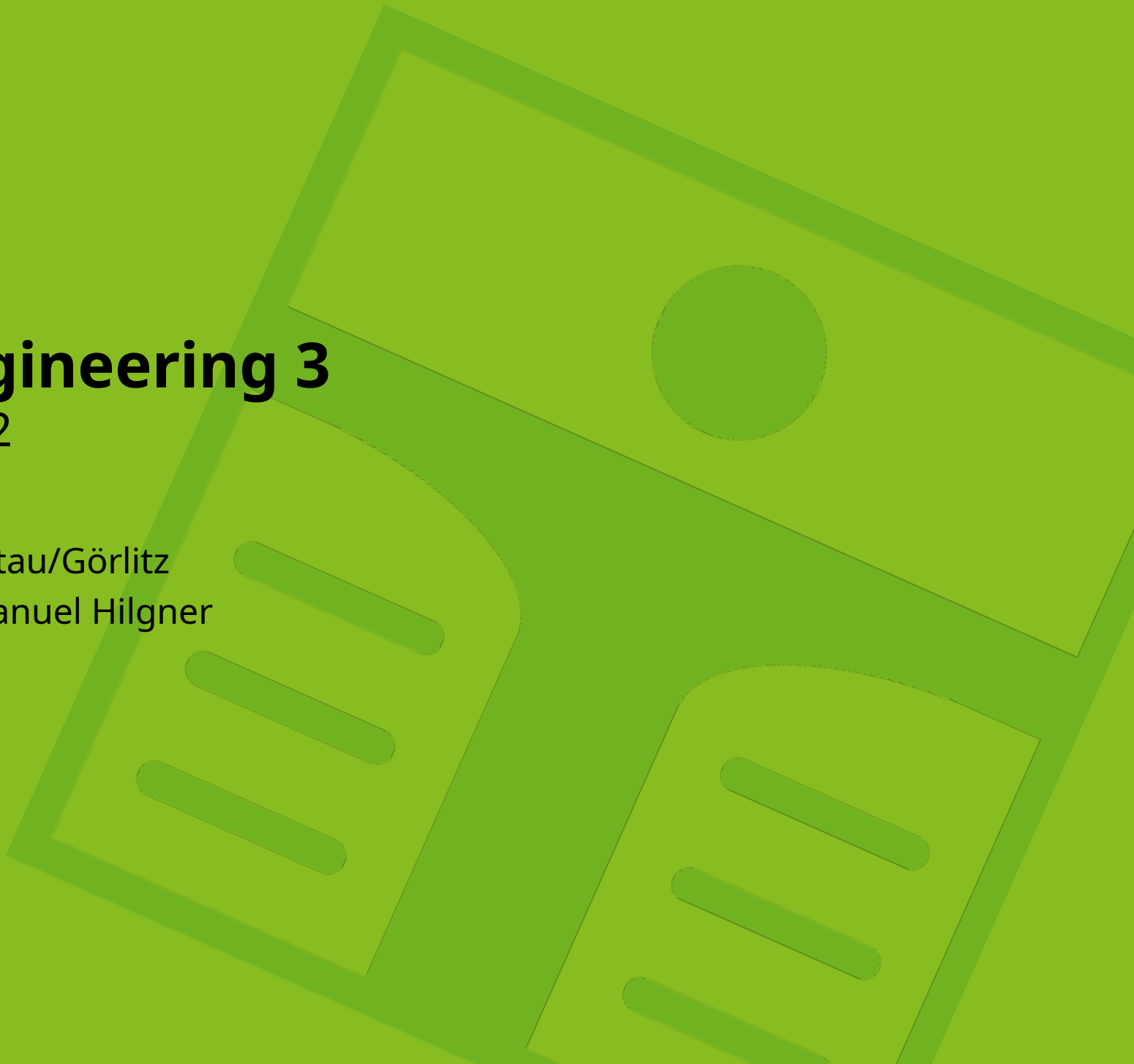
Web Engineering 3

Vorlesung 2

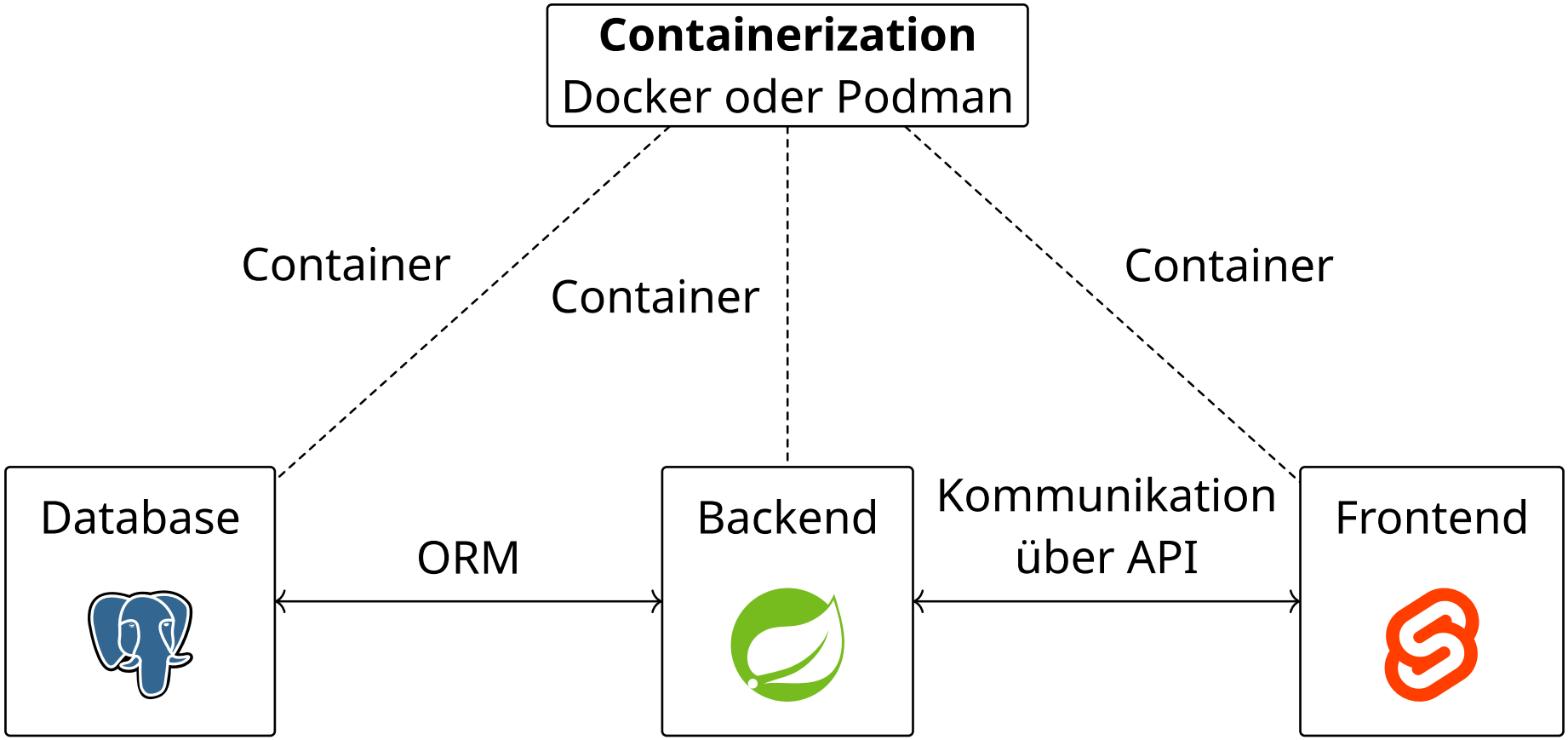
Hochschule Zittau/Görlitz
Christopher-Manuel Hilgner

