

Web Engineering 3

Vorlesung 4

Hochschule Zittau/Görlitz
Christopher-Manuel Hilgner

Agenda

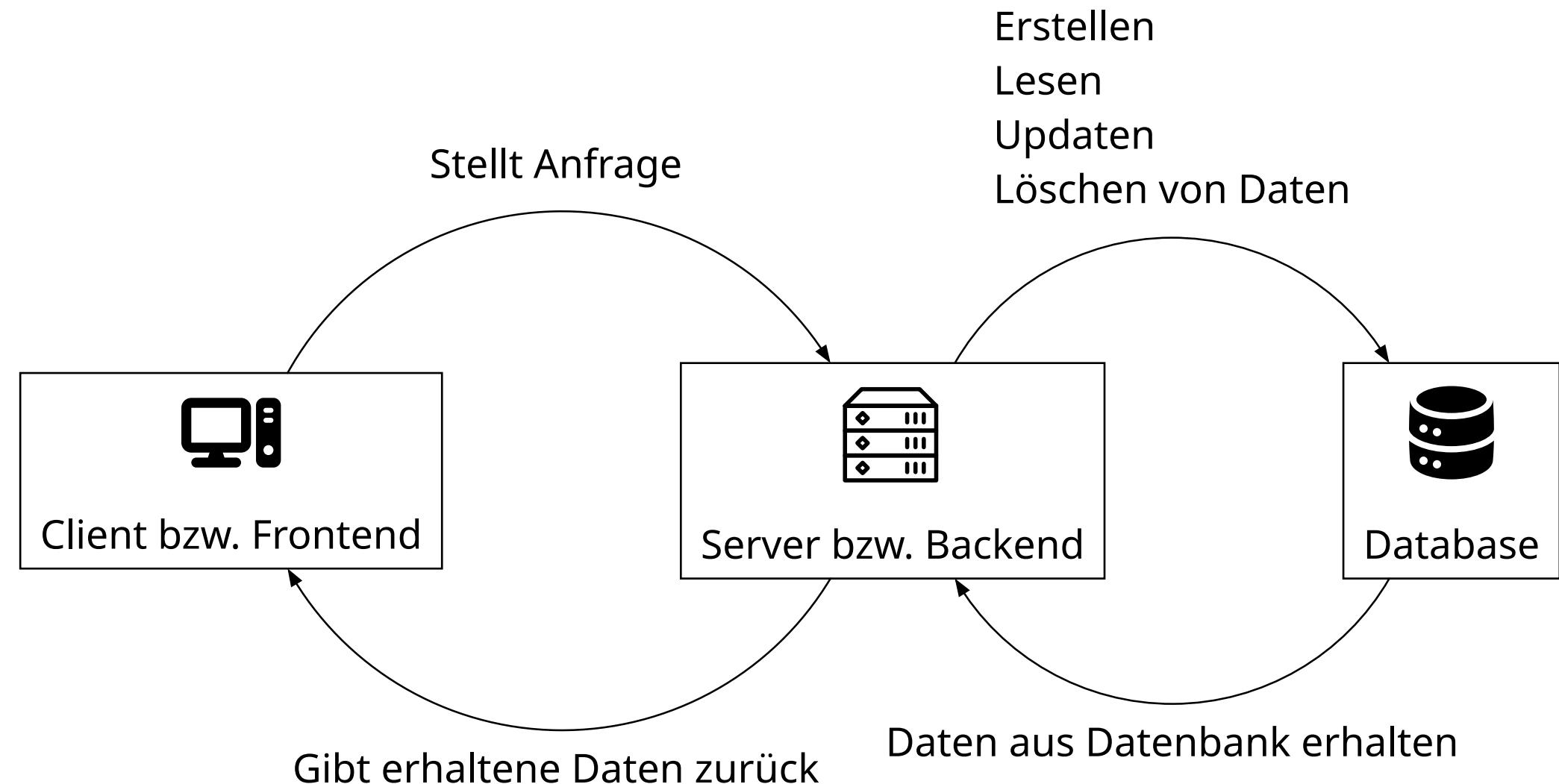
Wiederholung von Spring

- Kontext
- Komponenten
- Ablauf innerhalb der Spring Anwendung

Kontext einer Spring Anwendung in Fullstack

- Spring existiert im Backend
- Frontend stellt Anfragen an das Spring Backend
- Spring verarbeitet Anfragen vom Frontend
- Gibt bei Bedarf Daten wieder an das Frontend zurück

