SmartTrip Authentication Flow

1. High-Level Overview

- **Auth Provider**: Supabase (email/password + Google OAuth).
- **Frontend**: Next.js 14 (`/auth` , `/auth/callback`) uses Supabase JS SDK.
- **Backend**: Flask verifies Supabase JWT using `SUPABASE_JWT_SECRET` .
- **API calls**: Frontend attaches `Authorization: Bearer <jwt>` header when a user is logged in.
- **Guests**: Fully supported – the app and API work without any token.

There are two main flows:

1. **Email & Password Sign-In / Sign-Up**
2. **Google OAuth Sign-In**

Both end with **"user has a valid Supabase session + JWT"**, which is then sent to the Flask backend on every API call.

---

2. End-to-End Flow Diagram

```
flowchart TD
    A[User opens /auth] --> B{Supabase configured?}
    B -- No --> G[Guest mode only<br/>Show 'Continue as guest']
    B -- Yes --> C[User chooses auth method]

    C -->|Email & Password| D[Submit login/sign-up form]
    C -->|Google OAuth| E[Click 'Continue with Google']

    D --> D1[Supabase email/password auth<br/>supabase.auth.signInWithPassword / signUp]
    D1 --> F[Supabase creates session + JWT in browser]

    E --> E1[Supabase opens Google OAuth page]
    E1 --> E2[Google authenticates user]
    E2 --> E3[Supabase redirects to /auth/callback<br/>with session/JWT]
    E3 --> F

    F --> H[Next.js stores session (Supabase client)]
    H --> I[User redirected to /search (or redirect target)]
    I --> J[Frontend calls Flask API via src/lib/api.ts]
    J --> K[api.ts adds Authorization: Bearer <JWT> header if session exists]
    K --> L[Flask receives request]
    L --> M[backend/auth_supabase.py verifies JWT with SUPABASE_JWT_SECRET]
    M -->|Valid JWT| N[Authenticated user in g.current_user]
    M -->|No/Invalid JWT| O[Guest request (no user)]
```

---

3. Frontend – Supabase Client & Session Handling

3.1 Supabase Client (singleton)

File: `src/lib/supabaseClient.ts`

```
const supabaseUrl = process.env.NEXT_PUBLIC_SUPABASE_URL;
const supabaseAnonKey = process.env.NEXT_PUBLIC_SUPABASE_ANON_KEY;

const isSupabaseConfigured = !!(supabaseUrl && supabaseAnonKey);

let supabase: SupabaseClient | null = null;

try {
  if (isSupabaseConfigured) {
    supabase = createClient(supabaseUrl!, supabaseAnonKey!, {
      auth: {
        persistSession: true,
        autoRefreshToken: true,
        detectSessionInUrl: true,
      },
    });
  }
} catch (error) {
  console.error('[Auth] Failed to initialize Supabase client:', error);
  supabase = null;
}
```

Key ideas:

- Reads `NEXT_PUBLIC_SUPABASE_URL` and `NEXT_PUBLIC_SUPABASE_ANON_KEY`.
- Creates a single Supabase client with:
  - `persistSession: true` – keeps the user logged in across refreshes.
  - `autoRefreshToken: true` – refreshes JWT before it expires.
  - `detectSessionInUrl: true` – used during OAuth callback.

3.2 Getting the Access Token for API Requests

File: `src/lib/supabaseClient.ts`

```
export async function getAccessToken(): Promise<string | null> {
  if (!supabase) {
    return null;
  }

  const session = await getCurrentSession();
  return session?.access_token || null;
}
```

This function is consumed by the API client to include the JWT in every backend call when a user is logged in.

---

4. Frontend – API Client Attaching JWT

File: `src/lib/api.ts`

4.1 Building Auth Headers

```
async function getAuthHeaders(): Promise<Record<string, string>> {
  try {
    if (typeof window === 'undefined') {
      return {}; // SSR: no auth
    }

    const { getAccessToken } = await import('./supabaseClient');
    const token = await getAccessToken();

    if (token) {
      return {
        'Authorization': `Bearer ${token}`,
      };
    }
  } catch (error) {
    if (error instanceof Error && !error.message.includes('Supabase not configured')) {
      console.warn('[API] Could not get auth token:', error);
    }
  }

  return {};
}
```

4.2 Using the Headers in Every Request

```
async function apiFetch<T>(
  endpoint: string,
  options?: RequestInit
): Promise<ApiResponse<T>> {
  try {
    const authHeaders = await getAuthHeaders();

    const response = await fetch(`${API_BASE_URL}${endpoint}`, {
      headers: {
        'Content-Type': 'application/json',
        ...authHeaders,
        ...options?.headers,
      },
      ...options,
    });

    const data = await response.json();

    if (!response.ok) {
      throw new Error(data.error || 'API request failed');
    }

    return data;
  } catch (error) {
    console.error('API Error:', error);
```

```
    return {
      success: false,
      error: error instanceof Error ? error.message : 'Unknown error occurred',
    };
  }
}
```

Result: For authenticated users, every call to `/api/...` automatically includes `Authorization: Bearer <supabase_jwt>`. Guests simply send no `Authorization` header.