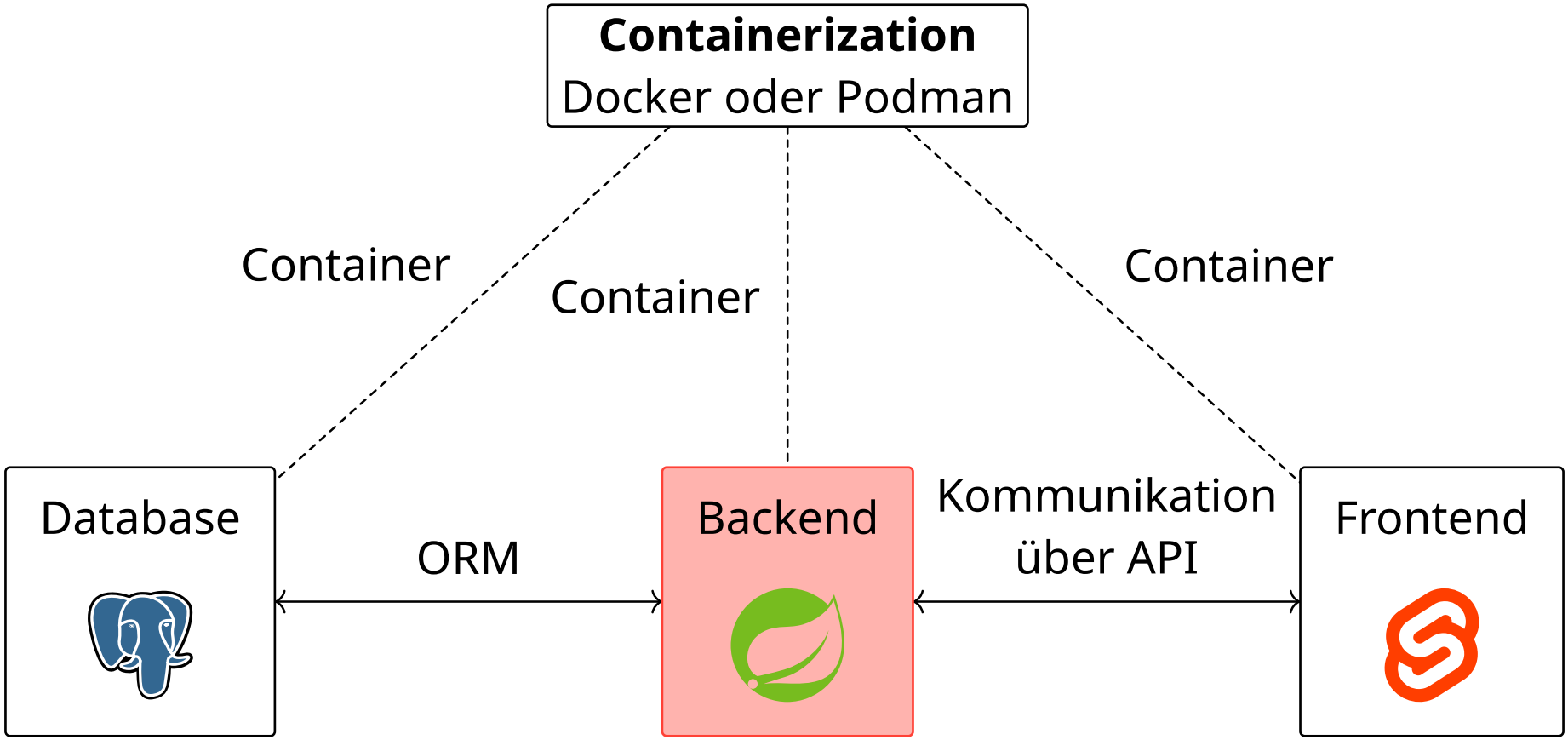


Agenda

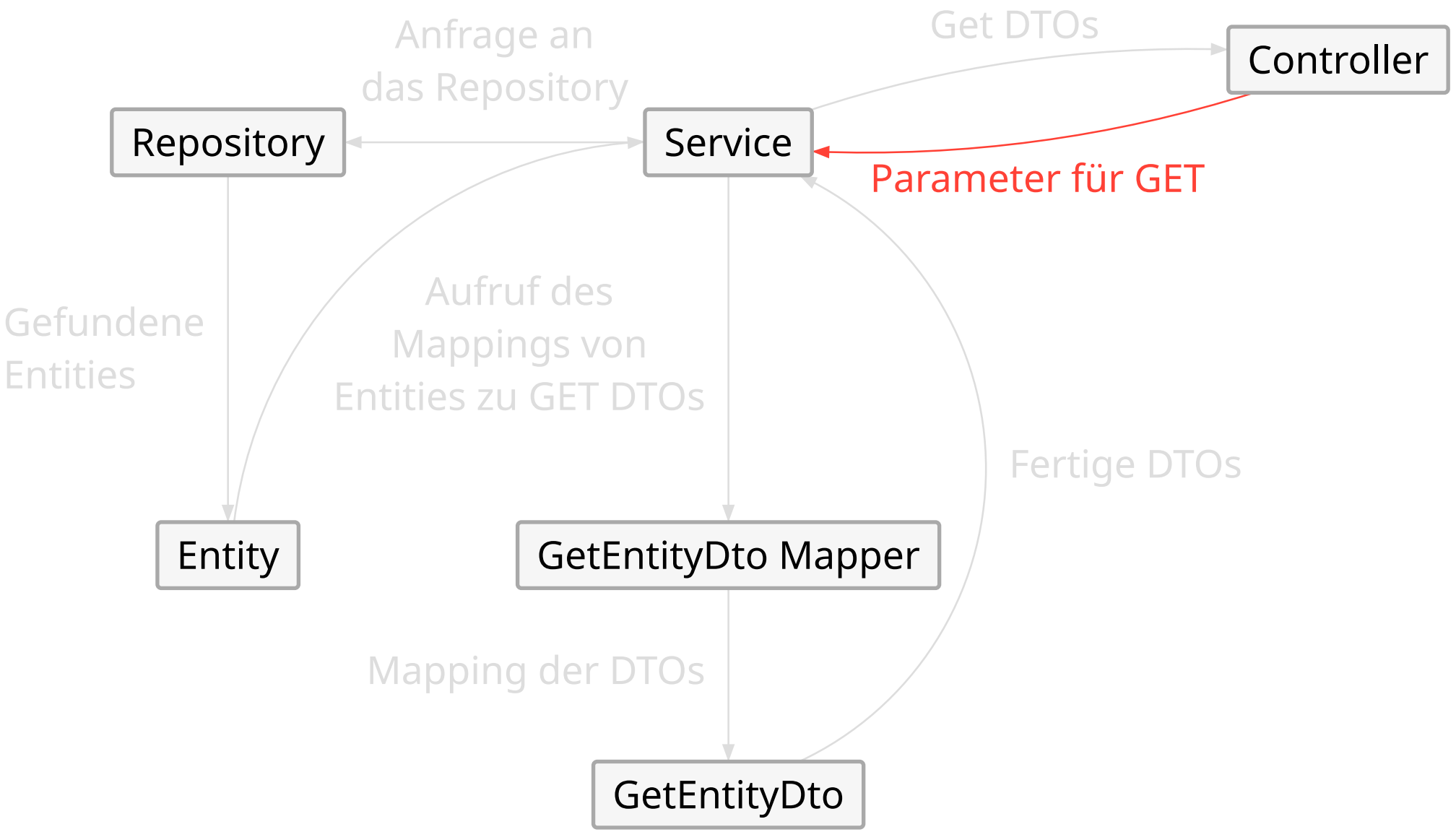
- Spring Komponenten
- Entity
- Repository
- Controller
- Service
- DTO
- Mapper

