Fakultät Informatik

CYBRAIL

IT-Projekt im Studiengang Bachelor Informatik

vorgelegt von

Mattis Krämer, Robin Rosner, Pascal Blank

Erstellungssemester SS2024 - WS2425

Betreuer: Dr. Martin Geier

©2024

Dieses Werk einschließlich seiner Teile ist **urheberrechtlich geschützt**. Jede Verwertung außerhalb der engen Grenzen des Urheberrechtgesetzes ist ohne Zustimmung des Autors unzulässig und strafbar. Das gilt insbesondere für Vervielfältigungen, Übersetzungen, Mikroverfilmungen sowie die Einspeicherung und Verarbeitung in elektronischen Systemen.

Inhaltsverzeichnis

1 Einleitung

Im Rahmen dieses IT-Projekts wurde das System Cyber Barrier for Reliable Academic Integrity and Log-analysis (CYBRAIL) entwickelt, um die Integrität und Transparenz akademischer Prüfungen signifikant zu verbessern.
Mit der fortschreitenden Digitalisierung von Prüfungsprozessen gewinnt der Einsatz von Tools wie dem Safe Exam Browser (SEB) zunehmend an Bedeutung, da sie gewährleisten, dass Prüfungen unter fairen Bedingungen durchgeführt werden und unerlaubte Hilfsmittel ausgeschlossen bleiben. Allerdings stellt die Auswertung der durch den SEB generierten Protokolle (Logs) bislang eine erhebliche Herausforderung dar, insbesondere wenn es darum geht, potenziell verdächtige Aktivitäten in Echtzeit oder im Nachhinein zu identifizieren und adäquat darauf zu reagieren.

1.1 Safe Exam Browser

Der SEB ist ein abgesicherter Browser, der speziell für Prüfungen an Schulen entwickelt wurde. Seine Funktionalität ist stark eingeschränkt, um die den Studierenden zur Verfügung stehenden Hilfsmittel erheblich zu beschränken. Der SEB wird auch an unserer Hochschule eingesetzt, weshalb der Fokus dieses Projekts auf diesen Browser ausgerichtet ist. Zu den Funktionen des SEB gehören unter anderem Prüfungen, ob unerlaubte Programme im Hintergrund laufen, ob der Bildschirm geteilt wird, und es wird verhindert, dass Studierende die SEB-Umgebung verlassen, andere Programme öffnen oder Copy-Paste verwenden. Somit bildet der SEB eine solide Grundlage, um faire Prüfungen zu ermöglichen und Betrug zu verhindern.

1.2 Cheating with SEB

Jedoch hat jedes System auch gewisse Schwächen.

1.2.1 Virtuelle Machine

Frühere Versionen des SEB konnten durch den einfachen Austausch einer Programmdaten so verändert werden, dass der Browser in einer Virtuellen Maschine (VM) gestartet werden konnte. Dies wird eigentlich durch mehrere komplexe Prüfungen unterbunden, die den Start in einer VM verhindern sollten. Diese Manipulation konnte jedoch mit einer etwa fünfminütigen Google-Suche leicht und unkompliziert durchgeführt werden. Dadurch ist es möglich, den SEB in einem Fenster oder auf einem zweiten Bildschirm auszuführen, während auf dem Hostsystem – also dem System, das die VM betreibt – nach Lösungen gegoogelt, Videoanrufe durchgeführt oder Aufzeichnungen gemacht werden können.

Dennoch gab es bereits in diesen Fällen gewisse Indizien in diversen Logdateien, die auf einen Missbrauch hinwiesen. Beispielsweise wird geloggt, wenn sich die Bildschirmauflösung ändert, was passiert, wenn die Fenstergröße der VM angepasst wird oder der Bildschirm in den Vollbildmodus wechselt. Der SEB verfügt zudem über eine Integritätsprüfung, um Modifikationen festzustellen.

Beide dieser Indizien werden jedoch lediglich in Logdateien geschrieben. Ohne eine Auswertung und Sammlung der Logdateien erfährt der Prüfer oder die Aufsichtsperson nichts von solchen Vorfällen.

1.2.2 Automatisches Tippen

Ein weiterer Angriffsvektor ist nahezu unentdeckbar: das automatische Eintippen von Zeichen anstelle des normalen Copy-Pastes. Dies könnte entweder durch auf dem Gerät laufende Software geschehen, die nicht durch den SEB blockiert wird – hier würde nur eine Whitelist helfen, da ein solches Programm beliebig benannt werden kann – oder durch einen Hardware-Dongle, der beispielsweise Text von einem anderen Gerät automatisch eintippt. So könnten Lösungen auf einem zweiten Gerät gesucht oder unter den Studierenden geteilt und anschließend schnell und bequem automatisch eingetippt werden.

