Yestoryd Authentication System Documentation

Version: 2.0

Last Updated: January 17, 2026

Status: ✅ Unified & Production Ready

---

1. Overview

Yestoryd uses a **unified authentication system** via `lib/api-auth.ts` that handles all API route authentication. This replaces the previous fragmented approach that used separate `admin-auth.ts` and direct NextAuth imports.

Architecture

```
┌─────────────────────────────────────────────────────────────────┐
│                  YESTORYD AUTH FLOW                             │
├─────────────────────────────────────────────────────────────────┤
│                                                                 │
│  Browser Request                                                │
│       │                                                         │
│       ▼                                                         │
│  ┌─────────────┐                                                │
│  │  API Route  │                                                │
│  └──────┬──────┘                                                │
│         │                                                       │
│         ▼                                                       │
│  ┌───────────────────────────────────────────┐                  │
│  │  lib/api-auth.ts                          │                  │
│  │  ┌─────────────────────────────────┐      │                  │
│  │  │ getAuthenticatedUser()          │      │                  │
│  │  │ 1. Check Bearer Token (header)  │      │                  │
│  │  │ 2. Check Cookie Session (SSR)   │      │                  │
│  │  └─────────────────────────────────┘      │                  │
│  │              │                            │                  │
│  │              ▼                            │                  │
│  │  ┌─────────────────────────────────┐      │                  │
│  │  │ Role Verification               │      │                  │
│  │  │ - Admin: Check ADMIN_EMAILS list│      │                  │
│  │  │ - Coach: Check coaches table    │      │                  │
│  │  │ - Parent: Check parents table   │      │                  │
│  │  └─────────────────────────────────┘      │                  │
│  └───────────────────────────────────────────┘                  │
│                                                                 │
└─────────────────────────────────────────────────────────────────┘
```

User Roles

Role	Login Method	Token Type	Dashboard
Admin	Google OAuth	Cookie Session	`/admin/*`
Coach	Google OAuth	Bearer Token	`/coach/*`
Parent	OTP (Email)	Bearer Token	`/parent/*`

