Projektdokumentation - Challenge – Bau eines eigenen Frameworks

Auftragsnummer: HFINFA.X.BA.3_4.135
Projektstart: 12.12.2024
Beratung: 13.12.2024
Abgabedatum: Letzter Unterrichtstag des Semesters
Bearbeitet von: Christian Herren
Gruppe: Jermain Huber, Raienthan Rasakumar, Marko Barutcu

Inhaltsverzeichnis

Zielsetzung und Aufgabenstellung

Zielsetzung

Anwendung erworbener Fähigkeiten:

Im Rahmen dieses Projekts werden Konzepte wie MVC, Dependency Injection, Datenbankzugriff mit EF Core, Excel-Import/Export, Validierung und modulare Architektur angewandt.

Aufbau eines eigenen Frameworks:

Es sollen wiederverwendbare Komponenten (Controller und Model) entwickelt werden, die in einem einfachen Anwendungskontext – einem Buchshop – eingesetzt werden können.

Praxisanwendung:

Der Buchshop ermöglicht es Benutzern, sich anzumelden, aus einer in Excel gehaltenen Produktliste Bücher auszuwählen und Bestellungen aufzugeben. Ausserdem wird geprüft, ob gewünschte Produkte vorhanden sind.

Aufgabenstellung

Die Aufgabe umfasst folgende Kernfunktionen:

Model-Schicht:

Schreiben und Lesen von Objekten in eine lineare Liste (leicht)
Schreiben in und Lesen aus einer Datenbank (mittel)
Lesen aus und Schreiben in Excel-Dateien (schwer)

Controller-Schicht:

Verifizieren, ob ein eingegebenes Datum möglich ist (leicht)
Abgleichen eines Eintrags mit einer in Excel angelegten Liste (mittel)

Praxisbeispiel:

Aufbau eines Buchshops, in dem:
Benutzer sich mittels Benutzername (E-Mail) und Passwort anmelden.
Eine Excel-basierte Produktliste zum Kauf zur Verfügung steht.

Bestellungen erfasst werden und das Bestelldatum validiert wird (soll zwischen dem 01.01.2020 und dem aktuellen Datum liegen).

Rahmenbedingungen und Bewertung

Gruppenarbeit: 2–3 Studenten
Verfügbare Zeit: Bis Ende des Semesters
Beratung: Am 13.12.2024 und danach per E-Mail oder Teams

Bewertung

Dokumentation der Komponenten in Form einer Gebrauchsanweisung (10 Punkte)
Pro erstellte und im Code dokumentierte Komponente (5 Punkte)
Zusatzpunkte für hohe Wartbarkeitsindizes (über 70 bzw. über 80; 2–5 Punkte)
Dokumentation des Praxisbeispiels (10 Punkte)
Zusätzliche Punkte für Excel-Import/Export (je 3 Punkte)

Architektur und Projektstruktur

Ordnerstruktur

Models:
Enthält alle Datenmodelle:
User.cs (und BaseUser.cs)
Product.cs
CustomerOrder.cs
OrderItem.cs

Services:
Implementieren die Geschäftslogik und Benutzerinteraktion:

- **AuthenticationService.cs:**
  Verantwortlich für Login und Registrierung. Hier wird auch die E-Mail-Validierung mithilfe von EmailValidation.cs angewandt.
- **InventoryService.cs:**
  Zuständig für die Verwaltung der Produkte (CRUD, Lagerverwaltung, Import/Export).
- **ShoppingService.cs:**
  Kümmert sich um den Bestellvorgang und den Einkaufsprozess (Warenkorb, Checkout).
- **OrderService.cs:**
  Erzeugt Bestellungen und verarbeitet den Kaufvorgang (inklusive Validierung des Bestelldatums).
- **MainMenuService.cs:**
  Das zentrale Menü, das den Login/Registrierungsprozess (via AuthenticationService) steuert und den Übergang in die Einkaufs- und Lagerverwaltungsfunktionen ermöglicht.
- **ImportExportService.cs:**
  Spezialisierte Logik für den Import und Export von Produktdaten aus Excel und Textdateien.

Framework/Controls (oder Utilities):
Enthält allgemeine Hilfsklassen und Validierungsregeln:

- **Validation.cs:**
  Stellt generische Validierungsregeln bereit, u.a. NotEmptyRule, RangeRule und eine DateRangeRule (prüft, ob das Datum zwischen 01.01.2020 und DateTime.Now liegt).
- **EmailValidation.cs:**
  Enthält eine statische Methode zur Validierung von E-Mail-Adressen mittels Regex.

Data:

- **AppDbContext.cs:**
  EF Core DbContext zur Konfiguration und Verwaltung der Datenbank.
  Weitere Klassen für Migrations (z. B. FixDatabaseSchema) und zur Unterstützung von Excel-Import/Export:
  - **ExcelExport.cs**
  - **ExcelImport.cs**
  - **ListExport.cs**
  - **ListImport.cs**

Program.cs:
Der Einstiegspunkt der Anwendung, in dem der Host aufgebaut, DI konfiguriert und das Hauptmenü gestartet wird.

