7.4-4 MovieService mit CRUD-Operationen

1. MovieService erstellen

Unter **min-api-with-mongo/WebApi** ein Interface **IMovieService** in Datei IMovieService.cs erstellen:

```
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
}
```

Im gleichen Verzeichnis ein Gerüst einer konkreten Implementierung als Klasse **MongoMovieService** (MongoMovieService.cs) erstellen, 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 Programm.cs als Singleton registriert werden:

```
...
var builder = WebApplication.CreateBuilder(args);

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

var app = builder.Build();
...
```

2. MovieService verwenden

GET “/api/movies/{id}”

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

```
// Get Movie by id
// Gibt das gewünschte Movie-Objekt mit Statuscode 200 OK zurück.
// Bei ungültiger id wird Statuscode 404 not found zurückgegeben.
app.MapGet("/api/movies/{id}", (IMovieService movieService, string id) =>
{
    var movie = movieService.Get(id);
    return movie != null
        ? Results.Ok(movie)
```

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

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

Get(string id)

in MongoMovieService.cs

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

POST "/api/movies"

```
// Insert Movie
// Wenn das übergebene Objekt eingefügt werden konnte,
// wird es mit Statuscode 200 zurückgegeben.
// Bei Fehler wird Statuscode 409 Conflict zurückgegeben.
app.MapPost("/api/movies", (IMovieService movieService, Movie movie) =>
{
    if (String.IsNullOrWhiteSpace(movie.Id) || String.IsNullOrWhiteSpace(movie.Title))
    {
        return Results.Problem("ID and Title can't be null.");
    }
    else if (movieService.Get(movie.Id) == null)
    {
        movieService.Create(movie);
        return Results.Ok(movie);
    }
    else
    {
        return Results.Conflict($"Movie with ID {movie.Id} already exists.");
    }
});
```

Create(Movie movie)

in MongoMovieService.cs

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

GET “/api/movies”

```
// Get all Movies
// Gibt alle vorhandenen Movie-Objekte mit Statuscode 200 OK zurück.
app.MapGet("/api/movies", (IMovieService movieService) ⇒
{
    IEnumerable<Movie> movies = movieService.Get();
    return Results.Ok(movies);
});
```

Get()

in MongoMovieService.cs

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

PUT “/api/movies/{id}”

```
// Update Movie
// Gibt das aktualisierte Movie-Objekt zurück.
// Bei ungültiger id wird Statuscode 404 not found zurückgegeben.
app.MapPut("/api/movies/{id}", (IMovieService movieService, string id, Movie movie) ⇒
{
    if (movieService.Get(id) != null)
    {
        movieService.Update(id, movie);
        return Results.Ok();
    }
    else
    {
        return Results.NotFound();
    }
});
```

Update(string id, Movie movie)

in MongoMovieService.cs

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

OR

```
public void Update(string id, Movie movie)
{
    var update = Builders<Movie>.Update
        .Set(m => m.Title, movie.Title)
        .Set(m => m.Year, movie.Year)
        .Set(m => m.Summary, movie.Summary)
        .Set(m => m.Actors, movie.Actors);

    _movieCollection.UpdateOne(m => m.Id == id, update);
}
```

DELETE "/api/movies/{id}"

```
// Delete Movie
// Gibt bei erfolgreicher Löschung Statuscode 200 OK zurück.
// Bei ungültiger id wird Statuscode 404 not found zurückgegeben.
app.MapDelete("/api/movies/{id}", (IMovieService movieService, string id) =>
{
    if (movieService.Get(id) != null)
    {
        movieService.Delete(id);
        return Results.Ok();
    }
    else
    {
        return Results.NotFound();
    }
});
```

Delete(string id)

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

3. Testen mit REST-Client

```
### Get all movies
GET http://localhost:5001/api/movies

### Get movie by Id
GET http://localhost:5001/api/movies/1

### Create new movie (1)
POST http://localhost:5001/api/movies
Content-Type: application/json

{
  "id": "1",
  "title": "The Great Adventure",
  "year": 2020,
  "summary": "A thrilling journey across continents to uncover a hidden secret.",
  "actors": [
    "Alice Johnson",
    "Bob Smith",
    "Catherine Lee"
  ]
}

### Create new movie (2)
POST http://localhost:5001/api/movies
Content-Type: application/json

{
  "id": "4",
  "title": "Silent Echoes",
  "year": 2023,
  "summary": "A gripping drama about memory and identity in a futuristic society.",
  "actors": [
    "Daniel Craig",
    "Emma Stone"
  ]
}

### Update movie
PUT http://localhost:5001/api/movies/2
Content-Type: application/json

{
  "id": "2",
  "title": "!MODIFIED! The Great Adventure",
  "year": 2023,
```

```
  "summary": "!MODIFIED! A thrilling journey across continents to uncover a hidden secret.",
  "actors": [
    "!MODIFIED! Daniel Craig",
    "!MODIFIED! Emma Stone"
  ]
}

### Delete movie
DELETE http://localhost:5001/api/movies/4
```