

Giacomo Graef



---

# DOKUMENTATION

---

Projekt: Elektrogroßhandel Graef



9. DEZEMBER 2025

GIACOMO GRAEF

<https://github.com/Plessu/Elektrgrosshandel>

# Inhaltsverzeichnis

1.	Projekteinführung.....	2
1.1.	Kurzbeschreibung.....	2
1.2.	Technische Daten.....	2
1.3.	NuGet Vorstellung .....	3
1.3.1.	Spectre Console.....	3
1.3.2.	Newtonsoft Json.....	3
2.	Technische Details .....	4
2.1.	Übersicht Klassen .....	4
Namespace: Elektrogrosshandel.....	4	
Namespace: Elektrogrosshandel.GUI .....	4	
Namespace: Elektrogrosshandel.GUI.GUI_Menus .....	5	
Namespace: Elektrogrosshandel.Hardware .....	5	
Namespace: Elektrogrosshandel.Functions .....	7	
Namespace: Elektrogrosshandel.User .....	8	
Sonstige Dateien / Dokumente (relevant für Kontext).....	8	
2.2.	Technische Erklärung.....	9
2.2.1.	Anzeige UI .....	9
2.2.2.	Login und User Registrierung .....	11
2.2.3.	Hardware Klasse .....	13
2.2.4.	Speichern und Laden .....	15
2.2.5.	Übersicht Shop Führung.....	16
2.3.	Schwierigkeiten bei der Umsetzung.....	17
3.	Verwendung von KI .....	17
4.	Weiterführende Links und Dateien .....	18

# 1. Projekteinführung

## 1.1. Kurzbeschreibung

Die Idee bestand darin für die fiktive Firma Graef Elektrogroßhandel eine App zur Abwicklung von Kundenbestellungen zu erzeugen ähnlich Amazon.

- Ziel: Verwaltung und Bereitstellung technischer Informationen zu Computer Hardware für einen Großhandel
- Zielgruppe: Entwickler, Integratoren, Tester und technische Redakteure



## 1.2. Technische Daten

- Plattform: `.NET 10` , Visual Studio (IDE mit integriertem Editor, Test-Runner, Output-Pane und Terminal).
- NuGet Erweiterungen:
  - Spectre.Console -> Zur Darstellung der UI, Markup Sprache für Konsolen Apps
  - Newtonsoft JSON -> Zur Speicherung der Datenbank als JSON Datei

## 1.3. NuGet Vorstellung

### 1.3.1. Spectre Console

---

**Spectre.Console** ist ein mächtiges .NET-Framework, um ansprechende und interaktive Konsolenanwendungen zu erstellen, das Funktionen wie Farbgebung mit Markup, Tabellen, Bäume, Fortschrittsbalken und Benutzer-Prompts für C# bietet, wodurch das Aussehen und die Funktionalität klassischer Konsolen-Apps enorm verbessert wird und die Erstellung moderner CLI-Tools vereinfacht

---

Die Spectre.Console wird in dem Projekt dafür verwendet das Layout über alle Menüpunkte hinweg einheitlich und übersichtlich zu halten.

Für genaue Beispiele bei der genauen Erklärung der Klassen schauen.

### 1.3.2. Newtonsoft Json

---

Newtonsoft JSON (auch bekannt als Json.NET) ist ein leistungsstarkes, kostenloses Open-Source-Framework für .NET, das die einfache Serialisierung und Deserialisierung von .NET-Objekten in das JSON-Format (JavaScript Object Notation) und umgekehrt ermöglicht, mit nützlichen Funktionen wie LINQ-Abfragen für JSON, Unterstützung für verschiedene Typen und einfache Handhabung komplexer Datenstrukturen, obwohl die aktive Entwicklung zugunsten von System.Text.Json eingestellt wurde. Es ist eine der am häufigsten heruntergeladenen .NET-Bibliotheken und wird in vielen Anwendungen genutzt, um Daten effizient auszutauschen.

---

Eigentlich war geplant die System.JSON Version zu benutzen. Dies stellte sich durch die Verwendung von privaten Feldern als äußerst schwierig heraus und nach intensivem Testen stellte sich heraus das die zwar etwas ältere aber sehr stabile Newtonsoft JSON das Problem deutlich eleganter und einfacher lösen kann.

Für genaue Beispiele bei der genauen Erklärung der Klassen schauen.

## 2. Technische Details

### 2.1. Übersicht Klassen

#### Namespace: Elektrogrosshandel

- Program
  - Accessibility: internal class
  - File: VSCode/Elektrgrosshandel/Elektrgrosshandel/Program.cs
  - Beschreibung: Hauptklasse / Einstiegspunkt (Main). Lädt Testdaten, Hardware- und Account-Storage, hält statische Referenz Program.ActiveUser und startet den Login-Flow.
  - Verweist auf (Referenzen, Definitionen in Suchergebnissen nicht gefunden): Account, TestData, HardWareStorage, AccountStorage, LogIn.

