



FOM Hochschule für Oekonomie & Management
Hochschulzentrum Nürnberg

Fallstudie / Wissenschaftliches Arbeiten
SS 2024 im Studiengang Wirtschaftsinformatik

über das Thema

Smartphone/Smartwatch App

von

Eray Yasar: 740108
Gürkan Turan: 731234

Dozent
Datum

Prof. Dr. Klemens Waldhör
31.07.2024

Inhaltsverzeichnis

Abbildungsverzeichnis	V
Tabellenverzeichnis	VI
1 Einleitung	2
1.1 Hinführung zum Thema	2
1.2 Forschungsfragen	2
1.3 Relevanz der Arbeit	4
1.4 Methodik	5
1.4.1 Programmieren	5
1.4.2 Literaturrecherche	6
2 Technologische Grundlagen	7
2.1 Einführung in die Sensortechnologie	7
2.1.1 Arten von Sensoren auf Smartphones und Smartwatches	8
2.1.2 Erfassung und Verarbeitung von Sensordaten	8
2.2 Bedeutung der Sensordaten	9
2.3 Methoden der Datenvisualisierung	10
2.3.1 Grafische Darstellungen	10
2.3.2 Nutzung von Bibliotheken und Tools	11
3 Erstellung der Smartphone-/Smartwatch-App in Android Studio	12
3.1 Anforderungen und Spezifikationen	12
3.2 Entwicklung der Smartphone-/Smartwatch-App	15
3.2.1 Einführung	15
3.2.2 Einrichtung der Entwicklungsumgebung	16
3.2.3 Integration von ChatGPT	17

III

3.2.4	Projektstruktur und Konfigurationsdateien	18
3.2.5	AndroidManifest.xml	18
3.2.6	build.gradle.....	19
3.2.7	DataRepository	20
3.2.8	SQLiteOpenHelper	21
3.2.9	SensorPagerAdapter	22
3.2.10	AccelFragment	23
3.2.11	GyroFragment	24
3.2.12	MainActivity	26
3.2.13	activity_main.xml.....	28
3.2.14	fragment_accel.xml.....	30
3.2.15	fragment_gyro.xml.....	30
3.2.16	Darstellung der Graphen und der Benutzeroberfläche.....	31
3.2.17	Verwendung des SimpleSqliteBrowser Plugins	33
3.2.18	Zusammenfassung.....	35
3.3	Architektur des Programms.....	35
4	Gegenüberstellung der Ansätze.....	41
4.1	Erstellung des Programms durch ChatGPT	42
4.1.1	Funktionale Vollständigkeit.....	42
4.1.2	Fehlerfreiheit des Programms	43
4.2	Verständlichkeit des Programms	43
4.2.1	Klarheit des Programmcodes	43
4.2.2	Verständlichkeit der ChatGPT-Antworten	44
4.3	Häufigkeit der Anfragen und Korrekturbedarf.....	44
4.3.1	Häufigkeit der Anfragen.....	45
4.3.2	Zeitaufwand für Korrektur und Validierung.....	45

IV

4.4	Präzision des Inputs und Konsistenz der Antworten.....	46
4.4.1	Präzision des Inputs.....	46
4.4.2	Konsistenz der Antworten.....	47
4.5	Effektivität von kleinen Anfragen.....	47
4.5.1	Präzision der Antworten.....	47
4.5.2	Aufwand der Anfragen.....	48
5	Diskussion.....	49
5.1	Auswertung der Ergebnisse.....	49
5.2	Beantwortung der Forschungsfragen und Interpretation der Ergebnisse.....	50
5.3	Limitationen der Arbeit.....	53
6	Fazit.....	53
6.1	Zusammenfassung der Ergebnisse.....	53
6.2	Schlussfolgerung.....	54
6.3	Ausblick.....	55
	Literaturverzeichnis.....	55
	Anhang.....	58

Abbildungsverzeichnis

Abbildung 1: Koordinatensystem, das von der Sensor API verwendet wird.	13
Abbildung 2: Definition SKD-Versionen in der build.gradle-Datei	16
Abbildung 3: Fehlerhandling Teil 1	17
Abbildung 4: Fehlerhandling Teil 2	17
Abbildung 5: Verzeichnis	18
Abbildung 6: AndroidManifest.xml Datei	19
Abbildung 7: Definition der Bibliotheken	20
Abbildung 8: DataRepository.kt Datei	21
Abbildung 9: SQLiteOpenHelper.kt bzw. DatabaseHelper	22
Abbildung 10: SensorPagerAdapter.kt Datei	23
Abbildung 11: AccelFragment.kt Datei Teil 2	24
Abbildung 12: AccelFragment.kt Datei Teil 1	24
Abbildung 13:GyroFragment.kt Datei Teil 1	25
Abbildung 14: GyroFragment Teil 2	25
Abbildung 15: MainActivity.kt Datei Teil 1	26
Abbildung 16: MainActivity.kt Datei Teil 3	27
Abbildung 17: MainActivity.kt Datei Teil 2	27
Abbildung 18:activity_main.xml Datei	29
Abbildung 19:fragment_accel.xml Datei	30
Abbildung 20: fragment_gyro.xml Datei	30
Abbildung 21:Diagramm in der App - Beschleunigungssensor	31
Abbildung 22: Diagramm in der App - Gyroskop	32
Abbildung 23:SimpleSqliteBrowser Plugin	34
Abbildung 24: MVC-Modell	37
Abbildung 25: UML-Klassendiagramm	39

Tabellenverzeichnis

Tabelle 1: Zusammenfassung der Bewertungen	49
--	----

1 Einleitung

1.1 Hinführung zum Thema

Seit der Veröffentlichung von ChatGPT im November 2022 gewinnen KI-Chatbots immer mehr an Aufmerksamkeit (vgl. Meyer et al. 2023). Diese Art von Chatbots interagiert mit Nutzern und verwendet dabei menschenähnliche Sprache, um Fragen zu beantworten, Informationen wiederzugeben und eigene Inhalte zu erstellen (vgl. ebd. 2023). Einerseits kann ChatGPT in Bereichen wie Forschung und Bildung zur Unterstützung dienen, um schnellstmöglich Antworten liefern zu können (vgl. ebd.). Andererseits gibt es viele Probleme, auf die geachtet werden muss, darunter Aspekte wie die Qualität und Richtigkeit der Antworten sowie die Feststellung, wer die Inhalte erstellt hat (vgl. ebd.). Herausforderungen wie diese verdeutlichen nochmals, wie wichtig es ist, die Probleme von Large Language Models, insbesondere von ChatGPT, näher zu betrachten.

Parallel zur zunehmenden Aufmerksamkeit für KI-Chatbots hat auch die Nutzung von Sensordaten aus Smartphones und Smartwatches erheblich an Bedeutung gewonnen (vgl. Masoumian Hosseini et al. 2023; vgl. Shin et al. 2019). Mithilfe dieser Daten wird es ermöglicht, wertvolle Einblicke in verschiedene Aspekte des täglichen Lebens zu bekommen, welche in Bereichen wie Gesundheit, Fitness und Forschung von großer Bedeutung sind (vgl. Shin et al. 2019). Sensordaten können unter anderem genutzt werden, um körperliche Aktivitäten zu überwachen, Schlafmuster zu analysieren und Gesundheitszustände zu verfolgen (vgl. ebd. 2019).

In der folgenden Arbeit werden die Technologien ChatGPT und Sensordaten von Smartphones und Smartwatches genutzt. Ziel ist es, ein Programm mithilfe von ChatGPT zu erstellen, das Sensordaten eines Smartphones oder einer Smartwatch ausliest, abspeichert und graphisch darstellt.

1.2 Forschungsfragen

Zur Untersuchung der Leistungsfähigkeit und der möglichen Herausforderungen, bei der Verwendung von ChatGPT zur Entwicklung einer App, welche Sensordaten erfasst und gleichzeitig visualisiert, werden folgende fünf Forschungsfragen und Hypothesen aufgestellt:

1. **Forschungsfrage:** Inwieweit kann ChatGPT ein Programm erstellen, das den spezifizierten funktionalen Anforderungen entspricht?

Hypothese: Das von ChatGPT erstellte Programm enthält Fehler, die zu Funktionsstörungen und unvollständigen Programmen führen kann.

2. **Forschungsfrage:** Wie verständlich sind das von ChatGPT erstellte Programm und dessen erklärende Antworten für den Benutzer?

Hypothese: Der von ChatGPT generierte Programmcode und die zugehörigen Antworten sind komplex strukturiert und schwer verständlich für den Anwender.

3. **Forschungsfrage:** Wie oft muss der Benutzer seine Anfragen an ChatGPT umformulieren, um die gewünschte Programmfunktionalität zu erreichen?

Hypothese: Der Anwender muss häufig Anpassungen an seinen Anfragen vornehmen, um die gewünschten Ergebnisse von ChatGPT zu erzielen.

4. **Forschungsfrage:** Wie beeinflusst die Präzision der Benutzereingaben die Qualität der von ChatGPT generierten Antworten und Programme?

Hypothese: Die Präzision der Benutzereingaben ist entscheidend für die Qualität des von ChatGPT generierten Programms

5. **Forschungsfrage:** Wie beeinflusst die Aufteilung einer komplexen Aufgabe in mehrere kleine Anfragen an ChatGPT die Effizienz und Qualität der erzielten Ergebnisse?

Hypothese: Umfangreiche Aufgaben werden effizienter und qualitativ hochwertiger gelöst, wenn sie in mehreren kleinen, präzisen Anfragen an ChatGPT gestellt werden, anstatt in einer einzigen großen Anfrage.

Diese Fragestellungen dienen dazu, anhand einer praktischen Anwendung die Leistungsfähigkeit bei der Nutzung von ChatGPT zur Lösung einer bestimmten Aufgabe zu bewerten.

1.3 Relevanz der Arbeit

Die Relevanz dieser Arbeit ist der fortschreitenden Integration von Künstlicher Intelligenz im Alltag geschuldet und der zunehmenden Bedeutung von Sensordaten (vgl. Masoumian Hosseini et al. 2023; vgl. Shin et al. 2019).

Komplexe Programmieraufgaben könnten dank der Fähigkeit von ChatGPT die Entwicklung von Anwendungen deutlich vereinfachen und beschleunigen. Bei der Entwicklung einer solchen Anwendung wird durch die Untersuchung der Leistungsfähigkeit von ChatGPT ein wichtiger Beitrag zur aktuellen Forschung in den Bereichen der KI und ihrer praktischen Anwendungsmöglichkeiten geleistet.

Die praktische Relevanz zeigt sich in der Effizienzsteigerung bei der Entwicklung von Anwendungen. Mithilfe von künstlicher Intelligenz könnten Programmierer ihre Entwicklungszeit verkürzen, indem ChatGPT viele Aufgaben automatisiert. Durch den Einsatz von ChatGPT in Entwicklungsprozessen könnten Unternehmen Kosten einsparen, die sonst für manuelle Programmierung und Fehlerbehebung anfallen würden. Außerdem könnten dadurch auch weniger erfahrene Entwickler eher komplexe Anwendungen erstellen als ohne solche Unterstützung.

Die Ergebnisse dieser Arbeit sind für verschiedene Zielgruppen von Bedeutung. Von den Beobachtungen profitieren Schüler, Studenten und Forscher, indem sie lernen, ihre Projekte mithilfe von KI-Tools effizienter zu gestalten. Programmierer und IT-Experten können Ergebnisse von Forschungen nutzen, um ihre Arbeitsabläufe zu verbessern. Dadurch haben sie die Kapazität, um sich auf anspruchsvollere und kreativere Aufgaben zu fokussieren. Im Gegensatz dazu haben Unternehmen eine eher vollautomatisierte Vision, bei der sie möglichst wenige Experten benötigen.

Des Weiteren unterstützt die Arbeit die zunehmende Digitalisierung und Integration von künstlicher Intelligenz in alltäglichen und beruflichen Anwendungen, was langfristig zur smarteren und effizienteren Arbeitsweise führt. Durch die Automatisierung trivialer und wiederkehrender Aufgaben können Ressourcen gespart und für anspruchsvollere Tätigkeiten genutzt werden. Dies führt zu einer höheren Produktivität und Zufriedenheit am Arbeitsplatz.

1.4 Methodik

Um die in dieser Arbeit aufgestellten Forschungsfragen zu beantworten und die Relevanz der Arbeit hervorzuheben, wurden zwei Methoden angewendet: die Programmierung und die Literaturrecherche.

Während sich die Programmierung auf den praktischen Teil der Umsetzung und die Bewertung von ChatGPT's Leistungsfähigkeit konzentrierte, wurde die Literaturrecherche genutzt, theoretische Grundlagen zu den relevanten Themenbereichen zu erarbeiten.

1.4.1 Programmieren

Der Hauptteil dieser Arbeit ist die Nutzung von ChatGPT zur Entwicklung eines Programms, das Sensordaten von einem Smartphone ausliest, speichert und grafisch darstellt. Um die Forschungsfragen beantworten zu können und die Fähigkeiten dieses KI-Modells zu bewerten, wurde ChatGPT in Form einer sorgfältig strukturierten Vorgehensweise eingesetzt.

Ausschlaggebend für die erzielten relevanten Ergebnisse war es, ChatGPT sehr umfangreich über das Thema der Arbeit zu informieren, damit das Modell die Aufgabe vollständig verstehen konnte. Verwendet wurden ChatGPT 4o und ein spezielles GPT-Modell namens Android Studio Developer, da sich dieses außerordentlich gut für die Entwicklungsumgebung Android Studio und die Programmiersprache Kotlin eignet. Da Kotlin die aktuell modernere und bevorzugtere Programmiersprache ist, wurde Kotlin anstelle von Java für die Entwicklung der Android-App verwendet.

ChatGPT erhielt detaillierte Vorgaben für die Erstellung des Codes. Diese umfassten genaue Anweisungen zur Struktur, ausführliche Erklärungen zu jedem Codeabschnitt und die Anforderung, vollständige Codeabschnitte zu liefern. Die Absicht war es, den generierten Code ohne zusätzliche Anpassungen direkt in das Programm aufzunehmen.

Zu Beginn wurden die Anfragen an ChatGPT sehr detailliert formuliert. In diesen Anfragen wurden die benötigten Methoden genau beschrieben, um deren Implementierung und Platzierung im Code sicherzustellen. Diese präzisen Vorgaben sollten sicherstellen, dass das Modell die Anforderungen korrekt erfüllt. Während des Entwicklungsprozesses sind

die Anfragen zur Fehlersuche und -behebung gestiegen, weshalb diese deutlich kürzer und oberflächlicher formuliert wurde, da nur noch nach Fehlern gefragt beziehungsweise der Fehlercode und die Fehlernachricht weitergegeben wurde.

Der erzeugte Code wurde direkt in das Programm eingefügt und anschließend getestet. Zum Testprozess gehörten das Neustarten der App und die Überprüfung, ob die implementierten Methoden im Programm wie gewünscht funktionieren. Durch jeden Testlauf wurden die Funktionalität und Stabilität der App überprüft. Bei Fehlern wie dem Abstürzen der App oder inkorrekten Ergebnissen wurden ChatGPT beziehungsweise Android Studio Developer erneut verwendet, um diese Probleme zu lösen.

Ein weiterer wichtiger Aspekt der Methodik war die Bewertung des von ChatGPT geschriebenen Codes. Die Bewertung erfolgte anhand mehrerer Kriterien: Verständlichkeit, Korrektheit und Aufwand. Die Verständlichkeit des Codes wurde geprüft, indem nachvollziehbar war, ob der Code auch ohne tiefere Kenntnisse der Programmiersprache Kotlin verständlich ist. Die Korrektheit des Codes wurde durch die erfolgreiche Implementierung und Funktionalität in der App beurteilt. Die Zeit für Fehlersuche und -behebung sowie die Anzahl der notwendigen Anfragen an ChatGPT beziehungsweise Android Studio Developer zählen zum Aufwand des Programms. Die genauen Kriterien für die Bewertung werden im späteren Verlauf der Arbeit detailliert erläutert.

1.4.2 Literaturrecherche

Die Literaturrecherche wurde als Grundlage für das theoretische Verständnis der Arbeit genutzt. Die Hauptthemen umfassten Sensordaten und ihre Bedeutung sowie Methoden der Datenvisualisierung. Ziel der Recherche war es, genügend Informationen zu diesen Themenbereichen zu sammeln, um die theoretischen Grundlagen für die Arbeit zu legen.

