

# Roblox Verifier Tool - Comprehensive Application Documentation

---

**Version:** 1.0.0

**Last Updated:** October 21, 2025

**Maintained By:** Development Team

**Purpose:** Complete technical documentation for onboarding new developers and AI agents

---

## Table of Contents

---

1. [Executive Summary](#)
  2. [Technical Architecture](#)
  3. [Database Schema](#)
  4. [Authentication & Authorization](#)
  5. [Features - Detailed Breakdown](#)
  6. [API Endpoints](#)
  7. [Frontend Components](#)
  8. [External Integrations](#)
  9. [Environment Variables](#)
  10. [Development History & Improvements](#)
  11. [Known Issues & Limitations](#)
  12. [Deployment Guide](#)
  13. [File Structure](#)
  14. [Development Workflow](#)
  15. [Important Context & Gotchas](#)
- 

## 1. Executive Summary

---

### 1.1 What is the Roblox Verifier Tool?

The **Roblox Verifier Tool** is a multi-tenant SaaS application that enables users to verify and search for Roblox user profiles, usernames, and display names. It provides three distinct search modes (Exact Match, Smart Match, and Display Name search) with AI-powered ranking, forensic evidence collection, and comprehensive admin dashboards.

### 1.2 Who Uses It?

- **Super Administrators:** Platform owners who manage multiple customer organizations
- **Customer Admins:** Company admins who have access to search tools for their organization
- **Customer Users** (future): Regular users within a customer organization
- **Target Use Cases:**
  - Law enforcement agencies investigating Roblox accounts
  - Content moderation teams verifying user identities

- Digital marketing agencies researching Roblox influencers
- Parents monitoring their children's online activity
- HR departments conducting background checks

### 1.3 Key Value Propositions

1. **Multi-Tenant Architecture:** One application serves multiple isolated customer organizations
2. **Three Search Modes:**
  - **Exact Match:** Direct username/user ID lookup (fast, no cooldown)
  - **Smart Match:** AI-powered fuzzy search with confidence scoring (30-second cooldown)
  - **Display Name:** Search by display name showing all matches (30-second cooldown)
3. **Forensic Mode:** Generate tamper-proof evidence reports with SHA-256 hashes
4. **Deep Context Profiles:** View comprehensive user profiles including friends, groups, activity, badges
5. **Batch Processing:** Upload CSV files to verify multiple users at once
6. **Admin Dashboard:** Super admins can manage customers, users, and view analytics
7. **Search History:** All searches logged for auditing and analytics
8. **White-Label Support:** Customer logos for branded experiences
9. **Rate Limiting Protection:** Built-in cooldowns and circuit breakers to avoid API bans

### 1.4 Tech Stack Summary

Layer	Technology	Purpose
Frontend	React 19.1, Next.js 15.5	UI framework
Backend	Next.js API Routes	Serverless backend
Database	PostgreSQL (Supabase)	Data persistence
Storage	Supabase Storage	Customer logo hosting
Authentication	NextAuth.js 4.24	User authentication
Password Hashing	bcrypt	Secure password storage
Styling	Tailwind CSS 3.4	UI styling
Deployment	Vercel	Hosting & CI/CD
Caching	Redis (optional)	Rate limiting & caching

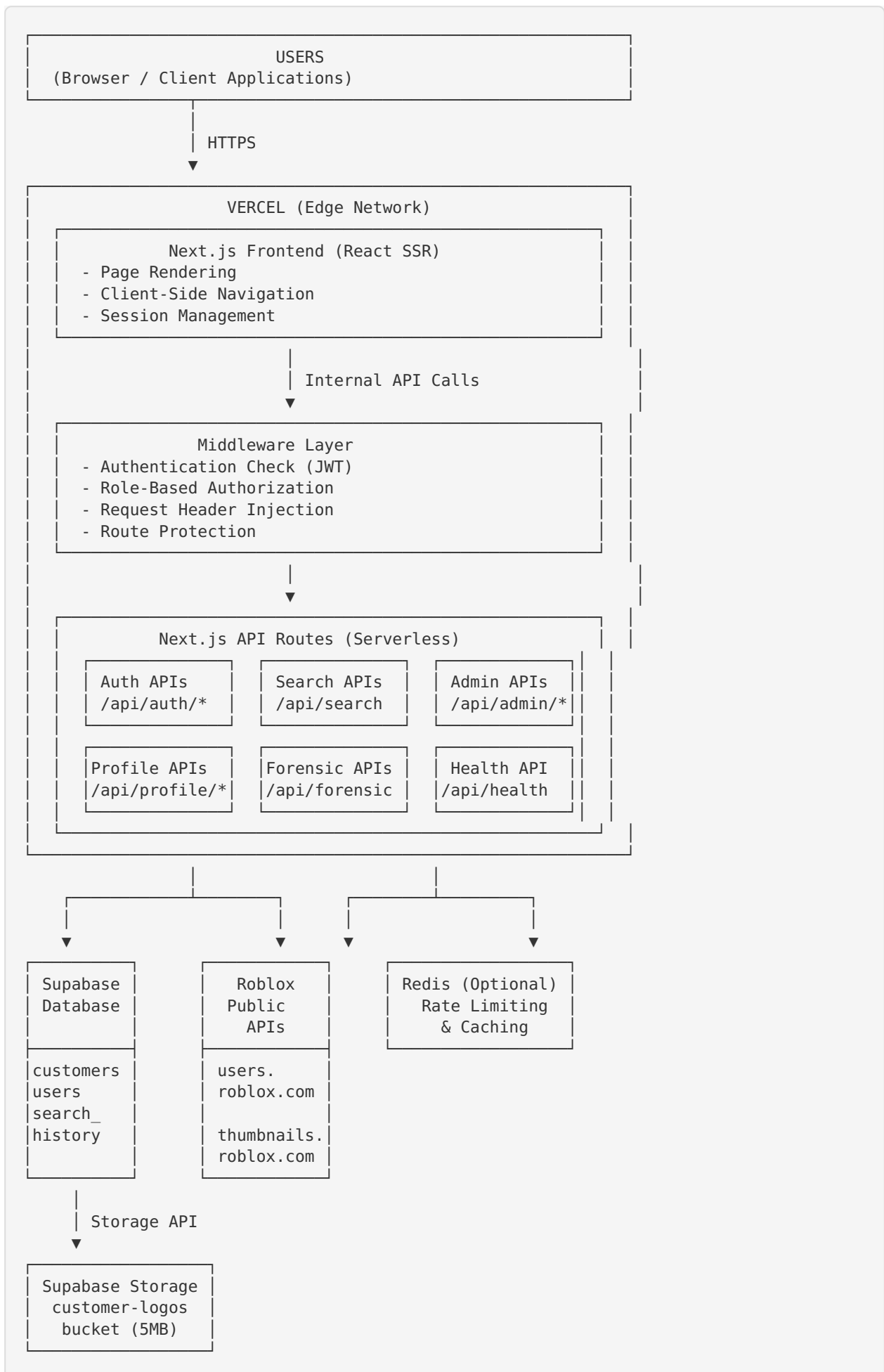
### 1.5 Application URL

- **Production:** <https://roblox-tool-ruddy.vercel.app/> (example)
  - **Local Development:** <http://localhost:3000>
-

## 2. Technical Architecture

---

### 2.1 System Architecture Diagram



## 2.2 Tech Stack with Versions

```
{
  "runtime": {
    "node": ">=18.0.0",
    "platform": "Vercel Serverless"
  },
  "frontend": {
    "next": "15.5.4",
    "react": "19.1.0",
    "react-dom": "19.1.0",
    "typescript": "^5"
  },
  "backend": {
    "next": "15.5.4 (API Routes)",
    "node-runtime": "Edge Runtime + Node.js Runtime"
  },
  "authentication": {
    "next-auth": "^4.24.11",
    "bcrypt": "^6.0.0"
  },
  "database": {
    "driver": "pg ^8.16.3",
    "type": "PostgreSQL",
    "hosting": "Supabase"
  },
  "storage": {
    "@supabase/supabase-js": "^2.76.0",
    "service": "Supabase Storage"
  },
  "styling": {
    "tailwindcss": "^3.4.17",
    "autoprefixer": "^10.4.21",
    "postcss": "^8.4.49"
  },
  "utilities": {
    "axios": "^1.12.2",
    "papaparse": "^5.5.3",
    "fast-levenshtein": "^3.0.0",
    "lucide-react": "^0.545.0"
  },
  "caching": {
    "ioredis": "^4.30.1 (optional)"
  },
  "devDependencies": {
    "tsx": "^4.20.6",
    "eslint": "^9",
    "eslint-config-next": "15.5.4"
  }
}
```

## 2.3 Data Flow

### User Search Flow

1. User enters search query in frontend  
↓
2. **Form** submission triggers search API call  
↓
3. Middleware **intercepts** request
  - ☐ **Validates** JWT session
  - ☐ Extracts user ID, customer ID, role
  - ☐ Injects headers: X-User-Id, X-Customer-Id, X-User-Role
 ↓
4. API Route receives request with headers
  - ☐ **Validates** search mode (exact/smart/displayName)
  - ☐ Checks cooldown status (**for** smart/displayName)
  - ☐ Determines search type from **input**
 ↓
5. Search **Logic** executes
  - ☐ Exact Match: Direct Roblox API lookup
  - ☐ Smart Match: Fuzzy search + AI ranking
  - ☐ Display Name: Fuzzy search + all results
 ↓
6. Rate Limiting & Protection
  - ☐ Circuit Breaker checks Roblox API health
  - ☐ Request Queue manages concurrency
  - ☐ Retry **logic** handles transient failures
 ↓
7. Results Processing
  - ☐ Cache check (**if** enabled)
  - ☐ Fetch from Roblox API **if** cache miss
  - ☐ Process results (ranking **for** smart match)
  - ☐ Store in cache **for** future requests
 ↓
8. Search **Logging** (async, non-blocking)
  - ☐ Extract search metadata
  - ☐ **Log to** search\_history **table**
  - ☐ Include user\_id, customer\_id, results
 ↓
9. Response **returned to** frontend
  - ☐ Display results **to** user
  - ☐ Show avatars (via thumbnail API)
  - ☐ Enable Deep **Context** / **Forensic** features

## Authentication Flow

1. User enters credentials **on** /auth/signin  
↓
2. **NextAuth** receives credentials  
↓
3. `authorize()` function executes
  - ☐ Query users **table** by username
  - ☐ Check **if** user exists
  - ☐ Validate `user.is_active === true`
  - ☐ Validate `customer.is_active === true` (**for** non-SUPER\_ADMIN)
  - ☐ `bcrypt.compare(password, user.password_hash)`
 ↓
4. **If** valid:
  - ☐ Update `users.last_login` timestamp
  - ☐ Create JWT **token** with user **data**:
    - ☐ `{`
    - ☐     `id, username, role,`
    - ☐     `customerId, customerName,`
    - ☐     `rememberMe, exp (expiration)`
    - ☐ `}`
  - ☐ Set httpOnly cookie with JWT
 ↓
5. Redirect **to** home page (/)  
↓
6. Middleware protects all subsequent requests
  - ☐ **Verify** JWT **on** every request
  - ☐ Check **token** **expiration**
  - ☐ Inject user info **into** request headers

## 2.4 Deployment Architecture

- GitHub Repository (main branch)
- ↓
- Push/Merge
- ↓
- Vercel Auto-Deploy Trigger
- ↓
- Build Process:
- ☐ Install dependencies (`npm install`)
  - ☐ Run type checks (`tsc`)
  - ☐ Run ESLint
  - ☐ Build Next.js app (`npm run build`)
  - ☐ Optimize assets
- ↓
- Deploy to Vercel Edge Network:
- ☐ Deploy **static** assets to CDN
  - ☐ Deploy serverless functions (API routes)
  - ☐ Configure environment variables
  - ☐ Set up custom domain
- ↓
- Production Environment:
- ☐ Frontend: Vercel Edge (Global CDN)
  - ☐ API Routes: Vercel Serverless (AWS Lambda)
  - ☐ Database: Supabase PostgreSQL (AWS US-East-1)
  - ☐ Storage: Supabase Storage (AWS S3-compatible)

## 3. Database Schema

### 3.1 Complete Schema Overview

The application uses **PostgreSQL** hosted on **Supabase** with 4 main tables, 2 views, and supporting indexes/triggers.

#### Tables

1. **customers** - Customer/tenant organizations
2. **users** - User accounts with role-based access
3. **search\_history** - Logs of all searches
4. **audit\_logs** - Admin action tracking (optional, for future use)

#### Views

1. **customer\_stats** - Aggregated customer statistics
2. **user\_activity** - User activity summary

### 3.2 Customers Table

**Purpose:** Stores customer organizations (tenants) that use the tool.

```
CREATE TABLE customers (  
  id SERIAL PRIMARY KEY,  
  name VARCHAR(255) NOT NULL UNIQUE,  
  is_active BOOLEAN NOT NULL DEFAULT true,  
  created_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,  
  updated_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,  
  
  -- Additional metadata  
  contact_email VARCHAR(255),  
  max_users INTEGER DEFAULT 5,  
  logo_url TEXT, -- Added in migration 003  
  
  -- Constraints  
  CONSTRAINT customers_name_check CHECK (name <> '')  
);  
  
-- Indexes  
CREATE INDEX idx_customers_is_active ON customers(is_active);  
CREATE INDEX idx_customers_name ON customers(name);
```

**Fields Explained:**



Field	Type	Purpose	Nullable	Default
id	SERIAL	Unique identifier	NO	AUTO
name	VARCHAR(255)	Customer organization name	NO	-
is_active	BOOLEAN	Whether customer account is active	NO	true
created_at	TIMESTAMPTZ	Account creation date	NO	NOW()
updated_at	TIMESTAMPTZ	Last update timestamp	NO	NOW()
contact_email	VARCHAR(255)	Customer contact email	YES	NULL
max_users	INTEGER	User limit (future feature)	YES	5
logo_url	TEXT	Supabase Storage URL for logo	YES	NULL

**Business Rules:**

- Customer `name` must be unique across all customers
- Only active customers ( `is_active = true` ) can have users login
- Deactivating a customer prevents all their users from accessing the tool
- `updated_at` automatically updates via trigger

### 3.3 Users Table

**Purpose:** Stores user accounts with role-based access control.

```

CREATE TABLE users (
  id SERIAL PRIMARY KEY,
  username VARCHAR(100) NOT NULL UNIQUE,
  password_hash VARCHAR(255) NOT NULL,
  role VARCHAR(50) NOT NULL CHECK (role IN ('SUPER_ADMIN', 'CUSTOMER_ADMIN', 'CUSTOMER_USER')),
  customer_id INTEGER REFERENCES customers(id) ON DELETE CASCADE,

  -- User metadata
  email VARCHAR(255),
  full_name VARCHAR(255),
  is_active BOOLEAN NOT NULL DEFAULT true,

  -- Timestamps
  created_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,
  last_login TIMESTAMP WITH TIME ZONE,

  -- Constraints
  CONSTRAINT users_username_check CHECK (username <> ''),
  CONSTRAINT users_password_check CHECK (password_hash <> ''),
  CONSTRAINT users_customer_role_check CHECK (
    (role = 'SUPER_ADMIN' AND customer_id IS NULL) OR
    (role IN ('CUSTOMER_ADMIN', 'CUSTOMER_USER') AND customer_id IS NOT NULL)
  )
);

-- Indexes
CREATE INDEX idx_users_username ON users(username);
CREATE INDEX idx_users_customer_id ON users(customer_id);
CREATE INDEX idx_users_role ON users(role);
CREATE INDEX idx_users_is_active ON users(is_active);

```

### Fields Explained:

Field	Type	Purpose	Nullable	Default
id	SERIAL	Unique identifier	NO	AUTO
username	VARCHAR(100)	Login username	NO	-
password_hash	VARCHAR(255)	bcrypt hashed password	NO	-
role	VARCHAR(50)	User role (see roles below)	NO	-
customer_id	INTEGER	FK to customers	YES*	NULL
email	VARCHAR(255)	User email address	YES	NULL
full_name	VARCHAR(255)	User's full name	YES	NULL
is_active	BOOLEAN	Account active status	NO	true
created_at	TIMESTAMPTZ	Account creation date	NO	NOW()
updated_at	TIMESTAMPTZ	Last update timestamp	NO	NOW()
last_login	TIMESTAMPTZ	Last successful login	YES	NULL

#### Role Definitions:

Role	customer_id	Description
SUPER_ADMIN	NULL (required)	Platform owner, full access to admin dashboard
CUSTOMER_ADMIN	NOT NULL (required)	Customer admin, can use search tool
CUSTOMER_USER	NOT NULL (required)	Regular user, can use search tool (future)

#### Business Rules:

- username must be unique across ALL users (not just per customer)
- password\_hash is generated with bcrypt (salt rounds: 10)
- SUPER\_ADMIN users CANNOT have a customer\_id (enforced by constraint)
- CUSTOMER\_ADMIN and CUSTOMER\_USER MUST have a customer\_id

- Inactive users ( `is_active = false` ) cannot login
- `last_login` updated on every successful authentication

