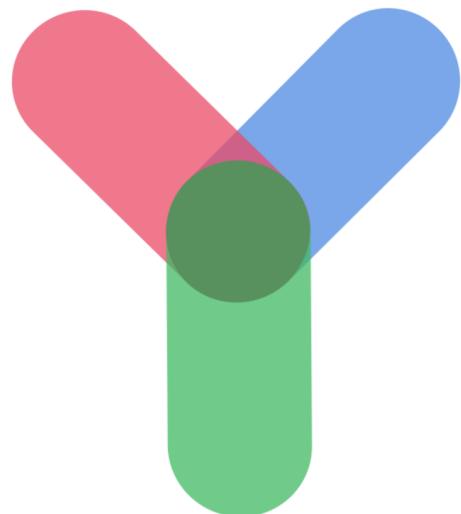


Plattform zur Analyse von Entwicklerzufriedenheit und Produktivität in agilen Teams

IP5 Projekt

Windisch, August 2025



Studenten: Xeno Isenegger, Gideon Monterosa
Fachbetreuer: Prof. Dr. Norbert Seyff, Dr. Nitish Patkar
Fachhochschule Nordwestschweiz, Hochschule für Informatik

Abstract

TODO: Abstract und Keywords schreiben

TODO: überprüfen ob sinnvolle namen und die richtigen titel angezeigt werden

Inhaltsverzeichnis

1 Einleitung	8
1.1 Motivation	8
1.1.1 Vision: Yappi als zentrale Plattform für Entwicklerzufriedenheit	9
1.2 Fragestellungen	9
1.3 Ausgangslage	10
1.4 Aktueller technischer Stand	11
1.5 Parallelle Entwicklung auf gemeinsamer Codebasis	12
1.6 Stakeholder	12
1.7 Projektmethodik	13
2 State of the Art	14
2.1 Definition von Entwicklerzufriedenheit	14
2.2 Stand der Forschung zur Entwicklerzufriedenheit	15
2.2.1 Entwicklerzufriedenheit in der Forschung	15
2.2.2 Entwicklerzufriedenheit in der Industrie	16
2.2.3 Mitarbeiterbindung und Entwicklerzufriedenheit	16
2.3 Gesundheitsdaten und Entwicklerzufriedenheit	17
2.4 Entwickler-Pain-Point: Meetings	18
2.5 Forschungslücken	19
2.6 Bestehende Lösungen und Wettbewerbsanalyse	19
2.7 Schlussfolgerungen für die Konzeptlösung	22
3 Konzeptlösung für die Erweiterungen an Yappi	23
3.1 Problemstellungen	24
3.1.1 Passive Erfassung der Entwicklerzufriedenheit	24
3.1.2 Einfluss einzelner Meetings	24
3.1.3 Schwierigkeiten beim Ableiten von Schlussfolgerungen	25
3.2 Idee	26

3.2.1	Automatisierte Aufforderung zur Erfassung von Zufriedenheitsdaten	26
3.2.2	Erfassung der Meetingqualität	27
3.2.3	Verknüpfung von Zufriedenheitswerten mit Kontext- und Gesundheitsdaten	27
3.2.4	Automatisierte Interpretation und Handlungsempfehlungen durch den Yappi Coach	28
3.3	Vision, Mission und Leitprinzipien	29
3.3.1	Vision: Yappi als zentrale Plattform für Entwicklerzufriedenheit . .	29
3.3.2	Mission: Nahtlose Erfassung, Analyse und Optimierung im Arbeitstag	30
3.3.3	Leitprinzipien: Werte und Ausrichtung der Plattform	30
3.4	Value Proposition	30
3.4.1	Mehrwert für Einzelpersonen – Persönliche Optimierung der Arbeitszufriedenheit	30
3.4.2	Mehrwert für Teams – Gemeinsame Verbesserung von Zusammenarbeit und Prozessen	31
3.5	Produktziele	31
3.5.1	Z-1: Automatisierte Aufforderung zur Zufriedenheitserfassung . . .	32
3.5.2	Z-2: Erfassung der Meetingqualität	32
3.5.3	Z-3: Verknüpfung mit Kontext- und Gesundheitsdaten	33
3.5.4	Z-4: Automatisierte Interpretation und Handlungsempfehlungen (Yappi Coach)	33
3.6	Konzeptevaluation	34
4	Implementierung	36
4.1	Yappi als Integrationsplattform	36
4.1.1	Kommunikationsmechanismus: API-Keys	37
4.1.2	Systemarchitektur der Integrationsplattform	37
4.1.3	Zugriffskontrolle über API-Keys	38
4.1.4	Sicherheit	40
4.1.5	Ablauf der API-Key Erstellung und -Verwendung	41
4.1.6	Benutzeroberfläche zur API-Key-Verwaltung	42
4.2	IntelliJ IDEA Companion	43
4.2.1	Architektur und Integration	43
4.2.2	Speicherung der Einstellungen	44
4.2.3	Post-Commit Dialog	45
4.2.4	Daten verarbeiten und Persistieren	46
4.3	Calendar Companion	47

4.3.1	Yappi Chrome Extension	48
4.3.2	Backend-Integration	52
4.3.3	Datenbankerweiterung	53
4.3.4	Kommunikationsbroker	54
4.3.5	Dynamische Umfragen	56
4.4	Health Companion	60
4.4.1	Quellen und Schnittstellen für Gesundheitsdaten	60
4.4.2	Daten aus HealthKit auslesen	61
4.4.3	Daten an Yappi übertragen	62
4.4.4	Benutzeroberfläche der iOS App	63
4.5	Deployment und CI/CD	65
4.5.1	Build- und Deployment-Pipeline	65
4.5.2	Serverinfrastruktur	67
4.5.3	NGINX-Proxykonfiguration	68
4.5.4	Docker-Compose-Struktur	68
4.5.5	MQTT-Broker-Konfiguration	68
4.5.6	Zusammenfassung	68
5	Evaluation	69
5.1	Usertesting am Hackathon	69
5.2	Evaluation durch Simulation	70
5.2.1	Testszenarien	70
5.3	Beantwortung der Fragestellung	73
6	Diskussion	75
6.1	Reflexion	75
Anhang		79
A	Verlinkungen	79
B	Mockup	79
C	Anhang	81

Abbildungsverzeichnis

1.1	Systemarchitektur von Yappi vor dem Projekt	11
3.1	Durchschnittsbewertung der Konzeptfunktionen auf einer Likert-Skala.	35
4.1	Systemarchitektur der Yappi-Integrationsplattform	38
4.2	Klassendiagramm für die Implementierung der API-Keys	39
4.3	Ablauf der API-Key Erstellung und -Verwendung in Yappi	41
4.4	Benutzeroberfläche für die Verwaltung von API-Keys im Frontend	42
4.5	Architektur des IntelliJ IDEA Companion	43
4.6	Eingabe des API-Keys in den IntelliJ Companion-Einstellungen	44
4.7	Post-Commit Dialog des IntelliJ Companions	45
4.8	Systemübersicht zur Kalenderintegration in Yappi	48
4.9	Systembenachrichtigung der Chrome Erweiterung in macOS	50
4.10	Benutzeroberfläche der Chrome Erweiterung	51
4.11	Ablauf nachdem ein Kalender-Event abgeschlossen wurde.	51
4.12	Feedbackformular zu Kalenderevent	58
4.13	Benutzeroberfläche der iOS Health Companion App	64
4.14	Deployment flow vom Backend	66
4.15	Deployment flow vom Frontend	66
4.16	Systemübersicht der Linuxumgebung	67

Tabellenverzeichnis

3.1	Problemstellung nach RUP: Passive erfassung der Entwicklerzufriedenheit	24
3.2	Problemstellung nach RUP: Einfluss auf die Zufriedenheit einzelner Meetings wird nicht erfasst	25
3.3	Problemstellung nach RUP: Schwierigkeiten beim ableiten von Schlussfolge- rungen aus Zufriedenheitsdaten	25
4.1	Schema der Tabelle <code>api_key</code>	40
4.2	Schema der Tabelle <code>commit_context</code>	47
4.3	Schema der Tabelle <code>calendar_events</code>	54
4.4	Schema der Tabelle <code>questionblocks</code>	56
4.5	Schema der Tabelle <code>health_metric</code>	63

Kapitel 1

Einleitung

1.1 Motivation

TODO: spellcheck

TODO: gendern überprüfen

TODO: unscharfe bilder fixen

Die Zufriedenheit von Entwicklerinnen und Entwicklern wird, wenn überhaupt, meist nur anhand der Menge ihrer geleisteten Arbeit gemessen. Dabei entstehen zwangsläufig Defizite, und ein halbjährliches Mitarbeitergespräch erweist sich oft als wenig wirksame Massnahme zur Problemlösung.

Dieses Projekt baut auf einer bestehenden Arbeit auf, in der eine Plattform zur Erfassung der Entwicklerzufriedenheit entwickelt wurde. Die Webapplikation Yappi ermöglicht es Entwicklerinnen und Entwicklern, ihre Zufriedenheit mit ihrer Arbeit und ihrer aktuellen Situation fortlaufend zu bewerten. Yappi erfasst emotionale Faktoren wie Happiness sowie weitere Zufriedenheitsindikatoren. Zusätzlich können spezifische Aufgaben und Arbeitstypen individuell bewertet werden. Die erhobenen Daten werden anonym auf Teamebene analysiert, um ein fundiertes Verständnis für die Stimmung innerhalb der Teams zu gewinnen.

Entwicklerinnen und Entwickler haben die Möglichkeit, ihre Zufriedenheit für verschiedene Teams zu erfassen, wodurch gezielte Analysen ermöglicht werden. Unternehmen erhalten dadurch wertvolle Einblicke, um das Arbeitsumfeld gezielt zu verbessern.

Dieses Projekt baut auf Yappi auf und zielt darauf ab, die Erfassung der Zufriedenheit weiter zu optimieren. Es wird untersucht, wie die Daten noch präziser erfasst und aus-

gewertet werden können, um langfristige Verbesserungen zu unterstützen. Diese Arbeit dient als Grundlage für ein weiterführendes Forschungsprojekt, das sich vertieft mit der Entwicklerzufriedenheit auseinandersetzt und zusätzliche Erkenntnisse gewinnen soll.

1.1.1 Vision: Yappi als zentrale Plattform für Entwicklerzufriedenheit

Aus dieser Zielsetzung leitet sich eine klare Vision für die Weiterentwicklung von Yappi ab:

Yappi soll sich zu einer zentralen, intelligenten Plattform entwickeln, die es Entwicklerinnen, Entwicklern und Teams ermöglicht, die Arbeitszufriedenheit kontinuierlich, präzise und im Kontext zu erfassen, zu verstehen und gezielt zu verbessern. Ziel ist ein Arbeitsumfeld, in dem Zufriedenheit als gleichwertiger Faktor neben Produktivität und Codequalität betrachtet wird und in dem datenbasierte Erkenntnisse zu einer nachhaltig gesunden und motivierenden Arbeitskultur beitragen.

1.2 Fragestellungen

Aufbauend auf der in der Motivation beschriebenen Vision sowie dem dargestellten Projektrahmen wurden drei zentrale Forschungsfragen formuliert. Sie bilden den inhaltlichen Leitfaden dieser Arbeit und definieren die Aspekte, die im Projekt vertieft untersucht werden.

- A. Durch welche Technologien und Schnittstellen kann Yappi erweitert werden, um ein reibungsloses und einfaches Erfassen von Zufriedenheitsdaten zu ermöglichen?
 - a. Entwicklung von Entwickler-Tool-Plugins, die nahtlos in bestehende Arbeitsumgebungen integriert werden können, um die Nutzung von Yappi angenehmer und effizienter zu gestalten. Diese Plugins sollen Entwicklern ermöglichen, direkt in ihrer bevorzugten Umgebung Feedback zu erfassen, ohne den Arbeitsfluss zu unterbrechen. Integration von Yappi in verschiedene Plattformen und Tools wie Webbrowser, IntelliJ, Microsoft Teams und Outlook.
- B. Wie können Gesundheitsdaten in die Auswertung der Entwicklerzufriedenheit einfließen?
 - a. Direkte Anbindung der Gesundheitsdaten-API, um relevante Gesundheitsmetriken wie Herzfrequenz, Schlafqualität oder Stresslevel automatisch in die Analyse der Entwicklerzufriedenheit zu integrieren. Dies ermöglicht eine genauere Einschätzung des Wohlbefindens und potenzieller Belastungsfaktoren.

- C. Wie kann Yappi Teams und Entwickler dabei unterstützen, aus den erfassten Zufriedenheitsdaten Handlungsempfehlungen abzuleiten, um die Zufriedenheit und Produktivität von Entwicklern zu erhöhen?
 - a. Entwicklung eines Yappi Coach, der anhand einer detaillierten Analyse der erfassten Daten gezielte Tipps zur Verbesserung der Arbeitsweise gibt. Beispielsweise könnte der Coach darauf hinweisen, dass Meetings nicht länger als 1,5 Stunden dauern sollten, da längere Sitzungen die Zufriedenheit und Konzentration der Entwickler negativ beeinflussen können.
 - b. Integration von KI-gestützten Diensten, die auf Basis der gesammelten Gesundheitsdaten sowie Zufriedenheits- und Produktivitätsmetriken individuelle Massnahmen vorschlagen. Diese KI-gestützten Empfehlungen können Teams dabei helfen, gezielt Optimierungen vorzunehmen, um die Arbeitsbedingungen und die Effizienz der Entwickler nachhaltig zu verbessern.

Die Forschungsfragen A und B werden im Rahmen dieser Arbeit praktisch umgesetzt. Die Forschungsfrage C wird aufgrund ihres Umfangs und der notwendigen Vorarbeiten in diesem Projekt konzeptionell ausgearbeitet und dient als Grundlage für eine mögliche Umsetzung in einem Folgeprojekt.

1.3 Ausgangslage

Yappi ist eine Webplattform zur Erfassung der Entwicklerzufriedenheit und produktionsnaher Kennzahlen. Ziel ist eine regelmässige, datenbasierte Diskussion im Team und ein besseres Verständnis wiederkehrender Muster der Teamstimmung. Die Vorgängerarbeit definiert dafür eine klare Produktvision und liefert ein erstes Minimum Viable Product (MVP) als Grundlage. Die Plattform fokussiert sich auf Selbst-Reporting zur Auswertung der Entwicklerzufriedenheit. Der Ansatz adressiert typische Schmerzpunkte agiler Teams und stützt sich auf Interviews, Literatur und abgeleitete Produktziele. Yappi erlaubt die Erfassung von Happiness-Daten, deren Auswertung im Dashboard und Vergleiche innerhalb eines Teams. Die Lösung ist so ausgelegt, dass in Retrospektiven datenbasierte Gespräche geführt werden können.

Yappi ist als Open-Source-Projekt veröffentlicht. Die Repositories für Backend, Frontend und Infrastruktur sind getrennt organisiert. Eine lauffähige Instanz steht zu Beginn dieses Projektes nicht zur Verfügung. Die Zielgruppe ist klar beschrieben. Angesprochen sind agile Software-Teams, Scrum Master und Product Owner unterschiedlicher Unternehmensgrößen. Die Vorgängerarbeit hat die methodische Basis gelegt. Sie umfasst Literaturrecherche, Interviews, abgeleitete Problemfelder, User Flows, Produktziele und die Validierung der

Konzeptlösung. Das vorliegende Projekt baut auf dieser Struktur auf und fokussiert nun auf Integration, Erweiterbarkeit und vertiefte Analytik.

1.4 Aktueller technischer Stand

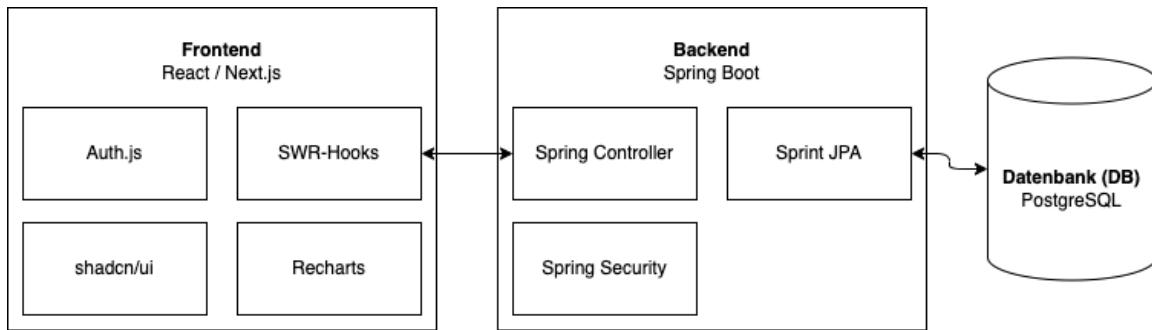


Abbildung 1.1: Systemarchitektur von Yappi vor dem Projekt

Abbildung 1.1 zeigt die Systemarchitektur von Yappi in ihrem ursprünglichen Zustand vor Beginn dieses Projekts. Die Plattform bestand aus einer klar getrennten Frontend- und Backend-Architektur, die über dokumentierte REST-Schnittstellen kommunizieren. Alle Hauptkomponenten (Webanwendung, API, Datenbank und unterstützende Services) sind funktional vorhanden, jedoch in Teilen nur als Minimal Viable Product (MVP) umgesetzt. Dieses Architektur-Setup bildet die Ausgangsbasis für die im weiteren Verlauf geplanten Erweiterungen.

Backend Das Backend ist mit Spring Boot umgesetzt. Als Datenbank ist PostgreSQL im Einsatz. Die Kommunikation zum Frontend erfolgt über REST-Schnittstellen, welche nach dem OpenAPI standard dokumentiert sind. Die Kommunikation zur Datenbank erfolgt über Spring JPA.

Frontend Das Frontend basiert auf React und Next.js mit TypeScript. Für Aufrufe werden SWR-Hooks verwendet, dabei handelt es sich um eine React library um Daten zu laden. Komponenten werden mit der Library shadcn/ui aufgebaut. Die Visualisierungen entstehen mit Recharts. Um die Codequalität sicherzustellen wird ESLint zur statischen Code analyse und Prettier zur automatischen Formatierung nach vordefinierten Guidelines verwendet.

Authentisierung und Autorisierung Die Anmeldung erfolgt über OAuth 2.0. Dabei werden Anbieter wie Google oder GitHub über Auth.js unterstützt. Passwörter werden nicht gespeichert. Die Anfragen ans Backend werden mit JSON Web Tokens (JWT) abgesichert. Die Sicherheitsmechanismen im Backend sind mit Spring Security implementiert.

Datenmodell Das relationale Schema umfasst unter anderem Benutzer, Teams, Sprints sowie Umfragetypen für Happiness, Emotionen und Work-Kinds.

Deployment Die Komponenten sind grösstenteils containerisiert mit Docker. Eine funktionierende CI/CD-Pipeline ist aktuell nicht vorhanden. Bestehende Images sind im GitHub Container Registry abgelegt. Für den Betrieb wird eine Linux-VM wie z.B. auf der Switch-Engine-Infrastruktur benötigt.

1.5 Parallele Entwicklung auf gemeinsamer Codebasis

Während der Projektlaufzeit wurde die bestehende Codebasis nicht ausschliesslich durch das im Rahmen dieser Arbeit verantwortliche Projektteam weiterentwickelt. Parallel dazu fanden Arbeiten eines weiteren Projekts auf derselben Codebasis statt.

Um Konsistenz, Qualität und Stabilität der Anwendung zu gewährleisten, übernahm das Projektteam die Verantwortung für die zentrale Verwaltung des Codes.

Alle externen Änderungen wurden vor der Integration einem Code Review unterzogen. Dabei prüften wir die Anpassungen insbesondere auf funktionale Korrektheit, technische Kompatibilität und grundlegende Einhaltung der bestehenden Architekturprinzipien. Erst nach erfolgreicher Durchsicht und Freigabe wurden die Änderungen in den main Branch übernommen.

Dieses Vorgehen stellte sicher, dass parallele Entwicklungsaktivitäten koordiniert abliefen und die gemeinsame Codebasis in einem stabilen, funktionsfähigen Zustand blieb.

1.6 Stakeholder

Das Projekt umfasst die folgenden Stakeholder:

Primäre Nutzergruppen Softwareentwickelnde, Scrum Master und Product Owner in agilen Teams.

Organisationen Unternehmen nutzen Yappi, um Kultur und Zusammenarbeit datenbasiert zu verbessern.

Akademische Stakeholder Betreuende Dozierende und das Institut an der FHNW welche das Projekt begleiten.

Betrieb und Entwicklung Projektteam dieses Projekts und anderer Projekte die auf Yappi aufbauen.

1.7 Projektmethodik

Dieses Kapitel beschreibt das methodische Vorgehen bei der Entwicklung und Umsetzung der im Rahmen dieses Projekts geplanten Erweiterungen an Yappi. Es erläutert, wie aus den Forschungsfragen und identifizierten Problemstellungen konkrete Ziele und Lösungsansätze abgeleitet wurden, welche Werkzeuge und Methoden zur Informationsgewinnung eingesetzt wurden und wie der Entwicklungsprozess strukturiert war.

1. Recherche- und Analysephase:

- **Google Scholar:** Recherche wissenschaftlicher Publikationen.
- **Google Search:** Analyse bestehender Marktangebote und Wettbewerber, um Funktionalitäten, Stärken und Schwächen zu identifizieren.
- **AI-gestützte Tools (z. B. ChatGPT):** Nutzung zur schnellen Explorati-
on von Ideen und Einholung komprimierter Literaturübersichten. Ergebnisse
wurden immer manuell überprüft und mit Primärquellen validiert. Zusätzlich
wurden AI-Tools verwendet um Texte hinsichtlich Sprache und Rechtschreibung
zu überprüfen.

2. **Zielentwicklung:** Ableitung der Produktziele (Kapitel 3.5) aus den Forschungsfragen (Kapitel 1.2), Problemstellungen (Kapitel 3.1) und den Schlussfolgerungen aus der Literaturrecherche und Marktanalyse (Kapitel 2.7).

3. **Architekturdokumentation:** Beschreibung der technischen Architektur nach dem arc42-Template.

4. **Agiles Projektmanagement:** Organisation der Arbeitsschritte mittels Kanban-Board, regelmässige Abstimmungen zur Fortschrittskontrolle und Prioritätsanpas-
sung.

5. **Mockups:** Erstellung Mockups (Figma) zur Visualisierung geplanter Benutzerober-
flächen, insbesondere für Companion-Apps und Meetingbewertungsdialoge.