Aufbau der Fullstack Anwendung



Beispiel: Finanzapp

- Eine Finanzapp erlaubt das Aufteilen von Geld auf Budgets
- Der Nutzer möchte Geld auf ein Budget verschieben
- In der App wird das Zielbudget und der Wert festgelegt
- Nach Bestätigung der Eingaben wird eine Anfrage an das Backend geschickt
- Die Anfrage könnte folgende Informationen enthalten:
 - ID vom Konto
 - ID vom Budget
 - Wert des Übertrags

Beispiel: Finanzapp

- Im Backend werden mit den IDs Konto und Budget gesucht
- Es wird überprüft, ob das Konto genug Mittel hat um den gewünschten Betrag zu verschieben
- Gutschreiben des Wertes, wenn das Konto genug Geld hat
- Abzug des Wertes vom Ursprungskonto
- Neue Werte von Konto und Budget werden an das Frontend geschickt
- Neue Werte werden in der UI gerendert

Entities

- Eine Klasse, die mit @Entity annotiert ist
- Beschreibung eines Tabelleneintrags in einer Datenbank
- Jedes Feld in der Tabelle wird als Variable der Entity Klasse erstellt
- Felder können genauer durch @Column definiert werden
- Jede Entity benötigt einen Primary Key, der durch @Id markiert wird

Erstellung von Entities in Kotlin oder Java

```
1 @Entity(name =  
         "todoitem")  
2 @Table(name = "todoitem")  
3 class TodoItem (  
4     @Id  
5         @GeneratedValue(strategy =  
6             GenerationType.AUTO)  
6     val id: Long? = null,  
7     @Column(name = "name",  
8             nullable = false)  
8     val name: String = "",  
9 )
```



```
1 @Entity(name =  
         "todoitem")  
2 @Table(name = "todoitem")  
3 public class TodoItem {  
4     @Id  
5         @GeneratedValue(strategy =  
6             GenerationType.AUTO)  
6     private Long id;  
7     @Column(name = "name",  
8             nullable = false)  
8     private String name;  
9 }
```



Entities

- Wenn aus der Datenbank mehrere Einträge aus einer Tabelle ausgelesen werden, wird eine Liste an passenden Entity Instanzen zurückgegeben
- Jede Instanz enthält die Daten ihres jeweils passenden Tabellen Eintrags

Beispiel: Abfrage von Todo Items

- Es wird nach allen Todos vor einem bestimmten Datum gesucht
- Jedes Todo Item, dessen Datum kleiner als das festgelegte Datum ist, wird als Entity Instanz als Teil einer Liste zurückgegeben

Repository

- Schnittstelle zur Datenbank
- Erlaubt CRUD Operationen auf der Datenbank
- Daten in der Datenbank werden auf Klassen gemapped (ORM)
- Jedes Repository kann sich nur um eine Entity kümmern
- Repository braucht Info über Typ der Entity und Typen der ID

```
1 @Repository
2 interface TodoItemRepository : JpaReposiroty<TodoItem, Long> {
3
4 }
```

Kotlin

```
1 @Repository
2 public interface TodoItemRepository extends
3 JpaReposiroty<TodoItem, Long> {
4 }
```

Java

Ablauf einer POST Request in Spring

POST Request vom Frontend

Request an das Backend

```
1 POST http://localhost:8080/todos HTTP/1.1
2 accept: /*
3 Content-Type: application/json
4
5 {
6     "name": "todoItemName",
7     "description": "newDescription",
8     "done": true,
9     "created": "2025-10-23T15:06:08.738Z",
10    "shouldBeDoneBy": "2025-11-04T08:06:06.297Z",
11 }
```

Request Line

} (1)

} (2)

JSON Body

} (3)

Request an das Backend

```
1  POST http://localhost:8080/todos HTTP/1.1
2  accept: /*
3  Content-Type: application/json
4
5  {
6      "name": "todoItemName",
7      "description": "newDescription",
8      "done": true,
9      "created": "2025-10-23T15:06:08.738Z",
10     "shouldBeDoneBy": "2025-11-04T08:06:06.297Z",
11 }
```

Headers Request Line

} (1)

} (2)

JSON Body

} (3)

Request an das Backend

```
1 POST http://localhost:8080/todos HTTP/1.1
```

} (1)

```
2 accept: */*
```

```
3 Content-Type: application/json
```

} (2)

```
4
```

```
5 {
```

```
6     "name": "todoItemName",
```

```
7     "description": "newDescription",
```

```
8     "done": true,
```

```
9     "created": "2025-10-23T15:06:08.738Z",
```

```
10    "shouldBeDoneBy": "2025-11-04T08:06:06.297Z",
```

```
11 }
```

