Projektbericht: Organizr-API

Entwicklung des Projekts «Organizr-API» mit Scrum

Anforderungs- und Fachspezifikation

Das Ziel des Projekts war die Entwicklung der «Organizr-API», einer zentralen Schnittstelle zur Verwaltung persönlicher Organisationsdaten wie Kalendereinträge, Aufgaben und Notizen. Als primärer Client zur Demonstration der Funktionalität wurde ein KI-basierter Chatbot entwickelt, der natürliche Spracheingaben in API-Aufrufe umwandelt.

Die Spezifikationen umfassten die folgenden Kernbereiche:

- Benutzerverwaltung:
    - Ein Administrator-Account mit vollen Rechten (CRUD für alle User-Daten).
    - Unterstützung für mehrere Endnutzer, die sich per User-ID und API-Key authentifizieren.
    - Schnittstelle, um die API als Backend oder Erweiterung einer separaten App zu nutzen (Apps Router), wobei die App eigene Nutzerdaten mit der API verknüpfen kann.
    - Strikte Trennung der Nutzerdaten, sodass jeder User nur auf seine eigenen Einträge zugreifen kann.
- Kalender-, Task- und Notizverwaltung:
    - Für alle drei Module (Kalender, Tasks, Notizen) wurden vollständige CRUD-Operationen (Create, Read, Update, Delete) implementiert.
    - Einträge können mit einem Titel, einer optionalen Beschreibung und Tags versehen werden.
    - Kalendereinträge und Tasks unterstützen Wiederholungsregeln nach dem iCalendar RRULE-Standard, auch im Querying.
    - Es wurden flexible Such- und Filterfunktionen implementiert, die das Abfragen von Einträgen nach Text, Zeiträumen (auch bei RRULE Einträgen), Status (bei Tasks) und Tags ermöglichen.
- Tech-Stack:
    - Backend: Python mit dem FastAPI-Framework.
    - Datenbank: MySQL zur persistenten Speicherung der Daten.
    - Deployment: Das gesamte Projekt wurde mit Docker containerisiert und lässt sich über Docker-Compose einfach ausrollen. Eine Demo der API und des Bots laufen auf einem Debian 13 VPS.

- Telegram Chatbot:
    - Der Bot nutzt die Telegram API und KI-Modelle (LLM, STT), um per Function-Calling mit der Organizr-API zu interagieren. Dies ermöglicht es dem Endnutzer, seine Daten in natürlicher Sprache in Form von Text und Ton zu verwalten.

Organisation und Vorgehen

Das Projekt wurde agil nach den Prinzipien von Scrum und der iterativen Planung entwickelt. Die zur Verfügung stehende Zeit wurde in Sprints aufgeteilt, wobei jede Doppellektion einem Sprint entsprach.

Die Entwicklung erfolgte schrittweise und baute auf den vorherigen Ergebnissen auf:

- Sprint 1: Zuerst wurden die Grundlagen geschaffen, darunter das GitHub-Repo, das Datenbankschema und die User-API für die Benutzerverwaltung.
- Sprint 2 und 3: Danach wurde die Kalender-API implementiert, zuerst mit den Kernfunktionen und danach erweitert um die komplexe Logik für wiederkehrende Termine (RRULE) und erweitertes Querying.
- Sprint 4: Basierend auf dem Code der Kalender-API konnten die Router für Tasks und Notizen sehr effizient umgesetzt werden, da die Funktionalität ähnlich war.
- Sprint 5 und Abschluss: Zum Schluss wurde der Telegram-Bot als Client für die API entwickelt, um die Funktionalität zu demonstrieren und eine benutzerfreundliche Interaktion zu ermöglichen.

Die wöchentlichen Fortschrittsberichte dienten dabei als Sprint Review, um den aktuellen Stand zu dokumentieren und die nächsten Schritte zu planen. Für die Versionskontrolle und das Projektmanagement wurde GitHub genutzt. Durch GitHub Actions wurden zudem CI/CD-Prozesse für automatisierte Code-Analyse und das Bauen von Docker-Images eingerichtet.

Erreichtes

Das Projekt wurde erfolgreich und vollumfänglich umgesetzt. Alle im Projektantrag definierten Ziele wurden erreicht, und in mehreren Bereichen konnte der ursprüngliche Plan sogar übertroffen werden. Die wesentlichen Erfolge des Projekts umfassen:

- Voll funktionsfähige API: Alle geplanten Endpunkte für die Verwaltung von Kalendereinträgen, Tasks und Notizen sind implementiert und funktionieren stabil. Dies beinhaltet auch die komplexeren Features wie die Expansion von RRULE-Einträgen für die korrekte Abfrage wiederkehrender Ereignisse.

