Plattform zur Analyse von Entwicklerzufriedenheit und Produktivität in agilen Teams

IP5 Project

Windisch, August 2025

Studenten: Xeno Isenegger, Gideon Monterosa
Fachbetreuer: Norbert Seyff, Nitish Patkar

Fachhochschule Nordwestschweiz, Hochschule für Informatik

Abstract

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

TODO: gendern überprüfen

Die Zufriedenheit von Entwicklerinnen und Entwicklern wird, wenn überhaupt, meist nur anhand der Menge ihrer geleisteten Arbeit gemessen. Dabei entstehen zwangsläufig Defizite, und ein halbjährli- ches 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 Ent- wicklerzufriedenheit 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. Zusätz- lich können spezifische Aufgaben und Arbeitstypen individuell bewertet werden. Die erhobenen Da- ten werden anonym auf Teamebene analysiert, um ein fundiertes Verständnis für die Stimmung in- nerhalb 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 opti- mieren. Es wird untersucht, wie die Daten noch präziser erfasst und ausgewertet 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.

1.2 Ziele und Vision

Yappi wird zu einer umfassenden Plattform weiterentwickelt, die nicht nur die Zufriedenheit misst, sondern sich nahtlos in den Arbeitsalltag integriert und wertvolle Handlungsempfehlungen liefert. Dazu werden folgende Kernaspekte umgesetzt:

1. **Produktivitätsfaktoren identifizieren**
   Durch eine tiefere Analyse von Zufriedenheitsindikatoren sollen zentrale Faktoren ermittelt werden, die sich positiv oder negativ auf die Produktivität und das Wohlbefinden von Entwicklerinnen und Entwicklern auswirken. Diese Erkenntnisse werden genutzt, um Vorschläge zu Verbesserungsmass- nahmen abzuleiten.

2. **Integration in den Arbeitsprozess** Yappi soll sich direkt in bestehende Arbeitsabläufe einfügen, um die Erfassung der Zufriedenheit möglichst intuitiv und effizient zu gestalten. Dies kann durch verschiedene Schnittstellen und Erwei- terungen erfolgen, die eine nahtlose Interaktion ermöglichen.

3. **Erweiterung um kontextbezogene Daten** Um ein umfassenderes Bild der Arbeitszufriedenheit zu erhalten, können weitere Einflussfaktoren berücksichtigt werden. Dazu gehören beispielsweise arbeitsbezogene Rahmenbedingungen oder individuelle Gesundheits- und Belastungs-

indikatoren. Diese Daten sollen helfen, ein besseres Ver- ständnis für langfristige Trends und Zusammenhänge zu entwickeln.

4. **Intelligente Analyse und Handlungsempfehlungen** Durch die Integration von AI schnittstellen können gezielte Analysen erstellt und individualisierte Empfehlungen abgeleitet werden. Dies kann sowohl auf individueller als auch auf Teamebene erfol- gen, um nachhaltige Verbesserungen im Arbeitsumfeld zu fördern.

Fazit

Mit diesen Erweiterungen wird Yappi zu einem essenziellen Bestandteil des Entwickleralltags. Es bietet nicht nur eine präzisere Erfassung der Zufriedenheit, sondern liefert auch wertvolle Einblicke und Handlungsempfehlungen, um die Arbeitsbedingungen nachhaltig zu verbessern. Unternehmen erhalten fundierte Analysen und können gezielt Massnahmen ergreifen, um eine motivierte und pro- duktive Entwicklergemeinschaft zu fördern.

In der Umsetzung dieses Projekts liegt der Schwerpunkt auf den ersten drei genannten Zielen, mit besonderem Fokus auf die Integration in bestehende Arbeitsprozesse sowie die Erweiterung um kontextbezogene Daten. Die Identifikation von Produktivitätsfaktoren wird im Rahmen der Datenerhebung und -analyse vorbereitet, um künftig eine fundierte Ableitung von Verbesserungsvorschlägen zu ermöglichen. Das vierte Ziel, die Entwicklung einer intelligenten Analyse- und Empfehlungskomponente, wird in diesem Projekt konzeptionell erarbeitet, jedoch noch nicht technisch umgesetzt. Auf diese Weise wird eine solide Grundlage geschaffen, um zukünftig auch das vierte Ziel zu erreichen. Die Umsetzung dieses Ziels wird in einer folgenden Bachelorarbeit genauer thematisiert.

1.3 Fragestellungen

TODO: Einleitung wie diese Fragen aus dem Zielbild entsteht und wie diese Frager

TODO: Fragen 1-2 werden umgesetzt Frage 3 wird konzeptionell ausgearbeitet

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 Produktivitätsme-

triken 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.

Kapitel 2

Hintergrund

TODO: kurze Einleitung

2.1 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.

2.2 Aktueller technischer Stand

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

Frontend Das Frontend basiert auf React und Next.js mit TypeScript. Für Aufrufe werden SWR-Hooks verwendet, dabei handelt es sich ume eine React library um Daten zu laden. Komponenten werden mit der Library wird 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 gepseichert. Die Anfragen ans Backend werden mit JSON Web Tokens (JWT) abgesichert.

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.

2.3 Parallele Entwicklung auf gemeinsamer Codebasis

TODO: Zusammenarbeit kurz erklären mit Auswirkungen auf unser Projekt

2.4 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.

Kapitel 3

Methoden

TODO: Kanban erwähen

3.1 Projektmethodik

3.2 Prototypen

3.3 Proof of Concepts

TODO: Abgrenzung bezüglich Datenschutz
TODO: arc42 erwähen

Kapitel 4

State of the Art

4.1 Definition von Entwiklerzufriedenheit

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 verwendt 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 dem 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 Zufriedenheiteher durch allgemeines Wohlbefinden und gute Laune charakerisiert 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, ...).

4.2 Stand der Forschung und verwandte Arbeiten

TODO: Definitionen für Erfolgsparameter einfügen

In den letzen Jahren haben zahlreiche Studien den Zusammenhang zwischen der Entwicklerzufriedenheit und Erfogsparametern wie Produktivität, Codequalität und Mitarbeiterbindung untersucht. Bei einer grossangelegten Studie wurden 317 Softwareentwickler befragt und dabei 42 Konsequenten von Unzufriedenheit sowie 32 Konsequenzen von Zufriedenheit beim Programmieren identifiziert. Die Ergebnisse Zeigen, dass Entwicklerzufriedenheit messbare Auswirkungen auf den Enwicklungsprozess, 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 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].

