Berichte

Zusammenfassung Refactoring

Wir waren bei der Entwicklung unseres Quellcodes schon immer sehr darauf bedacht, dass wir uns an allgemeine Clean-Code-Regeln orientieren. Ebenfalls ist durch die Entwicklung von HTML und JS mit NEXT eine sehr gute Struktur vorgeschrieben. Durch die Komponentenbauweise ist das Programm in viele einzelne kleinere Dateien unterteilt und es entstehen kaum größere Dateien, bei denen sich, aufgrund unserer Maßnahmen im Vorhinein, etwas nach Clean-Code-Regeln refaktorisieren lässt.

Wir haben jedoch bei der Kontrolle unseres Codes noch veralteten Code gefunden. Dieser diente als Ersatz für die API, die Rezepte bereitstellt. Da diese API jetzt problemlos funktioniert, kann der alte Code entfernt werden, um unnötigen Code zu vermeiden.

Um den Abschluss des Projekts vorzubereiten, haben wir den gesamten Code noch einmal gründlich durchgearbeitet. Dabei haben wir besonderen Wert darauf gelegt, überflüssige Kommentare zu entfernen, die nur während der Entwicklung zur Erklärung von Funktionen dienten. Ebenso haben wir sämtliche Konsolenausgaben, die zum Überprüfen von Funktionen gedacht waren, eliminiert. Diese Überarbeitungen tragen dazu bei, den Code klarer und verständlicher zu gestalten und ihn gemäß den Clean Code Prinzipien weiter zu optimieren.

Zusätzlich haben wir sichergestellt, dass alle Variablen- und Funktionsnamen aussagekräftig und konsistent sind. Dadurch wird der Code leichter wartbar und für andere Entwickler einfacher verständlich. Schließlich haben wir auch auf die Einhaltung der DRY-Prinzipien (Don't Repeat Yourself) geachtet und redundante Codeabschnitte entfernt oder refaktorisiert.

Durch diese Maßnahmen stellen wir sicher, dass unser Projekt nicht nur funktional, sondern auch nachhaltig und zukunftssicher ist. Dies erleichtert die Wartung und Weiterentwicklung und sorgt dafür, dass neue Teammitglieder sich schneller in den Code einarbeiten können.

Review-Protokoll

→ siehe Blogeintrag

Testbericht

Einleitung

Dieser Testbericht gibt einen umfassenden Überblick über den Softwaretestprozess für unser Projekt, das eine Website zur Rezeptsuche entwickelt hat. Der Umfang der Testaktivitäten umfasste die Überprüfung der API-Zugriffsfähigkeit, die Funktionalität der Login- und Registrierungsseiten, die Vorschau der Rezepte und das Abrufen einzelner Rezepte.

Teststrategie

Der Gesamtansatz für das Testen folgte einer systematischen Testmethodik. Aufgrund des zeitlichen Rahmens haben wir uns gegen Unittests entschieden, planen jedoch, diese bei einer längeren Projektlaufzeit zu integrieren. Die derzeitigen Tests konzentrieren sich auf Systemtests, die es uns ermöglichen, Fehlerquellen anhand der Fehlermeldungen einzugrenzen. Unsere Teststrategie umfasste folgende Testarten und -techniken:

1. API-Zugriffstest: Überprüfung, ob die Website auf die API zugreifen kann, um Rezepte und Benutzerdaten aus der Datenbank zu laden.
2. Funktionsprüfung der Login- und Registrierungsseiten: Sicherstellung, dass alle Elemente vorhanden sind und die Funktionalität gewährleistet ist.
3. Test der Rezeptvorschauen: Überprüfung, ob die Vorschauen korrekt angezeigt werden.
4. Test des Abrufens einzelner Rezepte: Überprüfung, ob einzelne Rezepte korrekt abgerufen werden.

Für diese Tests haben wir das Testframework Jest in Verbindung mit Next.js verwendet, um automatisierte Testfälle zu erstellen und durchzuführen. Jest ist ein JavaScript-Testing-Framework, das sich hervorragend für das Testen von Next.js-Anwendungen eignet.

Testplan

Der Testplan beinhaltete spezifische Testaufgaben, Zeitpläne und Ressourcen, um die Testziele zu erreichen:

Testaufgaben:

- Entwicklung und Ausführung von Testfällen für API-Zugriff, Login- und Registrierungsseiten, Rezeptvorschauen und Rezeptabrufe.
- Dokumentation der Testergebnisse und gefundenen Fehler.

