Nightlife OS - Architecture Addendum: Age, Trust & Birthday System

Erweiterungen für Phase 10+: Alterslogik, Jugendschutz, Trusted User & Geburtstags-Features

Version: 1.0
Erstellt am: 3. Dezember 2025
Basis: ARCHITECTURE.md (Phase 1-8), ARCHITECTURE_ADDENDUM_INTERACTIVE_STATUS.md (Phase 9)
Zweck: Altersverifizierung, automatisches Level-Up, erweiterte Trust-Mechanismen, Geburtstagsbenefits

---

Inhaltsverzeichnis

---

1. ÜBERBLICK & MOTIVATION

Dieses Addendum erweitert die bestehende Nightlife OS-Architektur um **drei zentrale Konzepte**:

Warum diese Erweiterungen?

Problem 1: Keine Altersprüfung

Herausforderung: System kann nicht zwischen Minderjährigen und Erwachsenen unterscheiden → Rechtliche Risiken bei Alkoholausschank.

Lösung: Geburtsdatum-basierte Alterslogik mit automatischem Level-Up und Altersgruppen-Segmentierung.

Problem 2: Unvollständiger Trust-Mechanismus

Herausforderung: Bestehendes Trust-System (aus Addendum 1) berücksichtigt kein Alter → Minderjährige könnten theoretisch Alkohol bestellen.

Lösung: Erweiterter Trust-Mechanismus mit Alters- UND Identitätsprüfung als Voraussetzung.

Problem 3: Fehlender Geburtstags-Context

Herausforderung: Keine automatische Erkennung von Geburtstagen → Verpasste Marketing-Chancen (Free Drinks, VIP-Treatment).

Lösung: Birthday-Perk-System mit Club-spezifischen Einstellungen und DJ-Ansagen.

Problem 4: Geo-Locating wird fälschlicherweise angenommen

Herausforderung: Frühere Dokumentationen könnten den Eindruck erwecken, dass aktives GPS-Tracking verwendet wird.

Lösung: Explizite Klarstellung, dass KEIN Geo-Locating verwendet wird.

Architektur-Prinzipien dieses Addendums

1. **Privacy-First:** Geburtsdatum wird nur wenn nötig gespeichert, Alter wird berechnet (nicht gespeichert)
2. **Server-Side-Validation:** Alle Altersprüfungen erfolgen serverseitig (nicht umgehbar)
3. **Automatisierung:** Altersgruppen-Wechsel und Geburtstags-Erkennung sind automatisch
4. **Club-Autonomie:** Jeder Club kann Birthday-Perks individuell konfigurieren
5. **No GPS:** Bestätigung der Anwesenheit nur durch QR-Check-In, nicht durch Standort

---

2. GEO-LOCATING: NICHT VERWENDEN (KLARSTELLUNG)

Wichtige Klarstellung

Das Nightlife OS-System verwendet **KEIN aktives Geo-Locating** (GPS-Tracking) für die Bestimmung der Club-Anwesenheit oder für Teilnahme-Logik.

Warum kein Geo-Locating?

Grund	Erklärung
Datenschutz	GPS-Tracking ist rechtlich problematisch (DSGVO) und erfordert explizite Einwilligung
Akku-Verbrauch	Permanente GPS-Abfragen entleeren den Akku schnell
Ungenauigkeit	GPS funktioniert in Gebäuden schlecht (metallische Wände, Keller)
User-Friktion	User müssen GPS-Berechtigung erteilen → erhöht Abbruchrate
Manipulation	GPS kann durch Mock-Location-Apps gefälscht werden

Wie wird “Ist im Club?” stattdessen beantwortet?

Die Anwesenheit im Club wird durch **folgende Mechanismen** bestätigt:

1. QR-Code-Check-In beim Eingang

- **Wer:** Türsteher scannt User-QR-Code (oder User scannt Club-QR-Code)
- **Wo:** Am Eingang des Clubs
- **Effekt:** Setzt `isInClub = true` und `currentClubId = clubId`
- **Vorteil:** 100% sicher, keine GPS-Berechtigung nötig

2. 6-Zahlen-Interaktionssystem

- **Wer:** User wählt eine der 6 täglichen Zahlen in der App
- **Wo:** Innerhalb der Club-App (bei aktivem Check-In)
- **Effekt:** Setzt `partyModeActive = true` und `selectedNumber = X`
- **Vorteil:** Beweist, dass User die DJ-Ansage hört (= physisch anwesend)

3. Party-Modus mit 30-Minuten-Timer

- **Wer:** User muss alle 30 Minuten neu eine Zahl wählen
- **Wo:** In der App
- **Effekt:** Erneuert `partyModeLastActivatedAt`
- **Vorteil:** Stellt sicher, dass User noch im Club ist (sonst hört er DJ-Ansagen nicht)

4. Manueller Check-Out oder Auto-Check-Out

- **Wer:** User checkt sich aus oder System nach 6 Stunden Inaktivität
- **Effekt:** Setzt `isInClub = false`
- **Vorteil:** Klare Status-Verwaltung ohne GPS

Was ist mit `lastKnownLocation`?

Falls in früheren Dokumentationen (z.B. ARCHITECTURE.md v1.0) ein Feld wie `lastKnownLocation` erwähnt wurde:

```
// DEPRECATED - Nicht mehr in aktiver Nutzung
interface UserDocument {
  // ...
  lastKnownLocation?: { lat: number; lng: number } | null; // ❌ Optional, aktuell
NICHT genutzt
}
```

Status:
- ✅ Kann **optional** im Schema bleiben (für zukünftige Features wie “Clubs in meiner Nähe anzeigen”)
- ❌ Wird **NICHT** für Check-In-Logik, Party-Modus oder Trust-System verwendet
- ❌ Wird **NICHT** automatisch erfasst (nur wenn User explizit “Standort teilen” aktiviert)

Zusammenfassung: Anwesenheits-Architektur

```
User betritt Club
  ↓
[Türsteher scannt QR-Code] ODER [User scannt Club-QR-Code]
  ↓
isInClub = true, currentClubId = clubId
  ↓
User öffnet App, sieht 6 tägliche Zahlen
  ↓
User hört DJ-Ansage ("Wählt die 42!")
  ↓
User wählt Zahl 42 in App
  ↓
partyModeActive = true, selectedNumber = 42
  ↓
DJ startet Aktion → System findet alle User mit selectedNumber = 42
  ↓
Nach 30 Minuten ohne neue Zahlen-Eingabe: partyModeActive = false
  ↓
Nach 6 Stunden ohne Aktivität: isInClub = false (Auto-Check-Out)
```

Kein GPS-Tracking an irgendeiner Stelle in diesem Flow!

---

3. ALTER, ALTERSGRUPPEN & AUTOMATISCHES LEVEL-UP

Konzept

Das **Alters-System** ermöglicht es, User basierend auf ihrem **Geburtsdatum** automatisch in Altersgruppen einzuteilen und bei Erreichen von Altersgrenzen (16, 18, 21) automatisch “hochzustufen”.

Kern-Idee

- User gibt einmalig Geburtsdatum an (z.B. bei Registrierung oder bei erster Bestellung)
- System berechnet **bei jedem Login** und bei kritischen Aktionen das aktuelle Alter
- System ordnet User automatisch in `ageGroup` ein: `u16`, `u18`, `u21`, `adult`
- Bei Überschreiten einer Altersgrenze: **Automatisches Level-Up** ohne manuelle Aktion

Warum Geburtsdatum statt “Alter”-Feld?

- ✅ **Automatische Aktualisierung:** Alter ändert sich jedes Jahr automatisch
- ✅ **Geburtstags-Erkennung:** Ermöglicht Birthday-Perks (siehe Kapitel 5)
- ✅ **Audit-Trail:** Club kann nachweisen, dass Alter zum Zeitpunkt X korrekt war

---

3.1 Datenmodell-Erweiterungen

User-Dokument: `clubs/{clubId}/users/{uid}`

Neue Felder für Alterslogik:

Feldname	Datentyp	Beschreibung	Beispielwert
`dateOfBirth`	`string | null`	Geburtsdatum im ISO-Format (YYYY-MM-DD)	`"2004-05-12"`
`ageVerified`	`boolean`	Ausweis durch Club/Personal geprüft?	`true`
`ageGroup`	`'u16' | 'u18' | 'u21' | 'adult' | null`	Berechnete Altersgruppe (automatisch)	`"u18"`

Wichtig: Das Feld `ageGroup` wird **NIEMALS manuell gesetzt**, sondern immer aus `dateOfBirth` berechnet!

