

Chatübergabe 17.09. bis zu 24.09.



Matbakh.app Projekt - Übergabebericht (Stand: 22.09.2025)



Projektüberblick

- Projektname:** matbakh.app
- Ziel:** Sichtbarkeits- und Performance-Plattform für Gastronomiebetriebe im arabischen Raum.
- Technologie-Stack:**
- **Frontend:** Next.js / React (Vercel)
 - **Backend:** AWS Lambda (Node.js), AWS API Gateway
 - **Datenbank:** AWS RDS (PostgreSQL)
 - **Auth:** AWS Cognito (User Pool)
 - **CI/CD:** GitHub Actions
 - **Testing:** Jest + ts-jest
 - **Tooling:** Kiro (lokaler Dev CLI)



Infrastruktur (Live in Produktion)

Komponente	Details
Domain	https://www.matbakh.app
AWS Region	eu-central-1 (Frankfurt)
S3 Bucket	matbakhvcstack-webbucket12880f5b-svct6cxfbip5
CloudFront Distribution	E2W4JULEW8BXSD
RDS Endpoint	matbakh-db.chq6q4cs0evx.eu-central-1.rds.amazonaws.com

Komponente	Details
Auth	AWS Cognito, Client ID per <code>.env</code> : <code>VITE_COGNITO_USER_POOL_CLIENT_ID</code>

Repositories

- **GitHub:** <https://github.com/matbakh-app/matbakh-visibility-boost>
- **Branch in Arbeit:** `test/green-core-validation`

Migration

Von Supabase zu AWS - Supabase wurde vollständig aus dem Backend entfernt. Derzeit erfolgt die finale Anpassung im Service Layer + CI.

Lokale Entwicklung (MacBook Pro)

Setup-Befehle:

```
aws sso login --profile matbakh-dev
export AWS_PROFILE=matbakh-dev
export VITE_COGNITO_USER_POOL_CLIENT_ID=<id>
export NODE_ENV=development
export REPO="/Users/matbakh-visibility-boost.20250920"
```

Kiro starten:

```
kiro dev
```

Aktueller Task (22.09.2025) - "Green Core Validation"

Problem:

- Die CI-Tests für `persona-api.ts` schlagen fehl, weil `jest@29` mit `jest-environment-jsdom@30` inkompatibel ist.

Symptome:

- CI bricht beim Schritt **"Persona Service Core Functions"** ab

- Kein einziger Test wird ausgeführt, Runtime-Konflikt zwischen Jest-Versionen

Letzter Commit (HEAD)

```
git log -1  
commit a5a55ab... (HEAD → test/green-core-validation)
```

Aktive Branches:

- `main`
- `kiro-dev`
- `aws-deploy`
- `test/green-core-validation` — **CI-Testbranch**

Letzter Status:

CI läuft korrekt **bis zum Jest-Schritt**, dann Runtime-Crash.

ToDo / Fix:

Entweder

```
npm install --save-dev jest-environment-jsdom@29  
rm package-lock.json  
npm install
```

Oder komplett auf Jest 30 umsteigen (riskant)

Commit-Vorschlag:

```
git add package.json package-lock.json  
git commit -m "fix(ci): align jest-environment-jsdom with jest@29"  
git push origin test/green-core-validation
```



Nächste Schritte

1. **Green-Core-Tests reparieren** (siehe oben)
2. `test/green-core-validation` mergen nach `main`

3. Review: `.env` , `.github/workflows` , Supabase-Reste checken
4. Optional: neue persona-api Tests zu Edge-Cases ("low signal")

✉ Bei Problemen

- Kiro-Probleme meist durch fehlendes `REPO` oder `AWS_PROFILE`
 - `persona-api.ts` wirft KEINE `throw new Error` mehr – Tests erwarten strukturierte `{ success: false, error: ... }`
-

Übergabebericht – Matbakh.app Projekt (Stand: 23.09.2025)

1. Projektkontext

- **Produkt:** matbakh.app – Visibility & Performance Plattform
 - **Ziel:** Migration von Supabase/Vercel zu einer stabilen AWS-only Infrastruktur, Eliminierung von Legacy-Komponenten, Aufbau einer performanten, sicheren und skalierbaren Plattform.
 - **Status:** Supabase ist weitgehend entfernt, Backend läuft auf AWS, Frontend (React + Vite) ist umgestellt, Tests sind in finaler Bereinigung.
-

2. Infrastruktur

AWS

- **Account ID:** `055062860590`
- **Region:** `eu-central-1`
- **Dienste:**
 - **RDS:** PostgreSQL, Endpoint = `matbakh-db.chq6q4cs0evx.eu-central-1.rds.amazonaws.com`
 - **Cognito:** Auth (Frontend via `VITE_COGNITO_USER_POOL_CLIENT_ID` in `.env`)

- **S3:** Bucket `matbakhvcstack-webbucket12880f5b-svct6cxfbip5`
- **CloudFront:** Distribution ID `E2W4JULEW8BXSD` (Frontend Delivery)
- **Lambdas:** für Services wie Persona, File Upload, Scoring, DB Optimization
- **CloudWatch:** Monitoring + Alerts (teilweise integriert)

Frontend

- **Framework:** Vite + React + shadcn/ui
- **Styling:** Tailwind, Radix UI Components
- **State Mgmt:** React Query (TanStack)
- **Testing:** Jest, Playwright (smoke/e2e), ts-jest
- **CI/CD:** GitHub Actions (Green Core Validation, Purity Checks)

