BÄRNHÄCKT

# «Talus Devs»
# «Clever planen, bewusster essen»

**Technische Informationen für die Jury**

**Technische Informationen für die Jury**

Mit FoodQuest wird das alltägliche Kochen zu einem spielerischen Abenteuer. Die App kombiniert gesunde Ernährung mit Gamification und sorgt dafür, dass die Suche nach Rezepten nicht mehr langweilig und mühsam, sondern motivierend und spannend ist. Damit wird aus einer alltäglichen Pflicht eine Herausforderung, die Spaß macht und gleichzeitig die Kochfähigkeiten stärkt.

**Aktueller Stand des Sourcecodes**
Github Repository: https://github.com/MeiLing90/food_quest_talus_dev

**Ausgangslage**
Viele Menschen möchten gesund und nachhaltig essen, stoßen dabei jedoch im Alltag auf verschiedene Hindernisse. Die Rezeptsuche gestaltet sich oft langweilig und zeitintensiv, und in stressigen Phasen fehlt die Motivation, etwas Neues auszuprobieren. Genau hier setzt FoodQuest an. Der Fokus lag von Beginn an auf der Verbindung von gesunder Ernährung und Gamification. Mit spielerischen Elementen wie Quests, Punkten und Highscores wird das Kochen motivierend gestaltet und die Auswahl von Rezepten vereinfacht. Ein wesentlicher Grundsatzentscheid war, FoodQuest als Web-App zu entwickeln, um plattformübergreifend verfügbar zu sein und damit eine breite Nutzerschaft zu erreichen. Zusätzlich haben wir uns entschieden, Rezepte nach Gesundheitsscore und Schwierigkeitsgrad zu kuratieren, um die Qualität sicherzustellen und gleichzeitig unterschiedliche Bedürfnisse abzudecken.

**Technischer Aufbau**
Der technische Aufbau von FoodQuest basiert auf einem klar strukturierten modernen Web-Stack. Im Frontend verwenden wir Vue 3 mit Single File Components, wobei einige Komponenten TypeScript einsetzen. Für den Entwicklungsworkflow kommt der Vite Dev Server und Build Tool zum Einsatz, ergänzt durch Vue Router 4, das die wichtigsten Routen abbildet (Startseite → FoodTinder/Swiper, Detailansicht /food/:id und /quests). Für das UI-Design nutzen wir Vuetify 3 in Kombination mit @mdi/font Icons, was eine schnelle und konsistente Gestaltung ermöglicht. Für einfache Prototyping-Fälle speichern wir Auswahlzustände lokal über localStorage.
Im Backend setzen wir auf Node.js 20 (Alpine) mit Express 4 und cors. Zum Prototyping wird eine einfache JSON-Store-Struktur genutzt (z. B. server/data/*.json). Das Backend stellt Endpunkte wie GET /api/food, GET /api/quests, POST /api/quests/:id/progress oder POST /api/events/recipe-cooked bereit. In der Entwicklungsumgebung nutzen wir Nodemon, um Hot-Reloading zu ermöglichen.
Für DevOps und Runtime verwenden wir Docker Compose. Im Entwicklungsprofil laufen Web (Vite) und Server (Express) in getrennten Containern, während im Produktionsprofil ein Multi-Stage-Build den Vite-Output ins Express-Projekt einbettet, sodass sowohl API als auch statische Inhalte aus einem Container ausgeliefert werden. In der Entwicklungsumgebung sorgt ein Vite-Proxy dafür, dass /api/*-Aufrufe an den Server (Port 5000) weitergeleitet werden.

**Implementation**
Besonders hervorzuheben ist die klare Trennung zwischen Frontend- und Backend-Containern im Entwicklungsprofil, was schnelle Iterationen erlaubt, sowie die konsistente Produktion mit nur einem Container. Auch die Nutzung von Vite bringt Vorteile durch schnelle Build- und Hot-Reload-Zeiten. Im Prototyping-Bereich erleichtert die JSON-Store-Lösung den schnellen Aufbau ohne Datenbank-Overhead. Mit geringem Setup lässt sich eine spielerische und reaktive App umsetzen, die dennoch erweiterbar bleibt. Perspektivisch lässt sich die Architektur problemlos erweitern. Denkbar ist etwa der Einsatz von Pinia für einen gemeinsamen State-Store im Frontend oder die Integration von Tools wie ESLint/Prettier für mehr Code-Qualität. Eine echte Datenbank ist aktuell noch nicht im Einsatz, könnte aber in Zukunft folgen. Bei der Implementation lag ein besonderer Fokus auf der einfachen Bedienung. Das Swipe-Interface wurde bewusst minimalistisch umgesetzt, um die Einstiegshürde so niedrig wie möglich zu halten. Zudem wurde die Gamification-Logik modular entwickelt, sodass neue Quests oder Belohnungen flexibel ergänzt werden können. Besonders hervorzuheben ist die dynamische Rezeptkurierung: Rezepte werden nicht nur

nach Schwierigkeitsgrad sortiert, sondern auch nach einem Gesundheitsscore bewertet, der Faktoren wie Nährwerte, Ausgewogenheit und Nachhaltigkeit berücksichtigt. Ein weiteres Highlight sind die Echtzeit-Score-Updates, die den Nutzer direkt spüren lassen, wie sich ihre Kochaktionen auf den Highscore auswirken.

**Abgrenzung und offene Punkte**
Um die Komplexität der ersten Version gering zu halten, haben wir bewusst auf einige Features verzichtet. Dazu gehört die automatische Erstellung von Einkaufslisten, da der Fokus klar auf dem Spielerlebnis liegen soll. Ebenso haben wir keine externen Rezeptdatenbanken integriert, um die Konsistenz und Qualität der Inhalte zu sichern. Social Features wie Freundes-Challenges wurden für die erste Version ebenfalls ausgeklammert, um die Einzelspieler-Erfahrung in den Mittelpunkt zu stellen. Offene Punkte für zukünftige Erweiterungen sind die Einführung von Multiplayer-Features für gemeinsame Koch-Challenges, die Erweiterung des Nachhaltigkeitsscores mit $CO_2$-Bewertungen sowie die Entwicklung nativer App-Versionen für eine noch bessere mobile Nutzererfahrung.