Neben akademischen Studien liefern auch Industireumfragen und Community-Studien wertvolle Einblicke. Der jährliche Stack Overflow Developer Survey etwa spiegelt wieder, dass weiterhin Verbesserungsbedarf besteht. laut der Umfrage 2024 bezeichen 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].

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.

TODO: evt. Methoden und Instrumente zur Messung der Entwicklerzufriedenheit
TODO: evt. Technologische und methodische Ansätze in verwandten Arbeiten
TODO: Forschungslücken

4.3 Bestehende Lösungen und Wettbewerbsanalyse

Der Markt bietet bereits verschiedene Tools und Plattformen, die Teilaspekte 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 allge-

meines 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 [7].

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 [8].

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 [9].

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 [10].

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 [11].

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 [12].

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.

Kapitel 5

Konzeptentwurf

Die im vorhergehenden Kapitel dargestellte Analyse bestehender Lösungen verdeutlicht, dass es derzeit keine Plattform gibt, welche emotionale Faktoren, technischen Kontext und physisches Wohlbefinden integriert erfasst und daraus konkrete Handlungsempfehlungen ableitet. Yappi soll genau diese Lücke schliessen und eine einheitliche, praxisorientierte Lösung bieten. Ziel ist es, die Entwicklerzufriedenheit umfassend zu erfassen und die gewonnenen Erkenntnisse in konkrete Massnahmen zu überführen. Die geplanten Kernfunktionen umfassen:

Integriertes Erfassen im Arbeitsfluss: Yappi lässt sich nahtlos in den täglichen Entwicklungsprozess einbetten. Durch Plugins für Entwicklungsumgebungen, Browser-Erweiterungen oder Integrationen in Kollaborationstools können Entwickler ihre Zufriedenheit direkt in ihrer Arbeitsumgebung erfassen, ohne den Kontext zu wechseln. Dies erhöht die Teilnahmebereitschaft und stellt sicher, dass Feedback leicht und unmittelbar gegeben werden kann ohne den Arbeitsfluss zu unterbrechen.

Automatischer Kontext durch Prozessdaten: Neben manuellen Stimmungsangaben bezieht Yappi automatisch Kontextinformationen aus dem Arbeitsprozess ein. Beispielsweise können Commit-Daten aus Git

und Kalendereinträge herangezogen werden, um die Stimmung in Bezug zu objektiven Ereignissen zu setzen. So liesse sich erkennen, ob z.B. eine Häufung negativer Stimmungswerte mit vielen Meetings korreliert.

Erweiterung um Gesundheitsmetriken: Um ein umfassenderes Bild des Wohlbefindens zu erhalten, soll Yappi optional auch Gesundheitsdaten einbeziehen. Dies kann durch die Anbindung an gängige Gesundheits-APIs oder Wearables (ähnlich Happimeter) erfolgen. Die so gewonnenen zusätzlichen Datenpunkte ermöglichen es, langfristige Entwicklungen zu erkennen und diese im Zusammenhang mit Prozess- sowie Zufriedenheitsdaten zu analysieren.

KI-gestützter Coach mit Handlungsempfehlungen: Über die reine Datenerfassung hinaus soll Yappi einen intelligenten Empfehlungsdienst bereitstellen. Dieser Yappi-Coach analysiert die erfassten Zufriedenheitsdaten, Produktivitätskennzahlen und gegebenenfalls Gesundheitsdaten mittels KI-gestützter Verfahren. Auf dieser Grundlage werden gezielte, evidenzbasierte Empfehlungen für einzelne Entwicklerinnen und Entwickler sowie für Teams generiert. Beispielsweise könnte der Coach vorschlagen, die Dauer von Team-Meetings auf höchstens 1,5 Stunden zu begrenzen, wenn längere Sitzungen mit sinkender Zufriedenheit einhergehen. Ebenso kann er empfehlen, nach vier Stunden konzentrierter Entwicklungsarbeit eine Pause einzulegen, sofern die Analyse zeigt, dass dies die Zufriedenheit steigert. Ziel ist es, konkrete Handlungsimpulse zu geben, die sowohl das individuelle Wohlbefinden als auch die Teamproduktivität verbessern.

Durch die Kombination dieser Aspekte entsteht mit Yappi eine zentrale Plattform, die kontinuierliches Stimmungs-Tracking, automatisierte Kontextdaten aus dem Arbeitsprozess, Gesundheitsmetriken und KI-gestützte Handlungsempfehlungen integriert. Ziel ist es, die Entwicklerzufriedenheit nicht nur präzise zu erfassen, sondern die gewonnenen Erkenntnisse unmittelbar in nachhaltige Verbesserungen des Arbeitsalltags zu überführen. Unternehmen erhalten damit eine fundierte Entscheidungsgrundlage, um gezielt Massnah-

men zur Förderung einer motivierten, gesunden und leistungsfähigen Entwicklergemeinschaft umzusetzen. Die nachfolgenden Abschnitte greifen diese Kernaspekte auf und verknüpfen sie mit weiteren Konzeptbestandteilen, um ein ganzheitliches Lösungsdesign zu entwickeln.

5.1 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.

Kommunikationsmechanismus

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.