Hierfür wurde Scholar GPT genutzt, ein erweitertes Sprachmodell, das auf verschiedene Literaturdatenbanken wie Google Scholar, PubMed und Arxiv zugreift (vgl. Scholar GPT 2024). Diese künstliche Intelligenz wurde von einer externen Person entwickelt und ist in ChatGPT als eigenes GPT integriert (vgl. ebd. 2024). Dadurch wurde eine effiziente und gezielte Suche nach relevanter Literatur ermöglicht. Zusätzlich dazu wurde auch eine eigene Suche in Google Scholar durchgeführt.

Um viele Ergebnisse bei der Suche nach Literatur zu erhalten, wurden für die Suchanfragen spezifische Schlüsselbegriffe wie Sensordaten von Smartphone/Smartwatch und Herausforderungen der künstlichen Intelligenz sowohl in deutscher als auch in englischer Sprache verwendet.

Um seriöse Quellen von Scholar GPT zu erhalten, wurde die Suche nach Literatur auf deutsche und englische Sprache eingeschränkt. Außerdem wurde versucht, nur Quellen aus Fachbüchern, allgemeinen und speziellen Lehrbüchern sowie Forschungsarbeiten von wissenschaftlichen Institutionen und Zeitschriftenbeiträgen zu berücksichtigen.

Pro Anfrage wurden in etwa 5-10 Ergebnisse ausgegeben, die durch präzise Eingaben im Prompt beeinflusst werden konnten. Um möglichst schnell entscheiden zu können, ob die Quelle relevant war, wurde zu jeder Literaturquelle eine kurze Zusammenfassung bereitgestellt. Nach dieser ersten Auswahl wurden die längeren Zusammenfassungen der ausgewählten Quellen gelesen, falls vorhanden. Zusätzlich wurden die Stellen genauer gelesen, die die Schlüsselbegriffe enthielten. Durch diese beiden Schritte konnte in kurzer Zeit noch genauer analysiert werden, ob die Literatur tatsächlich relevant für die Arbeit war.

2 Technologische Grundlagen

Nachdem die Methodik erläutert wurde, ist es wichtig, die technologischen Grundlagen zu verstehen, die dieser Arbeit zugrunde liegen. Diese Grundlagen umfassen die Sensortechnologie, die auf modernen Smartphones und Smartwatches eingesetzt wird.

2.1 Einführung in die Sensortechnologie

Die Sensortechnologie ist eine grundlegende Komponente moderner Smartphones und Smartwatches. Dank der Vielzahl an Sensoren, mit denen die Geräte ausgestattet sind, wird es ermöglicht, physikalische Parameter zu messen wie Bewegung, Orientierung und Umgebungsbedingungen. Der Beschleunigungssensor und das Gyroskop gehören zu den am häufigsten verwendeten Sensoren (vgl. Android Developer Bewegungssensoren 2024).

2.1.1 Arten von Sensoren auf Smartphones und Smartwatches

Beschleunigungssensor: Der Beschleunigungssensor hat die Aufgabe, die Beschleunigung zu messen, der das Gerät in allen drei Raumrichtungen ausgesetzt ist (vgl. Android Developer Bewegungssensoren 2024). Er erfasst die auf das Gerät wirkenden Kräfte und kann somit Bewegungen und Vibrationen erkennen (vgl. Andrejašić 2008:2). Typische Anwendungen sind Schrittzähler, Bewegungserkennung und Schüttelgesten (vgl. Android Developer Bewegungssensoren 2024).

Gyroskop: Das Gyroskop hat die Eigenschaft, die Drehbewegungen des Geräts um die drei Raumachsen zu messen (vgl. Javaid et al. 2021). Durch diesen Sensor werden wichtige Informationen über die Bewegungen und Orientierung des Geräts erfasst (vgl. ebd.). In vielen Anwendungen werden diese Daten genutzt, darunter beispielsweise in der Bildstabilisierung, bei der Navigation oder der Steuerung von Spielen (vgl. ebd.).

Geomagnetischer Sensor: Der geomagnetische Sensor, auch bekannt als Magnetometer, misst das Magnetfeld der Erde. Dieser wird oft mit dem Beschleunigungssensor kombiniert, um die exakte Ausrichtung des Geräts zu bestimmen (vgl. Android Developer Positionssensoren 2024). Zwei der bekanntesten Anwendungsfälle sind hierbei die Kompassfunktion (vgl. Sarmadi et al. 2024) und die erweiterte Realität (AR).

Barometer: Das Barometer misst den atmosphärischen Druck (Sankaran et al. 2014:192). Die Verwendung dieser Daten kann beispielsweise zur Bestimmung der Höhe über dem Meeresspiegel oder zur Vorhersage von Wetteränderungen dienen (vgl. Manivannan et al. 2020). Typische Anwendungen sind Höhenmesser in Fitness- und Outdoor-Apps sowie Wettervorhersagedienste (vgl. ebd.).

2.1.2 Erfassung und Verarbeitung von Sensordaten

Um Sensordaten erfassen zu können, werden diese in der Regel durch spezifische APIs erfolgen, die vom Betriebssystem des Geräts bereitgestellt werden. Bei Android, zum Beispiel, wird die SensorManager-API (Android Developer SensorManager 2024) verwendet, um auf die Sensordaten zuzugreifen. Folgende Schritte umfasst die Erfassung und Verarbeitung der Daten:

1. Registrierung des Sensors: Über die SensorManager-API wird der gewünschte Sensor (z. B. Beschleunigungssensor oder Gyroskop) registriert (vgl. ebd.).
2. Festlegung der Abtastrate: Die Häufigkeit, mit der Daten vom Sensor gelesen werden sollen, wird festgelegt.
3. Erfassung der Daten: Die Sensordaten werden kontinuierlich oder in festgelegten Intervallen erfasst.
4. Verarbeitung der Daten: Die rohen Sensordaten werden verarbeitet, um sie in nutzbare Informationen umzuwandeln.
5. Die verarbeiteten Daten können in einer lokalen Datenbank wie SQLite oder einen externen Server zur späteren Analyse gespeichert werden.

2.2 Bedeutung der Sensordaten

In den letzten Jahren hat die Nutzung von Sensordaten aus Smartphones und Smartwatches stark zugenommen und ermöglicht wertvolle Einblicke in verschiedene Aspekte des alltäglichen Lebens. Vor allem werden diese Daten in den Bereichen Gesundheit, Fitness und Forschung genutzt (vgl. Kataria et al. 2023).

Ein Hauptmerkmal der Sensordaten ist ihre Fähigkeit, genaue Informationen über die physische Aktivität und den Gesundheitszustand der Nutzer zu. Zum Beispiel können mHealth-Anwendungen mithilfe von EKG- oder Photoplethysmographie-Daten (PPG) Vorhofflimmern erkennen (vgl. Lawin et al. 2022). Diese Anwendungen haben sich als sehr zuverlässig in der Diagnose erwiesen, indem sie entweder weitere Sensoren oder die integrierte Sensorik von Smartphones und Smartwatches nutzen (vgl. ebd.).

Entscheidende Anwendungen von Sensordaten liegen in der Erkennung und Überwachung von Gesundheitszuständen. Durch die regelmäßige Überwachung können Anomalien oder gesundheitliche Risiken frühzeitig erkannt und entsprechende Maßnahmen ergriffen werden. Besonders wichtig ist dies für Patienten, die chronisch krank sind, oder für ältere Menschen, die eine regelmäßige Gesundheitsüberwachung benötigen.

Neben der Überwachung von Gesundheitszuständen bieten Sensordaten zusätzlich auch verschiedene Einblicke in das Verhalten und den Lebensstil der Nutzer. Dies umfasst die Überwachung von Schlafmustern, die Analyse von Bewegungsmustern und die

Erkennung spezifischer Aktivitäten. Die Informationen können genutzt werden, um personalisierte Empfehlungen zur Verbesserung der Gesundheit und des Wohlbefindens zu geben.

Die Bedeutung der Sensordaten liegt nicht nur in der Erfassung und Überwachung von Aktivitäten, sondern auch in der Analyse und Interpretation dieser Daten. Zum Einsatz kommen Datenanalysetechniken, die Muster und Trends erkennen können. Diese bieten wertvolle Einblicke in das Verhalten und die Gesundheit der Nutzer. Dadurch wird eine personalisierte und präventive Gesundheitsversorgung ermöglicht, die auf den individuellen Bedürfnissen und Risiken der Nutzer basiert.

2.3 Methoden der Datenvisualisierung

Die Visualisierung von Sensordaten ist ein grundlegender Bestandteil, der dafür sorgt, die erfassten Informationen verständlich und nutzbar zu gestalten. Für die Darstellung von Daten gibt es verschiedene Techniken und Methoden, die abhängig von der Art und dem Umfang der Daten sowie den Anforderungen der Anwendung ausgewählt werden können. Im folgenden Abschnitt werden mehrere der gängigsten Methoden der Datenvisualisierung erläutert, die besonders relevant im Kontext von Sensordaten auf Smartphones und Smartwatches sind.

2.3.1 Grafische Darstellungen

Liniendiagramme

Liniendiagramme gehören zu den häufigsten Methoden zur Darstellung zeitlich aufeinanderfolgender Datenpunkte. Sie eignen sich besonders gut zur Visualisierung von Sensordaten, die kontinuierlich über eine bestimmte Zeit erfasst werden, wie beispielsweise Beschleunigungs- oder Gyroskopdaten. In einem Liniendiagramm werden die Datenpunkte mithilfe von Linien verbunden, was es einfach macht, Trends und Muster im Verlauf der Zeit zu erkennen.

Balkendiagramme

Balkendiagramme eignen sich ideal für den Vergleich von mehreren Datenwerten über verschiedene Kategorien hinweg. Diese Art von Diagramm wird häufig verwendet, um aggregierte Sensordaten darzustellen, wie beispielsweise die durchschnittliche Anzahl an Schritten pro Tag oder die durchschnittliche Herzfrequenz in verschiedenen Zeitpunkten.

Streudiagramme

Um die Beziehung zwischen zweier verschiedenen Datensätzen zu visualisieren, werden Streudiagramme bevorzugt genutzt. Diese Methode ist besonders praktisch, wenn man die Korrelation zwischen verschiedenen Sensordaten beobachten möchte, wie zum Beispiel die Beschleunigung und der Drehgeschwindigkeit eines Geräts.

Echtzeit-Visualisierungen

Besonders in Anwendungen wie Gesundheits- oder Fitness-Tracking-Apps sind Echtzeit-Visualisierungen wichtig, da sie kontinuierlich aktualisierte Daten darstellen müssen. Durch diese Art der Visualisierung erhalten Benutzer eine sofortige Rückmeldung über ihre Aktivitäten oder Gesundheitszustände.

2.3.2 Nutzung von Bibliotheken und Tools

Verschiedene Bibliotheken und Tools sind speziell dafür entwickelt worden, um Daten auf mobilen Endgeräten visuell darzustellen. Zu den bekanntesten Bibliotheken gehören MPAndroidChart und GraphView, die beide eine Vielzahl von Diagrammtypen und Interaktionsmöglichkeiten bieten. Diese Bibliotheken eignen sich besonders gut für die Entwicklung von Android-Anwendungen, da sie eine einfache Integration und Nutzung in Projekten ermöglichen.

MPAndroidChart

MPAndroidChart ist eine Bibliothek zur Erstellung verschiedener Diagrammtypen, darunter Liniendiagramme, Balkendiagramme, Streudiagramme und vieles mehr (vgl. Phil-Jay 2021).

GraphView

Eine weitere Bibliothek für die Datenvisualisierung in Android-Anwendungen ist GraphView. Sie ermöglicht die Erstellung von Diagrammen und Graphen in verschiedenen Formen, darunter Linien-, Balken-, Punkt- und Kurvendiagramme (vgl. jjoe64 2020).

Bedeutung der Datenvisualisierung

Die Bedeutung der Datenvisualisierung besteht darin, komplexe Daten möglichst in einer leicht verständlichen und zugänglichen Form darzustellen. Sie bietet den Benutzern die Möglichkeit, Muster und Trends in den Daten zu erkennen, fundierte Entscheidungen zu treffen und dementsprechend Maßnahmen zu ergreifen. Gut gestaltete Visualisierungen spielen eine entscheidende Rolle, besonders in den Bereichen der Gesundheits- und Fitnessanwendungen, da sie die Benutzer dabei unterstützen, ihre Aktivitäten und Gesundheitszustände besser nachzuvollziehen und zu überwachen.

Mit den beschriebenen Methoden und Techniken der Datenvisualisierung lassen sich Sensordaten effektiv nutzen und interpretieren, sodass der größtmögliche Nutzen aus den erfassten Informationen gezogen werden kann.

3 Erstellung der Smartphone-/Smartwatch-App in Android Studio

3.1 Anforderungen und Spezifikationen

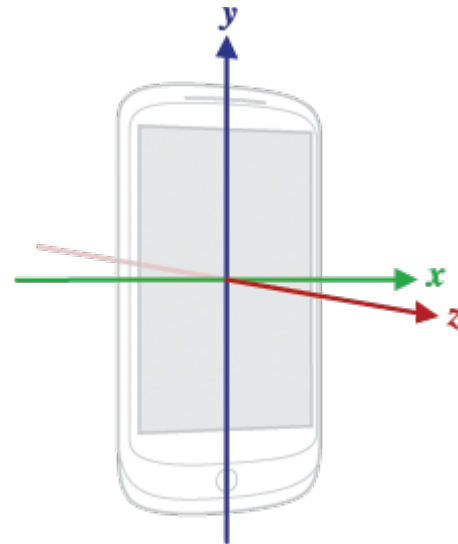
Die Entwicklung der App zur Erfassung von Sensordaten und deren visuellen Darstellung erfordert eine genaue Planung und Festlegung der erforderlichen Hard- und

Softwareanforderungen. Im folgenden Abschnitt werden nicht nur die wesentlichen Anforderungen und Spezifikationen der Entwicklungsumgebung und der Programmiersprache beschrieben, sondern auch die funktionalen Anforderungen.

Für die Entwicklung und den Betrieb der App wurde das Samsung Galaxy S22 Ultra ausgewählt, welches 256 GB Speicher, 12 GB RAM und einen 8-Kern-Prozessor besitzt (vgl. Samsung Mobile Press 2022). Um sicherzustellen, dass die neueste Technologie und die neuesten Sicherheitsfunktionen zum Einsatz kommen, haben wir das Gerät auf der Android-Version 14 laufen lassen. Eine Alternative zum Smartphone wäre die Smartwatch Samsung Watch 3 gewesen, doch die Wahl fiel schließlich auf das Samsung Galaxy S22 Ultra, da es auf dem neuesten Android-Betriebssystem basiert, während die Uhr

das Betriebssystem Tizen OS nutzt, welches seit der Samsung Watch 4 nicht mehr als Betriebssystem für Uhren verwendet wird, da nun Google Wear OS genutzt wird (vgl. Samsung ch 2022). Das Smartphone bietet dank der Vielzahl an Sensoren eine breite Auswahl an Möglichkeiten zur Erfassung von Daten. Einige davon sind: der Beschleunigungssensor, das Gyroskop, der geomagnetische Sensor, das Barometer und viele weitere (vgl. Samsung Mobile Press 2022). Für dieses Projekt wurden zwei Sensoren genutzt: der Beschleunigungssensor und das Gyroskop. Diese Sensoren eignen sich ideal für die Entwicklung und das Testen der App, da sie leicht vorstellbar und einfach nachzustellen sind. Der Beschleunigungssensor misst die Beschleunigung des Geräts in insgesamt drei Dimensionen: entlang der x-, y- und z-Achse. Wenn das Gerät in der „Standardausrichtung gehalten wird ist die X-Achse horizontal und zeigt nach rechts, die Y-Achse ist vertikal und zeigt nach oben und die Z-Achse zeigt zur Außenseite“ (siehe Abbildung 1; Android Developer Sensoren 2024). Das Gyroskop erfasst Drehbewegungen um diese drei Achsen (vgl. Android Developer Sensoren 2024). Im Graphen der App wird jedoch nur die x-Achse dargestellt, um die Daten übersichtlich und verständlich zu präsentieren. Diese Daten sind für viele Anwendungen von Nutzen, darunter Fitness-Tracking, Navigation und Spiele (vgl. Android Studio 2024).

