RICH CLIENT: SERVER ANWENDUNG

ÜBERLEITUNG

VERANTWORTLICHKEITEN - JSF

- View-Management
- Rendering
- Validation
- State-Management
- Events
- Routing
- Data-Management
- Persistence

Speaker notes

- these are some notes

VERANTWORTLICHKEITEN - RICH CLIENT

- View-Management
- Rendering
- Ensurance
- State-Management
- Events
- Routing

VERANTWORTLICHKEITEN - WEBSERVICE

- Data-Management
- Validation
- Persistence

WEBSERVICE

WEBSERVICE - STATELESS

- Kein Zustand
- Keine Session
- Anfrage ausschließlich mit fachlichen Informationen

Speaker notes

- Keine Übertragung von Zustandsänderungen
- Keine Übertragung von unzusammenhängenden Informationen

WEBSERVICE - STATELESS

- Keine nicht-persistenten Informationen
- Transparentes Caching ausgenommen
- Persistierung in Datenbank oder Dateisystem
- Transparente Datenbank oder Dateisystem

Speaker notes

- Keine anfrageübergreifende Informationen
- Transparente Caches agieren auf persistenten oder berechenbaren Daten
- Transparente Caches agieren niemals auf flüchtigen Daten
- Transparente Caches für persistente Daten müssen kurzlebig oder Änderungen bewusst sein
- Transparente Datenbanken/Dateisysteme (ver)teilen über mehrere Instanzen

WEBSERVICE - SKALIERBAR

- Abhängig von ausschließlich externen Informationen
- Eingaben des Clients
- Daten der Persistence
- Instanzen sind identitätslos
- Dynamisches hoch-/runterfahren von Instanzen

WEBSERVICE - UNTRUSTING

- Validierung aller Eingaben
- Isolierung aller Eingaben
- Durchgehende Prüfung der Authorisierung

Speaker notes

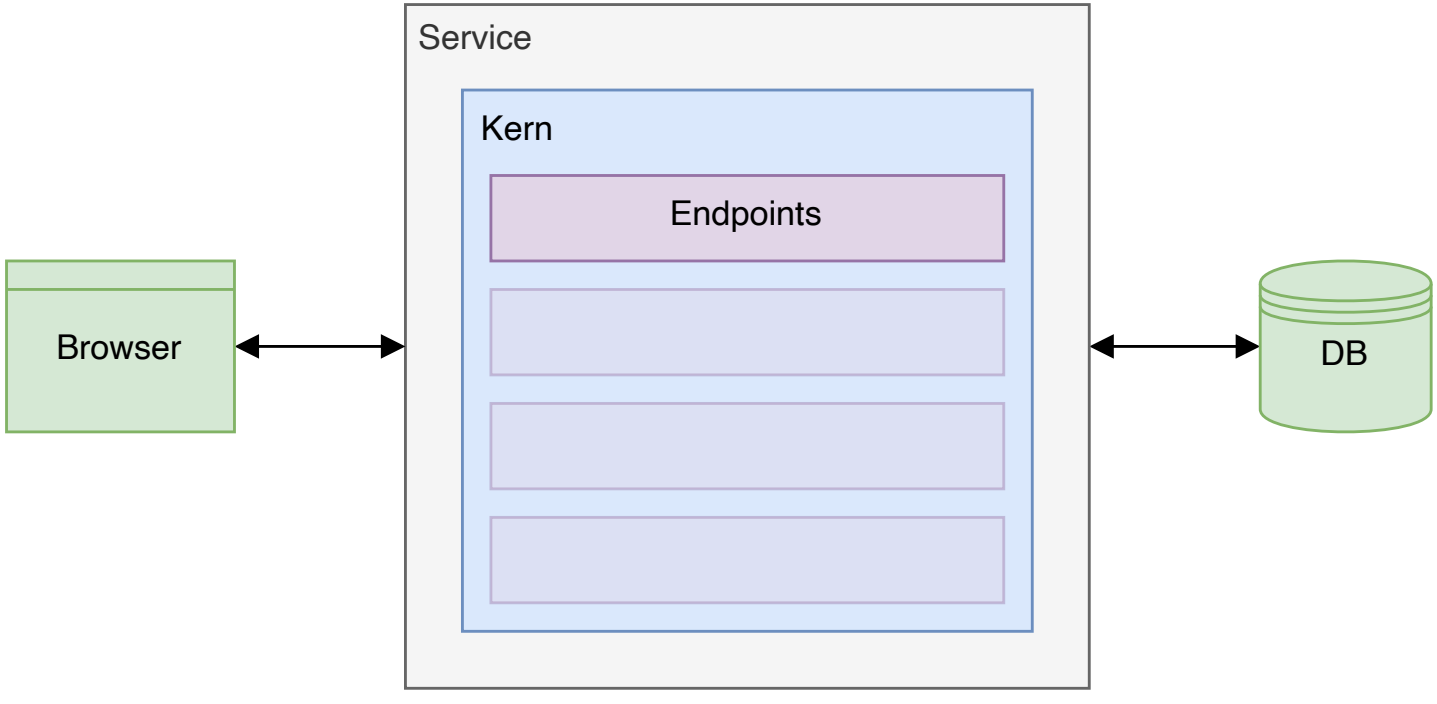
- Validierung stellt Korrektheit sicher
- Validierung stellt keine Sicherheit sicher
- Isolierung durch formlose Betrachtung der Eingaben
- Isolierung durch z.B. Prepared-Statements
- Mindestens Validierung des Tokens (zumeist über Signature/Secret)
- Eventuelle Überprüfung der Zugriffsrechte

