Nightlife OS - Architektur-Dokumentation

Multi-Mandanten-SaaS-Plattform für Club-Management

Version: 2.0
Erstellt am: 1. Dezember 2025
Basis: Demo-System (DJ-Konsole + Besucher-App)
Ziel: Skalierbare Multi-Club-SaaS-Plattform

Inhaltsverzeichnis

1. MONOREPO-ARCHITEKTUR
   - 1.1 Verzeichnisstruktur
   - 1.2 Technologie-Stack
2. FIRESTORE-DATENMODELL
   - 2.1 Plattformebene (Root-Level)
   - 2.2 Club-Ebene
3. ROLLEN- & RECHTE-MODELL
   - 3.1 Rollen-Definitionen
   - 3.2 Berechtigungsmatrix
   - 3.3 Firestore Security Rules
4. MIGRATIONSPLAN
   - 4.1 Bestehende Collections
   - 4.2 Migrationsstrategie
   - 4.3 Code-Migration
5. MEHRSPRACHIGKEIT (i18n)
   - 5.1 Struktur der JSON-Files
   - 5.2 Zugriff in Components
   - 5.3 Sprach-Speicherung
6. ZUSAMMENFASSUNG & NÄCHSTE SCHRITTE

---

1. MONOREPO-ARCHITEKTUR

1.1 Verzeichnisstruktur

Das Nightlife OS ist als **Monorepo** organisiert, um Code-Sharing zwischen allen Anwendungen zu maximieren und die Wartbarkeit zu gewährleisten.

Komplette Verzeichnisstruktur

```
nightlife_os/
├── apps/                                  # Alle eigenständigen Anwendungen
│   ├── club-app/                          # 🎵 Besucher-PWA (Progressive Web App)
│   │   ├── src/
│   │   │   ├── app/                       # Next.js App Router
│   │   │   │   ├── layout.tsx             # Root Layout mit Providers
│   │   │   │   ├── page.tsx               # Startseite (Login/Main)
│   │   │   │   └── manifest.json          # PWA Manifest
│   │   │   ├── components/                # React-Komponenten
│   │   │   │   ├── app-container.tsx      # Hauptcontainer mit Auth & State
│   │   │   │   ├── login-screen.tsx       # Login/Register
│   │   │   │   ├── home-view.tsx          # Startseite (QR-Code, Status)
│   │   │   │   ├── tab-navigation.tsx     # Bottom Navigation
│   │   │   │   ├── global-overlay.tsx     # Lichtshow/Messages/Countdown
│   │   │   │   ├── chat-system.tsx        # Chat-Hauptkomponente
│   │   │   │   ├── error-boundary.tsx     # Error Handling
│   │   │   │   └── chat/                  # Chat-Subkomponenten
│   │   │   │       ├── chat-list-view.tsx
│   │   │   │       ├── chat-room.tsx
│   │   │   │       ├── chat-add-friend-view.tsx
│   │   │   │       ├── chat-group-create-view.tsx
│   │   │   │       ├── chat-group-manage-view.tsx
│   │   │   │       └── ephemeral-image.tsx
│   │   │   ├── lib/                       # Utility-Funktionen
│   │   │   │   ├── qr-code.ts             # QR-Code-Generierung
│   │   │   │   └── image-compression.ts   # Bildkompression
│   │   │   └── styles/
│   │   │       └── globals.css            # Globale Styles
│   │   ├── public/                        # Statische Assets
│   │   │   ├── icons/                     # PWA Icons
│   │   │   └── sw.js                      # Service Worker
│   │   ├── package.json
│   │   ├── next.config.js
│   │   ├── tailwind.config.ts
│   │   └── tsconfig.json
│   │
│   ├── dj-console/                        # 🎛️ DJ/Lichtjockey-Steuerung
│   │   ├── src/
│   │   │   ├── app/
│   │   │   │   ├── layout.tsx
│   │   │   │   └── page.tsx
│   │   │   ├── components/
│   │   │   │   ├── admin-login.tsx        # Login für DJ/LJ
│   │   │   │   ├── admin-dashboard.tsx    # Hauptsteuerung
│   │   │   │   ├── light-control.tsx      # Farbauswahl + Effekte
│   │   │   │   ├── audio-sync.tsx         # Mikrofon-Sync
│   │   │   │   ├── lottery-system.tsx     # Gewinnspiel
│   │   │   │   ├── broadcast-messages.tsx # Nachrichten senden
│   │   │   │   └── guest-list.tsx         # Eingecheckte Gäste
│   │   │   ├── hooks/
│   │   │   │   ├── use-audio-analyzer.ts  # Web Audio API
│   │   │   │   └── use-guest-list.ts      # Realtime User-Listener
│   │   │   └── styles/
│   │   │       └── globals.css
│   │   ├── package.json
│   │   ├── next.config.js
│   │   ├── tailwind.config.ts
│   │   └── tsconfig.json
│   │
│   ├── club-admin/                        # 🏢 Club-Owner/Admin Dashboard
│   │   ├── src/
│   │   │   ├── app/
```

```
│   │   │   │   ├── layout.tsx
│   │   │   │   ├── page.tsx              # Dashboard-Übersicht
│   │   │   │   ├── settings/             # Club-Einstellungen
│   │   │   │   │   └── page.tsx
│   │   │   │   ├── staff/                # Personal-Verwaltung
│   │   │   │   │   └── page.tsx
│   │   │   │   ├── analytics/            # Statistiken
│   │   │   │   │   └── page.tsx
│   │   │   │   └── subscription/         # Abo-Verwaltung
│   │   │   │       └── page.tsx
│   │   │   ├── components/
│   │   │   │   ├── dashboard-overview.tsx
│   │   │   │   ├── staff-manager.tsx     # Personal hinzufügen/entfernen
│   │   │   │   ├── club-settings.tsx     # Farben, Features, Öffnungszeiten
│   │   │   │   ├── analytics-charts.tsx  # Charts mit Recharts/Plotly
│   │   │   │   └── subscription-panel.tsx
│   │   │   └── styles/
│   │   │       └── globals.css
│   │   ├── package.json
│   │   └── tsconfig.json
│   │
│   ├── staff-door/                       # 🚪 Türsteher-App
│   │   ├── src/
│   │   │   ├── app/
│   │   │   │   ├── layout.tsx
│   │   │   │   └── page.tsx
│   │   │   ├── components/
│   │   │   │   ├── door-login.tsx
│   │   │   │   ├── qr-scanner.tsx        # Gäste-QR-Code scannen
│   │   │   │   ├── user-verify.tsx       # Trust-Level prüfen
│   │   │   │   ├── manual-checkin.tsx    # Manueller Check-In
│   │   │   │   └── blacklist-check.tsx   # Blacklist-Prüfung
│   │   │   └── styles/
│   │   │       └── globals.css
│   │   ├── package.json
│   │   └── tsconfig.json
│   │
│   ├── staff-waiter/                     # 🍸 Kellner/Bar-App
│   │   ├── src/
│   │   │   ├── app/
│   │   │   │   ├── layout.tsx
│   │   │   │   └── page.tsx
│   │   │   ├── components/
│   │   │   │   ├── waiter-login.tsx
│   │   │   │   ├── order-list.tsx        # Offene Bestellungen
│   │   │   │   ├── table-map.tsx         # Tischplan
│   │   │   │   ├── order-create.tsx      # Neue Bestellung
│   │   │   │   └── payment-process.tsx   # Bezahlung abschließen
│   │   │   └── styles/
│   │   │       └── globals.css
│   │   ├── package.json
│   │   └── tsconfig.json
│   │
│   └── staff-cloakroom/                  # 🧥 Garderoben-App
│       ├── src/
│       │   ├── app/
│       │   │   ├── layout.tsx
│       │   │   └── page.tsx
│       │   ├── components/
│       │   │   ├── cloakroom-login.tsx
│       │   │   ├── ticket-scan.tsx       # QR-Code scannen
│       │   │   ├── item-checkin.tsx      # Gegenstand einlagern
│       │   │   └── item-checkout.tsx     # Gegenstand ausgeben
```

```
│       │   │   └── ticket-print.tsx      # Ticket drucken
│       │   └── styles/
│       │       └── globals.css
│       ├── package.json
│       └── tsconfig.json
│
├── packages/                              # Shared Packages (intern)
│   ├── core/                              # 🔥 Firebase, Hooks, Utils
│   │   ├── src/
│   │   │   ├── firebase/
│   │   │   │   ├── init.ts                # Firebase initialisieren
│   │   │   │   ├── auth.ts                # Auth-Helpers
│   │   │   │   ├── firestore.ts           # Firestore-Helpers
│   │   │   │   └── storage.ts             # Storage-Helpers
│   │   │   ├── hooks/
│   │   │   │   ├── use-auth.ts            # Auth-State Hook
│   │   │   │   ├── use-user-data.ts       # User-Doc Realtime
│   │   │   │   ├── use-club-state.ts      # Club-State Realtime
│   │   │   │   ├── use-friends.ts         # Freunde-Liste
│   │   │   │   ├── use-chats.ts           # Chat-Liste
│   │   │   │   └── use-i18n.ts            # Mehrsprachigkeit
│   │   │   ├── utils/
│   │   │   │   ├── friend-code.ts         # Code-Generator
│   │   │   │   ├── trust-score.ts         # Trust-Level-Berechnung
│   │   │   │   ├── validation.ts          # Input-Validierung
│   │   │   │   └── date-time.ts           # Datums-Formatting
│   │   │   ├── constants/
│   │   │   │   ├── roles.ts               # Rollen-Definitionen
│   │   │   │   ├── permissions.ts         # Berechtigungen
│   │   │   │   └── app-config.ts          # Globale Konfiguration
│   │   │   └── index.ts                   # Exports
│   │   ├── package.json
│   │   └── tsconfig.json
│   │
│   ├── ui/                                # 🎨 UI-Komponenten, Theming, i18n
│   │   ├── src/
│   │   │   ├── components/
│   │   │   │   ├── button.tsx             # Standard-Button
│   │   │   │   ├── input.tsx              # Input-Felder
│   │   │   │   ├── card.tsx               # Karten-Layout
│   │   │   │   ├── modal.tsx              # Modal-Dialog
│   │   │   │   ├── icon.tsx               # Icon-Wrapper (Lucide)
│   │   │   │   ├── loader.tsx             # Loading-Spinner
│   │   │   │   ├── toast.tsx              # Toast-Notifications
│   │   │   │   └── qr-code-display.tsx    # QR-Code-Komponente
│   │   │   ├── locales/                   # i18n JSON-Files
│   │   │   │   ├── de.json                # Deutsch
│   │   │   │   ├── en.json                # Englisch
│   │   │   │   ├── fr.json                # Französisch
│   │   │   │   ├── es.json                # Spanisch
│   │   │   │   └── it.json                # Italienisch
│   │   │   ├── theme/
│   │   │   │   ├── colors.ts              # Farbpaletten
│   │   │   │   ├── typography.ts          # Schriftarten
│   │   │   │   └── tailwind-preset.ts     # Tailwind-Config
│   │   │   └── index.ts
│   │   ├── package.json
│   │   └── tsconfig.json
│   │
│   └── shared-types/                      # 📦 TypeScript-Typen
│       ├── src/
│       │   ├── user.ts                    # User-Typen
│       │   └── club.ts                    # Club-Typen
```

```
│       │   ├── chat.ts                    # Chat-Typen
│       │   ├── order.ts                   # Order-Typen
│       │   ├── cloakroom.ts               # Garderoben-Typen
│       │   ├── roles.ts                   # Rollen-Typen
│       │   ├── api.ts                     # API-Response-Typen
│       │   └── index.ts                   # Exports
│       ├── package.json
│       └── tsconfig.json
│
├── package.json                           # Root Package (Workspace)
├── pnpm-workspace.yaml                    # PNPM Workspace Config
├── turbo.json                             # Turborepo Config
├── tsconfig.json                          # Base TypeScript Config
├── .gitignore
├── .env.example                           # Beispiel-Umgebungsvariablen
├── README.md
├── ARCHITECTURE.md                        # Diese Datei
├── DEPLOYMENT.md                          # Deployment-Guide
└── FIRESTORE_SCHEMA.md                    # Datenbank-Schema
```