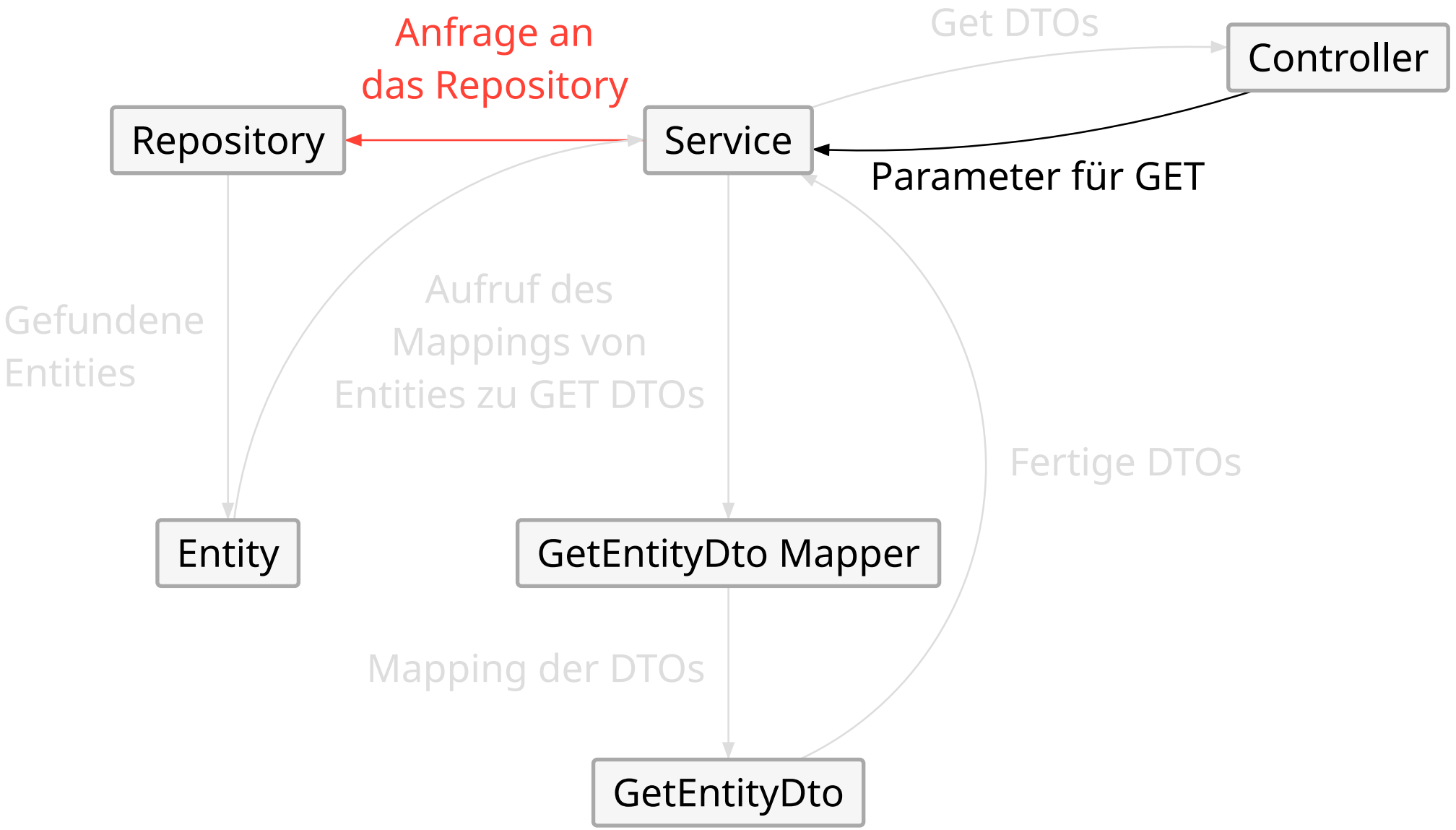
Letzte Woche

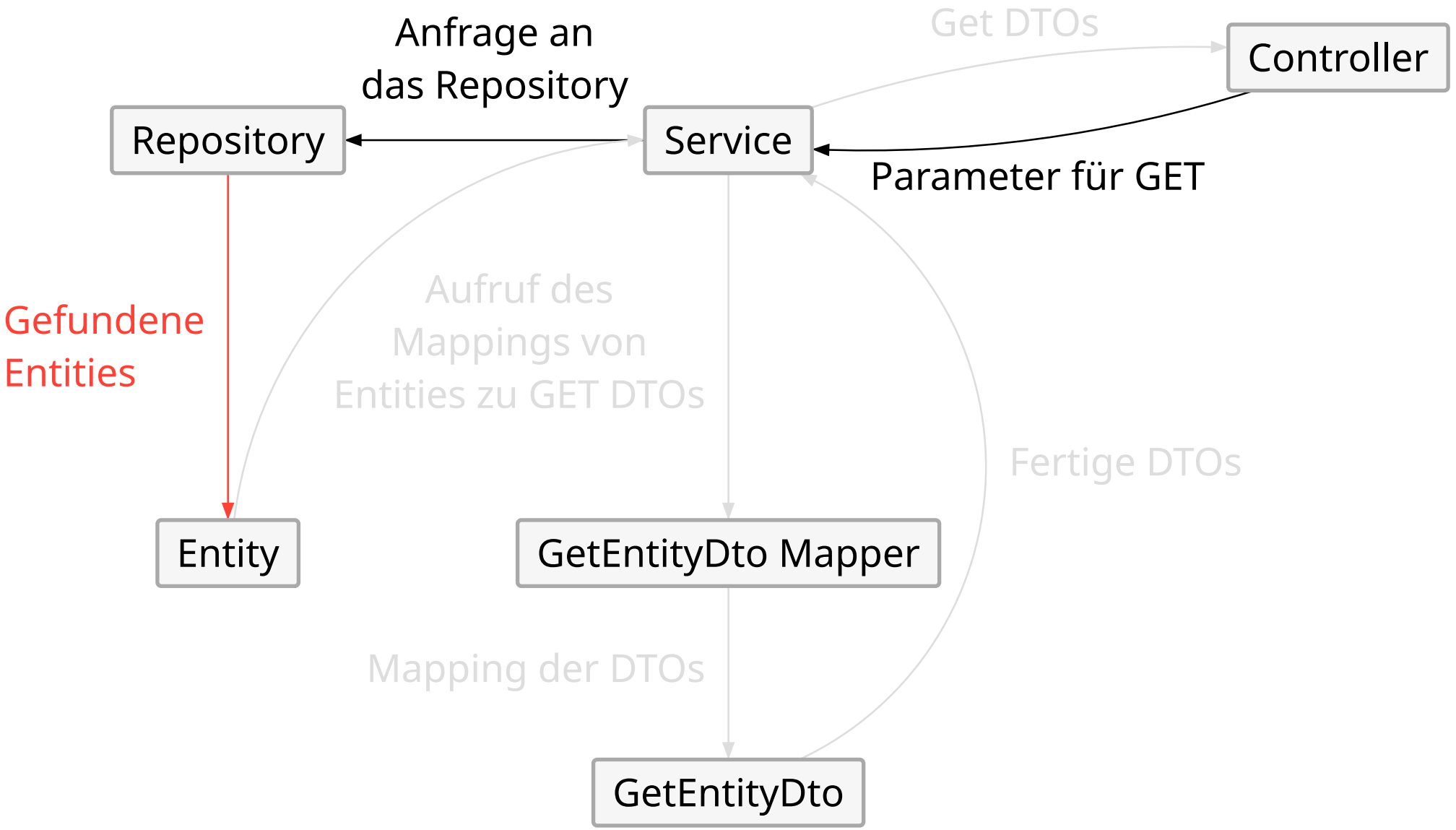


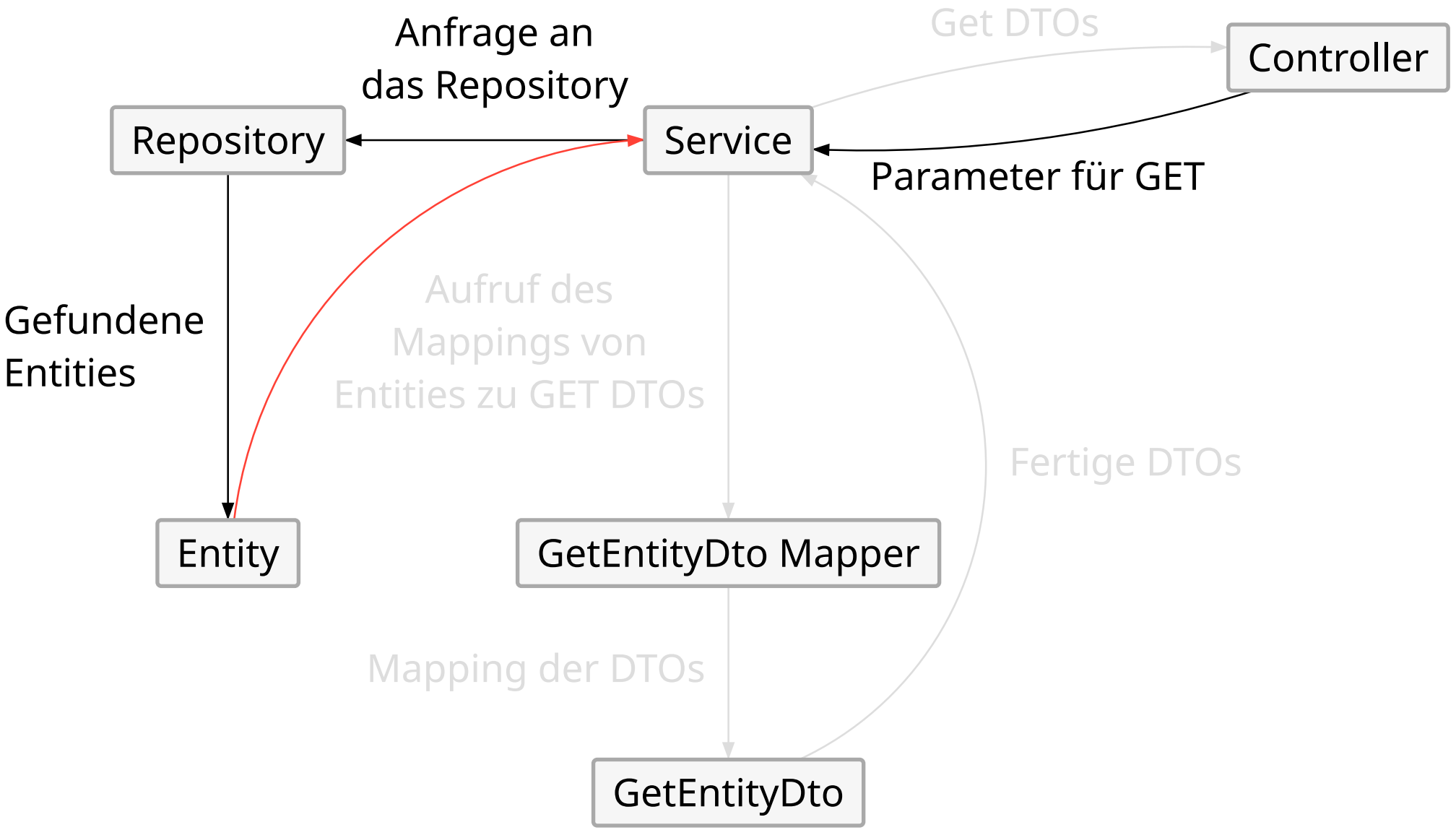

Containerization
Docker oder Podman
Container
Container
Container
Database
ORM
Backend
Kommunikation
über API
Frontend

