

HBF SWI

HÖHERE BERUFSFACHSCHULE FÜR WIRTSCHAFTSINFORMATIK

Hausarbeit

im Bildungsgang

„Staatlich geprüfte/r Wirtschaftsinformatiker/in“

gemäß §5 der Ausbildungs- und Prüfungsordnung

Entwicklung einer LegoSet- Verwaltungssoftware

vorgelegt von: Florian Emilio Fritz

Klasse: WI23A1

Adresse: Steinbachstraße 97

Ort: 66424 Homburg

E-Mail: florian.fritz.1@gmx.de

Abgabetermin: 15.05.2025

Betreuer/in: Herr Schuler

Inhaltsverzeichnis

Dokumentation	4
Vorwort	4
Benutzerhandbuch	4
API-Schlüssel eingeben	4
Login	5
Die Startseite	7
LegoSet-Suche	8
Sammlung.....	10
Statistiken	12
Implementierung	13
Klassendiagramm.....	13
Beschreibung wichtiger Quellcode Ausschnitte.....	14
Beschreibung der Schnittstellen (API)	20
Abweichungen vom Pflichtenheft/Lastenheft.....	21
Tests.....	23
Installationsanleitung.....	25
Anhang	27
Pflichten-/Lastenheft	27
1. Einführung	30
2. Ist-Situation	30
3. Soll-Situation	30
3.1 Soll-Zustand	30
3.2 Funktionale Anforderungen	31
3.3 Nicht-Funktionale Anforderungen.....	31
3.4 Schnittstellen	31
3.5 Risiken.....	31
4. Abnahmekriterien	32
5. Use-Case-Diagramme	33
6. Projektplan	35
7. Produktumgebung.....	35

8.	Skizze von GUI oder Webseite	35
9.	DB-Entwurf	38
10.	Link zu einem gehosteten Git-Repository	38
11.	Testplan	38
	Verwendete Hilfsmittel und Quellen	42
	Fazit	42
	Eidesstattliche Erklärung	42

Dokumentation

Vorwort

Diese Hausarbeit entstand im Rahmen der Abschlussarbeit im Bereich der Softwareentwicklung und behandelt die Planung und Umsetzung eines Lego-Set-Verwaltungssystems. Ziel des Projekts war es, eine benutzerfreundliche Anwendung zu entwickeln, mit der sich Lego-Sets strukturiert erfassen, verwalten und bei Bedarf auch automatisch über eine externe API suchen lassen.

Die Idee zum Projekt entstand aus persönlichem Interesse und der Überlegung, wie man eine stetig wachsende Sammlung effizient digital dokumentieren kann. Das Programm richtet sich an Sammler, die nicht nur Wert auf Ordnung legen, sondern auch ihre eigenen Preise, Notizen und Mengen verwalten möchten.

Für die Umsetzung kamen moderne Technologien wie C#, WPF für die Benutzeroberfläche und SQLite als lokale Datenbank zum Einsatz. Die Rebrickable-API ermöglicht eine automatische Ergänzung von Set-Informationen, sofern ein API-Key angegeben wird.

Alle Anforderungen wurden selbstständig auf Basis eines Pflichten-/Lastenhefts umgesetzt. Bei der Entwicklung wurde großer Wert auf einfache Handhabung, stabile Datenverarbeitung und Erweiterbarkeit gelegt.

Benutzerhandbuch

API-Schlüssel eingeben

Bitte gib deinen Rebrickable API-Key ein:

Beim ersten Start fragt das Programm nach deinem persönlichen Rebrickable-Schlüssel. Diesen bekommst du zusammen mit dem Programm (z. B. als Textdatei oder per E-Mail).

Rebrickable ist die API, die im Hintergrund die LEGO-Datenbank abfragt. Damit das funktioniert, muss dein Schlüssel bei jeder Abfrage mitgeschickt werden. Was passiert nach dem Klick auf Speichern?

- Der Schlüssel wird geprüft
Das Programm schaut, ob der eingegebene Schlüssel das richtige Format, hat also z. B. nicht zu kurz oder leer ist.
- Der Schlüssel wird dauerhaft gespeichert
Damit man ihn nicht jedes Mal neu eingeben muss, wird er automatisch in einer Textdatei auf deinem Rechner abgelegt.

- Die Verbindung zu Rebrickable wird aktiviert
Von jetzt an wird der Schlüssel bei jeder Online-Abfrage automatisch mitgesendet.
Dadurch kannst du Sets suchen, Themen vorladen oder Preise abrufen, ohne dich weiter darum kümmern zu müssen.

Login

Oben rechts ist der Login Button.

Wenn man da drauf klickt, öffnet sich ein neues Fenster, in dem man sich entweder einloggen, neu registrieren oder ein Passwort mit einem Sicherheitscode zurücksetzen kann.

Loginfenster

Benutzername

Passwort

- **Benutzername**
hier gibst du deinen registrierten Namen ein.

- **Passwort**
Das zugehörige Passwort wird verdeckt eingegeben.
- **Login-Button**
Melde dich an, nur möglich wenn deine Einlogdaten stimmen
- „**Registrieren**“
öffnet ein weiteres Fenster zur Neuanmeldung.
- „**Passwort vergessen**“
Dann öffnet das Zurücksetzungs-Fenster, in dem du über deinen persönlichen Sicherheitscode ein neues Passwort setzen kannst.

Registrierungsfenster

Benutzername

E-Mail

Passwort

Registrieren Zurück

Im Registrierungsfenster kannst du dich jetzt mit einem neuen Benutzerkonto registrieren.

Trag dafür deinen gewünschten Benutzernamen, deine E-Mail-Adresse und ein sicheres Passwort ein.

Nach dem Klick auf „Registrieren“ bekommst du eine wichtige Info: dein persönlicher Sicherheitscode.

Diesen Code brauchst du später, falls du dein Passwort mal vergessen solltest – also am besten gleich notieren.

Wichtiger Hinweis

X



Registrierung erfolgreich!

Dein persönlicher Sicherheitscode lautet:

FJBMTC

! Diesen Code brauchst du, um dein Passwort zurückzusetzen!

Bitte notiere ihn sicher.

OK

Dein Benutzer wird direkt in der lokalen Datenbank gespeichert, und du kannst dich danach sofort anmelden.

Mit dem Button „Zurück“ kommst du wieder zum Loginfenster.

Passwort vergessen

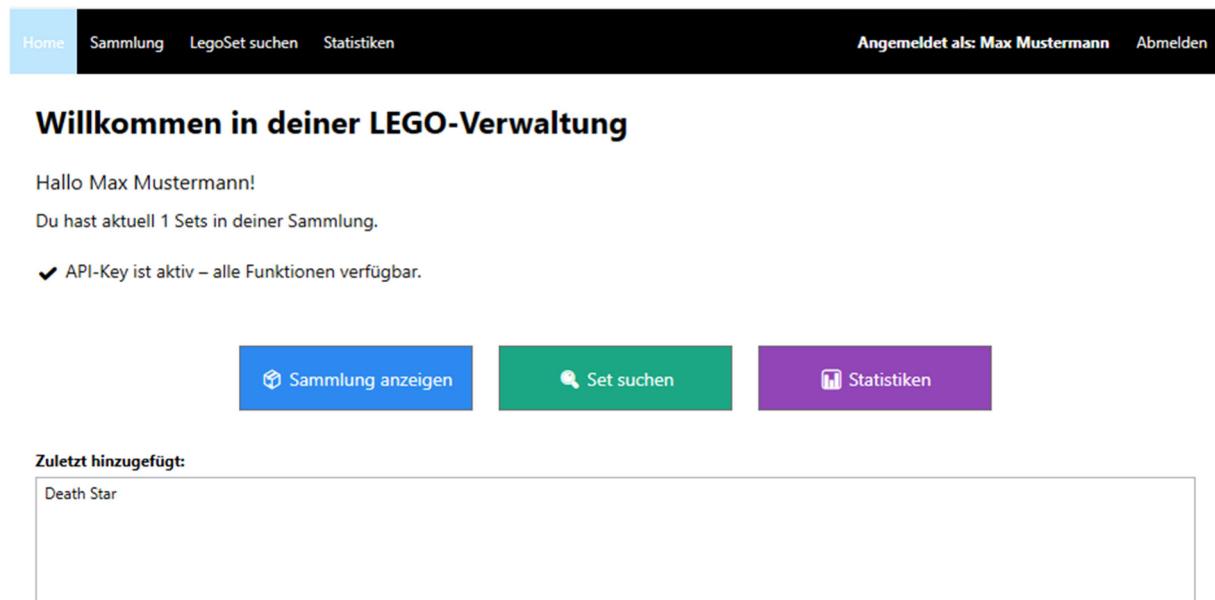
Benutzername

Sicherheitscode

Neues Passwort

Nachdem drücken auf Passwort vergessen öffnet sich ein neues Fenster in dem du dein Benutzername, Sicherheitscode und dein neues Passwort eingibst
Wenn der Sicherheitscode zu deinem Benutzernamen passt, wird dein neues Passwort übernommen.

Die Startseite



The screenshot shows the main interface of the LEGO collection management system. At the top, there is a navigation bar with links for "Home", "Sammlung", "LegoSet suchen", and "Statistiken". On the right side of the bar, it shows the user is "Angemeldet als: Max Mustermann" and has a "Abmelden" button. Below the navigation bar, the main content area is titled "Willkommen in deiner LEGO-Verwaltung". It greets the user "Hallo Max Mustermann!" and informs them that they have "aktuell 1 Sets in deiner Sammlung.". A note indicates that an API-Key is active and all functions are available. There are three prominent buttons at the bottom: "Sammlung anzeigen" (blue), "Set suchen" (green), and "Statistiken" (purple). A sidebar on the left lists "Zuletzt hinzugefügt:" followed by "Death Star".

Die Startseite begrüßt dich nach dem Login mit deinem Namen und zeigt dir direkt, wie viele Sets aktuell in deiner Sammlung gespeichert sind.

Außerdem siehst du auf einen Blick, ob dein API-Key aktiv ist – also ob die Verbindung zur Rebrickable-Datenbank funktioniert.

Über die farbigen Schaltflächen kannst du direkt in die wichtigsten Bereiche springen:

- Sammlung anzeigen: deine gespeicherten Sets verwalten
- Set suchen: neue Sets per Setnummer, Name oder Thema finden
- Statistiken: Auswertungen zu deinem Inventar (Wert, Gewinn)

Außerdem siehst du deine Zuletzt hinzugefügten Sets.

Oben rechts kannst du dich außerdem jederzeit wieder abmelden.

