

CRUD

Backend og frontend

DS3103 Webutvikling

Rolando Gonzalez, 2022

Innhold

- Utgangspunkt for kodeeksemplene gitt i slideserien
- Backend: Controller
- Frontend: Service med http request til Web Api
- Om POST og id
- Om POST og datatype
- Om PUT

Utgangspunkt for kodeeksemplene gitt i slideserien

- Temaet er et TodoApi hvor man eksempelvis kan lagre Todos (gjøremål) med id, tittel og minutter en todo tar.
- I denne slideserien tas det utgangspunkt i at det er opprettet en database. Koden for backend viser en Controllers metoder (endepunkter) og hvordan de utfører CRUD mot databasen.

Backend Controller

Starten på Controlleren

- Henter inn ressursene, blant annet Context-klassen og Todo-Model-klassen.
- Initierer context-objekt som lar oss gjøre CRUD mot databasen.

```
C# TodoController.cs X
○ TodoApi > Controllers > C# TodoController.cs > {} TodoApi.Controll

1  using Microsoft.AspNetCore.Mvc;
2  using Microsoft.EntityFrameworkCore;
3  using TodoApi.Contexts;
4  using TodoApi.Models;
5
6  namespace TodoApi.Controllers;
7
8  [ApiController]
9  [Route("[controller]")]
   0 references
10 public class TodoController : ControllerBase
11 {
   10 references
12     private readonly TodoContext context;
13
   0 references
14     public TodoController(TodoContext _context)
15     {
16         context = _context;
17     }
```

GET-metodene

- De 2 standard-GET-metodene er inkludert her: den om henter ut alle Todo fra databasen, og den som henter ut 1 Todo etter id.

```
19      [HttpGet]
20      0 references
21      public async Task<ActionResult<List<Todo>>> Get()
22      {
23          List<Todo> todos = await context.Todos.ToListAsync();
24          return todos;
25      }
26
27      [HttpGet("{id}")]
28      0 references
29      public async Task<ActionResult<Todo>> Get(int id)
30      {
31          Todo? todo = await context.Todos.FindAsync(id);
32          if( todo != null)
33          {
34              return Ok(todo);
35          }
36          else
37          {
38              return NotFound();
39          }
40      }
```

POST-metoden

- POST-metoden skal ta imot en ny Todo og legge den til i tabellen for Todoer i databasen.
- Den returnerer den nettopp lagt til Todo.

```
39
40  [HttpPost]
    0 references
41  public async Task<ActionResult<Todo>> Post(Todo newTodo)
42  {
43      context.Todos.Add(newTodo);
44      await context.SaveChangesAsync();
45      return CreatedAtAction("Get", new {id = newTodo.Id}, newTodo);
46  }
47
```

DELETE-metoden

- DELETE-metoden tar imot id til den Todo som skal slettes.
- Den returnerer NoContent() som er en 204-melding som betyr «alt er OK, trenger ikke returnere noe»

```
47
48     [HttpDelete("{id}")]
      0 references
49     public async Task<IActionResult> Delete(int id)
50     {
51         Todo? todo = await context.Todos.FindAsync(id);
52         if(todo != null){
53             context.Todos.Remove(todo);
54             await context.SaveChangesAsync();
55         }
56         return NoContent();
57     }
```


PUT-metoden

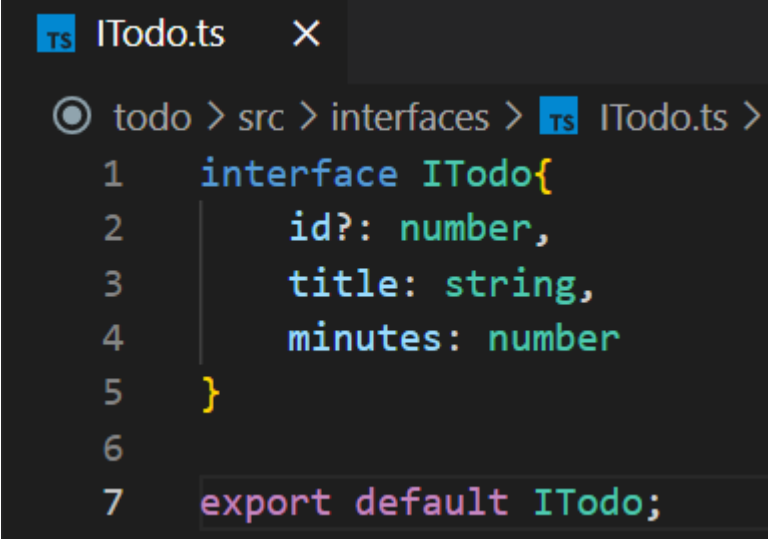
- PUT-metoden ligner på POST-metoden på den måten at den også skal ta imot et Todo-objekt, men mens POST legger til ny Todo, så skal PUT overskrive en allerede eksisterende Todo.

```
58
59     [HttpPut]
      0 references
60     public async Task<IActionResult> Put(Todo editedTodo)
61     {
62         context.Entry(editedTodo).State = EntityState.Modified;
63         await context.SaveChangesAsync();
64         return NoContent();
65     }
```