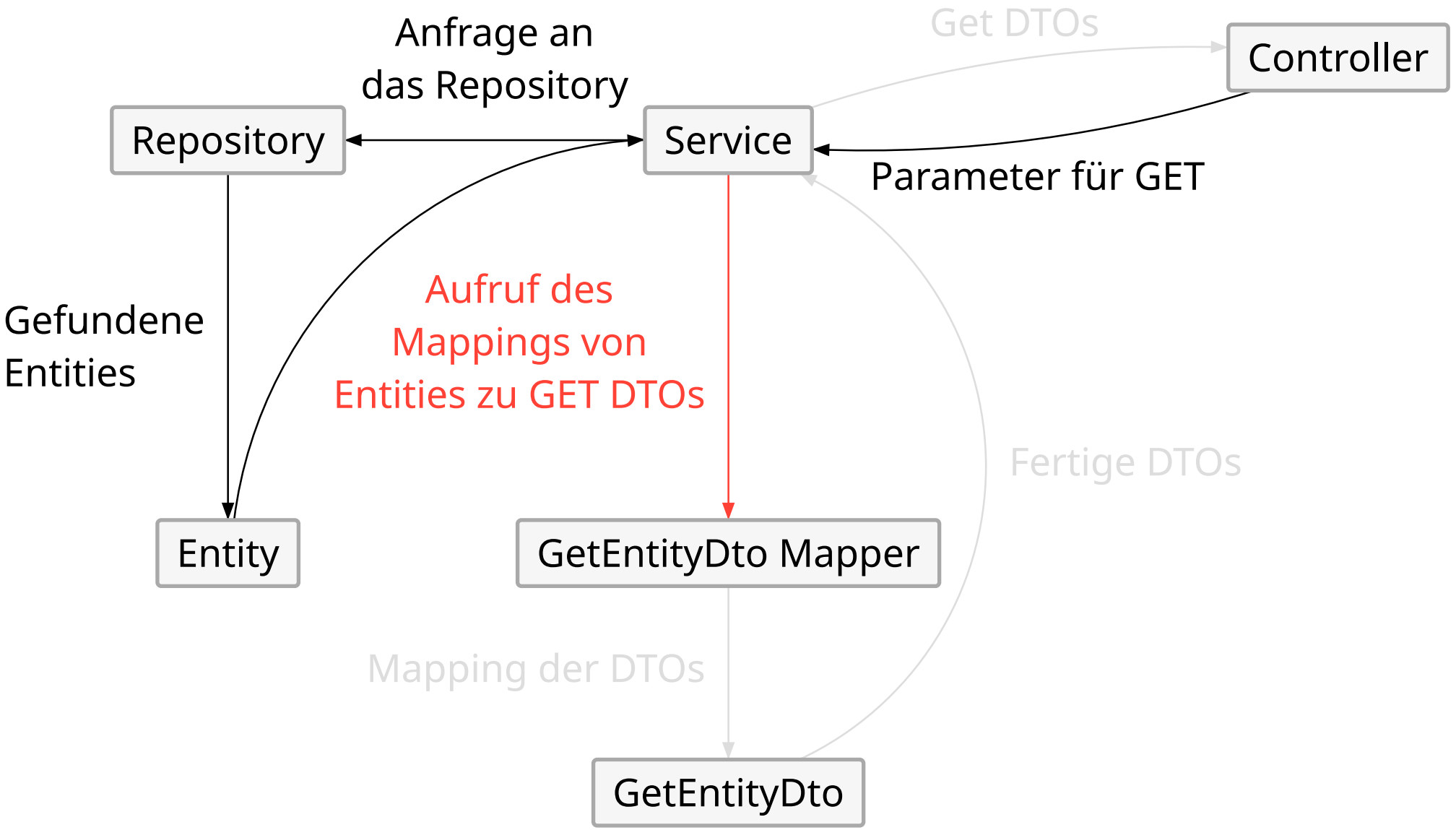
Komponenten und Ablauf einer GET Request

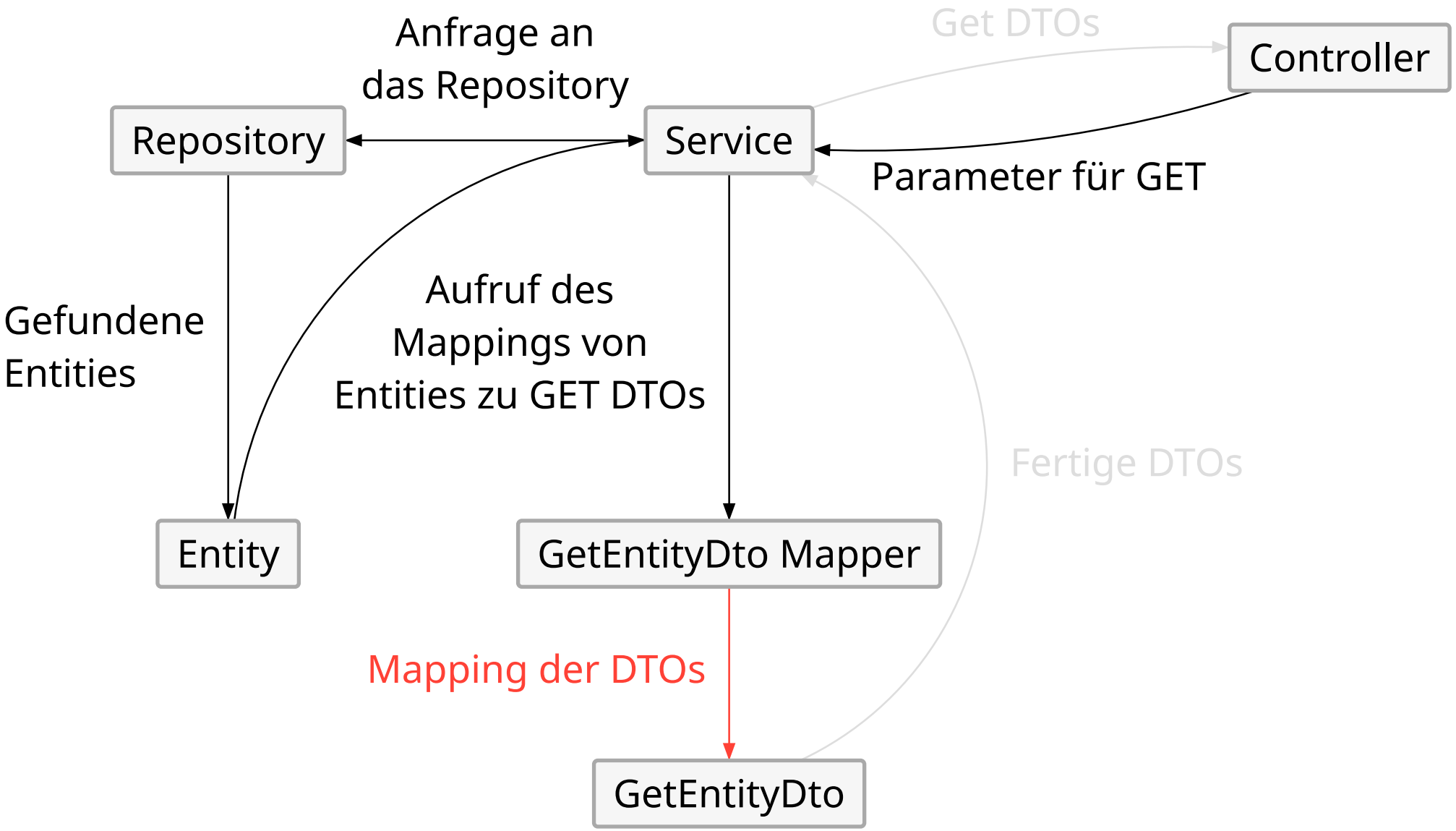

Anfrage an
das Repository
Get DTOs
Controller
Repository
Service
Parameter für GET
Gefundene
Entities
Aufruf des
Mappings von
Entities zu GET DTOs
Fertige DTOs
Entity
GetEntityDto Mapper
Mapping der DTOs
GetEntityDto

