Hochschule für Technik
und Wirtschaft Berlin
University of Applied Sciences

STORESERVER
eBusiness neu denken

Praktikumsbericht

STORESERVER

Vorgelegt von

Mohammed Al-Hamdi

s0575297

Mail:

mohammed.al-hamdi@Student.HTW-Berlin.de

Betreuer der Hochschule: Prof. Dr. Mohammad Abuosba

Copyright

Inhaltsverzeichnis

Inhalt

1 Einleitung

Webtechnologie und Softwareentwicklung sind heute unverzichtbar, da sie die Grundlage für die digitale Kommunikation und den Informationsaustausch bilden. Sie ermöglichen es Unternehmen, ihre Produkte und Dienstleistungen online anzubieten, was den globalen Handel und die Wirtschaft fördert. Zudem treiben sie Innovationen in vielen Bereichen wie Medizin, Bildung und Unterhaltung voran, was zu einem verbesserten Lebensstandard beiträgt. Schließlich spielen sie eine entscheidende Rolle bei der Automatisierung und Effizienzsteigerung von Geschäftsprozessen, was Zeit und Ressourcen spart.

Im Zeitraum vom 15.02. bis zum 15.05. hatte ich die spannende Gelegenheit, mein Praktikum beim der Storeserver Systems GmbH Firma zu absolvieren. Die Storeserver ist eine Softwarefirma, die sich mit Webentwicklung beschäftigt und PIM-Lösungen, eProcurement-Plattformen sowie elektronische Produktkataloge anbietet. Während meiner Praktikumszeit hatte ich die Möglichkeit, an einem faszinierenden Projekt mitzuwirken, das sich mit der Erarbeitung eines Konzepts und der Umsetzung einer Software zur Visualisierung der vorhandenen Kompetenzen und Technologiefelder der Mitarbeitenden und Projekte befasste. Im Rahmen meines Praktikums hatte ich die einzigartige Gelegenheit, in einem professionellen und forschungsorientierten Umfeld zu arbeiten.

Das Storeserver-Team war äußerst engagiert, kompetent und hilfsbereit, was mir ermöglichte, wertvolle Erfahrungen und Kenntnisse zu sammeln. Ich wurde herzlich in das Team aufgenommen und erhielt Unterstützung und Anleitung bei meinen Aufgaben. Während der täglichen Meetings konnte ich viele Erfahrungen sammeln, indem ich mit dem Team Entwicklungsprobleme besprach und löste.

Während meiner Zeit bei der Storeserver habe ich mit dem Blazor Server Framework gearbeitet und als Frontend-Technologien Syncfusion Bibliotheken sowie HTML und CSS verwendet.

Für das Backend habe ich in C# entwickelt, wobei ich bereits während meines Studiums umfangreiche Kenntnisse in dieser Sprache erworben habe. Ein großer Teil meiner Aufgaben bestand schließlich im Testen von Frontend-Komponenten mit bUnit und dem Backend mit xUnit.

Während meines Praktikums hatte ich auch die Möglichkeit, mit anderen talentierten und motivierten Studierenden und Mitarbeitenden zusammenzuarbeiten. Die regelmäßigen Meetings, Diskussionen und Teamarbeit halfen mir, mein Verständnis für das Projekt zu vertiefen und neue Perspektiven zu gewinnen.

Insgesamt war meine Praktikumszeit bei der StoreServer Firma eine äußerst bereichernde und lehrreiche Erfahrung. Das Unternehmen bot mir die Möglichkeit, meine theoretischen Kenntnisse in der Praxis anzuwenden und meine Fähigkeiten in der Softwareentwicklung weiterzuentwickeln. Ich bin dankbar für die Unterstützung und das Vertrauen, das mir während meines Praktikums entgegengebracht wurde, und freue mich, dass ich einen wertvollen Beitrag zu diesem innovativen Projekt leisten konnte.

2 Vorstellung des Praktikumsorts(StoreServer)

Die StoreServer Firma wurde 1999 gegründet und ist seitdem im Softwaregeschäft tätig. Mit einer Mitarbeiterzahl von 25 und einer Kundenanzahl von 100 hat sich das Unternehmen fest etabliert. Der Grundstein der Firma wurde von dem Geschäftsführer Dipl.-Inf. Sven Sprandel in Zusammenarbeit mit dem Fraunhofer IAO gelegt. Das aus dieser Zusammenarbeit resultierende Austauschformat (openTRANS/BMEcat) war das erste Thema, dem sich die Firma mit dem Storeserver BMEcat Generator widmete und dem sie sich bis zum heutigen Tag widmet.

Mit mehr als 15 Jahren Erfahrung am Markt ist die Firma nicht nur ein kompetenter, sondern auch ein vertrauenswürdiger Ansprechpartner.

3 Konzeption der Software

3.1 Auswahl der Technologien

3.1.1 Framework

3.1.1.1 ASP.NET Core Blazor

Blazor ist ein Webframework zur Erstellung von Komponenten von Webbenutzeroberflächen (Razor-Komponenten), die auf verschiedene Arten gehostet werden können. Razor-Komponenten können serverseitig in ASP.NET Core (Blazor Server) und clientseitig im Browser in einer WebAssembly-basierten .NET-Runtime (Blazor WebAssembly, Blazor WASM) ausgeführt werden.

Blazor Server

Mit dem Blazor Server-Hostingmodell werden Komponenten über eine ASP.NET Core-App auf dem Server ausgeführt. Benutzeroberflächenupdates, Ereignisbehandlung und JavaScript-Aufrufe werden über eine SignalR-Verbindung verarbeitet, welche das WebSockets-Protokoll verwendet. Der Zustand auf dem Server, der jedem verbundenen Client zugeordnet ist, wird als Leitung bezeichnet. Leitungen sind an keine spezifische Netzwerkverbindung gebunden und können vorübergehende Netzwerkunterbrechungen tolerieren sowie Versuche des Clients, die Verbindung mit dem Server bei einer Unterbrechung wiederherzustellen.