Systemarchitektur der Integrationsplattform

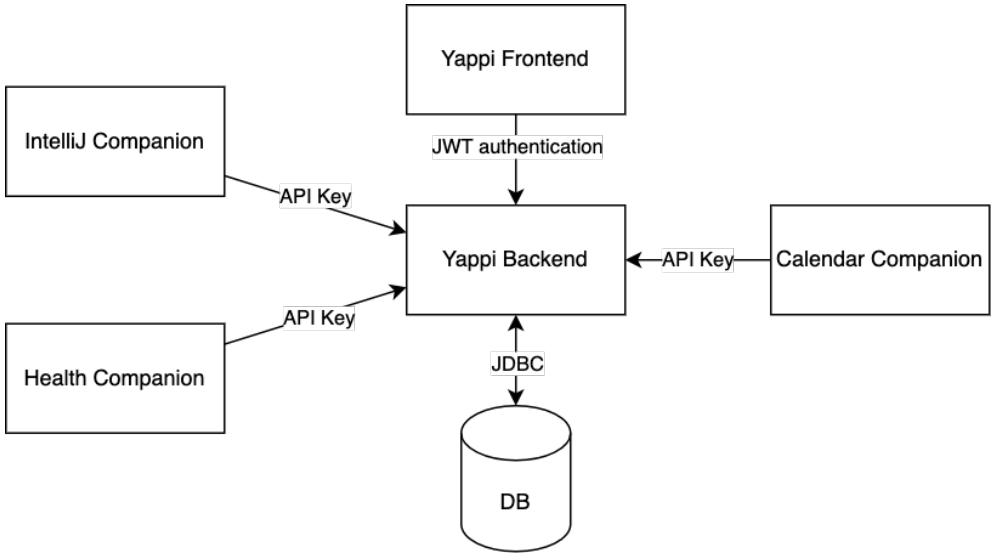

Abbildung 5.1: Systemarchitektur der Yappi-Integrationsplattform

Das in Abbildung 5.1 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 Authentifizie-

rung zur Verwaltung von Nutzersitzungen. Externe Companion-Apps, wie beispielsweise die IntelliJ-, Health- oder Calendar-Companion, sind über einen API-Key-Mechanismus angebunden. Dieser gewährleistet, dass nur registrierte und autorisierte Clients Daten an das System übermitteln können. Alle eingehenden Daten werden vom Backend in der dargestellten Datenbank (DB) persistiert.

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.

Ablauf der API-Key Erstellung und -Verwendung

Abbildung 5.2 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.

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

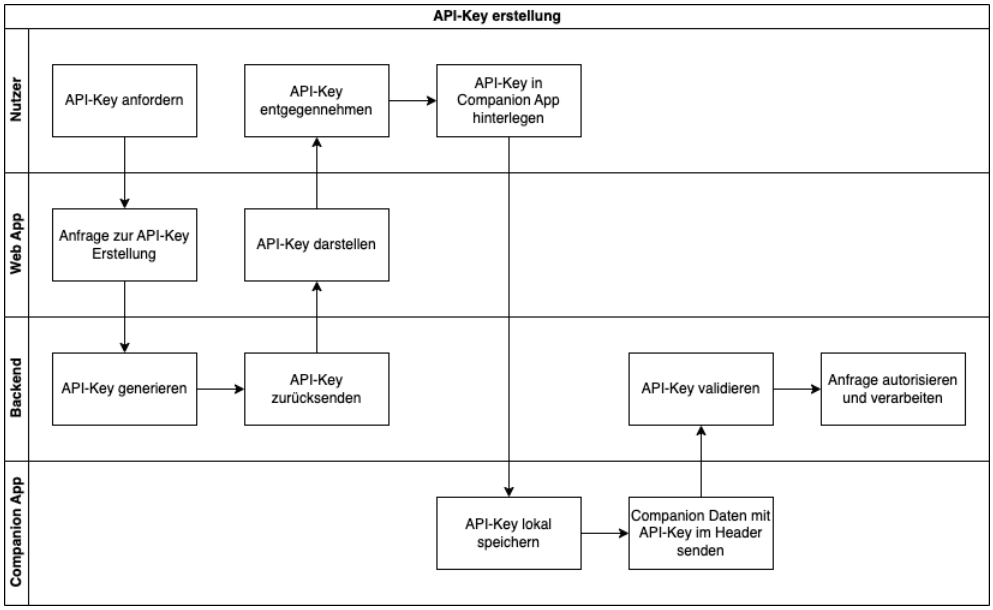

Abbildung 5.2: Ablauf der API-Key Erstellung und -Verwendung in Yappi

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.

5.2 Companion Apps

TODO: kurze Einleitung

5.3 Integration in die Entwicklungsumgebung

TODO XENO: rework

Yappi wird direkt in die integrierte Entwicklungsumgebung (IDE) eingebunden, um den Arbeitsfluss von Entwicklerinnen und Entwicklern nicht zu unterbrechen.Als primäre Zielplattform eignet sich ein Plugin für IntelliJ IDEA,

eine der weltweit am häufigsten genutzten Java-Entwicklungsumgebungen von JetBrains. Die Wahl von IntelliJ IDEA basiert auf drei Hauptfaktoren: hohe Verbreitung im professionellen Java-Umfeld, umfangreiche Plugin-Schnittstellen zur Integration externer Funktionen und Unterstützung mehrerer Programmiersprachen und Build-Systeme, was den Einsatz auch in heterogenen Entwicklungsteams ermöglicht. Die unmittelbare Einbettung in die gewohnte Arbeitsoberfläche erlaubt die Erfassung von Zufriedenheitsdaten ohne Kontextwechsel.

Die Integration trägt dazu bei, den Flow-Zustand zu erhalten. Flow bezeichnet einen Zustand tiefer Konzentration, der nur entsteht, wenn Anforderungen und Fähigkeiten im Gleichgewicht stehen und Unterbrechungen vermieden werden. Kontextwechsel zu externen Tools oder Browsern können diesen Zustand unterbrechen und damit Produktivität sowie Zufriedenheit verringern. Das IntelliJ-Plugin reduziert solche Unterbrechungen, indem es Feedbackprozesse unmittelbar in den bestehenden Arbeitskontext integriert.

Eine Post-Commit-Aktion ermöglicht es, direkt nach einem abgeschlossenen Commit ein kurzes Feedback abzugeben. Dadurch entfällt der Umweg über die Webplattform von Yappi, was die Bedienung beschleunigt und die Gefahr von Befragungsmüdigkeit verringert. Kurze, kontextbezogene Abfragen im richtigen Moment erhöhen die Teilnahmebereitschaft und verbessern die Datenqualität.

Ziel ist ein Erfassungsprozess, der unauffällig, effizient und vollständig in den Entwicklungsprozess integriert ist.

5.4 Integration von Kalenderdaten

TODO XENO:
TODO: Sytemdiagramm

Ein vollständiges erfassen der Entwicklerzufriedenheit erfordert die Berücksichtigung aller Arbeitsaspekte, insbesondere auch Besprechungen (Meetings).

