе данных.
* ***@Component*** - это общий стереотип для компонента, управляемого Spring. Он превращает класс в компонент Spring в момент автоматического сканирования. Классы, помеченные этой аннотацией, считаются как кандидаты для автоматического обнаружения при использовании конфигурации на основе аннотаций и сканирования пути к классам. @Repository, @Service, and @Controller – есть специализациями из @Component для более специфических случаев использования.
* ***@Service*** - Аннотация, объявляющая, что этот класс представляет собой сервис – компонент сервис-слоя. Сервис является подтипом класса @Component. Использование данной аннотации позволит искать бины-сервисы автоматически.
* ***@Repository*** - Аннотация показывает, что класс функционирует как репозиторий и требует наличия трансляции исключений. Преимуществом трансляции исключений является то, что слой сервиса будет иметь дело с общей иерархией исключений от Spring (DataAccessException) вне зависимости от используемых технологий доступа к данным в слое данных.
* ***@Controller*** - (Слой представления) Аннотация для маркировки java класса, как класса контроллера. Данный класс представляет собой компонент, похожий на обычный сервлет (HttpServlet) (работающий с объектами HttpServletRequest и HttpServletResponse), но с расширенными возможностями от Spring Framework.
* ***@Configuration*** - указывает, что класс является классом конфигурации, который может содержать определения bean.
* ***@ResponseBody*** - Аннотация показывает что данный метод может возвращать кастомный объект в виде xml, json...
* ***@RestController*** - Аннотация аккумулирует поведение двух аннотаций @Controller и @ResponseBody.
* ***@RequestMapping*** - Аннотация используется для маппинга урл-адреса запроса на указанный метод или класс. Можно указывать конкретный HTTP-метод, который будет обрабатываться (GET/POST), передавать параметры запроса.
* ***@PathVariable*** - Аннотация, которая показывает, что параметр метода должен быть связан с переменной из урл-адреса.
* ***@Bean*** - указывает, что метод создает Bean, которым управляет Spring.
* ***@Scope*** - Аннотация для установки области жизни бина:
  + *singleton* (только один экземпляр бина создается для IoC контейнера; значение по умолчанию)
  + *prototype* (создается новый экземпляр бина когда приходит запрос на его создание)
  + *request* (один экземпляр бина для каждого HTTP запроса)
  + *session* (один экземпляр бина для каждой сессии)
  + *globalSession* (один экземпляр бина для каждой глобальной сессии)
* ***@PostConstruct*** - Аннотация для метода, который будет вызван после вызова конструктора бина.
* ***@PreDestroy*** - Аннотация для метода, который будет вызван перед уничтожением бина.
* ***@Profile*** - Аннотация для создания профилей конфигурации проекта. Может применяться как к бинам так и к конфигурационным классам.
* ***@Autowired*** - Аннотация позволяет автоматически установить значение поля.
* ***@SpringBootApplication*** - включает сканирование компонентов, автоконфигурацию и показывает разным компонентам Spring (например, интеграционным тестам), что это **Spring Boot** приложение. По умолчанию он в себя включает @SpringBootConfiguration, @EnableAutoConfiguration и @ComponentScan аннотации.
* ***@ModelAttribute*** - Аннотация, связывающая параметр метода или возвращаемое значение метода с атрибутом модели, которая будет использоваться при выводе jsp-страницы.
* There are also Hibernate @Entity, @Table, @Id, and @GeneratedValue annotations in the example.

Для взаимодействия таблицы данных, Hibernate  и Spring Boot, создается POJO class

@Entity *помечает класс как сущность и сопоставляет с таблицей базы данных*   
@Table(name = **"data\_base"**)*указывается имя таблицы для сопоставления*  
**public class** DataBaseConnect {

*здесь указываются идентификаторы сущностей таблицы:*  
 @Id означает, что помеченное свойство является первичным ключом  
 @GeneratedValue (strategy = GenerationType.***AUTO***) *устанавливает стратегию генерации для сущности,*   
 @Column (name = **"id"**) *по-умолчанию используется стратегия****IDENTITY****.*  
 **private int id**;  
 @Column (name = **"firstName"**)  
 **private** String **firstName**;  
  
 **public** DataBaseConnect(){}*обязательный пустой конструктор*  
  
 DataBaseConnect(String firstName){ **firstName** = firstName; }

*Геттеры и сеттеры* **public int** getId() { **id**; }  
 **public void** setId(**int** id) { **this**.**id** = id;   
 **public** String getFirstName() {**return firstName**;   
 **public void** setFirstName(String firstName) {.**firstName** = firstName;   
  
 *@Override не обязательный метод toString()* **public** String *toString()* {**return id**+ "," +**firstName**;} }

# Lombok

http://jnb.ociweb.com/jnb/jnbJan2010.html

Это вспомогательная библиотека java, для минимизации рутинного кода вроде get/set-методов и конструкторов.

Пример зависимости:

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<version>1.18.2</version>

<scope>provided</scope>

</dependency>

* [***@Data***](https://projectlombok.org/features/Data) — добавляет все рутинные методы классов сущностей. Содержит в себе аннотации @ToString, @EqualsAndHashCode, @Getter and @Setter.
* [***@NoArgsConstructor***](https://projectlombok.org/features/constructor) добавляет конструктор без аргументов.
* ***@AllArgsConstructor*** добавляет конструктор со всеми аргументами.
* ***@Getter***and***@Setter*** annotations generate a getter and setter for a field, respectively. Следует отметить, что если класс, к которому принадлежит аннотированное поле, содержит метод с тем же именем, что и генерируемый геттер или сеттер, независимо от параметров или типов возврата, соответствующий метод не будет сгенерирован.
* ***@NonNull*** аннотация используется для указания необходимости проверки нуля с быстрой ошибкой на соответствующем элементе. При размещении в поле, для которого Lombok генерирует метод setter, генерируется нулевая проверка, которая приведет к исключению NullPointerException. Кроме того, если Lombok генерирует конструктор для класса-владельца, тогда поле будет добавлено в подпись конструктора, а нулевая проверка будет включена в сгенерированный код конструктора.
* ***@ToString*** Эта аннотация генерирует реализацию метода toString. По умолчанию любые нестатические поля будут включены в вывод метода в парах имя-значение. При желании включение имен свойств в выход может быть подавлено путем установки параметра аннотации includeFieldNames в значение false. Конкретные поля можно исключить из вывода сгенерированного метода, включив их имена полей в параметр exclude. В качестве альтернативы параметр может использоваться для перечисления только тех полей, которые желательны на выходе. Результат метода toString суперкласса также можно включить, установив для параметра callSuper значение true.

# Thymeleaf

**https://o7planning.org/ru/11545/spring-boot-and-thymeleaf-tutorial**

**https://alexkosarev.name/2017/08/08/thymeleaf-template-engine/**

**Thymeleaf**является **Java XML/XHTML/HTML5 Template Engine,** который может работать с Web и не Web средой. Он больше подходит при использовании для сервиса **XHTML/HTML5** на уровне **View**(View Layer) приложения **Web** основываясь на структуре  **MVC**. Он может обрабатывать любой файл  **XML**, даже среды offline. Он поддерживает полностью для интеракции с **Spring Framework**.

Шаблонные файлы (Template file) **Thymeleaf**на самом деле является обычными текстовым файлом, имеющим формат **XML/XHTML/HTML5**. **Thymeleaf Engine** (Машина Thymeleaf) прочитает шаблонный файл (template file) и комбинирует с объектами **Java**, чтобы генерировать (generate) другой документ.