



# HBFSWI

**HÖHERE BERUFSFACHSCHULE FÜR WIRTSCHAFTSINFORMATIK**

## Hausarbeit

im Bildungsgang  
„Staatlich geprüfte/r Wirtschaftsinformatiker/in“  
gemäß §5 der Ausbildungs- und Prüfungsordnung

### Entwicklung eines Garbage-Collection-Tools zur Löschung temporärer Dateien

vorgelegt von: Moritz Nicola Kreis  
Klasse: WI23ZA1  
Adresse: Blumenstraße 5  
Ort: 66583 Spiesen-Elversberg  
E-Mail: moritz.nicola.kreis@web.de  
Abgabetermin: 15.05.2025  
Betreuer/in: Herr Friedrich

# Inhaltsverzeichnis

<b>Vorwort</b>	<b>1</b>
<b>1 Lasten- und Pflichtenheft</b>	<b>1</b>
1.1 Einführung . . . . .	1
1.2 Ist-Situation . . . . .	2
1.3 Soll-Situation . . . . .	2
1.3.1 Soll-Zustand . . . . .	2
1.3.2 Funktionale Anforderungen . . . . .	3
1.3.3 Nicht-funktionale Anforderungen . . . . .	4
1.3.4 Schnittstellen . . . . .	4
1.3.5 Risiken . . . . .	4
1.4 Abnahmekriterien . . . . .	5
1.5 Use-Case-Diagramm und Use-Cases . . . . .	6
1.6 Projektplan . . . . .	12
1.7 Produktumgebung . . . . .	12
1.8 Skizze des GUI . . . . .	13
1.9 DB-Entwurf . . . . .	15
1.10 Link zum Git-Repository . . . . .	16
1.11 Testplan . . . . .	16
1.12 Unit Tests . . . . .	19
<b>2 Benutzerhandbuch</b>	<b>19</b>
2.1 Start und Anmeldung . . . . .	19
2.2 Einstellungen konfigurieren . . . . .	21
2.3 Bereinigungsfunktionen . . . . .	22
2.4 Administration . . . . .	25

2.5	Abmeldung und Beenden . . . . .	26
<b>3</b>	<b>Implementierung</b>	<b>27</b>
3.1	Architekturüberblick . . . . .	27
3.2	Datenbankanbindung und Docker . . . . .	30
<b>4</b>	<b>Tests</b>	<b>31</b>
<b>5</b>	<b>Installationsanleitung</b>	<b>32</b>
5.1	Systemvoraussetzungen . . . . .	32
5.2	Vorbereitung der Datenbank . . . . .	33
5.3	Konfiguration und Verbindung . . . . .	33
<b>6</b>	<b>Quellcode</b>	<b>33</b>
6.1	Die Klasse CleanupVM.cs im Garbage-Collector . . . . .	33
6.1.1	Initialisierung . . . . .	33
6.1.2	Manuelle Bereinigung . . . . .	35
6.1.3	Junk-Dateien erkennen und bereinigen . . . . .	36
6.1.4	Duplikate erkennen und entfernen . . . . .	38
6.2	Authentifizierung und Registrierung . . . . .	41
6.2.1	Registrierungsvorgang . . . . .	41
6.2.2	Sicheres Passwort-Hashing . . . . .	42
6.2.3	Anmeldevorgang . . . . .	43
<b>7</b>	<b>Ausblick und Fazit</b>	<b>45</b>
	<b>Quellen und Werkzeuge</b>	<b>46</b>

# Abbildungsverzeichnis

1	Use-Case-Diagramm des Garbage-Collection-Tools . . . . .	6
2	Projektplan als Gantt-Diagramm . . . . .	12
3	GUI-Ansicht: Home . . . . .	13
4	GUI-Ansicht: Einstellungen . . . . .	13
5	GUI-Ansicht: Cleanup . . . . .	14
6	GUI-Ansicht: Login . . . . .	14
7	ER-Diagramm der GarbageCollector-Datenbank . . . . .	15
8	Login-Maske mit Eingabefeldern für Benutzername und Passwort . . . . .	19
9	Registrierungsformular mit Passwortbestätigung . . . . .	20
10	Home-Bildschirm des Nutzers mit dem gesetzten Benutzernamen „moritz“ . . . . .	21
11	Einstellungsseite mit Optionen zur Konfiguration und Passwortänderung . . . . .	22
12	Cleanup-Seite mit Optionen zur manuellen und automatischen Bereinigung . . . . .	23
13	Cleanup-Seite mit laufendem Bereinigungsprozess und aktivem Scheduler . . . . .	24
14	Beispiel einer möglichen <code>config.json</code> . . . . .	24
15	Administrationsbereich zur Benutzerverwaltung mit Rollenzuweisung . . . . .	26
16	DataTemplate für die View-Bindung von HomeVM . . . . .	27
17	Command-Implementierung für Navigation . . . . .	28
18	ContentControl für dynamische View-Anzeige . . . . .	28
19	Grundstruktur eines RelayCommands . . . . .	29
20	PropertyChanged-Implementierung . . . . .	29
21	Architekturübersicht des Garbage-Collection-Tools (MVVM) . . . . .	30
22	Beispiel für einen <code>docker run</code> -Befehl zur Bereitstellung eines SQL-Servers . . . . .	31
23	Screenshot des Konstruktors von CleanupVM . . . . .	34
24	Screenshot der Methode <code>CleanupAsync</code> . . . . .	36
25	Screenshot der Methode <code>CleanJunkFilesAsync</code> . . . . .	37

26	Screenshot der Methode <code>IsJunkFile</code> . . . . .	37
27	Methode <code>RemoveDuplicateFilesAsync</code> – Teil 1: Hashing und Duplikaterkennung . . . . .	39
28	Methode <code>RemoveDuplicateFilesAsync</code> – Teil 2: Duplikatlöschung und Abschluss . . . . .	40
29	Screenshot der Methode <code>ComputeFileHash</code> . . . . .	41
30	Auszug aus der Methode <code>ExecuteRegister</code> . . . . .	42
31	Implementierung des Passwort-Hashings mit Argon2 . . . . .	43
32	Passwortverifikation durch erneutes Hashing mit Argon2 . . . . .	44

# Vorwort

In einer zunehmend digitalisierten Welt werden analoge Prozesse durch Softwarelösungen abgebildet, automatisiert und optimiert. Die Relevanz digitaler Anwendungen hat in den vergangenen Jahren stark zugenommen.

Auch bei **DataFlowSolutions** wird täglich mit großen Datenmengen gearbeitet, die durch verschiedenste betriebliche Prozesse entstehen. Neben relevanten und dauerhaft benötigten Informationen sammeln sich jedoch zunehmend temporäre oder überholte Daten an – sogenannte „Datenreste“ oder „Datenmüll“. Diese Dateien verlieren nach ihrer Nutzung schnell an Bedeutung, belegen jedoch weiterhin wertvollen Speicherplatz und beeinträchtigen langfristig die Systemleistung.

Aus dieser Problematik heraus entstand die Idee, ein maßgeschneidertes Garbage-Collection-Tool zu entwickeln, das diese temporären Dateien identifiziert und automatisiert bereinigt. Ziel war es, ein benutzerfreundliches Werkzeug zu schaffen, das flexibel auf unterschiedliche Dateitypen und Verzeichnisse reagieren kann, sowohl manuell als auch zeitgesteuert.

Mit dieser Hausarbeit wird nicht nur ein konkretes Softwareprodukt dokumentiert, sondern auch ein praxisnahes Problem adressiert, das in vielen Unternehmen von großer Bedeutung ist.

## 1 Lasten- und Pflichtenheft

### 1.1 Einführung

Die **DataFlow Solutions GmbH** ist ein IT-Unternehmen, das sich auf die Bereitstellung fortschrittlicher Datenanalyse- und Cloud-Computing-Dienstleistungen spezialisiert hat. Das Unternehmen operiert global und bedient eine breite Palette von Kunden, darunter große Konzerne, mittelständische Unternehmen und Start-ups.

Mit rund 500 Mitarbeitenden, darunter Datenwissenschaftler, Cloud-Spezialisten und Softwareentwickler, bietet DataFlow Solutions umfassende Lösungen in Bereichen wie *Big Data*, *maschinelles Lernen* und *Datenmigration* an. Das Unternehmen zeichnet sich durch seine Expertise in der Analyse und Verarbeitung großer Datenmengen aus und bietet sowohl standardisierte als auch maßgeschneiderte Lösungen an.

Der Hauptsitz befindet sich in Saarbrücken, im Herzen Europas, ergänzt durch weitere Büros in den USA, Großbritannien und Asien, was eine starke internationale Präsenz gewährleistet.

Ziel des Unternehmens ist es, eine führende Position in der IT-Branche einzunehmen, indem kontinuierlich in Forschung und Entwicklung investiert und durch enge Zusammenarbeit mit Kunden innovative Lösungen geschaffen werden, die den aktuellen technologischen Trends entsprechen. Durch seine globale Aufstellung und sein breit gefächertes

Dienstleistungsportfolio hat sich DataFlow Solutions als zuverlässiger Partner etabliert und strebt eine stetige Erweiterung seiner Marktpräsenz an.

## 1.2 Ist-Situation

Mit zunehmendem Wachstum und der Erweiterung ihrer Dienstleistungen stößt die **DataFlow Solutions GmbH** auf ein zentrales Problem:

Die Server und Rechner des Unternehmens sammeln im Laufe der Zeit eine große Menge an temporären und überflüssigen Dateien an, was zu einer spürbaren Verlangsamung der Systeme führt. Die IT-Infrastruktur leidet unter verminderter Leistungsfähigkeit und Zuverlässigkeit. Dies äußert sich unter anderem in verlängerten Ladezeiten bei Datenanalysen und beeinträchtigt die Effizienz der Cloud-Dienste – mit potenziell negativen Folgen für die Kundenzufriedenheit und damit einhergehenden finanziellen Verlusten.

Um dieser Entwicklung entgegenzuwirken, plant DataFlow Solutions die Einführung eines *Garbage-Collection-Tools*, das automatisch temporäre und unnötige Dateien erkennt und entfernt. Ziel ist es, die Systemleistung nachhaltig zu optimieren und gleichzeitig die Effizienz der Infrastruktur zu verbessern.

Durch die Implementierung dieser Lösung soll nicht nur die interne IT-Performance gesteigert, sondern auch die Kundenzufriedenheit erhöht und die operative Exzellenz des Unternehmens langfristig gesichert werden. Zudem erhofft sich DataFlow Solutions eine Stärkung seiner Position im zunehmend wettbewerbsintensiven Markt für Datenanalyse- und Cloud-Dienstleistungen.

## 1.3 Soll-Situation

### 1.3.1 Soll-Zustand

Die für DataFlow Solutions entwickelte Software soll folgende Leistungen erbringen und Mehrwert für das Unternehmen bieten:

- **Effizienzsteigerung:** Die Systeme des Unternehmens arbeiten wesentlich effizienter, unnötige Daten sollen regelmäßig und automatisch bereinigt werden können, was zu einer spürbaren Beschleunigung der Datenverarbeitung und Analyse führt.
- **Erhöhte Produktivität:** Mitarbeiter können sich auf Kernaufgaben konzentrieren, anstatt Zeit mit der manuellen Bereinigung von Daten zu verbringen
- **Verbesserte Kundenzufriedenheit:** Durch die gesteigerte Leistungsfähigkeit der Datenverarbeitung- und Analyse können die Kundenanforderungen schneller und präziser erfüllt werden, was zu einer höheren Kundenzufriedenheit führt.
- **Langfristige Kosteneinsparungen:** Durch die Reduzierung von Systemausfallzeiten und die Vermeidung von Leistungsproblemen werden langfristig Kosten gespart.

- **Verbesserte Lebensdauer der Hardware:** Die regelmäßige Entfernung von Datenmüll trägt dazu bei, die Lebensdauer der Hardware zu verlängern, indem Überlastungen und Verschleiß verringert werden.
- **Nachhaltigkeit:** Das Tool unterstützt das Unternehmen in seiner Bestrebung umweltfreundlicher zu agieren, indem es zur Effizienzsteigerung der IT-Ressourcen beiträgt.

Der Mehrwert, der durch die Nutzung der neuen Software bei DataFlow Solutions generiert wird, liegt vor allem in der deutlichen Steigerung der Effizienz und Produktivität. Durch die automatisierte Bereinigung von Systemdaten werden IT-Ressourcen entlastet, wodurch sich die Leistungsfähigkeit der gesamten IT-Infrastruktur verbessert. Dies führt nicht nur zu schnelleren und effektiveren Arbeitsabläufen innerhalb des Unternehmens, sondern erhöht auch die Zuverlässigkeit und Qualität der Dienstleistungen, die DataFlow Solutions seinen Kunden bietet. Langfristig trägt die Software damit zur Steigerung der Wettbewerbsfähigkeit und zur Kosteneinsparung bei.

### 1.3.2 Funktionale Anforderungen

Für das Garbage-Collection-Tool bei DataFlow Solutions ergeben sich auf Grundlage der Implementierung die folgenden funktionalen Anforderungen sowie deren geschätzte Entwicklungszeiten:

- **Verzeichnisauswahl und Musterfilterung:** Benutzer können ein Zielverzeichnis angeben und definieren, nach welchen Dateitypen (z. B. `.tmp`, `.log`) gesucht werden soll. *(20 Stunden)*
- **Erkennung veralteter Dateien:** Dateien werden anhand ihres Änderungsdatums mit einem einstellbaren Schwellenwert (z. B. älter als 10 Tage) gefiltert. *(10 Stunden)*
- **Automatische und manuelle Bereinigung:** Neben dem manuellen Start des Prozesses wird auch ein zeitgesteuerter, wiederkehrender Cleanup mittels Scheduler unterstützt. *(25 Stunden)*
- **Papierkorb statt Direktlöschung:** Gefundene Dateien können zunächst, je nach Einstellung, in den Papierkorb verschoben werden, um versehentliche Datenverluste zu vermeiden. *(15 Stunden)*
- **Einstellungen über Konfigurationsdatei:** Alle Parameter wie Pfad, Dateialter, Dateitypen werden über eine Konfigurationsdatei verwaltet. *(35 Stunden)*
- **Fortschrittsanzeige und Statusfeedback:** Der Benutzer erhält visuelles Feedback über den Status des Prozesses, inklusive Fortschrittsanzeige. *(20 Stunden)*
- **Logik für Doppelte Dateien:** Es ist eine Erweiterung geplant, um identische Dateien anhand von Hashwerten zu erkennen und als Duplikate zu behandeln. *(30 Stunden)*



### 1.3.3 Nicht-funktionale Anforderungen

Für das Garbage-Collection-Tool von DataFlow Solutions ergeben sich die folgenden nicht-funktionalen Anforderungen sowie geschätzte Entwicklungszeiten:

- **Optik gemäß Corporate-Design:** Die grafische Benutzeroberfläche wird unter Einhaltung der firmeninternen Designrichtlinien entwickelt (Farbschema, Icons, Logo, Layout). *(20 Stunden)*
- **Barrierefreie Benutzerführung:** Die WPF-Oberfläche soll auch für Nutzer mit eingeschränkter Sehfähigkeit gut nutzbar sein, z. B. durch kontrastreiche Darstellung, Tastaturnavigation und verständliche Statusmeldungen. *(25 Stunden)*
- **Performance bei großen Dateibeständen:** Die Anwendung verarbeitet mehrere Tausend Dateien performant und ohne merkliche Verlangsamung der Benutzeroberfläche. Dies wird durch den Einsatz von `async / await` und effizienter Dateifilterung gewährleistet. *(30 Stunden)*
- **Kompatibilität mit Windows-Systemen:** Die Software läuft zuverlässig unter Windows 10 und 11 und nutzt systemeigene Komponenten wie den Windows-Papierkorb oder den Datei-Explorer. *(35 Stunden)*

### 1.3.4 Schnittstellen

Das Garbage-Collection-Tool kommuniziert mit einer extern betriebenen MSSQL-Server-Datenbank. Diese wird aktuell in einem Linux-basierten Docker-Container ausgeführt.

