295 - Backend

Auftrag Fallstudie 5 - Controllertests

|  |  |
| --- | --- |
| Richtzeit  30 Minuten | Sozialform  Einzelarbeit |

## Beschreibung

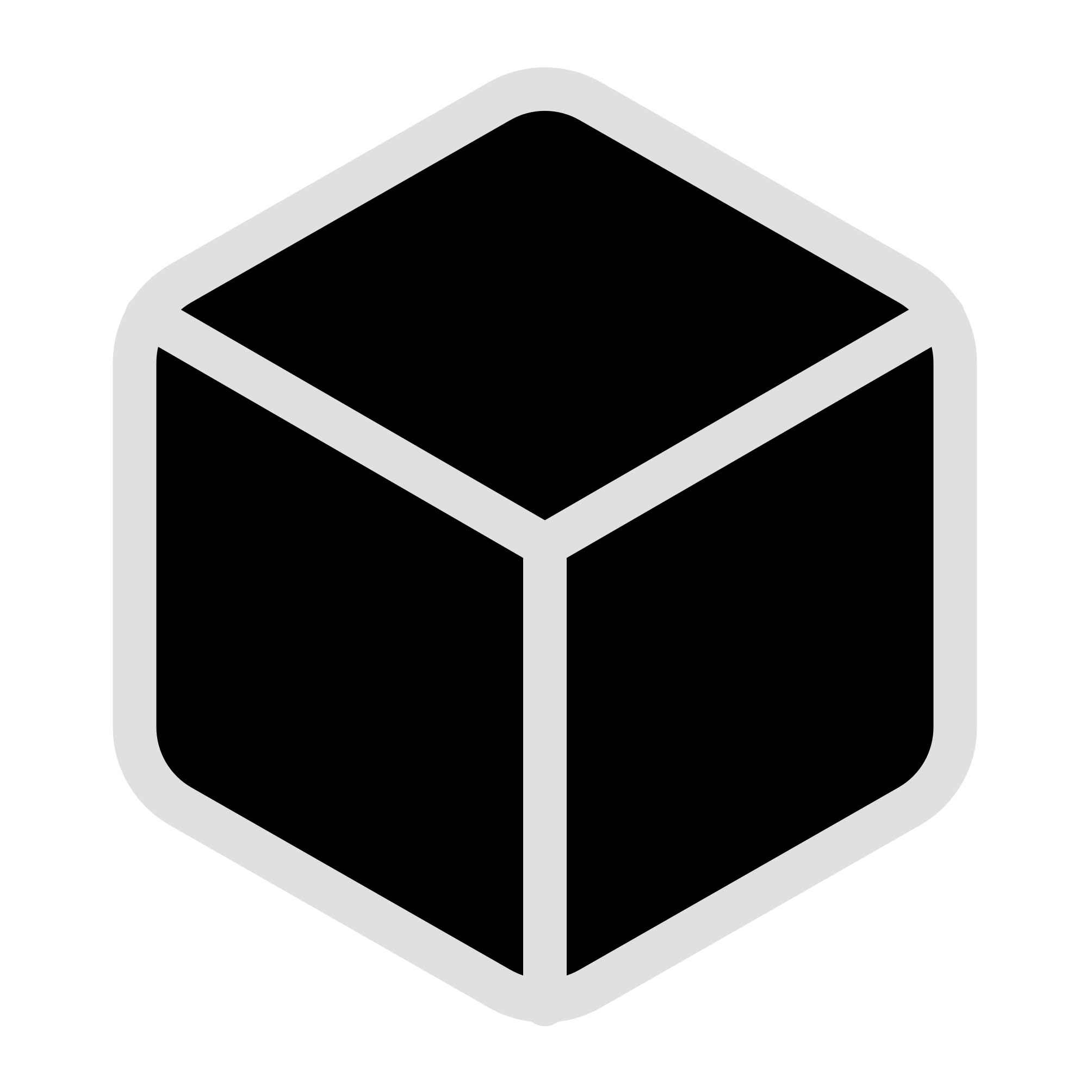
Testing ist ein wichtiger Bestandteil der Softwareentwicklung. Es muss sichergestellt werden, dass der Code auch das macht was er soll und keine Fehler oder Abstürze generiert. Eine Möglichkeit solche Tests zu machen ist mit Unittests. In diesem Auftrag wirst du den ItemController und den TagController testen.

## Auftrag

Die Klasse ItemControllerTest beinhaltet zwei Unittests. Kopiere diese in das Package ch.bbcag.backend.todolist.controllers im Ordner test. Der erste Test checkPost\_whenValidNewItem\_thenIsCreated prüft das Erstellen eines neuen Items.

@Test  
public void checkPost\_whenValidNewItem\_thenIsCreated() throws Exception {  
 Mockito.*when*(itemService.insert(*any*())).thenReturn(TestDataDTOUtil.*getTestItemResponseDTO*());  
  
 mockMvc.perform(*post*(ItemController.*PATH*)  
 .contentType("application/json")  
 .content("{\"name\":\"Item1\", \"personId\":\"1\"}"))  
 .andExpect(*status*().isCreated())  
 .andExpect(*jsonPath*("$.name", *is*("Item1")));  
}

Da von der insert-Methode im Service etwas zurückgegeben wird, muss der Aufruf der Methode gemockt werden. Wir müssen also dem Service sagen, was er dort zurückgeben soll. Die Zeile ist eigentlich selbstsprechend. Wir sagen Mockito wenn (.when()) die Methode itemService.insert() mit irgend einem Objekt (any()) aufgerufen wird, soll sie das ItemResponseDTO aus der TestDataDTOUtil-Klasse zurückgeben (.thenReturn()).

Das mockMvc wird uns von Spring Boot zur Verfügung gestellt und ermöglicht das Testen von Controllern. Der Test ist in einer «Fluent-Syntax» Schreibweise geschrieben. Dies ist für Unittest typisch, lass dich davon nicht abschrecken.

.perform führt einen HTTP Post Request auf der URL /items aus.   
.content beschreibt die übergebenen Request Body Daten.   
.contentType setzt den Contenttype-Header.  
.andExpect beschreibt den zu erwartenden HTTP Response Status Code, hier: 201 Created

Der zweite Unittest prüft, dass bei ungültigen Request Daten der Status Code 400 von der API zurückgegeben wird.

@Test  
public void checkPost\_whenInvalidItem\_thenIsBadRequest() throws Exception {  
 mockMvc.perform(*post*(ItemController.*PATH*)  
 .contentType("application/json")  
 .content("{\"wrongFieldName\":\"Item1\"}"))  
 .andExpect(*status*().isBadRequest());  
}

## Tag Tests schreiben

Kopiere die Unittest Methoden in die Klasse TagControllerTest und passe den Code an, sodass die entsprechenden Methoden auf dem TagController ebenfalls getestet werden. Du kannst noch mehr der Tests adaptieren.