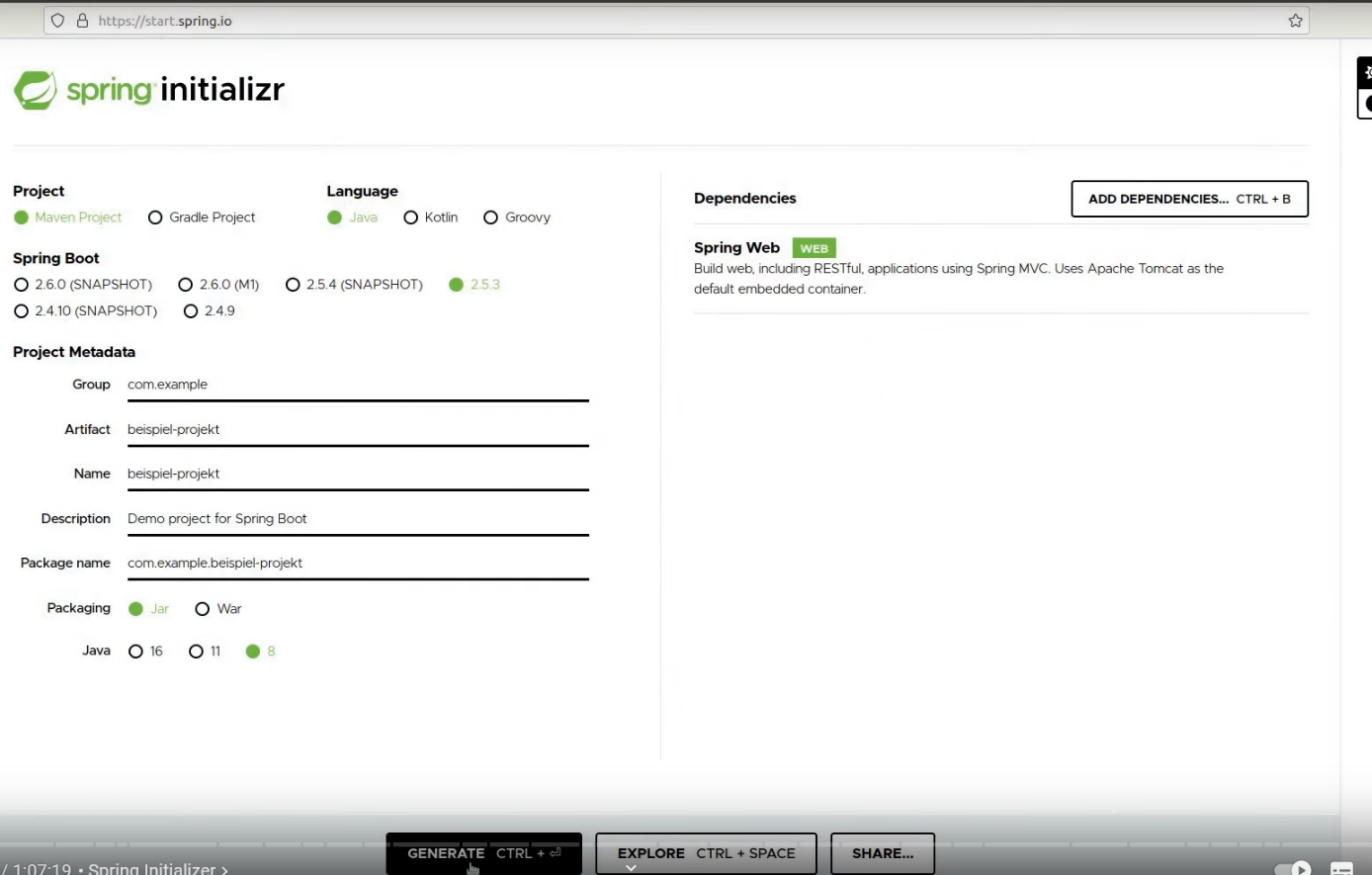
Ziele von Spring Boot:

Abhängigkeiten vermeiden, so dass die Anwendung leichter abegeändert werden kann. Z.b. Bei Klassen. Es gibt sozusagen nicht nur einen Motor, sondern mehrere. Falls einer ausfällt, läuft der andere weiter