3.1.2 Frontend

3.1.2.1 Syncfusion Blazor

Syncfusion ist eine moderne Bibliothek, die sich auf die Entwicklung benutzerfreundlicher und reaktionsfähiger Webanwendungen spezialisiert hat. Sie zeichnet sich durch ihre Flexibilität, Modularität und leichte Erlernbarkeit aus. Die klar strukturierte Syntax von Syncfusion erleichtert die Entwicklung und verbessert die Wartbarkeit des Codes erheblich. Die Entscheidung für Syncfusion als Frontend-Lösung wurde durch mehrere Faktoren beeinflusst. Erstens unterstützt Syncfusion die Entwicklung interaktiver Benutzeroberflächen und ermöglicht eine nahtlose Integration von Datenvisualisierungen und Komponenten. Dies war entscheidend, um die Anforderungen der Software hinsichtlich der Visualisierung von Kompetenzen und Technologiefeldern zu erfüllen. Zweitens verfügt Syncfusion über eine große und aktive Entwicklergemeinschaft, die kontinuierlich neue Erweiterungen, Komponenten und Tools entwickelt. Dadurch wird die Weiterentwicklung der Software erleichtert und es besteht Zugang zu einer Vielzahl von Ressourcen und Lösungen für eventuell auftretende Herausforderungen während der Entwicklung.

3.1.3 Backend

3.1.3.1 C#

Die Programmiersprache C# unterstützt Blazor Server, und die Entscheidung für C# und Blazor bietet Entwicklern die Möglichkeit, die Vorteile von C# und .NET in der Webentwicklung zu nutzen, ohne auf JavaScript oder andere Sprachen angewiesen zu sein. Diese Kombination erleichtert die Wiederverwendung von Code und ermöglicht das Schreiben konsistenter Anwendungen in einem vertrauten Sprachumfeld. Durch die Verwendung von WebAssembly kann Blazor eine bemerkenswerte Leistung erzielen, da der C#-Code direkt im Browser ausgeführt wird. Dies führt zu verkürzten Ladezeiten und verbessert die Reaktionsfähigkeit der Anwendung. Zusätzlich ermöglicht Blazor die serverseitige Synchronisierung von Zuständen, was den Netzwerkverkehr minimiert.

3.1.4 Testing

3.1.4.1 bUnit Testing

bUnit ist ein Framework für das Testen von Razor-Komponenten in Blazor-Anwendungen. Es ermöglicht das Schreiben von Unit-Tests für die Benutzeroberfläche auf Komponentenebene. Hier sind einige Hauptmerkmale und Funktionen von bUnit:

1. Razor-Komponenten-Tests: Mit bUnit können Sie Tests für Ihre Razor-Komponenten schreiben. Dies ermöglicht es Ihnen, das Verhalten und die Darstellung Ihrer Benutzeroberfläche zu überprüfen.

2. Simulation von Benutzerinteraktionen: Sie können mit bUnit Benutzerinteraktionen simulieren, wie z.B. das Klicken auf Buttons oder das Ausfüllen von Formularen. Dadurch können Sie überprüfen, ob Ihre Benutzeroberfläche wie erwartet auf Benutzeraktionen reagiert.

3. Assert-Methoden: bUnit bietet eine Reihe von Assert-Methoden, mit denen Sie das Verhalten und die Darstellung Ihrer Razor-Komponenten überprüfen können. Sie können z.B. überprüfen, ob bestimmte Elemente auf der Seite vorhanden sind oder ob bestimmte Eigenschaften von Komponenten korrekt gerendert wurden.

4. Integrierte Mocking-Funktionen: bUnit integriert sich nahtlos mit anderen Mocking-Frameworks wie Moq, um das Mocking von Abhängigkeiten in Ihren Tests zu erleichtern.

3.1.4.2 xUnit Testing

Beim xUnit backend Testing" handelt es sich um das Testen des Backend-Codes einer Anwendung mit dem xUnit-Framework. Hier sind einige Merkmale und Funktionen des xUnit-Testframeworks im Kontext von Backend-Tests:

1. **Unit-Tests**: Mit xUnit können Entwickler Unit-Tests schreiben, um sicherzustellen, dass einzelne Komponenten des Backends korrekt funktionieren. Unit-Tests zielen darauf ab, einzelne Einheiten des Codes, wie z.B. Methoden oder Funktionen, zu isolieren und zu testen.

2. **Testmethoden und Assert-Statements**: xUnit bietet verschiedene Attribute für Testmethoden und Assert-Statements, um Testfälle zu definieren und zu überprüfen, ob das erwartete Verhalten eintritt.

3. **Setup und Teardown**: xUnit ermöglicht das Ausführen von Setup- und Teardown-Logik vor bzw. nach jedem Testfall, um die Umgebung für den Test vorzubereiten und nach dem Test wiederherzustellen.

4. **Parameterized Tests**: Mit xUnit können Parameterized Tests erstellt werden, um denselben Testcode mit verschiedenen Eingabewerten auszuführen und die Ergebnisse zu überprüfen.

5. **Testausführung und Berichterstattung**: xUnit bietet verschiedene Möglichkeiten, Tests auszuführen und Berichte über die Testergebnisse zu generieren, einschließlich der Verwendung von Test-Runner-Tools und integrierten Berichterstattungsfunktionen.

3.2 Architektur des Systems

Allgemeine Vorstellung des Projektes

Das Projekt verwendet das Blazor Server Framework und ist im Backend in der Programmiersprache C# entwickelt. OData spielt eine wichtige Rolle als Verbindung zwischen Frontend und Backend. Darüber hinaus ist das Frontend mit Syncfusion UI integriert.

