
Palantir

Inhalt

Planung

Bei der Planung unseres Projekts standen die spezifischen Anforderungen im Mittelpunkt unseres Interesses. Um dieser Herausforderung gerecht zu werden, haben wir uns für ein bereits existierendes Projekt entschieden, in dem jedoch der Bereich des Testings bisher stark vernachlässigt wurde. Unser Hauptziel bestand darin, das Frontend komplett neu zu gestalten, das Backend entsprechend anzupassen und gleichzeitig eine umfassende Unit-Test-Abdeckung zu implementieren, um die Qualität des Codes sicherzustellen. Darüber hinaus legten wir besonderen Wert darauf, die Testabdeckung nicht nur zu verbessern, sondern auch transparent und nachvollziehbar darzustellen.

Um dieses Ziel zu erreichen, entschieden wir uns für die Integration eines automatisierten Reports in unsere GitHub-Pipeline. Diese Strategie ermöglicht es uns, die Testabdeckung kontinuierlich zu überwachen und sicherzustellen, dass alle Bereiche des Codes ausreichend getestet sind. Die automatisierte Berichterstattung stellt somit ein zentrales Element unserer Qualitätssicherungsstrategie dar, da sie hilft, Schwachstellen im Code schnell zu identifizieren und zu beheben. Dies trägt nicht nur zu einer erhöhten Softwarequalität bei, sondern optimiert auch unsere Arbeitsprozesse, indem es uns ermöglicht, effizienter und zielgerichteter zu arbeiten.

Durch die konsequente Umsetzung dieser Strategie und die kontinuierliche Verbesserung unserer Testprozesse sind wir überzeugt, dass wir die Qualität und Zuverlässigkeit unserer Software maßgeblich steigern und somit den Erfolg unseres Projekts sichern können.

Grobe Architektur

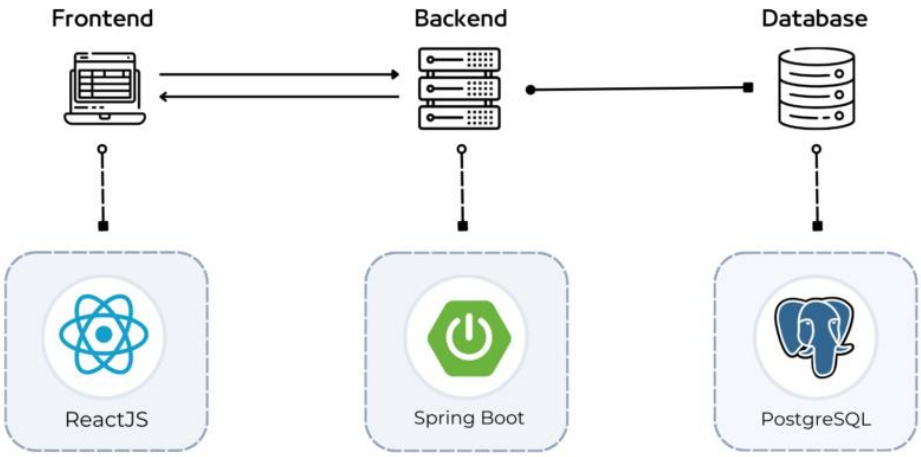
Die Architektur unserer Applikation ist in drei Hauptkomponenten unterteilt: eine Datenbank, ein Frontend und ein Backend. Jedes dieser Elemente spielt eine entscheidende Rolle in der Funktionsweise und Leistungsfähigkeit unserer Anwendung.

Für die Datenhaltung setzen wir auf eine robuste und leistungsfähige Postgres-Datenbank. Diese Datenbanklösung zeichnet sich durch ihre Zuverlässigkeit, Skalierbarkeit und ihre umfangreichen Funktionen aus, die eine effiziente Verwaltung und Abfrage großer Datenmengen ermöglichen. Durch ihre bewährte Stabilität stellt die Postgres-Datenbank das solide Fundament unserer Anwendung dar und garantiert eine sichere und schnelle Datenverarbeitung.