LegoSet-Suche

The screenshot shows a web-based application interface. At the top, there is a navigation bar with links for 'Home', 'Sammlung', 'LegoSet suchen', and 'Statistiken'. On the right side of the navigation bar, it says 'Angemeldet als: Max Mustermann' and 'Abmelden'. Below the navigation bar, there is a search section with three input fields: 'Setnummer' (with a dropdown arrow), 'Name' (with a dropdown arrow), and 'Thema vorladen' (with a dropdown arrow). Below these fields is a 'Jahr:' dropdown set to 'Alle'. Underneath the search section is a table with columns: Setnummer, Name, Thema, Jahr, Anzahl, and Aktion. The table currently has no data rows.

Hier kannst du gezielt nach LEGO-Sets per Api-Abfrage suchen und neue Sets in deiner Sammlung speichern.

Dafür stehen dir zwei Möglichkeiten zur Verfügung:

- Manuelle Suche über die Eingabezeile (Setnummer, Name oder Thema)
Optional mit Jahres Filter.
- Vorladen ganzer Themen (z. B. „Star Wars“) direkt aus der Rebrickable-Datenbank

Die Ergebnisse werden dir in einer ListView angezeigt.

Mit einem Klick auf „Hinzufügen“ wird das gewünschte Set zu deiner lokalen Sammlung hinzugefügt.

Im Hintergrund wird geprüft, ob das Set bereits vorhanden ist.

Falls es noch nicht in deiner Datenbank gespeichert ist, wird es dort neu angelegt (inkl. Name, Setnummer, Jahr, Thema usw.).

Der Datensatz wird dauerhaft in deiner lokalen SQLite-Datenbank gespeichert.

The screenshot shows a search interface for LEGO sets. At the top, there are navigation links: Home, Sammlung, LegoSet suchen, Statistiken, and a user login message: Angemeldet als: Max Mustermann, Abmelden. Below this is a search bar with the input 'Falcon'. A dropdown menu is open next to it, showing options: Name, Setnummer, Name, and Thema. The 'Name' option is selected. To the right of the dropdown is a 'Suchen' button. Below the search bar is a date selector labeled 'Jahr:' with the value '2024'. The main area displays a table of search results:

Setnummer	Name	Thema	Jahr	Anzahl	Aktion
6523826-1	Millennium Falcon	Star Wars	2024	1	Hinzufügen
75375-1	Millennium Falcon	Star Wars	2024	1	Hinzufügen
75389-1	The Dark Falcon	Star Wars	2024	1	Hinzufügen

Mit dem Suchfeld am oberen Rand kannst du gezielt nach bestimmten LEGO-Sets suchen.

Dabei wählst du im Dropdown-Menü aus, wonach du suchen möchtest:

- nach einer Setnummer (z. B. „10188“)
- nach einem Namen (z. B. „Death Star“)
- nach einem Thema (z. B. „Ultimate Collector Series“)
- Optional Jahres-Filter (z.B „2024“)

Nach dem Klick auf „Suchen“ wird die Rebrickable-Datenbank abgefragt.

Alle passenden Sets werden in einer Liste darunter angezeigt – inklusive Setnummer, Name, Thema und Jahr.

Außerdem kannst du direkt die gewünschte Anzahl an Sets angeben, die du zu deiner Sammlung hinzufügen möchtest.

Mit „Hinzufügen“ wird das ausgewählte Set in deiner lokalen Datenbank gespeichert. Falls es noch nicht vorhanden ist, wird es automatisch neu angelegt.

Die Angabe zur Anzahl wird in der Tabelle (BenutzerSet) hinterlegt und steht dann direkt in deiner Sammlung zur Verfügung.

The screenshot shows a search interface for LEGO sets. At the top, there are navigation links: Home, Sammlung, LegoSet suchen, Statistiken, and a Login button. Below the navigation is a search bar with fields for 'Setnummer' and a 'Suchen' button. To the left of the search bar is a dropdown menu titled 'Thema vorladen' containing a list of LEGO themes. The themes listed are: 12V, 4 Juniors, 4.5V, 9V, Action Wheelers, Activity Books, Activity Books with LEGO Parts, Advent, Adventurers, Agents, Airlines, Airport, Alien Conquest, and Alpha Team.

Mit der Funktion „Thema vorladen“ kannst du ganze LEGO-Themen wie z. B. „City“, „Star Wars“ oder „Technic“ vollständig in deine lokale Datenbank speichern. Um die wichtigsten Sets auch dann griffbereit zu haben, wenn du das Programm mal offline genutzt werden soll.

Wähle dazu einfach ein Thema aus der Liste aus und klicke auf „Thema vorladen“. Das Programm lädt alle zugehörigen Sets über die Rebrickable-API herunter und speichert sie dauerhaft auf deinem Rechner. Bereits vorhandene Sets werden dabei automatisch übersprungen.

Sammlung

The screenshot shows the 'Meine Sammlung' (My Collection) view. At the top, there are navigation links: Home, Sammlung, LegoSet suchen, Statistiken, and a user status message 'Angemeldet als: Test' with a 'Abmelden' button. Below the navigation is a heading 'Meine Sammlung' and a 'Set manuell hinzufügen' button. The main area is a table displaying the collection items:

Setnummer	Name	Thema	Jahr	Anzahl	Gezahlt Preis	UVP (€)	Notizen	Aktionen
10281	Bonsai Tree	Botanical	2021	1	39.99	49.99	Test	Speichern Löschen
75304	Darth Vader Helm	Star Wars	2021	1	69.99	69.99	Test	Speichern Löschen
75350	Clone Commander Cody	Star Wars	2023	1	59.99	69.99	Test	Speichern Löschen
75375	Millennium Falcon	Star Wars	2024	1	84.99	84.99	Test	Speichern Löschen
75328	Mando Helm	Star Wars	2022	1	64.99	69.99	Test	Speichern Löschen
76191	Infinity Gauntlet	Marvel	2021	1	59.99	79.99	Test	Speichern Löschen
75349	Captain Rex Helm	Star Wars	2023	1	69.99	69.99	Test	Speichern Löschen

In der Sammlungsansicht siehst du alle LEGO-Sets, die du deiner persönlichen Sammlung hinzugefügt hast.

Für jedes Set werden dir wichtige Informationen angezeigt – wie Setnummer, Name, Thema, Jahr, Anzahl, gezahlter Preis und UVP.

Man kann die Anzahl, den gezahlten Preis, denn UVP (€) und die Notizen jederzeit ändern.

Mit einem Klick auf „Speichern“ werden deine Änderungen direkt in der lokalen Datenbank gespeichert.

Falls du ein Set nicht mehr hast, kannst du es über den Button „Löschen“ aus deiner Sammlung löschen.

Mit dem Button „Set manuell hinzufügen“ oben rechts kannst du eigene Sets eintragen, die du nicht über die Rebrickable-Suche gefunden hast.

So lassen sich z. B. seltene oder eigene Sets ebenfalls erfassen.

Set manuell hinzufügen

Setnummer	<input type="text"/>
Name	<input type="text"/>
Thema	<input type="text"/>
Jahr	<input type="text"/>
Preis	<input type="text"/>
Anzahl	<input type="text"/>
Notiz	<input type="text"/>

Im Fenster „Set manuell hinzufügen“ kannst du LEGO-Sets eintragen, die nicht über die Rebrickable-Suche verfügbar sind – zum Beispiel seltene Modelle, Sondereditionen oder eigene Sets.

Du gibst dafür einfach die wichtigsten Daten per Hand ein:

Setnummer, Name, Thema, Jahr, Preis, Anzahl und optional eine Notiz.

Mit einem Klick auf „Speichern“ wird das neue Set in deiner lokalen Datenbank gespeichert und automatisch zu deiner Sammlung hinzugefügt.

Es verhält sich anschließend genauso wie ein regulär importiertes Set und kann jederzeit bearbeitet oder gelöscht werden.

Statistiken

Home Sammlung LegoSet suchen Statistiken Angemeldet als: Test Abmelden

Statistiken

Gesamtanzahl Sets:	7
Gesamtwert bezahlt (€):	449,93 €
Gesamtwert UVP (€):	494,93 €
Differenz UVP vs. Bezahlte (€):	45,00 €

In den Statistiken bekommst du eine Übersicht über deine LEGO-Sammlung. Alle Werte beziehen sich auf den aktuell angemeldeten Benutzer.

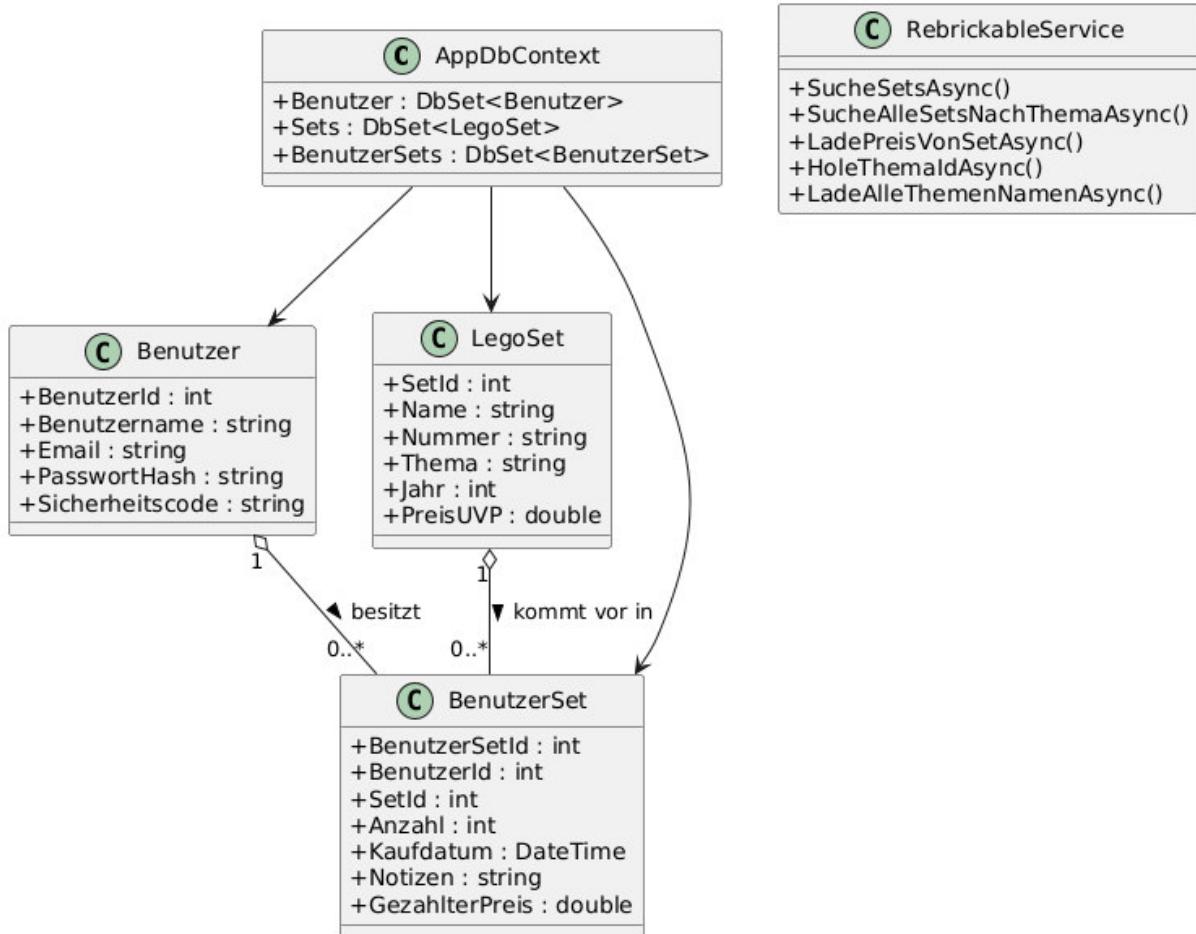
- Gesamtanzahl Sets
Wie viele Sets du insgesamt in deiner Sammlung hast (unabhängig vom Thema).
- Gesamtwert bezahlt (€)
Der Gesamtwert basierend auf dem Preis, den du tatsächlich gezahlt hast.
- Gesamtwert UVP (€)
Der Gesamtwert basierend auf dem offiziellen Listenpreis (UVP) der Sets.
- Differenz UVP vs. Bezahlte (€)
Zeigt dir, ob du günstiger oder teurer eingekauft hast.

