HTL Leonding
next level

HÖHERE TECHNISCHE
BUNDESLEHRANSTALT
LEONDING

Unser tolles Thema – wir sind genial

DIPLOMARBEIT

verfasst im Rahmen der

Reife- und Diplomprüfung

an der

Höheren Abteilung für Informatik

Eingereicht von:
Robert Freiseisen
Philipp Füreder

Betreuer:
Rainer Stropek

Leonding, August 2023

Ich erkläre an Eides statt, dass ich die vorliegende Diplomarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe.

Die Arbeit wurde bisher in gleicher oder ähnlicher Weise keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Die vorliegende Diplomarbeit ist mit dem elektronisch übermittelten Textdokument identisch.

Leonding, April 2022 S. Schwammal & S. Schwammal

Abstract

Brief summary of our amazing work. In English. This is the only time we have to include a picture within the text. The picture should somehow represent your thesis. This is untypical for scientific work but required by the powers that are. Suspendisse vel felis. Ut lorem lorem, interdum eu, tincidunt sit amet, laoreet vitae, arcu. Aenean faucibus pede eu ante. Praesent enim elit, rutrum at, molestie non, nonummy vel, nisl. Ut lectus eros, malesuada sit amet, fermentum eu, sodales cursus, magna. Donec eu purus. Quisque vehicula, urna sed ultricies auctor, pede lorem egestas dui, et convallis elit erat sed nulla. Donec luctus. Curabitur et nunc. Aliquam dolor odio, commodo pretium, ultricies non, pharetra in, velit. Integer arcu est, nonummy in, fermentum faucibus, egestas vel, odio.

Zusammenfassung

Zusammenfassung unserer genialen Arbeit. Auf Deutsch. Das ist das einzige Mal, dass eine Grafik in den Textfluss eingebunden wird. Die gewählte Grafik soll irgendwie eure Arbeit repräsentieren. Das ist ungewöhnlich für eine wissenschaftliche Arbeit aber eine Anforderung der Obrigkeit. *Bitte auf keinen Fall mit der Zusammenfassung verwechseln, die den Abschluss der Arbeit bildet!* Suspendisse vel felis. Ut lorem lorem, interdum eu, tincidunt sit amet, laoreet vitae, arcu. Aenean faucibus pede eu ante. Praesent enim elit, rutrum at, molestie non, nonummy vel, nisl. Ut lectus eros, malesuada sit amet, fermentum eu, sodales cursus, magna. Donec eu purus. Quisque vehicula, urna sed ultricies auctor, pede lorem egestas dui, et convallis elit erat sed nulla. Donec luctus. Curabitur et nunc. Aliquam dolor odio, commodo pretium, ultricies non, pharetra in, velit. Integer arcu est, nonummy in, fermentum faucibus, egestas vel, odio.

Inhaltsverzeichnis

1 Einleitung

1.1 Standardsoftware vs Individualsoftware

In der heutigen digitalisierten Welt ist Software nicht mehr wegzudenken, für den Erfolg von Unternehmen oder auch für den privaten Gebrauch. Für diese Diplomarbeit unterscheiden wir zwischen zwei Hauptarten von Software, die von Unternehmen genutzt werden: **Standardsoftware** und **Individualsoftware**. Nun wollen wir diese beiden Softwaretypen in einem direkten Vergleich gegenüberstellen, um ihre jeweiligen Vor- und Nachteile zu zeigen.

Standardsoftware

Standardsoftware bietet Vorteile, die sie für viele Unternehmen und Einzelpersonen zu einer attraktiven Wahl macht. Einer der Hauptvorteile ist die Kosteneffizienz: Da die Entwicklungskosten auf eine Vielzahl von Kunden verteilt werden, sind die initialen Anschaffungskosten in der Regel geringer als bei einer individuell entwickelten Lösung. Ein weiterer Pluspunkt ist die schnelle Implementierung. Da die Software bereits entwickelt ist, kann sie in der Regel schnell installiert und in Betrieb genommen werden, wodurch Zeit und Ressourcen gespart werden. Zudem profitieren Benutzer von einer breiten Community und einem umfangreichen Support, was sowohl die Problembehebung als auch die Weiterentwicklung erleichtert. Hinzu kommt die breite Verfügbarkeit von Schulungsmaterialien und Kursen für gängige Standardsoftware. Diese Ressourcen erleichtern die Einarbeitung und ermöglichen es den Benutzern, das volle Potenzial der Software auszuschöpfen.

Ein Nachteil von Standardsoftware besteht darin, dass sie nicht immer genau den individuellen Anforderungen eines Unternehmens gerecht wird. Es könnte vorkommen, dass spezifische Funktionen fehlen oder die Software nicht optimal auf die Arbeitsprozesse des Unternehmens zugeschnitten ist.

Individualsoftware

Individualsoftware bietet Unternehmen die Möglichkeit, eine Softwarelösung zu erhalten, die vollständig an ihre speziellen Bedürfnisse und Anforderungen angepasst ist. Diese Anpassungsfähigkeit erlaubt nicht nur effizientere Arbeitsprozesse, sondern schafft auch Raum für Flexibilität und Skalierbarkeit. Da die Software im Laufe der Zeit sich ändernden Unternehmensbedürfnissen angepasst werden kann, bleibt sie stets ein dynamisches und anpassungsfähiges Werkzeug. Zudem kann Individualsoftware langfristig kosteneffizient sein, da keine laufenden Lizenzgebühren für nicht benötigte Funktionen anfallen. So vereint Individualsoftware in sich die Vorteile von vollständiger Anpassung, Flexibilität, Skalierbarkeit und wirtschaftlicher Effizienz.

Trotz der vielen Vorteile kommt Individualsoftware oft mit einem wesentlichen Nachteil: den hohen Anschaffungskosten. Die Entwicklung einer maßgeschneiderten Softwarelösung erfordert in der Regel eine erhebliche Investition in Zeit und Ressourcen. Diese initialen Kosten können beträchtlich sein und stellen daher besonders für kleinere Unternehmen oder Organisationen mit begrenztem Budget eine große Hürde dar. Daher ist es wichtig, diese Investition sorgfältig abzuwägen und sie in den Kontext der erwarteten langfristigen Vorteile und Kostenersparnisse zu setzen.

1.2 Problemverständnis

Standardsoftware ist häufig so konzipiert, dass sie den Anforderungen einer breiten Zielgruppe gerecht wird, was jedoch oft dazu führt, dass spezifische Funktionen fehlen, die für einzelne Unternehmen oder Organisationen von Bedeutung sein könnten. Eine individuelle Anpassung dieser Standardsoftware an die Bedürfnisse jedes einzelnen Kunden wäre zwar theoretisch möglich, brächte jedoch erhebliche Nachteile mit sich. Insbesondere würde die Software dadurch zunehmend komplizierter und schwieriger zu pflegen. Jede spezielle Anpassung könnte zu Konflikten mit anderen Funktionen führen oder zukünftige Updates erschweren. Die Software würde an Übersichtlichkeit verlieren und die Fehleranfälligkeit könnte steigen. Darüber hinaus wäre es für die Softwareanbieter schwierig, allen individuellen Anforderungen zu folgen und gleichzeitig eine stabile, einheitliche Version des Produkts zu erhalten. Daher wird bei Standardsoftware oft ein Kompromiss angestrebt, der zwar viele, aber nicht alle Bedürfnisse abdeckt, um die Software so einfach und wartbar wie möglich zu halten.

Der Fokus auf vorhandene Ressourcen ist ein entscheidendes Argument gegen die Implementierung kundenspezifischer Funktionen in einer Softwarelösung für jedes einzelne Unternehmen. Softwareentwicklung umfasst nicht nur die Programmierung selbst, sondern auch die Planung, das Design, die Qualitätssicherung und die Wartung. Unternehmen haben in der Regel begrenzte Entwicklungsressourcen, sowohl in Bezug auf die Anzahl der Entwickler als auch die Zeit und das Budget. Diese Ressourcen müssen sorgfältig auf die Entwicklung von Funktionen konzentriert werden, die den größten Mehrwert für die Mehrheit der Kunden bieten. Das Hinzufügen von kundenspezifischen Funktionen würde diese begrenzten Ressourcen auf Projekte lenken, die nur für eine kleine Anzahl von Nutzern relevant sind, und damit den Wert der Software für die allgemeine Kundschaft potenziell verringern. Infolgedessen müssen Unternehmen Prioritäten setzen und ihre Entwicklungsressourcen auf Aktivitäten fokussieren, die das Produkt als Ganzes verbessern und den meisten Kunden zugutekommen.

1.3 Lösungsansatz: Scripting

Eine effektive Lösung für das Problem von fehlenden Funktionen in Standardsoftware können Anpassungs- und Erweiterungsmöglichkeiten mit Hilfe von Skripting sein. Durch das Bereitstellen von Skripting-Schnittstellen wird den Nutzer/innen ermöglicht, die Funktionalität der Software nach ihren individuellen Bedürfnissen zu erweitern, ohne dass die Kernsoftware selbst modifiziert werden muss. Dies schafft einen Mittelweg zwischen der Flexibilität von individueller Software und der Effizienz und Zuverlässigkeit von Standardlösungen. Benutzer/innen können Skripte schreiben, die spezielle Funktionen hinzufügen, die in der Standardversion fehlen. Diese Methode ist nicht nur ressourceneffizient, sondern ermöglicht auch eine schnellere Anpassung, da sie in der Regel keinen Eingriff in den Code der Software erfordert. So können Unternehmen die Software an ihre speziellen Anforderungen anpassen, während sie gleichzeitig von den Vorteilen der Standardsoftware, wie regelmäßigen Updates und einer breiten Benutzerbasis, profitieren. Skripting stellt daher eine skalierbare und anpassungsfähige Lösung dar, die den Bedürfnissen vieler verschiedener Kunden gerecht werden kann.

Allgemeines zu Scripting

In dieser Diplomarbeit meint der Begriff Scripting den Einsatz von Skriptsprachen, um Automatisierungsprozesse zu steuern, spezielle Funktionen auszuführen oder die Funktionalität einer bestehenden Softwareanwendung zu erweitern. Im Gegensatz zu vollwertigen Programmiersprachen, die in der Regel kompiliert werden müssen und oft für die Entwicklung komplexer Anwendungen verwendet werden, sind Skriptsprachen in der Regel interpretiert, das bedeutet sie werden von einem Interpreter ausgeführt, der den Code Zeile für Zeile umsetzt. Moderne Scriptsprachen wie Javascript werden jedoch auch bei jeder Ausführung kompiliert. Sie sind oft einfacher zu erlernen und schneller zu implementieren als vollwertige Programmiersprachen, was sie ideal für kleinere Aufgaben und schnelle Anpassungen macht. Skripting wird in einer Vielzahl von Kontexten verwendet, darunter Webentwicklung, Systemadministration und Datenanalyse.

Ein zentraler Aspekt dieser Diplomarbeit war die Untersuchung der Rolle des Skripting bei der dynamischen Anpassung von .NET-Anwendungen. Der Einsatz von Skriptsprachen ermöglicht es, fehlende oder zusätzliche Funktionen zur Laufzeit in eine Anwendung zu integrieren. Im Rahmen unserer Arbeit wurde mit verschiedenen Skript-

sprachen experimentiert, um deren Eignung für die Integration in .NET-Anwendungen zu bewerten.

Vorteile

- **Schnelle Entwicklung und Anpassung:** Skriptsprachen sind in der Regel leicht zu erlernen und ermöglichen eine schnelle Entwicklung, was ideal für die Anpassung und Erweiterung von Software ist.
- **Flexibilität:** Skripting ermöglicht es Benutzern und Entwicklern, die Funktionalität einer Anwendung nach Bedarf zu ändern, ohne die ursprüngliche Software zu modifizieren.
- **Automatisierung:** Eines der Hauptanwendungsgebiete von Skripting ist die Automatisierung wiederkehrender Aufgaben, wodurch Zeit und Ressourcen gespart werden können.
- **Kostenersparnis:** Da Skripting meist schneller und mit weniger Aufwand umsetzbar ist, kann es kosteneffizienter sein als die Entwicklung von kundenspezifischen Softwarelösungen.

Nachteile

- **Leistungsprobleme:** Skriptsprachen sind oft langsamer als kompilierte Sprachen, da sie interpretiert werden. Dies kann bei rechenintensiven Aufgaben ein Problem darstellen.
- **Kompatibilitätsprobleme:** Nicht alle Skripting-Sprachen oder -Tools sind mit allen Plattformen oder Anwendungen kompatibel. Manchmal sind spezielle Anpassungen erforderlich.
- **Fehlermeldungen:** Durch Scripts enstehen oft komplex zu diagnostizierende Fehler, die den Support auf der Herstellerseite erschweren.

Levels von Scripting

In Microsoft Excel gibt es die Möglichkeit sich wiederholende Aufgaben mithilfe von Office Scripts zu automatisieren.
Man kann auf zwei Arten ein neues Office-Skript erstellen:

- Man kann seine Handlungen mithilfe des Actionrecorders aufnehmen. Diese Methode eignet sich besonders gut, wenn man sich wiederholende Schritte in dem Dokument merken möchte. Dazu sind keine Programmierkenntnisse oder ähnliches erforderlich. Die Aufgezeichneten Skripte können abgespeichert und verändert werden.
- Die zweite Möglichkeit ist, dass Office-Skript selbst mithilfe von TypeScript zu schreiben.

[1]

Folgendes Office-Skript Beispiel gibt den Wert von der Zelle A1 auf der Konsole aus:

Listing 1: Office-Skript

```
1      function main(workbook: ExcelScript.Workbook) {
2    // Get the current worksheet.
3    let selectedSheet = workbook.getActiveWorksheet();
4
5    // Get the value of cell A1.
6    let range = selectedSheet.getRange("A1");
7
8    // Print the value of A1.
9    console.log(range.getValue());
10 }
```

[2]

1.4 Alternativen

Scripting ist eine weitverbreitete Möglichkeit zur Automatisierung und Steuerung von Aufgaben. Obwohl Scripting in vielen Kontexten als effizient und flexibel gilt, ist es nicht die einzige verfügbare Methode zur Problemlösung. Tatsächlich gibt es eine Reihe von Alternativen zu Scripting, die in verschiedenen Anwendungsfällen Vorteile bieten können. Diese Alternativen können von grafischen Benutzeroberflächen für die Automatisierung bis hin zu deklarativen Programmieransätzen und Frameworks reichen. Dieses Kapitel beschreibt einige dieser Alternativen, ohne eine tiefgreifende Evaluation ihrer Eignung oder Effizienz vorzunehmen, um einen Überblick über die Möglichkeiten in diesem Bereich zu bieten.