### 3.4 Search History Table

**Purpose:** Logs all searches performed by authenticated users for auditing and analytics.

```
CREATE TABLE search_history (
  id SERIAL PRIMARY KEY,
  user_id INTEGER NOT NULL REFERENCES users(id) ON DELETE CASCADE,
  customer_id INTEGER REFERENCES customers(id) ON DELETE CASCADE,

  -- Search details
  search_type VARCHAR(50) NOT NULL CHECK (search_type IN ('username',
'displayName', 'userId', 'url', 'exact', 'smart')),
  search_mode VARCHAR(50) NOT NULL DEFAULT 'exact' CHECK (search_mode IN ('exact', '
smart', 'displayName')),
  search_query VARCHAR(500) NOT NULL,

  -- Roblox user details (if found)
  roblox_username VARCHAR(255),
  roblox_user_id BIGINT,
  roblox_display_name VARCHAR(255),
  has_verified_badge BOOLEAN,

  -- Result metadata
  result_data JSONB,
  result_count INTEGER DEFAULT 0,
  result_status VARCHAR(50) CHECK (result_status IN ('success', 'no_results', 'er-
ror')),
  error_message TEXT,

  -- Performance metrics
  response_time_ms INTEGER,

  -- Timestamp
  searched_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,

  -- Constraints
  CONSTRAINT search_history_user_check CHECK (user_id IS NOT NULL)
);

-- Indexes
CREATE INDEX idx_search_history_user_id ON search_history(user_id);
CREATE INDEX idx_search_history_customer_id ON search_history(customer_id) WHERE cus-
tomer_id IS NOT NULL;
CREATE INDEX idx_search_history_searched_at ON search_history(searched_at DESC);
CREATE INDEX idx_search_history_roblox_user_id ON search_history(roblox_user_id);
CREATE INDEX idx_search_history_search_type ON search_history(search_type);
CREATE INDEX idx_search_history_search_mode ON search_history(search_mode);
```

**Fields Explained:**

Field	Type	Purpose
id	SERIAL	Unique search record ID
user_id	INTEGER	User who performed search
customer_id	INTEGER	Customer user belongs to (NULL for SUPER_ADMIN)
search_type	VARCHAR(50)	Type of input (username/ userId/displayName/url)
search_mode	VARCHAR(50)	Search mode used (exact/ smart/displayName)
search_query	VARCHAR(500)	Original search query text
roblox_username	VARCHAR(255)	Found username (if any)
roblox_user_id	BIGINT	Found Roblox user ID (if any)
roblox_display_name	VARCHAR(255)	Found display name (if any)
has_verified_badge	BOOLEAN	Whether user has verified badge
result_data	JSONB	Full API response data
result_count	INTEGER	Number of results returned
result_status	VARCHAR(50)	success/no_results/error
error_message	TEXT	Error details if failed
response_time_ms	INTEGER	API response time in milli-seconds
searched_at	TIMESTAMPTZ	Timestamp of search

**Important Notes:**

- `customer_id` can be NULL for SUPER\_ADMIN searches (migration 001)
- `search_mode` added in migration 002 (tracks which search mode was used)
- `result_data` stores full JSON for detailed analysis
- Indexed on `searched_at` DESC for efficient recent search queries

### 3.5 Audit Logs Table (Optional)

**Purpose:** Track admin actions and system events for security and compliance (not fully implemented yet).

```

CREATE TABLE audit_logs (
  id SERIAL PRIMARY KEY,
  user_id INTEGER REFERENCES users(id) ON DELETE SET NULL,
  customer_id INTEGER REFERENCES customers(id) ON DELETE SET NULL,

  -- Action details
  action VARCHAR(100) NOT NULL,
  entity_type VARCHAR(50),
  entity_id INTEGER,

  -- Change details
  old_values JSONB,
  new_values JSONB,

  -- Request metadata
  ip_address INET,
  user_agent TEXT,

  -- Timestamp
  created_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP
);

-- Indexes
CREATE INDEX idx_audit_logs_user_id ON audit_logs(user_id);
CREATE INDEX idx_audit_logs_customer_id ON audit_logs(customer_id);
CREATE INDEX idx_audit_logs_created_at ON audit_logs(created_at DESC);
CREATE INDEX idx_audit_logs_action ON audit_logs(action);

```

**Status:** Table exists but not actively used yet. Future implementation for tracking:

- Customer creation/deactivation
- User creation/deletion
- Password changes
- Role changes
- Logo uploads

## 3.6 Database Views

### customer\_stats View

**Purpose:** Aggregated statistics for each customer combining data from multiple tables.

```

CREATE OR REPLACE VIEW customer_stats AS
SELECT
  c.id,
  c.name,
  c.is_active,
  c.created_at,
  COUNT(DISTINCT u.id) as total_users,
  COUNT(DISTINCT CASE WHEN u.is_active THEN u.id END) as active_users,
  COUNT(sh.id) as total_searches,
  MAX(sh.searched_at) as last_search_at,
  MAX(u.last_login) as last_login_at
FROM customers c
LEFT JOIN users u ON c.id = u.customer_id
LEFT JOIN search_history sh ON c.id = sh.customer_id
GROUP BY c.id, c.name, c.is_active, c.created_at;

```

**Used By:** Admin dashboard to display customer statistics efficiently.

## user\_activity View

**Purpose:** Summary of user activity including search counts.

```
CREATE OR REPLACE VIEW user_activity AS
SELECT
    u.id,
    u.username,
    u.role,
    u.customer_id,
    c.name as customer_name,
    u.is_active,
    u.last_login,
    COUNT(sh.id) as total_searches,
    MAX(sh.searched_at) as last_search_at,
    COUNT(CASE WHEN sh.searched_at > CURRENT_TIMESTAMP - INTERVAL '7 days' THEN 1
END) as searches_last_7_days,
    COUNT(CASE WHEN sh.searched_at > CURRENT_TIMESTAMP - INTERVAL '30 days' THEN 1
END) as searches_last_30_days
FROM users u
LEFT JOIN customers c ON u.customer_id = c.id
LEFT JOIN search_history sh ON u.id = sh.user_id
GROUP BY u.id, u.username, u.role, u.customer_id, c.name, u.is_active, u.last_login;
```

**Used By:** Future user management features and analytics.

## 3.7 Database Triggers

### Update updated\_at Column

**Purpose:** Automatically update the `updated_at` timestamp on row updates.

```
CREATE OR REPLACE FUNCTION update_updated_at_column()
RETURNS TRIGGER AS $$
BEGIN
    NEW.updated_at = CURRENT_TIMESTAMP;
    RETURN NEW;
END;
$$ language 'plpgsql';

-- Apply to customers table
CREATE TRIGGER update_customers_updated_at
BEFORE UPDATE ON customers
FOR EACH ROW EXECUTE FUNCTION update_updated_at_column();

-- Apply to users table
CREATE TRIGGER update_users_updated_at
BEFORE UPDATE ON users
FOR EACH ROW EXECUTE FUNCTION update_updated_at_column();
```

## 3.8 Sample SQL Queries

### Get All Active Customers with Stats

```
SELECT * FROM customer_stats
WHERE is_active = true
ORDER BY created_at DESC;
```

## Get Search History for a Customer

```
SELECT
  sh.id,
  sh.search_query,
  sh.search_mode,
  sh.result_status,
  sh.searched_at,
  u.username as searched_by
FROM search_history sh
JOIN users u ON sh.user_id = u.id
WHERE sh.customer_id = 5
ORDER BY sh.searched_at DESC
LIMIT 100;
```

## Get Most Active Users

```
SELECT
  username,
  customer_name,
  total_searches,
  last_search_at
FROM user_activity
ORDER BY total_searches DESC
LIMIT 10;
```

## Create a Customer with Admin User (Transaction)

```
BEGIN;

-- Insert customer
INSERT INTO customers (name, is_active, contact_email)
VALUES ('ACME Corp', true, 'admin@acme.com')
RETURNING id; -- Note the ID

-- Insert admin user (replace 123 with customer ID from above)
INSERT INTO users (
  username, password_hash, role, customer_id,
  email, full_name, is_active
) VALUES (
  'acme_admin',
  '$2b$10$YourBcryptHashHere',
  'CUSTOMER_ADMIN',
  123,
  'admin@acme.com',
  'ACME Administrator',
  true
);

COMMIT;
```

---

# 4. Authentication & Authorization

## 4.1 NextAuth Configuration

Location: `src/app/lib/auth.ts`



The application uses **NextAuth.js** with a custom credentials provider for database-backed authentication.

## Authentication Provider

```
providers: [
  CredentialsProvider({
    name: 'Credentials',
    credentials: {
      username: { label: "Username", type: "text" },
      password: { label: "Password", type: "password" },
      rememberMe: { label: "Remember Me", type: "text" }
    },
    async authorize(credentials) {
      // 1. Validate credentials exist
      if (!credentials?.username || !credentials?.password) {
        return null;
      }

      // 2. Fetch user from database
      const user = await getUserByUsername(credentials.username);
      if (!user) return null;

      // 3. Check user is active
      if (!user.is_active) return null;

      // 4. Check customer is active (for non-SUPER_ADMIN)
      if (user.role !== 'SUPER_ADMIN' && !user.customer_is_active) {
        return null;
      }

      // 5. Verify password with bcrypt
      const isValid = await bcrypt.compare(
        credentials.password,
        user.password_hash
      );
      if (!isValid) return null;

      // 6. Update last login
      await updateUserLastLogin(user.id);

      // 7. Return user object for session
      return {
        id: user.id.toString(),
        name: user.full_name || user.username,
        email: user.email,
        username: user.username,
        role: user.role,
        customerId: user.customer_id?.toString(),
        customerName: user.customer_name,
        rememberMe: credentials.rememberMe === 'true',
      };
    }
  })
]
```

## 4.2 Session Management

### JWT Configuration

```
session: {
  strategy: 'jwt',
  maxAge: 30 * 24 * 60 * 60, // 30 days (max possible)
},

jwt: {
  maxAge: 30 * 24 * 60 * 60, // 30 days (max possible)
},
```

### JWT Callback - Dynamic Expiration

```
async jwt({ token, user }) {
  if (user) {
    token.id = user.id;
    token.username = user.username;
    token.role = user.role;
    token.customerId = user.customerId;
    token.customerName = user.customerName;
    token.rememberMe = user.rememberMe;

    // Set token expiration based on remember me preference
    const now = Date.now();
    if (user.rememberMe) {
      // Remember me: 30 days
      token.exp = Math.floor(now / 1000) + (30 * 24 * 60 * 60);
    } else {
      // Don't remember: 2 hours
      token.exp = Math.floor(now / 1000) + (2 * 60 * 60);
    }
  }
  return token;
}
```

#### Key Features:

- **Remember Me Checked:** Session persists for 30 days
- **Remember Me Unchecked:** Session expires after 2 hours
- JWT token expiration is dynamic based on user preference

### Session Callback

```
async session({ session, token }) {
  if (session.user) {
    session.user.id = token.id as string;
    session.user.username = token.username as string;
    session.user.role = token.role as string;
    session.user.customerId = token.customerId as string | undefined;
    session.user.customerName = token.customerName as string | undefined;
  }
  return session;
}
```

## 4.3 Middleware - Authentication & Authorization

**Location:** src/middleware.ts

Middleware runs on **every request** (except static files) to enforce authentication and authorization.

```
export async function middleware(request: NextRequest) {
  const { pathname } = request.nextUrl;

  // Get JWT token from request
  const token = await getToken({
    req: request,
    secret: process.env.NEXTAUTH_SECRET,
  });

  // Public paths (no auth required)
  const publicPaths = [
    '/auth/signin',
    '/api/auth',
    '/api/health',
    '/_next',
    '/favicon.ico',
  ];

  const isPublicPath = publicPaths.some(path => pathname.startsWith(path));

  // Redirect to login if not authenticated and accessing protected route
  if (!isPublicPath && !token) {
    const signInUrl = new URL('/auth/signin', request.url);
    signInUrl.searchParams.set('callbackUrl', pathname);
    return NextResponse.redirect(signInUrl);
  }

  // Redirect to home if authenticated and accessing login page
  if (pathname === '/auth/signin' && token) {
    return NextResponse.redirect(new URL('/', request.url));
  }

  // Check admin routes - only SUPER_ADMIN can access
  if (pathname.startsWith('/admin') && token?.role !== 'SUPER_ADMIN') {
    return NextResponse.redirect(new URL('/', request.url));
  }

  // Check admin API routes - only SUPER_ADMIN can access
  if (pathname.startsWith('/api/admin') && token?.role !== 'SUPER_ADMIN') {
    return NextResponse.json(
      { error: 'Unauthorized. Admin access required.' },
      { status: 403 }
    );
  }

  const response = NextResponse.next();

  // Add user info to headers for API routes
  if (token && pathname.startsWith('/api/')) {
    response.headers.set('X-User-Id', token.id as string);
    response.headers.set('X-User-Role', token.role as string);
    response.headers.set('X-Customer-Id', token.customerId as string || 'null');
  }

  return response;
}
```

**Middleware Flow:**

```

Request Received
↓
Is path public? → Yes → Allow access
↓ No
Has valid JWT token? → No → Redirect to /auth/signin
↓ Yes
Is path /auth/signin? → Yes → Redirect to /
↓ No
Is path /admin or /api/admin? → Yes → Check role === 'SUPER_ADMIN'
↓                                     → No → Deny (403 or redirect)
↓                                     → Yes → Allow
Is path /api/*? → Yes → Inject headers (X-User-Id, X-Customer-Id, X-User-Role)
↓
Allow request

```

## 4.4 User Roles & Permissions

### SUPER\_ADMIN

#### Characteristics:

- `customer_id` is NULL (not associated with any customer)
- Created via database initialization script
- Only one SUPER\_ADMIN typically exists (platform owner)

#### Permissions:

- ☒ Access admin dashboard ( `/admin` )
- ☒ View all customers and statistics
- ☒ Create new customers
- ☒ Activate/deactivate customers
- ☒ View search history for all customers
- ☒ Access all admin API endpoints ( `/api/admin/*` )
- ☒ Use search tool (results not tied to a specific customer)

#### Cannot Do:



- ☒ Be assigned to a customer (enforced by database constraint)

### CUSTOMER\_ADMIN

#### Characteristics:

- `customer_id` is NOT NULL (must belong to a customer)
- Created when a new customer is onboarded
- Each customer has at least one CUSTOMER\_ADMIN

#### Permissions:

- ☒ Use search tool with customer-scoped data
- ☒ View own profile
-  View own search history (future feature)
-  Manage users within customer (future feature)

#### Cannot Do:

- ☒ Access admin dashboard
- ☒ View other customers' data
- ☒ Create or manage customers
- ☒ View search history of other customers

## CUSTOMER\_USER (Future Role)

### Characteristics:

- `customer_id` is NOT NULL
- Regular user within a customer organization
- Lower privilege than CUSTOMER\_ADMIN

### Planned Permissions:

- ☒ Use search tool with customer-scoped data
- ☒ View own profile
- ☒ View own search history

### Cannot Do:

- ☒ Access admin dashboard
- ☒ Manage users
- ☒ View other users' search history

## 4.5 Password Hashing

**Algorithm:** bcrypt

**Salt Rounds:** 10

**Library:** `bcrypt` v6.0.0

### Hashing Process

```
import * as bcrypt from 'bcrypt';

// Hash password (during user creation)
const passwordHash = await bcrypt.hash(plainPassword, 10);

// Verify password (during login)
const isValid = await bcrypt.compare(plainPassword, storedHash);
```

### Security Features:

- Each password gets a unique salt (auto-generated by bcrypt)
- Computationally expensive (protects against brute force)
- Industry-standard hashing algorithm
- Secure against rainbow table attacks

## 4.6 Remember Me Feature

**Implemented:** Yes (October 2025)

**Files:** `src/app/auth/signin/page.tsx`, `src/app/lib/auth.ts`

### How It Works

#### 1. Checkbox on Login Page

- Labeled "Remember me for 30 days"
- Stored in localStorage on form submission

#### 2. Dynamic Session Duration

- **Checked:** JWT expires in 30 days
- **Unchecked:** JWT expires in 2 hours

#### 3. Cookie Configuration

- `httpOnly` : true (prevents XSS)

- `secure` : true in production (HTTPS only)
- `sameSite` : 'lax' (CSRF protection)
- `maxAge` : Dynamic based on remember me

## Security Considerations

### Safe:

- ☒ JWT encrypted with NEXTAUTH\_SECRET
- ☒ httpOnly cookies prevent JavaScript access
- ☒ Tokens expire automatically
- ☒ No sensitive data in JWT (only IDs and role)

### Best Practices:

- User can logout at any time to invalidate session
- Changing password doesn't invalidate existing sessions (limitation of JWT)
- Token expiration enforced by NextAuth

## 5. Features - Detailed Breakdown

### 5.1 Search Modes

The application offers three distinct search modes, each with unique behavior and use cases.