Diese Werte werden direkt aus der lokalen Datenbank berechnet und live aktualisiert, sobald du in der Sammlung etwas änderst.

Sie helfen dir, einen Überblick über den Wert deiner Sammlung und deine Investitionen zu behalten.

Implementierung

Klassendiagramm



In diesem Projekt geht es um eine Anwendung zur Verwaltung von LEGO-Sets. Das Klassendiagramm zeigt, wie die verschiedenen Teile der Anwendung zusammenhängen und welche Daten dabei eine Rolle spielen.

Im Mittelpunkt steht die **AppDbContext**-Klasse. Sie ist sozusagen das Herzstück der Datenbankanbindung und sorgt dafür, dass alle Informationen über Benutzer, Sets und deren Besitz sauber gespeichert und abgerufen werden können. Dabei greift sie auf drei wichtige Klassen zu: **Benutzer**, **LegoSet** und **BenutzerSet**.

Benutzer:

Diese Klasse repräsentiert eine Person, die sich bei der Anwendung anmeldet. Gespeichert werden z. B. Benutzername, E-Mail und Passwort (als Hash). Ein Benutzer kann natürlich mehrere Sets besitzen.

LegoSet:

Jedes LEGO-Set hat hier seine eigene Klasse mit Infos wie Name, Set-Nummer, Thema, Jahr und Preis. Auch ein Set kann von mehreren Benutzern gekauft worden sein.

BenutzerSet:

Diese Klasse verbindet Benutzer und Sets. Sie speichert z. B., wann ein Set gekauft wurde, wie viel es gekostet hat, wie oft es gekauft wurde und ob es irgendwelche Notizen dazu gibt. So lässt sich genau nachvollziehen, wer was besitzt.

Externer Service: RebrickableService

Ein weiterer Teil des Projekts ist der RebrickableService. Der verbindet die App mit der offiziellen Rebrickable-API im Internet. Darüber kann man z. B. nach neuen Sets suchen oder sich alle Sets eines bestimmten Themas anzeigen lassen. Auch Preise lassen sich damit abrufen. Die Klasse ist komplett statisch und hilft einfach dabei, Live-Daten von außen in die App zu bringen.

Beschreibung wichtiger Quellcode Ausschnitte.

- *Login*

```
private void BtnLogin_Click(object sender, RoutedEventArgs e)
{
    string benutzername = txtBenutzername.Text.Trim();
    string passwort = txtPasswort.Password.Trim();
    string hash = HashPasswort(passwort);

    using (var db = new AppDbContext())
    {
        var user = db.Benutzer.FirstOrDefault(u => u.Benutzername == benutzername
&& u.PasswortHash == hash);
        if (user != null)
        {
            App.Current.Properties["BenutzerId"] = user.BenutzerId;
            App.Current.Properties["BenutzerName"] = user.Benutzername;
            this.DialogResult = true;
            this.Close();
        }
        else
        {
            MessageBox.Show("Falscher Benutzername oder Passwort.");
        }
    }
}
```

Hier handelt es sich um die Methode, die aufgerufen wird, wenn man im Login-Fenster auf den Button klickt. Der Ablauf ist recht simpel, aber effektiv.

Zuerst holt sich die Methode den Benutzernamen und das Passwort, die der User eingegeben hat. Beide Werte werden von überflüssigen Leerzeichen befreit. Das Passwort wird direkt in einen Hash umgewandelt, damit es nicht im Klartext weiterverarbeitet wird – wie es sich gehört.

Dann wird eine Verbindung zur Datenbank geöffnet (über den AppDbContext). In dieser wird geprüft, ob es einen Benutzer gibt, dessen Benutzername und Passwort-Hash zu dem eingegebenen passen. Also ein klassischer Login-Vergleich.

Wenn ein passender Benutzer gefunden wird, werden dessen ID und Name in den globalen App-Eigenschaften gespeichert, damit man später im Programm noch darauf zugreifen kann. Dann wird das Login-Fenster geschlossen und der Benutzer ist eingeloggt.

Falls der Benutzername oder das Passwort falsch war, wird einfach eine Fehlermeldung ausgegeben.

- Set per API suchen

```

private async void BtnSuchen_Click(object sender, RoutedEventArgs e)
{
    string suchbegriff = txtSuche.Text.Trim();
    if (string.IsNullOrEmpty(suchbegriff))
    {
        MessageBox.Show("Bitte einen Suchbegriff eingeben.");
        return;
    }

    string suchtyp = (cmbSuchtyp.SelectedItem as
ComboBoxItem)?.Content?.ToString();

    // Sets suchen (immer über API)
    suchErgebnisse = suchtyp switch
    {
        "Setnummer" => await RebrickableService.SucheSetsAsync(suchbegriff,
"set_num"),
        "Name" => await RebrickableService.SucheSetsAsync(suchbegriff, "name"),
        "Thema" => await RebrickableService.SucheSetsAsync(suchbegriff, "theme"),
        _ => new List<LegoSet>()
    };

    // Jahr auslesen
    string jahrText = cmbJahr.SelectedItem?.ToString();
    int? filterJahr = (jahrText != "Alle" && int.TryParse(jahrText, out int jahr))
? jahr : null;

    // ✎ Jahrfilter anwenden – unabhängig von Datenquelle
    if (filterJahr != null)
    {
        suchErgebnisse = suchErgebnisse
            .Where(set => set.Jahr == filterJahr.Value)
            .ToList();
    }

    // Anzeige aktualisieren
    lvErgebnisse.ItemsSource = suchErgebnisse;
}

```

Diese Methode wird ausgeführt, wenn man in der App auf den "Suchen"-Button klickt. Die Methode sorgt dafür, dass man nach bestimmten LEGO-Sets suchen kann – entweder nach Name, Thema oder Setnummer.

Es wird zuerst überprüft, ob überhaupt ein Suchbegriff eingegeben wurde. Wenn das Feld leer ist, bekommt der Nutzer direkt eine Meldung und der ganze Vorgang wird abgebrochen.

Wenn aber ein Begriff drinsteht, wird geschaut, welcher Suchtyp im Dropdown ausgewählt wurde (also ob man z. B. nach dem Namen oder der Setnummer suchen will). Danach wird eine passende Abfrage an die Rebrickable-API geschickt also direkt online gesucht. Das Ganze läuft asynchron, damit die App dabei nicht hängen bleibt.

Dann geht's weiter mit einem optionalen Filter: Wenn im Jahr-Dropdown ein konkretes Jahr ausgewählt wurde (z. B. 2021), werden die Suchergebnisse nochmal darauf gefiltert. So sieht man nur die Sets aus diesem Jahr.

Zum Schluss werden die Ergebnisse in der ListView angezeigt, also schön im UI dargestellt.

- *Thema Vorabladen*

```

private async void BtnThemaVorladen_Click(object sender, RoutedEventArgs e)
{
    if (cmbVorladeThema.SelectedItem == null)
    {
        MessageBox.Show("Bitte ein Thema auswählen.");
        return;
    }

    string thema = cmbVorladeThema.SelectedItem as string;

    var sets = await RebrickableService.SucheAlleSetsNachThemaAsync(thema);

    if (sets == null || sets.Count == 0)
    {
        MessageBox.Show("Keine Sets gefunden.");
        return;
    }

    // Fortschrittsbalken konfigurieren
    progressBar.Visibility = Visibility.Visible;
    progressBar.Minimum = 0;
    progressBar.Maximum = sets.Count;
    progressBar.Value = 0;

    int neueSets = 0;

    // Sets lokal in der Datenbank speichern, wenn sie noch nicht existieren
    using (var db = new AppDbContext())
    {
        foreach (var set in sets)
        {
            bool existiert = db.Sets.Any(s => s.Nummer == set.Nummer);
            if (!existiert)
            {
                db.Sets.Add(set);
                neueSets++;
            }

            progressBar.Value += 1;
            await Task.Delay(5); // kleine Pause für optisches Feedback
        }
        db.SaveChanges();
    }

    progressBar.Visibility = Visibility.Collapsed;

    MessageBox.Show($"{sets.Count} Sets gefunden.\nDavon {neueSets} neue Sets gespeichert.");
}

```

Diese Methode wird ausgeführt, wenn man auf den Button klickt, um LEGO-Sets zu einem bestimmten Thema vorzuladen. Ziel ist es, Sets aus der Rebrickable-API zu holen und sie lokal in der eigenen Datenbank zu speichern aber natürlich nur, wenn sie dort noch nicht existieren.

Zuerst wird geprüft, ob überhaupt ein Thema ausgewählt wurde. Falls nicht, bekommt man einen Hinweis und es passiert nichts weiter. Wenn ein Thema gewählt wurde, wird über den RebrickableService nach passenden Sets gesucht das läuft wieder asynchron, also im Hintergrund.

Wenn keine Sets gefunden wurden, gibt's ebenfalls eine Meldung. Wenn doch, geht's los:

Ein Fortschrittsbalken wird eingeblendet, damit man sieht, dass gerade etwas passiert. Danach wird durch alle gefundenen Sets durchiteriert und jeweils geprüft, ob das Set anhand seiner Nummer schon in der lokalen Datenbank vorhanden ist. Falls nicht, wird es hinzugefügt und gezählt.

Die kleine Pause (await Task.Delay(5)) sorgt dafür, dass der Fortschrittsbalken flüssiger aussieht, rein fürs Auge.

