Software Engineering II

Mehr zu Web Development (Maven, Spring Boot, MongoDB)

Anna-Lena Lamprecht (mit vielen Folien von Andy „Balu“ Großhennig)

Letzte Woche

- Organisatorisches
- MVC-Architektur für Webanwendungen
- Erste Schritte mit Spring Boot

- Setup der Arbeitsumgebung und “Hello World” mit Spring Boot

Plan für heute

- Build-Tool Maven
- Nächste Schritte mit Spring Boot
- Formulare mit HTML
- Datenbank MongoDB

Apache Maven

- Verbreitetes Build-Werkzeug
- Kann mehr als nur Kompilieren:
  - Abhängigkeitsverwaltung
  - Qualitätsanalysen von Programmcode
  - Erzeugung von API-Dokumentationen
  - …
- Entwickelt vornehmlich für Java
- Integriert in viele IDEs (u.a. Eclipse)
- Aktuelle Version: 3.9.9
- An Version 4 wird seit 2021 gearbeitet

https://maven.apache.org/

Maven

Klingt kompliziert - brauch ich das?

HelloworldApplication.java

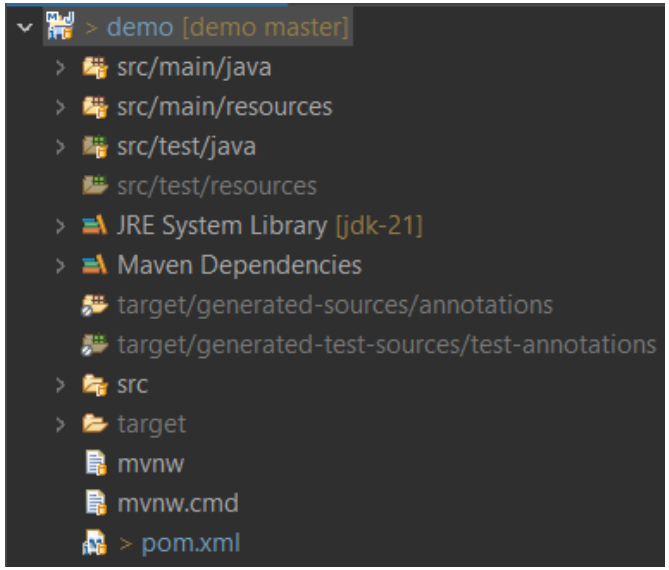
```java
package se2.demo;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@SpringBootApplication
@RestController
public class HelloworldApplication {

  public static void main(String[] args) {
    SpringApplication.run(HelloworldApplication.class, args);
  }

  @GetMapping("/")
  public String hello() {
    return "Hello World!";
  }

}
```

```
PS C:\Users\andyg\Desktop> javac .\HelloworldApplication.java
.\HelloworldApplication.java:3: error: package org.springframework.boot does not exist
import org.springframework.boot.SpringApplication;
                              ^
.\HelloworldApplication.java:4: error: package org.springframework.boot.autoconfigure does not exist
import org.springframework.boot.autoconfigure.SpringBootApplication;
                                              ^
.\HelloworldApplication.java:5: error: package org.springframework.web.bind.annotation does not exist
import org.springframework.web.bind.annotation.GetMapping;
                                               ^
.\HelloworldApplication.java:6: error: package org.springframework.web.bind.annotation does not exist
import org.springframework.web.bind.annotation.RestController;
                                               ^
.\HelloworldApplication.java:8: error: cannot find symbol
@SpringBootApplication
 ^
  symbol: class SpringBootApplication
.\HelloworldApplication.java:9: error: cannot find symbol
@RestController
 ^
  symbol: class RestController
.\HelloworldApplication.java:16: error: cannot find symbol
       @GetMapping("/")
        ^
  symbol:   class GetMapping
  location: class HelloworldApplication
.\HelloworldApplication.java:13: error: cannot find symbol
              SpringApplication.run(HelloworldApplication.class, args);
              ^
  symbol:   variable SpringApplication
  location: class HelloworldApplication
8 errors
```

Maven

Klingt kompliziert - brauch ich das?

