

Einzelreflexion Vincent Karmanoczki

Rolle und Einordnung in den SE-Prozess

Im Projekt habe ich überwiegend die Rolle eines Entwicklers eingenommen und sowohl im Frontend als auch - im späteren Projektverlauf - im Backend gearbeitet. Meine Hauptaufgaben lagen in den Phasen Entwurf und Implementierung, mit zusätzlichen Berührungspunkten zum Anforderungsmanagement, zur Architektur sowie zur Qualitätssicherung.

Unsere Arbeitsweise folgte grundsätzlich einem iterativen Vorgehen, wobei GitHub Projects zur Aufgaben- und Backlog-Verwaltung genutzt wurde. In frühen Projektphasen war diese Arbeitsweise jedoch noch wenig formalisiert, da es zunächst keine klaren Sprint Plannings gab und Aufgaben häufig erst im Weekly neu verteilt wurden. Dies wirkte sich insbesondere auf die Klarheit von Anforderungen und Schnittstellen aus.

Zu Projektbeginn arbeitete ich primär im Frontend und setzte dort auf Basis der vorhandenen Wireframes konkrete Views um. Im weiteren Verlauf eignete ich mir zusätzlich Kenntnisse im Backend (Django) an, um Abhängigkeiten zwischen Frontend und Backend besser auflösen und Features end-to-end umsetzen zu können. Da ich zu Beginn keine Vorerfahrung mit den eingesetzten Technologien hatte, bestand eine zusätzliche Herausforderung darin, mich parallel in neue Frameworks, Programmiersprachen und Projektstrukturen einzuarbeiten.

Eigene Beiträge und Lösungswege

1. Umsetzung zentraler Frontend-Views im Kontext unklarer Anforderungen

Zu meinen frühen Aufgaben zählte die Implementierung mehrerer Frontend-Views in Vue.js, unter anderem des `EditQuizView` und `EditQuestionView` ([Issue #64](#)). Diese Aufgaben waren eng mit noch nicht vollständig geklärten Backend-Schnittstellen und Anforderungen verknüpft.

Da zu diesem Zeitpunkt weder die API-Endpunkte final definiert noch ausreichend dokumentiert waren, fehlte eine klare einheitliche Schnittstelle zwischen Client und Server. Dies erschwerte die Umsetzung funktionaler Anforderungen erheblich, da unklar war, welche Daten persistent gespeichert werden und welche lediglich als temporäre Platzhalter dienen sollten. Infolgedessen mussten die Views im weiteren Projektverlauf mehrfach angepasst werden, etwa beim Routing oder nach Fertigstellung der API-Endpunkte (z. B. [Commit f10e83a](#)).

Aus Sicht des Software-Engineering-Prozesses lässt sich diese Phase der frühen Entwurfs- und Implementierungsphase zuordnen, in der der Fokus primär auf einem lauffähigen Produkt lag. Aspekte wie saubere Schnittstellendefinitionen, Dokumentation und Code-Qualität traten zunächst in den Hintergrund. Für mich wurde hier besonders deutlich, wie wichtig frühzeitiges Anforderungsmanagement sowie explizite API-Endpunkte für eine nachhaltige und wartbare

Implementierung sind.

2. Backend-Einarbeitung und Architekturverbesserung

Im weiteren Projektverlauf zeigte sich, dass viele Features ohne Anpassungen im Backend nur eingeschränkt umsetzbar waren. Da im Team insgesamt wenig Erfahrung mit Django vorhanden war, habe ich begonnen, mich selbstständig in das Backend einzuarbeiten, um Abhängigkeiten zwischen Frontend und Backend besser auflösen zu können.

Ein konkretes Beispiel ist ein Fehler bei der Quiz-Erstellung, bei dem Antwortmöglichkeiten nicht korrekt gespeichert wurden ([Issue #112](#)). Der initiale Bugfix erfolgte in einem eigenen Branch ([bugfix/create-quiz](#)). Im weiteren Verlauf stellte sich jedoch heraus, dass das zugrundeliegende Architekturkonzept - eine separate API-Anfrage pro Frage - nicht performant war. Auf Basis dieser Erkenntnis habe ich die Lösung refaktoriert und die gesamte Quiz-Erstellung in einer einzigen API-Anfrage gebündelt ([bugfix/create-quiz-single-api-call](#)).

