

Bachelorarbeit

Bjarne Haumann

Evaluierung KI-gestützter Code Review Tools am Beispiel
der Entwicklung einer Alkohol Tracking App

Bjarne Haumann

Evaluierung KI-gestützter Code Review Tools am Beispiel der Entwicklung einer Alkohol Tracking App

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Angewandte Informatik*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr.-Ing. Lars Hamann
Zweitgutachter: Prof. Dr. Stefan Sarstedt

Eingereicht am: 18.11.2025

Bjarne Haumann

Thema der Arbeit

Evaluierung KI-gestützter Code Review Tools am Beispiel der Entwicklung einer Alkohol Tracking App

Stichworte

KI-gestütztes Code Review, Codequalität, Softwareentwicklung, Alkohol Tracking App, React Native, Supabase, Automatisierte Codeanalyse

Kurzzusammenfassung

Diese Arbeit untersucht den Einsatz von KI-gestützten Code Review Tools anhand der Entwicklung einer Alkohol Tracking App. Die App wurde mit React Native und Supabase umgesetzt und dient als realistische Codebasis, anhand derer die Tools Shippie, Copilot und CodeRabbit evaluiert werden. Die Analyse kombiniert quantitative Metriken wie Precision, Recall und F1-Score mit qualitativen Bewertung der Nützlichkeit und Verständlichkeit der generierten Vorschläge. Ergänzend wurde ein Experiment durchgeführt, bei dem gezielt Fehler in die Codebasis eingeführt wurden. Die Ergebnisse zeigen, dass die Tools den Code Review Prozess sinnvoll ergänzen können, jedoch menschliche Expertise nicht vollständig ersetzen.

Bjarne Haumann

Title of Thesis

Evaluation of AI-based code review tools using the example of developing an alcohol tracking app

Keywords

AI-based Code Review, Code Quality, Software Development, Alcohol Tracking App, React Native, Supabase, Automated Code Analysis

Abstract

This thesis examines the use of AI-assisted code review tools based on the development of an alcohol tracking app. The app was built using React Native and Supabase and serves as a realistic codebase for evaluating the tools Shippie, Copilot, and CodeRabbit. The analysis combines quantitative metrics such as precision, recall, and F1-score with a qualitative assessment of the usefulness and clarity of the generated suggestions. Additionally, an experiment was conducted in which errors were intentionally introduced into the codebase. The results show that these tools can meaningfully complement the code review process but cannot fully replace human expertise.

Inhaltsverzeichnis

Abbildungsverzeichnis	vii
Tabellenverzeichnis	viii
Abkürzungen	ix
1 Einleitung	2
1.1 Motivation und Problemstellung	2
1.2 Zielsetzung	3
1.3 Aufbau der Arbeit	4
2 Grundlagen	5
2.1 Theorie und Praxis von Code Reviews	5
2.2 Grundlagen KI-gestütztes Code Review	6
2.3 Technologische Grundlagen der App	7
2.3.1 React Native und Expo	7
2.3.2 Supabase	8
2.4 Blutalkoholkonzentration Berechnung	11
2.4.1 Mathematische Grundlagen	11
2.4.2 Relevanz für die App Funktionalität	12
3 Implementierung der Alkohol Tracking App	14
3.1 Anforderungen und Architektur	14
3.1.1 Anforderungen an die Anwendung	14
3.1.2 Architektur der App	15
3.2 Implementierung der Kern-Features	16
3.2.1 Eingabe des Konsums	16
3.2.2 Echtzeit BAK Berechnung	17
3.2.3 Benachrichtigungen	18

3.3	Entwicklungsprozess	21
3.3.1	Entwicklungsworkflow	21
3.3.2	Integration der Code Review Tools	22
4	Evaluation der KI-gestützten Code Review Tools	23
4.1	Auswahl und Vorstellung der Tools	23
4.1.1	Shippie	23
4.1.2	Copilot	24
4.1.3	CodeRabbit	25
4.2	Evaluierungsmethodik	27
4.3	Aufbau des Evaluierungsexperiments	28
5	Diskussion und Bewertung	30
5.1	Quantitative Ergebnisse	30
5.2	Qualitative Analyse	31
5.3	Ergebnisse des Experiments	32
5.4	Bewertung der Ergebnisse	34
5.4.1	Grenzen und Empfehlungen für die Praxis	35
6	Fazit und Ausblick	36
6.1	Zusammenfassung der Ergebnisse	36
6.2	Ausblick und zukünftige Entwicklung	36
6.2.1	Weiterentwicklung der App	36
6.2.2	Verbesserungspotenzial der KI Tools	37
6.2.3	Zukunft der KI in der Softwareentwicklung	38
	Literaturverzeichnis	39
A	Anhang	43
A.1	Code der Test Anwendung	43
A.2	Beispielausgaben der Tools	44
A.3	Ergebnisse der Pull Requests	55
A.4	Fehler beim Evaluierungsexperiment	58
A.5	Verwendete Hilfsmittel	59
	Selbstständigkeitserklärung	60

Abbildungsverzeichnis

2.1	Beispiel einer React Native Komponente	9
2.2	Supabase anfrage mit der TypeScript SDK	10
2.3	Row Level Security (RLS) Regel für das Selektieren der Tabelle <i>drinks</i> . .	11
3.1	Ordner Struktur für <i>File-Based Routing</i>	16
3.2	Screens der App zur Erstellung und Auswahl von Getränken: (a) Getränk erstellen, (b) Liste mit Getränken zur Auswahl, (c) Übersicht der konsumierten Getränke.	17
3.3	Beispielhafte Benachrichtigungen der App zur Erinnerung an die Konsumerfassung (oben: letzte Benachrichtigung nach 60 Minuten, unten: erste Benachrichtigung nach 30 Minuten)	19
3.4	Implementierung in TypeScript um Benachrichtigungen abzubrechen . . .	19
3.5	Implementierung der Planung der Benachrichtigungen in TypeScript . . .	20
A.1	Beispiel für einen Zusammenfassungs-Kommentar von Shippie	44
A.2	Beispiel für ein Verbesserungsvorschlag von Shippie	45
A.3	Beispiel für einen Zusammenfassungs-Kommentar von Copilot	46
A.4	Beispiel für ein Verbesserungsvorschlag von Copilot	47
A.5	Beispiel für einen Chat-Kommentar von CodeRabbit	48
A.6	Beispiel für ein Learning von CodeRabbit	49
A.7	Beispiel für einen Zusammenfassungs-Kommentar von CodeRabbit	50
A.8	Beispiel für ein Sequenzdiagramm von CodeRabbit	51
A.9	Beispiel für ein Verbesserungsvorschlag von CodeRabbit	52
A.10	Beispiel für ein direkten Vorschlag von CodeRabbit	53
A.11	Korrekte <i>is-valid-barcode</i> Funktion	58
A.12	Korrekte Migration der Getränke mit Barcode in neue Tabelle	58

Tabellenverzeichnis

4.1	Confusion Matrix	27
5.1	Precision Berechnung der einzelnen Tools	30
5.2	Ergebnisse aus dem Experiment	33
5.3	Berechnete Metriken für den 1. Pull Request (PR)	33
5.4	Berechnete Metriken für den 2. PR	34
5.5	Berechnete Metriken für beide PRs zusammengefasst	34
A.1	Vergleich von Shippie, GitHub Copilot und CodeRabbit bei der KI-Code-Analyse Stand: 28.10.2025	54
A.2	Auswertung der Kommentare von CodeRabbit	55
A.3	Auswertung der Kommentare von Copilot	56
A.4	Auswertung der Kommentare von Shippie	57
A.5	Verwendete Hilfsmittel und Werkzeuge	59

Abkürzungen

BaaS Backend-as-a-Service.

BAK Blutalkoholkonzentration.

KI Künstliche Intelligenz.

LLM Large Language Model.

MCP Model Context Protocol.

PR Pull Request.

RLS Row Level Security.

SaaS Software-as-a-Service.

Vorwort

Aus Gründen der besseren Lesbarkeit wird in den folgenden Kapiteln auf die gleichzeitige Verwendung weiblicher, männlicher und diverser Sprachformen verzichtet. Stattdessen wird das generische Maskulinum verwendet. Sämtliche Personenbezeichnungen gelten gleichermaßen für Männer, Frauen und nicht-binäre Personen. In Fällen, in denen explizit auf bestimmte Geschlechter eingegangen wird, erfolgt dies aus sachlichen oder inhaltlichen Gründen und dient der Präzisierung.

1 Einleitung

1.1 Motivation und Problemstellung

Die rasante Entwicklung von Künstliche Intelligenz (KI) in den letzten Jahren hat neue Möglichkeiten für Automatisierungen und qualitative Verbesserungen im Softwareentwicklungsprozess geschaffen. Code Review, als zentraler Bestandteil der Qualitätssicherung, ist traditionell ein zeitaufwendiger und oft manueller Prozess, der stark von der Expertise und Verfügbarkeit der beteiligten Entwickler abhängt. Diese Abhängigkeit erschwert eine kontinuierliche hohe Qualität von Review und führt in der Praxis häufig zu inkonsistenten Ergebnissen [31, 16].

Klassische automatisierte Verfahren wie statische Codeanalyse können syntaktische Fehler und bekannte Anti-Patterns zuverlässig identifizieren, stoßen jedoch bei der Bewertung von kontextspezifischen Aspekten an ihre Grenzen. Hier eröffnet der Einsatz von KI neue Potenziale. KI-gestützte Code Review Tools versprechen nicht nur eine Effizienzsteigerung, sondern auch eine Verbesserung der Konsistenz und Objektivität von Bewertungen. Dennoch ist bislang unklar, in welchem Maß diese Versprechen in der Praxis tatsächlich eingelöst werden können und welche Grenzen solche Systeme aufweisen. Diese Unsicherheit bildet den technischen Hintergrund für die vorliegende Arbeit.

Neben der technischen Motivation dient die Entwicklung einer mobilen Alkohol Tracking App als konkretes Anwendungsbeispiel für die Implementierung und Evaluation verschiedener KI-gestützter Code Review Tools. Die App bietet ein sinnvolles und praxisnahes Szenario, um die Leistungsfähigkeit und die Grenzen der eingesetzten KI Tools im Softwareentwicklungsprozess zu analysieren.

Der gesellschaftliche Hintergrund des Alkoholkonsums als weit verbreitetes Phänomen und dessen gesundheitliche Auswirkungen [30, 11] liefert dabei den Kontext für die Anwendung, ohne dass die App selbst Gegenstand einer detaillierten Untersuchung im Sinne einer eigenen Problemstellung ist.

Die Relevanz der Arbeit ergibt sich dadurch, dass eine technische Fragestellung zur Evaluation von KI-gestützten Code Review Tools mit einem praxisnahen Anwendungsbeispiel verknüpft wird. Damit werden sowohl aktuelle Herausforderungen der Integration von KI in professionelle Arbeitsprozesse, als auch der gezielte Einsatz digitaler Technologien in gesellschaftlich relevanten Kontexten thematisiert.

1.2 Zielsetzung

Das Hauptziel dieser Arbeit besteht darin, die Leistungsfähigkeit und praktische Anwendbarkeit KI-gestützter Code Review Tools systematisch zu evaluieren. Dies erfolgt anhand der Entwicklung einer mobilen Alkohol Tracking App als konkretes Anwendungsbeispiel, deren Entstehungsprozess den Rahmen für die Untersuchung bildet. Im Einzelnen werden folgende Teilziele verfolgt:

- 1. Entwicklung einer funktionsfähigen Alkohol Tracking App:**
Die Implementierung der mobilen Anwendung zur Erfassung und Auswertung des Alkoholkonsums dient als Grundlage für die praktische Evaluation
- 2. Systematische Evaluation ausgewählter KI-gestützter Code Review Tools:**
Verschiedene KI Tools werden hinsichtlich ihrer Fähigkeiten untersucht, Code Review durchzuführen. Die Evaluation konzentriert sich dabei auf Aspekte wie Fehlererkennung, Vorschlagsqualität und die Integration in den Entwicklungsprozess
- 3. Identifikation von Stärken, Schwächen und Grenzen:**
Auf Basis der durchgeführten Evaluation sollen konkrete Stärken und Schwächen der einzelnen Tools herausgearbeitet werden und erläutert werden welches Tool die besten Ergebnisse liefert.

Somit verfolgt die Arbeit einen dualen Ansatz. Durch die App Entwicklung wird ein praxisnahes Szenario geschaffen, welches als Grundlage für das Code Review dient. Der Fokus liegt dabei klar auf der technischen Evaluation der Code Review Tools. Zudem soll am Ende noch eine Empfehlung ausgesprochen werden, welches der analysierten Tools die besten Ergebnisse liefert und ob es sich lohnt dieses einzusetzen.

1.3 Aufbau der Arbeit

In der vorliegenden Arbeit wird eine Alkohol Tracking App entwickelt, welche während des Entwicklungsprozesses mithilfe von KI-gestützten Code Review Tools überwacht und bewertet wird. Dafür werden in Kapitel 2 zunächst die Grundlagen von Code Review erklärt und wie diese im Zusammenhang mit KI aussieht. Darüber hinaus werden noch die technischen Grundlagen für die Ausgewählten Technologien erläutert und auch warum diese ausgewählt wurden. Neben den technischen Grundlagen wird die mathematische Formel für die Blutalkoholkonzentration (BAK) Berechnung aufgestellt, da diese Bestandteil eines zentralen Features der App ist. In Kapitel 3 werden die wichtigsten Features der App erklärt und warum sich genau für diese Features entschieden wurde. Neben den Features wird zudem noch die Architektur der Applikation dargestellt. Außerdem wird der Entwicklungsworkflow und die Integration der Code Review Tools in diesen Entwicklungsprozess erläutert. Damit die verschiedenen Tools untereinander evaluiert werden können, werden diese zu Beginn von 4 vorgestellt. Nach dieser Vorstellung werden die Evaluierungsmethodiken aufgestellt, sowie der Aufbau des Evaluierungsexperiments erklärt. Durch diese beiden Methodiken können die Tools in Kapitel 5 evaluiert und verglichen werden. Hier werden die Stärken und Schwächen der einzelnen Tools herausgearbeitet und die praktische Anwendbarkeit bewertet. Darüber hinaus wird eine Empfehlung für oder gegen die einzelnen Tools ausgesprochen. Das Ende dieser Arbeit bildet das Kapitel 6, in welchem die Ergebnisse der Arbeit zusammengefasst werden. Zudem werden noch Ausblick in die Zukunft gegeben, wie sich die Weiterentwicklung der App gestaltet, sowie das Verbesserungspotenzial für die KI Tools aussieht.

2 Grundlagen

2.1 Theorie und Praxis von Code Reviews

Code Review ist ein etablierter Prozess in der Software Engineering Praxis. Bei diesem Prozess wird der Quellcode systematisch von anderen Entwicklern als dem ursprünglichen Autor untersucht, bevor dieser in die Codebasis integriert wird. Während dieses Prozess stellen andere Entwickler Fragen oder geben Feedback zu dem geschriebenen Code. Der Autor des erstellten PR übernimmt dann das Feedback der anderen Entwickler oder argumentiert warum dieses Feedback in diesem Fall nicht zu treffend ist. Die Hauptgründe für die Durchführung von Code Reviews lassen sich in mehrere zentrale Kategorien einteilen.