ARCHITEKTUREN

ARCHITEKTUREN

- Monolith
- Modulith
- Services
- Microservice

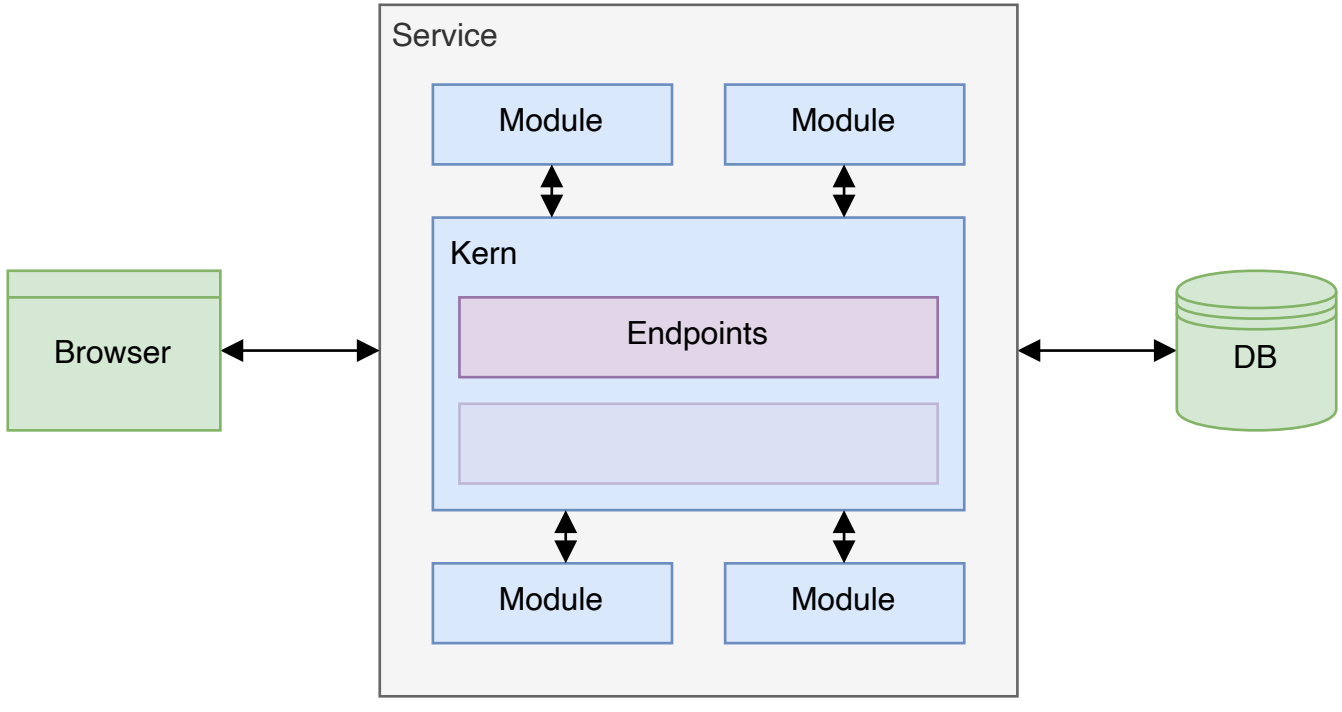
ARCHITEKTUREN - MONOLITH



ARCHITEKTUREN - MONOLITH

- Alle Aspekte der Anwendung in einem Projekt
- Keine Trennung zwischen Fachlichkeiten
- Keine externen Abhängigkeiten zur Laufzeit