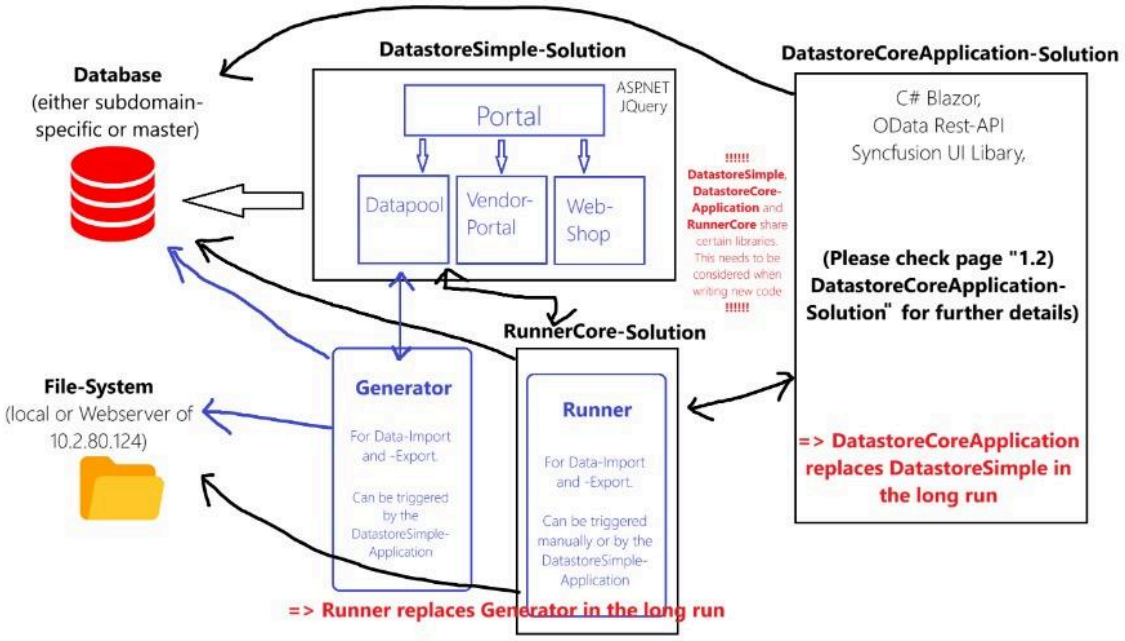


Abbildung 1: Allgemeine Vorstellung des Projektes

Backend

Als Backend verwendet das Projekt zuerst Services, das für SQL-Befehle zur Verbindung mit der Datenbank und dem Projekt geeignet ist. Danach kommt der Controller, der für die Anzeige, Übermittlung und Löschung von Elementen über verschiedene Endpunkte verantwortlich ist (HttpGet für die Anzeige von Elementen aus der Datenbank, HttpPost für das Senden von Elementen an die Datenbank und HttpDelete für das Löschen von Elementen aus der Datenbank). Dann folgt der DataAdapter, der für die Verbindung zwischen UI und Controllern zuständig ist.

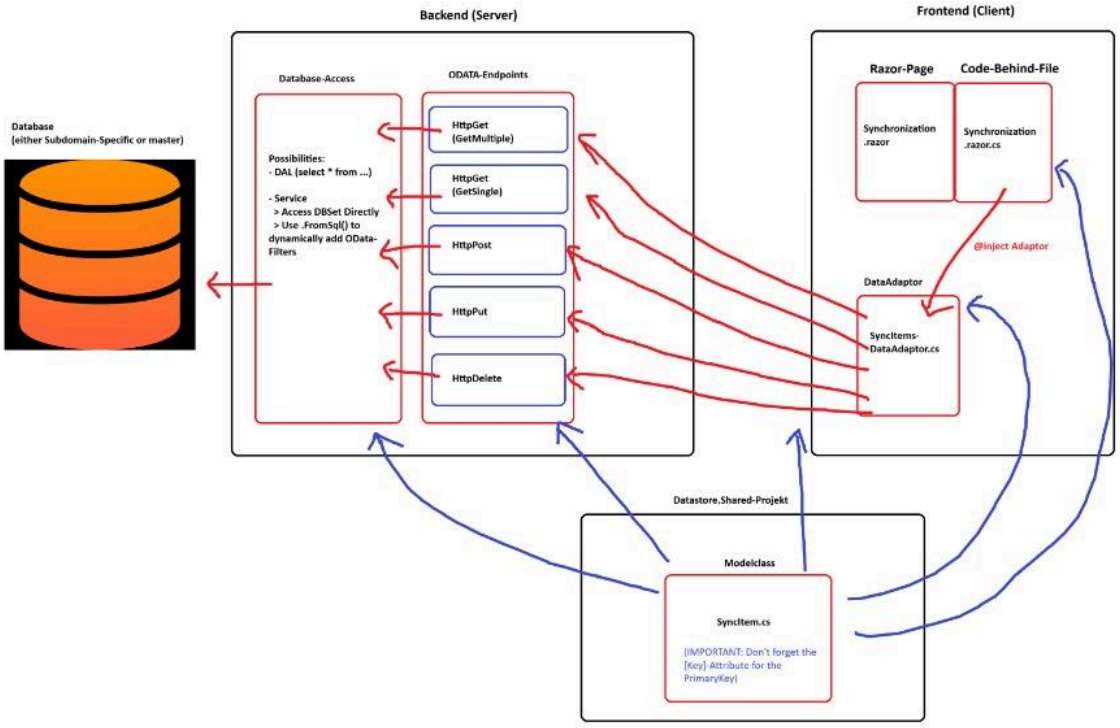


Abbildung 2: Vorstellung des Backenden

Wichtigste Projekte

Insgesamt gibt es 11 Projekte

DatastoreCoreServices: Projekt zur Bereitstellung des Großteils des Backends, einschließlich ODATA-Endpunkten.