} (3)

Headers Request Line

JSON Body

Request an das Backend

```
1 POST http://localhost:8080/todos HTTP/1.1
2 accept: /*
3 Content-Type: application/json
4
5 {
6     "name": "todoItemName",
7     "description": "newDescription",
8     "done": true,
9     "created": "2025-10-23T15:06:08.738Z",
10    "shouldBeDoneBy": "2025-11-04T08:06:06.297Z",
11 }
```

Request Line

JSON Body

} (1)

} (2)

} (3)

Content der Anfrage

```
1  {
2      "name": "todoItemName",
3      "description": "newDescription",
4      "done": true,
5      "created": "2025-10-23T15:06:08.738Z",
6      "shouldBeDoneBy": "2025-11-04T08:06:06.297Z",
7  }
```

JSON

Annahme der Request im Backend

Finden des Passenden Controllers

- Die Anfrage muss auf einen Controller und danach auf eine Methode gemapped werden
- Entscheidung läuft wie folgt ab:
- Jede Klasse mit `@RestController` wird betrachtet
- URL wird mit den `@RequestMappings` der Controller abgeglichen

```
1 http://localhost:8080 /todos
```

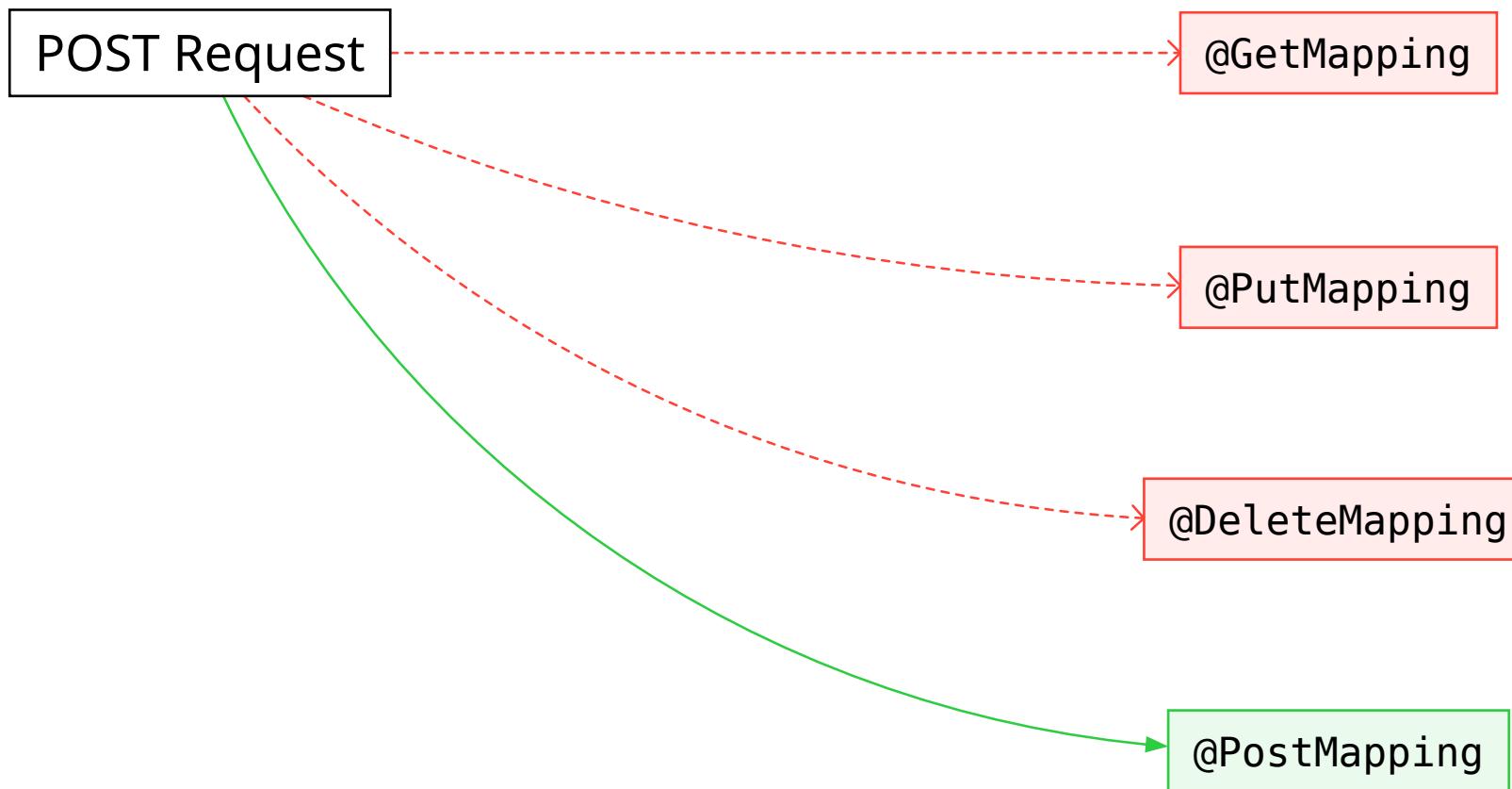
- Der Controller mit dem passenden Request Mapping wird für die Anfrage genutzt

```
1 @RestController
2 @RequestMapping("/todos")
3 class TodoItemController {
4     // Methoden
5 }
```