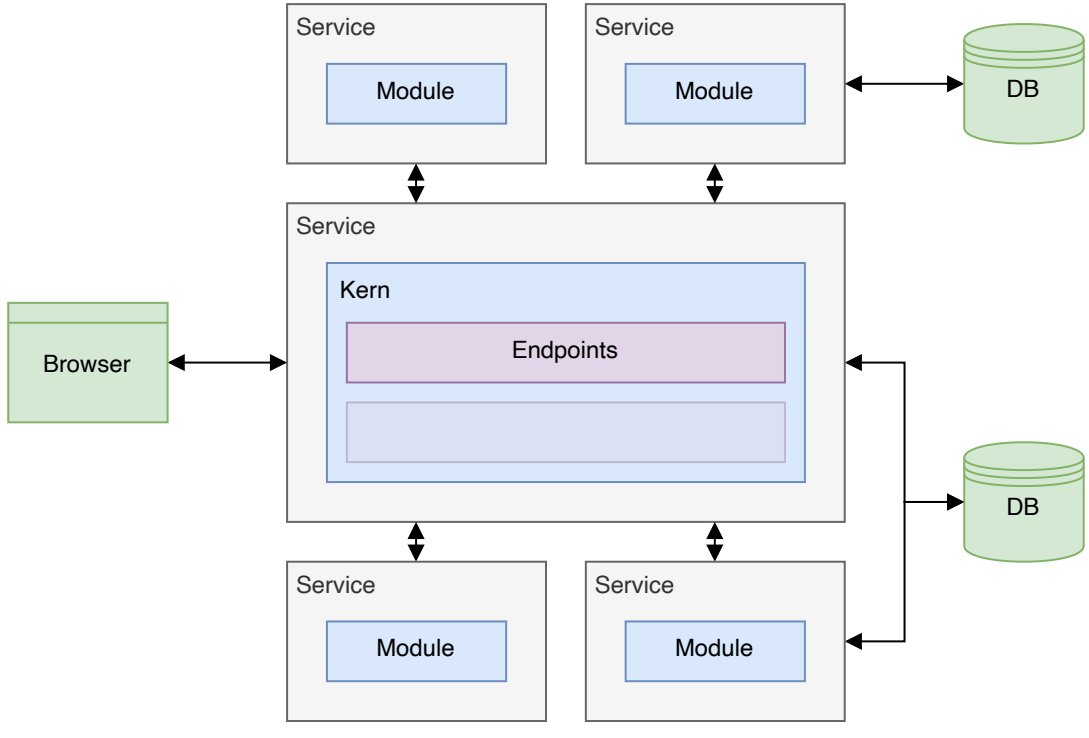
ARCHITEKTUREN - MODULITH



ARCHITEKTUREN - MODULITH

- Unterteilung der Anwendung in Fachlichkeiten
- Auslagerung der Fachlichkeiten in Module
- Module definieren öffentliche Schnittstellen
- Auslagerung in Form von Package, Modul, Projekt
- Keine Auslagerung zur Laufzeit
- Zusammengeführt durch Kern