Platform-User-Dokument: `platform/users/{uid}` (Global)

Neue Felder (synchronisiert):

Feldname	Datentyp	Beschreibung	Beispielwert
`dateOfBirth`	`string | null`	Geburtsdatum (global)	`"2004-05-12"`
`ageVerified`	`boolean`	Ausweis verifiziert?	`true`

Synchronisations-Logik:
- `dateOfBirth` und `ageVerified` werden in `platform/users/{uid}` **UND** in allen `clubs/{clubId}/users/{uid}` gespeichert
- Bei Update in einem Club → Cloud Function synchronisiert zu `platform/users/{uid}`
- Bei Update in Platform → Cloud Function synchronisiert zu allen Club-Dokumenten

---

3.2 Kernregeln & Business Logic

Regel 1: Alters-Berechnung bei jedem relevanten Event

Wann wird Alter berechnet?
- ✅ Bei jedem **Login** (in AuthContext)
- ✅ Beim **Laden des Userprofils** (in useUserData Hook)
- ✅ Bei **kritischen Aktionen**:
- Bestellung aufgeben
- Check-In durchführen
- Dating-Status ändern (wenn relevant)
- Birthday-Perk einlösen

Wo wird berechnet?
- **Client-seitig** für UI-Anzeige (z.B. “Du wirst bald 18! 🎉”)
- **Server-seitig** für Validierung (Cloud Functions, Security Rules)

Regel 2: Altersgruppen-Definitionen

```
type AgeGroup = 'u16' | 'u18' | 'u21' | 'adult';

// Logik:
// u16   = Alter < 16   (Kind/Jugendlicher)
// u18   = 16 <= Alter < 18
// u21   = 18 <= Alter < 21
// adult = Alter >= 21
```

Land-spezifische Anpassung (optional für später):
- In den USA: Relevante Grenze ist 21 (Alkohol)
- In Deutschland: Relevante Grenze ist 18 (harter Alkohol), 16 (Bier/Wein)
- In Zukunft kann `Club.settings.country` die Logik beeinflussen

Regel 3: Automatisches Level-Up

Szenario: User wird 18

```
Heute: 11. Mai 2025, User-Geburtsdatum: 12. Mai 2007
  ↓
ageGroup = 'u18'
  ↓
Morgen: 12. Mai 2025 (Geburtstag!)
  ↓
User öffnet App → calculateAgeGroup() wird ausgeführt
  ↓
Alter = 18 → ageGroup = 'u21' (in Deutschland) oder 'adult' (USA-Clubs)
  ↓
System updated ageGroup automatisch im Firestore
  ↓
KEINE manuelle Aktion nötig!
```

Implementierungs-Optionen:

Option A: Lazy Evaluation (Empfohlen für Phase 10)

- Alter wird bei **jedem Login** neu berechnet
- Wenn `ageGroup` sich geändert hat → Update in Firestore

- **Vorteil:** Einfach, keine Cron-Jobs nötig
- **Nachteil:** Update erfolgt erst beim nächsten Login

```
// In AuthContext oder useUserData Hook
useEffect(() => {
  if (userData?.dateOfBirth) {
    const currentAgeGroup = calculateAgeGroup(userData.dateOfBirth, new Date());

    if (currentAgeGroup !== userData.ageGroup) {
      // Level-Up!
      updateDoc(userDocRef, { ageGroup: currentAgeGroup });

      // Optional: Zeige Congratulations-Nachricht
      if (currentAgeGroup === 'adult') {
        toast.success('Du bist jetzt volljährig! 🎉');
      }
    }
  }
}, [userData]);
```

Option B: Scheduled Cloud Function (Production-Grade)

- Täglich um 00:01 Uhr: Prüfe alle User mit `dateOfBirth`
- Wenn heute Geburtstag → Update `ageGroup`
- **Vorteil:** Update erfolgt pünktlich um Mitternacht
- **Nachteil:** Höhere Kosten bei vielen Usern

```
// Cloud Function (Cron-Job)
export const updateAgeGroups = functions.pubsub
  .schedule('1 0 * * *') // Täglich um 00:01 Uhr
  .timeZone('Europe/Berlin')
  .onRun(async () => {
    const today = new Date();

    // Hole alle User mit Geburtsdatum
    const usersSnapshot = await db.collectionGroup('users')
      .where('dateOfBirth', '!=', null)
      .get();

    const batch = db.batch();
    let updateCount = 0;

    for (const doc of usersSnapshot.docs) {
      const user = doc.data();
      const currentAgeGroup = calculateAgeGroup(user.dateOfBirth, today);

      if (currentAgeGroup !== user.ageGroup) {
        batch.update(doc.ref, { ageGroup: currentAgeGroup });
        updateCount++;

        // Optional: Sende Geburtstags-Notification
        if (isBirthdayToday(user.dateOfBirth, today)) {
          // Siehe Kapitel 5
        }
      }
    }

    await batch.commit();
    console.log(`Updated ${updateCount} user age groups`);
  });
```

Empfehlung für Phase 10: Nutze **Option A** (Lazy Evaluation) für Einfachheit. Später bei Scale → Option B.

Regel 4: Altersgruppe ist IMMER abgeleitet

```
ageGroup wird NIEMALS manuell gesetzt
ageGroup wird IMMER aus dateOfBirth berechnet
Client kann ageGroup NICHT überschreiben
```

Security Rules (Firestore):

```
// User darf ageGroup NICHT selbst ändern
match /clubs/{clubId}/users/{uid} {
  allow update: if request.auth.uid == uid &&
    !request.resource.data.diff(resource.data).affectedKeys().hasAny(['ageGroup', 'dat
eOfBirth', 'ageVerified']);

  // Nur Staff/Admin darf dateOfBirth und ageVerified setzen
  allow update: if hasAnyRole(clubId, ['admin', 'staff', 'door']) &&
    request.resource.data.diff(resource.data).affectedKeys().hasOnly(['dateOfBirth', '
ageVerified', 'ageGroup']);
}
```

Regel 5: `ageVerified` sagt aus, ob Ausweis geprüft wurde

```
ageVerified = false → Geburtsdatum selbst angegeben (nicht vertrauenswürdig)
ageVerified = true → Ausweis durch Türsteher/Personal geprüft
```

Wer kann `ageVerified` setzen?
- ✅ Türsteher (via `staff-door` App)
- ✅ Club-Admin (via `club-admin` Dashboard)
- ❌ User selbst (würde sofort in Security Rules blockiert)

Verwendung:
- Clubs können entscheiden: "Alkohol nur mit `ageVerified = true`"
- Oder: "Hochprozentige Getränke nur mit `ageVerified = true`"

3.3 Funktionen & Interfaces

TypeScript Interfaces

```typescript
// packages/shared-types/src/user.ts

export type AgeGroup = 'u16' | 'u18' | 'u21' | 'adult';

export interface UserAgeFields {
  /** Geburtsdatum im ISO-Format (YYYY-MM-DD) */
  dateOfBirth: string | null;

  /** Ausweis durch Club/Personal geprüft? */
  ageVerified: boolean;

  /** Berechnete Altersgruppe (automatisch, nicht manuell setzbar) */
  ageGroup: AgeGroup | null;
}

// Erweitere UserDocument
export interface UserDocument extends UserAgeFields {
  // ... bestehende Felder (uid, email, roles, etc.)
}
```