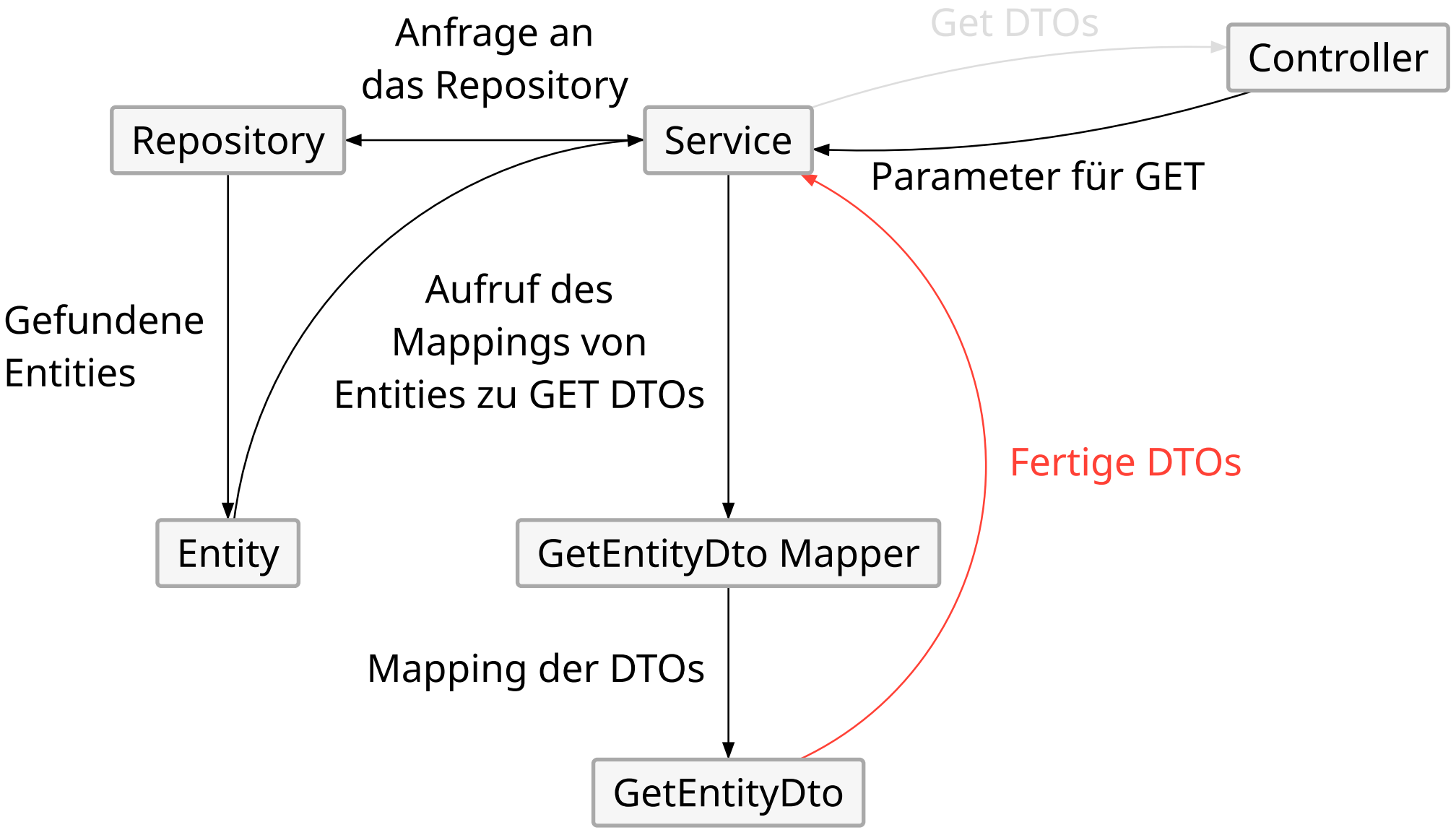

Anfrage an
das Repository
Get DTOs
Controller
Repository
Service
Parameter für GET
Gefundene
Entities
Aufruf des
Mappings von
Entities zu GET DTOs
Fertige DTOs
Entity
GetEntityDto Mapper
Mapping der DTOs
GetEntityDto

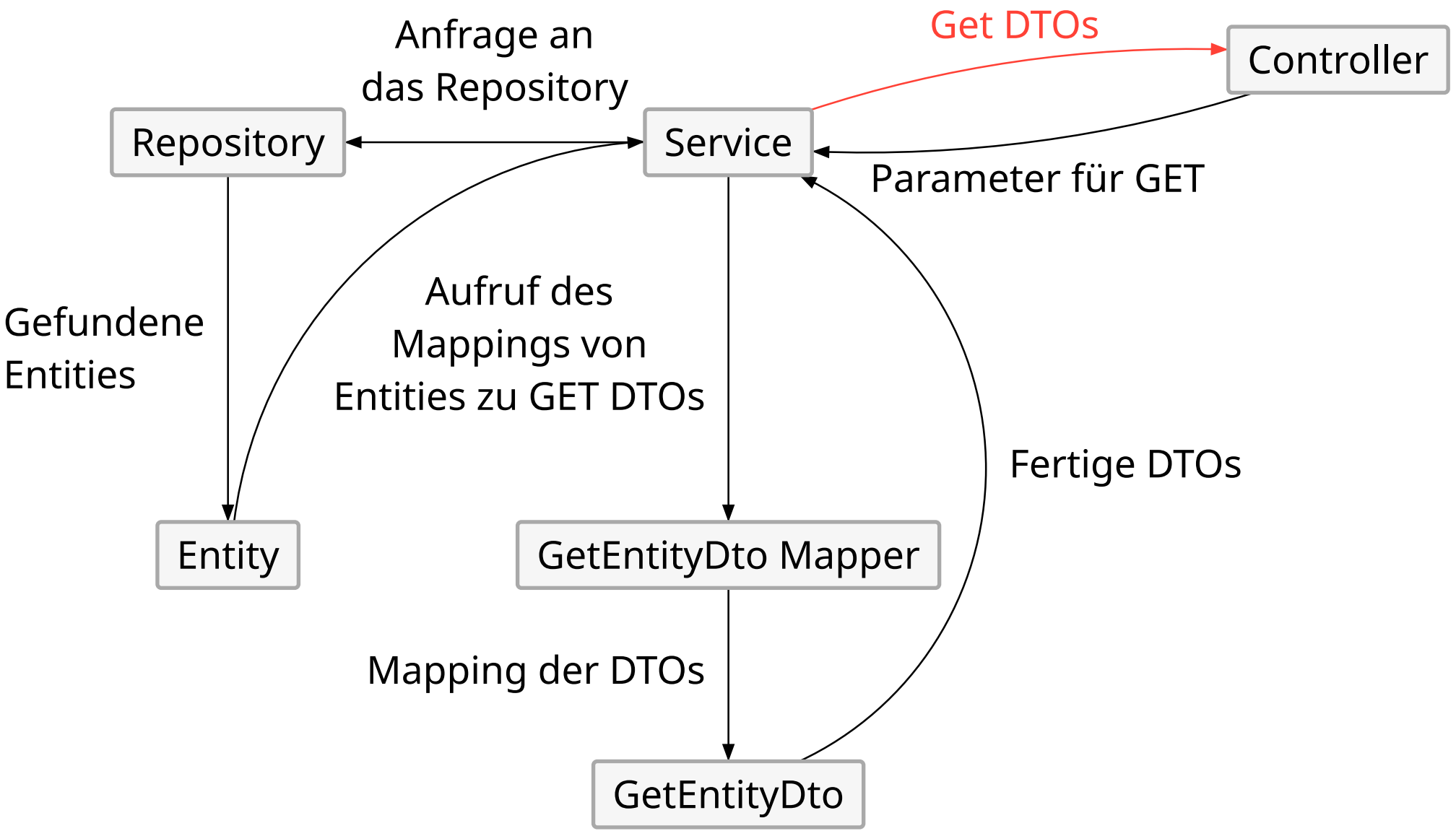

Anfrage an
das Repository
Get DTOs
Controller
Repository
Service
Parameter für GET
Gefundene
Entities
Aufruf des
Mappings von
Entities zu GET DTOs
Fertige DTOs
Entity
GetEntityDto Mapper
Mapping der DTOs
GetEntityDto


Anfrage an
das Repository
Get DTOs
Controller
Repository
Service
Parameter für GET
Gefundene
Entities
Aufruf des
Mappings von
Entities zu GET DTOs
Fertige DTOs
Entity
GetEntityDto Mapper
Mapping der DTOs
GetEntityDto


Anfrage an
das Repository
Get DTOs
Controller
Repository
Service
Parameter für GET
Gefundene
Entities
Aufruf des
Mappings von
Entities zu GET DTOs
Fertige DTOs
Entity
GetEntityDto Mapper
Mapping der DTOs
GetEntityDto


Anfrage an
das Repository
Get DTOs
Controller
Repository
Service
Parameter für GET
Gefundene
Entities
Aufruf des
Mappings von
Entities zu GET DTOs
Fertige DTOs
Entity
GetEntityDto Mapper
Mapping der DTOs
GetEntityDto


Anfrage an
das Repository
Get DTOs
Controller
Repository
Service
Parameter für GET
Gefundene
Entities
Aufruf des
Mappings von
Entities zu GET DTOs
Fertige DTOs
Entity
GetEntityDto Mapper
Mapping der DTOs
GetEntityDto


Get DTOs
Controller
Anfrage an
das Repository
Repository
Service
Parameter für GET
Gefundene
Entities
Aufruf des
Mappings von
Entities zu GET DTOs
Fertige DTOs
Entity
GetEntityDto Mapper
Mapping der DTOs
GetEntityDto

Komponenten

Entities

Repositories

Controllers

Services

DTOs

Mapper

Entities

Repositories

Controllers

Services

DTOs

Mapper

Entity

```kotlin
@Entity(name = "entityName")
@Table(name = "tableName")
class Entity {
  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  val id: Long? = null,
  // ...
}
```

`@Entity` Annotation

- Klasse wird zu einer Entity, wenn sie mit `@Entity` annotated ist
- Optionaler Name kann angegeben werden
- Name wird in Queries verwendet
- Wenn kein Name vorhanden ist, wird der Klassenname verwendet

```kotlin
@Entity(name = "student")
```

`@Table` Annotation

- Erlaubt definierung der Tabelle in der Datenbank
- Name und Schema können hier festgelegt werden
- Wenn `@Table` nicht vorhanden ist, wird eine Tabelle mit dem Entity Namen erstellt

```kotlin
@Table(name="STUDENT", schema="SCHOOL")
```

`@Id` Annotation

- Jede Entity benötigt einen Primary Key
- Datenfeld mit `@Id` wird zum Primary Key
- Die Id wird automatisch generiert
- Strategie zur Generierung kann mit `@GeneratedValue` festgelegt werden
  - AUTO
  - TABLE
  - SEQUENCE
  - IDENTITY
- Mit AUTO wird sich eine der anderen Strategien automatisch ausgewählt

`@Id` Annotation

Id einer Entity muss als **null** definiert sein

Hinweis 1

Eine Id muss nullable sein und als default Wert null besitzen. Wenn das nicht der Fall ist, wird Spring beim Erstellen einer neuen Entity einen Error werfen. Grund hierfür ist, dass eine definierte Id Spring davon ausgehen lässt, dass es diesen Eitrag bereits in der Datenbank gibt. Wenn die Id null ist, weiß Spring, dass es sich hier um einen neuen Eintrag handelt.