1.2.3 SEB Server

Alle Informationen, die in Logdateien gespeichert werden, liegen zunächst auf dem Client des Nutzers und sind somit unzugänglich für die unmittelbare Erkennung von auffälligem Verhalten. SEB bietet jedoch die Möglichkeit, einen optionalen SEB-Server einzusetzen. Dieser kann an der Hochschule aufgesetzt werden und nach korrekter Konfiguration des Servers sowie der Konfigurationsdatei für den Client – welche vor Beginn der Klausur direkt von Moodle geladen werden kann – Logdateien vom Client empfangen. Dies stellt jedoch die gesamte Funktionalität des Servers dar; es findet keine automatische Auswertung der Logs statt, was bedeutet, dass ein Mensch versucht – vermutlich stichprobenartig – Auffälligkeiten zu finden. Dies ist jedoch sehr schwierig, es sei denn, man ist mit der Funktionsweise des SEB vertraut.

1.3 Lösung

Eine offensichtliche Lösung wäre ein Tool, das diesen Prozess möglichst generisch und erweiterbar automatisiert: CYBRAIL – Cyber Barrier for Reliable Academic Integrity and Log-analysis. Das zentrale Ziel von CYBRAIL ist es, diesen Mangel durch die Automatisierung des Prüfungsprozesses zu beheben und den verantwortlichen Professoren eine effiziente, möglichst in Echtzeit ablaufende Kontrolle und Auswertung der SEB-Logs zu ermöglichen. Dies umfasst die Entwicklung von Algorithmen zur automatisierten Analyse und Klassifizierung der Log-Daten, um Verdachtsfälle unerlaubter Mittel schnell und präzise zu erkennen und gleichzeitig entsprechende Beweismittel bereitzustellen. Ein weiteres wichtiges Merkmal von CYBRAIL ist die Bereitstellung einer modularen Systemarchitektur mit flexiblen Schnittstellen, die es ermöglicht, weitere Prüfmechanismen bei Bedarf problemlos zu integrieren.

Durch diese Lösung sollen sowohl die Zuverlässigkeit und Transparenz des Prüfungsprozesses als auch die Nachvollziehbarkeit der Prüfungsergebnisse erheblich gesteigert werden.

2 Projektziele

Wir als Team haben uns mehrere Ziele für dieses Projekt gesetzt, um ein solches Tool zu entwickeln.

2.1 Log- und Informationssammlung

Unser erstes Ziel war es herauszufinden, inwiefern wir Zugriff auf die gespeicherten Logs erhalten können, wie diese übertragen werden und in welchem Format sie vorliegen. Darüber hinaus haben wir untersucht, wie diese Informationen von einem Computer aufbereitet werden können, um sie anschließend zu analysieren.

2.2 Demonstration der Erkennung von Auffälligkeiten

Ein weiteres wichtiges Ziel war es, herauszufinden, welche Betrugsversuche überhaupt erkannt werden können und welche davon relevant sind. Ebenso war es von Bedeutung zu klären, welche Informationen für die Erkennung benötigt werden. Darüber hinaus mussten wir identifizieren, welche Betrugsversuche vermutlich nicht erkannt werden können und welche zusätzlichen Informationen oder Logs dafür erforderlich wären.

Letztendlich galt es, zu bewerten, auf welche Betrugserkennungsmethoden wir uns im Rahmen dieses Projekts fokussieren wollen, um diese zu demonstrieren.

2.3 Benutzerfreundlichkeit und einfache Bereitstellung

Das letzte Ziel des Projekts war es, eine möglichst einfache Benutzeroberfläche sowie eine unkomplizierte Einrichtung zu ermöglichen. Unser Projekt sollte schnell und einfach in der Praxis einsetzbar sein, sei es auf einem Server oder einem PC, ohne dass ein komplexes Setup erforderlich ist.

3 Projektverlauf und Meilensteine

Dieses Kapitel beschreibt die einzelnen Entwicklungsschritte und die dadurch erreichten Meilensteine unserer Sprints.

3.1 Anforderungs- und Risikoanalyse

Im ersten Sprint haben wir uns mit der Definition der Anforderungen (Requirements) beschäftigt. Ziel war es, festzulegen, welche Aspekte unser Projekt abdecken soll und welche Eigenschaften möglicherweise unrealistisch oder optional sind, um einen klaren Fokus zu setzen. Eine detailliertere Beschreibung der Anforderungen findet sich in Kapitel 2.

Gleichzeitig mit der Festlegung der Anforderungen haben wir uns ebenfalls mit den daraus resultierenden Risiken auseinandergesetzt. Hierbei stand die Frage im Vordergrund, welche Fehlschläge am wahrscheinlichsten und gravierendsten sein könnten. Besonders besorgniserregend waren die Risiken in Bezug auf die Beschaffung der Logdateien und den Datenschutz. Eine genauere Beschreibung der Risikobewertung findet sich ebenfalls in Kapitel 5.