Core Functions (mit vollständigem Pseudocode)

```typescript
// packages/core/src/utils/age-system.ts

/**
 * Berechnet das Alter aus einem Geburtsdatum
 * @param dateOfBirth - ISO-String (YYYY-MM-DD)
 * @param now - Aktuelles Datum (für Tests mockbar)
 * @returns Alter in Jahren
 */
export function calculateAge(dateOfBirth: string, now: Date = new Date()): number {
  const birthDate = new Date(dateOfBirth);

  let age = now.getFullYear() - birthDate.getFullYear();
  const monthDiff = now.getMonth() - birthDate.getMonth();

  // Geburtstag dieses Jahr noch nicht erreicht?
  if (monthDiff < 0 || (monthDiff === 0 && now.getDate() < birthDate.getDate())) {
    age--;
  }

  return age;
}

/**
 * Berechnet die Altersgruppe aus einem Geburtsdatum
 * @param dateOfBirth - ISO-String (YYYY-MM-DD)
 * @param now - Aktuelles Datum
 * @param country - Land-Code für länderspezifische Regeln (optional, später)
 * @returns Altersgruppe
 */
export function calculateAgeGroup(
  dateOfBirth: string,
  now: Date = new Date(),
  country: string = 'DE' // Default: Deutschland
): AgeGroup {
  const age = calculateAge(dateOfBirth, now);

  // Standard-Logik (Deutschland/EU)
  if (age < 16) return 'u16';
  if (age < 18) return 'u18';
  if (age < 21) return 'u21';
  return 'adult';

  // Zukünftig: Land-spezifische Logik
  // if (country === 'US') {
  //   if (age < 18) return 'u18';
  //   if (age < 21) return 'u21';
  //   return 'adult';
  // }
}

/**
 * Prüft, ob User alt genug für eine bestimmte Aktion ist
 * @param user - User-Dokument
 * @param minAge - Mindestalter (z.B. 18)
 * @returns true wenn User alt genug, false wenn nicht
 */
export function isAgeEligible(user: UserDocument, minAge: number): boolean {
  if (!user.dateOfBirth) return false;

  const age = calculateAge(user.dateOfBirth);
  return age >= minAge;
}
```

```
/**
 * Aktualisiert die Altersgruppe eines Users (wenn nötig)
 * Wird bei jedem Login aufgerufen
 * @param userId - User-UID
 * @param clubId - Club-ID
 */
export async function updateAgeGroupIfNeeded(
  userId: string,
  clubId: string
): Promise<void> {
  const userDocRef = doc(db, `clubs/${clubId}/users/${userId}`);
  const userDoc = await getDoc(userDocRef);
  const user = userDoc.data() as UserDocument;

  if (!user.dateOfBirth) return; // Kein Geburtsdatum gesetzt

  const currentAgeGroup = calculateAgeGroup(user.dateOfBirth);

  // Hat sich Altersgruppe geändert?
  if (currentAgeGroup !== user.ageGroup) {
    await updateDoc(userDocRef, { ageGroup: currentAgeGroup });

    console.log(`User ${userId} age group updated: ${user.ageGroup} → ${current-
AgeGroup}`);

    // Optional: Zeige Notification
    if (currentAgeGroup === 'adult') {
      // TODO: Sende Congratulations-Notification
    }
  }
}

/**
 * Setzt Geburtsdatum und ageVerified (nur für Staff/Admin)
 * @param userId - User-UID
 * @param clubId - Club-ID
 * @param dateOfBirth - ISO-String (YYYY-MM-DD)
 * @param verified - Wurde Ausweis geprüft?
 */
export async function setDateOfBirth(
  userId: string,
  clubId: string,
  dateOfBirth: string,
  verified: boolean = false
): Promise<void> {
  const ageGroup = calculateAgeGroup(dateOfBirth);

  // Update in Club-Dokument
  await updateDoc(doc(db, `clubs/${clubId}/users/${userId}`), {
    dateOfBirth,
    ageVerified: verified,
    ageGroup
  });

  // Update in Platform-Dokument (für globale Synchronisation)
  await updateDoc(doc(db, `platform/users/${userId}`), {
    dateOfBirth,
    ageVerified: verified
  });
}
```

3.4 Firestore-Schema

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

Erweiterte Felder:

```
{
  // ... bestehende Felder (uid, email, displayName, roles, isInClub, etc.)

  // NEU: Alterslogik
  "dateOfBirth": "2004-05-12",              // string | null (ISO-Format YYYY-MM-DD)
  "ageVerified": true,                      // boolean
  "ageGroup": "u21"                         // "u16" | "u18" | "u21" | "adult" | null
}
```

Collection: `platform/users/{uid}` (Global)

Erweiterte Felder:

```
{
  // ... bestehende Felder (uid, email, isPlatformAdmin, etc.)

  // NEU: Alterslogik (global)
  "dateOfBirth": "2004-05-12",              // string | null
  "ageVerified": true                       // boolean
}
```

Firestore Composite Indexes (falls Query nötig)

Falls später Queries wie “Alle U18 im Club” nötig sind:

```
clubs/{clubId}/users
  - isInClub (ASC) + ageGroup (ASC)
```

---

4. TRUSTED USER & JUGENDSCHUTZ (ERWEITERT)

Konzept

Das **erweiterte Trust-System** kombiniert das bestehende Trust-Level-System (aus Addendum 1) mit der neuen **Alterslogik**, um einen vollständigen Jugendschutz- und Zechpreller-Schutz zu implementieren.

Erweiterungen gegenüber Addendum 1

Addendum 1 (Phase 9)	Addendum 2 (Phase 10)
`phoneVerified` erforderlich	`phoneVerified` + `ageVerified` + `Alter >= 18`
`trustLevel: 'normal' | 'trusted'`	Bleibt gleich
Keine Altersprüfung	**NEU:** Produkt-basierte Altersgrenzen
Keine Jugendschutz-Notifications	**NEU:** Automatische Jugendschutz-Benachrichtigungen

---

4.1 Datenmodell-Erweiterungen

User-Dokument: `clubs/{clubId}/users/{uid}`

Trust-Felder (aus Addendum 1, unverändert):
- `phoneVerified: boolean`
- `trustLevel: 'normal' | 'trusted'`
- `trustedVerifiedAt: number | null`
- `trustedVerifiedByClubId: string | null`

Neue Integration mit Alterslogik:
- Für `trustLevel = 'trusted'` muss jetzt AUCH gelten:
- `ageVerified = true`
- Berechnetes Alter >= 18 (oder club-/länderspezifische Grenze)

Produkt-Dokument: `clubs/{clubId}/products/{productId}` (NEU)

Neues Schema für Bestellungs-System:

Feldname	Datentyp	Beschreibung	Beispielwert
`productId`	`string`	Produkt-ID	Auto-generiert
`name`	`string`	Produktname	`"Bier (0,5l)"`
`category`	`string`	Kategorie	`"beer"`, `"spirits"`, `"softdrinks"`, `"food"`
`price`	`number`	Preis in EUR	`4.50`
`minAge`	`number | null`	Mindestalter für Kauf	`16`, `18`, `21`, `null`
`requiresAgeVerification`	`boolean`	Ausweis-Prüfung erforderlich?	`true`
`available`	`boolean`	Verfügbar?	`true`

Beispiel-Produkte:

```
// Softdrink (keine Altersbeschränkung)
{
  "productId": "prod_cola",
  "name": "Cola (0,3l)",
  "category": "softdrinks",
  "price": 3.50,
  "minAge": null,
  "requiresAgeVerification": false,
  "available": true
}

// Bier (ab 16 in Deutschland)
{
  "productId": "prod_beer",
  "name": "Bier (0,5l)",
  "category": "beer",
  "price": 4.50,
  "minAge": 16,
  "requiresAgeVerification": false,  // Selbstangabe OK
  "available": true
}

// Wodka (ab 18, Ausweis erforderlich)
{
  "productId": "prod_vodka",
  "name": "Wodka Shot",
  "category": "spirits",
  "price": 5.00,
  "minAge": 18,
  "requiresAgeVerification": true,   // ageVerified muss true sein
  "available": true
}
```

Club-Settings: `clubs/{clubId}/config/settings`

Neue Felder für Jugendschutz:

Feldname	Datentyp	Beschreibung	Beispielwert
`youthCurfewTime`	`string | null`	Sperrstunde für U18 (Format: “HH:MM”)	`"00:00"`
`youthCurfewEnabled`	`boolean`	Jugendschutz-Sperrstunde aktiv?	`true`
`minAgeForEntry`	`number`	Mindestalter für Einlass	`16`, `18`, `21`

4.2 Kernregeln & Trust-Mechanismus

Regel 1: Erweiterte Voraussetzungen für `trustLevel = 'trusted'`

```
trustLevel = 'trusted' ERFORDERT:
  ✅ phoneVerified = true
  ✅ ageVerified = true
  ✅ dateOfBirth vorhanden
  ✅ Berechnetes Alter >= 18 (oder club-/länderspezifischer Wert)
```

Implementierung (Server-seitige Validierung):

```
function canSetTrustedLevel(user: UserDocument, club: Club): { allowed: boolean; reason?: string } {
  // 1. Phone-Verifizierung
  if (!user.phoneVerified) {
    return { allowed: false, reason: 'Telefonnummer muss verifiziert sein' };
  }

  // 2. Alters-Verifizierung
  if (!user.ageVerified) {
    return { allowed: false, reason: 'Ausweis muss geprüft werden' };
  }

  // 3. Geburtsdatum vorhanden?
  if (!user.dateOfBirth) {
    return { allowed: false, reason: 'Geburtsdatum muss angegeben werden' };
  }

  // 4. Mindestalter
  const age = calculateAge(user.dateOfBirth);
  const minAge = club.settings?.minAgeForTrusted || 18; // Default: 18

  if (age < minAge) {
    return { allowed: false, reason: `Mindestalter für Trusted: ${minAge} Jahre` };
  }

  return { allowed: true };
}
```

Regel 2: Trusted-Verifizierung durch Türsteher (erweiterter Flow)

Neuer Ablauf in `staff-door` App:

```
1. Türsteher scannt User-QR-Code
   ↓
2. App zeigt User-Info:
   - Name, Foto
   - phoneVerified-Status
   - Geburtsdatum (falls vorhanden)
   - Aktuelles Alter (berechnet)
   ↓
3. Türsteher wählt: "Ausweis prüfen"
   ↓
4. Türsteher gibt Geburtsdatum vom Ausweis ein (falls noch nicht vorhanden)
   ↓
5. System berechnet Alter
   ↓
6. Wenn Alter >= 18 UND phoneVerified = true:
   ↓
   Button "Als Trusted markieren" wird aktiv
   ↓
7. Türsteher klickt Button
   ↓
8. System setzt:
   - ageVerified = true
   - ageGroup = calculateAgeGroup(dateOfBirth)
   - trustLevel = 'trusted' (falls alle Bedingungen erfüllt)
```

4.3 Altersgrenzen für Bestellungen

Regel 1: Produkt-basierte Altersvalidierung

```
User bestellt Produkt
  ↓
System prüft:
  1. Hat User dateOfBirth?
  2. Ist berechnetes Alter >= product.minAge?
  3. Falls product.requiresAgeVerification = true: Ist ageVerified = true?
  ↓
Falls NEIN bei einem der Punkte → Bestellung ABGELEHNT
```

Regel 2: Serverseitige Validierung (KRITISCH!)