---

2. Core Auth File: `lib/api-auth.ts`

Complete Implementation

```typescript
```

```
// ============================================================
// FILE: lib/api-auth.ts
// ============================================================
// Unified Authentication Helper for Yestoryd API Routes
//
// Usage:
//   import { requireAdmin, getServiceSupabase } from '@/lib/api-auth';
//
//   const auth = await requireAdmin();
//   if (!auth.authorized) {
//     return NextResponse.json({ error: auth.error }, { status: 401 });
//   }
// ============================================================

import { createClient } from '@supabase/supabase-js';
import { createServerClient } from '@supabase/ssr';
import { cookies } from 'next/headers';
import { headers } from 'next/headers';

// --- ADMIN EMAIL WHITELIST ---
const ADMIN_EMAILS = [
  'amitraiforyou@gmail.com',
  'ruchapatil1010@gmail.com',
] as const;

// --- TYPES ---
export interface AuthResult {
  authorized: boolean;
  email?: string;
  userId?: string;
  role?: 'admin' | 'coach' | 'parent';
  coachId?: string;
  parentId?: string;
  error?: string;
}

// --- SERVICE SUPABASE CLIENT ---
// Uses service_role key for full database access
export const getServiceSupabase = () => createClient(
  process.env.NEXT_PUBLIC_SUPABASE_URL!,
  process.env.SUPABASE_SERVICE_ROLE_KEY!
);

// --- INTERNAL: Get Authenticated User ---
// Checks both Bearer token and cookie session
async function getAuthenticatedUser(): Promise<{
  user: { id: string; email?: string } | null;
  error: string | null;
}> {
  // Method 1: Check Bearer Token (Authorization header)
  const headersList = headers();
  const authHeader = headersList.get('authorization');

  if (authHeader?.startsWith('Bearer ')) {
    const token = authHeader.substring(7);
    try {
      const supabase = createClient(
        process.env.NEXT_PUBLIC_SUPABASE_URL!,
        process.env.NEXT_PUBLIC_SUPABASE_ANON_KEY!
      );
      const { data, error } = await supabase.auth.getUser(token);

      if (!error && data.user) {
        return { user: data.user, error: null };
      }
    } catch (e) {
      // Token invalid, try cookie method
    }
  }

  // Method 2: Check Cookie Session (Supabase SSR)
  try {
    const cookieStore = cookies();
    const supabase = createServerClient(
      process.env.NEXT_PUBLIC_SUPABASE_URL!,
      process.env.NEXT_PUBLIC_SUPABASE_ANON_KEY!,
      {
        cookies: {
          getAll() {
            return cookieStore.getAll();
          },
        },
      }
    );
    const { data, error } = await supabase.auth.getUser();

    if (!error && data.user) {
      return { user: data.user, error: null };
    }
  } catch (e) {
    // Cookie session invalid
  }
  return { user: null, error: 'Authentication required' };
}

// ============================================================
// PUBLIC FUNCTIONS
// ============================================================

/**
 * Require admin access - validates against ADMIN_EMAILS whitelist
 * Use for: /api/admin/*, sensitive operations
 */
export async function requireAdmin(): Promise<AuthResult> {
  const { user, error } = await getAuthenticatedUser();

  if (error || !user) {
    return { authorized: false, error: error || 'Authentication required' };
  }

  const email = user.email?.toLowerCase();
  if (!email) {
    return { authorized: false, error: 'Invalid session - no email' };
  }

  if (!ADMIN_EMAILS.includes(email)) {
    console.warn(JSON.stringify({
      event: 'admin_access_denied',
      email,
      userId: user.id,
      timestamp: new Date().toISOString(),
    }));
    return { authorized: false, error: 'Admin access required', email };
  }

  return { authorized: true, email, userId: user.id, role: 'admin' };
}

/**
 * Require coach access - validates against coaches table
 * Use for: /api/coach/*, coach-only operations
 */
export async function requireCoach(): Promise<AuthResult> {
  const { user, error } = await getAuthenticatedUser();

  if (error || !user) {
    return { authorized: false, error: error || 'Authentication required' };
  }
  const email = user.email?.toLowerCase();
  if (!email) {
    return { authorized: false, error: 'Invalid session - no email' };
  }

  const supabase = getServiceSupabase();
  const { data: coach } = await supabase
    .from('coaches')
    .select('id, is_active')
    .eq('email', email)
    .single();

  if (!coach || !coach.is_active) {
    return { authorized: false, error: 'Coach access required', email };
  }

  return {
    authorized: true,
    email,
    userId: user.id,
    role: 'coach',
    coachId: coach.id
  };
}

/**
 * Require admin OR coach access
 * Use for: /api/discovery-call/*, shared admin/coach features
 */
export async function requireAdminOrCoach(): Promise<AuthResult> {
  const { user, error } = await getAuthenticatedUser();

  if (error || !user) {
    return { authorized: false, error: error || 'Authentication required' };
  }

  const email = user.email?.toLowerCase();
  if (!email) {
    return { authorized: false, error: 'Invalid session - no email' };
  }

  // Check admin first (fast - just array lookup)
  if (ADMIN_EMAILS.includes(email)) {
    return { authorized: true, email, userId: user.id, role: 'admin' };
  }

  // Check coach (requires DB lookup)
  const supabase = getServiceSupabase();
  const { data: coach } = await supabase
    .from('coaches')
    .select('id, is_active')
    .eq('email', email)
    .single();

  if (coach && coach.is_active) {
    return {
      authorized: true,
      email,
      userId: user.id,
      role: 'coach',
      coachId: coach.id
    };
  }

  return { authorized: false, error: 'Admin or Coach access required', email };
}

/**
 * Require authenticated user (any role)
 * Use for: General authenticated routes
 */
export async function requireAuth(): Promise<AuthResult> {
  const { user, error } = await getAuthenticatedUser();

  if (error || !user) {
    return { authorized: false, error: error || 'Authentication required' };
  }

  return {
    authorized: true,
    email: user.email?.toLowerCase(),
    userId: user.id
  };
}

/**
 * Get authenticated user if available (optional auth)
 * Returns user info if authenticated, null if not
 * Use for: Routes where auth enhances but isn't required
 */
export async function getOptionalAuth(): Promise<{
  email?: string;
  userId?: string;
  parentId?: string;
  coachId?: string;
  role?: string;
} | null> {
  const { user, error } = await getAuthenticatedUser();

  if (error || !user) {
    return null;
  }

  const email = user.email?.toLowerCase();

  // Check if admin
  if (email && ADMIN_EMAILS.includes(email)) {
    return { email, userId: user.id, role: 'admin' };
  }

  // Check if coach
  const supabase = getServiceSupabase();
  const { data: coach } = await supabase
    .from('coaches')
    .select('id')
    .eq('email', email)
    .single();

  if (coach) {
    return { email, userId: user.id, coachId: coach.id, role: 'coach' };
  }

  // Check if parent
  const { data: parent } = await supabase
    .from('parents')
    .select('id')
    .eq('email', email)
    .single();

  if (parent) {
    return { email, userId: user.id, parentId: parent.id, role: 'parent' };
  }

  return { email, userId: user.id };
}

// ============================================================
// UTILITY FUNCTIONS
// ============================================================

/**
```

```
 * Check if email is admin
 */
export function isAdminEmail(email: string | null | undefined): boolean {
  if (!email) return false;
  return ADMIN_EMAILS.includes(email.toLowerCase());
}

/**
 * Get admin emails list
 */
export function getAdminEmails(): readonly string[] {
  return ADMIN_EMAILS;
}
```