3.2 Programmentwurf und Architekturentwurf

Unser zweiter großer Meilenstein war die Ausarbeitung eines möglichen Programmentwurfs, also der strukturelle Aufbau des Programms.

Der Entwurf des Programms sowie die Architektur von CYBRAIL basieren auf einer klaren Strukturierung der Funktionalitäten, um eine effiziente und erweiterbare Lösung zu gewährleisten. Im folgenden Abschnitt wird der Architekturentwurf näher erläutert, gefolgt von einem detaillierten Blick auf die wichtigsten Klassen und deren Interaktionen.

3.2.1 Architekturentwurf

Das Architekturdiagramm (siehe Abbildung 3.1) verdeutlicht die modulare Struktur des Log Processing Systems von CYBRAIL, das in zwei Hauptbereiche unterteilt ist: den *InfoPrep*-Bereich und die *TestingPipeline*.

Im **InfoPrep**-Bereich werden die eingehenden Logdaten vorverarbeitet. Der wesentliche verbleibende Bestandteil dieses Bereichs ist der `LogParser`, der die Logs in ein standardisiertes JSON-Format umwandelt, das für die weitere Verarbeitung verwendet wird. Die vorverarbeiteten Daten werden dann an die **TestingPipeline** weitergeleitet.

Die **TestingPipeline** führt die eigentlichen Tests auf den vorverarbeiteten Logs durch. Dabei kommen verschiedene Module zum Einsatz, die auf unterschiedliche Anomalien testen. Diese umfassen:

- **MLAnomalyDetector:** Nutzt Machine-Learning-Modelle, wie zum Beispiel zur Erkennung automatischen Tippens (FastTyping Detection).
- **RuleBasedAnomalyDetector:** Führt regelbasierte Anomalieerkennungen durch, wie die VM-Erkennung (VM Detection) oder die Erkennung mehrerer Bildschirme (Number of Displays Detection).

Am Ende des Prozesses werden alle Ergebnisse von der `ResultAggregator`-Komponente gesammelt und zu einem Gesamtbericht zusammengeführt. Dieser Bericht wird anschließend durch die `Output`-Komponente ausgegeben oder gespeichert.

Durch diese modulare Architektur kann das System flexibel erweitert werden, indem neue Testmodule hinzugefügt oder bestehende angepasst werden. Ebenso bleibt das Parsing der Logs flexibel, um neue Logformate zu unterstützen.