ARCHITEKTUREN - SERVICES



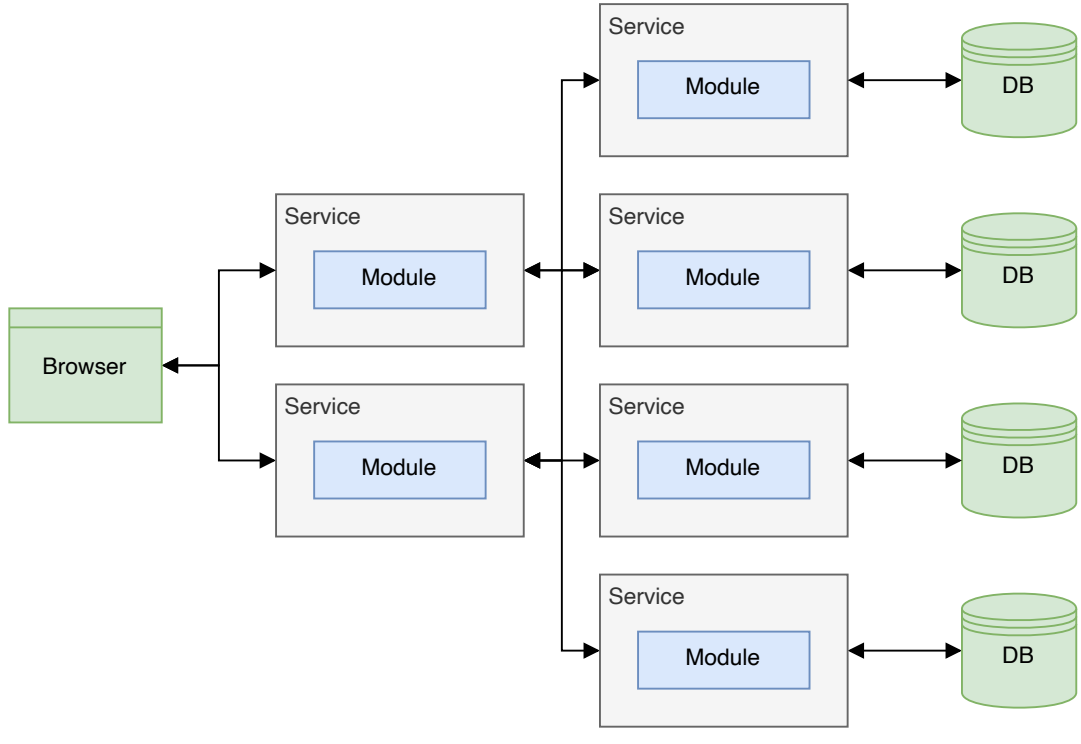
ARCHITEKTUREN - SERVICES

- Modulith als Kern
- Auslagerung einzelner Module in Services
- Services haben eigene Datenhaltung

Speaker notes

- Daten müssen nicht repliziert werden, da Kern das Mapping übernimmt

ARCHITEKTUREN - MICROSERVICES



ARCHITEKTUREN - MICROSERVICES

- Auslagerung jedes Modules in Services
- Expliziter Kern durch implizite Abhängigkeiten zwischen Services ersetzt
- Services replizieren Daten in eigener Datenhaltung

Speaker notes

- Daten müssen repliziert werden, da es keinen Kern fürs Mapping gibt

VERGLEICH

VERGLEICH - KRITERIEN

- Initialaufwand
- Wartungsaufwand
- Betriebsaufwand
- Personalaufwand

VERGLEICH - KRITERIEN

- Abhängigkeit
- Ausführbarkeit
- Testbarkeit
- Skalierbarkeit
- Zuverlässigkeit
- Ausfallsicherheit

VERGLEICH - INITIALAUFWAND

- Aufsetzten der Architektur

VERGLEICH - INITIALAUFWAND

Monolith	Modulith	Services	Microservices
Gering	Mittel	Mittel	Hoch

Speaker notes

- Monolith braucht kein Konzept
- Modulith braucht einfaches Konzept
- Services braucht erweitertes Konzept
- Microservices braucht allgemeines Konzept

VERGLEICH - WARTUNGSAUFWAND

- Einführung neuer Features
- Entfernung alter Features
- Behebung von Fehler
- Aktualisierung der Abhängigkeiten
- Refactoring

Speaker notes

- Abhängigkeiten sind Sprache, Frameworks, Libraries, Services, Datenbanken, Schnittstellen

VERGLEICH - WARTUNGSAUFWAND

Monolith	Modulith	Services	Microservices
Hoch	Mittel	Mittel	Gering

Speaker notes

- Monolith: stark erhöhter Aufwand
- Modulith: Module leicht erweiterbar; Kern erhöhter Aufwand
- Services: Aufwand abhängig von Kern oder Service
- Microservices: können vollständig neugeschrieben werden

VERGLEICH - BETRIEBSAUFWAND

- Betreiben der Services
- Instandhaltung der Umgebung
- Behebung von Störungen

VERGLEICH - BETRIEBSAUFWAND

Monolith	Modulith	Services	Microservices
Gering	Gering	Mittel	Hoch

Speaker notes

- Monolith: wenige, große Instanzen; wenige Server, physische Umgebung
- Modulith: s.h. Monolith
- Services: wenige, große Instanzen + kleine, evt. häufige Services; einige Server, evt. partiell virtualisierte Umgebung
- Microservices: viele, kleine Instanzen; viele Server, virtualisierte Umgebung

VERGLEICH - PERSONALAUFWAND

- Teamgröße sowie Teamanzahl
- Erhöhte Komplexität erfordert mehr Personal
- Mehr Personal erfordert erhöhte Flexibilität

VERGLEICH - PERSONALAUFWAND

Monolith	Modulith	Services	Microservices
Gering	Mittel	Mittel	Hoch

Speaker notes

- Monolith: sehr kleines Entwicklungsteam benötigt; Operations durch Entwickler
- Modulith: sehr kleines bis kleines Entwicklungsteam benötigt; evt. extra Personal für Operations
- Services: Entwicklungsteam abhängig von Größe des Projektes; extra Personal für Operations
- Microservices: ein, großes bis mehrere, kleine Entwicklungsteams; klein bis mittleres Operationsteam