---

5. Frontend – Auth Page (Email/Password + Google)

File: `src/app/auth/page.tsx`

5.1 Checking Existing Session on Load

```
useEffect(() => {
  if (!isAuthAvailable() || !supabase) {
    // Supabase not configured - allow guest access
    return;
  }

  supabase.auth.getSession().then(({ data: { session } }) => {
    if (session) {
      const redirectTo = searchParams.get('redirect') || '/search';
      router.push(redirectTo);
    }
  });
}, [router, searchParams]);
```

If the user already has a Supabase session, they are redirected away from the auth page to the desired screen (e.g. `/search`).

5.2 Email & Password Sign-Up

```
if (isSignUp) {
  const pwdError = validatePassword(password);
  if (pwdError) {
    setPasswordError(pwdError);
    setLoading(false);
    return;
  }

  if (password !== confirmPassword) {
    setPasswordError('הסיסמאות לא תואמות');
    setLoading(false);
    return;
  }
```

```
  const origin = typeof window !== 'undefined' ? window.location.origin : '';
  const { data, error: signUpError } = await supabase.auth.signUp({
    email,
    password,
    options: {
      emailRedirectTo: `${origin}/auth?redirect=/search`,
    },
  });

  if (signUpError) throw signUpError;

  if (data.user && !data.session) {
    setMessage('נא לבדוק את האימייל שלך לאישור החשבון');
    setEmail('');
    setPassword('');
    setConfirmPassword('');
  } else if (data.session) {
    const redirectTo = searchParams.get('redirect') || '/search';
    router.push(redirectTo);
  }
}
```

Key points:

- On sign-up, Supabase may:
  - Require email confirmation → `data.user && !data.session`.
  - Or auto-sign-in → `data.session` exists and the user is redirected.

5.3 Email & Password Sign-In

```
// Sign in
const { data, error: signInError } = await supabase.auth.signInWithPassword({
  email,
  password,
});

if (signInError) throw signInError;

if (data.session) {
  const redirectTo = searchParams.get('redirect') || '/search';
  router.push(redirectTo);
}
```

When credentials are correct, Supabase creates a session + JWT, and the user is redirected to the target page.

5.4 Google OAuth Flow Start

```
const redirectTo = searchParams.get('redirect') || '/search';
const origin = typeof window !== 'undefined' ? window.location.origin : '';
const callbackUrl = `${origin}/auth/callback?redirect=${encodeURIComponent(redirectTo)}`;
```

```
const { error: oauthError } = await supabase.auth.signInWithOAuth({
  provider: 'google',
  options: {
    redirectTo: callbackUrl,
  },
});
```

Important:

- `callbackUrl` must match a Redirect URL configured in the Supabase dashboard (`Settings → Authentication → URL Configuration`).
- Supabase will redirect back to `/auth/callback` after Google login.

5.5 Continue as Guest

```
const handleContinueAsGuest = (e: React.MouseEvent) => {
  e.preventDefault();
  e.stopPropagation();
  const redirectTo = searchParams.get('redirect') || '/search';
  window.location.href = redirectTo;
};
```

Guest users bypass Supabase entirely and just navigate to `search` (or any redirect target).

---

6. Frontend – OAuth Callback Handling

File: `src/app/auth/callback/page.tsx`

6.1 Reading the Supabase Session

```
useEffect(() => {
  const handleCallback = async () => {
    if (!isAuthAvailable() || !supabase) {
      const redirectTo = searchParams.get('redirect') || '/search';
      router.push(redirectTo);
      return;
    }

    try {
      const { data: { session }, error } = await supabase.auth.getSession();

      if (error) {
        router.push('/auth?error=authentication_failed');
        return;
      }

      if (session) {
        const redirectTo = searchParams.get('redirect') || '/search';
        router.push(redirectTo);
```

```
      } else {
        // Try to parse token from URL hash (misconfiguration cases)
        const hashParams = new URLSearchParams(window.location.hash.substring(1));
        const accessToken = hashParams.get('access_token');
        ...
      }
    } catch (err) {
      router.push('/auth?error=unexpected_error');
    }
  };

  handleCallback();
}, [router, searchParams]);
```

Flow:

1. User returns from Google → Supabase SDK parses the URL and creates a session.
2. `getSession()` returns the session; if present, the user is redirected to the original page.
3. If there is a misconfiguration (wrong domain), there is extra logic to detect Supabase's own domain and redirect to the correct Vercel URL with all params.