---

3. Usage Examples

Example 1: Admin-Only Route

```
typescript
```

```
// app/api/admin/dashboard/route.ts
import { NextRequest, NextResponse } from 'next/server';
import { requireAdmin, getServiceSupabase } from '@/lib/api-auth';

export async function GET(request: NextRequest) {
  // 1. Authenticate - Admin only
  const auth = await requireAdmin();
  if (!auth.authorized) {
    return NextResponse.json(
      { error: auth.error },
      { status: 401 }
    );
  }

  // 2. Authorized - proceed with admin logic
  const supabase = getServiceSupabase();
  const { data } = await supabase
    .from('enrollments')
    .select('*')
    .limit(10);

  return NextResponse.json({
    success: true,
    adminEmail: auth.email,
    data
  });
}
```

Example 2: Admin OR Coach Route

typescript

```typescript
// app/api/discovery-call/pending/route.ts
import { NextRequest, NextResponse } from 'next/server';
import { requireAdminOrCoach, getServiceSupabase } from '@/lib/api-auth';

export async function GET(request: NextRequest) {
  // 1. Authenticate - Admin or Coach
  const auth = await requireAdminOrCoach();
  if (!auth.authorized) {
    return NextResponse.json(
      { error: auth.error },
      { status: 401 }
    );
  }

  const supabase = getServiceSupabase();

  // 2. Role-based data filtering
  let query = supabase.from('discovery_calls').select('*');

  if (auth.role === 'coach') {
    // Coaches only see their assigned calls
    query = query.eq('assigned_coach_id', auth.coachId);
  }
  // Admins see all calls (no filter)

  const { data } = await query;

  return NextResponse.json({
    success: true,
    role: auth.role,
    data
  });
}
```

Example 3: Optional Auth Route

typescript

```
// app/api/coupons/validate/route.ts
import { NextRequest, NextResponse } from 'next/server';
import { getOptionalAuth, getServiceSupabase } from '@/lib/api-auth';

export async function POST(request: NextRequest) {
  const body = await request.json();
  const { coupon_code } = body;

  // 1. Optional auth - enhances but not required
  const session = await getOptionalAuth();
  const parentId = session?.parentId;

  const supabase = getServiceSupabase();

  // 2. Validate coupon
  const { data: coupon } = await supabase
    .from('coupons')
    .select('*')
    .eq('code', coupon_code)
    .single();

  if (!coupon) {
    return NextResponse.json({ error: 'Invalid coupon' }, { status: 400 });
  }

  // 3. Per-user limit check (only if authenticated)
  if (parentId && coupon.per_user_limit) {
    const { count } = await supabase
      .from('coupon_usages')
      .select('*', { count: 'exact', head: true })
      .eq('coupon_id', coupon.id)
      .eq('parent_id', parentId);

    if (count && count >= coupon.per_user_limit) {
      return NextResponse.json({ error: 'Coupon limit exceeded' }, { status: 400 });
    }
  }

  return NextResponse.json({
    valid: true,
    discount: coupon.discount_percent
  });
}
```

Example 4: Multi-Role Chat Route (Complex)

typescript

```
// app/api/chat/route.ts
import { NextRequest, NextResponse } from 'next/server';
import { getOptionalAuth, getServiceSupabase } from '@/lib/api-auth';

export async function POST(request: NextRequest) {
  const session = await getOptionalAuth();
  const supabase = getServiceSupabase();

  let userEmail: string;
  let userRole: 'admin' | 'coach' | 'parent';
  let coachId: string | undefined;

  if (session?.email) {
    // Authenticated via Supabase
    userEmail = session.email;

    if (session.role === 'admin') {
      userRole = 'admin';
    } else if (session.role === 'coach') {
      userRole = 'coach';
      coachId = session.coachId;
    } else if (session.role === 'parent') {
      userRole = 'parent';
    } else {
      return NextResponse.json({ error: 'Invalid role' }, { status: 403 });
    }
  } else {
    // Fallback: Check request body for credentials
    const body = await request.clone().json();
    const requestEmail = body.userEmail;
    const requestRole = body.userRole;

    if (!requestEmail || !requestRole) {
      return NextResponse.json({ error: 'Auth required' }, { status: 401 });
    }

    // Verify email exists in appropriate table
    if (requestRole === 'coach') {
      const { data: coach } = await supabase
        .from('coaches')
        .select('id')
        .eq('email', requestEmail)
        .single();

      if (!coach) {
        return NextResponse.json({ error: 'Coach not found' }, { status: 401 });
      }
      coachId = coach.id;
    }

    userEmail = requestEmail;
    userRole = requestRole;
  }

  // Continue with role-based chat logic...
  return NextResponse.json({
    success: true,
    userRole,
    userEmail
  });
}
```