Microsoft Power Automate

Power Automate ist eine cloudbasierte Plattform, die es Anwender/innen unkompliziert ermöglicht, Workflows zu erstellen. Diese Arbeitsabläufe automatisieren zeitaufwändige geschäftliche Aufgaben und Prozesse, indem sie Anwendungen und Dienste verbinden. [3]

Logik-Apps (ein Service von Azure), präsentiert vergleichbare Eigenschaften wie Power Automate. Zusätzlich dazu bietet es weitere Leistungsmerkmale, darunter die nahtlose Einbindung in den Azure Resource Manager, das Azure-Portal, PowerShell, die xPlat-CLI, Visual Studio und diverse weitere Verbindungselemente. [3]

Mit folgenden Dienstleistungen können Power Automate verbunden werden:

- Sharepoint
- Dynamics 365
- OneDrive
- Google Drive
- Google Sheets
- und noch einige mehr

Power Automate ist außerdem plattformübergreifend und kann auf allen modernen Geräten und Browsern ausgeführt werden. Zur Verwendung ist nur ein Webbrowser und eine E-Mail-Adresse erforderlich. [3]

Dynamische Modulsysteme

Ein Ansatz, der oft angewandt wird, um Anwendungen oder Systeme zu konstruieren, ist die Nutzung eines dynamischen Modulsystems. Diese Methode ermöglicht die Erstellung von Applikationen durch den Zusammenbau wiederverwendbarer Einzelmodule. Diese Herangehensweise erleichtert die Entwicklung komplexer Systeme, ohne dass jedes Mal von Grund auf ein völlig neues System erstellt werden muss. Ein weiterer Nutzen besteht darin, dass verschiedene Anwendungen oder Systeme auf diese Weise miteinander verknüpft werden können. Modulsysteme werden oft in der Softwareentwicklung, Webentwicklung, Datenbanken und Systemadministration verwendet. [4]

Beispiele für Modulsysteme:

Webanwendung

- Javascript-Frameworks wie React und Angular

Desktop- und Serveranwendung

- .NET und Java

Erstellung und Verwaltung von Datenbanken

- PostgreSQL und MongoDB

Verwaltung von Netzwerken und Systemen

- Puppet und Chef

[4]

Microservices

Microservices stellen einen Ansatz in der Softwareentwicklung dar. Er basiert darauf, dass Software aus einer Vielzahl kleiner, eigenständiger Dienste besteht. Diese Dienste interagieren miteinander über klar definierte Schnittstellen. Die Architektur von Microservices trägt dazu bei, Skalierbarkeit zu erleichtern und die Zeitspanne für die Entwicklung von Anwendungen zu verkürzen. Dies ermöglicht eine schnellere Umsetzung von Innovationen und beschleunigt die Einführung neuer Funktionen auf dem Markt. [5]

Eigenschaften von Microservices

Eigenständigkeit: In einer Architektur von Mikroservices besteht die Möglichkeit, jeden einzelnen Komponentenservice unabhängig zu entwickeln, bereitzustellen, zu betreiben und zu skalieren. Hierbei hat die Veränderung eines Services keine Auswirkungen auf die Funktionalität anderer Services. Es ist nicht erforderlich, dass Services Code oder Implementierungen miteinander teilen. Die Interaktion zwischen den verschiedenen Komponenten erfolgt ausschließlich über eindeutig definierte APIs. [5]

Spezialisierung: Jeder einzelne Service ist darauf ausgerichtet, eine spezifische Gruppe von Problemstellungen zu lösen. Sollte man im Laufe der Zeit zusätzlichen Code zu einem solchen Dienst hinzufügen, und dadurch der Service zu komplex wird, besteht die Option, diesen in kleinere Services aufzuteilen.

[5]

Dynamisches Laden von Assemblies

.NET bietet die Möglichkeit, Assembly-Dateien zur Laufzeit zu laden. Dies ermöglicht es, neue Funktionen in Form von Klassen und Methoden hinzuzufügen. Assemblies stellen die Grundbausteine von .NET-Anwendungen dar und treten in Form von ausführbaren Dateien (.exe) oder Dynamic Link Library-Dateien (.dll) auf. [6]

Folgender Beispielcode dient dazu, eine Assembly mit der Bezeichnung "example.dll" innerhalb der aktuellen Anwendungsdomäne zu laden. Anschließend wird ein Typ mit dem Namen "Example" aus dieser Assembly abgerufen. Auf diesen abgerufenen Typ wird die Methode "MethodA" ausgeführt. [6]

Listing 2: Assembly

```
1       using System;
2       using System.Reflection;
3
4       public class Asmload0
5       {
6           public static void Main()
7           {
8               // Use the file name to load the assembly into the current
9               // application domain.
10              Assembly a = Assembly.Load("example");
11              // Get the type to use.
12              Type myType = a.GetType("Example");
13              // Get the method to call.
14              MethodInfo myMethod = myType.GetMethod("MethodA");
15              // Create an instance.
16              object obj = Activator.CreateInstance(myType);
17              // Execute the method.
18              myMethod.Invoke(obj, null);
19          }
20      }
```

[6]

Plugins

Durch die Nutzung von Plugins eröffnet sich die Option, individuelle Funktionen zu entwickeln und anschließend in die Anwendung einzubauen. Dies eröffnet ein Fenster für eine dynamische Erweiterbarkeit, ohne auf die Implementierung von Skriptcode zurückgreifen zu müssen. Diese Methode gewährt eine flexible Herangehensweise an die Erweiterung und Anpassung von Funktionalitäten, während gleichzeitig die Sauberkeit der Gesamtlösung erhalten bleibt. [7]

Plugin Framework for .NET Core

Mit Plugin Frameworks für .NET Core wird jegliches Element zu einem Plugin. Das Plugin Framework stellt eine erstklassige Plattform für Plugins in .NET Core-Anwendungen dar, was sowohl ASP.NET Core, Blazor, WPF, Windows Forms als auch Konsolenanwendungen miteinschließt. Diese Plattform zeichnet sich durch eine geringe Systembelastung aus und bietet eine nahtlose Einbindungsmöglichkeit. Es kann verschiedene Plugin-Kataloge unterstützen, darunter .NET-Assemblies, NuGet-Pakete sowie Roslyn-Skripte. Die Flexibilität dieses Frameworks ermöglicht es, eine breite Palette an Anwendungsfällen zu bedienen und die Erweiterungsfunktionalität auf mehreren Ebenen zu steigern. [7]

Features:

- Einfache Integration in eine neue oder bestehende .NET Core-Anwendung
- Automatische Verwaltung von Abhängigkeiten
- Behandlung von plattformspezifischen Laufzeit-DLLs bei Verwendung von Nuget-packages

[7]

1.5 Beschreibung über die Durchführung der Diplomarbeit

Die Durchführung unserer Diplomarbeit wurde in mehrere Phasen unterteilt, die im Folgenden beschrieben werden.

Phase 1: Vorbereitung und Recherche
Zu Beginn des Projekts haben wir regelmäßige Meetings abgehalten, um die notwendige Recherche für den praktischen Teil unserer Arbeit durchzuführen. Diese Treffen ermöglichten es uns, eine gemeinsame Grundlage für die zu bewertenden Kriterien und den damit verbundenen Forschungsaufwand zu schaffen.

Phase 2: Erstellung des Kriterienkatalogs zum Vergleich von Scripting-Möglichkeiten für .NET
Nachdem wir genügend Informationen gesammelt hatten, fokussierten wir uns auf die Entwicklung eines Kriterienkatalogs. In dieser Phase teilten wir die Arbeit auf, wobei jeder von uns sich zwei Skriptsprachen aussuchte, um diese anhand des Kriterienkatalogs zu bewerten. Die regelmäßigen Meetings dienten als Checkpoints, um den Fortschritt zu überwachen und eventuelle Anpassungen am Kriterienkatalog vorzunehmen.

Phase 3: Programmierarbeit
Mit einem vollständigen Kriterienkatalog in der Hand begannen wir mit der Programmierung der einer Musteranwendung, in der der Praxiseinsatz von verschiedenen Scripting-Möglichkeiten untersucht werden sollte. Da wir bereits ein solides Verständnis der ausgewählten Skriptsprachen hatten, konnten wir effizient an der Umsetzung der praktischen Komponente unserer Diplomarbeit arbeiten.

Phase 4: Zwischenstand und Anpassungen

Während der Programmierphase hielten wir kurze, aber regelmäßige Meetings ab, um den aktuellen Stand der Arbeit zu besprechen. Diese Treffen ermöglichten es uns, eventuelle Herausforderungen oder Probleme frühzeitig zu identifizieren und entsprechend anzugehen.

Phase 5: Fertigstellung der Anwendung

Schließlich, nach mehreren Iterationen und Anpassungen, konnten wir unsere Beispielanwendung erfolgreich abschließen. Die finale Version erfüllte alle Kriterien, die in unserem Kriterienkatalog festgelegt waren, und diente als praktische Umsetzung der in der Diplomarbeit erworbenen Erkenntnisse.

1.6 Machbarkeitsnachweis (Beispielanwendung)

Unsere Beispielanwendung bietet Lehrer/innen eine Lösung zur automatisierten Notenberechnung. Anstatt sich auf vorgegebene Algorithmen oder Excel-Tabellen zu verlassen, können Lehrkräfte ihre eigenen individuellen Skripte direkt in die Anwendung hochladen. Diese Skripte können in vier verschiedenen Programmiersprachen verfasst sein (Javascript, Lua, Ironpython und C#-Skript) und ermöglichen eine anpassbare Berechnungslogik, die speziell auf die Anforderungen des jeweiligen Kurses zugeschnitten ist. Einmal hochgeladen, werden die Skripte zur Laufzeit ausgeführt, was eine flexible und zeitnahe Anpassung der Bewertungskriterien ermöglicht. Dies schafft eine hohe Flexibilität in der Notenberechnung, die es ermöglicht, unterschiedliche Berechnungsmethoden für Noten auf Basis der Leistungen von Schüler/innen umzusetzen.

Unsere Anwendung dient als Demonstrationswerkzeug und zeigt, wie individuelle Skripte zur Laufzeit in ein System integriert werden können. Im Kontext unserer Anwendung ermöglichen wir Lehrpersonen, eigene Skript-Files zur Notenberechnung hochzuladen und anzuwenden.

Zielsetzung

Das Hauptziel dieser Beispielanwendung ist es, die Machbarkeit und Flexibilität der Laufzeitintegration von Skriptes darzustellen. Sie richtet sich an Entwickler/innen, Bildungseinrichtungen und alle, die an anpassbaren Softwarelösungen interessiert sind.

In diesem Fall dient die Anwendung primär als Machbarkeitsnachweis für die Laufzeitintegration von benutzerdefinierten Skripts in Softwareanwendungen. Durch die Möglichkeit, individuelle Skripte zur Laufzeit zu integrieren und auszuführen, wollen wir zeigen, dass eine solche Technologie nicht nur realisierbar, sondern auch in verschiedenen Anwendungsbereichen, einschließlich des Bildungswesens, vielseitig einsetzbar ist. Dieses Projekt soll als Grundlage für zukünftige Entwicklungen dienen, die diese Technik in voll funktionsfähigen, benutzerorientierten Lösungen implementieren könnten. Es geht also nicht darum, eine marktreife Produktlösung zu präsentieren, sondern vielmehr die technologischen Möglichkeiten und die damit verbundene Flexibilität dieses Ansatzes zu veranschaulichen.

Funktionalitäten

Hochladen von Benutzerskripten:

- Nutzer/innen können ein eigenes Skript in der unterstützten Skriptsprache hochladen. Dieses dient als Basis für die individuelle Notenberechnung.

Speicherung in der Datenbank:

- Nach der Übermittlung wird das Skript in unserer Datenbank gespeichert, was zukünftigen Zugriff und Wiederverwendung ermöglicht.

Dynamische Ausführung:

- Das hochgeladene Skript wird zur Laufzeit interpretiert und angewandt, um die Noten der Schüler/innen gemäß den im Skript festgelegten Kriterien zu berechnen.

Ergebnisvisualisierung:

- Die berechneten Noten werden dargestellt und können für weitere Analysen gespeichert werden.

2 Evaluierung von Skriptsprachen

2.1 Auswahl der Skriptsprachen

Die Wahl der Scriptsprachen wurde auf Grund von Internet-Recherchen und Fachgesprächen mit Mitschüler*innen und Professor*innen gefällt.
Folgende Scriptsprachen wurden gewählt:

- Lua
- IronPython
- C#script
- Javascript

2.1.1 Lua

Lua ist eine Programmiersprache, die für ihre hohe Ausführungsgeschwindigkeit geschätzt wird. Diese Schnelligkeit, kombiniert mit dem geringen Speicherbedarf der Sprache, macht sie ideal für ressourcenbeschränkte Umgebungen und eingebettete Systeme. Lua bietet eine "von Haus aus"-Nutzbarkeit, die es Entwicklern ermöglicht, sofort nach der Installation loszulegen, ohne sich um eine Vielzahl von Abhängigkeiten kümmern zu müssen. Ihre Einbettbarkeit ist ein weiteres Kernelement, das sie besonders attraktiv für Softwareprojekte macht, die eine integrierte Skriptsprache benötigen. Besonders in der Spieleindustrie hat Lua sich als beliebte Wahl für das Scripting etabliert. Hier ermöglicht es Entwicklern, schnell interaktive und flexible Spielmechanismen zu implementieren, ohne die Hauptspiellogik zu beeinträchtigen. Als Open-Source-Software steht Lua zudem einer breiten Entwicklergemeinschaft zur Verfügung, die zur kontinuierlichen Verbesserung und Erweiterung der Sprache beiträgt. All diese Aspekte machen Lua zu einer vielseitigen Option für eine Reihe von Anwendungen, insbesondere für das Scripting in Videospielen, wie in "World of Warcraft" oder "Roblox". [8] [9] [10] [11]

2.1.2 IronPython

IronPython ist eine Open-Source-Implementierung der Programmiersprache Python, die auf der .NET-Plattform läuft. Es wurde ursprünglich von Jim Hugunin entwickelt und ist eng mit Microsoft assoziiert. IronPython ist in C# geschrieben und ermöglicht die nahtlose Integration von Python-Code mit .NET-Anwendungen. Mit IronPython können Entwickler sowohl auf .NET-Bibliotheken als auch auf Python-Bibliotheken zugreifen, wodurch eine erweiterte Interoperabilität und Flexibilität erreicht wird. IronPython unterstützt sowohl dynamische als auch statische Sprachfunktionen, und es ist durch die Common Language Runtime (CLR) von Microsoft vollständig integriert. Dies ermöglicht eine effiziente Ausführung von Python-Code auf der .NET-Plattform und erleichtert die Erstellung gemischter Anwendungen, die sowohl Python- als auch .NET-Code nutzen. Beim Speicherverbrauch ist IronPython in der Regel nicht so effizient wie CPython, da das .NET mehr Overhead haben kann. Durch die Nutzung der Dynamic Language Runtime (DLR) von .NET kann IronPython dynamische Typen und späte Bindungen effizienter verwalten als einige andere Implementierungen, wie CPython. Dies ermöglicht eine enge Integration mit .NET-Bibliotheken und erleichtert die schnelle Entwicklung und Iteration von Code. [12] [13] [14]

2.1.3 C#script

C#Script ist ein Bestandteil der .NET-Plattform. Es ermöglicht die dynamische Ausführung von C#-Code und ist in den .NET-Bibliotheken und -Laufzeitumgebungen enthalten. Einer der großen Vorteile von C#Script ist, dass es sowohl in gehosteten als auch in eigenständigen Ausführungsmodellen eingesetzt werden kann. In einem gehosteten Modell kann das Skript innerhalb einer bestehenden Anwendung laufen und Objekte und Funktionen der Anwendung nutzen oder modifizieren. In einem eigenständigen Modell kann das Skript als unabhängige Anwendung ausgeführt werden. Ein weiteres wichtiges Merkmal ist die Kompatibilität mit .NET 5/Core und höheren Versionen. Dies ermöglicht eine bessere Leistung, erweiterte APIs und die Möglichkeit, plattformübergreifende Anwendungen zu entwickeln. Die Bibliothek stellt eine Schnittstelle bereit, die die Integration von C#-Skripten in eine Vielzahl von Anwendungsdomänen vereinfacht.

[15]

2.1.4 Javascript

JavaScript ist eine populäre und vielseitige Programmiersprache, die vor allem in der Webentwicklung eingesetzt wird. Sie zeichnet sich durch ihre Einfachheit und Benutzerfreundlichkeit aus, was sie besonders für Einsteiger attraktiv macht. Die Sprache ist intuitiv aufgebaut, sodass die grundlegenden Konzepte schnell verstanden und angewendet werden können. Ein weiterer Vorteil ist, dass alle modernen Webbrowser eine JavaScript-Engine integriert haben. Dies ermöglicht es Entwicklern, Code unmittelbar im Browser auszuführen und zu testen, ohne zusätzliche Software installieren zu müssen. In Bezug auf die Sicherheit bietet JavaScript zwar einige Mechanismen, wie die Ausführung in einer Sandbox-Umgebung, um den Zugriff auf das Betriebssystem des Nutzers zu beschränken. Insgesamt bietet JavaScript eine ausgewogene Mischung aus Benutzerfreundlichkeit, Intuitivität und Funktionalität, allerdings müssen Entwickler stets wachsam in Bezug auf Sicherheitsrisiken sein.