Beschreibung der einzelnen Komponenten

AuthenticationService

Aufgabe:
Zuständig für den Login- und Registrierungsprozess.

Funktionen:
Login(): Liest E-Mail und Passwort von der Konsole, validiert den Benutzer gegen die Datenbank und gibt den entsprechenden Benutzer zurück.

RegisterNewUser():
Fragt Benutzerinformationen ab, validiert insbesondere die E-Mail-Adresse mithilfe der in EmailValidation.cs definierten Logik, und registriert einen neuen Benutzer in der Datenbank.

Besonderheiten:
Die E-Mail wird so lange abgefragt, bis sie dem gewünschten Format entspricht.

InventoryService

Aufgabe:
Verwaltung der Produkte und Lagerverwaltung.

Funktionen:
Produkte hinzufügen, suchen, Bestand aktualisieren
Import und Export von Produkten (Excel- und Textformate; ProductNumber wird dabei ignoriert, da die Datenbank den Wert generiert)

GetProducts():
Gibt alle Produkte aus der Datenbank zurück.

ImportExportService

Aufgabe:
Spezialisierte Logik zum Importieren und Exportieren von Produktdaten.

Funktionen:
ExportProducts():
Exportiert die Produktliste in Excel oder Textdateien, wobei ein Dateiname anhand des aktuellen Datums erzeugt wird.

ImportProducts():
Liest Produkte aus einer Excel- oder Textdatei ein, und aktualisiert oder fügt sie in die Datenbank ein (Vermeidung von Duplikaten anhand des Produktnamens).

ShoppingService und OrderService

ShoppingService:
Implementiert einen Warenkorb und ermöglicht es dem Benutzer, Produkte zu durchsuchen, hinzuzufügen und den Kauf abzuschliessen.
OrderService:
Führt den finalen Bestellvorgang durch. Dabei wird das Bestelldatum (DateTime.Now) gesetzt und mittels einer DateRangeRule (aus Validation.cs) validiert, sodass es nicht vor dem 01.01.2020 und nicht in der Zukunft liegt.

MainMenuService

Aufgabe:
Zentrales Navigationsmenü.

Funktionen:
Zeigt das Hauptmenü mit Optionen für Einloggen, Registrieren und Programm beenden.
Ruft den AuthenticationService für Login/Registrierung auf.

Nach erfolgreichem Login wird ein Benutzer-Menü angezeigt, in dem der Benutzer zwischen Einkauf, Lagerverwaltung oder Abmeldung wählen kann.

Validierung (Validation.cs und EmailValidation.cs)

Validation.cs:
Enthält generische Validierungsregeln:

- NotEmptyRule: Überprüft, ob ein String nicht leer ist.
- RangeRule: Überprüft, ob ein int-Wert in einem bestimmten Bereich liegt.
- DateRangeRule: Stellt sicher, dass ein Datum zwischen dem 01.01.2020 und dem aktuellen Datum liegt.

EmailValidation.cs:
Bietet eine statische Methode, um E-Mail-Adressen mittels Regex zu validieren.

Datenbank und AppDbContext

AppDbContext.cs:
Konfiguriert die Entitäten (User, Product, CustomerOrder, OrderItem) und deren Beziehungen.

Migrationen:
Die FixDatabaseSchema-Migration passt die Datenbankschemata an (z. B. Identity-Spalte bei ProductNumber, Unique Constraint auf UserEmail).

Excel- und Listen-Import/Export

ExcelExport.cs und ExcelImport.cs:
Ermöglichen den Export bzw. Import von Produktdaten in Excel-Dateien.

ListExport.cs und ListImport.cs:
Ermöglichen den Export bzw. Import von Produktdaten in Textdateien (mit Trennzeichen).

Installation, Konfiguration und Ausführung

Voraussetzungen

.NET SDK:

Stellt sicher, dass das .NET 8.0 SDK installiert ist.

Datenbank:

Eine Instanz von SQL Server (z. B. LocalDB) – der Connection String wird in appsettings.json konfiguriert.

NuGet-Pakete für Datenverarbeitung:

- Microsoft.EntityFrameworkCore und Microsoft.EntityFrameworkCore.SqlServer
- EPPlus (für Excel-Operationen)

Einrichtung

Projekt einrichten:

Klone oder lade das neue .NET 8 Projekt in Visual Studio.

Stelle sicher, dass die neue Projektdatei (csproj) den TargetFramework-Wert net8.0 enthält.

Pakete installieren:

Über den NuGet-Paketmanager oder die .NET-CLI werden alle notwendigen Pakete installiert.

Folgende Pakete müssen installiert sein:

- Microsoft.EntityFrameworkCore
- Microsoft.EntityFrameworkCore.SqlServer
- Microsoft.Extensions.Hosting
- Microsoft.Extensions.Configuration
- EPPlus (für Excel-Operationen)

Überprüfe in der Projektdatei (.csproj), ob die Referenzen korrekt vorhanden sind.

