Nightlife OS - Core Features Implementierung

📊 Übersicht

Diese Dokumentation beschreibt die Implementierung und Konsolidierung der Core-Features im Nightlife OS Monorepo.

Datum: 3. Dezember 2024
Status: ✅ Abgeschlossen

🎯 Implementierte Features

1. Auth-System (Firebase Authentication)

✅ Implementierte Funktionen in `packages/core/src/firebase/auth.ts`

- **signUp / signUpWithEmailAndPassword**
- Registrierung mit E-Mail und Passwort
- Automatische Erstellung von PlatformUser-Dokument
- Initiale Rolle: `visitor`
- Automatische Friend-Code-Generierung
- Push-Benachrichtigungen standardmäßig aktiviert
- **signIn / signInWithEmailAndPassword**
- Login mit E-Mail und Passwort
- Robuste Fehlerbehandlung
- **signOut / logout**
- Benutzer-Logout
- **resetPassword ✨ NEU**
- Sendet Passwort-Zurücksetzen-E-Mail
- Fehlerbehandlung für häufige Fehler (user-not-found, invalid-email)
- **onAuthStateChanged / onAuthStateChangedListener**
- Real-time Auth-State-Listener
- **getCurrentUser**
- Gibt aktuell eingeloggten User zurück

📝 Wichtiger Code-Snippet

```
/**
 * Registrierung mit E-Mail, Passwort und Display Name
 * Erstellt automatisch ein PlatformUser-Dokument mit initialer Rolle 'visitor' und
Friend-Code
 */
export async function signUpWithEmailAndPassword(
  email: string,
  password: string,
  displayName?: string
): Promise<UserCredential> {
  const auth = getAuthInstance();
  const userCredential = await createUserWithEmailAndPassword(auth, email, password);

  // Display Name setzen, falls angegeben
  if (displayName && userCredential.user) {
    await updateProfile(userCredential.user, { displayName });
  }

  // PlatformUser-Dokument erstellen mit initialer Rolle und Friend-Code
  if (userCredential.user) {
    await setDocument(`platform/users/${userCredential.user.uid}`, {
      uid: userCredential.user.uid,
      email: userCredential.user.email,
      displayName: displayName || null,
      photoURL: null,
      friendCode: generateFriendCode(), // Automatisch Friend-Code generieren
      createdAt: Date.now(),
      lastSeenAt: Date.now(),
      isPlatformAdmin: false,
      roles: ['visitor'], // Initiale Rolle: visitor
      clubs: [],
      ownedClubs: [],
      memberClubs: [],
      pushEnabled: true, // Push-Benachrichtigungen standardmäßig aktiviert
    });
  }

  return userCredential;
}
```

---

2. User-Profile (Firestore Integration)

✅ Neue Datei: `packages/core/src/user-profile.ts`

Vollständige CRUD-Funktionen für PlatformUser und ClubUser:

- **createUserProfile(userId, email, displayName, role)**
- Erstellt neues PlatformUser-Profil
- Automatische Friend-Code-Generierung
- Initiale Rolle konfigurierbar (Standard: `visitor`)
- **getUserProfile(userId)**
- Ruft PlatformUser-Profil ab
- **updateUserProfile(userId, data)**

- Aktualisiert PlatformUser-Profil
- Automatisches Update von `lastSeenAt`
- **deleteUserProfile(userId)**
- Löscht PlatformUser-Profil
- **createClubUserProfile(clubId, userId, platformUser?)**
- Erstellt ClubUser-Profil für einen Club
- **getClubUserProfile(clubId, userId)**
- Ruft ClubUser-Profil ab
- **updateClubUserProfile(clubId, userId, data)**
- Aktualisiert ClubUser-Profil

📝 Wichtiger Code-Snippet

```
/**
 * Erstellt ein neues PlatformUser-Profil
 */
export async function createUserProfile(
  userId: string,
  email: string,
  displayName?: string,
  role: 'visitor' | 'staff' | 'club_admin' | 'super_admin' = 'visitor'
): Promise<PlatformUser> {
  const newProfile: PlatformUser = {
    uid: userId,
    email,
    displayName: displayName || null,
    photoURL: null,
    friendCode: generateFriendCode(), // Automatisch Friend-Code generieren
    createdAt: Date.now(),
    lastSeenAt: Date.now(),
    isPlatformAdmin: role === 'super_admin',
    roles: [role],
    clubs: [],
    ownedClubs: [],
    memberClubs: [],
    pushEnabled: true,
  };

  try {
    await setDocument(`platform/users/${userId}`, newProfile);
    return newProfile;
  } catch (error) {
    console.error('Error creating user profile:', error);
    throw new Error('Fehler beim Erstellen des Benutzerprofils.');
  }
}
```