VERGLEICH - ABHÄNGIGKEIT

- Trennung der Fachlichkeiten
- Freiheit der Technologien

VERGLEICH - ABHÄNGIGKEIT

Monolith	Modulith	Services	Microservices
Hoch	Hoch	Mittel	Gering

Speaker notes

- Monolith: sehr starke Kopplung, keine Kohäsion
- Modulith: mäßige Kopplung, gewisse Kohäsion
- Services: losere Kopplung, partiell hohe Kohäsion; mehrere Sprachen/Frameworks möglich
- Microservices: lose Kopplung, hohe Kohäsion; mehrere Sprachen/Frameworks/Images möglich

VERGLEICH - AUSFÜHRBARKEIT

- Ausprobieren neuer Features
- Nachstellen von Fehler
- Aufsetzen der Umgebung

VERGLEICH - AUSFÜHRBARKEIT

Monolith	Modulith	Services	Microservices
Hoch	Hoch	Mittel	Gering

Speaker notes

- Monolith: nur eine Instanz benötigt
- Modulith: s.h. Monolith
- Services: mehrere Instanzen benötigt
- Microservices: mehrere Instanzen und Umgebung benötigt

VERGLEICH - TESTBARKEIT

- Validierung der Korrektheit
- Absichern von Entwicklungen

VERGLEICH - TESTBARKEIT

Monolith	Modulith	Services	Microservices
Gering	Mittel	Mittel	Hoch

Speaker notes

- Monolith: nur in Gesamtheit testbar
- Modulith: einzelne Module testbar, Kern nur in Gesamtheit
- Services: einzelne Module/Services testbar
- Microservices: einzelne Services modular testbar

VERGLEICH - SKALIERBARKEIT

- Reaktionsfähigkeit bei Fluktuationen
- Effiziente Nutzung der Resourcen

VERGLEICH - SKALIERBARKEIT

Monolith	Modulith	Services	Microservices
Keine	Gering	Mittel	Hoch

Speaker notes

- Monolith: nicht skalierbar
- Modulith: einzelne Module über Threading skalierbar
- Services: einzelne Services können skaliert werden
- Microservices: alle Services können skaliert werden

VERGLEICH - ZUVERLÄSSIGKEIT

- Störungsanfälligkeit
- Kommunikationsabbrüche
- Fehlerhafte Zustände
- Netzwerke, Hardware, Software

VERGLEICH - ZUVERLÄSSIGKEIT

Monolith	Modulith	Services	Microservices
Hoch	Hoch	Mittel	Gering

Speaker notes

- Monolith: kaum Netzwerkverbindung; keine Interaktion zwischen Servergruppen; wenige Teile
- Modulith: s.h Monolith
- Services: Kommunikation zwischen Kern und Service anfällig; mehrere Teile
- Microservices: Netzwerkvirtualisierung und Servergruppen anfällig; viele Teile

VERGLEICH - AUSFALLSICHERHEIT

- Ausfallsicherheit
- Redundanz

VERGLEICH - AUSFALLSICHERHEIT

Monolith	Modulith	Services	Microservices
Gering	Gering	Mittel	Hoch

Speaker notes

- Monolith: Single-Point-of-Failure
- Modulith: s.h Monolith
- Services: Kern Single-Point-of-Failure, Services redundant
- Microservices: alle Services redundant

VERGLEICH - ZUSAMMENFASSUNG

	Monolith	Modulith	Services	Microservices
Initialaufwand	Gering	Mittel	Mittel	Hoch
Wartungsaufwand	Hoch	Mittel	Mittel	Gering
Betriebsaufwand	Gering	Gering	Mittel	Hoch
Personalaufwand	Gering	Mittel	Mittel	Hoch
Abhängigkeit	Hoch	Hoch	Mittel	Gering
Ausführbarkeit	Hoch	Hoch	Mittel	Gering
Testbarkeit	Gering	Mittel	Mittel	Hoch
Skalierbarkeit	Keine	Gering	Mittel	Hoch
Zuverlässigkeit	Hoch	Hoch	Mittle	Gering
Ausfallsicherheit	Gering	Gering	Mittel	Hoch

VERGLEICH - ANFORDERUNGEN

Monolith	Modulith	Services	Microservi
Unbekannt - Einfach	Einfach - Umfangreich	Umfangreich - Komplex	Komplex