Kapitel 2

State of the Art

2.1 Definition von Entwicklerzufriedenheit

Entwicklerzufriedenheit wird in der Literatur als Balance zwischen positiven und negativen Erlebnissen bei der Arbeit definiert. Darunter versteht man eine Sequenz von Erfahrungen, bei der häufige positive Emotionen ein hohes Glücksgefühl erzeugen und häufige negative Erfahrungen das Gegenteil bewirken [1]. Auch Industriequellen fassen Entwicklerzufriedenheit als subjektives Wohlbefinden in Bezug auf Arbeitsinhalte und -umfeld auf, d.h. als Mass für Zufriedenheit, Freude oder innere Zufriedenheit bei der Arbeit [2]. Zufriedene Entwickler empfinden demnach mehr Arbeitsfreude und Inhaltlichkeit in ihrer Rolle, was eng mit der Arbeitsmotivation und dem Engagement bei der Arbeit verknüpft ist [3].

Eng verwendet mit der Zufriedenheit ist der Begriff **Flow**. In Anlehnung an Csikszentmihalyis Konzept beschreibt Flow einen Zustand von völliger Vertiefung und hohen Fokus beim Programmieren. Flow tritt dann auf, wenn die Anforderungen einer Aufgabe im Gleichgewicht mit den Fähigkeiten des Entwicklers stehen, wodurch man in einen Zustand von intensiver Konzentration gelangt. Zufriedene Entwickler gelangen einfacher in einen anhaltenden Flow-Zustand. Unzufriedenheit hingegen unterbricht diesen Flow, was zu Frustration führt und Schwierigkeiten führt, nach Unterbrechungen wieder in eine Aufgabe zurückzufinden. Teilnehmer einer Untersuchung berichten, negative Erlebnisse reissen einen aus dem Flow Zustand und machen es schwer, die Arbeit wieder aufzunehmen [1].

Motivation und Zufriedenheit hängen eng zusammen, sind aber konzeptionell unterscheidbar. Motivierte Entwickler sind zeigen hohes Engagement und Fokus auf ihre Aufgaben, während Zufriedenheiter durch allgemeines Wohlbefinden und gute Laune charakterisiert ist. Faktoren wie Autonomie, Kompetenzerleben und Zugehörigkeitsgefühl steigern die intrinsische Motivation von Entwicklern, was sich positiv auf ihre Zufriedenheit auswirkt.

Zufriedenheit ist zugleich das Ergebnis und die Voraussetzung von Motivation, zufriedenere Entwickler weisen in der Regel eine höhere Antriebskraft auf, was wiederum ihre Arbeitszufriedenheit weiter stärkt [3].

Schliesslich spielt auch das Team- und Organisationsklima eine fundamentale Rolle. Eine offene, unterstützende Kultur steigert nachweislich die Zufriedenheit von Entwicklern. Der DORA Report misst die Leistungsfähigkeit von Softwareentwicklungsteams anhand von vier Schlüsselkennzahlen: Deployment Frequency, Lead Time for Changes, Change Failure Rate und Time to Resore Service. Der DORA Report von Google basiert auf umfangreichen Wissenschaftlichen Studien und gilt als Branchenstandart. Der report von 2024 betont, dass Teams mit stabilem, ermutigendem Umfeld bessere Ergebnisse erzielen [4]. Positive Emotionen und ein Zugehörigkeitsgefühl im Team fördern den Gruppenzusammenhalt, was wiederum die Teamleistung von Teammitgliedern verbessert. Umgekehrt können Umgekehrt können toxische Kulturen oder ständig wechselnde Prioritäten die Zufriedenheit und Motivation untergraben, was sich negativ auf die Leistung auswirkt [1].

Somit unterstreichen sowohl akademische als auch industrielle Befunde: Entwicklerzufriedenheit entsteht in einem komplexen Zusammenspiel aus individuellen Faktoren (Flow, Motivation, ...) und Umfeldfaktoren (Team- und Organisationsklima, Arbeitskultur, ...).

2.2 Stand der Forschung zur Entwicklerzufriedenheit

2.2.1 Entwicklerzufriedenheit in der Forschung

In den letzten Jahren haben zahlreiche Studien den Zusammenhang zwischen der Entwicklerzufriedenheit und Erfogsparametern wie Produktivität, Codequalität und Mitarbeiterbindung untersucht. Produktivität beschreibt im Softwarekontext die Effizienz, mit der Entwickler innerhalb einer bestimmten Zeit funktionierende und qualitativ hochwertige Softwareartefakte erstellen. Codequalität bezeichnet das Mass, in dem Quellcode korrekt, wartbar, verständlich und effizient ist, und umfasst Aspekte wie Lesbarkeit, Modularität, Fehlertoleranz und Testbarkeit. Mitarbeiterbindung wiederum beschreibt das Ausmass, in dem Mitarbeitende langfristig und freiwillig im Unternehmen bleiben.

Bei einer grossangelegten Studie wurden 317 Softwareentwickler befragt und dabei 42 Konsequenzen von Unzufriedenheit sowie 32 Konsequenzen von Zufriedenheit beim Programmieren identifiziert. Die Ergebnisse Zeigen, dass Entwicklerzufriedenheit messbare Auswirkungen auf den Entwicklungsprozess, die erzeugten Software-Artefakte und das Wohlbefinden der Person hat. So führt Unzufriedenheit zu einer Reihe negativer Effekte: verzögerte Prozessabläufe, nachlässige Arbeitsweise und häufige unterbrechungen des Flows

wurden als typische Folgen von negativer Stimmung genannt. Unzufriedene Entwickler berichten von langen Verzögerungen oder Qualitätsproblemen, weil Frustration sie aus dem Konzept brachte. Zufriedenheit hingegen wirkt sich positiv aus: Zufriedene Entwickler zeigen bessere Problemlösungsfähigkeiten, höhere Konzentration, berichten von einem anhaltenden Flow Zustand und lernen schneller. Die Ergebnisse legen nahe, dass Zufriedenheit die Codequalität begünstigt. Zufriedene Entwickler treffen sorgfältigere langfristige Entscheidungen. Ein Teilnehmer beschrieb, er dokumentiert seinen Code gründlicher und achten stärker auf Wartbarkeit, wenn er Zufrieden ist [5].

2.2.2 Entwicklerzufriedenheit in der Industrie

Neben akademischen Studien liefern auch Industrienumfragen und Community-Studien wertvolle Einblicke. Der jährliche Stack Overflow Developer Survey etwa spiegelt wieder, dass weiterhin Verbesserungsbedarf besteht. laut der Umfrage 2024 bezeichnen sich nur rund 20% der Entwickler als wirklich zufrieden in ihrem Job, während etwa 80% unzufrieden oder "gelassen"(complacent) sind. Diese Zahl unterstreicht, dass ein grosser Teil der Entwicklergemeinschaft nicht glücklich im Arbeitsumfeld ist. Als Hauptgründe werden oft Faktoren wie schlechte Work-Life-Balance, zu viele Meetings oder fehlende Anerkennung genannt [6].

Eine Untersuchung von Zenhub kam zu ähnlichen Ergebnissen: Zwar gaben die meisten befragten Entwickler an, überwiegend zufrieden zu sein, doch lediglich 31% fühlten sich äusserst zufrieden in ihrer aktuellen Arbeitssituation [2].

2.2.3 Mitarbeiterbindung und Entwicklerzufriedenheit

Darüber hinaus tragen Anerkennung und Sinnhaftigkeit der Arbeit entscheidend zur Bindung bei. Wenn Entwicklerinnen und Entwickler stolz auf die Qualität ihrer Projekte sind und regelmässig Wertschätzung für ihre Arbeit erhalten, steigt sowohl ihre Zufriedenheit als auch ihre Loyalität gegenüber dem Unternehmen [1], [5]. Die empirischen Befunde deuten somit klar darauf hin, dass Entwicklerzufriedenheit kein rein „weiches“ Thema ist, sondern messbare Auswirkungen auf Produktivität, Codequalität und Personalbindung hat. Zufriedene Entwickler arbeiten effizienter, treffen sorgfältigere Entscheidungen und bleiben ihrem Team länger erhalten, während Unzufriedenheit mit erhöhten Kosten für Projekte und Rekrutierung verbunden ist.

2.3 Gesundheitsdaten und Entwicklerzufriedenheit

Neben rein arbeitsbezogenen Faktoren wie Arbeitsorganisation, technischer Ausstattung oder Teamdynamik rücken zunehmend auch physiologische und gesundheitsbezogene Parameter in den Fokus der Forschung zur Entwicklerzufriedenheit. Verschiedene Studien aus der Arbeitspsychologie und der Human Factors-Forschung belegen, dass körperliche und mentale Gesundheit einen messbaren Einfluss auf kognitive Leistungsfähigkeit, emotionale Stabilität und Motivation haben. Besonders in wissensintensiven und hochkonzentrativen Berufen wie der Softwareentwicklung können selbst moderate Abweichungen vom optimalen Gesundheitszustand die Arbeitsleistung und das subjektive Wohlbefinden deutlich beeinflussen.

1. Schlafrdauer

Schlafrdauer ist ein zentraler Prädiktor für kognitive Leistungsfähigkeit, emotionale Regulation und Motivation. Chronisch verkürzter Schlaf führt zu verminderter Aufmerksamkeitsspanne, reduzierter Problemlösefähigkeit und einer erhöhten Anfälligkeit für negative Stimmungslagen. Personen mit ausreichendem, qualitativ hochwertigem Schlaf weisen höhere Arbeitszufriedenheit, geringere Reizbarkeit und bessere soziale Interaktionen am Arbeitsplatz auf. Besonders in wissensintensiven Tätigkeiten wie der Softwareentwicklung, bei denen hohe Konzentration und Kreativität erforderlich sind, kann Schlaftypus die Produktivität stark mindern [7].

2. Ruheherzfrequenz (RHR)

Die Ruheherzfrequenz reflektiert den Grundzustand des Herz-Kreislauf-Systems und reagiert empfindlich auf chronische Belastungen wie Stress oder Überarbeitung. Eine dauerhaft erhöhte RHR ist mit reduzierten Erholungsphasen assoziiert. In arbeitspsychologischen Studien wurde ein signifikanter Zusammenhang zwischen hohem Job-Strain, erhöhter RHR und niedrigerer Arbeitszufriedenheit festgestellt. Für Entwicklerinnen und Entwickler kann eine kontinuierliche Überwachung der RHR helfen, Phasen erhöhter Belastung zu erkennen, noch bevor subjektive Erschöpfung spürbar wird [8].

3. Stress (Herzratenviariabilität, HRV)

Die Herzratenviariabilität beschreibt die zeitliche Variation zwischen aufeinanderfolgenden Herzschlägen und gilt als sensibler Indikator für die Funktionsbalance des autonomen Nervensystems. Eine hohe HRV weist auf ein flexibles, gut reguliertes System hin, das Belastungen effizient kompensieren kann. Eine niedrige HRV hingegen signalisiert anhaltenden Stress oder unzureichende Erholung. In Kombination mit subjektiven Zufriedenheitsdaten kann die HRV helfen, versteckte Stressmuster

zu identifizieren, die sonst unentdeckt bleiben würden [9].

4. Aktivitätsminuten und Schritte

Körperliche Aktivität wirkt sich positiv auf mentale Gesundheit, Energilevel und Stimmung aus. Schon moderate Bewegung reduziert Stress, verbessert die Schlafqualität und steigert die Arbeitszufriedenheit. Diese lässt sich durch die täglichen Schritte oder die aktiven Minuten messen. Für sitzende Berufe wie die Softwareentwicklung kann regelmässige Bewegung das Risiko von Ermüdung und Motivationsverlust verringern. Mitarbeitende mit höherem Aktivitätsniveau klagen seltener über emotionale Erschöpfung und haben eine insgesamt positivere Einstellung zur Arbeit [10].

Diese Metriken ermöglichen eine gezielte Verknüpfung zwischen erholungs- und gesundheitsbezogenen Faktoren sowie den erhobenen Zufriedenheits- und Prozessdaten. Durch die Analyse solcher Daten lassen sich Muster identifizieren, die potenzielle Belastungssituationen aufzeigen, bevor sie sich in Produktivitätsverlust oder Qualitätsproblemen niederschlagen.

2.4 Entwickler-Pain-Point: Meetings

Meetings sind geplante, zeitlich begrenzte Zusammenkünfte mehrerer Personen mit einem festgelegten Zweck. Sie finden in der Softwareentwicklung vor Ort oder virtuell statt und umfassen formelle Formate wie Sprint-Planungen, Retrospektiven oder Daily Stand-ups sowie informelle Abstimmungen, Workshops oder Ad-hoc-Besprechungen. Ziel ist in der Regel die Koordination, Entscheidungsfindung, Wissensweitergabe oder Problemlösung.

In der Praxis wirken Meetings jedoch häufig als Störfaktor für die Produktivität und Zufriedenheit von Entwicklerinnen und Entwicklern. Empirische Untersuchungen zeigen, dass Softwareentwickelnde durchschnittlich zwischen 10,9 und 16,5 Stunden pro Woche in Besprechungen verbringen. Ein erheblicher Teil dieser Zeit wird als wenig produktiv eingeschätzt. Besonders problematisch sind lange oder ungünstig terminierte Meetings, die den Arbeitstag fragmentieren. Dies unterbricht oft den Flow-Zustand, also eine Phase tiefer Konzentration und vollständiger Aufgabenfokussierung. Die Wiederaufnahme komplexer Programmieraufgaben nach einer Unterbrechung erfordert zusätzlichen kognitiven Aufwand und führt zu Zeitverlust. [11], [12]

Neben der Häufigkeit und Dauer beeinflussen auch qualitative Faktoren die Wahrnehmung von Meetings. Dazu gehören klare Zieldefinitionen, eine strukturierte Agenda, effektives Zeitmanagement und die aktive Einbindung der Teilnehmenden. Fehlende Moderation, unklare Ergebnisse oder das Ausufern in Nebenthemen werden von Entwicklerinnen und Entwicklern als belastend wahrgenommen.

Aus Sicht der Entwicklerzufriedenheit ist daher sowohl die quantitative Belastung durch Meetings als auch deren inhaltliche Qualität entscheidend. Die systematische Erfassung der Meetingwahrnehmung, kann helfen, problematische Muster zu identifizieren. Auf dieser Basis lassen sich gezielte Massnahmen zur Optimierung der Meetingkultur ableiten, beispielsweise eine Reduktion der Dauer, die Verbesserung der Agendaqualität oder die Anpassung der Terminplanung an produktive Arbeitsphasen.

2.5 Forschungslücken

Obwohl zahlreiche Studien konsistent zeigen, dass Entwicklerzufriedenheit mit positiven Ergebnissen wie höherer Produktivität, besserer Codequality und stärkerer Mitarbeiterbindung einhergeht **sadowksi_happiness_2019**, [5], ist die Kausalität bislang nicht abschliessend geklärt. Unklar bleibt, ob hohe Zufriedenheit die Produktivität steigert oder ob erfolgreiche Arbeitsergebnisse wiederum zu höherer Zufriedenheit führen. Wahrscheinlich wirken beide Richtungen wechselseitig zusammen.

Ein weiterer offener Punkt sind individuelle Unterschiede. Entwicklerzufriedenheit ist ein subjektives Empfinden, das von Persönlichkeit, Erfahrung und privatem Umfeld beeinflusst wird. Was für die eine Person ein Motivationsfaktor ist, zum Beispiel ein anspruchsvolles neues Projekt, kann für eine andere als belastend empfunden werden. Daraus ergibt sich die Notwendigkeit, personalisierte Ansätze zu entwickeln.

Zudem fehlen bislang belastbare Langzeitstudien aus realen Unternehmensumgebungen. Viele bisherige Arbeiten basieren auf Querschnittsumfragen oder Laborexperimenten, die nur Momentaufnahmen liefern. Langfristige Analysen könnten aufzeigen, wie sich dauerhaft hohe oder niedrige Zufriedenheit über mehrere Jahre auf zentrale Kennzahlen wie Codequality, Defektdichte oder Innovationsrate auswirkt.

2.6 Bestehende Lösungen und Wettbewerbsanalyse

Der Markt bietet bereits verschiedene Tools und Plattformen, die Teilespekte der Entwicklerzufriedenheit adressieren. Im Folgenden werden einige repräsentative bestehende Lösungen vorgestellt und hinsichtlich ihres Fokus und ihrer Lücken bewertet:

Officevibe: Ein Software-as-a-Service-Tool, das wöchentliche Pulse-Umfragen an Mitarbeitende verschickt. Ziel ist es, Engagement und Stimmung im Unternehmen kontinuierlich zu messen. Officevibe bietet anonyme wöchentliche Kurzbefragungen per E-Mail oder Chat an, deren Ergebnisse in übersichtlichen Dashboards für Teamleiter aufbereitet werden. Entwicklerteams erhalten dadurch Stimmungs-Trendkurven und

allgemeines Mitarbeiterfeedback. Allerdings ist Officevibe eher generisch auf Mitarbeiterengagement ausgerichtet und liefert keine speziell auf Entwickler zugeschnittenen Kontextdaten, z.B. werden keine technischen Metriken aus der Softwareentwicklung einbezogen [13].

TeamMood: Ein schlankes Stimmungsbarometer für Teams, das täglich eine einfache Mood-Abfrage durchführt. TeamMood sendet jedem Teammitglied jeden Tag einen kurzen Prompt (per E-Mail, Slack, microsoft teams etc.), in dem die Person mit einem Klick ihre aktuelle Stimmung angibt. Die Antworten werden als anonymer Team-Stimmungsverlauf visualisiert, was es erlaubt, Trends über die Zeit zu erkennen. Die Hürde zur Teilnahme ist sehr niedrig (niedrigschwelliges Feedback). Jedoch erfasst TeamMood keine technischen Prozessmetriken (wie Code-Commits, Buildzeiten o.ä.) und bietet keine automatisierten Empfehlungen. Das bedeutet, dass Entwickler und Manager die Stimmungsverläufe selbst interpretieren und Massnahmen ableiten müssen, ohne direkte Handlungsempfehlungen durch das Tool [14].

Happimeter: Hervorgegangen aus einem Forschungsprojekt (u.a. TU Wien und MIT) setzt Happimeter auf Wearable Sensoren, um einen persönlichen Happiness-Score zu bestimmen. Entwickler tragen z.B. eine Smartwatch mit der Happimeter-App, die kontinuierlich physiologische Daten wie Herzfrequenz, Bewegung oder Schlaf erfasst. Ein Machine-Learning-Modell sagt daraus die aktuelle Stimmung bzw. den Stresslevel der Person voraus. Auf diese Weise sollen objektive Gesundheitsmetriken mit dem subjektiven Wohlbefinden verknüpft werden. Der Ansatz liefert interessante biometrische Einblicke (z.B. Stressspitzen während der Arbeit), jedoch fehlt der direkte Bezug zur eigentlichen Entwicklungsarbeit. Happimeter weiss nichts über Tasks, Code oder Arbeitskontext, sodass die sensorbasierten Glücks-Werte ohne diesen Kontext schwer zu interpretieren sind [15].

Code Climate Velocity: Code Climate Velocity: Eine Datenplattform, die Entwicklungsmetriken aus Git-Repositories analysiert, um die Team-Performance zu bewerten. Velocity konzentriert sich voll auf quantitative Software-Engineering-Kennzahlen. Es misst etwa die Durchlaufzeit von Pull Requests, die Commit-Frequenz, die Review-Geschwindigkeit und diverse weitere Metriken der Entwicklungspipeline. Auch Industrie-Standards wie die vier DORA-Metriken (Deployment Frequency, Lead Time for Changes, Change Failure Rate, Mean Time to Restore Service) sind integriert. Dadurch erhalten Führungskräfte einen detaillierten Blick auf Code-Qualität, Liefergeschwindigkeit und Prozess-Effizienz. Allerdings fehlen subjektive Zufriedenheitsdaten vollständig, die menschliche Stimmungslage der Entwickler wird nicht erfasst. Etwaige Einflüsse von Motivation oder Frustration auf die gemessenen

Leistungsindikatoren bleiben somit unsichtbar [16].

GitHub Insights: Als integrierter Teil von GitHub bietet Insights grundlegende Analysen zur Repository-Aktivität. In jedem GitHub-Repository steht ein Insights-Dashboard zur Verfügung, das Statistiken zu Commits, Pull-Request-Aktivitäten, Issue-Verläufen und Release-Frequenzen visualisiert. Teams können so ihre Entwicklungsaktivität und Geschwindigkeit verfolgen (“Wie viele PRs werden pro Woche gemerged? Wie oft wird deployed?” etc.). Diese Metriken sind wertvolle Aktivitätsstatistiken, berücksichtigen jedoch keine emotionalen Faktoren. GitHub Insights liefert also Kennzahlen zur Produktivität, blendet aber das Stimmungsbild der Entwickler aus. Etwa ob eine Phase hoher Commit-Rate auf Überstunden und Stress zurückzuführen ist, bleibt unklar [17].

Microsoft Viva Insights: Viva Insights ist Teil von Microsoft 365 und zielt darauf ab, durch Analyse von Arbeitsmustern die Produktivität und das Wohlbefinden von Mitarbeitenden zu verbessern. Die Plattform wertet vor allem Kalender- und Kommunikationsdaten aus (z.B. E-Mail- und Teams-Nutzung, Meeting-Häufigkeit). Entwickler erhalten z.B. Hinweise, wenn Meeting-Überlast droht, oder Vorschläge, regelmässige Fokuszeiten für ungestörtes Arbeiten einzuplanen. Führungskräfte sehen aggregierte Team-Insights, etwa ob viele Überstunden anfallen oder wenig Konzentrationsphasen vorhanden sind. Zwar gibt Viva nützliche Empfehlungen (z.B. “Schützen Sie wöchentlich 4 Stunden Fokuszeit” oder “Vermeiden Sie Meetings über 1 Stunde”). Allerdings werden Wohlbefinden und Zufriedenheit nur indirekt aus den Verhaltensdaten abgeleitet, eine direkte Erfassung der Gefühlslage oder ein Bezug zu konkreten Entwickler-Tätigkeiten (Code, Tickets etc.) fehlt vollständig. Somit bleibt die emotionale Dimension in Viva Insights eher implizit und generalisiert [18].

Bestehende Lösungen decken jeweils nur einzelne Aspekte der Entwicklerzufriedenheit ab. Engagement-Tools erfassen Stimmungsdaten, stellen jedoch keinen Bezug zu technischen oder gesundheitlichen Kontextinformationen her. Engineering-Analytics-Tools analysieren Leistungskennzahlen, berücksichtigen jedoch die emotionale Dimension nicht. Gesundheits-Tracker wiederum messen physiologische Daten, ohne diese mit der konkreten Entwicklungsarbeit zu verknüpfen. Eine Plattform, die emotionale Faktoren, technischen Kontext und physisches Wohlbefinden gemeinsam erfasst und daraus konkrete Handlungsempfehlungen ableitet, ist derzeit nicht verfügbar.