Konfiguration:

Passe die Datei appsettings.json an, insbesondere den Connection String, damit die Datenbank korrekt verbunden wird.

Build:

Führe einen Clean und anschliessend einen Rebuild der Lösung durch, um sicherzustellen, dass alle Pakete korrekt eingebunden sind und der Code fehlerfrei kompiliert.

Migrationen:

Stelle sicher, dass die Datenbank mithilfe von EF Core-Migrationen aktualisiert wird.
Führe z. B. über die .NET-CLI oder den Package Manager Console den Befehl dotnet ef database update aus, um alle Migrationen (z. B. FixDatabaseSchema) anzuwenden.Ausführung:

Starte das Projekt in der Konsole (F5 oder Ctrl+F5).

Nutzung

Beim Start:

Das Hauptmenü erscheint, in dem der Benutzer sich einloggen, registrieren oder das Programm beenden kann.

Login/Registrierung:

Über den AuthenticationService werden E-Mail und Passwort abgefragt. Bei der Registrierung wird die E-Mail über EmailValidation geprüft.

Benutzer-Menü:

Nach erfolgreichem Login kann der Benutzer Produkte durchsuchen, einkaufen oder zur Lagerverwaltung wechseln.

Lagerverwaltung:

Über den InventoryService können Produkte hinzugefügt, gesucht, aktualisiert sowie importiert/exportiert werden.

Einkauf/Bestellvorgang:

Über ShoppingService und OrderService wird der Bestellvorgang durchgeführt, wobei das Bestelldatum validiert wird (DateRangeRule).

Ablauf des Codes (Sequenzübersicht)

Programmstart (Program.cs):
Initialisierung von EPPlus (LicenseContext)
Aufbau des DI-Containers und Erstellen des Hosts
Sicherstellen, dass die Datenbank existiert (AppDbContext)
Aufruf des Hauptmenüs (MainMenuService.DisplayMainMenu())

Hauptmenü (MainMenuService):
Anzeige der Optionen „Einloggen“, „Registrieren“, „Programm beenden“

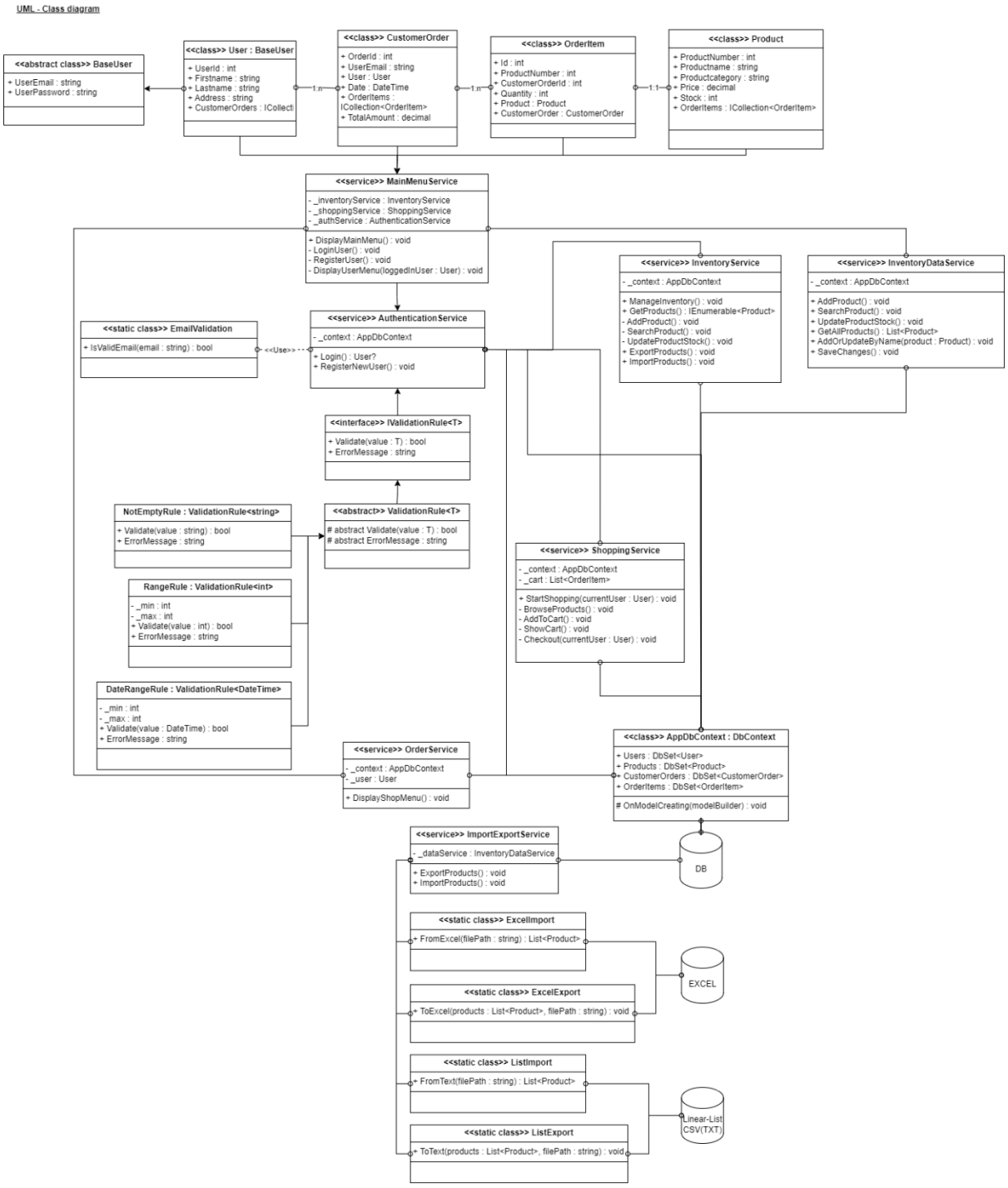
Bei Auswahl von „Einloggen“:
Aufruf von AuthenticationService.Login()
Bei Erfolg: Übergang ins Benutzer-Menü (ohne doppelte Begrüssung, da AuthenticationService bereits grüsst, wenn gewünscht)