Abbildung 1: Koordinatensystem, das von der Sensor API verwendet wird.



Quelle: Android Developer Sensoren 2024

Die Entwicklungsumgebung für dieses Projekt war die neueste Version von Android Studio Koala vom 01.01.2024. „Android Studio ist die offizielle integrierte Entwicklungsumgebung (Integrated Development Environment, IDE) für die Entwicklung von Android-Apps“ (Android Developer kennenlernen 2024). „Android Studio basiert auf dem leistungsstarken Codeeditor und den Entwicklertools von IntelliJ IDEA“ (ebd. 2024). Außerdem bietet die IDE eine Vielzahl von Tools und Funktionen, die speziell für die Android-Entwicklung optimiert sind. Dazu gehören Live-Edits zur Aktualisierung kombinierbarer Funktionen in Emulatoren und auf physischen Geräten in Echtzeit, die Integration von GitHub und Codebeispielen zur Erstellung üblicher Anwendungsfunktionen sowie zum Importieren von Beispielcode, ein flexibles, auf Gradle basierendes Build-System und vieles mehr (vgl. ebd. 2024). Gradle ist ein erweitertes Build-Toolkit, das den Build-Prozess automatisiert und flexible, benutzerdefinierte Build-Konfigurationen ermöglicht (vgl. Android Developer Build 2024).

Für die Programmierung der App wurde Kotlin verwendet, die offizielle und neueste Sprache für Android-Apps. Kotlin wurde gewählt, weil es moderne Funktionen bietet und von Google stark unterstützt wird. Es ist vollständig interoperabel mit Java, was die Nutzung bestehender Java-Bibliotheken ermöglicht (vgl. Android Developer Kotlin 2024). Kotlin zeichnet sich vor allem durch eine prägnante Syntax aus, die weniger fehleranfällig ist und die Lesbarkeit des Codes verbessert. Außerdem hat Kotlin eingebaute Sicherheitsmechanismen, unter anderem die strikte Handhabung von Null-Werten, die viele der gängigen Programmierfehler verhindern können. (vgl. Chidera 2023)

Zur grafischen Darstellung der Sensordaten wurde die Bibliothek MPAndroidChart genutzt. Diese ermöglicht es, verschiedene Diagramme zu erstellen. Für das Wechseln zwischen verschiedenen Diagrammen wurde die Bibliothek ViewPager2 verwendet. Dadurch kann der Benutzer durch Wischen nach links und rechts zwischen Diagrammen wechseln. Zur Erfassung der Sensordaten wurde die Android SensorManager API verwendet. Diese bietet erweiterte Funktionen für die Arbeit mit Sensoren. Die Datenspeicherung der Sensordaten erfolgte in einer SQLite-Datenbank. Um auf die gespeicherten Sensordaten zuzugreifen und sie anzeigen zu können, wurde das Plugin SimpleSqliteBrowser verwendet.

Die App hat mehrere wesentliche Funktionen, die sie erfüllen muss. Erstens muss sie in Echtzeit die Sensordaten erfassen können. Dies erfordert sowohl eine effiziente Nutzung

der Sensoren als auch eine reibungslose Datenübertragung an die App. Zweitens müssen die erfassten Daten in einer SQLite-Datenbank abgespeichert werden, damit sie für zukünftige Analysen verfügbar sind. Drittens muss die App eine visuelle Darstellung der Sensordaten bieten. Hierfür kann der Benutzer durch Wischen zwischen zwei Diagrammen wechseln. Diese Diagramme sollen die erfassten Daten anschaulich und verständlich präsentieren.

Ein weiterer Aspekt der App ist das Energieeffizienzmanagement. Um den Akkuverbrauch möglichst gering zu halten, soll beim Wechsel zwischen den Diagrammen der jeweils nicht verwendete Sensor abgemeldet und der benötigte Sensor angemeldet werden. Dadurch wird unnötiger Energieverbrauch durch nicht verwendete Sensoren verhindert. Die gezielte Software Development Kit (SDK) Version ist für die Android Version 14 vorgesehen, wobei auch ältere Geräte durch eine minimale SDK-Version unterstützt werden.

Diese umfassende Beschreibung der Anforderungen und Spezifikationen stellt die Basis für die erfolgreiche Entwicklung und korrekte Implementierung der App zur Erfassung und Darstellung von Sensordaten.

3.2 Entwicklung der Smartphone-/Smartwatch-App

Nach der Definition der Anforderungen und Spezifikationen folgt nun die detaillierte Beschreibung der Implementierung der Smartphone-/Smartwatch-App. Im folgenden Kapitel werden die für die Entwicklung verwendeten Dateien und Codeabschnitte erklärt. Hierbei werden die Programmcodes nur in kleinen Abschnitten gezeigt, die sich auf die wichtigsten Dateien beschränken und im Detail erläutert werden. Im Anhang können die Chatverläufe mit ChatGPT in chronologischer Reihenfolge gefunden werden.

3.2.1 Einführung

Die Implementierung dieser App verfolgt das Ziel, Sensordaten aus einem Smartphone oder einer Smartwatch auszulesen, abzuspeichern und grafisch darzustellen. Die Relevanz dieser Implementierung liegt vor allem in der Möglichkeit, mithilfe von ChatGPT

anspruchsvolle Programmieraufgaben zu evaluieren. Sollte ChatGPT in der Lage sein, diese Art von Aufgaben erfolgreich zu lösen, kann dies bedeuten, dass ChatGPT in Zukunft für die komplexe Automatisierung von Entwicklungsprozessen eingesetzt werden kann. Dies würde nicht nur Zeit und Kosten sparen, sondern auch die Effizienz in der Softwareentwicklung erhöhen.

```
android {
```

Abbildung 2: Definition SDK-Versionen in der build.gradle-Datei

```
    namespace = "com.example.smartphonesmartwatchapp"
    compileSdk = 34 // Festlegen der SDK-Version zum Kompilieren

    defaultConfig {
        applicationId = "com.example.smartphonesmartwatchapp" // Eindeutige Anwendungs-ID
        minSdk = 23 // Minimale unterstützte SDK-Version
        targetSdk = 34 // Ziel-SDK-Version
        versionCode = 1 // Versionscode der App
        versionName = "1.0" // Menschlich lesbarer Versionsname
    }
}
```

Quelle: Android Studio

3.2.2 Einrichtung der Entwicklungsumgebung

Die Anweisungen zur Installation von Android Studio stammen vollständig von ChatGPT. Zunächst wurde Android Studio von der offiziellen Webseite heruntergeladen und auf einem kompatiblen Windows Betriebssystem installiert. Um sicherzustellen, dass die App auf den meisten Geräten ohne Probleme läuft und dabei die neuesten Android-Funktionen und Sicherheitsstandards nutzt, wurden die notwendigen SDK-Versionen konfiguriert.

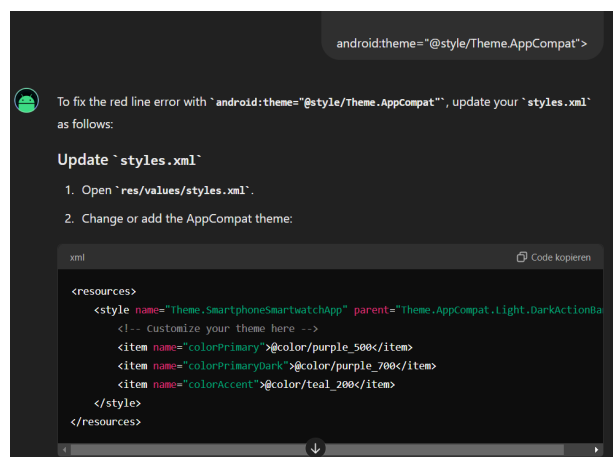
Um die Kompatibilität der App mit älteren Android-Versionen zu gewährleisten, wurde die minimale SDK-Version auf 23 gesetzt (siehe Abbildung 2 bzw. Markierungen). Die Ziel-SDK-Version wurde auf 34 festgelegt, um sicherzustellen, dass die App die neuesten Android-Features und Sicherheitsstandards nutzt (siehe ebd.). Damit bei der Entwicklung die neuesten Android-APIs genutzt werden können, wurde die compileSDK-Version ebenfalls auf 34 gesetzt (siehe ebd.).

Es waren bereits mehrere Plugins vorinstalliert, darunter unter anderem das Kotlin-Plugin, das GitLab-Plugin und viele andere. Zudem wurde das Plugin SimpleSQLiteBrowser heruntergeladen, damit die SQLite-Datenbank in Android Studio geöffnet und verwaltet werden kann. Dies war besonders wichtig, um die erfassten Sensordaten effektiv speichern und analysieren zu können.

3.2.3 Integration von ChatGPT

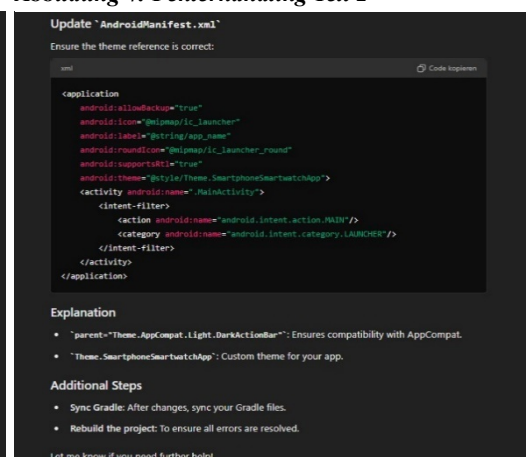
Um einen effektiven Entwicklungsprozess zu garantieren, wurden ChatGPT alle notwendigen Informationen gegeben, um das Thema vollständig zu verstehen. Danach wurde der Implementierungsprozess schrittweise durchgeführt. Nach und nach wurde jeder Teil der App entwickelt und bei Bedarf nachgebessert (siehe Anhang). Wenn Fehler auftraten, wurde der fehlerhafte Code mit der exakten Fehlerposition an ChatGPT übermittelt, damit es den Fehler identifizieren und beheben konnte (siehe ebd.). Ein Beispiel für eine typische Anfrage an ChatGPT wäre die Fehlermeldung und der dazugehörige Codeausschnitt, der zur Fehlermeldung geführt hat (siehe Abbildungen 3 und 4). ChatGPT hat daraufhin die gelieferten Informationen analysiert und einen passenden Lösungsvorschlag zur Behebung des Fehlers gegeben.

Abbildung 3: Fehlerhandling Teil 1



Quelle: ChatGPT bzw. Android Studio Developer

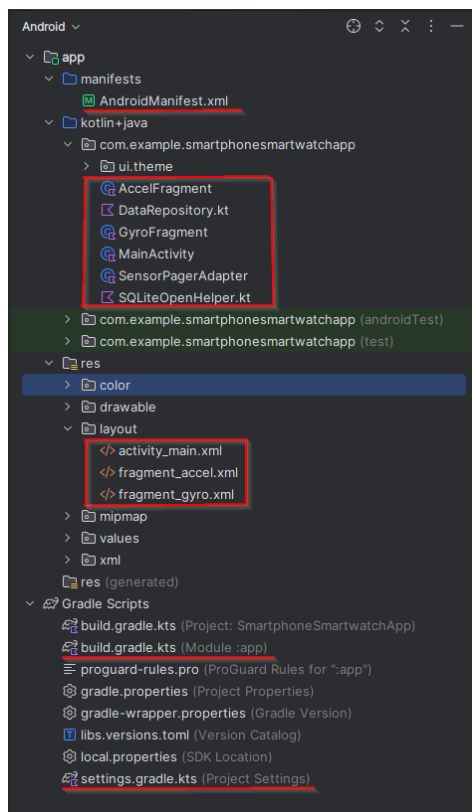
Abbildung 4: Fehlerhandling Teil 2



3.2.4 Projektstruktur und Konfigurationsdateien

Um eine einfache Navigation und Verwaltung zu ermöglichen, wurde die Verzeichnisstruktur des Projekts klar und organisiert angelegt. Ein Screenshot der Verzeichnisstruktur zeigt die wichtigsten Dateien, die für die Entwicklung der App notwendig sind (siehe Abbildung 5 bzw. Markierungen). Diese befinden sich in den Verzeichnissen `app/manifests/`, `app/kotlin+java/com.example.smartphonesmartwatchapp/`, `app/res/layout/` und Gradle Scripts.

Abbildung 5: Verzeichnis



3.2.5 AndroidManifest.xml

Quelle: Android Studio

Die `AndroidManifest.xml`-Datei definiert grundlegende Informationen über die App, wie ihre Komponenten (Aktivitäten, Dienste, etc.) und die benötigten Berechtigungen.

Abbildung 6: AndroidManifest.xml Datei

```

<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.smartphonesmartwatchapp">
    <!-- Berechtigungen, die von der App benötigt werden -->
    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.BODY_SENSORS"/>
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.HIGH_SAMPLING_RATE_SENSORS"/>
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>

    <application
        android:allowBackup="true"
        android:icon="@drawable/_9_03_59"
        android:label="SmartphoneSmartwatchApp"
        android:roundIcon="@drawable/_9_03_59"
        android:supportsRtl="true"
        android:theme="@style/Theme.SmartphoneSmartwatchApp">

        <activity android:name=".MainActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
    </application>
</manifest>

```

Quelle: Android Studio

In diesem Codeausschnitt kann gesehen werden, wie die notwendigen Berechtigungen wie Internetzugriff, Zugriff auf Körpersensoren und Berechtigungen für den Lese- und Schreibzugriff auf den externen Speicher definiert (siehe Abbildung 6 bzw. Markierungen). Zudem wird die Hauptaktivität der App deklariert, die beim Starten der App initialisiert wird. Die Berechtigungen sind notwendig, damit die App Zugriffsrechte erhält, um auf die Sensoren zuzugreifen und Daten zu speichern. Außerdem können in dieser Datei das Icon, das Label und weitere Attribute der App geändert werden (siehe ebd.).

3.2.6 build.gradle

Die build.gradle-Datei definiert die Abhängigkeiten und die Build-Konfigurationen des

Projekts. Diese Datei ist entscheidend für die Konfiguration des Projekts und die Verwaltung der Abhängigkeiten.

Abbildung 7: Definition der Bibliotheken

```
dependencies {
    // Core-Bibliotheken
    implementation(libs.androidx.core.ktx) // Android-Core-Erweiterungen für Kotlin
    implementation(libs.androidx.lifecycle.runtime.ktx) // Lifecycle-Laufzeit für Kotlin
    implementation(libs.androidx.activity.compose) // Compose-Unterstützung für Aktivitäten

    // Compose-Bibliotheken
    implementation(platform(libs.androidx.compose.bom)) // BOM für Compose-Bibliotheken
    implementation(libs.androidx.ui) // Haupt-UI-Bibliothek für Compose
    implementation(libs.androidx.ui.graphics) // Grafikbibliothek für Compose
    implementation(libs.androidx.ui.tooling.preview) // Vorschau-Tooling für Compose
    implementation(libs.androidx.material3) // Material Design-Komponenten für Compose

    // UI- und Kompatibilitätsbibliotheken
    implementation(libs.androidx.appcompat) // AppCompatActivity für Rückwärtskompatibilität

    // Testbibliotheken
    testImplementation(libs.junit) // JUnit für Unit-Tests
    androidTestImplementation(libs.androidx.junit) // JUnit-Erweiterungen für Android
    androidTestImplementation(libs.androidx.espresso.core) // Espresso für UI-Tests
    androidTestImplementation(platform(libs.androidx.compose.bom)) // BOM für Compose-Testabhängigkeiten
    androidTestImplementation(libs.androidx.ui.test.junit4) // JUnit4-Unterstützung für Compose-UI-Tests

    // Debugging und Tooling
    debugImplementation(libs.androidx.ui.tooling) // Tooling für das Debugging der Compose-UI
    debugImplementation(libs.androidx.ui.test.manifest) // Manifest-Unterstützung für Compose-UI-Tests

    // Zusätzliche Bibliotheken
    implementation(libs.mpandroidchart) // Diagrammbibliothek für Android
    implementation(libs.androidx.viewpager2) // ViewPager2 für durchblätterbare Seiten
    implementation(libs.material) // Materialkomponenten-
    implementation("androidx.sqlite.sqlite:2.1.0")
    implementation("androidx.sqlite.sqlite-framework:2.1.0")
}
```

Quelle: Android Studio

Dieser Code definiert die grundlegenden Build-Konfigurationen und Abhängigkeiten für das Projekt. Er stellt sicher, dass die notwendigen Bibliotheken und Plugins für die Entwicklung der App verfügbar sind (siehe Abbildung 7 bzw. Markierungen). Dazu gehören unter anderem die MPAndroidChart-Bibliothek für die Diagrammerstellung und die ViewPager2-Bibliothek für die Navigation (siehe ebd.).