Als .NET-Bibliothek wurde sich für "JavaScriptEngineSwitcher" mit "Jint" entschieden. "JavaScriptEngineSwitcher" ist eine .NET-Bibliothek, die es ermöglicht, verschiedene JavaScript-Engines zu verwenden, und "Jint" ist eine JavaScript-Engine für .NET.

[16] [17]

2.2 Kriterienkatalog

Jede der untersuchten Sprachen hat ihre eigenen Vor- und Nachteile, und die Auswahl der besten Option kann je nach Projektanforderungen variieren. In diesem Abschnitt betrachten wir diese vier Sprachen im Kontext von .NET unter verschiedenen Aspekten:

- Aktivität der Entwicklung:
  - Wie lebendig und aktiv ist die Community hinter der Sprache?
  - Wie oft werden Updates veröffentlicht, und wie gut ist die Dokumentation?
- Funktionalität:
  - Welche Features bietet die Sprache, und wie reichhaltig ist ihr Ökosystem?
  - Gibt es umfangreiche Bibliotheken, die die Entwicklung erleichtern?
- Einsetzbarkeit:
  - Wie einfach lässt sich die Sprache in bestehende oder neue .NET-Projekte integrieren?
  - Welche Voraussetzungen müssen erfüllt sein, und wie komplex ist die Integration?
- Performance:
  - Wie steht es um die Laufzeit-Performance des Codes?
  - Inwiefern beeinflusst die Wahl der Sprache die Geschwindigkeit und Ressourceneffizienz der fertigen Anwendung?
- Debugging:
  - Welche Möglichkeiten bietet die Sprache für das Debugging von Code?
  - Wie effektiv lassen sich Fehler finden und beheben, und welche Werkzeuge stehen zur Verfügung?

Alle Daten wurden auf zwei unterschiedlichen Geräten gemessen, da somit die Messungen gleichzeitig durchgeführt werden konnten. Weil zwei verschiedene PCs für einen Vergleich genutzt wurden, kann dies die Vergleichbarkeit der Ergebnisse beeinflussen.

Folgende Gründe können dabei ein Faktor sein:

- Hardware-Unterschiede:
  - Verschiedene PCs können erhebliche Hardware-Unterschiede aufweisen, darunter Prozessorgeschwindigkeit, Arbeitsspeicher, Grafikkarte, Festplattenkapazität und andere technische Spezifikationen. Diese Unterschiede können die Leistungsfähigkeit und Geschwindigkeit der beiden PCs stark beeinflussen.
- Betriebssystem und Software:
  - Verschiedene PCs können unterschiedliche Betriebssysteme (z.B. Windows, macOS, Linux) und Softwarekonfigurationen aufweisen. Dies kann die Vergleichbarkeit der Ergebnisse beeinflussen, da bestimmte Anwendungen oder Aufgaben auf unterschiedlichen Betriebssystemen möglicherweise unterschiedlich ausgeführt werden.
- Treiber und Updates:
  - Die Installation und Aktualisierung von Treibern und Software kann zwischen verschiedenen PCs variieren. Dies kann zu unterschiedlichem Verhalten von Hardwarekomponenten und zur Auswirkung auf die Leistung führen.
- Alter und Verschleiß:
  - Ältere PCs könnten aufgrund von Abnutzung und Alterungsprozessen möglicherweise nicht mehr die gleiche Leistung erbringen wie neuere Modelle. Dies kann zu Unterschieden in der Vergleichbarkeit der Ergebnisse führen.
- Systemkonfiguration:
  - Die Konfiguration von Betriebssystemeinstellungen, Energieeinstellungen und Hintergrundanwendungen kann zwischen den beiden PCs variieren, was sich auf die Ressourcennutzung und somit auf die Vergleichbarkeit auswirken kann.

Die Daten für IronPython und Lua wurden auf den PC von Robert Freiseisen unter folgenden Voraussetzungen gemessen:

- Gerätspezifikationen:

Hersteller	HP
Gerätname	LAPTOP-5U1879KR
Prozessor	Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz 1.80 GHz
Installierter RAM	8.00 GB (7.89 GB verwendbar)
Produkt-ID	00325-81357-65742-AAOEM
Systemtyp	64-Bit-Betriebssystem, x64-basierter Prozessor

- Betriebssystem:

Edition	Windows 11 Home
Version	21H2
Installiert am	24.11.2021
Betriebssystembuild	220.001.098
Leistung	Windows Feature Experience Pack 1000.22000.1098.0

Die Daten für C#script und Javascript wurden auf den PC von Philipp PC B unter folgenden Voraussetzungen gemessen:

- Gerätspezifikationen:

Hersteller	Selber Gebaut
Gerätname	PC-Philipp
Prozessor	AMD Ryzen 5 1600 Six-Core Processor 3.20 GHz
Installierter RAM	16.00 GB
Produkt-ID	00330-80000-00000-AA542
Systemtyp	64-Bit-Betriebssystem, x64-basierter Prozessor

- Betriebssystem:

Edition	Windows 10 Pro
Version	22H2
Installiert am	17.08.2020
Betriebssystembuild	19045.3393
Leistung	Windows Feature Experience Pack 1000.19044.1000.0

2.2.1 Aktivität der Entwicklung

In der nachfolgenden Tabelle wird dargestellt, wie intensiv die Entwicklerteams an den unterschiedlichen Scriptsprachen arbeiten und ob diese auch die Nuget-Pakete entwickeln. Es ist zu beachten, dass die Daten am 30.11.2022 erfasst wurden.

Aktivität	IronPython	Lua	C#Scripting	Javascript
Commits in den letzten 10 Monaten	179	27	702	1153
Nuget-Packages vom Sprachentwicklerteam selber	Nein	Nein	Ja	Nein
Releases in den letzten 10 Monaten	2 (2.7.1 und 3.4.0-beta1)	1 (v5.4.4)	v4.0.0 und höher	v4.6.2 (und höher)
Unterstützt aktuelle major Versionen von Scriptsprache	Ja	Ja	Ja	Ja
Unterstützt aktuelle Versionen von .NET	Ja	Ja	Ja	Ja

2.2.2 Einsetzbarkeit

Alle untersuchten Bibliotheken sind auf Windows, MAC und Linux lauffähig.

2.2.3 Performance

Der Speicherplatz auf dem Laufwerk wurde aus dem Windows-File-Explorer entnommen. Die Geschwindigkeitsmessungen erfolgten mit BenchmarkDotNet. BenchmarkDotNet wird im Abschnitt Verwendete Technologien näher erleutert.

IronPython und NLua müssen externe Runtimes (für Python bzw. Lua) in die .NET-Umgebung integrieren, was zu zusätzlichem Overhead und Speicherbedarf führt. Im Gegensatz dazu verwenden C#Script und JavaScriptEngineSwitcher die bereits in .NET integrierte C#-Sprache bzw. die .NET Common Language Runtime, was einen effizienteren Speicherverbrauch ermöglicht.

Performance	IronPython	Lua	C#Scripting	Javascript
Speicher einer einfachen Anwendung	13 MB	7.3 MB	168 KB	416 KB
Durchnittliche Laufzeit einer einfachen Anwendung	137.118 μs	8.088 μs	39.59 ms	128.14 ms
Durchnittliche Laufzeit einer Additionsfunktion	2340.688 μs	10.053 μs	39.59 ms	29.77 ms
Durchnittliche Laufzeit von Übergabe eines .NET-Objekts	9054.007 μs	7548.497 μs	66.72 ms	46.27 ms

Es folgen nun die Resultate von BenchmarkDotNet und der ausgeführte Code.

Für:

Lua und IronPython

Method	Mean	Error	StdDev
TestIronPython	137.118 μs	7.5278 μs	21.959 μs
TestIronPythonSum	2,340.688 μs	71.9924 μs	208.863 μs
TestLua	8.088 μs	0.3702 μs	1.020 μs
TestLuaSum	10.053 μs	0.4560 μs	1.330 μs
TestLua-PassDotNetObject-AndCallFunction	7,548.497 μs	1,258.6385 μs	3,711.124 μs
TestIronPython-PassDotNetObject-AndCallFunction	9,054.007 μs	471.9551 μs	1,346.514 μs

Der Code für die Methoden von IronPython:

Listing 3: IronPythonTestMethods

```
1               private readonly ScriptEngine engine = Python.CreateEngine();
2               [Benchmark]
3               public void TestIronPython() => IronPythonSimple();
4               [Benchmark]
5               public void TestIronPythonSum() => IronPythonSum();
6               [Benchmark]
7               public void TestIronPythonPassDotNetObjectsAndCallFunction() =>
                    IronPythonPassDotNetObjectsAndCallFunction();
8
9               #endregion
10
11              private readonly Lua state = new Lua();
12              #region Lua
13              [Benchmark]
14              public void TestLua() => LuaSimple();
15              [Benchmark]
16              public void TestLuaSum() => LuaSum();
17              [Benchmark]
18              public void TestLuaPassDotNetObjectsAndCallFunction() =>
                    LuaPassDotNetObjectAndCallFunction();
19              #endregion
20
21              #region IronPythonMethods
22              public void IronPythonSimple()
23              {
24                  var scope = engine.CreateScope();
25                  var source = engine.CreateScriptSourceFromString("def
                        fun():\r\n\treturn 42\r\n\r\n\r\ntest = fun()");
26                  source.Execute(scope);
27                  var test = scope.GetVariable("test");
28                  Console.WriteLine(test);
29              }
30
31              public void IronPythonSum()
32              {
33                  var scope = engine.CreateScope();
34                  var source = engine.CreateScriptSourceFromString("def
                        sum(x,y):\r\n\treturn x+y\n\r\n\rresult = sum(3,3)");
35                  source.Execute(scope);
36                  var result = scope.GetVariable("result");
37                  Console.WriteLine(result);
38              }
39
40              public void IronPythonPassDotNetObjectsAndCallFunction()
41              {
42                  var scope = engine.CreateScope();
43                  var obj = new SomeClass();
44                  scope.SetVariable("obj", obj);
45                  var source =
                        engine.CreateScriptSourceFromString(SetPythonScript());
46                  source.Execute(scope);
47                  var result = scope.GetVariable("result");
48                  Console.WriteLine(result);
49              }
50
51              private static string SetPythonScript()
52              {
53                  string s = "";
54                  s += "import clr" + "\r\n";
55                  s += "result=obj.Func1() " + "\r\n";
56                  return s;
57              }
```

Der Code für die Methoden von NLua:

Listing 4: NluaTestMethods

```
1       private readonly Lua state = new Lua();
2
3
4       public void LuaPassDotNetObjectAndCallFunction()
5       {
6           var obj = new SomeClass();
7           state["obj"] = obj;
8           state.DoString (@"result=obj:Func1()");
9           var result = state["result"];
10          Console.WriteLine(result);
11      }
12
13      public void LuaSimple()
14      {
15          state.DoString("function fun() \r\n\t  return 42 \r\n end \r\n
                test=fun()");
16          var test = state["test"];
17          Console.WriteLine(test);
18      }
19
20      public void LuaSum()
21      {
22
23
24          state.DoString("function sum(x,y) \r\n\t return x+y \r\n end \r\n
                result=sum(3,3)");
25          var result = state["result"];
26          Console.WriteLine(result);
27      }
```

C#script

Method	Mean	Error	StdDev
TestCsharpSimple	39.59 ms	0.354 ms	0.331 ms
TestCsharpSum	39.56 ms	0.321 ms	0.301 ms
TestCsharp- Objects	66.72 ms	0.875 ms	0.819 ms

Listing 5: #ScriptingTestMethods

```
1               public static async void ReturnNumber()
2           {
3               var state = await CSharp#Script.RunAsync("return 42;");
4               Console.WriteLine(state.ReturnValue);
5           }
6           public static async void MySum()
7           {
8               var state = await CSharpScript.RunAsync("return 3 + 3;");
9               Console.WriteLine(state.ReturnValue);
10          }
11          public static async void DotNetObject()
12          {
13              var obj = new Student("Hans", 18);
14
15              var globals = new Globals { student = obj };
16              var state = await CSharpScript.RunAsync("", globals: globals);
17          }
```

Javascript

Method	Mean	Error	StdDev
TestJavascriptSimple	128.14 ms	1,102.64 ms	286.35 ms
TestJavaScriptSum	29.77 ms	255.31 ms	66.30 ms
TestJavascript-DotNetObjects	46.27 ms	397.72 ms	103.29 ms

Listing 6: JavascriptTestMethods

```
1               public void JavascriptSimple()
2               {
3                   var engine = new JintJsEngine();
4                   engine.Execute(@"
5                                   function myFunction() {
6                                       return 42;
7                                   }");
8               }
9               public void JavascriptSum()
10              {
11                  var engine = new JintJsEngine();
12
13                  engine.Execute(@"
14                                  function mySum(x,y) {
15                                      return x+y;
16                                  }
17                                  var x = mySum(3,3);");
18              }
19              public void DotNetObjects()
20              {
21                  var engine = new JintJsEngine();
22                  var obj = new Student("Hans", 18);
23                  var student = JsonConvert.SerializeObject(obj);
24                  engine.SetVariableValue("student", student);
25              }
```

2.2.4 Funktionalität

In der nachfolgenden Tabelle sind die Recherche-Ergebnisse hinsichtlich der Funktionalität der Scriptsprachen in .NET dargestellt. Die Informationen wurden aus den offizellen Webseiten der Nuget-Pakete entnommen.

Funktion	IronPython	Lua	C#Scripting	Javascript
Kann auf .NET Variablen zugreifen	Ja	Ja	Ja	Ja
Kann globale Variablen	Ja	Ja	Ja	Ja
Unterstützt Erweiterungspakete der Scriptsprache	Ja (nicht numpy und pandas!)	Ja	Ja	Ja

2.2.5 Debugging

Das Debugging durch die Ausgabe auf der Konsole ist mit allen, in dieser Diplomarbeit untersuchten, Scriptsprachen möglich.

Breakpoints in VisualStudio bzw. in VisualStudio-Code sind nur mit IronPython möglich.

Es folgen nun Screenshots, um zu beweisen, dass die Angaben stimmen.