Speaker notes

- Anforderungen und Teamgröße limitieren Architekturmöglichkeiten
- Architektur nach Konvergierung von Anforderungen und Personalaufwand wählen
- Teamgröße muss sich mit Anforderungen decken

VERGLEICH - TEAMGRÖSSE

Monolith	Modulith	Services	Microservices
Klein	Klein - Groß	Mittel - Groß	Groß - Mehrere

Speaker notes

- Anforderungen und Teamgröße limitieren Architekturmöglichkeiten
- Architektur nach Konvergierung von Anforderungen und Personalaufwand wählen
- Teamgröße muss sich mit Anforderungen decken

VERGLEICH - FAZIT

- Anforderungen und Teamgröße limitieren jeweils Architekturmöglichkeiten
- Architektur aus Deckung der Architekturmöglichkeiten wählen
- Teamgröße muss sich mit Anforderungen decken

VERGLEICH - FAZIT

- Monolith für unbekannte Projekte
- Modulith für mehr Wartbarkeit
- Services für Skalierbarkeit
- Microservices für Zuverlässigkeit

SPRING

SPRING

- Application Framework
- Dependency-Injection-Container

SPRING-BOOT

- Basiert auf Spring
- Erweitert um Java EE
- Convention-over-Configuration
- Annotation-Base Configuration
- Spring ursprünglich eigentlich XML

Speaker notes

- Spring besteht aus Modulen die miteinander kombiniert werden können
- Spring Boot erweitert Spring um Java EE (Servlets)
- Standardkonfiguration wird bevorzugt, am besten keine komplizierte Konfiguration
- Abweichend davon kostet es Konfigurationsaufwand

BOOTSTRAP

- Aufbau des Objektgraphen
- Zwei primäre Quellen für Objekte
- Components
- Configurations
- Objektgraph ist normalerweise statisch
- Objektgraph erlaubt dynamische Erweiterung

Speaker notes

- Objektgraphen ist der Graph den der Dependency Injection Container aufbaut
- Quellen -> Einstiegspunkte für den Graphen
- zu jeder Zeit können Objekte zur Laufzeit hinzugefügt werden

BOOTSTRAP

BOOTSTRAP - VERWENDUNG

- `@SpringBootApplication` zur Deklaration des Einstiegspunkt
- `@ComponentScan` für komplexere Umstände

BOOTSTRAP - BEISPIEL

```
@SpringBootApplication
public class MySpringApplication {
    public static void main(String[] args) {
        SpringApplication.run(MySpringApplication.class, args)
    }
}
```

BOOTSTRAP - DETAILS

`@SpringBootApplication`

- `scanBasePackages` Base-Package für alle Configurations und Components
- Default ist das aktuelle Package

COMPONENTS

- Direkte Deklaration von Objekten
- Erzeugung durch den Dependency-Injection-Container

COMPONENTS - VERWENDUNG

- `@Component` zur Deklaration
- `@Order` zur Definition der Präzedenz

Speaker notes

- @Order wird nur gebraucht wenn man bestehende Components einer Library überschreiben möchte

COMPONENTS - BEISPIEL

```
@Component
public class MyComponent {
    ...
}
```

COMPONENTS - ALIASE

- `@Controller` für Endpoints
- `@RestController` für ReST-Endpoints
- `@Services` für Services
- `@Repository` für Datenbankschnittstellen

CONFIGURATIONS

- Indirekte Deklaration von Objekten
- Sowie Ändern und Erweitern bestehender Objekte
- Aufruf durch den Dependency-Injection-Container

CONFIGURATIONS - VERWENDUNG

- `@Configuration` zur Deklaration einer Konfiguration
- `@Bean` zur Deklaration eines Objektes
- `@Order` zur Definition der Präzedenz

CONFIGURATIONS - BEISPIEL

```
@Configuration
public class MyConfiguration {
    @Bean
    public MyComponent createComponent(){
        ...
    }
}
```

REFERENZIERUNG

- Benötigt Aufruf durch Dependency-Injection-Container
- Auflösung der Referenzen über Typ
- Mehrfach vorhandene Objekt über Namen ggf. Classifier
- Boostrap scheitert wenn Referenz nicht auslösbar
- keine entsprechendes Objekt
- mehrere entsprechende Objekte

REFERENZIERUNG - VERWENDUNG

- `@Autowired` zur Markierung eines Parameters

REFERENZIERUNG - BEISPIEL

```
@Component
public class MyComponentWithDependency {
    public MyComponentWithDependency(
        @Autowired MyRequiredComponent component
    ) {
        ...
    }
}
```

