Dokumentation &

Projekttagebuch

Innovation Lab 3

Jahr 2022/23

Projekt: Code Quality Tools | PHP

Team: 22

1. Allgemeine Informationen

**Projektname:** Code Quality Tools | PHP

**Supervisor:** Dipl.-Ing. (FH) Bernhard Wallisch

Innovation Lab 3, Wintersemester 2022/23

**Projektteam:**

Taha Samaha, [if20b150@technikum-wien.at](mailto:if20b150@technikum-wien.at), Projektleitung

Fatema Aly, [if20b057@technikum-wien.at](mailto:if20b057@technikum-wien.at)

Jan Loos, [if20b118@technikum-wien.at](mailto:if20b118@technikum-wien.at)

**Management-Summary des Projektes**

Das Projekt dient zur Reduzierung des Korrekturaufwands von PHP-Quellcode und soll zur Verbesserung der Code Qualität von Studenten beitragen. Der Fokus liegt primär auf Sicherheit. Klassische Programmierfehler, veraltete oder unsichere Funktionen sollen erkannt und verständlich aufgearbeitet werden, damit Probleme übersichtlich und leicht zu beheben sind. Nach Möglichkeit sollen auch bewährte Programmiertechniken (so genannte „Best Practices“) angewendet werden.

**Rahmenbedingungen und Projektumfeld**

Die Software wird in einem Docker Container laufen. Die Ausgabe sollte in einer für die FH ausgelegten HTML erfolgen. Die Code Quality wird hinsichtlich Safety und Security getestet, dies kann man sowohl mit Unit-Tests, also auch Code-Style oder Code Patterns bzw. best practice Ansätzen überprüft werden. In diesem Projekt wäre es Ziel hinsichtlich PHP eine CI/CD Pipeline zu erzeugen, die es Studierenden ermöglichen soll ihren Quellcode inklusive weiteren Aspekten einfach über das bestehende EspressoIT-System zu überprüfen. Die Dieser Prozess soll zu einer verbesserten Codequalität führen und als Resultat eine Bewertung in Form eines HTML Exports liefern.

**Semester-Roadmap**

**Aktueller Plan, Stand WS2022/23**

Timeline

Description automatically generated

**INNO2 Plan, Stand SS2022**

Timeline

Description automatically generated

**INNO1 Plan, Stand WS2021/22**

Timeline

Description automatically generated

**Collaboration & Tooling**

ALM: https://dev.azure.com/Fatema-aly/Inno\_Lab1

Trello: https://trello.com/codequalitytoolsphp

GitHub: https://github.com/Alyf92/innoLab\_1

**Espresso-IT Tool Integration**

https://git.technikum-wien.at/innolab/espresso-it/espresso-it/-/tree/feature/php-multi-linter

1. **Projekt-Kurzbeschreibung**

Ziel des Projekts ist es eine Verbesserung oder eine Erweiterung für das Moodle-Plugin EspressoIT zu entwickeln, um zusätzliche Code Quality und Safety Funktionen für PHP Dateien zu unterstützen. Die Dateien sollen automatisch im Hintergrund auf vorgefertigte Kriterien getestet werden und anschließend soll das Resultat in ein maschinenlesbares Format ausgegeben werden. Dies soll den Korrekturaufwand für Lektoren minimieren und Studenten übersichtliches und verständliches Feedback zu ihrem Code bieten.

Die größte Herausforderung wird bei der Integration in das bereits bestehende EspressoIT erwartet. Zugriff auf den Quellcode von EspressoIT und Testumgebungen wurden zur Verfügung gestellt, jedoch ist zurzeit noch nicht bekannt, wie kompliziert sich die Integration unserer Software, die wir im letzten Semester geschrieben haben, herausstellen wird. Es wird gegebenenfalls erstmal der fremde Quellcode studiert werden müssen.