Software basierend auf Frameworks lässt sich gegenenenfalls nicht mühelos kompilieren



- Ein Build Tool kümmert sich um
  - Dependency Management
  - Project building (compiling, linking, ...)
  - Project Structure
  - Scalability Problems

Standard-Verzeichnisstruktur

```
my-app/
│
├── pom.xml                          # Die zentrale Projektdatei
├── src/
│   ├── main/
│   │   ├── java/                    # Java-Quellcode
│   │   │   └── com/
│   │   │       └── beispiel/
│   │   │           └── App.java     # Beispielklasse
│   │   ├── resources/               # Ressourcen wie .properties, XML, etc.
│   │
│   ├── test/
│   │   ├── java/                    # Test-Quellcode (z.B. JUnit)
│   │   │   └── com/
│   │   │       └── beispiel/
│   │   │           └── AppTest.java
│   │   ├── resources/               # Test-Ressourcen
│
└── target/                          # Build-Verzeichnis (automatisch erstellt)
```

Konfigurationsdatei pom.xml (Project Object Model)

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project>
    <modelVersion>4.0.0</modelVersion>

    <groupId>org.sample</groupId>
    <artifactId>project-name</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>pom | jar | war | ear </packaging>
</project>
```

Minimale POM

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project  xmlns="http://maven.apache.org/POM/4.0.0"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                              http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.sonatype.mavenbook.simple</groupId>
  <artifactId>simple</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>simple</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

POM mit Dependencies

Nur direkte Dependencies notwendig.
Dependencies von Dependencies werden automatisch geladen!

Auflösung von Abhängigkeiten

Einer der wichtigsten Erfolgsfaktoren von Maven. ☺

- Externe Abhängigkeiten werden in der pom.xml notiert.
- Werden während des Buildvorgangs im target-Verzeichnis bereitsgestellt.
- Dabei prüft Maven zunächst, ob Artefakt bereits in `.m2/repository` vorhanden ist.
- Falls nicht, wird in entferntem Maven-Repository (z.B. “Maven Central”) danach gesucht und herunterladen
- Man kann auch eigenen Repository-Server betreiben

Lebenszyklen

Maven definiert drei Standard-Lebensyklen:

1. **`clean`** zum Löschen von Ergebnissen vorheriger Builds, mit den Phasen `pre-clean, clean, post-cl`
2. **`build`** (default) zum Erstellen des Projekts im Rahmen der untengenannten Phasen,
3. **`site`** zum Erstellen von Webseiten zur Projektdokumentation und Reports, mit den Phasen `pre-site, site, post-site, site-deploy`.

Über Plug-Ins weitere Funktionen verfügbar, z.B. `validate, compile, test, package, install, deploy, …`

Webframework: Spring Boot

Noch ein fancy Dingenskirchen?? Geht es nicht auch ohne?

Server Code

We are going to develop the following HTTP server code:

```java
server.createContext("/test", new  MyHttpHandler());
server.setExecutor(threadPoolExecutor);
server.start();
logger.info(" Server started on port 8001");
```

We created a context called, `test`. This is nothing but the context root of the application. The second parameter is a handler instance, which will handle the HTTP requests. We will look into this class shortly.

We can use a thread pool executor, along with this server instance. In our case, we created a thread pool with 10.

```java
ThreadPoolExecutor threadPoolExecutor = (ThreadPoolExecutor)Executors.newFixedThreadPool(10);
```

Next, we start the server:

```java
server.start();
```

```java
private class MyHttpHandler implements HttpHandler {
  @Override
  public void handle(HttpExchange httpExchange) throws IOException {
    String requestParamValue=null;
    if("GET".equals(httpExchange.getRequestMethod())) {
       requestParamValue = handleGetRequest(httpExchange);
     }else if("POST".equals(httpExchange)) {
       requestParamValue = handlePostRequest(httpExchange);
      }
    handleResponse(httpExchange,requestParamValue);
  }
   private String handleGetRequest(HttpExchange httpExchange) {
           return httpExchange.
                   getRequestURI()
                   .toString()
                   .split("\\?")[1]
                   .split("=")[1];
   }
   private void handleResponse(HttpExchange httpExchange, String requestParamValue)  throws  IOException {
           OutputStream outputStream = httpExchange.getResponseBody();
           StringBuilder htmlBuilder = new StringBuilder();

           htmlBuilder.append("<html>").
                   append("<body>").
                   append("<h1>").
                   append("Hello ")
                   .append(requestParamValue)
                   .append("</h1>")
                   .append("</body>")
                   .append("</html>");
           // encode HTML content
           String htmlResponse = StringEscapeUtils.escapeHtml4(htmlBuilder.toString());