REFERENZIERUNG - BEISPIEL

```
@Configuration
public class MyConfiguration {
    @Bean
    public MyComponentWithDependency createDependantComponent(
        @Autowired MyRequiredComponent component
    ) {
        ...
    }
}
```

REFERENZIERUNG - DETAILS

`@Autowired`

- `required` für optionale Objekte

Speaker notes

- required ist ein Attribut von Autowired
- wird z.B. für Libraries verwendet

SPRING SCHICHTEN

SPRING SCHICHTEN

- Frontend
- Middleware
- Backend

CONTROLLER

- Schnittstelle zur Außenwelt
- Abstraktes Konstrukt
- Verschiedene Arten von Schnittstellen möglich (ReST, GraphQL etc.)

CONTROLLER - EINORDNUNG

- Frontend
- Referenziert Services
- Wird von niemanden referenziert

CONTROLLER - VERWENDUNG

- `@Controller` zur Deklaration

REST-CONTROLLER

- Konkrete Ausprägung eines Controllers
- ReST basiert
- Definiert die Endpoints der Anwendung

REST-CONTROLLER - VERWENDUNG

- `@RestController` zur Deklaration

REST-CONTROLLER - VERWENDUNG

- `@RequestMapping` zur Definition des Endpoints
- `@PathVariable` für Pfad-Variablen
- `@RequestParam` für Query-Parameter
- `@RequestBody` für Bodies
- `@ResponseStatus` für besondere Http-Status

REST-CONTROLLER - BEISPIEL

```
@RestController
@RequestMapping(
    path = "/notes",
    produces = MediaType.APPLICATION_JSON
)
public class MyNoteController {
    ...
}
```

REST-CONTROLLER - BEISPIEL

Notizen holen `GET /notes`

Notizen suchen `GET /notes?q={search}`

REST-CONTROLLER - BEISPIEL

```
@RequestMapping(method = RequestMethod.GET)
public List<Note> getNotes(
    @RequestParam(name = "q", required = false) String search
) {
    ...
}
```

REST-CONTROLLER - BEISPIEL

Notiz anlegen `POST /notes`

REST-CONTROLLER - BEISPIEL

```
@RequestMapping(method = RequestMethod.POST)
public Note createNote(@RequestBody NoteProposal proposal) {
    ...
}
```

REST-CONTROLLER - BEISPIEL

Notiz ändern `PUT /notes/{note}`

REST-CONTROLLER - BEISPIEL

```
@RequestMapping(path = "/{note}", method = RequestMethod.PUT)
public Note updateNote(
    @PathVariable("note") Long noteId,
    @RequestBody NoteProposal proposal
) {
    ...
}
```

REST-CONTROLLER - BEISPIEL

Notiz ändern `DELETE /notes/{note}`

REST-CONTROLLER - BEISPIEL

```
@RequestMapping(
    path = "/{note}",
    method = RequestMethod.DELETE
)
public Note deleteNote(
    @PathVariable("note") Long noteId
) {
    ...
}
```

REST-CONTROLLER - BEISPIEL

Anhang holen `GET`

`/notes/{note}/attachment/{attachment}`

REST-CONTROLLER - BEISPIEL

```
@RequestMapping(
    path = "/{note}/attachment/{attachment}",
    method = RequestMethod.GET,
    produces = MediaType.APPLICATION_OCTET_STREAM_VALUE
)
public byte[] getAttachment(
    @PathVariable("note") Long noteId,
    @PathVariable("attachment") String attachmentId
) {
    ...
}
```

REST-CONTROLLER - DETAILS

`@RequestMapping`

- `path` Pfad ink. Pfad-Variablen
- `method` Erwartete Methode
- `consumes` Erwarteter Content-Type
- `produces` Erzeugter Content-Type

VALIDATION

- Überprüfung der Eingaben
- Client ist nicht vertrauenswürdig
- Spring unterstützt `javax.validation` Annotations

VALIDATION - VERWENDUNG

- `@Valid` zur Markierung von zu validierenden Parametern
- `@NotNull`
- `@NotBlank` mindestens EIN nicht Whitespace-Charakter
- `@Size` Einschränkung der Länge von Strings & Collections
- `@Min`, `@Max` Einschränkung des numerischen Wertebereichs
- `@Email` erwartet eine valide Email-Adresse

Speaker notes

- Für ausführliche Informationen über javax.validation: https://www.baeldung.com/javax-validation

VALIDATION - BEISPIEL

```
public class Note {
    ...
    @NotBlank(message = "description must not be blank")
    private String description;
    ...
}
```