Im Mittelpunkt des Code Review Prozesses steht die Qualitätssicherung und Fehlererkennung. Verschiedenen Studien zeigen, dass Code Review effektiv zur Identifikation von Fehlern beitragen [8, 27]. Durch die unabhängige Überprüfung durch andere Entwickler wird sichergestellt, dass der Code frei von funktionalen Fehlern, Sicherheitslücken, Performance Problemen und Verstößen gegen Coding Standards bleibt. Zudem kann durch diese Review die Lesbarkeit und Wartbarkeit des Codes überprüft werden. Dabei wird darauf geachtet, dass der Code verständlich und gut strukturiert ist. Dies ist besonders wichtig, damit sowohl andere Entwickler den geschriebenen Code verstehen aber auch langfristig pflegbar bleibt und darauf aufgebaut werden kann. Neben der Qualitätssicherung ist auch der Wissenstransfer ein wichtiger Bestandteil des Code Reviews. Dadurch ist es möglich, dass weniger Erfahrene Entwickler von Senior Entwicklern lernen und Best Practices verinnerlichen können. Außerdem werden Code Reviews für Wissensaustausch verwendet. Zum einen kennen jetzt auch weitere Entwickler den geschriebenen Code und können Rückfragen zu diesem Stellen. Dadurch lernen mehr Entwickler die Codestellen kennen, was die Abhängigkeit zu einzelnen Entwicklern reduziert. Die gestellten Rückfragen können zudem als Dokumentation zählen, da andere Entwickler im Nachhinein besser Nachvollziehen können warum sich für bestimmte Patterns entschieden wurde.

Auch wenn Code Review viele Positive Seiten mit sich bringt, hat der Prozess auch ein paar Nachteile und Herausforderungen. Besonders der hohe Zeitaufwand wird als häufigst Grund genannt. Code Reviews sind ressourcenintensive und beanspruchen erhebliche Zeit [10, 28] von Entwicklern, die andernfalls für Implementierung von Features genutzt werden könnte. Der Zeitaufwand ist dabei sehr variable und abhängig von Projekt und Umfang des gestellten PR. Im durchschnitt liegt die Wöchentliche Review Zeit von Entwicklern bei ungefähr fünf Stunden [5]. Das entspricht bei einer fünf Tage Woche genau eine Stunde pro Tag. Neben diesem Zeitaufwand ist die Subjektivität und Inkonsistenz beim Code Review ein weiteres Problem. Die Review Kommentare variieren erheblich in ihrer Qualität und Relevanz, da diese nicht immer gleichermaßen nützlich sind. Diese Diskrepanz lässt sich auf verschiedene Gründe zurückführen. Zum einen die unterschiedlichen Kompetenzen der Entwickler, als auch die Vertrautheit der Codebasis spielen dabei ein große Rolle. Entwickler die länger in einer Codebasis arbeiten, kennen mehr von dieser und können somit besser einschätzen ob eine Veränderung eventuelle Seiteneffekte mit sich bringt [14]. Auch die Subjektivität spielt beim Code Review eine Rolle, da diese Subjektivität zu Konflikten zwischen Entwicklern führen kann [15] und somit die Effektivität des Prozesses beeinträchtigen kann. Diese Meinungsverschiedenheiten können sowohl zwischen Reviewer zu Reviewer als auch Reviewer zu Author des PR auftreten.

Code Reviews sind somit eine unverzichtbare Praxis in der Software Entwicklung, welche Signifikante Vorteile in Themen wie Qualitätssicherung und Wissenstransfer mit sich bringt. Dadurch sind Code Reviews aus diesen Prozessen nicht mehr weg zu denken. Dennoch weisen sie einige Stellen auf, welche optimiert werden könnten. Besonders in dem zeitlichen Aufwand als auch der Subjektivität der einzelnen Code Review Kommentaren könnten diese Optimierungen hilfreich sein.

2.2 Grundlagen KI-gestütztes Code Review

KI-gestütztes Code Review funktioniert im Prinzip genauso wie das herkömmliche Code Review. Anstelle anderer Entwickler analysiert hierbei eine KI den geschriebenen Code. Large Language Model (LLM)'s wie GPT-4, Claude Sonnet oder andere spezialisierte Modelle bilden dabei das Rückgrat der KI-gestützten Code Review Systeme. Diese Modelle werden auf einer Anzahl von Repositories trainiert, welche Code Reviews, Bug Reports und Dokumentationen von Best Practices enthalten. Auf Basis dieser Trainingsdaten

sind sie in der Lage, Code zu verstehen und gezielte Kommentare zu hinterlassen. Die KI analysiert den geschriebene Code und versucht funktionale Fehler, Sicherheitslücken, Performance Probleme und Verstöße gegen Coding Standards ausfindig zu machen und erstellt darauf hin einen Kommentar um auf diese Fehler hinzuweisen. Ähnlich wie bei menschlichen Reviewer hängt die Qualität des Reviews stark von den einzelnen Modellen und deren Kontext ab [28]. LLM's werden nur auf Daten zu einem gewissen Zeitpunkt trainiert und haben somit nicht immer die neusten Versionen von Bibliotheken. Besonders die fehlenden Kontext Daten spielen eine große Rolle, da die KI nicht die ganze Codebasis kennt und somit keine übergreifenden Verbesserungsvorschläge treffen kann. Zudem entfällt der Wissensaustausch zwischen Entwicklern, wenn der Code ausschließlich von einer KI analysiert wird. Als Hauptargument für den Einsatz solcher Systeme wird jedoch die Zeitersparnis genannt. Entwickler müssen den Code nicht mehr vollständig selbst analysieren und können sich stattdessen auf die Implementierung neuer Features konzentrieren, was die Produktivität steigert. Außerdem müssen die Entwickler nicht mehr auf Review warten sondern bekommen dieses direkt und haben somit kein Kontext Switch zwischen den Features.

Trotz bestehender Einschränkungen zeigt sich, dass KI-gestütztes Code Review ein großes Potenzial besitzt, den Softwareentwicklungsprozess effizienter zu gestalten und die Qualitätssicherung langfristig zu unterstützen.

2.3 Technologische Grundlagen der App

2.3.1 React Native und Expo

React Native ist ein Open Source Framework von Meta, welches es ermöglicht, mobile Apps mit JavaScript und React zu erstellen. Während bei der nativen App Entwicklung separate Implementierungen für iOS und Android erforderlich sind, wird eine React Native Anwendung nur einmal entwickelt und kann anschließend auf beiden Plattformen eingesetzt werden. Die in React Native verwendeten Komponenten (2.1) werden in native Komponenten übersetzt, wodurch die App den Eindruck einer vollständig nativen Anwendung vermittelt. Durch die einmalige Entwicklung der App, reduziert sich der Entwicklungsaufwand deutlich. Ein weitere Vorteil ist die Möglichkeit, zahlreiche Bibliotheken aus der Webentwicklung auch in React Native zu verwenden. Dadurch profitieren

Entwickler nicht nur von etablierten Best Practices, sondern auch von einer großen Auswahl erprobter Bibliotheken.

Expo ist ein auf React Native basiertes Open Source Framework, welches viele Prozesse vereinfacht. Durch verschiedene Bibliotheken bietet Expo eine erleichterte Entwicklung an. Neben diesen Bibliotheken besitzt Expo zudem noch einen Cloud Service, worüber die App automatisch gebaut und an Nutzer oder Tester verteilt werden kann [1].

Unterschiede zu Flutter

Als alternatives Framework wurde auch Flutter in Betracht gezogen. Flutter ist ein von Google Entwickeltes Framework, welches genau so wie React Native für plattformübergreifende App Entwicklung verwendet werden kann. Im Gegensatz zu React Native verwendet Flutter die Programmiersprache Dart. Anstelle einer Übersetzung von React Komponenten in native Komponenten werden in Flutter sogenannte Widgets direkt auf einem Canvas gerendert. Dadurch hat Flutter häufig eine etwas besser Performance als React Native [19], jedoch sind diese Verbesserungen bei kleinen Apps kaum auffällig. Ein weitere Grund warum sich für React Native entschieden wurde ist, dass JavaScript die am häufigsten verwendet Programmiersprache [23, 24] und dadurch sowohl die Fehler suche im Internet leichter ist als auch LLM besseren Output liefern.

Neben den genannten fachlichen Vorteilen wurde React Native zudem ausgewählt, da die zugrundeliegende Programmiersprache JavaScript meinen Vorkenntnissen aus der Webentwicklung besonders entspricht. Diese vorhandenen Kenntnisse ermöglichen eine effiziente Entwicklung und eine höhere Codequalität.

2.3.2 Supabase

Supabase ist eine Open Source Backend-as-a-Service (BaaS) Lösung, welche als eine Alternative zu Diensten wie Firebase entwickelt wurde. Als Datenbanksystem kommt PostgreSQL zum Einsatz, das aufgrund seiner Erweiterbarkeit zahlreiche Funktionen von Supabase ermöglicht. Zu diesen Funktionen zählen unter anderem Authentifizierung, Echtzeitdatenbanken, serverlose Funktionen und die automatische Bereitstellung einer API. Letzter basiert auf der PostgreSQL-Erweiterung PostgREST [2]. Dadurch entfällt eine separate Backend Entwicklung, da die API direkt in der Applikation genutzt werden kann. Über die bereitgestellte TypeScript SDK werden SQL ähnliche Abfragen gestellt,

```
import { StyleSheet, View, Text, Button }
  from 'react-native';

export function ExampleComponent() {
  return (
    <View style={styles.box}>
      <Text>Das hier ist eine Beispiel Komponente</Text>
      <Button
        title='Beispiel'
        onPress={() => console.log('Hallo Welt!')}
      />
    </View>
  );
}

const styles = StyleSheet.create({
  box: {
    flex: 1,
  },
  text: {
    fontWeight: 'bold',
  },
  button: {
    backgroundColor: 'blue',
    color: 'white',
  },
});
```

Abbildung 2.1: Beispiel einer React Native Komponente

```
import { createClient } from '@supabase/supabase-js';

const supabase = createClient('SUPABASE_URL', 'SUPABASE_KEY');

export const findDrinkByBarcode = async (barcode: string) => {
  const { data, error } = await supabase
    .from('drinks')
    .select('name, alcohol, default_volume, type')
    .eq('barcode', barcode);

  if (error) {
    throw new Error(error.message);
  }

  return data;
};
```

Abbildung 2.2: Supabase anfrage mit der TypeScript SDK

die durch Supabase automatisch sanitisiert werden, um Angriffe wie SQL-Injection zu verhindern. Das in Abbildung 2.2 gezeigte Beispiel demonstriert eine Abfrage der Tabelle *drinks*, bei der Datensätze anhand eines Barcodes gefiltert und nur bestimmte Spalten wie Name, Alkoholgehalt, Typ und Standardmenge zurückgegeben werden. Durch diese einfache Query Sprache, können Features schnell iteriert werden, da kein neuer Endpunkt erstellt werden muss. Sobald eine neue Tabelle in Supabase erstellt wurde, ist diese über PostgREST API erreichbar. Für jede Anfrage wird eine Transaktion erstellt. Durch diese Transaction können Veränderungen von PostgreSQL zurückgerollt werden sollte ein Fehler auftreten. Die Sicherheit der API wird zusätzlich über RLS gewährleistet. RLS-Regeln werden tabellenspezifisch definiert und ermöglichen feingranulare Zugriffskontrolle auf Zeilenebene. RLS bedeutet, dass jede Zeile(Row) einzeln gesichert ist und verschiedene Nutzer nur auf Bestimmte Reihen zugreifen darf. Abbildung 2.3 zeigt eine RLS-Regel für die Tabelle *drinks*, bei der authentifizierte Nutzer nur Lesezugriff auf Datensätze haben, welche von ihnen selbst erstellt wurden oder die keinen Ersteller besitzen [25].

Supabase eignet sich besonders für dieses Projekt, da es eine schnelle und unkomplizierte Umsetzung der benötigten Backend-Funktionalitäten ermöglicht. Die Integration mit React Native über die TypeScript SDK erleichterte den direkten Zugriff auf Daten und

```
create policy "Enable read access for authenticated
    users based on who created"
on "public"."drinks"
as permissive
for select
to authenticated
using (((created_by IS NULL) OR
    (( SELECT auth.uid() AS uid) = created_by)));
```

Abbildung 2.3: RLS Regel für das Selektieren der Tabelle *drinks*

vereinfachte die Kommunikation zwischen App und Datenbank. Durch die automatische Bereitstellung einer API und integrierter Sicherheitsmechanismen wie Authentifizierung und Zugriffskontrolle konnte der Entwicklungsfokus auf die App Funktionalität und dem KI unterstützen Code Reviews gelegt werden.

2.4 Blutalkoholkonzentration Berechnung

2.4.1 Mathematische Grundlagen

Für die Berechnung des BAK-Wertes existieren verschiedene Ansätze, die größtenteils auf der von Widmark (1932) entwickelten Formel basieren. Diese liefert eine Näherung des theoretisch maximalen Blutalkoholgehalts im Körper:

$$C_0 = \frac{A}{r \cdot W} \quad (2.1)$$

C_0 bezeichnet den theoretisch maximalen BAK-Wert in Milligramm Ethanol pro Gramm Blut (mg/g), was numerisch einem Wert in Promille entspricht. A steht für die konsumierte Alkoholmenge in Gramm, W für das Körpergewicht in Kilogramm und r ist der Verteilungsfaktor, der die Verteilung des Ethanols im Körper beschreibt. Der r Wert wurde von Widmark als Konstante festgelegt und beträgt für Frauen 0,55 und für Männer 0.68 [22]. Die Berechnung von Widmark basiert auf der Annahme, dass sich der Alkohol gleichmäßig im Körper verteilt und berücksichtigt nicht individuelle Unterschiede in Metabolismus und Hydratation [29]. Um die BAK für einen bestimmten Zeitpunkt t zu

bestimmen, erweiterte Widmark die Formel wie folgt:

$$C_t = \frac{A}{r \cdot W} - \beta \cdot t \quad (2.2)$$

β beschreibt die stündliche Abbaurrate von Ethanol im Blut, welche typischerweise zwischen 0,1 und 0,2 mg/g/h liegt, was 0,1 - 0,2 ‰/h entspricht [22]. Der Wert hängt von individuellen Faktoren wie Stoffwechsel, Nahrungsaufnahme und Geschlecht ab. t steht für die vergangene Zeit in Stunden seit dem Konsum des letzten alkoholischen Getränkes. Zur Ermittlung der konsumierten Alkoholmenge A wird die folgende Formel verwendet:

$$A = V \cdot ABV \cdot 0,789 \quad (2.3)$$

Dabei steht V für das Getränkvolumen in Millilitern, ABV für den Alkoholgehalt in Volumenprozent. Der Faktor 0,789 ist die Dichte von reinem Ethanol in Gramm pro Milliliter [21].

Neuere Modelle, wie etwa die Seidl-Formel (2.4), berücksichtigen Körpergröße und Gewicht, um den Verteilungsfaktor r individuell zu bestimmen:

$$\begin{aligned} r_M &= 0,31608 - 0,004821W + 0,004632H \\ r_F &= 0,31223 - 0,006446W + 0,004466H \end{aligned} \quad (2.4)$$

Auch in dieser Formel steht das W für das Gewicht der Person in Kilogramm. H ist in diesem Fall die Größe der Person in Zentimetern. r_M ist der berechnete Verteilungsfaktor für Männer und r_F entsprechend für Frauen. Vergleichsstudien [22, 17] zeigen, dass diese Modelle tendenziell präzisere BAK-Werte liefern als die klassische Berechnung von Widmark mit fixen Konstantwerten.

2.4.2 Relevanz für die App Funktionalität

Die Blutalkoholberechnung stellt eine zentrale Grundlage für die implementierten Funktionen der Alkohol-Tracking-App dar. Für die Anwendung ist insbesondere eine praxisnahe, aber konservative Schätzung der BAK relevant.

Die Applikation verwendet dafür die Seidl-Formel, da sie trotz einer leichten Überschätzung des BAK-Wertes das Risiko von gefährlichen Unterschätzungen reduziert[17]. Dies unterstützt die angestrebte Zielsetzung, Nutzerinnen und Nutzer zu einem verantwortungsvolleren Trinkverhalten anzuregen. Die Berechnung erfolgt dabei dynamisch auf

Basis der von den Nutzern eingegebenen Parameter (Geschlecht, Gewicht, Größe und konsumierte Getränke). Somit bildet die BAK Berechnung sowohl ein domänenspezifisches Fundament als auch den funktionalen Kern der App.