           // this line is a must
           httpExchange.sendResponseHeaders(200, htmlResponse.length());
           outputStream.write(htmlResponse.getBytes());
           outputStream.flush();
           outputStream.close();
       }
}
```

Webentwickung mit Spring Boot

Spring / Spring Boot (https://spring.io/projects/spring-boot)

- Spring: beliebtes Open-Source-Framework zur Erstellung eigenständiger Java-Anwendungen, die auf der Java Virtual Machine (JVM) laufen.
- Spring Boot: vereinfacht die Entwicklung von Webanwendungen und Mikroservices aus Basis von Spring

Letzte Woche schon gesehen/gemacht:

- Projekterstellung mit Spring initalizr (https://start.spring.io)
- Weiterbearbeitung in IDE
- Ausführen der Anwendung started den Server (http://localhost:8080)
- Funktionalität hinzufügen über Klassen und Annotationen

Spring Framework

Will die Entwicklung mit Java/Jave EE vereinfachen
und gute Programmierpraktiken fördern.

Prinzipien (nach Rod Johnson):

1. Dependency Injection
2. Aspektorientierte Programmierung
3. Vorlagen

https://spring.io/projects/spring-framework

Ermöglicht dadurch POJO-basiertes Programmieren.

Dependency Injection in Spring

Grundidee: Ein Objekt bekommt seine Abhängigkeiten von außen geliefert, anstatt sie selbst zu erzeugen.

Dadurch wird Code modularer, besser testbar und lose gekoppelt.

Spring verwaltet die Objekte einer Anwendung im sogenannten **Application Context**. Diese Objekte nennt man **Beans**.

Wenn eine Bean eine andere braucht, „spritzt“ Spring ihr diese Abhängigkeit automatisch rein.

Dependency Injection in Spring

Woher weiß Spring, was es injecten soll?

- Durch **Annotationen** wie `@Component`, `@Service`, `@Repository`, `@Controller` erkennt Spring automatisch Klassen, die es als Beans verwalten soll.
- Mit `@Autowired` (oder ab Spring 4.3 sogar ohne) wird signalisiert: Hier wird etwas benötigt.

```java
@Component
public class Engine { }

@Component
public class Car {
    private final Engine engine;

    public Car(Engine engine) {
        this.engine = engine;
    }
}
```

Spring erkennt: „Ah, es gibt eine `Engine`-Bean, also nutze ich die, um `Car` zu bauen.“

Dependency Injection in Spring

Drei Hauptarten von Dependency Injection in Spring:

1. Konstruktor-Injektion (empfohlen, Beispiel rechts)
2. Setter-Injektion (im Allgemeinen nicht empfohlen, Ausnahme bei optionalen Abhängigkeiten)
3. Feld-Injektion (nicht empfohlen, da schwer testbar)

```java
@Component
public class Car {
    private final Engine engine;