- Der Beweis für das Debugging mit NLua:

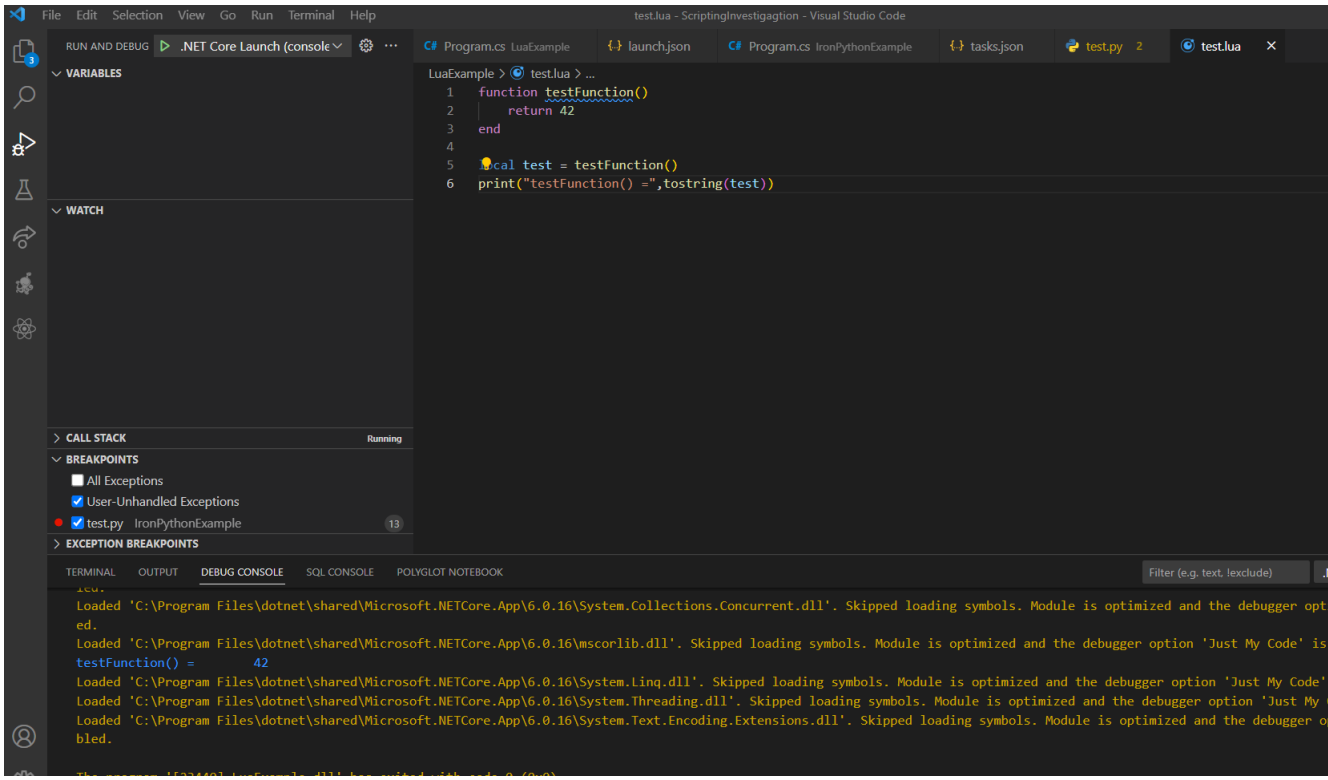

Abbildung 1: Lua-Konsolenausgabe

- Die Beweise für das Debugging mit IronPython:

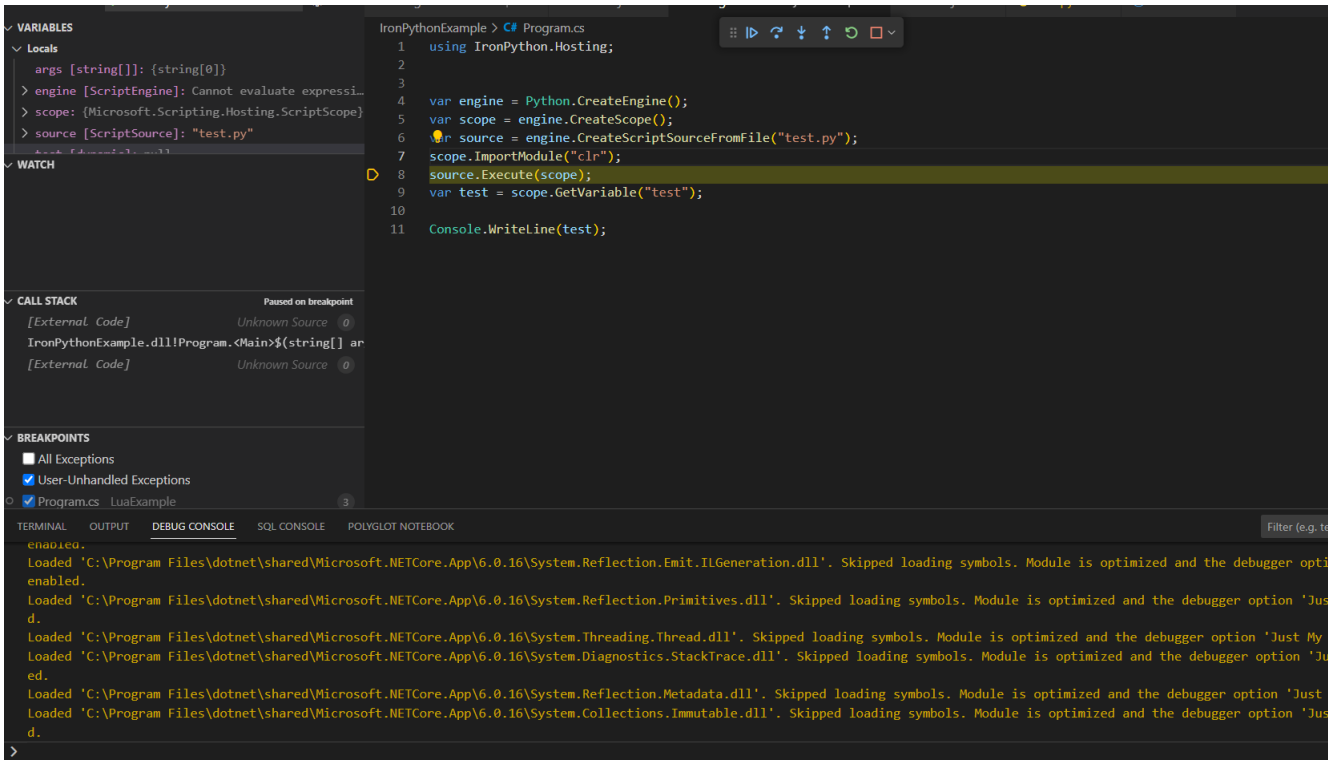

Abbildung 2: IronPython-VSCode-Breakpoint1

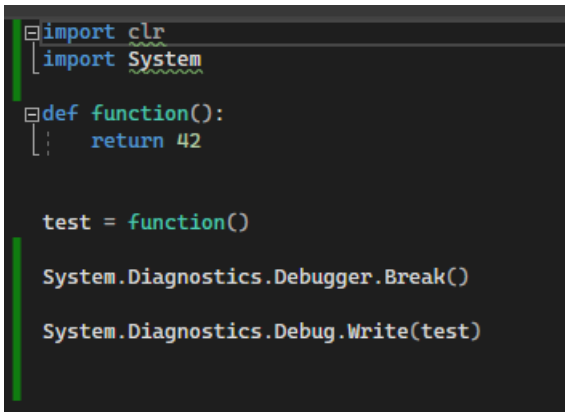

Abbildung 3: IronPython-VSBreakpoint2

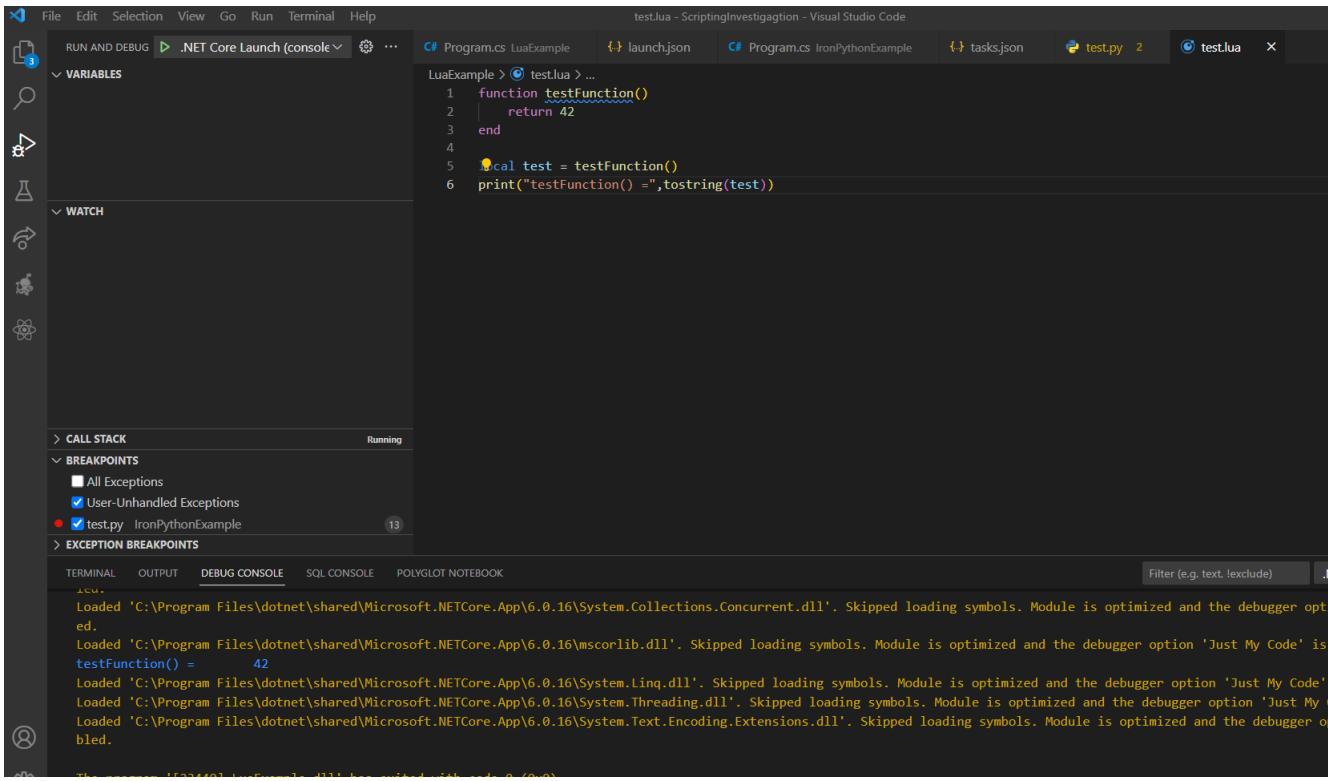

Abbildung 4: IronPython-Konsolenausgabe

3 Anwendung (Praxisteil)

3.1 Verwendete Technologien

ASP.NET Core

Unsere Webanwendung wurde unter Verwendung der ASP.NET Core-Plattform entwickelt. ASP.NET Core ist ein vielseitiges, plattformübergreifendes und leistungsfähiges Open-Source-Framework, das zur Entwicklung moderner, internetfähigen Anwendungen geeignet ist. Die Ausführung findet in der .NET Core-Laufzeitumgebung statt. ASP.NET Core bietet außerdem eine moderne und flexible Umgebung für die Entwicklung von Webanwendungen, die sowohl plattformübergreifend als auch hochgradig skalierbar sind. [18]

Mit ASP.NET Core kann man:

- Webanwendungen und Webdienste, Internet-der-Dinge (IoT)-Anwendungen und mobile Backends entwickeln.
- Auf verschiedene Betriebssysteme wie Windows, macOS und Linux arbeiten.
- Anwendungen sowohl in der Cloud als auch auf lokalen Systemen bereitstellen.

Dieses Framework eröffnet somit eine breite Palette an Möglichkeiten für Entwickler, um moderne Anwendungen zu erstellen, die sich nahtlos mit dem Internet verbinden und sowohl in Cloud- als auch lokalen Umgebungen effizient betrieben werden können. [18]

Git

Unsere Versionskontrolle haben wir mit Git gemacht. Git ist ein Versionskontrollsystem mit verteiltem Ansatz, das entworfen wurde, um sowohl kleine als auch äußerst umfangreiche Projekte auf schnelle und effiziente Weise zu verwalten. Die Erlernbarkeit von Git gestaltet sich einfach, und seine geringe Systembelastung geht einher mit herausragender Performance. Es setzt sich von anderen Versionskontrollsystemen wie Subversion, CVS, Perforce und ClearCase ab, indem es Funktionen wie kosteneffiziente lokale „Branches“, bequeme Staging-Bereiche und vielfältige Arbeitsabläufe bietet. Git erlaubt und begünstigt die Erstellung von mehreren unabhängigen lokalen Verzweigungen. Die Prozesse des Erstellens, Zusammenführens und Entfernens dieser Entwicklungsstränge nehmen lediglich Sekunden in Anspruch. [19]

Git ist kostenfrei und Open-Source. Es wurde dazu entwickelt, Projekte aller Größenordnungen, schnell und effizient zu verwalten. Es zeichnet sich als Open-Source aus, da es die Anpassungsfähigkeit bietet, den Quellcode nach den individuellen Bedürfnissen der Nutzer/innen anzupassen. Mit seiner Open-Source-Natur ermöglicht Git mehreren Personen gleichzeitig an einem Projekt zu arbeiten und ermöglicht eine äußerst einfache und effiziente Zusammenarbeit. Aus diesem Grund wird Git als das herausragende Versionskontrollsystem betrachtet, das in der heutigen Zeit zur Verfügung steht. [20]

Docker

Docker ist eine Plattform, welche Anwendungen gemeinsam mit ihren spezifischen Abhängigkeiten in Form von Containern bündelt. Dieser Ansatz gewährleistet, dass die Anwendung in jeder beliebigen Entwicklungsumgebung reibungslos funktioniert. Jede einzelne Anwendung läuft in separaten Containern und verfügt über ihre eigenen Satz an Abhängigkeiten und Bibliotheken. Dies gewährleistet, dass jede Applikation in völliger Unabhängigkeit von anderen Anwendungen agiert und die Sicherheit gibt, dass sie Anwendungen erstellen können, welche sich nicht gegenseitig beeinträchtigen. Wir haben in unserer Anwendung unser Frontend, Backend und die Datenbank über Docker laufen lassen. Das heißt wir haben 3 Docker-Container benutzt. [21]

Unter einem Container versteht man eine standardisierte Softwareeinheit, die sowohl den Programmcode als auch sämtliche damit verbundenen Abhängigkeiten zusammenfasst. Hierdurch wird sichergestellt, dass die Anwendung zuverlässig in unterschiedlichen Computerumgebungen ausgeführt werden kann. Ein Docker-Container-Image verkörpert eine autonome und ausführbare Softwareeinheit, welche sämtliche Komponenten für die Ausführung einer Applikation in sich trägt: den Code selbst, die Laufzeitumgebung, Systemwerkzeuge, Systembibliotheken und Konfigurationseinstellungen. [22]

Container-Images verwandeln sich zur Laufzeit in eigenständige Container. Im Fall von Docker geschieht dies durch das Ausführen der Images auf der Docker Engine. Containerisierte Software steht sowohl für Linux- als auch für Windows-basierte Anwendungen zur Verfügung und gewährleistet eine gleichbleibende Ausführung, unabhängig von der genutzten Infrastruktur. Container schaffen eine Isolierung der Software von ihrer Umgebung und gewährleisten somit, dass die Anwendung konsistent arbeitet, selbst bei Unterschieden zwischen Entwicklungs- und Staging-Umgebungen. [22]

PostgreSQL

Als Datenbanksystem haben wir uns für PostgreSQL entschieden, da wir bereits in anderen Projekten mit diesem gearbeitet haben. PostgreSQL ist Open-Source und verwendet die SQL-Sprache. Außerdem ist PostgreSQL auf allen gängigen Betriebssystemen kompatibel. PostgreSQL läuft bei uns über Docker, somit haben wir nichts Zusätzliches dafür installieren müssen. PostgreSQL ist sehr anpassbar, man kann beispielsweise eigene Datentypen definieren und individuelle Funktionen gestalten.
[23]

Datentypen in PostgreSQL:

- Zahlen und Zeichen: Integer, Numeric, String, Boolean
- Strukturierte: Date/Time, Array, Range/Multirange
- Geometrisch: Point, Line, Circle, Polygon
- Anpassbare: Composite, Custom Types

Integrität der Daten:

- UNIQUE, NOT NULL
- Primary Keys
- Foreign Keys
- Exclusion Constraints
- Explicit Locks, Advisory Locks

[23]

BenchmarkDotNet (0.13.2)

BenchmarkDotNet unterstützt Anwender/innen dabei, die Performance ihrer Methoden in Benchmark-Tests zu überprüfen. Ebenso ermöglicht es den Austausch von reproduzierbaren Messexperimenten. Diese Transformation gestaltet sich genauso unkompliziert wie die Erstellung von Unit-Tests. BenchmarkDotNet hilft dabei, übliche Fehler im Benchmarking-Prozess zu vermeiden und Nutzer zu informieren, sobald Unstimmigkeiten im Benchmark-Design oder den erfassten Messdaten auftreten. Die präsentierten Resultate erscheinen in einer nutzerfreundlichen Tabelle, die sämtliche relevanten Aspekte des Experiments herausstellt.
Anbei ein Beispiel von einem unserer BenchmarkDotNet-Tests: [24]

Listing 7: BenchmarkDotNet

```
namespace C_SharpExample
{
    [MarkdownExporter,
        HtmlExporter,
        SimpleJob(RunStrategy.ColdStart, launchCount: 1, warmupCount: 5,
            targetCount: 5, id: "FastAndDirtyJob")]
    public class C_SharpTesting
    {
        [Benchmark]
        public void TestC_Sharp_Simple() => ReturnNumber();

        [Benchmark]
        public void TestC_Sharp_Sum() => MySum();

        #region C_SharpFunctions
        public static async void ReturnNumber()
        {
            var state = await CSharpScript.RunAsync("return 42;");
            Console.WriteLine(state.ReturnValue);
        }
        public static async void MySum()
        {
            var state = await CSharpScript.RunAsync("return 3 + 3;");
            Console.WriteLine(state.ReturnValue);
        }
        #endregion
    }
}
```

Bogus

Bogus haben wir in unserem Projekt für die Generierung von Fake Daten benutzt. Wir haben damit Schüler- und Lehrernamen erstellen lassen die wir in unserer Anwendung als Testdaten benutzt haben. Bogus funktioniert ausschließlich für .NET-Sprachen wie C#, F# oder VB.NET. [25]

Es ist ganz unkompliziert zu verwenden. Wir haben in unserer Arbeit nur Namen generieren lassen, jedoch könnte man zu jedem Namen noch eine ganze Menge hinzufügen wie zum Beispiel Telefonnummern, E-Mail-Adressen, Wohnadressen oder auch die Herkunft.