2.7 Schlussfolgerungen für die Konzeptlösung

Die Analyse des aktuellen Forschungs- und Praxisstands zeigt, dass Entwicklerzufriedenheit aus einem Zusammenspiel individueller Faktoren (z.,B. Motivation, Flow), organisatorischer Rahmenbedingungen (Team- und Unternehmens-, Meetingkultur) sowie physiologischer Aspekte (Schlaf, Belastung, Aktivitätsniveau, Stress) entsteht. Studien belegen deutliche Zusammenhänge zwischen hoher Zufriedenheit und positiven Ergebnissen wie gesteigerter Produktivität, verbesserter Codequalität und höherer Mitarbeiterbindung. Gleichzeitig wird deutlich, dass bestehende Untersuchungen häufig auf Teilaspekte beschränkt sind und kausale Wirkmechanismen oder langfristige Entwicklungen nur unzureichend abbilden.

Die Betrachtung bestehender Lösungen verdeutlicht, dass aktuelle Ansätze meist entweder subjektive Stimmungsdaten, technische Leistungskennzahlen oder Gesundheitsmetriken isoliert erfassen. Eine integrierte Lösung, die alle relevanten Einflussfaktoren kombiniert, kontextualisiert und in konkrete Handlungsempfehlungen überführt, ist bislang nicht verfügbar.

Diese Lücke bildet den Ausgangspunkt für die im weiteren Verlauf entwickelte Konzeptlösung: Eine Plattform, die Entwicklerzufriedenheit kontinuierlich, kontextbezogen und datenbasiert misst, um fundierte Verbesserungen in Arbeitsabläufen, Teamkultur und individueller Arbeitsweise zu ermöglichen.

Kapitel 3

Konzeptlösung für die Erweiterungen an Yappi

Die im vorhergehenden Kapitel beschriebenen Problemstellungen und die Analyse bestehender Lösungen verdeutlichen, dass aktuelle Ansätze zur Erfassung der Entwicklerzufriedenheit häufig unvollständig sind. Sie erfassen den Einfluss einzelner Arbeitsereignisse nicht systematisch und bieten nur eingeschränkte Möglichkeiten, die erhobenen Daten im individuellen Arbeitskontext zu interpretieren. Zudem werden potenziell relevante Einflussfaktoren, wie das physische Wohlbefinden, nur selten berücksichtigt.

Mit den geplanten Erweiterungen soll Yappi genau diese Lücken schliessen und eine integrierte, praxisorientierte Lösung bereitstellen, die Entwicklerinnen, Entwickler und Teams gleichermaßen unterstützt. Ziel ist es, die Entwicklerzufriedenheit umfassend zu erfassen und die gewonnenen Erkenntnisse in konkrete Handlungsempfehlungen zu überführen.

Die Erweiterung basiert auf folgenden Ansätzen:

- Literaturrecherche zum Stand der Forschung im Bereich der Entwicklerzufriedenheit und deren Einfluss auf die Produktivität
- Analyse bestehender Lösungen zur Erfassung und Auswertung von Entwicklerzufriedenheit, Arbeitskontext und Gesundheitsdaten im Umfeld der Softwareentwicklung
- Befragungen von Entwicklerinnen und Entwicklern in den Unternehmen der Autorinnen und Autoren
- Brainstorming mit den Projektbetreuenden

In den folgenden Unterkapiteln wird die schrittweise Entwicklung der Konzeptlösung für

die geplanten Erweiterungen von Yappi detailliert beschrieben.

3.1 Problemstellungen

Die Erweiterung von Yappi verfolgt das Ziel, bestehende Schwächen in der Erfassung und Interpretation der Entwicklerzufriedenheit zu beheben. Dazu wurden die relevanten Herausforderungen nach dem Standard des Rational Unified Process (RUP) strukturiert beschrieben. Jede Problemstellung wird dabei in einem einheitlichen Schema dargestellt, um die Auswirkungen klar herauszuarbeiten und eine fundierte Basis für die Definition der Projektziele zu schaffen.

Die folgenden Abschnitte fassen die drei zentralen Problemstellungen zusammen, die mit der Erweiterung adressiert werden sollen.

3.1.1 Passive Erfassung der Entwicklerzufriedenheit

In vielen aktuellen Umsetzungen erfolgt die Erfassung der Zufriedenheit rein passiv, d. h., Entwicklerinnen und Entwickler müssen von sich aus aktiv eine Rückmeldung abgeben. Dieser Prozess ist stark von der Eigeninitiative abhängig und wird im Arbeitsalltag häufig vergessen oder aufgeschoben. Dadurch entsteht eine unvollständige und unregelmäßige Datengrundlage, die den tatsächlichen Verlauf der Zufriedenheit nur eingeschränkt widerspiegelt. Die vollständige Problemstellung ist in Tabelle 3.1 dargestellt.

Tabelle 3.1: Problemstellung nach RUP: Passive erfassung der Entwicklerzufriedenheit

Das Problem	der passiven Erfassung der Entwicklerzufriedenheit, bei der ein Entwickler von sich aus eine Rückmeldung abgeben muss
betrifft	Entwicklerinnen und Entwickler.
Die Auswirkung dieses Problems	ist eine geringe Erfassungsrate der Zufriedenheitsdaten, da Feedback oft vergessen oder aufgeschoben wird.
Eine erfolgreiche Lösung	fordert Entwicklerinnen und Entwickler zu relevanten und regelmässigen Zeitpunkten automatisiert dazu auf, Zufriedenheitsdaten zu erfassen. Dabei darf kein grosser Mehraufwand entstehen, um zur Abgabe zu motivieren.

3.1.2 Einfluss einzelner Meetings

Meetings nehmen einen wesentlichen Teil der Arbeitszeit von Entwicklerinnen und Entwicklern ein. Werden sie als unproduktiv oder belastend wahrgenommen, kann dies die Arbeitszufriedenheit deutlich beeinträchtigen. Derzeit wird jedoch der Einfluss einzelner

Besprechungen auf die Zufriedenheit nicht systematisch erfasst, was eine gezielte Verbesserung der Meetingkultur erschwert. Die vollständige Problemstellung ist in Tabelle 3.2 dargestellt.

Tabelle 3.2: Problemstellung nach RUP: Einfluss auf die Zufriedenheit einzelner Meetings wird nicht erfasst

Das Problem	dass der Einfluss einzelner Meetings auf die Zufriedenheit nicht erfasst wird
betrifft	Scrum Master, Product Owner, Entwicklerinnen und Entwickler.
Die Auswirkung dieses Problems	ist, dass unproduktive oder belastende Besprechungen schwer identifiziert werden können und deren Auswirkungen auf die tägliche Arbeit unbekannt bleiben.
Eine erfolgreiche Lösung	erfasst nach relevanten Besprechungen zeitnah die wahrgenommene Produktivität und Belastung, um den Einfluss einzelner Meetings auf die Arbeitszufriedenheit sichtbar zu machen.

3.1.3 Schwierigkeiten beim Ableiten von Schlussfolgerungen

Die Erhebung reiner Zufriedenheitswerte ohne weiterführende Informationen erschwert es, deren Ursachen oder Auslöser zu verstehen. Ohne zusätzlichen Kontext ist es schwierig, Muster zu erkennen oder Zusammenhänge zwischen bestimmten Arbeitsbedingungen und der Zufriedenheit herzustellen. Dies führt dazu, dass Massnahmen zur Verbesserung oft auf Annahmen basieren und nicht ausreichend datenbasiert sind. Die vollständige Problemstellung ist in Tabelle 3.3 dargestellt.

Tabelle 3.3: Problemstellung nach RUP: Schwierigkeiten beim ableiten von Schlussfolgerungen aus Zufriedenheitsdaten

Das Problem	dass es schwierig ist, aus den erfassten Zufriedenheitsdaten klare Schlussfolgerungen abzuleiten
betrifft	Scrum Master, Product Owner, Projektverantwortliche, Entwicklerinnen und Entwickler.
Die Auswirkung dieses Problems	ist, dass unklar bleibt, welche Faktoren oder Ereignisse zu den gemessenen Ergebnissen geführt haben, wodurch gezielte Massnahmen zur Verbesserung erschwert werden.
Eine erfolgreiche Lösung	ergänzt die Zufriedenheitsmessungen um automatisch erfasste Kontextinformationen aus dem Arbeitsumfeld sowie ausgewählte Gesundheitsdaten, um die Ergebnisse besser einordnen und deren Ursachen gezielter identifizieren zu können. Zusätzlich leitet sie konkrete Handlungsempfehlungen aus diesen Daten ab.

3.2 Idee

Die Erweiterung von Yappi verfolgt das Ziel, die Erfassung und Interpretation der Entwicklerzufriedenheit zu verbessern und deren Aussagekraft zu erhöhen. Grundlage ist die Annahme, dass eine ganzheitliche Erhebung, ergänzt durch Informationen aus dem Arbeitsumfeld, ein genaueres Verständnis der Einflussfaktoren ermöglicht. In bestehenden Lösungen werden Zufriedenheitswerte oft isoliert erfasst, was die Ableitung konkreter Verbesserungsmassnahmen erschwert.

3.2.1 Automatisierte Aufforderung zur Erfassung von Zufriedenheitsdaten

Ein zentraler Bestandteil der Erweiterung ist, dass Entwicklerinnen und Entwickler nicht mehr ausschliesslich aus eigener Initiative Zufriedenheitsdaten erfassen müssen. Stattdessen werden sie zu geeigneten Zeitpunkten automatisch dazu aufgefordert, kurze Rückmeldungen zu geben. Es wird angenommen, dass diese Form der aktiven Aufforderung die Häufigkeit der Erfassungen erhöht und somit eine vollständigere sowie aussagekräftigere Datengrundlage schafft.

Ein relevanter Zeitpunkt zur Aufforderung zur Erfassung von Zufriedenheitsdaten sollte so gewählt werden, dass der Entwickler nicht aus seinem Arbeitsfluss herausgerissen wird. Geeignete Zeitpunkte dafür sind:

- **Nach einem Commit:** Entwicklerinnen und Entwickler übermitteln im Arbeitsalltag regelmässig ihre vorgenommenen Codeänderungen an das Versionsverwaltungssystem (Commit). Direkt nach dem Abschluss einer Entwicklungsaufgabe oder eines Arbeitspakets wird der bestehende Arbeitsfluss ohnehin kurz unterbrochen, um den Commit auszuführen. Dieser Moment eignet sich daher, um eine kurze Rückmeldung zu erfassen, ohne den Entwickler aus einer konzentrierten Arbeitssituation herauszureißen. Gleichzeitig ist die Erinnerung an den gerade abgeschlossenen Arbeitsprozess noch präsent, was eine präzisere Einschätzung ermöglicht.
- **Nach Meetings:** Da Meetings selbst bereits eine Unterbrechung des regulären Arbeitsflusses darstellen, verursacht die Erfassung der Zufriedenheit direkt im Anschluss keine zusätzliche Störung. In diesem Moment können die Teilnehmenden ausserdem am besten einschätzen, wie produktiv, zielführend und angenehm die Besprechung empfunden wurde, wodurch die Angaben besonders aussagekräftig sind.

3.2.2 Erfassung der Meetingqualität

Ein weiterer Bestandteil der Erweiterung ist die Integration einer Funktion zur Bewertung von Meetings. Ziel ist es, nach Abschluss einer Besprechung zeitnah eine kurze Rückmeldung zur wahrgenommenen Produktivität und Relevanz zu erfassen. Es wird angenommen, dass durch die systematische Erhebung solcher Rückmeldungen der Einfluss einzelner Meetings auf die Arbeitszufriedenheit sichtbar wird und so eine fundierte Grundlage für Verbesserungen der Meetingkultur entsteht.

Zur Bewertung werden sieben Kriterien herangezogen, die in Anlehnung an Best Practices und gängige Meeting-Umfrageinstrumente ausgewählt wurden:

- **Zielklarheit und Relevanz:** Bewertet, ob die Ziele des Meetings klar formuliert und inhaltlich relevant waren. Eine präzise Agenda und eindeutige Zieldefinition gelten als zentrale Erfolgsfaktoren.
- **Inhaltliche Tiefe und Verständlichkeit:** Misst, ob die behandelten Themen angemessen tief und gleichzeitig verständlich präsentiert wurden. Unnötige Detailfülle wird vermieden.
- **Zeitmanagement:** Prüft, ob die geplante Dauer eingehalten und das Meeting pünktlich begonnen wurde. Studien zeigen, dass bewusst kürzere Meetings (z. B. 25 statt 30 Minuten) die Effizienz steigern können.
- **Moderation und Beteiligung:** Erfasst die Qualität der Moderation und die aktive Einbindung der Teilnehmenden. Eine hohe Beteiligungsquote gilt als Indikator für Engagement und Interaktivität.
- **Ergebnisorientierung:** Bewertet, ob aus dem Meeting konkrete Entscheidungen oder Aufgaben resultierten. Die Anzahl abgeschlossener Aktionspunkte kann als Kennzahl dienen.
- **Allgemeine Zufriedenheit:** Ermittelt das subjektive Gesamurteil der Teilnehmenden, beispielsweise auf einer Skala von 1 bis 10.
- **Meetingdauer:** Vergleicht die geplante mit der tatsächlichen Dauer und bewertet, ob diese angemessen war.

3.2.3 Verknüpfung von Zufriedenheitswerten mit Kontext- und Gesundheitsdaten

Ein weiterer Bestandteil der Erweiterung ist die Verknüpfung der erfassten Zufriedenheitswerte mit zusätzlichen, für die Interpretation relevanten Einflussfaktoren. Ziel ist es,

die Aussagekraft der Daten zu erhöhen und ein umfassenderes Bild der Zusammenhänge zwischen Arbeitsbedingungen und Arbeitszufriedenheit zu erhalten. Hierzu werden sowohl Daten aus dem Arbeitskontext als auch ausgewählte Gesundheitsdaten berücksichtigt.

Unter Arbeitskontextdaten fallen Commitinformationen, wie Zeitpunkt und die Commit Message, welche eine kurze Zusammenfassung der Codeänderungen darstellt. Außerdem werden zusätzliche Informationen zu Meetings erfasst, beispielsweise der Name des Meetings oder die Anzahl teilnehmender Personen.

Die Ergänzung der Zufriedenheitswerte um Gesundheitsdaten soll das subjektive Stimmungsbild erweitern und objektive Indikatoren für Belastung, Erholung und allgemeines Wohlbefinden einbeziehen. Während subjektive Angaben wertvolle Einblicke in die aktuelle emotionale Lage geben, erfassen physiologische Metriken Veränderungen, die den Betroffenen nicht immer unmittelbar bewusst sind. Die Kombination beider Perspektiven ermöglicht eine fundiertere Analyse möglicher Einflussfaktoren auf die Arbeitszufriedenheit von Entwicklerinnen und Entwicklern.

Für die Auswahl der relevanten Gesundheitsmetriken wurden drei Kriterien herangezogen:

- **Wissenschaftlich belegter Zusammenhang** mit Arbeitszufriedenheit oder Leistungsfähigkeit.
- **Technische Messbarkeit** mittels gängiger Wearables bzw. Smartphone-Sensoren.
- **Integrationsfähigkeit** über etablierte Schnittstellen wie Apple Health Kit oder die Garmin Health API.

Die im Kapitel 2.3 beschriebenen Gesundheitsmetriken erfüllen diese Kriterien und umfassen:

- **Schlafdauer**
- **Ruheherzfrequenz (RHR)**
- **Stress** (gemessen über die Herzratenvariabilität, HRV)
- **Aktivitätsminuten und Schritte**

3.2.4 Automatisierte Interpretation und Handlungsempfehlungen durch den Yappi Coach

Als letzter Bestandteil der Erweiterung wertet der Yappi Coach die erfassten Daten automatisiert aus. Ziel ist es, nicht nur Rohdaten bereitzustellen, sondern diese in einen

interpretierbaren Kontext zu setzen und daraus konkrete, umsetzbare Handlungsempfehlungen abzuleiten.

Der Yappi Coach kombiniert Zufriedenheitswerte, Kontextinformationen und ausgewählte Gesundheitsmetriken zu einem Gesamtbild der aktuellen Arbeitssituation. Dabei werden wiederkehrende Muster, Auffälligkeiten und Abweichungen von individuellen Referenzwerten identifiziert. Durch diese ganzheitliche Betrachtung können potenzielle Ursachen für Veränderungen in der Arbeitszufriedenheit erkannt werden.

Ein wesentliches Ziel ist es, die Entwicklerinnen und Entwickler direkt bei der Verbesserung ihrer Arbeitssituation zu unterstützen. Statt lediglich Zahlen und Diagramme anzuzeigen, formuliert der Yappi Coach daraus spezifische Empfehlungen, die auf die jeweilige Person zugeschnitten sind. Beispiele hierfür sind:

- Vorschläge zur Anpassung der Meetingfrequenz oder -dauer, wenn wiederholt ein negativer Zusammenhang zwischen Meetingbelastung und Zufriedenheit erkennbar ist.
- Hinweise auf mögliche Überlastung, wenn Gesundheitsmetriken wie Ruheherzfrequenz oder Schlafdauer über einen längeren Zeitraum ungünstige Werte aufweisen.
- Empfehlungen zur Optimierung des Arbeitsrhythmus, wenn bestimmte Tageszeiten oder Arbeitsphasen wiederholt mit höheren Zufriedenheitswerten korrelieren.

Der Yappi Coach soll somit die Brücke zwischen Datenerhebung und konkreter Verbesserung schlagen. Durch die kontinuierliche, automatisierte Analyse der Daten wird eine proaktive Unterstützung möglich, die nicht nur reaktiv auf bestehende Probleme eingeht, sondern auch frühzeitig präventive Massnahmen anstösst.

3.3 Vision, Mission und Leitprinzipien

Die Weiterentwicklung von Yappi folgt einer klaren strategischen Ausrichtung, die sowohl die langfristigen Ziele (Vision) als auch den konkreten Handlungsauftrag (Mission) definiert. Dieses Kapitel beschreibt, welches Zukunftsbild mit Yappi angestrebt wird und nach welchen Grundsätzen die Umsetzung erfolgt.

3.3.1 Vision: Yappi als zentrale Plattform für Entwicklerzufriedenheit

Yappi soll sich zu einer zentralen, intelligenten Plattform entwickeln, die es Entwicklerinnen, Entwicklern und Teams ermöglicht, die Arbeitszufriedenheit kontinuierlich, präzise und im

Kontext zu erfassen, zu verstehen und gezielt zu verbessern. Ziel ist ein Arbeitsumfeld, in dem Zufriedenheit als gleichwertiger Faktor neben Produktivität und Codequalität betrachtet wird und in dem datenbasierte Erkenntnisse zu einer nachhaltig gesunden und motivierenden Arbeitskultur beitragen.

3.3.2 Mission: Nahtlose Erfassung, Analyse und Optimierung im Arbeitsalltag

Die Mission von Yappi besteht darin, durch nahtlose Integration in den Arbeitsalltag verlässliche und kontextbezogene Zufriedenheitsdaten zu erfassen, diese mit relevanten Kontext- und Gesundheitsmetriken zu verknüpfen und mithilfe intelligenter Analysen konkrete Handlungsempfehlungen bereitzustellen. Dabei wird besonderer Wert darauf gelegt, den Erfassungs- und Auswertungsprozess so zu gestalten, dass er minimal störend und auf den individuellen wie auch den Nutzen für Teams ausgerichtet ist.

3.3.3 Leitprinzipien: Werte und Ausrichtung der Plattform

Die folgenden Leitsätze definieren den Rahmen, an dem sich die Weiterentwicklung von Yappi orientiert:

1. **Ganzheitlichkeit vor Fragmentierung:** Zufriedenheitswerte werden immer im Kontext weiterer relevanter Faktoren betrachtet.
2. **Nahtlose Integration:** Die Nutzung soll den Arbeitsfluss nicht unterbrechen, sondern sich harmonisch einfügen.
3. **Datenbasiert und handlungsorientiert:** Die Analysen zielen stets auf konkrete, umsetzbare Handlungsempfehlungen.

3.4 Value Proposition

Dieses Kapitel beschreibt den konkreten Mehrwert, den die Erweiterung von Yappi für verschiedene Zielgruppen bietet. Dabei wird zwischen Einzelpersonen und Teams unterschieden, um die Vorteile jeweils im passenden Anwendungskontext darzustellen.

3.4.1 Mehrwert für Einzelpersonen – Persönliche Optimierung der Arbeitszufriedenheit

Die Erweiterung von Yappi ermöglicht es Entwicklerinnen und Entwicklern, ihre Arbeitszufriedenheit regelmäßig und ohne hohen Mehraufwand zu erfassen. Durch die automatisierte

Aufforderung zu geeigneten Zeitpunkten, die Anreicherung der Daten mit Kontext- und Gesundheitsinformationen sowie die Interpretation durch den Yappi Coach, erhalten Einzelpersonen fundierte und individuell relevante Rückmeldungen.

Die wichtigsten Vorteile für Einzelpersonen sind:

- **Regelmässige, unkomplizierte Erfassung** von Zufriedenheitsdaten ohne manuelles Initiieren.
- **Individuell zugeschnittene Empfehlungen** zur Verbesserung der eigenen Arbeitssituation.
- **Frühzeitige Erkennung von Belastungstendenzen** durch Kombination subjektiver und objektiver Daten.
- **Bessere Selbstreflexion** durch kontinuierliche und leicht zugängliche Verlaufsauswertungen.

3.4.2 Mehrwert für Teams – Gemeinsame Verbesserung von Zusammenarbeit und Prozessen

Auf Teamebene bietet die Erweiterung die Möglichkeit, kollektive Muster in der Arbeitszufriedenheit zu erkennen. Die gesammelten Daten können aufzeigen, wie sich bestimmte Arbeitsweisen, Meetingstrukturen oder Prozessänderungen auf die Gesamtzufriedenheit im Team auswirken.

Die wichtigsten Vorteile für Teams sind:

- **Transparenz über gemeinsame Herausforderungen** durch aggregierte Zufriedenheitsdaten.
- **Fundierte Entscheidungsgrundlage** für Prozess- und Arbeitsablaufoptimierungen.
- **Verbesserung der Meetingkultur** durch systematische Erfassung der Meetingqualität.
- **Stärkung der Zusammenarbeit** durch gezielte, datenbasierte Anpassungen.

3.5 Produktziele

Die im vorangehenden Kapitel beschriebenen Erweiterungsideen bilden die Grundlage für die folgenden Produktziele. Sie fassen die wesentlichen Eigenschaften zusammen, die

Yappi im Rahmen der geplanten Weiterentwicklung erfüllen soll, um die identifizierten Problemstellungen wirksam zu adressieren.

3.5.1 Z-1: Automatisierte Aufforderung zur Zufriedenheitserfassung

Ziel:

Yappi soll Entwicklerinnen und Entwickler automatisch zu geeigneten Zeitpunkten auffordern, Zufriedenheitsdaten zu erfassen, ohne den Arbeitsfluss zu stören. Dadurch wird eine kontinuierliche und vollständige Datengrundlage geschaffen, die eine präzisere Analyse der Zufriedenheit ermöglicht.