Es war ursprünglich das Ziel zum Ende des zweiten Semesters erfolgreich eine PHP Datei mit EspressoIT zu verarbeiten, so dass am Ende ein Resultat herauskommt. Jedoch hat sich im Verlauf des zweiten Semesters herausgestellt, das EspressoIT für die Weiterentwicklung nicht geeignet war und uns daher keine Ressourcen zur Entwicklung zur Verfügung gestellt werden konnten, weshalb stattdessen eine eigene Software entwickelt wurde.

Im dritten Semester wird versucht unsere eigene Software in EspressoIT zu integrieren. Bis zum Abschluss des dritten Semesters soll es Lektoren möglich sein bei Abgaben auf Moodle unsere Software bei EspressoIT auszuwählen und Studierende sollen dazu in der Lage sein, Feedback für ihren Quellcode zu erhalten.

Um den besten Wert für AnwenderInnen bieten zu können, sollte das generierte Feedback qualitativ hochwertig und gut strukturiert sein. Schwerwiegende Fehler dürfen nicht in einer Flut Folgefehler untergehen. Das generierte Feedback sollte objektiv bleiben und sich möglichst auf Fakten beruhen und dem Anwender keine subjektiven Veränderungen aufzwingen.

Es ist wichtig klarzustellen, dass die Software nicht eigenständig Fehler behebt, sondern lediglich den Nutzer auf seine Fehler aufmerksam machen soll. Unter gewissen Umständen, zum Beispiel bei veralteten oder unsicheren Funktionen, kann die Software einen Verbesserungsvorschlag beziehungsweise mehr Informationen über den Fehler anbieten.

Da für die Umsetzung bereits existierende quelloffene Software in EspressoIT integriert wird, muss die Funktionsweise von EspressoIT analysiert und gegebenenfalls angepasst werden.

Am Ende des Projekts soll die Software von einem Docker Container aus im Hintergrund gestartet werden können. Die Software soll PHP Dateien auswerten und Ergebnisse in einer für die FH ausgelegten HTML ausgeben.

1. **Spezifikation der Lösung**

*< Nach Abschluss der Auftragsklärung (Vorprojekt-Phase) starten Sie die Projektdurchführung. Erstellen Sie parallel zur Umsetzung Ihres Projekts über die Sprints hinweg eine Spezifikation Ihrer Lösung!*

*Vor jedem Sprint müssen mindestens jene Details spezifiziert sein, die Sie im kommenden Sprint umsetzen werden. Verwenden Sie hierzu Techniken wie zum Beispiel das Schreiben von Epics & User Stories und bauen Sie ein Product Backlog auf (wenden Sie die LV-Inhalte aus der LV Agiles Projektmanagement an).*

*Verwenden Sie zur Spezifikation generell Visualisierungstechniken, die zur Aufgabenstellung passen. Zum Beispiel können ergänzend zu den Mockups und User Stories auch Datenbankdiagramme, Klassendiagramme, oder Sequenzdiagramme (Darstellung zeitlicher Abläufe) sinnvoll sein.*

*Normalerweise geht man vom Groben zum Detail vor. Die Gliederung dieses Abschnitts kann wie folgt aussehen:*

* *Systemumfeld: Beschreiben Sie die Abgrenzung der umzusetzenden Lösung (Systemgrenzen)*
* *Features (Funktionale Anforderungen): Alle geforderten Lösungseigenschaften - bei Software i.d.R. die Features bzw. eine Beschreibung dieser als User-Stories o.ä.)* 
  + *Erstellen Sie Screen Mockups aller wesentlichen UI-Ansichten!*
* *Schnittstellen: Alle relevanten Schnittstellen Ihrer Lösung.*
* *Qualitätseigenschaften, technische Anforderungen (Nicht-Funktionale Anforderungen): Performance, Skalierbarkeit, Verfügbarkeit, Benutzbarkeit, Angaben zur Architektur und Erweiterbarkeit, etc.*
* *Sonstige „nicht auf den ersten Blick klare“ aber wesentliche Lösungsmerkmale!*