Frontend Service

Interface IToDo

- I React-prosjektet med TypeScript matches Model-klassen med et interface.
- Vi ser her at id er satt som optional, noe som er viktig med tanke på at POST ikke skal inneholde en id siden backend-databasen autogenerer ider for oss.



```
TS IToDo.ts  X
● todo > src > interfaces > TS IToDo.ts >
1  interface IToDo{
2      id?: number,
3      title: string,
4      minutes: number
5  }
6
7  export default IToDo;
```

Starten på Service med GET-metoder

- Man oppretter Service.
- Endepunktet deklarerer og initieres øverst.
- Koden viser hvordan GET-metodene i Web Api kalles på.
- De neste slidene viser utvidelsen av Servicen

```
TS TodoService.ts X
○ todo > src > services > TS TodoService.ts > ...
1  import axios from "axios";
2  import IToDo from "../interfaces/ITodo";
3
4  const TodoService = (
5    () => {
6
7      const todoEndpoint = "https://localhost:7004/todo";
8
9      const getAll = async () => {
10         const result = await axios.get(todoEndpoint);
11         return result.data;
12       }
13
14       const getById = async (id: number) => {
15         const result = await axios.get(`${todoEndpoint}/${id}`);
16         return result.data;
17       }
18
19       return { getAll, getById }
20     }
21   )();
22
23
24  export default TodoService;
```

Service: POST

- Service sin POST-funksjon tar imot et Todo-objekt fra komponenten som gjør bruk av Service.
- `axios.post()` inneholder to argumenter: url til `TodoController` og den nye `Todo`en

```
18
19     const postTodo = async (newTodo: ITodo) => {
20         const result = await axios.post(todoEndpoint, newTodo);
21         console.log( result );
22         //Ideelt sjekke status om gikk OK eller ikke og handle deretter
23     }
24
```

Service: DELETE

- Delete-funksjonen i Service trenger å få tilsendt id til Todo som skal slettes fra komponenten som bruker Service.
- Som det står i kommentaren, som gjelder for alle HTTP requestene, så bør man egentlig sjekke hva result sier og evt. returnere noe ut fra funksjonen slik at riktig handling kan gjøres i grensesnittet-

```
24
25     const deleteTodo = async (id: number) => {
26         const result = await axios.delete(`${todoEndpoint}/${id}`);
27         console.log( result );
28         //Ideelt sjekke status om gikk OK eller ikke og handle deretter
29     }
30
```

Service: PUT

- PUT-funksjonen tar imot et redigert Todo-objekt. Kodes ellers som vi ser likt som POST.

```
30
31     const putTodo = async (editedTodo: ITodo) => {
32         const result = await axios.put(todoEndpoint, editedTodo);
33         console.log( result );
34         //Ideelt sjekke status om gikk OK eller ikke og handle deretter
35     }
36
```

Komprimert overblikk over TodoService

```
TodoService.ts X
todo > src > services > TodoService.ts > ...
1  import axios from "axios";
2  import ITodo from "../interfaces/ITodo";
3
4  const TodoService = (
5    () => {
6
7      const todoEndpoint = "https://localhost:7004/todo";
8
9      const getAll = async () => {
10         const result = await axios.get(todoEndpoint);
11         return result.data;
12     }
13
14     const getById = async (id: number) => {
15         const result = await axios.get(`${todoEndpoint}/${id}`);
16         return result.data;
17     }
18
19     const postTodo = async (newTodo: ITodo) => {
20         const result = await axios.post(todoEndpoint, newTodo);
21     }
22
23     const deleteTodo = async (id: number) => {
24         const result = await axios.delete(`${todoEndpoint}/${id}`);
25     }
26
27     const putTodo = async (editedTodo: ITodo) => {
28         const result = await axios.put(todoEndpoint, editedTodo);
29     }
30
31     return { getAll, getById, postTodo, putTodo, deleteTodo }
32   }
33 )();
34
35
36 export default TodoService;
```


Annet

Om POST og id

- Merk at databasen autogenerer id for oss.
- Dette betyr at når man oppretter nytt objekt i frontend som skal sendes med POST, så skal en ikke angi id.
- Dette betyr videre at interfacet i TypeScript for en type objekt (som matcher Model-klasse i Web Api) bør ha id som optional: `id?: number`

Om POST og datatype

- Merk at objektet som man sender fra frontend som ha properties som matcher properties til Model-klassen i backend.
- Som eksempel hvis en Model-klasse i Web Api som heter Todo har Title (string) og Minutes (int) så er det som følger:
 - Dette er **OK**: { title: "Jogge", minutes: 50 }
 - Dette er **ikke OK**, fordi 50 er string: { title: "Jogge", minutes: "50" }

Om PUT

- PUT kan være litt mer kompleks enn de andre 3 HTTP-metodene siden det ofte vil kunne kreve følgende et ekstra steg:
 - Man må først hente (GET) det du ønsker å endre ved å eksempelvis angi id
 - Endre det i grensesnittet
 - PUT endrede ting