Die Validierung MUSS serverseitig erfolgen (Cloud Function), da Client-seitige Checks umgehbar sind.

Cloud Function: validateOrder

```
// functions/src/orders.ts

export const validateOrder = functions.https.onCall(async (data, context) => {
  const { userId, clubId, productId, quantity } = data;

  if (!context.auth) {
    throw new functions.https.HttpsError('unauthenticated', 'User muss eingeloggt
sein');
  }

  // 1. Hole User-Daten
  const userDoc = await db.doc(`clubs/${clubId}/users/${userId}`).get();
  const user = userDoc.data() as UserDocument;

  // 2. Hole Produkt-Daten
  const productDoc = await db.doc(`clubs/${clubId}/products/${productId}`).get();
  const product = productDoc.data() as Product;

  // 3. Basis-Check: Phone-Verifizierung
  if (!user.phoneVerified) {
    throw new functions.https.HttpsError(
      'permission-denied',
      'Telefonnummer muss verifiziert sein. Gehe zu Einstellungen → Telefon verifiz-
ieren.'
    );
  }

  // 4. Alterscheck
  const canOrder = canUserOrderProduct(user, product);
  if (!canOrder.allowed) {
    throw new functions.https.HttpsError('permission-denied', canOrder.reason);
  }

  // 5. Erstelle Bestellung
  const orderRef = db.collection(`clubs/${clubId}/orders`).doc();
  await orderRef.set({
    orderId: orderRef.id,
    userId,
    productId,
    productName: product.name,
    quantity,
    totalPrice: product.price * quantity,
    status: 'open',
    createdAt: Date.now(),
    createdBy: userId
  });

  return { success: true, orderId: orderRef.id };
});
```

Regel 3: Client-seitige UI-Anzeige

Im Club-App Frontend:

```
// Zeige Warnung, wenn User zu jung ist
function ProductCard({ product, user }: { product: Product; user: UserDocument }) {
  const canOrder = canUserOrderProduct(user, product);

  if (!canOrder.allowed) {
    return (
      <div className="product-card disabled">
        <h3>{product.name}</h3>
        <p className="price">{product.price}€</p>
        <p className="age-warning">🚫 {canOrder.reason}</p>
      </div>
    );
  }

  return (
    <div className="product-card">
      <h3>{product.name}</h3>
      <p className="price">{product.price}€</p>
      <button onClick={() => addToCart(product)}>In den Warenkorb</button>
    </div>
  );
}
```

4.4 Jugendschutz-Benachrichtigungen

Konzept: Automatische Sperrstunden-Erinnerung für U18

Clubs können eine **Jugendschutz-Sperrstunde** konfigurieren (z.B. 00:00 Uhr). Das System sendet automatisch eine Notification/Push an alle eingecheckten U18-User.

Notification-Type (NEU)

```
export type NotificationType =
  // ... bestehende Types
  | 'YOUTH_PROTECTION_LEAVE_CLUB'
  | 'YOUTH_PROTECTION_ENTRY_DENIED';
```

Scheduled Cloud Function

```
// functions/src/youth-protection.ts

export const youthProtectionCurfewCheck = functions.pubsub
  .schedule('every 15 minutes')
  .onRun(async () => {
    const now = new Date();
    const currentTime = `${String(now.getHours()).padStart(2, '0')}:${String(now.getMi
nutes()).padStart(2, '0')}`;

    // Hole alle Clubs mit aktivierter Jugendschutz-Sperrstunde
    const clubsSnapshot = await db.collection('platform/clubs')
      .where('settings.youthCurfewEnabled', '==', true)
      .get();

    for (const clubDoc of clubsSnapshot.docs) {
      const club = clubDoc.data() as Club;
      const curfewTime = club.settings?.youthCurfewTime;

      if (!curfewTime) continue;

      // Ist jetzt die Sperrstunde?
      if (currentTime === curfewTime) {
        // Hole alle U18-User, die eingecheckt sind
        const u18UsersSnapshot = await db
          .collection(`clubs/${club.clubId}/users`)
          .where('isInClub', '==', true)
          .where('ageGroup', 'in', ['u16', 'u18'])
          .get();

        // Sende Notifications
        const batch = db.batch();
        for (const userDoc of u18UsersSnapshot.docs) {
          const user = userDoc.data() as UserDocument;

          // 1. Erstelle Notification-Dokument
          const notifRef = db.collection(`clubs/${club.clubId}/notifications`).doc();
          batch.set(notifRef, {
            notificationId: notifRef.id,
            userId: user.uid,
            type: 'YOUTH_PROTECTION_LEAVE_CLUB',
            title: 'Jugendschutz: Bitte verlasse den Club',
            message: `Es ist ${curfewTime} Uhr. Gemäß Jugendschutzgesetz musst du den
Club jetzt verlassen.`,
            createdAt: Date.now(),
            read: false,
            actionUrl: null
          });

          // 2. (Optional) Sende Push-Notification via FCM
          // TODO: FCM-Integration
        }

        await batch.commit();
        console.log(`Sent curfew notifications to ${u18UsersSnapshot.size} U18 users
in club ${club.clubId}`);
      }
    }
  });
```

UI-Integration

In `club-app/src/components/notifications.tsx`:

```
function NotificationItem({ notification }: { notification: Notification }) {
  if (notification.type === 'YOUTH_PROTECTION_LEAVE_CLUB') {
    return (
      <div className="notification youth-protection">
        <div className="icon">🚨</div>
        <div className="content">
          <h4>{notification.title}</h4>
          <p>{notification.message}</p>
          <button onClick={handleCheckOut}>Jetzt auschecken</button>
        </div>
      </div>
    );
  }

  // ... andere Notification-Types
}
```

4.5 Funktionen & Interfaces

TypeScript Interfaces

```
// packages/shared-types/src/product.ts (NEU)

export interface Product {
  /** Produkt-ID */
  productId: string;

  /** Produktname */
  name: string;

  /** Kategorie */
  category: 'beer' | 'spirits' | 'softdrinks' | 'food' | 'other';

  /** Preis in EUR */
  price: number;

  /** Mindestalter für Kauf (null = keine Beschränkung) */
  minAge: number | null;

  /** Ausweis-Prüfung erforderlich? */
  requiresAgeVerification: boolean;

  /** Verfügbar? */
  available: boolean;

  /** Bild-URL (optional) */
  imageUrl?: string | null;

  /** Beschreibung (optional) */
  description?: string | null;
}
```

```typescript
// packages/shared-types/src/notification.ts (ERWEITERT)

export type NotificationType =
  // ... bestehende Types (aus Addendum 1)
  | 'YOUTH_PROTECTION_LEAVE_CLUB'
  | 'YOUTH_PROTECTION_ENTRY_DENIED';

export interface Notification {
  notificationId: string;
  userId: string;
  type: NotificationType;
  title: string;
  message: string;
  createdAt: number;
  read: boolean;
  actionUrl?: string | null;
}
```