---

3. Friend-Code System

✅ Erweiterte Funktionen in `packages/core/src/utils/friend-code.ts`

- **generateFriendCode()**
- Generiert zufälligen 7-stelligen alphanumerischen Code
- Ohne Verwechslungsgefahr (keine 0, O, I, 1)
- **validateFriendCode(code)**
- Validiert Friend-Code-Format
- **formatFriendCode(code)**
- Formatiert Code (Großbuchstaben)
- **generateUniqueFriendCode(maxAttempts) ✨ NEU**
- Generiert eindeutigen Friend-Code
- Prüft in Firestore auf Eindeutigkeit
- Bis zu 10 Versuche (konfigurierbar)
- **findUserByFriendCode(friendCode) ✨ NEU**
- Findet User anhand des Friend-Codes
- Firestore-Query mit Format-Validierung

📝 Wichtiger Code-Snippet

```
/**
 * Generiert einen eindeutigen Friend-Code
 * Prüft in Firestore, ob der Code bereits existiert
 */
export async function generateUniqueFriendCode(maxAttempts: number = 10): Promise<stri
ng> {
  let attempts = 0;

  while (attempts < maxAttempts) {
    const code = generateFriendCode();

    // Prüfe, ob der Code bereits existiert
    const existingUsers = await getCollection<PlatformUser>(
      'platform/users',
      [where('friendCode', '==', code)]
    );

    if (!existingUsers || existingUsers.length === 0) {
      // Code ist eindeutig
      return code;
    }

    attempts++;
  }

  throw new Error('Konnte keinen eindeutigen Friend-Code generieren. Bitte versuchen
Sie es später erneut.');
}

/**
 * Findet einen User anhand des Friend-Codes
 */
export async function findUserByFriendCode(friendCode: string): Promise<PlatformUser
| null> {
  try {
    const formattedCode = formatFriendCode(friendCode);

    // Validiere den Friend-Code Format
    if (!validateFriendCode(formattedCode)) {
      throw new Error('Ungültiges Friend-Code-Format.');
    }

    // Suche in Firestore
    const users = await getCollection<PlatformUser>(
      'platform/users',
      [where('friendCode', '==', formattedCode)]
    );

    if (users && users.length > 0) {
      return users[0];
    }

    return null;
  } catch (error) {
    console.error('Error finding user by friend code:', error);
    throw new Error('Fehler beim Suchen des Benutzers.');
  }
}
```

4. Basic Check-In Struktur

✅ Neue Datei: `packages/core/src/check-in.ts`

Vollständige CRUD-Funktionen für Check-In/Check-Out:

- **createCheckIn(userId, clubId, via)**
- Erstellt neuen Check-In
- Automatisches Setzen von `checkedInAt` timestamp
- Aktualisiert Club-User-Status
- **getCheckInsByUser(userId, clubId?, limitCount?)**
- Ruft alle Check-Ins eines Users ab
- Optional nach Club filtern
- **updateCheckInStatus(clubId, checkInId, status)**
- Aktualisiert Check-In-Status (Check-Out)
- Setzt `checkedOutAt` timestamp
- **getCurrentCheckIn(userId, clubId)**
- Holt aktuellen aktiven Check-In
- **checkOutUser(userId, clubId)**
- Check-Out für einen User

📝 Wichtiger Code-Snippet

```
/**
 * Erstellt einen neuen Check-In
 */
export async function createCheckIn(
  userId: string,
  clubId: string,
  via: 'manual' | 'qr' | 'nfc' | 'auto' = 'manual'
): Promise<CheckInRecord> {
  const recordId = `${userId}_${Date.now()}`;
  const newCheckIn: CheckInRecord = {
    id: recordId,
    userId,
    clubId,
    checkedInAt: Date.now(), // Automatisches Setzen von timestamp
    checkedOutAt: null,
    via,
  };

  try {
    await setDocument(`clubs/${clubId}/checkins/${recordId}`, newCheckIn);

    // Optional: Club-User-Status aktualisieren
    await updateDocument(`clubs/${clubId}/users/${userId}`, {
      checkedIn: true,
      lastSeenAt: Date.now(),
    }).catch(err => {
      console.warn('Could not update club user status:', err);
    });

    return newCheckIn;
  } catch (error) {
    console.error('Error creating check-in:', error);
    throw new Error('Fehler beim Erstellen des Check-Ins.');
  }
}
```