Die Datenbank dient der Speicherung von Benutzerdaten, Konfigurationen sowie Protokollen über durchgeführte Bereinigungsvorgänge. Obwohl der Datenbankserver im Projektszenario lokal bereitgestellt wird, erfolgt die Anbindung netzwerkbasiert und kann daher als **externe Schnittstelle** bewertet werden.

In einem produktiven Unternehmensumfeld wäre eine ähnliche Infrastruktur üblich: Die Anwendung würde auf einem Client laufen, während die SQL-Datenbank zentral auf einem Datenbankserver innerhalb des Firmennetzwerks betrieben wird. Die Kommunikation würde ebenfalls über das Netzwerk erfolgen.

### 1.3.5 Risiken

Für das Garbage-Collection-Tool werden folgende Risiken und entsprechende Maßnahmen in Betracht gezogen:

- **Technische Inkompatibilität:** Das Tool könnte auf bestimmten Systemen oder Betriebssystemversionen nicht ordnungsgemäß funktionieren.

*Maßnahme:* Durchführung umfassender Kompatibilitätstests in der Entwicklungsphase, um technische Probleme bei der Ausführung frühzeitig zu erkennen und zu beheben.

- **Datenverlust:** Es besteht das Risiko, dass versehentlich wichtige Dateien gelöscht werden, die weiterhin benötigt werden.

*Maßnahme:* Anstatt Dateien direkt zu löschen, werden diese zunächst in den Papierkorb verschoben. Zusätzlich sorgen Sicherheitsprüfungen (z. B. durch geschützte Verzeichnisse oder Dateifilter) dafür, dass keine kritischen Dateien unbeabsichtigt entfernt werden.

- **Geringe Nutzerakzeptanz:** Die Bedienung könnte für Endnutzer nicht intuitiv genug sein oder nicht deren Erwartungen entsprechen.

*Maßnahme:* Die Einbindung der Zielnutzer in den Entwicklungsprozess sowie regelmäßiges Einholen von Feedback erhöhen die Nutzerfreundlichkeit. Die Erstellung einer leicht verständlichen Anleitung unterstützt zusätzlich den effektiven Einsatz des Tools.

## 1.4 Abnahmekriterien

### Muss-Kriterien (essenziell für die Projektabnahme)

- **Effektive Datenbereinigung:** Das Tool muss temporäre und unnötige Dateien zuverlässig identifizieren und entfernen können.
- **Übereinstimmung mit dem Corporate-Design:** Die Benutzeroberfläche der Software muss den visuellen Richtlinien des Unternehmens entsprechen.
- **Kompatibilität:** Die Anwendung muss auf den im Unternehmen eingesetzten Systemen lauffähig sein und stabil arbeiten.
- **Benutzerdefinierte Einstellungen:** Benutzer müssen in der Lage sein, den Bereinigungsprozess über eine Konfigurationsdatei oder GUI-Optionen individuell anzupassen.

### Kann-Kriterien (wünschenswert, aber nicht zwingend für die Abnahme)

- **Erweiterte Berichtsfunktionen:** Ausgabe detaillierter Informationen zu den durchgeführten Löschvorgängen und der Menge des freigegebenen Speicherplatzes.
- **Echtzeit-Monitoring und Warnungen:** Anzeige des aktuellen Bereinigungsstatus sowie Benachrichtigungen bei Problemen (wie zum Beispiel, wenn keine Dateien gefunden wurden).
- **Log-Export:** Möglichkeit zum Export der Bereinigungsprotokolle in ein CSV-Format.

- **Duplikaterkennung und -entfernung:** Identifikation und optionales Entfernen von mehrfach vorhandenen Dateien durch Hashvergleich.

## 1.5 Use-Case-Diagramm und Use-Cases

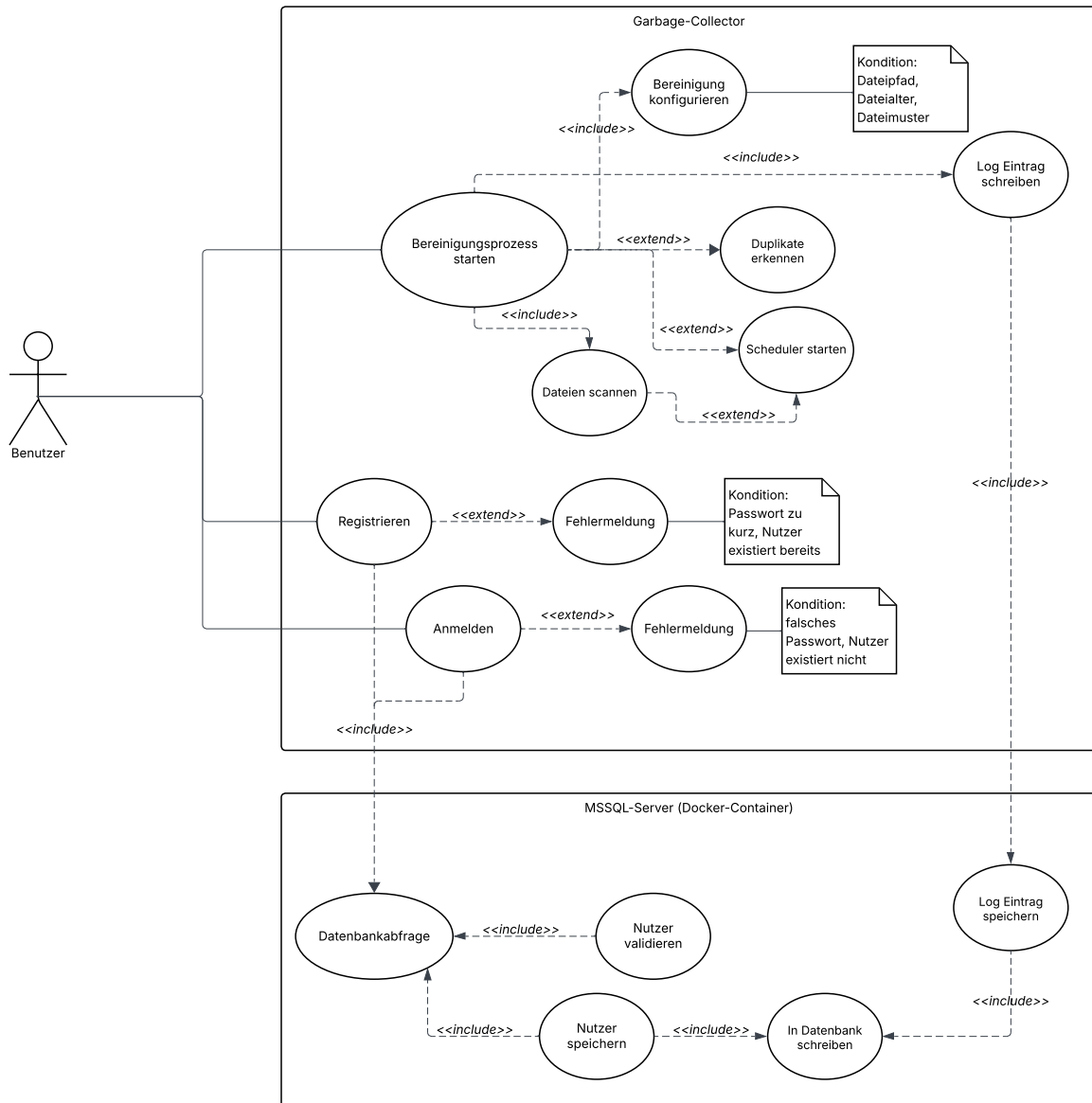


Abbildung 1: Use-Case-Diagramm des Garbage-Collection-Tools

*Hinweis: Das dargestellte Use-Case-Diagramm fokussiert sich auf die zentralen Anwendungsfälle des Garbage-Collection-Tools. Zusätzliche Funktionen wie Passwortänderung, Adminbereich oder weitere erweiterte Funktionen wurden aus Gründen der Übersichtlichkeit nicht aufgenommen und werden im weiteren Verlauf detaillierter beschrieben.*

## Use Case: Benutzer registrieren

<b>Kurzbeschreibung</b>	Ein Benutzer erstellt ein neues Benutzerkonto im Garbage-Collector-Tool.
<b>Voraussetzung</b>	Der Benutzer ist noch nicht registriert.
<b>Nachbedingung</b>	Der Benutzer ist erfolgreich in der Datenbank gespeichert und kann sich künftig anmelden.
<b>Fehlersituation</b>	Das eingegebene Passwort ist zu kurz oder der Benutzername existiert bereits.
<b>Systemzustand im Fehlerfall</b>	Der Benutzer wird nicht registriert, eine Fehlermeldung wird angezeigt.
<b>Akteure</b>	Benutzer
<b>Trigger</b>	Der Benutzer klickt auf „Registrieren“ und füllt das Formular aus.
<b>Standardablauf</b>	<ol style="list-style-type: none"><li>1. Der Benutzer öffnet das Garbage-Collector-Tool.</li><li>2. Der Benutzer klickt auf „Registrieren“.</li><li>3. Das Tool prüft die Eingaben (z. B. Passwortlänge, eindeutiger Benutzername).</li><li>4. Die Daten werden über eine Datenbankabfrage gespeichert.</li><li>5. Der Benutzer erhält eine Bestätigung zur erfolgreichen Registrierung.</li></ol>
<b>Alternativabläufe</b>	<ul style="list-style-type: none"><li>• Das Passwort ist zu kurz.</li><li>• Der Benutzername ist bereits vergeben.</li><li>• Eine Fehlermeldung wird angezeigt und der Benutzer muss die Eingaben korrigieren.</li></ul>

## Use Case: Benutzer anmelden

<b>Kurzbeschreibung</b>	Ein Benutzer meldet sich mit gültigen Zugangsdaten am System an.
<b>Voraussetzung</b>	Der Benutzer ist registriert und besitzt gültige Zugangsdaten.
<b>Nachbedingung</b>	Der Benutzer ist authentifiziert und erhält Zugriff auf die Funktionen der Software.
<b>Fehlersituation</b>	Das Passwort ist falsch oder der Benutzer existiert nicht.
<b>Systemzustand im Fehlerfall</b>	Keine Anmeldung, Zugriff wird verweigert, Fehlermeldung wird angezeigt.
<b>Akteure</b>	Benutzer
<b>Trigger</b>	Der Benutzer klickt auf „Anmelden“ und gibt Zugangsdaten ein.
<b>Standardablauf</b>	<ol style="list-style-type: none"><li>1. Der Benutzer öffnet das Tool und wählt „Anmelden“.</li><li>2. Die Zugangsdaten werden über eine Datenbankabfrage geprüft.</li><li>3. Die Anmeldung wird bestätigt.</li><li>4. Der Benutzer hat Zugriff auf die Funktionen.</li></ol>
<b>Alternativabläufe</b>	<ul style="list-style-type: none"><li>• Falsches Passwort oder nicht existierender Benutzer.</li><li>• Fehlermeldung wird angezeigt, keine Anmeldung erfolgt.</li></ul>

## Use Case: Duplikate erkennen

<b>Kurzbeschreibung</b>	Das Tool durchsucht die zu bereinigenden Dateien nach Duplikaten.
<b>Voraussetzung</b>	Die Bereinigung wurde gestartet.
<b>Nachbedingung</b>	Gefundene Duplikate werden zur weiteren Bearbeitung markiert oder entfernt.
<b>Fehlersituation</b>	Das System kann keine Duplikate erkennen (z. B. durch falsche Pfadangabe).
<b>Systemzustand im Fehlerfall</b>	Es werden keine Duplikate entfernt.
<b>Akteure</b>	Benutzer
<b>Trigger</b>	Der Benutzer startet die Bereinigung, das Duplikat-Erkennungsmodul wird automatisch aktiviert.
<b>Standardablauf</b>	<ol style="list-style-type: none"><li>1. Der Benutzer startet die Bereinigung.</li><li>2. Das System durchsucht die angegebenen Verzeichnisse.</li><li>3. Duplikate werden erkannt und entfernt oder dem Benutzer angezeigt.</li><li>4. Ein Log-Eintrag wird erstellt.</li></ol>
<b>Alternativabläufe</b>	<ul style="list-style-type: none"><li>• Keine Duplikate gefunden.</li><li>• Das Duplikatmodul schlägt fehl, Log wird trotzdem erstellt.</li></ul>

## Use Case: Scheduler starten

<b>Kurzbeschreibung</b>	Das Tool aktiviert eine Zeitsteuerung zur automatischen Durchführung von Bereinigungen.
<b>Voraussetzung</b>	Der Benutzer hat eine Konfiguration und den Zeitintervall festgelegt.
<b>Nachbedingung</b>	Der Scheduler ist aktiv und führt Bereinigungen gemäß Plan durch.
<b>Fehlersituation</b>	Es wurde kein oder ein falscher Pfad angegeben
<b>Systemzustand im Fehlerfall</b>	Keine automatische Bereinigung erfolgt.
<b>Akteure</b>	Benutzer
<b>Trigger</b>	Der Benutzer speichert eine Zeitsteuerung im Tool und startet den Scheduler.
<b>Standardablauf</b>	<ol style="list-style-type: none"><li>1. Der Benutzer öffnet die Einstellungen.</li><li>2. Er konfiguriert einen automatischen Zeitplan.</li><li>3. Der Benutzer aktiviert den internen Scheduler.</li><li>4. Die geplanten Bereinigungen werden zum festgelegten Zeitintervall durchgeführt.</li></ol>
<b>Alternativabläufe</b>	<ul style="list-style-type: none"><li>• Ungültige Zeitangabe verhindert die Aktivierung.</li><li>• Fehlerhafte Konfiguration wird vom Tool gemeldet.</li></ul>

## Use Case: Log-Eintrag schreiben und speichern

<b>Kurzbeschreibung</b>	Bei jedem Bereinigungsvorgang wird ein Log-Eintrag erstellt und in der Datenbank gespeichert.
<b>Voraussetzung</b>	Eine der drei Bereinigungstypen (Standard, Duplikate, Junk-Dateien) oder Scheduler wurden ausgeführt
<b>Nachbedingung</b>	Ein entsprechender Eintrag befindet sich in der Log-Datenbank und steht zum Download bereit
<b>Fehlersituation</b>	Die Verbindung zur Datenbank ist unterbrochen oder Schreibrechte fehlen.
<b>Systemzustand im Fehlerfall</b>	Kein Eintrag in der Datenbank oder keine Verbindung
<b>Akteure</b>	(Internes Systemverhalten, ausgelöst durch Benutzeraktionen)
<b>Trigger</b>	Es wurde eine Bereinigung durch den Benutzer oder den Scheduler durchgeführt
<b>Standardablauf</b>	<ol style="list-style-type: none"><li>1. Eine Benutzeraktion (z.B. Start Bereinigung) wird ausgeführt.</li><li>2. Das System erzeugt einen Log-Eintrag.</li><li>3. Der Log-Eintrag wird an den MSSQL-Server weitergeleitet.</li><li>4. Der Eintrag wird dauerhaft gespeichert.</li></ol>
<b>Alternativabläufe</b>	<ul style="list-style-type: none"><li>• Verbindung zur Datenbank schlägt fehl.</li></ul>