Empirische untersuchungen zeigen, dass Entwickler durchschnittlich zwischen 10,9 Stunden und 16,5 Stunden pro Woche in Besprechungen verbringen. Ein erheblicher Teil dieser Zeit wird insbesondere bei grossen oder ungünstig terminierten Besprechungen als wenig produktiv bewertet. Eine hohe Meetingfrequenz kann den Arbeitstag fragmentieren und den Flow unterbrechen [13], [14].

Ziel dieses Konzeptentwurf ist die Entwicklung einer Kalendererweiterung für Yappi, welche durch das Integrieren des Persönlichen Kalender in die bestehenden Webanwendung Yappi und einer Companion-App welche durch eine Browser-Erweiterung einsicht in die Meetings der Entwickler ermöglicht. Die Erweiterung soll nach Ende eines Meetings automatisch, den Entwickler nach Feedback abfragen und die Ergebnisse davon in Yappi für weitere Auswertungen bereitstellen.

Messung der Meetingqualität

TODO XENO: quelle einbetten

Zur Bewertung werden sieben Kriterien definiert. Diese sind in Anlehnung an Best Practices und gängige Meeting-Umfrageinstrumente ausgewählt:

1. **Zielklarheit und Relevanz**
   Prüft, ob die Meetingziele klar formuliert und relevant waren. Eine präzise Agenda und eindeutige Zieldefinition steigern die Produktivität.

2. **Inhaltliche Tiefe und Verständlichkeit**
   Bewertet, ob Themen angemessen tief und zugleich verständlich behandelt wurden. Unnötige Detailfülle ist zu vermeiden.

3. **Zeitmanagement**
   Umfasst Pünktlichkeit beim Start und Einhaltung der geplanten Dauer. Studien empfehlen, Meetingzeiten bewusst zu verkürzen (z. B. 25 statt 30 Minuten), um Effizienz zu erhöhen.

4. **Moderation und Beteiligung**
   Erfasst die Qualität der Moderation und die aktive Einbindung der

Teilnehmenden. Eine hohe Beteiligungsquote gilt als Indikator für Interaktivität.

5. **Ergebnisorientierung**
   Misst, ob konkrete Entscheidungen oder Aufgaben resultierten. Die Anzahl abgeschlossener Aktionspunkte kann als Kennzahl dienen.

6. **Allgemeine Zufriedenheit**
   Ermittelt das subjektive Gesamturteil, beispielsweise auf einer Skala von 1 bis 10.

7. **Meetingdauer**
   Vergleicht geplante mit tatsächlicher Dauer und bewertet die Angemessenheit.

Diese Kriterien ermöglichen ein umfassendes Bild der Meetingqualität und liefern Ansatzpunkte für Verbesserungen.

Kalenderintegration auf Basis offener Standards

Zur automatischen Erkennung relevanter Meetings wird eine Anbindung an das Kalendersysteme des Entwickler 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.

Companion-App als Feedback-Trigger

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.

Datenverarbeitung und Auswertung

Die erhobenen Feedbackdaten werden ausschliesslich auf dem Yappi-Server verarbeitet. Vor der Auswertung erfolgt eine Anonymisierung, um Rückschlüsse auf einzelne Personen zu verhindern. Für jedes Meeting werden aggregierte Kennzahlen, wie Durchschnittswerte der sieben Kriterien oder Verteilungen der Bewertungen, berechnet. Diese werden nur dann angezeigt, wenn eine vordefinierte Mindestanzahl an Rückmeldungen vorliegt. Neben der Auswertung einzelner Meetings können über längere Zeiträume Trends analysiert und Optimierungsmassnahmen evaluiert werden.

Integration in die Yappi Webanwendung

Die bestehenden Funktionen von Yappi werden um einen Bereich zur Meetinganalyse ergänzt. Entwickler sehen hier ihre eigenen Feedbackabgaben und offene Anfragen. Auf der Teamebene kann auf aggregierte, anonymisierte Auswertungen zu den Meetings ihres Teams zugegriffen werden. Die Konfiguration der Kalenderintegration und der Verbindung zur Companion-App wird im Nutzerprofil zentral verwaltet. Dadurch bleibt die Erweiterung vollständig in die bestehende Systemarchitektur eingebettet und benötigt keine separaten Plattformen.

5.5 Integration von Gesundheitsdaten

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.

Auswahl relevanter Gesundheitsmetriken

Die Integration von Gesundheitsdaten in Yappi soll das subjektive Stimmungsbild ergänzen und objektive Indikatoren für Belastung, Erholung und generelles Wohlbefinden liefern. Subjektive Zufriedenheitsangaben geben wertvolle Einblicke in die aktuelle emotionale Lage. Physiologische Metriken hingegen erfassen Veränderungen, die den Betroffenen nicht immer bewusst sind. Beide Perspektiven zu kombinieren ermöglicht eine fundiertere Analyse potenzieller 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.

Basierend darauf wurden folgende vier Metriken ausgewählt:

1. **Schlafdauer**
   Schlafdauer 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 Schlafmangel die Produktivität stark mindern [15].
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 [16].

3. **Stress (Herzratenvariabilität, HRV)**
   Die Herzratenvariabilität 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 [17].

4. **Aktivitätsminuten und Schritte**
   Körperliche Aktivität wirkt sich positiv auf mentale Gesundheit, Energielevel 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 [18].

Diese Metriken ermöglichen eine gezielte Verknüpfung zwischen erholungs- und gesundheitsbezogenen Faktoren sowie den erhobenen Zufriedenheits- und Prozessdaten. Damit lassen sich Muster identifizieren, die potenzielle Belastungssituationen aufzeigen und gezielte Massnahmen zur Förderung von Wohlbefinden, Motivation und Leistungsfähigkeit ermöglichen.

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 geeignetere 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.

5.6 Yappi Coach

TODO: Ausgearbeitets Konzept dokumentieren

QUESTION: hier erwähnen dass umsetzung erst im nächsten projekt?

5.7 Konzeptevaluation

Ziel der Evaluation war 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. An der Befragung nahmen acht individuelle Softwareentwickler teil.

