Firestore-Schema - Nightlife OS

Übersicht

Dieses Dokument beschreibt das vollständige Firestore-Datenmodell für Nightlife OS.

Wichtig: Die detaillierte Dokumentation mit allen Feldern, Datentypen, Security Rules und Berechtigungen findest du in:

→ ARCHITECTURE.md - Abschnitt 2: FIRESTORE-DATENMODELL (./ARCHITECTURE.md#2-firestore-datenmodell)

Struktur-Übersicht

Plattformebene (Root-Level)

Mandanten-übergreifende Daten für das SaaS-System:

```
platform/
├── clubs/{clubId}              # Stammdaten aller Clubs
├── groups/{groupId}            # Club-Ketten/Gruppen
└── users/{uid}                 # Globaler User-Stamm
```

Club-Ebene (Mandanten-Daten)

Club-spezifische Daten (isoliert pro Club):

```
clubs/{clubId}/
├── users/{uid}                 # Club-spezifische User-Daten
│   ├── friends/{friendId}     # Freundschaften
│   └── requests/{requesterId} # Freundschaftsanfragen
├── state/
│   └── global                 # Globaler Club-State (Lichtshow, etc.)
├── chats/{chatId}             # Chat-Metadaten
│   └── messages/{messageId}   # Chat-Nachrichten
├── orders/{orderId}           # Bestellungen
├── cloakroom/{ticketId}       # Garderoben-Tickets
└── config/
    └── settings               # Club-Einstellungen
```

Wichtige Collections

`platform/clubs/{clubId}`

Zweck: Stammdaten aller Clubs
Felder: name, slug, ownerId, subscriptionTier, address, features, theme, etc.
Berechtigung: Super-Admins (alle), Club-Owners (eigener Club)

`platform/users/{uid}`

Zweck: Globaler User-Account (plattformweit)
Felder: email, displayName, isPlatformAdmin, ownedClubs, memberClubs, etc.
Berechtigung: Jeder User (eigenes Dokument), Super-Admins (alle)

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

Zweck: User im Kontext eines spezifischen Clubs
Felder:
- Basis: email, displayName, roles, checkedIn, language, friendCode
- Trust-System: phoneVerified, trustedLevel, verifiedBy, blacklisted
- Historie: visitCount, lastVisits
Berechtigung: User selbst (eigenes Dokument), Staff (alle), Admins (alle)

`clubs/{clubId}/state/global`

Zweck: Globale Club-Zustände für Realtime-Features
Felder:
- Mode: mode (“normal”, “lightshow”, “message”, “countdown”, “lottery_result”)
- Lichtshow: lightColor, lightEffect, audioSyncIntensity
- Countdown: countdownActive, countdownEnd, countdownMessage
- Gewinnspiel: activeGame, winnerIds, prizeCode
- Broadcast: messageText, messageTarget
Berechtigung: Alle (read), DJ/Admin (write)

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

Zweck: Chat-Metadaten
Chat-ID-Format:
- 1:1: `{uid1}_{uid2}` (alphabetisch sortiert)
- Gruppe: Auto-generierte ID
Felder: type, name, participants, createdBy, lastMessageAt, lastMessagePreview
Berechtigung: Nur Participants

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

Zweck: Chat-Nachrichten
Felder: text, sender, senderName, image, ephemeral, viewedBy, createdAt, expiresAt
Berechtigung: Nur Participants (read), Sender (write/delete eigene)

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

Zweck: Bestellungen (Kellner/Bar)
Felder: userId, table, items, totalPrice, status, createdBy, paymentMethod
Berechtigung: Nur Staff (waiter, bar, admin)

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

Zweck: Garderoben-Tickets
Felder: ticketId, userId, itemDescription, depositedAt, retrievedAt, status
Berechtigung: Nur Staff (cloakroom, admin)

`clubs/{clubId}/config/settings`

Zweck: Club-Einstellungen
Felder: features, theme, openingHours, checkInRadius, location, languages, trustModeEnabled
Berechtigung: Alle (read), Admin (write)

Rollen & Berechtigungen

Rollen-Hierarchie

```
SUPER_ADMIN (Plattform)
└── CLUB_ADMIN (Club)
    ├── DJ
    └── STAFF
        ├── DOOR
        ├── WAITER
        ├── BAR
        └── CLOAKROOM
    └── GUEST (Standard)
```

Berechtigungsmatrix

Die vollständige Matrix findest du in `ARCHITECTURE.md` Abschnitt 3.2.

