295 - Backend

Auftrag Fallstudie 3 – Item mit Service und DTO

|  |  |
| --- | --- |
| Richtzeit  3 Stunden | Sozialform  Einzelarbeit |

Im letzten Auftrag sind wir über das Problem der zyklischen/zirkulären Referenzen gestolpert. In diesem Auftrag werden wir nun eine Lösung implementieren, die dieses Problem behebt.

## Service

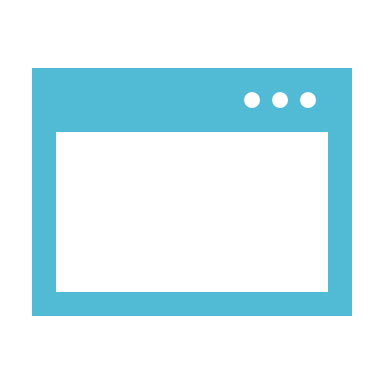
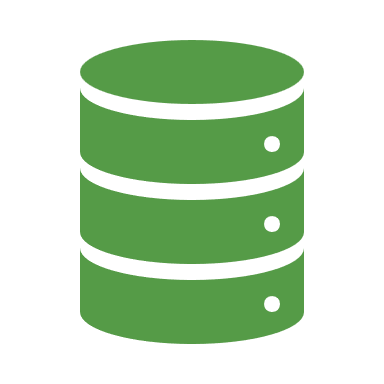
Im letzten Auftrag haben wir die Repositories direkt im Controller verwendet und haben mit dem Controller Entitäten direkt serialisiert und als Antwort verschickt. Der Service wird als Zwischenschicht dienen und beinhaltet möglichst alle Logik, die unsere Applikation benötigt. Deine aktuelle Lösung sieht ungefähr wie folgt aus:

Controller

Repository

findById  
findAll  
deleteById  
usw.

save  
findById  
deleteById  
usw.



Entity

Entity

Zwischen dem Client auf der linken Seite und der Datenbank auf der rechten Seite werden direkt die Entities ausgetauscht. Da die Entities Beziehungen haben und direkt an den Controller weitergegeben werden erhalten wir genau das Problem der Referenzen, die endlos aufgelöst werden. Um dieses Problem zu lösen, brauchen wir Logik und diese gehört in einen Service. So können wir selbst steuern, wie diese Beziehungen aufgelöst werden und welche Daten wir an den Controller und somit an den Client weitergeben. Da sich die Daten oder Typen nun unterscheiden werden braucht es noch eine weitere Komponente, nämlich Übertragungsobjekte, sogenannte Data Transfer Objects (DTO).

## DTO

Unter dem Begriff Data Transfer Object, kurz DTO, versteht man ein Objekt zum Übertragen von Daten. Es kann davon mehrere geben. Beispielsweise eins für den Request und eins für die Response. Vielleicht hast du im Package person bereits solche gesehen. Auch im Package security werden DTOs verwendet. Gerade bei der Person macht dies Sinn, da wir beispielsweise das Passwort nicht in der Response haben wollen, hingegen im Request zum Erstellen einer Person brauchen wir zwingend ein Passwort.

Wir können mit DTOs also direkt steuern, welche Daten wir an den Client weitergeben und welche Daten wir nicht nach aussen sichtbar machen wollen.

Diagram

Description automatically generated

## Architektur

Mit dieser Lösung erhalten wir am Ende eine Struktur, die wir in Form einer Schichtenarchitektur beschreiben können. Die Architektur wird dann später im üK 223 relevant werden. Hier also nur als kleiner Teaser.

Persistenz

Logik

Präsentation

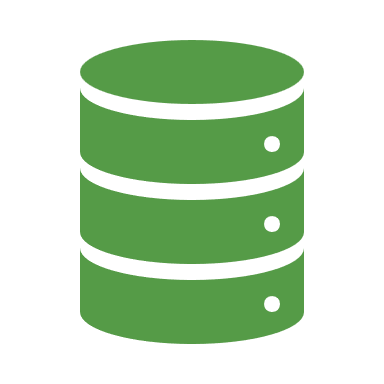
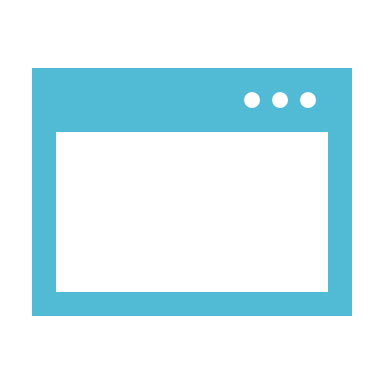
Repository

Service

Controller

save  
findById  
deleteById  
usw.

findById  
findAll  
deleteById  
usw.



Entity

DTO

findById  
findAll  
deleteById  
usw.

## Auftrag

Damit haben wir die Theorie behandelt und können mit der Umsetzung weiterfahren.