Beschreibung:

Die automatisierte Aufforderung erfolgt kontextbezogen, beispielsweise direkt nach einem Commit oder unmittelbar nach einem Meeting. Dies gewährleistet, dass die Erfassung in Momenten erfolgt, in denen der Arbeitsfluss ohnehin kurz unterbrochen ist, und reduziert so den wahrgenommenen Mehraufwand für die Nutzerinnen und Nutzer.

Bezug zur Forschungsfrage:

Bezieht sich auf Forschungsfrage A: Durch den Einsatz von Technologien und Schnittstellen wie IDE-Plugins, Browsererweiterungen oder Integrationen in Tools wie Microsoft Teams und Outlook wird ein reibungsloses und einfaches Erfassen von Zufriedenheitsdaten ermöglicht.

3.5.2 Z-2: Erfassung der Meetingqualität

Ziel:

Yappi soll nach relevanten Meetings zeitnah eine kurze Rückmeldung zur wahrgenommenen Produktivität, Relevanz und Belastung einholen, um den Einfluss einzelner Besprechungen auf die Arbeitszufriedenheit sichtbar zu machen.

Beschreibung:

Die Bewertung erfolgt anhand vordefinierter Kriterien wie Zielklarheit, Moderationsqualität, Zeitmanagement und Ergebnisorientierung. Die Integration in gängige Meeting-Tools stellt sicher, dass die Abfrage ohne zusätzlichen manuellen Aufwand in den Arbeitsalltag eingebettet wird.

Bezug zur Forschungsfrage:

Bezieht sich auf Forschungsfrage A: Durch die Integration in Meeting-Plattformen wie Microsoft Teams oder Outlook werden bestehende Technologien genutzt, um die Erfassung im direkten Arbeitskontext zu automatisieren und den Erfassungsprozess für die Nutzerinnen und Nutzer zu vereinfachen.

3.5.3 Z-3: Verknüpfung mit Kontext- und Gesundheitsdaten

Ziel:

Yappi soll Zufriedenheitswerte mit automatisch erfassten Kontextinformationen aus dem Arbeitsumfeld sowie ausgewählten Gesundheitsdaten verknüpfen, um ein vollständigeres Bild der Einflussfaktoren auf die Arbeitszufriedenheit zu erhalten.

Beschreibung:

Neben Arbeitskontextdaten wie Commit-Zeitpunkt, Commit-Message und Meetinginformationen werden Gesundheitsmetriken wie Schlafdauer, Ruheherzfrequenz, Stress (HRV) sowie Aktivitätsminuten und Schritte einbezogen. Diese Daten werden über etablierte Schnittstellen wie Apple HealthKit oder die Garmin Health API automatisiert integriert.

Bezug zur Forschungsfrage:

Bezieht sich auf Forschungsfrage B: Die Anbindung an Gesundheitsdaten-APIs ermöglicht es, wissenschaftlich relevante und technisch messbare Metriken in die Analyse der Entwicklerzufriedenheit einzubeziehen, um Zusammenhänge zwischen Wohlbefinden und Zufriedenheit datenbasiert zu identifizieren.

3.5.4 Z-4: Automatisierte Interpretation und Handlungsempfehlungen (Yappi Coach)

Ziel:

Yappi soll die erfassten Zufriedenheits-, Kontext- und Gesundheitsdaten automatisiert analysieren und daraus konkrete, individuell zugeschnittene Handlungsempfehlungen ableiten.

Beschreibung:

Der Yappi Coach erkennt Muster, Auffälligkeiten und Abweichungen von individuellen Referenzwerten, um frühzeitig potenzielle Probleme zu identifizieren. Auf Basis dieser

Analysen werden praxisnahe, datengestützte Empfehlungen gegeben, die sowohl präventive Massnahmen als auch Optimierungen im Arbeitsalltag umfassen.

Bezug zur Forschungsfrage:

Bezieht sich auf Forschungsfrage C: Durch die Nutzung KI-gestützter Auswertungen können aus den kombinierten Daten gezielte, evidenzbasierte Vorschläge zur Verbesserung der Arbeitsbedingungen und der Zufriedenheit abgeleitet werden.

Anmerkung:

Dieses Ziel wird im Rahmen der vorliegenden Arbeit nicht technisch umgesetzt, sondern ausschliesslich konzeptionell ausgearbeitet. Die praktische Implementierung ist als Bestandteil eines Nachfolgeprojekts vorgesehen.

3.6 Konzeptevaluation

Ziel der Evaluation ist es, die Relevanz der im Konzeptentwurf beschriebenen Erweiterungen von Yappi aus Sicht der Hauptnutzer zu überprüfen. Hierzu wurde eine Umfrage erstellt, die alle vorgesehenen Konzepte und deren Funktionen abdeckte. Die Bewertung erfolgte mit einer Likert-Skala von 0 bis 10, wobei 0 "sehr unwichtig" und 10 "sehr wichtig" bedeutet. Eine Likert-Skala ist ein in der Sozialforschung weit verbreitetes Verfahren, bei dem Teilnehmende ihre Zustimmung oder Ablehnung zu einer Aussage stufenweise angeben können.

Wichtigkeit der Features ($\bar{\sigma}$ -Bewertung)

Integration in Entwicklerumgebung

IntelliJ Integration

$\bar{\sigma} 8.50$

Browserextension

$\bar{\sigma} 4.17$

Integration von Meetingsdaten

Erfassen der Meetingdauer

$\bar{\sigma} 6.00$

Erfassen des Meetingthema

$\bar{\sigma} 4.50$

Erfassen eines Freitextkommentar

$\bar{\sigma} 2.33$

Yappi-Coach

Persönliche Tipps

$\bar{\sigma} 7.50$

Team Tipps

$\bar{\sigma} 6.67$

Gesundheitsdaten

Erfassen von Gesundheitsdaten für Yappi

$\bar{\sigma} 7.67$

Abbildung 3.1: Durchschnittsbewertung der Konzeptfunktionen auf einer Likert-Skala.

Die Abbildung 3.1 zeigt die nach Themen gruppierten Durchschnittsbewertungen. An der Befragung nahmen acht individuelle Softwareentwickler teil. Am höchsten bewertet wurden Funktionen mit direkter Integration in den Arbeitsfluss, wie das IntelliJ-Plugin (8,50) oder die Erfassung von Gesundheitsdaten (7,67). Niedrigere Werte erhielten Funktionen, die manuelle Eingaben erfordern, insbesondere Freitextkommentare zu Meetings (2,33).

Die Umfrage diente nicht nur der Bestätigung der Konzeptideen, sondern auch der Priorisierung und Optimierung einzelner Funktionen. So ergab sich aus den Rückmeldungen, dass die Meetingumfrage als kompakter Frageblock mit Auswahloptionen gestaltet wird, während Freitextfelder nur ergänzend zum Einsatz kommen. Damit wird der Aufwand für die Teilnehmenden reduziert und die Wahrscheinlichkeit für regelmäßige Rückmeldungen erhöht.

Kapitel 4

Implementierung

Dieses Kapitel beschreibt die technische Umsetzung der im Konzeptteil definierten Erweiterungen der Yappi-Plattform. Ziel der Implementierung war es, die im Rahmen der Produktziele Z-1 bis Z-3 formulierten Anforderungen umzusetzen und die notwendige Systemarchitektur entsprechend zu erweitern. Die Umsetzung erfolgte unter Berücksichtigung bestehender Komponenten von Yappi, um eine nahtlose Integration in die vorhandene Codebasis und Betriebsumgebung zu gewährleisten.

Im Folgenden werden die einzelnen Erweiterungen detailliert beschrieben. Beginnend bei der Anpassung der Backend- und Datenbankstrukturen, über die Entwicklung der Companion-Anwendungen (IntelliJ-Plugin, Kalender-Integration, Health-Companion) bis hin zur Sicherstellung der Datenübertragung und -sicherheit. Abschliessend werden die getroffenen Entscheidungen hinsichtlich Deployment und CI/CD-Prozess erläutert, um die vollständige Betriebsfähigkeit der erweiterten Plattform nachzuvollziehen.

Als Grundlage für alle weiteren Erweiterungen wurde Yappi zunächst zu einer Integrationsplattform ausgebaut. Die dafür geschaffene Integrationsarchitektur bildet den Ausgangspunkt der folgenden Unterkapitel.

4.1 Yappi als Integrationsplattform

Die Erweiterung von Yappi zu einer Integrationsplattform ist notwendig, um Daten aus unterschiedlichen Quellen sicher und zuverlässig zu erfassen. Neben Prozessdaten wie Commits oder Meeting-Aktivitäten sollen auch Gesundheitsdaten aus externen Companion Apps einbezogen werden. Dies erfordert eine Architektur, welche den Datenaustausch zwischen Yappi und externen Anwendungen standardisiert und absichert. Zentrales Element ist ein API-Key-basiertes Authentifizierungsverfahren, das autorisierte Anwendungen

eindeutig identifiziert und deren Zugriff kontrolliert. Auf dieser Basis können Daten aus heterogenen Quellen in ein einheitliches System integriert werden.

4.1.1 Kommunikationsmechanismus: API-Keys

Für die Kommunikation zwischen Companion-Anwendungen und dem Yappi Backend wurde bewusst auf einen API-Key-basierten Mechanismus gesetzt, anstatt die bestehende JWT-basierte Authentifizierung des Frontends zu verwenden. JWTs eignen sich vor allem für die Authentifizierung einzelner Nutzer in interaktiven Sitzungen. Sie erfordern in der Regel eine vorgelagerte Benutzeranmeldung und regelmässige Token-Erneuerung. Dieser Ablauf ist für Companion-Anwendungen, die im Hintergrund oder automatisiert Daten erfassen und übertragen, nicht optimal. Diese würden durch die Notwendigkeit einer manuellen Anmeldung in ihrer Funktionalität eingeschränkt.

Der API-Key-Mechanismus wurde entwickelt, um diesen Anforderungen gerecht zu werden. Jeder Nutzer verfügt über einen persönlichen API-Key, der in der Webanwendung einmalig generiert wird. Dieser Schlüssel wird manuell in den gewünschten Companion-Anwendungen hinterlegt. Bei jeder Anfrage an das Backend wird der API-Key im HTTP-Header übermittelt. Das Backend prüft die Gültigkeit des Schlüssels und ordnet die Anfrage dem entsprechenden Nutzerkonto zu. So wird sichergestellt, dass nur autorisierte Clients im Namen des Nutzers Daten übermitteln können. API-Keys lassen sich bei Bedarf widerrufen oder rotieren. Durch diesen Mechanismus können Companion-Anwendungen zuverlässig und ohne Benutzerinteraktion mit Yappi kommunizieren, während gleichzeitig ein hohes Mass an Sicherheit gewährleistet ist.

Durch diesen Mechanismus können Companion-Anwendungen zuverlässig und ohne wiederkehrende Benutzerinteraktion mit Yappi kommunizieren, während gleichzeitig ein hohes Mass an Sicherheit gewährleistet ist.

4.1.2 Systemarchitektur der Integrationsplattform

Das in Abbildung 4.1 dargestellte Architekturdiagramm zeigt die zentralen Komponenten der Yappi-Integrationsplattform sowie deren Anbindung an externe Companion-Anwendungen. Im Zentrum befindet sich das Yappi Backend, das als zentrale Schnittstelle für alle eingehenden Daten und Anfragen fungiert.

Das Yappi Frontend verwendet die bestehende JWT-basierte Authentifizierung zur Verwaltung von Nutzersitzungen. Externe Companion-Apps, wie beispielsweise die IntelliJ-, Health- oder Calendar-Companion, sind über den API-Key-Mechanismus angebunden. Dieser gewährleistet, dass nur registrierte und autorisierte Clients Daten an das System

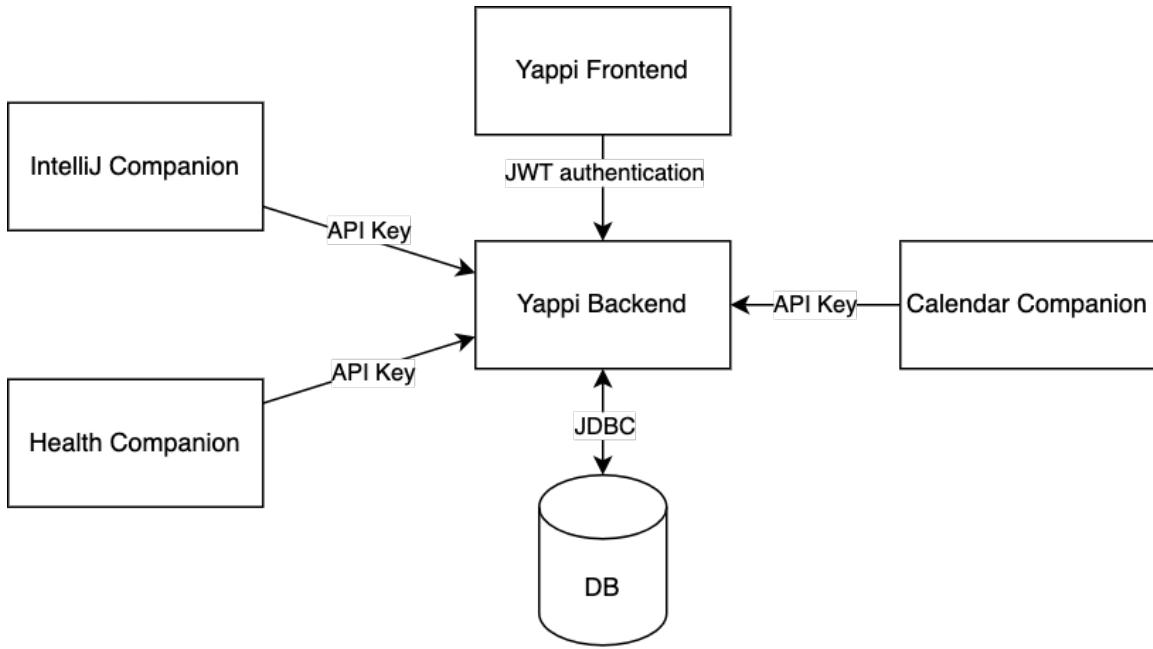


Abbildung 4.1: Systemarchitektur der Yappi-Integrationsplattform

übermitteln können. Alle eingehenden Daten werden vom Backend in der dargestellten Datenbank (DB) persistiert.

4.1.3 Zugriffskontrolle über API-Keys

Das API-Key-System dient der sicheren und eindeutigen Authentifizierung externer Dienste gegenüber der Yappi-Companion-API. Anstelle von Benutzeranmeldedaten verwenden Clients einen statisch generierten API-Key, der für den jeweiligen Nutzer erstellt und ausschliesslich diesem zugeordnet ist. Dies ermöglicht den kontrollierten Zugriff auf geschützte Endpunkte, ohne dass sensible Login-Daten offenliegen oder dauerhaft gespeichert werden müssen.

Über einen gültigen API-Key können autorisierte Clients Anfragen an Endpunkte mit dem Pfadpräfix `/companion/` stellen. Beispielsweise kann der Fragenblock mit der ID 1 wie folgt abgerufen werden:

```
GET https://yappi.dev/companion/questionblocks/1
X-API-KEY: <api-key>
```

Backend-Komponenten

Die Architektur der API-Key-Verwaltung im Backend ist in Abbildung 4.2 dargestellt. Das Klassendiagramm zeigt die zentralen Komponenten und ihre Beziehungen, die für die Erstellung, Verwaltung und Validierung von API-Keys erforderlich sind. Auf Grundlage

dieser Struktur wurde das Backend um die folgenden Elemente erweitert:

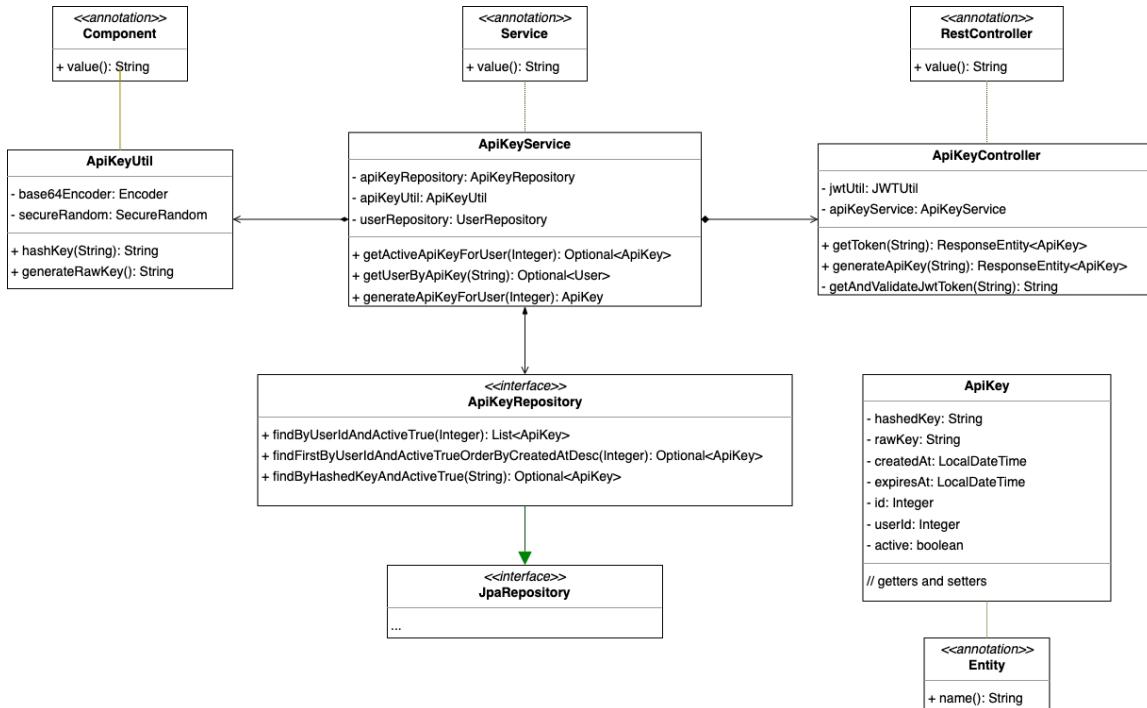


Abbildung 4.2: Klassendiagramm für die Implementierung der API-Keys

- **Entität (ApiKey.java)**: Abbildung der Tabelle `api_key` als JPA-Entität.
- **Repository (ApiKeyRepository.java)**: Datenbankzugriff auf API-Key-Datensätze.
- **Util-Klasse (ApiKeyUtil.java)**: Generierung zufälliger Keys, Hashing mit einem sicheren Algorithmus.
- **Service (ApiKeyService.java)**: Erstellung, Speicherung und Validierung von Keys.
- **Controller (ApiKeyController.java)**: REST-Endpunkte zur Verwaltung von Keys.

Datenbank Erweiterung

Das Datenmodell wurde um Die Tabelle 4.1 `api_key` erweitert. Diese speichert die Metadaten und API-Keys.

Die Speicherung des rohen API-Keys (`raw_key`) dient ausschliesslich dem Zweck, diesen dem Benutzer nach der Erstellung erneut anzeigen zu können. Diese Entscheidung stellt einen bewussten Kompromiss zwischen Benutzerfreundlichkeit und Sicherheit dar, wurde jedoch so umgesetzt, dass das Feld bei Bedarf ohne tiefgreifende Änderungen am System entfernt werden kann.

Spalte	Datentyp	Beschreibung
<code>id</code>	SERIAL	Primärschlüssel, auto-inkrementierend
<code>user_id</code>	INTEGER	Fremdschlüssel auf <code>users.id</code>
<code>hashed_key</code>	TEXT	Gesalteter Hash des API-Keys
<code>raw_key</code>	TEXT	Roher API-Key
<code>created_at</code>	TIMESTAMPTZ	Zeitpunkt der Erstellung des Keys
<code>expires_at</code>	TIMESTAMPTZ	Ablaufdatum des Keys
<code>active</code>	BOOLEAN	Flag, um Keys ohne Löschung zu sperren

Tabelle 4.1: Schema der Tabelle `api_key`

4.1.4 Sicherheit

Der API-Key-Mechanismus der Integrationsplattform ist so konzipiert, dass er eine sichere und kontrollierte Anbindung externer Companion-Anwendungen ermöglicht. Jeder API-Key ist eindeutig einem Nutzerkonto zugeordnet und wird in der Webanwendung einmalig generiert. Die Generierung erfolgt über eine abgesicherte Benutzeroberfläche, wodurch sichergestellt ist, dass ausschliesslich berechtigte Nutzer einen Schlüssel anlegen können. Zur Übertragungssicherheit wird der API-Key ausschliesslich über verschlüsselte Verbindungen (HTTPS) zwischen der Companion-Anwendung und dem Backend übertragen.

Im Backend erfolgt eine serverseitige Validierung, bei der der Schlüssel auf Gültigkeit und Zuordnung geprüft wird. Die Schlüssel werden dabei nicht im Klartext gespeichert, sondern in sicherer Form (z. B. gehasht) abgelegt, um auch bei einem möglichen Datenbankzugriff unbefugten Gebrauch zu verhindern. Ein kompromittierter API-Key kann jederzeit durch den Nutzer oder einen Administrator gesperrt oder durch einen neuen ersetzt werden.

Die technische Umsetzung der API-Key-Prüfung erfolgt über speziell in Spring Security integrierte Komponenten. Diese sorgen dafür, dass API-Key-Anfragen noch vor der regulären Benutzer-Authentifizierung ausgewertet und ggf. blockiert werden:

- **ApiKeyFilter** - Spring-Security-Filter, der den API-Key aus dem X-API-KEY-Header ausliest und validiert.
- **SecurityConfig** - Bindet den Filter vor dem UsernamePasswordAuthenticationFilter in die Filterkette ein, um API-Key-Anfragen vor anderen Authentifizierungsmethoden zu prüfen.

Dieser Prüfmechanismus wird ausschliesslich auf Endpunkten angewendet, deren Pfad den Bestandteil `/companion` enthält, sodass nur für Integrationen mit externen Companion-Anwendungen eine API-Key-Authentifizierung erforderlich ist.

4.1.5 Ablauf der API-Key Erstellung und -Verwendung

Abbildung 4.3 zeigt den Ablauf der Erstellung und Verwendung des API-Keys innerhalb der Yappi-Integrationsplattform. Die Darstellung erfolgt als Swimlane-Diagramm und verdeutlicht die beteiligten Akteure sowie deren Interaktionen während des gesamten Prozesses.