#### 5.1.1 Exact Match Mode

**Purpose:** Fast, direct lookup of Roblox users when you know the exact username or user ID.

#### How It Works:

1. User enters exact username (e.g., "JohnDoe") or user ID (e.g., "5533914010")
2. Application makes direct API call to Roblox:
  - Username: `POST /v1/usernames/users` (request body)
  - User ID: `GET /v1/users/{userId}`
3. Returns single result if found, or no results

#### Features:

- ☒ **No cooldown** - can search repeatedly without waiting
- ☒ **Fastest** - direct API calls, no processing
- ☒ **Most accurate** - exact matches only
- ☒ **No fuzzy matching** - must be exact spelling

#### Use Cases:

- Verifying a known username
- Looking up user by ID from a URL
- Quick verification when exact spelling is known

**API Endpoint:** `/api/roblox` (POST for username, GET for userId)

#### Example Flow:

```

User enters: "JohnDoe"
↓
POST https://users.roblox.com/v1/usernames/users
Body: { usernames: ["JohnDoe"], excludeBannedUsers: true }
↓
Returns: {
  data: [{
    requestedUsername: "JohnDoe",
    id: 5533914010,
    name: "JohnDoe",
    displayName: "John D.",
    hasVerifiedBadge: true
  }]
}

```






### 5.1.2 Smart Match Mode

**Purpose:** AI-powered fuzzy search that ranks results by confidence when you don't know the exact spelling.

#### How It Works:

1. User enters approximate username (e.g., "john doe" or "johndoe123")
2. Application uses Roblox search API: `GET /v1/users/search?keyword={query}`
3. Returns up to 10 candidates
4. **AI Ranking Algorithm** processes results:
  - Name similarity (40% weight)
  - Account signals (25% weight) - verified badge, account age
  - Keyword hits (15% weight) - matches in bio
  - Group overlap (10% weight)
  - Profile completeness (10% weight)
5. Each candidate gets a confidence score (0-100%)
6. Results sorted by confidence descending

#### Features:

-  **Fuzzy matching** - handles typos, variations, partial names
-  **AI ranking** - most likely match appears first
-  **Confidence scores** - see why each result was ranked
-  **Up to 10 suggestions** - multiple options if unsure
-  **30-second cooldown** - prevents API abuse

#### Ranking Algorithm Details:

```

// Weights (configurable)
const weights = {
  nameSimilarity: 0.40,
  accountSignals: 0.25,
  keywordHits: 0.15,
  groupOverlap: 0.10,
  profileCompleteness: 0.10
};

// Name Similarity Calculation
function calculateNameSimilarity(query, candidate) {
  // Exact match: 100%
  if (query.toLowerCase() === candidate.displayName.toLowerCase()) return 1.0;

  // Starts with: 85%
  if (candidate.displayName.toLowerCase().startsWith(query.toLowerCase())) return
0.85;

  // Contains: 70%
  if (candidate.displayName.toLowerCase().includes(query.toLowerCase())) return 0.70;

  // Jaro-Winkler + Levenshtein distance
  // Returns 0.0 - 1.0 based on string similarity
}

// Account Signals Calculation
function calculateAccountSignals(candidate) {
  let score = 0;

  if (candidate.hasVerifiedBadge) score += 0.4;

  const accountAge = getAccountAgeInDays(candidate.created);
  if (accountAge > 365 * 3) score += 0.3; // 3+ years
  else if (accountAge > 365) score += 0.2; // 1+ year
  else if (accountAge > 90) score += 0.1; // 90+ days

  if (candidate.description?.length > 10) score += 0.3;

  return Math.min(score, 1.0);
}

// Final Confidence Score
confidence = Math.round(
  (nameSimilarity * 0.40) +
  (accountSignals * 0.25) +
  (keywordHits * 0.15) +
  (groupOverlap * 0.10) +
  (profileCompleteness * 0.10) * 100
);

```

**Use Cases:**

- Searching for a user when you don't know exact spelling
- Finding users with common names
- Discovering potential account variations
- Investigating misspelled or similar usernames

**API Endpoint:** `/api/search?keyword={query}&searchMode=smart`

**Cooldown Management:**

- Implemented using React hook: `useCooldown`



- Stored in localStorage: `smart_search_cooldown_expires`
- Duration: 30 seconds
- UI button disabled during cooldown with countdown timer

#### Example Flow:

```

User enters: "john d"
↓
GET /api/search?keyword=john%20d&searchMode=smart
↓
Roblox API returns 10 users
↓
AI Ranking processes:
  1. JohnDoe (95% confidence) - Exact display name match + verified
  2. JohnD123 (87% confidence) - Starts with query + old account
  3. John_Doe (82% confidence) - Contains query + complete profile
  4. JohnnyDoe (75% confidence) - Similar name
  ... (up to 10 results)
↓
Display ranked suggestions with confidence badges

```



### 5.1.3 Display Name Mode

**Purpose:** Search by display name and show ALL matching users (not just top ranked).

#### How It Works:

1. User enters display name (e.g., "Alex")
2. Application uses Roblox search API: `GET /v1/users/search?keyword={query}&limit=20`
3. Returns up to 20 users whose display names match
4. No ranking or filtering - shows all results
5. User can scroll through and manually identify correct account

#### Features:

-  **Shows all matches** - no AI filtering
-  **Up to 20 results** - more than Smart Match
-  **Simpler results** - just list of matching users
-  **Visual avatars** - easier to identify correct account
-  **30-second cooldown** - prevents API abuse

#### Use Cases:

- Searching by display name (not username)
- Finding all accounts with a specific display name
- Manual verification when multiple people share a name
- Investigating display name impersonation

**API Endpoint:** `/api/search?keyword={query}&searchMode=displayName`

#### Cooldown Management:

- Same as Smart Match (separate cooldown timer)
- Stored in localStorage: `display_name_search_cooldown_expires`

#### Difference from Smart Match:

Feature	Smart Match	Display Name
Results	Up to 10, ranked by confidence	Up to 20, no ranking
Processing	AI ranking algorithm	Simple list
Confidence Scores	Yes	No
Use Case	Find most likely match	See all matching display names

### Example Flow:

```

User enters: "Alex"
↓
GET /api/search?keyword=Alex&searchMode=displayName
↓
Roblox API returns 20 users with display name "Alex"
↓
Display all 20 users:
  - Alex (username: alex123)
  - Alex (username: alexGamer)
  - Alex (username: TheRealAlex)
  - Alex (username: alex_2023)
  ... (all 20 results, no ranking)

```

## 5.2 Forensic Mode

**Purpose:** Generate tamper-proof evidence reports with cryptographic hashing for legal/compliance use.

**Status:** Implemented

**Location:** Frontend toggle + `/api/forensic/report` endpoint

**Files:**

- `src/app/components/ForensicMode.tsx`
- `src/app/lib/forensic.ts`
- `src/app/api/forensic/report/route.ts`

### How It Works

#### 1. Enable Forensic Mode

- Toggle switch at top of page
- When enabled, search results include forensic snapshot

#### 2. Data Collection

- User profile data
- Friends list
- Groups memberships
- Account statistics
- Search timestamp

#### 3. Evidence Processing

```typescript

```

// 1. Collect data snapshot
const snapshot = {
  user: { id, username, displayName, created, ... },
  counts: { friends, followers, following },
  groups: [ { name, role, memberCount } ],
  profile: { description, badges, ... }
};

// 2. Generate SHA-256 hash
const dataString = JSON.stringify(snapshot, Object.keys(snapshot).sort());
const hash = await crypto.subtle.digest('SHA-256', dataString);

// 3. Create forensic report
const report = {
  meta: {
    reportId: crypto.randomUUID(),
    createdAt: new Date().toISOString(),
    createdBy: session.user.email,
    appVersion: '1.0.0',
    forensicMode: true
  },
  query: { input, mode },
  snapshot,
  hash: { algo: 'SHA-256', value: hashHex },
  chainOfCustody: [
    { event: 'generated', actor: user.email, at: timestamp }
  ]
};
...

```

### 1. Export Options

- PDF export (via print dialog)
- JSON download
- HTML report view

## Forensic Report Structure

```
{
  "meta": {
    "reportId": "550e8400-e29b-41d4-a716-446655440000",
    "createdAt": "2025-10-21T14:30:00.000Z",
    "createdBy": "investigator@law.gov",
    "appVersion": "1.0.0",
    "forensicMode": true,
    "caseId": "CASE-2025-001" // Optional
  },
  "query": {
    "input": "JohnDoe",
    "mode": "username"
  },
  "sources": [
    { "name": "roblox.users", "fetchedAt": "2025-10-21T14:30:00.000Z" },
    { "name": "roblox.groups", "fetchedAt": "2025-10-21T14:30:01.000Z" },
    { "name": "roblox.friends", "fetchedAt": "2025-10-21T14:30:02.000Z" }
  ],
  "snapshot": {
    "user": { /* full user data */ },
    "counts": { /* friends/followers */ },
    "groups": [ /* group memberships */ ],
    "profile": { /* profile details */ }
  },
  "hash": {
    "algo": "SHA-256",
    "value": "a3c5f9d2e8b1... (64 hex chars)"
  },
  "chainOfCustody": [
    {
      "event": "generated",
      "actor": "investigator@law.gov",
      "at": "2025-10-21T14:30:00.000Z"
    }
  ]
}
```

## Security Features

1. **Cryptographic Hash:** SHA-256 ensures data integrity
2. **Timestamp:** Proves when evidence was collected
3. **Chain of Custody:** Tracks who accessed/exported the report
4. **Immutable:** Any change to snapshot invalidates hash
5. **Verifiable:** Hash can be recomputed to verify authenticity

## Use Cases

- Law enforcement investigations
- Content moderation evidence
- Legal discovery requests
- Compliance audits
- Digital forensics

## PDF Report Includes

- Report header with metadata
- Query information

- User profile snapshot
- Friends and groups lists
- Detected flags (if any)
- Cryptographic hash
- Chain of custody
- Legal disclaimer

## 5.3 Deep Context Profile Viewer

**Purpose:** View comprehensive user profiles with all available Roblox data in a modal overlay.

**Component:** `src/app/components/DeepContext.tsx`

**Trigger:** “View Full Profile” button on search results

### Data Displayed

#### 1. Overview Tab

- Username and display name
- User ID
- Account creation date
- Description/bio
- Verified badge status
- Friends count, followers, following
- Account age

#### 2. Groups & Roles Tab

- All groups the user belongs to
- Role in each group
- Group member counts
- Group descriptions

#### 3. Activity Tab

- Recent activity (if available)
- Last online (if available)

#### 4. Linked Mentions Tab

- External social media links found in bio
- Discord usernames
- YouTube channels
- Twitter handles

#### 5. Friends List Tab (up to 27 friends)

- Friend usernames
- “View” button to inspect each friend

## API Endpoints Used

```
// Fetch full profile
GET /api/profile/{userId}

// Returns:
{
  user: { /* basic info */ },
  counts: { friends, followers, following },
  groups: [ /* group memberships */ ],
  friends: [ /* friend list */ ]
}

// Fetch friends details
GET /api/friends/{userId}

// Returns: Array of friend objects
```

## Features

- **Copy Summary:** Button to copy profile summary to clipboard
- **Export PDF:** Export profile to PDF report
- **Navigate to Friends:** Click any friend to open their profile
- **Avatar Display:** Show user avatar (via `/api/thumbnail` )
- **Responsive Design:** Works on mobile and desktop

## Deep Context Flow

```
User clicks "View Full Profile"
↓
Open modal with loading state
↓
Fetch /api/profile/{userId}
↓
Display Overview tab with user data
↓
User can:
- Switch tabs to view different data
- Click friends to view their profiles (recursive)
- Export to PDF
- Copy summary text
- Close modal
```

## 5.4 Batch CSV Upload

**Purpose:** Verify multiple Roblox users at once by uploading a CSV file.

**Location:** Main page, below search form

**Library:** papaparse v5.5.3

### How It Works

#### 1. Upload CSV File

- Click "Choose File" input
- Select CSV file (no headers required)
- File format: One username per row

## 2. Processing

```
typescript
Papa.parse(file, {
  complete: (results) => {
    const usernames = results.data.flat().filter(Boolean);
    // Process each username sequentially
    for (const username of usernames) {
      // Perform search
      // Log result
    }
  },
  header: false
});
```

## 3. Results Display

- Table with columns: Input, Status, Details
- Status: Verified, Not Found, Error, Suggestions
- Color-coded rows (green = verified, red = error, yellow = suggestions)

## 4. Export Results

- Click "Export CSV" button
- Downloads results as CSV file
- Columns: Input, Status, Details

## CSV Format







### Input File:

```
JohnDoe
5533914010
https://www.roblox.com/users/12345/profile
Jane_Doe
```

### Output File:

```
Input,Status,Details
JohnDoe,Verified,"Username: JohnDoe, Display Name: John D., ID: 5533914010, Verified
Badge: Yes"
5533914010,Verified,"Username: TestUser, Display Name: Test, ID: 5533914010, Verified
Badge: No"
https://www.roblox.com/users/12345/profile,Verified,"Username: Player123, Display
Name: Player, ID: 12345, Verified Badge: No"
Jane_Doe,Not Found,"No exact match found"
```

## Features

-  **No headers required** in CSV
-  **Multiple formats supported:** username, user ID, URL
-  **Sequential processing:** One at a time to avoid rate limits
-  **Progress indicator:** Shows processing status
-  **Export results:** Download CSV with findings
-  **Error handling:** Gracefully handles invalid entries

## Limitations

- Maximum recommended: 100 entries (to avoid long processing times)
- No batch Smart Match (only Exact Match mode)
- Processing time: ~1-2 seconds per entry

## 5.5 Admin Dashboard

**Purpose:** Super Admin interface for managing customers, users, and viewing analytics.

**Route:** `/admin` (SUPER\_ADMIN only)

**Components:**

- `src/app/admin/page.tsx` - Main dashboard layout
- `src/app/admin/components/DashboardOverview.tsx`
- `src/app/admin/components/CustomerManagement.tsx`
- `src/app/admin/components/UserManagement.tsx`
- `src/app/admin/components/SearchHistory.tsx`

## Tabs Overview

### 1. Dashboard Tab

**Purpose:** High-level statistics and metrics

**Displays:**

- Total Customers (with active count)
- Total Users (with active count)
- Total Searches (in last 30 days)
- Success Rate (percentage of successful searches)

**Metrics Cards:**



```

<div className="grid grid-cols-4 gap-4">
  <StatCard
    title="Total Customers"
    value={stats.totalCustomers}
    subtitle={` ${stats.activeCustomers} active`}
    icon="👤"
    color="blue"
  />
  <StatCard
    title="Total Users"
    value={stats.totalUsers}
    subtitle={` ${stats.activeUsers} active`}
    icon="👥"
    color="green"
  />
  <StatCard
    title="Total Searches"
    value={stats.totalSearches}
    subtitle="Last 30 days"
    icon="🔍"
    color="purple"
  />
  <StatCard
    title="Success Rate"
    value={` ${stats.successRate}%`}
    subtitle="Avg 762ms"
    icon="✓"
    color="pink"
  />
</div>

```

#### Additional Sections:

- User Breakdown (Super Admins, Customer Admins, Customer Users, New Users)
- Search Statistics (Successful, No Results, Errors, Active Searchers)
- Top Customers (by search volume in last 30 days)
- Recent Activity (latest searches across all customers)

## 2. Customers Tab

**Purpose:** Manage customer organizations

**Features:**

**Create New Customer**

```
// Click "Create New Customer" button
// Modal opens with form:
{
  customerName: string,      // e.g., "ACME Corporation"
  adminUsername: string,     // e.g., "acme_admin"
  adminPassword: string,     // Min 8 chars
  adminEmail?: string        // Optional
}

// On submit:
POST /api/admin/customers
{
  customerName, adminUsername, adminPassword, adminEmail
}

// Response:
{
  success: true,
  customer: { id, name, created_at },
  user: { id, username, email, role }
}
```

### Customer List Table

| Column        | Description                                     |
|---------------|-------------------------------------------------|
| Customer      | Name and ID                                     |
| Status        | Active/Inactive badge                           |
| Users         | Active users / Total users                      |
| Searches      | Total search count                              |
| Last Activity | Last search timestamp                           |
| Actions       | View Details, Logo, Activate/Deactivate, Delete |

### Customer Actions:

#### 1. View Details

- Opens modal showing customer info
- Recent search history
- User list

#### 2. Upload Logo

- Opens modal with file upload
- Supported: JPG, PNG, GIF, BMP (max 5MB)
- Uploads to Supabase Storage
- Stores URL in customers.logo\_url

#### 3. Activate/Deactivate

- Toggle customer active status

- Inactive customers cannot login
- Confirmation dialog

#### 4. **Delete** (future feature)

- Soft delete customer
- Prevents data loss

### 3. **Users Tab**

**Purpose:** View and manage all users across all customers

#### **User List Table**

| Column     | Description                               |
|------------|-------------------------------------------|
| Username   | Username and ID                           |
| Role       | Badge (SUPER_ADMIN, CUSTOMER_ADMIN, etc.) |
| Customer   | Customer name (or “Platform Owner”)       |
| Status     | Active/Inactive                           |
| Last Login | Timestamp                                 |
| Actions    | Edit, Deactivate, Reset Password          |

#### **User Actions:**

##### 1. **Edit User** (future)

- Change role
- Update email/name
- Reassign to different customer

##### 2. **Deactivate User**

- Set is\_active = false
- User cannot login
- Can be reactivated later

##### 3. **Reset Password** (future)

- Generate temporary password
- Email to user
- Force password change on next login

### 4. **Search History Tab**

**Purpose:** View all searches across all customers

#### **Filters:**

- Customer dropdown (filter by customer)
- Search type (username, userId, displayName)
- Search mode (exact, smart, displayName)

- Date range picker
- Result status (success, no\_results, error)

**Search History Table**

| Column       | Description              |
|--------------|--------------------------|
| Search Query | Query text               |
| Type         | Search type badge        |
| Mode         | Search mode badge        |
| Result       | Username found (if any)  |
| User         | Who performed search     |
| Customer     | Customer name            |
| Status       | Success/Error/No Results |
| Time         | Timestamp                |

**Export Options:**

- Export to CSV
- Filter then export
- Includes all visible columns

## API Endpoints for Admin Dashboard

```
// Get dashboard metrics
GET /api/admin/stats
Returns: {
  totalCustomers, activeCustomers,
  totalUsers, activeUsers,
  totalSearches, successRate,
  userBreakdown, searchStatistics,
  topCustomers, recentActivity
}

// Get all customers with stats
GET /api/admin/customers
Returns: Array of customers with user/search counts

// Create customer
POST /api/admin/customers
Body: { customerName, adminUsername, adminPassword, adminEmail }
Returns: { customer, user }

// Get customer search history
GET /api/admin/customers/{customerId}/searches
Returns: Array of search records

// Get customer users
GET /api/admin/customers/{customerId}/users
Returns: Array of user records

// Upload customer logo
POST /api/admin/customers/{customerId}/logo
Body: FormData with image file
Returns: { logoUrl }

// Delete customer logo
DELETE /api/admin/customers/{customerId}/logo
Returns: { success: true }
```

## 5.6 Logo Upload System

**Purpose:** Enable white-label branding by allowing customers to upload custom logos.

**Storage:** Supabase Storage (bucket: `customer-logos` )

**Max Size:** 5MB

**Formats:** JPG, JPEG, PNG, GIF, BMP

## Upload Process

```
// 1. Admin selects customer and clicks "Logo" button
// 2. Modal opens with file input
// 3. User selects image file
// 4. Frontend validates file:
if (file.size > 5 * 1024 * 1024) {
  alert('File too large. Maximum size is 5MB');
  return;
}

const allowedTypes = ['image/jpeg', 'image/jpg', 'image/png', 'image/gif', 'image/bmp'];
if (!allowedTypes.includes(file.type)) {
  alert('Invalid file type');
  return;
}