    @Autowired
    public Car(Engine engine) {
        this.engine = engine;
    }
}
```

Aspektorientierte Programmierung (AOP)

Grundidee: Generische Funktionalitäten über mehrere Klassen hinweg verwenden und von der eigentlichen Geschäftslogik trennen.

Können an bestimmten Stellen im Code “eingehängt” werden, ohne den Code direkt zu verändern.

Beispiele für solche **Cross-Cutting Concerns**:

- Transaktionsverwaltung
- Logging
- Security
- Performance-Monitoring
- Caching
- Fehlerbehandlung

Aspektorientierte Programmierung (AOP)

Spring AOP basiert hauptsächlich auf **Proxies** und wird über **Annotations** oder XML-Konfiguration umgesetzt.

Die häufigsten Techniken basieren auf Annotations mit **AspectJ-Syntax**.

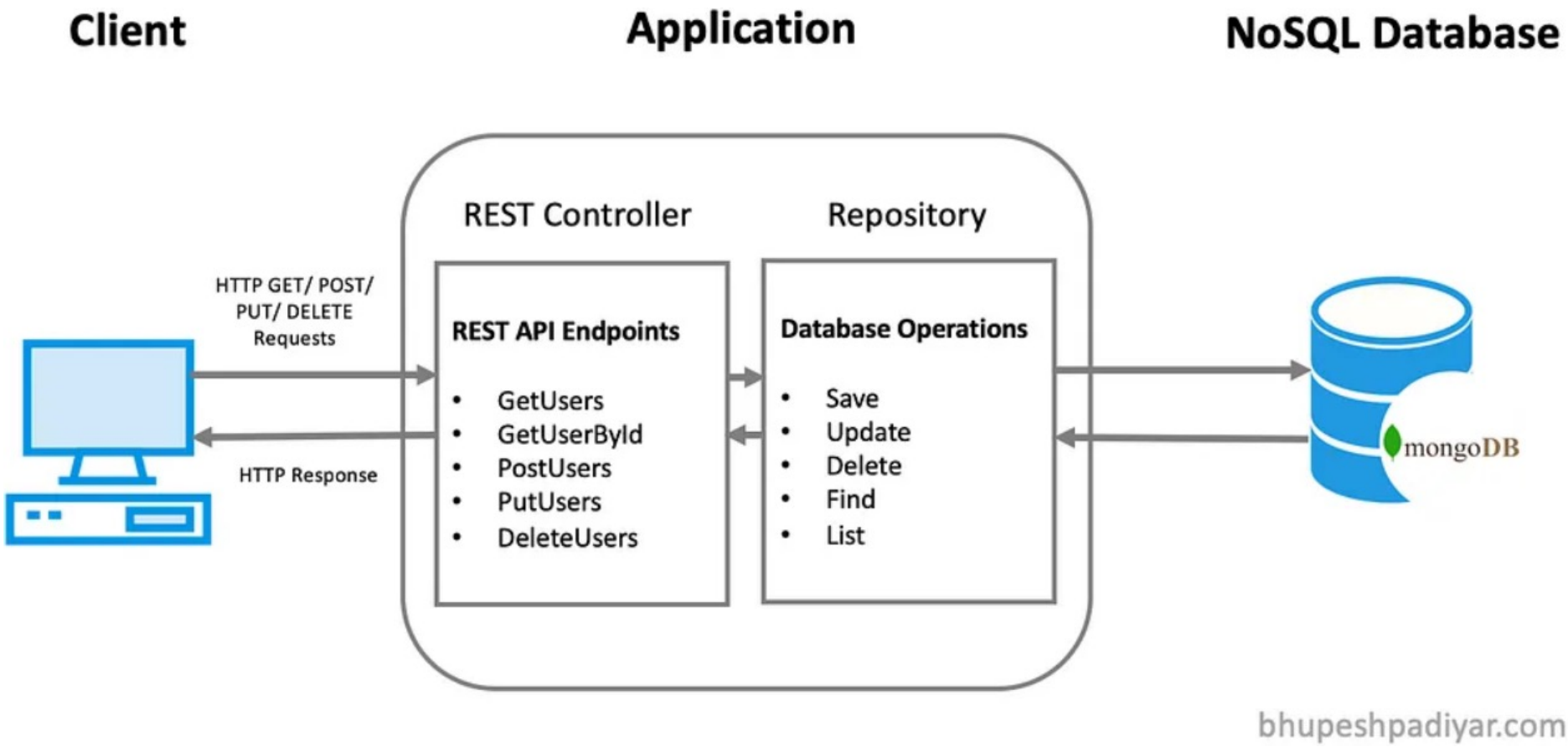
```java
@Aspect
@Component
public class LoggingAspect {

    @Before("execution(* com.beispiel.service.*.*(..))")
    public void logBefore(JoinPoint joinPoint) {
        System.out.println("Methode wird aufgerufen: " + joinPoint.getSignature().getName())
    }

