ENTWICKLUNG TODO-LISTEN-WEBANWENDUNG

Webanwendung mit ASP.NET Core und C#

Technichal Summary

Ziel der Anwendung:

- Die Anwendung wurde entwickelt, um Benutzern die Möglichkeit zu geben, Aufgaben zu verwalten und effizient zu organisieren. Das Hauptziel ist es, eine benutzerfreundliche Plattform bereitzustellen, auf der Aufgaben einfach hinzugefügt, angezeigt, bearbeitet und gelöscht werden können. Dabei wird besonderer Wert auf ein intuitives Benutzererlebnis gelegt, das durch eine klare Benutzeroberfläche und responsive Designelemente unterstützt wird.

Frontend:

- Das Frontend der Anwendung wurde mit HTML, CSS und JavaScript entwickelt, unter Verwendung des Bootstrap-Frameworks für ein ansprechendes Design und eine verbesserte Benutzererfahrung.

Backend:

- Für das Backend der Anwendung wurde C# verwendet, um die Logik zu implementieren, die die Benutzerinteraktionen verwaltet, Aufgaben speichert und abruft sowie die Verbindung zur Datenbank herstellt.

Datenbank:

- Als Datenbankmanagementsystem wurde Microsoft SQL Server (MSSQL) eingesetzt. MSSQL wird verwendet, um die Aufgaben persistent zu speichern und effizient abzurufen.

Funktionalitäten:

1. Aufgaben hinzufügen: Benutzer können neue Aufgaben eingeben und speichern.
2. Aufgaben anzeigen: Gespeicherte Aufgaben können angezeigt und nach Bedarf gelöscht werden.
3. Responsive Design: Das Frontend ist so gestaltet, dass es auf verschiedenen Geräten und Bildschirmgrößen gut funktioniert, dank Bootstrap.
4. Sicherheit: Geeignete Sicherheitsmaßnahmen wurden implementiert, um die Integrität der Daten zu gewährleisten und unautorisierten Zugriff zu verhindern.

Inhaltsverzeichnis

Projektauftrag

Ausgangslage:

Im Unternehmen besteht der Bedarf an einer einfachen und effektiven ToDo-Listen-Applikation, die es Benutzern ermöglicht, ihre Aufgaben zu organisieren und zu verwalten. Eine solche Applikation bietet eine praktische Möglichkeit, Aufgaben zu priorisieren, den Fortschritt zu verfolgen und die Produktivität zu steigern.

Ziel:

Das Ziel dieses Projekts ist die Entwicklung einer benutzerfreundlichen ToDo-Listen-Applikation, die auf verschiedenen Plattformen wie Web und Mobilgeräten genutzt werden kann. Die Applikation soll es den Benutzern ermöglichen, Aufgaben hinzuzufügen, zu bearbeiten, zu markieren und zu löschen. Zusätzlich sollen Funktionen wie die Priorisierung von Aufgaben, Erinnerungen und die Möglichkeit zur Kategorisierung integriert werden.

Aufwand:

Der geschätzte Aufwand für die Entwicklung dieses Projekts beträgt etwa 40 Stunden. Dies umfasst die Konzeption und Planung der Applikation, die Implementierung der ToDo-Liste in HTML, CSS und JavaScript, das Design der Benutzeroberfläche, die Integration der Datenbankfunktionalitäten zur Speicherung von Aufgaben sowie umfassende Tests, um die Stabilität und Benutzerfreundlichkeit sicherzustellen.

Geplantes Vorgehen:

- **Konzeption und Planung:** Erstellung eines detaillierten Konzepts für die ToDo-Listen-Applikation, Festlegung der Funktionalitäten und des Designs.
- **Entwicklung der Benutzeroberfläche:** Erstellung des HTML/CSS-Grundgerüsts für die ToDo-Liste und der entsprechenden Bedienelemente.
- **Implementierung der Logik:** Programmierung der JavaScript-Funktionen zur Erstellung und Verwaltung von Aufgaben, inklusive Hinzufügen, Bearbeiten, Löschen und Markieren von Aufgaben.
- **Datenbankintegration:** Einbindung einer MSSQL-Datenbank zur Speicherung von Aufgaben und Benutzereinstellungen.
- **Tests und Fehlerbehebung:** Durchführung umfassender Tests, um sicherzustellen, dass die ToDo-Listen-Applikation korrekt funktioniert und eine reibungslose Benutzererfahrung bietet.

Begründung:

Eine ToDo-Listen-Applikation ist ein unverzichtbares Werkzeug für die Organisation und Produktivität vieler Benutzer. Durch die Entwicklung dieser Applikation möchten wir unseren Mitarbeitern und Kunden eine praktische Lösung bieten, um ihre Aufgaben effektiv zu verwalten und den Arbeitsprozess zu optimieren. Dies trägt dazu bei, die Effizienz und Zufriedenheit der Benutzer zu steigern.

Projektplanung

- **Initialisierung**
  - Einrichtung eines neuen ASP.NET Core-Projekts in Visual Studio oder über die Befehlszeile.
  - Auswahl einer geeigneten Projektvorlage, z.B. leere Vorlage oder Web API-Vorlage.
- **Planung**
  - Erstellung von Projektordnern für das Backend und das Frontend, falls erforderlich.
  - Definition der Entitäten für die ToDo-Liste, z.B. Task-Modell.
  - Festlegung der Eigenschaften der Modelle, wie ID, Beschreibung, Fälligkeitsdatum usw.
- **Entscheiden**
  - Technologieauswahl:
    - ASP.NET Core für das Backend,
    - Entity Framework Core für die Datenzugriffsschicht,
    - HTML, CSS für den Frontend
- **Realisierung**
  - Implementierung von Controllern für CRUD-Operationen (Create, Read, Update, Delete) für die ToDo-Liste.
  - Konfiguration von Routen für die Controller-Methoden.
  - Einrichtung von Entity Framework Core für den Datenzugriff und die Datenbankverwaltung.
  - Erstellung der Benutzeroberfläche mit HTML und CSS
  - Implementierung von Formularen für das Hinzufügen, Bearbeiten und Löschen von Aufgaben.
- **Kontrolle**
  - Durchführung von Unit-Tests und Integrationstests, um die ordnungsgemäße Funktion der Anwendung sicherzustellen.
  - Überprüfung der Dokumentation und Tests, um sicherzustellen, dass alle Anforderungen erfüllt sind und die Anwendung bereit für den Einsatz ist.
- **Abgabe**
  - Erstellung von Dokumentationen für den Code, die Struktur und die Funktionalität der Anwendung.
  - Präsentation der abgeschlossenen Anwendung und der durchgeführten Tests.

