EXAMTRACKER INDIA

Authentication & Registration Architecture

Technical Reference • February 2026 • v1.0

Overview

ExamTracker uses a fully **passwordless authentication** system. There are no passwords to remember, no password resets, and no credential leaks. Users verify their identity either via a **6-digit OTP sent to their phone (SMS via MSG91)**, a **magic link sent to their email (via Resend)**, or **Google OAuth**. All three methods are handled by Supabase Auth as the central auth layer.

WHY PASSWORDLESS?	Indian aspirants use low-cost Android phones, often shared with family. Passwords get forgotten, especially for apps used infrequently between exam cycles. Phone OTP has the lowest friction for this demographic. Magic link covers users who prefer email. Google OAuth covers tech-comfortable users. No method requires remembering anything.

Three Authentication Methods

Method	Provider	Best For	Priority	Cost per use
Phone OTP	*MSG91 (India)*	90%+ of users — budget Android, rural, semi-urban users who prefer SMS	**PRIMARY**	₹0.25 / OTP (MSG91)
Magic Link Email	*Resend*	Users who prefer email or don't want to share phone number	**SECONDARY**	~₹0.006 / email (Resend)
Google OAuth	*Supabase + Google*	Tech-comfortable users, college students, users who already use Google	**TERTIARY**	Free

MSG91 vs TWILIO	Twilio charges $0.0832 (~₹7) per SMS to India. MSG91 charges ₹0.25. That is a 28x cost difference. At 10,000 signups/month, Twilio = ₹70,000/month on OTPs alone vs MSG91 = ₹2,500/month. MSG91 also handles India's mandatory TRAI DLT compliance (entity + template registration with telecom operators). Twilio does not. Use Supabase's SMS Hook to connect MSG91 — documented clearly since Dec 2025.

Flow 1: Phone OTP — New User Registration

This is the primary registration path. The user enters their phone number, receives a 6-digit OTP via SMS, enters it, and is registered. Supabase Auth manages the session. MSG91 delivers the SMS.

STEP 1 — **User enters phone number**
Landing page / Sign up screen

What the user sees: A single input field: "Enter your mobile number" with +91 pre-filled. A "Send OTP" button. No email, no name, nothing else on this screen. Name and other details are collected during onboarding — not at registration.

Frontend: Validate Indian phone format (+91XXXXXXXXXX, 10 digits after country code). Show error inline if invalid. Disable button until valid number entered.

Rate limit: User cannot request another OTP for 60 seconds. Show a countdown timer after first send. Maximum 5 OTP requests per phone number per hour (enforced by Supabase Auth config).

STEP 2 — **Supabase generates OTP + fires SMS Hook**
Backend / Supabase Auth

When the user hits "Send OTP", the frontend calls supabase.auth.signInWithOtp({ phone }). Supabase Auth generates a 6-digit numeric OTP, hashes it, stores it in auth.users along with confirmation_sent_at, then fires the **Send SMS Auth Hook** — an HTTP POST to a Supabase Edge Function you control.

SMS HOOK	The Edge Function receives: { phone, otp }. It calls MSG91's Send OTP API with the phone number and OTP. MSG91 delivers the SMS using your pre-approved DLT template. The hook returns 200 OK to Supabase. This is the only integration point — Supabase handles all token management, MSG91 handles only delivery.

STEP 3 — **User receives SMS and enters OTP**
User / Phone

The user receives an SMS: "Your ExamTracker OTP is 847291. Valid for 10 minutes. Do not share. -ExamTracker"

The screen shows a 6-box OTP input (one digit per box — standard pattern). Auto-focus advances to next box on each digit. Paste support for when SMS auto-fills on Android.

OTP expiry: 10 minutes (configured in Supabase Auth dashboard). After expiry, user must request a new OTP. Show countdown on screen.

Resend OTP: "Resend OTP" link appears after 30 seconds. Calls the same signInWithOtp again. Supabase generates a new OTP — the old one is invalidated immediately.

STEP 4 — **Frontend verifies OTP with Supabase**
Backend / Supabase Auth

Frontend calls supabase.auth.verifyOtp({ phone, token, type: "sms" }). Supabase validates the token against the stored hash.

Outcome	Supabase Response	Frontend Action
✅ **Valid OTP**	Returns session (access_token + refresh_token). Creates user in auth.users if new.	Store session. Check if user has a profile in user_profiles table. If yes → Dashboard. If no → Onboarding Flow.
❌ **Wrong OTP**	Returns error: "Token has expired or is invalid"	Show inline error: "Incorrect OTP. X attempts remaining." After 5 wrong attempts, lock for 10 minutes.

Outcome	Supabase Response	Frontend Action
⏰ Expired OTP	Returns error: "Token has expired"	Show: "OTP expired. Request a new one." Show Resend button immediately.

STEP 5 **New user profile created → Onboarding begins**
Backend + Frontend

On first sign-in (no existing profile), the backend creates a minimal **users** row with the phone number and a UUID. No other data yet — onboarding collects everything else. Frontend redirects to the 9-screen onboarding flow.