Zweitens: Boilerplate vermeiden ( da nochmal nachfragen anhand eines Beispiels was das ist)

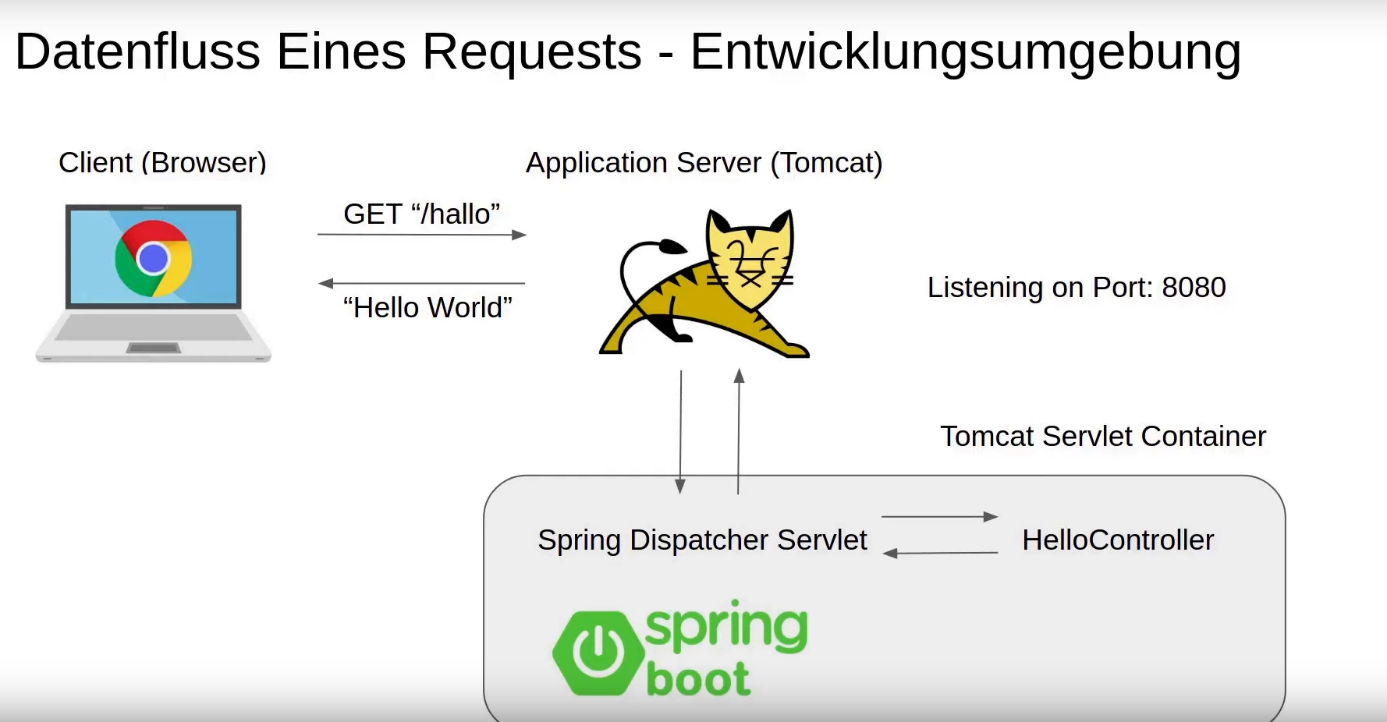
Auf der Website: <https://start.spring.io>

Kann man sein Projekt erstellen. 

POM.XML = dort werden die bibliotheken für das projekt hinzugefügt

Ein Bild, das Text, Screenshot, Software, Multimedia-Software enthält.