Core Functions (Pseudocode)

```
// packages/core/src/utils/order-validation.ts

/**
 * Prüft, ob User ein Produkt bestellen kann
 * @param user - User-Dokument
 * @param product - Produkt-Dokument
 * @returns { allowed: boolean, reason?: string }
 */
export function canUserOrderProduct(
  user: UserDocument,
  product: Product
): { allowed: boolean; reason?: string } {
  // 1. Phone-Verifizierung (Basis-Voraussetzung)
  if (!user.phoneVerified) {
    return {
      allowed: false,
      reason: 'Bitte verifiziere zuerst deine Telefonnummer in den Einstellungen.'
    };
  }

  // 2. Altersbeschränkung?
  if (product.minAge !== null) {
    // Hat User Geburtsdatum?
    if (!user.dateOfBirth) {
      return {
        allowed: false,
        reason: `Dieses Produkt hat eine Altersgrenze (${product.minAge}+). Bitte gib
dein Geburtsdatum in den Einstellungen an.`
      };
    }

    // Ist User alt genug?
    const age = calculateAge(user.dateOfBirth);
    if (age < product.minAge) {
      return {
        allowed: false,
        reason: `Dieses Produkt ist erst ab ${product.minAge} Jahren erhältlich. Du
bist ${age} Jahre alt.`
      };
    }

    // Ist Ausweis-Prüfung erforderlich?
    if (product.requiresAgeVerification && !user.ageVerified) {
      return {
        allowed: false,
        reason: `Für dieses Produkt muss dein Ausweis einmalig am Eingang geprüft
werden. Bitte wende dich an das Personal.`
      };
    }
  }

  // 3. Produkt verfügbar?
  if (!product.available) {
    return {
      allowed: false,
      reason: 'Dieses Produkt ist aktuell nicht verfügbar.'
    };
  }

  return { allowed: true };
}
```

```typescript
/**
 * Prüft, ob User den Trusted-Status erhalten kann
 * (Erweiterte Version von Addendum 1)
 */
export function canSetTrustedLevel(
  user: UserDocument,
  clubSettings: ClubSettings
): { allowed: boolean; reason?: string } {
  // 1. Phone-Verifizierung
  if (!user.phoneVerified) {
    return { allowed: false, reason: 'Telefonnummer muss verifiziert sein' };
  }

  // 2. Alters-Verifizierung (NEU)
  if (!user.ageVerified) {
    return { allowed: false, reason: 'Ausweis muss geprüft werden' };
  }

  // 3. Geburtsdatum vorhanden? (NEU)
  if (!user.dateOfBirth) {
    return { allowed: false, reason: 'Geburtsdatum muss angegeben werden' };
  }

  // 4. Mindestalter (NEU)
  const age = calculateAge(user.dateOfBirth);
  const minAge = clubSettings.minAgeForTrusted || 18;

  if (age < minAge) {
    return {
      allowed: false,
      reason: `Mindestalter für Trusted-Status: ${minAge} Jahre`
    };
  }

  return { allowed: true };
}

/**
 * Setzt Trust-Level auf 'trusted' (erweiterte Version)
 * Wird von Türsteher-App aufgerufen
 */
export async function setTrustedLevel(
  userId: string,
  staffId: string,
  clubId: string,
  dateOfBirth: string // NEU: Geburtsdatum vom Ausweis
): Promise<void> {
  // 1. Berechne Altersgruppe
  const ageGroup = calculateAgeGroup(dateOfBirth);

  // 2. Update in Club-Dokument
  await updateDoc(doc(db, `clubs/${clubId}/users/${userId}`), {
    dateOfBirth,
    ageVerified: true,
    ageGroup,
    trustLevel: 'trusted',
    trustedVerifiedAt: Date.now(),
    trustedVerifiedByClubId: clubId
  });

  // 3. Update in Platform-Dokument (global)
  await updateDoc(doc(db, `platform/users/${userId}`), {
    dateOfBirth,
```

```
    ageVerified: true,
    globalTrustLevel: 'trusted',
    trustedVerifiedAt: Date.now(),
    trustedVerifiedByClubId: clubId,
    trustedVerifiedByStaffId: staffId
  });

  // 4. Cloud Function synchronisiert automatisch in alle Club-Dokumente
  // (siehe Addendum 1, Regel 3)
}
```

4.6 Firestore-Schema

Collection: `clubs/{clubId}/products/{productId}` (NEU)

Produkt-Dokument:

```
{
  "productId": "prod_beer_001",
  "name": "Bier (0,5l)",
  "category": "beer",
  "price": 4.50,
  "minAge": 16,                              // number | null
  "requiresAgeVerification": false,          // boolean
  "available": true,                         // boolean
  "imageUrl": "https://images.pexels.com/photos/29290425/pexels-photo-29290425/free-
photo-of-pouring-a-fresh-pint-of-beer-in-montreal-bar.jpeg?
auto=compress&cs=tinysrgb&dpr=1&w=500",
  "description": "Frisch gezapftes Pils"
}
```

Collection: `clubs/{clubId}/config/settings`

Erweiterte Felder:

```
{
  // ... bestehende Felder (features, theme, openingHours, etc.)

  // NEU: Jugendschutz-Einstellungen
  "youthCurfewTime": "00:00",                // string | null (Format: "HH:MM")
  "youthCurfewEnabled": true,                // boolean
  "minAgeForEntry": 16,                      // number (16, 18, 21)
  "minAgeForTrusted": 18                     // number (für Trust-Level)
}
```

Collection: `clubs/{clubId}/notifications/{notificationId}` (NEU)

Notification-Dokument:

```
{
  "notificationId": "notif_abc123",
  "userId": "user_xyz",
  "type": "YOUTH_PROTECTION_LEAVE_CLUB",
  "title": "Jugendschutz: Bitte verlasse den Club",
  "message": "Es ist 00:00 Uhr. Gemäß Jugendschutzgesetz musst du den Club jetzt ver-
lassen.",
  "createdAt": 1701388800000,
  "read": false,
  "actionUrl": null
}
```

---

5. GEBURTSTAGE & CLUB-SPEZIFISCHE BIRTHDAY-PERKS

Konzept

Das **Birthday-Perk-System** erkennt automatisch, wenn ein User Geburtstag hat, und ermöglicht Clubs, spezielle Vorteile anzubieten:
- 🎁 **Gratis-Getränk**
- 🎤 **DJ-Ansage** (“Happy Birthday an Max!”)
- 🌟 **VIP-Entry** (Fast-Lane)

Warum wichtig?

- ✅ **Marketing:** Geburtstagsgäste bringen oft Freunde mit → mehr Umsatz
- ✅ **Loyalität:** Persönliche Aufmerksamkeit bindet Gäste langfristig
- ✅ **Viral:** “Ich hatte Geburtstag im Club XYZ und bekam einen Gratis-Drink!” → Social Media

---

5.1 Datenmodell-Erweiterungen

Club-Settings: `clubs/{clubId}/config/settings`

Neue Nested-Settings für Birthday-Perks:

```typescript
interface ClubBirthdaySettings {
  /** Birthday-Feature aktiviert? */
  enabled: boolean;

  /** Gratis-Getränk anbieten? */
  freeDrinkEnabled: boolean;

  /** VIP-Entry für Geburtstagskinder? */
  vipEntryEnabled: boolean;

  /** DJ-Ansagen für Geburtstagskinder? */
  djAnnouncementEnabled: boolean;

  /** Produkt-ID für Gratis-Getränk (falls freeDrinkEnabled) */
  freeDrinkProductId?: string | null;

  /** Maximale Anzahl an Gratis-Drinks pro Geburtstag */
  freeDrinkLimit?: number; // Default: 1
}
```

Integration in Club-Settings:

```
{
  // ... bestehende Felder (features, theme, openingHours, etc.)

  // NEU: Birthday-Settings
  "birthday": {
    "enabled": true,
    "freeDrinkEnabled": true,
    "vipEntryEnabled": true,
    "djAnnouncementEnabled": true,
    "freeDrinkProductId": "prod_beer_001",
    "freeDrinkLimit": 1
  }
}
```

User-Club-Day-State: `clubs/{clubId}/userDayStates/{userId}_{date}` (NEU)

Zweck: Tages-spezifischer State für User im Club (z.B. “Hat heute schon Geburtstagsperk eingelöst?”)

Document-ID-Format: `{userId}_{YYYY-MM-DD}`
Beispiel: `user_abc123_2025-12-03`

```typescript
interface UserClubDayState {
  /** User-ID */
  userId: string;

  /** Club-ID */
  clubId: string;

  /** Datum (YYYY-MM-DD) */
  eventDate: string;

  /** Ist heute der Geburtstag des Users? */
  isBirthdayToday: boolean;

  /** Wurde Birthday-Perk bereits eingelöst? */
  birthdayPerkClaimed: boolean;

  /** Timestamp der Perk-Einlösung */
  birthdayPerkClaimedAt?: number | null;

  /** Erstellungs-Timestamp */
  createdAt: number;
}
```

Firestore-Dokument:

```json
{
  "userId": "user_abc123",
  "clubId": "club_xyz",
  "eventDate": "2025-12-03",
  "isBirthdayToday": true,
  "birthdayPerkClaimed": false,
  "birthdayPerkClaimedAt": null,
  "createdAt": 1701388800000
}
```

5.2 Kernregeln & Business Logic

Regel 1: Geburtstags-Erkennung bei Check-In

Wann wird erkannt, dass User Geburtstag hat?
- ✅ Beim **Check-In** (QR-Code-Scan am Eingang)
- ✅ Beim **Öffnen des Club-Screens** in der App
- ✅ Bei **Login** während eines aktiven Events

Ablauf:

```
1. User checkt sich ein (isInClub = true)
   ↓
2. System prüft: Ist heute Geburtstag?
   → Vergleich: dateOfBirth (Tag/Monat) === today (Tag/Monat)
   ↓
3. Falls JA:
   → Erstelle/Update UserClubDayState-Dokument
   → isBirthdayToday = true
   ↓
4. Zeige Birthday-Banner in App:
   "🎉 Happy Birthday! Hol dir dein Gratis-Getränk an der Bar!"
```