*Stimmen Sie mit Ihrer Betreuer\*in ab, wie die Spezifikation aufgebaut sein soll!*

*Fragen Sie nach, wann immer Sie das Gefühl haben, es könnte sich um ein Missverständnis oder unterschiedliche Erwartungen handeln oder wenn Sie eine Anforderung nicht ganz klar verstanden haben! >*

1. **Aufwandschätzung**

Diese Aufwandsschätzung wurde mit der Delphi Methode durchgeführt, um den Aufwand für dieses Semester zu schätzen. Dabei haben wir intern im Team Brainstorming angewandt, um die kommenden Arbeitspakete zu erfassen. Eine Hierarchiestufe darunter zählten wir die Epics auf und in der letzten Stufe die User Story oder die Beschreibung der Aufgabe. Dieses Szenario durchliefen wir einige Male, um auf alle einzelnen Aufgaben einzugehen. Anschließend haben wir gemeinsam als Team die Stunden für die Spalten Optimistisch (Sp), Wahrscheinlich (Sp), Pessimistisch (Sp) geschätzt. Als Schlussfolgerung dieser Schätzung ist uns noch deutlicher geworden, dass die Aufwandsschätzung von der EspressoIT Implementation für uns schwer zu schätzen ist, in Hinsicht vom Zeitaufwand und den nötigen Arbeitspaketen. Hier ein Screenshot und in der Anlage befindet sich die Excel Datei.

Ein Bild, das Tisch enthält.

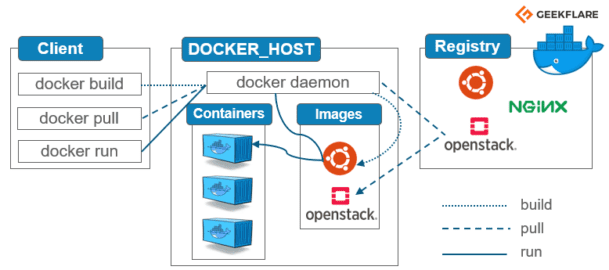
Automatisch generierte Beschreibung

1. **Auslieferung**

Die fertige Lösungskomponente kann der Branch feature/php-multi-linter vom Espresso-IT Git Repository entnommen werden: [Git-Link](https://git.technikum-wien.at/innolab/espresso-it/espresso-it/-/tree/feature/php-multi-linter)

Hierfür ist jedoch eine VPN-Verbindung zum FH Technikum-Wien nötig. Damit Sie von extern im LAN der FH Technikum-Wien arbeiten können, wurde ein VPN (Virtual Private Network) Zugang eingerichtet. Dieser Dienst steht für Studierende, MitarbeiterInnen und Lehrende der FH Technikum Wien zur Verfügung. Für den Zugang benötigen Sie das FH Technikum-Wien Zertifikat [rulebreaker.ca.crt](https://cis.technikum-wien.at/cms/dms.php?id=88057) und die VPN Konfigurationsdatei [rulebreaker.ovpn](https://cis.technikum-wien.at/cms/dms.php?id=88058). Die VPN Client Software (openvpn) und die dazugehörige vpn Konfigurationsdatei können Sie per Klick auf das jeweils verwendete Betriebssystem herunterladen. Bitte denken Sie daran, Ihren Browsercache zu löschen, falls die ursprünglich von Ihnen gewünschte Seite nicht erreicht werden kann. Sollten Sie Probleme beim Verbinden mit dem vpn haben, ist der ein Support unter der E-Mail-Adresse [support@technikum-wien.at](mailto:support@technikum-wien.at) für Sie erreichbar.