Bei Auswahl von „Registrieren“:
Aufruf von AuthenticationService.RegisterNewUser()

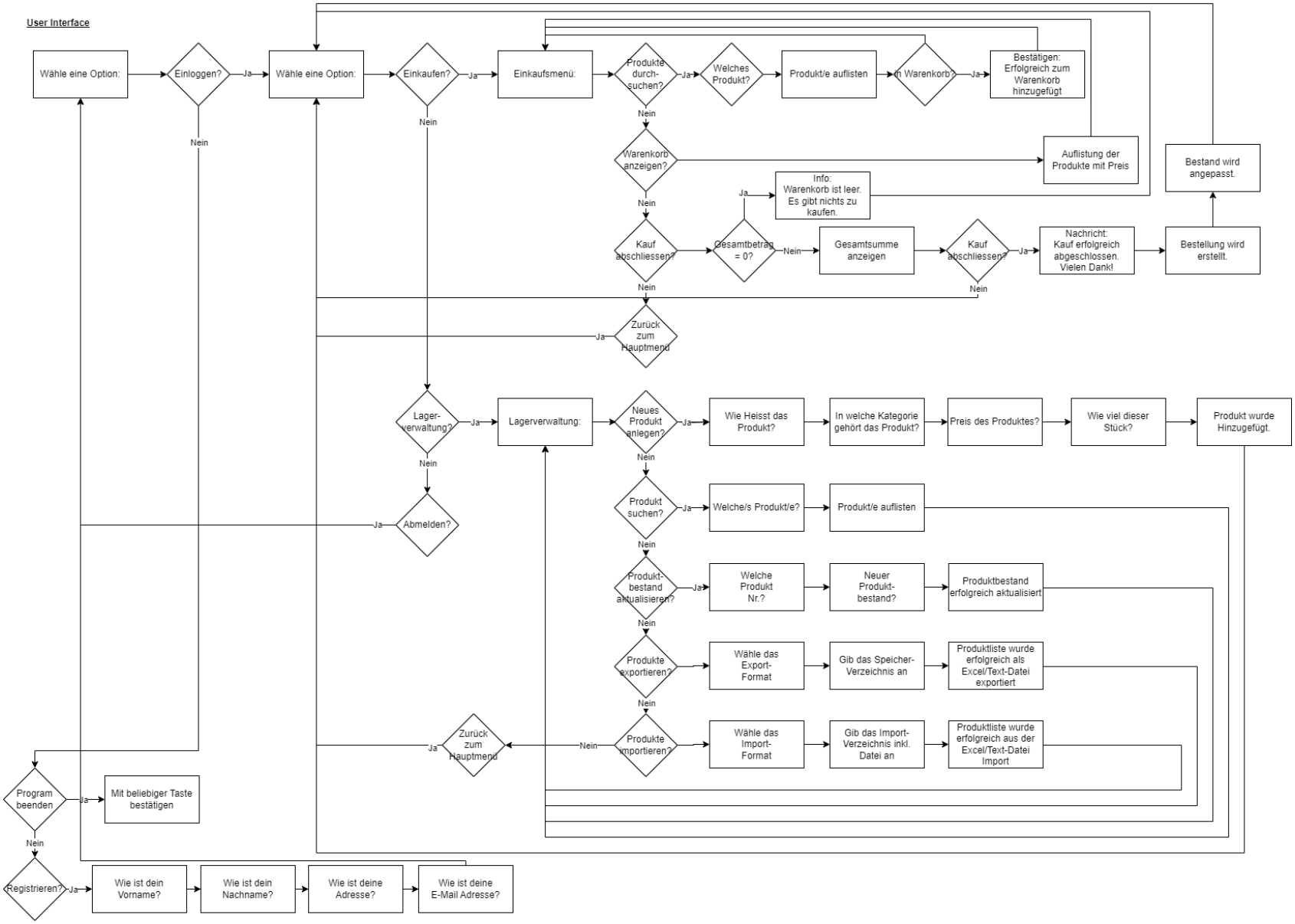
Benutzer-Menü:
Anzeige der Produktliste (über InventoryService.GetProducts())
Optionen für Einkaufen, Lagerverwaltung oder Abmelden
Bei „Einkaufen“: Start des Einkaufsprozesses über ShoppingService.StartShopping()
Bei „Lagerverwaltung“: Aufruf von InventoryService.ManageInventory()
Einkauf/Bestellvorgang (ShoppingService und OrderService):
Durchsuchen der Produkte
Hinzufügen von Produkten zum Warenkorb
Abschluss des Kaufs (Checkout), bei dem das Bestelldatum (DateTime.Now) mithilfe der DateRangeRule validiert wird

