VerifyLens API Implementation

Overview

This document describes the REST API implementation for the VerifyLens Roblox user verification service. The API allows clients to integrate Roblox user verification directly into their CRM or system.

Implementation Status

✅ Phase 1: Prisma Schema Updates (COMPLETED)

The database schema includes the following models:

User Model

- `id` : Unique identifier
- `email` : User email address (unique)
- `name` : User's full name
- `companyName` : Optional company/organization name
- `credits` : Current credit balance
- `stripeCustomerId` : Stripe customer ID for billing
- `isActive` : Account activation status
- `isAdmin` : Admin privileges flag
- Relations: ApiClient, VerificationCache

ApiClient Model

- `id` : Unique identifier
- `userId` : Reference to User
- `apiKey` : Plain API key (stored temporarily, shown once)
- `apiKeyHash` : Bcrypt hash of API key for verification
- `isActive` : API client activation status
- `lastUsedAt` : Timestamp of last API usage
- Relations: ApiUsageLog, ApiTransaction

ApiUsageLog Model

- Tracks every API request made
- Records: endpoint, requestId, creditsUsed, success status, response time, IP, user agent, errors
- Enables detailed usage analytics and debugging

ApiTransaction Model

- Records all credit transactions (purchases, deductions, refunds)
- Tracks: type, amount, credits changed, balance before/after
- Links to Stripe payment IDs for purchases

VerificationCache Model

- Caches search results for 30 days
- Uses SHA-256 hash of search parameters as key
- Stores result data as JSON

- Includes expiration timestamp

✅ Phase 2: Authentication Setup (COMPLETED)

API Key Authentication

Location: `lib/api-auth.ts`

- **API Key Format**: `vl_live_<64-character-hex-string>`
- **Storage**: Keys are hashed using bcrypt (12 salt rounds)
- **Verification**: Constant-time comparison using bcrypt
- **Security**: Keys shown only once upon generation

Key Functions:
- `authenticateApiRequest()` : Validates X-API-Key header
- `checkCredits()` : Verifies sufficient balance
- `deductCredits()` : Atomic credit deduction with transaction logging

API Key Generation

Location: `lib/api-key.ts`

Functions:
- `generateApiKey()` : Creates cryptographically secure API keys
- `hashApiKey()` : Hashes keys for secure storage
- `verifyApiKey()` : Validates keys against stored hashes
- `maskApiKey()` : Masks keys for display (first 8 chars + asterisks)

Rate Limiting

Location: `lib/rate-limit.ts`

Implementation: Redis-based cooldown system
- **Smart Verification**: 30-second cooldown
- **Exact Verification**: 5-second cooldown

Functions:
- `checkRateLimit()` : Checks if request is allowed
- `setCooldown()` : Sets cooldown after successful request
- `getRemainingCooldown()` : Returns seconds until next request allowed

Caching System

Location: `lib/cache.ts`

Features:
- SHA-256 hash-based cache keys
- 30-day TTL for cached results
- Automatic cache hit tracking
- Zero credits charged for cached results

Functions:
- `generateSearchHash()` : Creates deterministic cache key
- `getCachedResult()` : Retrieves cached data if not expired
- `setCachedResult()` : Stores result with expiration
- `cleanExpiredCache()` : Removes expired entries

✅ Phase 3: Core API Routes (COMPLETED)

1. Generate API Key

Endpoint: `POST /api/client/generate-key`

Purpose: Creates or regenerates API keys (admin/internal use)

Request Body:

```
{
  "userId": "user_id_here",
  "regenerate": false
}
```

Response:

```
{
  "success": true,
  "message": "API key generated successfully",
  "apiKey": "vl_live_abc123...",
  "clientId": "client_id",
  "createdAt": "2025-10-31T..."
}
```

Features:
- Prevents duplicate key generation (unless regenerate=true)
- Logs key generation as transaction
- Returns key only once for security

2. Exact Verification

Endpoint: `POST /api/v1/verify/exact`

Purpose: Precise Roblox username verification

Authentication: Requires `X-API-Key` header

Request Body:

```
{
  "username": "JohnDoe",
  "userId": "1234567890",
  "strictMatch": true,
  "includeProfile": true
}
```

Response:

```json
{
  "success": true,
  "data": {
    "user": {
      "id": "1234567890",
      "username": "JohnDoe",
      "displayName": "JohnDoe",
      "joinDate": "2019-03-12T00:00:00Z",
      "hasVerifiedBadge": true,
      "isOnline": false,
      "lastSeen": "2025-10-31T...",
      "accountAge": 365
    },
    "verification": {
      "exact": true,
      "confidence": 95,
      "method": "direct_api",
      "timestamp": "2025-10-31T..."
    }
  },
  "fromCache": false,
  "creditsUsed": 100,
  "currentBalance": 900,
  "requestId": "uuid"
}
```

