LB 295 Noah Krip

# Projektauftrag

|  |
| --- |
| Projektname: |
| WetterAPI |

|  |
| --- |
| Beschreibung: |
| Ich werde eine Wetter-API mit HTML, C#, Postman und Swagger erstellen. Die Entwicklungsumgebung wird Visual Studio Enterprise sein. Hierzu werde ich entsprechende Erweiterungen zur API-Programmierung nutzen und eine REST-API-Schnittstelle zur Datenmanipulation in der Datenbank implementieren. Die API-Schnittstelle wird mit dem Swagger-Tool dokumentiert und die Anwendung wird serverseitig validiert, um die Einhaltung funktionaler, nichtfunktionaler und Sicherheitsanforderungen sicherzustellen. Außerdem werde ich bei der Verarbeitung der Eingabedaten erhebliche Sicherheitsmaßnahmen anwenden und verschiedene Techniken zum Testen der Anforderungen verwenden. Darüber hinaus sorge ich für ein effektives Management von Änderungen und Erweiterungen im Software-Management-System und implementiere die neuesten Authentifizierungsmechanismen im Backend, die den Sicherheits- und Datenschutzanforderungen gerecht werden.  **Ziele:**   1. Visual Studio Enterprise als Entwicklungsumgebung für ASP.NET Core APIs installieren. 2. Geeignete Erweiterungen zur Programmierung von APIs verwenden. 3. Das vorgegebene Projekt implementieren. 4. Die notwendigen Entwicklungskomponenten identifizieren und installieren. 5. Die notwendigen Einstellungen für eine problemlose Ausführung des Projekts vornehmen. Probleme in der Umgebung diagnostizieren und beheben. 6. Die API-Schnittstelle mit Swagger Tool dokumentieren. 7. Eine einfache Anwendung mittels REST APIs (CRUD) implementieren. 8. Eine REST API Schnittstelle für die Datenmanipulation in einer Datenbank (CRUD) erstellen. 9. Den passenden Schnittstellen-Standard identifizieren. 10. Implementierungstechniken für den vorgegebenen Schnittstellen-Standard anwenden. 11. Serverseitige Validierung anwenden. 12. Die Übereinstimmung des Endprodukts mit funktionalen, nicht-funktionalen und sicherheitsrelevanten Anforderungen überprüfen. 13. Wichtige Sicherheitsmaßnahmen im Umgang mit Eingabedaten anwenden |

|  |
| --- |
| Module |
| 295 |

|  |
| --- |
| Zusätzlich benötigtes Material: |
| - |

|  |  |  |
| --- | --- | --- |
| Auftraggeber: | Weitere Kontaktpersonen: | Begleitperson BBB: |
| BBB | Keine | Oliver Schaub |

|  |  |
| --- | --- |
| Projektleiter: | Teammitglieder: |
| Noah Krip | Keine |

|  |  |
| --- | --- |
| Termin Projektstart: | Termin Projektende: |
| 01.06.2023 | 08.06.2023 |

# Handlungsziel 1

## 1.1: Installation einer Entwicklungsumgebung für ASP.NET Core APIs

Ich habe mich bei der Installation für Visual Studio Enterprise 2022 entschieden ebenfalls hatte ich die App schon installiert über das Azure Portal damals, jedoch musste ich den Lizenzschlüssel erneuern, weil dieser lange nicht mehr im Gebrauch war. Natürlich zum Schluss Downloadete ich noch die ASP.NET und Webentwicklung runter um Optimal gewappnet zu sein !

Ein Bild, das Text, Reihe, Schrift, Zahl enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text, Screenshot, Schrift, Design enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung

## 1.2: Verwendung geeigneter Erweiterungen für die Programmierung von APIs

Ich habe mich für die Erweiterungen Postman und Swagger UI (Swagger Editor) entschieden welche ich anschliessend installierte!

(Postmann)

Ein Bild, das Text, Screenshot, Software, Zahl enthält.