Datastore.Shared: Enthält Modellklassen, einschließlich Resx-Dateien.

DatastoreStandaloneServerPure: Das Projekt, das tatsächlich gehostet wird. Es verweist auf das DatastoreCoreRazorComponents-Projekt, welches alle Steuerelemente und UI-Elemente enthält.

Tests: Enthält Frontend test mit bUnit test und Backend test mit xUnit.

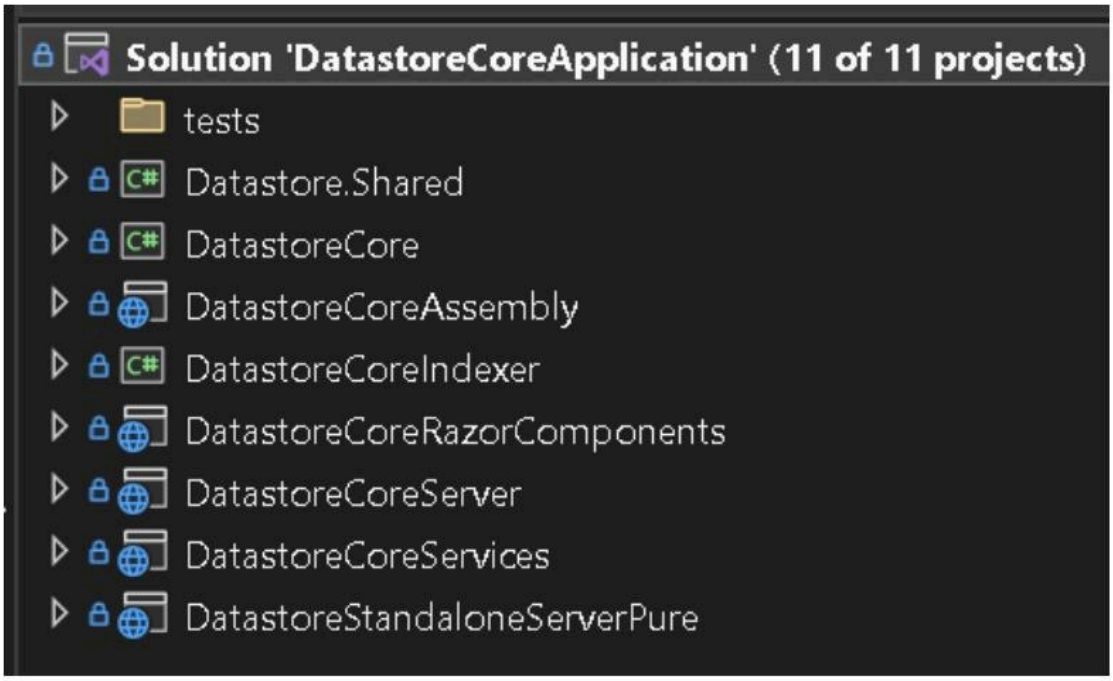


Abbildung 3: Alle Projekte

Projektstruktur

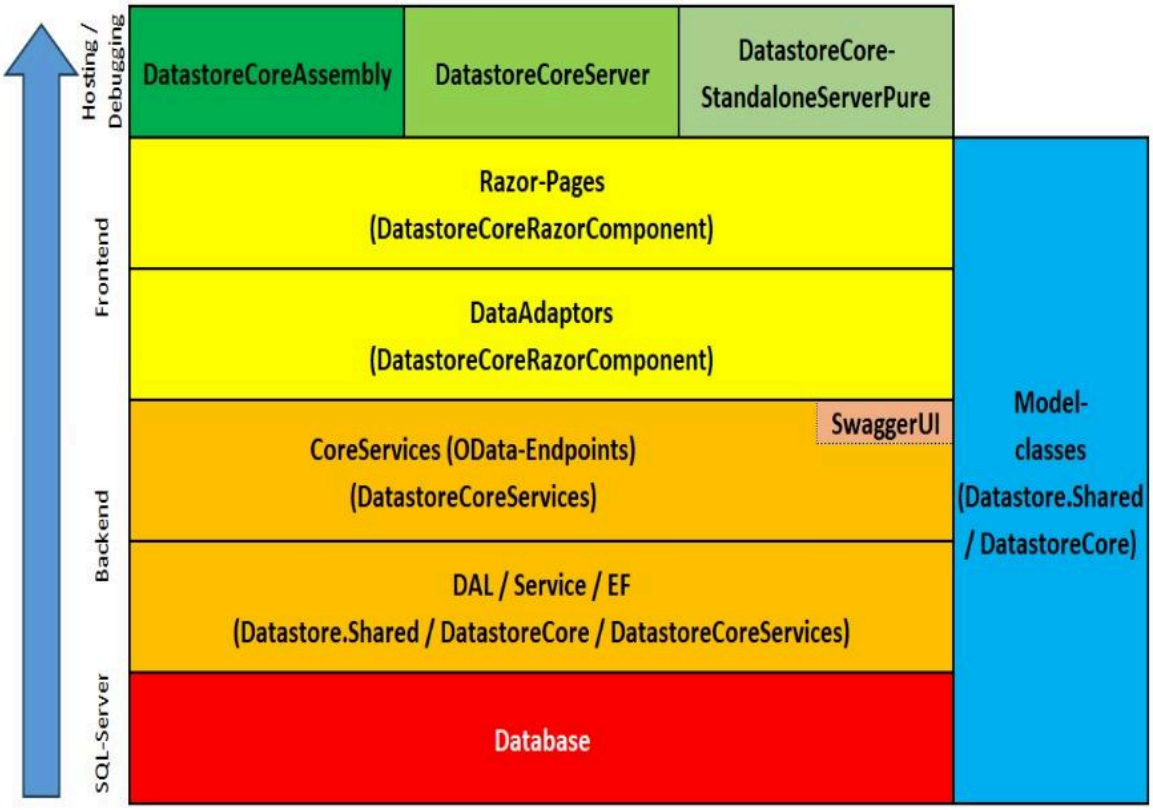


Abbildung 4: Projektstruktur

4 Aufgaben und Umsetzung

Im Rahmen des Praktikums wurden mir mehrere Projekte in der Abteilung Software-Webentwicklung zugeteilt. Meine Aufgaben lagen in der Entwicklung und dem Testing der Projekte.