Features:
- **Cost**: 100 credits per verification
- **Cooldown**: 5 seconds
- **Cache**: 30-day TTL
- Returns comprehensive user data
- Includes security and verification metadata

Error Responses:
- `401`: Invalid or missing API key
- `402`: Insufficient credits
- `404`: User not found
- `429`: Rate limit exceeded

3. Smart Verification

Endpoint: `POST /api/v1/verify/smart`

Purpose: Flexible fuzzy matching verification

Request Body:

```json
{
  "username": "john",
  "filters": {
    "minAge": 0,
    "maxAge": 20,
    "hasAvatar": true,
    "minFriends": 10,
    "verifiedBadge": true
  },
  "includeHistory": false
}
```

Features:
- **Cost**: 200 credits per verification
- **Cooldown**: 30 seconds
- AI-powered fuzzy matching
- Advanced filtering options
- Returns multiple potential matches

4. Account Information

Endpoint: `GET /api/v1/account`

Purpose: Retrieve account details, credit balance, and usage stats

Authentication: Requires `X-API-Key` header

Response:

```json
{
  "success": true,
  "data": {
    "user": {
      "id": "user_id",
      "email": "user@example.com",
      "name": "John Doe",
      "companyName": "Acme Corp",
      "credits": 1000,
      "isActive": true,
      "memberSince": "2025-10-01T..."
    },
    "apiAccess": {
      "clientId": "client_id",
      "isActive": true,
      "lastUsed": "2025-10-31T...",
      "createdAt": "2025-10-01T..."
    },
    "usage": {
      "last30Days": {
        "totalRequests": 150,
        "successfulRequests": 145,
        "duplicateRequests": 30,
        "totalCreditsUsed": 15000,
        "successRate": 97
      }
    },
    "rateLimits": {
      "smartVerify": {
        "cooldownSeconds": 30,
        "remainingCooldown": 0
      },
      "exactVerify": {
        "cooldownSeconds": 5,
        "remainingCooldown": 0
      }
    },
    "recentTransactions": []
  }
}
```

5. Usage History

Endpoint: `GET /api/v1/usage`

Purpose: Retrieve detailed API usage logs and statistics

Authentication: Requires `X-API-Key` header

Query Parameters:
- `page` : Page number (default: 1)
- `limit` : Items per page (default: 20, max: 100)
- `endpoint` : Filter by endpoint (smart_verify, exact_verify, all)
- `success` : Filter by success status (true, false, all)
- `dateFrom` : Start date (ISO 8601)
- `dateTo` : End date (ISO 8601)

Example Request:

```
GET /api/v1/usage?page=1&limit=20&endpoint=exact_verify&success=true
```

Response:

```json
{
  "success": true,
  "data": {
    "logs": [
      {
        "id": "log_id",
        "endpoint": "exact_verify",
        "requestId": "uuid",
        "creditsUsed": 100,
        "wasSuccessful": true,
        "wasDuplicate": false,
        "responseTime": 250,
        "ipAddress": "192.168.1.1",
        "createdAt": "2025-10-31T..."
      }
    ],
    "summary": {
      "totalRequests": 150,
      "successfulRequests": 145,
      "duplicateRequests": 30,
      "totalCreditsUsed": 15000,
      "averageResponseTime": 280,
      "successRate": 97,
      "duplicateRate": 20,
      "endpointBreakdown": [
        {
          "endpoint": "exact_verify",
          "requests": 100,
          "creditsUsed": 10000
        },
        {
          "endpoint": "smart_verify",
          "requests": 50,
          "creditsUsed": 10000
        }
      ]
    },
    "pagination": {
      "currentPage": 1,
      "totalPages": 8,
      "totalCount": 150,
      "limit": 20,
      "hasNextPage": true,
      "hasPrevPage": false
    }
  }
}
```

Security Features

API Key Security

- **Format Validation**: Keys must start with `vl_live_`
- **Hashing**: Bcrypt with 12 salt rounds

- **Single Display**: Keys shown only once at generation
- **No Storage**: Plain keys not stored in database

Rate Limiting

- **Redis-based**: Fast, distributed rate limiting
- **Per-user**: Individual cooldowns per user ID
- **Graceful Degradation**: Falls back to allowing requests if Redis fails

Request Security

- **Request IDs**: Unique UUID for every request
- **IP Logging**: Tracks client IP addresses
- **User Agent Logging**: Records client information
- **Error Logging**: Comprehensive error tracking

Error Handling

All API endpoints return consistent error responses:

```
{
  "error": "Error type",
  "message": "Detailed error message",
  "details": {},
  "requestId": "uuid"
}
```

