

# LB295

## Ylli Karakushi I21a

### Contents

Projektantrag .....	3
Aufgabe 1 .....	4
Installation einer Entwicklungsumgebung für ASP.NET Core APIs .....	4
Verwendung geeigneter Erweiterungen für die Programmierung von APIs .....	4
Implementierung eines vorgegebenen Projekts .....	4
Identifikation und Installation der Entwicklungskomponenten .....	4
Vornahme notwendiger Einstellungen für eine problemlose Ausführung des Projekts .....	5
Diagnostizieren und Beheben von Problemen in der Umgebung .....	5
Aufgabe 2 .....	6
Dokumentation der API-Schnittstelle mit Swagger Tool .....	6
Implementierung einer einfachen Anwendung mittels REST APIs (CRUD) .....	7
Erstellung einer REST API Schnittstelle für die Datenmanipulation in einer Datenbank (CRUD) .....	7
Identifikation des passenden Schnittstellen-Standards .....	8
Anwendung von Implementierungstechniken für den vorgegebenen Schnittstellen-Standard .....	8
Aufgabe 3 .....	10
Anwendung serverseitiger Validierung .....	10
Überprüfung der Übereinstimmung des Endprodukts mit funktionalen, nicht-funktionalen und sicherheitsrelevanten Anforderungen .....	10
Anwendung wichtiger Sicherheitsmaßnahmen im Umgang mit Eingabedaten .....	10
Verwendung verschiedener Techniken zum Testen der Anforderungen (automatisch und von Hand) .....	11
Aufgabe 4 .....	11
Konfiguration der Entwicklungsumgebung für die Überprüfung von Coderichtlinien .....	11
Überprüfung der Einhaltung von Coderichtlinien und Korrektur gegebenenfalls .....	11
Aufgabe 5 .....	11
Effektive Verwaltung von Änderungen und Erweiterungen im Softwareverwaltungssystem .....	11
Bedienung des Softwareverwaltungssystems und Ablage von Änderungen und Erweiterungen .....	12
Nutzung des Softwareverwaltungssystems zur Behebung von Fehlern, Problemlösung und Verbesserung des Endprodukts .....	14
Aufgabe 6 .....	15
Implementierung eines aktuellen Authentifizierungsmechanismus im Back-End .....	15
Schutz vor anonymen Zugriffen in mindestens einem Bereich des Back-Ends .....	17
Übereinstimmung des Authentifizierungsmechanismus im Back-End mit Anforderungen an Sicherheit und Datenschutz .....	17

## Bewertung:

Aufgabe	Gewichtung	Bewertung
1.1	1	Installation einer Entwicklungsumgebung für ASP.NET Core APIs
1.2	1	Verwendung geeigneter Erweiterungen für die Programmierung von APIs
1.3	3	Implementierung eines vorgegebenen Projekts
1.4	1	Identifikation und Installation der Entwicklungskomponenten
1.5	1	Vornahme notwendiger Einstellungen für eine problemlose Ausführung des Projekts
1.6	1	Diagnostizieren und Beheben von Problemen in der Umgebung
2.1	2	Dokumentation der API-Schnittstelle mit Swagger Tool
2.2	2	Implementierung einer einfachen Anwendung mittels REST APIs (CRUD)
2.3	3	Erstellung einer REST API Schnittstelle für die Datenmanipulation in einer Datenbank (CRUD)
2.4	1	Identifikation des passenden Schnittstellen-Standards
2.5	1	Anwendung von Implementierungstechniken für den vorgegebenen Schnittstellen-Standard
3.1	3	Anwendung serverseitiger Validierung
3.2	1	Überprüfung der Übereinstimmung des Endprodukts mit funktionalen, nicht-funktionalen und sicherheitsrelevanten Anforderungen
3.3	1	Anwendung wichtiger Sicherheitsmaßnahmen im Umgang mit Eingabedaten
3.4	2	Verwendung verschiedener Techniken zum Testen der Anforderungen (automatisch und von Hand)
4.1	1	Konfiguration der Entwicklungsumgebung für die Überprüfung von Coderichtlinien
4.2	2	Überprüfung der Einhaltung von Coderichtlinien und Korrektur gegebenenfalls
5.1	1	Effektive Verwaltung von Änderungen und Erweiterungen im Softwareverwaltungssystem
5.2	2	Bedienung des Softwareverwaltungssystems und Ablage von Änderungen und Erweiterungen
5.3	1	Nutzung des Softwareverwaltungssystems zur Behebung von Fehlern, Problemlösung und Verbesserung des Endprodukts
6.1	2	Implementierung eines aktuellen Authentifizierungsmechanismus im Back-End
6.2	1	Schutz vor anonymen Zugriffen in mindestens einem Bereich des Back-Ends
6.3	1	Übereinstimmung des Authentifizierungsmechanismus im Back-End mit Anforderungen an Sicherheit und Datenschutz
	1	Jeder Bewertungspunkt wird durch einen zusätzlichen Screenshot zum Text belegt
	1	Administrative Vorgaben sind eingehalten

## Projektantrag:

### *Beschreibung:*

Ich plane die Entwicklung einer Terminapp als Web API mit C#. Die App soll es Benutzern ermöglichen, Termine zu erstellen, zu bearbeiten, zu löschen und zu lesen. Dabei möchte ich eine Schnittstelle schaffen, die eine einfache und intuitive Verwaltung von Terminen ermöglicht.

### *Anforderungen:*

Die Hauptanforderung besteht darin, dass Benutzer über die API Termine erstellen, bearbeiten, löschen und lesen können. Die API sollte über entsprechende Endpunkte verfügen, um diese Operationen zu unterstützen und eine nahtlose Interaktion mit der Terminapp zu ermöglichen.

### *Testfälle:*

Die Testfälle sollen sicherstellen, dass die grundlegenden Funktionen der Terminapp ordnungsgemäss implementiert sind. Dazu gehören:

#### Testfall 1: Termin erstellen

Über die API wird ein neuer Termin erstellt.

Die erstellten Termindetails werden korrekt gespeichert.