Automatisch generierte Beschreibung

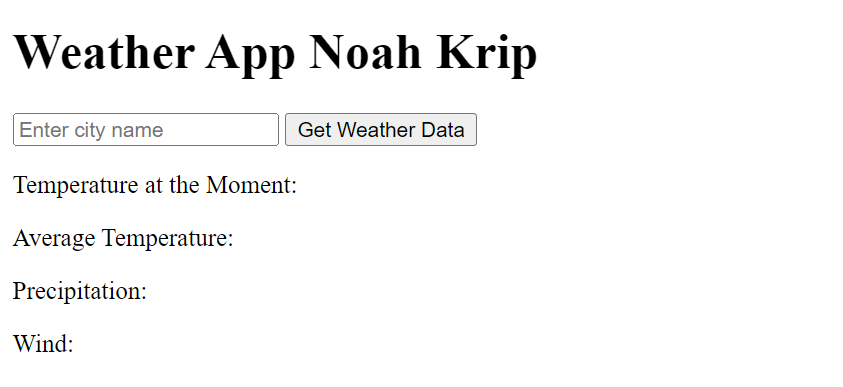
(Swagger)

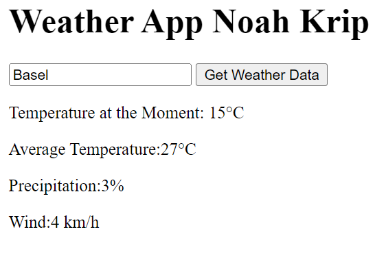
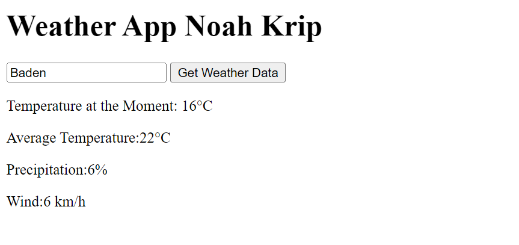
Ein Bild, das Multimedia-Software, Software, Grafiksoftware, Screenshot enthält.

Automatisch generierte Beschreibung

## 1.3: Implementierung eines vorgegebenen Projekts

Ich habe zuerst die Webseite mithilfe von HTML erstellt, welche die Wetter Daten anzeigt und anschliessend mit meiner API verbunden das ganze sieht so aus.