`@Id` Annotation

```kotlin
@Id
@GeneratedValue(strategy = GenerationType.AUTO)
val id: Long? = null
```

`@Column` Annotation

- Genauere Beschreibung von Datenfeldern
- Folgende Informationen können angegeben werden
  - `name`: Name des Columns in der Datenbank
  - `length`: Maximale Länge des Datenfeldes. Beispiel: Maximale Anzahl an Zeichen in einem String
  - `nullable`: Definiert, ob das Datenfeld null sein darf oder immer einen Wert besitzen muss
  - `unique`: Definiert, ob dieses Datenfeld ein unique Key sein soll

```kotlin
1    @Column(name = "dataField", nullable = false)
2    val dataField: Long = 0,
```

Referenzen auf andere Tables

- Relationen zwschen Entities
- Beispiel: User Entities und Roles Entities. Jedem User werden Roles zugewiesen
- `@OneToMany`, `@ManyToMany`, `@ManyToMany`, `@OneToOne`

`@OneToMany` und `@ManyToOne`

- Ein User kann eine Role besitzen, aber eine Role kann auf mehrere User zugewiesen werden
- User hier das Child und Role der Parent
- Child ist der Besitzer der Relation
- ID von der Role wird als Foreign Key im User gespeichert

```kotlin
@Entity(name = "user")
@Table(name = "user")
class User (

  @ManyToOne
  @JsonBackReference
  @JoinColumn(name = "role_id", referencedColumnName = "id")
  var role: Role? = null

)
```

```kotlin
@Entity(name = "role")
@Table(name = "role")
class Role(

  @OneToMany(
    mappedBy = "role"
  )
  var users: List<User> = emptyList()

)
```

`@OneToMany` und `@ManyToOne`

- Liste mit Usern wird mit `@OneToMany` annotated
- `mappedBy` gibt Variable an, die in der User Klasse genutzt wird
- `mappedBy` zeigt, dass hier keine Foreign Keys verwaltet werden

Weitere Optionen:

- `cascade`: Operationen angeben, die alle Child Objects betreffen. Wenn keine angegeben sind, werden Child Objects nicht gelöscht, wenn der Parent gelöscht wird. Kann zu Problemen mit übrigen Foreign Keys führen
- `orphanRemoval`: Automatische Lösung von Child Objects, wenn sie von ihrem Parent getrennt werden.

```kotlin
@OneToMany(
    mappedBy = "role"
)
var users: List<User> = emptyList()
```

`@OneToMany` und `@ManyToOne`

- `rule` Variable in User Klasse wird mit `@ManyToOne` annotated
- `@ManyToOne` erschafft eine bidirektionale Relation
- Auch in User Klasse können Daten über Role abgerufen werden
- `@JoinColumn` auf der besitzenden Seite der Relation
- `referencedColumnName` nicht zwingend nötig. Primary Key kann automatisch gesucht werden

Kotlin

```kotlin
@ManyToOne
@JsonBackReference
@JoinColumn(name = "role_id", referencedColumnName = "id")
var role: Role? = null
```

@OneToOne

- Verbindung zwischen zwei einzelnen Entities
- `@JoinColumn` auf der besitzenden Seite der Relation
- Nicht besitzende Seite besitzt `mappedBy` Attribute in `@OneToOne`

Kotlin

```kotlin
class User (
  @OneToOne
  @JsonBackReference
  @JoinColumn(name = "address_id", referencedColumnName = "id")
  var address: Address? = null
)
```

Kotlin

```kotlin
class Address (
  @OneToOne(mappedBy = "address")
  var user: User? = null
)
```

`@ManyToMany`

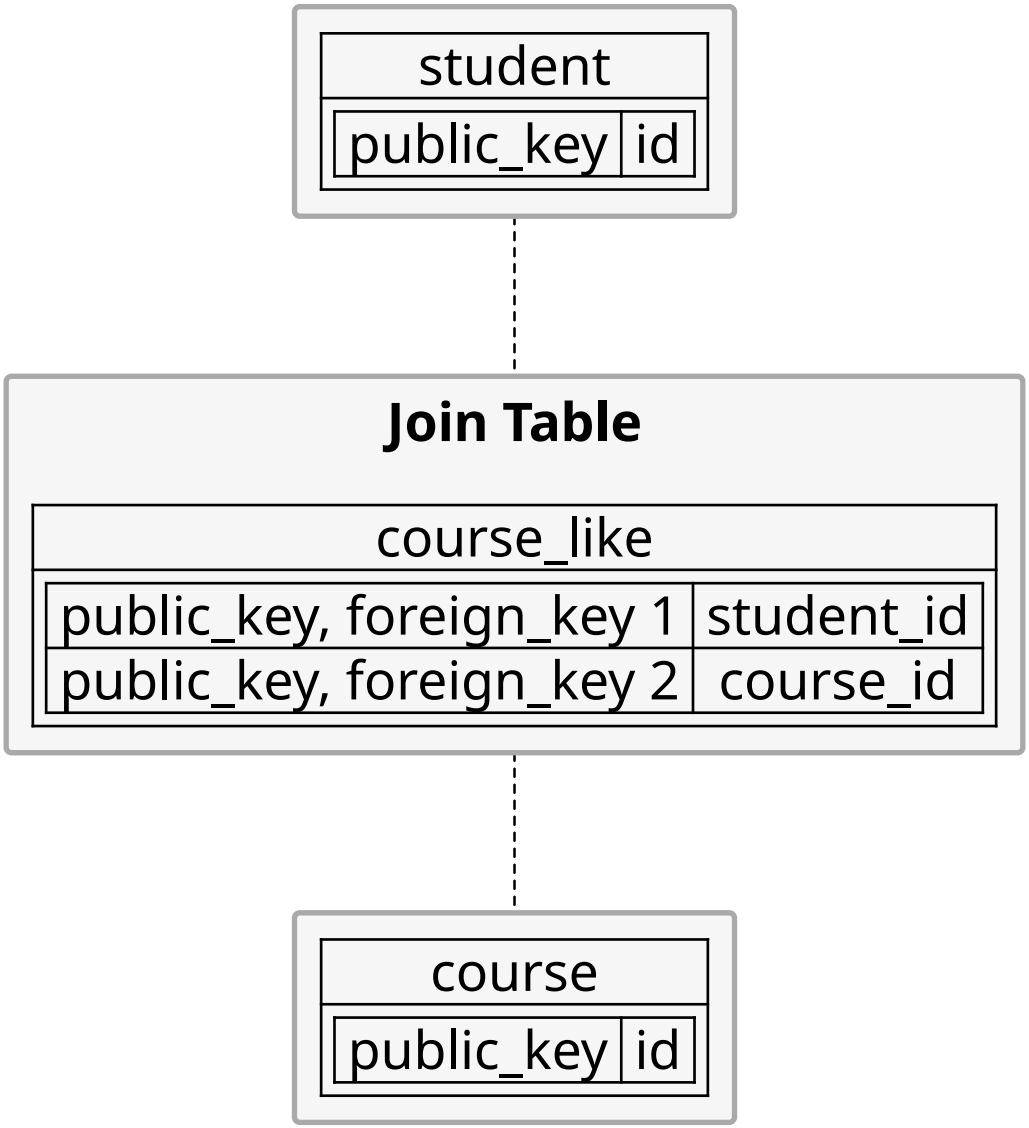


Figure 1: Aufbau einer Many to Many Relation mit einem Join Table

```kotlin
class Student (
  @ManyToMany
  @JoinTable(
    name = "course_like",
    joinColumns = @JoinColumn(name = "student_id"),
    inverseJoinColumns = @JoinColumn(name = "course_id")
  )
  val likedCourses: List<Course>
)
```

