Inhaltsverzeichnis

0.1 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.

0.1.1 Architektur und Integration

Dia Abbildung 1 zeigt den Aufbau des Yappi IntellJ IDEA Companion.

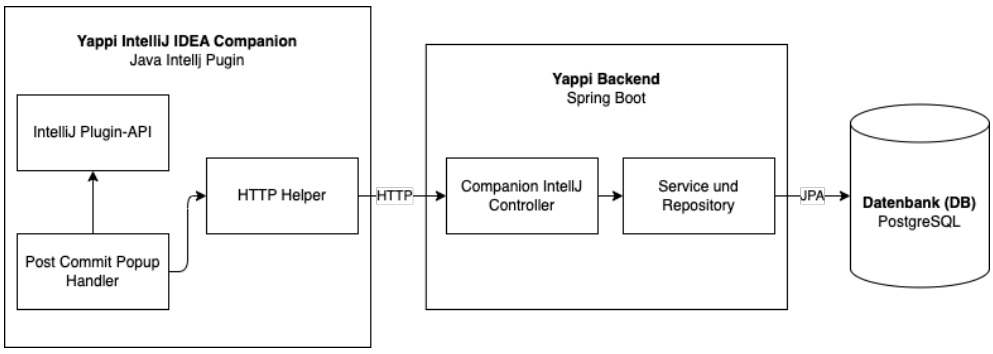

Abbildung 1: Architektur des IntelliJ IDEA Companion

Die Implementierung basiert auf den IntelliJ-Plugin-APIs. Ein (`PostCommitPopupHandler`) registriert sich auf erfolgreiche Commit-Events. Nach Bestätigung des Com-

mits 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.

0.1.2 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 2 zeigt den entsprechenden Menupunkt.

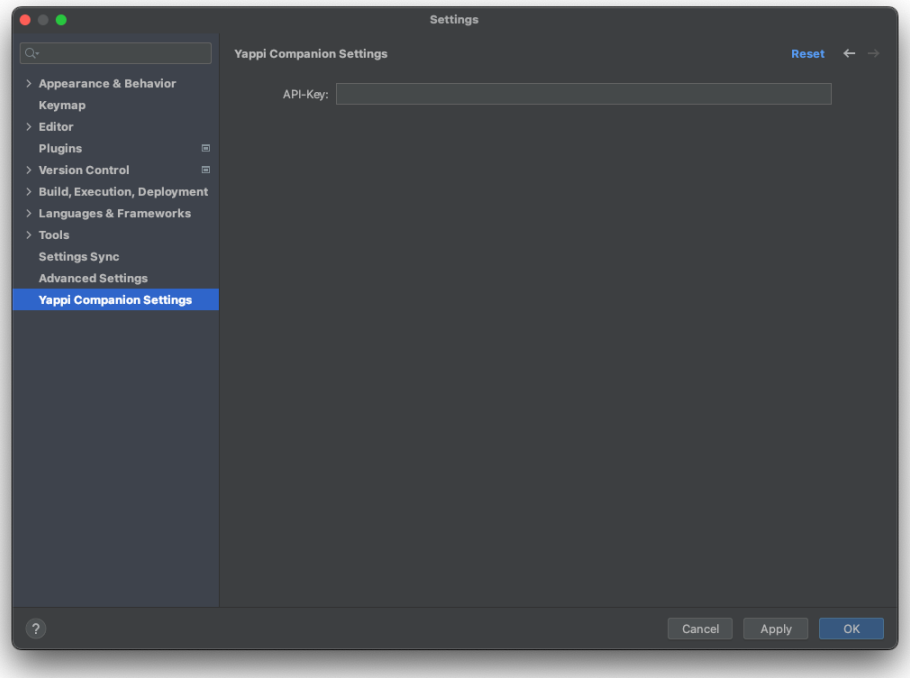

Abbildung 2: Eingabe des API-Keys in den IntelliJ Companion-Einstellungen

Der Key wird in der Klasse `CompanionStorage` mithilfe der IntelliJ-Persistenzmechanismen

gespeichert und beim nächsten Start automatisch geladen.

0.1.3 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

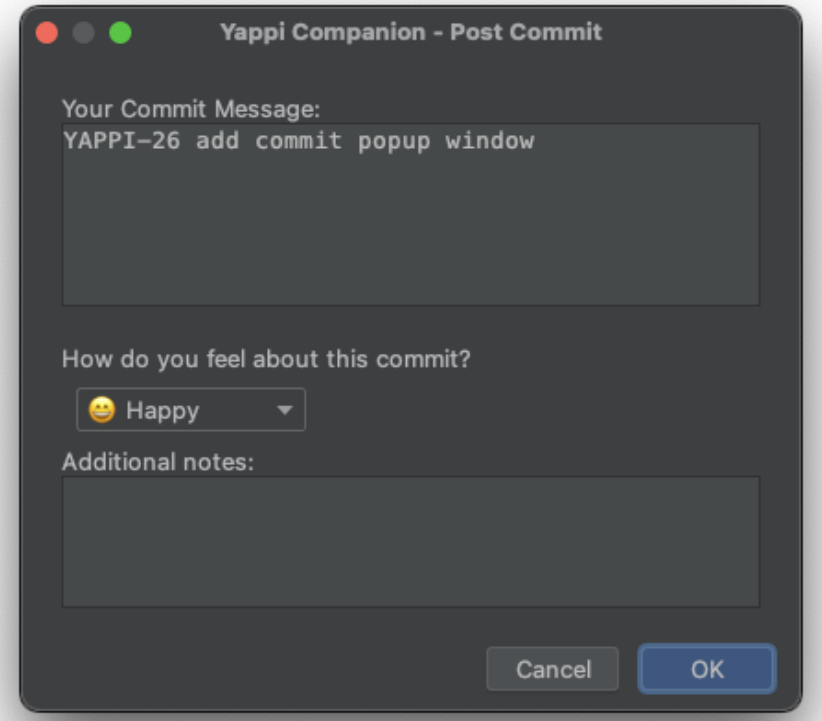

Abbildung 3: Post-Commit Dialog des IntelliJ Companions

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.

0.1.4 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 Zufriedenzeitsdaten und commit-spezifischen Kontextinformationen entgegengenommen werden, bereit. Der Endpoint ist volgendermassen aufgebaut.

`POST /companion/commit`
: Erfasst Zufriedenheitsdaten inklusive Kontexinformationen 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 Kontextinformationen persistiert. 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 Datenbankschama der Zufriedenheitsdaten zu integrieren.

Die Tabelle 1 zeigt den Aufbau der neuen Kontexttabelle `commit_context`.

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

Tabelle 1: 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.