#### Namespace: Elektrogrosshandel.GUI

- GUI\_Display
  - Accessibility: internal class
  - File: VSCode/Elektrgrosshandel/Elektrgrosshandel/GUI/GUI\_Display.cs
  - Beschreibung: Erzeugt und zeichnet das Anzeigen-Layout mit Spectre.Console (Header, Body, Footer). Hilfsmethoden zur Anzeige von Layout-Objekten.
  - Verweist auf: GUI\_Theme, Account, Program.
- GUI\_Theme
  - Accessibility: internal static class
  - File: VSCode/Elektrgrosshandel/Elektrgrosshandel/GUI/GUI\_Theme.cs
  - Beschreibung: Farbcodedefinitionen und Helfer (CreatePanelForRows, CreateMenuPanel, CreateInfoPanel) zur einheitlichen Erstellung von Panels/Markup.

## Namespace: Elektrogrosshandel.GUI.GUI\_Menus

- GUI\_AccountInfoMenu
  - Accessibility: internal class
  - File:  
VSCode/Elektrgrosshandel/Elektrgrosshandel/GUI/GUI\_Menus/GUI\_AccountInfoMenu.cs
  - Beschreibung: GUI-Layout für das Account-Info-Menü (Menu-Panel, Display-Panel). Nutzt Account-Edit-Helper zum Aufbau der Anzeige.
  - Verweist auf: AccountEdit, Program.ActiveUser.

(Hinweis: in den Suchergebnissen wurden mehrere GUI-Menüklassen referenziert, z. B. GUI\_Login, GUI\_PlaceOrder, GUI\_AdminMenuAddArticle, GUI\_ProductCatalogCategories etc., deren Implementierungen in den angezeigten Ergebnissen nicht enthalten waren.)

## Namespace: Elektrogrosshandel.Hardware

- Case
  - Accessibility: internal class
  - File: VSCode/Elektrgrosshandel/Elektrgrosshandel/Hardware/100\_Case.cs
  - Beschreibung: Erbt von ComputerHardware; speichert case-spezifische Eigenschaften (FormFactor, FanSlots, FrontPanelPorts). Registriert sich bei ComputerHardware.AddCase. Erzeugt Artikel-IDs.
- Motherboard
  - Accessibility: internal class
  - File:  
VSCode/Elektrgrosshandel/Elektrgrosshandel/Hardware/200\_Motherboard.cs
  - Beschreibung: Erbt von ComputerHardware; Eigenschaften wie Socket, RamType, FormFactor, PCIeVersion, StorageInterfaces. Registriert sich bei ComputerHardware.AddMotherboard und erzeugt Artikel-IDs.
- PowerSupply
  - Accessibility: internal class
  - File:  
VSCode/Elektrgrosshandel/Elektrgrosshandel/Hardware/300\_PowerSupply.cs
  - Beschreibung: Erbt von ComputerHardware; spezifische Eigenschaft Wattage; ruft ComputerHardware.AddPowerSupply. (Datei enthält Plan/Pseudocode + Implementierungsteil.)
- GraphicsCard
  - Accessibility: internal class
  - File:  
VSCode/Elektrgrosshandel/Elektrgrosshandel/Hardware/500\_GraphicsCard.cs
  - Beschreibung: Erbt von ComputerHardware; enthält VRAM, MemoryType, Taktwerte, TDP, Outputs, PowerConsumption, etc. Registriert sich bei ComputerHardware (AddGraphicsCard). Datei enthält ausführlichen Pseudocode/Implementierung.

- Ram
  - Accessibility: internal class
  - File: VSCode/Elektrorrosshandel/Elektrorrosshandel/Hardware/600\_Ram.cs
  - Beschreibung: Erbt von ComputerHardware; RAM-spezifische Felder (Kapazität, Frequenz, Typ, Module, ECC). Registriert sich bei ComputerHardware.AddRAM und erzeugt Artikel-IDs.
- StorageDevice
  - Accessibility: internal class
  - File:
    - VSCode/Elektrorrosshandel/Elektrorrosshandel/Hardware/700\_StorageDevices.cs
  - Beschreibung: Erbt von ComputerHardware; Felder: Kapazität, Typ (SSD/HDD), Interface, Read/Write-Speed, FormFactor. Datei enthält Pseudocode und Implementierungsskizze.
- CoolingSystem
  - Accessibility: internal class
  - File:
    - VSCode/Elektrorrosshandel/Elektrorrosshandel/Hardware/800\_CoolingSystem.cs
  - Beschreibung: Erbt von ComputerHardware; Felder wie FanSpeed, LiquidCooled, TDP, Kompatibilität. Registriert sich bei ComputerHardware.AddCoolingSystem. Enthält Pseudocode/Implementierung.
- Display
  - Accessibility: internal class
  - File: VSCode/Elektrorrosshandel/Elektrorrosshandel/Hardware/925\_Display.cs
  - Beschreibung: Erbt von ComputerHardware; Display-spezifische Felder (Resolution, RefreshRate, PanelType, Size, HDR, Ports, AdaptiveSync, Curved, Touch, Brightness, AspectRatio). Enthält Pseudocode + Implementierungsskizze.