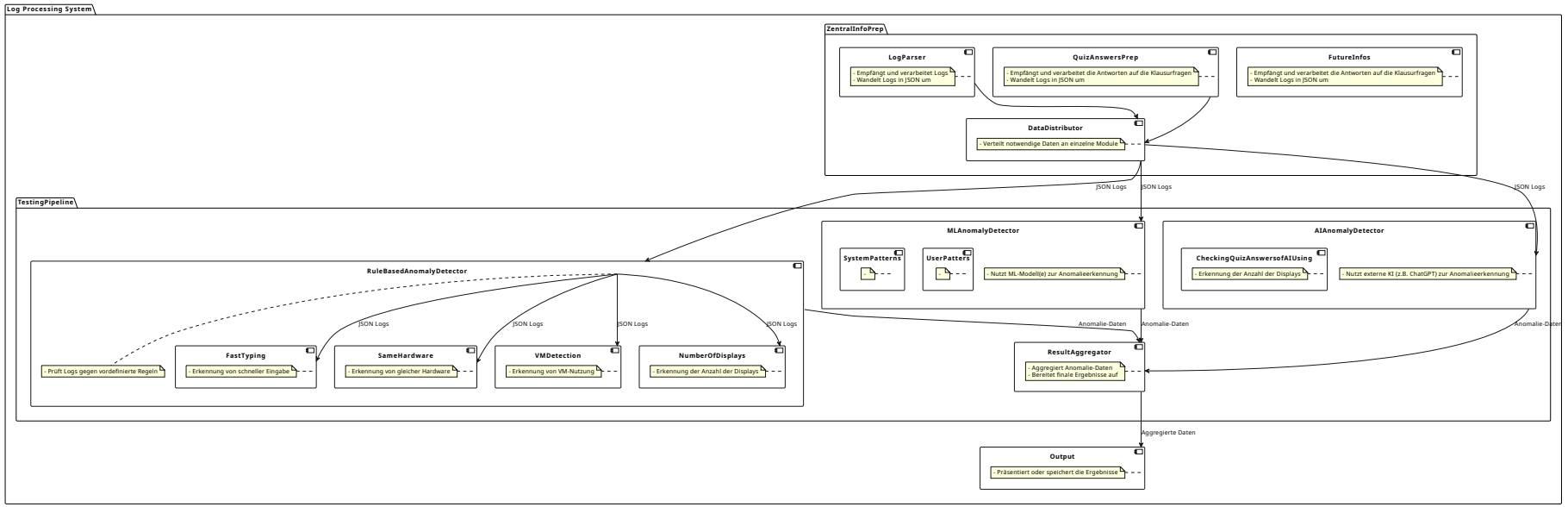

Abbildung 3.1: Architekturdiagramm des Log Processing Systems von CYBRAIL

3.2.2 Programmentwurf

Das Klassendiagramm (siehe Abbildung 3.2) zeigt das Log Processing System von CYBRAIL aus einer objektorientierten Perspektive. Die zentralen Klassen und deren Funktionen sind:

- `LogParser:` Verantwortlich für das Einlesen und Umwandeln von Logdateien in das JSON-Format.
- `DataDistributor:` Verteilt die vorverarbeiteten Daten an die verschiedenen Testmodule in der TestingPipeline.
- `RuleBasedAnomalyDetector:` Prüft die Logs auf Basis vordefinierter Regeln, wie der Erkennung von VM-Nutzung oder mehreren Bildschirmen.
- `MLAnomalyDetector:` Setzt Machine-Learning-Modelle zur Erkennung von Anomalien, wie schnellem Tippen, ein.
- `ResultAggregator:` Aggregiert die Ergebnisse aller Anomaliedetektoren und erstellt einen finalen Bericht.
- `Output:` Gibt die gesammelten Ergebnisse aus oder speichert sie.

Die modulare Struktur des Systems ermöglicht es, die einzelnen Testmodule unabhängig voneinander zu entwickeln und zu erweitern, ohne das Gesamtsystem zu beeinträchtigen. Dies erleichtert auch zukünftige Anpassungen und Erweiterungen.

