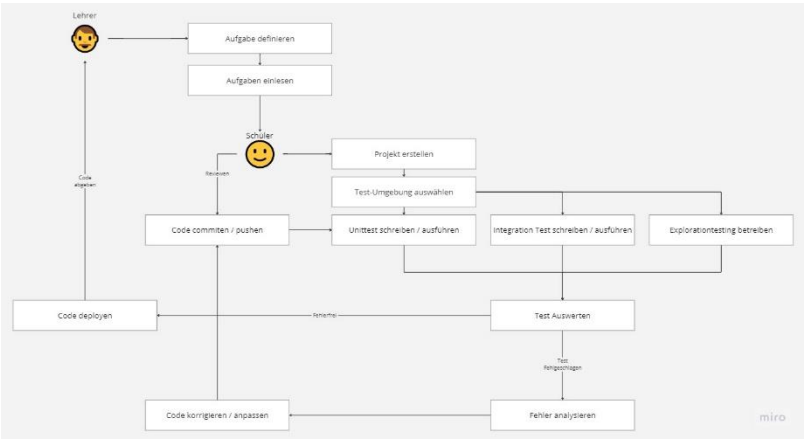
Testkonzept

Zusammenfassung

Wir haben und bei unserem Projekt für Unit-, Exploration- und Component Testing entschieden. Das Explorationtesting wurde von uns ausgeführt, und ohne Protokoll umgesetzt. Beim Unit-Testing haben wir ein Mock-Backend erstellt, um einzelne „units“ zu testen. Das Component-Testing wurde mittels einer Pipeline und einem externen Programm umgesetzt.

Big-Picture

Unser **Big- Picture** umfasst das Erhalten des Auftrages, bis zur Abgabe des getesteten Codes, wobei das Testing im Mittelpunkt steht. Nach Erhalten und Erfassen des Auftrages, erstellten wir eine Testumgebung, welche unsere Applikation und dessen Bestandteile testen kann. Wir haben uns auf die drei testvarianten Exploration-, Component- und Unit-Testing entschieden. Nach erstellen des Codes werden die Tests durchgeführt und das Resultat analysiert. Bei nicht erfolgreichen Testen werden die Fehler analysiert und der Code angepasst und von anderen Teammitglied kontrolliert und genehmigt. Dieser Prozess wurde so lang wiederholt, bis die Tests alle erfolgreich durchkommen. Anschliessend wird der code deployed und kann dem Auftraggeber (Lehrer) präsentiert werden.



Test-Features

Postman (Component Tests):

Getestetes Feature	Nicht getestetes Feature
userService(komplett)	MyEntryListService

JUnit (Unit Tests):

Getestetes Feature	Nicht getestetes Feature
myListEntryController	Post Endpoint
userController	Put Endpoint

Testvorgehen

- **Test Driven Development** (TDD) ist eine agile Entwicklungspraxis, die darauf abzielt, die Qualität und Stabilität von Software zu verbessern. Bei TDD erfolgt die Entwicklung in kleinen, inkrementellen Schritten, die in drei Phasen unterteilt werden:

1. **Red (Rot):** In dieser Phase schreibt der Entwickler zuerst einen Test, der das erwartete Verhalten des Codes beschreibt. Dieser Test schlägt fehl (rot), da der zu testende Code noch nicht existiert.
2. **Green (Grün):** Als Nächstes schreibt der Entwickler den minimalen Code, der erforderlich ist, um den Test zum Bestehen zu bringen. Der Fokus liegt auf der Implementierung des gewünschten Verhaltens.
3. **Refactor (Refaktorisieren):** Nachdem der Test erfolgreich ist, kann der Entwickler den Code refaktorisieren, um ihn sauberer und wartbarer zu gestalten. Der Test stellt sicher, dass dabei keine Funktionalität verloren geht.

Testumgebung

Postman

- **Postman** ist eine leistungsstarke Software für API-Entwicklung und -Tests. Sie ermöglicht das Erstellen, Testen und Dokumentieren von APIs. Teams können mithilfe von Postman effizient zusammenarbeiten und automatisierte Tests erstellen. Die automatische Dokumentation erleichtert anderen Entwicklern die Nutzung der APIs. Postman bietet Integrationen mit anderen Entwicklungs- und DevOps-Tools. Es ermöglicht die Überwachung und Analyse der API-Leistung. Postman ist ein unverzichtbares Tool für Entwickler, um die API-Entwicklung zu optimieren und die Produktivität zu steigern.

JUnit

- **JUnit** ist ein Open-Source-Framework für das Testen von Java-Anwendungen, speziell für Unit-Tests. Es ermöglicht Entwicklern, Testfälle zu definieren, organisieren und automatisiert auszuführen. Annotationen werden verwendet, um Testmethoden und -klassen zu markieren, während Assertions dazu dienen, erwartete Ergebnisse mit den tatsächlichen Ergebnissen abzugleichen. Test Suites helfen bei der Organisation von Testfällen. JUnit kann in Build-Tools wie Maven und Gradle integriert werden. Die neueste Version, JUnit 5, bietet erweiterte Funktionen, ist jedoch rückwärtskompatibel zu JUnit 4. Insgesamt ist JUnit ein unverzichtbares Werkzeug für Java-Entwickler, um Codequalität und -stabilität sicherzustellen. Es fördert bewährte Entwicklungspraktiken und trägt zur Verbesserung von Java-Anwendungen bei.

Planung

Die Planung wurde anhand der Zeit und Kenntnisse der Gruppenmittglieder aufgeteilt und den Stärken zugeordnet. Bei Blockaden oder Fehlermeldungen, welche das Weiterarbeiten blockieren, haben wir uns gemeinsam um eine Lösung bemüht.