Automatisch generierte BeschreibungAls nächstes gibt es Controller. Der Controller erkennt verschiedene Pfade in der Url und kann darauf zuAntworten ( also daten zurückzusenden)



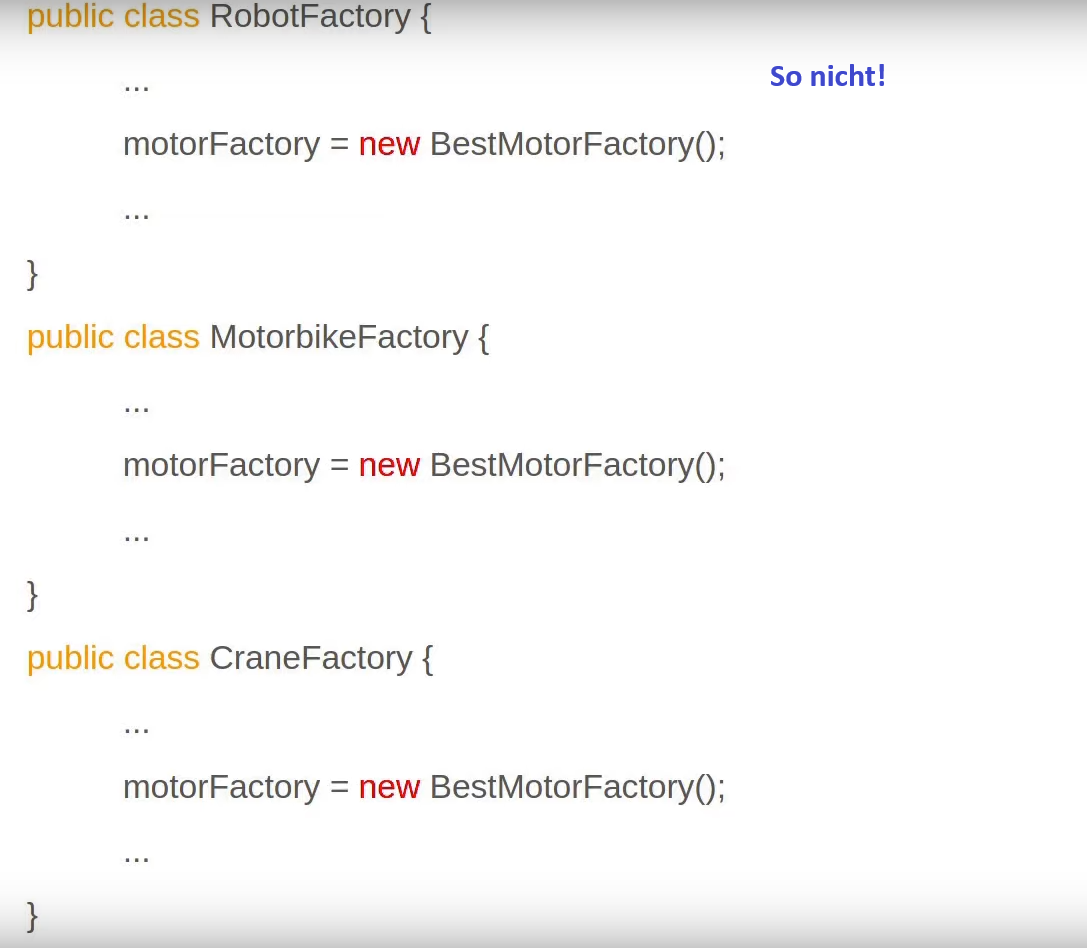
Der Begriff Java Servlet erklärt: Es sind Klassen die HttpRequest annehmen und bearbeiten in Java.

