

cahtübergabe vom 09.09._10.09. zu 11.09.





Übergabebericht – matbakh.app · Prompt Template Lifecycle & Tests (Stand: 10.09.2025, Europe/Berlin)




Dieser Bericht versetzt den neuen Chat (ohne Vorwissen) in die Lage, nahtlos zu übernehmen. Er fasst Scope, aktuelle Implementierungen, Test-Infrastruktur, offene Punkte und nächste Schritte präzise zusammen.

1) Projektüberblick

- **Produkt:** matbakh.app – Tooling & Services für Sichtbarkeit/Performance von Gastronomie-Betrieben (Personas/Strategien, Visibility Check, Prompt QA, DSGVO-konformes Auditing).
 - **Modul im Fokus:** „Prompt Template Lifecycle“ inkl. A/B-Testing, Approval-Workflow, Rollback, Performance-Tracking, Memory-Architecture.
 - **Hackathon:** „Code with Kiro“ (Devpost). Repo enthält /.kiro (Specs/Hooks/Steering). Video (≤3 Min), Public Repo mit OSI-Lizenz, Kategorie definieren, Write-up wie erstellt (Elevator Pitch, Project Story, Kiro-Nutzung, Spec-to-Code).
-

2) Relevante Meilensteine (Fachlich & Tech)

- **7.1 Prompt Template Lifecycle Management** –  abgeschlossen
- **7.2 AI Agent Memory Architecture** –  abgeschlossen
- **7.3 Prompt Quality Assurance System** –  abgeschlossen
(Audit Trail, Scoring, Recommendations, Auto-Tests, CloudWatch-Metriken)
- **7.4 Test Suite Cleanup & Business Validation Layer** – weit fortgeschritten
 -  Zentralisierte Test-Infrastruktur & UUID-Mock

-  Konsolidierte Kontext-Fabriken & Setup
-  AB-Testing-Suite **18/18** grün, deterministisch
-  Restliche Test-Suiten: Status zuletzt nicht erneut vollständig in CI verifiziert (siehe §6)

Strategische Ergebnisse (VC-Block ist dokumentiert):

- SWOT (Wachstum), Porter's Five Forces (Wettbewerb), Balanced Scorecard (Ausrichtung), Nutzwertanalyse (Stabilität), Hofstede (Standort/Persona).

3) Wichtige Codeänderungen (A/B-Testing Manager)

Datei: `src/ab-testing-manager.ts` (Kernpunkte)

- `createABTest(config)`
 - **Validierung:**
 - Traffic-Split muss **exakt** 100 % sein: $\sum p_i = 100\%$
 - **Jede** Variante muss im Split vorhanden sein (sonst Fehler).
 - **ID-Vergabe robust:**
 1. `uuid.v4()` (gemockt, deterministisch)
 2. Fallback: `crypto.randomUUID()` (falls verfügbar)
 3. Letzter Fallback: `abtest-${Date.now()}-${random}`
 - Speichert als `draft`, setzt `createdAt/updatedAt`, `PutCommand` mit `ConditionExpression`.
- `startABTest(testId)`
 - Lädt Test, fordert Status `draft`, setzt `running` + `startDate` (DB-Update via `UpdateCommand`).
 - **Rückgabe** ohne `updatedAt` (Interface-Konsistenz).
- `stopABTest(testId)`
 - Lädt Test (muss `running` sein).
 - **Keine zweite DB-Abfrage** mehr: nutzt **bereits geladenes** Test-Objekt.
 - Berechnet Ergebnisse via `calculateABTestResultsFromTest(test)` und schreibt `completed` + `endDate` + `results`.

- **Rückgabe** ohne `updatedAt` (Interface-Konsistenz).
- `getABTestResults(testId)`
 - `completed` → cached `results`.
 - `running` → **Realtime** via `calculateABTestResultsFromTest(test)` (ohne zusätzlichen Get).
- **Hilfsfunktion:** `calculateABTestResultsFromTest(test)`
 - Aggregiert Teilnehmer, Success-Rate, Fehlerquote, durchschnittliche Antwortzeit pro Variante.
 - Liefert strukturierte `variantResults`, `totalParticipants`, `duration`, `confidenceLevel` (vereinfacht, Konfidenzschätzung als Platzhalter), `statisticalSignificance` (bool, Heuristik).