### *Zusätzliche (referenzierte) Hardware-Komponenten*

- ComputerHardware (Basis-Klasse) — stark referenziert (z. B. ComputerHardware.AddCase, GetArticleByID, ArticleParentGroupID) — die konkrete Klasse/Datei war in den präsentierten Suchergebnissen nicht im vollen Quelltext enthalten; es gibt aber ein Dokument: VSCode/Elektrorrosshandel/Elektrorrosshandel/Hardware/\_TO\_ComputerHardware.md (Design/ToDo/Übersicht).
- HardWareStorage — mehrfach referenziert (LoadAllDevices/SaveAllDevices) — Implementierung nicht in den angezeigten Treffern.

## Namespace: Elektrogrosshandel.Functions

- PasswordHelper
  - Accessibility: internal class
  - File: VSCode/Elektrgrosshandel/Elektrgrosshandel/Functions/PasswordHelper.cs
  - Beschreibung: Passwort-Hashing (PBKDF2), Salt-Generierung, Hash-Vergleich, VerifyPassword(userName, password) ruft Account.GetAccountSalt/GetAccountPasswordHash.
- LogIn
  - Accessibility: internal class
  - File: VSCode/Elektrgrosshandel/Elektrgrosshandel/Functions/LogIn.cs
  - Beschreibung: Login-Flow (Eingabe Username/Password), Validierung via PasswordHelper, Setzen des Program.ActiveUser, Navigation zu MainMenu oder UserRegistration.
- UserInput
  - Accessibility: internal class
  - File: VSCode/Elektrgrosshandel/Elektrgrosshandel/Functions/UserInput.cs
  - Beschreibung: Helfer zur Eingabe via Spectre.Console (MenuChoice, GetStringInput, GetPasswordInput, GetIntInput).
- PlaceOrder
  - Accessibility: internal class
  - File: VSCode/Elektrgrosshandel/Elektrgrosshandel/Functions/PlaceOrder.cs
  - Beschreibung: Zeigt PlaceOrder-GUI, fragt Bestellnamen ab, erzeugt Order (Order.PlaceOrder) und fügt Order dem Account hinzu (Account.AddOrderToAccount). Sichert Hardware- und Account-Storage.
- AdminMenu
  - Accessibility: internal class
  - File: VSCode/Elektrgrosshandel/Elektrgrosshandel/Functions/AdminMenu.cs
  - Beschreibung: Admin-Menü-Logik (Add/Remove Artikel). Nutzt AddArticleFunctions und GUI-Menus.

- ProductCatalog
  - Accessibility: internal class
  - File: VSCode/Elektrorgrosshandel/Elektrorgrosshandel/Functions/ProductCatalog.cs
  - Beschreibung: Steuerung der Produktkatalog-Navigation (Kategorien, Hersteller). Nutzt ShopFunctions, GUI-Menu-Klassen.

Weitere in Functions referenzierte / erwartete Klassen (in Suchergebnissen nicht als volle Implementierung gefunden): Order, Account, AccountStorage, UserRegistration, MainMenu, ShopProductCategories, ShopProductManufacturer, diverse GUI\_\* Menüs.

## Namespace: Elektrogrosshandel.User

- Bucket
  - Accessibility: internal class
  - File: VSCode/Elektrorgrosshandel/Elektrorgrosshandel/User/Bucket.cs
  - Beschreibung: Repräsentiert einen Warenkorb (Bucket) mit ID, Wert, Name, Erstellzeit; Methoden: AddArticelToBucket, CreateBucket, GetBucketValue, ChangeBucketName, GetArticelsInBucket etc. Nutzt ComputerHardware zum Auflösen von Artikel-IDs.

## Sonstige Dateien / Dokumente (relevant für Kontext)

- VSCode/Elektrorgrosshandel/Elektrorgrosshandel/Hardware/\_TO\_ComputerHardware.md
  - Beschreibung: Design / ToDo / Empfehlungen rund um ComputerHardware, HardwareRegistry etc. Nützlich für Verständnis der Basis-Designentscheidungen.
- VSCode/Elektrorgrosshandel/Elektrorgrosshandel/Testcode.md
  - Beschreibung: Beispiel/Plan für Serialisierung von Account in JSON (DTO SerializableAccount). Enthält Pseudocode & Code-Snippets.

## 2.2. Technische Erklärung

### 2.2.1. Anzeige UI

Zur Darstellung wird Spectre.Console genutzt.

Dies ist eine Markup Sprache, mit der das Layout erzeugt wird. Zudem werden die Textelemente über Markups ausgegeben anstatt normaler Strings um sie ebenfalls optisch anpassen zu können.

#### *GUI.DisplayWindow*

```
//Create Layout and structure
Layout window = new Layout("Window")
    .SplitRows(
        new Layout("Header")
        .SplitColumns(
            new Layout("HeaderTitle"),
            new Layout("HeaderSubtitle")),
        new Layout("Body"),           //Bereitgestellt bei Aufruf
        new Layout("Footer"));

window["Window"].Size = 28;
window["Header"].Size = 3;
window["Footer"].Size = 3;
window["HeaderTitle"].Size = 35;

window["Body"].Update(Body).Size(bodyHeiht);
window["HeaderTitle"].Update(HeaderTitel);
window["HeaderSubtitle"].Update(HeaderSubtitle);

window["Footer"].Update(new Panel(
    new Markup("${Account.GetAccountNameByAccount(Program.ActiveUser)}")
    .Justify(Justify.Center)).Expand());

AnsiConsole.Clear();
AnsiConsole.Write(window);
```

Dabei übernimmt **GUI.Display.DisplayWindow** das Erstellen der finalen Ausgabe. Diese wird aufgerufen und ihr wird dabei eine der **GUI\_Menu.GUI\_\*** Dateien im GUI.GUI.Menu Namespace übergeben.