Die Abbildung 5.3 zeigt die nach Themen gruppierten Durchschnittsbewertungen. 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.

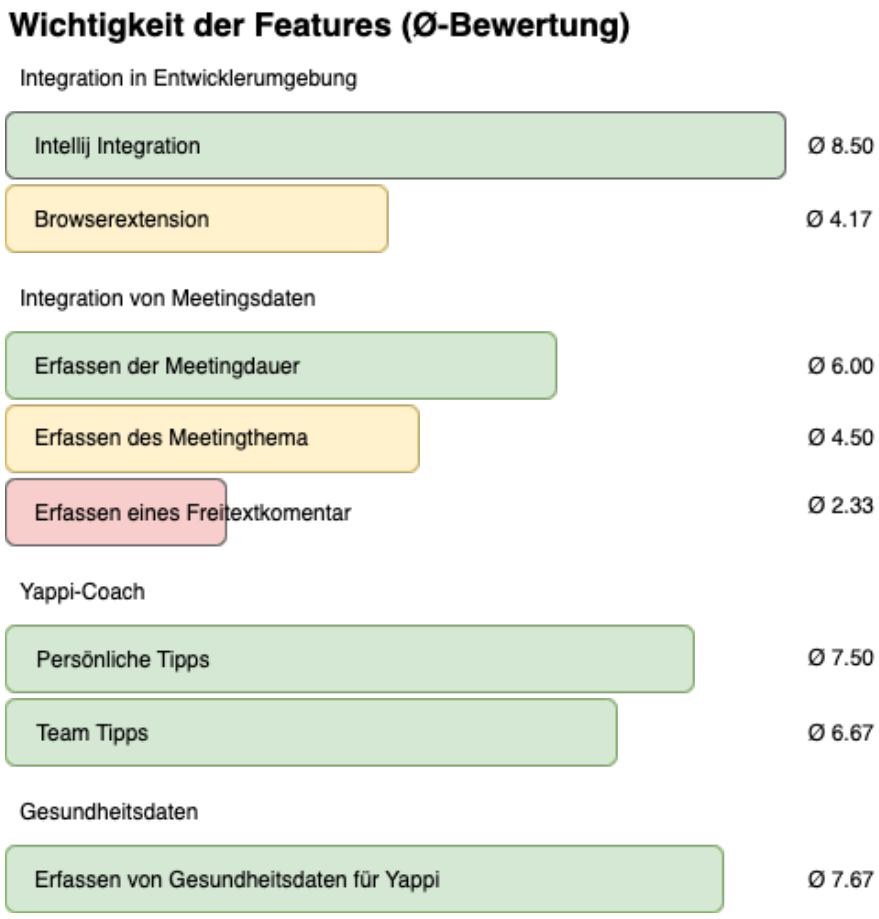

Abbildung 5.3: Durchschnittsbewertung der Konzeptfunktionen auf einer Likert-Skala von 0 = sehr unwichtig bis 10 = sehr wichtig.

Kapitel 6

Implementierung

6.1 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 offengelegt 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>
```

Datenbank

Die Tabelle `api_key` speichert die Metadaten und API-Keys.

Spalte	Datentyp	Beschreibung
`id`	SERIAL	Primärschlüssel, auto-inkrementierend
`user_id`	INTEGER	Fremdschlüssel auf `users.id`
`hashed_key`	TEXT	Gesalteter Hash des API-Keys (z. B. BCrypt oder Argon2)
`created_at`	TIMESTAMPTZ	Zeitpunkt der Erstellung des Keys
`expires_at`	TIMESTAMPTZ	Optionales Ablaufdatum
`active`	BOOLEAN	Aktivierungs-Flag, um Keys ohne Löschung zu sperren

Tabelle 6.1: Schema der Tabelle `api_key`

Backend-Komponenten

- **Entität** (`ApiKey.java`) - Abbildung der Tabelle `api_key` als JPA-Entität, ohne Feld für den rohen Key.
- **Repository** (`ApiKeyRepository.java`) - Zugriff auf API-Key-Datensätze, inkl. Suche nach Key-Präfix.
- **Util-Klasse** (`ApiKeyUtil.java`) - Generierung zufälliger Keys, Trennung von Präfix und geheimem Teil, Hashing des geheimen Teils mit einem sicheren Algorithmus.
- **Service** (`ApiKeyService.java`) - Erstellung, Speicherung und Validierung von Keys.
- **Controller** (`ApiKeyController.java`) - REST-Endpunkte zur Verwaltung von Keys.

API-Endpunkte

Basispfad: `/apikey`. Alle Endpunkte setzen einen gültigen JWT im `Authorization`-Header voraus (`Bearer <token>`).

- `GET /apikey`
  Liefert den aktiven API-Key des aktuell authentisierten Nutzers als Metadatenobjekt (kein Klartext-Key).
  Antworten: `200 OK`, `404 Not Found` (kein aktiver Key), `401 Unauthorized`

(fehlender/ungültiger Header).

- `POST /apikey/generate`
  Erzeugt einen neuen API-Key für den authentisierten Nutzer und gibt den Klartext-Key einmalig zurück.
  Antworten: `200 OK`, `401 Unauthorized`.

Nutzung geschützter Ressourcen: Externe Clients authentisieren sich anschliessend mit `X-API-KEY: <key>` gegenüber den Companion-/Backend-Endpunkten. Der Klartext-Key wird nicht gespeichert; in der Datenbank liegt nur `hashed_key`.

Beispiel (cURL):

```
# Key erzeugen
curl -H "Authorization: Bearer <JWT>" -X POST https://<host>/apikey/generate

