RICH CLIENT: SERVER ANWENDUNG

ÜBERLEITUNG

VERANTWORTLICHKEITEN - JSF

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

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

WEBSERVICE - STATELESS

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

WEBSERVICE - SKALIERBAR

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

WEBSERVICE - UNTRUSTING

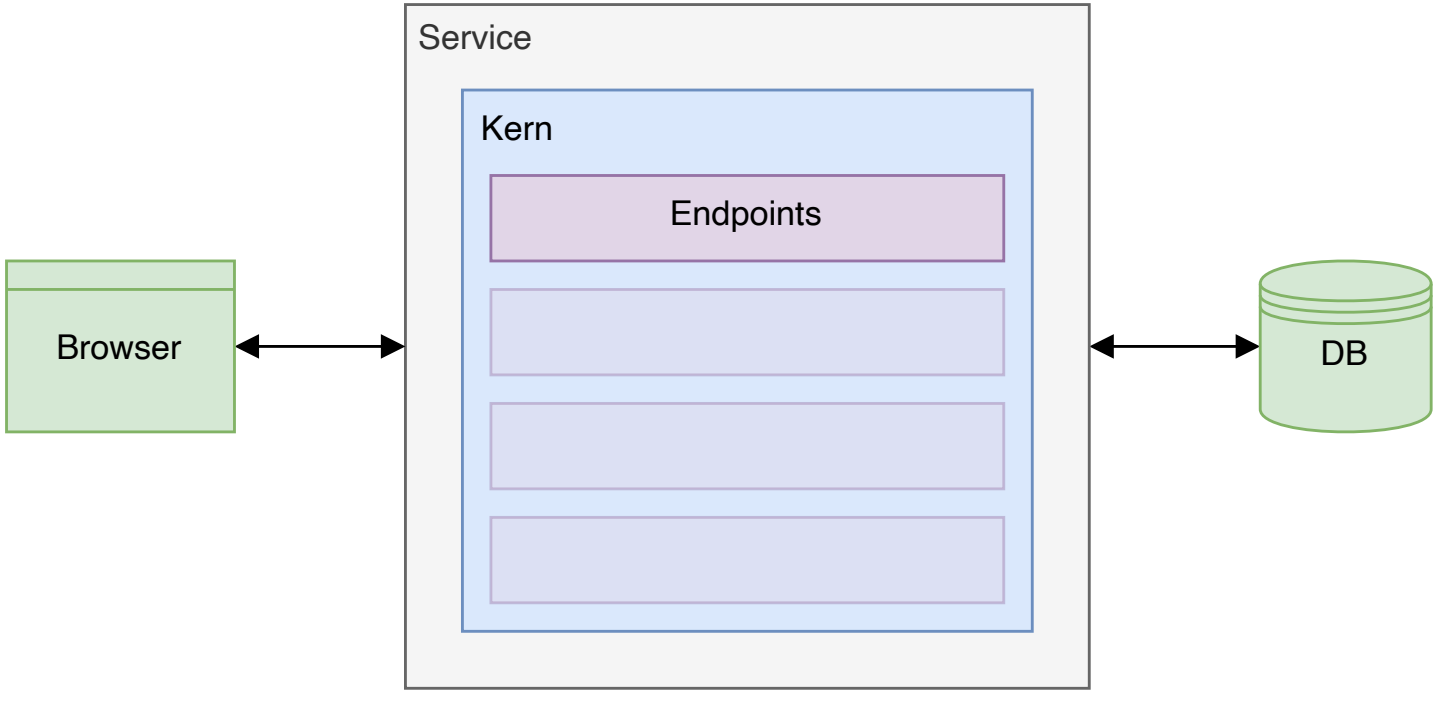
- Validierung aller Eingaben
- Isolierung aller Eingaben
- Durchgehende Prüfung der Authorisierung