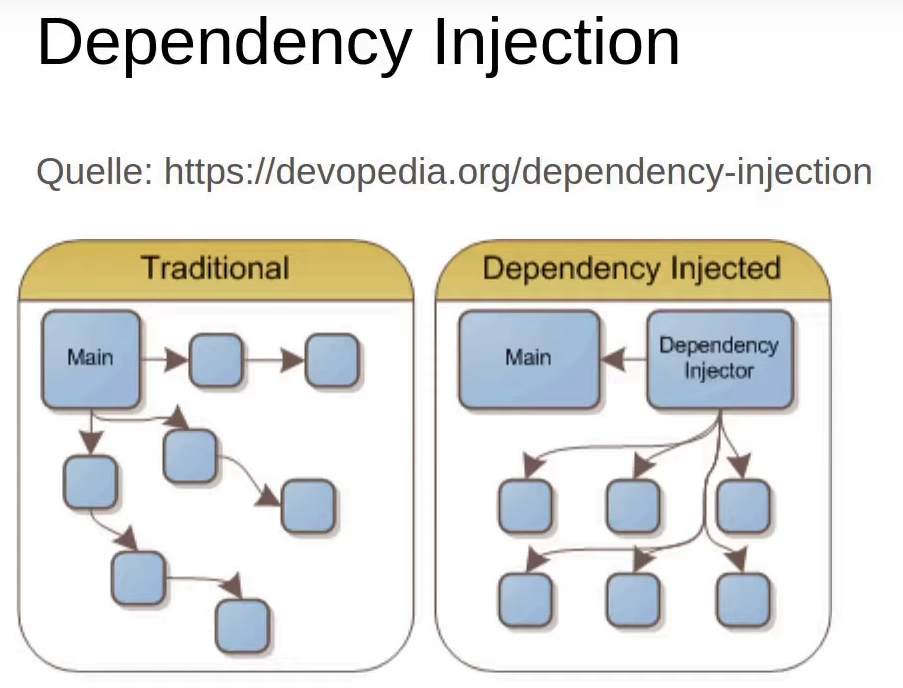
Was ist für ein Java Servlet nötig? Es ist ein Application Server notwendig z.B. Tomcat

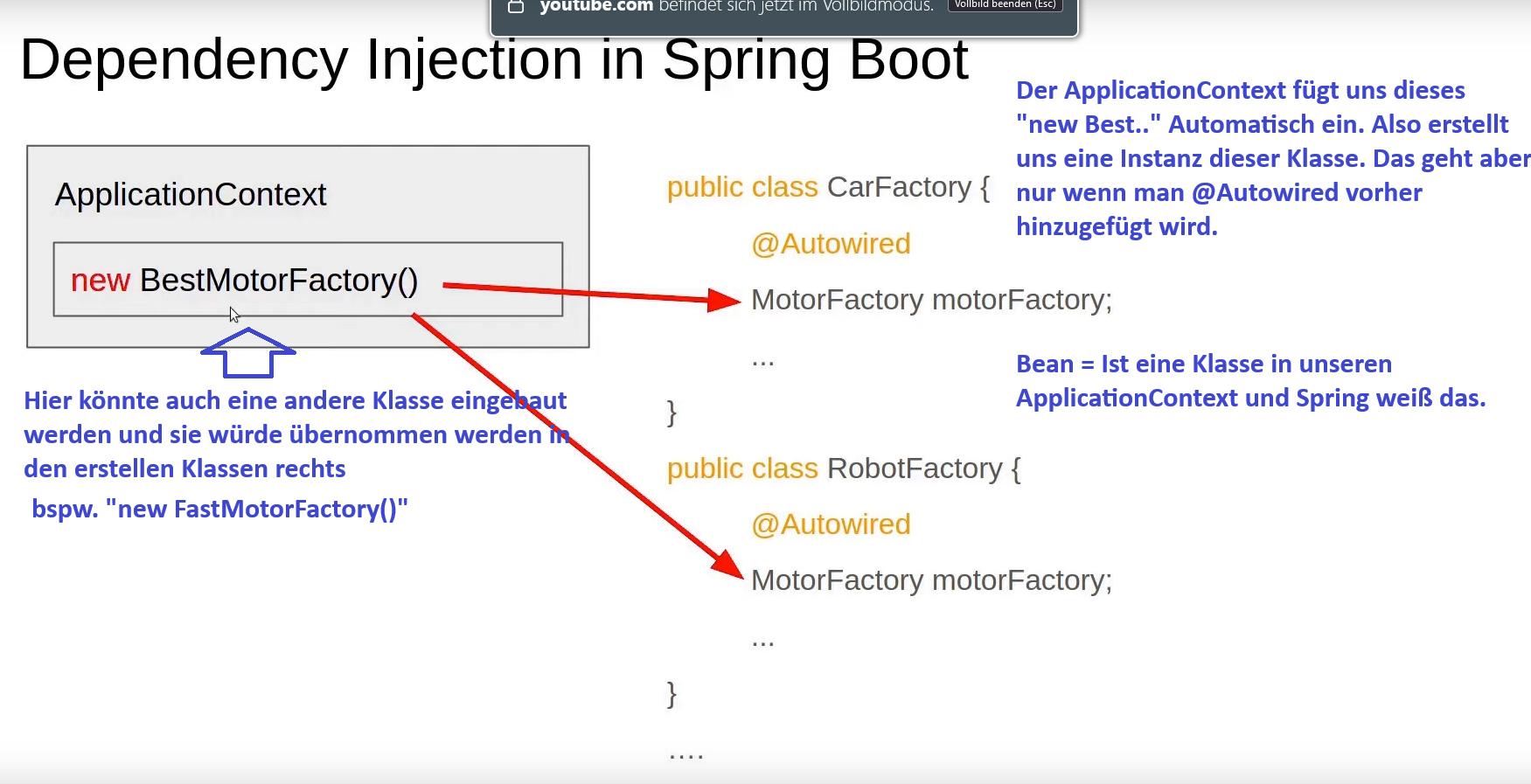
Spring Anwendungen nutzen die Klasse DispatcherServlet um eingehende http anfragen zu behandeln

