

7.4-4: Minimal API mit MongoDB - Teil 4

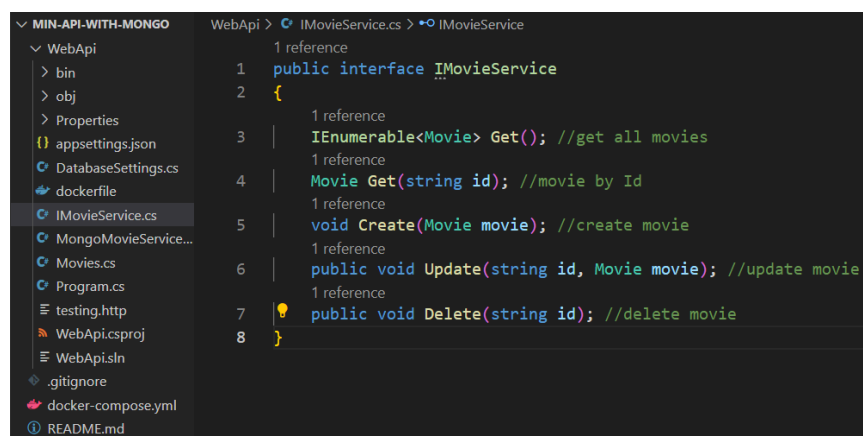
1. Ziele

- Sie erstellen einen MovieService mit CRUD-Operationen für eine MongoDB
- Sie verwenden den MovieService in den Endpunkten

2. MovieService erstellen

Alle Zugriffe auf die MongoDB sollen nun in eine eigene Komponente ausgelagert werden. In ASP.NET Core werden dazu Services verwendet, die dank dependency injection in der ganzen App verfügbar sind. Erstellen Sie dazu unter min-api-with-mongo/WebApi ein Interface IMovieService in Datei IMovieService.cs mit dem Codegerüst für die 5 Endpunkte:

```
public interface IMovieService
{
    IEnumerable<Movie> Get(); //get all movies
    Movie Get(string id); //movie by Id
    void Create(Movie movie); //create movie
    public void Update(string id, Movie movie); //update movie
    public void Delete(string id); //delete movie
}
```



Erstellen Sie im gleichen Verzeichnis ein Gerüst einer konkreten Implementierung als Klasse `MongoMovieService` (`MongoMovieService.cs`), die das Interface `IMovieService` implementiert.

```
using Microsoft.Extensions.Options;
using MongoDB.Driver;

public class MongoMovieService : IMovieService
{
    private readonly IMongoCollection<Movie> _movieCollection;
    private const string mongoDbDatabaseName = "gbs";
    private const string mongoDbCollectionName = "movies";

    // Constructor. Settings werden per dependency injection übergeben.
    public MongoMovieService(IOptions<DatabaseSettings> options)
    {
        var mongoDbConnectionString = options.Value.ConnectionString;
        var mongoClient = new MongoClient(mongoDbConnectionString);
        var database = mongoClient.GetDatabase(mongoDbDatabaseName);
        _movieCollection = database.GetCollection<Movie>(mongoDbCollectionName);
    }

    public void Create(Movie movie)
    {
        throw new NotImplementedException();
    }

    public IEnumerable<Movie> Get()
    {
        throw new NotImplementedException();
    }

    public Movie Get(string id)
    {
        throw new NotImplementedException();
    }
}
```

```

public void Update(string id, Movie movie)
{
    throw new NotImplementedException();
}

public void Delete(string id)
{
    throw new NotImplementedException();
}
}

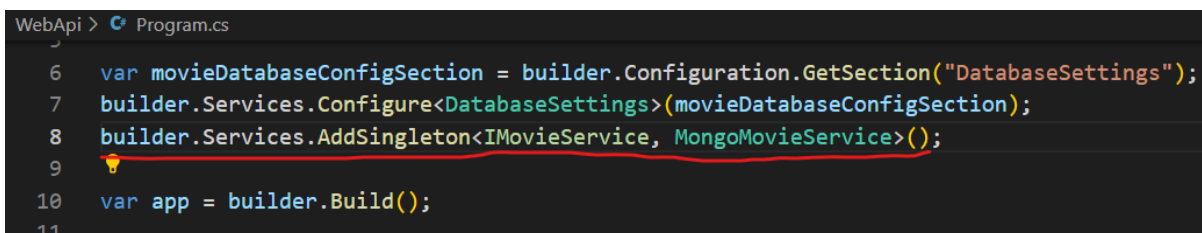
```

Im Konstruktor werden die DatabaseSettings übernommen und damit eine Referenz auf die angegebene Collection movies in der Datenbank gbs erstellt. Damit die DatabaseSettings via Dependency Injection übergeben werden, muss die Klasse in Program.cs als Singleton registriert werden:

```

...
builder.Services.AddSingleton<IMovieService, MongoMovieService>();
...