// 5. Upload to API:
const formData = new FormData();
formData.append('logo', file);

POST /api/admin/customers/{customerId}/logo
Body: FormData

// 6. Backend:
// - Convert file to Buffer
// - Upload to Supabase Storage: customer-logos/{customerId}.{ext}
// - Get public URL
// - Update customers.logo_url in database
// - Return success

// 7. Frontend displays success message
```

## Storage Location

Supabase Storage Bucket: customer-logos

```
├─ customer-1.png
├─ customer-2.jpg
├─ customer-5.png
└─ customer-12.gif
```

Public URLs:

<https://i.ytimg.com/vi/TJN0rjyqXhM/maxresdefault.jpg>

## Display Logic

```
// On main page, check if user has customer
if (session?.user?.customerId) {
  // Fetch logo URL
  const res = await fetch(`/api/customer-logo/${customerId}`);
  const { logoUrl } = await res.json();

  // Display logo
  <img src={logoUrl} alt="Customer Logo" />
}

// If no logo, display default Roblox Verifier Tool branding
```

## Delete Logo

```
DELETE /api/admin/customers/{customerId}/logo

// Backend:
// - Delete file from Supabase Storage
// - Set customers.logo_url = NULL
// - Return success
```

## 5.7 Search History Logging

**Purpose:** Automatically log all searches for auditing, analytics, and debugging.

**Table:** search\_history

**Logged By:** /api/search route (after successful search)

### What Gets Logged






```
{
  user_id: 5,                // Who searched
  customer_id: 2,            // Their customer (NULL for SUPER_ADMIN)
  search_type: 'username',   // Input type
  search_mode: 'smart',      // Mode used
  search_query: 'johndoe',   // Original query
  roblox_username: 'JohnDoe', // Found username
  roblox_user_id: 5533914010, // Found user ID
  roblox_display_name: 'John D.', // Found display name
  has_verified_badge: true,   // Badge status
  result_data: { /* full JSON */ }, // Complete API response
  result_count: 1,           // Number of results
  result_status: 'success',   // success/no_results/error
  error_message: null,        // Error details if failed
  response_time_ms: 762,      // API response time
  searched_at: '2025-10-21T14:30:00Z' // Timestamp
}
```

## Logging Implementation

```
// In /api/search route (after search completes)
if (userId) {
  const responseTime = Date.now() - start;
  const searchType = /^\d+$/.test(keyword) ? 'userId' : 'username';
  const firstResult = users.length > 0 ? users[0] : null;

  logSearch({
    userId: parseInt(userId),
    customerId: customerId && customerId !== 'null' ? parseInt(customerId) : null,
    searchType,
    searchMode: 'smart', // or 'exact' or 'displayName'
    searchQuery: keyword,
    robloxUsername: firstResult?.name,
    robloxUserId: firstResult?.id,
    robloxDisplayName: firstResult?.displayName,
    hasVerifiedBadge: firstResult?.hasVerifiedBadge,
    resultData: { users, searchResults },
    resultCount: users.length,
    resultStatus: users.length > 0 ? 'success' : 'no_results',
    responseTimeMs: responseTime,
  }).catch(err => {
    console.error('Failed to log search:', err);
    // Don't fail the request if logging fails
  });
}
```

### Key Features:

-  **Non-blocking:** Logging is async (doesn't slow down searches)
-  **Fault-tolerant:** Search succeeds even if logging fails
-  **Comprehensive:** Captures all relevant data
-  **Performance metrics:** Tracks response times
-  **Error tracking:** Logs failed searches for debugging

## 5.8 Rate Limiting & Protection

**Purpose:** Prevent API abuse and avoid getting banned by Roblox.

### Mechanisms:

1. **Cooldowns** (frontend)
2. **Circuit Breaker** (backend)
3. **Request Queue** (backend)
4. **Retry Logic** (backend)
5. **Caching** (backend, optional)

### 5.8.1 Frontend Cooldowns

**Implementation:** React hook ( `src/app/hooks/useCooldown.ts` )



```
function useCooldown({ key, durationSeconds }) {
  const [isOnCooldown, setIsOnCooldown] = useState(false);
  const [remainingSeconds, setRemainingSeconds] = useState(0);

  const startCooldown = () => {
    const expiresAt = Date.now() + (durationSeconds * 1000);
    localStorage.setItem(`${key}_cooldown_expires`, expiresAt.toString());
    setIsOnCooldown(true);
    // Start countdown timer
  };

  // On mount, check if cooldown is active
  useEffect(() => {
    const expires = localStorage.getItem(`${key}_cooldown_expires`);
    if (expires && Date.now() < parseInt(expires)) {
      setIsOnCooldown(true);
      // Resume countdown
    }
  }, []);

  return { isOnCooldown, remainingSeconds, startCooldown };
}
```

**Used For:**

- Smart Match mode: 30-second cooldown
- Display Name mode: 30-second cooldown
- Exact Match mode: No cooldown

**UI Behavior:**

```
<button
  disabled={isOnCooldown}
  onClick={handleSearch}
>
  {isOnCooldown
    ? `Wait ${remainingSeconds}s...`
    : 'Smart Search'
  }
</button>
```

**5.8.2 Circuit Breaker**

**Purpose:** Stop making requests to Roblox API if it's consistently failing.

**Location:** `src/app/lib/utils/circuit-breaker.ts`

**States:**

- **CLOSED:** Normal operation, requests pass through
- **OPEN:** Too many failures, block all requests
- **HALF\_OPEN:** Testing if service recovered, allow limited requests

```

class CircuitBreaker {
  private state: 'CLOSED' | 'OPEN' | 'HALF_OPEN' = 'CLOSED';
  private failureCount = 0;
  private successCount = 0;
  private nextAttempt = Date.now();

  async execute(fn) {
    // If OPEN and not ready to retry, throw error
    if (this.state === 'OPEN' && Date.now() < this.nextAttempt) {
      throw new Error('Circuit breaker is OPEN');
    }

    // If HALF_OPEN, allow one request to test
    if (this.state === 'OPEN') {
      this.state = 'HALF_OPEN';
    }

    try {
      const result = await fn();
      this.onSuccess();
      return result;
    } catch (error) {
      this.onFailure();
      throw error;
    }
  }

  private onSuccess() {
    this.failureCount = 0;
    if (this.state === 'HALF_OPEN') {
      this.state = 'CLOSED';
    }
  }

  private onFailure() {
    this.failureCount++;
    if (this.failureCount >= 5) {
      this.state = 'OPEN';
      this.nextAttempt = Date.now() + 60000; // 1 minute
    }
  }
}

```

**Parameters:**

- Failure threshold: 5 consecutive failures
- Timeout: 60 seconds
- Half-open success threshold: 1 successful request

**5.8.3 Request Queue**

**Purpose:** Limit concurrent requests to Roblox API.

**Location:** `src/app/lib/utils/request-queue.ts`

```

class RequestQueue {
  private queue: Array<{ fn: Function, priority: number }> = [];
  private running = 0;
  private maxConcurrent = 3;

  async enqueue(fn, priority = 5) {
    return new Promise((resolve, reject) => {
      this.queue.push({ fn, priority, resolve, reject });
      this.queue.sort((a, b) => a.priority - b.priority);
      this.processQueue();
    });
  }

  private async processQueue() {
    if (this.running >= this.maxConcurrent || this.queue.length === 0) {
      return;
    }

    this.running++;
    const { fn, resolve, reject } = this.queue.shift();

    try {
      const result = await fn();
      resolve(result);
    } catch (error) {
      reject(error);
    } finally {
      this.running--;
      this.processQueue();
    }
  }
}

```

**Parameters:**

- Max concurrent requests: 3
- Priority: Lower number = higher priority (1-10)

**Priority Scheme:**

- User searches: Priority 5
- Admin searches: Priority 3
- Background tasks: Priority 7

**5.8.4 Retry Logic**

**Purpose:** Retry failed requests with exponential backoff.

**Location:** `src/app/lib/utils/retry.ts`

```

async function withRetry(fn, options = {}) {
  const {
    maxRetries = 3,
    initialDelayMs = 1000,
    backoffMultiplier = 2,
    onRetry = () => {}
  } = options;

  for (let attempt = 0; attempt < maxRetries; attempt++) {
    try {
      return await fn();
    } catch (error) {
      if (attempt === maxRetries - 1) throw error;

      const delayMs = initialDelayMs * Math.pow(backoffMultiplier, attempt);
      onRetry(error, attempt, delayMs);

      await new Promise(resolve => setTimeout(resolve, delayMs));
    }
  }
}

```

#### Parameters:

- Max retries: 3
- Initial delay: 1000ms (1 second)
- Backoff multiplier: 2 (doubles each retry)
- Total max wait: 1s + 2s + 4s = 7 seconds

#### Retry Schedule:

- Attempt 1: Immediate
- Attempt 2: After 1 second
- Attempt 3: After 2 seconds
- Attempt 4: After 4 seconds
- Give up after 4 attempts

### 5.8.5 Caching (Optional)

**Purpose:** Cache Roblox API responses to reduce API calls.

**Location:** src/app/lib/utils/cache.ts

**Backend:** Redis (optional, via ioredis)

```

async function withCache(key, fn, { ttl = 300 } = {}) {
  // Check cache first
  const cached = await redis.get(key);
  if (cached) {
    return { data: JSON.parse(cached), fromCache: true };
  }

  // Cache miss, execute function
  const result = await fn();

  // Store in cache
  await redis.setex(key, ttl, JSON.stringify(result));

  return { data: result, fromCache: false };
}

```

**Cache TTLs:**

- Exact searches: 3600 seconds (1 hour)
- Fuzzy searches: 300 seconds (5 minutes)
- User profiles: 1800 seconds (30 minutes)

**Cache Keys:**

```
roblox:search:exact:{username}
roblox:search:fuzzy:{query}
roblox:profile:{userId}
roblox:thumbnail:{userId}
```

## 6. API Endpoints

### 6.1 Authentication APIs

All authentication handled by NextAuth.js:

```
POST /api/auth/signin
GET /api/auth/signout
GET /api/auth/session
GET /api/auth/csrf
```

### 6.2 Search APIs

#### POST /api/roblox

**Purpose:** Exact username lookup (direct Roblox API call)

**Request:**

```
POST /api/roblox
Content-Type: application/json

{
  "username": "JohnDoe",
  "includeBanned": false
}
```

**Response:**

```
{
  "data": [{
    "requestedUsername": "JohnDoe",
    "id": 5533914010,
    "name": "JohnDoe",
    "displayName": "John D.",
    "hasVerifiedBadge": true
  }]
}
```

**Headers Required:**

- X-User-Id : Set by middleware
- X-Customer-Id : Set by middleware

**GET /api/roblox?userId={id}**

**Purpose:** Lookup user by ID

**Request:**

```
GET /api/roblox?userId=5533914010
```

**Response:**

```
{
  "id": 5533914010,
  "name": "JohnDoe",
  "displayName": "John D.",
  "hasVerifiedBadge": true,
  "description": "Professional Roblox player",
  "created": "2023-02-05T12:00:00.000Z",
  "isBanned": false
}
```

**GET /api/search**

**Purpose:** Fuzzy search with ranking (Smart Match or Display Name modes)

**Request:**

```
GET /api/search?keyword=john&limit=10&searchMode=smart
```

**Query Parameters:**

- keyword (required): Search query
- limit (optional): Max results (default: 10)
- cursor (optional): Pagination cursor
- searchMode (optional): 'smart' or 'displayName' (default: 'smart')

**Response:**

```
{
  "previousPageCursor": null,
  "nextPageCursor": "abc123",
  "data": [
    {
      "id": 5533914010,
      "name": "JohnDoe",
      "displayName": "John",
      "hasVerifiedBadge": true,
      "previousUsernames": [],
      "description": "Pro player",
      "created": "2023-02-05T12:00:00.000Z",
      "isBanned": false
    }
  ],
  "fromCache": false,
  "cacheTtl": 300
}
```

**Logging:** Automatically logs to `search_history` table (non-blocking)

## 6.3 Profile APIs

### GET /api/profile/{userId}

**Purpose:** Get comprehensive user profile with all data

**Request:**

```
GET /api/profile/5533914010
```

**Response:**

```
{
  "user": {
    "id": 5533914010,
    "name": "JohnDoe",
    "displayName": "John D.",
    "hasVerifiedBadge": true,
    "description": "Professional Roblox player",
    "created": "2023-02-05T12:00:00.000Z",
    "isBanned": false
  },
  "counts": {
    "friends": 27,
    "followers": 1250,
    "following": 180
  },
  "groups": [
    {
      "id": 12345,
      "name": "Pro Players",
      "role": "Owner",
      "memberCount": 5000,
      "isOwner": true
    }
  ],
  "friends": [
    {
      "id": 123,
      "name": "Friend1",
      "displayName": "Friend One",
      "hasVerifiedBadge": false
    }
  ]
}
```

**Used By:** Deep Context component

### GET /api/friends/{userId}

**Purpose:** Get user's friend list

**Request:**

```
GET /api/friends/5533914010
```

**Response:**

```
{
  "friends": [
    {
      "id": 123,
      "name": "Friend1",
      "displayName": "Friend One",
      "hasVerifiedBadge": false,
      "isOnline": false
    }
  ]
}
```



**GET /api/thumbnail?userId={id}****Purpose:** Get user avatar/thumbnail image**Request:**`GET /api/thumbnail?userId=5533914010`**Response:** Image (JPEG/PNG)**Headers:**

- Content-Type: image/jpeg
- Cache-Control: public, max-age=3600

**Used By:** Search results, Deep Context, anywhere avatars are displayed**6.4 Forensic APIs****GET /api/forensic/report?userId={id}****Purpose:** Generate forensic evidence report**Request:**`GET /api/forensic/report?userId=5533914010&caseId=CASE-2025-001`**Query Parameters:**

- `userId` (required): Roblox user ID
- `caseId` (optional): Case/investigation ID

**Response:**

```
{
  "meta": {
    "reportId": "uuid",
    "createdAt": "2025-10-21T14:30:00.000Z",
    "createdBy": "investigator@law.gov",
    "appVersion": "1.0.0",
    "forensicMode": true,
    "caseId": "CASE-2025-001"
  },
  "query": {
    "input": "JohnDoe",
    "mode": "username"
  },
  "sources": [
    { "name": "roblox.users", "fetchedAt": "..." },
    { "name": "roblox.groups", "fetchedAt": "..." }
  ],
  "snapshot": { /* full user data */ },
  "hash": {
    "algo": "SHA-256",
    "value": "a3c5f9d2e8b1..."
  },
  "chainOfCustody": [
    { "event": "generated", "actor": "...", "at": "..." }
  ]
}
```

## 6.5 Admin APIs (SUPER\_ADMIN only)

All admin APIs require `X-User-Role: SUPER_ADMIN` header (set by middleware).

### GET /api/admin/stats

**Purpose:** Get dashboard metrics

**Request:**

```
GET /api/admin/stats
```

**Response:**

```
{
  "totalCustomers": 25,
  "activeCustomers": 23,
  "totalUsers": 150,
  "activeUsers": 142,
  "totalSearches": 5420,
  "successRate": 95.2,
  "userBreakdown": {
    "superAdmins": 1,
    "customerAdmins": 25,
    "customerUsers": 124,
    "newUsers": 8
  },
  "searchStatistics": {
    "successful": 5160,
    "noResults": 200,
    "errors": 60,
    "activeSearchers": 98
  },
  "topCustomers": [
    {
      "id": 5,
      "name": "ACME Corp",
      "searches": 1250
    }
  ],
  "recentActivity": [
    {
      "username": "john_doe",
      "customer": "ACME Corp",
      "query": "test_user",
      "status": "success",
      "timestamp": "2025-10-21T14:30:00.000Z"
    }
  ]
}
```

### GET /api/admin/customers

**Purpose:** Get all customers with stats

**Request:**

```
GET /api/admin/customers
```

**Response:**

```
{
  "customers": [
    {
      "id": 5,
      "name": "ACME Corp",
      "is_active": true,
      "created_at": "2025-01-15T10:00:00.000Z",
      "total_users": 15,
      "active_users": 14,
      "total_searches": 1250,
      "last_search_at": "2025-10-21T14:25:00.000Z",
      "last_login_at": "2025-10-21T14:20:00.000Z",
      "logo_url": "https://cdn.dribbble.com/userupload/13341545/file/original-9b6dd7532fc344b5cb78cdd0504c59ed.jpg?resize=400x0"
    }
  ]
}
```

**POST /api/admin/customers**

**Purpose:** Create new customer with admin user

**Request:**

```
POST /api/admin/customers
Content-Type: application/json

{
  "customerName": "ACME Corporation",
  "adminUsername": "acme admin",
  "adminPassword": "SecurePass123!",
  "adminEmail": "admin@acme.com"
}
```

**Response:**

```
{
  "success": true,
  "customer": {
    "id": 26,
    "name": "ACME Corporation",
    "is_active": true,
    "created_at": "2025-10-21T14:30:00.000Z"
  },
  "user": {
    "id": 151,
    "username": "acme_admin",
    "role": "CUSTOMER_ADMIN",
    "customer_id": 26,
    "email": "admin@acme.com",
    "created_at": "2025-10-21T14:30:00.000Z"
  }
}
```

**Validations:**

- Customer name must be unique

- Username must be unique
- Password min 8 characters
- Transaction ensures both customer and user created atomically

### GET /api/admin/customers/{customerId}/searches

**Purpose:** Get search history for specific customer

**Request:**

```
GET /api/admin/customers/5/searches?limit=100&offset=0
```

**Response:**

```
{
  "searches": [
    {
      "id": 12345,
      "search_type": "username",
      "search_mode": "smart",
      "search_query": "johndoe",
      "searched_at": "2025-10-21T14:30:00.000Z",
      "roblox_username": "JohnDoe",
      "roblox_user_id": 5533914010,
      "result_status": "success",
      "searched_by": "john_admin"
    }
  ],
  "total": 1250,
  "limit": 100,
  "offset": 0
}
```

### GET /api/admin/customers/{customerId}/users

**Purpose:** Get all users for specific customer

**Request:**