3.2.7 DataRepository

Die DataRepository.kt-Datei verwaltet die Interaktionen mit der SQLite-Datenbank. Sie stellt Methoden zum Einfügen, Lesen und Verwalten von Sensordaten bereit.

Abbildung 8: DataRepository.kt Datei

```

class DataRepository(context: Context) {

    private val dbHelper: DatabaseHelper = DatabaseHelper(context)

    fun insertData(sensorType: String, xValue: Float, yValue: Float, zValue: Float, timestamp: String) {
        val db = dbHelper.writableDatabase
        val values = ContentValues().apply {
            put("sensorType", sensorType)
            put("xValue", xValue)
            put("yValue", yValue)
            put("zValue", zValue)
            put("timestamp", timestamp)
        }
        db.insert( table: "data", nullColumnHack: null, values)
    }

    fun getAllData(): List<Data> {
        val db = dbHelper.readableDatabase
        val cursor: Cursor = db.query(
            table: "data",
            arrayOf("id", "sensorType", "xValue", "yValue", "zValue", "timestamp"),
            selection: null, selectionArgs: null, groupBy: null, having: null, orderBy: null
        )

        val dataList = mutableListOf<Data>()
        with(cursor) {
            while (moveToNext()) {
                val id = getInt(getColumnIndexOrThrow("id"))
                val sensorType = getString(getColumnIndexOrThrow("sensorType"))
                val xValue = getFloat(getColumnIndexOrThrow("xValue"))
                val yValue = getFloat(getColumnIndexOrThrow("yValue"))
                val zValue = getFloat(getColumnIndexOrThrow("zValue"))
                val timestamp = getString(getColumnIndexOrThrow("timestamp"))
                dataList.add(Data(id, sensorType, xValue, yValue, zValue, timestamp))
            }
        }
        cursor.close()
        return dataList
    }
}

data class Data(val id: Int, val sensorType: String, val xValue: Float, val yValue: Float, val zValue: Float, val timestamp: String)

```

Quelle: Android Studio

In der Datei DataRepository.kt sind Methoden zum Einfügen von Sensordaten in die SQLite-Datenbank und zum Abrufen aller gespeicherten Daten enthalten. Die Methode insertData nimmt Sensordaten als Parameter entgegen und fügt diese in die Datenbank ein. Die Methode getAllData liest alle gespeicherten Datensätze aus der Datenbank aus und gibt sie als Liste zurück (siehe Abbildung 8 bzw. Markierungen).

3.2.8 SQLiteOpenHelper

Die SQLiteOpenHelper.kt-Datei bzw. die Klasse DatabaseHelper ist für die Erstellung und Verwaltung der SQLite-Datenbank zuständig. Sie definiert die Struktur der Datenbanktabellen und stellt Methoden zur Datenmanipulation bereit.

Abbildung 9: *SQLiteOpenHelper.kt* bzw. *DatabaseHelper*

```

class DatabaseHelper(context: Context) : SQLiteOpenHelper(context, DATABASE_NAME, factory: null, DATABASE_VERSION) {

    override fun onCreate(db: SQLiteDatabase) {
        val createTableSQL = """
            CREATE TABLE data (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                sensorType TEXT NOT NULL,
                xValue REAL,
                yValue REAL,
                zValue REAL,
                timestamp TEXT
            );
        """
        db.execSQL(createTableSQL)
    }

    override fun onUpgrade(db: SQLiteDatabase, oldVersion: Int, newVersion: Int) {
        db.execSQL("sql: \"DROP TABLE IF EXISTS data\"")
        onCreate(db)
    }

    companion object {
        private const val DATABASE_VERSION = 1
        private const val DATABASE_NAME = "sensor_data.db"
    }
}

```

Quelle: Android Studio

Die Klasse DatabaseHelper bzw. die SQLiteOpenHelper.kt Datei definiert die Struktur der Datenbank und stellt Methoden zum Erstellen und Aktualisieren der Datenbank bereit. Die Methode onCreate wird aufgerufen, wenn die Datenbank zum ersten Mal erstellt wird, und führt das SQL-Statement zum Erstellen der Tabelle aus (siehe Abbildung 9). Die Methode onUp-grade wird aufgerufen, wenn die Datenbankversion geändert wird, und ermöglicht die Aktualisierung der Datenbankstruktur (siehe ebd.).

3.2.9 SensorPagerAdapter

Die SensorPagerAdapter.kt-Datei verwaltet die Fragmente für die Anzeige der Sensordaten. Sie ermöglicht die Navigation zwischen verschiedenen Fragmenten innerhalb der App.

Abbildung 10: SensorPagerAdapter.kt Datei

```
// Adapter zum Verwalten mehrerer Fragmente in einem ViewPager2
class SensorPagerAdapter(
    activity: FragmentActivity, // Die Aktivität, die den ViewPager2 hostet
    private val fragments: List<Fragment> // Liste der anzuzeigenden Fragmente
) : FragmentStateAdapter(activity) {

    // Gibt die Anzahl der Fragmente zurück
    override fun getItemCount(): Int = fragments.size

    // Erstellt ein Fragment für die gegebene Position
    override fun createFragment(position: Int): Fragment = fragments[position]
}
```

Quelle: Android Studio

Mit Hilfe der Datei SensorPagerAdapter.kt, die einen Adapter darstellt, kann innerhalb der App zwischen verschiedenen Fragmenten navigiert werden. Der Adapter erhält eine Liste von Fragmenten und gibt das entsprechende Fragment basierend auf der aktuellen Position im ViewPager2 zurück. Er stellt sicher, dass die richtige Fragmentansicht angezeigt wird, wenn der Benutzer zwischen den Registerkarten wechselt (siehe Abbildung 10).

3.2.10 AccelFragment

Die AccelFragment.kt-Datei enthält die Logik für die Anzeige und Aktualisierung der Beschleunigungsmesserdaten in einem Diagramm.

Abbildung 12: AccelFragment.kt Datei Teil 1

```

class AccelFragment : Fragment() {

    // LineChart für die Anzeige der Beschleunigungsmesserdaten
    private lateinit var accelChart: LineChart
    private var accelEntries = ArrayList<Entry>() // Liste der Datenpunkte für das Diagramm
    private lateinit var accelDataSet: LineDataSet // DataSet für das Diagramm
    private lateinit var sensorManager: SensorManager // SensorManager für den Zugriff auf die Gerätesensoren
    private var accelSensor: Sensor? = null // Referenz zum Beschleunigungssensor
    private var lastUpdateTime: Long = 0 // Zeitstempel für die letzte Aktualisierung
    private val updateInterval: Long = 1000 // Intervall für die Aktualisierung in Millisekunden
    private var startTime: Long = 0 // Startzeit der Datenerfassung
    private var elapsedTimeOffset: Long = 0 // Offset zur Anpassung der vergangenen Zeit beim Fortsetzen

    // SensorEventListener zum Verarbeiten der Sensoraktualisierungen
    private val sensorEventListener = object : SensorEventListener {
        override fun onSensorChanged(event: SensorEvent?) {
            event?.let {
                val currentTime = System.currentTimeMillis()
                // Aktualisiere das Diagramm, wenn das Intervall überschritten ist
                if (currentTime - lastUpdateTime >= updateInterval) {
                    logSensorData(it.values)
                    updateGraph(it.values)
                    lastUpdateTime = currentTime
                }
            }
        }
    }
}

```

Quelle: Android Studio

Abbildung 11: AccelFragment.kt Datei Teil 2

```

// Aktualisiere das Diagramm mit neuen Beschleunigungsmesserdaten
fun updateGraph(values: FloatArray) {
    activity?.runOnUiThread {
        // Berechne die vergangene Zeit in Sekunden
        val elapsedTime = (System.currentTimeMillis() - startTime + elapsedTimeOffset) / 1000f
        // Füge einen neuen Datenpunkt mit der vergangenen Zeit und dem X-Wert hinzu
        accelEntries.add(Entry(elapsedTime, values[0]))
        accelDataSet.notifyDataSetChanged() // Benachrichtige, dass das DataSet aktualisiert wurde
        accelChart.data.notifyDataSetChanged() // Benachrichtige, dass die Daten geändert wurden
        accelChart.notifyDataSetChanged() // Benachrichtige das Diagramm, dass sich seine Daten geändert haben
        accelChart.invalidate() // Zeichne das Diagramm neu
    }
}

```

Quelle: Android Studio

Die AccelFragment.kt-Datei initialisiert das Diagramm zur Anzeige der Beschleunigungsmesserdaten und aktualisiert es bei jedem Sensorereignis. Der SensorEventListener erfasst die Daten vom Beschleunigungssensor und ruft die Methode updateGraph auf, um das Diagramm zu aktualisieren (Abbildung 11 und 12).

3.2.11 GyroFragment

Die GyroFragment.kt-Datei enthält die Logik für die Anzeige und Aktualisierung der Gyroskopdaten in einem Diagramm.

Abbildung 13:GyroFragment.kt Datei Teil 1

```

class GyroFragment : Fragment() {

    // LineChart für die Anzeige der Gyroskopdaten
    private lateinit var chart: LineChart
    private var gyroEntries = ArrayList<Entry>() // Liste der Datenpunkte für das Diagramm
    private lateinit var dataSet: LineDataSet // DataSet für das Diagramm
    private lateinit var sensorManager: SensorManager // SensorManager für den Zugriff auf die Gerätesensoren
    private var gyroSensor: Sensor? = null // Referenz zum Gyroskopsensor
    private var lastUpdateTime: Long = 0 // Zeitstempel für die letzte Aktualisierung
    private val updateInterval: Long = 1000 // Intervall für die Aktualisierung in Millisekunden
    private var startTime: Long = 0 // Startzeit der Datenerfassung
    private var elapsedTimeOffset: Long = 0 // Offset zur Anpassung der vergangenen Zeit beim Fortsetzen

    // SensorEventListener zum Verarbeiten der Sensoraktualisierungen
    private val sensorEventListener = object : SensorEventListener {
        override fun onSensorChanged(event: SensorEvent?) {
            event?.let {
                val currentTime = System.currentTimeMillis()
                // Aktualisiere das Diagramm, wenn das Intervall überschritten ist
                if (currentTime - lastUpdateTime >= updateInterval) {
                    logSensorData(it.values)
                    updateGraph(it.values)
                    lastUpdateTime = currentTime
                }
            }
        }
    }
}

```

Quelle: Android Studio

Abbildung 14: GyroFragment Teil 2

```

// Aktualisiere das Diagramm mit neuen Gyroskopdaten
fun updateGraph(values: FloatArray) {
    activity?.runOnUiThread {
        // Berechne die vergangene Zeit in Sekunden
        val elapsedTime = (System.currentTimeMillis() - startTime + elapsedTimeOffset) / 1000f
        // Füge einen neuen Datenpunkt mit der vergangenen Zeit und dem X-Wert hinzu
        gyroEntries.add(Entry(elapsedTime, values[0]))
        dataSet.notifyDataSetChanged() // Benachrichtige, dass das DataSet aktualisiert wurde
        chart.data.notifyDataChanged() // Benachrichtige, dass die Daten geändert wurden
        chart.notifyDataSetChanged() // Benachrichtige das Diagramm, dass sich seine Daten geändert haben
        chart.invalidate() // Zeichne das Diagramm neu
    }
}
}

```

Quelle: Android Studio

Die GyroFragment.kt-Datei initialisiert das Diagramm zur Anzeige der Gyroskopdaten und aktualisiert es bei jedem Sensorereignis. Der SensorEventListener erfasst die Daten vom Gyroskopsensor und ruft die Methode updateGraph auf, um das Diagramm zu aktualisieren (siehe Abbildung 13 und 14).

3.2.12 MainActivity

Die MainActivity.kt-Datei ist die zentrale Steuerungseinheit der App. Sie initialisiert die Sensoren, verwaltet die Fragmente und sorgt für die Datenspeicherung.

Abbildung 15: MainActivity.kt Datei Teil 1

```
// Hauptaktivität der App, die das SensorEventListener-Interface implementiert
class MainActivity : AppCompatActivity(), SensorEventListener {

    // SensorManager zum Verwalten der Sensoren
    private lateinit var sensorManager: SensorManager
    // Beschleunigungssensor
    private var accelerometer: Sensor? = null
    // Gyroskop-Sensor
    private var gyroscope: Sensor? = null

    // UI-Elemente: ViewPager und Fragmente für die Sensordatenanzeige
    private lateinit var viewPager: ViewPager2
    private lateinit var accelFragment: AccelFragment
    private lateinit var gyroFragment: GyroFragment

    // Datei-Writer für das Speichern der Sensordaten in einer Datei
    private var fileWriter: FileWriter? = null
    // Zeitstempel für die Aktualisierung der Sensordaten
    private var timestamp: Long = 0
    // Aktualisierungsintervall in Millisekunden (1 Sekunde)
    private val updateInterval: Long = 1000
    // Anfragecode für Berechtigungen
    private val REQUEST_CODE = 100

    // Repository für die Speicherung der Daten in einer SQLite-Datenbank
    private lateinit var dataRepository: DataRepository
```

Quelle: Android Studio

Abbildung 17: MainActivity.kt Datei Teil 2

```
// Methode, die aufgerufen wird, wenn sich die Sensordaten ändern
override fun onSensorChanged(event: SensorEvent?) {
    event?.let {
        val currentTime = System.currentTimeMillis()
        // Überprüfen, ob das Aktualisierungsintervall überschritten wurde
        if (currentTime - timestamp >= updateInterval) {
            val formattedTime = SimpleDateFormat("yyyy-MM-dd HH:mm:ss", Locale.getDefault()).format(Date(currentTime))
            when (it.sensor.type) {
                // Fall: Beschleunigungssensor
                Sensor.TYPE_ACCELEROMETER -> {
                    // Aktualisieren des Beschleunigungssensor-Graphs im Fragment
                    accelFragment.updateGraph(it.values)
                    // Speichern der Daten
                    saveData("Accelerometer", it.values, formattedTime)
                }
                // Fall: Gyroskop-Sensor
                Sensor.TYPE_GYROSCOPE -> {
                    // Aktualisieren des Gyroskop-Graphs im Fragment
                    gyroFragment.updateGraph(it.values)
                    // Speichern der Daten
                    saveData("Gyroscope", it.values, formattedTime)
                }
            }
            // Aktualisieren des Zeitstempels
            timestamp = currentTime
        }
    }
}

// Methode zum Speichern der Sensordaten in Datei und SQLite-Datenbank
private fun saveData(sensorType: String, values: FloatArray, time: String) {
    try {
        fileWriter?.apply {
            // Schreiben der Daten in die Datei
            write("$time - $sensorType: X=${values[0]}, Y=${values[1]}, Z=${values[2]}\n")
            flush()
        } ?: Log.e("FileWriter", "FileWriter nicht initialisiert")

        // Speichern der Daten in SQLite-Datenbank
        dataRepository.insertData(sensorType, values[0], values[1], values[2], time)
    } catch (e: IOException) {
        e.printStackTrace()
        Log.e("SaveDataError", "Fehler beim Speichern der Daten: ${e.message}")
    }
}
```

Quelle: Android Studio

Abbildung 16: MainActivity.kt Datei Teil 3

```
// Methode zum Einrichten des Datei-Writers
private fun setupFileWriter(append: Boolean) {
    try {
        // Datei im externen Speicher der App erstellen
        val file = File(getExternalFilesDir(null), "sensor_data.txt")
        Log.d("FilePath", "Dateipfad: ${file.absolutePath}")
        // Datei-Writer initialisieren
        fileWriter = FileWriter(file, append)
    } catch (e: IOException) {
        e.printStackTrace()
    }
}

// Methode zum Schließen des Datei-Writers
private fun closeFileWriter() {
    try {
        fileWriter?.close()
    } catch (e: IOException) {
        e.printStackTrace()
    }
}
```

Quelle: Android Studio

Die MainActivity.kt-Datei initialisiert die Sensoren, verwaltet die Fragmente und sorgt für die Datenspeicherung. Sie implementiert das SensorEventListener-Interface, um Sensorereignisse zu empfangen und zu verarbeiten. Die Methode onSensorChanged wird aufgerufen, wenn sich die Sensordaten ändern, und aktualisiert die Diagramme sowie die gespeicherten Daten. Die Methoden setupFileWriter und closeFileWriter verwalten die Dateioperationen zur Speicherung der Sensordaten (siehe Abbildung 15, 16 und 17).