#### Testfall 2: Termin bearbeiten

Ein vorhandener Termin wird über die API aktualisiert.

Die geänderten Termindetails werden korrekt aktualisiert und gespeichert.

#### Testfall 3: Termin löschen

Ein vorhandener Termin wird über die API gelöscht.

Der gelöschte Termin wird nicht mehr in der Terminliste angezeigt.

#### Testfall 4: Termin lesen

Über die API werden die vorhandenen Termine abgerufen.

Die abgerufenen Termindetails werden korrekt angezeigt und können gelesen werden.

#### Testfall 5: Login & Register

Über die kann der Benutzer sich registrieren und einloggen.

Der Benutzer sieht eine erfolgreiche Rückmeldung.

#### Testfall 6: Passwort entschlüsseln

Das Passwort wird entschlüsselt gespeichert bei login.

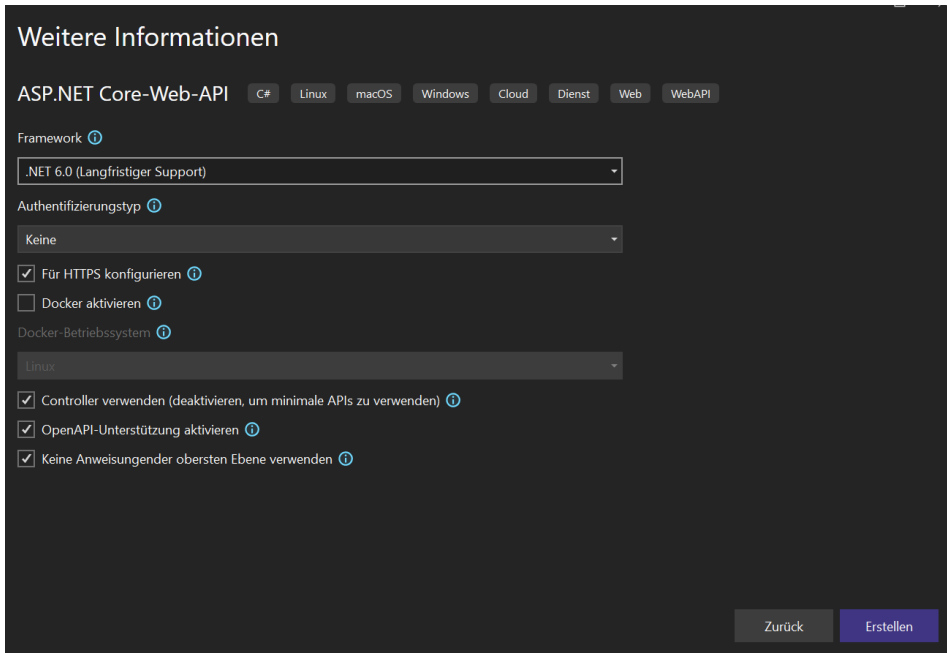
Das entschlüsselte Passwort wird gezeigt.

Diese Testfälle sollen sicherstellen, dass alle grundlegenden CRUD-Funktionen (Erstellen, Lesen, Aktualisieren, Löschen) für Termine erfolgreich durch die API ausgeführt werden können.

## Aufgabe 1

### *Installation einer Entwicklungsumgebung für ASP.NET Core APIs*

ASP.NET und Webentwicklung mit Visual Studio 2022 installieren und Projekt erstellen:



The screenshot shows the 'Weitere Informationen' (More Information) dialog in Visual Studio 2022. The title is 'Weitere Informationen'. Below it, the project type is 'ASP.NET Core-Web-API'. There are tabs for 'C#', 'Linux', 'macOS', 'Windows', 'Cloud', 'Dienst', 'Web', and 'WebAPI'. The 'Framework' dropdown is set to '.NET 6.0 (Langfristiger Support)'. The 'Authentifizierungstyp' (Authentication Type) dropdown is set to 'Keine' (None). There are three checkboxes: 'Für HTTPS konfigurieren' (checked), 'Docker aktivieren' (unchecked), and 'Docker-Betriebssystem' (Linux). Below these, there are three more checkboxes: 'Controller verwenden (deaktivieren, um minimale APIs zu verwenden)' (checked), 'OpenAPI-Unterstützung aktivieren' (checked), and 'Keine Anweisungender obersten Ebene verwenden' (checked). At the bottom right, there are two buttons: 'Zurück' (Back) and 'Erstellen' (Create).

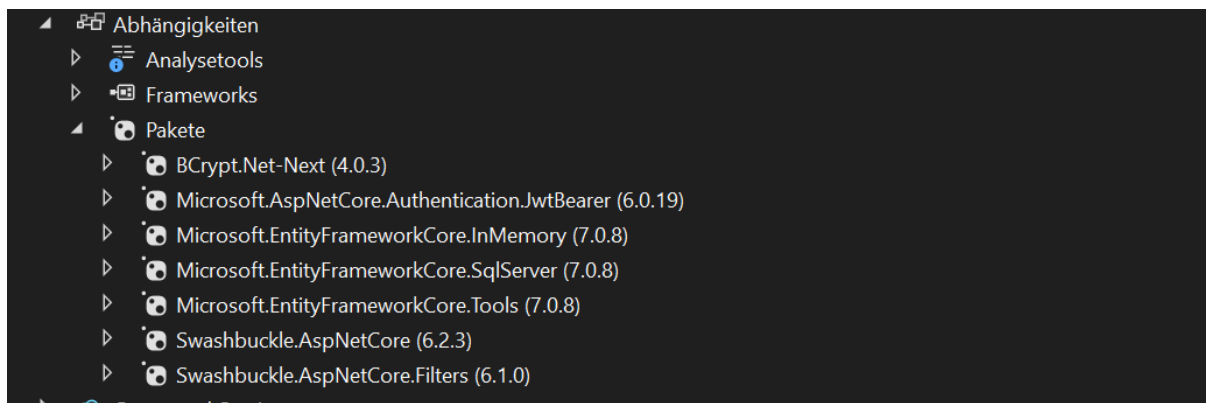
*Verwendung geeigneter Erweiterungen für die Programmierung von APIs*  
Swagger



### *Implementierung eines vorgegebenen Projekts*

Ich habe ein TerminPlaner-Projekt erstellt und eine ZIP-Datei davon erstellt.