Zeitplan

Schritt	Zeit geplant (in Lektionen)	Zeit effektiv (in Lektionen)
Einleitung	1	2
Technologieauswahl	2	2
Mockup	1	1
index.html	1	1
übersicht frontend	1	1
aufgaben frontend	1	1
übersicht js	1	2
aufgaben js	1	1
styles css	1	1
bootstraps implementierung	1	1
backend	2	3
CRUD funktionen	1	1
swagger implementation	1	1
testen	1	1
verbindung mit DB	1	2
verbindung mit Frontend	2	2
testing mit frontend	1	1
Dokumentation	10	15

Abbildung 1 Zeitplan

Arbeitsrapport

Einleitung:

- Einführung in das Projekt und Zielsetzung, Erstellung des Zeitplans

Technologieauswahl:

- Auswahl der Technologien für das Projekt: HTML, CSS, JavaScript, Bootstrap für das Frontend und C#, MSSQL für das Backend

Verbindung Front- und Backend:

- Festlegung der Schnittstellen zwischen Frontend und Backend
- Entwicklung der API zur Kommunikation zwischen beiden Schichten

Dokumentation:

- Erstellung der technischen Dokumentation zur Unterstützung der Entwicklung und Wartung
- Dokumentation der API-Endpunkte und deren Nutzung

Programmierschritte

Mockup:

- Erstellung eines Mockups für die Benutzeroberfläche
- Visualisierung der geplanten Funktionen und des Designs

Index:

- Entwicklung der Startseite (index.html) mit grundlegender Navigation

Übersicht Frontend:

- Implementierung der Hauptübersichtsseite für Aufgaben
- Design der Benutzeroberfläche für die Aufgabenliste

Aufgaben Frontend:

- Entwicklung der Seite zur detaillierten Ansicht und Bearbeitung einzelner Aufgaben

Übersicht JavaScript:

- Programmierung der JavaScript-Funktionen zur Darstellung und Verwaltung der Aufgabenübersicht

Aufgaben JavaScript:

- Implementierung der JavaScript-Logik zur Bearbeitung, Markierung und Priorisierung von Aufgaben

Styles CSS:

- Erstellung und Anwendung der CSS-Stile für ein einheitliches und ansprechendes Design
- Integration von Bootstrap zur Unterstützung responsiven Designs

Bootstrap Implementierung:

- Einbindung von Bootstrap-Komponenten zur Verbesserung der Benutzerfreundlichkeit und des Layouts

Backend:

- Entwicklung der Server-seitigen Logik mit C#
- Implementierung der CRUD-Funktionen (Create, Read, Update, Delete) für Aufgaben

Swagger Implementierung:

- Integration von Swagger zur Dokumentation und Testen der API-Endpunkte

Testen:

- Durchführung umfassender Tests zur Sicherstellung der Funktionalität und Stabilität
- Testen der einzelnen Komponenten und deren Zusammenspiel

Verbindung mit Datenbank:

- Einrichtung und Konfiguration der MSSQL-Datenbank
- Implementierung der Datenbankverbindungen und -abfragen

Verbindung mit Frontend:

- Integration der Frontend-Oberfläche mit den Backend-APIs
- Sicherstellung der reibungslosen Kommunikation zwischen Frontend und Backend

Testing mit Frontend:

- End-to-End-Tests zur Überprüfung der vollständigen Anwendung
- Behebung von Fehlern und Optimierung der Benutzererfahrung

Arbeitszeit und Aufwand:

Geschätzter Aufwand: 40 Stunden

Tatsächlicher Aufwand: [Anzahl der geleisteten Stunden]

Bemerkungen:

Unterschätzung der Verbindung mit dem Frontend, Probleme mit den Ports

Davud Ponjevic

Genehmigt durch:

Marco Frei

Datum der Genehmigung:

12. Mai 2024

Systemdokumentation

Hier erwähne ich nur die wichtigsten Zeilen, welchen dieses Projekt auszeichnet

Backend

Organisation

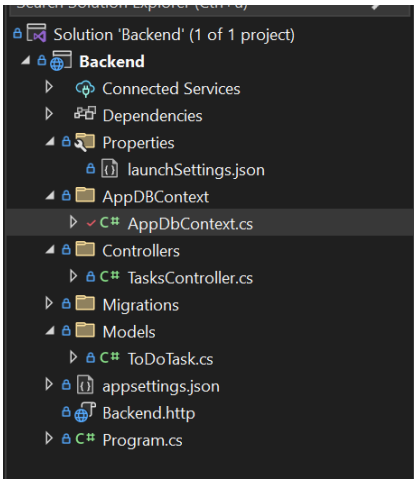


Abbildung 2 Ordnung meiner SLN

Hier sehen wir wie meine SLN (Solution = Ordner des Backend). Ich erwähne nur die wichtigsten und die Dateien, welche dieses Projekt auszeichnen.

Data/AppDbContext.cs

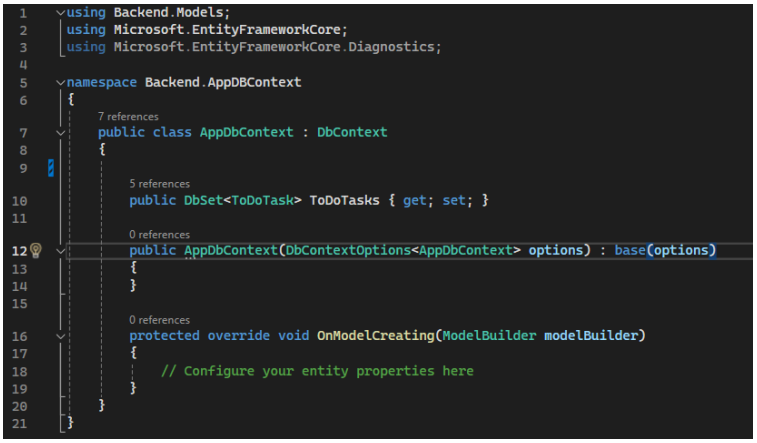
Beschreibung:

Diese Datei definiert den Datenbankkontext für die Anwendung, der mit Entity Framework Core arbeitet. Sie verwaltet die Verbindung zur Datenbank und konfiguriert die Zuordnung der Modellklassen zu den Datenbanktabellen.

Wichtig:

- Verwaltet die Datenbankverbindung.
- Definiert DbSet-Eigenschaften für Modellklassen.
- Konfiguriert die Datenbank.