Wenn alles durchgelaufen ist, wird gespeichert und der Fortschrittsbalken wieder ausgeblendet. Zum Schluss gibt's nochmal eine kurze Zusammenfassung: Wie viele Sets wurden insgesamt gefunden und wie viele davon waren wirklich neu.

- *Set zur Sammlung hinzufügen*

```
private async void BtnHinzufuegen_Click(object sender, RoutedEventArgs e)
{
    if (sender is Button btn && btn.Tag is LegoSet selectedSet)
    {
        using (var db = new AppDbContext())
        {
            var setInDb = db.Sets.FirstOrDefault(s => s.Nummer == selectedSet.Nummer);

            // Set noch nicht in DB → Preis laden + speichern
            if (setInDb == null)
            {
                double preis = await
RebrickableService.LadePreisVonSetAsync(selectedSet.Nummer);
                selectedSet.PreisUVP = preis;

                db.Sets.Add(selectedSet);
                db.SaveChanges();
                setInDb = selectedSet;
            }

            // Benutzer-ID aus globalem Speicher holen
            if (App.Current.Properties["BenutzerId"] is int benutzerId)
            {
                // Anzahl aus der TextBox auslesen (pro Zeile)
                int anzahl = 1;
                foreach (var item in lvErgebnisse.Items)
                {
                    if (item == selectedSet)
                    {
                        var container =
lvErgebnisse.ItemContainerGenerator.ContainerFromItem(item) as ListViewItem;
                        if (container != null)
                        {
                            var textBoxes = FindVisualChildren<TextBox>(container);
                            foreach (var tb in textBoxes)
                            {
                                if (int.TryParse(tb.Text, out int parsedAnzahl) &&
parsedAnzahl > 0)
                                {
                                    anzahl = parsedAnzahl;
                                }
                            }
                        }
                    }
                }

                // BenutzerSet erstellen oder erhöhen
                var vorhandenesBenutzerSet = db.BenutzerSets.FirstOrDefault(bs =>
bs.BenutzerId == benutzerId && bs.SetId == setInDb.SetId);

                if (vorhandenesBenutzerSet == null)
                {

```

```
        var neuesBenutzerSet = new BenutzerSet
    {
        BenutzerId = benutzerId,
        SetId = setInDb.SetId,
        Kaufdatum = DateTime.Now,
        Notizen = "",
        GezahlterPreis = setInDb.PreisUVP,
        Anzahl = anzahl
    };
    db.BenutzerSets.Add(neuesBenutzerSet);
}
else
{
    vorhandenesBenutzerSet.Anzahl += anzahl;
}

db.SaveChanges();
MessageBox.Show($"Set {setInDb.Name} wurde deiner Sammlung hinzugefügt!");
}
else
{
    MessageBox.Show("Bitte zuerst einloggen, um Sets zu speichern.");
}
}
```

Diese Methode wird aufgerufen, wenn man in der Trefferliste auf den „Hinzufügen“-Button klickt, also wenn ein bestimmtes LEGO-Set zur eigenen Sammlung gespeichert werden soll.

Zuerst wird geprüft, welches Set überhaupt gemeint ist (über das Tag-Attribut des Buttons). Dann wird eine Verbindung zur Datenbank aufgebaut.

Falls das ausgewählte Set noch nicht in der lokalen Datenbank existiert, wird zuerst der Preis über die Rebrickable-API geladen und das Set wird neu in die Datenbank eingetragen. Das passiert also nur einmal pro Set.

Anschließend wird geschaut, ob der Nutzer eingeloggt ist, die Benutzer-ID wird dabei aus dem globalen Speicher geholt (App.Current.Properties). Wenn kein Benutzer angemeldet ist, gibt's eine Meldung, dass das so nicht funktioniert.

Jetzt kommt der etwas aufwendigere Teil: Die Methode versucht herauszufinden, wie oft der Nutzer das Set hinzufügen will. Dafür wird in der Trefferliste (ListView) nach der passenden TextBox gesucht, die die Anzahl enthält. Falls ein gültiger Wert drinsteht, wird dieser verwendet, ansonsten bleibt es bei 1.

Dann wird geprüft, ob der Nutzer dieses Set schon in seiner Sammlung hat. Falls nicht, wird ein neuer Eintrag erstellt mit aktuellem Datum, Preis und Anzahl. Falls das Set schon da ist, wird einfach nur die Anzahl erhöht.

Zum Schluss wird alles gespeichert und es gibt eine kleine Erfolgsmeldung, dass das Set jetzt in der Sammlung ist.

- *Datenmodell-Konfiguration*

```
public class AppDbContext : DbContext
{
    public DbSet<Benutzer> Benutzer { get; set; }
    public DbSet<LegoSet> Sets { get; set; }
    public DbSet<BenutzerSet> BenutzerSets { get; set; }

    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        optionsBuilder.UseSqlite("Data Source=legosets.db");
    }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<LegoSet>()
            .HasKey(x => x.SetId); // <<< Hier explizit sagen: SetId ist der Primärschlüssel

        modelBuilder.Entity<BenutzerSet>()
            .HasKey(bs => bs.BenutzerSetId);

        modelBuilder.Entity<BenutzerSet>()
            .Property(bs => bs.BenutzerSetId)
            .ValueGeneratedOnAdd(); // <-- AutoIncrement aktivieren

        modelBuilder.Entity<BenutzerSet>()
            .HasOne(bs => bs.Benutzer)
            .WithMany(b => b.BenutzerSets)
            .HasForeignKey(bs => bs.BenutzerId);

        modelBuilder.Entity<BenutzerSet>()
            .HasOne(bs => bs.Set)
            .WithMany(s => s.BenutzerSets)
            .HasForeignKey(bs => bs.SetId);
    }

    public void EnsureDatabase()
    {
        this.Database.EnsureCreated();
    }
}
```

Wie im Pflichten-/Lastenheft vorgesehen, verwendet die Anwendung SQLite als eingebettete Datenbank. Diese ermöglicht eine einfache Verteilung der Software, da keine zusätzliche Datenbankinstallation erforderlich ist. Alle Benutzerdaten, Sets und Zuordnungen werden dauerhaft lokal in einer .db-Datei gespeichert.
Die Konfiguration der Datenbankstruktur erfolgt über die Methode `OnModelCreating` in der `AppDbContext`-Klasse

In der Methode `OnModelCreating` wird die Struktur der Datenbank festgelegt:

- SetId ist der Primärschlüssel der Tabelle `LegoSet`.
- BenutzerSetId ist der Primärschlüssel der Tabelle `BenutzerSet` und wird automatisch fortlaufend vergeben.
- Zwischen `BenutzerSet` und `Benutzer` besteht eine 1:n-Beziehung: Ein Benutzer kann mehrere Sets besitzen.
- Zwischen `BenutzerSet` und `LegoSet` besteht ebenfalls eine 1:n-Beziehung: Ein Set kann in mehreren Sammlungen vorkommen.
- Die Fremdschlüsseleien sorgen dafür, dass `BenutzerSets` mit den passenden Benutzern und Sets verknüpft sind.

- Die Methode EnsureCreated() prüft beim Programmstart, ob die Datenbank bereits existiert, und erstellt sie automatisch, wenn sie fehlt.

Beschreibung der Schnittstellen (API)

Für die Datenerweiterung nutzt das Programm die **Rebrickable API** eine externe Schnittstelle, über die Informationen zu offiziellen LEGO-Sets abgefragt werden können.

Die API liefert alle relevanten Set-Daten wie Name, Nummer, Thema, Erscheinungsjahr und sofern verfügbar den UVP-Preis.

Die Schnittstelle wird im Programm über die statische Klasse RebrickableService verwendet.

Der Zugriff erfolgt über HTTPS-Anfragen mit einem persönlichen API-Key, den der Nutzer beim ersten Programmstart eingibt.

Die erhaltenen JSON-Daten werden mithilfe von System.Text.Json deserialisiert und in eigene C#-Modelle (RebrickableSet, RebrickableTheme) überführt.

Verwendete Endpunkte:

GET /sets/?search={suchbegriff}

→ Sucht LEGO-Sets anhand von Name, Nummer oder Thema.

GET /sets/?theme_id={id}&page={page}

→ Lädt alle Sets eines bestimmten Themas wichtig für die Vorab-Ladefunktion.

GET /sets/{set_num}/

→ Liefert Detailinformationen zu einem bestimmten Set (z. B. UVP in USD).

Verarbeitung im Code:

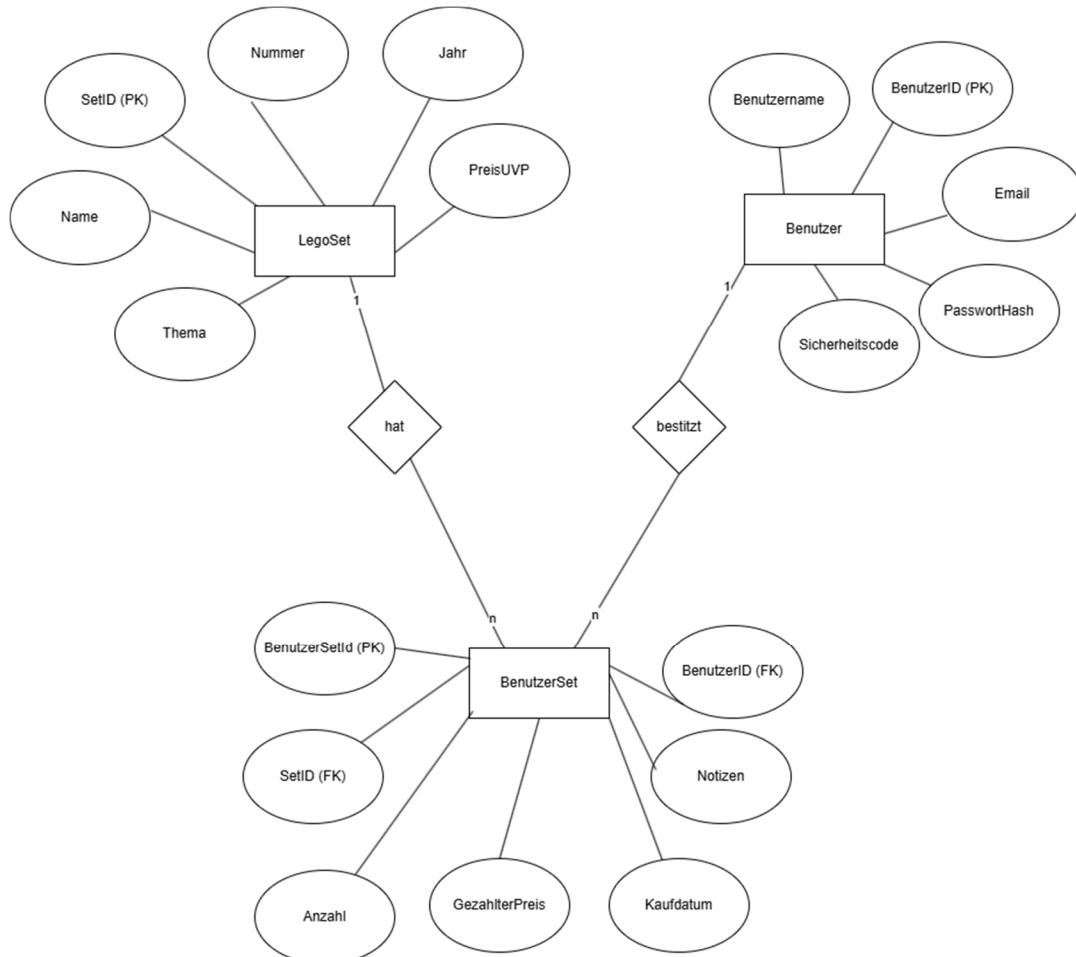
Der Zugriff erfolgt über die Klasse HttpClient, wobei der API-Key im Header übergeben wird:

Authorization: key <API_KEY>

Die empfangenen Daten werden in Listen von LegoSet-Objekten umgewandelt und bei Bedarf dauerhaft in der lokalen SQLite-Datenbank gespeichert.