Folgendes Codebeispiel zeigt die Generierung unserer Fake Daten für eine Schulklasse:

Listing 8: Bogus

```
List<Student> firstStudents = new List<Student>();

#region Create Fake Students for each Schoolclass
for (int i = 0; i < 10; i++)
{
    var studentFaker = new Faker<Student>()
        .RuleFor(x => x.Name, x => x.Person.FullName)
        .Generate();
    firstStudents.Add(studentFaker);
}
```

In der RuleFor() Methode kann man genau die Sachen angeben die man benötigt. In unserem Fall haben wir nur Vornamen und Nachnamen benötigt.

3.1.1 Entity Framework

Entity Framework ist ein Object-Relational Mapping (ORM) Framework für .NET-Anwendungen, das von Microsoft entwickelt wurde. Es ermöglicht Entwicklern, mit relationalen Daten in einer objektorientierten Weise zu arbeiten, ohne sich direkt mit der Datenbankkommunikation beschäftigen zu müssen. Das Framework bietet Funktionen für Datenzugriff, Modellierung und Abfragen und unterstützt eine Vielzahl von Datenbank-Engines.

Entity Framework gibt es in verschiedenen Versionen:

- Entity Framework 6 (EF6):
  - Eine ausgereifte Version, die vor allem in .NET Framework-Anwendungen eingesetzt wird.
- Entity Framework Core (EF Core)
  - Eine leichtere und erweiterbare Version, die für .NET Core entwickelt wurde, aber auch in .NET Framework und .NET 5+ verwendet werden kann.

Mit Entity Framework können Entwickler:

- Datenmodelle aus Datenbanken generieren (Database-First)
- Datenbanken aus Datenmodellen generieren (Code-First)
- Abfragen mit LINQ-to-Entities durchführen
- Beziehungen zwischen Tabellen als Objektbeziehungen abbilden
- Datenänderungen in der Datenbank über das DataContext-Objekt verfolgen und speichern

In dieser Untersuchung wurde Entity Framework Core verwendet.

3.1.2 AutoMapper

AutoMapper ist eine Open-Source-Bibliothek in .NET, die das Mapping von einem Objekttyp auf einen anderen vereinfacht. Es automatisiert den Prozess der Datenübertragung zwischen Objekten, meist DTOs (Data Transfer Objects), und Domain-Modellen. Dies spart Entwicklern die Mühe, manuellen Code für die Zuordnung von Feldern zwischen verschiedenen Objektklassen zu schreiben. AutoMapper verwendet Konventionen, um automatisch zu erkennen, wie die Zuordnung zwischen den Feldern verschiedener Objekte erfolgen sollte. Entwickler können jedoch auch spezielle Regeln und Transformationen konfigurieren, wenn komplexe Zuordnungen erforderlich sind.

Folgende Code Ausschnitte zeigen diesen Vorgang:

Listing 9: DTO

```
1     /// <summary>
2     /// Dtos for Grades
3     /// </summary>
4     namespace Shared.Dtos
5     {
6         public class GradeGetDto
7         {
8             public int Id { get; set; }
9             public string GradeKind { get; set;} = string.Empty;
10            public int Graduate { get; set; }
11            public int SubjectId { get; set; }
12            public int TeacherId { get; set; }
13            public int StudentId { get; set; }
14            public string? Note { get; set; } = string.Empty;
15            public string StudentName { get; set; } = string.Empty;
16        }
17
18        public class GradePostDto
19        {
20            public string GradeKind { get; set;} = string.Empty;
21            public int SubjectId { get; set; }
22            public int Graduate { get; set; }
23            public int TeacherId { get; set; }
24            public int StudentId { get; set; }
25            public string? Note { get; set; } = string.Empty;
26        }
27    }
```

Listing 10: Definitation for Mapping

```
1       /// <summary>
2       /// Defines the Mapping for Grades and GradeDtos
3       /// </summary>
4       public class GradePofile : Profile
5       {
6           public GradePofile()
7           {
8               /// <summary>
9               /// Map from DbGradKind to GradeKindGetDto
10              /// </summary>
11              /// <typeparam name="GradeKind"></typeparam>
12              /// <typeparam name="GradeKindGetDto"></typeparam>
13              /// <returns></returns>
14              CreateMap<GradeKind, GradeKindGetDto>().DisableCtorValidation()
15              .ForMember(
16                  des => des.Id,
17                  opt => opt.MapFrom(src => src.Id)
18
19              )
20              .ForMember(
21                  des => des.Name,
22                  opt => opt.MapFrom(src => src.Name)
23              );
24              /// <summary>
25              /// DbGrade to GradeGetDto
26              /// </summary>
27              /// <typeparam name="Grade"></typeparam>
28              /// <typeparam name="GradeGetDto"></typeparam>
29              /// <returns></returns>
30              CreateMap<Grade, GradeGetDto>()
31              .ForPath(
32                  dest => dest.StudentName,
33                  opt => opt.MapFrom(src => src.Student!.Name)
34              )
35              .ForPath(
36                  dest => dest.GradeKind,
37                  opt => opt.MapFrom(src => src.GradeKind.Name)
38              );
39
40
41              /// <summary>
42              /// GradePostDto to Grade
43              /// </summary>
44              /// <typeparam name="GradePostDto"></typeparam>
45              /// <typeparam name="Grade"></typeparam>
46              /// <returns></returns>
47              CreateMap<GradePostDto, Grade>().DisableCtorValidation()
48              .ForMember(
49                  dest => dest.Id,
50                  opt => opt.MapFrom(src => 0)
51              )
52              .ForPath(
53                  dest => dest.GradeKind.Name,
54                  opt => opt.MapFrom(src => src.GradeKind )
55              );
56
57          }
58      }
```

Listing 11: HttpGetMethod

```
1       [HttpGet("/gradesForClass/{schoolClassId}")]
2       [ProducesResponseType(StatusCodes.Status200OK)]
3       [ProducesResponseType(StatusCodes.Status400BadRequest)]
4       /// <summary>
5       /// Get all grades from backend
6       /// </summary>
7       /// <returns>IEnumberable<GradeGetDto> result </returns>
8       public async Task<ActionResult<IEnumerable<GradeGetDto>>>
            GetGradesForSchoolClassAsync(int schoolClassId)
9       {
10          var grades = await DbContext.Grades
11          .Include(g => g.Student)
12          .Include(g => g.GradeKind)
13          .Where(g => g.Student!.SchoolClassId == schoolClassId)
14          .ToListAsync();
15
16          var result = grades.Select(grade =>
                _mapper.Map<GradeGetDto>(grade)).ToList();
17
18          return Ok(result);
19      }
```

3.1.3 Blazor

Blazor ist ein Web-Framework von Microsoft, das Entwicklern ermöglicht, interaktive Web-Apps mit Csharp anstelle von JavaScript zu bauen. Es ist Teil des .NET-Ökosystems und verwendet WebAssembly, um Csharp-Code im Browser auszuführen. Blazor bietet zwei Hosting-Modelle: Blazor WebAssembly und Blazor Server.

- Blazor Server: Führt Csharp-Code auf dem Server aus. Interaktionen mit der Benutzeroberfläche werden über SignalR-Kommunikation zwischen Client und Server abgewickelt. Dies hat eine schnellere Anfangsladezeit, erfordert jedoch eine ständige Verbindung zum Server.
- Blazor WebAssembly: Läuft Csharp-Code direkt im Browser mithilfe von WebAssembly. Dies ermöglicht eine schnellere Interaktion, da keine Serverkommunikation erforderlich ist, erhöht jedoch die Anfangsladezeit.

Blazor ermöglicht die Wiederverwendung von Code und Bibliotheken, erleichtert das Testen und ist in Visual Studio und andere Entwicklungsumgebungen integriert.

3.2 Aufbau

Um einen Überblick über die Beispielanwendung zu erhalten folgt nun ein Komponentendiagramm:

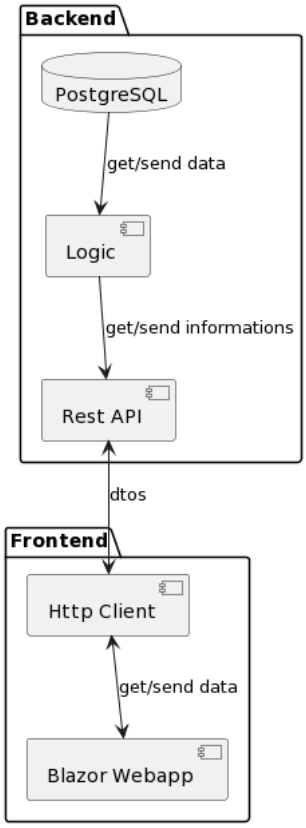

Abbildung 5: Komponenten – UML Diagramm

Die Schnittstelle zur Kommunikation auf der Seite des Frontends ist in Services pro Entität implementiert:

Listing 12: Services für Kommunikation

```
1       private readonly HttpClient _http;
2       public GradeService(HttpClient http)
3       {
4           _http = http;
5       }
6
7       public List<GradeKeyGetDto> GradeKeys { get; set; } = new
            List<GradeKeyGetDto>();
8       public List<SchoolClassGetDto> Schooclasses { get; set;} = new
            List<SchoolClassGetDto>();
9       public List<GradeGetDto> Grades { get; set; } = new List<GradeGetDto>();
10      public List<GradeKindGetDto> Kinds { get; set; } = new List<GradeKindGetDto>();
11
12      public async Task CreateGradeAsync(Grade grade)
13      {
14          var result = await _http.PostAsJsonAsync("/grades", grade);
15          await SetGradesAsync(result);
16      }
```

Die API ist in Controllern pro Entität aufgebaut:

Listing 13: API - Controller

```
1       private readonly GradeCalculator _gradeCalculator;
2       private readonly IMapper _mapper;
3       private IConfiguration Config { get; }
4       private ApplicationDbContext DbContext { get; }
5       //private readonly ILogger<StudentsController> _logger;
6       public GradeController(IConfiguration config, ApplicationDbContext dbContext,
            IMapper mapper, GradeCalculator gradeCalculator)
7       {
8           this.Config = config;
9           this.DbContext = dbContext;
10          _mapper = mapper;
11          _gradeCalculator = gradeCalculator;
12      }
```

Damit der Aufbau der .NET-Solution noch klarer wird ist nun die YAML-Datei dargestellt:

```yaml
version: '3.8'
services:
  grades_db:
    image: postgres
    ports:
      - 5432:5432
    environment:
      POSTGRES_PASSWORD: postgres
      POSTGRES_USER: postgres
      POSTGRES_DB: GradeDb
    healthcheck:
          test: ["CMD-SHELL", "sh -c 'pg_isready -U postgres -d
              GradeDb'"]
          interval: 10s
          timeout: 3s
          retries: 55

  grades_backend:
    container_name: ${DOCKER_REGISTRY-}grades_backend
    image: grades_backend
    privileged: true
    ports:
      - 5000:80
    environment:
      - ASPNETCORE_ENVIRONMENT=Development
      -
          ConnectionStrings__DefaultConnection=UserID=postgres;Password=postgres;
          Security=true;Pooling=true;",

    build:
      context: .
      dockerfile: ./API/Dockerfile
    depends_on:
      grades_db:
        condition: service_healthy

  grades_web:
    image: ${DOCKER_REGISTRY-}grades_web
    container_name: grades_web
    volumes:
    - /Client:/Client
    ports:
      - 8080:80
    environment:
      - ASPNETCORE_ENVIRONMENT=Development

    build:
       context: .
       dockerfile: ./Client/Dockerfile
    depends_on:
      - grades_backend
```

Die verwendeten Entitäten und ihre Relationen im Backend sind in der folgenden Abbildung dargestellt.

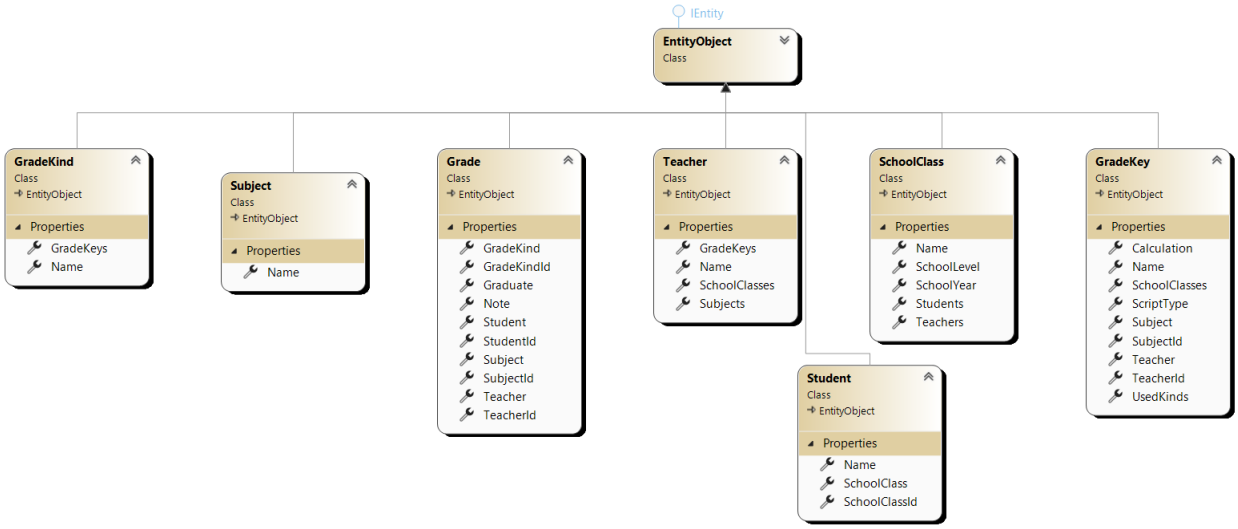

Abbildung 6: Entitäten – UML Diagramm

Es gibt viele Möglichkeiten Skripte in eine Anwendung zu importieren. Eine Möglichkeit ist die Skripte als Datei zu importieren. Anstatt alle benötigten Daten direkt in den Code einzubetten oder sie manuell über die Befehlszeile einzugeben, können Entwickler*innen eine oder mehrere Dateien als Input verwenden, die das Skript dann liest und verarbeitet.