```



```

WebApi > Program.cs
6  var movieDatabaseConfigSection = builder.Configuration.GetSection("DatabaseSettings");
7  builder.Services.Configure<DatabaseSettings>(movieDatabaseConfigSection);
8  builder.Services.AddSingleton<IMovieService, MongoMovieService>();
9
10 var app = builder.Build();
11

```

3. MovieService verwenden

Der Movieservice kann nun in den 5 Endpunkten aufgerufen werden, hier der Endpunkt für get movie by id:

```

// Get Movie by id
app.MapGet("api/movies/{id}", (IMovieService movieService, string id) =>
{
    var movie = movieService.Get(id);
    return movie != null
        ? Results.Ok(movie)

```

```
        : Results.NotFound();  
    });
```

```
WebApi > Program.cs  
31  
32 // Get Movie by id  
33 app.MapGet("api/movies/{id}", (IMovieService movieService, string id) =>  
34 {  
35     var movie = movieService.Get(id);  
36     return movie != null  
37         ? Results.Ok(movie)  
38         : Results.NotFound();  
39 });  
40
```

Wie Sie sehen wird einfach die Get-Methode des MovieService aufgerufen und in Abhängigkeit des Resultates der Movie zurückgegeben oder NotFound.

Hier der Endpunkt des POST-Endpunktes:

```
// Insert Movie  
app.MapPost("/api/movies", (IMovieService movieService, Movie movie) =>  
{  
    movieService.Create(movie);  
    return Results.Ok(movie);  
});
```

```
WebApi > Program.cs  
102 // Insert Movie  
103 app.MapPost("/api/movies", (IMovieService movieService, Movie movie) =>  
104 {  
105     movieService.Create(movie);  
106     return Results.Ok(movie);  
107 });
```

Nun fehlt noch die Implementierung der Get-Methode:

```
public Movie Get(string id)  
{  
    return _movieCollection.Find(m => m.Id == id).FirstOrDefault();  
}
```

```

WebApi > MongoMovieService.cs > MongoMovieService
3 public class MongoMovieService : IMovieService
40
    2 references
29 public Movie Get(string id)
30 {
31     return _movieCollection.Find(m => m.Id == id).FirstOrDefault();
32 }
33

```

Die Create-Methode lässt sich so implementieren:

```

public void Create(Movie movie)
{
    _movieCollection.InsertOne(movie);
}

```

```

WebApi > MongoMovieService.cs > MongoMovieService
3 public class MongoMovieService : IMovieService
    1 reference
19 public void Create(Movie movie)
20 {
21     _movieCollection.InsertOne(movie);
22 }
23

```

Vervollständigen Sie nun die restlichen Methoden und Endpunkte. Als Hilfsmittel können Sie z.B. diese Referenz verwenden:

<https://www.mongodb.com/docs/drivers/csharp/current/quick-reference/>

▼ Delete

- Program.cs:

```

// Delete Movie
app.MapDelete("api/movies/{id}", (IMovieService movieService, string
{
    var movie = movieService.Get(id);
    if(movie == null)
    {
        return Results.NotFound();
    }

    movieService.Delete(id);
}

```

```
return Results.Ok();  
});
```

```
WebApi > Program.cs  
47 // Delete Movie  
48 app.MapDelete("api/movies/{id}", (IMovieService movieService, string id) =>  
49 {  
50     var movie = movieService.Get(id);  
51     if(movie == null)  
52     {  
53         return Results.NotFound();  
54     }  
55  
56     movieService.Delete(id);  
57     return Results.Ok();  
58 });
```

- MongoMovieService.cs:

```
public void Delete(string id)  
{  
    _movieCollection.DeleteOne(m => m.Id == id);  
}
```