Abweichungen vom Pflichtenheft/Lastenheft

- *ER-Diagramm*



Im Laufe der praktischen Umsetzung wurde das ursprünglich im Pflichten-/Lastenheft entworfene ER-Diagramm überarbeitet und an die tatsächlichen Anforderungen und technischen Rahmenbedingungen angepasst. Ziel war es, ein realistisches, sauberes Datenmodell zu entwickeln, das sowohl manuelle Eingaben als auch API-basierte Datensätze zuverlässig abbilden kann.

Im ursprünglichen Modell wurde eine eher abstrakte Struktur verwendet, mit Entitäten wie „API Daten“ und „Inventar“. Diese wurden im finalen Datenmodell ersetzt durch die konkreteren Entitäten **LegoSet** und **BenutzerSet**. Dabei ergaben sich folgende wichtige Änderungen:

Die Entität „API Daten“ wurde zu **LegoSet**, da dort nicht nur API-Daten gespeichert werden, sondern auch manuell erfasste Sets.

Die Entität „Inventar“ wurde durch **BenutzerSet** ersetzt. Diese neue Entität verbindet Benutzer und **LegoSet** miteinander, enthält jedoch zusätzlich wichtige Informationen wie Anzahl, GezahilterPreis, Kaufdatum und Notizen.

Die Beziehungen wurden präzisiert: Ein **LegoSet** „hat“ viele **BenutzerSet**-Einträge, ein Benutzer „besitzt“ viele **BenutzerSet**-Einträge. Damit wurde klar abgebildet, dass mehrere Benutzer dasselbe Set besitzen können – mit jeweils eigenen Informationen.

Attribute wie Teil-Anzahl, Genre oder Preis pro Stück wurden verworfen, da sie im praktischen Einsatz keine Relevanz hatten bzw. nicht über die API bereitgestellt wurden.

Zusätzlich wurde ein Sicherheitscode beim Benutzer eingeführt, um ein Zurücksetzen des Passworts zu ermöglichen. Auch dies war im ursprünglichen Modell noch nicht vorgesehen.

Durch diese Anpassungen ist ein praxisnahes, technisch sauberes Datenmodell entstanden, das sowohl funktionale als auch sicherheitsrelevante Anforderungen zuverlässig erfüllt.

- *Weitere Änderungen gegenüber dem Pflichten-/Lastenheft*

1. Benutzerverwaltung erweitert

- **Sicherheitscode-Funktion** wurde ergänzt, damit Benutzer ihr Passwort zurücksetzen können.
- Dies war im ursprünglichen Lastenheft nicht vorgesehen, erhöht aber die Sicherheit und Benutzerfreundlichkeit.

2. Keine separate Inventar-Tabelle

- Im ursprünglichen Modell war Inventar eine eigene Entität.
- Diese wurde vollständig durch die Verknüpfung BenutzerSet ersetzt, wodurch eine klarere und relationale Struktur entstanden ist.

3. Offline-Funktion durch „Thema vorladen“

- Ursprünglich war nur die API-Suche geplant.
- Durch das **Thema-Vorladen** können Nutzer nun auch Sets lokal speichern und später offline durchsuchen.

4. Verzicht auf überflüssige Felder

- Felder wie „Teil-Anzahl“, „Genre“ oder „Preis pro Stück“ aus der ursprünglichen Planung wurden entfernt, da sie:
 - über die API nicht konsistent vorhanden waren
 - für die Nutzung keinen praktischen Mehrwert geboten hätten

5. API-Key-System eingeführt

- Um die Nutzung transparent und steuerbar zu machen, wird beim ersten Start ein **API-Key** abgefragt und gespeichert.
- Dies war nicht im Pflichten-/Lastenheft enthalten, ist aber in der Praxis notwendig, da Rebrickable eine Authentifizierung verlangt.

6. Testdaten beim ersten Start

- In der Umsetzung wird beim ersten Start ein Benutzer „Test“ mit mehreren Beispielsets erstellt.
- Diese Daten dienen zur Demonstration und zum Testen, waren aber ursprünglich nicht Teil des Pflichtenhefts.

7. Verzicht auf automatisierte Unit-Tests

- Auf klassische Unit-Tests wurde verzichtet, da alle wesentlichen Funktionen manuell getestet wurden (siehe Testfälle T01–T09).
- Die Tests decken zentrale Anwendungsfälle wie Registrierung, Login, Set-Verwaltung und API-Nutzung ab.
- Diese manuelle Prüfung hat gezeigt, dass die Anwendung stabil funktioniert, wodurch der zusätzliche Aufwand für automatisierte Tests in diesem Projekt nicht als notwendig erachtet wurde.

- Beim Test T03 erfolgte die Änderung, das beim Einloggen eines registrierten Benutzers die Startseite öffnet.
- Beim Test T07 wird der Gesamtwert der Sammlung bei Statistiken angezeigt.
- Bei den Tests T08 und T09 wurde auf die Anzeige von Bild-URLs verzichtet, da die Rebrickable-API hierfür keine direkte Unterstützung bot.

Tests

Es wurden manuelle Test durchgeführt, die folgende Übersicht zeigt die Testergebnisse der Tests T01-T09.

ID	T01
Beschreibung	Ein Benutzer registriert sich mit Benutzername, E-Mail und Passwort.
Datum	12.05.2025
Tester	Florian Fritz
Beobachtes Resultat	Die Registrierung wird als erfolgreich in der message box angezeigt, es wird ein Sicherheitscode genannt, der zum eventuellen Zurücksetzen des Passwortes gebraucht wird.
Testergebnis	true

ID	T02
Beschreibung	Ein Benutzer versucht, sich mit einem bereits registrierten Benutzernamen zu registrieren.
Datum	12.05.2025
Tester	Florian Fritz
Beobachtes Resultat	Registrierung ist nicht möglich, es erscheint eine Meldung in der message box, dass der Benutzer bereits registriert ist
Testergebnis	true

ID	T03
Beschreibung	Ein registrierter Benutzer loggt sich ein.
Datum	12.05.2025
Tester	Florian Fritz
Beobachtes Resultat	Der Benutzer wird angemeldet, man bleibt auf der Homepage, das Inventar wird nicht direkt angezeigt.
Testergebnis	true

ID	T04
Beschreibung	Ein Benutzer versucht, sich mit einem falschen Passwort einzuloggen.
Datum	12.05.2025
Tester	Florian Fritz
Beobachtes Resultat	Das System gibt eine Fehlermeldung mit falscher Benutzername oder Passwort via message box.
Testergebnis	true

ID	T05
Beschreibung	Der Benutzer fügt ein Set manuell hinzu.
Datum	12.05.2025
Tester	Florian Fritz
Beobachtes Resultat	Nach manueller Eingabe eines Sets werden die Daten in der Datenbank gespeichert und anschließend in der Sammlung angezeigt.
Testergebnis	true

ID	T06
Beschreibung	Ein Set wird aus dem Inventar gelöscht.
Datum	12.05.2025
Tester	Florian Fritz
Beobachtes Resultat	Durch das Drücken auf Löschen wird das Set aus der Sammlung entfernt und die Verbindung Benutzer und Sets im BenutzerSets gelöscht.
Testergebnis	true

ID	T07
Beschreibung	Der Gesamtwert des Inventars wird berechnet und angezeigt.
Datum	12.05.2025
Tester	Florian Fritz
Beobachtes Resultat	Der Gesamtwert wird erfolgreich berechnet und angezeigt.
Testergebnis	true

ID	T08
Beschreibung	Ein Set wird über die API hinzugefügt
Datum	12.05.2025
Tester	Florian Fritz
Beobachtes Resultat	Die Auswahl über API ist möglich, in einer Listview kann das entsprechende Set ausgewählt werden und über den Button hinzufügen zur Sammlung hinzugefügt werden. Es erscheint eine messagebox, mit Set hinzugefügt.
Testergebnis	true

ID	T09
Beschreibung	Benutzer sieht die letzten 5 hinzugefügten Sets auf der Startseite.
Datum	12.05.2025
Tester	Florian Fritz
Beobachtetes Resultat	Auf der Startseite werden die zuletzt hinzugefügten Sets angezeigt.
Testergebnis	true

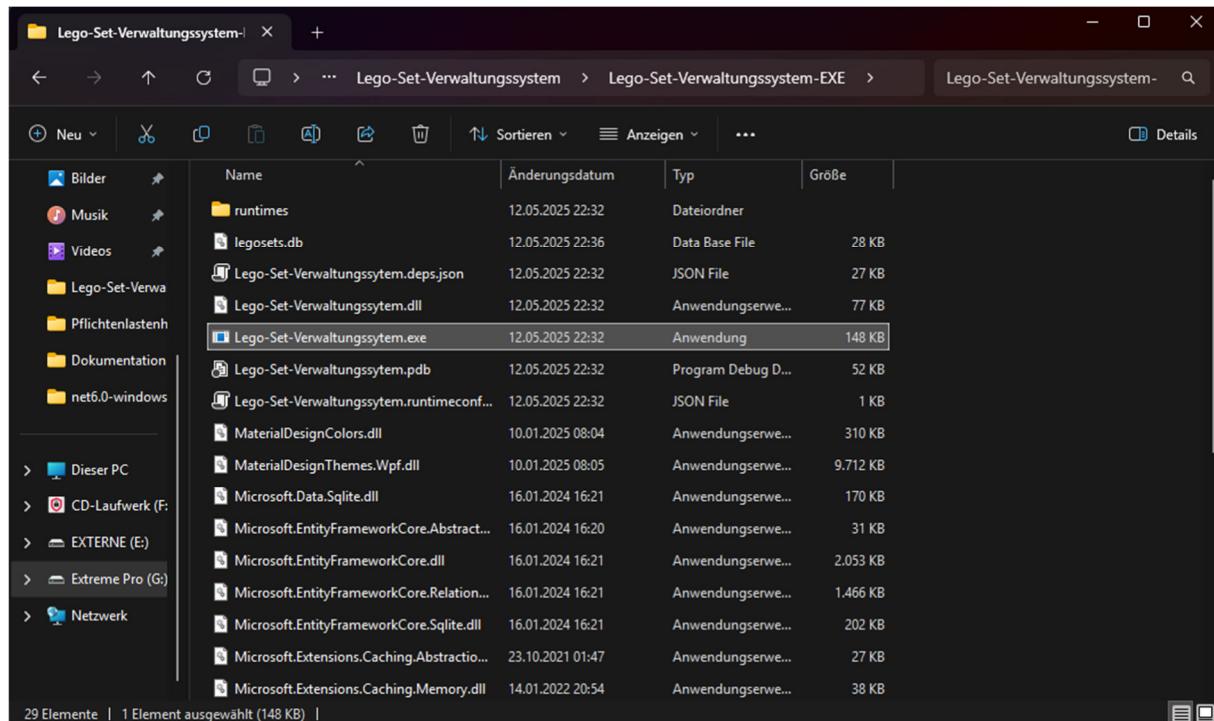
Installationsanleitung

Systemanforderungen:

Windows 10 oder höher

- .NET 6 Desktop Runtime (falls nicht vorhanden)
- Internetverbindung für API-Funktionen

Starten der Anwendung:



- Öffne die Datei LegoSetVerwaltung.exe unter diesem Pfad "G:\Lego-Set-Verwaltungssystem\Lego-Set-Verwaltungssystem-EXE\Lego-Set-Verwaltungssystem.exe".
- Doppelklick auf die .exe, um die Anwendung zu starten.

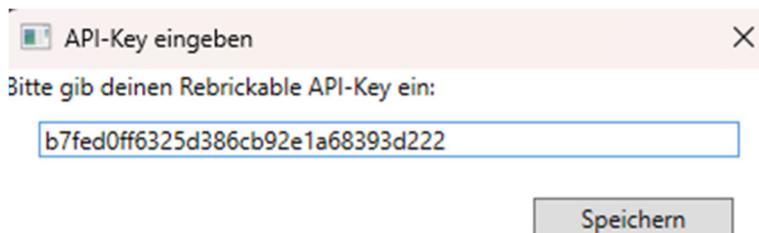
Datenbank-Erstellung:

- Beim ersten Start wird automatisch die SQLite-Datenbank legosets.db im Projektverzeichnis erzeugt.
- Es sind bereits Beispielsets und ein Testnutzer enthalten.

Name	Änderungsdatum	Typ	Größe
runtimes	12.05.2025 22:32	Dateiordner	
legosets.db	12.05.2025 22:36	Data Base File	28 KB
Lego-Set-Verwaltungssystem.deps.json	12.05.2025 22:32	JSON File	27 KB
Lego-Set-Verwaltungssystem.dll	12.05.2025 22:32	Anwendungserwe...	77 KB
Lego-Set-Verwaltungssystem.exe	12.05.2025 22:32	Anwendung	148 KB
Lego-Set-Verwaltungssystem.pdb	12.05.2025 22:32	Program Debug D...	52 KB
Lego-Set-Verwaltungssystem.runtimeconf...	12.05.2025 22:32	JSON File	1 KB

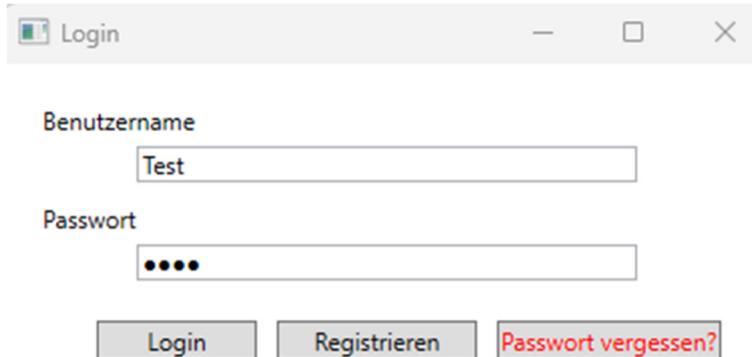
API-Key (Rebrickable):

- Beim Start des Programms wird ein API-Key abgefragt.
- Dieser ist erforderlich, um auf die Rebrickable-Schnittstelle zuzugreifen (Sets automatisch suchen/hinzufügen).
- Dein Persönlicher API-Key lautet= **cdfb9123e018d1ee0af12af8a51931b7**
- Wird kein API-Key angegeben, sind die API-Funktionen (z. B. Set-Suche) deaktiviert.



Login-Daten zum Testen:

- Benutzername: Test
- Passwort: Test



Keine Standard Installation erforderlich:

- Es handelt sich um eine portable Anwendung – es ist keine Installation oder Admin-Berechtigung nötig.
- Die Daten werden lokal gespeichert. Es ist keine dauerhafte Internetverbindung erforderlich, außer für die API.

Support:

- Bei Problemen oder Fragen zur Anwendung bitte per E-Mail melden:
florian.fritz.1@gmx.de

Anhang

Pflichten-/Lastenheft

Florian Fritz

Lego-Set- Verwaltungssystem

Pflichten und Lastenheft

Florian Emilio Fritz
22.1.2025

Inhalt

1.	Einführung	30
2.	Ist-Situation	30
3.	Soll-Situation	30
3.1	Soll-Zustand	30
3.2	Funktionale Anforderungen	31
3.3	Nicht-Funktionale Anforderungen.....	31
3.4	Schnittstellen	31
3.5	Risiken.....	31
4.	Abnahmekriterien	32
5.	Use-Case-Diagramme	33
6.	Projektplan	35
7.	Produktumgebung.....	35
8.	Skizze von GUI oder Webseite.....	35
9.	DB-Entwurf	38
10.	Link zu einem gehosteten Git-Repository	38
11.	Testplan	38

1. Einführung

Das Projekt wird im Rahmen eines internen Entwicklungsprojekts durchgeführt. Das Projekt soll die Verwaltung von Lego-Sets vereinfachen. Das Programm ist grundsätzlich für jeden Lego-Fan mit besonderer Zielgruppe von Sammlern, die ihre Sammlung/Inventar verwalten wollen, um eine besseren Übersicht zu haben. Durch eine zusätzliche Datenbank lassen sich Daten perfekt abspeichern.

2. Ist-Situation

Als großer Lego-Fan mit einem größeren Inventar kann es schnell passieren das man den Überblick seiner Sets verliert. Hier kommt das Projekt zum Einsatz, es soll dem Nutzer die Verwaltung seiner Sammlung vereinfachen. Vorher haben viele es mit einer Tabelle oder auf Papier von Hand zu Fuß gemacht, was bei größeren Sammlungen schnell zu Fehlern führen kann. Dies soll das Programm erleichtern.

3. Soll-Situation

3.1 Soll-Zustand

Nach Abschluss des Projekts wird eine einfache Software bereitstehen, mit der Lego-Sammler ihre Sets verwalten können. Die Software soll den Wunsch erfüllen, alle Informationen zu einer Sammlung an einem zentralen Ort zu speichern und leicht abrufbar zu sein.

Mit der Anwendung können Nutzer ihre Sets erfassen, Informationen wie Name, Nummer, Thema und Preis(UVP) speichern sowie den Gesamtwert der Sammlung automatisch berechnen lassen. Durch die Anbindung an eine externe Datenbank (Rebrickable) wird es möglich sein, Sets direkt zu suchen und hinzuzufügen, was Zeit spart, und die manuelle Eingabe reduziert.

Die Vorteile der Software liegen vor allem darin, dass Sammler ihre Sammlung besser im Blick haben und schneller Änderungen vornehmen können.

Insgesamt bietet die Software eine sinnvolle Lösung, um die Verwaltung einer Lego-Sammlung zu erleichtern und übersichtlicher zu gestalten.

3.2 Funktionale Anforderungen

Funktion	Beschreibung	Aufwand (Stunden)
Set hinzufügen	Nutzer können neue Lego-Sets manuell anlegen oder über eine API-Suche hinzufügen.	12
Set löschen	Sets können aus der Sammlung entfernt werden.	8
Sets durchsuchen	Sets per Nummer suchen	12
Inventarwert berechnen	Gesamtwert des Inventars Berechnen	8
Integration einer API	Integration von API(Rebrickable) für Daten abruf	20
Datenbankanbindung	Verbindung zur Datenbank zum Speichern und Abrufen von Sets.	15

3.3 Nicht-Funktionale Anforderungen

Funktion	Beschreibung	Aufwand (Stunden)
GUI erstellen	Eine Intuitive GUI erstellen	25

3.4 Schnittstellen

-**Rebrickable API:** Sucht Lego-Sets anhand eines Namens oder einer Nummer.

3.5 Risiken

Risiko	Verantwortlicher	Alternative Lösung
API nicht verfügbar	Entwickler	Manuelles Hinzufügen von Sets als Alternative anbieten. API Daten in der Datenbank speichern
Datenbankzugriffsprobleme	Entwickler	Lokale Backups erstellen

4. Abnahmekriterien

Muss-Kriterien

1. Sets verwalten

- Sets können hinzugefügt und gelöscht werden.

2. Inventarwert anzeigen

- Der Gesamtwert der Sammlung wird korrekt berechnet.

3. Filterfunktion nach Nummer

- Sets können gezielt anhand ihrer Set-Nummer gefiltert werden.

4. Benutzeroberfläche

- Die Software verfügt über eine funktionsfähige und intuitive grafische Benutzeroberfläche.

5. Login Service

- Man kann sich registrieren und Anmelden

Kann-Kriterien

1. Exportfunktion

- Die Sammlung kann als PDF exportiert werden.

2. Zuletzt hinzugefügt

- Eine Fenster das die Zuletzt hinzugefügten Sets anzeigt

3. Erweiterte Filterfunktionen

- Weitere Filtermöglichkeiten wie nach Namen, Thema oder Jahr.