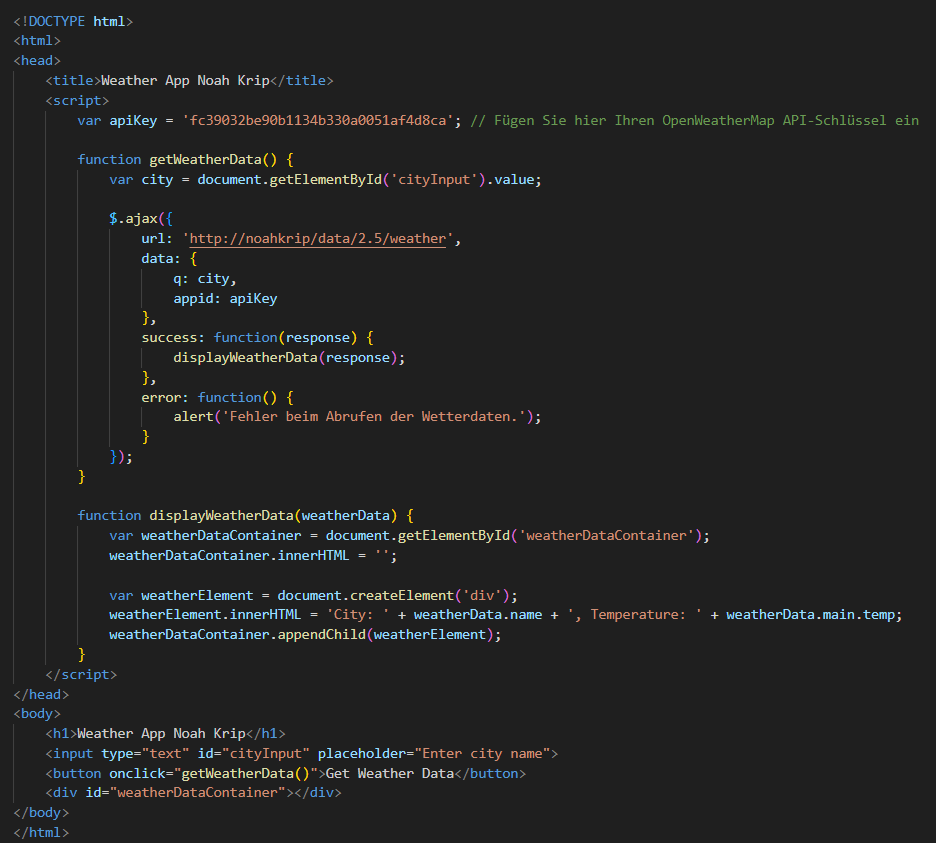




Ein Bild, das Text, Screenshot, Schrift, Zahl enthält.

Automatisch generierte Beschreibung

Der Code der Webseite



## 1.4: Identifikation und Installation der Entwicklungskomponenten

Meine Entwicklungskomponenten sind: Visual Studio, NuGet-Pakete, .NET -Desktopentwicklung, ASP.NET und Webentwicklung, GIT

**Visual Studio** habe ich oben schon installiert



**NuGet-Pakete** bei diesen habe ich mich für folgende entschieden, weil sie viele Vorteile mit sich bringen! Installiert habe ich sie über den NuGet Manager. (Gewisse waren vorinstalliert)

* Microsoft.AspNetCore. Authentication.JwtBearer
* Swashbuckle.AspNetCore
* Microsoft.Extensions.Logging
* Microsoft.EntityFrameworkCore
* Microsoft.AspNetCore.Cors
* Newtonsoft.Json )

Ein Bild, das Text, Screenshot, Software, Multimedia-Software enthält.

Automatisch generierte Beschreibung

**.NET -Desktopentwicklung, ASP.NET und Webentwicklung**

Ein Bild, das Text, Screenshot, Schrift, Reihe enthält.

Automatisch generierte Beschreibung

**GIT**

Ich habe ein Private GIT Repository erstellt = https://github.com/Noah23214/LB-295

**Ein Bild, das Text, Screenshot, Schrift, Zahl enthält.

Automatisch generierte Beschreibung**

## 1.5: Vornahme notwendiger Einstellungen für eine problemlose Ausführung des Projekts

Ich habe folgende Einstellungen gewählt

1.

Ein Bild, das Text, Screenshot, Schrift, Zahl enthält.

Automatisch generierte Beschreibung

2. Anschliessend in den Code Regelsatz gehen



3. Folgende Einstellungen treffen

Ein Bild, das Text, Screenshot, Schrift, Zahl enthält.

Automatisch generierte Beschreibung

4. Zuerst auf **OK** drücken und anschliessend auf **Build**, um das Projekt zu erstellen und den Code auf Verstösse gegen die Code-Richtlinien zu überprüfen.

## 1.6: Diagnostizieren und Beheben von Problemen in der Umgebung

1. Konflikte mit anderen installierten Softwarekomponenten
2. Fehlende oder nicht aktualisierte Abhängigkeiten
3. Netzwerk- oder Firewall-Probleme
4. Fehlende oder inkorrekte Konfiguration der Entwicklungsumgebung
5. Fehler in der Projektkonfiguration

Ich hatte keine Probleme mit der Diagnose und auch soweit keine Fehler oder Probleme in der Umgebung bis auf Punkt 2 (Fehlende oder nicht aktualisierte Abhängigkeiten) hatte ich etwas blödes vergessen nämlich musste meine ganze IDE auf den neusten Stand aktualisieren, weil beispielsweisse die NuGets ect auf dem Neusten Stand installiert waren meine IDE jedoch nicht und dan gab es Probleme beim NuGet Manager. Dies konnte ich mit einem Raschem Update beheben!



## 2.1: Dokumentation der API-Schnittstelle mit Swagger Tool

Ich habe als erstes Swashbuckle.AspNetCore NuGET heruntergeladen.