Wichtigste Regeln:
- **Super-Admins:** Voller Zugriff auf alle Plattform-Daten
- **Club-Admins:** Voller Zugriff auf eigenen Club
- **DJ:** Lesen aller Club-Daten, Schreiben auf `state/global`
- **Staff (Door):** User-Verifizierung, Check-In/Out
- **Staff (Waiter/Bar):** Bestellungen verwalten
- **Staff (Cloakroom):** Garderoben-Tickets verwalten
- **Guests:** Eigene Daten, Chat mit Freunden, Lichtshow sehen

---

Security Rules

Die kompletten Firestore Security Rules sind in `ARCHITECTURE.md` Abschnitt 3.3 dokumentiert.

Wichtigste Prinzipien:

1. **Multi-Tenancy:** Jeder Club ist isoliert - User können nur Daten ihres Clubs sehen
2. **Rollen-basiert:** Zugriff basiert auf `roles`-Array in User-Dokument
3. **Participants-Only:** Chats nur für Teilnehmer lesbar
4. **Sender-Restriction:** Nur Sender kann eigene Nachrichten löschen
5. **Admin-Override:** Super-Admins und Club-Admins haben erweiterte Rechte

Beispiel-Rule (Chat-Zugriff)

```
match /clubs/{clubId}/chats/{chatId}/messages/{messageId} {
  // Nur Participants können lesen
  allow read: if request.auth != null &&
    request.auth.uid in get(/databases/$(database)/documents/clubs/$(clubId)/chats/$
(chatId)).data.participants;

  // Participants können Nachrichten senden
  allow create: if request.auth != null &&
    isChatParticipant(clubId, chatId) &&
    request.resource.data.sender == request.auth.uid;

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

TypeScript-Typen

Alle Firestore-Dokumente haben entsprechende TypeScript-Typen in:

→ `packages/shared-types/src/*`

Wichtigste Typen

```typescript
// packages/shared-types/src/user.ts
export interface PlatformUser {
  uid: string;
  email: string;
  displayName: string | null;
  isPlatformAdmin: boolean;
  ownedClubs: string[];
  memberClubs: string[];
  createdAt: number;
  lastSeenAt: number;
}

export interface ClubUser {
  uid: string;
  email: string;
  displayName: string | null;
  roles: string[];
  checkedIn: boolean;
  checkedInAt: number | null;
  language: string;
  friendCode: string;
  trustedLevel: number;
  // ... weitere Felder
}

// packages/shared-types/src/club.ts
export interface Club {
  clubId: string;
  name: string;
  slug: string;
  ownerId: string;
  subscriptionTier: string;
  features: string[];
  theme: {
    primaryColor: string;
    secondaryColor: string;
    logo?: string;
  };
  // ... weitere Felder
}

// packages/shared-types/src/chat.ts
export interface Chat {
  chatId: string;
  type: 'private' | 'group';
  participants: string[];
  name?: string;
  createdBy?: string;
  lastMessageAt: number;
  lastMessagePreview?: string;
}

export interface Message {
  messageId: string;
  text: string;
  sender: string;
  senderName: string;
  image?: string;
  ephemeral?: number;
  viewedBy: string[];
  createdAt: number;
  expiresAt?: number;
```

```
  deleted: boolean;
}
```

Migration vom Demo-System

Das bisherige Demo-System verwendet eine flache Struktur ohne Multi-Tenancy:

```
ALT (Demo):
users/{uid}
state/global
chats/{chatId}/messages
```

Die Migration auf das neue Schema ist in `ARCHITECTURE.md` Abschnitt 4 dokumentiert.

Wichtig: Das alte Schema wird NICHT gelöscht - beide Systeme laufen parallel während der Migration.

Weitere Informationen

Für detaillierte Informationen siehe:

- **ARCHITECTURE.md (./ARCHITECTURE.md)** - Vollständige Architektur-Dokumentation
- **README.md (./README.md)** - Projekt-Übersicht und Entwicklungs-Guide
- **DEPLOYMENT.md (./DEPLOYMENT.md)** - Deployment-Anleitung

Letzte Aktualisierung: 1. Dezember 2025
Version: 2.0