Überblick über den betrieblichen Alltag: Der Arbeitstag umfasst 8 Arbeitsstunden. Jeden Tag beginnt um 11:30 Uhr das Team-Meeting, in dem sich alle Mitarbeiter*innen zusammensetzen und von dem Abteilungsleiter die Ereignisse des vergangenen Tages zusammengefasst sowie bevorstehende Events eingeführt werden. Außerdem soll jeder Mitarbeiter seinen Teil des vergangenen Tages vorstellen und auch die Probleme diskutieren, wenn es welche gibt.

Zu Beginn des Praktikums wurde mir zunächst eine Führung durch die Büros und die Projektinhalte gegeben. Daraufhin wurde mir ein Arbeitslaptop zur Verfügung gestellt. Über diesen konnte ich mit anderen Angestellten kommunizieren und an den zugeteilten Aufgaben arbeiten.

Insgesamt ist das Projekt gut aufgeteilt, und jeder hilft jedem bei der Bewältigung der Projektinhalte.

Im Folgenden werden die mir zugeteilten Teilprojekte näher beschrieben.

4.1.1 Erstellung von SearchindexStatus Seite

Anforderung: Erstellung einer vollständigen Seite mit Syncfusion UI-Frontend und C# als Backend mit OData zur Verbindung zwischen Frontend, Backend und Datenbank.

Was macht die Seite: Die Seite stellt Bereiche von Elementen aus der Datenbank in alphabetischer Ordnung dar. Unter jedem Bereich erfolgen indizierte Elemente. Die Seite hat zwei Buttons (Neu indizieren und Aktualisieren)

1. Neu indizieren-Button: Der Button soll die indizierten Elemente filtern, um eine schnellere Auffindbarkeit der Inhalte zu ermöglichen.

2. Aktualisieren-Button: Der Button aktualisiert die Elemente.

Aufbau der Seite: Die Seite hat 5 Teile:

1. SearchIndexStatusModel.cs: Hier ist die Modellklasse mit 3 Eigenschaften, in denen die Elemente gespeichert werden.

2. SearchIndexStatusService.cs: In dieser Klasse befinden sich Methoden mit SQL-Befehlen zur Kommunikation mit der Datenbank. Beispiel für einen SQL-Befehl in der Get-Methode:

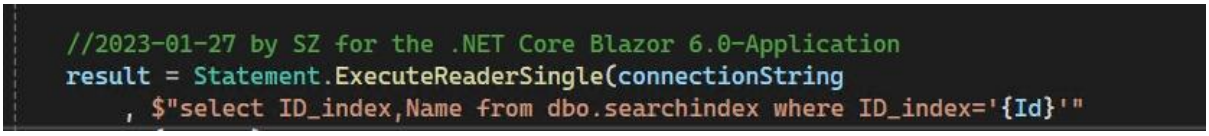


Abbildung 5: SQL-Befehl

3. SearchIndexStatusController.cs: Diese Klasse leitet von der Service-Klasse ab und kontrolliert die Anzeige und Filterung der Elemente aus der Datenbank mit zwei Methoden: Get-Methode für die Anzeige der Elemente aus der Datenbank und Post-Methode für die Filterung und das Einfügen der Elemente in die Datenbank.

4. SearchIndexStatusDataAdapter.cs: Der DataAdapter dient als Brücke zwischen einem DataSet und einer Datenquelle zum Abrufen und Speichern von Daten. Der DataAdapter stellt diese Brücke durch die Zuordnung von Fill zur Verfügung, wodurch die Daten im DataSet so geändert werden, dass sie den Daten in der Datenquelle entsprechen, und Update, wodurch die Daten in der Datenquelle so geändert werden, dass sie mit den Daten im DataSet übereinstimmen. Diese Klasse leitet vom Controller ab und wird im Frontend verwendet, um eine sichere und erfolgreiche Verbindung zu gewährleisten.

5. ManageSearchIndexStatus.razor: In dieser Razor-Datei wird das Frontend geschrieben. Auf dieser Seite wird die Syncfusion UI-Bibliothek verwendet, um eine moderne Benutzeroberfläche zu erstellen. Die Elemente werden durch ein CustomGrid in einer Tabelle dargestellt, und es gibt zwei Buttons zur Steuerung der Elemente. Im Code-Behind sind die Methoden vom DataAdapter abgeleitet und werden verwendet.