On subsequent sign-ins (profile exists), skip onboarding entirely and go straight to the dashboard.

Flow 2: Phone OTP — Returning User Login

Login and registration use the exact same flow. There is no separate "login" vs "register" path. The user enters their phone number, gets OTP, verifies it. Supabase Auth checks if the phone number exists in auth.users — if yes, it's a login; if no, it creates a new account.

NO SEPARATE LOGIN/REGISTER SCREENS This is intentional. Having two screens (one saying "Login", one saying "Register") confuses users who don't remember if they've signed up before. One screen that says "Enter your phone number" handles both. If the number is new → account created. If the number exists → logged in. User never has to think about it.

Step	Actor	Action	Result / Data
1	*User*	**Enters phone number on the same sign-in screen**	Frontend validates +91XXXXXXXXXX format
2	*Frontend → Supabase*	**Calls signInWithOtp({ phone })**	Supabase fires SMS Hook → MSG91 sends OTP SMS
3	*User*	**Enters 6-digit OTP**	OTP input screen identical to registration
4	*Frontend → Supabase*	**Calls verifyOtp({ phone, token, type: "sms" })**	Supabase checks: phone exists in auth.users?
5a	*Supabase (existing user)*	**Validates OTP, returns existing session**	user_id found → has profile → redirect to Dashboard
5b	*Supabase (new user)*	**Validates OTP, creates new auth.users record**	No profile found → redirect to Onboarding

Flow 3: Magic Link via Email

For users who prefer email over phone. The user enters their email address, receives a one-time sign-in link (valid for 1 hour), clicks it, and is authenticated. No OTP to type, no password. The link contains a token that Supabase validates.

STEP 1 **User enters email address**
Sign-in screen — "Use email instead" option

A secondary "Use email instead" option below the phone input. Clicking it swaps to an email input field. User enters email and clicks "Send Magic Link".

Frontend call: supabase.auth.signInWithOtp({ email, options: { emailRedirectTo: "https://examtracker.in/auth/callback" } })

STEP 2 — Supabase generates token + Resend delivers email

Backend / Supabase Auth + Resend

Supabase Auth generates a secure one-time token and stores its hash. It then calls Resend's API to send a transactional email to the user.

RESEND INTEGRATION	Supabase Auth has native Resend integration. In the Supabase dashboard → Auth → SMTP settings, set custom SMTP to Resend's SMTP endpoint (smtp.resend.com:465) with your Resend API key. All auth emails (magic link, email confirmation) then route through Resend instead of Supabase's default email. You already use Resend from Prepleague so this is familiar.

Email subject: "Sign in to ExamTracker"

Email body (simple): "Click the button below to sign in to ExamTracker. This link expires in 1 hour and can only be used once."

Email template: Design in Resend dashboard. Keep it minimal — ExamTracker logo, one big "Sign In" button, small text with expiry info. No marketing content in auth emails.

STEP 3 — User clicks link — Supabase validates token

Browser redirect

The link contains the token as a URL fragment: https://examtracker.in/auth/callback#access_token=...&type=magiclink

The **/auth/callback** page on the frontend calls supabase.auth.getSession() which reads the token from the URL fragment and completes the authentication. Supabase validates it server-side, creates the session, and returns the user.

Same branching as phone OTP: new user → Onboarding. Returning user → Dashboard.

MAGIC LINK EDGE CASE	A user who first signed up with phone OTP then tries magic link with the same email will create a separate unlinked account. Handle this in onboarding: if a user with an existing phone account tries to add email, use supabase.auth.updateUser({ email }) to link the email to their existing account, not create a new one. Identity linking is a Supabase Auth feature — use it.

Flow 4: Google OAuth

One-tap Google sign-in. Supabase handles the entire OAuth flow. No custom backend code needed.

Step	Actor	Action	Result / Data
1	*User*	**Clicks "Continue with Google" button**	Frontend calls supabase.auth.signInWithOAuth({ provider: "google" })
2	*Supabase → Google*	**Redirects to Google's OAuth consent screen**	User sees Google account picker
3	*User → Google*	**Selects their Google account**	Google authenticates user, returns auth code to Supabase callback URL
4	*Google → Supabase*	**Supabase exchanges code for tokens**	Supabase creates/updates user in auth.users with Google identity

Step	Actor	Action	Result / Data
5	*Supabase → Frontend*	**Redirects to /auth/callback with session**	Frontend reads session, checks for profile → Onboarding or Dashboard

Setup required: Add ExamTracker as OAuth app in Google Cloud Console. Set redirect URI to https://[your-supabase-project].supabase.co/auth/v1/callback. Copy Client ID + Secret into Supabase Auth → Providers → Google.

Session Management

How sessions work

Supabase Auth issues a JWT **access_token** (valid 1 hour) and a **refresh_token** (valid 7 days by default — extend to 30 days for this product). The Supabase JS client handles token refresh automatically in the background.

Token	Lifetime	Notes
access_token (JWT)	60 minutes	Auto-refreshed by supabase-js. Contains user_id, phone/email, role.
refresh_token	30 days (configure in Supabase)	Stored in localStorage. Used to get new access_token when expired. Single-use — rotated on every refresh.