Die [Docker Architektur](https://geekflare.com/de/docker-architecture/) erklärt die Systemarchitektur des Espresso-IT und den dazugehörigen Tools im Projekt sehr gut. Beim Docker-Container haben Sie jetzt ein einziges Betriebssystem, und die Ressourcen werden von den Containern gemeinsam genutzt. Daher ist es leicht und bootet in Sekunden. Unten sehen Sie das einfache Diagramm einer Docker-Architektur.



Das Installieren des gesamten Systems wurde ausführlich im [Git-Wiki](https://git.technikum-wien.at/innolab/espresso-it/espresso-it/-/wikis/home) beschrieben. Das Tool PHPML läuft automatisch mittels eines Dockerfile vom Ordner [tool\_php\_multilinter](https://git.technikum-wien.at/innolab/espresso-it/espresso-it/-/tree/feature/php-multi-linter/tool_php_multilinter) aus als Container.

**Verwendete Dritt-Software / Frameworks**

* PHPMD <https://phpmd.org> By Manuel Pichler licensed under BSD 3-Clause
* PHPStan <https://phpstan.org> © 2016–2023 Ondřej Mirtes

**System Anforderung**

Hier halten wir uns an den Systemanforderungen von Docker Desktop bzw. Docker WSL

* Für Windows siehe: <https://docs.docker.com/desktop/install/windows-install/>
* Für Linux siehe: <https://docs.docker.com/desktop/install/linux-install/>
* Für Mac siehe: <https://docs.docker.com/desktop/install/mac-install/>

1. **Unser Projekt-Tagebuch**

Sprint 1 bis 3

**Taha Samaha:**

Meine Aufgaben waren Kommunikation mit unserem Betreuer, Aufgabenaufteilung und die erste Anlaufstelle bei Komplikationen zu sein. Die erste Espresso-IT/Moodle Einrichtung wurde im Laufe eines Zoom Calls auf meinem lokalen PC eingerichtet und diente zugleich als Anleitung für meine Teamkollegen. Momentan hat das gesamte Team das Projekt lokal eingerichtet. Nächste Aufgabenschritte wurden mit unserem Betreuer besprochen und nach Austausch im Team sind wir zur folgenden Aufteilung gekommen:

* Docker Container - (Java: Kramreiter, PHP: Samaha)
* tool.py / run file - (Java: Aly, PHP: Loos)

Weiters machen wir sich noch mit der Projektumgebung vertraut und notieren sich aufkommende Fragen.

**Jan Loos:**

Für das PHP Tool Skript konnten wir unsere Arbeit aus den ersten beiden Semestern zum größten Teil übernehmen. Konkret musste unser Code dafür lediglich in Funktionen eingepackt und ein paar Variablen ausgetauscht werden. Wir haben das tatsächliche Auswerten der Dateien und die Integration in EspressoIT in zwei getrennten Skripten gelassen, da es unseres Erachtens nach dem Code übersichtlicher macht. Um die Funktionen von EspressoIT weitestgehend ausnutzen zu können, haben wir außerdem das Auslesen der tatsächlichen Zeilen Code in unserem Run Skript mit aufgenommen. Für das Tool Skript selbst, haben wir uns bereits zuvor geschriebene Tool Skripts angeschaut und das nötigste abgeschaut. Während der Entwicklung ist es zu keinen Problemen gekommen und auch vereinzeltes Testen verlief einwandfrei, jedoch wurde das Tool Skript noch nicht in Zusammenhang mit EspressoIT getestet werden können.

**Fatema Aly:**

Da die meisten von uns in den letzten beiden Semestern nicht mit dem Java-Tool gearbeitet haben, sollten wir uns für das Java-Tool-Skript zunächst mit dem von der anderen Gruppe ausgewählten Tool beschäftigen. Zu diesem Zweck wurde Java eingerichtet und das Tool Checkstyle lokal installiert. Beim Installieren und Starten des Tools stießen wir auf einige Fehlermeldungen, z.B.: Error " **A JNI error has occurred. Please check your installation and try again** ". Das Problem war, dass Java JDK 17 bereits installiert war. Gleichzeitig hatten wir aber Java 8 (JRE) installiert. Die Fehlermeldung konnte durch die Deinstallation von JAVA 8 (JRE) dann behoben werden. Für die Implementierung des Java Tool skripts selbst haben wir uns das PHPStan Tool Skript, das von Lektoren auf GIT geschrieben wurde, genauer angeschaut sowie weitere Tools in Python online recherchiert und uns mit ihnen auseinandergesetzt, um bald mit der Entwicklung des Tools beginnen zu können.