Backend / Services

- **Language:** TypeScript (Node.js)
 - **Services (nach Migration):**
 - ProfileService → AWS RDS
 - Score History Service → RDS `score_history`
 - Benchmark Comparison → RDS queries
 - Persona API → läuft als Lambda
 - S3 Upload Decorator → Dateigrößenvalidierung + Security
 - Database Performance Optimizer → Connection Pooling, Redis Cache, Index Optimizer
 - **Caching:** Redis (ioredis) – für Query Cache und Invalidierung
 - **Email/Notifications:** SES / SNS (devDependencies installiert)
-






3. Code-Struktur

- **src/lib/database/** → Optimizer, Connection Pool, Redis Cache, Index Optimizer
- **src/hooks/** → `useDatabaseOptimization` , `useDatabasePerformance` (React Hooks)

- **src/services/** → APIs (Persona, Profile, Benchmark, Score History, etc.)
 - **infra/lambda/** → Lambda Handlers + Tests
-

4. Migration / Cleanup Tasks

Abgeschlossene Haupttasks

-  Service Layer Migration (Supabase entfernt → AWS RDS only)
-  Environment Cleanup (`.env` konsolidiert, Supabase Variablen entfernt)
-  Test Layer Migration (Jest grün für Core DB + Pool + Optimizer)
-  Dependency Cleanup (Supabase SDK entfernt, neue AWS SDKs integriert)
-  Database Performance Optimization (Optimizer, Pool, Cache, Indexer)

Offene / Aktuelle Tasks

1. **S3 Upload Decorator** – Dateigröße korrekt erfassen
 2. **Test Hook Cleanup** – `useDatabaseOptimization.test.ts` (noch 4 Tests failing: Metrics/Status/Score)
 3. **Architekturdiagramm** – noch ausstehend (mit AI-Integration wie Gemini)
 4. **Langfristig** – Integration von KI-Systemen (Gemini, OpenAI, Vibe Coding) für Scoring + Recommendations
-

5. CI/CD & Governance

- **CI:** GitHub Actions
 - Green Core Validation → Jest Tests für Core Services
 - Purity Checks → Kiro-Script (`scripts/run-kiro-purity-working.ts`)
 - **Governance Tools:**
 - Pre-commit Hooks (Linting, Test Enforcement)
 - Automated Purity Validation
 - Rollback Scripts (`archive/rollback.sh`)
-

6. Monitoring & Recovery

- **CloudWatch** für Logs, Alarms (DB Utilization, Errors, Lambda Timeouts)
 - **Rollback:**
 - Komplette Archive in `src/archive/consolidated-legacy-archive-*`
 - `ARCHIVED-FEATURES.md` listet 391 entfernte Komponenten mit Recovery Steps
 - **Recovery Plan:** Emergency Procedures + Automated Rollback
-

7. Business-Ebene (für PO-Rolle)

- **Value Proposition:** Enterprise-grade Visibility & Performance Monitoring für KMU/Enterprise Kunden (DACH/MENA)
 - **Target Market:** DACH (Deutschland/Österreich/Schweiz) & MENA (Saudi, VAE, Ägypten)
 - **USP:**
 - KI-basierte Empfehlungen (Performance/Scoring)
 - AWS-only Infrastruktur → Stability + Scalability
 - Plug-and-play Monitoring für Datenbanken & Systeme
-

8. Aktueller Stand Tests

- **DB Optimizer Suite:** ✅ 100%
 - **Connection Pool Suite:** ✅ 100%
 - **Hook Tests (useDatabaseOptimization):** ❌ 4 Tests failing (Metrics & Status Calculation)
 - **Persona Service Tests:** 🟡 Teilweise migrated, CI muss angepasst werden
 - **Green Core Validation:** 🟡 noch instabil durch Hook-Tests
-

9. Next Steps (Prioritäten)

1. **Fix Hook Metrics Tests** (`useDatabaseOptimization.test.ts`)
 - sicherstellen, dass `metrics` sofort nach `loadPerformanceData()` gesetzt wird

- `performanceStatus` & `optimizationScore` robust gegen fehlende Felder machen
2. **S3 Upload Decorator** fertigstellen (Dateigrößenvalidierung → Security wichtig)
 3. **Architekturdiagramm** (inkl. AI-Komponenten wie Gemini) zeichnen und dokumentieren
 4. **Green Core Validation** Pipeline stabilisieren (alle Jest-Tests grün)
 5. **Vorbereitung Production Deployment** (Monitoring + Rollback verifizieren)
-



Gesamtübergabebericht – matbakh.app (Stand 23.09.2025)

Dieser Bericht fasst die gesamte Historie der Übergaben bis heute zusammen (inkl. PDFs vom **08.–14.09.**

Was bisher geschah bis 09.09.20...

Chatübergabe_von_08.09-2_auf_0...

Cahtübergabe_vom_09._10.09._zu...

Chatbergabe_11.09_12.09_13.09._... sowie aktueller Status vom 22.–23.09.) und soll den nächsten Chat in die Lage versetzen, **die PO- und CTO-Rolle sofort zu übernehmen.**

1. Produkt & Ziel

- **Name:** matbakh.app
 - **Ziel:** Visibility & Performance Plattform für Gastronomiebetriebe (DACH & MENA).
 - **Vision:** Gastronomen können ihre digitale Sichtbarkeit (Google Business, Instagram, FB) prüfen, optimieren und Forecasts + KI-Empfehlungen erhalten.
 - **USP:** Kombination aus AI-gestützten Handlungsempfehlungen (Claude 3.5 Sonnet via Bedrock), automatisiertem Benchmarking, Forecasting und DSGVO-konformer Datenverarbeitung.
-