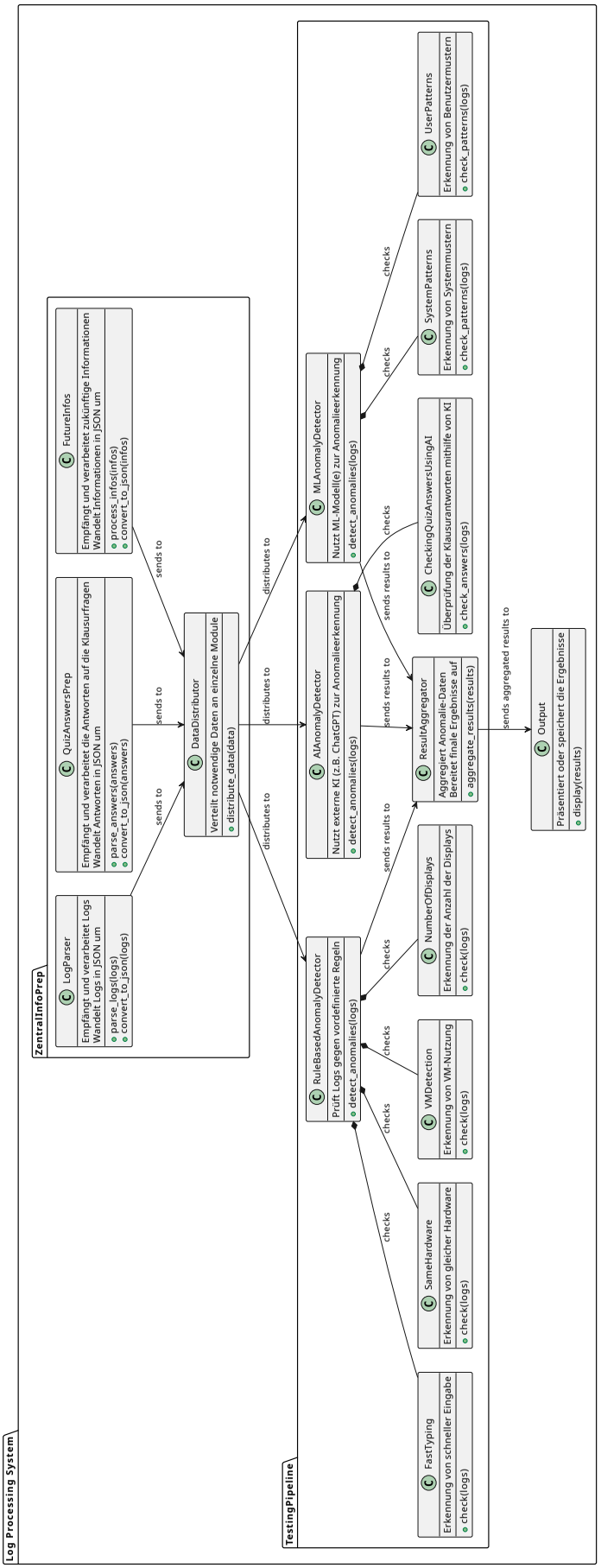

Abbildung 3.2: Klassendiagramm des Log Processing Systems von CYBRAIL

3.2.3 Zusammenfassung

Der Architektur- und Programmentwurf von CYBRAIL zeigt einen modularen Ansatz, der es ermöglicht, das System flexibel zu erweitern und an neue Anforderungen anzupassen. Die klare Trennung von Vorverarbeitung, Anomalieerkennung und Ergebnisaggregation gewährleistet, dass das System sowohl effizient arbeitet als auch gut skalierbar ist. Neue Testmodule und Anpassungen am Logparsing können einfach integriert werden, ohne die bestehende Systemarchitektur zu beeinflussen.

3.3 Logparsing und erstes MVP

Zu Beginn war es wichtig, die eigentlichen Daten, mit denen wir arbeiten wollten, in ein computerlesbares Format zu bringen und somit ein erstes Minimal Viable Product (MVP) zu erstellen.

Wir haben einen modularen Ansatz entwickelt, der mithilfe eines Strategy Patterns die unterschiedlichen Arten von Logs – sowohl die der SEB als auch andere Logtypen – unterscheiden kann und einfach um neue Logtypen erweiterbar ist. Dieses Modul nimmt alle Logdateien als Input, identifiziert den Logtyp anhand des Dateinamens und wendet schließlich die passende Strategie an, um das Log in parsierbare JavaScript Object Notation (JSON)-Dateien zu konvertieren, welche später als Grundlage für die Analysen dienen.

Für einen ersten Test haben wir außerdem ein Python-Modul entwickelt, das die geparsten Logs von einem Studenten einliest und anhand des Systemnamens erkennt, ob es sich dabei um eine virtuelle Maschine (VM) handelt.

Beide Funktionalitäten haben wir in ein erstes MVP integriert, das wir in Go geschrieben haben. Dieses MVP konnte ein Set von Logdateien parsen, das erste Modul darauf ausführen und einen Status zurückmelden.

Für die Kommunikation zwischen dem Hauptprogramm und den Modulen haben wir uns auf die Eingabe von Argumenten und die Ausgabe eines parsierbaren Outputs über `std::out` geeinigt.

3.4 UI und Batchverarbeitung

Bisher konnte unser Tool nur über die Konsole bedient werden. Um die Bedienung zu vereinfachen, haben wir uns entschieden, es mit einer Weboberfläche auszustatten. Dies haben wir direkt aus Go heraus umgesetzt, indem wir einige Methoden in API-Calls refaktoriert haben, um Programm-Ein- und Ausgaben zu realisieren.

Zeitgleich wurde das Programm erweitert, um Logdateien von beliebig vielen Studierenden auszuwerten und sowohl ein Gesamtergebnis als auch Einzelauswertungen zu erstellen. Hierfür wurde eine Programmhilfe entwickelt, die zunächst auf der Konsole den Benutzer unterstützte, alle Logs den richtigen Studierenden mit Namen und Matrikelnummer zuzuordnen. Im Verlauf wurde diese Funktionalität auch in die Webversion portiert. Über einen einfachen Knopfdruck konnten alle Logs eingesehen und den Studierenden zugeordnet werden. Sobald alle Logs zugeordnet waren, startete das Webinterface eine Analyse für jeden Studierenden und sammelte die Resultate aus allen Modulen, um diese mit detaillierten Auffälligkeiten zu präsentieren.

Um Auffälligkeiten über `std::out` ausführlich zurückzugeben, wurde ein JSON-Schema entwickelt, das genau beschreibt, welche Stelle die Auffälligkeit enthält und was daran auffällig ist.

Des Weiteren wurde eine standardisierte Modulkonfiguration eingeführt, mit der Einstellungen für einzelne Module festgelegt werden können, wie zum Beispiel Empfindlichkeit, Keywords etc. Die Anpassung dieser Konfigurationen wurde ebenfalls in das Webinterface integriert. Diese Einstellungen sind generell vorhanden, und das User Interface (UI) baut sich automatisch auf Grundlage der Konfigurationen auf, die in der jeweiligen Datei für ein Modul gefunden werden.

Neben der Weiterentwicklung von Modulen, um Auffälligkeiten wie Netzwerkänderungen, Bildschirmaktivitäten oder Integritätschecks zu demonstrieren, wurde auch eine Python-Bibliothek entwickelt, um die einzelnen Module zu vereinheitlichen und doppelten Code zu refaktorisieren. In jedem Modul sind das Einlesen der Logdateien und der Modulkonfigurationen sowie die Rückgabe von Auffälligkeiten einheitlich gestaltet.