3 Implementierung der Alkohol Tracking App

3.1 Anforderungen und Architektur

3.1.1 Anforderungen an die Anwendung

Das primäre Ziel der App Entwicklung basiert darauf, eine funktionsfähige Alkohol Tracking App zu entwickeln, welche als Grundlage für die Evaluation der Code Review Tools dient. Dabei liegt der Fokus auf der Implementierung zentraler Features, die ausreichend Komplexität für eine aussagekräftige Bewertung der Tools bietet. Die funktionalen Anforderungen umfassen die in Kapitel 3.2 ausführlich beschriebenen Kernfunktionalitäten. Neben den funktionalen Anforderungen werden auch nicht funktionale Anforderungen definiert:

- Codequalität: Der Code soll gut strukturiert und wartbar sein um eine sinnvolle Evaluation zu ermöglichen
- Plattformübergreifende Funktionalität: Die App soll sowohl auf iOS als auch auf Android lauffähig sein
- Textabdeckung: Kritische Funktionen müssen durch automatisierte Tests abgesichert sein und in einer CI/CD Pipeline automatisch laufen

Diese Anforderungen bilden die Grundlage für die technische Umsetzung und stellen sicher, dass die entwickelte App sowohl als vollständiges Produkt fungiert als auch ausreichend Code für eine umfassend Evaluation der Code Review Tools bereitstellt.

3.1.2 Architektur der App

Die Architektur der App folgt einer modernen Client-Server Architektur, bei der React Native als Frontend Framework und Supabase als Backend Lösung zum Einsatz kommen. Die Kommunikation der zwischen React Native und Supabase erfolgt über die bereitgestellte TypeScript SDK (siehe Abbildung 2.2). Um klare Trennung zwischen der Präsentationslogik und der Datenzugriffslogik zu schaffen, wurde in der App eine Service Schicht implementiert. Diese Services kapseln die Interaktion mit der Supabase API und stellen einheitliche Schnittstellen für die React Native Komponenten bereit. Durch diese Abstraktion wird der Code wartbarer und testbarer, da die Services unabhängig von den UI-Komponenten entwickelt und getestet werden können. Zudem kann die Implementierung ausgetauscht werden, sollte in Zukunft eine andere Backend Lösung an stelle von Supabase gewählt werden. Die Dateistruktur der App folgt bewährten Konventionen aus der React Native Entwicklung und ist so gestaltet, dass sie eine klar Trennung zwischen verschiedenen Verantwortlichkeitsbereichen ermöglicht.

- **api:** Fachliche Services, Interfaces und Modelle der API
- **app:** UI-Logik, Screens und Navigation
- **components:** Wiederverwendbare UI-Elemente, geordnet nach Domänen
- **lib:** Utilities wie Berechnungen, Queries und Modelle
- **store:** Redux-Logik für lokalen Zustand
- **supabase:** Datenbank Migrationen Daten Schema

Die in dem *app* Ordner platzierten Dateien werden für die Navigation innerhalb der App verwendet. Diese Strukturierung wird *File-Based Routing* genannt und wird durch die Bibliothek *expo-router* bereitgestellt [9]. Die in Abbildung 3.1 gezeigte Ordner Struktur würde den Routen */*, */drinks* und */drinks/session* entsprechen.

Durch diese Bibliothek wird somit bereits festgelegt wie die Ordner Struktur für die einzelnen Screens auszusehen hat.

```
app/  
|-- index.tsx  
|-- drinks/  
|   |-- index.tsx  
|   |-- session.tsx
```

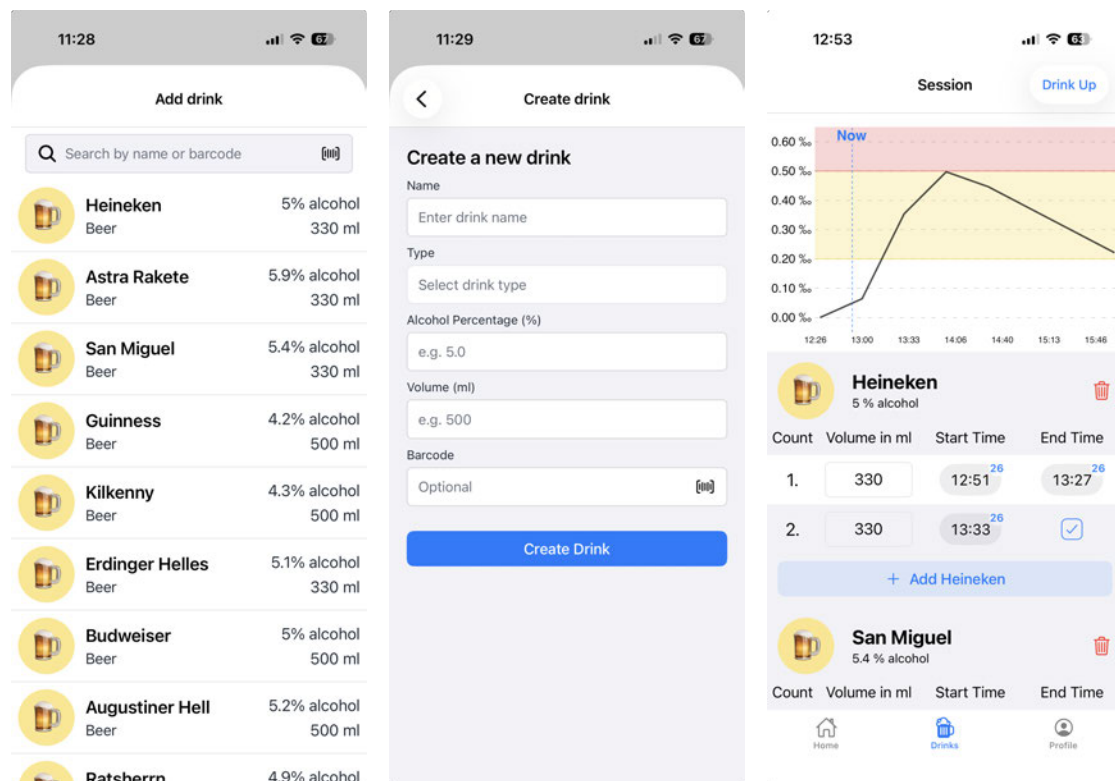
Abbildung 3.1: Ordner Struktur für *File-Based Routing*

3.2 Implementierung der Kern-Features

3.2.1 Eingabe des Konsums

Für eine lückenlose Erfassung des Konsums benötigt die Tracking-App eine benutzerfreundliche und intuitive Oberfläche. Die Auswahl des Getränkes erfolgt über eine List aus Getränken, welche aus der Datenbank geladen wird. In dieser Liste sind bereits der Alkoholgehalt und das Volumen der Getränke hinterlegt. Durch diese Vorauswahl wird die Eingabe für den Nutzer deutlich vereinfacht und beschleunigt (siehe Abbildung 3.2a). Abweichende Mengen lassen sich manuell anpassen. Zusätzlich ermöglicht die integrierte Suchfunktion ein schnelleres Auffinden des gewünschten Getränkes. Der Nutzer kann entweder nach dem Namen des Getränkes suchen oder den Barcode des Getränkes scannen. Sollte ein Getränk nicht auffindbar sein, besteht die Möglichkeit, dieses manuell mit Namen, Volumen und Alkoholgehalt einzupflegen (siehe Abbildung 3.2b).

Wird ein Getränk an einem Abend mehrmals konsumiert, kann der Nutzer dieses Getränk durch Betätigung eines Buttons unter dem jeweiligen Eintrag erneut hinzufügen. Die Beendigung des Konsums eines Getränkes erfolgt durch das Anklicken einer Checkbox, womit der Zeitraum des Konsums dokumentiert wird (siehe Abbildung 3.2c). Diese Eingabe lässt sich im Nachhinein durch ein Zeitpicker anpassen, falls der Nutzer das Getränk zu einem anderen Zeitpunkt konsumiert hat. Diese einfache und benutzerfreundliche Eingabemöglichkeit stellt sicher, dass der Nutzer den Konsum vollständig erfassen kann, ohne dabei viel Aufwand betreiben zu müssen. Die erfassten Daten dienen als Grundlage für die BAK Berechnung und ermöglichen eine genauere Analyse des Konsumverhaltens.



(a) Liste mit Getränken zur Auswahl

(b) Getränk erstellen

(c) Übersicht der konsumierten Getränke

Abbildung 3.2: Screens der App zur Erstellung und Auswahl von Getränken: (a) Getränk erstellen, (b) Liste mit Getränken zur Auswahl, (c) Übersicht der konsumierten Getränke.

3.2.2 Echtzeit BAK Berechnung

Um dem Nutzer ein besseres Verständnis seines Alkoholkonsums zu vermitteln, wird der BAK in Echtzeit berechnet und grafisch dargestellt (siehe Abbildung 3.2c). Die Berechnung folgt der in 2.4 beschriebenen Methodik und berücksichtigt dabei die individuellen Eingaben zu Körpergewicht, Größe, Geschlecht und konsumierten Getränke. Auf Grundlage dieser Werte werden aktuelle Messpunkte sowie kurzfristige Prognosen des BAK berechnet, sodass der zeitliche Verlauf für den Nutzer nachvollziehbar bleibt.

Die Ergebnisse erscheinen in einem interaktiven Graphen, der historische Werte, den aktuellen Stand sowie Prognosen visualisiert. Der Graph ist entlang der y-Achse in drei Bereiche unterteilt, die dem Nutzer eine schnelle Einschätzung seines aktuellen BAK ermöglichen:

- 0,0‰ bis 0,2‰ (farblos): unbedenklicher Bereich
- 0,2‰ bis 0,5‰ (gelb): moderater Bereich, Vorsicht empfohlen
- 0,5‰ und mehr (rot): kritischer Bereich, Konsumverhalten sollte überdacht werden

Diese Kombination aus Echtzeitberechnung und Visualisierung unterstützt den Nutzer dabei, Konsummuster zu erkennen und seine Selbststeuerung zu stärken. Durch das unmittelbare Erkennen von Spitzenwerten, das Verständnis zeitlicher Zusammenhänge und die Einschätzung der Erholungsphase können fundierte Entscheidungen getroffen und gegebenenfalls der Konsum reduziert werden.

3.2.3 Benachrichtigungen

Eine weitere Kernfunktion der App sind die Benachrichtigungen, die den Nutzer während des Konsums unterstützen sollen. Dafür wird er regelmäßig daran erinnert seinen Konsum zu erfassen. Dadurch soll sichergestellt werden, dass der Nutzer seinen Konsum vollständig dokumentiert und nicht über den Abend vergisst, weitere Getränke hinzuzufügen. Verschiedene Studien [3, 26] zeigen, dass gezielte Benachrichtigungen Nutzer dazu motivieren können, die Anwendung aktiver zu nutzen und somit eine höhere Datenqualität erreicht wird.

Die Benachrichtigungen werden automatisiert ausgelöst, wenn der Nutzer seit mindestens 30 Minuten keine Aktion in der App durchgeführt hat. Sollten innerhalb der 30 Minuten eine Aktion durchgeführt worden sein, wird der Timer zurückgesetzt (siehe Abbildung 3.4). Die erste Benachrichtigung erfolgt nach 30 Minuten Inaktivität. Diese weist den Nutzer freundlich darauf hin, seinen Konsum zu erfassen. Falls der Nutzer weiterhin inaktive bleibt, wird eine weitere Benachrichtigung nach 15 Minuten gesendet. In dieser wird der Nutzer danach gefragt, ob er weiterhin Getränke konsumiert oder ob er den Konsum beenden möchte. Nach weiteren 15 Minuten Inaktivität wird eine letzte Benachrichtigung gesendet, die den Nutzer zum einen darauf aufmerksam macht, dass diese die letzte Erinnerung ist und zum anderen die Sitzung zu beenden oder weitere Getränke hinzuzufügen. Somit werden nach 30, 45 und 60 Minuten Inaktivität Benachrichtigungen gesendet (siehe Abbildung 3.3). Diese Benachrichtigungen können vom Nutzer in den Einstellungen der App deaktiviert werden, falls dieser dies wünscht.

Die Implementierung der Benachrichtigungen erfolgt lokal über das Package *expo-notifications*. Mit diesem Package können die Push-Benachrichtigungen zu festgelegten Zeitpunkten gesendet und wieder entfernt werden.



Abbildung 3.3: Beispielhafte Benachrichtigungen der App zur Erinnerung an die Konsumerfassung (oben: letzte Benachrichtigung nach 60 Minuten, unten: erste Benachrichtigung nach 30 Minuten)

```
export async function cancelSessionReminderNotifications() {
  await Promise.all(
    REMINDER_NOTIFICATIONS.map(async ({ id }) => {
      await Notifications.cancelScheduledNotificationAsync(id);
    })
  );
}
```

Abbildung 3.4: Implementierung in TypeScript um Benachrichtigungen abzubrechen

```
export async function scheduleSessionReminderNotifications() {
  const currentBadgeCount = await Notifications.getBadgeCountAsync();
  await Promise.all(
    REMINDER_NOTIFICATIONS.map(async (notification, index) => {
      await Notifications.scheduleNotificationAsync({
        identifier: notification.id,
        content: {
          title: notification.title,
          body: notification.body,
          sound: true,
          badge: currentBadgeCount + index + 1,
          data: {
            url: '/drinks/session',
          },
        },
        trigger: {
          type: Notifications.SchedulableTriggerInputTypes.TIME_INTERVAL,
          seconds: minutesToSeconds(notification.scheduledInMinutes),
        },
      });
    })
  );
}
```

Abbildung 3.5: Implementierung der Planung der Benachrichtigungen in TypeScript

3.3 Entwicklungsprozess

3.3.1 Entwicklungsworkflow

Die Entwicklung der App erfolgt nach einem agilen Softwareentwicklungsprozess, um ein strukturiertes und nachvollziehbares Arbeiten zu gewährleisten. Als Projektmanagement- und Issue-Tracking-Tool wurde Linear¹ eingesetzt. In den dort angelegten Tickets wurden Aufgaben klar definiert, indem zu jedem Feature eine Beschreibung sowie entsprechende Akzeptanzkriterien formuliert wurden. Die Tickets wurden mithilfe eines Kanban Boards organisiert, um den Fortschritt zu visualisieren. Es wurde ein ticketbasiertes Arbeiten gewählt, da diese Arbeitsweise auch in der professionellen Softwareentwicklung gängige Praxis ist. Somit können auch die verschiedenen KI Tools, beim Code Review darauf achten, ob die vorher definierten Akzeptanzkriterien im Ticket erfüllt wurden oder nicht. Als Branching-Strategie wurde das von Driessen vorgeschlagene Gitflow-Modell verwendet, das insbesondere in Kombination mit einer ticketbasierten Entwicklung deutliche Vorteile hinsichtlich Nachvollziehbarkeit und Struktur bietet. Jeder Branch wurde nach der jeweiligen Ticket-ID benannt (z.B. **DRINK-42**), wodurch eine eindeutige Zuordnung zwischen Ticket und Branch gewährleistet war. Auch die Commit-Messages wurden mit der Ticket-ID versehen, um die Zugehörigkeit der commits zu einem Ticket sicherzustellen. Für jedes beendete Feature wurde ein Pull Request erstellt. Im Zuge des Merge-Prozesses mussten verschiedene Pre-Checks erfolgreich durchlaufen werden, um den Qualitätsstandard der Anwendung zu sichern. Unter anderem wurde Prettier verwendet um den Code einheitlich zu formatieren. Neben Prettier wurde auch ESLint als statisches Code Analyse Tool verwendet. Durch diese statische Analyse konnte bereits häufige Fehler verhindert werden. Diese Tools werden auch im professionellen Raum verwendet, damit bei dem Code Review nicht auf solche Fehler geachtet werden müssen, sondern der Reviewer sich auf die wesentlichen Teile fokussieren kann. Zusätzlich war das erfolgreiche Ausführen der Test-Suite Voraussetzung für das Mergen eines PR in den Hauptbranch, um funktionale Regressionen frühzeitig abzufangen. Sollten alle Tests erfolgreich durchlaufen oder das Code Review keine weiteren Fehler aufweisen, können die Änderungen in den Hauptbranch integriert werden. Damit wurde eine kontinuierliche Sicherung der Codequalität über den gesamten Entwicklungsprozess hinweg erreicht.

