Innovation through Spec-Driven Development

ITMC Innovationsforum Wintersemester 2025/2026

Marvin Kruse

Über mich

Marvin Kruse
Technical Lead bei OTTO

Background
- Bei OTTO seit Juni 2024
- Vorher: Statista & AIRBUS
- Schwerpunkte: Tech Strategy & Talent Development
- Studium: Dipl. Wirtschaftsinformatik (Uni Hamburg)

Agenda

1. Was ist Spec-Driven Development?
2. Innovation durch Spec-Driven Development
3. Key Takeaways

Spec-Driven Development erklärt

Frage: Wer hat schonmal "Vibe Coding" gemacht?

Was ist "Vibe Coding"?

— 💡 Direkt loscoden ohne formale Spezifikation

— 🚙 Details kann man "unterwegs" klären

— 🚀 Sehr viel Output in kurzer Zeit

Herausforderungen beim Vibe Coding

— 🎲 Sehr viel Code, aber oft unzureichende Qualität

— 📏 Standards & Best Practices werden ignoriert

— 📦 Riesige, unwartbare PRs

— 🚫 Schwierig zu skalieren

Die Lösung: Spec-Driven Development

Was ist Spec-Driven Development?

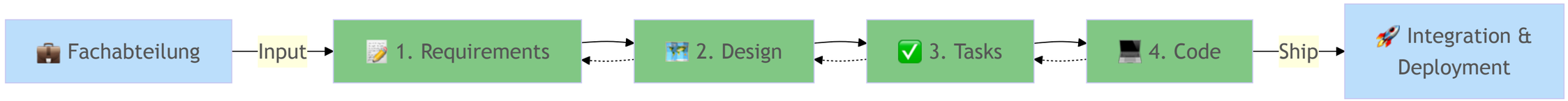
-  **Formale Spezifikation** vor der Implementierung
- 🎯 Die Spezifikation ist **Teil der Anwendung** - sie lebt nicht außerhalb, z.B. in einem Wiki
- 🤝 Die Anforderungen existieren an **einer Stelle**
- ✅ **Validierung, Testing & Code-Generierung** anhand der Spezifikation