🎨 UI-Komponenten

Club-App (User-Facing)

1. Login-Seite: `/auth/login`

- **Pfad:** `apps/club-app/src/app/auth/login/page.tsx`
- **Status:** ✅ Bereits vorhanden, erweitert
- **Features:**
- E-Mail & Passwort Login
- Link zu Signup
- **NEU:** Link zu Reset-Password

2. Signup-Seite: `/auth/signup`

- **Pfad:** `apps/club-app/src/app/auth/signup/page.tsx`
- **Status:** ✅ Bereits vorhanden
- **Features:**

- E-Mail, Passwort & Display Name Registrierung
- Automatische Friend-Code-Generierung
- Link zu Login

3. Reset-Password-Seite: `/auth/reset-password` ✨ NEU

- **Pfad:** `apps/club-app/src/app/auth/reset-password/page.tsx`
- **Status:** ✅ Neu erstellt
- **Features:**
- E-Mail-Eingabe
- Sendet Passwort-Zurücksetzen-E-Mail
- Erfolgs-/Fehlermeldungen
- Zurück zu Login

4. Profil-Seite: `/auth/profile`

- **Pfad:** `apps/club-app/src/app/auth/profile/page.tsx`
- **Status:** ✅ Bereits vorhanden, stark erweitert
- **Features:**
- Anzeige von E-Mail, Display Name, Friend-Code
- **NEU:** Display Name bearbeiten (In-Place-Editing)
- Friend-Code generieren (falls nicht vorhanden)
- **NEU:** Friend-Code-Suche mit Ergebnis-Anzeige
- Logout-Funktion

Club-Admin (Admin-Facing)

5. Check-In-Seite: `/checkin` ✨ NEU

- **Pfad:** `apps/club-admin/src/app/checkin/page.tsx`
- **Status:** ✅ Neu erstellt
- **Features:**
- Club-Auswahl (Platzhalter)
- Benutzer-Suche via Friend-Code
- Anzeige von Benutzer-Informationen
- Check-In-Status-Anzeige
- Check-In / Check-Out Buttons
- **Platzhalter:** Cloakroom-Funktionalität
- Erfolgs-/Fehlermeldungen

📁 Liste der erstellten/geänderten Dateien

Packages - Core

✅ **GEÄNDERT:**
- `packages/core/src/firebase/auth.ts`
- Hinzugefügt: `resetPassword()` Funktion
- Erweitert: `signUpWithEmailAndPassword()` mit automatischer Friend-Code-Generierung und Rollen-Zuweisung

- `packages/core/src/index.ts`

- Hinzugefügt: Export von `user-profile` und `check-in`

✨ **NEU ERSTELLT:**
- `packages/core/src/user-profile.ts`
- CRUD-Funktionen für PlatformUser und ClubUser

- `packages/core/src/check-in.ts`
- CRUD-Funktionen für Check-In/Check-Out

✅ **ERWEITERT:**
- `packages/core/src/utils/friend-code.ts`
- Hinzugefügt: `generateUniqueFriendCode()`, `findUserByFriendCode()`

Apps - Club-App (User)

✅ **GEÄNDERT:**
- `apps/club-app/src/app/auth/login/page.tsx`
- Hinzugefügt: Link zu Reset-Password

- `apps/club-app/src/app/auth/profile/page.tsx`
- Hinzugefügt: Display Name bearbeiten
- Hinzugefügt: Friend-Code-Suche UI

✨ **NEU ERSTELLT:**
- `apps/club-app/src/app/auth/reset-password/page.tsx`
- Passwort-Zurücksetzen-Seite

Apps - Club-Admin

✨ **NEU ERSTELLT:**
- `apps/club-admin/src/app/checkin/page.tsx`
- Check-In/Check-Out Management UI

Dokumentation

✨ **NEU ERSTELLT:**
- `CORE_FEATURES_IMPLEMENTATION.md` (diese Datei)
- Vollständige Dokumentation der Implementierung