2. Infrastruktur

- **Cloud:** AWS-only (Supabase & Vercel sind entfernt, nur noch Reste in Tests/Env bis 13.09.).
 - **Region:** eu-central-1 (Frankfurt)
 - **Account ID:** 055062860590
 - **Komponenten:**
 - **Backend:** AWS Lambda (Node.js 20, teils Python 3.11)
 - **DB:** RDS PostgreSQL (`matbakh-db.chq6q4cs0evx.eu-central-1.rds.amazonaws.com`)
 - **Auth:** AWS Cognito (User Pools), `.env` → `VITE_COGNITO_USER_POOL_CLIENT_ID`
 - **Storage/CDN:** S3 `matbakhvcstack-webbucket12880f5b-svct6cxfbip5` + CloudFront `E2W4JULEW8BXSD`
 - **Monitoring:** CloudWatch (teilweise), Rollback-Skripte archiviert
 - **AI:** AWS Bedrock (Claude 3.5 Sonnet), vorbereitet für Gemini/Meta/Opal
-

3. Frontend

- **Framework:** React + Vite + Tailwind + shadcn/ui
 - **State Mgmt:** Zustand + React Query
 - **Testing:** Jest (ts-jest), Playwright (geplant), Percy (geplant)
 - **UI-Module:** VCResult, ForecastChart, BenchmarkComparison, PersonaOutputBox, SWOTDisplay
 - **Flows:**
 - VC Wizard (Visibility Check → Claude-Auswertung → Forecast → Benchmark)
 - Consent/Upload Flow (noch DSGVO-Lücken)
 - Persona-Adaptive UI (in Arbeit)
-

4. Backend & Services

- **ProfileService** → RDS
- **ScoreHistoryService** → RDS (Tests )
- **BenchmarkComparisonService** → RDS (Tests noch , siehe 11.09 Bericht)

- **Persona API** → Lambda (Tests teils rot, Login-Flow noch nicht final)
 - **Database Performance Optimizer** → Connection Pool, Redis-Cache, Index-Empfehlungen
 - **Upload Service** → S3-Decorator (File Size Validation in Arbeit)
 - **AI Services** → Claude VC Generator, Prompt Manager, Persona Detection
-

5. Migrationstatus (08.–14.09.)

- **08.–09.09:** Fokus Test-Suite Cleanup (Task 7.4), DSGVO Prompt-Logging noch offen Was bisher geschah bis 09.09.20... Chatübergabe_von_08.09-2_auf_0...
 - **09.–11.09:** Prompt Lifecycle + A/B-Testing-System abgeschlossen (18/18 grün), Memory-Architecture stabil Chatübergabe_vom_09._10.09._zu...
 - **11.–14.09:** Advanced Persona System fertig, Auth-Migration Cognito aktiv, Supabase-Env noch nicht ganz clean, Staging Tests blockiert Chatübergabe_11.09_12.09_13.09._...
-




6. CI/CD & Tests

- **CI:** GitHub Actions (Green Core Validation, Purity Validator via Kiro)
- **Testing-Probleme:**
 - Jest 29.7.0 inkompatibel mit jest-environment-jsdom 30 (aktuelles Problem)
 - Persona Service Tests brechen in CI ab (Runtime-Konflikt)
 - Hook `useDatabaseOptimization` → 4 Tests failing (Metrics & Status)
- **Letzter Commit (HEAD, 22.09):** `a5a55ab` auf Branch `test/green-core-validation`

Fix-Strategie:

- Kurzfristig → `jest-environment-jsdom@29` installieren und lockfile neu bauen
 - Langfristig → Upgrade auf Jest 30 (riskant, erfordert Mocks/Polyfills Update)
-

7. Aktueller Status (23.09.2025)

- **System Optimization Phase gestartet** (nach Cleanup-Phase)
 - **Phase 1 (Performance Optimization):**
 - Echtzeit-Monitoring 
 - Automatic Optimization Engine 
 - DB Performance Optimization  (in Arbeit → Query Caching, Indexer, Pool Scaling)
 - **Green Core Validation CI** noch rot wegen Jest-Konflikt
 - **Architekturdiagramm mit AI-Komponenten** ausstehend
 - **S3 Upload Decorator** muss noch File Size korrekt loggen
-

8. Nächste Schritte (Priorität)

1. **CI-Tests reparieren (Jest/Persona Service)**
 - `npm install --save-dev jest-environment-jsdom@29`
 - Lockfile neu erstellen
 - Commit & Push auf `test/green-core-validation`
 2. **Fix Hook Tests (`useDatabaseOptimization.test.ts`)**
 - Metrics sofort nach `loadPerformanceData()` setzen
 - `performanceStatus` & `optimizationScore` robust machen
 3. **S3 Upload Decorator** (Security & File Size)
 4. **Architekturdiagramm inkl. AI** dokumentieren
 5. **Merge Branch** `test/green-core-validation` → `main` sobald CI grün
-

9. Business/PO-Ebene

- **Target Market:** DACH + MENA (Saudi, VAE, Ägypten)
- **Value Prop:** Stabilität durch AWS-only, AI-gestützte Empfehlungen, DSGVO-Konformität
- **Investorenstrategie:** Enterprise SaaS mit KI-Differenzierung
- **Risiken:** DSGVO-Logging noch offen, Auth-Flow muss 100% stabil sein

Übergabebericht – matbakh.app (PO/CTO)

Ziel dieses Dokuments: Vollständige, operative Übergabe an den neuen Chat/Owner. Der neue Chat soll mich als **PO** und **CTO** zu 100 % ersetzen können.