Zweck der einzelnen Apps

App	Zweck	Hauptnutzer	Plattform
club-app	Besucher-App mit Chat, Check-In, Freunde, Lichtshow-Display	Gäste	Mobile (PWA)
dj-console	Steuerung von Licht, Musik-Sync, Gewinnspiele, Broadcasts	DJ/Lichtjockey	Tablet/Desktop
club-admin	Club-Verwaltung, Personal, Analytics, Abo-Verwaltung	Club-Owner/Admin	Desktop
staff-door	Türsteher-App mit QR-Scanner, Trust-Verifizierung	Türsteher	Tablet/Mobile
staff-waiter	Bestellungs-Management, Tischplan	Kellner/Bar	Tablet/Mobile
staff-cloakroom	Garderoben-Verwaltung mit Ticket-System	Garderoben-Personal	Tablet

Zweck der Packages

Package	Zweck	Verwendet von
core	Firebase-Integration, Hooks, Utils, Konstanten	Alle Apps
ui	Wiederverwendbare UI-Komponenten, i18n, Theming	Alle Apps
shared-types	TypeScript-Typen für konsistente Datenstrukturen	Alle Apps + Packages

---

1.2 Technologie-Stack

Framework: Next.js 14+ (App Router)

Begründung:
- ✅ **Server-Side Rendering (SSR):** Schnelle Initial Loads für PWA
- ✅ **API Routes:** Kann Backend-Funktionen integrieren (z.B. Stripe-Webhooks)
- ✅ **File-based Routing:** Intuitive Struktur für Multi-Page-Apps
- ✅ **Optimierungen:** Image-Optimization, Code-Splitting out-of-the-box
- ✅ **PWA-Support:** Mit `next-pwa` einfach integrierbar
- ✅ **TypeScript First:** Native TS-Unterstützung
- ✅ **Vercel Deployment:** Optimale Performance (wenn gewünscht)

Alternative Vite:
- ❌ Kein SSR out-of-the-box
- ❌ Kein API-Routes-System
- ✅ Schnellerer Dev-Server (aber Next.js Turbopack ist auch sehr schnell)

Entscheidung: Next.js für Production-Grade Features

Monorepo-Tool: Turborepo

Begründung:
- ✅ **Optimierte Builds:** Intelligentes Caching basierend auf Dependencies
- ✅ **Parallel Execution:** Baut alle Apps gleichzeitig
- ✅ **Pipeline-Config:** Klare Task-Dependencies (build → test → lint)
- ✅ **Vercel Integration:** Vom Next.js-Team entwickelt
- ✅ **Einfache Konfiguration:** `turbo.json` ist sehr übersichtlich
- ✅ **Remote Caching:** Kann Build-Caches im Team teilen

Alternative Nx:
- ❌ Komplexere Konfiguration
- ❌ Mehr Overhead für kleinere Projekte
- ✅ Bessere Code-Generatoren

Entscheidung: Turborepo für Einfachheit und Geschwindigkeit

turbo.json Beispiel:

```
{
  "$schema": "https://turbo.build/schema.json",
  "pipeline": {
    "build": {
      "dependsOn": ["^build"],
      "outputs": [".next/**", "dist/**"]
    },
    "lint": {
      "dependsOn": ["^lint"]
    },
    "test": {
      "dependsOn": ["^build"]
    },
    "dev": {
      "cache": false
    }
  }
}
```

State Management: React Context + Firebase Realtime

Begründung:
- ✅ **Keine zusätzliche Library:** React Context reicht für User-State
- ✅ **Firebase als Source of Truth:** Alle Daten kommen aus Firestore
- ✅ **Realtime Updates:** `onSnapshot` für Live-Synchronisation
- ✅ **Optimistic Updates:** Lokale State-Updates vor Firebase-Confirm

Nicht benötigt:
- ❌ Redux: Zu viel Boilerplate für Realtime-App
- ❌ Zustand: Firebase-State ist bereits global
- ❌ Recoil: Zu komplex für diesen Use-Case

State-Architektur:

```
Global Context (AuthContext)
├── user: FirebaseUser | null
├── userData: UserDocument | null
└── clubId: string

Component-Level State
├── UI-State (Tabs, Modals, Forms)
└── Local Caching (optimistic updates)

Firebase Realtime Listeners
├── onSnapshot(users/{uid}) → userData
├── onSnapshot(clubs/{clubId}/state/global) → clubState
└── onSnapshot(clubs/{clubId}/chats) → chats
```

UI-Framework: Tailwind CSS + Shadcn/ui

Begründung:
- ✅ **Tailwind CSS:** Utility-First, schnelle Entwicklung, konsistentes Design
- ✅ **Shadcn/ui:** Kopierbare Komponenten statt Library (keine Vendor Lock-in)
- ✅ **Headless UI:** Für komplexe Komponenten (Modals, Dropdowns)
- ✅ **Lucide React:** Moderne Icon-Library (bereits im Demo verwendet)
- ✅ **Recharts/Plotly:** Für Analytics-Charts im Admin-Dashboard

Component Library:

```
// packages/ui/src/components/button.tsx
import { cva, type VariantProps } from 'class-variance-authority'

const buttonVariants = cva(
  'inline-flex items-center justify-center rounded-md font-medium transition-colors',
  {
    variants: {
      variant: {
        default: 'bg-cyan-600 text-white hover:bg-cyan-700',
        ghost: 'hover:bg-slate-800',
        danger: 'bg-red-600 text-white hover:bg-red-700'
      },
      size: {
        sm: 'h-9 px-3 text-sm',
        md: 'h-10 px-4 text-base',
        lg: 'h-12 px-6 text-lg'
      }
    },
    defaultVariants: {
      variant: 'default',
      size: 'md'
    }
  }
)
```

Testing: Vitest + React Testing Library + Playwright

Begründung:
- ✅ **Vitest:** Schneller als Jest, native ESM-Support
- ✅ **React Testing Library:** Best Practice für Component-Tests
- ✅ **Playwright:** E2E-Tests für kritische User-Flows
- ✅ **MSW (Mock Service Worker):** Mock Firebase Calls in Tests

Test-Strategie:

```
Unit Tests (Vitest)
├── packages/core/utils/*.test.ts
├── packages/core/hooks/*.test.ts
└── apps/*/components/*.test.tsx

Integration Tests (Vitest + RTL)
├── Chat-System vollständig
├── Check-In/Out Flow
└── Friend-Request Flow

E2E Tests (Playwright)
├── User Registration → Login → Check-In
├── DJ Console → Lichtshow → User sieht Overlay
└── Chat senden → Empfänger erhält Nachricht
```

Weitere Tools

Tool	Zweck	Begründung
TypeScript	Typsicherheit	Production-Grade Apps brauchen Types
ESLint + Prettier	Code-Quality	Konsistenter Code-Style
Husky + Lint-Staged	Pre-Commit Hooks	Verhindert fehlerhafte Commits
pnpm	Package Manager	Schneller als npm, weniger Speicher als yarn
Firebase SDK v10	Backend	Bereits im Demo verwendet
html5-qrcode	QR-Scanner	Bereits im Demo verwendet
QRCode.js	QR-Generierung	Bereits im Demo verwendet
Compressor.js	Bild-Kompression	Bereits im Demo verwendet
Zod	Runtime-Validierung	Validierung von Firebase-Daten
Date-fns	Datums-Formatting	Leichtgewichtig vs. Moment.js

---

2. FIRESTORE-DATENMODELL

2.1 Plattformebene (Root-Level)

Die Plattformebene enthält **mandanten-übergreifende** Daten, die für das SaaS-System benötigt werden.

Collection: `platform/clubs/{clubId}`

Zweck: Stammdaten aller Clubs auf der Plattform

Feldname	Datentyp	Beschreibung	Beispielwert
`clubId`	`string`	Eindeutige Club-ID (Document-ID)	`"club_abc123"`
`name`	`string`	Offizieller Club-Name	`"Matrix Club Berlin"`
`slug`	`string`	URL-freundlicher Name	`"matrix-berlin"`
`groupId`	`string | null`	Zugehörige Club-Gruppe	`"group_xyz"`
`ownerId`	`string`	Firebase UID des Besitzers	`"user_owner123"`
`subscriptionTier`	`string`	Abo-Stufe	`"pro"`, `"enterprise"`
`subscriptionStatus`	`string`	Abo-Status	`"active"`, `"paused"`, `"cancelled"`
`subscriptionEndsAt`	`number`	Unix-Timestamp (ms)	`1704067200000`
`address`	`object`	Adresse	`{ street, city, zip, country }`
`contact`	`object`	Kontaktdaten	`{ email, phone, website }`
`openingHours`	`object`	Öffnungszeiten	`{ monday: "22:00-05:00", ... }`
`capacity`	`number`	Max. Gästeanzahl	`500`
`features`	`string[]`	Aktivierte Features	`["chat", "lightshow", "orders"]`
`theme`	`object`	Branding	`{ primaryColor, secondaryColor, logo }`
`trustedMode`	`boolean`	Trust-System aktiv?	`true`
`language`	`string`	Standard-Sprache	`"de"`
`createdAt`	`number`		`1701388800000`

Feldname	Datentyp	Beschreibung	Beispielwert
		Erstellungs-Timestamp	
`updatedAt`	`number`	Letztes Update	`1701388800000`

Security Rules:

```
// Nur Super-Admins können alle Clubs lesen
// Club-Admins können nur ihren eigenen Club lesen
allow read: if request.auth != null &&
  (isSuper Admin() || resource.data.ownerId == request.auth.uid);

// Nur Super-Admins können Clubs erstellen/löschen
allow create, delete: if isSuperAdmin();

// Club-Admins können ihren eigenen Club updaten (außer subscriptionTier)
allow update: if request.auth.uid == resource.data.ownerId &&
  !
request.resource.data.diff(resource.data).affectedKeys().hasAny(['subscriptionTier']);
```

Collection: `platform/groups/{groupId}`

Zweck: Club-Ketten (mehrere Clubs unter einem Dach)

Feldname	Datentyp	Beschreibung	Beispielwert
`groupId`	`string`	Eindeutige Gruppen-ID	`"group_xyz"`
`name`	`string`	Gruppen-Name	`"Nachtwerk Group"`
`ownerId`	`string`	Firebase UID des Besitzers	`"user_owner123"`
`clubIds`	`string[]`	Alle Clubs in der Gruppe	`["club_abc", "club_def"]`
`sharedFeatures`	`string[]`	Features für alle Clubs	`["cent-ral_analytics"]`
`createdAt`	`number`	Erstellungs-Timestamp	`1701388800000`

Collection: `platform/users/{uid}`

Zweck: Globaler User-Stamm (plattformweit)

Feldname	Datentyp	Beschreibung	Beispielwert
`uid`	`string`	Firebase Auth UID	`"user_abc123"`
`email`	`string`	Email-Adresse	`"user@example.com"`
`displayName`	`string | null`	Anzeigename	`"Max Mustermann"`
`photoURL`	`string | null`	Profilbild-URL	`` `”https://pbs.twimg.com/profile_images/1475176970349527047/gHvVahDG.jpg ``
`createdAt`	`number`	Registrierungs-Timestamp	`1701388800000`
`lastSeenAt`	`number`	Letzter Login	`1701388800000`
`isPlatformAdmin`	`boolean`	Super-Admin?	`false`
`ownedClubs`	`string[]`	Clubs, die der User besitzt	`["club_abc"]`
`memberClubs`	`string[]`	Clubs, in denen User Mitglied ist	`["club_abc", "club_xyz"]`

Hinweis: Dies ist der **globale** User-Account. Die club-spezifischen Daten (Rolle, Check-In, etc.) liegen in `clubs/{clubId}/users/{uid}`.