## 1.6 Projektplan

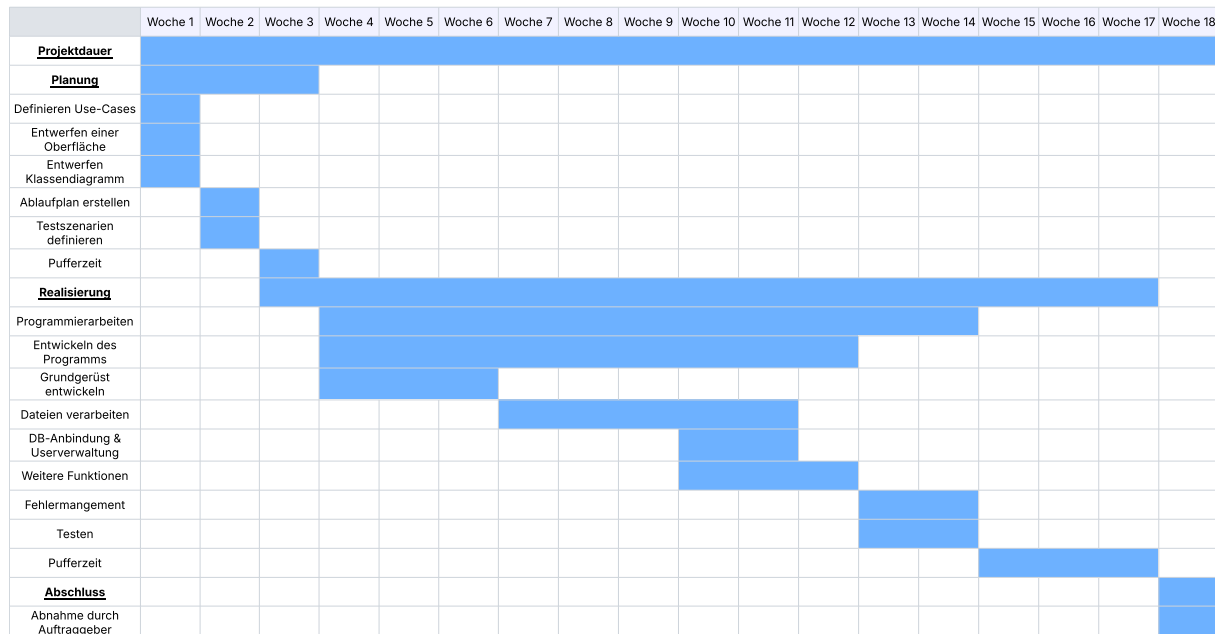


Abbildung 2: Projektplan als Gantt-Diagramm

## 1.7 Produktumgebung

Für die Entwicklung und den Einsatz des Garbage-Collection-Tools wurde folgende technische Produktumgebung verwendet:

- **Programmiersprache:** Die Anwendung wurde in **C#** entwickelt, da die Sprache eine umfangreiche Unterstützung für Windows-basierte Desktop-Anwendungen bietet und eine moderne objektorientierte Programmierung ermöglicht.
- **UI-Design:** Das Layout und die Benutzeroberfläche wurden zunächst mit **Figma** entworfen. Dieses Tool ermöglicht eine effiziente Erstellung von Prototypen und UI-Komponenten, die anschließend in die WPF-Oberfläche übertragen wurden.
- **Entwicklungsumgebung:** Zur Umsetzung kam **Visual Studio** zum Einsatz. Die integrierte Entwicklungsumgebung (IDE) bietet umfassende Werkzeuge für C#-Projekte, Debugging, Versionskontrolle und Benutzeroberflächendesign.
- **Datenbanken:** Für die persistente Speicherung von Konfigurationsdaten, Benutzerinformationen und Bereinigungsprotokollen wird ein **SQL Server** verwendet. Dieser wird in einem Docker-Container unter Linux betrieben und ist über das Netzwerk angebunden.
- **Betriebssysteme:** Die Anwendung ist kompatibel mit **Windows 10** und **Windows 11**. Die Entwicklung erfolgte unter macOS mithilfe von **Parallels**, was unter einer Windows 11 VM lief.

## 1.8 Skizze des GUI

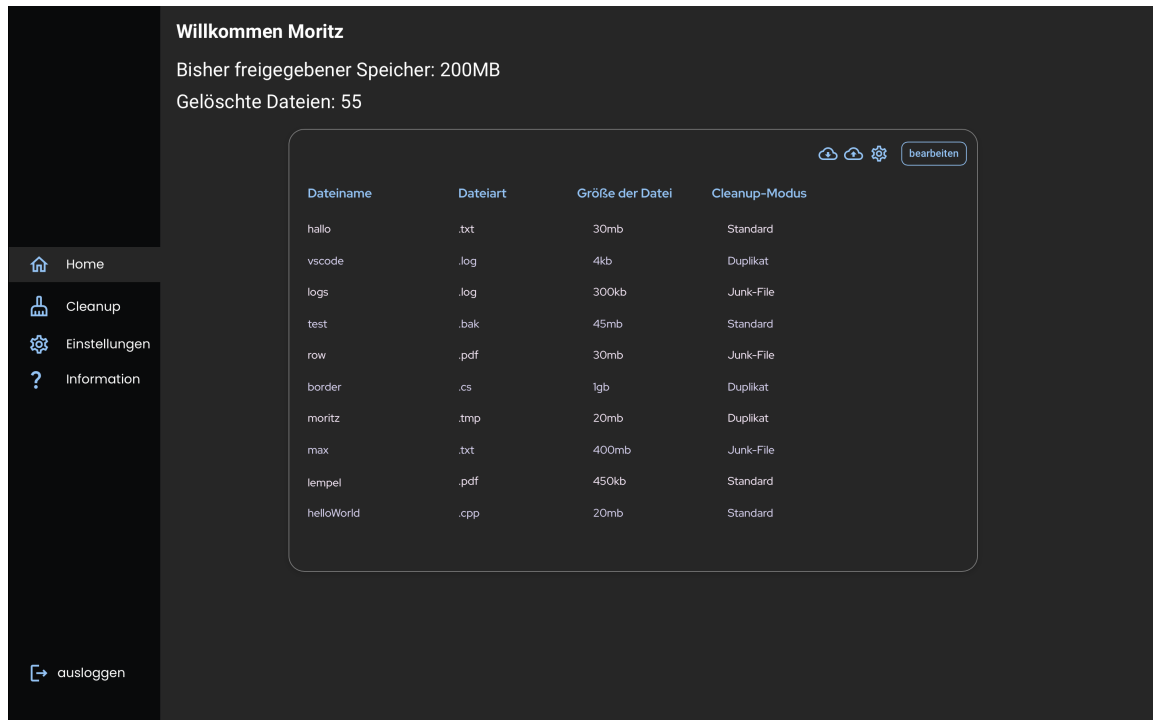


Abbildung 3: GUI-Ansicht: Home

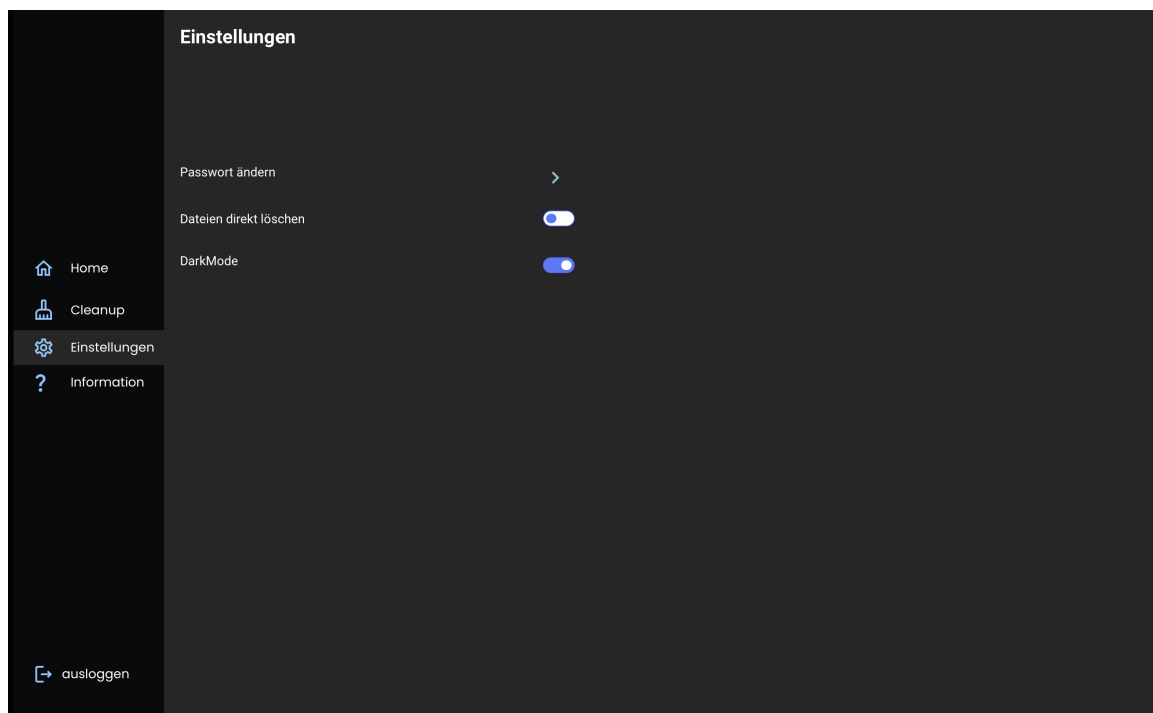


Abbildung 4: GUI-Ansicht: Einstellungen

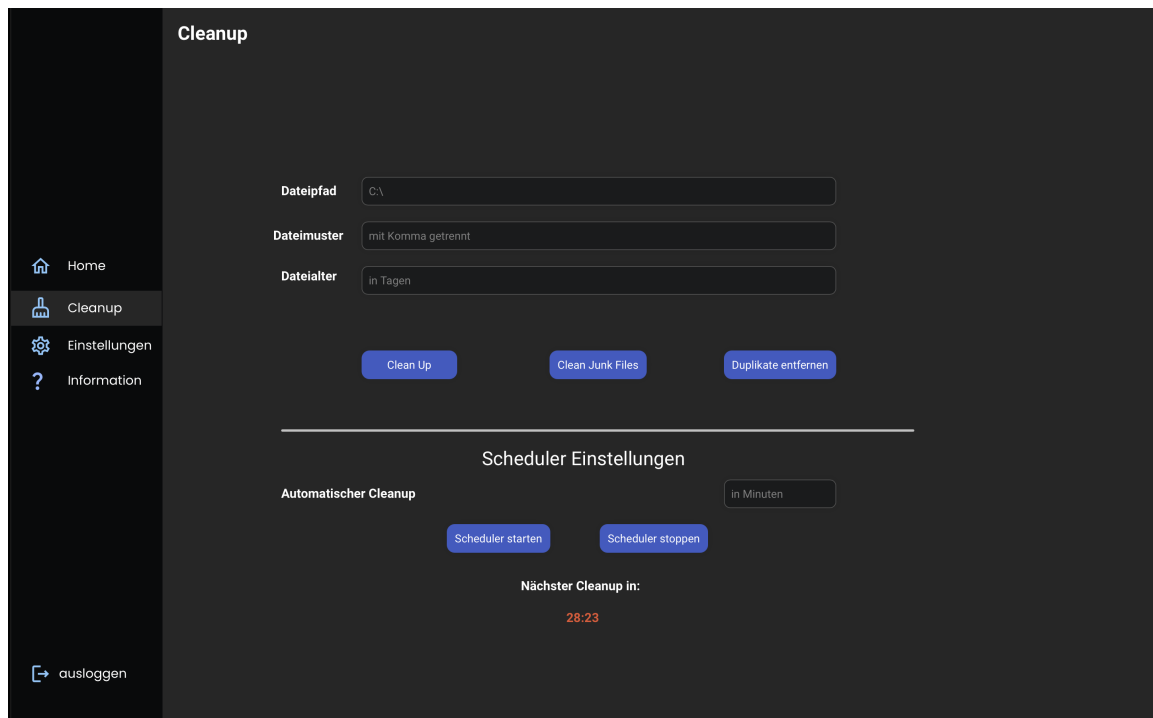


Abbildung 5: GUI-Ansicht: Cleanup

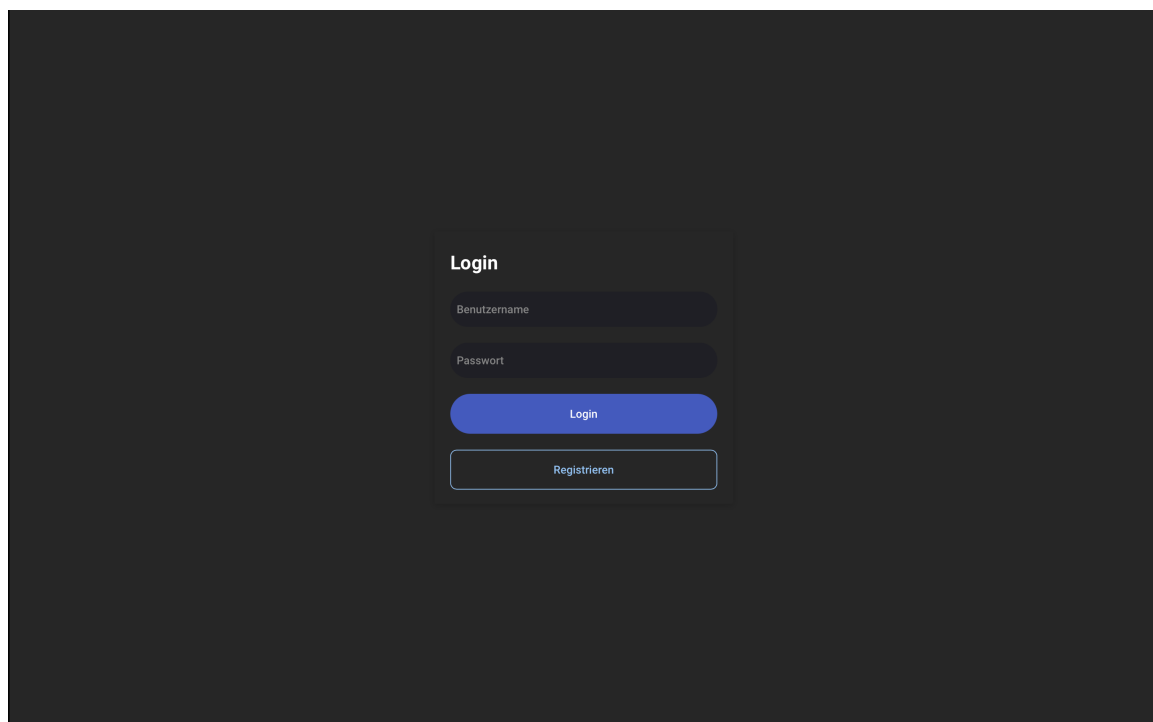


Abbildung 6: GUI-Ansicht: Login

## 1.9 DB-Entwurf

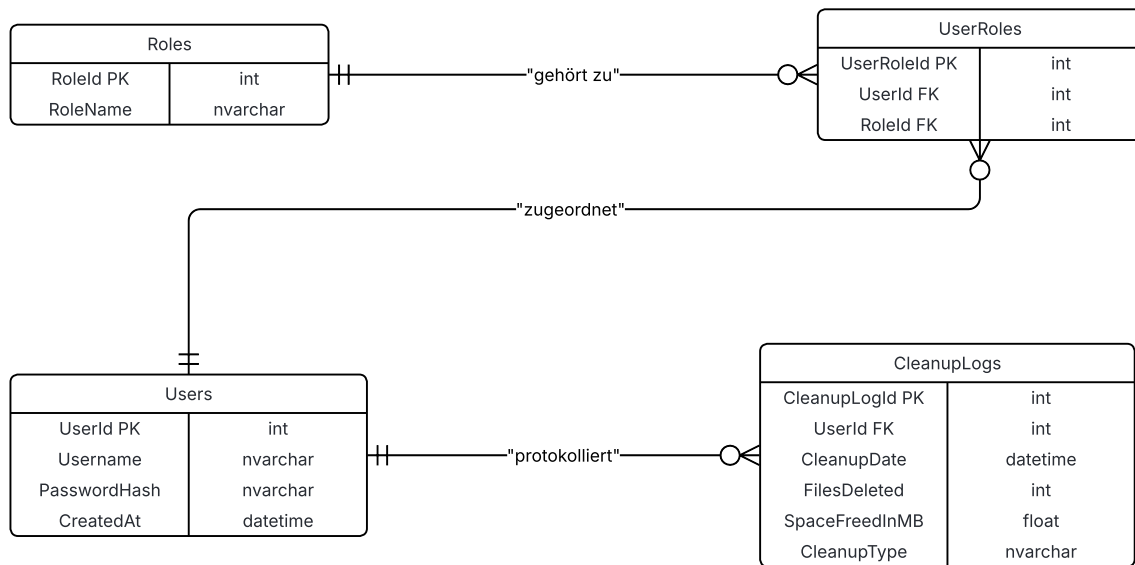


Abbildung 7: ER-Diagramm der GarbageCollector-Datenbank

### Users (Benutzer)

- **UserId (PK)**: Eindeutige Kennung für jeden Benutzer.
- **Username**: Benutzername.
- **PasswordHash**: Gehashter Wert des Benutzerpassworts.
- **CreatedAt**: Zeitstempel bei Erstellung des Benutzers.