In unseren Fall werden die Datei in Blazor eingelesen. Um das einlesen noch effizienter zu gestalten ist ein "Drag and Drop" für Dateien implementiert:

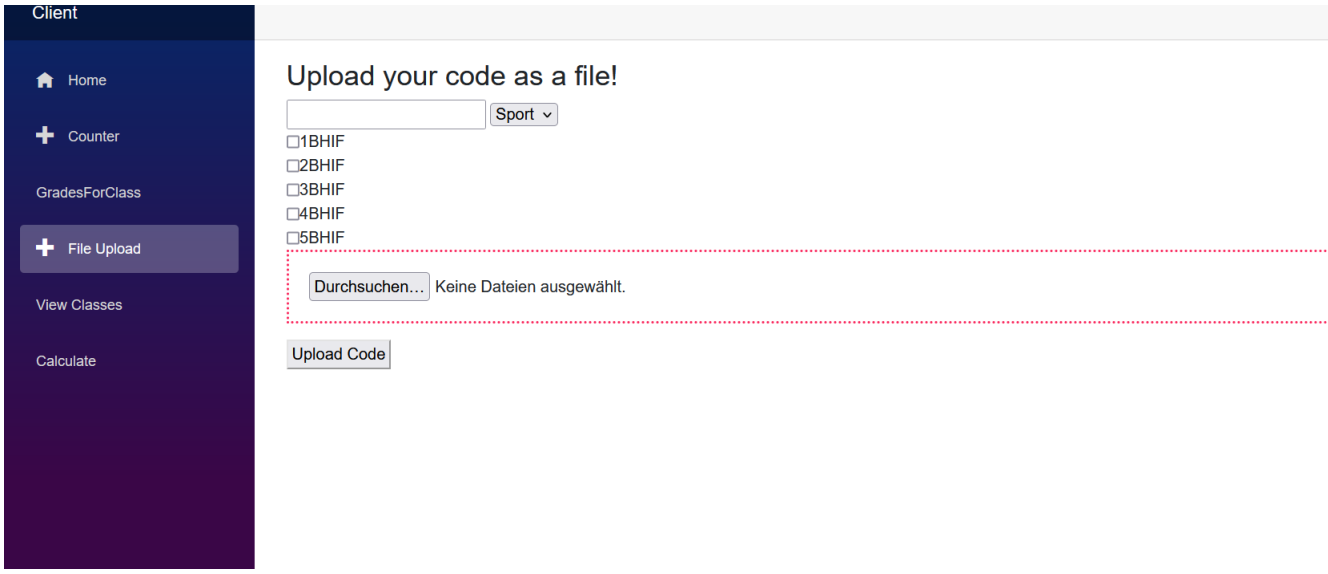

Abbildung 7: File Upload

Es folgt nun der Code:

Listing 14: Code for FileUpload

```
@page "/fileupload"

@using Client.Services;
@using global::Shared.Entities;
@using global::Shared.Dtos;

@implements IAsyncDisposable;
@inject IJSRuntime JSRuntime;
@inject ISubjectService SubjectService;
@inject IGradeService GradeService;
@inject NavigationManager NavigationManager;

<h3>Upload your code as a file!</h3>

<EditForm Model="@gradeKeyPost">
    <InputText id="name" @bind-Value="gradeKeyPost.Name" />

    <InputSelect id="subject" @bind-Value="gradeKeyPost.SubjectId">
        @foreach (var item in @SubjectService.Subjects)
        {
            <option value="@item.Id"> @item.Name </option>
        }
    </InputSelect>

    <CheckBoxList Data="@GradeService.Schooclasses"
        TextField="@((item)=>item.Name)" ValueField="@((item)=>item.Name)"
        SelectedValues="@SelectedSchoolClasses" />

```

```
28          <div @ref="fileDropContainer" class="file-drop-zone @HoverClass"
                @ondragenter="OnDragEnter"
29              @ondragleave="OnDragLeave" @ondragover="OnDragEnter">
30              <InputFile OnChange="@OnChange" multiple />
31          </div>
32
33          <div class="error-message-container">
34              <p>@ErrorMessage</p>
35          </div>
36
37          <div>
38              <button type="submit" @onclick="AddGradeKeyAsync">
39                  Upload Code
40              </button>
41          </div>
42
43      </EditForm>
44
45
46
47      @code
48      {
49          protected List<string> SelectedSchoolClasses = new List<string>();
50          private string HoverClass = string.Empty;
51          void OnDragEnter(DragEventArgs e) => HoverClass = "hover";
52          void OnDragLeave(DragEventArgs e) => HoverClass = string.Empty;
53          IJSObjectReference? _filePasteModule = default(IJSObjectReference);
54          IJSObjectReference? _filePasteFunctionReference =
                default(IJSObjectReference);
55          ElementReference fileDropContainer;
56          IBrowserFile? file;
57          ScriptType scriptType;
58          string stringCode = string.Empty;
59          private string ErrorMessage = string.Empty;
60          private const int maxAllowedFiles = 1;
61          private int _userId = 1;
62          private GradeKeyPostDto gradeKeyPost = new();
63          protected override async Task OnInitializedAsync()
64          {
65              try
66              {
67                  await SubjectService.GetSubjectsByTeacherAsync(_userId);
68                  await
                        GradeService.GetSchoolclassesByTeacherAndSubjectAsync(_userId,
                        SubjectService.Subjects.First().Id);
69                  await GradeService.GetAllKindsAsync();
70                  gradeKeyPost.SubjectId = SubjectService.Subjects.First().Id;
71              }
72
73              catch (Exception)
74              {
75                  throw;
76              }
77          }
78          private async Task AddGradeKeyAsync()
79          {
80              if (stringCode == string.Empty)
81              {
82                  ErrorMessage = "Script is needed!";
83              }
84              var sc = GradeService.Schooclasses.SingleOrDefault(s => s.Name ==
                    SelectedSchoolClasses.First());
85
86              var kinds = GradeService.Kinds.Select(k => k.Name).ToList();
87              gradeKeyPost.Calculation = stringCode;
88              gradeKeyPost.SchoolClasses = SelectedSchoolClasses;
89              gradeKeyPost.ScriptType = scriptType;
90              gradeKeyPost.UsedKinds = kinds;
91              gradeKeyPost.TeacherId = _userId;
92
93              await GradeService.CreateGradeKeyAsync(gradeKeyPost);
94
95              //await GradeService.CalcGradesForClass(sc!.Id, _userId);
96
97              NavigationManager.NavigateTo($"/calculation");
98          }
99
100         public async ValueTask DisposeAsync()
101         {
102             if (_filePasteFunctionReference != null)
103             {
104                 await _filePasteFunctionReference.InvokeVoidAsync("dispose");
```

```
105                 await _filePasteFunctionReference.DisposeAsync();
106             }
107             if (_filePasteModule != null)
108             {
109                 await _filePasteModule.DisposeAsync();
110             }
111         }
112 
113         protected async override Task OnAfterRenderAsync(bool firstRender)
114         {
115             if (firstRender)
116             {
117                 _filePasteModule = await
                        JSRuntime.InvokeAsync<IJSObjectReference>("import",
                        "./js/filePaste.js");
118                 _filePasteFunctionReference = await
                        _filePasteModule.InvokeAsync<IJSObjectReference>("initializeFilePaste",
119                 fileDropContainer, file);
120             }
121         }
122 
123         async Task OnChange(InputFileChangeEventArgs e)
124         {
125             stringCode = string.Empty;
126             ErrorMessage = string.Empty;
127 
128             using var content = new MultipartFormDataContent();
129 
130             if (e.FileCount > maxAllowedFiles)
131             {
132                 ErrorMessage = $"Only {maxAllowedFiles} files can be uploaded";
133                 return;
134             }
135 
136 
137             foreach (var file in e.GetMultipleFiles())
138             {
139                 var isAllowed = false;
140                 var fileContent = new StreamContent(file.OpenReadStream());
141                 var info = new FileInfo(file.Name);
142 
143                 switch (info.Extension)
144                 {
145                     case ".lua":
146                         scriptType = ScriptType.Lua;
147                         isAllowed = true;
148                         break;
149                     case ".py":
150                         scriptType = ScriptType.Python;
151                         isAllowed = true;
152                         break;
153                     case ".csc":
154                         scriptType = ScriptType.CSharpScript;
155                         isAllowed = true;
156                         break;
157                     case ".js":
158                         isAllowed = true;
159                         scriptType = ScriptType.JavaScript;
160                         break;
161                     default:
162                         ErrorMessage = "Content Type not sopported";
163                         break;
164                 }
165 
166                 if (isAllowed)
167                 {
168                     stringCode = await fileContent.ReadAsStringAsync();
169                 }
170             }
171         }
172     }
```

Nach dem ein Skript hochgeladen wurde, wird man automatisch auf die Kalkulations-Seite geleitet:

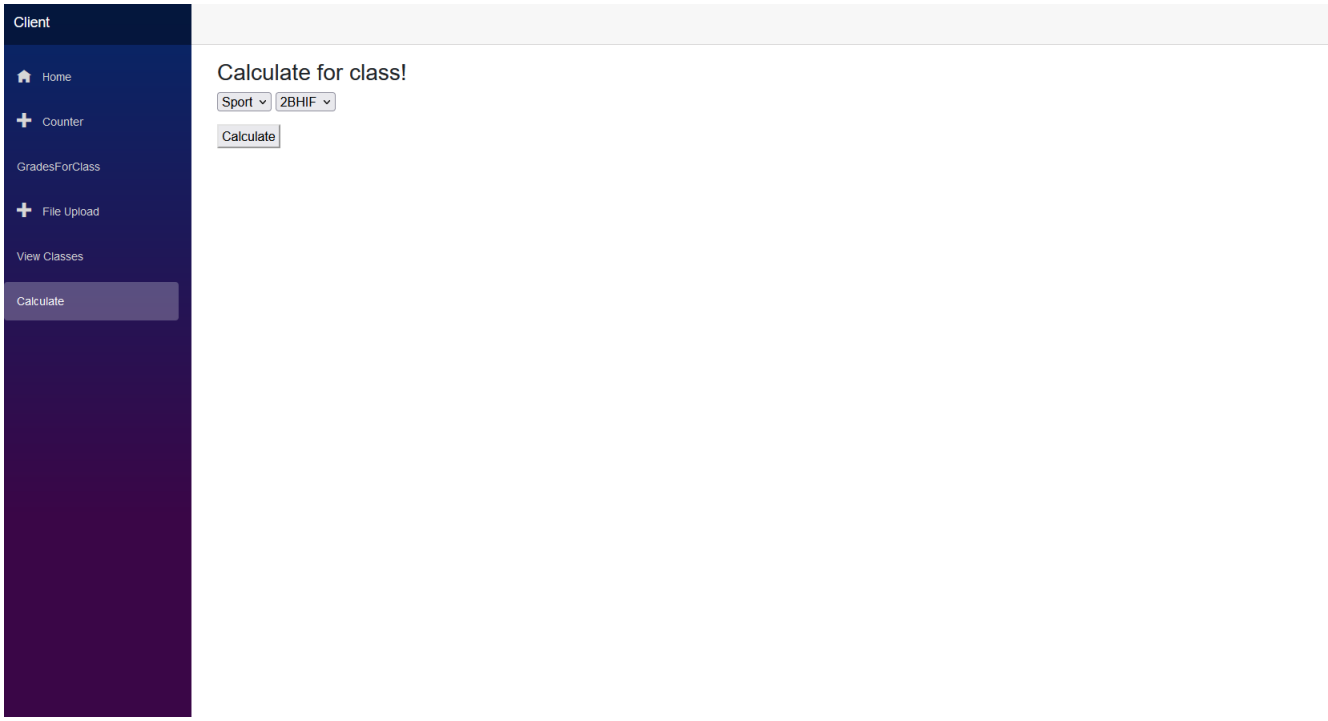

Abbildung 8: Calculation – UI

Wenn eine Berrechnung erfolgt, werden im GradeCalculator alle benötigen Daten von der Datenbank geladen und weiter zur eigentlichen Berrechnung gegeben:

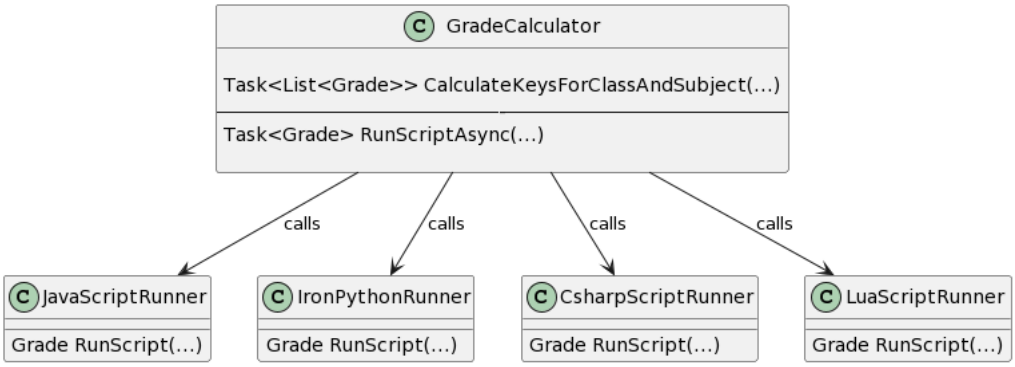

Abbildung 9: UI– FileUpload

Es folgt nun der Code für den GradeCalculator:

Listing 15: Code for loading data for Calculation

```
1       public async Task<List<Grade>> CalculateKeysForClassAndSubject(int
            schoolClassId, int subject)
2       {
3               var grades = await context.Grades
4                   .Include(g => g.Subject)
5                   .Include(g => g.Student)
6                   .Where(g => g.SubjectId == subject && g.Student!.SchoolClassId ==
                        schoolClassId)
7                   .ToListAsync();
8
9               if (grades == null || grades.Count() == 0)
10              {
11                  return new List<Grade>();
12              }
13
14              var schoolClass = await context.SchoolClasses.SingleAsync(sc => sc.Id
                    == schoolClassId);
15
16              var g1 = grades.First();
17              var key = await context.GradeKeys
18                  .SingleOrDefaultAsync(k => k.SubjectId == subject && k.TeacherId
                        == g1.TeacherId && k.SchoolClasses.Contains(schoolClass));
19
20              if(key == null)
21              {
22                  throw new NullReferenceException("No Key found");
23              }
24
25              var result = new List<Grade>();
26
27              foreach (var item in grades.DistinctBy(g => g.StudentId).Select(g =>
                    g.StudentId))
28              {
29                  var student = context.Students.Single(s => s.Id == item);
30                  var gradesForCalc =  grades.Where(g => g.StudentId ==
                        student.Id).ToList();
31                  // Run Script for one student
32                  var gradeForStudent = await this.RunScriptAsync(key,
33                                                      gradesForCalc,
34                                                      student);
35                  gradeForStudent.TeacherId = key.TeacherId;
36                  gradeForStudent.StudentId = item;
37                  gradeForStudent.SubjectId = key.SubjectId;
38                  gradeForStudent.GradeKindId = 1;
39                  result.Add(gradeForStudent);
40              }
41
42              return result;
43          }
44
45
46          /// Handels the script type
47          private async Task<Grade> RunScriptAsync(GradeKey gradeKey, List<Grade>
                grades, Student student)
48          {
49              var result = new Grade();
50              result.Student = student;
51              switch (gradeKey.ScriptType)
52              {
53                  case ScriptType.None:
54                      break;
55                  case ScriptType.Lua:
56                      result = luaScriptRunner.RunScript(gradeKey, grades);
57                      break;
58                  case ScriptType.Python:
59                      result = pythonScriptRunner.RunScript(gradeKey, grades);
60                      break;
61                  case ScriptType.JavaScript:
62                      result = jsRunner.RunScript(gradeKey, grades);
63                      break;
64                  case ScriptType.CSharpScript:
65                      result = await csScriptRunner.RunScriptAsync(gradeKey, grades);
66                      break;
67                  default:
68                      result = null;
69                      break;
70              }
```

```
71
72              if (result == null)
73              {
74                  throw new Exception("Erorr in Calculation");
75              }
76
77              result.Student = student;
78
79              result.Note = $"{student.Name} : {DateTime.Now}";
80
81              return result;
82          }
```

Im folgenden Abschnitt werden einige Code-Ausschnitte betrachtet, die zeigen, wie man solche Datei-Übergaben in den untersuchten Scriptsprachen realisieren kann.