¹<https://linear.app>

3.3.2 Integration der Code Review Tools

Sobald ein PR erstellt wird, kann der Entwickler an die verschiedenen automatisierten Code Review Tools eine Review Anfrage stellen. Diese Tools untersuchen den im PR veränderten Code auf Fehler, Sicherheitslücken und Qualitätsmängel und geben dem Entwickler Feedback zu den Änderungen. Außerdem wird für andere manuelle Code Reviewer hilfreiche Kommentare und Kontextinformationen hinterlassen, wodurch die Effizienz und Qualität des manuellen Reviews verbessert werden können. Bei einer Änderung an dem Pull Request wird automatisch ein erneutes Review gestartet, sodass der Code kontinuierliche auf neue Fehler untersucht wird. Wie genau die in dieser Arbeit betrachteten Tools funktionieren und welche Funktionen sie im Detail bieten, wird im Kapitel 4.1 näher dargestellt.

4 Evaluation der KI-gestützten Code Review Tools

4.1 Auswahl und Vorstellung der Tools

Um zu untersuchen, inwieweit die Tools den Code Review Prozess unterstützen können, wurden die folgenden drei KI-gestützten Code Review Tools ausgewählt. Die Tools decken verschiedene Tiefen der Integration und Funktionalität ab. Bei allen Tools wird automatisch ein Code Review durchgeführt, sobald ein PR geöffnet wird.

4.1.1 Shippie

Shippie¹ ist ein Typescript Open Source Projekt, das eine GitHub Action bereitstellt, um verschiedene LLMs über deren jeweilige APIs anzusprechen [4]. Hierbei wird auf das Konzept des "*Bring your own key*" gesetzt. Entwickler können selbst entscheiden, welchen LLM-Provider und welches Modell sie verwenden möchten. Zudem ist kein Abonnement erforderlich, sondern es wird *per usage* bezahlt. Die Kosten steigen somit proportional zur Häufigkeit der Nutzung der GitHub Action.

Für die Bachelorarbeit wurde sich für das LLM Modell 4.1-mini von OpenAI entschieden. Neben dem relativ großen Kontextfenster von 200.000 Token pro Minute². Des Weiteren ist es im Vergleich zu anderen Modellen kostengünstiger, aber auch ein wenig leistungsfähiger. Pro Millionen Input Token kostet das Modell 0,40USD und pro Millionen Output Token 1,60USD [20].

Da es sich um ein Open Source Projekt handelt, lässt sich der Quellcode unkompliziert an spezifische Anwendungsfälle anpassen. So kann auch der Prompt, der für die KI verwendet wird, angepasst werden. Der Prompt kann auch als Argument in der GitHub Action

¹Wurde von *Code Review GPT* umbenannt, hat aber die gleiche Funktionalität beibehalten

²Da immer eine Request pro PR gemacht wird, muss ein relativ großes Tokenkontingent da sein

übergeben werden. Neben der Anpassung des Base Prompts, bietet Shippie außerdem noch die Möglichkeit, sogenannte *Rules* in der Code Base zu hinterlegen [4].

Rules sind vordefinierte Richtlinien oder Qualitätsanforderungen, die festlegen, welche Aspekte des Codes besonders überprüft werden sollen ³. Sie dienen als strukturiertes Regelwerk, das den Prompt gezielt erweitert und dem LLM zusätzliche Kontextinformationen liefert. Auf diese Weise können Entwickler bestimmte Standards (z.B. Namenskonventionen oder Code-Architektur) festlegen, ohne sie direkt im Prompt zu formulieren [7]. Regeln werden in separaten Dateien im Projekt hinterlegt und beim Ausführen der Action automatisch in den Kontext eingefügt. Shippie stellt hierfür den Pfad `.shippie/*` bereit, damit verschiedenen *Rules* für verschiedene Tools getrennt werden können. Zu den eben genannten *Rules* können Entwickler auch sogenannte Model Context Protocol (MCP)-Server hinterlegen. MCP ist ein aufkommender Standard, der den strukturierten Datenaustausch zwischen LLM und externen Datenquellen ermöglicht. Über MCP können somit LLM's mit externen Systemen kommunizieren, um zusätzliche Informationen abzurufen, wie etwa interne Projektmetriken, Dokumentation oder andere relevante Daten [18]. Dadurch erweitert MCP den Kontext der Analyse und ermöglicht fundierte und projektspezifische Bewertungen.

Als Code Review hinterlässt das Tool einen Kommentar mit einer Zusammenfassung der Änderungen (A.1). In diesem Kommentar wird zudem bewertet, wie sich die Änderungen auf das Projekt auswirken könnten. Diese Bewertung wird mit einer Nummer von 1 bis 5 versehen, wobei je höher die Nummer ist, desto höher ist auch die Wahrscheinlichkeit, dass diese Änderung zu einem Bug führen könnte. Neben dem Zusammenfassungskommentar werden an den jeweiligen Dateien auch Kommentare und Code Verbesserungsvorschläge (A.2) hinterlassen. Wenn das Tool keine Probleme findet, werden entweder keine Kommentare hinterlassen, oder es wird explizit darauf hingewiesen, dass keine Probleme gefunden wurden.

4.1.2 Copilot

GitHub Copilot ist ein von Microsoft entwickeltes Tool, das ursprünglich für Code Completions verwendet wurde. Inzwischen wurde die Funktionalität auch auf Code Reviews erweitert. Durch die enge Integration in die GitHub-Plattform, die sowohl das Versionskontrollsystem als auch die Code Review Funktionalität bereitstellt, bietet es eine

³Beispiele für diese Rules lassen sich unter <https://cursor.directory/> finden

nahtlose Benutzererfahrung. Auch hier können zur Anpassung des Verhaltens *Rules* hinterlegt werden. Diese sind in diesem Fall im Ordner `.github/instructions` zu speichern und müssen mit `instructions.md` enden. Zusätzlich besteht die Möglichkeit, globale Coding Guidelines zu definieren. Diese Funktion ist jedoch ausschließlich im Copilot Enterprise Plan verfügbar und richtet sich somit primär an Unternehmen. Im Gegensatz zu Shippie basiert Copilot auf einem Abonnement basierten Lizenzmodell, deren Nutzungskosten 10USD pro Monat betragen. Neben der Review Funktionalität erhalten Nutzer auch Zugriff auf die Code Completion Funktionalität im Code Editor Visual Studio Code.

Ähnlich wie Shippie fügt auch GitHub Copilot einen Kommentar hinzu, in dem der gesamte PR zusammengefasst wird (A.3). Außerdem wird eine Zusammenfassung pro Datei erstellt. Darüber hinaus werden kontextbezogene Kommentare mit konkreten Verbesserungsvorschlägen direkt an den betroffenen Dateien hinterlassen. Über den *Commit Suggestion* Button können diese Vorschläge direkt in der GitHub Oberfläche angenommen und comitted werden (A.4). GitHub Copilot bietet zum Zeitpunkt der Erstellung dieser Arbeit keine Möglichkeit, MCP-Server zu integrieren [12, 13].

4.1.3 CodeRabbit

CodeRabbit ist ein Software-as-a-Service (SaaS)-Tool, welches für verschiedene Versionskontrollsysteme wie GitHub, GitLab und Bitbucket verfügbar ist. Wie bei Copilot basiert die Nutzung von CodeRabbit auf einem Abonnementmodell, wobei der Service für Open-Source-Projekte kostenlos angeboten wird. Neben der kostenlosen Version existieren zwei kostenpflichtige Modelle. Die Lite-Version zum Preis von 15USD monatlich und die Pro-Version für 30USD monatlich. Für diese Bachelorarbeit kam die Pro-Version zum Einsatz, weshalb sich im folgenden nur auf diese Version bezogen wird.

Ein besonders Feature, welches sowohl Copilot als auch Shippie nicht besitzen ist der *CodeRabbit Chat* ermöglicht es den Entwicklern direkt in dem PR mit der KI zu interagieren. Dies ermöglicht es, Rückfragen zu Kommentaren zu stellen oder spezifische Verbesserungsvorschläge zu erfragen. Außerdem ist es möglich allgemeine Fragen zur Codebase zu stellen, um ein besseres Verständnis zu erlangen (A.5).

Ähnlich wie die anderen beiden Tools unterstützt auch CodeRabbit MCP-Server, um den Kontext für das Code Review zu erweitern. Ein weiteres Feature von CodeRabbit ist die Websuche. Dadurch hat CodeRabbit die Möglichkeit das Internet zu durchsuchen um auf mögliche Fragen von Entwicklern einzugehen oder auch Kommentare von

ihnen zu überprüfen. Das Feature *Knowledge Base* setzt sich aus drei kleineren Features zusammen:

- **Code Guidelines**

Die Code Guidelines können ähnlich wie bei den anderen Tools in Form von *Rules* hinterlegt werden, damit die KI diese beim Code Review Prozess berücksichtigen kann. Standardmäßig sucht CodeRabbit bereits standardisierte Pfade wie etwa *.github/instructions* ab. Es können aber auch eigene Pfade in den Einstellungen hinterlegt werden oder die Standardpfade deaktiviert werden.

- **Issue Tracking**

Dieses Feature ermöglicht die Integration von Projektmanagement-Tools wie Jira, Linear oder GitHub Issues. Durch diese Integration kann CodeRabbit direkt die Tickets mit dem Code Review Prozess verknüpfen und darauf achten, dass alle Änderungen den Anforderungen im Ticket entsprechen.

- **Adaptive Learnings**

Bei den *Adaptive Learnings* handelt es sich um ein Feature, welches es CodeRabbit ermöglicht, aus dem Feedback der Entwickler zu lernen. Dafür können Entwickler ganz einfach auf einen Kommentar von CodeRabbit antworten und das Feedback wird automatisch erkannt und gespeichert. Sollt das Feedback bereits existieren und es sich geändert haben, wird dieses *Learning* automatisch aktualisiert. Entwickler können zudem direkt über die Weboberfläche von CodeRabbit spezifische *Learnings* hinzufügen, bearbeiten oder löschen (A.6).

Genauso wie die anderen Tools hinterlässt CodeRabbit eine Zusammenfassung des gesamten PRs. Diese Zusammenfassung wird allerdings nicht wie bei den anderen Tools als Kommentar hinzugefügt, sondern es wird direkt die Beschreibung des PRs angepasst (A.7). Neben dieser Zusammenfassung wird außerdem noch ein Kommentar hinzugefügt, welcher für die einzelnen Dateien oder Datei Gruppen eine Zusammenfassung erstellt. In diesem Kommentar werden ebenfalls Sequenzdiagramme generiert, um die Abläufe im Code besser zu veranschaulichen (A.8). Es wird ein weiterer Kommentar erstellt, in welchem *Nitpicks* zusammengefasst werden. *Nitpicks* sind kleinere Verbesserungsvorschläge, die keinen direkten Einfluss auf die Funktionalität des Codes haben, aber zur Verbesserung der Codequalität beitragen können. Neben dem *Nitpick* Kommentar werden auch an den einzelnen Dateien Kommentare hinzugefügt, die spezifische Verbesserungsvorschläge enthalten. In diesen Kommentaren wird erklärt, warum die vorgeschlagene Änderung sinnvoll ist. Außerdem kann ähnlich wie bei Copilot der *Commit Suggestion* Button

verwendet werden, um die vorgeschlagenen Änderungen direkt zu übernehmen. Des weiteren wird ein Text generiert, welcher einem AI Agent als Prompt übergeben werden könnte, damit dieser die vorgeschlagenen Änderungen durchführen kann [6] (Abb. A.9 und A.10).

Vergleich der Tools

Alle drei Tools haben somit unterschiedliche Ansätze und Funktionalitäten, um den Code Review Prozess zu unterstützen. Shippie bietet eine flexible und kosteneffiziente Lösung für Entwickler, die verschiedene LLMs nutzen möchten. GitHub Copilot überzeugt durch seine nahtlose Integration in die GitHub-Plattform und die einfache Bedienung. CodeRabbit hingegen bietet eine umfassende Suite von Funktionen, die über den reinen Review hinausgehen, einschließlich direkter Interaktion über den Chat und erweiterter Kontextintegration durch MCP und Websuche. Ein Vergleich der drei Tools ist in Tabelle (A.1) dargestellt.

4.2 Evaluierungsmethodik

In Kapitel 3.3.2 wurde beschrieben wie die Tools in den Entwicklungsprozess integriert wurden. Damit die Tools untereinander evaluiert werden können, werden verschiedene Metriken festgelegt. Für die Durchführung dieser Analyse wird die Confusion in Abbildung 4.1 aufgestellt. Diese Confusion Matrix bildet die Grundlage der quantitativen Analyse, welche eine strukturierte Zuordnung der Vorschläge in die jeweiligen Kategorien zuordnet. Außerdem dient die Matrix als Basis für die Berechnung der Precision

Tabelle 4.1: Confusion Matrix

	Vorhergesagt: Fehler	Vorhergesagt: Kein Fehler
Tatsächlich: Fehler	<i>True Positive (TP)</i> Tool markiert Fehler und es ist ein Fehler	<i>False Negative (FN)</i> Tool markiert keinen Fehler und es ist ein Fehler
Tatsächlich: Kein Fehler	<i>False Positive (FP)</i> Tool markiert Fehler und es ist kein Fehler	<i>True Negative (TN)</i> Tool markiert keinen Fehler und es ist kein Fehler

Metrik.

$$Precision = \frac{TP}{TP + FP} \quad (4.1)$$

Die Precision beschreibt das Verhältnis von korrekter und Tatsächlich umgesetzter Vorschläge (TP) zu allen als relevant eingestuften Vorschlägen ($TP + FP$).

Neben dieser quantitativen Analyse wird zudem die Anwendbarkeit und Qualität der Vorschläge qualitative bewertet. Die Anwendbarkeit bezieht sich dabei auf die Praktikabilität und Umsetzbarkeit der Vorschläge im Kontext der Codebasis. Zusätzlich soll bewertet werden, wie gut die Kommentare formuliert sind. Dabei soll insbesondere darauf geachtet werden, ob die Kommentare lediglich auf einen Fehler hinweisen oder auch direkt erläutert werden. Die qualitative Bewertung erfolgt durch subjektive Einschätzung auf Basis der Erfahrung während des Code Review Prozesses und ergänzt die qualitative Ergebnisse.

Diese Methodik gewährleistet eine umfassende und wissenschaftlich fundierte Evaluation der KI-gestützten Code Review Tools und ermöglicht eine differenzierte Beurteilung ihrer Stärken und Schwächen.