Das Backend unserer Applikation wurde mit Spring Boot entwickelt, einem populären Java-Framework, das die Erstellung von hochleistungsfähigen und skalierbaren Webanwendungen ermöglicht. Durch die Verwendung von JSON Web Tokens (JWT) gewährleisten wir eine sichere Authentifizierung und Autorisierung der Nutzer. Dieser Ansatz stellt sicher, dass sensible Benutzerdaten und -aktionen zuverlässig geschützt sind und trägt somit maßgeblich zur Sicherheit unserer Anwendung bei.

Auf der Client-Seite setzen wir auf React, ein modernes und flexibles JavaScript-Framework, das für die Entwicklung von benutzerfreundlichen und reaktiven Benutzeroberflächen konzipiert wurde. React ermöglicht uns eine effiziente und modulare Gestaltung des Frontends, wodurch wir ein ansprechendes und intuitives Nutzererlebnis schaffen können.

Um die Qualität und Zuverlässigkeit unserer Software kontinuierlich zu überwachen und zu verbessern, nutzen wir die Infrastruktur von GitHub und Pipelines für das Generieren und Analysieren der Testabdeckung. Durch die Automatisierung dieses Prozesses stellen wir sicher, dass jeder Pull Request einer gründlichen Prüfung unterzogen wird und potenzielle Schwachstellen oder Fehler frühzeitig erkannt und behoben werden können. Diese Herangehensweise ermöglicht es uns, ein hohes Maß an Softwarequalität zu gewährleisten und das Vertrauen unserer Nutzer in unsere Anwendung zu stärken.



Zusammenfassend bildet unsere Applikation eine synergetische Verbindung aus einer soliden Datenbank, einem sicheren Backend und einem benutzerfreundlichen Frontend, ergänzt durch eine effektive Nutzung der GitHub-Infrastruktur zur Gewährleistung höchster Softwarequalität.

Testkonzept

Einleitung

Die vorliegende Applikation ist eine interaktive Plattform, auf der sich Nutzer registrieren und ein persönliches Profil erstellen können. Unser Testkonzept konzentriert sich darauf, die fehlerfreie Funktion dieser Kernfunktionalitäten sicherzustellen und ein reibungsloses Benutzererlebnis zu gewährleisten. Es beinhaltet sowohl manuelle als auch automatisierte Testverfahren, um eine umfassende Abdeckung und effiziente Durchführung der Tests zu gewährleisten. Im Folgenden werden die spezifischen Testziele und -methoden, die im Rahmen dieses Projekts zur Anwendung kommen, detailliert dargestellt.

Test Item

Für unser Testkonzept haben wir die wesentlichen Bestandteile der Applikation identifiziert, die einer eingehenden Überprüfung bedürfen. Im Fokus stehen hierbei das Backend und das Frontend.

1. **Backend**: Hierbei handelt es sich um das Herzstück unserer Applikation, in dem die zentralen Geschäftslogiken, Datenverarbeitungsprozesse und Schnittstellenanbindungen implementiert sind. Das Backend spielt eine entscheidende Rolle für die Stabilität und Leistungsfähigkeit der

gesamten Plattform. Aus diesem Grund werden wir umfangreiche Tests durchführen, um sicherzustellen, dass die Server-Logik korrekt funktioniert, die Datenintegrität gewahrt bleibt und alle Endpunkte wie erwartet reagieren.

2. **Frontend**: Das Frontend stellt die Benutzeroberfläche unserer Applikation dar und ist somit für die Interaktion mit dem Nutzer verantwortlich. Wir werden sicherstellen, dass alle Benutzeroberflächenelemente korrekt dargestellt werden, die Navigation intuitiv ist und die User Inputs korrekt verarbeitet und an das Backend weitergeleitet werden. Ebenso werden wir prüfen, ob die Applikation über verschiedene Browser und Geräte hinweg konsistent und benutzerfreundlich funktioniert.

Features to be tested

Im Rahmen unseres Testkonzepts legen wir den Schwerpunkt ausschließlich auf das Backend des Projekts. Hierbei rücken die Funktionen und Dienste, die im Zusammenhang mit der Verwaltung und Abfrage von Benutzerprofilen stehen, in den Vordergrund.