Import/Export:
Produkte können über Excel- oder Textdateien exportiert und importiert werden (ImportExportService und unterstützende Klassen: ExcelExport, ExcelImport, ListExport, ListImport).

Klassendiagramm



User Interface



Besonderheiten und Validierung

E-Mail-Validierung:

Die Klasse EmailValidation (im Framework/Controls) stellt sicher, dass eingegebene E-Mail-Adressen einem typischen Format entsprechen.

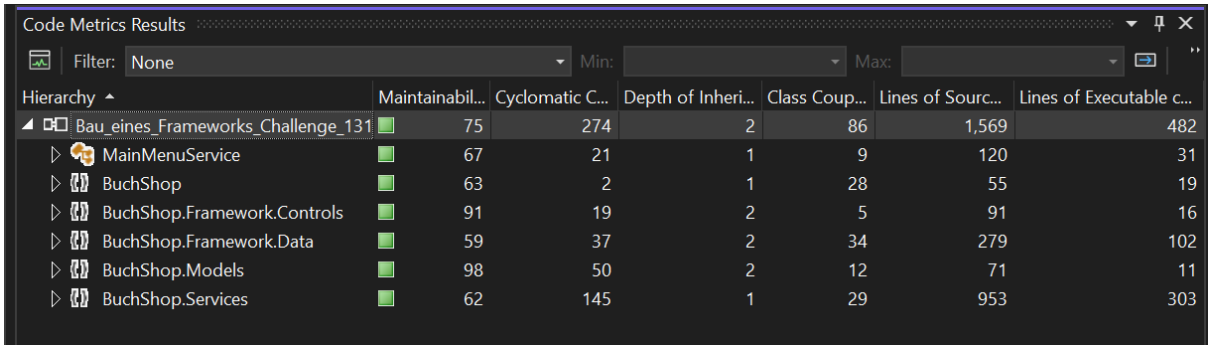
Datumvalidierung:

Über die DateRangeRule in Validation.cs wird sichergestellt, dass das Bestelldatum zwischen dem 01.01.2020 und dem aktuellen Datum liegt.

Wartbarkeit:

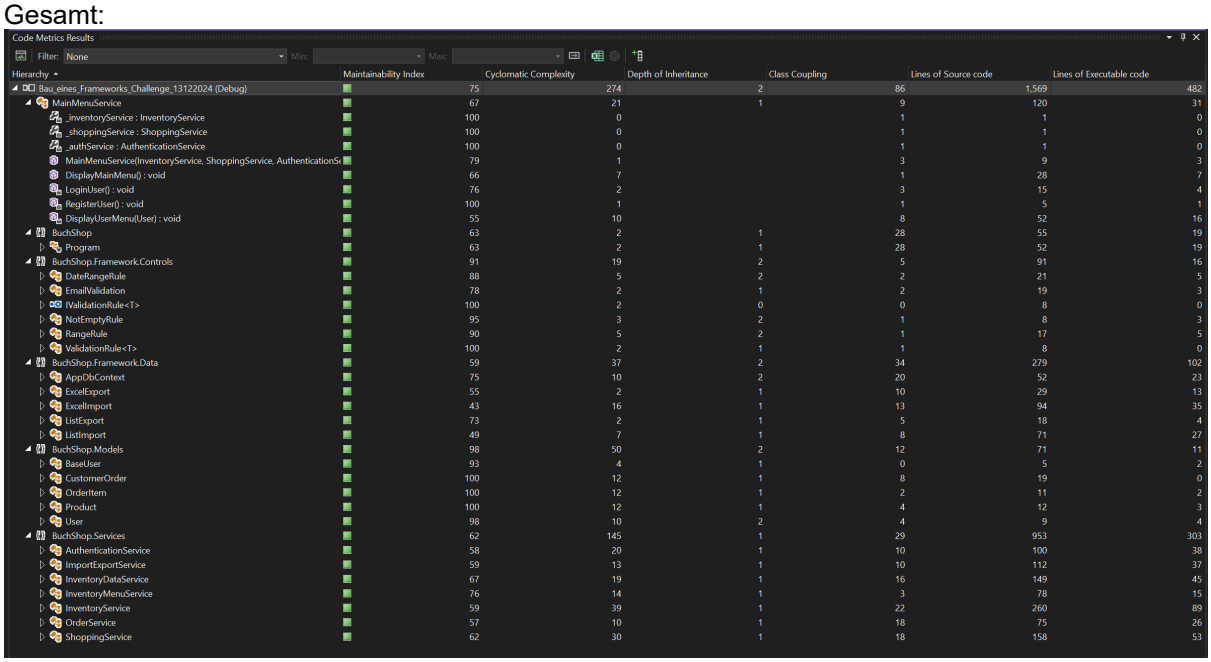
Durch klare Trennung der Verantwortlichkeiten (Services, Models, Framework/Controls) und Einsatz von Dependency Injection ist das Projekt modular aufgebaut – was den Wartbarkeitsindex erhöht und zukünftige Erweiterungen erleichtert.