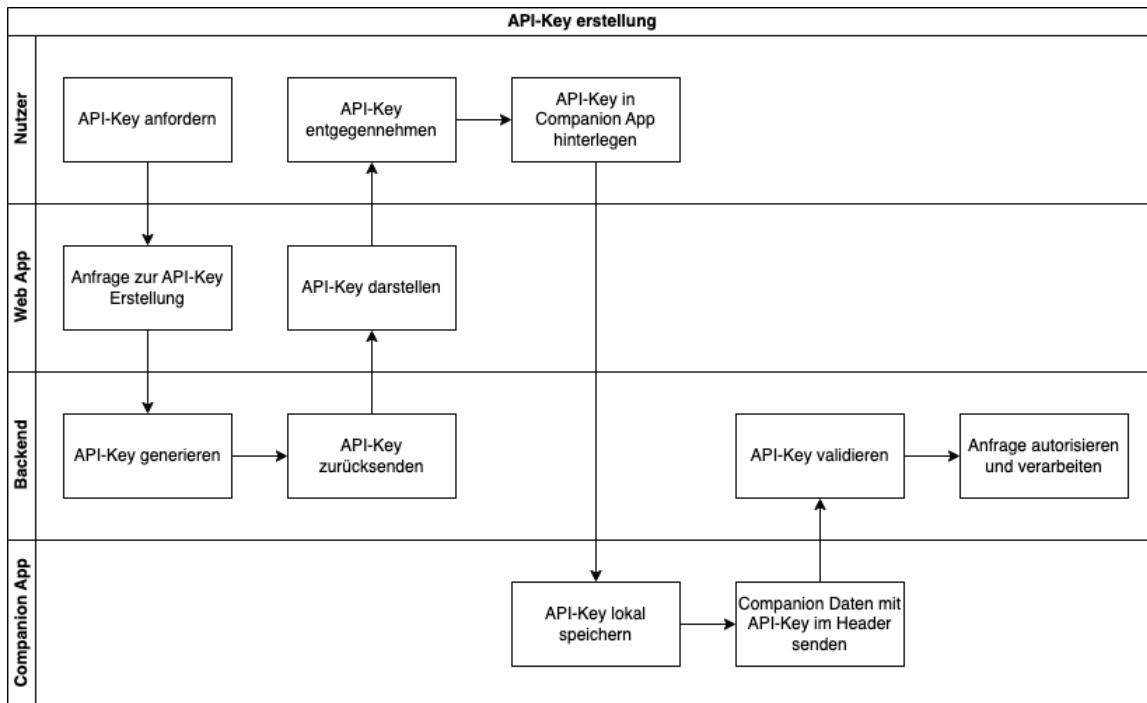


Abbildung 4.3: Ablauf der API-Key Erstellung und -Verwendung in Yappi

Der Prozess beginnt mit der einmaligen Generierung des API-Keys durch den Nutzer in der Webanwendung. Nach der Erstellung wird der Schlüssel in der Companion-App hinterlegt und dort dauerhaft gespeichert. Jede Datenanfrage der Companion-App an das Backend enthält den API-Key im HTTP-Header, woraufhin das Backend den Schlüssel validiert und die Anfrage bei positiver Prüfung autorisiert. Dieser Ablauf stellt sicher, dass ausschliesslich autorisierte Anwendungen im Namen eines Nutzers Daten an Yappi übermitteln können.

API-Endpunkte zur API-Key Generierung

Um einen API-Key über die Webapplikation zu erstellen stehen die folgenden Endpunkte zur Verfügung.

Diese Endpunkte setzen einen gültigen JWT im Authorization-Header voraus (Bearer <token>) und sind somit nur über das Yappi Frontend erreichbar.

- GET /apikey

Liefert den aktiven API-Key des aktuell authentisierten Nutzers.

Antworten: 200 OK, 404 Not Found, 401 Unauthorized (fehlender/ungültiger Header).

- POST /apikey/generate

Erzeugt einen neuen API-Key für den authentisierten Nutzer und gibt den API-Key zurück.

Antworten: 200 OK, 401 Unauthorized.

4.1.6 Benutzeroberfläche zur API-Key-Verwaltung

Die API-Key-Verwaltung wurde im Frontend in den Benutzereinstellungen integriert, um Companion-Anwendungen schnell und unkompliziert mit Yappi verbinden zu können. Die Benutzeroberfläche ermöglicht es, vorhandene API-Keys einzusehen, neue Keys zu generieren und diese für die weitere Nutzung bequem zu exportieren.

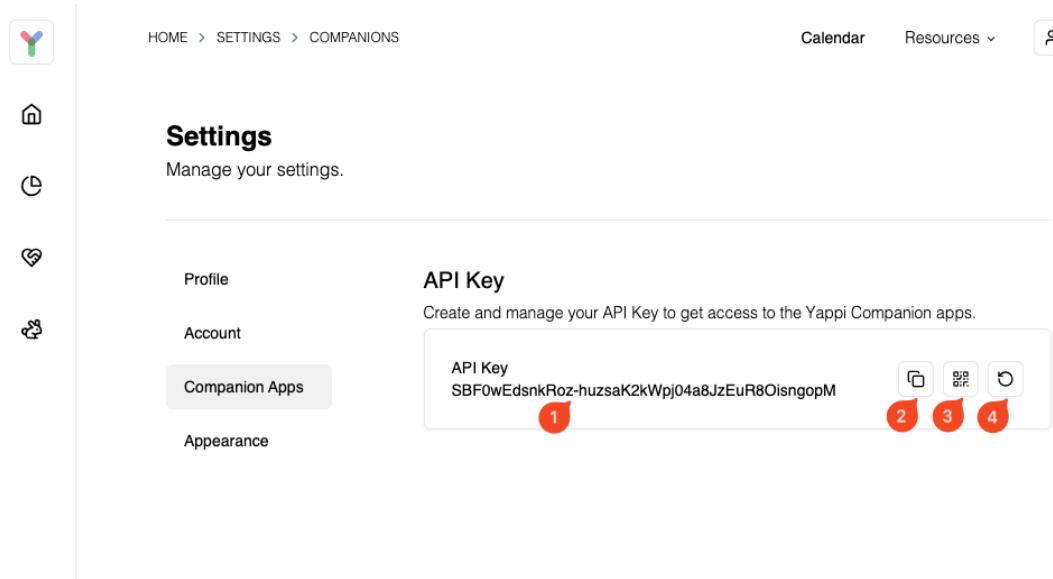


Abbildung 4.4: Benutzeroberfläche für die Verwaltung von API-Keys im Frontend

Abbildung 4.4 zeigt die zentralen Elemente und Funktionen der Implementierung im Frontend:

1. **API-Key-Anzeige:** Zeigt den aktuell gültigen API-Key an.
2. **Kopierfunktion:** Kopiert den API-Key in die Zwischenablage, um ihn in Companion-Anwendungen einzufügen.

3. **QR-Code-Anzeige:** Zeigt den API-Key als QR-Code an, um ihn mit der mobilen Health Companion App zu scannen.
4. **Rotation des API-Keys:** Erzeugt einen neuen API-Key und deaktiviert den bisherigen Schlüssel sofort.

4.2 IntelliJ IDEA Companion

Die Implementierung nutzt die von JetBrains bereitgestellten IntelliJ-Plugin-APIs, um sich direkt in den Entwicklungsablauf der IDE einzuklinken. Diese APIs ermöglichen es, Ereignisse wie einen erfolgreichen Code-Commit abzufangen, eigene Dialogfenster oder Benachrichtigungen anzuzeigen und auf gespeicherte Plugin-Einstellungen zuzugreifen. Dadurch kann das Yappi Companion-Plugin nahtlos in die bestehende IntelliJ-Oberfläche integriert werden, ohne den gewohnten Workflow der Entwickler zu unterbrechen.

Der IntelliJ Companion erweitert Yappi um die Möglichkeit, direkt aus der Entwicklungsumgebung heraus Feedback zu Code-Commits zu erfassen. Ziel ist es, die Erfassung von Stimmungsdaten möglichst nahtlos in den Entwickler-Workflow zu integrieren, ohne den Arbeitsfluss zu unterbrechen. Nach jedem erfolgreichen Commit in IntelliJ IDEA öffnet sich automatisch ein Dialogfenster, in dem der Entwickler seine Stimmung zum Commit angeben kann. Diese Daten werden anschliessend an Yappi übertragen und dort gespeichert.

4.2.1 Architektur und Integration

Die Abbildung 4.5 zeigt den Aufbau des Yappi IntelliJ IDEA Companion.

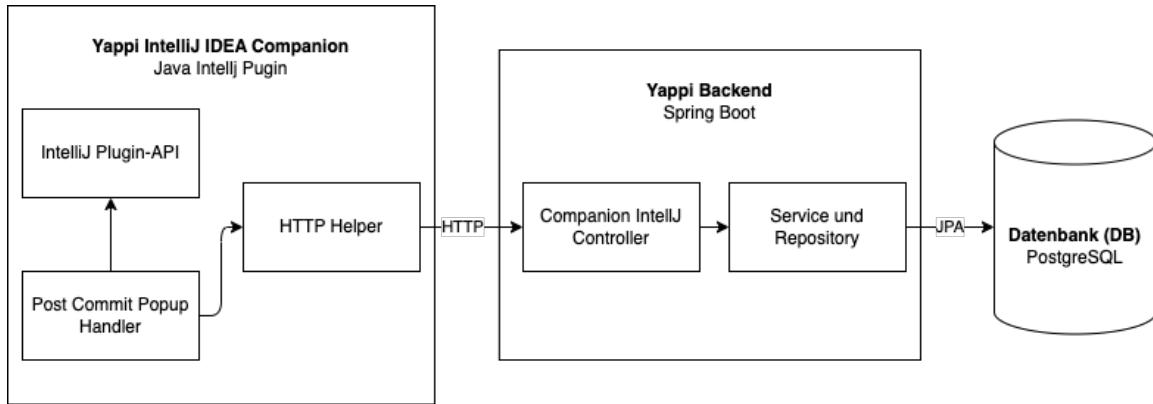


Abbildung 4.5: Architektur des IntelliJ IDEA Companion

Die Implementierung basiert auf den IntelliJ-Plugin-APIs. Ein (`PostCommitPopupHandler`)

registriert sich auf erfolgreiche Commit-Events. Nach Bestätigung des Commits wird der Post-Commit Dialog angezeigt. Der API-Key, der zur Authentifizierung gegenüber Yappi benötigt wird, wird über die Plugin-Einstellungen eingegeben und in einer persistenten Komponente (`CompanionStorage`) gespeichert. Die Kommunikation mit dem Backend erfolgt über eine dedizierte Hilfsklasse (`HttpHelper`), welche die Daten an den entsprechenden Endpoint überträgt.

4.2.2 Speicherung der Einstellungen

Damit sich die Companion App Authentifizieren kann muss ein gültiger API Key hinterlegt werden. Die Eingabe des API-Keys erfolgt über den Menüpunkt Settings → Tools → Yappi Companion in IntelliJ. Die Abbildung 4.6 zeigt den entsprechenden Menupunkt.

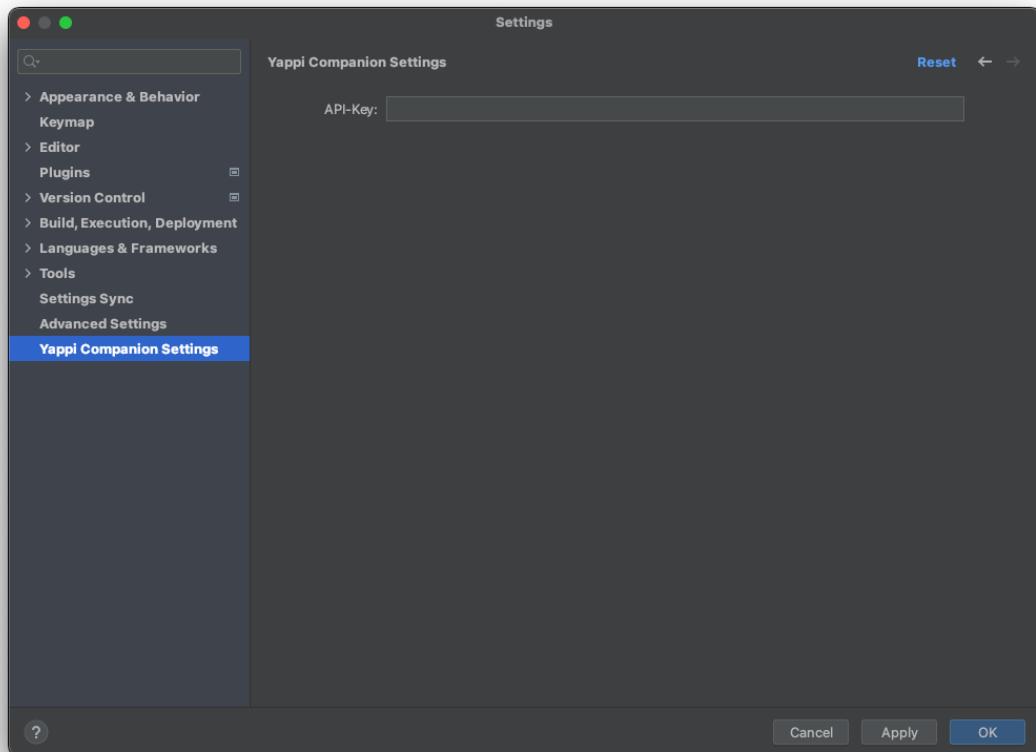


Abbildung 4.6: Eingabe des API-Keys in den IntelliJ Companion-Einstellungen

Der Key wird in der Klasse `CompanionStorage` mithilfe der IntelliJ-Persistenzmechanismen gespeichert und beim nächsten Start automatisch geladen.

4.2.3 Post-Commit Dialog

Das Post-Commit-Popup wird automatisch unmittelbar nach einem erfolgreichen Commit in IntelliJ IDEA angezeigt. Ziel ist es, die Erfassung von Stimmungsdaten so nahtlos wie möglich in den normalen Entwicklungs- und Versionskontrollprozess einzubinden. Der Dialog öffnet sich als eigenständiges Fenster und blockiert die IDE nicht, sodass Entwickler sofort weiterarbeiten können. Die Abbildung /reffig:intellij-post-commit zeigt die Benutzeroberfläche.

TODO: neues design

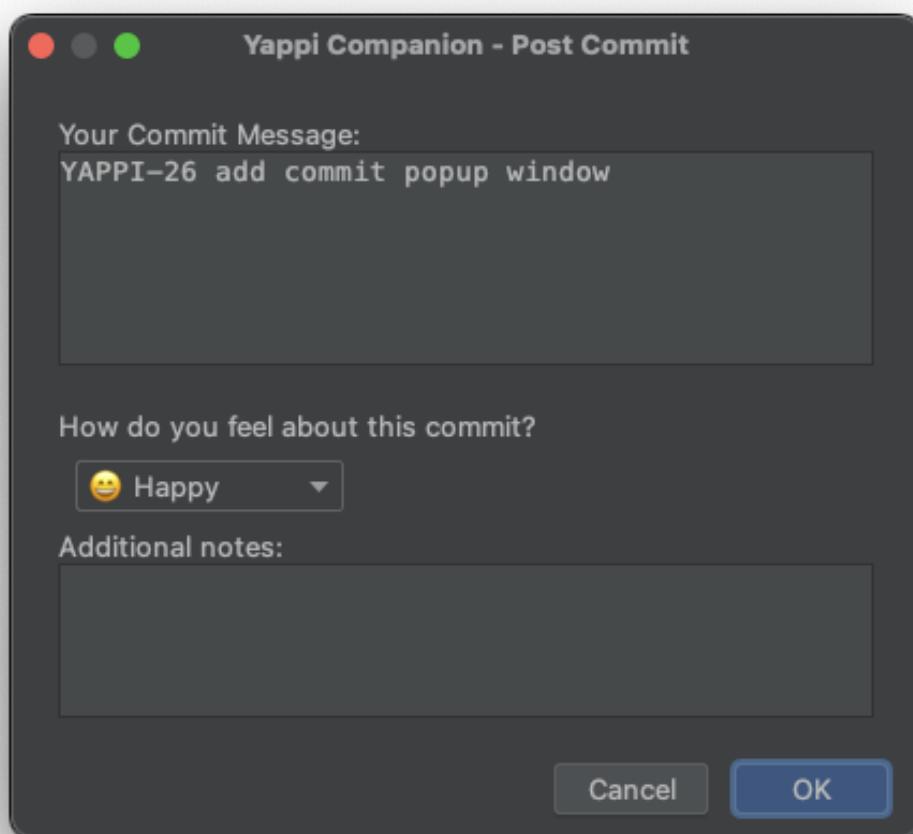


Abbildung 4.7: Post-Commit Dialog des IntelliJ Companions

Die Benutzeroberfläche umfasst zwei zentrale Elemente:

- **Commit-Nachricht:** Im oberen Bereich wird die soeben erfasste Commit-Nachricht

angezeigt.

- **Stimmungsauswahl:** Über ein Dropdown-Menü können vordefinierte Stimmungen ausgewählt werden, jeweils ergänzt durch ein Emoji zur schnellen Orientierung.

Nach Bestätigung des Dialogs werden die eingegebenen Daten an das Yappi-Backend gesendet. Dieser Vorgang läuft vollständig im Hintergrund ab, sodass der Entwickler direkt mit seiner Arbeit fortfahren kann.

4.2.4 Daten verarbeiten und Persistieren

Die vom IntelliJ Companion erfassten Commit-Daten werden über eine gesicherte Verbindung an das Yappi-Backend übertragen. Dort erfolgt zunächst die Authentifizierung anhand des API-Keys, der zuvor in den Plugin-Einstellungen hinterlegt wurde. Anschliessend werden die empfangenen Daten verarbeitet und persistiert.

Backend-Komponente

Für die Verarbeitung wurde eine eigene Backend-Komponente entwickelt, bestehend aus einem neuen Controller, einem Service und einem Repository für den Datenbankzugriff.

Der Controller `IntelliJCompanionController` stellt den Endpoint, über den die Zufriedenheitsdaten und commit-spezifischen Kontextinformationen entgegengenommen werden, bereit. Der Endpoint ist volgndermassen aufgebaut.

`POST /companion/commit`

Erfasst Zufriedenheitsdaten inklusive Kontextinformationen nach einem Commit.

Antworten: 200 OK, 400 Bad Request, 401 Unauthorized.

Die Speicherung erfolgt über ein Repository, das direkt auf die Datenbank zugreift.

Datenbankerweiterung

Die vom IntelliJ Companion übermittelten Zufriedenheitsdaten werden im Backend zusammen mit den zusätzlichen Kontextinformationenpersistiert. Während die eigentlichen Stimmungsdaten in der bestehenden Tabelle für Zufriedenheitsmessungen gespeichert werden, wird für die Commit-bezogenen Zusatzinformationen eine separate, flexible Kontexttabelle angelegt. Diese Trennung erlaubt es, in Zukunft weitere Arten von Kontextdaten ohne Änderungen Datenbankschema der Zufriedenheitsdaten zu integrieren.

Die Tabelle 4.2 zeigt den Aufbau der neuen Kontexttabelle `commit_context`.

Spalte	Datentyp	Beschreibung
<code>id</code>	SERIAL	Primärschlüssel, auto-inkrementierend
<code>satisfaction_id</code>	INTEGER	Fremdschlüssel auf <code>satisfaction_data.id</code> , verknüpft die Kontextdaten mit einer Zufriedenheitsmessung
<code>context_type</code>	TEXT	Die Art von Kontextinformation die gespeichert wird (z.B. Commit-Message oder Number of Lines Changed)
<code>value</code>	TEXT	Der Wert welcher gespeichert wird. Z.B. die Commit-Nachricht, wie sie in IntelliJ beim Commit eingegeben wurde
<code>created_at</code>	TIMESTAMPTZ	Zeitpunkt, zu dem der Datensatz erstellt wurde

Tabelle 4.2: Schema der Tabelle `commit_context`

Die Speicherung in einer separaten Tabelle stellt sicher, dass die Kernlogik der Zufriedenheitsmessungen unverändert bleibt, während gleichzeitig ein flexibles Datenmodell für kontextbezogene Informationen zur Verfügung steht. Dadurch lässt sich der Funktionsumfang von Yappi künftig unkompliziert erweitern, ohne bestehende Datenstrukturen zu beeinträchtigen.

4.3 Calendar Companion

Der *Calendar Companion* erweitert die Yappi-Plattform um eine Kalenderintegration mit Echtzeitbenachrichtigung und integriertem Feedbackprozess. Er kombiniert Backend-Services, eine Browsererweiterung sowie einen Kommunikationsbroker, um Kalendereinträge zu synchronisieren und direktes Feedback zu ermöglichen.

Zur automatischen Erkennung relevanter Meetings wird eine Anbindung an das Kalendersysteme des Entwicklers vorgesehen. Wichtig dabei ist es einen Offenen Standard zu verwenden um eine möglichst hohe Kompatibilität mit Kalendersystemen zu schaffen. Grundlage dazu bildet das weit verbreitete iCalendar-Format (ICS), das von nahezu allen gängigen Plattformen unterstützt wird. Das Konzept sieht vor, dass der Nutzer in Yappi auf seinem Profil den zugriff auf den Kalender hinterlegen kann. Regelmässig werden die auf einen vom Nutzer im Kalender erstellten Kalendereinträge in die Datenbank synchronisiert. Auf dieser Basis kann die Anwendung automatisch identifizieren, wann ein Meeting stattfindet. Dadurch lassen sich Feedback-Anfragen unmittelbar nach dem Ende einer Besprechung auslösen, ohne dass der Nutzer manuell eingreifen muss.

Die Synchronisation des Kalender erfolgt ausschliesslich lesend, um Eingriffe in persönliche

Kalender zu vermeiden. Alle übermittelten Kalenderinformationen werden vertraulich behandelt und ausschliesslich für den definierten Zweck genutzt. Informationen zu Kalendereinträge sind nur für den Nutzer persönlich einsehbar.

Durch diese Anbindung entfällt das manuelle Erfassung von Kalendereinträgen. Meetings werden automatisch erkannt, Änderungen übernommen und die Nutzer benachrichtigt. So entsteht ein nahtloser integrierter feedback flow um die Zufriedenheitsdaten von Meetings für die Entwickler zu erheben.

Dieses Kapitel beschreibt die technische Implementierung der einzelnen Komponenten, deren Zusammenspiel sowie die zugrunde liegenden Datenstrukturen.