1. **Service-Methoden**: Neben den API-Endpunkten werden auch die dahinterliegenden Service-Methoden getestet. Diese sind entscheidend für die Geschäftslogik und Datenverarbeitung im Backend. Wir stellen sicher, dass diese Methoden wie erwartet funktionieren und robust gegenüber Fehlern und unerwarteten Eingaben sind.
2. **Authentifizierung und Autorisierung**: Ein weiterer kritischer Bereich, den wir prüfen werden, ist die Authentifizierung und Autorisierung von Benutzern. Es ist entscheidend, dass nur autorisierte Benutzer Zugang zu sensiblen Daten und Funktionen erhalten. Daher werden wir sowohl die Mechanismen zur Überprüfung der Benutzeridentität (Authentifizierung) als auch die Verfahren zur Bestimmung ihrer Zugriffsrechte (Autorisierung) eingehend testen.

Durch die detaillierte Überprüfung dieser Features und Funktionen im Backend gewährleisten wir, dass die Plattform sicher, stabil und zuverlässig ist und den Benutzern einen Mehrwert bietet.

Approach

Unser Ansatz zur Qualitätssicherung und zum Testen von Features im Rahmen der Softwareentwicklung ist proaktiv und integriert die Erstellung von Tests als integralen Bestandteil des Entwicklungsprozesses. Jedoch folgen wir dabei nicht strikt der TDD Methodik.

1. **Unit Tests**: Für jedes neue Feature oder jede wesentliche Änderung im Code schreibt der Entwickler begleitende Unit Tests. Diese Tests sind darauf ausgerichtet, die Funktionalität der kleinsten testbaren Teile des Codes (wie Funktionen oder Methoden) isoliert zu überprüfen. Der Fokus liegt auf der Validierung der korrekten Ausführung und dem korrekten Verhalten dieser Code-Segmente unter verschiedenen Bedingungen.
2. **Component Tests**: Neben den Unit Tests erstellen die Entwickler auch Component Tests, um die Interaktionen zwischen verschiedenen Teilen des Systems auf einer höheren Ebene zu überprüfen. Diese Tests zielen darauf ab, die korrekte Funktion von zusammenhängenden Code-Bereichen und deren Schnittstellen sicherzustellen.

3. **Nicht nach TDD**: Obwohl wir die Bedeutung von Tests anerkennen und ihre Erstellung in den Entwicklungsprozess integrieren, folgen wir nicht dem TDD-Ansatz, bei dem Tests vor der eigentlichen Code-Implementierung geschrieben werden. Stattdessen wählen wir einen flexibleren Ansatz, bei dem die Tests parallel oder unmittelbar nach der Code-Entwicklung erstellt werden. Dies ermöglicht es, uns auf die Implementierung der geforderten Funktionalitäten zu konzentrieren und gleichzeitig die Qualität und Zuverlässigkeit des Codes sicherzustellen.

Unser Ziel war es, eine Balance zwischen effizienter Softwareentwicklung und rigoroser Qualitätssicherung zu finden, um eine hochwertige und zuverlässige Anwendung zu liefern. Durch diesen pragmatischen Ansatz zum Testing sind wir in der Lage, die Vorteile des Testens zu nutzen, ohne den Entwicklungsprozess unnötig zu verkomplizieren oder zu verlangsamen.

Test Deliverables

Die Ergebnisse und Materialien, die im Rahmen des Testprozesses erstellt und geliefert werden, sind entscheidend für den Erfolg des Projekts und die Sicherstellung der Softwarequalität. Hierbei nutzen wir eine Kombination aus verschiedenen Tools und Methoden, um eine umfassende Testabdeckung zu gewährleisten:

1. **JUnit für Unit Tests**: JUnit ist ein weit verbreitetes Framework für das Schreiben und Ausführen von Unit Tests in Java-Projekten. Es ermöglicht den Entwicklern, präzise und isolierte Tests für individuelle Komponenten und Methoden zu schreiben, um sicherzustellen, dass sie wie erwartet funktionieren. Die mit JUnit erstellten Unit Tests bilden einen wesentlichen Bestandteil unserer Testdokumentation und tragen maßgeblich zur Qualitätssicherung bei.
2. **Postman für Component Tests**: Für die Durchführung von Component Tests und zur Überprüfung der API-Endpunkte verwenden wir Postman. Dieses Tool erlaubt es uns, HTTP-Anfragen an das Backend zu senden, die Antworten zu analysieren und die korrekte Funktionalität der API zu verifizieren. Postman bietet zudem die Möglichkeit, Tests und Skripte zu schreiben, um automatisierte Tests für die API-Endpunkte durchzuführen.
3. **Postman für Sicherheitstests**: Neben den Component Tests nutzen wir Postman auch, um spezifische Sicherheitstests durchzuführen. Dabei fokussieren wir uns auf die Überprüfung der Authentifizierungs- und Autorisierungsmechanismen, um sicherzustellen, dass nur berechtigte Nutzer Zugang zu sensiblen Funktionen und Daten haben.
4. **Testberichte und -dokumentation**: Alle durchgeführten Tests, ihre Ergebnisse und etwaige gefundene Fehler oder Schwachstellen werden detailliert dokumentiert. Diese Dokumentation dient als wichtige Ressource für die Entwicklung, um Probleme zu beheben, die Qualität der Software kontinuierlich zu verbessern und einen transparenten Überblick über den Zustand der Anwendung zu bieten.

Zusammengefasst stellen JUnit, Postman und die begleitende Testdokumentation die zentralen Liefergegenstände unseres Testprozesses dar, um die Funktionalität, Sicherheit und Zuverlässigkeit der Applikation zu gewährleisten.

Testing Tasks

Unser Testprozess konzentriert sich hauptsächlich auf die Unit- und Component-Teststufen, um die Qualität und Zuverlässigkeit unserer Applikation zu gewährleisten.

Unit Test Stufe:

1. **Isolierte Funktionsprüfung**: Jede Funktion und Methode werden einzeln getestet.
2. **Logiküberprüfung**: Überprüfen aller logischen Pfade und Datenflüsse.
3. **Fehlerbehandlungstests**: Sicherstellen, dass Fehler und Ausnahmen korrekt behandelt werden.
4. **Grenzwerttests**: Testen mit Grenzwerten und ungewöhnlichen Eingaben.

Component Test Stufe:

1. **Integrationstests**: Sicherstellen, dass verschiedene Module und Dienste korrekt zusammenarbeiten.
2. **Sicherheitsüberprüfungen**: Überprüfen der Authentifizierungs- und Autorisierungsmechanismen.
3. **Performance-Tests**: Überwachung der Leistung unter Last.

Durch diese gezielten Testaufgaben in beiden Teststufen stellen wir sicher, dass jede Komponente korrekt funktioniert und das Gesamtsystem den erwarteten Anforderungen entspricht.

Environmental Needs

Für ein effizientes Testing sind folgende Umgebungsbedingungen notwendig:

1. **Laufendes Backend**: Das Backend muss in der Entwicklungsumgebung, vorzugsweise IntelliJ, ausgeführt werden.
2. **Datenbankverbindung**: Eine funktionierende Postgres-Datenbank ist erforderlich, um das Backend zu starten und zu testen.
3. **Unit Tests**: Die Unit Tests können direkt in IntelliJ ausgeführt werden.

Durch dieses Setup stellen wir sicher, dass das Testing reibungslos und effektiv ablaufen kann.

Reflexion TDD & Code Review

TDD

Obwohl TDD eine bewährte Entwicklungsmethode ist, haben wir im Laufe dieses Projekts entschieden, dass sie nicht unseren Vorstellungen entspricht. Die Grundidee von TDD, erst Tests und

dann den dazugehörigen Code zu schreiben, ist zwar schlüssig, wir sehen jedoch die Gefahr eines zu starken Fokus auf das Bestehen der Tests.