#### Beziehungen:

- **1:n** zu **CleanupLogs**: Ein Benutzer kann mehrere Bereinigungen durchgeführt haben.
- **n:m** zu **Roles** über **UserRoles**.

### Roles (Rollen)

- **RoleId (PK)**: Eindeutige Kennung der Rolle.
- **RoleName (UNQ)**: Rollenbezeichnung, z. B. „Admin“ oder „User“.

#### Beziehung:

- **1:n** zu **UserRoles**.

### UserRoles (Benutzer-Rollen-Verknüpfung)

- **UserRoleId (PK)**: Eindeutige Kennung für jeden Eintrag.
- **UserId (FK)**: Verweis auf einen Benutzer.
- **RoleId (FK)**: Verweis auf eine Rolle.

#### Beziehungen:

- **n:1** zu **Users**, **n:1** zu **Roles**.

## CleanupLogs (Bereinigungsprotokolle)

- **CleanupLogId (PK)**: Eindeutige ID für den Protokolleintrag.
- **UserId (FK)**: Verweis auf den Benutzer.
- **CleanupDate**: Datum/Uhrzeit des Vorgangs.
- **FilesDeleted**: Anzahl gelöschter Dateien.
- **SpaceFreedInMB**: Freigegebener Speicherplatz in MB.
- **CleanupType**: Typ des Vorgangs (Standard, Junk, Duplicates).

### Beziehung:

- *n:1* zu Users.

## 1.10 Link zum Git-Repository

Link zum gehosteten GitHub-Repository:

<https://github.com/PilliPalli/Garbage-Collection-Tool-zur-Loeschung-temporaerer-Dateien>

Das Repository enthält:

- den vollständigen Quellcode,
- das SQL-Skript zum Erstellen der Datenbank,
- eine README.md

## 1.11 Testplan

### Testfall GC01 – Automatische Datenbereinigung

<b>ID:</b>	GC01
<b>Beschreibung:</b>	Überprüfung der automatischen Datenbereinigung.
<b>Vorbedingung:</b>	Das Tool ist installiert, konfiguriert und der Bereinigungsprozess ist aktiviert. Es existieren alte und neue Dateien.
<b>Test-Schritte:</b>	<ol style="list-style-type: none"><li>1. Es werden Testdateien in einem temporären Verzeichnis erstellt, einige davon sind älter als der angegebene Schwellenwert (z. B. 10 Tage).</li><li>2. Das Tool wird gestartet, und der Bereinigungsprozess ausgeführt.</li><li>3. Überprüfung, ob alte Dateien entfernt wurden und neue Dateien erhalten bleiben.</li></ol>
<b>Erwartetes Resultat:</b>	<ul style="list-style-type: none"><li>- Die alten Dateien werden erfolgreich entfernt.</li><li>- Die neuen Dateien bleiben bestehen.</li></ul>

## Testfall GC02 – Sicherheitsfunktion (Papierkorb)

<b>ID:</b>	GC02
<b>Beschreibung:</b>	Test der Sicherheitsfunktion, bei der Dateien in den Papierkorb verschoben werden.
<b>Vorbedingung:</b>	Das Tool ist installiert und konfiguriert, und die Option zum Verschieben von Dateien in den Papierkorb ist aktiviert.
<b>Test-Schritte:</b>	<ol style="list-style-type: none"><li>1. Testdateien werden in einem temporären Verzeichnis erstellt.</li><li>2. Der Bereinigungsprozess wird gestartet, und überprüft, ob die Dateien in den Papierkorb verschoben werden.</li><li>3. Der Papierkorb wird auf die verschobenen Dateien überprüft.</li></ol>
<b>Erwartetes Resultat:</b>	- Die Dateien werden nicht gelöscht, sondern in den Papierkorb verschoben.

## Testfall GC03 – Benutzerdefinierte Einstellungen

<b>ID:</b>	GC03
<b>Beschreibung:</b>	Test der benutzerdefinierten Einstellungen.
<b>Vorbedingung:</b>	Das Tool ist installiert und konfiguriert, der Bereinigungsprozess ist aktiviert.
<b>Test-Schritte:</b>	<ol style="list-style-type: none"><li>1. Zugriff auf die Einstellungsoptionen des Tools.</li><li>2. Anpassung der Parameter in der Konfigurationsdatei (z.B. Verzeichnis, Dateimuster, Alter der zu löschenden Dateien).</li><li>3. Ausführung des Bereinigungsprozesses und Überprüfung, ob die Änderungen korrekt umgesetzt wurden.</li></ol>
<b>Erwartetes Resultat:</b>	<ul style="list-style-type: none"><li>- Die neuen Einstellungen werden korrekt angewendet.</li><li>- Der Bereinigungsprozess arbeitet entsprechend den benutzerdefinierten Einstellungen.</li></ul>

## Testfall GC04 – Duplikatentfernung

<b>ID:</b>	GC04
<b>Beschreibung:</b>	Test der Duplikatentfernung.
<b>Vorbedingung:</b>	Doppelte Dateien befinden sich im Verzeichnis.
<b>Test-Schritte:</b>	<ol style="list-style-type: none"><li>1. Erstellen von doppelten Dateien (gleicher Inhalt, unterschiedlicher Dateiname).</li><li>2. Start des Bereinigungsprozesses.</li><li>3. Überprüfung, ob die Duplikate entfernt werden und nur eine Datei bestehen bleibt.</li></ol>
<b>Erwartetes Resultat:</b>	- Nur eine Kopie jeder doppelten Datei bleibt bestehen.

## Testfall GC05 – Passwortänderung

<b>ID:</b>	GC05
<b>Beschreibung:</b>	Überprüfung der Passwortänderungsfunktion.
<b>Vorbedingung:</b>	Der Benutzer ist eingeloggt, und es gibt einen Testbenutzer mit einem bekannten Passwort.
<b>Test-Schritte:</b>	<ol style="list-style-type: none"><li>1. Zugriff auf die Einstellungen, um das Passwort zu ändern.</li><li>2. Eingeben des aktuellen Passworts, gefolgt von einem neuen Passwort.</li><li>3. Überprüfen, ob die Passwortänderung korrekt durchgeführt wurde.</li></ol>
<b>Erwartetes Resultat:</b>	- Das Passwort wird erfolgreich geändert.

## 1.12 Unit Tests

Aktuell ist die Implementierung von Unit-Tests nicht geplant. Stattdessen wird der Fokus auf manuelle Tests gelegt, um sicherzustellen, dass die Schlüsselkomponenten des Tools wie erwartet funktionieren. Manuelle Tests werden für kritische Funktionen durchgeführt. Eine spätere Einführung von Unit-Tests bleibt optional, abhängig von den zukünftigen Anforderungen und der Projektentwicklung.

### Begründung:

1. **Kein Verzicht auf Qualitätssicherung:**

Statt automatisierter Tests werden manuelle Tests für zentrale Funktionen priorisiert.

2. **Flexibilität für die Zukunft:**

Die Möglichkeit zur Einführung von Unit-Tests bleibt offen, falls sich Anforderungen oder Ressourcen ändern.

3. **Zeitmanagement:**

Fokus auf funktionierende Hauptfunktionen, anstatt zusätzliche Komplexität durch Tests hinzuzufügen.

## 2 Benutzerhandbuch

### 2.1 Start und Anmeldung

Nach dem Start der Anwendung wird der Benutzer direkt zum Anmeldebildschirm weitergeleitet. Dort kann sich ein bestehender Nutzer mit seinem Benutzernamen und Passwort anmelden. Alternativ kann über den Link „Registrierung“ ein neuer Account erstellt werden.

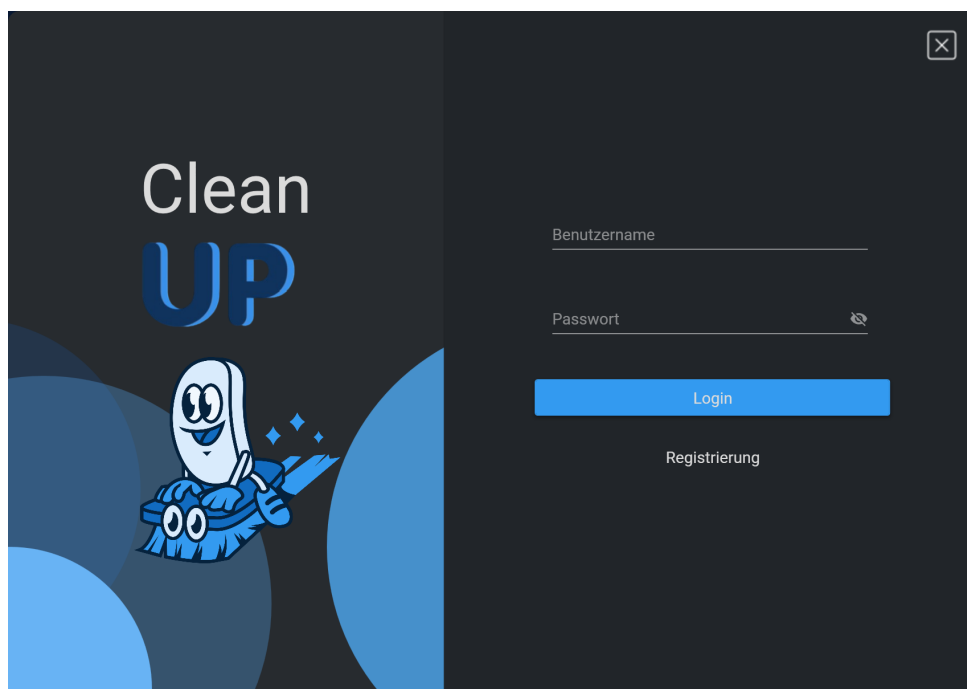


Abbildung 8: Login-Maske mit Eingabefeldern für Benutzername und Passwort



Bei der Registrierung müssen ein Benutzername sowie ein sicheres Passwort (mindestens 8 Zeichen) eingegeben und bestätigt werden. Das System überprüft die Eingaben und gibt bei fehlerhafter Eingabe eine visuelle Rückmeldung. Nach erfolgreicher Registrierung wird eine grüne Bestätigung angezeigt. Der Benutzer kann anschließend über den Link „Zurück zum Login“ zum Anmeldebildschirm zurückkehren und sich mit den zuvor gewählten Anmeldedaten einloggen.

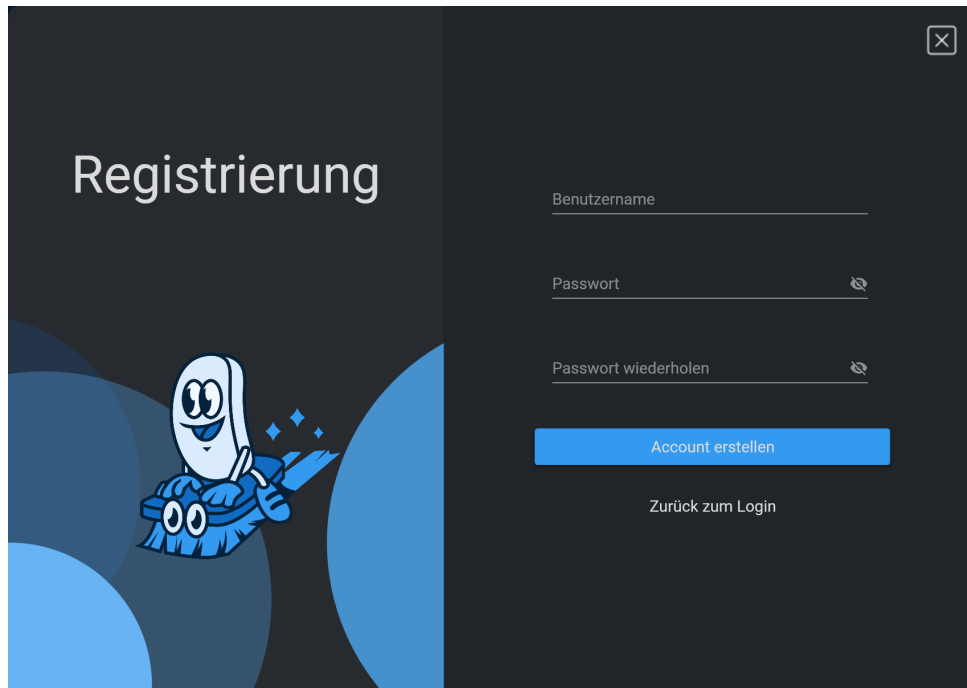
The image shows a registration form on a dark background. On the left, there is a cartoon character with a white body and blue eyes, wearing a blue ruffled collar, set against a blue abstract background. The title 'Registrierung' is written in white. On the right, the form contains three input fields: 'Benutzername', 'Passwort' (with an eye icon for toggling visibility), and 'Passwort wiederholen' (also with an eye icon). Below these fields is a blue button labeled 'Account erstellen' and a link labeled 'Zurück zum Login'. A close button (X icon) is in the top right corner.

Abbildung 9: Registrierungsformular mit Passwortbestätigung

Nach erfolgreicher Anmeldung wird der Benutzer auf den Home-Bildschirm der Anwendung weitergeleitet und dort mit einer persönlichen Willkommensnachricht begrüßt. Zusätzlich werden ihm Statistiken zur Anwendung angezeigt, darunter die Gesamtanzahl gelöschter Dateien, der freigegebene Speicherplatz sowie eine Historie der letzten fünf durchgeführten Bereinigungsvorgänge.

Bei Bedarf kann der Nutzer seine gesamte Historie einsehen, indem er seine Statistiken über den Button „Logs exportieren“ als .csv-Datei herunterlädt. Auf Wunsch können die Statistiken auch vollständig zurückgesetzt und gelöscht werden.

Über das Menü, welches als Sidebar umgesetzt wurde, kann der Nutzer nach Belieben zwischen den einzelnen Ansichten hin- und herwechseln.