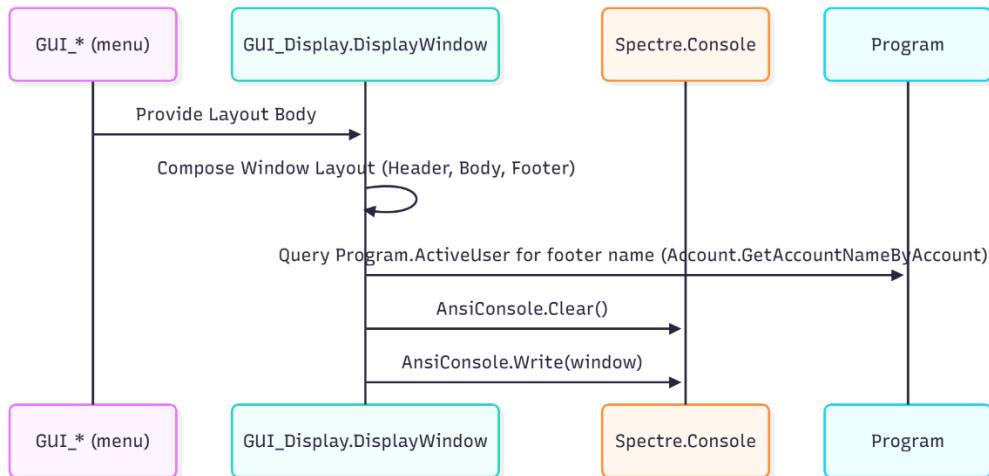
DisplayWindow verwendet nun die bereitgestellte Datei, um dieser einen Header und einen Footer hinzuzufügen und gibt das ganze dann an die Konsole aus.

Die Parameter für die Größe des Layouts werden eingestellt und dann die entsprechenden Bereiche geupdated.

Beim Erstellen der Panels können diesen ebenfalls Eigenschaften zugewiesen werden. Dazu gehören zum Beispiel der Style der Boarder und die Größe des Panel.

Der Footer zeigt dynamisch den Namen des gerade eingeloggten Accounts an.

Der Header übernimmt die Aufgabe der Logo Anzeige für das Unternehmen.



Zu jeder GUI\_Menu Datei gibt es die passende Klasse im Functions Namespace. Diese steuern die Logik und die Menüführung. Je nach dem was angezeigt werden soll übergeben sie zudem die erforderlichen Parameter.

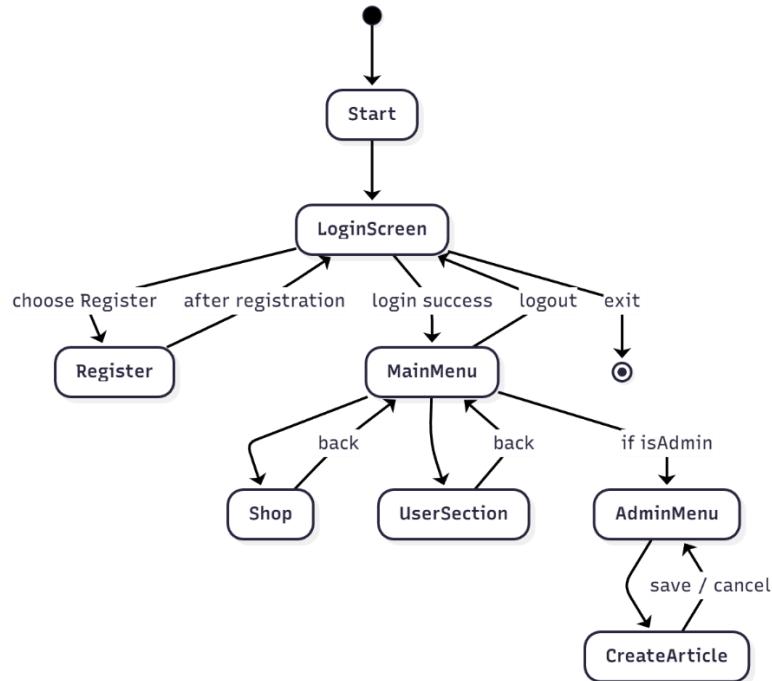
```
// Öffentliche Methode zum Anzeigen der detaillierten Bestellinformationen
/// Diese Methode kann von anderen Klassen aufgerufen werden
/// Benötigt ein Order-Objekt als Parameter
/// Generiert aus dem Order-Objekt ein Layout mit den Bestelldetails
1 Verweis
public static Layout DisplayDetailedOrderInfo(Order order)
{
    return AccountOrders(order);
}
```

Implementierungsbeispiel des MainMenu

```
GUI_Display.DisplayWindow(GUI_MainMenu.ShowMainMenu());
MenuSelection(UserInput.MenuChoice(menuChoices));
```

## 2.2.2. Login und User Registrierung

Die Steuerung des Logins wird von **Functions.Login** geregelt. Dies stellt zugleich auch den Einstiegspunkt des Users dar.