    @After("execution(* com.beispiel.service.*.*(..))")
    public void logAfter(JoinPoint joinPoint) {
        System.out.println("Methode abgeschlossen: " + joinPoint.getSignature().getName());
    }
}

```

Spring Boot

Erweiterung des Spring-Frameworks für die **einfache Entwicklung eigenständig lauffähiger Spring-Anwendungen per Konvention vor Konfiguration**, die ohne XML-Konfiguration auskommen und alle nötigen Klassenbibliotheken mitbringen.

- Im **Spring Initalizer** können Abhängigkeiten (Web-Frameworks, Datenbanktreiber, …) ausgewählt werden, manuelle Konfiguration entfällt.
- Einbindung von "**Startern**" in Maven stellt gängige Standardkonfiguration bereit, z.B. startet "spring-boot-starter-web" in der Standardeinstellung automatisch einen integrierten Tomcat-Webserver.
- Weitere Starter z.B. für Persistierung mit Hibernate oder Spring Security.

Typischer Aufbau von Webanwendungen mit Spring Boot



Representational State Transfer (REST)

Architekturstil für Web-APIs, der auf standardisierten HTTP-Methoden basiert:

- **GET** – Daten lessen (z.B. alle Nutzer abfragen)
- **POST** – Daten erstellen (z.B. neuen Nutzer erstellen)
- **PUT** – Daten aktualisieren (z.B. Nutzer bearbeiten)
- **DELETE** – Daten löschen (z.B. Nutzer löschen)

Außerdem:

- **Client-Server-Modell**: Trennung von Frontend und Backend
- **Zustandslosigkeit**: Jeder Request enthält alle nötigen Infos
- **Ressourcenbasiert**: Jede Ressource hat eine eindeutige URL

@RestController Annotation

Markiert eine Klasse als Controller, der REST-konforme HTTP-Antworten liefert.

Kombiniert intern:

- @Controller → sagt Spring: "Das ist ein Web-Controller"
- @ResponseBody → sagt: "Der Rückgabewert jeder Methode wird direkt als HTTP-Antwort geliefert (z. B. als JSON), nicht als View (z.B. HTML)"

HelloworldApplication.java

```java
package se2.demo;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@SpringBootApplication
@RestController
public class HelloworldApplication {

  public static void main(String[] args) {
    SpringApplication.run(HelloworldApplication.class, args);
  }

  @GetMapping("/")
  public String hello() {
    return "Hello World!";
  }

}
```

@Controller Annotation

Markiert eine Klasse als Controller für den Web-Layer

- Spring scannt diese Klasse und registriert sie als Bean
- Methoden darin können HTTP-Requests entgegennehmen (z. B. mit @GetMapping, @PostMapping, etc.)
- Der Rückgabewert ist in der Regel ein View-Name (z. B. HTML-Seite)

```java
@Controller
public class HelloController {

    @GetMapping("/hello")
    public String hello(Model model) {
        model.addAttribute("name", "Anna");
        return "hello";  // => zeigt hello.html
    }
}
```

@RestController vs. @Controller

Mit @RestController:

- **Antwort = rohe Daten** (z. B. JSON, XML, Text)
- Der Client (Browser) muss diese **Antwort selbst verarbeiten** → z. B. mit JavaScript → Daten holen → HTML manipulieren

Mit @Controller + Thymeleaf:

- **Antwort = fertiges HTML**
- Spring rendert die Seite auf dem Server
- Browser zeigt sie direkt an → **Keine weitere Verarbeitung nötig**

Welche Variante wofür verwenden?

Situation	Empfehlung
Interaktive Web-App / SPA	`@RestController` + JavaScript
Klassische Seiten mit Formularen	`@Controller` + Thymeleaf
Mobile App oder externes System nutzt API	`@RestController` (z. B. für JSON)

💡 **Merksatz:**

`@Controller` ist für Seiten.

`@RestController` ist für Daten.

HTML

HTML (HyperText Markup Language) ist vermutlich/hoffentlich den meisten grundsätzlich bekannt.

- Grundgerüst
- Head und Body
- Absätze und Zeilenumbrüche
- Hyperlinks
- Bilder
- Tabellen
- Überschriften

Zum Auffrischen oder Selbstlernen der Basics gibt es viele gute Quellen, z.B.

- https://www.heise.de/tipps-tricks/HTML-Grundlagen-Was-Einsteiger-wissen-muessen-3887124.html

```
<html>
    <head>
        <title>Titel der Datei</title>
    </head>
    <body>
        Inhalt der Datei
    </body>
</html>
```

Formulare mit HTML

Rückgrat jeder interaktiven Webseite (Login, Suche, Anmeldung, Kontakt-formulare usw.)

```
<form action="/ziel" method="post">
    <label>Benutzername:</label>
    <input type="text" name="username" />

    <label>Passwort:</label>
    <input type="password" name="password" />

    <input type="submit" value="Absenden" />
</form>
```

Grundaufbau eines HTML-Formulars

Tag / Attribut	Bedeutung
<form>	Start des Formulars
action="/ziel"	Ziel-URL (wohin die Daten geschickt werden)
method="post"	Methode (meist POST oder GET)
<input>	Eingabefeld
type="text"	Einfaches Texteingabefeld
type="password"	Versteckte Eingabe (z. B. für Passwörter)
type="submit"	Button zum Absenden des Formulars
name="..."	Der Schlüssel, unter dem das Feld gesendet wird

HTML Input-Typen

- `text` Einfaches Textfeld
- `textarea` Mehrzeiliges Textfeld
- `password` Versteckter Text (•••)
- `email` Für E-Mail-Adressen
- `checkbox` Auswahlbox zum Ankreuzen
- `radio` Eine von mehreren Optionen
- `file` Datei-Upload
- `date` Datumsauswahl

Mehr dazu nächste Woche in der Übung ☺