Dieses Vorgehen lässt sich als inkrementelle Architekturverbesserung im Rahmen iterativer Entwicklung einordnen. Durch die initiale Implementierung wurde ein funktionierendes Feature geschaffen, das anschließend auf Basis von Performance- und Wartbarkeitsanforderungen weiterentwickelt wurde.

Durch diese Einarbeitung konnte ich im weiteren Verlauf zusätzliche Backend-Features übernehmen, unter anderem: - Leaderboard-Funktionalität ([Issue #165](#), [feature/streak-leaderboard](#)) - Streak-Tracking ([feature/streak-date](#)) - Bewertung von Quiz-Ergebnissen ([feature/calculate-iq-points](#)) - Backend-Teil des Freundesystems ([feature/backend-friends](#))

Die technische Umsetzung dieser Features wurde jeweils im Weekly abgestimmt, um sicherzustellen, dass sie mit der bestehenden Client-Server-Architektur vereinbar sind.

3. Refactoring, Code-Qualität und Korrektur früher Architekturentscheidungen

Im späteren Projektverlauf wurde im Team reflektiert, dass sowohl Code-Struktur als auch Dokumentation und Datenmodell an Qualität verloren hatten. In diesem Zusammenhang übernahm ich die Aufgabe, Backend-Code zu dokumentieren und gemäß Clean-Code- und PEP8-Prinzipien zu refactoren ([Issue #228](#)).

Dabei zeigte sich, dass zuvor nicht konsequent nach dem YAGNI-Prinzip gearbeitet worden war, was zu ungenutzten Datenbankfeldern und unnötiger Komplexität führte. Diese Altlasten wurden teilweise bereinigt. Parallel erfolgte die Umstellung der Datenbank von SQLite auf MySQL, wobei die Datenbankdatei nicht länger im Repository versioniert wurde - eine wichtige Verbesserung im Bereich Versions- und Konfigurationsmanagement.

Eine weitere zentrale Korrektur betraf eine frühe Architekturentscheidung, bei der dem Client zu stark vertraut wurde. Die fehlende serverseitige Validierung stellte ein Sicherheits- und Fairnessproblem dar, insbesondere im Zusammenhang mit dem Leaderboard. Die nachträgliche

Einführung robuster Validierungsmechanismen (z. B. [feature/session-id](#)) war funktional notwendig, jedoch aufwendig, da sie tief in bestehende Logik eingriff.

Zusätzlich wurden UI-Inkonsistenzen sowie veraltete Datenflüsse bereinigt, die aus verworfenen Architekturkonzepten resultierten ([bugfix/fill-quiz-overview-fields](#)).

Kritische Bewertung und Lernerfahrungen

Rückblickend hätte ich insbesondere in frühen Projektphasen stärker auf eine explizite Klärung von Anforderungen und Schnittstellen bestehen sollen. Die fehlende Initialdokumentation von API-Endpunkten sowie unklare Task-Abgrenzungen führten zu Mehraufwand und unnötigen Überschneidungen.

Zudem habe ich mehrfach vollständige Features allein umgesetzt, einschließlich Frontend, Backend und API-Logik (z. B. [#165](#), [#250](#), [#294](#)). Zwar ermöglichte dies eine geschlossene Umsetzung im Sinne eines vertikalen Feature-Schnitts, stellte jedoch eine hohe persönliche Belastung dar und wäre durch frühzeitigere Abstimmung vermeidbar gewesen.

Positiv hervorzuheben ist meine deutliche fachliche Weiterentwicklung in den Bereichen Backend-Architekturen, Refactoring, Clean Code, Versionsmanagement sowie im Umgang mit iterativen Entwicklungsprozessen. Für zukünftige Projekte nehme ich mir vor, Anforderungen früher zu klären, Aufgaben klarer abzugrenzen und kollaborative Entwicklungspraktiken konsequenter zu nutzen.