Diese Klasse regelt die Menüführung und die Logik der Menüoptionen.

Dabei werden **Functions.PasswordHelper** und **User.UserRegistration** aufgerufen, um die Logik auszulagern und die Klasse an sich übersichtlich zu halten.

Das Passwort wird dabei per Hash Funktion überprüft und mit einem Salt versehen um gleiche Passwort Hashs zu vermeiden. Bei der Registrierung wird ebenfalls die

```

do
{
    // Update GUI with username
    GUI_Display.DisplayWindow(GUI_LogIn.ShowLoginMenu(userName));

    // Prompt for password
    password = UserInput.GetPasswordInput("Please enter your password: ");

    // Verify password
    passwordIsValid = PasswordHelper.VerifyPassword(userName, password);

    // Handle invalid password
    if (passwordIsValid == false)
    {
        AnsiConsole.MarkupLine("[bold red]Invalid password.[/]");
        Thread.Sleep(500);
        continue;
    }
    else
    {
        break;
    }
} while (true);

// Set active user
Account ActiveUser = Account.GetAccountByUserName(userName);
Program.SetActiveUser(ActiveUser);

// Confirm successful login
AnsiConsole.MarkupLine("[bold green]Login successful![/]");
Thread.Sleep(1000);

// Show main menu
MainMenu.ShowMainMenu();

break;

```

```

// Hash password with provided salt
// Used for verifying passwords
1 Verweis
public static string HashPassword(string password, byte[] salt)
{
    var hash = Rfc2898DeriveBytes.Pbkdf2(          // PBKDF2 password-based key derivation function
        password,
        salt,
        _iterations,
        HashAlgorithmName.SHA256,
        _hashSize);
    return Convert.ToString(hash);
}

// Hash password and generate new salt
// Used for creating new accounts or changing passwords
2 Verweise
public static string HashPassword(string password, out byte[] Salt)
{
    byte[] salt = GenerateSalt();                  // Generate new salt
    var hash = Rfc2898DeriveBytes.Pbkdf2(
        password,
        salt,
        _iterations,
        HashAlgorithmName.SHA256,
        _hashSize);

    Salt = salt;
    return Convert.ToString(hash);
}

// Verify password against stored hash and salt
// Used for login
2 Verweise
public static bool VerifyPassword(string userName, string password)           // Verify input password for given username
{
    byte[] salt = Account.GetAccountSalt(userName);                           // Retrieve stored salt
    string storedHash = Account.GetAccountPasswordHash(userName);            // Retrieve stored hash
    string passwordHash = HashPassword(password, salt);                      // Hash input password with stored salt

    if (passwordHash == storedHash)                                            // Compare hashes
    {
        return true;                                                       // Password matches
    }

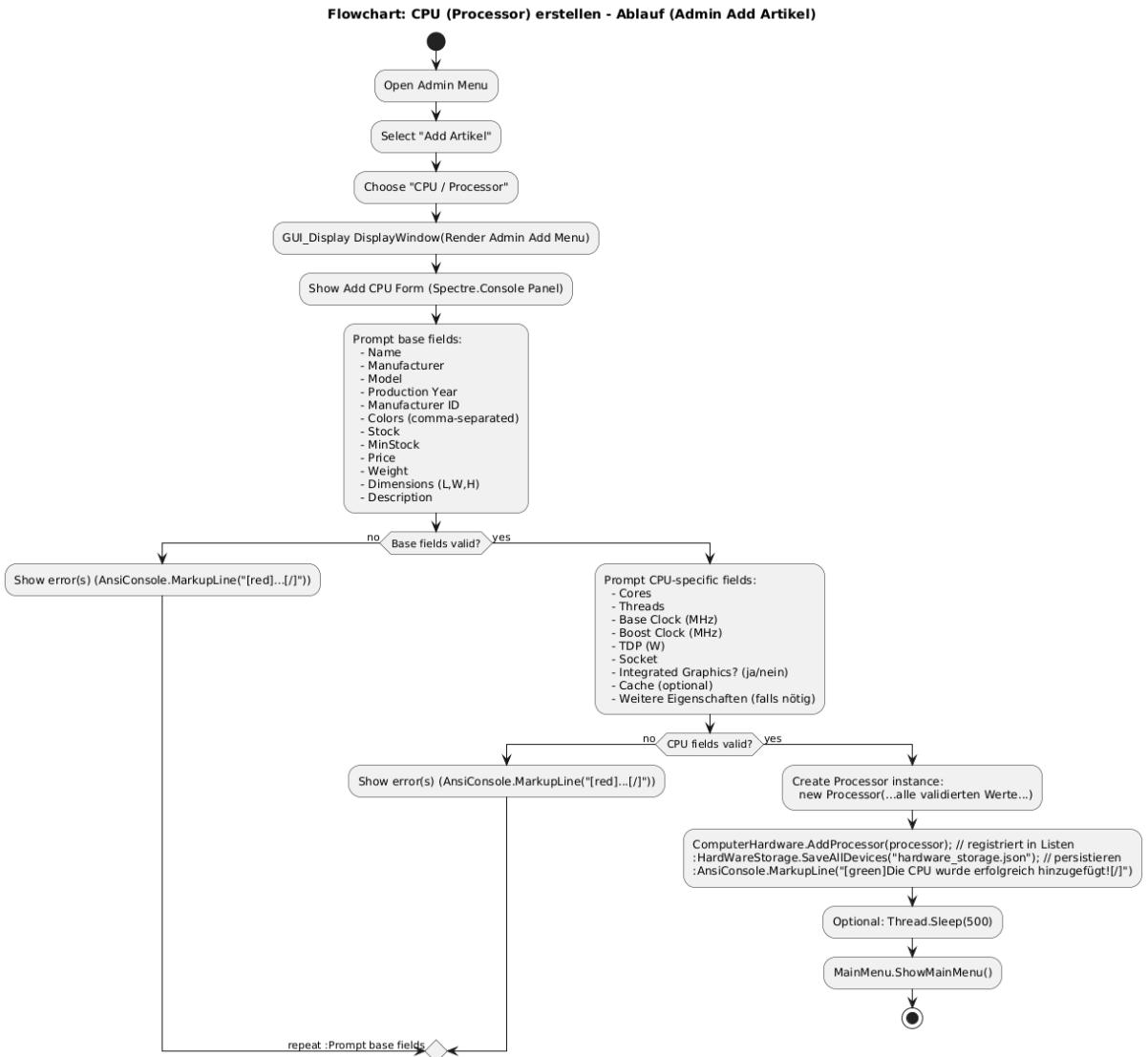
    return false;                                                       // Password does not match
}