Kotlin

Finden der passenden Methode

- Im Controller muss die passende Methode gefunden werden
- Im Beispiel kommt hinter /todo kein Pfad mehr
- Es wird also die HTTP Methode abgeglichen
- Die Methode mit dem passenden Mapping wird genommen



Finden der passenden Methode

```
1 @PostMapping  
2 @ResponseStatus(HttpStatus.CREATED)  
3 fun createTodoItem(  
4     @RequestBody createTodoItemDto: CreateTodoItemDto  
5 ): GetTodoItemDto {  
6     return todoItemService.createTodoItem(createTodoItemDto)  
7 }
```



Controller Methode DTO

- Die Methode nimmt ein DTO entgegen
- Das DTO ist eine Repräsentation des JSON Bodys in Form eines Objekts in Kotlin/Java
- Jede Variable im DTO enthält dabei ein Datenfeld aus dem JSON Body

```
1  {  
2      "name": "todoItemName",  
3      "description":  
4      "newDescription",  
5      "done": true,  
6      "created":  
7      "2025-10-23T15:06:08.738Z",  
8      "shouldBeDoneBy":  
9      "2025-11-04T08:06:06.297Z",  
10     }  
11 }
```

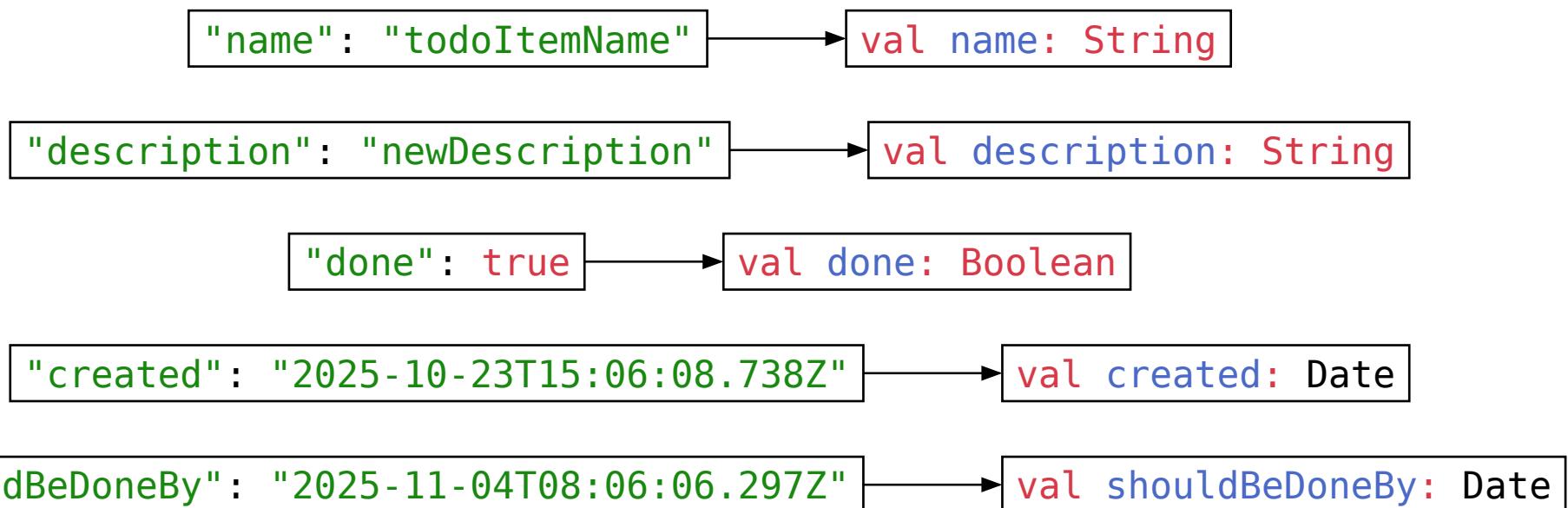
JSON

```
1  data class  
2      CreateTodoItemDto(  
3          val name: String,  
4          val description: String,  
5          val done: Boolean,  
6          val created: Date,  
7          val shouldBeDoneBy: Date  
8      )  
9  
```