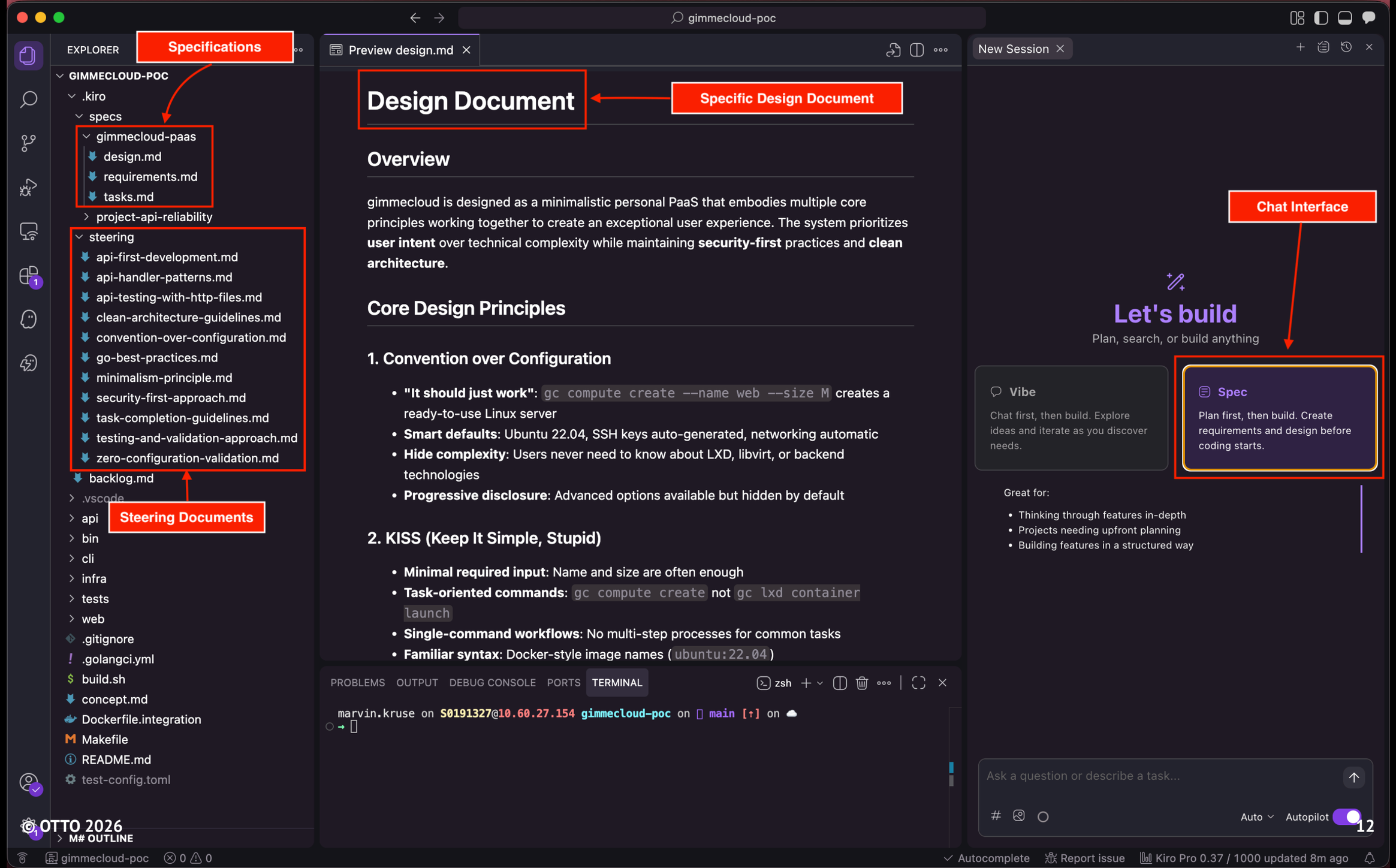
Der Spec-Driven Flow (anhand von Kiro)

1. **Requirements** Strukturierte Anforderungen
2. **Design** Architektur & technisches Design
3. **Tasks** Konkrete Umsetzungsschritte
4. **Code** Implementierung