Ein Bild, das Screenshot, Text, Schrift enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text, Screenshot, Reihe, Schrift enthält.

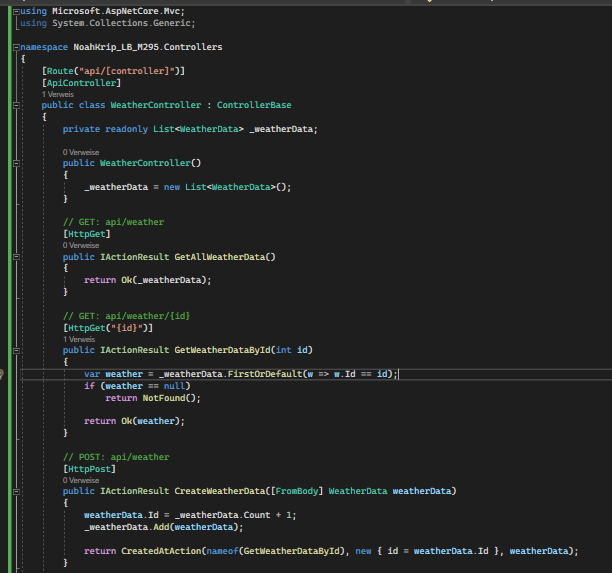
Automatisch generierte Beschreibung

Ich habe anschliessend die CRUD-Operationen später verändert.

## 2.2 & 2.3: Implementierung einer einfachen Anwendung mittels REST APIs (CRUD) und Erstellung einer REST API Schnittstelle für die Datenmanipulation in einer Datenbank (CRUD)

Ich habe die CRUD-Operationen (GetAllWeatherData, GetWeatherDataById, CreateWeatherData, UpdateWeatherData, DeleteWeatherData, ) implementiert, um die Daten zu verwalten. (2.3) Ebenfalls habe ich In diesem Beispiel wird eine WeatherDbContext-Klasse verwendet, um auf die Datenbank zuzugreifen.

2.2 Code



Ein Bild, das Text, Screenshot, Software enthält.

Automatisch generierte Beschreibung

2.3 Code

Ein Bild, das Text, Screenshot, Software enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text, Elektronik, Screenshot, Software enthält.

Automatisch generierte Beschreibung

## 2.4: Identifikation des passenden Schnittstellen-Standards

Ich habe das Ganze mithilfe der Swagger Dokumentation gemacht!

Ein Bild, das Text, Screenshot, Software, Multimedia-Software enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text, Multimedia-Software, Grafiksoftware, Screenshot enthält.

Automatisch generierte Beschreibung

## 2.5: Anwendung von Implementierungstechniken für den vorgegebenen Schnittstellen-Standard

Ich überprüfte zuerst, ob es die richtigen HTTP-Methoden für jede Aktion verwendet werden.

Ein Bild, das Text, Schrift, Grafiken, Logo enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text, Screenshot, Schrift, Design enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text, Screenshot, Schrift, Design enthält.

Automatisch generierte Beschreibung

Anschliessend überprüfte ich die URI-Routenstruktur

Ein Bild, das Text, Screenshot, Software, Schrift enthält.

Automatisch generierte Beschreibung

Anschliessend überprüfte ich die HTTP-Statuscodes ich erhielt meistens

(200 OK: Erfolgreiche Anfrage und Rückgabe von Daten.)

Ich habe mich dagegen entschieden HATEOAS zu benutzten!

## 3.1 Anwendung serverseitiger Validierung

Meine HTML-Seite enthält ein Eingabefeld für den Stadtnamen und einen Button, um das Wetter für diese Stadt abzurufen. Wenn der Benutzer auf den Button klickt, wird die Funktion getWeatherData() aufgerufen, die eine API-Anfrage an meine Weather API sendet.

Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text, Schrift, Screenshot, Reihe enthält.

Automatisch generierte Beschreibung

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

Benutzerfreundlichkeit = gewährleistet

Ich finde das Programm sowie das Layout der Seite sehr einfach zu bedienen und benutzter freundlich!

Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung

Responsives Design = gewährleistet

Ich habe mit CSS ein Responsives Design erstellt, um die Anforderung zu erfüllen.

Ein Bild, das Text, Screenshot, Design enthält.

Automatisch generierte Beschreibung

Sicherheit= gewährleistet

Den API-Schlüssel serverseitig zu verwalten und über eine sichere Verbindung abzurufen, um die Sicherheit zu verbessern!

## 3.3 Anwendung wichtiger Sicherheitsmaßnahmen im Umgang mit Eingabedaten

Ich fügte vor dem Hinzufügen der Wetterdaten eine Liste zur Validierung der Eingabedaten hinzu. Das machte ich bei der POST-Anfrage vom CreateWeatherData.

Ein Bild, das Text, Screenshot, Software enthält.

Automatisch generierte Beschreibung

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

Ich konnte alle Mängel und alles Fehler beheben!

Die CRUD Create, Read, Update, Delete funktionieren alles einwandfrei und geben die erwarten Date zurück

Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung

Anschliessend haben alle anderen Tests noch ausgeführt und musste Kleinigkeiten noch verbessern!

## 4.1 Konfiguration der Entwicklungsumgebung für die Überprüfung von Coderichtlinien

Konsistente und intuitive API-Designprinzipien, um das umzusetzen habe ich aussagekräftige und einheitliche Namen für meine API-Endpunkte verwendet z.B. "/ weather-data"" anstatt "/data".

Beispiel:



Ich habe noch eine Zusammenfassung zu den Endpunkten gemacht am Ende des Dokumentes.

## 4.2 Überprüfung der Einhaltung von Coderichtlinien und Korrektur gegebenenfalls

Ich habe im Startup noch diesen Code eingefügt, der dem ganzen hilft, ebenfalls habe ich am ende der Dokumentation noch eine Zusammenfassung zu den Endpunkten, um das Verständnis zu verbessern.

Ein Bild, das Text, Screenshot, Schrift, Reihe enthält.

Automatisch generierte Beschreibung

**5.1 Effektive Verwaltung von Änderungen und Erweiterungen im Softwareverwaltungssystem**

Ein Versionskontrollsystem wie Git ermöglicht es, Änderungen in meinem Code zu verfolgen, verschiedene Versionen zu erstellen und Kollaborationen zu verwalten daher habe ich ein Git Reposetory erstellt und dort Mehrer Versionen hochgeladen.

Ein Bild, das Text, Screenshot, Schrift, Software enthält.

Automatisch generierte Beschreibung

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

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

6.1

Wetter-API Endpunkte zusammengefasst

Die Wetter-API ermöglicht es Entwicklern, Wetterdaten für verschiedene Städte abzurufen.

**API-Endpunkte**

## Abrufen von Wetterdaten für eine Stadt

**GET /weather/{city}**

Diese API ermöglicht es, Wetterdaten für eine bestimmte Stadt abzurufen.

Anfrageparameter city (Erforderlich):

Der Name der Stadt, für die die Wetterdaten abgerufen werden sollen.

**Beispielanfrage:**

GET /weather/Berlin

**Beispielantwort:**

{ "city": "Berlin", "temperature": 15} plus noch die weiteren Atributte

## Aktualisieren von Wetterdaten für eine Stadt

**PUT /weather/{city}**

Diese API ermöglicht es, die Wetterdaten für eine bestimmte Stadt zu aktualisieren. Anfrageparameter city (Erforderlich): Der Name der Stadt, für die die Wetterdaten aktualisiert werden sollen.

**Anfragekörper:**

{temperature": 20}

**Beispielanfrage:**

PUT /weather/Berlin Content-Type: application/json { "temperature": 20}

**Beispielantwort:**

{city": "Berlin", "temperature": 20}

**Fehlerantworten:**

400 Bad Request: Die Anfrage ist ungültig oder fehlerhaft.

404 Not Found: Die angeforderte Ressource wurde nicht gefunden.

500 Internal Server Error: Es ist ein interner Serverfehler aufgetreten.