Kotlin

Controller Methode DTO

- Das JSON Objekt muss in das DTO überführt werden
- Zu jeder Variable im DTO wird der passende Wert aus dem JSON Body eingetragen
- Folgende Spezifikation ist dabei wichtig: Die Keys in dem JSON Objekt müssen den gleichen Namen haben wie die Variablen im DTO



Controller Methode DTO

Praktische Vorstellung: Es wird ein Konstruktor aufgerufen, der einen Wert pro Feld im JSON objekt enthält und diesen Wert auf die passende Variable anwendet.

```
1 CreateTodoItemDto(  
2     name = "todoItemname",  
3     description = "newDescription",  
4     done = true,  
5     created = "2025-10-23T15:06:08.738Z",  
6     shouldBeDoneBy = "2025-11-04T08:06:06.297Z"  
7 )
```



Service

- Der Controller ruft eine Funktion im Service auf
- Service enthält Logik der Anwendung
- Im einfachsten Fall: Funktionen für die CRUD Operationen bzw. für die HTTP Methoden
- Könnte auch direkt im Controller stehen aber Separierung ist besser
- Methode soll hier ein neues Todo Item erstellen

Folgende Schritte:

1. Umwandlung des DTOs zu einer Entity (Mapper)
2. Speichern der Entity im Repository
3. Umwandeln der neuen Entity in ein Get DTO (Mapper)
4. Rückgabe des Get DTOs an den Controller und zum Frontend

Mapper für CreateTodoItemDto



- Mapper überführt alle Felder des DTOs in eine Entity

```
1  fun fromCreateTodoItemDto(Kotlin
2      createTodoItemDto: CreateTodoItemDto
3  ): TodoItem {
4      return TodoItem(
5          name = createTodoItemDto.name,
6          description = createTodoItemDto.description,
7          done = createTodoItemDto.done,
8          created = createTodoItemDto.created,
9          shouldBeDoneBy = createTodoItemDto.shouldBeDoneBy
10     )
11 }
```

Aufruf des Mappers im Service

- Mapper wird im Service aufgerufen um Instanz der neuen Entity zu erhalten

```
1 val newTodoItem =  
    fromCreateTodoItemDto(createTodoItemDto)
```

Kotlin

Speichern der Entity im Repository

- save Funktion des Repositories erlaubt Speichern einer neuen Entity oder Updated einer vorhandenen

Welche Funktion wird genutzt?:

- Wenn die ID der Entity null ist, wird ein neuer Eintrag in der Datenbank erstellt
- Wenn die ID der Entity einen Wert hat, wird die vorhandene Entity, mit dieser ID, geupdated

```
1 val savedTodoItem = todoItemRepository.save(newTodoItem)
```

Kotlin

Rückgabe der neuen Entity

- Neu erstelle Entity soll wieder an das Frontend zurückgegeben werden
- save Methode gibt die neue Entity zurück
- Es muss diese Instanz der Entity genutzt werden, da diese die ID enthält
- Entity muss in ein DTO überführt werden - GetTodoItemDto



- DTO wird an den Controller übergeben
- Controller gibt das DTO als JSON Objekt im Body der Response zurück

```
1 connection: keep-alive
2 content-type: application/json
3 date: Tue, 04 Nov 2025 08:06:30 GMT
4 keep-alive: timeout=60
5 transfer-encoding: chunked
6
7 {
8     "id": 1,
9     "name": "string",
10    "description": "string",
11    "done": true,
12    "created": "2025-11-04T08:06:06.296+00:00",
13    "shouldBeDoneBy": "2025-11-04T08:06:06.297+00:00",
14    "userId": 1
15 }
```

(1) Headers

(2) JSON Body

MENSA IST VORBEI