```

### 2.2.3. Hardware Klasse

**Hardware.ComputerHardWare** regelt durch Vererbung die Erstellung von Artikeln.

Dabei werden Variablen die alle Produkte besitzen wie zum Beispiel Artikel IDs an die Base Klasse weitergeleitet und nur die Hardware spezifischen Variablen wie zum Beispiel Core Count bei einer CPU und den abgeleiteten Klassen.



Die Felder der Artikel sind private und können nur durch Aufruf von Methoden der ComputerHardware Klasse ausgelesen werden.

Das Speichern der neu Erstellten Hardware wird dann über verschiedenen Listen in der Base Klasse übernommen.

```
internal static List<Case> Cases = new List<Case>();
internal static List<Motherboard> Motherboards = new List<Motherboard>();
internal static List<PowerSupply> PowerSupplies = new List<PowerSupply>();
internal static List<Processor> Processors = new List<Processor>();
internal static List<GraphicsCard> GraphicsCards = new List<GraphicsCard>();
internal static List<Ram> RAMs = new List<Ram>();
internal static List<StorageDevice> StorageDevices = new List<StorageDevice>();
internal static List<CoolingSystem> CoolingSystems = new List<CoolingSystem>();
internal static List<Peripheral> Peripherals = new List<Peripheral>();
internal static List<Display> Displays = new List<Display>();
internal static List<Software> Softwares = new List<Software>();

internal static List<ComputerHardware> Devices = new List<ComputerHardware>(0);
internal static List<string> Manufacturers = new List<string>();
```

## Verfügbare Hardware Klassen

### Namespace: Elektrogrosshandel.Hardware

- Case
  - Accessibility: internal class
  - File: VSCode/Elektrogrosshandel/Elektrogrosshandel/Hardware/100\_Case.cs
  - Beschreibung: Erbt von ComputerHardware; speichert case-spezifische Eigenschaften (FormFactor, FanSlots, FrontPanelPorts). Registriert sich bei ComputerHardware.AddCase. Erzeugt Artikel-IDs.
- Motherboard
  - Accessibility: internal class
  - File: VSCode/Elektrogrosshandel/Elektrogrosshandel/Hardware/200\_Motherboard.cs
  - Beschreibung: Erbt von ComputerHardware; Eigenschaften wie Socket, RamType, FormFactor, PCIeVersion, StorageInterfaces. Registriert sich bei ComputerHardware.AddMotherboard und erzeugt Artikel-IDs.
- PowerSupply
  - Accessibility: internal class
  - File: VSCode/Elektrogrosshandel/Elektrogrosshandel/Hardware/300\_PowerSupply.cs
  - Beschreibung: Erbt von ComputerHardware; spezifische Eigenschaft Wattage; ruft ComputerHardware.AddPowerSupply. (Datei enthält Plan/Pseudocode + Implementierungsteil.)
- GraphicsCard
  - Accessibility: internal class
  - File: VSCode/Elektrogrosshandel/Elektrogrosshandel/Hardware/500\_GraphicsCard.cs
  - Beschreibung: Erbt von ComputerHardware; enthält VRAM, MemoryType, Taktwerte, TDP, Outputs, PowerConsumption, etc. Registriert sich bei ComputerHardware (AddGraphicsCard). Datei enthält ausführlichen Pseudocode/Implementierung.
- Ram
  - Accessibility: internal class
  - File: VSCode/Elektrogrosshandel/Elektrogrosshandel/Hardware/600\_Ram.cs
  - Beschreibung: Erbt von ComputerHardware; RAM-spezifische Felder (Kapazität, Frequenz, Typ, Module, ECC). Registriert sich bei ComputerHardware.AddRAM und erzeugt Artikel-IDs.
- StorageDevice
  - Accessibility: internal class
  - File: VSCode/Elektrogrosshandel/Elektrogrosshandel/Hardware/700\_StorageDevices.cs
  - Beschreibung: Erbt von ComputerHardware; Felder: Kapazität, Typ (SSD/HDD), Interface, Read/Write-Speed, FormFactor. Datei enthält Pseudocode und Implementierungsskizze.