```java
@GetMapping("/record")
public String record(Model model, @CookieValue(value = "sessionId", defaultValue = "") String sessionId) {
  if (sessionId.isEmpty()) return "redirect:/login";

  // Number of timestamps even: Next action is clock in and if odd: clock out
  boolean isClockIn = records.findByemail(sessionId).getWorkingTimes().size() % 2 == 0;

  model.addAttribute("isClockIn", isClockIn);
        return "record";
}
```

```html
<!DOCTYPE HTML>
<html lang="de">
<head>
    <title>Zeiterfassung</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body>
        <h1>Zeiterfassung</h1>
    <form action="" method="post">
      <p><input th:disabled="!${isClockIn}" type="submit" value="Einstempeln"/> <input th:disabled="${isClockIn}" type="submit" value="Ausstempe
    </form>
</body>
</html>
```

Datenbank MongoDB

Populäres, dokumentenzentriertes Datenbankmanagementsystem für Webanwendungen.

Verwaltet Sammlungen von JSON-ähnlichen Dokumenten.

Populärste NoSQL-Datenbank.

https://www.mongodb.com/

NoSQL Datenbank?!

- NoSQL? Was ist denn erstmal YesSQL?
  - SQL ist eine Datenbanksprache zur Manipulation relationaler Datenbanken
  - Relational: Die Tabellen und ihre Inhalte sind also untereinander verknüpft
  - MySQL, SQLite, ... sind SQL Datenbanken
  - Beispiel-Query: “SELECT * FROM Student;”

- NoSQL!
  - Verschiedenste Datenverwaltungsvarianten:
    - Dokument (JSON, XML, ...)
    - Graph-Speicher (Knoten & Kanten)
    - Key-Value-Speicher
  - MongoDB, Redis, ... sind noSQL Datenbanken
  - Beispiel-Query: “db.collection('inventory').find({ status: 'D' });”

NoSQL Datenbank: MongoDB

- Aufbau
  - Datenbanken
  - Collections
  - Documents
  - Fields
  - Values

NoSQL Datenbank: MongoDB

- Integration in Spring Boot
  - Datenbankkonfiguration in src/main/resources/application.properties
  - “Model”- Klassen, welche MongoDB Collections entspricht
  - Annotations, um Elemente aus der Anwendung mit Inhalten der Datenbank zu verbinden
  - “Repository” - Interface für die Bereitstellung der Datenbankoperationen

NoSQL Datenbank: MongoDB

- Konfiguration
  - Beinhaltet ggf. auch Datenbank-Login-Daten

application.properties

```
spring.application.name=helloworld
spring.data.mongodb.host=localhost
spring.data.mongodb.port=27017
spring.data.mongodb.database=Worlds
```

- “Model” - Klasse
  - Bekommt Annotations, um Attribute mit Collection-Feldern zu verknüpfen

World.java

```java
package se2.demo;

import org.springframework.data.annotation.Id;

public class World {
  @Id
  public String id;
  public String star;

  public World(String star) {
    this.star = star;
  }
}
```

NoSQL Datenbank: MongoDB

- Interface
  - Query Creation by Method Names

WorldRepository.java

```java
package se2.demo;

import org.springframework.data.mongodb.repository.MongoRepository;

public interface WorldRepository extends MongoRepository<World, String>{
  public List<World> findByStar(String star);
}
```

- Anwendung

HelloworldApplication.java

```java
package se2.demo;

import org.springframework.beans.factory.annotation.Autowired;

@SpringBootApplication
@RestController
public class HelloworldApplication implements CommandLineRunner{
  @Autowired
  private WorldRepository wr;

  public static void main(String[] args) {
    SpringApplication.run(HelloworldApplication.class, args);
  }

  @Override
  public void run(String... args) {
    wr.save(new World("Sol"));
    System.out.println(wr.findByStar("Sol").get(0));
  }
```

NoSQL Datenbank: MongoDB

- Query Creation by Method Names