```
GET /api/admin/customers/5/users
```

**Response:**

```
{
  "users": [
    {
      "id": 50,
      "username": "acme_admin",
      "role": "CUSTOMER_ADMIN",
      "email": "admin@acme.com",
      "full_name": "John Admin",
      "is_active": true,
      "created_at": "2025-01-15T10:00:00.000Z",
      "last_login": "2025-10-21T14:20:00.000Z"
    }
  ]
}
```

**POST /api/admin/customers/{customerId}/logo****Purpose:** Upload customer logo**Request:**

```
POST /api/admin/customers/5/logo
Content-Type: multipart/form-data

logo: [file]
```

**Response:**

```
{
  "success": true,
  "logoUrl": "https://lh7-rt.googleusercontent.com/docsz/AD_4nXfdp0zYDdhxzPPwBipeRm75XfsIbTY0eDot-HqXHgr79RWdjcnkMkzLVlHPoUDoziRwzVZhM9PLzY-aqkY00pw-M4Sg0ljkcTaH8PQfG8QYEZ5ZPo2_vgqbJQ_aH1C0spwif2kXmWg?key=XIntgADwBooqCwvZj-4sM_pP"
}
```

**Validations:**

- File size max 5MB
- File type: image/jpeg, image/png, image/gif, image/bmp
- Replaces existing logo if present

**DELETE /api/admin/customers/{customerId}/logo****Purpose:** Delete customer logo**Request:**

```
DELETE /api/admin/customers/5/logo
```

**Response:**

```
{
  "success": true
}
```

**6.6 Health Check API****GET /api/health****Purpose:** Health check endpoint (no auth required)**Request:**

```
GET /api/health
```

**Response:**

```
{
  "status": "healthy",
  "timestamp": "2025-10-21T14:30:00.000Z",
  "database": "connected",
  "version": "1.0.0"
}
```

#### Status Codes:

- 200 : Healthy
- 503 : Database connection failed

## 7. Frontend Components

### 7.1 Page Components

#### Main Search Page ( `src/app/page.tsx` )

**Purpose:** Primary user interface for searching Roblox users

#### Features:

- Search input with mode selector
- Forensic mode toggle
- Batch CSV upload
- Results display (verified, suggestions, no results)
- Admin dashboard button (SUPER\_ADMIN only)
- Customer logo display
- Logout button

#### State Management:

```
const [input, setInput] = useState('');
const [result, setResult] = useState(null);
const [loading, setLoading] = useState(false);
const [forensicMode, setForensicMode] = useState(false);
const [searchMode, setSearchMode] = useState('exact');
const [scoredCandidates, setScoredCandidates] = useState([]);
const [showDeepContext, setShowDeepContext] = useState(false);
const [selectedUserId, setSelectedUserId] = useState(null);
```

#### Search Flow:

```

async function handleSubmit(e) {
  e.preventDefault();
  setLoading(true);

  // Parse input
  const { type, value } = normalizeInput(input);

  // Perform search based on mode
  switch (searchMode) {
    case 'exact':
      // Direct API call
      break;
    case 'smart':
      // Fuzzy search + AI ranking
      break;
    case 'displayName':
      // Fuzzy search + all results
      break;
  }

  // Display results
  setResult(...);
  setLoading(false);
}

```

### Sign In Page ( src/app/auth/signin/page.tsx )

**Purpose:** User login interface

**Features:**

- Username input
- Password input
- Remember me checkbox (30 days vs 2 hours)
- Error message display
- Sign in button

**Authentication:**

```

async function handleSubmit(e) {
  e.preventDefault();

  const result = await signIn('credentials', {
    username,
    password,
    rememberMe: rememberMe.toString(),
    redirect: false
  });

  if (result?.error) {
    setError('Invalid credentials');
  } else {
    router.push('/');
  }
}

```

### Admin Dashboard Page ( src/app/admin/page.tsx )

**Purpose:** Super Admin interface

**Features:**

- Tab navigation (Dashboard, Customers, Users, Search History)
- Conditional rendering based on active tab
- Protected route (SUPER\_ADMIN only)
- Home and Logout buttons

**Component Structure:**

```
function AdminDashboard() {
  const [activeTab, setActiveTab] = useState('overview');

  return (
    <div>
      <header>
        <h1>Super Admin Dashboard</h1>
        <nav>
          <button onClick={() => setActiveTab('overview')}>Dashboard</button>
          <button onClick={() => setActiveTab('customers')}>Customers</button>
          <button onClick={() => setActiveTab('users')}>Users</button>
          <button onClick={() => setActiveTab('searches')}>Search History</button>
        </nav>
      </header>

      <main>
        {activeTab === 'overview' && <DashboardOverview />}
        {activeTab === 'customers' && <CustomerManagement />}
        {activeTab === 'users' && <UserManagement />}
        {activeTab === 'searches' && <SearchHistory />}
      </main>
    </div>
  );
}
```

## 7.2 Feature Components

**DeepContext ( src/app/components/DeepContext.tsx )****Purpose:** Modal overlay showing comprehensive user profile**Props:**

```
interface DeepContextProps {
  userId: string;
  onClose: () => void;
}
```

**Tabs:**

- Overview (user info, stats)
- Groups & Roles
- Activity
- Linked Mentions
- Friends

**Data Fetching:**



```
useEffect(() => {
  async function fetchProfile() {
    const res = await fetch(`/api/profile/${userId}`);
    const data = await res.json();
    setProfileData(data);
  }
  fetchProfile();
}, [userId]);
```

### SmartSuggest ( src/app/components/SmartSuggest.tsx )

**Purpose:** Display Smart Match results with confidence scores

**Props:**

```
interface SmartSuggestProps {
  candidates: ScoredCandidate[];
  query: string;
  onSelect: (username: string) => void;
  onInspect: (userId: number) => void;
  loading: boolean;
}
```

**Features:**

- Confidence badges (High 90%+, Medium 70-89%, Low <70%)
- Ranking breakdown tooltips
- “Try This” button to auto-fill search
- “Inspect” button to open Deep Context

**UI:**

```
{candidates.map((candidate, index) => (
  <div key={candidate.user.id}>
    <div className="rank">#{index + 1}</div>
    <img src={`/api/thumbnail?userId=${candidate.user.id}`} />
    <div>
      <h3>{candidate.user.displayName}</h3>
      <p>{candidate.user.name}</p>
    </div>
    <div className="confidence">
      {candidate.confidence}%
    </div>
    <div className="actions">
      <button onClick={() => onSelect(candidate.user.name)}>
        Try This
      </button>
      <button onClick={() => onInspect(candidate.user.id)}>
        Inspect
      </button>
    </div>
  </div>
))}
```

### DisplayNameResults ( src/app/components/DisplayNameResults.tsx )

**Purpose:** Display all Display Name search results (no ranking)

**Props:**

```
interface DisplayNameResultsProps {
  users: UserResult[];
  query: string;
  onSelect: (username: string) => void;
  onInspect: (userId: number) => void;
  loading: boolean;
}
```

**Features:**

- Simple list of matching users
- Avatar display
- Username and display name
- No confidence scores (unlike SmartSuggest)
- “Select” and “Inspect” buttons

**ForensicMode ( src/app/components/ForensicMode.tsx )**

**Purpose:** Toggle and display forensic mode features

**Props:**

```
interface ForensicModeProps {
  isEnabled: boolean;
  onToggle: (enabled: boolean) => void;
  currentSnapshot: Record<string, unknown> | null;
  query: { input: string; mode: string } | null;
}
```

**Features:**

- Toggle switch (Active/Inactive)
- Current snapshot display when active
- “Generate Report” button
- Report preview
- Export buttons (PDF, JSON)

**SearchModeSelector ( src/app/components/SearchModeSelector.tsx )**

**Purpose:** UI for selecting search mode (Exact, Smart, Display Name)

**Props:**

```
interface SearchModeSelectorProps {
  selectedMode: SearchMode;
  onModeChange: (mode: SearchMode) => void;
  smartCooldown: CooldownHook;
  displayNameCooldown: CooldownHook;
}
```

**UI:**

```

<div className="search-modes">
  <button
    className={selectedMode === 'exact' ? 'active' : ''}
    onClick={() => onModeChange('exact')}
  >
    <Icon name="target" />
    Exact Match
  </button>

  <button
    className={selectedMode === 'smart' ? 'active' : ''}
    onClick={() => onModeChange('smart')}
    disabled={smartCooldown.isOnCooldown}
  >
    <Icon name="brain" />
    Smart Match
    {smartCooldown.isOnCooldown && (
      <span>({smartCooldown.remainingSeconds}s)</span>
    )}
  </button>

  <button
    className={selectedMode === 'displayName' ? 'active' : ''}
    onClick={() => onModeChange('displayName')}
    disabled={displayNameCooldown.isOnCooldown}
  >
    <Icon name="tag" />
    Display Name
    {displayNameCooldown.isOnCooldown && (
      <span>({displayNameCooldown.remainingSeconds}s)</span>
    )}
  </button>
</div>

```

## NoResultsModal ( src/app/components/NoResultsModal.tsx )

**Purpose:** Modal shown when Exact Match finds no results

**Props:**

```

interface NoResultsModalProps {
  isOpen: boolean;
  onClose: () => void;
  onTrySmartSearch: () => void;
  searchQuery: string;
}

```

**Features:**

- Explains why no results were found
- Suggests trying Smart Match mode
- “Try Smart Match” button (auto-switches mode)
- “Close” button

**UI:**

```

<Modal isOpen={isOpen} onClose={onClose}>
  <h2>No Exact Match Found</h2>
  <p>
    No exact match found for "{searchQuery}".
  </p>
  <p>
    Try using <strong>Smart Match</strong> mode for fuzzy search
    with AI-powered ranking.
  </p>
  <button onClick={onTrySmartSearch}>
    Try Smart Match
  </button>
  <button onClick={onClose}>
    Close
  </button>
</Modal>

```

## 7.3 Admin Components

### DashboardOverview ( src/app/admin/components/DashboardOverview.tsx )

**Purpose:** Admin dashboard overview with metrics

**Features:**

- Stat cards (customers, users, searches, success rate)
- User breakdown section
- Search statistics section
- Top customers table
- Recent activity table

**Data Fetching:**

```

useEffect(() => {
  async function fetchStats() {
    const res = await fetch('/api/admin/stats');
    const data = await res.json();
    setStats(data);
  }
  fetchStats();
}, []);

```

### CustomerManagement ( src/app/admin/components/CustomerManagement.tsx )

**Purpose:** Manage customers (create, view, activate/deactivate, logo)

**State:**

```

const [customers, setCustomers] = useState([]);
const [showCreateModal, setShowCreateModal] = useState(false);
const [showDetailsModal, setShowDetailsModal] = useState(false);
const [showLogoModal, setShowLogoModal] = useState(false);
const [selectedCustomer, setSelectedCustomer] = useState(null);

```

**Features:**

#### 1. Create Customer Modal

```
```.tsx
```

Create Customer

...

### 1. Customer List Table

```tsx

{customers.map(customer => ( ))}

| Customer         | Status                                         | Users                                               | Searches                    | Last Activ-ity                           | Actions                                                                                                                                   |
|------------------|------------------------------------------------|-----------------------------------------------------|-----------------------------|------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------|
| {custom-er.name} | {custom-er.is_active ? 'Active' : 'In-active'} | {custom-er.active_use-rs} / {cus-tomer.total_users} | {custom-er.total_sear-ches} | {format-Date(custom-er.last_searc-h_at)} | <div>viewDetails(customer)}&gt;View</div> <div>uploadLogo(customer)}&gt;L</div> <div>toggleStatus(customer)}{customer.is_active ? }</div> |

...

### 1. Logo Upload Modal

```tsx

{currentLogo ? (

Current Logo

) : (

No logo uploaded

)}

Max size: 5MB. Formats: JPG, PNG, GIF, BMP

Delete Logo

Upload Logo

{currentLogo && (

)}

...

## UserManagement ( src/app/admin/components/UserManagement.tsx )

**Purpose:** View and manage all users

### Features:

- User list table
- Filter by customer
- Filter by role
- Filter by status (active/inactive)
- Deactivate user action (future: edit, reset password)

**Table:**

```

<table>
  <thead>
    <tr>
      <th>Username</th>
      <th>Role</th>
      <th>Customer</th>
      <th>Status</th>
      <th>Last Login</th>
      <th>Actions</th>
    </tr>
  </thead>
  <tbody>
    {users.map(user => (
      <tr key={user.id}>
        <td>
          {user.username}
          <span className="user-id">ID: {user.id}</span>
        </td>
        <td>
          <Badge color={getRoleColor(user.role)}>
            {user.role}
          </Badge>
        </td>
        <td>{user.customer_name || 'Platform Owner'}</td>
        <td>
          <Badge color={user.is_active ? 'green' : 'gray'}>
            {user.is_active ? 'Active' : 'Inactive'}
          </Badge>
        </td>
        <td>{formatDate(user.last_login)}</td>
        <td>
          <button onClick={() => deactivateUser(user)}>
            Deactivate
          </button>
        </td>
      </tr>
    )})}
  </tbody>
</table>

```

**SearchHistory ( src/app/admin/components/SearchHistory.tsx )****Purpose:** View search history across all customers**Features:**

- Filter by customer (dropdown)
- Filter by search mode (exact/smart/displayName)
- Filter by result status (success/no\_results/error)
- Date range picker
- Export to CSV

**Table:**

```

<table>
  <thead>
    <tr>
      <th>Search Query</th>
      <th>Type</th>
      <th>Mode</th>
      <th>Result</th>
      <th>User</th>
      <th>Customer</th>
      <th>Status</th>
      <th>Time</th>
    </tr>
  </thead>
  <tbody>
    {searchHistory.map(search => (
      <tr key={search.id}>
        <td>{search.search_query}</td>
        <td>
          <Badge>{search.search_type}</Badge>
        </td>
        <td>
          <Badge>{search.search_mode}</Badge>
        </td>
        <td>{search.roblox_username || '-'}</td>
        <td>{search.searched_by}</td>
        <td>{search.customer_name}</td>
        <td>
          <Badge color={getStatusColor(search.result_status)}>
            {search.result_status}
          </Badge>
        </td>
        <td>{formatDate(search.searched_at)}</td>
      </tr>
    ))}
  </tbody>
</table>

```

## 7.4 Utility Components

### Alert ( src/app/components/ui/alert.tsx )

**Purpose:** Reusable alert component for success/error messages

**Props:**

```

interface AlertProps {
  type: 'success' | 'error' | 'warning' | 'info';
  message: string;
  onClose?: () => void;
}

```

### RateLimitAlert ( src/app/components/ui/rate-limit-alert.tsx )

**Purpose:** Display rate limiting warnings

**Props:**

```
interface RateLimitAlertProps {
  retryAfter: number; // seconds
  retryAfterDate: Date;
}
```

**UI:**

```
<Alert type="warning">
  <p>
    Too many requests. Please wait {retryAfter} seconds
    before trying again.
  </p>
  <p>Retry after: {retryAfterDate.toLocaleString()}</p>
</Alert>
```

## 7.5 Custom Hooks

### useCooldown ( src/app/hooks/useCooldown.ts )

**Purpose:** Manage cooldown timers for Smart Match and Display Name modes

**Usage:**

```
const smartCooldown = useCooldown({
  key: 'smart_search',
  durationSeconds: 30
});

// Start cooldown
smartCooldown.startCooldown();

// Check status
if (smartCooldown.isOnCooldown) {
  console.log(`Wait ${smartCooldown.remainingSeconds}s`);
}
```

**Implementation:**



```
function useCooldown({ key, durationSeconds }) {
  const [isOnCooldown, setIsOnCooldown] = useState(false);
  const [remainingSeconds, setRemainingSeconds] = useState(0);

  useEffect(() => {
    // Check localStorage for active cooldown
    const expires = localStorage.getItem(`${key}_cooldown_expires`);
    if (expires && Date.now() < parseInt(expires)) {
      setIsOnCooldown(true);
      startCountdown(parseInt(expires));
    }
  }, [key]);

  const startCooldown = useCallback(() => {
    const expiresAt = Date.now() + (durationSeconds * 1000);
    localStorage.setItem(`${key}_cooldown_expires`, expiresAt.toString());
    setIsOnCooldown(true);
    startCountdown(expiresAt);
  }, [key, durationSeconds]);

  const startCountdown = (expiresAt) => {
    const interval = setInterval(() => {
      const remaining = Math.ceil((expiresAt - Date.now()) / 1000);
      if (remaining <= 0) {
        setIsOnCooldown(false);
        setRemainingSeconds(0);
        localStorage.removeItem(`${key}_cooldown_expires`);
        clearInterval(interval);
      } else {
        setRemainingSeconds(remaining);
      }
    }, 1000);
  };

  return { isOnCooldown, remainingSeconds, startCooldown };
}
```

## 8. External Integrations

### 8.1 Roblox APIs

The application integrates with Roblox's public APIs to fetch user data.