Code Erklärung:



```
using Backend.Models;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Diagnostics;

namespace Backend.AppDBContext
{
    7 references
    public class AppDbContext : DbContext
    {

        5 references
        public DbSet<ToDoTask> ToDoTasks { get; set; }

        0 references
        public AppDbContext(DbContextOptions<AppDbContext> options) : base(options)
        {
        }

        0 references
        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            // Configure your entity properties here
        }
    }
}
```

Diese Datei definiert den Datenbankkontext der Anwendung und konfiguriert die Verbindung zur Datenbank. Sie enthält eine DbSet-Eigenschaft für ToDoTask.cs, die die ToDoTasks-Tabelle in der Datenbank repräsentiert.

Abbildung 3 AppDbContext.cs

Controllers/TasksController.cs

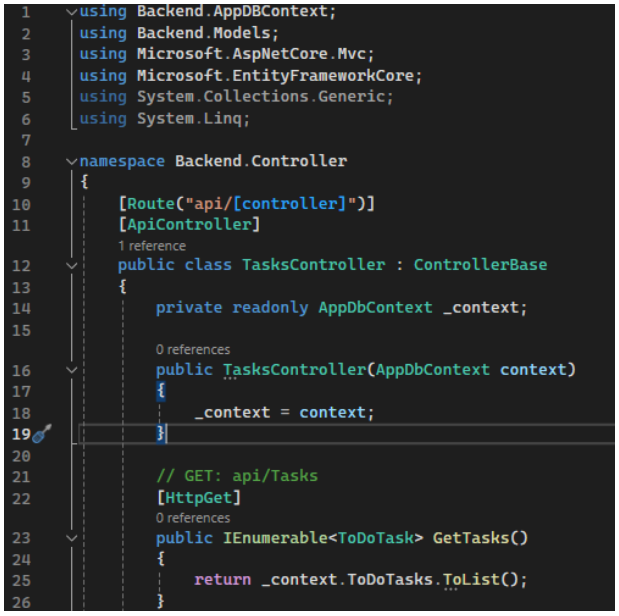
Beschreibung:

Diese Datei enthält den Controller, der die API-Endpunkte für die Aufgabenverwaltung bereitstellt. Sie implementiert die CRUD-Operationen (Create, Read, Update, Delete) und steuert die Logik für die Bearbeitung von HTTP-Anfragen.

Wichtig:

- Definiert API-Routen.
- Implementiert Methoden für CRUD-Operationen.
- Verwaltet die Kommunikation zwischen Frontend und Backend.

Code Erklärung:



```
using Backend.AppDBContext;
using Backend.Models;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using System.Collections.Generic;
using System.Linq;

namespace Backend.Controller
{
    [Route("api/[controller]")]
    [ApiController]
    1 reference
    public class TasksController : ControllerBase
    {
        private readonly AppDbContext _context;

        0 references
        public TasksController(AppDbContext context)
        {
            _context = context;
        }

        // GET: api/Tasks
        [HttpGet]
        0 references
        public IEnumerable<ToDoTask> GetTasks()
        {
            return _context.ToDoTasks.ToList();
        }
```

Diese Datei enthält den Controller, der die API-Endpunkte für die Aufgabenverwaltung bereitstellt. Sie implementiert Methoden zum Erstellen, Abrufen, Aktualisieren und Löschen von Aufgaben(nicht Im Bild).

Abbildung 4 TasksController.cs

Models/ToDoTask.cs

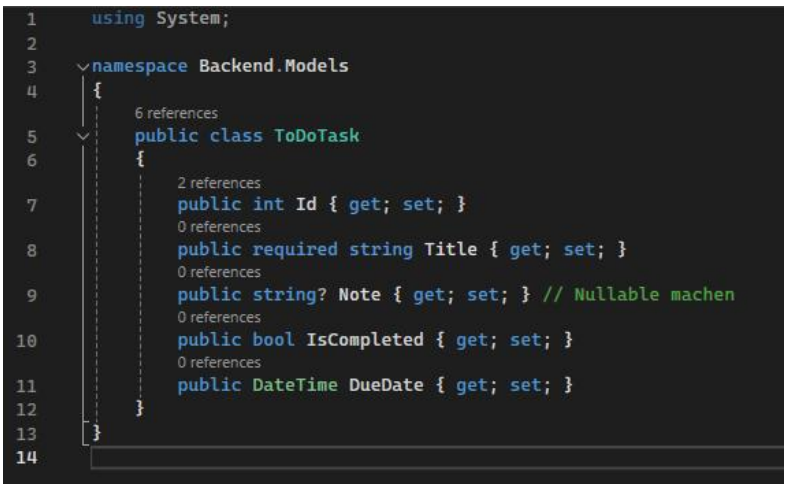
Beschreibung:

Diese Datei enthält die Modellklasse, die die Struktur und Eigenschaften einer Aufgabe definiert. Diese Klasse wird verwendet, um die Daten in der Datenbank zu repräsentieren und zwischen Datenbank und Anwendung zu übertragen.

Wichtig:

- Definiert die Eigenschaften einer Aufgabe.
- Enthält Datenannotations zur Validierung.
- Repräsentiert eine Datenbanktabelle.

Code Erklärung:



```
1   using System;
2
3   namespace Backend.Models
4   {
        6 references
5       public class ToDoTask
6       {
            2 references
7           public int Id { get; set; }
            0 references
8           public required string Title { get; set; }
            0 references
9           public string? Note { get; set; } // Nullable machen
            0 references
10          public bool IsCompleted { get; set; }
            0 references
11          public DateTime DueDate { get; set; }
12      }
13  }
14
```

Diese Datei definiert die Modellklasse `ToDoTask`, die die Struktur und Eigenschaften einer Aufgabe wie ID, Titel, Notiz, Abschlussstatus und Fälligkeitsdatum enthält.

Abbildung 5 Model/ToDoTask.cs

appsettings.json

Beschreibung:

Diese Datei enthält Konfigurationsinformationen für die Anwendung. Sie wird genutzt, um Einstellungen wie Datenbankverbindungen, Logging-Konfigurationen und andere anwendungsspezifische Parameter zu speichern.

Wichtig:

- Speichert Verbindungszeichenfolgen zur Datenbank.
- Enthält Konfigurationen für verschiedene Umgebungen.
- Wird zur Laufzeit geladen und von der Anwendung verwendet.

Code Erklärung:

```
{
  "ConnectionStrings": {
    "DefaultConnection": "Server=localhost;Database=SQLProjekt;Trusted_Connection=True;TrustServerCertificate=True;MultipleActiveResultSets=true"
  },
  "AllowedHosts": "*",
  "CORS": {
    "AllowedOrigins": "http://127.0.0.1:5500"
  }
```

Abbildung 6 appsettings.json

Diese Datei enthält Konfigurationsinformationen für die Anwendung, einschließlich der Datenbankverbindungszeichenfolge und CORS-Einstellungen.