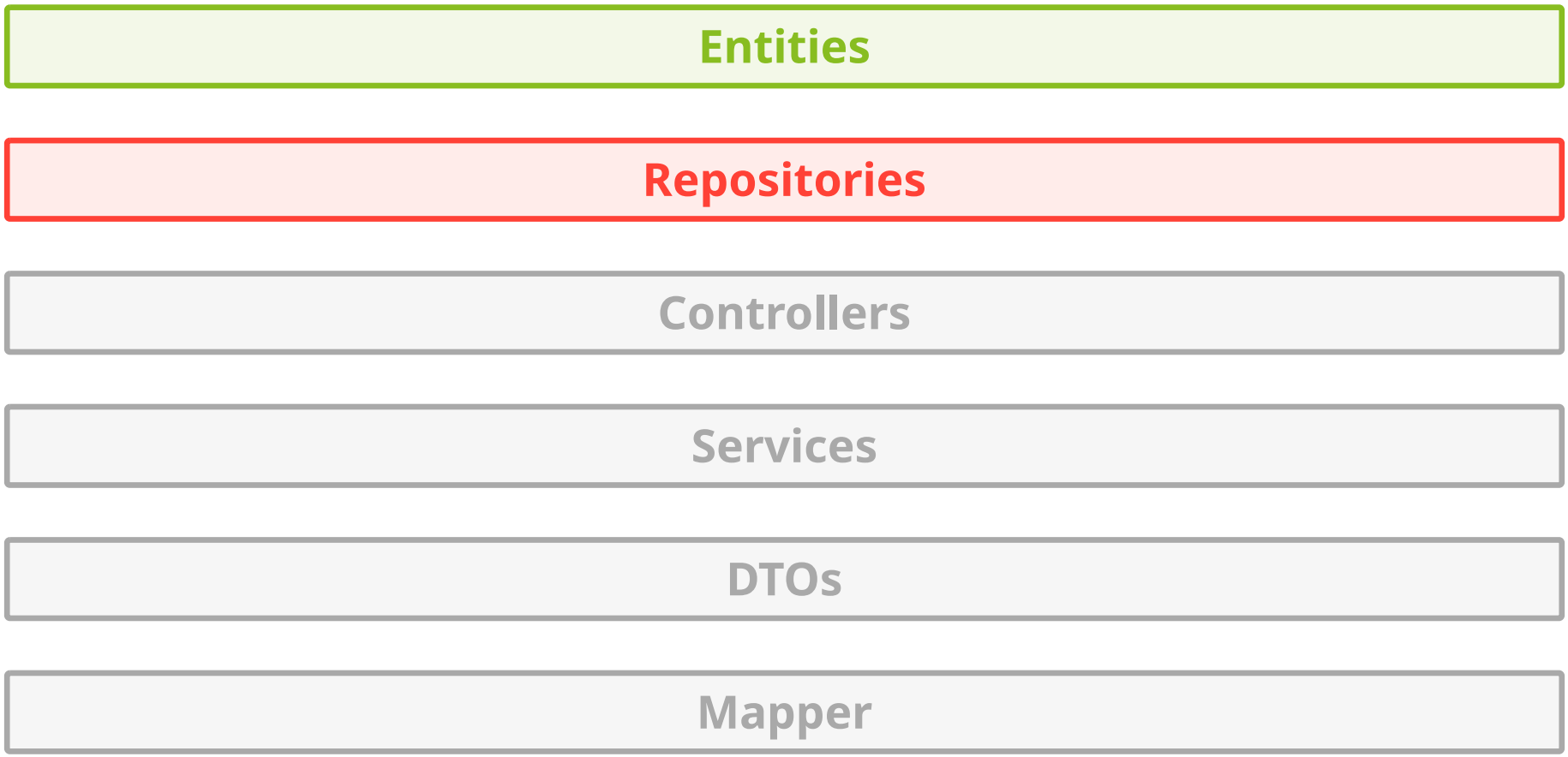
```kotlin
class Course (
  @ManyToMany(mappedBy = "likedCourses")
  val likes: List<Students>
)
```

`@ManyToMany`

- Bei Many to Many Beziehungen muss ein neuer Table angelegt werden
- Neuer Table enthält Foreign Keys aller Entities in der Relation
- Neuer Table ist ein Join Table
- Datenfelder auf beiden Seiten erhalten `@ManyToMany` Annotation
- Besitzende Seite enthält Definition für den Join Table mit `@JoinTable`
- Nicht besitzende Seite erhält `mappedBy` Attribut mit Variablennamen der besitzenden Seite

`@ManyToMany` - `@JoinTable`

- Enthält Name für den Join Table
- `joinColumns` enthält Verbindung zu der besitzenden Seite mit `@JoinColumn` und dem Namen der ID
- `inverseJoinColumns` definiert die nicht besitzende Seite der Relation mit `@JoinColumn` und dem Namen der ID
- `@JoinColumn` und `@JoinTable` sind nicht zwingend notwending
- JPA kann Verbindungen selbst herstellen
- Namenskonventionen müssen dafür allerdings übereinstimmen und Strategie zur Verbidung kann nicht ausgesucht werden


Entities
Repositories
Controllers
Services
DTOs
Mapper

Repository

- Hauptinteraktionspunkt mit der Datenbank
- Wird erstellt mit `@Repository` Annotation in einem **Interface**
- Vererbung bestimmt Art des Repositories
- Unterschiedliche Repositories bringen unterschiedliche eingebaute Funktionen mit
- Repository benötigt Klasse mit ID Typ, die es verwalten soll

Kotlin

```kotlin
@Repository
interface EntityReposirory: JpaRepository<Entity, Long> {

}
```

Operationen und Queries

- Einfach CRUD Operation werden direkt untersützt
- Möglichkeiten, eigene SQL Queries zu verfassen oder generieren zu lassen:
  - Definierung einer neuen Methode im Interface
  - SQL Query bereitstellen über die `@Query` Annotation
  - Nutzung von Querydsl
  - Eigene Queries erstellen mit JPA Named Queries

Automatische Generierung von Queries

- Spring Data analysiert jede Methode in einem Repository
- Versucht aus Methodennamen eine Query zu erstellen
- Untersützter Syntax ist dabei sehr breit (mehr im Seminar)

```kotlin
@Repository
interface UserInterface : JpaRepository<User, Long> {
  fun findUserByName(name: String): MutableList<User>
}
```

Manuelle Queries

- `@Query` Annotation erlaubt Erstellen von selbst geschriebenen Queries

```kotlin
@Query(
  "SELECT u FROM User u WHERE LOWER(u.name) = LOWER(:name)"
)
fun retrieveUserByName(
  @Param("name") name: String
): User
```

Entities

Repositories

Controllers

Services

DTOs

Mapper

Controller

Kotlin

```kotlin
@RestController
@RequestMapping("/path/to/controller")
class Controller(
  private val service: Service
) {
  @GetMapping
  @ReponseStatus(HttpStatus.OK)
  fun getEntities(): List<GetEntityDto> {

  }
}
```

Mappings

- `@RequestMapping` Annotation erlaubt das Mapping von Anfragen auf Methoden in der annotierten Klasse
- Unterschiedliche Parameter für die Zuordnung:
  - URL
  - Http Methode
  - Request Parameter
  - Header und Media Typen

```kotlin
@RequestMapping("/path/to/controller")
```

HTTP Methoden Mappings

- Für HTTP Methoden gibt es Annotationen, die als Abkürzung von `@RequestMapping` dienen:
  - `@GetMapping`
  - `@PostMapping`
  - `@PutMapping`
  - `@DeleteMapping`
- Sie nutzen selbst `@RequestMapping`
- Annotationen existieren, da ein Controller die meisten seinr Methoden auf HTTP Methoden mappen sollte
- `@RequestMapping` würde jede beliebige HTTP Methode mappen

HTTP Methoden Mappings

Nuzten von mehreren `@RequestMapping` Annotationen Hinweis 2

Es kann immer nur eine `@RequestMapping` an einem Element geben. Ein Element ist in diesem Fall eine Klasse, Methode oder ein Interface. Sollten dennoch mehrere Annotationen an einem Element vorhanden sein, wird nur die erste genutzt. Diese Regel gilt auch für Annotationen, die auf `@RequestMapping` basieren.