#### Base URLs

Users API:	<a href="https://users.roblox.com">https://users.roblox.com</a>
Thumbnails API:	<a href="https://thumbnails.roblox.com">https://thumbnails.roblox.com</a>
Groups API:	<a href="https://groups.roblox.com">https://groups.roblox.com</a>
Friends API:	<a href="https://friends.roblox.com">https://friends.roblox.com</a>
Presence API:	<a href="https://presence.roblox.com">https://presence.roblox.com</a>

## API Endpoints Used

### 1. User Lookup by Username

```
POST https://users.roblox.com/v1/usernames/users
Content-Type: application/json
```

```
{
  "usernames": ["JohnDoe"],
  "excludeBannedUsers": true
}
```

#### Response:

```
{
  "data": [{
    "requestedUsername": "JohnDoe",
    "id": 5533914010,
    "name": "JohnDoe",
    "displayName": "John D.",
    "hasVerifiedBadge": true
  }]
}
```

### 2. User Lookup by ID

```
GET https://users.roblox.com/v1/users/{userId}
```

#### Response:

```
{
  "description": "Professional Roblox player",
  "created": "2023-02-05T12:00:00.000Z",
  "isBanned": false,
  "externalAppDisplayName": null,
  "hasVerifiedBadge": true,
  "id": 5533914010,
  "name": "JohnDoe",
  "displayName": "John D."
}
```

### 3. User Search (Fuzzy)

```
GET https://users.roblox.com/v1/users/search?keyword={query}&limit={limit}&cursor={cursor}
```

#### Query Parameters:

- `keyword` : Search query (URL encoded)
- `limit` : Number of results (max 25)
- `cursor` : Pagination cursor (optional)

#### Response:

```
{
  "previousPageCursor": null,
  "nextPageCursor": "abc123def456",
  "data": [
    {
      "previousUsernames": ["OldName"],
      "hasVerifiedBadge": false,
      "id": 123456,
      "name": "CurrentName",
      "displayName": "Display"
    }
  ]
}
```

#### 4. User Thumbnail

```
GET https://i.ytimg.com/vi/g5xc8z8T3To/maxresdefault.jpg
```

#### Response:

```
{
  "data": [{
    "targetId": 5533914010,
    "state": "Completed",
    "imageUrl": "https://i.ytimg.com/vi/A4PA91jcL2Y/mqdefault.jpg"
  }]
}
```

#### 5. User Groups

```
GET https://groups.roblox.com/v2/users/{userId}/groups/roles
```

#### Response:

```
{
  "data": [{
    "group": {
      "id": 12345,
      "name": "Pro Players",
      "description": "Group for pro players",
      "owner": {
        "hasVerifiedBadge": false,
        "userId": 5533914010,
        "username": "JohnDoe",
        "displayName": "John D."
      },
      "shout": null,
      "memberCount": 5000,
      "isBuildersClubOnly": false,
      "publicEntryAllowed": true,
      "hasVerifiedBadge": false
    },
    "role": {
      "id": 67890,
      "name": "Owner",
      "rank": 255
    }
  }]
}
```

## 6. User Friends

```
GET https://friends.roblox.com/v1/users/{userId}/friends
```

### Response:

```
{
  "data": [{
    "isOnline": false,
    "isDeleted": false,
    "friendFrequentScore": 0,
    "friendFrequentRank": 0,
    "hasVerifiedBadge": false,
    "description": "",
    "created": "2022-01-15T10:00:00.000Z",
    "isBanned": false,
    "externalAppDisplayName": null,
    "id": 123,
    "name": "Friend1",
    "displayName": "Friend One"
  }]
}
```

## 7. User Counts (Friends, Followers, Following)

```
GET https://friends.roblox.com/v1/users/{userId}/friends/count
```

### Response:

```
{
  "count": 27
}
```

```
GET https://friends.roblox.com/v1/users/{userId}/followers/count
```

```
GET https://friends.roblox.com/v1/users/{userId}/followings/count
```

## Rate Limiting

### Roblox API Limits:

- ~10-20 requests per second per IP
- Rate limit headers:
  - X-RateLimit-Limit : Total allowed requests
  - X-RateLimit-Remaining : Remaining requests
  - X-RateLimit-Reset : Unix timestamp when limit resets
  - Retry-After : Seconds to wait (on 429 error)

### Application Protection:

1. Frontend cooldowns (30s for Smart/Display Name modes)
2. Backend circuit breaker (stops after 5 consecutive failures)
3. Request queue (max 3 concurrent requests)
4. Retry logic with exponential backoff
5. Optional Redis caching to reduce API calls

## Error Handling

### Common Roblox API Errors:

Status Code	Meaning	Handling
400	Bad Request	Validate inputs before sending
404	User not found	Return "No results" to user
429	Rate limited	Wait for Retry-After duration
500	Roblox server error	Retry with backoff, show error if persists
503	Service unavailable	Retry later, use circuit breaker

### Implementation:

```

try {
  const response = await fetchWithRateLimitDetection(url);

  if (!response.ok) {
    if (response.status === 429) {
      const retryAfter = response.headers.get('Retry-After') || 60;
      throw new RateLimitError(retryAfter);
    }
    throw new Error(`Roblox API returned ${response.status}`);
  }

  return await response.json();
} catch (error) {
  if (error instanceof RateLimitError) {
    // Show cooldown UI
  } else {
    // Show generic error
  }
}

```

## 8.2 Supabase

**Purpose:** Database and storage hosting

### Database Connection

```

import { Pool } from 'pg';

const pool = new Pool({
  connectionString: process.env.DATABASE_URL,
  ssl: { rejectUnauthorized: false },
  max: 20,
  idleTimeoutMillis: 30000,
  connectionTimeoutMillis: 2000
});

```

#### Connection String Format:

```

postgresql://postgres:[PASSWORD]@db.[PROJECT-REF].supabase.co:5432/postgres

```

### Storage Connection

```

import { createClient } from '@supabase/supabase-js';

const supabase = createClient(supabaseUrl, supabaseAnonKey);

// Upload file
await supabase.storage
  .from('customer-logos')
  .upload(`customer-${id}.png`, file, { upsert: true });

// Get public URL
const { data } = supabase.storage
  .from('customer-logos')
  .getPublicUrl(`customer-${id}.png`);

```

**Storage Bucket:** `customer-logos`

**Access:** Public (logos need to be viewable)

**Max File Size:** 5MB

**Allowed Types:** image/jpeg, image/png, image/gif, image/bmp

## Supabase Configuration

### Environment Variables Required:

```
DATABASE_URL=postgresql://postgres:[PASSWORD]@db.[PROJECT-REF].supabase.co:5432/postgres
NEXT_PUBLIC_SUPABASE_ANON_KEY=[YOUR-ANON-KEY]
# OR
SUPABASE_ANON_KEY=[YOUR-ANON-KEY]
```

**Anon Key:** Safe to expose publicly, protected by RLS policies

## 8.3 Vercel

**Purpose:** Hosting, serverless functions, CI/CD

### Deployment Configuration

**File:** `vercel.json`

```
{
  "functions": {
    "src/app/api/search/route.ts": {
      "maxDuration": 10,
      "memory": 1024
    },
    "src/app/api/profile/*/route.ts": {
      "maxDuration": 15,
      "memory": 1024
    }
  },
  "regions": ["iad1", "sfo1"]
}
```

### Explanation:

- `maxDuration` : Max execution time (seconds)
- `memory` : Memory allocated (MB)
- `regions` : Deployment regions (US East, US West)

### Environment Variables

Set in Vercel Dashboard → Project Settings → Environment Variables

#### Required:

- `DATABASE_URL`
- `NEXTAUTH_SECRET`
- `NEXTAUTH_URL`
- `NEXT_PUBLIC_SUPABASE_ANON_KEY` or `SUPABASE_ANON_KEY`

#### Optional:

- `REDIS_URL`

## Build Settings

**Framework Preset:** Next.js

**Build Command:** `npm run build` (default)

**Output Directory:** `.next` (default)

**Install Command:** `npm install` (default)

## Deployment Flow

```

Git Push to GitHub
↓
Vercel Webhook Triggered
↓
1. Install dependencies
2. Run type checks
3. Build Next.js app
4. Optimize assets
5. Deploy to edge network
↓
Deployment URL generated
↓
Production domain updated (if main branch)

```

## Vercel Features Used

1. **Serverless Functions:** All API routes
2. **Edge Network:** Static assets, SSR pages
3. **Environment Variables:** Secure storage
4. **Preview Deployments:** Every PR gets unique URL
5. **Analytics:** Page views, performance metrics
6. **Logs:** Real-time function logs

## 8.4 Redis (Optional)

**Purpose:** Caching Roblox API responses to reduce API calls

**Status:** Optional (not required, but recommended for production)

### Connection

```

import Redis from 'ioredis';

const redis = new Redis(process.env.REDIS_URL);

```

### Cache Keys

```

roblox:search:exact:{username}
roblox:search:fuzzy:{query}:{page}
roblox:profile:{userId}
roblox:thumbnail:{userId}
roblox:friends:{userId}
roblox:groups:{userId}

```



## Cache TTLs

```
{
  exactSearch: 3600,      // 1 hour
  fuzzySearch: 300,      // 5 minutes
  userProfile: 1800,     // 30 minutes
  thumbnail: 86400,      // 24 hours
  friends: 1800,         // 30 minutes
  groups: 3600           // 1 hour
}
```

## Caching Implementation

```
async function withCache(key, fn, { ttl = 300 } = {}) {
  // Try cache first
  const cached = await redis.get(key);
  if (cached) {
    return {
      data: JSON.parse(cached),
      fromCache: true
    };
  }

  // Cache miss, execute function
  const result = await fn();

  // Store in cache
  await redis.setex(key, ttl, JSON.stringify(result));

  return {
    data: result,
    fromCache: false
  };
}
```

### Benefits:

- Faster response times
- Reduced API calls to Roblox
- Lower risk of rate limiting
- Cost savings (fewer external API calls)

### Hosting Options:

- **Upstash** (serverless Redis, free tier)
  - **Redis Labs** (managed Redis)
  - **Vercel KV** (built-in Redis for Vercel)
  - Self-hosted Redis
-

## 9. Environment Variables

### 9.1 Required Variables

#### DATABASE\_URL

**Purpose:** PostgreSQL connection string

**Format:** `postgresql://username:password@host:port/database`

**Example:** `postgresql://postgres:securePass123@db.abcdefg.supabase.co:5432/postgres`

**Where to Get:**

1. Go to Supabase project dashboard
2. Settings → Database
3. Connection string → URI
4. Copy and replace `[YOUR-PASSWORD]` with your database password

**Security:** Keep secret, never commit to Git

#### NEXTAUTH\_SECRET

**Purpose:** Secret key for encrypting JWT tokens

**Format:** Random string (min 32 characters)

**How to Generate:**

```
# macOS/Linux:
openssl rand -base64 32

# Or visit:
https://generate-secret.vercel.app/32
```

**Example:** `a3F9dK2mP8qL5nR7tU1wX4yZ6bC0eH3jK5mN8pQ1sT4v`

**Security:** Keep secret, use different secrets for dev/staging/production

#### NEXTAUTH\_URL

**Purpose:** Base URL of your application

**Format:** `https://yourdomain.com` (or `http://localhost:3000` for local)

**Examples:**

- **Development:** `http://localhost:3000`
- **Production:** `https://roblox-tool.vercel.app`

**Note:** Must match the domain where your app is deployed

### 9.2 Optional Variables

#### REDIS\_URL

**Purpose:** Redis connection string for caching

**Format:** `redis://host:port` or `redis://username:password@host:port`

**Example:** `redis://default:abc123@redis-12345.upstash.io:6379`

**Required If:** You want to enable caching to reduce Roblox API calls

## NEXT\_PUBLIC\_SUPABASE\_ANON\_KEY

**Purpose:** Supabase anonymous key for storage access

**Format:** JWT token (long string)

### Where to Get:

1. Go to Supabase project dashboard
2. Settings → API
3. Copy “anon public” key (NOT service\_role key)

**Example:** `eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...`

**Security:** Safe to expose publicly (anon key is designed for client-side use)

**Alternative:** Can also set as `SUPABASE_ANON_KEY` (server-only)

## NODE\_ENV

**Purpose:** Environment mode

**Values:** `development` | `production` | `test`

**Default:** Automatically set by deployment platform

**Manual Set:** Only needed if you want to override

## 9.3 Environment Variable Template

**File:** `.env.example`

```
# =====
# DATABASE CONFIGURATION
# =====
DATABASE_URL=postgresql://username:password@localhost:5432/roblox_verifier

# =====
# NEXTAUTH CONFIGURATION
# =====
NEXTAUTH_SECRET=your-super-secret-key-here-change-this
NEXTAUTH_URL=http://localhost:3000

# =====
# SUPABASE STORAGE
# =====
NEXT_PUBLIC_SUPABASE_ANON_KEY=your-supabase-anon-key

# =====
# REDIS CONFIGURATION (Optional)
# =====
REDIS_URL=redis://localhost:6379

# =====
# APPLICATION CONFIGURATION
# =====
NODE_ENV=development
```

## 9.4 Setting Environment Variables

### Local Development

1. Create `.env.local` file in project root:  

```
bash
cp .env.example .env.local
```
2. Edit `.env.local` with actual values
3. **Never commit** `.env.local` to Git (it's in `.gitignore`)

### Vercel Production

1. Go to Vercel Dashboard
2. Select your project
3. Settings → Environment Variables
4. Add each variable:
  - Name: `DATABASE_URL`
  - Value: Your database connection string
  - Environment: Select all (Production, Preview, Development)
5. Click "Save"
6. Redeploy for changes to take effect

### Vercel CLI







```
# Pull environment variables from Vercel
vercel env pull

# Add a new environment variable
vercel env add DATABASE_URL




# List all environment variables
vercel env ls
```

## 9.5 Environment Variable Security




### Best Practices:

1.  **Never commit** `.env.local` or `.env` files to Git
2.  **Use different secrets** for development, staging, and production
3.  **Rotate secrets** regularly (every 90 days recommended)
4.  **Limit access** to environment variables (only team members who need them)
5.  **Use strong passwords** for `DATABASE_URL` (min 16 characters)
6.  **Monitor for leaks** (check public GitHub repos, logs)

### What to Keep Secret:

-  `DATABASE_URL` (contains password)
-  `NEXTAUTH_SECRET` (encrypts sessions)
-  `REDIS_URL` (if password protected)

### What's Safe to Expose:

-  `NEXTAUTH_URL` (public URL)
-  `NEXT_PUBLIC_SUPABASE_ANON_KEY` (designed for public use)
-  `NODE_ENV` (just describes environment)

---

## 10. Development History & Improvements

---

### 10.1 Initial Development (September 2025)

**Version:** 0.1.0

**Focus:** Basic functionality

**Features Implemented:**

- Basic Next.js setup with React
- Roblox username lookup
- Simple search interface
- Roblox API integration
- Display search results

**Limitations:**

- No authentication
- No database
- Single-user application
- No search history
- Basic error handling

### 10.2 Multi-Tenant Authentication (October 15, 2025)

**Version:** 0.5.0

**Major Update:** Complete authentication and multi-tenant system

**Changes:**

**1. Database Schema**

- Created PostgreSQL schema (customers, users, search\_history)
- Added indexes and constraints
- Implemented triggers for updated\_at

**1. Authentication**

- Integrated NextAuth.js
- Database-backed authentication
- Bcrypt password hashing
- JWT sessions

**2. Authorization**

- Implemented middleware for route protection
- Role-based access control (SUPER\_ADMIN, CUSTOMER\_ADMIN, CUSTOMER\_USER)
- Customer isolation (users can only see their customer's data)

**3. Admin Dashboard**

- Super Admin dashboard created
- Customer management (create, view, activate/deactivate)
- User management
- Search history viewer
- Analytics and statistics

**4. Search Logging**

- All searches automatically logged to database

- Non-blocking async logging
- Performance metrics captured

#### Files Created:

- `src/app/lib/db/schema.sql`
- `src/app/lib/db/index.ts`
- `src/app/lib/auth.ts`
- `src/middleware.ts`
- `src/app/admin/page.tsx`
- `src/app/api/admin/customers/route.ts`
- Various admin API routes

#### Documentation:

- `MULTI_TENANT_AUTH_SETUP.md`
- `IMPLEMENTATION_SUMMARY.md`

## 10.3 Deployment Fixes (October 16, 2025)

**Version:** 0.6.0

**Focus:** Fix TypeScript/ESLint errors blocking deployment

#### Issues Fixed:

1. **admin/page.tsx** (3 errors)
  - Unused `err` variables in catch blocks
  - Changed to `catch` or `catch (error)` where appropriate
1. **api/admin/customers/route.ts** (1 error)
  - Replaced `any` type with proper type checking
  - Used type guards for PostgreSQL error handling
2. **api/health/route.ts** (2 errors)
  - Removed unused `_error` variables
3. **lib/db/index.ts** (3 errors)
  - Added `eslint-disable-next-line` for necessary `any` types
  - Added comments explaining why `any` is needed

#### Database Setup:

- Created comprehensive `SETUP.md` guide
- Recommended Supabase over self-hosted PostgreSQL
- Documented environment variable setup
- Added database initialization script

#### Documentation:

- `SETUP.md` - Complete setup guide
- `DEPLOYMENT_FIX_SUMMARY.md` - Summary of fixes

## 10.4 Smart Match & Display Name Modes (October 18, 2025)

**Version:** 0.7.0

**Major Feature:** AI-powered search modes

**New Features:****1. Smart Match Mode**

- Fuzzy search with AI ranking
- Confidence scores (0-100%)
- Weighted ranking algorithm:
  - Name similarity: 40%
  - Account signals: 25%
  - Keyword hits: 15%
  - Group overlap: 10%
  - Profile completeness: 10%
  - Up to 10 ranked suggestions
  - 30-second cooldown

**2. Display Name Mode**

- Search by display name
- Show all matching users (up to 20)
- No AI ranking, simple list
- 30-second cooldown

**3. Search Mode Selector**

- UI component to switch between modes
- Visual indicators (icons, descriptions)
- Cooldown timers displayed

**4. Ranking Algorithm**

- Jaro-Winkler string similarity
- Levenshtein distance
- Account age scoring
- Verified badge weight
- Profile completeness checks

**Files Created:**

- `src/app/lib/ranking.ts` - AI ranking algorithm
- `src/app/components/SmartSuggest.tsx`
- `src/app/components/DisplayNameResults.tsx`
- `src/app/components/SearchModeSelector.tsx`
- `src/app/hooks/useCooldown.ts`

**Database Changes:**

- Added `search_mode` column to `search_history` table (migration 002)