4.3 Aufbau des Evaluierungsexperiments

Neben den genannten Metriken, welche während des gesamten Entwicklungsprozesses erfasst wurden, wird zudem noch ein Evaluierungsexperiment aufgestellt. Dafür wurden zwei PR's im Projekt erstellt, welche absichtlich verschiedene Fehler enthielten. Die Fehler in den einzelnen PR sehen wie folgt aus:

1. DRINK-42 Barcodescanner und filtern nach Barcode
 - a) Fehlendes `\in is-valid-barcode` Regex (siehe Abbildung A.11)
 - b) Fehlende Supabase RLS
 - c) Kein Index auf Barcode Spalte
 - d) `use-search-drinks` sucht nach Namen, wenn Barcode valide ist
 - e) `BarcodeScannerModal` zeigt keine Fehler Komponente an, wenn der User die Kamera Nutzung ablehnt

- f) Daten Migration der Barcodes sucht nach Getränken, welche **keine** Barcodes hat (siehe Abbildung A.12)

2. DRINK-49 Daten als Json Exportieren

- a) Datei wird in Ordner *Document* abgelegt, sollte aber in Cache Ordner gespeichert werden
- b) Datei wird nach export nicht automatisch gelöscht
- c) Es werden alle Sessions für den Export geladen und nicht nur die vom User angeforderten
- d) *handleClose* ruft nicht das *AbortSignal* auf um den API Aufruf abzubrechen
- e) Hintergrundfarbe ist dauerhaft weiß und ist somit nicht Dark Mode kompatibel
- f) Es wurden Funktionen aus *expo-filesystem/legacy* verwendet

Diese Fehler zeigen typische Herausforderungen im Entwicklungsprozess und ermöglichen eine differenzierte Bewertung der Stärken und Schwächen der Tools. Bei den Fehlern in PR 2 wurde besonders darauf geachtet die neuesten Bibliotheken zu verwenden um zu schauen in wie weit diese von den KI erkannt werden können. Auf Basis der in Tabelle 4.1 definierten Matrix wurde neben der Precision zusätzlich der Recall sowie F1-Score berechnet, um eine umfassendere Bewertung der Ergebnisse zu ermöglichen.

$$Recall = \frac{TP}{TP + FN} \quad (4.2)$$

Der Recall beschreibt das Verhältnis der korrekt erkannten Fehler zu allen tatsächlichen vorhandenen Fehlern.

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \quad (4.3)$$

Der F1-Score kombiniert Precision und Recall zu einem harmonischen Mittel und liefert somit eine balancierte Kennzahl für die Gesamtleistung der Tools. Die Berechnung dieser Metriken erfolgt für beide Pull Requests getrennt, um mögliche Unterschiede in der Fehlererkennungsleistung bei unterschiedlichen Code Änderungen zu identifizieren.

5 Diskussion und Bewertung

5.1 Quantitative Ergebnisse

Die quantitative Analyse der generierten Kommentare zeigt deutliche Unterschiede in der Leistungsfähigkeit der drei untersuchten Tools. Insgesamt wurden 24 PR's ausgewertet, wobei die Anzahl der erzeugten Kommentare sowie deren Korrektheit zur Bewertung herangezogen wurde (siehe Tabellen A.2 bis A.4).

CodeRabbit lieferte mit 96 Anmerkungen die höchste Anzahl an Kommentaren. Davon wurden 56 als korrekt und 40 als fehlerhaft bewertet. Diese Werte entsprechen einer Precision von 0,58 (siehe Tabelle 5.1) und zeigt, dass CodeRabbit in mehr als der Hälfte der Fälle korrekte Vorschläge gemacht hat. Copilot erzeugte insgesamt nur 28 Kommentare, was ungefähr ein Drittel der Kommentare von CodeRabbit entspricht. Von diesen wurden 17 als korrekt und 11 als fehlerhaft eingestuft. Dies führt zu einer Precision von 0,61, was darauf hinweist, dass Copilot trotz der geringeren Anzahl an Kommentaren eine etwas höhere Genauigkeit aufweist als CodeRabbit. Shippie hatte mit 12 Kommentaren die geringste Anzahl an generierten Anmerkungen. Das entspricht nicht mal einem Kommentar pro erstellten PR. Zwar hat Shippie zu jedem PR einen Kommentar abgegeben, allerdings hat dieser nur darauf hingewiesen, dass keine Fehler gefunden wurden. Genauso wie Shippie hatten auch Copilot und CodeRabbit einige PR's, bei denen keine Kommentare generiert wurden. Von den 12 Kommentaren von Shippie wurden 3 als korrekt und 9 als fehlerhaft bewertet, was eine Precision von 0,25, was die geringste Genauigkeit unter den drei Tools darstellt.

Tool	Precision
CodeRabbit	0,58
Copilot	0,61
Shippie	0,25

Tabelle 5.1: Precision Berechnung der einzelnen Tools

Die Ergebnisse zeigen, dass die untersuchten Tools sich deutlich in ihrem Verhalten unterscheiden. CodeRabbit generiert eine große Menge an Kommentaren, was auf eine umfassend, aber weniger selektive Analyse hinweist. Diese hohe Aktivität führt zu mehr Feedback, allerdings nimmt die Genauigkeit dabei ab. Copilot hingegen kommentiert deutlich weniger, wodurch die Präzision steigt. Shippie fällt dagegen durch eine insgesamt geringe Aktivität auf. Auch die wenigen korrekten Kommentare sind nicht ausreichend, um eine nennenswerte Unterstützung im Code Review Prozess zu bieten.

5.2 Qualitative Analyse

Auch im Rahmen der qualitativen Analyse unterscheiden sich die drei untersuchten Tools deutlich hinsichtlich der Verständlichkeit, Nützlichkeit und Anwendbarkeit der generierten Kommentare.

Von den Tools zeigt CodeRabbit die höchste Qualität in der Erklärungstiefe und Anwendbarkeit der Vorschläge. Besonders durch die klar strukturierten Kommentare, welche nicht nur auf den Fehler hinweisen, sondern auch durch die Bereitstellung von *Committable suggestions* (siehe Abb. A.9). Diese Vorschläge können direkt in den Code übernommen werden, was den Review Prozess erheblich vereinfacht. Neben diesen direkten Vorschlägen fügt CodeRabbit auch einen Prompt für KI Agenten hinzu, welcher direkt in KI Editoren wie Cursor oder VSCode verwendet werden können. Diese Funktionalität ist auch besser bekannt als *Meta Prompting*, in welchem die KI einen ausführlichen Prompt erstellt, welche daraufhin von anderen KI Modellen verwendet werden kann [32] um spezifische Aufgaben zu lösen (siehe Abb. A.10). Das einzigartige Feature des Chats in CodeRabbit ermöglicht es den Nutzern, weiterführende Fragen zu den generierten Kommentaren zu stellen. Dadurch können Entwickler ein tieferes Verständnis für die vorgeschlagenen Änderungen gewinnen und diese besser in ihren Code integrieren. Durch diesen Chat wird zudem das adaptive Learning von CodeRabbit unterstützt, wodurch die Vorschläge im Laufe der Zeit immer präziser und relevanter werden. Auch das erstellte Sequenzdiagramm bietet eine gute visuelle Unterstützung, um den Kontext der vorgeschlagenen Änderungen besser zu verstehen (siehe Abb. A.8).

Auch Copilot liefert in der qualitativen Betrachtung überzeugende Ergebnisse. Die generierten Kommentare sind klar formuliert bieten allerdings im Vergleich zu CodeRabbit häufig weniger ausführliche Erklärungen. Genauso wie CodeRabbit bietet auch Copilot

direkte Verbesserungsvorschläge an, welche direkt über die Github Oberfläche übernommen werden können (siehe Abb. A.4).

Shippie schneidet im Vergleich der drei Tools deutlich schwächer ab. Das Tool liefert meist nur knappe Beschreibungen der Fehler. Zwar werden in diesem Tool auch Verbesserungsvorschläge gemacht, diese sind jedoch weniger detailliert und bieten keine direkte Möglichkeit zur Übernahme in den Code (siehe Abb. A.2). Dadurch fällt der Mehrwert für den eigentlichen Entwicklungsprozess deutlich geringer aus.

Zusammenfassend zeigt die qualitative Analyse, dass CodeRabbit durch Interaktivität und den adaptiven Lernmechanismus besonders überzeugt. Copilot bietet eine solide Leistung mit klaren, wenn auch weniger ausführlichen Kommentaren. Shippie hingegen liefert die geringste Qualität in Bezug auf Verständlichkeit und Anwendbarkeit der Vorschläge.

5.3 Ergebnisse des Experiments

Die Auswertung des Experiments (siehe Tabellen 5.2 bis 5.5) zeigt deutlich die Unterschiede in der Fehlererkennungsleistung der verschiedenen KI-gestützten Code Review Tools.

Bei der Auswertung des 1. PR (siehe Tabelle 5.3) konnte CodeRabbit 5 der 6 eingeführten Fehler korrekt identifizieren, was zu einem hohen Recall von $\frac{5}{6}$ und einem F1-Score von ungefähr 0,91 führte. Copilot erkannte nur 3 der Fehler, was einen Recall von 0,5 und einen F1-Score von etwa 0,66 zur Folge hatte. Shippie identifizierte lediglich 2 der Fehler, was zu einem Recall von etwa 0,33 und einem F1-Score von 0,5 führte. Im 2. PR (siehe Tabelle 5.4) zeigte CodeRabbit erneut die beste Leistung, indem es 2 der 6 eingeführten Fehler erkannte. Dies führte zu einem Recall von etwa 0,33 und einem F1-Score von 0,5. Sowohl Copilot als auch Shippie konnten in diesem PR keine der eingeführten Fehler erkennen, was zu einem Recall und F1-Score von 0 führte. Bei beiden PR's hatten alle Tools eine Precision von 1, was bedeutet, dass alle von den Tools gemeldeten Fehler tatsächlich Fehler waren und somit in diesem Experiment keine False Positives (FP) auftraten. Jedoch konnten Copilot und Shippie im zweiten PR keine Fehler erkennen, weshalb die Precision hier nicht berechnet werden konnte. Die schlechtere Leistung der Tools im 2. PR lässt sich damit erklären, dass in diesem PR moderne Bibliotheken verwendet wurden, die möglicherweise nicht im Trainingsdatensatz der KI Modelle enthalten

Fehler	CodeRabbit	Copilot	Shippie
1a	✓	✓	✓
1b	✓		
1c			
1d	✓	✓	✓
1e	✓		
1f	✓	✓	
2a			
2b			
2c			
2d			
2e	✓		
2f	✓		
Gesamt:	7 / 12	3 / 12	2 / 12

Tabelle 5.2: Ergebnisse aus dem Experiment

Tool	TP	FP	FN	Precision	Recall	F1
CodeRabbit	5	0	1	1	$\frac{5}{6}$	$\frac{10}{11} \approx 0,91$
Copilot	3	0	3	1	0,5	$\frac{2}{3} \approx 0,66$
Shippie	2	0	4	1	$\frac{1}{3} \approx 0,33$	0,5

Tabelle 5.3: Berechnete Metriken für den 1. PR

waren. CodeRabbit konnte dennoch einige der Fehler erkennen, was auf eine überlegende Fehlererkennungsfähigkeit hinweist.

Die zusammengeführten Metriken für beide PR's (siehe Tabelle 5.5) zeigen, dass CodeRabbit den höchsten Recall von etwa 0,58 und einen F1-Score von ungefähr 0,736 erreichte. Copilot erzielte einen Recall von etwa 0,42 und einen F1-Score von 0,4, während Shippie mit einem Recall von etwa 0,16 und einem F1-Score von etwa 0,285 die geringste Leistung zeigte. Durch diese Ergebnisse wird deutlich, dass CodeRabbit die beste Leistung in diesem Experiment erbacht hat und somit als das effektivste Tool zur Unterstützung im Code Review Prozess hervorgeht.

Tool	TP	FP	FN	Precision	Recall	F1
CodeRabbit	2	0	4	1	$\frac{1}{3}$	0,5
Copilot	0	0	6	/	0	0
Shippie	0	0	6	/	0	0

Tabelle 5.4: Berechnete Metriken für den 2. PR

Tool	TP	FP	FN	Precision	Recall	F1
CodeRabbit	7	0	5	1	$\frac{7}{12} \approx 0,58$	$\frac{14}{19} \approx 0,736$
Copilot	3	0	9	1	$\frac{5}{12} \approx 0,42$	0,4
Shippie	2	0	10	1	$\frac{2}{12} \approx 0,16$	$\frac{2}{7} \approx 0,285$

Tabelle 5.5: Berechnete Metriken für beide PRs zusammengefasst

5.4 Bewertung der Ergebnisse

Im Rahmen der durchgeführten Evaluation hat CodeRabbit im direkten Vergleich der drei Tools die beste Leistung erbracht. Das Tool überzeugte besonders durch die hohe Anzahl an generierten Kommentaren, von denen mehr als die Hälfte als korrekt bewertet wurden. Dies unterstreicht die umfassende Analysestärke des Tools, auch wenn die Precision in der quantitativen Analyse leicht hinter der von Copilot zurückblieb. Jedoch konnte CodeRabbit in dem Experiment insgesamt die meisten Fehler erkennen, was es zu einer wertvollen Unterstützung im Code Review Prozess macht. Auch die qualitative Analyse zeigte, dass die Kommentare von CodeRabbit besonders verständlich und nützlich waren. Die zusätzlichen Funktionalitäten, wie der integrierte Chat, sowie die adaptiven Lernmechanismen, bieten einen erheblichen Mehrwert für Entwickler. Durch die Chat Funktion können Entwickler durch direkte Rückfragen ein tieferes Verständnis für die vorgeschlagenen Änderungen gewinnen, was den Review Prozess erheblich erleichtert. Die adaptiven Lernmechanismen tragen dazu bei, dass die Vorschläge im Laufe der Zeit immer präziser und relevanter werden.

Copilot lieferte ebenfalls solide Ergebnisse, wenn auch mit einer geringeren Anzahl an generierten Kommentaren. Die Precision von Copilot war in der quantitativen Analyse

etwas höher als die von CodeRabbit, jedoch fehlte die Tiefe der Analyse und die Möglichkeit zur detaillierten Interaktion. Auch die Kommentare von Copilot zeigen sich zwar klar und umsetzungsorientiert, fallen aber inhaltlich kürzer aus.

Shippie konnte in diese Evaluation nicht überzeugen. Insbesondere durch die geringe Anzahl an generierten Kommentaren und deren niedrige Korrektheit bietet das Tool nur begrenzten Mehrwert im Code Review Prozess. Dennoch ist Shippie als Open Source Tool interessant, da Entwickler die Möglichkeit haben, die Funktionsweise des Tools selbst anzupassen. Außerdem bietet es eine gute Grundlage mit verschiedenen KI Modellen zu experimentieren. Für die Praxis ist es jedoch ratsam, auf Tools wie CodeRabbit oder Copilot zurückzugreifen, die eine höhere Leistungsfähigkeit und bessere Unterstützung bieten.

5.4.1 Grenzen und Empfehlungen für die Praxis

Insgesamt zeigt sich, dass KI-gestützte Code Review Tools ein großes Potenzial besitzen, den Code Review Prozess zu unterstützen und zu verbessern. Dennoch ist festzuhalten, dass diese Tools derzeit noch nicht in der Lage sind das menschliche Code Review vollständig zu ersetzen. Besonders komplexe, kontextspezifische Fehler, die tiefgreifendes System oder Architekturverständnis erfordern, werden weiterhin besser von erfahrenen Entwicklern erkannt und gelöst. Auch der fehlende Wissenstransfer, der durch das menschliche Review stattfindet, ist ein wichtiger Aspekt, der durch die Nutzung von KI Tools nicht abgedeckt wird. Daher wird ein hybrider Ansatz empfohlen, welcher die Tools als unterstützende Werkzeuge im Code Review Prozess einsetzt. Durch die Kombination von menschlichem Fachwissen und der Effizienz der KI-Tools kann sowohl die Effizienz gesteigert als auch eine höhere Konsistenz im Review Prozess erzielt werden.

6 Fazit und Ausblick

6.1 Zusammenfassung der Ergebnisse

Diese Arbeit zeigt, dass KI-gestützte Code Review Tools bereits eine wertvolle Unterstützung im Softwareentwicklungsprozess bieten können. Die Evaluation der drei Tools verdeutlicht, dass jedes Tools unterschiedliche Stärken und Schwächen aufweist. Dennoch ersetzen sie menschliche Expertise nicht vollständig, weshalb eine Kombination aus menschlichem Review und KI-Unterstützung empfohlen wird.