Regel 2: Einmaliges Birthday-Perk pro Tag

```
User mit isBirthdayToday = true
  ↓
Zeige "Gratis-Getränk einlösen"-Button
  ↓
User klickt Button
  ↓
System prüft: birthdayPerkClaimed === false?
  ↓
Falls JA:
  → Erstelle automatische Bestellung (Order mit price = 0)
  → Setze birthdayPerkClaimed = true
  → Setze birthdayPerkClaimedAt = Date.now()
  ↓
Falls NEIN:
  → Zeige Nachricht: "Du hast dein Geburtstagsperk bereits eingelöst!"
```

Regel 3: DJ-Ansagen für Geburtstagskinder

DJ-Console-Integration:

Im `dj-console` Dashboard:

```
// DJ-Console-Screen: Geburtstagsliste
function BirthdayGuestList({ clubId }: { clubId: string }) {
  const [birthdayGuests, setBirthdayGuests] = useState<UserDocument[]>([]);

  useEffect(() => {
    // Query: Alle eingecheckten User mit Geburtstag heute
    const today = formatDate(new Date(), 'YYYY-MM-DD'); // "2025-12-03"

    const q = query(
      collection(db, `clubs/${clubId}/userDayStates`),
      where('eventDate', '==', today),
      where('isBirthdayToday', '==', true)
    );

    const unsubscribe = onSnapshot(q, async (snapshot) => {
      const userIds = snapshot.docs.map(doc => doc.data().userId);

      // Hole User-Details
      const users = await Promise.all(
        userIds.map(uid => getDoc(doc(db, `clubs/${clubId}/users/${uid}`)))
      );

      setBirthdayGuests(users.map(doc => doc.data() as UserDocument));
    });

    return unsubscribe;
  }, [clubId]);

  return (
    <div className="birthday-guests">
      <h2>🎂 Geburtstagsgäste heute ({birthdayGuests.length})</h2>
      <ul>
        {birthdayGuests.map(guest => (
          <li key={guest.uid}>
            {guest.displayName} - {calculateAge(guest.dateOfBirth!)} Jahre
            <button onClick={() => announceBirthday(guest.displayName)}>
              📢 Ansagen
            </button>
          </li>
        ))}
      </ul>
    </div>
  );
}

function announceBirthday(name: string) {
  // Zeige Overlay auf allen User-Screens
  updateDoc(doc(db, `clubs/${clubId}/state/global`), {
    mode: 'message',
    messageText: `🎉 Happy Birthday, ${name}! 🎉`,
    messageTarget: 'all'
  });

  // Optional: Audio-Ausgabe über Club-Sound-System
}
```

Regel 4: VIP-Entry für Geburtstagskinder

Türsteher-App-Integration:

Im `staff-door` Check-In-Screen:

```
function CheckInScreen({ scannedUserId }: { scannedUserId: string }) {
  const user = useUserData(scannedUserId);
  const isBirthdayToday = checkIfBirthdayToday(user.dateOfBirth);

  return (
    <div className="checkin-screen">
      <h2>{user.displayName}</h2>

      {isBirthdayToday && (
        <div className="birthday-banner">
          🎉 GEBURTSTAG HEUTE! 🎉
          <p>Alter: {calculateAge(user.dateOfBirth)} Jahre</p>
          <button onClick={() => handleVIPEntry(user.uid)}>
            ⭐ VIP-Einlass gewähren
          </button>
        </div>
      )}

      <button onClick={() => handleCheckIn(user.uid)}>
        ✅ Check-In durchführen
      </button>
    </div>
  );
}
```

5.3 Funktionen & Interfaces

TypeScript Interfaces

```
// packages/shared-types/src/club.ts (ERWEITERT)

export interface ClubBirthdaySettings {
  /** Birthday-Feature aktiviert? */
  enabled: boolean;

  /** Gratis-Getränk anbieten? */
  freeDrinkEnabled: boolean;

  /** VIP-Entry für Geburtstagskinder? */
  vipEntryEnabled: boolean;

  /** DJ-Ansagen für Geburtstagskinder? */
  djAnnouncementEnabled: boolean;

  /** Produkt-ID für Gratis-Getränk */
  freeDrinkProductId?: string | null;

  /** Maximale Anzahl an Gratis-Drinks pro Geburtstag */
  freeDrinkLimit?: number;
}

export interface ClubSettings {
  // ... bestehende Felder

  /** Birthday-Perk-Einstellungen */
  birthday?: ClubBirthdaySettings;
}
```

```ts
// packages/shared-types/src/user-day-state.ts (NEU)

export interface UserClubDayState {
  /** User-ID */
  userId: string;

  /** Club-ID */
  clubId: string;

  /** Datum (YYYY-MM-DD) */
  eventDate: string;

  /** Ist heute der Geburtstag des Users? */
  isBirthdayToday: boolean;

  /** Wurde Birthday-Perk bereits eingelöst? */
  birthdayPerkClaimed: boolean;

  /** Timestamp der Perk-Einlösung */
  birthdayPerkClaimedAt?: number | null;

  /** Erstellungs-Timestamp */
  createdAt: number;
}
```

Core Functions (Pseudocode)

```
// packages/core/src/utils/birthday-system.ts

/**
 * Prüft, ob heute der Geburtstag des Users ist
 * @param dateOfBirth - ISO-String (YYYY-MM-DD)
 * @param today - Aktuelles Datum (für Tests mockbar)
 * @returns true wenn heute Geburtstag
 */
export function isBirthdayToday(dateOfBirth: string, today: Date = new Date()): boolea
n {
  const birthDate = new Date(dateOfBirth);

  return (
    birthDate.getMonth() === today.getMonth() &&
    birthDate.getDate() === today.getDate()
  );
}

/**
 * Erstellt/Updated UserClubDayState beim Check-In
 * @param userId - User-UID
 * @param clubId - Club-ID
 */
export async function createUserDayState(userId: string, clubId: string):
Promise<void> {
  const userDoc = await getDoc(doc(db, `clubs/${clubId}/users/${userId}`));
  const user = userDoc.data() as UserDocument;

  if (!user.dateOfBirth) return; // Kein Geburtsdatum gesetzt

  const today = formatDate(new Date(), 'YYYY-MM-DD'); // "2025-12-03"
  const docId = `${userId}_${today}`;
  const docRef = doc(db, `clubs/${clubId}/userDayStates/${docId}`);

  // Prüfe ob Dokument schon existiert
  const existingDoc = await getDoc(docRef);
  if (existingDoc.exists()) return; // Bereits erstellt

  // Erstelle neues Dokument
  await setDoc(docRef, {
    userId,
    clubId,
    eventDate: today,
    isBirthdayToday: isBirthdayToday(user.dateOfBirth),
    birthdayPerkClaimed: false,
    birthdayPerkClaimedAt: null,
    createdAt: Date.now()
  });
}

/**
 * Löst Birthday-Perk ein (Gratis-Getränk)
 * @param userId - User-UID
 * @param clubId - Club-ID
 * @returns { success: boolean, message: string }
 */
export async function claimBirthdayPerk(
  userId: string,
  clubId: string
): Promise<{ success: boolean; message: string }> {
  const today = formatDate(new Date(), 'YYYY-MM-DD');
  const docId = `${userId}_${today}`;
```

```
  const docRef = doc(db, `clubs/${clubId}/userDayStates/${docId}`);

  // 1. Prüfe UserDayState
  const dayStateDoc = await getDoc(docRef);
  if (!dayStateDoc.exists()) {
    return { success: false, message: 'Kein Geburtstags-State gefunden' };
  }

  const dayState = dayStateDoc.data() as UserClubDayState;

  // 2. Ist heute überhaupt Geburtstag?
  if (!dayState.isBirthdayToday) {
    return { success: false, message: 'Heute ist nicht dein Geburtstag' };
  }

  // 3. Wurde Perk bereits eingelöst?
  if (dayState.birthdayPerkClaimed) {
    return { success: false, message: 'Du hast dein Geburtstagsperk bereits ein-
gelöst!' };
  }

  // 4. Hole Club-Settings
  const clubDoc = await getDoc(doc(db, `platform/clubs/${clubId}`));
  const club = clubDoc.data() as Club;
  const birthdaySettings = club.settings?.birthday;

  if (!birthdaySettings?.enabled || !birthdaySettings.freeDrinkEnabled) {
    return { success: false, message: 'Birthday-Perks sind in diesem Club nicht akt-
iviert' };
  }

  // 5. Erstelle automatische Bestellung (Gratis-Getränk)
  const productId = birthdaySettings.freeDrinkProductId;
  if (!productId) {
    return { success: false, message: 'Kein Gratis-Getränk konfiguriert' };
  }

  const productDoc = await getDoc(doc(db, `clubs/${clubId}/products/${productId}`));
  const product = productDoc.data() as Product;

  const orderRef = doc(collection(db, `clubs/${clubId}/orders`));
  await setDoc(orderRef, {
    orderId: orderRef.id,
    userId,
    productId,
    productName: product.name,
    quantity: 1,
    totalPrice: 0, // GRATIS!
    status: 'open',
    isBirthdayPerk: true, // Spezial-Flag
    createdAt: Date.now(),
    createdBy: userId
  });

  // 6. Markiere Perk als eingelöst
  await updateDoc(docRef, {
    birthdayPerkClaimed: true,
    birthdayPerkClaimedAt: Date.now()
  });

  return {
    success: true,
    message: `Dein Gratis-${product.name} wartet an der Bar! 🎉`
```

```
  };
}

/**
 * Holt alle Geburtstagsgäste, die heute im Club sind
 * (Für DJ-Console)
 * @param clubId - Club-ID
 * @returns Liste von Usern mit Geburtstag heute
 */
export async function getTodaysBirthdayGuests(clubId: string):
Promise<UserDocument[]> {
  const today = formatDate(new Date(), 'YYYY-MM-DD');

  // Query: Alle UserDayStates mit Geburtstag heute
  const snapshot = await getDocs(
    query(
      collection(db, `clubs/${clubId}/userDayStates`),
      where('eventDate', '==', today),
      where('isBirthdayToday', '==', true)
    )
  );

  // Hole User-Details
  const userIds = snapshot.docs.map(doc => doc.data().userId);
  const users = await Promise.all(
    userIds.map(uid => getDoc(doc(db, `clubs/${clubId}/users/${uid}`)))
  );

  return users
    .filter(doc => doc.exists())
    .map(doc => doc.data() as UserDocument)
    .filter(user => user.isInClub); // Nur eingecheckte User
}
```