2.2 Club-Ebene

Die Club-Ebene enthält **club-spezifische** Daten. Jeder Club ist isoliert.

Collection: `clubs/{clubId}/users/{uid}`

Zweck: User im Kontext eines spezifischen Clubs

Feldname	Datentyp	Beschreibung	Beispielwert
`uid`	`string`	Firebase Auth UID	`"user_abc123"`
`email`	`string`	Email (kopiert aus Auth)	`"user@example.com"`
`displayName`	`string | null`	Anzeigename	`"Max Mustermann"`
`photoURL`	`string | null`	Profilbild	`”https://pbs.twimg.com/profile_images/1869027156287803392/LKsHuvw_.jpg`
`roles`	`string[]`	Rollen in diesem Club	`["guest"]`, `["staff", "door"]`
`checkedIn`	`boolean`	Im Club eingecheckt?	`true`
`checkedInAt`	`number | null`	Check-In-Timestamp	`1701388800000`
`lastSeen`	`number`	Letzter App-Zugriff	`1701388800000`
`language`	`string`	Bevorzugte Sprache	`"de"`, `"en"`
`friendCode`	`string`	7-stelliger Friend-Code	`"ABXY489"`

Trust-System-Felder:

Feldname	Datentyp	Beschreibung	Beispielwert
`phoneNumber`	`string | null`	Verifizierte Telefonnummer	`"+491234567890"`
`phoneVerified`	`boolean`	Telefon verifiziert?	`true`
`deviceIdHash`	`string | null`	Hash der Device-ID	`"sha256..."`
`faceHash`	`string | null`	Hash des Gesichts (FaceID)	`"sha256..."`
`trustedLevel`	`number`	Trust-Score (0-100)	`75`
`verifiedBy`	`string | null`	UID des Staff, der verifiziert hat	`"staff_user123"`
`verifiedAt`	`number | null`	Verifizierungs-Timestamp	`1701388800000`
`blacklisted`	`boolean`	Auf Blacklist?	`false`
`blacklistReason`	`string | null`	Grund für Blacklist	`"Störendes Verhalten"`

Check-In-Historie:

Feldname	Datentyp	Beschreibung	Beispielwert
`visitCount`	`number`	Anzahl Besuche	`12`
`lastVisits`	`number[]`	Timestamps der letzten 10 Besuche	`[1701388800000, ...]`

Freundschafts-System:

| Feldname | Datentyp | Beschreibung | Beispielwert |

|----------|----------|--------------|--------------|
| `friendIds` | `string[]` | UIDs der Freunde (denormalisiert für Queries) | `["friend1", "friend2"]` |

SubCollections:
- `clubs/{clubId}/users/{uid}/friends/{friendId}` - Freundschafts-Dokumente
- `clubs/{clubId}/users/{uid}/requests/{requesterId}` - Freundschaftsanfragen

Collection: `clubs/{clubId}/state/global`

Zweck: Globale Club-Zustände (Lichtshow, Countdown, Games, etc.)

Dokument-ID: `global` (Fixed Document)

Feldname	Datentyp	Beschreibung	Beispielwert
`mode`	`string`	Aktueller Modus	`"normal"`, `"light-show"`, `"message"`, `"countdown"`, `"lot-tery_result"`

Lichtshow-Felder:
Feldname	Datentyp	Beschreibung	Beispielwert
`lightColor`	`string | null`	Aktuelle Farbe	`"#ff0000"`
`lightEffect`	`string | null`	Effekt-Typ	`"color"`, `"strobe"`, `"psychedelic"`, `"audio_sync"`
`audioSyncIntensity`	`number | null`	Lautstärke-Level (0-255)	`150`

Countdown-Felder:
Feldname	Datentyp	Beschreibung	Beispielwert
`countdownActive`	`boolean`	Countdown läuft?	`true`
`countdownEnd`	`number | null`	Ziel-Timestamp	`1701388800000`
`countdownMessage`	`string | null`	Nachricht	`"VERLOSUNG"`

Gewinnspiel-Felder:
Feldname	Datentyp	Beschreibung	Beispielwert
`activeGame`	`string | null`	Spiel-Typ	`"lottery"`, `null`
`winnerIds`	`string[]`	UIDs der Gewinner	`["user1", "user2"]`
`prizeCode`	`string | null`	Gewinn-Code	`"FREIGETRÄNK"`

Broadcast-Message-Felder:
Feldname	Datentyp	Beschreibung	Beispielwert
`messageText`	`string | null`	Broadcast-Nachricht	`"HAPPY HOUR JETZT!"`
`messageTarget`	`string | null`	Zielgruppe	`"in"`, `"out"`, `"all"`

Security Rules:

```
// Alle Users können lesen (für Realtime-Updates)
allow read: if request.auth != null && isClubMember(clubId);

// Nur DJ/Admin kann schreiben
allow write: if request.auth != null &&
  (hasRole(clubId, 'admin') || hasRole(clubId, 'dj'));
```

Collection: `clubs/{clubId}/chats/{chatId}`

Zweck: Chat-Metadaten (1:1 und Gruppen)

Chat-ID-Format:
- 1:1: `{uid1}_{uid2}` (alphabetisch sortiert)
- Gruppe: Auto-generierte Firestore-ID

Feldname	Datentyp	Beschreibung	Beispielwert
`chatId`	`string`	Chat-ID	`"user1_user2"`, `"chat_abc123"`
`type`	`string`	Chat-Typ	`"private"`, `"group"`
`name`	`string | null`	Gruppen-Name (nur bei type=group)	`"Party-Crew"`
`participants`	`string[]`	UIDs der Teilnehmer	`["user1", "user2", "user3"]`
`createdBy`	`string | null`	Creator (nur bei Gruppen)	`"user1"`
`createdAt`	`number`	Erstellungs-Timestamp	`1701388800000`
`lastMessageAt`	`number`	Letzter Nachricht-Timestamp	`1701388800000`
`lastMessagePreview`	`string | null`	Vorschau der letzten Nachricht	`"Hallo! 👋"`

Security Rules:

```
// Nur Teilnehmer können lesen
allow read: if request.auth != null &&
  request.auth.uid in resource.data.participants;

// 1:1-Chats: Beide Teilnehmer können erstellen
// Gruppen: Creator kann erstellen
allow create: if request.auth != null &&
  request.auth.uid in request.resource.data.participants;

// Nur Creator kann Gruppe updaten
allow update: if request.auth != null &&
  (resource.data.type == 'private' ||
   resource.data.createdBy == request.auth.uid);

// Nur Creator kann Gruppe löschen
allow delete: if request.auth != null &&
  resource.data.createdBy == request.auth.uid;
```

Collection: `clubs/{clubId}/chats/{chatId}/messages/{messageId}`

Zweck: Chat-Nachrichten

Feldname	Datentyp	Beschreibung	Beispielwert
`messageId`	`string`	Nachricht-ID	Auto-generiert
`text`	`string`	Nachrichtentext	`"Hallo! 👋"`
`sender`	`string`	UID des Senders	`"user1"`
`senderName`	`string`	Anzeigename des Senders	`"Max"`
`image`	`string | null`	Base64-Bild oder Storage-URL	`"..."`
`ephemeral`	`number | null`	Sekunden bis Auto-Löschung	`5` , `null`
`viewedBy`	`string[]`	UIDs, die Nachricht gesehen haben	`["user1", "user2"]`
`createdAt`	`number`	Sende-Timestamp	`1701388800000`
`expiresAt`	`number | null`	Ablauf-Timestamp (bei ephemeral)	`1701388805000`
`deleted`	`boolean`	Gelöscht?	`false`

Security Rules:

```
// Nur Teilnehmer des Chats können lesen
allow read: if request.auth != null &&
  request.auth.uid in get(/databases/$(database)/documents/clubs/$(clubId)/chats/$
(chatId)).data.participants;

// Teilnehmer können Nachrichten senden
allow create: if request.auth != null &&
  request.auth.uid in get(/databases/$(database)/documents/clubs/$(clubId)/chats/$
(chatId)).data.participants &&
  request.resource.data.sender == request.auth.uid;

// Sender kann eigene Nachricht löschen
allow update, delete: if request.auth != null &&
  resource.data.sender == request.auth.uid;
```

Collection: `clubs/{clubId}/orders/{orderId}`

Zweck: Bestellungen (für Kellner/Bar-App)

Feldname	Datentyp	Beschreibung	Beispielwert
`orderId`	`string`	Bestellungs-ID	Auto-generiert
`userId`	`string | null`	UID des Bestellers (falls eingeloggt)	`"user1"`
`table`	`string | number`	Tisch-Nummer	`"A5"`, `12`
`items`	`object[]`	Bestellte Items	`[{ name: "Bier", qty: 2, price: 4.50 }]`
`totalPrice`	`number`	Gesamtpreis (EUR)	`9.00`
`status`	`string`	Status	`"open"`, `"preparing"`, `"served"`, `"paid"`
`createdBy`	`string`	UID des Kellners	`"staff_waiter1"`
`createdAt`	`number`	Erstellungs-Timestamp	`1701388800000`
`updatedAt`	`number`	Letztes Update	`1701388800000`
`paidAt`	`number | null`	Bezahlt-Timestamp	`1701388900000`
`paymentMethod`	`string | null`	Zahlungsart	`"cash"`, `"card"`, `"app"`

Security Rules:

```
// Nur Staff kann Orders lesen/schreiben
allow read, write: if request.auth != null &&
  hasAnyRole(clubId, ['staff', 'waiter', 'bar', 'admin']);
```

Collection: `clubs/{clubId}/cloakroom/{ticketId}`

Zweck: Garderoben-Tickets

Feldname	Datentyp	Beschreibung	Beispielwert
`ticketId`	`string`	Ticket-Nummer	`"T-001234"`
`userId`	`string | null`	UID des Gastes (falls eingeloggt)	`"user1"`
`itemDescription`	`string`	Beschreibung	`"Schwarze Lederjacke"`
`depositedAt`	`number`	Abgabe-Timestamp	`1701388800000`
`retrievedAt`	`number | null`	Ausgabe-Timestamp	`1701395000000`
`depositedBy`	`string`	UID des Garderoben-Staff	`"staff_cloak1"`
`retrievedBy`	`string | null`	UID des Ausgabe-Staff	`"staff_cloak1"`
`status`	`string`	Status	`"deposited"`, `"retrieved"`, `"lost"`
`notes`	`string | null`	Notizen	`"Wertgegenstand in Tasche"`

Security Rules:

```
// Nur Cloakroom-Staff kann lesen/schreiben
allow read, write: if request.auth != null &&
  hasAnyRole(clubId, ['staff', 'cloakroom', 'admin']);
```

Collection: `clubs/{clubId}/config`

Zweck: Club-Einstellungen (Feature-Flags, Farben, Öffnungszeiten)

Dokument-ID: `settings` (Fixed Document)