Die entwickelte Alkohol Tracking App stellte ein geeignetes Anwendungsbeispiel dar und zeigte, dass moderne Technologien wie React Native und Supabase eine effiziente plattformübergreifende Entwicklung ermöglichen. Insgesamt liefert diese Arbeit wichtige Erkenntnisse über die praktische Einsetzung von KI-gestützten Code Review Tools und den Grenzen ihrer aktuellen Leistungsfähigkeit.

6.2 Ausblick und zukünftige Entwicklung

6.2.1 Weiterentwicklung der App

Die durchgeführte Entwicklung der App zeigt großes Potenzial für zukünftige Verbesserungen und Erweiterungen. Die aktuelle Version stellt bereits eine solide Grundlage dar, auf der aufgebaut werden kann, um neue Funktionalitäten zu implementieren.

Ein zentraler Schwerpunkt für die Weiterentwicklung sollte auf der Datenanalyse liegen. Diese würde dem Nutzer eine bessere Möglichkeit bieten sein Trinkverhalten zu verstehen. Dazu könnten Visualisierungen bereitgestellt werden, die darstellen, wie sich das individuelle Konsumverhalten über Wochen oder Monate verändert hat. Zudem könnte der Nutzer jeden Monat eine Zusammenfassung erhalten, in der Zeitpunkt und Menge

des konsumierten Alkohols aufgeführt werden, um den eigenen Konsum regelmäßig zu reflektieren.

Auch auf technischer Ebene können Optimierungen getroffen werden. Der Fokus auf einer Offline First Architektur könnte dazu beisteuern, dass der Nutzer Aktiver seinen Konsum erfasst. Durch die Offline First Architektur könnte der Nutzer die App auch verwenden, sollte er über keinen Internetzugang haben. Dies ist besonders wichtig bei großen Events, bei denen die Netzabdeckung häufig eingeschränkt ist.

Auch die Implementierung von Sozialen Features könnte in Betracht gezogen werden. Eine optionale Funktionalität zum Teilen der Konsummuster mit vertrauten Personen könnte motivierend wirken und soziale Unterstützung fördern. Während eines Konsumanlasses könnten Mitmenschen so auf einen potenziell riskanten Konsum aufmerksam machen. Darüber hinaus könnten Nutzer ihre persönliche Erfahrung oder Strategien zur Reduktion des Alkoholkonsums in einer Community-Funktion teilen. Ein solcher Erfahrungsaustausch würde nicht nur anderen Nutzern helfen, sondern auch die Adhärenz und Motivation Einzelner erhöhen. Insgesamt könnten diese sozialen Features die App von einer reinen Tracking App zu einer umfassenden Plattform für Prävention und gegenseitige Unterstützung transformieren.

6.2.2 Verbesserungspotenzial der KI Tools

Auch bei den Verschiedenen Tools wird sich einiges Verändern. Besonders die Anbindung verschiedener MCP kann den Kontext der LLM deutlich verbessern. Ein weiteres Verbesserungspotenzial ist zudem das Context Window der LLM. Durch die Erweiterung dieses Fenster, können die Tools mehr von der Code Base verstehen und somit auch besser Vorschläge machen. Dadurch könnte auch das Problem bei neueren Versionen von Frameworks oder Bibliotheken reduziert werden, da die Dokumentation mit in das Context Window geladen werden kann. Auch hier könnten die Tools verbessert werden, indem sie neben den Rules auch Dokumentationen als Kontext verwenden. Diese könnten entweder mithilfe von MCP integriert werden oder bei CodeRabbit direkt über die Weboberfläche hinzugefügt werden. Auch werden die LLM immer leistungsfähiger und können komplexere Zusammenhänge verstehen. Zudem werden die Modelle immer wieder mit neuen Daten trainiert, was die Qualität der Vorschläge weiter verbessert. Besonders das Tool CodeRabbit zeigt besonders Potenzial. Durch das einsetzen der Adaptiven Learnings (4.1.3) kann das Tool ganz einfach mit der Code Base wachsen.

6.2.3 Zukunft der KI in der Softwareentwicklung

Die Integration von KI in den Softwareentwicklungsprozess wird in den kommenden Jahren weiter zunehmen. Derzeit liegt der Fokus vieler KI Anwendungen auf der Code Generierung, während das Code Review noch meist manuell erfolgt. Auch weitere Bereiche der Softwareentwicklung, werden zunehmend von KI unterstützt. Dazu gehören beispielsweise die Anforderungsanalyse, das Testen und Designsysteme. Dennoch wird das menschliche Fachwissen weiterhin eine zentrale Rolle spielen. Komplexe, kontextspezifische Entscheidungen erfordern nach wie vor das Verständnis und die Erfahrung von Entwicklern.

Insgesamt ist zu erwarten, dass KI in der Softwareentwicklung eine immer zentralere Rolle einnehmen wird. Während derzeit der Schwerpunkt auf der Code Generierung liegt, könnte in Zukunft eine umfassendere Unterstützung des gesamten Softwareentwicklungsprozesses erfolgen, welchen den Entwicklungsprozess effizienter und qualitativ hochwertiger gestaltet.

Literaturverzeichnis

- [1] *Expo Documentation* — *docs.expo.dev*. <https://docs.expo.dev/>. – [Letzter Zugriff: 05.11.2025]
- [2] *PostgREST Documentation* — *docs.postgrest.org*. <https://docs.postgrest.org/en/v14/>. – [Letzter Zugriff: 31.10.2025]
- [3] BIDARGADDI, Niranjana ; ALMIRALL, Daniel ; MURPHY, Susan ; NAHUM-SHANI, Inbal ; KOVALCIK, Michael ; PITUCH, Timothy ; MAAIEH, Haitham ; STRECHER, Victor: To Prompt or Not to Prompt? A Microrandomized Trial of Time-Varying Push Notifications to Increase Proximal Engagement With a Mobile Health App. In: *JMIR Mhealth Uhealth* 6 (2018), Nov, Nr. 11, S. e10123. – URL <http://mhealth.jmir.org/2018/11/e10123/>. – ISSN 2291-5222
- [4] CAREY, Matt: *shippie: extendable code review and QA agent*. <https://github.com/mattzcarey/shippie>. – [Letzter Zugriff: 31.07.2025]
- [5] CIHAN, Umut ; HARATIAN, Vahid ; İÇÖZ, Arda ; GÜL, Mert K. ; DEVRAN Ömercan ; BAYENDUR, Emircan F. ; UÇAR, Baykal M. ; TÜZÜN, Eray: *Automated Code Review In Practice*. 2024. – URL <https://arxiv.org/abs/2412.18531>
- [6] CODERABBIT: *Introduction - CodeRabbit* — *docs.coderabbit.ai*. <https://docs.coderabbit.ai/overview/introduction>. – [Letzter Zugriff: 25.10.2025]
- [7] CURSOR: *Rules | Cursor Docs* — *cursor.com*. <https://cursor.com/docs/context/rules>. – [Letzter Zugriff: 24.10.2025]
- [8] EISTY, Nasir U. ; CARVER, Jeffrey C.: Developers perception of peer code review in research software development. In: *Empirical Software Engineering* 27 (2021), Nr. 1, S. 13. – URL <https://doi.org/10.1007/s10664-021-10053-x>. ISBN 1573-7616

- [9] EXPO: *Core concepts of file-based routing in Expo Router* — *docs.expo.dev*. <https://docs.expo.dev/router/basics/core-concepts/>. – [Letzter Zugriff: 10.11.2025]
- [10] FROEMMGEN, Alexander ; AUSTIN, Jacob ; CHOY, Peter ; GHELANI, Nimesh ; KHARATYAN, Lera ; SURITA, Gabriela ; KHRAPKO, Elena ; LAMBLIN, Pascal ; MANZAGOL, Pierre-Antoine ; REVAJ, Marcus ; TABACHNYK, Maxim ; TARLOW, Daniel ; VILLELA, Kevin ; ZHENG, Daniel ; CHANDRA, Satish ; MANIATIS, Petros: Resolving Code Review Comments with Machine Learning. In: *Proceedings of the 46th International Conference on Software Engineering: Software Engineering in Practice*. New York, NY, USA : Association for Computing Machinery, 2024 (ICSE-SEIP '24), S. 204–215. – URL <https://doi.org/10.1145/3639477.3639746>. – ISBN 9798400705014
- [11] GARRISSON, Harriet ; SCHOLEY, Andrew ; OGDEN, Edward ; BENSON, Sarah: The effects of alcohol intoxication on cognitive functions critical for driving: A systematic review. In: *Accident Analysis & Prevention* 154 (2021), S. 106052. – URL <https://www.sciencedirect.com/science/article/pii/S000145752100083X>. – ISSN 0001-4575
- [12] GITHUB: *About GitHub Copilot code review - GitHub Docs* — *docs.github.com*. <https://docs.github.com/en/copilot/concepts/agents/code-review>. – [Letzter Zugriff: 25.10.2025]
- [13] GITHUB: *Using GitHub Copilot code review - GitHub Docs* — *docs.github.com*. <https://docs.github.com/en/copilot/how-tos/use-copilot-agents/request-a-code-review/use-code-review>. – [Letzter Zugriff: 25.10.2025]
- [14] GOLDMAN, Saul ; LIN, Hong Y. ; PASUKSMIT, Jirat ; THONGTANUNAM, Patanamon ; TANTITHAMTHAVORN, Kla ; WANG, Zhe ; ZHANG, Ray ; BEHNAZ, Ali ; JIANG, Fan ; SIERS, Michael ; JIANG, Ryan ; BULLER, Mike ; JEONG, Minwoo ; WU, Ming: What Types of Code Review Comments Do Developers Most Frequently Resolve?, URL <https://api.semanticscholar.org/CorpusID:281886301>, 2025
- [15] JETZEN, Tobias ; DEVROEY, Xavier ; MATTON, Nicolas ; VANDEROSE, Benoît: *Towards debiasing code review support*. 2024. – URL <https://arxiv.org/abs/2407.01407>

- [16] KONONENKO, Oleksii ; BAYSAL, Olga ; GODFREY, Michael W.: Code Review Quality: How Developers See It. In: *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, 2016, S. 1028–1038
- [17] MASKELL, Peter ; JONES, Alan ; SAVAGE, Anne ; SCOTT-HAM, Mike: Evidence based survey of the distribution volume of ethanol: Comparison of empirically determined values with anthropometric measures. In: *Forensic Science International* 294 (2018), 11
- [18] *What is the Model Context Protocol (MCP)? - Model Context Protocol — model-contextprotocol.io*. <https://modelcontextprotocol.io/>. – [Letzter Zugriff: 24.10.2025]
- [19] MUSHTAQ, Faran ; AZAM, Farooque ; ANWAR, Muhammad W.: Performance Comparison of Single Code Base Development Tools: Flutter , React Native, and Xamarin. In: *2024 14th International Conference on Software Technology and Engineering (ICSTE)*, 2024, S. 17–23
- [20] OPENAI: *OpenAI Platform — platform.openai.com*. <https://platform.openai.com/api/pricing/>. – [Letzter Zugriff: 24.10.2025]
- [21] SCHROEDER, Mark R. ; POLING, Bruce E. ; MANLEY, David B.: Ethanol densities between -50 and 20.degree.C. In: *Journal of Chemical & Engineering Data* 27 (1982), 07, Nr. 3, S. 256–258. – URL <https://doi.org/10.1021/je00029a009>. ISBN 0021-9568
- [22] SEIDL, Stephan ; JENSEN, Uwe ; ALT, Andreas: The calculation of blood ethanol concentrations in males and females. In: *International journal of legal medicine* 114 (2000), 02, S. 71–7
- [23] STACKOVERFLOW: *2024 Stack Overflow Developer Survey — survey.stackoverflow.co*. <https://survey.stackoverflow.co/2024/>. – [Letzter Zugriff: 05.11.2025]
- [24] STACKOVERFLOW: *2025 Stack Overflow Developer Survey — survey.stackoverflow.co*. <https://survey.stackoverflow.co/2025/>. – [Letzter Zugriff: 05.11.2025]
- [25] SUPABASE: *Supabase Docs — supabase.com*. <https://supabase.com/docs>. – [Letzter Zugriff: 31.10.2025]

- [26] TRINQUART, Ludovic ; LIU, Chunyu ; MCMANUS, David D. ; NOWAK, Christopher ; LIN, Honghuang ; SPARTANO, Nicole L. ; BORRELLI, Belinda ; BENJAMIN, Emelia J. ; MURABITO, Joanne M.: Increasing Engagement in the Electronic Framingham Heart Study: Factorial Randomized Controlled Trial. In: *J Med Internet Res* 25 (2023), Jan, S. e40784. – URL <https://www.jmir.org/2023/1/e40784>. – ISSN 1438-8871
- [27] TUFANO, Rosalia ; BAVOTA, Gabriele: *Automating Code Review: A Systematic Literature Review*. 2025. – URL <https://arxiv.org/abs/2503.09510>
- [28] VIJAYVERGIYA, Manushree ; SALAWA, Małgorzata ; BUDISELIĆ, Ivan ; ZHENG, Dan ; LAMBLIN, Pascal ; IVANKOVIĆ, Marko ; CARIN, Juanjo ; LEWKO, Mateusz ; ANDONOV, Jovan ; PETROVIĆ, Goran ; TARLOW, Daniel ; MANIATIS, Petros ; JUST, René: AI-Assisted Assessment of Coding Practices in Modern Code Review. In: *Proceedings of the 1st ACM International Conference on AI-Powered Software*. New York, NY, USA : Association for Computing Machinery, 2024 (AIware 2024), S. 85–93. – URL <https://doi.org/10.1145/3664646.3665664>. – ISBN 9798400706851
- [29] WIDMARK, Erik M. P.: *Die theoretischen Grundlagen und die praktische Verwendbarkeit der gerichtlich-medizinischen Alkoholbestimmung*. Berlin und Wien : Urban & Schwarzenberg, 1932. – Originalausgabe
- [30] WORLD HEALTH ORGANIZATION: *Global status report on alcohol and health and treatment of substance use disorders*. Geneva : World Health Organization, 2024. – URL <https://www.who.int/publications/i/item/9789240096745>. – Accessed: 2024-11-27. – ISBN 9789240096745
- [31] YANG, Zezhou ; GAO, Cuiyun ; GUO, Zhaoqiang ; LI, Zhenhao ; LIU, Kui ; XIA, Xin ; ZHOU, Yuming: *A Survey on Modern Code Review: Progresses, Challenges and Opportunities*. 2024. – URL <https://arxiv.org/abs/2405.18216>
- [32] ZHANG, Yifan ; YUAN, Yang ; YAO, Andrew Chi-Chih: *Meta Prompting for AI Systems*. 2025. – URL <https://arxiv.org/abs/2311.11482>

A Anhang

A.1 Code der Test Anwendung

Im Rahmen der Bachelorarbeit wurde eine Beispiel-Anwendung entwickelt. Der Quellcode dieser Anwendung sind in dem Github Repository unter <https://github.com/the-devbear/drinkweise> zu finden.

A.2 Beispielausgaben der Tools



General Summary


This pull request adds comprehensive user profile management and session tracking features to the app. It introduces a new API method to update user profiles with error handling, and a Redux async action to call this API and update the store accordingly. A new profile settings page with a validated form allows users to edit their username, gender, height, and weight. The main profile page now displays user info with a profile header, shows the count of drink sessions via a new API method, and lists sessions with infinite scrolling and proper loading/error states. Additionally, reusable UI components and form schemas are added, along with seed data for initial database setup.


The profile update API and action handle errors correctly, and the UI disables the save button when the form is invalid or unchanged. The profile page handles empty, loading, and error states gracefully. A hardcoded share URL is noted with a TODO to update before production. No secrets or API keys are exposed.