# Aktiven Key (Metadaten) abrufen
curl -H "Authorization: Bearer <JWT>" https://<host>/apikey
```

Verarbeitungslogik

// **Erstellung:**

1. Generierung eines neuen Keys (Präfix + geheimer Teil).
2. Hashing des geheimen Teils.
3. Speicherung des Hashes und der Metadaten in der Tabelle `api_key`.
4. Rückgabe des vollständigen Keys an den Client (nur einmalig).

Validierung:

1. Extraktion des Präfixes und geheimen Teils aus dem API-Key-Header.
2. Abruf des Datensatzes anhand des Präfixes.

3. Prüfung auf Existenz, Aktivstatus und Ablaufdatum.
4. Vergleich des Hashes mit dem übergebenen geheimen Teil.
5. Bei Erfolg: Authentifizierung des zugehörigen Benutzers.

Sicherheitsintegration

- **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.

Vorteile

- Schlüssel werden nie im Klartext gespeichert.
- Flexible Verwaltung durch Ablaufdaten und Aktivierungs-Flag.
- Sichere Hashing-Verfahren verhindern Missbrauch bei Datenbankkompromittierung.

TODO: Quelle für spring Securtiy Architektur
https://docs.spring.io/spring-security/reference/servlet/architecture.html **TODO: Hash Algori**
TODO: Datenbank Erweiterung
TODO: Security "Workarroundërklären
TODO: Beispiel Request (Sequenzdiagramm)
TODO: Front End Anpassung

6.2 IntelliJ IDEA Companion

TODO:

6.3 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 aus standardisierten iCalendar Quellen zu synchronisieren und direktes Feedback zu ermöglichen. Dieses Kapitel beschreibt die technische Implementierung der einzelnen Komponenten, deren Zusammenspiel sowie die zugrunde liegenden Datenstrukturen.

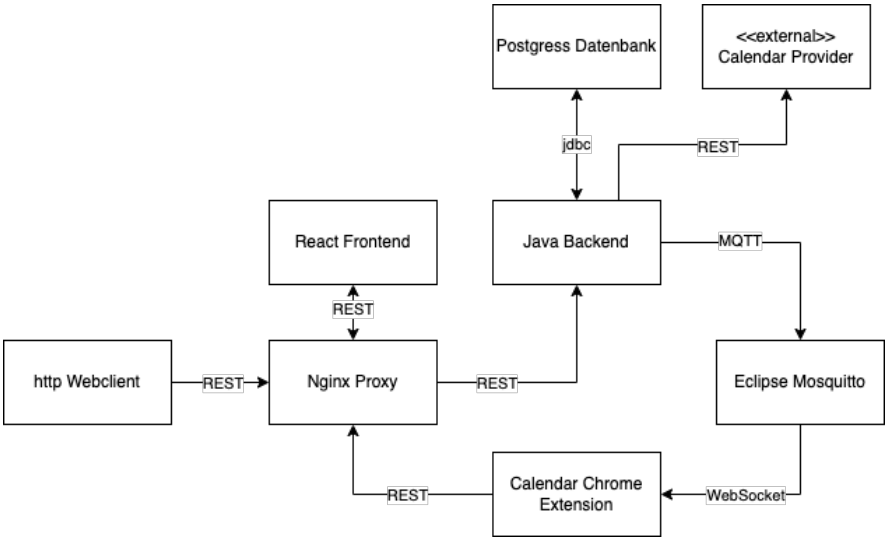

Abbildung 6.1: Systemübersicht zur Kalenderintegration in Yappi

Die Abbildung 6.1 zeigt eine Systemübersicht der Kalenderintegration.

6.3.1 Yappi Chrome Extension

Die Yappi Chrome Extension dient als Companion-Anwendung zur Erfassung von Zufriedenheits-Feedback und Benachrichtigen über das beenden von Kalendereinträgen direkt im Webbrowser. Sie ist als ereignisgesteuerte Anwendung konzipiert und unterstützt sowohl synchrone Datenabfragen als auch asynchrone Echtzeitbenachrichtigungen.

Die Extension ermöglicht es, Umfragen und Benachrichtigungen ohne den Um-

weg über die Yappi-Webplattform zu verarbeiten. Dadurch werden Feedbackprozesse in den Arbeits Flow des Nutzers verlagert und Befragungsmüdigkeit reduziert.

Architektur

Die Anwendung besteht aus zwei Hauptkomponenten:

1. **Persistenter Background-Prozess** Läuft dauerhaft im Hintergrund und ist für die Kommunikation mit externen Diensten, 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 von Inhalten sowie der Erfassung von Feedback.

Benutzeroberfläche (UI)

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

Systembenachrichtigungen

Beim Ende eines Meetings, erzeugt die Extension eine native Systembenachrichtigung siehe Abbildung 6.2. 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 6.2: Systembenachrichtigung der Chrome Erweiterung in macOS

Popup-Ansicht

Das Browser-Popup siehe Abbildung 6.3 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 Backend (z. B. `Connected` mit grüner Statusanzeige),
- Benutzerhinweis zum erfolgreichen Setup,
- eine Liste kürzlich abgeschlossener Meetings mit Datum und Uhrzeit,
- Aktionsbuttons für jedes Meeting:
  - `Review` Öffnet die zugehörige Feedback-Umfrage.
  - `Delete` Entfernt den Eintrag aus der Liste der letzten Meetings.

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

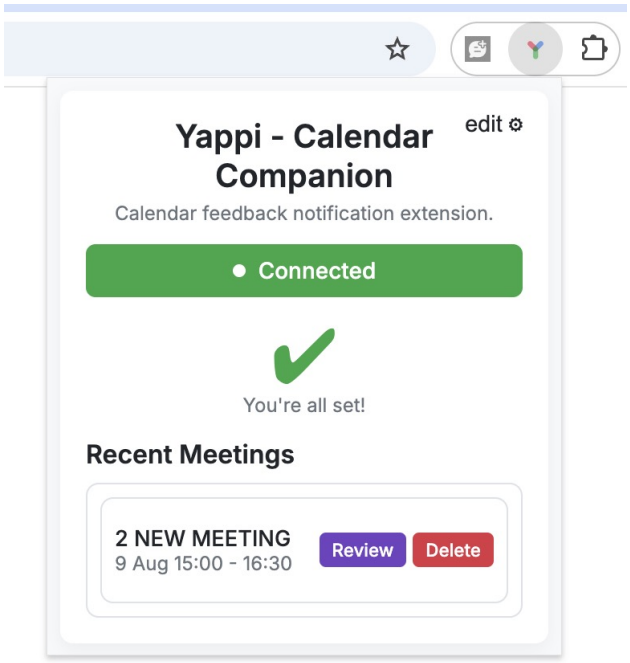

Abbildung 6.3: Benutzeroberfläche der Chrome Erweiterung

Kommunikationskanäle

Die Verbindung zum Yappi-Backend erfolgt über zwei Kanäle:

- **REST-API** Für synchronen Abruf von Daten (z. B. Umfragen) und Übermittlung von Antworten.
- **MQTT-Broker** Für asynchrone Benachrichtigungen in Echtzeit, z. B. über neu verfügbare Umfragen.

Die interne Kommunikation zwischen Background-Prozess und Popup-UI nutzt die Browser-API `chrome.storage`.

Komponenten im Detail

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

- Aufbau und Aufrechterhaltung einer MQTT-Verbindung (über `lib/mqtt.js`)

zu einem Broker via WebSocket.

- Abonnieren benutzerspezifischer Topics zur Echtzeitbenachrichtigung.
- 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 Vorlagen (`popup/questionblock.html`, `popup/questionblock.js`) und integriert diese in die Ansicht.
- Feedback-Übermittlung (`popup/popup.js`): Sammelt Antworten, strukturiert sie als JSON und sendet sie per POST-Request an die REST-API.
- Verlaufsansicht (`popup/history.html`, `popup/history.js`): Liest die letzten zehn gespeicherten Ereignisse aus und stellt sie in einer separaten Ansicht dar.

Externe Abhängigkeiten

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

6.3.2 Backend-Integration

Das Kalendermodul 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 von Clients in Echtzeit. Letzteres erfolgt über das Publish-Subscribe-Protokoll *MQTT* (Mes-

sage Queuing Telemetry Transport). Zudem werden externe iCalendar-Daten (ICS) der Nutzer automatisch synchronisiert.

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.

6.3.3 Datenbankerweiterung

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

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

Tabelle 6.2: Schema der Tabelle `calendar_events`

Backend-Komponenten

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

6.3.4 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:

```
{
    "eventId": 3002,
    "userId": 1,
    "title": "Meeting IP5 - Standup",
    "startDate": "2025-08-09T15:00",
```

```
    "endDate": "2025-08-09T15:15",
    "location": "",
    "questionBlockId": 1,
    "teams": [
        {
            "id": 1,
            "name": "Test Team 1"
        },
        {
            "id": 3,
            "name": "Test Team 2"
        }
    ]
}
```

Vorteile

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

6.3.5 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
`id`	SERIAL	Eindeutiger, automatisch inkrementierender Primärschlüssel
`configuration`	JSONB	Vollständige Konfiguration des Fragenblocks als JSON-Objekt. Ermöglicht flexible Strukturierung und effiziente Abfragen

Tabelle 6.3: 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 6.4 ist eine Grafische-Darstellung einer Beispielumfrage zu sehen.

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

```
{
"id": "1000",
"title": "Basic Meeting Feedback",
"description": "This is am Meeting feedback survey.",
"sections": [
{
    "id": "q1",
    "type": "likert",
    "title": "1. Clarity of Goals & Relevance",
    "prompt": "The goals of the meeting were clearly defined.",
    "required": true,
    "min": 1,
    "label_min": "strongly disagree",
```

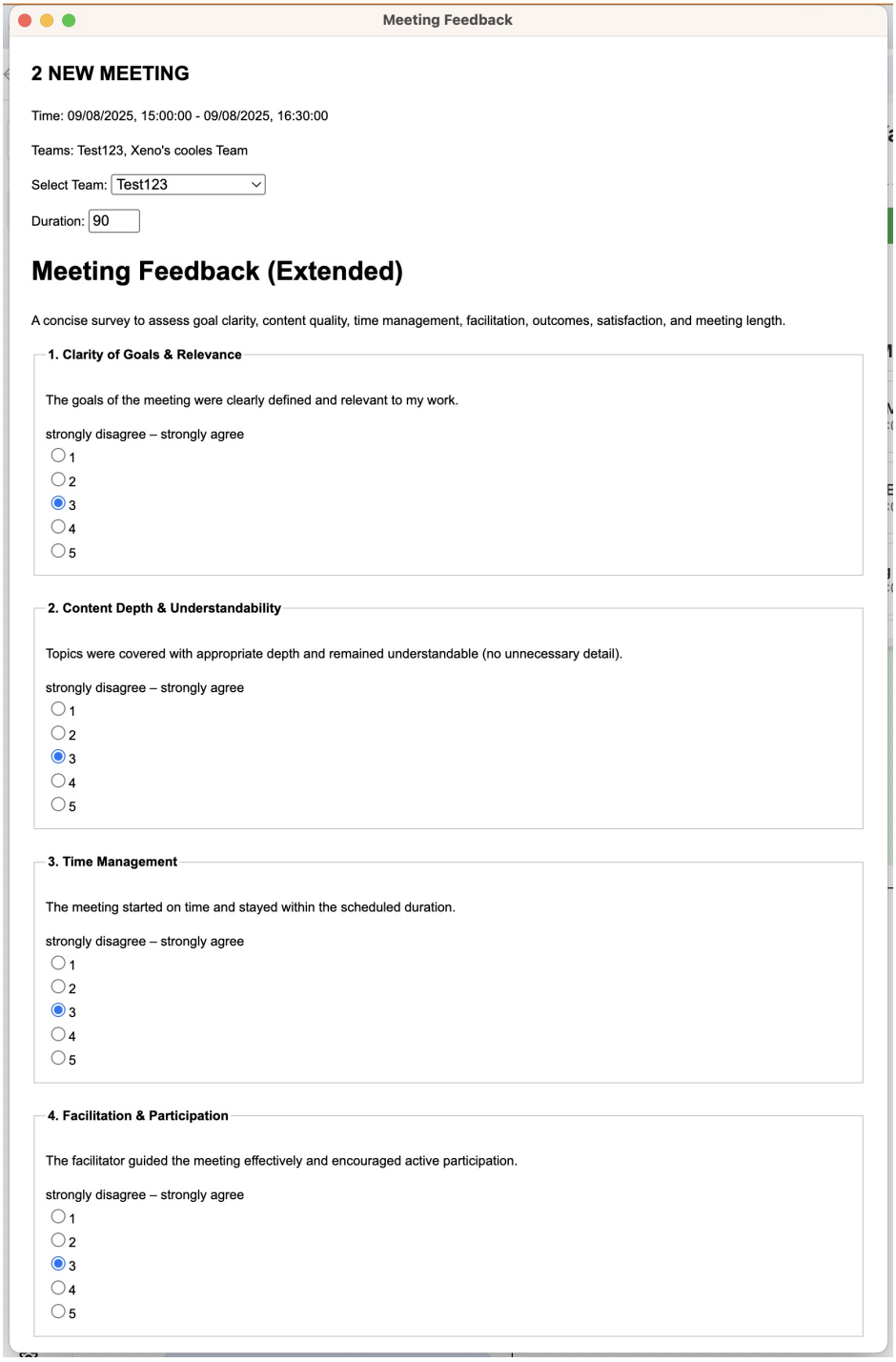

Abbildung 6.4: Feedbackformular zu Kalenderevent