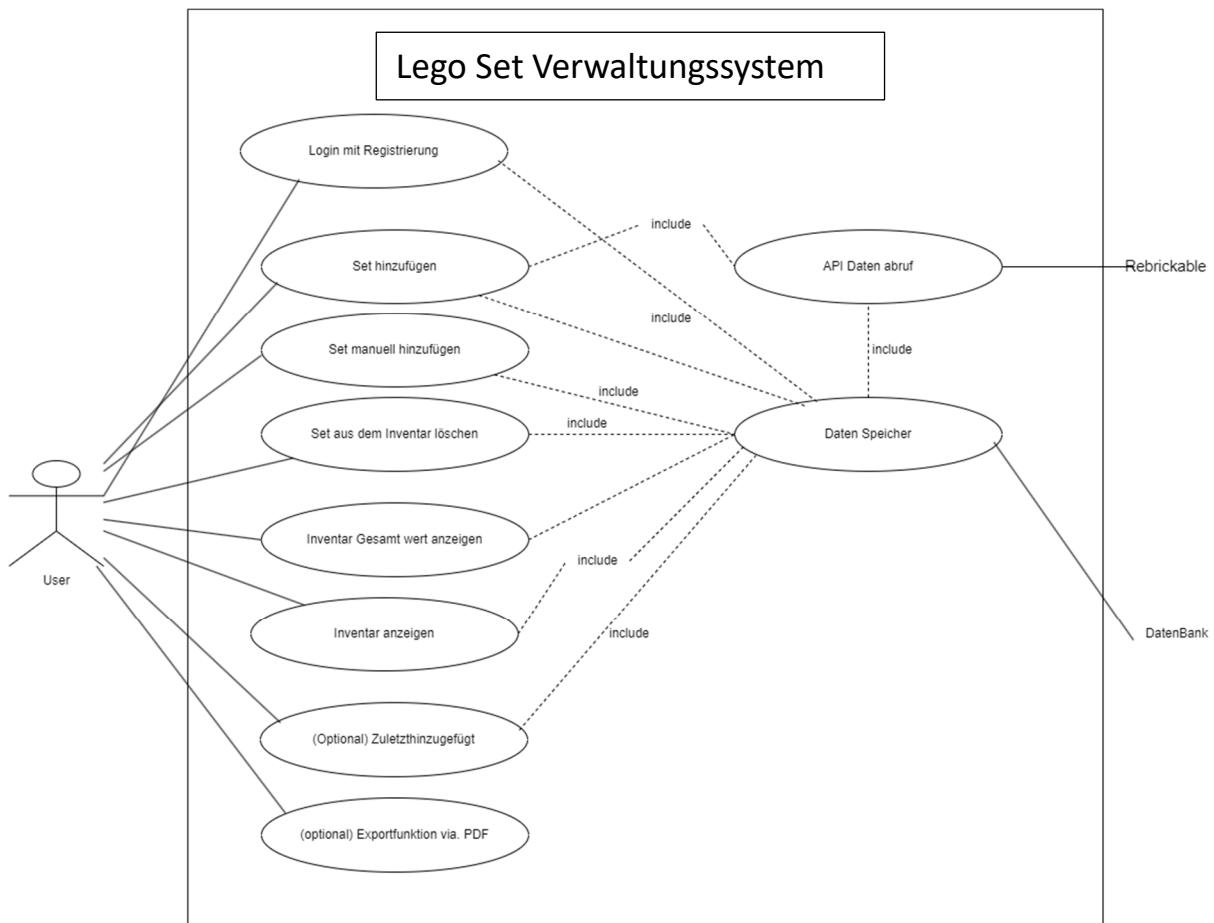
4. Dynamische GUI

- Die GUI passt sich unterschiedlichen Bildschirmgrößen an.

5. Passwort zurücksetzen

- Das Passwort via. Email zurück setzen

5. Use-Case-Diagramme



Tabellarische Beschreibung der Use-Cases

Use Case	Beschreibung
Set hinzufügen	Der Benutzer fügt ein Lego-Set über die API-Suche hinzu. Hierfür werden die Daten über den Use-Case „API-Daten abrufen“ abgerufen und gespeichert.
Set manuell hinzufügen	Der Benutzer gibt die Informationen zu einem Set manuell ein, ohne die API zu nutzen. Die eingegebenen Daten werden über „Daten speichern“ gespeichert.
Set aus dem Inventar löschen	Der Benutzer kann ein Set aus seinem Inventar entfernen. Dabei werden die Änderungen über den Use-Case „Daten speichern“ in der Datenbank aktualisiert.
Inventar Gesamtwert anzeigen	Der Benutzer kann den Gesamtwert seines Inventars basierend auf den gespeicherten Sets anzeigen lassen.
API-Daten abrufen	Für „Set hinzufügen“ werden die Daten über die API-Schnittstelle abgerufen, um

	automatisch Informationen wie Name, Nummer und Preis einzutragen.
Daten speichern	Die zentralen Datenänderungen (Hinzufügen, Löschen, Bearbeiten) werden in der Datenbank gespeichert.
Login mit Registrierung (optional) zuletzt hinzugefügt	Man kann sich anmelden und Registrieren Man kann die letzten gespeicherten Einträge anschauen
Inventar Anzeigen	Man kann sich das Inventar in einer Tabelle anzeigen
(Optional) Exportfunktion via PDF	Der Benutzer kann seine Sammlung in einem PDF-Format exportieren.

Beziehungen zwischen Use-Cases

Beziehung	Beschreibung
Set hinzufügen => API-Daten abrufen	Set hinzufügen ruft immer die API-Daten auf, um die benötigten Set-Informationen zu laden.
API-Daten abrufen => Daten Speicher	API-Daten abrufen ruft Daten Speicher auf um die API daten langfristig zu speichern das bewirkt keine langen API-Abfragen so wie ein kleines Backup falls die API mal nicht verfügbar ist
Set hinzufügen => Daten speichern	Nach dem Hinzufügen eines Sets werden die Daten in der Datenbank gespeichert.
Set manuell hinzufügen => Daten speichern	Set manuell hinzufügen ruft Daten speichern auf, um die eingegebenen Daten dauerhaft zu speichern.
Set aus dem Inventar löschen => Daten speichern	Beim Löschen eines Sets aus dem Inventar wird Daten speichern verwendet, um die Änderungen zu aktualisieren.
Login mit Registrierung => Daten speichern	Login und Registrierung rufen Daten Speicher auf um bei Registrierung Daten zu speichern und beim Login Daten Vergleichen
Inventar anzeigen => Daten speichern	Um zu schauen welche Sets in seinem Inventar sind
Inventar Gesamtwert anzeigen => Daten speichern	Der Gesamtwert wird anhand der gespeicherten Daten in der Datenbank berechnet.

6. Projektplan



7. Produktumgebung

- **C#:**

Das Projekt wird in C# programmiert

- **WPF:**

WPF wird benutzt, um eine benutzerfreundliche GUI zu schaffen

Datenbank

- **SQLite:**

Wird verwendet, um die Daten dauerhaft zu speichern.

- **Rebrickable API:**

Eine externe Schnittstelle, die genutzt wird, um Daten zu Lego-Sets wie Name, Nummer, Jahr und Thema abzurufen. Dies erleichtert das Hinzufügen von Sets und reduziert manuelle Eingaben.

- **Visual Studio:**

- Das ganze Projekt wird in Visual Studio geschrieben, getestet und gedebugget

- **Git/GitHub:**

- Zur Versionierung des Projekts wird Git/GitHub benutzt. Das Projekt wird in einem GitHub-Repository gehostet.

8. Skizze von GUI oder Webseite

Lego-Set Verwaltung

[Inventar](#)[Set Hinzufügen](#)[Export](#)[Login](#)[Set Suche](#) [Inventar](#) [Set Hinzufügen](#) [Export](#)

Set-Name oder ID suchen

[Suchen](#)

Lego-Set Verwaltung

[Inventar](#)[Set Hinzufügen](#)[Export](#)[Login](#)[Set Suche](#) [Inventar](#) [Set Hinzufügen](#) [Export](#)

Inventar

ID	Name	Wert (€)	Aktionen

Gesamtwert des Inventars: 0,00 €

Lego-Set Verwaltung

[Inventar](#)[Set Hinzufügen](#)[Export](#)[Login](#)[Set Suche](#) [Inventar](#) [Set Hinzufügen](#) [Export](#)

Set Hinzufügen

Name:

Nummer:

Genre:

Jahr:

Preis (UVP):

[Set Hinzufügen](#)

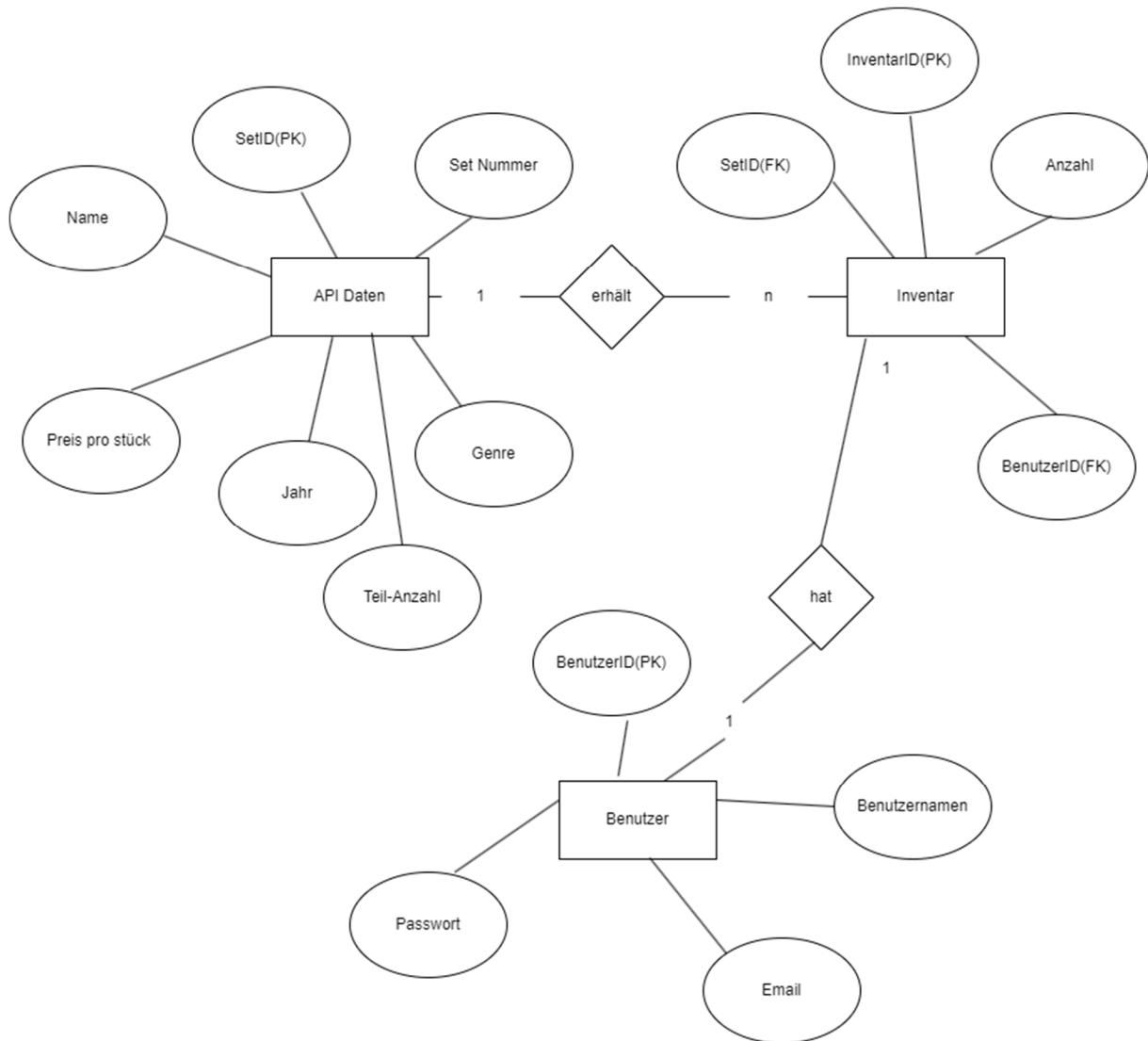
Lego-Set Verwaltung

[Inventar](#)[Set Hinzufügen](#)[Export](#)[Login](#)[Set Suche](#) [Inventar](#) [Set Hinzufügen](#) [Export](#)