Program.cs

Beschreibung:

Diese Datei ist der Einstiegspunkt der Anwendung. Sie enthält die Main-Methode, die die Anwendung startet und den Webhost konfiguriert.

Wichtig:

Startet die Anwendung.

- Konfiguriert den Webhost.
- Registriert Dienste und Middleware-Komponenten.

Code Erklärung:

```
namespace Backend
{
    0 references
    public class Program
    {
        0 references
        public static void Main(string[] args)
        {
            CreateHostBuilder(args).Build().Run();
        }

        1 reference
        public static IHostBuilder CreateHostBuilder(string[] args) =>
            Host.CreateDefaultBuilder(args)
                .ConfigureWebHostDefaults(webBuilder =>
                {
                    webBuilder.ConfigureServices((hostContext, services) =>
                    {
                        services.AddControllers();

                        // Swagger-Dienste hinzufügen
                        services.AddSwaggerGen(c =>
                        {
                            c.SwaggerDoc("v1", new OpenApiInfo { Title = "Backend", Version = "v1" });
                        });

                        // EndpointsApiExplorer-Dienste hinzufügen
                        services.AddEndpointsApiExplorer();

                        // Konfiguration laden
                        var configuration = new ConfigurationBuilder()
                            .SetBasePath(AppDomain.CurrentDomain.BaseDirectory)
                            .AddJsonFile("appsettings.json")
                            .Build();

                        // Verbindungszeichenfolge aus der Konfiguration lesen
                        var connectionString = configuration.GetConnectionString("DefaultConnection");

                        // AppDbContext mit Verbindungszeichenfolge registrieren
                        services.AddDbContext<AppDbContext>(options =>
                        {
```

Diese Datei ist der Einstiegspunkt der Anwendung. Sie konfiguriert den Webhost, registriert Dienste wie DbContext und CORS, und richtet die Middleware-Pipeline für Routing, Autorisierung und Swagger ein.

Abbildung 7 Program.cs

Frontend

index.html

Beschreibung:

Die Startseite der Anwendung, die eine Begrüßung und Navigation bietet.

Wichtigkeit:

Dies ist die Einstiegsseite der Anwendung und bietet den Benutzern eine Übersicht und Links zu den Hauptfunktionen.

Code-Erklärung:

- Die grundlegende HTML-Dokumentstruktur ist definiert.
- Bootstrap und eigene CSS- und JavaScript-Dateien werden eingebunden.
- Der Header enthält eine Navigation mit Links zu verschiedenen Seiten der Anwendung.
- Der Hauptteil fordert die Benutzer auf, neue Aufgaben hinzuzufügen.

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Task Management</title>
    <link href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css" rel="stylesheet">
    <link rel="stylesheet" href="styles.css">
    <script src="script.js"></script>
</head>
<body>
    <header class="bg-primary text-white text-center py-3">
        <h1>Willkommen bei unserer To-DO Web-App</h1>
        <nav class="navbar navbar-expand-lg navbar-dark bg-dark">
            <ul class="navbar-nav mr-auto">
                <li class="nav-item"><a class="nav-link" href="tasks.html">Neue Aufgabe hinzufügen</a></li>
                <li class="nav-item"><a class="nav-link" href="posts.html">Übersicht</a></li>
            </ul>
        </nav>
    </header>
    <main class="container mt-4">
        <section class="mb-4">
            <h2>Haben sie in kürze einen Aufgabe welche sie erledigen müssen</h2>
            <p>Hinterlegen sie diese Aufgabe jetzt um es sicherich nicht zu vergessen</p>
            <a href="tasks.html" class="btn btn-primary">Aufgaben</a>
        </section>

    </main>
    <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>
    <script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.9.2/dist/umd/popper.min.js"></script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>
</body>
</html>
```

Abbildung 8 index.html

posts.html

Beschreibung:

Seite zur Übersicht der vorhandenen Aufgaben.

Wichtigkeit:

Ermöglicht es den Benutzern, eine Liste ihrer Aufgaben zu sehen und zu verwalten.

Code-Erklärung:

- Ähnlich wie index.html werden Bootstrap und eigene Dateien eingebunden.
- Der Header enthält die Navigation.
- Der Hauptteil zeigt eine Liste der Aufgaben an, die dynamisch durch JavaScript erstellt wird.

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Aufgaben Hinzufügen</title>
    <link href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css" rel="stylesheet">
    <link rel="stylesheet" href="styles.css">
    <script src="posts_fetch.js"></script>
</head>
<body>
    <header class="bg-primary text-white text-center py-3">
        <h1>Übersicht</h1>
        <nav class="navbar navbar-expand-lg navbar-dark bg-dark">
            <ul class="navbar-nav mr-auto">
                <li class="nav-item"><a class="nav-link" href="index.html">Home Seite</a></li>
                <li class="nav-item"><a class="nav-link" href="tasks.html">Neue Aufgabe hinzufügen</a></li>
            </ul>
        </nav>
    </header>
    <main class="container mt-4">
        <section id="tasksSection" class="mb-4">
            <h2>Was ich zu tun habe</h2>
            <div id="tasks">
                <!-- Hier werden die Aufgaben angezeigt -->
            </div>
        </section>

    </main>
    <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>
    <script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.9.2/dist/umd/popper.min.js"></script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>
</body>
</html>
```

Abbildung 9 posts.html

tasks.html

Beschreibung:

Seite zum Hinzufügen neuer Aufgaben.

Wichtigkeit:

Ermöglicht es den Benutzern, neue Aufgaben hinzuzufügen.

Code-Erklärung:

- Bootstrap und eigene Dateien werden eingebunden.
- Der Header enthält die Navigation.
- Ein Formular im Hauptteil ermöglicht es den Benutzern, neue Aufgaben mit Titel, Notiz und Fälligkeitsdatum einzugeben.