---

7. Backend – JWT Verification

File: `backend/auth_supabase.py`

7.1 Reading the Secret

```
SUPABASE_JWT_SECRET = os.environ.get('SUPABASE_JWT_SECRET')

if not SUPABASE_JWT_SECRET:
    print("[WARNING] SUPABASE_JWT_SECRET not set. JWT verification will be disabled.")
```

The value should be copied from Supabase → **Settings** → **API** → **JWT Secret** and set as an environment variable on Render.

7.2 Extracting and Verifying the Token

```
def get_current_user() -> Optional[Dict[str, Any]]:
    if not SUPABASE_JWT_SECRET:
        return None

    auth_header = request.headers.get('Authorization', '')

    if not auth_header.startswith('Bearer '):
        return None

    token = auth_header.split(' ', 1)[1]

    try:
        payload = jwt.decode(
```

```
            token,
            SUPABASE_JWT_SECRET,
            algorithms=['HS256'],
            audience='authenticated',
            options={
                'verify_signature': True,
                'verify_exp': True,
                'verify_aud': True,
            }
        )

        user_id = payload.get('sub')
        email = payload.get('email')

        if not user_id:
            return None

        return {
            'id': user_id,
            'email': email,
            'supabase_user_id': user_id,
        }
    except jwt.ExpiredSignatureError:
        print("[AUTH] JWT token expired")
        return None
    except jwt.InvalidTokenError as e:
        print(f"[AUTH] Invalid JWT token: {e}")
        return None
    except Exception as e:
        print(f"[AUTH] Error verifying JWT: {e}")
        return None
```

Notes:

- Uses `HS256` – the default Supabase signing algorithm.
- Checks:
  - Signature
  - Expiration
  - Audience (`"authenticated"`)
- Exposes a simple `dict` with `id` and `email` .

7.3 Decorating Protected Routes

File: `backend/auth_supabase.py`

```
def require_auth(f):
    @wraps(f)
    def decorated_function(*args, **kwargs):
        user = get_current_user()

        if not user:
            return jsonify({
```

```
                'success': False,
                'error': 'Authentication required'
            }), 401

        g.current_user = user
        return f(*args, **kwargs)

    return decorated_function
```

Use this for **strictly protected APIs**:

```
from auth_supabase import require_auth

@app.route('/api/user/preferences', methods=['GET'])
@require_auth
def get_preferences():
    user = g.current_user
    ...
```

7.4 Optional Authentication

```
def optional_auth(f):
    @wraps(f)
    def decorated_function(*args, **kwargs):
        user = get_current_user()
        g.current_user = user  # Can be None for guests
        return f(*args, **kwargs)

    return decorated_function
```

Use this for routes that should work for **both guests and logged-in users**, e.g. recommendations and analytics.

---

8. Putting It Together – Request Lifecycle Example

Scenario: Authenticated user opens the search page and triggers recommendations.

1. User logs in via `/auth` (email/password or Google).
2. Supabase JS creates a persistent session in the browser (contains `access_token`).
3. Frontend calls `getRecommendations(preferences)` from `src/lib/api.ts`.
4. `apiFetch`:
   - Calls `getAuthHeaders()`.
   - `getAuthHeaders()` imports `getAccessToken()` from `supabaseClient`.
   - `getAccessToken()` reads the current session and returns `access_token`.
   - `apiFetch` sends `Authorization: Bearer <access_token>` to Flask.
5. Flask receives request:
   - `get_current_user()` in `auth_supabase.py` verifies the JWT using `SUPABASE_JWT_SECRET`.
   - On success, it returns `{ id, email }` and attaches it to `g.current_user`.

6. Downstream code (e.g. event logging, future "save preferences" endpoints) can use `g.current_user` to store per-user data.
7. If the JWT is missing/invalid/expired:
   - `get_current_user()` returns `None`.
   - `optional_auth` makes the route behave as guest mode.
   - `require_auth` returns `401` with a clear error.

---

9. How to Generate This Guide as PDF

From the project root:

```
npx --yes md-to-pdf docs/AUTHENTICATION_FLOW.md
```

If you prefer keeping all guides under `docs/sap/`, you can move this file there and adjust the path:

```
npx --yes md-to-pdf docs/sap/AUTHENTICATION_FLOW.md
```

> *If `md-to-pdf` complains about a missing config file, you can ignore the warning or add a simple `docs/md-to-pdf-config.json` later. The core flow and examples in this guide will still work.*