```
    "max": 5,
    "label_max": "strongly agree",
    "step": 1,
    "default": 3
},
{
    "id": "q2",
    "type": "single-choice",
    "title": "2. Meeting Length",
    "prompt": "How would you rate the length of the meeting?",
    "required": true,
    "options": [
        { "value": "too-short", "label": "Too short" },
        { "value": "just-right", "label": "Just right" },
        { "value": "too-long", "label": "Too long" }
    ],
    "default": "just-right"
},
{
"id": "q3",
"type": "multiple-choice",
"title": "3. Meeting Highlights",
"prompt": "Which aspects of the meeting did you find valuable? (Sele
"required": false,
"options": [
    { "value": "clear-agenda", "label": "Clear agenda" },
    { "value": "useful-content", "label": "Useful content" },
    { "value": "good-discussion", "label": "Good discussion" },
    { "value": "action-items", "label": "Clear action items" }
],
"default": ["clear-agenda", "good-discussion"]
},
{
```

```
        "id": "q5",
        "type": "free-text",
        "title": "4. Additional Comments",
        "prompt": "Please share any additional feedback or suggestions."
        "required": false,
        "placeholder": "Type your feedback here...",
        "maxLength": 500
    }
    ]
    }
```

Vorteile

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

TODO: Sequenzdiagramm
TODO: Front End Anpassung

6.3.6 Health Companion

TODO: Health Kit API
TODO: Kommunikation
TODO: Benutzeroberfläche
TODO: Datenbankerweiterung

6.4 Deployment und CI/CD

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.

6.4.1 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 6.5 zeigt die einzelnen Schritte des automatisierten Build- und Deployment-Prozesses für das Backend (vom Quellcode-Checkout bis zum Neustart der Container).

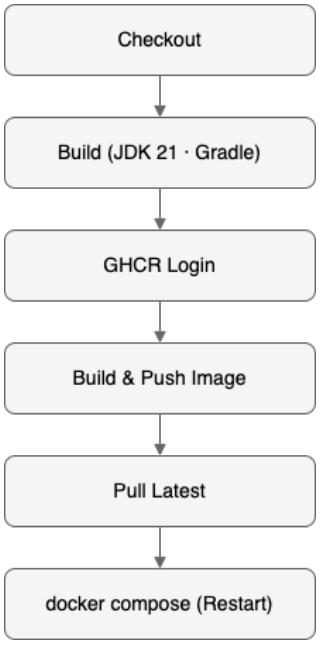

Abbildung 6.5: Deployment flow vom Backend

Analog dazu veranschaulicht Abbildung 6.6 den entsprechenden Ablauf für das Frontend, inklusive Linting, Next.js-Build und Bereitstellung des Docker-Images.

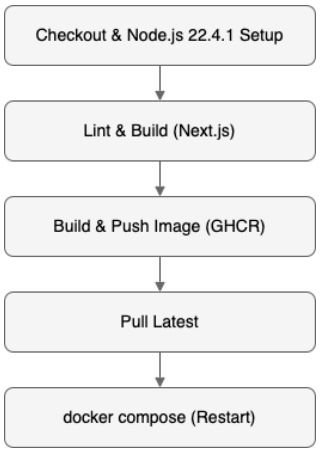

Abbildung 6.6: Deployment flow vom Frontend

6.4.2 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 6.7.

- **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.
- **Eclipse Mosquitto** - MQTT-Broker für die Echtzeitkommunikation mit Companion-Apps.

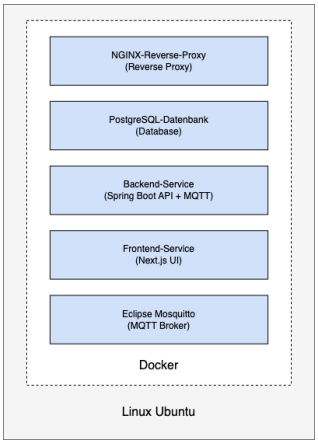

Abbildung 6.7: Systemübersicht der Linuxumgebung

6.4.3 NGINX-Proxykonfiguration

Der NGINX-Proxy übernimmt Routing und Zugriffskontrolle:

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

6.4.4 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.

6.4.5 MQTT-Broker-Konfiguration

Der **Eclipse Mosquitto**-Broker ist mit zwei Listenern konfiguriert:

- **TCP-Port 1883** für Backend-Services.
- **WebSocket-Port 9001** für Browser-Clients (z. B. die Chrome Extension).

Für den Prototypenbetrieb ist die Authentifizierung deaktiviert (`allow_anonymous true`).

6.4.6 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.

TODO XENO:
TODO: UML Deployment Diagramm

Kapitel 7

Evaluation

TODO XENO:
TODO: Hackathon
TODO: kleine evaluation vor projektende

7.1 Beantwortung der Fragestellung

TODO:

Kapitel 8

Diskussion

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] "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.

[8] "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.

[9] 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.

[10] "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.

[11] "Git analytics: Challenges, tools & key metrics", besucht am 9. Aug. 2025. Adresse: https://axify.io/blog/git-analytics.

[12] zachminers. "Introduction to viva insights", besucht am 9. Aug. 2025. Adresse: https://learn.microsoft.com/en-us/viva/insights/introduction.

[13] 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. 110 717, 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.

[14] 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/.

[15] 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. 1 149 367, 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/.

[16] 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.

[17] 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.

[18] 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.

[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.