```
interface PersonRepository extends Repository<Person, Long> {

  List<Person> findByEmailAddressAndLastname(EmailAddress emailAddress, String lastname);

  // Enables the distinct flag for the query
  List<Person> findDistinctPeopleByLastnameOrFirstname(String lastname, String firstname);
  List<Person> findPeopleDistinctByLastnameOrFirstname(String lastname, String firstname);

  // Enabling ignoring case for an individual property
  List<Person> findByLastnameIgnoreCase(String lastname);
  // Enabling ignoring case for all suitable properties
  List<Person> findByLastnameAndFirstnameAllIgnoreCase(String lastname, String firstname);

  // Enabling static ORDER BY for a query
  List<Person> findByLastnameOrderByFirstnameAsc(String lastname);
  List<Person> findByLastnameOrderByFirstnameDesc(String lastname);
}
```

Das war’s für heute

- Build-Tool Maven
- Nächste Schritte mit Spring Boot
- Formulare mit HTML
- Datenbank MongoDB

Zum Weiterlesen

- Einführung in Build Tools
  - https://www.browserstack.com/guide/build-tools
- Einführung in Maven
  - https://maven.apache.org/what-is-maven.html
  - https://maven.apache.org/guides/getting-started/maven-in-five-minutes.html
- Einführung in Spring
  - https://www.marcobehler.com/guides/spring-framework
- Einführung in Spring Boot
  - https://www.tutorialspoint.com/spring_boot/spring_boot_introduction.htm
  - https://medium.com/sliit-foss/springboot-and-rest-apis-in-java-c853361e9a12
- Einführung in MongoDB in Spring Boot Umgebung
  - https://spring.io/guides/gs/accessing-data-mongodb
  - https://www.mongodb.com/compatibility/spring-boot
  - https://docs.spring.io/spring-data/mongodb/reference/repositories/query-methods-details.html
  - https://medium.com/@dangeabunea/how-to-create-a-custom-mongodb-spring-data-repository-e51c343064e1

Semesterplan (Änderungen möglich)

Woche	Gruppentermine (Mo, Di)	Vorlesungstermin (Do)	Abgaben (Fr)
15 (7.4.-11.4.)	Selbständig: Übung 0 (Arbeitsumgebung)	Einführung, Organisatorisches, erste Schritte mit Spring Boot	
16 (14.4.-18.4.)	Setup, “Hello World” mit Spring Boot	Mehr zu Web Development (nächste Schritte mit Spring Boot, MongoDB, Maven etc.)	
17 (21.4.-25.4.)	Beispiel Registrierungs-App, Vorbereitung Mini-Projekt	Scrum	
18 (28.4.-2.5.)	Hilfe Mini-Projekte	*(1. Mai!)*	Mini-Projekte (PNL)
19 (5.5.-9.5.)	Projektplanung (Rollen etc.), Grundlegende Architektur- und Designentscheidungen	Kundengespräch (Design Review)	
20 (12.5.-16.5.)	Sprint Planning	Git	
21 (19.5.-23.5.)	Daily Scrum und gemeinsame Arbeitszeit	Kundengespräch (Sprint Review)	Sprintbericht 1
22 (26.5.-30.5.)	Retrospektive und Sprint Planning	*(Himmelfahrt!)*	
23 (2.6.-6.6.)	Daily Scrum und gemeinsame Arbeitszeit	Kundengespräch (Sprint Review)	Sprintbericht 2
24 (9.6.-13.6.)	*(Pfingstwoche)*	*(Pfingstwoche)*	
25 (16.6.-20.6.)	Retrospektive und Sprint Planning	Herausforderungen bei der Teamarbeit angehen	
26 (23.6.-27.6.)	Daily Scrum und gemeinsame Arbeitszeit	Kundengespräch (Sprint Review)	Sprintbericht 3
27 (30.6.-4.7.)	Retrospektive und Sprint Planning	Scrum in der Praxis (UP Transfer und Externe)	
28 (7.7.-11.7.)	Daily Scrum und gemeinsame Arbeitszeit	Kundengespräch (Sprint Review)	Sprintbericht 4
29 (14.7.-18.7.)	Retrospektive	Abschlusspräsentationen	Finales Release

Was ist jetzt wichtig?

- Nächste Übung besuchen und Übungsblatt 2 bearbeiten
- Hausaufgabe von Übungsblatt 2 abgeben (erster Teil der PNL!)
- Nächste Woche wieder in die Vorlesung kommen