FULLSTACKDEVELOPMENT MIT SPRING BOOT

Fortgeschrittene Software-Entwicklung

Name: Julian Meilinger

Datum: 20.10.2021

# Einleitung

Spring Boot ist ein Open-Source-Micro-Framework. Es bietet Java-Entwicklern eine Plattform für den Einstieg in eine automatisch konfigurierbare Spring-Anwendung in Produktionsqualität.

Die Merkmale von Spring Boot lassen sich wie folgt zusammenfassen:

* direktes Einbetten von Webserver-/Container-Anwendungen wie Apache Tomcat oder Jetty möglich, wodurch kein Einsatz von WAR-Dateien (Web Application Archive) erforderlich ist
* vereinfachte Maven-Konfiguration dank „Starter“-POMs (Project Object Models)
* automatische Spring-Konfiguration, wann immer dies möglich ist
* Bereitstellung nichtfunktionaler Features wie Metriken oder ausgelagerter Konfigurationen

# Spring Boot Fullstack User-Management

In den folgenden Absätzen wird die erste CRUD-Anwendung in Spring Boot erstellt

## Projekt erstellen

Als erstes wurde ein GitHub-Projekt erstellt. <https://github.com/meilinger7/SbFirstCrud>

Spring Boot kann man direkt in der IntelliJ IDE erstellen, oder man legt sie über den Spring-Initializr an. <https://start.spring.io/>

Mit der IDE von IntelliJ:

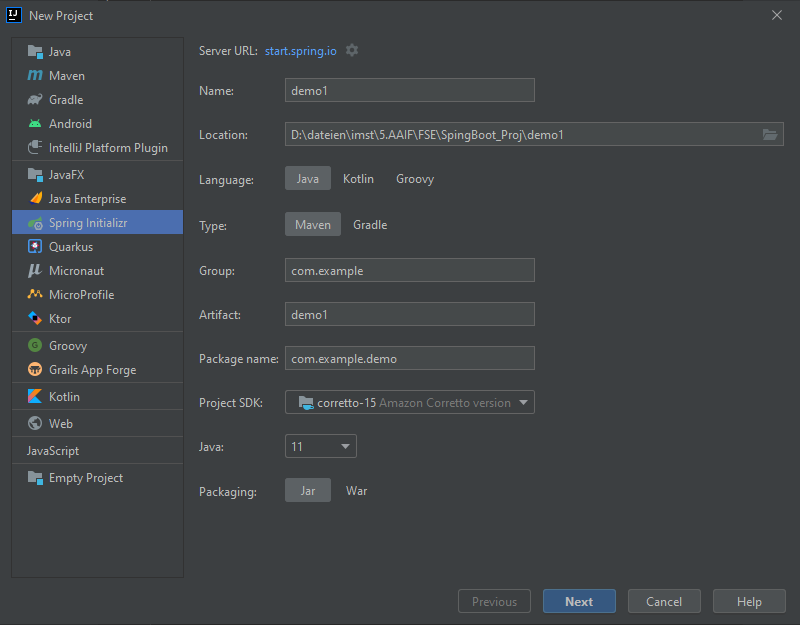


Abbildung : Settings des Spring Boot Projekt einstellen

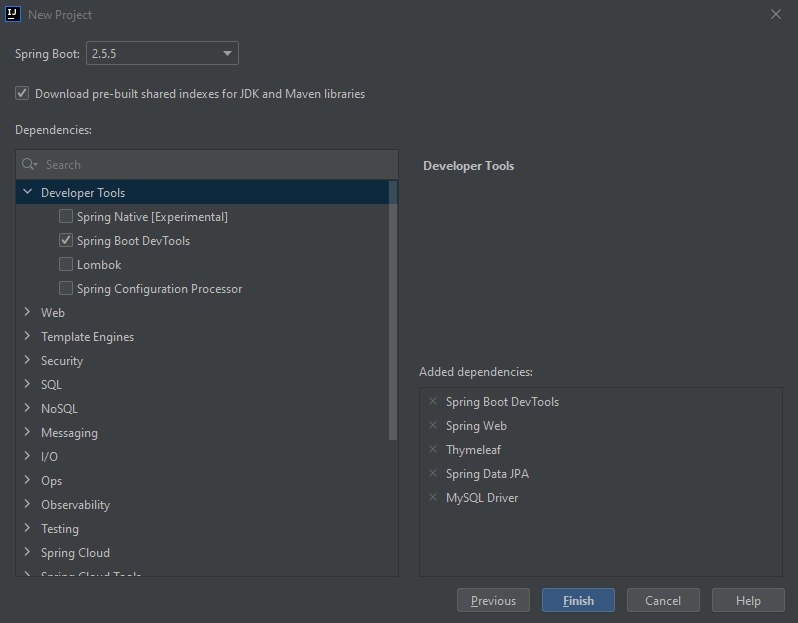


Abbildung : Dependencies von Spring Boot einstellen

Damit man nicht immer das Projekt neu starten muss, wenn man Veränderungen anzeigen möchte setzt man einen Hacken unter *File>Settings>Advanced Settings>Compiler.*