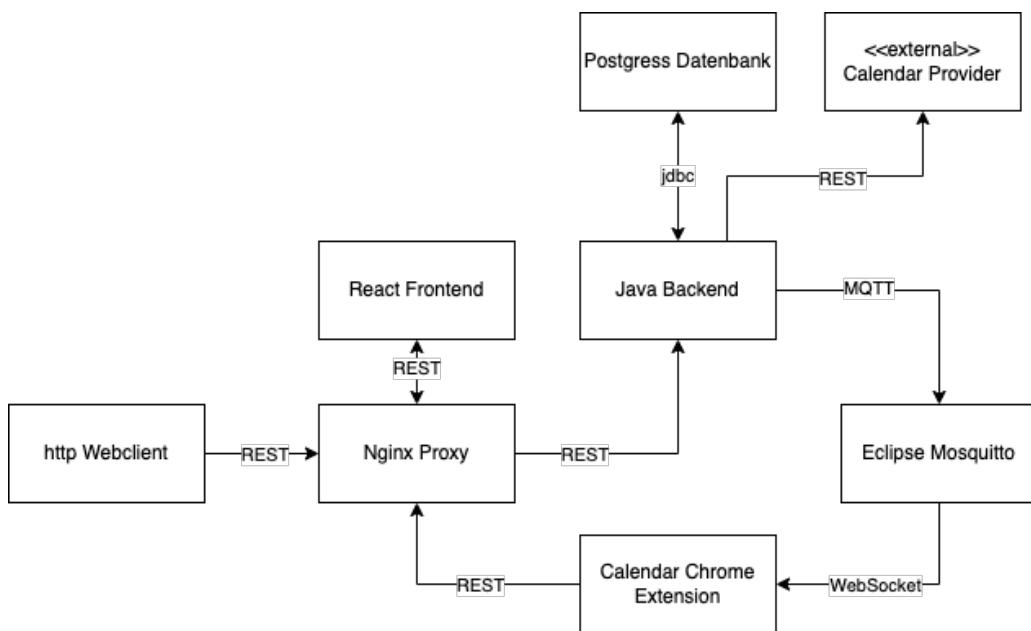


Abbildung 4.8: Systemübersicht zur Kalenderintegration in Yappi

Die Abbildung 4.8 zeigt eine Systemübersicht der Kalenderintegration.

4.3.1 Yappi Chrome Extension

Die Companion-App wird als Browsererweiterung umgesetzt, um eine direkte Interaktion mit den Entwicklern zum richtigen Zeitpunkt zu ermöglichen. Nach dem Ende eines im Kalender erfassten Meetings erhält der Nutzer eine Benachrichtigung, die zur Abgabe einer kurzen Bewertung auffordert. Diese zeitnahe Erhebung stellt sicher, dass Eindrücke und Wahrnehmungen noch frisch sind und die Datenqualität hoch bleibt. Die Interaktion erfolgt freiwillig, um Befragungsmüdigkeit zu vermeiden, und ist so gestaltet, dass die Beantwortung wenige Sekunden benötigt. Durch die Integration in den Browser wird kein zusätzlicher Systemwechsel nötig, wodurch die Hemmschwelle zur Teilnahme sinkt.

Architektur

Die Anwendung besteht aus zwei Hauptkomponenten:

1. **Persistenter Background-Prozess** Läuft dauerhaft im Hintergrund und ist für die Kommunikation mit dem Eclipse Mosquitto Nachrichtenbroker, die Verwaltung des Zustands und das proaktive Benachrichtigen des Nutzers zuständig.
2. **Temporäre Popup-Benutzeroberfläche (UI)** Wird nur bei aktiver Benutzerinteraktion geladen und dient der Darstellung der Benutzeroberfläche sowie dem Erfassen von Feedback.

Komponenten im Detail

Background-Prozess (`background/background.js`)

- Aufbau und Aufrechterhaltung einer Websockets-Verbindung (über `lib/mqtt.js`) zum Yappi Kommunikationsbroker.
- Abonnieren der benutzerspezifischen Topics zur Echtzeitbenachrichtigung (`companion/event/calender`).
- Verarbeitung eingehender Nachrichten und Erzeugung nativer Desktop-Benachrichtigungen (`chrome.notifications`).
- Persistente Speicherung der letzten zehn Ereignisse in `chrome.storage.local` nach FIFO-Prinzip.

Popup-Benutzeroberfläche (`popup/home.html`)

- Dynamischer Umfrage-Builder (`popup/survey/builder.js`): Ruft aktuelle Umfragen über die REST-API ab, generiert DOM-Elemente aus JSON Vorlagen (`popup/questionblock.html` und `popup/questionblock.js`) und integriert diese in die Ansicht. Mehr Details dazu in Abschnitt ??.
- Feedback-Übermittlung (`popup/popup.js`): Sammelt Antworten, strukturiert sie als JSON und sendet sie per POST-Request an das Yappi-Backend.

Benutzeroberfläche (UI)

Die Benutzeroberfläche der Yappi Chrome Extension besteht aus zwei zentralen Interaktionselementen: Systembenachrichtigungen und der Popup-Ansicht im Browser.

Systembenachrichtigungen

Beim Ende eines Events im Kalender des Benutzer, erzeugt die Extension eine native Systembenachrichtigung siehe Abbildung 4.9. Diese Benachrichtigungen enthalten:

- Titel und Beschreibung des Meetings (z. B. „Meeting IP5 - Standup“),
- Uhrzeit des Termins,
- das Yappi-Logo als Icon,
- interaktive Schaltflächen wie `Review Event` zur direkten Abgabe von Feedback.

Durch diese direkte Interaktionsmöglichkeit kann Feedback ohne Umweg über die Hauptanwendung erfasst werden.

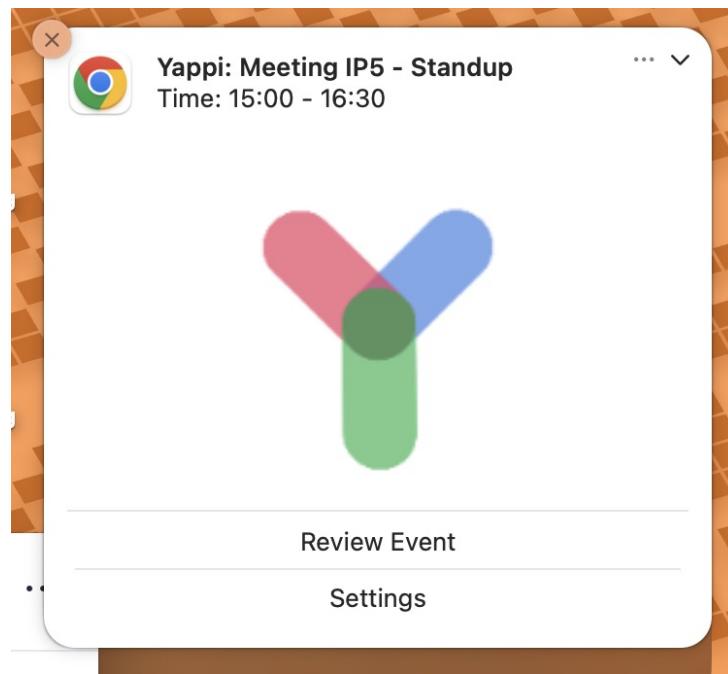


Abbildung 4.9: Systembenachrichtigung der Chrome Erweiterung in macOS

Popup-Ansicht

Das Browser-Popup siehe Abbildung 4.10 dient als Haupt-UI der Extension und wird durch einen Klick auf das Yappi-Icon in der Browser-Toolbar geöffnet. Die Ansicht enthält:

- Verbindungsstatus zum Nachrichtenbroker,
- Benutzerhinweis zum erfolgreichen Setup,
- eine Liste kürzlich abgeschlossener Meetings zu welchen noch kein Feedback gegeben wurde,
- Aktionsbuttons für jedes Meeting:

- **Review** Öffnet die zugehörige Feedback-Umfrage.
- **Delete** Entfernt den Eintrag aus der Liste der letzten Meetings.

Das Design stellt sicher, dass Benutzer sowohl über Echtzeitbenachrichtigungen als auch über die Popup-Ansicht jederzeit auf anstehende Feedback-Aufgaben zugreifen können.

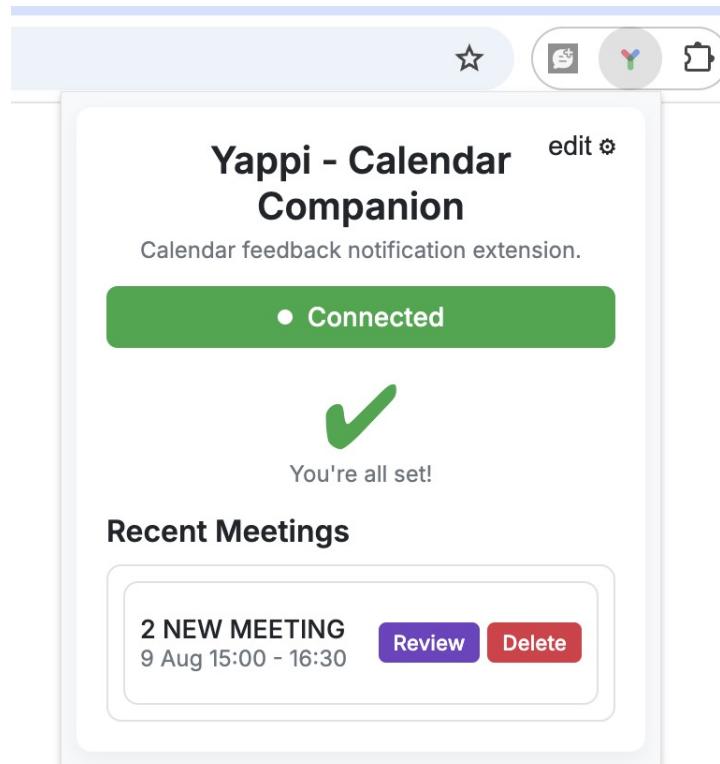


Abbildung 4.10: Benutzeroberfläche der Chrome Erweiterung

Kommunikationskanäle



Abbildung 4.11: Ablauf nachdem ein Kalender-Event abgeschlossen wurde.

Die Verbindung zum Yappi-Backend erfolgt über zwei Kommunikationskanäle:

- **REST-API** Für synchronen Abruf von Daten und Übermittlung von Antworten.
- **MQTT-Broker** Für asynchrone Benachrichtigungen in Echtzeit.

Abbildung 4.11 zeigt wie Kommunikation abläuft nach beenden eines Kalender-Event.

Die interne Kommunikation zwischen Background-Prozess und Popup-UI nutzt die Browser-API `chrome.storage`. Dort werden alle Informationen, wie Clientdaten oder die letzte Kalenderevents gespeichert und ausgetauscht.

Externe Abhängigkeiten

- **Yappi-Backend** Bereitstellung der REST-API und MQTT-Broker.
- **MQTT.js (lib/mqtt.js)** JavaScript-Clientbibliothek für MQTT-Kommunikation.

4.3.2 Backend-Integration

Das Yappi-Backend fungiert als Zentraler Controller der gesamten Kalenderintegration von Yappi. Es ist zuständig für jeglichen Datenaustausch, Datenspeicherung sowie Datenverarbeitung und besteht aus verschiedenen modulen welche alle ihre verantwortlichkeiten haben.

Für den Calendar Companion wurde ein eigenes Modul erstellt.

Das Kalendermodul im Yappi-Backend ist für die Verwaltung und Bereitstellung von Kalendereinträgen verantwortlich. Es unterstützt sowohl den Abruf von Einträgen über eine REST-API als auch eine proaktive Benachrichtigung der Chrome-Extension in Echtzeit. Letzteres erfolgt über das Publish-Subscribe-Protokoll *MQTT* (Message Queuing Telemetry Transport). Zudem ist es Zuständig für das Laden und synchronisieren der externen iCalendar-Daten (ICS) welche von den Nutzer hinterlegt werden.

Model-Klasse

Die zentrale Datenstruktur bildet die JPA-Entität `CalendarEvent.java (entities/)`. Sie repräsentiert einen einzelnen Kalendereintrag mit Attributen wie `title`, `startDate`, `endDate`, `allDay` sowie der Zuordnung zu einem Nutzer (`userId`). Diese Klasse wird innerhalb des Backends zwischen Services und Controllern verwendet.

API-Endpunkte (`CalendarController`)

Die Interaktion externer Clients mit dem Kalendermodul erfolgt über folgende REST-Endpunkte:

- `GET /api/calendar/{userId}` Ruft alle Kalendereinträge einer bestimmten Nutzer-ID ab. Der Controller delegiert die Anfrage an den `CalendarService`, der die Daten lädt und als JSON zurückgibt.

- POST `/api/calendar-sync` Stößt die Synchronisation manuell an, z.B. für sofortige Aktualisierungen oder Debugging. Nutzt dieselbe Logik wie die automatische Synchronisation durch den `CalendarEventScheduler`.

Service-Schicht (`CalendarService`)

Der `CalendarService` enthält die Geschäftslogik und entkoppelt Controller und geplante Aufgaben von der Datenverarbeitung. Er übernimmt:

- Abruf von Kalendereinträgen,
- Erstellung und Aktualisierung von Einträgen durch Abgleich mit externen Daten,
- Benachrichtigung über MQTT nach dem Ende von Terminen.

Geplante Aufgaben (`CalendarEventScheduler`)

Zwei periodische Jobs sorgen für Aktualität und Benachrichtigungen:

1. **Prüfung beendeter Events** (alle 60 Sekunden): Erkennt neu abgeschlossene Kalendereinträge und benachrichtigt betroffene Clients über MQTT.
2. **Kalendersynchronisation** (alle 5 Minuten): Aktualisiert Kalendereinträge auf Basis externer ICS-Daten. Dabei werden neue Events erstellt, geänderte aktualisiert und nicht mehr vorhandene gelöscht. Änderungen lösen eine Update-Benachrichtigung an die Clients aus.

Echtzeit-Benachrichtigungen (`MqttService`)

Nach Änderungen publiziert der `MqttService` Nachrichten auf themenspezifischen Topics (z.B. `deardev/teams/{teamId}/calendar`). Clients abonnieren diese Topics und können ihre UI unmittelbar aktualisieren, ohne Polling durchführen zu müssen.

4.3.3 Datenbankerweiterung

Zur Speicherung der Kalendereinträge wurde die neue Tabelle `calendar_events` eingeführt. Die Struktur wird durch die JPA-Entität `CalendarEvent.java` definiert.

Backend-Komponenten

- **Entität** (`CalendarEvent.java`): Datenmodell und Tabellenstruktur.
- **Repository** (`CalendarEventRepository.java`): CRUD-Operationen auf `calendar_events`.

Spalte	Datentyp	Beschreibung
<code>id</code>	INTEGER	Primärschlüssel des Eintrags
<code>title</code>	VARCHAR	Titel des Kalenderevents (z. B. „Sprint 1“)
<code>start_date</code>	TIMESTAMP	Startdatum und -zeit
<code>end_date</code>	TIMESTAMP	Enddatum und -zeit
<code>all_day</code>	BOOLEAN	Kennzeichen für ganztägige Events
<code>user_id</code>	INTEGER	Fremdschlüssel zur Tabelle <code>users</code>
<code>team_id</code>	INTEGER	Fremdschlüssel zur Tabelle <code>teams</code>

Tabelle 4.3: Schema der Tabelle `calendar_events`

- **Service** (`CalendarService.java`): Geschäftslogik zum Erstellen, Abrufen und Löschen von Events.
- **Controller** (`CalendarController.java`): REST-Endpunkte zum Abrufen von Events pro Nutzer.
- **Scheduler** (`CalendarEventScheduler.java`): Zeitgesteuerte Erstellung und Aktualisierung der Events.
- **Factory** (`CalendarEventFactory.java`): Hilfsklasse zur einfachen Erstellung neuer Eventobjekte.

Automatisierte Erstellung

Der `CalendarEventScheduler` generiert automatisch Einträge auf Basis aktiver Sprints:

1. Abruf aller aktiven Sprints für jedes Team,
2. Prüfung, ob für jedes Teammitglied bereits ein Event existiert,
3. Erstellung neuer Events mittels `CalendarEventFactory`, falls erforderlich,
4. Speicherung der Events über das `CalendarEventRepository`.

4.3.4 Kommunikationsbroker

Für die Echtzeitübertragung von Benachrichtigungen an Clients wird der Open-Source-MQTT-Broker **Eclipse Mosquitto** eingesetzt. *MQTT* (Message Queuing Telemetry Transport) ist ein leichtgewichtiges Publish-Subscribe-Protokoll, das sich besonders für Anwendungen mit geringer Latenz und hohem Aktualisierungsbedarf eignet.

Zweck

Der Kommunikationsbroker ermöglicht die sofortige Benachrichtigung von Companion-Anwendungen, insbesondere der Google-Chrome-Erweiterung des Calendar Companions. Damit entfällt die Notwendigkeit von Polling-Anfragen an das Backend, was die Netzwerklast reduziert und eine reaktive Benutzeroberfläche ermöglicht.

Kommunikationsfluss

1. Das Backend erkennt eine relevante Änderung, z. B. das Ende eines Meetings oder die Erstellung eines neuen Kalendereintrags.
2. Der `MqttService` des Backends verbindet sich mit dem Mosquitto-Broker und publiziert die Nachricht auf einem benutzerspezifischen Topic: `calendar/event/{user-api-key}`.
3. Die Chrome-Erweiterung des Calendar Companions ist als MQTT-Client auf diesem Topic angemeldet (*subscribed*).
4. Beim Empfang einer Nachricht aktualisiert der Client die lokale Ansicht und kann bei Bedarf Folgeaktionen anstoßen (z. B. Anzeige eines Feedback-Dialogs).

Nachrichtenformat

Die vom Backend publizierten Benachrichtigungen werden als JSON-Objekte gesendet. Eine typische Nachricht enthält die folgenden Felder:

```
1  {
2      "eventId": 3002,
3      "userId": 1,
4      "title": "Meeting IP5 - Standup",
5      "startDate": "2025-08-09T15:00",
6      "endDate": "2025-08-09T15:15",
7      "location": "",
8      "questionBlockId": 1,
9      "teams": [
10         {
11             "id": 1,
12             "name": "Test Team 1"
13         },
14         {
15             "id": 3,
16             "name": "Test Team 2"
17         }
18     ]
19 }
```

Vorteile

- Sofortige Zustellung ohne Polling.
- Geringe Netzwerklast durch Publish-Subscribe-Architektur.
- Einfaches Routing über benutzerspezifische Topics.
- Erweiterbar für weitere Eventtypen oder Companion-Anwendungen.

4.3.5 Dynamische Umfragen

Die dynamischen Umfragen in Yappi basieren auf flexibel konfigurierbaren Fragenblöcken, die als JSON-Objekte in der Datenbank gespeichert werden. Diese Struktur ermöglicht es, Umfragen anzupassen oder zu erweitern, ohne Änderungen am Anwendungscode vorzunehmen.

Datenbank

Die Tabelle `questionblocks` speichert die Konfiguration einzelner Fragenblöcke.

Spalte	Datentyp	Beschreibung
<code>id</code>	SERIAL	Eindeutiger, automatisch inkrementierender Primärschlüssel
<code>configuration</code>	JSONB	Vollständige Konfiguration des Fragenblocks als JSON-Objekt. Ermöglicht flexible Strukturierung und effiziente Abfragen

Tabelle 4.4: Schema der Tabelle `questionblocks`

Anmerkung: Der Datentyp `JSONB` (Binary JSON) erlaubt eine performante Speicherung und gezielte Abfragen innerhalb der JSON-Daten.

Backend-Komponenten

- **Entität** (`QuestionBlock.java`) - Abbildung der Tabelle `questionblocks` als JPA-Entität. Attribute: `id` (INTEGER) und `configuration` (`JsonNode`).
- **Repository** (`QuestionBlockRepository.java`) - CRUD-Operationen auf der Tabelle.
- **Service** (`QuestionBlockService.java`) - Geschäftslogik für den Zugriff und die Validierung von Fragenblöcken.
- **Controller** (`QuestionBlockController.java`) - Bereitstellung der API-Endpunkte.

API-Endpunkte

- GET /companion/questionblocks/{id} Ruft den Fragenblock mit der angegebenen ID ab. **Antworten:** 200 OK mit JSON-Objekt oder 404 Not Found, falls nicht vorhanden.

Darstellung in der Companion-App

Die Chrome Extension ruft den entsprechenden Fragenblock per REST-API ab. Das Skript `popup/survey/builder.js` parst die JSON-Daten und erstellt für jede Frage die passenden HTML-Elemente basierend auf den Templates (`popup/questionblock.html`) und der Logik in `popup/questionblock.js`. Unterstützt werden Fragetypen wie:

- Likert-Skalen
- Single-Choice
- Multiple-Choice
- Freitext

Beispiel Fragebogen

Das UI des Fragebogen ist sehr einfach gehalten. In der Abbildung 4.12 ist eine Grafische-Darstellung einer Beispielumfrage zu sehen.

Meeting Feedback

2 NEW MEETING

Time: 09/08/2025, 15:00:00 - 09/08/2025, 16:30:00

Teams: Test123, Xeno's cooles Team

Select Team: ✓

Duration:

Meeting Feedback (Extended)

A concise survey to assess goal clarity, content quality, time management, facilitation, outcomes, satisfaction, and meeting length.

1. Clarity of Goals & Relevance

The goals of the meeting were clearly defined and relevant to my work.

strongly disagree – strongly agree

1
 2
 3
 4
 5

2. Content Depth & Understandability

Topics were covered with appropriate depth and remained understandable (no unnecessary detail).

strongly disagree – strongly agree

1
 2
 3
 4
 5

3. Time Management

The meeting started on time and stayed within the scheduled duration.

strongly disagree – strongly agree

1
 2
 3
 4
 5

4. Facilitation & Participation

The facilitator guided the meeting effectively and encouraged active participation.

strongly disagree – strongly agree

1
 2
 3
 4
 5

Abbildung 4.12: Feedbackformular zu Kalenderevent

Die Konfiguration des Fragebogen kann aus folgender JSON Struktur abgeleitet werden.

```
1  {
2      "id": "1000",
3      "title": "Basic Meeting Feedback",
4      "description": "This is am Meeting feedback survey.",
5      "sections": [
6          {
7              "id": "q1",
8              "type": "likert",
9              "title": "1. Clarity of Goals & Relevance",
10             "prompt": "The goals of the meeting were clearly defined.",
11             "required": true,
12             "min": 1,
13             "label_min": "strongly disagree",
14             "max": 5,
15             "label_max": "strongly agree",
16             "step": 1,
17             "default": 3
18         },
19         {
20             "id": "q2",
21             "type": "single-choice",
22             "title": "2. Meeting Length",
23             "prompt": "How would you rate the length of the meeting?",
24             "required": true,
25             "options": [
26                 { "value": "too-short", "label": "Too short" },
27                 { "value": "just-right", "label": "Just right" },
28                 { "value": "too-long", "label": "Too long" }
29             ],
30             "default": "just-right"
31         },
32         {
33             "id": "q3",
34             "type": "multiple-choice",
35             "title": "3. Meeting Highlights",
36             "prompt": "Which aspects of the meeting did you find valuable? (Select → all that apply)",
37             "required": false,
38             "options": [
39                 { "value": "clear-agenda", "label": "Clear agenda" },
40                 { "value": "useful-content", "label": "Useful content" },
41                 { "value": "good-discussion", "label": "Good discussion" },
42                 { "value": "action-items", "label": "Clear action items" }
43             ],
44             "default": ["clear-agenda", "good-discussion"]
45         },
46         {
47             "id": "q5",
```

```

48     "type": "free-text",
49     "title": "4. Additional Comments",
50     "prompt": "Please share any additional feedback or suggestions.",
51     "required": false,
52     "placeholder": "Type your feedback here...",
53     "maxLength": 500
54   }
55 ]
56 }
```

Vorteile

- Anpassung und Erweiterung von Umfragen ohne Codeänderungen.
- Unterstützung verschiedener Fragetypen und Layouts.
- Zentrale Verwaltung und Wiederverwendung von Fragenblöcken.

