Complete Users → Accounts Migration Guide

🚨 Critical: System-Wide Impact

If you've migrated from "users" to "accounts", this affects:

1. **Database Schema**
   - Table name: `users` → `accounts`
   - Foreign keys: `user_id` → `account_id`
   - All references in other tables
2. **API Endpoints**
   - `/api/auth/user` → `/api/auth/account`
   - `/api/users/*` → `/api/accounts/*`
   - All request/response bodies
3. **Frontend Code**
   - All TypeScript types
   - All React hooks and queries
   - All component props
   - All state management
4. **Backend Services**
   - All method names
   - All variable names
   - All error messages
   - All logs

🔍 Comprehensive Search & Replace

1. Database Level

sql

```sql
-- Check all foreign key references
SELECT
    tc.table_name,
    kcu.column_name,
    ccu.table_name AS foreign_table_name,
    ccu.column_name AS foreign_column_name
FROM
    information_schema.table_constraints AS tc
    JOIN information_schema.key_column_usage AS kcu
      ON tc.constraint_name = kcu.constraint_name
    JOIN information_schema.constraint_column_usage AS ccu
      ON ccu.constraint_name = tc.constraint_name
WHERE constraint_type = 'FOREIGN KEY'
    AND (ccu.table_name='users' OR kcu.column_name='user_id');
```

2. Backend Code Updates

File: server/db/schema.ts

typescript

```typescript
// OLD
export const users = pgTable('users', {
  id: text('id').primaryKey(),
  // ...
});

// NEW
export const accounts = pgTable('accounts', {
  id: text('id').primaryKey(),
  email: text('email').notNull().unique(),
  name: text('name'),
  image: text('image'),
  youtube_channel_id: text('youtube_channel_id'),
  youtube_channel_title: text('youtube_channel_title'),
  // OAuth fields
  provider: text('provider').notNull().default('google'),
  providerAccountId: text('provider_account_id'),
  accessToken: text('access_token'),
  refreshToken: text('refresh_token'),
  expiresAt: bigint('expires_at', { mode: 'number' }),
  // Subscription fields
  subscriptionStatus: text('subscription_status').default('inactive'),
  subscriptionTier: text('subscription_tier'),
  // Timestamps
  createdAt: timestamp('created_at').defaultNow(),
  updatedAt: timestamp('updated_at').defaultNow()
});

// Update all foreign keys
export const tests = pgTable('tests', {
  id: text('id').primaryKey(),
  accountId: text('account_id').notNull().references(() => accounts.id), // Changed from userId
  // ...
});
```

File: server/storage.ts

typescript

```typescript
// Complete replacement of all user methods
export class DatabaseStorage implements IStorage {
  // Account methods (replacing ALL user methods)
  async getAccount(id: string): Promise<Account | undefined> {
    const [account] = await db.select().from(accounts).where(eq(accounts.id, id));
    return account || undefined;
  }

  async getAccountByEmail(email: string): Promise<Account | undefined> {
    const [account] = await db.select().from(accounts).where(eq(accounts.email, email));
    return account || undefined;
  }

  async createAccount(insertAccount: InsertAccount): Promise<Account> {
    const [account] = await db
      .insert(accounts)
      .values(insertAccount)
      .returning();
    return account;
  }

  async updateAccount(id: string, updateAccount: Partial<Account>): Promise<Account> {
    const [account] = await db
      .update(accounts)
      .set({ ...updateAccount, updatedAt: new Date() })
      .where(eq(accounts.id, id))
      .returning();
    return account;
  }

  // Update all test methods to use accountId
  async getTestsByAccountId(accountId: string): Promise<Test[]> {
    return await db
      .select()
      .from(tests)
      .where(eq(tests.accountId, accountId))
      .orderBy(desc(tests.createdAt));
  }

  // ... continue for ALL methods
}
```

File: server/routes.ts

typescript

```typescript
// Update ALL routes
app.get('/api/auth/account', requireAuth, async (req, res) => {
  const account = req.account!; // Changed from req.user
  res.json({ account });
});

app.get('/api/accounts/:id', requireAuth, async (req, res) => {
  const account = await storage.getAccount(req.params.id);
  if (!account) {
    return res.status(404).json({ error: 'Account not found' });
  }
  res.json({ account });
});

app.get('/api/tests', requireAuth, async (req, res) => {
  const tests = await storage.getTestsByAccountId(req.account!.id);
  res.json({ tests });
});
```

File: server/middleware/auth.ts

typescript

```typescript
// Update auth middleware
export const requireAuth = async (req: Request, res: Response, next: NextFunction) => {
  const token = req.cookies['sb-access-token'];

  if (!token) {
    return res.status(401).json({ error: 'Not authenticated' });
  }

  try {
    const { data: { user }, error } = await supabase.auth.getUser(token);
    if (error || !user) {
      return res.status(401).json({ error: 'Invalid session' });
    }

    const account = await storage.getAccountByEmail(user.email!);
    if (!account) {
      return res.status(401).json({ error: 'Account not found' });
    }

    req.account = account; // Changed from req.user
    next();
  } catch (error) {
    res.status(401).json({ error: 'Authentication failed' });
  }
};

// Update Express type definitions
declare global {
  namespace Express {
    interface Request {
      account?: Account; // Changed from user?: User
    }
  }
}
```

3. Frontend Code Updates

File: client/src/types/index.ts

typescript

```typescript
// Update all type definitions
export interface Account {
  id: string;
  email: string;
  name: string | null;
  image: string | null;
  youtube_channel_id: string | null;
  youtube_channel_title: string | null;
  subscriptionStatus: string;
  subscriptionTier: string | null;
  createdAt: Date;
  updatedAt: Date;
}

export interface Test {
  id: string;
  accountId: string; // Changed from userId
  videoId: string;
  // ...
}
```