---

4. Auth Function Reference

Function	Use Case	Returns
`requireAdmin()`	Admin-only routes	`AuthResult` with admin verification
`requireCoach()`	Coach-only routes	`AuthResult` with coachId
`requireAdminOrCoach()`	Shared routes	`AuthResult` with role + optional coachId
`requireAuth()`	Any authenticated user	`AuthResult` with basic user info
`getOptionalAuth()`	Optional auth routes	User info or `null`
`getServiceSupabase()`	Database access	Supabase client with service_role
`isAdminEmail(email)`	Quick admin check	`boolean`

---

5. Route Authorization Matrix

Admin Routes (`/api/admin/*`)

Route	Auth Function	Access
`/api/admin/dashboard`	`requireAdmin()`	Admin only
`/api/admin/crm/*`	`requireAdmin()`	Admin only

Route	Auth Function	Access
`/api/admin/payouts`	`requireAdmin()`	Admin only
`/api/admin/settings`	`requireAdmin()`	Admin only
`/api/admin/coaches/*`	`requireAdmin()`	Admin only

Discovery Call Routes

Route	Auth Function	Access
`/api/discovery-call/pending`	`requireAdminOrCoach()`	Admin + Coach
`/api/discovery-call/[id]`	`requireAdminOrCoach()`	Admin + Coach
`/api/discovery-call/assign`	`requireAdminOrCoach()`	Admin + Coach
`/api/discovery-call/[id]/questionnaire`	`requireAdminOrCoach()`	Admin + Coach
`/api/discovery-call/[id]/send-payment-link`	`requireAdminOrCoach()`	Admin + Coach

Public/Optional Auth Routes

Route	Auth Function	Access
`/api/coupons/validate`	`getOptionalAuth()`	Anyone (enhanced if logged in)
`/api/coupons/calculate`	`getOptionalAuth()`	Anyone (enhanced if logged in)
`/api/chat`	`getOptionalAuth()`	Requires valid credentials
`/api/assessment/*`	None	Public
`/api/payment/*`	None	Public (Razorpay webhooks)

---

6. Security Features

1. Admin Whitelist

```
typescript
```

```
const ADMIN_EMAILS = [
  'amitraiforyou@gmail.com',
  'ruchapatil1010@gmail.com',
] as const;
```

2. Dual Token Support

- **Bearer Token**: `Authorization: Bearer <token>` header
- **Cookie Session**: Supabase SSR cookie authentication

3. Database Verification

- Coach access: Verified against `coaches` table + `is_active` check
- Parent access: Verified against `parents` table

4. Request Logging

```typescript
console.warn(JSON.stringify({
  event: 'admin_access_denied',
  email,
  userId: user.id,
  timestamp: new Date().toISOString(),
}));
```

---

7. Migration Notes

From `admin-auth.ts` to `api-auth.ts`

Before:

```typescript
import { requireAdmin, getSupabase } from '@/lib/admin-auth';
```

After:

```typescript
import { requireAdmin, getServiceSupabase } from '@/lib/api-auth';
```

Key Changes

Old	New
`getSupabase()`	`getServiceSupabase()`
`getSupabase as getAdminSupabase`	`getServiceSupabase`
`from '@/lib/admin-auth'`	`from '@/lib/api-auth'`
`getServerSession(authOptions)`	`getOptionalAuth()`
`session?.user?.email`	`session?.email`
`(session.user as any).role`	`session.role`

---

8. Troubleshooting

Error: 401 Unauthorized

Possible causes:

1. No Authorization header or cookie
2. Expired token
3. User not in authorized list (admin) or table (coach)

Debug:

typescript

```typescript
const auth = await requireAdmin();
console.log('Auth result:', auth);
// Check auth.error for specific message
```

Error: Coach can't access discovery calls

Solution: Use `requireAdminOrCoach()` instead of `requireAdmin()`

Error: Token valid but auth fails

Check:

1. `coaches.is_active` is `true` for coach
2. Email matches exactly (lowercase comparison)

---

9. Files Reference

File	Purpose
`lib/api-auth.ts`	**Main auth file** - all functions
`lib/auth-options.ts`	NextAuth configuration (for OAuth)
`app/api/auth/[...nextauth]/route.ts`	NextAuth handler
`middleware.ts`	Route protection (redirects)

---

10. Summary

- ✅ **55 API routes** using unified `lib/api-auth.ts`
- ✅ **Supports**: Admin, Coach, Parent roles
- ✅ **Token types**: Bearer + Cookie
- ✅ **Functions**: 6 auth helpers
- ✅ **Old file deleted**: `lib/admin-auth.ts`

Document generated: January 17, 2026
Yestoryd - AI-Powered Reading Intelligence Platform