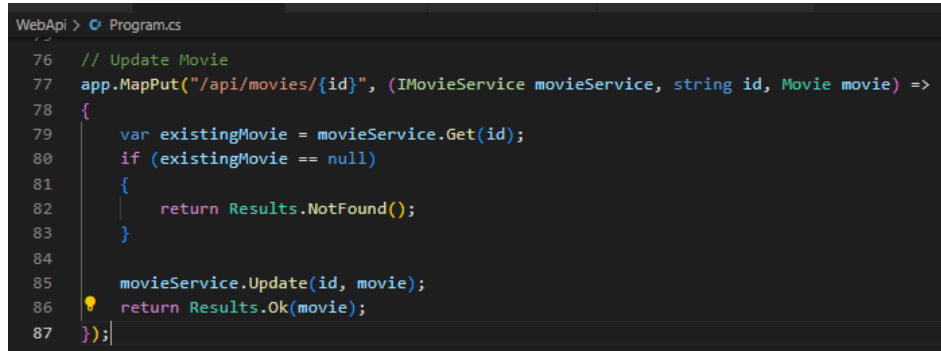
```
WebApi > MongoMovieService.cs > MongoMovieService  
4 public class MongoMovieService : IMovieService  
  
40     public void Delete(string id)  
41     {  
42         _movieCollection.DeleteOne(m => m.Id == id);  
43     }
```

▼ Update

- Program.cs:

```
// Update Movie  
app.MapPut("/api/movies/{id}", (IMovieService movieService, string id)  
{  
    var existingMovie = movieService.Get(id);  
    if (existingMovie == null)  
    {  
        return Results.NotFound();  
    }  
  
    movieService.Update(id, movie);
```

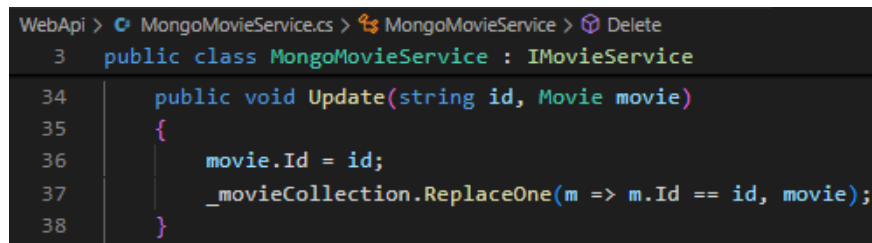
```
return Results.Ok(movie);
});
```



```
WebApi > Program.cs
76 // Update Movie
77 app.MapPut("/api/movies/{id}", (IMovieService movieService, string id, Movie movie) =>
78 {
79     var existingMovie = movieService.Get(id);
80     if (existingMovie == null)
81     {
82         return Results.NotFound();
83     }
84
85     movieService.Update(id, movie);
86     return Results.Ok(movie);
87 });
```

- MongoMovieService.cs:

```
public void Update(string id, Movie movie)
{
    movie.Id = id;
    _movieCollection.ReplaceOne(m => m.Id == id, movie);
}
```



```
WebApi > MongoMovieService.cs > MongoMovieService > Delete
3 public class MongoMovieService : IMovieService
34     public void Update(string id, Movie movie)
35     {
36         movie.Id = id;
37         _movieCollection.ReplaceOne(m => m.Id == id, movie);
38     }
```

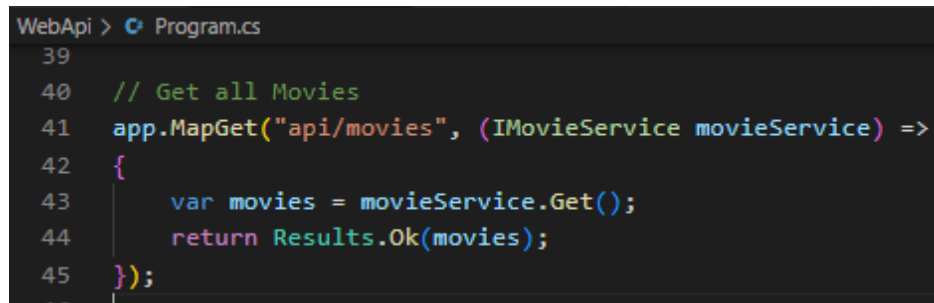
- public IEnumerable<Movie> Get():

```
public IEnumerable<Movie> Get()
{
    return _movieCollection.Find(_ => true).ToList();
}
```