- Umfassender Telegram-Bot: Der KI-Bot wurde erfolgreich entwickelt und kann, entgegen der ursprünglichen, vorsichtigen Planung, nicht nur Notizen, sondern den vollen Funktionsumfang der API (Kalender, Tasks, Notizen) über natürliche Sprache steuern. Der Bot wickelt dabei die Benutzer-Authentifizierung im Hintergrund ab und dient als primärer Client.
- Robuster technischer Unterbau: Die Anwendung ist vollständig mit Docker und Docker-Compose containerisiert, was ein einfaches und reproduzierbares Deployment ermöglicht. Die Code-Struktur ist klar in die Hauptanwendung (/app), den Bot (/bot), die Docker-Konfiguration (/docker) und die Tests (/unit-tests) unterteilt.
- Code-Qualität und Automatisierung: Über die ursprüngliche Planung hinaus wurden erweiterte Entwicklungspraktiken umgesetzt. Mittels GitHub Actions wird bei jedem Push eine statische Code-Analyse mit SonarQube durchgeführt (sonarqube-sca.yml) und bei neuen Releases werden automatisch Docker-Images gebaut und auf GHCR veröffentlicht (ghcr-release.yml).

Fazit und Ausblick

Das Projekt ist funktional vollständig. Das einzig womöglich fehlende sind Unit-Tests für den Bot und zusätzlich eine Erweiterung der bisher vorhandenen Unit-Tests. Diese Punkte gehören jedoch nicht zum ursprünglich geplanten Umfang und sind somit nur im Sinne der Vollständigkeit, jedoch nicht der Funktion oder der Projektplanung erforderlich.

Testbericht

Die Teststrategie entwickelte sich im Laufe des Projekts von rein manuellen Tests zu einer Kombination aus manuellen und automatisierten Verfahren, um eine hohe Zuverlässigkeit der API zu gewährleisten.

- Manuelles Testing: Während der gesamten Entwicklungsphase wurde die API kontinuierlich manuell getestet. Hierfür kamen die in FastAPI integrierte Swagger UI und der API-Client Insomnia (ähnlich Postman) zum Einsatz. Dies ermöglichte eine schnelle Überprüfung der Endpunkte nach jeder Code-Änderung.
- Automatisiertes Unit-Testing: Um die Qualität nachhaltig zu sichern, wurde eine umfassende Suite von Unit-Tests mit pytest geschrieben. Diese Tests werden durch die GitHub-Actions-Pipeline bei jedem Commit automatisch ausgeführt, um Fehler zu erkennen. Die Testabdeckung wird dabei ebenfalls erfasst und an SonarQube übermittelt, wie in der pytest.ini-Konfiguration beziehungsweise den Grafiken in der README-Datei ersichtlich ist.

- Integrationstests: Das Deployment des gesamten Stacks (API, Bot, Datenbank) via Docker-Compose auf einem separaten Linux-Server diente als praxisnaher Integrationstest und stellte sicher, dass die einzelnen Komponenten auch im Zusammenspiel reibungslos funktionieren.
- Test durch Nutzung: Der Bot selbst wurde durch Interaktion direkt in Telegram getestet. Zudem wurde durch regelmässige Nutzung des Bots mit verschiedensten Szenarien auch automatisch die API ein weiteres Mal getestet, dieses Mal mit echten Nutzungsszenarien.

Fehler die während des manuellen Testing, meist direkt während dem Schreiben von Code, festgestellt wurden sind umgehend behoben wurden. Zudem ergeben auch die Unit-Tests für die getesteten Bereiche des Codes (API-Code, ohne Bot-Code) mittlerweile nur positive Resultate (ersichtlich in GitHub Actions Logs), wobei diese Resultate zuerst durch einige Bugfixes hergeführt werden mussten. Der Bot ist schon seit frühen Entwicklungsphasen im automatischen Deployment mit der API auf dem Server enthalten und wurde dementsprechend während dem ganzen Entwicklungsprozess genutzt. Allfällige Fehler des Bots oder der API sind seit frühen Phasen selten aufgetreten, da sie umgehend behoben wurden.

Materialien

Das gesamte Projekt mitsamt Erklärung und Images ist unter https://github.com/cwhde/organizr-api öffentlich einsehbar.

Zudem läuft eine Version des Bots unter https://t.me/organizr_bot und ist bis auf weiteres öffentlich verfügbar. Dieser Bot ist gleichzeitig mit einer Instanz der Organizr-API auf einem privaten VPS installiert, wobei die API-Instanz unter https://api.organizr.ch/ zu erreichen ist.

Arbeitsaufteilung

Sämtliche relevanten und grundlegenden Features, die in der originalen Projektspezifikation enthalten waren, wurden rein während der Unterrichtszeit erarbeitet. Nur erweiterte Funktionen wurden teils ausserhalb der Unterrichtszeit erarbeitet, darunter:

- Sämtliche Unit-Tests
- GitHub Code-Analysis-Workflow
- GitHub Release-Workflow
- Sprachnachrichtenkompatibilität des Telegram-Bot