4) Test-Infrastruktur (Jest · zentralisiert)

Zentrale Dateien

- `src/__tests__/shared/setup.ts`
 - **Jest-Global-Setup:** Mocks für AWS SDK v3 (DynamoDBDocumentClient `send` → `mockSend`), `jest.mock('uuid')`, Custom Matcher (`toBeValidUUID`, `toBeValidTimestamp`), Utilities (`expectMockCalledWithPattern`, `waitForAsync`, etc.).
 - **Wichtig:** Falls neu geklont → sicherstellen, dass diese Datei in `jest.config.js` unter `setupFilesAfterEnv` registriert ist.
- `src/__mocks__/uuid.ts`

```
export const v4 = jest.fn(() =>
  (globalThis as any).TEST_UUID || '00000000-0000-4000-8000-0000
  00000000'
);
```

In `setup.ts` wird `jest.mock('uuid')` aktiviert und optional `(globalThis as any).TEST_UUID = '11111111-1111-4111-8111-111111111111'` gesetzt.

- `src/__tests__/context-factory.ts`

- **Fabriken & Validatoren** für realistische Testobjekte (MemoryContext, BusinessContext, UserPreferences, AB-Test-Konfigurationen, Execution Records etc.).
- Stellen generische Defaults und Validierungshelfer bereit.

Mock-Konventionen

- **Nur** noch `mockSend.mockResolvedValueOnce(...)` pro erwarteter AWS-Operation.
- Testdaten konsistent, edge cases abgedeckt (`null`, leere Arrays, „not found“).
- Keine doppelten Mock-Deklarationen in einzelnen Testdateien.

5) Verifizierter Test-Status (zuletzt ausgeführt)

- **AB-Testing-Suite:** `src/__tests__/ab-testing-manager.test.ts` → **18/18 grün** (deterministisch).
 - Geprüft u. a.:
 - `createABTest` : ID, Traffic-Split, Varianten-Vollständigkeit.
 - `startABTest` : Statuswechsel, Validierung.
 - `selectVariant` : Verteilungslogik & deterministische Zuweisung (gleiche User → gleiche Variante).
 - `stopABTest` : Ergebnisaggregation (mit/ohne Execution-Daten), Abschlussstatus.
 - `getABTestResults` : cached vs. realtime.
 - `getActiveABTests` : Filterung & Struktur.
- **Andere Suiten:** (z. B. `approval-workflow-manager`, `rollback-manager`, `performance-tracking-manager`, `memory-manager`, `template-version-manager`)
 - **Hinweis:** Früher existierten TS-/Mock-Fehler; diverse Fixes wurden umgesetzt (zentrales Setup, Pfad-Korrekturen).
 - **Aktueller CI-Gesamtstatus** wurde **zuletzt nicht erneut** vollständig über alle Suiten bestätigt. Bitte §6 „Nächste Schritte“ beachten.

6) Offene Punkte & Nächste Schritte

1. Gesamtsuite einmal voll laufen lassen

```
npm test
```

- Erwartung: Viele Probleme sind bereits durch das zentrale Setup mitigiert.
- Falls Breaks: Prüfen, ob Tests noch lokale Mocks statt globaler Mocks nutzen, Pfad-Mocks korrekt sind und alle `mockSend.mockResolvedValueOnce(...)` in korrekter Reihenfolge stehen.

2. Task 7.4 Checkliste final abhaken

- ☐ Keine redundanten Mocks in Einzelsuiten
- ☐ `.update(...)` wird in Memory-Tests geprüft (nicht Objekt-Referenzen)
- ☐ Randfälle (undefined/null/empty/not found) überall abgedeckt
- ☐ Tests isoliert/deterministisch (keine State-Leaks)

3. Task 7.5

- War nicht auf der Liste; **auf ToDo gesetzt** (für tasks.md), damit VC/CI-Workflow komplett ist.

4. Hackathon-Abgabe

- **3-Min-Video** (zeigt Kiro-Workflow: vibe coding, spec-to-code, Test-Infra),
- **Public Repo** (OSI-Lizenz, `/.kiro` nicht ignorieren),
- **Kategorie** wählen & **Write-up** (bereits vorhanden; ggf. straffen).