Wir haben einen Warbarkeitsindex von 75 Punkten erreicht, würden wir uns in der von dem Entity Framework trennen, sobald wir den App Launch hätten würden wir die Punkte über 80 bestimmt erreichen.



Hierarchy	Maintainabil...	Cyclomatic C...	Depth of Inheri...	Class Coup...	Lines of Sourc...	Lines of Executable c...
Bau_eines_Frameworks_Challenge_131	75	274	2	86	1,569	482
MainMenuService	67	21	1	9	120	31
BuchShop	63	2	1	28	55	19
BuchShop.Framework.Controls	91	19	2	5	91	16
BuchShop.Framework.Data	59	37	2	34	279	102
BuchShop.Models	98	50	2	12	71	11
BuchShop.Services	62	145	1	29	953	303

Gesamt:



Hierarchy	Maintainability Index	Cyclomatic Complexity	Depth of Inheritance	Class Coupling	Lines of Source code	Lines of Executable code
Bau_eines_Frameworks_Challenge_13122024 (Debug)	75	274	2	86	1,569	482
MainMenuService	67	21	1	9	120	31
_inventoryService : InventoryService	100	0		1	1	0
_shoppingService : ShoppingService	100	0		1	1	0
_authService : AuthenticationService	100	0		1	1	0
MainMenuService(InventoryService, ShoppingService, AuthenticationS	79	1		3	9	3
DisplayMainMenu() : void	66	7		1	28	7
LoginUser() : void	76	2		3	15	4
RegisterUser() : void	100	1		1	5	1
DisplayUserMenu(User) : void	55	10		8	52	16
BuchShop	63	2	1	28	55	19
Program	63	2	1	28	52	19
BuchShop.Framework.Controls	91	19	2	5	91	16
DateRangeRule	88	5	2	2	21	5
EmailValidation	78	2	1	2	19	3
IValidationRule<T>	100	2	0	0	8	0
NotEmptyRule	95	3	2	1	8	3
RangeRule	90	5	2	1	17	5
ValidationRule<T>	100	2	1	1	8	0
BuchShop.Framework.Data	59	37	2	34	279	102
AppDbContext	75	10	2	20	52	23
ExcelExport	55	2	1	10	29	13
ExcelImport	43	16	1	13	94	35
ListExport	73	2	1	5	18	4
ListImport	49	7	1	8	71	27
BuchShop.Models	98	50	2	12	71	11
BaseUser	93	4	1	0	5	2
CustomerOrder	100	12	1	8	19	0
OrderItem	100	12	1	2	11	2
Product	100	12	1	4	12	3
User	98	10	2	4	9	4
BuchShop.Services	62	145	1	29	953	303
AuthenticationService	58	20	1	10	100	38
ImportExportService	59	13	1	10	112	37
InventoryDataService	67	19	1	16	149	45
InventoryMenuService	76	14	1	3	78	15
InventoryService	59	39	1	22	260	89
OrderService	57	10	1	18	75	26
ShoppingService	62	30	1	18	158	53

Zusammenfassung Buchshop Projekt

Dieses Projekt demonstriert den Bau eines eigenen Frameworks, das wiederverwendbare Komponenten für einen einfachen Buchshop bereitstellt. Wichtige Aspekte sind:

Modulare Architektur:

Die Trennung in Models, Services, Data und Framework/Controls sorgt für klare Verantwortlichkeiten.

Validierung:

E-Mail-Adressen und Bestelldaten werden über spezialisierte Klassen validiert.

Datenbankzugriff:

EF Core (AppDbContext) wird genutzt, um die Entitäten (User, Product, CustomerOrder, OrderItem) zu verwalten.