Listing 16: Code for IronPython

```
1     /// <summary>
2     /// Runs Python-scripts
3     /// </summary>
4     public class PythonScriptRunner
5     {
6         private readonly ScriptEngine engine;
7
8         public PythonScriptRunner()
9         {
10            this.engine = Python.CreateEngine();
11        }
12        public Grade RunScript(GradeKey key, List<Grade> grades)
13        {
14            if (key.Calculation == string.Empty || key.UsedKinds == null || grades
                  == null)
15            {
16                throw new NullReferenceException("Not enough information for
                      Calculation");
17            }
18
19            var code = key.Calculation;
20
21            var result = new Grade();
22            try
23            {
24                var scope = engine.CreateScope();
25                engine.GetClrModule();
26
27                var parseGrades = grades.ToArray()!;
28                scope.SetVariable("grades", parseGrades);
29                var source = engine.CreateScriptSourceFromString(code);
30                source.Execute();
31                var graduate = scope.GetVariable("graduate");
32                result.Teacher = key.Teacher;
33                result = Convert.ToInt32(graduate);
34            }
35            catch (Exception)
36            {
37                throw;
38            }
39
40            return result;
41        }
42    }
```

Als Beispiel Skript für IronPython kann folgender Code genommen werden:

Listing 17: IronPython-Code

```
def calculate():
    makGradesCount = 0
    makGrade = 0
    testGrade = 0
    testCount = 0
    hwCount = 0
    hwGrade = 0

    for grade in grades:

        if grade.GradeKind.Name == 'MAK':
            makGrade = makGrade + grade.Graduate
            makGradesCount = makGradesCount + 1

        elif grade.GradeKind.Name == 'TEST' :
            testCount = testCount + 1
            testGrade = testGrade + grade.Graduate

        elif grade.GradeKind.Name == 'HOMEWORK' :
            hwCount = hwCount + 1
            hwGrade = hwGrade + grade.Graduate

    mak = makGrade / makGradesCount
    test =  testGrade / testCount
    hw = hwGrade / hwCount


    result = (mak + test + hw) / 3

    return result


graduate = calculate()
```

Wenn es sich um einen Javascript-Code handelt wird folgende Methode aufgerufen:

Listing 18: Code for Javascript

```
1   public class JavascriptRunner
2   {
3       public Grade RunScript(GradeKey key, List<Grade> grades)
4       {
5           var engine = new JintJsEngine();
6           Grade result = new Grade();
7           List<string>? logs = new List<string>();
8
9           try
10          {
11              // Definiere eine Variable im JavaScript-Code, um die
                    console.log-Ausgaben zu speichern
12              engine.Execute("var consoleOutput = [];");
13
14              // Definiere die console.log-Funktion im JavaScript-Code
15              engine.Execute(@"
16                              var console = {
17                                  log: function() {
18                                      consoleOutput.push(Array.from(arguments).join('
                                            '));
19                                  }
20                              };
21                          ");
22
23
24              var gradeKindsList = JsonConvert.SerializeObject(key.UsedKinds);
25              var gradesList = JsonConvert.SerializeObject(grades);
26
27              engine.SetVariableValue("gradeKindsList", gradeKindsList);
28              engine.SetVariableValue("gradesList", gradesList);
29
30              if (key.Calculation != null)
31              {
32                  engine.Execute(key.Calculation);
33              }
34
35              //Die Ausgabe der console.log-Anweisungen als JSON-String
36              string jsonOutput =
                    engine.Evaluate<string>("JSON.stringify(consoleOutput)");
37
38              // Konvertiere den JSON-String in eine Liste von strings
39              if (jsonOutput != null)
40              {
41                  logs = JsonConvert.DeserializeObject<List<string>>(jsonOutput);
42
43                  if (logs != null)
44                  {
45                      DisplayOutput(logs);
46                  }
47              }
48
49              // Get Return from Script
50              var resultGrade = engine.GetVariableValue("result");
51
52              result.Teacher = key.Teacher;
53              result.Graduate = Convert.ToInt32(resultGrade);
54          }
55          catch (Exception)
56          {
57              result.Teacher = null;
58              result.Graduate = 0;
59          }
60          return result;
61      }
62
63      private static void DisplayOutput(List<string> logs)
64      {
65          foreach (string output in logs)
66          {
67              Debug.WriteLine(output);
68          }
69      }
70  }
```

Als Beispiel hierfür ein Skript:

Listing 19: Javascript-Skript

```
var grades = JSON.parse(gradesList);
var gradeKinds = JSON.parse(gradeKindsList);

function calculate() {
    let makCounter = 0;
    let testCounter = 0;
    let homeworkCounter = 0;

    let mak = 0;
    let test = 0;
    let homework = 0;

    for (let i = 0; i < grades.length; i++) {

        if (grades[i].GradeKind.Name == 'MAK') {
            mak += grades[i].Graduate;
            makCounter++;
        }
        if (grades[i].GradeKind.Name == 'TEST') {
            test += grades[i].Graduate;
            testCounter++;
        }
        if (grades[i].GradeKind.Name == 'HOMEWORK') {
            homework += grades[i].Graduate;
            homeworkCounter++;
        }
    }
    return ((mak / makCounter) + (test / testCounter) + (homework /
        homeworkCounter)) / 3;
}
var result = calculate();
console.log('Wert: ' +grades[1].GradeKind.Name);
```

Bei einem Lua-Skript wird folgende Methode aufgerufen:

Listing 20: Code for NLua

```
1       /// <summary>
2       /// Runs lua-scripts
3       /// </summary>
4       public class LuaScriptRunner
5       {
6           private readonly Lua state;
7
8           public LuaScriptRunner()
9           {
10              this.state = new Lua();
11          }
12
13          public Grade RunScript(GradeKey key, List<Grade> grades)
14          {
15              if (key.Calculation == string.Empty || key.UsedKinds == null || grades
                    == null)
16              {
17                  throw new NullReferenceException("Not enough information for
                        Calculation");
18              }
19
20              var code = key.Calculation;
21
22              var result = new Grade();
23              try
24              {
25                  state.DoString(code);
26                  state.LoadCLRPackage();
27                  state["grades"] = grades;
28                  state.DoString(@"graduate = calculate()");
29                  result.Teacher = key.Teacher;
30                  var gr = state["graduate"];
31                  if (gr != null)
32                  {
33                      result.Graduate = Convert.ToInt32(gr);
34                  }
35              }
36              catch (Exception)
37              {
38                  throw;
39              }
40
41              return result;
42          }
43      }
```

Hierfür kann folgendes Skript verwendet werden:

Listing 21: Lua Example

```
function calculate()

    local makGradesCount = 0
    local makGrade = 0
    local testGrade = 0
    local testCount = 0
    local hwCount = 0
    local hwGrade = 0

    for countGrade = 0, grades.Count - 1 do
        if grades[countGrade].GradeKind.Name == 'MAK' then
            makGrade = makGrade + grades[countGrade].Graduate
            makGradesCount = makGradesCount + 1
        elseif grades[countGrade].GradeKind.Name == 'TEST' then
            testCount = testCount + 1
            testGrade = testGrade + grades[countGrade].Graduate
        elseif grades[countGrade].GradeKind.Name == 'HOMEWORK' then
            hwCount = hwCount + 1
            hwGrade = hwGrade + grades[countGrade].Graduate
        end
    end

    local result = ((makGrade / makGradesCount) + ( testGrade / testCount ) +
        (hwGrade / hwCount)) / 3

    return math.floor(result)
end
```

Für ein Csharpscript wird folgender Code ausgeführt:

Listing 22: Code for CsharpScripting

```
1   public class CsScriptMicrosoftRunner
2   {
3       public async Task<Grade> RunScriptAsync(GradeKey key, List<Grade> grades)
4       {
5           var result = new Grade();
6           var globals = new Globals { GradeKey = key, Grades = grades };
7           var options = ScriptOptions.Default
8               .WithEmitDebugInformation(true);
9
10          try
11          {
12              var state = await CSharpScript.RunAsync(key.Calculation, options,
                    globals: globals);
13              WriteAllVariablesFromScript(state.Variables);
14
15              double resultGrade = 0.0;
16              foreach (var variable in state.Variables)
17              {
18                  if (variable.Name == "result")
19                  {
20                      resultGrade = (double)variable.Value;
21                  }
22              }
23
24              result.Teacher = key.Teacher;
25              result.Graduate = Convert.ToInt32(resultGrade);
26
27          }
28          catch (Exception)
29          {
30              result.Teacher = null;
31              result.Graduate = 0;
32          }
33
34          return result;
35      }
36
37      private static void
            WriteAllVariablesFromScript(ImmutableArray<ScriptVariable> variables)
38      {
39          foreach (var variable in variables)
40          {
41              Debug.WriteLine(variable.Name + ": " + variable.Value);
42          }
43      }
44
45      public class Globals
46      {
47          public GradeKey? GradeKey;
48          public List<Grade>? Grades;
49      }
50  }
```

Für ein Csharpscript wird folgender Code ausgeführt:

Listing 23: Csharpscript

```
    int makCounter = 0;
int testCounter = 0;
int homeworkCounter = 0;

int mak = 0;
int test = 0;
int homework = 0;

foreach (var item in Grades)
{
    if (item.GradeKind.Name == "MAK")
    {
        mak += item.Graduate;
        makCounter++;
    }
    if (item.GradeKind.Name == "TEST")
    {
        test += item.Graduate;
        testCounter++;
    }
    if (item.GradeKind.Name == "HOMEWORK")
    {
        homework += item.Graduate;
        homeworkCounter++;
    }
}

double result = ((mak / makCounter) + (test / testCounter) + (homework /
    homeworkCounter)) / 3.0;
```

Nach der Berrechnung werden die Jahresnoten in einer Tabelle dargestellt:

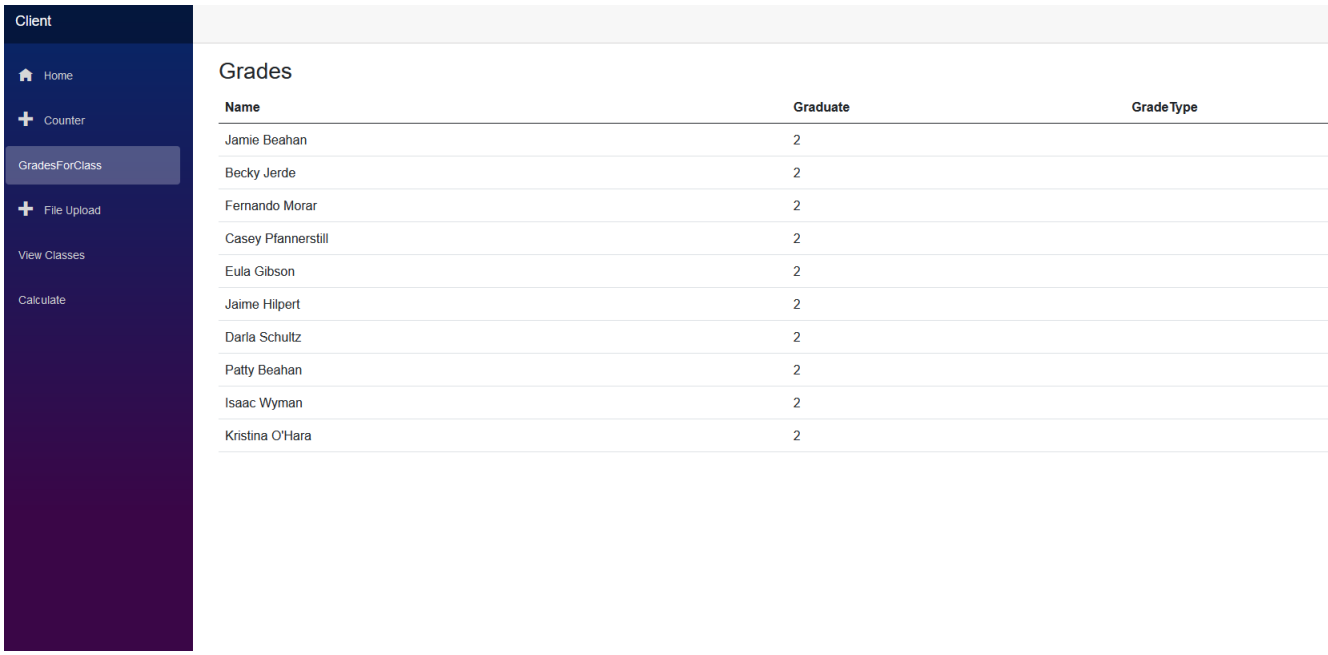

Abbildung 10: UI – Jahesnoten

3.3 Unit-Tests zur Überprüfung der Notenberechnung

Eine der zentralen Komponenten unserer Beispielanwendung war die Implementierung von Skripten zur Notenberechnung in .NET-Anwendungen zur Laufzeit. Um die Zuverlässigkeit und Genauigkeit dieser Skripte sicherzustellen, haben wir einen Satz von Unit-Tests entwickelt und durchgeführt. Dieser Ansatz gewährleistet nicht nur die Integrität des Codes, sondern bietet auch eine robuste Basis für zukünftige Erweiterungen und Anpassungen.

Auswahl der Testfälle

Die Testfälle wurden so ausgewählt, um ein breites Spektrum an Szenarien abzudecken, die in realen Anwendungen vorkommen könnten. Dazu gehören Standardfälle, Grenzfälle und auch potenzielle Fehlerzustände. Das hat uns ermöglicht, die Robustheit für unsere Test-Scripts umfassend zu überprüfen.

Ergebnisse der Unit-Tests

Alle entwickelten Skripte zur Notenberechnung haben die Tests letztenendes erfolgreich bestanden. Dies gab uns ein hohes Maß an Vertrauen in die Funktionsfähigkeit und Zuverlässigkeit der implementierten Lösungen. Darüber hinaus haben die Tests dazu beigetragen, einige nicht offensichtliche Fehler und Unklarheiten im ursprünglichen Design zu identifizieren, die wir entsprechend beheben konnten.

Testbeispiel

In folgendem Testbeispiel haben wir ein C#-Skript getestet:

Listing 24: Test for CsharpScripting

```
1  [TestMethod]
2      public void CsScript_T02()
3      {
4          List<Grade> grades = new List<Grade>
5          {
6              new Grade { GradeKind = gradeKinds.Single(g => g.Name =="MAK") ,
                   Graduate = 1 },
7              new Grade { GradeKind = gradeKinds.Single(g => g.Name =="MAK") ,
                   Graduate = 1 },
8              new Grade { GradeKind = gradeKinds.Single(g => g.Name =="MAK") ,
                   Graduate = 1 },
9              new Grade { GradeKind = gradeKinds.Single(g => g.Name =="MAK") ,
                   Graduate = 2 },
10             new Grade { GradeKind = gradeKinds.Single(g => g.Name =="MAK") ,
                   Graduate = 1 },
11             new Grade { GradeKind = gradeKinds.Single(g => g.Name == "TEST"),
                   Graduate = 1},
12             new Grade { GradeKind = gradeKinds.Single(g => g.Name == "TEST"),
                   Graduate = 2},
13             new Grade { GradeKind = gradeKinds.Single(g => g.Name == "TEST"),
                   Graduate = 3},
14             new Grade { GradeKind = gradeKinds.Single(g => g.Name == "TEST"),
                   Graduate = 4},
15             new Grade { GradeKind = gradeKinds.Single(g => g.Name == "TEST"),
                   Graduate = 2},
16             new Grade { GradeKind = gradeKinds.Single(g => g.Name == "HOMEWORK"),
                   Graduate = 1},
17             new Grade { GradeKind = gradeKinds.Single(g => g.Name == "HOMEWORK"),
                   Graduate = 2},
18             new Grade { GradeKind = gradeKinds.Single(g => g.Name == "HOMEWORK"),
                   Graduate = 3},
19             new Grade { GradeKind = gradeKinds.Single(g => g.Name == "HOMEWORK"),
                   Graduate = 4},
20             new Grade { GradeKind = gradeKinds.Single(g => g.Name == "HOMEWORK"),
                   Graduate = 5},
21
22         };
23
24         var code = File.ReadAllText("test.cs");
25         var key = new GradeKey { Name = "CsScriptTest", UsedKinds = gradeKinds,
               Calculation = code };
26
27         var result = CsScriptRunner.RunScript(key, grades);
28
29         Assert.AreEqual(2, result.Graduate, "Calculation is right");
```

Dabei beinhaltet die Liste "grades" Schulnoten von drei verschiedenen Typen:

- MAK (Mitarbeitskontrolle)
- Test
- Homework

Test-Skript

Dieses Test-Skript haben wir geschrieben um die Funktionalitäten und Notenberechnungen zu testen. Einfachheitshalber wird in diesem Skript jeder Notentyp äquivalent gerechnet.