Während der Entwicklung entdecken wir oft neue Edge-Cases oder Anforderungen, die ursprünglich nicht bedacht wurden. Bei TDD könnten diese Aspekte leicht übersehen werden, da die Tests bereits im Voraus festgelegt sind. Dies könnte die Robustheit und Flexibilität des Codes beeinträchtigen.

Daher haben wir uns gegen die weitere Verwendung von TDD entschieden und bevorzugen einen flexibleren, iterativen Entwicklungsansatz, der es uns ermöglicht, unsere Praktiken kontinuierlich zu verbessern und einen Code zu erstellen, der nicht nur die Tests besteht, sondern auch zukünftigen Anforderungen gewachsen ist.

Code Review

Code Reviews sind in unserem Projekt essenziell, um die Code-Qualität sicherzustellen und Wissen im Team zu teilen. Durch die Überprüfung des Codes durch Teamkollegen werden Fehler schneller entdeckt und es eröffnen sich Möglichkeiten für Verbesserungen und alternative Lösungsansätze. Diese Praxis fördert zudem den konstruktiven Umgang mit Kritik und trägt zur persönlichen und fachlichen Weiterentwicklung bei. Insgesamt stärken Code Reviews die Konsistenz und Robustheit unseres Codes und sind ein unverzichtbares Instrument in unserem Entwicklungsprozess.

Reflexion Mikka

Persönlich find ich die Planung des Modul Ablaufs sehr schwierig. Ich hätte es bevorzugt, wenn wir entweder nur die Aufgaben oder nur das Projekt gehabt hätten. Die Zeit, die wir für das Projekt hatten, war nicht ausreichend um etwas Abliefern zu können mit dem man zufrieden ist. Der Einzige Grund, warum wir Schluss endlich nun doch etwas haben, dass wir mit Stoltz abgeben können, ist die Zeit, die wir Privat noch aufwendeten. Grundlegend im Projekt bin sehr zufrieden, die Teamdynamik hat in der Vergangenheit schon sehr gut funktioniert und hat es auch dieses Mal wieder. Ich denke wir konnten trotz des Zeitmangels noch etwas abliefern, womit wir zufrieden sein können.

Durch das, dass wir die meisten Themen in diesem Modul schon einmal behandelt haben, oder sogar aktiv im Berufsalltag anwenden, war die Motivation auch etwas geringer als sonst. Man macht sich häufiger die Gedanken, warum man etwas beweisen muss, obwohl man es doch schon kann. Durch das entstehen dann auch minimalistischere Arbeiten und eine generell pragmatischere Ansicht auf das Modul und die dazugehörigen Aufgaben.

Ich verstehe auch noch nicht, warum wir das Projekt präsentieren müssen, wir hatten zu wenig Zeit, um etwas zu machen, das Präsentation würdig ist. Daher finde ich den Präsentations-Teil dieses Moduls doch eher unsinnig in diesem Fall. (Mir ist bewusst das, es das erste Mal ist, daher ist dies alles nur konstruktive Kritik)

Reflexion Cedric

Grundsätzlich waren die kleinen Aufgaben keine schlechte Sache. Natürlich gab es auch da Kritik von uns die wir auch schon geäussert haben. Die Kombination von Projekt und Aufgaben haben wir schon in Vergangenen Modulen gehabt und es kam zu Ähnlichen Ergebnissen, die Zeit für beides ist sehr knapp und es kam dann meist zu Stress im Projekt. Darunter litt die Qualität des Projektes. Es hätte mich noch mehr gewundert, wenn tatsächlich alle Aufgaben erledigt werden sollten statt der fünf die wir jetzt schlussendlich machen mussten plus das Projekt.

Das Thema Testing ist nicht unbedingt das beliebteste. Als ich den Stundenplan gesehen habe mit diesem Modul war ich schon demotiviert dafür. Keineswegs lag dies an der Lehrperson, sondern lediglich am Thema. Eher das Gegenteil, der Unterricht mit SOL lief gut und wir konnten in der eigenen Geschwindigkeit Arbeiten, ohne gross von Inputs unterbrochen zu werden. Ziel war es einfach das Modul hinter mich zu bringen, nicht mehr.