Steering Documents Anweisungen, die für **alle** Schritte relevant sind

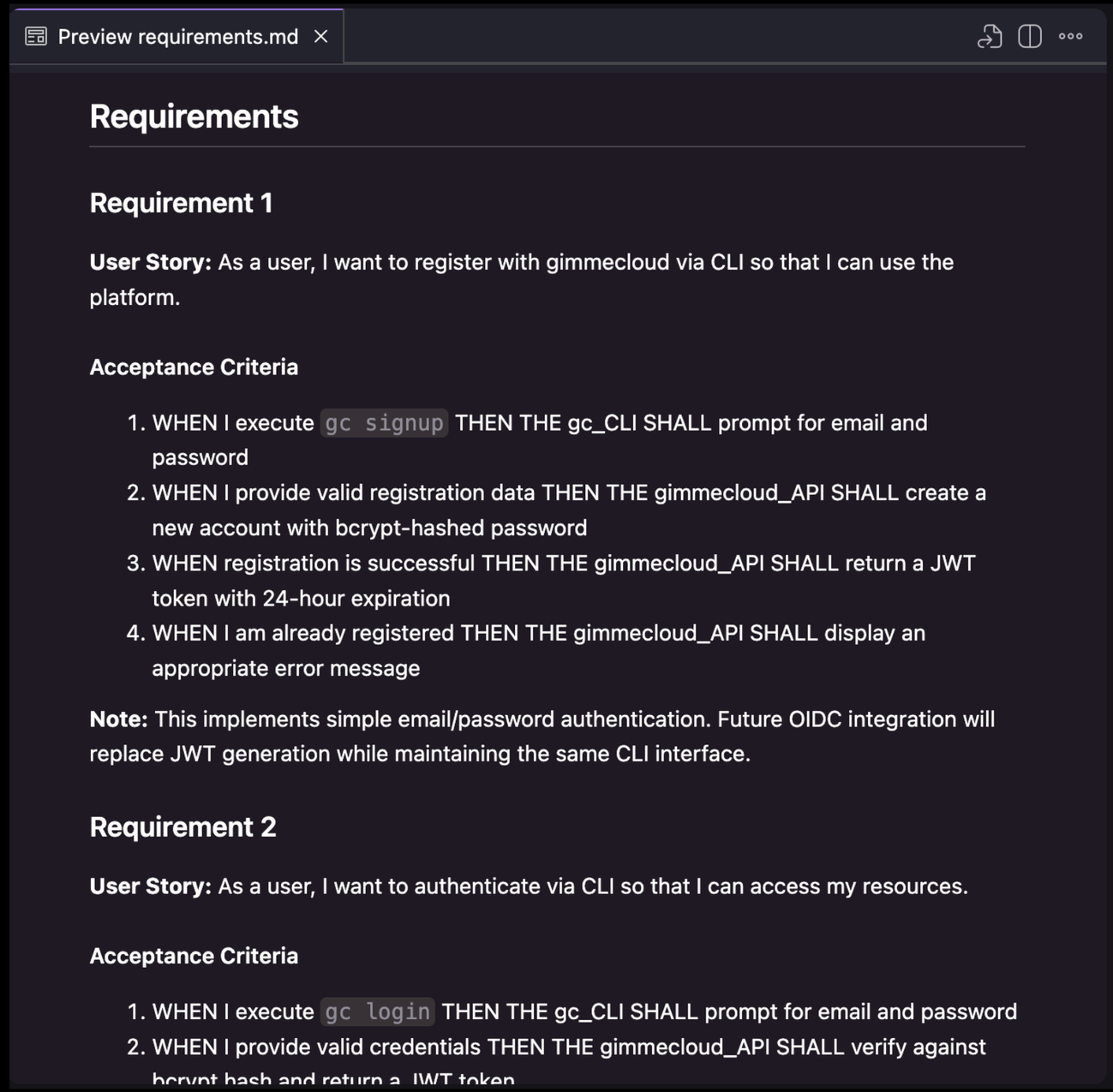
Der Spec-Driven Flow (anhand von Kiro) - Cont'd




gimmecloud-poc
EXPLORER
Specifications
GIMMECLOUD-POC
.kiro
specs
gimmecloud-paas
design.md
requirements.md
tasks.md
project-api-reliability
steering
api-first-development.md
api-handler-patterns.md
api-testing-with-http-files.md
clean-architecture-guidelines.md
convention-over-configuration.md
go-best-practices.md
minimalism-principle.md
security-first-approach.md
task-completion-guidelines.md
testing-and-validation-approach.md
zero-configuration-validation.md
backlog.md
.vscode
api
bin
cli
infra
tests
web
.gitignore
.golangci.yml
build.sh
concept.md
Dockerfile.integration
Makefile
README.md
test-config.toml
Steering Documents
M# OUTLINE
Preview design.md
Design Document
Specific Design Document
Overview
gimmecloud is designed as a minimalistic personal PaaS that embodies multiple core principles working together to create an exceptional user experience. The system prioritizes user intent over technical complexity while maintaining security-first practices and clean architecture.
Core Design Principles
1. Convention over Configuration
"It should just work": gc compute create --name web --size M creates a ready-to-use Linux server
Smart defaults: Ubuntu 22.04, SSH keys auto-generated, networking automatic
Hide complexity: Users never need to know about LXD, libvirt, or backend technologies
Progressive disclosure: Advanced options available but hidden by default
2. KISS (Keep It Simple, Stupid)
Minimal required input: Name and size are often enough
Task-oriented commands: gc compute create not gc lxd container launch
Single-command workflows: No multi-step processes for common tasks
Familiar syntax: Docker-style image names (ubuntu:22.04)
PROBLEMS
OUTPUT
DEBUG CONSOLE
PORTS
TERMINAL
zsh
marvin.kruse on S0191327@10.60.27.154 gimmecloud-poc on main [↑] on
New Session
Chat Interface
Let's build
Plan, search, or build anything
Vibe
Chat first, then build. Explore ideas and iterate as you discover needs.
Spec
Plan first, then build. Create requirements and design before coding starts.
Great for:
Thinking through features in-depth
Projects needing upfront planning
Building features in a structured way
Ask a question or describe a task...
Auto
Autopilot
OTTO 2026
12
gimmecloud-poc
0 0
Autocomplete
Report issue
Kiro Pro 0.37 / 1000 updated 8m ago