Import/Export:

Excel- und Textdateien ermöglichen das Ein- und Auslesen von Produktdaten, was als zusätzliche Komponente im Rahmen der Challenge gewertet wird.

Anwendungsfluss:

Das Programm startet über Program.cs, führt den Benutzer über ein Hauptmenü (MainMenuService) in den Login/Registrierungsprozess und leitet nach erfolgreichem Login in den Einkauf und die Lagerverwaltung.

Dokumentation der Framework-Komponenten

In diesem Kapitel werden die **Framework-Klassen** vorgestellt, die im Ordner *Framework* unterteilt in **Controls** und **Data** liegen. Diese Klassen sind so konzipiert, dass sie in **anderen Projekten** problemlos wiederverwendet werden können. Nachfolgend findest du eine Beschreibung des Zwecks, der Methoden und eine kurze Anleitung zur Einbindung in fremde Projekte.

Framework/Controls

EmailValidation.cs

Namespace: BuchShop.Framework.Controls
Klasse: public static class EmailValidation

Aufgabe:

Stellt eine statische Methode bereit, um E-Mail-Adressen auf ein typisches Format zu prüfen.

Wichtige Mitglieder:

public static bool IsValidEmail(string email):

Verwendet einen einfachen Regex, um zu überprüfen, ob email ein valides Format besitzt.

Gibt true zurück, wenn das Format gültig ist, ansonsten false.

Verwendung in einem anderen Projekt:

Kopiere die Datei EmailValidation.cs in deinen Projektordner (z. B. in einen Ordner Framework/Controls). Achte darauf, dass der Namespace (BuchShop.Framework.Controls) korrekt in deinen Quellcode importiert wird:

```
using BuchShop.Framework.Controls;
```

Rufe die statische Methode wie folgt auf:

```
if (EmailValidation.IsValidEmail("test@example.com"))
   {
      // E-Mail ist gültig
   }
   else
   {
      // E-Mail ist ungültig
   }
```

Validation.cs

Namespace: BuchShop.Framework.Controls
Datei: Validation.cs

Aufgabe:

Enthält generische Validierungsregeln sowie ein Interface und eine abstrakte Basisklasse, um eigene Regeln zu definieren.

Wichtige Mitglieder:

Interface:

```
public interface IValidationRule<T>
{
    bool Validate(T value);
    string ErrorMessage { get; }
}
```

Ermöglicht es, beliebige Validierungsregeln zu definieren, die einen Wert vom Typ T prüfen.

Abstrakte Basisklasse:

```
public abstract class ValidationRule<T> : IValidationRule<T>
    {
        public abstract bool Validate(T value);
        public abstract string ErrorMessage { get; }
    }
```

Stellt die Grundstruktur einer Validierungsregel bereit.

Konkrete Regeln:

- **NotEmptyRule:**
  Prüft, ob ein string nicht leer oder nur Whitespace ist.
- **RangeRule:**
  Prüft, ob ein int innerhalb eines Zahlenbereichs [min, max] liegt.
- **DateRangeRule:**
  Stellt sicher, dass ein DateTime zwischen dem 01.01.2020 und dem aktuellen Datum liegt.

Verwendung in einem anderen Projekt:

Kopiere Validation.cs in deinen Ordner (z. B. Framework/Controls).
Importiere den Namespace:

```
using BuchShop.Framework.Controls;
```

Definiere und nutze eine Regel, zum Beispiel:

```
var rule = new NotEmptyRule();
if (!rule.Validate(userInput))
{
    Console.WriteLine(rule.ErrorMessage);
}
```

Eigene Regeln kannst du erstellen, indem du die ValidationRule<T> erbst:

```
public class CustomRule : ValidationRule<string>
    {
        public override bool Validate(string value)
        {
            return value.Contains("X");
        }
        public override string ErrorMessage => "Der Wert muss ein 'X' enthalten.";
    }
```

Framework/Data

AppDbContext.cs

Namespace: BuchShop.Framework.Data
Klasse: public class AppDbContext : DbContext

Aufgabe:

Dient als EF Core-DbContext, verwaltet Tabellen (DbSet) für User, Product, CustomerOrder und OrderItem. Stellt in OnModelCreating Konfigurationen für Beziehungen und Spalten bereit.

Wichtige Mitglieder:

```
public DbSet<User> Users { get; set; }
public DbSet<Product> Products { get; set; }
public DbSet<CustomerOrder> CustomerOrders { get; set; }
public DbSet<OrderItem> OrderItems { get; set; }
```

Verwendung in einem anderen Projekt:

Kopiere AppDbContext.cs in deinen Ordner (z. B. Framework/Data).
Passe den Namespace an deine Struktur an.
Registriere den Kontext in Program.cs (oder in deinem DI-Setup):

```
services.AddDbContext<AppDbContext>(options =>
options.UseSqlServer(connectionString));
```

Nutze den Kontext in deinen Services:

```
public class MyService
{
        private readonly AppDbContext _context;
        public MyService(AppDbContext context)
    {
      context = context;
    }
    // ...
}
```

Erstelle ggf. EF-Migrationen und führe sie aus (dotnet ef migrations add ... → dotnet ef database update).