### Timeline Description automatically generated with low confidenceDTO und Mapper

Erstelle im Package ch.bbcag.backend.todolist.item eine Klasse **ItemRequestDTO** und eine Klasse **ItemResponseDTO**. Orientiere dich dabei an den beiden Diagrammen. Dazu gehören selbstverständlich auch die Getter, Setter, equals und hashCode.

Da du nun Entities und DTOs hast musst du diese umwandeln können. Dafür brauchen wir einen **ItemMapper**, der aus einem Item ein ItemResponseDTO und aus einem ItemRequestDTO ein Item macht. Ein Item wird jedoch **nie** in ein ItemRequestDTO umgewandelt und ein ItemResponseDTO **nie** in ein Item. Es braucht also nur zwei Methoden, **fromRequestDTO** und **toResponseDTO**.

Icon

Description automatically generated

**Hinweis**

Der gefüllte Pfeil von der Response zum Request steht für Vererbung. Die Antwort hat zusätzlich zu den Attributen des Requests die ID und den Erstellzeitpunkt darin.

Die Signatur diese Methoden soll wie folgt aussehen:

public static ItemResponseDTO toResponseDTO(Item item)

und

public static Item fromRequestDTO(ItemRequestDTO itemRequestDTO)

Innerhalb dieser Methoden werden alle Werte, die vorhanden sind, übertragen. Z.B. alle Werte vom Item, die im ItemResponseDTO vorhanden sind werden in eine neue Instanz des DTOs geschrieben.

Icon

Description automatically generated

**Achtung!**

Beim Umwandeln vom DTO ins Objekt kann es sein, dass eine ID null ist. In dem Fall wollen wir offensichtlich nicht ein Objekt mit der id = null bekommen, sondern direkt null ins Attribut schreiben.

Der Achtung-Block bedeutet in diesem Fall, dass du beim Umwandeln vom ItemRequestDTO zum Item prüfen musst, ob personId nicht null ist. Nur wenn personId nicht null ist erstellst du eine neues Person-Objekt und fügst diese dem Item hinzu. Sonst soll person null sein.

Der Mapper wird später im Service verwendet werden, da dort die Schnittstelle zwischen dem Controller und dem Repository ist. Zwischen dem Service und dem Controller werden DTOs übergeben. Zwischen dem Service und dem Repository werden Entities übergeben.

### Service

Sobald die DTOs und der ItemMapper erstellt sind kümmern wir uns um den Service. Der Service wird vom Controller jeweils ein ItemRequestDTO erhalten, dieses in ein Item umwandeln und dem Repository übergeben. Die Items, die vom Repository zurückkommen werden in ein ItemResponseDTO umgewandelt und dem Controller zurückgegeben.

Graphical user interface, text, application

Description automatically generatedErstelle dafür im Package ch.bbcag.backend.todolist.item die Klasse ItemService. Im ItemService muss für jede Methode im Controller eine Passende Methode erstellt werden. Wenn der Controller ein findById hat, muss auch der Service ein findById haben. Sinnvollerweise benennst du diese Methoden auch gerade gleich. In der Methode des Services wird dann die entsprechende Methode des Repositories aufgerufen. Halte dich an das nebenstehende Diagramm.

Im Moment kann ein Item nur erstellt, gelesen, gesucht und gelöscht werden. Nun wollen wir die Funktionalität unserer Applikation noch um eine Update-Funktion erweitern.

**Hinweis**

Beim Update und Insert wird die gleiche Methode auf dem Repository verwendet. Diese erkennt automatisch, ob ein Item zum Updaten oder neu Erstellen ist (es wird geprüft, ob item.getId() eine gültige Id zurückgibt).

Icon

Description automatically generated

#### Update und Merge

Bei der Update-Funktion wollen wir einzelne Werte eines Items verändern können. Beispielsweise den Namen oder die Beschreibung usw. Dafür werden wir eine mergeItem-Methode benötigen. Dort schreiben wir die neuen Werte ins Item aus der Datenbank. In der folgenden Anleitung werden wir die Update-Funktionalität aufbauen und umsetzten.

Als erstes definieren wir die Signatur der update-Methode. Diese soll ein ItemRequestDTO und einen Integer entgegennehmen. Als Rückgabetyp werden wir ItemResponseDTO verwenden.

public ItemResponseDTO update(ItemRequestDTO itemRequestDTO, Integer itemId)

Mit dem Parameter itemId holen wir uns das Item mit dieser Id aus der Datenbank. Dafür verwenden wir die Methode findById().orElseThrow() des Repositories.

Item existingItem = itemRepository.findById(itemId).orElseThrow(EntityNotFoundException::new);

Nun haben wir in der Variable existingItem das Item aus der Datenbank. In der Variable itemRequestDTO befindet sich das Item mit den Werten, die in der Datenbank verändert werden sollen. Dieses müssen wir noch vom DTO in ein Item umwandeln.