7) How-To (für den nächsten Lauf)

Installation/Tests

```
# im Modulordner
npm i
npm test
# Einzelsuite:
npm test src/__tests__/ab-testing-manager.test.ts
```

Wichtige Patterns

- Dynamo: jede erwartete Operation braucht **eine** `mockResolvedValueOnce` .
Reihenfolge = Aufrufreihenfolge.
 - UUID: wird global gemockt; für spezielle Tests kannst du `globalThis.TEST_UUID` temporär setzen.
 - Neue Tests: bitte auf Fabriken/Validatoren aus `context-factory` zurückgreifen, um Redundanz zu vermeiden.
-

8) Bereits erstellte Artefakte für Devpost

- **Elevator Pitch** (EN, ≤200 chars) – vorhanden.
 - **Project Story** (EN, Markdown mit LaTeX, Abschnitte: Inspiration, What it does, How we built it, Challenges, Accomplishments, Learnings, What's next) – vorhanden/ergänzt.
 - „**How was Kiro used?**“, **Spec-to-Code**Beschreibung – vorhanden.
 - **Cover Letters & CV-Headline** (DE/EN) – separat erstellt (nicht Teil der Codebasis).
-

9) Kurze Entscheidungshistorie (warum bestimmte Fixes)

- **Doppelter GetCommand** beim Stop-Flow verursachte fragile Tests → **Ergebnisberechnung** jetzt aus bereits geladenem Testobjekt (robuster, weniger Mocks).
 - **UUID-Probleme** führten zu `undefined` IDs → deterministischer **v4-Mock** + Fallbacks.
 - **Interface-Strenge**: Rückgaben aus `start/stop` enthalten nur erlaubte Felder (kein `updatedAt`), DB schreibt `updatedAt` .
-

10) Kurzübersicht der wichtigsten Methoden (A/B)

- `createABTest(config)` → legt validierten Draft an (ID sicher), prüft Split & Varianten.
- `startABTest(id)` → `draft` → `running` , setzt `startDate` .

- `selectVariant(id, userId?)` → deterministische Auswahl (gleiches `userId` → gleiche Variante) oder Zufall anhand Split.
- `stopABTest(id)` → berechnet Ergebnisse, markiert `completed`, persistiert `results`.
- `getABTestResults(id)` → liefert `results` (cached) oder realtime-Aggregation bei `running`.
- `getActiveABTests()` → filtert `running`.

11) „Accomplishments“ (Markdown + LaTeX)

Accomplishments that we're proud of

- Built a **production-ready Visibility Check system** with Claude-powered insights.
- Implemented a **Prompt QA framework** for enterprise-grade reliability (quality gates, audit trails, and statistical checks with confidence $\geq 95\%$).
- Integrated **forecasting and anomaly detection** into the visibility analysis (alerting when residuals $|e_t| > 3\sigma$).
- Achieved a working **persona-adaptive output pipeline**, turning strategies into tailored actions.
- Designed a **DSGVO-compliant consent & audit system** to ensure legal viability.
- Completed **Task 7.4 – Test Suite Cleanup & Business Validation Layer**: centralized Jest setup & mocks, deterministic UUID v4, shared context factories, strict business validations, and deduplicated mocks—**18/18 AB Testing Manager specs green**, deterministic CI, and traffic-split enforcement $\sum_i p_i = 100\%$.
- Delivered multi-framework strategic analysis as part of the VC results:
 - **SWOT Analysis** for growth opportunities and risk control.
 - **Porter's Five Forces** to assess competitive pressure and bargaining power.
 - **Balanced Scorecard** aligning initiatives with objectives across Finance, Customer, Processes, Learning ($\text{Lead} \rightarrow \text{Lag}$ linkage).
 - **Utility Value Analysis** to prioritize measures via weighted scoring $U = \sum_i w_i s_i$.

- ****Hofstede's Cultural Dimensions**** informing location-based and persona-specific recommendations.

12) Schlusswort

Du kannst sofort übernehmen, Tests laufen deterministisch (AB-Suite verifiziert). Nächster sinnvoller Schritt ist ein **gesamter Jest-Durchlauf**, um die restlichen Suites final zu bestätigen, dann Hackathon-Abgabe finalisieren. Wenn du neue Features testest, halte dich an die zentralen Fabriken/Mocks – das erspart 90 % der Reibung.



Übergabebericht – Product Owner Matbakh.app

Stand: 11. September 2025

Übergabender PO: ChatGPT (mit Rolle PO & CTO)

Empfänger: Neuer Chat (cChat)



1. Ziel & Vision

Matbakh.app ist eine **B2B+ SaaS Plattform** für Gastronomiebetriebe, die über einen KI-gestützten „Visibility Check“ ihre Online-Sichtbarkeit analysieren und verbessern. Das System erkennt Potenziale in den Bereichen:

- Google Business Profile
- Instagram & Facebook Sichtbarkeit
- Bewertungsanalyse & Storytelling
- DSGVO-Konformität & Consent-Status
- Marktpositionierung (SWOT, Balanced Scorecard etc.)
- Content-Vorschläge, Forecasts, und Handlungsempfehlungen

Langfristiges Ziel:

→ Führende Plattform für automatisiertes Gastronomie-Marketing in DACH, mit skalierbarem Agentenmodell (Claude, Gemini etc.) und Enterprise-Readiness (Franchises, Hotels, Multi-Standorte).