3.2.13 activity_main.xml

Die Datei activity_main.xml definiert das Hauptlayout der App. Sie enthält ein LinearLayout, das die Hauptkomponenten der Benutzeroberfläche organisiert:

Abbildung 18: activity_main.xml Datei

```

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="16dp">

    <!-- TabLayout zur Anzeige der Sensorregisterkarten -->
    <com.google.android.material.tabs.TabLayout
        android:id="@+id/tabLayout"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        app:tabIndicatorColor="@color/purple_500"
        app:tabSelectedTextColor="@color/black"
        app:tabTextColor="@color/gray"
        app:tabMode="fixed"
        app:tabGravity="fill" />

    <!-- ViewPager2 zur Aufnahme der Sensorfragmente -->
    <androidx.viewpager2.widget.ViewPager2
        android:id="@+id/viewPager"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1" />

    <!-- TextView zur Anzeige zusätzlicher Sensordaten (derzeit ausgeblendet) -->
    <TextView
        android:id="@+id/sensorData"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Sensor Data"
        android:textSize="18sp"
        android:layout_marginTop="16dp"
        android:visibility="gone" />

```

Quelle: Android Studio

Das TabLayout dient zur Anzeige der Registerkarten für die verschiedenen Sensoren, während der ViewPager2 die Fragmente für die Sensoranzeige verwaltet und das Durchblättern zwischen den verschiedenen Sensordaten ermöglicht (siehe Abbildung 18).

3.2.14 fragment_accel.xml

Die Datei fragment_accel.xml definiert das Layout für das Beschleunigungssensor-Fragment:

Abbildung 19: fragment_accel.xml Datei

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <!-- LineChart zur Anzeige der Beschleunigungsmesserdaten -->
    <com.github.mikephil.charting.charts.LineChart
        android:id="@+id/accelChart"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_margin="16dp" /> <!-- Rand für bessere Abstände -->

</RelativeLayout>
```

Quelle: Android Studio

Das LineChart dient zur Anzeige der Beschleunigungsmesserdaten als Liniendiagramm. Es visualisiert die Daten des Beschleunigungssensors in Echtzeit (siehe Abbildung 19).

3.2.15 fragment_gyro.xml

Die Datei fragment_gyro.xml definiert das Layout für das Gyroskop-Fragment:

Abbildung 20: fragment_gyro.xml Datei

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <!-- LineChart zur Anzeige der Gyroskopdaten -->
    <com.github.mikephil.charting.charts.LineChart
        android:id="@+id/gyroChart"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_margin="16dp" /> <!-- Rand für bessere Abstände -->

</RelativeLayout>
```

Quelle: Android Studio

Das LineChart dient zur Anzeige der Gyroskopdaten als Liniendiagramm. Es visualisiert die Daten des Gyroskopsensors in Echtzeit (siehe Abbildung 20).

3.2.16 Darstellung der Graphen und der Benutzeroberfläche

In diesem Abschnitt werden die erstellten Graphen und die Benutzeroberfläche der App gezeigt. Die Screenshots veranschaulichen, wie die Sensordaten in der App visualisiert werden.

Beschleunigungsmesserdaten im AccelFragment

Dieser Screenshot zeigt das LineChart mit den aktuellen Beschleunigungsmesserdaten als Liniendiagramm. Die Daten werden in Echtzeit erfasst und visualisiert, um dem Benutzer einen sofortigen Überblick über die Bewegungen des Geräts zu geben (siehe Abbildung 21).

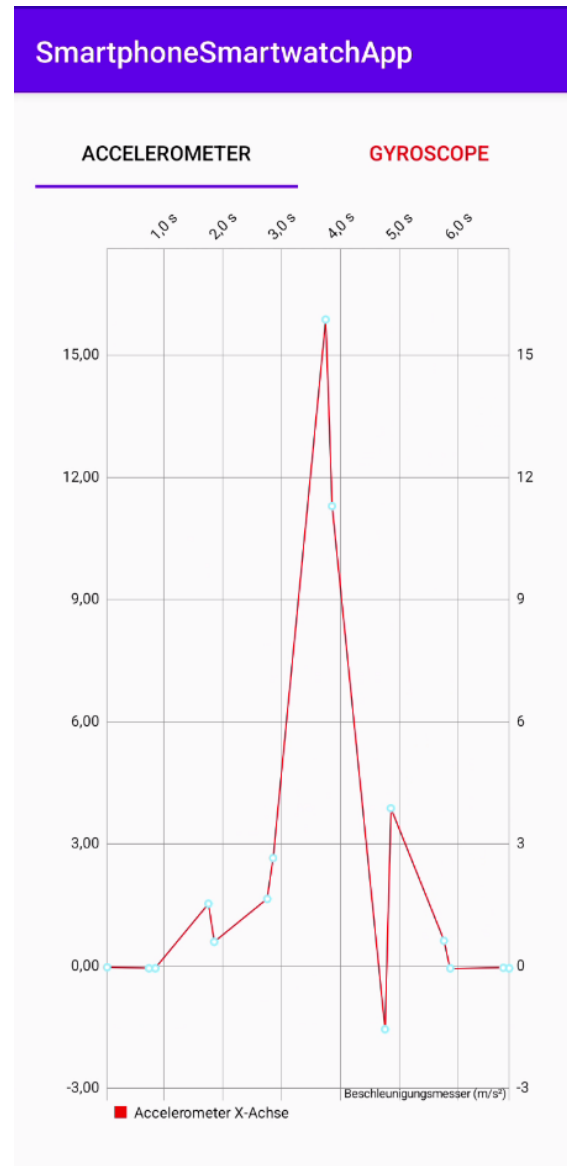
Gyroskopdaten im GyroFragment

Dieser Screenshot zeigt das LineChart mit den aktuellen Gyroskopdaten als Liniendiagramm. Auch hier

werden die Daten in Echtzeit erfasst und angezeigt, was dem Benutzer hilft, die Rotationsbewegungen des Geräts zu überwachen. (siehe Abbildung 22)

Die Benutzeroberfläche besteht aus verschiedenen Komponenten, die nahtlos zusammenarbeiten, um die Sensordaten in Echtzeit anzuzeigen. Der TabLayout und ViewPager2 ermöglichen die Navigation zwischen

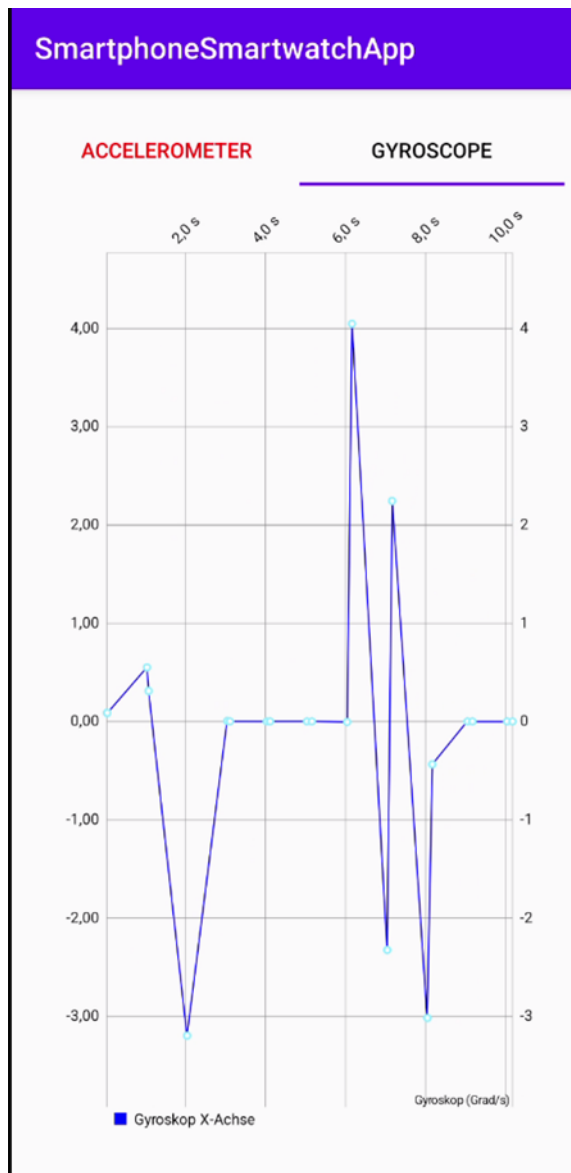
Abbildung 21: Diagramm in der App - Beschleunigungssensor



Quelle: Smartphone/Smartwatch App

den Sensorfragmenten, während die LineChart-Elemente in den Fragmenten die Sensordaten grafisch darstellen.

Abbildung 22: Diagramm in der App - Gyroskop

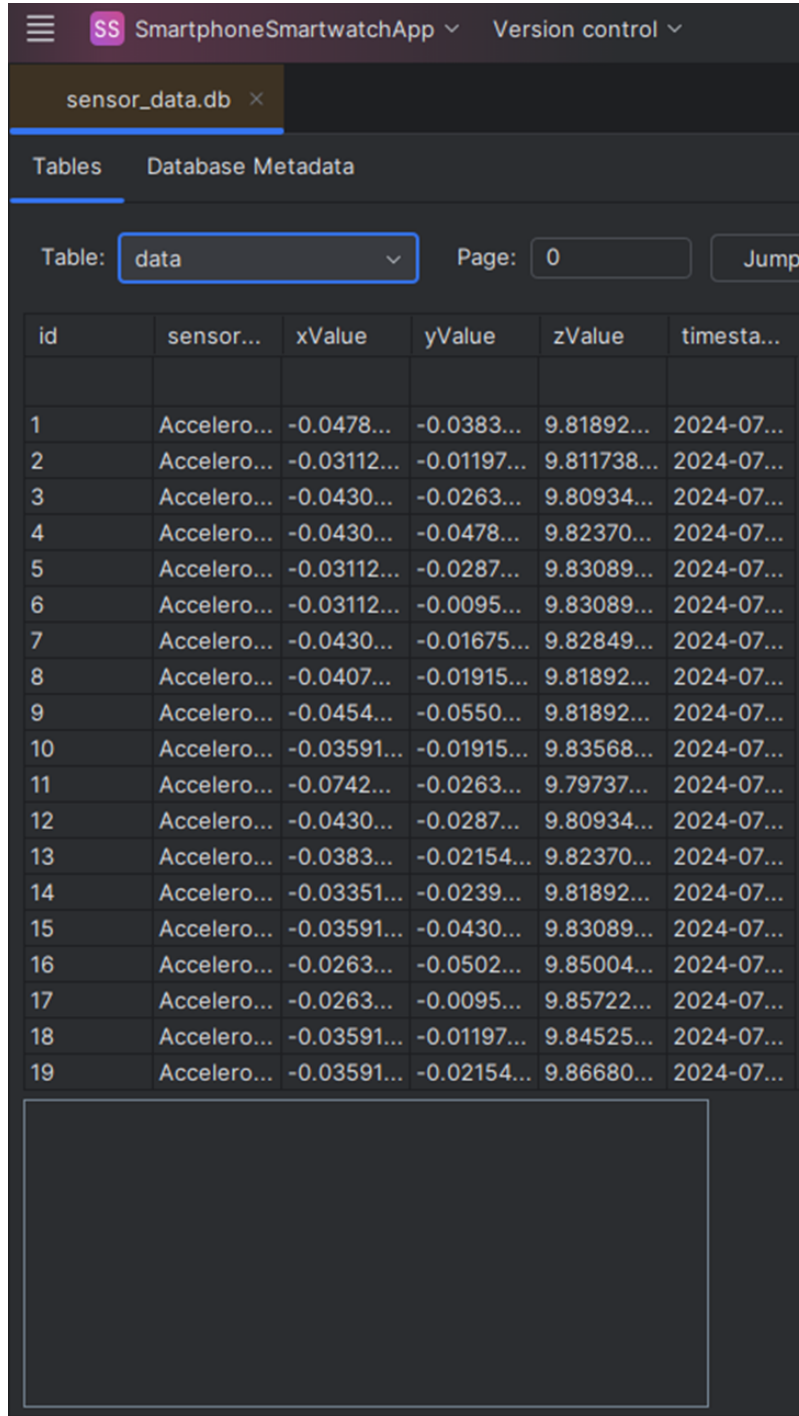


Quelle: Smartphone/Smartwatch App

3.2.17 Verwendung des SimpleSqliteBrowser Plugins

Die Daten werden in einer SQLite-Datenbank gespeichert, und das Plugin SimpleSqlite-Browser wurde verwendet, um diese Daten innerhalb von Android Studio zu visualisieren. Der Screenshot unten zeigt, wie die gespeicherten Sensordaten in der Datenbank angezeigt werden können. Die Tabelle data zeigt verschiedene Spalten, darunter die ID, den Sensortyp, die X-, Y- und Z-Werte sowie den Zeitstempel, an dem die Daten erfasst wurden. Durch Anklicken der einzelnen Einträge können die genauen Datenwerte und deren Details eingesehen werden (siehe Abbildung 23).

Abbildung 23: SimpleSqliteBrowser Plugin



SmartphoneSmartwatchApp Version control

sensor_data.db

Tables Database Metadata

Table: data Page: 0 Jump

id	sensor...	xValue	yValue	zValue	timesta...
1	Accelerometer	-0.0478...	-0.0383...	9.81892...	2024-07...
2	Accelerometer	-0.03112...	-0.01197...	9.811738...	2024-07...
3	Accelerometer	-0.0430...	-0.0263...	9.80934...	2024-07...
4	Accelerometer	-0.0430...	-0.0478...	9.82370...	2024-07...
5	Accelerometer	-0.03112...	-0.0287...	9.83089...	2024-07...
6	Accelerometer	-0.03112...	-0.0095...	9.83089...	2024-07...
7	Accelerometer	-0.0430...	-0.01675...	9.82849...	2024-07...
8	Accelerometer	-0.0407...	-0.01915...	9.81892...	2024-07...
9	Accelerometer	-0.0454...	-0.0550...	9.81892...	2024-07...
10	Accelerometer	-0.03591...	-0.01915...	9.83568...	2024-07...
11	Accelerometer	-0.0742...	-0.0263...	9.79737...	2024-07...
12	Accelerometer	-0.0430...	-0.0287...	9.80934...	2024-07...
13	Accelerometer	-0.0383...	-0.02154...	9.82370...	2024-07...
14	Accelerometer	-0.03351...	-0.0239...	9.81892...	2024-07...
15	Accelerometer	-0.03591...	-0.0430...	9.83089...	2024-07...
16	Accelerometer	-0.0263...	-0.0502...	9.85004...	2024-07...
17	Accelerometer	-0.0263...	-0.0095...	9.85722...	2024-07...
18	Accelerometer	-0.03591...	-0.01197...	9.84525...	2024-07...
19	Accelerometer	-0.03591...	-0.02154...	9.86680...	2024-07...

Quelle: Android Studio

Das Plugin ermöglicht es, die Struktur und den Inhalt der Datenbanktabellen einfach zu durchsuchen und zu verwalten. Dies erleichtert die Überprüfung und Analyse der gesammelten Sensordaten direkt in der Entwicklungsumgebung (siehe ebd.).

3.2.18 Zusammenfassung

Diese ausführliche Beschreibung der Implementierung zeigt die Dateien und Methoden, die zur Erstellung der Smartphone-/Smartwatch-App notwendig waren. Der Entwicklungsprozess umfasste die Integration von Sensoren, die Verarbeitung und Speicherung der Daten sowie deren Visualisierung und Navigation auf mobilen Geräten.