Feldname	Datentyp	Beschreibung	Beispielwert
`features`	`object`	Feature-Flags	`{ chat: true, light-show: true, orders: false }`
`theme`	`object`	Branding	`{ primaryColor: "#00ffff", logo: "https://res.cloudinary.com/vistaprint/images/w_1000,h_600,c_scale/f_auto,q_auto/v1705580305/ideas-and-advice-prod/en-us/fea-tured_14223857a51/fea-tured_14223857a51.png?_i=AA" }`
`openingHours`	`object`	Öffnungszeiten	`{ friday: "22:00-05:00", ... }`
`checkInRadius`	`number`	Geo-Radius für Check-In (Meter)	`100`
`location`	`object`	GPS-Koordinaten	`{ lat: 52.5200, lng: 13.4050 }`
`languages`	`string[]`	Verfügbare Sprachen	`["de", "en", "fr"]`
`defaultLanguage`	`string`	Standard-Sprache	`"de"`
`trustModeEnabled`	`boolean`	Trust-System aktiv?	`true`
`minTrust-LevelForEntry`	`number`	Min. Trust-Level (0-100)	`30`
`autoCheckoutAfter-Hours`	`number`	Auto-Checkout nach X Stunden	`6`

Security Rules:

```
// Alle Club-Mitglieder können lesen
allow read: if request.auth != null && isClubMember(clubId);

// Nur Admins können schreiben
allow write: if request.auth != null && hasRole(clubId, 'admin');
```

---

3. ROLLEN- & RECHTE-MODELL

3.1 Rollen-Definitionen

Rollen-Hierarchie

```
SUPER_ADMIN (Plattform-Ebene)
└── CLUB_ADMIN / OWNER (Club-Ebene)
    ├── DJ / LICHTJOCKEY (Steuerung)
    └── STAFF (Personal)
        ├── DOOR (Türsteher)
        ├── WAITER (Kellner)
        ├── BAR (Bar-Personal)
        └── CLOAKROOM (Garderoben-Personal)
    └── GUEST (Gast)
```

Rollen-Details

`SUPER_ADMIN`

- **Ebene:** Plattform
- **Speicherort:** `platform/users/{uid}.isPlatformAdmin = true`
- **Berechtigungen:**
- Alle Clubs lesen/schreiben
- Clubs erstellen/löschen
- Subscriptions verwalten
- Alle User sehen
- Plattform-Analytics

`CLUB_ADMIN` / `OWNER`

- **Ebene:** Club
- **Speicherort:** `clubs/{clubId}/users/{uid}.roles = ["admin"]` ODER `platform/clubs/{clubId}.ownerId = uid`
- **Berechtigungen:**
- Club-Einstellungen ändern
- Personal hinzufügen/entfernen
- Analytics einsehen
- Alle Club-Daten lesen/schreiben
- DJ-Console nutzen

`DJ` / `LICHTJOCKEY`

- **Ebene:** Club

- **Speicherort:** `clubs/{clubId}/users/{uid}.roles = ["dj"]`
- **Berechtigungen:**
- DJ-Console nutzen
- Lichtshow steuern
- Gewinnspiele starten
- Broadcasts senden
- Gästeliste lesen

`STAFF` (Basis-Rolle für alle Personal-Rollen)

- **Ebene:** Club
- **Speicherort:** `clubs/{clubId}/users/{uid}.roles = ["staff", "door"|"waiter"|"bar"|"cloakroom"]`
- **Berechtigungen (alle Staff):**
- Gästeliste lesen
- Eigene Check-Ins protokollieren

`DOOR` (Türsteher)

- **Zusatz-Rolle:** `"staff"` + `"door"`
- **Berechtigungen:**
- QR-Codes scannen
- User-Trust-Level einsehen
- Check-In/Out durchführen
- Blacklist prüfen
- Trust-Level erhöhen (nach Verifizierung)

`WAITER` (Kellner)

- **Zusatz-Rolle:** `"staff"` + `"waiter"`
- **Berechtigungen:**
- Bestellungen erstellen
- Bestellungen aktualisieren
- Bestellungen abschließen
- Tischplan sehen

`BAR` (Bar-Personal)

- **Zusatz-Rolle:** `"staff"` + `"bar"`
- **Berechtigungen:**
- Bestellungen lesen
- Bestellungen als “preparing” markieren
- Bestellungen als “served” markieren

`CLOAKROOM` (Garderoben-Personal)

- **Zusatz-Rolle:** `"staff"` + `"cloakroom"`
- **Berechtigungen:**
- Tickets erstellen
- Tickets scannen
- Items ausgeben
- Items als verloren markieren

`GUEST` (Gast)

- **Ebene:** Club
- **Speicherort:** `clubs/{clubId}/users/{uid}.roles = ["guest"]` (Default)
- **Berechtigungen:**
- Check-In/Out (selbst)
- Chat mit Freunden
- Freunde hinzufügen
- Gruppen erstellen
- Lichtshow/Messages/Countdowns sehen
- Gewinnspiele teilnehmen

3.2 Berechtigungsmatrix

Collection/Dokument	SUPER_ADMIN	CLUB_ADMIN	DJ	DOOR	WAITER	BAR	CLOAKROOM	GUEST
PLATTFORM-EBENE								
`platform/clubs`	R/W	R (own)	-	-	-	-	-	-
`platform/groups`	R/W	R (own)	-	-	-	-	-	-
`platform/users`	R/W	R (club users)	-	-	-	-	-	-
CLUB-EBENE								
`clubs/{clubId}/users`	R/W	R/W	R	R/W (verify)	R	R	R	R (own + friends)
`clubs/{clubId}/state/global`	R/W	R/W	R/W	R	R	R	R	R
`clubs/{clubId}/chats`	R	R	-	-	-	-	-	R (own)
`clubs/{clubId}/chats/{chatId}/messages`	R	R	-	-	-	-	-	R/W (own chats)

Collection/Dokument	SUPER_ADMIN	CLUB_ADMIN	DJ	DOOR	WAITER	BAR	CLOAKROOM	GUEST
`clubs/{clubId}/orders`	R/W	R/W	-	-	R/W	R/W	-	-
`clubs/{clubId}/cloakroom`	R/W	R/W	-	-	-	-	R/W	-
`clubs/{clubId}/config`	R/W	R/W	R	R	R	R	R	R

Legende:
- `R` = Read (Lesen)
- `W` = Write (Schreiben/Löschen)
- `-` = Kein Zugriff

Spezielle Regeln:
- **DOOR** kann User-Dokumente updaten (nur Felder: `trustedLevel`, `verifiedBy`, `verifiedAt`, `checkedIn`, `checkedInAt`)
- **GUEST** kann nur eigenes User-Dokument updaten (nur Felder: `displayName`, `photoURL`, `checkedIn`, `language`)
- **GUEST** kann nur Chats lesen, in denen er `participant` ist
- **GUEST** kann nur Nachrichten in eigenen Chats senden/löschen

---

3.3 Firestore Security Rules (Policy-Logik)

Basis-Funktionen (Helper Functions)

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {

    // ===== HELPER FUNCTIONS =====

    // Prüfe ob User eingeloggt ist
    function isAuthenticated() {
      return request.auth != null;
    }

    // Prüfe ob User Super-Admin ist
    function isSuperAdmin() {
      return isAuthenticated() &&
        get(/databases/$(database)/documents/platform/users/$(re-
quest.auth.uid)).data.isPlatformAdmin == true;
    }

    // Prüfe ob User Club-Owner ist
    function isClubOwner(clubId) {
      return isAuthenticated() &&
        get(/databases/$(database)/documents/platform/clubs/$(clubId)).data.ownerId
== request.auth.uid;
    }

    // Prüfe ob User eine bestimmte Rolle im Club hat
    function hasRole(clubId, role) {
      return isAuthenticated() &&
        role in get(/databases/$(database)/documents/clubs/$(clubId)/users/$(re-
quest.auth.uid)).data.roles;
    }

    // Prüfe ob User eine von mehreren Rollen hat
    function hasAnyRole(clubId, rolesList) {
      let userRoles = get(/databases/$(database)/documents/clubs/$(clubId)/users/$(re-
quest.auth.uid)).data.roles;
      return isAuthenticated() &&
        userRoles.hasAny(rolesList);
    }

    // Prüfe ob User Mitglied des Clubs ist
    function isClubMember(clubId) {
      return isAuthenticated() &&
        exists(/databases/$(database)/documents/clubs/$(clubId)/users/$(re-
quest.auth.uid));
    }

    // Prüfe ob User Teilnehmer eines Chats ist
    function isChatParticipant(clubId, chatId) {
      return isAuthenticated() &&
        request.auth.uid in get(/databases/$(database)/documents/clubs/$(clubId)/
chats/$(chatId)).data.participants;
    }

    // ===== PLATTFORM-EBENE =====

    match /platform/clubs/{clubId} {
      // Super-Admins: Alles
      // Club-Owners: Nur eigenen Club lesen/updaten
      allow read: if isSuperAdmin() || isClubOwner(clubId);
      allow create, delete: if isSuperAdmin();
      allow update: if isSuperAdmin() ||
```

```
        (isClubOwner(clubId) &&
         !request.resource.data.diff(resource.data).affectedKeys().hasAny(['subscrip-
tionTier', 'subscriptionStatus']));
    }

    match /platform/groups/{groupId} {
      allow read: if isSuperAdmin() ||
        get(/databases/$(database)/documents/platform/groups/$(groupId)).data.ownerId
== request.auth.uid;
      allow write: if isSuperAdmin();
    }

    match /platform/users/{uid} {
      // Jeder kann sein eigenes Dokument lesen/updaten
      // Super-Admins können alle lesen
      allow read: if isSuperAdmin() || request.auth.uid == uid;
      allow update: if request.auth.uid == uid &&
        !request.resource.data.diff(resource.data).affectedKeys().hasAny(['isPlatform-
Admin', 'ownedClubs']);
      allow create: if request.auth.uid == uid;
    }

    // ===== CLUB-EBENE =====

    match /clubs/{clubId}/users/{uid} {
      // Lesen:
      // - Super-Admins: Alle
      // - Club-Admins: Alle
      // - Staff: Alle (für Check-In/Verifizierung)
      // - Guests: Nur eigenes Dokument + Freunde
      allow read: if isSuperAdmin() ||
        hasRole(clubId, 'admin') ||
        hasAnyRole(clubId, ['staff', 'door', 'waiter', 'bar', 'cloakroom', 'dj']) ||
        (request.auth.uid == uid) ||
        (request.auth.uid in resource.data.friendIds);

      // Erstellen: Nur beim ersten Login (Auto-Create)
      allow create: if request.auth.uid == uid;

      // Updaten:
      // - Admins: Alle Felder
      // - Door-Staff: Nur Trust-Felder
      // - Guests: Nur eigene Basis-Felder
      allow update: if hasRole(clubId, 'admin') ||
        (hasRole(clubId, 'door') &&
         request.resource.data.diff(resource.data).affectedKeys().hasOnly(['trusted-
Level', 'verifiedBy', 'verifiedAt', 'checkedIn', 'checkedInAt', 'blacklisted', 'black-
listReason'])) ||
        (request.auth.uid == uid &&
         request.resource.data.diff(resource.data).affectedKeys().hasOnly(['display-
Name', 'photoURL', 'checkedIn', 'checkedInAt', 'language', 'lastSeen']));
    }

    match /clubs/{clubId}/users/{uid}/friends/{friendId} {
      // Nur User selbst kann seine Freunde verwalten
      allow read, write: if request.auth.uid == uid;
    }

    match /clubs/{clubId}/users/{uid}/requests/{requesterId} {
      // User kann seine Anfragen lesen/löschen (akzeptieren)
      allow read, write: if request.auth.uid == uid;
      // Requester kann seine eigene Anfrage erstellen
      allow create: if request.auth.uid == requesterId;
```

