**Hello World**

**Maven oder Gradle**

**pom.xml**parent  
dependencies  
Java 1.8

**Application.java**   
@RestController   
Nimmt HTTP Request entgegen, fügt @ResponseBody bei Methoden mit @RequestMapping hinzu.  
@EnableAutoConfiguration  
Spring soll versuchen anhand der JAR Dependencies Spring automatisch zu konfigurieren. In Unserem Fall spring-boot-starter-web 🡪 Tomcat, Srping MVC  
  
@RequestMapping("/hello")  
Bei einem HTTP Request auf /hello wird die Methode aufegrufen.  
Main Methode  
Startet den Spring Container, Primäre String Komponente, Commandline Arguments

**Spring Starten**Browser Zugriff auf <http://localhost:8080/hello>  
21 Zeilen Java Code, Rest Service

**Powershell**mvn package  
Fat JARs, JAR welche alle benötigten Abhängigkeiten beinhaltet.  
java –jar target/\*.jar  
self contained = Bringt alles mit was die App zum Leben braucht. (In sich geschlossen)

**Recap**

**REST Service**

**Ticketsystem**

**start.spring.io**

**Projekt im Eclipse Zeigen**

**Application.java**@Configuration  
Spring Annotation, dadurch kann der Container konfiguriert werden (@ComponentScan) und man kann mit @Bean Beans deklarieren. z.B. eine andere Datenbank  
@ComponentScan  
Gibt an, dass Spring in dieses Package und in allen unter Packages nach Komponenten scannen soll.

**pom.xml**parent, dependencies  
Datenbank Treiber einfügen: H2 Datenbank  
Spring konfiguriert in-memory embedded database automatisch (H2, HSQL, Derby).

Andere DB und Datenbank Treiber oder @Bean dataSource:  
application.properties  
spring.datasource.url=jdbc:mysql://localhost/test  
spring.datasource.username=dbuser  
spring.datasource.password=dbpass  
spring.datasource.driverClassName=com.mysql.jdbc.Driver

**Datenmodell**Ticket, JPA Entity

**Ticket Repository**extend JpaRepository<Entity, Primary Key>  
Repository bringt typische CRUD Operationen (save, delete, findAll, findOne)  
Später darauf eingehen was das Ticket Repository noch alles kann  
  
**Ticket Controller**@RestController  
@RequestMapping("/tickets")  
Alle Request auf /tickets werden an diesen Controller weitergeleitet.  
  
TicketRepository, Constructor Injection, Testbarkeit  
getAllTickets - gibt alle Ticket zurück  
saveTicket - speichert Ticket in DB   
ResponseEntity  
Wrapper Objekt. Dadurch können wir die HTTP Headers und entsprechende Status Code zurück geben.

**Applikation Starten**

**Postman**3 Tickets erstellen  
Alle Tickets anzeigen

**Recap**Lauffähiger Restservice mit Datenbank Anbindung  
 **Lets go in the Cloud  
Heroku: PaaS**Procfile  
Gibt an welcher Befehlt beim Starten ausgeführt werden soll. Und das wir einen Web Dyno starten wollen.  
  
$JAVA\_OPTS  
JVM Optionen (Heap Size)  
  
-server.port=$PORT  
Tomcat soll auf den Übergebenen Port hören. Wichtig, weil Heroku den Port festlegt. Spring Boot erlaubt es die Applikation über Commandline Arguments zu konfigurieren. Mehr dazu später, wenn die Zeit reicht.  
  
system.properties  
java 1.8 verwenden  
  
Heroku verwende für das Deployment der Applikationen git.  
.gitignore

Init Git repo  
Befehle

Create App on Heroku

Deploy your code

Heroku Open, /tickets

POST und GET zeigen

**Recap**

**Spring Data**  
Projekt Ziel Datenzugriff vereinfachen (Rationale DB aber auch NoSQL z.B. MongoDB).  
Definierte Schlüssel Wörter (findByPropertyAndOrderByDateAsc)  
Generiert und Validiert Queries beim Starten von Spring Boot