1) TL;DR / Status

- **Projektname:** matbakh.app
 - **Fokus dieses Meilensteins:** *Task 5 – Quality Assurance Automation* (QA-Portal + CI/CD-Checks + Berichte)
 - **Status:**  *Task 5 abgeschlossen, alle relevanten QA-Tests grün*
 - **Build-/Runtime:** Node 20, Vite + React + TypeScript
 - **Lokale QA-API:** Express-Server (`scripts/qa-server.ts`) – speichert Reports in `qa-reports/`
 - **Kritische ENV-Keys:** `VITE_QA_API_URL` , `AWS_*` , optional `SNYK_TOKEN` , optional `SONAR_TOKEN` , `QA_SERVER_PORT`
-

2) Produktziele (technische Sicht)

Hinweis: Diese Übergabe fokussiert den technischen QA-Unterbau der Web-App. Produktseitige Domänen-Annahmen werden bewusst vermieden; Kernziel ist Qualitätssicherung, Nachvollziehbarkeit und Automatisierung in Dev-/CI-Flows.

Leitziele:

1. **Frühe Fehlererkennung** (Codequalität, Sicherheit, Accessibility) – automatisiert, reproduzierbar, messbar.
2. **Standardisierte Quality Gates** (Coverage, Bugs, Vulnerabilities, Duplicates, Ratings).
3. **Einheitliche Reports** für Menschen (Markdown) und Maschinen (JSON) – lokal und in CI.
4. **Skalierbarkeit:** Konfiguration per ENV/Secrets, austauschbare Scanner, parallele Ausführung, Caching-Potenzial.

Definition of Done (QA-relevant):

- Alle QA-Jobs in CI durchlaufen.
 - Keine kritischen/harten Verstöße (Policies einhaltbar via Config).
 - Reports erzeugt, archiviert und bei PRs sichtbar (Kommentar/Artefakte).
-

3) Architektur (High Level)

Frontend

- **Stack:** Vite, React 18, TypeScript, Tailwind (+ `tailwindcss-animate`), shadcn/ui, lucide-react, React Router, TanStack Query, i18n (i18next), recharts.
- **QA-UI:**
 - `QADashboard` – Bedienelemente, Status, Detail-Tabs, Downloads.
 - `useQualityAssurance` Hook – orchestriert API-Calls zum QA-Server.
 - **Test Page:** `QATestPage.tsx` – Demos/Interaktion.

QA-Server (Node)

- **Datei:** `scripts/qa-server.ts` (Express)
- **Endpoints:**
 - `POST /api/qa/analyze` – Vollanalyse (Code Review, Security, Accessibility, Quality Gates)
 - `POST /api/qa/quick-scan` – Code Review + Security
 - `POST /api/qa/code-review`
 - `POST /api/qa/security-scan`
 - `POST /api/qa/accessibility-test`
 - `POST /api/qa/quality-gates`
 - `GET /api/qa/reports` – Liste vorhandener Reports
 - `GET /api/qa/reports/:filename` – Report lesen
 - `GET /health` – Healthcheck
- **Reports:** JSON-Dateien unter `qa-reports/` mit Zeitstempel.

Orchestrierung (Library)

- **Modul:** `src/lib/quality-assurance` (Orchestrator + Subsysteme)
 - **AI Code Review** (`ai-code-reviewer.ts`) via **AWS Bedrock** (Claude 3.5 Sonnet) – robust mit Fallback/Parsing.
 - **Security Scanner** (`security-scanner.ts`) – npm audit, Snyk (optional), statische Checks.
 - **Accessibility** (`accessibility-tester.ts`) – `@axe-core/playwright` (WCAG A/AA/AAA), Playwright-Läufe.
 - **Quality Gates** (`code-quality-gates.ts`) – Coverage/ESLint/Sonar (optional über Token) + Schwellen.
 - **QA Orchestrator** (`qa-orchestrator.ts`) – `runFullQAAanalysis`, `runQuickScan`, `runCodeReviewOnly`, `runSecurityOnlyScan`, `runAccessibilityOnlyScan`, `runQualityGate`, Reportgenerierung (MD/JSON).

CI/CD

- **Workflow:** *Quality Assurance Pipeline* (GitHub Actions)
 - Jobs: `ai-code-review` , `security-scan` , `accessibility-test` , `quality-gates` , `qa-summary` , `notify-results`
 - Artefakte-Upload (Reports, Coverage, ESLint, Snyk)
 - Optional: PR-Kommentar mit QA-Summary, Slack-Notifies

4) Environments & Konfiguration

.env (lokal):

```
VITE_QA_API_URL=http://localhost:3001
QA_SERVER_PORT=3001

AWS_REGION=eu-central-1
AWS_ACCESS_KEY_ID=...
AWS_SECRET_ACCESS_KEY=...

# optional
SNYK_TOKEN=...
SONAR_TOKEN=...
```

Ports:

- Vite Preview/Dev: **5173**
- QA-Server: **3001**

Secrets in CI:

- `AWS_REGION` , `AWS_ACCESS_KEY_ID` , `AWS_SECRET_ACCESS_KEY`
 - optional `SNYK_TOKEN` , optional `SONAR_TOKEN` , optional `SLACK_WEBHOOK`
-

5) Wichtige NPM-Skripte

Entwicklung & App:

- `npm run dev` – Vite dev server
- `npm run preview` – Vite preview (Prod-Build)
- `npm run build` – TS + Vite build

QA lokal (via Orchestrator-Scripts):

- `npm run qa:server` / `qa:server:dev` – QA-API starten
- `npm run qa:full` – komplette QA-Pipeline
- `npm run qa:quick` – schneller Scan (Code Review + Security)
- `npm run qa:code-review` / `qa:security` / `qa:accessibility` / `qa:quality-gates`
- `npm run qa:summary` – fasst Artefakte zusammen

Tests:

- `npm test` – Jest Unit/Component
 - `npm run test:e2e` – Playwright
 - `npm run test:performance` – Lighthouse CI
 - `npm run test:qa` – Kombi (e2e, visual, cross-browser, performance)
 - `npm run test:coverage` – Jest Coverage
-