```
    }

    match /clubs/{clubId}/state/global {
      // Alle Club-Mitglieder können lesen
      allow read: if isClubMember(clubId);
      // Nur DJ/Admin kann schreiben
      allow write: if hasAnyRole(clubId, ['admin', 'dj']);
    }

    match /clubs/{clubId}/chats/{chatId} {
      // Nur Teilnehmer können lesen
      allow read: if isChatParticipant(clubId, chatId);
      // Teilnehmer können Chat erstellen
      allow create: if request.auth.uid in request.resource.data.participants;
      // Creator kann Gruppe updaten/löschen
      allow update, delete: if resource.data.createdBy == request.auth.uid || re-
source.data.type == 'private';
    }

    match /clubs/{clubId}/chats/{chatId}/messages/{messageId} {
      // Nur Teilnehmer können lesen
      allow read: if isChatParticipant(clubId, chatId);
      // Teilnehmer können Nachrichten senden
      allow create: if isChatParticipant(clubId, chatId) &&
        request.resource.data.sender == request.auth.uid;
      // Sender kann eigene Nachricht löschen/updaten
      allow update, delete: if resource.data.sender == request.auth.uid;
    }

    match /clubs/{clubId}/orders/{orderId} {
      // Nur Staff kann Orders verwalten
      allow read, write: if hasAnyRole(clubId, ['admin', 'staff', 'waiter', 'bar']);
    }

    match /clubs/{clubId}/cloakroom/{ticketId} {
      // Nur Cloakroom-Staff kann verwalten
      allow read, write: if hasAnyRole(clubId, ['admin', 'staff', 'cloakroom']);
    }

    match /clubs/{clubId}/config/settings {
      // Alle Club-Mitglieder können lesen
      allow read: if isClubMember(clubId);
      // Nur Admins können schreiben
      allow write: if hasRole(clubId, 'admin');
    }
  }
}
```

Beispiel-Szenarien für Security Rules

Szenario 1: Gast checkt sich selbst ein

```
// User: guest_user123
// Action: updateDoc(clubs/club_abc/users/guest_user123, { checkedIn: true })

// Rule-Evaluation:
// 1. isAuthenticated() ✅
// 2. request.auth.uid == uid ✅ (guest_user123 == guest_user123)
// 3. affectedKeys = ['checkedIn'] ✅ (erlaubt)
// → ALLOWED
```

Szenario 2: Türsteher erhöht Trust-Level

```
// User: staff_door1 (Rolle: ["staff", "door"])
// Action: updateDoc(clubs/club_abc/users/guest_user123, { trustedLevel: 50,
verifiedBy: 'staff_door1' })

// Rule-Evaluation:
// 1. hasRole(clubId, 'door') ✅
// 2. affectedKeys = ['trustedLevel', 'verifiedBy'] ✅ (alle erlaubt für Door)
// → ALLOWED
```

Szenario 3: Gast versucht, sich selbst Trust-Level zu geben

```
// User: guest_user123
// Action: updateDoc(clubs/club_abc/users/guest_user123, { trustedLevel: 100 })

// Rule-Evaluation:
// 1. isAuthenticated() ✅
// 2. request.auth.uid == uid ✅
// 3. affectedKeys = ['trustedLevel'] ❌ (NICHT in erlaubten Feldern für Guests)
// → DENIED
```

Szenario 4: DJ startet Lichtshow

```
// User: dj_user1 (Rolle: ["dj"])
// Action: updateDoc(clubs/club_abc/state/global, { mode: 'lightshow', lightColor:
'#ff0000' })

// Rule-Evaluation:
// 1. hasAnyRole(clubId, ['admin', 'dj']) ✅
// → ALLOWED
```

Szenario 5: Gast versucht, fremden Chat zu lesen

```
// User: guest_user123
// Action: getDocs(clubs/club_abc/chats/user456_user789/messages)

// Rule-Evaluation:
// 1. isChatParticipant(clubId, chatId) ❌ (guest_user123 ist NICHT in participants)
// → DENIED
```

---

4. MIGRATIONSPLAN

4.1 Bestehende Collections (NICHT löschen!)

Das Demo-System verwendet folgende Firestore-Struktur:

```
DEMO (Alt-System):
├── users/{uid}                              ← User-Stammdaten
│   ├── friends/{friendId}                  ← Freundschaften
│   └── requests/{requesterId}              ← Freundschaftsanfragen
├── chats/{chatId}                           ← Chat-Metadaten
│   └── messages/{messageId}                ← Nachrichten
└── public/globalState                       ← Globaler Club-Status
```

WICHTIG: Diese Collections dürfen **NICHT gelöscht** werden, da das Demo-System produktiv läuft. Die Migration erfolgt **parallel** zum Betrieb.

4.2 Migrationsstrategie

Phase 1: Neue Struktur aufbauen (PARALLEL)

Ziel: Neue Multi-Club-Struktur aufbauen, ohne Alt-System zu stören

Schritte:

1. **Club-Dokument erstellen:**

```
javascript
  // Einmaliges Setup für den ersten Club (Demo-Club)
  await setDoc(doc(db, 'platform/clubs/demo_club'), {
    clubId: 'demo_club',
    name: 'Demo Club',
    slug: 'demo',
    ownerId: 'admin_uid_hier',
    subscriptionTier: 'pro',
    subscriptionStatus: 'active',
    features: ['chat', 'lightshow', 'lottery'],
    theme: { primaryColor: '#00ffff', secondaryColor: '#ff00ff' },
    trustedMode: false,  // Erst später aktivieren
    language: 'de',
    createdAt: Date.now(),
    updatedAt: Date.now()
  })
```

1. **Config-Dokument erstellen:**

```
javascript
  await setDoc(doc(db, 'clubs/demo_club/config/settings'), {
    features: { chat: true, lightshow: true, orders: false },
    theme: { primaryColor: '#00ffff' },
    languages: ['de', 'en'],
    defaultLanguage: 'de',
    trustModeEnabled: false  // Später aktivieren
  })
```

2. **State-Dokument erstellen:**

```
javascript
  await setDoc(doc(db, 'clubs/demo_club/state/global'), {
    mode: 'normal',
```

```
    lightColor: null,
    lightEffect: null,
    // ... alle Felder aus public/globalState kopieren
  })
```

Status nach Phase 1:

```
ALT (läuft weiter):
├── users/{uid}
├── chats/{chatId}
└── public/globalState

NEU (parallel):
├── platform/clubs/demo_club
├── clubs/demo_club/config/settings
└── clubs/demo_club/state/global
```

Phase 2: User-Daten migrieren (BATCH)

Ziel: Alle User von `users/` nach `clubs/demo_club/users/` kopieren

Migrations-Script:

```
import { getDocs, collection, setDoc, doc } from 'firebase/firestore'

async function migrateUsers() {
  const oldUsersSnap = await getDocs(collection(db, 'users'))

  for (const userDoc of oldUsersSnap.docs) {
    const oldData = userDoc.data()
    const uid = userDoc.id

    // Neues User-Dokument erstellen
    await setDoc(doc(db, 'clubs/demo_club/users', uid), {
      // Basis-Daten (1:1 übernehmen)
      uid: oldData.uid,
      email: oldData.email,
      displayName: oldData.displayName || null,
      photoURL: oldData.photoURL || null,

      // Rollen (Default: guest, außer admin)
      roles: oldData.role === 'admin' ? ['admin'] : ['guest'],

      // Check-In (1:1 übernehmen)
      checkedIn: oldData.checkedIn || false,
      checkedInAt: oldData.checkedInAt || null,
      lastSeen: oldData.lastSeen || Date.now(),

      // Friend-Code (1:1 übernehmen)
      friendCode: oldData.friendCode || generateFriendCode(),

      // Sprache (Default: de)
      language: 'de',

      // Trust-System (NEU - erstmal leer)
      phoneNumber: null,
      phoneVerified: false,
      deviceIdHash: null,
      faceHash: null,
      trustedLevel: 0,
      verifiedBy: null,
      verifiedAt: null,
      blacklisted: false,
      blacklistReason: null,

      // Statistiken (NEU)
      visitCount: 0,
      lastVisits: [],

      // Freunde (denormalisiert für Queries)
      friendIds: []  // Wird in Phase 3 gefüllt
    }, { merge: true })

    console.log(`✅ Migriert: ${uid}`)
  }
}
```

Status nach Phase 2:

```
ALT (läuft weiter):
├── users/{uid} ← NICHT löschen!
├── chats/{chatId}
└── public/globalState

NEU:
├── platform/clubs/demo_club
├── clubs/demo_club/users/{uid} ← NEU kopiert
├── clubs/demo_club/config/settings
└── clubs/demo_club/state/global
```

Phase 3: Freundschaften migrieren (BATCH)

Migrations-Script:

```
async function migrateFriends() {
  const usersSnap = await getDocs(collection(db, 'users'))

  for (const userDoc of usersSnap.docs) {
    const uid = userDoc.id

    // Alte Freunde lesen
    const oldFriendsSnap = await getDocs(collection(db, `users/${uid}/friends`))
    const friendIds: string[] = []

    for (const friendDoc of oldFriendsSnap.docs) {
      const friendData = friendDoc.data()
      const friendId = friendDoc.id

      // Neues Friend-Dokument erstellen
      await setDoc(doc(db, `clubs/demo_club/users/${uid}/friends`, friendId), {
        uid: friendData.uid,
        email: friendData.email
      })

      friendIds.push(friendId)
    }

    // friendIds im User-Dokument aktualisieren (für Queries)
    await setDoc(doc(db, `clubs/demo_club/users`, uid), {
      friendIds: friendIds
    }, { merge: true })

    console.log(`✅ ${uid}: ${friendIds.length} Freunde migriert`)
  }
}
```

Phase 4: Freundschaftsanfragen migrieren (BATCH)

Migrations-Script:

```
async function migrateRequests() {
  const usersSnap = await getDocs(collection(db, 'users'))

  for (const userDoc of usersSnap.docs) {
    const uid = userDoc.id

    // Alte Anfragen lesen
    const oldRequestsSnap = await getDocs(collection(db, `users/${uid}/requests`))

    for (const requestDoc of oldRequestsSnap.docs) {
      const requestData = requestDoc.data()
      const requesterId = requestDoc.id

      // Neue Request erstellen
      await setDoc(doc(db, `clubs/demo_club/users/${uid}/requests`, requesterId), {
        uid: requestData.uid,
        email: requestData.email,
        friendCode: requestData.friendCode,
        message: requestData.message || 'Hi! 👋',
        timestamp: requestData.timestamp || Date.now()
      })
    }

    console.log(`✅ ${uid}: Anfragen migriert`)
  }
}
```

Phase 5: Chats migrieren (BATCH)

Migrations-Script:

```
async function migrateChats() {
  const oldChatsSnap = await getDocs(collection(db, 'chats'))

  for (const chatDoc of oldChatsSnap.docs) {
    const oldChatData = chatDoc.data()
    const oldChatId = chatDoc.id

    // Neues Chat-Dokument erstellen
    await setDoc(doc(db, `clubs/demo_club/chats`, oldChatId), {
      chatId: oldChatId,
      type: oldChatData.type || 'group',
      name: oldChatData.name || null,
      participants: oldChatData.participants || [],
      createdBy: oldChatData.createdBy || null,
      createdAt: oldChatData.createdAt || Date.now(),
      lastMessageAt: Date.now(),
      lastMessagePreview: null
    })