URI Patterns für Mappings

Beispiele:

- `"/resources/ima?e.png"` - Ein Zeichen wird abgeglichen
- `"/resources/*.png"` - Eine beliebige Anzahl an Zeichen wird abgeglichen
- `"/resources/**"` - Mehrere Pfad Segmente werden abgeglichen
- `"/projects/{project}/versions"` - Pfad Element wird abgeglichen und als Variable ausgelesen
- `"/projects/{project:[a-z]+}/versions"` - Pfad ELement wird abgeglichen mit Regex und als Variable ausgelesen

Variablen in Pfaden

- Variablen können in Pfaden mit `{}` angegeben werden
- Mit `@PathVariable` können diese Variablen ausgelesen werden

Kotlin

```kotlin
@GetMapping("/owners/{ownerId}/pets/{petId}")
fun findPet(
  @PathVariable ownerId: Long,
  @PathVariable petId: Long
): Pet {

}
```

Variablen in Pfaden

- Pfad Variablen können bereits auf Klassenebene deklariert werden

Kotlin

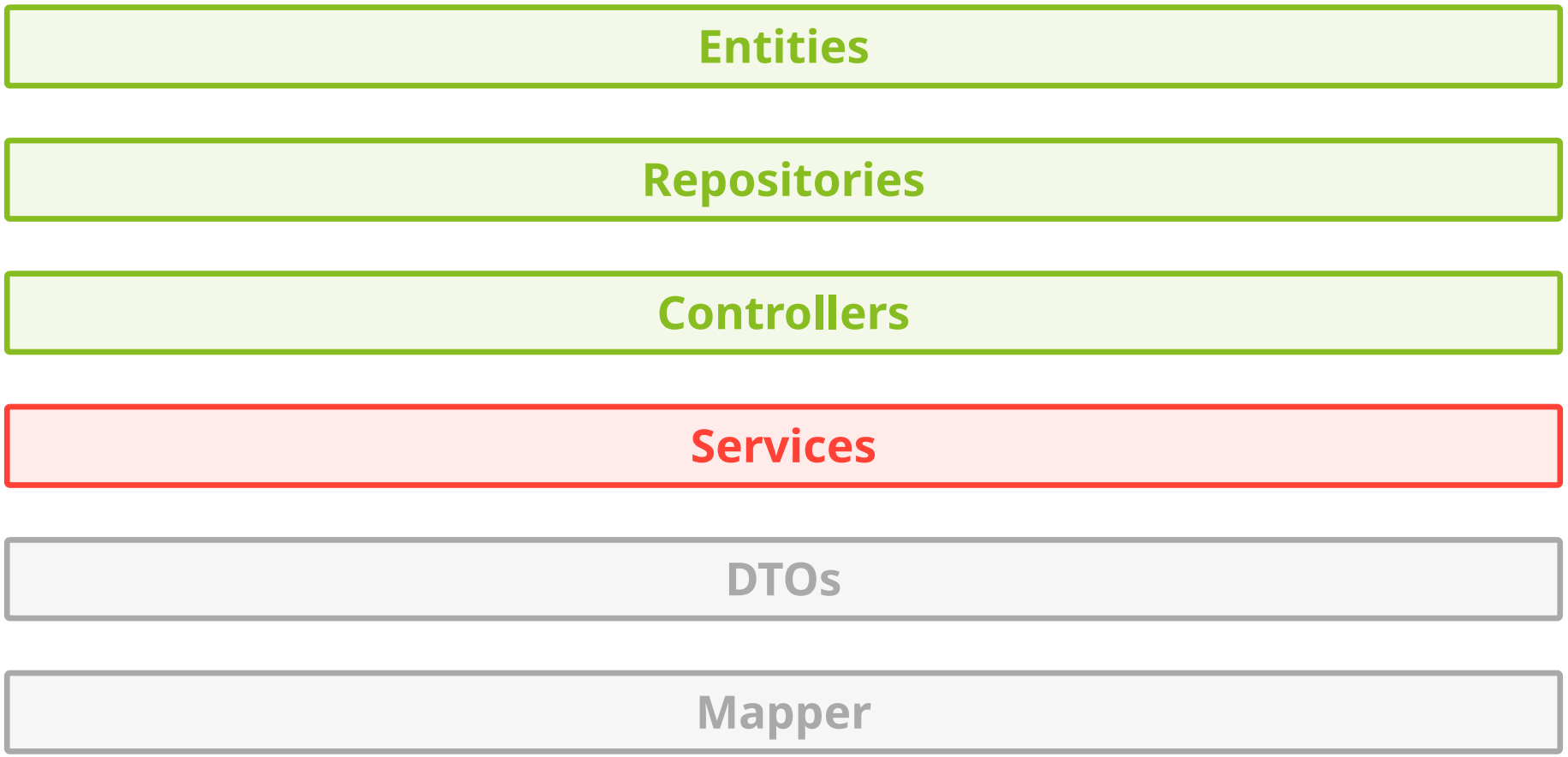
```kotlin
@RestController
@RequestMapping("/owners/{ownerId}")
class OwnerController {
  @GetMapping("/pets/{petId}")
  fun findPet(
    @PathVariable ownerId: Long,
    @PathVariable petId: Long
  ): Pet {

  }
}
```

Variablen in Pfaden

- Konvertierung zu den gewollten Typen findet automatisch statt
- Einfache Typen wie `int`, `long` oder `Date` werden untersützt
- Eigene Konvertierungen können für komplexere Typen implementiert werden
- Wenn der Name im Pfad nicht mit dem Variablennamen übereinstimmt, kann ein Name angegeben werden

```kotlin
1    @PathVariable("customName")
```


Entities
Repositories
Controllers
Services
DTOs
Mapper

Service

- Wird erstellt mit der `@Service` Annotation an einer Klasse
- Enthält Business Logik der Anwedung
- Typischerweise enthält der Service eine Funktion pro Controller Mapping
- Der Controller sollte nur diese Funktion aufrufen

Kotlin

```kotlin
@Service
class EntityService (
  private val entityReposirory: EntityReposirory
) {
  // service functions
}
```

Service Funktionen

Kotlin

```kotlin
@RestController
@RequestMapping("/path/to/controller")
class Controller(
  private val entityService: EntityService
) {
  @GetMapping
  @ReponseStatus(HttpStatus.OK)
  fun getEntities(
    @RequestParam(required = false) id: Long?,
    @RequestParam(required = false) name: String?
  ): List<GetEntityDto> {
    return entityService.getEntities(id, name);
  }
}
```

Service Funktionen

Kotlin

```kotlin
@Service
class EntityService (
  private val entityRepository: EntityRepository
) {
  fun getEntities(id: Long?, name: String?): List<Entity> {
    var entities = entityReposirory
      .getEntitiesByFields(id, name)
      .map{
        toGetEntityDto(it);
      };
    return entities;
  }
}
```

Transactional Methoden

Kotlin

```kotlin
@Transactional
fun deleteUser(id: Long) {
  val toDeleteUser = getUserById(id)
  userRepository.delete(toDeleteUser)
}
```

Transactional Methoden

```kotlin
@Entity(name = "user")
@Table(name = "user")
class User(
  @Id
  @GeneratedValue(strategy = GenerationType.AUTO)
  val id: Long? = null,
  @OneToMany(
      mappedBy = "user",
      cascade = [CascadeType.REMOVE]
  )
  val roles: List<Role> = emptyList()
)
```

Transactional Methoden

- Manche Funktionen im Service werden mit `@Transactional` annotiert
- Annotation verlangt Nutzung des Spring Transaction Managements
- Sonst würde würden andere Persitence Teile der Anwendung die Transaktion durchführen
- `@Transactional` stellt sicher, dass alle Änderungen an der Datenbank erfolgreich waren, bevor die Änderungen eingefügt werden

Wann sollte `@Transactional` genutzt werden

- **Datenbank Operation mit mehreren Schritten**
  - Mehrere Schritte müssen Erfolgreich sein
  - Beispiel: Transaktion in einem Banksystem. Entferne Geld beim Sender und füge es beim Empfänger hinzu
- **Wenn die Operation kaskadierende Updates durchführt**
  - Operation beeinflusst Daten in anderen Tabellen
  - Beispiel: Löschen eines Users, der Konten besitzt, die auch gelöscht werden müssten
- **Wenn volatile Daten benutzt werden**
  - Fehler sind wahrscheinlich oder werden erwartet
- **Wenn bei Daten parallelität bzw. concurrency erwartet wird**
  - Mehrere Nutzer arbeiten parallel an Datenbank
  - `@Transactional` isoliert Operationen voneinander
  - Änderungen werden erst nach Erfolg sichtbar

Wann wird `@Transactional` nicht benötigt

- Updates die nur einen Schritt enthalten und nicht kaskadieren
- Wenn strikte konsistenz nicht benötigt wird
- Methoden, die keine Daten modifizieren

Entities

Repositories

Controllers

Services

DTOs

Mapper

DTO

- Bündelung mehrerer Datenfelder
- Inhalt Richtet sich nach Einsatzzweck
- Beispiel: DTO, welches zu einer GET Request gehört und die Daten enthält, die zurückgegeben werden sollen
- Verweise auf andere Tabellen können als ID gespeichert werden oder direkt mit allen Daten eingetragen werden

Beispiele

```kotlin
class Entity(
  val id: Long?,
  val name: String,
  val age: Integer
)
```

```kotlin
data class GetEntityDto (
  val id: Long,
  val name: String,
  val age: Integer
)
```

Beispiele

```kotlin
class Entity(
  val id: Long?,
  val name: String,
  val age: Integer
)
```

```kotlin
data class PostEntityDto (
  val name: String,
  val age: Integer
)
```

Entities

Repositories

Controllers

Services

DTOs

Mapper

Mapper

- Umwandlung von Entities in DTOs oder umgekehrt
- Meist trivial, da nur Kopieren von Werten stattfindet

```kotlin
class Entity(
  val id: Long?,
  val name: String,
  val age: Integer
)
```

```kotlin
fun convertEntityToGetEntityDto(entity: Entity): GetEntityDto {
  return GetEntityDto(
    id = entity.id!!,
    name = entity.name,
    age = entity.age
  )
}
```

Mensa im Osten

Studentenwerk Dresden