"Lost Access" — What replaces Forgot Password

Since there are no passwords, "forgot password" does not exist. Instead there are two lost access scenarios:

- Lost phone number or new SIM: User switches to magic link email on the sign-in screen. If they have no email linked, there is currently no self-service recovery — support contact required. This is rare and acceptable at MVP.
- Lost access to email: User switches to phone OTP. Same phone = same account.
- Best practice during onboarding: After completing onboarding, nudge user to add both phone AND email to their account ("Add your email as backup — so you never lose access"). This reduces lost access cases to near zero.

Account Linking — One User, Multiple Sign-In Methods

A user who first signs in with phone OTP and later wants to also use magic link (or vice versa) should not end up with two separate accounts. Supabase Auth supports identity linking to prevent this.

When does this matter?

- User signed up with phone OTP. Months later, tries "Sign in with Google" using their Gmail. Without linking, this creates a duplicate account with no profile or tracked exams.
- User signed up with magic link. Later wants to add phone for WhatsApp notifications. Should link phone to existing account, not create new one.

How to handle it:

- During onboarding Step 2 (after initial registration), ask for the secondary identifier: "Add your email to never lose access" (for phone signups) or "Add your phone for WhatsApp alerts" (for email signups).

- Use supabase.auth.updateUser({ email }) or supabase.auth.updateUser({ phone }) on the already-authenticated session. This adds the new identifier to the existing auth.users record — not a new record.
- For Google OAuth: if a user authenticates with Google and their Gmail matches an existing magic-link account, Supabase v2 auto-links them. Verify this is enabled in Supabase Auth settings (Link identities between providers).

Security Rules & Rate Limits

Rule	Limit	Why
OTP request cooldown	**60 seconds**	Prevents SMS flooding on a single number
Max OTP requests per phone/hour	**5 attempts**	Prevents abuse of MSG91 credits
Wrong OTP attempts before lock	**5 attempts**	After 5 wrong attempts, lock phone number for 10 minutes
OTP expiry	**10 minutes**	Standard — long enough for slow typers, short enough for security
Magic link expiry	**60 minutes**	Standard Supabase default — sufficient time to check email
Refresh token lifetime	**30 days**	Keep users logged in across sessions; user expects this from a tracking app
Session per device	**Unlimited**	Users may use phone + computer + shared family tablet — don't force logout
Supabase RLS on users table	**User can only read/write own row**	Row-Level Security — no user can access another user's data

Supabase Auth Configuration Checklist

Everything that must be configured in the Supabase dashboard before auth goes live:

Auth → General Settings

- Site URL: https://examtracker.in
- Redirect URLs: https://examtracker.in/auth/callback (add localhost:3000 for dev)
- JWT expiry: 3600 (1 hour)
- Refresh token rotation: ENABLED
- Refresh token reuse interval: 10 seconds (prevents race conditions)
- Password login: DISABLED (you are passwordless)

Auth → Providers

- Phone: ENABLED. SMS Provider: Custom (SMS Hook, not Twilio). Configure the Edge Function URL.
- Email: ENABLED. Disable "Confirm email" (magic link handles this). Custom SMTP → Resend credentials.
- Google: ENABLED. Add Client ID + Client Secret from Google Cloud Console.

Auth → Email Templates

- Magic Link template: Customise in Resend dashboard, set Supabase to use Resend SMTP.
- Confirm signup: Disabled (not needed for magic link flow).

Auth → Rate Limits

- OTP request rate: 5 per hour per phone
- Sign in rate: 10 per 5 minutes per IP (Supabase default — keep it)

MSG91 Setup (one-time)

- Register on msg91.com. Complete KYC.
- Register entity on TRAI DLT portal (mandatory for India SMS). MSG91 can guide through this.
- Create OTP SMS template: "Your ExamTracker OTP is {{otp}}. Valid for 10 minutes. Do not share. -ExamTracker" — submit for DLT approval (takes 24-48 hours).
- Get API key from MSG91 dashboard. Add to Supabase Edge Function environment variables.
- Create Supabase Edge Function: receives { phone, otp } from SMS Hook, calls MSG91 Send OTP API, returns 200.

Bridge to Onboarding: What Happens After Auth

Authentication just answers: "Who are you?" Onboarding answers: "Tell us about yourself so we can match you to exams." These are two separate flows, not one.

User State	Condition Check	Redirect To
Brand new user	auth.users exists, but NO row in user_profiles for this user_id	**/onboarding — begin 9-screen profile collection flow**
Partially onboarded	user_profiles row exists, onboarding_completed = FALSE. Check onboarding_step.	**/onboarding?step={onboarding_step} — resume from where they left off**
Fully onboarded	user_profiles row exists, onboarding_completed = TRUE	**/dashboard — straight to their personalised exam list**

The condition check happens in the **/auth/callback** Next.js route handler — a single server-side check immediately after Supabase confirms the session. It queries SELECT onboarding_completed, onboarding_step FROM user_profiles WHERE user_id = $1 and redirects accordingly.