6) QA-Subsysteme – Details & Policies

6.1 AI Code Review

- **Modell:** Claude 3.5 Sonnet via AWS Bedrock (konfigurierbar über ENV)
- **Input:** Mehrere Dateien (TS/TSX/JS/JSX) – Hook/Server übergibt Inhalt/Pfade
- **Output:** Score, Suggestion-Liste (Kategorie, Severity, Zeile, Konfidenz), AI-Analyse
- **Robustheit:** JSON-Extraktion aus Markdown, Fallback-Ergebnis bei API-Failure

6.2 Security Scans

- **Dependency:** `npm audit` (immer), **Snyk** (wenn Token vorhanden)
- **Static Checks:** Heuristiken (z. B. `eval`, XSS-Risiken, Credentials)
- **Policy:** erlaubte Severities und Max-Anzahlen konfigurierbar (z. B. *keine critical/high*)

6.3 Accessibility (WCAG)

- **Tooling:** Playwright + `@axe-core/playwright`
- **Tags:** `wcag2a`, `wcag2aa`, `wcag21aa` (je nach Zielniveau)
- **Output:** Score, Violations nach Impact, Empfehlungen, Pass/Fail gemäß erlaubten Verstößen
- **Reports:** pro URL, zusätzlich kombinierte Zusammenfassung

6.4 Quality Gates

- **Quellen:** Jest Coverage (Icov), ESLint (JSON), Sonar (optional)
- **Schwellen (Beispiel):** Coverage $\geq 80\%$, Duplicates $\leq 3\%$, Maintainability $\geq B$, Reliability $\geq A$, Security $\geq A$, Bugs 0, Vulnerabilities 0
- **Ergebnis:** Status (*passed/warning/failed*) + Quality-Score

6.5 Reports & Aggregation

- **Speicherort:** `qa-reports/qa-report-<timestamp>.json` (Full), zusätzliche Artefakte je Teil-System
 - **Formate:** Markdown (menschenslesbar), JSON (maschinell)
 - **Endpoints:** `GET /api/qa/reports` und `GET /api/qa/reports/:filename`
-

7) Teststrategie (Stand: grün)

- **Unit/Lib:** Jest – Orchestrator und Reviewer/Scanner-Interfaces
 - **Component:** Jest + Testing Library – `QADashboard` (Render, API-Fehlerfälle, Resultdarstellung)
 - **E2E/Accessibility:** Playwright + axe-core – mehrere Routen + Spezial-Checks (Kontrast, Keyboard, ARIA)
 - **Stabilität:** Problemursachen (Dateizugriff, Mock-Lücken) behoben; Mocks für FS/Scanner/Bedrock konsistent; Transform/ESM via `ts-jest` konfiguriert
-

8) Betriebs-Playbook (neuer Owner/Chat)

8.1 Daily / PR-Flow

1. **PR geöffnet:** CI startet *Quality Assurance Pipeline*.
2. **Artefakte prüfen:** `code-review-results` , `security-scan-results` , `accessibility-test-results` , `quality-gate-results` , `qa-summary-report` .
3. **PR-Kommentar (falls aktiv):** Ampelfarbe + Kernempfehlungen; bei *failed* → Blocker fixen.
4. **Merge-Regeln:** Keine kritischen Sicherheitslücken; Quality Gates **passed**; Accessibility AA im tolerierten Rahmen.

8.2 Lokale Verifikation (vor PR)

- QA-Server starten: `npm run qa:server` (ENV `VITE_QA_API_URL=http://localhost:3001`)
- App (Preview oder Dev) starten: `npm run preview` **oder** `npm run dev`
- Komplettlauf: `npm run qa:full` – Reports anschließend unter `qa-reports/`

8.3 Incident-Response (QA bricht)

- Logs checken (Server + CI-Job-Logs)
 - ENV/Secrets prüfen (AWS/Snyk/Sonar vorhanden?)
 - Upstream-Abhängigkeiten (Playwright/Chromium) installiert?
 - Falls *axe-core* flakey: Timeout/Navigation/Selektoren prüfen; einzelne Tests gezielt ausführen.
-

9) Risiken & Annahmen

- **Externe Dienste:** Bedrock/Snyk/Sonar → Ausfälle/Quoten/Kosten möglich; Tests sind gemockt, CI nutzt echte Services nur, wenn Secrets gesetzt.
 - **Parsing-Resilienz:** AI-JSON Parsing robust, dennoch Edge-Cases möglich → Fallback vorhanden.
 - **Cross-Env Unterschiede:** CI-Linux vs. lokale Macs/Windows (Fonts, Headless-Browser) – Playwright-Flags im Blick behalten.
 - **ESM/TS Tooling:** `ts-jest` + `transformIgnorePatterns` bereits justiert; neue ESM-Pakete ggf. whitelisten.
-

10) Roadmap (empfohlen)

- **PR-Kommentar anreichern:** Top-Verstöße inline pro Datei (Links zu Codezeilen) + kurze Fix-Howtos.
 - **Caching:** Zwischenspeicher für AI-Reviews & Snyk-Ergebnisse (Hash über Datei+lockfile), um Laufzeiten/Kosten zu reduzieren.
 - **Quality Gates erweitern:** Cyclomatic Complexity/Bundle Size/Perf-Budgets (Lighthouse) einbeziehen.
 - **A11y-Playbook:** Fix-Patterns (Kontrast, ARIA, Fokusmanagement) als Dev-Dokumentation hinterlegen.
 - **Security-Hardening:** Secret-Scanning (z. B. Gitleaks), Dependabot/renovate Integration.
 - **Observability:** Metriken (CloudWatch/S3) für QA-Trends je Branch/Team.
-

11) Rollen & Verantwortungen (für neuen Chat)