ARCHITEKTUREN

ARCHITEKTUREN

- Monolith
- Modulith
- Services
- Microservice

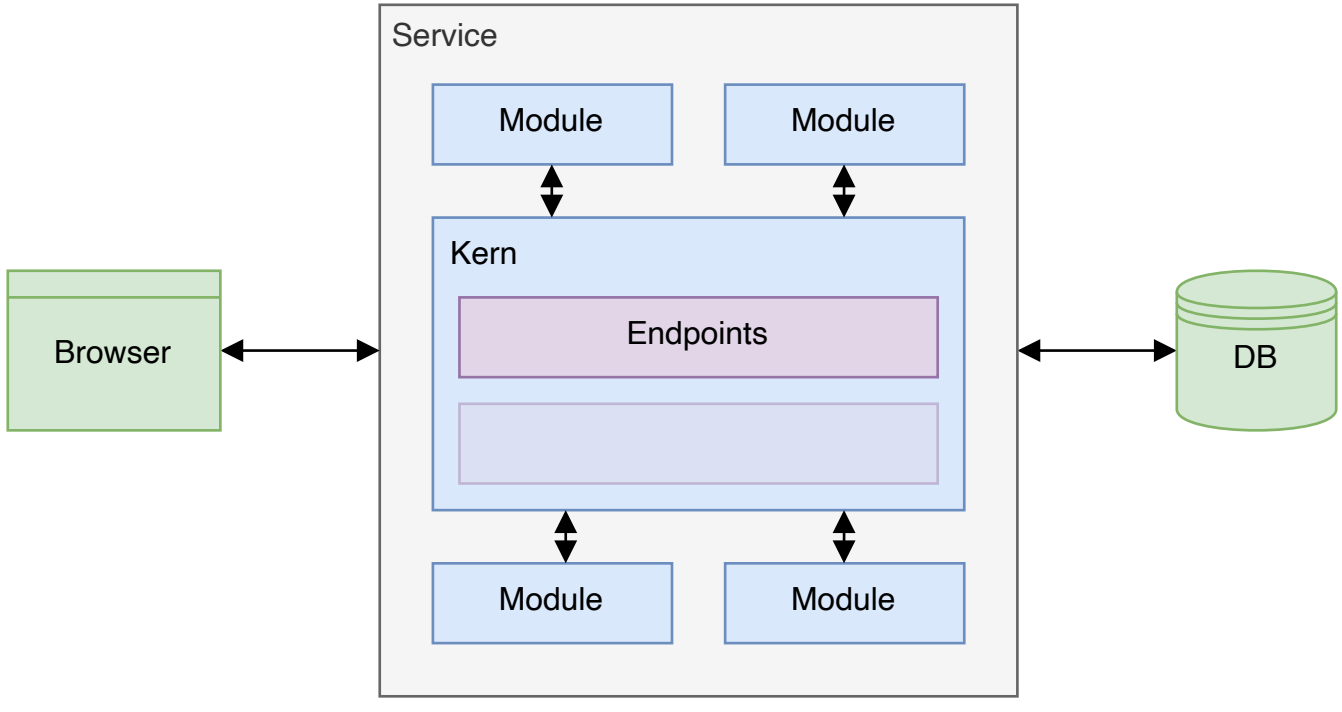
ARCHITEKTUREN - MONOLITH



ARCHITEKTUREN - MONOLITH

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

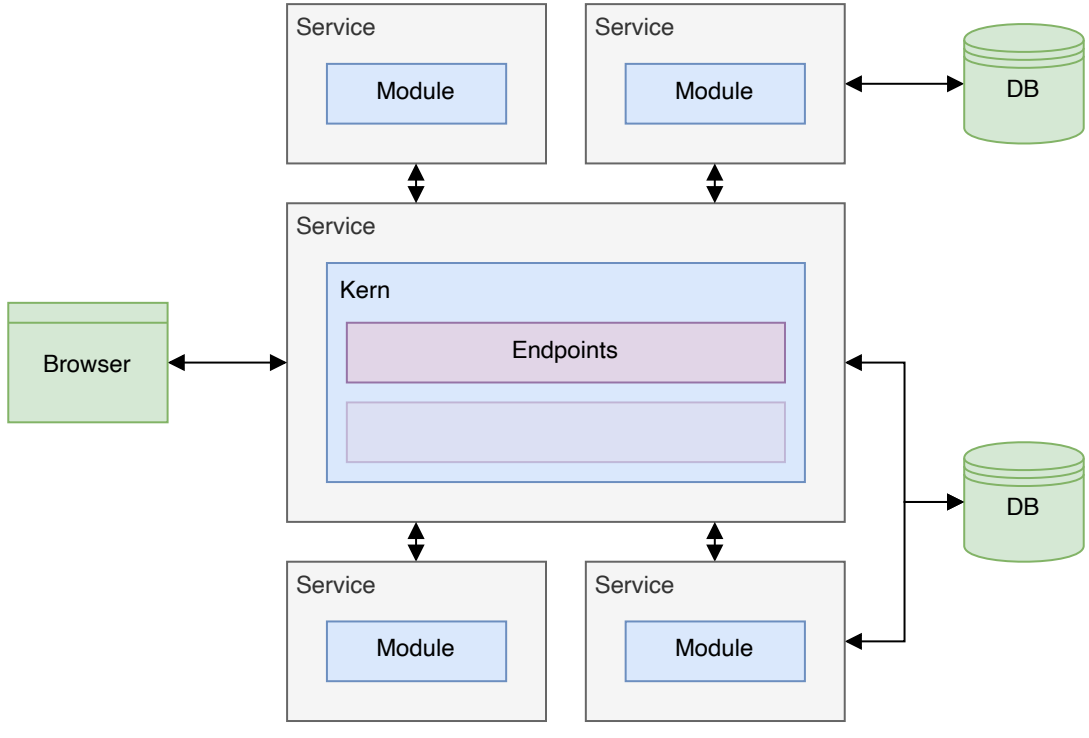
ARCHITEKTUREN - MODULITH



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

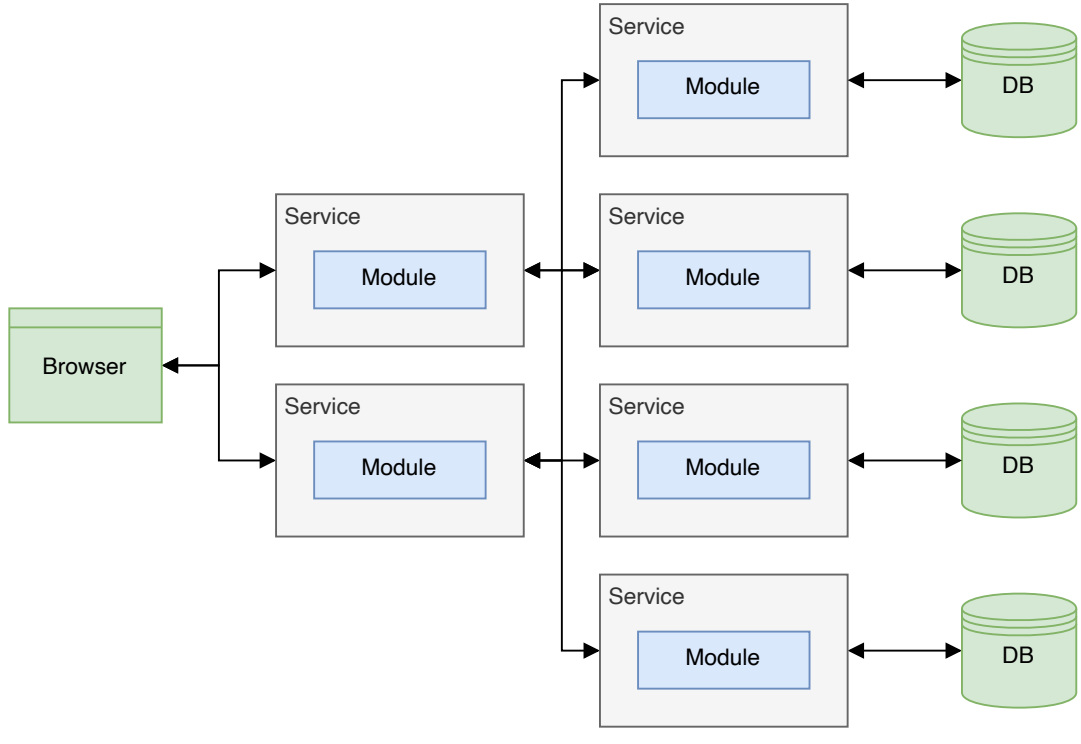
ARCHITEKTUREN - SERVICES



ARCHITEKTUREN - SERVICES

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

ARCHITEKTUREN - MICROSERVICES



ARCHITEKTUREN - MICROSERVICES

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

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

VERGLEICH - WARTUNGSAUFWAND

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

VERGLEICH - WARTUNGSAUFWAND

Monolith	Modulith	Services	Microservices
Hoch	Mittel	Mittel	Gering

VERGLEICH - BETRIEBSAUFWAND

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

VERGLEICH - BETRIEBSAUFWAND

Monolith	Modulith	Services	Microservices
Gering	Gering	Mittel	Hoch

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

VERGLEICH - ABHÄNGIGKEIT

- Trennung der Fachlichkeiten
- Freiheit der Technologien

VERGLEICH - ABHÄNGIGKEIT

Monolith	Modulith	Services	Microservices
Hoch	Hoch	Mittel	Gering

VERGLEICH - AUSFÜHRBARKEIT

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

VERGLEICH - AUSFÜHRBARKEIT

Monolith	Modulith	Services	Microservices
Hoch	Hoch	Mittel	Gering

VERGLEICH - TESTBARKEIT

- Validierung der Korrektheit
- Absichern von Entwicklungen

VERGLEICH - TESTBARKEIT