File: client/src/lib/auth.ts

typescript

```typescript
export const authService = {
  async getCurrentAccount(): Promise<Account> {
    const response = await fetch('/api/auth/account', {
      credentials: 'include'
    });

    if (!response.ok) {
      if (response.status === 401) {
        throw new Error('Not authenticated');
      }
      throw new Error('Failed to get account');
    }

    const data = await response.json();
    return data.account;
  },

  async logout(): Promise<void> {
    await fetch('/api/auth/signout', {
      method: 'POST',
      credentials: 'include'
    });
  }
};
```

File: client/src/hooks/useAccount.ts

typescript

```typescript
// Rename from useUser.ts
import { useQuery } from '@tanstack/react-query';
import { authService } from '@/lib/auth';

export function useAccount() {
  return useQuery({
    queryKey: ['/api/auth/account'],
    queryFn: () => authService.getCurrentAccount(),
    staleTime: 5 * 60 * 1000, // 5 minutes
    retry: false
  });
}
```

Update ALL React components:

typescript

```typescript
// Example: Dashboard.tsx
export function Dashboard() {
  const { data: account, isLoading } = useAccount(); // Changed from useUser

  if (isLoading) return <Loading />;
  if (!account) return <Navigate to="/login" />;

  return (
    <div>
      <h1>Welcome, {account.youtube_channel_title || account.name}!</h1>
      <TestList accountId={account.id} /> {/* Changed from userId */}
    </div>
  );
}
```

4. Global Search & Replace Commands

bash

```bash
# Backend TypeScript/JavaScript files
find server -type f \( -name "*.ts" -o -name "*.js" \) -exec sed -i '' \
  -e 's/userId/accountId/g' \
  -e 's/user_id/account_id/g' \
  -e 's/getUser/getAccount/g' \
  -e 's/createUser/createAccount/g' \
  -e 's/updateUser/updateAccount/g' \
  -e 's/deleteUser/deleteAccount/g' \
  -e 's/req\.user/req.account/g' \
  -e 's/User\[\]/Account[]/g' \
  -e 's/: User/: Account/g' \
  -e 's/<User>/<Account>/g' \
  {} \;

# Frontend TypeScript/JavaScript files
find client/src -type f \( -name "*.ts" -o -name "*.tsx" -o -name "*.js" -o -name "*.jsx" \) -exec sed -i '' \
  -e 's/userId/accountId/g' \
  -e 's/useUser/useAccount/g' \
  -e 's/currentUser/currentAccount/g' \
  -e 's/data\.user/data.account/g' \
  -e 's/UserContext/AccountContext/g' \
  -e 's/UserProvider/AccountProvider/g' \
  {} \;
```

5. Database Migration Script

sql

```sql
-- Complete migration from users to accounts
BEGIN;

-- Rename the table
ALTER TABLE users RENAME TO accounts;

-- Rename all foreign key columns
ALTER TABLE tests RENAME COLUMN user_id TO account_id;
ALTER TABLE titles RENAME COLUMN user_id TO account_id;
ALTER TABLE analytics_polls RENAME COLUMN user_id TO account_id;
ALTER TABLE title_summaries RENAME COLUMN user_id TO account_id;
ALTER TABLE sessions RENAME COLUMN user_id TO account_id;
ALTER TABLE payment_records RENAME COLUMN user_id TO account_id;

-- Update all foreign key constraints
ALTER TABLE tests
  DROP CONSTRAINT tests_user_id_fkey,
  ADD CONSTRAINT tests_account_id_fkey
  FOREIGN KEY (account_id) REFERENCES accounts(id) ON DELETE CASCADE;

-- Repeat for all tables...

-- Update indexes
DROP INDEX IF EXISTS idx_users_email;
CREATE INDEX idx_accounts_email ON accounts(email);

DROP INDEX IF EXISTS idx_tests_user_id;
CREATE INDEX idx_tests_account_id ON tests(account_id);

-- Update any stored procedures/functions
-- Update any triggers

COMMIT;
```

⚠️ Critical Checklist

Before deploying these changes:

- [ ] **Database backup created**
- [ ] **All API endpoints updated**
- [ ] **All frontend queries updated**

- [ ] **All TypeScript types updated**
- [ ] **All error messages updated**
- [ ] **All logs updated**
- [ ] **All tests updated**
- [ ] **Documentation updated**
- [ ] **Environment variables checked** (any USER_* → ACCOUNT_*)

🚫 Common Pitfalls

1. **Mixed terminology** - Don't leave any "user" references
2. **Query keys** - React Query cache keys must all be updated
3. **Local storage** - Any stored user data needs migration
4. **Error messages** - "User not found" → "Account not found"
5. **Third-party integrations** - Stripe, YouTube API calls may need updates

🔄 Rollback Plan

Keep the old schema in a backup:

```sql
-- Create backup before migration
CREATE TABLE users_backup AS SELECT * FROM users;
CREATE TABLE tests_backup AS SELECT * FROM tests;
-- etc...
```

📝 Testing Strategy

1. **Unit tests** - Update all test files
2. **Integration tests** - Test full auth flow
3. **E2E tests** - Verify UI shows "account" terminology
4. **API tests** - Verify all endpoints work with new structure

This is a MAJOR architectural change that requires careful planning and execution. The artifacts I provided earlier are NOT ready for download because they mix users/accounts terminology. You need a comprehensive migration strategy first.