### *Identifikation und Installation der Entwicklungskomponenten*



### *Vornahme notwendiger Einstellungen für eine problemlose Ausführung des Projekts*

Zu Beginn ist es erforderlich, einen eigenen Microsoft SQL Server einzurichten und anschliessend eine Datenbank zu erstellen. Zusätzlich muss der Server in der Datei "appsettings.json" deklariert werden:

"TerminApiConnectionString":

"Server=Name;Database=Name;Trusted\_Connection=true;TrustServerCertificate=False;Encrypt=false  
;"

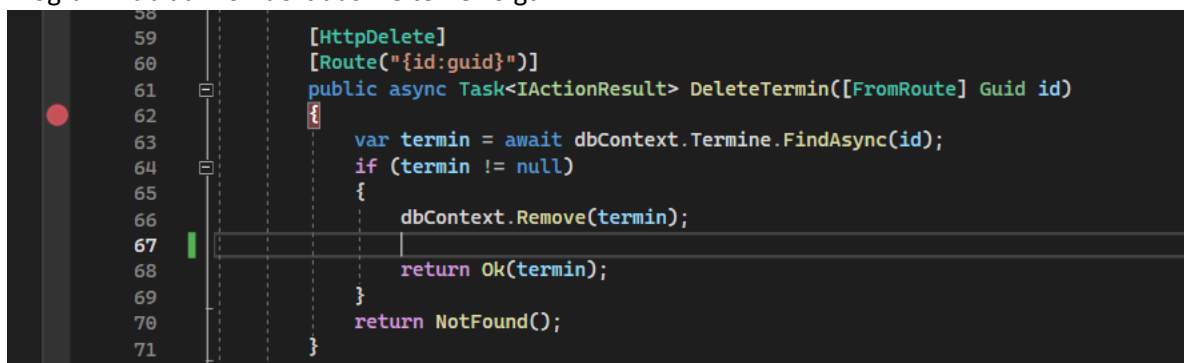
Kurz vor dem Ende ist es erforderlich, in der NuGet-Konsole den Befehl "Update-Database" auszuführen. Abschliessend kann das Projekt mit Visual Studio gestartet werden.

### *Diagnostizieren und Beheben von Problemen in der Umgebung*

Mit Visual Studio: Während des Programmablaufs kann man die Verwendung von Variablen verfolgen und ihren Inhalt überwachen. Man kann auch den Code Schritt für Schritt durchgehen und sehen, wie das Programm ausgeführt wird. Dies erleichtert das Finden und Beheben von Fehlern erheblich.

Beispiel:

Nachdem meine DELETE-Anfrage nicht funktioniert hat, habe ich einen Breakpoint gesetzt und den Programmablauf von dort aus weiterverfolgt.



Dabei ist mir aufgefallen, dass es sich um einen logischen Fehler handelte – ich habe vergessen, die Änderungen zu speichern.

Lösung:

await dbContext.SaveChangesAsync();

## Aufgabe 2

### *Dokumentation der API-Schnittstelle mit Swagger Tool*

Auth		^
POST	/api/Auth/register	▼
POST	/api/Auth/login	▼
Termin		^
GET	/api/Termin	▼
POST	/api/Termin	▼ 🔒
PUT	/api/Termin/{id}	▼
DELETE	/api/Termin/{id}	▼
GET	/api/Termin/{id}	▼

Mit dem Swagger-Tool ist es einfach möglich, über den Browser verschiedene implementierte Requests zu sehen und auszuführen. Es bietet eine benutzerfreundliche Oberfläche, die es ermöglicht, alle verfügbaren Schemas einzusehen. Dadurch kann man die API-Dokumentation interaktiv erkunden und die verfügbaren Endpunkte und deren Parameter überblicken. Swagger erleichtert somit das Testen und Verstehen einer API.

## Implementierung einer einfachen Anwendung mittels REST APIs (CRUD)

```
[HttpGet]
public async Task<IActionResult> GetTermin()
{
    return Ok(await dbContext.Termin.ToListAsync()); ;
}

[HttpPost, Authorize]
public async Task<IActionResult> AddTermin(AddTermin addTermin)
{
    var termin = new Termin()
    {
        Id = Guid.NewGuid(),
        Datum = addTermin.Datum,
        Beschreibung = addTermin.Beschreibung
    };
    await dbContext.Termin.AddAsync(termin);
    await dbContext.SaveChangesAsync();

    return Ok(termin);
}

[HttpPut]
[Route("{id:guid}")]
public async Task<IActionResult> UpdateTermin([FromRoute] Guid id, UpdateTermin updateTerminRequest)
{
    var termin = await dbContext.Termin.FindAsync(id);
    if (termin != null)
    {
        termin.Datum = updateTerminRequest.Datum;

        await dbContext.SaveChangesAsync();

        return Ok(termin);
    }

    return NotFound();
}

[HttpDelete]
[Route("{id:guid}")]
public async Task<IActionResult> DeleteTermin([FromRoute] Guid id)
{
    var termin = await dbContext.Termin.FindAsync(id);
    if (termin != null)
    {
        dbContext.Remove(termin);
        await dbContext.SaveChangesAsync();

        return Ok(termin);
    }
    return NotFound();
}

[HttpGet]
[Route("{id:guid}")]
public async Task<IActionResult> GetTermin([FromRoute] Guid id)
{
    var termin = await dbContext.Termin.FindAsync(id);
    if (termin == null)
    {
        return NotFound();
    }
    return Ok(termin);
}
```