    // Nachrichten migrieren
    const oldMessagesSnap = await getDocs(collection(db, `chats/${oldChatId}/mes-
sages`))

    for (const msgDoc of oldMessagesSnap.docs) {
      const msgData = msgDoc.data()

      await setDoc(doc(db, `clubs/demo_club/chats/${oldChatId}/messages`, msgDoc.id),
{
        text: msgData.text || '',
        sender: msgData.sender,
        senderName: msgData.senderName || 'Unknown',
        image: msgData.image || null,
        ephemeral: msgData.ephemeral || null,
        viewedBy: msgData.viewedBy || [msgData.sender],
        createdAt: msgData.createdAt || Date.now(),
        expiresAt: msgData.expiresAt || null,
        deleted: msgData.deleted || false
      })
    }

    console.log(`✅ Chat ${oldChatId}: ${oldMessagesSnap.size} Nachrichten migriert`)
  }
}
```

Phase 6: Code auf neue Struktur umstellen

Änderungen in der App:

Vorher (Demo):

```
// users/{uid} lesen
onSnapshot(doc(db, 'users', uid), (snap) => {
  setUserData(snap.data())
})

// globalState lesen
onSnapshot(doc(db, 'public', 'globalState'), (snap) => {
  setAppState(snap.data())
})
```

Nachher (Multi-Club):

```
// clubs/{clubId}/users/{uid} lesen
const clubId = 'demo_club'  // Später dynamisch
onSnapshot(doc(db, `clubs/${clubId}/users`, uid), (snap) => {
  setUserData(snap.data())
})

// clubs/{clubId}/state/global lesen
onSnapshot(doc(db, `clubs/${clubId}/state/global`), (snap) => {
  setAppState(snap.data())
})
```

Zentrale Änderungen:

1. **Firebase Init Hook erstellen:**

`typescript`
`  // packages/core/src/hooks/use-club.ts`
`  export const useClub = () => {`
`    const [clubId, setClubId] = useState<string>('demo_club')`
`    // Später: aus URL/Context lesen`
`    return { clubId }`
`  }`

1. **Alle Firestore-Calls anpassen:**
   - `users/` → `clubs/{clubId}/users/`
   - `chats/` → `clubs/{clubId}/chats/`
   - `public/globalState` → `clubs/{clubId}/state/global`

Phase 7: Alt-System abschalten (OPTIONAL)

Nach erfolgreicher Migration und Testing:

1. **Neue App ausrollen:**
   - Neue Version deployen (mit `clubs/` Struktur)
   - Alt-App offline nehmen

2. **Alte Collections archivieren:**

   `typescript`
   `  // NICHT löschen, sondern umbenennen für Backup`
   `  // Manuell in Firebase Console:`
   `  // users → users_backup_2025_12_01`

```
   // chats → chats_backup_2025_12_01
   // public → public_backup_2025_12_01
```

3. **Security Rules updaten:**
   - Alte Collections read-only setzen
   - Neue Collections aktivieren

4.3 Code-Migration

Beispiel 1: Lichtshow (DJ-Console)

Vorher (demo - admin.php):

```
// public/globalState updaten
const updateGlobal = (data) =>
  setDoc(doc(db, 'public', 'globalState'), data, { merge: true })

// Farbe senden
const sendColor = (color) => {
  updateGlobal({
    mode: 'lightshow',
    lightConfig: { type: 'color', color: color }
  })
}
```

Nachher (Multi-Club - dj-console):

```
// packages/core/src/hooks/use-club-state.ts
export const useClubState = (clubId: string) => {
  const updateClubState = async (data: Partial<ClubState>) => {
    await setDoc(
      doc(db, `clubs/${clubId}/state`, 'global'),
      data,
      { merge: true }
    )
  }

  return { updateClubState }
}

// apps/dj-console/src/components/light-control.tsx
import { useClub } from '@nightlife/core'
import { useClubState } from '@nightlife/core'

export const LightControl = () => {
  const { clubId } = useClub()  // 'demo_club'
  const { updateClubState } = useClubState(clubId)

  const sendColor = (color: string) => {
    updateClubState({
      mode: 'lightshow',
      lightColor: color,
      lightEffect: 'color'
    })
  }

  return (
    <button onClick={() => sendColor('#ff0000')}>
      Rot
    </button>
  )
}
```

Änderungen:

- ✅ `public/globalState` → `clubs/{clubId}/state/global`
- ✅ `lightConfig.type` → `lightEffect` (flache Struktur)
- ✅ `lightConfig.color` → `lightColor` (flache Struktur)
- ✅ TypeScript-Typen hinzugefügt
- ✅ Hook-basierte Architektur

Beispiel 2: GlobalOverlay (Club-App)

Vorher (demo - index.php):

```
// Global State Listener
useEffect(() => {
  const unsub = onSnapshot(doc(db, 'public', 'globalState'), (snap) => {
    if(snap.exists()) setAppState(snap.data())
  })
  return () => unsub()
}, [])

// Rendering
if (appState.mode === 'lightshow') {
  const cfg = appState.lightConfig
  if (cfg.type === 'color') {
    return <div style={{ backgroundColor: cfg.color }} />
  }
}
```

Nachher (Multi-Club - club-app):

```tsx
// apps/club-app/src/components/global-overlay.tsx
import { useClub } from '@nightlife/core'
import { ClubState } from '@nightlife/shared-types'

export const GlobalOverlay = () => {
  const { clubId } = useClub()
  const [clubState, setClubState] = useState<ClubState | null>(null)

  useEffect(() => {
    const unsub = onSnapshot(
      doc(db, `clubs/${clubId}/state`, 'global'),
      (snap) => {
        if (snap.exists()) {
          setClubState(snap.data() as ClubState)
        }
      }
    )
    return () => unsub()
  }, [clubId])

  if (!clubState) return null

  // Lichtshow
  if (clubState.mode === 'lightshow') {
    if (clubState.lightEffect === 'color') {
      return (
        <div
          className="fixed inset-0 z-50 transition-colors duration-300"
          style={{ backgroundColor: clubState.lightColor }}
        />
      )
    }

    if (clubState.lightEffect === 'strobe') {
      return <div className="fixed inset-0 z-50 bg-white animate-pulse" />
    }

    // ... weitere Effekte
  }

  // Nachrichten
  if (clubState.mode === 'message') {
    return (
      <div className="fixed inset-0 z-50 bg-black flex items-center justify-center">
        <h1 className="text-yellow-400 text-5xl font-black animate-pulse">
          {clubState.messageText}
        </h1>
      </div>
    )
  }

  return null
}
```

Änderungen:

- ✅ `public/globalState` → `clubs/{clubId}/state/global`
- ✅ `lightConfig` → flache Felder (`lightEffect`, `lightColor`)
- ✅ TypeScript-Typen
- ✅ Sauberere Component-Struktur

Beispiel 3: Chat-System

Vorher (demo - index.php):

```
// Chats laden
useEffect(() => {
  const unsub = onSnapshot(
    query(
      collection(db, 'chats'),
      where('participants', 'array-contains', user.uid)
    ),
    s => setChats(s.docs.map(d => ({id: d.id, ...d.data()})))
  )
  return () => unsub()
}, [user])
```

Nachher (Multi-Club - club-app):

```
// apps/club-app/src/components/chat-system.tsx
import { useClub } from '@nightlife/core'

export const ChatSystem = () => {
  const { clubId } = useClub()
  const [chats, setChats] = useState<Chat[]>([])

  useEffect(() => {
    const unsub = onSnapshot(
      query(
        collection(db, `clubs/${clubId}/chats`),
        where('participants', 'array-contains', user.uid)
      ),
      s => setChats(s.docs.map(d => ({ id: d.id, ...d.data() } as Chat)))
    )
    return () => unsub()
  }, [clubId, user])

  return (
    // ... UI
  )
}
```

Änderungen:
- ✅ `chats/` → `clubs/{clubId}/chats/`
- ✅ TypeScript-Typen

Beispiel 4: Friend-System

Vorher (demo - index.php):

```
// Freund hinzufügen
const sendReq = async () => {
  await setDoc(
    doc(db, 'users', targetUser.id, 'requests', user.uid),
    {
      email: user.email,
      uid: user.uid,
      friendCode: userData.friendCode,
      message: msg,
      timestamp: Date.now()
    }
  )
}

// Anfrage akzeptieren
const handleAcceptRequest = async (req) => {
  // Beidseitige Freundschaft
  await setDoc(doc(db, 'users', user.uid, 'friends', req.uid), {
    email: req.email,
    uid: req.uid
  })
  await setDoc(doc(db, 'users', req.uid, 'friends', user.uid), {
    email: user.email,
    uid: user.uid
  })

  // Anfrage löschen
  await deleteDoc(doc(db, 'users', user.uid, 'requests', req.id))
}
```

Nachher (Multi-Club - club-app):

```
// packages/core/src/hooks/use-friends.ts
export const useFriends = (clubId: string, userId: string) => {
  const sendFriendRequest = async (targetUserId: string, message: string) => {
    await setDoc(
      doc(db, `clubs/${clubId}/users/${targetUserId}/requests`, userId),
      {
        uid: userId,
        email: user.email,
        friendCode: userData.friendCode,
        message,
        timestamp: Date.now()
      }
    )
  }

  const acceptFriendRequest = async (requesterUserId: string) => {
    const batch = writeBatch(db)

    // Beidseitige Freundschaft
    batch.set(
      doc(db, `clubs/${clubId}/users/${userId}/friends`, requesterUserId),
      { uid: requesterUserId, email: requester.email }
    )
    batch.set(
      doc(db, `clubs/${clubId}/users/${requesterUserId}/friends`, userId),
      { uid: userId, email: user.email }
    )

    // Anfrage löschen
    batch.delete(
      doc(db, `clubs/${clubId}/users/${userId}/requests`, requesterUserId)
    )

    // friendIds aktualisieren (denormalisiert)
    batch.update(
      doc(db, `clubs/${clubId}/users`, userId),
      { friendIds: arrayUnion(requesterUserId) }
    )
    batch.update(
      doc(db, `clubs/${clubId}/users`, requesterUserId),
      { friendIds: arrayUnion(userId) }
    )

    await batch.commit()
  }

  return { sendFriendRequest, acceptFriendRequest }
}
```

Änderungen:
- ✅ `users/` → `clubs/{clubId}/users/`
- ✅ Batch-Writes für atomare Operationen
- ✅ `friendIds` Array im User-Dokument (für Queries)
- ✅ Hook-basierte Logik

---

5. MEHRSPRACHIGKEIT (i18n)

5.1 Struktur der JSON-Files

Dateistruktur:

```
packages/ui/src/locales/
├── de.json     # Deutsch (Standard)
├── en.json     # Englisch
├── fr.json     # Französisch
├── es.json     # Spanisch
└── it.json     # Italienisch
```

Beispiel: `de.json`

```
{
  "common": {
    "welcome": "Willkommen",
    "cancel": "Abbrechen",
    "confirm": "Bestätigen",
    "save": "Speichern",
    "delete": "Löschen",
    "edit": "Bearbeiten",
    "loading": "Laden..."
  },

  "auth": {
    "login": {
      "title": "Club App",
      "email": "E-Mail (oder 'admin')",
      "password": "Passwort",
      "submit": "Anmelden",
      "forgotPassword": "Passwort vergessen?",
      "noAccount": "Noch kein Account?",
      "register": "Registrieren"
    },
    "register": {
      "title": "Registrieren",
      "hasAccount": "Bereits registriert?",
      "login": "Anmelden"
    },
    "errors": {
      "invalidEmail": "Ungültige E-Mail-Adresse",
      "wrongPassword": "Falsches Passwort",
      "userNotFound": "User nicht gefunden",
      "emailInUse": "E-Mail bereits verwendet"
    }
  },

  "home": {
    "status": {
      "title": "Dein Status",
      "inClub": "IM CLUB 🎵",
      "outside": "DRAUSSEN"
    },
    "checkIn": {
      "buttonIn": "JETZT EINCHECKEN",
      "buttonOut": "AUSCHECKEN"
    },
    "friendRequest": {
      "title": "Neue Anfrage! 👋",
      "accept": "ANNEHMEN"
    },
    "qrCode": {
      "hideHint": "Tippen zum Zeigen",
      "yourCode": "Dein Code"
    }
  },