3.5 Client-App und KI-Auswertung

Eines unserer ersten Ziele war es, sowohl KI-generierte Inhalte als auch automatisches Tippen zu erkennen. Die Erkennung von Künstliche Intelligenz (KI)-generierten Inhalten mussten wir jedoch ausschließen, da wir hierfür die Antworten aller Studierenden zusätzlich aus Moodle hätten extrahieren müssen.

Für die Erkennung, ob automatisches Tippen verwendet wurde – eine Alternative zu Copy-Paste – haben wir zwei Szenarien identifiziert:

1. Anschläge mit konstanten Zeitabständen
2. Anschläge mit variierenden Zeitabständen, die unauffälliger wirken

Um beide Fälle erkennen zu können, haben wir uns dazu entschieden, eine KI zu trainieren, die mit hoher Wahrscheinlichkeit diese beiden Szenarien als potenzielle Betrugsversuche erkennt. Dazu wurde ein Programm geschrieben, das automatisch getippte Daten generiert. Als Input dienten KI-generierte Texte, die in Wortwahl, Grammatik etc. den Klausurantworten ähneln sollten. Diese Texte wurden aufgeteilt: Ein Teil wurde mit konstanten Intervallen aufgezeichnet, der andere Teil mit Variationen, die durchschnittliches menschliches Tippverhalten nachahmen sollten.

Als Grundlage für die Verarbeitung haben wir ein Format entwickelt, das lediglich die Zeit zwischen jedem Tastenanschlag speichert. Nach der manuellen Generierung von menschlichen Testdaten wurden sowohl die maschinellen als auch die menschlichen Timing-Daten der KI für das Training bereitgestellt. Nach kurzer Trainingszeit erzielte die KI auf ihrem Testdatensatz eine hohe Genauigkeit und konnte auch in Praxistests gut vorhersagen, ob ein Mensch oder eine Maschine getippt hatte.

Um diese Timinginformationen zu erhalten – die im Standardlog des SEB nicht enthalten sind – haben wir begonnen, eine Client-App zu entwickeln. Im aktuellen MVP speichert diese App die Timing-Daten in einer Logdatei, die dann vom KI-Modul im Programm ausgewertet werden kann.

3.6 Kompletter Prototype

Um unseren Prototypen zu verfollständigen wurde nun noch die Clientapp verfollständigt, so das diese am Ende einer Klausur alle Logdateien an unser Programm zur auswertung sendet. Auch wurde das Program als ein Dockercontainer verpackt um einfaches Deployment zu gewährleisten. Die Clientapp kann einfach als .exe auf dem zielsystem ausgeführt werden.

4 Ergebnisse und Leistungen

Als Gesamtergebnis haben wir nun einen Docker-Server und eine Client-App entwickelt, die alle Logdateien des SEB senden und Timing-Informationen der Eingaben der Studierenden speichern kann, um diese später auf ihre Legitimität hin zu überprüfen.

Der Docker-Server verfügt über ein eigenes Webinterface, mit dem die Auswertung der Logdateien einer Klausur für alle Studierenden verwaltet werden kann. Alle Logdateien der Studierenden werden an den Server gesendet und dort gesammelt. Nach der Zuordnung der Logs zu den jeweiligen Studierenden wird ein modularer Analyseprozess gestartet. Das Programm kann beliebige Module aufrufen, die dem Standard entsprechen, und sammelt von jedem dieser Module Ergebnisse für jeden Studierenden. Sollte ein neues Modul erstellt oder Anpassungen vorgenommen werden müssen, kann dies einfach ohne Neukompilierung geschehen.

Ähnliches gilt für die Konfigurationen der einzelnen Module. Jede Konfiguration ist ebenfalls generisch, und das UI zur Konfiguration der Module setzt sich automatisch aus den vorhandenen Konfigurationsdateien zusammen.

Auch das Parsing der Logs funktioniert ähnlich. Mithilfe eines Strategy-Patterns können neue Logtypen einfach integriert und in einem Modul verarbeitet werden.

Aktuell stehen folgende Module zur Verfügung, die:

- Copy-Paste-ähnliches Autotyping erkennen können
- Auffälligkeiten in der Displaykonfiguration (z.B. zu viele Displays oder Änderungen der Auflösung) erkennen können

- Auffälligkeiten bei Neuinitialisierungen des SEB feststellen können
- Die Integrität des SEB prüfen können
- Eine VM erkennen können, wenn der SEB innerhalb einer VM gestartet wird, was normalerweise durch den SEB verhindert wird
- Ungewöhnliche Netzwerkaktivitäten feststellen können
- Ungewöhnliches Shutdown-Verhalten des SEB erkennen können

Zusätzlich zu diesen Modulen steht eine Python-Bibliothek zur Verfügung, die wiederkehrende Methoden in den Modulen abstrahiert.