Monolith	Modulith	Services	Microservices
Gering	Mittel	Mittel	Hoch

VERGLEICH - SKALIERBARKEIT

- Reaktionsfähigkeit bei Fluktuationen
- Effiziente Nutzung der Resourcen

VERGLEICH - SKALIERBARKEIT

Monolith	Modulith	Services	Microservices
Keine	Gering	Mittel	Hoch

VERGLEICH - ZUVERLÄSSIGKEIT

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

VERGLEICH - ZUVERLÄSSIGKEIT

Monolith	Modulith	Services	Microservices
Hoch	Hoch	Mittel	Gering

VERGLEICH - AUSFALLSICHERHEIT

- Ausfallsicherheit
- Redundanz

VERGLEICH - AUSFALLSICHERHEIT

Monolith	Modulith	Services	Microservices
Gering	Gering	Mittel	Hoch

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

VERGLEICH - TEAMGRÖSSE

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

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

BOOTSTRAP

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

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

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

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
- `@QueryParam` 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

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)
}
}
```

TODO ANWENDUNG

TODO ANWENDUNG - ANFORDERUNGEN

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

TODO ANWENDUNG - ANFORDERUNGEN

TODOs abfragen `GET /todo`

TODO ANWENDUNG - ANFORDERUNGEN

TODO anlegen `POST /todo`

TODO ANWENDUNG - ANFORDERUNGEN

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

TODO ANWENDUNG - ANFORDERUNGEN

TODO löschen `DELETE /todo/{id}`