  "chat": {
    "tabs": {
      "crews": "Crews",
      "friends": "Freunde"
    },
    "actions": {
      "addFriend": "FREUND HINZUFÜGEN",
      "createGroup": "GRUPPE",
```

```
      "send": "Senden",
      "delete": "Löschen"
    },
    "addFriend": {
      "title": "Freund hinzufügen",
      "scanButton": "QR-CODE SCANNEN",
      "codeInput": "Code eingeben",
      "submit": "Suchen",
      "notFound": "User nicht gefunden",
      "foundTitle": "Anfrage an...",
      "messageLabel": "Nachricht wählen:",
      "messages": {
        "hi": "Hi! 👋",
        "cheers": "Lass anstoßen! 🥂",
        "outfit": "Cooles Outfit! 🔥"
      }
    },
    "group": {
      "createTitle": "Neue Crew",
      "nameInput": "Crew-Name",
      "selectLabel": "Wer soll rein?",
      "createButton": "ERSTELLEN",
      "manageTitle": "Crew verwalten",
      "membersLabel": "Mitglieder:",
      "youLabel": "Du",
      "leaveButton": "VERLASSEN",
      "deleteButton": "LÖSCHEN"
    },
    "room": {
      "imageTimer": "Ansehen ({{seconds}}s)",
      "imageDeleted": "♻️ Bild gelöscht.",
      "uploadPrompt": "Timer? (5 für 5s, Leer lassen für immer)"
    }
  },

  "djConsole": {
    "login": {
      "title": "DJ Console",
      "passwordPlaceholder": "Zugangscode",
      "submitButton": "UNLOCK"
    },
    "dashboard": {
      "title": "DJ CONSOLE",
      "guestCount": "Gäste",
      "logout": "Logout"
    },
    "lights": {
      "title": "Lichtsteuerung",
      "audioSync": "MIC SYNC",
      "audioSyncActive": "LIVE SYNC ON",
      "colors": {
        "red": "Rot",
        "green": "Grün",
        "blue": "Blau",
        "yellow": "Gelb",
        "magenta": "Magenta",
        "cyan": "Cyan",
        "white": "Weiß",
        "off": "Aus"
      },
      "effects": {
        "psychedelic": "🌈 PSYCHEDELIC",
        "strobe": "⚡ STROBO"
```

```
      }
    },
    "lottery": {
      "title": "Gewinnspiel",
      "winnersLabel": "Anzahl Gewinner",
      "prizeLabel": "Gewinn-Code",
      "startButton": "VERLOSUNG STARTEN"
    },
    "broadcast": {
      "title": "Nachricht",
      "messageInput": "Nachricht eingeben",
      "targetLabels": {
        "in": "IM CLUB",
        "out": "DRAUSSEN",
        "all": "ALLE"
      },
      "sendButton": "SEND"
    },
    "guestList": {
      "title": "Gästeliste (Im Club)",
      "empty": "Niemand eingecheckt."
    },
    "stopButton": "🛑 STOP / RESET"
  },

  "admin": {
    "dashboard": {
      "title": "Club Dashboard",
      "overview": "Übersicht",
      "settings": "Einstellungen",
      "staff": "Personal",
      "analytics": "Statistiken",
      "subscription": "Abo"
    },
    "settings": {
      "general": "Allgemein",
      "clubName": "Club-Name",
      "address": "Adresse",
      "openingHours": "Öffnungszeiten",
      "capacity": "Kapazität",
      "features": "Features",
      "theme": "Design",
      "primaryColor": "Primärfarbe",
      "secondaryColor": "Sekundärfarbe",
      "logo": "Logo",
      "trustMode": "Trust-System aktivieren"
    },
    "staff": {
      "title": "Personal-Verwaltung",
      "addButton": "PERSONAL HINZUFÜGEN",
      "roles": {
        "admin": "Admin",
        "dj": "DJ",
        "door": "Türsteher",
        "waiter": "Kellner",
        "bar": "Bar",
        "cloakroom": "Garderobe"
      },
      "removeButton": "Entfernen"
    }
  },

  "staff": {
```

```
    "door": {
      "title": "Türsteher-App",
      "scanButton": "QR-CODE SCANNEN",
      "userInfo": "User-Info",
      "trustLevel": "Trust-Level",
      "visitCount": "Besuche",
      "checkInButton": "EINCHECKEN",
      "verifyButton": "VERIFIZIEREN",
      "blacklistButton": "BLACKLIST"
    },
    "waiter": {
      "title": "Kellner-App",
      "orders": "Bestellungen",
      "newOrder": "NEUE BESTELLUNG",
      "table": "Tisch",
      "status": {
        "open": "Offen",
        "preparing": "In Vorbereitung",
        "served": "Serviert",
        "paid": "Bezahlt"
      }
    },
    "cloakroom": {
      "title": "Garderoben-App",
      "checkIn": "EINLAGERN",
      "checkOut": "AUSGEBEN",
      "ticketNumber": "Ticket-Nummer",
      "itemDescription": "Beschreibung"
    }
  },

  "navigation": {
    "home": "Start",
    "chat": "Crew",
    "profile": "Profil",
    "settings": "Einstellungen"
  },

  "errors": {
    "generic": "Ein Fehler ist aufgetreten",
    "network": "Netzwerkfehler",
    "permission": "Keine Berechtigung",
    "notFound": "Nicht gefunden"
  }
}
```

Beispiel: `en.json`

```
{
  "common": {
    "welcome": "Welcome",
    "cancel": "Cancel",
    "confirm": "Confirm",
    "save": "Save",
    "delete": "Delete",
    "edit": "Edit",
    "loading": "Loading..."
  },

  "auth": {
    "login": {
      "title": "Club App",
      "email": "Email (or 'admin')",
      "password": "Password",
      "submit": "Sign In",
      "forgotPassword": "Forgot password?",
      "noAccount": "No account yet?",
      "register": "Register"
    },
    "register": {
      "title": "Register",
      "hasAccount": "Already registered?",
      "login": "Sign In"
    },
    "errors": {
      "invalidEmail": "Invalid email address",
      "wrongPassword": "Wrong password",
      "userNotFound": "User not found",
      "emailInUse": "Email already in use"
    }
  },

  "home": {
    "status": {
      "title": "Your Status",
      "inClub": "IN CLUB 🎵",
      "outside": "OUTSIDE"
    },
    "checkIn": {
      "buttonIn": "CHECK IN NOW",
      "buttonOut": "CHECK OUT"
    },
    "friendRequest": {
      "title": "New Request! 👋",
      "accept": "ACCEPT"
    },
    "qrCode": {
      "hideHint": "Tap to reveal",
      "yourCode": "Your Code"
    }
  },

  "chat": {
    "tabs": {
      "crews": "Crews",
      "friends": "Friends"
    },
    "actions": {
      "addFriend": "ADD FRIEND",
      "createGroup": "GROUP",
```

```
      "send": "Send",
      "delete": "Delete"
    },
    "addFriend": {
      "title": "Add Friend",
      "scanButton": "SCAN QR CODE",
      "codeInput": "Enter code",
      "submit": "Search",
      "notFound": "User not found",
      "foundTitle": "Request to...",
      "messageLabel": "Choose message:",
      "messages": {
        "hi": "Hi! 👋",
        "cheers": "Let's cheers! 🥂",
        "outfit": "Cool outfit! 🔥"
      }
    },
    "group": {
      "createTitle": "New Crew",
      "nameInput": "Crew Name",
      "selectLabel": "Who should join?",
      "createButton": "CREATE",
      "manageTitle": "Manage Crew",
      "membersLabel": "Members:",
      "youLabel": "You",
      "leaveButton": "LEAVE",
      "deleteButton": "DELETE"
    },
    "room": {
      "imageTimer": "View ({{seconds}}s)",
      "imageDeleted": "♻️ Image deleted.",
      "uploadPrompt": "Timer? (5 for 5s, leave empty for permanent)"
    }
  },

  // ... (rest analog zu de.json)
}
```

5.2 Zugriff in Components

i18n Hook erstellen

```tsx
// packages/core/src/hooks/use-i18n.ts
import { useState, useEffect } from 'react'
import de from '@nightlife/ui/locales/de.json'
import en from '@nightlife/ui/locales/en.json'
import fr from '@nightlife/ui/locales/fr.json'
import es from '@nightlife/ui/locales/es.json'
import it from '@nightlife/ui/locales/it.json'

const translations = { de, en, fr, es, it }

type Language = 'de' | 'en' | 'fr' | 'es' | 'it'

export const useI18n = (userLanguage?: string) => {
  const [lang, setLang] = useState<Language>('de')

  useEffect(() => {
    // Priorität: User-Profil → Browser → Default
    const detectedLang =
      userLanguage ||
      navigator.language.split('-')[0] ||
      'de'

    if (detectedLang in translations) {
      setLang(detectedLang as Language)
    } else {
      setLang('de')  // Fallback
    }
  }, [userLanguage])

  const t = (key: string, params?: Record<string, string | number>) => {
    const keys = key.split('.')
    let value: any = translations[lang]

    // Nested key navigation
    for (const k of keys) {
      if (value && typeof value === 'object' && k in value) {
        value = value[k]
      } else {
        console.warn(`Translation key not found: ${key}`)
        return key  // Fallback: Return key itself
      }
    }

    // Parameter-Ersetzung (z.B. {{name}})
    if (params && typeof value === 'string') {
      return value.replace(/\{\{(\w+)\}\}/g, (match, paramKey) => {
        return params[paramKey]?.toString() || match
      })
    }

    return value
  }

  return { t, lang, setLang }
}
```

Verwendung in Components

```
// apps/club-app/src/components/login-screen.tsx
import { useI18n } from '@nightlife/core'

export const LoginScreen = () => {
  const { t } = useI18n()
  const [email, setEmail] = useState('')
  const [password, setPassword] = useState('')

  return (
    <div className="flex flex-col items-center justify-center min-h-screen p-6">
      <h1 className="text-4xl font-black mb-8 bg-gradient-to-r from-fuchsia-500 to-
cyan-500 bg-clip-text text-transparent">
        {t('auth.login.title')}
      </h1>

      <form className="w-full max-w-sm space-y-4">
        <input
          type="email"
          value={email}
          onChange={(e) => setEmail(e.target.value)}
          placeholder={t('auth.login.email')}
          className="w-full px-4 py-3 rounded-lg bg-slate-800 text-white"
        />

        <input
          type="password"
          value={password}
          onChange={(e) => setPassword(e.target.value)}
          placeholder={t('auth.login.password')}
          className="w-full px-4 py-3 rounded-lg bg-slate-800 text-white"
        />

        <button
          type="submit"

className="w-full py-3 rounded-lg bg-gradient-to-r from-cyan-600 to-blue-600 text-
white font-bold"
        >
          {t('auth.login.submit')}
        </button>

        <div className="text-center text-sm text-slate-400">
          {t('auth.login.noAccount')}{' '}
          <button className="text-cyan-400 font-bold">
            {t('auth.login.register')}
          </button>
        </div>
      </form>
    </div>
  )
}
```

Parameter-Verwendung

```
// Beispiel: Countdown mit Timer
export const Countdown = ({ seconds }: { seconds: number }) => {
  const { t } = useI18n()

  return (
    <div>
      {t('chat.room.imageTimer', { seconds })}
      {/* Output (de): "Ansehen (5s)" */}
      {/* Output (en): "View (5s)" */}
    </div>
  )
}
```

Pluralisierung

```
// Erweiterte t() Funktion mit Plural-Support
const t = (key: string, params?: Record<string, string | number>) => {
  // ... (wie oben)