Item changingItem = ItemMapper.*fromRequestDTO*(itemRequestDTO);

Jetzt haben wir in der Variable existingItem das Item aus der Datenbank und in der Variable changingItem das Item, welches wir vom Controller/Client erhalten haben.  
Als nächsten Schritt wollen wir die neuen Werte aus changingItem ins existingItem mergen.

Dafür schreiben wir eine mergeItem-Methode, die die beiden Items entgegennimmt.

**Hinweis**

Da bei Java die Referenz zum Objekt im Speicher übergeben wird muss das veränderte Item nicht zurückgegeben werden. Die mergeItem-Methode kann also void als Rückgabetyp haben und die Änderungen werden trotzdem übernommen.

Icon

Description automatically generated

Folgende Grafik zeigt, was innerhalb der mergeItem-Methode genau passieren soll.

changingItem

existingItem

name: “ItemName”  
description: “Foo”  
person: …  
doneAt: null

name: “NewName”  
description: null  
person: null  
doneAt: “2023-01-01”

merge

name: “NewName”  
description: null  
person: null  
doneAt: “2023-01-01”

changingItem

existingItem

name: “NewName”  
description: “Foo”  
person: …  
doneAt: “2023-01-01”

Wir nehmen also alle Felder, die nicht null sind und schreiben sie in das existierende Item.

Die Signatur der Methode mergeItem soll wie folgt aussehen:

private void mergeItems(Item existingItem, Item changingItem)

So können wir in der update-Methode die mergeItem-Methode aufrufen und die beiden Items existingItem und changingItem mitgeben. Dann werden auf der Instanz von existingItem die Werte angepasst und wir können existingItem mit der save-Methode des Repositories wieder in die Datenbank speichern.

mergeItems(existingItem, changingItem);

Als letztes müssen wir noch ein ItemResponseDTO zurückgeben. In diesem Beispiel wird der Rückgabewert der save-Methode direkt in die toResponseDTO-Methode des ItemMappers gegeben und dessen Rückgabewert, das ItemResponseDTO, zurückgegeben.

return ItemMapper.*toResponseDTO*(itemRepository.save(existingItem));

### Controller anpassen

Da nun der Service steht und die DTOs umgesetzt wurden muss noch der ItemController angepasst werden. Aktuell nimmt der Controller direkt Items entgegen und gibt auch Items zurück. Da aber der Service DTOs entgegennimmt und auch zurück gibt müssen diese auch im Controller verwendet werden. Ausserdem ruf er direkt das ItemRepository auf. Entsprechend soll der Controller in seinen Methoden neu ein ItemRequestDTO entgegennehmen und nicht mehr direkt auf das Repository zugreifen, sondern auf den Service. Auf den Rückgabetyp kommen wir später noch zurück.

Als erstes muss statt dem Repository neu der Service eingebunden werden. Wir ersetzten also den aktuellen Konstruktor und die Instanzvariable itemRepository mit folgendem Codeausschnitt.

private final ItemService itemService;  
  
@Autowired  
public ItemController(ItemService itemService) {  
 this.itemService = itemService;  
}

Als nächstes passen wir die Signaturen der Methoden an, die ein Item als Parameter haben. Beispielsweise die Methode insert wird neu folgende Signatur haben.

public ResponseEntity<?> insert(@RequestBody ItemRequestDTO newItem)

Innerhalb der Methode rufen wir dann die entsprechende Servicemethode auf. In diesem Beispiel wäre das die Methode itemService.insert().

Als neuer Rückgabetyp verwenden wir ResponseEntity<?>. Diese Klasse ist speziell für http-Responses. Wir können damit sehr Einfach http-Antworten erzeugen, Statuscodes setzen und Daten in den Body oder den Header der Antwort einfügen.

return ResponseEntity.*status*(HttpStatus.CREATED).body(itemService.insert(newItem));

Mit obenstehendem Code erstellen wir eine neue ResponseEntity, setzten den Statuscode 201 (CREATED) und packen das neue Item, dass vom Service in Form eines ItemResponseDTOs zurückgegeben wird, in den Body der Antwort.

Auf der ResponseEntity hat es weitere statische Methoden: Mit ResponseEntity.ok(itemService.findAll()) wird der Statuscode 200 (OK) gesetzt und die Rückgabe von findAll() in den Body geschrieben. Soll eine Antwort ohne Body zurückgegeben werden kann dies semantisch korrekt mit ResponseEntity.noContent().build() für den Statuscode 204 oder mit ReponseEntity.status(HttpStatus).build() gemacht werden.

Setz update-Methode auch im Controller um. Verwende dafür eine PATCH-Mapping, welches eine Id als Pfadvariable und ein ItemRequestDTO im Body entgegennimmt.