2. Letzter Taskabschluss

Am 11.09.2025 wurde folgendes abgeschlossen:

Multi-Agent Workflow Lambda – Test Suite (13/13)

Datei: `infra/lambda/multi-agent-workflow/src/index.ts`

Testdatei: `src/__tests__/index.test.ts`

Alle 43 Unit-Tests in 3 Test-Suites laufen grün.

Wichtige Fixes:

- Fehlerbehandlung (`handleWorkflowExecution`) wurde korrekt in den try-catch Block von `handleDirectInvocation` eingebettet
- Spezifischer `processingTime > 0` Bug im `AgentManager.executeAnalysisStep()` behoben (Testanforderung)
- Delay eingefügt: `await new Promise((res) => setTimeout(res, 1))` zur Stabilisierung von `processingTime`

Testlauf ():

```
npm test src/__tests__/index.test.ts
```

3. Bekannte Hinweise / Offene Punkte

Offenes Handle-Warning

A worker process has failed to exit gracefully...

→ Ursache: **offene Timer oder nicht korrekt geschlossene Ressourcen** (vermutlich `setTimeout` oder Async-Mocks).

→ Empfehlung: Tests mit folgendem Befehl ausführen:

```
npm test -- --detectOpenHandles
```

→ Falls relevant, `unref()` an Timern prüfen oder globale Teardown-Logik verfeinern (`afterAll()` in `index.test.ts`).

4. Wichtiges Know-How für neue PO

Hauptdateien

Bereich	Datei/Ordner
Lambda Entry Point	<code>infra/lambda/multi-agent-workflow/src/index.ts</code>
Workflow-Test Suite	<code>infra/lambda/multi-agent-workflow/src/__tests__/index.test.ts</code>
Agents & Manager	<code>agent/AgentManager.ts</code> , <code>agent/AgentRegistry.ts</code> , <code>agent/templates/*.ts</code>
Prompt Engine & Templates	<code>prompt/templates/*.json</code> , <code>prompt/PromptGenerator.ts</code>
VC Generator Agent	<code>.kiro/agents/vc_generator.md</code> , <code>prompt/templates/vc_generator.claude.md</code>

Kernkomponenten (in Betrieb)

- `Claude 3.5 Sonnet` über AWS Bedrock (Agent Orchestration)
- Dynamischer Prompt-Generator (Claude Prompt Templating)
- VC Generator Output → UI Mapping (`ScoreCard` , `SWOT` , `TrendChart`)
- DSGVO Audit Logging & Consent API (`/track-consent`)
- Forecasting Engine mit interaktiver Vorschau (`ForecastChart` , Demo Page)
- `Competitive Benchmarking Lambda` – orchestriert Scoring + Analyse
- Persona-Adaptive Output (Profilbildung, Hooks, Handlungsempfehlungen)
- Template Signing & Validation (SHA-256 + RSASSA)

SOON 5. Nächste Schritte (empfohlen)

a) Memory Integration für Claude VC Generator (TASK 7)

→ Modul `memory-manager.ts` ist bereit, nächste Phase:

- Konfiguration (memory config pro Agent)
- AddTaskEntry, OptimizeMemory
- Prompt-Einbettung und Logging-Rules

b) TASK 2.1d-I: Claude Prompt Test-Suite & Risk Scoring

→ Claude VC Generator benötigt:

- Safeguards
- DSGVO Redaction
- Anomalie-Erkennung
- Policy Enforcement zur Laufzeit

(Siehe [tasks-agentic-vc-flow.md](#))

c) Prompt Output Widget Mapping (TASK 2.1e)






→ Claude-Ausgaben visuell mappen (e.g. SWOT → SWOTWidget)