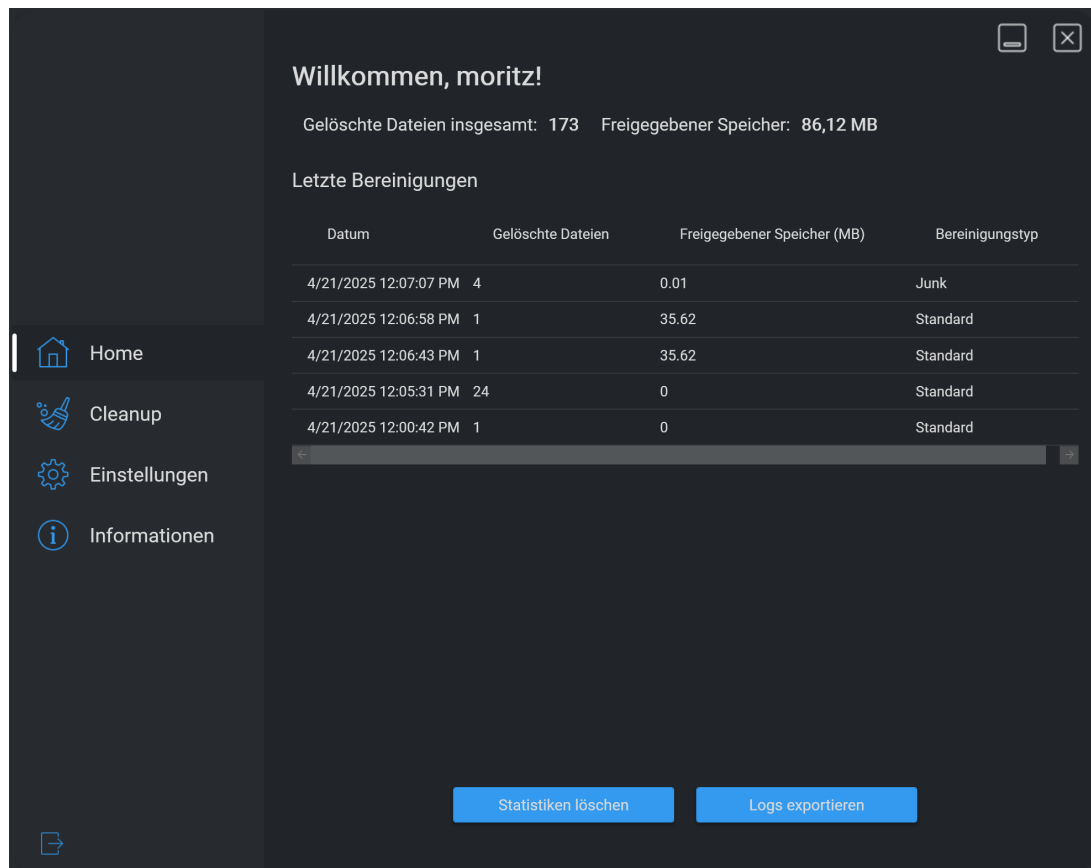


Abbildung 10: Home-Bildschirm des Nutzers mit dem gesetzten Benutzernamen „moritz“

## 2.2 Einstellungen konfigurieren

Über die Navigationsleiste auf der linken Seite kann der Benutzer die Einstellungen aufrufen. In diesem Bereich lassen sich verschiedene Verhaltensweisen der Anwendung anpassen.

- **Dateien direkt löschen:** Wenn diese Option aktiviert ist, werden erkannte Dateien unmittelbar entfernt. Ist sie deaktiviert, landen die Dateien zunächst im Papierkorb, was eine Wiederherstellung ermöglicht.
- **Rekursiv löschen:** Diese Option bestimmt, ob auch Unterordner des angegebenen Verzeichnisses durchsucht und bereinigt werden sollen.

Zusätzlich besteht die Möglichkeit, das Benutzerpasswort zu ändern. Dazu muss zunächst das aktuelle Passwort eingegeben werden, gefolgt von der Eingabe und Bestätigung eines neuen Passworts. Das System prüft, ob das neue Passwort den Sicherheitsrichtlinien entspricht und ob beide Eingaben übereinstimmen.

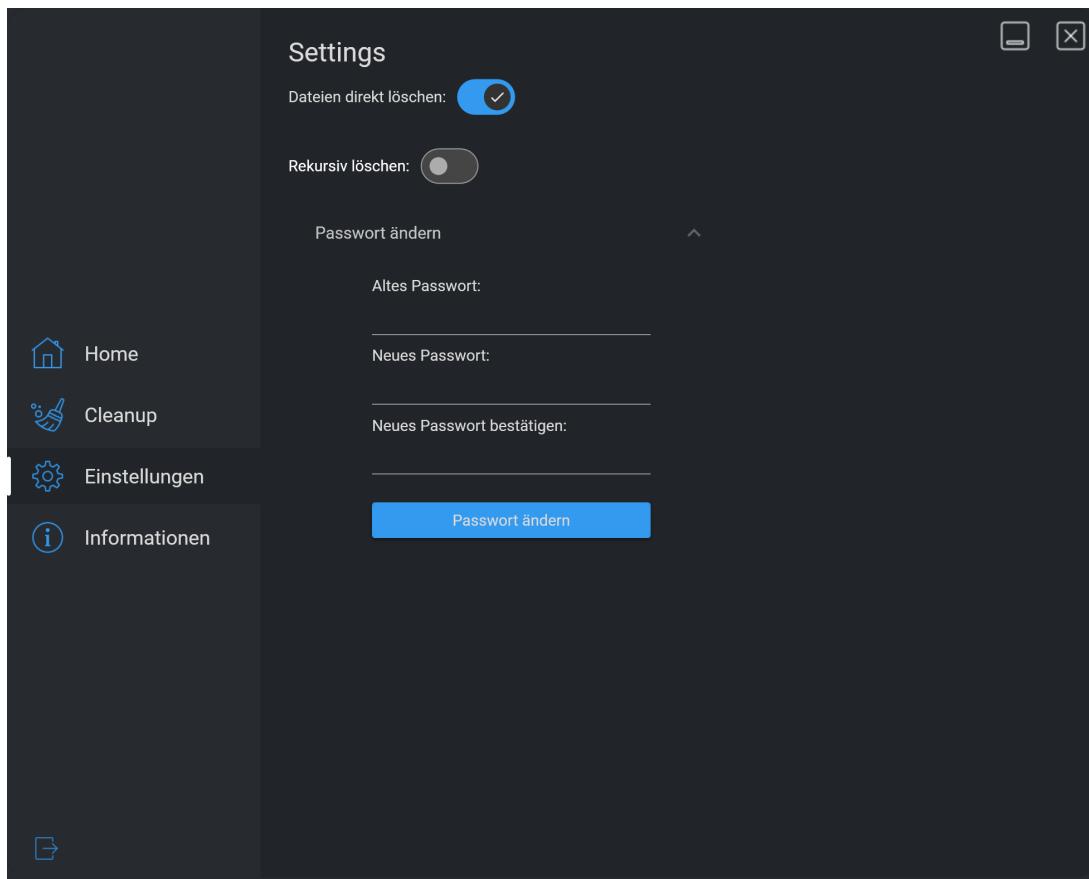


Abbildung 11: Einstellungsseite mit Optionen zur Konfiguration und Passwortänderung

## 2.3 Bereinigungsfunktionen

Über den Menüpunkt „Cleanup“ gelangt der Nutzer zur zentralen Funktionalität der Anwendung: dem Löschen temporärer und überflüssiger Dateien. In diesem Bereich kann der Benutzer gezielt konfigurieren, welche Dateien gelöscht werden sollen und in welchem Ordner die Bereinigung erfolgen soll.

- **Pfad des Ordners:** Angabe des zu bereinigenden Verzeichnisses.
- **Älter als (Tage):** Legt fest, wie alt Dateien mindestens sein müssen, um als löschwürdig zu gelten.
- **Dateimuster:** Ermöglicht die Einschränkung der Löschung auf bestimmte Dateitypen (z. B. \*.tmp, \*.log); auch einzelne Dateien können gezielt gefiltert werden.

Der Pfad des Ordners muss nicht manuell eingegeben werden, sondern kann auch über den Button links neben dem Eingabefeld ausgewählt werden. Hierbei öffnet sich ein Dialogfenster (Windows-Explorer), über das der gewünschte Zielpfad bequem ausgewählt werden kann.

Dem Benutzer stehen drei Buttons zur Verfügung:

- **Clean Up:** Startet die Standard-Bereinigung gemäß der angegebenen Kriterien.

- **Junk Files löschen:** Sucht gezielt nach typischen temporären oder nutzlosen Dateien und entfernt diese.
- **Duplikate löschen:** Erkennt und entfernt doppelt vorhandene Dateien im gewählten Verzeichnis. Wichtig ist, dass nur die Duplikate gelöscht werden – die zuerst gefundene Originaldatei bleibt erhalten.

Darunter befindet sich der Bereich **Scheduler-Einstellungen**. Hier kann der Benutzer ein automatisches Cleanup-Intervall (in Minuten) festlegen. Die Anwendung führt dann selbstständig in regelmäßigen Abständen eine Bereinigung durch.

- **Scheduler starten:** Aktiviert die automatische Bereinigung.
- **Scheduler stoppen:** Deaktiviert den geplanten Bereinigungsprozess.

Eine Statusanzeige informiert den Nutzer darüber, wann die nächste automatische Bereinigung stattfinden wird oder ob der Vorgang bereits abgeschlossen ist.

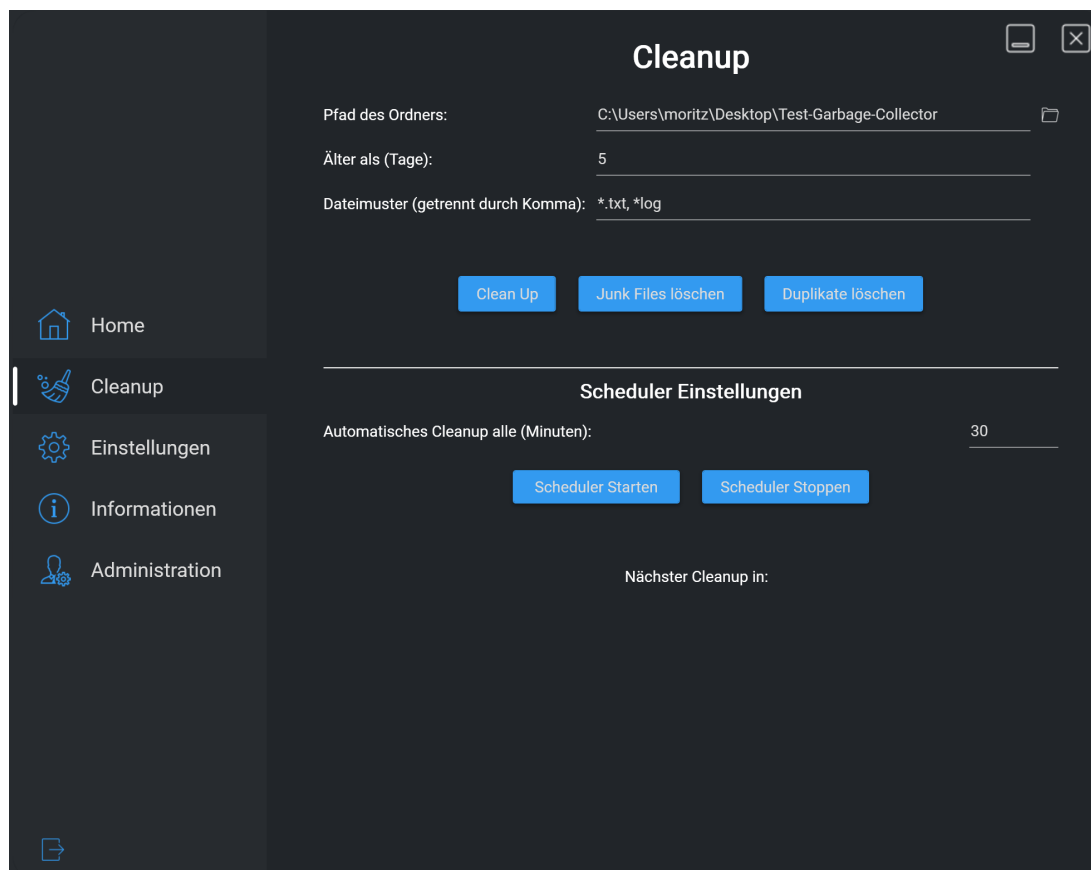


Abbildung 12: Cleanup-Seite mit Optionen zur manuellen und automatischen Bereinigung

Im folgenden Screenshot ist die Bereinigungsfunktion inklusive laufendem Scheduler in Aktion zu sehen. Der Benutzer wird in Echtzeit über den Bereinigungsprozess informiert – inklusive Fortschrittsbalken und Anzeige der aktuell gelöschten Datei samt Pfadangabe. Während der Bereinigung sind alle Buttons deaktiviert (ausgegraut), um gleichzeitige Aktionen zu verhindern. Der Scheduler zeigt präzise an, wann der nächste automatische Cleanup durchgeführt wird.

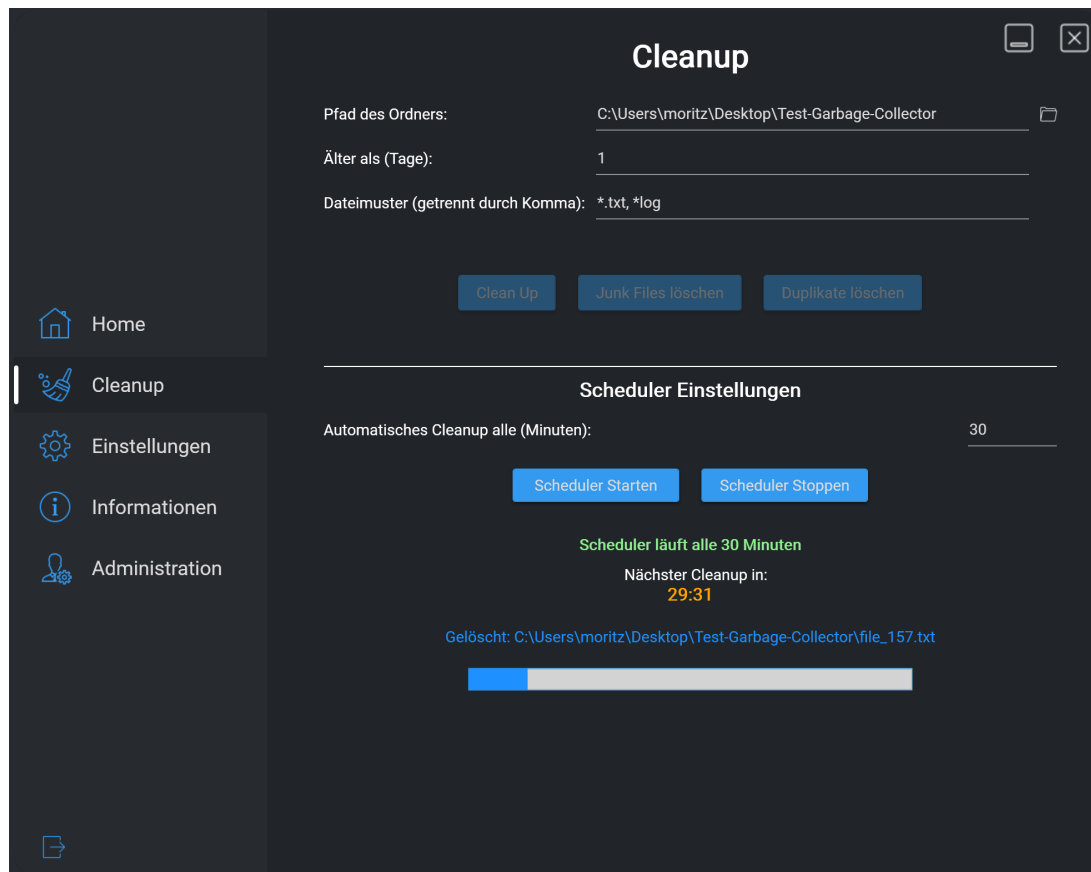


Abbildung 13: Cleanup-Seite mit laufendem Bereinigungsprozess und aktivem Scheduler

Da es sich um ein konfigurationsdateigesteuertes Programm handelt, können die meisten Einstellungen auch direkt über eine Konfigurationsdatei (`config.json`) vorgenommen werden. Diese Datei wird beim ersten Start automatisch im Programmverzeichnis mit Beispieldaten angelegt und bleibt stets synchron mit den Einstellungen der Benutzeroberfläche. Auch der Connection String zur Datenbank lässt sich dort zentral anpassen. Details zur internen Handhabung und zur Konfiguration der Datenbankverbindung sind im Kapitel **3.2 Datenbankanbindung und Docker** erläutert.

```

1 {
2   "SearchPath": "C:\\Users\\moritz\\Desktop\\Test-Garbage-Collector",
3   "FilePatterns": [
4     "*.txt"
5   ],
6   "OlderThanDays": 5,
7   "DeleteDirectly": true,
8   "DeleteRecursively": false,
9   "ConnectionString": "Data Source=192.168.178.111;Initial Catalog=GarbageCollectorDB;"
10 }

```

Abbildung 14: Beispiel einer möglichen `config.json`

## 2.4 Administration

Benutzer mit Administratorrechten haben Zugriff auf den Menüpunkt „Administration“, über den sie die Benutzerverwaltung der Anwendung steuern können.