🔧 Firestore-Schema

Platform-Level Collections

`platform/users/{uid}`

```
{
  uid: string;
  email?: string;
  displayName?: string;
  photoURL?: string;
  friendCode?: string;              // 7-stelliger alphanumerischer Code
  createdAt: number;
  lastSeenAt?: number;
  isPlatformAdmin?: boolean;
  language?: string;
  roles: PlatformRole[];            // ['visitor', 'staff', 'club_admin', 'su-
per_admin']
  clubs?: string[];
  ownedClubs?: string[];
  memberClubs?: string[];
  fcmTokens?: string[];
  pushEnabled?: boolean;            // Standard: true
}
```

Club-Level Collections

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

```
{
  uid: string;
  email?: string;
  displayName?: string;
  photoURL?: string;
  friendCode?: string;
  createdAt: number;
  lastSeenAt?: number;
  checkedIn?: boolean;
  roles?: string[];
  visitCount?: number;
  phoneVerified?: boolean;
  deviceIdHash?: string;
  faceHash?: string;
  staffVerified?: boolean;
}
```

`clubs/{clubId}/checkins/{checkInId}`

```
{
  id: string;                       // z.B. userId_timestamp
  userId: string;
  clubId: string;
  checkedInAt: number;              // Unix-Timestamp (ms)
  checkedOutAt: number | null;      // Unix-Timestamp (ms) oder null
  via: 'manual' | 'qr' | 'nfc' | 'auto';
}
```

✨ Wichtigste Code-Snippets

1. Auth - Reset Password

```
export async function resetPassword(email: string): Promise<void> {
  const auth = getAuthInstance();
  try {
    await firebaseSendPasswordResetEmail(auth, email);
  } catch (error: any) {
    // Fehlerbehandlung für häufige Fehler
    if (error.code === 'auth/user-not-found') {
      throw new Error('Kein Benutzer mit dieser E-Mail-Adresse gefunden.');
    } else if (error.code === 'auth/invalid-email') {
      throw new Error('Ungültige E-Mail-Adresse.');
    } else {
      throw new Error('Fehler beim Senden der Passwort-Zurücksetzen-E-Mail.');
    }
  }
}
```

2. User-Profile - Update

```
export async function updateUserProfile(
  userId: string,
  data: Partial<PlatformUser>
): Promise<void> {
  try {
    // Aktualisiere lastSeenAt automatisch
    const updateData = {
      ...data,
      lastSeenAt: Date.now(),
    };

    await updateDocument(`platform/users/${userId}`, updateData);
  } catch (error) {
    console.error('Error updating user profile:', error);
    throw new Error('Fehler beim Aktualisieren des Benutzerprofils.');
  }
}
```

3. Friend-Code - Suche

```
export async function findUserByFriendCode(friendCode: string): Promise<PlatformUser
| null> {
  try {
    const formattedCode = formatFriendCode(friendCode);

    if (!validateFriendCode(formattedCode)) {
      throw new Error('Ungültiges Friend-Code-Format.');
    }

    const users = await getCollection<PlatformUser>(
      'platform/users',
      [where('friendCode', '==', formattedCode)]
    );

    if (users && users.length > 0) {
      return users[0];
    }

    return null;
  } catch (error) {
    console.error('Error finding user by friend code:', error);
    throw new Error('Fehler beim Suchen des Benutzers.');
  }
}
```

4. Check-In - Erstellen

```
export async function createCheckIn(
  userId: string,
  clubId: string,
  via: 'manual' | 'qr' | 'nfc' | 'auto' = 'manual'
): Promise<CheckInRecord> {
  const recordId = `${userId}_${Date.now()}`;
  const newCheckIn: CheckInRecord = {
    id: recordId,
    userId,
    clubId,
    checkedInAt: Date.now(),
    checkedOutAt: null,
    via,
  };

  await setDocument(`clubs/${clubId}/checkins/${recordId}`, newCheckIn);

  await updateDocument(`clubs/${clubId}/users/${userId}`, {
    checkedIn: true,
    lastSeenAt: Date.now(),
  }).catch(err => {
    console.warn('Could not update club user status:', err);
  });

  return newCheckIn;
}
```

🧪 Testing-Hinweise

Auth-System testen

1. **Signup:**
   - Navigate zu `/auth/signup`
   - Registriere neuen User mit E-Mail, Passwort, Display Name
   - Prüfe, ob Friend-Code automatisch generiert wurde
   - Prüfe Firestore: `platform/users/{uid}` sollte existieren mit `roles: ['visitor']`

2. **Login:**
   - Navigate zu `/auth/login`
   - Logge dich mit erstelltem Account ein
   - Prüfe Redirect zu Profil-Seite