4.4 Health Companion

Gesundheitsdaten können wertvolle Zusatzinformationen zur Entwicklerzufriedenheit liefern. Sie ergänzen subjektive Stimmungsangaben um objektive Messwerte, die Rückschlüsse auf Belastung und Erholung ermöglichen. Durch die Anbindung an etablierte Schnittstellen wie Apple Health Kit oder Garmin Health API lassen sich Metriken wie Schlafqualität, Herzfrequenz oder Stressindikatoren automatisiert erfassen. Diese Daten bilden zusammen mit den Prozess- und Zufriedenheitsmetriken eine erweiterte Grundlage, um Korrelationen zwischen physischem Wohlbefinden und Arbeitszufriedenheit zu erkennen und gezielte Massnahmen abzuleiten.

4.4.1 Quellen und Schnittstellen für Gesundheitsdaten

Für die Integration von Gesundheitsdaten wurden mehrere Optionen geprüft. Im Mittelpunkt standen die Garmin Health API sowie Apple Health Kit. Beide Quellen decken relevante Metriken wie Schritte, Herzfrequenz, Schlaf und Stress/HRV ab, unterscheiden sich jedoch deutlich hinsichtlich Zugangsmodell, Datenschutzmechanismen, Integrationsaufwand und Eignung für nicht-kommerzielle Forschungsprojekte.

Garmin Health API Die Garmin Health API stellt über eine REST-Schnittstelle umfangreiche Tages- und Aktivitätsmetriken bereit. Die Daten werden in der Regel nach dem Gerätesync via über die Mobile Applikation von Garmin bereitgestellt und in JSON ausgeliefert. die Plattform unterstützt Pull- und Push-Modelle, sodass

auch eine ereignisgetriebene Integration möglich ist. Der Zugang ist für genehmigte Business-Developer vorgesehen. Für die kommerzielle Nutzung fallen in der Regel Lizenzgebühren an. Diese geschäftliche Ausrichtung sowie die erforderliche formale Zulassung machen die direkte Nutzung für ein kleines, nicht-kommerzielles Forschungsprojekt unpassend. Zwar existieren Forschungsprogramme und Partnerlösungen rund um Garmin, diese sind jedoch üblicherweise an formelle Kooperationen, Vertragsaufwände und teils Kosten gebunden, was die Umsetzungshürde erhöht [19].

Apple Health Kit Health Kit dient auf iOS als zentrales, lokales Datenrepository für Gesundheits- und Fitnessdaten von iPhone, Apple Watch und angebundenen Geräten/Apps. Der Zugriff erfolgt feingranular pro Datentyp und erfordert stets explizite Nutzerzustimmung (Lesen/Schreiben getrennt). Daten liegen verschlüsselt auf dem Gerät. Apps erhalten nur Zugriff auf die explizit freigegebenen Kategorien. Für die Integration stehen dokumentierte APIs und Query-Muster zur Verfügung. Eine serverseitige Drittplattform ist nicht erforderlich, wodurch Architektur, Betrieb und Datenschutzfolgenabschätzung vereinfacht werden [20].

Da die Garmin Health API primär für kommerzielle Anwendungen vorgesehen ist und der beantragte Zugang für dieses Projekt nicht gewährt wurde, wurde Apple Health Kit als geeignete Datenquelle ausgewählt. Für die Umsetzung bedeutet dies, dass eine iOS-Anwendung entwickelt wird, die die relevanten Daten aus Apple HealthKit ausliest und an Yappi überträgt.

4.4.2 Daten aus HealthKit auslesen

Um relevante Gesundheitsmetriken automatisiert zu erfassen, nutzt die iOS-Anwendung die von Apple bereitgestellte HealthKit-Schnittstelle. In diesem Kapitel wird der Zugriff auf die Schrittzählungsdaten erklärt, die anderen Metriken werden auf die gleiche Art und Weise implementiert.

Der Zugriff auf HealthKit-Daten erfordert die explizite Zustimmung des Nutzers für jede einzelne Datenkategorie. Beim ersten Start der App wird daher eine Berechtigungsanfrage angezeigt, in der der Zugriff auf Schrittzählungsdaten (`HKQuantityTypeIdentifier.stepCount`) angefordert wird. Erst nach Bestätigung durch den Nutzer beginnt die App mit der Erfassung.

Die Umsetzung der Datenerfassung erfolgt über eine Kombination aus einem `HKObserverQuery` und zeitlich begrenzten `HKStatisticsQuery`-Abfragen. Der `HKObserverQuery` registriert sich auf Änderungen in den beobachteten Metriken und wird von HealthKit automatisch ausgelöst, sobald neue Daten vorliegen. Für jede Verarbeitung wird der zuletzt verarbeitete

Zeitraum gespeichert, sodass bei der nächsten Abfrage nahtlos an dieser Stelle fortgesetzt werden kann, ohne bereits erfasste Daten erneut zu verarbeiten.

Die gesammelten Daten werden anschliessend über einen API-Endpoint an das Backend der Yappi-Plattform übertragen und dort gespeichert. Dadurch stehen sie zentral zur Verfügung und können gemeinsam mit anderen erfassten Metriken ausgewertet werden.

4.4.3 Daten an Yappi übertragen

Um die Verarbeitung auf Serverseite zu vereinfachen und die Erweiterbarkeit auf weitere Metriken zu ermöglichen, werden die Daten in einer einheitlichen JSON-Struktur übertragen. Diese enthält zum einen den Typ der Metrik (`metric`), zum anderen ein Array von Messwerten, das für jeden Eintrag den erfassten Zeitraum sowie den zugehörigen Wert enthält. Die Zeiträume werden im Format `yyyy-MM-dd HH:mm/yyyy-MM-dd HH:mm` angegeben, um sowohl das Start- als auch das Enddatum einer Messung eindeutig zu definieren.

```
1 {
2     "metric": "STEPS",
3     "data": [
4         {
5             "timeRange": "2025-08-06 08:00/2025-08-06 09:00",
6             "value": 38
7         }
8     ]
9 }
```

Diese Struktur erlaubt eine konsistente Verarbeitung unabhängig von der Art der Metrik und bildet die Grundlage für die serverseitige Validierung, Speicherung und spätere Analyse. Sie kann ohne Anpassungen um weitere Metriken erweitert werden, indem lediglich der Wert von `metric` angepasst und das `data`-Array mit den entsprechenden Messpunkten befüllt wird.

Im folgenden Abschnitt werden die notwendigen Anpassungen im Backend und in der Datenbank erläutert, um die empfangenen Messdaten zu verarbeiten und dauerhaft zu speichern.

Backend-Komponenten

Für die Verarbeitung der eingehenden Gesundheitsdaten wurde im Backend ein Controller mit zugehörigem Service und Repository implementiert. Der Controller stellt einen REST-Endpoint bereit, über den die Companion-App die Daten im beschriebenen JSON-Format

an die Plattform übermitteln kann. Der folgende Endpoint wird von dem Controller bereitgestellt.

POST /companion/health

Nimmt Gesundheitsdaten im definierten JSON-Format entgegen und speichert sie für den authentisierten Nutzer. Bereits vorhandene Einträge im angegebenen Zeitraum werden aktualisiert, andernfalls werden neue Einträge angelegt.

Antworten: 200 OK, 400 Bad Request, 401 Unauthorized.

Die eingehenden Anfragen werden an den Service weitergeleitet, der die eigentliche Geschäftslogik übernimmt. Der Service prüft zunächst, ob für den übermittelten Benutzer und die angegebene Metrik im gesendeten Zeitraum bereits ein Eintrag in der Datenbank vorhanden ist. Falls ja, wird der bestehende Datensatz überschrieben, um stets den aktuellsten Stand zu gewährleisten. Falls kein Eintrag vorhanden ist, wird ein neuer Datensatz angelegt.

Das Repository kapselt den Zugriff auf die Datenbank und übernimmt das Einfügen sowie das Aktualisieren der Messwerte. Durch diese Aufteilung in Controller, Service und Repository wird eine klare Trennung von Zuständigkeiten erreicht, was die Wartbarkeit und Erweiterbarkeit der Implementierung erleichtert.

Datenbankerweiterung

Zur Speicherung der von der Companion-App übermittelten Gesundheitsdaten wurde das Datenbankschema um die Tabelle `health_metric` erweitert. Diese Tabelle dient als generische Ablage für unterschiedliche Metriken. Durch diese Struktur lassen sich weitere Gesundheitsmetriken ohne Schemaänderungen hinzufügen.

Spalte	Datentyp	Beschreibung
<code>id</code>	SERIAL	Primärschlüssel, auto-inkrementierend
<code>user_id</code>	INTEGER	Fremdschlüssel auf <code>users.id</code>
<code>metric</code>	TEXT	Typ der Metrik (z. B. STEPS, SLEEP)
<code>start_time</code>	TIMESTAMPTZ	Beginn des Messzeitraums
<code>end_time</code>	TIMESTAMPTZ	Ende des Messzeitraums
<code>value</code>	NUMERIC	Messwert der Metrik
<code>created_at</code>	TIMESTAMPTZ	Zeitpunkt der Erfassung in der Datenbank

Tabelle 4.5: Schema der Tabelle `health_metric`

4.4.4 Benutzeroberfläche der iOS App

Nachfolgend wird die Benutzeroberfläche der iOS App vorgestellt. Die Benutzeroberfläche der iOS-Anwendung ist bewusst minimalistisch gehaltet und darauf ausgelegt, den API-Key

einmalig zu hinterlegen, ohne dass danach weitere Eingaben erforderlich sind.

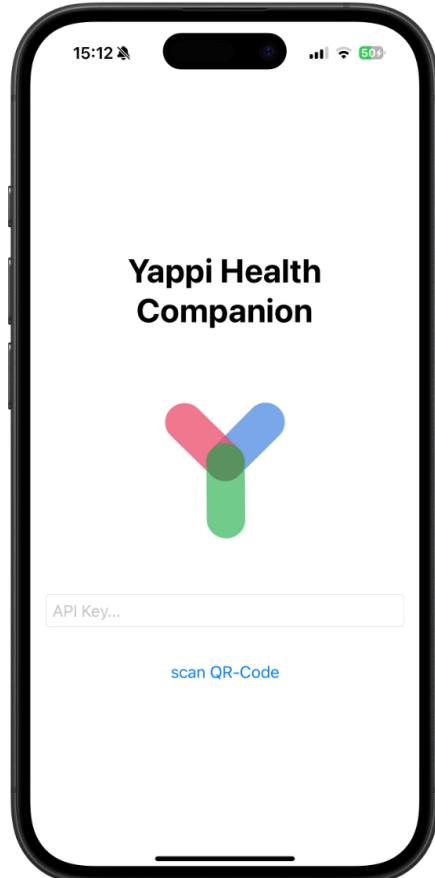


Abbildung 4.13: Benutzeroberfläche der iOS Health Companion App

Die Abbildung 4.13 zeigt die Benutzeroberfläche der App. Für die Eingabe des API-Keys stehen mehrere Wege zur Verfügung, entweder durch manuelle Eingabe oder durch das Scannen eines QR-Codes. Der integrierte QR-Code-Scanner öffnet sich im Vollbildmodus, erkennt den Schlüssel automatisch und trägt ihn direkt in das Eingabefeld ein. Nach der Eingabe wird der API-Key in den UserDefaults des Geräts gespeichert, sodass er bei späteren App-Starts automatisch geladen wird.

Die Verarbeitung der Gesundheitsdaten erfolgt vollständig im Hintergrund. Sobald neue Werte vorliegen, werden diese automatisch an Yappi übertragen. Dadurch entfällt für den Nutzer jede manuelle Auslösung und der Betrieb der App erfordert keine zusätzliche Interaktion.

4.5 Deployment und CI/CD

Das Deployment ist für dieses Projekt ein zentraler Schritt. Es dient der Vorbereitung auf die Durchführung eines User Testing, indem eine lauffähige Version von Yappi bereitgestellt wird. So kann gewährleistet werden, dass dieses Testing unter möglichst realitätsnahen Bedingungen stattfinden kann.

Zudem ermöglicht ein funktionierendes Deployment die Überprüfung der Zusammenarbeit der verschiedenen Companion Apps. Nur in einer produktiven Umgebung lässt sich sicherstellen, dass diese Komponenten korrekt miteinander interagieren und Daten wie vorgesehen austauschen.

Das Deployment der Yappi-Plattform erfolgt automatisiert mittels *Continuous Integration/Continuous Deployment* (CI/CD) über GitHub Actions. Die Ausführung erfolgt auf einem SwitchEngines-Cloud-Server mit Debian Linux. Das System ist vollständig Dockerisiert. Der externe Zugriff wird durch einen NGINX-Reverse-Proxy ermöglicht.

Ziel ist ein einfaches, schnelles und reproduzierbares Deployment direkt nach der Continuous-Integration-Phase. Der Build erzeugt versionierte, unveränderliche Docker-Images, die in ein Repository publiziert und auf Zielumgebungen mit Docker Compose oder Kubernetes ausgerollt werden kann. Die Konfiguration erfolgt über Umgebungsvariablen, Secrets werden zentral verwaltet. Durch die vollständige Containerisierung bleibt die Lösung host-agnostisch und kann mittels Infrastructure as Code (z. B. Terraform) und standardisierten Manifesten ohne Codeanpassungen auf neue Hosting-Systeme migriert werden.

4.5.1 Build- und Deployment-Pipeline

Für Backend und Frontend bestehen separate GitHub-Actions-Workflows. Beide nutzen einen self-hosted GitHub Runner, der direkt auf dem Zielserver installiert ist. Die Ausführung der Workflows erfolgt automatisch bei Pushes auf die Branches `develop`. Zusätzlich kann der Prozess manuell über `workflow_dispatch` gestartet werden.

Die Abbildung 4.14 zeigt die einzelnen Schritte des automatisierten Build- und Deployment-Prozesses für das Backend (vom Quellcode-Checkout bis zum Neustart der Container).

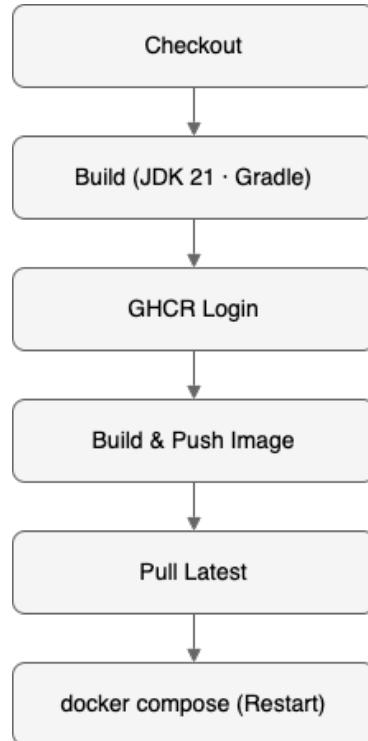


Abbildung 4.14: Deployment flow vom Backend

Analog dazu veranschaulicht Abbildung 4.15 den entsprechenden Ablauf für das Frontend, inklusive Linting, Next.js-Build und Bereitstellung des Docker-Images.

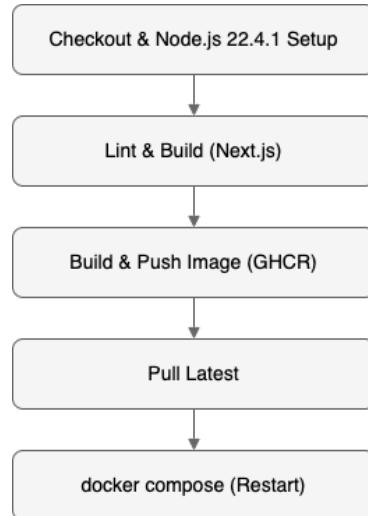


Abbildung 4.15: Deployment flow vom Frontend

4.5.2 Serverinfrastruktur

Der Betrieb erfolgt auf einem SwitchEngines-Cloud-Server mit 24/7-Verfügbarkeit. Als Betriebssystem wird Ubuntu 24.04 LTS (Codename Noble) eingesetzt. Die Serverressourcen sind für den produktiven Betrieb und die gleichzeitige Ausführung aller Dienste ausgelegt:

- **Arbeitsspeicher (RAM):** 32 GB
- **Virtuelle CPUs (vCPUs):** 8
- **Speicher:** 120 GB SSD
- **Netzwerk (Öffentlich):** IPv4-Adresse: 86.119.48.26

Auf dem Server laufen die folgenden Container-Services:

Beschriftung und Beschreibung zur Abbildung 4.16.

- **NGINX-Reverse-Proxy** - aggregiert Netzwerkanfragen
- **PostgreSQL-Datenbank** - persistente Speicherung der Applikationsdaten.
- **Backend-Service** - Spring-Boot-Anwendung, welche API- und MQTT-Funktionen bereitstellt.
- **Frontend-Service** - Next.js-Anwendung für die Benutzeroberfläche im Web.
- **Eclipse Mosquitto** - MQTT-Broker für die Echtzeitkommunikation mit Companion-Apps.

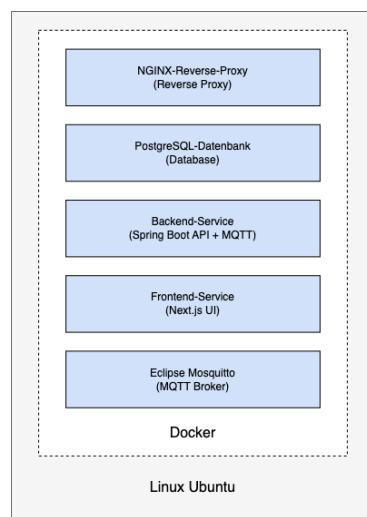


Abbildung 4.16: Systemübersicht der Linuxumgebung

4.5.3 NGINX-Proxykonfiguration

Der NGINX-Proxy übernimmt Routing und Zugriffskontrolle. Alle Anfragen werden per http/https direkt auf den Proxy gemacht. Dieser aggregiert dann automatisch zum gewünschten Endpunkte. Das Mapping der Endpunkte sieht wie folgt aus:

- `/v1/, /apikey/, /auth/` → Weiterleitung an das Backend.
- `/companion/` → Weiterleitung an das Backend mit speziellen CORS-Headern für API-Token Authentifizierung.
- `/` (Root) → Weiterleitung an das Frontend.

4.5.4 Docker-Compose-Struktur

Die Infrastruktur ist in zwei `docker-compose.yml`-Dateien organisiert:

- **Backend-Stack:** NGINX, PostgreSQL, Backend-Service, Mosquitto-Broker.
- **Frontend-Stack:** Separater Container, über das gemeinsame Netzwerk `yappi_network` mit dem Backend verbunden.

4.5.5 MQTT-Broker-Konfiguration

Eclipse Mosquitto ist ein leichtgewichtiger MQTT-Broker, der im Projekt als zentraler Kommunikationsbroker eingesetzt wird. Er sorgt dafür, dass Echtzeit-Benachrichtigungen unabhängig und effizient aggregiert werden können. Mosquitto verwaltet den Nachrichtenaustausch zwischen dem Backend, das über MQTT kommuniziert, und der Chrome Extension, die zwingendermassen Websockets verwendet, sodass beide Systeme zuverlässig miteinander Daten austauschen können.

- **TCP-Port 1883** für Backend-Services.
- **WebSocket-Port 9001** für die Chrome Extension.

4.5.6 Zusammenfassung

Durch die Kombination aus GitHub Actions, Docker Compose und NGINX erfolgt der Betrieb der Yappi-Plattform weitgehend automatisiert. Codeänderungen werden unmittelbar getestet, gebaut und auf dem SwitchEngines-Server ausgerollt. Dies gewährleistet eine kontinuierliche und zuverlässige Auslieferung neuer Versionen.

Kapitel 5

Evaluation

5.1 Usertesting am Hackathon

Ein geplanter Usability-Test wird im Rahmen des an der FHNW stattfindenden Hackathons *Energy Data Hackdays* durchgeführt. Der Hackathon findet am 11. und 12. September 2025 statt und liegt somit zeitlich nach der Abgabe dieser Arbeit.

Die Veranstaltung bringt Entwicklerinnen und Entwickler mit unterschiedlichem fachlichen Hintergrund zusammen, die in kurzer Zeit kollaborativ Projekte umsetzen. Diese Rahmenbedingungen bieten eine geeignete Gelegenheit, Yappi unter realistischen Arbeitsbedingungen zu testen.

Zum Zeitpunkt der Erstellung dieser Arbeit wurde der Test noch nicht durchgeführt. Es besteht jedoch bereits Kontakt mit den Organisatoren, um vor Ort entweder:

- Yappi direkt in einem Entwicklerteam einzusetzen, das die Plattform während des Hackathons aktiv nutzt, oder
- eine Befragung von Teilnehmenden an einem Informations- und Teststand durchzuführen.

Die Durchführung dieses Tests wird es ermöglichen, praxisnahe Rückmeldungen zur Benutzerfreundlichkeit, Funktionalität und Integration von Yappi in den Arbeitsfluss zu erhalten. Zusätzlich bieten dieser Test die Möglichkeit Testdaten zu sammeln um weitere Erkenntnisse zwischen den gesammelten Daten und der Zufriedenheit zu schaffen. Die dabei gewonnenen Erkenntnisse sollen in ein Folgeprojekt (IP6) einfließen.

5.2 Evaluation durch Simulation

Neben dem Praxistest am Hackathon wurde Yappi in einer kontrollierten Testumgebung evaluiert. Ziel ist es, spezifische Funktionen ohne Abhängigkeit von realen Projektdaten oder sensiblen Informationen zu überprüfen.

Die Tests erfolgen in mehreren Schritten:

1. Szenarien entwickeln: Erstellung eines Realistischen nutzerszenario welches das verwenden von Yappi prüft.
2. Testpersonen auswählen: Es wurden Arbeitskollegen (Softwareentwickler) rekrutiert um Testfälle durchzuführen.
3. Testdurchführung: Schrittweises Durchspielen der Szenarien, begleitet von der Methode des lauten Denkens, um Nutzungshürden und Verständnisprobleme zu identifizieren.
4. Ergebniserfassung: Dokumentation der Beobachtungen in einem Protokoll.