In der Übersicht werden alle registrierten Benutzer angezeigt, inklusive:

- **Benutzername**
- **Erstellt am (Zeitstempel)**
- **Rolle** (z. B. Admin oder User)

Folgende Funktionen stehen zur Verfügung:

- **Benutzer löschen:** Entfernt den ausgewählten Benutzer dauerhaft aus dem System.
- **Passwort zurücksetzen:** Setzt das Passwort des ausgewählten Benutzers auf „defaultPassword“ zurück. Nach dem Login kann der Benutzer das Passwort über die Einstellungen ändern.
- **Rolle zuweisen:** Ermöglicht die Zuweisung einer neuen Rolle (z. B. Admin oder User) an den ausgewählten Benutzer.

Diese Funktionen sind ausschließlich für Administratoren verfügbar und ermöglichen eine zentrale und sichere Verwaltung der Benutzerkonten.

Besondere Regeln gelten für zwei spezielle Fälle:

- Der **erste registrierte Benutzer** erhält automatisch die Rolle Admin.
- Der Benutzer mit dem festen Benutzernamen **admin** ist dauerhaft Administrator und kann aus Sicherheitsgründen weder bearbeitet noch gelöscht werden.

Diese Schutzmechanismen stellen sicher, dass stets mindestens ein voll berechtigter Administrator vorhanden ist, der die vollständige Kontrolle über das System behält.

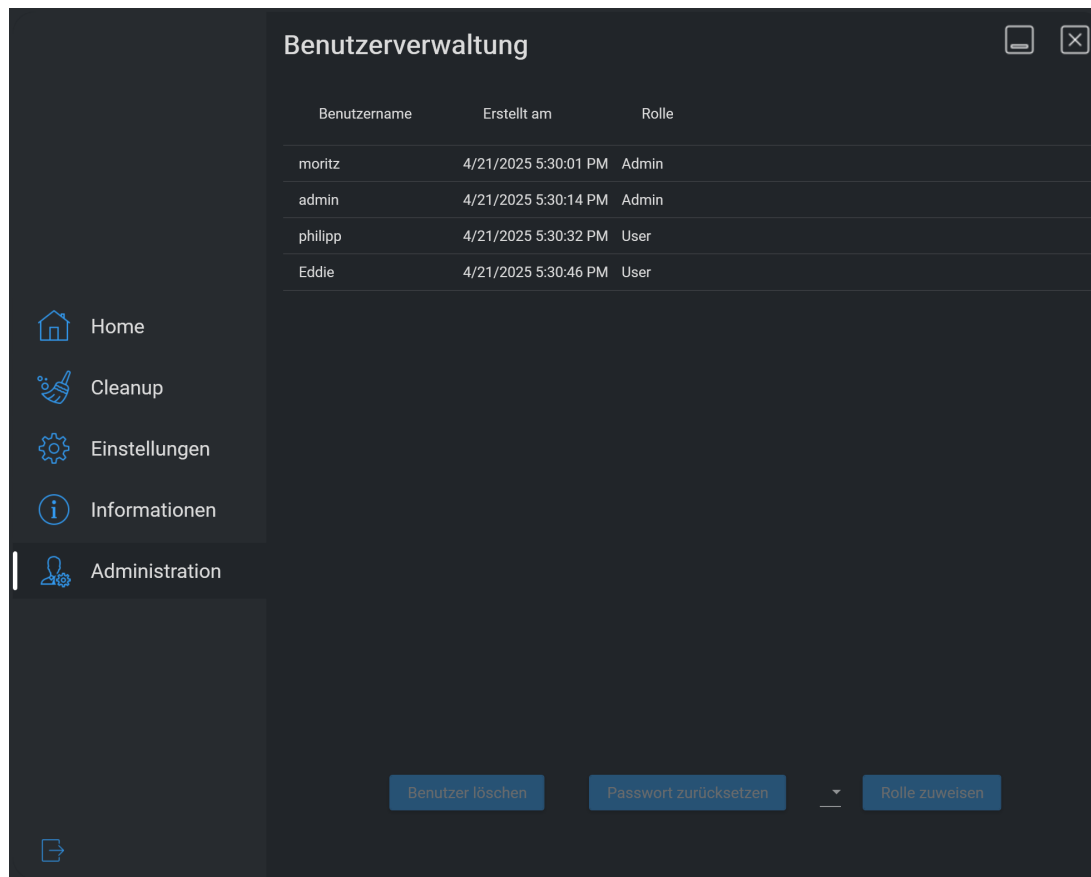


Abbildung 15: Administrationsbereich zur Benutzerverwaltung mit Rollenzuweisung

## 2.5 Abmeldung und Beenden

Im rechten oberen Bereich der Anwendung befinden sich zwei Buttons zur Fenstersteuerung:

- **Minimieren:** Reduziert das Fenster in die Taskleiste.
- **Schließen:** Beendet die Anwendung vollständig.

Am unteren linken Rand der Navigation befindet sich der Button zum Ausloggen. Dieser ermöglicht es dem Benutzer, die aktuelle Sitzung zu beenden und sich anschließend mit einem anderen Benutzerkonto neu anzumelden.

### Hinweis zur Informationsseite

Über die Menüoption „Informationen“ kann eine kompakte Bedienhilfe innerhalb der Anwendung aufgerufen werden. Diese enthält in Kurzform die wichtigsten Funktionen und Anwendungsbereiche der Software. Die Informationsseite dient als unterstützende Hilfe für neue Nutzer oder zur schnellen Auffrischung der Funktionsweise.

## 3 Implementierung

### 3.1 Architekturüberblick

Das Programm basiert auf dem **Model-View-ViewModel (MVVM)**-Pattern, welches eine klare Trennung zwischen Benutzeroberfläche (View), Präsentationslogik (ViewModel) und Datenmodell (Model) gewährleistet. Diese Architektur ist in WPF-Anwendungen weit verbreitet und fördert die Wartbarkeit, Testbarkeit sowie Erweiterbarkeit der Software.

#### Aufbau der Anwendung

Die Anwendung gliedert sich in drei zentrale Komponenten:

- **Model:**  
Die Model-Schicht beinhaltet die zentralen Datenklassen wie **User**, **Role**, **UserRole**, **CleanupLog** sowie **AppConfig**. Sie spiegeln die Datenbankstruktur wider und sind mittels Entity Framework Core mit der zugrunde liegenden Datenbank verbunden.
- **ViewModel:**  
Die ViewModels enthalten die Anwendungslogik sowie Bindings zu den Views. Dazu gehören Properties, Commands und Methoden. Zentrale Klassen sind u. a.:
  - **NavigationVM** – steuert den Seitenwechsel
  - **LoginVM**, **RegisterVM** – enthalten die Authentifizierungslogik
  - **HomeVM**, **CleanupVM**, **SettingsVM**, **InformationVM**, **AdministrationVM** – zuständig für die jeweilige Seitenlogik
- **View:**  
Die Views werden in XAML definiert und über **DataTemplates** im **App.xaml**-Ressourcendictionary an ihre zugehörigen ViewModels gebunden:

```
1 <DataTemplate DataType="{x:Type vm:HomeVM}">
2   <view:Home />
3 </DataTemplate>
```

Abbildung 16: DataTemplate für die View-Bindung von HomeVM

Durch diese Bindung wird beim Wechsel der Eigenschaft **CurrentView** automatisch die zugehörige View angezeigt.

#### Navigation

Die Navigation wird zentral im **NavigationVM** gesteuert. Dieses besitzt die Property **CurrentView**, die während der Laufzeit dynamisch auf das gewünschte ViewModel gesetzt wird. Die Navigation erfolgt über **RelayCommands**, die den Seitenwechsel auslösen:



```

1 private void Home(object obj) => CurrentView = new HomeVM();
2 private void Cleanup(object obj) => CurrentView = new CleanupVM();
3 private void Settings(object obj) => CurrentView = new SettingsVM();
4 private void Information(object obj) => CurrentView = new InformationVM();
5 private void Administration(object obj) => CurrentView = new AdministrationVM();
6
7 public NavigationVM()
8 {
9     HomeCommand = new RelayCommand(Home);
10    CleanupCommand = new RelayCommand(Cleanup);
11    SettingsCommand = new RelayCommand(Settings);
12    InformationCommand = new RelayCommand(Information);
13    AdministrationCommand = new RelayCommand(Administration);
14    LogoutCommand = new RelayCommand(Logout);
15
16    CurrentView = new HomeVM();
17 }

```

Abbildung 17: Command-Implementierung für Navigation

Im `MainWindow.xaml` wird ein `ContentControl` verwendet, das stets den aktuellen Inhalt anhand der Property `CurrentView` darstellt:

```

1 <Grid Grid.Column="1" HorizontalAlignment="Stretch" VerticalAlignment="Stretch">
2     <ContentControl x:Name="Pages" Content="{Binding CurrentView}" />
3 </Grid>

```

Abbildung 18: ContentControl für dynamische View-Anzeige

Ein `AdminVisibilityConverter` regelt die Sichtbarkeit bestimmter Bereiche – etwa des Administrationsmenüs – abhängig von der Benutzerrolle.

### Command-Handling mit RelayCommand

Zur Umsetzung der Benutzerinteraktionen – wie z. B. Button-Klicks zur Navigation – nutzt die Anwendung das `RelayCommand`-Pattern. Dabei handelt es sich um eine flexible Implementierung des `ICommand`-Interfaces. Sie erlaubt es, Logik direkt im ViewModel zu definieren, ohne Code-Behind-Dateien zu verwenden.

`RelayCommand` kapselt dabei eine Aktion (`Action<object>`) und eine optionale Bedingung zur Ausführbarkeit (`Func<object, bool>`).

```

1  class RelayCommand : ICommand
2  {
3      private readonly Action<object> _execute;
4      private readonly Func<object, bool> _canExecute;
5
6      public event EventHandler CanExecuteChanged
7      {
8          add { CommandManager.RequerySuggested += value; }
9          remove { CommandManager.RequerySuggested -= value; }
10     }
11     public RelayCommand(Action<object> execute, Func<object, bool> canExecute = null)
12     {
13         _execute = execute;
14         _canExecute = canExecute;
15     }
16     public bool CanExecute(object parameter) => _canExecute == null ||
17     ↪ _canExecute(parameter);
18     public void Execute(object parameter) => _execute(parameter);
19 }

```

Abbildung 19: Grundstruktur eines RelayCommands

Diese Trennung fördert die Wiederverwendbarkeit und testbare Logik, was das MVVM-Muster ideal ergänzt.

### Property-Änderungen mit ViewModelBase

Alle ViewModels erben von ViewModelBase, welche die Schnittstelle INotifyPropertyChanged implementiert. Diese sorgt dafür, dass Änderungen an Eigenschaften automatisch an die View gemeldet werden. Ein zentrales Prinzip von Data Binding in WPF.

```

1  public class ViewModelBase : INotifyPropertyChanged
2  {
3      public event PropertyChangedEventHandler PropertyChanged;
4      public void OnPropertyChanged([CallerMemberName] string propName = null)
5      {
6          PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propName));
7      }
8  }

```

Abbildung 20: PropertyChanged-Implementierung

## Architekturübersicht

Die folgende Grafik zeigt das vereinfachte Klassendiagramm der Anwendung und veranschaulicht das Zusammenspiel zwischen View, ViewModel und Model gemäß dem MVVM-Muster:

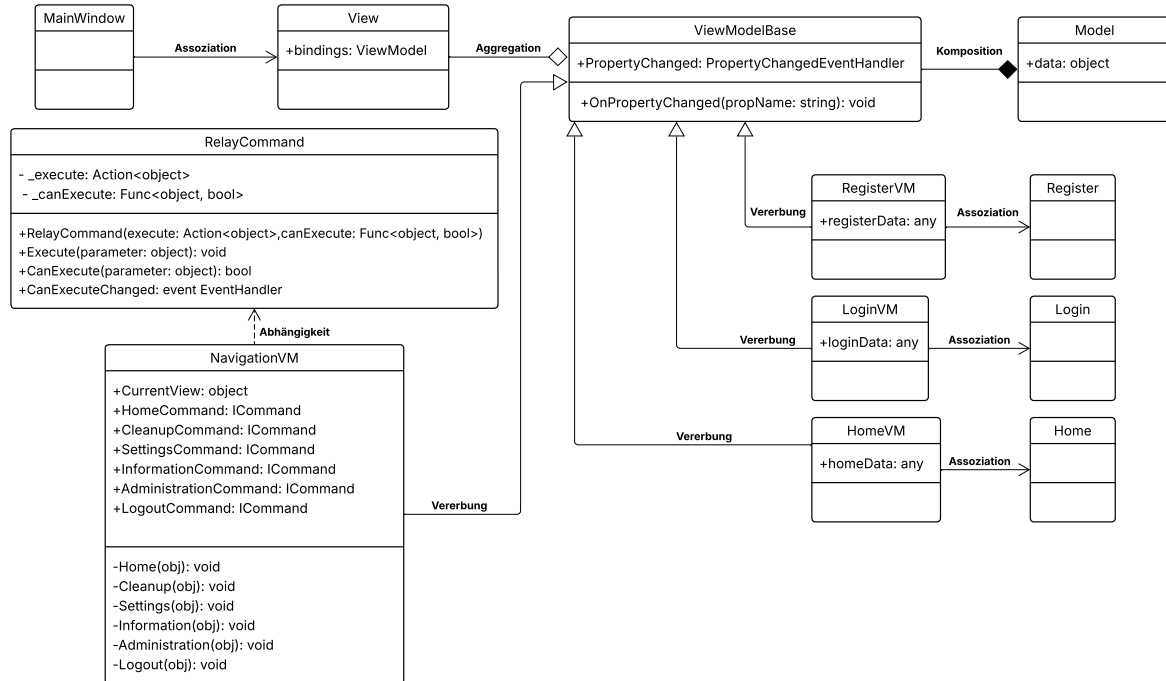


Abbildung 21: Architekturübersicht des Garbage-Collection-Tools (MVVM)

## 3.2 Datenbankanbindung und Docker

Für die Speicherung von Benutzerdaten, Rollen, Protokollen sowie Konfigurationen wird eine relationale Datenbank auf Basis von **Microsoft SQL Server** eingesetzt. Die Anbindung erfolgt über **Entity Framework Core**, wodurch eine objektorientierte Kommunikation mit der Datenbank möglich ist. Die Datenbankmodelle – beispielsweise **User**, **CleanupLog** oder **Role** – sind als C#-Klassen im Projekt umgesetzt und werden über einen zentralen **DbContext** verwaltet.