HTTP Status Codes

- `200` : Success
- `201` : Resource created
- `400` : Bad request (validation error)
- `401` : Unauthorized (invalid API key)
- `402` : Payment required (insufficient credits)
- `404` : Not found
- `409` : Conflict (duplicate resource)
- `429` : Too many requests (rate limit)
- `500` : Internal server error

Response Headers

All API responses include:
- `X-Request-ID` : Unique request identifier
- `X-Cache` : Cache status (HIT/MISS)
- `X-Credits-Used` : Credits deducted for request
- `X-Credits-Remaining` : Remaining credit balance
- `X-RateLimit-Limit` : Rate limit threshold
- `X-RateLimit-Remaining` : Remaining requests
- `X-RateLimit-Reset` : Timestamp when limit resets
- `Retry-After` : Seconds to wait (on 429 errors)

Database Schema

Indexes

- `users.email` : Unique index for fast email lookup
- `users.stripeCustomerId` : Unique index for Stripe integration
- `apiClient.userId` : Unique index (one client per user)
- `apiClient.apiKey` : Unique index for key lookup
- `apiClient.apiKeyHash` : Unique index for verification
- `apiUsageLog.requestId` : Unique index for deduplication
- `verificationCache.searchHash` : Unique index for cache lookup
- `verificationCache.expiresAt` : Index for efficient cleanup
- `verificationCache[userId, searchHash]` : Composite index for user cache lookups

Environment Variables

Required environment variables:

```
# Database
DATABASE_URL="postgresql://user:password@host:5432/database"

# Redis (for rate limiting and caching)
REDIS_URL="redis://localhost:6379"

# Stripe (for Phase 4 - credit purchases)
STRIPE_SECRET_KEY="sk_live_..."
NEXT_PUBLIC_STRIPE_PUBLISHABLE_KEY="pk_live_..."

# Application URLs
NEXT_PUBLIC_APP_URL="https://www.verifylens.com"
NEXT_PUBLIC_SITE_URL="https://site.verifylens.com"
```

Testing

Generate Test API Key

```
curl -X POST https://api.verifylens.com/api/client/generate-key \
  -H "Content-Type: application/json" \
  -d '{
    "userId": "user_id_here",
    "regenerate": false
  }'
```

Test Exact Verification

```
curl -X POST https://api.verifylens.com/api/v1/verify/exact \
  -H "Content-Type: application/json" \
  -H "X-API-Key: vl_live_abc123..." \
  -d '{
    "username": "JohnDoe",
    "strictMatch": true,
    "includeProfile": true
  }'
```

Check Account Balance

```
curl -X GET https://api.verifylens.com/api/v1/account \
  -H "X-API-Key: vl_live_abc123..."
```

Get Usage History

```
curl -X GET "https://api.verifylens.com/api/v1/usage?page=1&limit=10" \
  -H "X-API-Key: vl_live_abc123..."
```

Pending Phases

Phase 4: Stripe Integration (NOT YET IMPLEMENTED)

- Credit package purchases
- Webhook handlers for payment events
- Automatic credit provisioning
- Payment history

Phase 5: Dashboard UI (NOT YET IMPLEMENTED)

- Client dashboard for API key management
- Usage analytics and graphs
- Credit purchase interface
- Transaction history
- Real-time usage monitoring

Files Changed/Created

Phase 1: Schema

- ✅ `prisma/schema.prisma` - Complete database schema

Phase 2: Authentication & Utilities

- ✅ `lib/api-auth.ts` - Authentication and credit management
- ✅ `lib/api-key.ts` - API key generation and validation
- ✅ `lib/rate-limit.ts` - Rate limiting logic
- ✅ `lib/cache.ts` - Caching system
- ✅ `lib/redis.ts` - Redis client
- ✅ `lib/db.ts` - Prisma client

Phase 3: API Routes

- ✅ `app/api/client/generate-key/route.ts` - API key generation
- ✅ `app/api/v1/verify/exact/route.ts` - Exact verification
- ✅ `app/api/v1/verify/smart/route.ts` - Smart verification
- ✅ `app/api/v1/account/route.ts` - Account information
- ✅ `app/api/v1/usage/route.ts` - Usage history

Next Steps

1. **Redis Configuration**: Ensure Redis is running and REDIS_URL is set
2. **Testing**: Test all API endpoints with real Roblox API integration
3. **Phase 4**: Implement Stripe credit purchasing system
4. **Phase 5**: Build client dashboard UI
5. **Documentation**: Create comprehensive API documentation site
6. **Monitoring**: Set up logging and monitoring for production

Notes

- All timestamps are in ISO 8601 format (UTC)
- Credit costs are configurable per endpoint
- Cache TTL is 30 days by default
- Rate limits are per-user, not global
- All transactions are logged for audit purposes
- Database is PostgreSQL with Prisma ORM
- Redis is used for rate limiting (fallback to allowing requests if unavailable)