Phase 1: Requirements

- Given, When, Then - Notation
- Strukturierte Anforderungen
- (JIRA)-Tickets folgen oft dem Format



Phase 2: Design

— APIs
— Architektur
— Sprachen, Frameworks
— Datenstrukturen
— Security Aspekte

Preview design.md

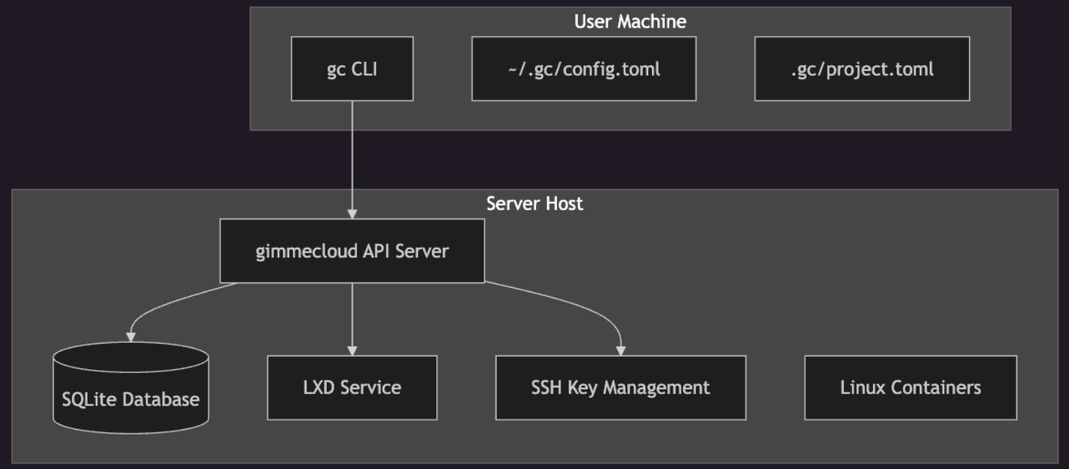
Preview design.md

…ture creep: Focus on home user needs, not enterprise complexity

- **Simple data structures**: Prefer simple types over complex hierarchies
- **Delete before adding**: Look for code to remove before adding new code