Sprint 4

**Taha Samaha**

Mithilfe der bestehenden Branch feature/phpstan konnten wir sehen, wie die zu erstellenden Dateien ausschauen sollen. Da unser zusammengesetztes Tool PHPStan und PHPMD nutzt, habe ich mir erlaubt das Dockerfile von dieser Branch zu nutzen und anzupassen. Für die Konfiguration des Docker Container habe ich Einträge in den Dateien **.env (Port), initial\_db\_data.json und docker-compose.yml** eingetragen. Beim builden des Docker Container kamen einige Komplikationen auf, jedoch waren, dass größtenteils Tippfehler und wurden dann korrigiert. Momentan ist der Container build fähig, jedoch besteht noch ein Package Fehler im tool.py File, worum sich mein Kollege Jan Loos kümmert. (siehe Screenshot). Währenddessen versuche ich aufkommende neue Begriffe nachzuschlagen und zu verstehen. Unser nächster Schritt ist das Testen des integrierten Tools am Testserver.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

**Fatema Aly:**

In diesem Sprint begannen wir mit der Implementierung des Skripts für das Java-Tool, wobei wir auf mehrere Probleme mit Python stießen. Das erste Problem war, dass Python zwar auf Ubuntu VM installiert war, aber nicht funktionierte. Als wir versuchten, den Python-Interpreter in VSCode auf Ubuntu auszuwählen, zeigte VSCode diese Fehlermeldung an: “Python is not installed, please download and install it“, daher beschlossen wir, das Skript unter Windows zu implementieren.

Mit subprocess() haben wir das Tool in Python unter Verwendung von einem Java-code Snippet ausgeführt und folgende Ergebnisse erhalten:

Text

Description automatically generated

Nächstes wurden die Ergebnisse in einem XML-Report generiert. Jedoch wurde das Tool noch nicht fertig implementiert.

A picture containing background pattern

Description automatically generated

Sprint 5

**Taha Samaha**

Nun läuft das Tool als eigener Container über Docker und ist betriebsbereit. Nachdem Jan Loos den Fehler vom Sprint 4 behoben hat, habe ich versucht die Änderungen an das neue Docker Image zu adaptieren jedoch lief es einige Male schief und habe letztendlich das komplette espresso-it neu aufgesetzt mit dem Command „docker-compose build“. Momentan teste ich das Tool durch das Moodle und zeichne die Ergebnisse auf.

Bemerkung: Um Docker in der Ubuntu Konsole nutzen zu können, muss Docker Desktop laufen (Screenshot)

**Jan Loos**

Da das Docker Container für unseren PHP Multilinter bereit ist, kann nun das Plugin getestet werden. Damit das Plugin erfolgreich Ergebnisse ausgibt, mussten die Pfade der Linter angepasst werden, damit die mit denen des Containers übereinstimmen. Außerdem musste die „dominate“ Implementierung auskommentiert werden, da diese zurzeit nicht benötigt wird. Des Weiteren wurde vergessen, die Ergebnisse überhaupt auszugeben. Beim ersten Testanlauf fiel auf, dass das Plugin denselben Fehler mehrmals ausgibt. Es stellte sich heraus, dass Python Variablen als Referenzen übergibt und die Ergebnisse daher konstant überschrieben hat. Dies wurde mit der copy Funktion von Python behoben. Auch aufgefallen ist, dass die Dateipfade falsch angezeigt werden, welches an einem Fehler in der Implementierung lag. Der Code wurde überarbeitet, damit solche Fehler in Zukunft seltener passieren. Ansonsten sind keine weiteren Fehler beim Testen aufgefallen und es wird davon ausgegangen, dass der PHP Multilinter wie erwartet funktioniert.