### Entwicklungsumgebung mit Docker

Während der Entwicklung kommt **Docker** zum Einsatz, um die Datenbank in einem isolierten Container bereitzustellen. Dies bietet mehrere Vorteile:

- **Plattformunabhängigkeit:** Die Umgebung lässt sich auf jedem System mit Docker identisch starten.
- **Reproduzierbarkeit:** Fehler durch unterschiedliche Systemkonfigurationen werden minimiert.
- **Schnelles Setup:** Die gesamte Datenbank ist mit einem einzigen Befehl einsatzbereit.

- **Isolierte Testumgebung:** Änderungen an der Datenbankstruktur können risikolos getestet werden.

```
sudo docker run -e "ACCEPT_EULA=Y" -e "MSSQL_SA_PASSWORD=VeryStr0ngP@ssw0rd" --name sql
↪ -p 1433:1433 -v sql_server:/var/opt/mssql -d --restart=always --hostname sql
↪ --platform linux/amd64 -d mcr.microsoft.com/mssql/server:2022-latest
```

Abbildung 22: Beispiel für einen `docker run`-Befehl zur Bereitstellung eines SQL-Servers

## Praxisnähe durch serverseitige Architektur

Auch im Unternehmenskontext liegt die Datenbank typischerweise nicht auf dem Client, sondern auf einem zentralen Server innerhalb des Netzwerks. Durch diese Trennung von Anwendung und Datenhaltung:

- wird die Datensicherheit erhöht,
- ist eine zentrale Sicherung und Verwaltung möglich,
- können mehrere Benutzer gleichzeitig auf die Daten zugreifen.

Der Einsatz von Docker bildet diese Struktur realitätsnah ab und erleichtert die spätere Überführung in eine produktive Infrastruktur.

## Connection-String-Konfiguration

Der Connection String zur Datenbank ist in einer `config.json`-Datei gespeichert. In realen Unternehmensszenarien würde dieser sensible Wert jedoch nicht für den Endnutzer sichtbar abgelegt, sondern zentral verwaltet.

Für Demonstrationszwecke und zur Erleichterung der Konfiguration wurde in diesem Projekt bewusst auf die Lösung mittels `config.json` zurückgegriffen.

## 4 Tests

Die im Kapitel **1.11 Testplan** dokumentierten Testfälle wurden manuell durchgeführt. In der folgenden Tabelle ist für jeden Testfall ein Protokoll mit den wichtigsten Informationen zur Durchführung festgehalten.

Jeder Test wurde auf einem Windows-System mit Docker-gestützter Datenbankumgebung ausgeführt.

### Testprotokoll GC01 – Automatische Datenbereinigung

<b>Test-ID:</b>	GC01
<b>Datum:</b>	19.04.2025
<b>Tester:</b>	Moritz Nicola Kreis
<b>Beobachtetes Resultat:</b>	Alte Testdateien wurden korrekt entfernt. Neue Dateien blieben erhalten. Keine unerwarteten Löschungen.
<b>Testergebnis:</b>	<b>Bestanden.</b> Ergebnis entspricht dem erwarteten Verhalten.

## Testprotokoll GC02 – Sicherheitsfunktion (Papierkorb)

Test-ID:	GC02
Datum:	19.04.2025
Tester:	Moritz Nicola Kreis
Beobachtetes Resultat:	Dateien wurden wie konfiguriert in den Windows-Papierkorb verschoben.
Testergebnis:	<b>Bestanden.</b> Das Verhalten entspricht den Erwartungen.

## Testprotokoll GC03 – Benutzerdefinierte Einstellungen

Test-ID:	GC03
Datum:	20.04.2025
Tester:	Moritz Nicola Kreis
Beobachtetes Resultat:	Einstellungen aus der <code>config.json</code> wurden korrekt übernommen und angewendet.
Testergebnis:	<b>Bestanden.</b> Der Bereinigungsprozess funktionierte wie konfiguriert.

## Testprotokoll GC04 – Duplikatentfernung

Test-ID:	GC04
Datum:	20.04.2025
Tester:	Moritz Nicola Kreis
Beobachtetes Resultat:	Alle identischen Dateien (Inhalt) bis auf eine pro Gruppe wurden erfolgreich entfernt.
Testergebnis:	<b>Bestanden.</b> Keine unerwünschten Löschungen.

## Testprotokoll GC05 – Passwortänderung

Test-ID:	GC05
Datum:	21.04.2025
Tester:	Moritz Nicola Kreis
Beobachtetes Resultat:	Passwort wurde korrekt geändert. Login mit neuem Passwort funktionierte.
Testergebnis:	<b>Bestanden.</b> Keine Abweichungen festgestellt.

# 5 Installationsanleitung

## 5.1 Systemvoraussetzungen

- Windows 10 oder neuer
- .NET Runtime 8.0 oder höher
- Microsoft SQL Server (lokal oder im Netzwerk)

- Microsoft SQL Server Management Studio (SSMS) zur Verwaltung der Datenbank
- optional: Git, falls der Quellcode über ein Repository geladen wird

## 5.2 Vorbereitung der Datenbank

- Eine neue Datenbank mit dem Namen „GarbageCollectorDB“ anlegen.
- Das mitgelieferte SQL-Skript `GarbageCollectorDB.sql` ausführen. Dieses erstellt alle notwendigen Tabellen und Beziehungen.
- Es werden keine Testdaten benötigt, da sensible Informationen (z. B. Passwörter) in der Anwendung verschlüsselt gespeichert werden.

## 5.3 Konfiguration und Verbindung

- Der Connection String wird in der Datei `config.json` gespeichert.
- Diese Datei befindet sich im gleichen Verzeichnis wie die ausführbare Datei der Anwendung.
- Das Programm kann nun ausgeführt werden, entweder über VisualStudio oder über eine Release-Build Version

# 6 Quellcode

## 6.1 Die Klasse `CleanupVM.cs` im Garbage-Collector

### 6.1.1 Initialisierung

Die Klasse `CleanupVM` ist das zentrale `ViewModel` und das Herzstück des Programms. Sie fungiert als Vermittler zwischen der grafischen Benutzeroberfläche (View) und der Anwendungslogik (Model) und kapselt die gesamte Funktionalität rund um den Datei-Bereinigungsprozess.

**Eigenschaften** Die Klasse stellt zahlreiche an die Oberfläche gebundene Eigenschaften bereit, unter anderem:

- `DirectoryPath`: Der Pfad des Verzeichnisses, das bereinigt werden soll. Änderungen triggern `OnPropertyChanged`, um die UI zu aktualisieren.
- `OlderThanDays`: Schwellenwert in Tagen für zu löschende Dateien.
- `FilePatterns`: Liste mit Dateimustern (z. B. `*.tmp`, `*.log`) zur Filterung.
- `IntervalInMinutes`: Intervall in Minuten für den automatisierten Bereinigungsscheduler.
- `SchedulerStatus`, `TimeUntilNextCleanup`: Informationen zum aktuellen Scheduler-Status.
- `StatusMessage`, `StatusCode`: Visualisierung von Status- und Fehlermeldungen.

- `ProgressBarVisibility`, `ProgressValue`, `ProgressMaximum`: Steuerung der Fortschrittsanzeige.
- `AreButtonsEnabled`: De-/Aktivierung von Schaltflächen bei laufenden Prozessen.

**Befehle** Die folgenden `ICommand`-Befehle werden zur Interaktion mit der Oberfläche bereitgestellt:

- `CleanupCommand`: Führt die Standard-Bereinigung aus.
- `CleanJunkFilesCommand`, `RemoveDuplicateFilesCommand`: Löschen von temporären Dateien bzw. Duplikaten.
- `StartSchedulerCommand`, `StopSchedulerCommand`: Steuerung des Schedulers.
- `SearchDirectoryPathCommand`: Öffnet einen Dialog zur Verzeichnisauswahl.

**Konstruktor** Im Konstruktor erfolgt die Initialisierung der zentralen Komponenten:

- Laden der Konfiguration aus der `config.json`-Datei.
- Initialisierung der Befehle (`RelayCommand`).
- Einrichtung eines Countdowns für den nächsten automatischen Cleanup.
- Laden bisheriger Protokolle.
- Setzen von Startwerten für Statusanzeigen und Schaltflächen.

```

1 public CleanupVM()
2 {
3     _config = AppConfig.LoadFromJson("config.json");
4     CleanupCommand = new RelayCommand(async obj => await
5         ↳ ExecuteWithButtonDisable(CleanupAsync));
6     CleanJunkFilesCommand = new RelayCommand(async obj => await
7         ↳ ExecuteWithButtonDisable(CleanJunkFilesAsync));
8     RemoveDuplicateFilesCommand = new RelayCommand(async obj => await
9         ↳ ExecuteWithButtonDisable(RemoveDuplicateFilesAsync));
10    StartSchedulerCommand = new RelayCommand(obj => StartScheduler());
11    StopSchedulerCommand = new RelayCommand(obj => StopScheduler());
12    SearchDirectoryPathCommand = new RelayCommand(obj => SearchDirectoryPath());
13    ProgressBarVisibility = Visibility.Collapsed;
14
15    _countdownTimer = new System.Timers.Timer(1000);
16    _countdownTimer.Elapsed += CountdownElapsed;
17 }

```

Abbildung 23: Screenshot des Konstruktors von `CleanupVM`

**Hilfsmethoden** Neben dem Konstruktor enthält die Klasse unterstützende Methoden wie zum Beispiel:

- **OnPropertyChanged:** Aktualisiert die Oberfläche bei Änderungen.
- **RestartTimer:** Startet den Timer neu, falls sich das Intervall ändert.
- **SearchDirectoryPath:** Öffnet einen Systemdialog zur Verzeichnisauswahl.
- **ExecuteWithButtonDisable:** Führt eine Methode aus und deaktiviert temporär die Schaltflächen.

Diese Initialisierungslogik sorgt dafür, dass die Bereinigungsfunktionen unmittelbar nach dem Laden der View voll einsatzbereit sind.

### 6.1.2 Manuelle Bereinigung

Die manuelle Bereinigung ist eine zentrale Funktion des Garbage-Collection-Tools. Sie wird durch den Befehl `CleanupCommand` ausgelöst, welcher beim Klick auf den „CleanUp“-Button in der Oberfläche aktiviert wird. Der zugehörige Delegat ruft die Methode `CleanupAsync()` auf, eingebettet in den Wrapper `ExecuteWithButtonDisable()`, um Mehrfachaktionen zu verhindern.

**Ablauf** Die Methode `CleanupAsync()` prüft zunächst, ob das angegebene Verzeichnis existiert. Anschließend werden alle Dateien geladen, die älter als ein definierter Schwellenwert sind und dem Dateimuster entsprechen. Die Methode berechnet den freigegebenen Speicherplatz und erstellt anschließend einen Protokolleintrag. Die Dateien werden dann in `DeleteFilesAsync()` gelöscht.

Während des Vorgangs wird eine Fortschrittsanzeige aktualisiert und Statusmeldungen informieren den Benutzer über den Bearbeitungsstand.



```

1 public async Task CleanupAsync()
2 {
3     if (!Directory.Exists(DirectoryPath))
4     {
5         StatusMessage = "Der angegebene Pfad existiert nicht.";
6         return;
7     }
8
9     var deletionThreshold = DateTime.Now.AddDays(-OlderThanDays);
10
11     var filesToDelete = await Task.Run(() => GetFilesToProcess()
12         .Where(file => File.GetLastWriteTime(file) < deletionThreshold)
13         .ToList());
14
15     if (!filesToDelete.Any())
16     {
17         StatusMessage = "Keine Dateien zum Löschen gefunden.";
18         return;
19     }
20
21     long totalBytes = filesToDelete.Sum(f => new FileInfo(f).Length);
22     double spaceFreedInMb = totalBytes / (1024.0 * 1024.0);
23     await LogCleanupAsync(filesToDelete.Count, spaceFreedInMb, "Standard");
24     await DeleteFilesAsync(filesToDelete, "Löschvorgang");
25 }

```

Abbildung 24: Screenshot der Methode CleanupAsync

## Wichtige Details

- CleanupCommand ist mit der Methode CleanupAsync() verknüpft, die asynchron arbeitet.
- Über GetFilesToProcess() werden nur Dateien selektiert, die das Alter und Muster erfüllen.
- Die Methode DeleteFilesAsync() übernimmt das Löschen unter Berücksichtigung von Sperrungen.
- Ein Protokoll mit Dateizahl und freigegebenem Speicher wird über LogCleanupAsync() erfasst.

Die Methode CleanupAsync() stellt somit die zentrale Logik zur klassischen, manuellen Bereinigung dar.

### 6.1.3 Junk-Dateien erkennen und bereinigen

Zusätzlich zur Standard-Bereinigung bietet das Garbage-Collection-Tool die gezielte Entfernung sogenannter „Junk“-Dateien an. Dabei handelt es sich um temporäre Dateien mit

bestimmten typischen Endungen wie `.tmp`, `.log`, `.bak`, `.old`, `.dmp` oder `.swp`. Solche Dateien sammeln sich häufig in systemweiten oder anwendungsspezifischen Verzeichnissen an und können Speicherressourcen unnötig blockieren.

Die Bereinigungsfunktion wird durch den Befehl `CleanJunkFilesCommand` angestoßen, welcher in der Benutzeroberfläche über einen eigenen Button erreichbar ist. Intern ruft dieser Befehl die Methode `CleanJunkFilesAsync()` auf. Diese Methode lokalisiert Junk-Dateien im temporären Systemverzeichnis, berechnet die freigegebenen Ressourcen und löscht die Dateien, sofern sie nicht gesperrt sind.

Im folgenden Screenshot ist die zentrale Logik zur Erkennung und Löschung dieser Dateien dargestellt:

```
1 private async Task CleanJunkFilesAsync()
2 {
3     string tempPath = Path.GetTempPath();
4     var junkFiles = Directory.GetFiles(tempPath, "*.*",
5     ↪ System.IO.SearchOption.AllDirectories)
6         .Where(f => IsJunkFile(f))
7         .ToList();
8
9     if (!junkFiles.Any())
10    {
11        StatusMessage = "Keine Junk-Dateien gefunden.";
12        return;
13    }
14
15    long totalBytes = junkFiles.Sum(f => new FileInfo(f).Length);
16    double spaceFreedInMb = totalBytes / (1024.0 * 1024.0);
17    await LogCleanupAsync(junkFiles.Count, spaceFreedInMb, "Junk");
18    await DeleteFilesAsync(junkFiles, "Löschen der Junk-Dateien");
19 }
```

Abbildung 25: Screenshot der Methode `CleanJunkFilesAsync`

Die Erkennung basiert auf einer Hilfsmethode namens `IsJunkFile()`, die eine Liste typischer Junk-Endungen prüft und bei Übereinstimmung einen `true`-Wert zurückliefert. Diese Funktion stellt sicher, dass nur wirklich überflüssige Dateien bereinigt werden. Die folgende Abbildung zeigt die Umsetzung dieser Methode:

```
1 private bool IsJunkFile(string filePath)
2 {
3     string[] junkExtensions = { ".tmp", ".log", ".bak", ".old", ".dmp", ".swp" };
4     return junkExtensions.Contains(Path.GetExtension(filePath).ToLower());
5 }
```

Abbildung 26: Screenshot der Methode `IsJunkFile`

Die gezielte Junk-Bereinigung stellt eine wichtige Ergänzung zur klassischen Zeit-basierten Löschung dar. Sie ermöglicht eine zusätzliche Entlastung des Systems und sorgt für eine effizientere Verwaltung temporärer Ressourcen.