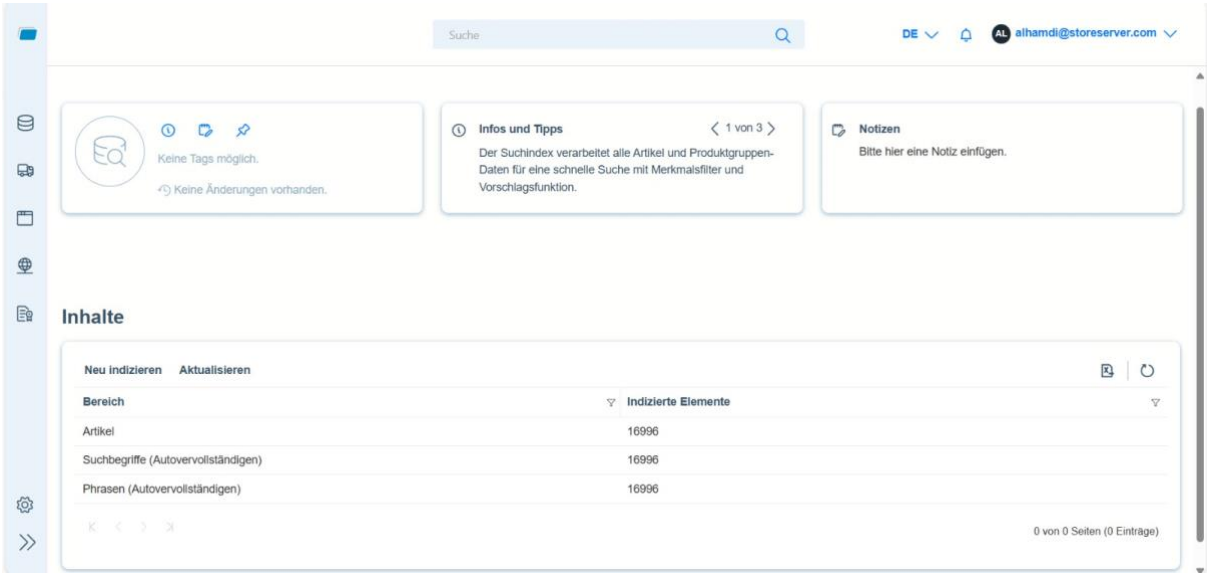


Abbildung 6: SearchindexStatus Seite

4.1.2 Implementierung einer Such Box für Tabellen

Anforderung: Erstellung einer Such Box für Grid-Elemente, um eine schnelle Suche nach Elementen auf allen Projektseiten zu ermöglichen.

Hinweis: Die Such Box wurde auf Seiten implementiert, die Tabellen mit Grid-Elementen enthalten, und die Anzahl dieser Seiten beträgt 24.

Aufbau der Such Box: Die Implementierung erfolgte in der Controller.cs der Seiten. Sobald ein Buchstabe in die Such Box eingegeben wird, werden schnell alle Ergebnisse mit diesem Buchstaben alphabetisch angezeigt. Wenn spezifisch nach einem Wort gesucht wird, soll das genaue Wort gefunden werden.

Vorteil der Such Box ist, dass alle Elemente schnell und effektiv gefunden werden können.

4.1.3 Dropdown-Liste alphabetisch ordnen

Problem: Alle Dropdown-Listen hatten die Elemente zufällig alphabetisch geordnet, was viel Zeit kostet, um die Elemente in der Dropdown-Liste zu finden.

Lösung: Die Dropdown-Listen wurden mit der Syncfusion-Bibliothek erstellt, und Syncfusion bietet ein Feature, das mit diesem Code alle Elemente in der Dropdown-Liste alphabetisch ordnet.

```
<SfDropDownList @ref=DropdownValueList Enabled=@_controlsEnabled TValue="string" SortOrder="Syncfusion.Blazor.DropDowns.SortOrder.Ascending" TItem="ClassificationSystemValueList"
    <SfDataManager Url="@ClassificationSystemValueListUrl" Headers=@HeaderData CrossDomain="true" Adaptor="Adaptors.ODataV4Adaptor">
    </SfDataManager>
    <DropDownListTemplates TItem="ClassificationSystemValueList">
        <NoRecordsTemplate>
            <SfButton IconCss="e-icons e-plus" Content="@Datastore.Shared.Resources.Classification.ClassificationSystemClassFeature.Button_AddNewFeature"  OnClick="HandleAddFeatu
        </NoRecordsTemplate>
    </DropDownListTemplates>
    <DropDownListEvents TValue=string TItem="ClassificationSystemValueList" Filtering="OnFilteringValueList"></DropDownListEvents>
    <DropDownListFieldSettings Text="Name" Value="ValueListId"></DropDownListFieldSettings>
</SfDropDownList>
```

Abbildung 7:alphabetisch ordnen Syncfuion UI

Hinweis: Die Anzahl der Seiten mit Dropdown-Listen beträgt 16.

4.1.4 Teste das Frontend mit Bunit Testing

Anforderung: Teste die Frontend-Seiten, die Grid-Elemente und Buttons haben.