Architecture

High-Level Architecture

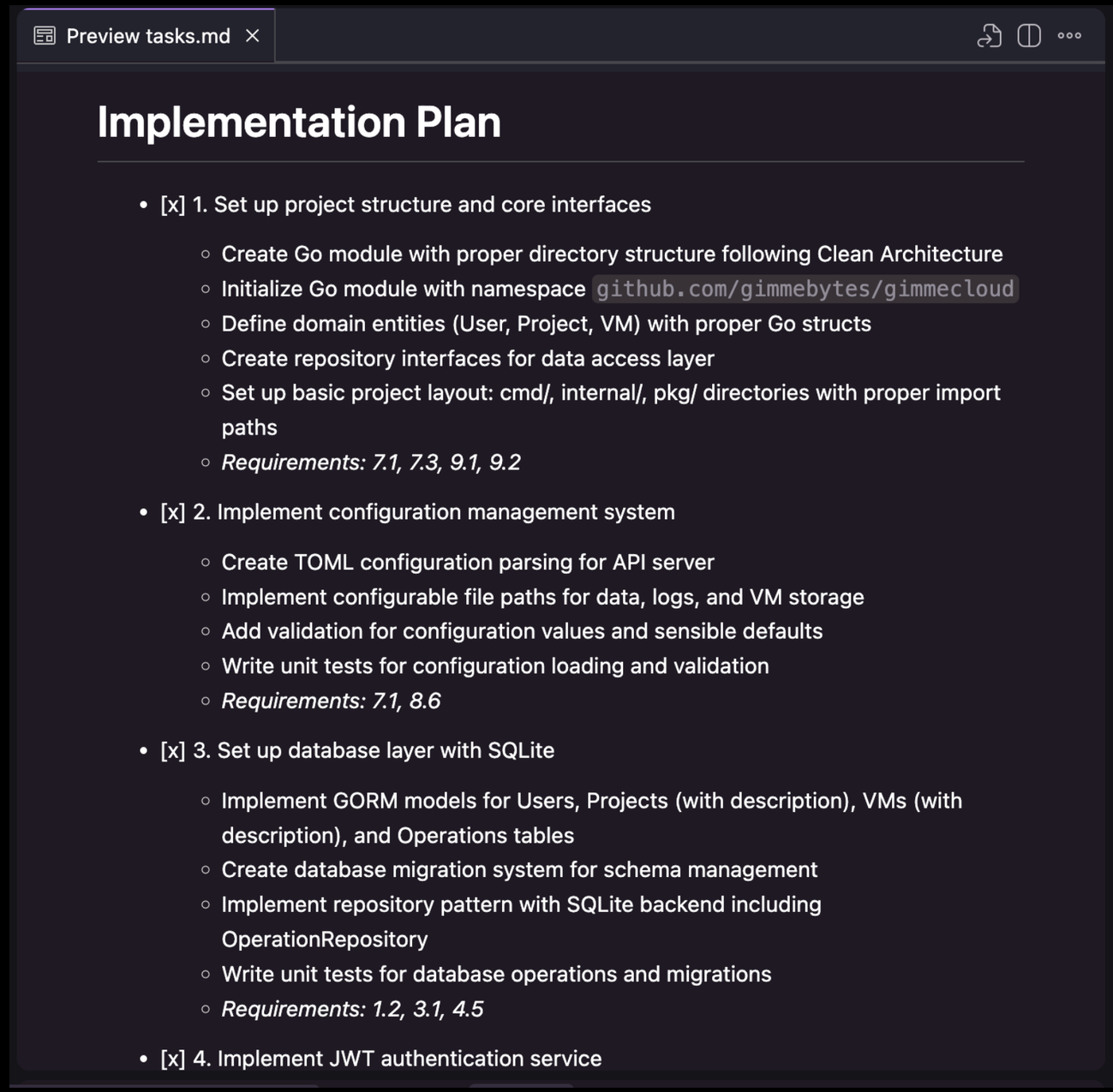


Component Overview

1. **CLI Client (`gc`)** - Go-based command-line interface
2. **API Server** - Go REST API server using Gin framework
3. **Database** - SQLite for data persistence
4. **Container Management** - LXD integration for Linux containers
5. **SSH Management** - Automated SSH key generation and injection

Phase 3: Tasks

- Isolierte (Teil)-Aufgaben
- Unabhängig testbar
- Jede Aufgabe hat einen eindeutigen Bezug zu den gestellten Anforderungen



Phase 4: Code

Spec-Driven vs. Vibe Coding

Aspekt	Vibe Coding	Spec-Driven
Start	Schnell	Langsamer
Kommunikation	Informell	Formalisiert
Skalierung[1]	Schwierig	Gut skalierbar
Breaking Changes	Häufig	Seltener
Parallelarbeit	Eingeschränkt	Optimal

Wann was?
→ Vibe Coding: Prototypen, kleine Teams, schnelle Experimente
→ Spec-Driven: Skalierung, mehrere Teams, Enterprise-Kontext

[1] Im Unternehmenskontext

Innovation durch Spec-Driven Development in der Praxis

Case 1
Fehlende Expertise ausgleichen

"Mit Frontends kenne ich mich nicht aus..."