- Get all Movies:

```
// Get all Movies
app.MapGet("api/movies", (IMovieService movieService) =>
{
    var movies = movieService.Get();
}
```

```
return Results.Ok(movies);  
});
```



```
WebApi > Program.cs  
39  
40 // Get all Movies  
41 app.MapGet("api/movies", (IMovieService movieService) =>  
42 {  
43     var movies = movieService.Get();  
44     return Results.Ok(movies);  
45 });
```

▼ **ganzer Code von MongoMovieService.cs**

```
using Microsoft.Extensions.Options;  
using MongoDB.Driver;  
  
public class MongoMovieService : IMovieService  
{  
    private readonly IMongoCollection<Movie> _movieCollection;  
    private const string mongoDbDatabaseName = "gbs";  
    private const string mongoDbCollectionName = "movies";  
  
    public MongoMovieService(IOptions<DatabaseSettings> options)  
    {  
        var mongoDbConnectionString = options.Value.ConnectionString;  
  
        var mongoClient = new MongoClient(mongoDbConnectionString);  
        var database = mongoClient.GetDatabase(mongoDbDatabaseName);  
        _movieCollection = database.GetCollection<Movie>(mongoDbCollectionName);  
    }  
  
    public IEnumerable<Movie> Get()  
    {  
        return _movieCollection.Find(m => true).ToList();  
    }  
  
    public Movie Get(string id)  
    {  
        return _movieCollection.Find(m => m.Id == id).FirstOrDefault();  
    }  
}
```



```

    }

    public void Create(Movie movie)
    {
        _movieCollection.InsertOne(movie);
    }

    public void Update(string id, Movie movie)
    {
        movie.Id = id;
        _movieCollection.ReplaceOne(m => m.Id == id, movie);
    }

    public void Delete(string id)
    {
        _movieCollection.DeleteOne(m => m.Id == id);
    }
}

```

▼ **ganzer Code von Program.cs**

```

var builder = WebApplication.CreateBuilder(args);

var movieDatabaseConfigSection = builder.Configuration.GetSection("Data");
builder.Services.Configure<DatabaseSettings>(movieDatabaseConfigSection);
builder.Services.AddSingleton<IMovieService, MongoMovieService>();

var app = builder.Build();

app.MapGet("/", () => "Movies API");

// Get all Movies
app.MapGet("api/movies", (IMovieService movieService) =>
{
    var movies = movieService.Get();
    return Results.Ok(movies);
});

```

```

// Get Movie by id
app.MapGet("api/movies/{id}", (IMovieService movieService, string id) =>
{
    var movie = movieService.Get(id);
    return movie != null
        ? Results.Ok(movie)
        : Results.NotFound();
});

// Insert Movie
app.MapPost("/api/movies", (IMovieService movieService, Movie movie) =
{
    movieService.Create(movie);
    return Results.Ok(movie);
});

// Update Movie
app.MapPut("/api/movies/{id}", (IMovieService movieService, string id, M
{
    var existingMovie = movieService.Get(id);
    if (existingMovie == null)
    {
        return Results.NotFound();
    }

    movieService.Update(id, movie);
    return Results.Ok(movie);
});

// Delete Movie
app.MapDelete("api/movies/{id}", (IMovieService movieService, string id)
{
    var movie = movieService.Get(id);
    if(movie == null)
    {
        return Results.NotFound();
    }
}

```

```
movieService.Delete(id);  
return Results.Ok();  
});  
  
app.Run();
```

4. Testen mit REST-Client

Die Datenbank und die Collection enthalten zu Beginn keine Filme. Filme können Sie mit einem POST-Request mit Hilfe des REST-Clients hinzufügen. Sie können die bereits zuvor erstellte Datei testing.http verwenden. Fügen Sie den POST-Request nach drei Hashtags ### hinzu.

Beachten Sie, dass Sie im Header des Request den Content-Type: application/json angeben müssen. Der Body mit den JSON-Daten muss durch eine zusätzliche Zeile abgetrennt sein. Die Struktur der JSON-Daten muss der Struktur der Movie-Klasse entsprechen. Weitere Requests können Sie nach weiteren Hashtags hinzufügen.

POST:

```
POST http://localhost:5001/api/movies  
content-type: application/json  
  
{  
  "id": "1",  
  "title": "The Imitation Game",  
  "year": "2014",  
  "summary": "Das wahre Rätsel war der Mann, der den Code knackte",  
  "actors": ["Benedict Cumberbatch", "Keira Knightley"]  
}
```

DELETE:

```
DELETE http://localhost:5001/api/movies/1
```