1. Seiten mit Grid- Elementen testen

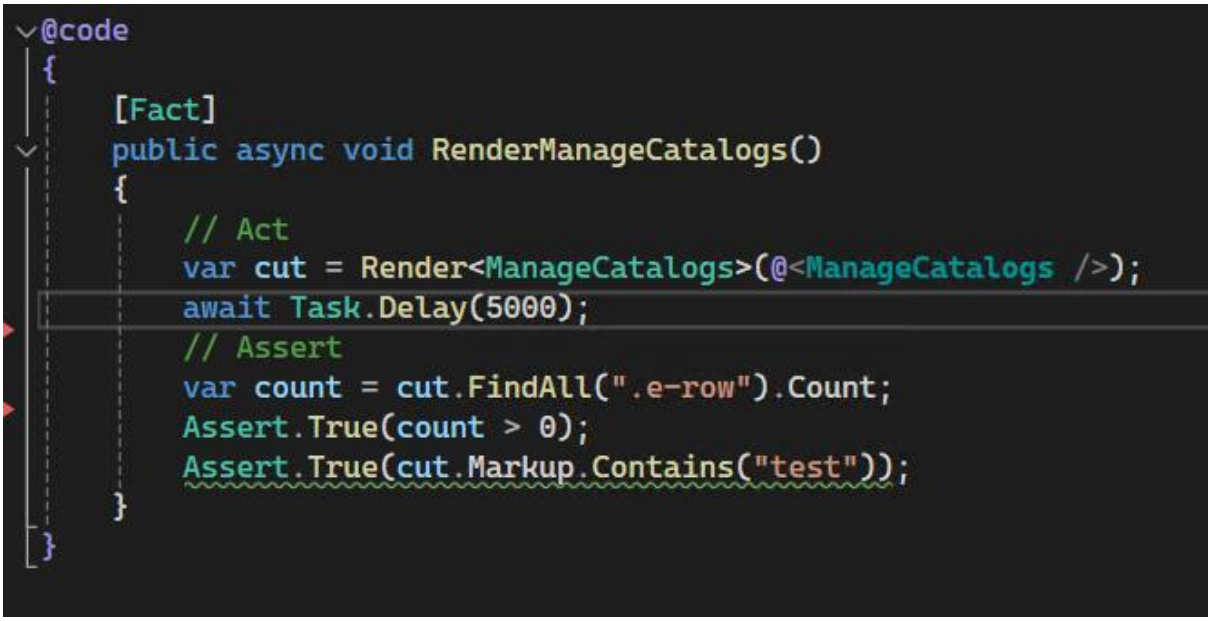
   Problem: Die Frontend-Grid-Elemente wurden mit der Syncfusion-Bibliothek entwickelt, und diese Bibliothek unterstützt keine bUnit-Tests mit OData-Verbindung, da der Adapter als Verbindung zwischen Controller und backend dient.

   Lösung: Erstellung einer Klasse, die alle HTTP-Anfragen mockt, ohne den DataAdapter zu verwenden. Das heißt, die Klasse erstellt eine Fake-Anfrage, ähnlich wie der DataAdapter.

```
// Mock all http requests
MockHttpRequest = Services.AddMockHttpClient();
MockHttpRequest.When("*").Respond((HttpRequestMessage request) =>
{
    var client = Services.GetRequiredService<IHttpClientFactory>().CreateClient();
    request = request.Clone();
    return client.SendAsync(request);
});
```

Abbildung 8: HTTP Mock Code

Implementierung: Zuerst wurde die Frontend-Seite mit bUnit gerendert, dann werden die Grid-Elemente geladen. Anschließend wird geprüft, ob die Elemente die gleichen Namen haben und ob die Anzahl der Elemente mit der Tabelle auf der Webseite übereinstimmt. Hier ist ein Beispiel:



```
@code
{
    [Fact]
    public async void RenderManageCatalogs()
    {
        // Act
        var cut = Render<ManageCatalogs>(@<ManageCatalogs />);
        await Task.Delay(5000);
        // Assert
        var count = cut.FindAll(".e-row").Count;
        Assert.True(count > 0);
        Assert.True(cut.Markup.Contains("test"));
    }
}
```

Abbildung 9: Frontend Test mit bUnit

2. Seiten mit Buttons testen:

   Anforderungen: Die Seiten mit den Buttons zum Hinzufügen, Bearbeiten und Löschen von Elementen sollen überprüft werden, ob sie richtig funktionieren.

   Implementierung: Beim Hinzufügen von Elementen wird zuerst die Frontend-Seite mit bUnit gerendert. Dann werden die Test-Textboxen mit den passenden Datentypen gefüllt. Danach wird der Speichern-Button mit bUnit geklickt, um die ausgefüllten Elemente zu speichern. Anschließend wird überprüft, ob die ausgefüllten Elemente mit den gespeicherten Elementen übereinstimmen.

   Ziel des Frontend-Testings mit bUnit: Es soll die Struktur und die Aktionen, die der Benutzer im Frontend durchführen kann, geprüft werden, um mögliche Fehler oder Bugs frühzeitig und effizient zu erkennen.

4.1.5 Teste das Backend mit xUnit

Anforderungen: Teste die Backend-Seiten, um Fehler im Backend-Code zu erkennen.

Implementierung: Da zwischen den Klassen Controller und Frontend eine Verbindung mit einem DataAdapter besteht, wurde ein Mock-DataAdapter in den Backend-Tests implementiert. Dies ermöglicht es, das Verhalten des Adapters für jeden Test individuell zu steuern. Während ein reales Objekt beispielsweise aufwendige Datenbankzugriffe ausführt und aufgrund von Datenaktualisierungen bei jedem Aufruf unterschiedliche Ergebnisse liefert, kann ein Mock-Objekt so konfiguriert werden, dass es bei jedem Aufruf ein genau festgelegtes Ergebnis an das zu testende Objekt zurückgibt. Da aussagekräftige Tests konstante und konsistente Testbedingungen erfordern, sind Mock-Objekte oft eine sinnvolle Alternative zu realen Objekten.

Indem Mock-Objekte einen Teil der vorhandenen Objekte in einer Anwendung während eines Unit-Tests ersetzen, kann das zu testende Objekt unabhängig vom Rest der Anwendung getestet werden. Die Testabläufe können durch das anpassbare Verhalten der Mock-Objekte gesteuert werden. Gleichzeitig können die erwarteten Methodenaufrufe spezifiziert werden, die das zu testende Objekt an die Mock-Objekte ausführt. Auf diese Weise können idealerweise alle möglichen Zustände, in denen sich das zu testende Objekt befinden kann, künstlich herbeigeführt und überprüft werden.

Nach dem Einsatz des Mock-DataAdapters wurde überprüft, ob der Code in der Controller-Klasse in der richtigen Reihenfolge ausgeführt wird und ob alle Ergebnisse korrekt zurückgegeben werden.

Testergebnisse: Man kann die Testergebnisse nach dem Start des Tests einsehen. Wenn ein Test nicht bestanden wird, muss man zuerst den Testcode und danach den getesteten Code (Frontend und Backend) überprüfen.