## Datenbank verbinden

Damit man sich mit einer Datenbank verbinden kann muss zuerst ein Datenbankservice laufen. Dafür habe ich den XAMPP MySQL Dienst gestartet.

In IntelliJ muss man mit dem Database-Tool sich nur noch mit der Datenbank verbinden und ein neues Schema erstellen.

### DataSource konfigurieren

Um eine Verbindung aufzubauen muss in der *application.properties* einige Metadaten zum Zugang der Datenbank eingefügt werden.

|  |
| --- |
| spring.datasource.url=jdbc:mysql://localhost:3306/userdb  spring.datasource.username=root  spring.datasource.password=  spring.jpa.hibernate.ddl-auto=update  spring.jpa.properties.hibernate.show\_sql=true |

## Homepage programmieren

Zuerst muss eine neue Java Klasse als Controller erstellt werden, damit die *index.html* beim Aufruf der Webseite aufgerufen wird.

|  |
| --- |
| @GetMapping("")  public String showHomePage() {  return "index";  } |

Eine index.html muss als nächstes im *templates* Ordner erstellt werden.

Beim Ausführen der main-funktion soll nun die Webseite über localhost:8080 erreichbar sein.

## Bootstrap hinzufügen

In der *pom.xml* müssen folgende Dependencies eingetragen werden:

|  |
| --- |
| <dependency>  <groupId>org.webjars</groupId>  <artifactId>bootstrap</artifactId>  <version>4.3.1</version>  </dependency>  <dependency>  <groupId>org.webjars</groupId>  <artifactId>webjars-locator-core</artifactId>  </dependency> |

Bei Fehlermeldungen muss Maven neu geladen werden.

In der index.html muss folgendes hinzugefügt werden:

|  |
| --- |
| <html lang="en" xmlns:th="http://www.thymeleaf.org">  …  <link rel="stylesheet" type="text/css" th:href="@{/webjars/bootstrap/css/bootstrap.min.css}"> |

Somit ist Bootstrap auch eingebunden.

## Data Access Layer programmieren

Damit man Objekte der User-Tabelle mit dessen Entitäten erstellen kann, muss es eine Klasse mit den Passenden Eigenschaften geben.

Dazu wird die Klasse User im Subpackege user erstellt. Danach muss die @Entity und die @Table(name = „users“) Annotation hinzugefügt werden.

In dieser Klasse müssen folgende Eigenschaften samt Annotation eingefügt werden.

|  |
| --- |
| @Id  @GeneratedValue(strategy = GenerationType.IDENTITY)  private Integer id;  @Column(nullable = false, unique = true, length = 45)  private String email;  @Column(length = 15, nullable = false)  private String password;  @Column(length = 45, nullable = false, name = "first\_name")  private String firstname;  @Column(length = 45, nullable = false, name = "last\_name")  private String lastname; |

Die Annotationen dienen dazu die Tabellen-Entitäten einzustellen.

Es müssen auch automatisch Getter und Setter erstellt werden.  
Rechtsklick auf Eigenschaft > Refactor > Encapsulate Fields > Alle Eigenschaften auswählen > Refactor

### User Ropository

Es muss ein Interface mit der Bezeichnung UserRepository erstellt werden. Diese erweitert sich nach CrudRepository<User, Integer>.

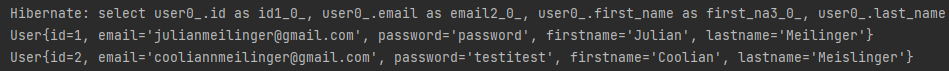
Wenn man nun den Code Kompiliert erstellt Hibernate automatisch die Tabelle mit entsprechenden Eigenschaften.



Ein Bild, das Text enthält.

Automatisch generierte Beschreibung







Ein Bild, das Text, Screenshot, schwarz, Bildschirm enthält.

Automatisch generierte Beschreibung