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Task Management</title>
    <link href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css" rel="stylesheet">
    <link rel="stylesheet" href="styles.css">
    <script src="Task_fetch.js"></script>
</head>
<body>
    <header class="bg-primary text-white text-center py-3">
        <h1>Aufgaben Hinzufügen</h1>
        <nav class="navbar navbar-expand-lg navbar-dark bg-dark">
            <ul class="navbar-nav mr-auto">
                <li class="nav-item"><a class="nav-link" href="index.html">Home Seite</a></li>
                <li class="nav-item"><a class="nav-link" href="posts.html">Übersicht</a></li>
            </ul>
        </nav>
    </header>

    <main class="container mt-4">
        <form id="taskForm" class="mb-4">
            <h2>Add Task</h2>
            <div class="form-group">
                <label for="taskTitle">Title:</label>
                <input type="text" class="form-control" id="taskTitle" name="title" required>
            </div>
            <div class="form-group">
                <label for="taskNote">Note:</label>
                <textarea class="form-control" id="taskNote" name="note"></textarea>
            </div>
            <div class="form-group">
                <label for="taskDueDate">Due Date:</label>
                <input type="date" class="form-control" id="taskDueDate" name="dueDate" required>
            </div>
            <button type="submit" class="btn btn-primary">Add Task</button>
        </form>
```

Abbildung 10 tasks.html

styles.css

Beschreibung:

CSS-Datei für benutzerdefinierte Stile der Anwendung.

Wichtigkeit:

Verbessert das visuelle Erscheinungsbild der Anwendung und passt das Design an.

Code-Erklärung:

- Enthält allgemeine Stile für das Layout und die Struktur der Seite.
- Stile für Header, Footer, Navigation, Buttons und Aufgaben.

```css
body {
    font-family: Arial, sans-serif;
    margin: 0;
    padding: 20px;
}

h1 {
    text-align: center;
}

.task, .event {
    border: 1px solid #ccc;
    margin-bottom: 10px;
    padding: 10px;
}
header, footer {
    background-color: rgba(0, 0, 0, 0.7);
    color: #fff;
    padding: 10px;
}
nav ul {
    list-style-type: none;
    padding: 0;
    text-align: center;
}
a {
    color: #fff;
    text-decoration: none;
    padding: 8px 16px;
    border-radius: 5px;
    transition: background-color 0.3s ease;
}
.button {
    background-color: #000;
    color: #fff;
    padding: 10px 20px;
    text-align: center;
    text-decoration: none;
    display: inline-block;
    font-size: 16px;
    margin-top: 10px;
```

Abbildung 11 styles.css

post_fetch.js

Beschreibung:

JavaScript-Datei zum Abrufen und Verwalten der Aufgaben der API.

Wichtigkeit:

Ermöglichten die dynamische Anzeige und Verwaltung von Aufgaben auf der posts.html Seite.

Code-Erklärung:

- Lädt Aufgaben von der API und zeigt sie in der HTML-Seite an.
- Enthält Funktionen zum Erstellen von HTML-Elementen für Aufgaben.
- Beinhaltet Event-Listener für das Hinzufügen und Löschen von Aufgaben.

```
document.addEventListener('DOMContentLoaded', () => {
    fetchTasks();

    // Event-Listener für das Löschen von Aufgaben
    document.getElementById('tasks').addEventListener('click', handleTaskDelete);

    // Event-Listener für das Hinzufügen einer neuen Aufgabe
    document.getElementById('addTaskForm').addEventListener('submit', handleTaskFormSubmit);
});

function fetchTasks() {
    fetch('https://localhost:7230/api/Tasks')
        .then(response => {
            if (!response.ok) {
                throw new Error('Fehler beim Abrufen der Aufgaben');
            }
            return response.json();
        })
        .then(tasks => {
            const tasksContainer = document.getElementById('tasks');
            tasksContainer.innerHTML = '';

            tasks.forEach(task => {
                const taskElement = createTaskElement(task);
                tasksContainer.appendChild(taskElement);
            });
        })
        .catch(error => console.error('Fehler beim Abrufen der Aufgaben:', error));
}

function createTaskElement(task) {
    const taskElement = document.createElement('div');
    taskElement.classList.add('card', 'mb-3');
    taskElement.innerHTML = `
        <div class="card-body">
            <h5 class="card-title">${task.title}</h5>
            <p class="card-text">${task.note}</p>
            <p class="card-text"><small class="text-muted">Fällig am: ${task.dueDate}</small></p>
            <button class="btn btn-danger btn-sm delete-task" data-id="${task.id}">Löschen Aufgabe</button>
        </div>
```

function createTaskElement(task: any): HTMLDivElement

Abbildung 12 posts_fetch.js

task_fetch.js

Beschreibung:

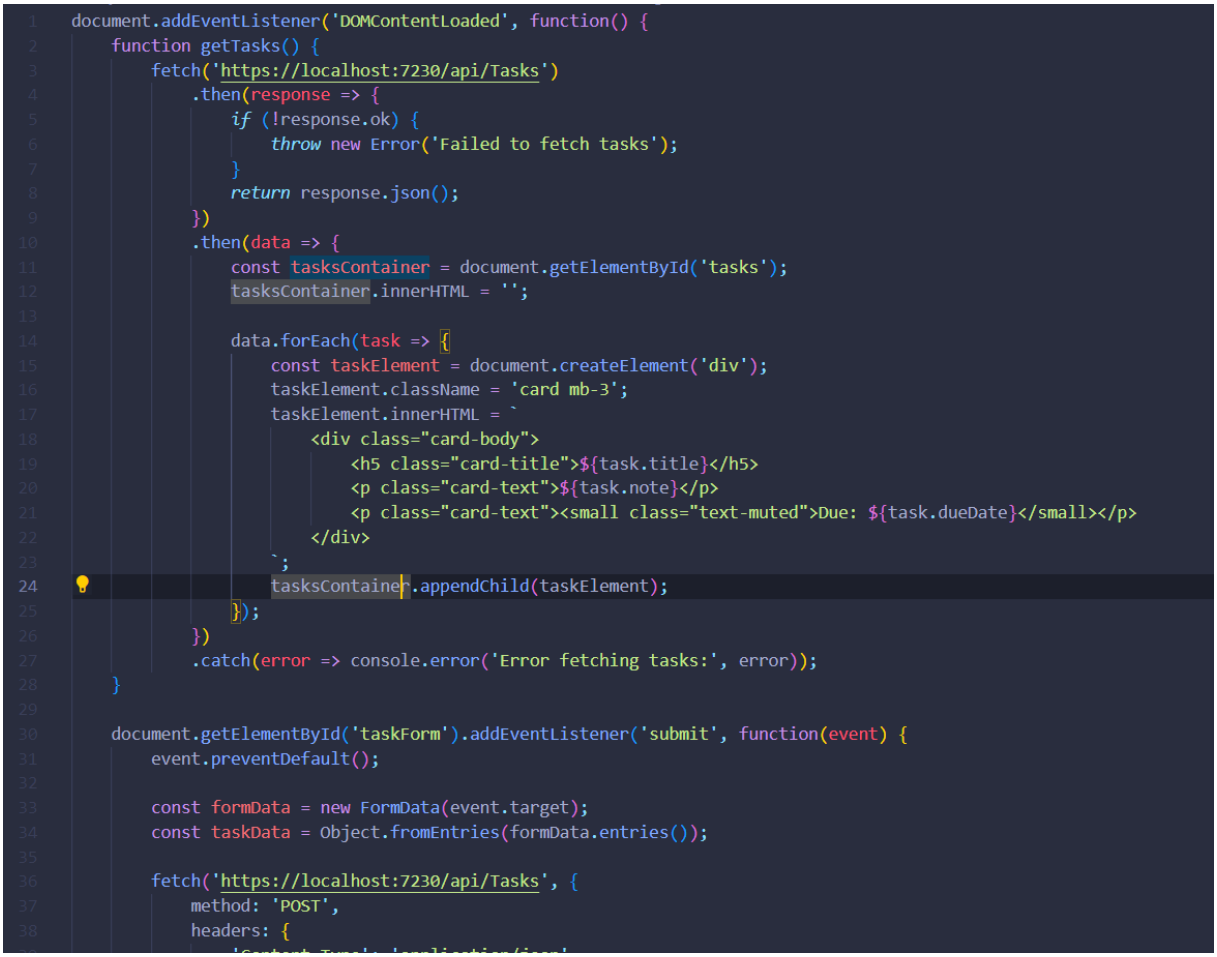
JavaScript-Datei zum Hinzufügen von Aufgaben zur API und Abrufen der aktuellen Aufgaben.

Wichtigkeit:

Ermöglicht es den Benutzern, neue Aufgaben hinzuzufügen und die aktuelle Liste der Aufgaben zu sehen.

Code-Erklärung:

- Lädt Aufgaben von der API und zeigt sie in der HTML-Seite an.
- Enthält Funktionen zum Hinzufügen neuer Aufgaben durch ein Formular.
- Beinhaltet Event-Listener für das Formular zum Hinzufügen von Aufgaben.