#### **6.1.4 Duplikate erkennen und entfernen**

Eine weitere Funktion des Garbage-Collection-Tools ist die Erkennung und Beseitigung doppelter Dateien. In vielen Arbeitsverzeichnissen entstehen durch Kopiervorgänge, fehlerhafte Exporte oder wiederholte Downloads schnell Dateiduplikate, die unnötig Speicherplatz belegen.

Die Bereinigung erfolgt über den Befehl `RemoveDuplicateFilesCommand`, der mit der Methode `RemoveDuplicateFilesAsync()` verknüpft ist. Nach Ausführung durchsucht die Methode die vom Benutzer gewählten Verzeichnisse und vergleicht Dateien anhand ihrer Inhalte.

Hierzu wird für jede Datei ein Hashwert mittels MD5-Algorithmus berechnet. Dateien mit identischem Hash werden als Duplikate betrachtet. Von jeder identifizierten Duplikatengruppe bleibt nur eine Datei bestehen, alle weiteren werden gelöscht. Die freigegebenen Ressourcen werden dokumentiert.

```

1 private async Task RemoveDuplicateFilesAsync()
2 {
3     if (!Directory.Exists(DirectoryPath))
4     {
5         StatusMessage = "Der angegebene Pfad existiert nicht.";
6         return;
7     }
8
9     var fileHashes = new Dictionary<string, List<string>>>();
10    var filesToDelete = new List<string>();
11    double totalFreedSpace = 0;
12
13    var patterns = FilePatterns.Split(new[] { ',' },
14    ↪    StringSplitOptions.RemoveEmptyEntries)
15        .Select(p => p.Trim())
16        .ToList();
17
18    await Task.Run(() =>
19    {
20        try
21        {
22            foreach (var pattern in patterns)
23            {
24                var files = GetFilesSafely(DirectoryPath, pattern);
25
26                foreach (var file in files)
27                {
28                    try
29                    {
30                        string fileHash = ComputeFileHash(file);
31
32                        if (!fileHashes.ContainsKey(fileHash))
33                        {
34                            fileHashes[fileHash] = new List<string>();
35                        }
36                        fileHashes[fileHash].Add(file);
37                    }
38                    catch (Exception ex)
39                    {
40                        StatusMessage = $"Fehler beim Verarbeiten von {file}: {ex.Message}";
41                    }
42                }
43            }
44        }
45    });
46
47    // ... (rest of the method)
48
49    return totalFreedSpace;
50 }

```

Abbildung 27: Methode RemoveDuplicateFilesAsync – Teil 1: Hashing und Duplikaterkennung

```

43     foreach (var hash in fileHashes.Keys)
44     {
45         var duplicateFiles = fileHashes[hash];
46         if (duplicateFiles.Count > 1)
47         {
48             for (int i = 1; i < duplicateFiles.Count; i++)
49             {
50                 filesToDelete.Add(duplicateFiles[i]);
51             }
52         }
53     }
54 }
55 catch (Exception ex)
56 {
57     StatusMessage = $"Fehler beim Durchsuchen des Verzeichnisses: {ex.Message}";
58 }
59 });
60
61 if (filesToDelete.Any())
62 {
63     long totalBytes = filesToDelete.Sum(f => new FileInfo(f).Length);
64     double spaceFreedInMb = totalBytes / (1024.0 * 1024.0);
65     await LogCleanupAsync(filesToDelete.Count, spaceFreedInMb, "Duplikate");
66     await DeleteFilesAsync(filesToDelete, "Löschen der Duplikate");
67 }
68 else
69 {
70     StatusMessage = "Keine Duplikate gefunden.";
71 }
72 }

```

Abbildung 28: Methode RemoveDuplicateFilesAsync – Teil 2: Duplikatlöschung und Abschluss

Die Berechnung der Dateihashes erfolgt über die Methode `ComputeFileHash()`, welche den Inhalt einer Datei einliest und daraus einen eindeutigen MD5-Hash generiert. Dieses Verfahren ist effizient und ermöglicht eine sichere Duplikaterkennung auf Inhaltsebene – unabhängig von Dateinamen oder Speicherort.

```

1 private string ComputeFileHash(string filePath)
2 {
3     using (var md5 = MD5.Create())
4     {
5         using (var stream = File.OpenRead(filePath))
6         {
7             byte[] hash = md5.ComputeHash(stream);
8             return BitConverter.ToString(hash).Replace("-", "").ToLower();
9         }
10    }
11 }

```

Abbildung 29: Screenshot der Methode ComputeFileHash

Die Duplikaterkennung ist besonders nützlich in datenintensiven Umgebungen, in denen Benutzer versehentlich mehrfach identische Dateien erzeugen oder speichern. Durch die gezielte Beseitigung solcher Redundanzen wird nicht nur Speicherplatz eingespart, sondern auch die Übersichtlichkeit der Dateistruktur erhöht.

## 6.2 Authentifizierung und Registrierung

Das Login- und Registrierungssystem der Anwendung ist von zentraler Bedeutung. Besonders das Passwort-Hashing stellt einen sicherheitskritischen Bestandteil dar, da Passwörter aus Datenschutzgründen niemals im Klartext in einer Datenbank gespeichert werden dürfen. Daher war es entscheidend, eine robuste und sichere Lösung zu implementieren, besonders im Hinblick auf die spätere Rollen- und Benutzerverwaltung.

Im Folgenden werden die zentralen Komponenten der Registrierung und Authentifizierung erläutert.

### 6.2.1 Registrierungsvorgang

Die Klasse `RegisterVM` ermöglicht die Erstellung neuer Benutzerkonten. Die eingegebenen Daten werden validiert und in der Datenbank gespeichert. Zusätzlich erhält jeder neue Benutzer automatisch eine Rolle, entweder `User` oder bei bestimmten Bedingungen `Admin`.

```

1 private void ExecuteRegister(object parameter)
2 {
3     if (string.IsNullOrEmpty(Username) || string.IsNullOrEmpty>Password)) {
4         ↪ ... }
5
6     if (Password.Length < 8) { ... }
7
8     if (Password != ConfirmPassword) { ... }
9
10    using (var context = new GarbageCollectorDbContext())
11    {
12        if (context.Users.Any(u => u.Username == Username.ToLower())) { ... }
13
14        var newUser = new User
15        {
16            Username = Username,
17            PasswordHash = HashPassword>Password)
18        };
19
20        context.Users.Add(newUser);
21        ...
22    }
23
24    StatusMessage = "Registrierung erfolgreich";
25 }

```

Abbildung 30: Auszug aus der Methode ExecuteRegister

In diesem Auszug wird deutlich, dass mehrere Schritte zur Validierung stattfinden (einige sind zur besseren Lesbarkeit gekürzt dargestellt). Unter anderem wird überprüft, ob bereits ein Benutzer mit demselben (normalisierten) Benutzernamen existiert. Die Normalisierung mittels `.ToLower()` stellt sicher, dass Groß- und Kleinschreibung ignoriert wird, so kann zum Beispiel ein Benutzername **Moritz** nicht doppelt unter Schreibweisen wie **moRITZ** registriert werden.

### 6.2.2 Sicheres Passwort-Hashing

Ein besonderer Fokus lag auf der sicheren Speicherung der Passwörter. Hier fiel die Wahl auf den modernen Algorithmus Argon2. Dieser nutzt gezielt CPU- und RAM-Ressourcen, um den Aufwand für Brute-Force-Angriffe zu erhöhen. Durch frei wählbare Parameter wie Speichergröße, Iterationen und Parallelisierung kann das Sicherheitsniveau flexibel angepasst werden. Die folgende Methode implementiert das Hashing mittels des NuGet-Pakets `Konscious.Security.Cryptography`:

```

1 private static string HashPassword(string password)
2 {
3     byte[] salt = new byte[16];
4     RandomNumberGenerator.Fill(salt);
5
6     var argon2 = new Argon2id(Encoding.UTF8.GetBytes(password))
7     {
8         Salt = salt,
9         DegreeOfParallelism = 8,
10        Iterations = 4,
11        MemorySize = 65536
12    };
13
14    byte[] hash = argon2.GetBytes(32);
15
16    byte[] hashBytes = new byte[48];
17    Array.Copy(salt, 0, hashBytes, 0, 16);
18    Array.Copy(hash, 0, hashBytes, 16, 32);
19
20    return Convert.ToBase64String(hashBytes);
21 }

```

Abbildung 31: Implementierung des Passwort-Hashings mit Argon2

### 6.2.3 Anmeldevorgang

Beim Anmeldevorgang wird der in der Datenbank gespeicherte Hashwert geladen und mit dem eingegebenen Passwort verifiziert. Dafür wird aus dem gespeicherten Hash zunächst der ursprüngliche Salt extrahiert, der bei der Registrierung verwendet wurde. Anschließend wird das eingegebene Passwort mit diesem Salt erneut mittels Argon2 gehasht. Der Salt ist ein zufällig generierter Wert, meist in Form eines Byte-Arrays, der bei jedem Hashing-Vorgang neu erzeugt wird. Durch seine Einzigartigkeit wird sichergestellt, dass selbst identische Passwörter bei unterschiedlichen Benutzern zu unterschiedlichen Hashwerten führen. Stimmen der neu berechnete und der gespeicherte Hashwert überein, gilt das Passwort als korrekt, und der Benutzer wird erfolgreich authentifiziert.



```

1 private static bool VerifyPassword(string password, string storedHash)
2 {
3     byte[] hashBytes = Convert.FromBase64String(storedHash);
4     byte[] salt = new byte[16];
5     Array.Copy(hashBytes, 0, salt, 0, 16);
6
7     var argon2 = new Argon2id(Encoding.UTF8.GetBytes(password))
8     {
9         Salt = salt,
10        DegreeOfParallelism = 8,
11        Iterations = 4,
12        MemorySize = 65536
13    };
14
15    byte[] hash = argon2.GetBytes(32);
16
17    for (int i = 0; i < 32; i++)
18    {
19        if (hash[i] != hashBytes[16 + i])
20        {
21            return false;
22        }
23    }
24
25    return true;
26 }

```

Abbildung 32: Passwortverifikation durch erneutes Hashing mit Argon2

## 7 Ausblick und Fazit

Mit dem entwickelten Garbage-Collection-Tool hat das Unternehmen *DataFlow Solutions* eine leistungsfähige Lösung geschaffen, um seine IT-Infrastruktur nachhaltig zu entlasten. Temporäre und überflüssige Dateien werden zuverlässig erkannt und entfernt, wodurch Systemressourcen geschont, die Lebensdauer der Hardware verlängert und Betriebskosten eingespart werden können.

Dank der Architektur auf Basis des MVVM-Patterns ist das Tool flexibel erweiterbar und zukunftssicher. Perspektivisch lassen sich weitere Funktionen integrieren, wie zum Beispiel intelligentere Algorithmen zur Dateianalyse, eine automatische E-Mail-Benachrichtigung bei kritischen Datenmengen oder eine erweiterte Benutzer- und Rollenkontrolle.

Im Verlauf dieser Arbeit konnte ich nicht nur ein konkretes Softwareprodukt entwickeln, sondern auch wertvolle Erfahrungen im Bereich der Softwarearchitektur sammeln. Besonders das Zusammenspiel von Benutzeroberfläche und Backend sowie die konsequente Trennung von Logik und Darstellung haben mein Verständnis für nachhaltige Softwareentwicklung geschärft. Dieses Wissen wird mir auch bei zukünftigen Projekten und in meiner beruflichen Laufbahn als Wirtschaftsinformatiker von großem Nutzen sein.

# Quellen und Werkzeuge

## Verwendete Quellen

- Microsoft Docs – .NET und C# Dokumentation
  - <https://learn.microsoft.com>
  - MVVM-Architektur:  
<https://learn.microsoft.com/de-de/dotnet/architecture/maui/mvvm>
  - Die File-Klasse:  
<https://learn.microsoft.com/de-de/dotnet/api/system.io.file?view=net-8.0>
  - Parallele-Programmierung:  
<https://learn.microsoft.com/de-de/dotnet/standard/parallel-programming/>
  - Asynchrone Programmierung:  
<https://learn.microsoft.com/de-de/dotnet/csharp/asynchronous-programming/task-asynchronous-programming-model>
- Argon2 Passwort-Hashing
  - Offizielles Argon2-Repository auf GitHub:  
<https://github.com/P-H-C/phc-winner-argon2>
  - Argon2 auf Wikipedia:  
<https://en.wikipedia.org/wiki/Argon2>
  - Implementierung von Argon2id in C#:  
<https://www.thatsoftwaredude.com/content/14030/implementing-argon2id-password-hashing-in-c>
- Microsoft SQL auf dem Mac mit Docker:  
<https://kb.parallels.com/129699>
- YouTube – MVVM for Beginners: Model-View-ViewModel Architecture for Xamarin.Forms, .NET MAUI, WPF, UWP, & More von James Montemagno:  
[https://youtu.be/Pso1MeX\\_HvI](https://youtu.be/Pso1MeX_HvI)
- YouTube – C# WPF Tutorial #22 – What is MVVM? von Kampa Plays:  
<https://youtu.be/dsjTZxIVxxo>
- YouTube – C# WPF Tutorial #23 – Using ViewModels in MVVM von Kampa Plays:  
<https://youtu.be/Fs2gwb6Dqjk>
- YouTube – C# WPF Tutorial #24 – Using RelayCommand in MVVM von Kampa Plays:  
<https://youtu.be/s7pt3EkDyq4>

- YouTube - C# Async/Await/Task Explained (Deep Dive) von Raw Coding:  
<https://youtu.be/il9gl8MH17s>
- Thomas Claudius Huber: *Windows Presentation Foundation – Das umfassende Handbuch*.  
Rheinwerk Verlag, 5. Auflage, 2019. ISBN 978-3-8362-7202-5

## Verwendete Werkzeuge

- **Visual Studio 2022** - Hauptentwicklungsumgebung (IDE)
- **.NET 8 SDK** - Framework zur Erstellung der WPF-Anwendung
- **Entity Framework Core** - Datenbankzugriff über ORM
- **MSSQL Server und SQL Server Management Studio (SSMS)** - Datenbank und Verwaltung
- **Docker Desktop** - lokale Entwicklungsumgebung für MSSQL
- **Parallels** - Virtualisierung von Windows 11 in MacOS
- **Newtonsoft.Json** - JSON-Serialisierung
- **Konsconscious.Security.Cryptography** - Passwort Verschlüsselung mit Argon2
- **ChatGPT 4o**
- **Material Design In XAML Toolkit** - Design von XAML-Elementen
- **Design.com** - Erstellung des Logos
- **Icons8** - Icons für die Menüpunkte und Buttons
- **Uizard** - Design der grafischen Benutzeroberfläche (GUI)
- **Roboto von GoogleFonts** - verwendete Schriftart innerhalb der Anwendung
- **GitHub** - Versionsverwaltung und Quellcode-Hosting
- **LucidChart** - Erstellung der UML-Diagramme
- **Overleaf** - Erstellung dieser Hausarbeit in LaTeX

# 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 benutzt habe.

---

Datum

---

Unterschrift