- CoolingSystem
  - Accessibility: internal class
  - File: VSCode/Elektrorgrosshandel/Elektrorgrosshandel/Hardware/800\_CoolingSystem.cs
  - Beschreibung: Erbt von ComputerHardware; Felder wie FanSpeed, LiquidCooled, TDP, Kompatibilität. Registriert sich bei ComputerHardware.AddCoolingSystem. Enthält Pseudocode/Implementierung.
- Display
  - Accessibility: internal class
  - File: VSCode/Elektrorgrosshandel/Elektrorgrosshandel/Hardware/925\_Display.cs
  - Beschreibung: Erbt von ComputerHardware; Display-spezifische Felder (Resolution, RefreshRate, PanelType, Size, HDR, Ports, AdaptiveSync, Curved, Touch, Brightness, AspectRatio). Enthält Pseudocode + Implementierungsskizze.

## 2.2.4. Speichern und Laden

Functions.HardWareStorage und Functions.AccountStorrage übernehmen das Speichern der Hardware und Accounts. Dies geschieht über die Newtonsoft JSON Erweiterung. Diese bietet beim speichern von privaten Feldern deutlich einfachere Lösungen.

```
2 Verweise
private static JsonSerializerSettings GetSerializerSettings()
{
    return new JsonSerializerSettings
    {
        Formatting = Formatting.Indented,
        TypeNameHandling = TypeNameHandling.Auto,
        ConstructorHandling = ConstructorHandling.AllowNonPublicDefaultConstructor,
        ContractResolver = new DefaultContractResolver
        {
            DefaultMemberSearchFlags = BindingFlags.Public | BindingFlags.NonPublic | BindingFlags.Instance
        },
        PreserveReferencesHandling = PreserveReferencesHandling.None,
        NullValueHandling = NullValueHandling.Ignore,
        Converters = new List<JsonConverter> { new MarkupJsonConverter() }
    };
}

5 Verweise
public static void SaveAllAccounts(string filePath)
{
    if (string.IsNullOrWhiteSpace(filePath)) throw new ArgumentException("Pfad ist ungültig.", nameof(filePath));

    var accounts = Elektrogrosshandel.Account.GetAllAccounts() ?? new List<Elektrogrosshandel.Account>();
    var settings = GetSerializerSettings();

    string json = JsonConvert.SerializeObject(accounts, settings);

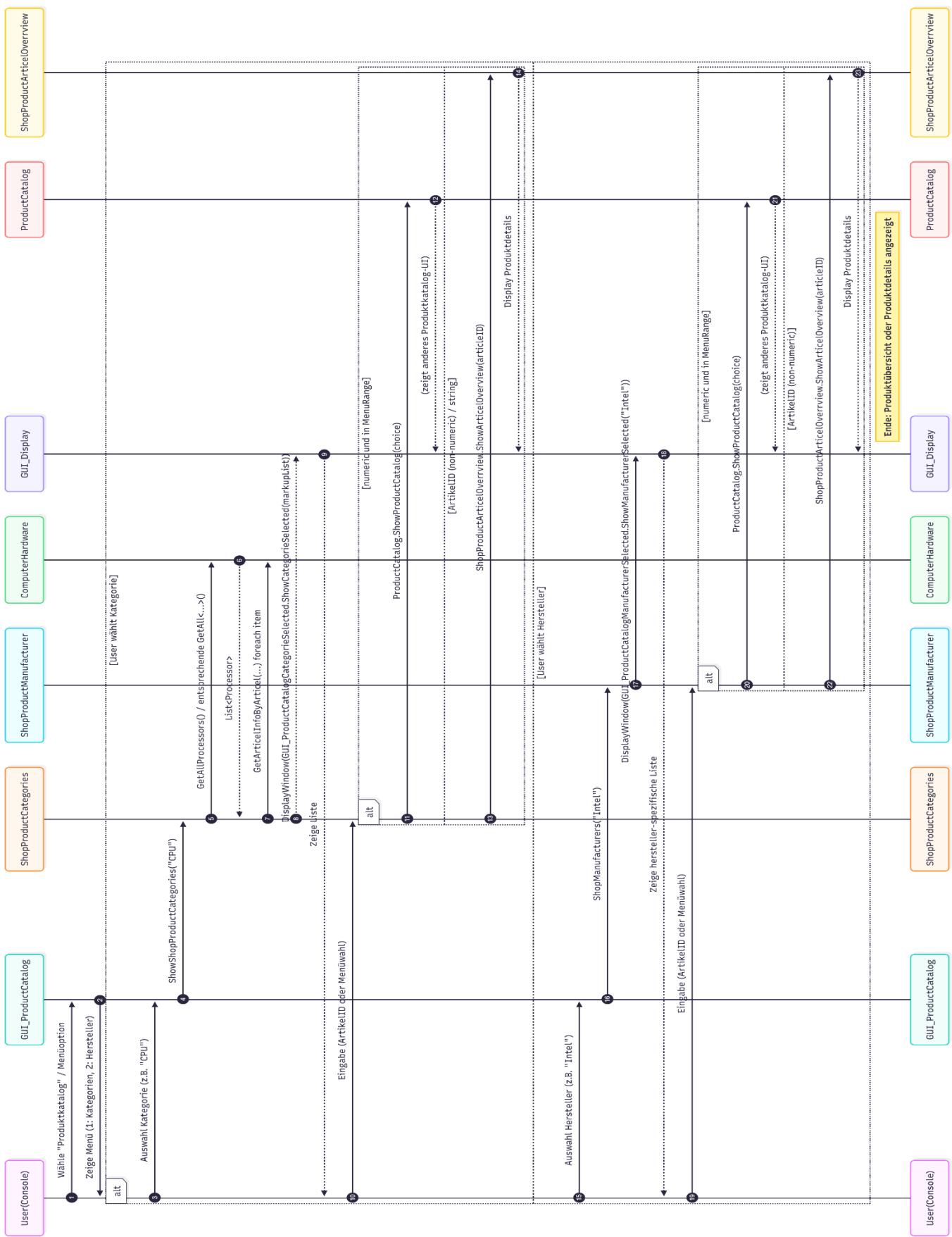
    var dir = Path.GetDirectoryName(filePath);
    if (!string.IsNullOrEmpty(dir) && !Directory.Exists(dir))
    {
        Directory.CreateDirectory(dir);
    }

    File.WriteAllText(filePath, json);
}
```