3.3 Architektur des Programms

Die entwickelte App basiert auf der Model-View-Controller (MVC) Architektur, die eine klare Unterscheidung zwischen der Logik und der Benutzeroberfläche bietet (vgl. Garcia 2023; vgl. Muntenescu 2016). Durch diese Unterscheidung wird die Wartbarkeit und Erweiterbarkeit des Codes erleichtert, da unabhängig von der Logik Änderungen an der Benutzeroberfläche vorgenommen werden können und umgekehrt (vgl. Garcia 2023).

Hauptkomponenten

Die Hauptkomponenten der App können sich in drei Kategorien unterteilen lassen: Controller, Model und View.

Controller:

- **MainActivity:** Die MainActivity dient als zentrale Steuerungseinheit und Initialisierungspunkt der App. Sie ist für die Einrichtung des SensorManagers und die Verwaltung der Fragmente verantwortlich. Die MainActivity fungiert als Vermittler zwischen der Verarbeitung der Sensordaten und den Benutzereingaben.

- **SensorPagerAdapter:** Die einzelnen Fragmente der App werden durch die SensorPagerAdapter-Komponente verwaltet. Dadurch ist das nahtlose Wechseln zwischen verschiedenen Ansichten, in denen die Sensordaten dargestellt werden, möglich.

Model:

- **DataRepository:** Für das Verwalten der Zugriffe auf die Datenbank ist das DataRepository zuständig. Dieses steht in direkter Verbindung mit dem DatabaseHelper, um Sensordaten zu speichern und abzurufen. Die Datenverwaltung wird durch die Kapselung der Datenbankoperationen im DataRepository abstrahiert und zentralisiert.
- **DatabaseHelper:** Diese Klasse ist verantwortlich für die Erstellung und Verwaltung der SQLite-Datenbank. Der DatabaseHelper definiert die Struktur der Datenbanktabellen und stellt Methoden zur Verfügung für die Datenmanipulation.

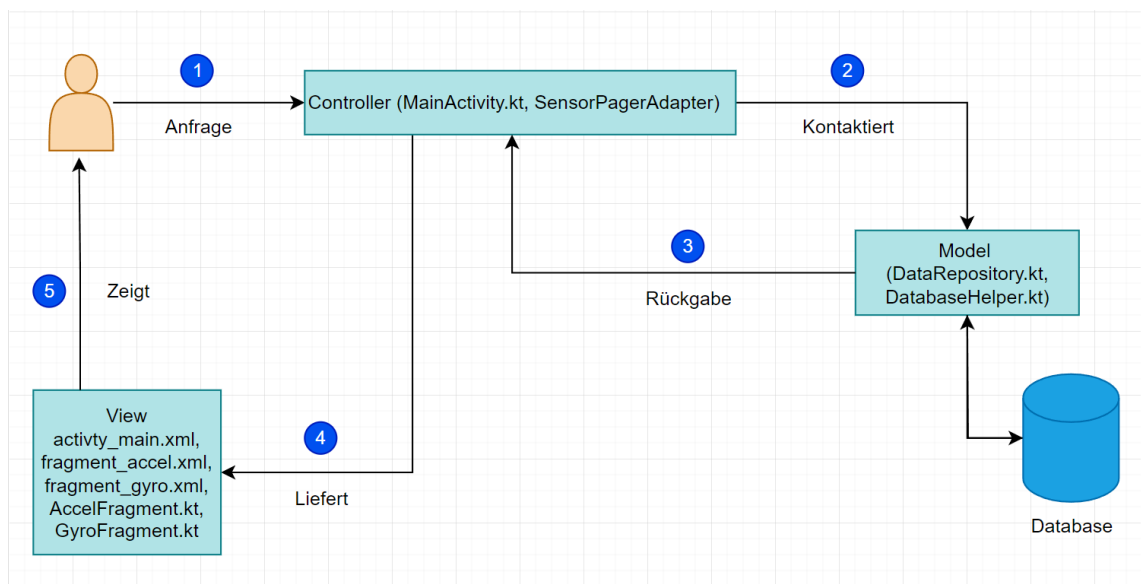
View:

- **XML-Layouts:** Verschiedene XML-Layouts definieren die Benutzeroberfläche der App, darunter `activity_main.xml`, `fragment_accel.xml` und `fragment_gyro.xml`. Diese Layouts bestimmen die Anordnung und das Design der Benutzeroberflächenelemente.
- **Fragmente:** Für die Interaktion mit dem Nutzer und für die Darstellung der Sensordaten sind Fragmente zuständig. Jedes Fragment innerhalb der App repräsentiert eine spezifische Ansicht, beispielsweise die Anzeige von Beschleunigungs- oder Gyroskopdaten.

Interaktionen zwischen den Komponenten

Entscheidend für das reibungslose Funktionieren der App sind die Interaktionen zwischen den einzelnen Komponenten.

Abbildung 24: MVC-Modell



Quelle: In Anlehnung an GeeksforGeeks, 2024

1. Benutzeranfrage: Der Benutzer interagiert mit der App, beispielsweise durch das Drücken eines Buttons oder das Navigieren zwischen Ansichten (siehe Abbildung 24).
2. Controller: Die Anfrage des Benutzers wird vom Controller, bestehend aus der MainActivity und dem SensorPagerAdapter, entgegengenommen. Der Controller verarbeitet die Anfrage und leitet sie gegebenenfalls an das Model weiter (siehe ebd.).
3. Model: Das Model, bestehend aus dem DataRepository und dem DatabaseHelper, ist für die Datenverarbeitung zuständig. Es speichert die Sensordaten in der Datenbank und stellt sie bei Bedarf dem Controller zur Verfügung (siehe ebd.).
4. Rückgabe: Die vom Model verarbeiteten Daten werden an den Controller zurückgegeben (siehe ebd.).

5. View: Der Controller aktualisiert die View-Komponenten basierend auf den erhaltenen Daten. Die View zeigt die aktualisierten Sensordaten an den Benutzer an (siehe ebd.).

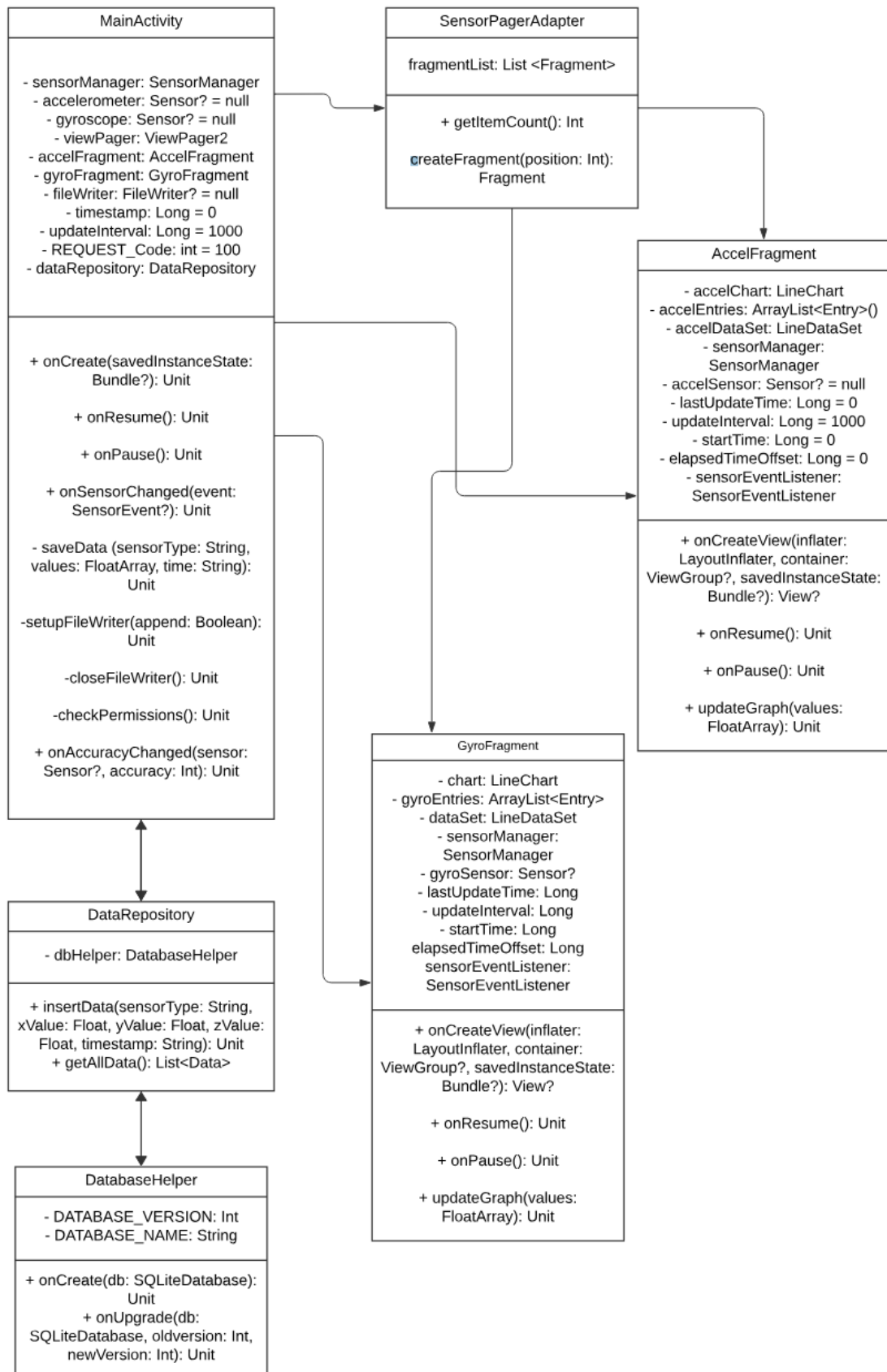
Die Verwendung des MVC-Modells in der App bietet eine erleichterte Wartbarkeit und Erweiterbarkeit des Codes (vgl. Garcia 2023). Zudem ermöglicht es eine klare Trennung der Verantwortlichkeiten. Durch die Kapselung der Datenbankoperationen im DataRepository und die Steuerung durch die MainActivity bleibt die Struktur der App übersichtlich und modular. Durch die Fragmente und XML-Layouts wird eine flexible und ansprechende Benutzeroberfläche geschaffen, die einfach an unterschiedliche Anforderungen angepasst werden kann.

Um die Struktur und Interaktionen der verschiedenen Klassen innerhalb der App noch deutlicher zu machen, wird nun ein UML-Klassendiagramm vorgestellt.

UML-Klassendiagramm

Das UML-Klassendiagramm zeigt, die Klassen, ihre Attribute und Methoden sowie die Beziehungen zwischen den Klassen. Dies hilft, die Architektur und die Zusammenhänge innerhalb der App auf einer detaillierten Ebene zu verstehen.

Abbildung 25: UML-Klassendiagramm



Quelle: Eigene Darstellung

Erklärung der UML-Klassendiagramm-Komponente und deren Beziehungen

MainActivity

- **Beschreibung:** Die Hauptaktivität der App, die die Benutzeroberfläche initialisiert und Sensorereignisse verwaltet. Sie ist die zentrale Steuerungseinheit und verwaltet die Sensoren, Fragmente und die Datenverwaltung.
- **Verbindung:**
 - Verwendet SensorPagerAdapter, um Fragmente zu verwalten.
 - Interagiert direkt mit AccelFragment und GyroFragment zur Anzeige von Sensordaten.
 - Verwendet DataRepository, um Sensordaten zu speichern und abzurufen.

AccelFragment

- **Beschreibung:** Fragment zur Anzeige und Aktualisierung der Beschleunigungsmesserdaten.
- **Verbindung:**
 - Wird von SensorPagerAdapter verwaltet.
 - Interagiert mit MainActivity für Sensordatenaktualisierungen.

GyroFragment

- **Beschreibung:** Fragment zur Anzeige und Aktualisierung der Gyroskopdaten.
- **Verbindung:**
 - Wird von SensorPagerAdapter verwaltet.
 - Interagiert mit MainActivity für Sensordatenaktualisierungen.

SensorPagerAdapter

- **Beschreibung:** Adapter zur Verwaltung der Fragmente, die die Sensordaten anzeigen. Er ermöglicht die Navigation zwischen den verschiedenen Fragmenten innerhalb der App.

- **Verbindung:**
 - Wird von MainActivity verwendet, um Fragmente zu verwalten.
 - Verwaltet AccelFragment und GyroFragment.

DataRepository

- **Beschreibung:** Verwaltet die Datenbankoperationen, speichert und ruft Sensordaten ab.
- **Verbindung:** Verwendet DatabaseHelper zur Verwaltung der SQLite-Datenbank.

DatabaseHelper

- **Beschreibung:** Hilfsklasse zur Verwaltung der SQLite-Datenbank, erstellt und aktualisiert die Datenbankstruktur.
- **Verbindung:** Wird von DataRepository für Datenbankoperationen verwendet.

Diese Struktur und die Beziehungen zwischen den Klassen zeigen die zentrale Rolle der MainActivity bei der Verwaltung der App und der Interaktion mit den anderen Komponenten, um die Sensordaten zu erfassen, anzuzeigen und zu speichern.

4 Gegenüberstellung der Ansätze

Nachdem ChatGPT die App erstellt hat, wird im folgenden Kapitel die Leistung und Effektivität von ChatGPT bei der Programmierung der App umfassend bewertet. Der Bewertungsprozess fokussiert sich auf die fünf zentralen Forschungsfragen dieser Arbeit. Es wurden zwei spezifische Kriterien pro Frage festgelegt, die zur Beantwortung und Bewertung genutzt werden. Zudem bieten diese Kriterien eine klare Struktur, um genau analysieren und dokumentieren zu können, welche Stärken und Schwächen, das durch ChatGPT generierte Programm hat.

Die Bewertung beinhaltet verschiedene Aspekte wie die funktionale Vollständigkeit und Fehlerfreiheit des Programms, die Verständlichkeit des generierten Codes und der

Ausgaben von ChatGPT, die Häufigkeit und den Aufwand der Anfragen sowie die Genauigkeit der Eingaben. Grund dafür ist, dass durch die detaillierte Analyse herausgearbeitet werden soll, inwieweit sich ChatGPT als Werkzeug zur Entwicklung von Anwendungen eignet und welche Herausforderungen beziehungsweise Möglichkeiten dabei bestehen.

4.1 Erstellung des Programms durch ChatGPT

Forschungsfrage: Inwieweit kann ChatGPT ein Programm erstellen, das den spezifizierten funktionalen Anforderungen entspricht?

4.1.1 Funktionale Vollständigkeit

Zuallererst wird die Bewertungsskala für die funktionale Vollständigkeit festgelegt:

- **Vollständig:** Alle vorgesehenen Funktionen sind implementiert und funktionieren korrekt.
- **Teilweise vollständig:** Die meisten vorgesehenen Funktionen sind implementiert, aber einige funktionieren nicht korrekt oder fehlen.
- **Unvollständig:** Viele der vorgesehenen Funktionen fehlen oder funktionieren nicht korrekt.

Bewertung: 4.1.1 Funktionale Vollständigkeit

Vollständig: ChatGPT hat die grundlegende Aufgabe erfolgreich erfüllt, Daten von einem Smartphone auszulesen, zu speichern und grafisch darzustellen. Die Daten werden in einer Datenbankdatei gespeichert, die zugreifbar ist und alle bisher gespeicherten Daten anzeigt. Außerdem können die Sensordaten im Logcat von Android Studio angezeigt und abgelesen werden. Die Grafiken werden vollständig dargestellt, wobei es gelegentlich zu Problemen kommt, dass sie ineinander geraten. Da dies nicht jedes Mal auftritt, hat ChatGPT alle Funktionen korrekt implementiert.

4.1.2 Fehlerfreiheit des Programms

Die Bewertungsskala für die Fehlerfreiheit des Programms lautet:

- **Keine Fehler:** Das Programm läuft fehlerfrei.
- **Moderate Fehler:** Das Programm enthält einige Fehler, die die Funktionalität nicht wesentlich beeinträchtigen.
- **Viele Fehler:** Das Programm weist zahlreiche Fehler auf, die die Funktionalität stark beeinträchtigen.

Bewertung: 4.1.2 Fehlerfreiheit des Programms

Moderate Fehler: ChatGPT weist nur einen kleinen Fehler auf, der das Programm nicht stark beeinträchtigt. Das auftretende Problem ist, dass die Grafiken gelegentlich ineinander geraten, was die Darstellung beeinträchtigt. Insgesamt funktioniert die App wie vorgesehen, hat aber ein kleines Performanceproblem, das ChatGPT nicht lösen konnte, da dieser Fehler nicht immer aufgetreten ist.