**Fatema Aly**

Für das Java-Tool haben wir versucht, die Xml-Output zu parsen, was bisher leider nicht funktioniert hat. Zu diesem Zweck haben wir verschiedene Module ausprobiert, wie **xml.dom.minidom** und **lxml Ertree**, die uns nicht geholfen haben.

Dann stießen wir auf ein weiteres Modul, nämlich **xmltodict**, was die Arbeit mit XML wie die Arbeit mit JSON aussehen lässt. Hier wurden wir mit der Fehlermeldung konfrontiert: "not well-formed (inavlid token)".

Text

Description automatically generated

Um den Fehler zu beheben, wurden mehrere Wege ausprobiert, z. B. "replace", was eine Kopie zurückgibt, in der alle Vorkommen des Subprozesses durch neue ersetzt sind. Auch auf diese Weise konnte das Problem nicht behoben werden, deswegen wird weitergesucht, wie der Fehler behoben werden kann.

Text

Description automatically generated

Sprint 6

**Taha Samaha**

Anfangs des Sprints gab es einige Komplikationen mit der VPN-Verbindung und dem Moodle Container, der nach einigen Sekunden sich wieder herunterfuhr Doch nach löschen aller Images und Container samt Hintergrunddaten und anschließendem Rebuilden lief er wieder einwandfrei. Beim Anlegen eines Assignments konnte die Config Files nicht an den Server gesendet werden. Nach dem Neueintragen des Espresso-IT URL in den Moodle Settings konnte die Config Files wieder gesendet werden. Weiters wurde noch an der Projekt Dokumentation gearbeitet.

**Ein Bild, das Text, Screenshot, Monitor, Bildschirm enthält.

Automatisch generierte Beschreibung**

**Ein Bild, das Text enthält.

Automatisch generierte Beschreibung**

**Jan Loos**

Während dieses Sprints wurde der PHP multilinter ausführlich getestet. Es wurden unterschiedliche Konfigurationen und Abgaben ausprobiert, um qualitative Unterschiede feststellen zu können. Es konnten keine auffallende Qualitätsverluste festgestellt werden, jedoch entgehen PHPMD und PHPStan manche Inhalte, die verbessert werden könnten, da beide Tools sehr spezialisiert sind und somit nicht alle möglichen Inhalte abdecken können. Es wäre daher vielleicht ratsam, den multilinter um weitere Tools zu erweitern für noch ausführlichere und weitreichendere Resultate.

**Fatema Aly**

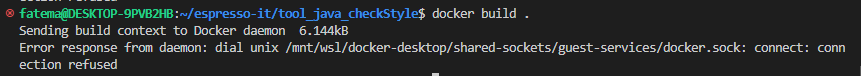
In diesem Sprint wurde Das Python Skript mittlerweile fertig implementiert und ist nun in der Lage das CheckStyle Tool auszuführen und die Ergebnisse Temporär in einer HTML-Datei zu sortieren.

Text

Description automatically generated with medium confidence

Als Nächstes begannen wir mit dem Dockerfile, wo wir auf mehrere Probleme stießen.  
  
Text

Description automatically generated

Als wir versuchten, das Image aus dem Dockerfile zu erstellen, stießen wir auf die folgende Fehlermeldung:  


Das Problem war, dass das Image ziemlich groß ist und die einzige Lösung dafür war, andere Docker-Images zu löschen, um den Fehler zu beheben.

Das nächste Problem betrifft den Moodel-Container, der immer wieder herunterfährt, obwohl er mit "docker compose up -d" gestartet wurde.Text

Description automatically generated