Listing 25: CsharpScript-Testscript

```
int makCounter = 0;
int testCounter = 0;
int homeworkCounter = 0;

int mak = 0;
int test = 0;
int homework = 0;

foreach (var item in Grades)
{
    if (item.GradeKind.Name == "MAK")
    {
        mak += item.Graduate;
        makCounter++;
    }
    if (item.GradeKind.Name == "TEST")
    {
        test += item.Graduate;
        testCounter++;
    }
    if (item.GradeKind.Name == "HOMEWORK")
    {
        homework += item.Graduate;
        homeworkCounter++;
    }
}

return ((mak / makCounter) + (test / testCounter) + (homework / homeworkCounter))
    / 3.0;
```

Das Skript unterscheidet zwar zwischen allen Notentypen, wertet jedoch alle gleich und berechnet den Durchschnitt.

4 Technologien

5 Umsetzung

Siehe tolle Daten in Tab. 1.

Siehe und staune in Abb. 11. Suspendisse vel felis. Ut lorem lorem, interdum eu, tincidunt sit amet, laoreet vitae, arcu. Aenean faucibus pede eu ante. Praesent enim elit, rutrum at, molestie non, nonummy vel, nisl. Ut lectus eros, malesuada sit amet, fermentum eu, sodales cursus, magna. Donec eu purus. Quisque vehicula, urna sed ultricies auctor, pede lorem egestas dui, et convallis elit erat sed nulla. Donec luctus. Curabitur et nunc. Aliquam dolor odio, commodo pretium, ultricies non, pharetra in, velit. Integer arcu est, nonummy in, fermentum faucibus, egestas vel, odio.

Sed commodo posuere pede. Mauris ut est. Ut quis purus. Sed ac odio. Sed vehicula hendrerit sem. Duis non odio. Morbi ut dui. Sed accumsan risus eget odio. In hac habitasse platea dictumst. Pellentesque non elit. Fusce sed justo eu urna porta tincidunt. Mauris felis odio, sollicitudin sed, volutpat a, ornare ac, erat. Morbi quis dolor. Donec pellentesque, erat ac sagittis semper, nunc dui lobortis purus, quis congue purus metus ultricies tellus. Proin et quam. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent sapien turpis, fermentum vel, eleifend faucibus, vehicula eu, lacus.

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Donec odio elit, dictum in, hendrerit sit amet, egestas sed, leo. Praesent feugiat sapien aliquet odio. Integer vitae justo. Aliquam vestibulum fringilla lorem. Sed neque lectus, consectetuer at, consectetuer sed, eleifend ac, lectus. Nulla facilisi. Pellentesque eget lectus. Proin eu metus. Sed porttitor. In hac habitasse platea dictumst. Suspendisse eu lectus. Ut mi mi, lacinia sit amet, placerat et, mollis vitae, dui. Sed ante tellus, tristique ut, iaculis eu, malesuada ac, dui. Mauris nibh leo, facilisis non, adipiscing quis, ultrices a, dui.

	Regular Customers	**Random Customers**
Age	20-40	>60
Education	university	high school

Tabelle 1: Ein paar tabellarische Daten

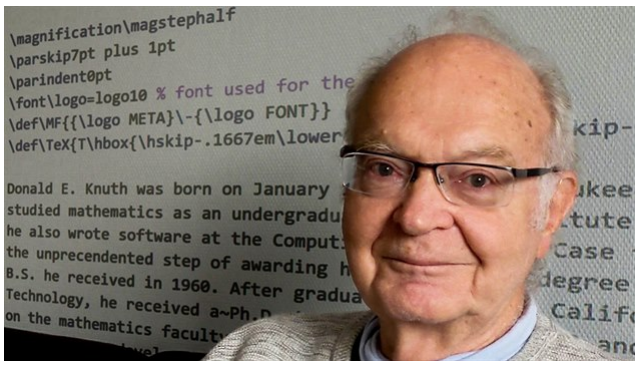

Abbildung 11: Don Knuth – CS Allfather

Morbi luctus, wisi viverra faucibus pretium, nibh est placerat odio, nec commodo wisi enim eget quam. Quisque libero justo, consectetuer a, feugiat vitae, porttitor eu, libero. Suspendisse sed mauris vitae elit sollicitudin malesuada. Maecenas ultricies eros sit amet ante. Ut venenatis velit. Maecenas sed mi eget dui varius euismod. Phasellus aliquet volutpat odio. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Pellentesque sit amet pede ac sem eleifend consectetuer. Nullam elementum, urna vel imperdiet sodales, elit ipsum pharetra ligula, ac pretium ante justo a nulla. Curabitur tristique arcu eu metus. Vestibulum lectus. Proin mauris. Proin eu nunc eu urna hendrerit faucibus. Aliquam auctor, pede consequat laoreet varius, eros tellus scelerisque quam, pellentesque hendrerit ipsum dolor sed augue. Nulla nec lacus. Dann betrachte den Code in Listing 8.

6 Zusammenfassung

6.1 Schlussfolgerungen

Diese Diplomarbeit hat sich mit der Möglichkeit von Scripting in .NET-Anwendungen zur Laufzeit auseinandergesetzt. Ziel der Arbeit war es, die Möglichkeiten und Grenzen dieser Technologie sowohl aus Entwickler/in- als auch aus als Benutzer/in zu untersuchen.

Entwickler/innen können die Erkenntnisse nutzen, um anpassungsfähigere .NET-Anwendungen zu bauen. Zum Beispiel können sie Scripting einsetzen, um fehlende Funktionen von Anwendungen zu erstellen, ohne viele Codeänderungen zu machen. Die Untersuchung des Scripting in .NET-Anwendungen zur Laufzeit hat eine Reihe von wichtigen Erkenntnissen geliefert. Diese können dazu beitragen, die theoretischen Grundlagen in diesem Bereich zu erweitern und gleichzeitig praxisorientierte Lösungen für die Softwareentwicklung und -sicherheit zu bieten. Die Arbeit legt somit einen wichtigen Grundstein für weiterführende Untersuchungen und Entwicklungen in diesem Bereich.

Während das Scripting in .NET-Anwendungen zur Laufzeit eine leistungsstarke Funktion für die dynamische Modifikation und Anpassung darstellt, hat es einen wesentlichen Nachteil: das Debugging des zur Laufzeit integrierten Skripts ist nicht möglich. Diese Limitation stellt eine Herausforderung für Entwickler/innen dar. Das bedeutet, dass zwar von der Flexibilität des Scripting profitiert werden kann, jedoch Schwierigkeiten Fehler im Code effizient zu identifizieren und zu beheben. Unsere Alternative für das Debugging ist die Konsolen-Ausgabe. Obwohl diese Methode weniger umfassend ist als Debugging-Tools, ermöglicht sie dennoch eine gewisse Überwachung und Fehleridentifikation in Echtzeit. Sie erlaubt es Entwickler/innen, wichtige Informationen, Zustände oder Fehlermeldungen direkt in der Konsole auszugeben, um so das Verhalten des Skripts zur Laufzeit besser nachvollziehen zu können.

6.2 Herausforderungen und Probleme

Im Verlauf unserer Diplomarbeit über Scripting in .NET-Anwendungen zur Laufzeit sind wir auf mehrere Herausforderungen gestoßen, die unsere Arbeit zunächst erschwert haben, uns aber letztlich zu wichtigen Erkenntnissen geführt haben.

Verständnis der Problemstellung

Einer der ersten Stolpersteine war das grundlegende Verständnis der Problemstellung. Scripting in .NET-Anwendungen zur Laufzeit ist ein komplexes Thema, das sowohl technisches als auch konzeptionelles Verständnis erfordert. Es dauerte einige Zeit, bis wir die Kernprobleme und -fragen unserer Forschung vollständig erfassen konnten.

Overengineering

Eine unserer größten Herausforderungen bestand darin, dass wir die Anwendung in der Anfangsphase übermäßig komplex gestaltet hatten. Dieses Over Engineering führte zu unerwarteten Problemen und machte eine Korrektur notwendig. Nach einer kritischen Überprüfung unseres Ansatzes entschieden wir, mehrere Komponenten zu entfernen, um die Anwendung zu vereinfachen und den Fokus auf die Kernthemen zu legen.

Implementierung der Konsolenausgabe

Die Konsolenausgabe erschien zunächst als einfache Lösung für unsere Debugging-Anforderungen. Allerdings stellte die tatsächliche Implementierung eine größere Herausforderung dar als erwartet, insbesondere in Bezug auf die Synchronisierung der Konsolenausgabe mit den zur Laufzeit generierten Skripten.

Fazit

Diese anfänglichen Herausforderungen haben unseren Lernprozess und unsere methodische Herangehensweise wesentlich geprägt. Jedes dieser Probleme wurde überwunden, aber die dabei gesammelten Erfahrungen haben wesentlich zu unserer fachlichen Entwicklung beigetragen und werden zweifellos von Nutzen sein, wenn wir uns zukünftigen Forschungsprojekten stellen.

6.3 Kritische Betrachtung der Ergebnisse

Während unsere Diplomarbeit wichtige Einblicke in die Möglichkeiten und Herausforderungen des Scripting in .NET-Anwendungen zur Laufzeit liefert, muss eine kritische Betrachtung unserer Ergebnisse einige Einschränkungen berücksichtigen. Erstens ist unser Forschungsumfeld auf eine begrenzte Anzahl von Skriptsprachen und Anwendungsbeispielen beschränkt, was die Allgemeingültigkeit der Erkenntnisse einschränken könnte. Zweitens basiert unsere alternative Debugging-Methode über die Konsolenausgabe auf einer vereinfachten Annahme des Systemverhaltens, die in komplexeren oder sicherheitskritischen Anwendungen nicht ausreichend sein könnte.

6.4 Mögliche weitere Untersuchungsthemen

Die Untersuchung von Scripting in .NET-Anwendungen zur Laufzeit stellt nur die Spitze des Eisbergs dar, wenn es um die Komplexität und Vielfältigkeit des Themenfelds geht. Die Ergebnisse der vorliegenden Arbeit legen zahlreiche Ansatzpunkte für zukünftige Forschungsprojekte nahe.

Sicherheitsaspekte

Zukünftige Studien könnten spezifische Angriffsszenarien und ihre Abwehrmöglichkeiten analysieren. Besonders die sich ständig weiterentwickelnde Landschaft von Sicherheitsbedrohungen stellt einen fruchtbaren Boden für weiterführende Untersuchungen dar.

Performance-Optimierung

Ein weiteres interessantes Forschungsfeld könnte die Performance-Optimierung von .NET-Anwendungen sein, die intensiv Scripting zur Laufzeit nutzen. Hier könnte untersucht werden, wie sich verschiedene Scripting-Techniken auf die Laufzeitleistung der Anwendung auswirken und wie sich diese Performance am besten optimieren lässt.

Erweiterte Debugging-Methoden

Angesichts der Schwierigkeiten beim Debugging zur Laufzeit wäre es lohnend, innovative Methoden oder Tools für diese spezifische Herausforderung zu entwickeln und zu evaluieren. Wie können Entwickler und Sicherheitsexperten noch effektiver das Verhalten von Skripten in Echtzeit nachvollziehen?

Literaturverzeichnis

[1] Microsoft, „Office-Script," Geöffnet am 26.08.2023. Online verfügbar: https://support.microsoft.com/en-au/office/introduction-to-office-scripts-in-excel-9fbe283d-adb8-4f13-a75b-a81c6baf163a

[2] ——, „Office-Script-Example," Geöffnet am 26.08.2023. Online verfügbar: https://learn.microsoft.com/en-us/office/dev/scripts/resources/samples/samples-overview

[3] ——, „Power-Automate," Geöffnet am 26.08.2023. Online verfügbar: https://learn.microsoft.com/de-de/power-automate/frequently-asked-questions

[4] Franchise-Net, „Modulsystem," Geöffnet am 27.08.2023. Online verfügbar: https://www.franchise-net.de/lexikon/modulsystem/

[5] AWS, „Microservices," Geöffnet am 27.08.2023. Online verfügbar: https://aws.amazon.com/de/microservices/#:~:text=Microservices%20sind%20ein%20architekturbezogener%20und,Services%20geh%C3%B6ren%20kleinen%2C%20eigenst%C3%A4ndigen%20Teams

[6] Microsoft, „Assemblies," Geöffnet am 27.08.2023. Online verfügbar: https://learn.microsoft.com/de-de/dotnet/framework/app-domains/how-to-load-assemblies-into-an-application-domain

[7] Weikio, „Pluginframework," Geöffnet am 28.08.2023. Online verfügbar: https://github.com/weikio/PluginFramework

[8] A. Varanese, *Game Scripting Mastery.* Course Technology PTR, 2002.

[9] L. Developerteam, „Lua - Documentation," Geöffnet am 05.09.2023. Online verfügbar: https://www.lua.org/docs.html

[10] R. Ierusalimschy, *Programmieren in Lua.* Open Source Press, 2001.

[11] Nlua, „Nlua," Geöffnet am 05.09.2023. Online verfügbar: http://nlua.org/

[12] IronPython, „IronPython," Geöffnet am 05.09.2023. Online verfügbar: http://ironpython.net/

[13] ——, „IronPythonGithub," Geöffnet am 05.09.2023. Online verfügbar: https://github.com/IronLanguages/ironpython3

[14] D. Viehland, *IronPython in Action.* Manning Publications, 2009.

[15] M. Michaelis, „Csharp Scripting," Geöffnet am 05.09.2023. Online verfügbar: https://learn.microsoft.com/en-us/archive/msdn-magazine/2016/january/essential-net-csharp-scripting

[16] M. W. Docs, „MDN Web Docs," Geöffnet am 05.09.2023. Online verfügbar: https://developer.mozilla.org/en-US/

[17] OWASP, „OWASP," Geöffnet am 05.09.2023. Online verfügbar: https://owasp.org/

[18] Microsoft, „ASP.NET Core," Geöffnet am 25.08.2023. Online verfügbar: https://learn.microsoft.com/de-de/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-7.0

[19] Git, „git," Geöffnet am 25.08.2023. Online verfügbar: https://git-scm.com/

[20] Abhinav96, „git-features," Geöffnet am 25.08.2023. Online verfügbar: https://www.geeksforgeeks.org/git-features/

[21] S. Kappagantula, „docker-explained," Geöffnet am 26.08.2023. Online verfügbar: https://www.edureka.co/blog/docker-explained/

[22] Docker, „docker-container," Geöffnet am 26.08.2023. Online verfügbar: https://www.docker.com/resources/what-container/

[23] Postgresql, „Postgresql," Geöffnet am 26.08.2023. Online verfügbar: https://www.postgresql.org/about/

[24] A. Akinshin, „BenchmarkDotNet," Geöffnet am 26.08.2023. Online verfügbar: https://github.com/dotnet/BenchmarkDotNet

[25] B. Chavez, „Bogus," Geöffnet am 26.08.2023. Online verfügbar: https://github.com/bchavez/Bogus

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

Anhang