```
document.addEventListener('DOMContentLoaded', function() {
    function getTasks() {
        fetch('https://localhost:7230/api/Tasks')
            .then(response => {
                if (!response.ok) {
                    throw new Error('Failed to fetch tasks');
                }
                return response.json();
            })
            .then(data => {
                const tasksContainer = document.getElementById('tasks');
                tasksContainer.innerHTML = '';

                data.forEach(task => {
                    const taskElement = document.createElement('div');
                    taskElement.className = 'card mb-3';
                    taskElement.innerHTML = `
                        <div class="card-body">
                            <h5 class="card-title">${task.title}</h5>
                            <p class="card-text">${task.note}</p>
                            <p class="card-text"><small class="text-muted">Due: ${task.dueDate}</small></p>
                        </div>
                    `;
                    tasksContainer.appendChild(taskElement);
                });
            })
            .catch(error => console.error('Error fetching tasks:', error));
    }

    document.getElementById('taskForm').addEventListener('submit', function(event) {
        event.preventDefault();

        const formData = new FormData(event.target);
        const taskData = Object.fromEntries(formData.entries());

        fetch('https://localhost:7230/api/Tasks', {
            method: 'POST',
            headers: {
```

Abbildung 13 tasks.html

Planung

Beschreibung: Die Planungsphase eines Softwareprojekts ist der erste Schritt im Entwicklungszyklus und beinhaltet die Festlegung der Projektziele, die Identifikation der Anforderungen und die Erstellung eines detaillierten Projektplans.

Wichtigkeit: Eine gründliche Planung ist entscheidend, um sicherzustellen, dass alle Anforderungen und Erwartungen klar definiert sind. Dies reduziert das Risiko von Missverständnissen und Problemen während der späteren Phasen des Projekts.

Aktivitäten:

- **Anforderungsanalyse:** Identifizierung und Dokumentation der funktionalen und nicht-funktionalen Anforderungen.
- **Machbarkeitsstudie:** Bewertung der technischen, wirtschaftlichen und zeitlichen Machbarkeit des Projekts.
- **Projektplanung:** Erstellung eines Projektzeitplans, Festlegung von Meilensteinen und Zuweisung von Ressourcen.
- **Risikoanalyse:** Identifikation potenzieller Risiken und Entwicklung von Strategien zur Risikominderung.

Spezifikation

Beschreibung: Die Spezifikationsphase, auch als Anforderungsdefinition bekannt, ist der Prozess der detaillierten Dokumentation aller Anforderungen und Spezifikationen, die das System erfüllen muss.

Wichtigkeit: Eine klare und präzise Spezifikation ist die Grundlage für die Entwicklung und dient als Referenzpunkt für alle nachfolgenden Phasen. Sie stellt sicher, dass alle Beteiligten eine gemeinsame Vorstellung vom Endprodukt haben.

Aktivitäten:

- **Erstellung des Lastenhefts:** Dokumentation der Anforderungen aus Sicht des Auftraggebers.
- **Erstellung des Pflichtenhefts:** Detaillierte Beschreibung der technischen Umsetzung und Spezifikationen aus Sicht des Auftragnehmers.
- **Modellierung:** Einsatz von UML-Diagrammen und anderen Modellierungstechniken zur visuellen Darstellung der Systemarchitektur und -prozesse.
- **Review und Validierung:** Überprüfung der Spezifikationen durch alle relevanten Stakeholder und Validierung der Anforderungen.

Bau

Beschreibung: Die Bauphase, auch als Entwicklungsphase bekannt, umfasst die eigentliche Programmierung und Implementierung des Systems gemäß den in der Spezifikationsphase festgelegten Anforderungen.

Wichtigkeit: In dieser Phase wird die theoretische Planung in ein funktionierendes System umgesetzt. Effizienz und Qualität in der Entwicklung sind entscheidend für den Erfolg des Projekts.

Aktivitäten:

- **Architekturdesign:** Erstellung der Softwarearchitektur und des Designs.