5.2.1 Testszenarien

Für die Evaluation von Yappi wurden drei Szenarien definiert, die zentrale Funktionen der Plattform abdecken. Die Szenarien wurden für die Simulation entwickelt um die funktionen von Yappi zu Testen.

Szenario 1: Erfassung der Zufriedenheit nach einem Code-Commit

Ziel: Überprüfung der intuitiven Nutzung und Unterbrechungswirkung der Zufriedenheits erfassung über das IDE-Plugin.

Ablauf:

1. Die Testperson führt eine Programmieraufgabe in IntelliJ IDEA durch.
2. Nach dem Commit erscheint automatisch eine Benachrichtigung des Yappi-Plugins.
3. Die Testperson gibt die aktuelle Zufriedenheit über die Eingabemaske ein.
4. Rückmeldungen zu Verständlichkeit, Geschwindigkeit und Arbeitsfluss werden dokumentiert.

Bewertungskriterien:

- Verständlichkeit und Auffindbarkeit der Funktion
- Geschwindigkeit des Erfassungsprozesses

- Wahrgenommene Unterbrechung des Arbeitsflusses

Szenario 2: API-Key mit der Kalenderintegration verbinden

Ziel: Überprüfung, ob die Verbindung zwischen Yappi und dem persönlichen Kalender mittels API-Key einfach und verständlich hergestellt werden kann. Dabei soll festgestellt werden, ob Nutzerinnen und Nutzer ohne technische Unterstützung den Vorgang erfolgreich durchführen.

Ablauf:

1. Die Testperson ruft in Yappi den Bereich für das Benutzerprofil auf.
2. Sie generiert oder kopiert einen bestehenden API-Key.
3. Der API-Key wird in das dafür vorgesehene Feld in der Browsererweiterung eingefügt.
4. Yappi benachrichtigt automatisch über Kalenderevents.
5. Rückmeldungen zu Verständlichkeit, benötigter Zeit und auftretenden Schwierigkeiten werden dokumentiert.

Bewertungskriterien:

- Verständlichkeit der Anleitung in Yappi
- Einfachheit des Kopier- und Einfügevorgangs des API-Keys
- Zuverlässigkeit der Synchronisation der Kalenderdaten
- Wahrgenommener Zeitaufwand

Szenario 3: Meeting-Erfassung über Kalenderintegration

Ziel: Validierung der automatischen Erkennung und zeitnahen Erfassung von Meetingwahrnehmungen.

Ablauf:

1. Ein Testmeeting wird im Kalender der Testperson eingetragen.
2. Nach beenden des Meeting erhält die Testperson automatisch eine Erinnerung zur Bewertung.
3. Die Testperson füllt das Feedbackformular aus.
4. Feedback zu Auslösezeitpunkt, Relevanz und Eingabeprozess wird dokumentiert.

Bewertungskriterien:

- Zuverlässigkeit der Meeting-Erkennung
- Angemessenheit des Erinnerungszeitpunkts
- Vollständigkeit der abgefragten Informationen

Testpersonen und Testdurchführung

Kurzfristig konnten bei der Arbeit sowie im Freundeskreis drei Testpersonen rekrutiert werden, um die definierten Szenarien durchzuspielen. Jede Testperson absolvierte jeweils zwei der vorgesehenen Testszenarien.

Zu Beginn erhielten die Testpersonen eine kurze Einführung zu Yappi. Anschliessend wurde ihnen anhand des jeweiligen Testfalls eine Aufgabe zugeteilt, die sie selbstständig ausführen sollten.

Für die Durchführung stand ein Testgerät zur Verfügung, auf dem die Yappi-Webseite in Google Chrome geöffnet war und die Yappi-Chrome-Extension installiert wurde. Zudem war in IntelliJ IDEA das Yappi-Plugin vorinstalliert und ein kleines Java-Projekt vorbereitet.

Testergebnisse

Positive Rückmeldungen:

- Einfache und verständliche Lösung.
- Direkte Benachrichtigung im System.
- Kurzer und klar strukturierter Feedback-Block.
- Integration, die direkt nach einem Commit zur Eingabe auffordert.
- Benutzeroberfläche wird als intuitiv wahrgenommen.

Anmerkung: Positive Aspekte der Benutzeroberfläche wurden nur selten explizit genannt, da ein intuitives Design von den Testpersonen als selbstverständlich bzw. neutral bewertet wurde.

Verbesserungsvorschläge:

- Aktualisierung der FAQ, da Inhalte teilweise veraltet sind.
- Beschleunigung der Synchronisierung bei einer hohen Anzahl von Kalendereinträgen.
- Ergänzung einer klaren Anleitung zur Erstellung eines ICS-Links.

5.3 Beantwortung der Fragestellung

Die im Projekt verfolgten Forschungsfragen wurden anhand der definierten Produktziele überprüft. Die Beantwortung erfolgt nachfolgend in direktem Bezug zu den formulierten Zielen und den erzielten Ergebnissen.

Forschungsfrage A

Durch welche Technologien und Schnittstellen kann Yappi erweitert werden, um ein reibungsloses und einfaches Erfassen von Zufriedenheitsdaten zu ermöglichen?

Dieses Ziel wurde im Rahmen der Entwicklung vollständig adressiert. Die Integration in bestehende Arbeitsumgebungen erfolgt über:

- ein **IntelliJ-IDEA-Plugin** zur direkten Erfassung nach einem Code-Commit (Ziel Z-1),
- eine **Kalenderintegration** über das iCalendar-Format zur automatisierten Erkennung von Meetings (Ziel Z-2),
- eine **Browsererweiterung** zur zeitnahen Erfassung der Meetingwahrnehmung (Ziel Z-2).

Die automatisierte und kontextbezogene Aufforderung reduziert den manuellen Aufwand und erhöht die Regelmässigkeit der Erfassungen. Damit wird das Ziel einer nahtlosen Integration in den Arbeitsprozess erreicht.

Forschungsfrage B

Wie können Gesundheitsdaten in die Auswertung der Entwicklerzufriedenheit einfließen?

Es wurde eine konzeptionelle und teilweise technische Lösung für die Anbindung an Apple HealthKit entwickelt (Ziel Z-3). Dadurch können wissenschaftlich relevante Metriken wie Schlafdauer, Ruheherzfrequenz, Stress (HRV) sowie Aktivitätsminuten automatisiert erfasst und mit den Zufriedenheitsdaten verknüpft werden. Die Integration schafft die Grundlage für weiterführende Analysen zu möglichen Zusammenhängen zwischen physiologischen Faktoren und der subjektiven Arbeitszufriedenheit. Aufgrund zeitlicher Einschränkungen wurde diese Anbindung nicht vollständig produktiv umgesetzt, die technische Machbarkeit jedoch validiert.

Forschungsfrage C

Wie kann Yappi Teams und Entwickler dabei unterstützen, aus den erfassten Zufriedenheitsdaten Handlungsempfehlungen abzuleiten?

Die Entwicklung einer intelligenten Empfehlungskomponente, Yappi Coach, wurde konzeptionell ausgearbeitet (Ziel Z-4). Das Konzept sieht vor, Zufriedenheitsdaten, Arbeitskontext und Gesundheitsmetriken zu kombinieren, um datenbasierte und individuell zugeschnittene Handlungsempfehlungen zu generieren. Eine technische Implementierung erfolgte in diesem Projekt nicht, da sie den Rahmen der Arbeit überschreiten würde. Das Konzept bildet jedoch eine belastbare Grundlage für ein Folgeprojekt.

Zusammenfassung

Die Ziele Z-1 bis Z-3 wurden in diesem Projekt vollständig oder prototypisch umgesetzt, womit die Forschungsfragen A und B beantwortet werden können. Die Forschungsfrage C wurde konzeptionell beantwortet, eine praktische Umsetzung ist als nächster Entwicklungsschritt vorgesehen. Insgesamt trägt die Arbeit zur Verbesserung der kontinuierlichen, kontextbezogenen und nutzerfreundlichen Erfassung von Entwicklerzufriedenheit bei.

Kapitel 6

Diskussion

6.1 Reflexion

TODO:

Literatur

- [1] D. Graziotin und F. Fagerholm, “Happiness and the productivity of software engineers”, in *Rethinking Productivity in Software Engineering*, C. Sadowski und T. Zimmermann, Hrsg., Berkeley, CA: Apress, 2019, S. 109–124, ISBN: 9781484242209 9781484242216. DOI: 10.1007/978-1-4842-4221-6_10. besucht am 8. Aug. 2025. Adresse: https://link.springer.com/10.1007/978-1-4842-4221-6_10.
- [2] “2022 software developer happiness report - developer productivity data”, Zenhub, besucht am 8. Aug. 2025. Adresse: <https://www.zenhub.com/reports/software-developer-happiness>.
- [3] C. França, F. Q. B. da Silva und H. Sharp, “Motivation and Satisfaction of Software Engineers”, *IEEE Transactions on Software Engineering*, Jg. 46, Nr. 2, S. 118–140, Feb. 2020, ISSN: 1939-3520. DOI: 10.1109/TSE.2018.2842201. besucht am 8. Aug. 2025. Adresse: <https://ieeexplore.ieee.org/document/8370133>.
- [4] “Announcing the 2024 DORA report”, Google Cloud Blog, besucht am 8. Aug. 2025. Adresse: <https://cloud.google.com/blog/products/devops-sre/announcing-the-2024-dora-report>.
- [5] D. Graziotin, F. Fagerholm, X. Wang und P. Abrahamsson, *What happens when software developers are (un)happy*, 23. Apr. 2018. DOI: 10.48550/arXiv.1707.00432. arXiv: 1707.00432. besucht am 8. Aug. 2025. Adresse: <http://arxiv.org/abs/1707.00432>.
- [6] “Not just a vibe, the stack overflow developer survey is really here - stack overflow”, besucht am 9. Aug. 2025. Adresse: <https://stackoverflow.blog/2025/05/29/not-just-a-vibe-the-stack-overflow-developer-survey-is-really-here/>.
- [7] M. A. Opoku, S.-W. Kang und S. B. Choi, “The influence of sleep on job satisfaction: examining a serial mediation model of psychological capital and burnout”, *Frontiers in Public Health*, Jg. 11, S. 1149367, 24. Aug. 2023, ISSN: 2296-2565. DOI: 10.3389/fpubh.2023.1149367. besucht am 9. Aug. 2025. Adresse: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC10483141/>.

- [8] P. Eriksson, L. Schiöler, M. Söderberg, A. Rosengren und K. Torén, "Job strain and resting heart rate: a cross-sectional study in a Swedish random working sample", *BMC public health*, Jg. 16, S. 228, 5. März 2016, ISSN: 1471-2458. doi: 10.1186/s12889-016-2900-9.
- [9] R. Borchini und M. M. Ferrario, "[Job strain and heart rate variability. New evidence and new prospects]", *Giornale Italiano Di Medicina Del Lavoro Ed Ergonomia*, Jg. 34, Nr. 3, S. 174–176, 2012, ISSN: 1592-7830.
- [10] S. Dallmeyer, P. Wicker und C. Breuer, "The relationship between leisure-time physical activity and job satisfaction: A dynamic panel data approach", *Journal of Occupational Health*, Jg. 65, Nr. 1, e12382, Jan. 2023, ISSN: 1348-9585. doi: 10.1002/1348-9585.12382.
- [11] V. Stray und N. B. Moe, "Understanding coordination in global software engineering: A mixed-methods study on the use of meetings and Slack", *Journal of Systems and Software*, Jg. 170, S. 110717, Dez. 2020, ISSN: 0164-1212. doi: 10.1016/j.jss.2020.110717. besucht am 8. Aug. 2025. Adresse: <https://www.sciencedirect.com/science/article/pii/S0164121220301564>.
- [12] A. Meyer, E. T. Barr, C. Bird und T. Zimmermann, "Today was a Good Day: The Daily Life of Software Developers", eng, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Jg. 47, Nr. 5, S. 863–880, Mai 2021, ISSN: 0098-5589. doi: 10.1109/TSE.2019.2904957. besucht am 8. Aug. 2025. Adresse: <https://www.zora.uzh.ch/id/eprint/170375/>.
- [13] "With courier, officevibe enables open communication for teams in the workplace.", courier, besucht am 9. Aug. 2025. Adresse: <https://www.courier.com/use-cases/workleap-officevibe>.
- [14] "Tech team health check: An overall guide", besucht am 9. Aug. 2025. Adresse: <https://www.revelo.com/blog/tech-team-health-check-a-template-for-your-organization>.
- [15] P. Budner, J. Eirich und P. A. Gloor, "Making you happy makes me happy" – *Measuring Individual Mood with Smartwatches*, 14. Nov. 2017. doi: 10.48550/arXiv.1711.06134. arXiv: 1711.06134. besucht am 9. Aug. 2025. Adresse: <http://arxiv.org/abs/1711.06134>.
- [16] "How software engineering intelligence platforms boost developer productivity", InfoWorld, besucht am 9. Aug. 2025. Adresse: <https://www.infoworld.com/article/2335502/how-software-engineering-intelligence-platforms-boost-developer-productivity.html>.

- [17] “Git analytics: Challenges, tools & key metrics”, besucht am 9. Aug. 2025. Adresse: <https://axify.io/blog/git-analytics>.
- [18] zachminers. “Introduction to viva insights”, besucht am 9. Aug. 2025. Adresse: <https://learn.microsoft.com/en-us/viva/insights/introduction>.
- [19] “Health API | Garmin Connect Developer Program | Garmin Developers”, besucht am 10. Aug. 2025. Adresse: <https://developer.garmin.com/gc-developer-program/health-api/>.
- [20] “HealthKit”, Apple Developer Documentation, besucht am 10. Aug. 2025. Adresse: <https://docs.developer.apple.com/documentation/healthkit>.

A Verlinkungen

Alle änderungen im Backend und Frontend befinden sich auf dem Branch "develop"

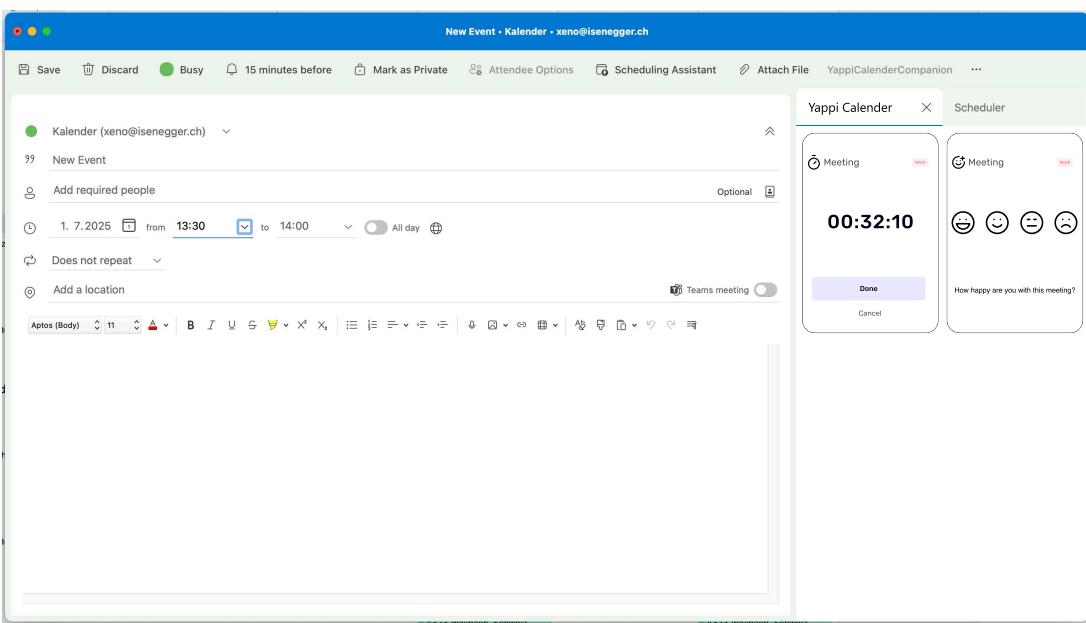
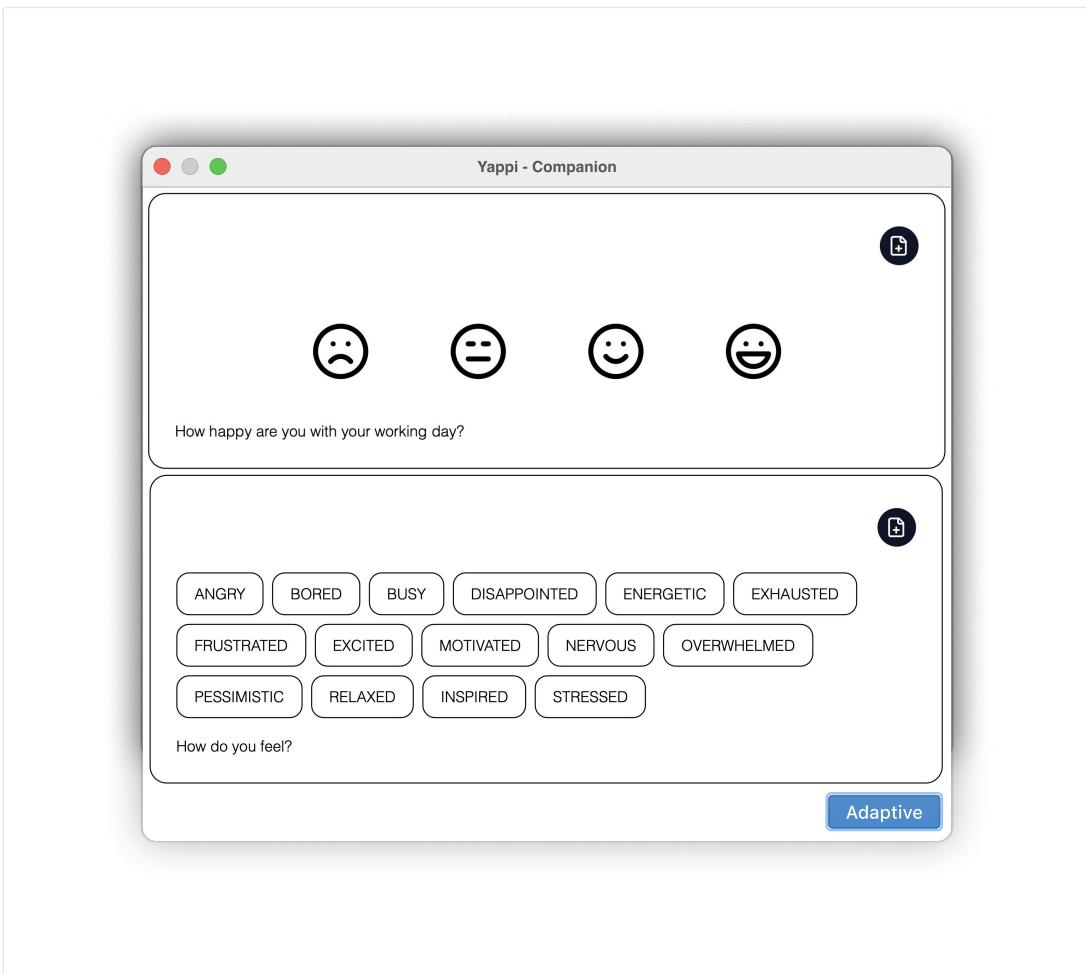
GitHub Backend GitHub Frontend

Alle Companion Apps befinden sich jeweils dem Branch "main"

GitHub IntelliJ Companion GitHub Chrome Extension GitHub Health Companion

B Mockup





C Anhang

25FS_IIT32: Plattform zur Analyse von Entwicklerzufriedenheit und Produktivität in agilen Teams

Betreuer: [Norbert Seyff](#)
[Nitish Patkar](#)

Priorität 1
Arbeitsumfang: P5 (180h pro Student)
Priorität 2
Teamgrösse: 2er Team

Sprachen: Deutsch oder Englisch
Studiengang: Informatik

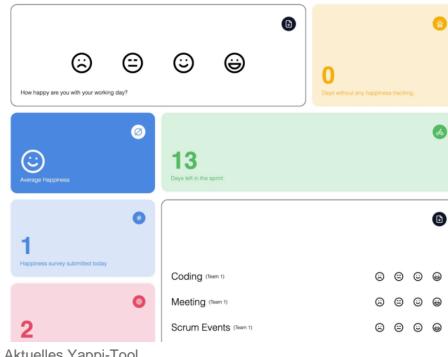
Ausgangslage

Die Produktivität in agilen Teams wird oft durch technische Metriken wie Commit-Frequenz oder geschlossene Tickets gemessen. Studien zeigen jedoch, dass nicht-technische Faktoren wie Zufriedenheit, Arbeitsunterbrechungen und äußere Einflüsse ebenfalls eine wesentliche Rolle spielen.

Das Proof-of-Concept-Tool "Yappi" wurde entwickelt, um Zufriedenheitsdaten durch Selbstberichterstattung zu erfassen und grundlegende GitHub-Daten zu integrieren.

Dashboard

Manage your surveys, track worktypes, stay updated, and optimize your day — all in one place.



Ziel der Arbeit

Das Projekt zielt darauf ab, Yappi zu einem umfassenden Werkzeug für agile Teams und Studierendenprojekte weiterzuentwickeln. Dabei sollen:

- Zusätzliche Datenquellen integriert werden, darunter GitHub-Metriken wie Pull Requests und Commit-Historien sowie Kalenderdaten zur Analyse von Meetings und Unterbrechungen.
- Ein benutzerfreundliches Dashboard entstehen, das technische und nicht-technische Metriken kombiniert und die Beziehungen zwischen Produktivität, Zufriedenheit und Teamdynamik anschaulich darstellt.
- Die Lösung in retrospektiven Meetings auf Praxistauglichkeit und Mehrwert für Teamdynamik und Produktivität evaluiert werden.

Problemstellung

Das Projekt zielt darauf ab, die folgenden Fragen zu klären:

- Was wissen Praktiker über Produktivitätsmessungen, und welche verwenden sie in der Praxis?
- Welche Datenquellen sind für die Entwicklung eines umfassenden Produktivitätsverständnisses unerlässlich?
- Welche Visualisierungen veranschaulichen am besten die Zusammenhänge zur Teamdynamik und machen sie für Retrospektiven nutzbar?
- Welche Schlussfolgerungen lassen sich aus der Verwendung des Tools während Retrospektiven ziehen?

Technologien/Fachliche Schwerpunkte/Referenzen

Backend: Spring Boot, Frontend: React,

Relevante Literatur:

- Graziotin, D., Wang, X., & Abrahamsson, P. (2014). Happy software developers solve problems better: psychological measurements in agile environments.