Als PO:

- Backlog-Pflege für Qualitätsthemen (A11y, Security, Tech Debt)
- Akzeptanzkriterien pro Story mit QA-Gates verankern
- Release-Freigabe nur bei *passed* QA-Status

Als CTO:

- Guardrails/Policies definieren (Severity-Limits, Coverage-Schwellwerte)

- Secret-Management & Compliance (AWS/Snyk/Sonar)
- Architekturentscheidungen (z. B. Austausch/Erweiterung Scanner, Kostensteuerung)

Checklisten (Schnellstart):

- **Neues Repo/Feature:** ENV prüfen → QA-Server startbar? `qa:full` läuft lokal?
- **PR-Review:** QA-Summary lesen → Blocks/Warnungen in Tasks umsetzen
- **Build-Fail:** Logs öffnen → betroffener Job → Report-Dateien ansehen → Owner/Team beauftragen

12) Glossar

- **QA:** Quality Assurance (Qualitätssicherung)
- **WCAG:** Web Content Accessibility Guidelines
- **Axe-core:** A11y-Test-Engine
- **Snyk:** Security-Scanner für Dependencies/Code
- **Sonar:** Code-Analyse/Quality Gate Plattform
- **Bedrock (AWS):** Foundation Model Hosting (hier: Claude für Code Review)

13) Nützliche Orte im Repo

- **QA-Orchestrierung:** `src/lib/quality-assurance/`
- **QA-Dashboard:** `src/components/quality-assurance/QADashboard.tsx`
- **QA-Testseite:** `src/pages/QATestPage.tsx`
- **QA-Server:** `scripts/qa-server.ts`
- **Reports:** `qa-reports/`
- **CI-Workflow:** `.github/workflows/quality-assurance.yml`
- **Tests (Unit/Comp):** `src/**/__tests__/_`

Ende der Übergabe

Dieser Bericht bildet den operativen Rahmen ab, damit der neue Chat den PO/CTO-Part für die QA-Architektur nahtlos übernehmen kann.

Übergabebericht (PO & CTO) – matbakh.app

****Datum:**** 2025-09-27

****Rollen-Übergabe an:**** neuer Chat (PO & CTO)

****Vision:**** **"10x besser"** für Nutzer – radikal kundenorientiert, sicher, skalierbar, kosteneffizient.*

1) Executive Summary

* ****Architektur-Stand:**** Multi-Region AWS (eu-central-1 / eu-west-1), ECS Fargate, App Mesh, Cloud Map, S3/ECR/CloudWatch, Secrets Manager.

* ****AI-Orchestration (Task 13 Scaffold):**** Bedrock + Google Gemini + Meta Llama via einheitlicher Router/Policy Engine, Bandit-Controller, Gateway (Fastify), Evidently für A/B.

* ****Frontend/DevX:**** React Dashboard & Hooks für Microservices; CI via Jest/ts-jest; CDK Stacks.

* ****Qualität/Testlage:**** Große Fortschritte bei CDK-Tests; verbleibende ****Unit/UI-Test-Gaps**** (Hook/Server), plus 3 HealthChecker-Tests (CloudWatch-Mocking).


* ****Nächste 2–3 Wochen (kritisch):**** Absicherung Security/Netz (Cross-Cloud KI-Zugriff), Streaming, Rate-Limiting/Quotas, SageMaker-Pipeline, Guardrails.

* ****PR-Serie:**** Inkrementell möglich (empfohlen), Feature-Flags + Dark-Launch, Tests pro PR (siehe Abschnitt 8).

2) Produkt-Leitplanken (PO)

* ****10x-Kriterium:**** Jede Funktion muss messbar ***einen Größenordnungs-Mehrwert*** liefern (Zeitersparnis, Erfolgsquote, Kosten/Inference).

* ****Customer-backlog:**** Alle Entscheide gegen reale User-Jobs-to-be-done spiegeln; Telemetrie-gestütztes A/B als Default.

* ****Qualitätsgates:**** (a) Security , (b) P95 Latenz <200ms extern / <50ms intern, (c) Error-Rate <1%, (d) Kosten pro Vorgang im Budget.

3) Technische Architektur (CTO)

3.1 Plattform

- * **Compute:** ECS Fargate (Cluster pro Env), Auto-Scaling (FARGATE/FARGATE_SPOT).
- * **Netz:** VPC (3 AZ, 1 NAT, Endpoints: ECR, Logs, S3, Bedrock, STS, Secrets).
- * **Mesh/Discovery:** App Mesh (Envoy mTLS), Cloud Map (svc.local).
- * **Edge/LB:** ALB (zugeschnittene Namen <32 Zeichen), HTTPS-Termination.
- * **Observability:** CloudWatch Logs/Metrics/Dashboards, X-Ray optional über ADOT.

3.2 AI-Orchestration (Multi-Provider)

- * **Router Policy Engine:** Score = f(Latenz, Kosten, Domäne, Tool-Needs, SLA, Budget-Tier).
- * **Adapters:**
 - * **Bedrock:** Anthropic Claude Tool-Use via Bedrock API.
 - * **Google:** Gemini Function Calling (REST, API-Key/Svc-Acc).
 - * **Meta:** Llama (Text/JSON-Modus), Prompt-Normalisierung.
- * **Bandit Controller:** Thompson Sampling (auch kontextuell: Domäne/Budget/Toolbedarf) + Evidently Metriken.
- * **Gateway (Fastify HTTP/gRPC):**

/v1/route

,

/v1/models

,

/v1/stats

, Admin-Ops.

* **A/B & Autoskalierung:** Evidently + ECS Scale-Policies.