- **Implementierung:** Programmierung der einzelnen Softwarekomponenten und Integration in das Gesamtsystem.
- **Testen:** Durchführung von Unit-Tests, Integrationstests und Systemtests, um sicherzustellen, dass das System die Anforderungen erfüllt und frei von Fehlern ist.
- **Dokumentation:** Erstellung der technischen Dokumentation und Benutzerhandbücher.
- **Deployment:** Bereitstellung des Systems in der produktiven Umgebung

Testkonzept

Einführung

Das Testkonzept beschreibt die geplanten Testaktivitäten und -verfahren für ein Softwareprojekt. Ziel ist es, die Qualität des Produkts zu gewährleisten, Fehler zu identifizieren und sicherzustellen, dass die Software den definierten Anforderungen entspricht.

Testziele

- Sicherstellen, dass alle funktionalen und nicht-funktionalen Anforderungen erfüllt sind.
- Identifizieren und Beheben von Fehlern und Mängeln.
- Überprüfung der Systemleistung, Zuverlässigkeit und Sicherheit.
- Sicherstellen der Benutzerfreundlichkeit und Kompatibilität.

Teststrategie

Die Teststrategie legt fest, welche Testarten und -methoden angewendet werden, um die definierten Testziele zu erreichen.

Teststufen:

1. **Unit-Tests:** Testen einzelner Komponenten oder Module auf Funktionalität.
2. **Integrationstests:** Überprüfung des Zusammenspiels zwischen dem Frontend und dem Backend.
3. **Abnahmetests:** Tests durch den Endanwender zur Validierung der Benutzerfreundlichkeit.

Testumgebung

Die Testumgebung umfasst die Hardware, Software, Netzwerkkonfigurationen und sonstigen Tools, die für die Durchführung der Tests erforderlich sind.

Testdaten

- **Testdatenbanken:** Realistische Daten, die verschiedene Szenarien abdecken.
- **Anonymisierte echte Daten:** Zur Sicherstellung der Datenschutzbestimmungen.

Testrollen und Verantwortlichkeiten

- **Testmanager:** Verantwortlich für die Planung und Koordination aller Testaktivitäten.
- **Tester:** Durchführung der Tests und Dokumentation der Ergebnisse.
- **Entwickler:** Unterstützung bei Unit-Tests und Fehlerbehebung.
- **Fachanwender:** Durchführung der Abnahmetests und Bereitstellung von Feedback.

Testprozess

1. **Testplanung:** Erstellung des Testkonzepts und detaillierter Testpläne.
2. **Testvorbereitung:** Einrichtung der Testumgebung, Erstellung der Testdaten und Entwicklung der Testfälle.
3. **Testdurchführung:** Durchführung der geplanten Tests, Dokumentation der Ergebnisse und Erfassung von Fehlern.
4. **Testauswertung:** Analyse der Testergebnisse, Bewertung der Qualität und notwendiger Korrektur.
5. **Testabschluss:** Erstellung eines Testabschlussberichts und Durchführung einer Lessons-Learned.

Testdurchführung

Einleitung

In der Testdurchführung zeige ich die Resultate, welche ich bekommen habe. Ich habe resultat einmal vom Backend, welcher mit Swagger getestet wurde, und einmal den Frontend welche ich selbst getestet habe mit 'Prototyp Daten'. Ich teste hier nicht alle Funktionen, sondern nur die Post, Get und Delete Funktionen.

Swagger / Backend

Post

Als erstes sende ich Daten vom Swagger und probiere diesen in der Datenbank zu sehen.

Abbildung 14 Post ueber Swagger

Ich habe einen 200- Respnse das heisst das die Daten erfolgreich gesendet wurden.

Get

Im Swagger

Hier will ich sehen, ob ich die Daten überhaupt noch sehen kann und diese erfolgreich gespeichert wurden.

Abbildung 15 Get ueber Swagger

Ich sehe das ich die Daten erfolgreich gespeichert habe und ich alles zurückbekomme

Im Datenbanksystem

	Id	Title	Note	IsCompleted	DueDate	CalendarEventId
1	35	Testing über den Swagger	Diese Daten wurden vom Swagger gesendet	1	2024-06-17 13:54:21.1060000	NULL

Abbildung 16 Get im Datenbanksystem

Wir sehen hier das ich die Daten erfolgreich in der Datenbank gespeichert habe

Delete

Hier will ich sehen, ob ich die Daten auch löschen kann.

Abbildung 17 Delete-Response

Wir sehen das ich die Daten gelöst habe.

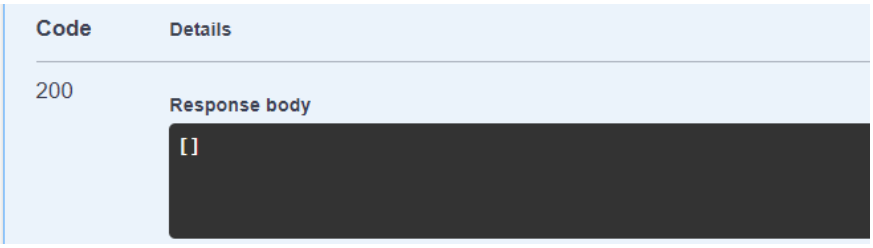
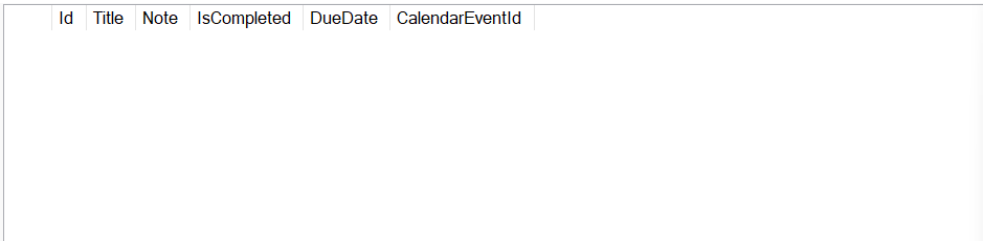