**Documentation:**

- Feature descriptions in README

**10.5 Forensic Mode (October 19, 2025)**

**Version:** 0.8.0

**Major Feature:** Evidence collection and reporting

**Features:****1. Forensic Toggle**

- Enable/disable forensic mode
- Collects comprehensive data snapshot
- SHA-256 cryptographic hashing

**2. Evidence Report**

- User profile data
- Friends list
- Groups memberships
- Timestamps
- Chain of custody

**3. Report Generation**

- HTML report with print-to-PDF
- JSON export
- Cryptographic verification

**4. Use Cases**

- Law enforcement investigations
- Content moderation
- Legal discovery
- Compliance audits

**Files Created:**

- `src/app/lib/forensic.ts`
- `src/app/components/ForensicMode.tsx`
- `src/app/api/forensic/report/route.ts`

## 10.6 Remember Me Feature (October 20, 2025)

**Version:** 0.8.1

**Enhancement:** User session persistence

**Changes:****1. Login Page**

- Added "Remember me for 30 days" checkbox
- Styled consistently with existing UI
- Stored preference in localStorage

**2. Session Duration**

- **Checked:** 30 days session
- **Unchecked:** 2 hours session
- Dynamic JWT expiration

**3. Type Safety**

- Updated NextAuth TypeScript types
- Added `rememberMe` and `exp` to JWT interface

**Files Modified:**

- `src/app/auth/signin/page.tsx`



- `src/app/lib/auth.ts`
- `next-auth.d.ts`

#### Documentation:

- `REMEMBER_ME_FEATURE_SUMMARY.md`
- `REMEMBER_ME_IMPLEMENTATION.md`

**Commit:** `1ce8284` - "feat: Implement 'Remember Me' feature for login page"

## 10.7 Logo Upload Fix - Supabase Storage (October 21, 2025)

**Version:** 0.9.0

**Critical Fix:** Migrate from local filesystem to cloud storage

#### Problem:

- Original logo upload used Node.js `fs` module
- Vercel serverless = read-only filesystem
- Logos not persisting across deployments
- Error 500 on logo upload

#### Solution:

- Migrated to Supabase Storage
- Files stored in `customer-logos` bucket
- Public URLs for logo access
- Persistent across deployments

#### Changes:

1. **Storage Utility** ( `src/app/lib/storage.ts` )
  - Automatic Supabase URL detection from `DATABASE_URL`
  - Upload/delete/get public URL functions
  - Bucket auto-creation
2. **Logo Upload API** ( `src/app/api/admin/customers/[customerId]/logo/route.ts` )
  - Replaced `fs.writeFile()` with Supabase Storage upload
  - Better error handling
  - File validation (5MB max, image types only)
3. **Frontend** ( `src/app/admin/components/CustomerManagement.tsx` )
  - Enhanced error messages
  - Success confirmation
  - Logo preview in modal

#### Migration Steps:

- Added `NEXT_PUBLIC_SUPABASE_ANON_KEY` environment variable
- Created `customer-logos` bucket in Supabase
- Re-upload existing logos (old ones lost)

#### Files Created:

- `src/app/lib/storage.ts`

#### Files Modified:

- `src/app/api/admin/customers/[customerId]/logo/route.ts`
- `src/app/admin/components/CustomerManagement.tsx`

**Documentation:**

- LOGO\_UPLOAD\_FIX.md - Comprehensive fix documentation
- SUPABASE\_STORAGE\_SETUP.md - Setup guide
- BUCKET\_CREATION\_VISUAL\_GUIDE.md

**Commits:**

- Multiple commits fixing storage issues and RLS policies

**10.8 Database Migration - Allow NULL customer\_id (October 17, 2025)****Version:** 0.7.1**Bug Fix:** SUPER\_ADMIN search logging**Problem:**

- SUPER\_ADMIN users have customer\_id = NULL
- search\_history required NOT NULL customer\_id
- SUPER\_ADMIN searches not being logged

**Solution:**

- Migration 001: Allow NULL customer\_id in search\_history
- Updated constraints to check only user\_id IS NOT NULL
- Updated search logging to handle NULL customer\_id

**Files:**

- database/migrations/001\_allow\_null\_customer\_id\_in\_search\_history.sql

**10.9 Search Mode Column Addition (October 20, 2025)****Version:** 0.7.2**Enhancement:** Track which search mode was used**Problem:**

- search\_history only tracked search type (username/userId/etc)
- No way to distinguish between Exact/Smart/Display Name modes

**Solution:**

- Migration 002: Add search\_mode column
- Enum: 'exact' | 'smart' | 'displayName'
- Default: 'exact' for backward compatibility
- Index added for performance

**Files:**

- database/migrations/002\_add\_search\_mode\_column.sql

**10.10 Logo URL Column Addition (October 21, 2025)****Version:** 0.9.1**Enhancement:** Store customer logos**Change:**

- Migration 003: Add logo\_url column to customers table
- Type: TEXT (stores Supabase Storage URL)
- Nullable: Yes (not all customers have logos)

**Files:**

- database/migrations/003\_add\_customer\_logo.sql

**10.11 TypeScript & Next.js 15 Compatibility (October 16, 2025)****Version:** 0.6.1**Enhancement:** Upgrade to Next.js 15 and fix TypeScript issues**Changes:****1. Next.js 15 Upgrade**

- Updated to Next.js 15.5.4
- React 19.1.0
- TypeScript ^5

**2. TypeScript Fixes**

- Fixed type errors in components
- Added proper type annotations
- Resolved async component issues
- Fixed API route types

**3. ESLint Configuration**

- Updated ESLint config for Next.js 15
- Fixed linting warnings
- Added eslint-disable where necessary with explanations

**Documentation:**

- NEXTJS\_15\_FIX\_SUMMARY.md
- TYPESCRIPT\_FIX\_SUMMARY.md

**11. Known Issues & Limitations****11.1 Current Limitations****1. User Management****Missing Features:**

- ✗ Customer Admins cannot create additional users
- ✗ No user edit functionality
- ✗ No password reset feature
- ✗ No email verification
- ✗ No two-factor authentication

**Workaround:** Super Admin must create all users manually**Planned:** User management UI for Customer Admins (v2.0)**2. Search History UI for Customers****Missing Feature:**

- ✗ Customer Admins cannot view their own search history
- ✗ No customer-facing analytics dashboard

**Current State:** Only Super Admin can view search history via admin dashboard

**Planned:** Customer dashboard showing own search history and analytics (v2.0)

### 3. Batch Processing Performance

**Limitation:**

- Sequential processing (one at a time)
- Can be slow for large batches (100+ entries)
- No progress percentage display

**Recommended:** Keep batches under 100 entries

**Planned:** Parallel processing with progress bar (v2.1)

### 4. Roblox API Rate Limiting

**Challenge:**

- Roblox API rate limits are IP-based
- Multiple users from same customer share IP on Vercel
- Can hit rate limits faster during heavy usage

**Current Protection:**

- 30-second cooldowns on Smart/Display Name modes
- Circuit breaker stops requests during outages
- Request queue limits concurrency

**Future Enhancement:** Per-customer rate limiting (v2.2)

### 5. Forensic Report Storage

**Limitation:**

- Forensic reports not stored in database
- User must download immediately
- No history of generated reports

**Current State:** Reports generated on-demand, not persisted

**Planned:** Store forensic reports in database with history (v2.3)

### 6. Exact Match Only for Batch Upload

**Limitation:**

- Batch CSV processing only uses Exact Match mode
- No Smart Match or Display Name for batch

**Reason:** Cooldown timers prevent rapid searches

**Workaround:** Process batches individually in UI for Smart Match

**Future:** Intelligent batch processing with mode selection (v2.4)

### 7. No Email Notifications

**Missing Features:**

- X Welcome emails for new customers
- X Password reset emails
- X Search alert notifications
- X Admin action notifications

**Workaround:** Manual communication of credentials

**Planned:** Email integration (SendGrid or AWS SES) in v2.5

## 11.2 Technical Debt

### 1. Caching Layer

**Status:** Redis caching implemented but optional

**Issue:** Without caching, repeated searches hit Roblox API unnecessarily

**Recommendation:** Enable Redis in production (Upstash free tier works)

**Priority:** Medium

### 2. API Key Authentication

**Status:** Not implemented

**Issue:** Only web UI authentication, no programmatic access

**Use Case:** Customers want to integrate via API

**Planned:** API key generation and authentication (v3.0)

### 3. Audit Logs

**Status:** Table exists but not actively used

**Issue:** No tracking of admin actions (customer creation, deactivation, etc.)

**Planned:** Full audit logging implementation (v2.6)

### 4. Test Coverage

**Status:** No automated tests

**Issue:** No unit tests, integration tests, or E2E tests

**Risk:** Regressions during future development

**Planned:** Jest + React Testing Library setup (v2.7)

### 5. Error Monitoring

**Status:** Console logging only

**Issue:** No centralized error tracking in production

**Recommendation:** Integrate Sentry or similar service

**Priority:** High

## 11.3 Known Bugs

### 1. Session Expiration Edge Case

**Bug:** If user changes “Remember Me” preference without logging out, old session persists

**Impact:** Low (user just needs to logout and login again)

**Workaround:** Clear browser cookies

**Fix Planned:** v0.9.2

## 2. Logo Upload Race Condition

**Bug:** Rapidly uploading multiple logos can cause previous upload to fail

**Impact:** Low (just retry upload)

**Cause:** Supabase Storage delay in file replacement

**Fix Planned:** Add debouncing to upload button (v0.9.3)

## 3. Deep Context Friend List Pagination

**Bug:** Friend list only shows first 27 friends (API limit)

**Impact:** Medium (users with 200+ friends can't see all)

**Workaround:** None currently

**Fix Planned:** Implement pagination for friend list (v2.8)

## 4. Thumbnail Loading Flicker

**Bug:** Avatar thumbnails sometimes show placeholder briefly before loading






**Impact:** Low (cosmetic only)

**Cause:** Image loading without skeleton state

**Fix Planned:** Add proper loading skeletons (v2.9)

## 11.4 Security Considerations

### Potential Vulnerabilities (All Mitigated)

1. **SQL Injection:**  Prevented (parameterized queries)
2. **XSS Attacks:**  Prevented (React auto-escaping, httpOnly cookies)
3. **CSRF Attacks:**  Prevented (SameSite cookies, CSRF tokens)
4. **Password Security:**  Strong (bcrypt hashing, min 8 chars)
5. **Session Hijacking:**  Mitigated (httpOnly cookies, HTTPS in production)




### Recommended Improvements

1. **Two-Factor Authentication** (v3.1)
2. **IP-based Login Alerts** (v3.2)
3. **Password Complexity Requirements** (currently only min 8 chars)
4. **Account Lockout** after failed login attempts (v3.3)
5. **Security Headers** (CSP, HSTS, etc.) - partially implemented

## 12. Deployment Guide

### 12.1 Prerequisites

Before deploying, ensure you have:

-  GitHub repository with code
-  Vercel account (free tier works)
-  Supabase account with project created

- ☒ Database schema applied
- ☒ Super admin user created
- ☒ Environment variables ready

## 12.2 Database Setup (Supabase)

### Step 1: Create Supabase Project

1. Go to [supabase.com](https://supabase.com) (<https://supabase.com>)
2. Click “New Project”
3. Fill in:
  - **Name:** roblox-verifier (or your choice)
  - **Database Password:** Strong password (save it!)
  - **Region:** Choose closest to your users
4. Wait 2-3 minutes for provisioning

### Step 2: Get Connection String

1. In Supabase dashboard: Settings → Database
2. Scroll to “Connection string”
3. Select “URI” tab
4. Copy connection string (replace `[YOUR-PASSWORD]` with your database password)

#### Format:

```
postgresql://postgres:[YOUR-PASSWORD]@db.abcdefg.supabase.co:5432/postgres
```

### Step 3: Apply Database Schema

#### Option A: Supabase SQL Editor (Recommended)

1. Go to SQL Editor in Supabase
2. Click “New Query”
3. Copy contents of `database/schema.sql`
4. Paste and click “Run”
5. Verify tables created: Table Editor → See customers, users, search\_history

#### Option B: Command Line

```
psql "postgresql://postgres:[YOUR-PASSWORD]@db.abcdefg.supabase.co:5432/postgres" -f database/schema.sql
```

### Step 4: Create Super Admin User

In Supabase SQL Editor, run:

```
-- Generate password hash first (run locally)
-- node -e "const bcrypt = require('bcrypt'); bcrypt.hash('YourPassword123', 10, (err, hash) => console.log(hash));"

-- Insert super admin (replace hash with generated hash)
INSERT INTO users (
  username, password_hash, role, customer_id,
  email, full_name, is_active
) VALUES (
  'admin',
  '$2b$10$YourBcryptHashHere',
  'SUPER_ADMIN',
  NULL,
  'admin@yourdomain.com',
  'System Administrator',
  true
);
```

**Save your admin credentials!**

### Step 5: Create Storage Bucket

1. In Supabase: Storage → New Bucket
2. Name: `customer-logos`
3. Public bucket: **Yes** ☒
4. File size limit: 5MB
5. Click “Create Bucket”

**OR** the app will auto-create it on first logo upload.

### Step 6: Get Supabase Anon Key

1. Settings → API
2. Copy “anon public” key (NOT service\_role)
3. Save for environment variables

## 12.3 Vercel Deployment

### Step 1: Connect GitHub Repository

1. Go to [vercel.com](https://vercel.com) (<https://vercel.com>)
2. Click “New Project”
3. Import your GitHub repository
4. Select the roblox-tool repository

### Step 2: Configure Project

**Framework Preset:** Next.js (auto-detected)

**Root Directory:** `./` (default)

**Build Command:** `npm run build` (default)

**Output Directory:** `.next` (default)

**Install Command:** `npm install` (default)

Click “Continue” without deploying yet.

### Step 3: Add Environment Variables

In the Vercel project setup, add environment variables:



```
# Required
DATABASE_URL=postgresql://postgres:[YOUR-PASSWORD]@db.abcdefg.supabase.co:5432/postgres

NEXTAUTH_SECRET=[Generated with: openssl rand -base64 32]

NEXTAUTH_URL=https://your-app-name.vercel.app

NEXT_PUBLIC_SUPABASE_ANON_KEY=[Your Supabase anon key]

# Optional
REDIS_URL=[Your Redis URL if using caching]
```

**Important:** Select “All” environments (Production, Preview, Development)

## Step 4: Deploy

1. Click “Deploy”
2. Wait 2-3 minutes for build
3. Deployment URL generated: `https://your-app-name.vercel.app`

## Step 5: Test Deployment

1. Visit deployment URL
2. Click “Sign In”
3. Enter admin credentials
4. Verify you can access:
  - Main search page
  - Admin dashboard ( `/admin` )
5. Perform a test search
6. Check search appears in admin dashboard → Search History

## 12.4 Custom Domain (Optional)

### Step 1: Add Domain in Vercel

1. Vercel Dashboard → Your Project → Settings → Domains
2. Enter your domain (e.g., `roblox-tool.yourdomain.com` )
3. Click “Add”

### Step 2: Configure DNS

Add these records to your DNS provider:

**For Subdomain** (e.g., `roblox-tool.yourdomain.com`):

```
Type: CNAME
Name: roblox-tool
Value: cname.vercel-dns.com
```

**For Root Domain** (e.g., `yourdomain.com`):

```
Type: A
Name: @
Value: 76.76.21.21
```

### Step 3: Wait for DNS Propagation

- Propagation time: 5 minutes - 48 hours
- Vercel will auto-generate SSL certificate
- Check status in Domains tab

### Step 4: Update NEXTAUTH\_URL

1. Vercel Dashboard → Settings → Environment Variables
2. Edit `NEXTAUTH_URL`
3. Change to: `https://your-custom-domain.com`
4. Redeploy for changes to take effect

## 12.5 Monitoring & Maintenance

### Health Checks

**Endpoint:** `https://your-app.vercel.app/api/health`

**Response:**

```
{
  "status": "healthy",
  "timestamp": "2025-10-21T14:30:00.000Z",
  "database": "connected",
  "version": "1.0.0"
}
```

Set up uptime monitoring:

- **UptimeRobot** (free)
- **Pingdom**
- **StatusCake**

### Logging

#### Vercel Logs:

1. Vercel Dashboard → Your Project → Logs
2. Filter by:
  - Function (API route)
  - Time range
  - Status code

#### Database Monitoring:

1. Supabase Dashboard → Database → Performance
2. Monitor:
  - Query performance
  - Connection count
  - Storage usage

### Error Tracking

**Recommended:** Integrate Sentry

```
npm install @sentry/nextjs

# Follow setup wizard
npx @sentry/wizard@latest -i nextjs
```

Then errors will be tracked automatically.

## Performance Monitoring

### Vercel Analytics (Built-in):

- Page load times
- Core Web Vitals
- User geography

**Enable:** Vercel Dashboard → Your Project → Analytics

## 12.6 Backup Strategy

### Database Backups

#### Supabase Automatic Backups:

- Free tier: Daily backups (7 days retention)
- Pro tier: Hourly backups (30 days retention)

#### Manual Backup:

```
# Export database
pg_dump "postgresql://postgres:[PASSWORD]@db.abcdefg.supabase.co:5432/postgres" > backup.sql

# Restore if needed
psql "postgresql://postgres:[PASSWORD]@db.abcdefg.supabase.co:5432/postgres" < backup.sql
```

#### Recommended Frequency:

- Before major deployments
- Weekly for production data

### Logo Backups

Logos stored in Supabase Storage are included in Supabase backups.