Das Laden und Speichern der Hardware ist ebenfalls so eingerichtet und läuft über eine separate Klasse.

Beim speichern werden JSON-Dateien erstellt die in der MAIN Methode aufgerufen werden. Sollten die Dateien nicht existieren werden sie erstellt.

## 2.2.5. Übersicht Shop Führung



## 2.3. Schwierigkeiten bei der Umsetzung

Die meisten Probleme hat das Erstellen der Speicher Funktionen und Lade Funktionen bereitet. Durch die Verwendung der privaten Variablen ergaben sich ungeahnte Probleme.

Das Speichern und Laden hatte zwar ohne Fehler zu verursachen funktioniert, allerdings waren alle gespeicherten Variablen 0 bzw. null.

Zudem war der geplante Funktionsumfang zu Projektbeginn zu groß gesteckt für die zur Verfügung stehende Zeit.

Dadurch war ich leider gezwungen die Computer Builder Funktion, in der man sich einen Pc hätte bestellen können (Computer besteht jeweils aus einer Hardware jeden benötigten Typs). Die Logik die noch fehlt wäre einfach aber Zeitaufwendig wenn man mit checks für Leistungs-, Sockelkompatibilität usw.

## 3. Verwendung von KI

Die KI wurde immer dann verwendet wenn ich viele gleiche Klassen wie z.B. all die abgeleiteten Hardware Klassen. Hierbei habe ich die Base Klasse und die erste abgeleitete Klasse selbst erstellt und dann durch KI die übrigen Klassen ergänzen lassen.

Ebenso bei allen Add\* Funktionen, diese sind jeweils das selbe nur mit anderen List Elementen die abgerufen werden.

Zudem hat mich die KI bei den Problemen mit dem Laden und Speichern sowie beim richtigen erstellen der Hash Funktion unterstützt da diese nach dem Speichern des Hashs ebenfalls Probleme verursacht hat.

Ansonsten wurde die KI während des Codeerstellens nur zur Codebereinigung und zum Farbschema anpassen genutzt.

Bei der Erstellung der Dokumentation und der dazugehörigen Flowcharts hab ich das Projekt durch die KI analysieren lassen.

Sowie bei der Erstellung der dazugehörigen \_\_VI (Versionsinfo) und \_\_TO(TechnicalOverview)

## 4. Weiterführende Links und Dateien

Das komplette Projekt ist mit allen dazugehörigen Dateien auf GitHub zu finden.

Repository: <https://github.com/Plessu/Elektrgrosshandel>

Zusätzlich gibt es zu jedem Namespace eine passende \_\_TO und \_\_VI Datei die einen technischen Überblick und einen Versionsverlauf bieten. Alle arbeiten an dem Projekt sind sauber Comites und gebracht zur besseren Verfolgbarkeit der Arbeiten.

Bei Fragen stehe ich natürlich auch gerne jederzeit zur Verfügung.