Abbildung 18 Get mit geloeschten Daten

Ich sehe diese Daten bei der Get Funktion auch nicht.



In dem Datenbanktool sehe ich die Daten auch nicht

Frontend

Nachdem ich das einiger meinen Kollegen gegeben habe und diese mit Bestätigten das es benutzerfreundlich sein, fing ich mit dem Testen an, ob ich Daten auch senden kann

Post

Diese Daten sende ich vom Frontend aus

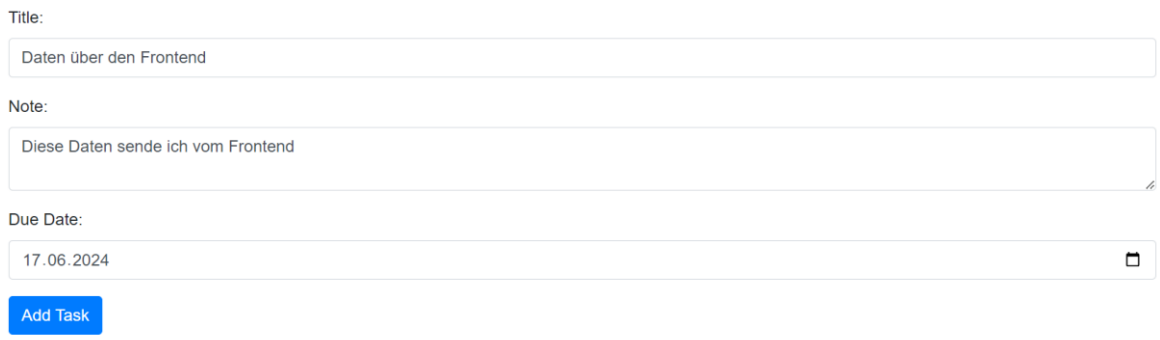


Abbildung 19 Post ueber Frontend

Get

Im Frontend

Diese Daten sehen ich dann auch auf meiner Übersicht Seite.

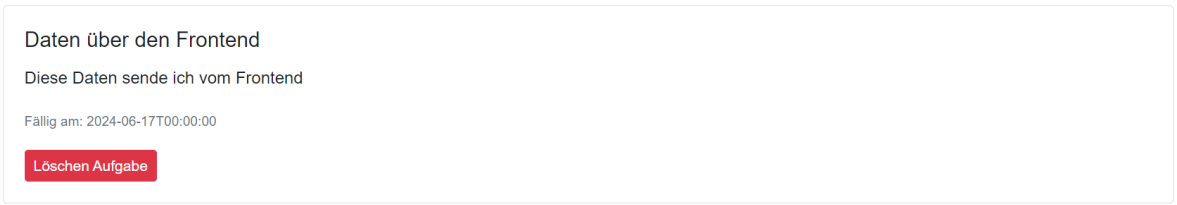


Abbildung 20 Get im Frontend

Im Datenbanksystem

	Id	Title	Note	IsCompleted	DueDate	CalendarEventId
1	36	Daten über den Frontend	Diese Daten sende ich vom Frontend	0	2024-06-17 00:00:00.0000000	NULL

Abbildung 21 Get Im Datenbanksystem

Wir sehen das ich diese Daten auch in der Datenbank sehe. Sie Sehen das es eine ID gibt, obwohl ich keine angegeben habe, diese wird automatisch generiert und hilft dem Datenbankverwalter bei Suche.

Delete

Vielleicht haben sie oben gesehen das es eine schöne taste gibt wo steht ‘Löschen Aufgabe’. Hier testen wir, ob diese funktioniert.

Im Frontend

Was ich zu tun habe

Abbildung 22 Frontend Delete

Wir sehen das auf dem Frontend nach betätigen dieser Taste nichts mehr zu sehen ist.

Im Datenbanksystem

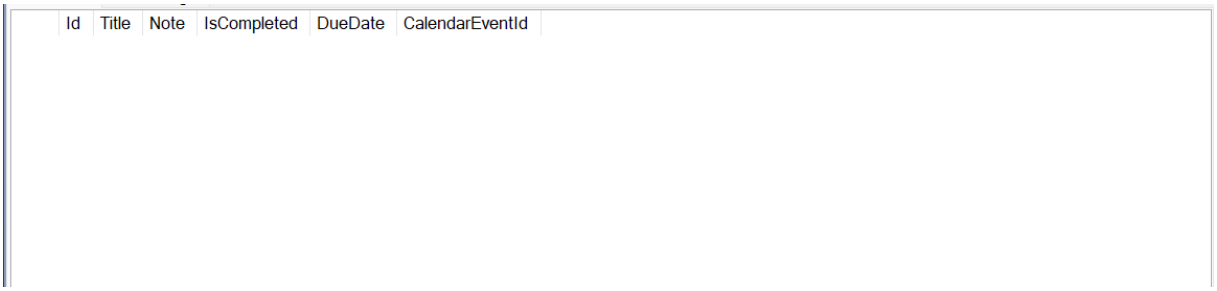


Abbildung 23 Delete vom Frontend in der Datenbank

Wir sehen das in der Datenbank auch nichts mehr zu sehen ist von dieser Aufgabe.

Reflexion

Dieses Projekt verlief insgesamt sehr erfolgreich und die meisten Ziele wurden erreicht. Allerdings habe ich die benötigte Zeit unterschätzt, insbesondere für die Dokumentation, die viel aufwändiger war als erwartet. Zudem hatte ich Schwierigkeiten bei der Datenübertragung vom Frontend zum Backend, was zusätzliche Zeit für Fehlersuche und Anpassungen erforderte. Trotz dieser Herausforderungen konnte ich die Probleme lösen und wertvolle Erfahrungen sammeln, die mir in zukünftigen Projekten helfen werden. Insgesamt war es eine lehrreiche und erfolgreiche Erfahrung.

Ausblick

Für die nächste Version dieses Projekts plane ich, die Funktionalität zu erweitern, indem die Möglichkeit hinzugefügt wird, Termine zu den Aufgaben hinzuzufügen. Dies wird den Nutzern helfen, ihre Aufgaben noch effizienter zu verwalten und sicherzustellen, dass wichtige Deadlines nicht übersehen werden. Ich freue mich darauf, diese Verbesserungen umzusetzen und das Projekt weiter zu optimieren.

Glossar

API (Application Programming Interface)
Eine Schnittstelle, die es verschiedenen Softwareanwendungen ermöglicht, miteinander zu kommunizieren.

Backend
Der serverseitige Teil einer Anwendung, der die Geschäftslogik und Datenbankinteraktionen verwaltet.

Bootstrap
Ein CSS-Framework zur Entwicklung von responsive Webanwendungen.

C#
Eine Programmiersprache von Microsoft, die auf der .NET-Plattform verwendet wird.

CSS (Cascading Style Sheets)
Eine Sprache zur Gestaltung und Layout von HTML-Dokumenten.

Datenbank
Eine strukturierte Sammlung von Daten, die elektronisch gespeichert wird. In diesem Projekt wurde MSSQL verwendet.

Frontend
Der clientseitige Teil einer Anwendung, der die Benutzeroberfläche und Interaktionen bereitstellt.

HTML (Hypertext Markup Language)
Die Standard-Auszeichnungssprache für die Erstellung von Webseiten.

JavaScript
Eine Programmiersprache zur Erstellung interaktiver Webinhalte.

JSON (JavaScript Object Notation)
Ein Datenformat zum Austausch von Daten zwischen einem Server und einer Webanwendung.

MSSQL (Microsoft SQL Server)
Ein relationales Datenbankmanagementsystem von Microsoft.

Swagger
Ein Framework zur Dokumentation und Nutzung von RESTful Web-APIs.

Unit-Test
Ein Test, der einzelne Komponenten einer Software isoliert überprüft.

Webhost
Ein Dienst, der Webseiten und Webanwendungen im Internet bereitstellt.

Abbildungsverzeichnis

GitHub

Der Link zu meinem GitHub in welchem man das ganze Projekt hat;

https://github.com/Davud-Ponjevic/SOL_Projekt