> **Zielbild:** Bedrock bleibt „Default-Orchestrator“, kann aber Google/Meta dynamisch auswählen/chainen; Multi-Agent bereits vorhanden → Router orchestriert Tools/Agents.

4) Task-Status & offene Punkte

Task 11 – Multi-Region Health/Failover

* **Status:** FailoverManager ✓; HealthChecker teils offen.

* **Offene Tests:**

* **Repl. Lag Assertions:** erwartete

30000/120000 ms

, erhalten

0

.

* **Summary Metric:** erwartet

45000

, erhalten

0

.

* **Wahrscheinliche Ursache:** CloudWatch GetMetricData-Mock in Tests liefert keine Lags; Implementation setzt Details.replicationLag auf default 0.

* **Fix-Plan (kurzfristig):**

1. In Tests

```
mockGetMetricData()
```

für

```
AWS/RDS ReplicaLag
```

mit Samples füllen.

2. In Code

```
replicationLag
```

aus Response auf Service-Details setzen.

3. Für „no data“ →

```
undefined
```

& degrade statt

```
0
```

.

Task 12 – Microservices Foundation

* **Status:** Stack & Dashboard vorhanden.

* **CDK Tests:** Deprecations gefixt (ipAddresses, containerInsights), SG-Lookups ersetzt, ALB-Name gekürzt → neue Suite ****besteht lokal**** (Berichte).

* **Hook-Tests (useMicroservices) – FAIL:**

* **Fehler:**

```
ReferenceError: MicroserviceOrchestrator is not defined
```

.

* **Ursache:** fehlender Import/Export in

```
src/lib/microservices/index.ts
```

oder falscher Pfad/Named Export.

****Fix:****

```
import { MicroserviceOrchestrator } from './microservice-orchestrator'
```

+ Re-export in

```
index.ts
```

; Tests anpassen.

Task 13 – AI Orchestration (Scaffold fertig)

****Erreicht:**** CDK Stack, Router, Adapters (Bedrock/Google/Meta), Bandit, Gateway, 34 Tests grün.

****Gaps (Production Roadmap):****

****Security/Netz:**** Cross-Cloud egress → PrivateLink/Proxies, ausgehende IP-Kontrolle, Secret-Rotation, Provider-spezifische IAM/Keys.

****Streaming:**** Einheitliches SSE/Chunking über alle Provider.

****Limits/Quotas:**** Rate-Limiting global & per-Provider, Circuit-Breaker.

****Registry:**** Versionierte Model-Registry (DynamoDB + SSM ParamStore).

****Kosten:**** Präzise Cost Attribution pro Call.

****SageMaker:**** Trainings/Batch-Jobs + Feature Store + MLOps (Pipeline fehlt).

****Guardrails:**** Prompt/Response-Filter (Provider-agnostisch), Safety-Policies.

5) Akute Build-/Test-Fehler & Lösungen

5.1 Gateway-Server Tests

****Logs:****


```
hostname: "MacBookPro.fritz.box"
```

ist nur lokaler Hostname → unkritisch.

****Fehler 1:****

```
setImmediate is not defined
```

(Jest jsdom).

****Lösung A (empfohlen):****

```
testEnvironment: "node"
```

****nur für Server-Tests**.**

****Lösung B:**** Polyfill in

```
jest.setup.server.ts
```

:

```
ts
// Polyfill
// @ts-ignore
global.setImmediate ||= (fn: (...a:any[])⇒void, ...args:any[]) ⇒ setTimeout
(fn, 0);
```

****Fehler 2:**** „Reply was already sent ... return reply?“

****Fix:**** In Fastify-Handler ****immer****

```
return reply.code(200).send(payload);
```

verwenden.

5.2 useMicroservices Hook-Tests

****Fehler:****

MicroserviceOrchestrator is not defined

.
* **Quick-Fix:**

```
ts
// src/lib/microservices/index.ts
export { MicroserviceOrchestrator } from './microservice-orchestrator';
// + sicherstellen, dass createMicroserviceFoundation den Import nutzt
```

* **Test-Setup:** Mock für

AppMeshManager

,

ServiceDiscoveryManager

, Orchestrator Methoden (

scale

,

deploy

,

remove

), sowie Timer (fake timers) für Auto-Refresh.

5.3 HealthChecker Repl. Lag

* **Test-Mock:**

```
ts
mockGetMetricData({
```

```
id: 'replicaLag',  
values: [30_000], // oder 120_000  
timestamps: [new Date()]  
});
```

* **Code:** Mappe

```
MetricDataResults[]
```

→

```
details.replicationLag = max(values)
```

; Summary aggregiert Mittelwert.

6) Security & Compliance (kritisch)

* **Secrets:** AWS Secrets Manager (Rotation 60–90d), KMS CMKs per Env, Provider-Keys (Google/Meta) getrennt.

* **Egress-Kontrolle:** Nur whitelisted KI-Endpoints; NAT + egress-SG/Firewall, optional PrivateLink/Proxy (wo möglich).

* **PII/GDPR:** EU-Residency, Logging ohne PII, Redaction in Traces.

* **Guardrails:** Prompt-Filter (Regex/LLM-Classifer), Blocklisten, Safe-completion Policies, Audit-Trail.

7) Observability & SLOs

* **Dashboards:** Pro Service (Requests, Errors, Latency P95, CPU/Mem), pro Provider (Bedrock/Gemini/Llama).

* **Alarmer:** ErrorRate >5%, P95 >200ms, Throttle/429, Budget 80/95%, Bandit-Drift (Reward < Threshold).

* **Trace Correlation:** Request-ID → Model Call → Tool Calls.

8) PR-Serie (empfohlen)

PR-0 (Infra-Test-Stabilisierung)

* Jest split:

```
jest.server.config.cjs
```

(env=node),

```
jest.ui.config.cjs
```

(env=jsdom), SetupFiles getrennt.