3. **Reset Password:**
   - Navigate zu `/auth/reset-password`
   - Gib E-Mail-Adresse ein
   - Prüfe, ob E-Mail empfangen wurde

4. **Logout:**
   - Auf Profil-Seite: Klicke Logout
   - Prüfe Redirect zu Login-Seite

User-Profile testen

1. **Profil anzeigen:**
   - Navigate zu `/auth/profile`
   - Prüfe Anzeige von E-Mail, Display Name, Friend-Code

2. **Display Name bearbeiten:**
   - Klicke Edit-Button neben Display Name
   - Ändere Namen und speichere
   - Prüfe Firestore-Update

3. **Friend-Code-Suche:**
   - Gib Friend-Code eines anderen Users ein
   - Prüfe Anzeige der Benutzer-Informationen

Check-In testen

1. **User suchen:**
   - Navigate zu `/checkin` (in club-admin)
   - Gib Friend-Code ein
   - Prüfe, ob Benutzer gefunden wird

2. **Check-In durchführen:**
   - Klicke “Check-In” Button
   - Prüfe Firestore: `clubs/{clubId}/checkins/{checkInId}` sollte erstellt sein
   - Prüfe Status-Anzeige: “Eingecheckt seit…”

3. **Check-Out durchführen:**
   - Klicke “Check-Out” Button
   - Prüfe Firestore: `checkedOutAt` sollte gesetzt sein
   - Prüfe Status-Anzeige: “Nicht eingecheckt”

🚀 Nächste Schritte

Sofort umsetzbar:

1. ✅ Firebase-Konfiguration vervollständigen (`.env` Dateien)
2. ✅ TypeScript-Build testen: `pnpm build`
3. ✅ Dev-Server starten: `pnpm dev`
4. ✅ Manuelles Testing der Features

Mittel-/Langfristig:

1. **Unit-Tests schreiben:**
   - Auth-Funktionen testen
   - CRUD-Funktionen testen
   - Friend-Code-Suche testen

2. **Integration-Tests:**
   - End-to-End-Tests für Auth-Flow
   - Check-In/Check-Out Flow

3. **Fehlerbehandlung verbessern:**
   - Mehr detaillierte Fehlermeldungen
   - Retry-Logik für Firestore-Operationen

4. **Performance-Optimierung:**
   - Firestore-Caching
   - Real-time Updates optimieren

5. **Security Rules:**
   - Firestore Security Rules aktualisieren
   - Auth-basierte Zugriffskontrolle

6. **Weitere Features:**
   - QR-Code-Scanner für Friend-Codes
   - Cloakroom-Integration
   - Benachrichtigungen bei Check-In

📊 Status-Übersicht

Feature	Status	Bemerkung
Auth - Login	✅ Abgeschlossen	Bereits vorhanden
Auth - Signup	✅ Abgeschlossen	Erweitert mit Friend-Code
Auth - Logout	✅ Abgeschlossen	Bereits vorhanden
Auth - Reset Password	✅ Abgeschlossen	Neu implementiert
Auth - getCurrentUser	✅ Abgeschlossen	Bereits vorhanden
User-Profile CRUD	✅ Abgeschlossen	Neu implementiert
Friend-Code Generation	✅ Abgeschlossen	Bereits vorhanden
Friend-Code Unique	✅ Abgeschlossen	Neu implementiert
Friend-Code Search	✅ Abgeschlossen	Neu implementiert
Check-In CRUD	✅ Abgeschlossen	Neu implementiert
Check-In UI (Admin)	✅ Abgeschlossen	Neu implementiert
Profile UI	✅ Abgeschlossen	Erweitert
Login UI	✅ Abgeschlossen	Erweitert
Reset-Password UI	✅ Abgeschlossen	Neu implementiert

🎉 Zusammenfassung

Alle Core-Features wurden erfolgreich implementiert und konsolidiert!

- ✅ **Auth-System:** Vollständig mit resetPassword() erweitert
- ✅ **User-Profile:** Vollständige CRUD-Funktionen implementiert
- ✅ **Friend-Code:** Eindeutigkeit und Suche implementiert
- ✅ **Check-In:** Vollständige CRUD-Funktionen und UI implementiert
- ✅ **UI-Komponenten:** Alle notwendigen Seiten erstellt/erweitert

Nächste Schritte:
1. Firebase-Konfiguration vervollständigen
2. Testing durchführen
3. Weitere Features wie QR-Scanner und Cloakroom integrieren

Viel Erfolg mit Nightlife OS! 🎊