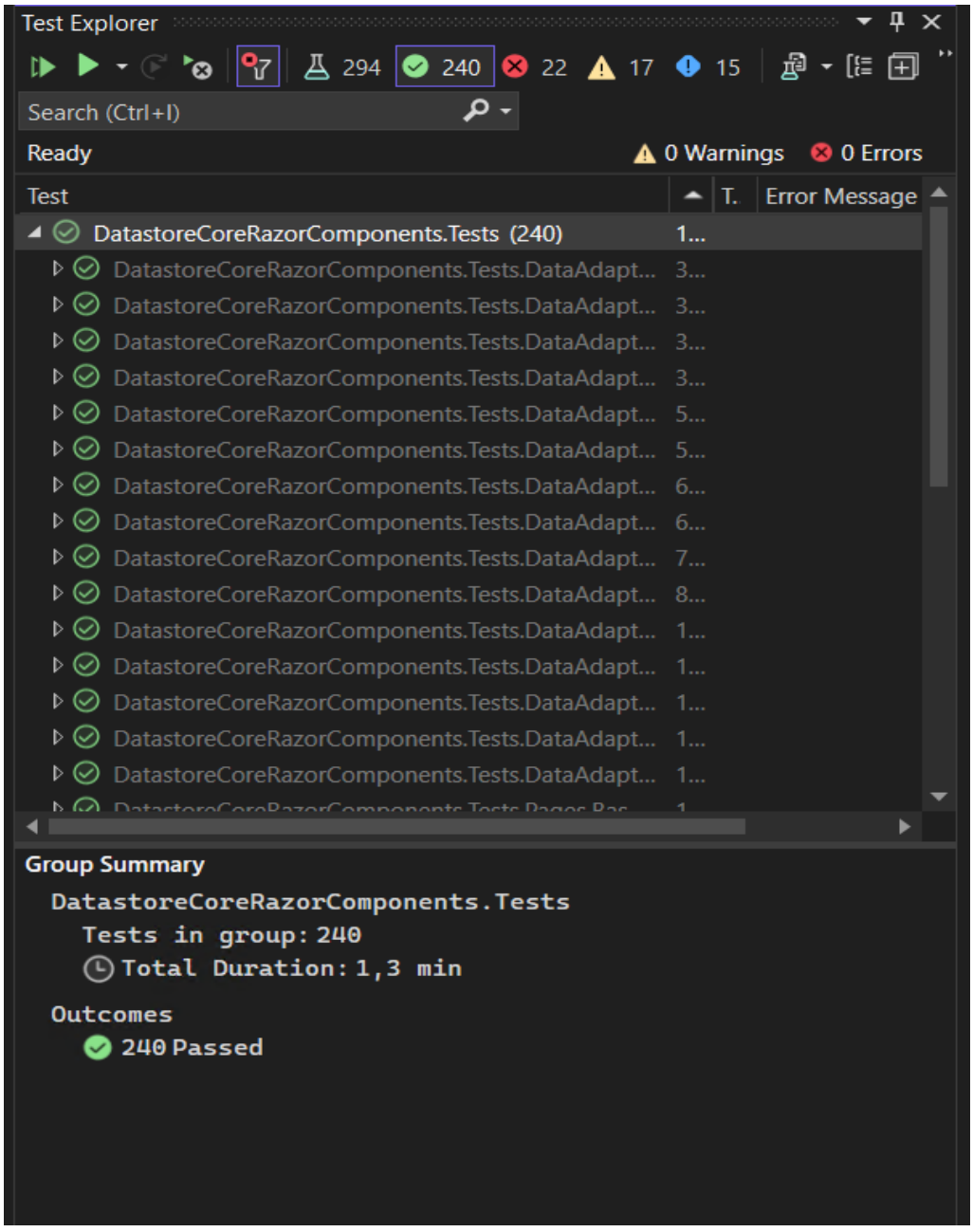


Abbildung 10: Projekt Testergebnisse

4.1.6 Dokumentation aller erledigten Aufgaben

Anforderung: Nach jeder Implementierung habe ich eine kurze Dokumentation erstellt, um dem restlichen Team Erklärungen zu geben. Außerdem soll ich täglich meine Aufgaben kurz präsentieren und meine Schwierigkeiten aufzeigen.

Struktur der Dokumentation:
Die Dokumentation ist in zwei Teile unterteilt:
1. Welche Dateien und Klassen von mir geändert wurden.
2. Wie die Probleme gelöst wurden bzw. wie der Code geschrieben und weiterentwickelt wurde.

5 Fazit

5.1 Meine Erwartungen

Als Praktikant mit drei Monaten Pflichtpraktikum wollte ich Erfahrungen im Arbeitsfeld sammeln, meine Studienkenntnisse anwenden und vertiefen sowie in einem gut organisierten Team arbeiten. Ich habe den Bereich Software- und Webentwicklung gewählt, da ich mich sehr für neue Technologien im Softwarebereich interessiere und einen Überblick über zukünftige Webtechnologien gewinnen möchte.

5.2 Meine Erfahrungen

In der Firma Soreserver habe ich viel gelernt, da das Team sehr hilfreich war, wenn ich Probleme bei der Arbeit hatte. Durch die täglichen Meetings konnte ich meine Probleme und Lösungen präsentieren, was meine Präsentationskenntnisse verbessert hat. Die Kommunikation mit dem Team spielte eine wichtige Rolle, da wir gleichzeitig an Projekten gearbeitet haben. Ich habe neue Frameworks und Technologien wie Syncfusion UI gelernt und meine Programmierkenntnisse in C# verbessert. Die Zeitorganisation war sehr wichtig, um Aufgaben effizient zu erledigen. Zudem habe ich erfahren, wie eine Firma strukturiert ist.

04.06.2024

Ort, Datum

Unterschrift