* Fastify handlers

```
return reply.send(...)
```

.

* **Ziel:** Gateway/Auth/Hook Tests grün.

PR-1 (HealthChecker Fix)

* CloudWatch-Mock +

```
replicationLag
```

Mapping; 3 failing Tests grün.

* Summary-Metriken prüfen.

PR-2 (AI Streaming & Limits)

* SSE/Chunk über alle Provider, Global/Per-Provider Rate-Limiter, Retry/Backoff.

* Tests: Stream snapshot & backpressure.

PR-3 (Model Registry & Config)

- * DynamoDB + SSM für versionierte Model-Konfig; Admin API: publish/rollback.
- * Tests: Rollback/Canary.

****PR-4 (Guardrails & Safety)****

- * Prompt/Response Filter, Policy YAML, Blocklisten, Audit-Log.
- * Tests: Policy enforcement.

****PR-5 (SageMaker Pipeline)****

- * Training/Batch Inference, Feature Store, CI/CD (CodeBuild/CodePipeline).
- * Smoke/E2E Tests (mocked).

(Jeder PR: Δ Docs, Δ Runbooks, Δ Dashboards)

9) Commands & Pfade (Cheat Sheet)

*** **Unit/Integration Tests****

*** Health/Multi-Region:**

```
npm run test:mr:unit
```

*** CDK Microservices:**

```
npm run test:ms:cdk
```

*** Server/Auth:**

```
npm test -- --testPathPattern=gateway-server-auth.test.ts
```

*** **Start Gateway lokal:****

```
npm run dev:ai-gateway
```

(Port per

```
.env
```

```
)
```

```
* **CDK Deploy:**
```

```
* Foundation:
```

```
npx cdk deploy MicroservicesFoundationStack
```

```
* AI Orchestration:
```

```
npx cdk deploy AiOrchestrationStack
```

```
* **Logs lokal:** Fastify pino JSON (level, time, reqId)
```

```
**Pfad-Highlights**
```

```
*
```

```
infra/cdk/microservices-foundation-stack.ts
```

```
*
```

```
infra/cdk/ai-orchestration-stack.ts
```

```
*
```

```
src/lib/multi-region/*
```

```
(Failover/Health)
```

```
*
```

```
src/lib/ai-orchestrator/*
```

```
(Router, Adapters, Bandit, Gateway)
```

```
*
```

```
src/hooks/useMicroservices.ts
```

+ Tests

*

```
src/components/microservices/MicroservicesDashboard.tsx
```

10) Offene Risiken & Mitigation

* **Cross-Cloud KI-Abhängigkeit:** Provider-APIs/Quotas ändern → **Mitigation:** Registry/Feature-Flags, Fallback-Pfad je Provider.

* **Kosten-Volatilität:** Prompt-Länge/Tool-Calls → **Mitigation:** Budget-Tiers, Pre-flight Cost-Estimates, Hard-Caps.

* **Latenz/Verfügbarkeit:** Netzpfade/Provider-Region → **Mitigation:** Multi-Region, Retry mit Jitter, Local caching.

* **Recht/Safety:** UGC & KI-Outputs → **Mitigation:** Guardrails, Audit, Moderation-Pipeline.

11) Definition of Done & Quality Gates

* **DoD:**

1. Tests grün (Unit+Integration für betroffene Teile).
2. Dashboards/Alarme aktualisiert.
3. Runbook & Docs aktualisiert.
4. Security Review (Secrets/PII).
5. Kostenabschätzung dokumentiert.

* **Gates:** SAST/Lint, Type-strict, API-Contracts stabil, Backwards-compatible Flags.

12) Konkrete To-Dos (1–3 Tage)

1. **Jest-Split & Polyfills** (Server vs UI) + Fastify return-fix.
2. **Import-Fix**

MicroserviceOrchestrator

in

src/lib/microservices/index.ts

.

3. **CloudWatch-Mocks** für HealthChecker (ReplicaLag) + Mapping.
4. **.env.example** mit KI-Provider-Creds (ohne Secrets), Docs aktualisieren.
5. **Evidently Dashboards**: Bandit-KPIs sichtbar machen.

13) Entscheide/Beschlüsse (kurz)

- * **Tests zuerst fixen, dann PR-Serie starten.**
- * **Inkrementelle Ausrollung** (Dark-Launch/Feature-Flags) für AI-Orchestration.
- * **Bedrock als Default-Orchestrator**, aber **Google/Meta first-class** in Router.
- * **A/B autopilot**: Bandit kontrolliert Traffic, menschliches Eingreifen optional.

14) Kontaktpunkte & Verantwortlichkeiten

- * **Kiro**: Implementierung Task 13/14, Provider-Adapters, Bandit, Streaming, Registry.
- * **PO (neuer Chat)**: Priorisierung Roadmap, KPI-Definition, A/B-Hypothesen.
- * **CTO (neuer Chat)**: Security/Netz, Kosten, SLOs, Architekturreviews, Releases.

15) Anhänge (im Repo)

*

task-11-multi-region-tests-completion-report.md

*

cdk-infrastructure-tests-repair-completion-report.md

*

README_ai-orchestration.md

(API, Adapter, Bandit, Examples)

*

RUNBOOK_failover.md

,

RUNBOOK_ai-gateway.md

Schlusswort

Alles Wesentliche ist vorbereitet. Die ****nächsten PRs**** stabilisieren Tests, schließen Security/Streaming-Lücken und führen die ****A/B-Autopilot-Orchestrierung**** in Produktion.

Wenn du übernimmst: starte mit ****PR-0/1****, halte die Gates ein – und bleib dem ***10x-Nutzerwert*** treu.