5 Risikoanalyse und Risikomanagement

Bei der Entwicklung von CYBRAIL wurden verschiedene Risiken identifiziert, um mögliche Hindernisse oder Probleme während des Projekts frühzeitig zu erkennen und Gegenmaßnahmen zu definieren. Die Risiken lassen sich in fünf Hauptkategorien unterteilen:

- **Technische Risiken:** Diese umfassen Softwarefehler, Kompatibilitätsprobleme zwischen verschiedenen Versionen des Safe Exam Browsers (SEB) und Sicherheitslücken.
- **Datenschutz- und Compliance-Risiken:** Hier geht es vor allem um Datenschutzverletzungen und die Einhaltung von Vorschriften zum Schutz der Daten der Studierenden.
- **Betriebliche Risiken:** Diese betreffen Ausfälle oder Unterbrechungen des Systems sowie Skalierbarkeitsprobleme.
- **Annahme- und Akzeptanzrisiken:** Hierzu gehören der Widerstand von Lehrenden und Studierenden sowie der Mangel an Schulungen und Unterstützung.
- **Projektspezifische Risiken:** Diese umfassen den fehlenden Zugriff auf Logdaten der Studierenden und die Anbindung an universitäre Dienste wie Moodle nach der Implementierung.

Basierend auf der Wahrscheinlichkeit und dem Schadensausmaß jedes Risikos wurde eine Risikoprioritätszahl (RPZ) berechnet. Diese gibt an, wie dringend das Risiko behandelt werden muss. Für jedes Risiko wurden außerdem Gegenmaßnahmen entwickelt, um dessen Auswirkungen zu minimieren oder ganz zu vermeiden.

Die folgende Tabelle fasst die wichtigsten Risiken zusammen, die im Projekt identifiziert wurden:

Risiko	**Schadenshöhe (1–10)**	**Wahrscheinlichkeit**	**RPZ**	**Gegenmaßnahmen**
Softwarefehler und -probleme: Wie man sie handhabt und was nach dem Projekt passiert	2	0,7	1,4	FAQ, Tests, abhängig von der Geschichte
Kompatibilitätsprobleme mit verschiedenen Versionen des SEB	8	0,1	0,8	Konfigurierbar
Sicherheitslücken	7	0,5	3,5	Sicherheitskonzepte, Penetrationstests
Datenschutzverletzungen	8	0,4	3,2	Sicherheitskonzepte, Penetrationstests, Datenspeicherlebensdauer
Einhaltung der Datenschutzvorschriften der Studierenden	7	0,2	1,4	Datensparsamkeit
Ausfallzeiten und Unterbrechungen	2	0,2	0,4	–
Skalierungsprobleme	2	0,2	0,4	–
Widerstand von Lehrenden und Studierenden	2	0,6	1,2	–
Mangel an Schulungen und Unterstützung	8	0,3	2,4	Dokumentation, Benutzeroberfläche
Kein Zugriff auf Logdaten der Studierenden	10	0,25	2,5	–
Kein Zugriff auf universitäre Dienste wie Moodle nach der Implementierung	10	0,1	1	–

Tabelle 5.1: Risikobewertung und Gegenmaßnahmen

18

5.1 Zusammenfassung der Risikobewertung

Die Risikobewertung zeigt, dass die kritischsten Risiken in den Bereichen Datenschutzverletzungen und Sicherheitslücken liegen, die beide durch entsprechende Sicherheitsmaßnahmen und Penetrationstests gemindert werden können. Gravierend wäre jedoch der fehlende Zugriff auf die Logdaten der Studierenden, da diese essenziell für die Funktion unseres Tools sind. Ohne diese Daten wäre eine Auswertung und Erkennung von Betrugsversuchen nicht möglich. Allerdings schätzen wir die Wahrscheinlichkeit, dass hierfür keine Lösung gefunden werden kann, als sehr gering ein. Mit den entsprechenden technischen und organisatorischen Maßnahmen, wie der Nutzung des SEB-Servers und der korrekten Konfiguration des Systems, sollten wir in der Lage sein, auf die notwendigen Logs zuzugreifen und diese sicher zu analysieren.

6 Kommunikation und Qualitätssicherung

Um eine gute Kommunikation zu gewährleisten, haben wir uns entschieden, alle 2–3 Wochen ein Meeting auf agile Weise abzuhalten. In diesen Meetings wurde jeweils ein MVP vorgestellt und besprochen, was im nächsten Sprint bis zum folgenden Meeting umgesetzt werden sollte.

Zur Unterstützung dieser agilen Arbeitsweise haben wir eine eigene OpenProject-Instanz gehostet, mit der wir unsere Workpackages (Epics, User Stories, Tasks, Bugs, etc.) verwalteten. Zudem wurden hier Notizen zur Vor- und Nachbereitung der Meetings geführt, um eine klare Dokumentation sicherzustellen.