VALIDATION - BEISPIEL

```
@RequestMapping(method = RequestMethod.POST)
public Note createNote(@RequestBody @Valid NoteProposal propos
    ...
}
```

SERVICE

- Implementiert Businesslogik
- Oftmals durch ein Interface abstrahiert

SERVICE - EINORDNUNG

- Middleware
- Referenziert Repositories und andere Services
- Wird von Controller und Services referenziert

SERVICE - VERWENDUNG

- `@Service` zur Deklaration

SERVICE - BEISPIEL

```
@Service
public interface MyNoteService {
    ...
}
```

SERVICE - BEISPIEL

```
@Service
public class MyNoteServiceImpl implements MyNoteService {
    ...
}
```

REPOSITORY

- Implementiert Datenbankschnittstelle für eine Entity
- Abstraktes Konstrukt
- Verschiedene Arten von Datenbankschnittstelle möglich (JPA, ElasticSearch etc.)

REPOSITORY - EINORDNUNG

- Backend
- Referenziert andere Repositories
- Wird von Services referenziert

REPOSITORY - VERWENDUNG

- `@Repository` zur Deklaration

JPA-REPOSITORY

- Basiert auf Java-Persistence-API
- Implementation per Proxy
- Erweiterung durch Annotationen

JPA-REPOSITORY - VERWENDUNG

- `@Repository` zur Deklaration
- `@Query` zur Definition komplexer Queries

JPA-REPOSITORY - BEISPIEL

```
@Repository
public interface MyNoteRepository
    extends JpaRepository<Note, Long> {
    ...
}
```

JPA-REPOSITORY - BEISPIEL

```
List<Note> findAll();
```

JPA-REPOSITORY - BEISPIEL

```
Note findById(Long id);
```

JPA-REPOSITORY - BEISPIEL

```
Note findByNameAndDescription(
    String name,
    String description
);
```

JPA-REPOSITORY - BEISPIEL

```
@Query("SELECT n FROM Notes n "
    + "WHERE n.tag IN (:tags) "
    + "AND n.creationDate >= :timestamp")
List<Note> findWithTagsAfter(
    String[] tags,
    OffsetDateTime timestamp
);
```

JPA-REPOSITORY - BETTER PRACTICE

- Vielzahl an vordefinierten Operationen
- Wrapper-Klasse für explizite Schnittstellen
- Mehr Aufwand - Mehr Konsistenz
- Projekt-spezifisches Wording
- Verändern der Methodensignatur
- Keine ungewollten Operationen

JPA-REPOSITORY - BEISPIEL

```
@Repository
public class MyNoteRepository {
    private final MySpringNoteRepository delegate;

    public MyNoteRepository(
        @Autowired MySpringNoteRepository delegate
    ) {
        this.delegate = delegate;
    }

    public @Nullable Note find(@NotNull Long id) {
        return delegate.findById(id).orElse(null)
    }
}
```

JPA - ÜBERSICHT

- Objekt-Relationales-Mapping
- Entities
- Abbildungen von Objekten auf Tabellen
- Transaktionsmanagement
- Aggregation von zusammengehörigen Änderungen
- Gewährleistung von Datenintegrität

JPA - ENTITIES

- `@Entity` zur Deklaration eine Entity
- `@Id` zur Markierung des ID-Feldes
- `@GeneratedValue` zur automatischen Generierung
- usw.

Speaker notes

- JPA Annotations sind zahlreich
- Objekt-Relationales-Mapping mithilfe von JPA ist nicht Teil des Rahmens der Vorlesung
- Für ausführliche Informationen zu JPA: https://www.baeldung.com/learn-jpa-hibernate

PRAXIS

PRAXIS - API

- TODOs abfragen
- TODO anlegen
- TODO als Done markieren
- TODO löschen

PRAXIS - API

TODOs abfragen `GET /todo`

PRAXIS - API

TODO anlegen `POST /todo`

PRAXIS - API

TODO als Done markieren `PUT /todo/{id}`

PRAXIS - API

TODO löschen `DELETE /todo/{id}`

PRAXIS - REST-CONTROLLER

- Klasse
  - Annotations hinzufügen
  - Repository injecten
- Methoden
  - anhand von API modellieren
  - Annotations hinzufügen
  - Repository aufrufen

PRAXIS - REPOSITORY

- Klasse
  - Annotations hinzufügen
  - Spring-Repository injecten
- Methoden
  - anhand der Anforderungen modellieren
  - an Spring-Repository weiterleiten

PRAXIS - VALIDIERUNG

- `Todo.title`
  - nicht Leer
  - maximal 50 Zeichen