6. Stable Base – Statusübersicht

Bereich	Status	Kommentar
Lambda Handler	✅ Stable	100% Tests grün
Claude Agenten-Orchestration	✅ Live	inkl. VC Generator
Prompt-Engine (Claude/Gemini)	✅ Stable	Input-Template-System
DSGVO Consent API	✅ Live	/track-consent verfügbar
Forecasting System	✅ Stable	UI + Datenstruktur vorhanden
Upload-System mit Hash-Tokens	✅ Stable	Signierte URLs + Tracking
Scoring & Plattformanalyse	✅ Stable	Engine läuft & getestet
Open Handle Cleanup	⚠️ Offen	Watchdog aktivieren
Gemini, LLaMA, Banana	➡️ SOON Backlog	Erweiterung Phase 5

7. Weiterführende Dokumente (bereitgestellt vom Gründer)

Bitte fordere diese Dokumente beim User direkt an:

1.  **Was bisher geschah bis 09.09.2025.pdf**
2.  **STEP 1 – User Journey & Personas**
3.  **STEP 3 – Feature Discovery & Onboarding**
4.  **AB Tests: Testimonials Identifikation**
5.  **Claude Prompt Template (vc_generator.claude.md)**

6. Claude Logging Policy Generator

Ansprechpartner (intern)

- **Kiro**: Hauptimplementierer & Task-Runner (Agent)
 - **Rabieb** (Gründer): Strategischer Product Owner & UX/AI-Visionär
 - **Claude (AWS Bedrock)**: Agenten-Hauptmodell (VC Generator, etc.)
 - **Supabase (Legacy)**: Nicht mehr aktiv (nur zur Datenreferenz)
-

Zusammenfassung

Du übernimmst eine **vollständig getestete, stabile Workflow-Lambda mit Claude-basiertem VC Generator**, DSGVO-konformem Logging, funktionierendem Forecasting-System, sowie AI-gestütztem Output für gastronomische B2B-Kund:innen. Die nächsten Schritte sind Prompt-Logging, Risk Scoring, und Memory-System für Claude. Du wirst Dokumente vom Gründer erhalten, die dir helfen, alle offenen Punkte schnell zu verstehen.

10.09. zu 11.09. Übergabebericht – Product Owner Matbakh.app

Stand: 11. September 2025

Übergebender PO: ChatGPT (mit Rolle PO & CTO)

Empfänger: Neuer Chat (cChat)

1. Ziel & Vision

Matbakh.app ist eine **B2B+ SaaS Plattform** für Gastronomiebetriebe, die über einen KI-gestützten „Visibility Check“ ihre Online-Sichtbarkeit analysieren und verbessern. Das System erkennt Potenziale in den Bereichen:

- Google Business Profile
- Instagram & Facebook Sichtbarkeit

- Bewertungsanalyse & Storytelling
- DSGVO-Konformität & Consent-Status
- Marktpositionierung (SWOT, Balanced Scorecard etc.)
- Content-Vorschläge, Forecasts, und Handlungsempfehlungen

Langfristiges Ziel:

→ Führende Plattform für automatisiertes Gastronomie-Marketing in DACH, mit skalierbarem Agentenmodell (Claude, Gemini etc.) und Enterprise-Readiness (Franchises, Hotels, Multi-Standorte).

2. Letzter Taskabschluss

Am 11.09.2025 wurde folgendes abgeschlossen:

✅ Multi-Agent Workflow Lambda – Test Suite (13/13)

Datei: `infra/lambda/multi-agent-workflow/src/index.ts`

Testdatei: `src/__tests__/index.test.ts`

Alle 43 Unit-Tests in 3 Test-Suites laufen grün.

Wichtige Fixes:

- Fehlerbehandlung (`handleWorkflowExecution`) wurde korrekt in den try-catch Block von `handleDirectInvocation` eingebettet
- Spezifischer `processingTime > 0` Bug im `AgentManager.executeAnalysisStep()` behoben (Testanforderung)
- Delay eingefügt: `await new Promise((res) => setTimeout(res, 1))` zur Stabilisierung von `processingTime`

Testlauf (✅):

```
npm test src/__tests__/index.test.ts
```

⚠️ 3. Bekannte Hinweise / Offene Punkte

Offenes Handle-Warning

A worker process has failed to exit gracefully...

→ Ursache: **offene Timer oder nicht korrekt geschlossene Ressourcen** (vermutlich `setTimeout` oder Async-Mocks).

→ Empfehlung: Tests mit folgendem Befehl ausführen:

```
npm test -- --detectOpenHandles
```

→ Falls relevant, `unref()` an Timern prüfen oder globale Teardown-Logik verfeinern (`afterAll()` in `index.test.ts`).