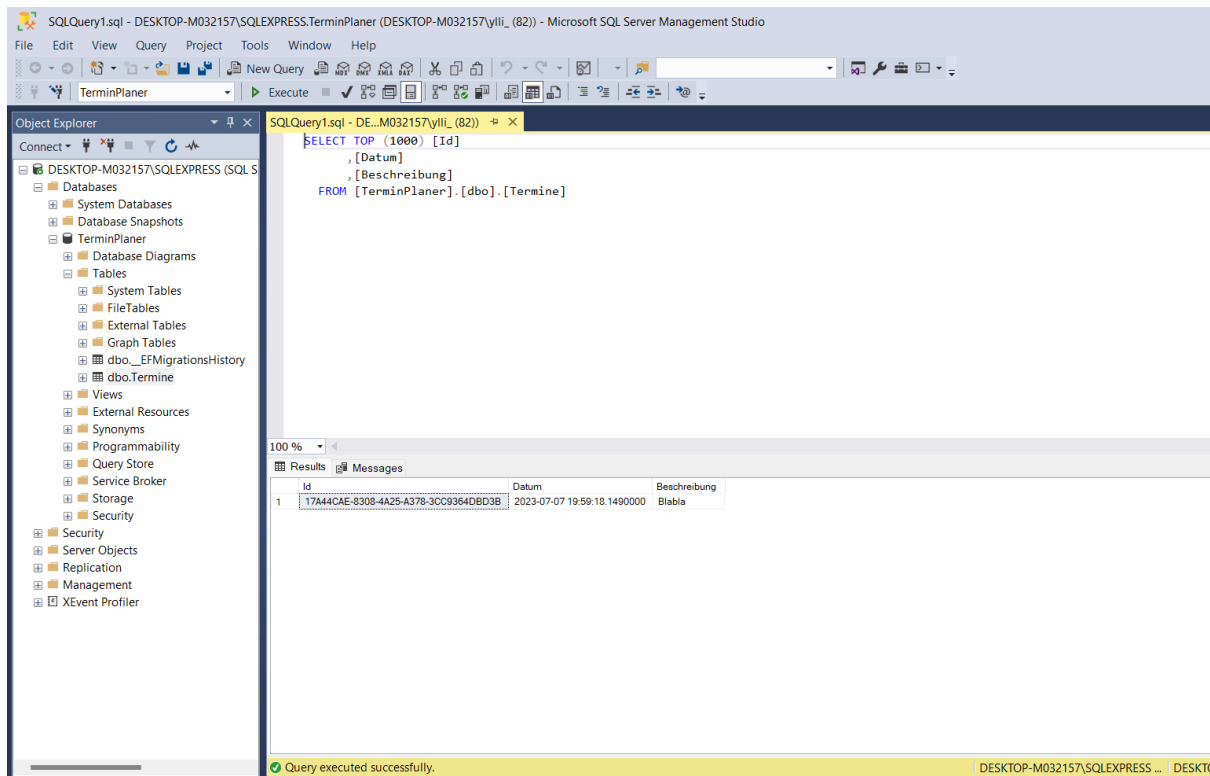
## Erstellung einer REST API Schnittstelle für die Datenmanipulation in einer Datenbank (CRUD)

```
[HttpPost, Authorize]
public async Task<IActionResult> AddTermin(AddTermin addTermin)
{
    var termin = new Termin()
    {
        Id = Guid.NewGuid(),
        Datum = addTermin.Datum,
        Beschreibung = addTermin.Beschreibung
    };
    await dbContext.Termin.AddAsync(termin);
    await dbContext.SaveChangesAsync();

    return Ok(termin);
}
```

dbContext ist die Datenbank.

Nun starte ich das Programm und log mich ein und mache eine Post request:



### *Identifikation des passenden Schnittstellen-Standards*

REST (Representational State Transfer) ist ein Architekturstil für Netzwerkanwendungen, der auf dem HTTP-Protokoll basiert. REST definiert Konventionen für die Gestaltung von APIs. Dabei werden HTTP-Methoden wie GET, POST, PUT und DELETE verwendet, um Ressourcen zu lesen, zu erstellen, zu aktualisieren und zu löschen. Durch die Verwendung dieser Standard-HTTP-Methoden wird eine einheitliche und intuitive Schnittstelle für die Kommunikation zwischen Client und Server geschaffen. REST legt auch Wert auf die Darstellung von Ressourcen in einem einheitlichen Format, wie zum Beispiel JSON oder XML. Dies ermöglicht eine lose Kopplung zwischen Client und Server und unterstützt die Skalierbarkeit und Wiederverwendbarkeit von APIs.

### *Anwendung von Implementierungstechniken für den vorgegebenen Schnittstellen-Standard*

Mithilfe der HTTP-Methoden (GET, POST, PUT, DELETE) werden im Code CRUD-Operationen gemäß dem REST-Prinzip implementiert. Durch Hinzufügen des [ApiController]-Attributs wird der Controller als RESTful API-Controller gekennzeichnet, wodurch verschiedene Funktionen und Eigenschaften für die Behandlung der API aktiviert werden. Das [Route]-Attribut wird genutzt, um die Routenstruktur für die verschiedenen Endpunkte der API festzulegen. Dadurch wird eine klare Zuordnung von URLs zu den entsprechenden Controller-Methoden ermöglicht, um die Manipulation der Ressourcen durchzuführen.

Hier zum Beispiel eine Get, Post und Put Methode:



```

[HttpGet]
public async Task<IActionResult> GetTermin()
{
    return Ok(await dbContext.Termin.ToListAsync()); ;
}

[HttpPost, Authorize]
public async Task<IActionResult> AddTermin(AddTermin addTermin)
{
    var termin = new Termin()
    {
        Id = Guid.NewGuid(),
        Datum = addTermin.Datum,
        Beschreibung = addTermin.Beschreibung
    };
    await dbContext.Termin.AddAsync(termin);
    await dbContext.SaveChangesAsync();

    return Ok(termin);
}

[HttpPut]
[Route("{id:guid}")]
public async Task<IActionResult> UpdateTermin([FromRoute] Guid id, UpdateTermin updateTerminRequest)
{
    var termin = await dbContext.Termin.FindAsync(id);
    if (termin != null)
    {
        termin.Datum = updateTerminRequest.Datum;

        await dbContext.SaveChangesAsync();

        return Ok(termin);
    }

    return NotFound();
}