  // Plural-Handling
  if (params && 'count' in params) {
    const count = params.count as number
    const pluralKey = count === 1 ? `${key}_one` : `${key}_other`

    // Versuche plural key
    const pluralValue = getNestedValue(translations[lang], pluralKey)
    if (pluralValue) {
      return replaceParams(pluralValue, params)
    }
  }

  return replaceParams(value, params)
}

// JSON Beispiel:
{
  "chat": {
    "friendCount_one": "{{count}} Freund",
    "friendCount_other": "{{count}} Freunde"
  }
}

// Usage:
t('chat.friendCount', { count: 1 })  // "1 Freund"
t('chat.friendCount', { count: 5 })  // "5 Freunde"
```

5.3 Sprach-Speicherung

Speicherort

User-spezifische Sprache:

```
clubs/{clubId}/users/{uid}/language = "de" | "en" | "fr" | "es" | "it"
```

Club-Standard-Sprache:

```
clubs/{clubId}/config/settings/defaultLanguage = "de"
```

Sprach-Auswahl Flow

```
// packages/core/src/hooks/use-language.ts
import { doc, updateDoc } from 'firebase/firestore'
import { useI18n } from './use-i18n'

export const useLanguage = (clubId: string, userId: string, userData: any) => {
  const { t, lang, setLang } = useI18n(userData?.language)

  const changeLanguage = async (newLang: string) => {
    // Lokal sofort updaten (optimistic)
    setLang(newLang as any)

    // In Firestore speichern
    await updateDoc(doc(db, `clubs/${clubId}/users`, userId), {
      language: newLang
    })
  }

  return { t, lang, changeLanguage }
}
```

Sprach-Switcher Component

```tsx
// packages/ui/src/components/language-switcher.tsx
import { useLanguage } from '@nightlife/core'

const languages = [
  { code: 'de', name: 'Deutsch', flag: '🇩🇪' },
  { code: 'en', name: 'English', flag: '🇬🇧' },
  { code: 'fr', name: 'Français', flag: '🇫🇷' },
  { code: 'es', name: 'Español', flag: '🇪🇸' },
  { code: 'it', name: 'Italiano', flag: '🇮🇹' }
]

export const LanguageSwitcher = () => {
  const { lang, changeLanguage } = useLanguage()

  return (
    <div className="flex gap-2">
      {languages.map((l) => (
        <button
          key={l.code}
          onClick={() => changeLanguage(l.code)}
          className={`px-3 py-2 rounded-lg ${
            lang === l.code
              ? 'bg-cyan-600 text-white'
              : 'bg-slate-800 text-slate-400'
          }`}
        >
          {l.flag} {l.name}
        </button>
      ))}
    </div>
  )
}
```

Sprach-Fallback-Logik

```
1. User-Profil prüfen: clubs/{clubId}/users/{uid}/language
   ↓ (falls null)
2. Browser-Sprache prüfen: navigator.language
   ↓ (falls nicht unterstützt)
3. Club-Standard prüfen: clubs/{clubId}/config/settings/defaultLanguage
   ↓ (falls null)
4. System-Standard: "de"
```

Implementation:

```typescript
const detectLanguage = (userData: any, clubConfig: any): string => {
  // 1. User-Profil
  if (userData?.language) {
    return userData.language
  }

  // 2. Browser
  const browserLang = navigator.language.split('-')[0]
  if (['de', 'en', 'fr', 'es', 'it'].includes(browserLang)) {
    return browserLang
  }

  // 3. Club-Standard
  if (clubConfig?.defaultLanguage) {
    return clubConfig.defaultLanguage
  }

  // 4. System-Standard
  return 'de'
}
```

---

6. ZUSAMMENFASSUNG & NÄCHSTE SCHRITTE

Zusammenfassung

Diese Architektur-Dokumentation definiert die Transformation von einem Single-Club-Demo-System zu einer skalierbaren **Multi-Mandanten-SaaS-Plattform** für Club-Management.

Kernelemente

1. Monorepo-Struktur
- ✅ 6 Apps (club-app, dj-console, club-admin, staff-door, staff-waiter, staff-cloakroom)
- ✅ 3 Packages (core, ui, shared-types)
- ✅ Turborepo für Build-Optimierung
- ✅ Next.js 14+ mit App Router

2. Datenmodell
- ✅ Plattformebene: `platform/clubs`, `platform/groups`, `platform/users`
- ✅ Club-Ebene: `clubs/{clubId}/users`, `clubs/{clubId}/chats`, `clubs/{clubId}/state`, etc.
- ✅ Trust-System für Türsteher-App
- ✅ Bestellungs-System für Kellner-App
- ✅ Garderoben-System

3. Rollen & Rechte
- ✅ 8 Rollen (SUPER_ADMIN, CLUB_ADMIN, DJ, DOOR, WAITER, BAR, CLOAKROOM, GUEST)
- ✅ Granulare Firestore Security Rules
- ✅ Berechtigungsmatrix für alle Collections

4. Migration
- ✅ 7-Phasen-Plan (parallel zum Betrieb)

- ✅ Keine Downtime
- ✅ Datensicherung durch Backup

5. Mehrsprachigkeit
- ✅ 5 Sprachen (DE, EN, FR, ES, IT)
- ✅ User-spezifische Sprach-Persistierung
- ✅ Fallback-Logik

Nächste Schritte

Phase 1: Setup & Foundation (Woche 1-2)

Ziel: Monorepo aufbauen, Basis-Packages erstellen

Tasks:
1. ✅ Monorepo initialisieren (Turborepo + pnpm)
2. ✅ `packages/shared-types` erstellen (alle TypeScript-Typen)
3. ✅ `packages/core` erstellen
- Firebase Init
- Auth-Hooks
- Firestore-Hooks
- i18n-Hook
4. ✅ `packages/ui` erstellen
- Basis-Komponenten (Button, Input, Card, Modal)
- Icon-System
- i18n JSON-Files (DE + EN)
5. ✅ Firebase-Projekt aufsetzen
- Auth aktivieren (Email/Password)
- Firestore-Datenbank erstellen
- Security Rules deployen (Basis-Version)

Phase 2: Club-App Migration (Woche 3-4)

Ziel: Demo-App (index.php) nach Next.js migrieren

Tasks:
1. ✅ `apps/club-app` erstellen (Next.js)
2. ✅ Komponenten migrieren:
- LoginScreen
- HomeView
- ChatSystem (alle Sub-Komponenten)
- GlobalOverlay
3. ✅ Hooks integrieren:
- useAuth
- useUserData
- useClubState
- useFriends
- useChats

4. ✅ PWA-Setup (Service Worker, Manifest)
5. ✅ Testing (Unit + E2E)

Phase 3: DJ-Console Migration (Woche 5)

Ziel: Admin-App (admin.php) nach Next.js migrieren

Tasks:
1. ✅ `apps/dj-console` erstellen
2. ✅ Komponenten migrieren:
- AdminLogin
- LightControl
- AudioSync
- LotterySystem
- BroadcastMessages
- GuestList
3. ✅ Web Audio API Integration (Mikrofon-Sync)
4. ✅ Testing

Phase 4: Daten-Migration (Woche 6)

Ziel: Alt-Daten in neue Struktur migrieren

Tasks:
1. ✅ Platform/Club-Dokumente erstellen
2. ✅ Migrations-Scripts schreiben:
- Users migrieren
- Friends migrieren
- Requests migrieren
- Chats + Messages migrieren
3. ✅ Migrations-Scripts ausführen (Backup vorher!)
4. ✅ Validierung: Alt vs. Neu vergleichen

Phase 5: Club-Admin Dashboard (Woche 7-8)

Ziel: Admin-Dashboard für Club-Owner erstellen

Tasks:
1. ✅ `apps/club-admin` erstellen
2. ✅ Komponenten bauen:
- DashboardOverview (Statistiken)
- ClubSettings (Name, Farben, Features)
- StaffManager (Personal hinzufügen/entfernen)
- AnalyticsCharts (Recharts/Plotly)
3. ✅ Firestore-Operationen:
- Club-Daten updaten
- Staff-Rollen verwalten
4. ✅ Testing

Phase 6: Staff-Apps (Woche 9-11)

Ziel: Türsteher-, Kellner-, Garderoben-Apps bauen

Tasks:

Woche 9: Türsteher-App
1. ✅ `apps/staff-door` erstellen
2. ✅ QR-Scanner Integration (html5-qrcode)
3. ✅ Trust-System UI:
- User-Info anzeigen
- Trust-Level erhöhen
- Blacklist-Funktion
4. ✅ Check-In/Out durchführen

Woche 10: Kellner-App
1. ✅ `apps/staff-waiter` erstellen
2. ✅ Bestellungs-System:
- Bestellungen erstellen
- Bestellungen aktualisieren (Status)
- Tischplan anzeigen
3. ✅ Firestore-Operationen (`clubs/{clubId}/orders`)

Woche 11: Garderoben-App
1. ✅ `apps/staff-cloakroom` erstellen
2. ✅ Ticket-System:
- Ticket erstellen (QR-Code generieren)
- Ticket scannen
- Item ausgeben
3. ✅ Firestore-Operationen (`clubs/{clubId}/cloakroom`)

Phase 7: Testing & Deployment (Woche 12)

Ziel: Production-Ready machen

Tasks:
1. ✅ E2E-Tests für alle kritischen Flows
2. ✅ Performance-Optimierung (Lighthouse)
3. ✅ Security-Audit (Firestore Rules)
4. ✅ Deployment:
- Vercel (Club-App, DJ-Console, Club-Admin)
- Firebase Hosting (Alternative)
5. ✅ Monitoring Setup (Sentry, Firebase Analytics)
6. ✅ Dokumentation finalisieren

Phase 8: Features & Skalierung (ab Woche 13+)

Erweiterungen:
- 🔮 Subscription-System (Stripe Integration)

- 🔮 Multi-Club-Support (Club-Switcher in Apps)
- 🔮 Advanced Analytics (Custom Dashboards)
- 🔮 Push-Notifications (FCM)
- 🔮 Real-time Collaboration (Firestore Presence)
- 🔮 Admin-Super-Dashboard (alle Clubs verwalten)
- 🔮 Payment-Integration (In-App Bestellungen)
- 🔮 Social Features (Feed, Stories)

Erfolgskriterien

Technisch:
- ✅ Alle 6 Apps lauffähig
- ✅ Monorepo Build < 5 Minuten
- ✅ Lighthouse Score > 90
- ✅ 100% TypeScript Coverage
- ✅ Security Rules getestet

Funktional:
- ✅ Demo-System vollständig migriert
- ✅ Neue Features (Trust, Orders, Cloakroom) funktionsfähig
- ✅ Multi-Sprach-Support aktiv
- ✅ Rollen-System funktioniert

Business:
- ✅ Erster zahlender Club onboarded
- ✅ Subscription-System live
- ✅ Dokumentation vollständig

Ressourcen & Links

Dokumentation:
- Next.js Docs (https://nextjs.org/docs)
- Turborepo Docs (https://turbo.build/repo/docs)
- Firebase Docs (https://firebase.google.com/docs)
- Tailwind CSS Docs (https://tailwindcss.com/docs)

Tools:
- Firestore Security Rules Playground (https://firebase.google.com/docs/rules/simulator)
- TypeScript Playground (https://www.typescriptlang.org/play)
- Excalidraw (https://excalidraw.com/) (für Diagramme)

Community:
- Next.js Discord (https://nextjs.org/discord)
- Firebase Discord (https://discord.gg/firebase)
- Turborepo Discord (https://turbo.build/repo/docs/community)

Ende der Architektur-Dokumentation

Letzte Aktualisierung: 1. Dezember 2025
Version: 2.0
Autor: Nightlife OS Team