4.2 Verständlichkeit des Programms

Forschungsfrage: Wie verständlich sind das von ChatGPT erstellte Programm und dessen erklärende Antworten für den Benutzer?

4.2.1 Klarheit des Programmcodes

Die Bewertungsskala für die Klarheit des Programmcodes lautet:

- **Sehr klar:** Der Programmcode ist gut strukturiert, kommentiert und leicht verständlich.
- **Moderat klar:** Der Programmcode ist größtenteils verständlich, es gibt jedoch einige unklare oder schlecht kommentierte Abschnitte.
- **Unklar:** Der Programmcode ist schwer verständlich und schlecht dokumentiert.

Bewertung: 4.2.1 Klarheit des Programmcodes

Moderat klar: Der Programmcodes ist größtenteils verständlich, es gibt jedoch einige unklare oder schlecht kommentierte Abschnitte. Daher ist eine erneute Abfrage notwendig, um diese Teile besser zu verstehen. Es muss ausdrücklich der KI mitgeteilt werden, dass Kommentare zum Code hinzugefügt werden sollen, da ChatGPT dies nicht eigenständig tut.

4.2.2 Verständlichkeit der ChatGPT-Antworten

Die Bewertungsskala für die Verständlichkeit der ChatGPT-Antworten lautet:

- **Sehr verständlich:** Die Antworten von ChatGPT sind klar, präzise und leicht nachvollziehbar.
- **Moderat verständlich:** Die Antworten von ChatGPT sind größtenteils verständlich, es gibt jedoch einige komplexe oder unklare Erklärungen.
- **Unverständlich:** Die Antworten von ChatGPT sind schwer nachvollziehbar und verwirrend.

Bewertung: 4.2.2 Verständlichkeit der ChatGPT-Antworten

Moderat verständlich: Die Antworten, die ChatGPT auf unsere Anfragen gibt, sind größtenteils verständlich. Jedoch gibt es im Verlauf des Chats einige komplexe oder unklare Erklärungen. Anfangs erklärt die KI alles ausführlich, aber im Laufe der Zeit werden die Erklärungen weniger detailliert. Außerdem zeigt ChatGPT oft nur den geänderten Teil des Programmcodes, ohne genau zu erklären, wo bzw. in welcher Datei die Änderungen vorgenommen werden sollten, was zu Verwirrung führen kann.

4.3 Häufigkeit der Anfragen und Korrekturbedarf

Forschungsfrage: Wie oft muss der Benutzer seine Anfragen an ChatGPT umformulieren, um die gewünschte Programmfunktionalität zu erreichen?

4.3.1 Häufigkeit der Anfragen

Die Bewertungsskala für die Häufigkeit der Anfragen lautet:

- **0 = nie:** Die KI gibt sofort die richtige Antwort.
- **1-2 = wenig:** Es sind ein oder zwei Versuche nötig, um die richtige Antwort zu erhalten.
- **3-4 = moderat:** Es sind drei bis vier Versuche nötig, um die richtige Antwort zu erhalten.
- **>= 5 = viel:** Es sind fünf oder mehr Versuche nötig, um die richtige Antwort zu erhalten.

Bewertung: 4.3.1 Häufigkeit der Anfragen

Viele Anfragen: Die Häufigkeit der erneuten Anfragen zu derselben Frage variiert stark. Bei eher einfachen Themen, wie z. B. Anleitungen zum Herunterladen von Android Studio, sind keine weiteren Anfragen nötig, da alles gut erklärt und verständlich ist. Allerdings hat sich die Häufigkeit der Anfragen beim Programmieren deutlich erhöht. In einigen Fällen waren es 3-4 Anfragen, oft jedoch mehr als 5, da ChatGPT entweder den Code nicht geändert hat oder der Code dazu geführt hat, dass die App abgestürzt ist. Die erneute Anfrage hängt somit stark vom Umfang und der Schwierigkeit des Themas ab. Allerdings mussten im Durchschnitt viele Anfragen zur selben Frage gestellt werden.

4.3.2 Zeitaufwand für Korrektur und Validierung

Die Bewertungsskala für den Zeitaufwand für Korrektur und Validierung lautet:

- **Geringer Zeitaufwand:** Weniger als 5 Minuten pro Antwort.
- **Moderater Zeitaufwand:** 5-15 Minuten pro Antwort.
- **Hoher Zeitaufwand:** Mehr als 15 Minuten pro Antwort.

Bewertung: 4.3.2 Zeitaufwand für Korrektur und Validierung

Hoher Zeitaufwand: Der Korrekturaufwand für Fehler von ChatGPT hängt stark vom Umfang und der Schwierigkeit des Themas ab. Bei einigen Programmteilen war der Zeitaufwand gering. Oft jedoch erforderte es einen hohen Zeitaufwand, da viele Fehler von ChatGPT gemacht wurden, die zu einem Absturz des Programms führten oder dazu, dass es gar nicht startete.

4.4 Präzision des Inputs und Konsistenz der Antworten

Forschungsfrage: Wie beeinflusst die Präzision der Benutzereingaben die Qualität der von ChatGPT generierten Antworten und Programme?

4.4.1 Präzision des Inputs

Die Bewertungsskala für die Präzision des Inputs lautet:

- **Gering:** Allgemeiner Input reicht aus.
- **Moderat:** Klare und einigermaßen genaue Informationen sind erforderlich.
- **Hoch:** Höchste Präzision ist notwendig.

Bewertung: 4.4.1 Präzision des Inputs

Hohe Präzision: Der Input, den wir ChatGPT gegeben haben, musste beim Programmieren sehr präzise sein, da die Verständlichkeit der Antworten von ChatGPT stark beeinträchtigt wurde und ChatGPT dadurch auch mehr macht als eigentlich gefordert. Das Programm war bei hoher Präzision wahrscheinlicher korrekt als bei allgemeineren Anfragen. Bei einfachen Fragen, wie dem Herunterladen von Android Studio, reicht allgemeiner Input völlig aus. Im Durchschnitt musste die Anfrage für das Programmieren eine hohe Präzision aufweisen, damit ChatGPT das tut, was von ihm verlangt wird.

4.4.2 Konsistenz der Antworten

Die Bewertungsskala für die Konsistenz der Antworten lautet:

- **Sehr konsistent:** Die KI liefert über mehrere Anfragen hinweg gleichbleibend präzise Antworten.
- **Teilweise konsistent:** Die KI liefert überwiegend konsistente Antworten mit gelegentlichen Abweichungen.
- **Inkonsistent:** Die Antworten der KI variieren stark und sind unzuverlässig.

Bewertung: 4.4.2 Konsistenz der Antworten

Inkonsistent: In der Programmierung waren die Antworten von ChatGPT sehr unzuverlässig und variierten stark. Häufig entsprachen die Änderungen, die die KI vorschlug, nicht unseren Vorgaben und führten oft dazu, dass die App abstürzte oder gar nicht startete, da viele Fehler gemacht wurden. Beispielsweise gaben wir häufig Hinweise, dass die Antworten ausführlich erklärt werden sollten. Dies wurde anfangs beachtet, aber nach einiger Zeit nicht mehr, sodass wir oft darauf hinweisen mussten. Häufig musste auch der Chat gewechselt werden, da die KI langsamer wurde und die Fehler zunahmen bzw. keine Änderungen im Code zu sehen waren, da der Chat bereits sehr lang war.

4.5 Effektivität von kleinen Anfragen

Forschungsfrage: Wie beeinflusst die Aufteilung einer komplexen Aufgabe in mehrere kleine Anfragen an ChatGPT die Effizienz und Qualität der erzielten Ergebnisse?

4.5.1 Präzision der Antworten

Die Bewertungsskala für die Präzision der Antworten lautet:

- **Sehr präzise:** Die kleinen Anfragen führen zu außerordentlich genauen und spezifischen Antworten, die die Aufgabe vollständig erfüllen.

- **Moderat präzise:** Die kleinen Anfragen führen zu weitgehend genauen Antworten, erfordern jedoch manchmal zusätzliche Klärungen oder Anpassungen.
- **Wenig präzise:** Die kleinen Anfragen führen oft zu ungenauen oder unvollständigen Antworten, die häufige Nachfragen erfordern.

Bewertung: 4.5.1 Präzision der Antworten

Moderat präzise: Die Aufgabe in mehrere kleine Anfragen zu unterteilen, ist oftmals besser, da ChatGPT oft Fehler macht, wenn er zu viele Dateien auf einmal erhält. Dann vergisst er ein bis zwei Dateien mit einzubeziehen. Es ist einfacher für ChatGPT, Dateien einzeln zu geben und nach Verbesserungen oder Hilfen zu fragen. Das Problem dabei ist, dass Änderungen in einer Datei zu Fehlern in einer anderen führen können, da viele Dateien in Zusammenhang stehen und deshalb zusammen abgeschickt werden müssen. Dieses Problem kann gelöst werden, indem zusätzliche Informationen mitgegeben werden.

4.5.2 Aufwand der Anfragen

Die Bewertungsskala für den Aufwand der Anfragen lautet:

- **Geringer Aufwand:** Die Aufteilung der Aufgabe in kleine Anfragen erfordert minimalen zusätzlichen Aufwand und Zeit.
- **Moderater Aufwand:** Die Aufteilung der Aufgabe in kleine Anfragen erfordert einen moderaten zusätzlichen Aufwand und Zeit.
- **Hoher Aufwand:** Die Aufteilung der Aufgabe in kleine Anfragen erfordert erheblichen zusätzlichen Aufwand und Zeit.

Bewertung: 4.5.2 Aufwand der Anfragen

Hoher Aufwand: Die Aufteilung der Aufgabe in kleine Anfragen erfordert einen Hohen zusätzlichen Aufwand und Zeit. Da die Aufgabe in kleinere Anfragen aufgeteilt wird, muss jeweils gewartet werden, bis eine Anfrage abgeschlossen ist, bevor die nächste gestellt werden kann. Außerdem muss man ChatGPT zusätzliche Informationen zu anderen Dateien geben, die in der einzelnen Anfrage nicht enthalten waren, um diese kleiner zu halten.

5 Diskussion

Alle gewonnenen Erkenntnisse in den vorherigen Kapiteln werden im Kontext der Forschungsfragen diskutiert. Dazu werden die Ergebnisse der Bewertungen ausgewertet und in einer übersichtlichen Tabelle zusammengefasst. Anschließend erfolgt die Beantwortung der einzelnen Forschungsfragen und die Interpretation der Ergebnisse, um herauszufinden, wie gut ChatGPT in der Softwareentwicklung ist.

5.1 Auswertung der Ergebnisse

Tabelle 1: Zusammenfassung der Bewertungen

Forschungsfrage	Kriterium	Bewertung	Begründung
1	Funktionale Vollständigkeit	Vollständig	Grundlegende Aufgabe erfüllt
1	Fehlerfreiheit des Programms	Moderater Fehler	Graphen überlappen gelegentlich
2	Klarheit des Programms	Moderat klar	Kommentare teilweise verständlich, Variablennamen klar
2	Verständlichkeit der ChatGPT-Antworten	Moderat verständlich	Anfangs detailliert, später klarer
3	Häufigkeiten der Anfragen	Viele Anfragen	Häufig Fehler, Programmabstürze oder keine Änderungen gemacht
3	Zeitaufwand für Korrektur und Validierung	Hoher Zeitaufwand	Viele Anfragen nötig, um Programm zum Laufen zu bringen
4	Präzision des Inputs	Hohe Präzision	Verständlichkeit von ChatGPT Antworten stark beeinträchtigt; ChatGPT macht mehr als gefordert
4	Konsistenz der Antworten	Inkonsistent	Nach einer Zeit wurden Hinweise vergessen; Oft keine Veränderungen im Code zu sehen gewesen
5	Präzision der Antworten	Moderat präzise	Kleine Anfragen besser, da ChatGPT bei vielen Dateien oft Fehler macht und einige vergisst. Änderungen in einer Datei können

			jedoch Fehler in anderen verursachen.
5	Aufwand der Anfragen	Hoher Aufwand	Hoher Aufwand und Zeit, da jede Anfrage einzeln gestellt und zusätzliche Informationen bereitgestellt werden müssen.

Quelle: Eigene Darstellung

5.2 Beantwortung der Forschungsfragen und Interpretation der Ergebnisse

Funktionale Vollständigkeit und Fehlerfreiheit

Laut der Hypothese enthält das von ChatGPT erstellte Programm Fehler, welche zu Funktionsstörungen und unvollständigen Programmen führen können. Das Ergebnis zeigte, dass ChatGPT die grundlegende Aufgabe bis auf einen kleinen Fehler erfolgreich erfüllt hat. Bei diesem kleinen Fehler handelt es sich um das Überlappen der Graphen während der Nutzung, welches nicht jedes Mal auftritt. Dieser Fehler weist keine erhebliche Beeinträchtigung der Gesamtfunktionalität auf. Aus diesem Grund wird die Hypothese widerlegt: ChatGPT konnte das Programm erstellen, das den spezifischen funktionalen Anforderungen entspricht, und der kleine Fehler ist nicht gravierend genug, um die Vollständigkeit zu beeinträchtigen.

Interpretation: Die Tatsache, dass die grundlegende Aufgabe von ChatGPT erfüllt wurde und nur einen kleinen Fehler aufwies, zeigt das Potenzial, welches KI zur Unterstützung der Softwareentwicklung bietet. Dabei ist wichtig zu beachten, dass dieser kleine Fehler die Gesamtfunktionalität des Programms nicht erheblich beeinträchtigt, was die Möglichkeit nochmals unterstreicht, dass es effektiv ist, KI-Tools in den Entwicklungsprozess zu integrieren. Es bleibt dennoch eine manuelle Überprüfung zur Behebung kleinerer Fehler erforderlich.

Verständlichkeit des Programms und der Antworten

Die Hypothese besagte, dass der mithilfe von ChatGPT generierte Programmcode und die dazugehörigen Antworten komplex strukturiert und schwer verständlich für den Anwender sind. Jedoch zeigt das Endresultat, dass ChatGPT anfangs keine Kommentare verwendet hat, diese aber bei präziseren Angaben hinzufügte. Viele dieser Kommentare neigten dazu, nicht klar genug zu sein, aber dies wurde nach gezielter Nachfrage behoben. Die Variablennamen waren von Anfang an sehr klar. Zu beachten ist jedoch, dass die Verständlichkeit des Codes durch bekanntes Vorwissen, vor allem in Java und anderen Programmiersprachen, erleichtert wurde. Dennoch benötigte man keine spezifischen Kenntnisse in der verwendeten Programmiersprache Kotlin. Daher wird die Hypothese widerlegt: Der Code war verständlicher als erwartet, insbesondere bei präzisen Eingaben und genauer Nachfrage.

Interpretation: Der generierte Code war zwar anfangs ohne Kommentare, doch bei präziseren Anfragen fügte ChatGPT Kommentare hinzu. Nicht alle Kommentare waren vollständig klar, aber dies wurde nach gezielter Nachfrage verständlicher. Die klaren Variablennamen trugen ebenfalls zur Verständlichkeit bei. Der Umstand, dass das Verständnis des Codes durch Erfahrung in verschiedenen anderen Programmiersprachen erleichtert wurde, zeigt, dass die KI verständlichen Code erzeugen kann, wenn die Eingaben präzise formuliert sind. Dies lässt sagen, dass ChatGPT zum jetzigen Stand nicht die Klarheit und Dokumentationsfähigkeit eines menschlichen Entwicklers erreicht.

Häufigkeit der Anfragen und Zeitaufwand für Korrekturen

Die Hypothese besagte, dass der Anwender häufig Anpassungen an seinen Anfragen vornehmen muss, um die gewünschten Ergebnisse von ChatGPT zu erhalten. Das Ergebnis zeigt, dass viele Anfragen notwendig waren, häufig mehr als fünf, um die gewünschten Resultate zu erzielen. Somit wird die Hypothese bestätigt: Viele Anfragen und Anpassungen sind erforderlich.