```

## Aufgabe 3

### Anwendung serverseitiger Validierung

```
[HttpPost, Authorize]
0 references | karakushi, 13 minutes ago | 1 author, 1 change
public async Task<IActionResult> AddTermin(AddTermin addTermin)
{
    var termin = new Termin()
    {
        Id = Guid.NewGuid(),
        Datum = addTermin.Datum,
        Beschreibung = addTermin.Beschreibung
    };
    if (termin.Datum.Day < DateTime.Now.Day)
    {
        return BadRequest("Falsches Datum");
    }

    await dbContext.Termin.AddAsync(termin);
    await dbContext.SaveChangesAsync();

    return Ok(termin);
}
```

Ich überprüfe, ob das Datum in der Vergangenheit liegt, falls schon, gibt es eine Fehlermeldung. Wenn das Datum in der Zukunft ist oder am gleichem Tag wird es funktionieren.

### Überprüfung der Übereinstimmung des Endprodukts mit funktionalen, nicht-funktionalen und sicherheitsrelevanten Anforderungen

Tesfallnummer	Datum	OK/NOK	Unterschrift
1	7/7/2023	OK	YK
2	7/7/2023	OK	YK
3	7/7/2023	OK	YK
4	7/7/2023	OK	YK
5	7/7/2023	OK	YK
6	7/7/2023	OK	YK

Alles funktioniert 😊

### Anwendung wichtiger Sicherheitsmaßnahmen im Umgang mit Eingabedaten

```
[HttpPost, Authorize]
0 references | karakushi, 13 minutes ago | 1 author, 2 changes
public async Task<IActionResult> AddTermin(AddTermin addTermin)
{
    if (string.IsNullOrEmpty(addTermin.Beschreibung) || addTermin.Beschreibung.Length < 5)
    {
        return BadRequest("Beschreibung zu kurz");
    }

    var termin = new Termin()
    {
        Id = Guid.NewGuid(),
        Datum = addTermin.Datum,
```

Ich validiere die Beschreibung bei einer Post-Anfrage. Wenn sie kleiner als 5 ist oder leer gibt es eine Fehlermeldung.

*Verwendung verschiedener Techniken zum Testen der Anforderungen (automatisch und von Hand)*

Testnr.	Automatisch/von Hand
1	von Hand
2	von Hand
3	von Hand
4	von Hand
5	von Hand
6	von Hand

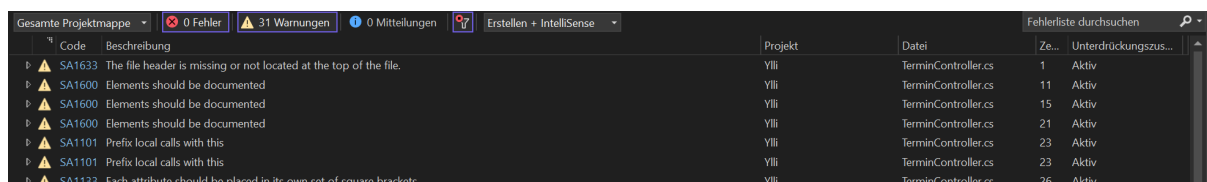
## Aufgabe 4

*Konfiguration der Entwicklungsumgebung für die Überprüfung von Coderichtlinien*  
StyleCop.Analyzers NuGet Packet installieren.

Es zeigt mehr Warnungen:



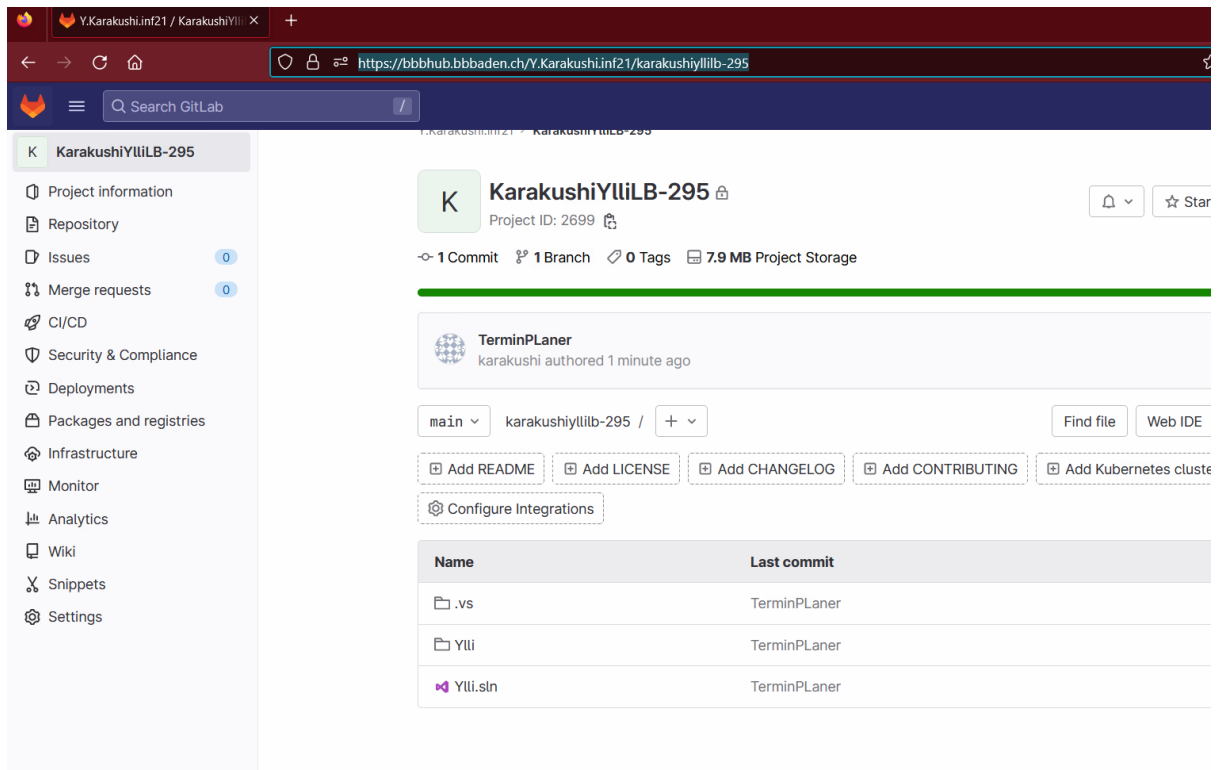
*Überprüfung der Einhaltung von Coderichtlinien und Korrektur gegebenenfalls*  
Nun sind es weniger:



## Aufgabe 5

*Effektive Verwaltung von Änderungen und Erweiterungen im Softwareverwaltungssystem*

<https://bbhub.bbbaden.ch/Y.Karakushi.inf21/karakushiylilb-295>



### *Bedienung des Softwareverwaltungssystems und Ablage von Änderungen und Erweiterungen*

Die Variable "origin" mit dem Befehl "git remote add origin 'GitLab-Link'" festlegen. Dadurch wird der Remote-Zweig "origin" auf GitLab verweisen.

Um bestimmte Änderungen zu übernehmen, verwende ich den Befehl "git add --all" und füge sie dem Index hinzu. Anschliessend committe ich die Änderungen mit einer Beschreibung, um sie für den Push vorzubereiten. Der Befehl lautet "git commit -m 'inhalt'".

Schliesslich sende ich die Änderungen an GitLab mit dem Befehl "git push origin main".

```
ylli@DESKTOP-M032157 MINGW64 ~/source/repos/Ylli
$ git init
Initialized empty Git repository in C:/Users/ylli_/source/repos/Ylli/.git/

ylli@DESKTOP-M032157 MINGW64 ~/source/repos/Ylli (master)
$ git remote add origin https://bbhub.bbbaden.ch/Y.Karakushi.inf21/karakushiylb-295

ylli@DESKTOP-M032157 MINGW64 ~/source/repos/Ylli (master)
$ git checkout -b main
Switched to a new branch 'main'
```

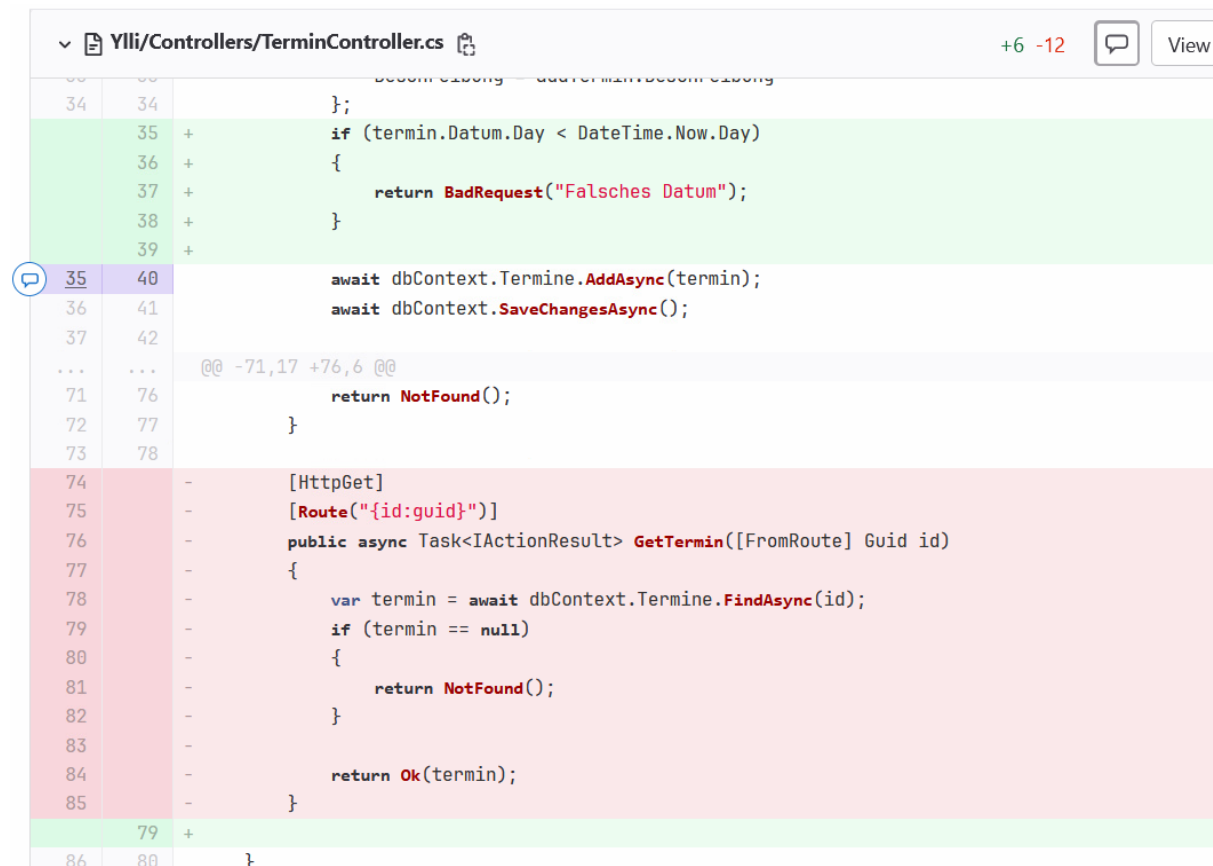
```
ylli@DESKTOP-M032157 MINGW64 ~/source/repos/Ylli (main)
$ git add --all

ylli@DESKTOP-M032157 MINGW64 ~/source/repos/Ylli (main)
$ git commit -m "TerminPLaner"
[main (root-commit) e3482a6] TerminPLaner
128 files changed, 7463 insertions(+)
create mode 100644 .vs/ProjectEvaluation/ylli.metadata.v6.1
create mode 100644 .vs/ProjectEvaluation/ylli.projects.v6.1
create mode 100644 .vs/Ylli/DesignTimeBuild/.dtbcache.v2
create mode 100644 .vs/Ylli/FileContentIndex/10c55523-a32c-4521-9aa4-3b6e.vsix
create mode 100644 .vs/Ylli/FileContentIndex/3c5ca9ea-24a6-4349-a7aa-7e
```

```
ylli@DESKTOP-M032157 MINGW64 ~/source/repos/Ylli (main)
$ git push origin main
warning: redirecting to https://bbhub.bbbaden.ch/Y.Karakushi.inf21/karakushiylb-295.git/
Enumerating objects: 164, done.
Counting objects: 100% (164/164), done.
Delta compression using up to 12 threads
Compressing objects: 100% (145/145), done.
Writing objects: 100% (164/164), 7.82 MiB | 5.86 MiB/s, done.
Total 164 (delta 12), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (12/12), done.
To https://bbhub.bbbaden.ch/Y.Karakushi.inf21/karakushiylb-295
* [new branch]      main -> main
```