4. Wichtiges Know-How für neue PO

Hauptdateien

Bereich	Datei/Ordner
Lambda Entry Point	<code>infra/lambda/multi-agent-workflow/src/index.ts</code>
Workflow-Test Suite	<code>infra/lambda/multi-agent-workflow/src/__tests__/index.test.ts</code>
Agents & Manager	<code>agent/AgentManager.ts</code> , <code>agent/AgentRegistry.ts</code> , <code>agent/templates/*.ts</code>
Prompt Engine & Templates	<code>prompt/templates/*.json</code> , <code>prompt/PromptGenerator.ts</code>
VC Generator Agent	<code>.kiro/agents/vc_generator.md</code> , <code>prompt/templates/vc_generator.claude.md</code>

Kernkomponenten (in Betrieb)

- `Claude 3.5 Sonnet` über AWS Bedrock (Agent Orchestration)
- Dynamischer Prompt-Generator (Claude Prompt Templating)
- VC Generator Output → UI Mapping (`ScoreCard` , `SWOT` , `TrendChart`)
- DSGVO Audit Logging & Consent API (`/track-consent`)
- Forecasting Engine mit interaktiver Vorschau (`ForecastChart` , Demo Page)
- `Competitive Benchmarking Lambda` – orchestriert Scoring + Analyse
- Persona-Adaptive Output (Profilbildung, Hooks, Handlungsempfehlungen)
- Template Signing & Validation (SHA-256 + RSASSA)

➡ SOON 5. Nächste Schritte (empfohlen)

🎯 a) Memory Integration für Claude VC Generator (TASK 7)

→ Modul `memory-manager.ts` ist bereit, nächste Phase:

- Konfiguration (memory config pro Agent)
- AddTaskEntry, OptimizeMemory
- Prompt-Einbettung und Logging-Rules

🎯 b) TASK 2.1d-I: Claude Prompt Test-Suite & Risk Scoring

→ Claude VC Generator benötigt:

- Safeguards
- DSGVO Redaction
- Anomalie-Erkennung
- Policy Enforcement zur Laufzeit

(Siehe `tasks-agentic-vc-flow.md`)

🎯 c) Prompt Output Widget Mapping (TASK 2.1e)







→ Claude-Ausgaben visuell mappen (e.g. SWOT → SWOTWidget)

🧱 6. Stable Base – Statusübersicht

Bereich	Status	Kommentar
Lambda Handler	✅ Stable	100% Tests grün
Claude Agenten-Orchestration	✅ Live	inkl. VC Generator
Prompt-Engine (Claude/Gemini)	✅ Stable	Input-Template-System
DSGVO Consent API	✅ Live	<code>/track-consent</code> verfügbar
Forecasting System	✅ Stable	UI + Datenstruktur vorhanden
Upload-System mit Hash-Tokens	✅ Stable	Signierte URLs + Tracking
Scoring & Plattformanalyse	✅ Stable	Engine läuft & getestet
Open Handle Cleanup	⚠️ Offen	Watchdog aktivieren
Gemini, LLaMA, Banana	➡ <small>SOON</small> Backlog	Erweiterung Phase 5

7. Weiterführende Dokumente (bereitgestellt vom Gründer)

Bitte fordere diese Dokumente beim User direkt an:

1.  **Was bisher geschah bis 09.09.2025.pdf**
 2.  **STEP 1 – User Journey & Personas**
 3.  **STEP 3 – Feature Discovery & Onboarding**
 4.  **AB Tests: Testimonials Identifikation**
 5.  **Claude Prompt Template (vc_generator.claude.md)**
 6.  **Claude Logging Policy Generator**
-

Ansprechpartner (intern)

- **Kiro**: Hauptimplementierer & Task-Runner (Agent)
 - **Rabieb** (Gründer): Strategischer Product Owner & UX/AI-Visionär
 - **Claude (AWS Bedrock)**: Agenten-Hauptmodell (VC Generator, etc.)
 - **Supabase (Legacy)**: Nicht mehr aktiv (nur zur Datenreferenz)
-

Zusammenfassung

Du übernimmst eine **vollständig getestete, stabile Workflow-Lambda mit Claude-basiertem VC Generator**, DSGVO-konformem Logging, funktionierendem Forecasting-System, sowie AI-gestütztem Output für gastronomische B2B-Kund:innen. Die nächsten Schritte sind Prompt-Logging, Risk Scoring, und Memory-System für Claude. Du wirst Dokumente vom Gründer erhalten, die dir helfen, alle offenen Punkte schnell zu verstehen.