Zeitpläne:

- Testvorbereitung: 1 Woche
- Testdurchführung: 2 Wochen
- Fehlerbehebung und Retests: 2 Wochen

Ressourcen:

- Testteam: Mika Keßler
- Testumgebung: Git-Branches
- Tools: Jest, Next.js, Github Actions

Testfälle

Die spezifischen Testfälle wurden wie folgt ausgeführt (aktueller Status):

API-Zugriffstest:

- Status: Bestanden
- Fehler: Die API hatte 2-mal keine Verbindung mehr. Konnte über die Tests identifiziert werden

Login- und Registrierungsseiten:

- Status: Bestanden
- Fehler: Ein kleiner Fehler im Registrierungsformular wurde gefunden und behoben.

Rezeptvorschauen:

- Status: Bestanden
- Fehler: Keine

Abrufen einzelner Rezepte:

- Status: Bestanden
- Fehler: Beim Abrufen eines Rezeptes wurde nicht das korrekt hinterlegte Rezept angezeigt.

Testergebnisse

Die Schweregrade sind von 1-5 definiert. Wobei 1 für geringfügige Beeinträchtigung steht und 5 für eine sehr schwerwiegende Beeinträchtigung des Systems.

Fehler 1 – API-Verbindung nicht erfolgreich:

- Schweregrad: 5

- Fehler: Die API wurde unerwartet gestoppt
- Lösung: Die API musste neugestartet werden

Fehler 2 – Registrierungsformular:

- Schweregrad: 1
- Fehler: Die Anzeigen für fehlerhafte Eingaben hat nicht die richtigen Fehler zurückgegeben
- Lösung: Überprüfung der zurückgesendeten Fehlercodes und Anpassung der angezeigten Fehlermeldungen

Fehler 3 – Rezeptanzeige:

- Schweregrad: 5
- Fehler: Beim Auswählen eines Rezeptes ist man nicht auf das korrekte Rezept weitergeleitet worden
- Lösung: Überprüfung des Codes, wie entschieden wird, welches Rezept gerendert werden soll. Die Speicherstruktur der Rezepte wurde überarbeitet.

Aktuellsten Testresultate:

```
> next@0.1.0 test
> jest

 PASS  __test__/register.test.js
 PASS  __test__/previews.test.js
 PASS  __test__/recipe.test.js
 PASS  __test__/login.test.js
 PASS  __test__/api.test.js

Test Suites: 5 passed, 5 total
Tests:       5 passed, 5 total
Snapshots:   0 total
Time:        2.564 s
Ran all test suites.
```

Metriken

Die folgenden Metriken wurden während des Testprozesses erfasst:

- Anzahl der gefundenen Fehler: 4
- Zeit zur Fehlerbehebung: 5 Stunden
- Erreichte Testabdeckung: 90% der definierten Systemtests

Empfehlungen

Um den Testprozess und die Qualität der Software weiter zu verbessern, werden folgende Empfehlungen gegeben:

Integration von Unittests:

- Bei längerer Projektlaufzeit sollten Unittests integriert werden, um frühzeitig Fehler auf Code-Ebene zu erkennen.

Erweiterung der Testautomatisierung:

- Weitere Testszenarien sollten automatisiert werden, um die Testeffizienz zu steigern.

Schlussfolgerung

Die durchgeführten Tests haben gezeigt, dass die wesentlichen Funktionen der Website zur Rezeptsuche zuverlässig arbeiten. Die gefundenen Fehler waren geringfügig und wurden schnell behoben. Insgesamt ist die Softwarequalität als hoch zu bewerten, und die Website ist bereit für den produktiven Einsatz. Die automatische Ausführung der Tests bei Änderungen im Git-Dev-Branch hat sich als sehr nützlich erwiesen und sollte als Teil des kontinuierlichen Integrationsprozesses beibehalten werden.

Software-Metriken

→ siehe auch: Blogeintrag

Wir haben uns für die Metriken:

- Vollständigkeit
- Fertigstellungsrate
- Komplexität der Schnittstellen

entschieden.

Mit der Metrik *Vollständigkeit* haben wir kontinuierlich, insbesondere aber nach Projektende, gemessen welche Ziele wir abschließen konnten und welche nicht.
Mit der Metrik *Fertigstellungsrate* haben wir die geplanten und gebrauchten Zeiten aller Tickets geprüft und damit auch die Qualität unserer Retrospektiven.
Mit der Metrik *Komplexität der Schnittstellen* konnten wir potenzielle Engpässe und Schwachstellen der API messen.