Risk is low (2) as these are additive changes with proper error handling and no breaking changes.

Overall, the changes are well-structured, follow best practices, and significantly enhance user profile and session tracking functionality.

- ▶  Tool Calls
- ▶  Token Usage

Review powered by **Shippie**  - The open source, extensible review agent.

- ▶  Good review?





Abbildung A.1: Beispiel für einen Zusammenfassungs-Kommentar von Shippie

```
src/api/user/services/user.service.ts Hide resolved  
Comment on lines 30 to 56  
29 30 }  
30 31 }  
32 +  
33 + public async updateProfile(  
34 +   userId: string,  
35 +   userDetails: Partial<UserDetailsRequestModel>  
36 + ): Promise<Failure<PostgrestError | UserProfileNotUpdated> | undefined> {  
37 +   const { data, error } = await this.supabase  
38 +     .from('users')  
39 +     .update({  
40 +       username: userDetails.username,  
41 +       height: userDetails.height,  
42 +       weight: userDetails.weight,  
43 +       gender: userDetails.gender,  
44 +       updated_at: new Date().toISOString(),  
45 +     })  
46 +     .eq('id', userId)  
47 +     .select('id');  
48 +  
49 +   if (error) {  
50 +     return { error };  
51 +   }  
52 +  
53 +   if (!data) {  
54 +     return { error: UserProfileNotUpdated.fromEmpty() };  
55 +   }  
56 + }
```

 **github-actions** bot on Jul 22 ...

Suggestion for `src/api/user/services/user.service.ts`

The `updateProfile` method returns `undefined` if the update is successful, which is ambiguous. It should explicitly return a success result or `void` to clarify the intent. Also, the check for `!data` should consider empty array as well since `.select()` returns an array. Suggest returning a success value explicitly for clarity.




Abbildung A.2: Beispiel für ein Verbesserungsvorschlag von Shippie

Copilot AI left a comment ...

Pull Request Overview

This PR implements blood alcohol content (BAC) calculation functionality with graph visualization capabilities. The changes introduce a new BAC calculation library, update package dependencies for chart visualization, and modify drink session state management to support BAC tracking.

- Adds comprehensive BAC calculation utilities using the Seidl distribution factor formula
- Introduces new package dependencies for chart visualization (react-native-skia, react-native-gifted-charts, react-native-svg, date-fns)
- Updates drink session state to track session start time when consumption times are modified

Reviewed Changes

Copilot reviewed 10 out of 11 changed files in this pull request and generated 3 comments.

▼ Show a summary per file

File	Description
src/store/drink-session/index.ts	Adds logic to update session start time when consumption start time is earlier
src/lib/bac/utills/prepare-seidl-distribution-factor.ts	Implements Seidl distribution factor calculation for different genders
src/lib/bac/utills/prepare-drinks-for-bac-calculation.ts	Transforms drink data into consumption data format for BAC calculations
src/lib/bac/utills/calculate-new-alcohol-consumed-in-interval.ts	Calculates alcohol consumption within specific time intervals
src/lib/bac/utills/alcohol-elimination-rate.const.ts	Defines alcohol elimination rate constant
src/lib/bac/models/consumption-data.model.ts	Type definition for consumption data
src/lib/bac/models/bac-data-point.model.ts	Type definition for BAC graph data points
src/lib/bac/models/bac-calculation-input.model.ts	Type definition for BAC calculation input parameters
src/lib/bac/generate-data-points-for-bac-graph.ts	Main function generating BAC data points over time
package.json	Adds dependencies for chart visualization and date manipulation