ExcelExport.cs

Namespace: BuchShop.Framework.Data
Klasse: public static class ExcelExport

Aufgabe:

Ermöglicht das Exportieren einer Liste von Product-Objekten in eine Excel-Datei (über EPPlus).

Wichtige Mitglieder:

public static void ToExcel(List<Product> products, string filePath):
Erzeugt ein Worksheet namens „Produkte“.
Schreibt in Spalten: Name, Kategorie, Preis, Bestand.
Speichert das Ergebnis in filePath.

Verwendung in einem anderen Projekt:

Stelle sicher, dass EPPlus per NuGet installiert ist.
Kopiere ExcelExport.cs in dein Projekt (z. B. Framework/Data).
Importiere den Namespace:

```
using BuchShop.Framework.Data;
```

Rufe die Methode auf:

```
ExcelExport.ToExcel(meineProduktListe, "C:\\Temp\\products.xlsx");
```

ExcelImport.cs

Namespace: BuchShop.Framework.Data
Klasse: public static class ExcelImport

Aufgabe:

Liest Produktdaten aus einer Excel-Datei (über EPPlus) und erzeugt eine List<Product>.

Wichtige Mitglieder:

public static List<Product> FromExcel(string filePath):
Liest das erste Worksheet.
Erwartet Spalten: Name, Kategorie, Preis, Bestand (ohne ProductNumber).
Gibt eine Liste von Product zurück.

Verwendung in einem anderen Projekt:

EPPlus installieren.
Datei ExcelImport.cs ins Projekt kopieren, Namespace importieren:

```
using BuchShop.Framework.Data;
```

Verwenden:

```
List<Product> importedProducts = ExcelImport.FromExcel("C:\\Temp\\products.xlsx");
```

ListExport.cs

Namespace: BuchShop.Framework.Data
Klasse: public static class ListExport

Aufgabe:

Exportiert Produktdaten (Liste von Product) in eine Textdatei (z. B. CSV-ähnlich mit ;-Trennzeichen).

Wichtige Mitglieder:

public static void ToText(List<Product> products, string filePath):

Schreibt pro Zeile: productName;productCategory;price;stock.

Verwendung in einem anderen Projekt:

Kopiere ListExport.cs in dein Projekt.
Namespace importieren:

```
using BuchShop.Framework.Data;
```

Aufruf:

```
ListExport.ToText(meineProduktListe, "C:\\Temp\\products.txt");
```

ListImport.cs

Namespace: BuchShop.Framework.Data
Klasse: public static class ListImport

Aufgabe:

Liest Produktdaten aus einer Textdatei ein, in der pro Zeile vier Felder stehen:
name;category;price;stock.

Wichtige Mitglieder:

public static List<Product> FromText(string filePath):

Liest alle Zeilen der Datei.

Erwartet vier Felder pro Zeile.

Parsen von price und stock.

Gibt eine List<Product> zurück.

Verwendung in einem anderen Projekt:

Kopiere ListImport.cs in dein Projekt.
Namespace importieren:

```
using BuchShop.Framework.Data;
```

Aufruf:

```
List<Product> products = ListImport.FromText("C:\\Temp\\products.txt");
```

Allgemeine Hinweise zur Integration in andere Projekte

Namespaces anpassen:

Wenn du die Klassen in andere Projekte kopierst, solltest du den Namespace entsprechend deiner Projektstruktur anpassen (z. B. MyCompany.Framework.Data statt BuchShop.Framework.Data).

Abhängigkeiten:

EPPlus:
Für ExcelExport und ExcelImport muss EPPlus installiert sein.

EF Core:
AppDbContext erfordert Microsoft.EntityFrameworkCore (und ggf. SqlServer oder ein anderes Datenbankprovider-Paket).

System.IO:
Für ListExport und ListImport benötigst du Standard-Bibliotheken (sollte in .NET integriert sein).

Konfiguration:

Bei AppDbContext musst du den Connection String in deiner Program.cs oder appsettings.json verwalten und den Kontext über Dependency Injection registrieren.

Bei den statischen Klassen (ExcelExport, ExcelImport, etc.) ist keine DI-Registrierung nötig, da sie statische Methoden anbieten.

Fazit

Die Framework-Klassen im Ordner Framework/Controls und Framework/Data bieten dir:
Validierung (E-Mail und generische Regeln)
EF Core-Datenbankkontext
Excel-Import/Export
Listen-Import/Export

Sie sind modular aufgebaut, sodass du sie in jedem beliebigen .NET-Projekt wiederverwenden kannst. Achte lediglich auf die notwendigen NuGet-Abhängigkeiten (EPPlus, EF Core) und passe bei Bedarf die Namespaces an deine Projektstruktur an.