5.4 Firestore-Schema

Collection: `clubs/{clubId}/config/settings`

Erweiterte Felder:

```
{
  // ... bestehende Felder

  // NEU: Birthday-Settings
  "birthday": {
    "enabled": true,
    "freeDrinkEnabled": true,
    "vipEntryEnabled": true,
    "djAnnouncementEnabled": true,
    "freeDrinkProductId": "prod_beer_001",
    "freeDrinkLimit": 1
  }
}
```

Collection: `clubs/{clubId}/userDayStates/{userId}_{date}` (NEU)

Document-ID-Format: `{userId}_{YYYY-MM-DD}`

Dokument:

```
{
  "userId": "user_abc123",
  "clubId": "club_xyz",
  "eventDate": "2025-12-03",
  "isBirthdayToday": true,
  "birthdayPerkClaimed": false,
  "birthdayPerkClaimedAt": null,
  "createdAt": 1701388800000
}
```

Firestore Queries

Query 1: Alle Geburtstagsgäste heute

```
const today = formatDate(new Date(), 'YYYY-MM-DD');
const birthdayGuests = await getDocs(
  query(
    collection(db, `clubs/${clubId}/userDayStates`),
    where('eventDate', '==', today),
    where('isBirthdayToday', '==', true)
  )
);
```

Benötigte Composite Indexes:

```
clubs/{clubId}/userDayStates
  - eventDate (ASC) + isBirthdayToday (ASC)
```

---

6. ZUSAMMENFASSUNG & VERKNÜPFUNG

Dieses Addendum führt drei neue Systeme ein, die nahtlos miteinander und mit den bestehenden Features (aus ARCHITECTURE.md und Addendum 1) zusammenarbeiten:

Wie alles zusammenspielt

1. Alterslogik → Trusted User

```
User gibt Geburtsdatum an (dateOfBirth)
  ↓
System berechnet Alter & ageGroup automatisch
  ↓
Türsteher prüft Ausweis → ageVerified = true
  ↓
Phone verifiziert + Alter >= 18 + ageVerified
  ↓
trustLevel = 'trusted' (kann jetzt unbegrenzt bestellen)
```

2. Alterslogik → Produkt-Bestellungen

```
User will Wodka bestellen (minAge = 18, requiresAgeVerification = true)
  ↓
System prüft:
  - dateOfBirth vorhanden? ✅
  - Alter >= 18? ✅
  - ageVerified = true? ✅
  ↓
Bestellung erlaubt
```

3. Alterslogik → Jugendschutz-Notifications

```
User ist U18 und im Club eingecheckt
  ↓
Uhr schlägt 00:00 (youthCurfewTime)
  ↓
Cloud Function sendet Notification:
  "Bitte verlasse den Club (Jugendschutzgesetz)"
  ↓
User checkt sich aus
```

4. Geburtsdatum → Birthday-Perks

```
User checkt sich ein
  ↓
System erkennt: Heute ist Geburtstag! (dateOfBirth)
  ↓
Erstellt UserClubDayState mit isBirthdayToday = true
  ↓
App zeigt: "🎉 Hol dir dein Gratis-Getränk!"
  ↓
User löst Perk ein → birthdayPerkClaimed = true
  ↓
DJ sieht User in "Geburtstagsliste"
  ↓
DJ macht Ansage: "Happy Birthday an Max!"
```

5. Party-Modus (Addendum 1) → Alters-basierte Features

```
User checkt sich ein (QR-Code, KEIN GPS!)
  ↓
isInClub = true
  ↓
User aktiviert Party-Modus (wählt Zahl)
  ↓
partyModeActive = true
  ↓
DJ startet Gewinnspiel → findet User mit selectedNumber
  ↓
Gewinn: Gratis-Getränk
  ↓
System prüft Alter vor Ausgabe (falls Alkohol)
```

6. Trust-Level → Bestellungen → Alters-Check

```
User ist Trusted (phoneVerified + ageVerified + Alter >= 18)
  ↓
Kann unbegrenzt bestellen (kein Limit)
  ↓
Bei jeder Bestellung: Serverseitige Altersprüfung
  ↓
Falls Produkt.minAge > User-Alter → Bestellung abgelehnt
```

Gesamtarchitektur: Alle Systeme im Überblick

```
┌─────────────────────────────────────────────────────────────┐
│                    NIGHTLIFE OS                             │
│                   (Phase 1-10)                              │
└─────────────────────────────────────────────────────────────┘
           │
           ├─── [Phase 1-8] Basis-System (ARCHITECTURE.md)
           │     ├── User-Auth & Roles
           │     ├── Chat-System
           │     ├── Freundschafts-System
           │     ├── Lichtshow & DJ-Console
           │     └── Club-Management
           │
           ├─── [Phase 9] Interactive Status (Addendum 1)
           │     ├── Party-Modus (6 Zahlen, 30min Timer)
           │     ├── Dating-Status (6h-Regel)
           │     ├── Trust-System (Phone-Verifizierung)
           │     ├── Club-Pläne (Free vs. Premium)
           │     └── Event-Lifecycle
           │
           └─── [Phase 10] Age & Birthday (Addendum 2)
                 ├── Alterslogik (dateOfBirth → ageGroup)
                 ├── Automatisches Level-Up (16→18→21)
                 ├── Erweiterter Trust (Phone + Age + Verifizierung)
                 ├── Produkt-Altersgrenzen
                 ├── Jugendschutz-Notifications
                 └── Birthday-Perks (Gratis-Drinks, DJ-Ansagen)
```

Datenfluss: Von Check-In bis Geburtstagsperk

```
1. USER BETRITT CLUB
   ├─ Türsteher scannt QR-Code (KEIN GPS!)
   ├─ isInClub = true
   └─ currentClubId = clubId

2. ALTERS-CHECK
   ├─ System berechnet Alter aus dateOfBirth
   ├─ ageGroup wird aktualisiert (falls nötig)
   └─ Falls U18 + youthCurfewEnabled → Warnung anzeigen

3. GEBURTSTAGS-CHECK
   ├─ System prüft: Ist heute Geburtstag?
   ├─ Falls JA: UserClubDayState erstellen
   └─ Birthday-Banner in App anzeigen

4. PARTY-MODUS
   ├─ User wählt eine der 6 Zahlen
   ├─ partyModeActive = true
   └─ Kann an DJ-Aktionen teilnehmen

5. BESTELLUNG
   ├─ User bestellt Getränk
   ├─ System prüft:
   │  ├─ phoneVerified?
   │  ├─ Alter >= product.minAge?
   │  └─ Falls requiresAgeVerification: ageVerified?
   └─ Bestellung wird freigegeben/abgelehnt

6. BIRTHDAY-PERK
   ├─ User klickt "Gratis-Getränk einlösen"
   ├─ System prüft: birthdayPerkClaimed?
   ├─ Falls NEIN: Erstelle Order mit price = 0
   └─ birthdayPerkClaimed = true

7. DJ-ANSAGE
   ├─ DJ öffnet "Geburtstagsliste"
   ├─ Sieht alle User mit isBirthdayToday = true
   └─ Macht Ansage über Mikrofon
```