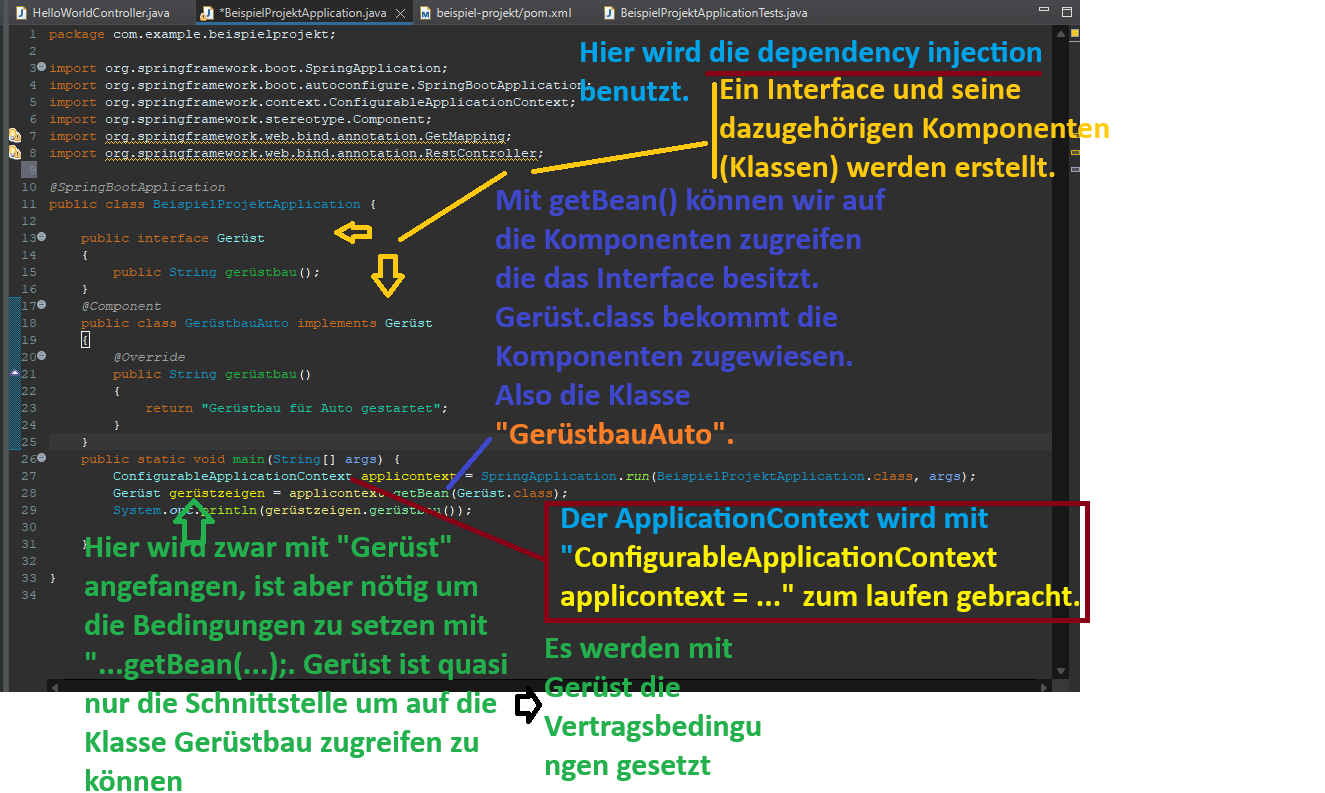
**Dependency Injections:**

Dependency = Abhängigkeit ; Wollen Abhängigkeiten vermeiden als Entwickler



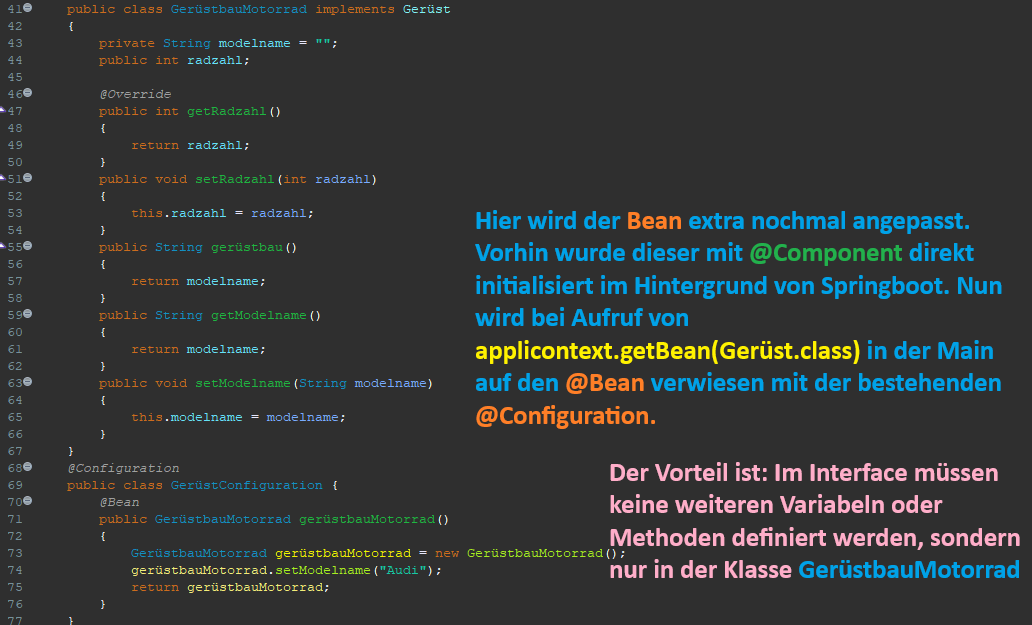


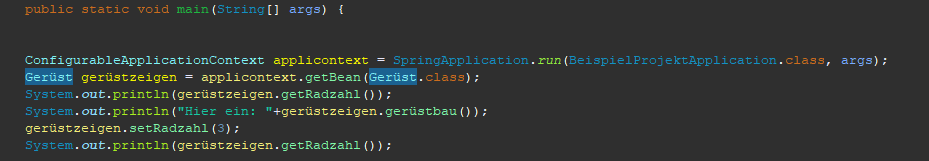


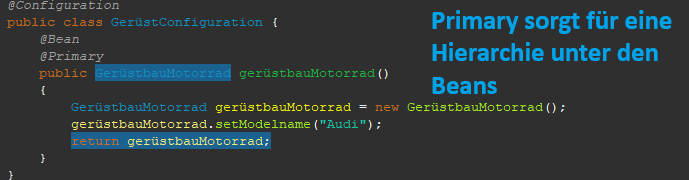
Dieses einsetzen der Klassen oder auch instanzieren mit Springboot nennt sich auch „Inversion of Control“. Alllokation = „zuweisen von“ hier in dem fall das überlassen von Klassen einer dritten Partei. (dem Spring Application Context). Vorteil davon ist: weniger Code, weniger abhängigkeiten, testen wird vereinfacht.