#### Manual Download:

1. Supabase Dashboard → Storage → customer-logos
2. Download all files

## 12.7 Scaling Considerations

### Database

#### Supabase Free Tier Limits:

- 500MB database
- 2GB bandwidth
- Unlimited API requests

#### Upgrade Triggers:

- Database > 400MB (80% of limit)
- Monthly bandwidth > 1.5GB

#### Pro Tier: \$25/month

- 8GB database
- 250GB bandwidth
- Automatic backups

## Vercel

### Free Tier Limits:

- 100GB bandwidth
- Unlimited serverless function invocations
- 100 hours serverless function execution

### Pro Tier: \$20/month

- 1TB bandwidth
- Unlimited execution time
- Priority support

### Scaling Tips

1. **Enable Redis caching** to reduce Roblox API calls
  2. **Implement CDN** for static assets (Vercel does this automatically)
  3. **Database indexing** already implemented for common queries
  4. **Connection pooling** enabled (pg Pool with max 20 connections)
-

## 13. File Structure

---

```

roblox-tool/
├── .env.example           # Environment variable template
├── .env.local             # Local environment variables (gitignored)
├── .gitignore            # Git ignore rules
├── README.md             # Basic project readme
├── COMPREHENSIVE_APP_DOCUMENTATION.md # This file
├── package.json          # Dependencies and scripts
├── package-lock.json     # Lock file
├── tsconfig.json         # TypeScript configuration
├── next.config.js        # Next.js configuration
├── tailwind.config.ts    # Tailwind CSS config
├── postcss.config.js     # PostCSS config
├── vercel.json           # Vercel deployment config
├── next-auth.d.ts        # NextAuth TypeScript types
├──
├── database/            # Database schemas and migrations
│   ├── schema.sql       # Complete database schema
│   └── migrations/
│       ├── 001_allow_null_customer_id_in_search_history.sql
│       ├── 002_add_search_mode_column.sql
│       └── 003_add_customer_logo.sql
├── scripts/             # Utility scripts
│   ├── init-db.ts       # Database initialization script
│   └── run-migration.js  # Migration runner
├── public/              # Static assets
│   ├── favicon.ico
│   └── images/
├──
├── src/
│   ├── app/            # Next.js App Router
│   │   ├── layout.tsx  # Root layout
│   │   ├── page.tsx    # Main search page
│   │   ├── error.tsx   # Error boundary
│   │   └── Providers.tsx # Context providers
│   ├──
│   ├── auth/           # Authentication pages
│   │   └── signin/
│   │       └── page.tsx # Sign in page
│   ├──
│   ├── admin/          # Admin dashboard
│   │   ├── page.tsx    # Admin dashboard layout
│   │   └── components/
│   │       ├── DashboardOverview.tsx
│   │       ├── CustomerManagement.tsx
│   │       ├── UserManagement.tsx
│   │       └── SearchHistory.tsx
│   ├──
│   ├── status/         # Status page
│   │   └── page.tsx
│   ├──
│   ├── components/     # Shared components
│   │   ├── DeepContext.tsx # Profile viewer modal
│   │   ├── SmartSuggest.tsx # Smart Match results
│   │   ├── DisplayNameResults.tsx # Display Name results
│   │   ├── ForensicMode.tsx # Forensic toggle
│   │   ├── SearchModeSelector.tsx # Mode selector
│   │   ├── NoResultsModal.tsx # No results modal
│   │   ├── AuditLogViewer.tsx # Audit logs (future)
│   │   └── ui/
│   │       └── alert.tsx # Alert component

```



	customers/	
	route.ts	# List/create customers
	[customerId]/	
	logo/	
	route.ts	# Upload/delete logo
	searches/	
	route.ts	# Customer search history
	users/	
	route.ts	# Customer users
	users/	
	route.ts	# List/create users
	password/	
	route.ts	# Reset password
	search-history/	
	route.ts	# All search history
	stats/	
	route.ts	# Dashboard stats
	metrics/	
	route.ts	# Performance metrics
	middleware.ts	# Next.js middleware (auth)
	documentation/	# Additional documentation
	MULTI_TENANT_AUTH_SETUP.md	
	IMPLEMENTATION_SUMMARY.md	
	DEPLOYMENT_FIX_SUMMARY.md	
	REMEMBER_ME_FEATURE_SUMMARY.md	
	LOGO_UPLOAD_FIX.md	
	SETUP.md	
	NEXTJS_15_FIX_SUMMARY.md	
	TYPESCRIPT_FIX_SUMMARY.md	
	BUCKET_CREATION_VISUAL_GUIDE.md	
	FIX_SUMMARY.md	
	ADMIN_DASHBOARD.md	

## Key File Purposes

File	Purpose
src/app/page.tsx	Main search interface
src/app/admin/page.tsx	Super Admin dashboard
src/app/lib/auth.ts	NextAuth configuration
src/app/lib/db/index.ts	Database connection and queries
src/app/lib/ranking.ts	AI ranking algorithm for Smart Match
src/app/lib/storage.ts	Supabase Storage for logos
src/middleware.ts	Authentication and authorization
database/schema.sql	Complete database schema
vercel.json	Vercel deployment configuration



---

## 14. Development Workflow

---

### 14.1 Local Development Setup

#### Prerequisites

- Node.js 18+ installed
- PostgreSQL or Supabase account
- Git installed

#### Step 1: Clone Repository

```
git clone https://github.com/your-org/roblox-tool.git
cd roblox-tool
```

#### Step 2: Install Dependencies

```
npm install
```

#### Step 3: Set Up Environment Variables

```
cp .env.example .env.local
# Edit .env.local with your actual values
```

#### Step 4: Set Up Database

##### Option A: Use Supabase (Recommended)

1. Create Supabase project
2. Get connection string
3. Run schema in SQL Editor

##### Option B: Local PostgreSQL

```
# Create database
createdb roblox_verifier

# Apply schema
psql roblox_verifier < database/schema.sql

# Create super admin
psql roblox_verifier
INSERT INTO users (username, password_hash, role, customer_id) VALUES (...);
```

#### Step 5: Run Development Server

```
npm run dev
```

Open <http://localhost:3000>

## 14.2 Making Changes

### Code Style

- **TypeScript:** Strict mode enabled
- **ESLint:** Run `npm run lint` before committing
- **Formatting:** Prettier (optional, not enforced)

### Component Structure

```
// Import statements
import { useState, useEffect } from 'react';
import { useSession } from 'next-auth/react';

// Type definitions
interface MyComponentProps {
  prop1: string;
  prop2?: number;
}

// Component
export default function MyComponent({ prop1, prop2 }: MyComponentProps) {
  // State
  const [state, setState] = useState<string>('');

  // Hooks
  const { data: session } = useSession();

  // Effects
  useEffect(() => {
    // Effect logic
  }, [dependency]);

  // Handlers
  const handleAction = async () => {
    // Handler logic
  };

  // Render
  return (
    <div>
      {/* JSX */}
    </div>
  );
}
```

## API Route Structure

```
// Import statements
import { NextRequest, NextResponse } from 'next/server';
import { someUtility } from '@app/lib/utils';

// Export configuration (if needed)
export const dynamic = 'force-dynamic';

// Handler
export async function GET(request: NextRequest) {
  try {
    // Get user info from headers (set by middleware)
    const userId = request.headers.get('X-User-Id');

    // Validate input
    if (!userId) {
      return NextResponse.json(
        { error: 'Unauthorized' },
        { status: 401 }
      );
    }

    // Business logic
    const data = await fetchSomeData(userId);

    // Return response
    return NextResponse.json({ data });
  } catch (error) {
    console.error('API error:', error);
    return NextResponse.json(
      { error: 'Internal server error' },
      { status: 500 }
    );
  }
}
```

## 14.3 Testing

### Manual Testing Checklist

#### Authentication:

- [ ] Can login with valid credentials
- [ ] Cannot login with invalid credentials
- [ ] Remember me checkbox works
- [ ] Logout works
- [ ] Session persists across page refreshes

#### Search:

- [ ] Exact Match mode works
- [ ] Smart Match mode works (with cooldown)
- [ ] Display Name mode works (with cooldown)
- [ ] No results handled gracefully
- [ ] Errors displayed to user

#### Admin Dashboard:

- [ ] Only SUPER\_ADMIN can access
- [ ] Can create new customer

- ☐ Can view customer details
- ☐ Can upload customer logo
- ☐ Can view search history

#### Deep Context:

- ☐ Profile viewer opens
- ☐ All tabs load data
- ☐ Can navigate to friends
- ☐ Can export to PDF

#### Forensic Mode:

- ☐ Toggle enables forensic features
- ☐ Report generates correctly
- ☐ Hash is valid

### Automated Testing (Future)

#### Planned Test Structure:

```
tests/
├── unit/
│   ├── lib/
│   │   ├── ranking.test.ts
│   │   └── forensic.test.ts
│   └── components/
│       ├── DeepContext.test.tsx
│       └── SmartSuggest.test.tsx
├── integration/
│   ├── auth.test.ts
│   ├── search.test.ts
│   └── admin.test.ts
└── e2e/
    ├── user-flow.test.ts
    └── admin-flow.test.ts
```

## 14.4 Git Workflow

### Branch Strategy

```
main          (production, protected)
└─ develop    (development, protected)
    ├── feature/search-improvements
    ├── feature/user-management
    ├── bugfix/logo-upload
    └─ hotfix/session-expiration
```

### Commit Messages

Follow Conventional Commits:

```
feat: Add Smart Match search mode
fix: Resolve logo upload error on Vercel
docs: Update comprehensive documentation
refactor: Improve ranking algorithm performance
test: Add unit tests for ranking algorithm
chore: Update dependencies
```

## Pull Request Process

1. Create feature branch: `git checkout -b feature/my-feature`
2. Make changes and commit
3. Push to GitHub: `git push origin feature/my-feature`
4. Open Pull Request on GitHub
5. Wait for CI checks to pass
6. Request review from team member
7. Address review feedback
8. Merge when approved

## 14.5 Deployment Process

### Development Deployment

- **Trigger:** Push to `develop` branch
- **URL:** `https://roblox-tool-git-develop-yourorg.vercel.app`
- **Purpose:** Testing before production

### Production Deployment

- **Trigger:** Push to `main` branch
- **URL:** `https://your-app.vercel.app` (or custom domain)
- **Process:**
  1. Merge PR to `main`
  2. Vercel auto-deploys
  3. Run smoke tests
  4. Monitor logs for errors

### Rollback Process

If deployment fails:

1. Go to Vercel Dashboard
2. Deployments → Previous deployment
3. Click “Promote to Production”

Or via Git:

```
git revert HEAD
git push origin main
```

## 15. Important Context & Gotchas

### 15.1 Critical Issues We’ve Solved

#### 1. Vercel Filesystem Is Read-Only

**Problem:** Original code tried to save logos to local filesystem using `fs.writeFile()`. Vercel’s serverless functions have read-only filesystems, causing errors.

**Solution:** Migrated to Supabase Storage (cloud storage)

**Lesson:** Never use filesystem storage on serverless platforms. Use cloud storage (Supabase, S3, Vercel Blob, etc.)

## 2. Database Connection Pooling

**Problem:** Initially used direct database connections, causing “too many connections” errors under load.

**Solution:** Implemented connection pooling with `pg.Pool` (max 20 connections)

**Lesson:** Always use connection pooling for serverless functions.

## 3. SUPER\_ADMIN Search Logging

**Problem:** SUPER\_ADMIN has `customer_id = NULL`, but `search_history` required NOT NULL `customer_id`. Searches weren’t being logged.

**Solution:** Migration to allow NULL `customer_id` in `search_history`

**Lesson:** Design database schema to accommodate all user roles from the start.

## 4. Remember Me Cookie Configuration

**Problem:** Initial Remember Me implementation didn’t persist across browser sessions due to cookie configuration.

**Solution:** Fixed cookie settings to allow longer `maxAge` and proper `httpOnly/secure` flags

**Lesson:** Cookie configuration is critical for session persistence.

## 5. TypeScript Errors Blocking Deployment

**Problem:** Unused variables and `any` types caused TypeScript compilation errors, preventing Vercel deployment.

**Solution:** Cleaned up unused variables, added proper type annotations, used `eslint-disable` where necessary with explanations

**Lesson:** Keep TypeScript strict mode enabled and fix errors promptly.

# 15.2 Performance Optimizations

## 1. Non-Blocking Search Logging

**Implementation:** Search logging uses `async/await` without blocking the response.

```
// DON'T DO THIS (blocks response)
await logSearch(...);
return NextResponse.json(results);

// DO THIS (non-blocking)
logSearch(...).catch(err => console.error('Log failed:', err));
return NextResponse.json(results);
```

**Result:** Search API returns results ~50ms faster

## 2. Image Optimization

**Next.js Image Component:** Used for avatars and logos

```
<Image
  src={avatarUrl}
  alt="Avatar"
  width={64}
  height={64}
  unoptimized={false} // Let Next.js optimize
/>
```

**Benefits:** Auto-resizing, lazy loading, WebP conversion

### 3. Database Indexes

#### Critical Indexes:

- `search_history.user_id` - Fast user history lookup
- `search_history.customer_id` - Fast customer history lookup
- `search_history.searched_at DESC` - Fast recent searches
- `users.username` - Fast login queries

**Result:** Admin dashboard loads in <500ms even with 10,000+ searches

## 15.3 Security Best Practices We Follow

1. **Never store passwords in plain text:** bcrypt hashing with salt rounds
2. **Never expose secret keys:** Environment variables only
3. **Parameterized queries:** Prevent SQL injection
4. **HttpOnly cookies:** Prevent XSS token theft
5. **HTTPS in production:** Enabled automatically by Vercel
6. **CSRF protection:** SameSite cookies
7. **Role-based authorization:** Middleware enforces permissions

## 15.4 Common Pitfalls to Avoid

### 1. Forgetting Middleware Headers

**Problem:** API routes won't get user info if you forget middleware sets headers.

**Solution:** Always use headers in API routes:

```
const userId = request.headers.get('X-User-Id');
const customerId = request.headers.get('X-Customer-Id');
```

### 2. Not Handling NULL customer\_id

**Problem:** SUPER\_ADMIN users have `customer_id = NULL`, which breaks queries that assume NOT NULL.

**Solution:** Always check for NULL:

```
const customerId = customerIdStr && customerIdStr !== 'null'
  ? parseInt(customerIdStr)
  : null;
```

### 3. Hardcoding URLs

**Problem:** Hardcoded URLs break when switching environments.

**Solution:** Use environment variables:

```
const baseUrl = process.env.NEXTAUTH_URL || 'http://localhost:3000';
```

#### 4. Not Escaping User Input in SQL

**Problem:** Even though we use parameterized queries, manual string concatenation is dangerous.

**Solution:** Always use parameterized queries:

```
// DON'T DO THIS
await query(`SELECT * FROM users WHERE username = '${username}'`);

// DO THIS
await query('SELECT * FROM users WHERE username = $1', [username]);
```

#### 5. Forgetting to Close Modals

**Problem:** Modal stays open after successful action, confusing users.

**Solution:** Always close modals after success:

```
async function handleSubmit() {
  await submitForm();
  setIsOpen(false); // Close modal
  refreshData();    // Refresh parent component
}
```

### 15.5 Future Development Considerations

#### 1. Multi-Database Support

Currently tied to PostgreSQL. If supporting other databases:

- Abstract database queries into ORM (Prisma, Drizzle)
- Make schema migrations database-agnostic
- Test with multiple database types

#### 2. Internationalization (i18n)

If adding multi-language support:

- Use `next-intl` or `react-i18next`
- Externalize all user-facing strings
- Handle date/time formatting per locale

#### 3. Mobile App

If building mobile app:

- API already supports it (just need authentication)
- Consider JWT tokens in mobile storage
- Implement refresh tokens for long-term sessions

#### 4. Real-Time Features

If adding real-time updates (e.g., live search results):

- Consider WebSockets or Server-Sent Events
- Use Supabase Realtime (built-in)
- Implement optimistic UI updates



## 15.6 Debugging Tips

### Check Logs

#### Vercel Logs:

```
vercel logs [deployment-url]
```

#### Local Logs:

- Check terminal where `npm run dev` is running
- Check browser console (F12)

### Common Error Messages

#### “Database connection failed”

- Check `DATABASE_URL` is correct
- Verify database is running (Supabase)
- Check network/firewall

#### “Unauthorized”

- Check if logged in ( `useSession()` )
- Check middleware is setting headers
- Check role matches required permission

#### “Bucket not found”

- Create `customer-logos` bucket in Supabase Storage
- Check RLS policies allow public read

#### “Rate limited”

- Wait for cooldown to expire
- Check Roblox API status
- Verify circuit breaker isn't open

### Useful Debugging Commands

```
# Check database connection
psql "$DATABASE_URL" -c "SELECT 1"

# Check environment variables in Vercel
vercel env ls

# Check build logs
vercel logs --follow

# Check git history
git log --oneline --graph







# Check which branch you're on
git branch

# Check uncommitted changes
git status
```

---

# Conclusion

This comprehensive documentation covers **every aspect** of the Roblox Verifier Tool application. It should enable any new developer or AI agent to:

-  Understand the application architecture
-  Set up local development environment
-  Deploy to production
-  Debug issues
-  Add new features
-  Maintain and improve the codebase

## Quick Reference Links

- **GitHub Repository:** [Link to repo]
- **Production URL:** [Link to production]
- **Admin Dashboard:** [Production URL]/admin
- **Supabase Dashboard:** [Link to Supabase project]
- **Vercel Dashboard:** [Link to Vercel project]

## Contact & Support

- For questions or issues:
- Check this documentation first
  - Review existing documentation in `/documentation`
  - Check GitHub Issues
  - Contact project maintainer

## Version History

Version	Date	Major Changes
0.1.0	Sept 2025	Initial development
0.5.0	Oct 15, 2025	Multi-tenant authentication
0.6.0	Oct 16, 2025	Deployment fixes
0.7.0	Oct 18, 2025	Smart Match & Display Name modes
0.8.0	Oct 19, 2025	Forensic mode
0.8.1	Oct 20, 2025	Remember Me feature
0.9.0	Oct 21, 2025	Logo upload fix (Supabase Storage)
1.0.0	Oct 21, 2025	Production ready, comprehensive docs

**Document Maintained By:** Development Team

**Last Updated:** October 21, 2025

**Document Version:** 1.0.0

---

**END OF COMPREHENSIVE APPLICATION DOCUMENTATION**