## Nutzung des Softwareverwaltungssystems zur Behebung von Fehlern, Problemlösung und Verbesserung des Endprodukts

Es wurde eine Funktion gelöscht, diese habe ich mit GitLab wiederhergestellt.



```
Ylli/Controllers/TerminController.cs +6 -12 View
@@ -34,17 +35,6 @@
34 34 };
35 + if (termin.Datum.Day < DateTime.Now.Day)
36 + {
37 +     return BadRequest("Falsches Datum");
38 + }
39 +
35 40 await dbContext.Termine.AddAsync(termin);
36 41 await dbContext.SaveChangesAsync();
37 42
... @@ -71,17 +76,6 @@
71 76 return NotFound();
72 77 }
73 78
74 - [HttpGet]
75 - [Route("{id:guid}")]
76 - public async Task<ActionResult> GetTermin([FromRoute] Guid id)
77 - {
78 -     var termin = await dbContext.Termine.FindAsync(id);
79 -     if (termin == null)
80 -     {
81 -         return NotFound();
82 -     }
83 -
84 -     return Ok(termin);
85 - }
79 +
86 80 }
```

Hier sieht man was ich gelöscht habe und habe dies danach wieder eingefügt und push gemacht.

## Aufgabe 6

### *Implementierung eines aktuellen Authentifizierungsmechanismus im Back-End*

In diesem Codebeispiel habe ich einen AuthController erstellt, der die Endpunkte /register und /login für die Authentifizierung bereitstellt. Der AuthController erbt von der ControllerBase und ist mit dem [ApiController]-Attribut versehen, um die API-Funktionalitäten zu aktivieren.

Im Register-Endpunkt erfolgt die Registrierung eines Benutzers. Das übergebene BenutzerDto enthält Informationen wie den Benutzernamen und das Passwort. Das Passwort wird mit Hilfe von BCrypt.Net.BCrypt.HashPassword gehasht und zusammen mit dem Benutzernamen im benutzer-Objekt gespeichert. Das benutzer-Objekt wird dann als Erfolgsantwort zurückgegeben.

Im Login-Endpunkt wird der Benutzername und das Passwort überprüft. Falls der Benutzer nicht gefunden wird oder das Passwort falsch ist, wird eine entsprechende Fehlermeldung zurückgegeben. Andernfalls wird mit Hilfe der Methode CreateToken ein JWT-Token generiert und als Erfolgsantwort zurückgegeben.

Die Methode CreateToken generiert ein JWT-Token basierend auf dem Benutzerobjekt. Dabei werden Claims erstellt, die den Benutzernamen enthalten. Ein Schlüssel (SymmetricSecurityKey) wird aus der Konfigurationsdatei geladen und zur Signierung des Tokens verwendet. Das Token hat eine Gültigkeit von einem Tag (expires: DateTime.Now.AddDays(1)) und wird mit Hilfe der JwtSecurityTokenHandler-Klasse in einen JWT-String konvertiert. Dieser String wird zurückgegeben und kann vom Client zur Authentifizierung verwendet werden.

```

[Route("api/[controller]")]
[ApiController]
public class AuthController : ControllerBase
{
    public static Benutzer benutzer = new Benutzer();
    private readonly IConfiguration _configuration;

    public AuthController(IConfiguration configuration)
    {
        _configuration = configuration;
    }

    [HttpPost("register")]
    public ActionResult<Benutzer> Register(BenutzerDto request)
    {
        string passwordHash =
            BCrypt.Net.BCrypt.HashPassword(request.Passwort);
        benutzer.Benutzername = request.Benutzername;
        benutzer.PasswortHash = passwordHash;

        return Ok(benutzer);
    }

    [HttpPost("login")]
    public ActionResult<Benutzer> Login(BenutzerDto request)
    {
        if (benutzer.Benutzername != request.Benutzername)
        {
            return BadRequest("Benutzer nicht gefunden");
        }

        if (!BCrypt.Net.BCrypt.Verify(request.Passwort, benutzer.PasswortHash))
        {
            return BadRequest("Falsches Passwort");
        }

        string token = CreateToken(benutzer);

        return Ok(token);
    }

    private string CreateToken(Benutzer benutzer)
    {
        List<Claim> claims = new List<Claim>
        {
            new Claim(ClaimTypes.Name, benutzer.Benutzername)
        };
        var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(
            _configuration.GetSection("AppSettings:Token").Value!));
        var creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha512Signature);

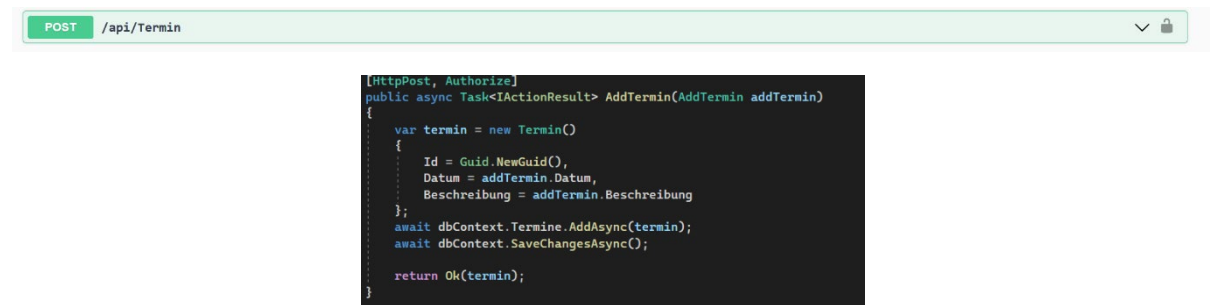
        var token = new JwtSecurityToken(
            claims: claims,
            expires: DateTime.Now.AddDays(1),
            signingCredentials: creds
        );
        var jwt = new JwtSecurityTokenHandler().WriteToken(token);
        return jwt;
    }
}