Interpretation: Die Notwendigkeit häufiger Anfragen und der hohe Zeitaufwand für Nachbesserungen zeigen, dass die Nutzung von ChatGPT in der Softwareentwicklung

derzeit noch nicht zu einer spürbaren Zeitersparnis führt. Der wiederholte Prozess der Anfragen und Korrekturen kann dazu führen, dass sich der Entwicklungsprozess verlängert, was Entwickler davon abhalten könnte, KI für komplexe Projekte zu verwenden. Diese Ergebnisse veranschaulichen, dass ChatGPT als unterstützendes Werkzeug nützlich sein kann, aber genaue menschliche Überwachung und Eingriffe erfordert.

Präzision des Inputs und Konsistenz der Antworten

Die Hypothese besagte, dass es entscheidend ist, die Benutzereingaben so präzise wie möglich zu formulieren, um die Qualität des von ChatGPT generierten Programms zu gewährleisten. Das Ergebnis zeigte, dass eine hohe Genauigkeit benötigt wurde, um brauchbare Ergebnisse zu erzielen. Die Hypothese wird daher bestätigt: Präzise Eingaben sind ausschlaggebend für die Qualität der Ergebnisse.

Interpretation: Die für die Eingaben erforderliche hohe Präzision und die Inkonsistenz der Antworten zeigen, dass die Interaktion mit der KI genaustens geplant und ausgeführt werden muss. Diese zusätzliche Herausforderung könnte den eigentlichen Vorteil der KI als einfaches und intuitives Werkzeug einschränken. Um brauchbare Ergebnisse zu erzielen, müssen Entwickler präzise und detaillierte Anfragen ausformulieren, was den Arbeitsaufwand erhöht. Außerdem macht die Inkonsistenz der Antworten die kontinuierliche Nutzung der KI deutlich schwieriger.

Effektivität von kleinen Anfragen

Die Hypothese besagte, dass Aufgaben, die als besonders umfangreich gelten, effizienter und qualitativ hochwertiger gelöst werden, indem sie in mehreren kleinen, genaueren Anfragen an ChatGPT gestellt werden, statt in einer einzigen großen Anfrage. Das Ergebnis hat gezeigt, dass es effektiver war, kleinere Anfragen zu stellen, jedoch erfordert dies auch einen höheren Aufwand. Die Hypothese wird somit bestätigt: Kleinere Anfragen führen zu besseren Ergebnissen, sind jedoch zeitintensiver.

Interpretation: Die Ergebnisse zeigen, dass kleinere Anfragen zwar einen höheren Aufwand erfordern, jedoch deutlich präzisere Antworten liefern. Dies zeigt, dass eine Balance

zwischen der Unterteilung der Anfragen und der Effizienz der Arbeitsprozesse gefunden werden muss. Entwickler müssen ihre Arbeitsweise so anpassen, dass sie die Vorteile der KI bestmöglich nutzen können. Das heißt, kleinere Anfragen sind effizienter in der Fehlervermeidung, aber zeitaufwendiger in der Durchführung.

5.3 Limitationen der Arbeit

Technische Limitationen: Die erzielten Ergebnisse sind abhängig von der spezifischen Version von ChatGPT und Android Studio und können die Reproduzierbarkeit beeinträchtigen. Unterschiedliche Versionen können zu variierenden Ergebnissen führen.

Generalisierbarkeit der Ergebnisse: Die erzielten Ergebnisse sind spezifisch für die Nutzung von ChatGPT 4o zur Programmierung einer App zur Datenerfassung und visuellen Darstellung. Möglicherweise sind sie nicht auf andere Programmierprojekte oder KI-Modelle übertragbar.

6 Fazit

6.1 Zusammenfassung der Ergebnisse

Diese Arbeit untersuchte die Leistungsfähigkeit von ChatGPT bei der Erstellung eines Programms zur Datenerfassung und -visualisierung. Die Ergebnisse zeigen auf, dass ChatGPT in der Lage ist, Programme zu erstellen, die den spezifischen funktionalen Anforderungen weitgehend entsprechen. Auch wenn das erstellte Programm einen kleinen Fehler aufwies, beeinträchtigte dieser nicht deutlich die Gesamtfunktionalität. Somit konnte ChatGPT die grundlegende Aufgabe erfolgreich erfüllen, was nochmals unterstreicht, wie viel Potenzial KI in der Unterstützung der Softwareentwicklung hat.

Der generierte Code und die dazugehörigen Antworten zeigten eine bessere Verständlichkeit als erwartet, vorwiegend bei präzisen Eingaben und präziser Nachfrage. Dennoch hat sich herausgestellt, dass Entwickler weiterhin eine entscheidende Rolle bei der manuellen Überwachung und Korrektur des von KI-Tools erstellten Codes spielen müssen. Die

Notwendigkeit mehrerer Anfragen und Korrekturen deutet darauf hin, dass ChatGPT in der Softwareentwicklung derzeit noch nicht imstande ist, eine signifikante Zeitersparnis zu leisten.

Als entscheidend für die Qualität von ChatGPT-generierten Programmen erwies sich vor allem die Präzision der Benutzereingaben. Umfangreiche Aufgaben, die in mehrere kleine, präzisere Anfragen aufgeteilt wurden, wurden effizienter und qualitativ hochwertiger gelöst.

6.2 Schlussfolgerung

Die Untersuchungen haben gezeigt, dass ChatGPT ein nützliches Werkzeug zur Unterstützung bei der Programmierung sein kann, insbesondere für Entwickler mit weniger Erfahrung. Das Modell ist in der Lage, Code zu generieren, der funktioniert, den Anforderungen größtenteils entspricht und eine potenzielle Steigerung der Effizienz im Entwicklungsprozess bietet. Dennoch ist eine menschliche Überwachung und Nacharbeitung unerlässlich, um einen qualitativ hochwertigen und voll funktionsfähigen Code sicherzustellen.

Es darf nicht vergessen werden, dass grundlegende Programmierkenntnisse wichtig sind, um diese Aufgabe erfolgreich bewältigen zu können. Zwar sind keine spezifischen Kenntnisse in Kotlin erforderlich, aber allgemeine Programmierkenntnisse reichen aus. Für Nutzer ohne jegliche Erfahrung in Programmiersprachen kann sich die Nutzung von ChatGPT zur Erfüllung komplexer Programmieraufgaben wesentlich schwieriger und komplexer darstellen. Ein gewisses Maß an Programmierverständnis ist erforderlich, um möglichst präzise Anfragen zu formulieren und den generierten Code zu verstehen, um eventuelle Fehler zu korrigieren.

Dadurch, dass viele Anpassungen und die Antworten von der Präzision abhängig sind, sollte ChatGPT derzeit eher als ein unterstützendes Werkzeug betrachtet werden als ein eigenständiges Entwicklungstool.

6.3 Ausblick

Für zukünftige Forschungen und Entwicklung besteht Bedarf an weiteren Untersuchungen zu Optimierung der Nutzung von KI-Tools in der Softwareentwicklung. Zukünftige Arbeiten könnten sich darauf konzentrieren, die Konsistenz und Fehlerfreiheit der von KI generierten Codes zu verbessern. Außerdem könnten verschiedene Strategien entwickelt werden, um die Präzision der Eingaben zu erhöhen und den Anfragestellungsprozess effizienter zu gestalten.

Durch die Integration von KI-Tools in den Softwareentwicklungsprozess bieten sich vielversprechende Möglichkeiten zur Effizienzsteigerung und darauffolgenden Kostensenkung. Ein zentraler Bestandteil ist weiterhin der menschliche Entwickler, insbesondere wenn es um die Überwachung, Korrektur und Optimierung der Ergebnisse geht, die von KI-Tools erstellt worden sind.

Literaturverzeichnis

Android Developer Bewegungssensoren (2024). Bewegungssensoren. Online verfügbar unter https://developer.android.com/develop/sensors-and-location/sensors/sensors_motion?hl=de (abgerufen am 17.07.2024).

Android Developer Build (2024). Build konfigurieren. Online verfügbar unter <https://developer.android.com/build?hl=de> (abgerufen am 17.07.2024).

Android Developer kennenlernen (2024). Android Studio kennenlernen. Online verfügbar unter <https://developer.android.com/studio/intro?hl=de> (abgerufen am 16.07.2024).

Android Developer Kotlin (2024). Kotlin-Übersicht. Online verfügbar unter <https://developer.android.com/kotlin/overview?hl=de> (abgerufen am 17.07.2024).

Android Developer Positionssensoren (2024). Positionssensoren. Online verfügbar unter https://developer.android.com/develop/sensors-and-location/sensors/sensors_position?hl=de (abgerufen am 24.07.2024).

Android Developer Sensoren (2024). Sensoren – Übersicht. Online verfügbar unter https://developer.android.com/develop/sensors-and-location/sensors/sensors_overview?hl=de (abgerufen am 17.07.2024).

- Android Developers SensorManager (2024). SensorManager Android Developers. Online verfügbar unter <https://developer.android.com/reference/android/hardware/SensorManager> (abgerufen am 24.07.2024).
- Chidera, Edeh Israel (2023). Kotlin VS Java – What's the Difference? freeCodeCamp.org vom 31.03.2023. Online verfügbar unter <https://www.freecodecamp.org/news/kotlin-vs-java-whats-the-difference/> (abgerufen am 16.07.2024).
- Garcia, Carlos (2023). Architectural Patterns: MVC, MVP, and MVVM Explained. AppMaster vom 28.08.2023. Online verfügbar unter <https://appmaster.io/blog/architectural-patterns-mvc-mvp-and-mvvm> (abgerufen am 18.07.2024).
- GeeksforGeeks (2017). MVC Design Pattern. GeeksforGeeks vom 18.08.2017. Online verfügbar unter <https://www.geeksforgeeks.org/mvc-design-pattern/> (abgerufen am 18.07.2024).
- Institute of Electrical and Electronics Engineers (Hg.) (2023). Human Activity Classification using G-XGB, 2023 International Conference on Data Science and Network Security (ICDSNS), Tiptur, India, 2023. IEEE.
- Javaid, Mohd/Haleem, Abid/Rab, Shanay/Pratap Singh, Ravi/Suman, Rajiv (2021). Sensors for daily life: A review. *Sensors International* 2, 100121. <https://doi.org/10.1016/j.sintl.2021.100121>.
- jjoe64 (2024). GitHub - jjoe64/GraphView: Android Graph Library for creating zoomable and scrollable line and bar graphs. Online verfügbar unter <https://github.com/jjoe64/GraphView> (abgerufen am 30.07.2024).
- Lawin, Dennis/Albrecht, Urs-Vito/Oftling, Zoe Sophie/Lawrenz, Thorsten/Stellbrink, Christoph/Kuhn, Sebastian (2022). Mobile Health zur Detektion von Vorhofflimmern – Status quo und Perspektiven. *Der Internist* 63 (3), 274–280. <https://doi.org/10.1007/s00108-022-01267-2>.
- Manivannan, Ajaykumar/Chin, Wei Chien Benny/Barrat, Alain/Bouffanais, Roland (2020). On the Challenges and Potential of Using Barometric Sensors to Track Human Activity. *Sensors* 20 (23), 6786. <https://doi.org/10.3390/s20236786>.
- Masoumian Hosseini, Mohsen/Masoumian Hosseini, Seyedeh Toktam/Qayumi, Karim/Hossein-zadeh, Shahriar/Sajadi Tabar, Seyedeh Saba (2023). Smartwatches in healthcare medicine: assistance and monitoring; a scoping review. *BMC Medical Informatics and Decision Making* 23 (1), 248. <https://doi.org/10.1186/s12911-023-02350-w>.

- Matej Andrejašić (März/2008). MEMS ACCELEROMETERS. Online verfügbar unter https://dl1wqtxts1xzle7.cloudfront.net/50775354/MEMS_accelerometers-koncna-libre.pdf?1481163559=&response-content-disposition=inline%3B+file-name%3DMEMS_accelerometers_koncna.pdf&Expires=1721808557&Signature=chnYOL-VVwcXXNTkd~wfkxK~zPNXo7eFj7625dhTJ~XG6DqDTcU11rx2lj5WkJrb-WaUx4SMpPZIbembhQslP9MXZ6gVsO12tPziZIKMpKMzE17m5b0PD3SHvIT-fKKzvr03xrpODrksq5XxB~lwJt0zEYzR2ydFfJvxNNkKHGxe6wEvasON-a8F5PGgxCc-0SfHhHX-lj6mLdZ4H5W55vq1zNLtTtTlzM1Nzei7TcIXs~Z0AHBZHI-Vvhcqilu6iDR6A4Q0Kh0IRDrE19wGPqvzmuza7a5MOW4Ymn5HLWeEIKZXKMqs~sUaiJr5JmeN393txJ9G3RA4nu1IVVktQ8B9xw__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA (abgerufen am 23.07.2024).
- Meyer, Jesse G./Urbanowicz, Ryan J./Martin, Patrick C. N./O'Connor, Karen/Li, Ruowang/Peng, Pei-Chen/Bright, Tiffani J./Tatonetti, Nicholas/Won, Kyoung Jae/Gonzalez-Hernandez, Graciela/Moore, Jason H. (2023). ChatGPT and large language models in academia: opportunities and challenges. *BioData Mining* 16 (1). <https://doi.org/10.1186/s13040-023-00339-9>.
- Muntenescu, Florina (2016). Android Architecture Patterns Part 1: Model-View-Controller. medium upday devs vom 01.11.2016. Online verfügbar unter <https://medium.com/upday-devs/android-architecture-patterns-part-1-model-view-controller-3baecef5f2b6> (abgerufen am 18.07.2024).
- PhilJay (2024). GitHub - PhilJay/MPAndroidChart: A powerful 🚀 Android chart view / graph view library, supporting line- bar- pie- radar- bubble- and candlestick charts as well as scaling, panning and animations. Online verfügbar unter <https://github.com/PhilJay/MPAndroidChart> (abgerufen am 30.07.2024).
- Samsung ch (2022). Vergleich: Galaxy Watch5 vs. Watch4 vs. Watch3 | Samsung Schweiz. Online verfügbar unter <https://www.samsung.com/ch/support/mobile-devices/vergleich-galaxy-watch5-watch4-watch3/> (abgerufen am 16.07.2024).
- Samsung Mobile Press (2022). Galaxy S22 Ultra. Online verfügbar unter <https://www.samsung-mobilepress.com/media-assets/galaxy-s22-ultra/?tab=specs> (abgerufen am 16.07.2024).
- Sarmadi, Hassan/Entezami, Alireza/Yuen, Ka-Veng/Behkamal, Bahareh (2023). Review on smartphone sensing technology for structural health monitoring. *Measurement* 223, 113716. <https://doi.org/10.1016/j.measurement.2023.113716>.
- Scholar GPT (2024). Informationen zu ScholarGPT. Online verfügbar unter <https://chat-gpt.com/g/g-kZ0eYXlJe-scholar-gpt> (abgerufen am 30.07.2024).

Shin, Grace/Jarrahi, Mohammad Hossein/Fei, Yu/Karami, Amir/Gafinowitz, Nicci/Byun, Ahjung/Lu, Xiaopeng (2019). Wearable activity trackers, accuracy, adoption, acceptance and health impact: A systematic literature review. *Journal of Biomedical Informatics* 93, 103153. <https://doi.org/10.1016/j.jbi.2019.103153>.

Anhang



Anhang Chats.zip

Ehrenwörtliche Erklärung

Hiermit versichere ich, dass die vorliegende Arbeit von mir selbstständig und ohne unerlaubte Hilfe angefertigt worden ist, insbesondere, dass ich alle Stellen, die wörtlich oder annähernd wörtlich aus Veröffentlichungen entnommen sind, durch Zitate als solche gekennzeichnet habe. Ich versichere auch, dass die von mir eingereichte schriftliche Version mit der digitalen Version übereinstimmt. Weiterhin erkläre ich, dass die Arbeit in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde/Prüfungsstelle vorgelegen hat. Ich erkläre mich damit nicht einverstanden, dass die Arbeit der Öffentlichkeit zugänglich gemacht wird. Ich erkläre mich damit einverstanden, dass die Digitalversion dieser Arbeit zwecks Plagiatsprüfung auf die Server externer Anbieter hochgeladen werden darf. Die Plagiatsprüfung stellt keine Zurverfügungstellung für die Öffentlichkeit dar.

(Ort, Datum)

90441 Nürnberg, 31.07.2024

(Eigenhändige Unterschrift)

Yasar E

Turan