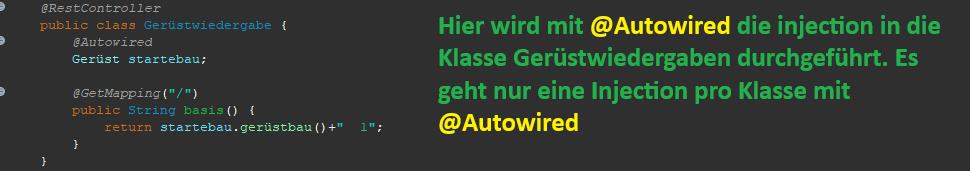
**Klein Randnotiz zu static und final**

Static übergibt z.b der klasse oder methode immer den selben Wert, ist aber abänderbar im weiterenverlauf. Final ist nicht abänderbar und eine konstante

**Anpassen von Beans ohne @Component sondern mit @Configuration**



@Primary

@Autowired

@Qualifier : Überschreibt @Primary

Das muss in das pom.xml

**JPA**

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

<dependency>

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

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

<version>3.1.5</version>

</dependency>

**JDBC**

<!-- https://mvnrepository.com/artifact/org.postgresql/postgresql -->

<dependency>

<groupId>org.postgresql</groupId>

<artifactId>postgresql</artifactId>

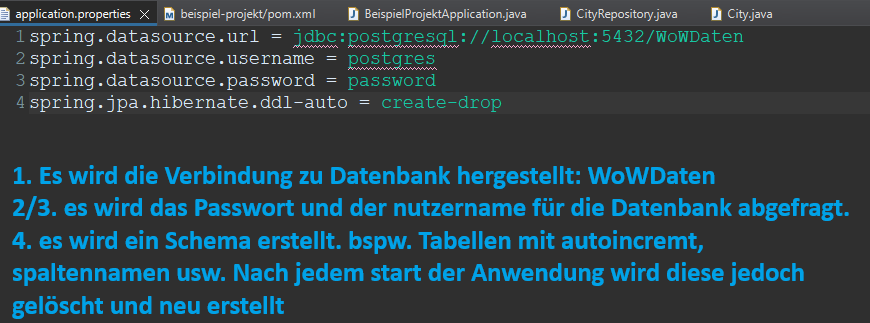
<version>42.6.0</version>

</dependency>

**Hier findet man die verschiedenen Versionen für JDPC und JPA:**

<https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-data-jpa/3.1.5>

<https://mvnrepository.com/artifact/org.postgresql/postgresql/42.6.0>

**Herstellen der Datenbankverbindung in Java**

Es gibt außerdem noch weitere Möglichkeiten statt create- drop.

1. **validate**: Hibernate validiert das Datenbankschema, gibt jedoch keine DDL-Anweisungen aus und bringt keine Änderungen an der Datenbankstruktur vor.
2. **update**: Hibernate aktualisiert das Datenbankschema, indem es Änderungen an vorhandenen Tabellen vornimmt. Es erstellt jedoch keine neuen Tabellen oder löscht bestehende.
3. **create**: Hibernate erstellt das Datenbankschema neu, indem es alle Tabellen gemäß den Entitätsklassen erstellt. Es löscht jedoch bestehende Tabellen, bevor es neue erstellt.
4. **none**: Hibernate führt keine automatischen Änderungen am Datenbankschema durch. Die Datenbankstruktur muss manuell erstellt und verwaltet werden.
5. **create-drop**: Hibernate erstellt das Schema beim Start der Anwendung und löscht es beim Beenden der Anwendung.
6. **validate-drop**: Hibernate validiert das Datenbankschema und löscht es beim Beenden der Anwendung.
7. **create-only**: Hibernate erstellt das Datenbankschema beim Start, führt jedoch keine Updates durch und löscht es nicht beim Beenden der Anwendung.
8. **update-only**: Hibernate führt nur Aktualisierungen am Schema durch, erstellt jedoch keine neuen Tabellen und löscht keine vorhandenen Tabellen.