Case 1: Fehlende Expertise ausgleichen

Ein Team soll ein neues Feature entwickeln, aber...

- 🎯 Team hat primär Backend-Expertise
- 📱 Feature benötigt aber Frontend-Expertise
- 🏝️ Tickets werden daher de-priorisiert

Die Herausforderungen

→ Erst Know-How aufbauen? → Zu langsam 🐢

→ Externes Team dazuholen? → Koordinationsaufwand 💥

→ "Quick & Dirty" lösen? → Tech Debt 🏚️

Die Lösung mit Spec-Driven

Case 1: Fehlende Expertise ausgleichen

Das Team nimmt die **Anforderungen** auf:
"Was soll das Feature können?" statt **"Wie geht React?"**

Mit Kiro wird die **Spezifikation** geschärft

Die KI übernimmt **die Implementierung** vollständig

Das Team kann sich aufs **Ausrollen und Testen** konzentrieren

Das Ergebnis

Case 1: Fehlende Expertise ausgleichen

✅ Feature **schneller geliefert** - trotz fehlender Frontend-Expertise
✅ Die automatische Validierung stellt die **Qualität** sicher
✅ Specs senken die **Einstiegshürde** - Das Team kann sich auf das "**Was**" fokussieren
✅ Das Team kann Features liefern, die vorher als "**zu kompliziert**" erachtet wurden

👉 Spec-Driven Development als Beschleuniger für Innovation

Case 2
Wissensverlust verhindern

"Wie soll das nochmal funktionieren?"

Case 2: Wissensverlust verhindern

Wissen und Fachlogik gehen verloren

- 📚 **Dokumentation veraltet schnell** - lebt nicht mit dem Code
- 🤔 **Fachprozesse sind unklar** - Code zeigt "**Wie**", nicht "**Warum**"
- 🔄 **Teammitglieder wechseln** - Wissen geht mit ihnen

Die Herausforderungen

→ Ähnliche Fachlichkeit? → Neubau statt Wiederverwendung 🔨
→ Onboarding neuer Devs? → Mühsames Code-Reading 📖
→ 6 Monate später? → "Warum haben wir das so gemacht?" 🤷

Die Lösung mit Spec-Driven

Case 2: Wissensverlust verhindern

Specs dokumentieren "**Warum**", nicht nur "**Wie**" - Requirements, Design & Code **leben zusammen** an einer Stelle

Die Dokumentation wird ausführbar und **veraltet nie** - Jede Iteration erfordert ein Update

Über die Zeit entsteht ein **Fundus an Fachlogik** - wiederverwendbar für neue Features

Wissen bleibt erhalten - auch wenn Teammitglieder wechseln

Das Ergebnis

Case 2: Wissensverlust verhindern

✅ **Klarheit** reduziert Reibungsverluste
✅ **Lebende Dokumentation** des IST-Zustands
✅ **Weniger Regression** durch Wiederverwendung bestehender Anforderungen
✅ **Schnelleres Onboarding** für neue Teammitglieder

👉 Spec-Driven Development reduziert Aufwände, und fördert dadurch Innovation

📋 Oder: Die Renaissance guter Software-Dokumentation

Key Takeaways

Key Takeaways

1. **Spec-Driven skaliert, Vibe Coding nicht**
   → Bei vielen Teams ist Formalisierung essentiell
2. **Innovation durch Enablement**
   → Specs senken Einstiegshürden und schaffen Klarheit
3. **Wiederverwendbarkeit ist kein Zufall**
   → Gemeinsame Specs schaffen gemeinsame Sprache
4. **Investment lohnt sich**
   → Langsamerer Start, aber schnellere Iteration danach

Vielen Dank!

Kontakt

📧 marvin.kruse@otto.de
💼 linkedin.com/in/marvin-kruse

Links

- GitHub Spec-Kit
- Kiro.dev
- Understanding Spec-Driven-Development
- Spec-driven development with AI
- OTTO Tech Blog