Durch diese regelmäßige Kommunikation und die Vorstellung des aktuellen Standes konnten wir eine hohe Qualität gewährleisten und sinnvolle Architekturen sowie Programmier-Patterns von Anfang an planen oder durch Refactoring verbessern, um eine übersichtliche Code-Basis zu erhalten.

In diesem Kapitel werden die wichtigsten Erkenntnisse aus dem Projekt zusammengefasst. Diese betreffen sowohl die Kommunikation im Team als auch das Management und technische Herausforderungen, die im Verlauf des Projekts aufgetreten sind.

6.1 Kommunikation

Die Kommunikation mit dem Product Owner und den Kollegen ist entscheidend, um Problemstellen effizient zu lösen. Unterschiedliche Sichtweisen haben oft dazu beigetragen, eine gemeinsame Lösung zu finden, die sich aus mehreren Teilansätzen und Ideen zusammensetzte.

6.2 Vereinbarung von Terminen

Eine klare Vereinbarung von Meeting-Terminen und Deadlines hat nach anfänglichen Schwierigkeiten dazu geführt, dass alle Teammitglieder pünktlich an den Besprechungen teilnehmen konnten.

6.3 Strukturiertes Projektmanagement

Eine klare Definition von User Stories und Tasks hat maßgeblich dazu beigetragen, die Zusammenarbeit zu verbessern. Dies stand im deutlichen Kontrast zu unstrukturiertem Arbeiten, bei dem Teammitglieder unkoordiniert implementierten.

6.4 Remote-Prüfungen

Das gesamte Thema der Remote-Prüfungen hat sich als äußerst anspruchsvoll herausgestellt, insbesondere in Bezug auf die Frage, wie Betrug effektiv verhindert werden kann. Am Ende bleibt kein System vollkommen sicher, insbesondere wenn die Klausurteilnehmer Informatik studieren – nach dem Motto „alles ist Open Source, wenn du Assembly verstehst“. Letztendlich kann jedes System so modifiziert werden, dass es den Anschein erweckt, als sei alles rechtmäßig abgelaufen. Die einzige Möglichkeit besteht darin, die Hürde für Betrug so hoch zu setzen, dass

nur wenige Studierende sie überwinden können – ähnlich wie bei Präsenzklausuren, wo Betrug ebenfalls nicht vollständig ausgeschlossen werden kann.

7 Fazit und Ausblick

Abschließend lässt sich sagen, dass unser Projekt nahezu alle Ziele erreicht hat. Wir konnten erfolgreich zeigen, dass es möglich ist, automatisch die Logdateien von Studierenden auszuwerten und diese auf Auffälligkeiten zu untersuchen.

All dies geschah in einem sehr flexiblen Framework, das sich an veränderte Gegebenheiten anpassen kann.

Es bleibt jedoch fraglich, ob ein solches oder ähnliches System in der Praxis eingesetzt werden darf. Letztlich verfügt der SEB mit dem SEB-Server bereits über eine eigene Möglichkeit, Logdateien an einen zentralen Server zu übermitteln.

Es besteht jedoch großes Potenzial, weitere Testmodule zu entwickeln und diese auf ein praxisnahes Szenario anzupassen. Es gab viele Vorschläge, was man zusätzlich in den Logs erkennen könnte. Dank des generischen Ansatzes der gesamten Architektur ist es durchaus möglich, das Programm in Zukunft wieder aufzugreifen und weiterzuentwickeln.

Disclaimer

Dieses Projekt wurde im Rahmen einer Machbarkeitsstudie entwickelt, um zu prüfen, ob es technisch möglich ist, die Logdateien von Studierenden automatisch auszuwerten. Das Programm spiegelt in seinem aktuellen Zustand den experimentellen Charakter dieser Studie wider und ist nicht für den produktiven Einsatz vorgesehen.

Der Quellcode ist unter folgendem Link verfügbar: https://github.com/Lexxn0x3/CYBRAIL. Das Projekt steht unter der `GPL-3.0`-Lizenz, die ebenfalls im Repository zu finden ist.

Abbildungsverzeichnis

Tabellenverzeichnis

Akronyme

CYBRAIL Cyber Barrier for Reliable Academic Integrity and Log-analysis. i, 1, 3, 6, 17

JSON JavaScript Object Notation. i, 11, 12

KI Künstliche Intelligenz. i, 13

MVP Minimal Viable Product. i, 11, 13, 20

SEB Safe Exam Browser. i, 1, 2, 3, 11, 13, 15, 16, 19, 23

UI User Interface. i, 12, 15

VM Virtuellen Maschine. i, 2