---

7. SECURITY RULES-ERWEITERUNGEN

Die folgenden Security Rules erweitern die bestehenden Rules aus ARCHITECTURE.md und Addendum 1:

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

    // ===== HELPER FUNCTIONS (aus ARCHITECTURE.md + Addendum 1) =====
    // ... (bestehende Funktionen bleiben)

    // NEU: Alters-Helpers
    function calculateAge(dateOfBirth) {
      // Vereinfachte Logik (exakte Berechnung im Client/Cloud Function)
      let birthYear = int(dateOfBirth.split('-')[0]);
      let currentYear = request.time.year();
      return currentYear - birthYear;
    }

    // ===== PLATFORM-EBENE =====

    match /platform/users/{uid} {
      // User kann sein eigenes Dokument lesen/updaten
      // ABER: dateOfBirth und ageVerified NICHT selbst ändern
      allow update: if request.auth.uid == uid &&
        !request.resource.data.diff(resource.data).affectedKeys().hasAny([
          'dateOfBirth',
          'ageVerified',
          'globalTrustLevel',
          'isPlatformAdmin',
          'ownedClubs'
        ]);
    }

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

    match /clubs/{clubId}/users/{uid} {
      // User kann NICHT selbst dateOfBirth, ageVerified, ageGroup ändern
      allow update: if request.auth.uid == uid &&
        !request.resource.data.diff(resource.data).affectedKeys().hasAny([
          'dateOfBirth',
          'ageVerified',
          'ageGroup',
          'trustLevel'
        ]);

      // Staff/Admin kann dateOfBirth, ageVerified, ageGroup setzen
      allow update: if hasAnyRole(clubId, ['admin', 'staff', 'door']) &&
        request.resource.data.diff(resource.data).affectedKeys().hasOnly([
          'dateOfBirth',
          'ageVerified',
          'ageGroup',
          'trustLevel',
          'trustedVerifiedAt',
          'trustedVerifiedByClubId'
        ]);
    }

    match /clubs/{clubId}/products/{productId} {
      // Alle Club-Mitglieder können Produkte lesen
      allow read: if isClubMember(clubId);

      // Nur Admins/Staff können Produkte verwalten
      allow write: if hasAnyRole(clubId, ['admin', 'staff', 'bar', 'waiter']);
    }
```

```
    match /clubs/{clubId}/userDayStates/{dayStateId} {
      // User kann nur seinen eigenen DayState lesen
      allow read: if isAuthenticated() &&
        resource.data.userId == request.auth.uid;

      // System/Cloud Function kann DayStates erstellen/updaten
      // (via Admin SDK, keine explizite Rule nötig)

      // Staff/DJ kann alle DayStates lesen (für Geburtstagsliste)
      allow read: if hasAnyRole(clubId, ['admin', 'dj', 'staff']);
    }

    match /clubs/{clubId}/notifications/{notificationId} {
      // User kann nur seine eigenen Notifications lesen/updaten
      allow read, update: if isAuthenticated() &&
        resource.data.userId == request.auth.uid;

      // System/Cloud Function erstellt Notifications (via Admin SDK)
    }

    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');
    }
  }
}
```

---

8. INTEGRATION MIT BESTEHENDEN SYSTEMEN

8.1 Kompatibilität mit Phase 1-8 (ARCHITECTURE.md)

Bestehende Collection	Änderungen	Backward-Compatible?
`platform/users/{uid}`	+2 Felder (`dateOfBirth` , `ageVerified`)	✅ Ja (optional)
`clubs/{clubId}/users/{uid}`	+3 Felder (`dateOfBirth` , `ageVerified` , `ageGroup`)	✅ Ja (optional)
`clubs/{clubId}/config/settings`	+2 Nested Objects (`birthday` , `youthCurfew*`)	✅ Ja (optional)
`clubs/{clubId}/orders/{orderId}`	Keine Änderungen (Produkt-Check ist neu)	✅ Ja

Fazit: Alle Erweiterungen sind **optional** und brechen bestehende Funktionalität nicht.

8.2 Kompatibilität mit Phase 9 (Addendum 1)

Feature (Addendum 1)	Änderungen durch Addendum 2	Kompatibilität
Party-Modus	Keine Änderungen	✅ 100% kompatibel
Dating-Status	Keine Änderungen	✅ 100% kompatibel
Trust-System	Erweitert um `ageVerified` + Alters-Check	⚠️ Erweitert (nicht Breaking)
Club-Pläne	Keine Änderungen	✅ 100% kompatibel
Event-Lifecycle	Birthday-Perks nutzen Event-Context	✅ Ergänzt (nicht Breaking)

Fazit: Addendum 2 **erweitert** Addendum 1, bricht aber nichts.

8.3 Migrations-Checkliste

Wenn ein bestehendes Nightlife OS-System von Phase 9 auf Phase 10 upgraded wird:

Schritt 1: Firestore-Schema erweitern

- [ ] Neue Felder in `platform/users` hinzufügen (optional, erst bei Verwendung)
- [ ] Neue Felder in `clubs/{clubId}/users` hinzufügen (optional)
- [ ] Neue Collection `clubs/{clubId}/products` erstellen (falls Bestellsystem genutzt)
- [ ] Neue Collection `clubs/{clubId}/userDayStates` erstellen (für Birthday-Perks)

Schritt 2: Security Rules updaten

- [ ] Neue Rules für `dateOfBirth`, `ageVerified`, `ageGroup` hinzufügen
- [ ] Rules für `products` Collection hinzufügen
- [ ] Rules für `userDayStates` Collection hinzufügen

Schritt 3: Cloud Functions deployen

- [ ] `updateAgeGroupIfNeeded` (Option A: Lazy) ODER `updateAgeGroups` (Option B: Cron)
- [ ] `validateOrder` (für Bestellungs-Validierung)
- [ ] `youthProtectionCurfewCheck` (für Jugendschutz-Notifications)
- [ ] `syncTrustLevel` (für globale Trust-Level-Synchronisation, aus Addendum 1)

Schritt 4: UI-Komponenten erweitern

- [ ] Birthday-Banner in `club-app`
- [ ] Geburtstagsliste in `dj-console`
- [ ] Ausweis-Prüfung in `staff-door`
- [ ] Produkt-Altersgrenzen-Anzeige in `club-app` (Bestellsystem)

Schritt 5: Testing

- [ ] Test: User wird 18 → ageGroup updated automatisch

- [ ] Test: U18 User erhält Jugendschutz-Notification um 00:00
- [ ] Test: User mit Geburtstag erhält Birthday-Perk
- [ ] Test: Bestellung von Alkohol wird abgelehnt für U18
- [ ] Test: Trusted-Level erfordert ageVerified + Alter >= 18

---

9. NÄCHSTE SCHRITTE (PHASE 11+)

Die in diesem Addendum eingeführten Systeme legen das Fundament für zukünftige Features:

9.1 Vorgeschlagene Features (basierend auf Alterslogik)

1. Alters-basierte Event-Segmentierung

- **Konzept:** Events mit Altersbeschränkung (z.B. “U21 Party”, “Ü30 Night”)
- **Datenmodell:** `Event.minAge`, `Event.maxAge`
- **Logik:** User sieht nur Events, für die er alt genug ist

2. Dynamische Produkt-Anzeige nach Alter

- **Konzept:** Cocktail-Karte passt sich an Alter an (U18 sieht nur Softdrinks)
- **UI:** Filter in Produkt-Liste basierend auf `ageGroup`

3. Geburtstags-Kampagnen

- **Konzept:** 1 Woche vor Geburtstag: “Feiere bei uns! 20% Rabatt”
- **Datenmodell:** Scheduled Notifications via Cloud Function

4. Alters-basierte Analytics

- **Konzept:** Club-Admin sieht “Durchschnittsalter der Gäste heute”
- **Queries:** Aggregation von `ageGroup`-Verteilung

9.2 Vorgeschlagene Features (basierend auf Birthday-System)

1. Birthday-Countdown in App

- **Konzept:** “Noch 30 Tage bis zu deinem Geburtstag! 🎉”
- **UI:** Widget in Home-View

2. Birthday-Loyalty-Punkte

- **Konzept:** Geburtstagsgäste bekommen doppelte Loyalty-Punkte
- **Integration:** Mit späterem Loyalty-System (Phase 12+)

3. Group-Birthday-Bookings

- **Konzept:** “Feier mit 10 Freunden, einer trinkt gratis”
- **Datenmodell:** `Event.groupBirthdayPackages`

Ende des Addendums

Dieses Dokument beschreibt ausschließlich **Datenmodelle, Regellogik und Interfaces**. Die tatsächliche Implementierung erfolgt in späteren Phasen durch Code-Generierung und UI-Entwicklung.