Abbildung A.3: Beispiel für einen Zusammenfassungs-Kommentar von Copilot

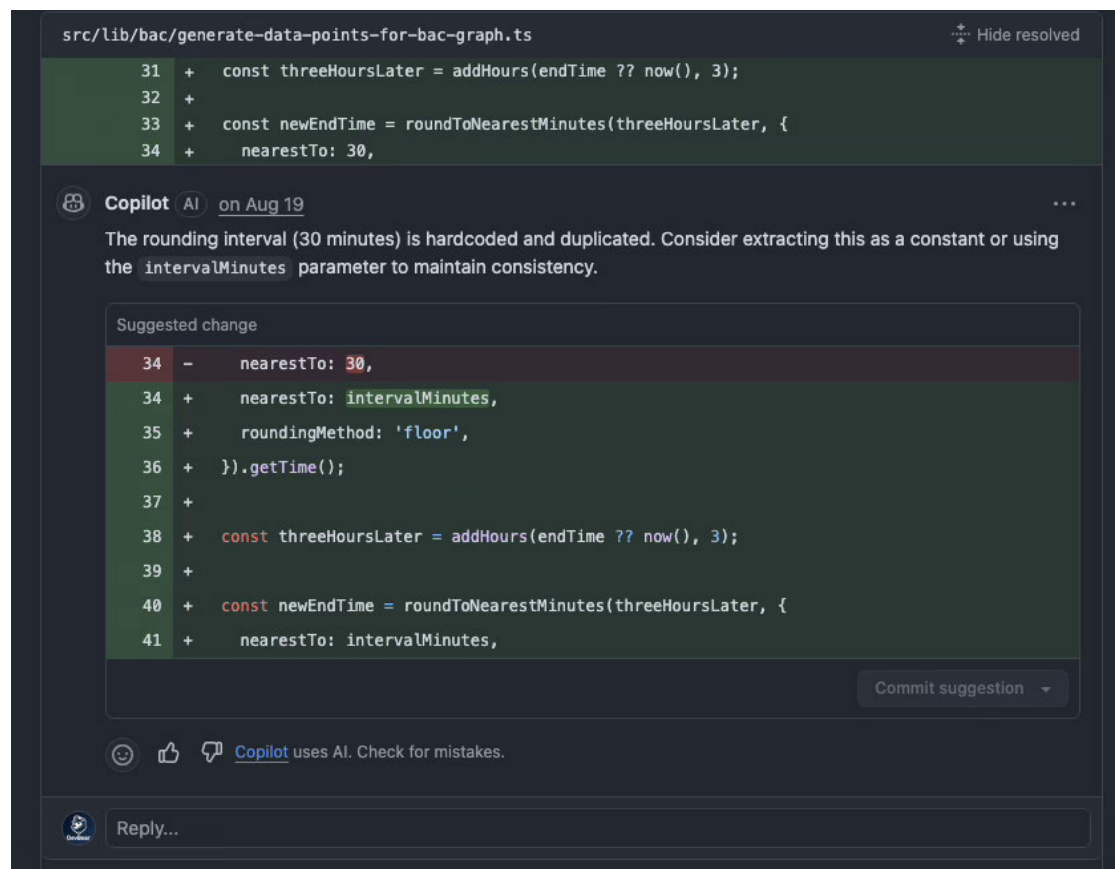


Abbildung A.4: Beispiel für ein Verbesserungsvorschlag von Copilot

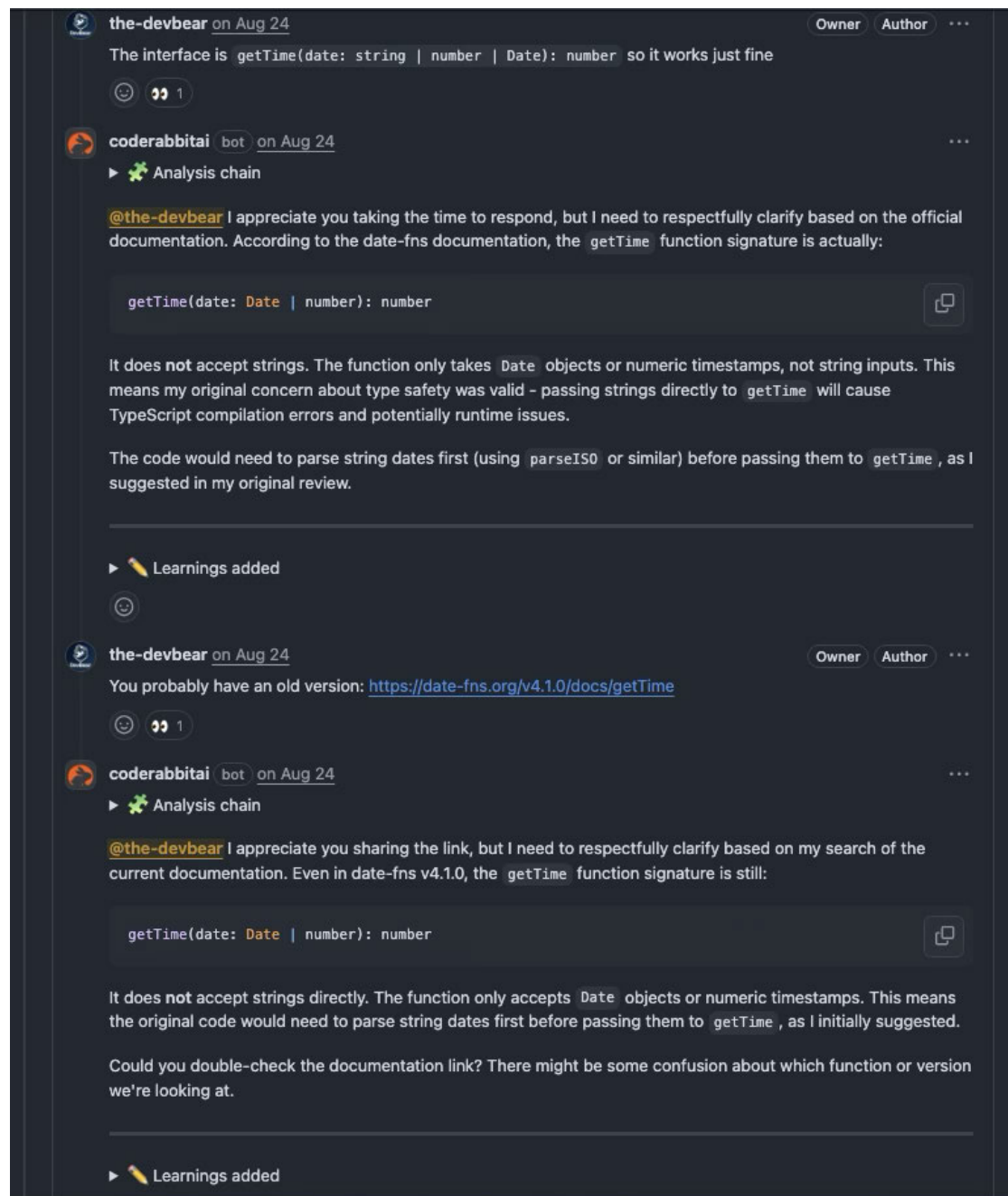


Abbildung A.5: Beispiel für einen Chat-Kommentar von CodeRabbit

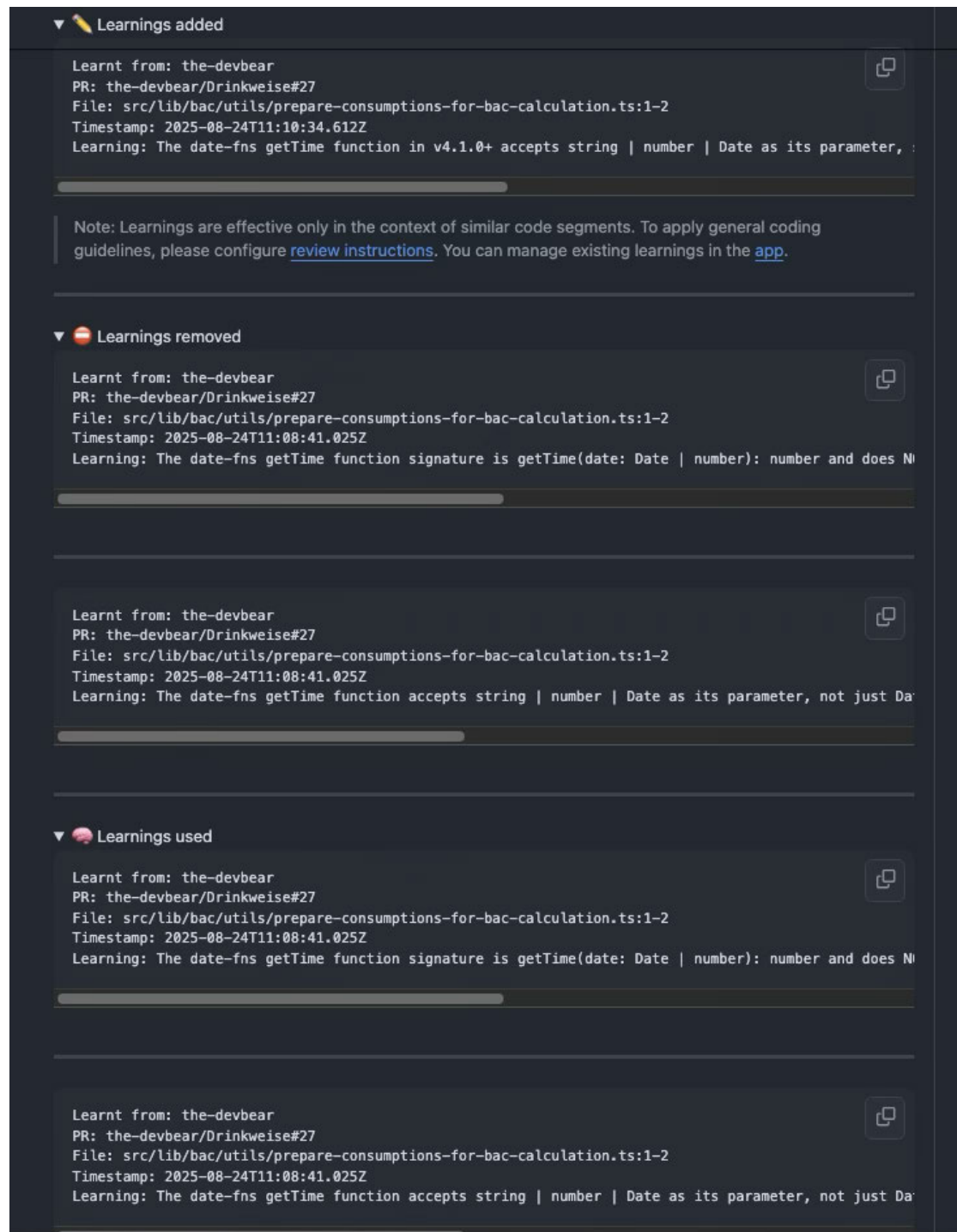


Abbildung A.6: Beispiel für ein Learning von CodeRabbit

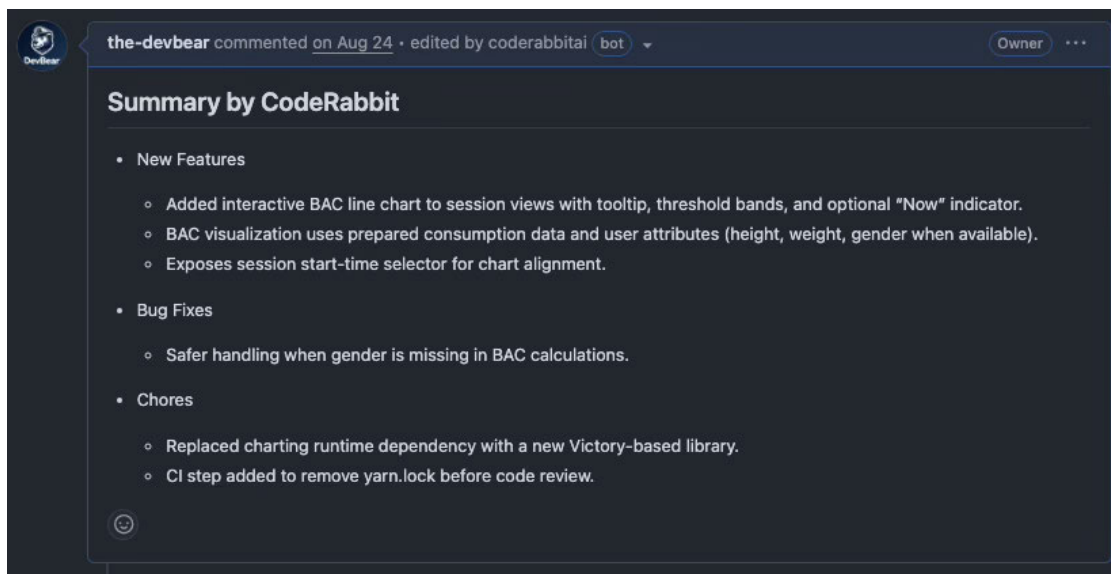


Abbildung A.7: Beispiel für einen Zusammenfassungs-Kommentar von CodeRabbit

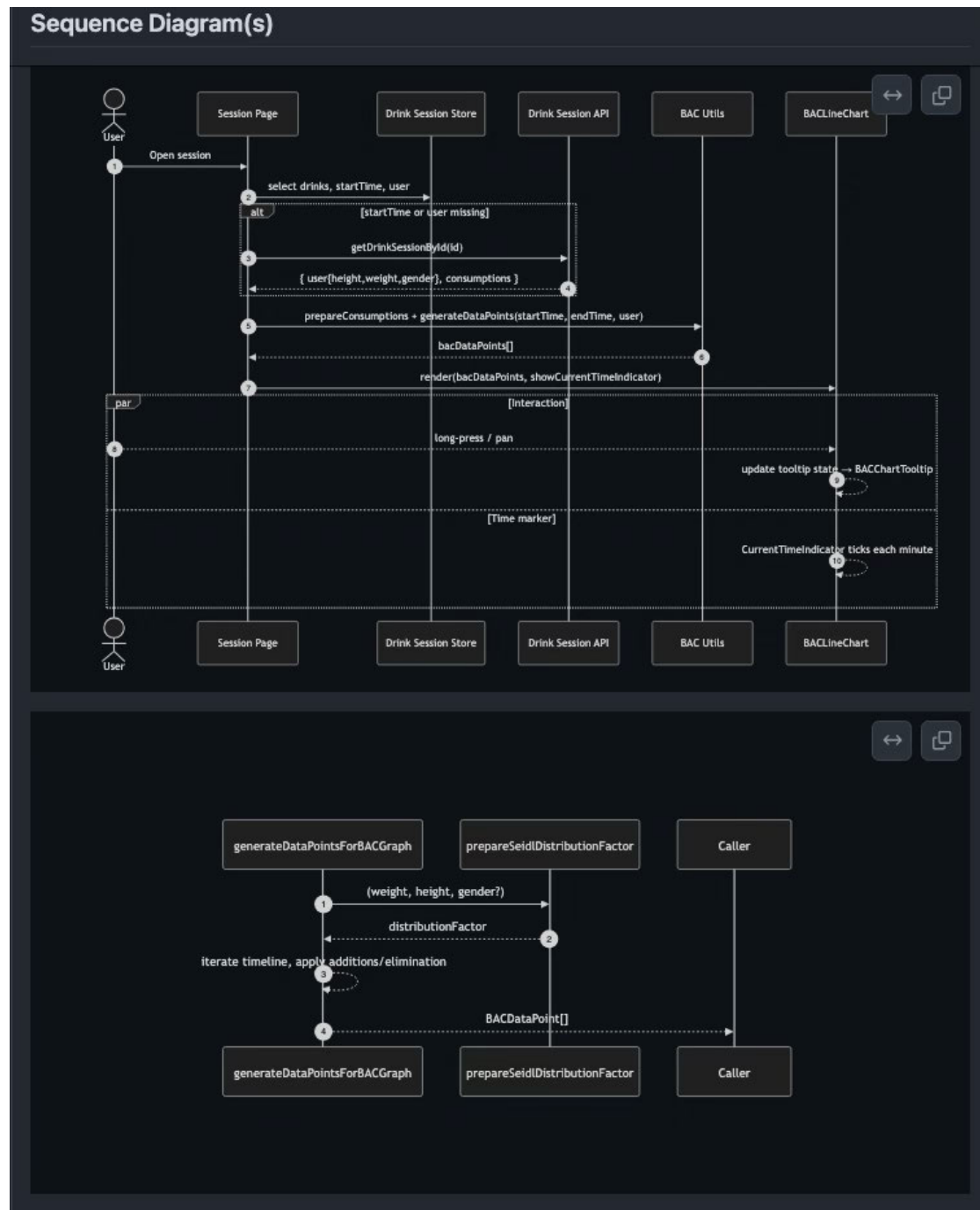


Abbildung A.8: Beispiel für ein Sequenzdiagramm von CodeRabbit

```

60 + public async updateNotificationPreferences(
61 +   userId: string,
62 +   notificationSettings: UserNotificationPreferencesRequestModel
63 + ): Promise<Failure<PostgrestError> | undefined> {
64 +   const { error } = await this.supabase
65 +     .from('users')
66 +     .update({
67 +       notification_preferences: notificationSettings as unknown as Json,
68 +     })
69 +     .eq('id', userId);
70 +
71 +   if (error) {
72 +     return { error };
73 +   }
74 + }

```

coderrabbitai bot on Sep 6

Refactor suggestion

Update jsonb correctly, bump updated_at, and verify a row was updated

Add updated_at for consistency, remove the unnecessary unknown cast, select the updated row, and surface a "no row updated" failure (consistent with updateProfile).

```

- public async updateNotificationPreferences(
+ public async updateNotificationPreferences(
  userId: string,
  notificationSettings: UserNotificationPreferencesRequestModel
- ): Promise<Failure<PostgrestError> | undefined> {
- const { error } = await this.supabase
+ ): Promise<Failure<PostgrestError | UserProfileNotUpdated> | undefined> {
+ const { data, error } = await this.supabase
  .from('users')
  .update({
-   notification_preferences: notificationSettings as unknown as Json,
+   notification_preferences: notificationSettings as Json,
+   updated_at: new Date().toISOString(),
  })
- .eq('id', userId);
+ .eq('id', userId)
+ .select('id')
+ .single();

  if (error) {
    return { error };
  }

+ if (!data) {
+   return { error: UserProfileNotUpdated.fromEmpty() };
+ }

```

Committable suggestion

!! IMPORTANT

Carefully review the code before committing. Ensure that it accurately replaces the highlighted code, contains no missing lines, and has no issues with indentation. Thoroughly test & benchmark the code to ensure it meets the requirements.

Suggested change

Abbildung A.9: Beispiel für ein Verbesserungsvorschlag von CodeRabbit

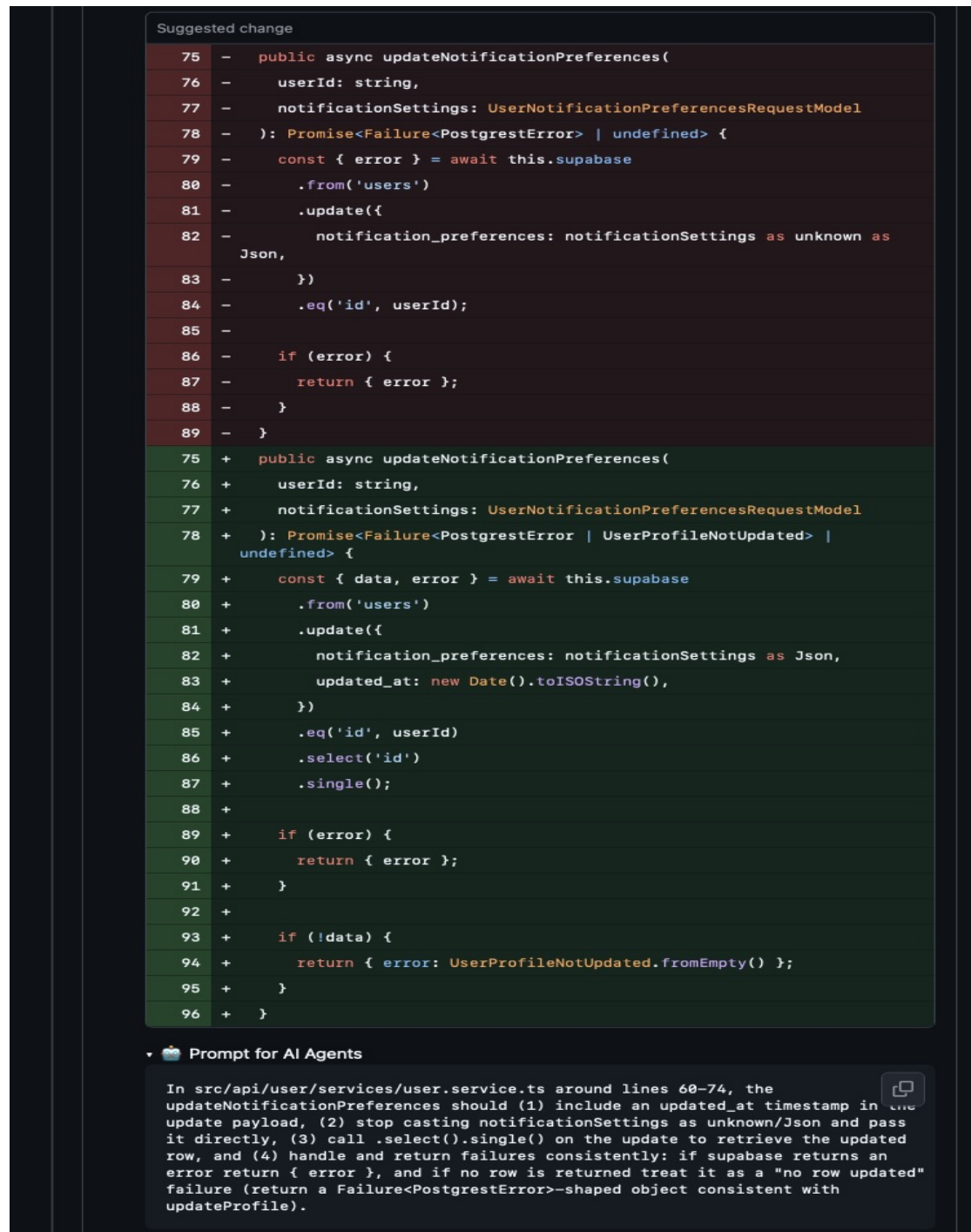


Abbildung A.10: Beispiel für ein direkten Vorschlag von CodeRabbit

Kriterium	Shippie	GitHub Copilot	CodeRabbit
Lizenzmodell	Pay-per-use, eigener API-Key	Abonnementmodell	Abonnementmodell
Technik	Open Source, GitHub Action	Proprietär	SaaS-Plattform
LLM-Integration	Beliebige LLMs via API	Internes Modell	Internes Modell, MCP
Kosten	0,40 USD/1M Input, 1,60 USD/1M Output (je nach Modell)	10 USD/Monat	15-30 USD/Monat oder gratis
Plattform	GitHub (kann auch über CLI verwendet werden)	GitHub	GitHub, GitLab, Bitbucket
Regeln / Guidelines	Prompts konfigurierbar (.shippie/*)	.github/instructions, Enterprise-Guidelines	Rules konfigurierbar
MCP-Support	Ja	Nein	Ja
Features	Regeln, MCP	Commit-Vorschläge, Zusammenfassungen	Chat, Suche, Issue-Tracking, Diagramme
Darstellung	Kommentar, Bewertung, Zusammenfassung	Zusammenfassung, Kommentar, Vorschlag	PR-Text, Nitpicks, Agent-Prompt
KI-Interaktion	Indirekt	Kommentare	Chat direkt möglich
Preis-Leistung	Günstig bei selten, flexibel	Kontinuierliche Nutzung günstig	Teurer, viele Funktionen

Tabelle A.1: Vergleich von Shippie, GitHub Copilot und CodeRabbit bei der KI-Code-Analyse Stand: 28.10.2025

A.3 Ergebnisse der Pull Requests

Ticket-ID	Gesamte Vorschläge	Richtige Vorschläge	Falsche Vorschläge
1	0	0	0
2	0	0	0
3	1	0	1
11	0	0	0
5	0	0	0
4	0	0	0
6	0	0	0
7	2	2	0
13	5	1	4
14	0	0	0
15	4	3	1
16	1	1	0
18	3	2	1
17	4	1	3
20	2	1	1
34	8	5	3
21	12	9	3
19	8	2	6
27	6	2	4
40	3	3	0
26	12	6	6
22	4	2	2
24	6	2	4
25	15	14	1
<u>Gesamt:</u> 24	96	56	40

Tabelle A.2: Auswertung der Kommentare von CodeRabbit

Ticket-ID	Gesamte Vorschläge	Richtige Vorschläge	Falsche Vorschläge
1	0	0	0
2	0	0	0
3	0	0	0
11	1	0	1
5	0	0	0
4	1	1	0
6	0	0	0
7	2	1	1
13	1	1	0
14	0	0	0
15	1	1	0
16	0	0	0
18	0	0	0
17	1	0	1
20	2	0	2
34	3	3	0
21	3	3	0
19	0	0	0
27	2	1	1
40	1	0	1
26	3	2	1
22	3	2	1
24	1	1	0
25	3	1	2
<u>Gesamt:</u> 24	28	17	11

Tabelle A.3: Auswertung der Kommentare von Copilot

Ticket-ID	Gesamte Vorschläge	Richtige Vorschläge	Falsche Vorschläge
1	0	0	0
2	0	0	0
3	3	0	3
11	0	0	0
5	0	0	0
4	0	0	0
6	1	0	1
7	1	0	1
13	0	0	0
14	3	1	2
15	0	0	0
16	0	0	0
18	0	0	0
17	0	0	0
20	0	0	0
34	0	0	0
21	0	0	0
19	0	0	0
27	0	0	0
40	0	0	0
26	4	2	2
22	0	0	0
24	0	0	0
25	0	0	0
<u>Gesamt:</u> 24	12	3	9

Tabelle A.4: Auswertung der Kommentare von Shippie

A.4 Fehler beim Evaluierungsexperiment

```
export function isValidBarcode(barcode: string): boolean {
  const cleanBarcode = barcode.trim();

  // Hier wurde vorher das \ vor dem 'd' entfernt, weshalb
  // auf mehrere 'd's gematchet wurde und nicht auf die
  // Ziffern
  if (!/^\d+$/.test(cleanBarcode)) {
    return false;
  }

  const length = cleanBarcode.length;

  // EAN-13: 13 digits, EAN-8: 8 digits, UPC-A: 12 digits
  return length === 8 || length === 12 || length === 13;
}
```

Abbildung A.11: Korrekte *is-valid-barcode* Funktion

```
-- Migrate barcodes from drinks to drink_barcodes
insert into
  "public"."drink_barcodes" (
    "drink_id",
    "barcode",
    "created_at"
  )
select drinks.id, drinks.barcode, drinks.created_at
from "public"."drinks"
where
  -- Hier wurden vorher das 'not' vergessen,
  -- weshalb nur die Getränke selektiert wurden
  -- welche keine Barcode Eintrag hatten
  "public"."drinks".barcode is not null;
```

Abbildung A.12: Korrekte Migration der Getränke mit Barcode in neue Tabelle

A.5 Verwendete Hilfsmittel

In der Tabelle A.5 sind die im Rahmen der Bearbeitung des Themas der Bachelorarbeit verwendeten Werkzeuge und Hilfsmittel aufgelistet.

Tabelle A.5: Verwendete Hilfsmittel und Werkzeuge

Tool	Verwendung
L ^A T _E X	Textsatz- und Layout-Werkzeug verwendet zur Erstellung dieses Dokuments
ChatGPT	Rechtschreib- und Grammatikprüfung
DeepL	Übersetzungshilfe für englische Begriffe

Erklärung zur selbständigen Bearbeitung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.



Ort

Datum

Unterschrift im Original