Inventar Export

[Inventar als PDF exportieren](#)

9. DB-Entwurf



10. Link zu einem gehosteten Git-Repository

<https://github.com/FloFritz/Lego-Set-Verwaltungssystem>

11. Testplan

ID	T01	
Beschreibung	Ein Benutzer registriert sich mit Benutzernamen, E-Mail und Passwort.	
Vorbedingung	Das Registrierungsformular ist geöffnet.	
Test-Schritte	1. Benutzer gibt gültigen Benutzernamen, E-Mail und Passwort ein. 2. Benutzer klickt auf „Registrieren“.	
Erwartetes Resultat	Der Benutzer wird erfolgreich in der Datenbank gespeichert und erhält eine Bestätigung.	

ID	T02	<input type="checkbox"/>
Beschreibung	Ein Benutzer versucht, sich mit einem bereits registrierten Benutzernamen zu registrieren.	
Vorbedingung	Der Benutzername existiert bereits in der Datenbank.	
Test-Schritte	1. Benutzer gibt existierenden Benutzernamen ein. 2. Benutzer klickt auf „Registrieren“.	
Erwartetes Resultat	Das System gibt eine Fehlermeldung zurück: „Benutzername bereits vergeben.“	
ID	T03	<input type="checkbox"/>
Beschreibung	Ein registrierter Benutzer loggt sich ein.	
Vorbedingung	Der Benutzer existiert in der Datenbank und ist registriert.	
Test-Schritte	1. Benutzer gibt Benutzernamne und Passwort ein. 2. Benutzer klickt auf „Login“.	
Erwartetes Resultat	Der Benutzer wird eingeloggt und das Inventar wird angezeigt.	
ID	T04	<input type="checkbox"/>
Beschreibung	Ein Benutzer versucht, sich mit einem falschen Passwort einzuloggen.	
Vorbedingung	Der Benutzername existiert in der Datenbank, aber das Passwort ist falsch.	
Test-Schritte	1. Benutzer gibt Benutzernamen und falsches Passwort ein. 2. Benutzer klickt auf „Login“.	
Erwartetes Resultat	Das System gibt eine Fehlermeldung zurück: „Falsches Passwort.“	
ID	T05	<input type="checkbox"/>
Beschreibung	Der Benutzer fügt ein Set manuell hinzu.	
Vorbedingung	Der Benutzer ist eingeloggt.	
Test-Schritte	1. Benutzer gibt Name, Nummer, Thema, Jahr und Preis ein. 2. Benutzer klickt auf „Hinzufügen“.	
Erwartetes Resultat	Das Set wird erfolgreich zum Inventar hinzugefügt.	
ID	T06	<input type="checkbox"/>
Beschreibung	Ein Set wird aus dem Inventar gelöscht.	
Vorbedingung	Das Set existiert im Inventar.	
Test-Schritte	1. Benutzer wählt das Set aus. 2. Benutzer klickt auf „Löschen“.	
Erwartetes Resultat	Das Set wird aus der Datenbank entfernt und nicht mehr angezeigt	
ID	T07	<input type="checkbox"/>
Beschreibung	Der Gesamtwert des Inventars wird berechnet und angezeigt.	
Vorbedingung	Es existieren mindestens zwei Sets mit Preis und Anzahl im Inventar.	
Test-Schritte	1. Benutzer öffnet die Inventarseite. 2. Benutzer überprüft den Gesamtwert.	
Erwartetes Resultat	Der Gesamtwert entspricht der Summe der Einzelwerte (Preis * Anzahl).	
ID	T08	<input type="checkbox"/>
Beschreibung	Ein Set wird über die API hinzugefügt, einschließlich der Bild-URL.	
Vorbedingung	Die API liefert gültige Daten für das Set.	
Test-Schritte	1. Benutzer gibt die Set-Nummer ein. 2. System ruft Daten von der API ab. 3. Benutzer bestätigt das Hinzufügen.	
Erwartetes Resultat	Das Set, einschließlich der Bild-URL, wird in der Datenbank gespeichert.	
ID	T09	<input type="checkbox"/>
Beschreibung	Benutzer sieht die letzten 5 hinzugefügten Sets auf der Startseite.	
Vorbedingung	Es existieren mindestens 5 Sets in der Datenbank mit Bild-URLs.	
Test-Schritte	1. Benutzer öffnet die Anwendung. 2. Benutzer prüft die Startseite. 3. Bilder und Namen der Sets werden angezeigt.	
Erwartetes Resultat	Die letzten 5 Sets werden korrekt angezeigt (Name und Bild).	
ID	T10 (optional)	<input type="checkbox"/>
Beschreibung	Ein Benutzer setzt sein Passwort zurück.	
Vorbedingung	Der Benutzer existiert in der Datenbank.	
Test-Schritte	1. Benutzer klickt auf „Passwort vergessen“. 2. System sendet eine E-Mail mit einem Link. 3. Benutzer öffnet den Link und gibt ein neues Passwort ein.	
Erwartetes Resultat	Das Passwort wird aktualisiert und der Benutzer kann sich mit dem neuen Passwort einloggen.	

ID	T11 (optional)	
Beschreibung	Der Benutzer exportiert das aktuelle Inventar in eine PDF-Datei.	<input type="checkbox"/>
Vorbedingung	- Der Benutzer ist eingeloggt. - Es befinden sich Sets im Inventar.	
Test-Schritte	1. Der Benutzer öffnet die Inventarseite. 2. Der Benutzer klickt auf den Button „Exportieren als PDF“. 3. Das System generiert eine PDF-Datei mit den Details der Sets (z. B. Name, Anzahl, Preis). 4. Der Benutzer prüft, ob die PDF-Datei korrekt erstellt wurde und alle Informationen enthält.	
Erwartetes Resultat	Eine PDF-Datei wird erfolgreich erstellt und gespeichert. Die Datei enthält die Inventardetails in tabellarischer Form.	
ID	T12(optional)	
Beschreibung	Der Benutzer versucht, ein leeres Inventar als PDF zu exportieren.	<input type="checkbox"/>
Vorbedingung	- Der Benutzer ist eingeloggt. - Es befinden sich keine Sets im Inventar.	
Test-Schritte	1. Der Benutzer öffnet die Inventarseite. 2. Der Benutzer klickt auf den Button „Exportieren als PDF“.	
Erwartetes Resultat	Das System zeigt eine Meldung an: „Kein Inventar zum Exportieren vorhanden.“ Keine PDF-Datei wird erstellt.	

UnitTests

1. Registrierung

- Ziel:** Sicherstellen, dass die Registrierung fehlerfrei funktioniert.
- Testfälle:**
 - Erfolgreiche Registrierung eines neuen Benutzers.
 - Fehler bei der Registrierung mit bereits existierendem Benutzernamen oder E-Mail.

2. Login

- Ziel:** Prüfen, ob Benutzer erfolgreich eingeloggt werden und Fehler korrekt behandelt werden.
- Testfälle:**
 - Erfolgreicher Login mit gültigen Benutzerdaten.
 - Fehler bei falschem Passwort.
 - Fehler bei nicht existierendem Benutzer.

3. Inventarverwaltung

- Ziel:** Sicherstellen, dass Sets korrekt hinzugefügt, gelöscht und angezeigt werden.
- Testfälle:**
 - Hinzufügen eines neuen Sets, sowohl über die API als auch manuell.
 - Verhindern von doppelten Einträgen im Inventar.
 - Löschen eines Sets aus dem Inventar.

4. Berechnung des Inventarwerts

- Ziel:** Überprüfen, ob der Gesamtwert des Inventars korrekt berechnet wird.

- **Testfälle:**
 - Korrekte Berechnung des Wertes bei mehreren Sets.
 - Fehlerfreies Verhalten bei leerem Inventar.

5. (Optional) Anzeige der zuletzt hinzugefügten Sets

- **Ziel:** Sicherstellen, dass die letzten 5 hinzugefügten Sets korrekt angezeigt werden.
- **Testfälle:**
 - Anzeige der Sets in der korrekten Reihenfolge (nach Hinzufügedatum).
 - Korrekte Anzeige bei weniger als 5 vorhandenen Sets.

6. (Optional) Passwort zurücksetzen

- **Ziel:** Prüfen, ob Benutzer ihr Passwort sicher zurücksetzen können.
- **Testfälle:**
 - Senden einer E-Mail mit einem Reset-Link.

Verwendete Hilfsmittel und Quellen

1. Entity Framework Core
2. Rebrickable-API
 - 2.1. <https://rebrickable.com/api/>
3. WPF
4. SQLite
5. DB Browser(SQLite)
6. Lern Microsoft
 - 6.1. <https://learn.microsoft.com/de-de/search/?terms=c%23>
7. ChatGPT
 - 7.1. <https://openai.com/index/chatgpt/>
8. GitHub
 - 8.1. <https://github.com/>

Fazit

Das Projektziel, eine funktionsfähige und benutzerfreundliche Software zur Verwaltung von Lego-Sets zu entwickeln, wurde erfolgreich erreicht. Die Anwendung deckt alle geplanten Anforderungen ab, darunter die Verwaltung eigener Sets, eine automatische API-gestützte Datenerfassung sowie das Speichern zusätzlicher Informationen wie Preis, Menge und Notizen.

Während der Umsetzung konnten sowohl technische als auch konzeptionelle Fähigkeiten vertieft werden – insbesondere im Umgang mit Datenbanken, Schnittstellen und Benutzeroberflächen. Die Anwendung ist erweiterbar, stabil und praxisnah nutzbar.

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, dass ich diese Hausarbeit selbstständig und ohne fremde Hilfe angefertigt habe und keine anderen als die angegebenen Hilfsmittel und Quellen genutzt habe.

Unterschrift