```

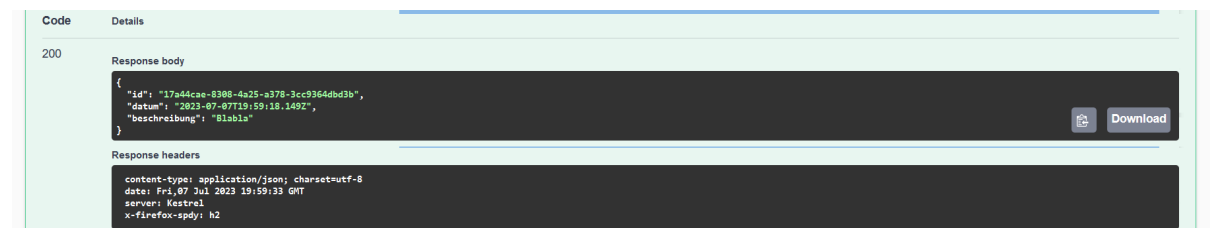


## Schutz vor anonymen Zugriffen in mindestens einem Bereich des Back-Ends

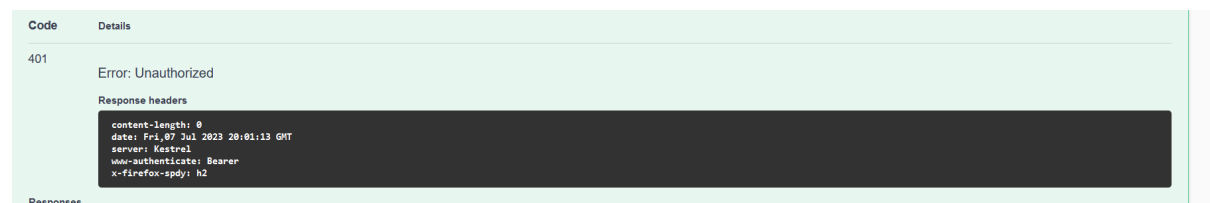
Um einen POST-Request durchführen zu können, ist eine vorherige Anmeldung erforderlich.



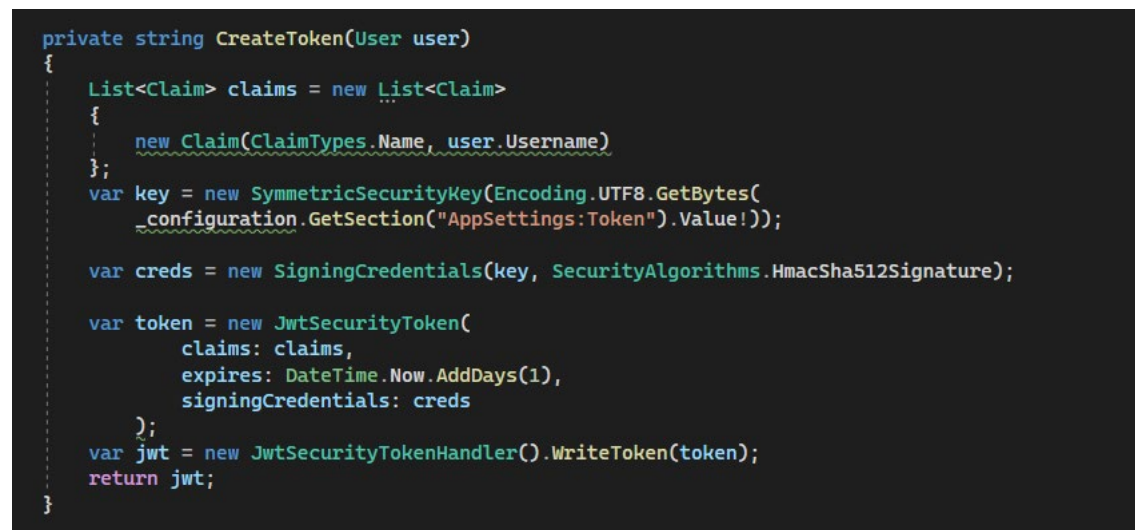
Wenn man eingeloggt ist hat man dieses Resultat:



Ohne login:



## Übereinstimmung des Authentifizierungsmechanismus im Back-End mit Anforderungen an Sicherheit und Datenschutz



Mein Code für den Authentifizierungsmechanismus im Backend stimmt mit den Anforderungen an Sicherheit und Datenschutz überein. Ich habe die Passwörter sicher gehasht, indem ich BCrypt verwendet habe, was eine bewährte Methode ist. Die Verwendung von JWTs zur Authentifizierung und Autorisierung ist ebenfalls eine gängige Praxis, und ich habe darauf geachtet, dass das Token

korrekt generiert und signiert wird. Ich habe auch die Vertraulichkeit der sensiblen Informationen berücksichtigt und mich bemüht, bewährte Sicherheitspraktiken zu implementieren.

Natürlich ist es wichtig zu beachten, dass die Sicherheit und der Datenschutz von vielen Faktoren abhängen und nicht nur vom Code selbst. Daher habe ich auch andere Aspekte wie die sichere Konfiguration des Servers und den Schutz der Daten während der Übertragung berücksichtigt. Ich werde den Code regelmässig überprüfen und sicherstellen, dass er den aktuellen Sicherheitsstandards entspricht, um die Sicherheit und den Schutz der Benutzerdaten zu gewährleisten.